Sentiment Analysis for Mental Health Monitoring 🤯

> [dataset source](#) 📎

About The Datset 🧷

This comprehensive dataset is a meticulously curated collection of mental health statuses tagged from various statements. The dataset amalgamates raw data from multiple sources, cleaned and compiled to create a robust resource for developing chatbots and performing sentiment analysis.

Data Souce 🌍

The dataset integrates information from the following Kaggle datasets:

- 3k Conversations Dataset for Chatbot
- Depression Reddit Cleaned
- Human Stress Prediction
- Predicting Anxiety in Mental Health Data
- Mental Health Dataset Bipolar
- Reddit Mental Health Data
- Students Anxiety and Depression Dataset
- Suicidal Mental Health Dataset
- Suicidal Tweet Detection Dataset

Datset Over View :

- **Description:**

  This dataset is a comprehensive collection of 50,000 text statements related to mental health, each tagged with one of seven mental health statuses. The primary purpose of this dataset is to assist in building machine learning models for classifying mental health conditions based on textual data, such as social media posts or other user-generated content.

- **Columns:**
  1. **unique_id**: A unique identifier for each entry.
  2. **statement**: A piece of text, typically a statement or comment, associated with a particular mental health status.
  3. **status**: The mental health status assigned to the statement. The possible categories are:
     - Normal
     - Depression
     - Suicidal
     - Anxiety
     - Stress
     - Bi-Polar

- Personality Disorder

Usage :

This dataset is ideal for training machine learning models aimed at understanding and predicting mental health conditions based on textual data. It can be used in various applications such as:

- Chatbot development for mental health support.
- Sentiment analysis to gauge mental health trends.
-

How we use NLP Concepts

In our text processing methodology, we begin by removing punctuation, URLs, and hyperlinks from the statements. Additionally, we eliminate stop words, such as "is," "are," and "the," to enhance the focus on more informative terms.

In the realm of morphological analysis, we extract the stems of the words. Subsequently, we employ a TF-IDF (Term Frequency-Inverse Document Frequency) vectorizer to transform the processed text into a vector representation. This vector is then utilized to fit the mode

---

Conlusion

in the conclusion we train and tune a four model wich is :

- SVM
- Logistic Regression
- Neural Ntwork
- KNN the first three give a Convergent results with accuracy near to 76 % , and the last one give a 65 % acc .

Import Libraries :

lets start by importing the nessacary libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from nltk.corpus import stopwords
from imblearn.over_sampling import SMOTE

import re
import random
#from imblearn.over_sampling import RandomOverSampler
from scipy.sparse import hstack  # To combine sparse matrices
from wordcloud import WordCloud
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.layers import BatchNormalization
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from tensorflow.keras.regularizers import l2

from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report


import nltk
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import BernoulliNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

```
import warnings
warnings.filterwarnings("ignore")
```

```
import tensorflow as tf

# Check if GPU is available
print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
```

```
Num GPUs Available:  0
```

Loading the Dataset

```
df = pd.read_csv('/content/drive/MyDrive/CombinedData.csv', index_col=0)
```

```
df.head(3)
```

	statement	status
0	oh my gosh	Anxiety
1	trouble sleeping, confused mind, restless hear...	Anxiety
2	All wrong, back off dear, forward doubt. Stay ...	Anxiety

notes1: as we se at first , we have a three column the id which we should drop it, the statment that represent the "input" and the output which is the status. so our problem is just a classification and we need to make a nlp pre processing to extract feature as we can from the statment to make them as input to the model.

note2: as we say, also we wil add some statistical feature (test len) to nlp extracted features.

some random example :

- to see the statement how itis

```
# Group by status and get a random statement from each group
random_statements = df.groupby('status')['statement'].apply(lambda x: x.sample(n=1).iloc[0])

# Print the results
for status, statement in random_statements.items():
    print(f"Status: {status}\n")
    print(f"Statement: {statement}\n")
    print("-----------------------------------")
```

```
Status: Anxiety

Statement: Can abandoned/old syringe needles give you HIVs? * Age: 16
* Sex: Male
* Height: 5'4
* Weight: 100 lbs
* Race: American
* Duration of complaint: 1 day

Recently, was doing a small club cleanup and we've found like 6-7 syringe needles (sor

Can abandoned/old syringe needles give you HIVs? And is it only possible when injectec

-----------------------------------
Status: Bipolar

Statement: Recent experience with lamictal &amp; painkillers Hello friends.

I've been on lamictal for about 8 months. It was a godsend to me. Starting from a very

Here's the wrinkle: I initially spiraled into crippling depression and anxiety last ye

I was about 9 months post-op last year and doing really well, feeling really great abo

I have spent the last month carefully tapering off of oxycodone. I am now down to 25%

I guess I'm posting here because I feel frightened and a bit trapped. Last year, lamic

Typing this out, it seems unreasonable to expect anything from a small increase in my

1. Has anyone experienced a marked increase in depression or anxiety upon increasing t
2. Does anyone have experience with oxycodone / other opiate painkillers that impactec

For what it's worth, I also take Effexor XR 75 mg daily, Dexedrine SR 5 mg 2x/day, anc

-----------------------------------
Status: Depression

Statement: I am a 22 European female who started going to university last year. I am j

-----------------------------------
Status: Normal

Statement: In the morning, I want to talk to a foreign dog

-----------------------------------
```

```
Status: Personality disorder

Statement: What the hell do I do, lmao? I'm a mental cripple. I'm 25 years old, and I

----------------------------------
Status: Stress

Statement: I can't stop thinking and I need to vent!! Hi Reddit, I'm here to vent abou

During the pandemic, my concern was how to get myself out there when I was very limit
```

Data set Descriping and statistics

```
df.describe()
```

	statement	status
count	52681	53043
unique	51073	7
top	what do you mean?	Normal
freq	22	16351

