

✓ Sentiment Analysis for Mental Health Monitoring 🤖



✓ Sentiment Analysis for Mental Health Monitoring 🤖

[dataset source](#) 🔗

About The Dataset 📝

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 Source 🌐

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 .

Double-click (or enter) to edit

✓ Two Level Classification Approach

About Two-Level Classification

In a two-level classification approach, we break down a multi-class classification problem into two stages:

1. Stage 1 - Binary Classification:

- This stage first classifies data into two categories: **Normal** and **Non-Normal**.
- By creating this separation, we simplify the problem and reduce the initial complexity, allowing the model to focus on distinguishing between these high-level classes.
- This binary classifier uses features from the full dataset, ensuring that it can accurately detect if a statement falls into the "Normal" or "Non-Normal" category.

2. Stage 2 - Multi-Class Classification:

- In this stage, we take statements classified as **Non-Normal** from Stage 1 and further classify them into specific mental health categories such as **Depression, Suicidal, Anxiety, Stress, Bi-Polar, and Personality Disorder**.
- This multi-class classifier uses data that has already been identified as "Non-Normal," focusing on differentiating among various mental health conditions.

Why Two-Level Classification?

- 🟢 **Improved Accuracy:** This approach allows each classifier to focus on a subset of the problem, potentially increasing accuracy by reducing the complexity each model has to handle.
- 🟢 **Efficient Use of Resources:** By filtering out "Normal" cases in Stage 1, Stage 2 can concentrate only on differentiating among the specific mental health conditions, making it more efficient.
- 🟢 **Handling Imbalance:** Many datasets are imbalanced, with "Normal" cases being more common. Splitting the classification into two levels allows us to address class imbalance in each level separately.

How It Works in This Model

1. Data Preparation:

- First, we preprocess the dataset by cleaning the text and extracting features (e.g., TF-IDF, POS tags, emojis).
- Then, we split the data for training and testing each stage separately.

2. Training:

- **Stage 1:** We train a binary classifier on the full training data to classify statements as **Normal** or **Non-Normal**.
- **Stage 2:** We train a multi-class classifier on the subset of training data labeled as **Non-Normal**, focusing only on the six specific mental health conditions.

3. Testing:

- For each test instance, we first use Stage 1 to determine if it's "Normal" or "Non-Normal."
- If classified as **Non-Normal**, the instance is passed to Stage 2, where it is classified into one of the specific mental health categories.

Evaluation and Metrics

- **Stage-Specific Metrics:** Each stage is evaluated separately, with accuracy, recall, precision, and F1-score measured for both the binary and multi-class classifiers.
- **Combined Evaluation:** Finally, we combine predictions from both stages to evaluate the overall performance of the two-level classification system.

This two-level approach allows us to accurately identify both "Normal" and specific mental health conditions, leveraging targeted classification strategies to improve overall accuracy and interpretability.

✓ 📄 *Dataset Loading and Preparing*

✓ Constants

```
data_read_path = '/content/drive/MyDrive/CombinedData.csv'
data_clean_path = '/content/drive/MyDrive/CombinedDataCleaned.csv'
```

▼ Import Libraries

lets start by importing the nessacary libraries

▼ import

```
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
from sklearn.feature_extraction.text import TfidfVectorizer
from scipy.sparse import hstack
#import emoji

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
from google.colab import drive
from textblob import TextBlob

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

#import stanza
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

▼ Download packages

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

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data] Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data] Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data] Unzipping tokenizers/punkt_tab.zip.
[nltk_data] Downloading package averaged_perceptron_tagger_eng to
[nltk_data] /root/nltk_data...
[nltk_data] Unzipping taggers/averaged_perceptron_tagger_eng.zip.
True
```

```
pip install stanza
```

```
Collecting stanza
  Downloading stanza-1.9.2-py3-none-any.whl.metadata (13 kB)
Collecting emoji (from stanza)
  Downloading emoji-2.14.0-py3-none-any.whl.metadata (5.7 kB)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from stanza) (1.26.4)
Requirement already satisfied: protobuf>=3.15.0 in /usr/local/lib/python3.10/dist-packages (from stanza) (4.25.5)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from stanza) (2.32.3)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from stanza) (3.4.2)
Requirement already satisfied: torch>=1.3.0 in /usr/local/lib/python3.10/dist-packages (from stanza) (2.5.1+cu121)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from stanza) (4.66.6)
Requirement already satisfied: tomli in /usr/local/lib/python3.10/dist-packages (from stanza) (2.1.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch>=1.3.0->stanza) (3.16.1)
Requirement already satisfied: typing-extensions>=4.8.0 in /usr/local/lib/python3.10/dist-packages (from torch>=1.3.0->stanza) (4.12.2)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch>=1.3.0->stanza) (3.1.4)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from torch>=1.3.0->stanza) (2024.10.0)
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.10/dist-packages (from torch>=1.3.0->stanza) (1.13.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from sympy==1.13.1->torch>=1.3.0->stanza) (3.4.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->stanza) (3.4.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->stanza) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->stanza) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->stanza) (2024.8.30)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->torch>=1.3.0->stanza) (3.0.2)
Downloading stanza-1.9.2-py3-none-any.whl (1.1 MB)
1.1/1.1 MB 14.2 MB/s eta 0:00:00
Downloading emoji-2.14.0-py3-none-any.whl (586 kB)
586.9/586.9 kB 30.2 MB/s eta 0:00:00
Installing collected packages: emoji, stanza
Successfully installed emoji-2.14.0 stanza-1.9.2
```

▼ Loading the dataset

```
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
df = pd.read_csv(data_read_path, 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 it is

```
# 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: Sweating Whenever I visit in a group of 3 or more in the same room after about 45 minutes I start to sweat a lot and smell an
Does anyone else feel like this? Or have any suggestions?

Status: Bipolar

Statement: Bipolar Reddit, what's the best life lesson you've learned after coming out of a Psychosis? After my first one I basically ha

What's yours???

Edit: Apparently Bipolar people are wise as fuck.

Status: Depression

Statement: I will keep this short.My wife and I have been together since the age of 15. 21 years without break. we are the type of coupl

Status: Normal

Statement: I'm the type who won't start anyway

Status: Personality disorder

Statement: Anyone here try making online friends? For me its difficult because everyone is more advanced than me or younger. Many people

Another issue that would happen is the few people I found remotely on my level would ditch me after some months. I think it does not wor

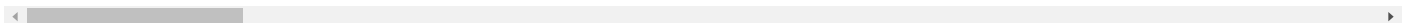
In theory I should just go ask people if they want to play certain games but it does not feel right if people just opening up about thei

Status: Stress

Statement: He would play really loud music on the speaker. It got so bad when he threw the living room table at me.. we got kicked out c


Status: Suicidal

Statement: My partner says they love me and care but they fucking do not. They just do not want to admit they fucked up and had a child

◀  ▶

✓ Data set Describing

```
df.describe()
```



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

✓ Missings Value Removal


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



	0
statement	362
status	0

dtype: int64

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




	0
statement	0
status	0


dtype: int64

✓ What is our Labels 'target'?

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

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

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



	count
status	
Normal	16343
Depression	15404
Suicidal	10652
Anxiety	3841
Bipolar	2777
Stress	2587
Personality disorder	1077

dtype: int64

✓ How the samples are distributed over the status ?

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

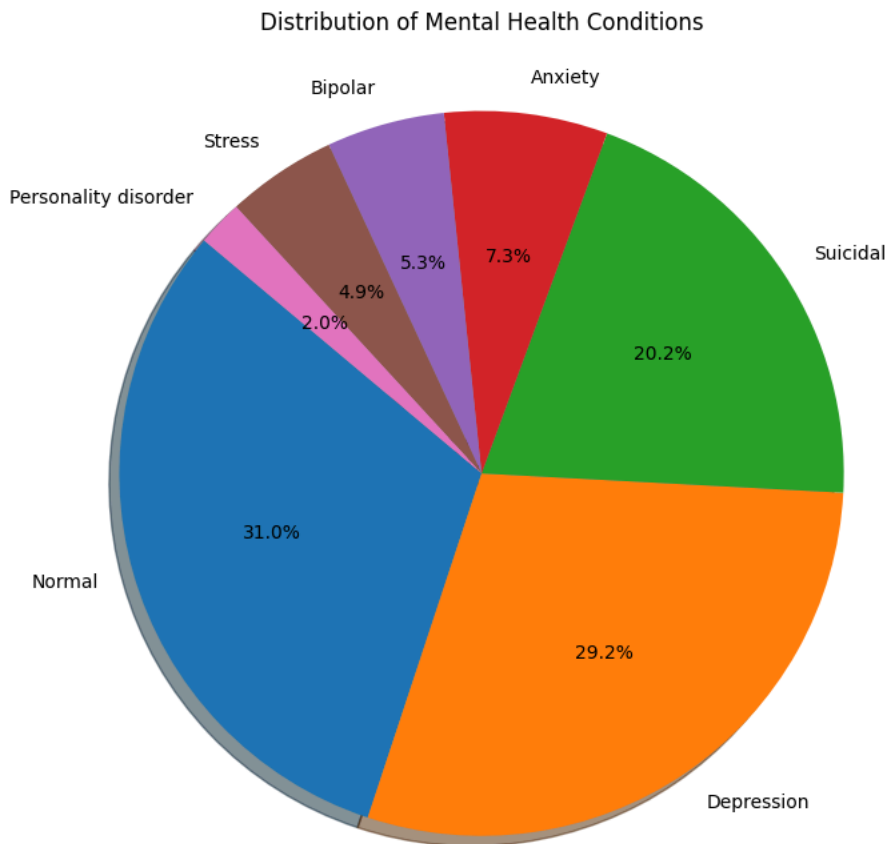
```
# Define colors for each category (7 colors for 7 categories)
colors = ['#1F77B4', '#FF7F0E', '#2CA02C', '#D62728', '#9467BD', '#8C564B', '#E377C2']
```

```
# 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 distributed?

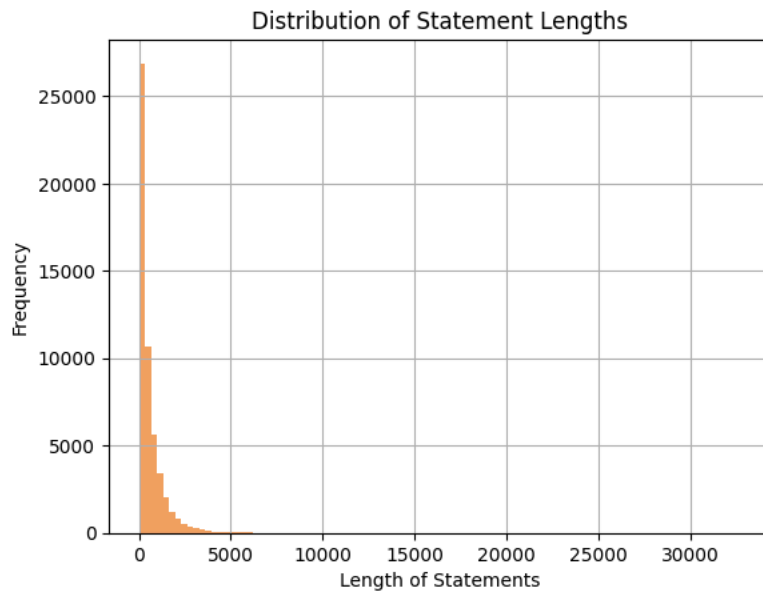
✓ define statement length column

