MENTAL HEALTH AND DEPRESSION

Subtitle: A Comparison of Logistic Regression and Naive Bayes Models

Cheeno 1103558
Mlandvo 1113523
Rey 1111549
Zithile 1113552

INTRODUCTION

- Objective: Predict post if related to depression using machine learning.
- Focus: Compare Logistic Regression vs. Naive Bayes.
- Importance: Identifying depression accurately is critical in mental health applications.

MENTAL HEALTH & DEPRESSION

- **Mental health:**

A person’s emotional, psychological, and social well-being—how they think, feel, and handle life.

- **Depression:**

A mental health condition that causes persistent sadness, loss of interest, and affects daily functioning.

BRIEF OVERVIEW

Depression can be difficult to detect and identify, a large number of people suffer on a global scale from depression. However due to it’s difficulty to properly diagnose and detect it could be considered an invisible illness

Note:
This project's goal doesn't assess any user if they are clinically "depressed" or not, nor should it be used as an alternative to professional care

OUR SOLUTION TO THE INVISIBLE ILLNESS

We have devised two alogirthms to potentially detect signs of depression based on speech patterns and collected data based on what frequently used words those with depression use

Detection judged from:

- **Prediction**
- **Confidence**

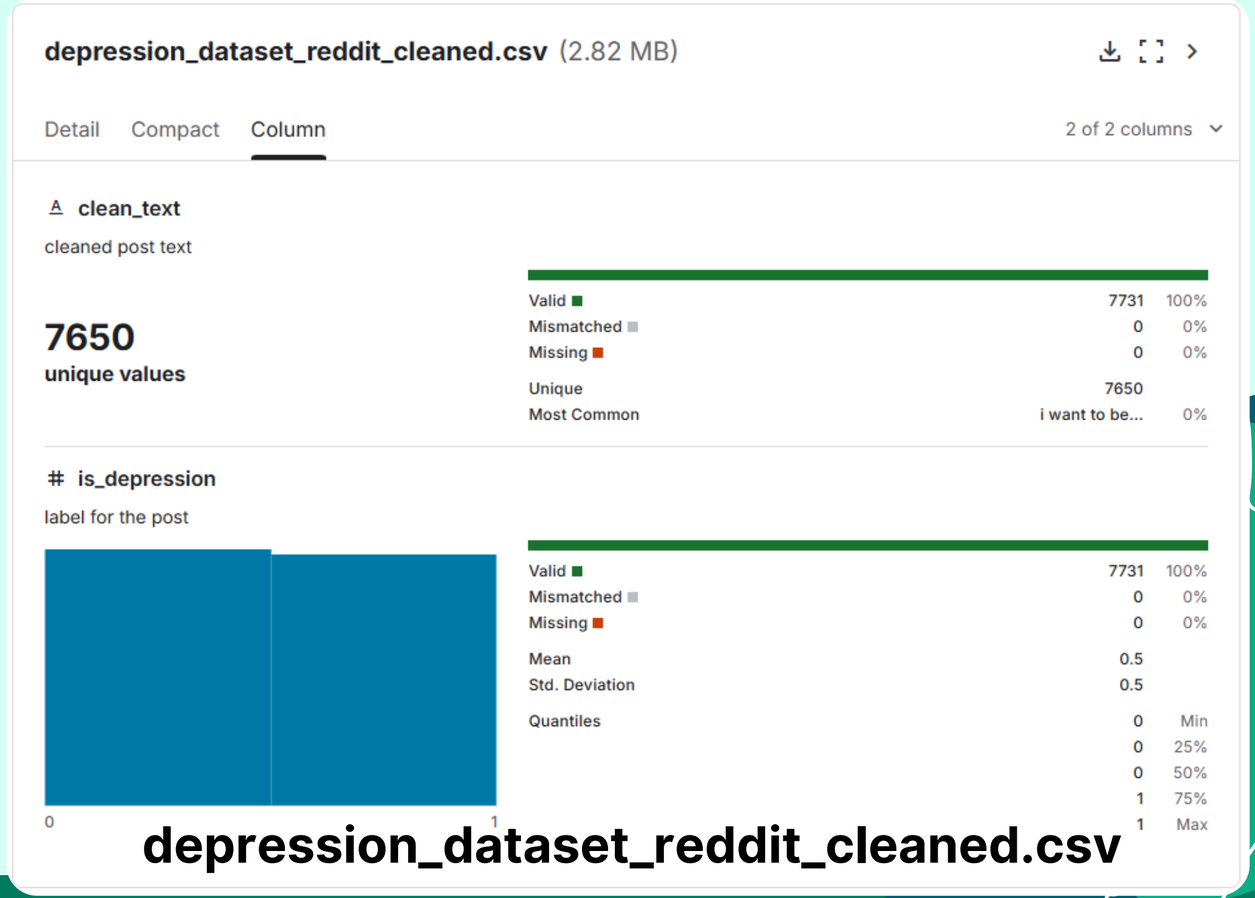
IMPLEMENTATION

Import Required Libraries:

IMPLEMENTATION

Load the dataset

- Our dataset was extracted from reddit.
- The raw data is collected through web scrapping Subreddits and is cleaned using multiple NLP techniques.
- The data is only in English language.
- It mainly targets mental health classification.
- 7,731 posts



depression_dataset_reddit_cleaned.csv

IMPLEMENTATION

Data Cleaning & Preprocessing

1. Remove posts with less than 5 words. (These are not typically useful for our project goal.)

```python
# Initialize tools
stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()

# POS tag mapper
# This is important for the lemmatizer to work properly.
def get_wordnet_pos(tag):
    if tag.startswith('J'):
        return wordnet.ADJ
    elif tag.startswith('V'):
        return wordnet.VERB
    elif tag.startswith('N'):
        return wordnet.NOUN
    elif tag.startswith('R'):
        return wordnet.ADV
    else:
        return wordnet.NOUN  # default

# Preprocessing function
def preprocess_text_better(text):
    text = text.lower()
    text = re.sub(r"[^a-zA-Z\s]", "", text)
