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-Great Britain & United States
df_gb = load_csv_from_zip(os.path.join(datasets_path, 'GBvideos.csv.zip'
df_us = load_csv_from_zip(os.path.join(datasets_path, 'USvideos.csv.zip'


# Merge 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())
```

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

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
dtype: int64
```

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

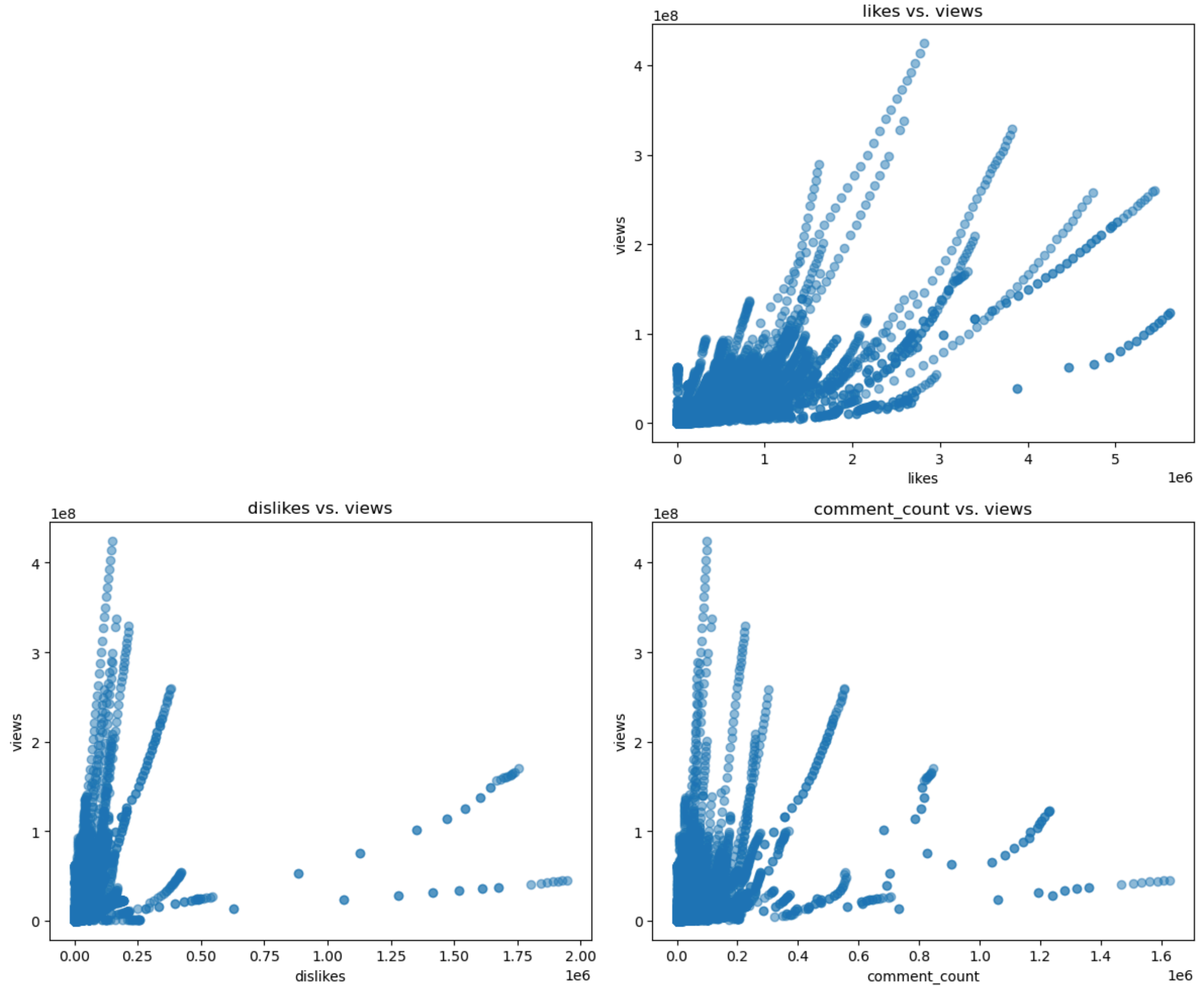
In [4]:
```
# Check for missing values in the merged DataFrame
print("Missing values")
print(df.isnull().sum())
```

```
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
dtype: int64
```

Exploratory Data Analysis (EDA)

