Trending US YouTube Videos Analysis

Tran Khoi Nguyen
20225453
nguyen.tk225453@sis.hust.edu.vn

Tran Thanh Vinh
20225539
vinh.tt225539@sis.hust.edu.vn

Vu Hoang Nhat Anh
20225471
anh.vhn225471@sis.hust.edu.vn

Tran Nam Tuan Vuong
20225540
vuong.tnt225540@sis.hust.edu.vn

Tran Nhat Minh
20225511
minh.tn225511@sis.hust.edu.vn

***Abstract*— YouTube is one of the most, in fact, it's the most popular video streaming platforms nowadays, with billions of users worldwide. As a major hub for social interaction, YouTube offers an extensive range of contents, serving billons of views daily. As such, the platform plays a vital role in shaping what content become viral or trending. This project aims to explore the patterns behind trending YouTube videos in United States (US) via data analytics techniques.**

I. Introduction

YouTube has become a central platform for video and video-like content, with billions of users globally engaging every day. From entertainment and education to news and social commentary, YouTube serves as a powerful tool for creators, brands, and viewers alike.

One of the fascinating aspects of YouTube is the *"trending"* videos—content that gains **rapid popularity and widespread attention in a short amount of time.** Trending videos take significant part in shaping internet culture and influencing global trends. However, understanding the precise patterns behind what makes a video trending on YouTube remains a challenging task.

By examining various data points such as viewer interaction, video attributes, we seek to uncover the key elements that drive a video's success. Through this analysis, we hope to gain a deeper understanding of the dynamics at play on YouTube, offering insights that could benefit content creators, marketers, and anyone interested in the evolving landscape of online video consumption.

II. The datasets - Introduction

A. Why we choose US

Firstly, it's an English-speaking country, which is the dominant language of YouTube's global user base. Thus, English content tends to have a broader reach and provides a clearer picture of contents that resonates with a wide range of viewers, domestically and internationally.

Secondly, US is one of the largest and most diverse markets for YouTube usage. With a wide variety of interests, demographics, and cultural influences, trends in the US often reflect broader global patterns, making it an ideal region for capturing viral content.

B. Where trending videos are curated

We extract the videos featured on YouTube's Trending page: https://www.youtube.com/feed/trending.

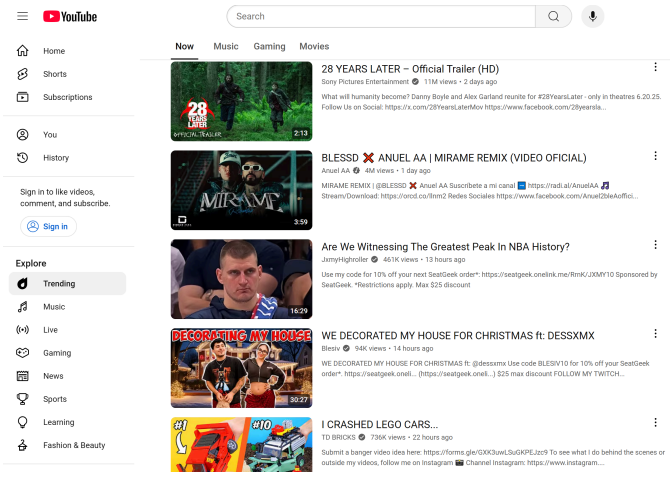


Figure 1: The YouTube Trending page for US

This page showcases a selection of videos that have gained significant traction and are rapidly gaining views across a variety of categories, such as music, gaming, news, entertainment, and more. The videos listed on this page are determined a combination of factors, such as **view count, engagement (likes, comments, shares), and the video's overall popularity within a specific time frame (usually around 1 week).**

The videos featured on this page reflect a snapshot of what is currently capturing the attention of users on the platform,.

For this project, we focus specifically on analyzing the videos listed on this Trending page.

III. The datasets - Details

We will curate two separate unprocessed datasets, which will be refined during cleaning:

- Trending video dataset: Includes 200 videos featured on the Trending page, re-collected on 11/12 for an update (as the Trending page changes frequently). The dataset reflects the Trending page as of **December 2024.**
- Channel video dataset: Contains videos from all channels featured on the Trending page, primarily for comparison and prediction purposes (detailed later).

To obtain these, we will use **Google's YouTube Data API v3**, which enables access to various statistics collected by YouTube. It provides users with information on different properties of a video.

A. Basic properties

In this context, we will only include the specific properties that we have gathered for our datasets. For a comprehensive list of all available video properties, please refer to the official YouTube Data API guide at this **link**[1]

1. `publishedAt`: The video's publication time in UTC (ISO 8601 format).
2. `title`: The video title
3. `channelTitle`: The channel which posted the video.
4. `channelId`: The channel id of `channelTitle`. Used for collecting the channel's videos.
5. `category`: A numerical category value from the API, converted into a string representing the video's category (based on official mapping in **Figure 2**).
6. `duration`: The video's duration (ISO 8601 format)
7. `licensedContent`: Whether the video is licensed (True/False)
8. `viewCount`, `likeCount`, `commentCount`: The video's view, like, and comment counts, respectively.
9. `topicCategories`: A list of Wikipedia URLs that provide a high-level description of the video's content, converted it into a list of string categories.

For example, `topicCategories` in API return this list *["https://en.wikipedia.org/wiki/Action-adventure_game"]*, result will be *["Action adventure game"]*

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

Figure 2: YouTube category mapping

[1]https://developers.google.com/youtube/v3/docs/videos#properties

B. Additional properties

Below are some more properties that we derive from the above data to enhance our dataset, and serve prediction stage:

1. `fetchedDate`: The time that the dataset was collected. Because Trending page can change significantly within a few weeks' time, we include this to accurately track the time to measure 'how much' a certain video is Trending (will be specified in later part)
2. `elapsedDays`: Number of days elapsed from the published time `publishedAt` to the day the dataset was collected `fetchedDate`, rounded to the nearest 4 numbers. This serves the creation of another metric.
3. `avgDailyViews`: The average daily number of views per day, calculated by taking `viewCount` divide by `elapsedDays`. This is a key metric in predicting which videos are likely to Trend.

Importance of `AvgDailyViews`

- As noted, Trending videos typically experience **rapid popularity** in a short time. This metric quantifies the **rate of attention** a video receives over time. Given that the API does not support continuous crawling, we assume a uniform distribution of views per day to estimate this rate.
- This allows us to factor in older videos that may have garnered massive attention in the past. For example, a video with 2 billion views posted 6 years ago would have an `avgDailyViews` of about 1.1 million, similar to a new video with that same daily view count, making it more likely to trend.
- This metric also 'punishes' the older, less relevant content that might still be considered popular. For instance, a video with 300 million views posted 10 years ago would have an `avgDailyViews` of around 80k, making it less likely to trend today despite its overall popularity.

4. `engagementRate`: The engagement rate of the video, calculated by taking `(likeCount + commentCount) / viewCount`. This is an another important metric in determining whether a video is genuinely Trending.

Importance of `engagementRate`

- The engagement rate offers a more reliable measure of a video's resonance with its audience compared to raw view counts, which can be skewed by clickbait, algorithmic promotion, or bots.
- A high engagement rate indicates that viewers are actively interacting with the content through likes, comments, and shares, signaling the video's quality and relevance.
- While Trending videos generally have high views, a strong engagement rate suggests sustained, organic interaction, distinguishing genuinely engaging content from videos driven by external factors, and providing a more accurate gauge of content impact and appeal.

