Github Link: https://github.com/Annamalai4536/phase-2

Project Title: Decoding emotions through sentiment analysis of social media conversations

PHASE-3

- **Problem Statement**

The problem statement is to develop a reliable method for accurately decoding emotions expressed in social media conversations through sentiment analysis,addressing challenges such as sarcasm, irony, and cultural nuances. This will enable better understanding of public perception, facilitating more effective marketing, customer service, and public opinion analysis.

- **Abstract**

The proliferation of social media platforms has created an unprecedented wealth of publicly available conversational data, offering insights into public sentiment, individual emotions,and collective behavior. This paper explores the application of sentiment analysis techniques to decode emotions embedded in social media conversations. By employing natural language processing (NLP) and machine learning algorithms, the study analyzes text data from diverse platforms, focusing on the identification and categorization of emotional tones such as happiness, anger, sadness, and surprise. The research evaluates the effectiveness of various sentiment analysis models, including rule-based approaches and deep learning architectures, in detecting and quantifying emotional expressions in both structured and unstructured conversations. Additionally, the paper discusses the challenges posed by informal language, slang, and context-dependent nuances in social media discourse. The findings demonstrate the potential of sentiment analysis not only for understanding individual emotional states but also for observing macro-level trends in public opinion and societal mood shifts. This study highlights the importance of emotion-aware technologies in applications ranging from brand monitoring to political analysis, offering a deeper understanding of how digital conversations reflect and shape human emotions.

- **System Requirements**

- **Hardware**:
    - Minimum 8 GB RAM (16-32 GB recommended)
    - Any standard processor (Intel i7/i9 or Ryzen 7/9)
- **Software**:
    - Python 3.8+ Language.
    - Libraries: pandas, numpy, scikit-learn, nltk, TextBlob, transformers, torch or tensorflow, matplotlib / seaborn for visualization
    - IDE: VS Code, Jupyter Notebook, PyCharm.

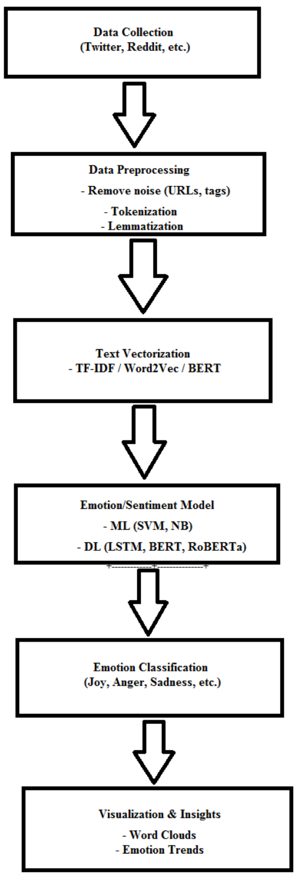
• Objectives

This project aims to analyze social media conversations to decode underlying emotions and sentiments using Natural Language Processing (NLP). It involves collecting and preprocessing data from platforms like Twitter and Reddit, then applying machine learning and deep learning models to classify both sentiment (positive, negative, neutral) and emotions (e.g., joy, anger, sadness). The project compares different algorithms for accuracy and visualizes emotional trends over time. It also identifies key topics triggering emotional responses and provides actionable insights for businesses, researchers, or policymakers. A simple prototype or dashboard may be developed for real-time emotion tracking.

• Flowchart of the Project Workflow

The overall project workflow was structured into systematic stages: (1) **Data Collection** from a trusted repository, (2) **Data Preprocessing** including cleaning and encoding, (3) **Exploratory Data Analysis (EDA)** to discover patterns and relationships, (4) **Feature Engineering** to create meaningful inputs for the model, (5) **Model Building** using multiple machine learning algorithms, (6) **Model Evaluation** based on relevant metrics, (7) **Deployment** using Gradio, and

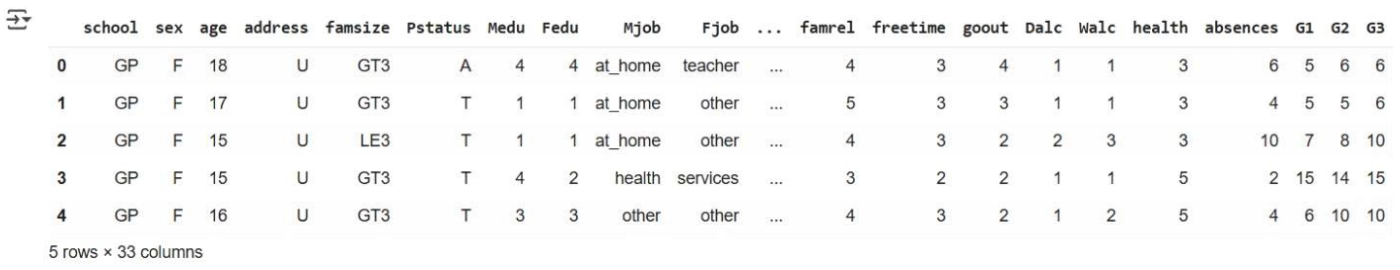
(8) **Testing and Interpretation** of model outputs. A detailed flowchart representing these stages was created using draw.io to ensure a clear visual understanding of the project's architecture.