    tokens = nltk.word_tokenize(text)
    tokens = [word for word in tokens if word not in stop_words]
    pos_tags = pos_tag(tokens)
    lemmatized = [lemmatizer.lemmatize(word, get_wordnet_pos(pos)) for word, pos in pos_tags]
    return " ".join(lemmatized)

# Remove posts with fewer than 5 words
df['word_count'] = df['clean_text'].apply(lambda x: len(x.split()))
df = df[df['word_count'] >= 5]

# Remove duplicate posts
df = df.drop_duplicates(subset='clean_text')

# Apply preprocessing
df['processed_text'] = df['clean_text'].apply(preprocess_text_better)

# Save as CSV File
df[['clean_text', 'processed_text', 'is_depression']].to_csv("reddit_processed_depression_data.csv", index=False)

print("Original dataset size: 7731")
print("Final dataset size after filtering:", df.shape[0])


df[['clean_text', 'processed_text']].head()
```

IMPLEMENTATION

Data Cleaning & Preprocessing

1. Remove posts with less than 5 words. (These are not typically useful for our project goal.)
2. Tokenization (Splitting text into words)

```python
# Initialize tools
stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()

# POS tag mapper
# This is important for the lemmatizer to work properly.
def get_wordnet_pos(tag):
    if tag.startswith('J'):
        return wordnet.ADJ
    elif tag.startswith('V'):
        return wordnet.VERB
    elif tag.startswith('N'):
        return wordnet.NOUN
    elif tag.startswith('R'):
        return wordnet.ADV
    else:
        return wordnet.NOUN  # default

# Preprocessing function
def preprocess_text_better(text):
    text = text.lower()
    text = re.sub(r"[^a-zA-Z\s]", "", text)
    tokens = nltk.word_tokenize(text)
    tokens = [word for word in tokens if word not in stop_words]
    pos_tags = pos_tag(tokens)
    lemmatized = [lemmatizer.lemmatize(word, get_wordnet_pos(pos)) for word, pos in pos_tags]
    return " ".join(lemmatized)

# Remove posts with fewer than 5 words
df['word_count'] = df['clean_text'].apply(lambda x: len(x.split()))
df = df[df['word_count'] >= 5]

# Remove duplicate posts
df = df.drop_duplicates(subset='clean_text')

# Apply preprocessing
df['processed_text'] = df['clean_text'].apply(preprocess_text_better)

# Save as CSV File
df[['clean_text', 'processed_text', 'is_depression']].to_csv("reddit_processed_depression_data.csv", index=False)

print("Original dataset size: 7731")
print("Final dataset size after filtering:", df.shape[0])


df[['clean_text', 'processed_text']].head()
```

