HSNC UNIVERSITY, MUMBAI
School of Applied Sciences
Data Science and Business Analytics

CERTIFICATE

This is to certify that the project "Case study on google pixel 9" carried by a group of four students during the academic year 2024-2025

The team comprises of:

Complete names of the group members

1. Rodrigues Shannan(10)
2. Ujjainwala Hussain(62)
3. Sayyed Moaviya(48)
4. Chheda Yash(11)

This work is best of our knowledge and Belief

Assistant Professor
(Data Science and Business Analytics)

Ms.Shweta Maitri

INDEX

Introduction

Sentiment Analysis is a technique used in natural language processing (NLP) and text analysis to identify and extract subjective information from source materials. In simpler terms, it's the process of determining the emotional tone behind a series of words, used to gain an understanding of the attitudes, opinions, and emotions expressed within an online mention

1. Purpose: It aims to determine whether the attitude towards a particular topic, product, or service is positive, negative, or neutral.

2. Applications: It's widely used in various fields, including:
- Business: To understand customer opinions about products or services
- Politics: To gauge public opinion on policies or candidates
- Social media monitoring: To track brand reputation and customer feedback
- Market research: To analyze consumer attitudes and trends

2. Techniques in Sentiment Analysis

2.1 Rule-Based Approaches

- Uses pre-defined rules to classify text
- Often involves lexicons (lists of words) with associated sentiment scores
- Example: VADER (Valence Aware Dictionary and sEntiment Reasoner)
- Pros: Interpretable, doesn't require training data
- Cons: Can be inflexible, requires manual creation of rules

2.2 Machine Learning Approaches

2.2.1 Supervised Learning

- Requires labeled training data
- Common algorithms: Naive Bayes, Support Vector Machines, Random Forests
- Deep Learning models: Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM) networks

2.2.2 Unsupervised Learning

- Doesn't require labeled data
- Uses techniques like clustering to group similar sentiments

2.3 Hybrid Approaches

- Combines rule-based and machine learning methods
- Aims to leverage strengths of both approaches

3. Levels of Analysis

3.1 Document-Level

- Classifies the sentiment of an entire document
- Assumes the document expresses opinions on a single entity

3.2 Sentence-Level

- Determines the sentiment of each sentence
- More granular than document-level analysis

3.3 Aspect-Based

- Identifies sentiments towards specific aspects of entities
- Example: In a restaurant review, separate sentiments for food, service, ambiance

4. Advanced Concepts

4.1 Emotion Detection

- Goes beyond positive/negative classification
- Identifies specific emotions: joy, anger, sadness, fear, etc.
- Often uses emotion lexicons or machine learning models trained on emotion-labelled data

4.2 Sarcasm Detection

- Aims to identify sarcastic statements which may invert the apparent sentiment

- Challenging due to subtle linguistic cues and context dependence
- Often requires advanced NLP techniques and consideration of broader context

4.3 Multilingual Sentiment Analysis

- Extends sentiment analysis to multiple languages
- Challenges include varying linguistic structures and cultural differences in expressing sentiment

5. Applications

5.1 Business and Marketing

- Brand monitoring
- Product feedback analysis
- Customer service optimization

5.2 Politics and Social Sciences

- Public opinion analysis
- Political campaign strategy
- Social media trend analysis

5.3 Healthcare

- Patient feedback analysis
- Mental health monitoring through social media

5.4 Finance

- Stock market prediction based on news sentiment
- Risk assessment in financial documents

6. Challenges in Sentiment Analysis

6.1 Context and Ambiguity

- Words can have different meanings in different contexts
- Requires sophisticated algorithms to understand context

6.2 Negation Handling

- Negations can invert sentiment
- Simple approaches might misclassify "not bad" as negative

6.3 Domain Specificity

- Sentiment indicators can vary across domains
- Models often need to be tailored for specific industries or topics

6.4 Subjectivity and Tone

- Distinguishing between objective and subjective text
- Identifying subtle tones like sarcasm or irony

7. Future Directions

- Integration with other NLP tasks (e.g., named entity recognition)
- Real-time sentiment analysis for streaming data
- Multimodal sentiment analysis (combining text, speech, and visual data)
- Improved handling of context and implicit sentiment

8. Ethical Considerations

- Privacy concerns in analyzing personal expressions
- Potential for misuse in surveillance or manipulation
- Bias in training data and algorithms

Objective

The primary objective of this project is to collect and analyze Twitter data related to specific keywords, with a focus on understanding public sentiment. By scraping tweets using targeted queries, such as mentions of "Pixel 9," the project aims to measure how users are discussing the product on social media. This data will be stored in a structured format, such as CSV, and used to identify trends in public sentiment, track engagement metrics like retweets and likes, and understand the broader conversation around the topic.

The main focus of the analysis will be sentiment classification of the tweets. Using natural language processing (NLP) techniques, the content of each tweet will be categorized into positive, negative, or neutral sentiments. This will provide insights into how well-received the product or topic is, allowing businesses or stakeholders to assess user satisfaction and identify areas that may need improvement. By tracking sentiment over time, the analysis will also reveal changes in public opinion, providing valuable feedback for future strategies.

Scraping

Web scraping is the process of automatically extracting data from websites. It allows users to gather large volumes of information from web pages in a structured format, which can be used for analysis, reporting, or integration into other applications. The extracted data can range from text, images, and links to more specific information such as prices, reviews, or, in this case, social media posts like tweets. This method is widely used in industries ranging from e-commerce, where it helps track competitor pricing, to research, where it gathers data for analysis.