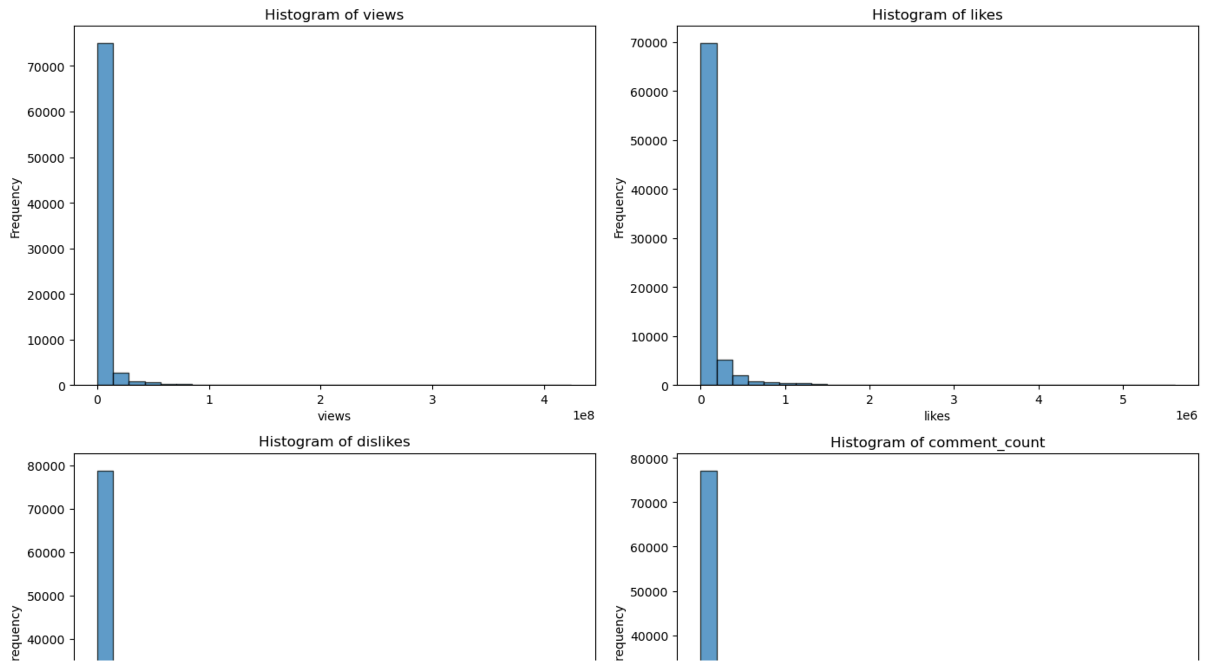
Scatter Plot of Numerical Variables

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

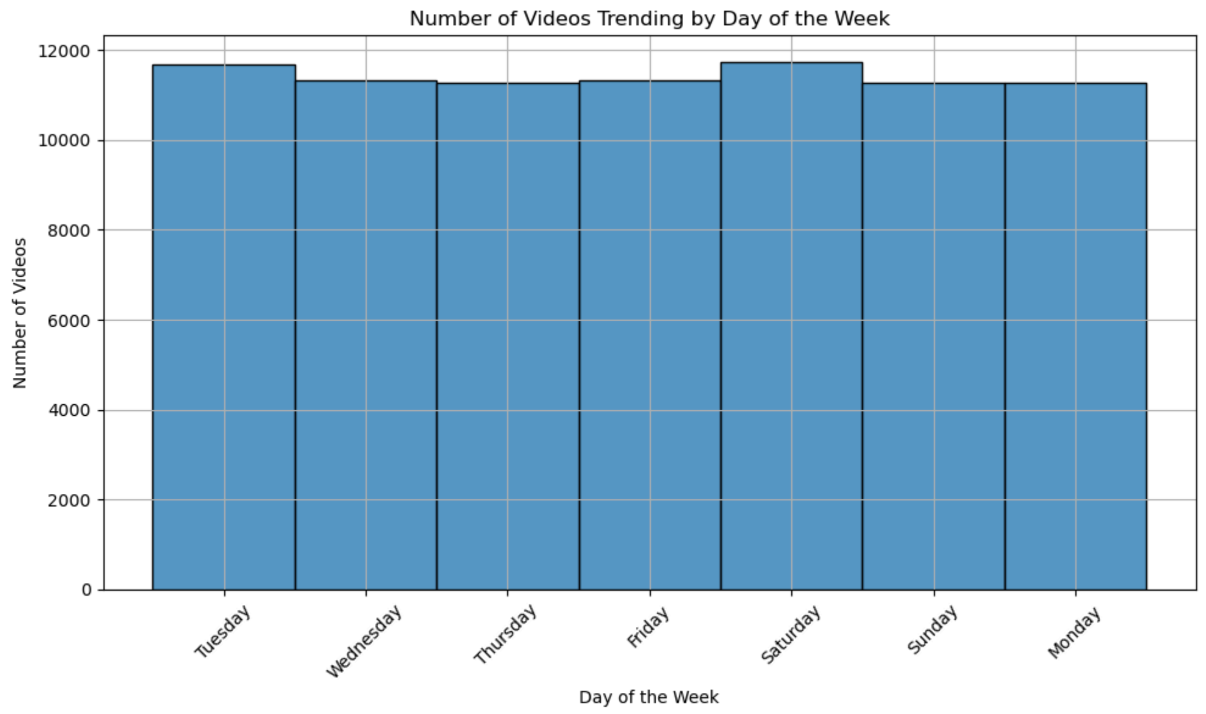
In [6]:
```
#data exploration for numerical columns
import matplotlib.pyplot as plt

# 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 Date-Time Variables

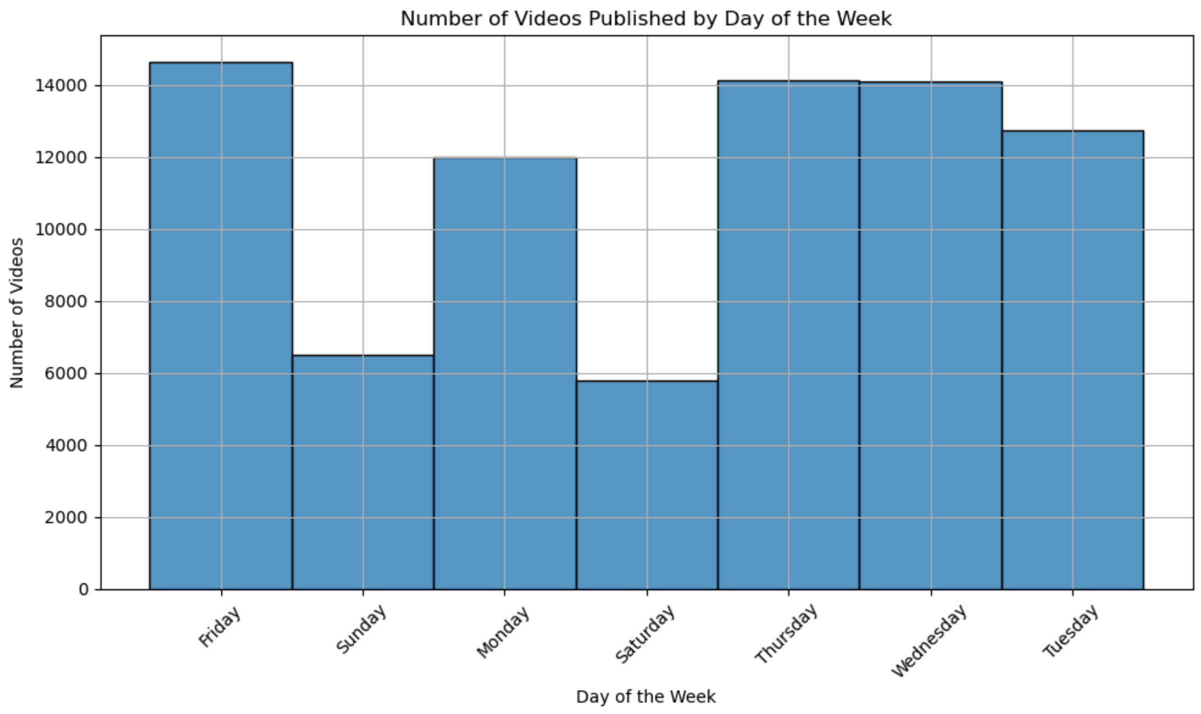
In [7]:
```
# convert the trending_date to datetime type
merged_df['trending_date'] = pd.to_datetime(merged_df['trending_date'],
# Extract day of the week from 'trending_date'
merged_df['trending_day_of_week'] = merged_df['trending_date'].dt.day_na

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

```
/var/folders/yn/hnpfh1r15tq8t0xq_j4_rzmh0000gn/T/ipykernel_32302/455230
003.py:8: UserWarning: Ignoring `palette` because no `hue` variable has
been assigned.
  sns.histplot(merged_df['trending_day_of_week'], discrete=True, palett
e='viridis')
```



In [8]:
```
#convert the publish_date to datetime type
merged_df['publish_time'] = pd.to_datetime(merged_df['publish_time'], fo
# Extract day of the week from 'publish_time'
merged_df['day_of_week'] = merged_df['publish_time'].dt.day_name()

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

```
/var/folders/yn/hnpfh1r15tq8t0xq_j4_rzmh0000gn/T/ipykernel_32302/134607
7495.py:8: UserWarning: Ignoring `palette` because no `hue` variable ha
s been assigned.
  sns.histplot(merged_df['day_of_week'], discrete=True, palette='viridi
s')
```



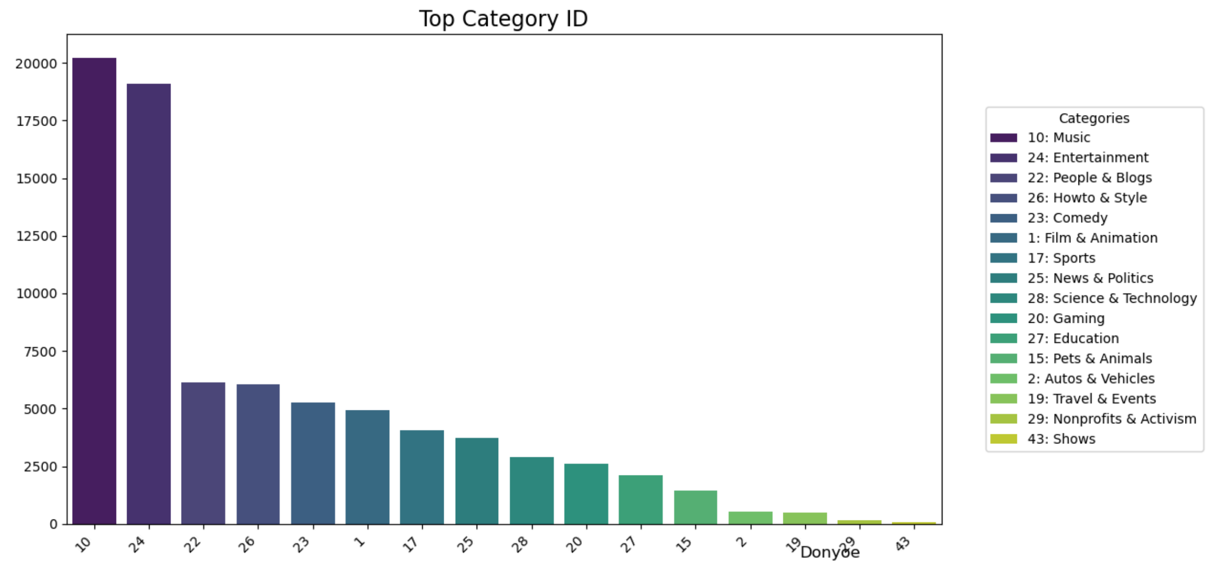
Statistical Description

In [9]:
```
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   
```

Visualization for Categorical ID

In [10]:
```
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(na

