analysis

August 13, 2024

1 YouTube Data Collection and Analysis with Python

[19]:
```python
import pandas as pd
from googleapiclient.discovery import build
import warnings
warnings.filterwarnings('ignore')
```

[21]:
```python
# replace with your own API key
API_KEY = 'AIzaSyBg23zBSyH_1jZlXE7ET_eYsG4tew4VA6g'

def get_trending_videos(api_key, max_results=200):
    # build the youtube service
    youtube = build('youtube', 'v3', developerKey=api_key)

    # initialize the list to hold video details
    videos = []

    # fetch the most popular videos
    request = youtube.videos().list(
        part='snippet,contentDetails,statistics',
        chart='mostPopular',
        regionCode='US',
        maxResults=50
    )

    # paginate through the results if max_results > 50
    while request and len(videos) < max_results:
        response = request.execute()
        for item in response['items']:
            video_details = {
                'video_id': item['id'],
                'title': item['snippet']['title'],
                'description': item['snippet']['description'],
                'published_at': item['snippet']['publishedAt'],
                'channel_id': item['snippet']['channelId'],
                'channel_title': item['snippet']['channelTitle'],
                'category_id': item['snippet']['categoryId'],
                'tags': item['snippet'].get('tags', []),
                'duration': item['contentDetails']['duration'],
                'definition': item['contentDetails']['definition'],
                'caption': item['contentDetails'].get('caption', 'false'),
                'view_count': item['statistics'].get('viewCount', 0),
                'like_count': item['statistics'].get('likeCount', 0),
                'dislike_count': item['statistics'].get('dislikeCount', 0),
                'favorite_count': item['statistics'].get('favoriteCount', 0),
                'comment_count': item['statistics'].get('commentCount', 0)
            }
            videos.append(video_details)

        # get the next page token
        request = youtube.videos().list_next(request, response)

    return videos[:max_results]

def save_to_csv(data, filename):
    df = pd.DataFrame(data)
    df.to_csv(filename, index=False)

def main():
    trending_videos = get_trending_videos(API_KEY)
    filename = 'trending_videos.csv'
    save_to_csv(trending_videos, filename)
    print(f'Trending videos saved to {filename}')

if __name__ == '__main__':
    main()
```

```
Trending videos saved to trending_videos.csv
```

1) In the above code, we are using the YouTube Data API to fetch details of the top 200 trending videos in the US, iterating through the API's paginated responses to collect video details such as title, description, published date, channel information, tags, duration, definition, captions, and various engagement metrics like views, likes, and comments. The script compiles this information into a list, converts it into a pandas DataFrame, and saves the data to a CSV file named trending_videos.csv, allowing us to analyze trends and patterns in the collected video data.

[3]:
```
import pandas as pd

trending_videos = pd.read_csv('trending_videos.csv')
print(trending_videos.head())
```

```
      video_id                                             title  \
0  5uE0XFJSVZA  Fortnite Battle Royale Chapter 5 Season 4 - Ab…
1  Q4Qo6we8yu8     Five Nights at Freddy's: Into The Pit - Part 1
2  CskyDVPbN1g  My Daughter's FIRST DAY OF HIGH SCHOOL *Hidden…
```

```
3  hH5gf4Yf980  I Tried Every Viral Fast Food Menu Item In Ame…
4  FrDIrMHOXy4              Aokiji vs Garp (Part 2/2) | One Piece

                                           description          published_at  \
0  Suit up for Fortnite Battle Royale Chapter 5 S…  2024-08-11T04:20:35Z
1  Are you ready to jump back into the Five Night…  2024-08-10T16:02:18Z
2  To make sure my daughter Salish is ready for h…  2024-08-10T14:00:07Z
3  Are all these fancy new menu items better or w…  2024-08-11T14:30:15Z
4  Episode 1115: Aokiji and Garp's fight continue…  2024-08-11T15:00:13Z

                 channel_id    channel_title  category_id  \
0  UClG8odDC8TS6Zpqk9CGVQiQ         Fortnite           22
1  UC7_YxT-KID8kRbqZo7MyscQ       Markiplier           20
2  UCKaCalz5N5ienIbfPzEbYuA    Jordan Matter           24
3  UChBEbMKI1eCcejTtmI32UEw  Joshua Weissman           26
4  UC6pGDc4bFGD1_36IKv3FnYg      Crunchyroll            1

                                              tags  duration definition  \
0  ['yt:cc=on', 'Marvel', 'Fortnite', 'Battle Roy…   PT1M32S         hd
1  ['markiplier five nights at freddys', "five ni…  PT59M19S         hd
2  ['salish matter', 'saysay matter', 'gymnastics…  PT29M24S         hd
3  ['sat bawl pro', 'joshua weissman', 'i tried e…  PT22M18S         hd
4  ['one piece', 'one piece anime', 'one piece eg…   PT1M38S         hd

   caption  view_count  like_count  dislike_count  favorite_count  \
0    False     3350770      153593              0               0
1    False     3305857      205292              0               0
2    False     7110201      112103              0               0
3    False      849956       26988              0               0
4    False      610672       14708              0               0

   comment_count
0          14943
1           6211
2          12916
3           1209
4           1296
```