In the context of this project, web scraping focuses on collecting tweets related to specific keywords. By automating the process of retrieving data from Twitter, this technique enables efficient data collection over time, which can be stored and analyzed further. Using libraries that handle requests to websites or APIs, web scraping scripts mimic a user browsing the web, retrieving the necessary data without manual effort. This method is particularly useful for sentiment analysis, where large volumes of data are needed to understand patterns and trends in public opinion.

```python
import asyncio
from fake_useragent import UserAgent

from twikit import Client

async def main():
    ua = UserAgent()

    client = Client(user_agent = str(ua.chrome), language ='en-US')

    # Log in to your Twitter account
    await client.login(
        auth_info_1='',
        auth_info_2='',
        password=''
    )
    client.save_cookies('cookies.json')

if __name__ == "__main__":
    asyncio.run(main())
```

test.py - Testing Environment for Web Scraping

Test.py file is a small script designed to establish a connection with Twitter through a client and simulate logging in using Twitter's API or a similar service, i.e twikit, which handles the Twitter API interaction.

- Client Setup: The script uses the twikit library to create a client object for interacting with Twitter's API.
- User-Agent Spoofing: The script imports the fake_useragent module to generate a random user agent using UserAgent(), allowing it to simulate a web browser. This step is important to avoid detection when scraping data, as many websites block requests from suspicious or bot-like user agents.
- Async Functionality: Twitter's API can throttle requests, meaning that the script needs to efficiently handle asynchronous tasks. The function async def main() is used for non-blocking login operations.
- Login: The script attempts to log in to Twitter, by using the following credentials (auth_info_1, auth_info_2, and password)
- Cookies Management: Once logged in, the session cookies are saved into a file called cookies.json, allowing the main script to reuse these cookies in subsequent requests.

This test script ensures that the login and client configuration works as expected without hitting rate limits or other restrictions.

```python
from twikit import Client, TooManyRequests
import time
from datetime import datetime
import csv
from random import randint

MINIMUM_TWEETS = 1000
QUERY = '(pixel) (#pixel9) lang:en'

def get_tweets(tweets):
    if tweets is None:
        print(f'{datetime.now()} - Getting tweets...')
        tweets = client.search_tweet(QUERY, product='Latest')
    else:
        wait_time = randint(5, 10)
        print(f'{datetime.now()} - Getting next tweets after {wait_time} seconds ...')
        time.sleep(wait_time)
        tweets = tweets.next()

    return tweets
```

```python
with open('tweets4.csv', 'w', newline='',encoding='utf-8') as file:
    writer = csv.writer(file)
    writer.writerow(['Tweet_count', 'Username', 'Text', 'Created At', 'Retweets', 'Likes'])

client = Client(language='en-US')
client.load_cookies('cookies.json')

tweet_count = 0
tweets = None

while tweet_count < MINIMUM_TWEETS:

    try:
        tweets = get_tweets(tweets)
    except TooManyRequests as e:
        rate_limit_reset = datetime.fromtimestamp(e.rate_limit_reset)
        print(f'{datetime.now()} - Rate limit reached. Waiting until {rate_limit_reset}')
        wait_time = rate_limit_reset - datetime.now()
        time.sleep(wait_time.total_seconds())
        continue

    if not tweets:
        print(f'{datetime.now()} - No more tweets found')
        break

    for tweet in tweets:
        tweet_count += 1
        tweet_data = [tweet_count, tweet.user.name, tweet.text, tweet.created_at, tweet.retweet_count, tweet.favorite_count]

        with open('tweets4.csv', 'a', newline='',encoding="UTF-8") as file:
            writer = csv.writer(file)
            writer.writerow(tweet_data)

    print(f'{datetime.now()} - Got {tweet_count} tweets')
```

main.py - Main Scraping Logic

- The main.py file is the core script responsible for scraping tweets based on a specific query and saving the results into a CSV file.
- Query Definition: The query (pixel) (#pixel9) lang:en is set to search for tweets containing "pixel" or "#pixel9" in English (lang:en). The search is configured to retrieve the latest tweets, likely using the client.search_tweet method from the twikit library.
- Fetching Tweets: The function get_tweets(tweets) is responsible for sending the search request and returning a batch of tweets. It implements a waiting period between successive requests (5 to 10 seconds) to avoid exceeding the API rate limits. The tweets.next() method indicates pagination, where additional tweets are fetched when available.
- CSV File Writing: The script initializes a CSV file called tweets4.csv to store the tweet data. Each tweet is logged with details like username, tweet text, creation date, retweets, and likes. The data is appended to the CSV file incrementally as more tweets are scraped.
- Rate Limiting Handling: The script gracefully handles Twitter's API rate limits using a try-except block. When the TooManyRequests exception is raised, the script waits until the rate limit reset time before continuing to scrape more tweets.

- Looping and Termination: The script runs in a loop, fetching and writing tweets until it reaches a predefined minimum count (MINIMUM_TWEETS = 1000). After all the tweets are collected, the script terminates with a success message.

About The Data

The data is collected from a scraping session, which provides insight into Twitter activity related to a google Pixel 9. Below is a description of the expected columns and the data:

Tweet_count: This is a sequential number assigned to each tweet as it is scraped. It indicates the order in which tweets were collected during the scraping session.

Username: The `Username` field contains the Twitter handle of the user who posted the tweet. This data is useful for identifying the source of the tweet and performing further analysis on specific users or patterns of behaviour.

Text: The `Text` column holds the content of the tweet. This is a key field for text analysis tasks like sentiment analysis, topic modeling, or simply understanding what users are saying about "pixel" or "#pixel9". This column also contain hashtags, mentions, or links that can provide additional information.