5. `isTrending`: Indicates whether a video will appear on the Trending page in the US (1 for yes, 0 for no). The specification of this value will be explained in the subsequent sections.

On the omission of descriptions and tags

Initially, we did collect the video descriptions and tags, and attempted to process them using large language models (LLMs) for summaries. However, we chose not to prioritize this data for several reasons.

First, descriptions and tags can be highly variable in quality and relevance, often manipulated by creators for SEO purposes, which may not accurately reflect the video's content. Relying on this data could lead to misleading interpretations of the video's subject matter.

Second, YouTube's algorithm increasingly prioritizes engagement metrics, such as views, likes, and comments, over textual content when determining trends. These metrics offer more direct and reliable insights into a video's popularity and audience reception.

Last, collecting and processing large amounts of textual data is resource-intensive and introduces complexity, potential noise, and bias, which could compromise the dataset's quality and analysis.

For these reasons, we have focused on quantitative metrics that provide clearer, more objective indicators of a video's performance and its likelihood of trending.

IV. The datasets - Collection

A. Trending videos dataset

The `YouTubeTrendingScraper` class will be responsible for collecting the video information featured on the Trending page. It will conduct these following tasks:

1) *Make a request:*

```python
def api_request(self, page_token, country_code):
    request_url = (
        f"https://www.googleapis.com/youtube/v3/videos"
        f"?part=snippet,statistics,contentDetails,topicDetails&chart=mostPopular"
        f"&regionCode={country_code}&maxResults=50"
        f"&key={self.api_key}"
    )
    if page_token:
        request_url += f"&pageToken={page_token}"

    request = requests.get(request_url)

    if request.status_code == 429:
        print("Temp-banned due to excess requests, please wait and continue later")
        sys.exit()
    elif request.status_code != 200:
        print(f"Error: {request.status_code} - {request.text}")
        return {}

    return request.json()
```

Figure 3: The api_request function

This method will make a request to the page containing the information of videos feature on Trending page. The maximum number of featured videos is 50 per page, so if `pageToken` is provided, it's appended to `request_url` to handle pagination (go to the next page).

2) *Get the videos:*

```python
def get_videos(self, items):

    videos = {}

    for video in items:
        if "statistics" not in video or "contentDetails" not in video or "snippet" not in video:
            continue

        video_id = video.get("id", "")
        snippet = video["snippet"]
        statistics = video["statistics"]
        content_details = video["contentDetails"]
        topic_details = video.get("topicDetails", {})

        category_id = snippet.get('categoryId', "N/A")
        category_name = self.CATEGORY_MAPPING.get(category_id, "Unknown")

        raw_topic_categories = topic_details.get('topicCategories', None)
        processed_topic_categories = self.process_topic_categories(raw_topic_categories)

        published_at = snippet.get('publishedAt', None)
        view_count = statistics.get('viewCount', 0)

        exact_elapsed_days = self.calculate_exact_elapsed_days(published_at)
        average_views_per_day = self.calculate_average_views_per_day(view_count, exact_elapsed_days)

        # Construct video data
        video_data = {
            'fetchedDate': self.RECORDED_UTC_TIME,
            "publishedAt": published_at,
            'elapsedDays': round(float(exact_elapsed_days), 4),
            "title": snippet.get("title", ""),
            # "description": snippet.get("description", ""),
            "channelTitle": snippet.get("channelTitle", ""),
            "channelId": snippet.get("channelId", ""),
            # "tags": snippet.get("tags", None),
            "category": category_name,
            "duration": self.parse_duration(content_details.get("duration", "")),
            "licensedContent": content_details.get("licensedContent", False),
            "viewCount": int(view_count),
            'avgDailyViews': round(float(average_views_per_day), 2),
            "likeCount": int(statistics.get("likeCount", 0)),
            "commentCount": int(statistics.get("commentCount", 0)),
            "topicCategories": processed_topic_categories,
        }

        # Add video data to dictionary with video_id as key
        videos[video_id] = video_data

    return videos
```

Figure 4: The get_videos method, `description` and `tags` are omitted due to aforementioned reasons

This method will initialize an empty dictionary, then fetch the video data in the page, with all the necessary properties.

Processing methods inside `get_videos`:

- `category_name`: Parse the mapped `category_id`.
- `process_topic_categories`:

```python
def process_topic_categories(self, topic_categories):

    if not topic_categories or not isinstance(topic_categories, list):
        return []

    processed_categories = []
    for url in topic_categories:
        try:
            # Extract the last part of the URL after the last "/"
            category_name = url.split("/")[-1]
            # Replace underscores with spaces to make it more readable
            category_name = category_name.replace("_", " ")
            processed_categories.append(category_name)
        except Exception as e:
            print(f"Error processing URL '{url}': {e}")

    return processed_categories
```

This method will process the list of Wikipedia URLs into readable topics.

Before processing:

```
"topicCategories": [
    "https://en.wikipedia.org/wiki/Entertainment",
    "https://en.wikipedia.org/wiki/Film"
]
```

After processing:

```
"topicCategories":[
    "Entertainment",
    "Film"
],
```

- Calculate `avgDailyViews`

```python
def calculate_exact_elapsed_days(self, published_time):

    '''
    Calculate the exact elapsed days from the published time
    '''

    published_dt = datetime.strptime(published_time, "%Y-%m-%dT%H:%M:%SZ")
    elapsed = datetime.strptime(self.RECORDED_UTC_TIME, "%Y-%m-%dT%H:%M:%SZ") - published_dt
    elapsed_days = elapsed.total_seconds() / 60 / 60 / 24
    return f'{elapsed_days:.4f}'
```

```python
def calculate_average_views_per_day(self, view_count, exact_elapsed_days):

    '''
    Calculate the average views per day
    '''

    return f'{int(view_count) / float(exact_elapsed_days):.2f}'
```

3) *Write data to JSON file:*

Initially, we used a JSON format to process the textual data. This approach was chosen because certain special characters in the textual content of videos cannot be prop-

erly encoded in the CSV file format. JSON, being more flexible and capable of handling various character encodings, allows us to preserve the integrity of the data while enabling easier parsing and manipulation.

```python
def write_to_file(self, country_code, country_data):
    print(f"Writing {country_code} data to file...")

    if not os.path.exists(self.output_dir):
        os.makedirs(self.output_dir)

    file_path = os.path.join(self.output_dir, f"{time.strftime('%y.%d.%m')}_{country_code}_trending_videos.json")
    with open(file_path, "w", encoding="utf-8") as file:
        json.dump(country_data, file, ensure_ascii=False, indent=4)

    print(f"Data successfully written to {file_path}")

def scrape_data(self):
    for country_code in self.country_codes:
        print(f"Scraping data for country: {country_code}")
        country_data = self.get_pages(country_code)
        self.write_to_file(country_code, country_data)
```

Figure 5: Write to JSON method

This method will write the collected data to a JSON file.

B. *Channel video dataset*

1) *Reason for collection:*

After collecting data from the Trending page, we will proceed to collect video data from all the unique channels featured there. This will enable us to analyze not only the current trending content but also the performance of videos across different channels.

Additionally, we will identify videos that, while not currently trending, may have the potential to trend, or maintain a certain degree of relevance if posted today. This includes older videos that have sustained strong engagement or accumulated high view counts over time.

Those videos may not be trending at present, but they remain close to trending status due to their ongoing popularity and sustained audience interaction. Despite their age, they could still have significant impact and relevance within the current digital landscape if their historical performance and current engagement metrics are to be considered.

