PHASE 2
PROJECT- 3

Data Analyst Project:

Twitter Sentiment Analysis

NAME : AARYA KASHYAP
EMAIL : aarya07kashyap@gmail.com
PHONE NO. : 7042940270

PROJECT OBJECTIVE :

- The main objective of 'Twitter Sentiment Analysis' is to analyze a dataset of tweets to determine the sentiment expressed in each tweet—whether it is positive, negative, or neutral.
- The project also aims to gain insights into public opinions, trends, and sentiments shared on Twitter, utilizing data analytics techniques.

I have created a complete documentation which covers the overall data preprocessing steps, the issues that I encountered during that, themodel implementation, and also the analysis findings.

---

1. DATA EXPLORATION:

i. During the data loading phase, I encountered an issue with the default encoding used by pandas when reading the CSV file, which I never encountered during the first 2 projects . Initially, I attempted to load the data using the **'unicode_escape' encoding**, but this resulted in a **unicodeescape codec error** due to unescaped backslashes in the file. To resolve this, I experimented with several alternative encodings and found that the **'ISO-8859-1 (Latin-1)'** encoding successfully read the file without errors.

ii. Our dataset had no column names and was identifying the first record as the header of each column so we needed to specify each column names in our dataset for a better understanding.

```
# Basic information about the dataframe df: rows , columns , types
dataframe.info()
✓ 0.2s

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599999 entries, 0 to 1599998
Data columns (total 6 columns):
 #   Column                                                                                                    Non-Null Count    Dtype
---  ------                                                                                                    --------------    -----
 0   0                                                                                                         1599999 non-null  int64
 1   1467810369                                                                                                1599999 non-null  int64
 2   Mon Apr 06 22:19:45 PDT 2009                                                                              1599999 non-null  object
 3   NO_QUERY                                                                                                  1599999 non-null  object
 4   _TheSpecialOne_                                                                                           1599999 non-null  object
 5   @switchfoot http://twitpic.com/2y1zl - Awww, that's a bummer.  You shoulda got David Carr of Third Day to do it. ;D  1599999 non-null  object
dtypes: int64(2), object(4)
memory usage: 73.2+ MB
```

```
# Defining the column names
column_names = ['Target', 'ID', 'Date', 'Flag', 'Username', 'Tweet']

# Reading the CSV file into a DataFrame with specified column names
df = pd.read_csv( r"C:\Users\Aarya\Desktop\NEXUS\TwitterSentiment.csv", names=column_names )
✓ 3.0s

df.info()
✓ 0.1s

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1600000 entries, 0 to 1599999
Data columns (total 6 columns):
 #   Column    Non-Null Count    Dtype
---  ------    --------------    -----
 0   Target    1600000 non-null  int64
 1   ID        1600000 non-null  int64
 2   Date      1600000 non-null  object
 3   Flag      1600000 non-null  object
 4   Username  1600000 non-null  object
 5   Tweet     1600000 non-null  object
dtypes: int64(2), object(4)
memory usage: 73.2+ MB
```

```
# Finding the number of unique IDs

print("Number of total IDs:", df['ID'].count())
print("Number of unique IDs:", df['ID'].nunique())
[15] ✓ 0.1s

... Number of total IDs: 1600000
    Number of unique IDs: 1598315
```

KEY INSIGHTS :
i. Our datset contains a total of 1600000 records which is a very huge amount.
ii. The data belongs to the month of 'April' , 'May' and 'June' in the year 2009.

2. DATA CLEANING :

```
# checking for any null values in the dataframe
df.isnull().sum()
[10]  ✓ 0.1s
...  Target      0
     ID          0
     Date        0
     Flag        0
     Username    0
     Tweet       0
     dtype: int64
```

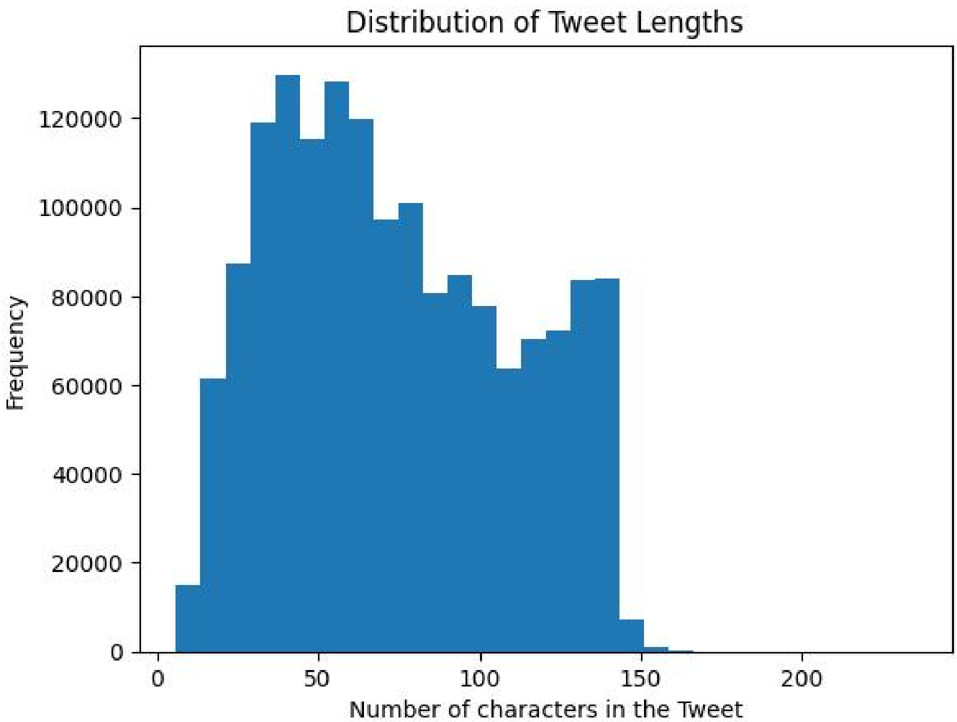
```
# Finding duplicate entries
duplicate_entries = df.duplicated()
print("Duplicate entries:")
print(df[duplicate_entries])
✓ 1.4s
Duplicate entries:
Empty DataFrame
Columns: [Target, ID, Date, Flag, Username, Tweet]
Index: []
```

CONCLUSION : Our data set contained neitherl values or any sort of duplicate values , hence our data set was clean.