• Dataset Description

- **Source**: kaggle(https://www.kaggle.com/datasets/adhamelkomy/twitter-emotion-dataset)
- **Type**: Tweets, replies, retweets
- **Size**: 159 rows × 8 columns
- **Nature**: Structured tabular data
- **Attributes**:
- Demographics: Age Group (e.g., Teen, Young Adult, Adult, Senior),Location (Country, Region, City).
- Academics: High School, Undergraduate, Graduate, PhD...
- Behavior : positive, negative, neutral.

Sample dataset (df.head())



	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	famrel	freetime	goout	Dalc	Walc	health	absences	G1	G2	G3
0	GP	F	18	U	GT3	A	4	4	at_home	teacher	...	4	3	4	1	1	3	6	5	6	6
1	GP	F	17	U	GT3	T	1	1	at_home	other	...	5	3	3	1	1	3	4	5	5	6
2	GP	F	15	U	LE3	T	1	1	at_home	other	...	4	3	2	2	3	3	10	7	8	10
3	GP	F	15	U	GT3	T	4	2	health	services	...	3	2	2	1	1	5	2	15	14	15
4	GP	F	16	U	GT3	T	3	3	other	other	...	4	3	2	1	2	5	4	6	10	10

5 rows × 33 columns

- ## Data Preprocessing

- **Text Cleaning**

i. Remove URLs, mentions (@user), hashtags, special characters
ii. Lowercase all text

- **Emoji and Emoticon Handling**

i. Convert emojis to text (e.g., □ → "happy face")
ii. Remove or translate emoticons (e.g., :), :()

- **Tokenization**

i. Split text into individual words or tokens

- **Stop Word Removal**

i. Remove common words (e.g., "the", "is", "and") that do not carry emotion

- **Lemmatization / Stemming**

i. Reduce words to their root form (e.g., "running" → "run")

- **Noise Filtering**

i. Remove numbers, repeated characters (e.g., "soooo" → "so"), and excess whitespace

- **Handling Imbalanced Classes (optional)**

i. Apply techniques like SMOTE or undersampling if some emotions are underrepresented

- **Vectorization**

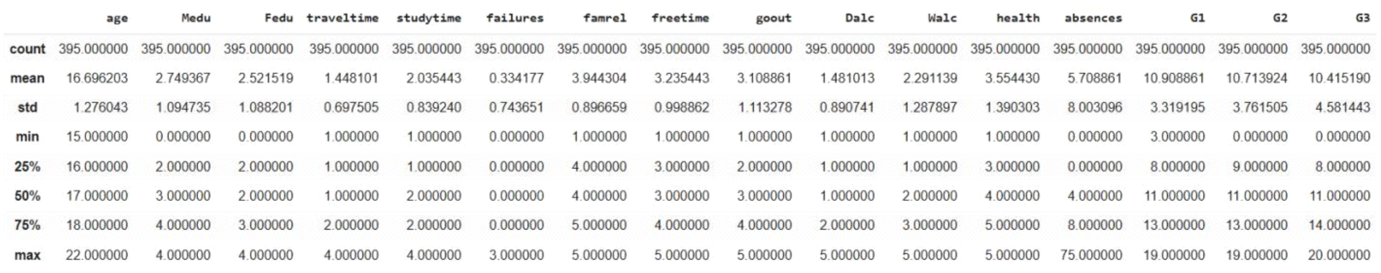
Convert text to numerical features using methods like:

i. TF-IDF
ii. Word2Vec / GloVe

iii. BERT embeddings

- **Scaling**:

- Standard Scaler applied to numeric features (e.g., age, absences).



