Project Title:

Sentimental-analysis-and-Emotion-detection

(By SEMESTER – VII of IV Year M.Sc(CA & IT) 2022-23)

Submitted By:

Roll Number	Name
4179	**Bharvad Nisha Bhalabhai**
4184	**Chauhan Sneha Maheshbhai**
4235	**Salaliya Krupali Nimeshbhai**

Date of submission: 27 December,2023

Submitted To

K. S. School of Business Management

M.Sc. - Computer Application and Information Technology

INDEX

1.Introduction

The Sentiment Analysis and Emotion Detection project is a desktop application designed to analyze and interpret sentiments and emotions in textual data. Natural Language Processing (NLP) techniques, this tool offers a user-friendly interface for both live text input and the analysis of text files, with the goal of determining whether the content expresses positive, negative, or neutral sentiments and identifying the predominant emotions present in the text.

1.1 Project Context

In an era where digital communication prevails, understanding the sentiment and emotion embedded in textual data is paramount. The Sentiment and Emotion Analysis Tool aims to bridge the gap between raw text and actionable insights, offering users a comprehensive solution for interpreting the emotional tone of live text and text files. Leveraging advanced Natural Language Processing (NLP) techniques, this tool is poised to provide users with a versatile and user-friendly platform for sentiment and emotion analysis.

1.2 Motivation

The motivation behind this project lies in the growing need for businesses, researchers, and individuals to extract valuable insights from textual data. Whether gauging customer sentiment, monitoring social media trends, or understanding the emotional impact of written content, the Sentiment and Emotion Analysis Tool addresses these needs by offering a robust and accessible solution.

1.3 Objectives

- Develop a versatile sentiment analysis engine for live text input.
- Implement an advanced emotion analysis module using pre-trained language models.
- Enable users to perform sentiment and emotion analysis on text files.
- Provide users with visually informative data representations, including sentiment distribution plots and word clouds.
- Create an interactive Graphical User Interface (GUI) using Tkinter for seamless user interaction.
- Ensure efficient handling of large text files for analysis.

2.PROJECT OVERVIEW

The project utilizes natural language processing (NLP) techniques and pre-trained models to perform sentiment analysis and emotion detection. It features a graphical user interface (GUI) implemented using Tkinter, allowing users to interact with the application seamlessly. The functionality is divided into two main tabs: one for analyzing individual text entries and another for processing text files. The application incorporates visualizations such as word clouds to enhance the interpretation of results.

2.1 Scope

The project encompasses the development of a user-friendly Sentiment and Emotion Analysis Tool with a broad scope, including live text input analysis and the processing of text files. The tool will incorporate features for sentiment analysis, emotion analysis, and data visualization, ensuring a holistic approach to understanding textual data.

2.2 The project includes several key functions:

- **Sentiment Analysis (Live Text):** Analyzes sentiment of live text input and displays the results.
- **Emotion Analysis (Live Text):** Analyzes emotion of live text input and displays the results.
- **Word Cloud Generation:** Generates word clouds based on the input text.
- **File Processing:** Opens and processes text files for sentiment and emotion analysis.

2.3 Expected Outcomes

- A functional Sentiment and Emotion Analysis Tool with an intuitive user interface.
- Accurate sentiment and emotion analysis results for both live text and text files.
- Informative data visualizations, including sentiment distribution plots and word clouds.

2.4 Core Packages and Technologies Used

- **NLTK (Natural Language Toolkit):** For tokenization, lemmatization, and sentiment analysis.
- **Matplotlib and Seaborn:** For data visualization and plotting.
- **Transformers:** For utilizing pre-trained language models.
- **Pandas:** For data manipulation and analysis.
- **WordCloud:** For generating word clouds based on textual data.
- **Tkinter:** For building the graphical user interface (GUI).

2.5 Project Structure and Layout

- **Tab 1 - Text Sentiment and Emotion Analysis:** Allows users to input live text for sentiment and emotion analysis. Results are displayed along with options to visualize word clouds.

- **Tab 2 - File Sentiment and Emotion Analysis**: Enables users to open and analyze text files. The tool provides sentiment and emotion analysis for the first 80 lines of the file and visualizes the results through plots and word clouds.

- **Tab 3 - About:** Provides information about the project and its purpose.

2.6 Future Enhancements

As this tool evolves, potential future enhancements may include:

- Integration with additional pre-trained models for sentiment and emotion analysis, expanding the tool's capabilities.
- Enhanced visualization options for a more comprehensive analysis of sentiment and emotion patterns.
- Improved file processing capabilities to accommodate larger datasets and diverse file formats.

3.METHODOLOGY

The methodology involves several key steps. For individual text analysis, the text undergoes preprocessing, including lowercasing, punctuation removal, and lemmatization. Sentiment analysis is performed using the VADER sentiment analyzer. Emotion detection, on the other hand, leverages a pre-trained model ("arpanghoshal/EmoRoBERTa") from the Transformers library. The GUI is implemented with Tkinter for an intuitive user experience.

1. Data Input

1.1 Live Text Analysis

For live text analysis, the user inputs textual data through the dedicated input field provided in the "Text Sentiment and Emotion Analysis" tab. The input is then preprocessed to ensure consistency and accuracy in subsequent analysis steps.