---

3. EXPLORATORY DATA ANALYSIS (EDA)

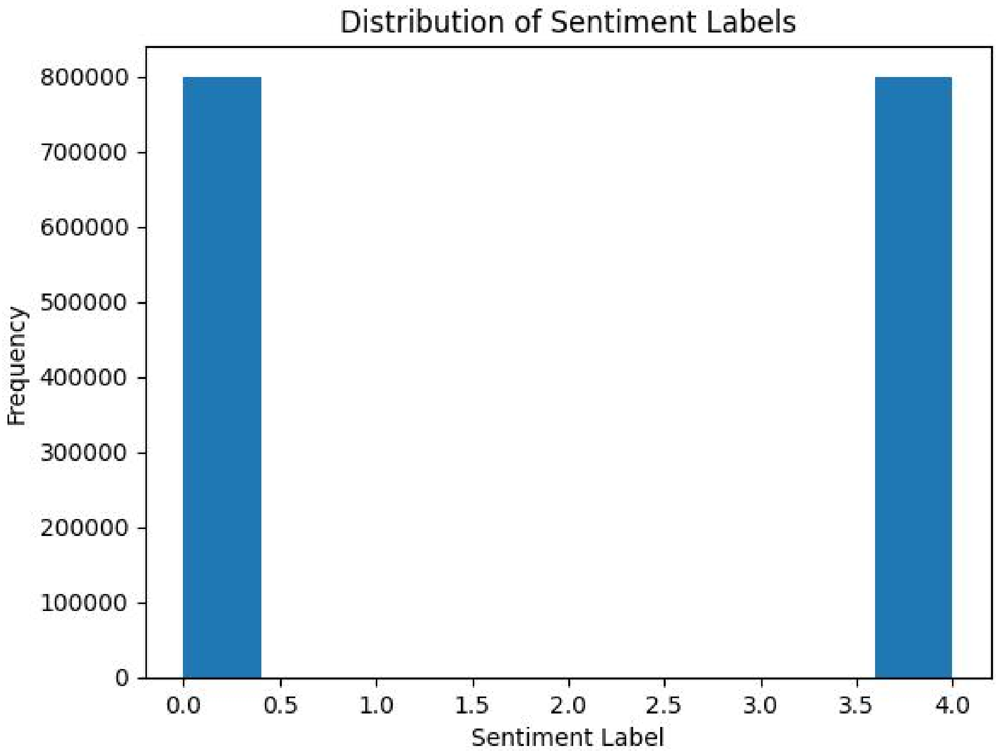
```
# Ploting histogram of sentiment labels
plt.hist(df['Target'])
plt.xlabel('Sentiment Label')
plt.ylabel('Frequency')
plt.title('Distribution of Sentiment Labels')
plt.show()
✓ 0.1s
```



KEY INSIGHTS :
1. Social media platforms often impose character limits on tweets, such as Twitter's 280-character limit. As a result, most tweets tend to fall within a relatively narrow range of lengths, typically shorter than the maximum allowed characters.
2. Users may tend to write tweets of similar lengths due to various factors such as attention span, readability, or the nature of the content being shared.

