COMP5721M: Programming for Data Science

Coursework 3: Data Analysis Project

Last modified: 3 December 2023

Analysis Of Youtube Trending Data

Give names and emails of group members here:

- Asish Panda, mm23ap@leeds.ac.uk
- Mohamed Imthiyas Abdul Rasheeth, mm23m2ia@leeds.ac.uk
- Naveen Sabarinath Babu, mm23nsb@leeds.ac.uk
- Roshan ., mm23rt@leeds.ac.uk

Project Plan

The Data (10 marks)

YouTube provides an API that provides information about trending data country-wise. The dataset on Kaggle is a real-time (daily) updating dataset derived from the API consisting of attributes for various countries, which is used for analysis in this project. This project's scope limited our use to a specific date range and limited the country to Great Britain.

The dataset consists of two files, _GB_category_id.json and GB_youtube_trending*data.csv*. The files contain the following column ID's

CSV:

The CSV file is approximately 336MB in size and contains 16 columns and 238391 rows of data describing the videos properties.

```
['video_id',
'title',
'publishedAt',
'channelId',
'channelTitle',
'categoryId',
'trending_date',
'tags',
'view_count',
'likes',
'dislikes',
'comment_count',
'thumbnail_link',
'comments_disabled',
```

```
 'ratings_disabled',
 'description']
```

JSON:

The JSON file contains a data structure that links the categoryId column in each file. The JSON file also has information about each file category, i.e. ['family',' Entertainment',' Education']. The structure of the file is as follows. It is approximately 10 KB in size.

```
{
    "kind": "youtube#videoCategoryListResponse",
    "etag": "kBCr3I9kLHHU79W4Ip5196LDptI",
    "items": [
        {
            "kind": "youtube#videoCategory",
            "etag": "IfWa37JGcqZs-jZeAyFGkbeh6bc",
            "id": "1",
            "snippet": {
                "title": "{{category_string}}",
                "assignable":{{boolean}},
                "channelId": "{{string}}"
            }
        },
        {
            "kind": "youtube#videoCategory",
            "etag": "5XGylIs7zkjHh5940dsT5862m1Y",
            "id": "2",
            "snippet": {
                "title": "{{category_string}}",
                "assignable": {{boolean}},
                "channelId": "{{string}}"
            }
        },
   ]
}
```

Reference for the Dataset

Youtube. (2023). *YouTube Trending Video Dataset (updated daily)* [Data set]. Kaggle. https://doi.org/10.34740/KAGGLE/DSV/7112357

Project Aim and Objectives (5 marks)

The primary objective of this project is to perform an in-depth analysis of the "Trending Youtube API Dataset" to identify and understand the essential attributes that result in the uploaded video being "trending" on the platform. The project aims to find the patterns in the dataset that would push the videos uploaded by the users into the trending category, thus helping the content creators optimize their video uploads accordingly, contributing to the YouTube algorithm.

The aim of the project is achieved by using data analysis techniques and machine learning algorithms to find correlations between the different attributes in the dataset. We will use columns such as [categoryId, title,view_count, likes, dislikes,comment_count] to recognize the patterns resulting in a trending video. After a thorough analysis of the various aspects, we aim to discern the commonalities among the trending videos while also identifying the factors that may vary across the different genres of YouTube.

In summary, the project aims to provide insight into the dynamics of YouTube trending videos, thereby helping content creators share content that would satisfy the YouTube algorithm and result in the video trending on the platform. The results will help in improving the experience of the users and the YouTube platform.

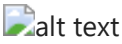
Specific Objective(s)

- **Objective 1:** *Visualize the genres that trend in the United Kingdom*
- **Objective 2:** *Checks the key words to be used in the Title and/or Description for a video to be in the trending category*
- **Objective 3:** *Calculates whether the Likes, Dislikes and View Count influences a video to trend*
- **Objective 4:** *Implementing Regression Models to predict View Counts given Attributes*

  ## System Design (5 marks)

  ### Architecture

The architecture implemented in this project has the following diagram.

 alt text

1. Data Source

The dataset is hosted on Kaggle and comprises two files. The .csv file contains data about each trending video, while the .json file contains information about the video category

2. Data Processing

There are three steps performed while processing the data

- The data is combined from the .csv and .json files into a single pandas data frame.
- The data is scanned for duplicates and null values.
- Columns such as dates are expanded to provide finer details during analysis. #### 3. Data Analysis

The data is analysed according to the first three objectives to determine the attributes of videos that affect their ability to trend on YouTube.

4. Visualisation

The analysis results are visualised at each objective level to understand better the relationship between attributes and the ability of a video to trend on the platform.

5. Modelling

Using information gained during analysis and visualisation, we proceed towards modelling our prediction algorithm.

- Categorical variables are encoded using One Hot Encoding
- Numerical variables are scaled such that they lie between 0 and 1
- A test and train set is created for our regression model
- The model is trained, and its predictions are verified using the test dataset

Processing Modules and Algorithms

Briefly list and describe the most significant computational components of your system and the algorithms you will use to implement them. This could include things like:

- *Combination of data from .csv and .json dataset*
- *WordCloud generation: A word cloud is generated using a library to visualise better any words that are frequently associated with the titles and tags of trending videos*
- *Dummy Variables and Scaling: OneHotEncoding adds dummy variables to categorical data. MinMaxScaler is used to scale numerical data. Together, they are used to create correlation matrices to understand the relationship between dependent and independent variables.*
- *LinearRegression: sklearn's LinearRegression module is used to implement a baseline regression model*
- *RandomForestRegression: sklearn's RandomForestRegression module is used to implement the final regression model that allows us to predict view counts given video attributes*

Program Code (15 marks)

Module Imports

We are importing `pandas` since it is used for DataFrame operations. Along with Pandas, we are also using the built-in `JSON` to import our `GB_category_id.json`

In [5]:
```
import pandas as pd
import json
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
```

Reading And Combining the two files

In the following code block, we read the CSV as `dfc` and transform it into a pandas data frame called `df`. We then read the JSON file into a variable called `categories`. We then select the category title from `categories` and append it into a new column called `category` in the `df` data frame.

We use the `head()` method to see how our data frame looks.

In [6]:
```
dfc= pd.read_csv('GB_youtube_trending_data.csv')
df = pd.DataFrame(dfc)

# Load the categories
with open('GB_category_id.json') as f:
    categories = json.load(f)

# Create a dictionary to map category IDs to category names
category_dict = {int(item['id']): item['snippet']['title'] for item in categories['items

# Map the category IDs in the dataframe to the category names
df['category'] = df['categoryId'].map(category_dict)

df.head()
```

Out[6]:

	video_id	title	publishedAt	channelId	channelTitle	categoryId	trending_date
0	J78aPJ3VyNs	I left youtube for a	2020-08-11T16:34:06Z	UCYzPXprvl5Y-Sf0g4vX-m6g	jacksepticeye	24	2020-08-12T00:00:00Z

		month and THIS is what ha...					
1	9nidKH8cM38	TAXI CAB SLAYER KILLS 'TO KNOW HOW IT FEELS'	2020-08-11T20:00:45Z	UCFMbX7frWZfuWdjAML0babA	Eleanor Neale	27	2020-08-12T00:00:00Z
2	M9Pmf9AB4Mo	Apex Legends | Stories from the Outlands – "Th...	2020-08-11T17:00:10Z	UC0ZV6M2THA81QT9hrVWJG3A	Apex Legends	20	2020-08-12T00:00:00Z
3	kgUV1MaD_M8	Nines - Clout (Official Video)	2020-08-10T18:30:28Z	UCvDkzrj8ZPlBqRd6flxdhTw	Nines	24	2020-08-12T00:00:00Z
4	49Z6Mv4_WCA	i don't know what im doing anymore	2020-08-11T20:24:34Z	UCtinbF-Q-fVthA0qrFQTgXQ	CaseyNeistat	22	2020-08-12T00:00:00Z

Data Quality Check

In [7]:
```
df.shape
```

Out[7]:
```
(238391, 17)
```

We see that there are `17` features (columns) and `238191` trending videos (rows). To ensure correct analysis, we check if we do not have null items in any column by using isna(), which returns a boolean value and then sums it column-wise.

In [8]:
```
df.isna().sum()
```

Out[8]:
```
video_id                0
title                   0
publishedAt             0
channelId               0
channelTitle            0
categoryId              0
trending_date           0
tags                    0
view_count              0
likes                   0
dislikes                0
comment_count           0
thumbnail_link          0
comments_disabled       0
ratings_disabled        0
description          4281
```

```
category              102
dtype: int64
```

The above output shows that `3800` missing descriptions and `92` missing categories. Since missing values in these columns will not cause problems in our analysis, we will not drop those rows.

Duplicate Check

We also check for duplicate rows in the data frame using the pandas' `duplicated()` function, as duplicate values could lead to bias. We then sum up the total to see how many rows have duplicates. From the result, we see that 124 rows have duplicates.

In [9]:
```
df.duplicated().sum()
```

Out[9]:
```
124
```

We now use the `drop_duplicates()` function to drop those rows. We can then check the data frame's shape to cross-verify the deletion of duplicates.

In [10]:
```
df = df.drop_duplicates()
df.shape
```

Out[10]:
```
(238267, 17)
```

Check the important features

Convert the `date` column to datetime format and create new columns for year and month. The months are labeled to their names in English from their respective integer value to enhance clarity.

In [11]:
```
df['date'] = pd.to_datetime(df['trending_date'])
df['month'] = df['date'].dt.month
df['year'] = df['date'].dt.year
df['month'] = df['month'].replace((1,2,3,4,5,6,7,8,9,10,11,12),('Jan','Feb','Mar','Apr',
df['month'] = pd.Categorical(df['month'], categories=['Jan','Feb','Mar','Apr','May','Jun
```

We will convert the `publishedAt` column to Date-time datatype, since it is in object datatype currently. Then, we will take the published month and hour and append them in the two new columns called `num_month` and `hour` respectively. We have chosen `publishedAt` since the hour in trending videos were 00:00.

In [12]:
```
df['test_hour'] = df['date'].dt.hour
df['test_hour'].unique()
```

Out[12]:
```
array([0])
```

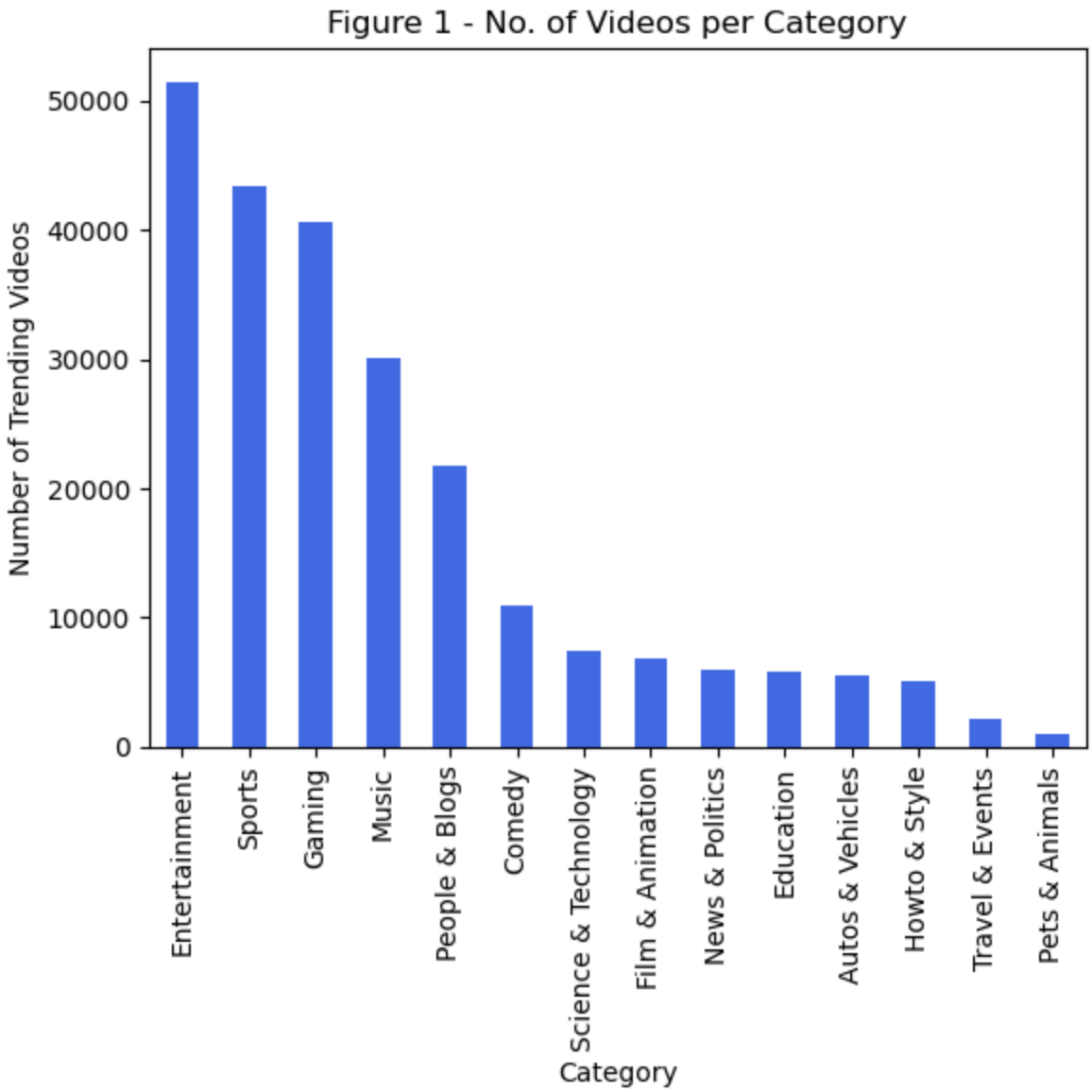
In [13]:
```
df['publishedAt'] = pd.to_datetime(df['publishedAt'])
df['num_month'] = df['publishedAt'].dt.month
df['hour'] = df['publishedAt'].dt.hour
```

Objective 1 Code Cells: *Visualize the genres that trend in the United Kingdom*

Firstly, we will explore which category of video trends the most, to comprehend the kind of videos the viewers are more interested in viewing. We plot a `barplot` that shows the number of videos trending in each category.

In [14]:
```
df['category'].value_counts().plot.bar(title='Figure 1 - No. of Videos per Category',xla
```

Out[14]:
```
<Axes: title={'center': 'Figure 1 - No. of Videos per Category'}, xlabel='Category', yla
bel='Number of Trending Videos'>
```



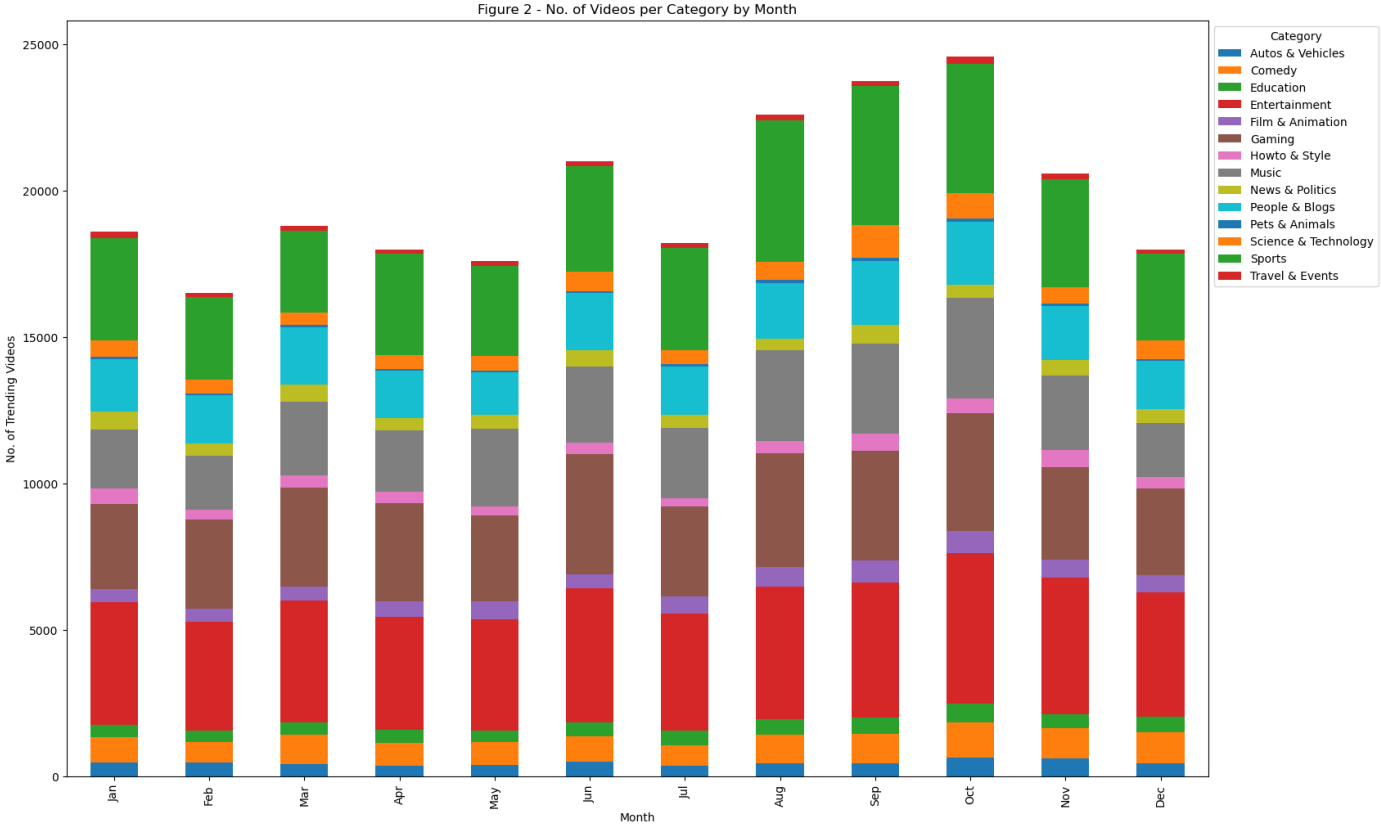
As we can see from Figure 1, Entertainment tops the list, followed by Sports, Gaming, Music, and People & Blogs as the Top 5 Categories across the dataset for the year between Aug 2020 and Nov 2023.