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

# 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(ca

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

# 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=1

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

# Position the legend on the right of the plot using 'bbox_to_anchor'
plt.legend(handles=handles[:len(legend_labels)], labels=legend_labels, t
           bbox_to_anchor=(1.05, 0.5), loc='center left', borderaxespad=

plt.show()
```



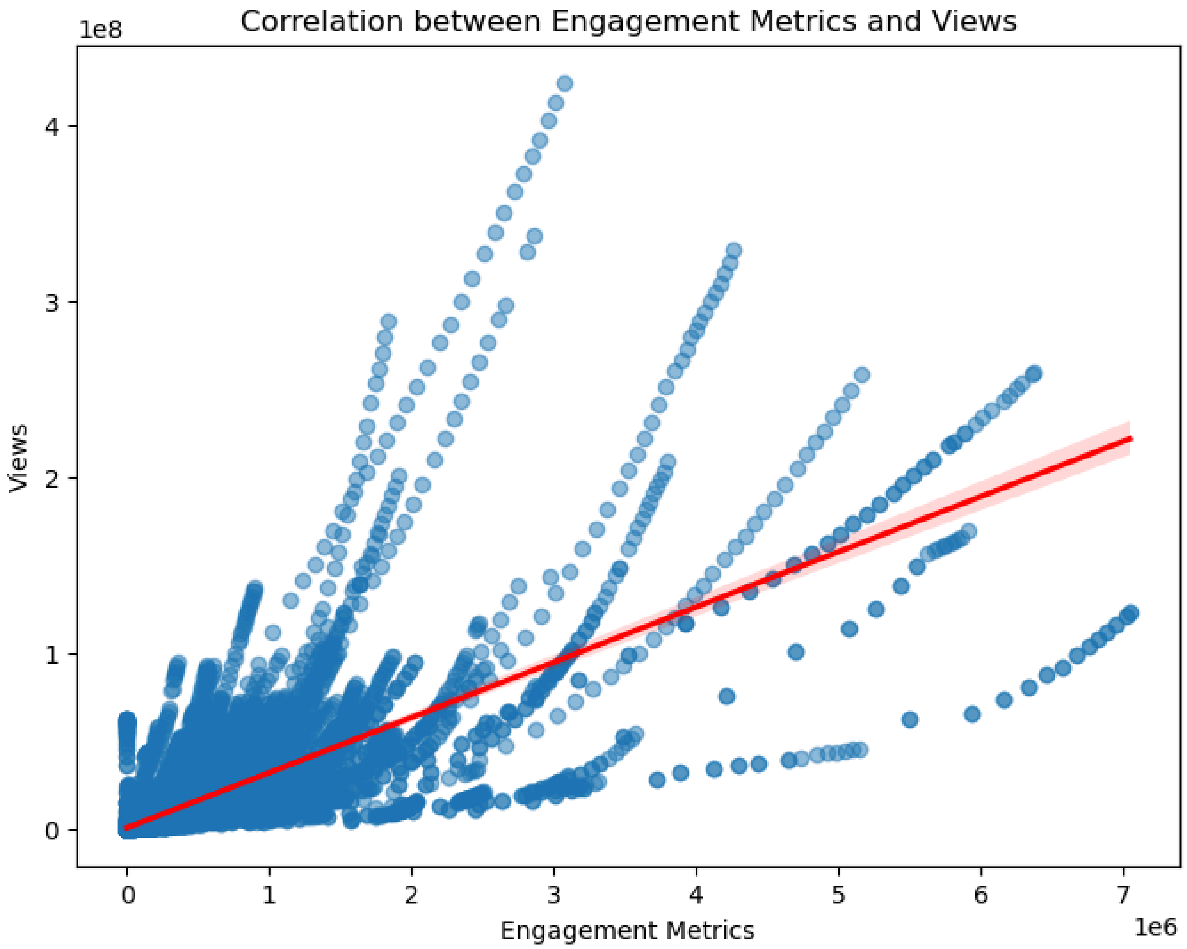
Data Transformation-Create Engagement Metrics

In [11]:
```
# Create a new column
merged_df['Engagement Metrics'] = merged_df['likes'] + merged_df['dislik
# Display the DataFrame to check the new column
print(merged_df[['likes', 'dislikes', 'comment_count', 'Engagement Metri
```

```
    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
```

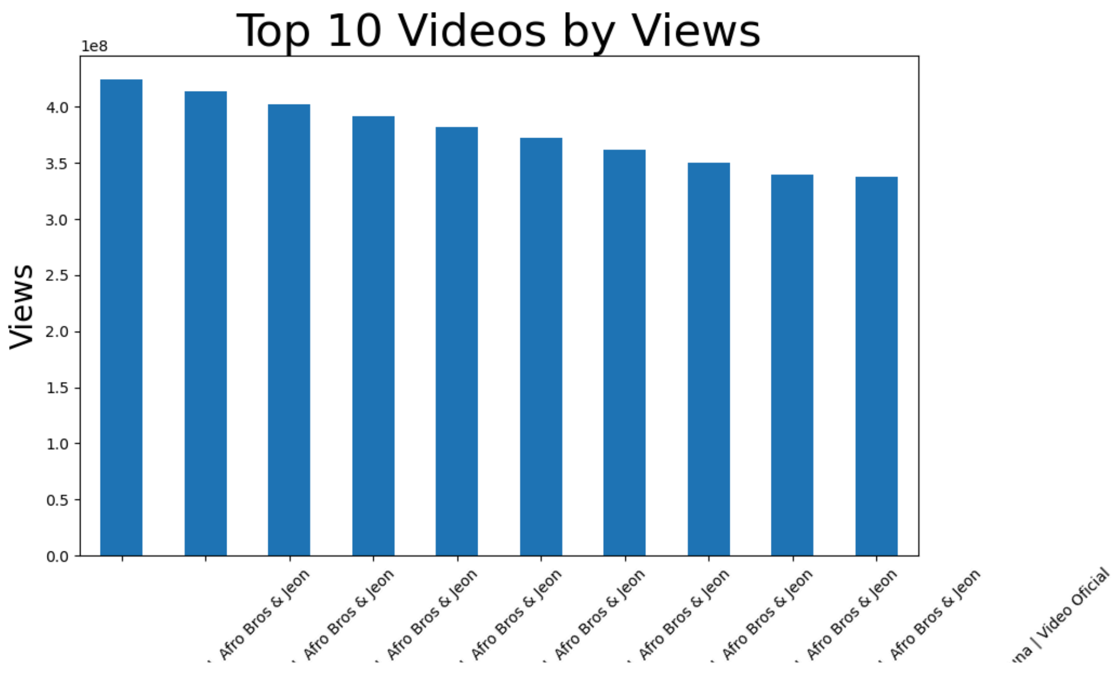
Scatter Plot for Engagement Metrics and Views

In [12]:
```
# Create a scatter plot with a regression line
plt.figure(figsize=(8, 6))
sns.regplot(x='Engagement Metrics', y='views', data=merged_df, scatter_kw
plt.title('Correlation between Engagement Metrics and Views')
plt.xlabel('Engagement Metrics')
plt.ylabel('Views')
plt.show()
```



Visualization of Top Viewed Videos