[4]:
```
# check for missing values
missing_values = trending_videos.isnull().sum()

# display data types
data_types = trending_videos.dtypes

missing_values, data_types
```

[4]:
```
(video_id          0
 title             0
 description       1
 published_at      0
 channel_id        0
 channel_title     0
 category_id       0
 tags              0
 duration          0
 definition        0
 caption           0
 view_count        0
 like_count        0
 dislike_count     0
 favorite_count    0
 comment_count     0
 dtype: int64,
 video_id          object
 title             object
 description       object
 published_at      object
 channel_id        object
 channel_title     object
 category_id        int64
 tags              object
 duration          object
 definition        object
 caption             bool
 view_count         int64
 like_count         int64
 dislike_count      int64
 favorite_count     int64
 comment_count      int64
 dtype: object)
```

1) The description column has 4 missing values. This is minor and can be handled as needed. The data types seem appropriate for most columns, but we may need to convert the published_at column to a datetime format and tags might need further processing. Let's fix these changes:

[5]:
```python
# fill missing descriptions with "No description"
trending_videos['description'].fillna('No description', inplace=True)

# convert `published_at` to datetime
trending_videos['published_at'] = pd.
 ↪to_datetime(trending_videos['published_at'])

# convert tags from string representation of list to actual list
```

```
trending_videos['tags'] = trending_videos['tags'].apply(lambda x: eval(x) if␣
↪isinstance(x, str) else x)
```

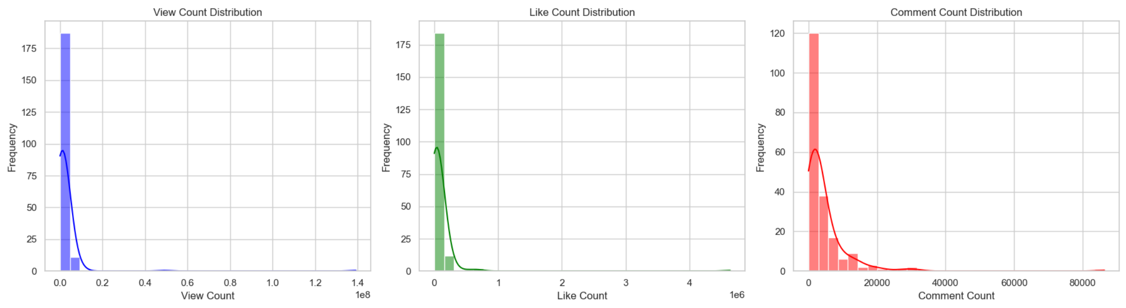
[6]:
```
# descriptive statistics
descriptive_stats = trending_videos[['view_count', 'like_count',␣
↪'dislike_count', 'comment_count']].describe()

descriptive_stats
```

[6]:
```
         view_count    like_count  dislike_count  comment_count
count  2.000000e+02  2.000000e+02          200.0      200.00000
mean   2.371173e+06  7.925626e+04            0.0     4440.96500
std    1.040635e+07  3.358209e+05            0.0     7773.83778
min    4.658900e+04  0.000000e+00            0.0        0.00000
25%    5.174428e+05  1.255400e+04            0.0     1091.25000
50%    8.635990e+05  2.901200e+04            0.0     1925.50000
75%    1.671098e+06  6.635775e+04            0.0     4437.50000
max    1.391897e+08  4.639393e+06            0.0    86350.00000
```

1.0.1 let's have a look at the distribution of views, likes and comments of all the videos in the data:

[7]:
```
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(style="whitegrid")

fig, axes = plt.subplots(1, 3, figsize=(18, 5))

# view count distribution
sns.histplot(trending_videos['view_count'], bins=30, kde=True, ax=axes[0],␣
↪color='blue')
axes[0].set_title('View Count Distribution')
axes[0].set_xlabel('View Count')
axes[0].set_ylabel('Frequency')

# like count distribution
sns.histplot(trending_videos['like_count'], bins=30, kde=True, ax=axes[1],␣
↪color='green')
axes[1].set_title('Like Count Distribution')
axes[1].set_xlabel('Like Count')
axes[1].set_ylabel('Frequency')

# comment count distribution
sns.histplot(trending_videos['comment_count'], bins=30, kde=True, ax=axes[2],␣
↪color='red')
axes[2].set_title('Comment Count Distribution')
axes[2].set_xlabel('Comment Count')
```