```
# calculate the length of each statement
df['statement_length'] = df['statement'].apply(lambda text : len(text))
```

```
# calculate the length of each statement
df['num_of_words'] = df['statement'].apply(lambda text : len(text.split()))
```

✓ statement length distribution

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

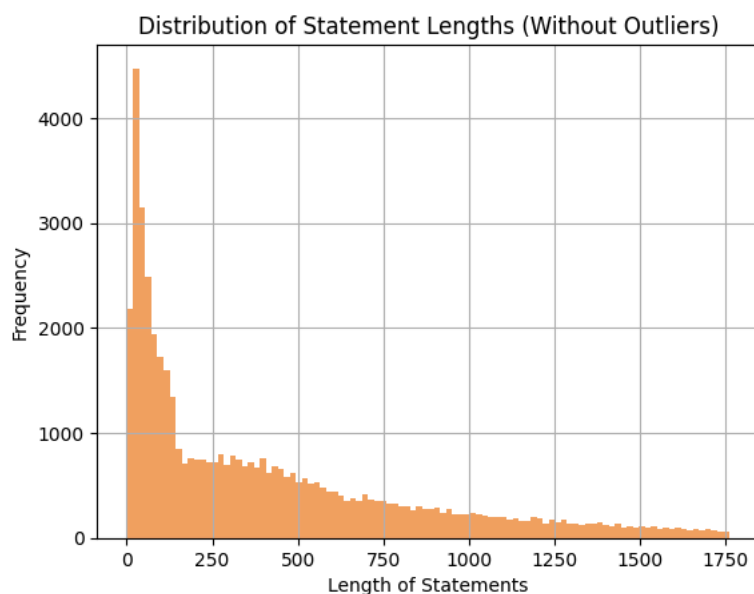



▼ Distribution of Statement Lengths (Without Outliers)

```
# Calculate Q1 (25th percentile) and Q3 (75th percentile)
Q1 = df['statement_length'].quantile(0.25)
Q3 = df['statement_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['statement_length'] >= lower_bound) & (df['statement_length'] <= upper_bound)]
# Plot the distribution of statement lengths without outliers
filtered_df['statement_length'].hist(bins=100, color='#F4A261')
plt.title('Distribution of Statement Lengths (Without Outliers)')
plt.xlabel('Length of Statements')
plt.ylabel('Frequency')
plt.show()
```



✓ Distributions : *Can we find simple classifier ?*

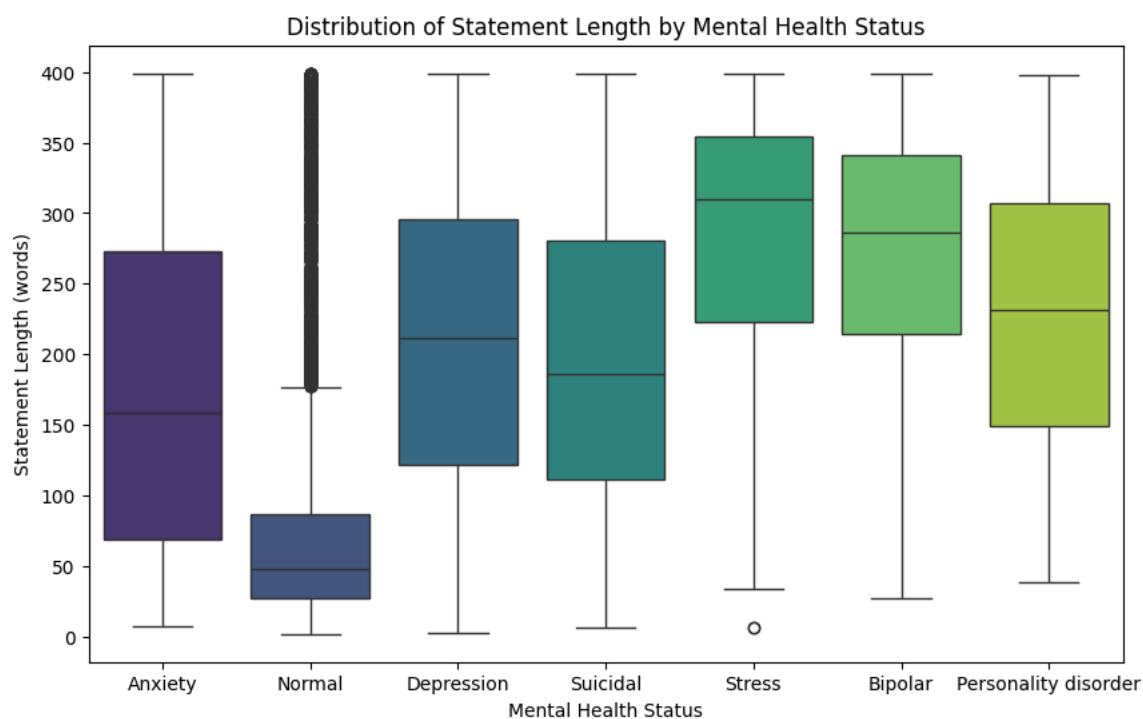
✓ Helper function

```
status_analysis = df.groupby('status').agg({
    'statement_length': ['mean', 'median', 'std'],
    'num_of_words': ['mean', 'median', 'std']
})
```

✓ Box plot of statement length vs status

```
# Filter the DataFrame to show only rows where the 'sentence_length' is less than 10
df_short_sentences = df[df['statement_length'] < 400]
```

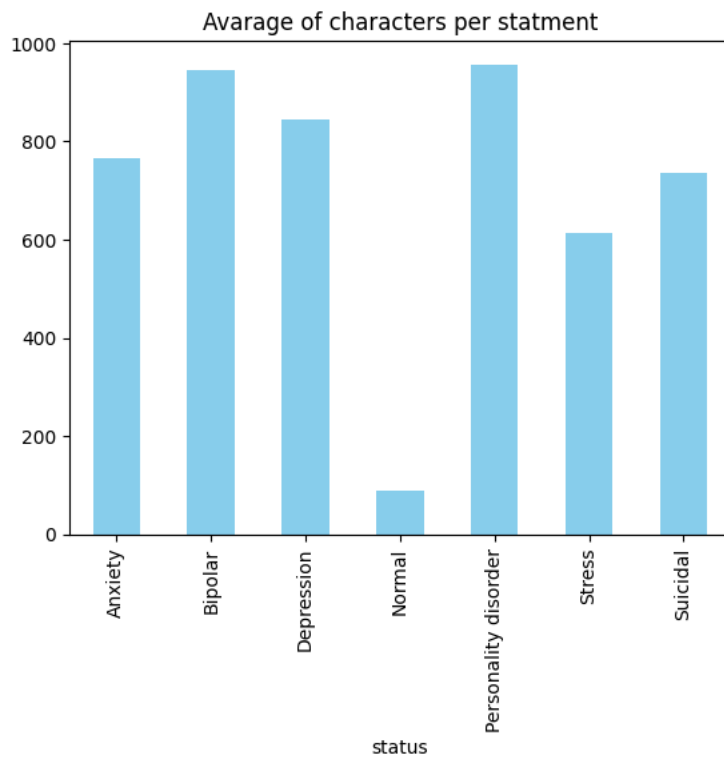
```
# Plot a box plot of statement length by status
plt.figure(figsize=(10, 6))
sns.boxplot(data=df_short_sentences, x='status', y='statement_length', palette="viridis")
plt.title("Distribution of Statement Length by Mental Health Status")
plt.xlabel("Mental Health Status")
plt.ylabel("Statement Length (words)")
plt.show()
```



✓ Average of characters per statment

```
status_analysis['statement_length']['mean'].plot(kind='bar',color='skyblue')  
plt.title('Avarage of characters per statment')
```

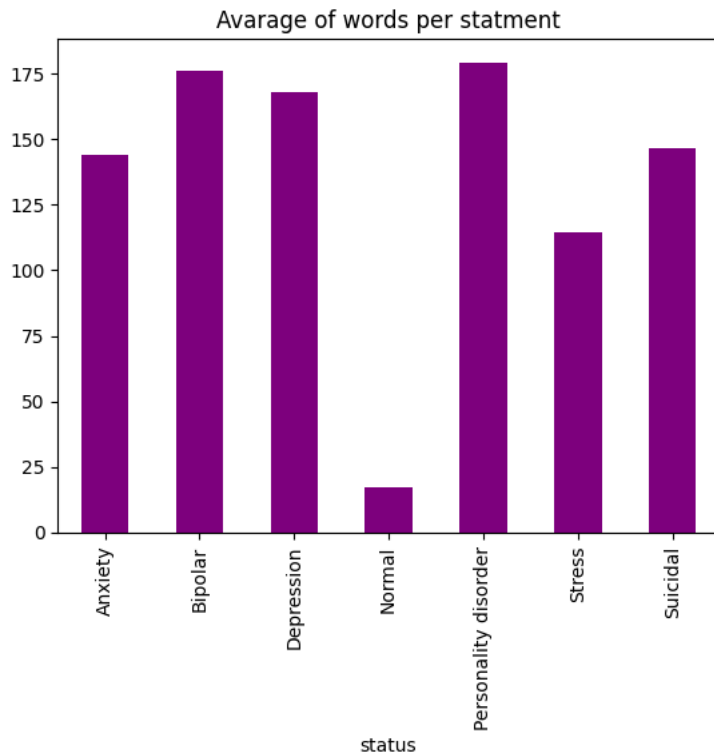
↻ Text(0.5, 1.0, 'Avarage of characters per statment')



✓ Avarage of words per statment