IMPLEMENTATION

Data Cleaning & Preprocessing

1. Remove posts with less than 5 words. (These are not typically useful for our project goal.)
2. Tokenization (Splitting text into words)
3. Removed duplicates

```python
# Initialize tools
stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()

# POS tag mapper
# This is important for the lemmatizer to work properly.
def get_wordnet_pos(tag):
    if tag.startswith('J'):
        return wordnet.ADJ
    elif tag.startswith('V'):
        return wordnet.VERB
    elif tag.startswith('N'):
        return wordnet.NOUN
    elif tag.startswith('R'):
        return wordnet.ADV
    else:
        return wordnet.NOUN  # default

# Preprocessing function
def preprocess_text_better(text):
    text = text.lower()
    text = re.sub(r"[^a-zA-Z\s]", "", text)
    tokens = nltk.word_tokenize(text)
    tokens = [word for word in tokens if word not in stop_words]
    pos_tags = pos_tag(tokens)
    lemmatized = [lemmatizer.lemmatize(word, get_wordnet_pos(pos)) for word, pos in pos_tags]
    return " ".join(lemmatized)

# Remove posts with fewer than 5 words
df['word_count'] = df['clean_text'].apply(lambda x: len(x.split()))
df = df[df['word_count'] >= 5]

# Remove duplicate posts
df = df.drop_duplicates(subset='clean_text')

# Apply preprocessing
df['processed_text'] = df['clean_text'].apply(preprocess_text_better)

# Save as CSV File
df[['clean_text', 'processed_text', 'is_depression']].to_csv("reddit_processed_depression_data.csv", index=False)

print("Original dataset size: 7731")
print("Final dataset size after filtering:", df.shape[0])


df[['clean_text', 'processed_text']].head()
```

IMPLEMENTATION

Data Cleaning & Preprocessing

1. Remove posts with less than 5 words. (These are not typically useful for our project goal.)
2. Tokenization (Splitting text into words)
3. Removed duplicates
4. Convert to lowercase

```python
# Initialize tools
stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()

# POS tag mapper
# This is important for the lemmatizer to work properly.
def get_wordnet_pos(tag):
    if tag.startswith('J'):
        return wordnet.ADJ
    elif tag.startswith('V'):
        return wordnet.VERB
    elif tag.startswith('N'):
        return wordnet.NOUN
    elif tag.startswith('R'):
        return wordnet.ADV
    else:
        return wordnet.NOUN  # default

# Preprocessing function
def preprocess_text_better(text):
    text = text.lower()
    text = re.sub(r"[^a-zA-Z\s]", "", text)
    tokens = nltk.word_tokenize(text)
    tokens = [word for word in tokens if word not in stop_words]
    pos_tags = pos_tag(tokens)
    lemmatized = [lemmatizer.lemmatize(word, get_wordnet_pos(pos)) for word, pos in pos_tags]
    return " ".join(lemmatized)

# Remove posts with fewer than 5 words
df['word_count'] = df['clean_text'].apply(lambda x: len(x.split()))
df = df[df['word_count'] >= 5]

# Remove duplicate posts
df = df.drop_duplicates(subset='clean_text')

# Apply preprocessing
df['processed_text'] = df['clean_text'].apply(preprocess_text_better)

# Save as CSV File
df[['clean_text', 'processed_text', 'is_depression']].to_csv("reddit_processed_depression_data.csv", index=False)

print("Original dataset size: 7731")
print("Final dataset size after filtering:", df.shape[0])


df[['clean_text', 'processed_text']].head()
```