Created At: This field captures the timestamp of when the tweet was posted. The `Created At` column allows analysts to examine trends over time, such as peak tweeting times or the popularity of a topic during specific days or events.

Retweets: The number of retweets a tweet has received. This is an important indicator of how far-reaching or viral a tweet was. Retweet counts can also help identify influential users whose tweets are widely shared within the Twitter community.

Likes: The `Likes` column reflects the number of likes or "favourites" that a tweet has garnered. High numbers of likes usually indicate that the tweet resonated with many users or contains content that users found valuable or entertaining.

Preprocessing

Preprocessing is a crucial step in sentiment analysis, especially when working with raw text data such as tweets. The goal is to clean and prepare the data to improve the performance of machine learning models. Text data often contains noise such as special characters, inconsistent formatting, or irrelevant information, and these need to be standardized or removed before analysis.

Preprocessing Steps:

1. Lowercasing:
    - The text is first converted to lowercase using text.lower(). This ensures uniformity, so words like "Pixel" and "pixel" are treated the same during analysis.
2. Removing URLs:
    - The regular expression re.sub(r"http\S+|www\S+|https\S+", "", text, flags=re.MULTILINE) removes URLs from the text. URLs don't contribute to sentiment or keyword analysis, so they are excluded to reduce noise.
3. Removing Mentions and Hashtags:
    - The script removes Twitter-specific mentions (e.g., @username) and hashtags using re.sub(r'@\w+|#\w+', '', text). This helps focus on the core content of the tweet rather than metadata.
4. Removing Punctuation:
    - The script eliminates all punctuation using text.translate(str.maketrans('', '', string.punctuation)). Punctuation generally does not add meaning in most NLP tasks, so it is often removed.
5. Tokenizing Text:
    - The function word_tokenize(text) splits the text into individual words or tokens. Tokenization is essential for applying further transformations like lemmatization and stopword removal.
6. Stopword Removal and Lemmatization:
    - After tokenizing the text, the script removes common stopwords (e.g., "the," "and") that don't contribute meaningfully to the sentiment or context of the tweet.

    - Lemmatization (lemmatizer.lemmatize(word)) reduces words to their base form (e.g., "running" to "run") to ensure consistency across different forms of the same word.
7. Rejoining the Processed Text:
    - Finally, the cleaned tokens are rejoined into a single string using ' '.join(words) to store the cleaned tweet in the dataset.

Sentiment Analysis

Sentiment analysis is a process used to determine the emotional tone behind a piece of text. It involves classifying the sentiment as positive, negative, or neutral, often utilising natural language processing (NLP) and machine learning techniques. This analysis is commonly applied to social media, reviews, and surveys to gauge public opinion, customer satisfaction, or emotional responses to events or products. Essentially, it helps businesses and researchers understand how people feel about specific topics or brands.

The key aspect of sentiment analysis is to analyze a body of text for understanding the opinion expressed by it and other factors like mood and modality. Usually sentiment analysis works best on text that has a subjective context than on that with only an objective context. This is because when a body of text has an objective context or perspective to it, the text usually depicts some normal statements or facts without expressing any emotion, feelings, or mood. Subjective text contains text that is usually expressed by a human having typical moods, emotions, and feelings. Sentiment analysis is widely used, especially as a part of social media analysis for any domain, be it a business, a recent movie, or a product launch, to understand its reception by the people and what they think of it based on their opinions or, you guessed it, sentiment.

The goal that sentiment mining tries to gain is to analyse people's opinions in a way that can help businesses expand. It focuses not only on polarity(positive,negative & neutral) but also on emotions(happy,sad,angry etc)

In the analysis there are three methods for the google pixel 9 data which are:
1.WordCloud
2. Tf-Idf
3.TextBlob

1.WordCloud

A word cloud is a visual representation of text data where the size of each word indicates its frequency or importance within the dataset. Larger words appear more frequently, while smaller words are less common. Word clouds are often

used in data visualization to quickly convey key themes or topics in a body of text, such as customer feedback, survey responses, or social media posts.

The creation of a word cloud typically involves the following steps:

1. Text Processing: Cleaning and preprocessing the text to remove common stop words (like "and," "the," etc.), punctuation, and irrelevant information.
2. Frequency Calculation: Counting the occurrences of each unique word.
3. Visualization: Displaying the words in a visually appealing format, often using different fonts and colors.

Word clouds are useful for quickly identifying trends and focal points in large sets of textual data, making complex information more accessible and engaging.

Word cloud is used to:

- Get insight into the most popular concepts or to reveal sentiment.
- It provides instant analysis & visualization of word data and feedback.
- It engages & sparks excitement among participants.
- It can be used as a quick summary slide or take away after a lesson or meeting, aiding in meeting & class retention.
- Start friendly debates.
- Get some demographic from your participants by asking questions such as where they are from, what company do they represent, where do they currently study, etc.

Code:

```
wordcloud = WordCloud(width=800, height=500, max_font_size=110, collocations=False).generate(all_words)
plt.figure(figsize=(10, 7))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis('off')
plt.title('WordCloud of Google Pixel Tweets')
plt.show()
```

Creating the Word Cloud:

`WordCloud(...)`: This initializes a new word cloud object with specific parameters:

- **`width=800`**: Sets the width of the word cloud image to 800 pixels.
- **`height=500`**: Sets the height of the word cloud image to 500 pixels.
- **`max_font_size=110`**: Specifies the maximum font size for the most frequent word.
- **`collocations=False`**: Prevents the word cloud from considering common phrases (collocations) as single words.