---

4. SENTIMENT DISTRIBUTION

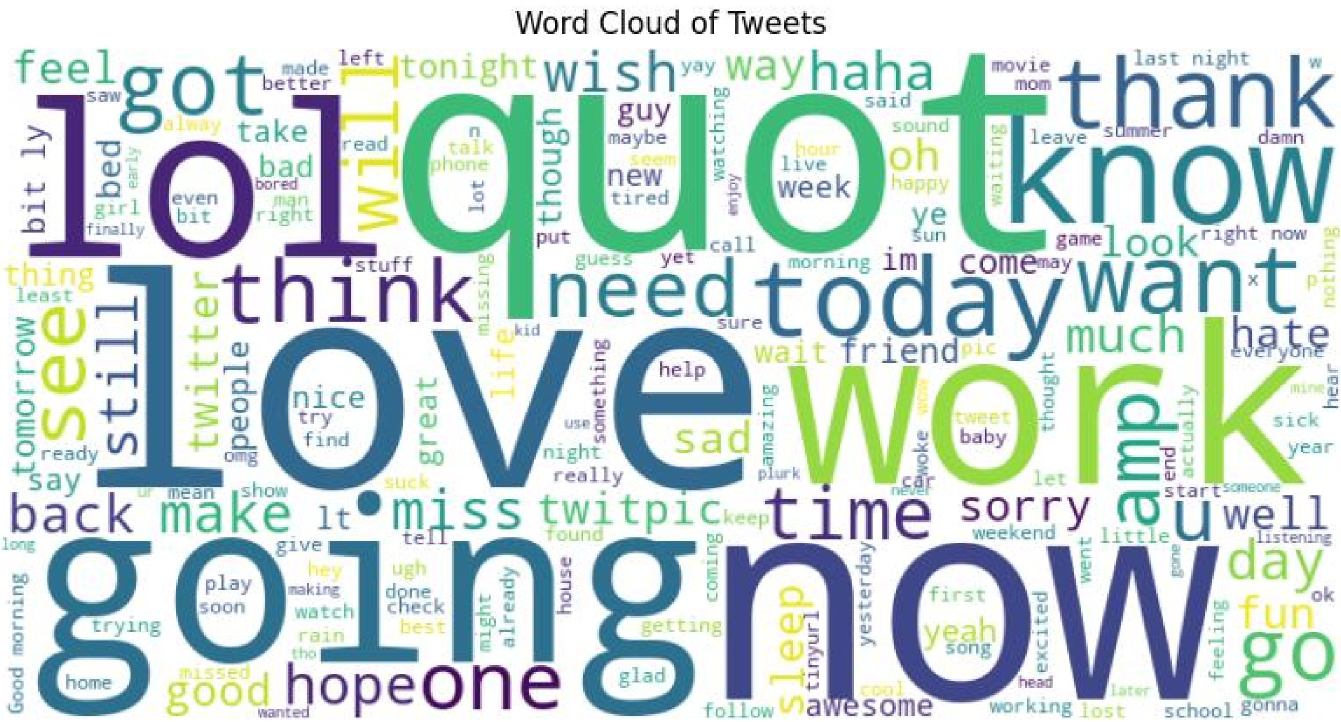


CONCLUSION:

1. It suggests that the dataset is evenly balanced between the two sentiment classes.
2. Balanced datasets can be beneficial for training machine learning models, as they prevent biases towards one class over the other.
3. However, it's essential to be cautious of potential biases or limitations that may arise from a balanced dataset. For example, the balanced distribution may not reflect real-world sentiment distribution.

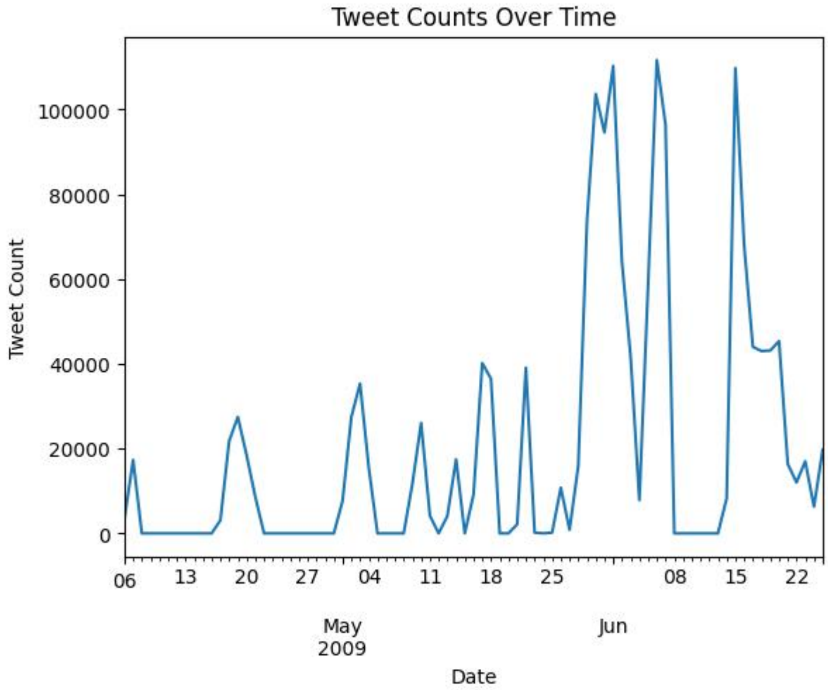
5. WORD FREQUENCY ANALYSIS

```python
from wordcloud import WordCloud
wordcloud = WordCloud(width=800, height=400, background_color='white').generate(' '.join(df['Tweet']))
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('Word Cloud of Tweets')
plt.show()
```
1m 7.7s

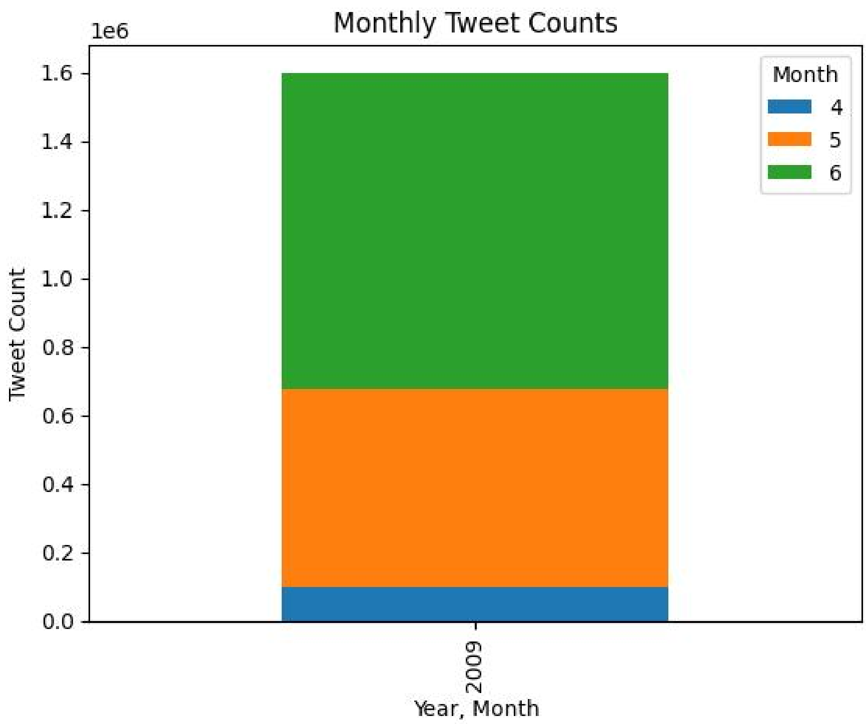


CONCLUSION:
The larger and more prominent words in the word cloud represent the most frequently occurring words in the tweets. These words are likely to be common topics or themes discussed by users on the platform.