Now that we know the ranking of the trending categories, we further investigate the nature of the trending categories month-wise. A graph is plotted based on category and month to show the results of the investigation.

In [15]:
```
#plot bar graph based on category and date by month
df.groupby(['month','category']).size().unstack().plot(kind='bar',stacked=True,figsize=(
```

Out[15]:
```
<matplotlib.legend.Legend at 0x1c143d1d810>
```

Figure 2 - No. of Videos per Category by Month



We are using a stacked barplot to compare the number of trending videos each month. As shown in Figure 2, October has the highest number of trending videos, followed by September and August. The least number of trending videos is for February. The top 5 categories from Figure 1 have almost a similar spread across the plot.

The spike in the number of trending videos between August and November could be because of the range of months of the trending videos in the dataset. That is, our range is between August 2020 to November 2023. We will further do analysis into finding how big of a difference is produced between each year by comparing only the top 5 categories.

The data frame is further filtered to the Top 5 categories per Figure 1.

In [16]:
```
# plot grouped bar graph for top 5 category per year
df.groupby(['year','category']).size().groupby(level=0).nlargest(5).unstack().plot(kind=
plt.xticks(np.array([0,1,2,3]), ('2020','2021','2022','2023'),rotation=0)
```

Out[16]:
```
([<matplotlib.axis.XTick at 0x1c14470ed50>,
  <matplotlib.axis.XTick at 0x1c1446960d0>,
  <matplotlib.axis.XTick at 0x1c151149790>,
  <matplotlib.axis.XTick at 0x1c15114b410>],
 [Text(0, 0, '2020'),
  Text(1, 0, '2021'),
  Text(2, 0, '2022'),
  Text(3, 0, '2023')])
```

Figure 3 - Top 5 Categories per Year

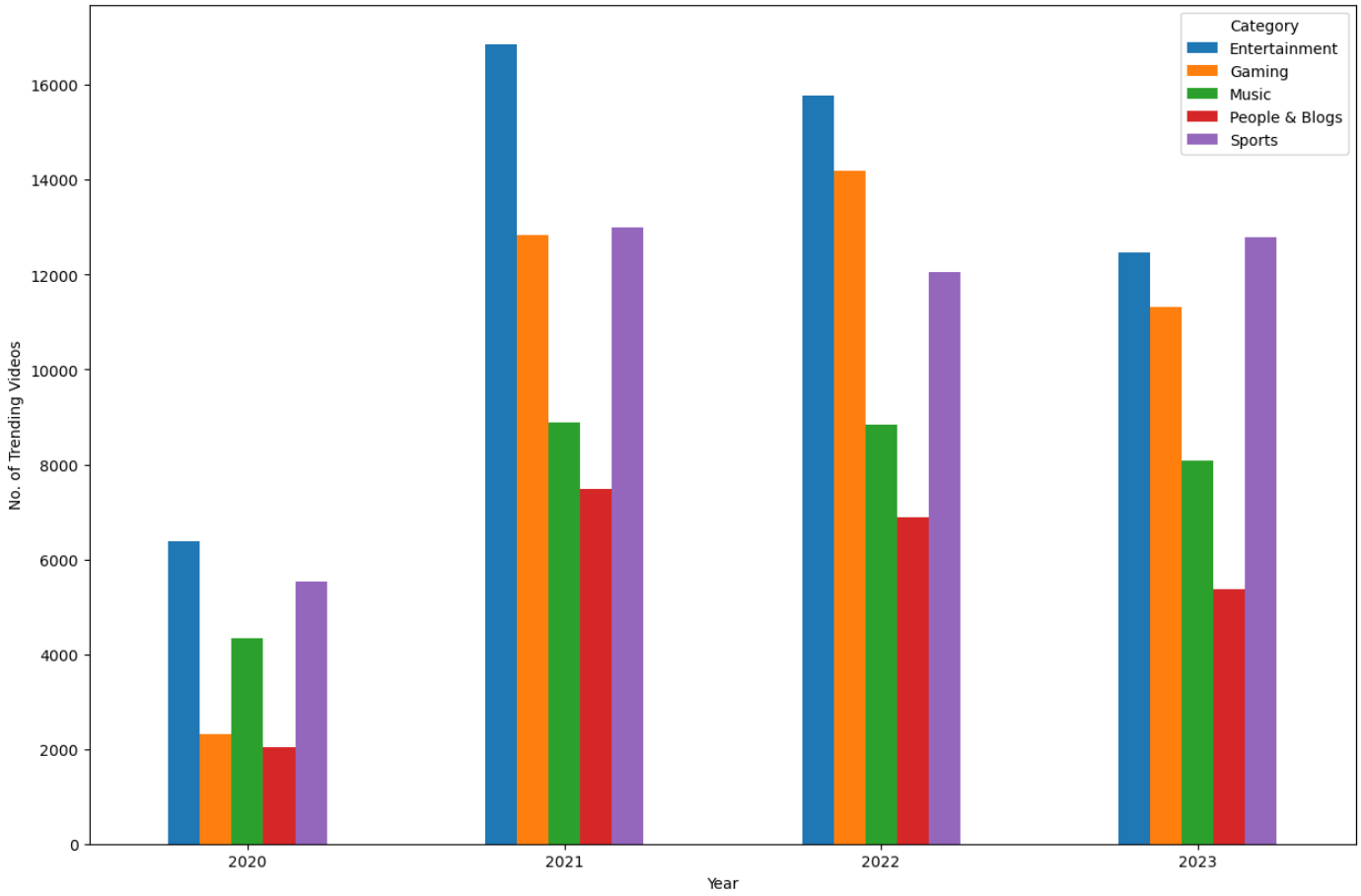


Figure 3 shows that in 2020, there were fewer trending videos than in any other year. As for 2023, the number of trending videos is also less compared to 2021 and 2022. This shows that since we have incomplete data for 2020 and 2023, the height is reduced, which made it spike between August and November in Figure 2.

Objective 2 Code Cells: *Checks the key words to be used in the Title and/or Description for a video to be in the trending category*

To aide in our understanding of how keywords in title affect a videos abilty to trend, we use an external python module called `worldcloud` *. This will allow us to visualize frequently mentioned words in the title.*

In [17]:
```
!pip install wordcloud
```

```
Requirement already satisfied: wordcloud in c:\users\asish\anaconda3\lib\site-packages
(1.9.2)
Requirement already satisfied: numpy>=1.6.1 in c:\users\asish\anaconda3\lib\site-package
s (from wordcloud) (1.24.3)
Requirement already satisfied: pillow in c:\users\asish\anaconda3\lib\site-packages (fro
m wordcloud) (9.4.0)
Requirement already satisfied: matplotlib in c:\users\asish\anaconda3\lib\site-packages
(from wordcloud) (3.7.2)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\asish\anaconda3\lib\site-pac
kages (from matplotlib->wordcloud) (1.0.5)
Requirement already satisfied: cycler>=0.10 in c:\users\asish\anaconda3\lib\site-package
s (from matplotlib->wordcloud) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\asish\anaconda3\lib\site-pa
ckages (from matplotlib->wordcloud) (4.25.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\asish\anaconda3\lib\site-pa
ckages (from matplotlib->wordcloud) (1.4.4)
Requirement already satisfied: packaging>=20.0 in c:\users\asish\appdata\roaming\python
\python311\site-packages (from matplotlib->wordcloud) (23.2)
```

```
Requirement already satisfied: pyparsing<3.1,>=2.3.1 in c:\users\asish\anaconda3\lib\site-packages (from matplotlib->wordcloud) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\asish\appdata\roaming\python\python311\site-packages (from matplotlib->wordcloud) (2.8.2)
Requirement already satisfied: six>=1.5 in c:\users\asish\appdata\roaming\python\python311\site-packages (from python-dateutil>=2.7->matplotlib->wordcloud) (1.16.0)
```

In [18]:
```
from wordcloud import WordCloud
```

_We will calculate the total number of words in the title using the split() and len() functions and store them in a new column called title_length.Then, we take the mean value of title*length to see how many average words are there in each title.*

In [19]:
```
#find the average length of title in words
df_test3 = df.copy()
df_test3['title_length'] = df_test3['title'].str.split().str.len()
df_test3['title_length'].mean()
```

Out[19]:
```
9.04523496749445
```

We plot a histogram for `title_length` with 20 equal intervals.The data points are collected in the given interval, where the x-axis represents the number of words in the title and y-axis represents its count.

In [20]:
```
df_test3['title_length'].plot.hist(bins=20,figsize=(10,5),title="Figure 4 - Distribution
```

Out[20]:
```
<Axes: title={'center': 'Figure 4 - Distribution of Title Length'}, xlabel='Title Length', ylabel='Frequency'>
```

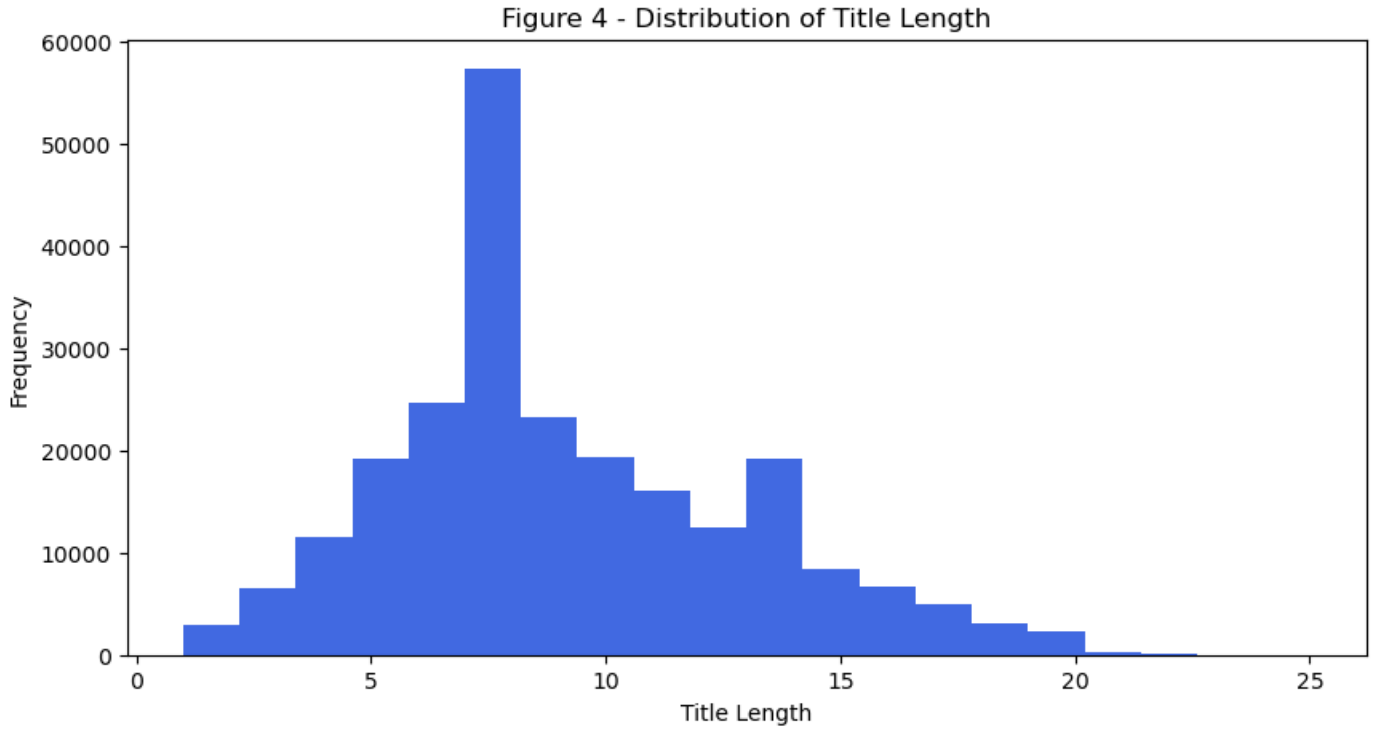


Figure 4 shows that most of the videos that made it into the trending list has around 6 to 8 words. Thus, we could say that having either a very low or a very high word count in titles may not be appealing towards the viewers.

Now, we will create a word cloud to see the most used keywords for the title in the trending video for each category.

_ A text variable is created that takes the words in the `title` column based on categories and if the text is not empty, then generate the wordclouds for each category.

In [21]:
```
# Multiple graphs in the same cell
fig, axes = plt.subplots(7, 2, figsize=(15, 30))

# Get unique categories
categories = df_test3['category'].unique()

# Limit the number of categories to the number of subplots
categories = categories[:len(axes.flatten())]

# Plot the wordcloud for each category
for i, ax in enumerate(axes.flatten()):
    category = categories[i]
    title = df_test3[df_test3['category'] == category]['title']
    text = ' '.join(title)
    if text != ' ':
        wordcloud = WordCloud(max_font_size=50, max_words=10000, background_color="white
        ax.imshow(wordcloud, interpolation="bilinear")
        ax.set_title(category)
        ax.axis("off")
        fig.suptitle('Figure 5 - Wordcloud of Trending Video Titles by Category',fontsiz
        fig.tight_layout(rect=[0, 0.03, 1, 0.95])
```

```
C:\Users\asish\AppData\Local\Temp\ipykernel_15344\1344624593.py:21: UserWarning: The fig
ure layout has changed to tight
  fig.tight_layout(rect=[0, 0.03, 1, 0.95])
```

Figure 5 - Wordcloud of Trending Video Titles by Category

Figure 5 shows the words that appears the most in each category. Considering the top 5 categories (from Objective 1), we can conclude that these words would increase the chances of the publishers video to make it into the trending list:

- Entertainment : Ofiicial Trailer, Manchester United, Hot ones, Beta Squad, Official Teaser etc.
- Sports : Premier League, Manchester United, League Highlights, Grand Prix, Man City etc.
- Gaming : Minecraft, HARDCORE Minecraft, Minecraft Hardcore, Survived Days, Among Us etc.
- Music : Official Music, Music Video, Offical Video, Lyric Video, Official Lyric etc.
- People and Blogs : Official Video, Among Us, Short, Sidemen Among Us, Music Video etc.

Words like ‘Slow Mo’ is of lesser significance since they are of a smaller font

In [22]:
```
df_test4 = df.copy()
df_test4['tag_length'] = df_test3['tags'].str.split().str.len()
df_test4['tag_length'].mean()
```

Out[22]: 17.736253866460736

Here, will create a word cloud to see the most used keywords for the tags in the trending video for each category.

A text variable is created that takes the words in the `tags` *column based on categories and if the text is not empty, then generate the wordclouds for each category.*

In [23]:
```
# Multiple graphs in the same cell
fig, axes = plt.subplots(7, 2, figsize=(15, 30))

# Get unique categories
categories = df_test3['category'].unique()

# Limit the number of categories to the number of subplots
categories = categories[:len(axes.flatten())]

# Plot the wordcloud for each category
for i, ax in enumerate(axes.flatten()):
    category = categories[i]
    tags = df_test3[df_test3['category'] == category]['tags']
    text = ' '.join(tags)
    if text != ' ' or text == 'None':
        wordcloud = WordCloud(max_font_size=50, max_words=10000, background_color="white
        ax.imshow(wordcloud, interpolation="bilinear")
        ax.set_title(category)
        ax.axis("off")
        fig.suptitle('Figure 6 - Wordcloud of Trending Video tags by Category',fontsize=
        fig.tight_layout(rect=[0, 0.03, 1, 0.95])
```

```
C:\Users\asish\AppData\Local\Temp\ipykernel_15344\1958005003.py:21: UserWarning: The fig
ure layout has changed to tight
  fig.tight_layout(rect=[0, 0.03, 1, 0.95])
```

Figure 6 - Wordcloud of Trending Video tags by Category

Figure 6 shows the wordcloud for the most appearing words in tags, segregated by categories. Considering the top 5 categories (from Objective 1), we can conclude that these words would increase the chances of the publishers video to make it into the trending list:

- Entertainment : Manchester United, Man Utd, United Man, Beta Squad, Transfer News etc.
- Sports : Sky Sport, Premier League, Manchester United, League football, Man Utd etc.
- Gaming : Minecraft, battle royale, genshin impact, apex legends, Among Us etc.
- Music : music video, hip hop, ed sheeran, taylor swift, lil durk etc.

