

Project Report on Sentiment Analysis Platform.

DSA3030: Group 2 Members:

- Chesia Anyika
- Wayne Chilonje
- Mark Bilahi
- Ryan Rumanzi

I. Problem Domain

YouTube serves as a vast repository of user-generated content, spanning various genres such as entertainment, education, and advertising. This project aims to create a sentiment analysis system tailored for YouTube comments. By analyzing comments, this platform will help businesses gauge audience sentiment towards their content, products, or brand, thus empowering businesses to engage with their audience effectively and make informed decisions about content creation and marketing strategies.

II. Project Plan

A. Requirements Gathering

Social Media Platform Selection

We focused solely on Youtube due to its vast user base, vast repository of user generated content, real-time nature, and availability of comprehensive APIs for data access.

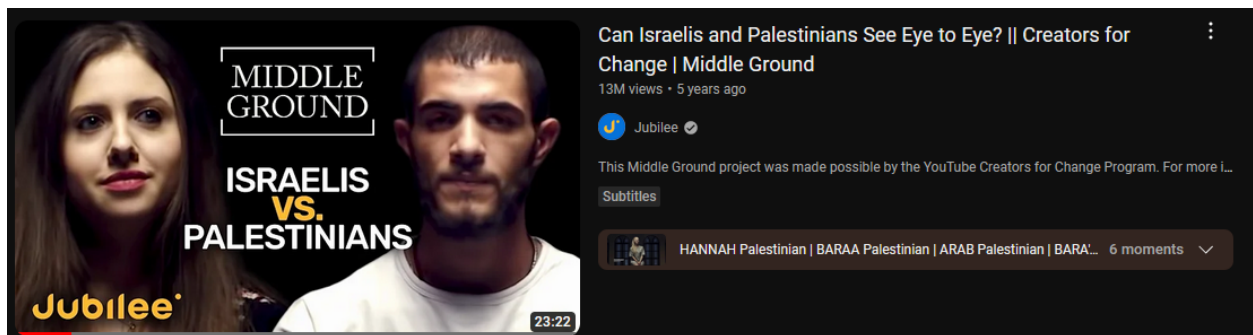
Feature Definition

We defined the key feature for sentiment analysis in our data as the verbatim comments users left on our chosen video. With that, we used natural language processing to determine comment sentiments as either positive, negative, or compound.

B. Data Ingestion and Preparation

Data Scraping

We utilised the Youtube API to collect real time comments from a specific video. The chosen video is titled '*Can Israelis and Palestinians See Eye to Eye? || Creators for Change | Middle Ground*', from the *Jubilee* youtube channel.



[Link to Video](#)

The video was chosen due to the following reasons:

- The video is pertinent to current social events, thus engagement on the video is high, having 13 million views and 100,000 comments to date. Thus, we are likely to get comments from a wide variety of youtube users.
- The video covers a highly divisive topic, thus comments are likely to showcase a wide variety of sentiments to train our model on.

The data scraping will be performed on Google Colab, and the data saved to a json file.

Streaming Data using integrated kafka-pyspark pipeline.

An Integrated Kafka-PySpark pipeline refers to the combination of Apache Kafka and PySpark, typically used for real-time data processing and analysis. Here's a breakdown:

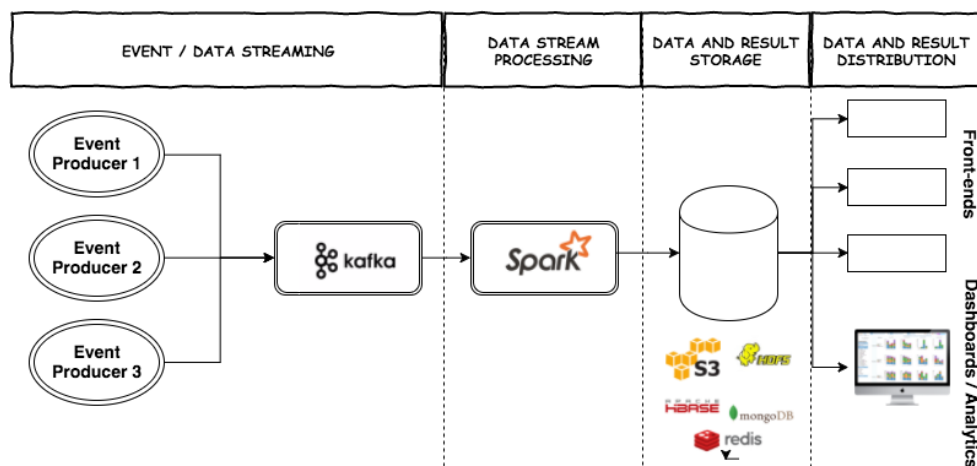
1. **Apache Kafka:** Kafka is a distributed streaming platform that is commonly used for building real-time data pipelines and streaming applications. It is designed to handle high-throughput, fault-tolerant, and scalable data streams. Kafka allows producers to publish data to topics and consumers to subscribe to these topics and process the data in real-time.
2. **PySpark:** PySpark is the Python API for Apache Spark, which is a powerful open-source framework for distributed data processing and analysis. PySpark allows you to write Spark applications using Python, enabling scalable and high-performance data processing, machine learning, and analytics.

A Kafka-PySpark integration typically involves the following steps:

- **Data ingestion:** Apache Kafka serves as the data source, where producers publish data to Kafka topics. This data can be logs, events, sensor readings, or any other streaming data.
- **Data processing:** PySpark is used to consume the data from Kafka topics, perform various transformations, analyses, and computations on the data streams. PySpark provides rich libraries and APIs for data manipulation, including SQL queries, machine learning, and graph processing.
- **Real-time analytics:** By integrating Kafka with PySpark, organizations can perform real-time analytics on streaming data. This includes tasks such as aggregations, filtering, enrichment, pattern detection, and anomaly detection, enabling businesses to gain insights and make data-driven decisions in real-time.
- **Scalability and fault-tolerance:** Both Kafka and Spark are designed for scalability and fault-tolerance, making them well-suited for handling large volumes of data and ensuring continuous processing even in the event of failures or network partitions.

There are several benefits of implementing Spark-Kafka integration. You can ensure minimum data loss through Spark Streaming while saving all the received Kafka data synchronously for an easy recovery. Users can read messages from a single topic or multiple Kafka topics.

Along with this level of flexibility you can also access high scalability, throughput and fault-tolerance and a range of other benefits by using Spark and Kafka in tandem. This integration can be understood with a data pipeline that functions in the methodology shown below:



This pipeline would allow us to stream the comments data scraped of the Youtube api for batch processing.

Creation of Docker containers

Docker is a containerization platform that allows developers to package applications application and its dependencies into a standardized unit called a container. Containers are great for *Continuous Integration and Continuous Delivery (CI/CD)* workflows, which involve frequently integrating code changes, early error detection and automating the deployment of code changes to production.

For the Kafka-PySpark pipeline:

- Using a Docker container ensures a consistent environment for running the Kafka and PySpark application across development, testing, and production.

- The Containers enable scalability, as multiple instances of Kafka and PySpark containers can be spun up easily to handle varying workloads.

