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.

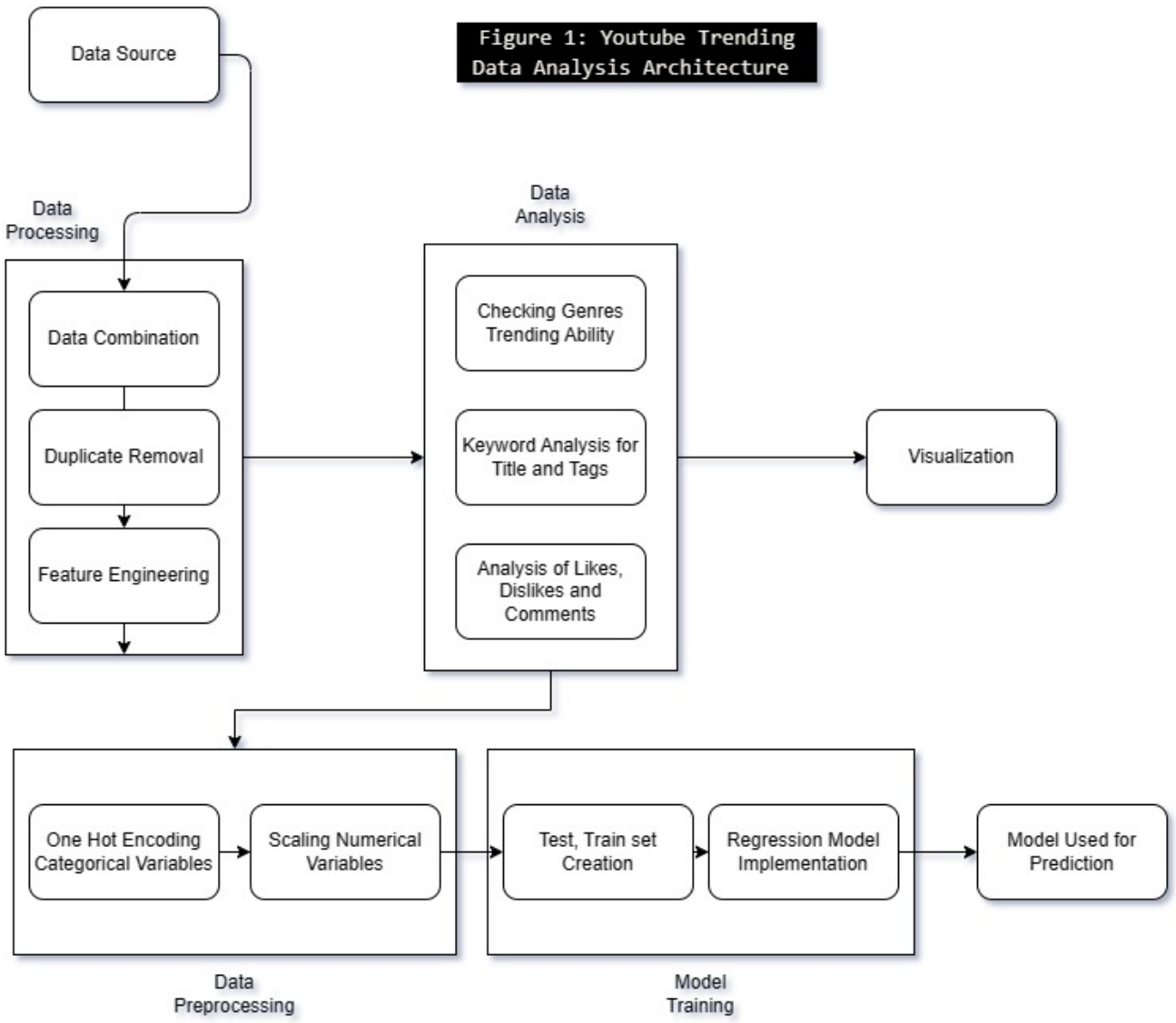


Figure 1: Youtube Trending Data Analysis Architecture

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 [1]:
```
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 [2]:
```
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 categor
ies['items']}

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

df.head()
```

Out[2]:

	video_id	title	publishedAt	channelId	channelTitle	categoryId	trend
0	J78aPJ3VyNs	I left youtube for a month and THIS is what ha...	2020-08-11T16:34:06Z	UCYzPXprvl5Y-Sf0g4vX-m6g	jacksepticeye	24	12T
1	9nidKH8cM38	TAXI CAB SLAYER KILLS 'TO KNOW HOW IT FEELS'	2020-08-11T20:00:45Z	UCFMbX7frWZfuWdjAML0babA	Eleanor Neale	27	12T
2	M9Pmf9AB4Mo	Apex Legends | Stories from the Outlands – “Th...	2020-08-11T17:00:10Z	UC0ZV6M2THA81QT9hrVWJG3A	Apex Legends	20	12T
3	kgUV1MaD_M8	Nines - Clout (Official	2020-08-10T18:30:28Z	UCvDkzrj8ZPlBqRd6flxdhTw	Nines	24	12T

		(Official Video)					
4	49Z6Mv4_WCA	i don't know what im doing anymore	2020-08-11T20:24:34Z	UCtinbF-Q-fVthA0qrFQTgXQ	CaseyNeistat	22	12T

Data Quality Check

In [3]:
```
df.shape
```

Out[3]:
```
(238391, 17)
```

We see that there are `17` features (columns) and `238391` 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 [4]:
```
df.isna().sum()
```

Out[4]:
```
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 `4281` missing descriptions and `102` 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 [5]:
```
df.duplicated().sum()
```

Out[5]:
```
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 [6]:
```
df = df.drop_duplicates()
df.shape
```

Out[6]:
```
(238267, 17)
```

Check the important features

Convert the `trending_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 [7]:
```
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','May','Jun','Jul','Aug','Sep','Oct','Nov','Dec'))
df['month'] = pd.Categorical(df['month'], categories=['Jan','Feb','Mar','Apr','May','Jun','Jul','Aug','Sep','Oct','Nov','Dec'], ordered=True)
```

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 [8]:
```
df['test_hour'] = df['date'].dt.hour
df['test_hour'].unique()
```

Out[8]:
```
array([0])
```

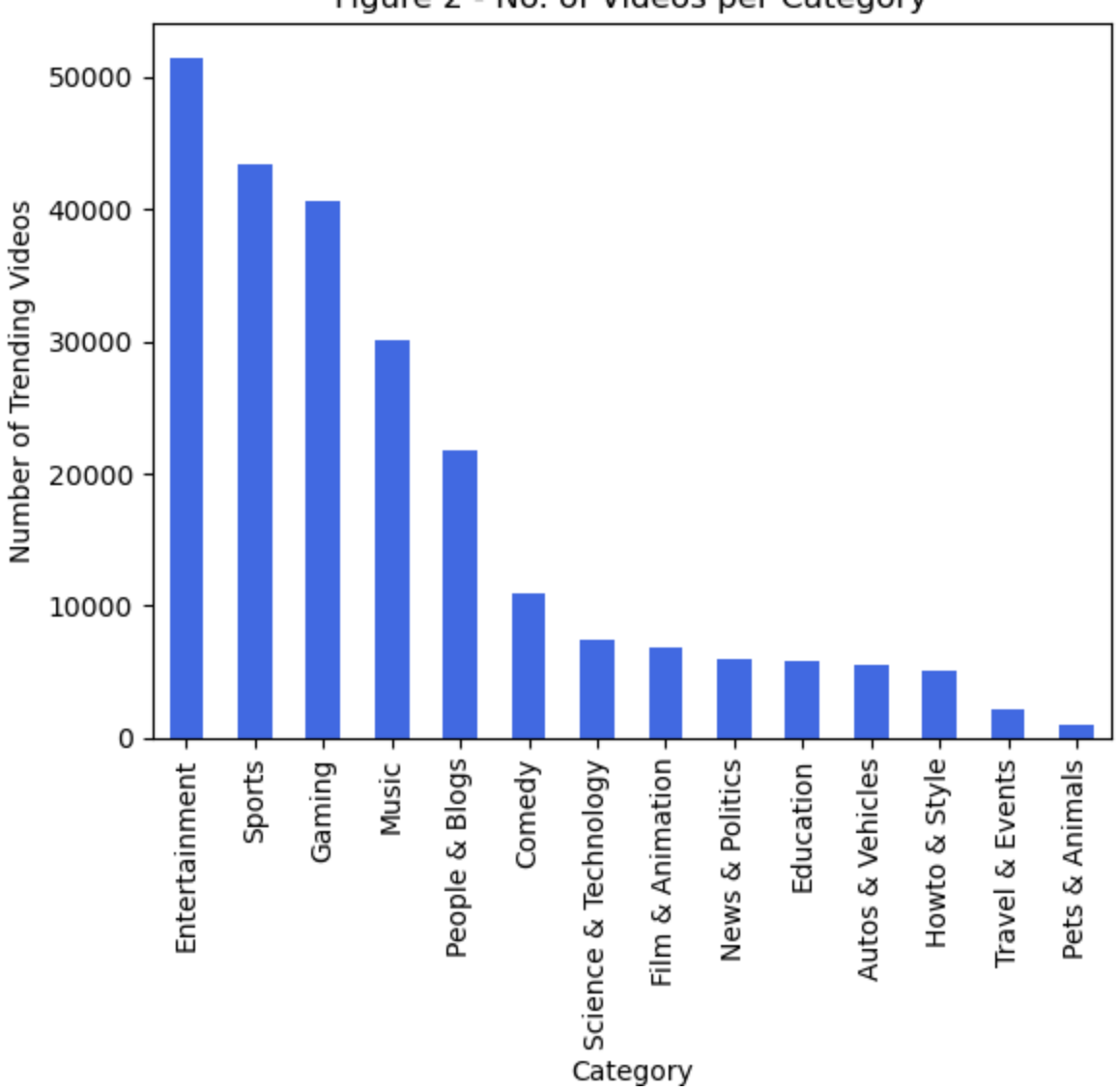
In [9]:
```
df['publishedAt'] = pd.to_datetime(df['publishedAt'])
df['num_month'] = df['publishedAt'].dt.month
df['hour'] = df['publishedAt'].dt.hour
```

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 [10]:
```
fig2 = df['category'].value_counts()
fig2.plot.bar(title='Figure 2 - No. of Videos per Category',xlabel='Category',ylabel='Number of Trending Videos',color='royalblue')
```

Out[10]:
```
<Axes: title={'center': 'Figure 2 - No. of Videos per Category'}, xlabel='Category', ylabel='Number of Trending Videos'>
```

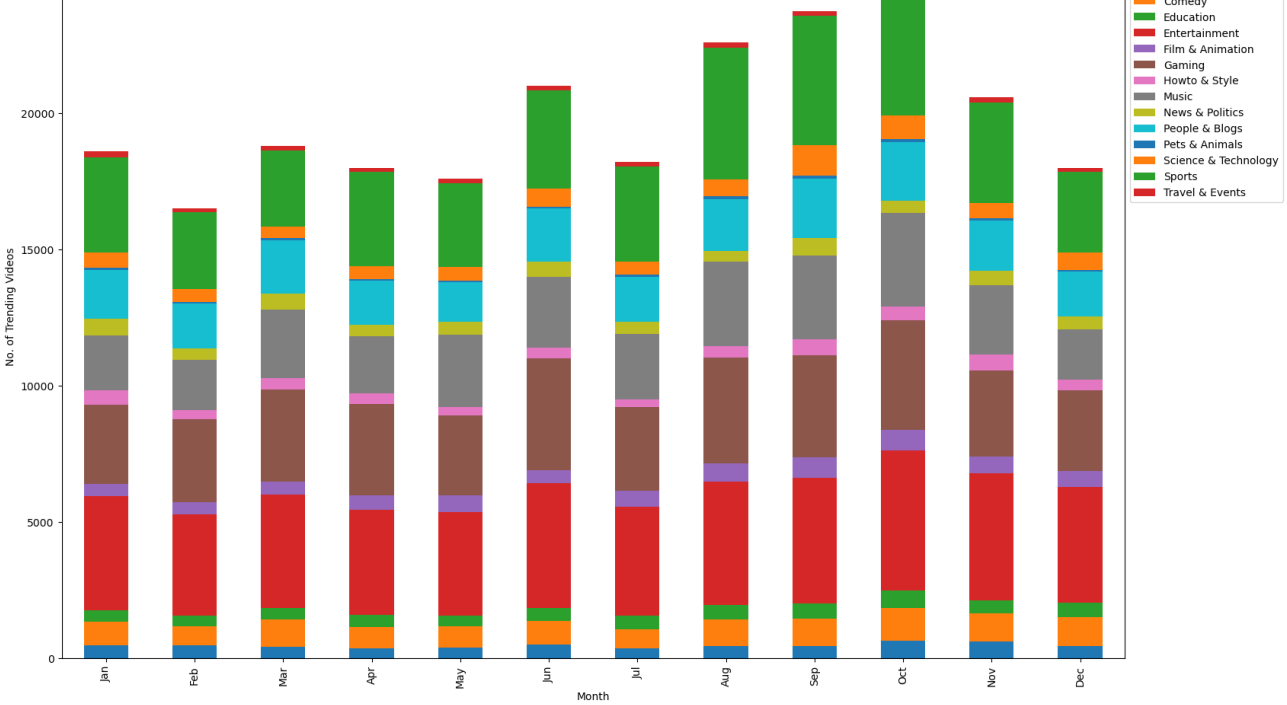


As we can see from `Figure 2`, Entertainment tops the list, followed by Sports, Gaming, Music, and People & Blogs as the Top 5 Categories across the dataset for the year between Aug 2020 and Nov 2023.

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 [11]:
```
#plot bar graph based on category and date by month
fig3 = df.groupby(['month','category']).size().unstack()
fig3.plot(kind='bar',stacked=True,figsize=(18,12),title="Figure 3 - No. of Vid
eos per Category by Month",xlabel="Month",ylabel='No. of Trending Videos').leg
end(bbox_to_anchor=(1.0, 1.0),title="Category")
```

Out[11]:
```
<matplotlib.legend.Legend at 0x1a5bcaf6590>
```



We are using a stacked barplot to compare the number of trending videos each month. As shown in `Figure 3`, 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 2 have almost a similar spread across the plot.

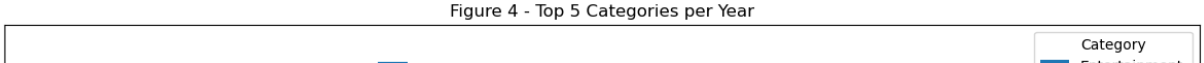
The spike in the number of trending videos between August and November could be because of the range of months of the trending videos in the dataset. That is, our range is between August 2020 to November 2023. We will further do analysis into finding how big of a difference is produced between each year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

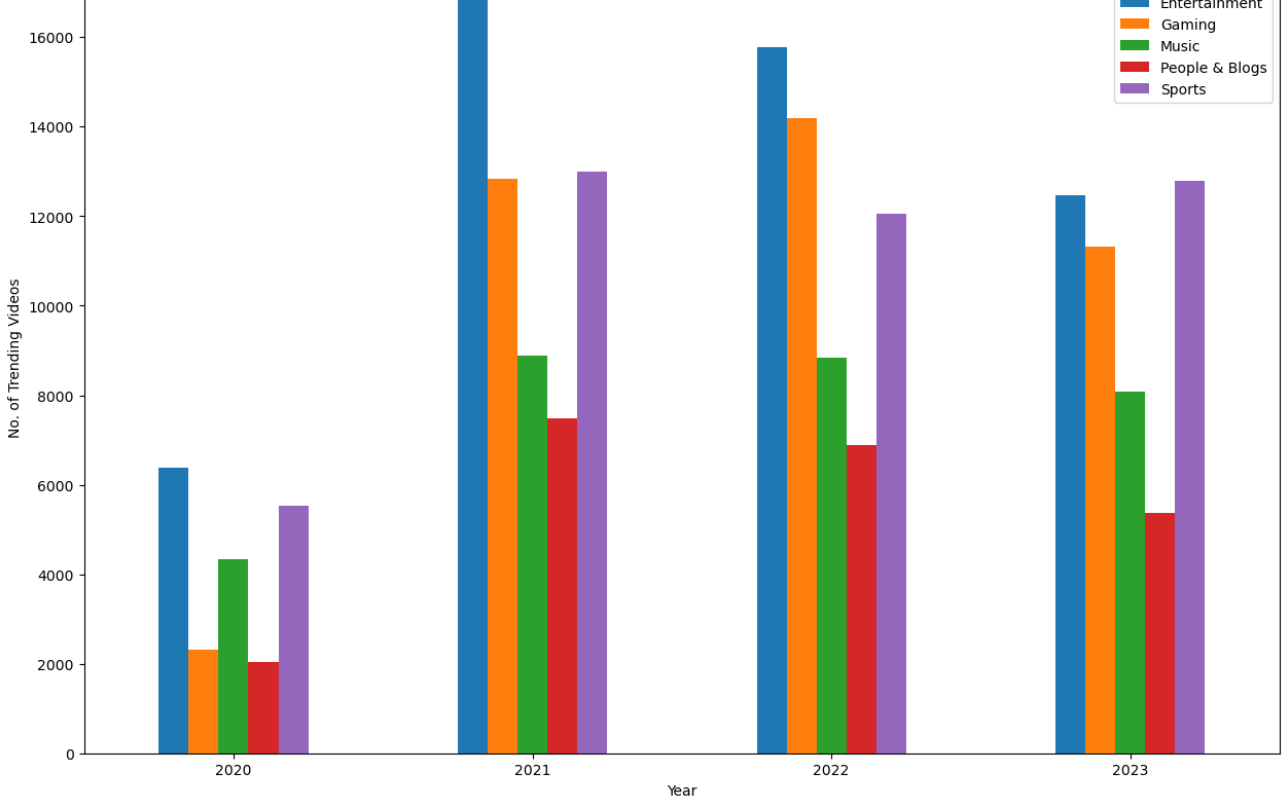
The data frame is further filtered to the Top 5 categories per `Figure 2`.

In [12]:
```
# plot grouped bar graph for top 5 category per year
fig4 = df.groupby(['year','category']).size().groupby(level=0).nlargest(5).unstack()
fig4.plot(kind='bar',stacked=False,figsize=(15,10),xlabel="Year",ylabel='No. of Trending Videos',title="Figure 4 - Top 5 Categories per Year").legend(bbox_to_anchor=(1.0, 1.0),title="Category")
plt.xticks(np.array([0,1,2,3]), ('2020','2021','2022','2023'),rotation=0)
```

Out[12]:
```
([<matplotlib.axis.XTick at 0x1a5c1c489d0>,
  <matplotlib.axis.XTick at 0x1a5bbae2490>,
  <matplotlib.axis.XTick at 0x1a5c1c3dd90>,
  <matplotlib.axis.XTick at 0x1a5c1ec6850>],
 [Text(0, 0, '2020'),
  Text(1, 0, '2021'),
  Text(2, 0, '2022'),
  Text(3, 0, '2023')])
```

Figure 4 - Top 5 Categories per Year





`Figure 4` 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 [13]:
```
!pip install wordcloud
```

```
Requirement already satisfied: wordcloud in c:\users\asish\anaconda3\lib\sit
e-packages (1.9.2)
Requirement already satisfied: numpy>=1.6.1 in c:\users\asish\anaconda3\lib
\site-packages (from wordcloud) (1.24.3)
Requirement already satisfied: pillow in c:\users\asish\anaconda3\lib\site-p
ackages (from wordcloud) (9.4.0)
Requirement already satisfied: matplotlib in c:\users\asish\anaconda3\lib\si
te-packages (from wordcloud) (3.7.2)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\asish\anaconda3
\lib\site-packages (from matplotlib->wordcloud) (1.0.5)
Requirement already satisfied: cycler>=0.10 in c:\users\asish\anaconda3\lib
\site-packages (from matplotlib->wordcloud) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\asish\anaconda3
\lib\site-packages (from matplotlib->wordcloud) (4.25.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\asish\anaconda3
\lib\site-packages (from matplotlib->wordcloud) (1.4.4)
Requirement already satisfied: packaging>=20.0 in c:\users\asish\appdata\roa
ming\python\python311\site-packages (from matplotlib->wordcloud) (23.2)
Requirement already satisfied: pyparsing<3.1,>=2.3.1 in c:\users\asish\anaco
nda3\lib\site-packages (from matplotlib->wordcloud) (3.0.9)
```

```
Requirement already satisfied: python-dateutil>=2.7 in c:\users\asish\appdat
a\roaming\python\python311\site-packages (from matplotlib->wordcloud) (2.8.
2)
Requirement already satisfied: six>=1.5 in c:\users\asish\appdata\roaming\py
thon\python311\site-packages (from python-dateutil>=2.7->matplotlib->wordclo
ud) (1.16.0)
```

In [14]:
```
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.

