Real-time Data Processing and Analysis of YouTube Analytics using Pyspark and MapReduce

Big Data Processing (IT 609) **Course Instructor: Dr Sandip Modha**

202218036@daiict.ac.in (Rutul Patel)
202218002@daiict.ac.in (Rohan Shah)
202218010@daiict.ac.in (Nauman Shaikh)
202218004@daiict.ac.in (Karan Parashar)

Abstract: Our work is centered around implementing different functionalities of YouTube Data API (v3) from Google Cloud Platform (GCP) using PySpark and MapReduce. We have explained how video metrics can be inferred from the outputs and demonstrated one use case—defining a function that returns the metrics of the top fifty YouTube videos for a user given the word (search_query). Furthermore, we have also generated streaming data (comments) of popular/trending video(s).

Keyword(s): PySpark, MapReduce, Google Cloud Platform (GCP)

Index

I. Introduction.....[2-4]
II. Required Packages and Libraries.....[5-7]
III. Channel statistics.....[8]
IV. User-defined video statistics.....[9-10]
V. Visualization.....[10-12]
VI. Use case .....[13]
VII. Streaming Data.....[13]
VIII. Future Work.....[13]
IX. References.....[13]
X. ADD-ON (Implementing Apache Kafka).....[14]

I. Introduction:

As the amount of data on YouTube continues to grow exponentially, developing efficient ways to process and analyse this data. Has become increasingly essential. In this project, we have demonstrated the usage of the YouTube Data API to retrieve and process large amounts of data from YouTube.

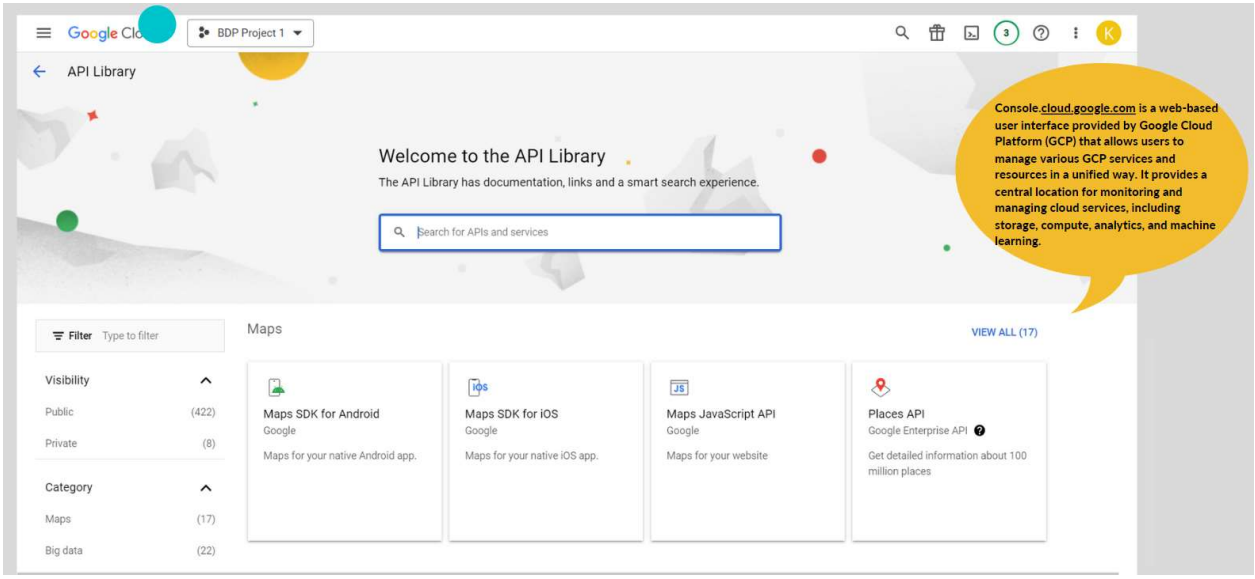


Figure 1. Home age of GCP API library

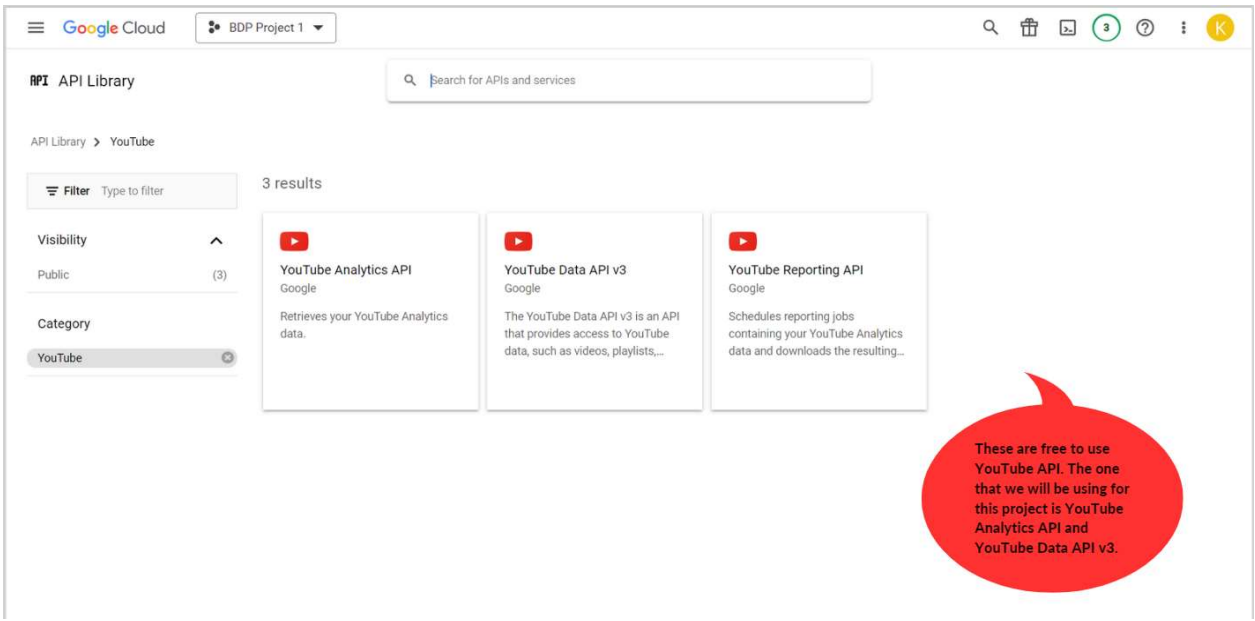


Figure 2. YouTube APIs (we have used YouTube Data API v3)

