

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 filenames
df_gb = load_csv_from_zip(os.path.join(datasets_path, 'GBvideos.csv.zip'), 'GB')
df_us = load_csv_from_zip(os.path.join(datasets_path, 'USvideos.csv.zip'), 'US')

# 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
3  Salford City Football Club        17 2017-11-13T02:30:38.000Z
4       Cute Girl Videos        25 2017-11-13T01:45:13.000Z

      tags      views   likes \
0  christmas|"john lewis christmas"|"john lewis"|...  7224515  55681
1  SNL|"Saturday Night Live"|"SNL Season 43"|"Epi...  1053632  25561
2  Eminem|"Walk"|"On"|"Water"|"Aftermath/Shady/In...  17158579  787420
3  Salford City FC|"Salford City"|"Salford"|"Clas...  27833    193
4                  [none]         9815     30

      dislikes  comment_count      thumbnail_link \
0      10247        9479  https://i.ytimg.com/vi/Jw1Y-zhQURU/default.jpg
1      2294        2757  https://i.ytimg.com/vi/3s1rvMFUweQ/default.jpg
2      43420       125882  https://i.ytimg.com/vi/n1WpP7iowLc/default.jpg
3       12          37  https://i.ytimg.com/vi/PUTEiSjKwJU/default.jpg
4        2          30  https://i.ytimg.com/vi/rHwDegptbI4/default.jpg

      comments_disabled  ratings_disabled  video_error_or_removed \
0             False            False                False
1             False            False                False
2             False            False                False
3             False            False                False
4             False            False                False

                           description      location
0  Click here to continue the story and make your...  Great Britain
1  Musical guest Taylor Swift performs ...Ready for...  Great Britain
2  Eminem's new track Walk on Water ft. Beyoncé i...  Great Britain
3  Salford drew 4-4 against the Class of 92 and F...  Great Britain
4  Dashcam captures truck's near miss with child ...  Great Britain

```

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)

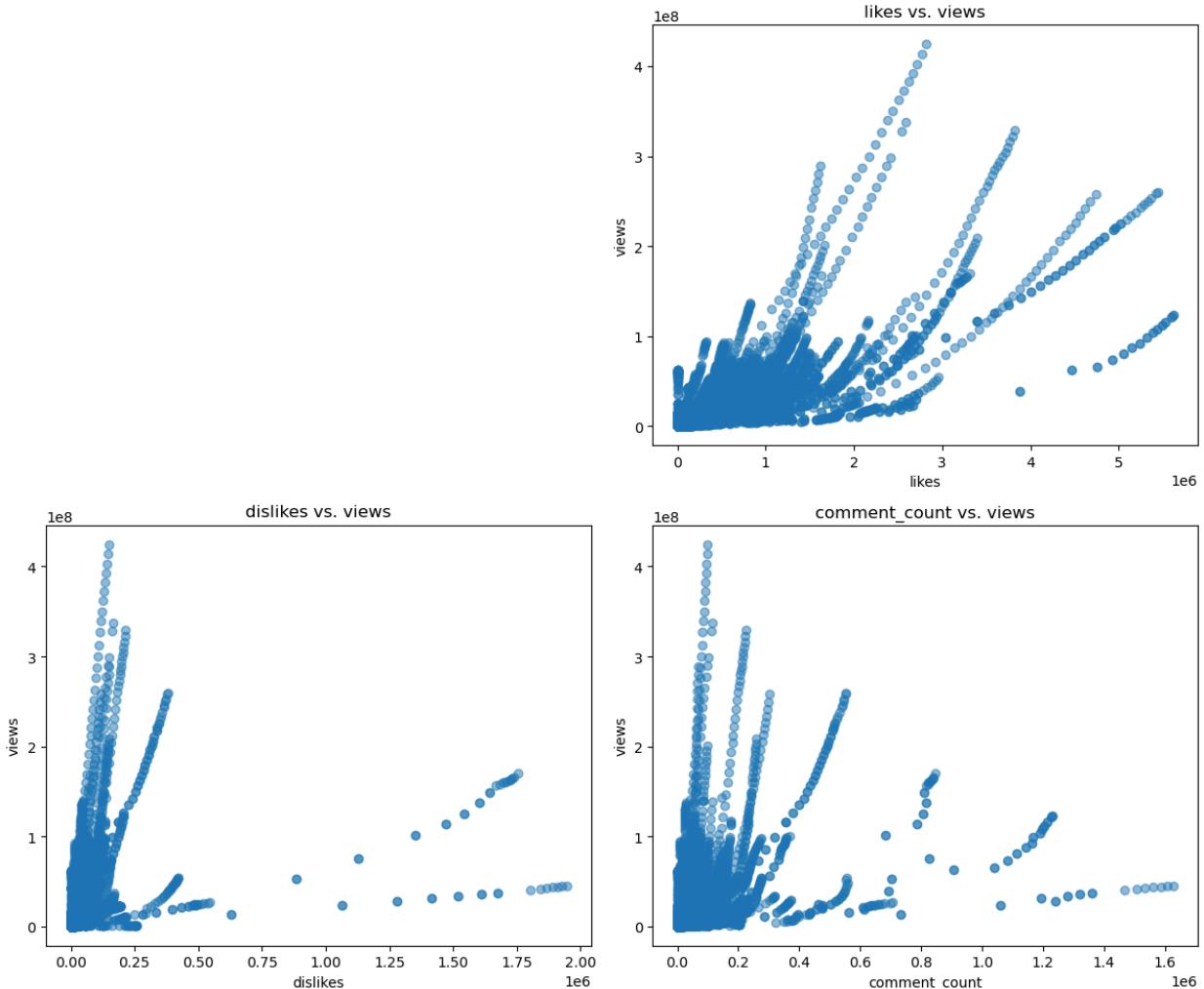
Check Outliers

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']

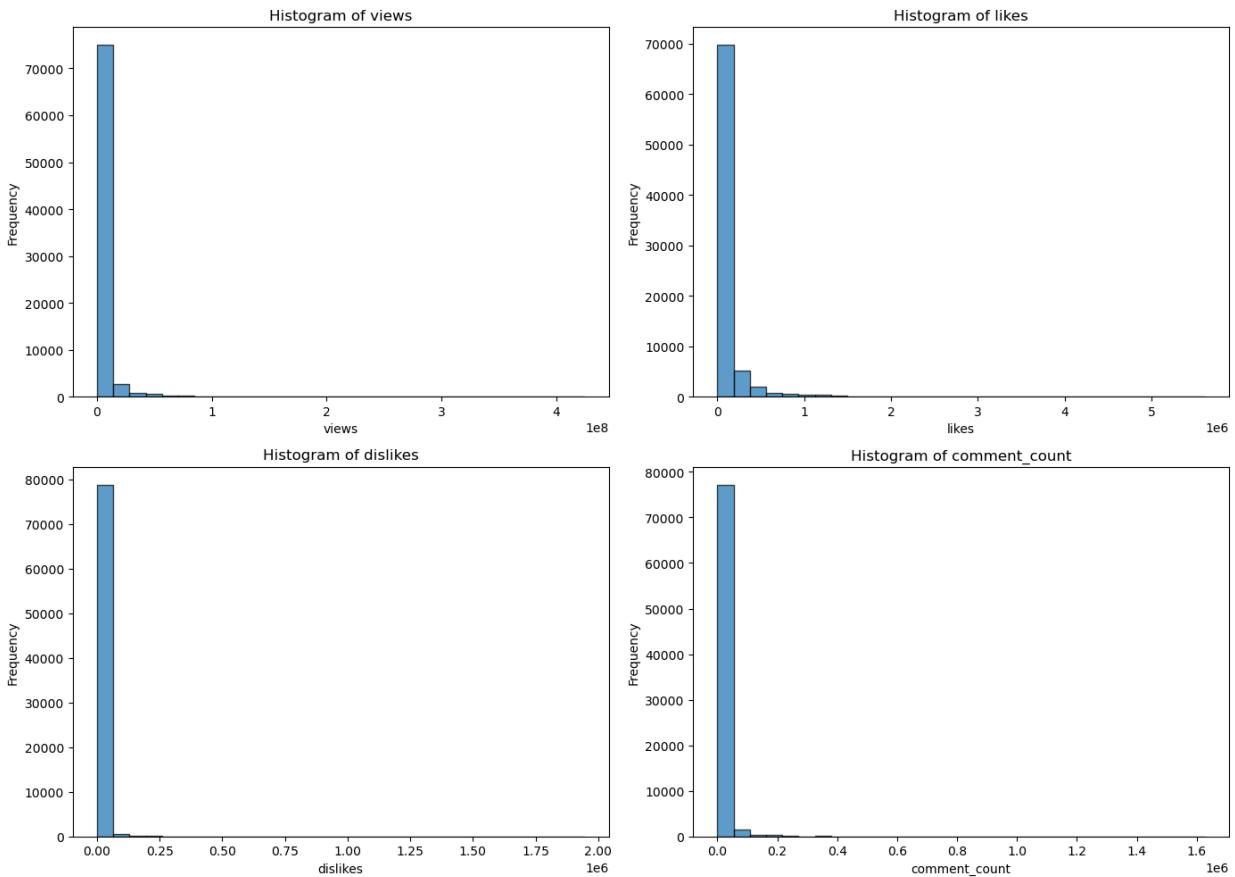
# Create histograms for each numerical column
plt.figure(figsize=(14, 10))
for i, column in enumerate(numerical_columns, 1):
```

```

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

```

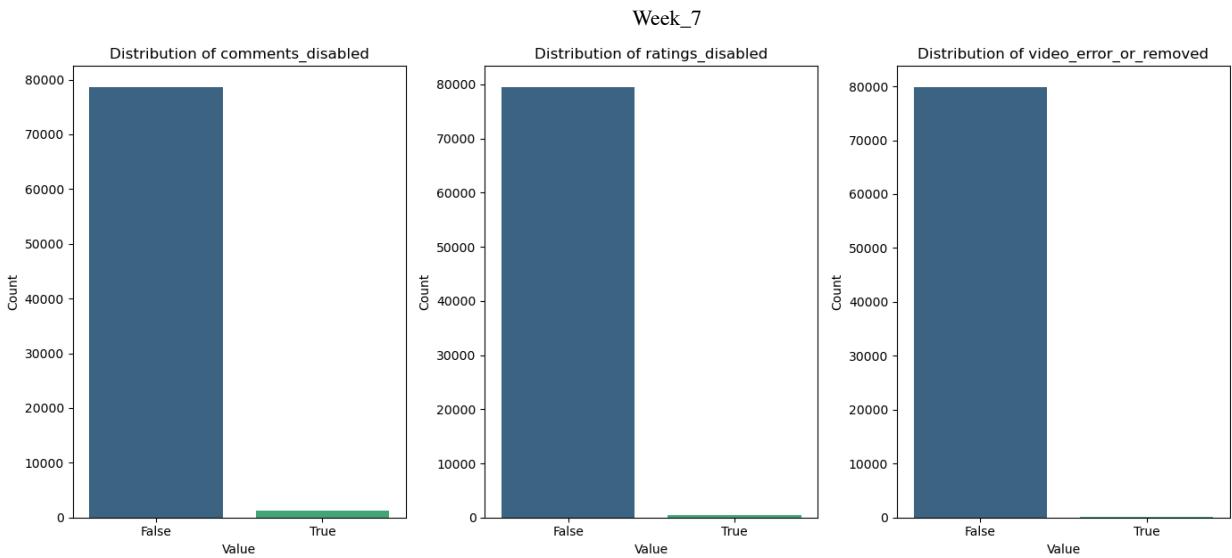
In [7]: import seaborn as sns

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

# 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()

```



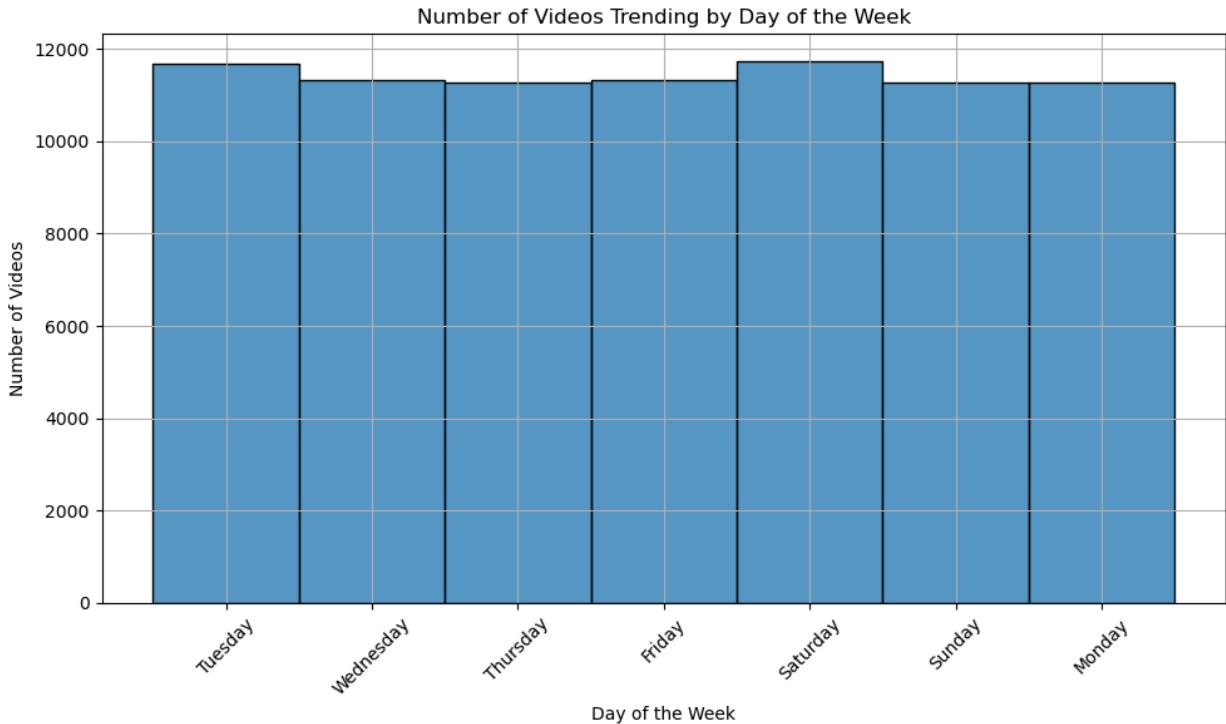
EDA for Date-Time Variables

```
In [8]: # convert the trending_date to datetime type
merged_df['trending_date'] = pd.to_datetime(merged_df['trending_date'], format='%Y-%m-%d')
# Extract day of the week from 'trending_date'
merged_df['trending_day_of_week'] = merged_df['trending_date'].dt.day_name()

# Plot histogram of trending day of the week
plt.figure(figsize=(10, 6))
sns.histplot(merged_df['trending_day_of_week'], discrete=True, palette='viridis')
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/hnph1r15tq8t0xq_j4_rzmh0000gn/T/ipykernel_27027/1621989413.py:8: UserWarning: Ignoring `palette` because no `hue` variable has been assigned.

```
    sns.histplot(merged_df['trending_day_of_week'], discrete=True, palette='viridis')
```

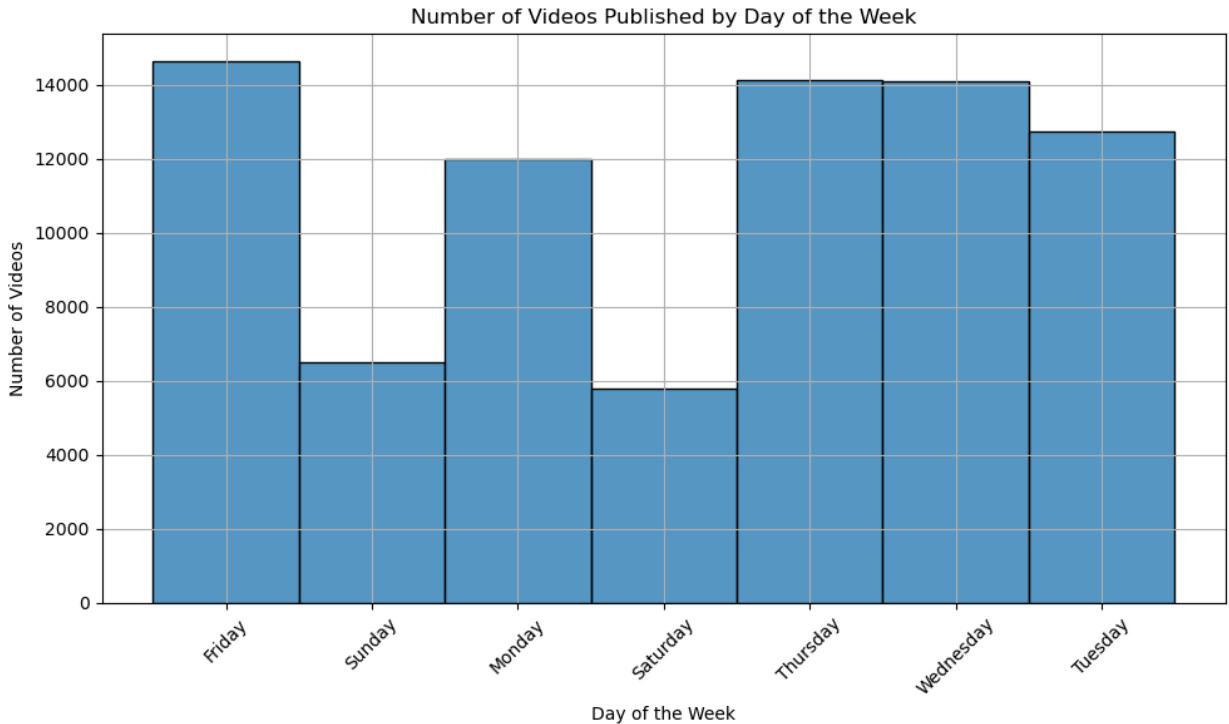


```
In [9]: #convert the publish_date to datetime type
merged_df['publish_time'] = pd.to_datetime(merged_df['publish_time'], format='%Y-%m-%d %H:%M:%S')
# Extract day of the week from 'publish_time'
merged_df['day_of_week'] = merged_df['publish_time'].dt.day_name()

# 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/hnph1r15tq8t0xq_j4_rzmh0000gn/T/ipykernel_27027/1346077495.py:8: UserWarning: Ignoring `palette` because no `hue` variable has been assigned.