Missings Value

```
df.isna().sum()
```

	0
statement	362
status	0

dtype: int64

```
df.dropna(inplace = True)
df.isna().sum()
```

	0
statement	0
status	0

dtype: int64

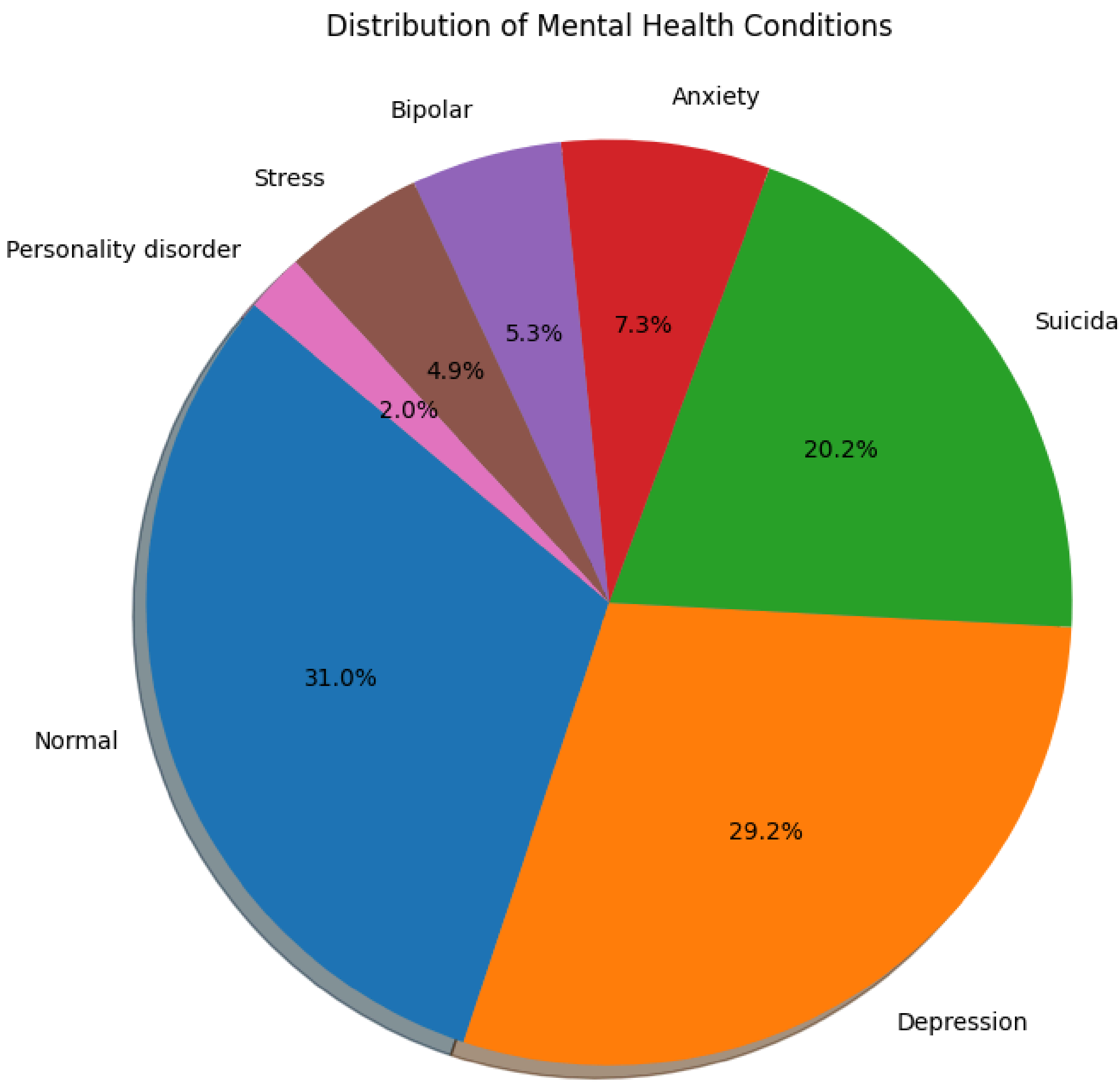
Labels

```
df['status'].unique()
```

```
array(['Anxiety', 'Normal', 'Depression', 'Suicidal', 'Stress', 'Bipolar',
       'Personality disorder'], dtype=object)
```

How the samples are distributed over the status ?

```
# Count the occurrences of each category
status_counts = df['status'].value_counts()

# Define colors for each category (7 colors for 7 categories)
colors = ['#1F77B4', '#FF7F0E', '#2CA02C', '#D62728', '#9467BD', '#8C564B', '#E377C2']

# Create the pie chart
plt.figure(figsize=(7, 7))
plt.pie(status_counts, labels=status_counts.index, autopct='%1.1f%%',
        startangle=140, colors=colors, shadow=True)

plt.title('Distribution of Mental Health Conditions')
plt.axis('equal')

# Display the chart
plt.tight_layout()
plt.show()
```

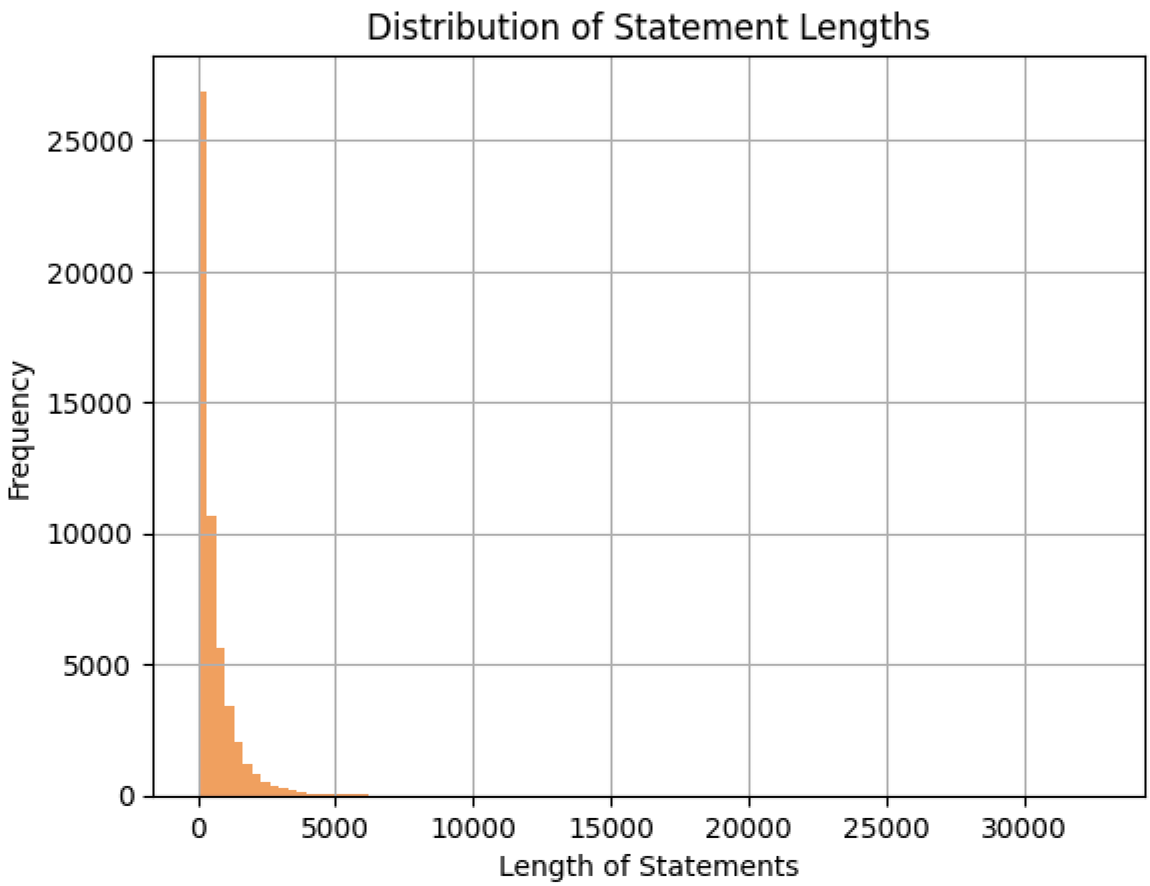


How the statements length are distibuted?