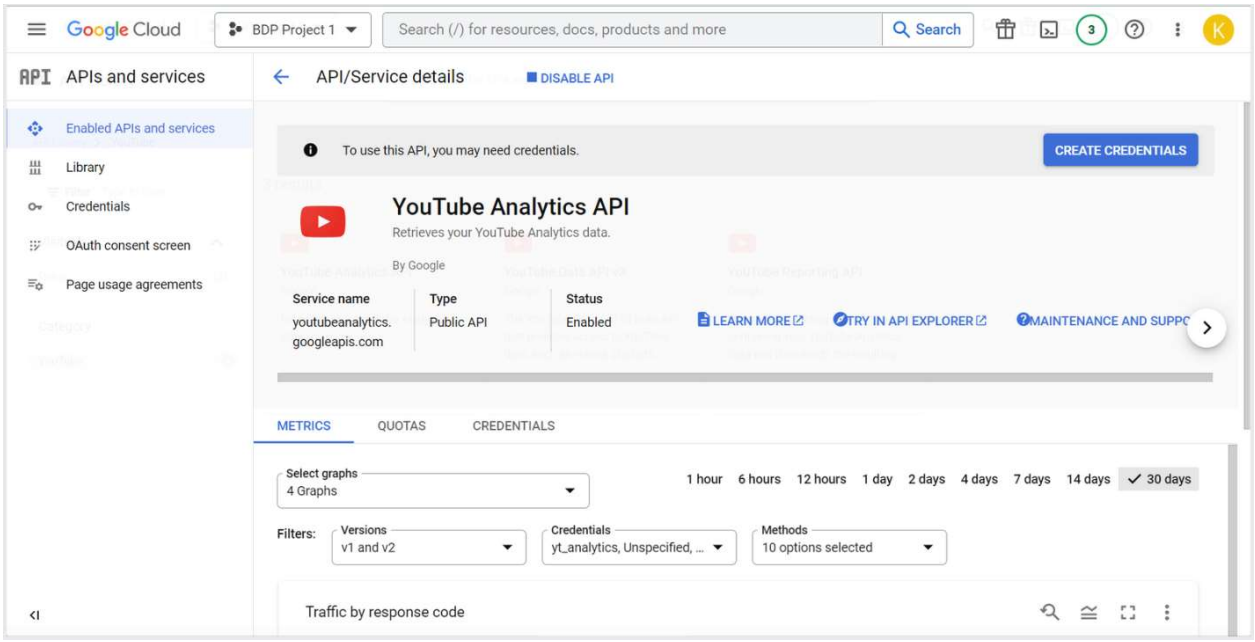


Figure 3: CREATING CREDENTIALS for the API

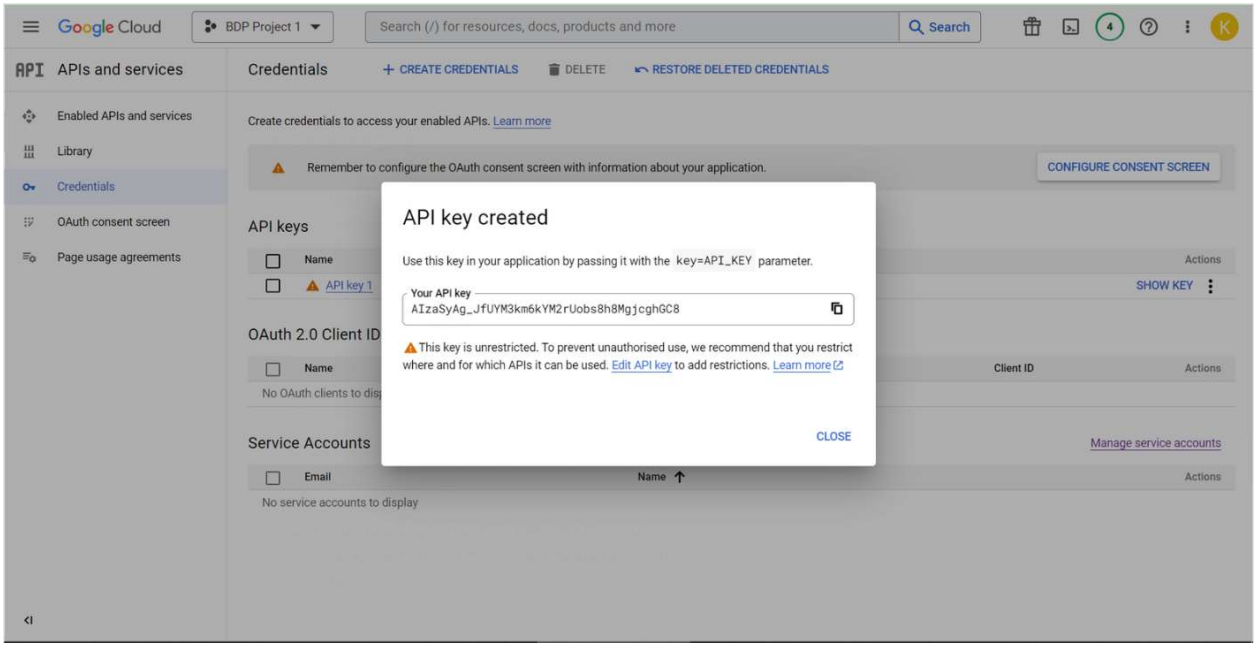


Figure 4: API key generated

II. Required Packages and Libraries:

OS: The os module in Python provides an operating system-dependent functionality like reading or writing to the file system, working with processes, and obtaining system information.

re: There (regular expression) library in Python supports pattern matching and manipulation of strings. It is used for searching and replacing specific patterns within a string and is commonly used in text processing and data cleaning tasks.