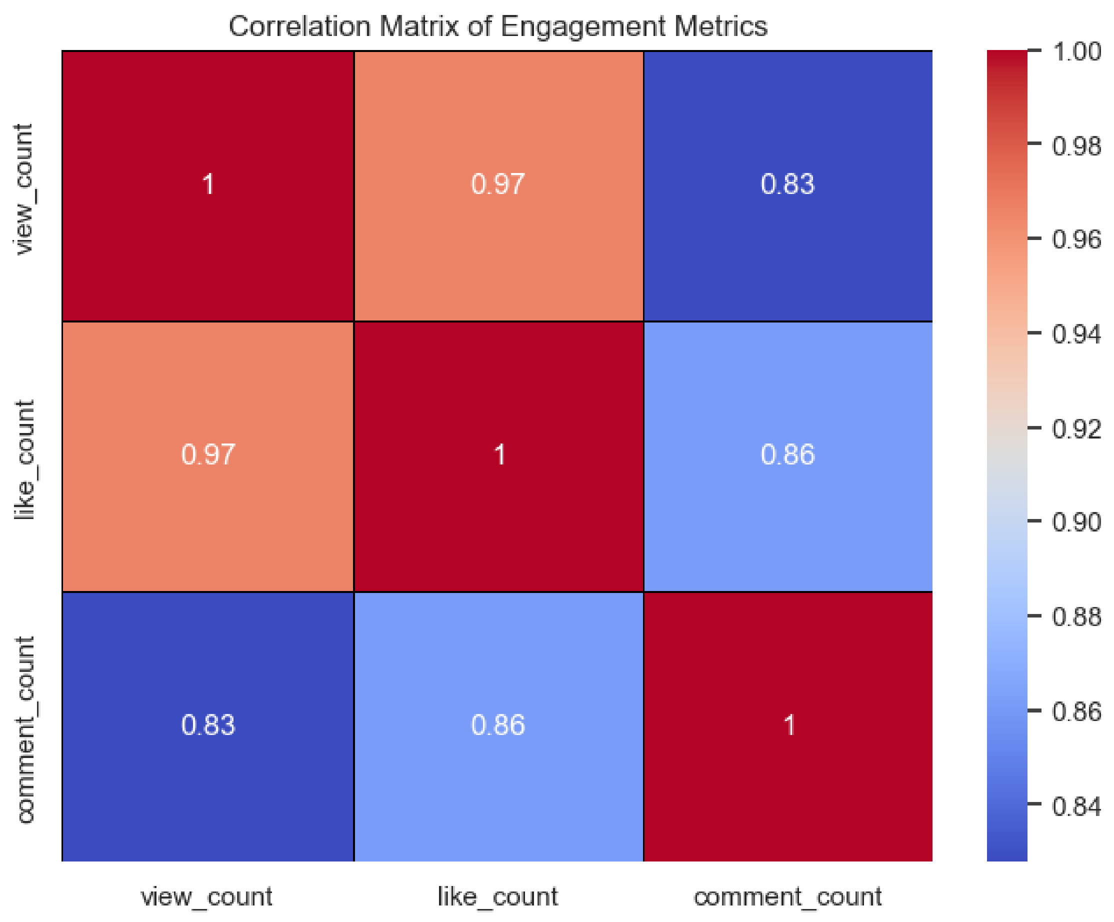
```
axes[2].set_ylabel('Frequency')

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



1) The histograms show that the distributions of view counts, like counts, and comment counts are right-skewed, with most videos having lower counts and a few videos having very high counts.

1.0.2 let's have a look at the correlation between likes, views, and comments:

[8]:
```
# correlation matrix
correlation_matrix = trending_videos[['view_count', 'like_count',␣
↪'comment_count']].corr()

plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5,␣
↪linecolor='black')
plt.title('Correlation Matrix of Engagement Metrics')
plt.show()
```



1) The heatmap confirms strong positive correlations between views, likes, and comments.

1.0.3 collecting the category names as well to analyze the categories of the trending videos:

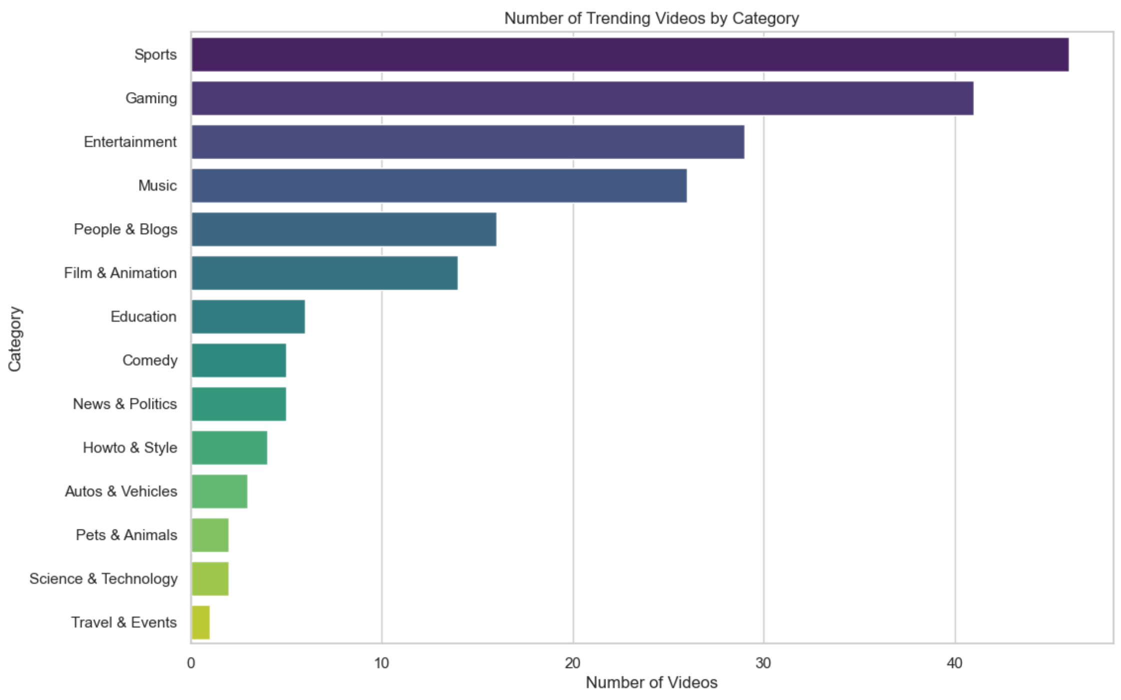
[10]:
```
youtube = build('youtube', 'v3', developerKey=API_KEY)
def get_category_mapping():
    request = youtube.videoCategories().list(
        part='snippet',
        regionCode='US'
    )
    response = request.execute()
    category_mapping = {}
    for item in response['items']:
        category_id = int(item['id'])
        category_name = item['snippet']['title']
        category_mapping[category_id] = category_name
    return category_mapping
```

```
# get the category mapping
category_mapping = get_category_mapping()
print(category_mapping)
```

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

1.0.4 Let's analyze the number of trending videos by category:

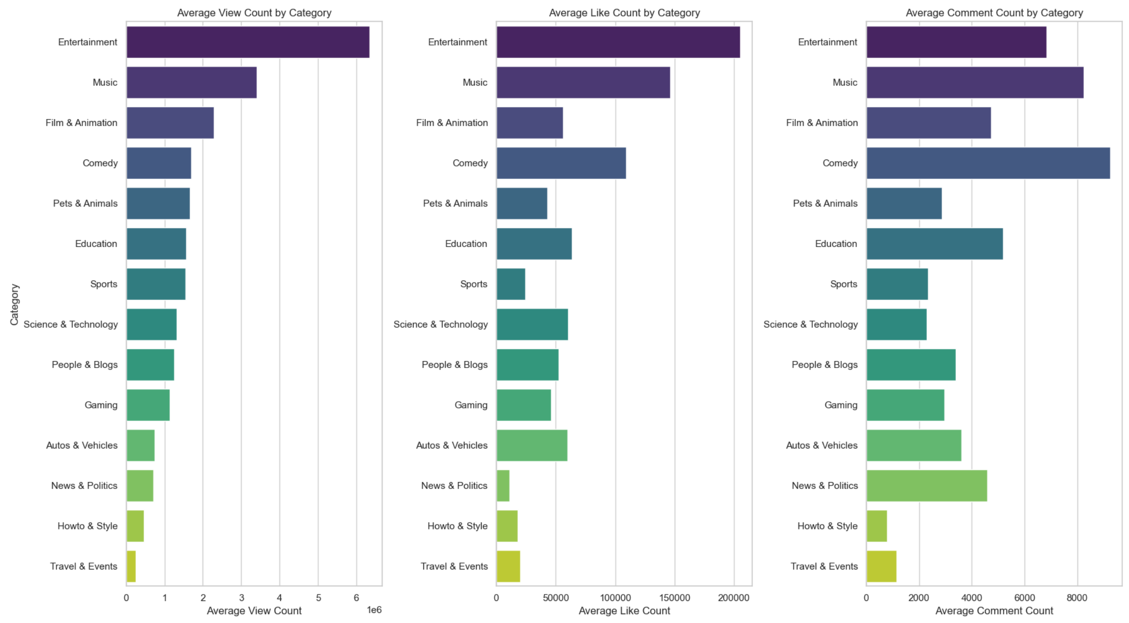
[11]:
```
trending_videos['category_name'] = trending_videos['category_id'].
  ↪map(category_mapping)