`.generate(all_words)`: This method generates the word cloud from the text data contained in the variable `all_words`, which should be a string of words.

Setting Up the Plot:

`plt.figure(figsize=(10, 7))`: This creates a new figure for the plot with a size of 10 inches in width and 7 inches in height.

Displaying the Word Cloud:

`plt.imshow(wordcloud, interpolation="bilinear")`: This displays the generated word cloud as an image. The `interpolation="bilinear"` argument helps to smooth the image, making it look nicer.

Removing Axes:

`plt.axis('off')`: This hides the axes of the plot, focusing attention on the word cloud itself rather than any grid lines or ticks.

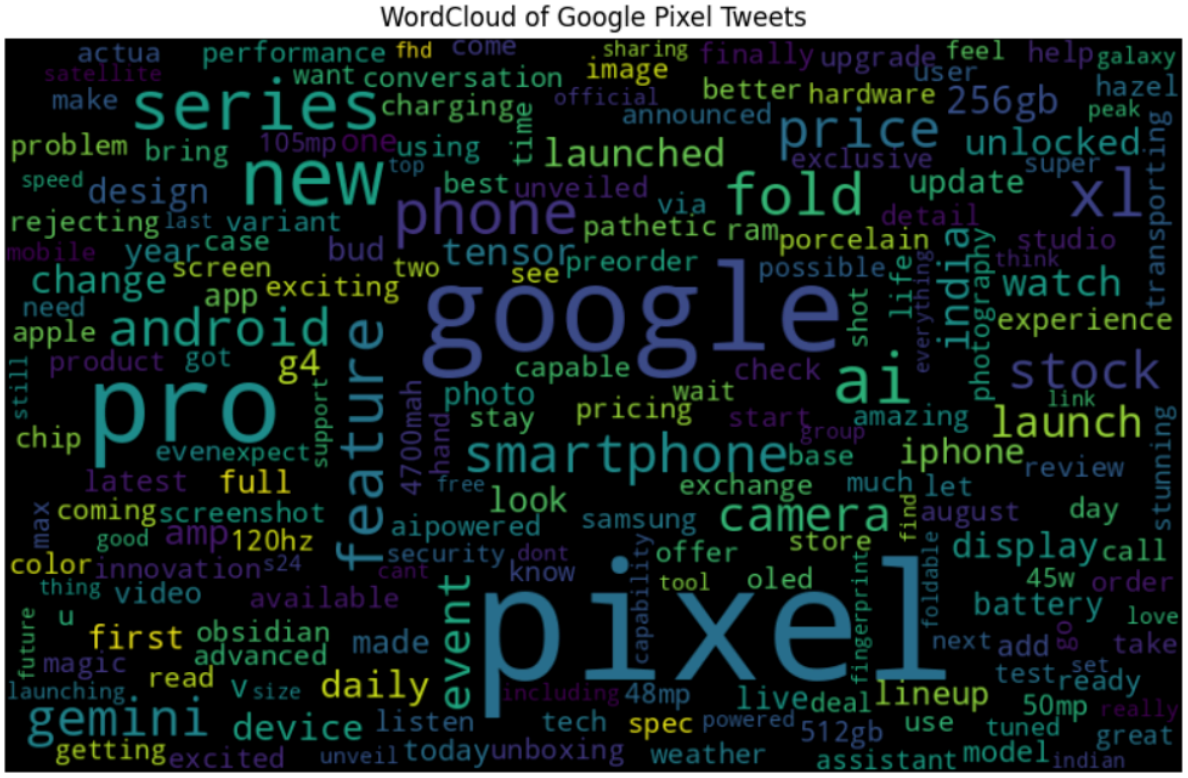
Adding a Title:

`plt.title(...)`: This adds a title to the plot, labeling it as “WordCloud of Google Pixel Tweets.”

Displaying the Plot:

`plt.show()`: This command renders and displays the figure with the word cloud in a window.

Output:



Interpretation:

Here in the wordcloud the word pixel has appeared most of the times in the data so this word has more importance in the google pixel data.

2.Tf-Idf

TF-IDF is the importance of a term is inversely related to its frequency across documents.TF gives us information on how often a term appears in a document and IDF gives us information about the relative rarity of a term in the collection of documents.

TF-IDF (Term Frequency-Inverse Document Frequency) is a statistical measure used to evaluate the importance of a word in a document relative to a collection of documents (corpus). It helps in text analysis and information retrieval, particularly in applications like search engines and text mining. Here's a brief breakdown of the two components:

1. Term Frequency (TF)

TF measures how frequently a term appears in a document. It is calculated as:

$$\text{TF}(t, d) = \frac{\text{Number of times term } t \text{ appears in document } d}{\text{Total number of terms in document } d}$$

This gives higher weight to terms that are more common in a specific document.

2. Inverse Document Frequency (IDF)

IDF measures how important a term is across a collection of documents. It helps reduce the weight of common terms that appear in many documents. It is calculated as:

$$\text{IDF}(t) = \log\left(\frac{N}{df(t)}\right)$$

Where:

- N is the total number of documents in the corpus.
- $df(t)$ is the number of documents containing the term t.

A high IDF score indicates that the term is rare in the corpus, while a low score indicates it is common.

TF-IDF Calculation

The TF-IDF score is computed by multiplying the TF and IDF values:

Formula for Tf-idf:

```
tf-idf(t, d) = tf(t, d) * idf(t)
```

Applications

- **Information Retrieval**: Enhancing search engine relevance.
- **Text Classification**: Helping categorize documents based on content.
- **Recommender Systems**: Improving recommendations based on user preferences.