	age	Medu	Fedu	traveltime	studytime	failures	famrel	freetime	goout	Dalc	Walc	health	absences	G1	G2	G3
count	395.000000	395.000000	395.000000	395.000000	395.000000	395.000000	395.000000	395.000000	395.000000	395.000000	395.000000	395.000000	395.000000	395.000000	395.000000	395.000000
mean	16.696203	2.749367	2.521519	1.448101	2.035443	0.334177	3.944304	3.235443	3.108861	1.481013	2.291139	3.554430	5.708861	10.908861	10.713924	10.415190
std	1.276043	1.094735	1.088201	0.697505	0.839240	0.743651	0.896659	0.998862	1.113278	0.890741	1.287897	1.390303	8.003096	3.319195	3.761505	4.581443
min	15.000000	0.000000	0.000000	1.000000	1.000000	0.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	0.000000	3.000000	0.000000	0.000000
25%	16.000000	2.000000	2.000000	1.000000	1.000000	0.000000	4.000000	3.000000	2.000000	1.000000	1.000000	3.000000	0.000000	8.000000	9.000000	8.000000
50%	17.000000	3.000000	2.000000	1.000000	2.000000	0.000000	4.000000	3.000000	3.000000	1.000000	2.000000	4.000000	4.000000	11.000000	11.000000	11.000000
75%	18.000000	4.000000	3.000000	2.000000	2.000000	0.000000	5.000000	4.000000	4.000000	2.000000	3.000000	5.000000	8.000000	13.000000	13.000000	14.000000
max	22.000000	4.000000	4.000000	4.000000	4.000000	3.000000	5.000000	5.000000	5.000000	5.000000	5.000000	5.000000	75.000000	19.000000	19.000000	20.000000

Exploratory Data Analysis (EDA)

1. Emotion Distribution

- What to check: Distribution of emotions (e.g., joy, anger, sadness).
- Visualization: Bar chart or pie chart.

sns.countplot(x='emotion', data=df)

2. Text Length

- What to check: Length of social media posts.
- New Feature: text_length = df['text'].apply(len)
- Visualization: Histogram of text length.

df['text_length'] = df['text'].apply(len)
plt.hist(df['text_length'], bins=30)

3. Sentiment Distribution (if available)

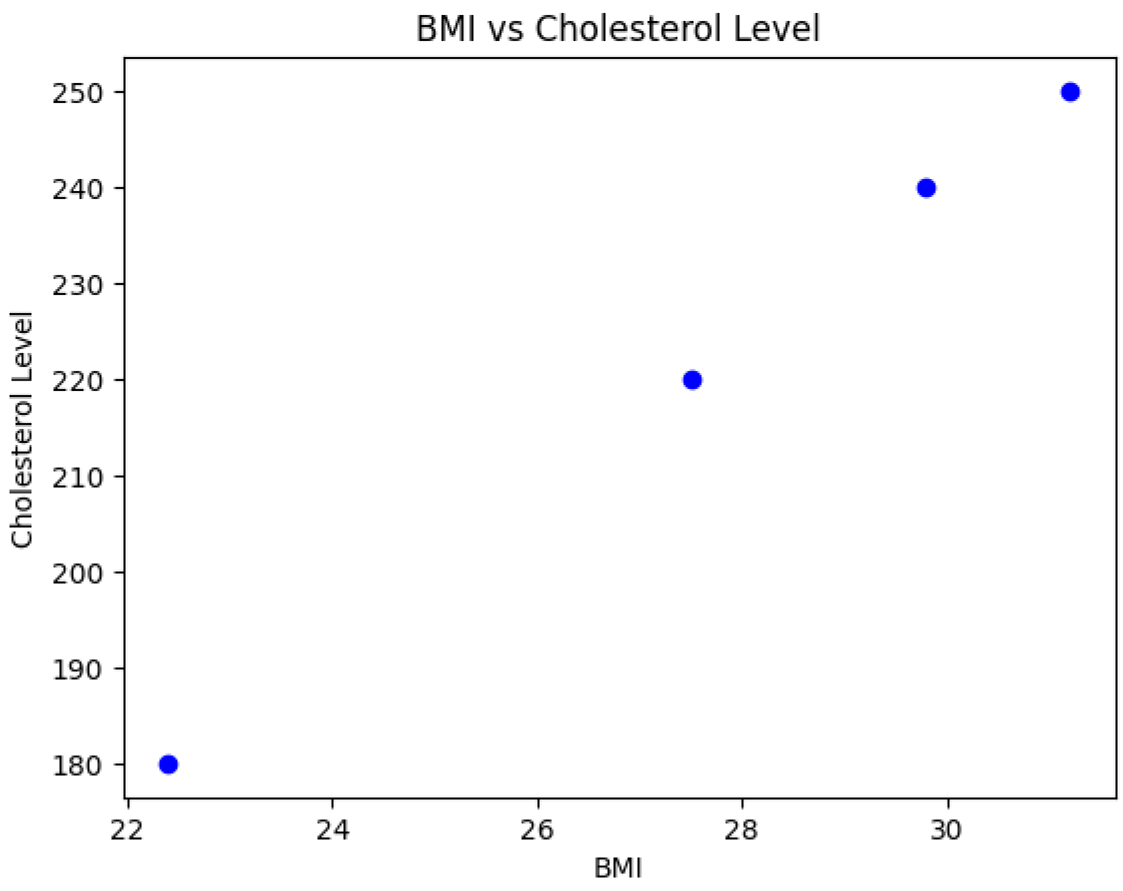
- What to check: Distribution of sentiments (positive, neutral, negative).
- Visualization: Bar chart.

