FINAL PROJECT REPORT TEMPLATE

Index **Page No.**

1. INTRODUCTION

1.1. Project overviews

"Unlocking the Minds: Analyzing Mental Health with NLP" leverages advanced Natural Language Processing (NLP) techniques to analyze textual data related to mental health. The project explores linguistic patterns, sentiment trends, and emotional expressions to gain deeper insights into the complexities of mental health conditions. It focuses on three core scenarios:

1. **Social Media Sentiment Analysis**: Analyzing mental health-related discussions on social platforms to identify trends, challenges, and support networks.
2. **Therapist-Patient Communication Analysis**: Assisting therapists by analyzing therapy session transcripts to inform personalized treatment plans and enhance therapeutic outcomes.
3. **Academic Research Paper Analysis**: Extracting key insights from scholarly articles to identify knowledge gaps, trends, and emerging areas in mental health research.

By integrating these scenarios, the project aims to bridge the gap between NLP advancements and practical mental health applications, contributing to research, advocacy, and clinical support.

1.2 Objectives

The main objectives of this project are:

- **To apply NLP techniques** to analyze linguistic patterns, sentiments, and themes in mental health-related text.
- **To support mental health advocacy and intervention efforts** by identifying trends and challenges through social media analysis.
- **To enhance clinical practice** by providing insights from therapy session analyses, enabling more personalized and effective treatments.
- **To advance mental health research** by analyzing academic literature, highlighting knowledge gaps, and informing future research directions.
- **To reduce stigma surrounding mental health** by fostering awareness and understanding through data-driven insights.

These objectives collectively aim to facilitate better research, support, and intervention strategies in the domain of mental health.

2. PROJECT INITIALIZATION AND PLANNING PHASE

2.1 Define Problem Statement

Mental health remains a critical yet under-researched aspect of overall well-being, often surrounded by stigma and a lack of understanding. Despite the abundance of textual data related to mental health—ranging from social media discussions to therapy session transcripts and academic research—this wealth of information remains largely untapped due to the complexity of analyzing unstructured data. Current methods for understanding mental health trends, personalizing treatment, and advancing research are limited by scalability and depth. There is a pressing need for innovative solutions to extract meaningful insights from this data to improve mental health advocacy, support, and intervention strategies.

2.2 Project Proposal (Proposed Solution)

The proposed solution, "Unlocking the Minds: Analyzing Mental Health with NLP," aims to address the problem through advanced Natural Language Processing (NLP) techniques. This project proposes:

1. **Social Media Sentiment Analysis**: Using NLP models to analyze online discussions about mental health, identifying sentiment trends, recurring themes, and potential support networks within digital communities.
2. **Therapist-Patient Communication Analysis**: Employing NLP tools to examine transcripts of therapy sessions, providing therapists with actionable insights on emotional patterns, progress, and areas for intervention.
3. **Academic Research Paper Analysis**: Applying NLP algorithms to extract key concepts, trends, and gaps from mental health literature, guiding future research and evidence-based practices.

This solution bridges technology and mental health, offering scalable and impactful tools to enhance understanding, research, and treatment in the field.

2.3 Initial Project Planning

1. **Data Collection**:
    - Gather textual data from diverse sources such as social media platforms, anonymized therapy transcripts, and academic databases.
    - Ensure compliance with ethical standards, privacy laws, and informed consent guidelines.
2. **NLP Model Selection and Development**:
    - Select appropriate NLP techniques (e.g., sentiment analysis, topic modeling, and named entity recognition).
    - Fine-tune pre-trained models (e.g., BERT, GPT) for domain-specific analysis.
3. **Implementation Phases**:

- **Phase 1**: Pilot analysis of social media posts to test NLP algorithms and validate findings.
- **Phase 2**: Develop a prototype for therapist-patient transcript analysis, collaborating with mental health professionals for feedback.
- **Phase 3**: Analyze academic papers and present insights to researchers to identify gaps and trends.

4. **Evaluation and Iteration**:
   - Validate the accuracy and relevance of NLP outputs through domain expert review.
   - Refine models based on feedback and expand data sources to enhance analysis.
5. **Output and Deliverables**:
   - Generate dashboards, reports, and insights for stakeholders, including researchers, clinicians, and mental health advocates.
   - Provide an open-access toolkit or framework for further applications in mental health analysis.

3. DATA COLLECTION AND PREPROCESSING PHASE

3.1 Data Collection Plan and Raw Data Sources Identified

Data Collection Plan

The data collection process will involve acquiring textual data from diverse and relevant sources while ensuring compliance with ethical guidelines, privacy laws, and informed consent requirements. Key steps include:

1. **Source Identification**: Select sources that provide varied perspectives on mental health, ensuring coverage of social, clinical, and academic domains.
2. **Data Acquisition**: Use APIs, scraping tools, or manual collection methods to retrieve textual data, adhering to platform and database policies.
3. **Data Annotation**: Prepare annotated datasets, if needed, to train and evaluate NLP models.
4. **Ethical Compliance**: Secure necessary permissions and anonymize sensitive data to protect privacy.

Raw Data Sources Identified

1. **Social Media Platforms**: Posts and discussions from platforms such as Twitter, Reddit, and public mental health forums (e.g., r/mentalhealth on Reddit).
2. **Clinical Transcripts**: Anonymized therapy session transcripts from collaborating mental health practitioners or public repositories.
3. **Academic Research Databases**: Scholarly articles from sources like PubMed, SpringerLink, or arXiv, focusing on mental health topics.
4. **Public Datasets**: Existing datasets such as:

- CLPsych dataset (shared in Computational Linguistics and Clinical Psychology workshops).
- Sentiment140 for social media sentiment analysis.

3.2 Data Quality Report

Preliminary Assessment Metrics

- **Completeness**: Evaluate if data covers the required aspects of mental health topics across all identified sources.
- **Consistency**: Check for uniform formatting (e.g., consistent timestamps, encoding, language).
- **Accuracy**: Validate the reliability of data sources (e.g., authentic posts, verified research articles).
- **Redundancy**: Identify and handle duplicate entries (e.g., repeated posts or papers).
- **Noise**: Analyze and quantify irrelevant content (e.g., advertisements or spam in social media posts).

Observed Issues (Anticipated):

- Social media data may include noisy and unstructured text (e.g., typos, abbreviations).
- Clinical data might have varying formats and require additional anonymization steps.
- Academic articles often include domain-specific jargon, necessitating pre-processing for better NLP interpretation.

3.3 Data Preprocessing

1. **Text Cleaning**:
   - Remove irrelevant characters (e.g., emojis, HTML tags, URLs).
   - Normalize text (e.g., converting to lowercase, expanding contractions).
2. **Tokenization and Stopword Removal**:
   - Break text into smaller units (words, phrases).
   - Remove common stopwords (e.g., "the," "is," "and") to focus on meaningful content.
3. **Noise Filtering**:
   - Remove spam, duplicate entries, and non-mental-health-related content.
4. **Language Standardization**:
   - Translate text into a single language (if needed) using machine translation tools.
5. **Lemmatization and Stemming**:
   - Reduce words to their root forms (e.g., "running" → "run").
6. **Feature Engineering**:
   - Extract linguistic features such as sentiment, topics, and named entities for further analysis.
7. **Data Splitting**:
   - Partition the data into training, validation, and testing sets for NLP model development.

By following this structured plan, the project ensures high-quality, ready-to-use data for NLP analysis.

4. MODEL DEVELOPMENT PHASE

4.1. Model Selection Report

In this project, various machine learning models were evaluated to analyze mental health data using Natural Language Processing (NLP) techniques. The selection process involved assessing performance, complexity, and computational efficiency to determine the most effective model for the given task.

DecisionTree Classifier	Decision Tree Classifier is particularly useful for "Unlocking the Minds: Analyzing Mental Health with NLP" due to its interpretability and ability to handle both categorical and numerical data. It excels in capturing non-linear patterns often present in mental health-related text data.
RandomForest Classifier	Random Forest Classifier is well-suited for this project due to its capability to efficiently handle large feature spaces and its robustness against overfitting through ensemble averaging. This makes it ideal for analyzing complex patterns in mental health text data.
AdaBoost Classifier	AdaBoost Classifier enhances text prediction accuracy by combining weak learners into a strong model. It excels in handling imbalanced datasets and is particularly effective in improving classification performance in mental health-related text analysis through adaptive boosting techniques.
Gradient Boosting Classifier	Gradient Boosting Classifier is ideal for mental health text analysis as it builds robust models by iteratively optimizing a loss function. Its ability to capture intricate patterns in the data ensures high accuracy for nuanced NLP tasks in this domain.
Logistic Regression	Logistic Regression is effective for binary or multiclass text classification tasks within this project. Its simplicity, interpretability, and use of the sigmoid function make it suitable for distinguishing subtle patterns in mental health-related text data.

SupportVector Classifier	Support Vector Classifier (SVM) is particularly effective for high-dimensional textual data. Its kernel-based transformations allow it to classify complex patterns, and its robustness to overfitting ensures reliable predictions, even with limited labeled data in mental health analysis.

4.2. Initial Model Training Code, Model Validation and Evaluation Report

The initial model training code will be showcased in the future through a screenshot. The model validation and evaluation report will include classification reports, accuracy, and confusion matrices for multiple models, presented through respective screenshots.