```
In [13]: 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()
```

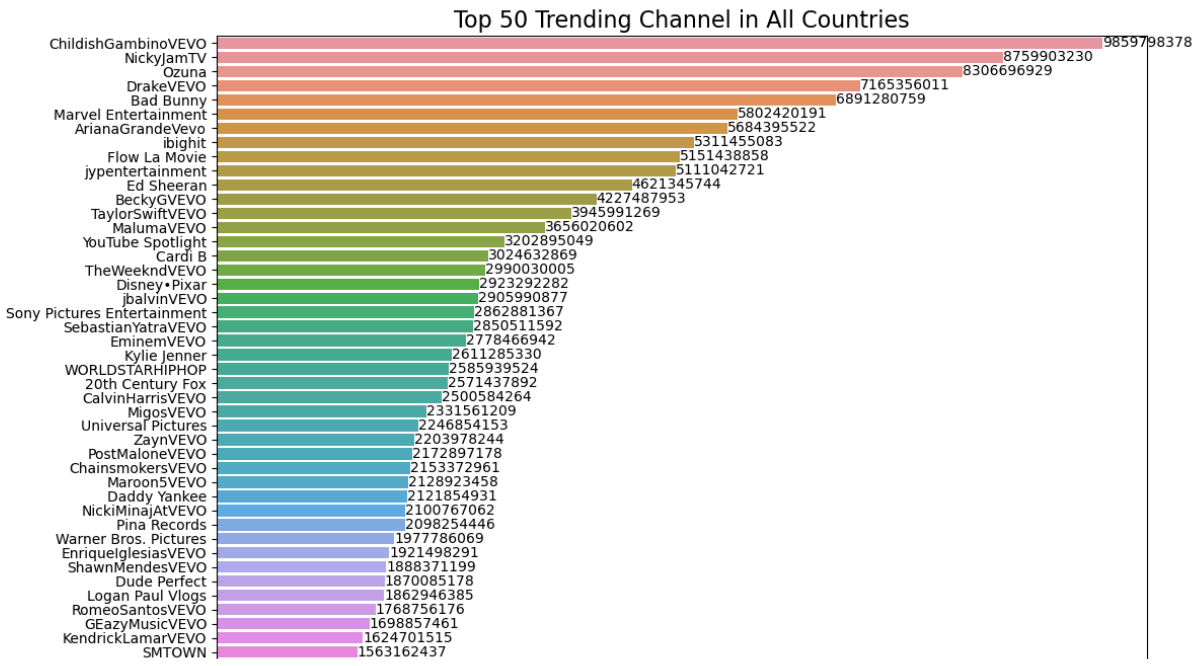


```
In [14]: # Engagement metrics for top 50 videos
         top_50_videos = merged_df.nlargest(50, 'views')
         print(top_50_videos[['title', 'Engagement Metrics']])
```

```
                                                    title  Engagement Me
trics
28412  Nicky Jam x J. Balvin - X (EQUIS) | Video Ofic...              30
67426
28212  Nicky Jam x J. Balvin - X (EQUIS) | Video Ofic...              30
11515
28008  Nicky Jam x J. Balvin - X (EQUIS) | Video Ofic...              29
56724
27811  Nicky Jam x J. Balvin - X (EQUIS) | Video Ofic...              29
02891
27615  Nicky Jam x J. Balvin - X (EQUIS) | Video Ofic...              28
45332
27424  Nicky Jam x J. Balvin - X (EQUIS) | Video Ofic...              27
86627
27241  Nicky Jam x J. Balvin - X (EQUIS) | Video Ofic...              27
23032
27052  Nicky Jam x J. Balvin - X (EQUIS) | Video Ofic...              26
50114
26861  Nicky Jam x J. Balvin - X (EQUIS) | Video Ofic...              25
```

Top 50 Channels

In [15]:
```
channel_counts = merged_df.groupby('channel_title')['views'].sum().reset_

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

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

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

# Customize the plot
plt.title('Top 50 Trending Channel 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
plt.tight_layout()

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

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

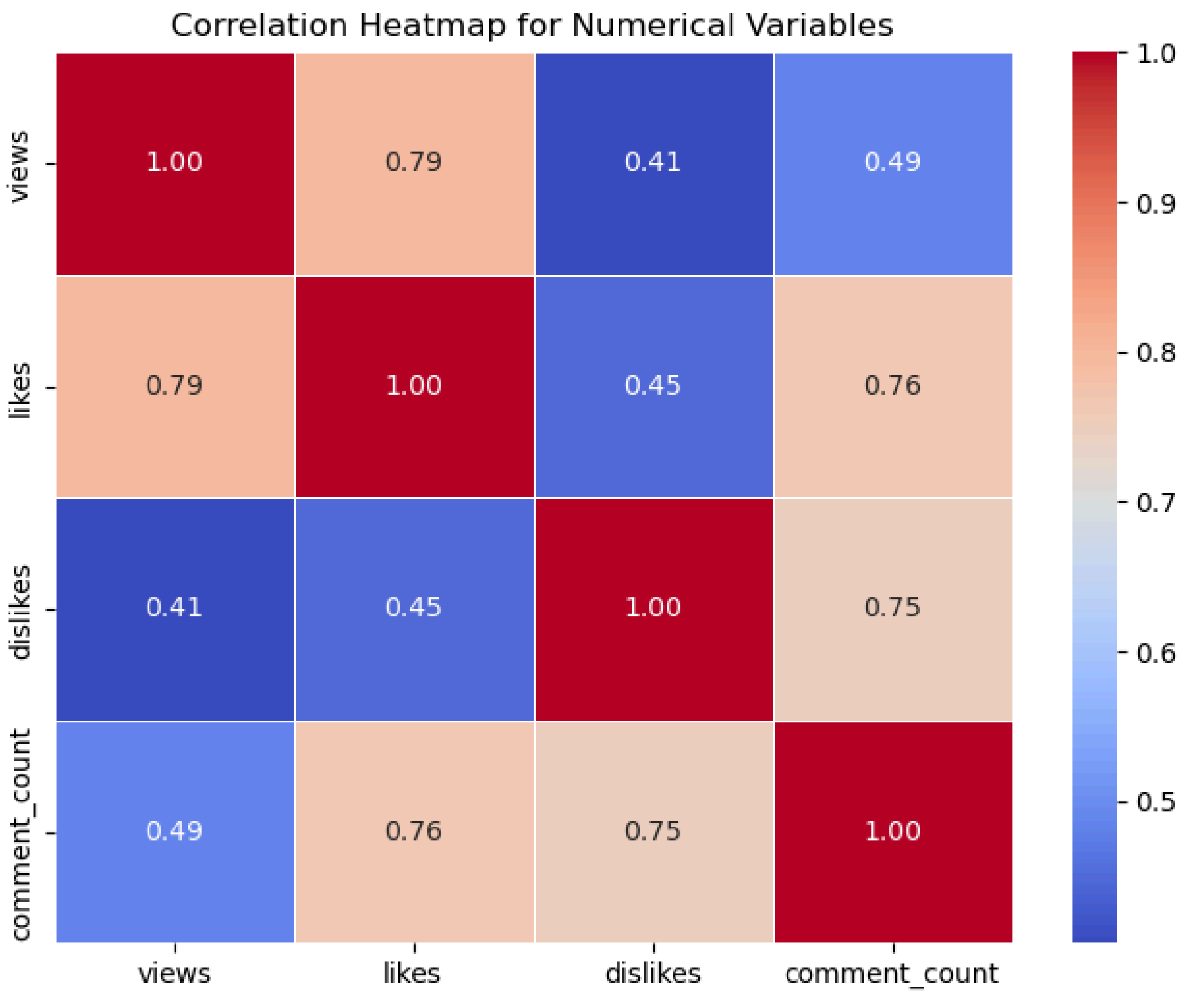


Normalize and Standardize Data

Heatmap for Views and Key Factors

In [16]:
```
numerical_columns = ['views', 'likes', 'dislikes', 'comment_count', ]

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

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

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



Assign Weights for Key Factors

In [17]:
```python
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