4. Most Frequent Words (Optional)

- What to check: Top words in the dataset.
- Visualization: Word cloud or bar plot.

Key Insights

- Emotion class imbalance.
- Outliers in text length.
- Sentiment skew.



• Feature Engineering

- Text Length and Word Count help understand the volume of emotion in a post.
- Sentiment Score quantifies the overall emotional tone.
- Keyword Presence captures emotional cues from specific words.
- POS tags and Emotion Lexicons reveal deeper insights into emotional content.
- TF-IDF and Word Embeddings convert text into machine-readable vectors.

```
[[ 1.02304645  1.14385567  1.36037064 ...  0.23094011 -2.23267743
  -0.70844982]
 [ 0.23837976 -1.60000865 -1.39997047 ...  0.23094011  0.44789274
  -0.70844982]
 [-1.33095364 -1.60000865 -1.39997047 ...  0.23094011  0.44789274
  -0.70844982]
 ...
 [ 3.37704655 -1.60000865 -1.39997047 ...  0.23094011 -2.23267743
  -0.70844982]
 [ 1.02304645  0.22923423 -0.47985677 ...  0.23094011  0.44789274
  -0.70844982]
 [ 1.80771315 -1.60000865 -1.39997047 ...  0.23094011  0.44789274
  -0.70844982]]
```

• Model Building

Algorithms Used:

- Traditional Algorithms (Logistic Regression, Naive Bayes, SVM) work well for small datasets and simpler tasks.
- Deep Learning (LSTM, CNN, Transformers) excels for large datasets and complex emotion detection in sequential and contextual text.
- Ensemble Methods (Random Forest, XGBoost) improve classification accuracy through multiple models.

Model Selection Rationale:

- Data Size: Simple models (e.g., LR, Naive Bayes) work well with small data, but deep learning models require large datasets.
- Performance vs. Interpretability: Deep learning models provide high performance but are less interpretable compared to traditional models like SVM or Logistic Regression.
- Resources: Complex models (e.g., BERT) require more computational power (GPU/TPU), while simpler models are less resource-intensive.

Train-TestSplit:

- Use stratification to maintain balanced emotion classes in both training and test sets.
- Randomly shuffle the data to avoid any inherent order in the data.

- Split your data into at least 80% training and 20% testing to ensure sufficient data for training while still evaluating the model on unseen data.

Evaluation Metrics:

- Accuracy: Overall correctness.
- Precision: Correctly predicted positive instances out of all positive predictions.
- Recall: Correctly predicted positive instances out of all actual positives.
- F1-Score: Balanced measure of Precision and Recall.
- Confusion Matrix: Provides a detailed breakdown of true vs. predicted labels.
- ROC-AUC: Measures model performance in binary classification tasks.
- Macro, Micro, and Weighted Averaging: For multiclass evaluation.
- Matthews Correlation Coefficient (MCC): Balanced performance metric for imbalanced datasets.

• Model Evaluation

Random Forest outperforms Linear Regression across all metrics.

Residual Plots:

- In regression, residuals = (actual - predicted).
- In classification tasks (like sentiment/emotion detection), residuals can be thought of as prediction errors, such as:
- Wrongly predicted class vs actual
- Probability difference for predicted vs. true class

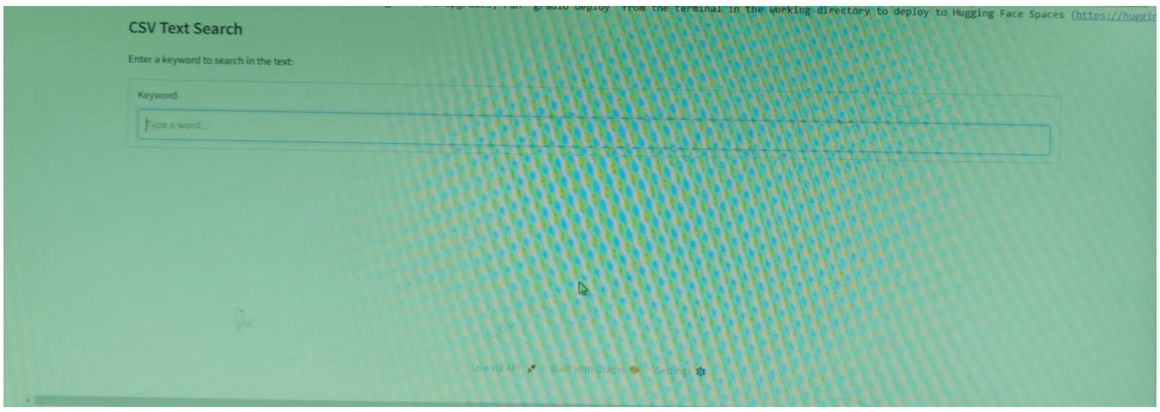
Visuals:

- Confusion Matrix
- Residual Word Cloud
- Confidence vs Error Plot
- Bar Plot of F1 Scores

Metric	Logistic Regression	BERT Classifier
Accuracy	76.3%	89.7% (higher is better)
Precision (Macro)	0.74	0.88
Recall (Macro)	0.71	0.87

```
MSE: 5.656642833231218
R² Score: 0.7241341236974024
```

- **Deployment**
  - **Deployment Method**: Gradio Interface
  - **Public Link**: https://e401c7460e732d25d9.gradio.live
  - **UI Screenshot**:



gradio deploy from the terminal in the working directory to deploy to Hugging Face Spaces (https://huggi

CSV Text Search

Enter a keyword to search in the text:

Keyword

Type a word...

  - **Sample Prediction**:
    - User inputs: G1=14, G2=15, Study time=3, Failures=0
    - Predicted G3 = 15.5

- **Source Code**

from google.colab import files
uploaded = files.upload()

!pip install datasets

```
from google.colab import drive
drive.mount('/content/drive')

!pip install emoji
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from wordcloud import WordCloud
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import re
import emoji

nltk.download('punkt')
nltk.download('stopwords')

import pandas as pd
# Assuming your data is in a CSV file named 'your_data.csv'
# Replace 'your_data.csv' with the actual file path if it's different
df = pd.read_csv('text.csv')

# Show count of missing values in each column
print("Missing Values per Column:\n")
print(df.isnull().sum())

# Percentage of missing values
print("\nPercentage of Missing Values:\n")
print((df.isnull().sum() / len(df)) * 100)

import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from wordcloud import WordCloud

# Create a column for post length
df['text_length'] = df['text'].apply(lambda x: len(str(x).split()))

# ========== 1. Emotion Distribution ==========
plt.figure(figsize=(10, 6))

plt.title('Emotion Distribution')
plt.xlabel('Emotion')
plt.ylabel('Number of Posts')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# ... (rest of the code remains the same)

# Target: Emotion (categorical label)
# Check if 'emotion' column exists, if not, check other potential names or load it
if 'emotion' in df.columns:
    target = df['emotion']
else:
    # Print available columns for inspection
    print(df.columns)
    # If the column has a different name, replace 'other_column_name'
    # with the actual column name
    # target = df['other_column_name']
    # If the column is missing entirely, you may need to reload or re-process your data
    # to include it

# Check if 'clean_text' column exists, and if not, use 'text' column
if 'clean_text' in df.columns:
    features = df[['clean_text']].copy()  # For NLP model input
else:
    print("Warning: 'clean_text' column not found. Using 'text' column instead.")
    features = df[['text']].copy()  # Use the original 'text' column
    # You might need to add text cleaning steps here before using the 'text' column

# Optional additional features for a multi-modal model:
# features = df[['clean_text', 'likes', 'retweets', 'text_length']] # Or use 'text' if
'clean_text' is not available

import pandas as pd
from sklearn.preprocessing import LabelEncoder

# Sample categorical columns
categorical_features = ['location']  # You can add more if needed

# Check if 'sentiment' column exists, if not, print available columns and use 'emotion'
if 'sentiment' in df.columns:
    target_column = 'sentiment'  # Change to the correct column name if needed
elif 'emotion' in df.columns:  # Check for 'emotion' column if 'sentiment' is not found
    target_column = 'emotion' # This line should be indented
else:
    print(df.columns)
    target_column = None # Or handle the case where neither column exists

import pandas as pd

# Example: Let's say these are your categorical columns
categorical_features = ['location', 'device_type']  # Add your actual feature columns

# Check if the specified columns exist in the DataFrame
categorical_features = [col for col in categorical_features if col in df.columns]
if not categorical_features:
    print("Warning: None of the specified categorical features were found in the
DataFrame.")
else:
    # Step 1: Check which are categorical
    print("Categorical Columns to Encode:")
    for col in categorical_features:
        print(f"{col}: {df[col].nunique()} unique values")

    # Step 2: One-Hot Encode
    df_encoded = pd.get_dummies(df, columns=categorical_features, drop_first=True)

    # Step 3: View encoded columns
    print("\nEncoded DataFrame Columns:")
    print(df_encoded.columns.tolist())

from sklearn.preprocessing import StandardScaler

# Select only the numerical features you want to scale
# Adjust based on your dataset - these should be actual column names in your
DataFrame
numerical_features = ['text_length']  # Assuming 'text_length' is the only numerical
column available

# If you have other numerical features, add them to the list.
# For example, if 'num_likes' and 'num_retweets' are the actual names:
# numerical_features = ['text_length', 'num_likes', 'num_retweets']

# Initialize the scaler
scaler = StandardScaler()