In essence, our analysis will not only focus on the immediate trending content but also consider videos from the past that are still relevant and capable of regaining traction, providing a more robust prediction model.

2) *Collection steps:*

The `YTStatsProMax` class will be responsible for this step. After collecting data for the trending dataset, we will use a helper function to fetch all the unique channels (`channelID`) into a `.txt` file to feed into this class.

This class will handle these tasks:

*Get the channel statistics **(depreciated)**:*

This method was initially designed to provide channel statistics for visualization tasks and offer an entry point for retrieving all video IDs from a channel.

Later, we found that this approach does not accurately reflect the total number of videos on a channel, which could result in missing data and other issues. The only reliable metric from this method is `subscriberCount`, which indicates the number of subscribers to a channel.

However, since the Trending page primarily focuses on the performance of individual videos rather than the channel's overall subscriber base (with some channels having relatively low subscriber counts still appearing on Trending), this method is no longer useful for our purposes. Despite this, we will include it here as a cross-reference in the code for completeness.

```python
def get_channel_statistics(self):
    '''
    Fetch channel statistics, basically it presses on the link and fetch those statistics the url.

    This method only serves for making filenames now as most of the analytically significant properties
    provided in here are also provided in video_data, as well as some 'malfunctioned' ones
    (like incorrect videoCount - may lead to incorrect viewCount), etc.
    '''
    url = f'https://www.googleapis.com/youtube/v3/channels?part=snippet,statistics&id={self.channel_id}&key={self.api_key}'
    data = self._make_request(url)
    if not data:
        print("Error: Failed to fetch channel statistics.")
        return {}

    try:
        items = data.get('items', [])
        if not items:
            print("Error: No items found in channel statistics response.")
            return {}

        overall_channel_data = items[0]['statistics']
        additional_channel_data = items[0]['snippet']


        self.channel_statistics = {
            'channelTitle': additional_channel_data['title'],
            'viewCount': int(overall_channel_data.get('viewCount', 0)),
            'subscriberCount': int(overall_channel_data.get('subscriberCount', 0)),
            'videoCount': int(overall_channel_data.get('videoCount', 0)), # provided in statistics, but it's incorrect
            # 'country': additional_channel_data.get('country', None), (optional since not every channel provide this)
            'channelCategory': self.get_channel_category()
        }

    except (KeyError, IndexError) as e:
        print(f"Error processing channel statistics: {e}")
        self.channel_statistics = {}
    return self.channel_statistics
```

Figure 6: Get channel statistics method

Access the channel playlists and get the video IDs:

```python
def _get_channel_videos(self, limit=None):

    '''
    We're taking channel videos in 2 of these links (playlists).
    The video data for those playlist are provided in the form of those urls (url1, url2)
    Those urls can be derived from channel_id as below.
    For example, if the channel_id is: UCeLHszkByNZtPKcaVXOCOQQ
    Then, the popular_playlist_id will be: UULPeLHszkByNZtPKcaVXOCOQQ (change middle C to 'ULP')
    And the popular_short_playlist_id will be: UUPSeLHszkByNZtPKcaVXOCOQQ (change middle C to 'UPS')
    The full upload playlist id is: UUeLHszkByNZtPKcaVXOCOQQ (change middle C to 'U')

    Why these playlists: To ensure the most popular videos of all channels are included in the dataset.
    But not all videos in this so-called 'popular' playlist are all popular videos.
    'popular' is different from 'trend'
    '''
    popular_playlist_id = self.channel_id[:1] + "ULP" + self.channel_id[2:]
    popular_short_playlist_id = self.channel_id[:1] + "UPS" + self.channel_id[2:]

    url1 = f'https://www.googleapis.com/youtube/v3/playlistItems?part=snippet&playlistId={popular_playlist_id}&key={self.api_key}'
    url2 = f'https://www.googleapis.com/youtube/v3/playlistItems?part=snippet&playlistId={popular_short_playlist_id}&key={self.api_key}'

    if limit:
        url1 += f"&maxResults={limit}"
        url2 += f"&maxResults={limit}"

    videos = {}
    videos.update(self._fetch_videos_from_playlist(url1))
    videos.update(self._fetch_videos_from_playlist(url2))

    return videos
```

Figure 7: Access the playlists method

```python
def _fetch_videos_from_playlist(self, url):
    videos = {}
    next_page_token = None

    for _ in range(3):
        '''
        Each playlist, we take 150 videos (50 videos per page, 3 loops)
        So, the maximum number of videos for each channel is 300.
        '''
        if next_page_token:
            url_with_token = f"{url}&pageToken={next_page_token}"
        else:
            url_with_token = url

        data = self._make_request(url_with_token)
        if not data:
            print("Error: Failed to fetch playlist videos.")
            break

        items = data.get('items', [])
        for item in items:
            try:
                video_id = item['snippet']['resourceId']['videoId']
                videos[video_id] = {}
            except KeyError as e:
                print(f"Error extracting video ID: {e}")

        next_page_token = data.get('nextPageToken')
        if not next_page_token:
            break

    return videos
```

Figure 8: Handle pagination and access the necessary parts, similarly to the make_request method in trending videos

This method will make a request to YouTube URLs in similar ways to the trending videos crawler (fetch the links to the page, as well as pagination handling)

To manage API limitations and dataset size, we restricted the crawler to **300 videos per channel**, resulting in a dataset of **18,000 videos** (before pre-processing).

Initially, we fetched videos from the channel's upload playlist, but this approach missed popular videos, as it starts with the most recent, and our restriction may not allow the crawler to reach the channel's highest performance videos.

To address this, we adopted a two-page fetching method: one for the **popular videos playlist** (likely sorted by view count) and another for **the popular short videos playlist** (videos under one minute).

We collect **150 videos** from each, ensuring a balanced dataset that includes both long-form and short content, while capturing the highest-performing videos for trending analysis. To see how each playlist is accessed, please refer to the documentation in **Figure 7** for details.

Get the videos:

```python
def get_channel_video_data(self):
    channel_videos = self._get_channel_videos(limit=50)
    if not channel_videos:
        print("Error: No videos found for this channel.")
        return {}

    print(f"Total videos found: {len(channel_videos)}")

    for video_id in tqdm(channel_videos.keys(), desc="Fetching video data"):
        video_url = f'https://www.googleapis.com/youtube/v3/videos
        ?part=snippet,statistics,contentDetails,topicDetails&id={video_id}&key={self.api_key}'

        data = self._make_request(video_url)
        if not data:
            print(f"Error: Failed to fetch data for video ID {video_id}.")
            continue

        try:
            items = data.get('items', [])
            if not items:
                print(f"Error: No items found for video ID {video_id}.")
                continue

            video_data = items[0]
            snippet = video_data['snippet']
            content_details = video_data['contentDetails']
            statistics = video_data['statistics']
            topic_details = video_data.get('topicDetails', {})

            view_count = int(statistics.get('viewCount', 0))
            if int(view_count) == 0:
                print(f"Skipping video ID {video_id} due to zero views.")
                continue

            category_id = snippet.get('categoryId', "N/A")
            category_name = self.CATEGORY_MAPPING.get(category_id, "Unknown")

            raw_topic_categories = topic_details.get('topicCategories', None)
            processed_topic_categories = self.process_topic_categories(raw_topic_categories)

            published_at = snippet.get('publishedAt', None)
            exact_elapsed_days = self.calculate_exact_elapsed_days(published_at)
            average_views_per_day = self.calculate_average_views_per_day(view_count, exact_elapsed_days)

            like_count = int(statistics.get('likeCount', 0))
            comment_count = int(statistics.get('commentCount', 0))

            engagement_rate = round((like_count + comment_count) / view_count, 4)

            self.video_data[video_id] = {
                'fetchedDate': self.RECORDED_UTC_TIME,
                'publishedAt': published_at,
                'elapsedDays': round(float(exact_elapsed_days), 4),
                'title': snippet.get('title', ""),
                'description': snippet.get('description', ""),
                'channelTitle': snippet.get('channelTitle', ""),
                'tags': snippet.get('tags', None),
                'category': category_name,
                'duration': content_details.get('duration', ""),
                'licensedContent': content_details.get('licensedContent', False),
                'viewCount': int(view_count),
                'avgDailyViews': round(float(average_views_per_day), 2),
                'likeCount': like_count,
                'commentCount': comment_count,
                'engagementRate': engagement_rate,
                'topicCategories': processed_topic_categories
            }
        except (KeyError, IndexError) as e:
            print(f"Error processing video data for video ID {video_id}: {e}")

    return self.video_data
```

