

# Exploratory Data Analysing Using Youtube Video Data from Most Popular Gaming Channels

## 1. Aims, objectives and background

### 1.1. Introduction

Founded in 2005, Youtube has grown to become the second largest search engine in the world (behind Google) that processes more than 3 billion searches per month. [1]. It is, however, generally a myth how the Youtube algorithm works, what makes a video get views and be recommended over another. In fact, YouTube has one of the largest scale and most sophisticated industrial recommendation systems in existence [2]. For new content creators, it is a challenge to understand why a video gets video and others do not. There are many "myths" around the success of a Youtube video [3], for example if the video has more likes or comments, or if the video is of a certain duration. It is also worth experimenting and looking for "trends" in the topics that Youtube channels are covering in a certain niche.

In this project will explore the statistics of around 10 most successful Gaming Youtube channel.

### 1.2. Aims and objectives

Within this project, I would like to explore the following:

- Getting to know Youtube API and how to obtain video data.
- Analyzing video data and verify different common "myths" about what makes a video do well on Youtube, for example:
  - Does the number of likes and comments matter for a video to get more views?
  - Does the video duration matter for views and interaction (likes/ comments)?
  - Does title length matter for views?
  - How many tags do good performing videos have? What are the common tags among these videos?
  - Across all the creators I take into consideration, how often do they upload new videos? On which days in the week?
- Explore the trending topics using NLP techniques
  - Which popular topics are being covered in the videos (e.g. using wordcloud for video titles)?

### 1.3. Steps of the project

1. Obtain video meta data via Youtube API for the top 10-15 channels in the gaming niche (this includes several small steps: create a developer key, request data and transform the responses into a usable data format)

2. Preprocess data and engineer additional features for analysis
3. Exploratory data analysis
4. Using clustering model for the video's titles.
5. Conclusions

#### 1.4. Dataset

```
!pip install lib
!pip install isodate
!pip install kneed

Looking in indexes: https://pypi.org/simple, https://us-
python.pkg.dev/colab-wheels/public/simple/
Collecting lib
  Downloading lib-4.0.0-py3-none-any.whl (4.0 kB)
Installing collected packages: lib
Successfully installed lib-4.0.0
Looking in indexes: https://pypi.org/simple, https://us-
python.pkg.dev/colab-wheels/public/simple/
Collecting isodate
  Downloading isodate-0.6.1-py2.py3-none-any.whl (41 kB)
  41.7/41.7 KB 1.5 MB/s eta
0:00:00
Requirement already satisfied: six in /usr/local/lib/python3.8/dist-packages
(from isodate) (1.15.0)
Installing collected packages: isodate
Successfully installed isodate-0.6.1
Looking in indexes: https://pypi.org/simple, https://us-
python.pkg.dev/colab-wheels/public/simple/
Collecting kneed
  Downloading kneed-0.8.2-py3-none-any.whl (10 kB)
Requirement already satisfied: numpy>=1.14.2 in
/usr/local/lib/python3.8/dist-packages (from kneed) (1.21.6)
Requirement already satisfied: scipy>=1.0.0 in
/usr/local/lib/python3.8/dist-packages (from kneed) (1.7.3)
Installing collected packages: kneed
Successfully installed kneed-0.8.2

import pandas as pd
import numpy as np
from dateutil import parser
import isodate

# Data visualization libraries
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
import seaborn as sns
sns.set(style="darkgrid", color_codes=True)
```

```

# Google API
from googleapiclient.discovery import build

# NLP libraries
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
nltk.download('stopwords')
nltk.download('punkt')
from wordcloud import WordCloud

[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data] Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data] Unzipping tokenizers/punkt.zip.

from yellowbrick.cluster import KElbowVisualizer
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from kneed import KneeLocator
import plotly.graph_objects as go
from plotly.subplots import make_subplots

```

## 2. Data creation with Youtube API

I first created a project on Google Developers Console, then requested an authorization credential (API key). Afterwards, I enabled Youtube API for my application, so that I can send API requests to Youtube API services. Then, I went on Youtube and checked the channel ID of each of the channels that I would like to include in my research scope (using their URLs). Then I created the functions for getting the channel statistics via the API.

```

api_key = 'AIzaSyDhvPkboT1Se80yLy0hHz0f8dqKuRHojzQ'

channel_ids = ['UCKqH_9mk1waLgBiL2vT5b9g', # vannossgaming
               'UC7_YxT-KID8kRbqZo7MyscQ', # markiplier
               'UCYzPXprvL5Y-Sf0g4vX-m6g', # jacksepticeye
               'UCS50z6CHmeoF7vSad0qqXfw', # DanTDM
               'UCpGdL9Sn3Q5YUWH2DVUW1Ug', # PopularMMOs
               'UCjtL0fx1yt1NlnFIDyAX3Ug', # W2S
               'UCYVinkwSX7szARULgYpvhLw', # Ali-A
               'UCClNRixXlagwAd--5MwJKCw', # H2ODelirious
               'UC1ieoHqKW-yYgDhLHicx28w', # Syndicate
               ]

youtube = build('youtube', 'v3', developerKey=api_key)

def get_channel_stats(youtube, channel_ids):
    """
    Get channel statistics: title, subscriber count, view count, video
    count, upload playlist
    """

```

*Params:*

*youtube: the build object from googleapiclient.discovery*  
*channels\_ids: list of channel IDs*

*Returns:*

*Dataframe containing the channel statistics for all channels in the provided list: title, subscriber count, view count, video count, upload playlist*

```
"""
all_data = []
request = youtube.channels().list(
    part='snippet,contentDetails,statistics',
    id=', '.join(channel_ids))
response = request.execute()

for i in range(len(response['items'])):
    data = dict(channelName = response['items'][i]['snippet']
['title'],
                subscribers = response['items'][i]['statistics']
['subscriberCount'],
                views = response['items'][i]['statistics']
['viewCount'],
                totalVideos = response['items'][i]['statistics']
['videoCount'],
                playlistId = response['items'][i]
['contentDetails']['relatedPlaylists']['uploads'])
    all_data.append(data)

return pd.DataFrame(all_data)
```