Request: requests is a popular Python library for making HTTP requests to web servers.
> It simplifies making HTTP requests and handling responses, providing a more Pythonic interface than the built-in urllib library.

Pandas: Pandas are built on top of NumPy, which is fast and efficient.
It includes functions for data cleaning, transformation, and exploration.

Datetime: The DateTime library in Python provides classes to work with dates and times.
> It includes functions for parsing, formatting, and manipulating dates and times.

Google.auth: google-auth is a Python library that provides Google-authenticated access to Google APIs. It is used for managing and authenticating with Google Cloud services.
> It supports various authentication methods, including OAuth 2.0 and Service Accounts.

PySpark: PySpark is a Python API for Apache Spark, an open-source, distributed computing system for big data processing and analytics. PySpark provides an interface to work with Spark using Python programming language.

Spark is designed to work with large data sets and provides a unified programming model for batch processing, real-time processing, machine learning, and graph processing. It achieves high performance by utilising in-memory processing and data parallelism, making it suitable for processing large-scale datasets in a distributed manner across clusters of computers.

PySpark provides a high-level API that allows developers to write Spark applications using Python code. It includes support for distributed data processing, SQL queries, machine learning algorithms, graph processing, and more. PySpark integrates well with popular Python libraries such as NumPy, pandas, and scikit-learn.

Kafka: Apache Kafka is an open-source distributed streaming platform for building real-time data pipelines and streaming applications. LinkedIn initially developed it and later donated it to the Apache Software Foundation.

Kafka is designed to handle high-volume, high-velocity data streams by providing a reliable, scalable, and fault-tolerant way to process and store data. It allows developers to build real-time data processing applications to process and analyse data as it is generated or received.

At its core, Kafka uses a publish-subscribe model for data distribution. Producers write data about topics, and consumers read data from subjects. Kafka stores data in partitions distributed across a cluster of servers, allowing for horizontal scaling and fault tolerance.

Kafka has become famous for building modern data pipelines, streaming applications, and microservices-based architectures. It is widely used in finance, e-commerce, healthcare, and media industries, where real-time data processing is critical for business success.

Kafka provides a distributed streaming platform for building real-time data pipelines that can handle high-volume, high-velocity data streams and provides a reliable and scalable way to process and store data.

Google.oauth2.credentials: The google.oauth2.credentials module provides a class called Credentials, which represents the authorisation credentials used to access Google APIs.
> The Credentials class is used to authenticate users and authorise access to their Google APIs.
> The credentials object can be created using various methods, such as from_authorized_user_info for creating certificates from a JSON file or from_user_info for creating credentials from a dictionary.

Google_auth_oauthlib.flow: The InstalledAppFlow class from the google_auth_oauthlib.flow library provides a simple way to obtain user authorisation for accessing Google APIs. It is commonly used for desktop or mobile applications that require user authentication.
> The flow handles the OAuth2 process, including presenting the user with an authorisation prompt, obtaining and storing the access and refresh tokens, and refreshing the access token when necessary.

Googleapiclient.errors: The HttpError class represents errors that occur during HTTP requests made to the Google APIs. It contains information about the HTTP status code, error message, and other details of the error. This library is part of the Google API client library for Python.

Googleapiclient.discovery: The build function from googleapiclient. Discovery creates a service object for interacting with a Google API.
> Authenticating the API request requires parameters such as the API name, version, and credentials.
> The returned service object can be used to make API requests to the specified Google API

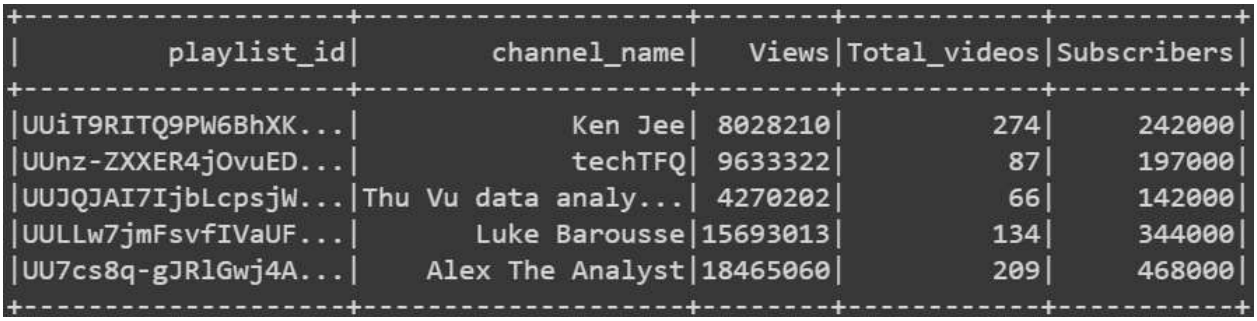
Matplotlib.pyplot: matplotlib.pyplot is a sub-library of Matplotlib, providing a convenient interface for visualisation creation.
> It contains functions that can be used to create line plots, scatter plots, bar plots, histograms, and many other types of visualisations.

plotly.graph_objs: plotly.graph_objs is a part of the Plotly library, which creates interactive data visualisations. Some minor points about this library are:

It provides a variety of objects and classes to create different types of charts and visualisations, such as scatter plots, bar charts, line charts, and more.
It allows customisation of the chart appearance and interactivity, such as adding annotations, zooming, and panning.