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix, f1_score

# Vectorize the text data using TF-IDF
vectorizer = TfidfVectorizer()
X_train_tfidf = vectorizer.fit_transform(x_train)  # Apply fit_transform to training data
X_test_tfidf = vectorizer.transform(x_test)  # Apply transform to test data

# Create and train the decision tree classifier
Dt = DecisionTreeClassifier()
Dt.fit(X_train_tfidf, y_train)  # Train the model on the vectorized data

# Make predictions
y_pred = Dt.predict(X_test_tfidf)
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix, f1_score

# Vectorize the text data using TF-IDF
vectorizer = TfidfVectorizer()
X_train_tfidf = vectorizer.fit_transform(x_train)  # Apply fit_transform to training data
X_test_tfidf = vectorizer.transform(x_test)  # Apply transform to test data

# Create and train the random forest classifier
rf = RandomForestClassifier()
rf.fit(X_train_tfidf, y_train)  # Train the model on the vectorized data

# Make predictions
y_pred = rf.predict(X_test_tfidf)
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix, f1_score

# Vectorize the text data using TF-IDF
vectorizer = TfidfVectorizer()
X_train_tfidf = vectorizer.fit_transform(x_train)  # Apply fit_transform to training data
X_test_tfidf = vectorizer.transform(x_test)  # Apply transform to test data

# Create and train the AdaBoost classifier
ab = AdaBoostClassifier()
ab.fit(X_train_tfidf, y_train)  # Train the model on the vectorized data

# Make predictions
y_pred = ab.predict(X_test_tfidf)
```

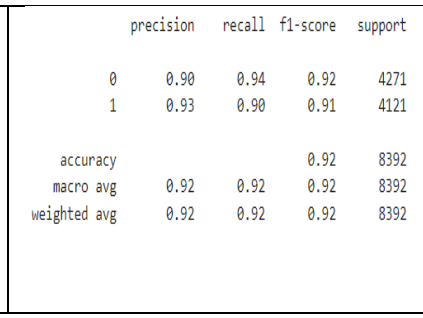
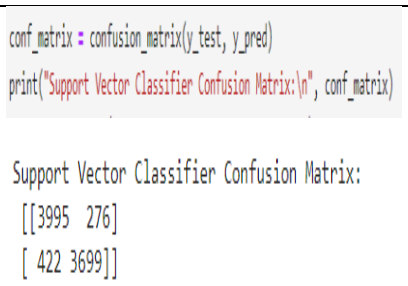
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix, f1_score

# Vectorize the text data using TF-IDF
vectorizer = TfidfVectorizer()
X_train_tfidf = vectorizer.fit_transform(x_train)  # Apply fit_transform to training data
X_test_tfidf = vectorizer.transform(x_test)  # Apply transform to test data

# Create and train the Gradient Boosting classifier
gb = GradientBoostingClassifier()
gb.fit(X_train_tfidf, y_train)  # Train the model on the vectorized data

# Make predictions
y_pred = gb.predict(X_test_tfidf)
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix, f1_score

# Vectorize the text data using TF-IDF
vectorizer = TfidfVectorizer()
X_train_tfidf = vectorizer.fit_transform(x_train)  # Apply fit_transform to training data
X_test_tfidf = vectorizer.transform(x_test)  # Apply transform to test data

# Create and train the Logistic Regression classifier
lr = LogisticRegression(max_iter=1000)
lr.fit(X_train_tfidf, y_train)  # Train the model on the vectorized data

# Make predictions
y_pred = lr.predict(X_test_tfidf)
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix, f1_score

# Vectorize the text data using TF-IDF
vectorizer = TfidfVectorizer()
X_train_tfidf = vectorizer.fit_transform(x_train)  # Apply fit_transform to training data
X_test_tfidf = vectorizer.transform(x_test)  # Apply transform to test data

# Create and train the Support Vector Classifier (SVC)
sv = SVC()
sv.fit(X_train_tfidf, y_train)  # Train the model on the vectorized data

# Make predictions
y_pred = sv.predict(X_test_tfidf)
```

Model Validation and Evaluation Report:

Model	Classification Report	F1 Score	Confusion Matrix
Decision Tree Classfier	precision recall f1-score support 0 0.82 0.83 0.82 4271 1 0.82 0.81 0.81 4121 accuracy 0.82 8392 macro avg 0.82 0.82 0.82 8392 weighted avg 0.82 0.82 0.82 8392 0.8169685414680649	81%	conf_matrix = confusion_matrix(y_test, y_pred) print("Decision Tree Confusion Matrix:\n", conf_matrix) Decision Tree Confusion Matrix: [[3524 747] [789 3332]]
Random Forest Classifier	precision recall f1-score support 0 0.88 0.90 0.89 4271 1 0.89 0.87 0.88 4121 accuracy 0.88 8392 macro avg 0.88 0.88 0.88 8392 weighted avg 0.88 0.88 0.88 8392 0.8831029551954243	88%	conf_matrix = confusion_matrix(y_test, y_pred) print("Random Forest Confusion Matrix:\n", conf_matrix) Random Forest Confusion Matrix: [[3836 435] [546 3575]]
Adaboost Classifier	precision recall f1-score support 0 0.84 0.91 0.87 4271 1 0.90 0.82 0.86 4121 accuracy 0.87 8392 macro avg 0.87 0.87 0.87 8392 weighted avg 0.87 0.87 0.87 8392 0.8666587225929456	86%	conf_matrix = confusion_matrix(y_test, y_pred) print("AdaBoost Confusion Matrix:\n", conf_matrix) AdaBoost Confusion Matrix: [[3906 365] [754 3367]]
Gradient Boosting Classifier	precision recall f1-score support 0 0.83 0.91 0.87 4271 1 0.90 0.81 0.85 4121 accuracy 0.86 8392 macro avg 0.87 0.86 0.86 8392 weighted avg 0.86 0.86 0.86 8392 0.8609389895138226	86%	conf_matrix = confusion_matrix(y_test, y_pred) print("Gradient Boosting Confusion Matrix:\n", conf_matrix) Gradient Boosting Confusion Matrix: [[3900 371] [796 3325]]
Logistic Regression	precision recall f1-score support 0 0.90 0.93 0.91 4271 1 0.93 0.89 0.91 4121 accuracy 0.91 8392 macro avg 0.91 0.91 0.91 8392 weighted avg 0.91 0.91 0.91 8392 0.911582459485224	91%	conf_matrix = confusion_matrix(y_test, y_pred) print("Logistic Regression Confusion Matrix:\n", conf_matrix) Logistic Regression Confusion Matrix: [[3975 296] [446 3675]]

Support Vector Classifier		92%	

Training and Validation Performance Metrics:

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
import joblib
import pickle
```

```
# Define and train the model
model = SVC()
model.fit(X_train, y_train)

# Save the model and vectorizer
with open('model.pkl', 'wb') as f:
    joblib.dump(model, f)
with open('TfidfVectorizer.pkl', 'wb') as f:
    joblib.dump(tfidf_vectorizer, f)

print("Model and vectorizer saved successfully!")
```

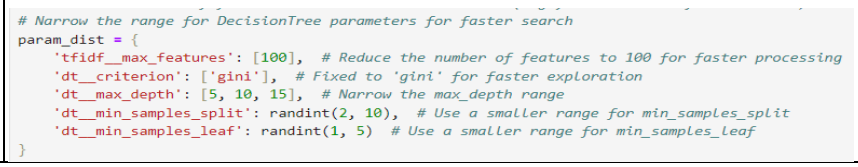
```
Model and vectorizer saved successfully!
```

5. MODEL OPTIMIZATION AND TUNING PHASE

5.1. Tuning Documentation

The Model Optimization and Tuning Phase involves refining machine learning models for peak performance. It includes optimized model code, fine-tuning hyperparameters, comparing performance metrics, and justifying the final model selection for enhanced predictive accuracy and efficiency.

Hyperparameter Tuning Documentation :

Model	Tuned Hyperparameters
Decision Tree Classifier	