Figure 9: The get_channel_videos method

Finally, we will collect videos from channels using a method similar to that employed for the Trending dataset. The process will follow the same approach for data retrieval and organization. Additionally, the side-processing methods, such as calculating `avgDailyViews`, determining the `category`, and other relevant metrics, will be consistent with those used in the Trending dataset to ensure uniformity in data handling and analysis.

Write to JSON:

```python
def dump_video_data(self, directory=None):
    """Dump video data to a JSON file."""
    if not self.video_data:
        print("Error: No video data to dump.")
        return
    if not self.channel_statistics:
        print("Getting channel statistics for filename")
        self.get_channel_statistics()

    filename = self._generate_filename(
        prefix=time.strftime('%y.%d.%m'),
        title=self.channel_statistics['channelTitle'],
        suffix="videos.json"
    )

    self._dump_to_json(self.video_data, filename, directory)
    print(f"Video data dumped to {filename}.")


def _generate_filename(self, prefix, title, suffix):

    safe_title = re.sub(r'[\\/*?:"<>|]', '_', title).replace(" ", "_").lower()
    return f"{prefix}_{safe_title}_{suffix}"


def _dump_to_json(self, data, filename, directory=None):
    # Determine the file path
    if directory:
        os.makedirs(directory, exist_ok=True)
        filepath = os.path.join(directory, filename)
    else:
        filepath = filename

    # Write
    with open(filepath, 'w', encoding='utf-8') as f:
        json.dump(data, f, indent=4, ensure_ascii=False)
```

Figure 10: Dump to JSON method

This method will dump the fetched channel video data from the previous steps, then dump to a JSON file, similar to the Trending dataset.

3) *Combine the data into 1 large dataset:*

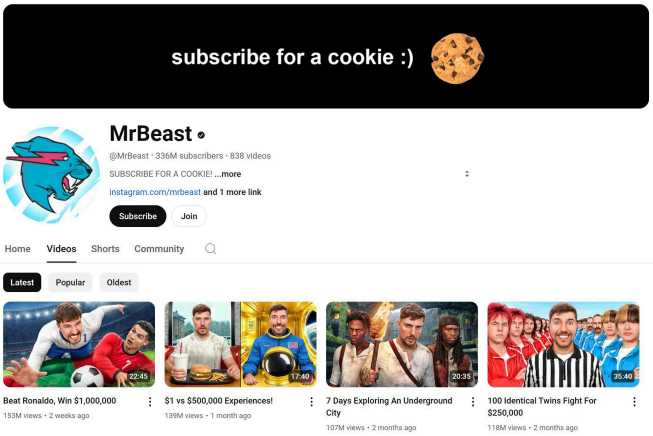
The `YTStatsProMax` class supports processing in batch, which means after crawling, we have multiple JSONs containing the channel video data of each channel featured in Trending. A simple helper script will combine these JSONs into a single concatenated dataset containing all channel video data of all channel featured in Trending dataset.

V. The Datasets - Preprocessing

After fetching both datasets, we obtained 2 datasets as followed:

- Trending videos dataset: 200 rows (200 videos)
- Channel videos dataset: 17651 rows (may contains duplicate videos from trending videos)

A. About a certain channel, MrBeast



MrBeast, with a record-breaking 330 million subscribers, consistently achieves exceptionally high view counts, likes, and comments. Notably, the top 10 videos in the channel's popular playlist each surpass **300 million views**, with engagement rates that are similarly outstanding. **In fact, most of his videos have more than 100 million views**, a number that hardly any YouTube channels can achieve.

Given MrBeast's unique and disproportionate influence on YouTube, its performance significantly deviates from the typical trends observed across most channels. This extreme level of success could distort our analysis, as content from MrBeast may not be directly comparable to similar content from other channels. As such, we have decided to exclude MrBeast from the primary analysis. To illustrate, we will present our after-cleaned trending dataset to compare MrBeast's performance to other channels.

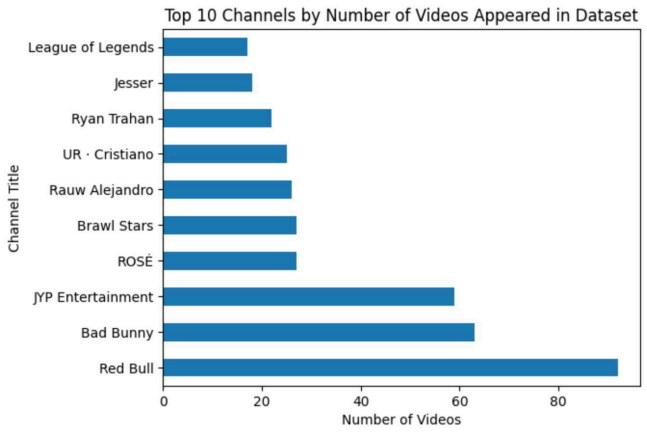


Figure 11: Top 10 channels that have the most videos appeared in Trending dataset

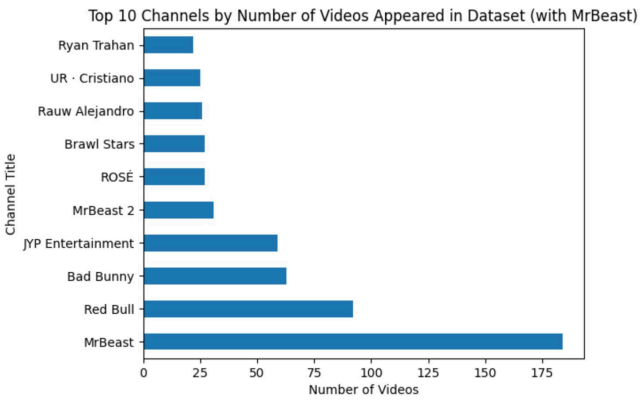


Figure 12: Top 10 channels that have the most videos appeared in Trending dataset, with MrBeast

Additionally, in our uncleaned channel dataset, MrBeast has 228 videos, of which approximately 180 have appeared in the Trending dataset. This further emphasizes the channel's dominant presence on YouTube and the potential skew it could introduce into our analysis.

B. *About the new stat, `isTrending`*

To support the prediction stage, we introduce a new variable, `isTrending`, which serves as a binary classification target for our prediction model.