Overall, the Docker containers will streamline the development, testing, and deployment of the Kafka-PySpark pipeline within a CI/CD workflow, providing consistency, reliability, and scalability.

Data Preprocessing

We will preprocess the collected data to remove noise, filter out duplicate comments and format it for analysis.

C. Application Development

Sentiment Analysis

We implemented sentiment analysis in Google Colab using *Natural Language Toolkit (NLTK)*. This is a leading platform for building Python programs to work with human language data. NLTK provides easy-to-use interfaces to over 50 corpora and lexical resources, such as WordNet, along with a suite of text processing libraries for tasks like tokenization, stemming, tagging, parsing, and semantic reasoning.

For the model, we used the *Valence Aware Dictionary and sEntiment Reasoner (VADER)* model which is a part of the NLTK library and is specifically designed for sentiment analysis of text. It's a lexicon and rule-based sentiment analysis tool that is particularly well-suited for analyzing sentiment in social media texts, online reviews, and other short, informal texts. we chose this model due to its high performance on informal texts that may contain emoticons, slang or other non-standard language features.

Visualisation

We integrated visualisation elements of our analysis using Matplotlib, to create visualisations of our sentiment analysis results.

D. Technologies and Tools Used

- Youtube API for data access.
- Docker Desktop for containerisation.
- Apache Kafka, Pyspark and Jupyter Notebook for data streaming.
- NLTK and VADER for sentiment analysis.
- Matplotlib for visualisation.
- Python programming language for implementation.

E. Deployment

Our project is deployed on Github, at the following link: <https://github.com/Chesia-Anyika/DSA3030-Project/blob/main/README.md>

The repository includes:

- The `docker-compose.yml` file used to create the docker images.
- The `kafka-python-streaming.ipynb` Jupyter Notebook code for streaming youtube comments.
- The `post_to_kafka.py` python code for posting streaming comments to kafka.
- The `YoutubeComments.ipynb` Colab Notebook where scraping comments and Sentiment analysis was conducted.

III. Project Execution

A. Data Scraping

We used the following code snippets in a Google Colab Notebook to scrape a number of youtube comments from the chosen video.

```
##import the necessary libraries
#for manipulation of comments into json format
```

```

import json
#for access to the youtube api
import googleapiclient.discovery
#for downloading and saving scraped comments
from google.colab import files

#specify youtube api credentials for access
api_service_name = "youtube"
api_version = "v3"
DEVELOPER_KEY = "AIzaSyD0yuJKSUf1XoRs220Q03ib-V6KsbCHIg8"

youtube = googleapiclient.discovery.build(
    api_service_name, api_version, developerKey=DEVELOPER_KEY)

#request a certain number of youtube comments
request = youtube.commentThreads().list(
    part="snippet",
    videoId="_Jj8vne0ca0", #This is the video ID for the chosen
    maxResults=100 #This is the maximum amount of comments we want
)
response = request.execute()

comments = []

#specify the schema of the data collected
for item in response['items']:
    comment = item['snippet']['topLevelComment']['snippet']
    comments.append({
        'author': comment['authorDisplayName'],
        'published_at': comment['publishedAt'],
        'updated_at': comment['updatedAt'],
        'like_count': comment['likeCount'],
        'text': comment['textDisplay']
    })

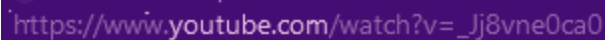
# Save comments as a JSON file

```

```
with open('comments.json', 'w') as json_file:
    json.dump(comments, json_file)

# Download the JSON file to your local machine
files.download('comments.json')
```

In the above code, we accessed the youtube api using our developer credentials, and scraped comments off of our chosen video using its unique video ID, which is `_Jj8vne0ca0`. This video id can be found at the end of a link to the youtube video, as `v = _Jj8vne0ca0`.



We then specify the schema for the comments scraped, and we specifically wanted:

- `author` - Shows the username of the commenter.
- `published_at` - shows the date and time that a comment was published.
- `updated_at` - represents the timestamp indicating when the YouTube comment was last updated or edited.
- `like_count` - Represents the total count of likes per comment.
- `text` - Represents the verbatim text of the comments themselves.

B. Streaming Data (Kafka-Pyspark Pipeline)

After obtaining the comments and saving them into a `JSON` file, we created an integrated kafka-pyspark pipeline using Docker, Command Prompt and Jupyter Notebook as follows.

1. Configuration of Docker Environment.

After downloading and running Docker Desktop, we used the `docker-compose.yaml` source file from [this github repository](#) to create the required docker containers for

our kafka-pyspark pipeline.

The following code snippet is run in the Command Prompt to create and compose the docker containers:

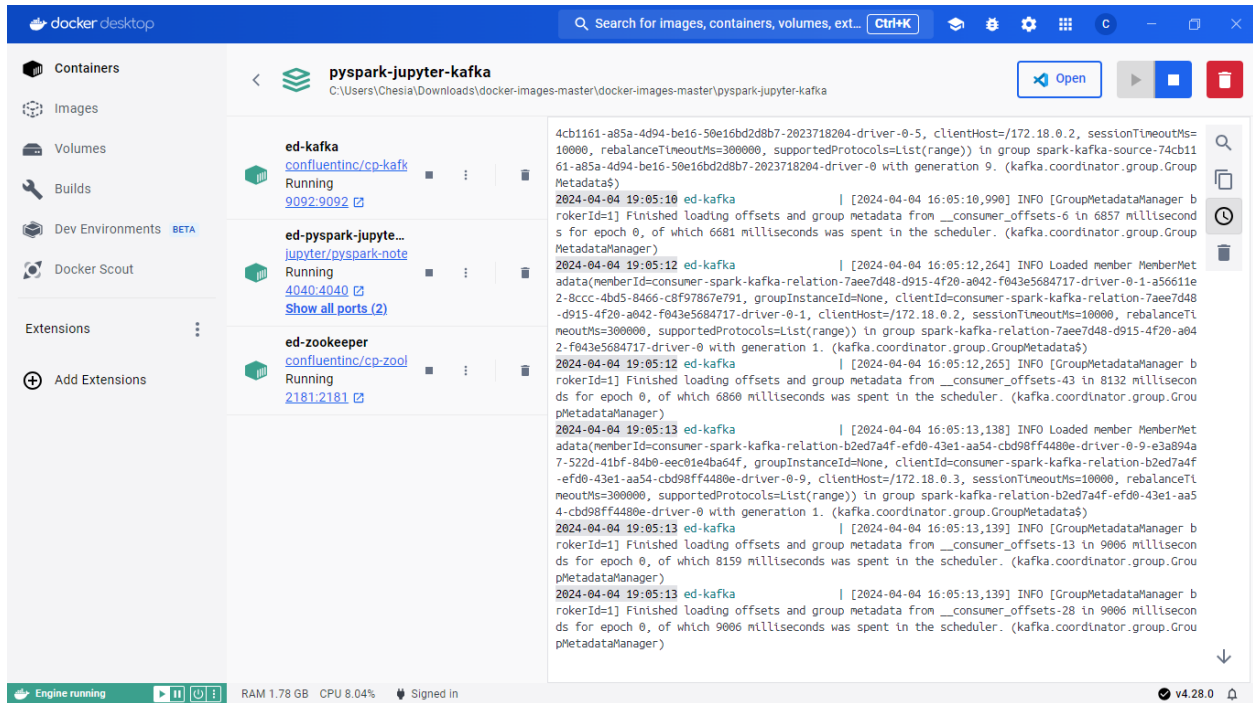
```
C:\Users\Chesia\Downloads\docker-images-master\docker-images-master>
```