The higher the TF-IDF score the more important or relevant the term is; as a term gets less relevant, its TF-IDF score will approach towards 0.

Overall, TF-IDF is a powerful technique for transforming text data into a meaningful representation that can be used for further analysis and modeling.

Code:

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer


tfidf_vectorizer=TfidfVectorizer()
tfidf_matrix=tfidf_vectorizer.fit_transform(tweets_df['cleaned_tweet'])


tfidf_df=pd.DataFrame(tfidf_matrix.toarray(),columns=tfidf_vectorizer.get_feature_names_out())

mean_tfidf=tfidf_df.mean().sort_values(ascending=False)

print(mean_tfidf)
```

In the above code to find tf-idf value we first imported all the necessary library which is TfidfVectorizer then we have just created a TfidfVectorizer object.After creating the object we have created a dataframe of the data and then

used to find mean in the ascending order and just print the values of Tf-idf scores.

Output:

```
pixel     0.094747
google    0.069618
pro       0.068665
new       0.033640
series    0.030181
             ...
f168      0.000120
f17       0.000120
f22       0.000120
105       0.000120
ga        0.000120
Length: 2803, dtype: float64
```

Interpretation:
Pixel (0.094747): This term has a relatively high TF-IDF score, which implies that "pixel" is significant in the document(s) where it appears. It is likely to be a key term in the context of the documents it appears in.

Google (0.069618): Similarly, "google" also has a high TF-IDF score, indicating its importance in the document(s). It might be a central topic or a key term related to the document's content.

Pro (0.068665): The term "pro" has a high score as well, suggesting it is also significant in the context of the documents.

New (0.033640): This term has a lower TF-IDF score compared to the others mentioned, suggesting it is less distinctive or less important in the context of the document(s) where it appears.

Series (0.030181): The term "series" also has a relatively low TF-IDF score, indicating it is less significant in distinguishing the document(s) within the corpus.

f168, f17, f22, 105, ga (0.000120): These terms have very low TF-IDF scores, indicating that they are either very common across many documents (such as numeric or categorical identifiers) or are not very relevant in the specific documents. They might not provide much insight into the content of the documents.

3.TextBlob

TextBlob is a Python library used for processing textual data. When it comes to sentiment analysis, TextBlob assigns values to reflect the sentiment of the text.

TextBlob's sentiment analysis yields two main values:

1.Polarity
2.Subjectivity

1.Polarity
It determines the sentiment of the text.The polarity score ranges from -1 to 1.

Interpretation:
-1.0: Extremely negative sentiment
0.0: Neutral sentiment
1.0: Extremely positive sentiment
.
2.Subjectivity
Subjectivity score, on the other hand, goes from 0 to 1. If it’s close to 1, it means the sentence has a lot of personal opinion instead of just facts.

Interpretation:
Close to 0.0: Very objective, fact-based
Values are in between: neutral subjectivity
Close to 1.0: Very subjective, opinion-based

Code:

```
def get_sentiment(text):
    analysis = TextBlob(text)
    return analysis.sentiment.polarity, analysis.sentiment.subjectivity

tweets_df[['polarity', 'subjectivity']] = tweets_df['cleaned_tweet'].apply(lambda x: pd.Series(get_sentiment(x)))

print("\nDataFrame with polarity and subjectivity:")
print(tweets_df[['cleaned_tweet', 'polarity', 'subjectivity']].head())
```

Output:

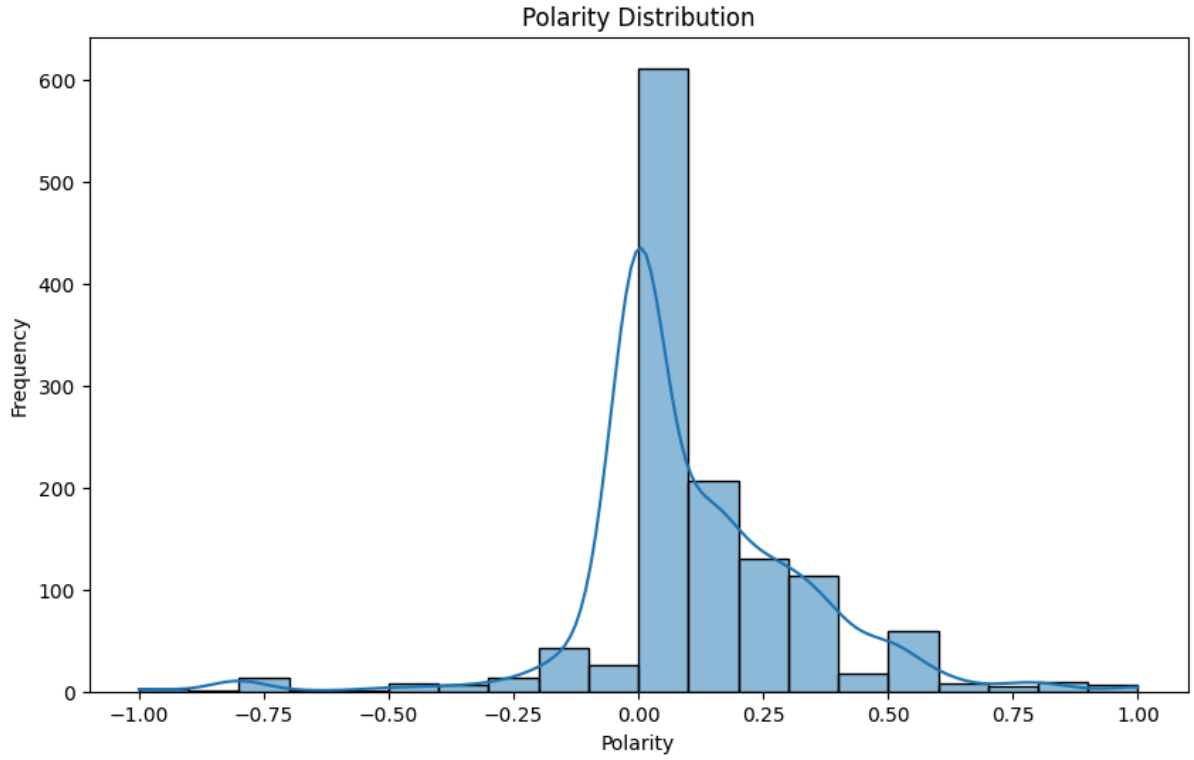
```
DataFrame with polarity and subjectivity:
                                       cleaned_tweet  polarity  subjectivity
0  pathetic rejecting order capable transporting ...   -0.1125        0.7375
1  pathetic rejecting order capable transporting ...   -0.1125        0.7375
2  pathetic rejecting order capable transporting ...   -0.1125        0.7375
3  pathetic rejecting order capable transporting ...   -0.1125        0.7375
4  pathetic rejecting order capable transporting ...   -0.1125        0.7375
```

Interpretation:

The DataFrame shows that the cleaned tweets have a slightly negative sentiment with a polarity score of -0.1125, and they are highly subjective, with a subjectivity score of 0.7375, indicating that the text is more opinion-based rather than factual.

