DATA ANALYTICS AND VISUALIZATION PROJECT

ANALYSIS AND VISUALIZING THE IMPACT OF G20 AND MODELLING THE REACTION ON THE DATA SCRAPPED FROM THE TWITTER AND YOUTUBE.

AYUSH KUMAR MISHRA, AKSHAT KUMAR,

ADITYA PRAKASH

| DATA ANALYTICS AND VISUALIZATION|

CONTENTS :

ABSTRACT:

This project aims to delve into public sentiment during the G20 Summit in Delhi, employing data collected from two major social media platforms: Twitter and YouTube. Through a systematic approach of data scraping, preprocessing, sentiment analysis, modeling, and visualization, the goal is to uncover insights into how the public perceives and reacts to this significant international event.

PROBLEM DESCRIPTION:

"Analysis and visualizing the impact of G20 and modelling the reaction on the data scrapped from the Twitter and Youtube ."

Our goal is to Model and Visualize the reactions and comments of the people during the G20 submit in the Delhi.G20 is held annually and this year this held in INDIA. So many world leaders came to India and so many things happened and all news outlets cover the main headlines.

But what about people voice i.e how people take this event?

What are their reactions? Is this generally positive or negative ? Lets work over this.

DOMAIN:

NLP, WEB-SCRAPPING, VISUALIZATION AND ANALYTICS

WORKFLOW For YOUTUBE Data:

->DATA COLLECTION :

We first need DATA during G20 period but as there is no proper already organized data available in the internet, we have to scrap the data from web.

The required tools for Scrapping the youtube data :

a) Selenium
b) Python(version>=3.60)
c) Chromedriver
d) urllib

Process of scapping data from youtube:

1. Setting Up the Environment:

I'm importing essential libraries like pandas, numpy, and Selenium to work with data and automate web interactions. Then, I'm configuring the Chrome web driver with specific options for headless browsing. Adding ('--incognito') chrome-option to the chrome

driver is important as we don't want our our search history to cause any problem our program.

2. Defining Functions:

I've created a function called get_links that uses Selenium to search for a given item in a list on YouTube by passing given query in search query.

```
query = urllib.parse.quote(item)
url = "https://www.youtube.com/results?search_query=" + query
```

.This function scrolls through the results, gathers video titles and links, and makes it easy to collect data.

3. Data Collection from YouTube:

Now, I've listed some search items related to the G20 summit.For each item, I'm using the get_links function to collect video titles and links. These are stored in DataFrames for each item and then combined into a final DataFrame final_df. Below is the ss of the final_df.

	Item	Video Title	Links
80	DELHI-SUMMIT 2023	G20 Summit 2023: US President Joe Biden to arr...	https://www.youtube.com/watch?v=P8qDdKKp8gY&pp...
49	INDIA-G20	LIVE: Amid Israel-Hamas War, Mob Storms Russia...	https://www.youtube.com/watch?v=BVjtvIGDjm0&pp...
84	DELHI-SUMMIT 2023	G20 Summit 2023: US President Joe Biden Lands ...	https://www.youtube.com/watch?v=G4gqQgjNv0A&pp...
23	G-20	Highlights from the P20 Summit	https://www.youtube.com/watch?v=-boSXvXIDjc&pp...
50	INDIA-G20	Why Maldives' New President Wants Indian Troop...	https://www.youtube.com/watch?v=MuKPepZXuJE&pp...
29	G-20	G20 Summit Delhi: Bilateral meeting between PM...	https://www.youtube.com/shorts/oLYcl1MeiOA
19	G20 2023	Highlights Day-1 of the G20 Summit | Bharat Ma...	https://www.youtube.com/watch?v=GHsSldQROvo&pp...
46	INDIA-G20	G20 Summit 2023: India's Modi chairs summit of...	https://www.youtube.com/watch?v=F1pUDSue8mQ&pp...
60	INDIA-G20	How India is clearing and hiding away slums as...	https://www.youtube.com/watch?v=Grspbh_1piA&pp...
78	DELHI-SUMMIT 2023	Sansad TV Special I North East India : Gateway ...	https://www.youtube.com/watch?v=iFyqgJsCELU&pp...

4. Scraping Video Information and Comments:

Next, I've set up a dictionary called videos_dictionary to store video information, comments, links, and associated items. There's a function called get_query_from_url to extract search queries from YouTube links.

The scrap function takes a video URL and an index as input, extracts video title, item, and comments, and updates the videos_dictionary.

It takes the link from final_df dataframe and open in incognito mode chrome and scrape all the comments by finding its `x-path`and do it until document.Element.Scrollheight became equal to the current scrapping element height. So In this way we collect all the commets of a video currently in the argument of the function scrap(url,i).

5. Checking Existing Data:

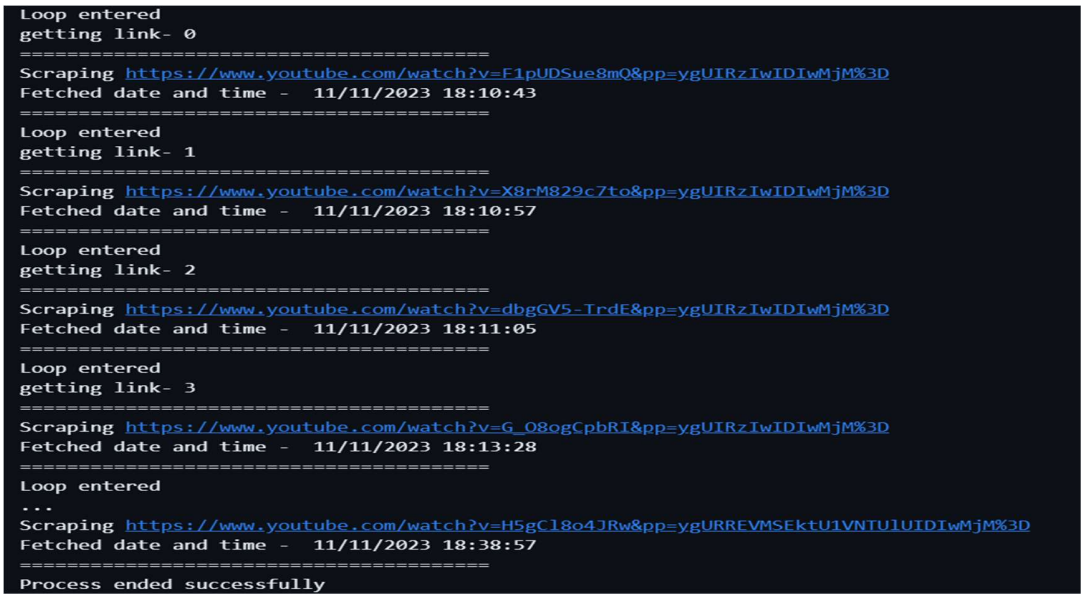
The code checks whether a CSV file named 'Youtube_scrapping_comments.csv' exists. If it does, it reads the file to determine where to start scraping. If not, it creates a new file. Check the file exist to save the

data that has been scrapped from the links. If the file does not exist, then create one. If it exist then check whether it is empty or not.If not empty, then extract data from the links not present in the file.

```
File exist.
Reading the file now
File is empty
Starting from position -  0
```

6. Scraping Loop:

Now, here comes the exciting part. I'm iterating through the video links in final_df. For each video link, I'm calling the scrap function to get video information and comments, and updating the videos dictonary. Clearly we can see how our function is entering each link and scrap and show the date&time and at last showing at last our process ended successfully.



7. Creating the Final DataFrame:

Finally, I'm creating a DataFrame named data from the collected information stored in the videos_dictionary.

```
data = pd.DataFrame.from_dict(videos_dictionary)
data
```

8. Output:

The script concludes by giving me a final DataFrame (data) containing crucial information about video titles, links, comments, and associated items.

```
data= pd.read_csv('Youtube_scrapping_comments.csv')
```

and then we stored our dataframe into ``Youtube_scrapping_comments.csv``.

In essence, this code automates the process of searching for G20-related videos on YouTube, extracting relevant information, and scraping comments for analysis. It uses Selenium for web scraping and pandas for efficient data manipulation.

->DATA ANALYSIS & VISUALISATION(YoutubeData)

Ok now I have to do data analysis and visualization part. So workflow for this is as follows:

1)Library import

The analysis begins with the importation of essential libraries for data manipulation, numerical operations, plotting, and machine learning tasks and I also added ignore warning code line to not make output long.

2) Reading Data:

The YouTube comments data is read from a CSV file into a pandas DataFrame I made in the scrapping file

named ``Youtube_scrapping_comments.csv``.Below is the ss of the dataframe.

	Video Title	Cleaned_Comments	item
21	G-4 | G-6 | G-7 | G-8 | G-10 | G-11 | G-20 | D...	- **GK & GS Brahmastra (ब्रह्मास्त्र) Batch By** ...	G-20
36	PM Modi's speech at public meeting in Hyderaba...	@AaswadamRecipes - Vasudhaiva Kutumbakam, whic...	G-20
28	World leaders arrive at the Bharat Mandapam in...	@mjfansumit - I proud that at this event I wor...	G-20
41	Gravitas: Russia stockpiles for winter attacks...	@lastChang - Xi is not attending because he is...	G-20
37	India To Turn New Delhi Into Fortress For G20 ...	@Luvallo - What about congested unruly traffic...	G-20

3)Data Summary:

A concise summary of the dataset is generated, including descriptive statistics and information about data types by calling desc()

	Video Title	Cleaned_Comments	item
count	49	49	49
unique	38	45	3
top	World leaders arrive at the Bharat Mandapam in...	@mjfansumit - I proud that at this event I wor...	G20 2023
freq	3	2	21

and info().

