

# YOUTUBE COMMENTS SENTIMENTAL ANALYSIS

## Members List

Amirthavarshini V (2248066)

Sneha S (2248055)

Prakriti Bhatt (2248047)

## ▼ AIM

1. To explore the sentiment and subjectivity of YouTube comments related to Trump's announcement to stand for election in 2024 and identify the dominant emotions and opinions expressed in the comments.

2. To evaluate the effectiveness of using sentiment analysis techniques, specifically polarity and subjectivity analysis, in analyzing YouTube comments related to political announcements and determine the potential use cases for such analysis in political campaigning and public opinion research.

## ▼ ABOUT VIDEO

Title and description: Trump Releases Video Statement In Response To Biden's Just-Announced 2024 Reelection Bid

Length and format: 3.16 minutes

Tone and style: Formal news

Context and timing: Former President Trump derided President Biden, reading from a statement he released in response to the President's just-announced 2024 presidential reelection bid.

## ▼ DATA COLLECTION

We collect the comments from the youtube video [https://www.youtube.com/watch?v=\\_rEtDNMBNh8&t=6s](https://www.youtube.com/watch?v=_rEtDNMBNh8&t=6s). We got the api\_key from google console.

```
import csv from googleapiclient.discovery import build
api_key = 'AlzaSyBCUWd0vob6U5-a3-10IbGrmBjQ7xoJfe4'
def video_comments(video_id):

    # creating youtube resource object
    youtube = build('youtube', 'v3', developerKey=api_key)

    # retrieve youtube video results
    video_response = youtube.commentThreads().list(
        part='snippet',
        videoId=video_id,
        textFormat='plainText'
    ).execute()

    # open a CSV file for writing
    with open('video_comments.csv', 'w', newline='', encoding='utf-8') as csvfile:
        csv_writer = csv.writer(csvfile)

        # write header row
        csv_writer.writerow(['Comment', 'Timestamp', 'Likes'])

        # iterate video response
        while video_response:
```

```

# extracting required info
# from each result object
for item in video_response['items']:
    # Extracting comments
    comment = item['snippet']['topLevelComment']['snippet']['textDisplay']
    # Extracting timestamp
    timestamp = item['snippet']['topLevelComment']['snippet']['publishedAt']
    # Extracting number of likes
    likes = item['snippet']['topLevelComment']['snippet']['likeCount']

    # write comment, timestamp, and likes to CSV file
    csv_writer.writerow([comment, timestamp, likes])

# Get the next page of comments
if 'nextPageToken' in video_response:
    video_response = youtube.commentThreads().list(
        part='snippet',
        videoId=video_id,
        textFormat='plainText',
        pageToken=video_response['nextPageToken']
    ).execute()
else:
    break

```

## ▼ Enter video id

```
video_id = "_rEtDNMBNh8"
```

## Call function

```
video_comments(video_id)
```

```

# utilities
import re
import numpy as np
import pandas as pd
# plotting
import seaborn as sns
from wordcloud import WordCloud
import matplotlib.pyplot as plt
# nltk
import nltk
nltk.download('punkt')
from nltk.corpus import wordnet
from nltk.stem import WordNetLemmatizer
nltk.download('averaged_perceptron_tagger')
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
from nltk import pos_tag
from textblob import TextBlob
from PIL import Image

[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data] Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data] /root/nltk_data...
[nltk_data] Unzipping taggers/averaged_perceptron_tagger.zip.

```

## ▼ DATA EXPLORATION

```
df = pd.read_csv("/content/youtube_comments.csv")
```

```
df.head()
```

	Comment	Timestamp	Likes
0	Yes President Trump we miss you 🇺🇸\nWe will ke...	2023-04-28T01:56:46Z	0
1	Let's Go Brandon...	2023-04-28T01:51:30Z	0
2	You are so right about everything you just sta...	2023-04-28T01:47:20Z	0
3	❤️	2023-04-28T01:41:57Z	0

```
df.columns
```

```
Index(['Comment', 'Timestamp', 'Likes'], dtype='object')
```

```
print('length of data is', len(df))
```

```
length of data is 4903
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4903 entries, 0 to 4902
Data columns (total 3 columns):
#   Column      Non-Null Count  Dtype
---  ---
0   Comment     4902 non-null   object
1   Timestamp   4903 non-null   object
2   Likes       4903 non-null   int64
dtypes: int64(1), object(2)
memory usage: 115.0+ KB
```