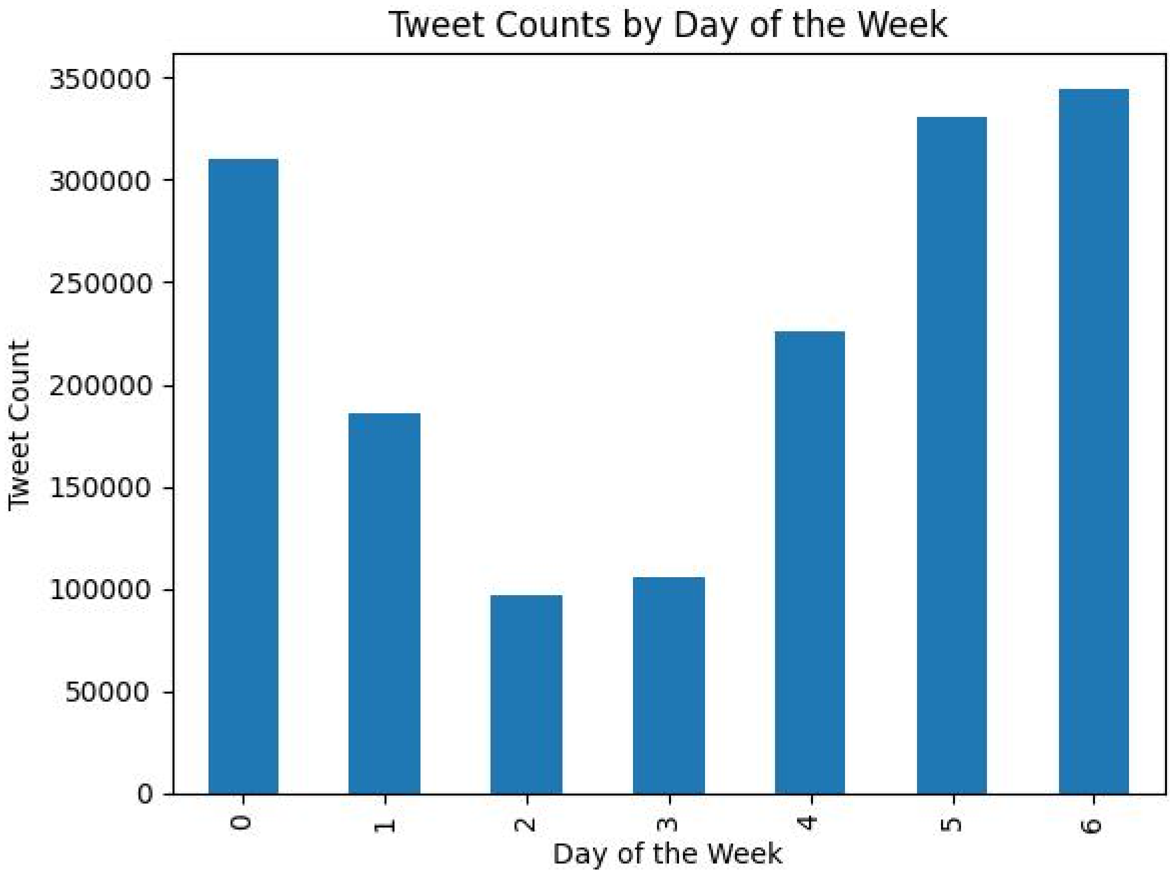
6. TEMPORAL ANALYSIS



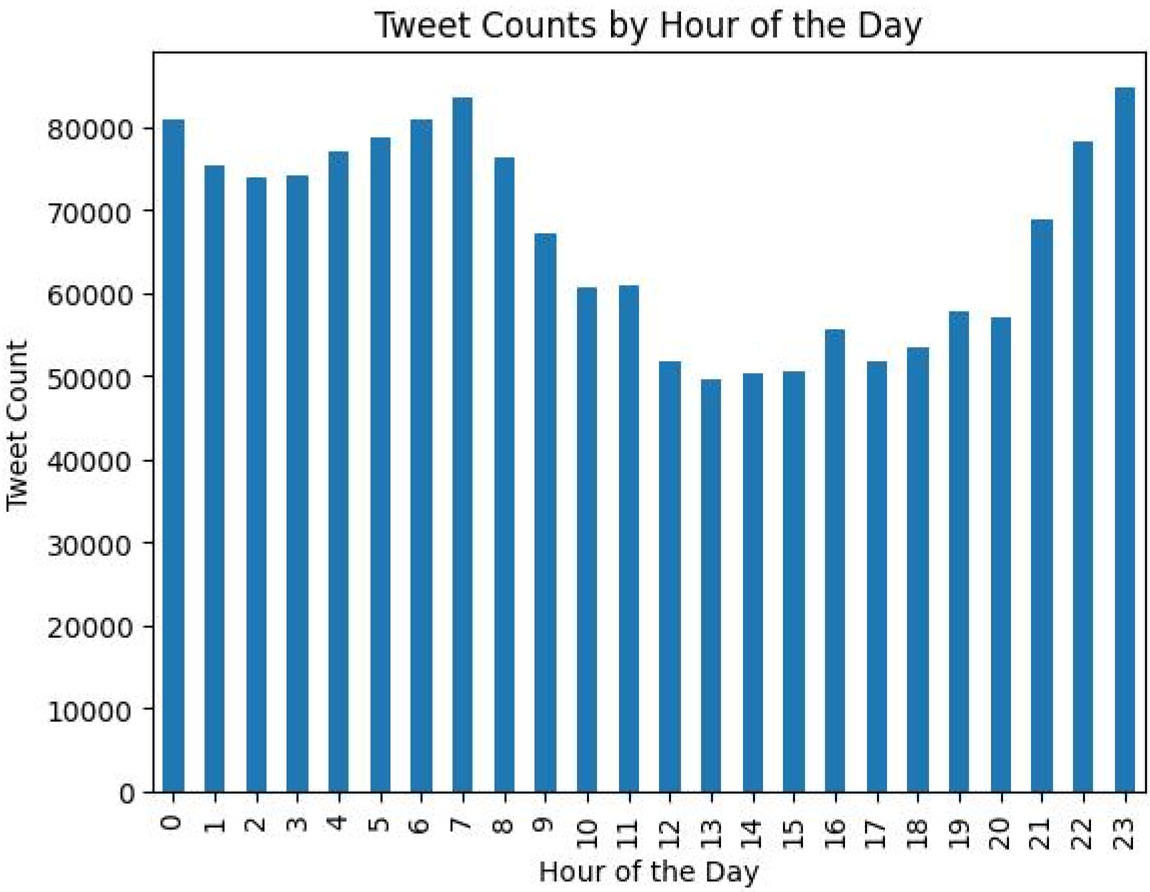
The tweet counts or we can say the user engagement peaked in the:
i. last week of May
ii. First week and third week of June