The beginning of the output is as follows

```
Microsoft Windows [Version 10.0.22631.3296]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Chesia\Downloads\docker-images-master\docker-images-master\pyspark-jupyter-kafka>docker compose up
[+] Running 3/0
  ✓ Container ed-zookeeper          Created                                0.0s
  ✓ Container ed-pyspark-jupyter-lab Created                                0.0s
  ✓ Container ed-kafka              Created                                0.0s
Attaching to ed-kafka, ed-pyspark-jupyter-lab, ed-zookeeper
ed-pyspark-jupyter-lab | Entered start.sh with args: jupyter lab
ed-pyspark-jupyter-lab | Granting jovyan passwordless sudo rights!
ed-pyspark-jupyter-lab | /usr/local/bin/start.sh: running hooks in /usr/local/bin/before-notebook.d as uid / gid: 0 / 0
ed-zookeeper           | ====> User
ed-pyspark-jupyter-lab | /usr/local/bin/start.sh: running script /usr/local/bin/before-notebook.d/spark-config.sh
ed-zookeeper           | uid=1000(appuser) gid=1000(appuser) groups=1000(appuser)
ed-zookeeper           | ====> Configuring ...
ed-pyspark-jupyter-lab | /usr/local/bin/start.sh: done running hooks in /usr/local/bin/before-notebook.d
ed-pyspark-jupyter-lab | Running as jovyan: jupyter lab
ed-kafka               | ====> User
```

After composing the docker containers, our docker container for `pyspark-jupyter-kafka` looks like this:

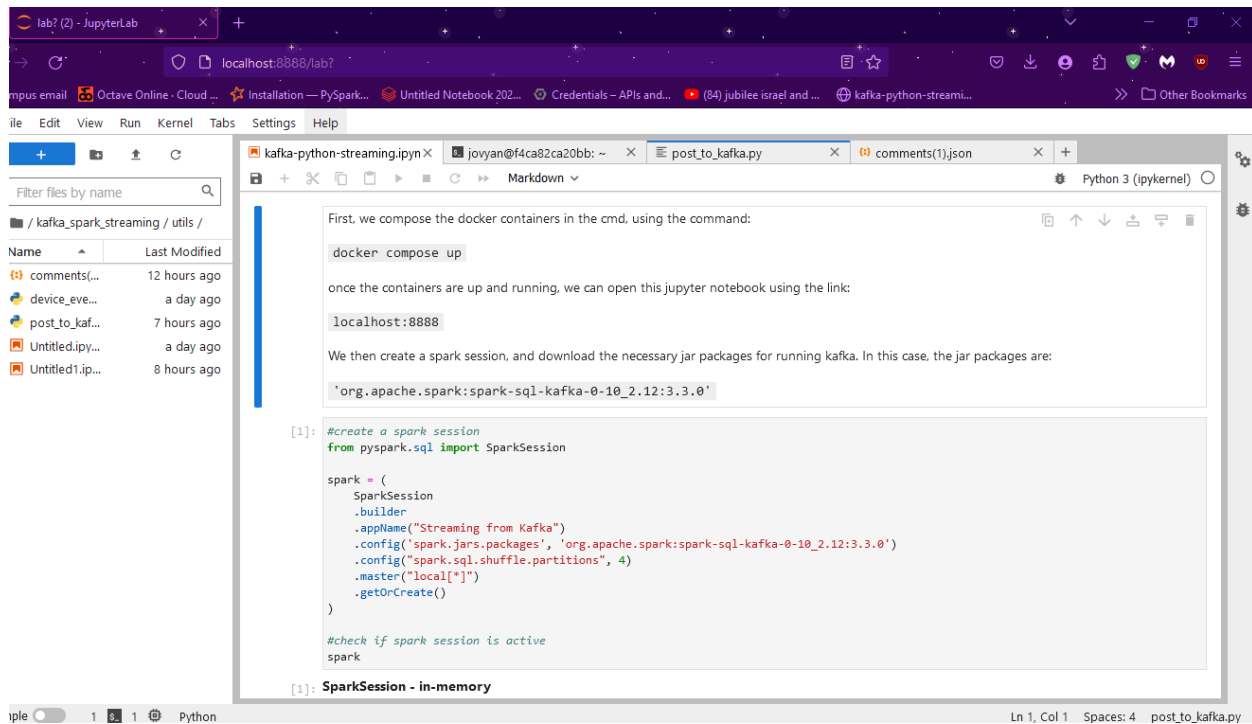


It has three sub-containers, namely:

- **ed-kafka** - This is a container that holds the kafka configuration, and we will be using it to hold the data we need for streaming.
- **ed-pyspark-jupyter-kafka** - This is a container that stores the configuration for jupyter notebook, specifically connected to this docker container and its files. We will use it to access jupyter notebook, where we will implement the streaming pipeline.
- **ed-zookeeper** - This container hosts Apache ZooKeeper, a vital service for coordinating distributed systems. Specifically for Kafka, it manages cluster state, broker coordination, leader election, and metadata, ensuring the smooth functioning of Kafka brokers and clients within the Docker environment.

2. Creation of Jupyter Notebook

After we have configured our docker containers, we can start up the jupyter notebook where we will set up the pipeline using the link localhost:8888. This is shown as follows:



3. Creation of SparkSession

With our jupyter notebook up and running, we can now use the following code to create a spark session:

```

#create a spark session
from pyspark.sql import SparkSession

spark = (
    SparkSession
    .builder
    .appName("Streaming from Kafka")
    .config('spark.jars.packages', 'org.apache.spark:spark-sql-kafka-0-10_2.12:3.3.0')
    .config("spark.sql.shuffle.partitions", 4)
    .master("local[*]")
    .getOrCreate()
)

```

```
#check if spark session is active
spark
```

Once we run this code snippet, the output will be as follows:

[1]: **SparkSession - in-memory**

SparkContext

[Spark UI](#)