IMPLEMENTATION

Data Cleaning & Preprocessing

1. Remove posts with less than 5 words. (These are not typically useful for our project goal.)
2. Tokenization (Splitting text into words)
3. Removed duplicates
4. Convert to lowercase
5. Remove punctuation, numbers & symbols

```python
# Initialize tools
stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()

# POS tag mapper
# This is important for the lemmatizer to work properly.
def get_wordnet_pos(tag):
    if tag.startswith('J'):
        return wordnet.ADJ
    elif tag.startswith('V'):
        return wordnet.VERB
    elif tag.startswith('N'):
        return wordnet.NOUN
    elif tag.startswith('R'):
        return wordnet.ADV
    else:
        return wordnet.NOUN  # default

# Preprocessing function
def preprocess_text_better(text):
    text = text.lower()
    text = re.sub(r"[^a-zA-Z\s]", "", text)
    tokens = nltk.word_tokenize(text)
    tokens = [word for word in tokens if word not in stop_words]
    pos_tags = pos_tag(tokens)
    lemmatized = [lemmatizer.lemmatize(word, get_wordnet_pos(pos)) for word, pos in pos_tags]
    return " ".join(lemmatized)

# Remove posts with fewer than 5 words
df['word_count'] = df['clean_text'].apply(lambda x: len(x.split()))
df = df[df['word_count'] >= 5]

# Remove duplicate posts
df = df.drop_duplicates(subset='clean_text')

# Apply preprocessing
df['processed_text'] = df['clean_text'].apply(preprocess_text_better)

# Save as CSV File
df[['clean_text', 'processed_text', 'is_depression']].to_csv("reddit_processed_depression_data.csv", index=False)

print("Original dataset size: 7731")
print("Final dataset size after filtering:", df.shape[0])


df[['clean_text', 'processed_text']].head()
```

IMPLEMENTATION

Data Cleaning & Preprocessing

1. Remove posts with less than 5 words. (These are not typically useful for our project goal.)
2. Tokenization (Splitting text into words)
3. Removed duplicates
4. Convert to lowercase
5. Remove punctuation, numbers & symbols
6. Remove stopwords (“the” , “is” , “at”, etc.)

```
# Initialize tools
stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()

# POS tag mapper
# This is important for the lemmatizer to work properly.
def get_wordnet_pos(tag):
    if tag.startswith('J'):
        return wordnet.ADJ
    elif tag.startswith('V'):
        return wordnet.VERB
    elif tag.startswith('N'):
        return wordnet.NOUN
    elif tag.startswith('R'):
        return wordnet.ADV
    else:
        return wordnet.NOUN  # default

# Preprocessing function
def preprocess_text_better(text):
    text = text.lower()
    text = re.sub(r"[^a-zA-Z\s]", "", text)
    tokens = nltk.word_tokenize(text)
    tokens = [word for word in tokens if word not in stop_words]
    pos_tags = pos_tag(tokens)
    lemmatized = [lemmatizer.lemmatize(word, get_wordnet_pos(pos)) for word, pos in pos_tags]
    return " ".join(lemmatized)

# Remove posts with fewer than 5 words
df['word_count'] = df['clean_text'].apply(lambda x: len(x.split()))
df = df[df['word_count'] >= 5]

# Remove duplicate posts
df = df.drop_duplicates(subset='clean_text')

# Apply preprocessing
df['processed_text'] = df['clean_text'].apply(preprocess_text_better)

# Save as CSV File
df[['clean_text', 'processed_text', 'is_depression']].to_csv("reddit_processed_depression_data.csv", index=False)

print("Original dataset size: 7731")
print("Final dataset size after filtering:", df.shape[0])


df[['clean_text', 'processed_text']].head()
```

