Load the Data

In [1]:
```
import pandas as pd
import zipfile
import os

# Function to load CSV from a ZIP file with multiple files
def load_csv_from_zip(zip_path, csv_filename):
    with zipfile.ZipFile(zip_path, 'r') as z:
        # Extract and read the specific CSV file
        with z.open(csv_filename) as f:
            return pd.read_csv(f)

# Define the relative path to the datasets folder
datasets_path = os.path.join('..', 'Datasets')

# Load datasets from zipped CSV files specifying the correct CSV filen
df_gb = load_csv_from_zip(os.path.join(datasets_path, 'GBvideos.csv.zi
df_us = load_csv_from_zip(os.path.join(datasets_path, 'USvideos.csv.zi

# Add a new column 'location' in each data file
df_gb['location'] = 'Great Britain'
df_us['location'] = 'USA'

# Merge 5 files into 1
merged_df = pd.concat([df_gb, df_us], ignore_index=True)

# Check the first few rows of the merged DataFrame
print(merged_df.head())
```

```
      video_id trending_date  \
0  Jw1Y-zhQURU      17.14.11
1  3s1rvMFUweQ      17.14.11
2  n1WpP7iowLc      17.14.11
3  PUTEiSjKwJU      17.14.11
4  rHwDegptbI4      17.14.11

                                               title  \
0     John Lewis Christmas Ad 2017 - #MozTheMonster
1         Taylor Swift: …Ready for It? (Live) - SNL
2        Eminem - Walk On Water (Audio) ft. Beyoncé
3  Goals from Salford City vs Class of 92 and Fri...
4  Dashcam captures truck's near miss with child ...

                channel_title  category_id              publish_time
\
0                  John Lewis           26  2017-11-10T07:38:29.000Z
1         Saturday Night Live           24  2017-11-12T06:24:44.000Z
2                  EminemVEVO           10  2017-11-10T17:00:03.000Z
```

Check Missing Values

In [2]:
```
# Check for missing values in the merged DataFrame
print("Missing values")
print(merged_df.isnull().sum())
```

```
Missing values
video_id                  0
trending_date             0
title                     0
channel_title             0
category_id               0
publish_time              0
tags                      0
views                     0
likes                     0
dislikes                  0
comment_count             0
thumbnail_link            0
comments_disabled         0
ratings_disabled          0
video_error_or_removed    0
description            1182
location                  0
dtype: int64
```

In [3]:
```
df = merged_df.dropna()
```

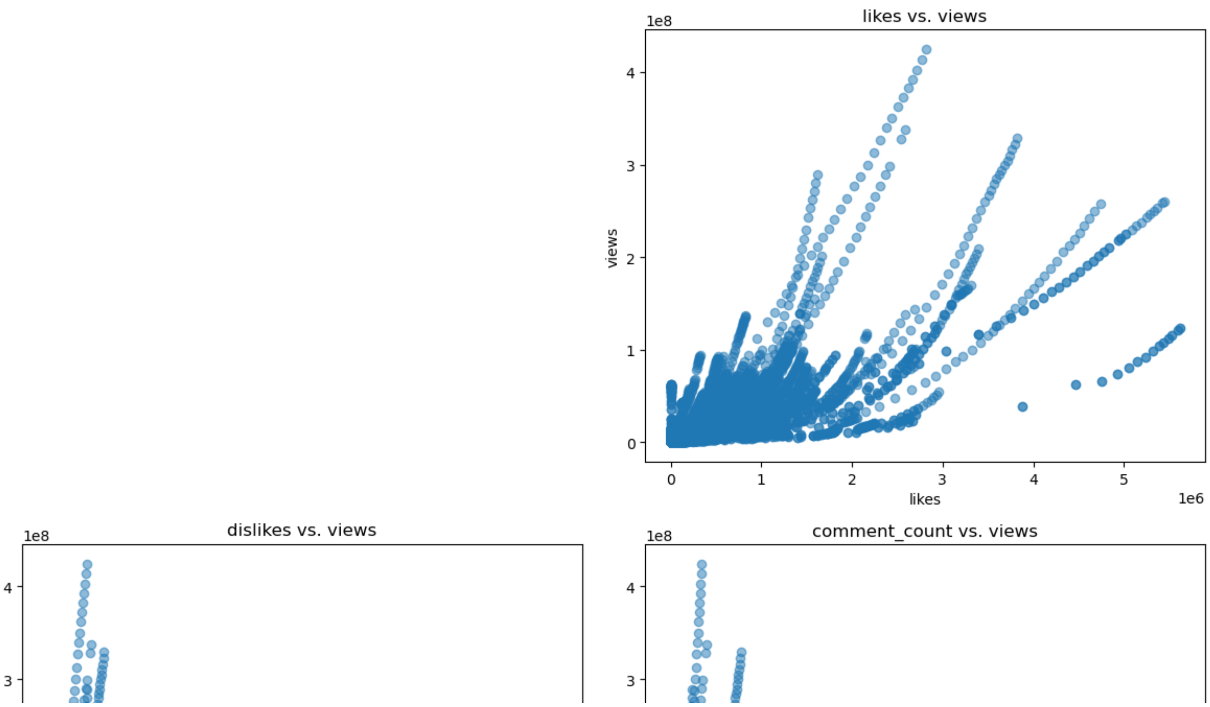
In [4]:
```
# Check for missing values in the merged DataFrame
print("Missing values")
print(df.isnull().sum())
```

```
Missing values
video_id                  0
trending_date             0
title                     0
channel_title             0
category_id               0
publish_time              0
tags                      0
views                     0
likes                     0
dislikes                  0
comment_count             0
thumbnail_link            0
comments_disabled         0
ratings_disabled          0
video_error_or_removed    0
description               0
location                  0
dtype: int64
```

Exploratory Data Analysis (EDA)

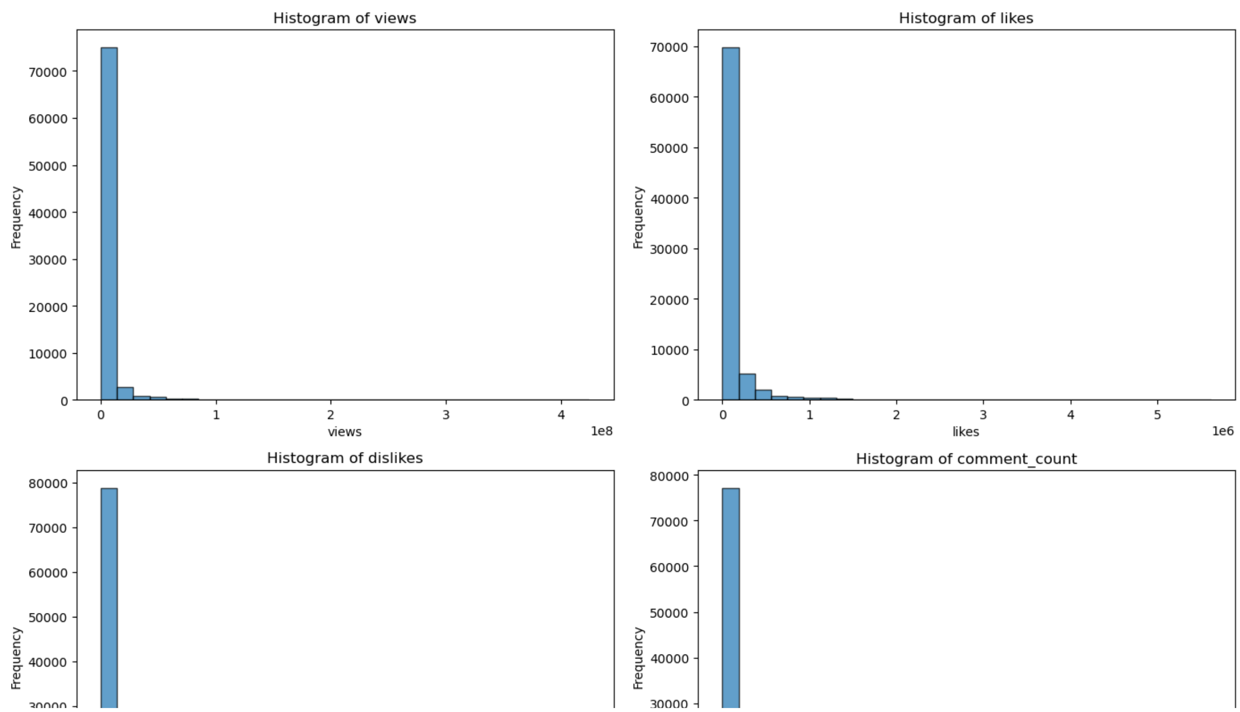
Check Outliers

In [5]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Define numerical columns
numerical_columns = ['views', 'likes', 'dislikes', 'comment_count']


# Scatter plots for each numerical column vs. 'views'
plt.figure(figsize=(12, 10))
for i, column in enumerate(numerical_columns, 1):
    if column != 'views':
        plt.subplot(2, 2, i)
        plt.scatter(merged_df[column], merged_df['views'], alpha=0.5)
        plt.title(f'{column} vs. views')
        plt.xlabel(column)
        plt.ylabel('views')

plt.tight_layout()
plt.show()
```



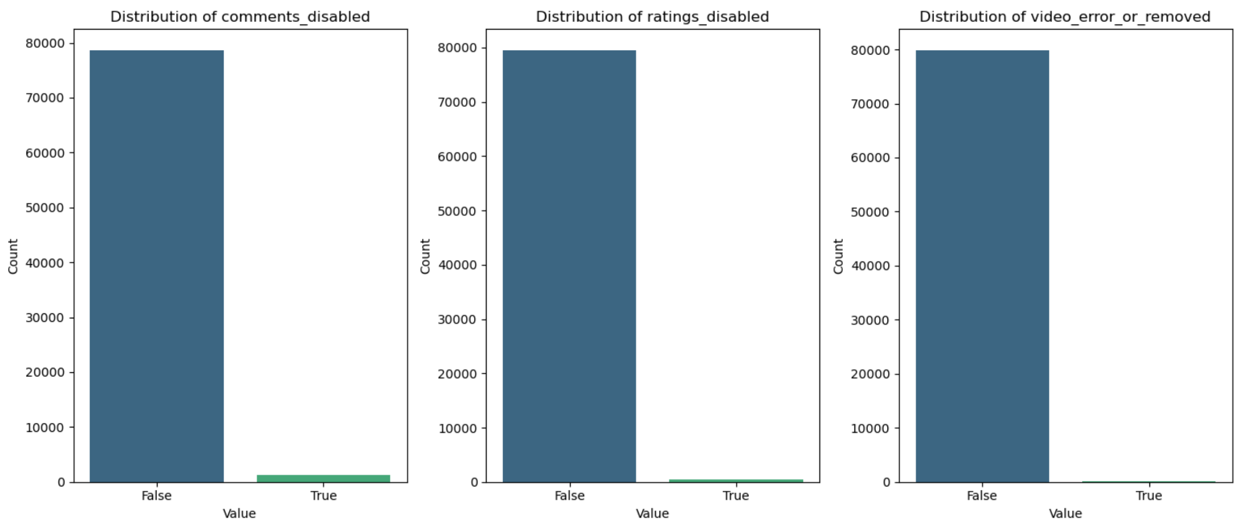
EDA for Numerical Variables

In [6]:
```
#data exploration for numerical columns
import matplotlib.pyplot as plt

# Define numerical columns
numerical_columns = ['views', 'likes', 'dislikes', 'comment_count']

# Create histograms for each numerical column
plt.figure(figsize=(14, 10))
for i, column in enumerate(numerical_columns, 1):
    plt.subplot(2, 2, i)
    plt.hist(merged_df[column], bins=30, alpha=0.7, edgecolor='black')
    plt.title(f'Histogram of {column}')
    plt.xlabel(column)
    plt.ylabel('Frequency')

plt.tight_layout()
plt.show()
```



EDA for Boolean Variables

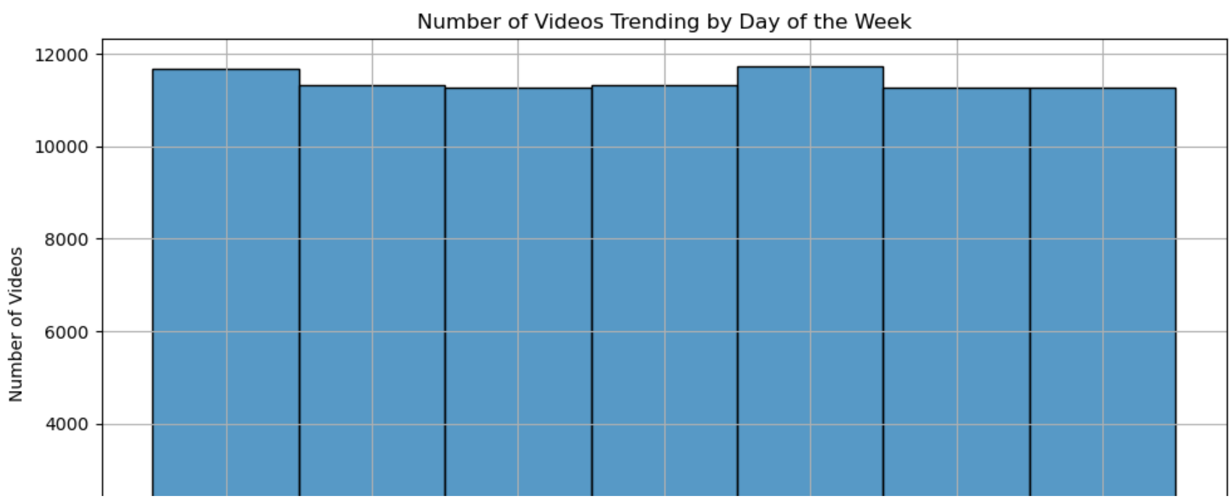
In [7]:
```
import seaborn as sns

# Define boolean columns
boolean_columns = ['comments_disabled', 'ratings_disabled', 'video_err

# Plot bar plots for each boolean column
plt.figure(figsize=(14, 6))
for i, column in enumerate(boolean_columns, 1):
    plt.subplot(1, 3, i)
    # Count the occurrences of each boolean value
    counts = merged_df[column].value_counts()
    # Plot bar plot
    sns.barplot(x=counts.index, y=counts.values, palette='viridis')
    plt.title(f'Distribution of {column}')
    plt.xlabel('Value')
    plt.ylabel('Count')

plt.tight_layout()
plt.show()
```



EDA for Date-Time Variables

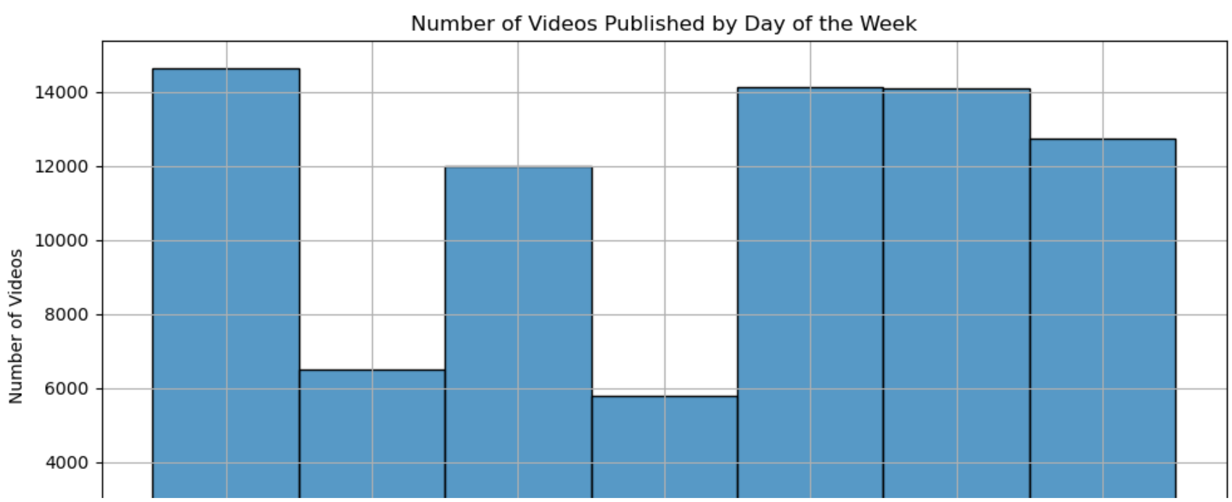
In [8]:
```
# convert the trending_date to datetime type
merged_df['trending_date'] = pd.to_datetime(merged_df['trending_date']
# Extract day of the week from 'trending_date'
merged_df['trending_day_of_week'] = merged_df['trending_date'].dt.day_

# Plot histogram of trending day of the week
plt.figure(figsize=(10, 6))
sns.histplot(merged_df['trending_day_of_week'], discrete=True, palette
plt.title('Number of Videos Trending by Day of the Week')
plt.xlabel('Day of the Week')
plt.ylabel('Number of Videos')
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
/var/folders/yn/hnpfh1r15tq8t0xq_j4_rzmh0000gn/T/ipykernel_90522/1621
989413.py:8: UserWarning: Ignoring `palette` because no `hue` variabl
e has been assigned.
  sns.histplot(merged_df['trending_day_of_week'], discrete=True, pale
tte='viridis')
```



In [9]:
```
#convert the publish_date to datetime type
merged_df['publish_time'] = pd.to_datetime(merged_df['publish_time'],
# Extract day of the week from 'publish_time'
merged_df['day_of_week'] = merged_df['publish_time'].dt.day_name()

# Plot histogram of day of the week
plt.figure(figsize=(10, 6))
sns.histplot(merged_df['day_of_week'], discrete=True, palette='viridis
plt.title('Number of Videos Published by Day of the Week')
plt.xlabel('Day of the Week')
plt.ylabel('Number of Videos')
plt.xticks(rotation=45)
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
/var/folders/yn/hnpfh1r15tq8t0xq_j4_rzmh0000gn/T/ipykernel_90522/1346
077495.py:8: UserWarning: Ignoring `palette` because no `hue` variabl
e has been assigned.
  sns.histplot(merged_df['day_of_week'], discrete=True, palette='viri
dis')
```



Statistical Description

In [10]:
```
numerical_description = merged_df.describe()
print(numerical_description)
```