```
status_analysis['num_of_words']['mean'].plot(kind='bar',color='purple')  
plt.title('Avarage of words per statment')
```

↻ Text(0.5, 1.0, 'Avarage of words per statment')



✓ What is the frequency of POS Tags ?

✓ Helper function

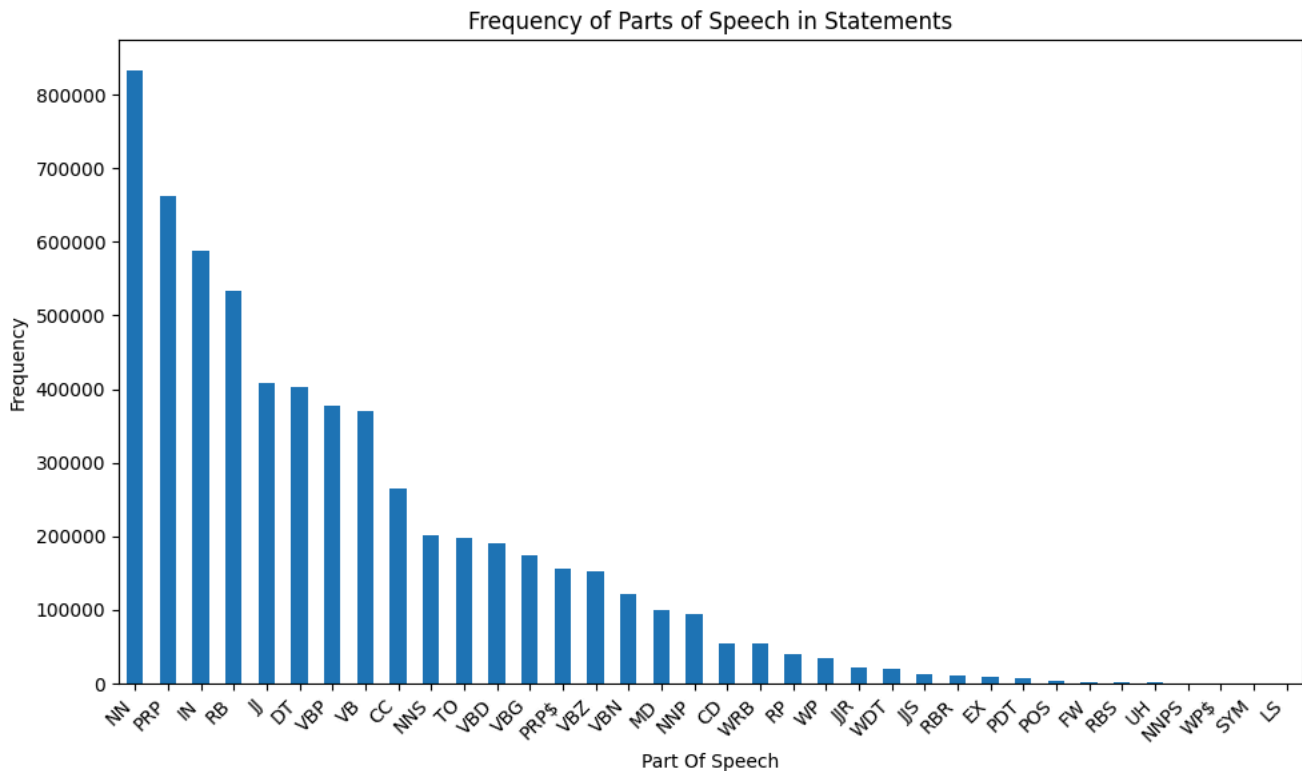
```
def plot_pos_frequency(df):
    # Combine all statements into one text
    blob = TextBlob(" ".join(df['statement'].astype(str)))

    # Extract POS tags and create DataFrame
    pos_df = pd.DataFrame(blob.tags, columns=['word', 'pos'])
    pos_counts = pos_df['pos'].value_counts()[:50] # Get top 50 POS tags by frequency

    # Plot the frequency of POS tags
    plt.figure(figsize=(10, 6))
    pos_counts.plot(kind='bar')
    plt.xlabel("Part Of Speech")
    plt.ylabel("Frequency")
    plt.title("Frequency of Parts of Speech in Statements")
    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()
    plt.show()
```

✓ POS Frequency

```
# Run the function on your DataFrame
plot_pos_frequency(df)
```



✓ NLP Pre-Processing

✓ what we will do

in this part of the ml pipeline, we perform several essential NLP preprocessing steps to prepare our data for machine learning. These steps include:

1. Text Processing:

- include :
 - pattern removing (links, emails and etc.)
 - lower casing
 - stop words removing

2. tokinezation & Stemming :

- Tokenization :
 - this step converts each text sample into an array of tokens (individual words or sub-words).
 - we use Stanza word tokenizer to ensure precise, language-specific tokenization.
- Stemming :
 - in this step, we transform each tokenized array into a stemmed version, reducing each word to its base or root form.
 - stanza NLP tools allow us to use various stemming techniques for extracting meaningful root words.

3. Part-of-Speech (POS) Tagging and Filtering:

- using Stanza, we apply POS tagging to identify and filter specific parts of speech, such as:
 - **Verbs:** Identify and include only verbs to capture action-related semantics.
 - **Nouns:** Extract nouns to focus on entity- or object-based information.
 - **Adjectives:** Include adjectives to analyze descriptive language and sentiment.
- This POS filtering allows us to tailor the input data by emphasizing different linguistic elements and can be customized based on the classification or NLP goals.

✓ 1. Text Processing:

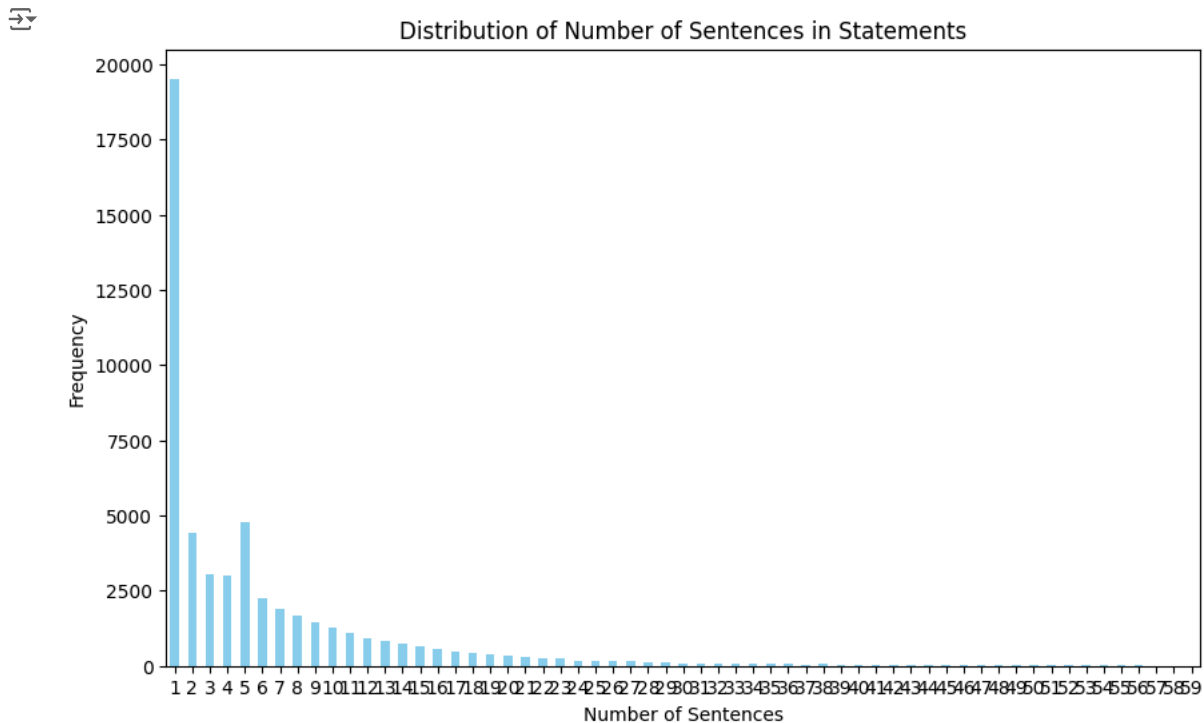
✓ How the sentences frequency is ?