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 49 entries, 0 to 48
Data columns (total 3 columns):
 #   Column            Non-Null Count  Dtype 
---  ------            --------------  ----- 
 0   Video Title       49 non-null     object
 1   Cleaned_Comments  49 non-null     object
 2   item              49 non-null     object
dtypes: object(3)
memory usage: 1.3+ KB
```

4)Handling Null Values:

Null values within the dataset are scrutinized, and the count of null values for each column is inspected. Remedial actions are then taken to address any null values, contributing to a more refined dataset. As we see above our has no null values(lucky!!) so no need to apply `dropna()`.

5)Data Cleaning:

Rows containing null values are systematically removed, yielding a dataset that has undergone cleaning for subsequent analysis.

a) first we remove all the links
b) then remove all the emojis
c) Remove all '@username' format from our comments

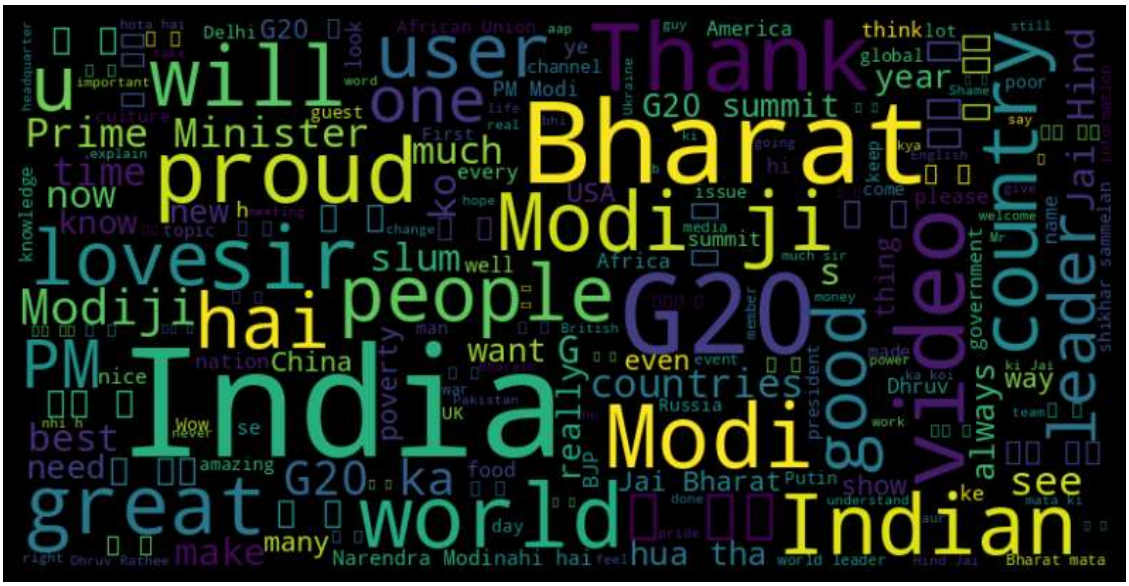
So after doing all these step our data looks like this:

	Video Title	Cleaned_Comments	item
3	G-4 | G-6 | G-7 | G-8 | G-10 | G-11 | G-20 | D...	- The list of invitees / participants is very...	G20 2023
18	Turkish Parliament Boycotts Coca Cola and Nest...	- Very much proud to see Bharat taking such l...	G20 2023
14	G20 Summit | Partnership for Global Infrastruc...	- Yes, not only do we have an extraordinary P...	G20 2023
1	PM Modi addresses The Hindustan Times Leadersh...	- I proud of you modi sir ....I love my INDIA...	G20 2023
32	India booms, China cools: What does it mean fo...	- As an Indian watching DW videos and followi...	G-20

6)Visualization - Count of Items:

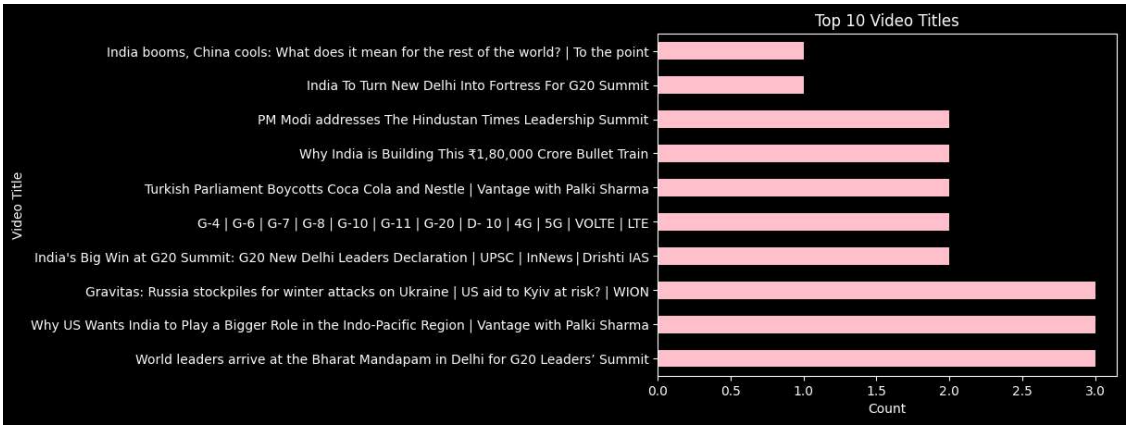
Seaborn and Matplotlib are employed to craft a count plot, visually portraying the distribution of items in the dataset. This plot serves as an insightful representation, elucidating the number of comments associated with each item and facilitating a nuanced understanding of the dataset's composition.

a)First we plot the most common words – We have to import wordcloud library .It will show most common words used in our entire corpus(it is the combination of all possible text in the data).



b) Plotting the top 10 most videos title

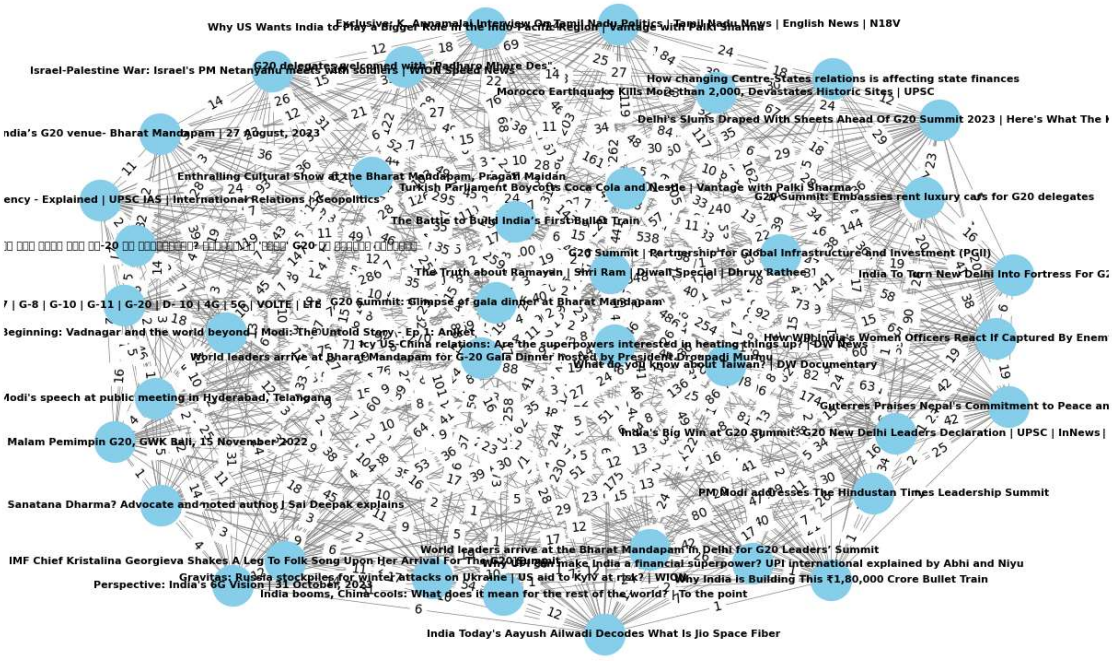
As we plot a bar-plot of 10 most occurring video titles in our dataframe.



C) Plotting the length of comments

We plot the length of the comments by items i.e ['G20','G20 2023','India G20'] and observe most lengthy commets is in 'G20 2023' item.

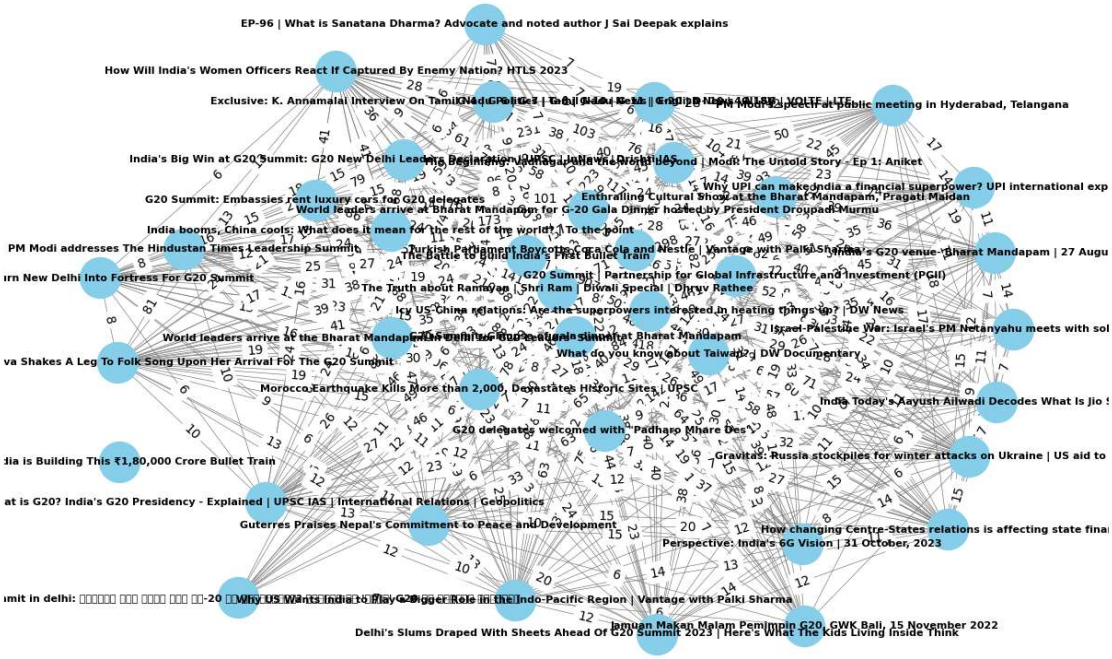
d)Network analysis- Now we want the relative order and relation between the text of the our data. For this very purpose we can use network graph. From this graph we can valuely take insight our data text is distributed and present.



Obviously we can not see overall picture of relation between the text in our data but we can definetly see how semantically close sentences are closer to each to other. Here distance show the semantically closeness to each other.

e)Plotting the relationship using network-graph but only among most common sentences:

What if we only plot previous plot using a filter that returns most occurring text? I think that would present a different picture i.e how close most of the text in the data is sema



ntically close to one another.

ANALYSIS PART:

As our data is very extensive and large in length and text. So it is not feasible for one to understand the meaning and ‘what is data overall??’ question which I think is important before doing any modelling as it presents a picture of what to do and in case somewhere there comes a irregularity in the results we can improve or correct as we know about our data well. This gives edge over someone who is blindly applying any model.

So for analysis part we will do the following steps:

1)CONTENT SUMMARISATION:

Steps for summarizing the corpus of our data:

Setting up My Summarizer:

I've got this cool tool in my toolkit called the BERT Summarizer. It's like having a super-smart assistant that can understand language really well.

Getting to Work:

```
from summarizer import Summarizer
bert_summarizer = Summarizer()
df['Cleaned_Comments'] = df['Cleaned_Comments'].apply(lambda x: re.sub(r"@[\w]+", "", x))
df['Summary'] = df['Cleaned_Comments'].apply(lambda x: bert_summarizer(x))
df.head()
```

Now, I'm putting my BERT Summarizer to work. I want it to read through all my cleaned comments and give me short, to-the-point summaries for each of them.

Creating a Space for Summaries:

I decided to create a new column in my DataFrame called 'Summary' to store these concise summaries. It's like having a dedicated section for the quick takeaways from each comment.

Checking out the Results:

Curious to see how well my assistant performed, I'm printing out the first few rows of my DataFrame. This way, I can quickly scan through the original cleaned comments alongside their freshly generated summaries.

In a nutshell, I've automated the process of summarizing my comments using BERT, making it much easier for me to get a handle on what's being said without diving into the details of each comment. It's like having a helpful assistant that condenses information for me!

1. Time Efficiency:

Benefit: Summarization allows users to quickly grasp the main points of a document without reading the entire content.

Example: Professionals can save time by quickly understanding the key insights of lengthy reports or articles.

2. Information Retrieval:

Benefit: Summarized content enhances the efficiency of information retrieval systems.

Example: Search engines can display concise summaries, enabling users to find relevant information more quickly.

3. Effective Communication:

Benefit: Summaries facilitate effective communication by conveying essential information concisely.

Example: Business executives can use summaries to communicate key findings and strategies to their teams.

4. Decision-Making Support:

Benefit: Summarized content aids decision-making by providing a quick overview of critical details.

Example: Executives can make informed decisions by reviewing summarized reports before diving into the full documents.

5. Scalability:

Benefit: Content summarization supports scalability by processing large volumes of information.

Example: News aggregation platforms can efficiently handle a vast number of articles by providing condensed summaries.

6. Learning and Education:

Benefit: Summarized educational content helps students grasp complex topics more easily.

Example: Students can use summaries to review and understand key concepts before exams.

7. Content Skimming:

Benefit: Summarization enables users to quickly skim through diverse content.

Example: Social media platforms can provide brief summaries of articles, encouraging users to engage with the content.

8. Multi-Document Summarization:

Benefit: Summarizing multiple documents provides a consolidated view, aiding comprehensive understanding.

Example: Researchers can quickly assess multiple papers on a topic by reviewing summarized versions.

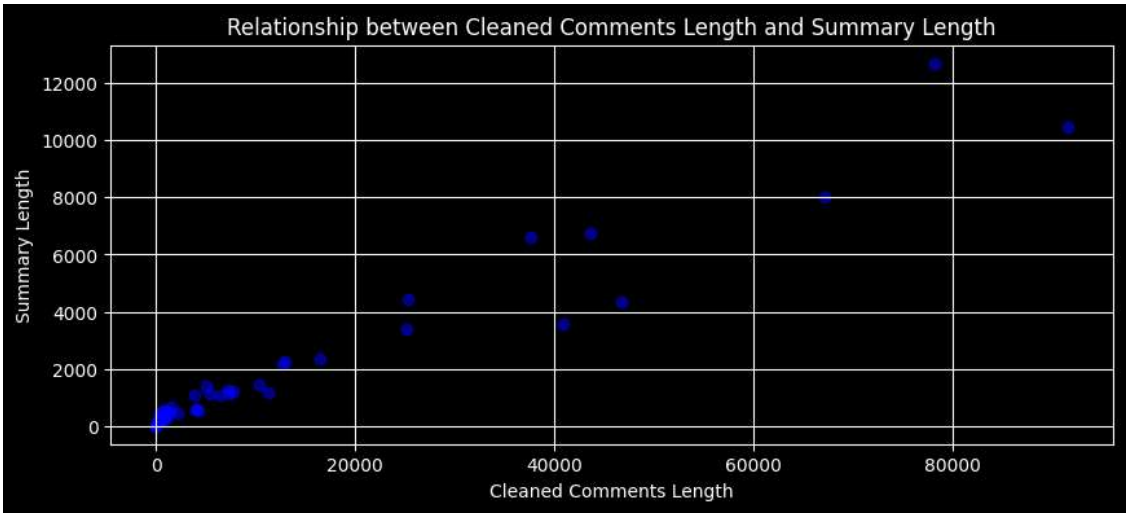
9. Language Translation:

Benefit: Summarization assists in translating content while maintaining essential meaning.

Example: Translators can provide concise translations that retain the original document's key information.

In conclusion, content summarization is a valuable tool that enhances efficiency, aids decision-making, and facilitates effective communication across various fields and applications. The ability to distill information into concise summaries is particularly beneficial in today's information-rich digital age.

One more thing to see here to which length our bert-model is compressing to. For this purpose, I plotted a graph between comment's length and its summary length.



Time taken- 45 mins(approx) and in the above plot we can see summary has 1/10 size of original text.

We can tell how much time consuming our model is

2)CONTENT-CATEGORISATION:

Using k-means approach

What I'm doing here is using a TF-IDF (Term Frequency-Inverse Document Frequency) vectorizer to process a bunch of cleaned comments. Let me break it down for you:

```
# Create a TF-IDF vectorizer
tfidf_vectorizer = TfidfVectorizer()
tfidf_matrix = tfidf_vectorizer.fit_transform(df['Cleaned_Comments'])
num_clusters = 3
kmeans = KMeans(n_clusters=num_clusters, random_state=42)
df['Cluster'] = kmeans.fit_predict(tfidf_matrix)
df.head()
```

Setting Up My Vectorizer:

First things first, I've got this TF-IDF vectorizer in play. It's like a language wizard that's going to convert my comments into numerical vectors, taking into account the importance of words in each comment compared to the entire dataset.

Creating a Matrix of TF-IDF Features:

Now, I'm applying this wizard, the TF-IDF vectorizer, to my cleaned comments. The result? A matrix of TF-IDF features. Think of it as a fancy table where each row corresponds to a comment, and the columns

represent unique words, capturing their significance in each comment.

Time to Cluster:

Next up, I'm putting on my clustering hat. I've decided to categorize these comments into clusters using the K-Means algorithm. It's like grouping similar comments together in a way that makes sense.

How Many Clusters, You Ask?

Good question! I've decided to go with three clusters. So, K-Means is going to analyze the TF-IDF features and sort these comments into three distinct groups based on their similarities.

The Final Touch: Adding Cluster Labels:

After the clustering magic happens, I'm adding a new column to my DataFrame called 'Cluster'. Each comment is now labeled with a cluster number, indicating which group it belongs to.

Let's Take a Peek:

Summary	Cluster
I proud that at this event I worked whole da...	2
I proud of you modi sir ....I love my INDIA ...	1
a. 1975 \n\nb. 1999 \n\nc. 2008 \n - First G20...	1
- The list of invitees / participants is very ...	1
- Very much proud to see Bharat taking such le...	2

I'm super excited to see the results, so I'm printing out the first few rows of my DataFrame. You'll notice that each comment now has a cluster label, showing how they've been grouped together based on their content.

In a nutshell, with just a few lines of code, I've transformed my comments into numerical representations, grouped them into clusters, and labeled each comment accordingly. It's like having an automated assistant that can organize and categorize comments for me!

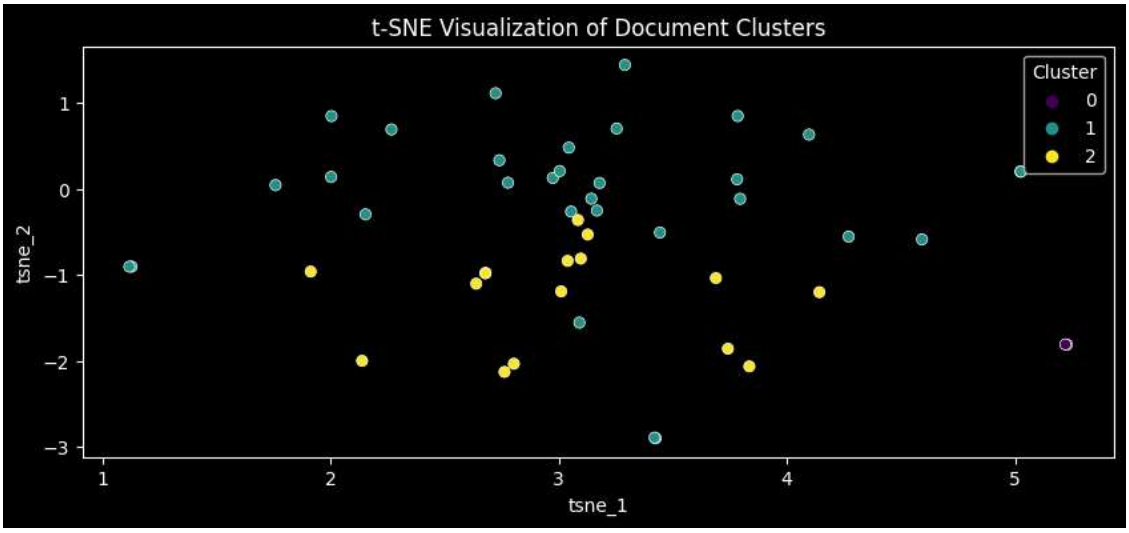
Now is the time for visualize this ... But how to do it??As this is very high dimensional but we can only see 2-d or 3-d data so here we have to do dimensional reduction . We have multiple ways to do dimensional

reduction like pca, svd or tsne. I am using TSNE(T-distributed Stochastic Neighbor Embedding).