III. Channel Statistics:

We have generated metrics using their VIDEO_IDs for a set of youtube videos. In this, we have created a spark session, namely, YouTube Metrics.



playlist_id	channel_name	Views	Total_videos	Subscribers
UUiT9RITQ9PW6BhXK...	Ken Jee	8028210	274	242000
UUnz-ZXXER4jOvuED...	techTFQ	9633322	87	197000
UUJQJAI7IjbLcpsjW...	Thu Vu data analy...	4270202	66	142000
UULLw7jmFsvfIVaUF...	Luke Barousse	15693013	134	344000
UU7cs8q-gJRlGwj4A...	Alex The Analyst	18465060	209	468000

Figure 5. Metrics RDD

Using the YouTube API, we created using build() and the channel_id's we are fetching the following data:

(1) **'Channel Name':** name of the channel
(2) **'Subscribers':** The number of subscribers the channel has
(3) **'Views':** Total number of views the channel has on every video combined
(4) **'Total Videos':** Total number of videos uploaded by the channel
(5) **'Playlist_id':** Id of the playlist that contains all the channel's uploaded videos

After fetching the data, we created a Spark dataframe or RDD of the data.

IV. User-defined video statistics:

We have defined **Build_video_ids(search_query, max_results)**, which takes two parameters, namely "search_query" and "max_results". It returns a list of "max_results" number VIDEO IDs which contains the string "search_query" in its title (in decreasing number of views)

```
1 # search for the top 50 (ascending number of views) videos on YT that has 'iPhone' in their title.
2
3 video_ids = Build_video_ids('iPhone', 50)
4 print(video_ids)
```

```
['u7A5J_l-wBc', 'FIC9GnDVYrE', 'pL1bMfIUr80', 'BFIgk34c0e8', 'OQYnqFZpc8o', 'UYlEEEZ_v7E', 'L1uoESXxxQ4',
```

Figure 6. For the "iPhone" keyword, extract upto 50 results

Metrics_Generator(search_query, max_results) is another function that takes the help of **Build_video_ids** to generate an RDD of video metrics of all the videos in video_ids. This function retrieves the video metrics, such as id, channel, video name, total views, likes, comments and length of videos.

id	channel	name	views	likes	comments	length
pL1bMfIUr80	TechRax	Dropping an iPhon...	111461727	1157115	68352	4.95
qK_qK_UKu-Q	Beast Reacts	Most Expensive iP...	48217353	861057	12073	8.05
CpfNJoIdWpQ	Mrwhosetheboss	Android vs iPhone...	42260878	2969368	11989	1.0
2iOhmXRe6jQ	Lokdhun Punjabi	Munda iPhone Warg...	29943289	262364	5641	3.8
cWU2uHIKbyI	Hassjack	1 rupaye me Iphon...	28621233	1248792	570	1.0
FT3ODSg1GFE	Apple	Introducing iPhon...	25997972	597199	0	4.33
K00UfYqG0n4	Luisito Comunica	Compré el iPhone ...	21817233	762890	56912	12.7
1HWUjMjaBJI	Apple	Privacy on iPhone...	20859153	22304	0	5.9
NNYdGWhkewA	JaiPhone	Lucky Day! Found ...	18301685	140564	2786	14.78
JoIdAhbvPrs	Singer Prashant D...	भारी नको भरु साली...	16442983	75330	2187	3.57
hqGxg8DTsK0	Aquatic Monkey	iPhone found in C...	16301327	654311	1553	1.0
ck3VfZcx-RI	Galaxy Restore	Oops...! Found a ...	16076628	144196	6171	21.37
WO0iOnCYlRw	Nasya Toys	Cara Main mesin c...	14581502	537270	22920	1.0
l31SuEk4m2M	Salman Noman	4 lakh ka Iphone ...	14096546	1355879	1306	1.0
8FpPSMIB4uA	Marques Brownlee	Reviewing EVERY i...	13603579	382546	21545	33.25
G15Yss6XcJA	Tech Master Shorts	iphone के साथ क्य...	12048918	986280	931	1.0
SdLShOCvVeM	Marques Brownlee	iPhone 14 Pro Rev...	11713904	338335	18834	22.35
xSS-nm_ldPE	Rafa & Luiz	DEMOS UM IPHONE 1...	9970643	285133	18755	72.2
pA1f__kYptg	Crazy XYZ	Buying iPhone 14 ...	9755428	506326	10230	16.92
cTs4T7HKzbU	Dowba Montana	Dowba Montana - I...	9671239	101420	2170	3.93

only showing top 20 rows

Figure 7:

Also, created the function **parse_duration** that converts the duration(time) of the video into **mm.ss** format. We created a function named MapReduce(), which takes

one parameter, “k”, an RDD. rdd.map() extracts views, likes and comments from rdd. reduce() method is used to calculate the sum of views, likes and comments individually.