```
# Calculate the number of sentences for each statement
df['num_of_sentences'] = df['statement'].apply(lambda text : len(nltk.sent_tokenize(text)))
```

bar chart for the number of sentences and its frequency

```
# Plot distribution of the number of sentences\n",
plt.figure(figsize=(10, 6))

df[df['num_of_sentences'] < 60]['num_of_sentences'].value_counts().sort_index().plot(kind='bar', color='skyblue')
plt.title('Distribution of Number of Sentences in Statements')
plt.xlabel('Number of Sentences')
plt.ylabel('Frequency')
plt.xticks(rotation=0)
plt.show()
```



✓ 1.1 Lower casing

```
df.sample(1)
```

	statement	status	statement_length	num_of_words	num_of_sentences
26087	I am 21 years old and there is no hope for me....	Suicidal	1180	228	14

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

	statement	status	statement_length	num_of_words	num_of_sentences
8197	i love sleep it is like a teaser trailer for d...	Depression	526	105	5

✓ 1.2 Special Patterns Removing

here we define our function to remove the special patterns which is :

1. links
2. emails
3. markdown text
4. handles
5. punctuation

```
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()
```

applying the special patterns removing

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

✓ 1.3 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)
```

✓ 1.4 Emoji-features Extractions

✓ Convert smiles to words

```
# Define the emoticon map with patterns and corresponding labels
emoticon_map = {
    r':\):': 'happy_face',
    r':D+': 'very_happy_face',
    r':\(+': 'sad_face',
    r':\)\(+': 'crying_face',
    r';\):': 'winking_face',
    r':O': 'surprised_face',
    r':>:\(+': 'angry_face',
    r':\||': 'neutral_face',
    r'<3': 'heart',
    r'B-\):': 'cool_face'
}
```

```
# Function to convert emojis and emoticons to descriptive text
def convert_emojis(text):

    for emoticon, replacement in emoticon_map.items():
        text = re.sub(emoticon, replacement, text)
    text = emoji.demojize(text, delimiters=(" ", ""))
```

```
return text
```

text of emojis only

```
# Function to extract emoticons and replace with descriptive labels
def extract_emojis(text):
    emoji_labels = []
    for pattern, label in emoticon_map.items():
        matches = re.findall(pattern, text)
        emoji_labels.extend([label] * len(matches)) # Add label for each matched emoticon
    return ' '.join(emoji_labels) # Join labels into a single string
```

apply the emoji converter

```
# Apply emoji conversion to the 'statement' column
df['statement_with_emojis'] = df['statement'].apply(convert_emojis)
df['statement'] = df['statement'].apply(convert_emojis)
```

emojis bar chart

```
from collections import Counter
```

```
# Split the emoji labels into individual words and flatten the list
all_emojis = ' '.join(df['statement_with_emojis']).split() # Flatten all labels into a single list
```

```
# Count occurrences of each emoji label
emoji_counts = Counter(all_emojis)
emoji_counts_df = pd.DataFrame(emoji_counts.items(), columns=['Emoji', 'Count'])
```

emoji_counts_df



	Emoji	Count
0	oh	970
1	my	119437
2	gosh	44
3	trouble	564
4	sleeping	904
...
77347	coprophobia	1
77348	strokeliike	1
77349	attacklike	1
77350	tingeling	1
77351	fatherdaughter	1

77352 rows × 2 columns

```
# import seaborn as sns
# import matplotlib.pyplot as plt

# # Set up the plot
# plt.figure(figsize=(12, 8))
# sns.heatmap(emoticon_counts, annot=True, fmt="d", cmap="YlGnBu", cbar_kws={'label': 'Frequency'})

# # Add titles and labels
# plt.title("Distribution of Emoticons by Mental Health Status")
# plt.xlabel("Emoticon (Symbol and Description)")
# plt.ylabel("Mental Health Status")
# plt.xticks(rotation=45, ha="right")
```



```
# plt.tight_layout()
# plt.show()
```

✓ 1.5 Slang Converting

✓ Define the set of slang

```
# Define a custom slang dictionary
slang_dict = {
    "idk": "I don't know",
    "omg": "oh my god",
    "rn": "right now",
    "brb": "be right back",
    "tbh": "to be honest",
    "ikr": "I know, right",
    "smh": "shaking my head",
    "lol": "laughing out loud",
    "lmao": "laughing my ass off",
    "rofl": "rolling on the floor laughing",
    "afaik": "as far as I know",
    "idc": "I don't care",
    "np": "no problem",
    "fyi": "for your information",
    "imo": "in my opinion",
    "pls": "please",
    "thx": "thanks",
    "w/": "with",
    "b4": "before",
    "bc": "because",
    "gr8": "great",
    "btwn": "between",
    "cya": "see you",
    "ty": "thank you",
    "bdy": "birthday",
}
```

✓ Define the expanding functino

```
changes = 0 # Count how many words are expanded
```

```
# Function to expand slang terms based on the custom dictionary
def expand_slang(text):
    words = text.split()
    expanded_words = []
    global changes
    for word in words:
        # Check if the word is in the slang dictionary
        expanded_word = slang_dict.get(word.lower(), word)
        if word.lower() in slang_dict:
            changes=changes+ 1 # Increment count if a word was expanded
            expanded_words.append(expanded_word)

    return ' '.join(expanded_words)
```

✓ Apply the expanding

```
# Apply the slang expansion function to the 'statement' column
df['statement'] = df['statement'].apply(expand_slang)

print(f"Total number of slang words expanded across all rows: {changes}")
```

↻ Total number of slang words expanded across all rows: 4525

1.6 Expand Contractions

```
df['statement'][37456]

↻ 'ih shoepreee disturbing gws idupp mulu fashion make pusink'

apply contraction fix funtionns

df['statement']=df['statement'].apply(contractions.fix)

df['statement'][37456]

↻ 'ih shoepreee disturbing gws idupp mulu fashion make pusink'
```

2.Tokenization & Stemming

Intialize Stanza Piplein

```
# Tokenization functions using NLTK
def word_tokenize(text):
    return nltk.word_tokenize(text)

def sentence_tokenize(text):
    return nltk.sent_tokenize(text)
```

2.1 Tokenization

Words tokenization

```
define word tokenizing funcyion

nltk.download('punkt')

def extract_tokens(text):
    # Tokenize the text into words using NLTK
    words = nltk.word_tokenize(text)
    return words

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

apply extract tokenz function

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

Sample after tokenizing

```
df.sample(2)
```

	statement	status	statemnent_length	num_of_words	num_of_sentences	statement_with_emojis	tokens
46559	what is your personal experience during highs ...	Bipolar	236	39	4	what is your personal experience during highs ...	[what, is, your, personal, experience, during,...
10017	i am so so tired of this	Suicidal	1172	222	10	i am so so tired of this after	[i, am, so, so, tired, of,

2.2 Stemming

Intializing the steemer

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

# Function to stem tokens
def tokens_stem(tokens):
    # Extract and stem each token
    stemmed_tokens = [stemmer.stem(token) for token in tokens]
    return ' '.join(stemmed_tokens)
```

Apply the stemming Function

```
df['tokens_stemmed'] = df['tokens'].apply(tokens_stem)
```

Sample

```
# Display sample of the DataFrame
df.sample(5)
```

	statement	status	statemnent_length	num_of_words	num_of_sentences	statement_with_emojis	tokens	tokens_stemmed
13760	small backstory i used to be a counselor on th...	Suicidal	2755	528	34	small backstory i used to be a counselor on th...	[small, backstory, i, used, to, be, a, counsel...	small backstori i use to be a counselor on the...
24820	i do not know what to do or how to go on i jus...	Suicidal	669	147	20	i do not know what to do or how to go on i jus...	[i, do, not, know, what, to, do, or, how, to, ...	i do not know what to do or how to go on i jus...
6952	uh my account is green or not	Normal	29	7	1	uh my account is green or not	[uh, my, account, is, green, or, not]	uh my account is green or not

2.3 word counts

```
colors = ['#16325B', '#227B94', '#7887D0', '#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: ' '.join(x)).tolist())

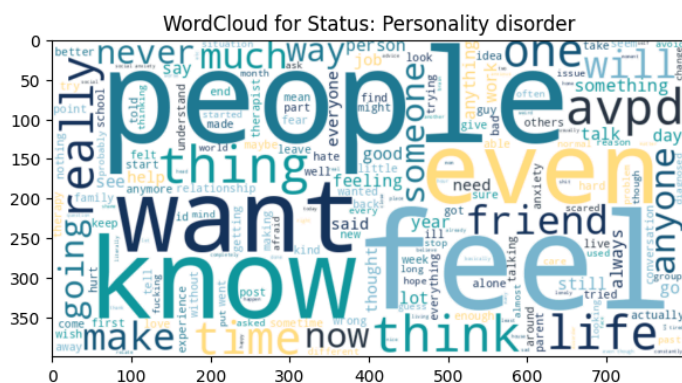
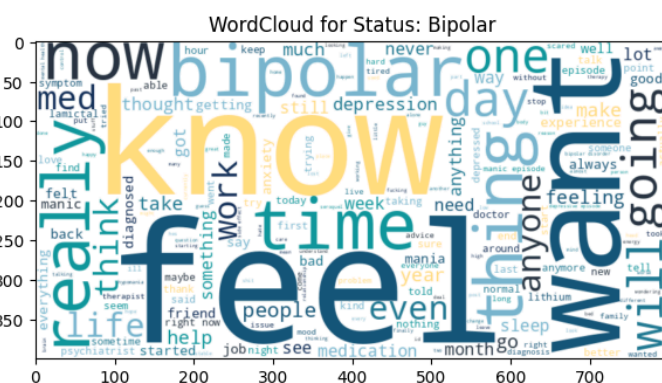
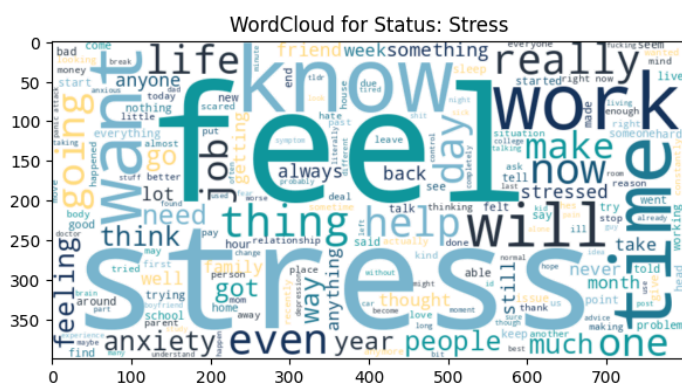
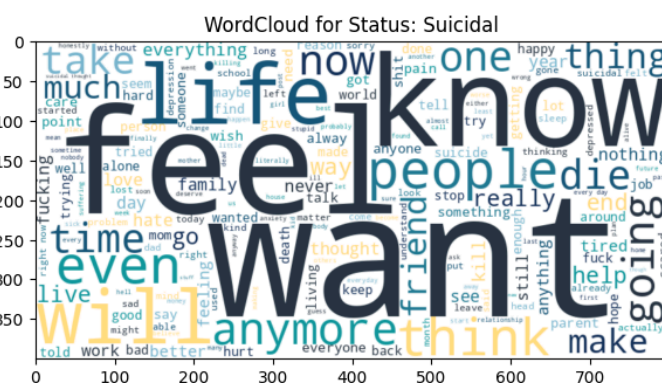
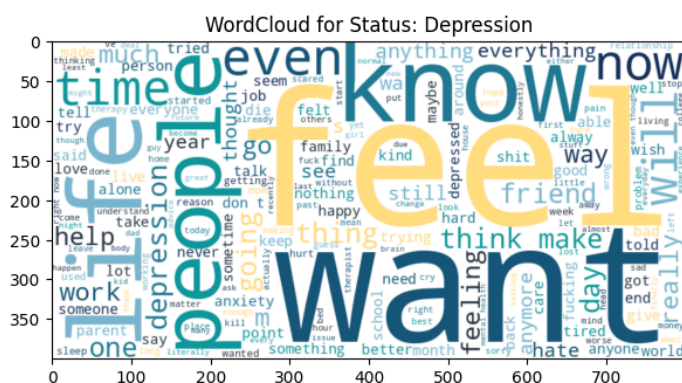
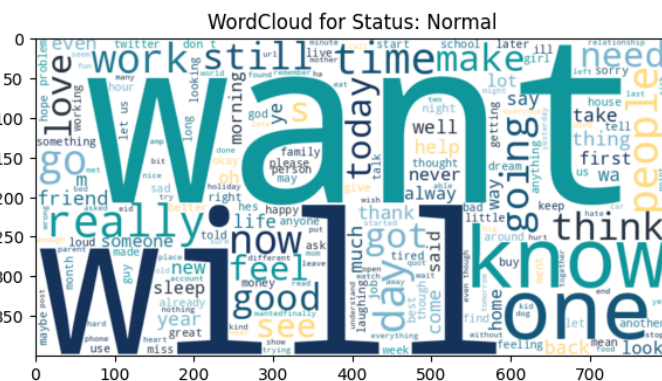
    # Generate the WordCloud
    wordcloud = WordCloud(width=800, height=400, background_color='white', color_func=color_func).generate(tokens_data)

    # Plot the WordCloud in a subplot
```

```
axes = plt.subplot(len(statuses) // 2 + 1, 2, i + 1) # Adjust number of rows and columns dynamically
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()
```