## ▼ DATA CLEANING

```
# Convert the Timestamp column to datetime data type
```

```
df['Timestamp'] = pd.to_datetime(df['Timestamp'])
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4903 entries, 0 to 4902
Data columns (total 3 columns):
#   Column      Non-Null Count  Dtype
---  ---
0   Comment     4902 non-null   object
1   Timestamp   4903 non-null   datetime64[ns, UTC]
2   Likes       4903 non-null   int64
dtypes: datetime64[ns, UTC](1), int64(1), object(1)
memory usage: 115.0+ KB
```

```
import re
```

```
# function to remove special characters
```

```
# function to remove special characters
```

```
def remove_special_characters(text):
```

```
    pattern = r'^a-zA-Z0-9\s$' # keep only alphanumeric characters and spaces
```

```
    text = str(text) # convert input to a string
```

```
    text = re.sub(pattern, '', text)
```

```
    return text
```

```
df.loc[:, 'Comment'] = df['Comment'].apply(remove_special_characters)
```

```
df.loc[:, 'Comment'] = df['Comment'].str.replace('\n', '')
```

```
# Dropping the null values
```

```
df.dropna(inplace=True)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4903 entries, 0 to 4902
Data columns (total 3 columns):
#   Column      Non-Null Count  Dtype
---  ---
0   Comment     4903 non-null   object
1   Timestamp   4903 non-null   datetime64[ns, UTC]
2   Likes       4903 non-null   int64
```

```

dtypes: datetime64[ns, UTC](1), int64(1), object(1)
memory usage: 115.0+ KB

# Find the number of duplicate comments
num_duplicates = df.duplicated(subset=['Comment']).sum()

# Drop duplicates
df.drop_duplicates(subset=['Comment'], inplace=True)

df.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 4637 entries, 0 to 4901
Data columns (total 3 columns):
#   Column      Non-Null Count  Dtype
---  ---
0   Comment     4637 non-null    object
1   Timestamp   4637 non-null    datetime64[ns, UTC]
2   Likes       4637 non-null    int64
dtypes: datetime64[ns, UTC](1), int64(1), object(1)
memory usage: 144.9+ KB

# We can see that the no of rows and range is different. So we reset the index
df = df.reset_index(drop=True)

#Finding the top comments
top_comments = df.nlargest(5, 'Likes')
for index, row in top_comments.iterrows():
    print(row['Comment'])

    Think for yourselves people dont let media and your own bias control you
    I dont care about mean tweets I care about America bring back a leader whos got a brain
    I dont think even a mafia run government would make it this horrible
    Imagine a campaign where they actually proposed ideas
    theres a reason biden didnt announce it live

df['Comment'] = df['Comment'].replace('', np.nan)

df["Comment"].isnull().sum()

1

df['Comment'] = df['Comment'].str.strip()
df.dropna(inplace=True)

df.head()

```

	Comment	Timestamp	Likes
0	Yes President Trump we miss you We will keep p...	2023-04-28 01:56:46+00:00	0
1	Lets Go Brandon	2023-04-28 01:51:30+00:00	0
2	You are so right about everything you just sta...	2023-04-28 01:47:20+00:00	0
4	This guy is so much out of touch	2023-04-28 01:34:01+00:00	0
5	1000000 dead and dying Americans due to Trump ...	2023-04-28 01:07:17+00:00	0

## ▼ DATA ENHANCEMENT

```

# Appending the hour column to the df
df['Hour'] = df['Timestamp'].dt.hour + df['Timestamp'].dt.day

from nltk.stem import WordNetLemmatizer
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')

# Define lemmatizer object
lemmatizer = WordNetLemmatizer()