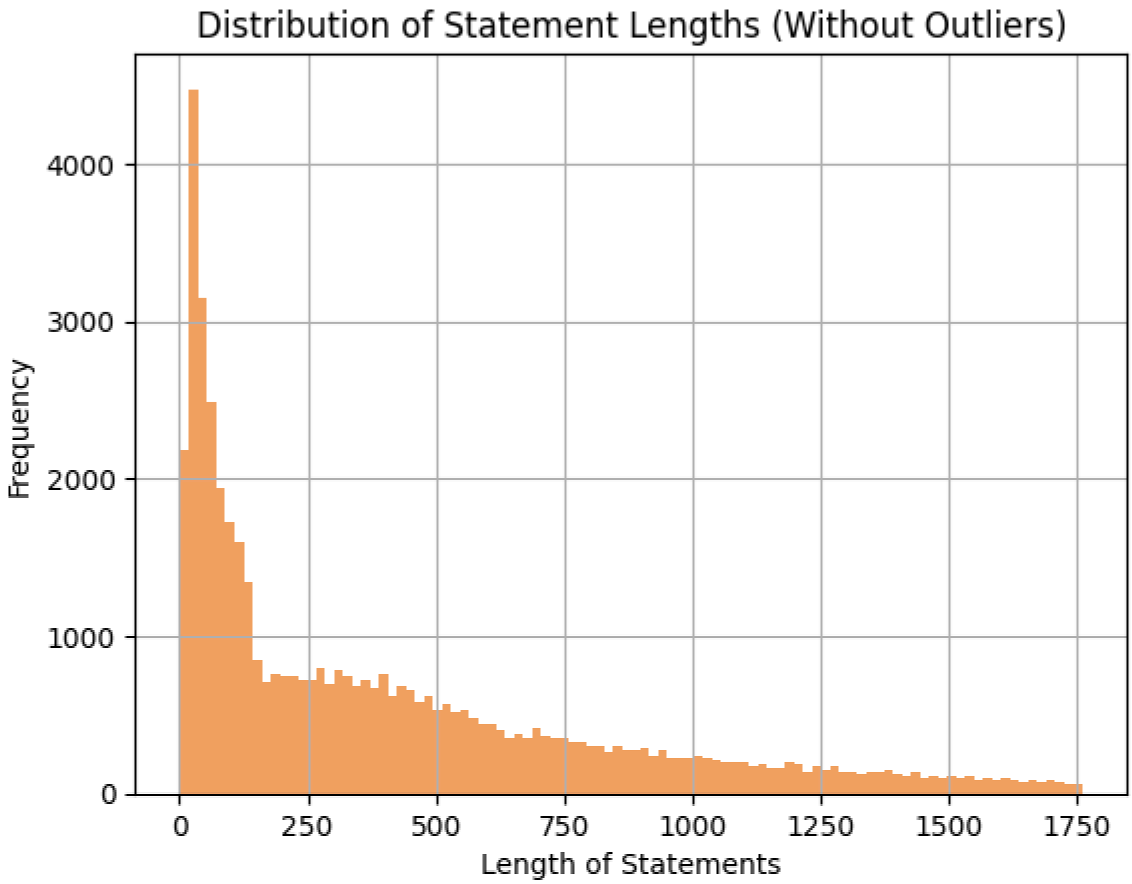
```
# calculate the length of each statement
df['statemnent_length'] = df['statement'].str.len()
```

```
# plot the distribution of statement lengths
df['statemnent_length'].hist(bins=100, color='#F4A261')
plt.title('Distribution of Statement Lengths')
plt.xlabel('Length of Statements')
```

```
plt.ylabel('Frequency')
plt.show()
```




```
# Calculate Q1 (25th percentile) and Q3 (75th percentile)
Q1 = df['statemnent_length'].quantile(0.25)
Q3 = df['statemnent_length'].quantile(0.75)
IQR = Q3 - Q1

# Define the lower and upper bound for outliers
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR


# Filter out the outliers
filtered_df = df[(df['statemnent_length'] >= lower_bound) & (df['statemnent_length'] <= uppe
# Plot the distribution of statement lengths without outliers
filtered_df['statemnent_length'].hist(bins=100, color='#F4A261')
plt.title('Distribution of Statement Lengths (Without Outliers)')
plt.xlabel('Length of Statements')
plt.ylabel('Frequency')
plt.show()
```




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

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
True
```

NLP Pre Processing

---

in this part of the file we do the following

1. lower casing and pattern reomving (links,emails and etc.)
2. tokinezation
3. stemming

1. Loweer Casing

```
df['statement']=df['statement'].str.lower()
df.sample(1)
```

	statement	status	statemnent_length
42475	seating here helping my baby with his paper we...	Normal	93

Special Patterns Removing

```
def remove_patterns(text):

    # remove URL's
    text = re.sub(r'http[s]?://\S+', '', text)

    # remove markdown-style links
    text = re.sub(r'\[.*?\]\(.*?\)', '', text)

    # remove handles (that start with '@')
    text = re.sub(r'@\w+', '', text)

    # remove punctuation and other special characters
    text = re.sub(r'[^\w\s]', '', text)
    return text.strip()

# apply the function to the statements
df['statement'] = df['statement'].apply(remove_patterns)
```

Remove Stop words

```
stop_words = set(stopwords.words('english'))

def remove_stopwords(text):
    words = text.split()
    filtered_words = [word for word in words if word not in stop_words]
    return " ".join(filtered_words)


df['statement'] = df['statement'].apply(remove_stopwords)
```

3.Tokenization

```
# Calculate the number of characters and sentences
df['num_of_sentences'] = df['statement'].apply(lambda x: len(nltk.sent_tokenize(x)))