```
sns.histplot(merged_df['day_of_week'], discrete=True, palette='viridis')
```



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
	dislikes	comment_count	
count	7.986500e+04	7.986500e+04	
mean	5.612328e+03	1.070850e+04	
min	0.000000e+00	0.000000e+00	
25%	2.010000e+02	6.420000e+02	
50%	7.120000e+02	2.099000e+03	
75%	2.527000e+03	7.220000e+03	
max	1.944971e+06	1.626501e+06	
std	4.124462e+04	4.443679e+04	

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

# Sort by 'N' in descending order
category_counts = category_counts.sort_values(by='N', ascending=False)
category_counts['category_id'] = pd.Categorical(category_counts['category_id'])

# 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(category_names)

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

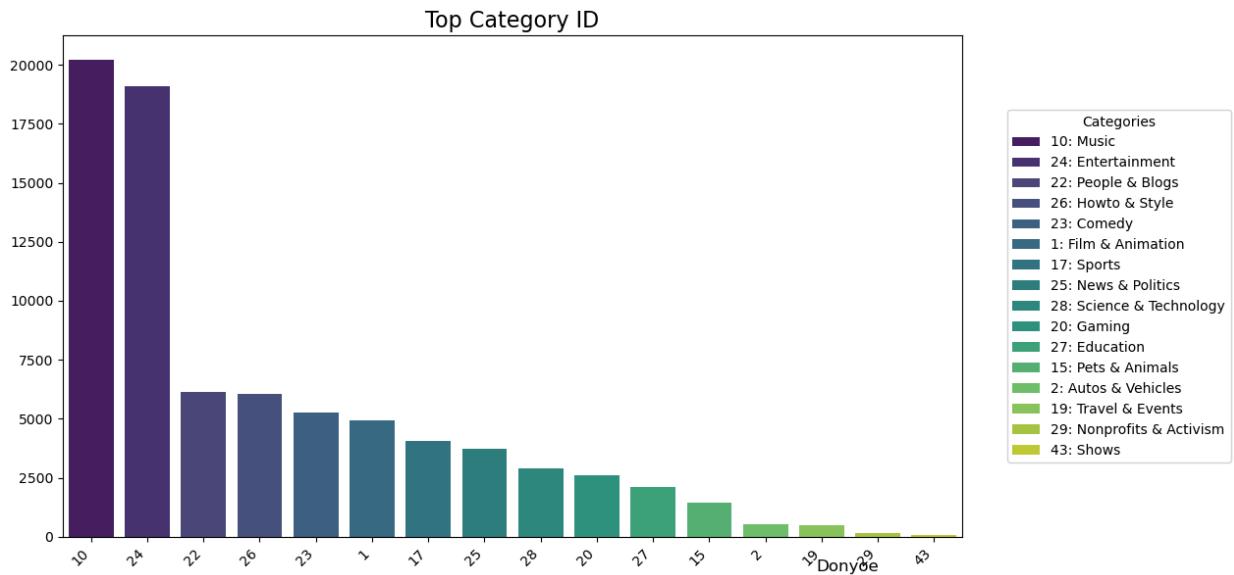
# Customize the plot to match your ggplot2 example
plt.title("Top Category ID", fontsize=16)
plt.xlabel(None)
plt.ylabel(None)
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.figtext(0.9, 0.02, "Donyoe", horizontalalignment='right', fontsize=12)

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

# Position the legend on the right of the plot using 'bbox_to_anchor'
plt.legend(handles[:len(legend_labels)], labels=legend_labels, title='Categories',
           bbox_to_anchor=(1.05, 0.5), loc='center left', borderaxespad=0)

plt.show()

```



Data Transformation-Create Engagement Metrics

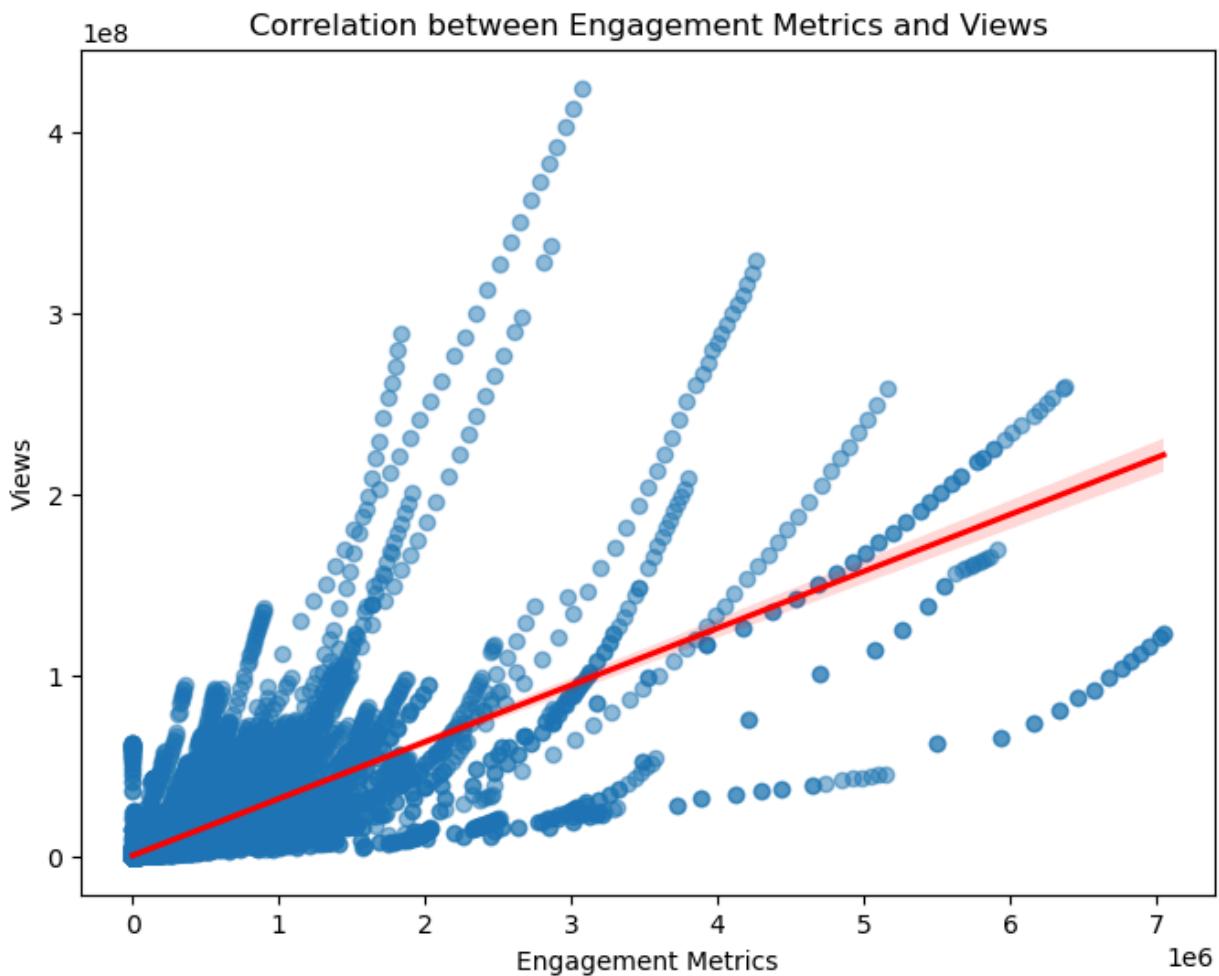
```
In [13]: # Create a new column
merged_df['Engagement Metrics'] = merged_df['likes'] + merged_df['dislikes'] +
# Display the DataFrame to check the new column
print(merged_df[['likes', 'dislikes', 'comment_count', 'Engagement Metrics']].head())

```

	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

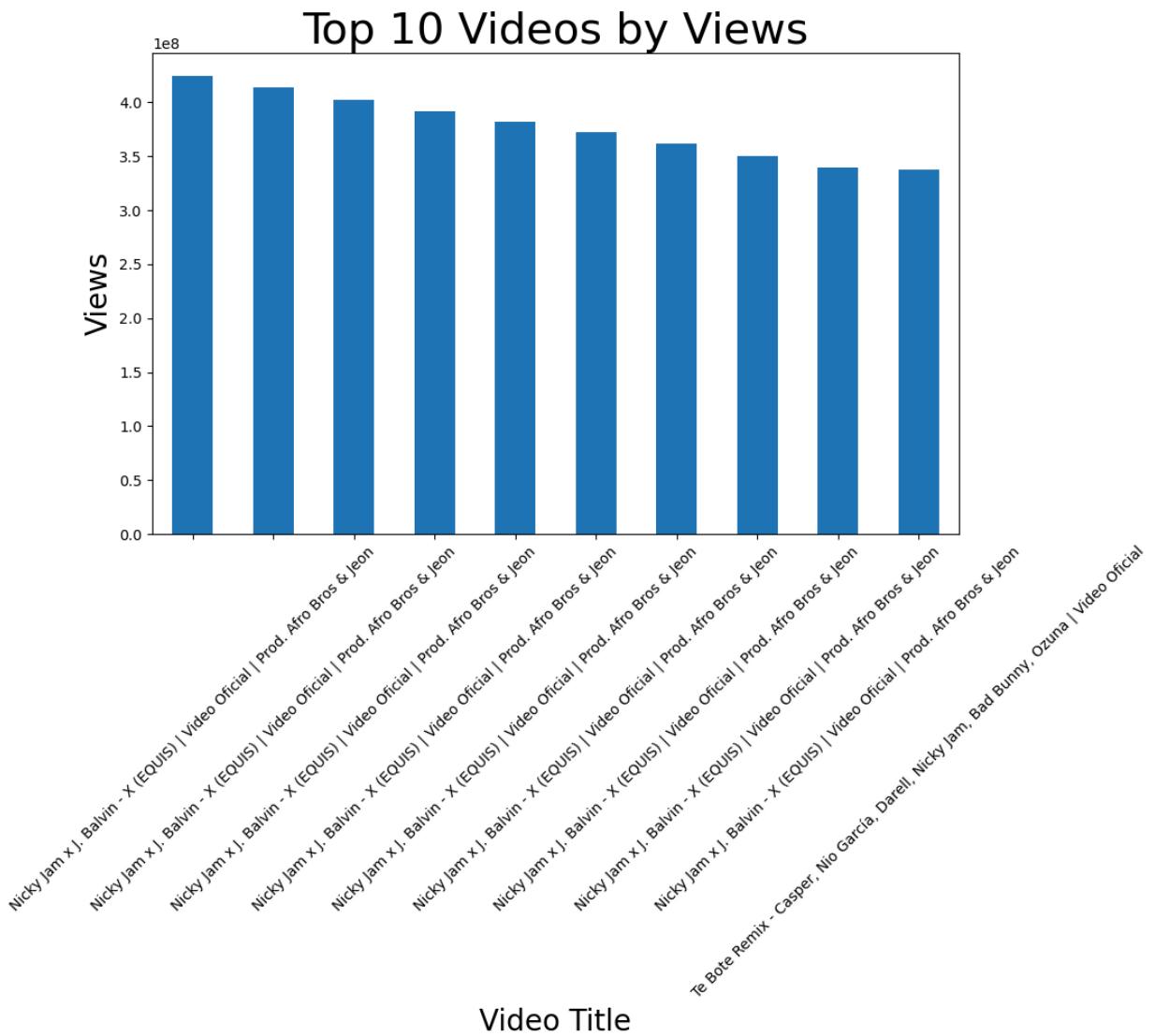
Visualization-Engagement Metrics

```
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_kws={'alpha': 0.5})
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...		2582218
34460	Te Bote Remix - Casper, Nio García, Darell, Ni...			2862074
25545	Bad Bunny - Amorfoda	Video Oficial		4264625
34292	Te Bote Remix - Casper, Nio García, Darell, Ni...			2813339
26671	Nicky Jam x J. Balvin - X (EQUIS)	Video Ofic...		2505131
25341	Bad Bunny - Amorfoda	Video Oficial		4231351
25137	Bad Bunny - Amorfoda	Video Oficial		4198350
26477	Nicky Jam x J. Balvin - X (EQUIS)	Video Ofic...		2427694
24935	Bad Bunny - Amorfoda	Video Oficial		4167420
24738	Bad Bunny - Amorfoda	Video Oficial		4135956
24540	Bad Bunny - Amorfoda	Video Oficial		4103146
26280	Nicky Jam x J. Balvin - X (EQUIS)	Video Ofic...		2350490
33729	Te Bote Remix - Casper, Nio García, Darell, Ni...			2661680
24339	Bad Bunny - Amorfoda	Video Oficial		4062651
33613	Te Bote Remix - Casper, Nio García, Darell, Ni...			2610210
24137	Bad Bunny - Amorfoda	Video Oficial		4026487
21162	Ozuna x Romeo Santos - El Farsante Remix			1836833
26079	Nicky Jam x J. Balvin - X (EQUIS)	Video Ofic...		2269962
23938	Bad Bunny - Amorfoda	Video Oficial		3996243
20957	Ozuna x Romeo Santos - El Farsante Remix			1815236
23728	Bad Bunny - Amorfoda	Video Oficial		3966424
33441	Te Bote Remix - Casper, Nio García, Darell, Ni...			2538184
25883	Nicky Jam x J. Balvin - X (EQUIS)	Video Ofic...		2192147
23528	Bad Bunny - Amorfoda	Video Oficial		3930504
20752	Ozuna x Romeo Santos - El Farsante Remix			1792693
23324	Bad Bunny - Amorfoda	Video Oficial		3894230
33270	Te Bote Remix - Casper, Nio García, Darell, Ni...			2474011
25688	Nicky Jam x J. Balvin - X (EQUIS)	Video Ofic...		2107200
20551	Ozuna x Romeo Santos - El Farsante Remix			1768430
23122	Bad Bunny - Amorfoda	Video Oficial		3849549
38401	Childish Gambino - This Is America (Official V...			6377774
38235	Childish Gambino - This Is America (Official V...			6356524
25936	Drake - God's Plan			5156827
33106	Te Bote Remix - Casper, Nio García, Darell, Ni...			2412367
38056	Childish Gambino - This Is America (Official V...			6286180
20353	Ozuna x Romeo Santos - El Farsante Remix			1741314
25482	Nicky Jam x J. Balvin - X (EQUIS)	Video Ofic...		2031387
22926	Bad Bunny - Amorfoda	Video Oficial		3791325
37879	Childish Gambino - This Is America (Official V...			6243463
25741	Drake - God's Plan			5089683
37705	Childish Gambino - This Is America (Official V...			6193738
32939	Te Bote Remix - Casper, Nio García, Darell, Ni...			2352426

location

28412	Great Britain
28212	Great Britain
28008	Great Britain
27811	Great Britain
27615	Great Britain
27424	Great Britain

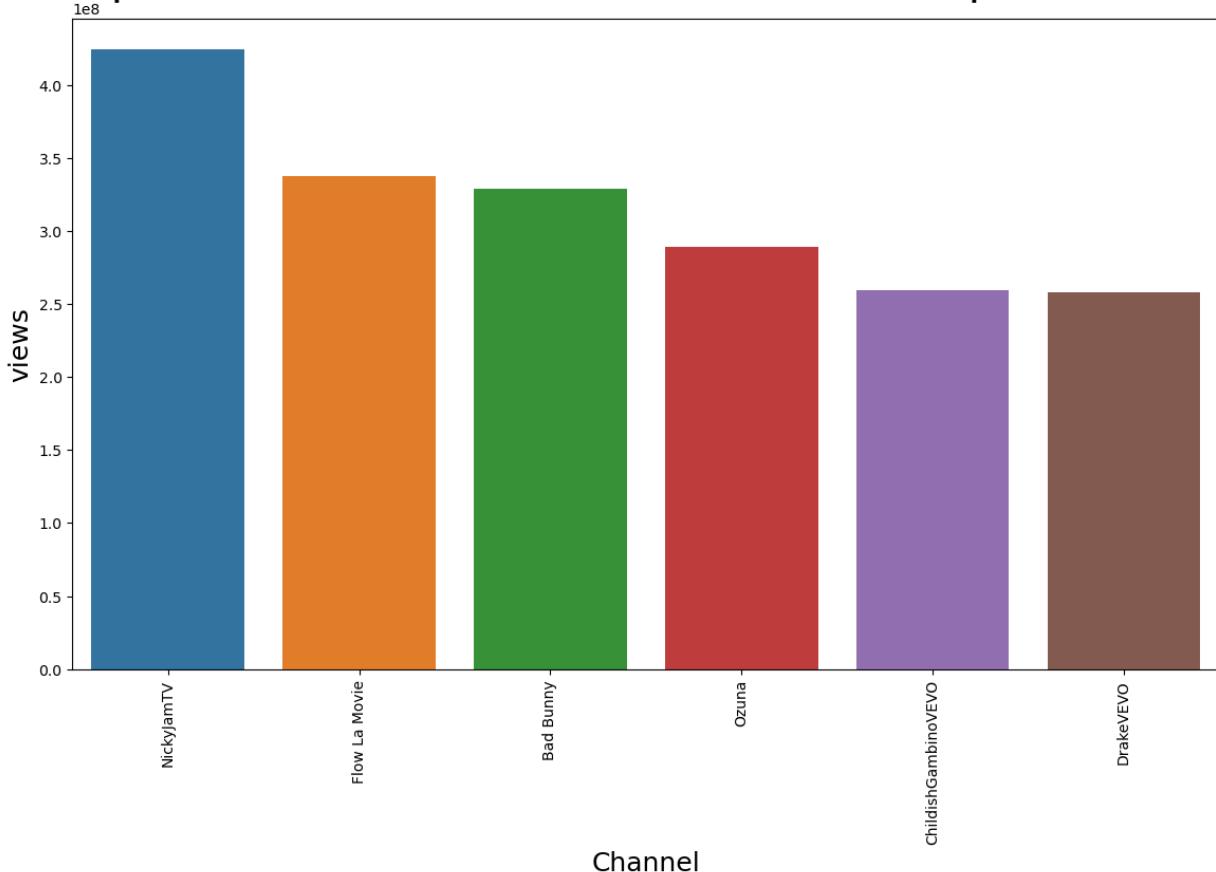
```
27241 Great Britain
27052 Great Britain
26861 Great Britain
34460 Great Britain
25545 Great Britain
34292 Great Britain
26671 Great Britain
25341 Great Britain
25137 Great Britain
26477 Great Britain
24935 Great Britain
24738 Great Britain
24540 Great Britain
26280 Great Britain
33729 Great Britain
24339 Great Britain
33613 Great Britain
24137 Great Britain
21162 Great Britain
26079 Great Britain
23938 Great Britain
20957 Great Britain
23728 Great Britain
33441 Great Britain
25883 Great Britain
23528 Great Britain
20752 Great Britain
23324 Great Britain
33270 Great Britain
25688 Great Britain
20551 Great Britain
23122 Great Britain
38401 Great Britain
38235 Great Britain
25936 Great Britain
33106 Great Britain
38056 Great Britain
20353 Great Britain
25482 Great Britain
22926 Great Britain
37879 Great Britain
25741 Great Britain
37705 Great Britain
32939 Great Britain
```