```

```

# Define function to get WordNet part of speech tag
def get_wordnet_pos(tag):
    if tag.startswith('J'):
        return wordnet.ADJ
    elif tag.startswith('V'):
        return wordnet.VERB
    elif tag.startswith('N'):
        return wordnet.NOUN
    elif tag.startswith('R'):
        return wordnet.ADV
    else:
        return wordnet.NOUN

# Function to lemmatize a sentence
def lemmatize_sentence(sentence):
    # Tokenize the sentence into words
    words = nltk.word_tokenize(sentence)
    # Get the part of speech tag for each word
    pos_tags = nltk.pos_tag(words)
    # Lemmatize each word using its part of speech tag
    lemmatized_words = [lemmatizer.lemmatize(word, get_wordnet_pos(tag)) for word, tag in pos_tags]
    # Join the lemmatized words back into a sentence
    lemmatized_sentence = ' '.join(lemmatized_words)
    return lemmatized_sentence

# Apply lemmatization to the comments column
df['Comment'] = df['Comment'].apply(lemmatize_sentence)

[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data] /root/nltk_data...
[nltk_data] Package averaged_perceptron_tagger is already up-to-
[nltk_data] date!
[nltk_data] Downloading package wordnet to /root/nltk_data...

# function to obtain adjectives from tweets
def getAdjectives(Comment):
    Comment = word_tokenize(Comment) # convert string to tokens
    Comment = [word for (word, tag) in pos_tag(Comment)
                if tag == "JJ"] # pos_tag module in NLTK library
    return " ".join(Comment) # join words with a space in between the

# Apply getAdjectives function to the new 'Processed Tweets' column to generate a new column called 'Tweets_Adjectives'
df['Comments_Adjectives'] = df['Comment'].apply(getAdjectives)

# Assuming your comments are in the 'comments' column
df['polarity'] = df['Comment'].apply(lambda x: TextBlob(x).sentiment.polarity)

# Define a function to get the sentiment of a comment
def get_sentiment(comment):
    blob = TextBlob(comment)
    if blob.sentiment.polarity > 0:
        return 'positive'
    elif blob.sentiment.polarity == 0:
        return 'neutral'
    else:
        return 'negative'

# Apply the function to the comments column to create a sentiment column
df['Sentiment'] = df['Comment'].apply(get_sentiment)

def get_subjectivity(comment):
    return TextBlob(comment).sentiment.subjectivity

df['Subjectivity'] = df['Comment'].apply(get_subjectivity)

df.head()

```

	Comment	Timestamp	Likes	Hour	Comments_Adjectives	polarity	Sentiment	Subjectivity
0	Yes President Trump we miss you We will keep p...	2023-04-28 01:56:46+00:00	0	29		0.000000	neutral	0.000000