- People and Blogs : None, Sidemen, moreSidemen, calorie challenge, eating challenge etc.

Figures 5, 6 and 7 explain the most used keywords based on their title and tag, where it has a common word in both wordclouds.

Next, we are creating a function `wordcloud_gen_v2()` that contains frequently used words in both attributes(tags and title).

This function takes the tags and titles and segregates it based on categories. The `common_words` function finds the common words in both the texts variables, which is then ran through a for loop to generate a word cloud based on categories

In [24]:
```
def wordcloud_gen_v2(category):

    text1 = ' '.join(df_test3[df_test3['category'] == category]['title'])
    text2 = ' '.join(df_test3[df_test3['category'] == category]['tags'])

    from collections import Counter

    # Generate word frequency dictionaries
    word_freq1 = Counter(text1.split())
    word_freq2 = Counter(text2.split())

    # Find common words
    common_words = word_freq1 & word_freq2

    # Generate a word cloud using only the common words
    common_text = ' '.join(set(common_words.elements()))
    if common_text != ' ' or common_text == 'None':
        wordcloud = WordCloud(max_font_size = 50,max_words=10000, background_color="whit
    return wordcloud
```

After grouping up the identical words of both attributes, We create a word cloud to show the similar words of trending videos by category.

In [25]:
```
# Multiple graphs in the same cell
fig, axes = plt.subplots(7, 2, figsize=(15, 30))

# Get unique categories
categories = df_test3['category'].unique()

# Limit the number of categories to the number of subplots
categories = categories[:len(axes.flatten())]

# Plot the wordcloud for each category
for i, ax in enumerate(axes.flatten()):
    category = categories[i]
    #tags = df_test3[df_test3['category'] == category]['tags']
    #text = ' '.join(tags)

    if text != ' ' or text == 'None':
        ax.imshow(wordcloud_gen_v2(category), interpolation="bilinear")
        ax.set_title(category)
        ax.axis("off")
        fig.suptitle('Figure 7 - Wordcloud of Common Words in Trending Video Tags and Ti
        fig.tight_layout(rect=[0, 0.03, 1, 0.95])
```

```
C:\Users\asish\AppData\Local\Temp\ipykernel_15344\2180393670.py:21: UserWarning: The fig
ure layout has changed to tight
  fig.tight_layout(rect=[0, 0.03, 1, 0.95])
```

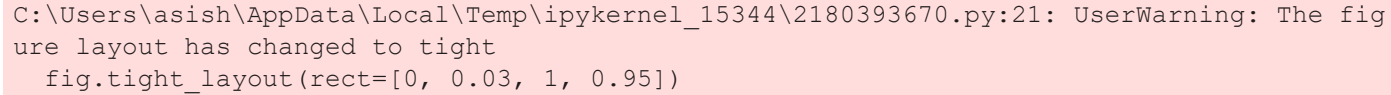
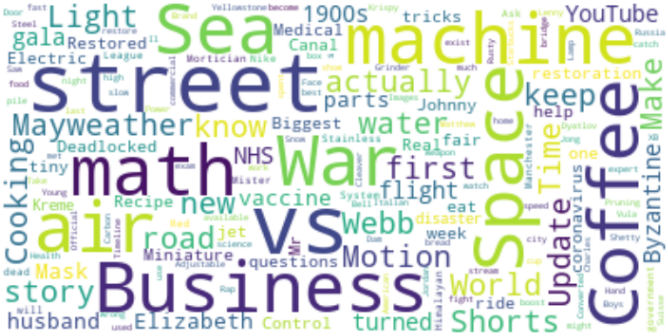


Figure 7 - Wordcloud of Common Words in Trending Video Tags and Title by Category

Entertainment

Education



Gaming

People & Blogs

Music

Howto & Style

Sports

News & Politics

Comedy

Film & Animation

Science & Technology

Travel & Events

Figure 7 shows common keywords in tags and titles for each category where the most repeated keywords are visualized as larger text and the least repeated as smaller text. We can conclude that these words would increase the chances of the publishers video to make it into the trending list if they are used in both titles and tags.

- Entertainment : Challenge, Hour, Year, Youtuber, Official etc.
- Sports : Fight, Final, GAME, Transfer, team etc.
- Gaming : Minecraft, Secret, World, Item, FIFA etc.
- Music : Official, ft, feat, song, Album etc.
- People and Blogs : Short, HOUSE, World, Dream, Videovlog etc.

Objective 3 Code Cells

We have to see all the columns but use only the ones we need for objective 3. Thus, we use the column method that returns the column names in the data frame.

In [26]:
```
df.columns
```

Out[26]:
```
Index(['video_id', 'title', 'publishedAt', 'channelId', 'channelTitle',
       'categoryId', 'trending_date', 'tags', 'view_count', 'likes',
       'dislikes', 'comment_count', 'thumbnail_link', 'comments_disabled',
       'ratings_disabled', 'description', 'category', 'date', 'month', 'year',
       'test_hour', 'num_month', 'hour'],
      dtype='object')
```

Drop the unnecessary columns to explore the data in the `df_ob3` dataset; using the `drop()` method.

In [27]:
```
df_ob3 = df.copy()
df_ob3 = df.drop(['video_id','channelId','tags','channelTitle','thumbnail_link','comment
df_ob3.head()
```

Out[27]:

	title	publishedAt	categoryId	trending_date	view_count	likes	dislikes	comment_count	categor
0	I left youtube for a month and THIS is what ha...	2020-08-11 16:34:06+00:00	24	2020-08-12T00:00:00Z	2038853	353790	2628	40228	Entertainmer

1	TAXI CAB SLAYER KILLS 'TO KNOW HOW IT FEELS'	2020-08-11 20:00:45+00:00	27	2020-08-12T00:00:00Z	236830	16423	209	1642	Educatio
2	Apex Legends | Stories from the Outlands – "Th...	2020-08-11 17:00:10+00:00	20	2020-08-12T00:00:00Z	2381688	146739	2794	16549	Gamin
3	Nines - Clout (Official Video)	2020-08-10 18:30:28+00:00	24	2020-08-12T00:00:00Z	613785	37567	669	2101	Entertainmer
4	i don't know what im doing anymore	2020-08-11 20:24:34+00:00	22	2020-08-12T00:00:00Z	940036	87113	1860	7052	People Blog

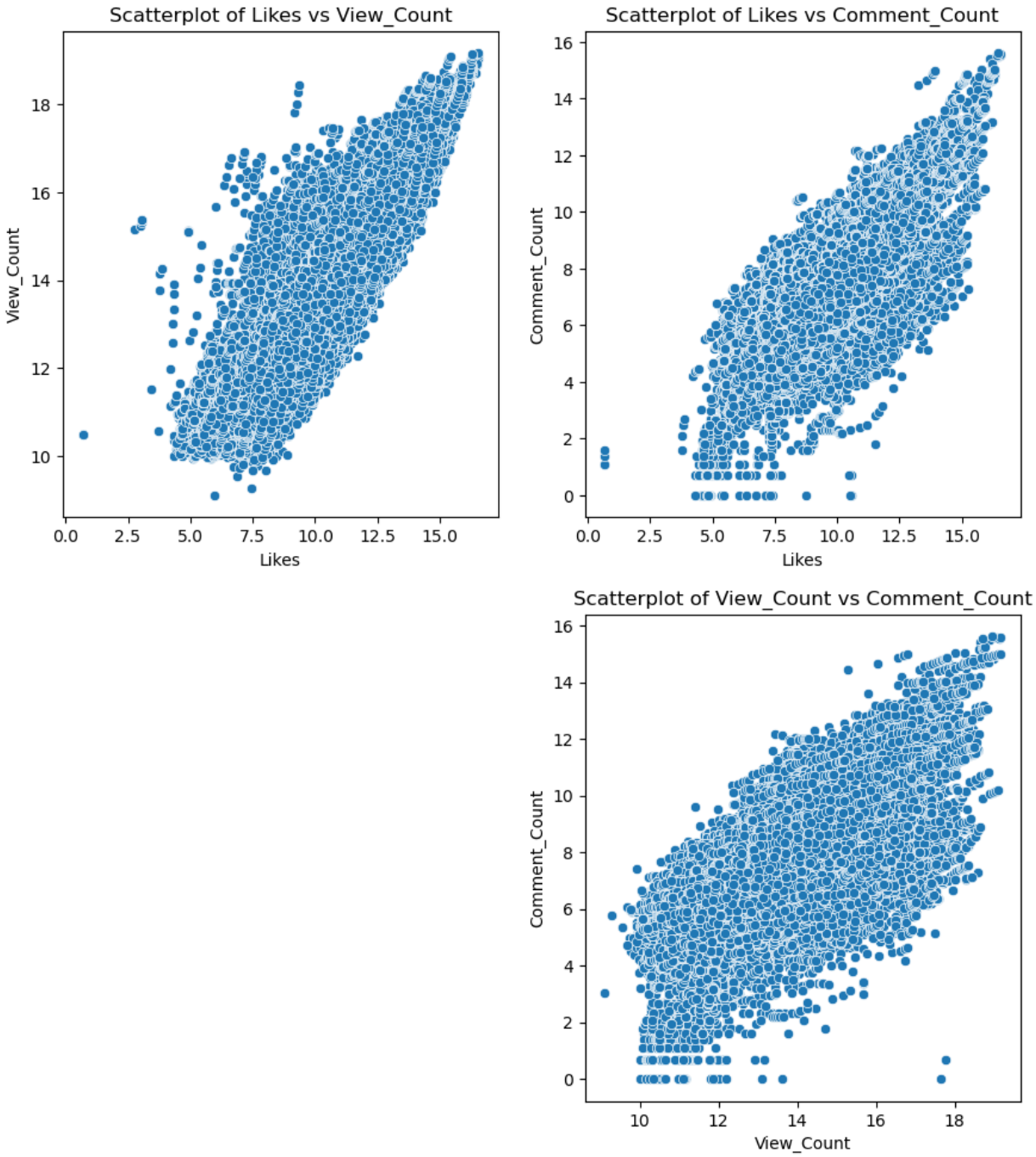
We now check the relation between likes, views, and comment count using scatterplot.

In [28]:
```
columns = ['likes', 'view_count', 'comment_count']
plt.figure(figsize=(16, 18))
#suptitle and align title to center
plt.suptitle('Figure 8 - Scatterplot of Likes, View Count and Comment Count',fontsize=20
#plt.suptitle('Figure 8 - Scatterplot of Likes, View Count and Comment Count',fontsize=2

for i in range(len(columns)):
    for j in range(i+1, len(columns)):
        plt.subplot(len(columns), len(columns), i*len(columns) + j + 1)
        sns.scatterplot(x=np.log(df[columns[i]]), y=np.log(df[columns[j]])).set(title=f'
```

```
c:\Users\asish\anaconda3\Lib\site-packages\pandas\core\arraylike.py:396: RuntimeWarning:
divide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
c:\Users\asish\anaconda3\Lib\site-packages\pandas\core\arraylike.py:396: RuntimeWarning:
divide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
c:\Users\asish\anaconda3\Lib\site-packages\pandas\core\arraylike.py:396: RuntimeWarning:
divide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

Figure 8 - Scatterplot of Likes, View Count and Comment Count



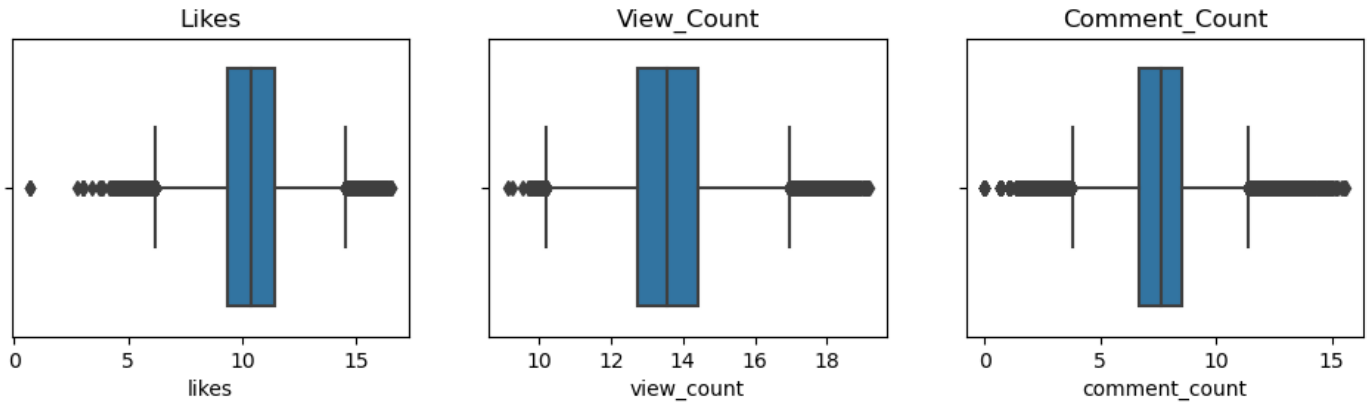
In Figure 8, we see that likes, view count, and comment count have a linear relationship, which means if the likes increase, the view count increases, and then the comment count also increases.

We then look deeper into likes and view counts for each category and look for the similarity. However, we also dropped the null rows.

In [29]:
```
columns = ['likes', 'view_count', 'comment_count']
plt.figure(figsize=(16, 6))
plt.suptitle('Figure 9 - Boxplot of Likes, View Count and Comment Count', fontsize=12,po
plt.subplots_adjust(top=0.85)
for i, column in enumerate(columns):
    plt.subplot(2, 4, i + 1)
    sns.boxplot(x=np.log(df[column]))
    plt.title(column.title())
```

```
c:\Users\asish\anaconda3\Lib\site-packages\pandas\core\arraylike.py:396: RuntimeWarning:
divide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
c:\Users\asish\anaconda3\Lib\site-packages\pandas\core\arraylike.py:396: RuntimeWarning:
divide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
c:\Users\asish\anaconda3\Lib\site-packages\pandas\core\arraylike.py:396: RuntimeWarning:
divide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

Figure 9 - Boxplot of Likes, View Count and Comment Count



Plotting the boxplot with the actual count of likes, view_count and comment_count did not provide a graph that was convenient to read. Therefore, we have opted to take the logarithm of each values in the respective columns to better visualise it, which can be seen in Fig 9.

Relation between view count and category

In [30]:
```
df_ob3 = df_ob3.dropna()

plt.scatter(df_ob3['view_count'],df_ob3['category'].astype(str))
plt.xlabel('View Count [in 100 million]')
plt.ylabel('Category')
plt.title('Figure 10 - View count vs. Category')
plt.show()
```

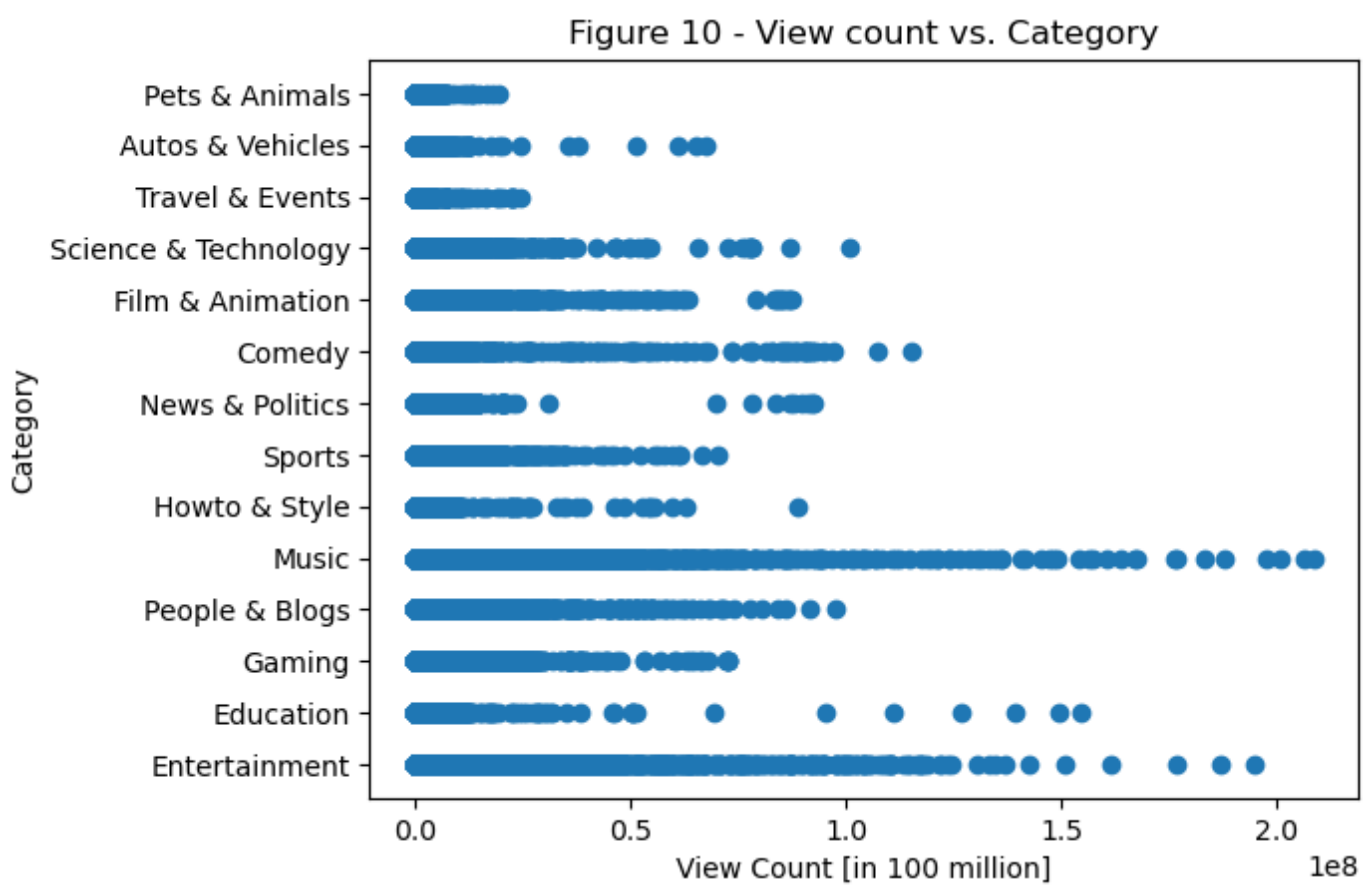


Figure 10 shows the view count in millions for each category, where Music has the highest number of views, followed by Entertainment and Education.