- A value of 1 indicates that the video has appeared on the Trending page at the time of data collection, meaning it meets the criteria for trending as defined by YouTube's algorithm (e.g., high view counts, engagement, etc.).
- A value of 0 indicates that the video has not appeared on the Trending page, regardless of its performance, but still exists within the dataset for comparison or predictive purposes.

For the Trending dataset, the `isTrending` value is straightforward: all videos in this dataset are marked as 1 since they have already been featured on the Trending page at the time of collection.

For the channel dataset, however, `isTrending` is assigned based on the video's potential to trend if it were posted at the time of the analysis. In this case, we rely on metrics like `viewCount`, `likeCount`, `commentCount`, and our standardized `avgDailyViews`. By setting thresholds on these metrics, we can assign `isTrending = 1` to videos that meet the benchmark of trending potential, even if they haven't appeared on the Trending page in the past.

In both cases, `isTrending` serves as a target variable for prediction. For the Trending dataset, it reflects whether a video is trending at a specific moment. For the channel dataset, it represents the likelihood of a video trending in the future, based on its historical and current performance metrics.

C. *Trending video dataset*

```
<class 'pandas.core.frame.DataFrame'>
Index: 200 entries, mcvLKldPM08 to ccTtAOI_kP4
Data columns (total 16 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   fetchedDate      200 non-null    object
 1   publishedAt      200 non-null    object
 2   elapsedDays      200 non-null    float64
 3   title            200 non-null    object
 4   description      200 non-null    object
 5   channelTitle     200 non-null    object
 6   channelId        200 non-null    object
 7   tags             150 non-null    object
 8   category         200 non-null    object
 9   duration         200 non-null    object
 10  licensedContent  200 non-null    bool
 11  viewCount        200 non-null    int64
 12  avgDailyViews    200 non-null    float64
 13  likeCount        200 non-null    int64
 14  commentCount     200 non-null    int64
 15  topicCategories  200 non-null    object
dtypes: bool(1), float64(2), int64(3), object(10)
memory usage: 25.2+ KB
```

Figure 13: Trending dataset before cleaning

First, we will add the `engagementRate` to this dataset. This metric was inadvertently omitted during the this dataset's collection process, but its inclusion will not affect the results, as the calculation can be performed retroactively without altering the underlying data, and drop the `description` and `tags` columns because of mentioned reasons above.

1) *Determining cut-off criteria:*

To further filter the dataset and ensure only the most relevant videos qualify as trending, we will use percentile-based cut-offs for key metrics.

Specifically, we will eliminate the bottom 20th percentile of videos based on the following features `'viewCount'`, `'likeCount'`, `'commentCount'`, `'engagementRate'`, `'avgDailyViews'`

```
percentiles = trending_df[['viewCount', 'likeCount', 'commentCount', 'engagementRate', 'avgDailyViews']].quantile(0.2)
print("20th percentiles:")
print(percentiles)
```
[10]

```
20th percentiles:
viewCount          390311.600000
likeCount            8898.800000
commentCount          972.000000
engagementRate          0.016104
avgDailyViews      117688.160000
Name: 0.2, dtype: float64
```

Figure 14: Cut-off criteria for trending dataset

We take rounded numbers for the filtered dataset (keep the original numbers won't change number of data points). Only the videos which satisfy all the conditions remain in the dataset, ensuring that all videos are suitable representations of trending topics.

```
trending_df = trending_df[
    (trending_df["viewCount"] >= 390000)
    & (trending_df['likeCount'] >= 9000)
    & (trending_df['commentCount'] >= 1000)
    & (trending_df['engagementRate'] >= 0.015)
    & (trending_df['avgDailyViews'] >= 100000)
]
```
[11]

```
trending_df.shape[0]
```
[12]

```
108
```

Figure 15: The trending dataset after cutting off. 108 videos remained inside.

2) *Assign trending label:*

As mentioned previously, we will assign `isTrending` to each datasets.

`isTrending` of all videos in this dataset will be marked as 1, as they have already appeared on the Trending page.

`isTrending` will be assigned based on key metrics (`viewCount, likeCount, commentCount`) used by YouTube and our standardized `avgDailyViews` to determine trending videos. As such, **the lowest threshold** of these criteria will serve as the benchmark for determining whether a video from the channel dataset qualifies as trending.

```
trending_df['isTrending'] = 1
```
✓ 0.0s

Figure 16: Assign `isTrending = 1` to the Trending dataset

```
benchmarks = trending_df.groupby('isTrending').agg({
    'avgDailyViews': 'min',
    'viewCount': 'min',
    'likeCount': 'min',
    'commentCount': 'min'
}).reset_index()

print("Trending Benchmarks:")
print(benchmarks)
```
[14]

```
Trending Benchmarks:
   isTrending  avgDailyViews  viewCount  likeCount  commentCount
0           1      102733.75     397425       9619          1086
```

Figure 17: The benchmark for a Trending video

```
<class 'pandas.core.frame.DataFrame'>
Index: 14744 entries, V0CniCFbxLs to 9Nx849WhPFc
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   publishedAt      14744 non-null  object
 1   elapsedDays      14744 non-null  float64
 2   title            14744 non-null  object
 3   channelTitle     14744 non-null  object
 4   category         14744 non-null  object
 5   topicCategories  14744 non-null  object
 6   duration         14744 non-null  object
 7   licensedContent  14744 non-null  bool
 8   viewCount        14744 non-null  int64
 9   likeCount        14744 non-null  int64
 10  commentCount     14744 non-null  int64
 11  avgDailyViews    14744 non-null  float64
 12  engagementRate   14744 non-null  float64
 13  isTrending       14744 non-null  int32
dtypes: bool(1), float64(3), int32(1), int64(3), object(6)
memory usage: 2.0+ MB
```

Figure 18: Trending dataset after cleaning

D. *Channel video dataset*

```
<class 'pandas.core.frame.DataFrame'>
Index: 17651 entries, V0CniCFbxLs to uF1YHaeAHEw
Data columns (total 16 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   fetchedDate      17651 non-null  object
 1   publishedAt      17651 non-null  object
 2   elapsedDays      17651 non-null  float64
 3   title            17651 non-null  object
 4   description      17651 non-null  object
 5   channelTitle     17651 non-null  object
 6   tags             13662 non-null  object
 7   category         17651 non-null  object
 8   duration         17651 non-null  object
 9   licensedContent  17651 non-null  bool
 10  viewCount        17651 non-null  int64
 11  avgDailyViews    17651 non-null  float64
 12  likeCount        17651 non-null  int64
 13  commentCount     17651 non-null  int64
 14  engagementRate   17651 non-null  float64
 15  topicCategories  17651 non-null  object
dtypes: bool(1), float64(3), int64(3), object(9)
memory usage: 2.2+ MB
```

Figure 19: Channel dataset before cleaning

For this dataset, we will apply similar cleaning steps as those used in the Trending dataset, including dropping unnecessary columns and applying percentile-based cut-offs.

1) *Determining cut-off criteria:*

However, this time we will limit the cut-off to the **bottom 10th percentile**. This approach will help eliminate invalid videos (those with very few views or unpublished content), while retaining a more relevant subset of the data.

```
percentiles = channel_df[['viewCount', 'likeCount', 'commentCount', 'engagementRate', 'avgDailyViews']].quantile(0.1)
print("10th percentiles:")
print(percentiles)
```

```
10th percentiles:
viewCount         89309.0000
likeCount          2298.0000
commentCount         70.0000
engagementRate        0.0077
avgDailyViews       170.6000
Name: 0.1, dtype: float64
```

Figure 20: Cutoff criteria for channel dataset

This time, we will lower the `engagementRate` cut-off to 0.001, as certain videos, particularly music videos, tend to have relatively low engagement rates for various reasons (repetitive viewing, autoplay, etc.).