```
tsne = TSNE(n_components=2, random_state=42, init="random")
tsne_result = tsne.fit_transform(tfidf_matrix.toarray())

df['tsne_1'] = tsne_result[:, 0]
```

Here is how I did implement tsne. This is actually a visualization technique to visualize the high dimensional data. Here is the graph:

See in the graph here are the three clusters but not so clear distinction between them. So what we can do to improve it like use some other method.



Now let's take the importance of content categorization-

- Enhanced Data Organization: Categorizing content allows for systematic organization of

information. It's akin to arranging a library where each book is placed on a specific shelf, making it easy to find relevant information quickly.

- Facilitates Quick Retrieval: When content is categorized, retrieving specific information becomes more efficient. It's like having a well-indexed book where you can turn directly to the chapter you need.

- Pattern Recognition: Categories help in identifying patterns and trends within the data. This is akin to recognizing recurring themes in a series of books, aiding in deeper insights and analysis.

- Scalable Data Handling: As the volume of data grows, content categorization becomes essential for scalable data handling. It's like having a scalable filing system that can accommodate an expanding collection.

But we used k-means what if we used some deep learning approach(transformers!!)

Using Sentence transformers

- First things first, I'm bringing in a heavyweight, a Sentence Transformer model named 'paraphrase-MiniLM-L6-v2'. This model is like a language maestro; it can understand the nuances and meanings behind sentences.

```
model = SentenceTransformer('paraphrase-MiniLM-L6-v2')
```

- Next, I'm using this model to encode my cleaned comments. It's like translating each comment into a unique numerical representation, capturing the essence of what's being said.

```
   embeddings = model.encode(df['Cleaned_Comments'], show_progress_bar=True)
```

- Now, enter UMAP (Uniform Manifold Approximation and Projection). It's a dimensionality reduction technique, and I'm applying it to my encoded

```
umap_embeddings = umap.UMAP(n_neighbors=15,
n_components=2, min_dist=0.0,
metric='cosine').fit_transform(embeddings)
```

- comment vectors. UMAP is helping me visualize these high-dimensional embeddings in a 2D space.

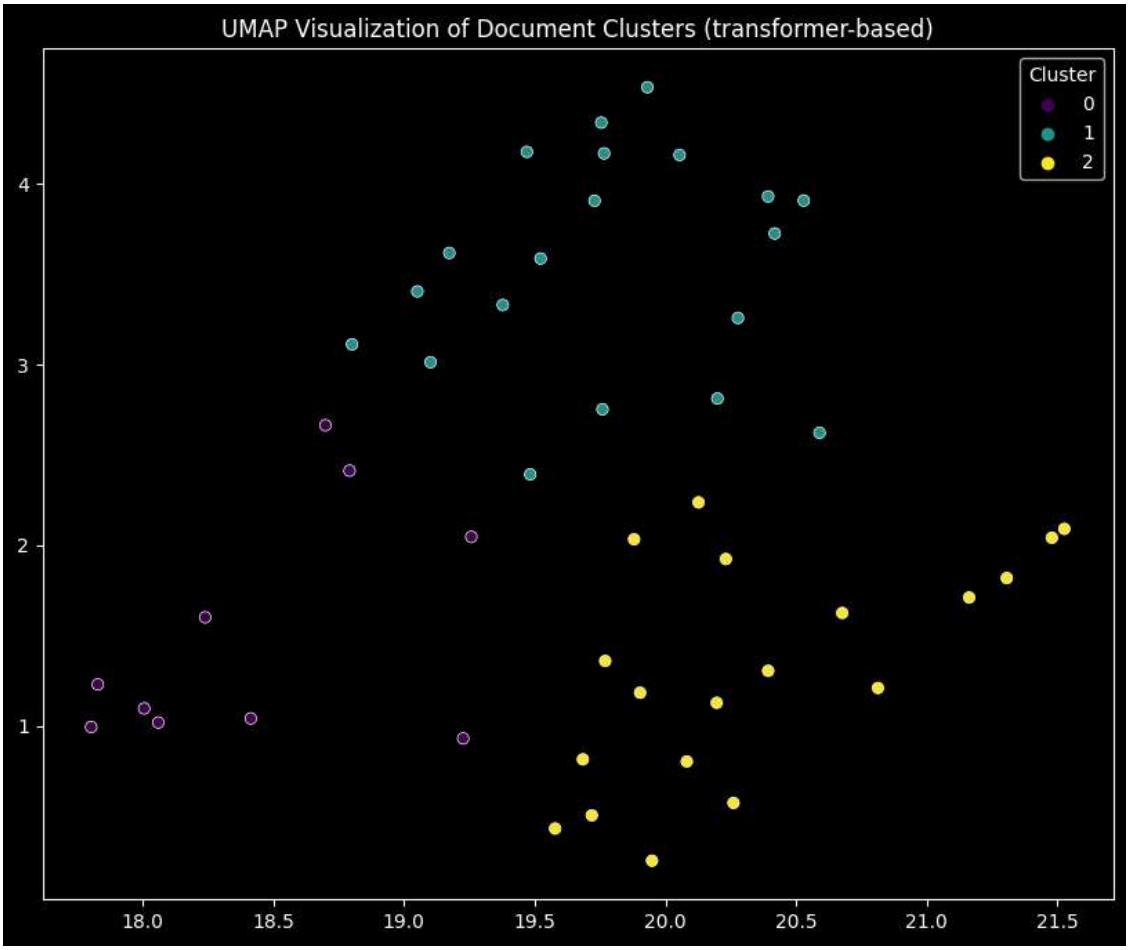
- Brace yourself, because here comes the clustering part. I'm using K-Means, a clustering algorithm, to group these comments into clusters. I've decided to go with three clusters because, well, variety is the spice of life!

```
kmeans = KMeans(n_clusters=num_clusters, random_state=42)

df['Cluster'] = kmeans.fit_predict(umap_embeddings)
```

- Each comment is now getting a label based on which cluster it belongs to. It's like sorting them into different baskets based on their content similarities.

- Now, the grand reveal! I'm plotting a scatter plot to visualize these clusters in 2D space. Each point on the plot represents a comment, and the colors

indicate which cluster they've been assigned to. It's like creating a beautiful map of comments where similar ones stick together.



- I'm displaying this plot, and the title says, 'UMAP Visualization of Document Clusters (transformer-based)'. It showcases how our comments are grouped and organized.

We can clearly see in the graph the three different clusters of color violet, green and yellow showing three clusters. So we can valuely assume our data text is of different semantic values(approx.). Lets see this aspect later.

We can plot this graph in 3d too. I did this but this is not possible to show case that here so I am attaching the html link of that graph.

https://github.com/Ayush-mishra-0-0/dav-project/blob/main/Twitter/3D-UMAP-Visualization-of-Document-Clusters%5B1%5D.html

3) Trend analysis:

First things first, we need a tool called Scattertext. So, I'm making sure it's installed.

```
!pip install scattertext
```

Next, we're creating something called a "corpus" from our comments that is basically combining all the comments together. It's like having a big basket with all our text.

‘’’ import scattertext as st

corpus = st.CorpusFromPandas(df, category_col=None, text_col='Cleaned_Comments').build()

html = st.produce_scattertext_explorer(corpus, category_name=None, not_category_name='Other', width_in_pixels=1000)

with open('trend_visualization.html', 'w') as f f.write(html) ‘’’

Now comes the exciting part. We're using Scattertext to create an explorer. An explorer is like a map that helps us navigate through trends in our comments.

```
html = st.produce_scattertext_explorer(corpus, category_name=None,
not_category_name='Other', width_in_pixels=1000)
```

We don't want our magic to disappear, right? So, we're saving all the trend exploration into an HTML file. It's like capturing our magical journey in a book.

```
with open('trend_visualization.html', 'w') as f:
    f.write(html)
```

Once you open the saved HTML file, it's like opening a treasure chest. You'll see trends, patterns, and shifts in our discussions. It's like having a special pair of glasses that reveal hidden insights.

And there you have it! With just a few lines of code, we've unleashed the power to explore trends in our comments. It's like having a magical guide that shows us what's buzzing and changing over time.

4) TOPIC SEARCH AND RETRIEVAL

For this I am using genism library

I recently explored an intriguing approach to understand the underlying themes within a collection of comments. This process involved utilizing a powerful tool called Gensim, which allowed me to uncover hidden topics and gain valuable insights from the textual data.This converts our corpus to few numerical labelling like 0,1,2 and like this... and this maps our comment to this topic labels which help in its search and easy retrieval.

Step 1: Gathering Insights:

So first we split all the data in dataframe together in tokenized docs.

```
from gensim import corpora, models
tokenized_docs = [doc.split() for doc in df['Cleaned_Comments']]
```

Step 2: Summoning the Oracle - LDA Model:

Next, I invoked a specialized model known as Latent Dirichlet Allocation (LDA). Think of it as consulting a wise oracle that can invoke latent topics within the comments. By configuring the model, I specified the number of topics I wished to extract – in this case, three.

```
dictionary = corpora.Dictionary(tokenized_docs)
corpus = [dictionary.doc2bow(doc) for doc in tokenized_docs]
lda_model = models.LdaModel(corpus, num_topics=3, id2word=dictionary, passes=15)
```

Step 3: Unveiling the Topics:

After running this part of the code , it shows the true meaning of each topic. Like we are solving some puzzle like unrolling ancient scrolls that disclose the thematic nuances hidden within the comments.

This step allowed me to grasp the key concepts hidden in each topic.

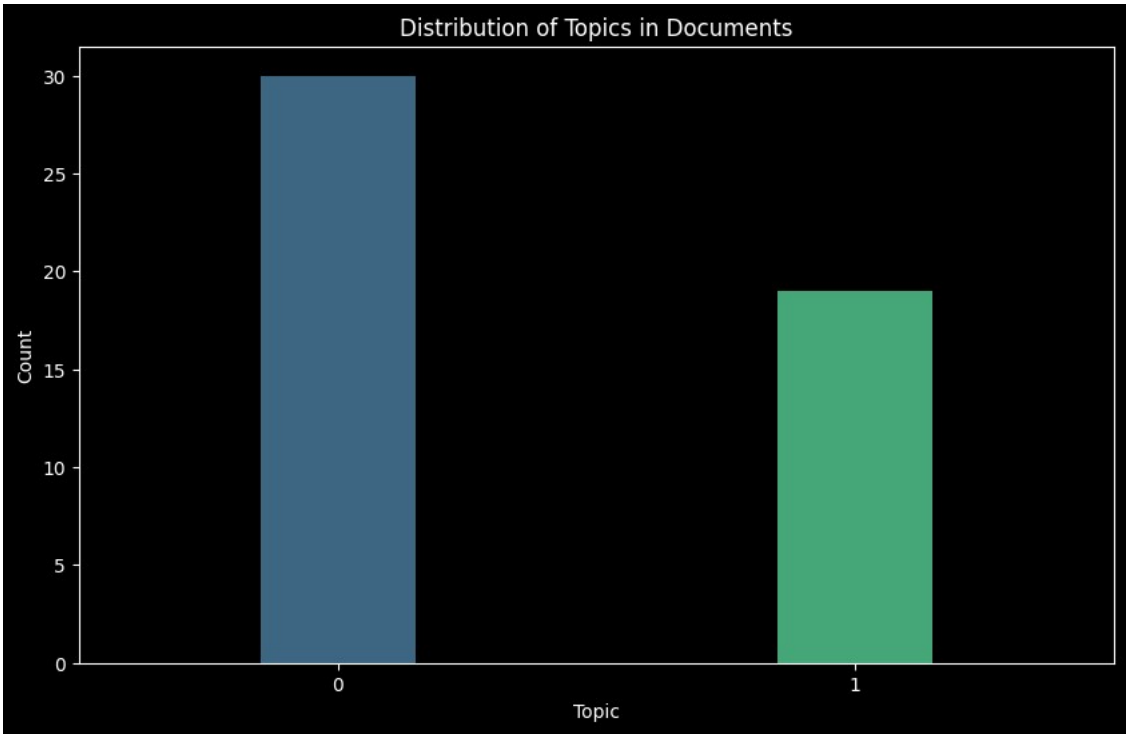
Step 4: Marking Each Comment:

Cleaned_Comments	item	Summary	Cluster	tsne_1	tsne_2	Topic
proud that at this event I worked whole d...	G20 2023	- I proud that at this event I worked whole da...	2	2.679146	-0.978640	0
- I proud of you modi sir ....I love my INDIA...	G20 2023	- I proud of you modi sir ....I love my INDIA ...	2	2.153263	-0.295764	0
- प्रश्न. G20 की स्थापना किस वर्ष हुई?\n\na. ...	G20 2023	a. 1975 \n\nb. 1999 \n\nc. 2008 \n - First G20...	1	3.424253	-2.902411	0
- The list of invitees / participants is very...	G20 2023	- The list of invitees / participants is very ...	1	2.777393	0.072326	1
Very much proud to see Bharat taking such l...	G20 2023	- Very much proud to see Bharat taking such le...	2	3.127025	-0.530775	0

Here we can see the topic column and observe the labels some are 0 and some are 1. To make sense of the results, I assigned each comment to its

predominant topic. This labeling process facilitated a easy understanding of the overarching themes, turning the raw data into actionable insights.Means which 0 are somewhat similar to '0' topic comments and similary for '1' types.

Lets see the simple plot how many are 0 and how many are '1':



Countplot of the distributed topic in the documents.

5) Content Recommendation Using LDA Topics

First of all what is LDA(Latent Dirichlet Allocation)

So here is formal meaning of LDA:-

LDA looks at a document to determine a set of topics that are likely to have generated that collection of words.

In the field of content recommendation, taking advantage of the power of Latent Dirichlet Allocation topics holds high importance. The main objective is to increase user experience by giving personalized and relevant content suggestions based on their individual interests or preferences.

The benefits of this approach are muti-faced. Users are shown with content that resonates with their specific topics of interest, leading to a more time-giving and tailored online experience.

The implementation of this strategy involves several key steps:

Train LDA Model:

The starting step involves training an LDA model on the whole document corpus. This process reveals the topics which are initially latent within the content , providing a structured representation.

```
from sklearn.metrics.pairwise import cosine_similarity
topic_similarity_matrix = cosine_similarity(lda_model.get_topics(), lda_model.get_topics())
```

Assign Topics to Documents:

After completing the training step , the LDA model is utilized to assign dominant topics to each document. This step categorizes the content based on the underlying themes revealed by the model.