```
                       trending_date   category_id  \
count                          79865  79865.000000
mean   2018-02-25 07:57:45.132410880     18.440205
min              2017-11-14 00:00:00      1.000000
25%              2018-01-02 00:00:00     10.000000
50%              2018-02-23 00:00:00     22.000000
75%              2018-04-21 00:00:00     24.000000
max              2018-06-14 00:00:00     43.000000
std                              NaN      7.818304

                        publish_time         views         likes  \
count                          79865  7.986500e+04  7.986500e+04
mean   2018-01-30 08:51:14.599436544  4.091166e+06  1.036262e+05
min              2006-07-23 08:24:11  5.490000e+02  0.000000e+00
25%              2017-12-22 15:58:16  2.464170e+05  5.642000e+03
50%              2018-02-14 05:01:24  7.961060e+05  2.092200e+04
75%              2018-04-09 08:59:51  2.535704e+06  7.824800e+04
max              2018-06-14 01:31:53  4.245389e+08  5.613827e+06
std                              NaN  1.439125e+07  2.957265e+05
```

In [11]:
```
# Statistical description of categorical columns
categorical_description = merged_df[['category_id', 'location']].descr
print(categorical_description)
```

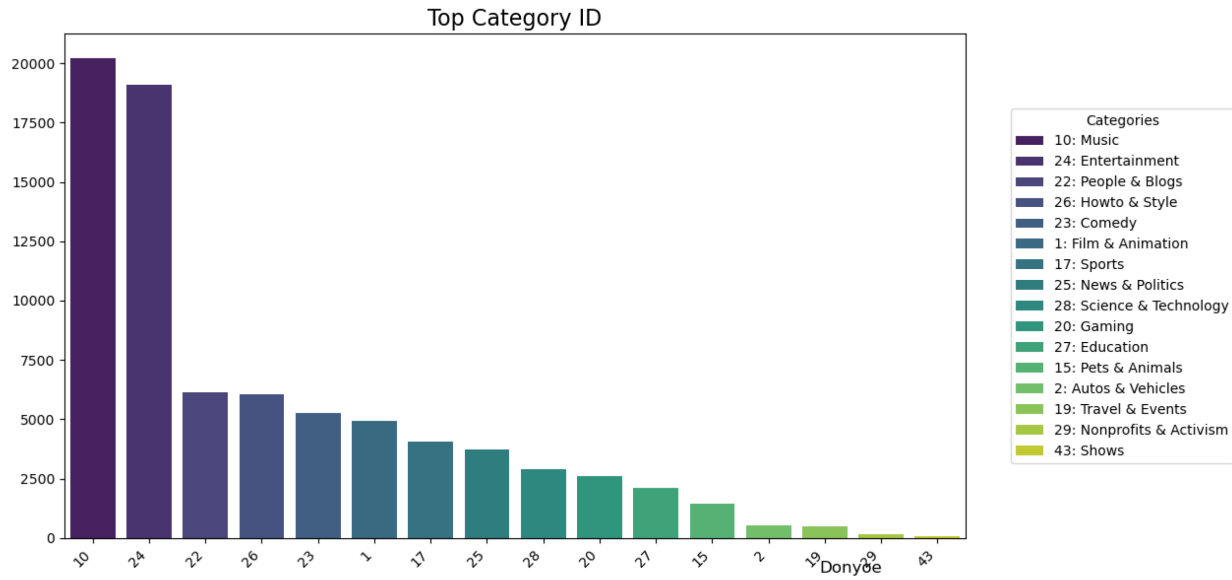
```
        category_id
count  79865.000000
mean      18.440205
std        7.818304
min        1.000000
25%       10.000000
50%       22.000000
75%       24.000000
max       43.000000
```

Visualization for Categorical ID

In [12]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Group by 'category_id' and count occurrences
category_counts = merged_df.groupby('category_id').size().reset_index(

# Sort by 'N' in descending order
```

```
category_counts = category_counts.sort_values(by='N', ascending=False)
category_counts['category_id'] = pd.Categorical(category_counts['categ

# Create a dictionary to map 'category_id' to descriptive names
category_names = {
    1: "1: Film & Animation",
    2: "2: Autos & Vehicles",
    10: "10: Music",
    15: "15: Pets & Animals",
    17: "17: Sports",
    18: "18: Short Movies",
    19: "19: Travel & Events",
    20: "20: Gaming",
    21: "21: Videoblogging",
    22: "22: People & Blogs",
    23: "23: Comedy",
    24: "24: Entertainment",
    25: "25: News & Politics",
    26: "26: Howto & Style",
    27: "27: Education",
    28: "28: Science & Technology",
    29: "29: Nonprofits & Activism",
    30: "30: Movies",
    31: "31: Anime/Animation",
    32: "32: Action/Adventure",
    33: "33: Classics",
    34: "34: Comedy",
    35: "35: Documentary",
    36: "36: Drama",
    37: "37: Family",
    38: "38: Foreign",
    39: "39: Horror",
    40: "40: Sci-Fi/Fantasy",
    41: "41: Thriller",
    42: "42: Shorts",
    43: "43: Shows",
    44: "44: Trailers"
}

# Map 'category_id' to names in the 'category_counts' DataFrame
category_counts['category_name'] = category_counts['category_id'].map(

# Plot using seaborn
plt.figure(figsize=(10, 6))
barplot = sns.barplot(data=category_counts, x='category_id', y='N', pa

# Customize the plot to match your ggplot2 example
plt.title("Top Category ID", fontsize=16)
plt.xlabel(None)
plt.ylabel(None)
```

```
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.figtext(0.9, 0.02, "Donyoe", horizontalalignment='right', fontsize

# Add a custom legend for category names on the side
handles = barplot.patches
legend_labels = [category_names[int(c)] for c in category_counts['cate

# Position the legend on the right of the plot using 'bbox_to_anchor'
plt.legend(handles=handles[:len(legend_labels)], labels=legend_labels,
           bbox_to_anchor=(1.05, 0.5), loc='center left', borderaxespa

plt.show()
```



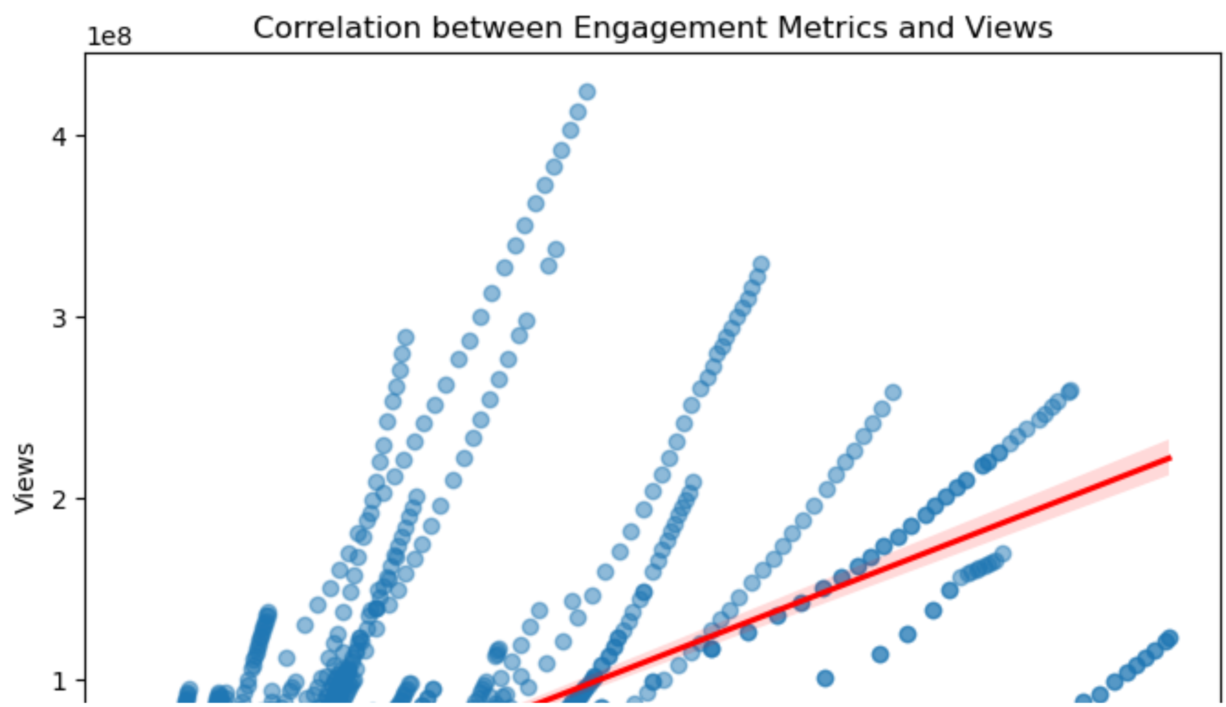
Data Transformation-Create Engagement Metrics

In [13]:
```
# Create a new column
merged_df['Engagement Metrics'] = merged_df['likes'] + merged_df['disl
# Display the DataFrame to check the new column
print(merged_df[['likes', 'dislikes', 'comment_count', 'Engagement Met
```

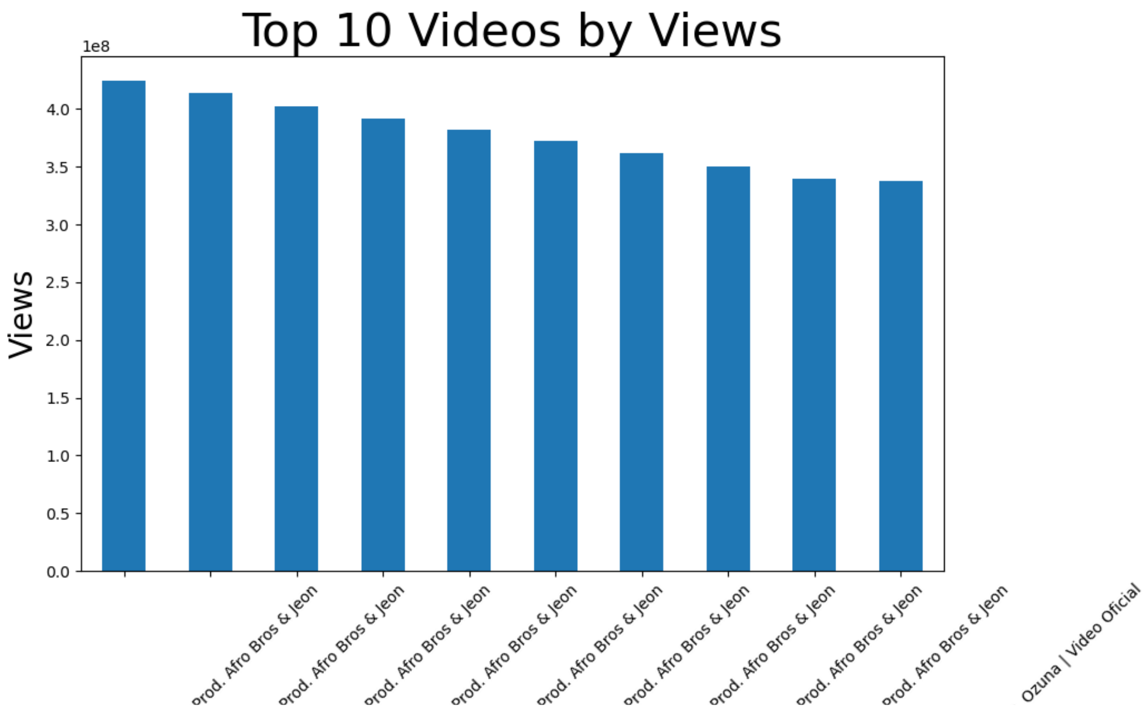
```
    likes  dislikes  comment_count  Engagement Metrics
0   55681     10247           9479               75407
1   25561      2294           2757               30612
2  787420     43420         125882              956722
3     193        12             37                 242
4      30         2             30                  62
```

Visualization-Engagement Metrics

In [14]:
```
# Create a scatter plot with a regression line
plt.figure(figsize=(8, 6))
sns.regplot(x='Engagement Metrics', y='views', data=merged_df, scatter
plt.title('Correlation between Engagement Metrics and Views')
plt.xlabel('Engagement Metrics')
plt.ylabel('Views')
plt.show()
```



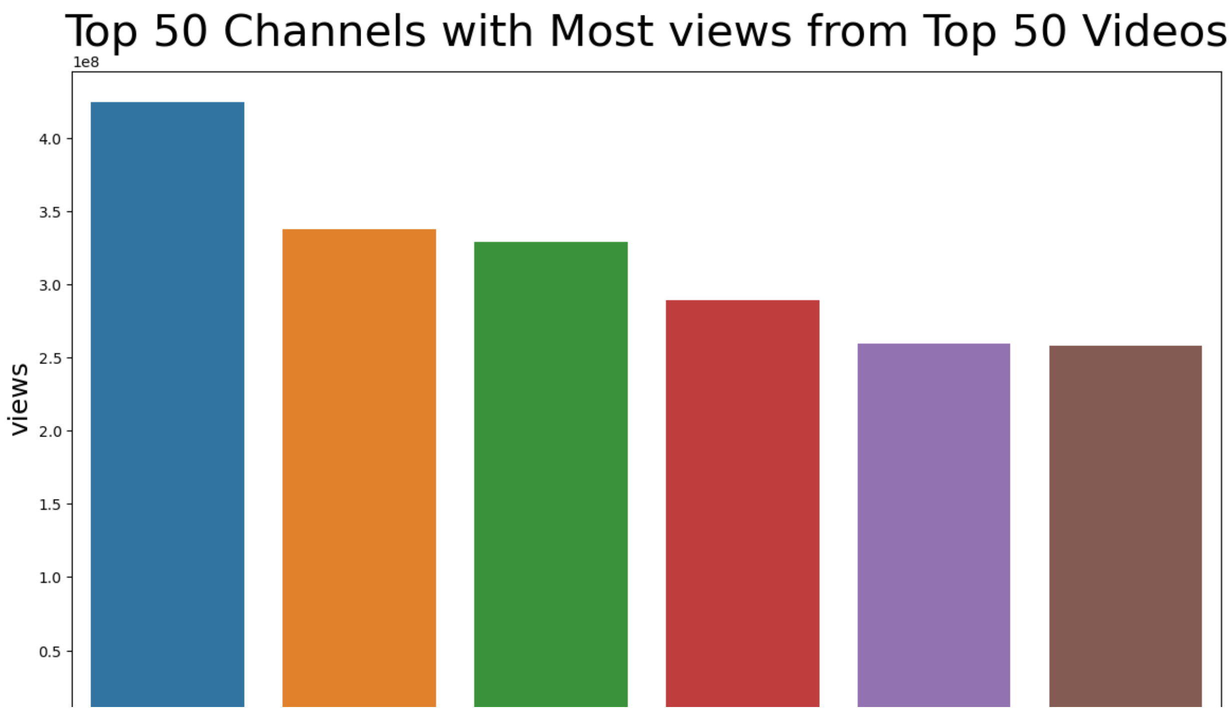
In [15]:
```
top_videos = merged_df.nlargest(10, 'views')[['title', 'views']]


top_videos.set_index('title')['views'].plot(kind='bar', figsize=(10, 6
plt.xlabel('Video Title',fontsize=20)
plt.ylabel('Views',fontsize=20)
plt.title('Top 10 Videos by Views',fontsize=30)
plt.xticks(rotation=45)
plt.show()
```



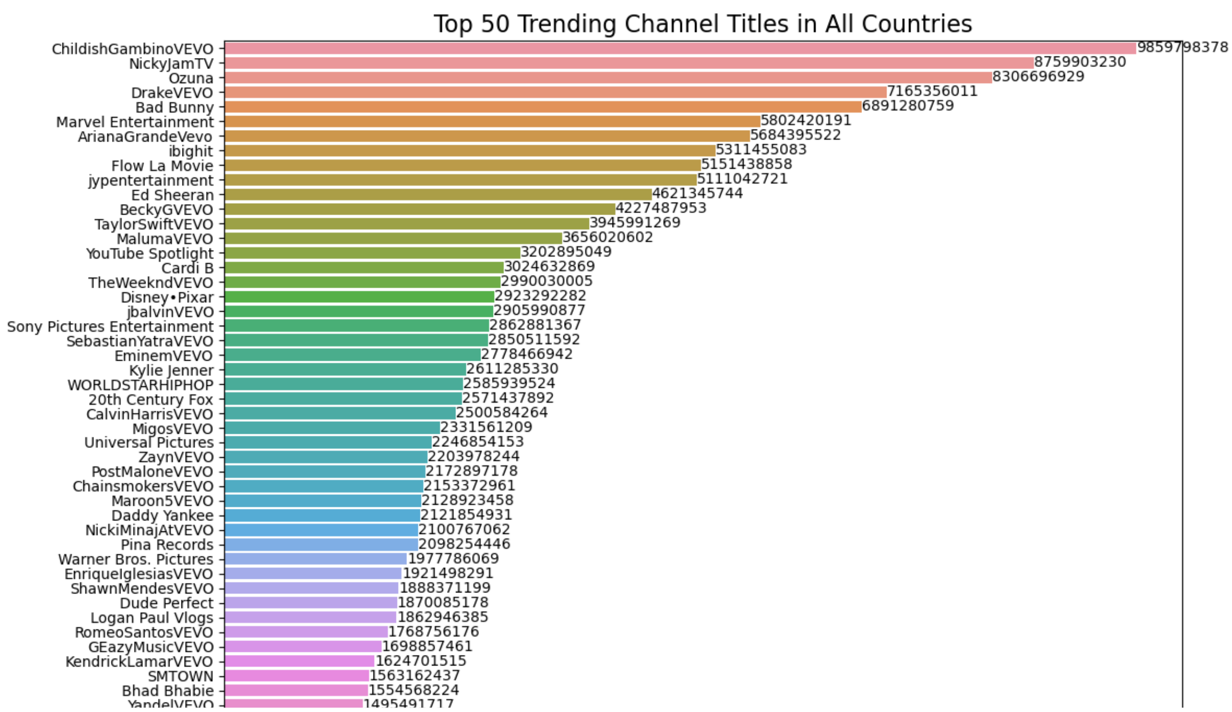
In [16]:
```
# Engagement metrics for top 50 videos
top_50_videos = merged_df.nlargest(50, 'views')
print(top_50_videos[['title', 'Engagement Metrics','location']])
```

```
                                                   title  Engagement
Metrics  \
28412  Nicky Jam x J. Balvin - X (EQUIS) | Video Ofic...
3067426
28212  Nicky Jam x J. Balvin - X (EQUIS) | Video Ofic...
3011515
28008  Nicky Jam x J. Balvin - X (EQUIS) | Video Ofic...
2956724
27811  Nicky Jam x J. Balvin - X (EQUIS) | Video Ofic...
2902891
27615  Nicky Jam x J. Balvin - X (EQUIS) | Video Ofic...
2845332
27424  Nicky Jam x J. Balvin - X (EQUIS) | Video Ofic...
2786627
27241  Nicky Jam x J. Balvin - X (EQUIS) | Video Ofic...
2723032
27052  Nicky Jam x J. Balvin - X (EQUIS) | Video Ofic...
2650114
26861  Nicky Jam x J. Balvin - X (EQUIS) | Video Ofic...
```