1.2 File Processing

When analyzing text files, the user selects a file using the "Open File" button in the "File Sentiment and Emotion Analysis" tab. The content of the file, limited to the first 80 lines for efficiency, is read and processed for sentiment and emotion analysis.

2. Text Preprocessing

Before conducting sentiment and emotion analysis, the input text undergoes a series of preprocessing steps to enhance the accuracy of the results. These steps include:

- **Lowercasing**: All text is converted to lowercase to ensure uniformity.
- **Punctuation Removal:** Special characters and punctuation marks are removed to focus on the semantic content.
- **Tokenization:** The text is tokenized into individual words for further analysis.
- **Stopword Removal**: Common English stopwords are removed to reduce noise in the analysis.
- **Lemmatization:** Words are lemmatized to convert them to their base form, improving analysis accuracy.
- The cleaned and preprocessed text is then ready for sentiment and emotion analysis.

3. Sentiment Analysis

- Sentiment analysis is performed using the Sentiment Intensity Analyzer from the NLTK library. This analyzer provides a compound sentiment score ranging from -1 (negative) to 1 (positive), with 0 indicating neutral sentiment. The sentiment labels (positive, negative, neutral) are determined based on the compound score.
- The results, including sentiment scores and labels, are displayed to the user in the "Text Sentiment and Emotion Analysis" tab.

4. Emotion Analysis

Emotion analysis is conducted using the EmoRoBERTa model from the Transformers library. This pre-trained model is specifically designed for emotion detection. The input text is passed through the model, and the primary emotion label is extracted. The detected emotion label is then presented to the user.

5. Data Visualization

5.1 Sentiment Distribution Plot

In the "File Sentiment and Emotion Analysis" tab, the tool generates a count plot to visualize the distribution of sentiment labels (positive, negative, neutral) in the first 80 lines of the analyzed text file. Matplotlib and Seaborn libraries are employed for creating informative and visually appealing plots.

5.2 Word Cloud Generation

The tool offers users the option to generate word clouds based on the input text. The WordCloud library is utilized to create visually striking representations of the most frequently occurring words in the text. This visual aid helps users quickly grasp the key themes and sentiments present in the analyzed text.

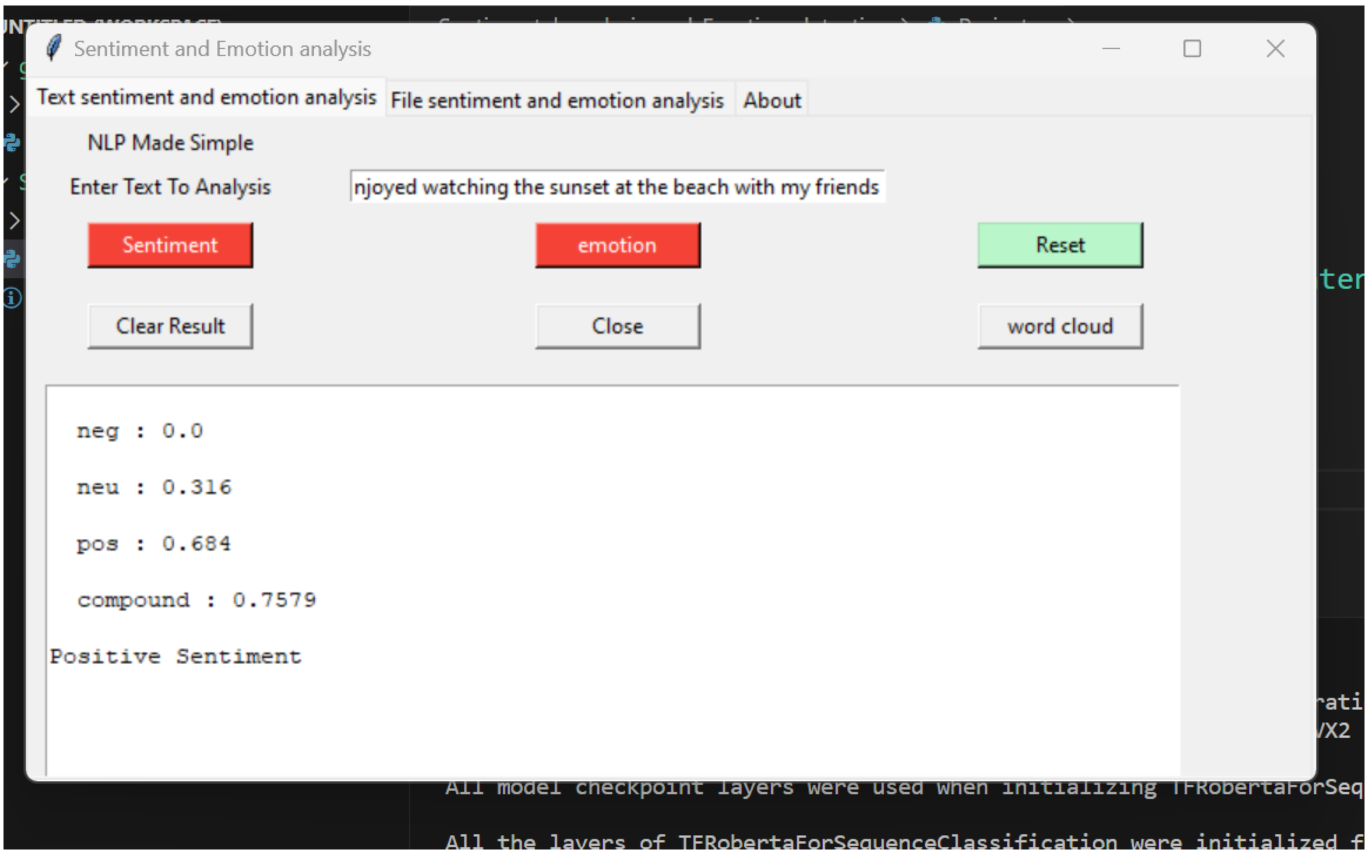
6. User Interaction and Interface

The Graphical User Interface (GUI) is developed using Tkinter, providing users with an intuitive and interactive experience. Tabs are organized to separate functionalities, ensuring clarity and ease of navigation. Buttons and entry fields enable user input and interaction with the tool.

4.RESULT

➢ Text Sentiment Analysis:

- Users can input text in the designated entry field.
- Sentiment analysis is performed using the VADER sentiment analyzer.
- The result is displayed in the GUI, indicating whether the sentiment is positive, negative, or neutral.
- "I enjoyed watching the sunset at the beach with my friends."


Sentiment and Emotion analysis
Text sentiment and emotion analysis
File sentiment and emotion analysis
About
NLP Made Simple
Enter Text To Analysis
Sentiment
emotion
Reset
Clear Result
Close
word cloud

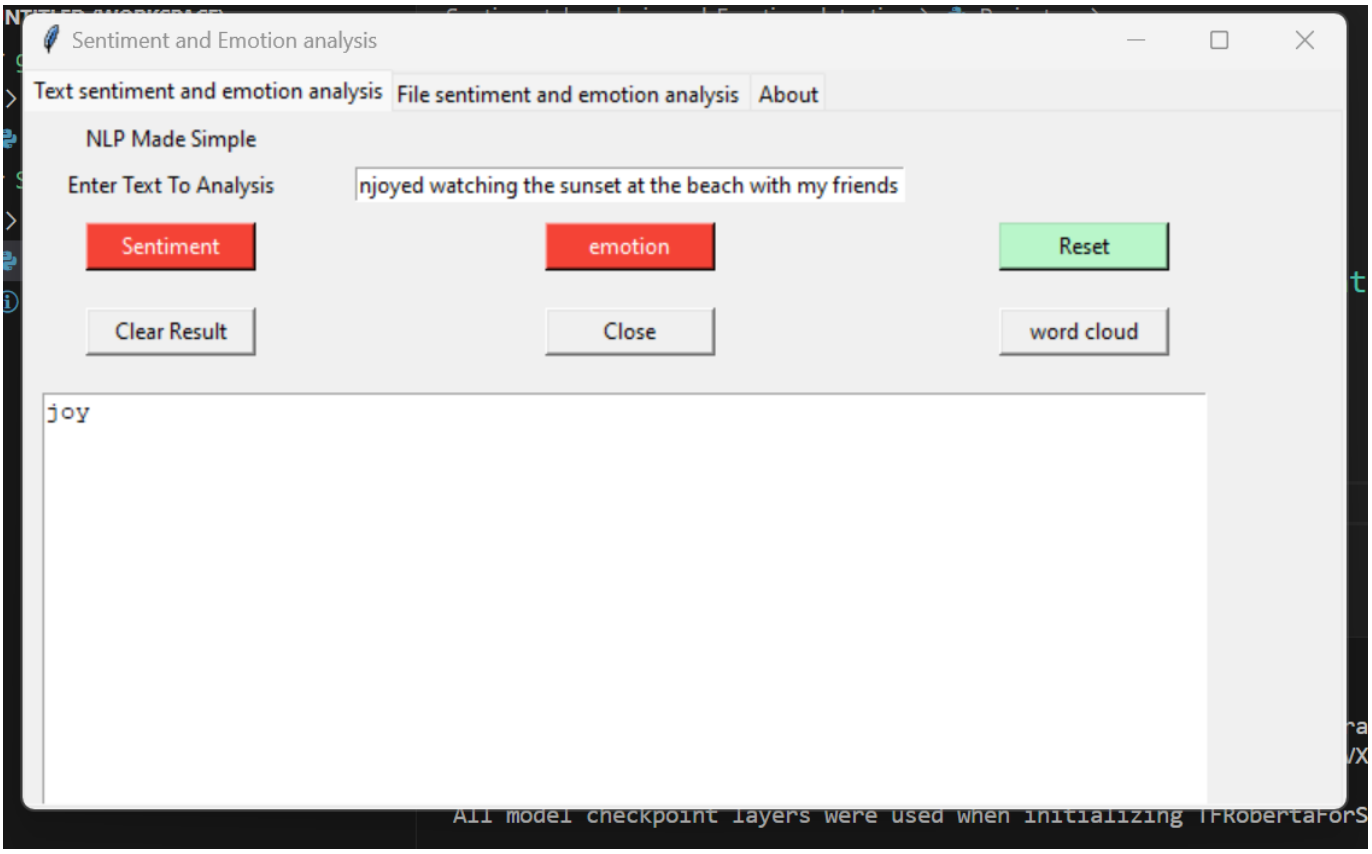
```
njoyed watching the sunset at the beach with my friends
```

```
neg : 0.0

neu : 0.316

pos : 0.684

compound : 0.7579

Positive Sentiment
```

```
All model checkpoint layers were used when initializing TFRobertaForSeq
All the layers of TFRobertaForSequenceClassification were initialized f
```

➢ Text Emotion Detection:

- Users input text for emotion analysis.
- Emotion detection is performed using a pre-trained model ("arpanghoshal/EmoRoBERTa") from the Transformers library.
- The predominant emotion label is displayed in the GUI.
- "I enjoyed watching the sunset at the beach with my friends."

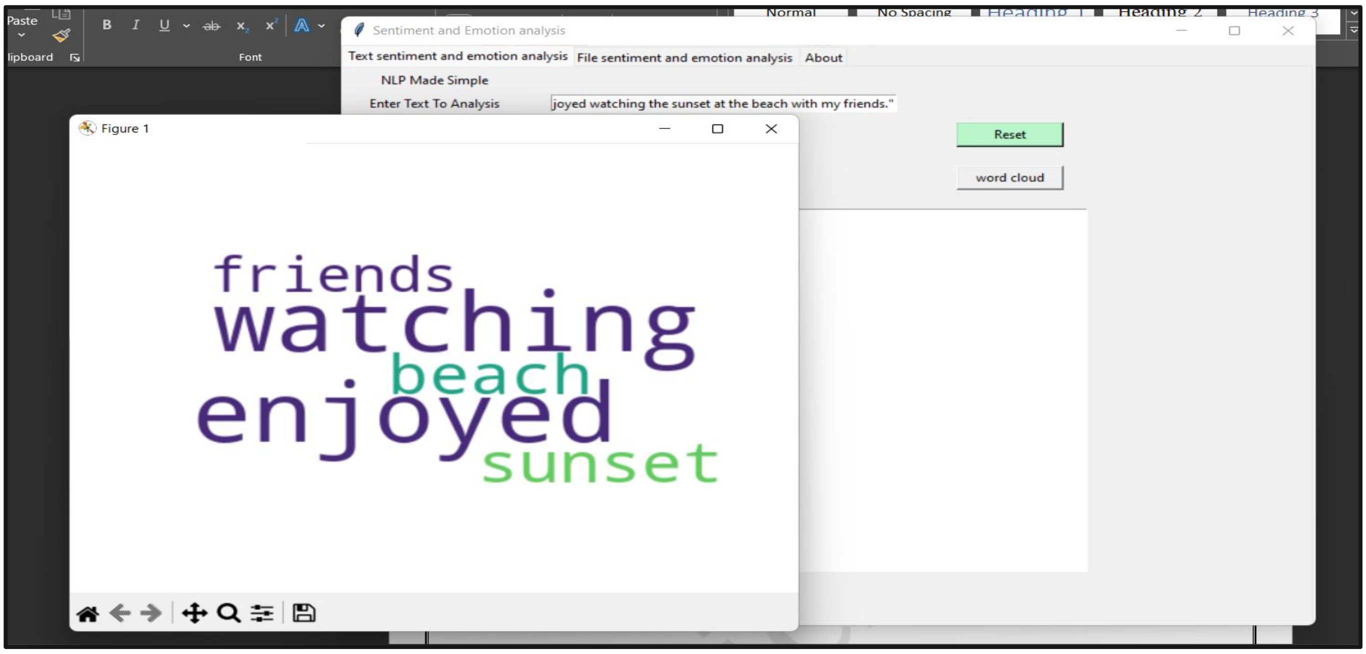

Sentiment and Emotion analysis
Text sentiment and emotion analysis
File sentiment and emotion analysis
About
NLP Made Simple
Enter Text To Analysis
Sentiment
emotion
Reset
Clear Result
Close
word cloud

njoyed watching the sunset at the beach with my friends

```
joy
```

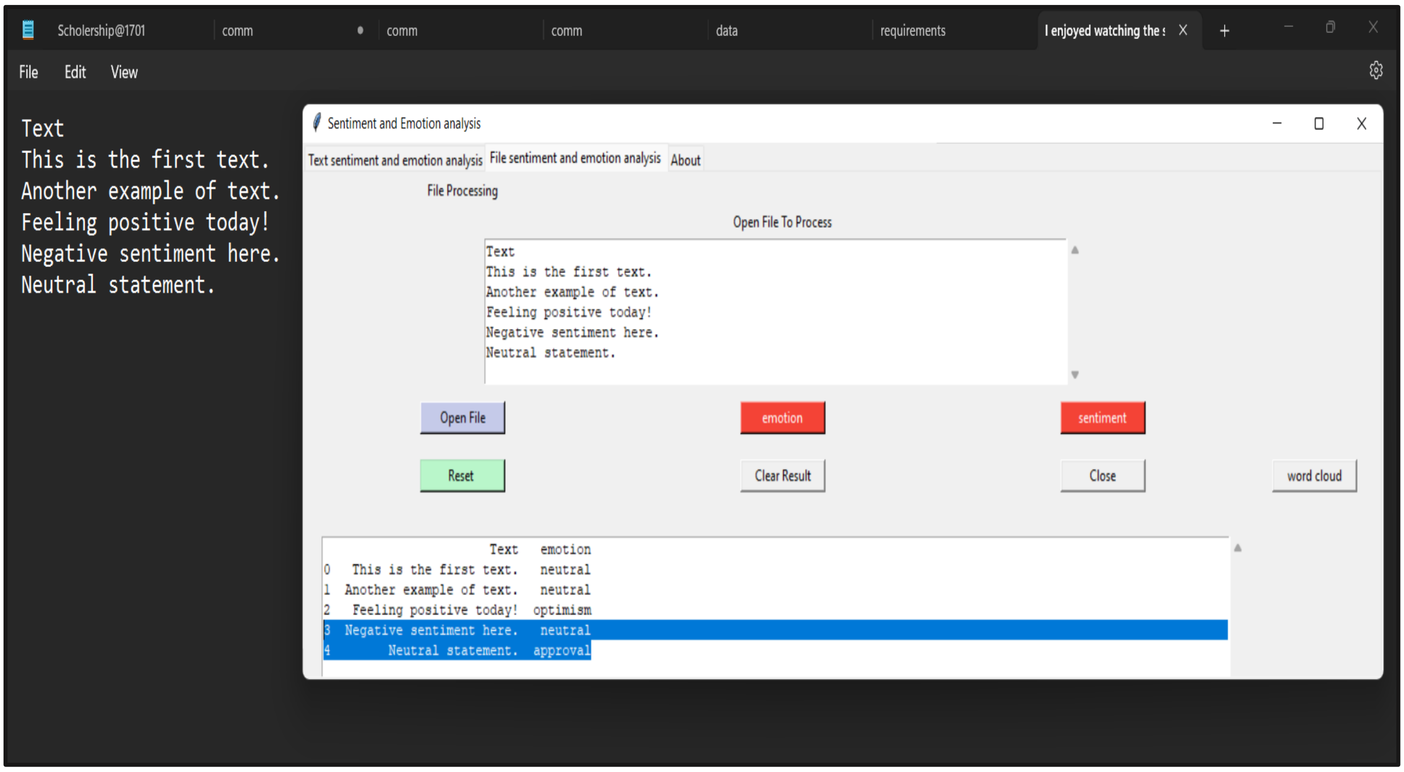
```
All model checkpoint layers were used when initializing TFRobertaForS
```

➢ **Word Cloud Generation:**
- Users can generate a word cloud based on the input text.
- The word cloud visually represents the frequency of words in the text.


Sentiment and Emotion analysis
Text sentiment and emotion analysis
File sentiment and emotion analysis
About
NLP Made Simple
Enter Text To Analysis
joyed watching the sunset at the beach with my friends."
Reset
word cloud
Figure 1
friends
watching
beach
enjoyed
sunset

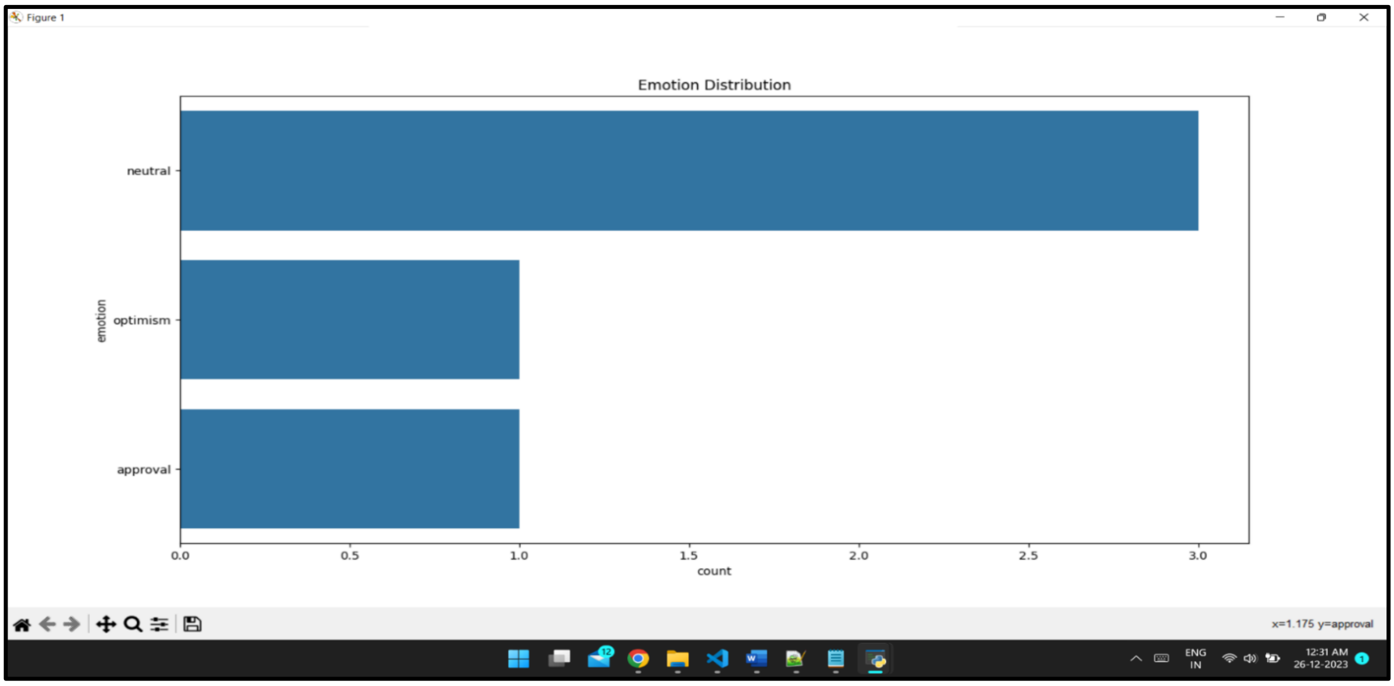
➢ File Processing:

- Users can open and read text files (supports formats such as .txt and .csv).
- Emotion detection and sentiment analysis are applied to the text in the file.
- The results are displayed in the GUI, along with visualizations such as count plots.



```
Text
This is the first text.
Another example of text.
Feeling positive today!
Negative sentiment here.
Neutral statement.
```

```
                        Text   emotion
0   This is the first text.   neutral
1  Another example of text.   neutral
2   Feeling positive today!  optimism
3  Negative sentiment here.   neutral
4        Neutral statement.  approval
```

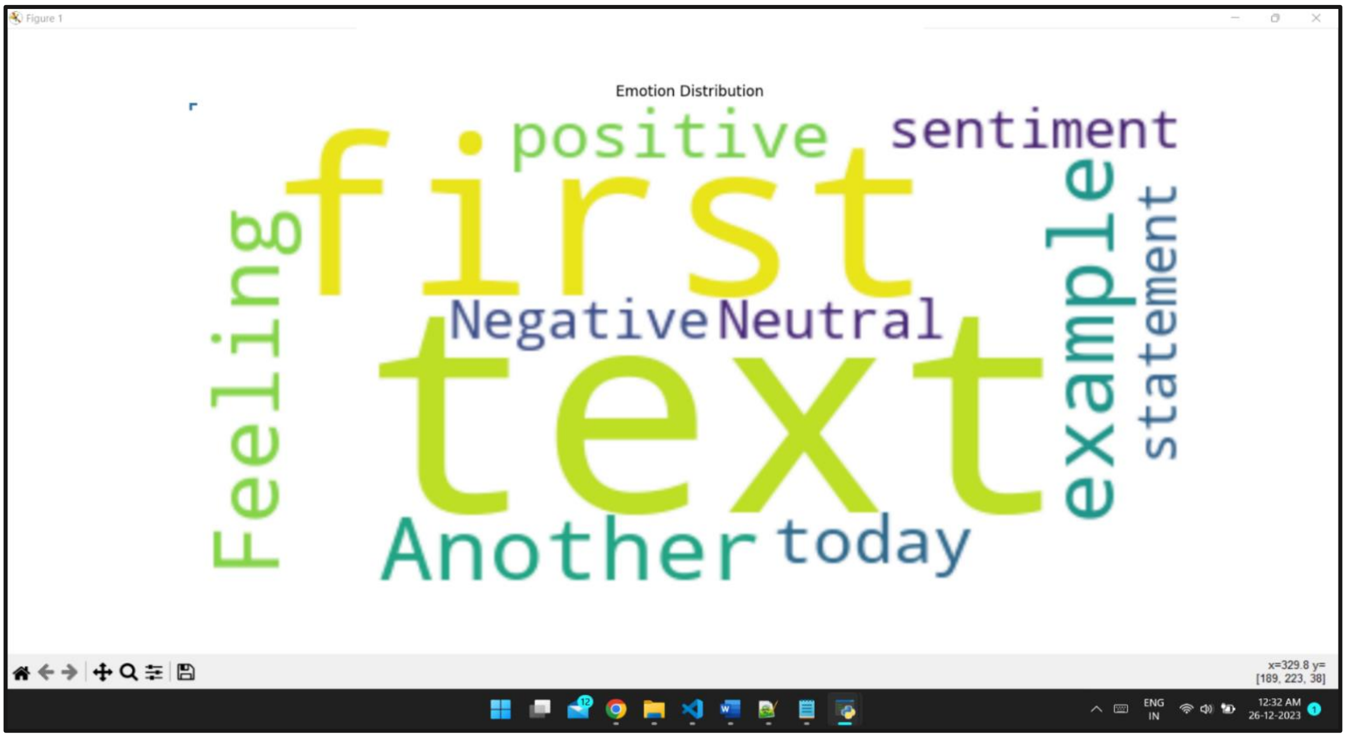


➢ Clearing and Resetting:

- Buttons are provided to clear the input text field, clear the display/result field, and reset the displayed text in the file processing tab.

➢ Visualization:

- The application utilizes visualizations, including count plots and word clouds, to enhance result interpretation.



CODE

```
# Core Packages
import string
from collections import Counter
import matplotlib.pyplot as plt
from nltk.corpus import stopwords
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
import tkinter as tk
from tkinter import *
from tkinter import ttk
from tkinter.scrolledtext import *
import tkinter.filedialog
from transformers import pipeline
import pandas as pd
from nltk.sentiment.vader import SentimentIntensityAnalyzer
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.pyplot as plt1
from wordcloud import WordCloud
from wordcloud import STOPWORDS

emotion = pipeline('sentiment-analysis', model='arpanghoshal/EmoRoBERTa')

# Structure and Layout
window = Tk()
window.title("Sentiment and Emotion analysis")
window.geometry("1000x400")
window.config(background='black')

# TAB LAYOUT
tab_control = ttk.Notebook(window)

tab1 = ttk.Frame(tab_control)
tab2 = ttk.Frame(tab_control)
tab3 = ttk.Frame(tab_control)

# ADD TABS TO NOTEBOOK
tab_control.add(tab1, text='Text sentiment and emotion analysis')
tab_control.add(tab2, text='File sentiment and emotion analysis ')
tab_control.add(tab3, text='About')

label1 = Label(tab1, text='NLP Made Simple', padx=5, pady=5)
label1.grid(column=0, row=0)
```

```
label2 = Label(tab2, text='File Processing', padx=5, pady=5)
label2.grid(column=0, row=0)

label3 = Label(tab3, text='About', padx=5, pady=5)
label3.grid(column=0, row=0)

tab_control.pack(expand=1, fill='both')

about_label = Label(tab3, text="emotions and sentiment analysis based on text and file \n  ", pady=5,
padx=5)
about_label.grid(column=0, row=1)


# Functions FOR NLP  FOR TAB ONE
def get_sentiment():
    live_text = str(raw_entry.get())
    lower_case = live_text.lower()
    cleaned_text = lower_case.translate(str.maketrans('', '', string.punctuation))
    # Using word_tokenize because it's faster than split()
    tokenized_words = word_tokenize(cleaned_text, "english")

    # Removing Stop Words
    final_words = []
    for word in tokenized_words:
        if word not in stopwords.words('english'):
            final_words.append(word)
    print(final_words)

    # Lemmatization - From plural to single + Base form of a word (example better-> good)
    lemma_words = []
    for word in final_words:
        word = WordNetLemmatizer().lemmatize(word)
        lemma_words.append(word)
    print(lemma_words)
    cleaned_text = " ".join(lemma_words)

    # Define a function for sentiment analysis
    def sentiment_analyse(sentiment_text):
         # Use SentimentIntensityAnalyzer to get sentiment scores
        score = SentimentIntensityAnalyzer().polarity_scores(sentiment_text)
        # Create a list of tuples for each sentiment score
        list = [(k, v) for k, v in score.items()]
         # Insert the sentiment scores into the Text widget
        for a, b in list:
            result = '\n  {} : {}  \n'.format(a, b)
            tab1_display.insert(tk.END, result)
        # Determine the overall sentiment based on the compound score
        if score['compound'] <= - 0.05:
            tab1_display.insert(tk.END, "\nNegative Sentiment\n")
        elif score['compound'] >= 0.05:
            tab1_display.insert(tk.END, "\nPositive Sentiment\n")
        else:
            tab1_display.insert(tk.END, "\nNeutral Sentiment\n")
    # Call the sentiment analysis function with the cleaned text
    sentiment_analyse(cleaned_text)


def get_emotion():
    # Get the text entered by the user
    live_emotion = str(raw_entry.get())
    # Convert the text to lowercase
    lower_case = live_emotion.lower()
    # Remove punctuation from the text
    cleaned_text = lower_case.translate(str.maketrans('', '', string.punctuation))
    # Apply an emotion analysis model to the cleaned text
    emotion_labels = emotion(cleaned_text)
    # Extract the predicted emotion label from the result
    result = emotion_labels[0]['label']
    # Insert the predicted emotion label into a text display widget
    tab1_display.insert(tk.END, result)


def word_cloud():
    # Get the text entered by the user
    live_text = str(raw_entry.get())
    # Remove punctuation from the text
    cleaned_text = live_text.translate(str.maketrans('', '', string.punctuation))
    # Generate a WordCloud object
    word_cloud = WordCloud(collocations=False, background_color='white').generate(cleaned_text)
    # Display the generated WordCloud using Matplotlib
    plt.imshow(word_cloud, interpolation='bilinear')
    # Turn off the axis for better visualization
    plt.axis("off")
    # Show the WordCloud plot
    plt.show()


# Clear entry widget
def clear_entry_text():
    entry1.delete(0, END)


def clear_display_result():
    tab1_display.delete('1.0', END)


# Clear Text  with position 1.0
```

```
def clear_text_file():
    displayed_file.delete('1.0', END)


# Clear Result of Functions
def clear_result():
    tab2_display_text.delete('1.0', END)


filename = []


def openfiles():
    # Open a file dialog to let the user choose a file
    file1 = tk.filedialog.askopenfilename(filetypes=(("Text Files", ".txt"), ("All files", "*"), ("csv files", ".csv")))
    # Read the content of the selected file
    read_text = open(file1).read()
    # Append the path of the selected file to the filename list
    filename.append(file1)
    # Insert the content of the selected file into the displayed_file text widget
    displayed_file.insert(tk.END, read_text)


def get_file_emotion():
    # Get the name of the file from the filename list
    file_name = filename[0]

    # Read the CSV file into a Pandas DataFrame
    large_text = pd.read_csv(file_name)

    # Define a function to get emotion label using the 'emotion' pipeline
    def get_emotion_label(text):
        return emotion(text)[0]['label']

    # Apply the get_emotion_label function to the 'Text' column for the first 80 rows
    large_text['emotion'] = large_text['Text'][0:80].apply(get_emotion_label)

    # Display the resulting DataFrame (first 10 rows) in the GUI
    result = large_text.head(10)
    tab2_display_text.insert(tk.END, result)

    # Create a count plot of emotion distribution and display it in the GUI
    sns.countplot(data=large_text, y='emotion').set_title("Emotion Distribution")
    plt.show()


def get_file_sentiment():
    # Get the name of the file from the filename list
    file_name = filename[0]
    # Create a SentimentIntensityAnalyzer instance
    sid = SentimentIntensityAnalyzer()

    # Read the CSV file into a Pandas DataFrame
    df = pd.read_csv(file_name)

    # Apply sentiment analysis to each review in the 'Text' column
    #The 'Text' column is necessary because it contains the textual content that you want to subject to
sentiment analysis.
    df['scores'] = df['Text'].apply(lambda review: sid.polarity_scores(review))

    # Extract the compound score from the scores dictionary
    df['compound'] = df['scores'].apply(lambda score_dict: score_dict['compound'])

    # Categorize the compound score as positive, negative, or neutral
    df['comp_score'] = df['compound'].apply(lambda c: 'positive' if c > 0 else ('negative' if c < 0 else
'Neutral'))

    # Display the resulting DataFrame (first 10 rows) in the GUI
    result = df.head(10)
    tab2_display_text.insert(tk.END, result)

    # Create a count plot of sentiment distribution and display it in the GUI
    sns.countplot(data=df, y='comp_score').set_title("Sentiment Distribution")
    plt.show()


def wordcloud():
    # Get the file name from the list of filenames
    file_name = filename[0]

    # Read the CSV file into a DataFrame
    df = pd.read_csv(file_name)

    # Concatenate all text entries in the 'Text' column into a single string
    text = " ".join(review for review in df.Text)

    # Generate a word cloud from the concatenated text
    word_cloud = WordCloud(collocations=False, background_color='white').generate(text)

    # Display the word cloud using Matplotlib
    plt.imshow(word_cloud, interpolation='bilinear')

    # Turn off the axis labels for better visualization
    plt.axis("off")

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

```
# MAIN NLP TAB
l1 = Label(tab1, text="Enter Text To Analysis")
l1.grid(row=1, column=0)

raw_entry = StringVar()
entry1 = Entry(tab1, textvariable=raw_entry, width=50)
entry1.grid(row=1, column=1)

# bUTTONS
button1 = Button(tab1, text="Sentiment", width=12, command=get_sentiment, bg='#f44336', fg='#fff')
button1.grid(row=3, column=0, padx=10, pady=10)
button2 = Button(tab1, text="emotion", width=12, command=get_emotion, bg='#f44336', fg='#fff')
button2.grid(row=3, column=1, padx=10, pady=10)

button3 = Button(tab1, text="Reset", width=12, command=clear_entry_text, bg="#b9f6ca")
button3.grid(row=3, column=2, padx=10, pady=10)

button4 = Button(tab1, text="Clear Result", width=12, command=clear_display_result)
button4.grid(row=4, column=0, padx=10, pady=10)
button5 = Button(tab1, text="Close", width=12, command=window.destroy)
button5.grid(row=4, column=1, padx=10, pady=10)
button5 = Button(tab1, text="word cloud", width=12, command=word_cloud)
button5.grid(row=4, column=2, padx=10, pady=10)

# Display Screen For Result
tab1_display = Text(tab1)
tab1_display.grid(row=7, column=0, columnspan=3, padx=10, pady=10)

# Allows you to edit
tab1_display.config(state=NORMAL)

# FILE READING  AND PROCESSING TAB
l1 = Label(tab2, text="Open File To Process")
l1.grid(row=1, column=1)

displayed_file = ScrolledText(tab2, height=7)  # Initial was Text(tab2)
displayed_file.grid(row=2, column=0, columnspan=3, padx=5, pady=3)

# BUTTONS FOR SECOND TAB/FILE READING TAB
b0 = Button(tab2, text="Open File", width=12, command=openfiles, bg='#c5cae9')
b0.grid(row=3, column=0, padx=10, pady=10)

b1 = Button(tab2, text="Reset ", width=12, command=clear_text_file, bg="#b9f6ca")
b1.grid(row=5, column=0, padx=10, pady=10)

b2 = Button(tab2, text="emotion", width=12, command=get_file_emotion, bg='#f44336', fg='#fff')
b2.grid(row=3, column=1, padx=10, pady=10)
```

```
b3 = Button(tab2, text="sentiment", width=12, command=get_file_sentiment, bg='#f44336', fg='#fff')
b3.grid(row=3, column=2, padx=10, pady=10)

b6 = Button(tab2, text="Clear Result", width=12, command=clear_result)
b6.grid(row=5, column=1, padx=10, pady=10)

b7 = Button(tab2, text="Close", width=12, command=window.destroy)
b7.grid(row=5, column=2, padx=10, pady=10)
b8 = Button(tab2, text="word cloud", width=12, command=wordcloud)
b8.grid(row=5, column=3, padx=10, pady=10)

# Display Screen

# tab2_display_text = Text(tab2)
tab2_display_text = ScrolledText(tab2, height=15, width=125)
tab2_display_text.grid(row=7, column=0, columnspan=3, padx=20, pady=25)

# Allows you to edit
tab2_display_text.config(state=NORMAL)

window.mainloop()
```

5.CONCLUSION

In conclusion, the sentiment and emotion analysis project offers a user-friendly interface for individuals to assess the sentiment and emotion of text. By combining VADER sentiment analysis and a pre-trained emotion detection model, users can gain insights into the emotional tone of both individual text entries and larger text files. The incorporation of visualizations, such as word clouds and count plots, enhances the interpretability of the results. Overall, the project provides a convenient tool for users seeking to understand the emotional content within textual data.

6.REFERENCES

- https://www.oracle.com/in/
- http://stackoverflow.com
- http://www.github.com