```
def recommend_content(user_topics, df, topic_similarity_matrix):
    user_topics_vector = [1 if i in user_topics else 0 for i in range(lda_model.num_topics)]
    document_topics_matrix = lda_model.get_document_topics(df['Cleaned_Content'].apply(lambda x:
dictionary.doc2bow(x.split())))
    document_topics_vector = [max(dict(doc_topics).items(), key=lambda x: x[1])[0] for doc_topics in
document_topics_matrix]
    similarities = cosine_similarity([user_topics_vector], [topic_similarity_matrix[topic] for topic in
document_topics_vector])
    df['Similarity'] = similarities[0]
    recommended_content = df.sort_values(by='Similarity', ascending=False)['Cleaned_Content'].tolist()

    return recommended_content
```

Build Topic Similarity Matrix:

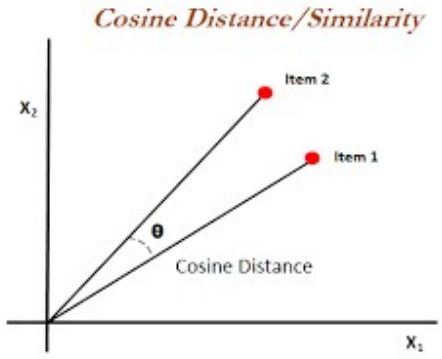
To view the proximity of topics, a topic similarity matrix is made . Cosine similarity metrics are used to measure the closeness between different LDA topics.

Content Recommendation:

Now we see the what is our interest and in that respect , the system recommends content with similar themes. The similarity between our given topics and document topics guides the recommendation process which in turn helps so many companies to promote their products.

```
user_preferred_topics = [1, 2]
recommended_content = recommend_content(user_preferred_topics, df, topic_similarity_matrix)
print("Recommended Content:")
print(recommended_content)
```

The implementation code involves calculating cosine similarity between LDA topics, creating a function to recommend content based on user topics, and integrating similarity scores into the dataset. The final step is sorting documents by similarity and extracting the recommended content.



In above diagram we can see the cosine similarity like suppose we have two items – item1 and item2 then we

Take dot product between them and divide them by their product

A=item1

B=item2

Cos(theta)= A.B/|A||B|

So this shows the similarity as similar== cos(theta).

This approach transforms content recommendation into a dynamic and user-centric process, aligning seamlessly with individual preferences and interests. The result is an enriched and personalized user experience, fostering deeper engagement and satisfaction.

This completes our analysis part and now is the time to implement a model on our data.

Before implementing any model we need to understand why there is need to even implement a model. So lets spend time over this:

Implementing a model is essential to derive meaningful insights and predictions from complex datasets, facilitating informed decision-making in various domains. In a data-driven world, models serve as invaluable tools to extract patterns, trends, and relationships that may remain obscured in vast and intricate datasets. The need for models arises from the desire to harness the potential of data to solve problems, optimize processes, and gain a deeper understanding of underlying phenomena.

Models are especially crucial when dealing with huge volumes of information, where human analysis alone may be impractical or prone to oversight. They enable the automation of tasks, prediction of future outcomes, and identification of factors influencing a system. Whether in business, healthcare, finance, or other sectors, models empower organizations to make evidence-based choices, enhance efficiency, and mitigate risks.

Moreover, models contribute to the advancement of scientific knowledge by validating hypotheses and exploring intricate relationships in intricate systems. They act as predictive tools, helping to anticipate future scenarios and plan accordingly. Ultimately, the implementation of models is a strategic imperative, unlocking the latent potential within data and driving innovation across diverse fields.

Now coming the modelling implement I did on youtube comments data collected using selenium.

WORKFLOW:

Here are the main steps-

a) Importing the main the libraries

First I imported the all the basic libraries like pandas(data handling and maintaining library), numpy(numerical handling library) , matplotlib(data visualization library) and then seaborn (which is also a data visualization library).

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

b) Reading the data from '`Youtube_scrapping_comments.csv`'

Our data looks this we have 4 columns i.e

	Video Link	Video Title	Cleaned_Comments	item
0	https://www.youtube.com/watch?v=F1pUDSue8mQ&pp...	World leaders arrive at the Bharat Mandapam in...	@mjfansumit - I proud that at this event I wor...	G20 2023
1	https://www.youtube.com/watch?v=X8rM829c7to&pp...	PM Modi addresses The Hindustan Times Leadersh...	@priyachattaraj9926 - I proud of you modi sir ...	G20 2023
2	https://www.youtube.com/watch?v=dbgGV5-TrdE&pp...	India's Big Win at G20 Summit: G20 New Delhi L...	- प्रश्न. G20 की स्थापना किस वर्ष हुई?\n\na. ...	G20 2023
3	https://www.youtube.com/watch?v=G_O8ogCpbRI&pp...	G-4 | G-6 | G-7 | G-8 | G-10 | G-11 | G-20 | D...	@mukundrajan8961 - The list of invitees / part...	G20 2023
4	https://www.youtube.com/watch?v=Lf-hYTjapiw&pp...	Turkish Parliament Boycotts Coca Cola and Nest...	@DhimanBharadwaj - Very much proud to see Bhar...	G20 2023

Video link , Video Title, Cleaned_Comments, Item.

Our main concern data column are cleaned-comments on which I will apply nltk and bert- base modelling.

First I will do the modelling using very popular nltk library of python.

So here are the steps in which I did this process-

a) Tokenisation-

It is the process of breaking down a text into smaller units called tokens. These tokens can be words, phrases, symbols, or other meaningful elements depending on the context. The purpose of tokenization is to simplify the text and make it more manageable for further analysis or processing.

In natural language processing (NLP), tokenization is a important step to prepare textual data for various tasks such as text analysis, machine learning, and information retrieval. The tokens obtained from tokenization acts as the building blocks for above said analyses, allowing different algorithms to operate on these segregated elements rather than the entire text.

For example, consider the sentence

"Tokenization is an important step in NLP."

After tokenization, this sentence can be written as a list of tokens:

["Tokenization", "is", "an", "important", "step", "in", "NLP"].

Tokenization can be implemented at various levels, including word-level tokenization, sentence-level tokenization, or even character-level tokenization, depending on the our requirements of the topic in the hand. It is a very basic pre-processing step in natural language processing pipelines.

```
import nltk
nltk.download('punkt')
from nltk.tokenize import word_tokenize

df['Tokenized_Comments'] = df['Cleaned_Comments'].apply(word_tokenize)
```

In code also it is very easy to implement like first download the 'punkt' from nltk and then just apply word_tokenizer as we need to apply alogorithm which will find semenatic meanings between the words so we are applying word tokenizer, this is the reason.

b) Removing the stopsword:

One important step before doing main modelling part is to remove the stopsword. So what are stopwords ??

Stopwords are the those words which do not have their own semantic value so we can remove them. Now how it improves the overall process.

Noise Reduction: Stopwords, such as articles, prepositions, and conjunctions, are commonly occurring words that don't carry significant meaning on their own. Removing them helps reduce noise in the text, allowing the focus to be on more meaningful words.

Computational Efficiency: Stopword removal reduces the size of the text data, leading to improved computational efficiency. Since stopwords appear frequently in a text but contribute little to the overall meaning, their removal streamlines subsequent analyses and model training.

Improved Accuracy: In tasks like sentiment analysis or document classification, the presence of stopwords can introduce noise and potentially mislead the model. By eliminating irrelevant words, the accuracy of these tasks is often improved.

Consistent Analysis: Removing stopwords ensures a more consistent and standardized analysis across different texts. It helps create a cleaner and more uniform representation of the text data, making it easier to compare and draw meaningful insights.

Focus on Content Words: Stopword removal allows analyses to focus on content words that carry the semantic value of the text. This is mainly important in tasks like keyword extraction, topic modeling, and information retrieval because it focus on main words only.

```
nltk.download('stopwords')
```

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

```
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))
df['Filtered_Comments'] = df['Tokenized_Comments'].apply(lambda tokens: [word for word in tokens if word.lower() not in stc
```

In Code also it is easy to implement just we have to download the stopword from the nltk library, we can also add some our custom words to this list which we find not important to the analyzes.

So after importing the stopwords from nltk, we add the English argument to the stopword as our data is

in English and then store the changed or filtered comments in a another column of the data frame named filtered_comments as we can in above code. We have use apply function it is same as applying the stopword function at each row but using apply function it automatically does, it is similar to a lambda function.

c) Lemmatization: It is the process which helps to convert our word to its root word like suppose we have a word 'giving' so after applying lemmatizating it will convert this word to 'give'

Example: word- 'giving' return word-'give/giv'

Now what is the importance?? Here are few of them:

Normalization of Words: Lemmatization reduces words to their base or root form, providing a normalized representation of the vocabulary. This ensures that different inflected forms of a word are treated as a single, consistent entity, facilitating more accurate analysis.

Improved Text Understanding: By converting words to their base form, lemmatization enhances the understanding of the underlying meaning in a text.

It helps capture the essence of words, particularly in languages with complex inflections, ensuring that variations in tense, number, or gender do not hinder comprehension.

Enhanced Model Performance: In tasks such as text classification, sentiment analysis, or information retrieval, lemmatization contributes to improved model performance. It reduces the dimensionality of the feature space, allowing models to focus on the core semantics of the text and generalize more effectively.

Consistent Analysis: Lemmatization promotes consistency in text analysis by ensuring that different grammatical forms of a word are treated uniformly. This consistency is crucial for tasks like topic modeling or keyword extraction, where variations in word forms could lead to fragmented or inaccurate results.

Information Retrieval: Lemmatization aids in information retrieval by consolidating related terms.

Like give or given or giving all are same after applying l lemmatization. Users searching for information may use different forms of a word, and lemmatization helps match these variations, improving the accuracy of search results.

Multilingual Applications: In multilingual cases, lemmatization is particularly important for maintaining coherence and consistency in the analysis of diverse languages, each with its own set of grammatical rules as it basically removes all the grammar rules.

```
nltk.download('wordnet')
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
True
```

```
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
df['Lemmatized_Comments'] = df['Filtered_Comments'].apply(lambda tokens: [lemmatizer.lemmatize(word) for word in tokens])
```

For code implementation, we first have to download the 'wordnet' package from nltk and then import WordnetLemmatizer and instancing to a variable. After this store the change or lemmatized comments in a separate column named as Lemmatized_Comments.

D) Modelling-

For this I am using “vader_lexicon” package downloaded from the nltk library. It is common package to do tasks related sentiment analysis. It is a lexicon and rule-based sentiment analysis tool. The term "VADER" stands for Valence Aware Dictionary and Sentiment Reasoner.

Lexicon-Based Approach: VADER is a lexicon and rule-based sentiment analysis tool. It relies on a pre-built lexicon (vader_lexicon) that contains a comprehensive list of words, along with their associated sentiment scores. Each word is assigned a polarity score ranging from -1 (most negative) to 1 (most positive).Like every word is assigned a vector which has numerous values where each denotes a semantic value which some other word.

Polarity Scoring: While analysis, the input text is tokenized into words and phrases, though we did step manually but this package also do this directly . VADER then assigns sentiment scores to each token based on the lexicon(in firsts step). The scores consider not only the individual words but also account for the context and sentiment intensity modifiers.

Valence Shifting: VADER handles valence shifting, where certain words or phrases can modify the sentiment of others. For example, negations ("not good") or intensifiers ("very good") impact the overall sentiment. VADER incorporates these rules to appropriately adjust the sentiment scores which ranges from -1 to 1.

Sentence-Level Analysis: VADER also operates at the sentence level, taking into account the sentiment expressed within each sentence(whole at a time). This allows for a more nuanced understanding of sentiment variations throughout the text and help in tasks we need sentence level analysis.

Compound Score: The algorithm calculates a compound score for the entire text, representing the overall sentiment polarity. The compound score provides a single metric on a scale from -1 to 1, where negative values indicate negative sentiment, positive values indicate positive sentiment, and values close to zero suggest neutrality.

Handling Emoticons and Slangs: VADER is designed to handle emoticons, acronyms, and common slang expressions often found in social media texts, making it suitable for sentiment analysis in informal and short-form content.

```
nltk.download('vader_lexicon')
```

```
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
True
```

```
from nltk.sentiment import SentimentIntensityAnalyzer

sia = SentimentIntensityAnalyzer()

df['Sentiment_Score'] = df['Cleaned_Comments'].apply(lambda x: sia.polarity_scores(x)['compound'])
df['Sentiment_Label'] = df['Sentiment_Score'].apply(lambda score: 'Positive' if score > 0 else ('Negative' if score < 0 els
```

Like in this code, we can see first we downloaded ‘vader-lexicon’ from nltk library and then I imported the SentimentIntensityAnalyser() from nltk which then instance to a variable to sia. After applying the polarity_score function of sia to each row of the data and storing the Sentiment_score(which ranages from -1 to 1) and sentiment_label(which is either positive or negative)

Lemmatized_Comments	Sentiment_Score	Sentiment_Label
[@, mjfansumit, -, proud, event, worked, whole...	0.9989	Positive
[@, priyachattaraj9926, -, proud, modi, sir, ...	0.9881	Positive
[-, प्रश्न, ., G20, की, स्थापना, किस, वर्ष, हु...	0.9993	Positive
[@, mukundrajan8961, -, list, invitee, /, part...	0.9986	Positive
[@, DhimanBharadwaj, -, much, proud, see, Bhar...	1.0000	Positive

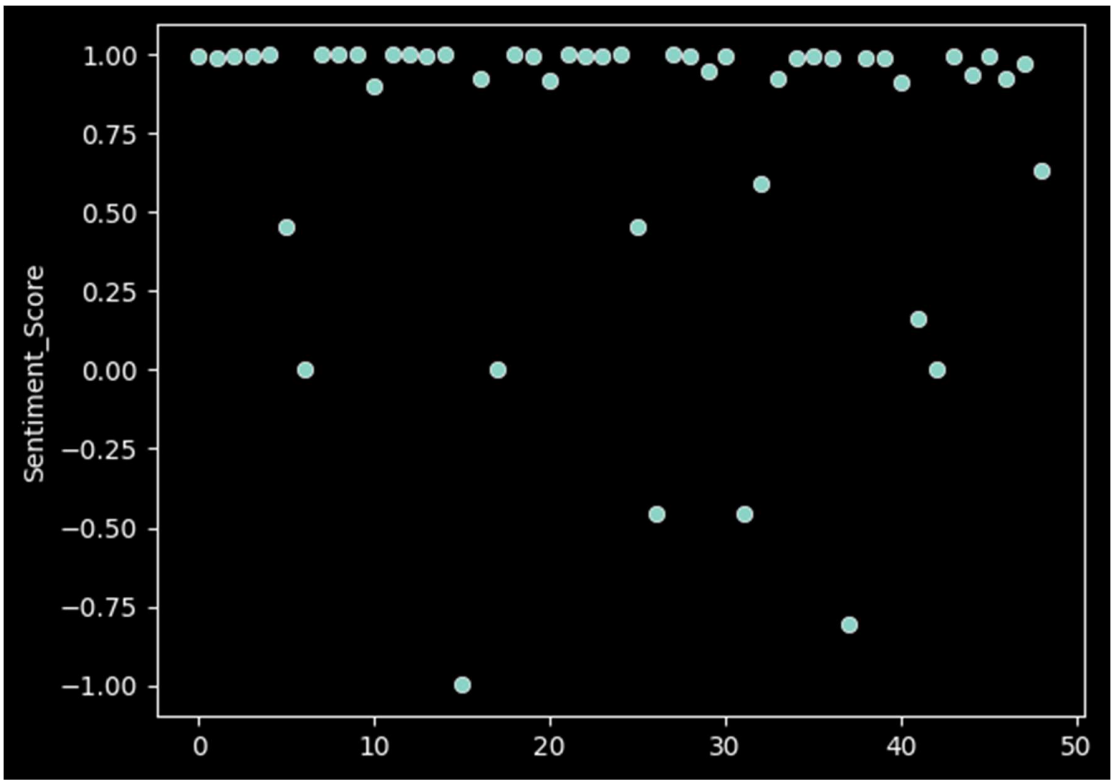
Here is screen shot of a section of the dataframe. We can the sentiment label are amjorily positive it may give one impression that are all the data row text are positive in sentiment sense ??

Lets find out this. For we will apply value_counts function on the sentiment label column which will return all the unique values and its count. Like in our case

```
df1.Sentiment_Label.value_counts()
```

```
Positive    34
Negative    15
Name: Sentiment_Label, dtype: int64
```

it is 34 postive points and 15 negative too. I plotted scatterplot of this tool .



Observe here that most of the dots are close to 1 which implies that they are positive and very few are negative.

So here I used the nltk library for the purpose, now i will sentence_transformer

which is deep learning, neural network approach.

- The framework is based on PyTorch and Transformers and offers a large collection of pre-trained models tuned for various tasks. Further, it is easy to fine-tune your own models.
- Its sentence embeddings are based on bert-networks, like Sentence Transformer relies on pre-trained transformer-based models. These models are typically large neural networks trained on vast amounts of textual data to learn contextualized embeddings. Examples of such models include BERT, RoBERTa, or DistilBERT.
- It involves feeding the aggregated sentence embeddings into a neural network classifier, often a simple feedforward neural network. This classifier is trained to predict the sentiment label (positive, negative, or neutral) based on the learned sentence embeddings.