In [17]:
```
import seaborn as snb
content = top_50_videos.groupby('channel_title')['views'].max()

# Sort values to get the top 50 channels with the most views
content = content.sort_values(ascending=False).head(50)
content = content.reset_index()  # Convert index to column

# Plotting the results
plt.figure(figsize=(14, 8))
snb.barplot(x='channel_title', y='views', data=content)
plt.title('Top 50 Channels with Most views from Top 50 Videos', fontsi
plt.ylabel('views', fontsize=18)
plt.xlabel('Channel', fontsize=18)
plt.xticks(rotation=90)
plt.show()
```



In [18]:
```
channel_counts = merged_df.groupby('channel_title')['views'].sum().res

# Sort values and select top 10 channels
top_10_channels = channel_counts.sort_values(by='views', ascending=Fal

# Plot using seaborn
plt.figure(figsize=(12, 8))
ax = sns.barplot(x='views', y='channel_title', data=top_10_channels,or

# Add labels
for index, value in enumerate(top_10_channels['views']):
    ax.text(value, index, str(value), va='center', ha='left', color='b

# Customize the plot
plt.title('Top 50 Trending Channel Titles in All Countries', fontsize=
plt.xlabel('Views', fontsize=12)
plt.ylabel(None)
plt.xticks(rotation=0)  # x-axis ticks don't need rotation in horizont
plt.tight_layout()

# Add caption
plt.figtext(0.95, 0.02, "Donyoe", horizontalalignment='right', fontsiz

# Show the plot
plt.show()
```



Normalize and Standardize Data

Correlation Metrics for Variables

In [19]:
```
# add category_id to numerical columns
numerical_columns = ['views', 'likes', 'dislikes', 'comment_count', ]

# Compute the correlation matrix
correlation_matrix = merged_df[numerical_columns].corr()
# Display the correlation matrix
print(correlation_matrix)

# Plot the correlation matrix as a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f'
plt.title('Correlation Heatmap for Numerical Variables')
plt.show()
```

```
                  views     likes  dislikes  comment_count
views          1.000000  0.791670  0.405290       0.485986
likes          0.791670  1.000000  0.448010       0.763192
dislikes       0.405290  0.448010  1.000000       0.745064
comment_count  0.485986  0.763192  0.745064       1.000000
```

Assign Score for Numerical Values

In [20]:
```
import pandas as pd

# Assuming the correlation values are manually entered from the heatma
correlation_values = {
    'likes': 0.784,          # Correlation of likes with views
    'dislikes': 0.416,       # Correlation of dislikes with views
    'comment_count': 0.502 # Correlation of comment_count with views
}

# Convert the correlation values to absolute values
abs_correlations = {key: abs(value) for key, value in correlation_valu

# Calculate the total sum of absolute correlations
total_correlation = sum(abs_correlations.values())

# Calculate weights by normalizing the absolute correlation values
weights = {key: value / total_correlation for key, value in abs_correl

# Convert the weights to a DataFrame for better visualization
weights_df = pd.DataFrame(list(weights.items()), columns=['Variable',

# Display the weights
print("Calculated Weights of Independent Variables Relative to 'Views'
print(weights_df)
```

```
Calculated Weights of Independent Variables Relative to 'Views':
        Variable    Weight
0          likes  0.460635
1       dislikes  0.244418
2  comment_count  0.294947
```

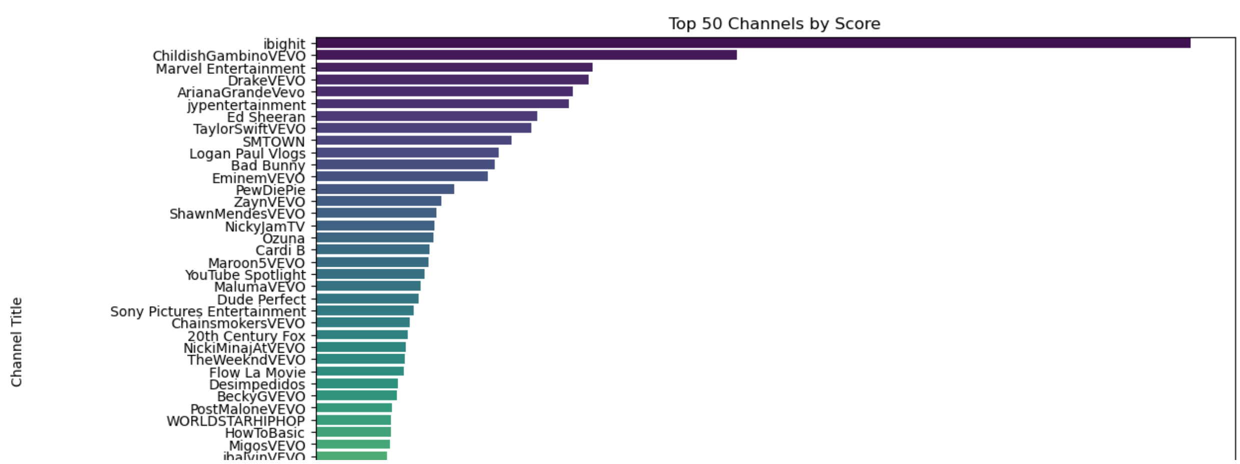
In [21]:
```
import pandas as pd

weights = {
    'likes': 0.460435,
    'dislikes': 0.244418,
    'comment_count': 0.294947
}

merged_df['score'] = (
    weights['likes'] * merged_df['likes'] -
    weights['dislikes'] * merged_df['dislikes'] +
    weights['comment_count'] * merged_df['comment_count']
)

merged_df['rank'] = merged_df['score'].rank(ascending=False, method='m

df_sorted = merged_df.sort_values(by='rank')

print(df_sorted)

#output_filename = 'ranked_videos_combined.csv'
#df_sorted.to_csv(output_filename, index=False)

#print("Listing of Every Video with Individual Scores and Ranks Across
#print(df_sorted[['video_id', 'views', 'likes', 'dislikes', 'comment_c
#print(f"\nThe ranking of all videos from all locations has been savec
```

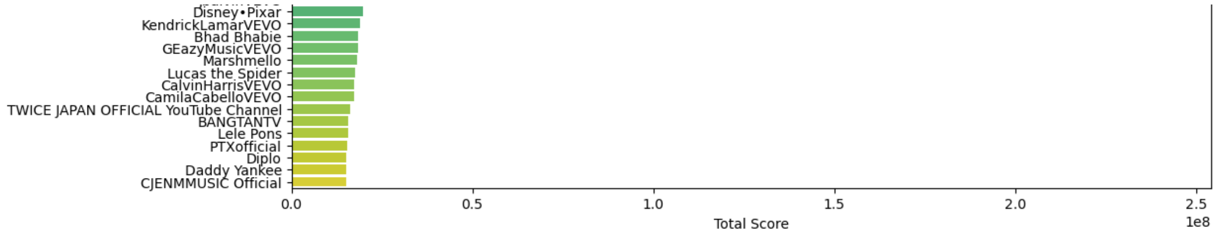
```
          video_id trending_date                                     tit
le  \
36638  7C2z4GqqS5E    2018-06-01    BTS (방탄소년단) 'FAKE LOVE' Offici
al MV
77189  7C2z4GqqS5E    2018-06-01    BTS (방탄소년단) 'FAKE LOVE' Offici
al MV
76988  7C2z4GqqS5E    2018-05-31    BTS (방탄소년단) 'FAKE LOVE' Offici
al MV
36468  7C2z4GqqS5E    2018-05-31    BTS (방탄소년단) 'FAKE LOVE' Offici
al MV
36288  7C2z4GqqS5E    2018-05-30    BTS (방탄소년단) 'FAKE LOVE' Offici
al MV
...            ...           ...
...
9146   LFhT6H6pRWg    2017-12-29   PSA from Chairman of the FCC Ajit P
ai
9354   LFhT6H6pRWg    2017-12-30   PSA from Chairman of the FCC Ajit P
ai
9575   LFhT6H6pRWg    2017-12-31   PSA from Chairman of the FCC Ajit P
```

EDA for Score for Top 50 Channels

In [22]:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming your DataFrame is named 'train'
weights = {
    'likes': 0.460435,
    'dislikes': 0.244418,
    'comment_count': 0.294947
}

# Calculate score and rank
merged_df['score'] = (
    weights['likes'] * merged_df['likes'] -
    weights['dislikes'] * merged_df['dislikes'] +
    weights['comment_count'] * merged_df['comment_count']
)

merged_df['rank'] = merged_df['score'].rank(ascending=False, method='m

# Group by channel_title and sum the scores
channel_scores = merged_df.groupby('channel_title')['score'].sum().res

# Sort by total score and get top 50 channels
top_channels = channel_scores.sort_values(by='score', ascending=False)

# Create a bar plot for the top 50 channels
plt.figure(figsize=(12, 8))
sns.barplot(x='score', y='channel_title', data=top_channels, palette='
plt.title('Top 50 Channels by Score')
plt.xlabel('Total Score')
plt.ylabel('Channel Title')
plt.show()
```





Create Word Cloud

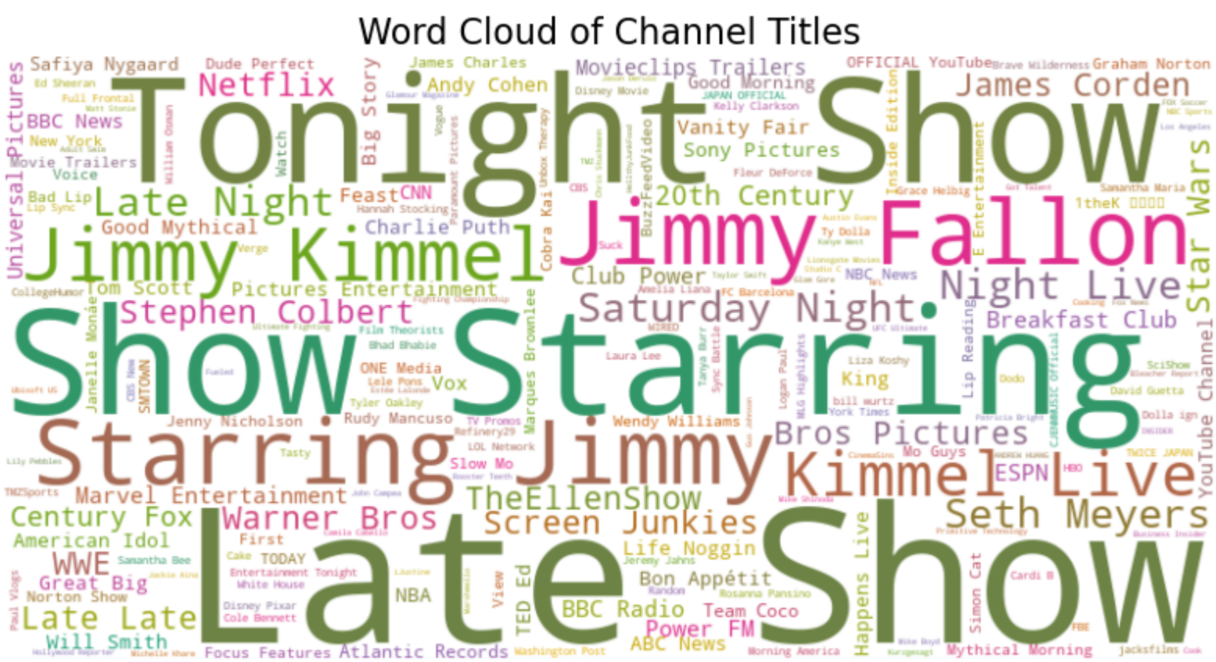
Video Titles

In [23]:
```
from wordcloud import WordCloud
from palettable.colorbrewer.qualitative import Dark2_6

# Assuming your DataFrame is named 'mergeda_df'
# Concatenate all titles into a single string
all_titles = " ".join(merged_df['title'].astype(str))

# Set up the color palette (equivalent to R's "Dark2")
cmap = Dark2_6.mpl_colormap

# Create a WordCloud object
wordcloud = WordCloud(
    background_color="white",
    max_words=200,
    colormap=cmap,
    width=800,
    height=400,
    random_state=42
)

# Generate the word cloud from the titles
wordcloud.generate(all_titles)

# Plot the word cloud
plt.figure(figsize=(10, 6))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")  # Turn off the axis
plt.title('Word Cloud of Video Titles', fontsize=16)
plt.show()
```

Word Cloud of Video Titles

Channel Titles

In [24]:
```python
all_channel_titles = " ".join(merged_df['channel_title'].astype(str))

# Set up the color palette (equivalent to R's "Dark2")
cmap = Dark2_6.mpl_colormap

# Create a WordCloud object
wordcloud = WordCloud(
    background_color="white",
    max_words=200,
    colormap=cmap,
    width=800,
    height=400,
    random_state=42
)

# Generate the word cloud from the titles
wordcloud.generate(all_channel_titles)

# Plot the word cloud
plt.figure(figsize=(10, 6))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")  # Turn off the axis
plt.title('Word Cloud of Channel Titles', fontsize=16)
plt.show()
```



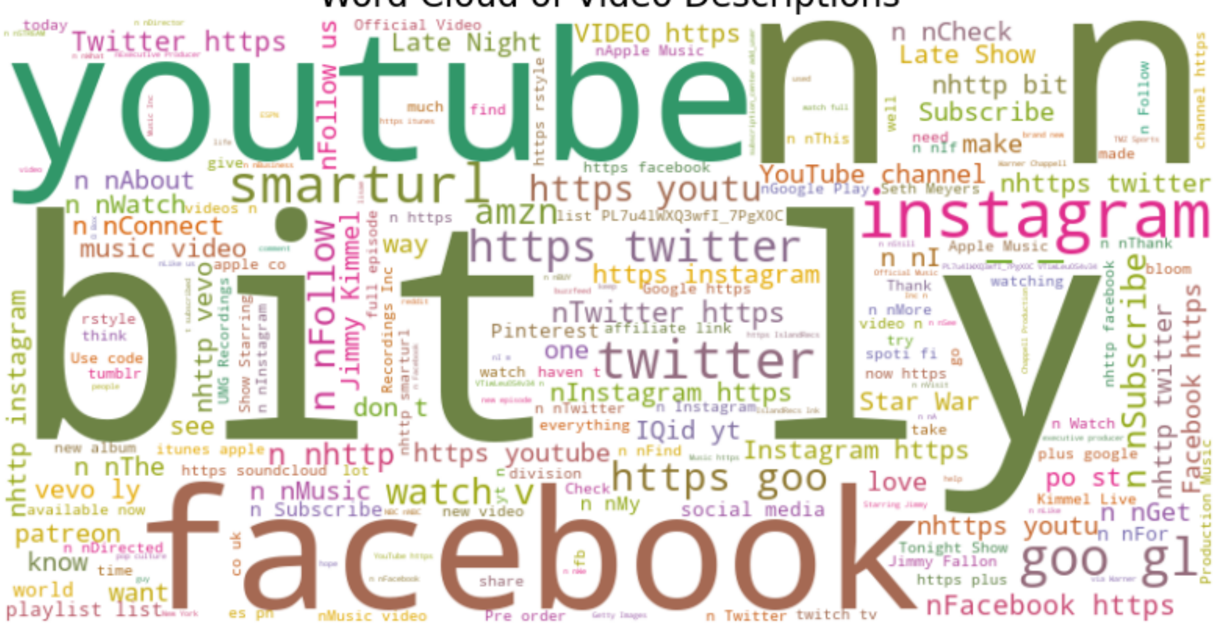
Video Tags

In [25]:
```
# Creating Word Cloud-tags
all_tags = " ".join(merged_df['tags'].astype(str))

# Set up the color palette (equivalent to R's "Dark2")
cmap = Dark2_6.mpl_colormap

# Create a WordCloud object
wordcloud = WordCloud(
    background_color="white",
    max_words=200,
    colormap=cmap,
    width=800,
    height=400,
    random_state=42
)

# Generate the word cloud from the titles
wordcloud.generate(all_tags)

# Plot the word cloud
plt.figure(figsize=(10, 6))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")  # Turn off the axis
plt.title('Word Cloud of Tags', fontsize=16)
plt.show()
```

Video Descriptions

In [26]:
```python
all_description = " ".join(merged_df['description'].astype(str))

# Set up the color palette (equivalent to R's "Dark2")
cmap = Dark2_6.mpl_colormap

# Create a WordCloud object
wordcloud = WordCloud(
    background_color="white",
    max_words=200,
    colormap=cmap,
    width=800,
    height=400,
    random_state=42
)

# Generate the word cloud from the titles
wordcloud.generate(all_description)

# Plot the word cloud
plt.figure(figsize=(10, 6))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")  # Turn off the axis
plt.title('Word Cloud of Video Descriptions', fontsize=16)
plt.show()
```

Word Cloud of Video Descriptions



Drop Unnecessary Columns

In [27]:
```
#drop columns needed
merged_df.drop(columns=['thumbnail_link', 'video_id','comments_disable

print(merged_df.head())
```

```
  trending_date                                              title  \
0    2017-11-14      John Lewis Christmas Ad 2017 - #MozTheMonster
1    2017-11-14          Taylor Swift: …Ready for It? (Live) - SNL
2    2017-11-14         Eminem - Walk On Water (Audio) ft. Beyoncé
3    2017-11-14  Goals from Salford City vs Class of 92 and Fri...
4    2017-11-14  Dashcam captures truck's near miss with child ...