✓ 3. Part-Of-Speech (POS) Tagging

✓ 3.1 Extract POS Taggs

✓ Nlp

```
nlk.download('averaged_perceptron_tagger')
```

```
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data] /root/nltk_data...
[nltk_data] Unzipping taggers/averaged_perceptron_tagger.zip.
True
```

```
nlp = stanza.Pipeline(lang='en', processors='tokenize,mwt,pos')
```

```
[nltk_data] INFO:stanza:Checking for updates to resources.json in case models have been updated. Note: this behavior can be turned off with downloa
Downloading https://raw.githubusercontent.com/stanfordnlp/stanza-resources/main/resources_1.9.0.json: 0%| ...
INFO:stanza:Downloaded file to /root/stanza_resources/resources.json
Downloading https://huggingface.co/stanfordnlp/stanza-en/resolve/v1.9.0/models/tokenize/combined.pt: 0%| ...
Downloading https://huggingface.co/stanfordnlp/stanza-en/resolve/v1.9.0/models/mwt/combined.pt: 0%| ...
Downloading https://huggingface.co/stanfordnlp/stanza-en/resolve/v1.9.0/models/pos/combined_charlm.pt: 0%| ...
Downloading https://huggingface.co/stanfordnlp/stanza-en/resolve/v1.9.0/models/forward_charlm/1billion.pt: 0...
Downloading https://huggingface.co/stanfordnlp/stanza-en/resolve/v1.9.0/models/backward_charlm/1billion.pt: ...
Downloading https://huggingface.co/stanfordnlp/stanza-en/resolve/v1.9.0/models/pretrain/conll17.pt: 0%| ...
INFO:stanza:Loading these models for language: en (English):
=====
| Processor | Package |
-----
| tokenize | combined |
| mwt | combined |
| pos | combined_charlm |
=====

INFO:stanza:Using device: cuda
INFO:stanza:Loading: tokenize
INFO:stanza:Loading: mwt
INFO:stanza:Loading: pos
INFO:stanza:Done loading processors!
```

✓ define function to extract pos

```
import nltk
```

```
def extract_pos_tokens(text, pos_tag_prefix):
```

```
# Tokenize the input text
tokens = nltk.word_tokenize(text)
# Get the POS tags
pos_tags = nltk.pos_tag(tokens)

# Extract tokens based on POS tag prefix
tokens_filtered = [word for word, pos in pos_tags if pos.startswith(pos_tag_prefix)]

return ' '.join(tokens_filtered)
```

✓ Extract Verbs

```
df['tokens_verbs'] = df['statement'].apply(lambda x: extract_pos_tokens(x, 'V'))
```

✓ Extract Nouns

```
df['tokens_nouns'] = df['statement'].apply(lambda x: extract_pos_tokens(x, 'N'))
```

✓ Extrac Adjectives

```
df['tokens_adj'] = df['statement'].apply(lambda x: extract_pos_tokens(x, 'J'))
```

✓ Sample

```
df.sample(2)
```

	statement	status	statemnent_length	num_of_words	num_of_sentences	statement_with_emojis	tokens	tokens_stemmed	tokens,
3400	the first one still sucks if you continue the ...	Normal	59	11	1	the first one still sucks if you continue the ...	[the, first, one, still, sucks, if, you, conti...	the first one still suck if you continu the pr...	o
	i catch myself						[i, catch,	i catch myself	catch t

✓ 4. Dependency Parsing

✓ Initialize Stanza pipelin

```
# Initialize Stanza pipeline with dependency parsing
nlp = stanza.Pipeline(lang='en', processors='tokenize,pos,lemma,depparse')
```

```

INFO:stanza:Checking for updates to resources.json in case models have been updated. Note: this behavior can be turned off with downloa
Downloading https://raw.githubusercontent.com/stanfordnlp/stanza-resources/main/resources\_1.9.0.json: 0%| ...
INFO:stanza:Downloaded file to /root/stanza_resources/resources.json
WARNING:stanza:Language en package default expects mwt, which has been added
Downloading https://huggingface.co/stanfordnlp/stanza-en/resolve/v1.9.0/models/tokenize/combined.pt: 0%| ...
Downloading https://huggingface.co/stanfordnlp/stanza-en/resolve/v1.9.0/models/mwt/combined.pt: 0%| ...
Downloading https://huggingface.co/stanfordnlp/stanza-en/resolve/v1.9.0/models/pos/combined\_charlm.pt: 0%| ...
Downloading https://huggingface.co/stanfordnlp/stanza-en/resolve/v1.9.0/models/lemma/combined\_nocharlm.pt: 0...
Downloading https://huggingface.co/stanfordnlp/stanza-en/resolve/v1.9.0/models/depparse/combined\_charlm.pt: ...
Downloading https://huggingface.co/stanfordnlp/stanza-en/resolve/v1.9.0/models/forward\_charlm/1billion.pt: 0...
Downloading https://huggingface.co/stanfordnlp/stanza-en/resolve/v1.9.0/models/pretrain/conll17.pt: 0%| ...
Downloading https://huggingface.co/stanfordnlp/stanza-en/resolve/v1.9.0/models/backward\_charlm/1billion.pt: ...
INFO:stanza:Loading these models for language: en (English):
=====
| Processor | Package |
|-----|
| tokenize | combined |
| mwt      | combined |
| pos      | combined_charlm |
| lemma    | combined_nocharlm |
| depparse | combined_charlm |
=====

INFO:stanza:Using device: cuda
INFO:stanza:Loading: tokenize
INFO:stanza:Loading: mwt
INFO:stanza:Loading: pos
INFO:stanza:Loading: lemma
INFO:stanza:Loading: depparse
INFO:stanza:Done loading processors!

```

▼ extract dependencies function

```

# Function to extract dependency relations
def extract_dependencies(text):
    doc = nlp(text)
    dependencies = []
    for sentence in doc.sentences:
        for word in sentence.words:
            # Append the word, its dependency relation, and its head word
            dependencies.append((word.text, word.deprel, sentence.words[word.head - 1].text if word.head > 0 else "ROOT"))
    return dependencies

```

consume alot of memory OutOfMemoryError Traceback (most recent call last) [/usr/local/lib/python3.10/dist-packages/stanza/pipeline/depparse_processor.py](#) in process(self, document) 64 for i, b in enumerate(batch): --> 65 preds += self.trainer.predict(b) 66 if batch.data_orig_idx is not None:

22 frames OutOfMemoryError: CUDA out of memory. Tried to allocate 12.35 GiB. GPU 0 has a total capacity of 14.75 GiB of which 12.09 GiB is free. Process 4177 has 2.65 GiB memory in use. Of the allocated memory 1.90 GiB is allocated by PyTorch, and 634.24 MiB is reserved by PyTorch but unallocated. If reserved but unallocated memory is large try setting PYTORCH_CUDA_ALLOC_CONF=expandable_segments:True to avoid fragmentation. See documentation for Memory Management (<https://pytorch.org/docs/stable/notes/cuda.html#environment-variables>)

```

# def extract_important_words(text):
#     doc = nlp(text)
#     important_words = []

#     for sentence in doc.sentences:
#         for word in sentence.words:
#             # Select words based on dependency relation
#             if word.deprel in {'root', 'nsubj', 'dobj', 'amod', 'advmod'}:
#                 important_words.append(word.text)

#     return ' '.join(important_words)

import spacy

```



```

nlp = spacy.load('en_core_web_sm')

def extract_important_words(text):
    doc = nlp(text)
    important_words = set() # Use a set to store unique words

    for token in doc:
        # Extract the head word if it has relevant dependency relations
        if token.dep_ in {'nsubj', 'dobj', 'amod', 'advmod', 'pobj', 'ROOT'}:
            important_words.add(token.text) # Add the current token
            important_words.add(token.head.text) # Add the head word

        # Include the children of the head word for additional context
        for child in token.head.children:
            important_words.add(child.text)

    return ' '.join(important_words)

```

✓ Applying dep parse

```

# Apply the function to extract important words
df['important_words'] = df['statement'].apply(extract_important_words)

```

✓ result

```

# Sample output
df[['important_words', 'statement']].sample(3)

```



	important_words	statement
38152	don time general all what with connect last wh...	i don t remember the last time i wa really abl...
20394	god 17 to it killed and kind should am so ever...	so this might be kind of long but here gose i ...
44553	snowing tennessee it too in is	it is snowing in tennessee too

✓ Vectorizing using TF-IDF

✓ About VEcutorizing Approach

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:

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

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.