V. Visualization:

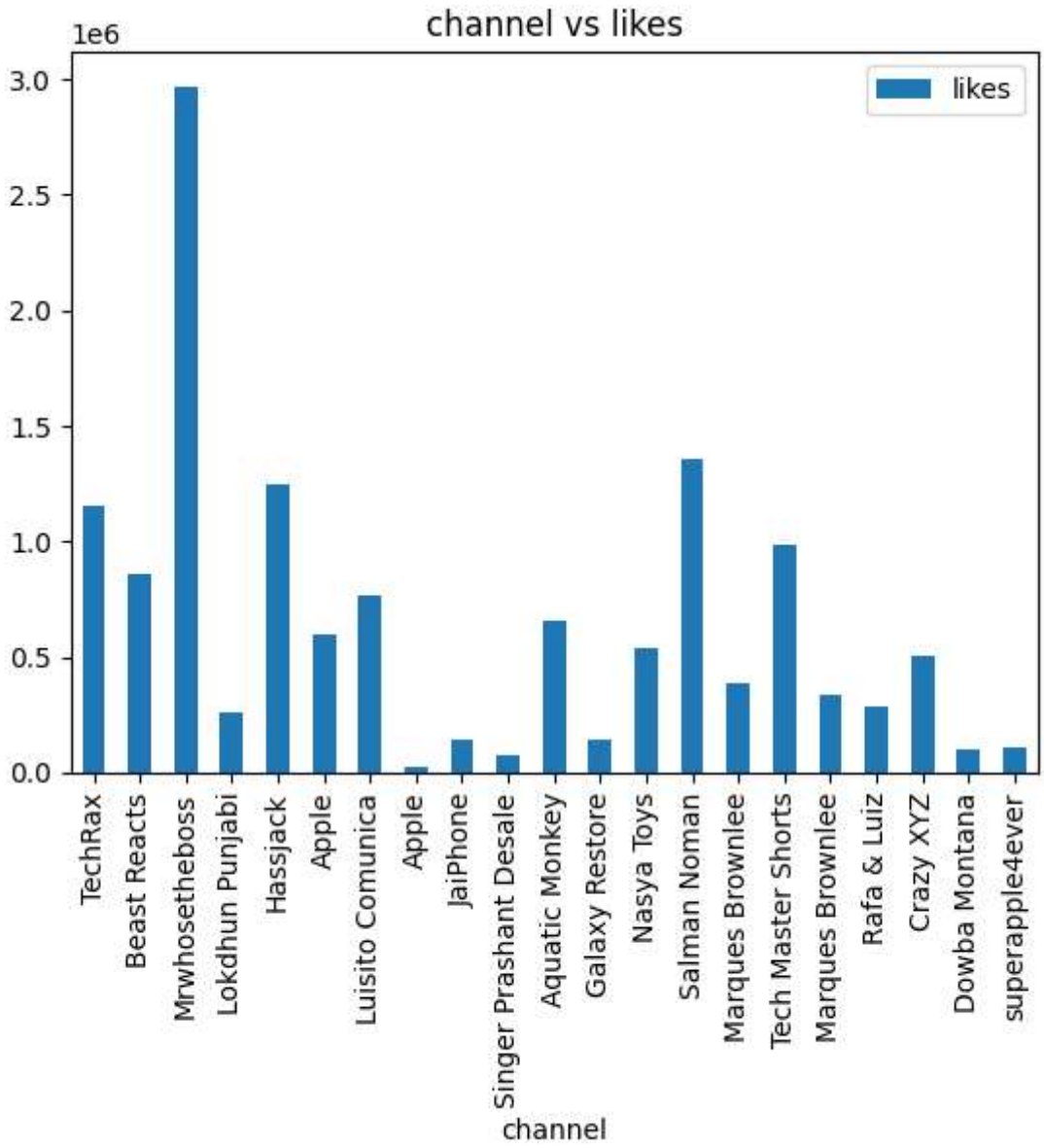


Figure 8: Bar chart of likes vs channels whose video appears in the RDD

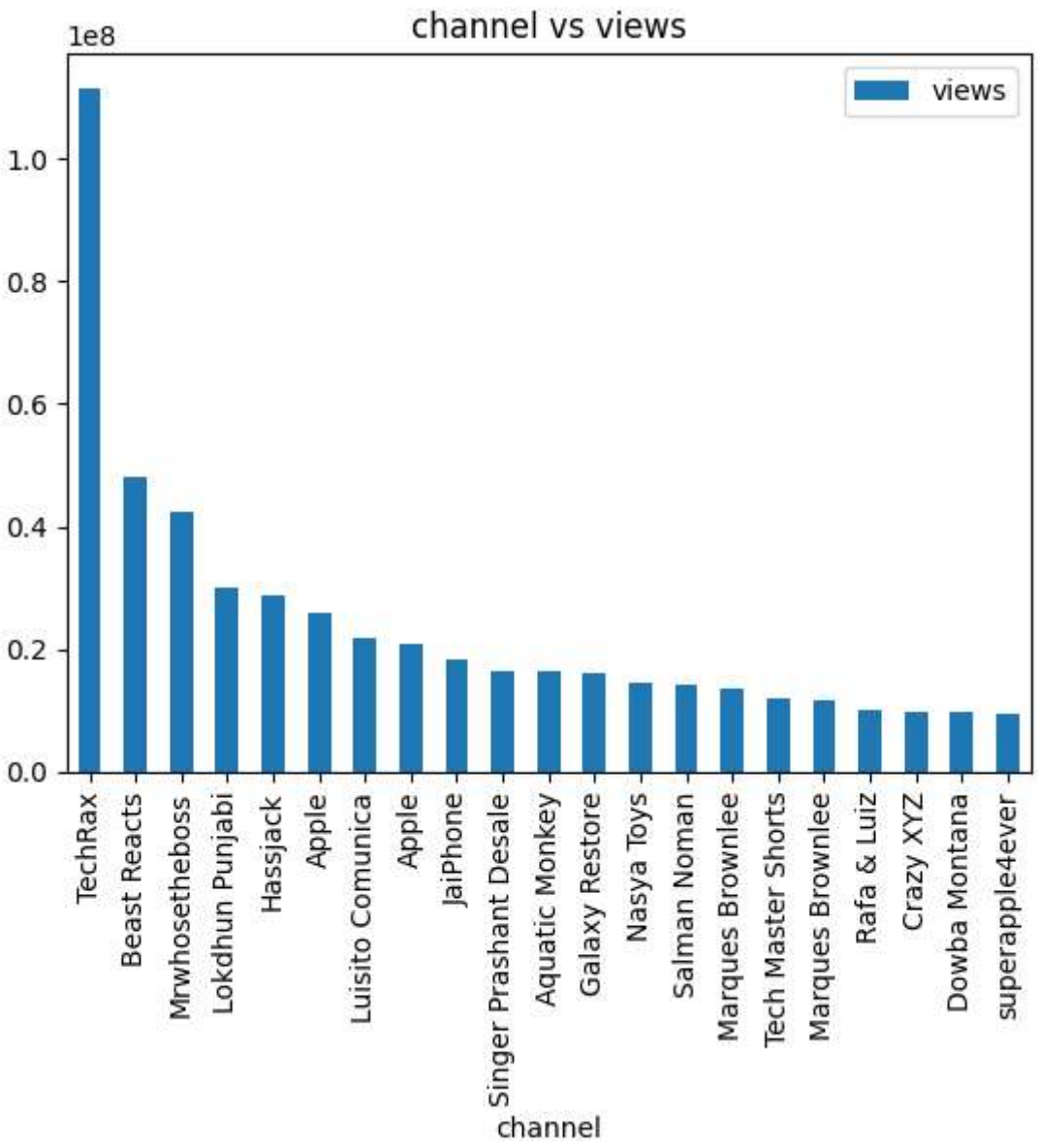


Figure9: Bar chart of views vs channels whose video appears in the RDD

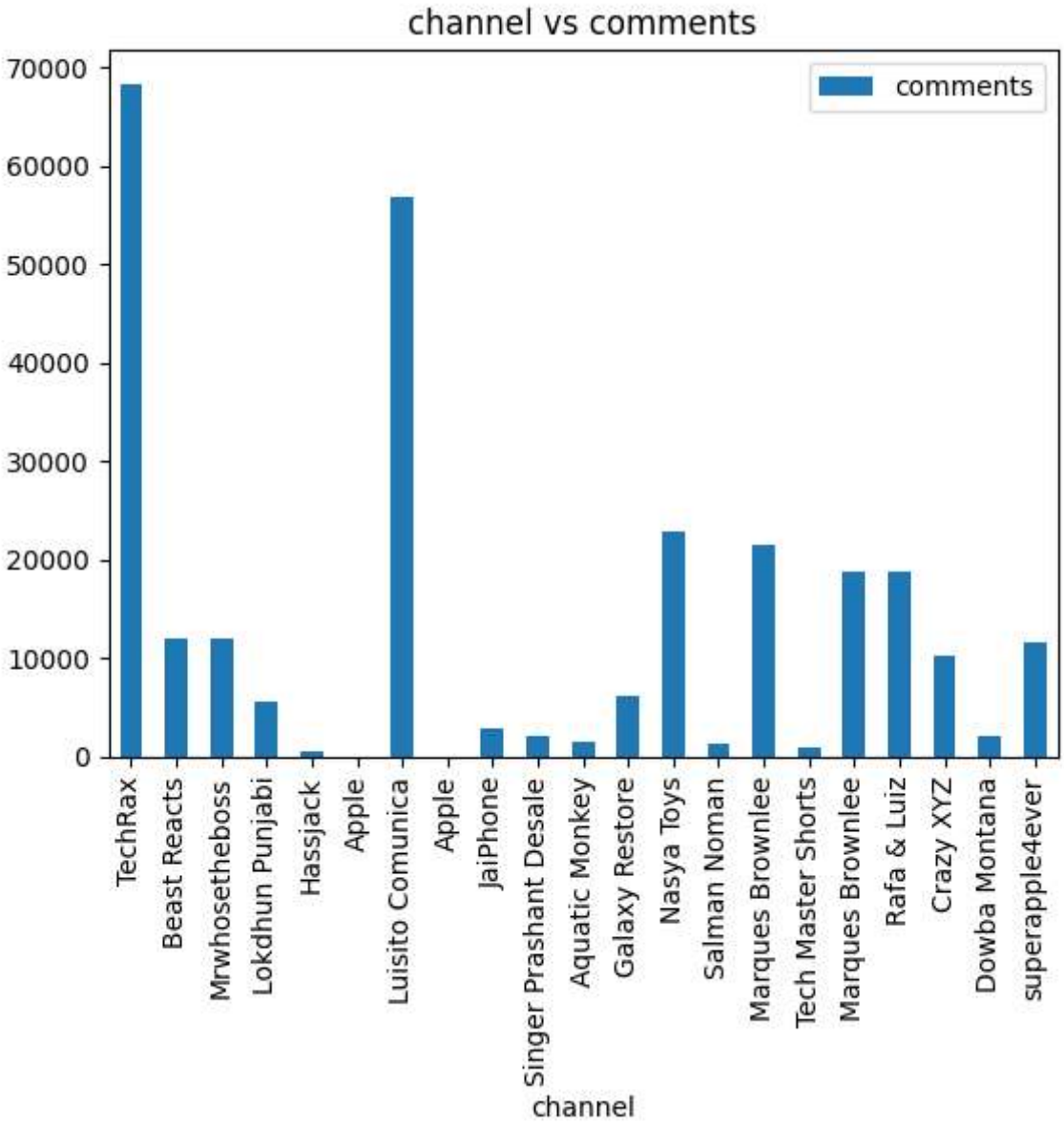

channel vs comments
comments
70000
60000
50000
40000
30000
20000
10000
0
TechRax
Beast Reacts
Mrwhosetheboss
Lokdhun Punjabi
Hassjack
Apple
Luisito Comunica
Apple
JaiPhone
Singer Prashant Desale
Aquatic Monkey
Galaxy Restore
Nasya Toys
Salman Noman
Marques Brownlee
Tech Master Shorts
Marques Brownlee
Rafa & Luiz
Crazy XYZ
Dowba Montana
superapple4ever
channel

Figure 10

VI. Use case:

We created a function named select_winner. It takes two parameters, 'Video_id', the video's id, and 'String', the keyword that must be in the comment. It retrieves the comments from the video using data API. The comments are retrieved using 'commentThreads().list()'. It takes all comments on the video. Then we filter the comments based on the keyword. Then we choose one comment randomly as the winner.

