



Social Media

Write-up Report

By

Group 5

Olivia Wang, Christina Zhang, Yanjun Jiang

Git Repository:

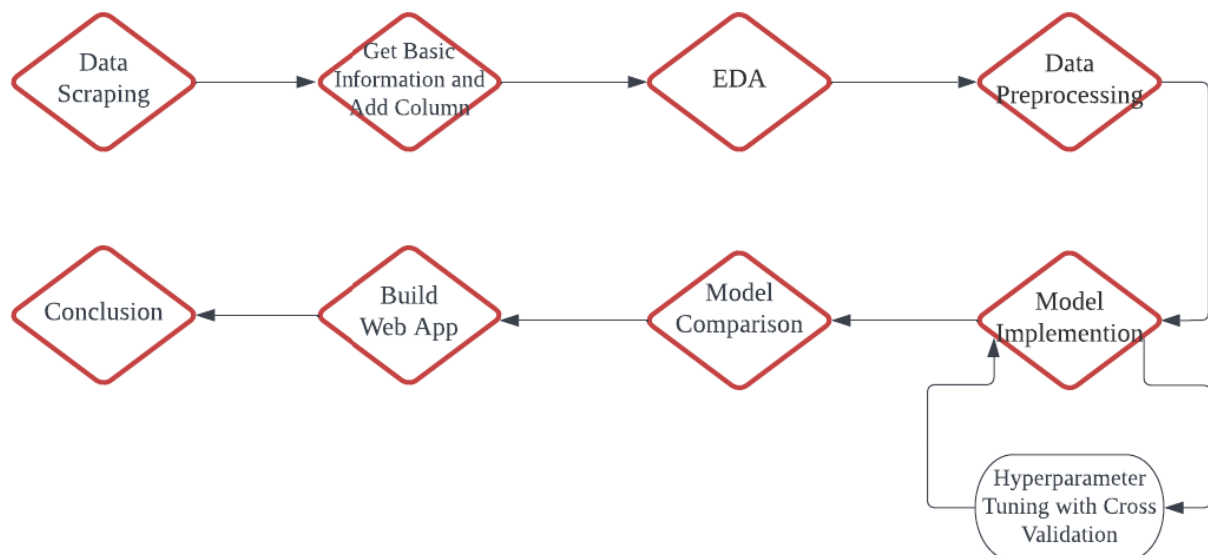
https://github.com/Oohlivia/Youtube_project.git

June 9, 2023

Overview

Social Media has become an inseparable part of today's society. Not only is social media a form of recreational activity, but more importantly, it is a money-making tool for some individuals and businesses. With more than 2 billion monthly active users, YouTube has been and continues to be one of the biggest social media gurus. Thus, we are interested in investigating how videos become "trendy" on the YouTube platform. This project will aim to develop an algorithm that predicts the popularity of user-uploaded videos on YouTube.

Our project extracts the information of videos from the top 25 channels on Youtube with API, including total view count across all their videos, comment count, and categories of videos etc. as their variables, and generate a large data frame with over thirty thousand observations. By analyzing the data and clean the sample data set, we want to find if there exist any relationship between any pairs of variables to help us build an algorithm with a machine learning model. We categorizes the view count into 10 subcategories and uses different machine learning models to train a high-accuracy algorithm in order to build a web app to help the users determine whether their videos would will be popular. Our final goal aims to create a web app that can help social media users, in particular, businesses and influencers, predict the potential view count as popularity of their videos. By entering duration, whether it made for kids, year, time of day, and title length, users can obtain the predicted view count of their videos.



The 4 main technical components of the project include web scraping YouTube API to extract data, data analysis with complex data visualization, training machine learning models that aim

to get the best performance in prediction accuracy, and building an interactive and dynamic website.

Technical Components

1. Web Scrapping

First of all, we use the package “googleapiclient” and “google” to get the YouTube API keys by following a tutorial that we found. For this project, focusing on the US channels would help our future prediction more directional and more accurate, so we extract the top channel IDs from US’s trending video page by using the function “trending_creators_by_country,” which returns a list of channel IDs from the specified country’s trending page. Each channel on YouTube has a unique channel ID. By taking the list of channel IDs in the function “channel_stats,” it returns a data frame containing overall channel stats, including total view count across all the videos in the channels, subscriber count, and video count etc. Since not all channels are representative for predicting a potential view count, and considering that the number of subscriber can be a helpful measure for the popularity of a channel in the long run, we shrink the range of the channels to the top 25 percentile based on their number of subscriber by the function “top_channels”. **Figure 1** shows the initial data frame of the top 25 percentile channels in US.

	title	country	viewCount	subscriberCount	videoCount	topic	description	channel_id
0	TUDN USA	US	1573519414	1660000	66015	[Sport, Association_football]	TUDN USA. \n\nLas mejores historias se viven e...	UCSo19KhHogXxu3sFsOpqrcQ
1	American Idol	US	306077663	3390000	1792	[Television_program, Music, Pop_music, Enterta...	Congrats to Season 20 winner, Noah Thompson! 🏆 ...	UCAMPco9PqjBbl_MLsDOO4Jw
2	Markiplier	US	19996689116	34800000	5454	[Video_game_culture, Action-adventure_game, Ac...	Hi, I'm Markiplier. I make videos. \n\nFrom qu...	UC7_YxT-KID8kRbqZo7MyscQ
3	PGA Championship	US	67427856	115000	1432	[Golf, Lifestyle_(sociology), Sport]	The official YouTube channel of the PGA Champi...	UCnUYyAtDiyeaobSzKqVE_BA
4	HYBE LABELS	KR	27844059138	70700000	1223	[Pop_music, Music_of_Asia, Music]	Welcome to the official YouTube channel of HYB...	UC3lZKseVpdzPSBaWxBxundA

Figure 1. Initial Data Frame of Top 25 Percentile Channels

Considering the basic popularity of certain famous companies and celebrities, and the fact that our project is more oriented to the normal people, we decided to filtering out their channels from the initial samples as shown in **Figure 2**.

```

drop_cols = ["Crunchyroll Collection", "Marvel Entertainment", "Universal Pictures", "BBC", "WWE",
             "Machine Gun Kelly", "JYP Entertainment", "NBA", "FORMULA 1", "Kylie Jenner",
             "Chris Brown", "David Guetta", "Warner Bros. Pictures", "Ed Sheeran", "Serie A",
             "Sun TV", "Genius", "Sony Pictures Entertainment", "SMTOWN", "Vogue", "Harry Styles",
             "Britain's Got Talent", "M2", "Fortnite", "Clash Royale", "Liverpool FC",
             "Harry Styles", "Bad Bunny", "Juice WRLD", "LanaDelReyVEVO", "BLACKPINK",
             "20th Century Studios", "Shakira", "MileyCyrusVEVO", "House of Highlights",
             "ESPN", "G)I-DLE (여자)아이들 (Official YouTube Channel)", "Netflix", "Clash of Clans",
             "DisneyMusicVEVO", "HYBE LABELS"]
ustop25 = us_top25[~us_top25.title.isin(drop_cols)]
ustop25 = ustop25.reset_index()
ustop25

```

Figure 2. Code Snippet of Filtering Channels

After getting the basic information of the top channels, we need to further extract the video information from these channels. Similar to how each channel has a unique ID, each video also has a unique ID. From the video IDs, we can get the basic information of the videos, such as video title and publish date. So we extract the video IDs by the function “get_videoID_list” and split the ID list into 4 sublists as shown in **Figure 3**.

```

ustop25_ids = ustop25["channel_id"]

video_id_list = []
for ID in ustop25_ids:
    video_id = get_videoID_list(youtube, ID)
    video_id_list.append(video_id)

video_id_list = list(itertools.chain(*video_id_list))
len(video_id_list), video_id_list[0:5]

(34646,
 ['H097Ai9hbKA', '3HGvOnEY_A0', '9812Me_JybU', '55p-3rVPD04', 'tU1iDmFd3o0'])

def split_list(lst, num_lists):
    size = len(lst) // num_lists
    return [lst[i*size:(i+1)*size] for i in range(num_lists)]

sublists = split_list(video_id_list, 4) # Split into four sublists
first_ids = sublists[0]
second_ids = sublists[1]
third_ids = sublists[2]
forth_ids = sublists[3]
len(first_ids), len(second_ids), len(third_ids), len(forth_ids)

(8661, 8661, 8661, 8661)