# 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_correlat

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

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

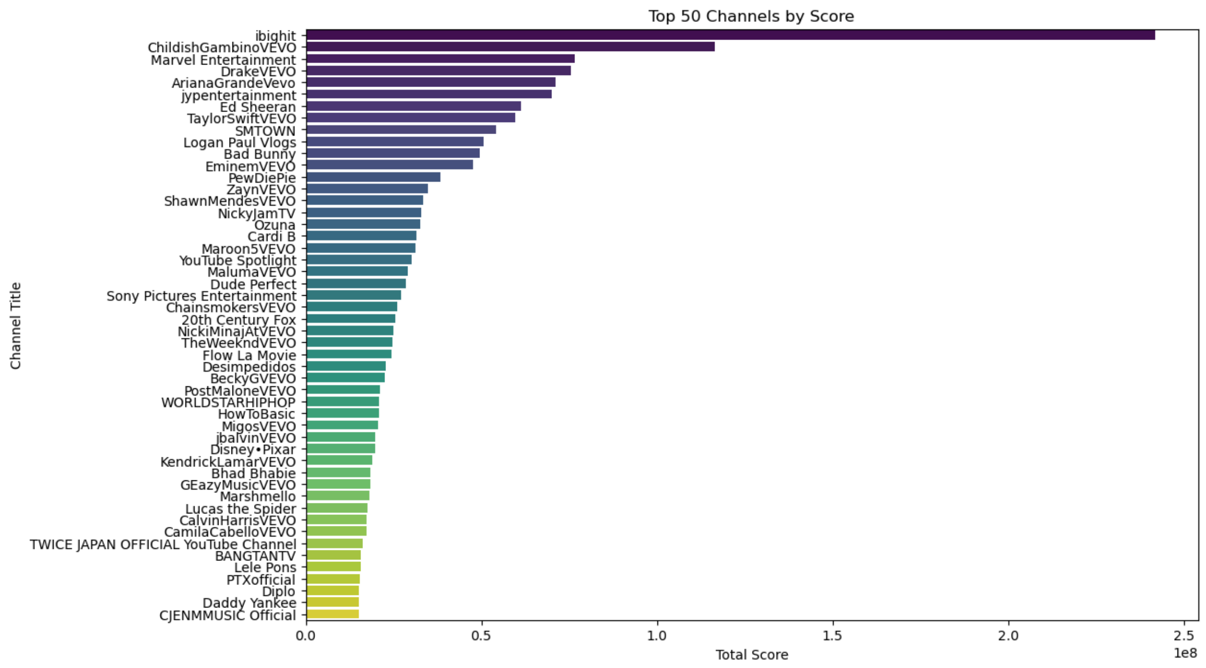
```
Calculated Weights of Independent Variables Relative to 'Views':
        Variable    Weight
0          likes  0.460635
1       dislikes  0.244418
2  comment_count  0.294947
```

In [18]:
```python
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)
```

```
          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
```

Refined EDA for Top 50 Channels

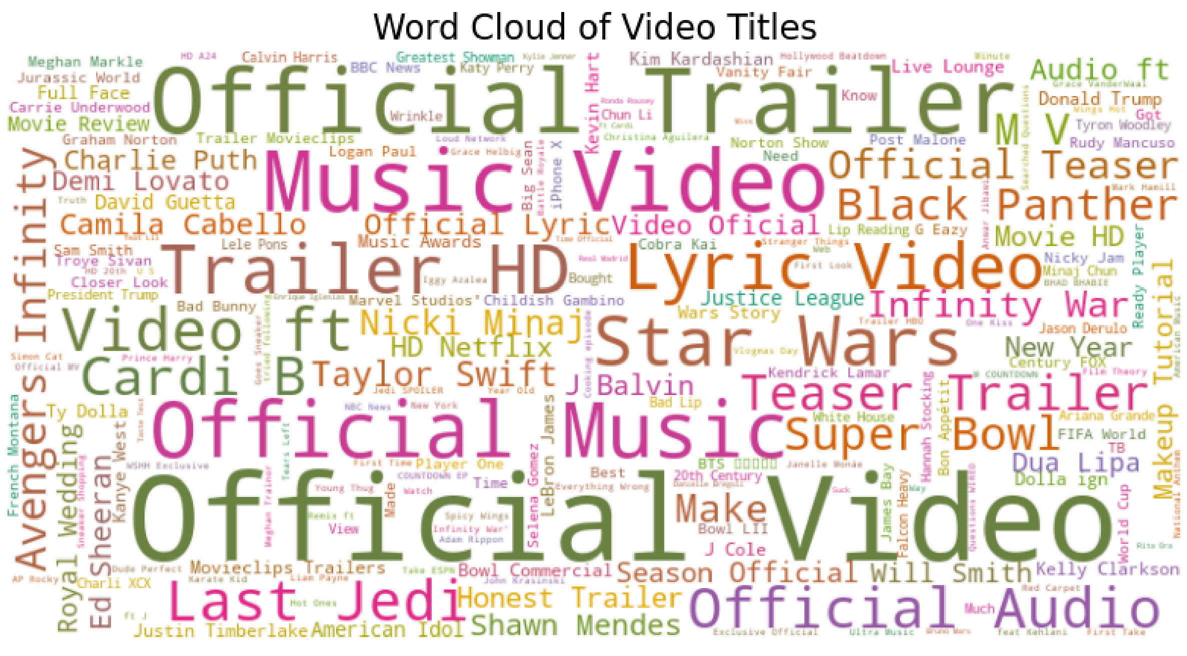
In [19]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# 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_

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

# 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='vi
plt.title('Top 50 Channels by Score')
plt.xlabel('Total Score')
plt.ylabel('Channel Title')
plt.show()
```



Create Word Cloud

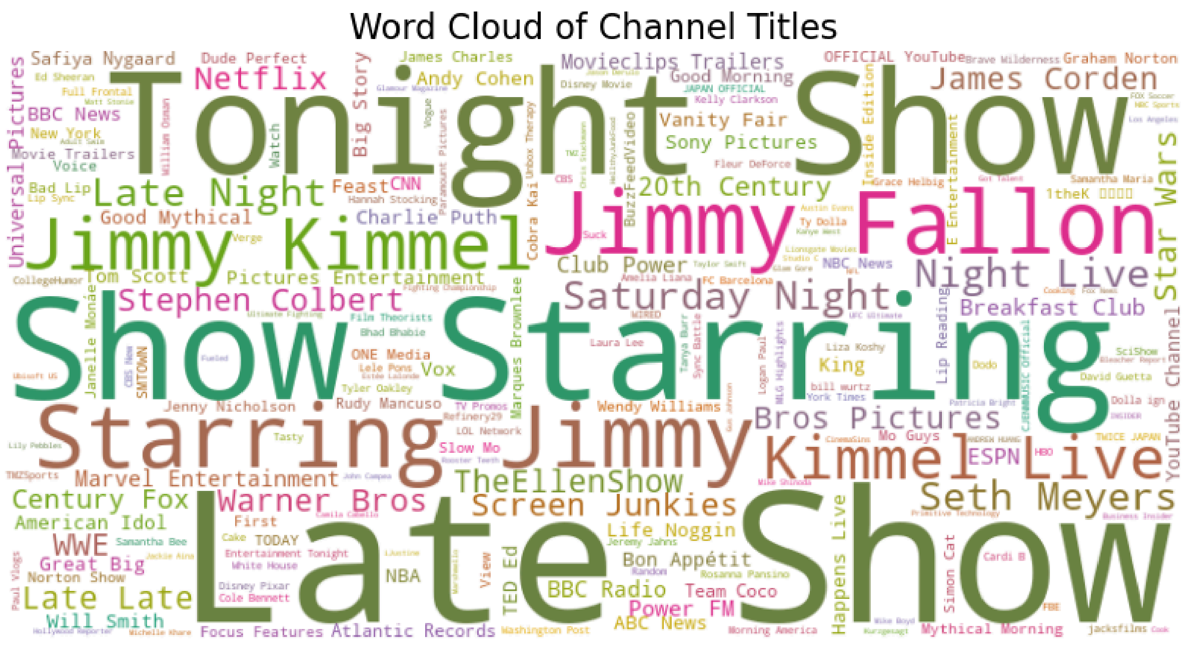
Video Titles

In [20]:
```python
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()
```



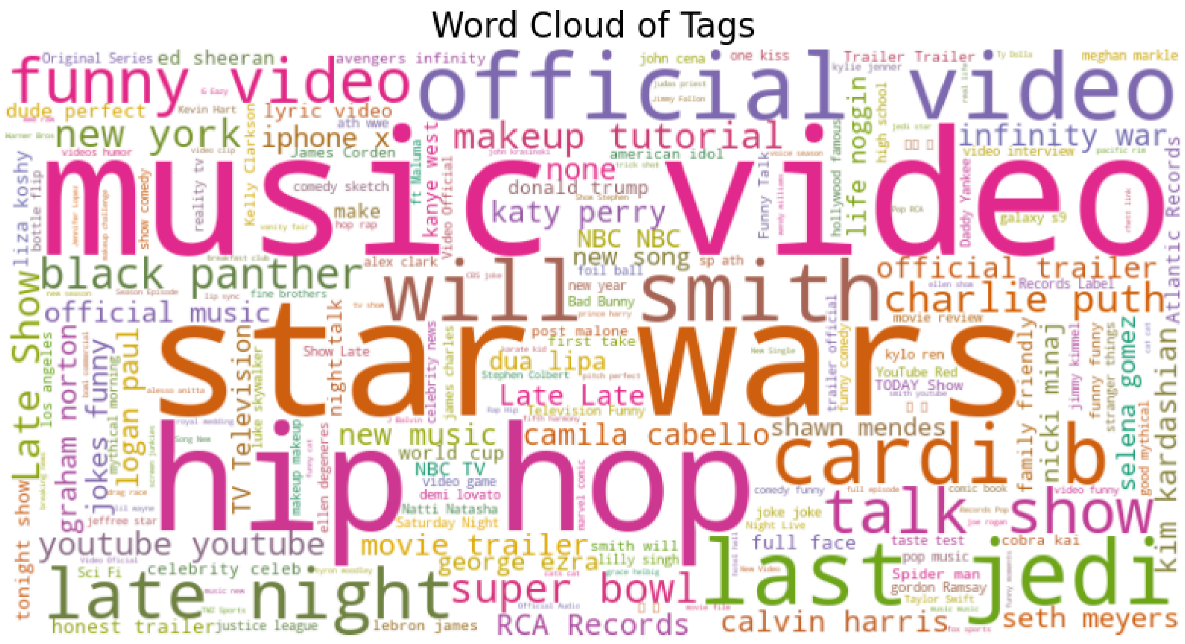
Channel Titles

In [21]:
```
all_channel_titles = " ".join(merged_df['channel_title'].astype(str))

