Phase-3 Submission

Student Name: Yashwanth KS

Register Number: AU410723104098

Institution: Dhanalakshmi College of Engineering

Department: computer science engineering

Date of Submission: 16-May-25

GitHub Repository link: https://github.com/KsYashwanth1109/Nm-yashwanth

---

Project Title: Decoding Emotions Through Sentiment Analysis of Social Media Conversation

1. Problem Statement

With the rise of social media, people express their thoughts and emotions online in real-time. Understanding these emotions is crucial for various domains like marketing, politics, and public health. However, analyzing emotional content is challenging due to slang, sarcasm, abbreviations, and noisy data.

2. Abstract

This project aims to detect and classify emotions from social media posts using Natural Language Processing (NLP) techniques. By preprocessing text data and applying machine learning and deep learning models, we can interpret users' sentiments and visualize emotional trends. The system helps in real-time emotion monitoring and public opinion analysis.

3. System Requirements

Hardware: *8GB RAM, i5 Processor or higher*

Software: *Python 3.8+, Jupyter/Colab, Streamlit*

Libraries: *NLTK, Scikit-learn, Transformers, TextBlob, Matplotlib*

Dataset: *Social media datasets (e.g., Twitter, GoEmotions)*

4. Data Description

Dataset Source: *Data is collected from the Twitter API, Reddit threads, or publicly available sentiment datasets on platforms like Kaggle.*

Data Type: *Unstructured text data containing user-generated content.*

Features: *Tweet/comment text, timestamps, user metadata (optional), sentiment labels.*

Target Variable*: Sentiment classification—commonly Positive, Negative, and Neutral.*

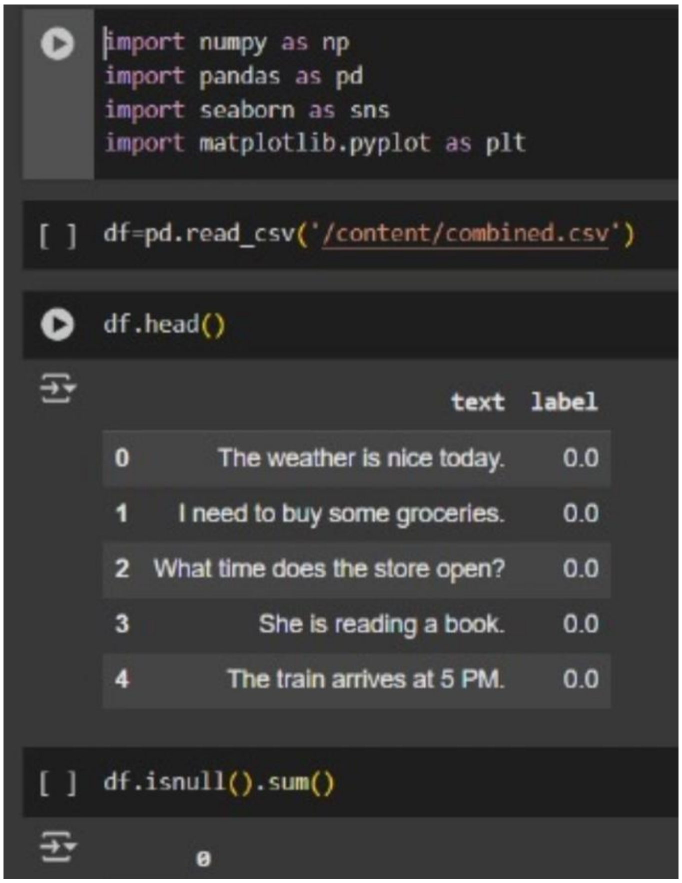
Nature of Data: *Can be static (archived datasets) or dynamic (real-time data from social APIs).*

Program:

```
import numpy as np
import pandas as pd
import seaborn as sns
```

import matplotlib.pyplot as plt
df=pd.read_csv('/content/combined.csv')
df.head()