**def** get\_video\_ids(youtube, playlist\_id):

"""

*Get list of video IDs of all videos in the given playlist*

*Params:*

*youtube: the build object from googleapiclient.discovery*  
*playlist\_id: playlist ID of the channel*

*Returns:*

*List of video IDs of all videos in the playlist*

"""

```
request = youtube.playlistItems().list(
    part='contentDetails',
    playlistId = playlist_id,
    maxResults = 50)
response = request.execute()
```

```

video_ids = []

for i in range(len(response['items'])):
    video_ids.append(response['items'][i]['contentDetails']
['videoId'])

next_page_token = response.get('nextPageToken')
more_pages = True

while more_pages:
    if next_page_token is None:
        more_pages = False
    else:
        request = youtube.playlistItems().list(
            part='contentDetails',
            playlistId = playlist_id,
            maxResults = 50,
            pageToken = next_page_token)
        response = request.execute()

        for i in range(len(response['items'])):
            video_ids.append(response['items'][i]
['contentDetails']['videoId'])

        next_page_token = response.get('nextPageToken')

return video_ids

def get_video_details(youtube, video_ids):
    """
    Get video statistics of all videos with given IDs
    Params:

    youtube: the build object from googleapiclient.discovery
    video_ids: list of video IDs

    Returns:
    Dataframe with statistics of videos, i.e.:
        'channelTitle', 'title', 'description', 'tags', 'publishedAt'
        'viewCount', 'likeCount', 'favoriteCount', 'commentCount'
        'duration', 'definition', 'caption'
    """

    all_video_info = []

    for i in range(0, len(video_ids), 50):
        request = youtube.videos().list(
            part="snippet,contentDetails,statistics",
            id=', '.join(video_ids[i:i+50])

```

```

    )
    response = request.execute()

    for video in response['items']:
        stats_to_keep = {'snippet': ['channelTitle', 'title',
'description', 'tags', 'publishedAt'],
                        'statistics': ['viewCount', 'likeCount',
'definition', 'caption']
                        'contentDetails': ['duration',
'description', 'caption']}
        video_info = {}
        video_info['video_id'] = video['id']

        for k in stats_to_keep.keys():
            for v in stats_to_keep[k]:
                try:
                    video_info[v] = video[k][v]
                except:
                    video_info[v] = None

        all_video_info.append(video_info)
    return pd.DataFrame(all_video_info)

def get_comments_in_videos(youtube, video_ids):
    """
    Get top level comments as text from all videos with given IDs
    (only the first 10 comments due to quote limit of Youtube API)
    Params:

    youtube: the build object from googleapiclient.discovery
    video_ids: list of video IDs

    Returns:
    Dataframe with video IDs and associated top level comment in text.
    """
    all_comments = []

    for video_id in video_ids:
        try:
            request = youtube.commentThreads().list(
                part="snippet,replies",
                videoId=video_id
            )
            response = request.execute()

            comments_in_video = [comment['snippet']['topLevelComment']
['snippet']['textOriginal'] for comment in response['items'][0:10]]
            comments_in_video_info = {'video_id': video_id,

```

```

'comments': comments_in_video}

    all_comments.append(comments_in_video_info)

    except:
        # When error occurs - most likely because comments are
disabled on a video
        print('Could not get comments for video ' + video_id)

    return pd.DataFrame(all_comments)

```

### Get channel statistics

Using the `get_channel_stats` function defined below, now we are going to obtain the channel statistics for the 9 channels in scope.

```
channel_data = get_channel_stats(youtube, channel_ids)
```

Now I can print out the data and take a look at the channel statistics overview.

```
channel_data
```

	channelName	subscribers	views	totalVideos	\
0	DanTDM	26300000	19005285299	3639	
1	Markiplier	34400000	19595365073	5410	
2	Syndicate	9690000	2126364229	3526	
3	jacksepticeye	29000000	15887438206	5052	
4	H2ODelirious	13300000	4295816083	3331	
5	W2S	16300000	4745348192	652	
6	VanossGaming	25800000	15266099400	1678	
7	Ali-A	18200000	5895310216	4026	
8	PopularMMOs	17200000	14530283729	4685	

	playlistId
0	UUS50z6CHmeoF7vSad0qqXfw
1	UU7_YxT-KID8kRbqZo7MyscQ
2	UU1ieoHqKW-yYgDhLHicx28w
3	UUYzPXprv15Y-Sf0g4vX-m6g
4	UUClnRixXlagwAd--5MwJKCw
5	UUjtL0fx1yt1NlnFIDyAX3Ug
6	UUKqH_9mk1waLgBiL2vT5b9g
7	UUYVinkwSX7szARULgYpvhLw
8	UUpGdL9Sn3Q5YWUH2DVUW1Ug

I noticed the count columns in `channel_data` is currently in string format, so I will convert them into numeric so that we can visualize and do numeric operations on them.