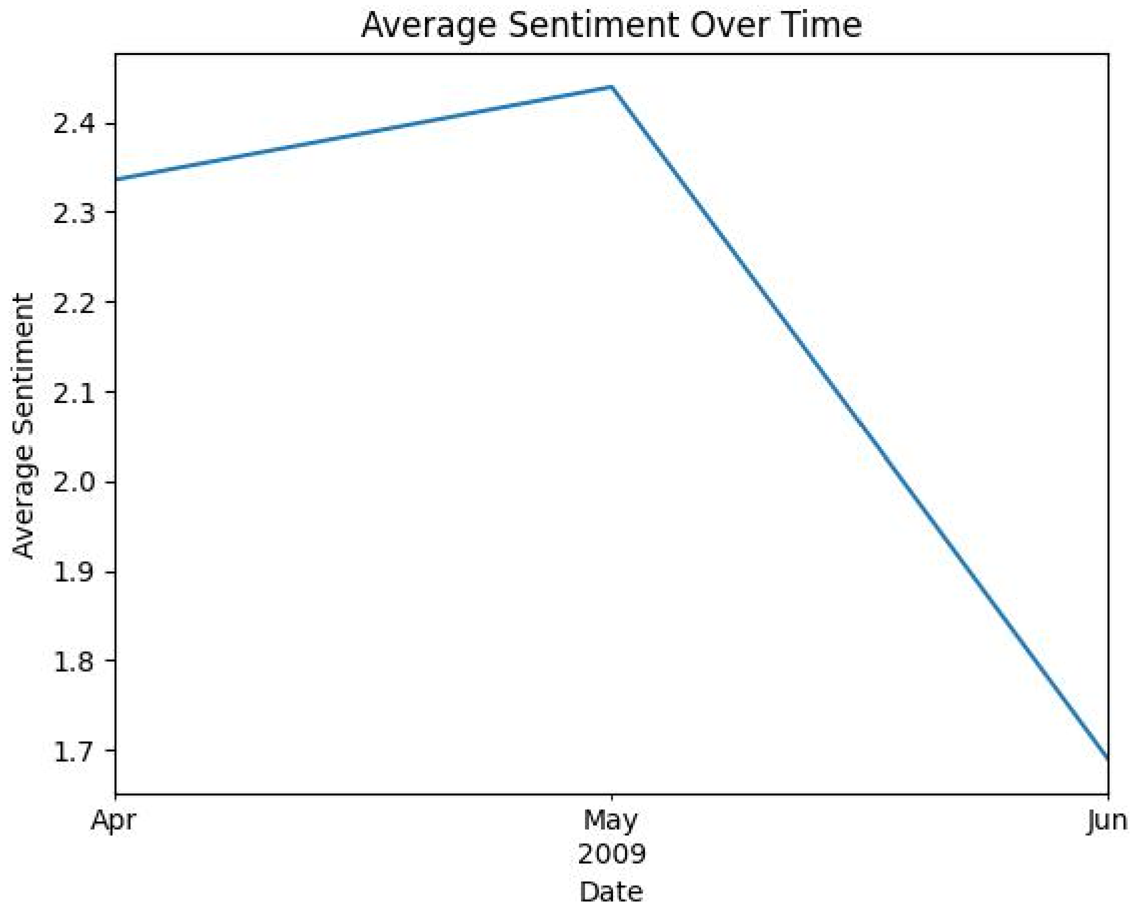
This again shows that the number of tweets peaked in the 6th month i.e. in the month of 'June'.