After adjusting this threshold, we will apply the cut-off to filter out videos with exceptionally low engagement.

```
channel_df = channel_df[
    (channel_df["viewCount"] >= 90000)
    & (channel_df['likeCount'] >= 2300)
    & (channel_df['commentCount'] >= 70)
    & (channel_df['engagementRate'] >= 0.001)
    & (channel_df['avgDailyViews'] >= 170)
]
```

```
channel_df.shape[0]
```

```
14744
```

Figure 21: Channel dataset after cutoff. 14744 videos remained inside

2) *Assign trending labels:*

Similar to the trending dataset, we will also assign `isTrending`. The result is a dataset with `isTrending` either `1` or `0`.

The videos with `1` (they satisfy Trending conditions) will be separated from videos with `0`, then get concatenated with trending videos to serve for prediction.

From trending dataset, we obtained the benchmark for trending videos are in **Figure 17**. The videos which satisfy **all these conditions** will be marked as trending (1) else (0)

```
trending_benchmarks = {
    'avgDailyViews': 102733.75,
    'viewCount': 397425,
    'likeCount': 9619,
    'commentCount': 1086
}

channel_df['isTrending'] = (
    (channel_df['avgDailyViews'] >= trending_benchmarks['avgDailyViews']) &
    (channel_df['viewCount'] >= trending_benchmarks['viewCount']) &
    (channel_df['likeCount'] >= trending_benchmarks['likeCount']) &
    (channel_df['commentCount'] >= trending_benchmarks['commentCount'])
).astype(int)
```

```
trending_count_df = channel_df[channel_df['isTrending'] == 1]

trending_count_df.shape[0] # videos in channel video dataset is marked as trending
```

```
613
```

Figure 22: `isTrending` assignment to all videos in channel dataset. 613 videos are marked as `isTrending`

```
<class 'pandas.core.frame.DataFrame'>
Index: 14744 entries, V0CniCFbxLs to 9Nx849WhPFc
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   publishedAt      14744 non-null  object
 1   elapsedDays      14744 non-null  float64
 2   title            14744 non-null  object
 3   channelTitle     14744 non-null  object
 4   category         14744 non-null  object
 5   topicCategories  14744 non-null  object
 6   duration         14744 non-null  object
 7   licensedContent  14744 non-null  bool
 8   viewCount        14744 non-null  int64
 9   likeCount        14744 non-null  int64
 10  commentCount     14744 non-null  int64
 11  avgDailyViews    14744 non-null  float64
 12  engagementRate   14744 non-null  float64
 13  isTrending       14744 non-null  int32
dtypes: bool(1), float64(3), int32(1), int64(3), object(6)
memory usage: 2.0+ MB
```

Figure 23: Channel dataset after cleaning

E. Some last touch-up

Separate `isTrending = 1` rows from `isTrending = 0` from the channel video dataset. Rows with `isTrending = 1` will join trending dataset as trending videos.

```
print(trending_df.shape[0])

channel_df_trending = channel_df[channel_df['isTrending'] == 1]
channel_df_not_trending = channel_df[channel_df['isTrending'] == 0]

print("Trending videos:")
print(channel_df_trending.shape[0])

print("\nNot Trending videos:")
print(channel_df_not_trending.shape[0])
```

```
108
Trending videos:
613

Not Trending videos:
14131
```

Figure 24: Channel dataset after cleaning

We then concatenate both datasets: the separated `isTrending = 1` videos from channel dataset with the trending dataset, then drop duplicate videos. Below is the result of the trending dataset and the channel dataset after all cleaning steps

```
<class 'pandas.core.frame.DataFrame'>
Index: 651 entries, 125 to 640
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   publishedAt      651 non-null    object
 1   elapsedDays      651 non-null    float64
 2   title            651 non-null    object
 3   channelTitle     651 non-null    object
 4   category         651 non-null    object
 5   topicCategories  651 non-null    object
 6   duration         651 non-null    object
 7   licensedContent  651 non-null    bool
 8   viewCount        651 non-null    int64
 9   likeCount        651 non-null    int64
 10  commentCount     651 non-null    int64
 11  avgDailyViews    651 non-null    float64
 12  engagementRate   651 non-null    float64
 13  isTrending       651 non-null    int64
dtypes: bool(1), float64(3), int64(4), object(6)
memory usage: 71.8+ KB
```

Figure 25: Trending dataset after all steps

```
<class 'pandas.core.frame.DataFrame'>
Index: 14131 entries, V0CniCFbxLs to 9Nx849WhPFc
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   publishedAt      14131 non-null  object
 1   elapsedDays      14131 non-null  float64
 2   title            14131 non-null  object
 3   channelTitle     14131 non-null  object
 4   category         14131 non-null  object
 5   topicCategories  14131 non-null  object
 6   duration         14131 non-null  object
 7   licensedContent  14131 non-null  bool
 8   viewCount        14131 non-null  int64
 9   likeCount        14131 non-null  int64
 10  commentCount     14131 non-null  int64
 11  avgDailyViews    14131 non-null  float64
 12  engagementRate   14131 non-null  float64
 13  isTrending       14131 non-null  int64
dtypes: bool(1), float64(3), int64(4), object(6)
memory usage: 1.5+ MB
```

Figure 26: Channel dataset after all steps

VI. Explonatory Data Analysis

To analyze the factors influencing our Trending dataset, we will provide some visualizations down below. The full visualization dashboard can be accessed via `YoutubeVideosDashboard.pbix` file in the submitted folder. (requires **Microsoft PowerBI**).

PowerBI is a popular software used for data visualization tasks. It allows interaction with the charts (on mouseclick) to uncover deeper data insights.