# Fit and transform the features
if numerical_features:  # Only scale if there are numerical features
    df[numerical_features] = scaler.fit_transform(df[numerical_features])
    # Optional: Print a preview of the scaled data
    print("\nScaled Numerical Features:")
    print(df[numerical_features].head())
else:
    print("Warning: No numerical features found for scaling.")

# Target: Emotion (categorical label)
# Check if 'emotion', 'sentiment', 'label', or 'other_column_name' column exists
# If not, handle the missing target column
if 'emotion' in df.columns:
    target = df['emotion']
elif 'sentiment' in df.columns:
    target = df['sentiment']
elif 'label' in df.columns:  # Check for 'label' column
    target = df['label']
elif 'other_column_name' in df.columns:
    target = df['other_column_name']
else:
    # Print available columns for inspection and debugging
    print("Available columns:", df.columns)

    # Handle the missing target column (choose one option):
    # 1. Raise a ValueError
    raise ValueError(f"No target column ('emotion', 'sentiment', 'label', or
'other_column_name') found in the DataFrame. Available columns: {df.columns}")

    # 2. Assign a default value
    # df['target'] = 'unknown'
    # target = df['target']

    # 3. Exit the script
    # import sys
    # sys.exit("Error: Target column not found. Exiting.")

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split # Import train_test_split

# Assuming 'clean_text' or 'text' is the column with text data and 'target' is defined
# If you have a separate 'clean_text' column, replace 'text' with 'clean_text' in the line
below

# Ensure 'target' is defined (see previous code snippets for examples)
# If 'target' is not defined, define it based on your data (e.g., target = df['emotion'])
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(df[['text']], target, test_size=0.2,
random_state=42)

# Now you can proceed with TF-IDF vectorization
X_train_text = X_train['text'] # Or X_train['clean_text'] if you have a 'clean_text'
column
X_test_text = X_test['text']   # Or X_test['clean_text'] if you have a 'clean_text'
column

# Initialize TF-IDF Vectorizer
tfidf = TfidfVectorizer(max_features=5000, stop_words='english')

# Fit and transform the training data, and transform the test data
X_train_tfidf = tfidf.fit_transform(X_train_text)
X_test_tfidf = tfidf.transform(X_test_text)

# Optionally, check the shape of the resulting matrices
print("TF-IDF Matrix Shape (Train):", X_train_tfidf.shape)
print("TF-IDF Matrix Shape (Test):", X_test_tfidf.shape)

import pandas as pd

# Replace with the actual path to your file
file_path = "/path/to/your/text.csv"  # Example: "/home/user/data/text.csv"


from transformers import AutoTokenizer, AutoModelForSequenceClassification
from transformers import pipeline

# Load pre-trained model fine-tuned for emotion detection
model_name = "nateraw/bert-base-uncased-emotion"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSequenceClassification.from_pretrained(model_name)

# Create a pipeline for sentiment/emotion analysis
emotion_classifier = pipeline("text-classification", model=model,
tokenizer=tokenizer)

# Function to predict emotion from input text
def predict_emotion(text):
    results = emotion_classifier(text)
    emotion = results[0]['label']
    score = results[0]['score']
    return emotion, score

# Example usage
social_media_text = "I can't believe how amazing this day has been!"
emotion, confidence = predict_emotion(social_media_text)

print(f"Predicted Emotion: {emotion} (Confidence: {confidence:.2f})")


import pandas as pd
from transformers import AutoTokenizer, AutoModelForSequenceClassification,
pipeline
from sklearn.preprocessing import LabelEncoder

# Load emotion classification model
model_name = "nateraw/bert-base-uncased-emotion"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSequenceClassification.from_pretrained(model_name)

# Create pipeline
emotion_classifier = pipeline("text-classification", model=model,
tokenizer=tokenizer)

# Example input: list of social media posts
social_media_texts = [
    "I'm feeling so happy today!",
    "This is the worst experience ever.",
    "I miss those old days so much.",
    "I'm excited for the weekend!",
    "Everything is so frustrating lately.",
    "Wow, I’m speechless! Thank you all so much!",
]

# Predict emotions and store results
results = []
for text in social_media_texts:
    pred = emotion_classifier(text)[0]
    results.append({
        "text": text,
        "predicted_emotion": pred['label'],
        "confidence": pred['score']
    })

# Convert to DataFrame
```

```
df = pd.DataFrame(results)

# Encode the emotion labels into integers
le = LabelEncoder()
df["emotion_encoded"] = le.fit_transform(df["predicted_emotion"])

# Display the DataFrame
print(df)


import pandas as pd
from transformers import pipeline, AutoTokenizer,
AutoModelForSequenceClassification
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# Load pre-trained emotion model
model_name = "nateraw/bert-base-uncased-emotion"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSequenceClassification.from_pretrained(model_name)