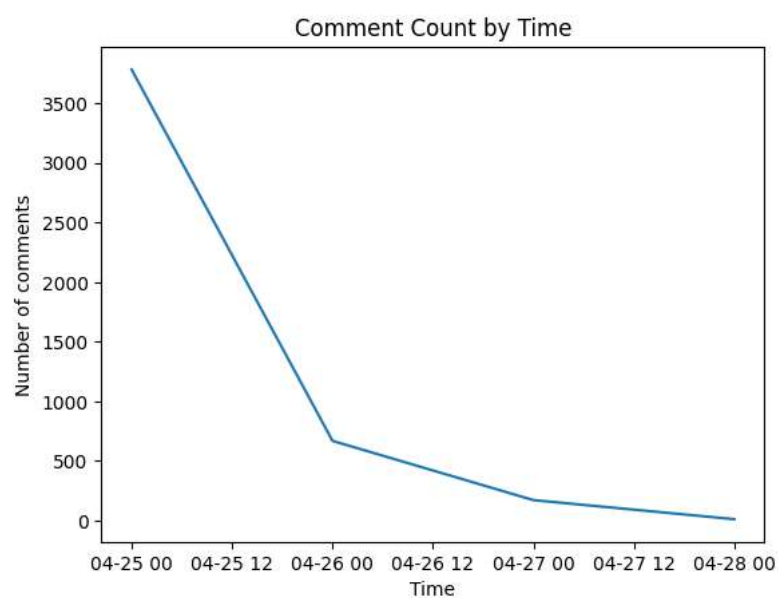
## DATA EXPLORATION

You be so right

## TIMESTAMP

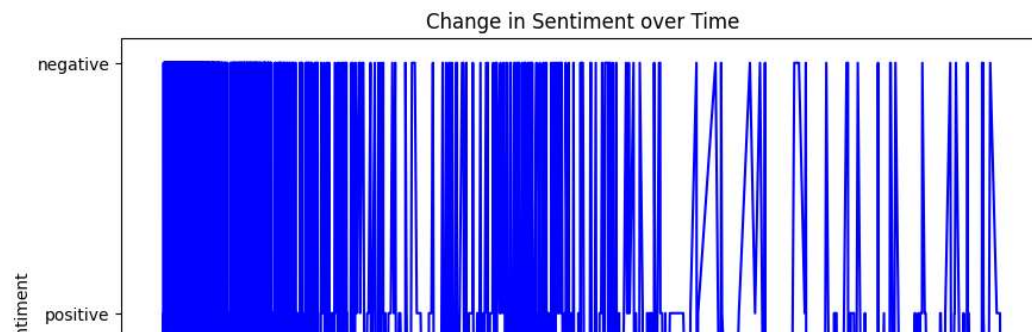
```
# group the data by timestamp and get the count of comments for each timestamp
count_by_time = df.groupby(pd.Grouper(key='Timestamp', freq='D')).count()['Comment']
```

```
# plot the line chart
plt.plot(count_by_time.index, count_by_time.values)
plt.xlabel('Time')
plt.ylabel('Number of comments')
plt.title('Comment Count by Time')
plt.show()
```



```
# Set the index to the 'Timestamp' column
df.set_index('Timestamp', inplace=True)
```

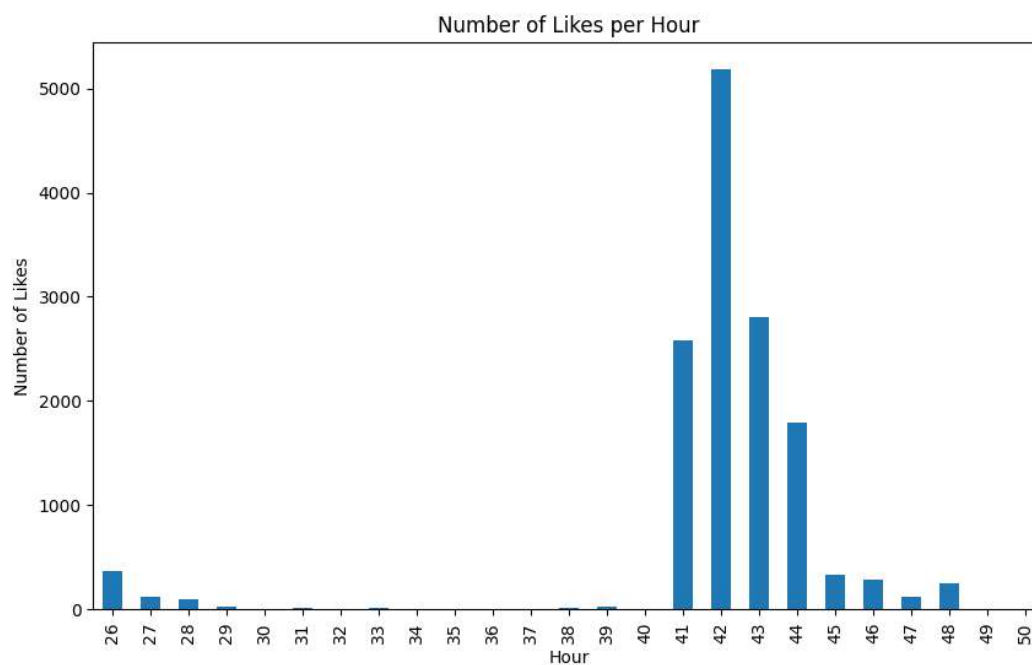
```
# Create a line plot of the 'Sentiment' column over time
plt.figure(figsize=(10, 6))
plt.plot(df['Sentiment'], color='blue')
plt.xlabel('Timestamp')
plt.ylabel('Sentiment')
plt.title('Change in Sentiment over Time')
plt.show()
```