# apply word tokenize to each element in the statements
df['tokens'] = df['statement'].apply(word_tokenize)

df.sample(5)
```

	statement	status	statemnent_length	num_of_sentences	tokens
39928	wait school counselor give update whole online...	Depression	1916	1	[wait, school, counselor, give, update, whole,...
5372	understand feel patient another level bullshit	Normal	68	1	[understand, feel, patient, another, level, bu...
	nothing seems				

4. Stemming

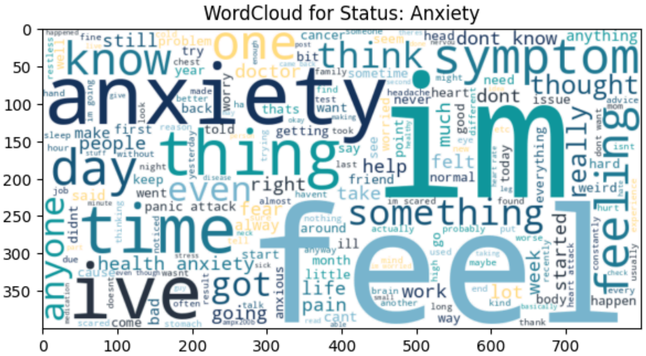
```
# Initialize the stemmer
stemmer = nltk.SnowballStemmer("english")

# Function to stem tokens and convert them to strings
def stem_tokens(tokens):
    return ' '.join(stemmer.stem(str(token)) for token in tokens)

# Apply the function to the 'tokens' column
df['tokens_stemmed'] = df['tokens'].apply(stem_tokens)

df.sample(5)
```

	statement	status	statemnent_length	num_of_sentences	tokens	tokens_st
51784	ive diagnosed bpd borderline personality disor...	Personality disorder	1056	1	[ive, diagnosed, bpd, borderline, personality,...	ive diagn borderlin p disord 4
28726	2 socialize sure evolve lone creatures	Normal	827	1	[2, socialize, sure, evolve, lone, creatures,...	2 soci evo creatur

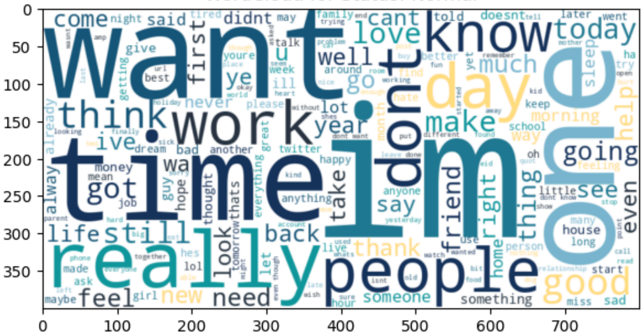
word counts

```
colors = ['#16325B', '#227B94', '#78B7D0', '#FFDC7F', '#18587A', '#11999E', '#283644']

# Define a color function
def color_func(word, font_size, position, orientation, random_state=101, **kwargs):
    return random.choice(colors)


statuses = df['status'].unique()

plt.figure(figsize=(12, 36))  # Adjust figure size as needed

# Generate and plot the WordCloud for each category
for i, status in enumerate(statuses):
    # Filter the tokens data for the current status
    tokens_data = ' '.join(df[df['status'] == status]['tokens'].dropna().apply(lambda x: ' '

    # Generate the WordCloud
    wordcloud = WordCloud(width=800, height=400, background_color='white', color_func=color_

    # Plot the WordCloud in a subplot
    axes = plt.subplot(len(statuses) // 2 + 1, 2, i + 1)  # Adjust number of rows and columr
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.title(f'WordCloud for Status: {status}')
    #plt.axis('off')  # Turn off axes for a cleaner look