```
# Narrow the range for DecisionTree parameters for faster search
param_dist = {
    'tfidf__max_features': [100],  # Reduce the number of features to 100 for faster processing
    'dt__criterion': ['gini'],  # Fixed to 'gini' for faster exploration
    'dt__max_depth': [5, 10, 15],  # Narrow the max_depth range
    'dt__min_samples_split': randint(2, 10),  # Use a smaller range for min_samples_split
    'dt__min_samples_leaf': randint(1, 5)  # Use a smaller range for min_samples_leaf
}
```

	Best Parameters for Decision Tree: {'dt_criterion': 'gini', 'dt_max_depth': 10, 'dt_min_samples_leaf': 3, 'dt_min_samples_split': 8, 'tfidf_max_features': 100}

Random Forest Classifier	`# Parameter distribution for RandomForestClassifier`
	`param_dist = {`
	`'tfidf__max_features': [100], # Reduce the number of features to 100 for faster processing`
	`'rf__n_estimators': [50, 100, 200], # Number of trees in the forest`
	`'rf__max_depth': [5, 10, 15], # Maximum depth of the tree`
	`'rf__min_samples_split': randint(2, 10), # Minimum samples required to split an internal node`
	`'rf__min_samples_leaf': randint(1, 5), # Minimum samples required to be at a leaf node`
	`'rf__bootstrap': [True, False] # Whether bootstrap samples are used when building trees`
	`}`
	
	Best Parameters for Random Forest: {'rf__bootstrap': False, 'rf__max_depth': 15, 'rf__min_samples_leaf': 4, 'rf__min_samples_split': 6, 'rf__n_estimators': 200, 'tfidf__max_features': 100}

AdaBoost Classifier	`# Define parameter distribution for AdaBoostClassifier`
	`param_dist = {`
	`'tfidf__max_features': [100], # Reduce the number of features to 100 for faster processing`
	`'adaboost__n_estimators': randint(10, 100), # Number of estimators in AdaBoost`
	`'adaboost__learning_rate': uniform(0.1, 1.0) # Learning rate for AdaBoost`
	`}`
	
	Best Parameters for AdaBoost: {'adaboost__learning_rate': 0.696850157946487, 'adaboost__n_estimators': 92, 'tfidf__max_features': 100}

Gradient Boosting Classifier	`# Narrow the range for GradientBoosting parameters for faster search`
	`param_dist = {`
	`'tfidf__max_features': [100], # Reduce the number of features to 100 for faster processing`
	`'gb__n_estimators': [50, 100, 150], # Limit the number of estimators`
	`'gb__learning_rate': uniform(0.01, 0.3), # Explore learning rates between 0.01 and 0.3`
	`'gb__max_depth': [3, 5, 7], # Use a smaller range for max_depth`
	`'gb__min_samples_split': randint(2, 10), # Use a smaller range for min_samples_split`
	`'gb__min_samples_leaf': randint(1, 5) # Use a smaller range for min_samples_leaf`
	`}`
	
	Best Parameters for Gradient Boosting Classifier: {'gb__learning_rate': 0.2149790556476374 7, 'gb__max_depth': 3, 'gb__min_samples_leaf': 4, 'gb__min_samples_split': 8, 'gb__n_estimators': 150, 'tfidf__max_features': 100}

Logistic Regression	`# Define parameter distribution for Logistic Regression`
	`param_dist = {`
	`'tfidf__max_features': [100], # Reduce the number of features to 100 for faster processing`
	`'lr__C': uniform(0.1, 10), # Regularization strength (search over 0.1 to 10)`
	`'lr__penalty': ['l2'], # Use only L2 regularization for simplicity`
	`'lr__solver': ['lbfgs'], # Use 'lbfgs' for faster convergence`
	`'lr__max_iter': [100, 200, 300] # Narrow max_iter range`
	`}`
	
	Best Parameters for Logistic Regression: {'lr__C': 6.274815096277165, 'lr__max_iter': 200, 'lr__penalty': 'l2', 'lr__solver': 'lbfgs', 'tfidf__max_features': 100}

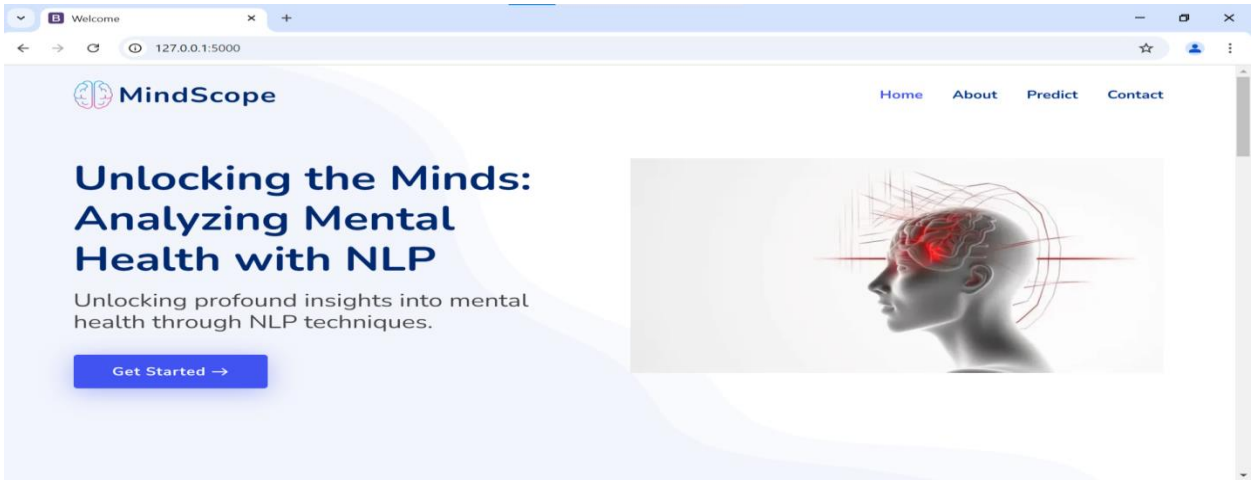
Support Vector Classifier	`# Narrow the range for SVC parameters for faster search`
	`param_dist = {`
	`'tfidf__max_features': [100], # Reduce the number of features to 100 for faster processing`
	`'svc__C': uniform(0.1, 10), # Narrow range of regularization parameter`
	`'svc__kernel': ['linear', 'rbf'], # Explore common kernels`
	`'svc__gamma': uniform(0.01, 1) # Smaller range for gamma`
	`}`
	
	Best Parameters for SVC: {'svc__C': 1.833646535077721, 'svc__gamma': 0.4010606075732408, 'svc__kernel': 'rbf', 'tfidf__max_features': 100}

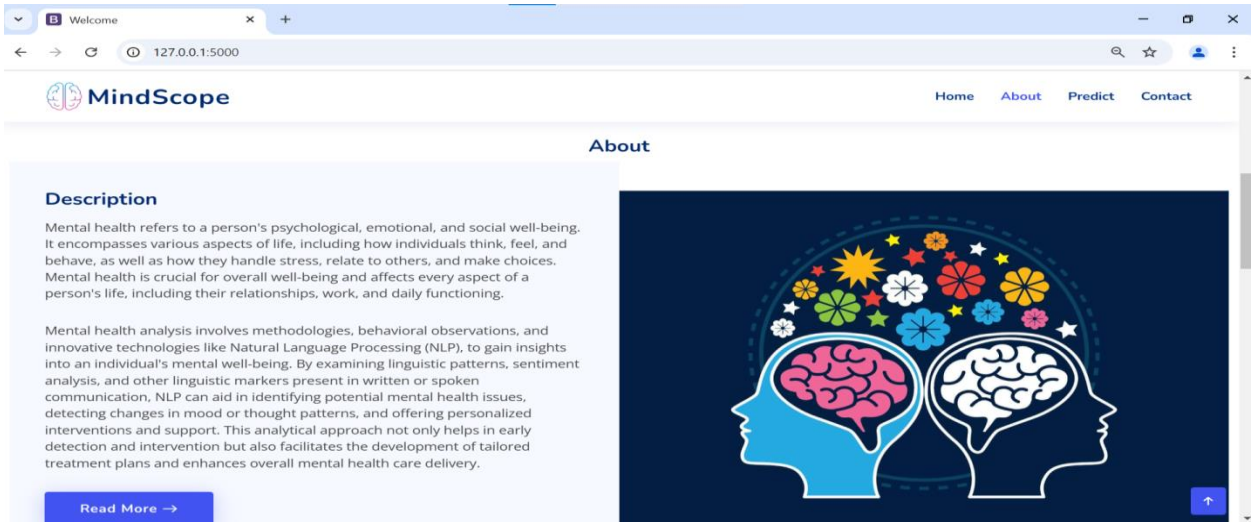
5.2. Final Model Selection Justification

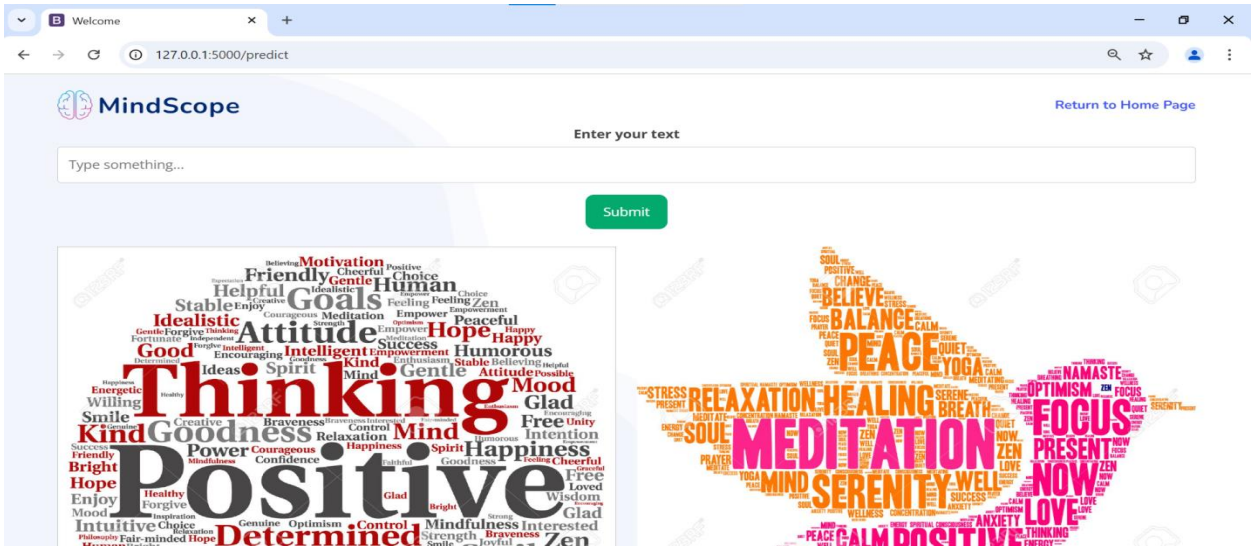
Final Model	Reasoning
Support Vector Classifier (SVC)	The Support Vector Classifier (SVC) was selected for its effectiveness in handling high-dimensional textual data, making it particularly suitable for mental health analysis. Its ability to classify complex patterns using kernel-based transformations ensures high accuracy in identifying nuanced text features. Additionally, SVC's robustness to overfitting provides reliable predictions even with limited labeled data, aligning well with the project's objectives in "Unlocking the Minds: Analyzing Mental Health with NLP."

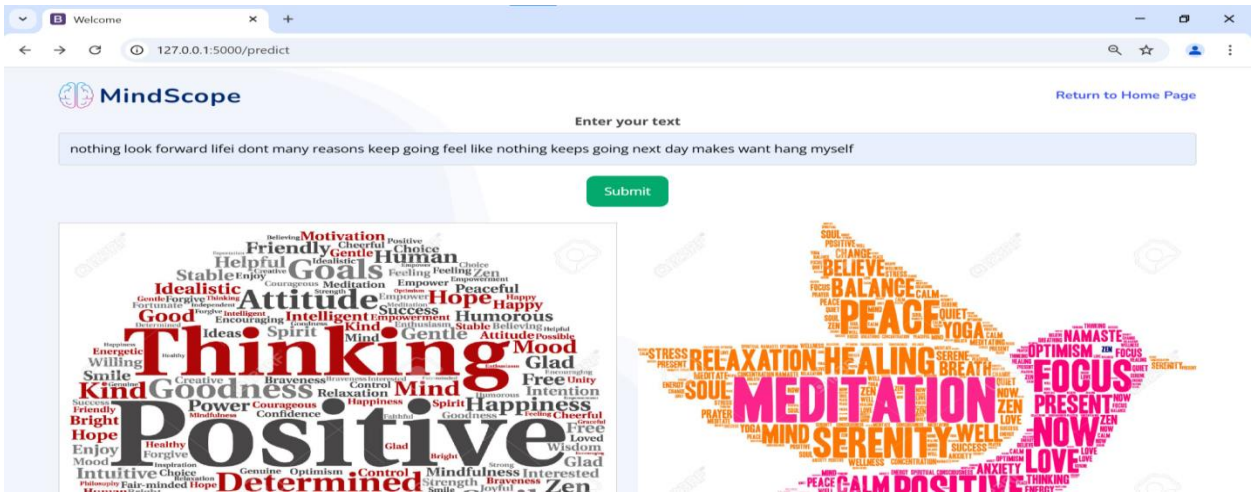
6. RESULTS

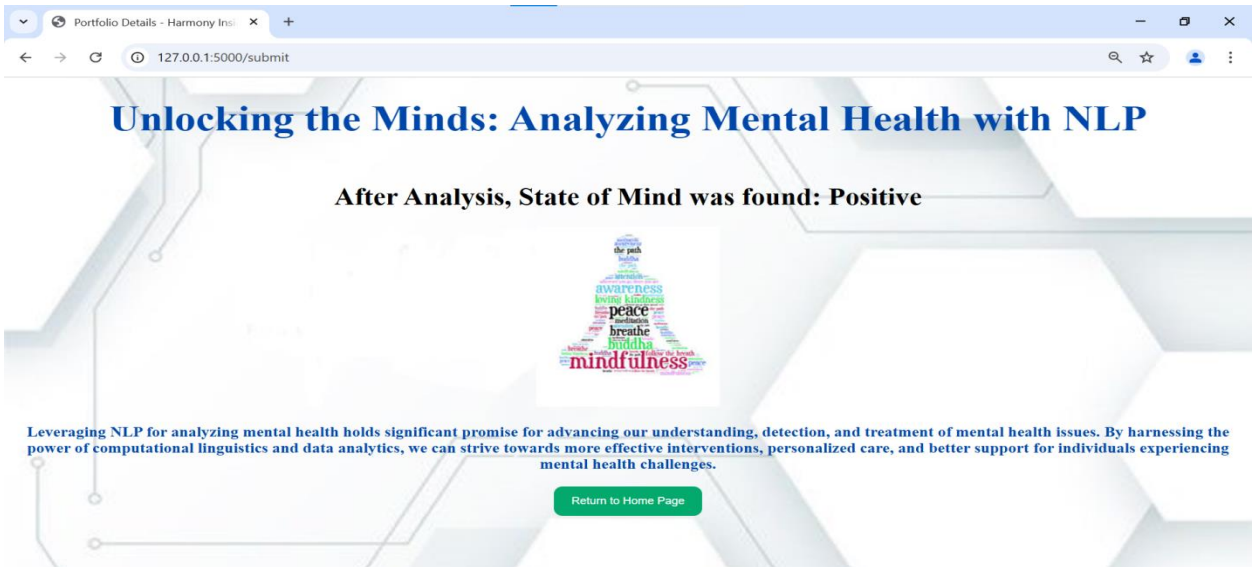
6.1. Output Screenshots

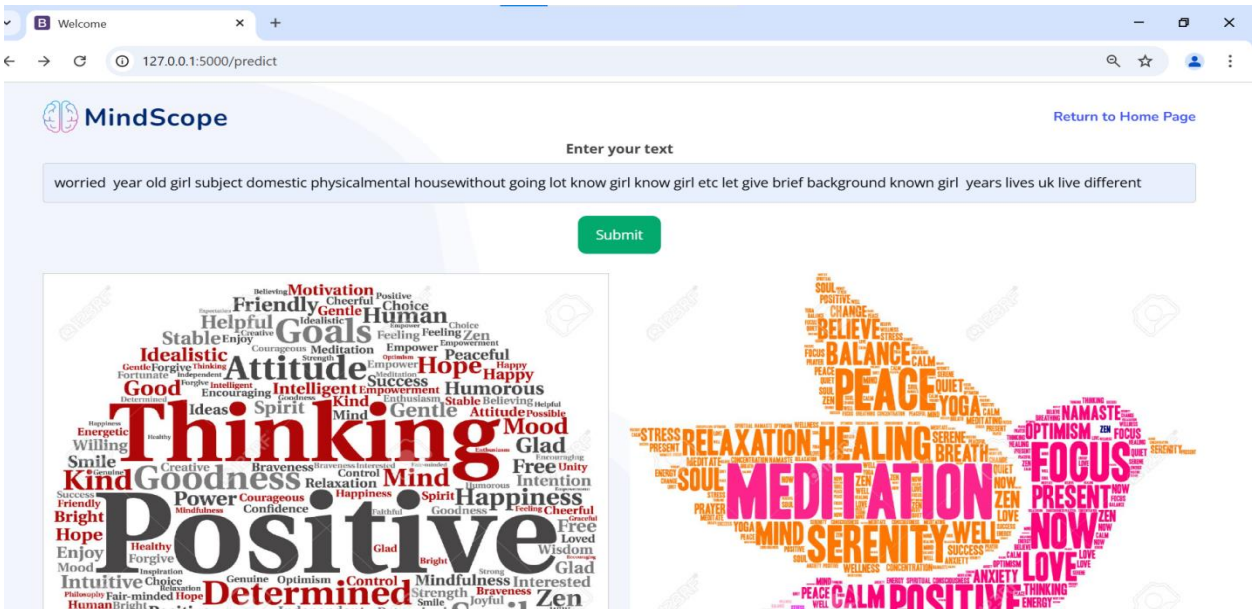


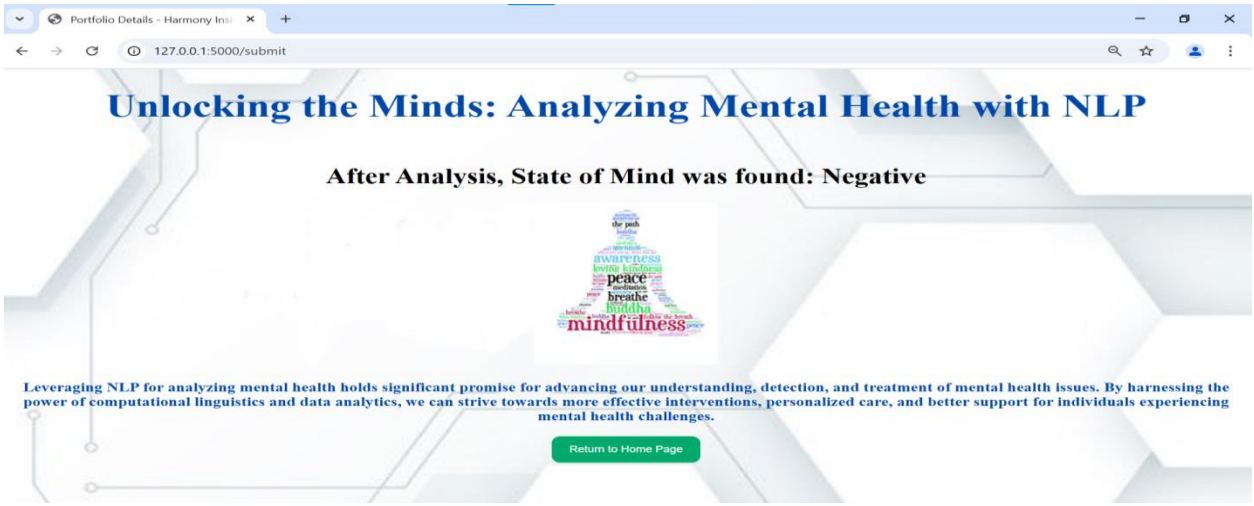



Welcome
127.0.0.1:5000/predict
MindScope
Return to Home Page
Enter your text
Type something...
Submit


Welcome
127.0.0.1:5000/predict
MindScope
Return to Home Page
Enter your text
nothing look forward lifei dont many reasons keep going feel like nothing keeps going next day makes want hang myself
Submit


Portfolio Details - Harmony Ins
127.0.0.1:5000/submit
Unlocking the Minds: Analyzing Mental Health with NLP
After Analysis, State of Mind was found: Positive
Leveraging NLP for analyzing mental health holds significant promise for advancing our understanding, detection, and treatment of mental health issues. By harnessing the power of computational linguistics and data analytics, we can strive towards more effective interventions, personalized care, and better support for individuals experiencing mental health challenges.
Return to Home Page





7. ADVANTAGES & DISADVANTAGES

Advantages

1. **Scalability**:
   - NLP techniques allow for the analysis of vast amounts of textual data from diverse sources (e.g., social media, clinical transcripts, academic papers).
   - Automates repetitive tasks like topic modeling and sentiment analysis, saving time and resources.
2. **Enhanced Insights**:
   - Identifies nuanced linguistic patterns, sentiment trends, and recurring themes that might be overlooked by manual analysis.
   - Provides actionable insights for mental health advocacy, personalized therapy, and research.

3. **Improved Mental Health Support**:
    - Social media sentiment analysis can detect emerging mental health trends and challenges in real-time.
    - Therapists can use communication analysis to better understand patients' emotions and tailor interventions.
4. **Data-Driven Research**:
    - Helps researchers extract key concepts and trends from academic literature, advancing evidence-based mental health practices.
5. **Destigmatization of Mental Health**:
    - Promotes awareness by providing data-backed insights into mental health struggles, reducing societal stigma.
6. **Versatility**:
    - Can be adapted to multiple applications, such as monitoring mental health campaigns, creating early-warning systems, and enhancing educational materials.

Disadvantages

1. **Data Privacy and Ethical Concerns**:
    - Analyzing sensitive mental health data (e.g., therapy sessions, social media posts) raises concerns about data security and user privacy.
    - Requires strict adherence to ethical standards, including informed consent and anonymization.
2. **Data Quality Issues**:
    - Text data from social media often contains noise, slang, and abbreviations, complicating analysis.
    - Clinical transcripts may vary in format, requiring extensive preprocessing.
3. **Interpretability of Results**:
    - NLP models, especially deep learning-based ones like BERT, can be difficult to interpret, leading to challenges in explaining results to non-technical stakeholders.
4. **Domain Adaptation Challenges**:
    - Pre-trained models may require significant fine-tuning to perform effectively on domain-specific mental health datasets.
    - Limited availability of annotated datasets for niche mental health topics can hinder model training.
5. **Potential Bias**:
    - NLP models can inherit biases from training data, leading to skewed or inaccurate insights. For instance, underrepresentation of certain groups in the data might result in inequitable outcomes.
6. **Dependence on Technology**:
    - Over-reliance on NLP tools might lead to overlooking human expertise, especially in clinical or therapeutic settings.
7. **Misinterpretation Risks**:
    - Sentiment or linguistic patterns alone might not fully capture the complexity of mental health conditions, leading to oversimplification or misinterpretation.

8. CONCLUSION

"Unlocking the Minds: Analyzing Mental Health with NLP" demonstrates the potential of leveraging advanced Natural Language Processing techniques to gain valuable insights into the complexities of mental health. By analyzing diverse textual data from social media, therapy sessions, and academic literature, the project highlights how linguistic patterns, sentiments, and themes can contribute to a deeper understanding of mental health conditions. The proposed solutions not only provide actionable insights for advocacy, research, and clinical support but also underscore the importance of integrating technology to address critical societal challenges.

While the project offers significant advantages in scalability, efficiency, and data-driven decision-making, it also acknowledges challenges such as ethical concerns, data privacy, and model biases. Addressing these limitations is crucial to ensure that the implementation of NLP in mental health aligns with ethical standards and delivers equitable outcomes. Overall, this project represents a step forward in using AI to bridge gaps in mental health research and practice.

9. FUTURE SCOPE

1. **Expanded Data Sources**:
    - Incorporate additional data, such as multilingual text, video or audio transcripts, and patient journals, to enrich analysis.
    - Explore global datasets to understand cultural differences in mental health expression.
2. **Early-Warning Systems**:
    - Develop systems that use real-time social media analysis to identify potential mental health crises or emerging trends.
    - Collaborate with public health organizations to create tools for early intervention.
3. **Integration with Healthcare Systems**:
    - Deploy NLP tools in clinical settings to support diagnosis and treatment planning.
    - Build platforms for therapists to visualize insights from session transcripts in real-time.
4. **Customization and Personalization**:
    - Enhance NLP models to provide personalized recommendations for individuals based on their unique linguistic patterns and mental health history.
5. **Ethical AI in Mental Health**:
    - Focus on developing explainable AI models to improve interpretability and trust among clinicians and researchers.
    - Implement advanced privacy-preserving techniques, such as federated learning and differential privacy, to secure sensitive data.
6. **Cross-Disciplinary Research**:
    - Collaborate with psychologists, linguists, and social scientists to refine NLP methods for mental health applications.
    - Conduct studies to validate the real-world impact of insights derived from NLP analysis.

7. **Educational and Advocacy Tools**:
    - Create user-friendly dashboards and applications for educators, policymakers, and mental health advocates to promote awareness and destigmatization.

10. APPENDIX

10.1. Source Code

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import nltk # spacy
import re # removing the special characters
nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
nltk.download("punkt")
from nltk.tokenize import word_tokenize
from sklearn.feature_extraction.text import TfidfVectorizer
df = pd.read_csv('C:/Users/lenovo/Desktop/Unlocking_minds/dataset/mental_health.csv')
df.shape
df.dtypes
df['label'].unique()
df.isnull().sum()
df.duplicated().sum()
df = df.drop_duplicates()
print('Number of Duplicates:', len(df[df.duplicated()]))
df['label'].value_counts()
df.describe(include='all')
plt.figure(figsize=(5,6))  # set the size of the plot
plt.pie(df['label'].value_counts(), labels=df['label'].value_counts().index, autopct='%1.1f%%',