```
In [17]: import seaborn as sns
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))
sns.barplot(x='channel_title', y='views', data=content)
plt.title('Top 50 Channels with Most views from Top 50 Videos', fontsize=30)
plt.ylabel('views', fontsize=18)
plt.xlabel('Channel', fontsize=18)
plt.xticks(rotation=90)
plt.show()
```

Top 50 Channels with Most views from Top 50 Videos



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

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

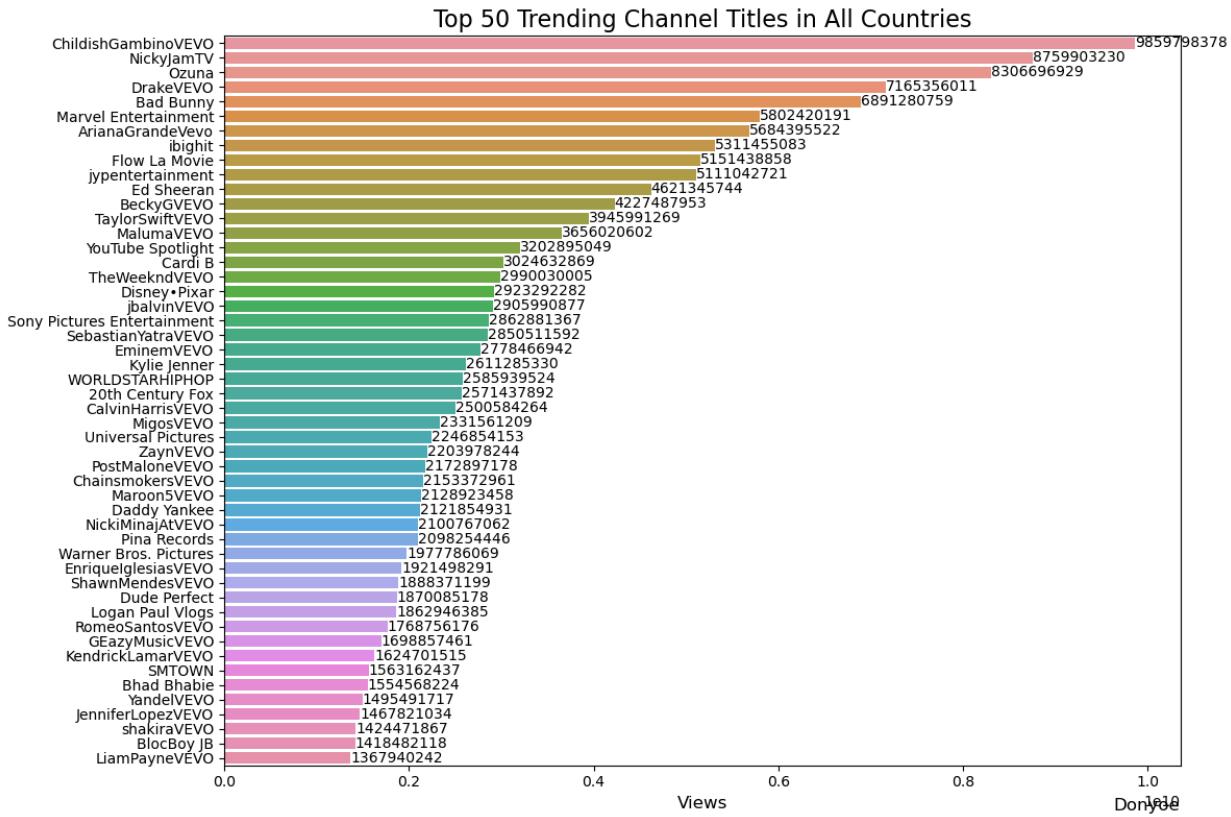
# Plot using seaborn
plt.figure(figsize=(12, 8))
ax = sns.barplot(x='views', y='channel_title', data=top_10_channels, orient='h'

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

# Customize the plot
plt.title('Top 50 Trending Channel Titles in All Countries', fontsize=16)
plt.xlabel('Views', fontsize=12)
plt.ylabel(None)
plt.xticks(rotation=0) # x-axis ticks don't need rotation in horizontal bar p
plt.tight_layout()

# Add caption
plt.figtext(0.95, 0.02, "Donyoe", horizontalalignment='right', fontsize=12)

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



Normalize and Standardize Data

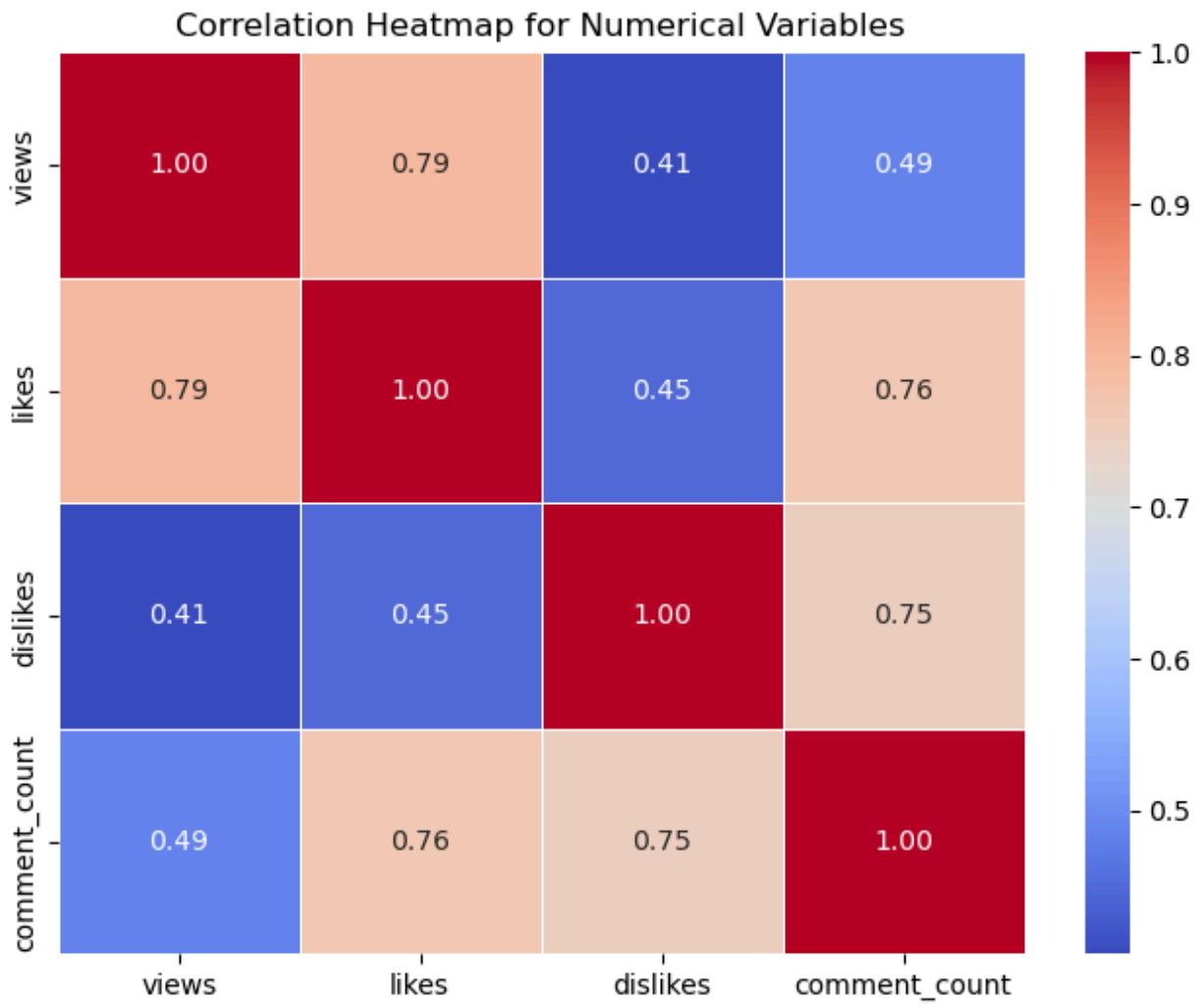
Correlation Metrics for Variables

```
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', linewidths=1)
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 heatmap
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_values.items}

# 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_correlations.items}

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

# 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='min')

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 All Locations")
#print(df_sorted[['video_id', 'views', 'likes', 'dislikes', 'comment_count', 'rank']])
#print(f"\nThe ranking of all videos from all locations has been saved to '{output_filename}'")
```

	video_id	trending_date	title	\	
36638	7C2z4GqqS5E	2018-06-01	BTS (방탄소년단) 'FAKE LOVE' Official MV		
77189	7C2z4GqqS5E	2018-06-01	BTS (방탄소년단) 'FAKE LOVE' Official MV		
76988	7C2z4GqqS5E	2018-05-31	BTS (방탄소년단) 'FAKE LOVE' Official MV		
36468	7C2z4GqqS5E	2018-05-31	BTS (방탄소년단) 'FAKE LOVE' Official MV		
36288	7C2z4GqqS5E	2018-05-30	BTS (방탄소년단) 'FAKE LOVE' Official MV		
...	
9146	LFhT6H6pRWg	2017-12-29	PSA from Chairman of the FCC Ajit Pai		
9354	LFhT6H6pRWg	2017-12-30	PSA from Chairman of the FCC Ajit Pai		
9575	LFhT6H6pRWg	2017-12-31	PSA from Chairman of the FCC Ajit Pai		
9777	LFhT6H6pRWg	2018-01-01	PSA from Chairman of the FCC Ajit Pai		
9985	LFhT6H6pRWg	2018-01-02	PSA from Chairman of the FCC Ajit Pai		
	channel_title	category_id	publish_time	\	
36638	ibighit	10	2018-05-18 09:00:02		
77189	ibighit	10	2018-05-18 09:00:02		
76988	ibighit	10	2018-05-18 09:00:02		
36468	ibighit	10	2018-05-18 09:00:02		
36288	ibighit	10	2018-05-18 09:00:02		
...	
9146	Daily Caller	22	2017-12-13 22:52:57		
9354	Daily Caller	22	2017-12-13 22:52:57		
9575	Daily Caller	22	2017-12-13 22:52:57		
9777	Daily Caller	22	2017-12-13 22:52:57		
9985	Daily Caller	22	2017-12-13 22:52:57		
		tags	views	likes	
\\					
36638	BIGHIT "빅히트" "방탄소년단" "BTS" "BANGTAN" "방탄" "FAK...	123010920	56138		
27					
77189	BIGHIT "빅히트" "방탄소년단" "BTS" "BANGTAN" "방탄" "FAK...	123010920	56138		
27					
76988	BIGHIT "빅히트" "방탄소년단" "BTS" "BANGTAN" "방탄" "FAK...	121219886	55952		
03					
36468	BIGHIT "빅히트" "방탄소년단" "BTS" "BANGTAN" "방탄" "FAK...	121219886	55952		
03					
36288	BIGHIT "빅히트" "방탄소년단" "BTS" "BANGTAN" "방탄" "FAK...	115664850	55305		
68					
...	
9146	thedc "dc" "washington dc" "washington" "the d...	1324657	10426		
9354	thedc "dc" "washington dc" "washington" "the d...	1331204	10463		
9575	thedc "dc" "washington dc" "washington" "the d...	1336646	10501		
9777	thedc "dc" "washington dc" "washington" "the d...	1342131	10538		
9985	thedc "dc" "washington dc" "washington" "the d...	1348067	10576		
	dislikes	...	comments_disabled	ratings_disabled	\
36638	206892	...	False	False	
77189	206892	...	False	False	
76988	205565	...	False	False	
36468	205565	...	False	False	
36288	200995	...	False	False	
...
9146	253677	...	False	False	
9354	254899	...	False	False	
9575	255956	...	False	False	
9777	256816	...	False	False	
9985	258504	...	False	False	
	video_error_or_removed	\			
36638		False			