# Bar chart for category counts
plt.figure(figsize=(12, 8))
sns.countplot(y=trending_videos['category_name'],␣
  ↪order=trending_videos['category_name'].value_counts().index,␣
  ↪palette='viridis')
plt.title('Number of Trending Videos by Category')
plt.xlabel('Number of Videos')
plt.ylabel('Category')
plt.show()
```



1) The bar chart shows that the Gaming, Entertainment, Sports, and Music categories have the highest number of trending videos.

1.0.5 let's have a look at the average engagement metrics by category:

[12]:
```
# average engagement metrics by category
category_engagement = trending_videos.groupby('category_name')[['view_count',␣
↪'like_count', 'comment_count']].mean().sort_values(by='view_count',␣
↪ascending=False)

fig, axes = plt.subplots(1, 3, figsize=(18, 10))

# view count by category
sns.barplot(y=category_engagement.index, x=category_engagement['view_count'],␣
↪ax=axes[0], palette='viridis')
axes[0].set_title('Average View Count by Category')
axes[0].set_xlabel('Average View Count')
axes[0].set_ylabel('Category')

# like count by category
sns.barplot(y=category_engagement.index, x=category_engagement['like_count'],␣
↪ax=axes[1], palette='viridis')
axes[1].set_title('Average Like Count by Category')
axes[1].set_xlabel('Average Like Count')
axes[1].set_ylabel('')

# comment count by category
sns.barplot(y=category_engagement.index,␣
↪x=category_engagement['comment_count'], ax=axes[2], palette='viridis')
axes[2].set_title('Average Comment Count by Category')
axes[2].set_xlabel('Average Comment Count')
axes[2].set_ylabel('')

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



1) Music and People & Blogs categories have the highest average view counts, likes, and comments. Film & Animation also shows high engagement, especially in view counts and like counts.

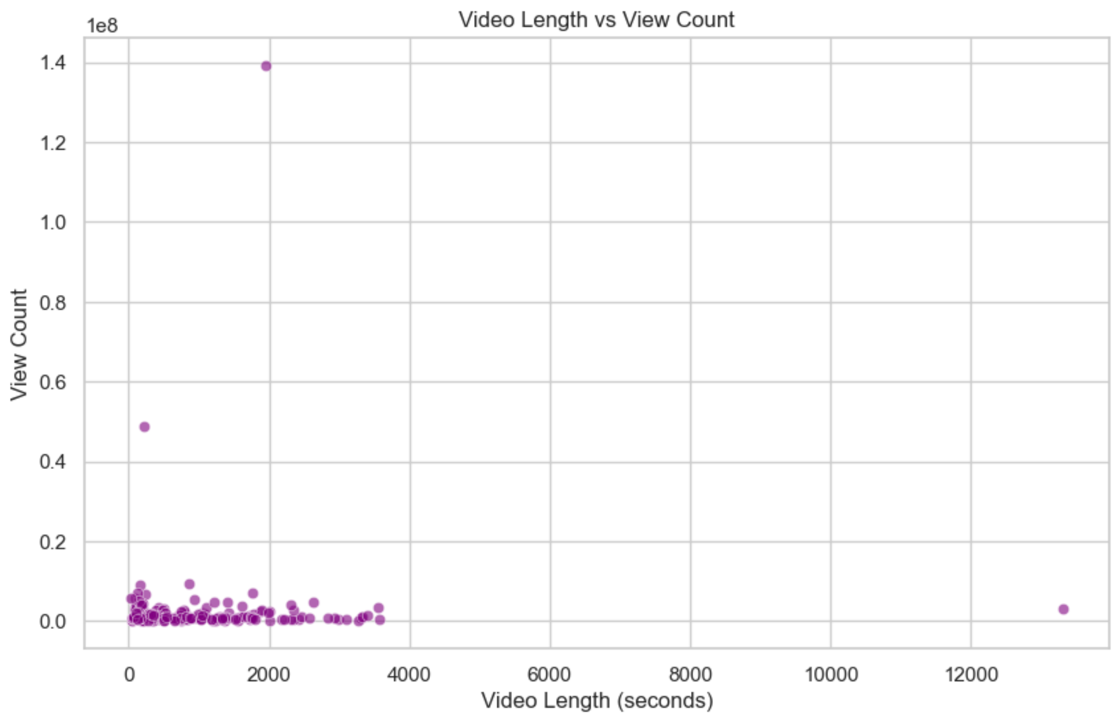
1.0.6 let's analyze the content and duration of the videos. But first, we need to convert the duration from ISO 8601 format to seconds:

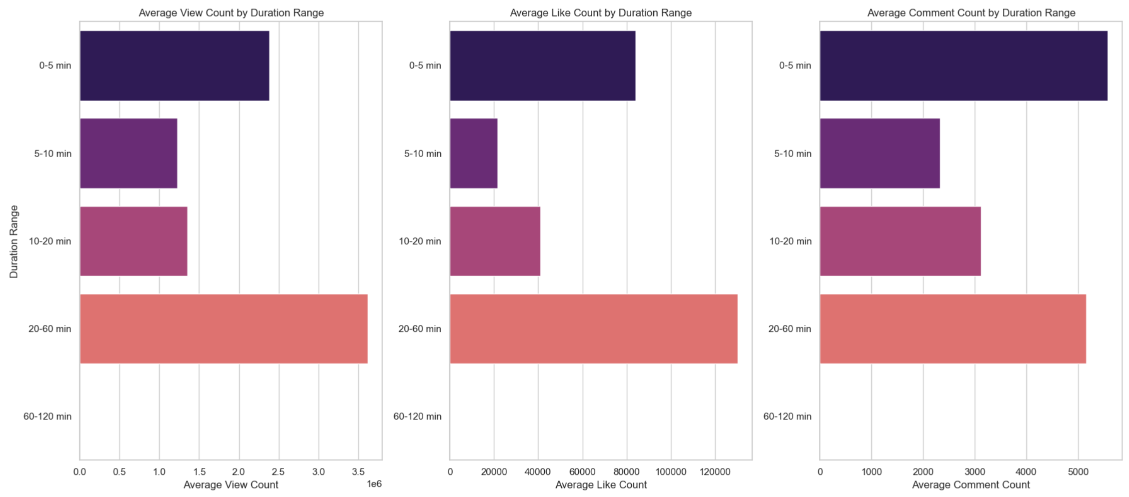
[13]:
```
import isodate

# convert ISO 8601 duration to seconds
trending_videos['duration_seconds'] = trending_videos['duration'].apply(lambda␣
↪x: isodate.parse_duration(x).total_seconds())

trending_videos['duration_range'] = pd.cut(trending_videos['duration_seconds'],␣
↪bins=[0, 300, 600, 1200, 3600, 7200], labels=['0-5 min', '5-10 min', '10-20␣
↪min', '20-60 min', '60-120 min'])
```

1) In the above code, we are using the isodate library to convert the duration of each video from the ISO 8601 format to seconds, which allows for numerical analysis. After converting the durations, we are categorizing the videos into different duration ranges (0-5 minutes, 5-10 minutes, 10-20 minutes, 20-60 minutes, and 60-120 minutes) by creating a new column called duration_range. This categorization enables us to analyze and compare the engagement metrics of videos within specific length intervals, providing insights into how video length influences viewer behaviour and video performance.

1.0.7 let's analyze the content and the duration of videos:

[14]:
```
# scatter plot for video length vs view count
plt.figure(figsize=(10, 6))
sns.scatterplot(x='duration_seconds', y='view_count', data=trending_videos,␣
 ↪alpha=0.6, color='purple')
plt.title('Video Length vs View Count')
plt.xlabel('Video Length (seconds)')
plt.ylabel('View Count')
plt.show()

# bar chart for engagement metrics by duration range
length_engagement = trending_videos.groupby('duration_range')[['view_count',␣
 ↪'like_count', 'comment_count']].mean()

fig, axes = plt.subplots(1, 3, figsize=(18, 8))

# view count by duration range
sns.barplot(y=length_engagement.index, x=length_engagement['view_count'],␣
 ↪ax=axes[0], palette='magma')
axes[0].set_title('Average View Count by Duration Range')
axes[0].set_xlabel('Average View Count')
axes[0].set_ylabel('Duration Range')

# like count by duration range
sns.barplot(y=length_engagement.index, x=length_engagement['like_count'],␣
 ↪ax=axes[1], palette='magma')
axes[1].set_title('Average Like Count by Duration Range')
axes[1].set_xlabel('Average Like Count')
axes[1].set_ylabel('')

# comment count by duration range
sns.barplot(y=length_engagement.index, x=length_engagement['comment_count'],␣
 ↪ax=axes[2], palette='magma')
axes[2].set_title('Average Comment Count by Duration Range')
axes[2].set_xlabel('Average Comment Count')
axes[2].set_ylabel('')

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

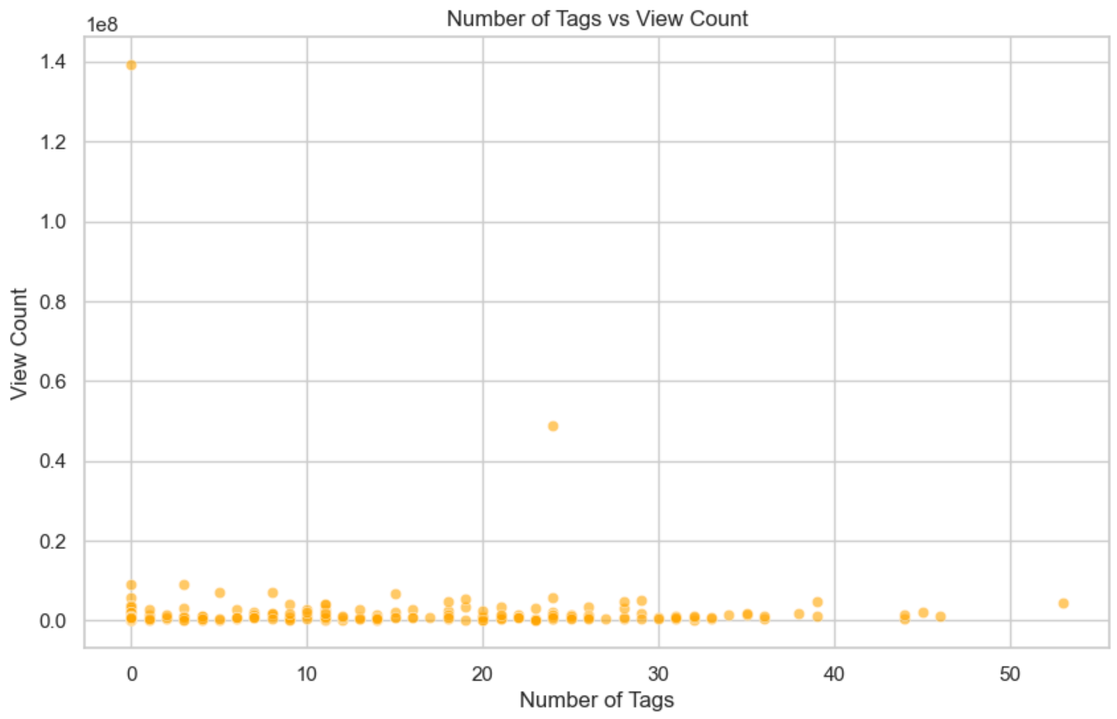




1) The scatter plot shows a slight negative correlation between video length and view count, indicating shorter videos tend to have higher view counts. Videos in the 0-5 minute range have the highest average view counts, likes, and comments. Engagement decreases as video length increases.

1.0.8 let's analyze the relationship between views and number of tags used in the video:

[15]:
```python
# calculate the number of tags for each video
trending_videos['tag_count'] = trending_videos['tags'].apply(len)