      explode=[0, 0.05], startangle=140)
plt.show()
def remove_stopwords(sentence):
   # List of stopwords
   stopwords = ["a", "about", "above", "after", "again", "against", "all", "am", "an", "and", "any",
            "are", "as", "at", "be", "because", "been", "before", "being", "below", "between",
            "both", "but", "by", "could", "did", "do", "does", "doing", "down", "during", "each",
            "few", "for", "from", "further", "had", "has", "have", "having", "he", "he'd", "he'll",
            "he's",  "her",  "here",  "here's",  "hers",  "herself",  "him",  "himself",  "his",  "how",
"how's", "i", "i'd", "i'll", "i'm", "i've", "if", "in", "into", "is", "it", "it's", "its", "itself",
            "let's", "me", "more", "most", "my", "myself", "nor", "of", "on", "once", "only", "or",
            "other", "ought", "our", "ours", "ourselves", "out", "over", "own", "same", "she",
```

```
            "she'd", "she'll", "she's", "should", "so", "some", "such", "than", "that", "that's",
            "the", "their", "theirs", "them", "themselves", "then", "there", "there's", "these",
            "they", "they'd", "they'll", "they're", "they've", "this", "those", "through", "to",
            "too", "under", "until", "up", "very", "was", "we", "we'd", "we'll", "we're", "we've",
            "were", "what", "what's", "when", "when's", "where", "where's", "which", "while",
"who","who's", "whom", "why", "why's", "with", "would", "you", "you'd", "you'll", "you're",
            "you've", "your", "yours", "yourself", "yourselves"]
    # Sentence converted to lowercase-only
    sentence = sentence.lower()
    words = sentence.split()
    no_words = [w for w in words if w not in stopwords]
    sentence = " ".join(no_words)
    return sentence
df['text1'] = (df['text'].apply(remove_stopwords))
msg=df['text1'].str.replace('[^a-zA-Z0-9]+',"")
msg
lemmatizer = WordNetLemmatizer()
data=[]
df_comp = pd.DataFrame()
# Original text and its length
df_comp['pre-clean text'] = df['text']
df_comp['pre-clean len'] = df['text'].apply(lambda x: len(str(x).split()))
# Cleaned text and its length
df_comp['post-clean text'] = df['text1']
df_comp['post-clean len'] = df['text1'].apply(lambda x: len(str(x).split()))
df_comp.head(20)
def preprocess_text(text):
    #Tokenize the text
    tokens = word_tokenize(text)
    # Filter tokens with length greater than 2
    filtered_tokens = [token for token in tokens if len(token) > 3]
    #Stem each token
    lemmed_tokens = [lemmatizer.lemmatize(token.lower()) for token in filtered_tokens]
    #Join stemmed tokens into a single string
    preprocessed_text = " ".join(lemmed_tokens)
    return preprocessed_text
import nltk
nltk.download('wordnet')
nltk.download('omw-1.4')
# Apply text preprocessing to each row of the DataFrame
df['preprocessed_text'] = df['text'].apply(preprocess_text)
df['preprocessed_text']
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
tf=TfidfVectorizer()
data_vec=tf.fit_transform(df['preprocessed_text'])
print(data_vec)
df['text_length'] = df['preprocessed_text'].apply(lambda x: len(str(x).split()))
df = df.drop(['text', 'text1'], axis=1)
df.head()
y=df['label'].values
y
print("Column names:", df.columns.tolist())
import pandas as pd
data = pd.DataFrame(data)
print(data.columns)
# Ensure X and y are correctly assigned before splitting
X = df['preprocessed_text']  # Features
y = df['label']  # Labels
# Split the dataset into training and testing sets
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)
# Now vectorize the text data using TF-IDF
vectorizer = TfidfVectorizer()
X_train_tfidf = vectorizer.fit_transform(x_train)  # Apply fit_transform to training data
X_test_tfidf = vectorizer.transform(x_test)  # Apply transform to test data
# Check the shapes
print(f"x_train_tfidf shape: {X_train_tfidf.shape}")
print(f"y_train shape: {y_train.shape}")
X = df['preprocessed_text']  # Assuming 'preprocessed_text' is the feature column
y = df['label']  # Label column
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)
X = df[['preprocessed_text', 'text_length']]  # Example with multiple features
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix, f1_score
# Vectorize the text data using TF-IDF
vectorizer = TfidfVectorizer()
X_train_tfidf = vectorizer.fit_transform(x_train)  # Apply fit_transform to training data
X_test_tfidf = vectorizer.transform(x_test)  # Apply transform to test data
# Create and train the decision tree classifier
Dt = DecisionTreeClassifier()
Dt.fit(X_train_tfidf, y_train)  # Train the model on the vectorized data
# Make predictions
y_pred = Dt.predict(X_test_tfidf)
# Print classification report and accuracy score
```

```
print(classification_report(y_test, y_pred))
print(accuracy_score(y_test, y_pred))
# Calculate and print confusion matrix and F1 score
conf_matrix = confusion_matrix(y_test, y_pred)
print("Decision Tree Confusion Matrix:\n", conf_matrix)
dt_f1 = f1_score(y_test, y_pred, average='weighted')
dt_f1_percentage = dt_f1 * 100
print(f"Decision Tree F1 Score: {dt_f1}")
print("Decision Tree F1 Score (%):", dt_f1_percentage)
dt_acc=accuracy_score(y_test,y_pred)
dt_acc
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix, f1_score
# Vectorize the text data using TF-IDF
vectorizer = TfidfVectorizer()
X_train_tfidf = vectorizer.fit_transform(x_train)  # Apply fit_transform to training data
X_test_tfidf = vectorizer.transform(x_test)  # Apply transform to test data
# Create and train the random forest classifier
rf = RandomForestClassifier()
rf.fit(X_train_tfidf, y_train)  # Train the model on the vectorized data
# Make predictions
y_pred = rf.predict(X_test_tfidf)
# Print classification report and accuracy score
print(classification_report(y_test, y_pred))
print(accuracy_score(y_test, y_pred))
# Calculate and print confusion matrix and F1 score
conf_matrix = confusion_matrix(y_test, y_pred)
print("Random Forest Confusion Matrix:\n", conf_matrix)
rf_f1 = f1_score(y_test, y_pred, average='weighted')
rf_f1_percentage = rf_f1 * 100
print(f"Random Forest F1 Score: {rf_f1}")
print("Random Forest F1 Score (%):", rf_f1_percentage)
rf_acc=accuracy_score(y_test,y_pred)
rf_acc
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix, f1_score
# Vectorize the text data using TF-IDF
vectorizer = TfidfVectorizer()
X_train_tfidf = vectorizer.fit_transform(x_train)  # Apply fit_transform to training data
X_test_tfidf = vectorizer.transform(x_test)  # Apply transform to test data
# Create and train the AdaBoost classifier
```

```
ab = AdaBoostClassifier()
ab.fit(X_train_tfidf, y_train)  # Train the model on the vectorized data
# Make predictions
y_pred = ab.predict(X_test_tfidf)

# Print classification report and accuracy score
print(classification_report(y_test, y_pred))
print(accuracy_score(y_test, y_pred))
# Calculate and print confusion matrix and F1 score
conf_matrix = confusion_matrix(y_test, y_pred)
print("AdaBoost Confusion Matrix:\n", conf_matrix)
ab_f1 = f1_score(y_test, y_pred, average='weighted')
ab_f1_percentage = ab_f1 * 100
print(f"AdaBoost F1 Score: {ab_f1}")
print("AdaBoost F1 Score (%):", ab_f1_percentage)
ab_acc=accuracy_score(y_test,y_pred)
ab_acc
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix, f1_score
# Vectorize the text data using TF-IDF
vectorizer = TfidfVectorizer()
X_train_tfidf = vectorizer.fit_transform(x_train)  # Apply fit_transform to training data
X_test_tfidf = vectorizer.transform(x_test)  # Apply transform to test data
# Create and train the Gradient Boosting classifier
gb = GradientBoostingClassifier()
gb.fit(X_train_tfidf, y_train)  # Train the model on the vectorized data
# Make predictions
y_pred = gb.predict(X_test_tfidf)
# Print classification report and accuracy score
print(classification_report(y_test, y_pred))
print(accuracy_score(y_test, y_pred))
# Calculate and print confusion matrix and F1 score
conf_matrix = confusion_matrix(y_test, y_pred)
print("Gradient Boosting Confusion Matrix:\n", conf_matrix)
gb_f1 = f1_score(y_test, y_pred, average='weighted')
gb_f1_percentage = gb_f1 * 100
print(f"Gradient Boosting F1 Score: {gb_f1}")
print("Gradient Boosting F1 Score (%):", gb_f1_percentage)
gb_acc=accuracy_score(y_test,y_pred)
gb_acc
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix, f1_score
# Vectorize the text data using TF-IDF
vectorizer = TfidfVectorizer()
X_train_tfidf = vectorizer.fit_transform(x_train)  # Apply fit_transform to training data
X_test_tfidf = vectorizer.transform(x_test)  # Apply transform to test data
# Create and train the Logistic Regression classifier
lr = LogisticRegression(max_iter=1000)
lr.fit(X_train_tfidf, y_train)  # Train the model on the vectorized data
# Make predictions
y_pred = lr.predict(X_test_tfidf)
# Print classification report and accuracy score
print(classification_report(y_test, y_pred))
print(accuracy_score(y_test, y_pred))
# Calculate and print confusion matrix and F1 score
conf_matrix = confusion_matrix(y_test, y_pred)
print("Logistic Regression Confusion Matrix:\n", conf_matrix)
lr_f1 = f1_score(y_test, y_pred, average='weighted')
lr_f1_percentage = lr_f1 * 100
print(f"Logistic Regression F1 Score: {lr_f1}")
print("Logistic Regression F1 Score (%):", lr_f1_percentage)
lr_acc=accuracy_score(y_test,y_pred)
lr_acc
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix, f1_score
# Vectorize the text data using TF-IDF
vectorizer = TfidfVectorizer()
X_train_tfidf = vectorizer.fit_transform(x_train)  # Apply fit_transform to training data
X_test_tfidf = vectorizer.transform(x_test)  # Apply transform to test data
# Create and train the Support Vector Classifier (SVC)
sv = SVC()
sv.fit(X_train_tfidf, y_train)  # Train the model on the vectorized data
# Make predictions
y_pred = sv.predict(X_test_tfidf)
# Print classification report and accuracy score
print(classification_report(y_test, y_pred))
print(accuracy_score(y_test, y_pred))
# Calculate and print confusion matrix and F1 score
conf_matrix = confusion_matrix(y_test, y_pred)
print("Support Vector Classifier Confusion Matrix:\n", conf_matrix)
svc_f1 = f1_score(y_test, y_pred, average='weighted')
svc_f1_percentage = svc_f1 * 100
print(f"Support Vector Classifier F1 Score: {svc_f1}")
```

```
print("Support Vector Classifier F1 Score (%):", svc_f1_percentage)
sv_acc=accuracy_score(y_test,y_pred)
sv_acc
model = pd.DataFrame({'Model': [ 'Decision Tree', 'Random Forest', 'AdaBoost',
'GradientBoosting', 'Logistic Regression','Support Vector Machine'], 'Score':
[dt_acc,rf_acc,ab_acc,gb_acc,lr_acc,sv_acc]})
model
# Use the same vectorizer (vectorizer) that was used during training
y_new = sv.predict(vectorizer.transform(["I'm overwhelmed with anxiety about the future."]))
if y_new == 1:  # Use '==' for comparison
   print("Positive")
elif y_new == 0:  # Use '==' for comparison
   print("Negative")
# Use the same vectorizer that was used during training
y_new = sv.predict(vectorizer.transform(["I had a great time with friends last night."]))
if y_new == 1:  # Check if the predicted label is 1
   print("Positive")
elif y_new == 0:  # Check if the predicted label is 0
   print("Negative")
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
import joblib
import pickle
# Load data from a CSV file
df = pd.read_csv('C:/Users/lenovo/Desktop/Unlocking_minds/dataset/mental_health.csv')
# Extract documents and labels
documents = df['text'].tolist()
labels = df['label'].tolist()
# Define and fit the TfidfVectorizer
tfidf_vectorizer = TfidfVectorizer()
X = tfidf_vectorizer.fit_transform(documents)
# Split data
X_train, X_test, y_train, y_test = train_test_split(X, labels, test_size=0.2, random_state=42)
# Define and train the model
model = SVC()
model.fit(X_train, y_train)
# Save the model and vectorizer
with open('model.pkl', 'wb') as f:
   joblib.dump(model, f)
with open('TfidfVectorizer.pkl', 'wb') as f:
   joblib.dump(tfidf_vectorizer, f)
```

```
print("Model and vectorizer saved successfully!")
import joblib
import pickle

try:
    # Load the model and vectorizer
    with open('model.pkl', 'rb') as f:
        model = joblib.load(f)
    with open('TfidfVectorizer.pkl', 'rb') as f:
        tfidf_vectorizer = joblib.load(f)
    print("Model and vectorizer loaded successfully!")
except Exception as e:
    print(f"Error during loading: {e}")
```