```
from transformers import pipeline
sentiment_pipeline = pipeline('sentiment-analysis')
df1['Sentiment_Prediction'] = df1['Cleaned_Comments'].apply(lambda x: sentiment_pipeline(x[:512])[0])
df1['Sentiment_Label'] = df1['Sentiment_Prediction'].apply(lambda prediction: prediction['label'])
sentiment_label_map = {'POSITIVE': 'Positive', 'NEGATIVE': 'Negative', 'NEUTRAL': 'Neutral','COMPOUND':'Compound'}
df1['Sentiment_Label'] = df1['Sentiment_Label'].map(sentiment_label_map)
```

Like in the code, first I imported pipeline. The pipeline module from the transformers library, is imported to create a pre-built sentiment analysis pipeline.

Create Sentiment Analysis Pipeline:

The pipeline function is used to create a sentiment analysis pipeline. The argument 'sentiment-analysis' specifies the type of task for which the pipeline will be used.

Apply Sentiment Analysis to DataFrame:

The sentiment pipeline is applied to the 'Cleaned_Comments' column of the DataFrame (df1). For each comment, sentiment analysis is performed, and the result (prediction) is stored in the 'Sentiment_Prediction' column. The lambda x: sentiment_pipeline(x[:512])[0] ensures that only the first 512 tokens of each comment are considered to fit within model constraints.

Map Sentiment Labels:

The 'label' field from the sentiment prediction result is extracted and stored in the 'Sentiment_Label' column.

Map Compound Sentiment Label:

A mapping dictionary (sentiment_label_map) is used to map sentiment labels to more readable labels ('Positive',

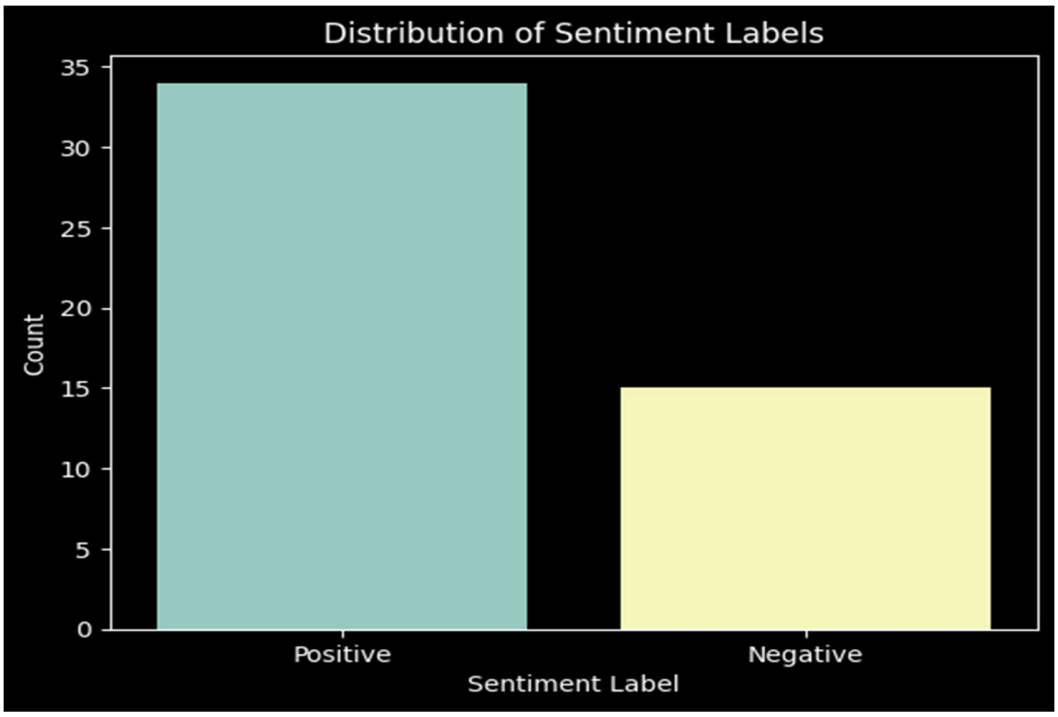
'Negative', 'Neutral', 'Compound'). The 'Sentiment_Label' column is updated accordingly.

In conclusion, now we check the results so for that we need to apply few functions like value_count and plot few graphs .

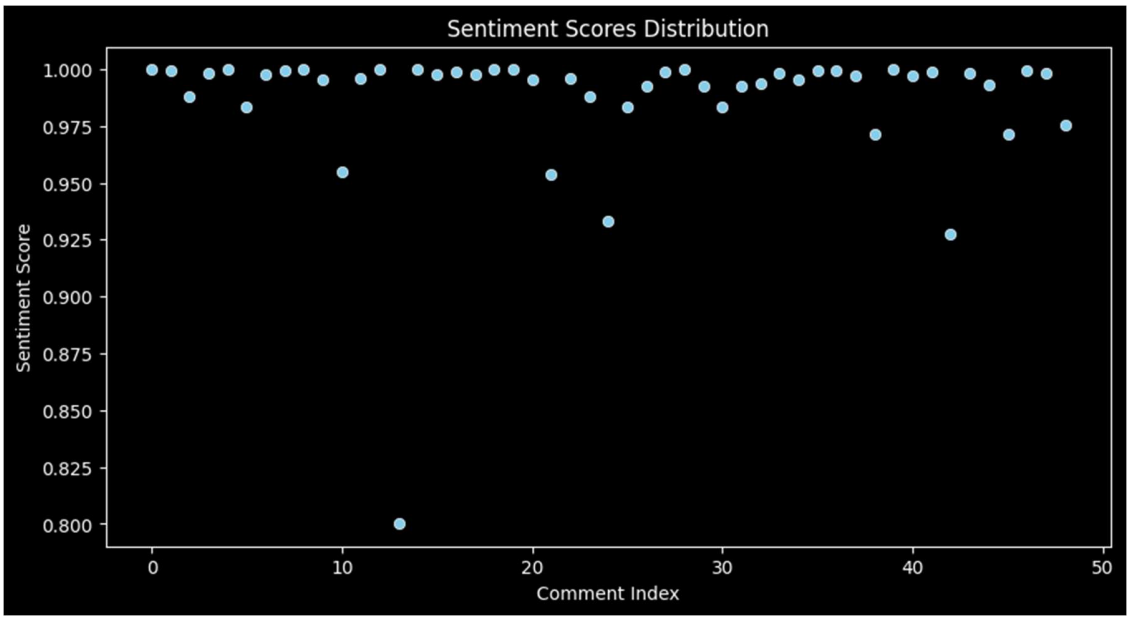
```
df1.Sentiment_Label.value_counts()
```

```
Positive    34
Negative    15
Name: Sentiment_Label, dtype: int64
```

Like here see the unique of the column and its count.



Its bar-plot between the labels and its count.



Observe here that most of the dots are close to 1 which implies that they are positive and very few are negative.

Which again implies that most of the reactions of the people in the youtube comments are positive in favour of G20 event that held in New Delhi, INDIA.

Workflow For Twitter data

1) Data collection- Here as we know Twitter does not allow the direct scapping . So Here we can not scrape the data using selenium like we did in youtube comments data collection. So there are 2 options
   a) Either use official twitter api- for this we have pay as this is not freely available. So we hae to first pay $100 and get our twitter api keys and tokens and use them to invoke the api call and then fetch the data as want and as much we want(though every call will require some money).
   b) Use Apify – It is a awesome website which gives every new user which sign up on this website free $5 to scrap. And one might think this is too less we can't scrape much data using this, but this is not true. As I scraped more than 1000 rows and it just costs me $0.1 and now u can think how modest this is. So all in all it is very easy to ui based scraping website where we just have to give the topic or links or the twitter account of which we have to scrape and we can also specify the time period during which we want our data.

So, I specified the topics and time period related to G20 search and call this api and get my data in a csv file which I named twitter-data.html.I am attaching its link.

https://github.com/Ayush-mishra-0-0/dav-project/blob/main/Twitter/twitter-data.html

Then I did some changes on this like following:

a) First I convert this html file to csv and named it to twitter-data.csv and using read_csv function, made our dataframe df. Here is a head of df.

	banner_image	fullname	images/0	images/1	images/2	images/3	images/4	images/5	in_reply_to/0	in_reply_to/1	...	tw
0	NaN	ORF Mumbai	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	
1	NaN	The Hindu	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	
2	NaN	ORF Kolkata	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	
3	NaN	Itamaraty Brazil BR	https://pbs.twimg.com/media/F-bIIACXwAAMiLU.jp...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	
4	NaN	The Centre for New Economic Diplomacy	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	

b) Next as we can see in our data there are many not useful columns like check how many columns there are. For this we use df.columns. This will

“ Index(['banner_image', 'fullname', 'images/0', 'images/1', 'images/2',
'images/3', 'images/4', 'images/5', 'in_reply_to/0', 'in_reply_to/1',
'in_reply_to/2', 'in_reply_to/3', 'in_reply_to/4', 'in_reply_to/5',
'in_reply_to/6', 'in_reply_to/7', 'in_reply_to/8', 'in_reply_to/9',
'in_reply_to/10', 'in_reply_to/11', 'in_reply_to/12', 'language',
'likes', 'num_followers', 'num_following', 'query', 'quotes', 'replies',
'retweets', 'text', 'timestamp', 'total_likes', 'total_tweets',
'tweet_avatar', 'tweet_hashtags/0', 'tweet_hashtags/1',
'tweet_hashtags/2', 'tweet_hashtags/3', 'tweet_hashtags/4',
'tweet_hashtags/5', 'tweet_hashtags/6', 'tweet_hashtags/7',
'tweet_hashtags/8', 'tweet_id', 'tweet_links/0', 'tweet_links/1',
'tweet_mentions/0', 'tweet_mentions/1', 'tweet_mentions/2',
'tweet_mentions/3', 'tweet_mentions/4', 'tweet_mentions/5',

'tweet_mentions/6', 'tweet_mentions/7', 'tweet_mentions/8',
'tweet_mentions/9', 'tweet_mentions/10', 'tweet_mentions/11',
'tweet_mentions/12', 'tweet_mentions/13', 'tweet_mentions/14',
'tweet_mentions/15', 'url', 'username'],
dtype='object')" give us. So We will only keep important columns in dataframe which are ['fullname','likes','query','replies','retweets','text','timestamp','url','username']

c) Again as it is un-supervised learning as we do not have labels like positive, negative or something else in that matter.So we have to do again k-means approach or sentence-transformation approach to first labelize the data and then see how well it is.

d) But this lets the method of this unsupervised learning. What is the fun of doing same thing as we had already see how we implement sentence-transfomer where task is sentiment analysis, instead of that, I am using a new think which is a classification thing instead of clustering. So lets think I am giving some labels and asking my model -tell me does this text belongs to this class and it asks the same question for every text of rows to each class and calculate their respective probabilities and arrange them in decreasing order and whose probability it matches maximum, that

class would be answer. So now see how I implemented this in my code-

```
from transformers import pipeline
zsc= pipeline(task= "zero-shot-classification")
```

-first importing the pipeline as it will connect out instance variable to the task which it imports from hugging-face. So task is ZERO-SHOT-CLASSIFICATION.
Zero-shot classification involves training a model to predict labels or categories that it has never seen during training. Here's an overview of how zero-shot classification works:

Pre-trained Language Models:

Zero-shot classification leverages the capabilities of pre-trained language models, such as BERT, GPT, or similar architectures. These models are trained on large amounts of text data for tasks like language modeling or masked language modeling.
Semantic Embeddings:

During pre-training, the model learns to generate semantic embeddings for words, phrases, and sentences. These embeddings capture the

contextual relationships between words and encode rich semantic information.

Task Formulation:

In zero-shot classification, the model is provided with a set of candidate labels or categories that it should predict for a given input text. These labels are often specified as part of the input, guiding the model about the types of predictions expected.

Prompt Engineering:

A key aspect of zero-shot classification is prompt engineering. The input to the model includes a prompt or a formulation that instructs the model about the nature of the classification task and how to associate the input with the provided labels.

Model Inference:

During inference, the model takes an input text along with the specified labels and generates predictions for the given labels. The model relies on the pre-learned semantic embeddings to understand the context of the input and make predictions.

Output and Evaluation:

The model produces a probability distribution over the candidate labels, indicating the likelihood of each label being relevant. The label with the highest probability is predicted as the output. Evaluation metrics like accuracy or F1 score are used to assess the model's performance on the zero-shot task.

So here are main things we need to know before seeing code of this task.
To perform zero-shot classification, the model is provided with a set of candidate labels or categories, and the goal is to predict the relevant label(s) for a given input text, even if those labels were not part of the training data.