Version

v3.3.0

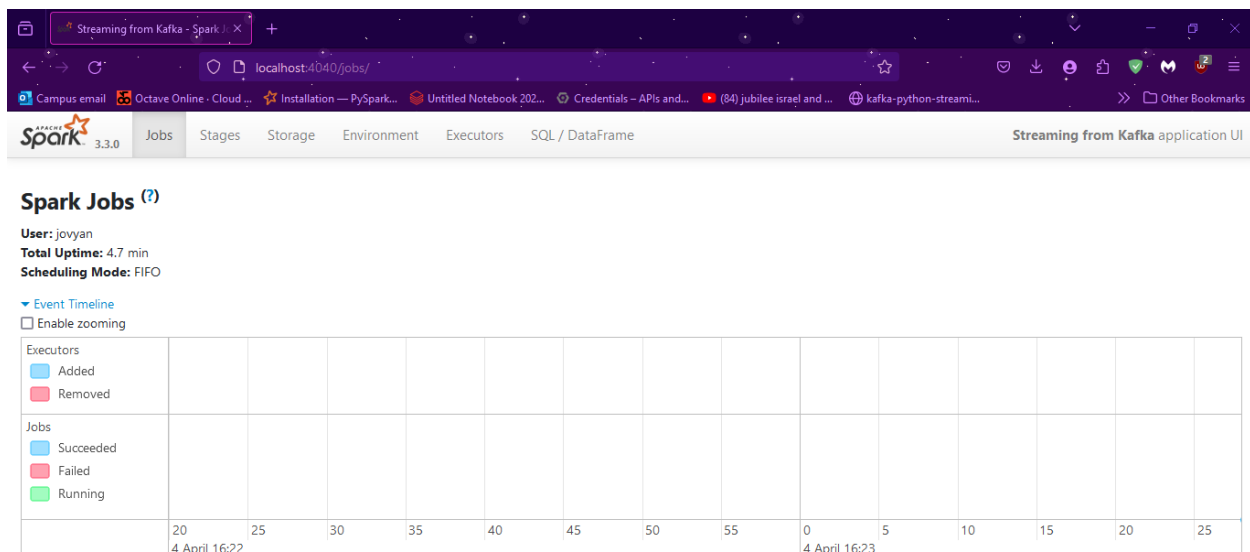
Master

local[*]

AppName

Streaming from Kafka

This indicates that our SparkSession is up and running, and we can access the Spark UI via the link localhost:4040 to verify that it is working. The page allows one to monitor the jobs being executed in the pipeline in a user friendly way, and looks as follows:



4. Configuration of ed-kafka container

Next we need to configure our kafka container so that it is ready to receive streaming data from our downloaded `comments.json` file.

We use the following series of command in a new command prompt to configure the kafka container:

```
#command to enter the kafka container, using the kafka container
docker exec -it abd99b4a9cf2a00a745dc72aa2358b375e09a38f25065020

#Command to create a new topic known as comments-data

kafka-topics --create --topic comments-data --bootstrap-server :

#Command to view if the topic has been created

kafka-topics --list --bootstrap-server localhost:29092

#Command to open up container for streaming of data

kafka-console-producer --topic comments-data --bootstrap-server
```

The output looks as follows:

```
@abd99b4a9cf2:~
Microsoft Windows [Version 10.0.22631.3296]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Chesia>docker exec -it abd99b4a9cf2a00a745dc72aa2358b375e09a38f2506502650fc67290577f466 /bin/bash
[appuser@abd99b4a9cf2 ~]$ kafka-topics --list --bootstrap-server localhost:29092
__consumer_offsets
comments-data
device-data
test-topict
tweet-data
[appuser@abd99b4a9cf2 ~]$ kafka-console-producer --topic comments-data -
--bootstrap-server localhost:29092
>
```

5. Creation of Kafka Dataframe Object

Next we will create a dataframe known as `kafka_df`, and specify its schema and data-types so as to use it as the data-frame to store the streaming data. The code is as follows:

```
[2]: #create the kafka_df to read from kafka

kafka_df = (
    spark
    .readStream
    .format("kafka")
    .option("kafka.bootstrap.servers", "ed-kafka:29092")
    .option("subscribe", "comments-data")
    .option("startingOffsets", "earliest")
    .load()
)

[3]: #View schema for raw kafka_df
kafka_df.printSchema()
#kafka_df.show()

root
|-- key: binary (nullable = true)
|-- value: binary (nullable = true)
|-- topic: string (nullable = true)
|-- partition: integer (nullable = true)
|-- offset: long (nullable = true)
|-- timestamp: timestamp (nullable = true)
|-- timestampType: integer (nullable = true)
```

```
[4]: #Parse value from binary to string into kafka_json_df
from pyspark.sql.functions import expr

kafka_json_df = kafka_df.withColumn("value", expr("cast(value as string)"))
```

```
[7]: #kafka_json_df.show()
```

key	value	topic	partition	offset	timestamp	timestampType
null	{"author": "@bish..."	comments-data	0	0	2024-04-04 03:35:...	0
null	comments-data	0	1	2024-04-04 03:35:...	0	
null	{"author": "@bish..."	comments-data	0	2	2024-04-04 03:36:...	0

We then specify the schema of the comments:

```
[5]: from pyspark.sql.types import StringType, StructField, StructType, ArrayType, LongType

json_schema = StructType([
    StructField('author', StringType(), True),
    StructField('published_at', StringType(), True),
    StructField('updated_at', StringType(), True),
    StructField('like_count', StringType(), True),
    StructField('text', StringType(), True)
])
```

```
#Apply the schema to payload to read data
from pyspark.sql.functions import from_json, col

streaming_df = kafka_json_df.withColumn("values_json", from_json(col("value"), json_schema)).selectExpr("values_json.*")
```

```
#To the schema of the data, place a sample json file and change readstream to read
streaming_df.printSchema()
#streaming_df.show(truncate=False)
```

```
root
|-- author: string (nullable = true)
|-- published_at: string (nullable = true)
|-- updated_at: string (nullable = true)
|-- like_count: string (nullable = true)
|-- text: string (nullable = true)
```

6. Streaming Data into the Kafka Container

We created a python code that periodically reads comments from our `comments.json` file and posts them to kafka. This python code is called `post_to_kafka.py` and is as follows:

```
from kafka import KafkaProducer
import time
import random
import uuid
```

```

__bootstrap_server = "ed-kafka:29092"

def post_to_kafka(data):
    producer = KafkaProducer(bootstrap_servers=__bootstrap_server)
    # Generate a unique key for each message
    key = str(uuid.uuid4()).encode('utf-16')
    # Convert data to bytes
    value = data.encode('utf-16')
    producer.send('comments-data', key=key, value=value)
    producer.flush()
    producer.close()
    print("Posted to Kafka")

if __name__ == "__main__":
    # Path to the JSON file
    json_file = "/home/jovyan/kafka_spark_streaming/utils/comments.json"

    while True:
        # Read data from the JSON file
        with open(json_file, 'r', encoding = 'utf-8') as f:
            data = f.read()

        # Post data to Kafka
        post_to_kafka(data)
        print(data)

        # Sleep for some time before reading the file again
        time.sleep(5)

```

We used the following commands in the Jupyter Notebook Terminal to run the above code:


```
#access the post_to_kafka.py file via its directory
python /home/jovyan/kafka_spark_streaming/utils/post_to_kafka.py
```

[illegible]

We then use the following `writeStream()` command to write our streamed data into the docker environment:

```
#write the output to console sink to check the output- batch mode
(streaming_df
```

```

.writeStream
.format("console")
.outputMode('append')
.trigger(processingTime = '10 seconds')
.option("checkpointLocation", "checkpoint_dir_kafka2")
.start()
.awaitTermination()

```

C. Sentiment Analysis

We conducted Sentiment analysis in Google Colab, using the VADER model in the NLTK library.

1. Exploratory Data Analysis

We began with some exploratory data analysis, to understand the structure of the comments we would be analysing.