Index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <meta content="width=device-width, initial-scale=1.0" name="viewport">
  <title>Welcome</title>
  <meta content="" name="description">
  <meta content="" name="keywords">
  <!-- Favicons -->
  <link href="static/img/favicon.png" rel="icon">
  <link href="static/img/apple-touch-icon.png" rel="apple-touch-icon">
  <!-- Google Fonts -->
  <link
href="https://fonts.googleapis.com/css?family=Open+Sans:300,300i,400,400i,600,600i,700,700i|
Nunito:300,300i,400,400i,600,600i,700,700i|Poppins:300,300i,400,400i,500,500i,600,600i,700,7
00i" rel="stylesheet">
  <!-- Vendor CSS Files -->
  <link href="static/vendor/aos/aos.css" rel="stylesheet">
  <link href="static/vendor/bootstrap/css/bootstrap.min.css" rel="stylesheet">
  <link href="static/vendor/bootstrap-icons/bootstrap-icons.css" rel="stylesheet">
  <link href="static/vendor/glightbox/css/glightbox.min.css" rel="stylesheet">
  <link href="static/vendor/remixicon/remixicon.css" rel="stylesheet">
  <link href="static/vendor/swiper/swiper-bundle.min.css" rel="stylesheet">
  <!-- Template Main CSS File -->
  <link href="static/css/style.css" rel="stylesheet">
  <!-- =======================================================
  * Template Name: FlexStart
  * Updated: Jan 29 2024 with Bootstrap v5.3.2
```