Relation between likes and category

In [31]:
```
plt.scatter(df_ob3['likes'],df_ob3['category'].astype(str))
plt.xlabel('Likes [in 100 million]')
plt.ylabel('Category')
plt.title('Figure 11 - Likes vs. Category')
plt.show()
```

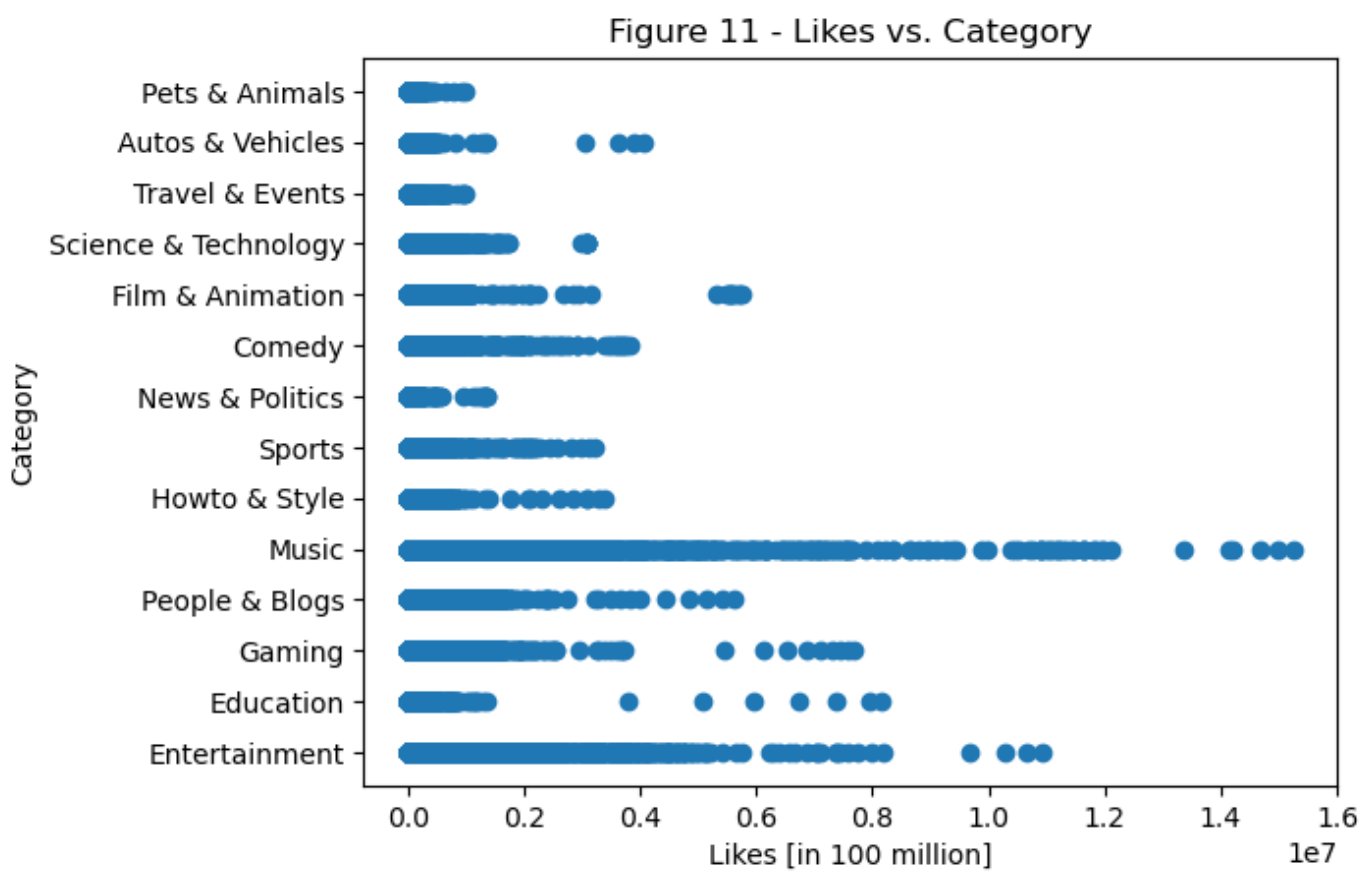


Figure 11 shows the likes in millions for each category, where Music has the most likes, followed by Entertainment and Education.

These two graphs show similar results, so looking from the perspective of view_count alone provides similar results. We chose view count since the values are higher than that of likes.

We then count the number of videos that are higher than the mean of the view count.

In [32]:
```
df_ob3['title'][df_ob3['view_count'] > df_ob3['view_count'].mean()].count()
```

Out[32]:
```
51065
```

In [33]:
```
df_ob3['title'][df_ob3['likes'] > df_ob3['likes'].mean()].count()
```

Out[33]:
```
49285
```

In [34]:
```
df_ob3['title'][df_ob3['comment_count'] > df_ob3['comment_count'].mean()].count()
```

Out[34]:
```
38395
```

The mean of the number of view counts is higher than the number of likes and the number of comments in the dataset. Since we have shown that all three would provide almost similar results, we have chosen to go with view_count column alone for further analysis.

In [35]:
```
df_ob3['title'][df_ob3['view_count'] == 0].count()
```

Out[35]:
```
94
```

Here we see 94 videos have made it to the trending dataset with 0 view count, so these can be considered as anomalies. From graphs, higher the view count -> more chances of it being recommended to users in that particular category than the ones with lower view count.

Three new columns have been added for visualisation purpose - `view_count_mean` , `view_count50` and `view_count_25` .

_view_count_mean - returns 1 if the view_count is greater than the mean of view_count _view_count_50 - returns 1 if view_count is greater than the .5 quantile or 50th percentile of the data view_count_25 - returns 1 if view_count is greater than the .25 quantile or 50th percentile of the data

In [36]:
```
df_ob3['view_count_mean'] = (df_ob3['view_count'] > df_ob3['view_count'].mean()).astype(
df_ob3['view_count_50'] = (df_ob3['view_count'] > df_ob3['view_count'].quantile(.5)).ast
df_ob3['view_count_25'] = (df_ob3['view_count'] > df_ob3['view_count'].quantile(.25)).as
```

In [37]:
```
df_ob3['view_count'].quantile(.5)
```

Out[37]:
```
781351.0
```

In [38]:
```
df_ob3['view_count'].quantile(.25)
```

Out[38]:
```
343374.0
```

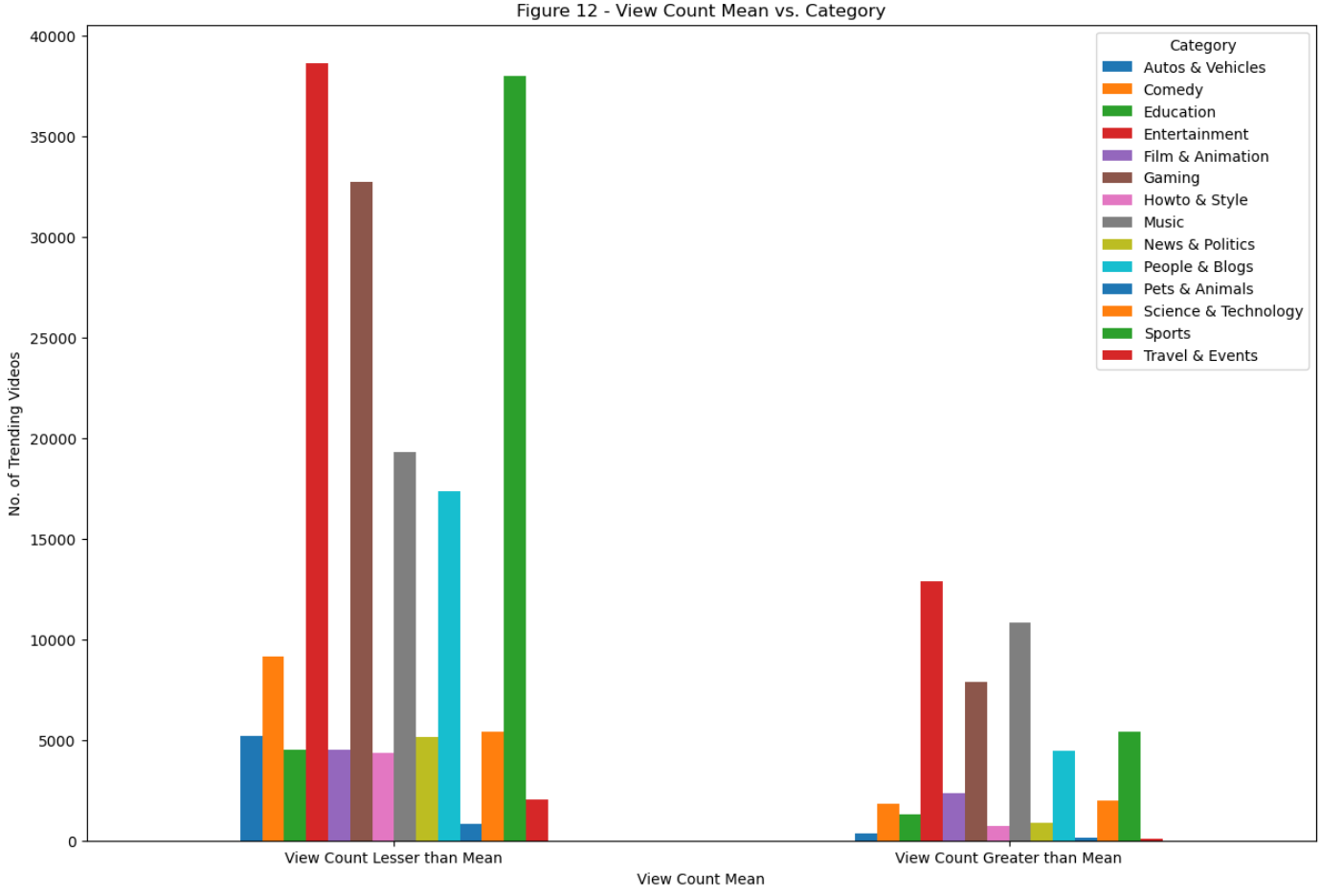
In [39]:
```
df_ob3['view_count'].mean()
```

Out[39]:
```
2168375.596808935
```

In [40]:
```
df_ob3.groupby(['view_count_mean','category']).size().unstack().plot(kind='bar',stacked=
```

Out[40]:
```
[Text(0, 0, 'View Count Lesser than Mean'),
 Text(1, 0, 'View Count Greater than Mean')]
```

Figure 12 - View Count Mean vs. Category



In Figure 12, we see that the majority of the videos have a view count that is lesser than the mean. We can conclude that vidoes are not required to accumulate views that cross the mean of the view counts of the trending videos (2168369.3647721303)

In [41]:
```
df_ob3.groupby(['view_count_50','category']).size().unstack().plot(kind='bar',stacked=Fa
```

Out[41]:
```
[Text(0, 0, 'View Count Lesser than 50%'),
 Text(1, 0, 'View Count Greater than 50%')]
```

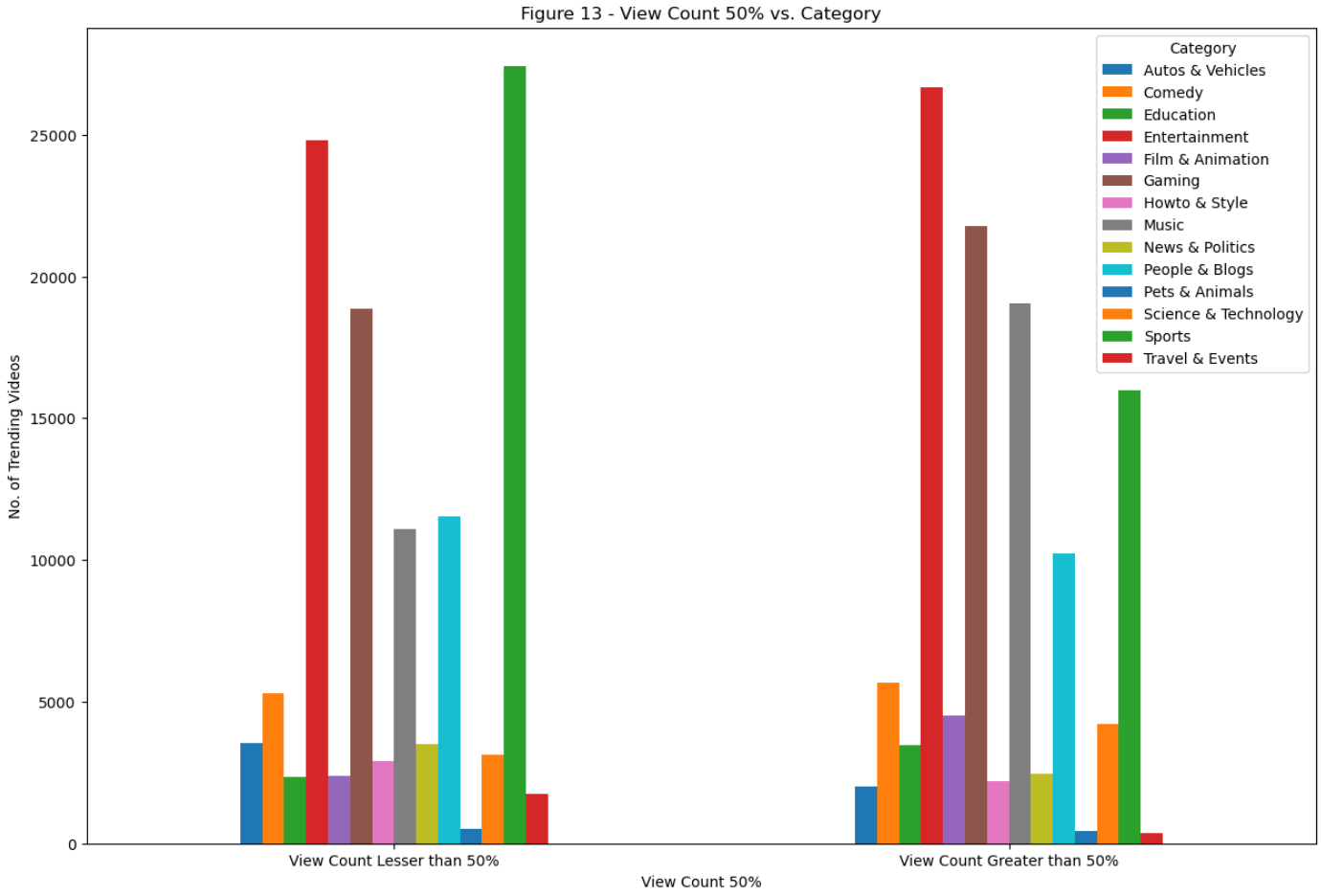
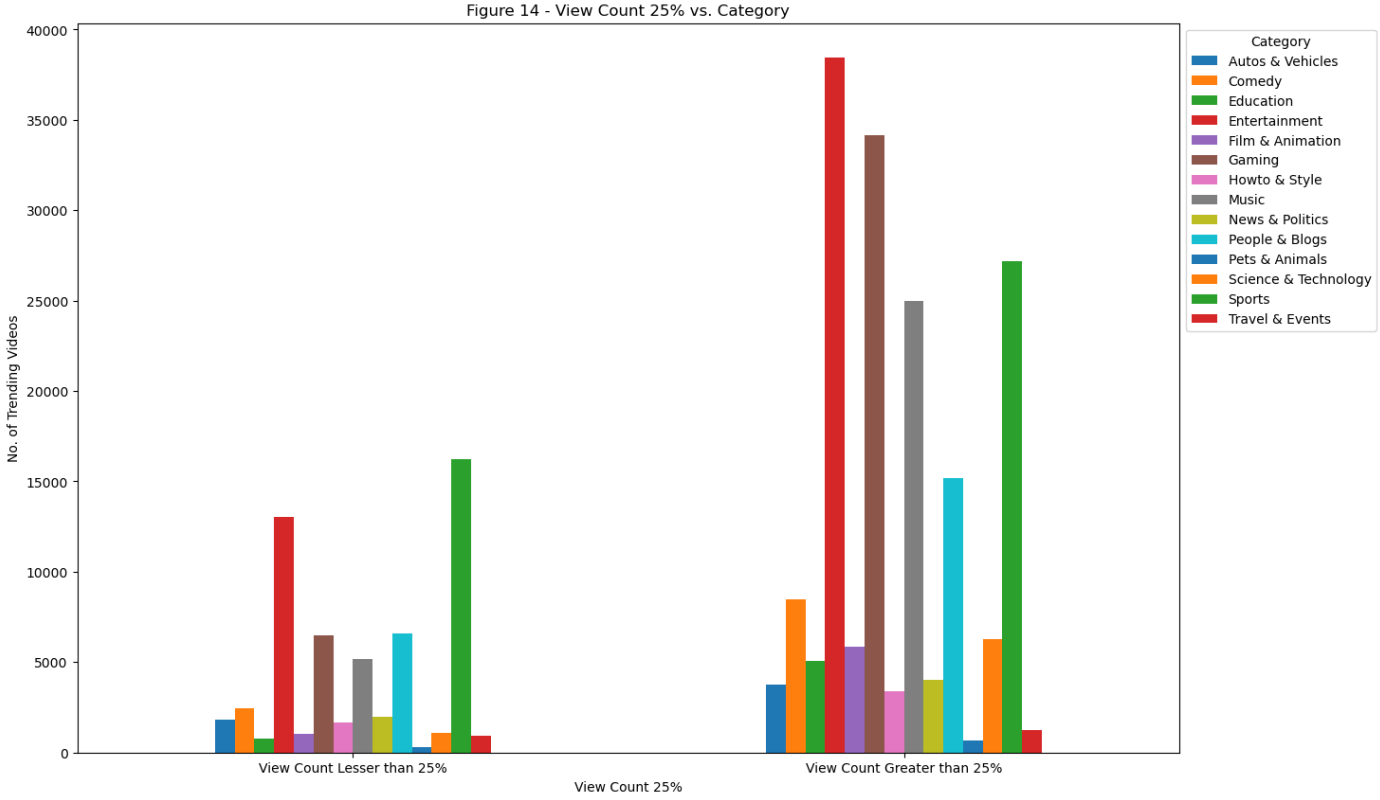


Figure 13 shows the 50 percentile in view count. Some have the same number of videos on both sides, while more trending Sports videos are less than the 50 percentile, and more Gaming, Entertainment, and Music videos are above the 50 percentile.