```

Figure 3. Code Snippet of Getting Video ID list and Split IDs

With function “get_video_details,” it takes in a list of video IDs and returns a data frame containing stats from every video, including video title, publish date, description, tags, view counts for that specific video, dislike count (if not made private), comment count, whether it

was made for kids, and much more. We input the 4 YouTube video ID lists to get 4 data frames of the video stats and finally concatenate the 4 data frames into one big data frame and write into a csv file as shown in **Figure 4**, which we would use for further exploration and prediction. The data frame shown in Figure 4 is automatically ordered based on the channels they belong to.

title	channelName	published	description	tags	category	viewCount	likeCount	dislikeCount	commentCount	duration	postingDate	made_for
Garten of BanBan 3	Markiplier	2023-05-22T16:27:31Z	Chapter 3 of Garten of BanBan is out! Where is...	['garten of banban 3', 'markiplier', 'scary ga...]	Gaming	419788	39271	private	2538	PT1H15M35S	2023-05-22T16:27:31Z	
The Return to Bloody Nights: Part 1	Markiplier	2023-05-21T17:15:35Z	The Return to Bloody Nights is an amazing Five...	['markiplier', 'the return to bloody nights', ...]	Gaming	2420031	156694	private	6633	PT35M48S	2023-05-21T17:15:35Z	
Amanda the Adventurer: ALL ENDINGS	Markiplier	2023-05-20T00:55:07Z	NEW CLOAK COLLECTION ► https://cloakbrand.com/...	['markiplier', 'amanda the adventurer', 'scary...']	Gaming	1718988	78836	private	2315	PT2H5M43S	2023-05-20T00:55:07Z	
BODY CAM HORROR GAME DEPART	Markiplier	2023-05-19T16:03:42Z	This horror game has a deeper level of immersi...	['markiplier', 'scary games', 'body cam horror...']	Gaming	2068246	119458	private	5293	PT22M8S	2023-05-19T16:03:42Z	

Figure 4. The Final Data Frame with Video Information

2. Complex Data Visualization

In this part, we first get basic information like number of channels and number of tags of each video, and add columns for further EDA, such as “tag Count” for counting the tag numbers if the video has tags. We also interested in how many views on average will have a comment for a video.

<pre> In [6]: # how many views on average will have a comment df["viewPerComment"] = df["viewCount"] / df["commentCount"] # number of tags # if it contains at least one character and one comma df["tagsCount"] = df["tags"].apply(lambda x: len(x.split(',')) if len(x) > 2 else 0) df.head() </pre>							
likeCount	dislikeCount	commentCount	duration	postingDate	made_for_kids	viewPerComment	tagsCount
39271	private	2538	PT1H15M35S	2023-05-22T16:27:31Z	False	165.401103	8
156694	private	6633	PT35M48S	2023-05-21T17:15:35Z	False	364.847128	6
78836	private	2315	PT2H5M43S	2023-05-20T00:55:07Z	False	742.543413	3
119458	private	5293	PT22M8S	2023-05-19T16:03:42Z	False	390.751181	9

Figure 5. Code Snippet and Updated Data Frame of Adding Columns

For the posting date of the videos, we process the “postingDate” column to universal format as in “datetime” package, and split the column to extract numeric “month” and “time” column. In addition, we add “Year” and “Weekday” columns to prepare for EDA as shown in **Figure 6**. Moreover, in **Figure 7**, we separate the publish time into 3 intervals: morning, afternoon, and evening. In “duration”, there is a value called “POD” which stands for streaming. In order to better assess the data, we removed any videos that are streaming and converted “duration” into minuets representation, as shown in **Figure 8**. The updated data frame is shown in **Figure 9**.

```

df['postingDate'] = pd.to_datetime(df['postingDate'])
# Add Year Column
df['year'] = df['postingDate'].apply(lambda x : x.strftime('%Y'))
# Add Weekday Column
df['weekday'] = df['postingDate'].apply(lambda x : x.weekday())
# Add Time Column
df['time'] = df['postingDate'].apply(lambda x : x.strftime("%H:%M"))
df['time'] = pd.to_datetime(df['time'])
# =====
# Add Month Column
df['month'] = df['postingDate'].apply(lambda x : x.strftime('%B'))

temp = pd.DataFrame({'postingDate': ['2022-01-01', '2022-02-01', '2022-03-01']})
# create a 'month' column with month names
temp['month'] = pd.to_datetime(temp['postingDate']).dt.strftime('%B')
# create a dictionary to map month names to numbers
month_dict = {month: datetime.strptime(str(month), "%B").month for month in pd.date_range(start='2022-01-01',
                                                                                          end='2022-12-31',
                                                                                          freq='MS').strftime("%B")}

df['month'] = df['month'].map(month_dict)

df.head()

```

Figure 6. Code Snippet for Adding “Year”, “Weekday”, “Time”, “Month” Columns

```

df['time'] = df['postingDate'].apply(lambda x : x.strftime('%H:%M'))
def categorize_time(x):
    hour = int(x.split(':')[0])
    if hour >= 6 and hour < 12:
        return 'morning'
    elif hour >= 12 and hour < 18:
        return 'afternoon'
    else:
        return 'evening'

df['time_of_day'] = df['time'].apply(lambda x: categorize_time(x))

```

Figure 7. Code Snippet of Separating into 3 Time Intervals

```

df = df[~(df['duration'] == 'POD')]

def convert_duration(duration_str):
    #Extract hours, minutes, and seconds using regex
    pattern = r'PT(?:\d+)H(?:\d+)M(?:\d+)S?'
    match = re.match(pattern, duration_str)
    if match:
        hours = int(match.group(1) or 0)
        minutes = int(match.group(2) or 0)
        seconds = int(match.group(3) or 0)
    else:
        raise ValueError("Invalid format")

    #Calculate duration in minutes
    total_min = hours*60 + minutes + seconds/60

    return total_min

df['duration'] = df['duration'].apply(convert_duration)

df.head()

```

Figure 8. Code Snippet of Converting “duration” into Minutes Representation

postingDate	made_for_kids	viewPerComment	tagsCount	year	weekday	time	month	time_of_day
2023-05-22 27:31+00:00	False	165.401103	8	2023	0	16:27	5	afternoon
2023-05-21 15:35+00:00	False	364.847128	6	2023	6	17:15	5	afternoon
2023-05-20 05:07+00:00	False	742.543413	3	2023	5	00:55	5	evening
2023-05-19 03:42+00:00	False	390.751181	9	2023	4	16:03	5	afternoon
2023-05-18 08:20+00:00	False	440.775436	9	2023	3	18:08	5	evening

Figure 9. The Update Data Frame

In EDA, we first created a heat map to check if there is any colinearity exist in any pair of variables. As shown in **Figure 9**, “likeCount”, “viewCount”, and “commentCount” are highly correlated to each other. Meanwhile, it is reasonable that “viewPerComment” is correlated to “viewCount” and “likeCount”. The heat map suggests that the correlated predictor variables are not able to independently predict the dependent variable. In other words, they explain some of the same variances in the dependent variable, reducing their statistical significance. Thus, when we build machine learning models in further steps, the 3 variables are not expected to be used as predictors at the same time. The same is true for “viewPerComment” and “viewCount” and “likeCount”.