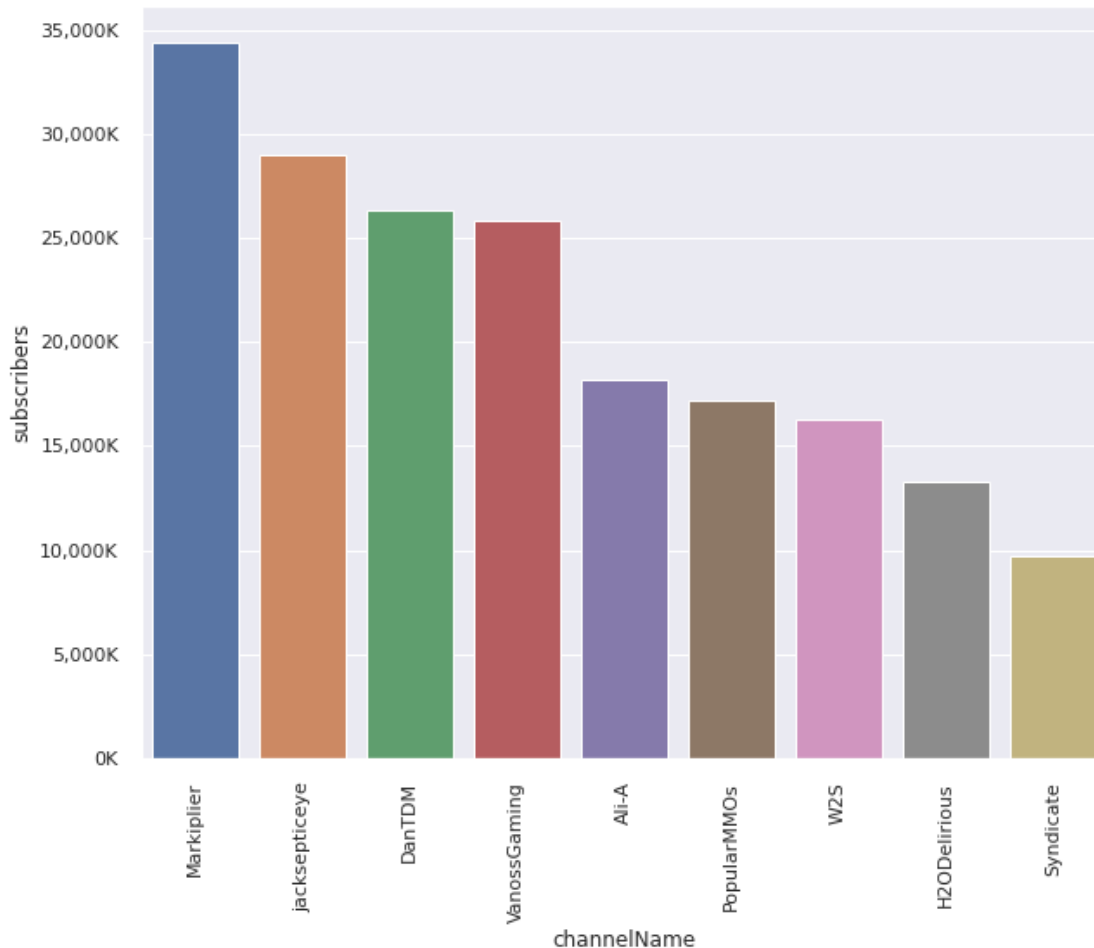
```

# Convert count columns to numeric columns
numeric_cols = ['subscribers', 'views', 'totalVideos']
channel_data[numeric_cols] =
channel_data[numeric_cols].apply(pd.to_numeric, errors='coerce')

```

Let's take a look at the number of subscribers per channel to have a view of how popular the channels are when compared with one another.

```
sns.set(rc={'figure.figsize':(10,8)})
ax = sns.barplot(x='channelName', y='subscribers',
data=channel_data.sort_values('subscribers', ascending=False))
ax.yaxis.set_major_formatter(ticker.FuncFormatter(lambda x, pos:
'{:, .0f}'.format(x/1000) + 'K'))
plot = ax.set_xticklabels(ax.get_xticklabels(),rotation = 90)
```