In [42]:
```
df_ob3.groupby(['view_count_25','category']).size().unstack().plot(kind='bar',stacked=Fa
```

Out[42]:
```
[Text(0, 0, 'View Count Lesser than 25%'),
 Text(1, 0, 'View Count Greater than 25%')]
```

Figure 14 - View Count 25% vs. Category



The above three graphs - Figures 12,13 and 14; gives us insights on the number of views a video has accumulated in the trending dataset. We can conclude that if a video successfully acquires views between the 25th percentile (358566.0 views) and 50th percentile (781394.0 views), then the video has a higher probability to make it into the Youtube Trending Algorithm.

We have not used dislikes for our analysis since Youtube's API has stopped registering dislikes count from 2022. We will plot a barplot to visualize this statistics.

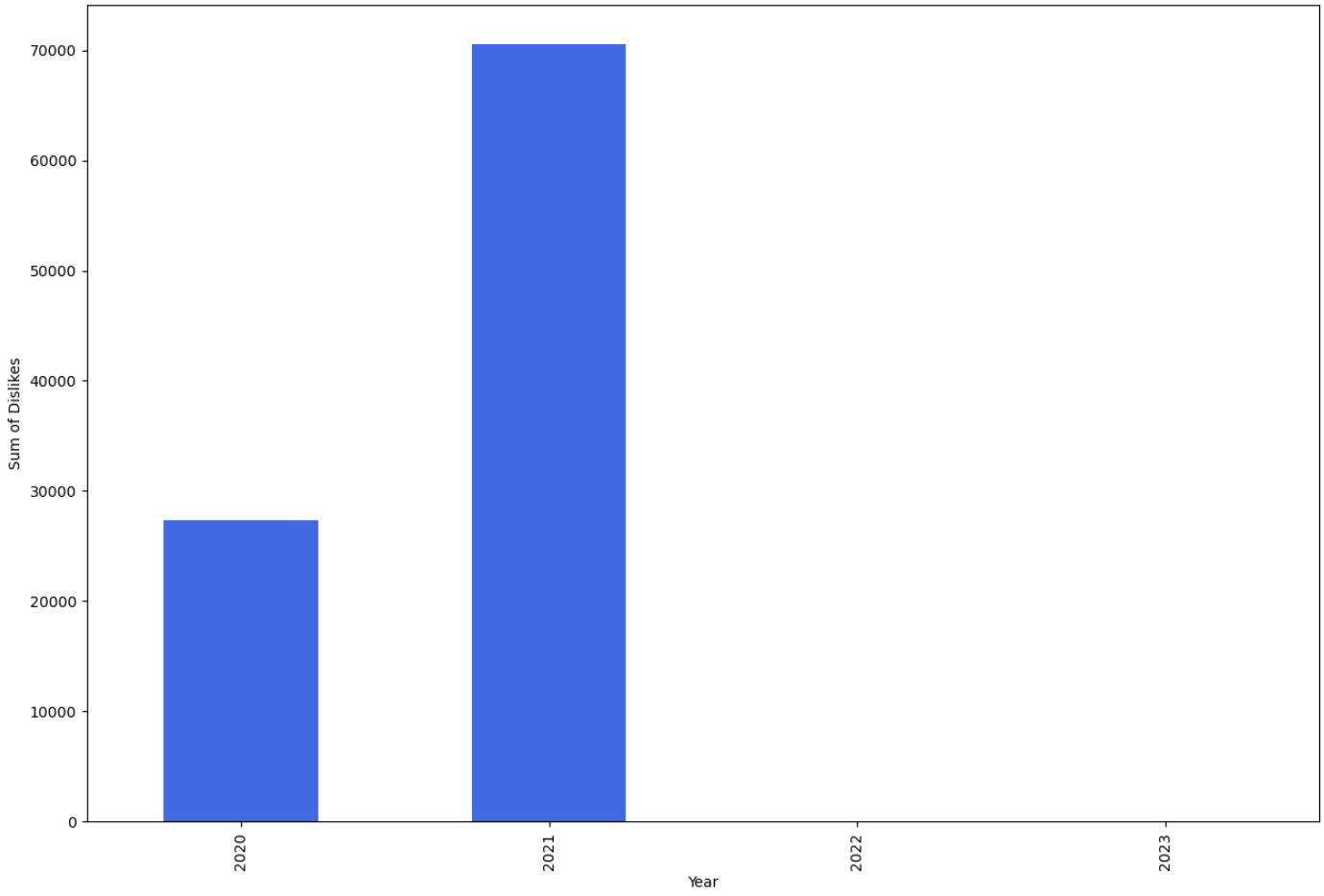
In [43]:
```
# Extract year from 'date' column
df_test = df.copy()
df_test['year'] = df_test['date'].dt.year
df_test['dislikes_nonzero'] = df['dislikes'].apply(lambda x: 0 if x == 0 else 1)

# Group by year and calculate sum of 'dislikes_zero'
yearly_dislikes_nonzero = df_test.groupby('year')['dislikes_nonzero'].sum()

# Plot bar graph
yearly_dislikes_nonzero.plot(kind='bar', figsize=(15,10), title="Figure 15 - Yearly Disl
```

Out[43]:
```
<Axes: title={'center': 'Figure 15 - Yearly Dislikes'}, xlabel='Year', ylabel='Sum of Di
slikes'>
```

Figure 15 - Yearly Dislikes



Thus, from Figure 15, we can see there are dislikes in 2020 and 2021 alone. The empty bins for 2022 and 2023 suggest no data for dislikes count. The reason behind the dislikes count for 2020 being less than 2021 is that we need the complete data for 2020 (From January to December).

In [44]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 238267 entries, 0 to 238390
Data columns (total 23 columns):
 #   Column             Non-Null Count   Dtype
---  ------             --------------   -----
 0   video_id           238267 non-null  object
 1   title              238267 non-null  object
 2   publishedAt        238267 non-null  datetime64[ns, UTC]
 3   channelId          238267 non-null  object
 4   channelTitle       238267 non-null  object
 5   categoryId         238267 non-null  int64
 6   trending_date      238267 non-null  object
 7   tags               238267 non-null  object
 8   view_count         238267 non-null  int64
 9   likes              238267 non-null  int64
 10  dislikes           238267 non-null  int64
 11  comment_count      238267 non-null  int64
 12  thumbnail_link     238267 non-null  object
 13  comments_disabled  238267 non-null  bool
 14  ratings_disabled   238267 non-null  bool
 15  description        233991 non-null  object
 16  category           238165 non-null  object
 17  date               238267 non-null  datetime64[ns, UTC]
 18  month              238267 non-null  category
 19  year               238267 non-null  int32
 20  test_hour          238267 non-null  int32
 21  num_month          238267 non-null  int32
```

```
 22  hour                238267 non-null  int32
dtypes: bool(2), category(1), datetime64[ns, UTC](2), int32(4), int64(5), object(9)
memory usage: 35.2+ MB
```

_ `corr_list` is a variable that includes all the columns that is to be checked with the correlation matrix._

Correlation means that there is a relationship between two things, but does not always mean that one causes the other. It ranges from -1 to 1, where 0 denotes 'No correlation' and 1 denotes positively correlated (If one variable increases, the other variable also increases), while -1 denotes negatively correlated (If one variable increases, the other variable decreases). We aim to see how close it is to -1 and 1 for correlation, the closer the more correlated.

We use the `corr()` method to get the correlation.

In [45]:
```
corr_list = ['hour','date','num_month','categoryId','year','likes','dislikes','comment_c
df[corr_list].corr()
```

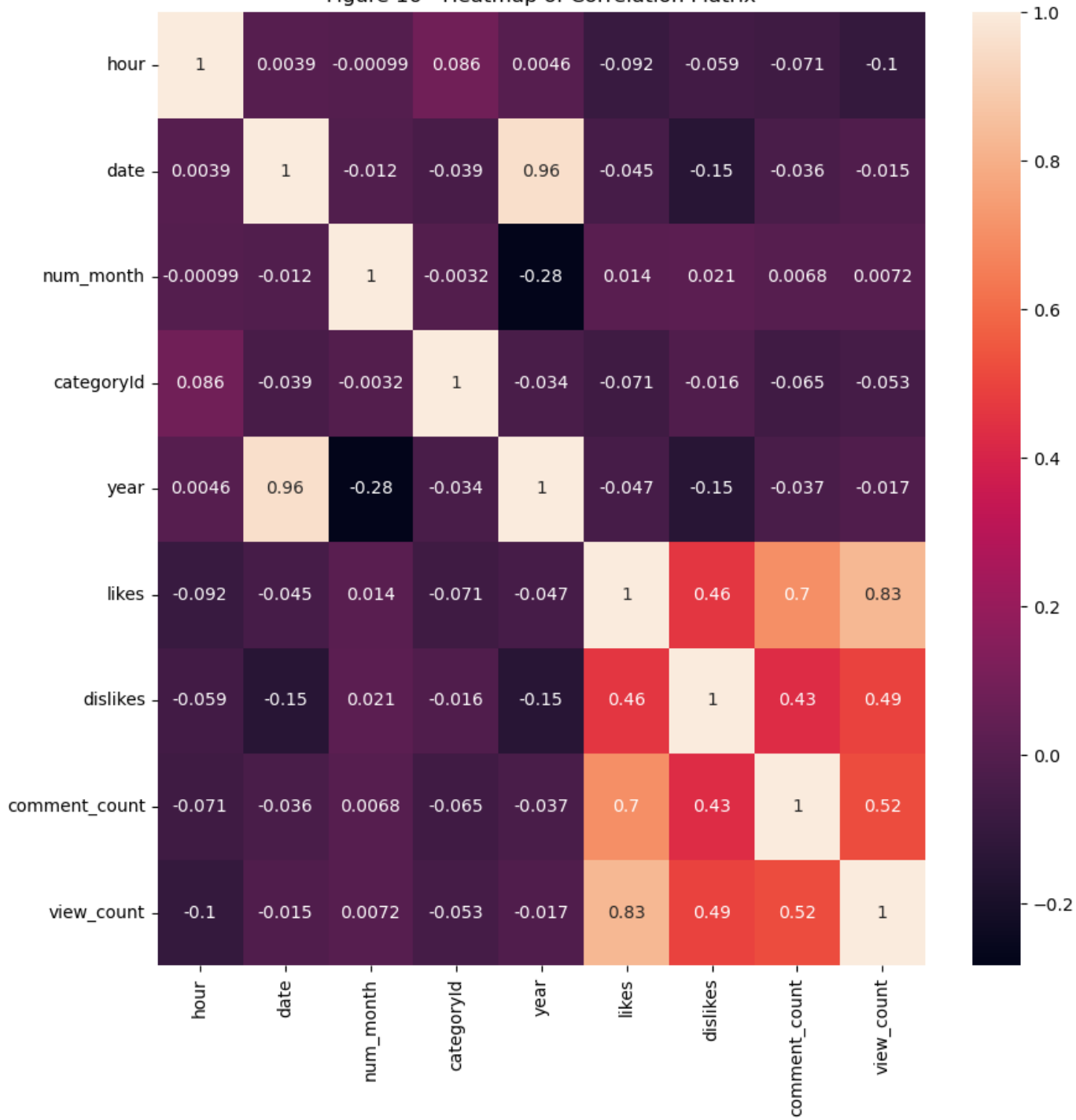
Out[45]:

	hour	date	num_month	categoryId	year	likes	dislikes	comment_count	v
hour	1.000000	0.003934	-0.000987	0.086237	0.004631	-0.092137	-0.059487	-0.070505	
date	0.003934	1.000000	-0.012425	-0.038589	0.957258	-0.044945	-0.146879	-0.036353	
num_month	-0.000987	-0.012425	1.000000	-0.003232	-0.283305	0.014440	0.020860	0.006839	
categoryId	0.086237	-0.038589	-0.003232	1.000000	-0.034452	-0.070634	-0.016357	-0.064997	
year	0.004631	0.957258	-0.283305	-0.034452	1.000000	-0.047108	-0.147124	-0.037089	
likes	-0.092137	-0.044945	0.014440	-0.070634	-0.047108	1.000000	0.461008	0.699849	
dislikes	-0.059487	-0.146879	0.020860	-0.016357	-0.147124	0.461008	1.000000	0.429602	
comment_count	-0.070505	-0.036353	0.006839	-0.064997	-0.037089	0.699849	0.429602	1.000000	
view_count	-0.104223	-0.015436	0.007151	-0.052694	-0.016781	0.829194	0.494547	0.519964	

Now that we have the correlation, we will use seaborn's feature called `heatmap()` to better visualize the correlation among them.

In [46]:
```
plt.figure(figsize=(10,10))
fig = sns.heatmap(data = df[corr_list].corr(), annot=True).set_title("Figure 16 - Heatma
```

Figure 16 - Heatmap of Correlation Matrix



From Figure 16, we see that the likes and view count is high-positively correlated with 0.83 value. The dislikes and view count, the comment count and view count, and the dislikes and comment count are also correlated to a certain extent. The rest of the variables are neglibly correlated as the value is closer to 0.

We can see a very high positive correlation between year and date, and further explore on it by comparing it with likes, comment count and view count. We will use the `lineplot` method to plot the graph.

In [47]:
```
f, ax = plt.subplots(2, 3, figsize=(25, 20))
variables = ['likes', 'view_count', 'comment_count']
x_values = ['hour','month']
plt.suptitle('Figure 17 - Lineplot of Likes, View Count and Comment Count vs Hour and Mo
plt.subplots_adjust(top=0.85)