```

77189          False
76988          False
36468          False
36288          False
...
9146          ...
9354          False
9575          False
9777          False
9985          False

                           description      location \
36638  BTS (방탄소년단) 'FAKE LOVE' Official MVDirector : ...  Great Britain
77189  BTS (방탄소년단) 'FAKE LOVE' Official MVDirector : ...       USA
76988  BTS (방탄소년단) 'FAKE LOVE' Official MVDirector : ...       USA
36468  BTS (방탄소년단) 'FAKE LOVE' Official MVDirector : ...  Great Britain
36288  BTS (방탄소년단) 'FAKE LOVE' Official MVDirector : ...  Great Britain
...
9146  Ajit Pai has been at the heart of the net neut...  Great Britain
9354  Ajit Pai has been at the heart of the net neut...  Great Britain
9575  Ajit Pai has been at the heart of the net neut...  Great Britain
9777  Ajit Pai has been at the heart of the net neut...  Great Britain
9985  Ajit Pai has been at the heart of the net neut...  Great Britain

trending_day_of_week  day_of_week Engagement Metrics      score \
36638            Friday     Friday    7049374  2.896622e+06
77189            Friday     Friday    7049374  2.896622e+06
76988           Thursday     Friday   7026094  2.887390e+06
36468           Thursday     Friday   7026094  2.887390e+06
36288           Wednesday     Friday   6944735  2.855162e+06
...
9146            ...
9354            ...
9575            ...
9777            ...
9985            ...

rank
36638    1.0
77189    1.0
76988    3.0
36468    3.0
36288    5.0
...
9146    79861.0
9354    79862.0
9575    79863.0
9777    79864.0
9985    79865.0

```

[79865 rows x 22 columns]

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']
)

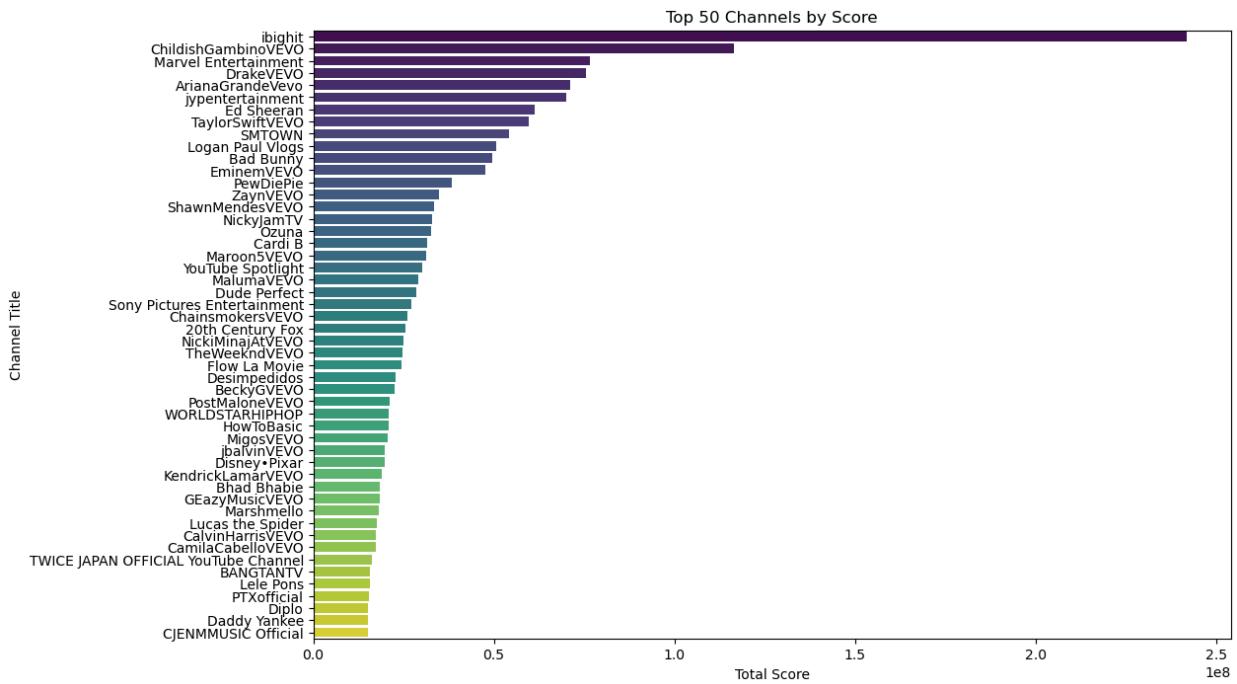
merged_df['rank'] = merged_df['score'].rank(ascending=False, method='min')

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

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

# 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='viridis')
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()
```



Channel Titles

```
In [24]: 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(
```



```
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

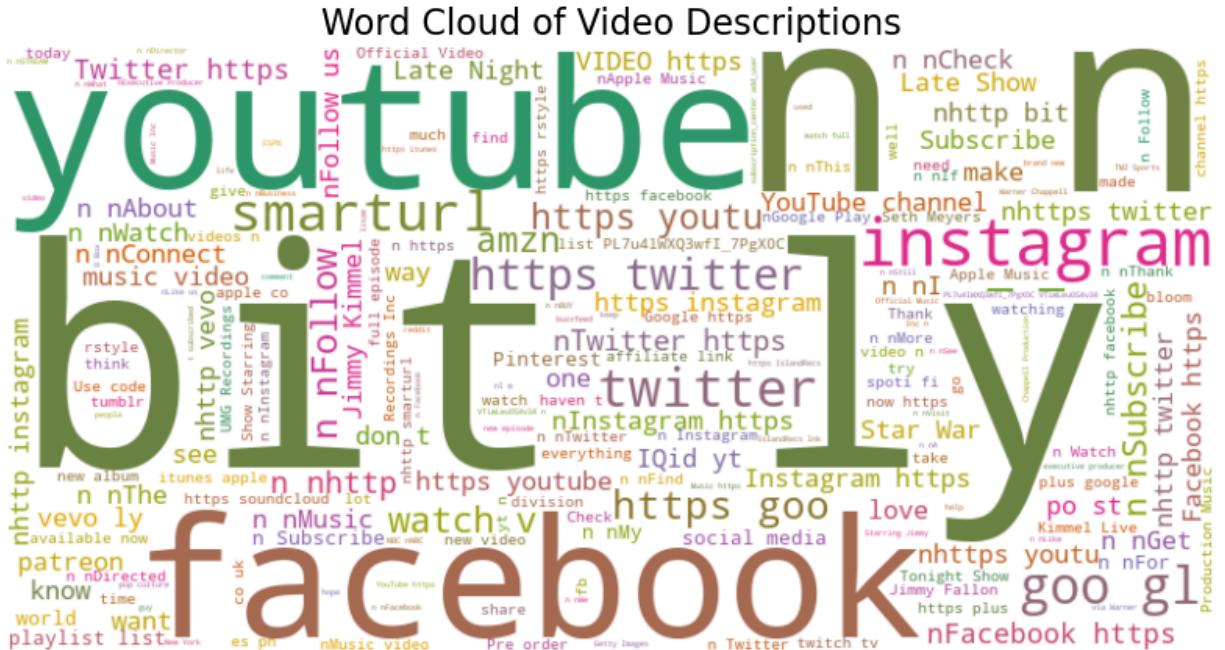
```
In [26]: all_description = " ".join(merged_df['description'].astype(str))

# 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()
```



Drop Unnecessary Columns

```
In [27]: #drop columns needed
merged_df.drop(columns=['thumbnail_link', 'video_id','comments_disabled','rating',
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     likes \
0  christmas|"john lewis christmas"|"john lewis"|...  7224515  55681
1  SNL|"Saturday Night Live"|"SNL Season 43"|"Epi...  1053632  25561
2  Eminem|"Walk"|"On"|"Water"|"Aftermath/Shady/In...  17158579  787420
3  Salford City FC|"Salford City"|"Salford"|"Clas...  27833    193
4                [none]          9815      30

dislikes  comment_count                      description
\
0      10247          9479 Click here to continue the story and make your...
1       2294          2757 Musical guest Taylor Swift performs ...Ready for...
2      43420         125882 Eminem's new track Walk on Water ft. Beyoncé i...
3        12            37 Salford drew 4-4 against the Class of 92 and F...
4        2            30 Dashcam captures truck's near miss with child ...

location trending_day_of_week day_of_week Engagement Metrics \
0  Great Britain          Tuesday     Friday      75407
1  Great Britain          Tuesday     Sunday      30612
2  Great Britain          Tuesday     Friday      956722
3  Great Britain          Tuesday     Monday      242
4  Great Britain          Tuesday     Monday       62

score      rank
0  25928.732602  24933.0
1  12021.653022  36735.0
2  389071.616394  1782.0
3   96.843978  77301.0
4  22.172624  78623.0

```

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 spaces)
text = re.sub(r'[^a-zA-Z\s]', '', text)

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

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

# 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	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 ...	
...	
79860	2018-06-14	The Cat Who Caught the Laser	
79861	2018-06-14	True Facts : Ant Mutualism	
79862	2018-06-14	I GAVE SAFIYA NYGAARD A PERFECT HAIR MAKEOVER ...	
79863	2018-06-14	How Black Panther Should Have Ended	
79864	2018-06-14	Official Call of Duty®: Black Ops 4 – Multipla...	
	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
...	
79860	AaronsAnimals	15	2018-05-18 13:00:04
79861	zefrank1	22	2018-05-18 01:00:06
79862	Brad Mondo	24	2018-05-18 17:34:22
79863	How It Should Have Ended	1	2018-05-17 17:00:04
79864	Call of Duty	20	2018-05-17 17:09:38
	tags	views	likes \
0	christmas "john lewis christmas" "john lewis" ...	7224515	55681
1	SNL "Saturday Night Live" "SNL Season 43" "Epi...	1053632	25561
2	Eminem "Walk" "On" "Water" "Aftermath/Shady/In...	17158579	787420
3	Salford City FC "Salford City" "Salford" "Clas...	27833	193
4	[none]	9815	30
...	
79860	aarons animals "aarons" "animals" "cat" "cats"...	1685609	38160
79861	[none]	1064798	60008
79862	I gave safiya nygaard a perfect hair makeover ...	1066451	48068
79863	Black Panther "HISHE" "Marvel" "Infinity War" ...	5660813	192957
79864	call of duty "cod" "activision" "Black Ops 4"	10306119	357079
	dislikes	comment_count \	
0	10247	9479	
1	2294	2757	
2	43420	125882	
3	12	37	
4	2	30	
...	
79860	1385	2657	
79861	382	3936	
79862	1032	3992	
79863	2846	13088	
79864	212976	144795	
	description	location \	
0	Click here to continue the story and make your...	Great Britain	
1	Musical guest Taylor Swift performs ...Ready for...	Great Britain	
2	Eminem's new track Walk on Water ft. Beyoncé i...	Great Britain	
3	Salford drew 4-4 against the Class of 92 and F...	Great Britain	
4	Dashcam captures truck's near miss with child ...	Great Britain	
...	
79860	The Cat Who Caught the Laser – Aaron's Animals	USA	

```

79861                               NaN      USA
79862 I had so much fun transforming Safiyas hair in...      USA
79863 How Black Panther Should Have EndedWatch More ...      USA
79864 Call of Duty: Black Ops 4 Multiplayer raises t...      USA

    trending_day_of_week day_of_week Engagement Metrics      score \
0                  Tuesday     Friday          75407  25928.732602
1                  Tuesday    Sunday          30612  12021.653022
2                  Tuesday     Friday         956722  389071.616394
3                  Tuesday    Monday           242   96.843978
4                  Tuesday    Monday            62   22.172624
...
79860                 ...       ...          ...        ...
79861                 ...       ...          ...        ...
79862                 ...       ...          ...        ...
79863                 ...       ...          ...        ...
79864                 ...       ...          ...        ...

    rank          new_text
0  24933.0  john lewis christmas ad mozthemonster
1  36735.0          taylor swift ready live snl
2  1782.0          eminem walk water audio ft beyonc
3  77301.0  goals salford city vs class friends peninsula ...
4  78623.0          dashcam captures trucks near miss child norway
...
79860  30213.0          ...          cat caught laser
79861  23409.0          ...          true facts ant mutualism
79862  26601.0          gave safiya nygaard perfect hair makeover base...
79863  9923.0          ...          black panther ended
79864  5863.0          official call duty black ops multiplayer revea...

```

[79865 rows x 18 columns]

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 get X
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, random_state=42)

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

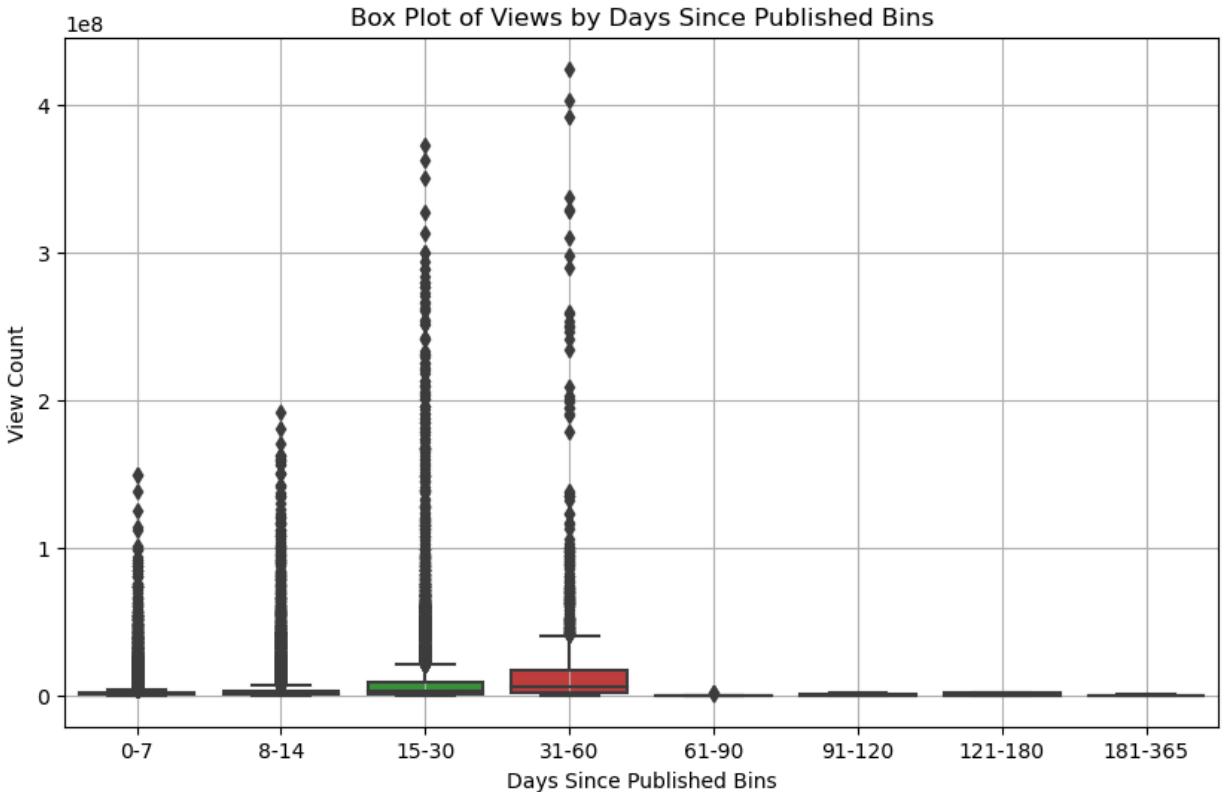
In [31]:

```
#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='%Y-%m-%d')

# Creating a new feature 'days_since_published'
train['days_since_published'] = (train['trending_date'] - train['publish_time']).dt.days

# 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', '181-365']
train['days_bins'] = pd.cut(train['days_since_published'], bins=bins, labels=labels)