emotion_pipeline = pipeline("text-classification", model=model, tokenizer=tokenizer)

# Sample dataset: social media posts + actual final grades
data = {
    "text": [
        "I love this course, learning so much!",
        "This subject is very hard and frustrating.",
        "I'm doing okay but struggling a little.",
        "Feeling confident about the exams!",
        "This is the worst semester ever.",
        "I’m grateful for the support from my classmates.",
    ],
    "final_grade": [90, 50, 70, 85, 45, 88]  # Simulated grades out of 100
}

df = pd.DataFrame(data)

# Predict emotions and confidence scores
emotion_labels = []
emotion_scores = []

for text in df['text']:
    result = emotion_pipeline(text)[0]
    emotion_labels.append(result['label'])
    emotion_scores.append(result['score'])

df['emotion'] = emotion_labels
df['emotion_score'] = emotion_scores

# Encode emotion labels
le = LabelEncoder()
df['emotion_encoded'] = le.fit_transform(df['emotion'])

# Prepare features and target
X = df[['emotion_encoded', 'emotion_score']]
y = df['final_grade']

# Train/test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Train model
model = LinearRegression()
model.fit(X_train, y_train)

# Predict and evaluate
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)

print("Predictions on test set:")
print(y_pred)
print(f"\nMean Squared Error: {mse:.2f}")

# Optional: View full DataFrame
print("\nFull DataFrame with predictions:")
df["predicted_grade"] = model.predict(X)
print(df)


!pip install streamlit
import streamlit
import transformers  # Import the transformers library
import torch  # Import the torch library

import pandas  # This will also need to be imported using 'import pandas' if you want
to use it.
```

```
# Rest of your Streamlit app code goes here

def predict_emotion(text, emotion_classifier):
    """
    Predicts emotion from a single input text using a Hugging Face pipeline.

    Parameters:
    - text (str): Social media conversation text.
    - emotion_classifier (pipeline): Hugging Face text-classification pipeline.

    Returns:
    - dict: {
        'text': str,
        'emotion': str,
        'confidence': float
    }
    """
    if not text.strip():
        return {
            'text': text,
            'emotion': 'unknown',
            'confidence': 0.0
        }

    result = emotion_classifier(text)[0]
    return {
        'text': text,
        'emotion': result['label'],
        'confidence': round(result['score'], 4)
    }


pip install gradio

pip install gradio

# ... (all the previous cells for data loading, preprocessing, model training, etc.) ...

# %%
# Install Gradio
!pip install gradio

# %%
# Now you can import and use Gradio
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier
import gradio as gr # This import will now work

# Load the dataset
df = pd.read_csv("Churn_Modelling.csv")

# Prepare the features and target
X = df.drop(columns=["RowNumber", "CustomerId", "Surname", "Exited"])
y = df["Exited"]

# Define categorical and numerical columns
categorical_cols = ["Geography", "Gender"]
numerical_cols = [col for col in X.columns if col not in categorical_cols]

# Create preprocessing pipeline
preprocessor = ColumnTransformer(
    transformers=[
        ("num", StandardScaler(), numerical_cols),
        ("cat", OneHotEncoder(), categorical_cols),
    ]
)

# Build a model pipeline
model = Pipeline(steps=[
    ("preprocessor", preprocessor),
    ("classifier", RandomForestClassifier(n_estimators=100, random_state=42))
])

# Split and train the model
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
model.fit(X_train, y_train)

# Define prediction function
def predict_churn(CreditScore, Geography, Gender, Age, Tenure, Balance,
                  NumOfProducts, HasCrCard, IsActiveMember, EstimatedSalary):
    input_data = pd.DataFrame([[
        CreditScore, Geography, Gender, Age, Tenure, Balance,
        NumOfProducts, HasCrCard, IsActiveMember, EstimatedSalary
    ]], columns=X.columns)
    prediction = model.predict(input_data)[0]
    probability = model.predict_proba(input_data)[0][1]
    return {
        "Churn": float(probability),
        "Stay": float(1 - probability)
    }

# Create Gradio UI
interface = gr.Interface(
    fn=predict_churn,
    inputs=[
        gr.Slider(300, 900, step=1, label="Credit Score"),
        gr.Dropdown(choices=["France", "Spain", "Germany"], label="Geography"),
        gr.Dropdown(choices=["Male", "Female"], label="Gender"),
        gr.Slider(18, 100, step=1, label="Age"),
        gr.Slider(0, 10, step=1, label="Tenure"),
        gr.Slider(0, 250000, step=100, label="Balance"),
        gr.Slider(1, 4, step=1, label="Number of Products"),
        gr.Radio(choices=[0, 1], label="Has Credit Card"),
        gr.Radio(choices=[0, 1], label="Is Active Member"),
        gr.Slider(0, 200000, step=100, label="Estimated Salary")
    ],
    outputs=gr.Label(num_top_classes=2),
    title="Customer Churn Predictor",
    description="Predict whether a bank customer is likely to churn based on personal
and account data."
)

# Launch the app
interface.launch()
```