for i, x in enumerate(x_values):
    for j, var in enumerate(variables):
        sns.lineplot(x = df[x], y = df[var], data = df, ax = ax[i,j], marker = 'o').set_
```

Figure 17 - Lineplot of Likes, View Count and Comment Count vs Hour and Month

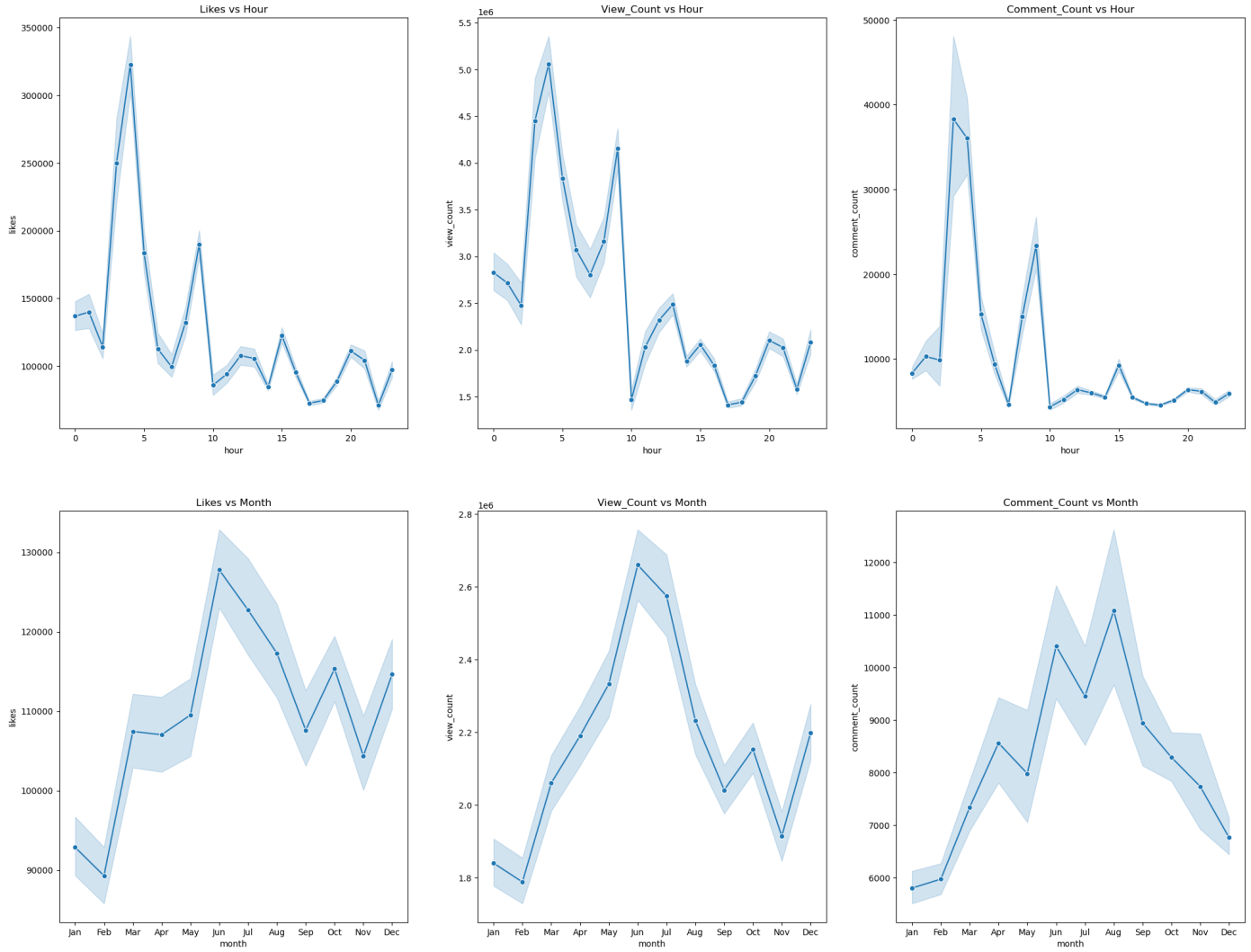


Figure 17 shows us the line plot for the number of likes, view count and comment count hourwise with respect to hours and month. From the figure 17, we see the majority of likes, view counts and comments are accrued for the videos pubished between 2 am and 7 am. We can also see the same trend for the videos published in the months between May and September.

Although we have a time frame to see a spike of the video, we will now use the `boxplot()` to better visualize the spike hourly and monthly.

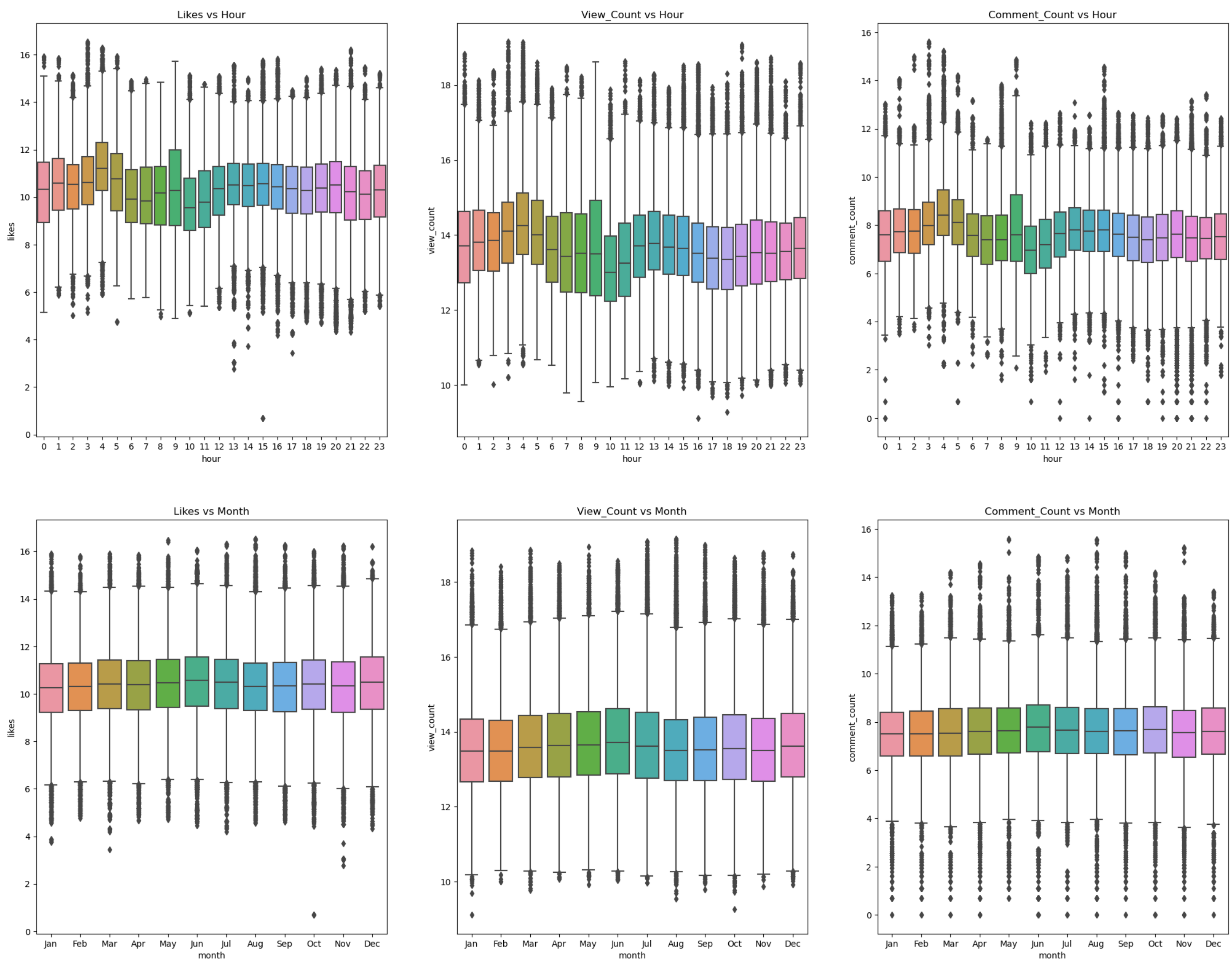
In [48]:
```
f, ax = plt.subplots(2, 3, figsize=(25, 20))
variables = ['likes', 'view_count', 'comment_count']
x_values = ['hour','month']
plt.suptitle('Figure 17 - Boxplot of Likes, View Count and Comment Count', fontsize=18,p
plt.subplots_adjust(top=0.85)

for i, x in enumerate(x_values):
    for j, var in enumerate(variables):
        sns.boxplot(x = df[x], y = np.log(df[var]), data = df, ax = ax[i,j]).set_title(f
```

```
c:\Users\asish\anaconda3\Lib\site-packages\pandas\core\arraylike.py:396: RuntimeWarning:
divide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
c:\Users\asish\anaconda3\Lib\site-packages\pandas\core\arraylike.py:396: RuntimeWarning:
```

```
divide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
c:\Users\asish\anaconda3\Lib\site-packages\pandas\core\arraylike.py:396: RuntimeWarning:
divide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
c:\Users\asish\anaconda3\Lib\site-packages\pandas\core\arraylike.py:396: RuntimeWarning:
divide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
c:\Users\asish\anaconda3\Lib\site-packages\pandas\core\arraylike.py:396: RuntimeWarning:
divide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
c:\Users\asish\anaconda3\Lib\site-packages\pandas\core\arraylike.py:396: RuntimeWarning:
divide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

Figure 17 - Boxplot of Likes, View Count and Comment Count



From Figure 18, we can now see the exact hour and month better. Thus, as per the graph, 5am is the best time and June is the best month to publish the video.

Objective 4 : *Implementing Regression Models to predict View Counts given Attributes*

Importing Regression models and creating Test/Train Datasets.

To begin with our analysis, we import the sci-kit-learn Python library. Scikit-learn contains a variety of practical classes that can help us with regression modelling. We import train_test_split, LinearRegression and RandomForestRegressor for modelling. We also import mean_squared_error and r2_score to help us in evaluating the model.

In [49]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import MinMaxScaler
```

Perform `one hot encoding` to create dummy variables for each categorical column, in order to produce a correlation matrix.

Use a `for loop` to create a `heatmap` for each of the produced correlation matrix

In [50]:
```
df_ohe = df.copy()
#list of columns to be one hot encoded
ohe_columns = ['categoryId','num_month','hour','year']
#one hot encode the columns
df_ohe = pd.get_dummies(df_ohe, columns=ohe_columns)

# drop columns that are not needed - title, publishedAt, trending_date, date, month, yea
df_ohe = df_ohe.drop(['title','publishedAt','trending_date','date','month','description'

# scale the numerical columns

scaler = MinMaxScaler()
df_ohe[['likes', 'dislikes', 'comment_count']] = scaler.fit_transform(df_ohe[['likes', '
df_ohe.head()
```

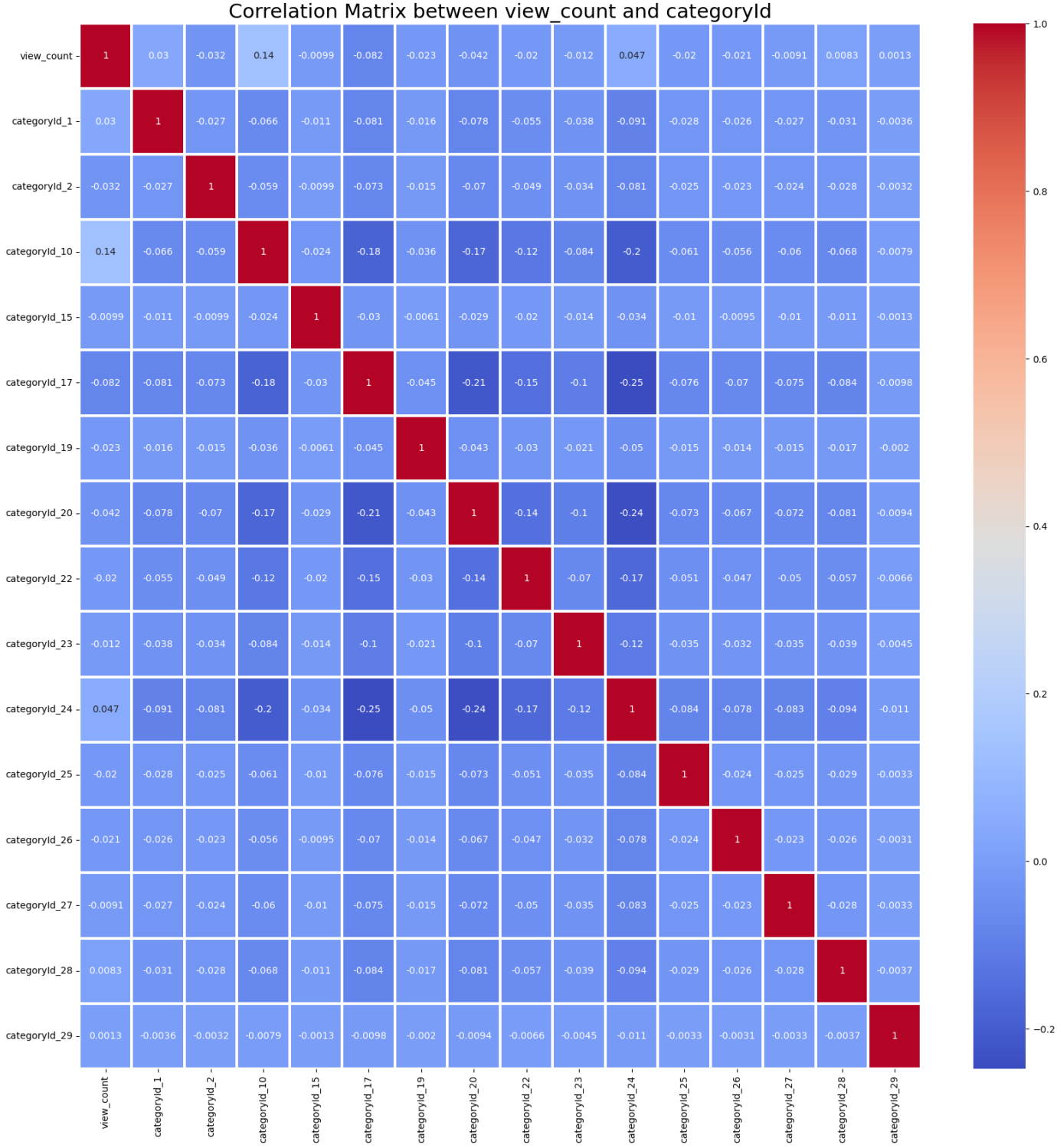
Out[50]:

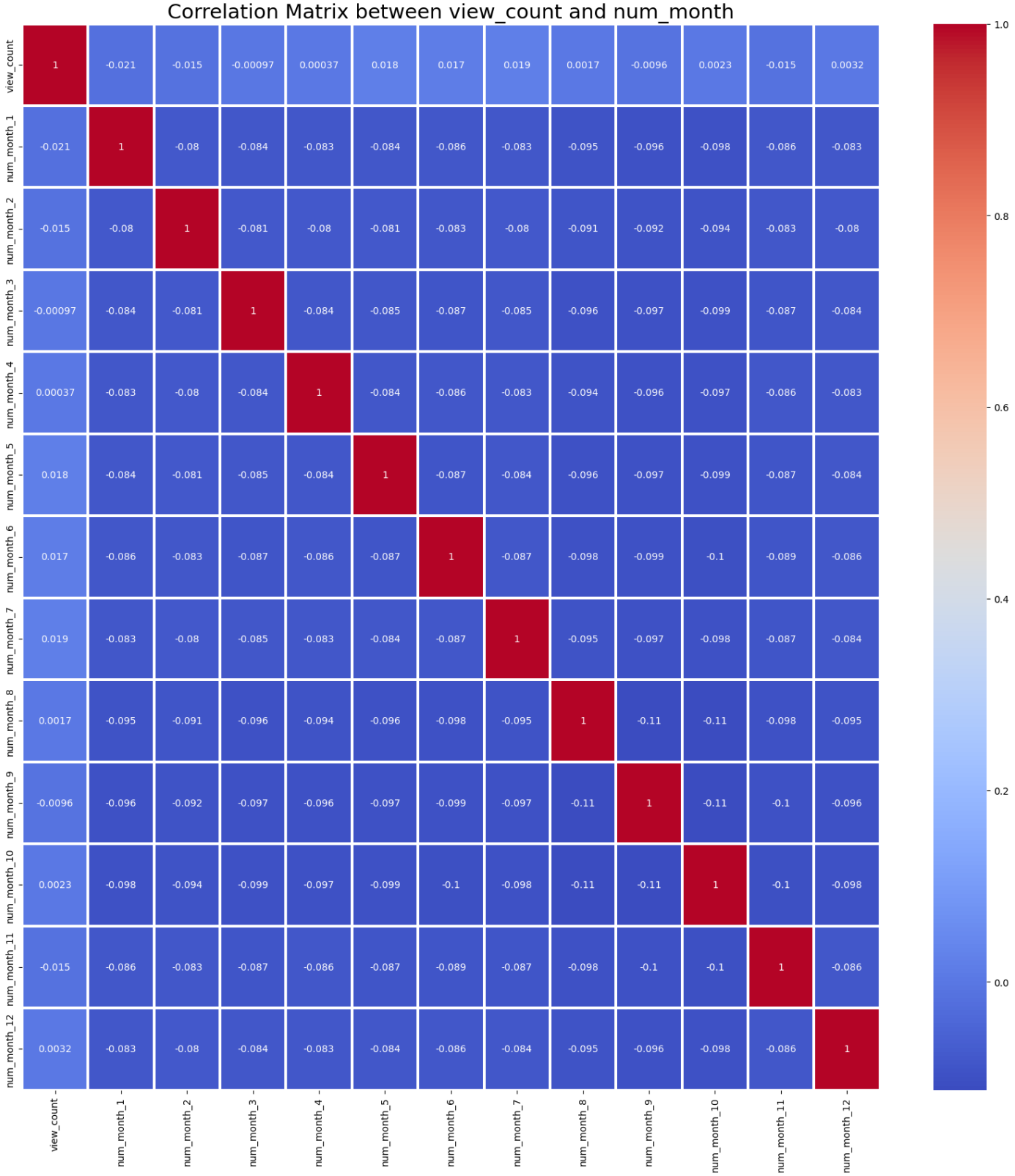
	view_count	likes	dislikes	comment_count	test_hour	categoryId_1	categoryId_2	categoryId_10	categor
0	2038853	0.023205	0.003038	0.006718	0	False	False	False	
1	236830	0.001077	0.000242	0.000274	0	False	False	False	
2	2381688	0.009624	0.003230	0.002764	0	False	False	False	
3	613785	0.002464	0.000773	0.000351	0	False	False	False	
4	940036	0.005714	0.002150	0.001178	0	False	False	False	

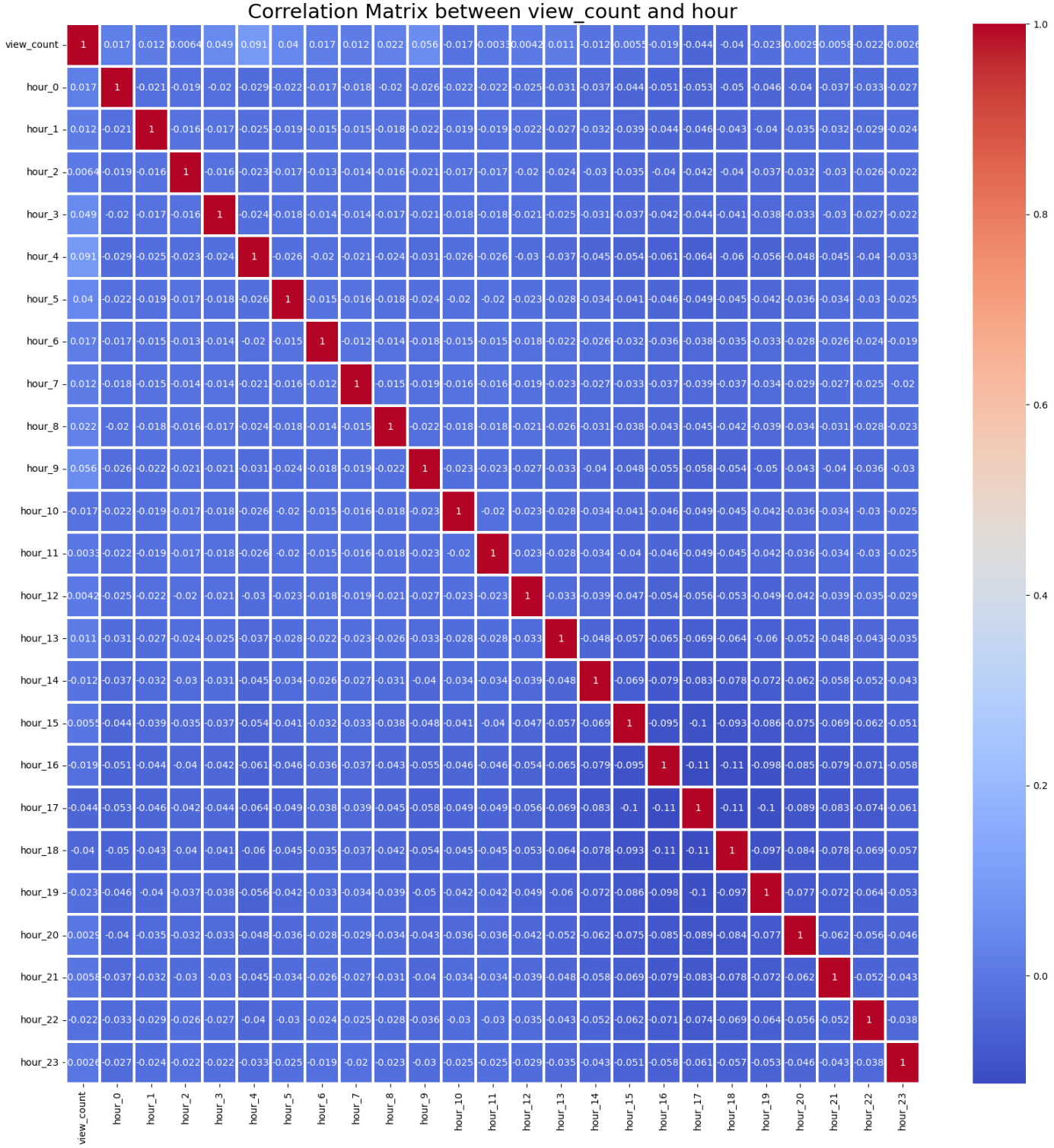
5 rows × 60 columns

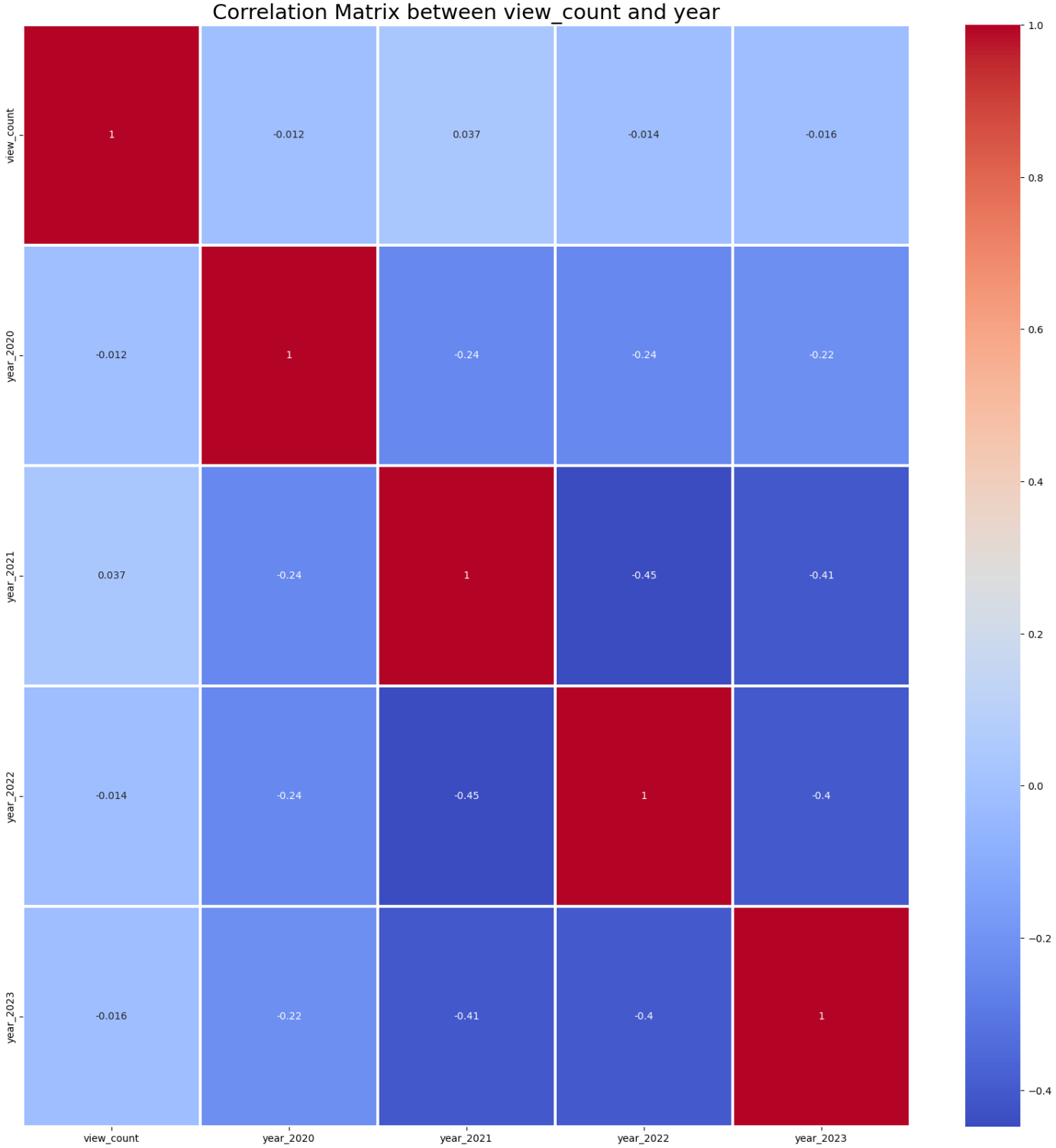
In [51]:
```
for col in ['categoryId','num_month','hour','year']:
    df_corr = df[['view_count',col]]
    df_ohe1 = pd.get_dummies(df_corr, columns= [col])
    corr = df_ohe1.corr()