```
plt.figure(figsize=(10, 6))
sns.histplot(tweets_df['polarity'], bins=20, kde=True)
plt.title('Polarity Distribution')
plt.xlabel('Polarity')
plt.ylabel('Frequency')
plt.show()
```

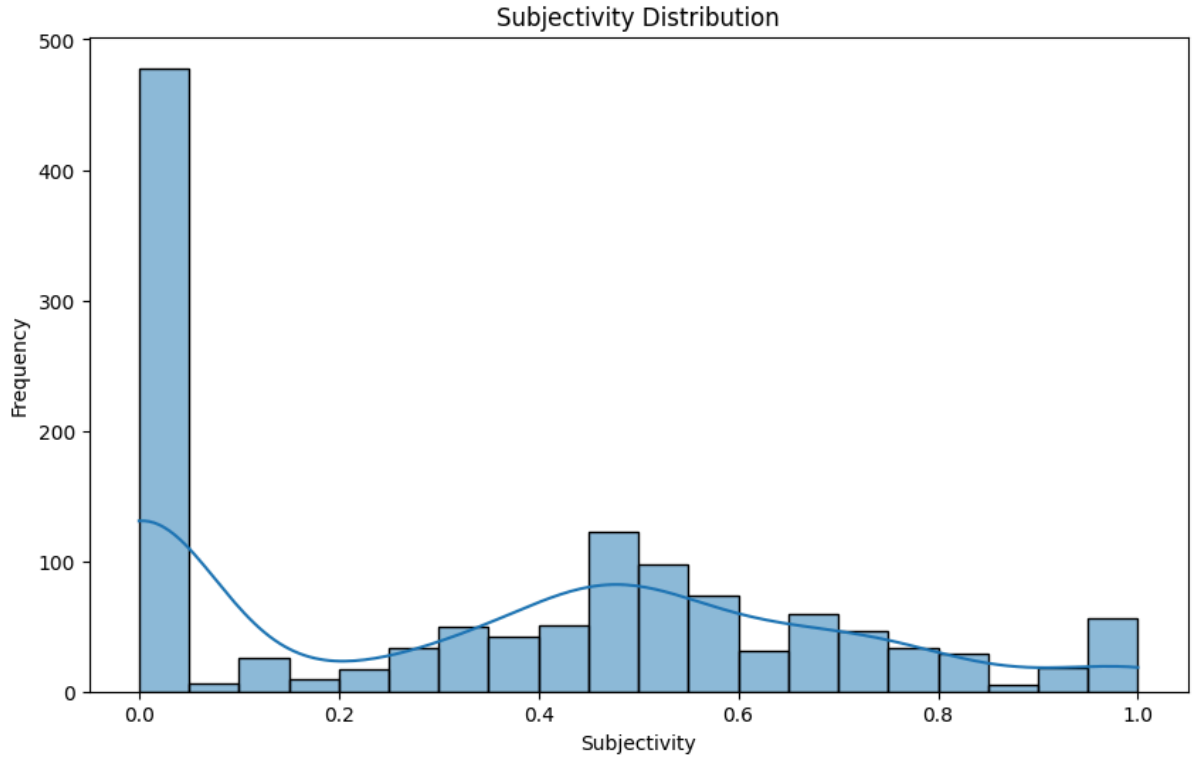
Now we have plotted a Histogram for Polarity



Most of the polarity values are concentrated around 0, indicating that the majority of the text exhibits a neutral sentiment. There is a noticeable skew toward the positive side, with a gradual decline as the polarity increases, indicating fewer strongly positive texts. Additionally, there is a small but significant presence of negative sentiment values, though much less frequent. Overall, the tweets contains predominantly neutral to mildly positive content.