```
  * Template URL: https://bootstrapmade.com/flexstart-bootstrap-startup-template/
  * Author: BootstrapMade.com
  * License: https://bootstrapmade.com/license/
  ======================================================== -->
</head>

<body>
  <!-- ======= Header ======= -->
  <header id="header" class="header fixed-top">
    <div class="container-fluid container-xl d-flex align-items-center justify-content-between">
      <a href="index.html" class="logo d-flex align-items-center">
        <img src="static/img/logo.png" alt="">
        <span>MindScope</span>
      </a>
      <nav id="navbar" class="navbar">
        <ul>
          <li><a class="nav-link scrollto active" href="#hero">Home</a></li>
          <li><a class="nav-link scrollto" href="#about">About</a></li>
          <li><a class="nav-link scrollto" href="/predict">Predict</a></li>
          <li><a class="nav-link scrollto" href="#contact">Contact</a></li>
        </ul>
        <i class="bi bi-list mobile-nav-toggle"></i>
      </nav><!-- .navbar -->
    </div>
  </header><!-- End Header -->
  <!-- ======= Hero Section ======= -->
  <section id="hero" class="hero d-flex align-items-center">
    <div class="container">
      <div class="row">
        <div class="col-lg-6 d-flex flex-column justify-content-center">
          <h1 data-aos="fade-up">Unlocking the Minds: Analyzing Mental Health with NLP</h1>
          <h2 data-aos="fade-up" data-aos-delay="400">Unlocking profound insights into mental
health through NLP techniques.</h2>
          <div data-aos="fade-up" data-aos-delay="600">
            <div class="text-center text-lg-start">
              <a href="#about" class="btn-get-started scrollto d-inline-flex align-items-center justify-
content-center align-self-center">
                <span>Get Started</span>
                <i class="bi bi-arrow-right"></i>
              </a>
            </div>
          </div>
        </div>
```