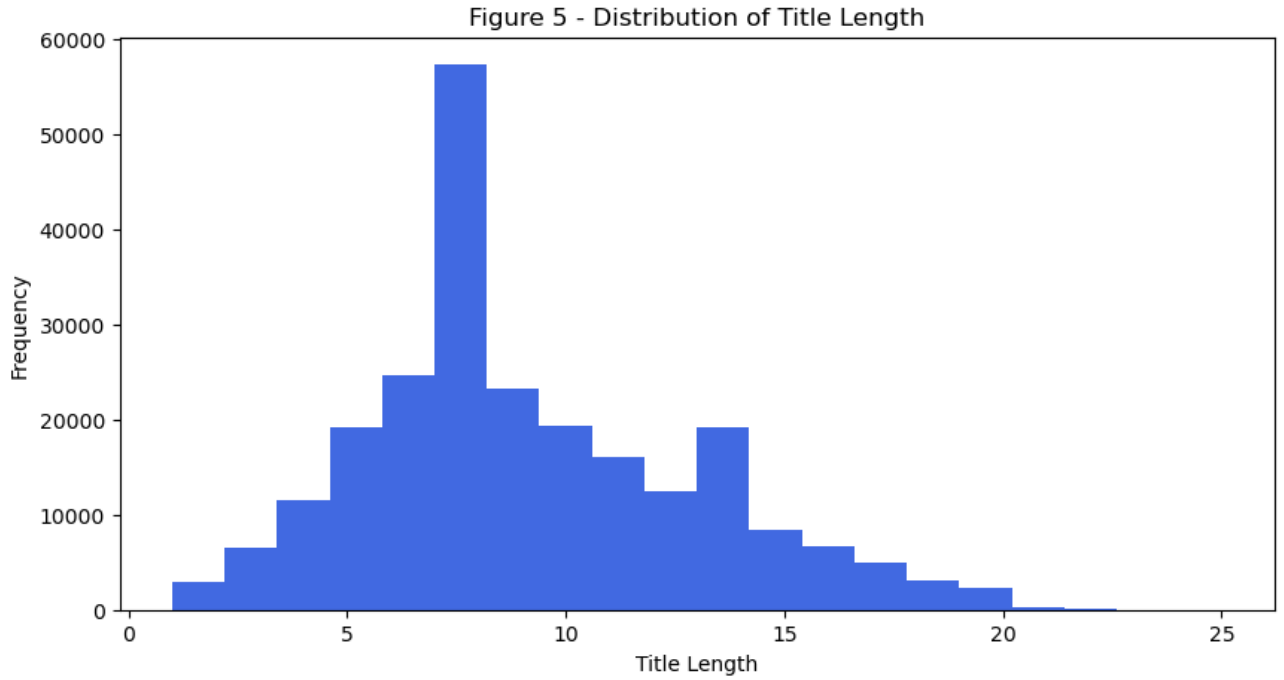
In [15]:
```
#find the average length of title in words
df_title_length = df.copy()
df_title_length['title_length'] = df_title_length['title'].str.split().str.len
()
df_title_length['title_length'].mean()
```

Out[15]:
```
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 [16]:
```
df_title_length['title_length'].plot.hist(bins=20,figsize=(10,5),title="Figure
 5 - Distribution of Title Length",xlabel="Title Length",ylabel="Frequency",co
lor='royalblue')
```

Out[16]:
```
<Axes: title={'center': 'Figure 5 - Distribution of Title Length'}, xlabel
='Title Length', ylabel='Frequency'>
```



`Figure 5` 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 [17]:
```
# Multiple graphs in the same cell
fig, axes = plt.subplots(7, 2, figsize=(15, 30))

# Get unique categories
categories = df_title_length['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_title_length[df_title_length['category'] == category]['title']
    text = ' '.join(title)
    if text != ' ':
        wordcloud = WordCloud(max_font_size=50, max_words=10000, background_co
lor="white").generate(text)
        ax.imshow(wordcloud, interpolation="bilinear")
        ax.set_title(category,fontsize=20)
        ax.axis("off")
        fig.suptitle('Figure 6 - Wordcloud of Trending Video Titles by Categor
y',fontsize=20)
        fig.tight_layout(rect=[0, 0.03, 1, 0.95])
```

```
C:\Users\asish\AppData\Local\Temp\ipykernel_10484\69008140.py:21: UserWarnin
g: The figure layout has changed to tight
  fig.tight_layout(rect=[0, 0.03, 1, 0.95])
```

Figure 6 - Wordcloud of Trending Video Titles by Category

`Figure 6` 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 [18]:
```
df_categories_cld = df.copy()
df_categories_cld['tag_length'] = df_title_length['tags'].str.split().str.len
()
df_categories_cld['tag_length'].mean()
```

Out[18]:
```
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 [19]:
```
# Multiple graphs in the same cell
fig, axes = plt.subplots(7, 2, figsize=(15, 30))

# Get unique categories
categories = df_title_length['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_title_length[df_title_length['category'] == category]['tags']
    text = ' '.join(tags)
    if text != ' ' or text == 'None':
        wordcloud = WordCloud(max_font_size=50, max_words=10000, background_co
lor="white").generate(text)
        ax.imshow(wordcloud, interpolation="bilinear")
        ax.set_title(category,fontsize=20)
        ax.axis("off")
        fig.suptitle('Figure 7 - Wordcloud of Trending Video tags by Category'
,fontsize=20)
        fig.tight_layout(rect=[0, 0.03, 1, 0.95])
```

```
C:\Users\asish\AppData\Local\Temp\ipykernel_10484\1252840587.py:21: UserWarn
ing: The figure layout has changed to tight
  fig.tight_layout(rect=[0, 0.03, 1, 0.95])
```

Figure 7 - Wordcloud of Trending Video tags by Category

`Figure 7` 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.

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 [20]:
```
def wordcloud_gen_v2(category):

    text1 = ' '.join(df_title_length[df_title_length['category'] == category]['title'])
    text2 = ' '.join(df_title_length[df_title_length['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="white").generate(common_text)
    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 [21]:
```
# Multiple graphs in the same cell
fig, axes = plt.subplots(7, 2, figsize=(15, 30))

# Get unique categories
categories = df_title_length['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,fontsize=20)
        ax.axis("off")
        fig.suptitle('Figure 8 - Wordcloud of Common Words in Trending Video T
ags and Title by Category',fontsize=20)
        fig.tight_layout(rect=[0, 0.03, 1, 0.95])
```

```
C:\Users\asish\AppData\Local\Temp\ipykernel_10484\292985644.py:21: UserWarni
ng: The figure layout has changed to tight
  fig.tight_layout(rect=[0, 0.03, 1, 0.95])
```

Figure 8 - Wordcloud of Common Words in Trending Video Tags and Title by Category

`Figure 8` 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 [22]:
```
df.columns
```

Out[22]:
```
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', 'yea
r',
       '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 [23]:
```
df_ob3 = df.copy()
df_ob3 = df.drop(['video_id','channelId','tags','channelTitle','thumbnail_lin
k','comments_disabled','ratings_disabled','date','month','year','description'
], axis=1)
df_ob3.head()
```

Out[23]:

	title	publishedAt	categoryId	trending_date	view_count	likes	dislikes	comment_count	
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	Er
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	
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	
3	Nines - Clout (Official Video)	2020-08-10 18:30:28+00:00	24	2020-08-12T00:00:00Z	613785	37567	669	2101	Er
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	

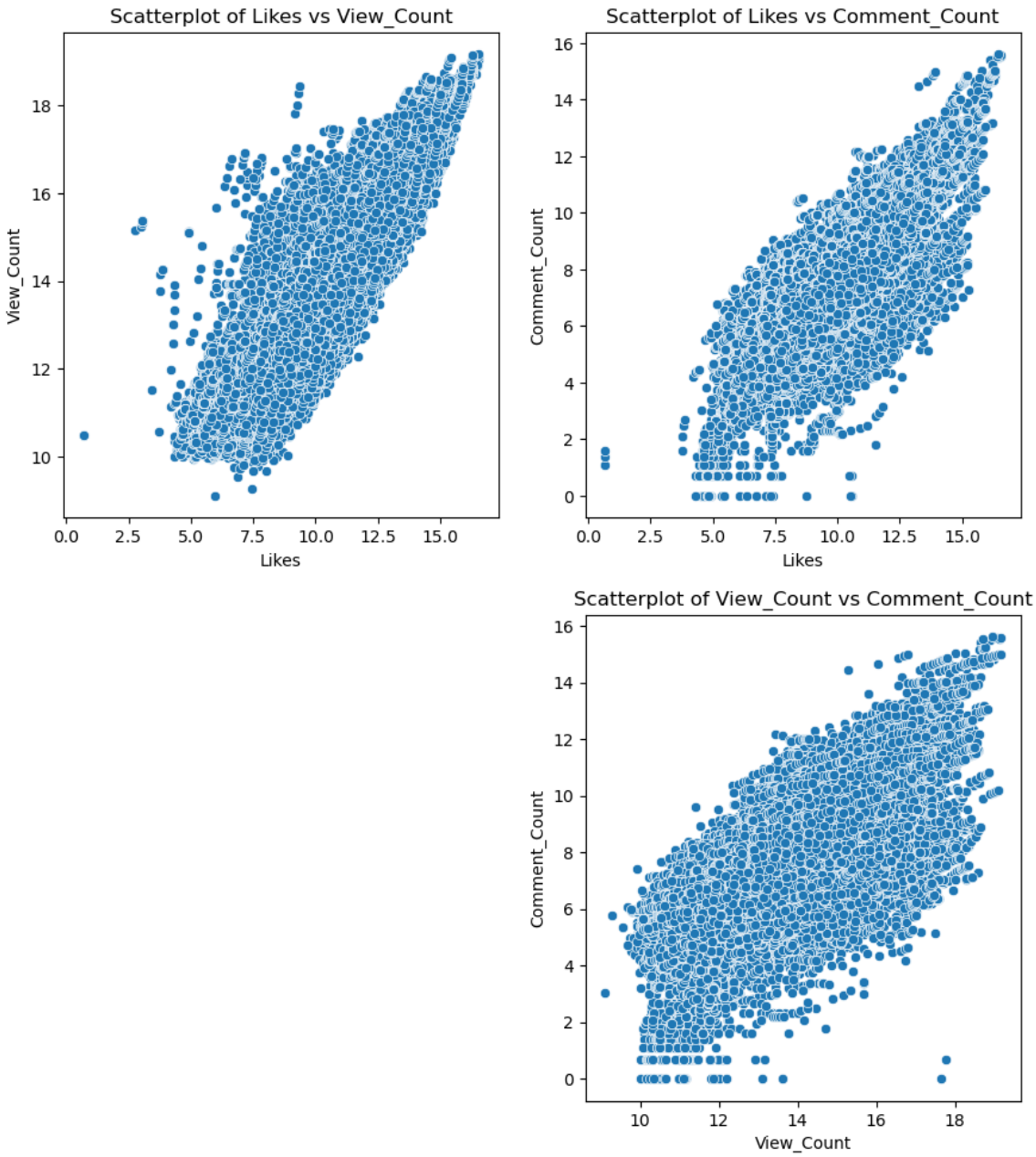
We now check the relation between likes, views, and comment count using scatterplot.

In [24]:
```
columns = ['likes', 'view_count', 'comment_count']
plt.figure(figsize=(16, 18))
#suptitle and align title to center
plt.suptitle('Figure 9 - Scatterplot of Likes, View Count and Comment Count',f
ontsize=20,x = .7, y = .95)


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]])).se
t(title=f'Scatterplot of {columns[i].title()} vs {columns[j].title()}', xlabel
=columns[i].title(), ylabel=columns[j].title())
```

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

```
timeWarning: divide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

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



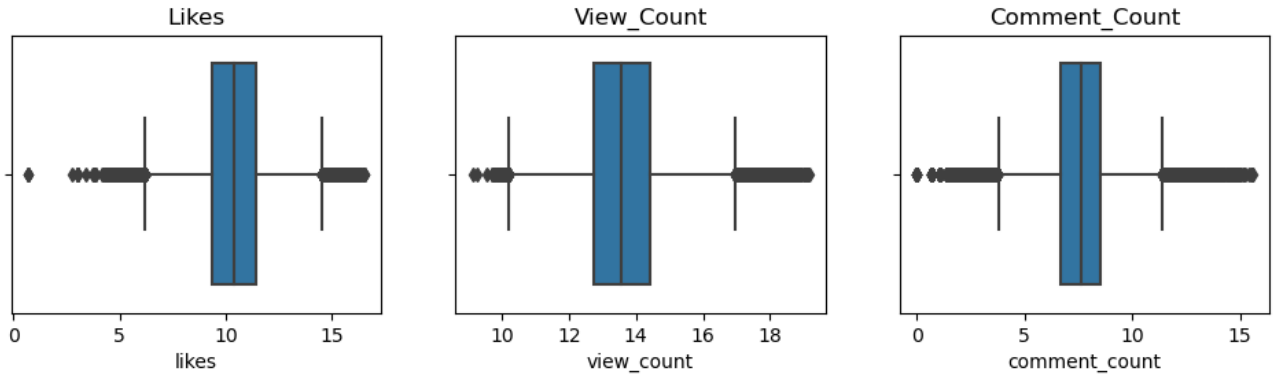
In `Figure 9`, we see that likes, view count, and comment count have a linear relationship, which means if the likes increase, the view count increases, and then the comment count also increases.

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 [25]:
```
columns = ['likes', 'view_count', 'comment_count']
plt.figure(figsize=(16, 6))
plt.suptitle('Figure 10 - Boxplot of Likes, View Count and Comment Count', fon
tsize=12,position=(0.4,1.05))
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 10 - Boxplot of Likes, View Count and Comment Count

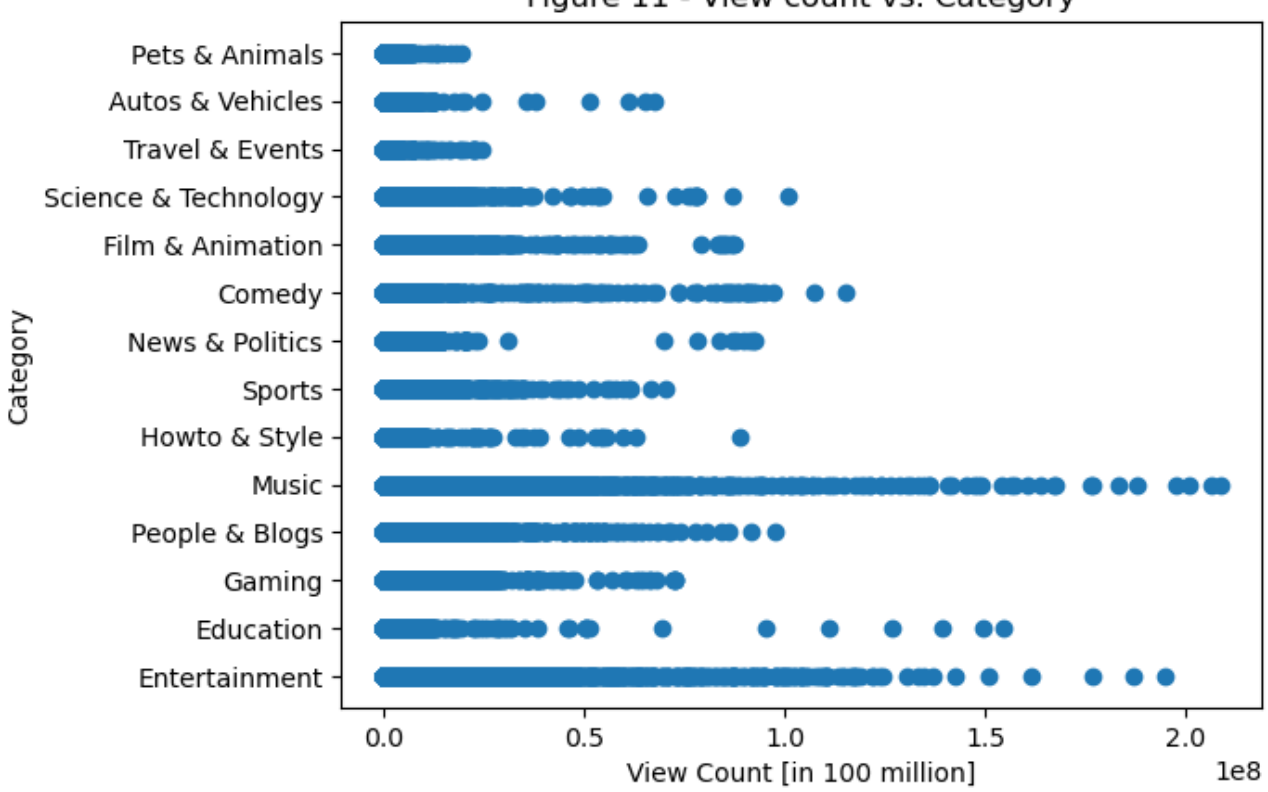


Plotting the boxplot with the actual count of likes, view_count and comment_count did not provide a graph that was convenient to read. Therefore, we have opted to take the logarithm of each values in the respective columns to better visualise it, which can be seen in `figure 10`.