```
plt.figure(figsize=(10, 6))
sns.histplot(tweets_df['subjectivity'], bins=20, kde=True)
plt.title('Subjectivity Distribution')
plt.xlabel('Subjectivity')
plt.ylabel('Frequency')
plt.show()
```

Now we have plotted a Histogram for Subjectivity



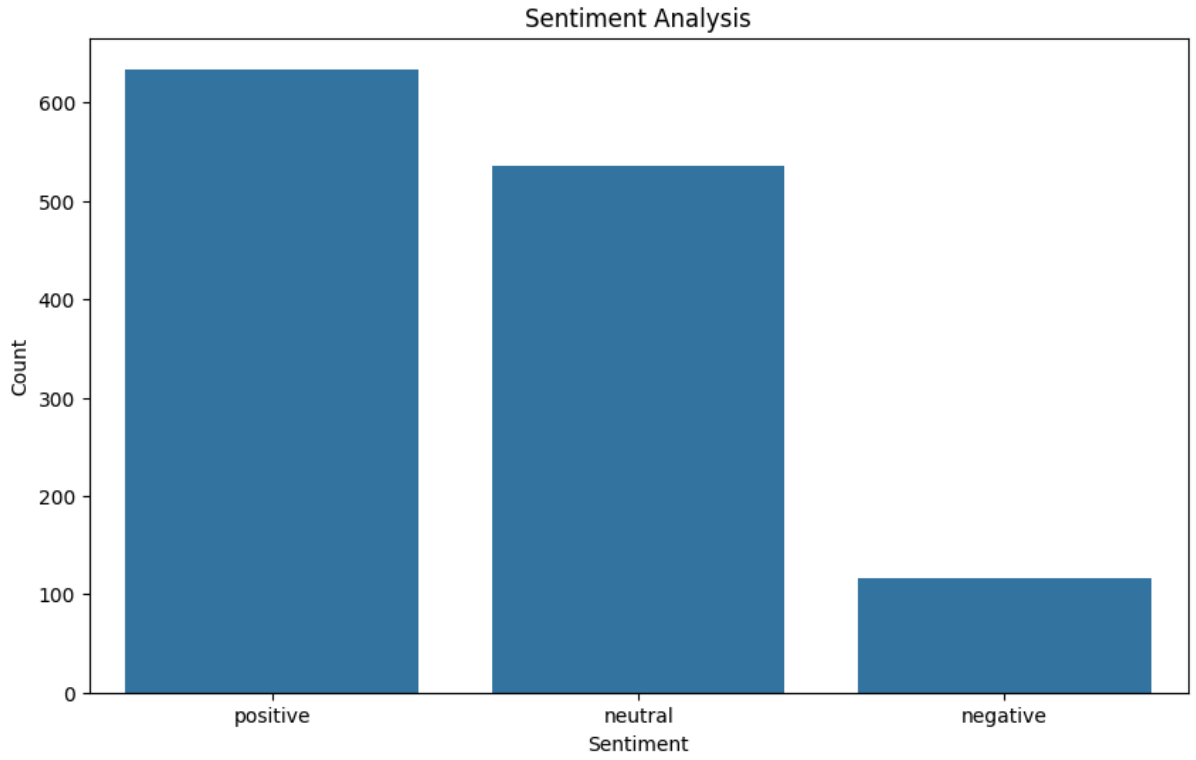
A large proportion of the data is concentrated at a subjectivity score of 0, indicating that many texts are highly objective. However, the distribution shows a gradual increase toward moderate subjectivity levels (around 0.4 to 0.6), with a small but noticeable portion of the data reaching high subjectivity scores closer to 1, meaning they express strong personal opinions or emotions. Overall, the tweets is dominated by objective content, with fewer texts being highly subjective.

```
tweets_df['sentiment'] = tweets_df['polarity'].apply(lambda x: 'positive' if x > 0 else ('negative' if x < 0 else 'neutral'))


sentiment_counts = tweets_df['sentiment'].value_counts()
```

The code assigns a sentiment label ('positive', 'negative', or 'neutral') to each tweet based on the polarity value. It then calculates the count of each sentiment type using the value_counts() function.