## LIKES



```
# Create a bar plot of the number of likes per hour
likes_by_hour = df.groupby('Hour')['Likes'].sum()
likes_by_hour.plot(kind='bar', figsize=(10, 6))
plt.xlabel('Hour')
plt.ylabel('Number of Likes')
plt.title('Number of Likes per Hour')
plt.show()
```



## Wordcloud

```
from wordcloud import WordCloud, STOPWORDS
stopwords = set(STOPWORDS)
stopwords.update(["trump", "president", "biden", "u"])

# Read the image and convert it to a numpy array
image = Image.open("/content/Trump.png")
image_array = np.array(image)

# Filter the dataframe to keep only the rows with positive sentiment
positive_df = df[df['Sentiment'] == 'positive']

# Concatenate all the comments into a single string
positive_comments = ' '.join(positive_df['Comment'])

# Remove stopwords from the comments
```

```
stopwords = set(STOPWORDS)
stopwords.add('video') # add additional stopwords if needed
positive_comments = ' '.join([word for word in positive_comments.split() if word.lower() not in stopwords])
# Create a mask from the image
mask = image_array.copy()
mask[mask.sum(axis=2) == 0] = 255

# Generate word cloud
wordcloud = WordCloud(background_color="white", max_words=2000, mask=mask, stopwords=STOPWORDS).generate(positive_comments)

# Display the generated image
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```



```
# Concatenate all the adjectives into a single string
adjectives = ' '.join(df['Comments_Adjectives'])

# Generate a word cloud from the concatenated string
wordcloud = WordCloud(background_color='white', stopwords=stopwords, mask = mask).generate(adjectives)