Relation between view count and category

In [26]:
```
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 11 - View count vs. Category')
plt.show()
```

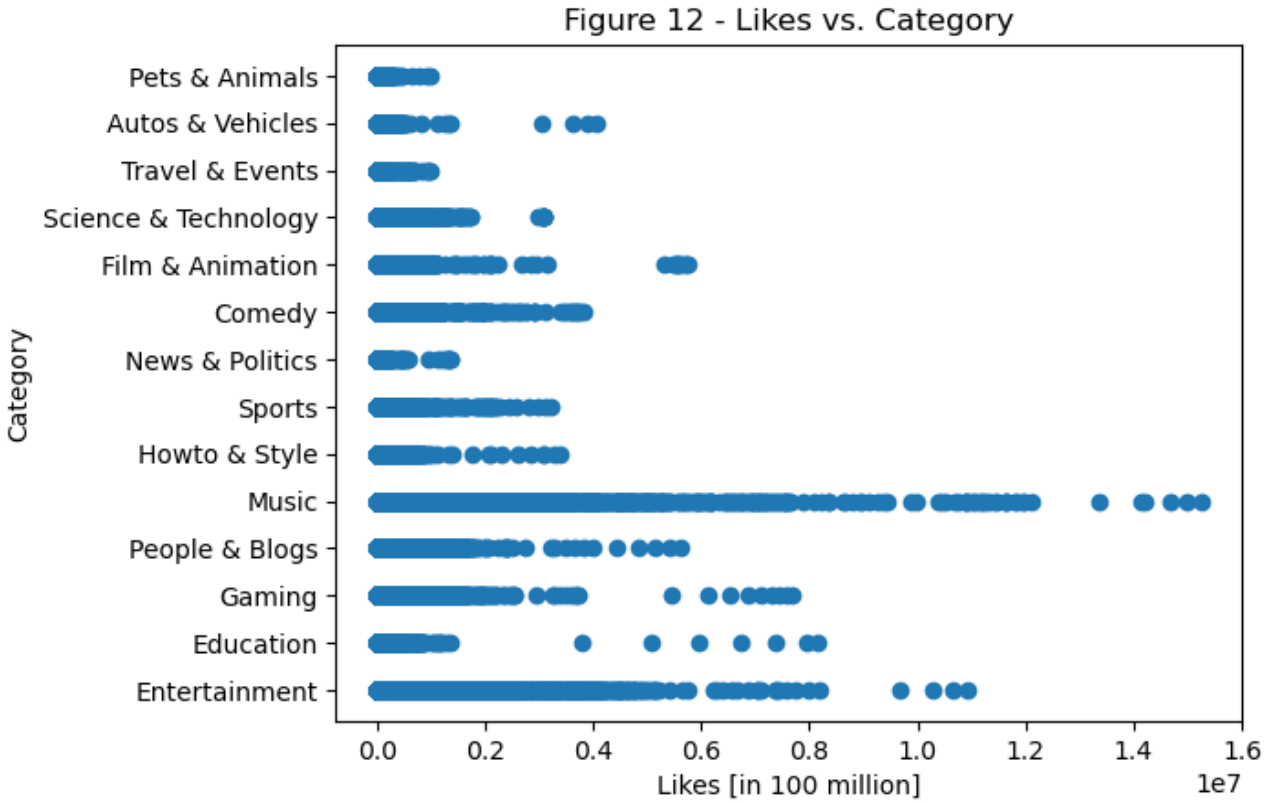
Figure 11 - View count vs. Category



`Figure 11` 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 [27]:
```
plt.scatter(df_ob3['likes'],df_ob3['category'].astype(str))
plt.xlabel('Likes [in 100 million]')
plt.ylabel('Category')
plt.title('Figure 12 - Likes vs. Category')
plt.show()
```



`Figure 12` 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 [28]:
```
df_ob3['title'][df_ob3['view_count'] > df_ob3['view_count'].mean()].count()
```

Out[28]:
```
51065
```

In [29]:
```
df_ob3['title'][df_ob3['likes'] > df_ob3['likes'].mean()].count()
```

Out[29]:
```
49285
```

In [30]:
```
df_ob3['title'][df_ob3['comment_count'] > df_ob3['comment_count'].mean()].coun
t()
```

Out[30]:
```
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 [31]:
```
df_ob3['title'][df_ob3['view_count'] == 0].count()
```

Out[31]:
```
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 [32]:
```
df_ob3['view_count_mean'] = (df_ob3['view_count'] > df_ob3['view_count'].mean
()).astype(int)
df_ob3['view_count_50'] = (df_ob3['view_count'] > df_ob3['view_count'].quantil
e(.5)).astype(int)
df_ob3['view_count_25'] = (df_ob3['view_count'] > df_ob3['view_count'].quantil
e(.25)).astype(int)
```

In [33]:
```
df_ob3['view_count'].quantile(.5)
```

Out[33]:
```
781351.0
```

In [34]:
```
df_ob3['view_count'].quantile(.25)
```

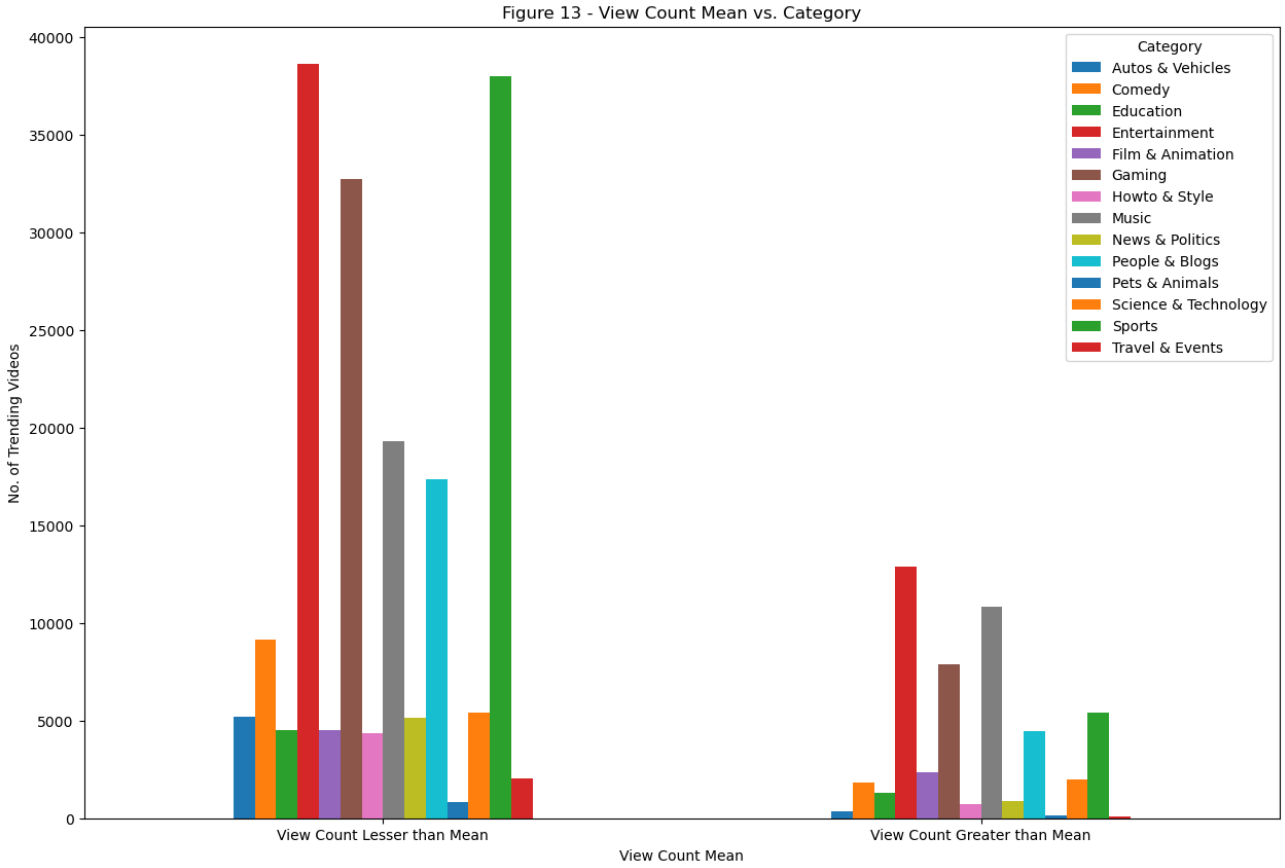
Out[34]:
```
343374.0
```

In [35]:
```
df_ob3['view_count'].mean()
```

Out[35]:
```
2168375.596808935
```

In [36]:
```
fig13 = df_ob3.groupby(['view_count_mean','category']).size().unstack()
fig13.plot(kind='bar',stacked=False,figsize=(15,10),title="Figure 13 - View Count Mean vs. Category",xlabel='View Count Mean',ylabel='No. of Trending Videos').legend(bbox_to_anchor=(1.0, 1.0),title="Category").axes.set_xticklabels(['View Count Lesser than Mean', 'View Count Greater than Mean'],rotation=0)
```

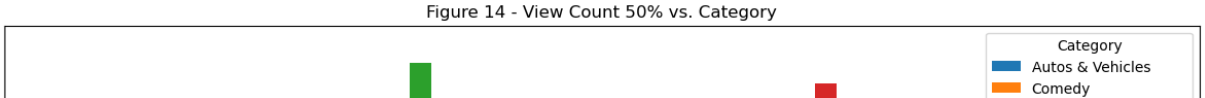
Out[36]:
```
[Text(0, 0, 'View Count Lesser than Mean'),
 Text(1, 0, 'View Count Greater than Mean')]
```

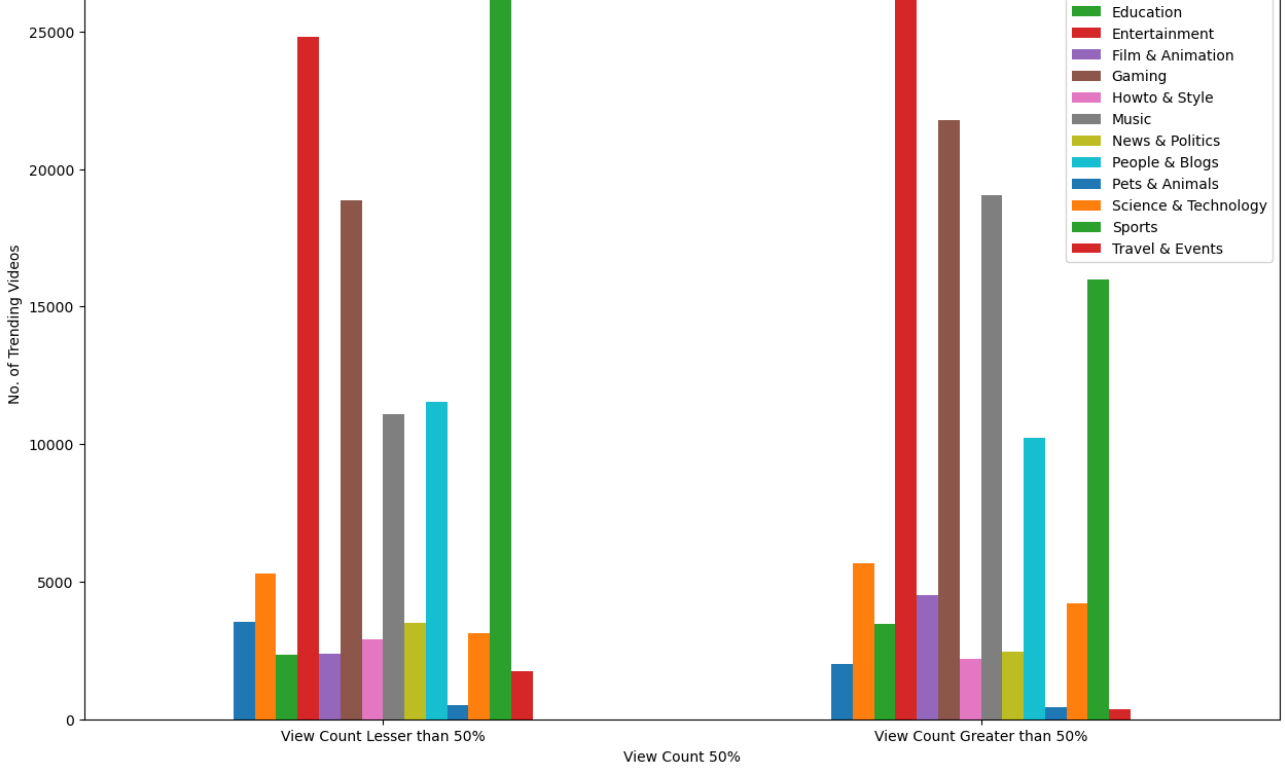


In `Figure 13`, we see that the majority of the videos have a view count that is lesser than the mean. We can conclude that vidoes are not required to accumulate views that cross the mean of the view counts of the trending videos (2168369.3647721303)

In [37]:
```
fig14 = df_ob3.groupby(['view_count_50','category']).size().unstack()
fig14.plot(kind='bar',stacked=False,figsize=(15,10),title="Figure 14 - View Count 50% vs. Category",xlabel='View Count 50%',ylabel='No. of Trending Videos').legend(bbox_to_anchor=(1.0, 1.0),title="Category").axes.set_xticklabels(['View Count Lesser than 50%', 'View Count Greater than 50%'],rotation=0)
```

Out[37]:
```
[Text(0, 0, 'View Count Lesser than 50%'),
 Text(1, 0, 'View Count Greater than 50%')]
```

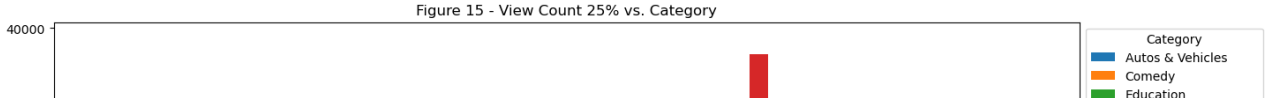


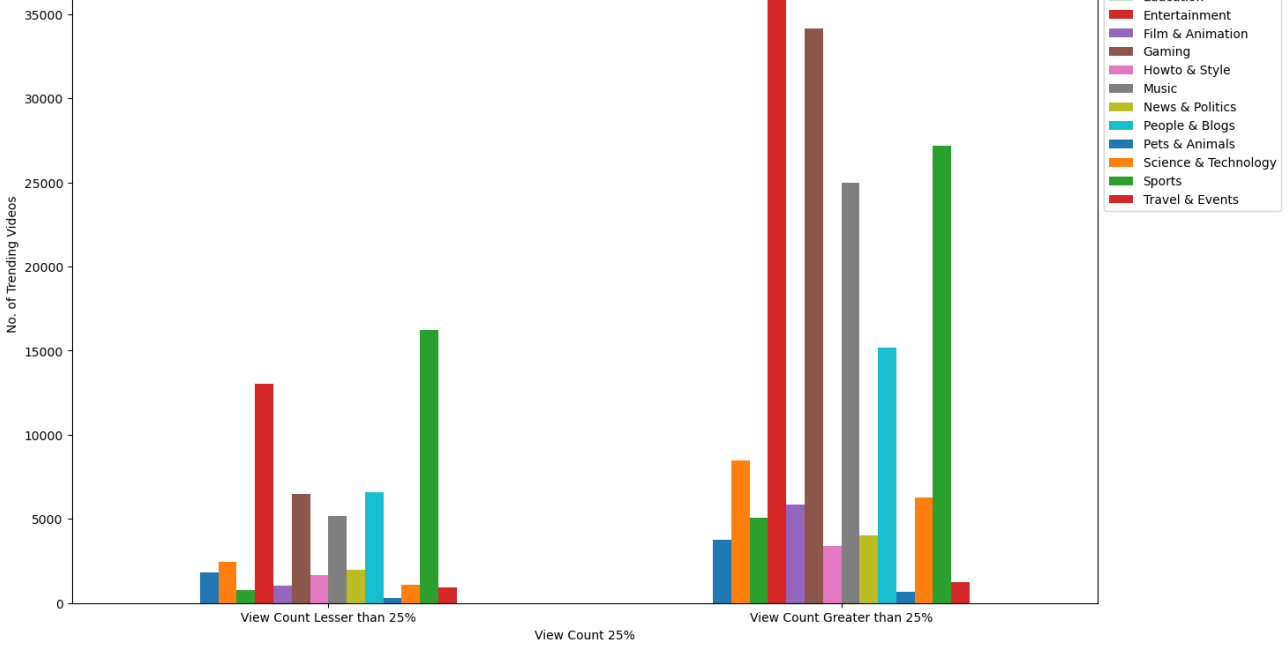


`Figure 14` shows the 50 percentile in view count. Some have the same number of videos on both sides, while more trending Sports videos are less than the 50 percentile, and more Gaming, Entertainment, and Music videos are above the 50 percentile.

In [38]:
```
fig15 = df_ob3.groupby(['view_count_25','category']).size().unstack()
fig15.plot(kind='bar',stacked=False,figsize=(15,10),title="Figure 15 - View Count 25% vs. Category",xlabel='View Count 25%',ylabel='No. of Trending Videos').legend(bbox_to_anchor=(1.0, 1.0),title="Category").axes.set_xticklabels(['View Count Lesser than 25%', 'View Count Greater than 25%'],rotation=0)
```

Out[38]:
```
[Text(0, 0, 'View Count Lesser than 25%'),
 Text(1, 0, 'View Count Greater than 25%')]
```





`Figure 15` shows the 25th percentile in view count. We see that the majority of the videos have a view count that is greater than the 25th percentile.