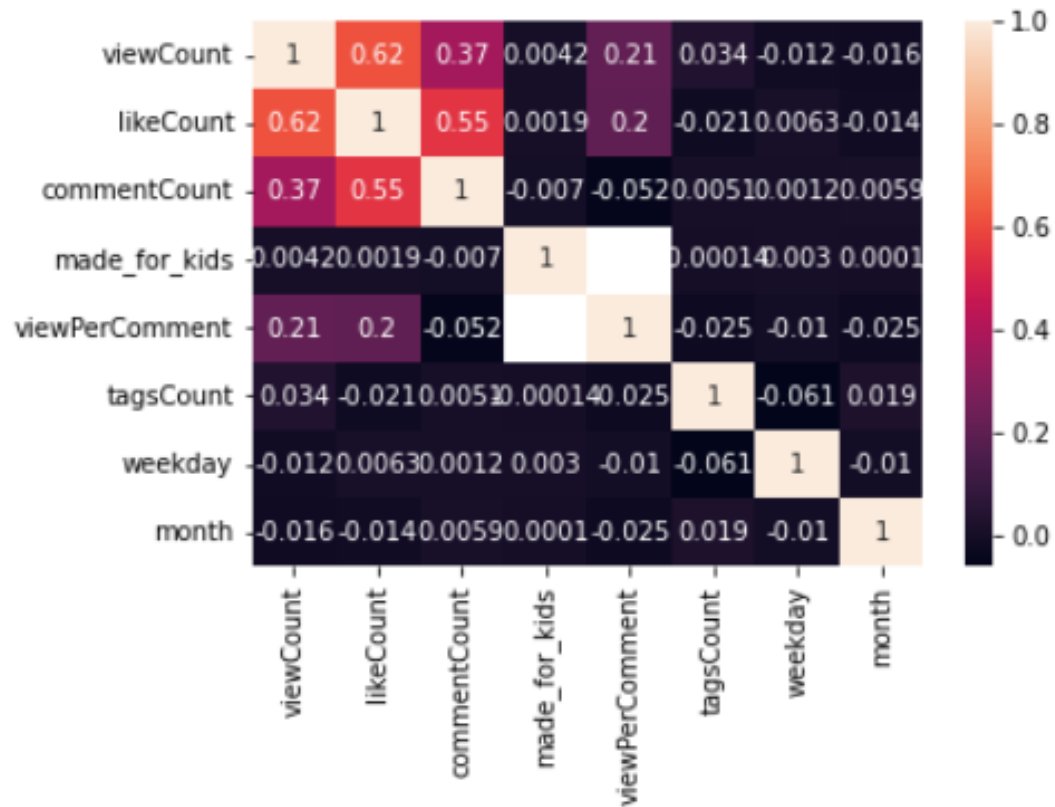


Figure 9. Heat Map of Variable Correlation

Furthermore, we sort the channels based on their view count and get that PewDiePie and MrBeast are the channels that have the highest view shown in **Figure 10**.

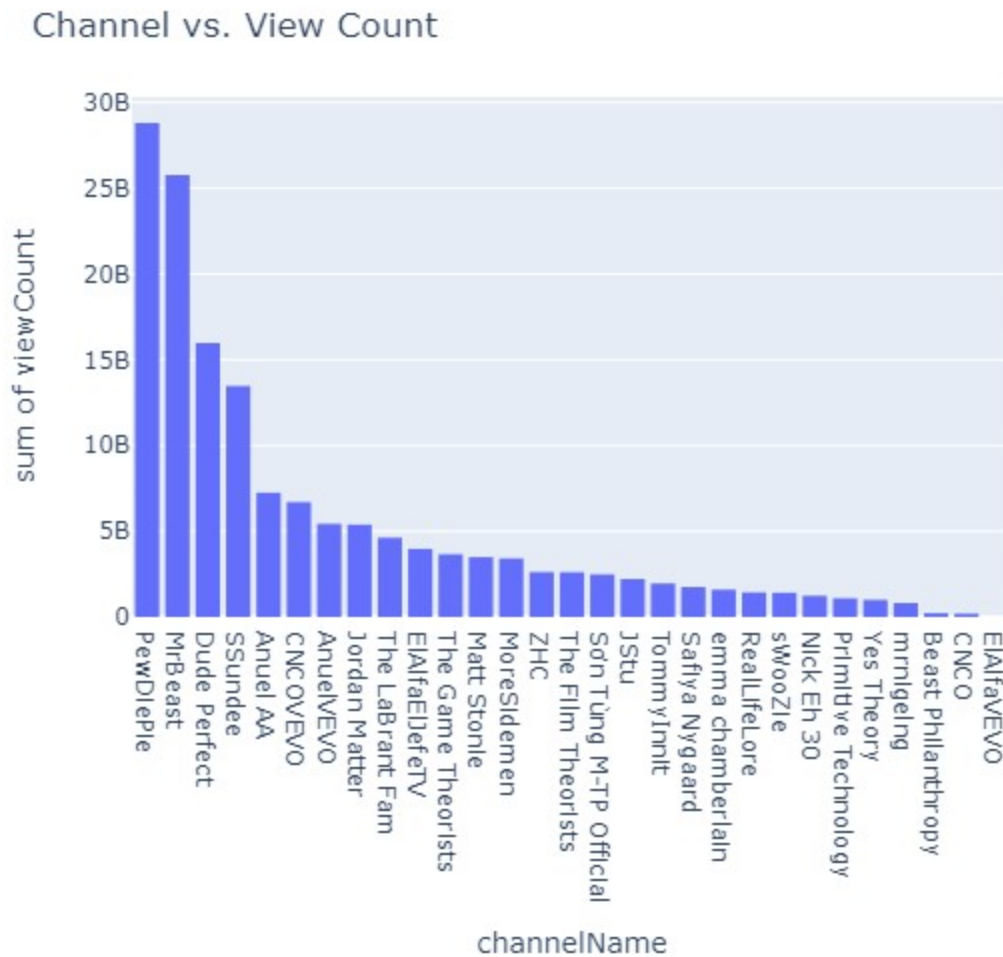


Figure 10. Channel vs. View Count

We are also interested in what specific time interval would help the video get relatively high view, so we create a histogram showing the publish time distribution. As shown in **Figure 11**, the videos published from 17:00 to 20:00 will acquire relatively high view. This also matches with the results shown in **Figure 12** that the videos published in the evening will get much higher view than the other 2 intervals.