# Display the word cloud
plt.figure(figsize=(12, 10))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```

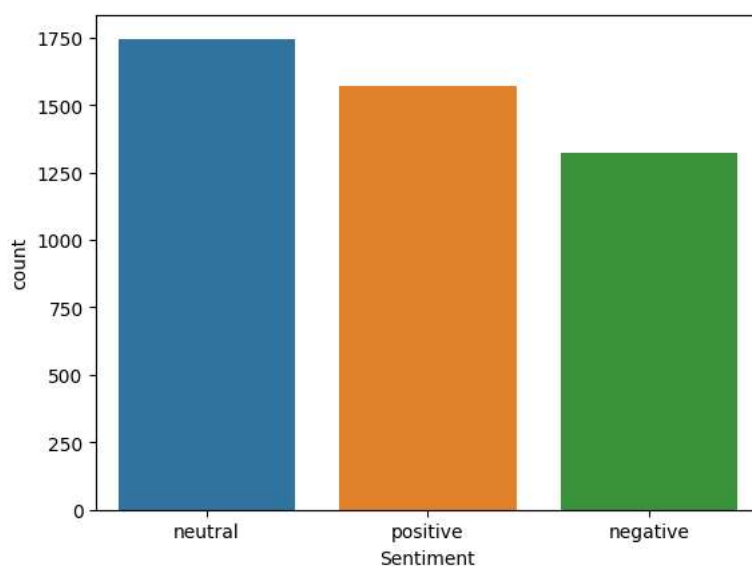




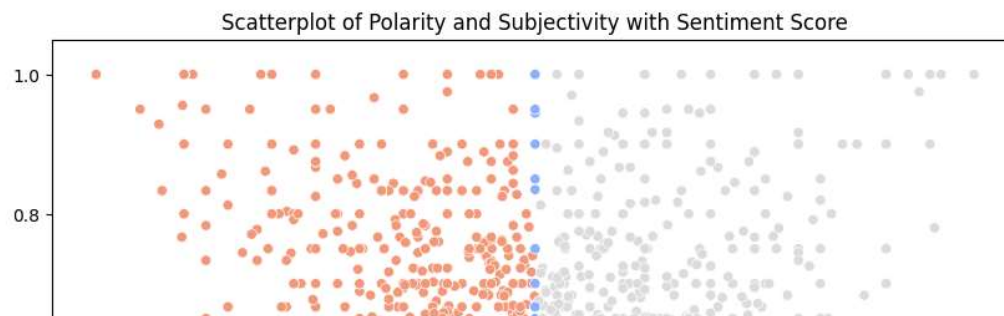
## SENTIMENT ANALYSIS

sns.countplot(x='Sentiment', data=df)

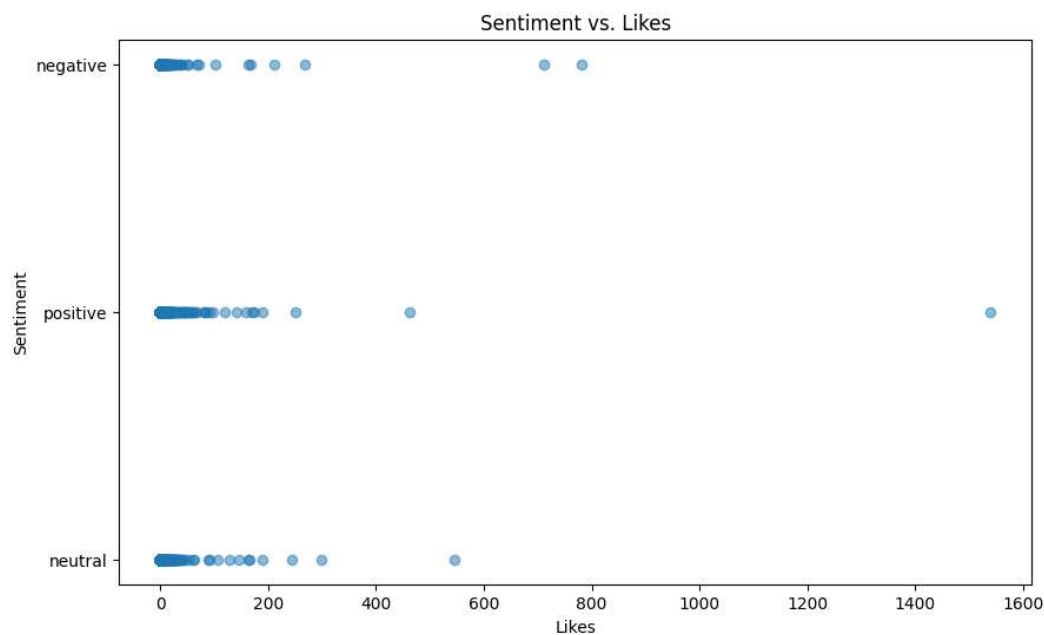
```
sns.countplot(x='Sentiment', data=df)
plt.show()
```



```
plt.figure(figsize=(10, 8))
sns.scatterplot(data=df, x='polarity', y='Subjectivity', hue='Sentiment', palette='coolwarm')
plt.title('Scatterplot of Polarity and Subjectivity with Sentiment Score')
plt.show()
```



```
# Create a scatter plot of sentiment vs. likes
plt.figure(figsize=(10, 6))
plt.scatter(df['Likes'], df['Sentiment'], alpha=0.5)
plt.xlabel('Likes')
plt.ylabel('Sentiment')
plt.title('Sentiment vs. Likes')
plt.show()
```