✓ Initialize separate TF-IDF Vectorizer

Initialize separate TF-IDF Vectorizer for verbs and nouns and adj, stemm

```
vectorizer_emojis = TfidfVectorizer(max_features=100, stop_words='english')

vectorizer_imp = TfidfVectorizer(ngram_range=(1, 2), max_features=10000, stop_words='english')

vectorizer_stemmes = TfidfVectorizer(ngram_range=(1, 2), max_features=2000, stop_words='english')

vectorizer_verbs = TfidfVectorizer(ngram_range=(1, 2), max_features=1000, stop_words='english')

vectorizer_nouns = TfidfVectorizer(ngram_range=(1, 2), max_features=1000, stop_words='english')

vectorizer_adjectives = TfidfVectorizer(ngram_range=(1, 2), max_features=1000, stop_words='english')
```

✓ Extract Verb, Nouns , ... Vectors

✓ important word

```
X_imp_tfidf = vectorizer_emojis.fit_transform(df['important_words'])
```

✓ Emojis

```
# Emojis
X_emojis_tfidf = vectorizer_emojis.fit_transform(df['statement_with_emojis'])
```

✓ stemmes

```
# Adjectives
X_stemm_tfidf = vectorizer_stemmes.fit_transform(df['tokens_stemmed'].astype(str))
```

✓ verbs

```
# Verbs
X_verbs_tfidf = vectorizer_verbs.fit_transform(df['tokens_verbs'].astype(str))
```

✓ nouns

```
# Nouns
X_nouns_tfidf = vectorizer_nouns.fit_transform(df['tokens_nouns'].astype(str))
```

✓ Adjectives

```
# Adjectives
X_adjectives_tfidf = vectorizer_adjectives.fit_transform(df['tokens_adj'].astype(str))
```

✓ comdine Vectors

```
# Combine all TF-IDF features into one matrix
# X_combined_tfidf = hstack([2*X_verbs_tfidf, X_nouns_tfidf, 0.5 * X_adjectives_tfidf, 0.5*X_emojis_tfidf,2*X_stemm_tfidf])
X_combined_tfidf = hstack([X_imp_tfidf,0.8*X_verbs_tfidf, 1.2*X_nouns_tfidf,X_stemm_tfidf])
print("Total number of combined features:", X_combined_tfidf.shape[1])
```

↗ Total number of combined features: 4100

✓ Feature Selection

✓ Splitting

```
# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X_combined_tfidf, df['status'], test_size=0.2, random_state=42)
```

✓ Select K Best

```
# Arrays to store results
feature_counts = []
train_accuracies = []
test_accuracies = []

# Iterate over different values of k
for k in range(100, X_train.shape[1], 200):
    # Select top k features using mutual information as the score function
    selector = SelectKBest(score_func=chi2, k=k)

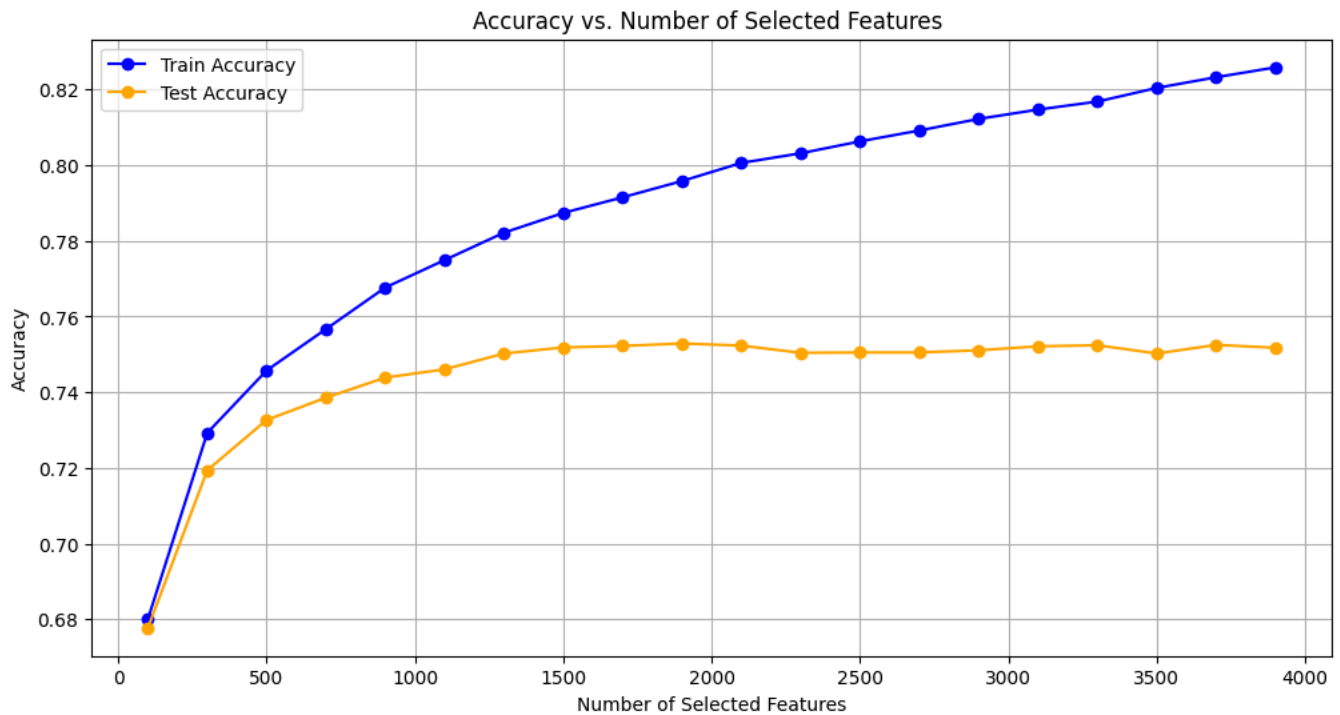
    # Fit and transform the training set
    X_train_selected = selector.fit_transform(X_train, y_train)
    X_test_selected = selector.transform(X_test)

    # Train a logistic regression model
    model = LogisticRegression(max_iter=600, random_state=42)
    model.fit(X_train_selected, y_train)

    # Record the number of features and accuracies
    feature_counts.append(k)
    train_accuracies.append(accuracy_score(y_train, model.predict(X_train_selected)))
    test_accuracies.append(accuracy_score(y_test, model.predict(X_test_selected)))
```

✓ Plot

```
# Plot the accuracy vs. number of features
plt.figure(figsize=(12, 6))
plt.plot(feature_counts, train_accuracies, label="Train Accuracy", marker="o", color="blue")
plt.plot(feature_counts, test_accuracies, label="Test Accuracy", marker="o", color="orange")
plt.xlabel("Number of Selected Features")
plt.ylabel("Accuracy")
plt.title("Accuracy vs. Number of Selected Features ")
plt.legend()
plt.grid()
plt.show()
```



✓ Dataset Splitting, Encoding and Staging

```
selector = SelectKBest(score_func=chi2, k=2000)

# Fit and transform the training set
X_train_selected = selector.fit_transform(X_train, y_train)
X_test_selected = selector.transform(X_test)

# Split the vectorized data into train and test sets
#X_train_full, X_test, y_train_full, y_test = train_test_split(X_combined_tfidf, df['status'], test_size=0.2, random_state=42)
X_train_full, X_test, y_train_full, y_test = X_train_selected, X_test_selected, y_train, y_test
```

✓ Encode LAbles and slpit status

✓ Encode LAbles

```
# Encode labels for binary classification (Stage 1)
binary_encoder = LabelEncoder()
y_train_full_binary = binary_encoder.fit_transform(y_train_full.apply(lambda x: 'Normal' if x == 'Normal' else 'Non-Normal'))
y_test_binary = binary_encoder.transform(y_test.apply(lambda x: 'Normal' if x == 'Normal' else 'Non-Normal'))

# Encode labels for multi-class classification (Stage 2)
multi_encoder = LabelEncoder()
y_train_full_multi = multi_encoder.fit_transform(y_train_full[y_train_full != 'Normal']) # Only for "Non-Normal" classes
y_test_multi = multi_encoder.transform(y_test[y_test != 'Normal']) # Only for "Non-Normal" classes
```

✓ split to Stages

```
# Split train data for Stage 1 and Stage 2
X_train_tfidf_stage1 = X_train_full
X_train_tfidf_stage2 = X_train_full[y_train_full != 'Normal']
y_train_tfidf_stage1 = y_train_full_binary
y_train_tfidf_stage2 = y_train_full_multi

X_test_tfidf_stage1 = X_test
X_test_tfidf_stage2 = X_test[y_test != 'Normal']
```

✓ OverSampling using SMOTE

```
# Apply SMOTE for oversampling on Stage 2
smote = SMOTE(random_state=42)

#X_train_tfidf_stage2, y_train_stage2 = smote.fit_resample(X_train_tfidf_stage2, y_train_tfidf_stage2)
#X_train_tfidf_stage1, y_train_stage1 = smote.fit_resample(X_train_tfidf_stage1, y_train_tfidf_stage1)
```