# Apply tight layout after generating all subplots
plt.tight_layout()

# Adjust the vertical spacing between subplots (hspace controls vertical space)
plt.subplots_adjust(hspace= -0.8)
plt.show()
```

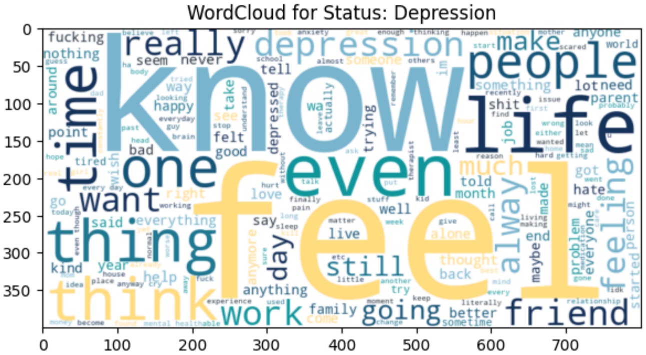
WordCloud for Status: Anxiety



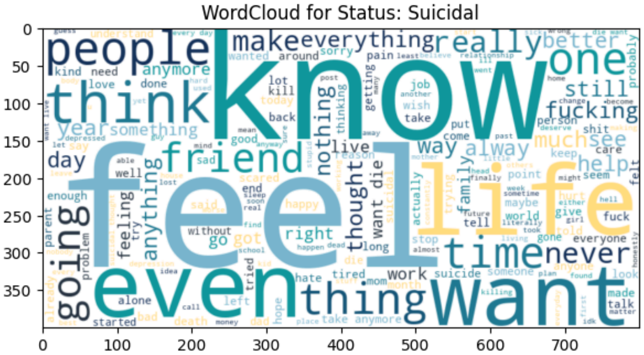
WordCloud for Status: Normal



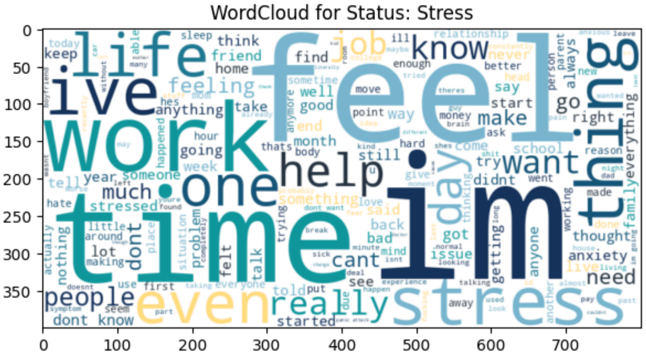
WordCloud for Status: Depression



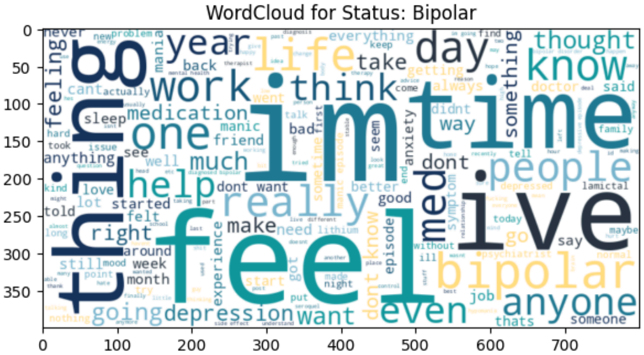
WordCloud for Status: Suicidal



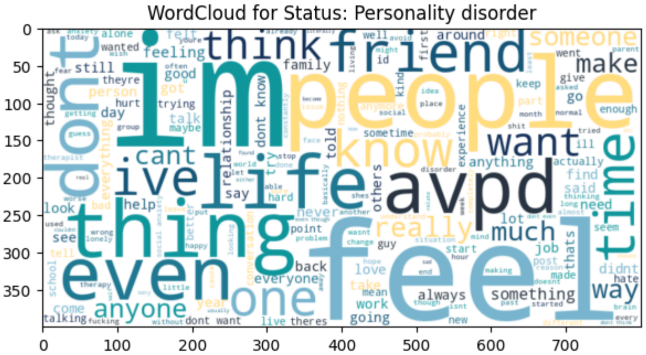
WordCloud for Status: Stress



WordCloud for Status: Bipolar



WordCloud for Status: Personality disorder



What is the TF (Term of Frequency) and IDF (Inverse Document Frequency)?

- **What is TF-IDF?**

TF-IDF (Term Frequency-Inverse Document Frequency) is a numerical statistic used to evaluate the importance of a word in a document relative to a collection of documents, typically in the context of text mining and Natural Language Processing (NLP). It is commonly used as a feature extraction method to convert textual data into numerical features that machine learning models can process.

- The TF-IDF measure combines two components:

1. **Term Frequency (TF):**

   Definition: The number of times a word (term) appears in a document, divided by the total number of words in that document. It gives us a sense of how important a word is within a single document.

   Formula:

$$\text{TF}(t,d) = \frac{\text{Number of times term } t \text{ appears in document } d}{\text{Total number of terms in document } d}$$

2. **Inverse Document Frequency (IDF):**

   Definition: The logarithm of the total number of documents in the corpus divided by the number of documents containing the term.

   It measures how important a word is across the whole corpus. Common words that appear in many documents (like "the," "is") have low IDF values because they are less informative, while rare terms have higher IDF values.

Formula:

$$\text{IDF}(t, D) = \log\left(\frac{\text{Total number of documents in the corpus}}{1 + \text{Number of documents containing the term } t}\right)$$

3. **TF-IDF:**

   Definition: The product of the term frequency (TF) and inverse document frequency (IDF) for a term in a document. It reflects both the term's frequency within a specific document and how unique or important the term is across the entire document corpus.

   Formula:

   $$\text{TF-IDF}(t, d, D) = \text{TF}(t, d) \times \text{IDF}(t, D)$$

   In these formulas:

   - (t) refers to a term (word),
   - (d) refers to a document,
   - (D) refers to the entire corpus (collection of documents),
   - (\log) is the logarithm function, typically base 10 or natural logarithm.

- **How TF-IDF is Useful Feature Extraction in NLP:** TF-IDF helps convert raw text data into a matrix of numerical features, making the data suitable for machine learning models. Each document is represented as a vector of TF-IDF values, where each value corresponds to the importance of a word in that document. This is a common technique in text classification tasks, such as sentiment analysis, spam detection, and topic classification.

  Captures the Importance of Words: By using TF-IDF, we can ignore frequently occurring but uninformative words (e.g., "the", "and") and focus on more meaningful terms that are key to the context of a document. This helps improve the accuracy and relevance of the model's predictions by emphasizing words that differentiate one document from another.

  Reduces Noise in Text: TF-IDF reduces the weight of very common words across all documents, as they are often not useful for distinguishing between documents. It assigns higher importance to rare and meaningful terms, which tend to carry more information about the content of the document.

  Improves Text Similarity Measures: TF-IDF vectors are often used to calculate the similarity between documents (e.g., using cosine similarity). This is useful in tasks like document clustering, search engines, and recommendation systems.

- **Use Case in Mental Health Sentiment Analysis**

  In our dataset, which contains mental health-related statements, applying TF-IDF can help the model understand which words are most significant for predicting the mental health status of a given statement. For example:

- Words like "hopeless," "depressed," and "suicidal" may have high TF-IDF scores in documents labeled with "Depression" or "Suicidal" mental health statuses, but these words may be rare across other categories.
    - Common words like "I," "the," and "is" will have low IDF values since they appear in many documents, making them less useful for classification.

By transforming the text data into TF-IDF vectors, you provide a more informative and compact representation of the statements, which can lead to better performance of machine learning models.

Data Pre Processsing

Taking Features

```
X = df[['tokens_stemmed']]
y = df['status']
```

labels encodig

```
lbl_enc = LabelEncoder()
y = lbl_enc.fit_transform(y.values)
```

Splitting the dataset

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=601)

# Second split: Temporary set into validation and test sets (50% of the temp set goes to val
# X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_stat
```

Convert text to features using TF-IDF vectoriser

```
# 1. Initialize TF-IDF Vectorizer and fit/transform on the 'tokens' column
vectorizer = TfidfVectorizer(max_features=4000)
X_train_tfidf = vectorizer.fit_transform(X_train['tokens_stemmed'])
# X_val_tfidf = vectorizer.transform(X_val['tokens_stemmed'])  # Transform validation set
X_test_tfidf = vectorizer.transform(X_test['tokens_stemmed'])  # Transform test set

# 2.  Extract numerical features for train, validation, and test
# X_train_num = X_train[['statement_length', 'num_of_sentences']].values
# X_val_num = X_val[['statement_length', 'num_of_sentences']].values
# X_test_num = X_test[['statement_length', 'num_of_sentences']].values

# 3. Combine TF-IDF features (and numerical features if used) for train, validation, and tes
X_train_combined = hstack([X_train_tfidf])
# X_val_combined = hstack([X_val_tfidf])
X_test_combined = hstack([X_test_tfidf])

# Print number of features used
print('Number of feature words: ', len(vectorizer.get_feature_names_out()))
```

```
Number of feature words:  4000
```

```
X_train_combined[0].toarray()
```

```
array([[0., 0., 0., ..., 0., 0., 0.]])
```

Resampling very importing remmember

```
# Convert TF-IDF sparse matrix to dense format for SMOTE
# X_train_dense = X_train_tfidf.toarray()

# Apply SMOTE to the dense matrix
# smote = SMOTE(random_state=42)

# X_train_combined, y_train = smote.fit_resample(X_train_dense, y_train)
```