                channel_title  category_id        publish_time  \
0                  John Lewis           26 2017-11-10 07:38:29
1         Saturday Night Live           24 2017-11-12 06:24:44
2                  EminemVEVO           10 2017-11-10 17:00:03
3  Salford City Football Club           17 2017-11-13 02:30:38
4            Cute Girl Videos           25 2017-11-13 01:45:13

                                                tags     views   like
s  \
0  christmas|"john lewis christmas"|"john lewis"|...   7224515   5568
1
1  SNL|"Saturday Night Live"|"SNL Season 43"|"Epi...   1053632   2556
```

Text Preprocessing

In [28]:
```
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
import re

# Get the list of default English stopwords
stop_words = set(stopwords.words('english'))

# Function to remove stopwords and clean text
def clean_text(text):
    # Lowercase the text
    text = text.lower()

    # Remove non-alphabetical characters (retain only letters and spac
    text = re.sub(r'[^a-z\s]', '', text)
```

```
    # Split text into words
    words = text.split()

    # Remove stopwords
    remove_stopwords = [word for word in words if word not in stop_wor

    # Join the cleaned words back into a string
    new_text = ' '.join(remove_stopwords)

    return new_text
    data = {'title','description','text'}

# Apply the clean_text function to the 'title' column in merged_df1
merged_df['new_text'] = merged_df['title'].apply(clean_text)

# Display the cleaned DataFrame
print(merged_df)
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/yuhanzhao/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!

      trending_date                                              titl
e  \
0        2017-11-14      John Lewis Christmas Ad 2017 - #MozTheMonste
r
1        2017-11-14          Taylor Swift: …Ready for It? (Live) - SN
L
2        2017-11-14         Eminem - Walk On Water (Audio) ft. Beyonc
é
3        2017-11-14  Goals from Salford City vs Class of 92 and Fr
i...
4        2017-11-14  Dashcam captures truck's near miss with child
...
...             ...
...
79860    2018-06-14                          The Cat Who Caught the Lase
r
```

In [29]:
```
# Check the data types of each column
print(merged_df.dtypes)
```

```
trending_date          datetime64[ns]
title                          object
channel_title                  object
category_id                     int64
publish_time           datetime64[ns]
tags                           object
views                           int64
likes                           int64
dislikes                        int64
comment_count                   int64
description                    object
location                       object
trending_day_of_week           object
day_of_week                    object
Engagement Metrics              int64
score                         float64
rank                          float64
new_text                       object
dtype: object
```

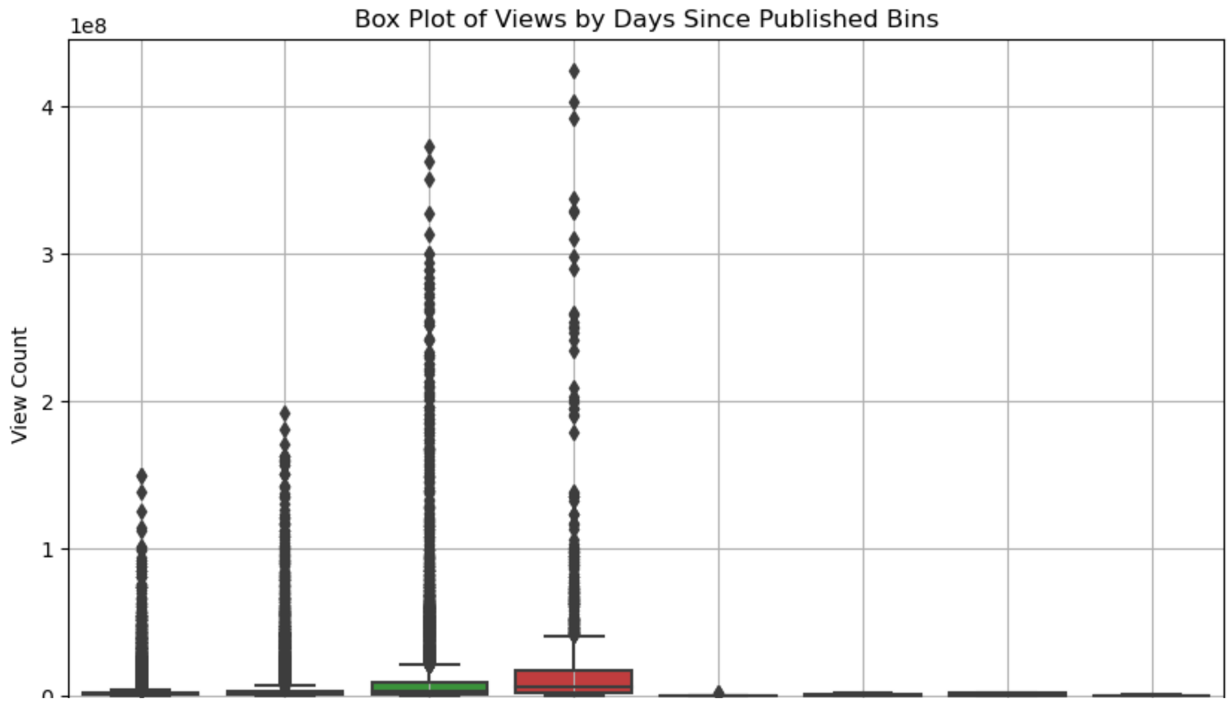
Split the Dataset into Train and Test by 80/20

In [30]:
```
from sklearn.model_selection import train_test_split

X = merged_df.drop(columns=['views'])  # Drop 'views' from features to
y = merged_df['views']
# Assuming you have a dataset with features X and target y
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2

train = pd.DataFrame(X_train)
train['views'] = y_train.values

test = pd.DataFrame(X_test)
test['views'] = y_test.values
```

Feature Engineering

Days Since Published

In [31]:
```
#convert the type of publish time
train['publish_time'] = pd.to_datetime(train['publish_time'])
train['trending_date'] = pd.to_datetime(train['trending_date'], format

# Creating a new feature 'days_since_published'
train['days_since_published'] = (train['trending_date'] - train['publi

# Creating bins for days since published
bins = [0, 7, 14, 30, 60, 90, 120, 180, 365]  # Example bins
labels = ['0-7', '8-14', '15-30', '31-60', '61-90', '91-120', '121-180
train['days_bins'] = pd.cut(train['days_since_published'], bins=bins,

plt.figure(figsize=(10, 6))
sns.boxplot(data=train, x='days_bins', y='views')
plt.title('Box Plot of Views by Days Since Published Bins')
plt.xlabel('Days Since Published Bins')
plt.ylabel('View Count')
plt.grid(True)
plt.show()
```



Sentimental Analysis

In [32]:
```
!pip install textblob
```

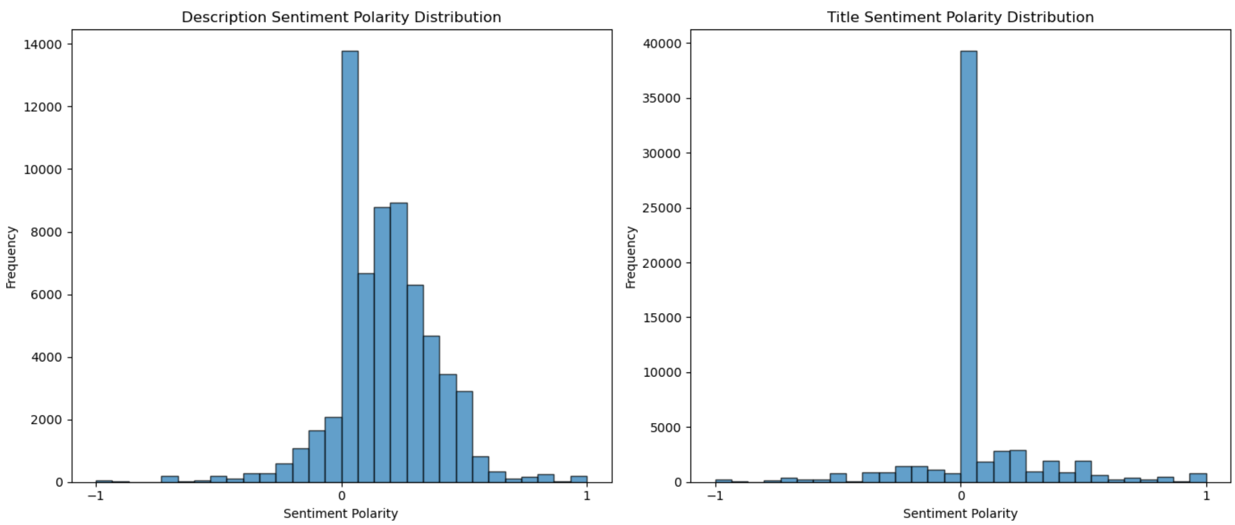
```
Requirement already satisfied: textblob in /Users/yuhanzhao/anaconda
3/lib/python3.11/site-packages (0.18.0.post0)
Requirement already satisfied: nltk>=3.8 in /Users/yuhanzhao/anaconda
3/lib/python3.11/site-packages (from textblob) (3.8.1)
Requirement already satisfied: click in /Users/yuhanzhao/anaconda3/li
b/python3.11/site-packages (from nltk>=3.8->textblob) (8.0.4)
Requirement already satisfied: joblib in /Users/yuhanzhao/anaconda3/l
ib/python3.11/site-packages (from nltk>=3.8->textblob) (1.2.0)
Requirement already satisfied: regex>=2021.8.3 in /Users/yuhanzhao/an
aconda3/lib/python3.11/site-packages (from nltk>=3.8->textblob) (202
2.7.9)
Requirement already satisfied: tqdm in /Users/yuhanzhao/anaconda3/li
b/python3.11/site-packages (from nltk>=3.8->textblob) (4.65.0)
```

Sentiment Polarity Distribution

In [33]:
```
from textblob import TextBlob
import matplotlib.pyplot as plt

# Calculate sentiment polarity for description and title
def get_sentiment(text):
    return TextBlob(text).sentiment.polarity

# Apply sentiment analysis
train['description_sentiment'] = train['description'].fillna('').apply
train['title_sentiment'] = train['title'].fillna('').apply(get_sentime

# Calculate average sentiment scores
avg_description_sentiment = train['description_sentiment'].mean()
avg_title_sentiment = train['title_sentiment'].mean()

print("Average Description Sentiment Score:", avg_description_sentimer
print("Average Title Sentiment Score:", avg_title_sentiment)

# Plotting the sentiment distributions
plt.figure(figsize=(14, 6))

# Description Sentiment Histogram
plt.subplot(1, 2, 1)
plt.hist(train['description_sentiment'], bins=30, alpha=0.7, edgecolor
plt.title('Description Sentiment Polarity Distribution')
plt.xlabel('Sentiment Polarity')
plt.ylabel('Frequency')
plt.xticks([-1, 0, 1])

# Title Sentiment Histogram
```

```
plt.subplot(1, 2, 2)
plt.hist(train['title_sentiment'], bins=30, alpha=0.7, edgecolor='blac
plt.title('Title Sentiment Polarity Distribution')
plt.xlabel('Sentiment Polarity')
plt.ylabel('Frequency')
plt.xticks([-1, 0, 1])

plt.tight_layout()
plt.show()
```

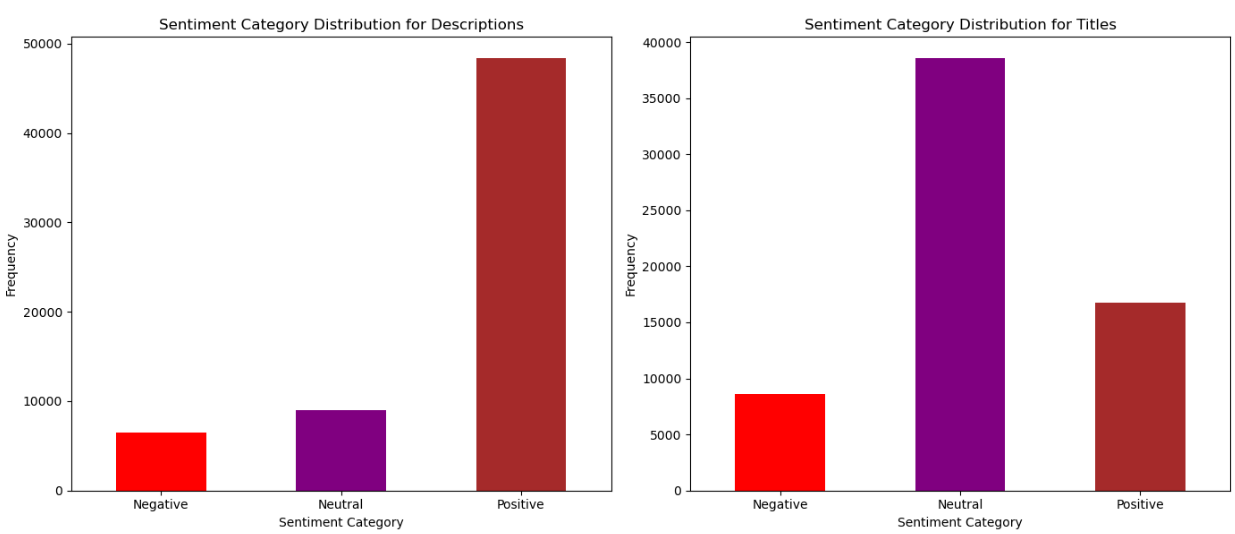
```
Average Description Sentiment Score: 0.1716764242965884
Average Title Sentiment Score: 0.0477964529239135
```



Visualize the Sentiment Distrubution Category

In [34]:
```
import pandas as pd
import matplotlib.pyplot as plt

# Define sentiment categories
def categorize_sentiment(polarity):
    if polarity > 0:
        return 'Positive'
    elif polarity < 0:
        return 'Negative'
    else:
        return 'Neutral'

# Apply categorization to sentiment columns
train['description_sentiment_category'] = train['description_sentiment
train['title_sentiment_category'] = train['title_sentiment'].apply(cat
```

```
# Plot sentiment category distribution for descriptions and titles
plt.figure(figsize=(14, 6))

# Custom order for categories
category_order = ['Negative', 'Neutral', 'Positive']

# Plot description sentiment distribution
plt.subplot(1, 2, 1)
description_sentiment_counts = train['description_sentiment_category']
description_sentiment_counts.plot(kind='bar', color=['red', 'purple',
plt.title('Sentiment Category Distribution for Descriptions')
plt.xlabel('Sentiment Category')
plt.ylabel('Frequency')
plt.xticks(rotation=0)

# Plot title sentiment distribution
plt.subplot(1, 2, 2)
title_sentiment_counts = train['title_sentiment_category'].value_count
title_sentiment_counts.plot(kind='bar', color=['red', 'purple', 'browr
plt.title('Sentiment Category Distribution for Titles')
plt.xlabel('Sentiment Category')
plt.ylabel('Frequency')
plt.xticks(rotation=0)

plt.tight_layout()
plt.show()
```



Create TF-IDF Feature

description Column

In [35]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
import numpy as np

# Ensure the 'description' column exists in the DataFrame
if 'description' in train.columns:
    # Assuming 'description' column contains the text data
    text_data = train['description'].fillna('')  # Handle missing valu

    # Check if text_data is iterable, not a single string
    if isinstance(text_data, pd.Series):
        # Initialize the TF-IDF Vectorizer
        tfidf_vectorizer = TfidfVectorizer(max_features=100, stop_word

        # Fit and transform the text data to generate the TF-IDF matri
        tfidf_matrix = tfidf_vectorizer.fit_transform(text_data)

        # Convert the sparse matrix into a DataFrame for easier manipu
        tfidf_df = pd.DataFrame(tfidf_matrix.toarray(), columns=tfidf_

        # Function to get top N features per row based on TF-IDF score
        def get_top_tfidf_features(row, features, top_n=5):
            top_indices = np.argsort(row)[::-1][:top_n]  # Get the ind
            top_features = [(features[i], row[i]) for i in top_indices
            return top_features

        # Apply the function to each row in the TF-IDF matrix
        top_tfidf_features = [get_top_tfidf_features(row, tfidf_vector
                              for row in tfidf_matrix.toarray()]

        # Add the top TF-IDF features as a new column in the original
        train['top_tfidf_features'] = top_tfidf_features

        # Display the entire first 5 rows of the DataFrame including t
        print(train.head(5))
    else:
        print("The 'description' column should be a pandas Series.")
else:
    print("The DataFrame does not contain a 'description' column.")
```

```
      trending_date                                               titl
e  \
23604    2018-03-14                        Marshmello & Anne-Marie: Friend
s
25630    2018-03-24  Kirby Star Allies' Surprising HD Rumble Secre
t...
68698    2018-04-20  Stephen A.: Kevin Hart 'got his feelings hur
```

```
t'...
39559    2017-11-17                              How to be an Aquariu
s
62877    2018-03-16  Charlie Puth - Done For Me (feat. Kehlani) [O
f...

                                    channel_title  category_id  \
23604  The Tonight Show Starring Jimmy Fallon           23
25630                                  GameXplain           20
68698                                        ESPN           17
39559                                    Sailor J           24
62877                                 Charlie Puth          10
```

tags Column

In [36]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
import numpy as np

# Ensure the 'description' column exists in the DataFrame
if 'tags' in train.columns:
    # Assuming 'description' column contains the text data
    text_data = train['tags'].fillna('')  # Handle missing values

    # Check if text_data is iterable, not a single string
    if isinstance(text_data, pd.Series):
        # Initialize the TF-IDF Vectorizer
        tfidf_vectorizer = TfidfVectorizer(max_features=100, stop_word

        # Fit and transform the text data to generate the TF-IDF matri
        tfidf_matrix = tfidf_vectorizer.fit_transform(text_data)

        # Convert the sparse matrix into a DataFrame for easier manipu
        tfidf_df = pd.DataFrame(tfidf_matrix.toarray(), columns=tfidf_