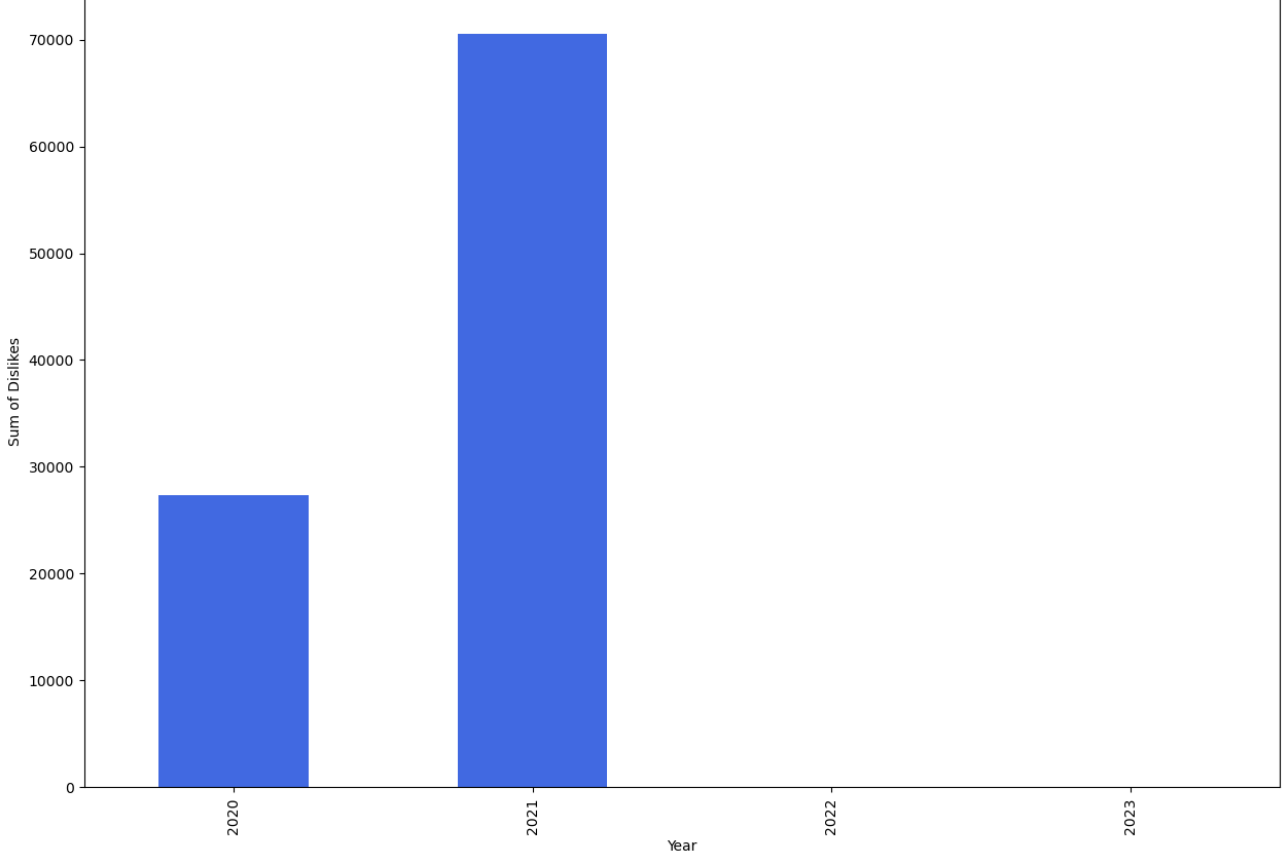
The above three graphs - Figures 13,14 and 15; gives us insights on the number of views a video has accumulated in the trending dataset. We can conclude that if a video successfully acquires views between the 25th percentile (358566.0 views) and 50th percentile (781394.0 views), then the video has a 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 [39]:
```
# Extract year from 'date' column
df_dislikes = df.copy()
df_dislikes['year'] = df_dislikes['date'].dt.year
df_dislikes['dislikes_nonzero'] = df['dislikes'].apply(lambda x: 0 if x == 0 e
lse 1)

# Group by year and calculate sum of 'dislikes_zero'
fig16 =yearly_dislikes_nonzero = df_dislikes.groupby('year')['dislikes_nonzer
o'].sum()

# Plot bar graph
fig16.plot(kind='bar', figsize=(15,10), title="Figure 16 - Yearly Dislikes", x
label='Year', ylabel='Sum of Dislikes', color='royalblue')
```

Out[39]:
```
<Axes: title={'center': 'Figure 16 - Yearly Dislikes'}, xlabel='Year', ylabe
l='Sum of Dislikes'>
```

Figure 16 - Yearly Dislikes



Thus, from `Figure 16`, 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 [40]:
```
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), ob
ject(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 [41]:
```
corr_list = ['hour','date','num_month','categoryId','year','likes','dislikes',
'comment_count','view_count']
df[corr_list].corr()
```

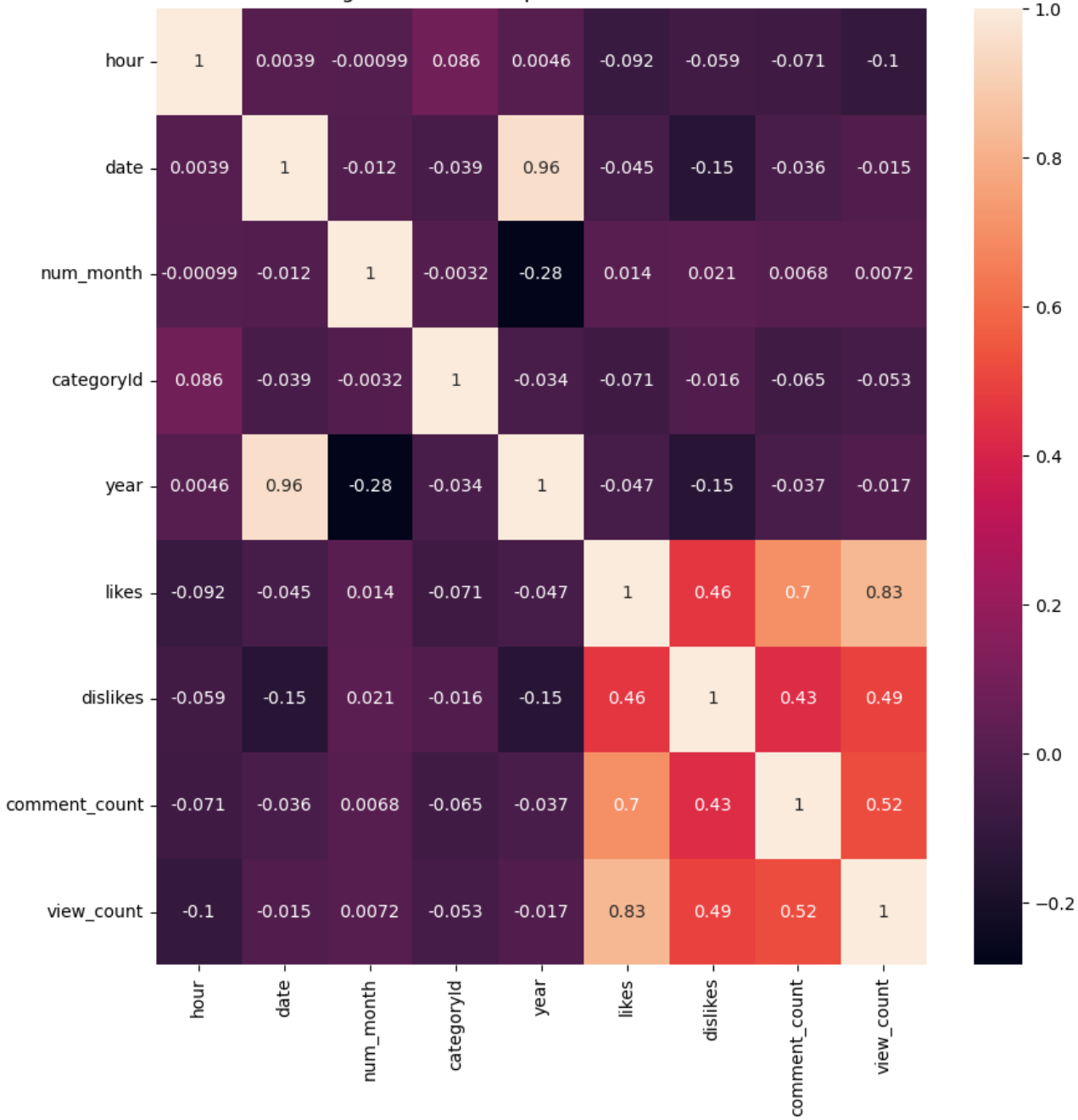
Out[41]:

	hour	date	num_month	categoryId	year	likes	dislikes	commen
hour	1.000000	0.003934	-0.000987	0.086237	0.004631	-0.092137	-0.059487	-0
date	0.003934	1.000000	-0.012425	-0.038589	0.957258	-0.044945	-0.146879	-0
num_month	-0.000987	-0.012425	1.000000	-0.003232	-0.283305	0.014440	0.020860	0
categoryId	0.086237	-0.038589	-0.003232	1.000000	-0.034452	-0.070634	-0.016357	-0
year	0.004631	0.957258	-0.283305	-0.034452	1.000000	-0.047108	-0.147124	-0
likes	-0.092137	-0.044945	0.014440	-0.070634	-0.047108	1.000000	0.461008	0
dislikes	-0.059487	-0.146879	0.020860	-0.016357	-0.147124	0.461008	1.000000	0
comment_count	-0.070505	-0.036353	0.006839	-0.064997	-0.037089	0.699849	0.429602	1
view_count	-0.104223	-0.015436	0.007151	-0.052694	-0.016781	0.829194	0.494547	0

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

In [42]:
```
plt.figure(figsize=(10,10))
fig17 = sns.heatmap(data = df[corr_list].corr(), annot=True).set_title("Figure
 16 - Heatmap of Correlation Matrix")
```

Figure 16 - Heatmap of Correlation Matrix

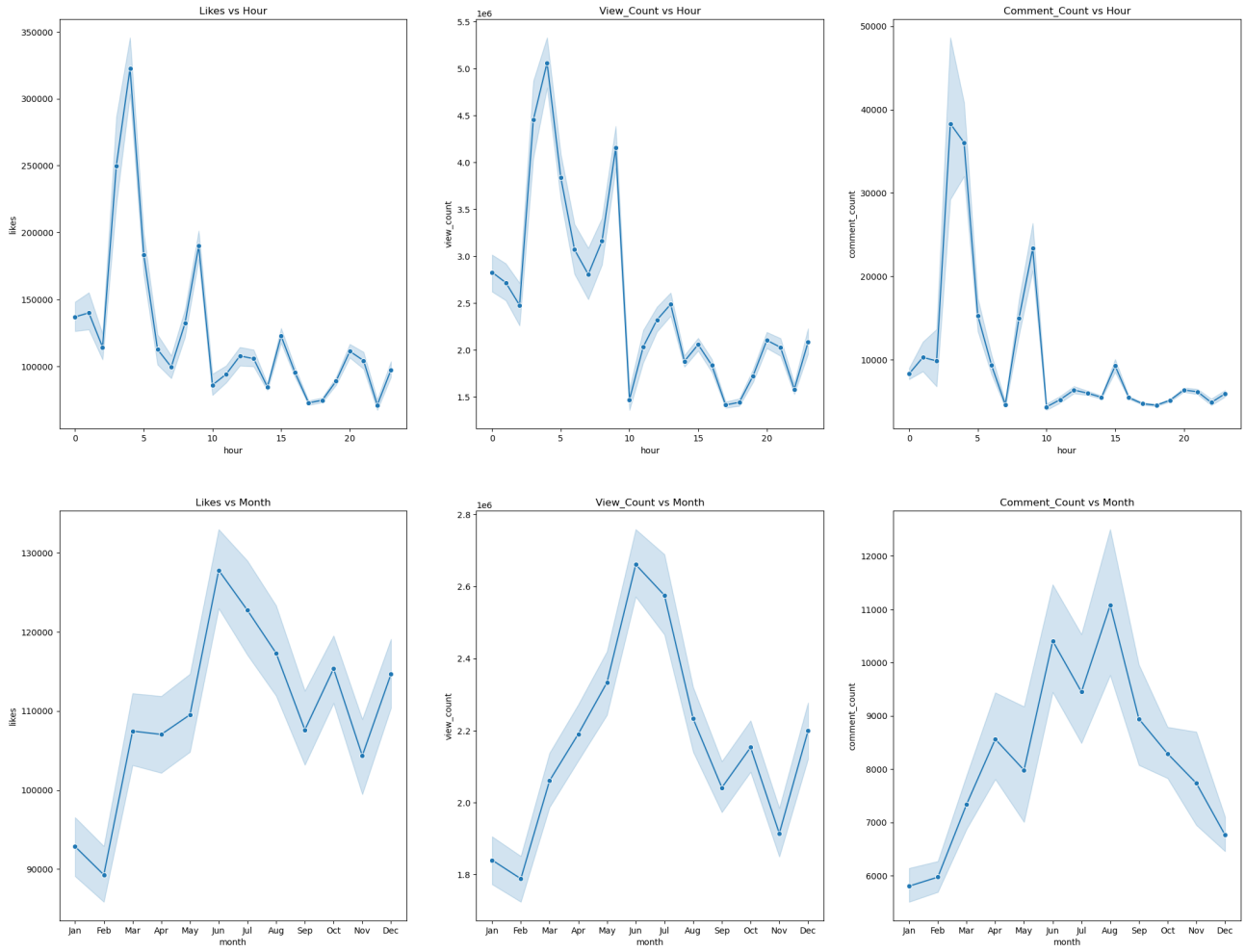


From `Figure 17`, we see that the likes and view count is high-positively correlated with 0.83 value. The dislikes and view count, the comment count and view count, and the dislikes and comment count are also correlated to a certain extent. The rest of the variables are neglibly correlated as the value 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 [43]:
```
f, ax = plt.subplots(2, 3, figsize=(25, 20))
variables = ['likes', 'view_count', 'comment_count']
x_values = ['hour','month']
plt.suptitle('Figure 18 - Lineplot of Likes, View Count and Comment Count vs H
our and Month',fontsize=20)
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_title(f'{var.title()} vs {x.title()}')
```

Figure 18 - Lineplot of Likes, View Count and Comment Count vs Hour and Month



`Figure 18` 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 18, 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.

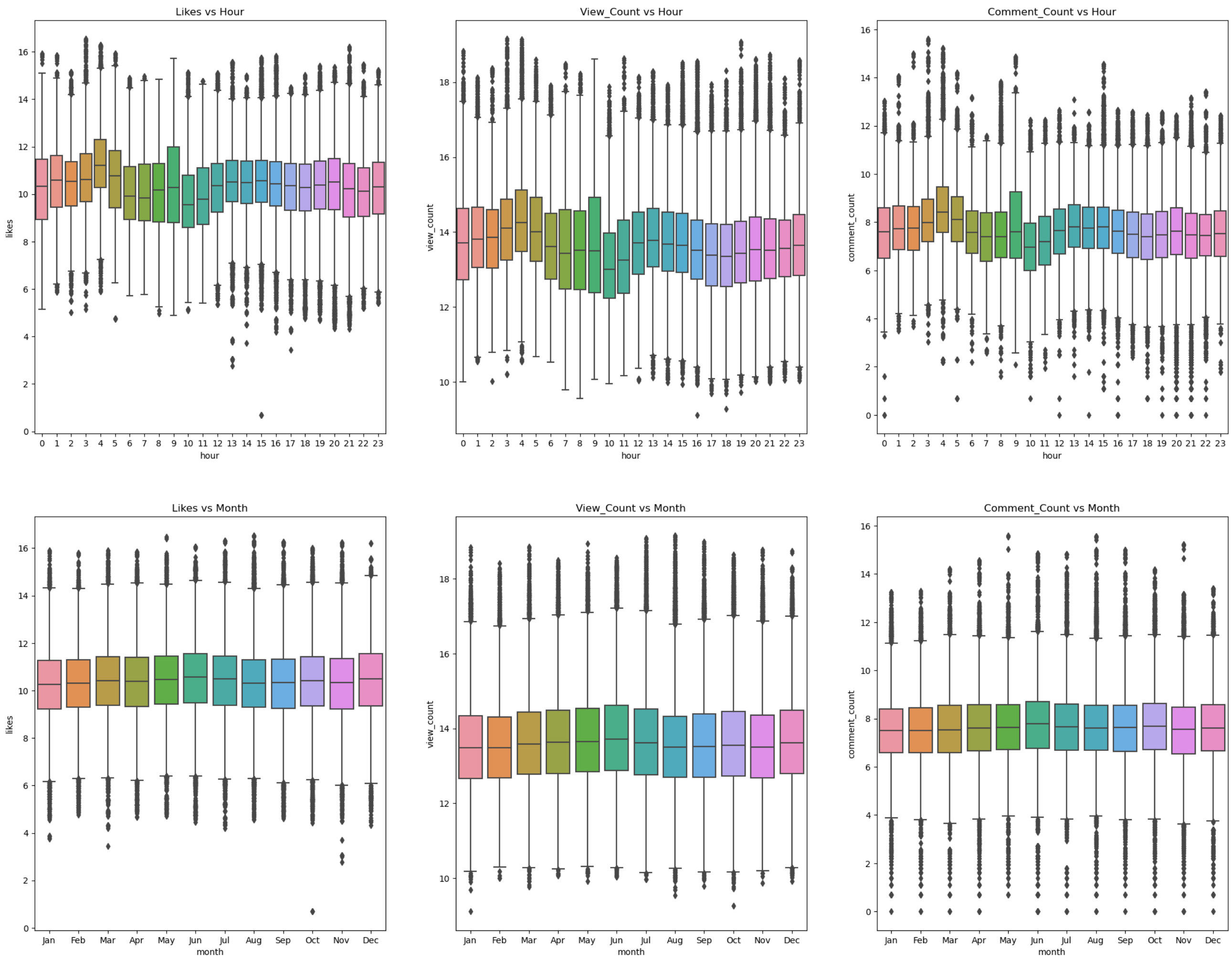
Although we have a time frame to see a spike of the video, we will now use the `boxplot()` to better visualize the spike hourly and monthly.

In [44]:
```
f, ax = plt.subplots(2, 3, figsize=(25, 20))
variables = ['likes', 'view_count', 'comment_count']
x_values = ['hour','month']
plt.suptitle('Figure 19 - Boxplot of Likes, View Count and Comment Count', fon
tsize= 20 ,position=(0.5,0.95))
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]).s
et_title(f'{var.title()} vs {x.title()}')
```

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

timeWarning: divide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

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

Figure 19 - Boxplot of Likes, View Count and Comment Count



From `Figure 19`, 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 [45]:
```
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
```

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 [46]:
```
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, year, description, video_id, channelId, tags, channelTitle, thumbnail_
link, comments_disabled, ratings_disabled, comment_count, likes, dislikes, vie
w_count, category, likes_mean, likes_50, view_count_mean, view_count_50, view_
count_25, view_to_like_ratio, like_to_view_ratio, dislikes_zero, dislikes_nonz
ero
df_ohe = df_ohe.drop(['title','publishedAt','trending_date','date','month','de
scription','video_id','channelId','tags','channelTitle','thumbnail_link','comm
ents_disabled','ratings_disabled','category'], axis=1)