```
plt.figure(figsize=(10, 6))
sns.barplot(x=sentiment_counts.index, y=sentiment_counts.values)
plt.title('Sentiment Analysis')
plt.xlabel('Sentiment')
plt.ylabel('Count')
plt.show()
```

We have now plot the count values in a Bar Graph



The bar chart shows the distribution of sentiment analysis across three categories: positive, neutral, and negative. Most of the tweets have a positive sentiment, followed by a significant number of neutral tweets. Negative tweets are the least frequent, indicating that the overall tone of the dataset is largely positive or neutral.

Sentiment Analysis Using Hugging Face

Hugging Face is a machine learning (ML) and data science platform and community that helps users build, deploy and train machine learning models. It provides the infrastructure to demo, run and deploy artificial intelligence (AI) in live applications. Users can also browse through models and data sets that other people have uploaded.

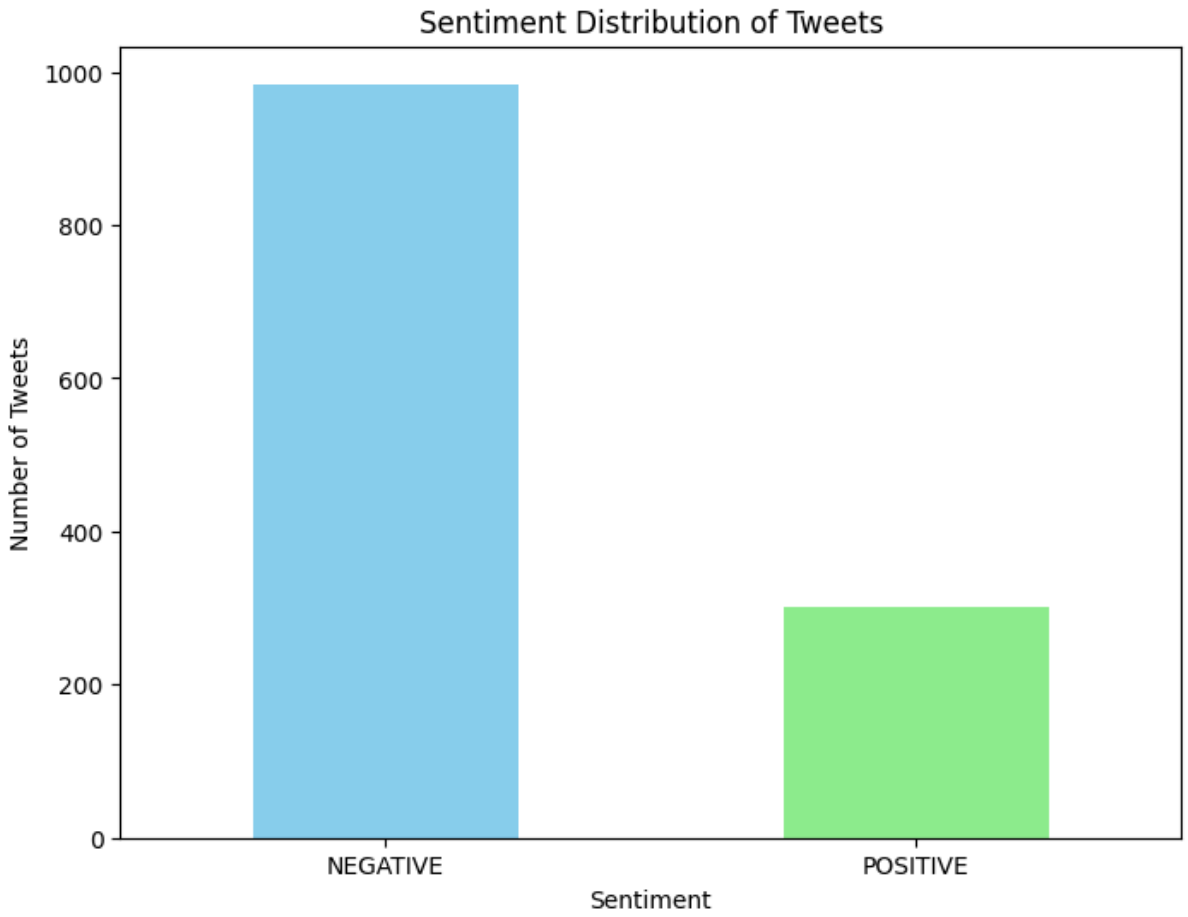
Hugging Face is known for its Transformers Python library, which simplifies the process of downloading and training ML models. The library gives developers an efficient way to include one of the ML models hosted on Hugging Face in their workflow and create ML pipelines.

```
from transformers import pipeline
import pandas as pd

df = pd.read_csv('/content/tweets3.csv')


sentiment_analyzer = pipeline('sentiment-analysis', model="distilbert-base-uncased-finetuned-sst-2-english")


df['Sentiment'] = df['Text'].apply(lambda tweet: sentiment_analyzer(tweet)[0]['label'])


print(df[[ 'Username', 'Text', 'Sentiment']])
```

Here we have applied a sentiment analysis model from Hugging Face (`distilbert-base-uncased-finetuned-sst-2-english`) to each tweet. It creates a sentiment analysis pipeline, which labels each tweet as either "positive" or "negative." These sentiment labels are added to a new 'Sentiment' column in the DataFrame. Finally, the code prints out a table showing the 'Username', 'Text', and 'Sentiment' for each tweet.

```
sentiment_counts = df['Sentiment'].value_counts()

# Plotting a bar chart for sentiment distribution
plt.figure(figsize=(8,6))
sentiment_counts.plot(kind='bar', color=['skyblue', 'lightgreen', 'salmon'])
plt.title('Sentiment Distribution of Tweets')
plt.xlabel('Sentiment')
plt.ylabel('Number of Tweets')
plt.xticks(rotation=0)
plt.show()
```

Now we plot the count of sentiments in a Bar plot



The bar chart shows the sentiment distribution of tweets, with the majority being labeled as "negative" and a much smaller portion labeled as "positive." The number of negative tweets is significantly higher, nearing 1000, while positive tweets are fewer, around 300, indicating that most of the analyzed tweets have a negative sentiment.

Conclusion

After comparing our initial bar plot with the new one generated using the Hugging Face model, we can see a clear discrepancy in the sentiment analysis results. Our initial model, trained on a limited dataset of 1200 rows, showed a more balanced distribution between positive and neutral sentiments. However, the Hugging Face BERT model, trained on a much larger dataset, indicates a significant skew towards negative sentiments. This suggests that our model might be inaccurate, likely due to insufficient training data or incorrect parameter tuning. The extensive training of the BERT model on millions of rows likely makes it more reliable and accurate, highlighting the limitations of our smaller dataset and simpler model.

The final bar plot, which analyzes tweets about the new Google Pixel 9 using the Hugging Face BERT model, shows that the majority of the tweets have a "negative" sentiment, with only a small proportion being "positive." This indicates that the overall public reaction to the Google Pixel 9 is largely negative, with users expressing more dissatisfaction, criticism, or disappointment. The smaller number of positive sentiments suggests that while there are some favourable opinions, they are far outweighed by the negative responses in the dataset. This sentiment analysis reveals a generally unfavourable reception of the Google Pixel 9 on social media.