```
      <div class="col-lg-6 hero-img" data-aos="zoom-out" data-aos-delay="200">
        <img src="static/img/hero-img.jpg" class="img-fluid" alt="">
      </div>
    </div>
  </div>
 </section><!-- End Hero -->
 <main id="main">
  <!-- ======= About Section ======= -->
  <section id="about" class="about">
   <div class="container-fluid" data-aos="fade-up">
    <div class="row gx-0">
     <div class="section-title" style="text-align: center;">
     <h2>About</h2>
    </div>
     <div class="col-lg-6 d-flex flex-column justify-content-center" data-aos="fade-up" data-aos-delay="200">
       <div class="content">
       <h2>Description</h2>
        <p>
      Mental health refers to a person's psychological, emotional, and social well-being. It encompasses various aspects of life, including how individuals think, feel, and behave, as well as how they handle stress, relate to others, and make choices. Mental health is crucial for overall well-being and affects every aspect of a person's life, including their relationships, work, and daily functioning.          </p>
      <p>Mental health analysis involves methodologies, behavioral observations, and innovative technologies like Natural Language Processing (NLP), to gain insights into an individual's mental well-being. By examining linguistic patterns, sentiment analysis, and other linguistic markers present in written or spoken communication, NLP can aid in identifying potential mental health issues, detecting changes in mood or thought patterns, and offering personalized interventions and support. This analytical approach not only helps in early detection and intervention but also facilitates the development of tailored treatment plans and enhances overall mental health care delivery.
</p>
       <div class="text-center text-lg-start">
        <a href="#" class="btn-read-more d-inline-flex align-items-center justify-content-center align-self-center">
          <span>Read More</span>
          <i class="bi bi-arrow-right"></i>
        </a>
       </div>
      </div>
     </div>
```

```
        <div class="col-lg-6 d-flex align-items-center" data-aos="zoom-out" data-aos-
delay="200">
          <img src="https://wallpapers.com/images/hd/health-pictures-ux6qxaoxuytyu82u.jpg"
class="img-fluid" alt="">
        </div>
       </div>
     </div>
   </section><!-- End About Section -->
   <!-- ======= Counts Section ======= -->
   <!-- ======= Features Section ======= -->
   <section id="features" class="features">
     <div class="container" data-aos="fade-up">
       <header class="section-header">
         <div class="section-title" style="text-align: center;">
       <h2 style="color: dark blue; font-weight: bold;font-size: 24px">Objective</h2>
   </div>
       </header>
       <div class="row">
         <div class="col-lg-6">
           <img src="static/img/features.png" class="img-fluid" alt="">
         </div>
<div class="col-lg-6 d-flex flex-column justify-content-center" data-aos="fade-up" data-aos-
delay="200">
          <div class="content">
            <p>Mental health analysis utilizing a binary classification system, where 1 denotes
comments indicative of mental health issues and 0 signifies normalcy or absence of such
concerns, is a vital tool in identifying and addressing psychological distress within textual data.
Through advanced techniques like Natural Language Processing (NLP), text mining, and
sentiment analysis, this approach enables the automated detection of linguistic markers
associated with mental health conditions such as depression, anxiety, or suicidal ideation. </p>
<p> By leveraging Natural Language Processing (NLP) and sentiment analysis, we aim to
identify linguistic patterns, semantic cues, and sentiment markers indicative of mental health
concerns within textual data. The ultimate goal is to provide a scalable and efficient tool for early
detection and intervention in mental health issues based on digital communication, enabling
timely support and resources for individuals at risk.</p>
            <div class="text-center text-lg-start">
            </div>
          </div>
        </div>
          </div>
        </div>
       </div> <!-- / row -->
   <!-- ======= F.A.Q Section ======= -->
```

```
<!-- ======= Portfolio Section ======= -->
<!-- ======= Contact Section ======= -->
<section id="contact" class="contact">
  <div class="container" data-aos="fade-up">
    <header class="section-header">
      <h2>Contact</h2>
      <p>Contact Us</p>
    </header>
    <div class="row gy-4">
      <div class="col-lg-6">
        <div class="row gy-4">
          <div class="col-md-6">
            <div class="info-box">
              <i class="bi bi-geo-alt"></i>
              <h3>Address</h3>
              <p>Smartbridge<br>Hyderabad 123123</p>
            </div>
          </div>
          <div class="col-md-6">
            <div class="info-box">
              <i class="bi bi-telephone"></i>
              <h3>Call Us</h3>
              <p>+1 12341243<br>+1 12341234</p>
            </div>
          </div>
          <div class="col-md-6">
            <div class="info-box">
              <i class="bi bi-envelope"></i>
              <h3>Email Us</h3>
              <p>info@example.com<br>contact@example.com</p>
            </div>
          </div>
          <div class="col-md-6">
            <div class="info-box">
              <i class="bi bi-clock"></i>
              <h3>Open Hours</h3>
              <p>Monday - Friday<br>9:00AM - 05:00PM</p>
            </div>
          </div>
        </div>
      </div>
      <div class="col-lg-6">
        <form action="forms/contact.php" method="post" class="php-email-form">
```

```
          <div class="row gy-4">
            <div class="col-md-6">
              <input type="text" name="name" class="form-control" placeholder="Your Name"
required>
            </div>
            <div class="col-md-6 ">
              <input type="email" class="form-control" name="email" placeholder="Your Email"
required>
            </div>
            <div class="col-md-12">
              <input type="text" class="form-control" name="subject" placeholder="Subject"
required>
            </div>
            <div class="col-md-12">
              <textarea class="form-control" name="message" rows="6" placeholder="Message"
required></textarea>
            </div>
            <div class="col-md-12 text-center">
              <div class="loading">Loading</div>
              <div class="error-message"></div>
              <div class="sent-message">Your message has been sent. Thank you!</div>
              <button type="submit">Send Message</button>
            </div>
          </div>
        </form>
      </div>
    </div>
  </div>
 </section><!-- End Contact Section -->
</main><!-- End #main -->
<!-- ======= Footer ======= -->
<footer id="footer" class="footer">
 <div class="footer-newsletter">
  <div class="container">
   <div class="row justify-content-center">
   </div>
  </div>
 </div>
 <div class="footer-top">
  <div class="container">
   <div class="row gy-4">
    <div class="col-lg-5 col-md-12 footer-info">
   </div>
```