# scale the numerical columns

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

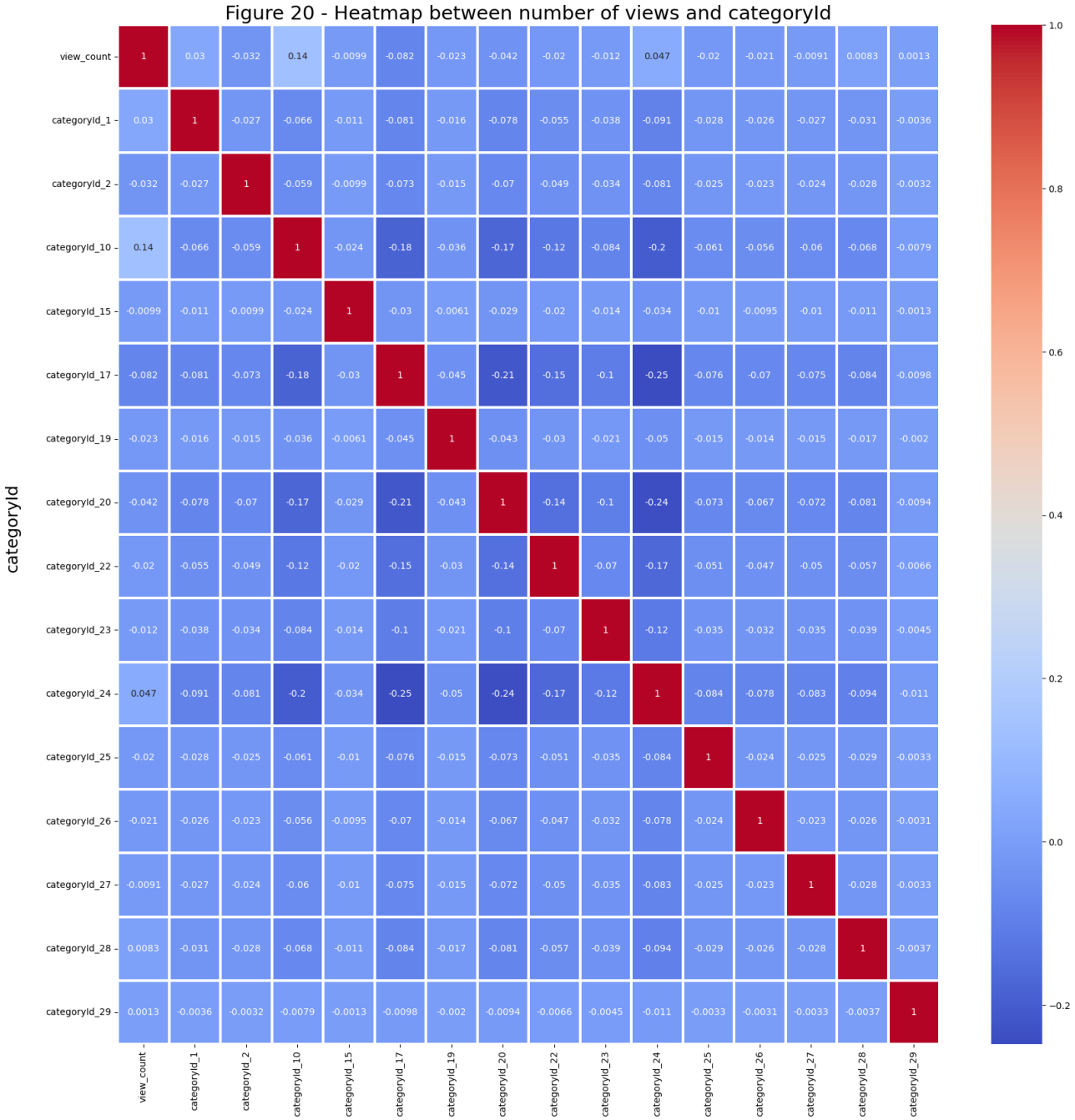
Out[46]:

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

5 rows × 60 columns

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

In [47]:
```
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"Figure 20 - Heatmap between number of views and {col}", fontsi
ze=21)

    f.text(0.5, 0.04, 'View Count', ha='center', fontsize=18)
    f.text(0.04, 0.5, f'{col}', va='center', rotation='vertical', fontsize=18)
```



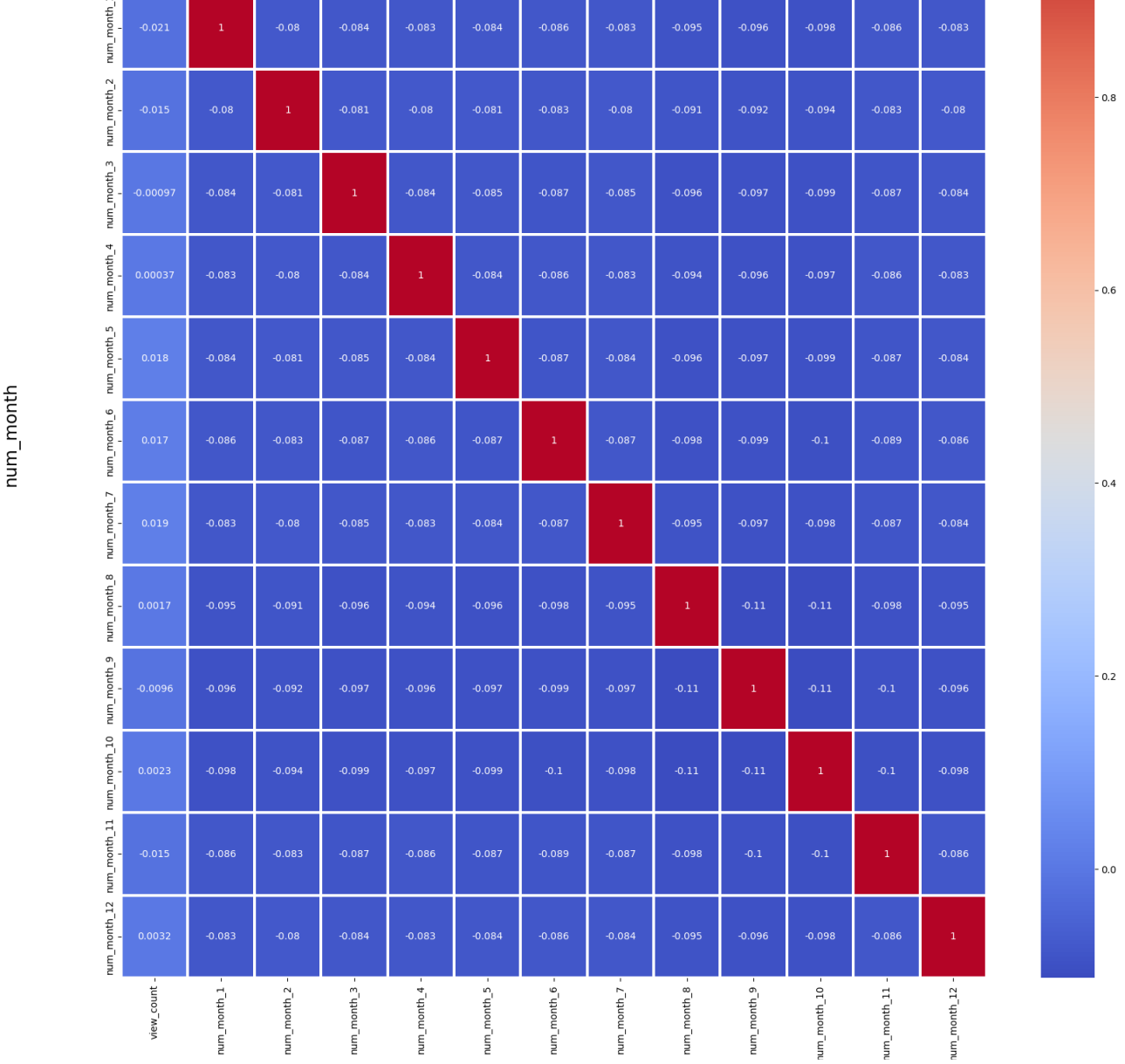
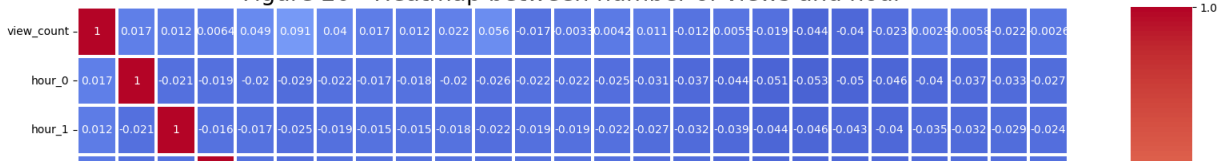


Figure 20 - Heatmap between number of views and hour



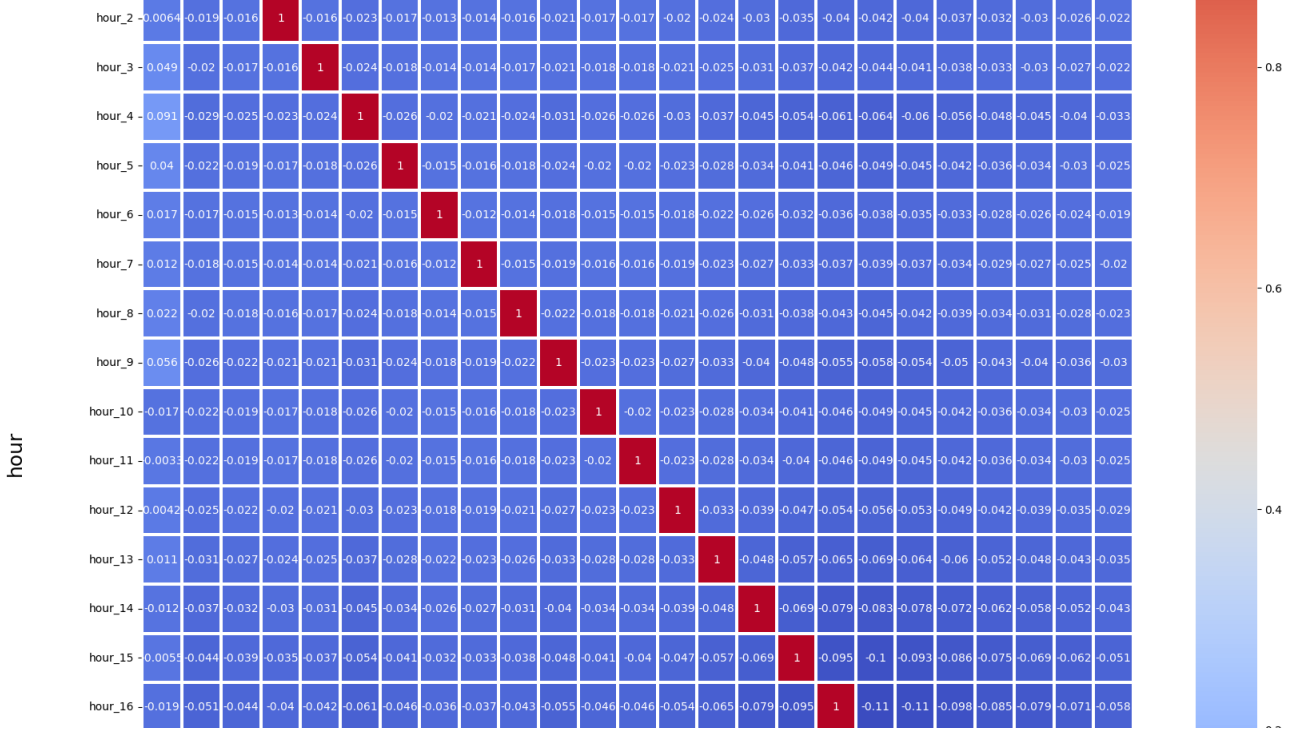
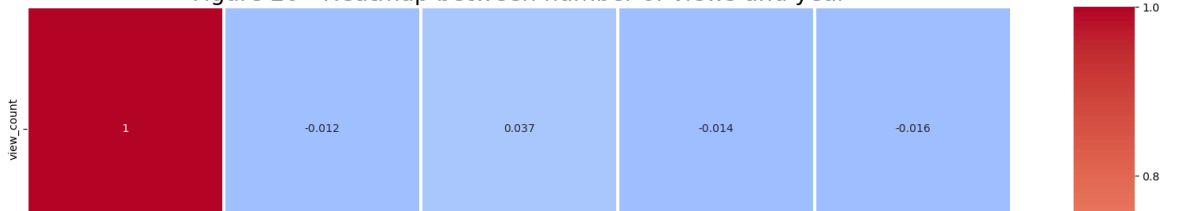
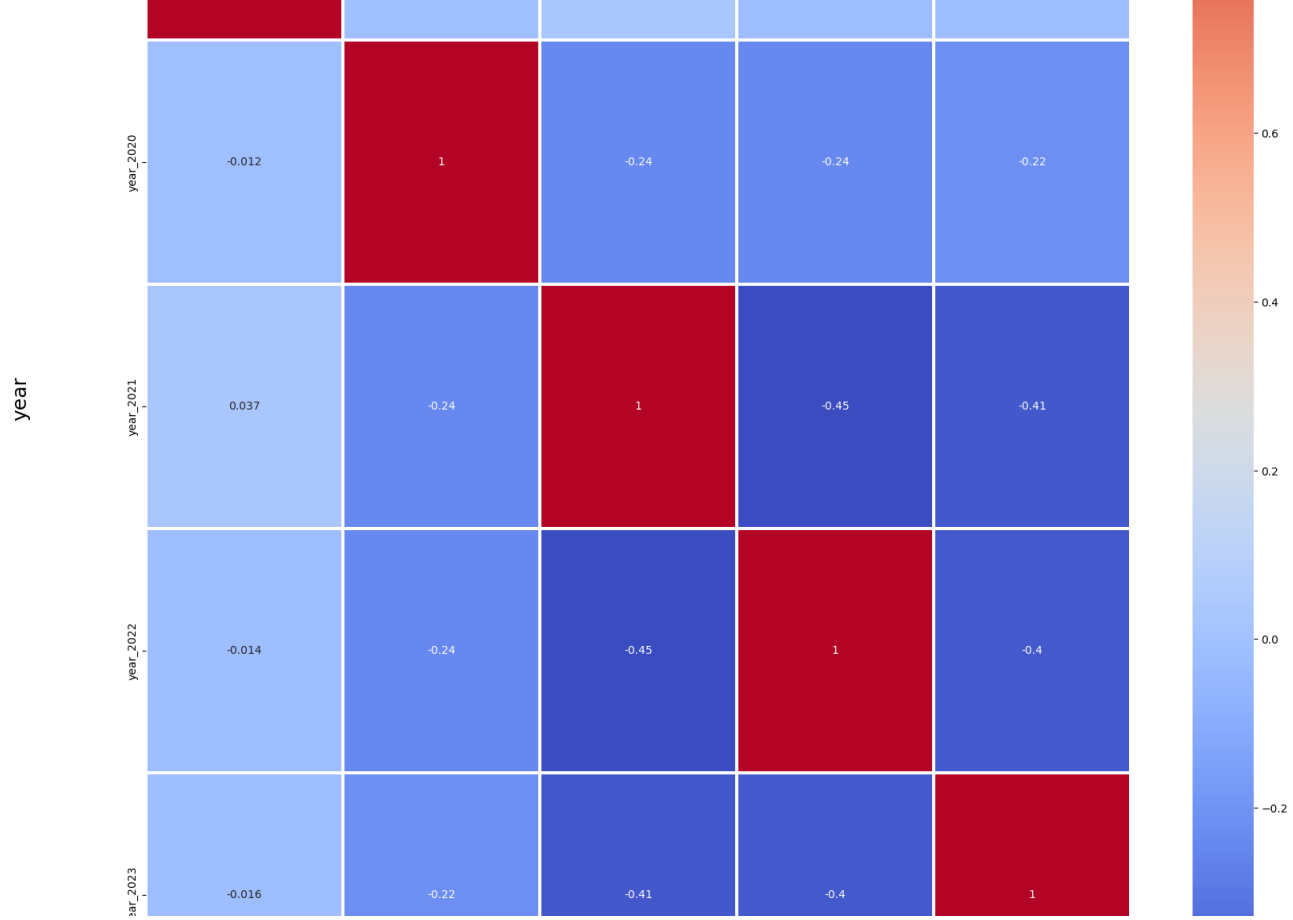


Figure 20 - Heatmap between number of views and year





Next, we can create a target and feature dataset and then split it inot a 20:80 test/train ratio

In [48]:
```
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, rando
m_state=42)
```

Creating and evaluating a LinearRegression model

Thirdly, 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 [57]:
```
#fit the model
model = LinearRegression()
model.fit(X_train, y_train)

#predict the model
y_pred1 = model.predict(X_test)
```

Evaluating the model

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

In [58]:
```
#evaluate the model
print("Mean squared error: %.2f" % mean_squared_error(y_test, y_pred1))
print('Variance score: %.2f' % r2_score(y_test, y_pred1))
```

```
print('Root Mean squared error: %.2f' % mean_squared_error(y_test, y_pred1, squared=False))
```

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

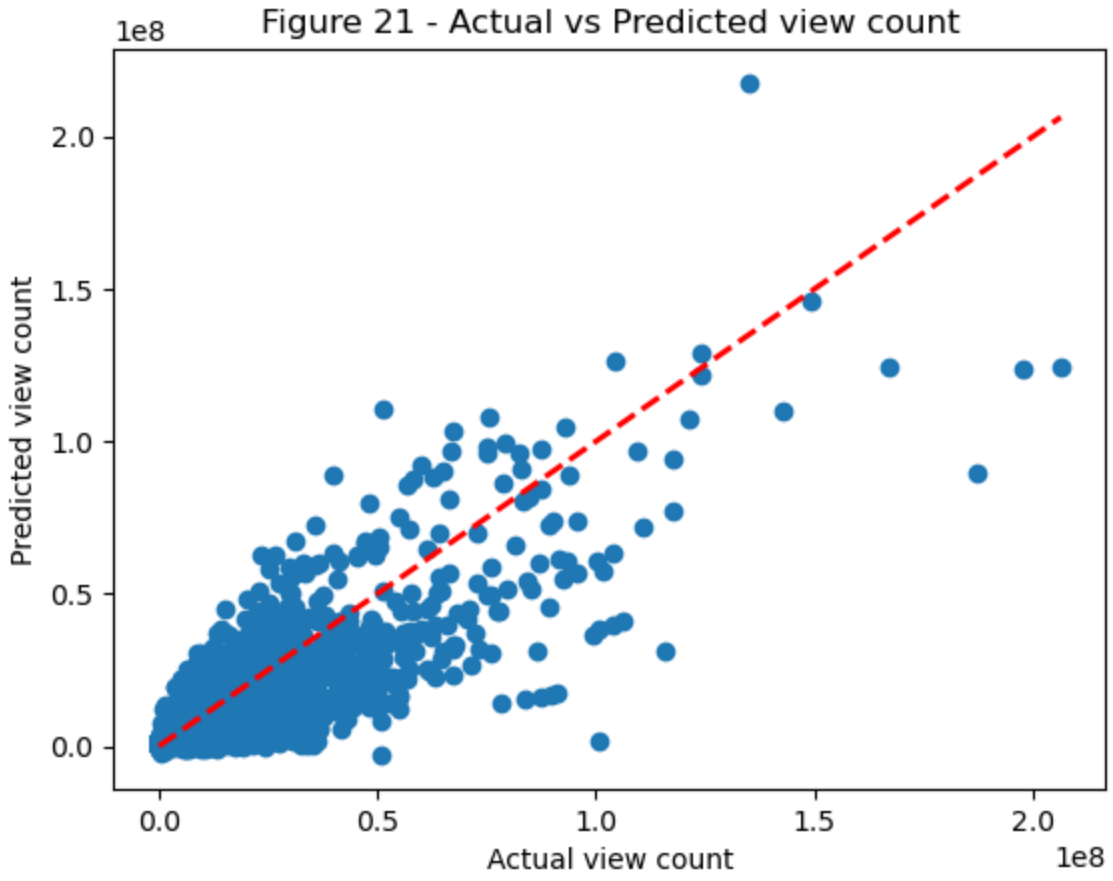
From the output above, we get a variance score of `0.72`. While this score is close to 1, we can try another model to ensure we get the best possible score.