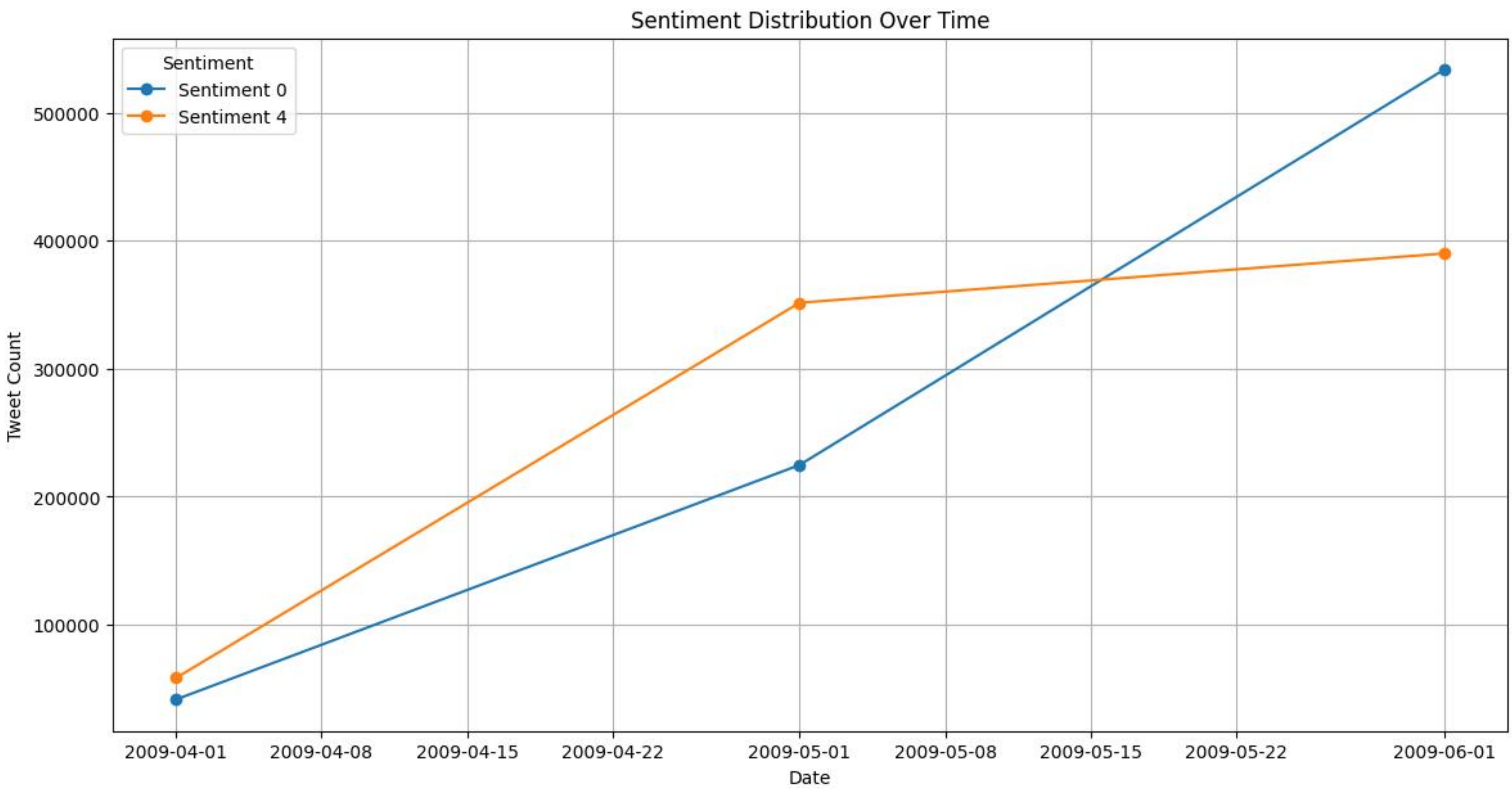


The higher user engagement to the Twitter and therefore higher number of tweets were ussually seen during the weekends and least during the weekdays and that too during the midweek.



The higher user engagement to the Twitter is usually seen during late night and early morning and it is least during the afternoon


Average Sentiment Over Time
Average Sentiment
2.4
2.3
2.2
2.1
2.0
1.9
1.8
1.7
Apr
May
2009
Jun
Date


Sentiment Distribution Over Time
Sentiment
Sentiment 0
Sentiment 4
Tweet Count
500000
400000
300000
200000
100000
2009-04-01
2009-04-08
2009-04-15
2009-04-22
2009-05-01
2009-05-08
2009-05-15
2009-05-22
2009-06-01
Date

7. TEXT PREPROCESSING

- Removing stop words helps in focusing on the more meaningful words that are likely to contribute more to the understanding or analysis of the text.
- Removing special characters helps in standardizing the text, making it easier to analyze and process.
- Removing these characters can improve the accuracy of tokenization by preventing words from being split incorrectly.

i. Removing stop words, special characters, and URLs from the 'Tweet' column.

```
import re
import nltk
from nltk.corpus import stopwords

nltk.download('stopwords')

# Defining a function to clean tweets

def clean_tweet(tweet):
    # Remove URLs
    tweet = re.sub(r'http\S+|www\S+|https\S+', '', tweet, flags=re.MULTILINE)

    # Remove special characters
    tweet = re.sub(r'\@\w+|\#','', tweet)
    tweet = re.sub(r'[^A-Za-z0-9\s]+', '', tweet)

    # Remove stop words
    stop_words = set(stopwords.words('english'))
    tweet = ' '.join([word for word in tweet.split() if word.lower() not in stop_words])
    return tweet


df['Cleaned_Tweet'] = df['Tweet'].apply(clean_tweet)
df.head()
```

	Target	ID	Date	Username	Tweet	Number of characters in the Tweet	Year	Month	Day	Hour	DayOfWeek	Cleaned_Tweet
0	0	1467810369	2009-04-06 22:19:45	_TheSpecialOne_	@switchfoot http://twitpic.com/2y1zl - Awww, t...	115	2009	4	6	22	0	thats bummer shoulda got David Carr Third Day
1	0	1467810672	2009-04-06 22:19:49	scotthamilton	is upset that he can't update his Facebook by ...	111	2009	4	6	22	0	upset cant update Facebook texting might cry r...
2	0	1467810917	2009-04-06 22:19:53	mattycus	@Kenichan I dived many times for the ball. Man...	89	2009	4	6	22	0	dived many times ball Managed save 50 rest go ...
3	0	1467811184	2009-04-06 22:19:57	ElleCTF	my whole body feels itchy and like its on fire	47	2009	4	6	22	0	whole body feels itchy like fire
4	0	1467811193	2009-04-06 22:19:57	Karoli	@nationwideclass no, it's not behaving at all....	111	2009	4	6	22	0	behaving im mad cant see

ii. Tokenization & Lemmatization

Tokenization is the process of breaking down a text into smaller units called tokens. These tokens could be words, phrases, symbols, or other meaningful elements.
The goal of lemmatization is to group together different inflected forms of a word so they can be analyzed as a single item.

```python
import pandas as pd
import nltk
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer

# Ensure the necessary NLTK data packages are downloaded
nltk.download('punkt')
nltk.download('wordnet')


# Function to tokenize and lemmatize text
def tokenize_and_lemmatize(text):
    # Tokenize the text into words
    tokens = word_tokenize(text)

    # Initialize WordNet lemmatizer
    lemmatizer = WordNetLemmatizer()

    # Lemmatize each tokenized word
    lemmatized_tokens = [lemmatizer.lemmatize(token) for token in tokens]

    # Return the lemmatized tokens as a string
    return ' '.join(lemmatized_tokens)

# Apply tokenization and lemmatization to the 'Tweet' column
df['Processed_Tweet'] = df['Tweet'].apply(tokenize_and_lemmatize)
```

```python
df.head()
```

✓ 0.0s Python

	Target	ID	Date	Username	Tweet	Number of characters in the Tweet	Year	Month	Day	Hour	DayOfWeek	Cleaned_Tweet	Processed_Tweet
0	0	1467810369	2009-04-06 22:19:45	_TheSpecialOne_	@switchfoot http://twitpic.com/2y1zl - Awww, t...	115	2009	4	6	22	0	thats bummer shoulda got David Carr Third Day	@ switchfoot http : //twitpic.com/2y1zl - Awww...
1	0	1467810672	2009-04-06 22:19:49	scotthamilton	is upset that he can't update his Facebook by ...	111	2009	4	6	22	0	upset cant update Facebook texting might cry r...	is upset that he ca n't update his Facebook by...
2	0	1467810917	2009-04-06 22:19:53	mattycus	@Kenichan I dived many times for the ball. Man...	89	2009	4	6	22	0	dived many times ball Managed save 50 rest go ...	@ Kenichan I dived many time for the ball . Ma...
3	0	1467811184	2009-04-06 22:19:57	ElleCTF	my whole body feels itchy and like its on fire	47	2009	4	6	22	0	whole body feels itchy like fire	my whole body feel itchy and like it on fire
4	0	1467811193	2009-04-06 22:19:57	Karoli	@nationwideclass no, it's not behaving at all....	111	2009	4	6	22	0	behaving im mad cant see	@ nationwideclass no , it 's not behaving at a...

8. SENTIMENT PREDICTION MODEL

i. BAG OF WORDS:

Initially when I tried to execute this code I came to realise that the code was memory inefficient as the dataset contained very alrge number of data

```
# Initialize the CountVectorizer
vectorizer = CountVectorizer()

# Fit and transform the cleaned tweets to BoW
bow_matrix = vectorizer.fit_transform(df['Cleaned_Tweet'])

# Convert the matrix to a DataFrame for better readability
bow_df = pd.DataFrame(bow_matrix.toarray(), columns=vectorizer.get_feature_names_out())

# Combine the BoW DataFrame with the original DataFrame
df_bow = pd.concat([df, bow_df], axis=1)

# Display the resulting DataFrame
print(df_bow.head())

[ ]
```

```
---------------------------------------------------------------------------
MemoryError                               Traceback (most recent call last)
Cell In[76], line 8
      5 bow_matrix = vectorizer.fit_transform(df['Cleaned_Tweet'])
      7 # Convert the matrix to a DataFrame for better readability
----> 8 bow_df = pd.DataFrame(bow_matrix.toarray(), columns=vectorizer.get_feature_names_out())
     10 # Combine the BoW DataFrame with the original DataFrame
     11 df_bow = pd.concat([df, bow_df], axis=1)

File c:\Users\Aarya\AppData\Local\Programs\Python\Python312\Lib\site-packages\scipy\sparse\_compressed.py:1106, in _cs_matrix.toarray(self, orde
   1104 if out is None and order is None:
   1105     order = self._swap('cf')[0]
-> 1106 out = self._process_toarray_args(order, out)
   1107 if not (out.flags.c_contiguous or out.flags.f_contiguous):
   1108     raise ValueError('Output array must be C or F contiguous')

File c:\Users\Aarya\AppData\Local\Programs\Python\Python312\Lib\site-packages\scipy\sparse\_base.py:1327, in _spbase._process_toarray_args(self,
   1325     return out
   1326 else:
-> 1327     return np.zeros(self.shape, dtype=self.dtype, order=order)

MemoryError: Unable to allocate 5.05 TiB for an array with shape (1600000, 434045) and data type int64
```

I then Applied 'CountVectorizer' to Transform Tweets into BoW (bag of words) . Instead of converting the sparse matrix to a dense DataFrame I used 'Sparse Representation' which was more memory efficient.

```
import nltk
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import CountVectorizer

# Ensure the necessary NLTK data packages are downloaded
nltk.download('punkt')
nltk.download('wordnet')
# Initializing the CountVectorizer
vectorizer = CountVectorizer()

# Fitting and transforming the cleaned tweets to BoW (keeping it in sparse format)
bow_matrix = vectorizer.fit_transform(df['Cleaned_Tweet'])

[34]  ✓  14.1s

...  [nltk_data] Downloading package punkt to
     [nltk_data]     C:\Users\Aarya\AppData\Roaming\nltk_data...
     [nltk_data]   Package punkt is already up-to-date!
     [nltk_data] Downloading package wordnet to
     [nltk_data]     C:\Users\Aarya\AppData\Roaming\nltk_data...
     [nltk_data]   Package wordnet is already up-to-date!
```

ii. **TF-IDF (Term Frequency-Inverse Document Frequency) : Adjusting the weight of words based on their frequency in the document relative to their frequency in the corpus.**

```python
import pandas as pd
import spacy

# Load the spaCy model
nlp = spacy.load("en_core_web_md")



# Function to preprocess text using spaCy (tokenize, lemmatize, and remove stop words)
def preprocess_text(text):
    doc = nlp(text)
    tokens = [token.lemma_ for token in doc if not token.is_stop and not token.is_punct]
    return ' '.join(tokens)

# Apply preprocessing to the 'Tweet' column
df['Processed_Tweet'] = df['Tweet'].apply(preprocess_text)

# Function to get word embeddings for a given text
def get_word_embeddings(text):
    doc = nlp(text)
    return doc.vector

# Apply word embeddings to the 'Processed_Tweet' column
df['Embeddings'] = df['Processed_Tweet'].apply(get_word_embeddings)

# Display the original, processed text, and embeddings
print("Original Text:")
print(df['Tweet'])
print("\nProcessed Text:")
print(df['Processed_Tweet'])
print("\nWord Embeddings:")
print(df['Embeddings'])
```

iii. **Feature Extraction Using TF-IDF. Converting the cleaned text data into numerical features using TF-IDF.**

```
from sklearn.feature_extraction.text import TfidfVectorizer
# Split the data
X_train, X_test, y_train, y_test = train_test_split(df['Cleaned_Tweet'], df['Target'], test_size=0.2, random_state=42)
# Initialize the TfidfVectorizer
tfidf_vectorizer = TfidfVectorizer()

# Fit and transform the training data
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)

# Transform the test data
X_test_tfidf = tfidf_vectorizer.transform(X_test)
```

✓ 11.9s

iv. **Model Training : training a logistic regression model using the TF-IDF features.**

```
from sklearn.linear_model import LogisticRegression

# Initialize the logistic regression model
model = LogisticRegression(max_iter=200)

# Train the model
model.fit(X_train_tfidf, y_train)
```

✓ 25.7s

```
LogisticRegression
LogisticRegression(max_iter=200)
```

v. **Model Evaluation: Evaluating the model's performance on the test data.**

```
from sklearn.metrics import classification_report

# Make predictions
y_pred = model.predict(X_test_tfidf)

# Evaluate the model
print(classification_report(y_test, y_pred))
```

✓ 0.9s

```
              precision    recall  f1-score   support

           0       0.79      0.77      0.78    159494
           4       0.78      0.80      0.79    160506

    accuracy                           0.78    320000
   macro avg       0.78      0.78      0.78    320000
weighted avg       0.78      0.78      0.78    320000
```

There is always room for improvement.
Hyperparameter Tuning: Wel used the technique like Grid Search to find the best hyperparameters for our model.

```python
from sklearn.model_selection import GridSearchCV

# Define the parameter grid
param_grid = {
    'C': [0.1, 1, 10, 100],  # Regularization strength
    'solver': ['liblinear', 'saga']  # Solver
}

# Initialize the logistic regression model
log_reg = LogisticRegression(max_iter=200)

# Initialize GridSearchCV
grid_search = GridSearchCV(log_reg, param_grid, cv=3, scoring='f1_weighted', verbose=2, n_jobs=-1)

# Fit GridSearchCV
grid_search.fit(X_train_tfidf, y_train)

# Get the best estimator
best_log_reg = grid_search.best_estimator_

# Make predictions with the best model
y_pred_best = best_log_reg.predict(X_test_tfidf)

# Evaluate the best model
print(classification_report(y_test, y_pred_best))
```

✓ 8m 4.3s

```
... Fitting 3 folds for each of 8 candidates, totalling 24 fits
              precision    recall  f1-score   support

           0       0.79      0.77      0.78    159494
           4       0.78      0.80      0.79    160506

    accuracy                           0.78    320000
   macro avg       0.79      0.78      0.78    320000
weighted avg       0.79      0.78      0.78    320000
```

CONCLUSION:

The classification report shows a solid performance with balanced precision, recall, and F1-scores for both classes.

9. FEATURE IMPORTANCE

```python
import pandas as pd
import re
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
import matplotlib.pyplot as plt
import seaborn as sns
from wordcloud import WordCloud

# Download necessary NLTK data
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')


# Get feature names
feature_names = tfidf_vectorizer.get_feature_names_out()

# Get the coefficients from the logistic regression model
coefficients = model.coef_[0]

# Create a DataFrame with feature names and their corresponding coefficients
feature_importance = pd.DataFrame({
    'feature': feature_names,
    'coefficient': coefficients
})

# Sort the DataFrame by the absolute value of the coefficients
feature_importance['abs_coefficient'] = feature_importance['coefficient'].abs()
feature_importance = feature_importance.sort_values(by='abs_coefficient', ascending=False)

# Display the top 10 most important features
print(feature_importance.head(10))

# Get the top 20 most important features
top_features = feature_importance.head(20)

# Create a bar plot
plt.figure(figsize=(10, 8))
sns.barplot(x='abs_coefficient', y='feature', data=top_features, palette='viridis')
plt.title('Top 20 Important Features for Sentiment Prediction')
plt.xlabel('Absolute Coefficient Value')
plt.ylabel('Feature')
plt.show()

# Generate a word cloud
wordcloud = WordCloud(width=800, height=400, background_color='white').generate_from_frequencies(
    {word: coeff for word, coeff in zip(feature_importance['feature'], feature_importance['coefficient'])}
)

# Display the word cloud
plt.figure(figsize=(10, 8))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('Word Cloud of Important Features for Sentiment Prediction')
plt.show()
```

✓ 6.1s

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\Aarya\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\Aarya\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\Aarya\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

	feature	coefficient	abs_coefficient
281446	sad	-11.526442	11.526442
206105	miss	-7.649917	7.649917
248568	poor	-7.198054	7.198054
154620	hurts	-6.874077	6.874077
344800	unfortunately	-6.788865	6.788865
281661	sadly	-6.732319	6.732319
294508	sick	-6.704086	6.704086
91584	disappointing	-6.596638	6.596638
137705	gutted	-6.559430	6.559430
206191	missing	-6.411375	6.411375

The aim is to identify the most important features (words or phrases) contributing to sentiment predictions and visualize feature importance using techniques such as bar charts or word clouds.