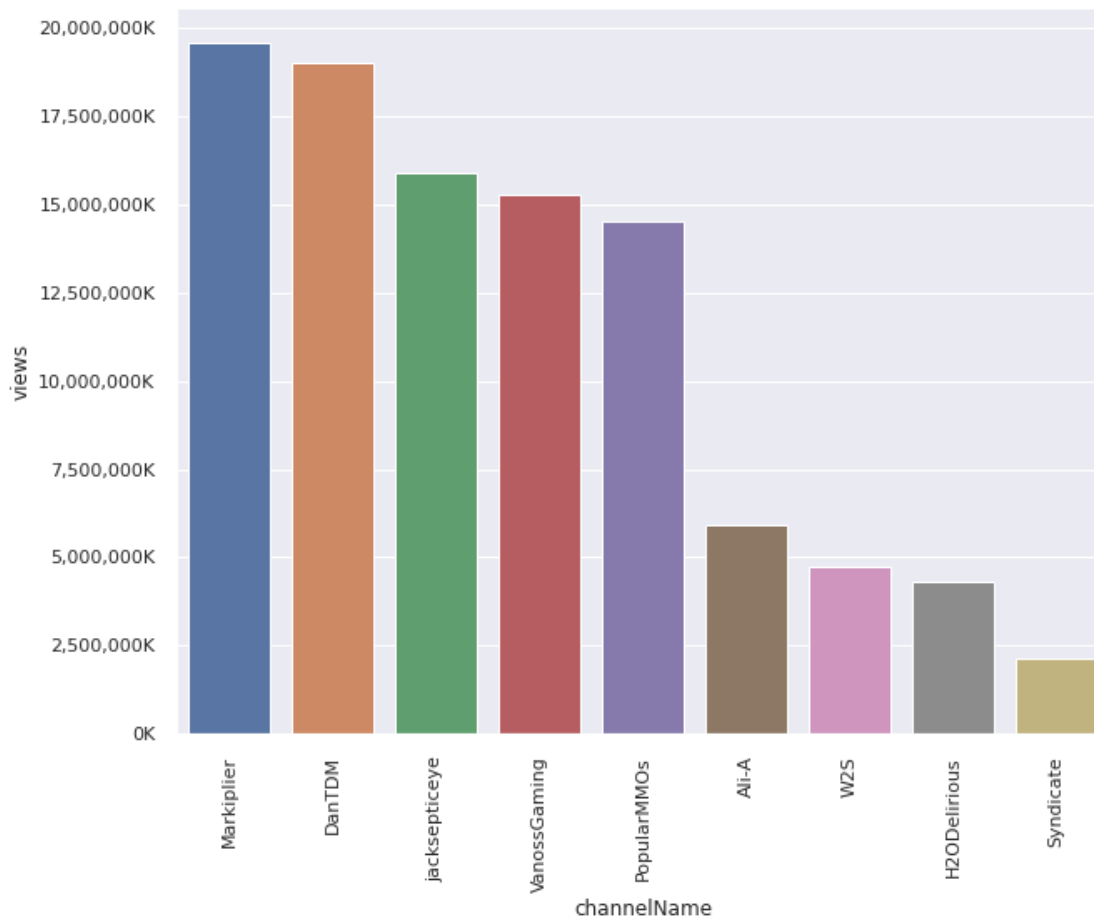


Next, we will look at the rank considering the total number of views of the channels. The rank is fairly similar to the subscriber count rank. Markiplier, jacksepticeye and DanTDM remain the three most popular channels considering both subscribers and views. Interestingly, some channels have more subscribers but less views and vice versa. For example, PopularMMOs channel's has equal number of subscribers than Ali-A Barousse channel, but significantly more views in total.

```
ax = sns.barplot(x='channelName', y='views',
data=channel_data.sort_values('views', ascending=False))
ax.yaxis.set_major_formatter(ticker.FuncFormatter(lambda x, pos:
```



```
'{:,.0f}').format(x/1000) + 'K'))
plot = ax.set_xticklabels(ax.get_xticklabels(),rotation = 90)
```



### Get video statistics for all the channels

In the next step, we will obtain the video statistics for all the channels. In total, we obtained 31751 videos as seen in below.

*# Create a dataframe with video statistics and comments from all channels*

```
video_df = pd.DataFrame()
comments_df = pd.DataFrame()

for c in channel_data['channelName'].unique():
    print("Getting video information from channel: " + c)
    playlist_id = channel_data.loc[channel_data['channelName']== c,
    'playlistId'].iloc[0]
    video_ids = get_video_ids(youtube, playlist_id)

    # get video data
    video_data = get_video_details(youtube, video_ids)
```

```

# get comment data
'''comments_data = get_comments_in_videos(youtube, video_ids)'''

# append video data together and comment data together
video_df = video_df.append(video_data, ignore_index=True)
'''comments_df = comments_df.append(comments_data,
ignore_index=True)'''

```

```

Getting video information from channel: DanTDM
Getting video information from channel: Markiplier
Getting video information from channel: Syndicate
Getting video information from channel: jacksepticeye
Getting video information from channel: H20Delirious
Getting video information from channel: W2S
Getting video information from channel: VanossGaming
Getting video information from channel: Ali-A
Getting video information from channel: PopularMMOs

```

```
len(video_df)
```

```
31905
```

```

# Write video data to CSV file for future references
video_df.to_csv('video_data.csv')

```

## Preprocessing & Feature engineering

To be able to make use of the data for analysis, we need to perform a few pre-processing steps. Firstly, we would like reformat some columns, especially the date and time columns such as "pushlishedAt" and "duration". In addition, we also think it is necessary to enrich the data with some new features that might be useful for understanding the videos' characteristics.

### Check for empty values

```
video_df.isnull().any()
```

```

video_id          False
channelTitle      False
title             False
description       False
tags              True
publishedAt       False
viewCount         True
likeCount         True
favouriteCount    True
commentCount      True
duration          False
definition        False
caption           False
dtype: bool

```

There's no strange dates in the publish date column, videos were published between 2012 and 2022.

```
video_df.publishedAt.sort_values().value_counts()
```

```
2012-11-17T18:18:46Z    15
2012-05-06T22:57:12Z     8
2015-09-07T19:00:01Z     3
2016-07-03T19:00:01Z     3
2015-05-15T19:00:01Z     3
..
2014-09-19T02:25:58Z     1
2014-09-19T01:59:29Z     1
2014-09-19T00:51:08Z     1
2014-09-18T21:24:07Z     1
2023-02-08T01:37:15Z     1
Name: publishedAt, Length: 31665, dtype: int64
```

Next, we need to check if the data type of the columns are correct. I have checked the data types and indeed some count columns such as view count and comment count are currently not in correct data type. In this step, we convert these count columns into integer.

```
cols = ['viewCount', 'likeCount', 'commentCount']
video_df[cols] = video_df[cols].apply(pd.to_numeric, errors='coerce',
axis=1)
```

### Enriching data

I want to enrich the data for further analyses, for example:

- create published date column with another column showing the day in the week the video was published, which will be useful for later analysis.
- convert video duration to seconds instead of the current default string format
- calculate number of tags for each video
- calculate comments and likes per 1000 view ratio
- calculate title character length

```
# Create publish day (in the week) column
video_df['publishedAt'] = video_df['publishedAt'].apply(lambda x:
parser.parse(x))
video_df['publishDayName'] = video_df['publishedAt'].apply(lambda x:
x.strftime("%A"))
```

```
# convert duration to seconds
video_df['durationSecs'] = video_df['duration'].apply(lambda x:
isodate.parse_duration(x))
video_df['durationSecs'] =
video_df['durationSecs'].astype('timedelta64[s]')
```

```
# Add number of tags
video_df['tagsCount'] = video_df['tags'].apply(lambda x: 0 if x is
None else len(x))

# Comments and likes per 1000 view ratio
video_df['likeRatio'] = video_df['likeCount']/ video_df['viewCount'] *
1000
video_df['commentRatio'] = video_df['commentCount']/
video_df['viewCount'] * 1000

# Title character length
video_df['titleLength'] = video_df['title'].apply(lambda x: len(x))
```