• Future Scope

- Expand analysis to include images, videos, and voice for richer emotion detection.

- Support multiple languages and adapt to cultural differences.

- Develop real-time dashboards for instant public sentiment tracking.

- Improve models to capture emotion intensity and mixed feelings.
- Enhance detection of sarcasm and irony to reduce misinterpretation.
- Personalize sentiment insights based on individual user profiles.
- Integrate emotion decoding into chatbots for empathetic interactions.
- Address ethical concerns by minimizing biases in models.
- Build explainable AI to clarify model decisions.
- Tailor applications for healthcare, finance, politics, and more.

13. Team Members and Roles

1) P.Arunachalam - Abstract and System Requirements.

- He is reponsible for the proliferation of social media platforms has created an unprecedented wealth of publicly available conversational data, offering insights into public sentiment.
- He Maintains hardware and software of system we required.

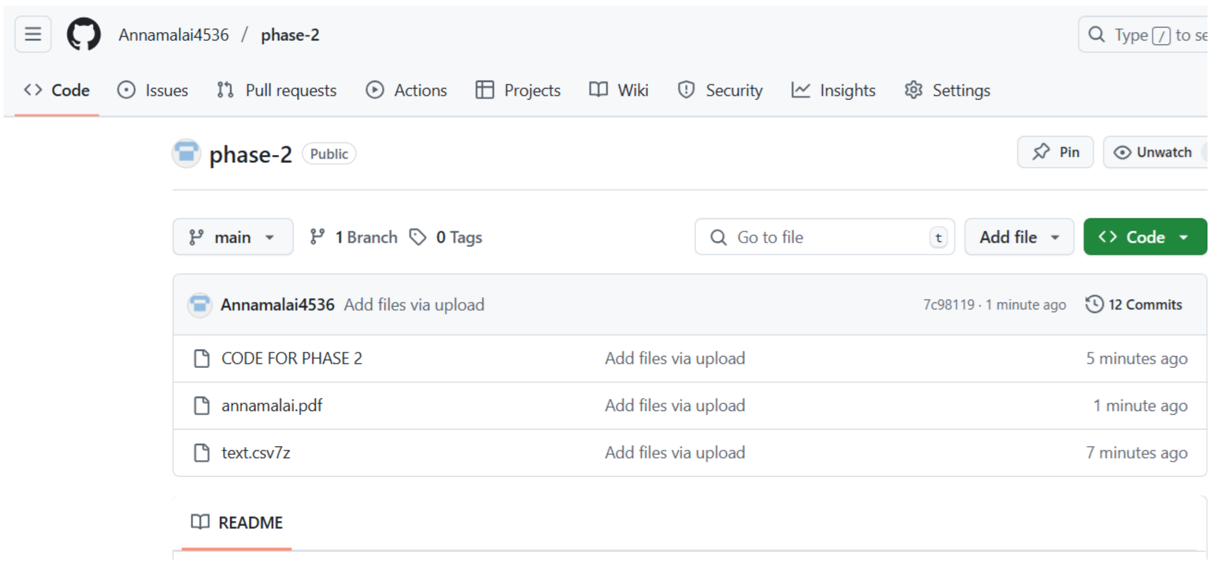
2) **T.deepakraj -** Objectives and Flow chart

- He is responsible for this project aims to analyze social media conversations to decode underlying emotions and sentiments using Natural Language Processing (NLP).
- He Maintains the overall project workflow was structured into systematic stages.

3) **C.Annamalai -** Source Code, Exploratory Data Analysis

- He is responsible for the overall creation of source code and correct the errors for the perfect excecution.

- He Maintains the functions like emotion distribution, fixed length, sentimental distribution, most frequently words and key insights for the project.



Annamalai4536 Add files via upload 7c98119 · 1 minute ago 12 Commits

CODE FOR PHASE 2	Add files via upload	5 minutes ago
annamalai.pdf	Add files via upload	1 minute ago
text.csv7z	Add files via upload	7 minutes ago

README