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

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

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

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



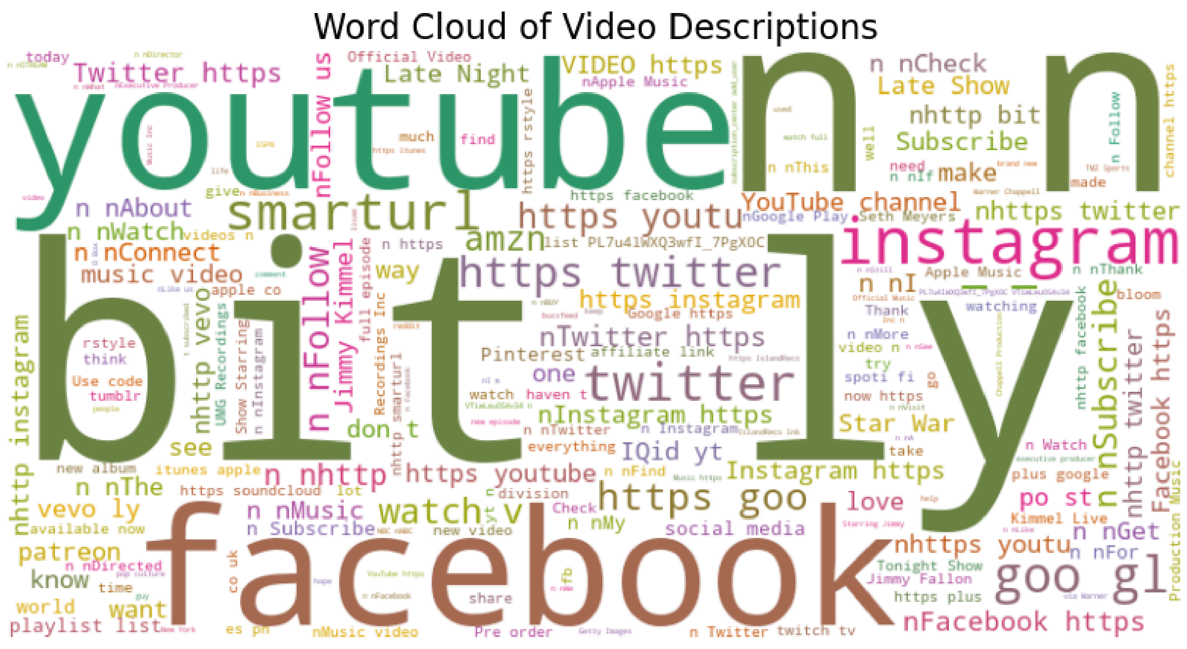
Video Tags

In [22]:
```
# Creating Word Cloud-tags
all_tags = " ".join(merged_df['tags'].astype(str))

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

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

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

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



Video Descriptions

In [23]:
```
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 [24]: #drop columns
         merged_df.drop(columns=['thumbnail_link', 'video_id','comments_disabled'

         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
```

Text Preprocessing

In [25]:
```
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-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)
```

```
uesday
3      Salford drew 4-4 against the Class of 92 and F...             T
uesday
4      Dashcam captures truck's near miss with child ...             T
uesday
...                                                  ...
...
79860     The Cat Who Caught the Laser - Aaron's Animals            Th
ursday
79861                                                NaN            Th
ursday
79862  I had so much fun transforming Safiyas hair in...            Th
ursday
79863  How Black Panther Should Have EndedWatch More ...            Th
ursday
79864  Call of Duty: Black Ops 4 Multiplayer raises t...            Th
ursday

      day_of_week  Engagement Metrics         score     rank  \
0          Friday               75407  25928.732602  24933.0
```

In [26]:
```
# 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
trending_day_of_week           object
day_of_week                    object
Engagement Metrics              int64
score                         float64
rank                          float64
new_text                       object
dtype: object
```

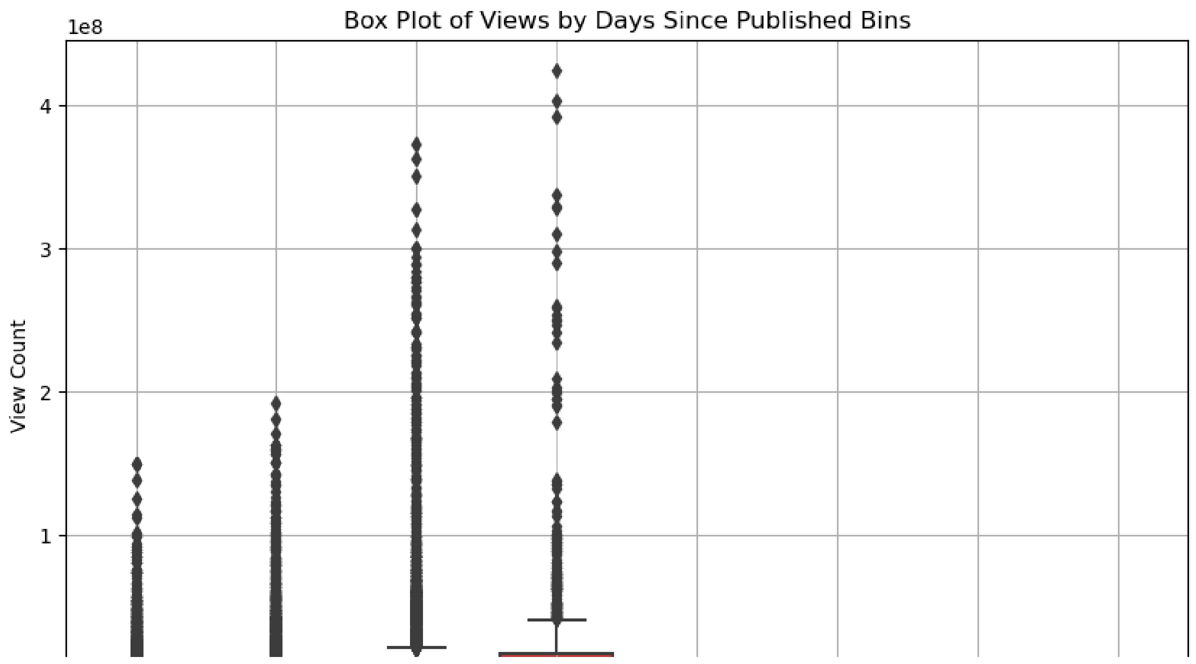
Split the Dataset into Train and Test by 80/20

In [27]:
```
from sklearn.model_selection import train_test_split

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

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

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

Feature Engineering

Days Since Published

In [28]:
```
#convert the type of publish time
train['publish_time'] = pd.to_datetime(train['publish_time'])
train['trending_date'] = pd.to_datetime(train['trending_date'], format='

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

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

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()
```