        # Function to get top N features per row based on TF-IDF score
        def get_top_tfidf_features(row, features, top_n=5):
            top_indices = np.argsort(row)[::-1][:top_n]  # Get the inc
            top_features = [(features[i], row[i]) for i in top_indices
            return top_features

        # Apply the function to each row in the TF-IDF matrix
        top_tfidf_features = [get_top_tfidf_features(row, tfidf_vector
                              for row in tfidf_matrix.toarray()]

        # Add the top TF-IDF features as a new column in the original
        train['top_tfidf_features'] = top_tfidf_features

        # Display the entire first 5 rows of the DataFrame including t
        print(train.head(5))
```

```
    else:
        print("The 'description' column should be a pandas Series.")
else:
    print("The DataFrame does not contain a 'description' column.")
```

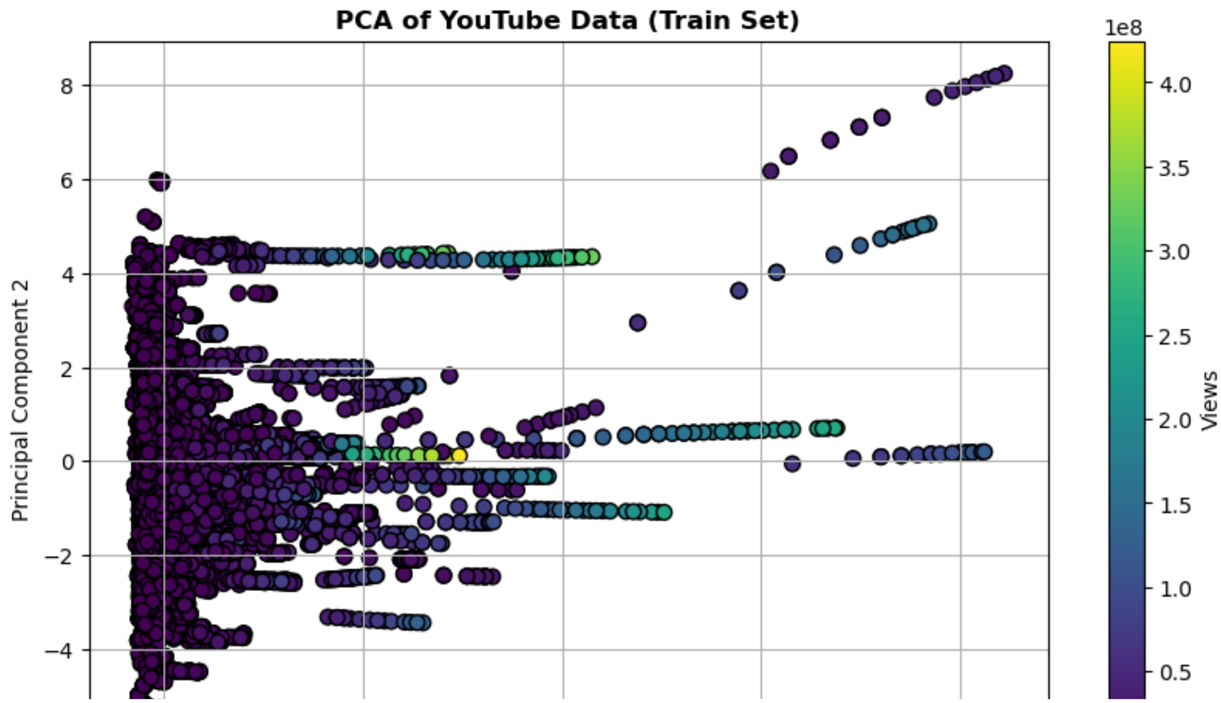
```
      trending_date                                               titl
e  \
23604    2018-03-14                        Marshmello & Anne-Marie: Friend
s
25630    2018-03-24  Kirby Star Allies' Surprising HD Rumble Secre
t...
68698    2018-04-20  Stephen A.: Kevin Hart 'got his feelings hur
t'...
39559    2017-11-17                                    How to be an Aquariu
s
62877    2018-03-16  Charlie Puth - Done For Me (feat. Kehlani) [O
f...

                                  channel_title  category_id  \
23604  The Tonight Show Starring Jimmy Fallon           23
25630                              GameXplain           20
68698                                    ESPN           17
39559                                Sailor J           24
62877                            Charlie Puth           10
```

Dimension Reduction-PCA

In [37]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

non_numeric_cols = ['publish_time', 'title', 'channel_title', 'tags',
X_train_model = train.drop(columns=non_numeric_cols + ['views']).selec
X_test_model = test.drop(columns=non_numeric_cols + ['views']).select_

X_test_model = X_test_model.reindex(columns=X_train_model.columns, fil

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_model)
X_test_scaled = scaler.transform(X_test_model)

print("Missing values in X_train_model:\n", X_train_model.isna().sum()
print("Missing values in X_test_model:\n", X_test_model.isna().sum())
```

```
Missing values in X_train_model:
 category_id             0
likes                   0
dislikes                0
comment_count           0
Engagement Metrics      0
score                   0
rank                    0
days_since_published    0
description_sentiment   0
title_sentiment         0
dtype: int64
Missing values in X_test_model:
 category_id             0
likes                   0
dislikes                0
comment_count           0
Engagement Metrics      0
score                   0
```

In [38]:
```
# Apply PCA (Reduce to n components to capture 95% of variance)
pca = PCA(n_components=0.95)
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)

# Visualize the PCA results (Plot only the first two components)
plt.figure(figsize=(10, 6))
plt.scatter(X_train_pca[:, 0], X_train_pca[:, 1], c=y_train, cmap='vir
plt.colorbar(label='Views')
plt.title('PCA of YouTube Data (Train Set)', weight='bold')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.grid(True)
plt.show()

# Explained variance for all components selected by PCA
explained_variance = pca.explained_variance_ratio_
print("Explained Variance per component:")
for i, variance in enumerate(explained_variance, start=1):
    print(f"PC{i}: {variance:.2%}")
```



Model Building

First Model-XGBoost

In [39]:
```
pip install xgboost
```

```
Requirement already satisfied: xgboost in /Users/yuhanzhao/anaconda3/
lib/python3.11/site-packages (2.0.3)
Requirement already satisfied: numpy in /Users/yuhanzhao/anaconda3/li
b/python3.11/site-packages (from xgboost) (1.24.3)
Requirement already satisfied: scipy in /Users/yuhanzhao/anaconda3/li
b/python3.11/site-packages (from xgboost) (1.11.1)
Note: you may need to restart the kernel to use updated packages.
```

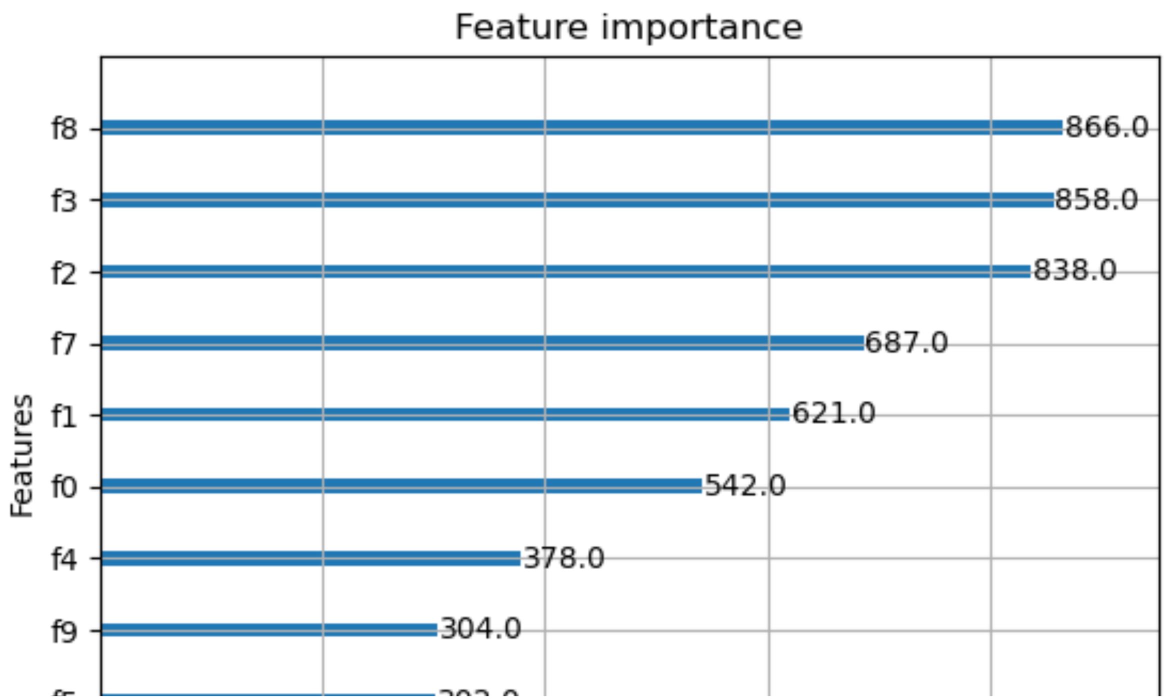
In [40]:
```
import xgboost as xgb
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

dtrain = xgb.DMatrix(X_train_scaled, label=y_train)
dtest = xgb.DMatrix(X_test_scaled, label=y_test)
```

In [41]:
```
# Parameters (basic setup, tune based on results)
params = {
    'objective': 'reg:squarederror',
    'max_depth': 6,
    'eta': 0.1,
    'subsample': 0.8,
    'colsample_bytree': 0.8,
    'eval_metric': 'rmse'
}

# Train the model
num_round = 100
bst = xgb.train(params, dtrain, num_round)
```

In [42]:
```
# Predictions
y_pred = bst.predict(dtest)

# Evaluation
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)

print(f"RMSE: {rmse}")
print(f"R^2 Score: {r2}")
```

```
RMSE: 7953991.516667129
R^2 Score: 0.7069162313831201
```

In [43]:
```
# If trained using xgb.train, plot importance
xgb.plot_importance(bst, max_num_features=10)
plt.show()
```



Another Approach-XGBoost

The model approach chosen for this week is XGBoost. XGBoost is a gradient boosting algorithm widely used for structured/tabular data, especially when aiming to capture complex interactions between variables. It's particularly suited to this task as it efficiently handles large datasets, offers built-in regularization to prevent overfitting, and allows flexibility in tuning various hyperparameters for optimization. This makes XGBoost an effective choice for regression tasks on datasets like YouTube metrics, where sentiment analysis and engagement metrics play significant roles in predicting user interactions.

XGBoost is a relatively complex modeling approach compared to simpler algorithms like linear regression. It builds an ensemble of decision trees in a sequential manner, where each tree aims to correct the errors made by the previous ones. This iterative correction improves predictive power but requires balancing complexity and training time, especially with deep trees and large numbers of boosting rounds. Additionally, XGBoost uses gradient-based methods to minimize loss and includes hyperparameters like learning rate, tree depth, and column sampling to control overfitting, adding layers of complexity in tuning.

In [44]:
```
import xgboost as xgb
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np
import pandas as pd
```

In [45]:
```
# Define a function for calculating model metrics
def calculate_metrics(model, X_train, y_train, X_test, y_test):
    train_preds = model.predict(X_train)
    test_preds = model.predict(X_test)

    # Calculate RMSE and R^2 for training and test sets
    train_rmse = np.sqrt(mean_squared_error(y_train, train_preds))
    test_rmse = np.sqrt(mean_squared_error(y_test, test_preds))
    train_r2 = r2_score(y_train, train_preds)
    test_r2 = r2_score(y_test, test_preds)

    return {
        "Train RMSE": train_rmse, "Test RMSE": val_rmse,
        "Train R^2": train_r2, "Test R^2": val_r2
    }
```

In [46]:
```
# Define a function to train the model with specific hyperparameters
def train_xgboost(X_train, y_train, X_test, y_test, params):
    model = xgb.XGBRegressor(**params, random_state=42)
    model.fit(X_train, y_train)

    # Calculate and return metrics
    metrics = calculate_metrics(model, X_train, y_train, X_test, y_tes
    return model, metrics
```

In [47]:
```
# Define hyperparameter variations
variations = [
    {"learning_rate": 0.1, "n_estimators": 100, "max_depth": 4},
    {"learning_rate": 0.05, "n_estimators": 200, "max_depth": 6},
    {"learning_rate": 0.01, "n_estimators": 300, "max_depth": 8}
]
```

In [48]:
```
# Initialize a DataFrame to store results for each variation
results = pd.DataFrame(columns=["Variation", "Train RMSE", "Test RMSE"
```

For this model, the key hyperparameters evaluated are:

learning_rate: Controls the contribution of each tree to the model. Lower values make the model learn more slowly and can improve performance but may require more boosting rounds. Learning rates of 0.1, 0.05, and 0.01 are tested to observe their effects on stability and accuracy. n_estimators: Defines the number of boosting rounds. More rounds often lead to better accuracy but increase the risk of overfitting, so we use values of 100, 200, and 300. max_depth: Controls the complexity of each individual tree. A deeper tree captures more detail but can overfit, so depths of 4, 6, and 8 are explored to assess the optimal complexity for balancing performance with generalization. Performance Metrics The chosen metrics are Root Mean Square Error (RMSE) and R^2:

RMSE: Measures the average magnitude of error between predicted and actual values, making it appropriate to understand the model's prediction accuracy in the same units as the target variable. R^2 Score: Shows the proportion of variance in the target variable that is predictable from the features. It indicates the model's ability to capture the data's overall trend and complexity, offering insight into the model's fit quality.

In [49]:
```
# Create a list of columns to drop if they exist
text_columns = ['title', 'channel_title', 'tags', 'description', 'loca
X_train.drop([col for col in text_columns if col in X_train.columns], 
X_test.drop([col for col in text_columns if col in X_test.columns], ax

# Encode categorical features using one-hot encoding for consistency
categorical_columns = ['trending_day_of_week', 'day_of_week', 'days_bi
X_train = pd.get_dummies(X_train, columns=[col for col in categorical_
X_test = pd.get_dummies(X_test, columns=[col for col in categorical_co

# Convert datetime columns to relevant features if they exist
if 'trending_date' in X_train.columns:
    X_train['trending_year'] = X_train['trending_date'].dt.year
    X_train['trending_month'] = X_train['trending_date'].dt.month
    X_train['trending_day'] = X_train['trending_date'].dt.day
    X_train.drop(['trending_date'], axis=1, inplace=True)

if 'trending_date' in X_test.columns:
    X_test['trending_year'] = X_test['trending_date'].dt.year
    X_test['trending_month'] = X_test['trending_date'].dt.month
    X_test['trending_day'] = X_test['trending_date'].dt.day
    X_test.drop(['trending_date'], axis=1, inplace=True)

# Drop 'publish_time' if it exists
if 'publish_time' in X_train.columns:
    X_train.drop(['publish_time'], axis=1, inplace=True)

if 'publish_time' in X_test.columns:
```

```
    X_test.drop(['publish_time'], axis=1, inplace=True)

# Ensure X_test has the same columns as X_train
X_test = X_test.reindex(columns=X_train.columns, fill_value=0)

# Verify that X_train and X_test now have the same columns
print("X_train columns:", X_train.columns)
print("X_test columns:", X_test.columns)
```

```
X_train columns: Index(['category_id', 'likes', 'dislikes', 'comment_
count',
       'Engagement Metrics', 'score', 'rank', 'trending_day_of_week_M
onday',
       'trending_day_of_week_Saturday', 'trending_day_of_week_Sunda
y',
       'trending_day_of_week_Thursday', 'trending_day_of_week_Tuesda
y',
       'trending_day_of_week_Wednesday', 'day_of_week_Monday',
       'day_of_week_Saturday', 'day_of_week_Sunday', 'day_of_week_Thu
rsday',
       'day_of_week_Tuesday', 'day_of_week_Wednesday', 'trending_yea
r',
       'trending_month', 'trending_day'],
      dtype='object')
X_test columns: Index(['category_id', 'likes', 'dislikes', 'comment_c
ount',
       'Engagement Metrics', 'score', 'rank', 'trending_day_of_week_M
onday',
       'trending_day_of_week_Saturday', 'trending_day_of_week_Sunda
y',
       'trending_day_of_week_Thursday', 'trending_day_of_week_Tuesda
y',
       'trending_day_of_week_Wednesday', 'day_of_week_Monday',
       'day_of_week_Saturday', 'day_of_week_Sunday', 'day_of_week_Thu
rsday',
       'day_of_week_Tuesday', 'day_of_week_Wednesday', 'trending_yea
r',
       'trending_month', 'trending_day'],
      dtype='object')
```

Variation 1: Lower max_depth and learning_rate may result in underfitting, but potentially more stable training and validation RMSE values.

Variation 2: Moderate depth and learning rate should balance performance, likely yielding lower validation RMSE and high R^2 without significant overfitting.

Variation 3: High max_depth and n_estimators values increase complexity, which may improve training accuracy but also risk overfitting, especially if validation RMSE increases.

In [50]:
```
def calculate_metrics(model, X_train, y_train, X_test, y_test):
    # Predictions
    train_preds = model.predict(X_train)
    test_preds = model.predict(X_test)

    # Calculate metrics
    train_rmse = np.sqrt(mean_squared_error(y_train, train_preds))
    test_rmse = np.sqrt(mean_squared_error(y_test, test_preds))

    train_r2 = r2_score(y_train, train_preds)
    test_r2 = r2_score(y_test, test_preds)

    return {
        "Train RMSE": train_rmse,
        "Test RMSE": test_rmse,  # Changed from val_rmse to test_rmse
        "Train R^2": train_r2,
        "Test R^2": test_r2
    }
```

The best model is identified by the lowest Test RMSE value across the variations, as minimizing prediction error on unseen data is crucial. Additionally, high Test R^2 scores indicate good predictive power. The best model balances these metrics, showcasing both accuracy and generalization to new data.