▼ Exploratory Data Analysis

```

✓ [4] import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# using ggplot style plots
plt.style.use('ggplot')

# natural language toolkit
import nltk

```

```

✓ [6] # type of data being used
df['text'].values[2]

'you can see that except for the israeli guy everyone wants peace'

```

```

✓ [7] df.shape

(100, 5)

```

```

**Description**

```

there are a total of `100` comments with `5` columns`

The key columns are the `text` and the `author` columns

The text column represents the comments posted. The following is

```

you can see that except for the israeli guy everyone wants peace

```

We then checked the data for any row-by-row duplicates to avoid skewing our analysis by using duplicate data.

✓ Checking for row-by-row duplicates

The following code checks for duplicates in the speech dataset

```
✓ [9] #define function that checks for row by row duplicates
0s def has_row_duplicates(df):
    seen_rows = set()
    for row in df:
        row_tuple = tuple(row) # Convert the row to a tuple since lists are unhashable
        if row_tuple in seen_rows:
            return True # Found a duplicate row
        seen_rows.add(row_tuple)
    return False # No duplicate rows found

#View results
print(has_row_duplicates(df))
```

False

The function returns `False`, indicating that there are no row-by-row duplicates in the dataset that need to be cleaned. As there are also no null values in the dataset, the data is clean and can be used for the NLP analysis.

2. Preliminary Analysis of Dataset

We did a Preliminary analysis of the `like_count` variable to determine the distribution of likes per comment.

✓ Analysis of distribution of likes

I omitted the first comment in the dataframe which is from the creator themselves, and has a highly skewed number of likes and would not add to the analysis.

```
[10] # Assuming df is your DataFrame
      comments = df.drop(df.index[0]).reset_index(drop=True)
```

```
[11] comments.head()
```

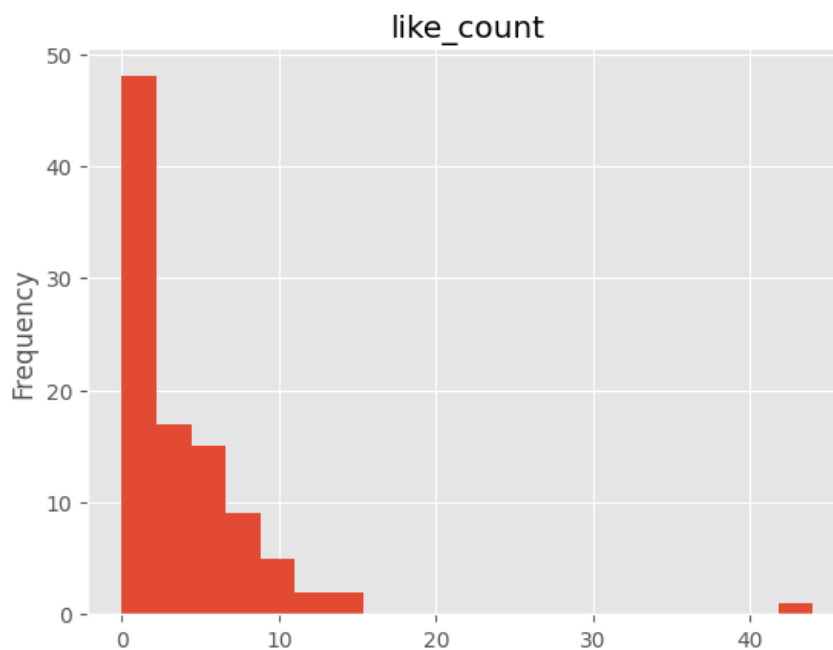
	author	published_at	updated_at	like_count	text
0	@salwanawaid5176	2024-04-04T07:55:45Z	2024-04-04T07:55:45Z	0	<a href="https://www.youtube.com/watch?v=_Jj8v...
1	@SanaAsiya-gt2lj	2024-04-04T07:22:31Z	2024-04-04T07:22:31Z	0	you can see that except for the israeli guy ev...
2	@afnan707	2024-04-04T02:50:33Z	2024-04-04T02:50:33Z	0	FREE FREE PALESTINE
3	@bishno6229	2024-04-03T00:33:45Z	2024-04-03T00:33:45Z	1	FREE PALESTINE FROM THE RIVER TO THE SEA 🇵🇸🇵🇸🇵🇸
4	@EliseUrmom	2024-04-02T23:55:56Z	2024-04-02T23:55:56Z	0	I could not deal with ran

Next steps: [Generate code with comments](#) [View recommended plots](#)

✓ Histogram displaying distribution of like_count variable

```
[12] # @title Histogram displaying distribution of like_count variable

from matplotlib import pyplot as plt
comments['like_count'].plot(kind='hist', bins=20, title='like_count')
plt.gca().spines[['top', 'right']].set_visible(False)
```



The Analysis shows that the distribution of the like_count variable is right-skewed, with a majority of comments having 0 likes, and clustered between having 0 to 10 likes. There are a few outliers, which have over 40 likes.

3. Basic NLTK

Before performing Sentiment analysis, we wanted to display a basic understanding of how the NLTK package works.

- **Tokenisation**

Tokenization in Natural Language Processing (NLP) is the process of breaking down a text into smaller units, typically words or subwords, called tokens. These tokens serve as the basic building blocks for further NLP tasks such as parsing, text analysis, and machine learning.

In the below code, we stored a sample comment into the `example` variable, and tokenised it as follows:

✓ Basic NLTK

```
✓ [14] # example
0s      example = comments['text'][1]
      print(example)

      you can see that except for the israeli guy everyone wants peace

✓ [15] nltk.download('punkt')
2s

[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
True
```

```
✓ # using the nltk to split the sentence into individual words
5 tokens = nltk.word_tokenize(example)
tokens[:30]
```

```
➞ ['you',
    'can',
    'see',
    'that',
    'except',
    'for',
    'the',
    'israeli',
    'guy',
    'everyone',
    'wants',
    'peace']
```

- **Perceptron Tagging**

Perceptron tagging is a technique used in natural language processing for part-of-speech tagging. It involves training a perceptron, a type of neural network, to predict the part of speech of each word in a given text based on its context and features such as surrounding words, prefixes, and suffixes. The perceptron learns from labeled training data and iteratively adjusts its weights to improve its accuracy in predicting part-of-speech tags.

We Used the following standard perceptron tags to tag our tokenised comment:

```
## Tags and Abbreviations
|Abbreviation|Meaning|
|:--|--:|
|CC|coordinating conjunction|
|CD|cardinal digit|
|DT|determiner|
|EX|existential there|
|FW foreign word
|IN|preposition/subordinating conjunction|
|JJ|This NLTK POS Tag is an adjective (large)|
|JJR|adjective, comparative (larger)|
```

JJS	adjective, superlative (largest)
LS	list market
MD	modal (could, will)
NN	noun, singular (cat, tree)
NNS	noun plural (desks)
NNP	proper noun, singular (sarah)
NNPS	proper noun, plural (indians or americans)
PDT	predeterminer (all, both, half)
POS	possessive ending (parent\ 's)
PRP	personal pronoun (hers, herself, him, himself)
PRP\$	possessive pronoun (her, his, mine, my, our)
RB	adverb (occasionally, swiftly)
RBR	adverb, comparative (greater)
RBS	adverb, superlative (biggest)
RP	particle (about)
TO	infinite marker (to)
UH	interjection (goodbye)
VB	verb (ask)
VBG	verb gerund (judging)
VBD	verb past tense (pleaded)
VBN	verb past participle (reunified)
VBP	verb, present tense not 3rd person singular(wrap)
VBZ	verb, present tense with 3rd person singular (bases)
WDT	wh-determiner (that, what)
WP	wh- pronoun (who)
WRB	wh- adverb (how)