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
```

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(get_sentiment)
train['title_sentiment'] = train['title'].fillna('').apply(get_sentiment)

# 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_sentiment)
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='black')
plt.title('Description Sentiment Polarity Distribution')
```

```

plt.xlabel('Sentiment Polarity')
plt.ylabel('Frequency')
plt.xticks([-1, 0, 1])

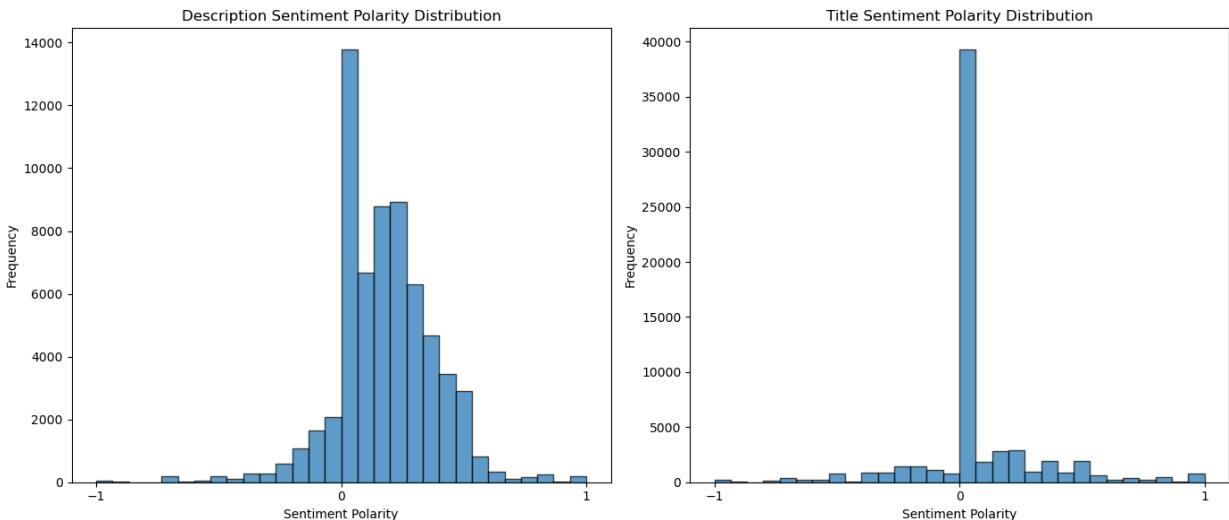
# Title Sentiment Histogram
plt.subplot(1, 2, 2)
plt.hist(train['title_sentiment'], bins=30, alpha=0.7, edgecolor='black')
plt.title('Title Sentiment Polarity Distribution')
plt.xlabel('Sentiment Polarity')
plt.ylabel('Frequency')
plt.xticks([-1, 0, 1])

plt.tight_layout()
plt.show()

```

Average Description Sentiment Score: 0.1716764242965884

Average Title Sentiment Score: 0.0477964529239135



Visualize the Sentiment Distribution 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'].apply(categorize_sentiment)
train['title_sentiment_category'] = train['title_sentiment'].apply(categorize_sentiment)

# 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

```

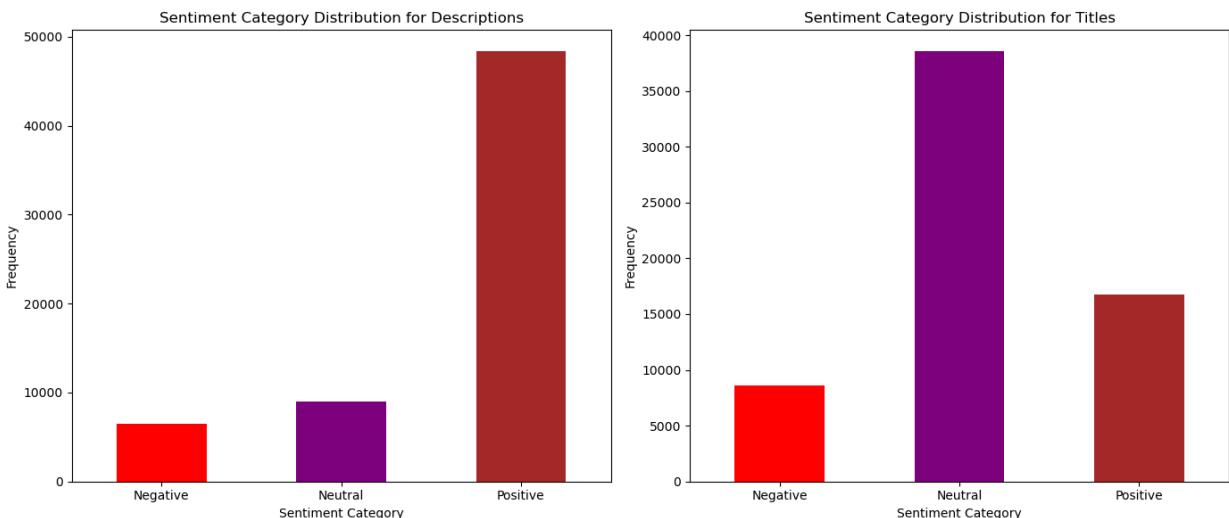
```

plt.subplot(1, 2, 1)
description_sentiment_counts = train['description_sentiment_category'].value_counts()
description_sentiment_counts.plot(kind='bar', color=['red', 'purple', 'brown'])
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_counts().reindex(['Negative', 'Neutral', 'Positive'])
title_sentiment_counts.plot(kind='bar', color=['red', 'purple', 'brown'])
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 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_words='english')

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

```
# Convert the sparse matrix into a DataFrame for easier manipulation
tfidf_df = pd.DataFrame(tfidf_matrix.toarray(), columns=tfidf_vectorizer.get_feature_names_out())

# 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 indices of top_n highest values
    top_features = [(features[i], row[i]) for i in top_indices] # Get the feature and its score
    return top_features

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

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

# Display the entire first 5 rows of the DataFrame including the top TF-IDF features
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 title \
23604 2018-03-14 Marshmello & Anne-Marie: Friends
25630 2018-03-24 Kirby Star Allies' Surprising HD Rumble Secret...
68698 2018-04-20 Stephen A.: Kevin Hart 'got his feelings hurt'...
39559 2017-11-17 How to be an Aquarius
62877 2018-03-16 Charlie Puth – Done For Me (feat. Kehlani) [Of...

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

publish_time tags \
\_
23604 2018-03-07 14:00:03 The Tonight Show|"Jimmy Fallon"|"Marshmello"|"...
25630 2018-03-16 04:00:01 Kirby|"Kirby Star Allies"|"Dedede"|"Meta Knight"|"...
68698 2018-04-17 14:55:31 espn|"dwyane wade"|"dwayne wade"|"d wade"|"76ers"|"...
39559 2017-11-15 13:29:28 Zodiac|"makeup"|"comedy"|"aquarius"
62877 2018-03-15 16:02:17 Charlie|"Puth"|"charlie puth"|"Charlie Puth – ...

likes dislikes comment_count \
23604 45011 1156 2365
25630 2716 52 450
68698 6829 537 1445
39559 5172 453 976
62877 84227 739 8663

description ... rank \
23604 Music guest Marshmello & Anne-Marie performs F... ... 27872.0
25630 Kirby Star Allies does something pretty fun wi... ... 67076.0
68698 First Take's Stephen A. Smith says Kevin Hart ... ... 57168.0
39559 Ya'll asked lol. What sign should I do next? D... ... 60832.0
62877 Download & Stream Done For Me (feat. Kehlani):... ... 18947.0

new_text views \
23604 marshmello annemarie friends 1443792
25630 kirby star allies surprising hd rumble secret ... 106398
68698 stephen kevin hart got feelings hurt dwyane wa... 976783
39559 aquarius 88644
62877 charlie puth done feat kehlani official audio 722009

days_since_published days_bins description_sentiment title_sentiment \
\_
23604 6 0-7 0.232292 0.000000
25630 7 0-7 0.137500 0.141667
68698 2 0-7 0.168333 0.250000
39559 1 0-7 0.400000 0.000000
62877 0 NaN 0.400000 0.000000

description_sentiment_category title_sentiment_category \
23604 Positive Neutral
25630 Positive Positive
68698 Positive Positive
39559 Positive Neutral
62877 Positive Neutral

top_tfidf_features
23604 [(jimmy, 0.6973441834478303), (nbc, 0.47929685...]
```

```
25630 [(patreon, 0.5185912254067347), (com, 0.407894...  
68698 [(http, 0.6127313163416526), (youtube, 0.33885...  
39559 [(ll, 0.5094338648331312), (don, 0.47521836732...  
62877 [(nhttp, 0.4915874738421169), (com, 0.44499439...  
[5 rows x 25 columns]
```

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_words='english')  
  
        # Fit and transform the text data to generate the TF-IDF matrix  
        tfidf_matrix = tfidf_vectorizer.fit_transform(text_data)  
  
        # Convert the sparse matrix into a DataFrame for easier manipulation  
        tfidf_df = pd.DataFrame(tfidf_matrix.toarray(), columns=tfidf_vectorizer.get_feature_names_out())  
  
        # 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 indices of top features  
            top_features = [(features[i], row[i]) for i in top_indices] # Get the top features  
            return top_features  
  
        # Apply the function to each row in the TF-IDF matrix  
        top_tfidf_features = [get_top_tfidf_features(row, tfidf_vectorizer.get_feature_names_out()) for row in tfidf_matrix.toarray()]  
  
        # Add the top TF-IDF features as a new column in the original DataFrame  
        train['top_tfidf_features'] = top_tfidf_features  
  
        # Display the entire first 5 rows of the DataFrame including the top Tf  
        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 title \
23604 2018-03-14 Marshmello & Anne-Marie: Friends
25630 2018-03-24 Kirby Star Allies' Surprising HD Rumble Secret...
68698 2018-04-20 Stephen A.: Kevin Hart 'got his feelings hurt'...
39559 2017-11-17 How to be an Aquarius
62877 2018-03-16 Charlie Puth – Done For Me (feat. Kehlani) [Of...

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

publish_time tags \
\
23604 2018-03-07 14:00:03 The Tonight Show|"Jimmy Fallon"|"Marshmello"|"...
25630 2018-03-16 04:00:01 Kirby|"Kirby Star Allies"|"Dedede"|"Meta Knight"|"...
68698 2018-04-17 14:55:31 espn|"dwyane wade"|"dwayne wade"|"d wade"|"76ers"|"...
39559 2017-11-15 13:29:28 Zodiac|"makeup"|"comedy"|"aquarius"|"...
62877 2018-03-15 16:02:17 Charlie|"Puth"|"charlie puth"|"Charlie Puth – ...

likes dislikes comment_count \
23604 45011 1156 2365
25630 2716 52 450
68698 6829 537 1445
39559 5172 453 976
62877 84227 739 8663

description ... rank \
23604 Music guest Marshmello & Anne-Marie performs F... ... 27872.0
25630 Kirby Star Allies does something pretty fun wi... ... 67076.0
68698 First Take's Stephen A. Smith says Kevin Hart ... ... 57168.0
39559 Ya'll asked lol. What sign should I do next? D... ... 60832.0
62877 Download & Stream Done For Me (feat. Kehlani):... ... 18947.0

new_text views \
23604 marshmello annemarie friends 1443792
25630 kirby star allies surprising hd rumble secret ... 106398
68698 stephen kevin hart got feelings hurt dwyane wa... 976783
39559 aquarius 88644
62877 charlie puth done feat kehlani official audio 722009

days_since_published days_bins description_sentiment title_sentiment \
\
23604 6 0-7 0.232292 0.000000
25630 7 0-7 0.137500 0.141667
68698 2 0-7 0.168333 0.250000
39559 1 0-7 0.400000 0.000000
62877 0 NaN 0.400000 0.000000

description_sentiment_category title_sentiment_category \
23604 Positive Neutral
25630 Positive Positive
68698 Positive Positive
39559 Positive Neutral
62877 Positive Neutral

top_tfidf_features
23604 [(nbc, 0.5956553280243344), (funny, 0.39716641...

```

```
25630 [(game, 0.7740105820359334), (review, 0.371423...  
68698 [(smith, 0.7615203128745558), (game, 0.4952950...  
39559 [(makeup, 0.7864204819004292), (comedy, 0.6176...  
62877 [(charlie, 0.988515653214018), (official, 0.11...
```

[5 rows x 25 columns]

Dimension Reduction-PCA

```
In [37]: from sklearn.preprocessing import StandardScaler  
from sklearn.decomposition import PCA  
  
non_numeric_cols = ['publish_time', 'title', 'channel_title', 'tags', 'description', 'comment_count', 'Engagement Metrics', 'score', 'rank', 'days_since_published', 'description_sentiment', 'title_sentiment']  
X_train_model = train.drop(columns=non_numeric_cols + ['views']).select_dtypes(exclude=[object])  
X_test_model = test.drop(columns=non_numeric_cols + ['views']).select_dtypes(exclude=[object])  
  
X_test_model = X_test_model.reindex(columns=X_train_model.columns, fill_value=0)  
  
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
rank	0
days_since_published	0
description_sentiment	0
title_sentiment	0

dtype: int64

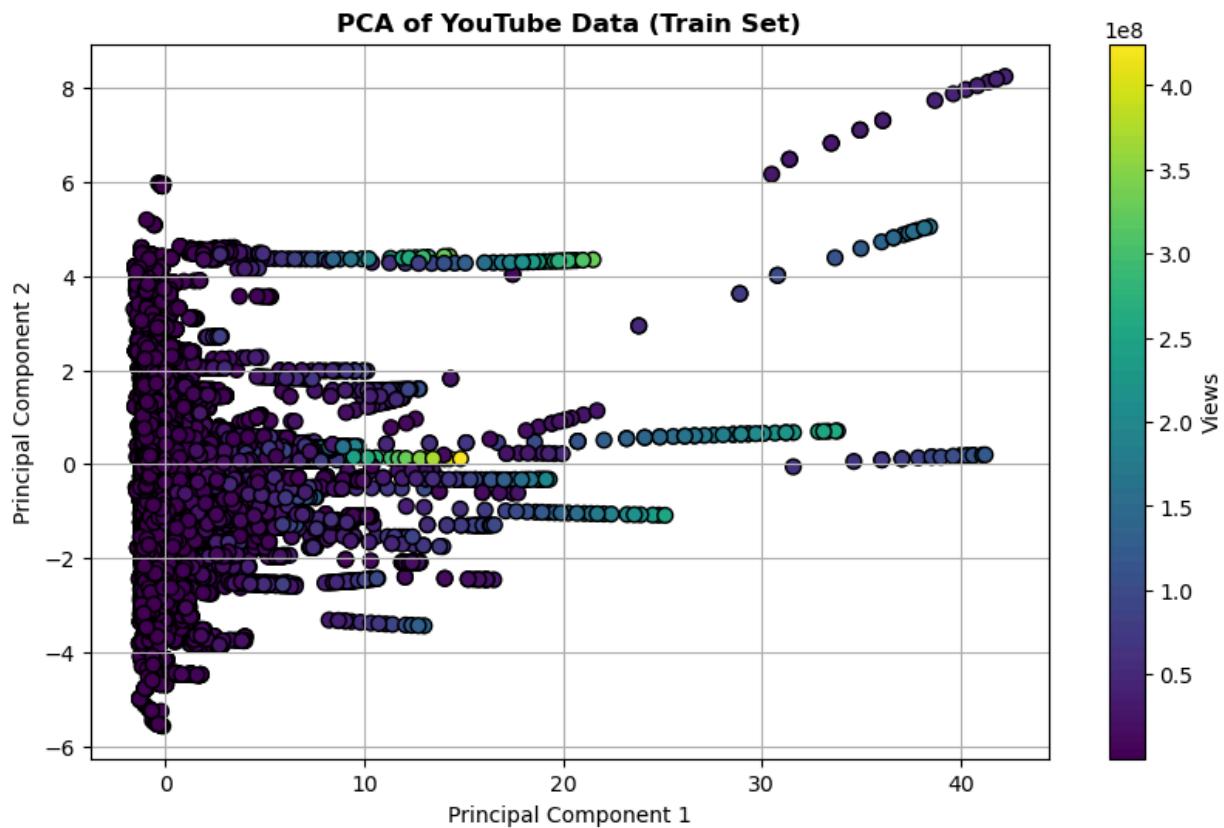
```
In [38]: # Apply PCA (Reduce to n components to capture 95% of variance)  
pca = PCA(n_components=0.95)  
X_train_pca = pca.fit_transform(X_train_scaled)  
X_test_pca = pca.transform(X_test_scaled)  
  
# Visualize the PCA results (Plot only the first two components)  
plt.figure(figsize=(10, 6))  
plt.scatter(X_train_pca[:, 0], X_train_pca[:, 1], c=y_train, cmap='viridis', edgecolor='black')
```

```

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%}")