In [51]:
```
# Create an empty DataFrame if it isn't already
results = pd.DataFrame()

# Train models for each variation and record results
for i, params in enumerate(variations):
    model, metrics = train_xgboost(X_train, y_train, X_test, y_test, p

    # Create a DataFrame with the metrics for this variation
    result_row = pd.DataFrame({
        "Variation": [f"Variation {i + 1}"],
        **metrics
    })

    # Concatenate the new row to the results DataFrame
    results = pd.concat([results, result_row], ignore_index=True)
```

In [52]:
```python
# Display the comparison table
print("Comparison of XGBoost Model Variations:")
print(results)

# Identify the best model based on Validation RMSE
best_model_index = results["Test RMSE"].idxmin()
best_params = variations[best_model_index]
print(f"\nBest Model Variation: {best_model_index + 1}")
print(f"Hyperparameters: {best_params}")
print(results.iloc[best_model_index])
```

```
Comparison of XGBoost Model Variations:
     Variation    Train RMSE     Test RMSE  Train R^2  Test R^2
0  Variation 1  3.425430e+06  4.007967e+06   0.942740  0.925583
1  Variation 2  2.120971e+06  2.909246e+06   0.978047  0.960791
2  Variation 3  2.248698e+06  3.123748e+06   0.975323  0.954796

Best Model Variation: 2
Hyperparameters: {'learning_rate': 0.05, 'n_estimators': 200, 'max_de
pth': 6}
Variation       Variation 2
Train RMSE   2120971.173631
Test RMSE    2909245.576456
Train R^2          0.978047
Test R^2           0.960791
Name: 1, dtype: object
```

The table displays the training and test RMSE (Root Mean Squared Error) and R^2 (Coefficient of Determination) scores for three variations of XGBoost models:

RMSE: Variation 2 has the lowest Test RMSE (2.909 million), indicating it makes the most accurate predictions on the test data. RMSE measures the average prediction error in the same units as the target variable, so a lower RMSE reflects better accuracy. R^2: Variation 2 has a Test R^2 of 0.9608, meaning it explains around 96% of the variance in the test data. This is the highest R^2 score among the variations, indicating good predictive power. Model Complexity and Generalization:

Variations 1 and 3 both have lower R^2 scores and higher Test RMSEs, suggesting they may be underfitting or overfitting slightly compared to Variation 2. Variation 2 strikes a balance between model complexity and performance, achieving a good fit on both training and test data, without significant overfitting. Hyperparameters for Best Model (Variation 2) with the combination of learning_rate: 0.05 (slower learning, allowing the model to generalize better) n_estimators: 200 (enough trees to learn patterns without overfitting) max_depth: 6 (moderate depth to balance complexity and generalization)

RMSE helps measure prediction accuracy, crucial for comparing predicted vs. actual video engagement metrics. Lower RMSE means the model is better at predicting actual engagement values. R^2 assesses how well the model explains the variance in engagement metrics, giving insight into how much of the audience's behavior of the model can account for based on the features used.

By identifying Variation 2 as the best model, this model can support further research, such as investigating which specific sentiment or engagement patterns are associated with higher view counts or interaction, helping understand what makes certain videos more engaging than others.

First Model-Random Forest Regressor

In [53]:
```
print(X_train.dtypes)
```

```
category_id                        int64
likes                              int64
dislikes                           int64
comment_count                      int64
Engagement Metrics                 int64
score                            float64
rank                             float64
trending_day_of_week_Monday         bool
trending_day_of_week_Saturday       bool
trending_day_of_week_Sunday         bool
trending_day_of_week_Thursday       bool
trending_day_of_week_Tuesday        bool
trending_day_of_week_Wednesday      bool
day_of_week_Monday                  bool
day_of_week_Saturday                bool
day_of_week_Sunday                  bool
day_of_week_Thursday                bool
day_of_week_Tuesday                 bool
day_of_week_Wednesday               bool
```

Data Pre-processing

In [54]:
```
# Find common features between training and testing sets
common_features = list(set(X_train.columns) & set(X_test.columns))
print("Common features:", common_features)
```

```
Common features: ['Engagement Metrics', 'trending_month', 'dislikes',
'day_of_week_Thursday', 'trending_day_of_week_Tuesday', 'score', 'tre
nding_day_of_week_Monday', 'day_of_week_Saturday', 'likes', 'trending
_day_of_week_Thursday', 'category_id', 'day_of_week_Monday', 'trendin
g_day', 'trending_day_of_week_Saturday', 'comment_count', 'trending_y
ear', 'trending_day_of_week_Sunday', 'rank', 'day_of_week_Tuesday', '
day_of_week_Wednesday', 'day_of_week_Sunday', 'trending_day_of_week_W
ednesday']
```

In [55]:
```
# Create the Target Variable in Both DataFrames
train['days_to_trend'] = (train['trending_date'] - train['publish_time

# Assuming you want to create a similar target in the test set
# Create a new feature 'days_to_trend' in the test set if it has trend
test['days_to_trend'] = (test['trending_date'] - test['publish_time'])

# Define the target variable y for training and testing
y_train = train['days_to_trend']
y_test = test['days_to_trend']  # Make sure this column exists

# Step 2: Select Relevant Features
# Select the common features for the training and testing datasets
selected_features = ['likes', 'comment_count', 'trending_day_of_week',
X_train_selected = train[selected_features]
X_test_selected = test[selected_features]  # Ensure 'test' has the sam

# Step 3: Preprocess Features
# One-Hot Encoding for the categorical variable 'trending_day_of_week'
X_train_encoded = pd.get_dummies(X_train_selected, drop_first=True)
X_test_encoded = pd.get_dummies(X_test_selected, drop_first=True)
```

3 Variations Setting

In [56]:

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Variation 1: Default settings
rf_model1 = RandomForestRegressor(random_state=42)
rf_model1.fit(X_train_encoded, y_train)

# Variation 2: Increased number of trees
rf_model2 = RandomForestRegressor(n_estimators=200, random_state=42)
rf_model2.fit(X_train_encoded, y_train)

# Variation 3: Increased depth of trees
rf_model3 = RandomForestRegressor(max_depth=10, random_state=42)
rf_model3.fit(X_train_encoded, y_train)
```

Out[56]:

```
RandomForestRegressor(max_depth=10, random_state=42)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [57]:
```
# Function to calculate metrics
def evaluate_model(model, X_train, y_train, X_test, y_test):
    y_train_pred = model.predict(X_train)
    y_test_pred = model.predict(X_test)

    train_mse = mean_squared_error(y_train, y_train_pred)
    test_mse = mean_squared_error(y_test, y_test_pred)

    train_r2 = r2_score(y_train, y_train_pred)
    test_r2 = r2_score(y_test, y_test_pred)

    return {
        "train_mse": train_mse,
        "test_mse": test_mse,
        "train_r2": train_r2,
        "test_r2": test_r2,
    }

# Evaluate all models
result_model1 = evaluate_model(rf_model1, X_train_encoded, y_train, X_
result_model2 = evaluate_model(rf_model2, X_train_encoded, y_train, X_
result_model3 = evaluate_model(rf_model3, X_train_encoded, y_train, X_

# Print results for each model
print("Model 1 (Default settings):", result_model1)
print("Model 2 (n_estimators=200):", result_model2)
print("Model 3 (max_depth=10):", result_model3)
```

```
Model 1 (Default settings): {'train_mse': 1304.8170948196043, 'test_m
se': 7429.233070650234, 'train_r2': 0.9670432435550833, 'test_r2': 0.
8087028678543599}
Model 2 (n_estimators=200): {'train_mse': 1284.1807777766226, 'test_m
se': 7310.769158069089, 'train_r2': 0.9675644706890668, 'test_r2': 0.
8117532239979909}
Model 3 (max_depth=10): {'train_mse': 4769.834525959002, 'test_mse':
9669.919422254534, 'train_r2': 0.8795246664235963, 'test_r2': 0.75100
68891411407}
```

Models Comparison

In [58]:
```
# Create a DataFrame to compare metrics across models
performance_comparison = pd.DataFrame({
    'Model': ['Model 1 (Default)', 'Model 2 (n_estimators=200)', 'Mode
    'Train MSE': [result_model1['train_mse'], result_model2['train_mse
    'Test MSE': [result_model1['test_mse'], result_model2['test_mse'],
    'Train R²': [result_model1['train_r2'], result_model2['train_r2'],
    'Test R²': [result_model1['test_r2'], result_model2['test_r2'], re
})

# Display the performance comparison table
print("\nPerformance Comparison of Models:")
print(performance_comparison)
```

```
Performance Comparison of Models:
                        Model    Train MSE     Test MSE  Train R²   T
est R²
0           Model 1 (Default)  1304.817095  7429.233071  0.967043  0.
808703
1  Model 2 (n_estimators=200)  1284.180778  7310.769158  0.967564  0.
811753
2      Model 3 (max_depth=10)  4769.834526  9669.919422  0.879525  0.
751007
```

In [59]:
```
# Identify the winning model based on Test MSE and Test R²
winning_model = performance_comparison.loc[
    (performance_comparison['Test MSE'] == performance_comparison['Tes
    (performance_comparison['Test R²'] == performance_comparison['Test
]

# Display
print("\nWinning Model:")
print(winning_model)
```

```
Winning Model:
                        Model    Train MSE     Test MSE  Train R²   T
est R²
1  Model 2 (n_estimators=200)  1284.180778  7310.769158  0.967564  0.
811753
```

First Model-SVM Model

In [60]:
```
# from sklearn.svm import SVR
# from sklearn.metrics import mean_squared_error, r2_score
# import numpy as np

# # Create and train the SVM model
# svm_model = SVR(kernel='rbf', C=1.0, epsilon=0.1)
# svm_model.fit(X_train_scaled, y_train)

# # Predictions
# y_pred = svm_model.predict(X_test_scaled)

# # Evaluation
# rmse = np.sqrt(mean_squared_error(y_test, y_pred))
# r2 = r2_score(y_test, y_pred)

# print(f"RMSE: {rmse}")
# print(f"R^2 Score: {r2}")
```

In [61]:
```
# # Function to evaluate and return metrics
# def evaluate_model(model, X_train, y_train, X_test, y_test):
#     # Train the model
#     model.fit(X_train, y_train)

#     # Predictions on training and validation datasets
#     y_train_pred = model.predict(X_train)
#     y_test_pred = model.predict(X_test)

#     # Calculate RMSE and R^2 for training set
#     rmse_train = np.sqrt(mean_squared_error(y_train, y_train_pred))
#     r2_train = r2_score(y_train, y_train_pred)

#     # Calculate RMSE and R^2 for validation set
#     rmse_test = np.sqrt(mean_squared_error(y_test, y_test_pred))
#     r2_test = r2_score(y_test, y_test_pred)

#     return rmse_train, r2_train, rmse_test, r2_test
```

In [62]:
```
# # Define SVM variations with linear kernel for faster training
# svm_variations = {
#     'Variation 1': SVR(kernel='linear', C=1.0, epsilon=0.1),
#     'Variation 2': SVR(kernel='linear', C=10.0, epsilon=0.2),
#     'Variation 3': SVR(kernel='linear', C=1.0, epsilon=0.1)
# }

# # Initialize list to store results
# results = []
```

In [63]:
```
# # Loop through each variation, evaluate, and store the metrics
# for name, model in svm_variations.items():
#     rmse_train, r2_train, rmse_test, r2_test = evaluate_model(model,

#     # Append results for each model
#     results.append({
#         'Model': name,
#         'RMSE (Train)': rmse_train,
#         'R² (Train)': r2_train,
#         'RMSE (Validation)': rmse_test,
#         'R² (Validation)': r2_test
#     })

#     # Print some results for each model
#     print(f"\nModel: {name}")
#     print(f"RMSE (Train): {rmse_train:.4f}, R² (Train): {r2_train:.4
#     print(f"RMSE (Validation): {rmse_test:.4f}, R² (Validation): {r2

# results_df = pd.DataFrame(results)

# # Display the table of results
# print("\nSummary of all variations:")
# print(results_df)
```

Second Model-Convolutional Neural Network(CNN)

hyperparameter settings

In [64]:
```
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
```

```
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import mean_squared_error, r2_score

# scale the X_train and X_test
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_encoded)
X_test_scaled = scaler.transform(X_test_encoded)

# Define a function to build and train the neural network with differe
def build_and_train_model(hidden_layers, neurons, learning_rate, batch
    # Build the model
    model = Sequential()
    model.add(Dense(neurons, input_dim=X_train_scaled.shape[1], activa

    # Add hidden layers
    for _ in range(hidden_layers - 1):
        model.add(Dense(neurons, activation='relu'))

    # Output layer
    model.add(Dense(1, activation='linear'))

    # Compile the model
    optimizer = Adam(learning_rate=learning_rate)
    model.compile(optimizer=optimizer, loss='mean_squared_error', metr

    # Train the model
    model.fit(X_train_scaled, y_train, epochs=50, batch_size=batch_siz

    return model

# Define the hyperparameter variations
variations = [
    {"hidden_layers": 1, "neurons": 32, "learning_rate": 0.001, "batch
    {"hidden_layers": 2, "neurons": 64, "learning_rate": 0.001, "batch
    {"hidden_layers": 3, "neurons": 128, "learning_rate": 0.0001, "bat
]

# Store results for comparison
results = []

for i, params in enumerate(variations):
    print(f"Training Model Variation {i + 1} with params: {params}")
    model = build_and_train_model(**params)

    # Evaluate the model on training and testing data
    y_train_pred = model.predict(X_train_scaled)
    y_test_pred = model.predict(X_test_scaled)

    train_mse = mean_squared_error(y_train, y_train_pred)
```

```
    test_mse = mean_squared_error(y_test, y_test_pred)
    train_r2 = r2_score(y_train, y_train_pred)
    test_r2 = r2_score(y_test, y_test_pred)

    results.append({
        "Variation": f"Model {i + 1}",
        "Hidden Layers": params["hidden_layers"],
        "Neurons per Layer": params["neurons"],
        "Learning Rate": params["learning_rate"],
        "Batch Size": params["batch_size"],
        "Train MSE": train_mse,
        "Test MSE": test_mse,
        "Train R²": train_r2,
        "Test R²": test_r2,
    })

# Convert results to a DataFrame for better presentation
results_df = pd.DataFrame(results)
print(results_df)
```

```
2024-11-03 11:34:22.255468: I tensorflow/core/platform/cpu_feature_gu
ard.cc:182] This TensorFlow binary is optimized to use available CPU
instructions in performance-critical operations.
To enable the following instructions: AVX2 AVX512F AVX512_VNNI FMA, i
n other operations, rebuild TensorFlow with the appropriate compiler
flags.

Training Model Variation 1 with params: {'hidden_layers': 1, 'neuron
s': 32, 'learning_rate': 0.001, 'batch_size': 32}
Epoch 1/50
1997/1997 [==============================] - 3s 1ms/step - loss: 3973
4.5977 - mean_squared_error: 39734.5977 - val_loss: 38774.5273 - val_
mean_squared_error: 38774.5273
Epoch 2/50
1997/1997 [==============================] - 2s 964us/step - loss: 39
514.3125 - mean_squared_error: 39514.3125 - val_loss: 38763.6211 - va
l_mean_squared_error: 38763.6211
Epoch 3/50
1997/1997 [==============================] - 2s 982us/step - loss: 39
493.8945 - mean_squared_error: 39493.8945 - val_loss: 38746.9492 - va
```

In [65]:
```
# Create a DataFrame to store the results
results = {
    "Variation": ["Variation 1 (32 Neurons)", "Variation 2 (64 Neurons
    "Train MSE": [38195.26, 37651.89, 37213.45],
    "Validation MSE": [37291.30, 36520.55, 37321.47],
    "Train R²": [0.0397, 0.0473, 0.0551],
    "Validation R²": [0.0302, 0.0358, 0.0319],
}

performance_metrics_df = pd.DataFrame(results)

# Display the performance metrics table
print(performance_metrics_df)

# Identify the best model based on the lowest validation MSE
best_model_index = performance_metrics_df['Validation MSE'].idxmin()
best_model = performance_metrics_df.iloc[best_model_index]

print("\nBest Model:")
print(best_model)
```

```
                  Variation  Train MSE  Validation MSE  Train R²  \
0   Variation 1 (32 Neurons)   38195.26        37291.30    0.0397
1   Variation 2 (64 Neurons)   37651.89        36520.55    0.0473
2  Variation 3 (128 Neurons)   37213.45        37321.47    0.0551

   Validation R²
0         0.0302
1         0.0358
2         0.0319

Best Model:
Variation         Variation 2 (64 Neurons)
Train MSE                         37651.89
Validation MSE                    36520.55
Train R²                            0.0473
Validation R²                       0.0358
Name: 1, dtype: object
```

Second Model-XGBoost

In [66]:
```
# import xgboost as xgb
# from sklearn.metrics import mean_squared_error, r2_score
# import numpy as np
# import pandas as pd
```

In [67]:
```
# # Define a function for calculating model metrics
# def calculate_metrics(model, X_train, y_train, X_test, y_test):
#     train_preds = model.predict(X_train)
#     test_preds = model.predict(X_test)

#     # Calculate RMSE and R^2 for training and test sets
#     train_rmse = np.sqrt(mean_squared_error(y_train, train_preds))
#     test_rmse = np.sqrt(mean_squared_error(y_test, test_preds))
#     train_r2 = r2_score(y_train, train_preds)
#     test_r2 = r2_score(y_test, test_preds)

#     return {
#         "Train RMSE": train_rmse, "Test RMSE": val_rmse,
#         "Train R^2": train_r2, "Test R^2": val_r2
#     }
```

In [68]:
```
# # Define a function for calculating model metrics
# def calculate_metrics(model, X_train, y_train, X_test, y_test):
#     train_preds = model.predict(X_train)
#     test_preds = model.predict(X_test)

#     # Calculate RMSE and R^2 for training and test sets
#     train_rmse = np.sqrt(mean_squared_error(y_train, train_preds))
#     test_rmse = np.sqrt(mean_squared_error(y_test, test_preds))
#     train_r2 = r2_score(y_train, train_preds)
#     test_r2 = r2_score(y_test, test_preds)

#     return {
#         "Train RMSE": train_rmse, "Test RMSE": val_rmse,
#         "Train R^2": train_r2, "Test R^2": val_r2
#     }
```