The Perceptron tagged comment is as follows:

```
✓ [17] nltk.download('averaged_perceptron_tagger')
0s
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data] /root/nltk_data...
[nltk_data] Unzipping taggers/averaged_perceptron_tagger.zip.
True
```

```
✓ [18] # finding the part of speech for each token
0s
tagged = nltk.pos_tag(tokens)
tagged[:30]
```

```
[('you', 'PRP'),
 ('can', 'MD'),
 ('see', 'VB'),
 ('that', 'DT'),
 ('except', 'IN'),
 ('for', 'IN'),
 ('the', 'DT'),
 ('israeli', 'JJ'),
 ('guy', 'NN'),
 ('everyone', 'NN'),
 ('wants', 'VBZ'),
 ('peace', 'NN')]
```

- **Named Entity Recognition (NER)**

Named Entity Recognition (NER) is a natural language processing task that involves identifying and categorizing named entities (such as persons, organizations, locations, etc.) mentioned in text into predefined categories. At this stage we chunk words which have been tagged the same into the same category.

We implemented this on our sample comment as follows:


```
✓ 1s [19] nltk.download('maxent_ne_chunker')

[nltk_data] Downloading package maxent_ne_chunker to
[nltk_data] /root/nltk_data...
[nltk_data] Unzipping chunkers/maxent_ne_chunker.zip.
True

✓ 0s [20] nltk.download('words')

[nltk_data] Downloading package words to /root/nltk_data...
[nltk_data] Unzipping corpora/words.zip.
True

✓ 9s [21] pip install svgling

Collecting svgling
  Downloading svgling-0.4.0-py3-none-any.whl (23 kB)
Collecting svgwrite (from svgling)
  Downloading svgwrite-1.4.3-py3-none-any.whl (67 kB)
  67.1/67.1 kB 2.8 MB/s eta 0:00:00
Installing collected packages: svgwrite, svgling
Successfully installed svgling-0.4.0 svgwrite-1.4.3

✓ 0s # group the data into chunks based on the tags

entities = nltk.chunk.ne_chunk(tagged)
entities.pprint()

(S
  you/PRP
  can/MD
  see/VB
  that/DT
  except/IN
  for/IN
  the/DT
  israeli/JJ
  guy/NN
  everyone/NN
  wants/VBZ
  peace/NN)
```

There were no duplicated tags, thus the comments individual words remain in their unique groups.

4. Example VADER Sentiment Scoring

We then further demonstrated examples of Vader Sentiment Scoring using:

- An obviously positive statement - 'I am so happy!'
- An obviously negative statement - 'Monday is the worst day ever!'
- Our example statement - 'you can see that except for the Israeli guy everyone wants peace'

This is illustrated as follows:

```
✓ [23] nltk.download('vader_lexicon')
0s
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
True
```

```
✓ [24] from nltk.sentiment import SentimentIntensityAnalyzer
1s
from tqdm.notebook import tqdm # progress bar tracker

sia = SentimentIntensityAnalyzer()
```

```
✓ [25] # example implementation
0s
sia.polarity_scores('I am so happy!')

{'neg': 0.0, 'neu': 0.318, 'pos': 0.682, 'compound': 0.6468}
```

```
✓ [26] sia.polarity_scores('Monday is the worst day ever!')
0s

{'neg': 0.468, 'neu': 0.532, 'pos': 0.0, 'compound': -0.6588}
```

```
✓ [27] # on the initial text example
0s
sia.polarity_scores(example)

{'neg': 0.0, 'neu': 0.759, 'pos': 0.241, 'compound': 0.5423}
```

Interpretation

The vader model will categorise statements on a scale of -1 to 1, and the score will be under the compound category. Scores greater than 0.5 indicate a positive categorisation, while scores less than -0.5 indicate a negative categorisation. Scores between -0.5 and 0.5 are categorised as neutral.

- The obviously positive statement 'I am so happy!' has a compound score of 0.6468, indicating that the vader model correctly categorised it as positive.
- The obviously negative statement 'Monday is the worst day ever!' has a compound score of -0.6588, indicating that the vader model correctly categorised it as negative.
- Our example statement 'you can see that except for the Israeli guy everyone wants peace' has a compound score of 0.5423, indicating that the vader model has categorised it as neutral, which intuitively is correct, as it is a statement of observation.

5. Sentiment Analysis on the Whole dataset

We then tokenised, tagged, chunked and applied the vader model to every comment in the whole dataset.

the example statement is mostly negative as the compound score value is -0.5209

```
✓ [28] # running polarity score on the whole data set
0s

# creating a smaller data frame for analysis

#df = covidData.head(1000)
df2 = comments.reset_index().rename(columns= {'index': 'id'})

df2.shape

(99, 6)

✓ [29] # holds the polarity score of the comment by each user and the user name
0s
results = {}

# perform the polarity scoring and show progres in a progress bar below
for i, row in tqdm(df2.iterrows(), total = len(df2)):
    text = row['text']
    index = row['id']
    results[index] = sia.polarity_scores(text)
```

100%  99/99 [00:00<00:00, 1083.79it/s]

```

✓ [30] results
1s
8: {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0},
9: {'neg': 0.119, 'neu': 0.595, 'pos': 0.286, 'compound': 0.4767},
10: {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0},
11: {'neg': 0.108, 'neu': 0.892, 'pos': 0.0, 'compound': -0.1779},
12: {'neg': 0.0, 'neu': 0.488, 'pos': 0.512, 'compound': 0.2732},
13: {'neg': 0.064, 'neu': 0.784, 'pos': 0.152, 'compound': 0.8869},
14: {'neg': 0.0, 'neu': 0.815, 'pos': 0.185, 'compound': 0.3612},
15: {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0},
16: {'neg': 0.15, 'neu': 0.85, 'pos': 0.0, 'compound': -0.5209},
17: {'neg': 0.0, 'neu': 0.7, 'pos': 0.3, 'compound': 0.7717},
18: {'neg': 0.289, 'neu': 0.711, 'pos': 0.0, 'compound': -0.8377},
19: {'neg': 0.333, 'neu': 0.569, 'pos': 0.098, 'compound': -0.5994},
20: {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0},
21: {'neg': 0.145, 'neu': 0.652, 'pos': 0.203, 'compound': 0.1553},
22: {'neg': 0.152, 'neu': 0.848, 'pos': 0.0, 'compound': -0.3252},
23: {'neg': 0.0, 'neu': 0.381, 'pos': 0.619, 'compound': 0.7579},
24: {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0},
25: {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0},
26: {'neg': 0.0, 'neu': 0.323, 'pos': 0.677, 'compound': 0.4939},
27: {'neg': 0.0, 'neu': 0.375, 'pos': 0.625, 'compound': 0.7177},
28: {'neg': 0.342, 'neu': 0.658, 'pos': 0.0, 'compound': -0.7783},
29: {'neg': 0.0, 'neu': 0.717, 'pos': 0.283, 'compound': 0.9652},
30: {'neg': 0.167, 'neu': 0.714, 'pos': 0.119, 'compound': -0.25},
31: {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0},
32: {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0},
33: {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0},
34: {'neg': 0.0, 'neu': 0.476, 'pos': 0.524, 'compound': 0.5106},
35: {'neg': 0.155, 'neu': 0.725, 'pos': 0.121, 'compound': -0.1779},