Publish Time Distribution

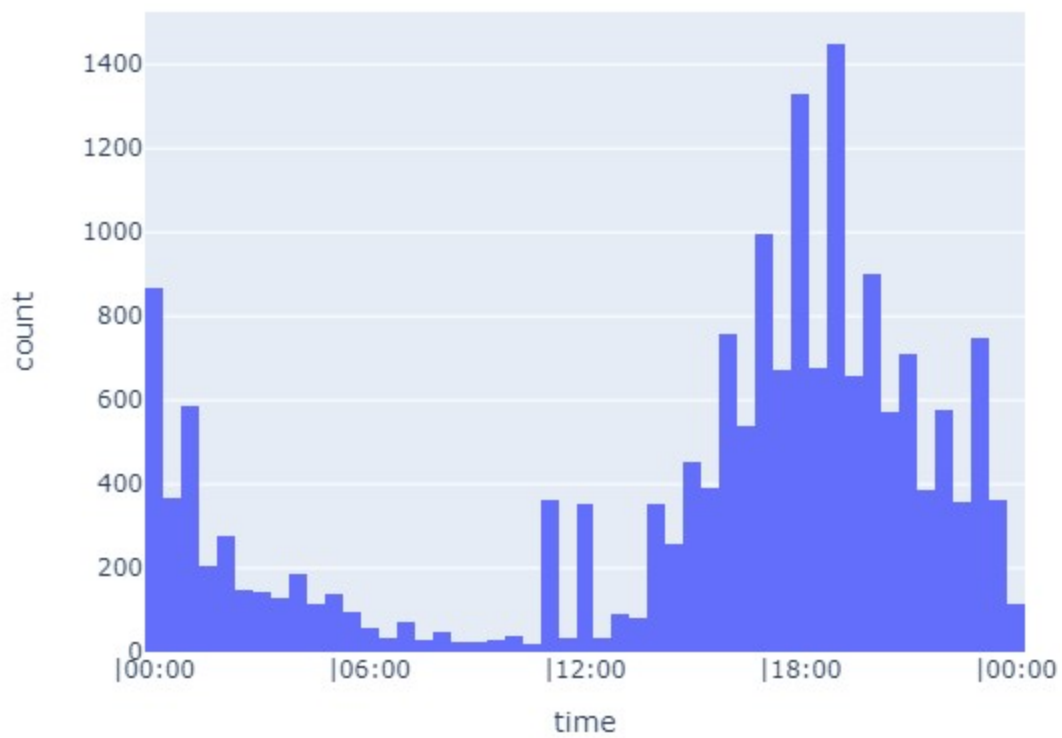


Figure 11. Publish Time Distribution

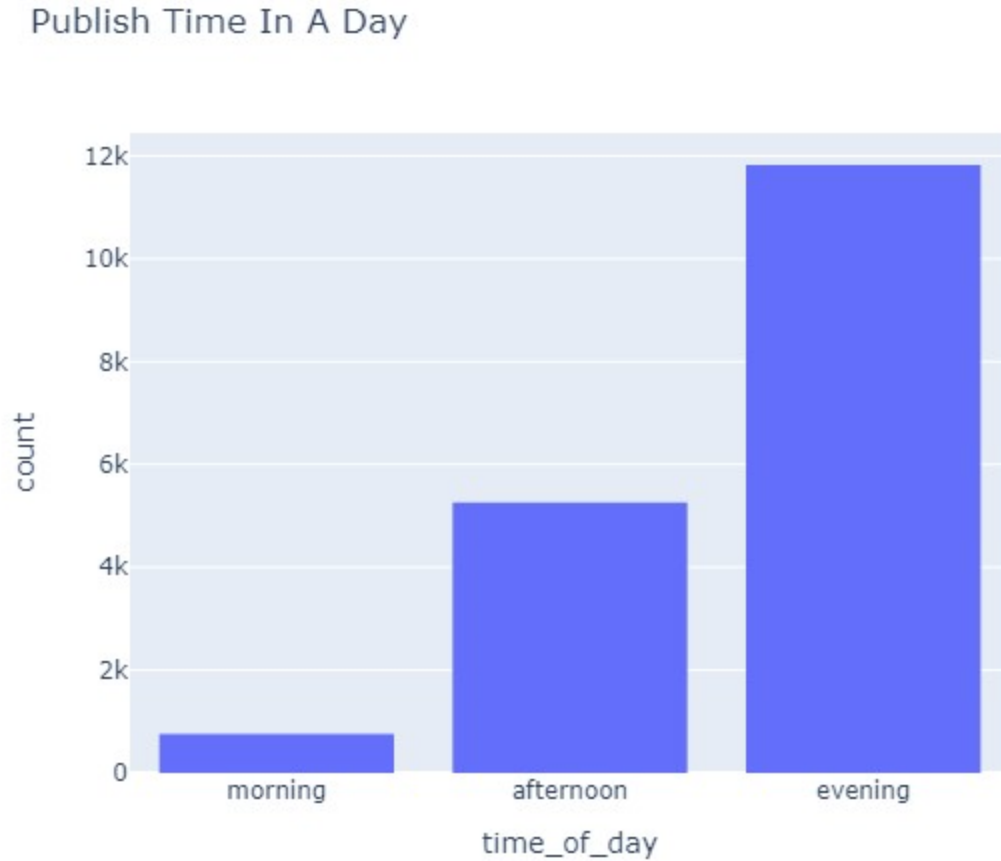


Figure 12. Publish Time Distribution in 3 Time Intervals

When exploring if there exist any apparent pattern between videos' view count, like count, and comment count and the month they published, we create 3 histograms but only the histogram of view count and month shows an obvious pattern that videos posted in January and March will have relatively higher view count than the other months, as shown in **Figure 13**.

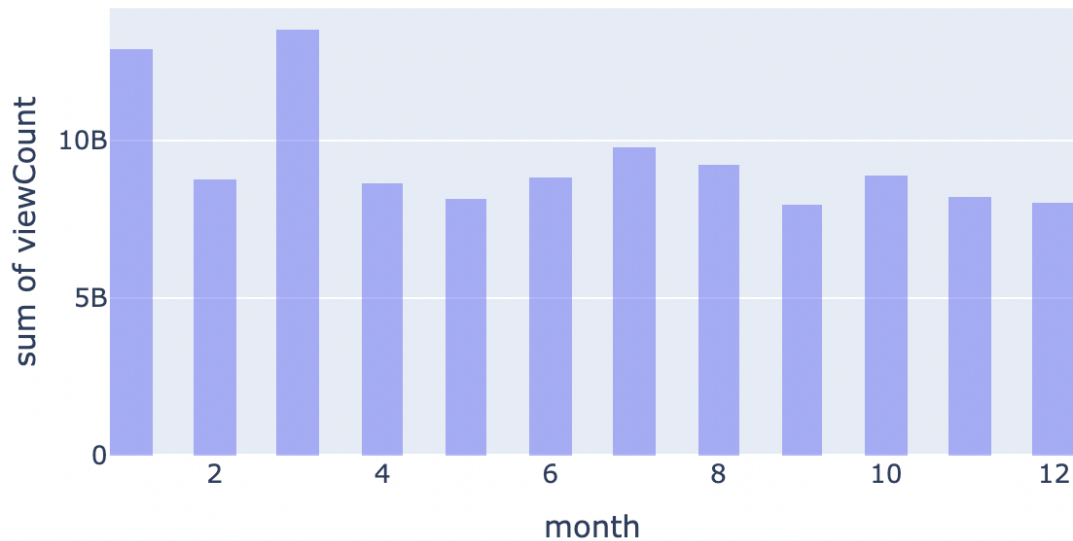


Figure 13. View Count vs. Month

We also create some scatter plot between some variables and title length or tag count that we think might show a significant pattern. And they show that the videos with title length in 20 to 50 characters would have relatively higher likes in **Figure 14**. In **Figure 15** of tag count against the comment counts, the videos with tags less than 50 would tend to have higher comments. In **Figure 16** of view count against title length, the videos with title length in between 20 and 50 would have relatively higher views.