    f, ax = plt.subplots(figsize=(20, 20))
    sns.heatmap(corr, annot=True, cmap='coolwarm', linewidths=1.5, ax=ax)
    plt.title(f"Correlation Matrix between view_count and {col}", fontsize=21)
```


Correlation Matrix between view_count and categoryId
view_count 1 0.03 -0.032 0.14 -0.0099 -0.082 -0.023 -0.042 -0.02 -0.012 0.047 -0.02 -0.021 -0.0091 0.0083 0.0013
categoryId_1 0.03 1 -0.027 -0.066 -0.011 -0.081 -0.016 -0.078 -0.055 -0.038 -0.091 -0.028 -0.026 -0.027 -0.031 -0.0036
categoryId_2 -0.032 -0.027 1 -0.059 -0.0099 -0.073 -0.015 -0.07 -0.049 -0.034 -0.081 -0.025 -0.023 -0.024 -0.028 -0.0032
categoryId_10 0.14 -0.066 -0.059 1 -0.024 -0.18 -0.036 -0.17 -0.12 -0.084 -0.2 -0.061 -0.056 -0.06 -0.068 -0.0079
categoryId_15 -0.0099 -0.011 -0.0099 -0.024 1 -0.03 -0.0061 -0.029 -0.02 -0.014 -0.034 -0.01 -0.0095 -0.01 -0.011 -0.0013
categoryId_17 -0.082 -0.081 -0.073 -0.18 -0.03 1 -0.045 -0.21 -0.15 -0.1 -0.25 -0.076 -0.07 -0.075 -0.084 -0.0098
categoryId_19 -0.023 -0.016 -0.015 -0.036 -0.0061 -0.045 1 -0.043 -0.03 -0.021 -0.05 -0.015 -0.014 -0.015 -0.017 -0.002
categoryId_20 -0.042 -0.078 -0.07 -0.17 -0.029 -0.21 -0.043 1 -0.14 -0.1 -0.24 -0.073 -0.067 -0.072 -0.081 -0.0094
categoryId_22 -0.02 -0.055 -0.049 -0.12 -0.02 -0.15 -0.03 -0.14 1 -0.07 -0.17 -0.051 -0.047 -0.05 -0.057 -0.0066
categoryId_23 -0.012 -0.038 -0.034 -0.084 -0.014 -0.1 -0.021 -0.1 -0.07 1 -0.12 -0.035 -0.032 -0.035 -0.039 -0.0045
categoryId_24 0.047 -0.091 -0.081 -0.2 -0.034 -0.25 -0.05 -0.24 -0.17 -0.12 1 -0.084 -0.078 -0.083 -0.094 -0.011
categoryId_25 -0.02 -0.028 -0.025 -0.061 -0.01 -0.076 -0.015 -0.073 -0.051 -0.035 -0.084 1 -0.024 -0.025 -0.029 -0.0033
categoryId_26 -0.021 -0.026 -0.023 -0.056 -0.0095 -0.07 -0.014 -0.067 -0.047 -0.032 -0.078 -0.024 1 -0.023 -0.026 -0.0031
categoryId_27 -0.0091 -0.027 -0.024 -0.06 -0.01 -0.075 -0.015 -0.072 -0.05 -0.035 -0.083 -0.025 -0.023 1 -0.028 -0.0033
categoryId_28 0.0083 -0.031 -0.028 -0.068 -0.011 -0.084 -0.017 -0.081 -0.057 -0.039 -0.094 -0.029 -0.026 -0.028 1 -0.0037
categoryId_29 0.0013 -0.0036 -0.0032 -0.0079 -0.0013 -0.0098 -0.002 -0.0094 -0.0066 -0.0045 -0.011 -0.0033 -0.0031 -0.0033 -0.0037 1
view_count categoryId_1 categoryId_2 categoryId_10 categoryId_15 categoryId_17 categoryId_19 categoryId_20 categoryId_22 categoryId_23 categoryId_24 categoryId_25 categoryId_26 categoryId_27 categoryId_28 categoryId_29
1.0 0.8 0.6 0.4 0.2 0.0 −0.2


Correlation Matrix between view_count and num_month
view_count
num_month_1
num_month_2
num_month_3
num_month_4
num_month_5
num_month_6
num_month_7
num_month_8
num_month_9
num_month_10
num_month_11
num_month_12
1 -0.021 -0.015 -0.00097 0.00037 0.018 0.017 0.019 0.0017 -0.0096 0.0023 -0.015 0.0032
-0.021 1 -0.08 -0.084 -0.083 -0.084 -0.086 -0.083 -0.095 -0.096 -0.098 -0.086 -0.083
-0.015 -0.08 1 -0.081 -0.08 -0.081 -0.083 -0.08 -0.091 -0.092 -0.094 -0.083 -0.08
-0.00097 -0.084 -0.081 1 -0.084 -0.085 -0.087 -0.085 -0.096 -0.097 -0.099 -0.087 -0.084
0.00037 -0.083 -0.08 -0.084 1 -0.084 -0.086 -0.083 -0.094 -0.096 -0.097 -0.086 -0.083
0.018 -0.084 -0.081 -0.085 -0.084 1 -0.087 -0.084 -0.096 -0.097 -0.099 -0.087 -0.084
0.017 -0.086 -0.083 -0.087 -0.086 -0.087 1 -0.087 -0.098 -0.099 -0.1 -0.089 -0.086
0.019 -0.083 -0.08 -0.085 -0.083 -0.084 -0.087 1 -0.095 -0.097 -0.098 -0.087 -0.084
0.0017 -0.095 -0.091 -0.096 -0.094 -0.096 -0.098 -0.095 1 -0.11 -0.11 -0.098 -0.095
-0.0096 -0.096 -0.092 -0.097 -0.096 -0.097 -0.099 -0.097 -0.11 1 -0.11 -0.1 -0.096
0.0023 -0.098 -0.094 -0.099 -0.097 -0.099 -0.1 -0.098 -0.11 -0.11 1 -0.1 -0.098
-0.015 -0.086 -0.083 -0.087 -0.086 -0.087 -0.089 -0.087 -0.098 -0.1 -0.1 1 -0.086
0.0032 -0.083 -0.08 -0.084 -0.083 -0.084 -0.086 -0.084 -0.095 -0.096 -0.098 -0.086 1
1.0
0.8
0.6
0.4
0.2
0.0


Correlation Matrix between view_count and hour
view_count
hour_0
hour_1
hour_2
hour_3
hour_4
hour_5
hour_6
hour_7
hour_8
hour_9
hour_10
hour_11
hour_12
hour_13
hour_14
hour_15
hour_16
hour_17
hour_18
hour_19
hour_20
hour_21
hour_22
hour_23
1.0
0.8
0.6
0.4
0.2
0.0



Encoding Categorical Variables and Scaling

Since our dataset contains categorical data like `categoryId`, `Month`, `Hour` and `year`, we use OneHotEncoding function to create dummy variables.

Similarly, our dataset contains numerical values in `likes`, `dislikes` and `comment_count` which have varying numerical scale. We can thus use `MinMaxScaler` from Scikit-Learn to scale our numerical columns. Scikit-Learn alos contains `StandardScaler`, but we are using `MinMaxScaler` since it scales the data from `[0,1]` and preserves the shape of the dataset.

Finally, we drop all unnecessary columns in our dataframe to reduce computation complexity and then create a 20:80 test/train datasets.

In [52]:
```
df_ohe = df.copy()
#list of columns to be one hot encoded
ohe_columns = ['categoryId','num_month','hour','year']
#one hot encode the columns
df_ohe = pd.get_dummies(df_ohe, columns=ohe_columns)

# drop columns that are not needed - title, publishedAt, trending_date, date, month, yea
df_ohe = df_ohe.drop(['title','publishedAt','trending_date','date','month','description'

# scale the numerical columns

scaler = MinMaxScaler()
df_ohe[['likes', 'dislikes', 'comment_count']] = scaler.fit_transform(df_ohe[['likes', '
df_ohe.head()

X = df_ohe.drop(['view_count'], axis=1)
Y = df_ohe['view_count']

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=42
```

Creating and evaluating a LinearRegression model

Secondly, we instantiate a linear regression model from the Scikit-Learn library and assign it to a variable called model. The object's fit method is used to train and create a linear regression model using our training dataset X_train. Finally, using the created model, we use our testing dataset X_test to predict our view count.

In [53]:
```
#fit the model
model = LinearRegression()
model.fit(X_train, y_train)

#predict the model
y_pred = model.predict(X_test)
```

Evaluating the model

We can then print the Mean Squared Error, R**2 and Root Mean Squared Error.

In [54]:
```
#evaluate the model
print("Mean squared error: %.2f" % mean_squared_error(y_test, y_pred))
print('Variance score: %.2f' % r2_score(y_test, y_pred))
print('Root Mean squared error: %.2f' % mean_squared_error(y_test, y_pred, squared=False
```