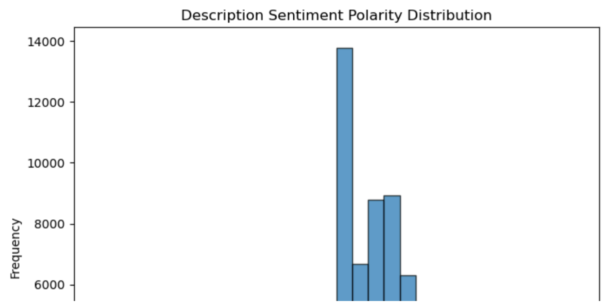


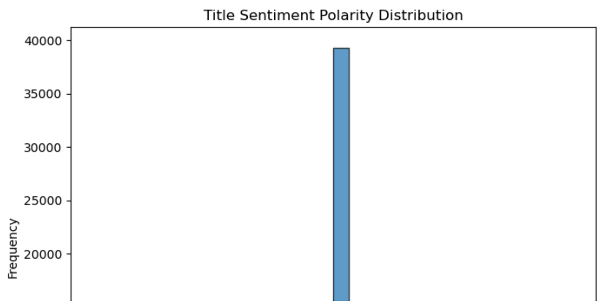
Sentimental Analysis

Sentiment Polarity Distribution

In [29]:
```
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(g
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='
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()
```

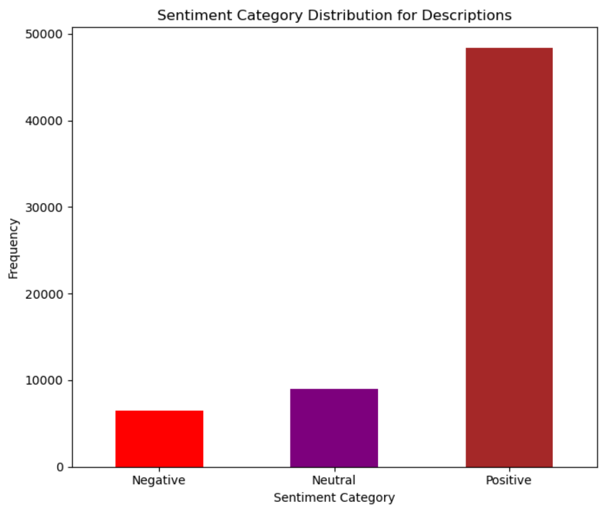
```
Average Description Sentiment Score: 0.1716764242965884
Average Title Sentiment Score: 0.0477964529239135
```

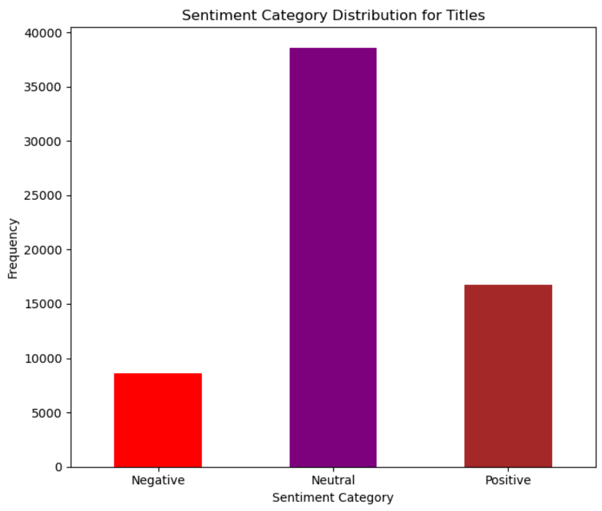

Description Sentiment Polarity Distribution
Frequency
14000
12000
10000
8000
6000


Title Sentiment Polarity Distribution
Frequency
40000
35000
30000
25000
20000

Visualize the Sentiment Distrubution

In [30]:
```
import pandas as pd
import matplotlib.pyplot as plt

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

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

# 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'].v
description_sentiment_counts.plot(kind='bar', color=['red', 'purple', 'b
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(
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()
```


Sentiment Category Distribution for Descriptions
Frequency
0
10000
20000
30000
40000
50000
Negative
Neutral
Positive
Sentiment Category


Sentiment Category Distribution for Titles
Frequency
0
5000
10000
15000
20000
25000
30000
35000
40000
Negative
Neutral
Positive
Sentiment Category

Create TF-IDF Feature

`description` Column

In [31]:
```
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=

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

        # 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 indic
            top_features = [(features[i], row[i]) for i in top_indices]
            return top_features

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

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

        # Display the entire first 5 rows of the DataFrame including the
        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"|"Marshmell
o"|"...   
25630 2018-03-16 04:00:01  Kirby|"Kirby Star Allies"|"Dedede"|"Meta Kni
gh...   
68698 2018-04-17 14:55:31  espn|"dwyane wade"|"dwayne wade"|"d wade"|"7
6e...   
39559 2017-11-15 13:29:28               Zodiac|"makeup"|"comedy"|"aqua
rius"   
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_sen
timent  \
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 24 columns]
```

tags Column

In [32]:
```
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=

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

        # 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 indic
            top_features = [(features[i], row[i]) for i in top_indices]
            return top_features

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

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

        # Display the entire first 5 rows of the DataFrame including the
        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"|"Marshmell
o"|"...   
25630 2018-03-16 04:00:01  Kirby|"Kirby Star Allies"|"Dedede"|"Meta Kni
```

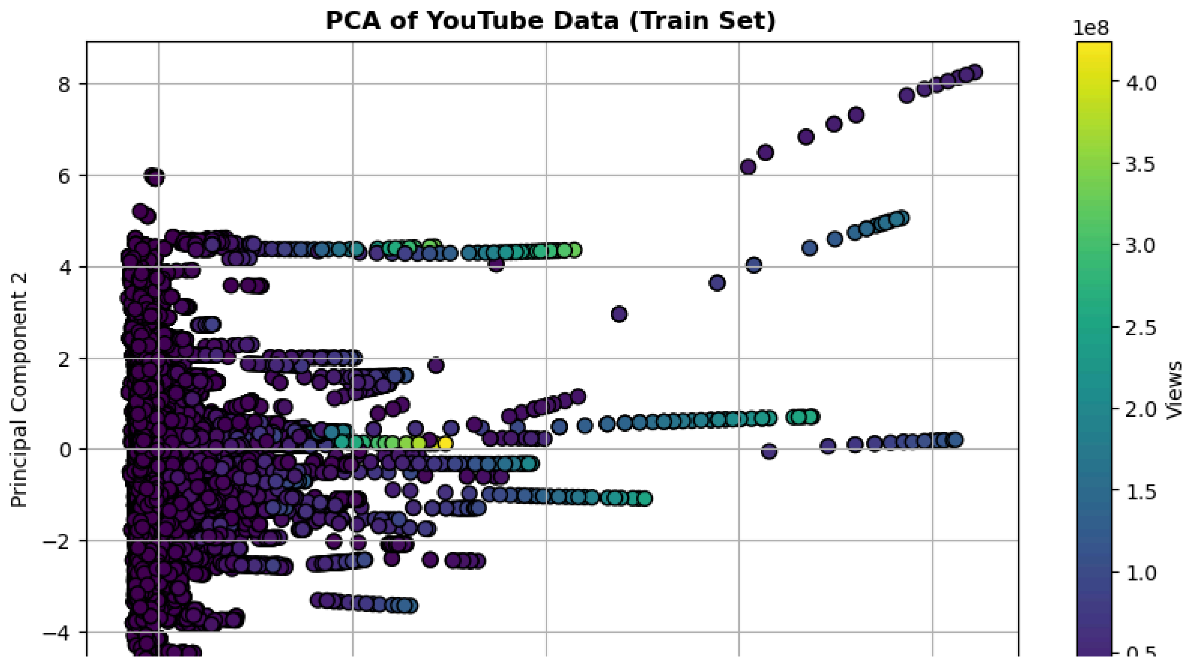
Dimension Reduction-PCA

In [33]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

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

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

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())
```

```
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
```

Visualize PCA

In [34]:
```
# 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='virid
plt.colorbar(label='Views')
plt.title('PCA of YouTube Data (Train Set)', weight='bold')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.grid(True)
plt.show()

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



Model Building

Load Library

In [35]:
```
import xgboost as xgb
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np
import pandas as pd
```