# scatter plot for number of tags vs view count
plt.figure(figsize=(10, 6))
sns.scatterplot(x='tag_count', y='view_count', data=trending_videos, alpha=0.6,␣
↪color='orange')
plt.title('Number of Tags vs View Count')
plt.xlabel('Number of Tags')
plt.ylabel('View Count')
plt.show()
```

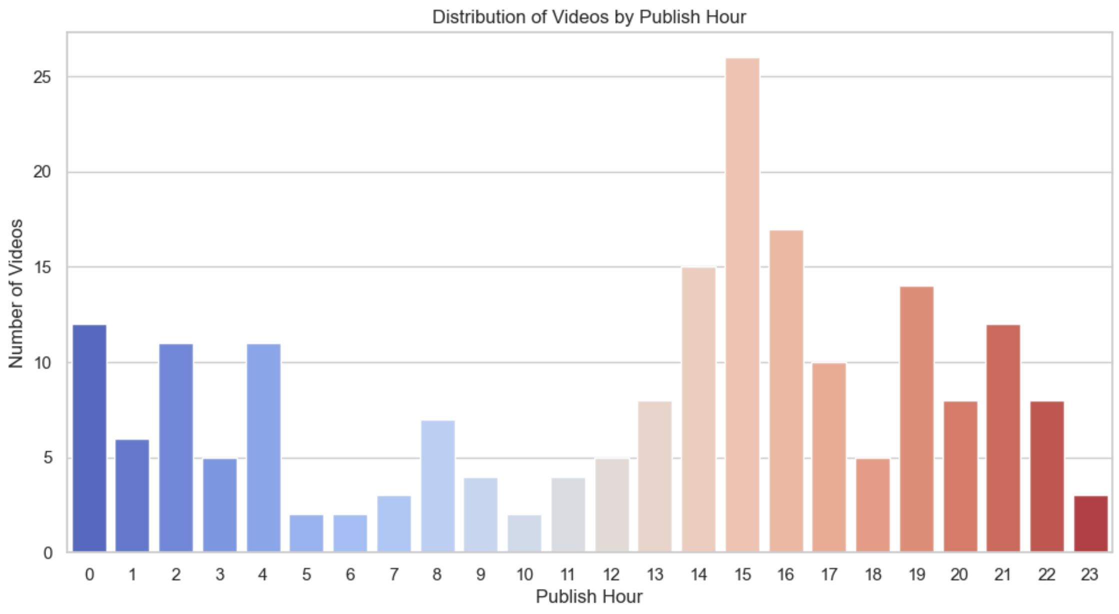


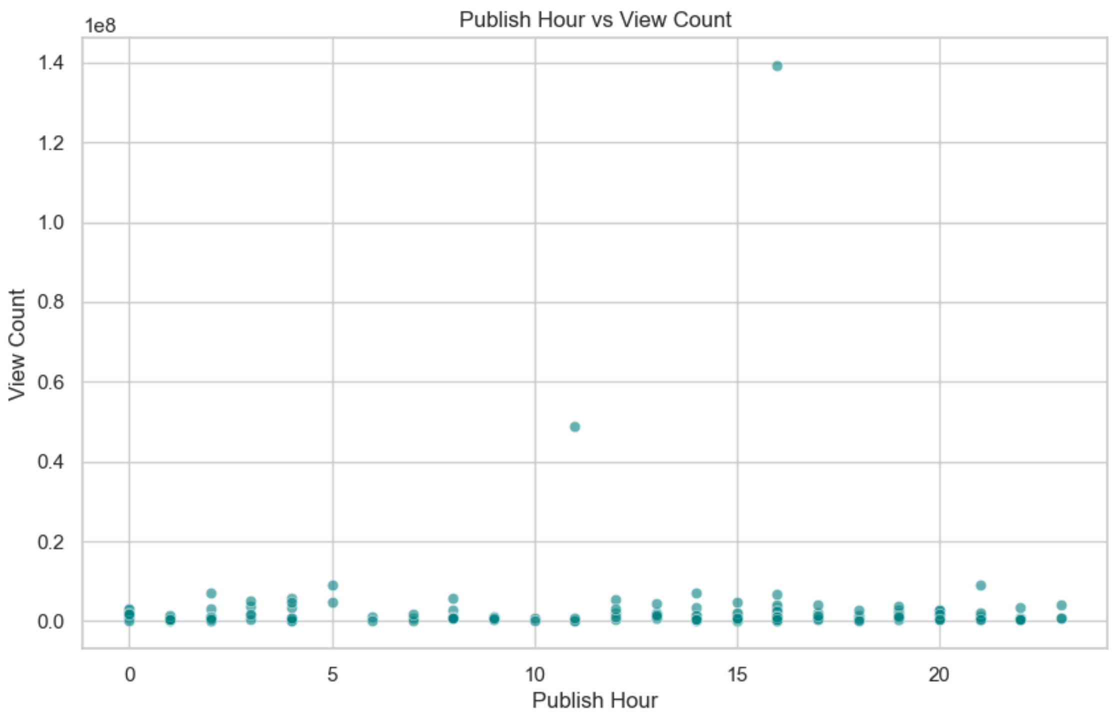
1) The scatter plot shows a very weak relationship between the number of tags and view count, suggesting that the number of tags has minimal impact on a video's view count.

let's see if there's an impact of the time a video is posted on its views:

[16]:
```python
# extract hour of publication
trending_videos['publish_hour'] = trending_videos['published_at'].dt.hour

# bar chart for publish hour distribution
plt.figure(figsize=(12, 6))
```

```
sns.countplot(x='publish_hour', data=trending_videos, palette='coolwarm')
plt.title('Distribution of Videos by Publish Hour')
plt.xlabel('Publish Hour')
plt.ylabel('Number of Videos')
plt.show()

# scatter plot for publish hour vs view count
plt.figure(figsize=(10, 6))
sns.scatterplot(x='publish_hour', y='view_count', data=trending_videos, alpha=0.
 ↪6, color='teal')
plt.title('Publish Hour vs View Count')
plt.xlabel('Publish Hour')
plt.ylabel('View Count')
plt.show()
```





The distribution shows that most videos are published between 14:00 and 20:00 hours (2 PM – 8 PM), indicating this may be an optimal time for uploading videos.

There is a very weak negative relationship between publish hour and view count, suggesting that the hour of publication has minimal impact on engagement metrics.

1.0.9 Conclusion:

1) Encourage viewers to like and comment on videos to boost engagement metrics.
2) Aim to create shorter videos (under 5 minutes) for higher engagement, especially for categories like Music and Entertainment.
3) Schedule video uploads around peak times (2 PM – 8 PM) to maximize initial views and engagement.

```
[ ]:
```