```
Mean squared error: 9738287750508.69
Variance score: 0.72
Root Mean squared error: 3120622.97
```

From the output above, we get a variance score of 0.72. While this score is close to 1, we can try another model to ensure we get the best possible score.

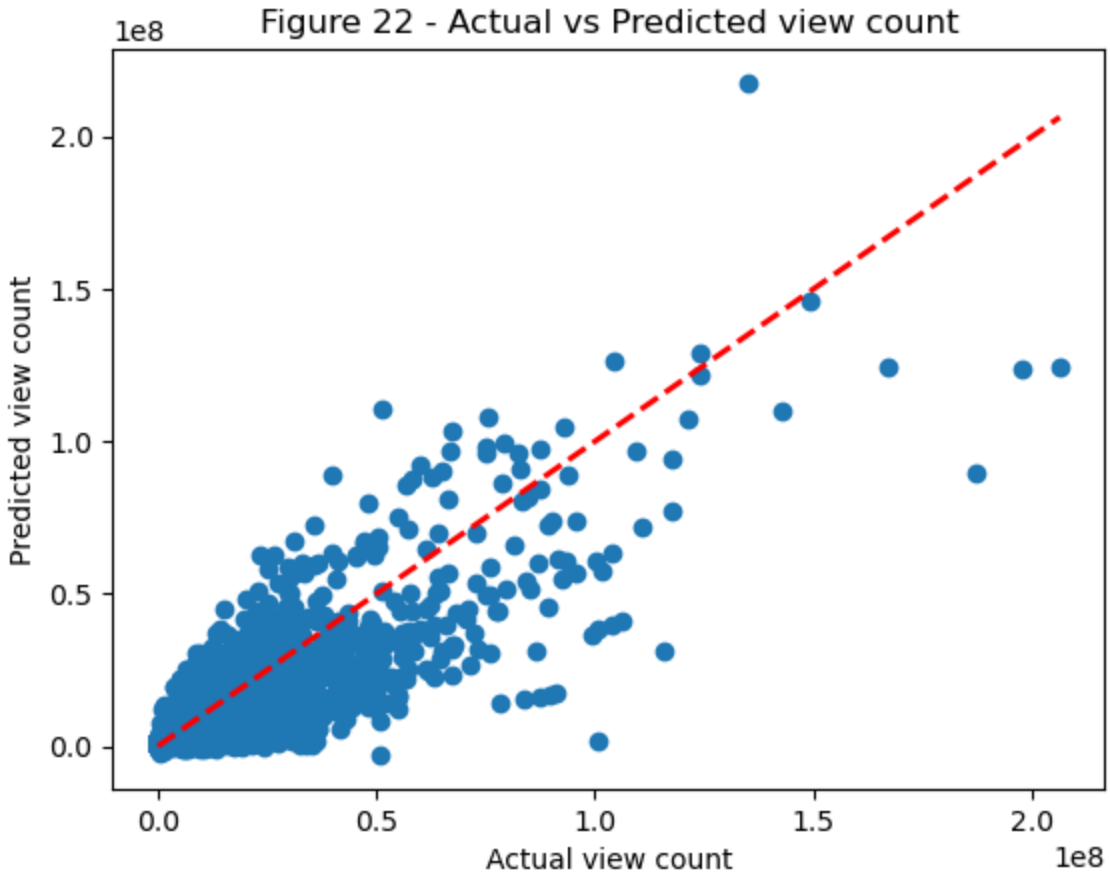
Plotting Predicted vs Actual View Count For Linear Regression

When we plot the graph for Linear Regression, we can observe that while Linear Regression can predict view counts for trending videos up to 1e8 views, it fails to accurately predict higher view counts, thus lowering our variance score.

In [55]:
```
#plot the model
plt.scatter(y_test, y_pred)
plt.xlabel("Actual view count")
```

```
plt.ylabel("Predicted view count")
plt.title("Figure 23 - Actual vs Predicted view count")
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=2,c = "r"
plt.show()
```

```
C:\Users\asish\AppData\Local\Temp\ipykernel_15344\2191979230.py:6: UserWarning: color is
redundantly defined by the 'color' keyword argument and the fmt string "k--" (-> color
='k'). The keyword argument will take precedence.
  plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=2,c =
"r")
```



Using Random Forest Regressor

We can now use Linear Regression as our baseline model to evaluate our next model. We are using RandomForestRegressor from Scikit-Learn to implement our model.

RandomForestRegressor needs an argument called n_estimtor that determines the number of decision tree classifiers to use. To check the appropriate integer for the n_estimator, we can create a function that takes an n as the number estimator. The function returns MSE, R2 and RMSE values.

In [56]:
```
def RandomForestList(n):
    model = RandomForestRegressor(n_estimators=n, random_state=201750985)
    model.fit(X_train, y_train.values.ravel())
    y_pred = model.predict(X_test)
    return mean_squared_error(y_test, y_pred), r2_score(y_test, y_pred), mean_squared_er
```

We can then pass the function through a for loop that iterates over the list of estimators as defined below. We can then append the resultant scores against the n_estimators to a new data frame.

_Note: The below code to find the best n_estimator has been commented out after execution in order to ensure the Jupyter Notebook completes execution in a reasonable amount of time. The only functionality of this code is to find best n*estimator and is not supposed to be executed everytime.*

In [57]:
```
rf_df = pd.DataFrame(columns=['n_estimators', 'MSE', 'RMSE', 'R2'])
n_estimates = [1, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50,55 ,60 ,65 ,70, 75 ,80 ,85 ,90 ,
for i in n_estimates:
    mse,r2,rmse = RandomForestList(i)
    rf_df = pd.concat([rf_df, pd.DataFrame([{'n_estimators': i, 'MSE': mse, 'RMSE': rmse
    #rf_df = rf_df.append(, ignore_index=True)
```

In [58]:
```
rf_df.head(100)
```

Selection of Estimator value

We can now plot RMSE vs n_estimator and R2 vs n_estimator. From the graph, we see that an increase in n_estimnator initially causes a rapid rise in R2 value, but only a marginal increase from n_estimator greater than 15. Furthermore, any increase in the value of n_estimnator does not cause any change in the value of R2. Thus, we decide that the value of n_estimator should be 50 to maximise our R2 while also minimising compute time.

_Note: The below code to plot the best n_estimator has been commented out after execution in order to ensure the Jupyter Notebook completes execution in a reasonable amount of time. The only functionality of this code is to plot the best n*estimator and is not supposed to be executed everytime.*

In [59]:
```
fig, axes = plt.subplots(1, 2, figsize=(15, 5))
axes[0].plot(rf_df['n_estimators'], rf_df['RMSE'])
axes[0].set_xlabel('n_estimators')
axes[0].set_ylabel('RMSE')
axes[0].set_title('Figure 24 - RMSE vs n_estimators')
axes[0].axvline(x=50, color='r', linestyle='--')
axes[1].plot(rf_df['n_estimators'], rf_df['R2'], color='orange')
axes[1].set_xlabel('n_estimators')
axes[1].set_ylabel('R2')
axes[1].set_title('Figure 25 - R2 vs n_estimators')
axes[1].axvline(x=50, color='r', linestyle='--')
plt.show()
```

Training RandomForest Modedel using Selected n_estimator

After determining 50 as the ideal value of n_estimator, we can start training our Random Forest Model. AFter training with X_train and y_train, we can test our model with the X_test dataset.

In [60]:
```
# create regressor object
regressor = RandomForestRegressor(n_estimators = 50, random_state = 201750985)
regressor.fit(X_train, y_train.values.ravel())
# predict the result
y_pred = regressor.predict(X_test)
```

Evaluating the Model

Printing the scores for our model shows a significant increase in the R2 value from 0.72 to 0.95. Thus, we can choose the RandomForestRegressor with n_estimator as 50 for our prediction Model.

In [61]:
```
print("Mean squared error: %.2f" % mean_squared_error(y_test, y_pred))
print('Variance score: %.2f' % r2_score(y_test, y_pred))
print('Root Mean squared error: %.2f' % mean_squared_error(y_test, y_pred, squared=False
```

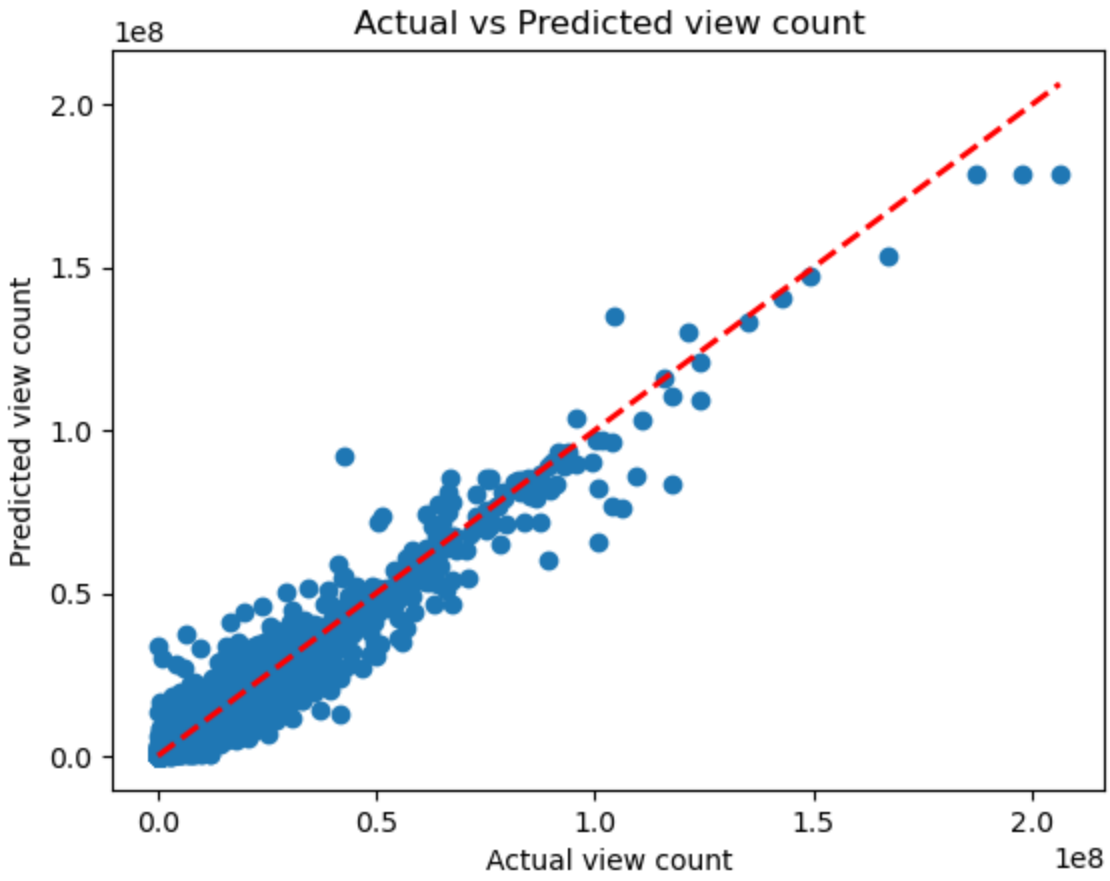
```
Mean squared error: 1859860906199.00
Variance score: 0.95
```

```
Root Mean squared error: 1363767.17
```

Plotting the graph of actual vs prediction shows our model has improved over just Linear Regression in determining View_count of Videos given a set of attributes about the video.

In [62]:
```
#plot the model
plt.scatter(y_test, y_pred)
plt.xlabel("Actual view count")
plt.ylabel("Predicted view count")
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=2,c = "r"
plt.title("Figure 26: Actual vs Predicted view count")
plt.show()
```

```
C:\Users\asish\AppData\Local\Temp\ipykernel_15344\2508738616.py:5: UserWarning: color is redundantly defined by the 'color' keyword argument and the fmt string "k--" (-> color ='k'). The keyword argument will take precedence.
  plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=2,c = "r")
```



Project Outcome (10 + 10 marks)

This section should describe the outcome of the project by means of both explanation of the results and by graphical visualisation in the form of graphs, charts or or other kinds of diagram

The section should begin with a general overview of the results and then have a section for each of the project objectives. For each of these objectives an explanation of more specific results relating to that objective shoud be given, followed by a section presenting some visualisation of the results obtained. (In the case where the project had just one objective, you should still have a section describing the results from a general perspective followed by a section that focuses on the particular objective.)

The marks for this section will be divided into 10 marks for Explanation and 10 marks for Visualisation. These marks will be awarded for the Project Outcome section as a whole, not for each objective individually. Hence, you do not have to pay equal attention to each. However, you are expected to have a some explanation and visualisation for each. It is suggested you have 200-400 words explanation for each objective.

Overview of Results

Give a general overview of the results (around 200 words).

Objective 1

We can deduce two conclusions:

1. In 2020, the number of bins in the plot is comparatively smaller than that of other bins. This is mainly because we only have the data for the last five months of 2020 (i.e., August to December 2020).
2. We also notice a change in trend across 2021 to 2023, where the videos in the gaming category are more trending, while the Entertainment, and People and blog categories are decreasing year by year. Sports and Music categories stay in almost similar trends.

Objective 2

Checks the key words to be used in the Title and/or Tags for a video to be in the trending category

Now, we will probe into investigating the keywords used in the title, description and how frequent the keywords are among the list of trending categories in each category. We first created a new variable 'title_length' to store the number of words in each title. As we can see in Figure 4, most of the trending videos have 8 words in the title, and there are almost negligible trending videos that have over 20 words.

We then studied into finding actual keywords that are significant in the title and Video tags. We used the word cloud to plot the keywords as Figure 5 for the title and Figure 6 for the video tags respectively, for each category. In Figure 5, we can see the ones that are dark and big are more used keywords and of the highest significance. Thus, in the Entertainment category, keywords like 'Official', and 'Trailer' are used more, while 'Slow Mo' is of lesser significance. Similarly in Figure 6, 'Manchester United' has the highest significance in Entertainment category.

The same goes for the rest of the categories, where 'Official Video' tops the list across a few categories, followed by the local keywords such as 'Prime Minister' for News & Politics, 'League Highlights' for Sports, 'iPhone Pro' for Science & Technology, etc. in regards to the keywords used in the title.

Now that we observed the keywords used, we then generated a word cloud that has the common keywords between the title and the video tags. From Figure 7, we spot that "Season" is the most commonly used keyword in the title as well as in the video tags in the Entertainment category.

Based on the observations made, we can deduce the following key points,

1. The audience is more intrigued to watch videos that have keywords relating to current affairs, in almost every category.

2. The keywords used in the title are also used in the video tags, with which one could assume that the frequency has an impact on the reach of the videos.

Objective 3

Objective 4

We aimed to predict the number of views based on dependent variables, such as likes, comment_count, categoryId and published date. We have implemented Linear Regression as a baseline model and a Random Forest Algorithm to create a prediction model.

We have used MSE (Mean Squared Error), RMSE (Root Mean Squared Error) and R2 (Variance) to evaluate the prediction accuracy of the model, and the following results were observed.

Linear Regression

- The MSE score is 9738287750508.69
- The RMSE score is 3120622.97
- The R2 value is 0.72 From the above scores, we can assert that our model does provide a satisfactory result but can be further improved by utilising Random Forest Regression. Figure 23 should show all the points on the dotted diagonal line, but our Linear Regression model shows a considerable diversion as our actual view count increases.

Random Forest Regression

Random Forest Regression uses a n_estimator, which is the number of decision tree classifiers to use. To determine the appropriate n_estimator, we have iterated over a range from 1 to 100. The results of the iteration have been plotted in `figure 24` and `figure 25`. The results show that while there is a considerable rise in variance for lesser values of n, the variance flatlines after n = 50. Thus, we choose 50 as our n_estimator.

Using 50 as our n_estimator for our model gives us the following results

- The MSE score is 1859860906199.00
- The RMSE score is 1363767.17
- The R2 value is 0.95

From the above scores, it is clear that Random Forest Regression is better at estimating the number of views in the dataset.

Conclusion (5 marks)

Your concluding section should be around 200-400 words. It is recommended that you divide it into the following sections.

Achievements

As we had expected, the most popular fridge magnets were of the 'meme' kind. We were surprised that 'smiley' fridge magnets were less common than expected. We conjecture that this is because, although they are apparently very popular, few fridges display more than one smiley. However, 'meme' based magnets can be found in large numbers, even on quite small fridges.

Limitations

The limitations include the following,

1. Insufficient Data: As we only have data from August 2020 to November 2023, we can only see the trend for the recent years. If we had enough data for at least five years, we could have better predicted trends and accuracy.

2. 0 Dislikes from 2022: YouTube has stopped publishing the dislikes count since 2022. If we had dislike counts for our entire dataset, we would have investigated to see if this factor has any impact on the trending of the videos, along with the likes, view counts, and comment counts.

3. Time Complexity: Modelling RandomForest for large-scale datasets like YouTube's API takes a lot of time to compute. RandomForest is an ensemble model of Decision Trees. The time complexity for building a complete unpruned decision tree is O(v *n* log(n)), where n is the number of records and v is the number of variables/attributes. In order to solve this problem, we can use VPC/VPS for better computational capability to make more complex models further.

Future Work

For future works, we can enhance the project by further probing into the following,

1. Add More Countries: The trends could be investigated for other countries such as the United States, India, Canada, France, etc., as well as the use of local languages in descriptions, titles, etc.
2. Make it more user-configurable: The code could be made more dynamic where the user can compare two or more countries of his choice and also select which variables to use.
3. Usage of different predition algorithms: Algorithms such as Classification, KNN, Neural Networks etc.

In [ ]:
```
import pickle

# save
with open('model.pkl','wb') as f:
    pickle.dump(model,f)
```

In [ ]:
```
import pickle


# load
with open('model.pkl', 'rb') as f:
    model = pickle.load(f)
```