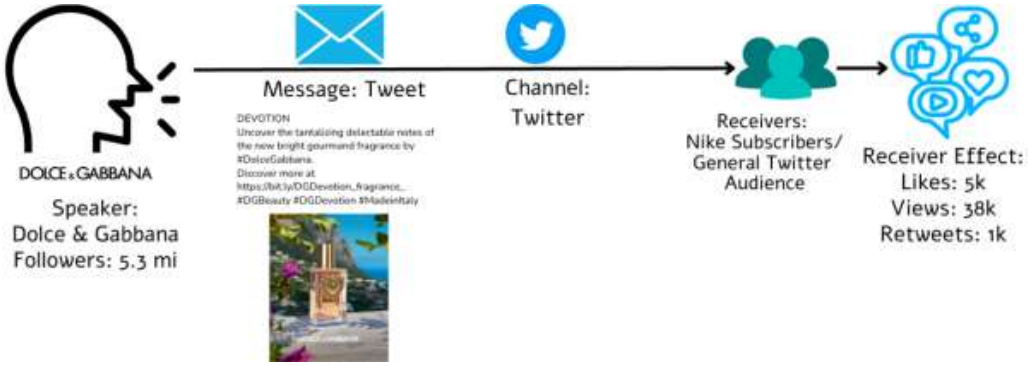
Data Preprocessing and Analysis Report

Rishi Roy

1 Exploratory Data Analysis

1.1 Introduction

The text data preprocessing and analysis pipeline described below consists of several stages, including data cleaning, exploratory data analysis (EDA), text preprocessing, vectorization, sentiment analysis, and visualization.



1.2 Reading the Dataset

The code reads a twitter dataset in the form of an an Excel file located at the specified path using the `pd.read_excel()` function from the pandas library. It then prints the first few rows of the dataset using `df.head()` to give a glimpse of the data.

1.3 EDA Summary

The initial stages of the pipeline involve reading the dataset from an Excel file, performing basic exploratory data analysis, and cleaning the text data. This includes steps such as removing stopwords, punctuation, URLs, mentions, and converting text to lowercase.

2 Text Preprocessing

2.1 Stopword Removal

We use the NLTK toolkit for preprocessing the content in the tweets. We utilize NLTK's stopwords corpus to obtain a set of English stopwords which can be used to remove simple words which may not bother our analysis of the dataset. Defines a function `data_processing()` to lowercase the text, remove URLs, mentions, punctuations, and stopwords using regular expressions and NLTK's `word_tokenize()` function. Applies the `data_processing()` function to the 'content' column of the dataframe (`df['content']`) using the `apply()` function. Removes duplicates from the dataframe based on the 'content' column using `drop_duplicates()`.

2.2 Custom Stopword List

We have defined a custom list `stopwordlist` containing additional stopwords and words to be excluded from analysis as a measure of precaution if some stopwords might still remain. We define a function `cleaning_stopwords()` to remove words present in `stopwordlist` from the text. Applied the `cleaning_stopwords()` function to the 'content' column using the `apply()` function.

2.3 Punctuation Removal

Defined a function `remove_punct()` to remove punctuations from the text using list comprehension. Applied the `remove_punct()` function to the 'content' column to create a new column 'Tweet_punct'.

2.4 Results Preview

Printed the first 10 rows of the dataframe using `df.head(10)` after applying text preprocessing steps to check if the preprocessing had occured in the correct manner.

3 Tokenization, Stopword Removal, Stemming, and Lemmatization

3.1 Tokenization

Defined a function `tokenization()` to tokenize the text using regular expression `re.split('\W+', text)`. Applies the `tokenization()` function to the 'Tweet_punct' column to tokenize the text into words. Stores the tokenized text in a new column 'Tweet_tokenized'.

3.2 Stopword Removal

Downloads the stopwords corpus from NLTK using `nltk.download('stopwords')`. Retrieves the list of English stopwords from the NLTK corpus. Defines a function `remove_stopwords()` to remove stopwords from the tokenized text. Applies the `remove_stopwords()` function to the 'Tweet_tokenized' column to remove stopwords. Stores the text without stopwords in a new column 'Tweet_nonstop'.

3.3 Stemming and Lemmatization

Initializes the PorterStemmer from NLTK as `ps` and the WordNetLemmatizer as `wn`. Defines functions `stemming()` and `lemmatizer()` to perform stemming and lemmatization on the text, respectively. Applies the `stemming()` function to the 'Tweet_nonstop' column to perform stemming. Applies the `lemmatizer()` function to the 'Tweet_nonstop' column to perform lemmatization. Stores the stemmed text in a new column 'Tweet_stemmed' and the lemmatized text in a new column 'Tweet_lemmatized'.

3.4 Cleaning Text

Defined a function `clean_text()` to clean the text by removing punctuation, numbers, and stopwords, and applying stemming. Split the text using `re.split('\W+', text_rc)` and removing punctuation. Removed numbers using `re.sub('[0-9]+', '', text_lc)`. Applied stemming and removed stopwords using the PorterStemmer and the previously defined stopword list. The function returns the cleaned text.

4 Vectorization

Initialized a CountVectorizer object `countVectorizer` with the `analyzer` parameter set to the custom `clean_text()` function defined earlier. This function performs text cleaning by removing punctuation, numbers, and stopwords, and applying stemming. Applies the `fit_transform()` method of the CountVectorizer to the 'content' column of the dataframe `df`, which contains the preprocessed text data. This step converts the text data into a matrix of token counts. Stores the resulting matrix of token counts in the variable `countVector`.

4.1 Results Output

Printed the number of tweets and the total number of unique words extracted from the text data using the CountVectorizer. This information is obtained from the shape of the `countVector` matrix.

```
275681 Number of tweets has 118531 words
```

5 Sentiment Analysis

Imported the TextBlob module from textblob, which is a library for processing textual data. Defined a function `polarity(text)` that calculates the polarity of each text using TextBlob, representing the sentiment of the text ranging from -1 (negative) to 1 (positive). Applied the `polarity()` function to the 'content' column of the dataframe `df`, storing the polarity scores in a new column 'polarity'. Defined another function `sentiment(label)` to categorize the polarity scores into three sentiment categories: Negative, Neutral, and Positive. Applies the `sentiment()` function to the 'polarity' column, storing the sentiment labels in a new column 'sentiment'.

6 Sentiment Analysis and Visualization

Sentiment analysis is conducted using TextBlob, categorizing text into negative, neutral, or positive sentiments. Visualization techniques such as countplots, pie charts, and word clouds are employed to visualize sentiment distributions and frequent words.

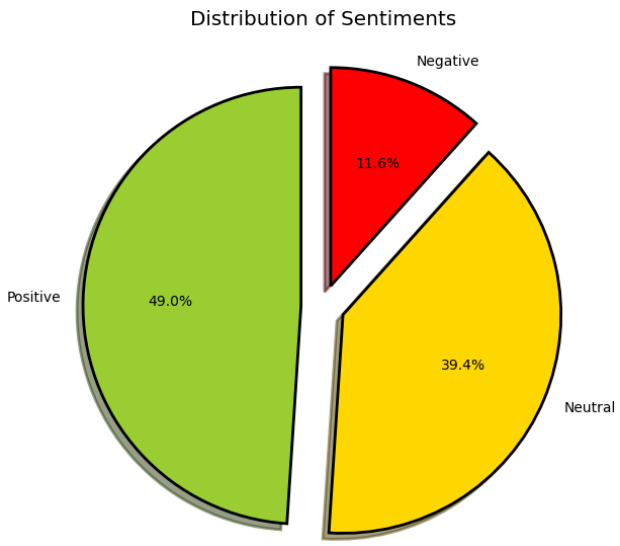


Figure 1: Pie Chart

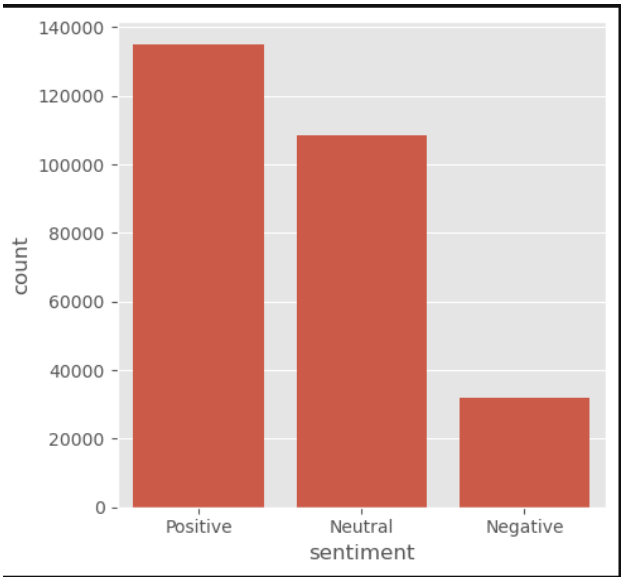


Figure 2: Bar Chart

Figure 3: Pie chart and bar chart in the same line

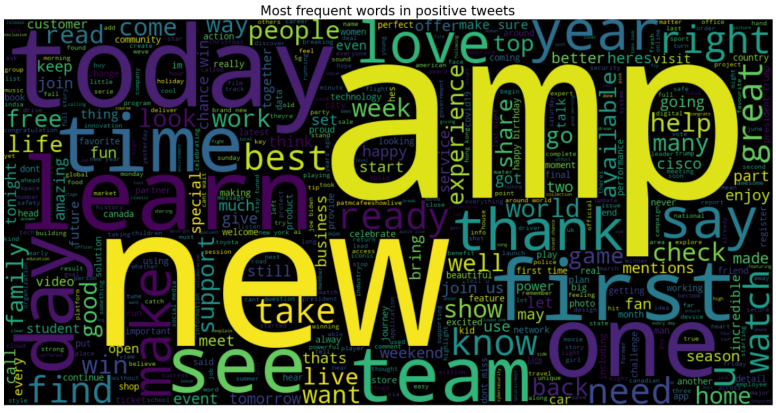


Figure 4: Word cloud representing frequent words

6.1 Visualization

Utilized seaborn and matplotlib for visualization. Created a countplot to visualize the distribution of sentiments ('Negative', 'Neutral', 'Positive'). Created a pie chart to visualize the proportion of each sentiment category. Generated word clouds to display the most frequent words in positive, negative, and neutral tweets, respectively.

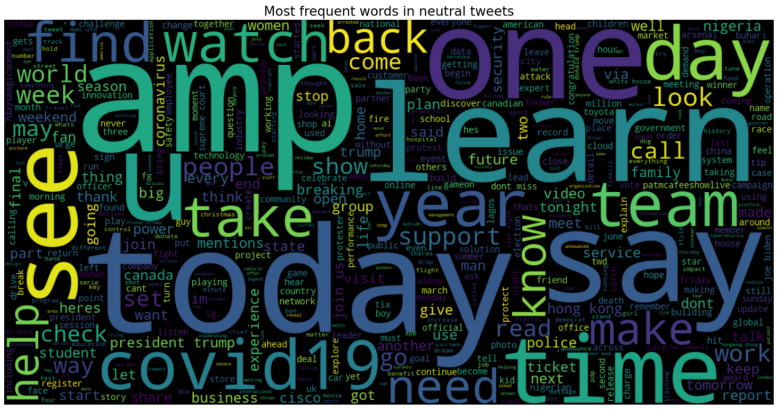


Figure 5: Word cloud representing frequent words

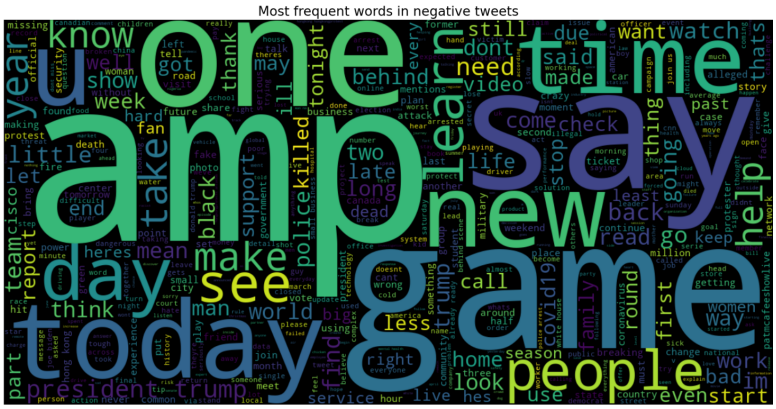


Figure 6: Word cloud representing frequent words

7 Conclusion

The code successfully preprocesses the text data by removing stopwords, punctuation, URLs, mentions, and converting text to lowercase. The resulting dataframe contains cleaned and processed text data, ready for further analysis or modeling. Further analysis or modeling can be conducted using the preprocessed text data to derive meaningful insights or predictions.

8 Recommendations

It's essential to evaluate the impact of each preprocessing step on the analysis or model performance to determine the most suitable preprocessing pipeline. Visualization or summary statistics of the preprocessed text data could help understand the distribution of words and identify potential patterns or insights. Experiment with different visualization techniques or parameters to enhance the presentation and interpretation of the sentiment analysis results.

9 References

1. Python Documentation. `https://docs.python.org/`
2. Pandas Documentation. `https://pandas.pydata.org/docs/`
3. NLTK Documentation. `https://www.nltk.org/`
4. Seaborn Documentation. `https://seaborn.pydata.org/`
5. Matplotlib Documentation. `https://matplotlib.org/`