Let's look at the video dataset at this point to see if everything went well. It looks good - now we will proceed to exploratory analysis part.

```
video_df.head()
```

	video_id	channelTitle	title	\
0	RtbxwxdHlPw	Markiplier	Dead Space: REMAKE - Part 6	
1	zaTi7YjDyDw	Markiplier	The Glitched Attraction - Part 7	
2	c7PC-Qh7Qq4	Markiplier	Dead Space: REMAKE - Part 5	
3	7T06GbaDFVI	Markiplier	3 SCARY GAMES #98	
4	9lIHJh0tJU	Markiplier	Dead Space: REMAKE - Part 4	

	description	\
0	Dead Space Remake continues with Chapter 6 and...	
1	We find the secret tape! What's Vanny doing he...	
2	Dead Space Remake continues with Chapter 5 and...	
3	There's 64 Harvest Festivals going on at A Mal...	
4	Plenty of necromorphs to STOMP in Chapter 4 of...	

	tags	\
0	[dead space, markiplier, dead space remake, ga...	
1	[the glitched attraction, five nights at fredd...	
2	[dead space, markiplier, dead space remake, ga...	
3	[markiplier, 3 scary games, indie horror, harv...	
4	[dead space, markiplier, dead space remake, ga...	

	publishedAt	viewCount	likeCount	favouriteCount	\
0	2023-02-07 18:45:41+00:00	157056.0	13584.0	None	
1	2023-02-06 20:11:36+00:00	1230129.0	117222.0	None	
2	2023-02-05 17:33:14+00:00	508017.0	28651.0	None	
3	2023-02-04 19:17:31+00:00	2153500.0	114763.0	None	
4	2023-02-03 19:42:06+00:00	643717.0	31150.0	None	

	commentCount	duration	definition	caption	pushblishDayName
0	1446.0	PT1H4M46S	hd	false	Tuesday
1	6058.0	PT15M48S	hd	false	Monday

948.0					
2	1915.0	PT42M14S	hd	false	Sunday
2534.0					
3	5763.0	PT44M49S	hd	false	Saturday
2689.0					
4	2027.0	PT46M25S	hd	false	Friday
2785.0					

	tagsCount	likeRatio	commentRatio	titleLength
0	10	86.491443	9.206907	27
1	7	95.292445	4.924687	32
2	10	56.397719	3.769559	27
3	7	53.291386	2.676109	17
4	10	48.390830	3.148899	27

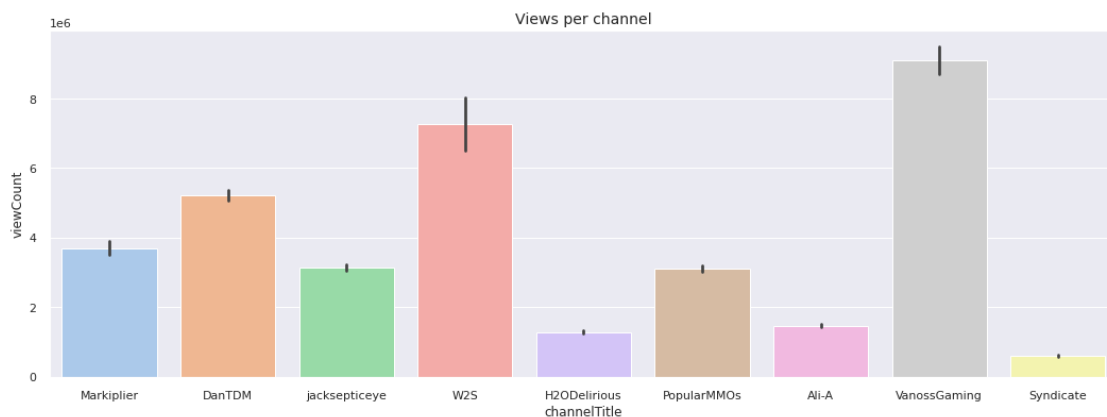
## Exploratory analysis

### Views distribution per channel

With the video statistics for all channel, now we can see how the views are distributed per channel. Some channels might have a lot of views on one of their videos and the rest do not receive many views. Other channels might have more evenly distribution views per video.

```
plt.rcParams['figure.figsize'] = (18, 6)
sns.barplot(video_df['channelTitle'], video_df['viewCount'], palette =
'pastel')
plt.title('Views per channel', fontsize = 14)
plt.show()
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/_decorators.py:36:
FutureWarning: Pass the following variables as keyword args: x, y.
From version 0.12, the only valid positional argument will be `data`,
and passing other arguments without an explicit keyword will result in
an error or misinterpretation.
warnings.warn(
```