output:



```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

```
[ ] df=pd.read_csv('/content/combined.csv')
```

```
df.head()
```

	text	label
0	The weather is nice today.	0.0
1	I need to buy some groceries.	0.0
2	What time does the store open?	0.0
3	She is reading a book.	0.0
4	The train arrives at 5 PM.	0.0

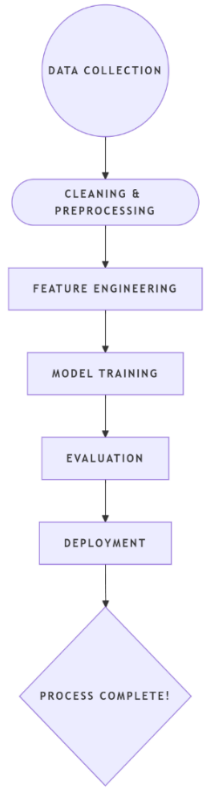
```
[ ] df.isnull().sum()
```

0

5. Objectives

- ***Extract emotional signals from text***
- ***Classify text into emotions like joy, anger, sadness, etc.***
- ***Develop a user-friendly emotion prediction app***
- ***Visualize public sentiment and emotion trends***

6. Flowchart:



7.Data processing:

1. ***Data Collection:***
   Collect customer queries, chat logs, or support tickets

2. ***Data Cleaning:***
   Remove noise (e.g., greetings, timestamps)
   Fix spelling errors and remove unnecessary characters.
   Convert text to lowercase and remove stop words.

   Program:

```
import numpy as np
import pandas as pd
```

```
import seaborn as sns
import matplotlib.pyplot as plt
df=pd.read_csv('/content/combined.csv')
df.head()
df.isnull().sum()
df.info()
df.drop_duplicates(inplace=True)
df
df.duplicated().sum()
df["label"].fillna(df["label"].mean(),inplace=True)
df.isnull().sum()
```

output:

```
df.isnull().sum()

        0
text    0
label   1

dtype: int64

[ ] df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15013 entries, 0 to 15012
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   text    15013 non-null  object
 1   label   15012 non-null  float64
dtypes: float64(1), object(1)
memory usage: 234.7+ KB

[ ] df.drop_duplicates(inplace=True)

[ ] df
```

```
df.drop_duplicates(inplace=True)

[ ] df
```

5003	Believe in yourself, you are capable of great ...	1.0
5004	You bring joy to those around you.	1.0
5005	Your hard work and dedication inspire others.	1.0
5006	Never stop chasing your dreams.	1.0
5007	You have a wonderful heart.	1.0
5008	Your positivity is contagious.	1.0
5009	Keep smiling, it brightens up everyone's day.	1.0
10000	I will hunt you down and make you suffer.	2.0
10001	You are the worst thing that ever happened to ...	2.0
10002	People like you deserve to be erased from exis...	2.0
10003	I will destroy everything you love.	2.0
10004	You should just disappear from this planet.	2.0
10005	You are a worthless piece of trash.	2.0

3. ***Tokenization:***
   Split text into individual words or sentences.

4. ***Normalization:***

Apply stemming or lemmatization (e.g., “running”-→ “run”.)

5. ***Entity Recognition:***
   Extract important entities such as order numbers, dates, ect.

8. Dataset Description

Source: Twitter, Kaggle (e.g., Sentiment140, GoEmotions*)*

Size: Thousands of labeled text samples

Labels: Emotions (Happy, Sad, Angry, etc.)

Format: CSV or JSON with text and emotion label

9. Exploratory Data Analysis (EDA)

- *Analyze label distribution*
- *Generate word clouds for each emotion*
- *Plot keyword frequency and sentiment trend over time*

10. Feature Engineering

- *Use TF-IDF, Bag of Words, or Word Embeddings*
- *Extract syntactic and semantic features*
- *Use pre-trained models like BERT for context-aware embeddings*

11. Model Building

- *Test models like Naive Bayes, Logistic Regression, LSTM, and BERT*
- *Use emotion classification or multi-label classification*
- *Fine-tune models for better accuracy*

Program :#model building

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
#knn clustering
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report
vectorizer = TfidfVectorizer()
X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)
#knn cluster
model = KNeighborsClassifier(n_neighbors=5)
model.fit(X_train_tfidf, y_train)
y_pred = model.predict(X_test_tfidf)
print( "y_pred",y_pred)
```

output:

```
[ ] #model building
    from sklearn.feature_extraction.text import TfidfVectorizer
    from sklearn.linear_model import LogisticRegression
    from sklearn.metrics import accuracy_score, classification_report

[ ] #knn clustering
    from sklearn.neighbors import KNeighborsClassifier
    from sklearn.metrics import accuracy_score, classification_report
    vectorizer = TfidfVectorizer()
    X_train_tfidf = vectorizer.fit_transform(X_train)
    X_test_tfidf = vectorizer.transform(X_test)

[ ] #knn cluster
    model = KNeighborsClassifier(n_neighbors=5)
    model.fit(X_train_tfidf, y_train)
    y_pred = model.predict(X_test_tfidf)
    print( "y_pred",y_pred)

    y_pred [-1  1 -1 -1  1 -1 -1 -1 -1 -1 -1]
```