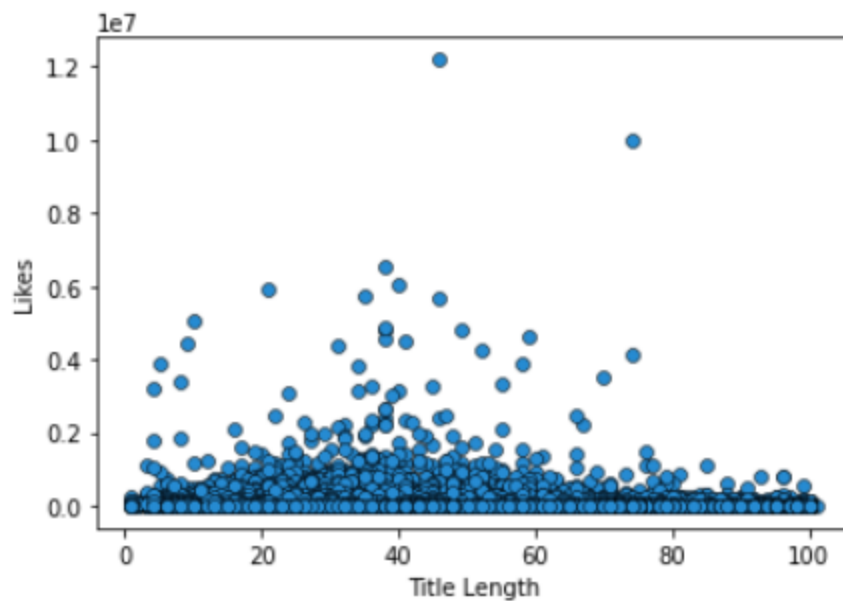


Figure 14. Title Length vs. Likes

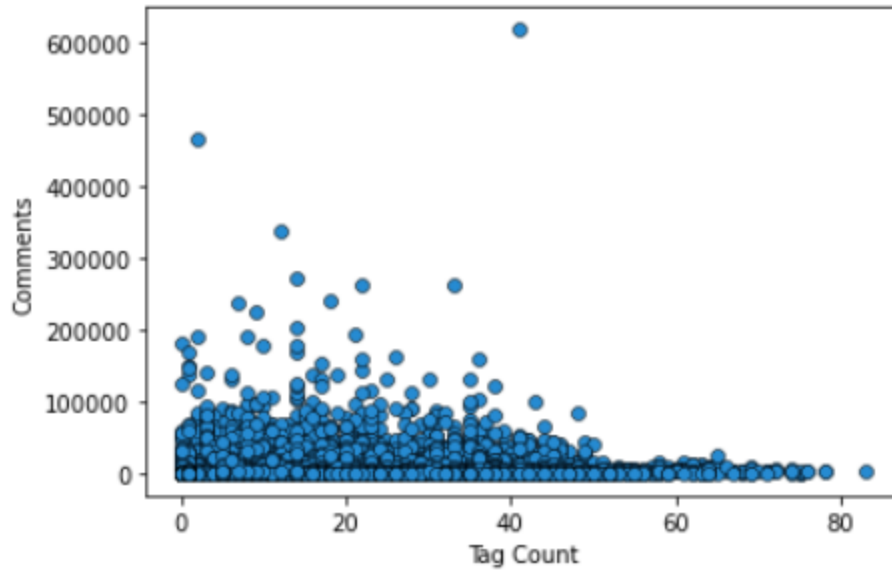


Figure 15. Tag Count vs. Comment Count

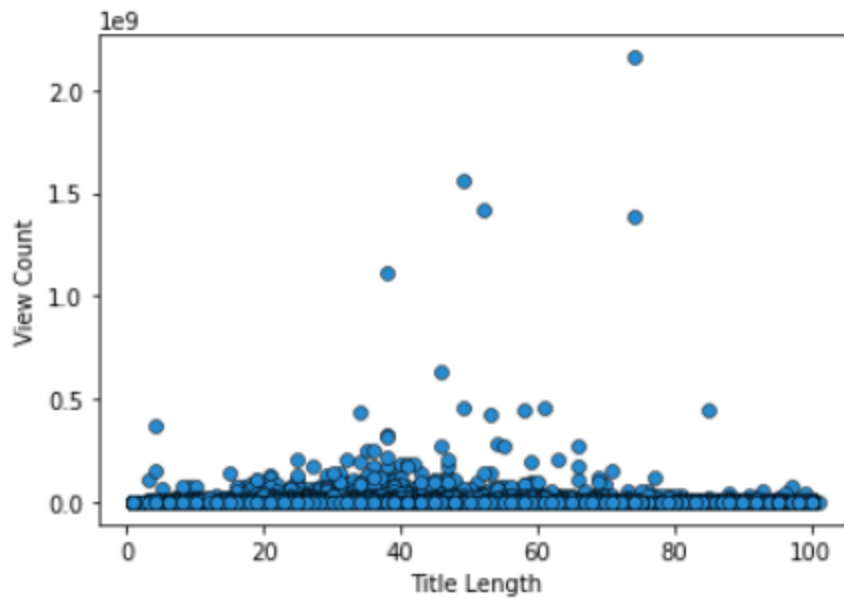


Figure 16. Title Length vs. View Count

Then, we want to explore further in the category column. The top 3 categories with highest view count are Gaming, Entertainment, and Music in **Figure 17**, and we observe that Gaming obtains a higher view count than the videos of other categories. When we are considering the possible relationship between categories and the time that video published, **Figure 18** shows a clear distribution of the 3 time intervals of day in each category. For all categories, videos published in the morning are much fewer than the other 2 intervals, suggesting that the videos

that posted in afternoon and evening would have higher possibility to become popular. Additionally, more videos posted in the evening in some categories such as Autos & Vehicles, Gaming, Music, Nonprofits & Activism, and Sports. Besides, for each category, we listed the percentage of videos in each weekday, and observed that Sports category tend to publish videos more on Tuesdays in **Figure 19**. Interestingly, Autos & Vehicles only publish videos on Thursday, Wednesday and Friday.

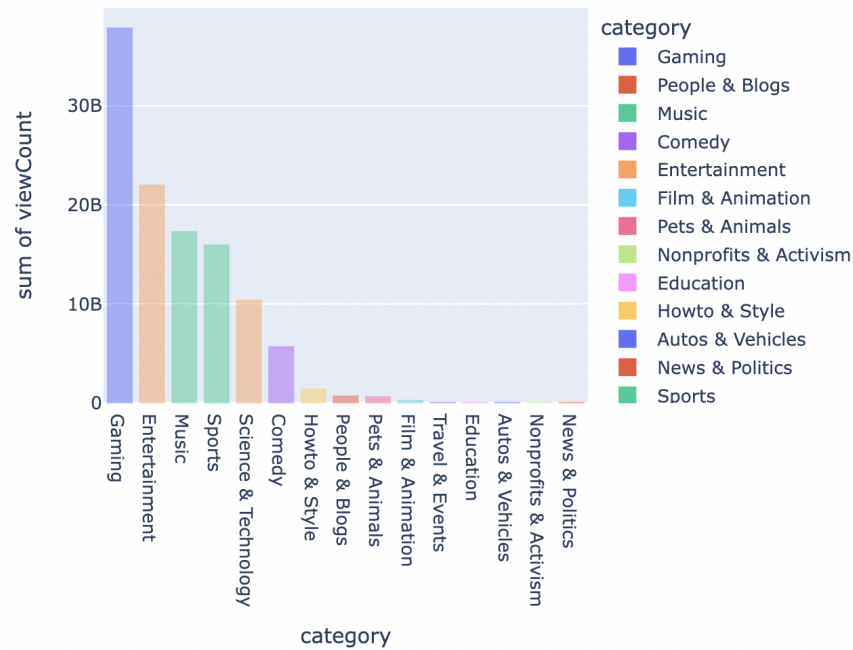


Figure 17. Category vs. View Count

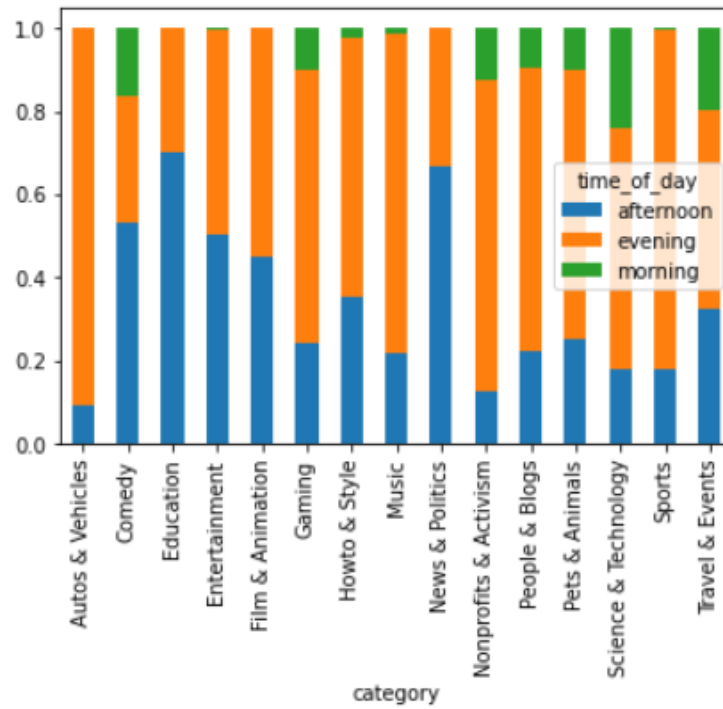


Figure 18. Category vs. Time Published



Figure 19. Distribution of Videos in Weekdays for Each Category

3. Implementation of Machine Learning Models

For building different machine learning models and choosing the model with best performance and highest accuracy, we firstly split our data frame into training and testing dataset and scaling the data with function “MinMaxScaler” for normalization of feature scales as shown in **Figure 20**. We also use recursive feature elimination for variable selection as shown in **Figure 21**, and it demonstrates that the variables “duration”, “made_for_kids”, “year”, and “time_of_day” are the most significant variables.