```
    </div>
   </div>
   <div class="container">
    <div class="copyright">
     &copy; Copyright <strong><span>MentalHealth</span></strong>. All Rights Reserved
    </div>
    <div class="credits">
     <!-- All the links in the footer should remain intact. -->
     <!-- You can delete the links only if you purchased the pro version. -->
     <!-- Licensing information: https://bootstrapmade.com/license/ -->
     <!-- Purchase the pro version with working PHP/AJAX contact form:
https://bootstrapmade.com/flexstart-bootstrap-startup-template/ -->
     Developed by <a >Anusha</a>
    </div>
   </div>
  </footer><!-- End Footer -->
  <a href="#" class="back-to-top d-flex align-items-center justify-content-center"><i class="bi
bi-arrow-up-short"></i></a>
  <!-- Vendor JS Files -->
  <script src="static/vendor/purecounter/purecounter_vanilla.js"></script>
  <script src="static/vendor/aos/aos.js"></script>
  <script src="static/vendor/bootstrap/js/bootstrap.bundle.min.js"></script>
  <script src="static/vendor/glightbox/js/glightbox.min.js"></script>
  <script src="static/vendor/isotope-layout/isotope.pkgd.min.js"></script>
  <script src="static/vendor/swiper/swiper-bundle.min.js"></script>
  <script src="static/vendor/php-email-form/validate.js"></script>
  <!-- Template Main JS File -->
  <script src="static/js/main.js"></script>
</body>
</html>
```

Input.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <title>Welcome</title>
    <meta charset="utf-8">
    <meta content="width=device-width, initial-scale=1.0" name="viewport">
    <title>Welcome</title>
    <meta content="" name="description">
    <meta content="" name="keywords">
    <!-- Favicons -->
```

```
    <link href="static/img/favicon.png" rel="icon">
    <link href="static/img/apple-touch-icon.png" rel="apple-touch-icon">
    <!-- Google Fonts -->
    <link
href="https://fonts.googleapis.com/css?family=Open+Sans:300,300i,400,400i,600,600i,700,700i|
Nunito:300,300i,400,400i,600,600i,700,700i|Poppins:300,300i,400,400i,500,500i,600,600i,700,7
00i" rel="stylesheet">
    <!-- Vendor CSS Files -->
    <link href="static/vendor/aos/aos.css" rel="stylesheet">
    <link href="static/vendor/bootstrap/css/bootstrap.min.css" rel="stylesheet">
    <link href="static/vendor/bootstrap-icons/bootstrap-icons.css" rel="stylesheet">
    <link href="static/vendor/glightbox/css/glightbox.min.css" rel="stylesheet">
    <link href="static/vendor/remixicon/remixicon.css" rel="stylesheet">
    <link href="static/vendor/swiper/swiper-bundle.min.css" rel="stylesheet">
    <!-- Template Main CSS File -->
    <link href="static/css/style.css" rel="stylesheet">
   <!-- Your CSS styles -->
   <style>
      /* Center-align the container */
      .container {
         text-align: center;
         margin-top: 20%; /* Adjust the margin-top as needed */
      }
      /* Style inputs with type="text", select elements and textareas */
      input[type=text], input[type=number], select, textarea {
         width: 100%; /* Full width */
         padding: 12px; /* Some padding */
         border: 1px solid #ccc; /* Gray border */
         border-radius: 4px; /* Rounded borders */
         box-sizing: border-box; /* Make sure that padding and width stays in place */
         margin-top: 6px; /* Add a top margin */
         margin-bottom: 16px; /* Bottom margin */
         resize: vertical; /* Allow the user to vertically resize the textarea (not horizontally) */
      }
      /* Style the submit button with a specific background color etc */
      input[type=submit] {
         background-color: #04AA6D;
         color: white;
         padding: 12px 20px;
         border: none;
         border-radius: 10px;
         cursor: pointer;
      }
```

```
        /* When moving the mouse over the submit button, add a darker green color */
        input[type=submit]:hover {
            background-color: #45a049;
        }
    </style>
</head>
<body>
 <section id="hero" class="hero d-flex align-items-center">
<header id="header" class="header fixed-top">
  <div class="container-fluid container-xl d-flex align-items-center justify-content-between">
    <a href="index.html" class="logo d-flex align-items-center">
      <img src="static/img/logo.png" alt="">
      <span>MindScope</span>
    </a>
    <nav id="navbar" class="navbar">
      <ul>
        <li><a class="nav-link scrollto active" href="/">Return to Home Page</a></li>
      </ul>
      <i class="bi bi-list mobile-nav-toggle"></i>
    </nav><!-- .navbar -->
  </div>
</header>
<div class="container">
    <form action="/submit" method="post">
        <label for="textInput"><b>Enter your text<b></label>
        <input type="text" id="textInput" name="userInput" class="form-input" placeholder="Type
something...">
        <input type="submit" value="Submit">
        <br>
            <br>
        <div class="row">
            <div class="col-lg-6">
                <img src="static/img/word.jpg" class="img-fluid" alt="Left Image">
            </div>
            <div class="col-lg-6">
                <img src="static/img/word2.jpg" class="img-fluid" alt="Right Image">
            </div>
        </div>
    </form>
</div>
</body>
</html>
```

Output.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <meta content="width=device-width, initial-scale=1.0" name="viewport">
  <title>Portfolio Details - Harmony Insights Unleashing the Power of NLP Analysis</title>
  <meta content="" name="description">
  <meta content="" name="keywords">
  <!-- Favicons -->
  <link href="static/assets/img/favicon.png" rel="icon">
  <link href="static/assets/img/apple-touch-icon.png" rel="apple-touch-icon">
  <!-- Google Fonts -->
  <link
href="https://fonts.googleapis.com/css?family=Open+Sans:300,300i,400,400i,600,600i,700,700i|
Raleway:300,300i,400,400i,500,500i,600,600i,700,700i|Poppins:300,300i,400,400i,500,500i,600
,600i,700,700i" rel="stylesheet">
  <!-- Vendor CSS Files -->
  <link href="static/assets/vendor/animate.css/animate.min.css" rel="stylesheet">
  <link href="static/assets/vendor/bootstrap/css/bootstrap.min.css" rel="stylesheet">
  <link href="static/assets/vendor/bootstrap-icons/bootstrap-icons.css" rel="stylesheet">
  <link href="static/assets/vendor/boxicons/css/boxicons.min.css" rel="stylesheet">
  <link href="static/assets/vendor/glightbox/css/glightbox.min.css" rel="stylesheet">
  <link href="static/assets/vendor/remixicon/remixicon.css" rel="stylesheet">
  <link href="static/assets/vendor/swiper/swiper-bundle.min.css" rel="stylesheet">
  <!-- Template Main CSS File -->
  <link href="static/assets/css/style.css" rel="stylesheet">
  <!-- Additional CSS for Background Image -->
  <style>
   body {
     background-image: url('static/img/result.jpg');
     background-size: cover;
     background-position: center;
   }
     input[type=submit] {
         background-color: #04AA6D;
         color: white;
         padding: 12px 20px;
         border: none;
         border-radius: 10px;
```

```
        cursor: pointer;
    }
  </style>
</head>
<body>
  <br>
  <center><b class="pd"><font color="#0047AB" size="15" font-family="Comic Sans MS"
>Unlocking the Minds: Analyzing Mental Health with NLP</font></b></center><br><br>
  <div>
    <center>
      <h1><font color="black"> {{prediction_text}} </h1>
      <div class="col-lg-12">
        <img src="static/img/word3.jpg" class="img-fluid" alt="Right Image">
      </div>
      <h3><font color="#0047AB">Leveraging NLP for analyzing mental health holds significant
promise for advancing our understanding, detection, and treatment of mental health issues. By
harnessing the power of computational linguistics and data analytics, we can strive towards more
effective interventions, personalized care, and better support for individuals experiencing mental
health challenges.</h3>
    <form action="/" method="get">
    <input type="submit" class="nav-link scrollto active" value="Return to Home Page">
</form>
    </center>
  </div>
  <!-- Footer and other content remain unchanged -->
</body>
</html>
```

App.py:

```
import joblib
import os
from flask import Flask, request, render_template
import re
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
# Uncomment these lines if you need to download NLTK resources
# import nltk
# nltk.download('punkt')
# nltk.download('stopwords')
# nltk.download('wordnet')
app = Flask(__name__)
```

```
# Load the model and vectorizer
try:
    with open('model.pkl', 'rb') as f:
        model = joblib.load(f)
    with open('TfidfVectorizer.pkl', 'rb') as f:
        tfidf_vectorizer = joblib.load(f)
    print("Model and vectorizer loaded successfully!")
except Exception as e:
    model, tfidf_vectorizer = None, None
    print(f"Error during loading: {e}")
@app.route('/')
def home():
    return render_template('index.html')
@app.route('/predict', methods=["POST", "GET"])
def predict():
    return render_template("input.html")
@app.route('/submit', methods=["POST"])
def submit():
    try:
        # Read the user input
        text = request.form['userInput']
        if not text:
            return render_template('output.html', prediction_text="Input text is empty. Please enter
some text.")
        # Clean the input text
        text = re.sub('[^a-zA-Z0-9]+', " ", text)
        # Tokenization, stopword removal, and lemmatization
        tokens = word_tokenize(text)
        stop_words = set(stopwords.words('english'))
        filtered_tokens = [word for word in tokens if word.lower() not in stop_words]
        lemmatizer = WordNetLemmatizer()
        lemmatized_tokens = [lemmatizer.lemmatize(token) for token in filtered_tokens]
        preprocess_text = ' '.join(lemmatized_tokens)
        # Transform the preprocessed text using the TF-IDF vectorizer
        text_vectorized = tfidf_vectorizer.transform([preprocess_text])
        # Print to check if vectorization is successful
        print("Vectorized text shape:", text_vectorized.shape)
        # Make predictions using the model
        prediction = model.predict(text_vectorized)[0]
        # Map prediction to a label
        label = "Positive" if prediction == 1 else "Negative"
        return render_template('output.html', prediction_text=f'After Analysis, State of Mind was
found: {label}')
```

```
    except AttributeError as e:
        print("AttributeError:", e)
        return render_template('output.html', prediction_text="AttributeError: The model or
vectorizer is not properly loaded.")
    except Exception as e:
        print(f"An error occurred: {e}")
        return render_template('output.html', prediction_text="An unexpected error occurred.
Please check the console for more details.")
if __name__ == "__main__":
    port = int(os.environ.get('PORT', 5000))
    app.run(debug=True)
```

10.2. GitHub & Project Demo Link

Github:

https://github.com/dasi-anusha26/Unlocking-the-Minds-Analyzing-Mental-Health-with-NLP

Project Demo Link:

https://drive.google.com/file/d/1ZHRgSTqojstJ3fxvaZ5edk_gYEawEOkX/view?usp=sharing