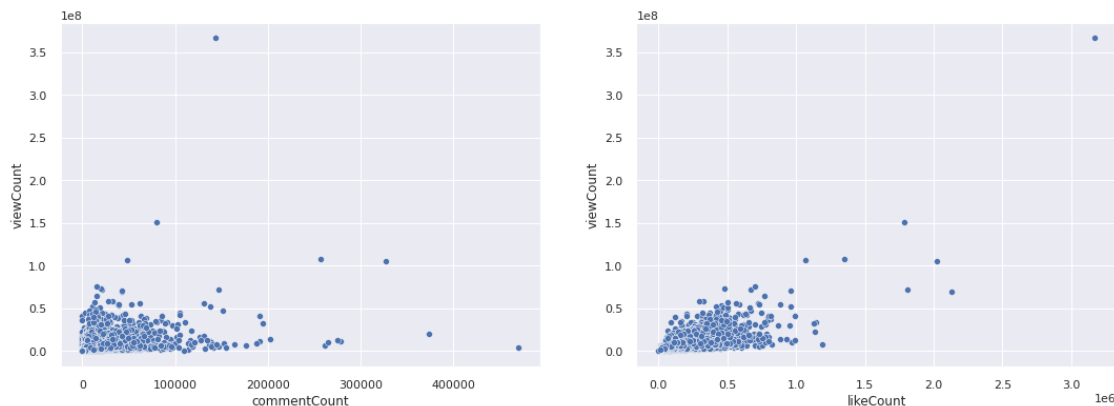


## Does the number of likes and comments matter for a video to get more views?

Firstly, I would like to check if comments and likes do correlate with how many views a video would get. In the plots below, it can be observed that the number of views and number of comments/ likes strongly correlated with each other. The number of likes seems to suggest stronger correlation than the number of comments. However, this is expected as the more people watching a video, the more likely this video will get comments and likes. To correct for this factor, we will plot these relationships again using the comments per 1000 view and likes per 1000 view ratios.

```
fig, ax =plt.subplots(1,2)
sns.scatterplot(data = video_df, x = "commentCount", y = "viewCount",
ax=ax[0])
sns.scatterplot(data = video_df, x = "likeCount", y = "viewCount",
ax=ax[1])
```

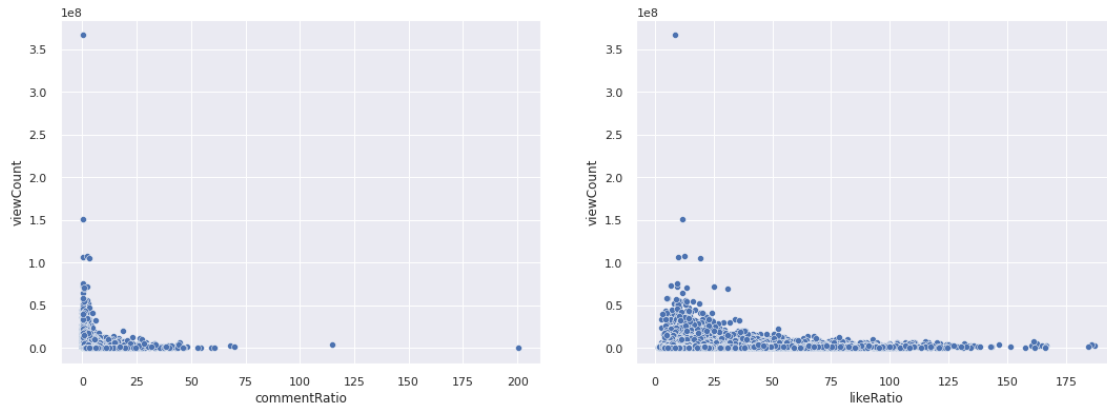
<matplotlib.axes.\_subplots.AxesSubplot at 0x7f7e0db48670>



Now we will take a look at the correlation if we look at the comment ratio and like ratio instead of the absolute number.

```
fig, ax =plt.subplots(1,2)
sns.scatterplot(data = video_df, x = "commentRatio", y = "viewCount",
ax=ax[0])
sns.scatterplot(data = video_df[video_df.likeRatio<10000], x =
"likeRatio", y = "viewCount", ax=ax[1])
```

<matplotlib.axes.\_subplots.AxesSubplot at 0x7f7e07561d30>

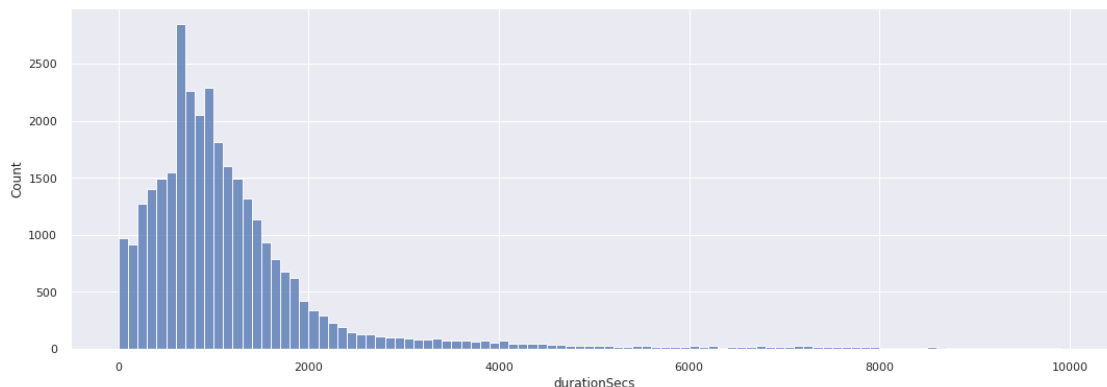


*Does the video duration matter for views and interaction (likes/ comments)?*

As can be seen in the histogram below, most videos are between 300 to 1200 seconds, which is about 5 to 20 minutes. Here I have to limit the duration to 10,000 because of some really long videos (potentially streaming videos).

```
sns.histplot(data=video_df[video_df['durationSecs'] < 10000],
x="durationSecs", bins=50,binwidth=100)
```