```

We then transformed the obtained results into a pandas data-frame for easier data manipulation, and concatenated it with the original dataset to compare their VADER scores with the tweets themselves. This is shown as follows:

```

[32] # transpose the dictionary
vaders_df = vaders_df.reset_index().rename(columns= {'index': 'id'})
vaders_df = pd.concat([vaders_df, comments], axis = 1) # add the original dataframe to the left

```

```

[33] vaders_df

```

	id	neg	neu	pos	compound	author	published_at	updated_at	like_count	text
0	0	0.000	1.000	0.000	0.0000	@salwanawaid5176	2024-04-04T07:55:45Z	2024-04-04T07:55:45Z	0	<a href="https://www.youtube.com/watch?v=_Jl8v...
1	1	0.000	0.759	0.241	0.5423	@SanaAsiya-gt2ij	2024-04-04T07:22:31Z	2024-04-04T07:22:31Z	0	you can see that except for the israeli guy ev...
2	2	0.000	0.132	0.868	0.7650	@afnan707	2024-04-04T02:50:33Z	2024-04-04T02:50:33Z	0	FREE FREE PALESTINE
3	3	0.000	0.665	0.335	0.6166	@bishno6229	2024-04-03T00:33:45Z	2024-04-03T00:33:45Z	1	FREE PALESTINE FROM THE RIVER TO THE SEA 🇵🇸🇵🇸🇵🇸
4	4	0.000	1.000	0.000	0.0000	@EliseUrmom	2024-04-02T23:55:56Z	2024-04-02T23:55:56Z	0	I could not deal with ran
...
94	94	0.000	1.000	0.000	0.0000	@user-tp6kr4fw2k	2024-03-18T08:42:29Z	2024-03-18T08:42:29Z	3	Jubilee, we need update about these people.
95	95	0.000	0.526	0.474	0.4019	@antonyschwarz8749	2024-03-18T08:16:23Z	2024-03-18T08:16:23Z	0	Lord Jesys help us
96	96	0.406	0.326	0.269	-0.3987	@antonyschwarz8749	2024-03-18T08:15:45Z	2024-03-18T08:15:45Z	2	Free Palestine from Evil Hamas !!
97	97	0.000	1.000	0.000	0.0000	@hajaranouar3394	2024-03-18T00:46:36Z	2024-03-18T00:46:36Z	2	🇵🇸🇵🇸
98	98	0.000	1.000	0.000	0.0000	@meimyself8739	2024-03-17T19:19:35Z	2024-03-17T19:19:45Z	6	it's been 5 years since this video imagine if ...

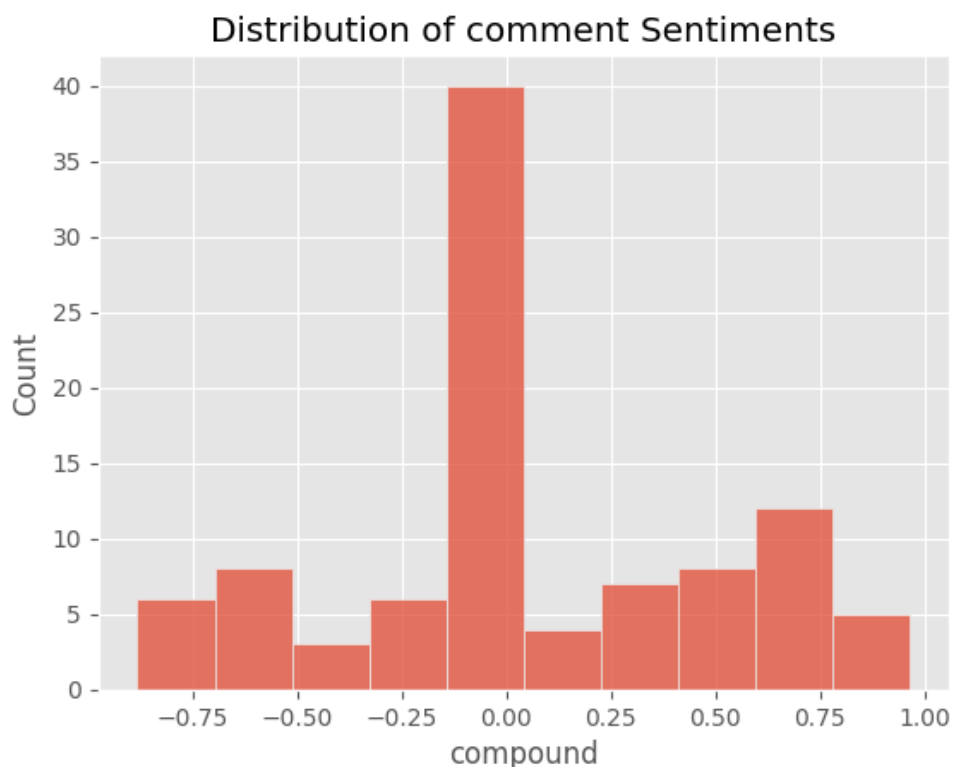
99 rows × 10 columns

6. Visualisations and Further Analysis

We then created some visualisations to display the distribution and percentage proportion of comment sentiments, as well as distribution of comment sentiments by average like_count.

a. Distribution of Comment Sentiments - Histogram

```
✓ [34] vader_plot = sns.histplot(data= vaders_df, x = 'compound')  
js      vader_plot.set_title("Distribution of comment Sentiments")  
      plt.show()
```



Interpretation

The above Histogram shows the distribution of sentiments analysed. It shows that majority of sentiments are neutral, with 40 counts having a 0 compound score. There are on average more positive sentiments as compared to negative sentiments, with more comments falling above 0.5 than below -0.5. This is surprising, due to the divisive nature of the topic at hand in the video.