IMPLEMENTATION

Data Cleaning & Preprocessing

1. Remove posts with less than 5 words. (These are not typically useful for our project goal.)
2. Tokenization (Splitting text into words)
3. Removed duplicates
4. Convert to lowercase
5. Remove punctuation, numbers & symbols
6. Remove stopwords (“the” , “is” , “at”, etc.)
7. POS Tag mapper (required for Lemmatizer)

```python
# Initialize tools
stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()

# POS tag mapper
# This is important for the lemmatizer to work properly.
def get_wordnet_pos(tag):
    if tag.startswith('J'):
        return wordnet.ADJ
    elif tag.startswith('V'):
        return wordnet.VERB
    elif tag.startswith('N'):
        return wordnet.NOUN
    elif tag.startswith('R'):
        return wordnet.ADV
    else:
        return wordnet.NOUN  # default

# Preprocessing function
def preprocess_text_better(text):
    text = text.lower()
    text = re.sub(r"[^a-zA-Z\s]", "", text)
    tokens = nltk.word_tokenize(text)
    tokens = [word for word in tokens if word not in stop_words]
    pos_tags = pos_tag(tokens)
    lemmatized = [lemmatizer.lemmatize(word, get_wordnet_pos(pos)) for word, pos in pos_tags]
    return " ".join(lemmatized)

# Remove posts with fewer than 5 words
df['word_count'] = df['clean_text'].apply(lambda x: len(x.split()))
df = df[df['word_count'] >= 5]

# Remove duplicate posts
df = df.drop_duplicates(subset='clean_text')

# Apply preprocessing
df['processed_text'] = df['clean_text'].apply(preprocess_text_better)

# Save as CSV File
df[['clean_text', 'processed_text', 'is_depression']].to_csv("reddit_processed_depression_data.csv", index=False)

print("Original dataset size: 7731")
print("Final dataset size after filtering:", df.shape[0])


df[['clean_text', 'processed_text']].head()
```

IMPLEMENTATION

Data Cleaning & Preprocessing

1. Remove posts with less than 5 words. (These are not typically useful for our project goal.)
2. Tokenization (Splitting text into words)
3. Removed duplicates
4. Convert to lowercase
5. Remove punctuation, numbers & symbols
6. Remove stopwords (“the” , “is” , “at”, etc.)
7. POS Tag mapper (needed for Lemmatizer)
8. Lemmatizer

```python
# Initialize tools
stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()

# POS tag mapper
# This is important for the lemmatizer to work properly.
def get_wordnet_pos(tag):
    if tag.startswith('J'):
        return wordnet.ADJ
    elif tag.startswith('V'):
        return wordnet.VERB
    elif tag.startswith('N'):
        return wordnet.NOUN
    elif tag.startswith('R'):
        return wordnet.ADV
    else:
        return wordnet.NOUN  # default

# Preprocessing function
def preprocess_text_better(text):
    text = text.lower()
    text = re.sub(r"[^a-zA-Z\s]", "", text)
    tokens = nltk.word_tokenize(text)
    tokens = [word for word in tokens if word not in stop_words]
    pos_tags = pos_tag(tokens)
    lemmatized = [lemmatizer.lemmatize(word, get_wordnet_pos(pos)) for word, pos in pos_tags]
    return " ".join(lemmatized)

# Remove posts with fewer than 5 words
df['word_count'] = df['clean_text'].apply(lambda x: len(x.split()))
df = df[df['word_count'] >= 5]

# Remove duplicate posts
df = df.drop_duplicates(subset='clean_text')

# Apply preprocessing
df['processed_text'] = df['clean_text'].apply(preprocess_text_better)

# Save as CSV File
df[['clean_text', 'processed_text', 'is_depression']].to_csv("reddit_processed_depression_data.csv", index=False)

print("Original dataset size: 7731")
print("Final dataset size after filtering:", df.shape[0])


df[['clean_text', 'processed_text']].head()
```

IMPLEMENTATION

Data Cleaning & Preprocessing

1. Remove posts with less than 5 words. (These are not typically useful for our project goal.)
2. Tokenization (Splitting text into words)
3. Removed duplicates
4. Convert to lowercase
5. Remove punctuation, numbers & symbols
6. Remove stopwords (“the” , “is” , “at”, etc.)
7. POS Tag mapper (needed for Lemmatizer)
8. Lemmatizer

After preprocessing:

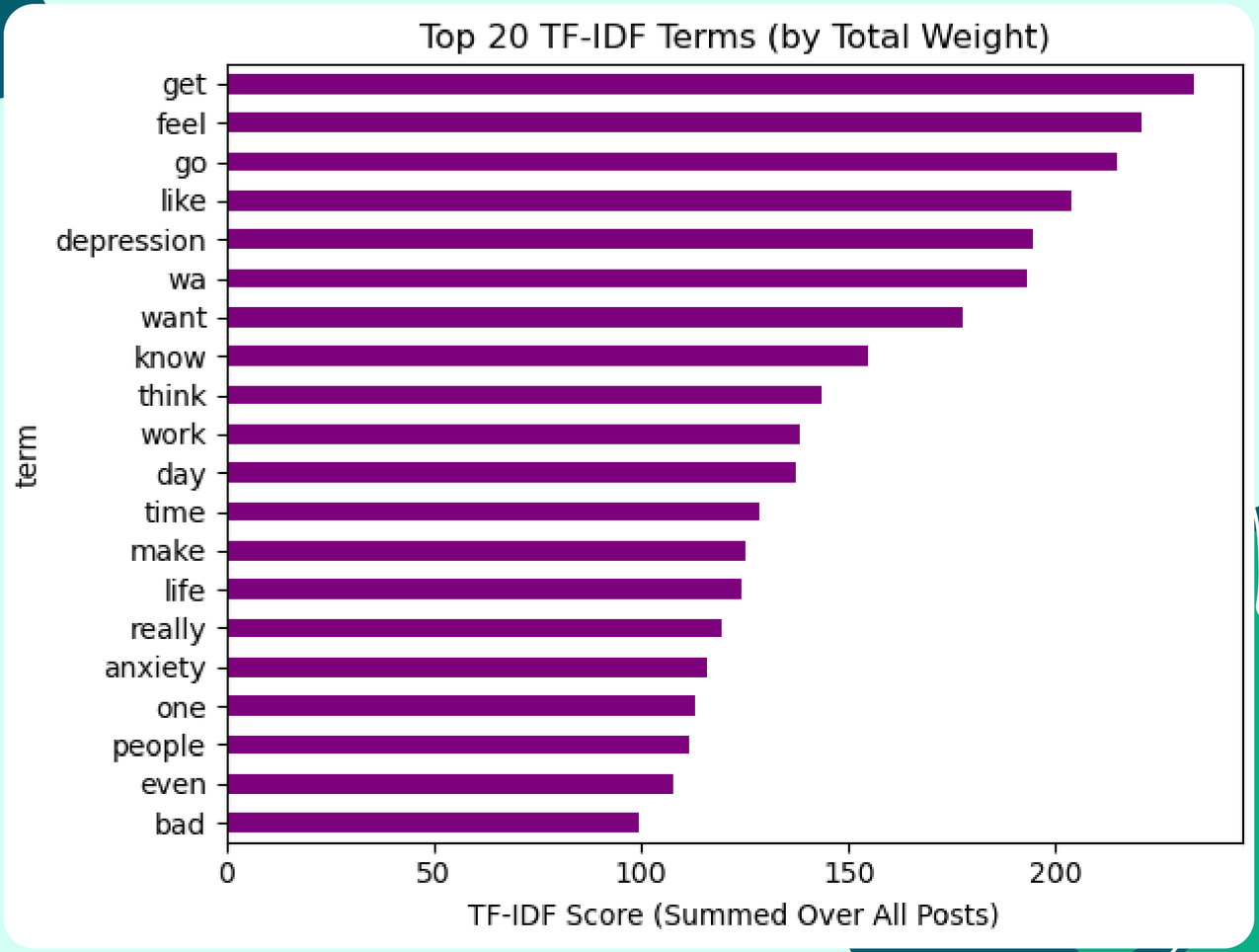
```
Original dataset size: 7731
Final dataset size after filtering: 7289
```

	clean_text	processed_text
0	we understand that most people who reply immed...	understand people reply immediately op invitat...
1	welcome to r depression s check in post a plac...	welcome r depression check post place take mom...
2	anyone else instead of sleeping more when depr...	anyone else instead sleep depressed stay night...
3	i ve kind of stuffed around a lot in my life d...	kind stuffed around lot life delay inevitable ...
4	sleep is my greatest and most comforting escap...	sleep great comfort escape whenever wake day l...

IMPLEMENTATION

Feature Extraction: TF-IDF Vectorizer

1. **TF** = How often a word appears in a post.
2. **IDF** = How rare that word is across all post
3. High TF-IDF Score means a word appears often in this post but rarely in others which implies that it is ***likely important***.
4. Numerical vectors->model can understand
5. Helps the model focus on emotionally significant language patterns.