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
X_train1 = scaler.fit_transform(df)
```

Figure 20. Scaling Data

```
from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression

model = LinearRegression()
rfe = RFE(estimator = model, n_features_to_select = 5)

rfe.fit(X_train, y_train)

selected_features = pd.DataFrame({'Feature': X_train.columns, 'Selected': rfe.support_,
                                'Rank': rfe.ranking_})

print(selected_features[selected_features['Selected']])

X_train = X_train[X_train.columns[rfe.support_]]
print(X_train.head())
```

	Feature	Selected	Rank			
3	duration	True	1			
4	made_for_kids	True	1			
7	year	True	1			
10	time_of_day	True	1			
11	title_length	True	1			
	duration	made_for_kids	year	time_of_day	title_length	
11186	4	0	2015	1	73	
2321	1	0	2016	0	31	
17872	1	0	2013	1	63	
12283	1	0	2020	1	9	
28972	0	0	2022	1	23	

Figure 21. Variable Selection with Recursive Feature Elimination

We then attempt 4 models, which are random forest, gradient boosting, neural network, and XGBoost regression model. By randomly choosing the hyperparameters and fitting the initial

models with the training datasets, we gain the initial testing scores and training scores of the candidate models in **Figure 22**. For improving the performance of the candidate models, we use cross validation to tune the hyperparameters of each model.

- Random Forest: The best parameters are max_depth = 9, n_estimators = 100.
- Gradient Boosting Classifier: The best parameters are learning_rate = 0.05, max_depth = 5, and n_estimators = 150.
- Neural Network: The best parameters are activation = logistic, alpha = 0.3, hidden_layer_sizes = 100, max_iter = 300, and solver = adam.
- XGBoost: The best parameters are learning_rate = 0.1, max_depth = 5, n_estimators = 200.

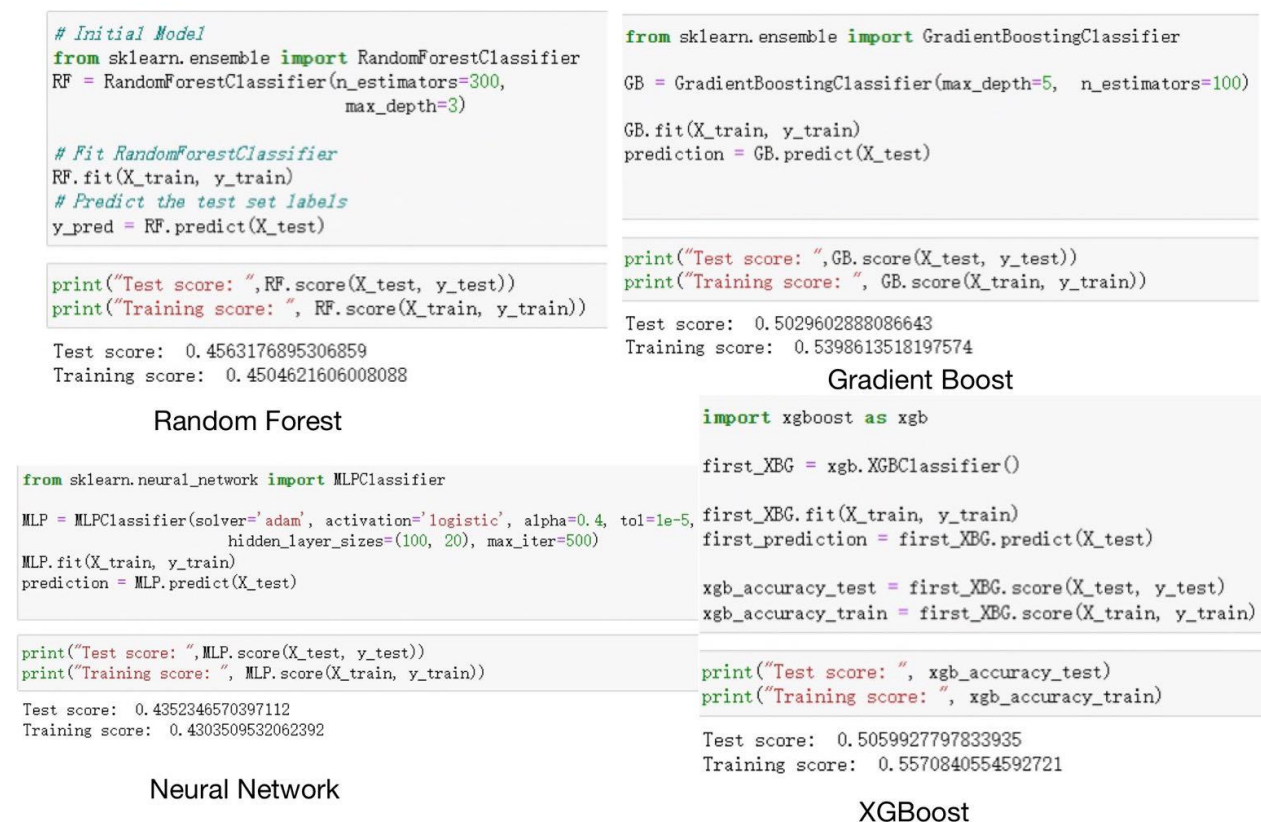


Figure 22. The Four Initial Models

After tuning the hyperparameters, we put the best parameters into the candidate models to get the new test and training score. We also check the precision, recall, and F1 score of the prediction.

- In Random Forest final model (**Figure 23**), the precision is about 0.4537, indicating that out of all the instances predicted as positive by the model, approximately 45.37% of them are actually true positives. The recall of the Random Forest model is also around 0.5017, suggesting that the model is able to correctly identify approximately 50.17% of the positive instances in the test data.
- In Gradient Boost final model (**Figure 24**), the precision score of 0.4777 indicates that out of all instances predicted as positive by the model, approximately 47.77% of them are actually true positives. This indicates a high level of accuracy in predicting positive instances. The recall score of 0.5037 also suggests that the model correctly identifies approximately 50.37% of the positive instances present in the test data. This indicates relatively low level of sensitivity in detecting positive instances.
- In Neural Network final model (**Figure 25**), the precision score of 0.3164 indicates that out of all instances predicted as positive by the model, approximately 31.64% of them are actually true positives. This indicates a high level of accuracy in predicting positive instances. The recall score of 0.4455 also suggests that the model correctly identifies approximately 44.55% of the positive instances present in the test data.
- In XGBoost final mode (**Figure 26**), since the test score is very close to the test score from our initial model, we decided to choose the initial model for simplicity reason. The precision score of 0.4731 indicates that out of all instances predicted as positive by the model, approximately 47.31% of them are actually true positives. This indicates a high level of accuracy in predicting positive instances. The recall score of 0.5060 also suggests that the model correctly identifies approximately 50.60% of the positive instances present in the test data.