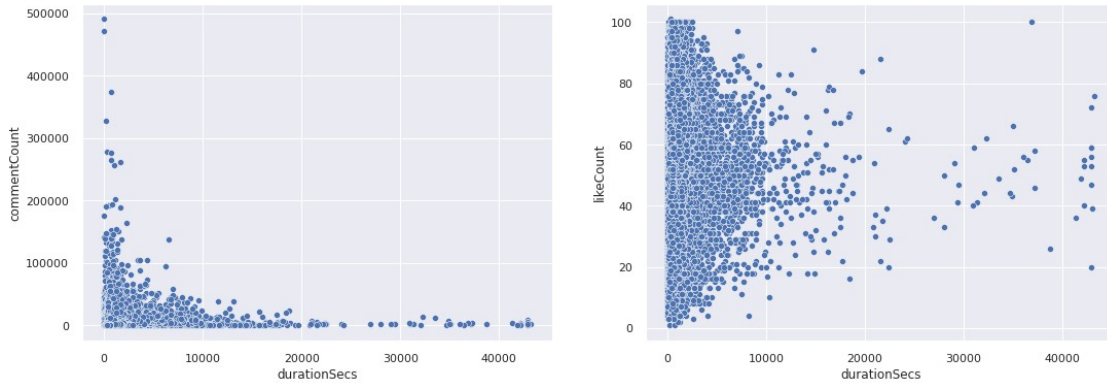
<matplotlib.axes.\_subplots.AxesSubplot at 0x7fcc9a4973d0>



Now we plot the duration against comment count and like count. It can be seen that actually shorter videos tend to get more likes and comments than very long videos.

```
fig, ax =plt.subplots(1,2)
sns.scatterplot(data = video_df, x = "durationSecs", y =
"commentCount", ax=ax[0])
sns.scatterplot(data = video_df, x = "durationSecs", y = "likeCount",
ax=ax[1])
```

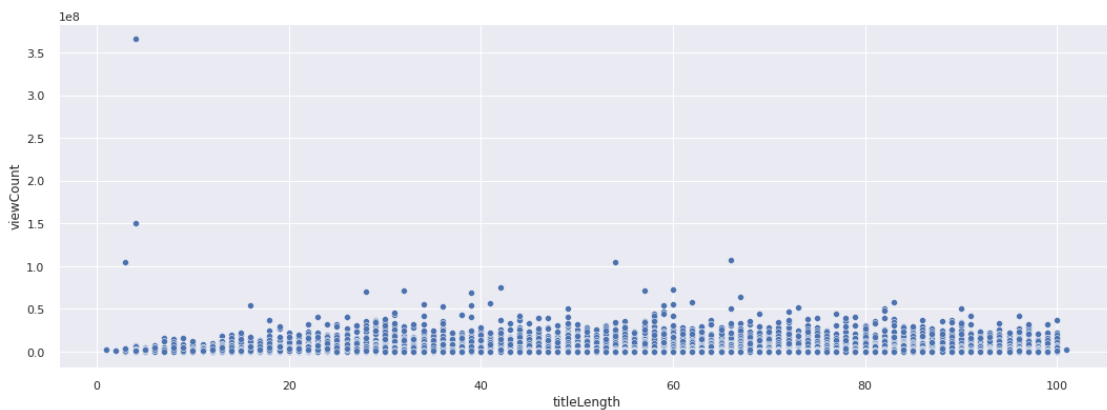
<matplotlib.axes.\_subplots.AxesSubplot at 0x7fcc9ad87250>



## Does title length matter for views?

There is no clear relationship between title length and views as seen the scatterplot below, but most-viewed videos tend to have average title length of 30-70 characters.

```
sns.scatterplot(data = video_df, x = "titleLength", y = "viewCount")
<matplotlib.axes._subplots.AxesSubplot at 0x7fcca7341c70>
```



## Wordcloud for words in title

As I'm interested to see what the creators are making videos about and which terms most frequently appear in their video titles, I will create a wordcloud for the most common words. We first need to remove the stopwords such as "you", "I", "the", etc. which do not contribute a lot to the meaning of the title.

```
def plot_cloud(wordcloud, title='none'):
    plt.figure(figsize=(10, 8))
    plt.imshow(wordcloud)
    if (title != 'none'):
        plt.title(title)
    plt.axis("off");
```

```
stop_words = set(stopwords.words('english'))
video_df['title_no_stopwords'] = video_df['title'].apply(lambda x:
[item for item in str(x).split() if item not in stop_words])
```



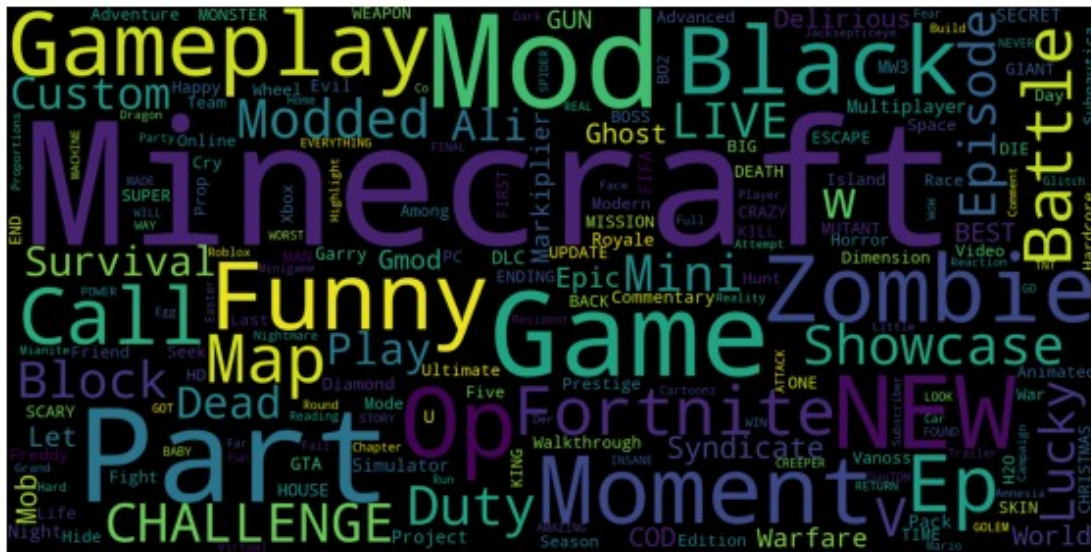
```

all_words = list([a for b in video_df['title_no_stopwords'].tolist()
for a in b])
all_words_str = ' '.join(all_words)

def plot_cloud(wordcloud,title='none'):
    plt.figure(figsize=(10, 8))
    plt.imshow(wordcloud)
    if (title != 'none'):
        plt.title(title)
    plt.axis("off");

wordcloud = WordCloud(width = 2000, height = 1000, random_state=1,
background_color='black',
                        colormap='viridis',
collocations=False).generate(all_words_str)
plot_cloud(wordcloud)