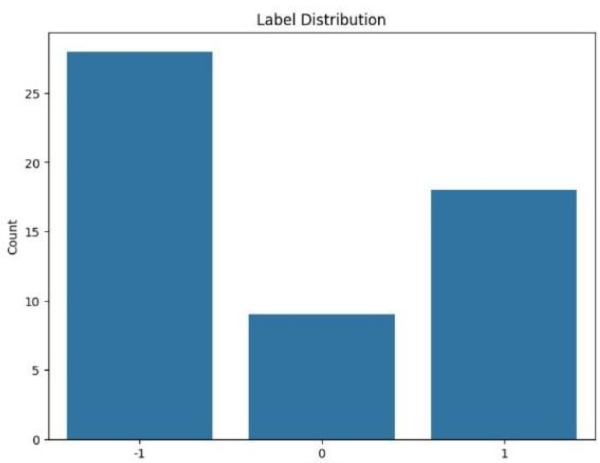
12. Model Evaluation

- *Use Accuracy, Precision, Recall, and F1-score*
- *Confusion matrix and classification reports*
- *Compare model performance and select the best one*

Program :#chart

```
plt.figure(figsize=(8, 6))
sns.countplot(x='label', data=df)
plt.title('Label Distribution')
plt.xlabel('Label')
plt.ylabel('Count')
plt.show()
```

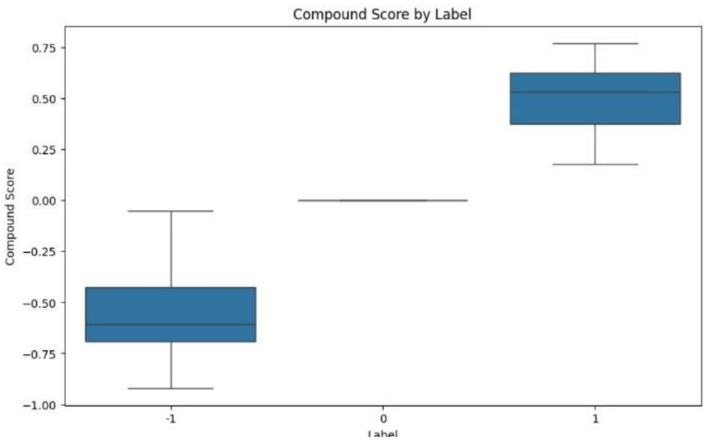
output:



Program: #bivariate analysis

```
plt.figure(figsize=(10, 6))
sns.boxplot(x='label', y='compound', data=df)
plt.title('Compound Score by Label')
plt.xlabel('Label')
plt.ylabel('Compound Score')
plt.show()
```

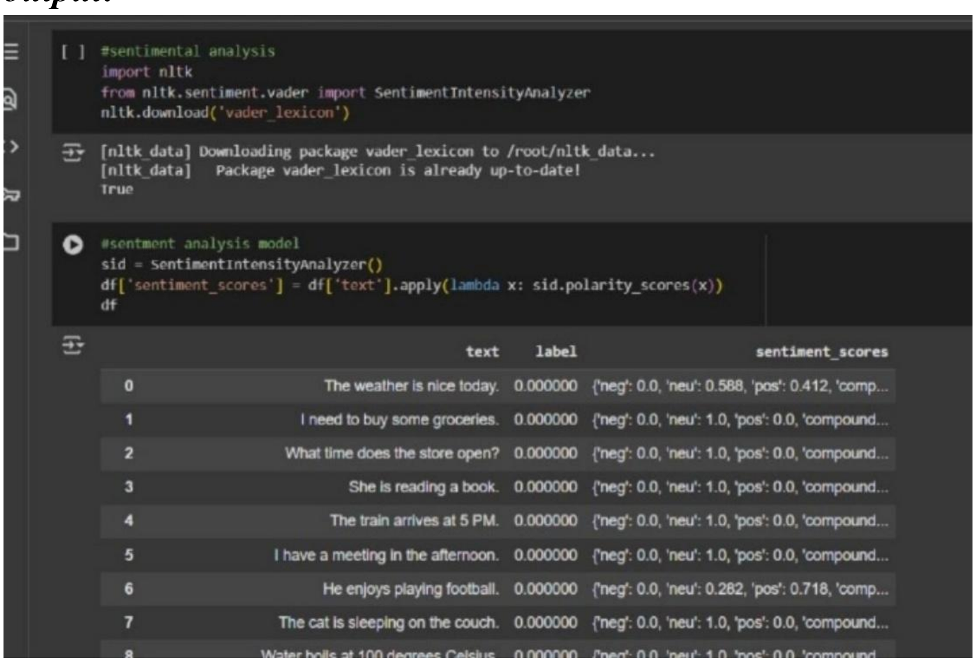
output:



Program: #import sentiment analysis model

```
from nltk.sentiment.vader import SentimentIntensityAnalyzer
sid = SentimentIntensityAnalyzer()
df['sentiment_scores'] = df['text'].apply(lambda x: sid.polarity_scores(x))
df
```

output:



```python
# Accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")

# Classification Report
print(classification_report(y_test, y_pred))

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=['Negative', 'Neutral', 'Positive'],
            yticklabels=['Negative', 'Neutral', 'Positive']) # Assuming labels are -1, 0, 1
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

```
ommands   + Code   + Text   Copy to Drive                                   Connect

[ ] model = GradientBoostingClassifier()
    model.fit(X_train, y_train)

    # Predictions and Evaluation
    y_pred = model.predict(X_test)
    print("Accuracy:", accuracy_score(y_test, y_pred))
    print("\nClassification Report:\n", classification_report(y_test, y_pred))

    Accuracy: 0.6363636363636364

    Classification Report:
                  precision    recall  f1-score   support

             0.0       0.50      1.00      0.67         2
             1.0       0.00      0.00      0.00         4
             2.0       0.71      1.00      0.83         5

        accuracy                           0.64        11
       macro avg       0.40      0.67      0.50        11
    weighted avg       0.42      0.64      0.50        11

    /usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted sampl
      _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
    /usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted sampl
      _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
    /usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted sampl
      _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))