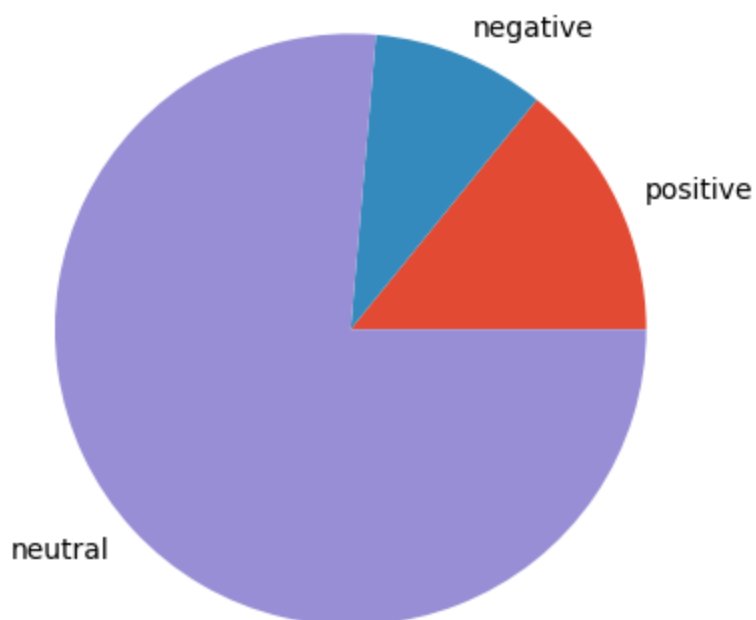
b. Percentage proportion of Comment Sentiments - Pie Chart

```
[39] # comment distribution as percentages
vaders_pos = vaders_df['pos'].mean()
vaders_neg = vaders_df['neg'].mean()
vaders_neu = vaders_df['neu'].mean()




vader_distrib = {'sentiment': ['positive', 'negative', 'neutral'],
                 'value' : [vaders_pos, vaders_neg, vaders_neu]}
vaders_perc_distrib = pd.DataFrame(vader_distrib)
plt.pie(vaders_perc_distrib['value'],
        labels = vaders_perc_distrib['sentiment'])
plt.title("Sentimet Distribution")
plt.show()
```



Sentimet Distribution



```
[40] vaders_perc_distrib
```

	sentiment	value	
0	positive	0.138707	
1	negative	0.092818	
2	neutral	0.748303	

Interpretation

The above pie-chart shows the percentage distribution of sentiments per category. As evidenced by the histogram, the largest percentage is taken up by neutral comments at 74.83%, followed by positive comments at 13.87% and finally negative comments at 9.28%.

c. Distribution of Likes per sentiment

We then analysed the Distribution of likes per sentiment by grouping comments as follows:

- Compound score ≤ 0.6 is positive.
- Compound score ≥ -0.6 is negative.
- Compound score $-0.6 \leq x \leq 0.6$ are neutral.

We visualised the average like counts per category in a barplot as follows:



```
# Define function to categorize compound scores
def categorize_sentiment(compound):
    if compound >= 0.6:
        return 'Positive'
    elif compound <= -0.6:
        return 'Negative'
    else:
        return 'Neutral'

# Apply sentiment categorization
vaders_df['sentiment'] = vaders_df['compound'].apply(categorize_sentiment)

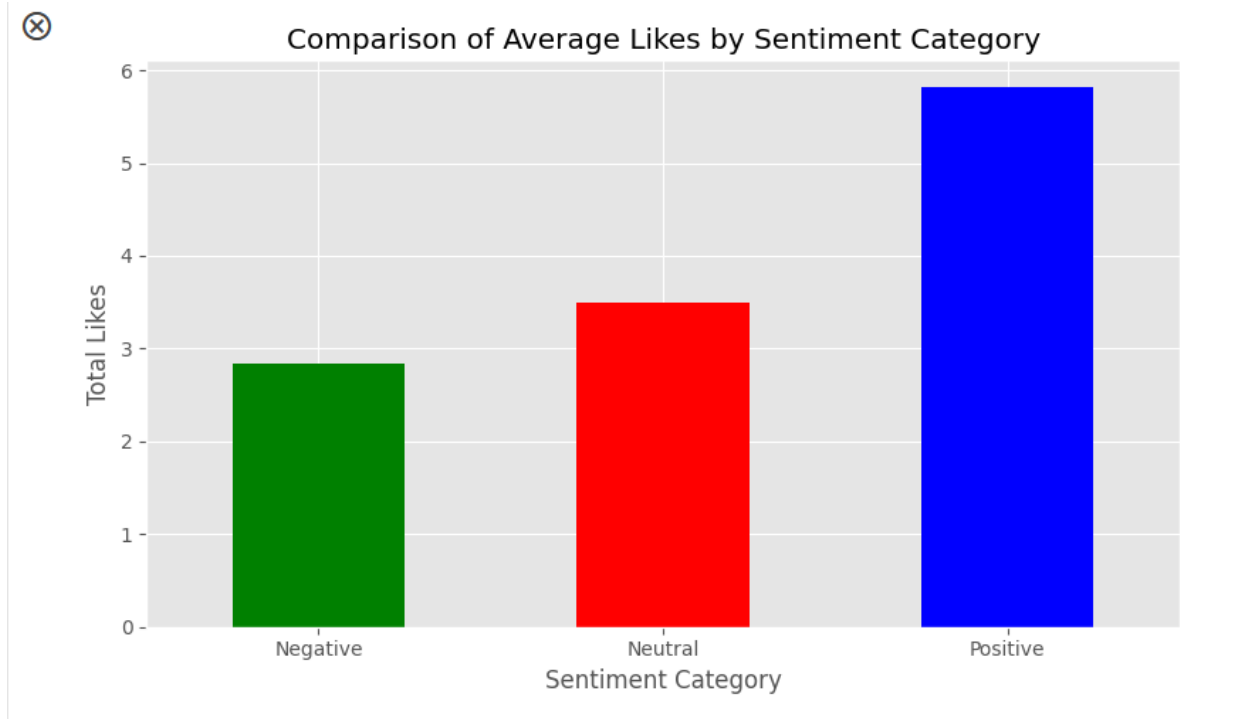
# Aggregate average like counts for each sentiment category
average_likes = vaders_df.groupby('sentiment')['like_count'].mean()

# Plotting
plt.figure(figsize=(8, 5))

# Bar plot
average_likes.plot(kind='bar', color=['green', 'red', 'blue'])

# Adding labels and title
plt.xlabel('Sentiment Category')
plt.ylabel('Total Likes')
plt.title('Comparison of Average Likes by Sentiment Category')

# Show plot
plt.xticks(rotation=0)
plt.tight_layout()
plt.show()
```

The visualisation shows that the positive comments have the highest average like counts, followed by neutral comments then negative comments. This suggests that even with a topic this divisive, positive sentiments were appreciated more by watchers than negative comments.

IV. Conclusion

Our application allows for streaming of real time youtube comments, sentiment analysis and visualisation of key metrics such as distribution of sentiments ranging from positive, to neutral and negative, percentage proportion of sentiments per category and distribution of like counts per category.

This application would be beneficial to various stakeholders, including:

1. **Content Creators/YouTubers:** They can gain valuable insights into the sentiment of their audience in real-time, allowing them to understand how their content is being received and make adjustments if necessary. For example, if they notice a high proportion of negative comments, they may reconsider their content strategy or address issues raised by viewers.

2. **Marketing Professionals:** They can monitor the sentiment of comments related to their brand or products on YouTube. This information can help them gauge the effectiveness of their marketing campaigns, identify areas for improvement, and engage with their audience more effectively.
3. **Social Media Managers:** They can use the sentiment analysis and visualization tools to track the sentiment of comments on their company's YouTube channel. This information can inform their social media strategies and help them maintain a positive brand image by addressing any negative sentiment promptly.
4. **Researchers and Analysts:** They can use the application to study trends in sentiment across different YouTube channels, topics, or time periods. This can provide valuable insights into public opinion, societal trends, and consumer behavior.

Overall, this application provides valuable insights into audience sentiment on YouTube, helping stakeholders make informed decisions, improve engagement, and maintain a positive online presence.