✓ *§ Normal Statement Classification*

✓ What we will do

Stage 1

1.1 Logistic regression classifier

2.2 neural network classifier

2.3 chosssing the best classifier

✓ 1.1 Stage 1: SVM

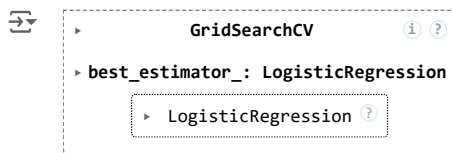
✓ Grid Search Param Definie

```
# Define the binary classifier
clf = LogisticRegression(max_iter=1000, random_state=42)

# Define the parameter grid for GridSearchCV
param_grid = {
    'C': [0.01, 0.1, 0.8, 1, 2, 10, 100], # Regularization strength
    'solver': ['lbfgs', 'liblinear'], # Optimization algorithms
    'penalty': ['l2'], # Regularization type
}
```

✓ Grid Search Fit

```
# Perform Grid Search with cross-validation
grid_search = GridSearchCV(clf, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train_tfidf_stage1, y_train_tfidf_stage1)
```



✓ best PARarms

```
# Get the best parameters and accuracy
best_params = grid_search.best_params_
```

```
best_accuracy = grid_search.best_score_
```

```
print(f"Best parameters: {best_params}")
print(f"Best accuracy: {best_accuracy}")
```

```
Best parameters: {'C': 2, 'penalty': 'l2', 'solver': 'liblinear'}
Best accuracy: 0.9390185211034092
```

```
logistic1_classifier= SVC(random_state=42,C=0.7)
logistic1_classifier.fit(X_train_tfidf_stage1, y_train_tfidf_stage1)
```

```
SVC
SVC(C=0.7, random_state=42)
```

```
#logistic1_classifier= grid_search.best_estimator_
```

classification report for train

```
# Evaluate binary classifier
y_pred_stage1 = logistic1_classifier.predict(X_train_tfidf_stage1)
print("Stage 1 (Binary Classification) - Normal vs Non-Normal")
print("Accuracy:", accuracy_score(y_train_tfidf_stage1, y_pred_stage1))
print(classification_report(y_train_tfidf_stage1, y_pred_stage1))
print("Confusion Matrix:\n", confusion_matrix(y_train_tfidf_stage1, y_pred_stage1))
```

```
Stage 1 (Binary Classification) - Normal vs Non-Normal
Accuracy: 0.9697465831435079
```

	precision	recall	f1-score	support
0	0.98	0.98	0.98	29109
1	0.95	0.95	0.95	13035
accuracy			0.97	42144
macro avg	0.97	0.96	0.96	42144
weighted avg	0.97	0.97	0.97	42144

```
Confusion Matrix:
[[28505  604]
 [ 671 12364]]
```

classification report

```
# Evaluate binary classifier
y_pred_stage1 = logistic1_classifier.predict(X_test_tfidf_stage1)
print("Stage 1 (Binary Classification) - Normal vs Non-Normal")
print("Accuracy:", accuracy_score(y_test_binary, y_pred_stage1))
print(classification_report(y_test_binary, y_pred_stage1))
print("Confusion Matrix:\n", confusion_matrix(y_test_binary, y_pred_stage1))
```

```
Stage 1 (Binary Classification) - Normal vs Non-Normal
Accuracy: 0.940115782480782
```

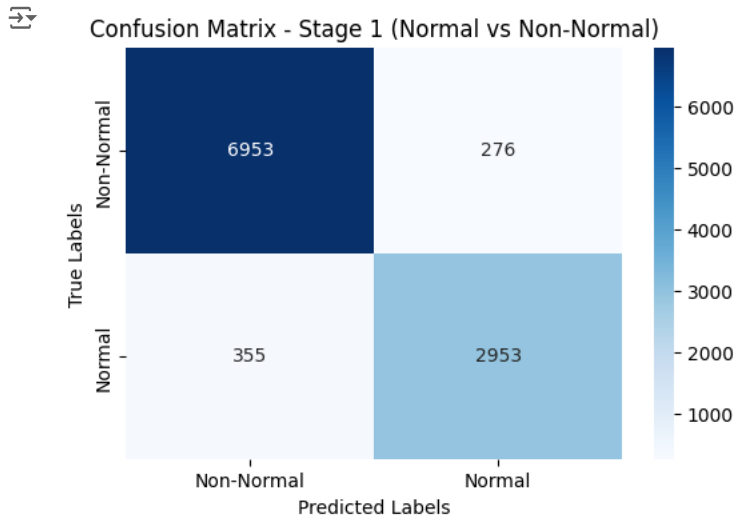
	precision	recall	f1-score	support
0	0.95	0.96	0.96	7229
1	0.91	0.89	0.90	3308
accuracy			0.94	10537
macro avg	0.93	0.93	0.93	10537
weighted avg	0.94	0.94	0.94	10537

```
Confusion Matrix:
[[6953  276]
 [ 355 2953]]
```

Heat Map

```
# Calculate confusion matrix for Stage 1
conf_matrix_stage1 = confusion_matrix(y_test_binary, y_pred_stage1)

# Plot heatmap for Stage 1
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix_stage1, annot=True, fmt="d", cmap="Blues", xticklabels=binary_encoder.classes_, yticklabels=binary_encoder.classes_)
plt.title("Confusion Matrix - Stage 1 (Normal vs Non-Normal)")
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.show()
```



1.2 Stage 1: Neural Network

Neural Network Layers

```
# Define the layers in an array
layers = [
    Dense(units=256, activation='relu', input_shape=(X_train_tfidf_stage1.shape[1],), kernel_regularizer=l2(0.01)), # Input Layer
    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

    Dense(units=2, activation='softmax') # Output Layer
]

# Initialize the Sequential model
model = Sequential(layers)

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

Model Training

```
# Train the model on training data with validation on the test data
history = model.fit(X_train_tfidf_stage1.toarray(),
                    y_train_tfidf_stage1,
                    epochs=20,
                    batch_size=64,
                    validation_split=0.3)
```



```

Epoch 1/20
461/461 ————— 10s 18ms/step - accuracy: 0.8831 - loss: 2.2164 - val_accuracy: 0.8464 - val_loss: 0.5045
Epoch 2/20
461/461 ————— 7s 12ms/step - accuracy: 0.9238 - loss: 0.3385 - val_accuracy: 0.9242 - val_loss: 0.2951
Epoch 3/20
461/461 ————— 11s 14ms/step - accuracy: 0.9291 - loss: 0.2835 - val_accuracy: 0.9274 - val_loss: 0.2728
Epoch 4/20
461/461 ————— 6s 12ms/step - accuracy: 0.9324 - loss: 0.2679 - val_accuracy: 0.9276 - val_loss: 0.2689
Epoch 5/20
461/461 ————— 10s 12ms/step - accuracy: 0.9326 - loss: 0.2597 - val_accuracy: 0.9295 - val_loss: 0.2609
Epoch 6/20
461/461 ————— 8s 17ms/step - accuracy: 0.9377 - loss: 0.2432 - val_accuracy: 0.9297 - val_loss: 0.2512
Epoch 7/20
461/461 ————— 8s 12ms/step - accuracy: 0.9373 - loss: 0.2348 - val_accuracy: 0.9318 - val_loss: 0.2477
Epoch 8/20
461/461 ————— 12s 16ms/step - accuracy: 0.9376 - loss: 0.2299 - val_accuracy: 0.9304 - val_loss: 0.2446
Epoch 9/20
461/461 ————— 11s 17ms/step - accuracy: 0.9348 - loss: 0.2351 - val_accuracy: 0.9301 - val_loss: 0.2433
Epoch 10/20
461/461 ————— 9s 14ms/step - accuracy: 0.9377 - loss: 0.2276 - val_accuracy: 0.9312 - val_loss: 0.2449
Epoch 11/20
461/461 ————— 9s 12ms/step - accuracy: 0.9392 - loss: 0.2229 - val_accuracy: 0.9235 - val_loss: 0.2500
Epoch 12/20
461/461 ————— 8s 16ms/step - accuracy: 0.9387 - loss: 0.2229 - val_accuracy: 0.9325 - val_loss: 0.2390
Epoch 13/20
461/461 ————— 8s 12ms/step - accuracy: 0.9385 - loss: 0.2221 - val_accuracy: 0.9276 - val_loss: 0.2475
Epoch 14/20
461/461 ————— 11s 14ms/step - accuracy: 0.9399 - loss: 0.2189 - val_accuracy: 0.9317 - val_loss: 0.2372
Epoch 15/20
461/461 ————— 12s 17ms/step - accuracy: 0.9413 - loss: 0.2167 - val_accuracy: 0.9295 - val_loss: 0.2369
Epoch 16/20
461/461 ————— 6s 12ms/step - accuracy: 0.9429 - loss: 0.2145 - val_accuracy: 0.9328 - val_loss: 0.2387
Epoch 17/20
461/461 ————— 10s 13ms/step - accuracy: 0.9412 - loss: 0.2159 - val_accuracy: 0.9337 - val_loss: 0.2336
Epoch 18/20
461/461 ————— 11s 15ms/step - accuracy: 0.9405 - loss: 0.2149 - val_accuracy: 0.9333 - val_loss: 0.2333
Epoch 19/20
461/461 ————— 11s 16ms/step - accuracy: 0.9437 - loss: 0.2115 - val_accuracy: 0.9325 - val_loss: 0.2362
Epoch 20/20
461/461 ————— 9s 13ms/step - accuracy: 0.9444 - loss: 0.2095 - val_accuracy: 0.9300 - val_loss: 0.2399

```

✓ Classification report on train

```

# Evaluate binary classifier
y_pred_stage1 = model.predict(X_train_tfidf_stage1)
y_pred_stage1 = y_pred_stage1.argmax(axis=1) # Convert probabilities to class predictions

print("Stage 1 (Binary Classification) - Normal vs Non-Normal")
print("Accuracy:", accuracy_score(y_train_tfidf_stage1, y_pred_stage1))
print(classification_report(y_train_tfidf_stage1, y_pred_stage1))
print("Confusion Matrix:\n", confusion_matrix(y_train_tfidf_stage1, y_pred_stage1))

```