Logistic Regression Model

```
# Define the LogisticRegression model
clf = LogisticRegression(solver='liblinear', random_state=101)

# Define the hyperparameter grid to search over
param_grid = {
    'penalty': ['l1', 'l2'],
    'C': [0.1, 1, 10, 100],
    'solver': ['liblinear']
}

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=clf, param_grid=param_grid, scoring='accuracy', cv=5, r

# Fit the grid search on the training data
grid_search.fit(X_train_combined, y_train)
```

```
Fitting 5 folds for each of 8 candidates, totalling 40 fits
```

GridSearchCV

best_estimator_: LogisticRegression

LogisticRegression

best params

```
# Get the best model and hyperparameters
best_clf = grid_search.best_estimator_
print("Best hyperparameters found: ", grid_search.best_params_)
```

```
Best hyperparameters found:  {'C': 1, 'penalty': 'l1', 'solver': 'liblinear'}
```

Confusion Matrix

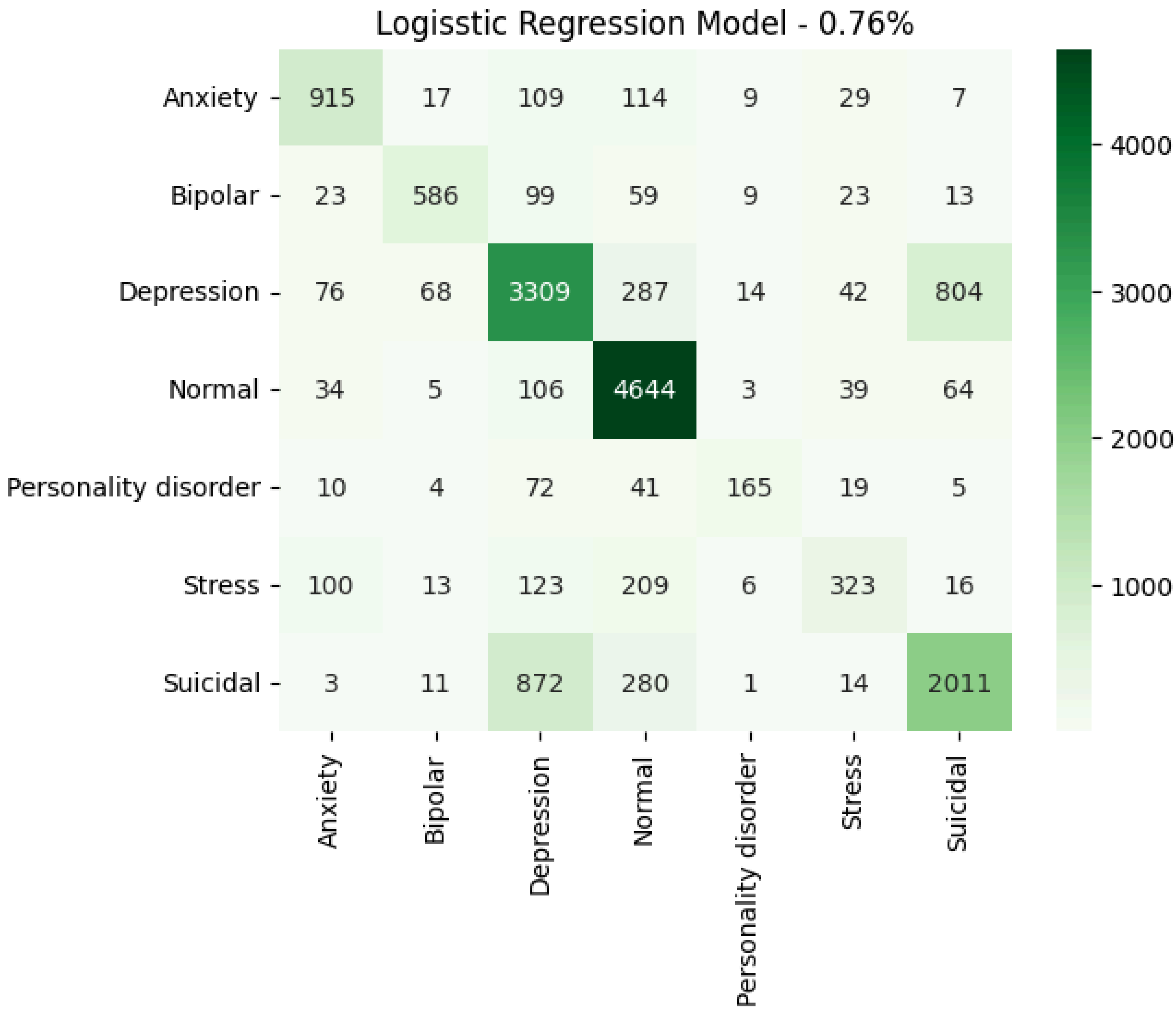
```
#  Predict on the test set using the best model
y_pred = best_clf.predict(X_test_combined)

#  Calculate accuracy
accuracy_reg = accuracy_score(y_test, y_pred)
print("\nAccuracy: ", accuracy_reg)
```

```
#  Compute the confusion matrix and classification report
conf_matrix_reg = confusion_matrix(y_test, y_pred)
labels = lbl_enc.classes_
print(classification_report(y_test, y_pred, target_names=labels))
```

```
Accuracy:  0.7562796583359697
                      precision    recall  f1-score   support

             Anxiety       0.79      0.76      0.78      1200
             Bipolar       0.83      0.72      0.77       812
          Depression       0.71      0.72      0.71      4600
              Normal       0.82      0.95      0.88      4895
Personality disorder       0.80      0.52      0.63       316
              Stress       0.66      0.41      0.51       790
            Suicidal       0.69      0.63      0.66      3192

            accuracy                           0.76     15805
           macro avg       0.76      0.67      0.71     15805
        weighted avg       0.75      0.76      0.75     15805
```

Heat Map

```
ax = sns.heatmap(conf_matrix_reg, annot = True, fmt='d', cmap='Greens', xticklabels=labels,
ax.set_title(f'Logisstic Regression Model - {accuracy_reg:.2}%')
plt.show()
```



🎭 KNN Model

Model taining

```
# Define the KNN model
clf = KNeighborsClassifier()

# Define the hyperparameter grid to search over
param_grid = {
    'n_neighbors': [ 30,70,90,100,150,180],  # Range of neighbors to try
    'weights': ['uniform', 'distance'],  # Whether to use uniform or distance-based weightir
    'metric': ['euclidean', 'manhattan', 'cosine']  # Different distance metrics
}

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=clf, param_grid=param_grid, scoring='accuracy', cv=5, r
```

```
# Fit the grid search on the training data
grid_search.fit(X_train_combined, y_train)
```

```
Fitting 5 folds for each of 2 candidates, totalling 10 fits
```