IMPLEMENTATION

Split the data

```
# Data Split (train-test splitting)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Load the model: Logistic Regression

```
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

Load the model: Naive Bayes

Multinomial works well with discrete features

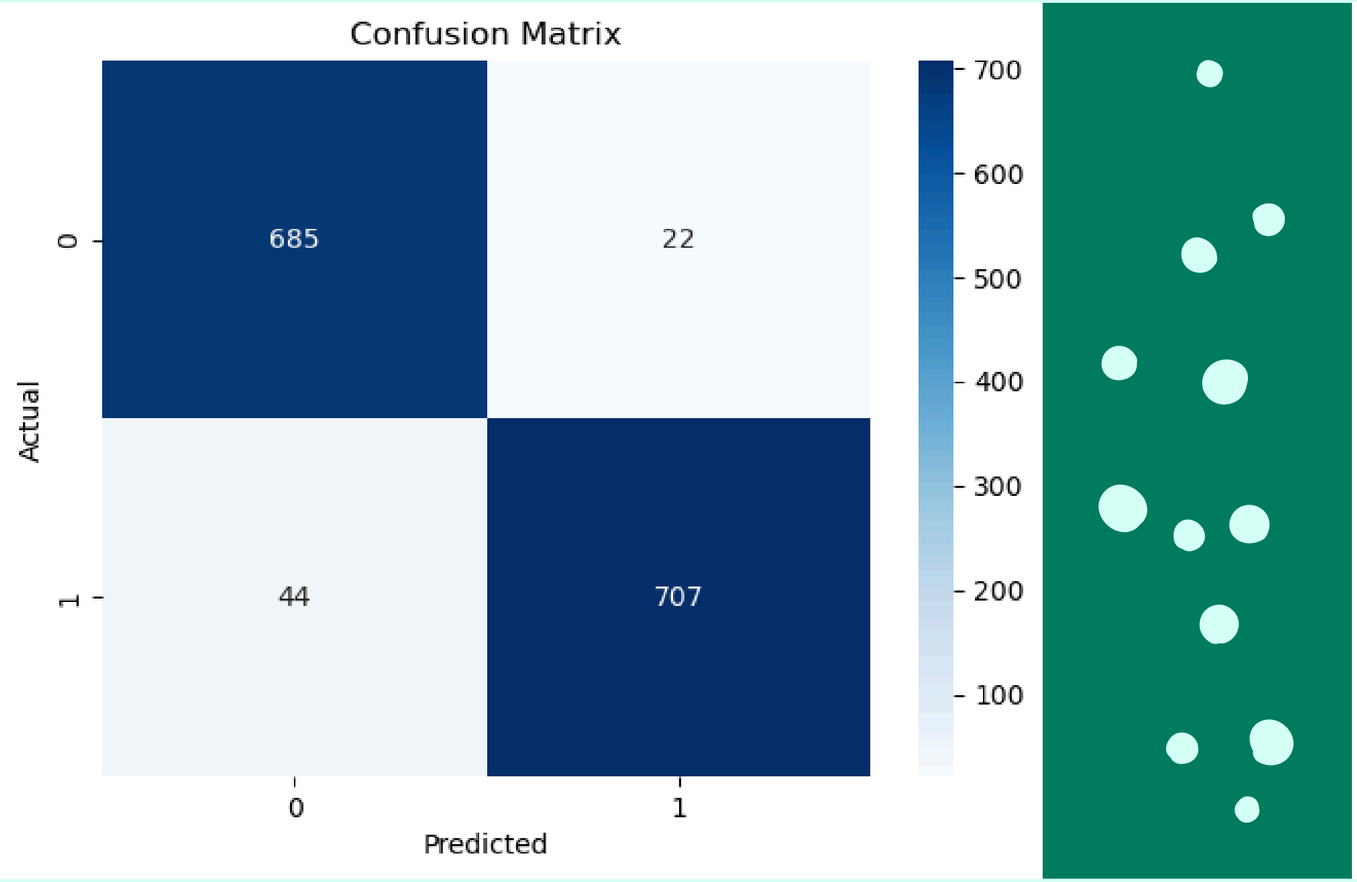
```
# Try a second model to compare performance

from sklearn.naive_bayes import MultinomialNB

nb_model = MultinomialNB()
nb_model.fit(X_train, y_train)
y_nb_pred = nb_model.predict(X_test)
```