A. General channel information

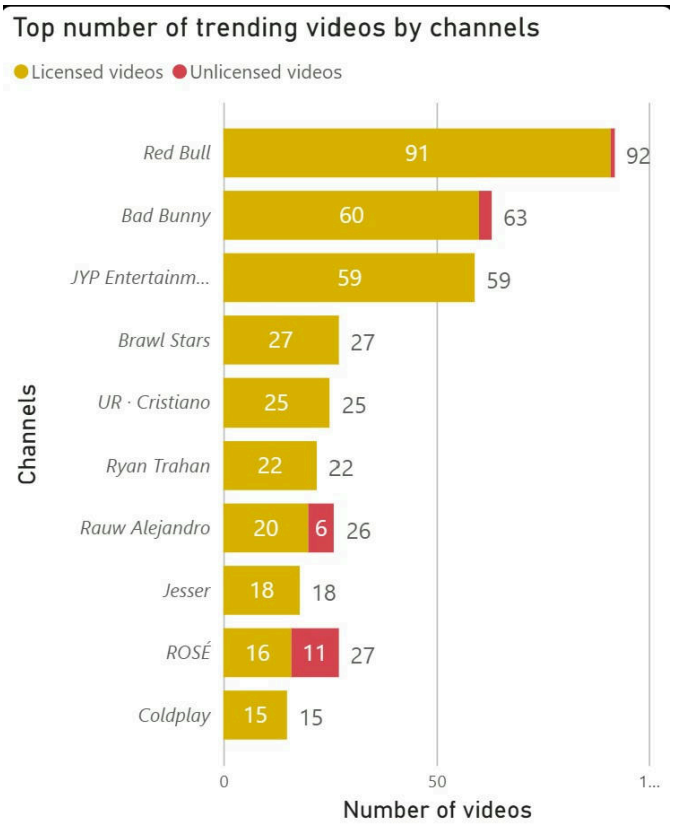


Figure 27: Top channels by number of videos appeared in Trending

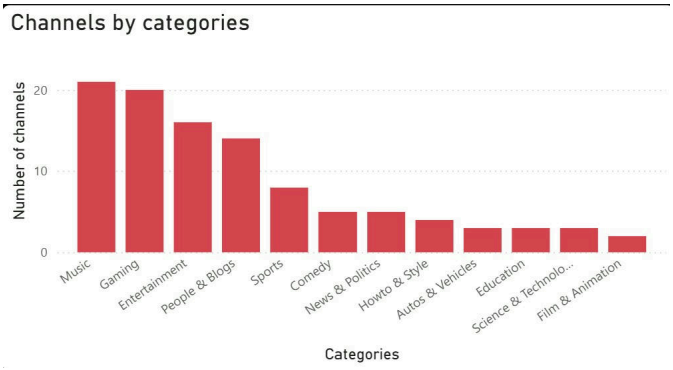


Figure 28: Top categories of channels on Trending

B. General Trending videos information

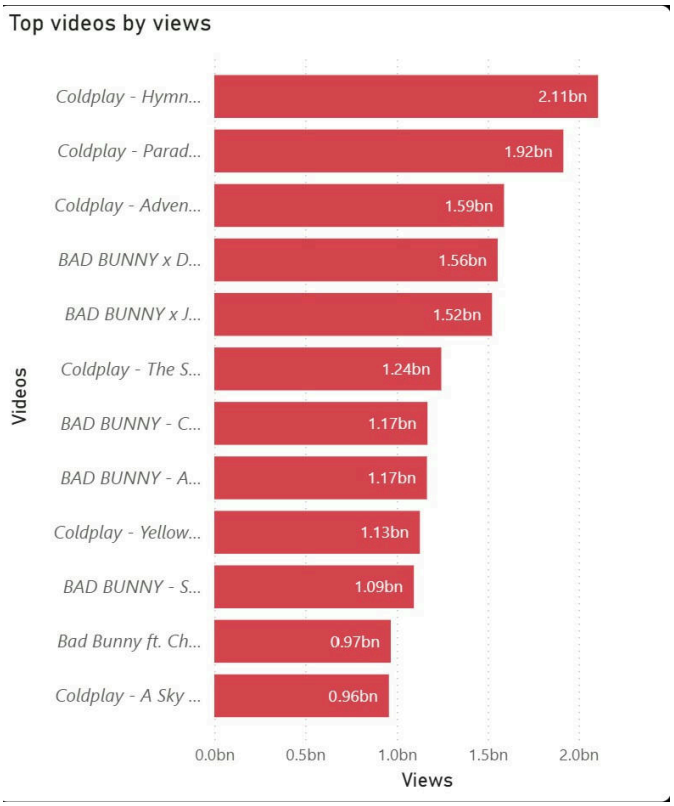


Figure 29: Videos with the highest views in Trending

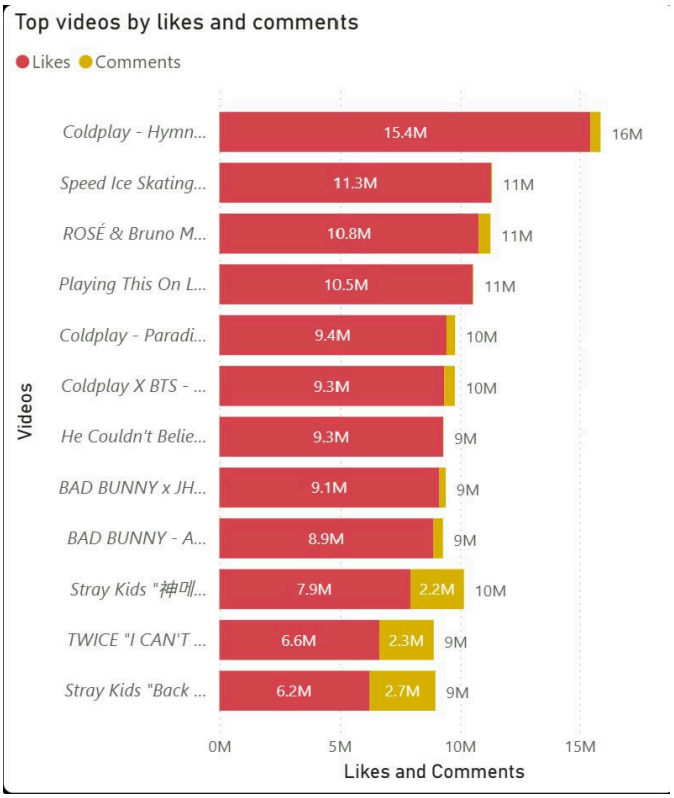


Figure 30: Videos with the highest likes & comments in Trending

VII. Prediction

This part is dedicated to outlines the development and evaluation of our prediction models using the dataset collected above. The primary goal of the model is to classify whether a video is trending based on its characteristics. This is a binary classification machine learning problem.

A. Data preparation

From the dataset, we use the following numerical features `viewCount`, `likeCount`, `commentCount`, `avgDailyViews`, `engagementRate`, `duration` and categorical features `topic` and `category`. The numerical values in the dataset have positive-skewness, so, in order to deal with that, we apply log transformation and StandardScaler:

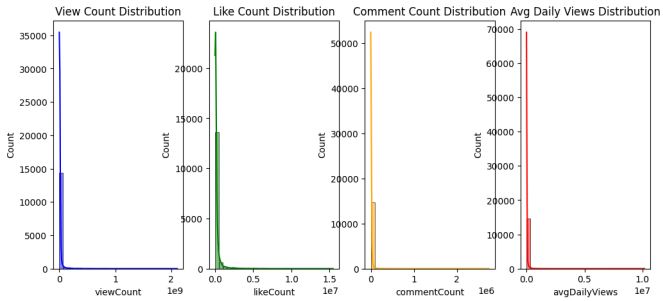


Figure 31: Numerical values distribution in the dataset

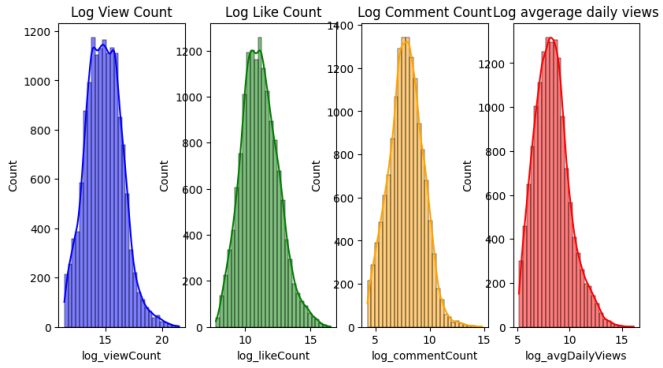


Figure 32: Numerical values distribution in the dataset

The `duration` and `publishedAt` columns are time-related, so we convert them into calculable forms. The `topicCategories` column is handled by calculating the weights sum based on how frequently the topic categories appear. By applying OneHotEncoder, we encoded 14 video categories of `category` column into usable data for predicting process.

```
topic_freq = {}
for topics in data['topicCategories']:
    for topic in topics:
        topic_freq[topic] = topic_freq.get(topic, 0) + 1

def topic_weight_sum(topics):
    return sum(topic_freq[topic] for topic in topics)
```

Figure 33: The *topicWeightsum* function

B. Model Development

We examined 4 machine learning models: Logistic Regression (LR), Random Forest (RF), Support Vector Machine (SVM) and Multi-layer Perception (MLP). The model were implemented using libraries from skicit-learn and trained on our dataset.