GridSearchCV

best_estimator_: KNeighborsClassifier

KNeighborsClassifier

```
#  Get the best model and hyperparameters
best_knn = grid_search.best_estimator_
print("Best hyperparameters found: ", grid_search.best_params_)
```

```
Best hyperparameters found:  {'metric': 'cosine', 'n_neighbors': 100, 'weights': 'distar
```

Confnusion matrix

```
# Make predictions on the test set using the best model
y_pred = best_knn.predict(X_test_combined)

# Calculate accuracy
accuracy_knn = accuracy_score(y_test, y_pred)
print("\nAccuracy: ", accuracy_knn)

# Compute the confusion matrix and classification report
conf_matrix_knn = confusion_matrix(y_test, y_pred)
labels = lbl_enc.classes_
print(classification_report(y_test, y_pred, target_names=labels))
```

```
Accuracy:  0.6666877570389117
                      precision    recall  f1-score   support

             Anxiety       0.83      0.65      0.73      1200
             Bipolar       0.85      0.55      0.67       812
          Depression       0.63      0.63      0.63      4600
              Normal       0.65      0.93      0.77      4895
Personality disorder       0.98      0.37      0.54       316
              Stress       0.87      0.23      0.37       790
            Suicidal       0.65      0.49      0.56      3192

            accuracy                           0.67     15805
```

```
   macro avg       0.78      0.55      0.61     15805
weighted avg       0.69      0.67      0.65     15805
```

🦽 Support Vector Machine Model

```
# Define the SVM model
clf = SVC()

# Define the hyperparameter grid to search over
param_grid = {
    'C': [ 0.1,1,10,100],  # Regularization parameter
    'kernel': ['linear', 'rbf', 'poly'],  # Different kernel types
    'gamma': [ 'auto','scale'],  # Kernel coefficient for 'rbf', 'poly'
    'degree': [2, 3]  # Degree for polynomial kernel (only relevant for 'poly' kernel)
}

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=clf, param_grid=param_grid, scoring='accuracy', cv=5, r

# Fit the grid search on the training data
grid_search.fit(X_train_combined, y_train)
```

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits
```

▸ **GridSearchCV** ⓘ ?

▸ **best_estimator_: SVC**

▸ SVC ?

```
# Get the best model and hyperparameters
best_svm = grid_search.best_estimator_
print("Best hyperparameters found: ", grid_search.best_params_)
```

```
Best hyperparameters found:  {'C': 1, 'degree': 2, 'gamma': 'scale', 'kernel': 'linear'}
```

Confusion Matrix

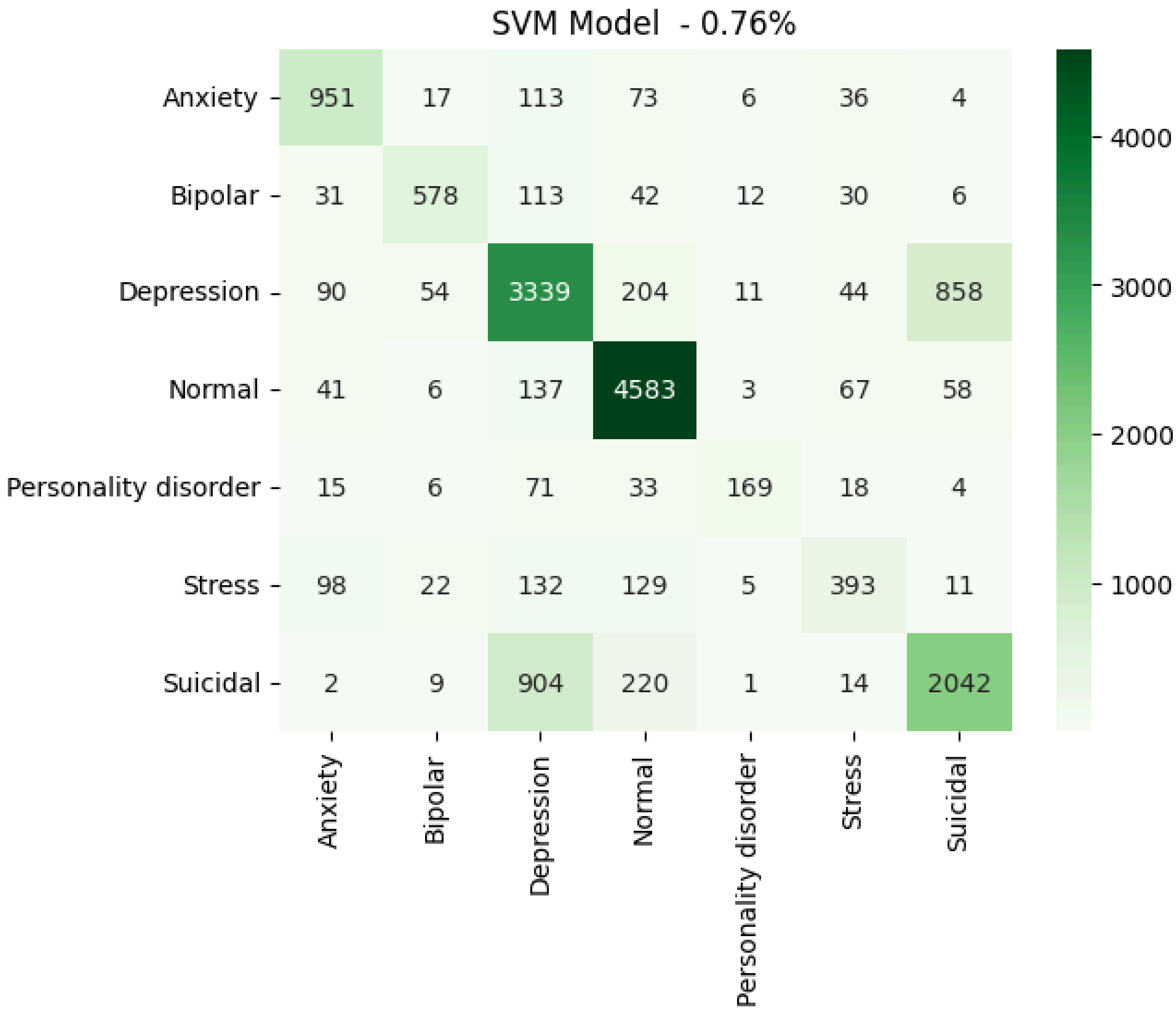
```
#  Make predictions on the test set using the best model
```

```
y_pred = best_svm.predict(X_test_combined)