[ ] from sklearn.metrics import classification_report, accuracy_score
```

13. Deployment

- *Build an interactive app using Streamlit*
- *Users can input social media text to get emotion predictions*
- *Display emotion probability scores and visual feedback*

14. Source Code

- *Organized and commented Python scripts*

- *GitHub repository for collaboration and version tracking*

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
df=pd.read_csv('/content/combined.csv')
df.head()
df.isnull().sum()
df.info()
df.drop_duplicates(inplace=True)
df
df.duplicated().sum()
df["label"].fillna(df["label"].mean(),inplace=True)
df.isnull().sum()
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
df_scaled=df.copy()
df_scaled[["label"]]=scaler.fit_transform(df[["label"]])
df_scaled
#minmax scaler
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler()
df_scaled=df.copy()
df_scaled[["label"]]=scaler.fit_transform(df[["label"]])
df_scaled
#sentimental analysis
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
nltk.download('vader_lexicon')
#sentment analysis model
sid = SentimentIntensityAnalyzer()
df['sentiment_scores'] = df['text'].apply(lambda x: sid.polarity_scores(x))
df
```

```
#import sentiment analysis model
from nltk.sentiment.vader import SentimentIntensityAnalyzer
sid = SentimentIntensityAnalyzer()
df['sentiment_scores'] = df['text'].apply(lambda x: sid.polarity_scores(x))
df
#Sentimental analysis split the data set
df['compound'] = df['sentiment_scores'].apply(lambda x: x['compound'])
df
#split the dataset
df['sentiment_label'] = df['compound'].apply(lambda x: 'positive' if x >= 0.05 else ('negative' if x <= -0.05 else 'neutral'))
df
#split the dataset
df['sentiment_label'] = df['compound'].apply(lambda x: 'positive' if x >= 0.05 else ('negative' if x <= -0.05 else 'neutral'))
df
#target variable
df['label'] = df['sentiment_label'].apply(lambda x: 1 if x == 'positive' else (0 if x == 'neutral' else -1))
df
#univariate analysis
df['label'].value_counts()
#chart
plt.figure(figsize=(8, 6))
sns.countplot(x='label', data=df)
plt.title('Label Distribution')
plt.xlabel('Label')
plt.ylabel('Count')
plt.show()
#bivariate analysis
plt.figure(figsize=(10, 6))
sns.boxplot(x='label', y='compound', data=df)
plt.title('Compound Score by Label')
plt.xlabel('Label')
```

```
plt.ylabel('Compound Score')
plt.show()
#split the train and test data
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df['text'], df['label'],
    test_size=0.2, random_state=42)
x=df['text']
y=df['label']
x
y
#model building
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
#knn clustering
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report
vectorizer = TfidfVectorizer()
X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)
#knn cluster
model = KNeighborsClassifier(n_neighbors=5)
model.fit(X_train_tfidf, y_train)
y_pred = model.predict(X_test_tfidf)
print( "y_pred",y_pred)
from sklearn.cluster import KMeans
# Assuming 'X_train_tfidf' is your preprocessed data (as in your previous code)
kmeans = KMeans(n_clusters=3, random_state=0) # Choose an appropriate
    number of clusters
kmeans.fit(X_train_tfidf)
cluster_labels = kmeans.labels_
# You can now analyze the clusters
# For example, print the cluster labels for each data point in X_train
print(cluster_labels)
```

```
# Or, analyze the cluster centers
kmeans.cluster_centers_
from sklearn.metrics import silhouette_score
# Calculate Silhouette Score
silhouette_avg = silhouette_score(X_train_tfidf, cluster_labels)
print(f"Silhouette Score: {silhouette_avg}")
# Evaluate KNN Classification
from sklearn.metrics import accuracy_score, precision_score, recall_score,
    f1_score, confusion_matrix
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
precision = precision_score(y_test, y_pred, average='weighted') # Use 'weighted'
for multi-class
print(f"Precision: {precision}")
recall = recall_score(y_test, y_pred, average='weighted')
print(f"Recall: {recall}")
f1 = f1_score(y_test, y_pred, average='weighted')
print(f"F1-score: {f1}")
conf_matrix = confusion_matrix(y_test, y_pred)
print(f"Confusion Matrix:\n{conf_matrix}")
print(classification_report(y_test, y_pred))
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score
import pandas as pd
# Load dataset
df = pd.read_csv("/content/combined.csv")
# Drop missing and duplicate values
df = df.dropna(subset=["label"])
df = df.drop_duplicates(subset=["text", "label"]).reset_index(drop=True)
# TF-IDF Vectorization
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(df["text"])
```

```
# Labels (no need to scale for classification)
y = df["label"]
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
    random_state=42)
# Gradient Boosting Classifier
model = GradientBoostingClassifier()
model.fit(X_train, y_train)
# Predictions and Evaluation
y_pred = model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
from sklearn.metrics import classification_report, accuracy_score
# After predictions
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
import numpy as np
# Get feature importance from the model
importances = model.feature_importances_
# Match them to feature names from TF-IDF
feature_names = vectorizer.get_feature_names_out()
important_features = sorted(zip(importances, feature_names), reverse=True)

# Show top 10 important words
print("Top 10 important features:")
for score, name in important_features[:10]:
   print(f"{name}: {score:.4f}")
import matplotlib.pyplot as plt
# Plot top 10 important features
top_features = important_features[:10]
names = [name for _, name in top_features]
scores = [score for score, _ in top_features]
plt.figure(figsize=(10, 6))
plt.barh(names[::-1], scores[::-1])
```

```
plt.xlabel("Feature Importance")
plt.title("Top 10 Important Words for Gradient Boosting")
plt.show()
import matplotlib.pyplot as plt
# Assuming y_test and y_pred are already defined from your model's predictions
plt.figure(figsize=(10, 6))
plt.plot(y_test.values, label='Actual')
plt.plot(y_pred, label='Predicted')
plt.xlabel('Data Point')
plt.ylabel('Label')
plt.title('Actual vs. Predicted Labels')
plt.legend()
plt.show()
# Accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
# Classification Report
print(classification_report(y_test, y_pred))
# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=['Negative', 'Neutral', 'Positive'],
 yticklabels=['Negative', 'Neutral', 'Positive'])
# Assuming labels are -1, 0, 1
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

15. Future Scope

- *Real-time emotion tracking via social media APIs*
- *Multilingual sentiment/emotion detection*
- *Detect sarcasm, irony, and mixed emotions*
- *Dashboard integration for visualization*

16. Team Members and roles

Name	***Role***	***Responsibilities***
Umesh	*Preprocessing & EDA*	*Clean and explore the data*
Srikanth	*Deployment*	*Build and test the web application*
Yashwanth	*Data Collection*	*Research and gather relevant datasets*
Ravikumaar	*Modeling*	*Train and fine-tune models*