Plotting Predicted vs Actual View Count For Linear Regression

When we plot the graph in `Figure 21` 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 [60]:
```
#plot the model
plt.scatter(y_test, y_pred1)
plt.xlabel("Actual view count")
plt.ylabel("Predicted view count")
plt.title("Figure 21 - 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_10484\3818589174.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_estimator` 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 [52]:
```
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_error(y_test, y_pred, squared=False)
```

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 [53]:
```
# 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 ,95 ,100]
# 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, 'R2': r2}])], ignore_index=True)
```

We can now see the dataframe with all the `n_estimators` along with their respective evaluation scores.

In [61]:
```
#rf_df.head(100)
```

Out[61]:

	n_estimators	MSE	RMSE	R2
0	1	5.601928e+12	2.366839e+06	0.841378
1	5	2.341829e+12	1.530304e+06	0.933689
2	10	2.068665e+12	1.438285e+06	0.941424
3	15	1.988388e+12	1.410102e+06	0.943697
4	20	1.877400e+12	1.370183e+06	0.946840
5	25	1.899484e+12	1.378218e+06	0.946215
6	30	1.902678e+12	1.379376e+06	0.946124
7	35	1.912979e+12	1.383105e+06	0.945833
8	40	1.895742e+12	1.376859e+06	0.946321
9	45	1.883224e+12	1.372306e+06	0.946675
10	50	1.859861e+12	1.363767e+06	0.947337
11	55	1.859777e+12	1.363736e+06	0.947339
12	60	1.861093e+12	1.364219e+06	0.947302
13	65	1.853895e+12	1.361578e+06	0.947506
14	70	1.862045e+12	1.364568e+06	0.947275
15	75	1.860892e+12	1.364145e+06	0.947308
16	80	1.868695e+12	1.367002e+06	0.947087
17	85	1.865324e+12	1.365769e+06	0.947182
18	90	1.864740e+12	1.365555e+06	0.947199
19	95	1.871739e+12	1.368115e+06	0.947000
20	100	1.865082e+12	1.365680e+06	0.947189

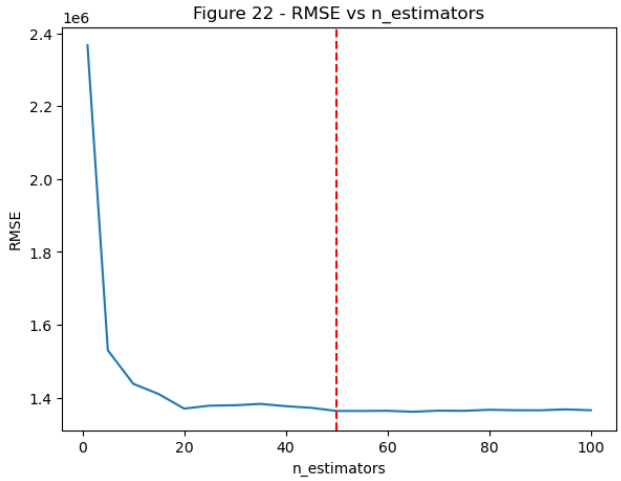
Selection of Estimator value

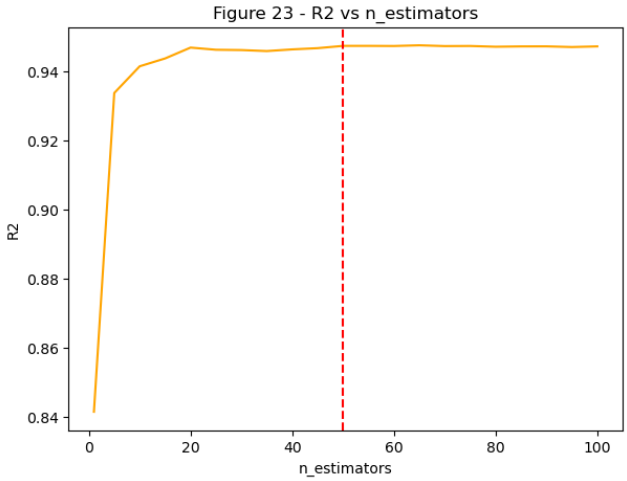
We can now plot `Figure 22` RMSE vs n_estimator and `Figure 23` R2 vs n_estimator. From the graph, we see that an increase in n_estimnator initially causes a rapid rise in R2 value, but only a marginal increase from `n_estimator` greater than 15. Furthermore, any increase in the value of `n_estimator` 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 [62]:
```
# 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 22 - 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 23 - 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 [64]:
```
# 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.

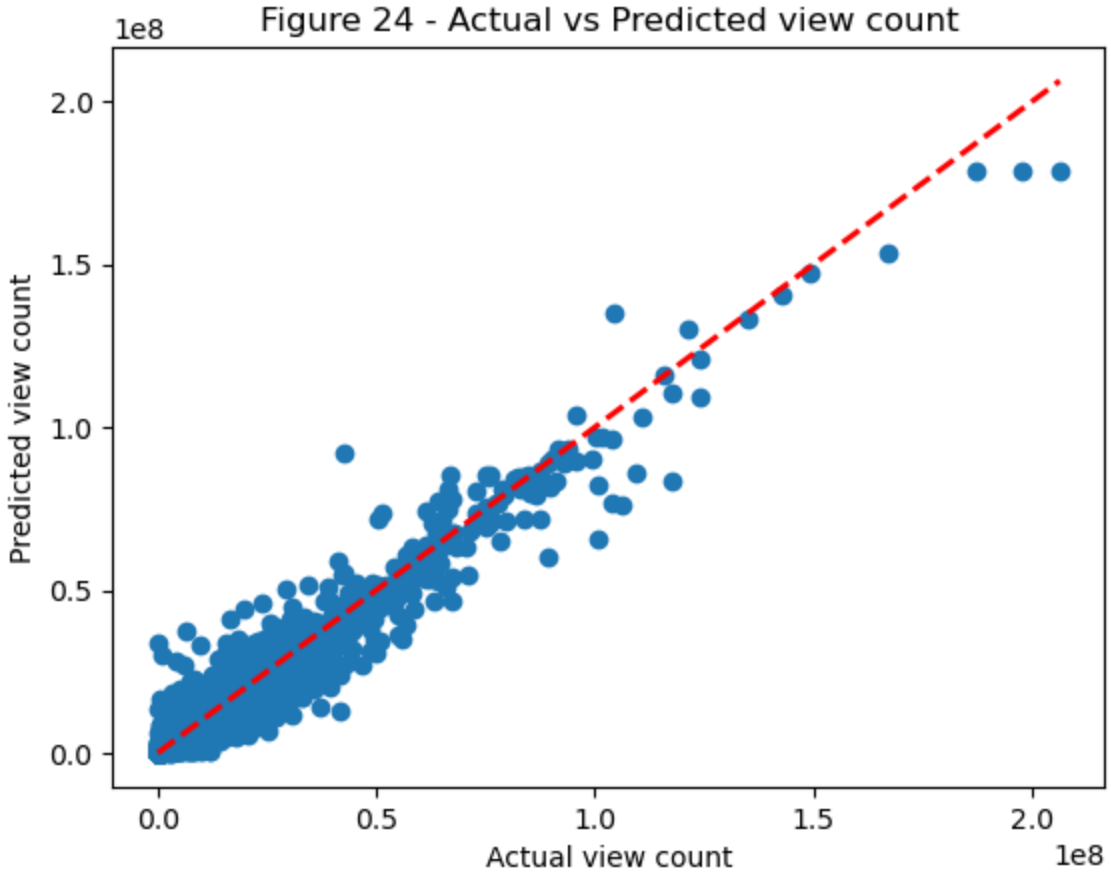
In [65]:
```
print("Mean squared error: %.2f" % mean_squared_error(y_test, y_pred))
print('Variance score: %.2f' % r2_score(y_test, y_pred))
print('Root Mean squared error: %.2f' % mean_squared_error(y_test, y_pred, squ
ared=False))
```

```
Mean squared error: 1859860906199.00
Variance score: 0.95
Root Mean squared error: 1363767.17
```

Plotting the graph of actual vs prediction in `Figure 24` shows our model has improved over just Linear Regression in determining `view_count` of Videos given a set of attributes about the video.

In [66]:
```
#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 24 - Actual vs Predicted view count")
plt.show()
```

```
C:\Users\asish\AppData\Local\Temp\ipykernel_10484\3876455301.py:5: UserWarni
ng: 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)

Overview of Results

Objective 1

Explanation of Results

We intended to find how each category trends and find out the top trending categories as well. So, we used the entire dataset to find its popularity for each category, and then we further investigated the trend of the categories for each month. Finally, we confirmed our findings by studying only the top 5 categories yearly.

We used the stacked and unstacked barplot to better understand the data. Below are our main findings:

- The Entertainment category has the most trending videos from August 2020 to November 2023, followed by Sports, Gaming, Music, and People & Blogs. We also noticed that the rest of the categories are fivefold less than the Entertainment Category. The Travel & Events and the Pets & Animals categories have the least trending videos.
- We also see a spike in the number of trending videos between August and November compared to other months in the graph. This is because our data range is between August 2020 and November

2023, which biased August, September, October, and November to have unusually more significant lengths than the rest. This assumption is proved in (`Figure 4`) bar plots across four years, where in 2020, the number of bins in the plot is comparatively smaller than that of other bins. 2023 is also smaller than 2021 and 2022, as we need data for December 2023.

- We can also see that the top five categories have the most spread in the stacked (`Figure 3`) barplot, which means that the top five categories are the same for each month. We also noticed a change in trend across 2021 to 2023, where the videos in the gaming category are more trending, while the Entertainment, People & Blogs categories are decreasing year by year. Sports and Music categories stay in almost similar trends.

Visualisation

In [67]:
```
fig2.plot.bar(title='Figure 2 - No. of Videos per Category',xlabel='Category',
ylabel='Number of Trending Videos',color='royalblue')
```

Out[67]:
```
<Axes: title={'center': 'Figure 2 - No. of Videos per Category'}, xlabel='Ca
tegory', ylabel='Number of Trending Videos'>
```

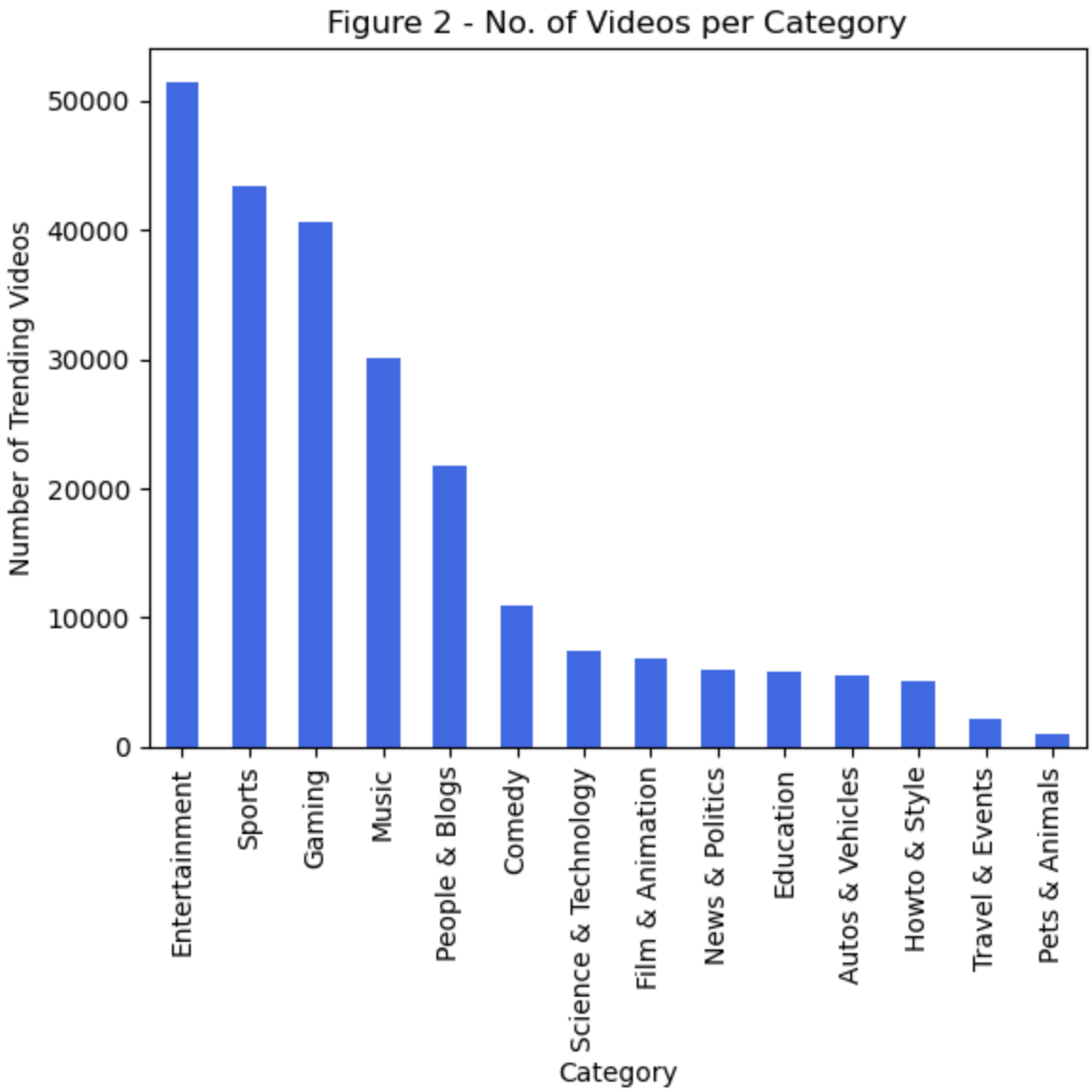


Figure 2 shows the distribution of the trending videos based on video category. It is observed that videos uploaded in the genres of `Entertainment, Sports, Gaming, Music and People & Blogs` *have a greater reception among the youtube userbase.*

In [68]:
```
fig3.plot(kind='bar',stacked=True,figsize=(18,12),title="Figure 3 - No. of Videos per Category by Month",xlabel="Month",ylabel='No. of Trending Videos').legend(bbox_to_anchor=(1.0, 1.0),title="Category")
```

Out[68]: <matplotlib.legend.Legend at 0x1a5bbb646d0>

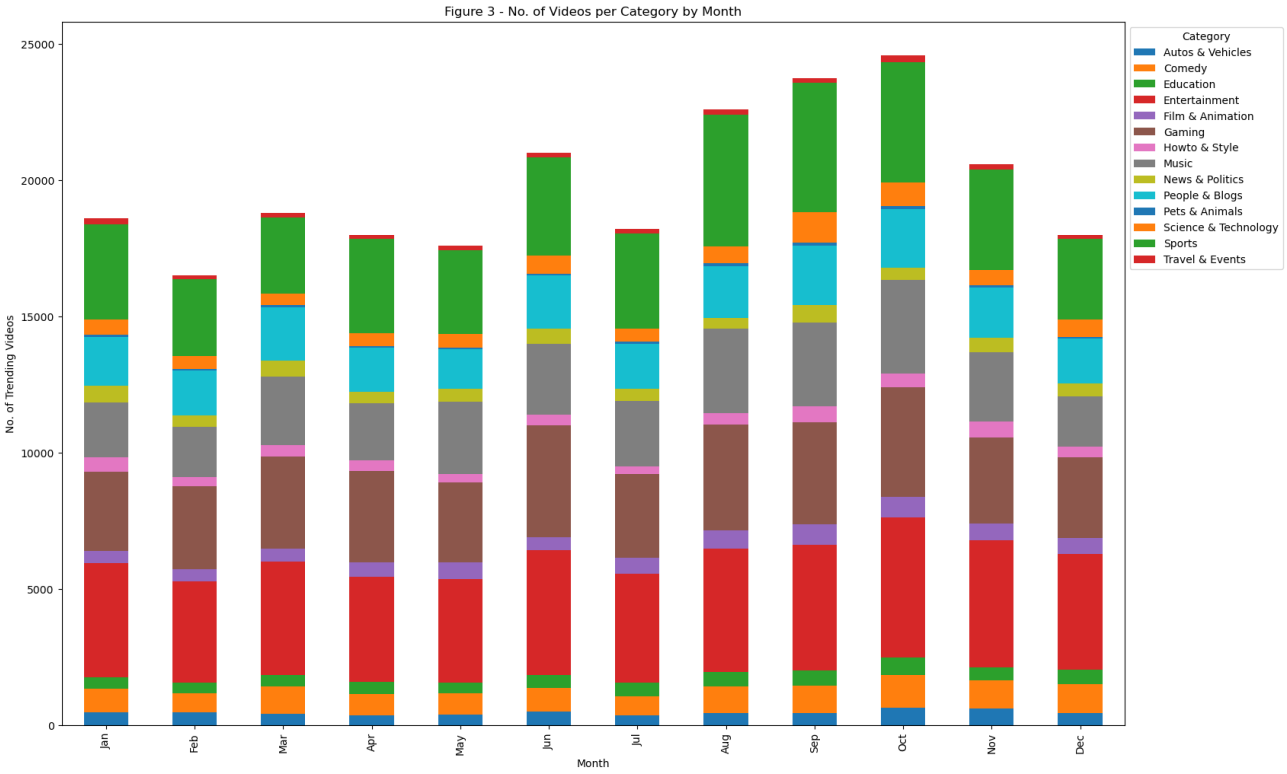


Figure 3 demonstrates the `distribution of the trending videos by categories over the 12 months of the year`. As seen in Figure 2, the top 5 categories is clearly performing better than the other categories throughout the year. We can also conclude that `more videos make it into the trending charts in the months of June to October`.