We can use this function to find winners of giveaways done by YouTubers.

VII. Streaming Data:

We want to extract real-time streaming data of comments for any particular youtube video. The average rate of new comments is shallow (one comment per 5 minutes), even in the videos with a considerably significant number of views.

Therefore, we can perform this exercise on trending video(s) to get the essence of streaming data. From the trending page, let us consider/take any one video and extract streaming data from it every 10 seconds.

VIII. Future work:

- In future work aspect, we can do sentiment analysis on the extracted comments.
- We can also develop an interactive dashboard to monitor each video metric.

IX. References:

- https://console.cloud.google.com/ (for accessing YouTube Data API v3)
- https://developers.google.com/youtube/v3 (for referring to the documentation of the API)
- https://spark.apache.org/docs/latest/rdd-programming-guide.html#rdd-operations (for referring to the PySpark RDD documentation)

XI. ADD-ON (Implementing Apache Kafka):

We tried implementing Kafka in Python, using "*!pip install kafka-python".* ***kafka-python*** is a popular Python client for Apache Kafka. It provides a simple and efficient way for Python programmers to interact with Kafka clusters.

After the installation of Kafka and JDK-8, we started…

(a). Zookeeper service using the following command

.\bin\windows\zookeeper-server-start.bat .\config\zookeeper.properties

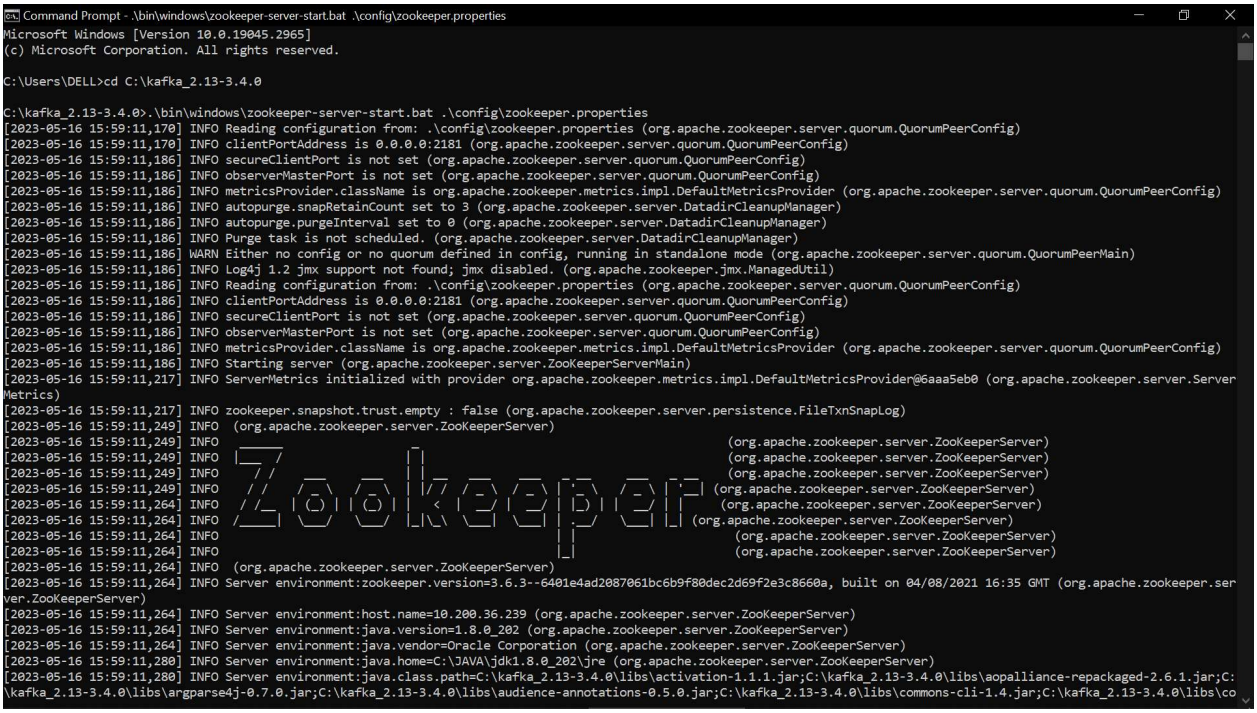


Figure 11: Zookeeper service

(b). Kafka service using the following command

.\bin\windows\kafka-server-start.bat .\config\server.properties

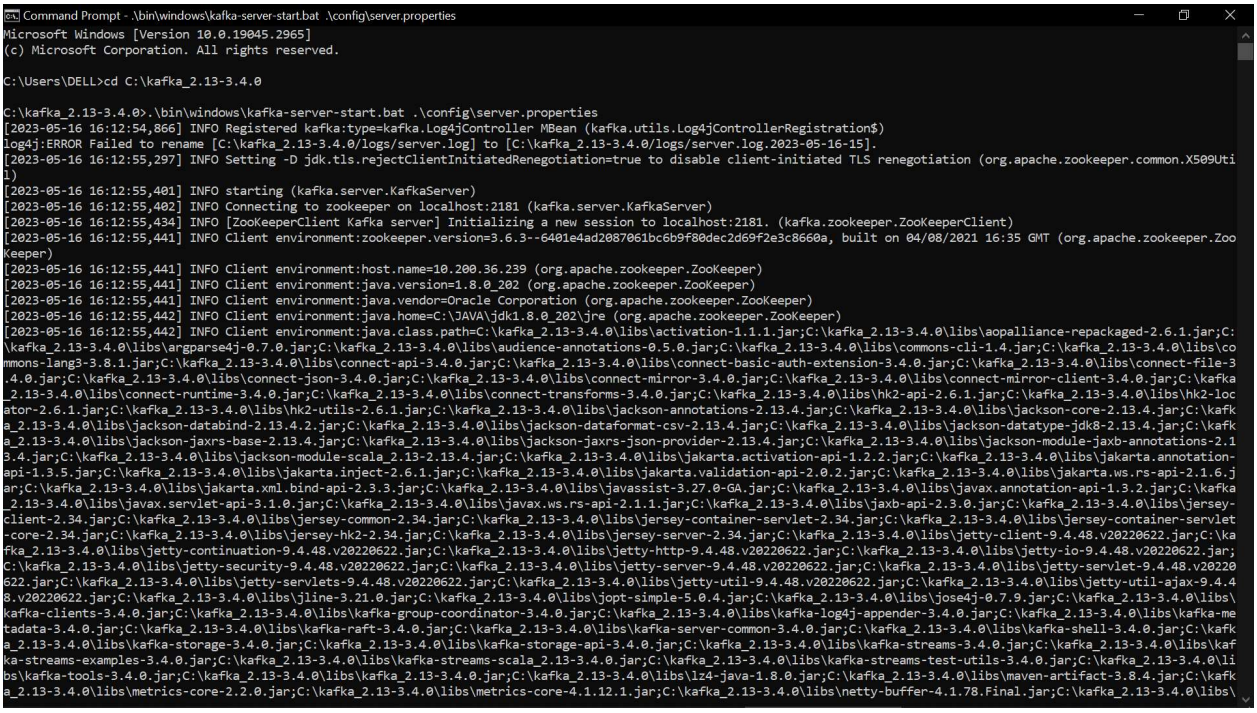


Figure 12: Kafka service

After completing this task, we created consumer and producer files in the Colab Notebook.

```python
from googleapiclient.discovery import build
from kafka import KafkaProducer
import time