```

from sklearn.ensemble import RandomForestClassifier
RF = RandomForestClassifier(n_estimators = 100,
                           max_depth = 9)

RF.fit(X_train, y_train)
y_pred = RF.predict(X_test)
rf_accuracy_test = RF.score(X_test, y_test)
rf_accuracy_train = RF.score(X_train, y_train)

print("Random Forest Test score: ", rf_accuracy_test)
print("Random Forest Training score: ", rf_accuracy_train)

Random Forest Test score:  0.5016606498194945
Random Forest Training score:  0.5309430964760254

## Check precision, recall and F1 score
from sklearn.metrics import precision_score, recall_score, f1_score

rf_precision = precision_score(y_test, y_pred, average='weighted')
rf_recall = recall_score(y_test, y_pred, average='weighted')
rf_f1 = f1_score(y_test, y_pred, average='weighted')

print("Random Forest Precision:", rf_precision)
print("Random Forest Recall:", rf_recall)
print("Random Forest F1 score:", rf_f1)

Random Forest Precision: 0.45372496210614993
Random Forest Recall: 0.5016606498194945
Random Forest F1 score: 0.4257839844163612

```

Figure 23. Random Forest Final Model

```

from sklearn.ensemble import GradientBoostingClassifier
GB = GradientBoostingClassifier(learning_rate=0.05, max_depth=5, n_estimators=150)

GB.fit(X_train, y_train)
prediction = GB.predict(X_test)
gb_accuracy_test = GB.score(X_test, y_test)
gb_accuracy_train = GB.score(X_train, y_train)

print("Test score: ", gb_accuracy_test)
print("Training score: ", gb_accuracy_train)

Test score:  0.5036823104693141
Training score:  0.5327845176198729

## Check precision, recall, and F1 score
gb_precision = precision_score(y_test, prediction, average='weighted')
gb_recall = recall_score(y_test, prediction, average='weighted')
gb_f1 = f1_score(y_test, prediction, average='weighted')

print("Gradient Boosting Classifier Precision:", gb_precision)
print("Gradient Boosting Classifier Recall:", gb_recall)
print("Gradient Boosting Classifier F1 score:", gb_f1)

Gradient Boosting Classifier Precision: 0.47771794850654065
Gradient Boosting Classifier Recall: 0.5036823104693141
Gradient Boosting Classifier F1 score: 0.43929449697331124

```

Figure 24. Gradient Boost Final Model

```

from sklearn.neural_network import MLPClassifier
MLP = MLPClassifier(solver='adam', activation='logistic', alpha=0.3,
                    hidden_layer_sizes=(100, ), max_iter=300)
MLP.fit(X_train, y_train)
prediction = MLP.predict(X_test)
mlp_accuracy_test = GB.score(X_test, y_test)
mlp_accuracy_train = GB.score(X_train, y_train)

print("Test score: ", mlp_accuracy_test)
print("Training score: ", mlp_accuracy_train)

Test score:  0.5036823104693141
Training score:  0.5327845176198729

## Check precision and recall
mlp_precision = precision_score(y_test, prediction, average='weighted')
mlp_recall = recall_score(y_test, prediction, average='weighted')
mlp_f1 = f1_score(y_test, prediction, average='weighted')

print("Nerual Network Precision:", mlp_precision)
print("Nerual Network Recall:", mlp_recall)
print("Nerual Network F1 score:", mlp_f1)

Nerual Network Precision: 0.31642019139600136
Nerual Network Recall: 0.44548736462093863
Nerual Network F1 score: 0.3521251823047284

```

Figure 25. Neural Network Final Model

```

from sklearn.metrics import precision_score, recall_score

xgb_precision = precision_score(y_test, first_prediction, average='weighted')
xgb_recall = recall_score(y_test, first_prediction, average='weighted')
xgb_f1 = f1_score(y_test, first_prediction, average = 'weighted')

print("XGBoost Classifier Precision:", xgb_precision)
print("XGBoost Classifier Recall:", xgb_recall)
print("XGBoost Classifier F1 score:", xgb_f1)

XGBoost Classifier Precision: 0.4731449244591162
XGBoost Classifier Recall: 0.5059927797833935
XGBoost Classifier F1 score: 0.44812284468874547

```

Figure 26. XGBoost Final Model

For comparing the 4 candidate models, we create bar charts for their accuracy, precision, recall, and F1-score as measures to consider their performances as shown in **Figure 27**. Based on the bar charts, gradient boosting classifier and XGBoost are the top 2 models with the best metrics. Although both Gradient Boosting and XGBoost show excellent performance, XGBoost is preferred since its initial model was picked. As a result, we chose the XGBoost model because to its performance, efficiency, and simplicity.

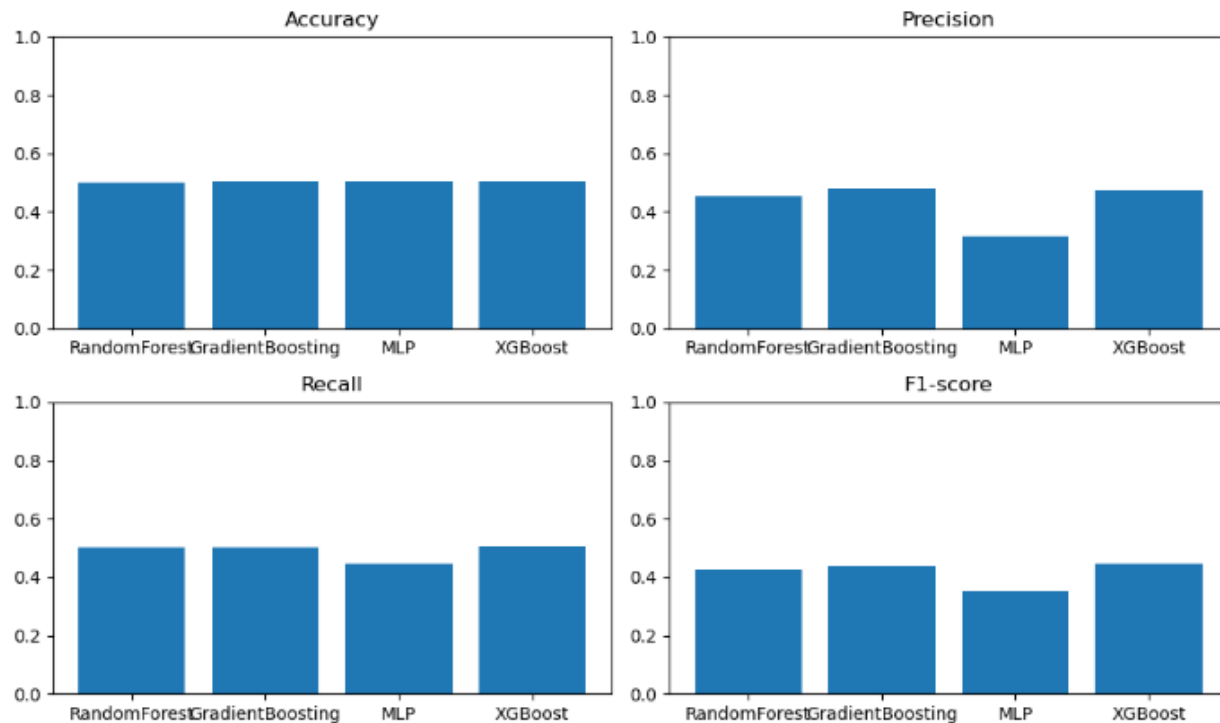


Figure 27. Model Comparison

4. Building a Dynamic Website

After deciding our best performance model, we deploy our final model using Flask, which is a web application framework written in Python. We set up a Flask application by creating a Python file and importing the necessary Flask modules. Then we load and preprocess the model by loading the trained final model and any required preprocessing steps within the Flask application. Moreover, we define the routes and views for the Flask application, specifying the endpoints for accessing the model predictions using the Pickle package in Python. When the general outline of the Flask application is built, we implement logic to handle user inputs through HTML forms and pass the input to the model for prediction. For example, from **Figure 28** of the HTML-related code, the structure we built includes setting up the YouTube logo, Website headers, reference tables, and etc. Additionally, we apply css file for stylistic modification, including font style, color, and margins. Each step of Flask developments is shown in the code snippet **Figure 29**. Finally, we display the model's prediction results to the users on a web page.