```



Explained Variance per component:

PC1: 43.35%
 PC2: 12.56%
 PC3: 11.40%
 PC4: 10.22%
 PC5: 7.73%
 PC6: 7.47%
 PC7: 5.90%

Model Building

First Model-XGBoost

In [39]: # pip install xgboost

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)
```

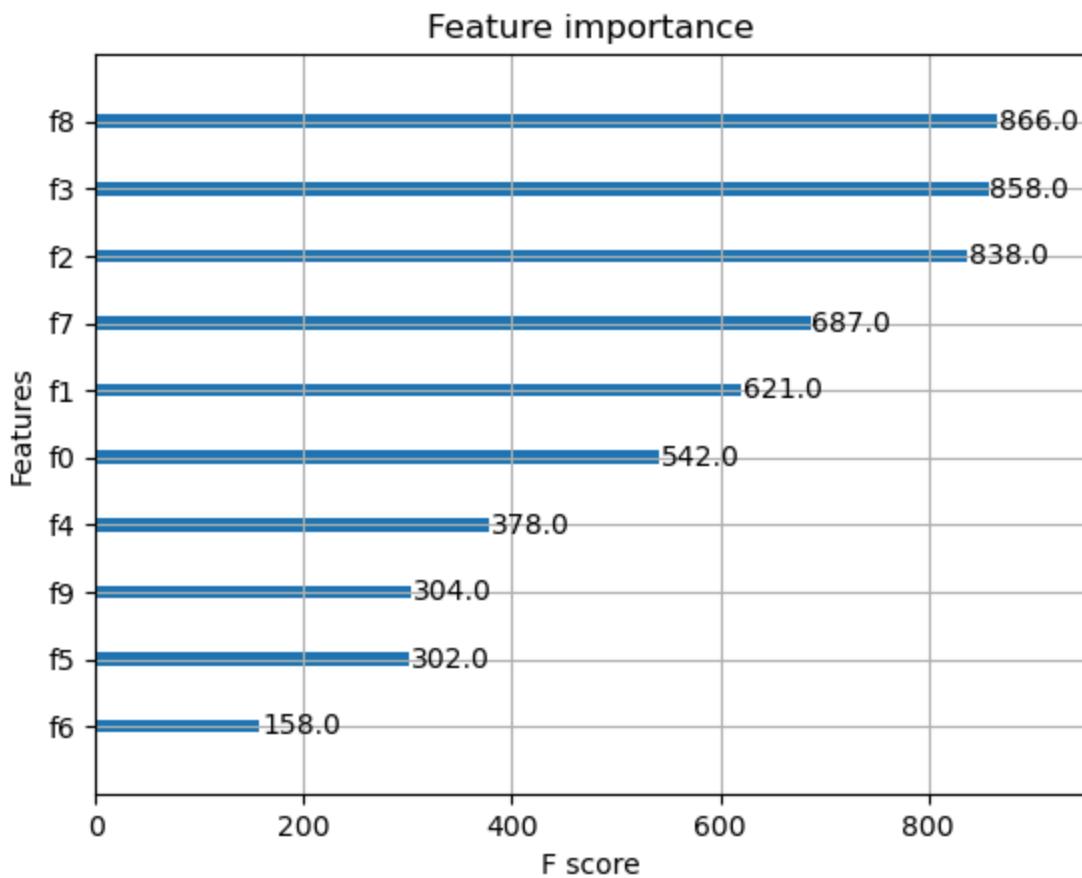
```
In [42]: # Predictions
y_pred = bst.predict(dtest)

# 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_test)
    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", "Train
```

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 R²:

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.

R² 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', 'location', '']
X_train.drop([col for col in text_columns if col in X_train.columns], axis=1, inplace=True)
X_test.drop([col for col in text_columns if col in X_test.columns], axis=1, inplace=True)

# Encode categorical features using one-hot encoding for consistency
categorical_columns = ['trending_day_of_week', 'day_of_week', 'days_bins', 'des']
X_train = pd.get_dummies(X_train, columns=[col for col in categorical_columns :])
X_test = pd.get_dummies(X_test, columns=[col for col in categorical_columns if col in X_test.columns])

# 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_Monday',
       'trending_day_of_week_Saturday', 'trending_day_of_week_Sunday',
       'trending_day_of_week_Thursday', 'trending_day_of_week_Tuesday',
       'trending_day_of_week_Wednesday', 'day_of_week_Monday',
       'day_of_week_Saturday', 'day_of_week_Sunday', 'day_of_week_Thursday',
       'day_of_week_Tuesday', 'day_of_week_Wednesday', 'trending_year',
       'trending_month', 'trending_day'],
      dtype='object')
X_test columns: Index(['category_id', 'likes', 'dislikes', 'comment_count',
       'Engagement Metrics', 'score', 'rank', 'trending_day_of_week_Monday',
       'trending_day_of_week_Saturday', 'trending_day_of_week_Sunday',
       'trending_day_of_week_Thursday', 'trending_day_of_week_Tuesday',
       'trending_day_of_week_Wednesday', 'day_of_week_Monday',
       'day_of_week_Saturday', 'day_of_week_Sunday', 'day_of_week_Thursday',
       'day_of_week_Tuesday', 'day_of_week_Wednesday', 'trending_year',
       '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, params)

    # 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]: # 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_depth': 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 R² (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. R²: Variation 2 has a Test R² of 0.9608, meaning it explains around 96% of the variance in the test data. This is the highest R² score among the variations, indicating good predictive power. Model Complexity and Generalization:

Variations 1 and 3 both have lower R² 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. R² assesses how well the model explains the variance in engagement metrics, giving insight into how much of the audience's behavior 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
trending_year            int32
trending_month           int32
trending_day              int32
dtype: object

```

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: ['trending_day_of_week_Tuesday', 'likes', 'trending_year', 'score', 'comment_count', 'trending_day_of_week_Sunday', 'trending_day_of_week_Thursday', 'dislikes', 'rank', 'day_of_week_Thursday', 'day_of_week_Sunday', 'category_id', 'trending_day', 'Engagement Metrics', 'trending_day_of_week_Wednesday', 'day_of_week_Tuesday', 'trending_month', 'trending_day_of_week_Monday', 'day_of_week_Saturday', 'day_of_week_Wednesday', 'trending_day_of_week_Saturday', 'day_of_week_Monday']

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

# 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 trending_date
test['days_to_trend'] = (test['trending_date'] - test['publish_time']).dt.days

# 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', 'views']
X_train_selected = train[selected_features]
X_test_selected = test[selected_features] # Ensure 'test' has the same features

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

```
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_test_encoded, y_test)
result_model2 = evaluate_model(rf_model2, X_train_encoded, y_train, X_test_encoded, y_test)
result_model3 = evaluate_model(rf_model3, X_train_encoded, y_train, X_test_encoded, y_test)

# 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_mse': 7429.233070650234, 'train_r2': 0.9670432435550833, 'test_r2': 0.8087028678543599}
Model 2 (n_estimators=200): {'train_mse': 1284.1807777766226, 'test_mse': 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': 0.7510068891411407}
```

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)', 'Model 3 (max_depth=10)'],
    'Train MSE': [result_model1['train_mse'], result_model2['train_mse'], result_model3['train_mse']],
    'Test MSE': [result_model1['test_mse'], result_model2['test_mse'], result_model3['test_mse']],
    'Train R22

```

Performance Comparison of Models:

	Model	Train MSE	Test MSE	Train R ²	Test R ²
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['Test MSE'].min()) &
    (performance_comparison['Test R2'] == performance_comparison['Test R2'].max())
]

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

Winning Model:

	Model	Train MSE	Test MSE	Train R ²	Test R ²
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}")
```

RMSE: 198.51277338889346
R² Score: -0.014708713421066388

```
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, X_train_p)

    # 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:.4f}")
    print(f"RMSE (Validation): {rmse_test:.4f}, R2 (Validation): {r2_test:.4f}")

results_df = pd.DataFrame(results)

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

```

```

Model: Variation 1
RMSE (Train): 0.2570, R2 (Train): 1.0000
RMSE (Validation): 198.6657, R2 (Validation): -0.0163

Model: Variation 2
RMSE (Train): 0.2588, R2 (Train): 1.0000
RMSE (Validation): 198.6690, R2 (Validation): -0.0163

Model: Variation 3
RMSE (Train): 0.2570, R2 (Train): 1.0000
RMSE (Validation): 198.6657, R2 (Validation): -0.0163

```

Summary of all variations:

	Model	RMSE (Train)	R ² (Train)	RMSE (Validation)	R ² (Validation)
0	Variation 1	0.257022	0.999998	198.665717	-0.016273
1	Variation 2	0.258780	0.999998	198.668958	-0.016306
2	Variation 3	0.257022	0.999998	198.665717	-0.016273

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 different hyperparameters
def build_and_train_model(hidden_layers, neurons, learning_rate, batch_size):
    # Build the model
    model = Sequential()
    model.add(Dense(neurons, input_dim=X_train_scaled.shape[1], activation='relu'))

    # 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', metrics=['mean_squared_error'])

    # Train the model
    model.fit(X_train_scaled, y_train, epochs=50, batch_size=batch_size, validation_split=0.2)

    return model

# Define the hyperparameter variations
variations = [
    {
        'hidden_layers': 1,
        'neurons': 10,
        'learning_rate': 0.001,
        'batch_size': 32
    },
    {
        'hidden_layers': 2,
        'neurons': 10,
        'learning_rate': 0.001,
        'batch_size': 32
    },
    {
        'hidden_layers': 3,
        'neurons': 10,
        'learning_rate': 0.001,
        'batch_size': 32
    },
    {
        'hidden_layers': 4,
        'neurons': 10,
        'learning_rate': 0.001,
        'batch_size': 32
    },
    {
        'hidden_layers': 5,
        'neurons': 10,
        'learning_rate': 0.001,
        'batch_size': 32
    },
    {
        'hidden_layers': 6,
        'neurons': 10,
        'learning_rate': 0.001,
        'batch_size': 32
    },
    {
        'hidden_layers': 7,
        'neurons': 10,
        'learning_rate': 0.001,
        'batch_size': 32
    },
    {
        'hidden_layers': 8,
        'neurons': 10,
        'learning_rate': 0.001,
        'batch_size': 32
    },
    {
        'hidden_layers': 9,
        'neurons': 10,
        'learning_rate': 0.001,
        'batch_size': 32
    },
    {
        'hidden_layers': 10,
        'neurons': 10,
        'learning_rate': 0.001,
        'batch_size': 32
    }
]

```

```

        {"hidden_layers": 1, "neurons": 32, "learning_rate": 0.001, "batch_size": 1000000},
        {"hidden_layers": 2, "neurons": 64, "learning_rate": 0.001, "batch_size": 1000000},
        {"hidden_layers": 3, "neurons": 128, "learning_rate": 0.0001, "batch_size": 1000000}
    ]

# 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)

```

2024-10-20 22:09:27.187378: I tensorflow/core/platform/cpu_feature_guard.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, in other operations, rebuild TensorFlow with the appropriate compiler flags.

```
Training Model Variation 1 with params: {'hidden_layers': 1, 'neurons': 32, 'learning_rate': 0.001, 'batch_size': 32}
Epoch 1/50
1997/1997 [=====] - 3s 1ms/step - loss: 39796.8086 - mean_squared_error: 39796.8086 - val_loss: 38804.3828 - val_mean_squared_error: 38804.3828
Epoch 2/50
1997/1997 [=====] - 3s 1ms/step - loss: 39533.0664 - mean_squared_error: 39533.0664 - val_loss: 38773.5977 - val_mean_squared_error: 38773.5977
Epoch 3/50
1997/1997 [=====] - 3s 1ms/step - loss: 39506.7188 - mean_squared_error: 39506.7188 - val_loss: 38759.2188 - val_mean_squared_error: 38759.2188
Epoch 4/50
1997/1997 [=====] - 3s 2ms/step - loss: 39491.6133 - mean_squared_error: 39491.6133 - val_loss: 38744.9727 - val_mean_squared_error: 38744.9727
Epoch 5/50
1997/1997 [=====] - 3s 1ms/step - loss: 39474.9805 - mean_squared_error: 39474.9805 - val_loss: 38733.4414 - val_mean_squared_error: 38733.4414
Epoch 6/50
1997/1997 [=====] - 3s 1ms/step - loss: 39457.3320 - mean_squared_error: 39457.3320 - val_loss: 38717.9883 - val_mean_squared_error: 38717.9883
Epoch 7/50
1997/1997 [=====] - 4s 2ms/step - loss: 39439.8398 - mean_squared_error: 39439.8398 - val_loss: 38698.0781 - val_mean_squared_error: 38698.0781
Epoch 8/50
1997/1997 [=====] - 4s 2ms/step - loss: 39425.5078 - mean_squared_error: 39425.5078 - val_loss: 38684.2227 - val_mean_squared_error: 38684.2227
Epoch 9/50
1997/1997 [=====] - 3s 2ms/step - loss: 39410.3984 - mean_squared_error: 39410.3984 - val_loss: 38669.3086 - val_mean_squared_error: 38669.3086
Epoch 10/50
1997/1997 [=====] - 3s 2ms/step - loss: 39396.6172 - mean_squared_error: 39396.6172 - val_loss: 38663.4141 - val_mean_squared_error: 38663.4141
Epoch 11/50
1997/1997 [=====] - 3s 2ms/step - loss: 39384.6914 - mean_squared_error: 39384.6914 - val_loss: 38650.8008 - val_mean_squared_error: 38650.8008
Epoch 12/50
1997/1997 [=====] - 3s 2ms/step - loss: 39373.7695 - mean_squared_error: 39373.7695 - val_loss: 38637.0117 - val_mean_squared_error: 38637.0117
Epoch 13/50
1997/1997 [=====] - 3s 1ms/step - loss: 39362.1992 - mean_squared_error: 39362.1992 - val_loss: 38628.6992 - val_mean_squared_error: 38628.6992
Epoch 14/50
1997/1997 [=====] - 3s 2ms/step - loss: 39351.4141 - mean_squared_error: 39351.4141 - val_loss: 38617.2617 - val_mean_squared_error: 38617.2617
Epoch 15/50
1997/1997 [=====] - 3s 1ms/step - loss: 39341.8984 -
```