In [69]:
```
fig4.plot(kind='bar',stacked=False,figsize=(15,10),xlabel="Year",ylabel='No. of Trending Videos',title="Figure 4 - Top 5 Categories per Year").legend(bbox_to_anchor=(1.0, 1.0),title="Category")
```

Out[69]: <matplotlib.legend.Legend at 0x1a5e7236550>

Figure 4 - Top 5 Categories per Year

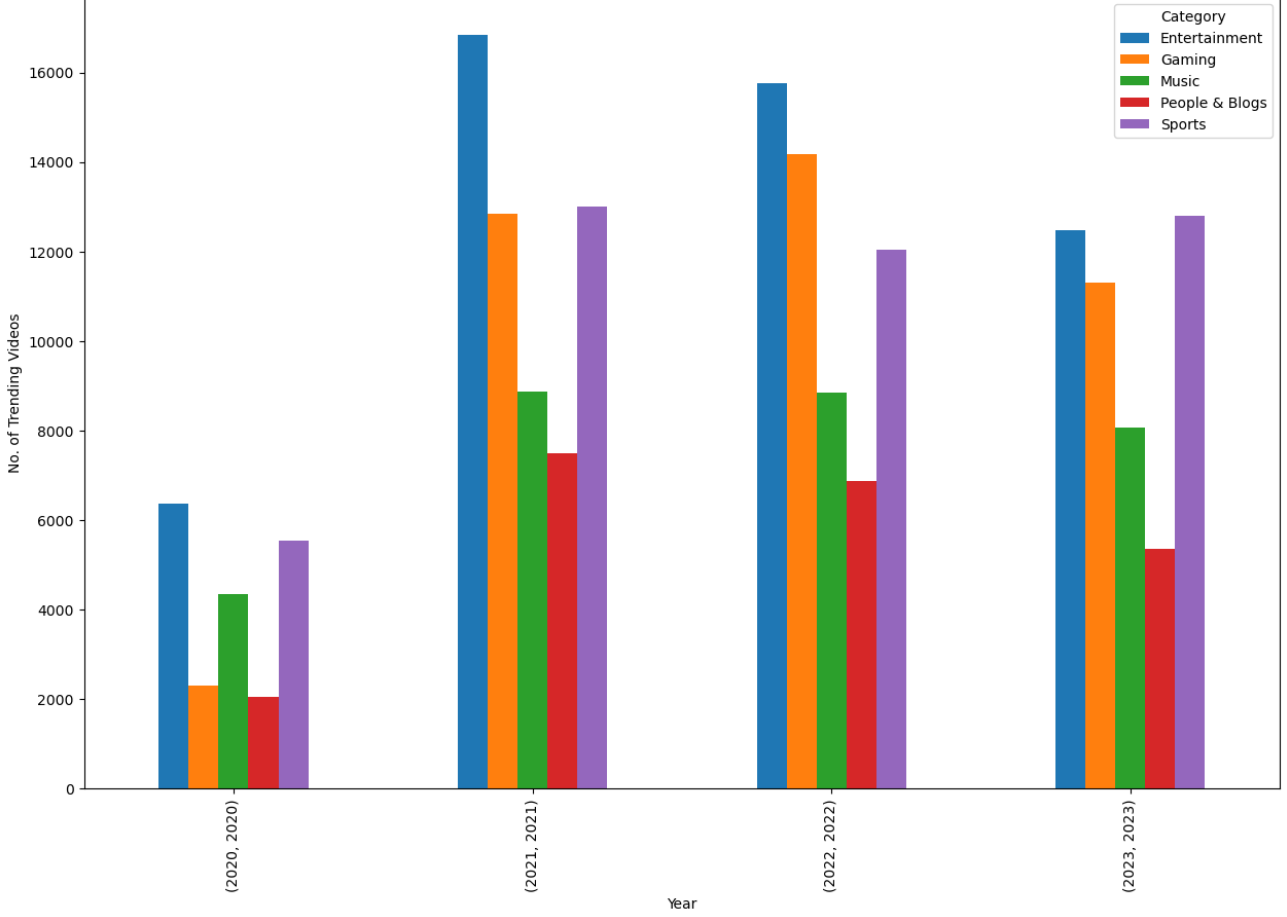


Figure 4 gives us a clearer picture of the distribution of videos based on categories per year. The decreased number of videos in the year 2020 and 2023 is due to the fact that the dataset has values ranging from the second half of 2020 (August) to November of 2023. Moreover, we can say that `Entertainment has always been the category that has been most number of videos in the Trending Charts` *, indicating the Youtubers creating content in the* `Entertainment Genre` *will have a higher chance of making it into the Trending Charts.*

Objective 2

Explanation of Results

We wanted to see if the keywords in the title or tags impact the video trend. So, we used word cloud to pull up the most used keywords in the trending videos. We observed the following outcomes :

- We noticed that, on average, `nine` words are used in the title.
- We can see that the dark and big words in the word cloud are more used keywords and have the highest significance. Thus, in the Entertainment category (in `Figure 6`), keywords like ‘Official’, and ’Trailer’ are used more, while ‘Slow Mo’ is used less in the title. 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 in regards to the keywords used in the title.
- The audience is more intrigued to watch videos with keywords relating to current affairs in almost every category. We verify this by seeing that keywords like 'Boris Johnson' and 'Prime Minister' are used more in the News & Politics category, which suggests that the keywords used from current affairs make the video trend.
- We spot that “Season” is the most commonly used keyword in the title and the video tags in the Entertainment category (from `Figure 8`). The keywords used in the title are also used in the video

tags, with which one could assume that the frequency of keywords used impacts the reach of the videos.

Visualisation

Figure 6 - Wordcloud of Trending Video Titles by Category

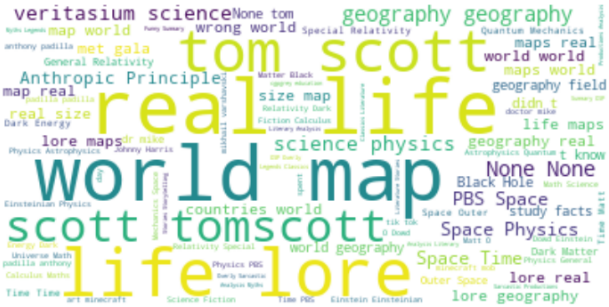
`Figure 6` shows the words that appears the most in each category. Considering the top 5 categories (from Objective 1), we can conclude that these words would increase the chances of the publishers video to make it into the trending list:

- 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.

Figure 7 - Wordcloud of Trending Video tags by Category

Entertainment

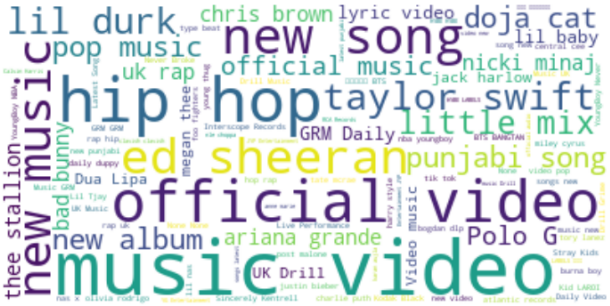
Education



Gaming

People & Blogs

Music

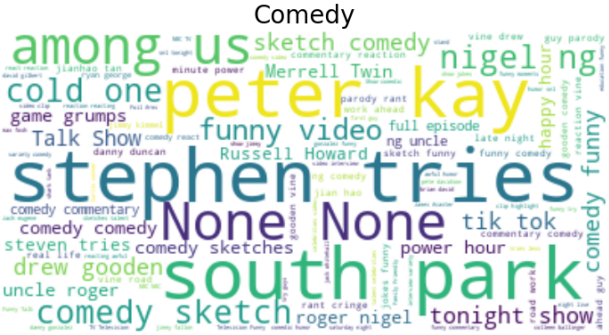


Howto & Style

Sports

News & Politics

Comedy



Film & Animation

Science & Technology

Travel & Events

`Figure 7` 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.

Figure 8 - Wordcloud of Common Words in Trending Video Tags and Title by Category

Autos & Vehicles

Pets & Animals

`Figure 8` 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

Explanation of Results

The aim of objective 3 was to analyse if the number of likes, views, comments or dislikes had an impact on deciding whether an uploaded video makes it into the trending list or not. The following points conclude our findings on this topic.

- We have concluded that dislikes cannot be used as a dependent factor, since Youtube API has discontinued the public display of likes from the year 2022, as shown in `Figure 15`. Thereby, leading us to continue our analysis using the other three variables.
- From `Figure 8`, it is evident that the number of likes, comments and views are almost synonymous, i.e., performing data analysis on these three factors would produce almost similar results. This statement is backed by `Figures 9` and `10`. Furthermore, the number of views were used for further analysis.
- `Figures 12`, `13` and `14`; give us insights on the number of views a video needs to accumulate in order to enter into the Youtube Trending Charts. We can conclude that if a video successfully acquires views between the 25th percentile (358566.0 views) and 50th percentile (781394.0 views), it increases the probability for the video to trend among the userbase.
- `Figure 16` shows the results of a heatmap that lets us know that likes, views,comments,dislikes, year and hour some of the positive significant factors for a video to trend. Using these factors, a lineplot and boxplot were plotted (as seen in `Figure 17` and `18`) which declared that most of the trending videos were uploaded in the months of June to September, in the early hours of 4AM to 6AM. This analysis could state the best possible times to upload a video in order to generate the maximum number of views,likes and comments (user interaction).

Thus, the user interaction in a video does play a crucial role in deciding whether a video makes it into the trending charts or not, so the higher the user interaction, more is the probability to trend. The factors that increase the odds of user interaction have also been found and analysed.

Visualisation

In [72]:
```
fig16.plot(kind='bar', figsize=(15,10), title="Figure 16 - Yearly Dislikes", xlabel='Year', ylabel='Sum of Dislikes', color='royalblue')
```

Out[72]:
```
<Axes: title={'center': 'Figure 16 - Yearly Dislikes'}, xlabel='Year', ylabel='Sum of Dislikes'>
```

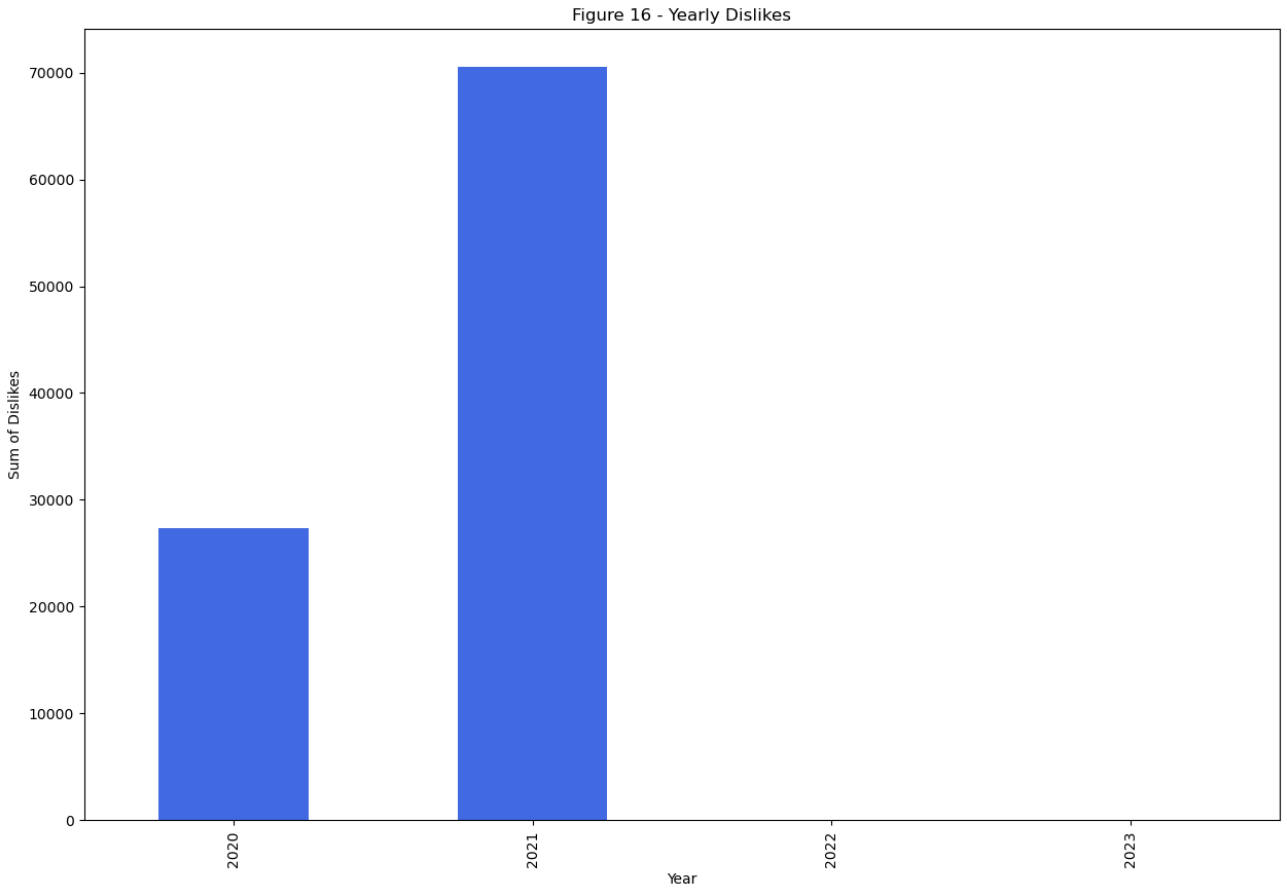


Figure 16 shows us that the `dislikes count has remained zero from 2022`, since Youtube API has removed the dislike count from being made public. Hence, we decided not to use the dislike count for our analysis.

In [71]:
```
plt.figure(figsize=(15, 15))
plt.suptitle('Figure 9 - Scatterplot of Likes, View Count and Comment Count',fontsize=20,x = .7, y = .95)
plt.tight_layout()
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'Scatterplot of {columns[i].title()} vs {columns[j].title()}', xlabel=columns[i].title(), ylabel=columns[j].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: Run
timeWarning: divide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

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

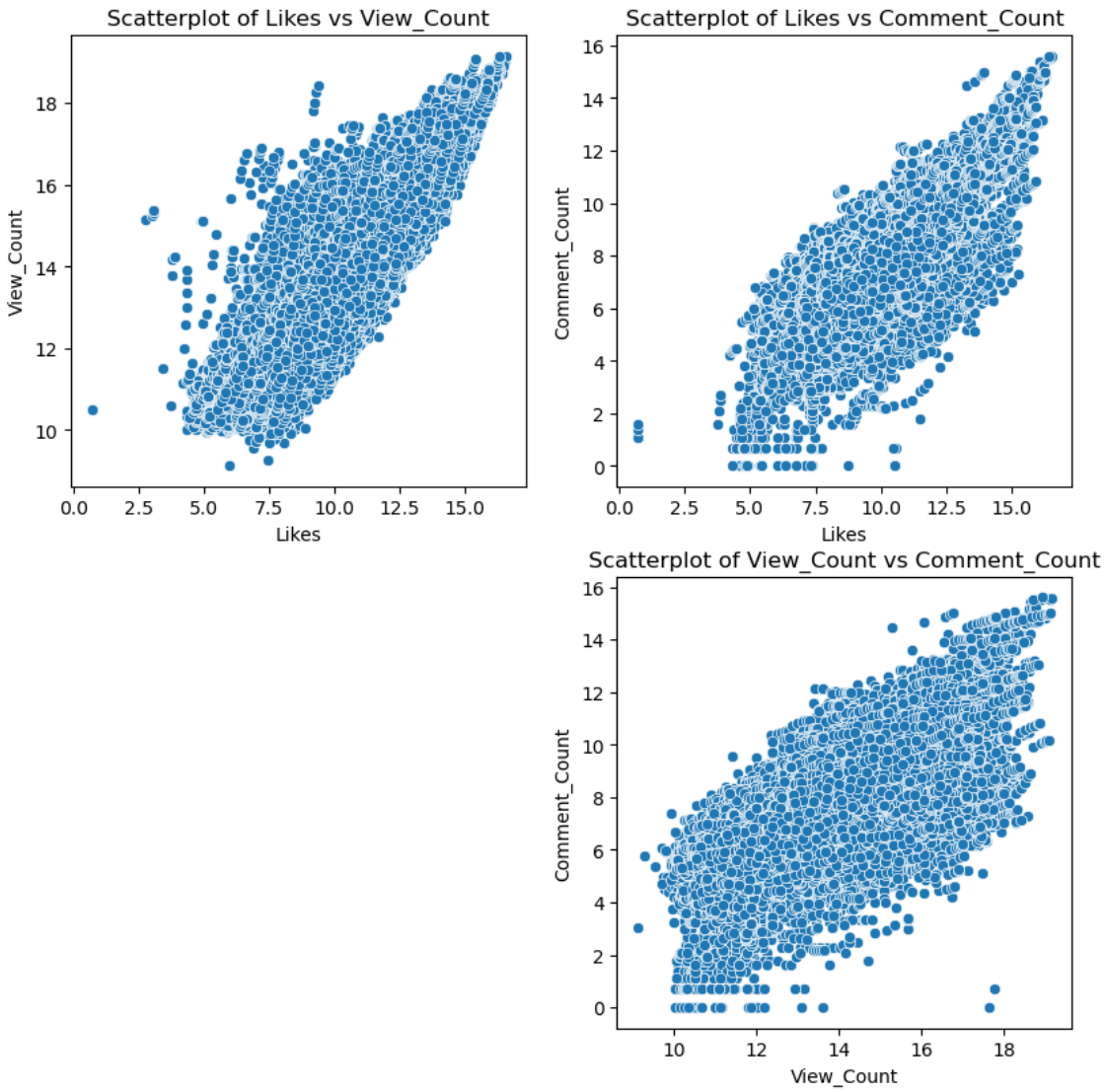
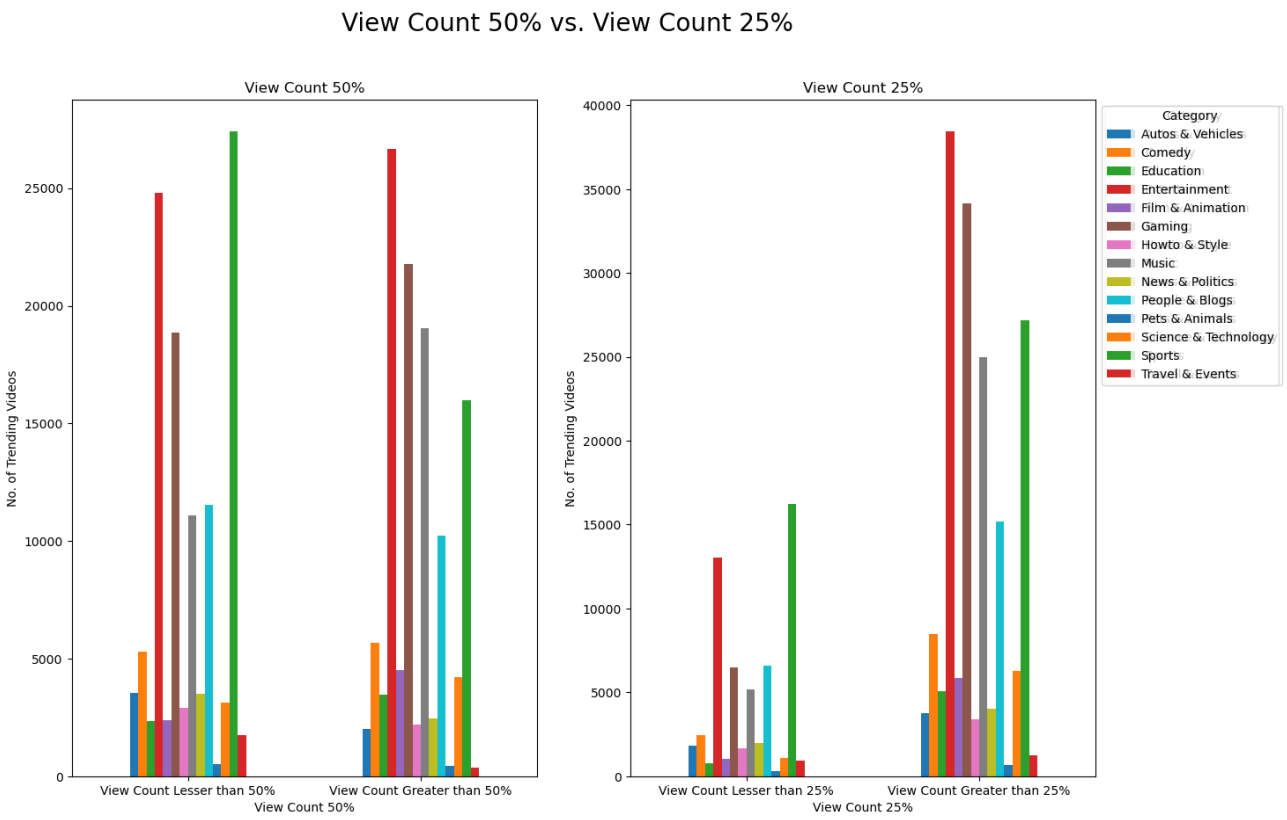


Figure 9 shows that the `number of likes, views and comments are almost synonymous`, so running an analysis using either of these factors would produce similar results.

In [73]:
```
fig, axes = plt.subplots(1, 2, figsize=(15, 5))