import json

# Set up YouTube API client
API_KEY = 'AIzaSyCQfMHIQhOJThti7waxwOBfCd4jqHKoVMQ'
YOUTUBE_API_SERVICE_NAME = 'youtube'
YOUTUBE_API_VERSION = 'v3'
youtube = build(YOUTUBE_API_SERVICE_NAME, YOUTUBE_API_VERSION, developerKey=API_KEY)

# Set up Kafka producer
producer = KafkaProducer(bootstrap_servers='192.168.0.10:9092')

# Set up variables
video_id = 'cAMHx-m9oh8'
comments = []
prev_comments_count = 0

# Infinite loop to fetch comments every minute and send data to Kafka topic
while True:
    # Fetch comments for the video using YouTube API
    results = youtube.commentThreads().list(
        part='snippet',
        videoId=video_id,
        textFormat='plainText',
        order='time'
    ).execute()
    new_comments_count = results['pageInfo']['totalResults']

    # Calculate comments per minute and send data to Kafka topic
    current_time = int(time.time() * 1000)
    comments_per_min = (new_comments_count - prev_comments_count) / 1
    data = {
        'time': current_time,
        'comments_per_min': comments_per_min
    }
    producer.send('comment-count', json.dumps(data).encode('utf-8'))

    # Update variables for next iteration
    prev_comments_count = new_comments_count
    time.sleep(60) # Wait for 1 minute before fetching comments again
```

Figure 13: The above Python code snippet fetches comments from a YouTube video using the YouTube API, calculates the number of comments per minute, and sends the data to a Kafka topic every minute.

We could not implement Kafka in our notebook file because of the incompatibility between the Kafka version and Google Colab.

Despite having the Kafka broker created and configuring everything as per the requirement, it shows the below error and fails to produce the o/p

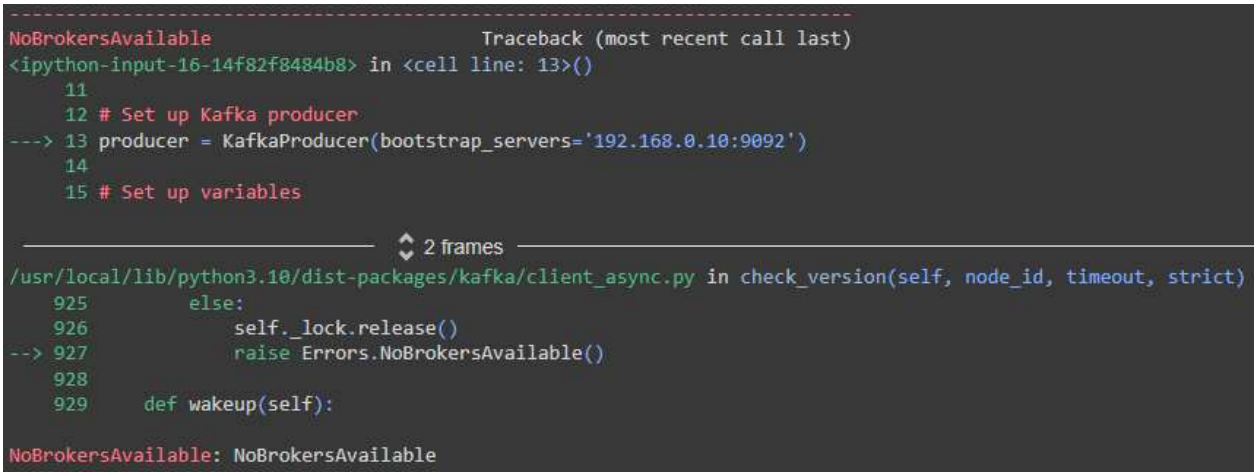


Figure 14:

LINK to our .ipynb file:

https://colab.research.google.com/drive/1ZX6aTWLgoAhUXxIpbRJN88-2Cu2Ypr8O?usp=sharing