```
mean_squared_error: 39341.8984 - val_loss: 38610.3711 - val_mean_squared_error: 38610.3711
Epoch 16/50
1997/1997 [=====] - 4s 2ms/step - loss: 39331.5547 - mean_squared_error: 39331.5547 - val_loss: 38601.7461 - val_mean_squared_error: 38601.7461
Epoch 17/50
1997/1997 [=====] - 4s 2ms/step - loss: 39325.1797 - mean_squared_error: 39325.1797 - val_loss: 38589.5391 - val_mean_squared_error: 38589.5391
Epoch 18/50
1997/1997 [=====] - 4s 2ms/step - loss: 39313.6055 - mean_squared_error: 39313.6055 - val_loss: 38582.9570 - val_mean_squared_error: 38582.9570
Epoch 19/50
1997/1997 [=====] - 3s 2ms/step - loss: 39306.5586 - mean_squared_error: 39306.5586 - val_loss: 38571.5898 - val_mean_squared_error: 38571.5898
Epoch 20/50
1997/1997 [=====] - 4s 2ms/step - loss: 39295.5234 - mean_squared_error: 39295.5234 - val_loss: 38577.3203 - val_mean_squared_error: 38577.3203
Epoch 21/50
1997/1997 [=====] - 4s 2ms/step - loss: 39289.0742 - mean_squared_error: 39289.0742 - val_loss: 38562.1953 - val_mean_squared_error: 38562.1953
Epoch 22/50
1997/1997 [=====] - 3s 2ms/step - loss: 39278.9688 - mean_squared_error: 39278.9688 - val_loss: 38559.9180 - val_mean_squared_error: 38559.9180
Epoch 23/50
1997/1997 [=====] - 3s 2ms/step - loss: 39269.0586 - mean_squared_error: 39269.0586 - val_loss: 38539.3984 - val_mean_squared_error: 38539.3984
Epoch 24/50
1997/1997 [=====] - 3s 2ms/step - loss: 39263.2695 - mean_squared_error: 39263.2695 - val_loss: 38532.0039 - val_mean_squared_error: 38532.0039
Epoch 25/50
1997/1997 [=====] - 3s 2ms/step - loss: 39250.2383 - mean_squared_error: 39250.2383 - val_loss: 38528.5820 - val_mean_squared_error: 38528.5820
Epoch 26/50
1997/1997 [=====] - 3s 2ms/step - loss: 39243.3281 - mean_squared_error: 39243.3281 - val_loss: 38525.7539 - val_mean_squared_error: 38525.7539
Epoch 27/50
1997/1997 [=====] - 4s 2ms/step - loss: 39233.5234 - mean_squared_error: 39233.5234 - val_loss: 38506.9922 - val_mean_squared_error: 38506.9922
Epoch 28/50
1997/1997 [=====] - 3s 2ms/step - loss: 39224.1328 - mean_squared_error: 39224.1328 - val_loss: 38503.3906 - val_mean_squared_error: 38503.3906
Epoch 29/50
1997/1997 [=====] - 3s 2ms/step - loss: 39213.0703 - mean_squared_error: 39213.0703 - val_loss: 38500.8359 - val_mean_squared_error: 38500.8359
Epoch 30/50
1997/1997 [=====] - 4s 2ms/step - loss: 39206.8125 -
```

```
mean_squared_error: 39206.8125 - val_loss: 38482.0117 - val_mean_squared_error: 38482.0117
Epoch 31/50
1997/1997 [=====] - 4s 2ms/step - loss: 39195.3789 - mean_squared_error: 39195.3789 - val_loss: 38468.7812 - val_mean_squared_error: 38468.7812
Epoch 32/50
1997/1997 [=====] - 3s 2ms/step - loss: 39185.0664 - mean_squared_error: 39185.0664 - val_loss: 38452.3555 - val_mean_squared_error: 38452.3555
Epoch 33/50
1997/1997 [=====] - 3s 2ms/step - loss: 39177.5352 - mean_squared_error: 39177.5352 - val_loss: 38467.9688 - val_mean_squared_error: 38467.9688
Epoch 34/50
1997/1997 [=====] - 3s 2ms/step - loss: 39168.6016 - mean_squared_error: 39168.6016 - val_loss: 38455.7461 - val_mean_squared_error: 38455.7461
Epoch 35/50
1997/1997 [=====] - 3s 2ms/step - loss: 39158.9844 - mean_squared_error: 39158.9844 - val_loss: 38436.0430 - val_mean_squared_error: 38436.0430
Epoch 36/50
1997/1997 [=====] - 3s 2ms/step - loss: 39150.0078 - mean_squared_error: 39150.0078 - val_loss: 38435.6133 - val_mean_squared_error: 38435.6133
Epoch 37/50
1997/1997 [=====] - 3s 2ms/step - loss: 39142.9492 - mean_squared_error: 39142.9492 - val_loss: 38427.7461 - val_mean_squared_error: 38427.7461
Epoch 38/50
1997/1997 [=====] - 3s 1ms/step - loss: 39129.5859 - mean_squared_error: 39129.5859 - val_loss: 38417.0156 - val_mean_squared_error: 38417.0156
Epoch 39/50
1997/1997 [=====] - 3s 2ms/step - loss: 39123.3086 - mean_squared_error: 39123.3086 - val_loss: 38405.2539 - val_mean_squared_error: 38405.2539
Epoch 40/50
1997/1997 [=====] - 3s 2ms/step - loss: 39113.7695 - mean_squared_error: 39113.7695 - val_loss: 38400.1914 - val_mean_squared_error: 38400.1914
Epoch 41/50
1997/1997 [=====] - 4s 2ms/step - loss: 39104.0977 - mean_squared_error: 39104.0977 - val_loss: 38398.4414 - val_mean_squared_error: 38398.4414
Epoch 42/50
1997/1997 [=====] - 4s 2ms/step - loss: 39094.6367 - mean_squared_error: 39094.6367 - val_loss: 38378.6875 - val_mean_squared_error: 38378.6875
Epoch 43/50
1997/1997 [=====] - 4s 2ms/step - loss: 39086.5820 - mean_squared_error: 39086.5820 - val_loss: 38368.6133 - val_mean_squared_error: 38368.6133
Epoch 44/50
1997/1997 [=====] - 4s 2ms/step - loss: 39078.7070 - mean_squared_error: 39078.7070 - val_loss: 38355.9258 - val_mean_squared_error: 38355.9258
Epoch 45/50
1997/1997 [=====] - 3s 2ms/step - loss: 39067.5781 -
```

```
mean_squared_error: 39067.5781 - val_loss: 38352.1055 - val_mean_squared_error: 38352.1055
Epoch 46/50
1997/1997 [=====] - 3s 2ms/step - loss: 39058.4922 - mean_squared_error: 39058.4922 - val_loss: 38346.4727 - val_mean_squared_error: 38346.4727
Epoch 47/50
1997/1997 [=====] - 3s 2ms/step - loss: 39046.0156 - mean_squared_error: 39046.0156 - val_loss: 38322.2227 - val_mean_squared_error: 38322.2227
Epoch 48/50
1997/1997 [=====] - 3s 1ms/step - loss: 39039.6953 - mean_squared_error: 39039.6953 - val_loss: 38323.5508 - val_mean_squared_error: 38323.5508
Epoch 49/50
1997/1997 [=====] - 3s 1ms/step - loss: 39028.5938 - mean_squared_error: 39028.5938 - val_loss: 38331.8281 - val_mean_squared_error: 38331.8281
Epoch 50/50
1997/1997 [=====] - 3s 2ms/step - loss: 39015.7070 - mean_squared_error: 39015.7070 - val_loss: 38301.1250 - val_mean_squared_error: 38301.1250
1997/1997 [=====] - 2s 937us/step
500/500 [=====] - 1s 2ms/step
Training Model Variation 2 with params: {'hidden_layers': 2, 'neurons': 64, 'learning_rate': 0.001, 'batch_size': 64}
Epoch 1/50
999/999 [=====] - 3s 2ms/step - loss: 39614.1836 - mean_squared_error: 39614.1836 - val_loss: 38743.6172 - val_mean_squared_error: 38743.6172
Epoch 2/50
999/999 [=====] - 2s 2ms/step - loss: 39486.3008 - mean_squared_error: 39486.3008 - val_loss: 38742.0664 - val_mean_squared_error: 38742.0664
Epoch 3/50
999/999 [=====] - 2s 2ms/step - loss: 39456.3711 - mean_squared_error: 39456.3711 - val_loss: 38711.8906 - val_mean_squared_error: 38711.8906
Epoch 4/50
999/999 [=====] - 2s 2ms/step - loss: 39416.8047 - mean_squared_error: 39416.8047 - val_loss: 38687.4844 - val_mean_squared_error: 38687.4844
Epoch 5/50
999/999 [=====] - 2s 2ms/step - loss: 39372.9258 - mean_squared_error: 39372.9258 - val_loss: 38627.5195 - val_mean_squared_error: 38627.5195
Epoch 6/50
999/999 [=====] - 2s 2ms/step - loss: 39324.6719 - mean_squared_error: 39324.6719 - val_loss: 38580.0664 - val_mean_squared_error: 38580.0664
Epoch 7/50
999/999 [=====] - 2s 2ms/step - loss: 39275.1484 - mean_squared_error: 39275.1484 - val_loss: 38530.0586 - val_mean_squared_error: 38530.0586
Epoch 8/50
999/999 [=====] - 2s 2ms/step - loss: 39221.8750 - mean_squared_error: 39221.8750 - val_loss: 38454.5664 - val_mean_squared_error: 38454.5664
Epoch 9/50
999/999 [=====] - 2s 2ms/step - loss: 39174.9922 - me
```

```
an_squared_error: 39174.9922 - val_loss: 38430.9180 - val_mean_squared_error:  
38430.9180  
Epoch 10/50  
999/999 [=====] - 2s 2ms/step - loss: 39122.3125 - me  
an_squared_error: 39122.3125 - val_loss: 38372.3242 - val_mean_squared_error:  
38372.3242  
Epoch 11/50  
999/999 [=====] - 2s 2ms/step - loss: 39059.9883 - me  
an_squared_error: 39059.9883 - val_loss: 38387.6523 - val_mean_squared_error:  
38387.6523  
Epoch 12/50  
999/999 [=====] - 2s 2ms/step - loss: 39014.7266 - me  
an_squared_error: 39014.7266 - val_loss: 38241.3945 - val_mean_squared_error:  
38241.3945  
Epoch 13/50  
999/999 [=====] - 2s 2ms/step - loss: 38959.1836 - me  
an_squared_error: 38959.1836 - val_loss: 38204.1953 - val_mean_squared_error:  
38204.1953  
Epoch 14/50  
999/999 [=====] - 2s 2ms/step - loss: 38889.8203 - me  
an_squared_error: 38889.8203 - val_loss: 38155.3203 - val_mean_squared_error:  
38155.3203  
Epoch 15/50  
999/999 [=====] - 2s 2ms/step - loss: 38871.4727 - me  
an_squared_error: 38871.4727 - val_loss: 38168.3867 - val_mean_squared_error:  
38168.3867  
Epoch 16/50  
999/999 [=====] - 2s 2ms/step - loss: 38815.2031 - me  
an_squared_error: 38815.2031 - val_loss: 38058.1016 - val_mean_squared_error:  
38058.1016  
Epoch 17/50  
999/999 [=====] - 2s 2ms/step - loss: 38775.1406 - me  
an_squared_error: 38775.1406 - val_loss: 37989.2617 - val_mean_squared_error:  
37989.2617  
Epoch 18/50  
999/999 [=====] - 2s 2ms/step - loss: 38701.9414 - me  
an_squared_error: 38701.9414 - val_loss: 38082.5859 - val_mean_squared_error:  
38082.5859  
Epoch 19/50  
999/999 [=====] - 2s 2ms/step - loss: 38676.5469 - me  
an_squared_error: 38676.5469 - val_loss: 38031.2969 - val_mean_squared_error:  
38031.2969  
Epoch 20/50  
999/999 [=====] - 2s 2ms/step - loss: 38627.8359 - me  
an_squared_error: 38627.8359 - val_loss: 38030.2422 - val_mean_squared_error:  
38030.2422  
Epoch 21/50  
999/999 [=====] - 2s 2ms/step - loss: 38611.5742 - me  
an_squared_error: 38611.5742 - val_loss: 38099.0273 - val_mean_squared_error:  
38099.0273  
Epoch 22/50  
999/999 [=====] - 2s 2ms/step - loss: 38545.4141 - me  
an_squared_error: 38545.4141 - val_loss: 38015.9570 - val_mean_squared_error:  
38015.9570  
Epoch 23/50  
999/999 [=====] - 2s 2ms/step - loss: 38514.4102 - me  
an_squared_error: 38514.4102 - val_loss: 37818.4531 - val_mean_squared_error:  
37818.4531  
Epoch 24/50  
999/999 [=====] - 2s 2ms/step - loss: 38488.9531 - me
```

```
an_squared_error: 38488.9531 - val_loss: 37879.4492 - val_mean_squared_error:  
37879.4492  
Epoch 25/50  
999/999 [=====] - 2s 2ms/step - loss: 38480.0703 - me  
an_squared_error: 38480.0703 - val_loss: 37949.2969 - val_mean_squared_error:  
37949.2969  
Epoch 26/50  
999/999 [=====] - 2s 2ms/step - loss: 38436.1133 - me  
an_squared_error: 38436.1133 - val_loss: 37828.6289 - val_mean_squared_error:  
37828.6289  
Epoch 27/50  
999/999 [=====] - 2s 2ms/step - loss: 38456.6680 - me  
an_squared_error: 38456.6680 - val_loss: 37864.1523 - val_mean_squared_error:  
37864.1523  
Epoch 28/50  
999/999 [=====] - 2s 2ms/step - loss: 38403.0312 - me  
an_squared_error: 38403.0312 - val_loss: 37702.7852 - val_mean_squared_error:  
37702.7852  
Epoch 29/50  
999/999 [=====] - 2s 2ms/step - loss: 38369.6016 - me  
an_squared_error: 38369.6016 - val_loss: 37560.6797 - val_mean_squared_error:  
37560.6797  
Epoch 30/50  
999/999 [=====] - 2s 2ms/step - loss: 38361.4844 - me  
an_squared_error: 38361.4844 - val_loss: 37530.6758 - val_mean_squared_error:  
37530.6758  
Epoch 31/50  
999/999 [=====] - 2s 2ms/step - loss: 38347.6875 - me  
an_squared_error: 38347.6875 - val_loss: 37828.0430 - val_mean_squared_error:  
37828.0430  
Epoch 32/50  
999/999 [=====] - 2s 2ms/step - loss: 38328.2070 - me  
an_squared_error: 38328.2070 - val_loss: 37522.1172 - val_mean_squared_error:  
37522.1172  
Epoch 33/50  
999/999 [=====] - 2s 2ms/step - loss: 38272.3516 - me  
an_squared_error: 38272.3516 - val_loss: 37852.4766 - val_mean_squared_error:  
37852.4766  
Epoch 34/50  
999/999 [=====] - 2s 2ms/step - loss: 38323.6094 - me  
an_squared_error: 38323.6094 - val_loss: 37712.8633 - val_mean_squared_error:  
37712.8633  
Epoch 35/50  
999/999 [=====] - 2s 2ms/step - loss: 38293.7305 - me  
an_squared_error: 38293.7305 - val_loss: 37733.8438 - val_mean_squared_error:  
37733.8438  
Epoch 36/50  
999/999 [=====] - 2s 2ms/step - loss: 38224.1211 - me  
an_squared_error: 38224.1211 - val_loss: 37657.0742 - val_mean_squared_error:  
37657.0742  
Epoch 37/50  
999/999 [=====] - 2s 2ms/step - loss: 38213.1602 - me  
an_squared_error: 38213.1602 - val_loss: 37520.1484 - val_mean_squared_error:  
37520.1484  
Epoch 38/50  
999/999 [=====] - 2s 2ms/step - loss: 38204.1055 - me  
an_squared_error: 38204.1055 - val_loss: 37493.5820 - val_mean_squared_error:  
37493.5820  
Epoch 39/50  
999/999 [=====] - 2s 2ms/step - loss: 38225.0781 - me
```

```
an_squared_error: 38225.0781 - val_loss: 37597.0859 - val_mean_squared_error:  
37597.0859  
Epoch 40/50  
999/999 [=====] - 2s 2ms/step - loss: 38194.0703 - me  
an_squared_error: 38194.0703 - val_loss: 37544.7422 - val_mean_squared_error:  
37544.7422  
Epoch 41/50  
999/999 [=====] - 2s 2ms/step - loss: 38179.0195 - me  
an_squared_error: 38179.0195 - val_loss: 37676.1328 - val_mean_squared_error:  
37676.1328  
Epoch 42/50  
999/999 [=====] - 2s 2ms/step - loss: 38165.5977 - me  
an_squared_error: 38165.5977 - val_loss: 37362.5078 - val_mean_squared_error:  
37362.5078  
Epoch 43/50  
999/999 [=====] - 2s 2ms/step - loss: 38153.7188 - me  
an_squared_error: 38153.7188 - val_loss: 37463.6602 - val_mean_squared_error:  
37463.6602  
Epoch 44/50  
999/999 [=====] - 2s 2ms/step - loss: 38108.2930 - me  
an_squared_error: 38108.2930 - val_loss: 37772.8750 - val_mean_squared_error:  
37772.8750  
Epoch 45/50  
999/999 [=====] - 2s 2ms/step - loss: 38153.2461 - me  
an_squared_error: 38153.2461 - val_loss: 37361.6914 - val_mean_squared_error:  
37361.6914  
Epoch 46/50  
999/999 [=====] - 2s 2ms/step - loss: 38063.3281 - me  
an_squared_error: 38063.3281 - val_loss: 37314.0352 - val_mean_squared_error:  
37314.0352  
Epoch 47/50  
999/999 [=====] - 2s 2ms/step - loss: 38127.7578 - me  
an_squared_error: 38127.7578 - val_loss: 37501.7070 - val_mean_squared_error:  
37501.7070  
Epoch 48/50  
999/999 [=====] - 2s 2ms/step - loss: 38112.3945 - me  
an_squared_error: 38112.3945 - val_loss: 37289.7305 - val_mean_squared_error:  
37289.7305  
Epoch 49/50  
999/999 [=====] - 2s 2ms/step - loss: 38099.5508 - me  
an_squared_error: 38099.5508 - val_loss: 37278.9688 - val_mean_squared_error:  
37278.9688  
Epoch 50/50  
999/999 [=====] - 2s 2ms/step - loss: 38103.5664 - me  
an_squared_error: 38103.5664 - val_loss: 37439.4609 - val_mean_squared_error:  
37439.4609  
1997/1997 [=====] - 2s 1ms/step  
500/500 [=====] - 1s 1ms/step  
Training Model Variation 3 with params: {'hidden_layers': 3, 'neurons': 128,  
'learning_rate': 0.0001, 'batch_size': 128}  
Epoch 1/50  
500/500 [=====] - 2s 3ms/step - loss: 39885.8555 - me  
an_squared_error: 39885.8555 - val_loss: 38829.3828 - val_mean_squared_error:  
38829.3828  
Epoch 2/50  
500/500 [=====] - 1s 3ms/step - loss: 39545.4492 - me  
an_squared_error: 39545.4492 - val_loss: 38792.3203 - val_mean_squared_error:  
38792.3203  
Epoch 3/50  
500/500 [=====] - 1s 3ms/step - loss: 39522.6445 - me
```