```
x=zsc(
    df1['text'][0],
    candidate_labels=['Positive','Negative','Positive criticism','Negative criticism','Neutral']
    )
```

Here is the code implementation.

Observe carefully in the code, The candidate labels are ‘positive’,’negative’,’negative cristism’,’positive cristism’ and ‘neutral’. To see the value counts of these labels in our data call value_count function after applying this ‘ZSC’ function on each row of our data.

After applying the above function, our dataframe looks like this .

	text	label
0	The #NewDelhi Declaration's #policy commitment...	Positive
1	As #India prepares to end its #G20 presidency ...	Negative criticism
2	Adding weight to #P20's representativeness is ...	Positive
3	The Secretary-General, Ambassador Maria Laura ...	Positive
4	Under #India's leadership during the #G20 pres...	Positive
...	...	...
519	With commitment, collaboration, and a comprehe...	Positive
520	This Policy Brief recommends the creation of a...	Positive
521	Big news! The African Union, representing all ...	Negative criticism
522	From trust deficit to trust management, #G20 N...	Positive
523	According to the foreign ministry statement, W...	Negative criticism

See the label ..

Its value-count data is

```
Positive              298
Negative criticism    165
Negative               56
Positive criticism      5
Name: label, dtype: int64
```

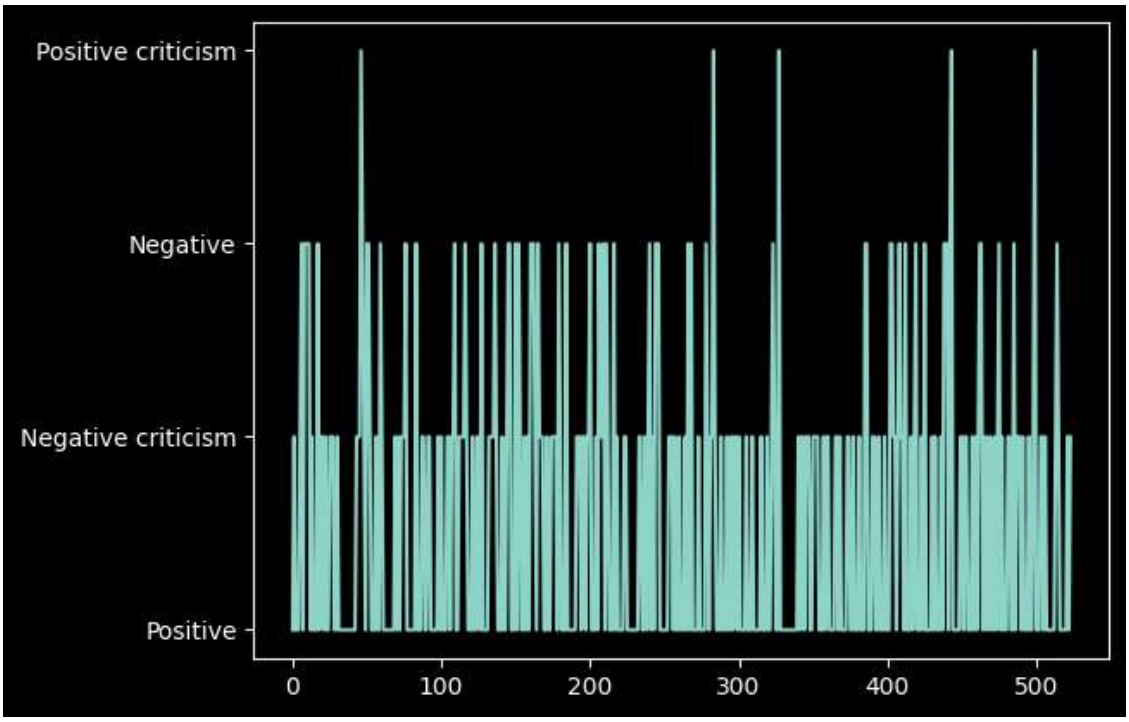
So text data in positive (in semantics sense) is highest and positive criticism is lowest just 5.

So lets plot the graphs to take a little better understanding of the data before modelling.

First I want to style my plots differently. plt.style.use('dark_background'): This line sets the plot style to a dark background, enhancing the visibility of the word cloud visualization.

Line Plot:

plt.plot(data['label']): This line plots a line chart using the 'label' data from the DataFrame. The purpose of this line plot may be to visualize trends or patterns in the label data.



Word Cloud Generation:

show_wordcloud(data['text'].values, title="Most freq words in our corpus"): This line calls a function named show_wordcloud with the 'text' data from the DataFrame as input. The function generates a word cloud visualization based on the provided text data.

Word Cloud Function:

The show_wordcloud function utilizes the WordCloud library to create the visualization. It sets parameters such as background color, maximum number of words, and maximum font size. The word cloud is generated from the concatenation of all text data in the 'text' column.

```
def show_wordcloud(data, title = None):
    wordcloud = WordCloud(
        background_color='black',
        max_words=200,
        max_font_size=40,
        scale=1,
        random_state=1
    ).generate(" ".join(data))

    fig = plt.figure(1, figsize=(15, 15))
    plt.axis('off')
    if title:
        fig.suptitle(title, fontsize=20)
        fig.subplots_adjust(top=2.3)

    plt.imshow(wordcloud)
    plt.show()
```

Display Word Cloud:

The resulting word cloud is displayed in the output. It visually represents the most frequent words in the corpus, with larger words indicating higher frequency.

Most freq words in our corpus

Initialize Subplots:

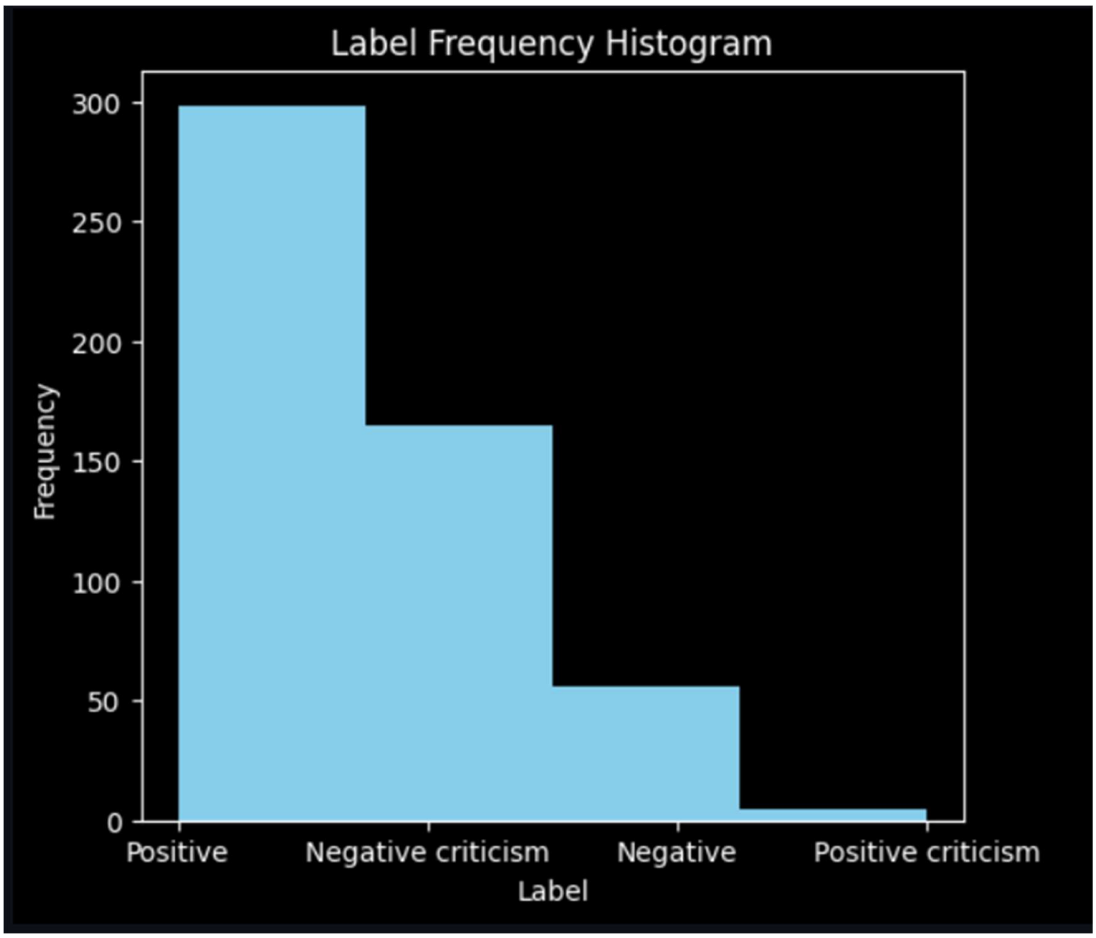
I create a figure and set of subplots with one row and two columns. The entire figure size is adjusted to 12 by 5 inches.

Set Plot Style:

I apply a dark background style to enhance the visual appeal of the plots.

Left Subplot - Histogram:

I generate a histogram of label frequencies using the 'label' data. The number of bins corresponds to the unique number of labels, and the bars are colored in sky blue.



I set the title of the left subplot to 'Label Frequency Histogram', and label the x and y-axes accordingly.

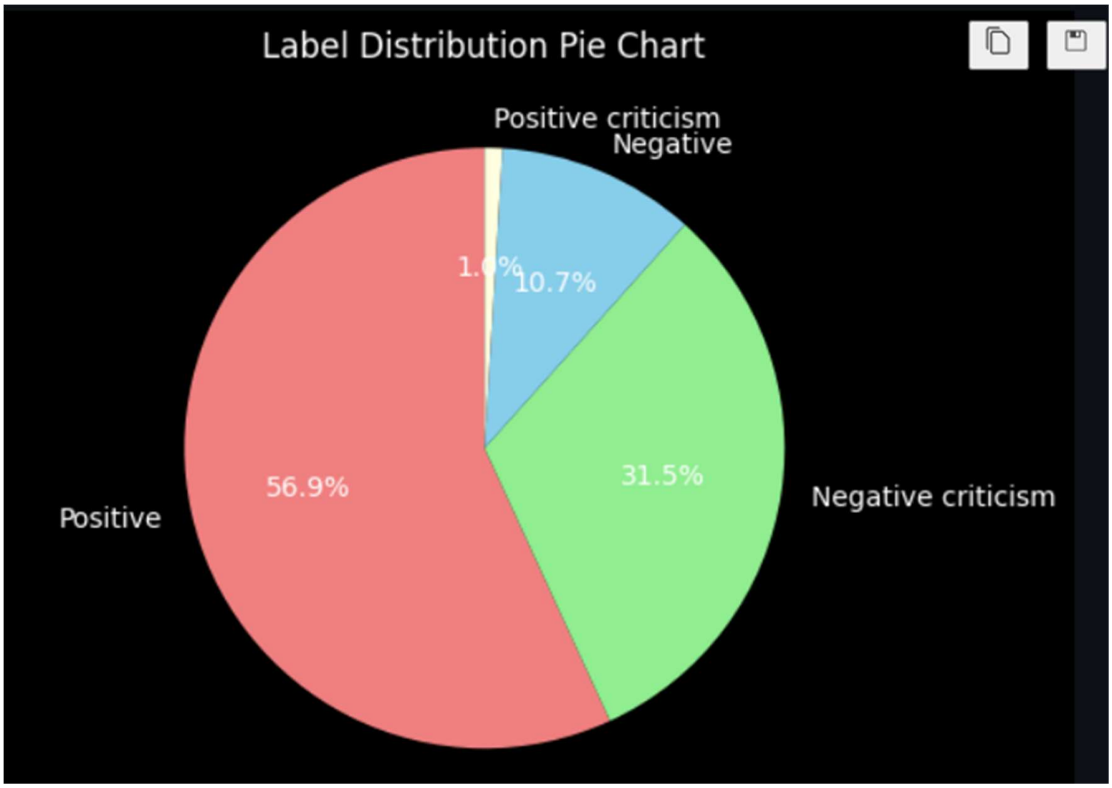
Right Subplot - Pie Chart:

I calculate the counts of each label in the 'label' column.

I create a pie chart using these counts, displaying percentages on each slice. Different colors are assigned to each label category.

The title of the right subplot is set to 'Label Distribution Pie Chart'.

Display the Plots:



I showcase both subplots together for a comprehensive view.

Tokenization Function:

```
def tokenize(text):
    """ basic tokenize method with word character, non word character and digits """
    text = re.sub(r" +", " ", str(text))
    text = re.split(r"(\d+|[a-zA-ZğüşıöçĞÜŞİÖÇ]+|\W)", text)
    text = list(filter(lambda x: x != '' and x != ' ', text))
    sent_tokenized = ' '.join(text)
    return sent_tokenized
```

I define a basic tokenizer function that splits the input text into words, non-word characters, and digits. It ensures proper removal of extra spaces.

Tokenize Data:

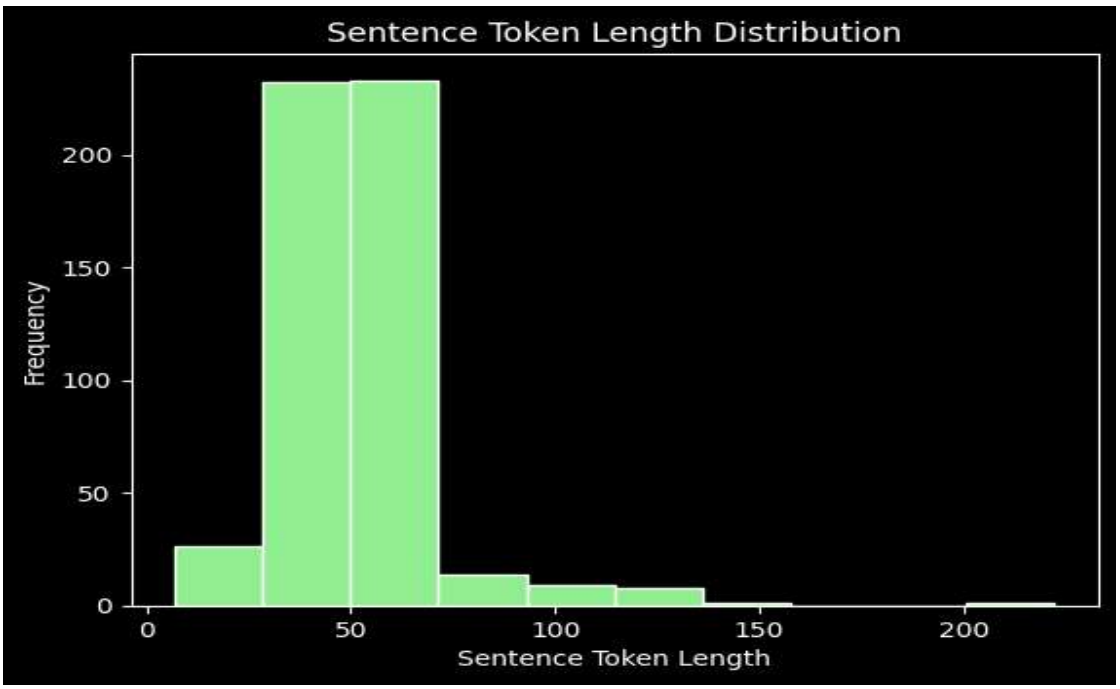
I apply the tokenization function to the 'text' column of the DataFrame, creating a new column named 'tokenized_review' with the tokenized text.

Calculate Token Length-

I determine the number of tokens in each tokenized review and store the results in a new column named 'sent_token_length'.

	tokenized_review	sent_token_length
The # NewDelhi Declaration ' s # policy commit...		56
As # India prepares to end its # G 20 presiden...		59
Adding weight to # P 20 ' s representativeness...		48
The Secretary - General , Ambassador Maria Lau...		58
Under # India ' s leadership during the # G 20...		43
...		...
With commitment , collaboration , and a compre...		54

So here is the sent_token_length. We can also plot this.Lets plot histogram of data['sent_token_length'].



As we can see low sentence token length has high frequency and high sentence token length has low frequency.

Now we will tokenize with the help of bert-tokenizer as it is considered as high accuracy model

So these are steps to followed:

a) Import BertTokenizer:

I import the BertTokenizer class from the transformers library. This class is designed to tokenize text in a manner compatible with BERT models.

b) Initialize Tokenizer:

I create an instance of the BertTokenizer class using the from_pretrained method and specifying the pre-trained BERT model 'bert-base-uncased'.

The do_lower_case parameter is set to True, indicating that the tokenizer should convert all text to lowercase during tokenization.

Plotting Sentence Token Length Distribution:

I use plt.hist to create a histogram of the sentence token lengths in the 'sent_bert_token_length' column of the data DataFrame.

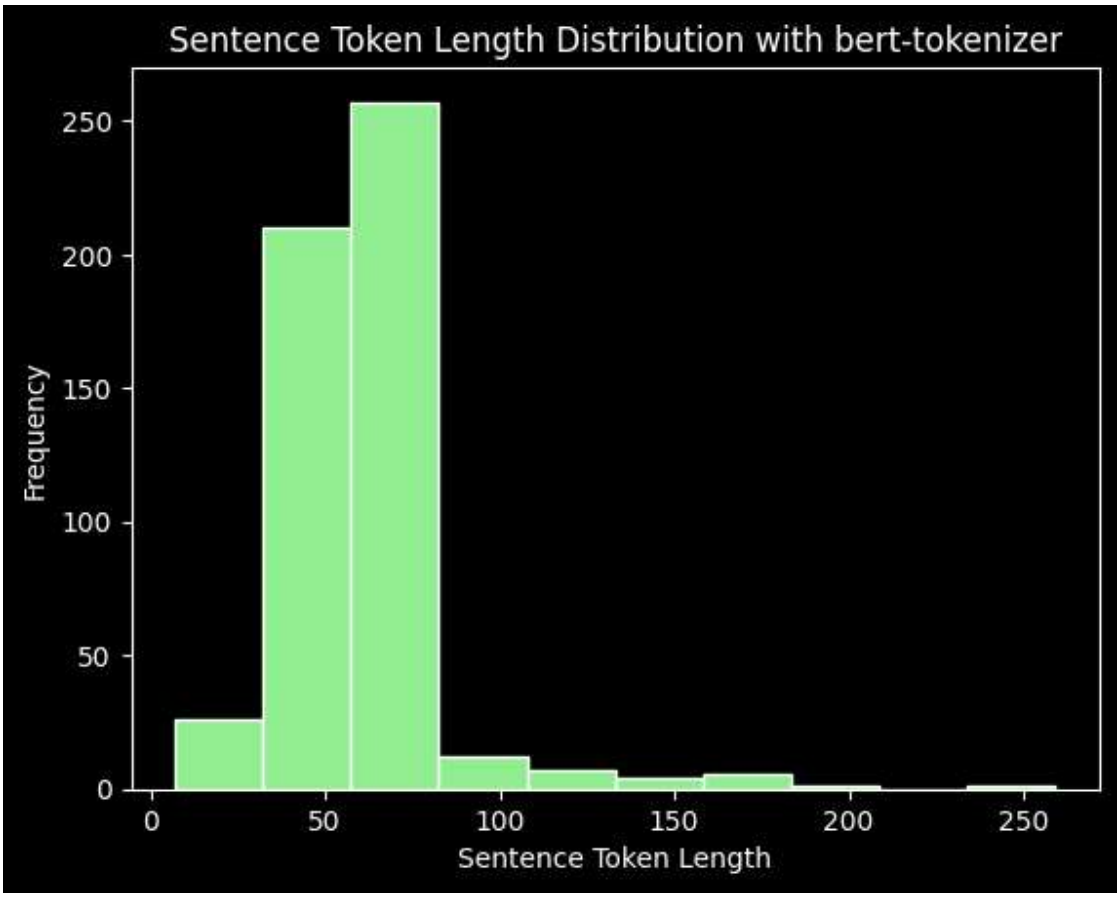
- histtype='bar' specifies the type of histogram as a bar chart.
- color='lightgreen' sets the color of the bars to light green.
- edgecolor='white' defines the color of the edges of the bars.

Setting Title and Labels:

- plt.title sets the title of the plot to 'Sentence Token Length Distribution with bert-tokenizer'.
- plt.xlabel and plt.ylabel label the x-axis and y-axis, respectively, as 'Sentence Token Length' and 'Frequency'.

Displaying the Plot:

- plt.show() is used to display the generated histogram plot.



here the observations are the same i.e low sentence token length has high frequency and high sentence token length has low frequency.

Now lets plot character count:

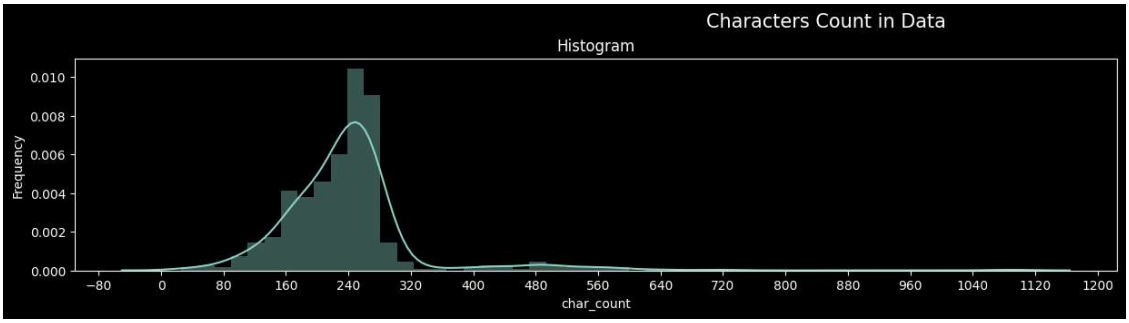
- In this code snippet, I'm effectively managing warnings to ensure a cleaner output. I'm calculating the character count for each review in the dataset and then visualizing the distribution of

these character counts.

```
import warnings
warnings.simplefilter('ignore', category=UserWarning)

# valvulate char count for each review
data['char_count'] = data['text'].apply(lambda x: len(str(x)))
def plot_graph(df,feat,title):
        fig = plt.figure(constrained_layout=True, figsize=(18, 8))
        grid = gridspec.GridSpec(ncols=3, nrows=3, figure=fig)
        ax1 = fig.add_subplot(grid[0, :2])
        ax1.set_title('Histogram')
        sns.distplot(df.loc[:, feat],
                     hist=True,
                     kde=True,
                     ax=ax1,
                     )
        ax1.set(ylabel='Frequency')
        ax1.xaxis.set_major_locator(MaxNLocator(nbins=20))

        plt.suptitle(f'{title}', fontsize=15)
```

- To achieve this, I've created a new column named 'char_count' in the dataset, representing the number of characters in each review. The Seaborn library is employed to generate a histogram and Kernel Density Estimate (KDE) simultaneously. This combined plot provides insights into the frequency distribution of character counts across reviews.

- The plot_graph function streamlines the plotting process. It takes care of setting up the figure, creating the histogram, and placing a Kernel Density Estimate on top. The resulting visualization offers a clear representation of how character counts are distributed, making it easier to understand the overall pattern in the dataset.



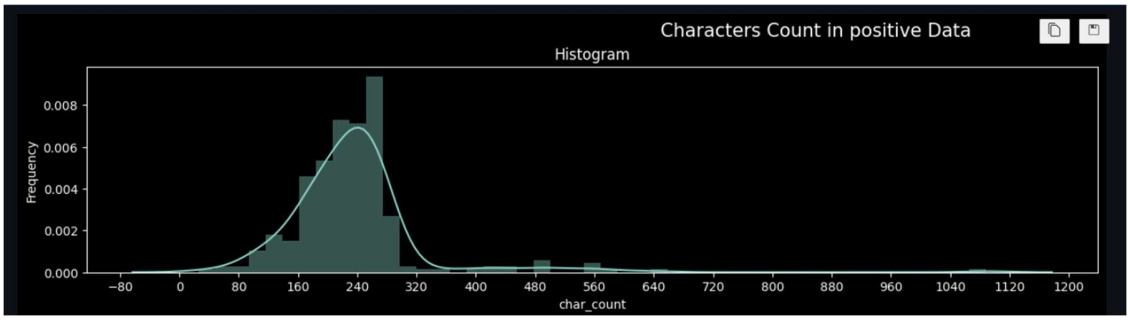
Now lets take a look on our data set after doing all these different -different stuffs.

	Unnamed: 0	text	label	tokenized_review	sent_token_length	sent_bert_token_length	char_count	Character Count	Count
2	182	As a part of its G20 Think Tank Workshop Serie...	Positive	As a part of its G 20 Think Tank Workshop Seri...	43	51	213	213	213
3	403	Don't miss UNA-UK CEO Marissa Conway + @GAPS_N...	Negative	Don t miss UNA UK CEO Marissa Conway G...	64	71	249	249	249
0	140	Saudi Arabia Achieves a Leading Position in th...	Positive	Saudi Arabia Achieves a Leading Position in th...	31	37	142	142	142
1	431	#G20 should promote Liquefied Petroleum Gas (#...	Positive	G 20 should promote Liquefied Petroleum Gas ...	34	45	143	143	143
7	397	With commitment, collaboration, and a comprehe...	Positive	With commitment collaboration and a compre...	54	72	257	257	257

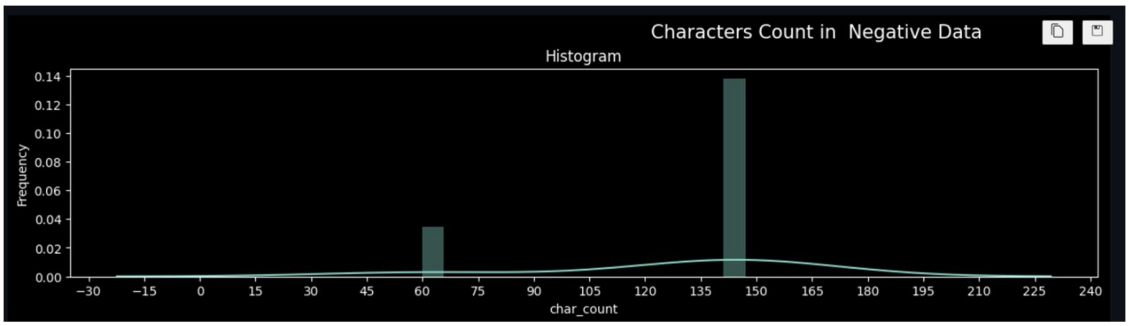
OK, so now I am going to plot above same graph but not for selectively df[df[column=[‘label’]]

```
plot_graph(data[data['label']=='Positive'],"char_count",title='Characters Count in positive Data')

plot_graph(data[data['label']=='Negative'],"char_count",title='Characters Count in negative Data')
plot_graph(data[data['label']=='Positive criticism'],"char_count",title='Characters Count in  Negative Data')
plot_graph(data[data['label']=='Negative criticism'],"char_count",title='Characters Count in Negative criticism Data')
```

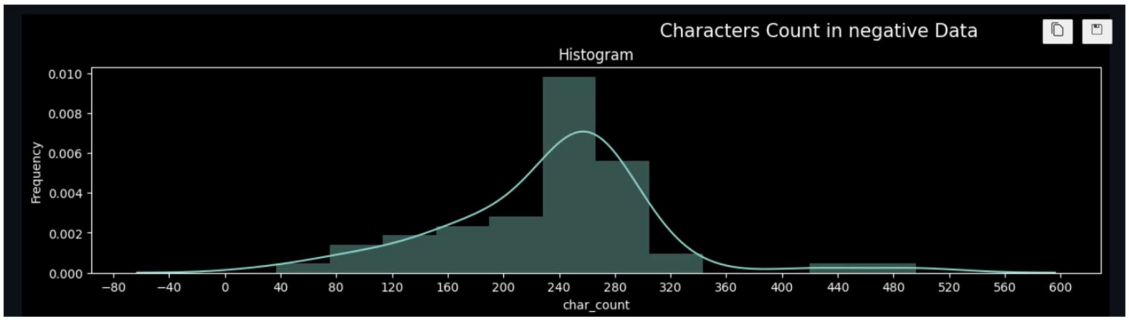
a) First for ‘positive’



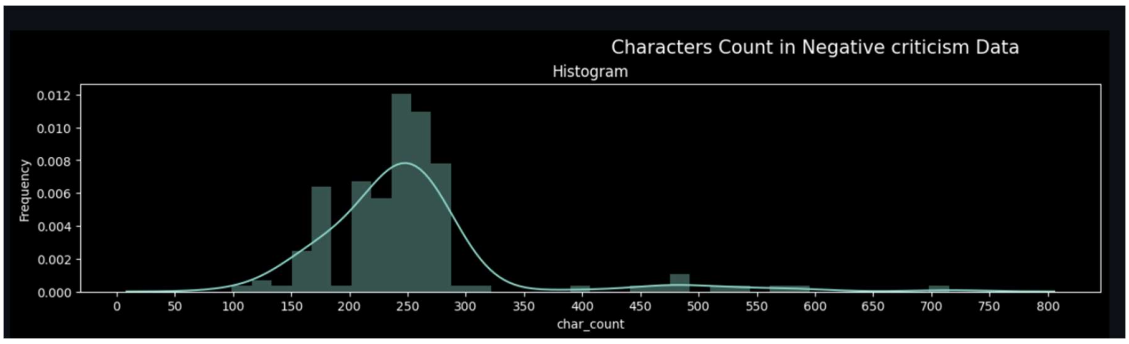
b) Now for ‘second



c) For third label that is ‘negative criticism’



d) For Fourth label that is ‘positive criticism’

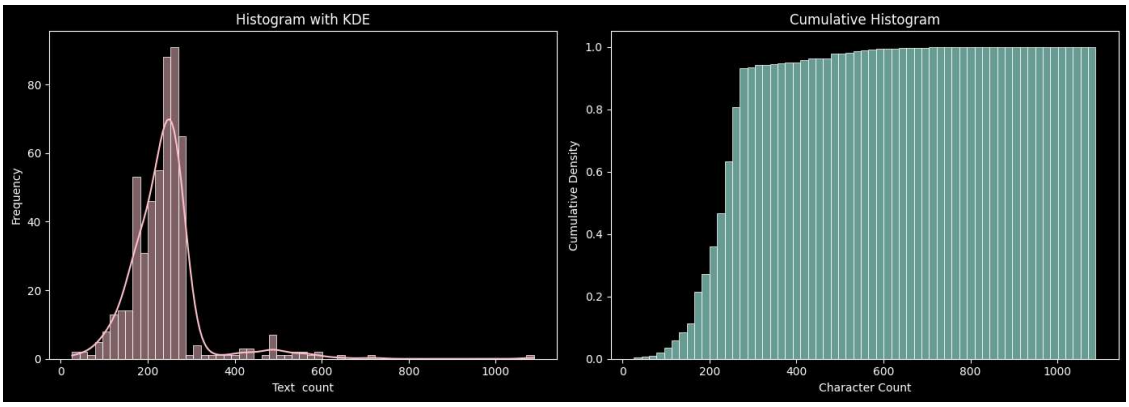


COUNTING THE LENGTH OF THE REVIEW

In this segment of code, I'm exploring and visualizing the distribution of text and character counts in the dataset. First, I've added a new column named 'Count' to capture the total number of words in each text entry. The subsequent creation of a histogram, accompanied by a Kernel Density Estimate (KDE) and cumulative density plot, provides valuable insights into the dataset.

The first subplot, a Histogram with KDE, illustrates the frequency distribution of text counts. The pink color and smooth KDE curve enhance the readability of the plot, showcasing the concentration of text lengths within the dataset.

Moving to the second subplot, a Cumulative Histogram, I've opted for a cumulative density representation of character counts. This plot allows us to observe the cumulative distribution of character lengths in the dataset, providing a holistic perspective on the dataset's overall structure.



By presenting both text and character count distributions, this code aids in understanding the composition and variability of textual data, paving the way for informed analyses and data-driven decisions.

MOST COMMON WORDS:

Here, I'm using the Natural Language Toolkit (nltk) library to do the preprocessing of textual data. Firstly, I download the stopwords dataset from nltk, a vast list of common English words that often do not contribute significant or do not have much semantic meaning in the context of text analysis. These stopwords include articles, prepositions, and conjunctions.