The dataset was split into training and test set with an ratio 80-20. To address class imbalance, SMOTE (Synthetic Minority Over-sampling Technique) was applied. We organized the preprocessing into pipelines, allowing integration of scaling, encoding, and oversampling. Initially, default hyperparameters were used for all models. We employed grid search for performance optimization, combining with Stratified K-Folds cross validation to ensure reliable evaluation.

C. Results and Evaluation

The evaluation metrics for the models are mainly precision and F1-score. We also evaluate using the confusion matrix, classification report, PR-AUC curve. The following is the results of our trained models:

Logistic Regression (LR):

Class	Precision	Recall	F1-Score	Support
0	1.00	0.99	1.00	2827
1	0.88	0.99	0.93	130

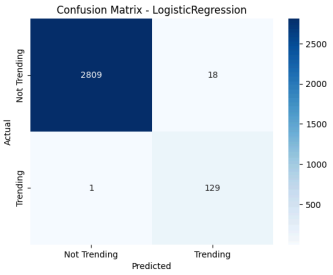


Figure 34: Confusion matrix of *LR*

Support Vector Machine classifier (SVM):

Class	Precision	Recall	F1-Score	Support
0	1.00	0.99	1.00	2827

Class	Precision	Recall	F1-Score	Support
1	0.90	0.98	0.94	130

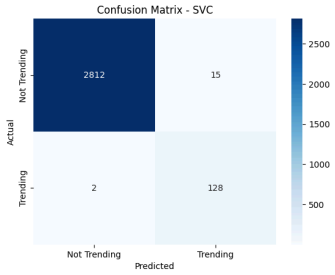


Figure 35: Confusion matrix of *SVM*

Multi-layer Perception (MLP):

Class	Precision	Recall	F1-Score	Support
0	1.00	0.99	0.98	2827
1	0.92	0.98	0.95	130

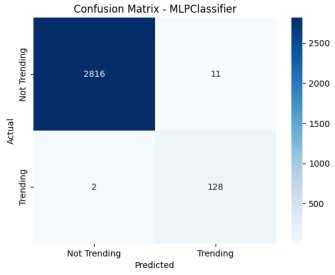


Figure 36: Confusion matrix of *MLP*

Random Forest (RF)

Class	Precision	Recall	F1-Score	Support
0	1.00	0.99	0.98	2827
1	1.00	0.99	1.00	130

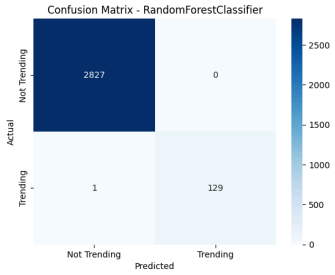


Figure 37: Confusion matrix of *RF*

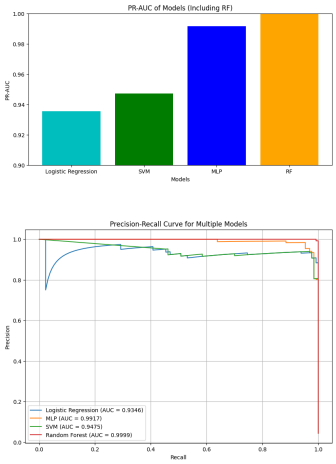


Figure 39: The PR_AUC curve of models

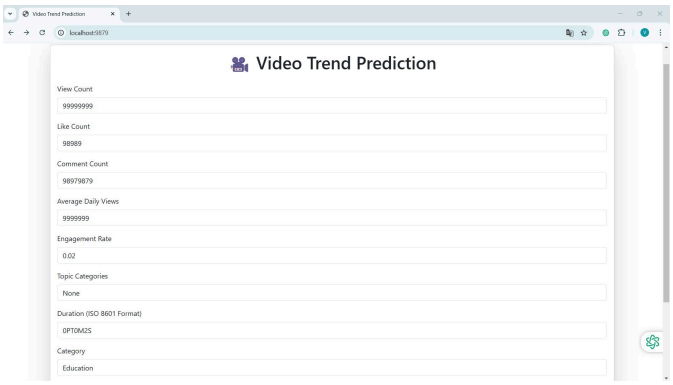
From the results, it looks like all four models perform fairly well, even with the imbalance in the test set. *Random Forest* stands out as the most reliable model, doing a good job of balancing precision and recall. *MLP* also performs very well, which suggests that a neural network-based approach can be effective. SVM does a decent job too but does not generalize as well as the top two models when trying to maximize both precision and recall. Logistic Regression, being the simplest model, has the weakest performance, especially when recall is important.

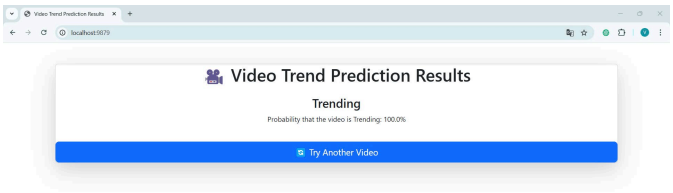
Overall, although there still exists some misclassified trending videos, but the number of cases is less than 1%, which is relatively small. In the future, more work can be done to improve the performance of our models such as: more hyperparameters tuning, explore different models, collecting more data to better handle the imbalance.

D. Prediction demonstration

This part outlines the process of creating predictive models, preparing a preprocessor for handling incoming data, and building a web-based graphical user interface (GUI) using Flask to make predictions using the trained models. The goal of this part is to enable users to input data, upload models, and preprocessors, and receive predictions through an interactive interface.

The Graphic User Interface (GUI) we created is a simple but user-friendly web interface that was built using HTML and Bootstrap. Flask is used to create a lightweight web application to interact with the model and preprocessor. The app provides the following functionalities: form submission, file upload, prediction display.





When an user submits data through the GUI, the form data is collected and preprocessed. The following features are extracted from the form: view count, like count, comment count, average daily views, engagement rate, topic categories, duration (ISO 8601 format), category, model file, preprocessor file.

We reuse the functions used to preprocess training data to process collected user data. The data will then be processed using the loaded preprocessor. In the context of our project, we use Standard Scaler to standardize numerical features and One Hot Encoder to convert categorical data into a numerical format that machine learning models can work with.

The preprocessed data is input into the trained model (loaded to the web). The model predicts whether the video is likely to trend (i.e., a binary outcome). Additionally, a prediction probability is also calculated, providing the user with a probability score indicating at what probability the video will be trending.

VIII. Conclusion

Implementations:

The datasets, visualization dashboard, as well as the prediction model can be accessed via the submitted folder on Teams.

Acknowledgements:

This work is supported and supervised by professor Than Quang Khoat under the course of Introduction to Data Science. Also, our research and project are partially supported by other students at School of Information and Communication Technology, Hanoi University of Science and Technology.