```



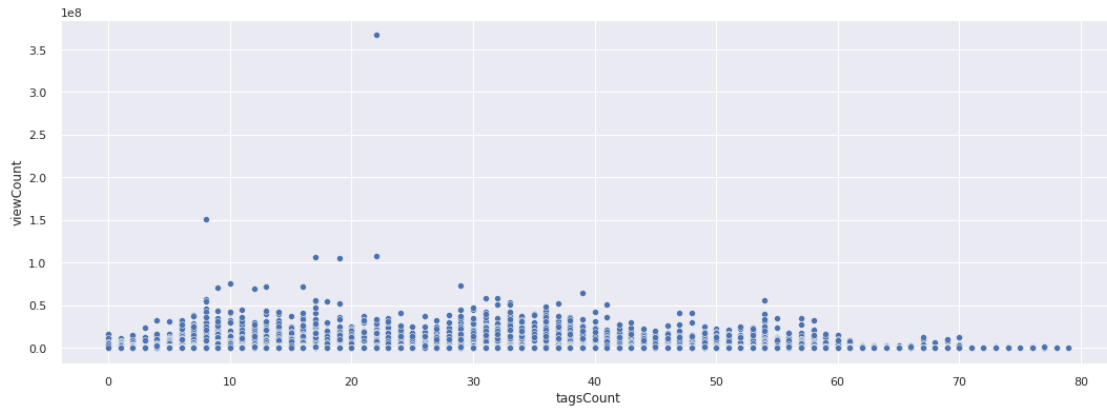
It can be seen that most common words are Minecraft, Game, Cakk, Funny, Zombie...

### Number of tags vs views

Most-viewed videos seems to have 0-35 tags.

```
sns.scatterplot(data = video_df, x = "tagsCount", y = "viewCount")
```

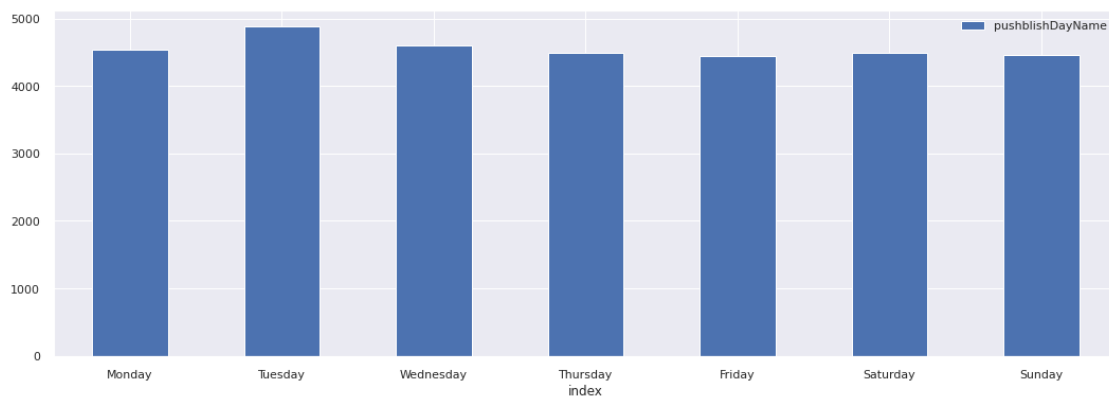
<matplotlib.axes.\_subplots.AxesSubplot at 0x7f65cc733c10>



### Which day in the week are most videos uploaded?

Tuesday seems to be the highest published day, but overall videos are uploaded equally for everyday.

```
day_df = pd.DataFrame(video_df['pushblishDayName'].value_counts())
weekdays = [ 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',
'Saturday', 'Sunday']
day_df = day_df.reindex(weekdays)
ax = day_df.reset_index().plot.bar(x='index', y='pushblishDayName',
rot=0)
```



## Conclusions

In this project, we have explored the video data of the 10 most popular gaming channels and we obtain insight:

- Higher views seems to be higher comments and likes though it is not guarantee. Likes seem to be a better indicator for interaction than comments and the number of likes seem to follow the "social proof".
- Most videos have between 0 and 35 tags.

- Most-viewed videos tend to have average title length of 30-70 characters. Too short or too long titles seem to harm viewership.
- Videos are usually uploaded equally everyday.
- After clustering, we find out there are several trends of gaming videos. Minecraft and fortnite seems to be the most popular games and have many extension : 'mod', 'gmod', 'showcase', 'battle'. Shooting games channel like 'Syndicate', 'Ali-A' achieve a really stable views. Moreover, people seems to like watching role-playing game videos or horror games like 'freedy'. Sometimes, videos go with series ( part 1, part 2).

### **Project limitation:**

The findings should also be taken with a grain of salt for a number of reasons:

- Even with over 30000 videos, the clustering part seems to be unstable and vague.
- We did not utilize the comments parts, which may lead us to a better insight.
- There are many other factors that haven't been taken into the analysis, including the marketing strategy of the creators and many random effects that would affect how successful a video is.