```

<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Youtube Video Popularity Prediction</title>
  <link href="https://fonts.googleapis.com/css?family=Pacifico" rel="stylesheet" type="text/css">
  <link href="https://fonts.googleapis.com/css?family=Arimo" rel="stylesheet" type="text/css">
  <link href="https://fonts.googleapis.com/css?family=Hind:300" rel="stylesheet" type="text/css">
  <link href="https://fonts.googleapis.com/css?family=Open+Sans+Condensed:300" rel="stylesheet" type="text/css">

  <link rel="stylesheet" href="{{ url_for('static', filename='css/v.css') }}" type="text/css">

  <style>
    .logo-image-1 {
      width: 40%;
      display: block;
    }

    .logo-image-2 {
      width: 70%;
      display: block;
      margin: 0 auto;
    }

    .table-container {
      display: flex;
      margin-right: 10px;
    }

    table {
      width: 300px;
      margin-right: 20px;
    }

    th, td {
      border: 1px solid black;
      padding: 8px;
      text-align: left;
    }

    th {
      background-color: #FF0000;
    }
  </style>
</head>
<body>
  <div class="login">
    
    <h1>Youtube Video Popularity Prediction</h1>
    <h3>Enter the following values to predict your Youtube video view count</h3>

    <!-- Main Input For Receiving Query to our ML -->
    <form id="prediction-form" action="/predict" method="post">
      <input type="text" name="durationSL" placeholder="Duration" required="required" />
      <input type="text" name="made_for_kidsSL" placeholder="Made for Kids" required="required" />
      <input type="text" name="yearSL" placeholder="Year" required="required" />
      <input type="text" name="time_of_daySL" placeholder="Time of Day" required="required" />
      <input type="text" name="title_lengthSL" placeholder="Title Length" required="required" />

      <button type="submit" class="btn">Predict</button>
    </form>
  </div>
</body>
</html>

```

Figure 28. HTML Structure

```

import numpy as np
import pickle
from flask import Flask, render_template, request, url_for

# Initialize the Flask application
app = Flask(__name__, static_url_path='/static')
# Load model
model = pickle.load(open('model.pkl', 'rb'))

# Default page of our web-app
# What we type into our browser to go to different pages
@app.route('/')
def home():
    return render_template('index.html')

# To use the predict button in our web-app
@app.route('/predict', methods=['POST'])
def predict():
    # For rendering results on HTML GUI
    int_features = [float(x) for x in request.form.values()]
    final_features = np.array(int_features).reshape(1, -1)

    # Make predictions using the loaded model
    prediction = model.predict(final_features)
    output = round(prediction[0], 2)

    # Determine the view count range based on the prediction result
    if output == 0:
        view_count_range = '0-10K'
    elif output == 1:
        view_count_range = '10K-100K'
    elif output == 2:
        view_count_range = '100K-500K'
    elif output == 3:
        view_count_range = '500K-1M'
    elif output == 4:
        view_count_range = '1M-5M'
    elif output == 5:
        view_count_range = '5M-10M'
    elif output == 6:
        view_count_range = '10M-50M'
    elif output == 7:
        view_count_range = '50M-100M'
    elif output == 8:
        view_count_range = '100M-500M'
    else:
        view_count_range = '>500M'

    # Render the prediction result on a new page
    return render_template('index.html', prediction_text=view_count_range)

if __name__ == "__main__":
    app.run(debug=True)

```

Figure 29. Flask Development

In **Figure 30**, during data preprocessing, we encode the response variable, view count and the video duration for convenience and efficiency. These separation of levels help us set up the reference tables for input and predict categorical levels of view count. As shown in **Figure 31**, our website required the users to enter totally 5 inputs, which are duration, made for kids (whether the video is made for kids), year (publish year), time of day (published time of day), and title length. These are the significant features selected by the recursive feature elimination, which will help users to get relatively accurate prediction by our model. We tested our website and it successfully shows prediction for the users' inputs, and the reference tables as menu for the users to check the inputs they entered also show in the website. It will show the prediction of levels of view count as final output.

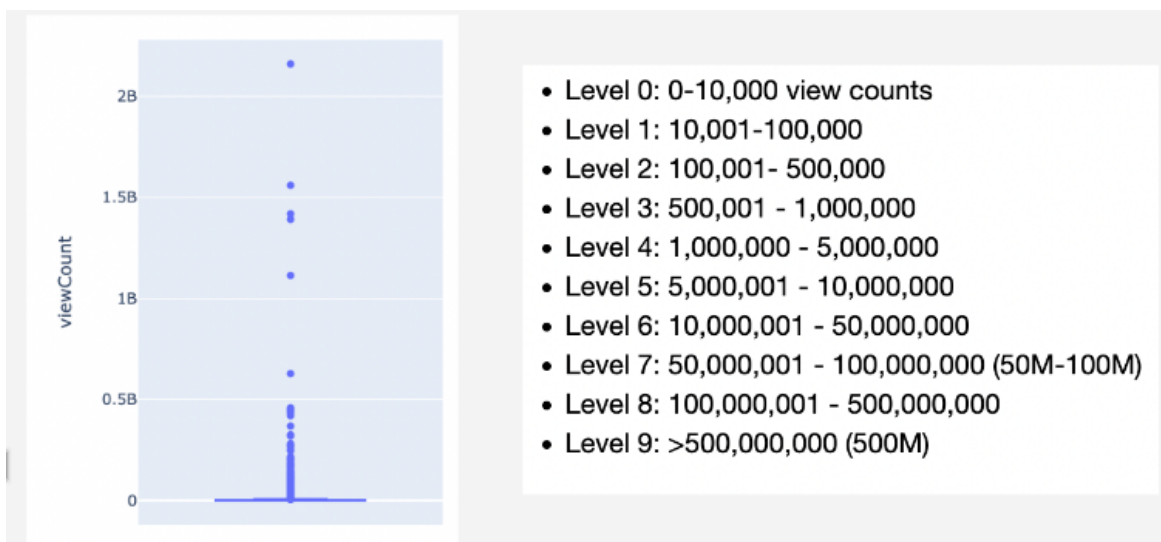



Figure 30. Levels of View Count



Youtube Video Popularity Prediction

Enter the following values to predict your Youtube video view count

Prediction Result: 100K-500K

Use tables below for reference.

ID	Duration
0	Less than 5 min
1	Less than 10 min
2	Less than 30 min
3	Less than 60 min
4	Less than 2 hours
5	> 2 hours

ID	Made for Kids
0	No
1	Yes

ID	Time of the Day
2	morning
0	afternoon
1	evening

Figure 31. Website, <https://youtube-prediction.herokuapp.com/>

Ethics & Conclusion

So far, our website is only applicable for the trendy influencers with a certain subscriptions and large view counts as popularity on YouTube since our model only applied to trendy video data. Thus, one of our next steps is to generalize the model to a broader user-base by acquiring additional data and non-trendy YouTube video data and retraining the model accordingly. Besides, we also consider employing more advanced techniques such as Natural Language Processing (NLP) to improve the accuracy of the model. In addition to only predicting the view count, we expect to predict other metrics such as like counts and comment counts for providing users a more detailed and accurate prediction. For improving the overall user experience on our website, we also want to enhance the web server functionality by incorporating user-friendly features, such as dropping down the reference menus, in the future.

For the potential ethical risks that may occur in our website, YouTube influencers who uploaded their video data may feel pressure to conform to the algorithm's prediction in order to increase the popularity of their videos. And this may also cause a homogenization of the content since the influencers are more likely to make videos that fit with the prediction rather than the videos they want to make. In addition, when the influencers change the type of their videos in order to increase the popularity, the fixed group of audiences of their previous videos would be lost and harmed. Audiences for niche videos will be hurt by the loss of videos that interest them. The essence of these two ethical ramifications comes from the homogeneous predictive tendencies of our prediction model. Thus, while attempting to increase the accuracy of our model, we will also try to reduce the homogeneous bias in our model and develop a more inclusive model in the future improvement.