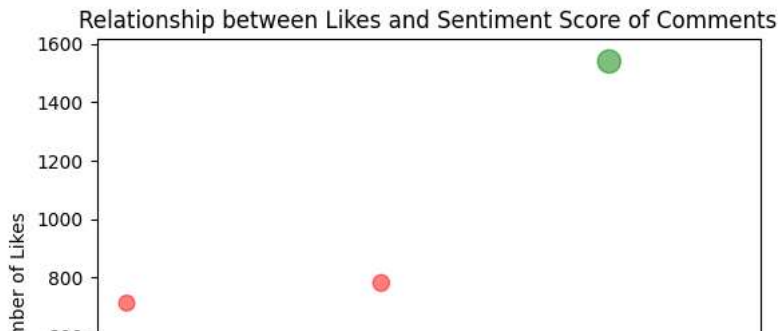


```
# Define a dictionary mapping sentiment strings to colors
color_dict = {'positive': 'green', 'negative': 'red', 'neutral': 'blue'}

# Create the bubble plot
plt.scatter(df['polarity'], df['Likes'], s=df['Likes']/10, c=df['Sentiment'].apply(lambda x: color_dict[x]), alpha=0.5)

# Set axis labels and title
plt.xlabel('Sentiment Score')
plt.ylabel('Number of Likes')
plt.title('Relationship between Likes and Sentiment Score of Comments')

# Show the plot
plt.show()
```



```
# group by hours and sentiment and count number of comments
df_sentiment = df.groupby(['Hour', 'Sentiment']).size().reset_index(name='count')

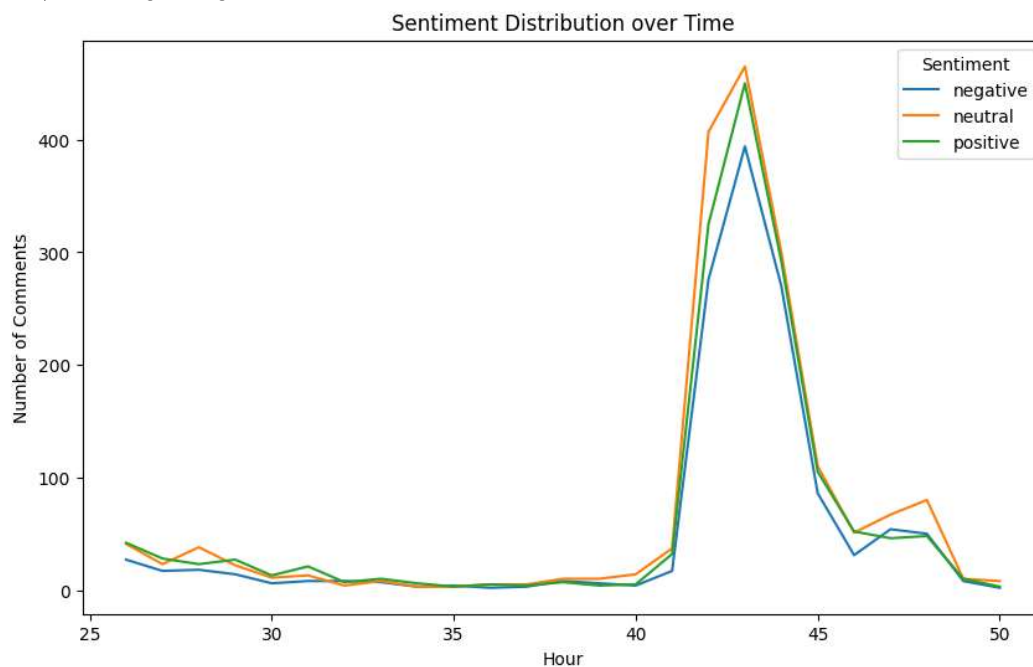
# pivot the table to have sentiment values as columns
df_sentiment = df_sentiment.pivot(index='Hour', columns='Sentiment', values='count')

# plot line chart
ax = df_sentiment.plot(kind='line', figsize=(10, 6), xlabel='Hour', ylabel='Number of Comments')

# set title
ax.set_title('Sentiment Distribution over Time')

# add legend
ax.legend(title='Sentiment')
```

<matplotlib.legend.Legend at 0x7ff42b046200>



## ▼ INFERENCE

From the above wordcloud and the plots, we can infer that the sentiment towards the trump speech is negative slightly greater than neutral. The comments section has so many vulgar use of languages reflecting the anger of people and their fights in comments. But we can see that neutral comments are also high. The comparison between the countries were also high. The average polarity is 0.009 which is close to zero.