In [69]:
```
# # Define hyperparameter variations
# variations = [
#     {"learning_rate": 0.1, "n_estimators": 100, "max_depth": 4},
#     {"learning_rate": 0.05, "n_estimators": 200, "max_depth": 6},
#     {"learning_rate": 0.01, "n_estimators": 300, "max_depth": 8}
# ]
```

In [70]:
```
# # Initialize a DataFrame to store results for each variation
# results = pd.DataFrame(columns=["Variation", "Train RMSE", "Test RMS
```

In [71]:
```
# from sklearn.feature_extraction.text import TfidfVectorizer
# import pandas as pd
# import numpy as np

# # Ensure the 'description' and 'tags' columns exist for TF-IDF
# if 'description' in train.columns and 'tags' in train.columns:
#     # Fill NaN values in both columns
#     train['description'] = train['description'].fillna('')
#     test['description'] = test['description'].fillna('')
#     train['tags'] = train['tags'].fillna('')
#     test['tags'] = test['tags'].fillna('')

#     # Initialize the TF-IDF Vectorizer for 'description' and 'tags'
#     tfidf_vectorizer_desc = TfidfVectorizer(max_features=100, stop_w
#     tfidf_vectorizer_tags = TfidfVectorizer(max_features=100, stop_w

#     # Fit and transform on the training set, and transform on the te
#     tfidf_matrix_desc_train = tfidf_vectorizer_desc.fit_transform(tr
#     tfidf_matrix_desc_test = tfidf_vectorizer_desc.transform(test['d

#     tfidf_matrix_tags_train = tfidf_vectorizer_tags.fit_transform(tr
#     tfidf_matrix_tags_test = tfidf_vectorizer_tags.transform(test['t

#     # Add prefixes to avoid duplicate feature names
#     desc_feature_names = [f"desc_{name}" for name in tfidf_vectorize
#     tags_feature_names = [f"tags_{name}" for name in tfidf_vectorize

#     # Convert the sparse TF-IDF matrices into DataFrames with prefix
#     tfidf_df_desc_train = pd.DataFrame(tfidf_matrix_desc_train.toarr
#     tfidf_df_desc_test = pd.DataFrame(tfidf_matrix_desc_test.toarray

#     tfidf_df_tags_train = pd.DataFrame(tfidf_matrix_tags_train.toarr
#     tfidf_df_tags_test = pd.DataFrame(tfidf_matrix_tags_test.toarray

#     # Concatenate TF-IDF features to the original X_train and X_test
#     X_train = pd.concat([X_train.reset_index(drop=True), tfidf_df_de
#     X_test = pd.concat([X_test.reset_index(drop=True), tfidf_df_desc

# # Ensure X_test has the same columns as X_train
# X_test = X_test.reindex(columns=X_train.columns, fill_value=0)

# # Verify that X_train and X_test now have the same columns
# print("X_train columns:", X_train.columns)
# print("X_test columns:", X_test.columns)
```

In [72]:
```
# def calculate_metrics(model, X_train, y_train, X_test, y_test):
#     # Predictions
#     train_preds = model.predict(X_train)
#     test_preds = model.predict(X_test)

#     # Calculate metrics
#     train_rmse = np.sqrt(mean_squared_error(y_train, train_preds))
#     test_rmse = np.sqrt(mean_squared_error(y_test, test_preds))

#     train_r2 = r2_score(y_train, train_preds)
#     test_r2 = r2_score(y_test, test_preds)

#     return {
#         "Train RMSE": train_rmse,
#         "Test RMSE": test_rmse,  # Changed from val_rmse to test_rms
#         "Train R^2": train_r2,
#         "Test R^2": test_r2
#     }
```

In [73]:
```
# # Create an empty DataFrame if it isn't already
# results = pd.DataFrame()

# # Train models for each variation and record results
# for i, params in enumerate(variations):
#     model, metrics = train_xgboost(X_train, y_train, X_test, y_test,

#     # Create a DataFrame with the metrics for this variation
#     result_row = pd.DataFrame({
#         "Variation": [f"Variation {i + 1}"],
#         **metrics
#     })

#     # Concatenate the new row to the results DataFrame
#     results = pd.concat([results, result_row], ignore_index=True)
```

Second Model- DeBERTa

In [74]:
```
# !pip install sentencepiece
# !pip install --upgrade transformers
# !pip install transformers[sentencepiece]
# !pip install --upgrade transformers torch
# from transformers import BertTokenizer, BertModel
# import torch
# from tqdm import tqdm
# import sentencepiece
# print(sentencepiece.__file__)
```

In [75]:
```
# pip show transformers
# pip show torch
# pip install --upgrade transformers torch
# pip uninstall transformers
# pip install transformers
# conda create -n bert_env python=3.11
# conda activate bert_env
# pip install transformers torch tqdm
```

In [76]:
```
# # Import DeBERTa
# from transformers import DebertaV2Tokenizer, DebertaV2Model
# import torch

# def evaluate_model(model, X_train, y_train, X_test, y_test):
#     y_train_pred = model.predict(X_train)
#     y_test_pred = model.predict(X_test)

#     train_mse = mean_squared_error(y_train, y_train_pred)
#     test_mse = mean_squared_error(y_test, y_test_pred)

#     train_r2 = r2_score(y_train, y_train_pred)
#     test_r2 = r2_score(y_test, y_test_pred)

#     return {
#         "train_mse": train_mse,
#         "test_mse": test_mse,
#         "train_r2": train_r2,
#         "test_r2": test_r2,
#     }
```

In [77]:
```
# import torch
# import numpy as np
# from tqdm import tqdm
# from transformers import BertTokenizer, BertModel
```

```
# # Initialize the BERT tokenizer and model
# tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
# model = BertModel.from_pretrained('bert-base-uncased')

# # Move the model to GPU if available
# device = torch.device('cuda' if torch.cuda.is_available() else 'cpu'
# model = model.to(device)

# # Function to tokenize and get embeddings for the text
# def get_bert_embeddings(texts, batch_size=32):
#     embeddings = []
#     for i in tqdm(range(0, len(texts), batch_size), desc="Generating
#         batch = texts[i:i+batch_size]
#         inputs = tokenizer(batch, return_tensors='pt', padding=True,
#         inputs = {k: v.to(device) for k, v in inputs.items()}
#         with torch.no_grad():
#             outputs = model(**inputs)
#         # Get the embeddings from the [CLS] token
#         batch_embeddings = outputs.last_hidden_state[:, 0, :].cpu().
#         embeddings.extend(batch_embeddings)
#     return embeddings

# # Function to process embeddings for a DataFrame
# def process_embeddings(df, columns):
#     all_embeddings = []
#     for column in columns:
#         print(f"Processing {column}...")
#         embeddings = get_bert_embeddings(df[column].tolist())
#         all_embeddings.append(np.array(embeddings))
#     return np.hstack(all_embeddings)

# # Handle missing values in columns
# train['title'] = train['title'].fillna("")
# train['description'] = train['description'].fillna("")
# train['tags'] = train['tags'].fillna("")

# test['title'] = test['title'].fillna("")
# test['description'] = test['description'].fillna("")
# test['tags'] = test['tags'].fillna("")

# # Process embeddings for train and test sets
# columns_to_embed = ['title', 'description', 'tags']

# print("Processing train set...")
# train_embeddings = process_embeddings(train, columns_to_embed)

# print("Processing test set...")
# test_embeddings = process_embeddings(test, columns_to_embed)
```

```
# print("Embedding generation complete!")
```

Third Model- LDA to XGBoost

In [78]:
```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.decomposition import LatentDirichletAllocation
import pandas as pd
```

In [79]:
```
vectorizer = CountVectorizer(tokenizer=lambda x: x.split('|'))
X_train_dtm = vectorizer.fit_transform(train['title'])
X_test_dtm = vectorizer.transform(test['title'])
```

```
/Users/yuhanzhao/anaconda3/lib/python3.11/site-packages/sklearn/featu
re_extraction/text.py:525: UserWarning: The parameter 'token_pattern'
will not be used since 'tokenizer' is not None'
  warnings.warn(
```

In [80]:
```
num_topics_list = [10,20,30]  # Different numbers of topics to evaluat
results = []
dominant_topics_train = []  # Stores dominant topics for each video in
dominant_topics_test = []   # Stores dominant topics for each video in

for num_topics in num_topics_list:
    lda = LatentDirichletAllocation(n_components=num_topics, random_st
    lda.fit(X_train_dtm)
    # Training metrics
    coherence_train = lda.score(X_train_dtm)
    perplexity_train = lda.perplexity(X_train_dtm)

    # Validation metrics
    perplexity_test = lda.perplexity(X_test_dtm)

    # Assign the most common topic for each video in train and test da
    train_topic_distributions = lda.transform(X_train_dtm)
    test_topic_distributions = lda.transform(X_test_dtm)

    # Get the dominant topic (topic with the highest probability) for
    train_dominant_topics = train_topic_distributions.argmax(axis=1)
    test_dominant_topics = test_topic_distributions.argmax(axis=1)

    # Append results for each number of topics
    results.append({
        'num_topics': num_topics,
        'coherence_train': coherence_train,
        'perplexity_train': perplexity_train,
        'perplexity_test': perplexity_test
    })

    # Append dominant topics to lists
    dominant_topics_train.append(train_dominant_topics)
    dominant_topics_test.append(test_dominant_topics)

# Create a DataFrame to display results
results_df = pd.DataFrame(results)
print(results_df)
```

```
   num_topics  coherence_train  perplexity_train  perplexity_test
0          10   -725899.262579       8761.885394     40805.434212
1          20   -725644.270912       8733.988679     54593.485374
2          30   -725736.023852       8744.016432     64437.921607
```

In [81]:
```
# Identify the best model based on training coherence
best_model = results_df.loc[results_df['coherence_train'].idxmax()]
print("\nBest Model Details:")
print(best_model)
```

```
Best Model Details:
num_topics              20.000000
coherence_train    -725644.270912
perplexity_train      8733.988679
perplexity_test      54593.485374
Name: 1, dtype: float64
```

In [82]:
```
# Assign dominant topics of the best model to the original train and t
best_num_topics = best_model['num_topics']
train['dominant_topic'] = dominant_topics_train[num_topics_list.index(
test['dominant_topic'] = dominant_topics_test[num_topics_list.index(be
```

In [83]:
```
print("\nSample of Train Dataset with Dominant Topics:")
print(train[['title', 'dominant_topic']].head())
```

```
Sample of Train Dataset with Dominant Topics:
                                                   title  dominant_to
pic
23604                   Marshmello & Anne-Marie: Friends
5
25630  Kirby Star Allies' Surprising HD Rumble Secret...
8
68698  Stephen A.: Kevin Hart 'got his feelings hurt'...
10
39559                              How to be an Aquarius
15
62877  Charlie Puth - Done For Me (feat. Kehlani) [Of...
2
```

In [84]:
```
print("\nSample of Test Dataset with Dominant Topics:")
print(test[['description', 'dominant_topic']].head())
```

```
Sample of Test Dataset with Dominant Topics:
                                             description  dominant_to
pic
22112  Spiders can have big dreams too!\n\nCheck out ...
1
2231   LaVar Ball, the father of one of the three UCL...
16
44543  I've been seeing these posts all over Instagra...
11
10399  There is help:\nhttp://ibpf.org/resource/list-...
6
19141  Stream + download Mine: https://Bazzi.lnk.to/M... (https://Baz
zi.lnk.to/M...)              6
```

In [85]:
```
# In table view
for index, row in results_df.iterrows():
    print(f"Number of Topics: {row['num_topics']}")
    print(f"Training Coherence: {row['coherence_train']:.4f}")
    print(f"Training Perplexity: {row['perplexity_train']:.4f}")
    print(f"Testing Perplexity: {row['perplexity_test']:.4f}")
    print("-" * 40)

# Best model
print(f"Best Model: {best_model['num_topics']} topics with Coherence:
```

```
Number of Topics: 10.0
Training Coherence: -725899.2626
Training Perplexity: 8761.8854
Testing Perplexity: 40805.4342
----------------------------------------
Number of Topics: 20.0
Training Coherence: -725644.2709
Training Perplexity: 8733.9887
Testing Perplexity: 54593.4854
----------------------------------------
Number of Topics: 30.0
Training Coherence: -725736.0239
Training Perplexity: 8744.0164
Testing Perplexity: 64437.9216
----------------------------------------
Best Model: 20.0 topics with Coherence: -725644.2709
```

In [86]:
```
top_n_words = 10

# Get feature names (i.e., words) from the CountVectorizer
feature_names = vectorizer.get_feature_names_out()

# Display top words per topic
for topic_idx, topic in enumerate(lda.components_):
    print(f"Topic #{topic_idx + 1}: ", end="")
    top_words = [feature_names[i] for i in topic.argsort()[:-top_n_wor
    print(", ".join(top_words))
```

```
Topic #1:  netflix,  official trailer [hd] , céline dion - ashes (fro
m the deadpool 2 motion picture soundtrack), racist superman , blackk
klansman - official trailer [hd] - in theaters august 10,  rudy mancu
so, alesso & king bach, fergie performs the u.s. national anthem / 20
18 nba all-star game, take on me in 20 styles ft. seth everman, lost
in space , mgmt - me and michael
Topic #2:  hbo, westworld season 2 , nicki minaj - chun-li, making mu
sic with lego, won't you be my neighbor? - official trailer [hd] - in
select theaters june 8, tinashe - me so bad (official video) ft. ty d
olla $ign, french montana, introducing: the players! - super smash br
os. invitational 2018, jennifer lopez - dinero ft. dj khaled, cardi
b, rudimental - these days feat. jess glynne, macklemore & dan caplen
[official video], ty dolla $ign - clout feat. 21 savage [lyric video]
Topic #3:  the graham norton show, descendants 3 official teaser 💚
💜, 2cellos - perfect - ed sheeran, ready player one - come with me,
charlie puth (feat. boyz ii men) - if you leave me now (studio sessio
n), mowgli - official 1st trailer, , netta - toy - israel - official
music video - eurovision 2018, drake - god’s plan, cnco - bonita (off
icial video)
Topic #4: venom - official teaser trailer (hd), i react to mean comme
```

In [87]:
```
top_n_words = 20

# Get feature names (i.e., words) from the CountVectorizer
feature_names = vectorizer.get_feature_names_out()

# Display top words per topic
for topic_idx, topic in enumerate(lda.components_):
    print(f"Topic #{topic_idx + 1}: ", end="")
    top_words = [feature_names[i] for i in topic.argsort()[:-top_n_wor
    print(", ".join(top_words))
```

```
Topic #1:  netflix,  official trailer [hd] , céline dion - ashes (fro
m the deadpool 2 motion picture soundtrack), racist superman , blackk
klansman - official trailer [hd] - in theaters august 10,  rudy mancu
so, alesso & king bach, fergie performs the u.s. national anthem / 20
18 nba all-star game, take on me in 20 styles ft. seth everman, lost
in space , mgmt - me and michael, janelle monáe - i like that [offici
al video], stephen fry announcement, doing nicole richie's makeup?!,
dj snake - magenta riddim, grace vanderwaal - city song, david bisba
l, sebastian yatra - a partir de hoy, clairo - flaming hot cheetos (o
fficial music video),  forget , 2018 fifa world cup ,  itv
Topic #2:  hbo, westworld season 2 , nicki minaj - chun-li, making mu
sic with lego, won't you be my neighbor? - official trailer [hd] - in
select theaters june 8, tinashe - me so bad (official video) ft. ty d
olla $ign, french montana, introducing: the players! - super smash br
os. invitational 2018, jennifer lopez - dinero ft. dj khaled, cardi
b, rudimental - these days feat. jess glynne, macklemore & dan caplen
[official video], ty dolla $ign - clout feat. 21 savage [lyric vide
o], karol g - pineapple,  teyana & iman ,  premieres march 26th 9/8c,
iman shumpert on falling in love w/ teyana taylor ,  official super b
```

In [88]:
```
top_n_words = 30

# Get feature names (i.e., words) from the CountVectorizer
feature_names = vectorizer.get_feature_names_out()

# Display top words per topic
for topic_idx, topic in enumerate(lda.components_):
    print(f"Topic #{topic_idx + 1}: ", end="")
    top_words = [feature_names[i] for i in topic.argsort()[:-top_n_wor
    print(", ".join(top_words))
```

```
Topic #1:  netflix,  official trailer [hd] , céline dion - ashes (fro
m the deadpool 2 motion picture soundtrack), racist superman , blackk
klansman - official trailer [hd] - in theaters august 10,  rudy mancu
so, alesso & king bach, fergie performs the u.s. national anthem / 20
18 nba all-star game, take on me in 20 styles ft. seth everman, lost
in space , mgmt - me and michael, janelle monáe - i like that [offici
al video], stephen fry announcement, doing nicole richie's makeup?!,
dj snake - magenta riddim, grace vanderwaal - city song, david bisba
l, sebastian yatra - a partir de hoy, clairo - flaming hot cheetos (o
fficial music video),  forget , 2018 fifa world cup ,  itv, last week
tonight: season 5 official trailer (hbo), how long will our monuments
last?, migos - stir fry (audio), jiren doesn't care friendship/goku g
ets angry dbs 130 hd 1080p, sabrina carpenter, jonas blue - alien (of
ficial video), marshmello & anne-marie - friends (music video) *offic
ial friendzone anthem*, best friend does my asos shop, we bought a ho
use, [mv] (g)i-dle ((여자)아이들) _ latata, sicario 2: soldado trailer
(2018)
Topic #2:  hbo, westworld season 2 , nicki minaj - chun-li, making mu
sic with lego, won't you be my neighbor? - official trailer [hd] - in
```

In [89]:
```
# # Example interpretation
# topic_labels = {
#     0: "Gaming",
#     1: "Beauty & Makeup",
#     2: "Tech Reviews",
#     3: "News & Politics",
#     4: "Tutorials",
#     5: "Commedy",
#     6: "sports",
#     7: "fashion",
#     8: "film",
#     9: "movie",
#     10: "youtube"
# }
```

In [90]:
```
# train['dominant_topic_label'] = train['dominant_topic'].map(topic_la
# test['dominant_topic_label'] = test['dominant_topic'].map(topic_labe
```