# Calculate accuracy
accuracy_svm = accuracy_score(y_test, y_pred)
print("\nAccuracy: ", accuracy_svm)

# Compute the confusion matrix and classification report
conf_matrix_Svm = confusion_matrix(y_test, y_pred)
labels = lbl_enc.classes_
print(classification_report(y_test, y_pred, target_names=labels))
```

```
Accuracy:  0.762733312242961
                      precision    recall  f1-score   support

             Anxiety       0.77      0.79      0.78      1200
             Bipolar       0.84      0.71      0.77       812
          Depression       0.69      0.73      0.71      4600
              Normal       0.87      0.94      0.90      4895
Personality disorder       0.82      0.53      0.65       316
              Stress       0.65      0.50      0.56       790
            Suicidal       0.68      0.64      0.66      3192

            accuracy                           0.76     15805
           macro avg       0.76      0.69      0.72     15805
        weighted avg       0.76      0.76      0.76     15805
```

Heat Map

```
ax = sns.heatmap(conf_matrix_Svm, annot = True, fmt='d', cmap='Greens', xticklabels=labels,
ax.set_title(f'SVM Model  - {accuracy_svm:.2}%')
plt.show()
```



Neural Network

```
# Define the layers in an array
layers = [
    Dense(units=128, activation='relu', input_shape=(X_train_combined.shape[1],),kernel_regu
    BatchNormalization(),
    Dropout(rate=0.2),  # Dropout Layer 1
    Dense(units=64, activation='relu',kernel_regularizer=l2(0.01)),  # Hidden Layer 1
    Dropout(rate=0.1),  # Dropout Layer 1
    Dense(units=16, activation='relu',kernel_regularizer=l2(0.01)),  # Hidden Layer 2

    Dropout(rate=0.2),  # Dropout Layer 2
    Dense(units=len(lbl_enc.classes_), activation='softmax')  # Output Layer
]

# Initialize the Sequential model
model = Sequential(layers)
```

```
# Compile the model: Using Adam optimizer, sparse categorical crossentropy loss, and accurac
model.compile(optimizer=Adam(learning_rate=0.001),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Train the model on training data with validation on the test data
history = model.fit(X_train_combined.toarray(),
                    y_train,
                    epochs=20,
                    batch_size=64,
                    validation_split=0.3
                    )
```

```
Epoch 1/20
404/404 ━━━━━━━━━━━━━━━━━━━━ 9s 15ms/step - accuracy: 0.5388 - loss: 2.5398 - val_accura
Epoch 2/20
404/404 ━━━━━━━━━━━━━━━━━━━━ 14s 24ms/step - accuracy: 0.7234 - loss: 1.1433 - val_accur
Epoch 3/20
404/404 ━━━━━━━━━━━━━━━━━━━━ 5s 13ms/step - accuracy: 0.7376 - loss: 1.0601 - val_accura
Epoch 4/20
404/404 ━━━━━━━━━━━━━━━━━━━━ 11s 16ms/step - accuracy: 0.7444 - loss: 1.0372 - val_accur
Epoch 5/20
404/404 ━━━━━━━━━━━━━━━━━━━━ 11s 17ms/step - accuracy: 0.7517 - loss: 1.0131 - val_accur
Epoch 6/20
404/404 ━━━━━━━━━━━━━━━━━━━━ 5s 13ms/step - accuracy: 0.7575 - loss: 0.9837 - val_accura
Epoch 7/20
404/404 ━━━━━━━━━━━━━━━━━━━━ 12s 17ms/step - accuracy: 0.7690 - loss: 0.9491 - val_accur
Epoch 8/20
404/404 ━━━━━━━━━━━━━━━━━━━━ 8s 13ms/step - accuracy: 0.7748 - loss: 0.9349 - val_accura
Epoch 9/20
404/404 ━━━━━━━━━━━━━━━━━━━━ 10s 13ms/step - accuracy: 0.7735 - loss: 0.9326 - val_accur
Epoch 10/20
404/404 ━━━━━━━━━━━━━━━━━━━━ 11s 15ms/step - accuracy: 0.7820 - loss: 0.9131 - val_accur
Epoch 11/20
404/404 ━━━━━━━━━━━━━━━━━━━━ 11s 17ms/step - accuracy: 0.7795 - loss: 0.9148 - val_accur
Epoch 12/20
404/404 ━━━━━━━━━━━━━━━━━━━━ 5s 13ms/step - accuracy: 0.7844 - loss: 0.8885 - val_accura
Epoch 13/20
404/404 ━━━━━━━━━━━━━━━━━━━━ 12s 17ms/step - accuracy: 0.7884 - loss: 0.8708 - val_accur
Epoch 14/20
404/404 ━━━━━━━━━━━━━━━━━━━━ 9s 15ms/step - accuracy: 0.7912 - loss: 0.8585 - val_accura
Epoch 15/20
404/404 ━━━━━━━━━━━━━━━━━━━━ 9s 13ms/step - accuracy: 0.7879 - loss: 0.8757 - val_accura
Epoch 16/20
404/404 ━━━━━━━━━━━━━━━━━━━━ 10s 13ms/step - accuracy: 0.7971 - loss: 0.8570 - val_accur
Epoch 17/20
404/404 ━━━━━━━━━━━━━━━━━━━━ 5s 13ms/step - accuracy: 0.7929 - loss: 0.8550 - val_accura
Epoch 18/20
404/404 ━━━━━━━━━━━━━━━━━━━━ 10s 13ms/step - accuracy: 0.7945 - loss: 0.8460 - val_accur
Epoch 19/20
404/404 ━━━━━━━━━━━━━━━━━━━━ 7s 17ms/step - accuracy: 0.7918 - loss: 0.8505 - val_accura
Epoch 20/20
```

```
404/404 ━━━━━━━━━━━━━━━━━━━━ 5s 13ms/step - accuracy: 0.7968 - loss: 0.8392 - val_accura
```

Confusion Matrix

```
# Make predictions on the test set
y_pred_prob = model.predict(X_test_combined.toarray())
y_pred = y_pred_prob.argmax(axis=1)  # Convert probabilities to class predictions

# Calculate the accuracy
accuracy_nn = accuracy_score(y_test, y_pred)
print("\n")
print("Accuracy:", accuracy_nn)

# Compute the confusion matrix
labels = lbl_enc.classes_
conf_matrix_nn = confusion_matrix(y_test, y_pred)

# Print classification report
print(classification_report(y_test, y_pred, target_names=labels))
```

```
494/494 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step


Accuracy: 0.7524201202151218
                      precision    recall  f1-score   support

             Anxiety       0.78      0.77      0.78      1200
             Bipolar       0.72      0.77      0.75       812
          Depression       0.72      0.69      0.70      4600
              Normal       0.90      0.89      0.90      4895
Personality disorder       0.65      0.60      0.62       316
```