IMPLEMENTATION

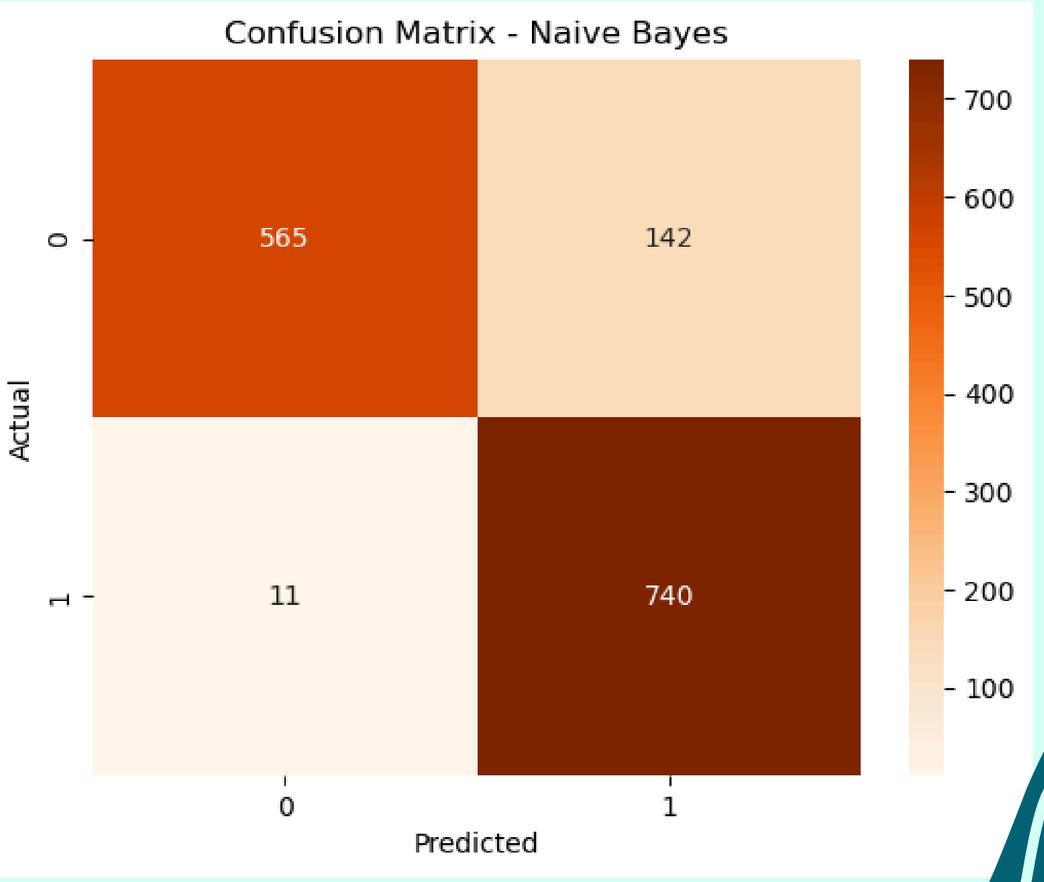
- Import required libraries
- Load the dataset
- Data Cleaning & Preprocessing
- Feature Extraction (TF-IDF Vectorizer)
- Split the data (train-test splitting)
- Train the model:
  - Logistic Regression
  - Naive Bayes (Multinomial)
- Analyze and Compare results for both models

RESULTS

Logistic Regression



Naive Bayes



PERFORMANCE COMPARISON TABLE

Metric	Logistic Regression.	Naive Bayes
Accuracy.	95.5%	89.5%
Precision (Class 1)	0.97	0.84
Recall (Class 1)	0.94	0.99
F1-score (Class 1)	0.96	0.90
False Negatives	44	11
False Positives	22	142

CONCLUSION

Logistic Regression offers higher accuracy and precision, making it better for general screening, Whilst, Naive Bayes achieves higher recall, reducing missed depression cases—valuable in clinical settings.

The choice depends on context:

- Precision-focused → Logistic Regression
- Recall-focused → Naive Bayes

Overall: Balancing precision and recall is key in mental health prediction.

DEMO TIME

THANK YOU