```
an_squared_error: 39522.6445 - val_loss: 38776.1836 - val_mean_squared_error:  
38776.1836  
Epoch 4/50  
500/500 [=====] - 1s 3ms/step - loss: 39506.0781 - me  
an_squared_error: 39506.0781 - val_loss: 38761.3242 - val_mean_squared_error:  
38761.3242  
Epoch 5/50  
500/500 [=====] - 1s 3ms/step - loss: 39498.5469 - me  
an_squared_error: 39498.5469 - val_loss: 38760.3516 - val_mean_squared_error:  
38760.3516  
Epoch 6/50  
500/500 [=====] - 2s 3ms/step - loss: 39491.6445 - me  
an_squared_error: 39491.6445 - val_loss: 38744.5703 - val_mean_squared_error:  
38744.5703  
Epoch 7/50  
500/500 [=====] - 1s 2ms/step - loss: 39478.1016 - me  
an_squared_error: 39478.1016 - val_loss: 38740.9102 - val_mean_squared_error:  
38740.9102  
Epoch 8/50  
500/500 [=====] - 1s 3ms/step - loss: 39470.2109 - me  
an_squared_error: 39470.2109 - val_loss: 38726.9648 - val_mean_squared_error:  
38726.9648  
Epoch 9/50  
500/500 [=====] - 1s 3ms/step - loss: 39459.6016 - me  
an_squared_error: 39459.6016 - val_loss: 38714.8516 - val_mean_squared_error:  
38714.8516  
Epoch 10/50  
500/500 [=====] - 1s 3ms/step - loss: 39449.3984 - me  
an_squared_error: 39449.3984 - val_loss: 38709.2305 - val_mean_squared_error:  
38709.2305  
Epoch 11/50  
500/500 [=====] - 1s 3ms/step - loss: 39437.2656 - me  
an_squared_error: 39437.2656 - val_loss: 38693.1797 - val_mean_squared_error:  
38693.1797  
Epoch 12/50  
500/500 [=====] - 1s 3ms/step - loss: 39423.2109 - me  
an_squared_error: 39423.2109 - val_loss: 38677.7148 - val_mean_squared_error:  
38677.7148  
Epoch 13/50  
500/500 [=====] - 2s 3ms/step - loss: 39414.4609 - me  
an_squared_error: 39414.4609 - val_loss: 38668.3008 - val_mean_squared_error:  
38668.3008  
Epoch 14/50  
500/500 [=====] - 2s 3ms/step - loss: 39398.0938 - me  
an_squared_error: 39398.0938 - val_loss: 38660.0508 - val_mean_squared_error:  
38660.0508  
Epoch 15/50  
500/500 [=====] - 2s 3ms/step - loss: 39384.0664 - me  
an_squared_error: 39384.0664 - val_loss: 38648.1055 - val_mean_squared_error:  
38648.1055  
Epoch 16/50  
500/500 [=====] - 1s 3ms/step - loss: 39370.5625 - me  
an_squared_error: 39370.5625 - val_loss: 38629.7891 - val_mean_squared_error:  
38629.7891  
Epoch 17/50  
500/500 [=====] - 1s 2ms/step - loss: 39352.2539 - me  
an_squared_error: 39352.2539 - val_loss: 38617.2617 - val_mean_squared_error:  
38617.2617  
Epoch 18/50  
500/500 [=====] - 1s 3ms/step - loss: 39340.8398 - me
```

```
an_squared_error: 39340.8398 - val_loss: 38603.4922 - val_mean_squared_error:  
38603.4922  
Epoch 19/50  
500/500 [=====] - 2s 3ms/step - loss: 39323.3906 - me  
an_squared_error: 39323.3906 - val_loss: 38596.8281 - val_mean_squared_error:  
38596.8281  
Epoch 20/50  
500/500 [=====] - 1s 3ms/step - loss: 39306.0586 - me  
an_squared_error: 39306.0586 - val_loss: 38568.1719 - val_mean_squared_error:  
38568.1719  
Epoch 21/50  
500/500 [=====] - 1s 3ms/step - loss: 39289.2852 - me  
an_squared_error: 39289.2852 - val_loss: 38553.1680 - val_mean_squared_error:  
38553.1680  
Epoch 22/50  
500/500 [=====] - 1s 3ms/step - loss: 39269.4414 - me  
an_squared_error: 39269.4414 - val_loss: 38540.1719 - val_mean_squared_error:  
38540.1719  
Epoch 23/50  
500/500 [=====] - 1s 3ms/step - loss: 39248.2539 - me  
an_squared_error: 39248.2539 - val_loss: 38524.4062 - val_mean_squared_error:  
38524.4062  
Epoch 24/50  
500/500 [=====] - 1s 3ms/step - loss: 39235.1289 - me  
an_squared_error: 39235.1289 - val_loss: 38511.4492 - val_mean_squared_error:  
38511.4492  
Epoch 25/50  
500/500 [=====] - 1s 2ms/step - loss: 39210.1484 - me  
an_squared_error: 39210.1484 - val_loss: 38486.4023 - val_mean_squared_error:  
38486.4023  
Epoch 26/50  
500/500 [=====] - 1s 3ms/step - loss: 39186.8594 - me  
an_squared_error: 39186.8594 - val_loss: 38467.4180 - val_mean_squared_error:  
38467.4180  
Epoch 27/50  
500/500 [=====] - 2s 4ms/step - loss: 39173.7930 - me  
an_squared_error: 39173.7930 - val_loss: 38434.6211 - val_mean_squared_error:  
38434.6211  
Epoch 28/50  
500/500 [=====] - 2s 3ms/step - loss: 39145.6133 - me  
an_squared_error: 39145.6133 - val_loss: 38416.8086 - val_mean_squared_error:  
38416.8086  
Epoch 29/50  
500/500 [=====] - 1s 3ms/step - loss: 39131.1484 - me  
an_squared_error: 39131.1484 - val_loss: 38410.0078 - val_mean_squared_error:  
38410.0078  
Epoch 30/50  
500/500 [=====] - 2s 3ms/step - loss: 39107.8242 - me  
an_squared_error: 39107.8242 - val_loss: 38413.7695 - val_mean_squared_error:  
38413.7695  
Epoch 31/50  
500/500 [=====] - 2s 3ms/step - loss: 39093.8633 - me  
an_squared_error: 39093.8633 - val_loss: 38371.0625 - val_mean_squared_error:  
38371.0625  
Epoch 32/50  
500/500 [=====] - 2s 3ms/step - loss: 39072.8789 - me  
an_squared_error: 39072.8789 - val_loss: 38364.8984 - val_mean_squared_error:  
38364.8984  
Epoch 33/50  
500/500 [=====] - 2s 3ms/step - loss: 39052.6992 - me
```

```
an_squared_error: 39052.6992 - val_loss: 38332.1523 - val_mean_squared_error:  
38332.1523  
Epoch 34/50  
500/500 [=====] - 1s 3ms/step - loss: 39048.2969 - me  
an_squared_error: 39048.2969 - val_loss: 38306.8984 - val_mean_squared_error:  
38306.8984  
Epoch 35/50  
500/500 [=====] - 1s 3ms/step - loss: 39005.2617 - me  
an_squared_error: 39005.2617 - val_loss: 38330.6406 - val_mean_squared_error:  
38330.6406  
Epoch 36/50  
500/500 [=====] - 2s 3ms/step - loss: 39002.0898 - me  
an_squared_error: 39002.0898 - val_loss: 38284.7461 - val_mean_squared_error:  
38284.7461  
Epoch 37/50  
500/500 [=====] - 1s 3ms/step - loss: 38984.5625 - me  
an_squared_error: 38984.5625 - val_loss: 38264.1094 - val_mean_squared_error:  
38264.1094  
Epoch 38/50  
500/500 [=====] - 2s 3ms/step - loss: 38960.2500 - me  
an_squared_error: 38960.2500 - val_loss: 38255.1680 - val_mean_squared_error:  
38255.1680  
Epoch 39/50  
500/500 [=====] - 2s 4ms/step - loss: 38942.8555 - me  
an_squared_error: 38942.8555 - val_loss: 38229.9062 - val_mean_squared_error:  
38229.9062  
Epoch 40/50  
500/500 [=====] - 2s 4ms/step - loss: 38920.8711 - me  
an_squared_error: 38920.8711 - val_loss: 38222.3320 - val_mean_squared_error:  
38222.3320  
Epoch 41/50  
500/500 [=====] - 2s 4ms/step - loss: 38900.5156 - me  
an_squared_error: 38900.5156 - val_loss: 38225.2109 - val_mean_squared_error:  
38225.2109  
Epoch 42/50  
500/500 [=====] - 2s 4ms/step - loss: 38894.9609 - me  
an_squared_error: 38894.9609 - val_loss: 38184.5625 - val_mean_squared_error:  
38184.5625  
Epoch 43/50  
500/500 [=====] - 2s 4ms/step - loss: 38858.2500 - me  
an_squared_error: 38858.2500 - val_loss: 38185.0586 - val_mean_squared_error:  
38185.0586  
Epoch 44/50  
500/500 [=====] - 2s 4ms/step - loss: 38860.2734 - me  
an_squared_error: 38860.2734 - val_loss: 38152.2891 - val_mean_squared_error:  
38152.2891  
Epoch 45/50  
500/500 [=====] - 2s 4ms/step - loss: 38844.8867 - me  
an_squared_error: 38844.8867 - val_loss: 38121.5508 - val_mean_squared_error:  
38121.5508  
Epoch 46/50  
500/500 [=====] - 2s 4ms/step - loss: 38830.1562 - me  
an_squared_error: 38830.1562 - val_loss: 38131.2344 - val_mean_squared_error:  
38131.2344  
Epoch 47/50  
500/500 [=====] - 2s 4ms/step - loss: 38807.4023 - me  
an_squared_error: 38807.4023 - val_loss: 38069.9648 - val_mean_squared_error:  
38069.9648  
Epoch 48/50  
500/500 [=====] - 2s 4ms/step - loss: 38785.3477 - me
```

```

an_squared_error: 38785.3477 - val_loss: 38054.4531 - val_mean_squared_error:
38054.4531
Epoch 49/50
500/500 [=====] - 2s 4ms/step - loss: 38775.8086 - me
an_squared_error: 38775.8086 - val_loss: 38107.9141 - val_mean_squared_error:
38107.9141
Epoch 50/50
500/500 [=====] - 2s 4ms/step - loss: 38758.1758 - me
an_squared_error: 38758.1758 - val_loss: 38029.2656 - val_mean_squared_error:
38029.2656
1997/1997 [=====] - 2s 1ms/step
500/500 [=====] - 1s 1ms/step
      Variation  Hidden Layers  Neurons per Layer  Learning Rate  Batch Size \
0    Model 1            1                  32        0.0010          32
1    Model 2            2                  64        0.0010          64
2    Model 3            3                 128        0.0001         128

      Train MSE     Test MSE  Train R2  Test R2
0  39010.192642  38301.106301  0.014690  0.013775
1  38045.479874  37439.457750  0.039056  0.035962
2  38721.602037  38029.264859  0.021979  0.020775

```

```

In [65]: # Create a DataFrame to store the results
results = {
    "Variation": ["Variation 1 (32 Neurons)", "Variation 2 (64 Neurons)", "Variation 3 (128 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 R ²	Validation R ²
0	Variation 1 (32 Neurons)	38195.26	37291.30	0.0397	0.0302
1	Variation 2 (64 Neurons)	37651.89	36520.55	0.0473	0.0358
2	Variation 3 (128 Neurons)	37213.45	37321.47	0.0551	0.0319

Best Model:

Variation	Variation 2 (64 Neurons)
Train MSE	37651.89
Validation MSE	36520.55
Train R ²	0.0473
Validation R ²	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 RMSE", "Train R^2", "Test R^2"])
```

```
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' columns
tfidf_vectorizer_desc = TfidfVectorizer(max_features=100, stop_words='english')
tfidf_vectorizer_tags = TfidfVectorizer(max_features=100, stop_words='english')

# Fit and transform on the training set, and transform on the test set
tfidf_matrix_desc_train = tfidf_vectorizer_desc.fit_transform(train['description'])
tfidf_matrix_desc_test = tfidf_vectorizer_desc.transform(test['description'])

tfidf_matrix_tags_train = tfidf_vectorizer_tags.fit_transform(train['tags'])
tfidf_matrix_tags_test = tfidf_vectorizer_tags.transform(test['tags'])

# Add prefixes to avoid duplicate feature names
desc_feature_names = [f"desc_{name}" for name in tfidf_vectorizer_desc.get_feature_names_out()]
tags_feature_names = [f"tags_{name}" for name in tfidf_vectorizer_tags.get_feature_names_out()]

# Convert the sparse TF-IDF matrices into DataFrames with prefixed columns
tfidf_df_desc_train = pd.DataFrame(tfidf_matrix_desc_train.toarray(), columns=desc_feature_names)
tfidf_df_desc_test = pd.DataFrame(tfidf_matrix_desc_test.toarray(), columns=desc_feature_names)

tfidf_df_tags_train = pd.DataFrame(tfidf_matrix_tags_train.toarray(), columns=tags_feature_names)
tfidf_df_tags_test = pd.DataFrame(tfidf_matrix_tags_test.toarray(), columns=tags_feature_names)

# Concatenate TF-IDF features to the original X_train and X_test
X_train = pd.concat([X_train.reset_index(drop=True), tfidf_df_desc_train, tfidf_df_tags_train], axis=1)
X_test = pd.concat([X_test.reset_index(drop=True), tfidf_df_desc_test, tfidf_df_tags_test], axis=1)

# 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_Monday', 'trending_day_of_week_Saturday', 'trending_day_of_week_Sunday', ..., 'tags_video', 'tags_videos', 'tags_vlog', 'tags_voice', 'tags_vs', 'tags_war', 'tags_wars', 'tags_world', 'tags_wwe', 'tags_youtube'], dtype='object', length=222)

X_test columns: Index(['category_id', 'likes', 'dislikes', 'comment_count', 'Engagement Metrics', 'score', 'rank', 'trending_day_of_week_Monday', 'trending_day_of_week_Saturday', 'trending_day_of_week_Sunday', ..., 'tags_video', 'tags_videos', 'tags_vlog', 'tags_voice', 'tags_vs', 'tags_war', 'tags_wars', 'tags_world', 'tags_wwe', 'tags_youtube'], dtype='object', length=222)

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_rmse
        "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, params)

    # 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 [ ]: 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 BERT embeddings"):
        batch = texts[i:i+batch_size]
        inputs = tokenizer(batch, return_tensors='pt', padding=True, truncation=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().numpy()
        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!")
```

Processing train set...

Processing title...

Generating BERT embeddings: 0% | 0/1997 [00:00<?, ?it/s]