fig14.plot(kind='bar',stacked=False,figsize=(15,10),title="Figure 14 - View Co
unt 50% vs. Category",xlabel='View Count 50%',ylabel='No. of Trending Videos',
ax = axes[0]).legend(bbox_to_anchor=(2.615, 1.0),title="Category").axes.set_xt
icklabels(['View Count Lesser than 50%', 'View Count Greater than 50%'],rotati
on=0)
axes[0].set_title('View Count 50%')

fig15.plot(kind='bar',stacked=False,figsize=(15,10),title="Figure 15 - View Co
unt 25% vs. Category",xlabel='View Count 25%',ylabel='No. of Trending Videos',
ax = axes[1]).legend(bbox_to_anchor=(1.0, 1.0),title="Category").axes.set_xtic
klabels(['View Count Lesser than 25%', 'View Count Greater than 25%'],rotation
=0)
axes[1].set_title('View Count 25%')
```

```
fig.suptitle('View Count 50% vs. View Count 25%',fontsize=20)
```

Out[73]: Text(0.5, 0.98, 'View Count 50% vs. View Count 25%')



The above graph shows the comparsion between the number of views of a video with respect to its 25th and 50th quartiles. From this graph we can say that if a video succesfully gets `views greater than the 25th quartile of view count, and somewhere around the 50th quartile of view count`; then the video has a higher chance to become trending.

In [74]:
```
f, ax = plt.subplots(2, 3, figsize=(25, 20))
variables = ['likes', 'view_count', 'comment_count']
x_values = ['hour','month']
plt.suptitle('Figure 19 - Boxplot of Likes, View Count and Comment Count', fon
tsize= 20 ,position=(0.5,0.95))
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]).s
et_title(f'{var.title()} vs {x.title()}')
```

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

  result = getattr(ufunc, method)(*inputs, **kwargs)
c:\Users\asish\anaconda3\Lib\site-packages\pandas\core\arraylike.py:396: Run
```

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

Figure 19 - Boxplot of Likes, View Count and Comment Count

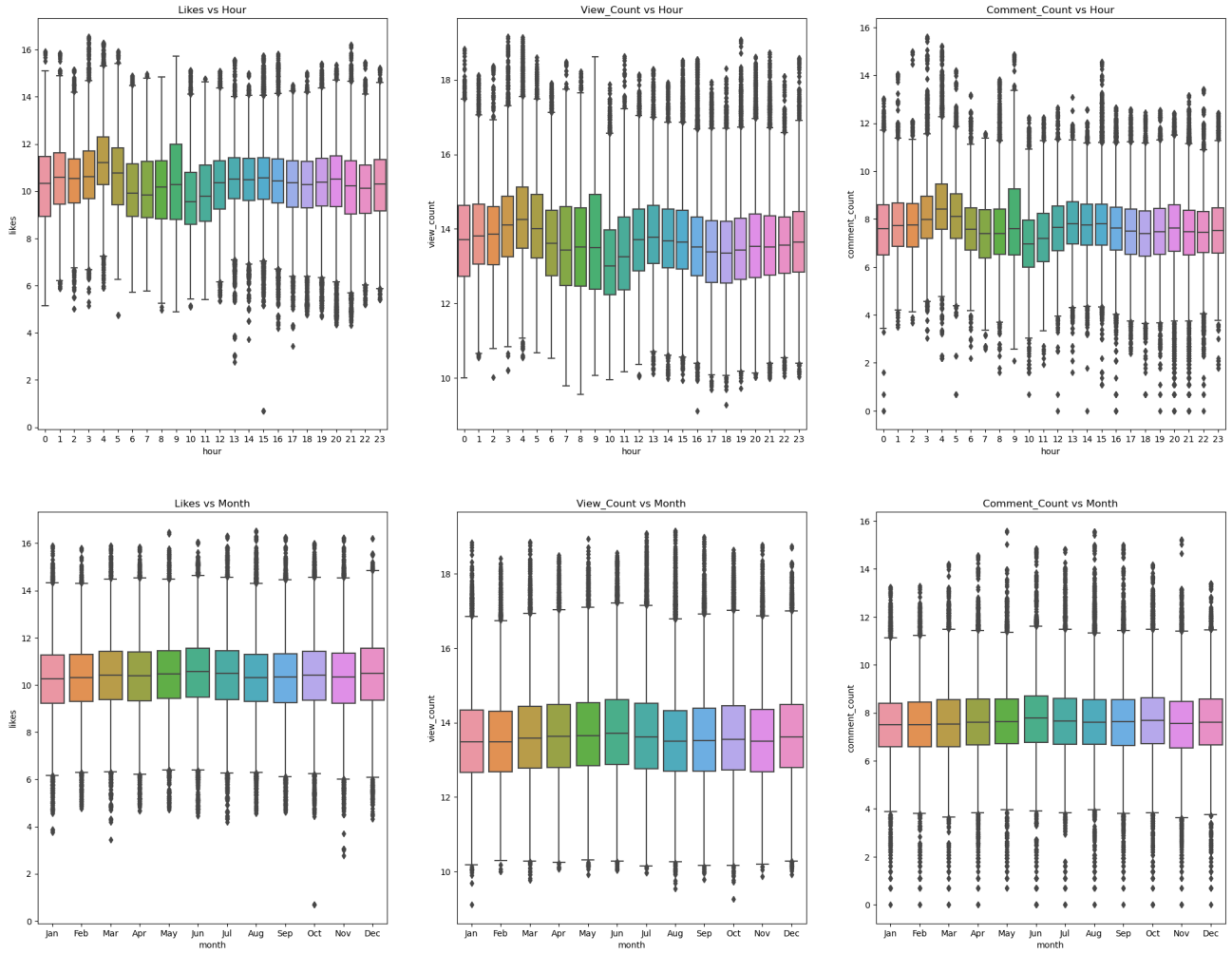


Figure 19 gives us an estimate to the best time to upload videos to generate more views, likes and comments and the graphs state that the `Month of June` *at hour* `5AM to 9AM` *would be the most appropriate times to upload videos to increase the chances of getting into the Trending List*

Objective 4

Explanation of Results

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 22` and `figure 23`. 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

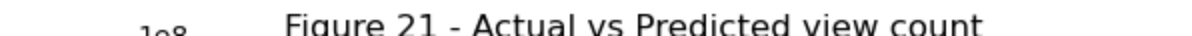
- The MSE score is 1859860906199.00
- The RMSE score is 1363767.17
- The R2 value is 0.95

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

Visualisation

In [75]:
```
plt.scatter(y_test, y_pred1)
plt.xlabel("Actual view count")
plt.ylabel("Predicted view count")
plt.title("Figure 21 - 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_10484\3556402904.py:5: UserWarni
ng: 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")
```



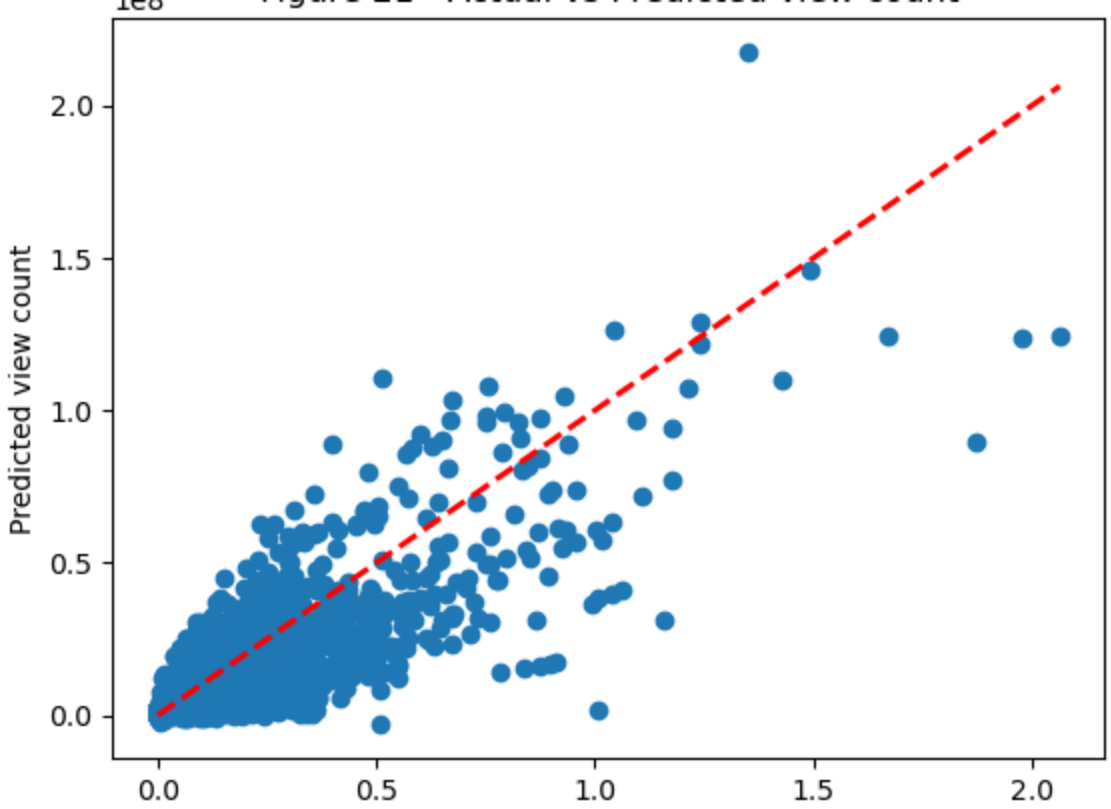


Figure 21 shows the results of the prediction done using `Linear Regression` in comparison with the actual outputs. We can see that the values stray away from the straight line, stating that the Linear Regression model gives us a fairly accurate prediction, but not highly precise. Upon running Evaluation tests, the accuracy was found to be `73%`.

In [76]:
```
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 22 - 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 23 - R2 vs n_estimators')
axes[1].axvline(x=50, color='r', linestyle='--')
plt.show()
```

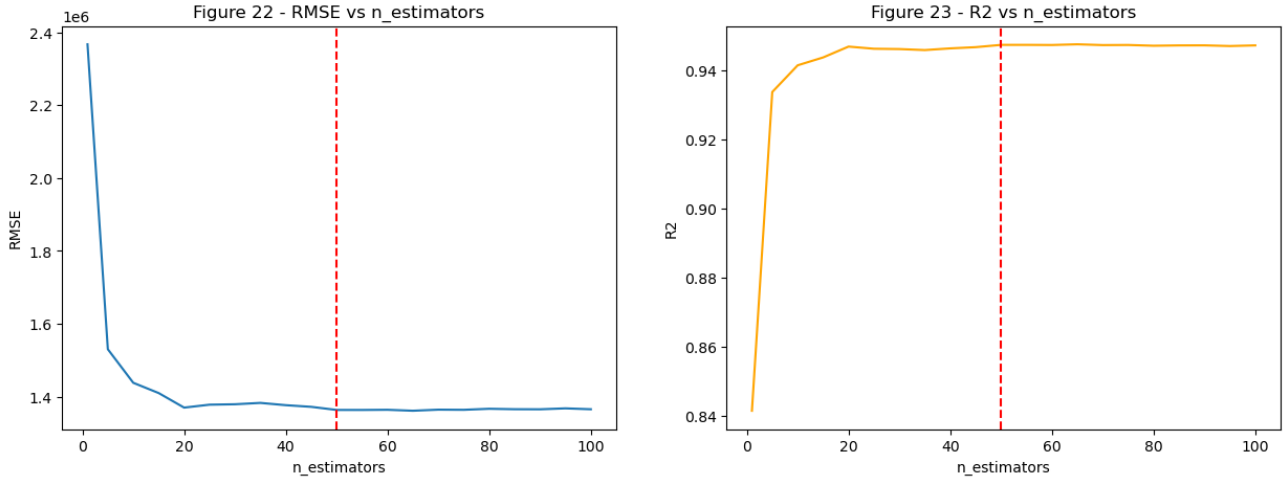


Figure 23 shows us how the estimates varies as the value of n increases. From this graph, we can

conclude that from `n = 50`, the value of R2 remains fairly constant. Thus choosing n as 50 will give us an accurate prediction.

In [77]:
```
plt.scatter(y_test, y_pred)
plt.xlabel("Actual view count")
plt.ylabel("Predicted view count")
plt.title("Figure 24 - 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_10484\1626088688.py:5: UserWarni
ng: 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")
```

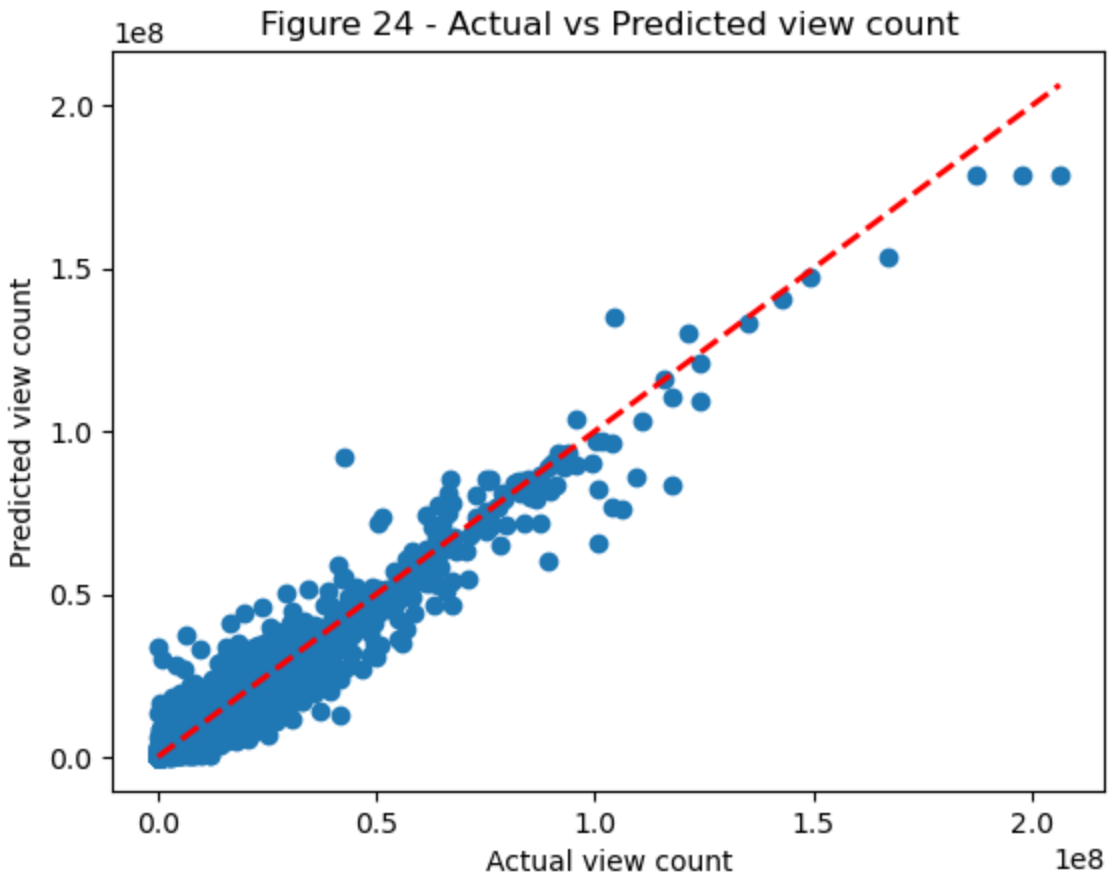


Figure 24 shows the results of the prediction done using `Random Forest Algorithm` in comparison with the actual outputs. We can see that the values stay close to the straight line, stating that the Random Forest model gives us a highly accurate prediction. Upon running Evaluation tests, the accuracy was found to be `95%`.

Conclusion (5 marks)

Achievements

The principal factors regarding the Youtube Trending Charts have been thoroughly analysed and we have concluded that the number of views is the factor that plays a major role in deciding the chances of a video making it into the trending charts, followed by the category of the video and the words used in the title. We

have therefore implemented algorithms to devise a model that predicts the number of views that a video would generate based on these dependent factors, and have been successful in predicting the output with an accuracy score of 95%.

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. ZeroDislikes 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.