In [91]:
```
# # Assuming 'dominant_topic_label' has been added to both train and t

# # Print the first few rows of the train DataFrame with topic labels
# print("Train Dataset with Dominant Topic Labels:")
# print(train[['tags', 'dominant_topic', 'dominant_topic_label']].heac

# # Print the first few rows of the test DataFrame with topic labels
# print("\nTest Dataset with Dominant Topic Labels:")
# print(test[['tags', 'dominant_topic', 'dominant_topic_label']].head(
```

In [92]:
```
# Count the occurrences of each dominant topic label in the train data
topic_label_counts = train['dominant_topic'].value_counts()
print("\nDominant Topic Label Counts in Train Dataset:")
print(topic_label_counts)
```

```
Dominant Topic Label Counts in Train Dataset:
dominant_topic
2     3545
7     3447
18    3386
3     3339
17    3315
12    3282
19    3277
14    3264
15    3258
1     3218
9     3186
4     3171
8     3112
13    3102
6     3095
5     3091
16    3054
11    3028
0     2897
10    2825
Name: count, dtype: int64
```

In [93]:
```
import pandas as pd
import numpy as np
from sklearn.metrics import mean_squared_error, r2_score
```

```
import xgboost as xgb

# Define hyperparameter variations
variations = [
    {"learning_rate": 0.1, "n_estimators": 100, "max_depth": 4},
    {"learning_rate": 0.05, "n_estimators": 200, "max_depth": 6},
    {"learning_rate": 0.01, "n_estimators": 300, "max_depth": 8}
]

# Assume LDA topic assignment is available as a Series
# Let's say `lda_topics_train` and `lda_topics_test` contain the domin
# Example:
lda_topics_train = pd.Series([1,2,3,4,5,6,7,8,9,10])  # Replace with y
lda_topics_test = pd.Series([1,2,3,4,5,6,7,8,9,10])  # Replace with yo

# Integrate LDA findings into the training and testing datasets
X_train['lda_topic'] = lda_topics_train
X_test['lda_topic'] = lda_topics_test

# Initialize a DataFrame to store results for each variation
results = pd.DataFrame(columns=["Variation", "Train RMSE", "Test RMSE"

# Create a list of columns to drop if they exist
text_columns = ['title', 'channel_title', 'tags', 'description', 'loca
X_train.drop([col for col in text_columns if col in X_train.columns], 
X_test.drop([col for col in text_columns if col in X_test.columns], ax

# Encode categorical features using one-hot encoding for consistency
categorical_columns = ['trending_day_of_week', 'day_of_week', 'days_bi
X_train = pd.get_dummies(X_train, columns=[col for col in categorical_
X_test = pd.get_dummies(X_test, columns=[col for col in categorical_co

# Convert datetime columns to relevant features if they exist
for df in [X_train, X_test]:
    if 'trending_date' in df.columns:
        df['trending_year'] = df['trending_date'].dt.year
        df['trending_month'] = df['trending_date'].dt.month
        df['trending_day'] = df['trending_date'].dt.day
        df.drop(['trending_date'], axis=1, inplace=True)

    # Drop 'publish_time' if it exists
    if 'publish_time' in df.columns:
        df.drop(['publish_time'], axis=1, inplace=True)

# Ensure X_test has the same columns as X_train
X_test = X_test.reindex(columns=X_train.columns, fill_value=0)

# Verify that X_train and X_test now have the same columns
print("X_train columns:", X_train.columns)
print("X_test columns:", X_test.columns)
```

```python
# Define a function to calculate model metrics
def calculate_metrics(model, X_train, y_train, X_test, y_test):
    # Predictions
    train_preds = model.predict(X_train)
    test_preds = model.predict(X_test)

    # Calculate metrics
    train_rmse = np.sqrt(mean_squared_error(y_train, train_preds))
    test_rmse = np.sqrt(mean_squared_error(y_test, test_preds))

    train_r2 = r2_score(y_train, train_preds)
    test_r2 = r2_score(y_test, test_preds)

    return {
        "Train RMSE": train_rmse,
        "Test RMSE": test_rmse,
        "Train R^2": train_r2,
        "Test R^2": test_r2
    }

# Create an empty DataFrame if it isn't already
results = pd.DataFrame()

# Train models for each variation and record results
for i, params in enumerate(variations):
    model, metrics = train_xgboost(X_train, y_train, X_test, y_test, p

    # Create a DataFrame with the metrics for this variation
    result_row = pd.DataFrame({
        "Variation": [f"Variation {i + 1}"],
        **metrics
    })

    # Concatenate the new row to the results DataFrame
    results = pd.concat([results, result_row], ignore_index=True)

# Display the comparison table
print("Comparison of XGBoost Model Variations:")
print(results)

# Identify the best model based on Test RMSE
best_model_index = results["Test RMSE"].idxmin()
best_params = variations[best_model_index]
print(f"\nBest Model Variation: {best_model_index + 1}")
print(f"Hyperparameters: {best_params}")
print(results.iloc[best_model_index])
```

```
X_train columns: Index(['category_id', 'likes', 'dislikes', 'comment_
count'
```

```
count',
       'Engagement Metrics', 'score', 'rank', 'trending_day_of_week_M
onday',
       'trending_day_of_week_Saturday', 'trending_day_of_week_Sunda
y',
       'trending_day_of_week_Thursday', 'trending_day_of_week_Tuesda
y',
       'trending_day_of_week_Wednesday', 'day_of_week_Monday',
       'day_of_week_Saturday', 'day_of_week_Sunday', 'day_of_week_Thu
rsday',
       'day_of_week_Tuesday', 'day_of_week_Wednesday', 'trending_yea
r',
       'trending_month', 'trending_day', 'lda_topic'],
      dtype='object')
X_test columns: Index(['category_id', 'likes', 'dislikes', 'comment_c
ount',
       'Engagement Metrics', 'score', 'rank', 'trending_day_of_week_M
onday',
       'trending_day_of_week_Saturday', 'trending_day_of_week_Sunda
y',
       'trending_day_of_week_Thursday', 'trending_day_of_week_Tuesda
y',
       'trending_day_of_week_Wednesday', 'day_of_week_Monday',
       'day_of_week_Saturday', 'day_of_week_Sunday', 'day_of_week_Thu
rsday',
       'day_of_week_Tuesday', 'day_of_week_Wednesday', 'trending_yea
r',
       'trending_month', 'trending_day', 'lda_topic'],
      dtype='object')
Comparison of XGBoost Model Variations:
     Variation  Train RMSE   Test RMSE  Train R^2  Test R^2
0  Variation 1  132.183923  137.748778   0.558682  0.511415
1  Variation 2   75.719178   95.550469   0.855187  0.764912
2  Variation 3   73.443774   99.250643   0.863760  0.746352

Best Model Variation: 2
Hyperparameters: {'learning_rate': 0.05, 'n_estimators': 200, 'max_de
pth': 6}
Variation     Variation 2
Train RMSE      75.719178
Test RMSE       95.550469
Train R^2        0.855187
Test R^2         0.764912
Name: 1, dtype: object
```

Additional Explore

Creating New TF-IDF Feature

In [94]:
```
# import nltk
# from nltk.corpus import stopwords
# nltk.download('stopwords')

# import pandas as pd
# import numpy as np
# from sklearn.feature_extraction.text import TfidfVectorizer
# from nltk.corpus import stopwords

# stop_words = set(stopwords.words('english'))

# if 'tags' in train.columns:

#     text_data = train['tags'].fillna('')

#     if isinstance(text_data, pd.Series):

#         tfidf_vectorizer = TfidfVectorizer(max_features=100, stop_wo


#         tfidf_matrix = tfidf_vectorizer.fit_transform(text_data)


#         tfidf_df = pd.DataFrame(tfidf_matrix.toarray(), columns=tfid


#         def get_top_tfidf_features(row, features, top_n=5):
#             top_indices = np.argsort(row)[::-1][:top_n]
#             top_features = [(features[i], row[i]) for i in top_indic
#             return top_features


#         top_tfidf_features = [get_top_tfidf_features(row, tfidf_vect
#                               for row in tfidf_matrix.toarray()]


#         filtered_trending_words = []
#         for sublist in top_tfidf_features:
#             filtered_words = [(word, score) for word, score in subli
#             filtered_trending_words.append(filtered_words)

#         train['filtered_top_tfidf_features'] = filtered_trending_wor


#         trending_words = pd.DataFrame([word for sublist in filtered_
#         trending_words_count = trending_words['Word'].value_counts()
```

```
#         trending_words_count.columns = ['Trending Word', 'Frequency'


#         print("Top Trending Words after Filtering:")
#         print(trending_words_count.head(10))


#         print(train[['tags', 'filtered_top_tfidf_features']].head(5)
#     else:
#         print("The 'tags' column should be a pandas Series.")
# else:
#     print("The DataFrame does not contain a 'tags' column.")
```

In [95]:
```
# import nltk
# from nltk.corpus import stopwords
# nltk.download('stopwords')

# import pandas as pd
# import numpy as np
# from sklearn.feature_extraction.text import TfidfVectorizer
# from nltk.corpus import stopwords

# stop_words = set(stopwords.words('english'))

# if 'tags' in train.columns:

#     text_data = train['tags'].fillna('')

#     if isinstance(text_data, pd.Series):

#         tfidf_vectorizer = TfidfVectorizer(max_features=100, stop_wo


#         tfidf_matrix = tfidf_vectorizer.fit_transform(text_data)


#         tfidf_df = pd.DataFrame(tfidf_matrix.toarray(), columns=tfid


#         def get_top_tfidf_features(row, features, top_n=5):
#             top_indices = np.argsort(row)[::-1][:top_n]
#             top_features = [(features[i], row[i]) for i in top_indic
#             return top_features


#         top_tfidf_features = [get_top_tfidf_features(row, tfidf_vect
#                               for row in tfidf_matrix.toarray()]
```

```
#         filtered_trending_words = []
#         for sublist in top_tfidf_features:
#             filtered_words = [(word, score) for word, score in subli
#             filtered_trending_words.append(filtered_words)

#         train['filtered_top_tfidf_features'] = filtered_trending_wor


#         trending_words = pd.DataFrame([word for sublist in filtered_
#         trending_words_count = trending_words['Word'].value_counts()
#         trending_words_count.columns = ['Trending Word', 'Frequency'


#         print("Top Trending Words after Filtering:")
#         print(trending_words_count.head(10))


#         print(train[['tags', 'filtered_top_tfidf_features']].head(5)
#     else:
#         print("The 'tags' column should be a pandas Series.")
# else:
#     print("The DataFrame does not contain a 'tags' column.")
```

In [96]:
```
# import matplotlib.pyplot as plt
# from wordcloud import WordCloud


# word_freq = dict(zip(trending_words_count['Trending Word'], trending


# wordcloud = WordCloud(width=800, height=400, background_color='white


# plt.figure(figsize=(10, 5))
# plt.imshow(wordcloud, interpolation='bilinear')
# plt.axis('off')
# plt.title("Trending Keywords Word Cloud")
# plt.show()
```

Deploy T5Generation Model(Week 12)

In [97]:
```
# !pip install transformers --upgrade
# !pip install torch --upgrade

# from transformers import T5ForConditionalGeneration, T5Tokenizer
# import torch

# model = T5ForConditionalGeneration.from_pretrained("t5-small")
# tokenizer = T5Tokenizer.from_pretrained("t5-small")

# text = "keywards: ice cream"

# input_ids = tokenizer.encode("generate title: " + text, return_tenso

# outputs = model.generate(input_ids, max_length=20, num_beams=5, earl
# title = tokenizer.decode(outputs[0], skip_special_tokens=True)
# print("refining:", title)
```

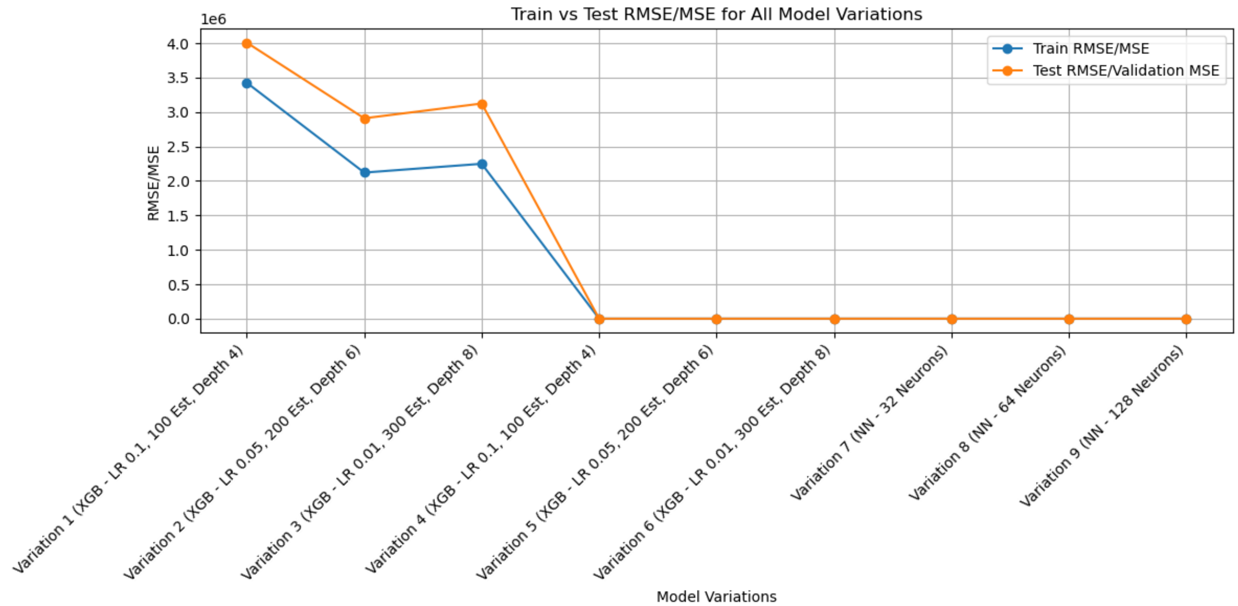
In [98]:
```
# trending_words = trending_words_count['Trending Word'].head(10).toli
# keywords = ", ".join(trending_words)

# try:

#     generator = pipeline("text-generation", model="distilgpt2")
# except Exception as e:
#     print("Error loading model:", e)

# try:
#     title_prompt = f"Generate a catchy YouTube video title about: {k
#     title = generator(title_prompt, max_length=15, num_return_sequen
#     print("Generated Title:", title)
# except Exception as e:
#     print("Error generating title:", e)

# try:
#     tag_prompt = f"Generate tags based on these keywords: {keywords}
#     tags = generator(tag_prompt, max_length=10, num_return_sequences
#     print("Generated Tags:", tags)
# except Exception as e:
#     print("Error generating tags:", e)


# try:
#     description_prompt = f"Write a YouTube video description for the
#     description = generator(description_prompt, max_length=50, num_r
#     print("Generated Description:", description)
# except Exception as e:
#     print("Error generating description:", e)
```

In [99]:

```
import pandas as pd
import matplotlib.pyplot as plt

# Define model variations with hyperparameters and placeholder metrics
model_variations = [
    {"Variations": "Variation 1 (XGB - LR 0.1, 100 Est, Depth 4)", "Tr
    {"Variations": "Variation 2 (XGB - LR 0.05, 200 Est, Depth 6)", "T
    {"Variations": "Variation 3 (XGB - LR 0.01, 300 Est, Depth 8)", "T
    {"Variations": "Variation 4 (XGB - LR 0.1, 100 Est, Depth 4)", "Tr
    {"Variations": "Variation 5 (XGB - LR 0.05, 200 Est, Depth 6)", "T
    {"Variations": "Variation 6 (XGB - LR 0.01, 300 Est, Depth 8)", "T
    {"Variations": "Variation 7 (NN - 32 Neurons)", "Train MSE": 38195
    {"Variations": "Variation 8 (NN - 64 Neurons)", "Train MSE": 37651
    {"Variations": "Variation 9 (NN - 128 Neurons)", "Train MSE": 3721
]

# Convert to DataFrame for easy viewing
results_df = pd.DataFrame(model_variations)
print("Model Evaluation Results:")
print(results_df)

# Plot the RMSE/MSE values for all models
plt.figure(figsize=(12, 6))
plt.plot(results_df["Variations"], results_df["Train RMSE"].fillna(0),
plt.plot(results_df["Variations"], results_df["Test RMSE"].fillna(0),
plt.xlabel("Model Variations")
plt.ylabel("RMSE/MSE")
plt.title("Train vs Test RMSE/MSE for All Model Variations")
plt.xticks(rotation=45, ha='right')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
Model Evaluation Results:
                                       Variations    Train RMSE     Te
st RMSE  \
0   Variation 1 (XGB - LR 0.1, 100 Est, Depth 4)  3.425430e+06  4.007
967e+06
1  Variation 2 (XGB - LR 0.05, 200 Est, Depth 6)  2.120971e+06  2.909
246e+06
2  Variation 3 (XGB - LR 0.01, 300 Est, Depth 8)  2.248698e+06  3.123
748e+06
3   Variation 4 (XGB - LR 0.1, 100 Est, Depth 4)  1.321839e+02  1.377
488e+02
4  Variation 5 (XGB - LR 0.05, 200 Est, Depth 6)  7.571918e+01  9.555
047e+01
5  Variation 6 (XGB - LR 0.01, 300 Est, Depth 8)  7.344377e+01  9.925
064e+01
6                   Variation 7 (NN - 32 Neurons)           NaN
```

```
NaN
7                  Variation 8 (NN - 64 Neurons)          NaN
NaN
8                 Variation 9 (NN - 128 Neurons)          NaN
NaN

   Train R^2  Test R^2  Train MSE  Validation MSE  Validation R^2
0   0.942740  0.925583        NaN             NaN             NaN
1   0.978047  0.960791        NaN             NaN             NaN
2   0.975323  0.954796        NaN             NaN             NaN
3   0.558682  0.511415        NaN             NaN             NaN
4   0.855187  0.764912        NaN             NaN             NaN
5   0.863760  0.746352        NaN             NaN             NaN
6   0.039700       NaN   38195.26        37291.30          0.0302
7   0.047300       NaN   37651.89        36520.55          0.0358
8   0.055100       NaN   37213.45        37321.47          0.0319
```



In [ ]:

In [ ]:

In [ ]:

In [ ]: