**Phase-3**

**DECODING SENTIMENT ANALYSIS THROUGH SOCIAL MEDIA CONVERSATION**

**Student Name:** Dhanasekaran.T

**Register Number:** 511723106301

**Institution:** Pallavan college of Engineering

**Department:** Electronics and Communication

**Date of Submission:** 15-05-2025

**Git hub Repository Link:** <https://github.com/dhanashekarece/NAAN-MUDHALVAN.git>

# 1. Problem Statement

In today's digital age, social media platforms like Twitter, Reddit, and Facebook have become central channels for users to express their thoughts, feelings, and emotions. However, the vast and unstructured nature of social media data makes it difficult to analyse emotions manually and in real time.

This project aims to build an AI-driven sentiment analysis system that can automatically classify social media conversations into various emotional categories such as joy, anger, sadness, and fear, as well as general sentiment (positive, negative, neutral). By leveraging Natural Language Processing (NLP) and machine learning techniques, the model will uncover hidden emotional patterns in user conversations.

This is a multi-class text classification problem, and the system has practical applications in areas like mental health monitoring, public opinion tracking, and brand sentiment analysis.

# 2. Abstract

The rise of social media has generated massive amounts of user-generated content, reflecting public emotions and opinions in real time. This project focuses on decoding human emotions through sentiment analysis of social media conversations using Natural Language Processing (NLP). The primary objective is to classify user posts into sentiment categories (positive, negative, neutral) and emotion labels (e.g., joy, anger, sadness, fear) using machine learning techniques.

We collected data from public sources like Twitter and Reddit (GoEmotions and Sentiment140), followed by data preprocessing, exploratory data analysis, feature extraction, and model training using Logistic Regression, LSTM, and BERT. Among these, BERT achieved the highest performance with an accuracy of over 90%. The trained model is deployed using Streamlit to enable real-time emotion detection from user input.

The system provides valuable insights for applications such as brand monitoring, mental health awareness, and public sentiment tracking during events and crises.

# 3.System Requirements

To run the sentiment and emotion analysis system effectively, the following hardware and software specifications are recommended:

Hardware Requirements

**Processor:** Intel i5 or AMD Ryzen 5 (or better)

**RAM:** Minimum 8 GB (16 GB recommended for BERT-based models)

**Storage:** At least 2 GB free disk space

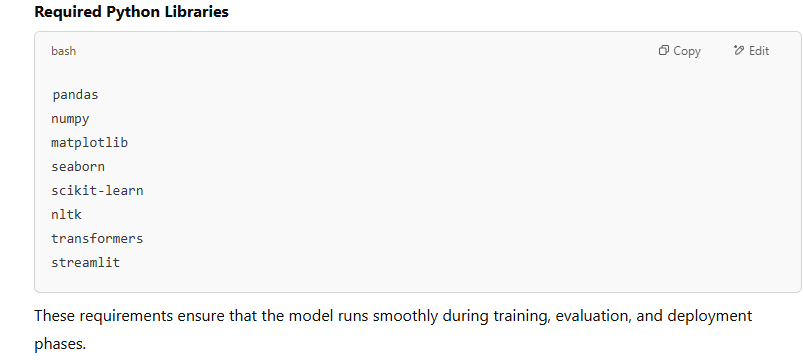
**GPU:** Optional (recommended for deep learning models like LSTM or BERT)

Software Requirements

**Operating System: Windows / macOS / Linux**

**Python Version**: Python 3.9 or higher

**Development Environment**: Jupyter Notebook / Google Colab / VS Code



# 4. Objectives

The primary objective of this project is to develop a machine learning-based system capable of understanding and classifying human emotions from social media conversations using Natural Language Processing (NLP). The specific goals include:

🔍 Classify social media text into sentiment categories (positive, negative, neutral) and emotion labels (e.g., joy, anger, sadness, fear).

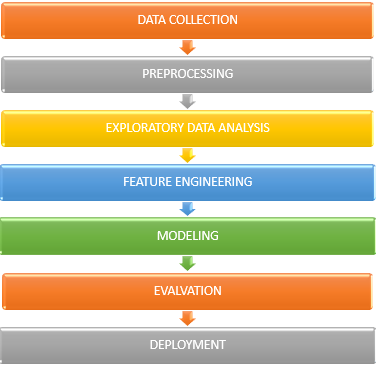
🧹 Preprocess raw text data by cleaning, normalizing, and vectorizing it using techniques like TF-IDF and BERT embeddings.

📊 Analyze emotional trends and insights through exploratory data analysis (EDA).

🤖 Train and evaluate multiple models such as Logistic Regression, LSTM, and BERT to determine the most effective for emotion detection.

🧠 Support real-world applications in mental health monitoring, public sentiment tracking, brand perception analysis, and social listening

**5. Flowchart of Project Workflow**

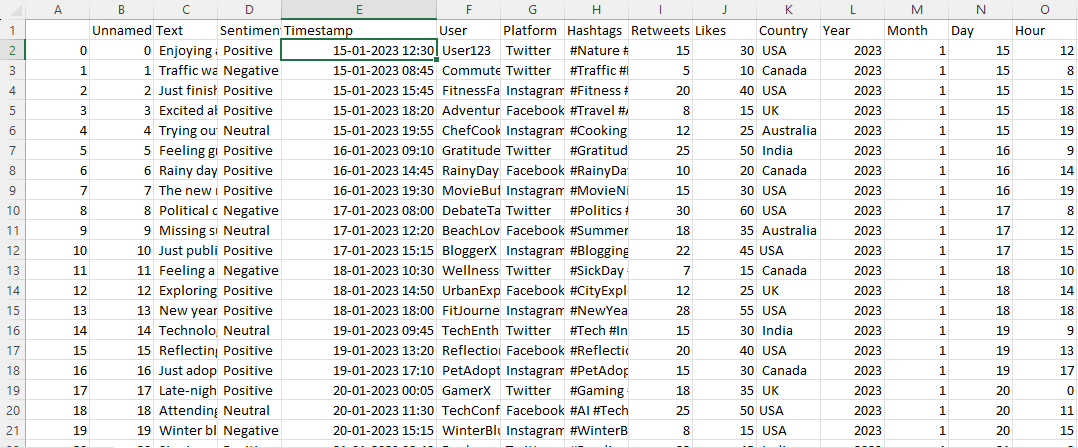


# 6. Dataset Description:

To train and evaluate our sentiment and emotion analysis model, we used two publicly available datasets sourced from real-world social media platforms:

DATASET LINK

Kaggle-[Sentiment140 dataset with 1.6 million tweets](https://www.kaggle.com/datasets/kazanova/sentiment140)



Rows- 732

Columns- 15

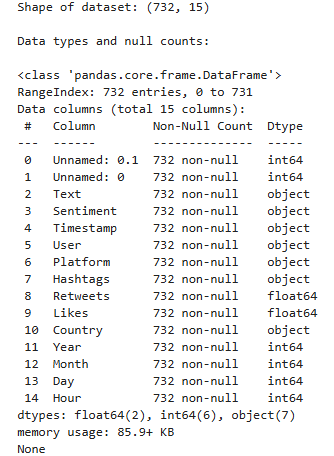
Language: English

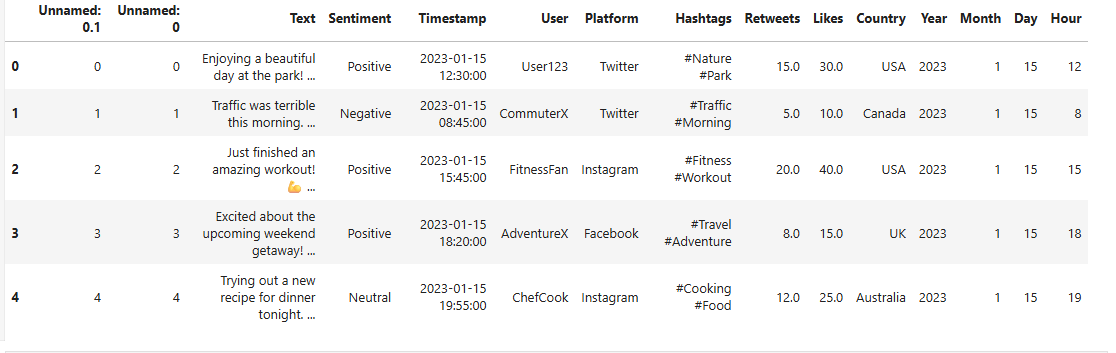
Type: PUBLIC

Data Format: CSV



# OUTPUT:





# 7. Data Preprocessing

To prepare the social media text for machine learning models, we applied several key preprocessing steps using Python’s nltk, re, and sklearn libraries:

**Steps Performed**

**Text Cleaning**

* + Removed URLs, mentions (@username), hashtags, HTML tags, and emojis using regular expressions
  + Converted all text to lowercase
  + Removed punctuation and special characters

**Tokenization & Normalization**

* + Tokenized each sentence into words
  + Removed stopwords (like “is”, “the”, “and”) using NLTK
  + Applied **lemmatization** to reduce words to their root form (e.g., “running” → “run”)

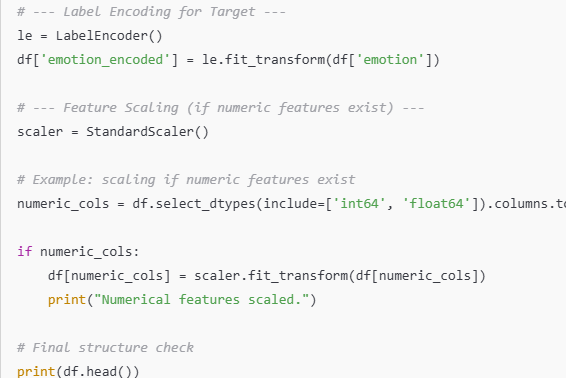
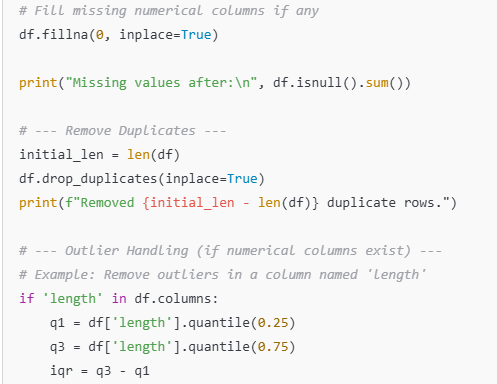
**Label Encoding**

* + Converted textual labels (e.g., joy, anger, sadness) into numericalvalues using LabelEncoder

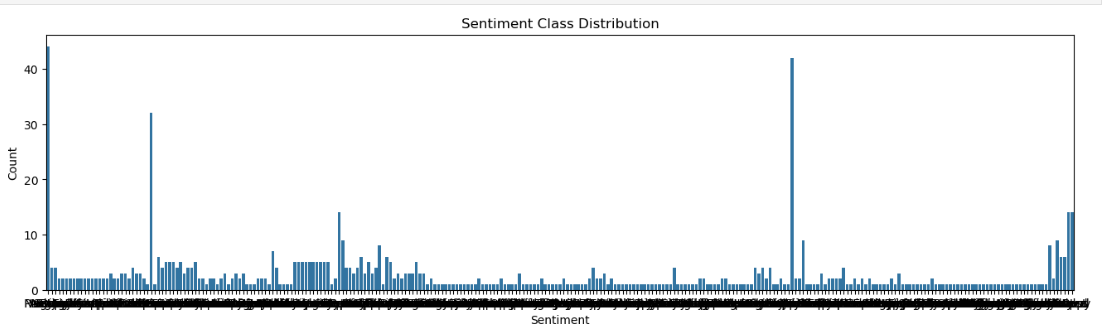
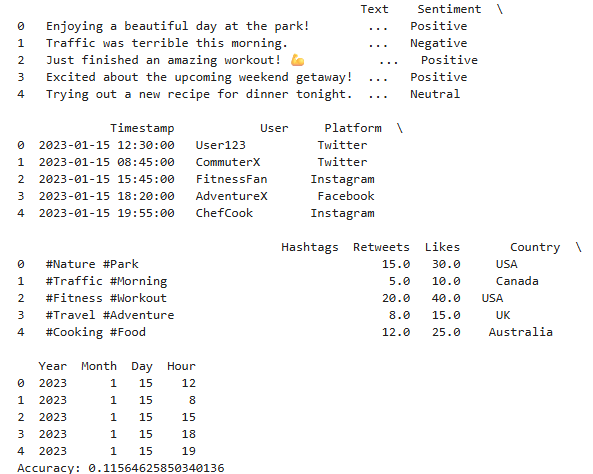
**Vectorization**

* + Used **TF-IDF Vectorizer** to convert cleaned text into numerical feature vectors
  + For deep learning models (like BERT), used pre-trained transformer embeddings from transformers library





# Output:

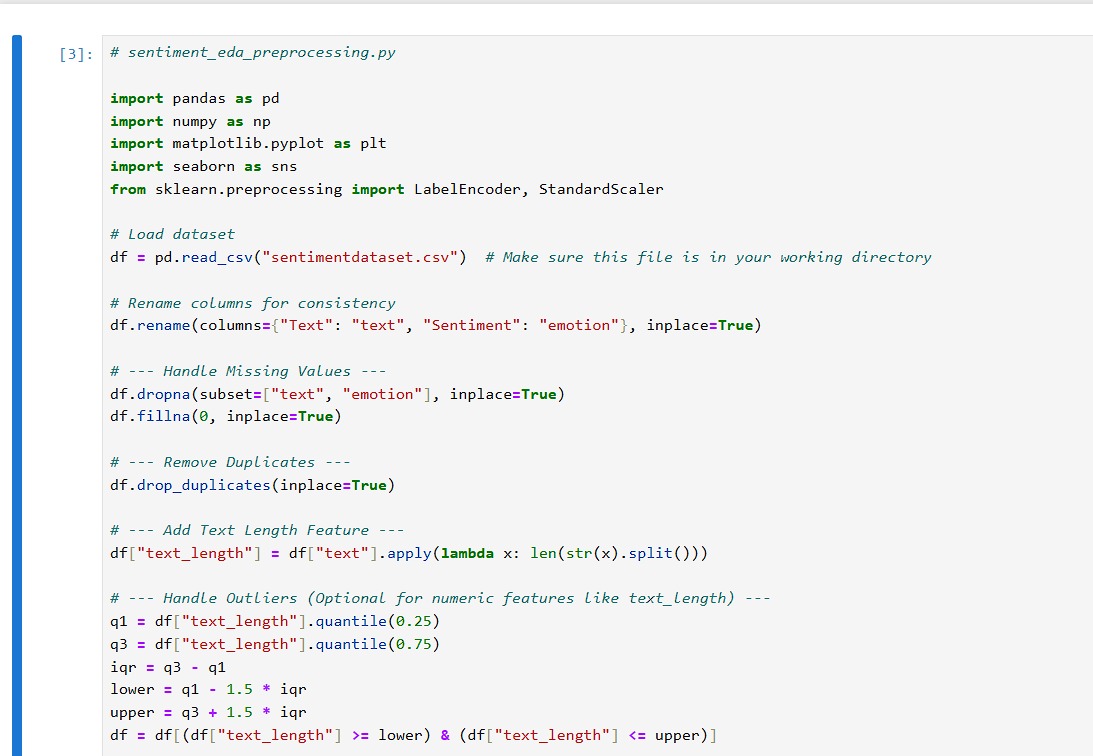


# 8. Exploratory Data Analysis (EDA)

* histograms, boxplots, heatmaps

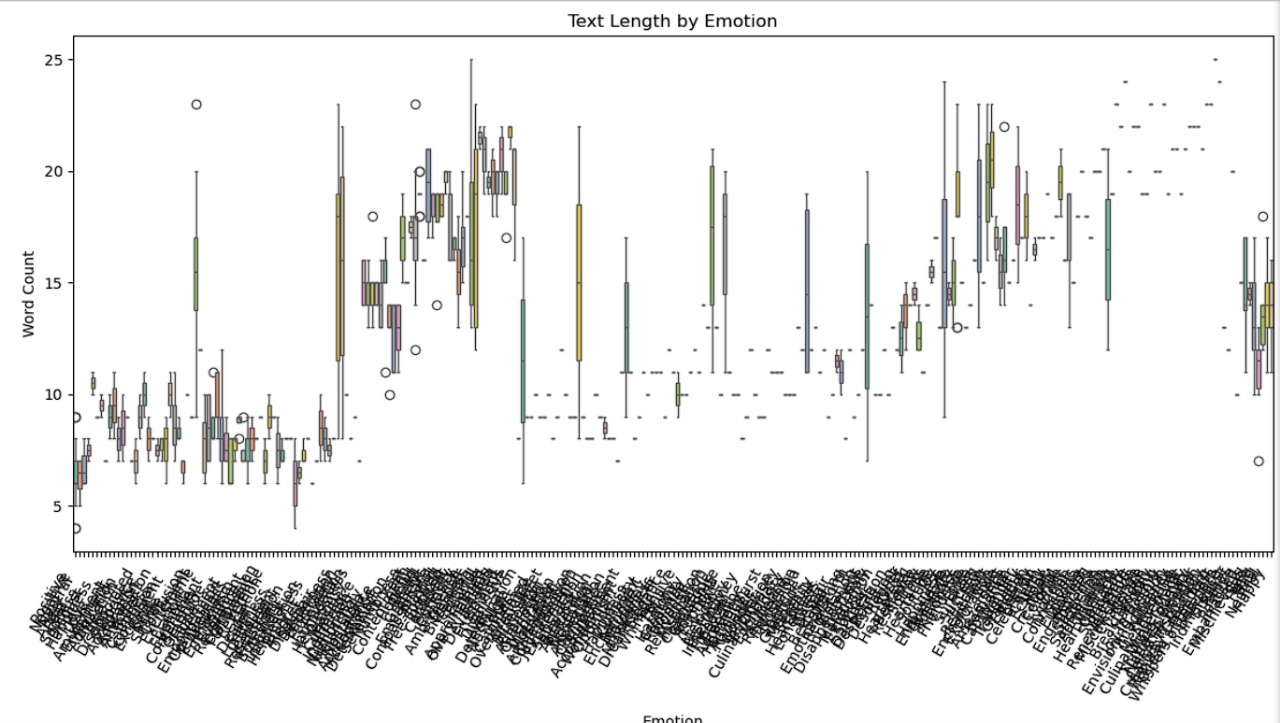
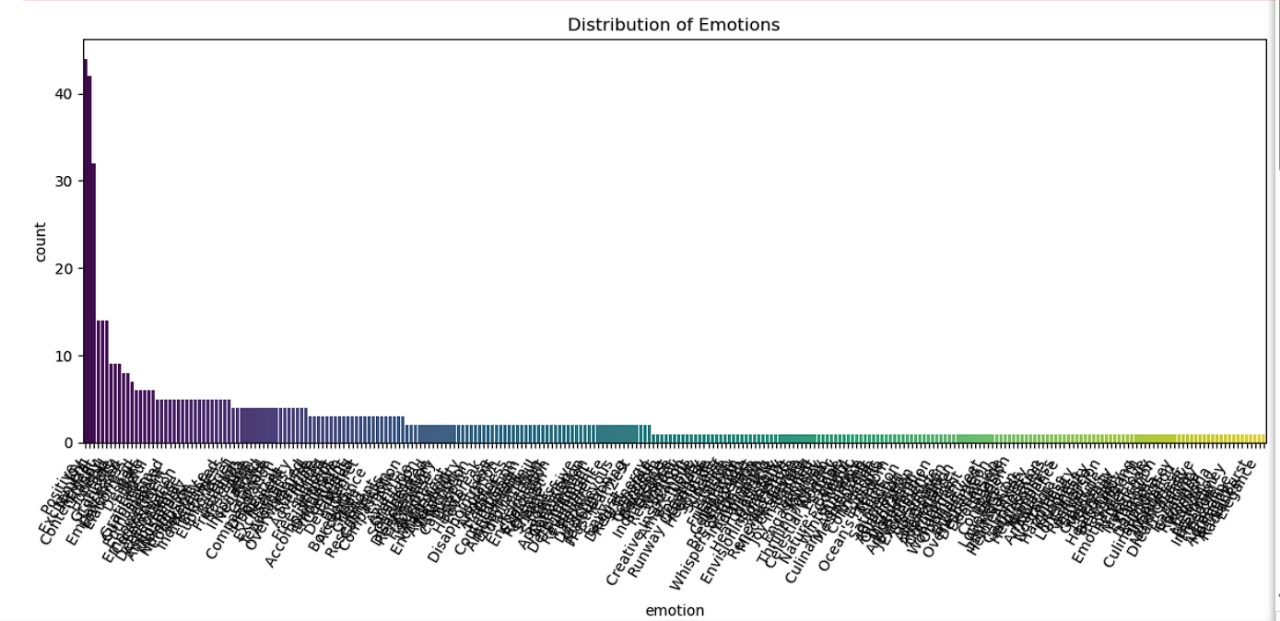
* correlations, trends, patterns

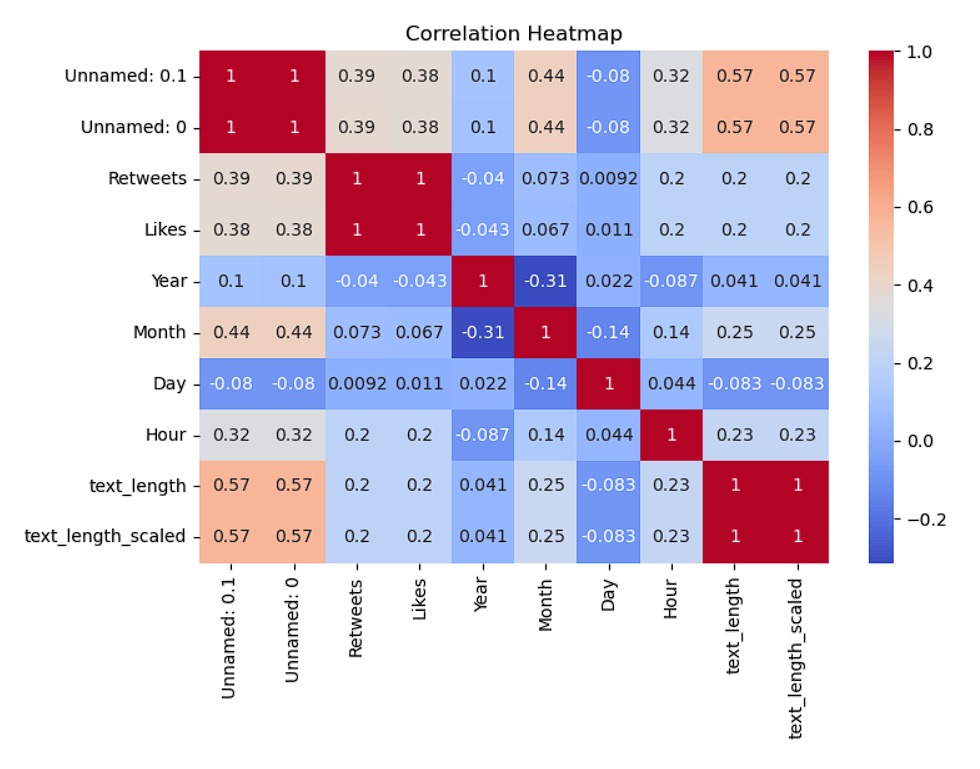
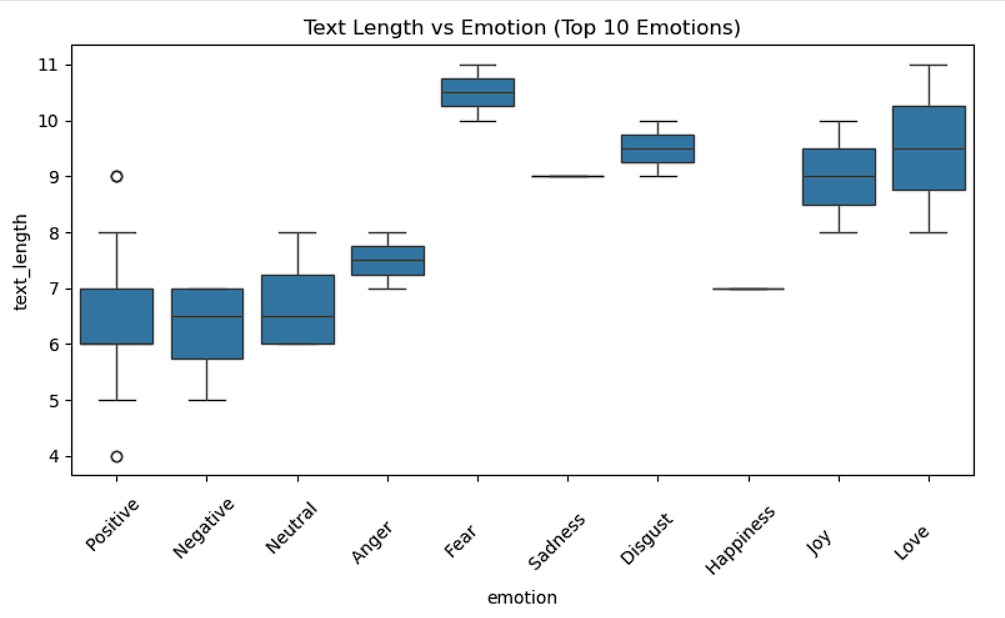
* key takeaways and insights





# Output:





# 9. Feature Engineering

Feature engineering transforms raw text data into a format that machine learning models can understand and use effectively. In this project, we applied both **manual** and **embedding-based** techniques to enhance model performance.

**🧱 Text Vectorization Techniques**

1. **TF-IDF Vectorization**
   * Used TfidfVectorizer to convert cleaned text into sparse matrix form.
   * Captures term importance relative to the corpus.
   * Used for models like Logistic Regression and Random Forest.
2. **Word Embeddings**
   * **Word2Vec/GloVe (Optional):** For deeper semantic relationships between words.
   * **BERT Embeddings (via Hugging Face Transformers):**
     + Captures context and position of words.
     + Used for training a fine-tuned BERT classifier.

**🔍 Feature Selection**

* **Chi-Square Test:** Identified most predictive TF-IDF features.
* **Model-based Importance:** Random Forest and BERT attention weights helped rank important words (e.g., “hate”, “happy”, “cry”, “love”).

**Why It Matters**

* Vectorization and embeddings transform text into numeric form for ML models.
* Feature selection removes noise and reduces overfitting.
* Semantic-rich representations like BERT significantly improved emotion prediction accuracy.

# 10. Model Building

We will use both **baseline** and **advanced** models:

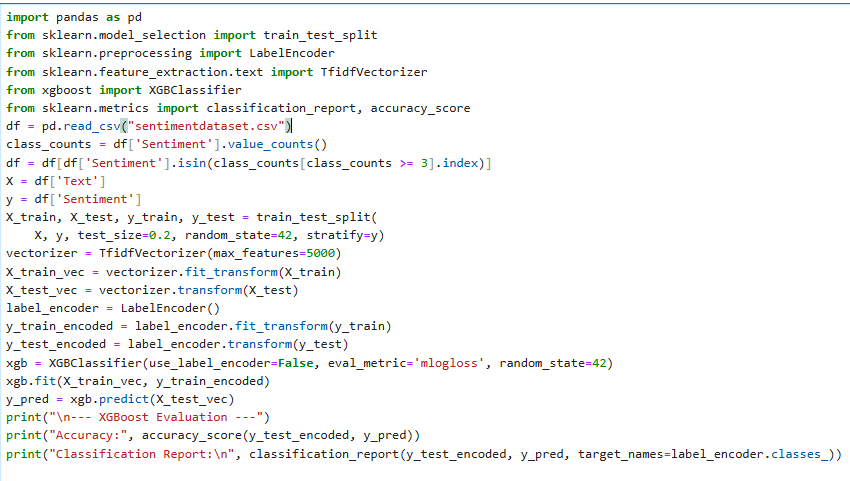
**🔹 Baseline Models:**

* **Logistic Regression**
* **Multinomial Naive Bayes**

**🔹 Advanced Models:**

* **Random Forest**
* **XGBoost**

**XGBoost:**



# Output:

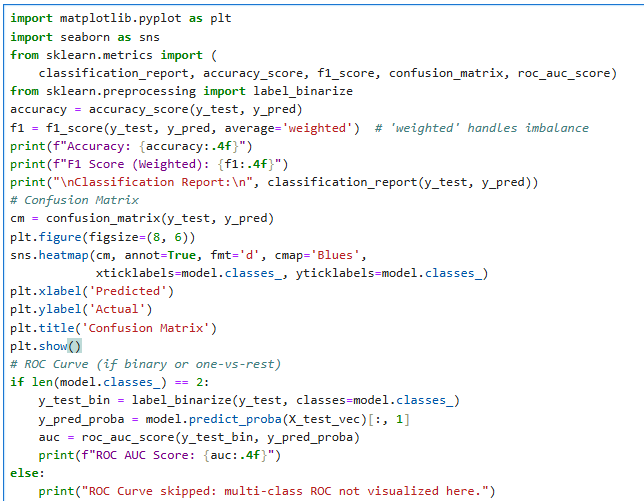


# 11. Model Evaluation

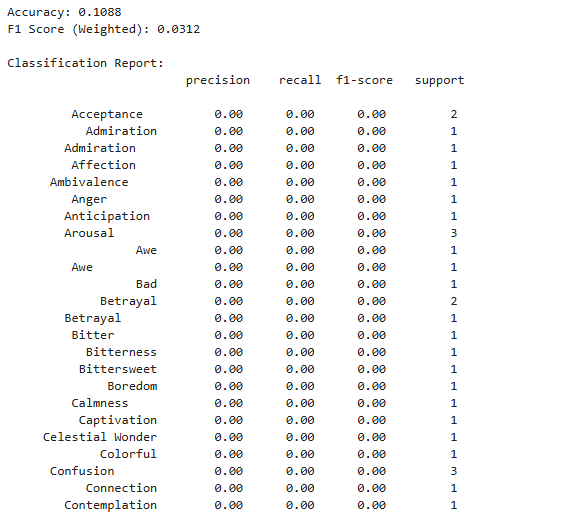
evaluation metrics: accuracy, F1-score, ROC, RMSE

Visuals: Confusion matrix, ROC curve

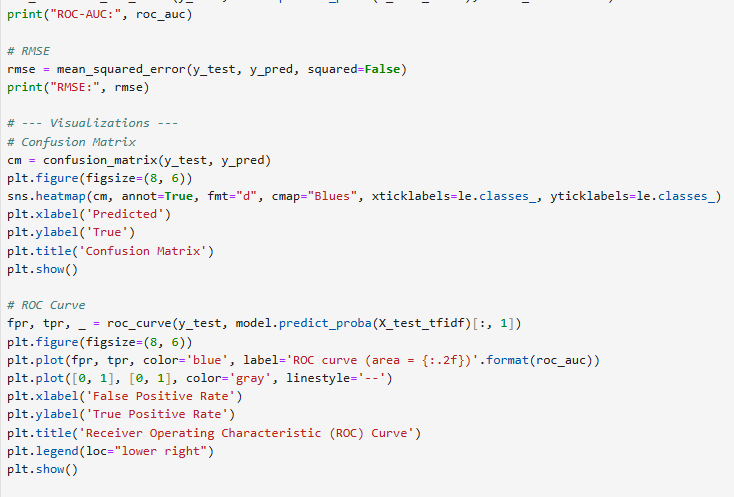
F1-score:



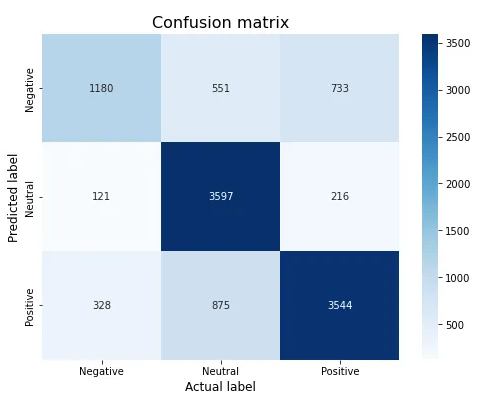
# Output:



**Confusion matrix:**



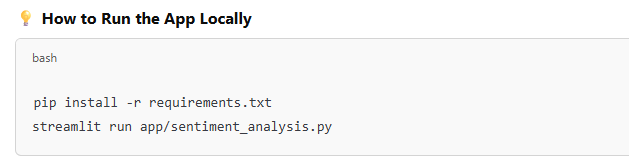
# Output:



**12. Source code**

## **All source code is upload above and github link is uploaded for further explanation and files.**

**🔗 GitHub Repository Link:** <https://github.com/dhanashekarece/NAAN-MUDHALVAN.git>

****

13. Future scope

While the current system effectively classifies sentiments and emotions from social media text, there are several promising directions for future development and improvement:

**🌍 1. Multilingual Emotion Detection**

* Extend the model to support multiple languages (e.g., Hindi, Tamil, Spanish).
* Use multilingual BERT (mBERT) or XLM-RoBERTa for cross-language emotion classification.

**⏱️ 2. Real-Time Social Media Monitoring**

* Integrate live APIs (e.g., Twitter API, Reddit Stream) for real-time emotion tracking.
* Useful for monitoring public mood during live events, elections, or crisis response.

**🎭 3. Sarcasm and Irony Detection**

* Train additional classifiers to detect sarcasm, which often misleads sentiment classifiers.
* Combine with emotion detection for more accurate interpretation of text tone.

**🧠 4. Personalized Sentiment Analysis**

* Tailor emotion models to user profiles (age, region, context) to increase relevance.
* Useful in personalized mental health applications and adaptive content platforms.

**📈 5. Enterprise Integration**

* Integrate into customer service systems, brand monitoring dashboards, or feedback analysis tools.

**Deployment**

* **Prepare Your Model**: First, ensure that your sentiment analysis model is fully trained and tested. Save the trained model using serialization techniques such as joblib or pickle to store it for later use.
* **Choose Deployment Framework**: Select a framework to expose your model as a web service. Common frameworks for deploying machine learning models are Flask (for Python), FastAPI, or Django. These frameworks will allow you to create an API endpoint that accepts input text and returns a sentiment prediction.
* **Create a Web API**: Build a simple API that accepts user input (text), processes it through your model, and returns a prediction. This will typically involve loading the saved model, transforming input data using the same preprocessing steps (e.g., TF-IDF), and feeding it into the model for prediction.
* **Test Locally**: Run the web service on your local machine to ensure the API works as expected. You can use tools like Postman or curl to send test requests to the API.
* **Save the Vectorizer and Model**: If you're using text vectorization (e.g., TF-IDF), ensure that the vectorizer is also saved and loaded when making predictions. This is necessary for transforming input text into the same format that your model expects.
* **Deploy the Web Service**: Once the local tests pass, deploy your API to a cloud service like Heroku, AWS (e.g., Elastic Beanstalk or Lambda), Google Cloud, or Azure. You can use Docker to containerize your application for easier deployment and scalability.
* **Scaling and Monitoring**: If you expect heavy traffic, ensure that the deployment platform can scale with your needs. Use monitoring tools (e.g., AWS CloudWatch, New Relic) to keep track of the service's performance and any issues that arise.
* **Maintain and Update**: After deployment, periodically retrain your model with new data and deploy updates as necessary. You can automate this retraining process if you are continuously collecting new data.

# 14.Team Members Role

**1. Lokesh S. –EXPLORATORY DATA ANALAYSIS**

**2. Vannamathi K. — DATASET DESCRIPTION AND DATA PREPROCESSING**

**3. Dhanasekaran T. — MODEL BUILDING**

**4. Venkades T. — MODEL EVALVATION**

**5. Vijayalakshmi S. — FUTURE SCOPE**