```
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
stopWords_nltk = set(stopwords.words('english'))
```

- After downloading the stopwords, I create a set named 'stopWords_nltk' to efficiently store and access these words. Stopwords play a crucial role in text preprocessing by allowing us to filter out common words that might introduce noise and hinder the extraction of meaningful insights.

- Moving on to the next step, I define a regular expression (regex) to identify and remove punctuation from the tokenized text. Punctuation marks, while prevalent, often lack semantic value and can distort the analysis if not handled appropriately

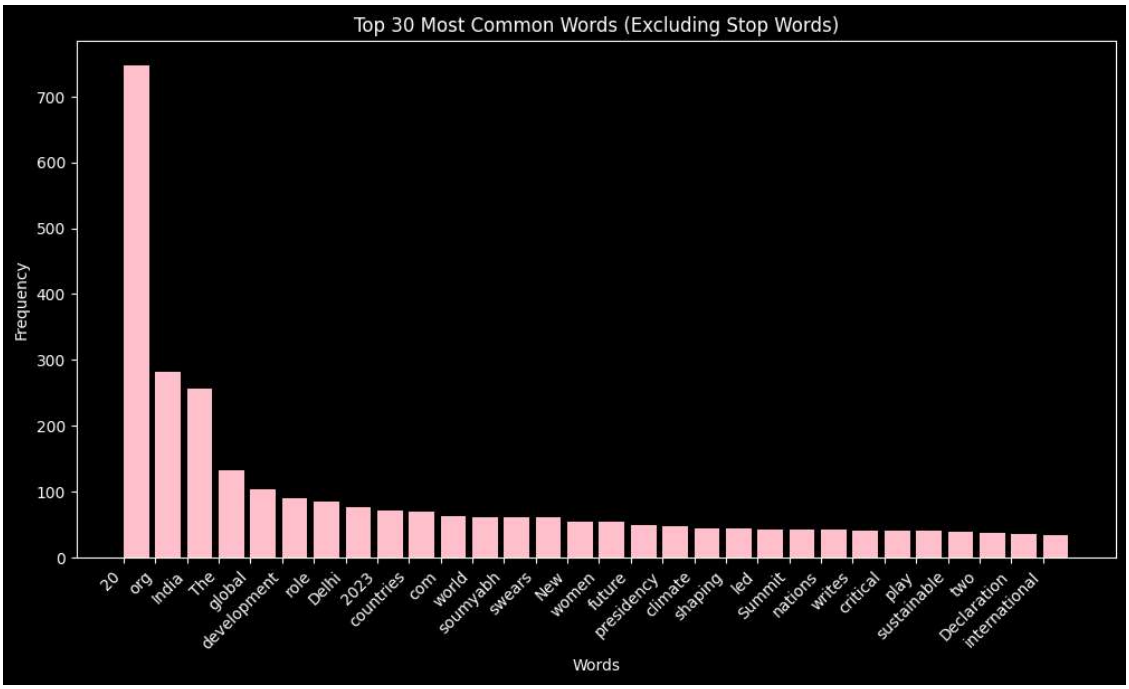
```
# first we have to remove punctation as they might be most common
token in our corpus
# but does not have semantic value so it makes sense to remove them

regex = re.compile('[%s]' % re.escape(string.punctuation))

def remove_punct(text):
    text = regex.sub(" ", text)
    return text

data['tokenized_review'] = data['tokenized_review'].apply(lambda x:
remove_punct(x))
```

- The core function, 'remove_punct,' applies this regex substitution to each tokenized review, effectively eliminating punctuation marks and

ensuring a cleaner and more focused dataset for subsequent analyses.

In summary, this code segment contributes to the initial stages of text preprocessing, aligning the dataset for downstream tasks such as sentiment analysis, topic modeling, or any other Natural Language Processing (NLP) application. By systematically addressing stopwords and punctuation, we lay the foundation for more accurate and insightful analyses of the underlying textual data.

This is the plot of most common 30 words after applying above mentioned techniques:



N-GRAMS

- Here, I'm using the Plotly library to create a subplot of bar charts, aiming to visualize the most common n-grams (word sequences) within different classes of text. The dataset seems to be labeled, with classes like "Positive," "Negative criticism," "Negative," and "Positive criticism."

```
fig = make_subplots(rows=2, cols=2, subplot_titles=["Positive",
"Negative criticism", "Negative", "Positive criticism"])

colors = ['gold', 'mediumturquoise', 'lightcoral', 'lightgreen']
title_ = ["Positive", "Negative criticism", "Negative", "Positive
criticism"]
```

- Firstly, I set up a subplot with two rows and two columns, each representing a different class or category. The subplot titles are appropriately set, distinguishing between positive and negative sentiments and criticisms.

- Next, I iterate through the four classes, extracting the tokenized reviews associated with each. These reviews are then split into individual words,

```
for i in range(4):
    texts = data[data["label"] == title_[i]]['tokenized_review']

    new = texts.str.split()
    new = new.values.tolist()
    corpus = [word for i in new for word in i]
    counter = Counter(corpus)
    most = counter.most_common()
    x, y = [], []

    for word, count in most[:100]:
        if word not in stopWords_nltk and len(word) != 1:
            x.append(word)
            y.append(count)

    fig.add_trace(go.Bar(
        x=y,
        y=x,
        orientation='h',
        type="bar",
        name=title_[i],
        marker=dict(color=colors[i])
```

- forming a corpus. Using a Counter, I tally the occurrences of each word in the corpus.

- To focus on meaningful insights, I filter out stopwords and single-character words. The top 100 most common words (n-grams) for each class are selected based on their frequency.

- For each class, a horizontal bar chart is added to the subplot, where the y-axis represents words,

and the x-axis represents their respective frequencies. Each subplot is assigned a distinct color for better visual clarity.

```
fig.update_layout(
    autosize=False,
    width=1000,
    height=800,
    title=dict(
        text='<b>Most Common ngrams per Classes</b>',
        x=0.5,
        y=0.95,
        font=dict(
            family="Courier New, monospace",
            size=24,
            color="RebeccaPurple"
        )
    ),
)

fig.show()
```

- The final layout adjustments are made to enhance the overall appearance of the visualization, including setting the size, title, and color scheme.

Overall, this code generates an insightful visualization that allows us to compare and contrast the most prevalent words in different classes of text, shedding

light on the key linguistic patterns associated with positive and negative sentiments and criticisms in the dataset.

This completes our data analysis and visualization part.Lets hop onto modelling part

a) Import all the important files- I'm using TensorFlow and Keras to build a neural network for sentiment analysis on Twitter data. The labels are transformed from strings to numerical values. The model includes layers like Dense, Embedding, Flatten, and Dropout. I use tokenization and padding for text processing. The neural network is trained on the dataset, and EarlyStopping is implemented. Overall, it's a setup for a sentiment analysis model using deep learning techniques.

b) Then we splits the data into training and testing sets for machine learning model evaluation. The 'text' column represents the input features, and the 'label' column contains the corresponding class labels. The data is divided into training

(80%) and testing (20%) sets using the train_test_split function from scikit-learn. This separation helps assess the model's performance on unseen data, ensuring a reliable evaluation of its generalization capabilities.

c) As our labels are string so we have to first encoder them and for this usecase we have to use labelencoder. So we employs the LabelEncoder from scikit-learn to convert categorical string labels into numerical values. It fits the encoder on the training set labels (y_train) using fit_transform, and then transforms the test set labels (y_test) using transform. This numeric encoding is essential for training machine learning models, ensuring compatibility with algorithms that require numerical inputs.

d) Then we tokenize both train data and test data. tokenizes the text data using the Tokenizer class from TensorFlow. It fits the tokenizer on the training set (X_train) to create a vocabulary and then converts the text into sequences of numerical indices using texts_to_sequences. This process is crucial for preparing the textual data as input for a neural network, allowing the

model to understand and learn from the sequential structure of the text.

e) Then we pads the sequences to ensure they have the same length. The pad_sequences function is used to add padding to the sequences so that they match the length of the longest sequence in the dataset. This step is essential for creating consistent input shapes for the neural network, as it requires fixed-size input dimensions. The maxlen parameter is set to the maximum length among all sequences in both the training and testing sets.

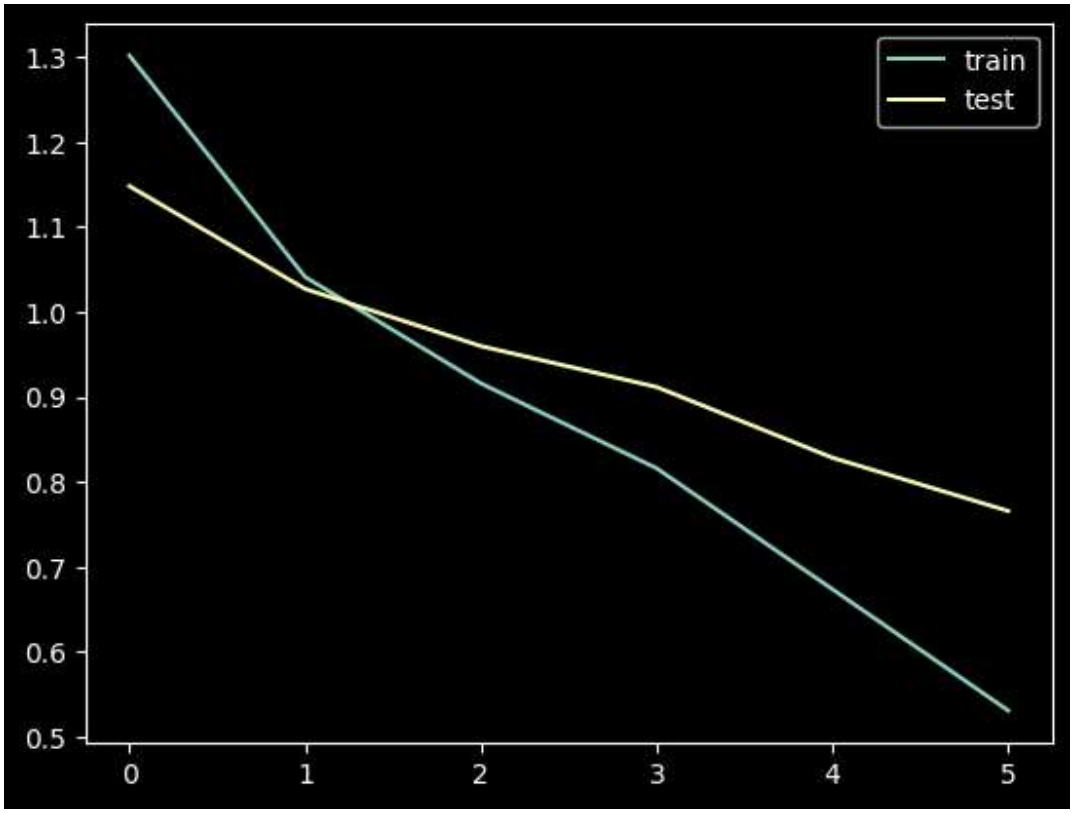
f) Now comes building the model

```
# Build the model
model = Sequential()
model.add(Embedding(input_dim=len(tokenizer.word_index) + 1, output_dim=16, input_length=max_len))
model.add(Flatten())
model.add(Dense(32, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(4, activation='softmax'))

model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# Fit the model
history = model.fit(X_train_padded, y_train_encoded, validation_data=(X_test_padded, y_test_encoded), epochs=6, verbose=1)
```

and as epoch is 6 this loop will run for 6 times.

```
Epoch 1/6
14/14 [==============================] - 5s 202ms/step - loss: 1.3018 - accuracy: 0.5107 - val_loss: 1.1482 - val_accuracy: 0.6095
Epoch 2/6
14/14 [==============================] - 1s 91ms/step - loss: 1.0412 - accuracy: 0.5585 - val_loss: 1.0272 - val_accuracy: 0.6095
Epoch 3/6
14/14 [==============================] - 2s 110ms/step - loss: 0.9160 - accuracy: 0.5847 - val_loss: 0.9599 - val_accuracy: 0.6571
Epoch 4/6
14/14 [==============================] - 1s 93ms/step - loss: 0.8160 - accuracy: 0.6850 - val_loss: 0.9118 - val_accuracy: 0.7333
Epoch 5/6
14/14 [==============================] - 1s 72ms/step - loss: 0.6737 - accuracy: 0.8115 - val_loss: 0.8285 - val_accuracy: 0.7429
Epoch 6/6
14/14 [==============================] - 1s 104ms/step - loss: 0.5310 - accuracy: 0.8687 - val_loss: 0.7658 - val_accuracy: 0.7429
```

We can see our error is continuously decreasing and accuracy simulateneously increasing .
Finally plot a graph of error with epoch of both train data and test data.



So in the graph also error of both train and test data is decreasing similarily.

CONCLUSIONS:

Extracting polarity from a text is a challenging phase. There are many frequent and common words, bigrams and even phrases occurring in the data extracted, which can be treated as the keywords for the G20 Delhi summit 2023. There is a major part of the positive sentiments, followed by the negative criticism, negative and at last the positive criticism thus leading to different clusters of data points. There is a dense net of relationships among all comments and also the most common ones. The public sentiment towards the G20 Summit 2023, as reflected in our report earlier, provides insights into the impact and relevance of the summit. Positive sentiments may indicate approval of the decisions made during the summit, while negative sentiments may reflect dissatisfaction. Neutral sentiments may suggest a more observational or informational stance. There are also both, positive and negative criticisms which indicates that the world population actively shared their own viewpoints. With the help of Data Analysis and visualizations techniques which we mentioned earlier we were able to extract meaningful insights from the huge sea of data which in future can be used for different research purposes and a more detailed analysis of specific topics of

discussions or a comparative analysis of public sentiments. With the use of pre-trained pipelines sometimes there can be misinterpretations. The text data from the YouTube and twitter contain a wide range of nuances and context that can be difficult to capture and analyze. The analysis might not fully account for the variations in texts, such as languages, different dialects or the use of slangs.

The G20 was a successful event with a huge participation from the crowds in form of their comments and tweets. Understanding public sentiments towards this event is one of paramount importance. Such understanding can greatly affect the policy decisions. In addition to it, understanding public sentiments contributes to more informed and engaged public discourse. It encourages open discussions and debates on public views and concerns.