Define Model Metrics

In [36]:
```
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
    }
```

Introduce Model

In [37]:
```
# 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
```

Define Optimal Hyperparameter Variations

In [38]:
```
# Define hyperparameter variations
variations = [
    {"learning_rate": 0.05, "n_estimators": 200, "max_depth": 6}
]
```

In [39]:
```
# Initialize a DataFrame to store results for each variation
results = pd.DataFrame(columns=["Variation", "Train RMSE", "Test RMSE",
```

Introduce Dataset to Fit Model

In [40]:
```
# Create a list of columns to drop if they exist
text_columns = ['title', 'channel_title', 'tags', 'description', 'locati
X_train.drop([col for col in text_columns if col in X_train.columns], ax
X_test.drop([col for col in text_columns if col in X_test.columns], axis

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

# 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_co
unt',
       'Engagement Metrics', 'score', 'rank', 'trending_day_of_week_Mon
day',
       '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_Thurs
day',
       '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_cou
nt',
       'Engagement Metrics', 'score', 'rank', 'trending_day_of_week_Mon
day',
       '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_Thurs
day',
       'day_of_week_Tuesday', 'day_of_week_Wednesday', 'trending_year',
       'trending_month', 'trending_day'],
      dtype='object')
```

Calculate the Metrics

In [41]:
```
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 [42]:
```python
# 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, par

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

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

Display the Model Metrics

In [43]:
```python
# Display the comparison table
print("Comparison of XGBoost Model Variations:")
print(results)
best_model_index = results["Test RMSE"].idxmin()
best_params = variations[best_model_index]
print(f"\n 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  2.120971e+06  2.909246e+06   0.978047  0.960791

 Model Variation: 1
Hyperparameters: {'learning_rate': 0.05, 'n_estimators': 200, 'max_dept
h': 6}
Variation       Variation 1
Train RMSE   2120971.173631
Test RMSE    2909245.576456
Train R^2          0.978047
Test R^2           0.960791
Name: 0, dtype: object
```

Explain the model

Define Features Importance

In [44]:
```
# Get feature importances directly
feature_importances = model.feature_importances_

feature_importances_adjusted = feature_importances[:len(X.columns)]

# Generate feature importance DataFrame
top_features = pd.DataFrame({'Feature': X.columns, 'Importance': feature
top_10_features = top_features.head(10)

print("Top 10 Features:\n", top_10_features)
```

```
Top 10 Features:
            Feature  Importance
1            title    0.437080
4     publish_time    0.193274
2    channel_title    0.085307
5             tags    0.051353
3      category_id    0.048652
14            rank    0.040345
15        new_text    0.027795
6            likes    0.009272
13           score    0.009106
0    trending_date    0.008889
```

Extracting and Analyzing 5 Individual Predictions by Using SHAP

In [45]:
```
import shap
import numpy as np


# Randomly select 5 samples from the test set
random_samples = X_test.sample(5, random_state=42)

explainer = shap.TreeExplainer(model)
shap_values = explainer.shap_values(random_samples)

# Generate and analyze SHAP values for random samples
for i, sample_index in enumerate(random_samples.index):
    print(f"\nExplanation for sample {i+1} (Index: {sample_index}):")

    # Display all columns' information for this prediction
    print("Sample details:")
    display(random_samples.loc[[sample_index]])  # Show the full row det

    # Calculate and display the SHAP force plot for visualization
    shap.initjs()
    shap.force_plot(explainer.expected_value, shap_values[i], random_sam

    # Retrieve SHAP values for the sample and sort by absolute importanc
    sample_shap_values = shap_values[i]
    feature_importances = pd.DataFrame({
        'feature': random_samples.columns,
        'shap_value': sample_shap_values
    }).set_index('feature').sort_values(by='shap_value', key=abs, ascend

    # Display the top features that influenced the prediction
    print("Top contributing features:")
    print(feature_importances.head())

    # Suggest changes for flipping the prediction
    current_prediction = model.predict(random_samples.iloc[[i]])[0]
    if current_prediction == 1:
        print(f"To flip from 1 to 0, consider reducing the values of the
    else:
        print(f"To flip from 0 to 1, consider increasing the values of t

    # Suggest how to change the feature values
    for feature, shap_value in feature_importances.head().itertuples():
        direction = "increase" if shap_value < 0 else "decrease"
        print(f" - {feature}: Consider a {direction} of {abs(shap_value)
```

```
Explanation for sample 1 (Index: 49053):
Sample details:
```

	category_id	likes	dislikes	comment_count	Engagement Metrics	score	rank	trending_c
49053	23	2837	86	191	3114	1341.569024	67271.0	

1 rows × 22 columns

Model Deployment, Data Drift and Concept Drift, and Model Monitoring

Serializing and Saving the Model with Pickle

In [46]:
```
import pickle

model_filename = 'XGBoost_model.pkl'

# Save the trained model to a file
with open(model_filename, 'wb') as file:
    pickle.dump(model, file)

print(f"Model saved to {model_filename}")
```

```
Model saved to XGBoost_model.pkl
```

In [47]:
```
# Load the model from the file
with open(model_filename, 'rb') as file:
    loaded_model = pickle.load(file)
```

In [48]:
```
predictions = loaded_model.predict(X_test)
```

Save and Load the Data

In [49]:
```
data_filename = 'data.pkl'

# Save datasets to a file
with open(data_filename, 'wb') as file:
    pickle.dump({'X_train': X_train, 'X_test': X_test, 'y_train': y_trai

print(f"Datasets saved to {data_filename}")
```

```
Datasets saved to data.pkl
```

In [50]:
```
# Load datasets from the file
with open(data_filename, 'rb') as file:
    data = pickle.load(file)

# Extract datasets
X_train = data['X_train']
X_test = data['X_test']
y_train = data['y_train']
y_test = data['y_test']
```

Obtain the Environment Dependencies

In [51]:
```
import platform
import sys
import pkg_resources

# Collect OS and Python version information
print("OS:", platform.system(), platform.version())
print(f"Operating System: {platform.system()} {platform.release()}")
print("Python Version:", sys.version)
```

```
OS: Darwin Darwin Kernel Version 23.3.0: Wed Dec 20 21:28:58 PST 2023;
root:xnu-10002.81.5~7/RELEASE_X86_64
Operating System: Darwin 23.3.0
Python Version: 3.11.5 (main, Sep 11 2023, 08:19:27) [Clang 14.0.6 ]
```

Identify the Packages Used and Version

In [52]:
```
import importlib
import pkg_resources

# List of packages identified from notebook
packages = [
    "matplotlib.pyplot", "nltk", "numpy", "os", "palettable.colorbrewer.
    "pandas", "re", "seaborn", "shap", "sklearn.decomposition",
    "sklearn.feature_extraction.text", "sklearn.metrics", "sklearn.model_
    "sklearn.preprocessing", "textblob", "wordcloud", "xgboost", "zipfil
]

# Function to get package version
def get_package_version(package):
    try:
        # Handle "matplotlib.pyplot" style imports
        if '.' in package:
            package = package.split('.')[0]
        # Try pkg_resources first
        version = pkg_resources.get_distribution(package).version
    except pkg_resources.DistributionNotFound:
        # If not found, try importlib
        try:
            module = importlib.import_module(package)
            version = getattr(module, '__version__', 'Version not found'
        except ImportError:
            version = 'Not installed'
    return version

# Collect versions
package_versions = {pkg: get_package_version(pkg) for pkg in packages}

# Display package versions
print("Package Versions:")
for pkg, version in package_versions.items():
    print(f"{pkg}: {version}")
```

```
Package Versions:
matplotlib.pyplot: 3.7.2
nltk: 3.8.1
numpy: 1.24.3
os: Version not found
palettable.colorbrewer.qualitative: 3.3.3
pandas: 2.0.3
re: 2.2.1
seaborn: 0.12.2
shap: 0.46.0
sklearn.decomposition: 1.3.0
sklearn.feature_extraction.text: 1.3.0
sklearn.metrics: 1.3.0
sklearn.model_selection: 1.3.0
sklearn.preprocessing: 1.3.0
textblob: 0.18.0.post0
wordcloud: 1.9.3
xgboost: 2.0.3
zipfile: Version not found
```