```

1317/1317 ————— 4s 3ms/step
Stage 1 (Binary Classification) - Normal vs Non-Normal
Accuracy: 0.9479166666666666

```

	precision	recall	f1-score	support
0	0.97	0.96	0.96	29109
1	0.90	0.93	0.92	13035
accuracy			0.95	42144
macro avg	0.94	0.94	0.94	42144
weighted avg	0.95	0.95	0.95	42144

```

Confusion Matrix:
[[27813 1296]
 [ 899 12136]]

```

✓ Confusion Matrix

```

# Make predictions on the test set
y_pred_prob = model.predict(X_test_tfidf_stage1.toarray())

```

```
y_pred = y_pred_prob.argmax(axis=1) # Convert probabilities to class predictions
```

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

```
# Compute the confusion matrix
```

```
conf_matrix_nn = confusion_matrix(y_test_binary,y_pred)
```

```
# Print classification report
print("Confusion Matrix:\n",)
print( conf_matrix_nn)
```

330/330 1s 3ms/step

```
Accuracy: 0.931099933567429
Confusion Matrix:
```

```
[[6814  415]
 [ 311 2997]]
```

```
print(classification_report(y_test_binary, y_pred))
```

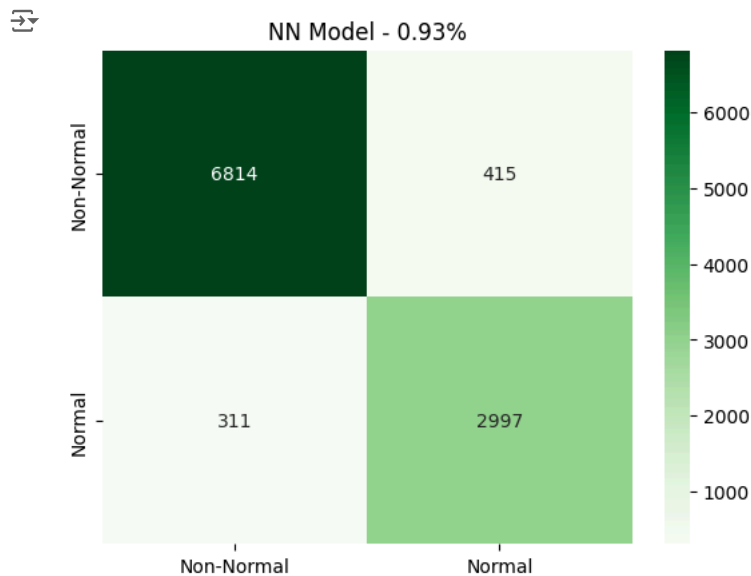
```
precision    recall  f1-score   support

     0       0.96     0.94     0.95       7229
     1       0.88     0.91     0.89       3308

 accuracy          0.93       10537
  macro avg       0.92     0.92     0.92       10537
 weighted avg     0.93     0.93     0.93       10537
```

Heat Map

```
ax = sns.heatmap(conf_matrix_nn, annot = True, fmt='d', cmap='Greens', xticklabels=binary_encoder.classes_, yticklabels=binary_encoder.class
ax.set_title(f'NN Model - {accuracy_nn:.2}%')
plt.show()
```



Model Loss and Accuracy

```
plt.figure(figsize=(12, 5))
```

```
# Plot training & validation loss values
plt.subplot(1, 2, 1) # 1 row, 2 columns, 1st subplot
```

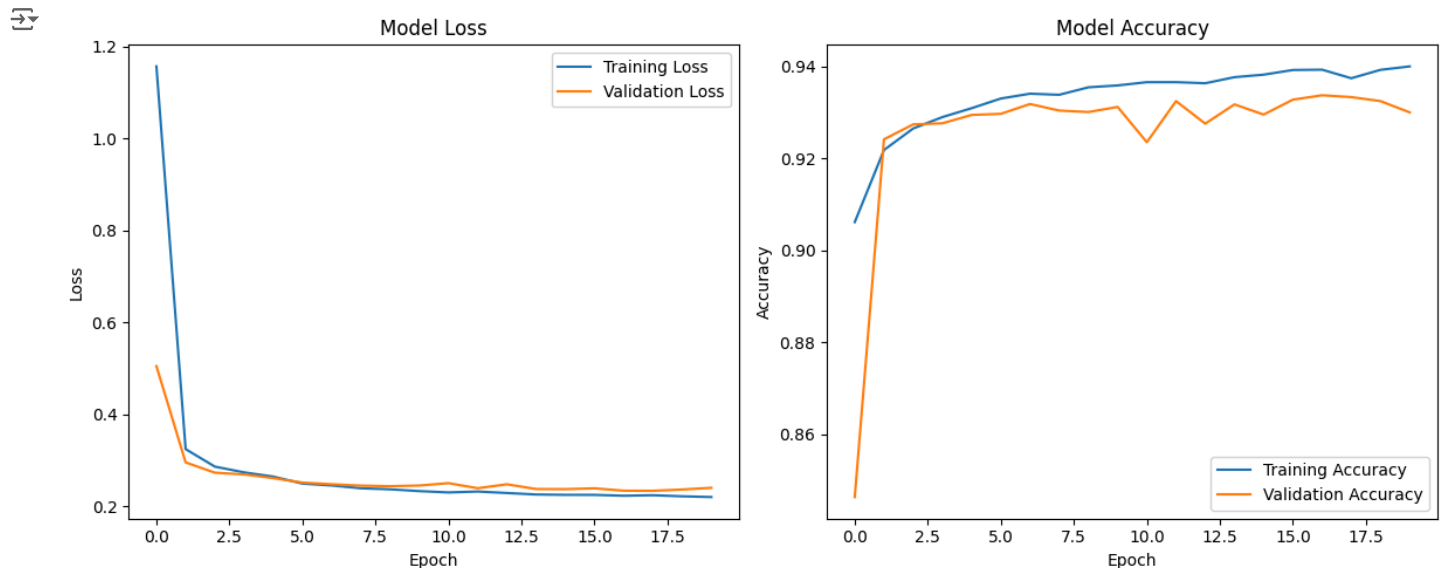
```

plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(loc='upper right')

# Plot training & validation accuracy values
plt.subplot(1, 2, 2) # 1 row, 2 columns, 2nd subplot
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')

plt.tight_layout() # Adjusts subplots to fit in the figure area.
plt.show() # Display the plots

```



✓ 1.3 Best Classifier for Stage 1

we chose the best classifier for this stage

```
binary_classifier = logistic1_classifier
```

✓ *§ Non Normal Statement Classification*

✓ 2.1 SVM Stage 2

✓ Grid Search Params

```

# Define the binary classifier
clf = LogisticRegression(max_iter=1000, random_state=42)

# Define the parameter grid for GridSearchCV
param_grid = {
    'C': [0.03, 0.01, 0.02], # Regularization strength
    'solver': ['lbfgs'], # Optimization algorithms
    'penalty': ['l2'], # Regularization type
}

```

```
# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=clf, param_grid=param_grid, scoring='accuracy', cv=5, n_jobs=-1, verbose=1)
```

✓ Train the Model

```
# Fit the grid search on the training data
grid_search.fit(X_train_tfidf_stage2, y_train_tfidf_stage2)
```

→ Fitting 5 folds for each of 3 candidates, totalling 15 fits

```
GridSearchCV
└─ best_estimator_: LogisticRegression
   └─ LogisticRegression
```

✓ Best Classifier

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

→ Best hyperparameters found: {'C': 2, 'penalty': 'l2', 'solver': 'liblinear'}

```
# Train multi-class classifier for Stage 2
logistic_classifier = best_clf
logistic_classifier = SVC(random_state=42, kernel='linear', C=0.3)

logistic_classifier.fit(X_train_tfidf_stage2, y_train_tfidf_stage2)
```

```
SVC
SVC(C=0.3, kernel='linear', random_state=42)
```

✓ Classification report on training

```
# Evaluate binary classifier
y_pred_stage2 = logistic_classifier.predict(X_train_tfidf_stage2)
print("Stage 1 (Binary Classification) - Normal vs Non-Normal")
print("Accuracy:", accuracy_score(y_train_tfidf_stage2, y_pred_stage2))
print(classification_report(y_train_tfidf_stage2, y_pred_stage2))
print("Confusion Matrix:\n", confusion_matrix(y_train_tfidf_stage2, y_pred_stage2))
```

→ Stage 1 (Binary Classification) - Normal vs Non-Normal

```
Accuracy: 0.748565735683122
```

	precision	recall	f1-score	support
0	0.83	0.84	0.84	3086
1	0.91	0.70	0.79	2250
2	0.71	0.82	0.76	12388
3	0.97	0.42	0.58	840
4	0.81	0.52	0.63	2051
5	0.73	0.72	0.72	8494
accuracy			0.75	29109
macro avg	0.83	0.67	0.72	29109
weighted avg	0.76	0.75	0.75	29109

```
Confusion Matrix:
[[ 2599  18  349   1   64   55]
 [  73 1567  481   2   29   98]
 [ 182   89 10130   3   81 1903]
 [  17   3   393  351  16   60]
```

```
[ 204   16   617    1 1057   156]
[   39   26  2288    2   53  6086]]
```

Classification Report

```
# Evaluate multi-class classifier
y_pred_stage2 = logistic_classifier.predict(X_test_tfidf_stage2)
print("Stage 2 (Multi-Class Classification) - Specific Conditions")
print("Accuracy:", accuracy_score(y_test_multi, y_pred_stage2))
print(classification_report(y_test_multi, y_pred_stage2))
conf_matrix_stage2= confusion_matrix(y_test_multi, y_pred_stage2)
print("Confusion Matrix:\n",conf_matrix_stage2)
```

Stage 2 (Multi-Class Classification) - Specific Conditions

Accuracy: 0.7018951445566468

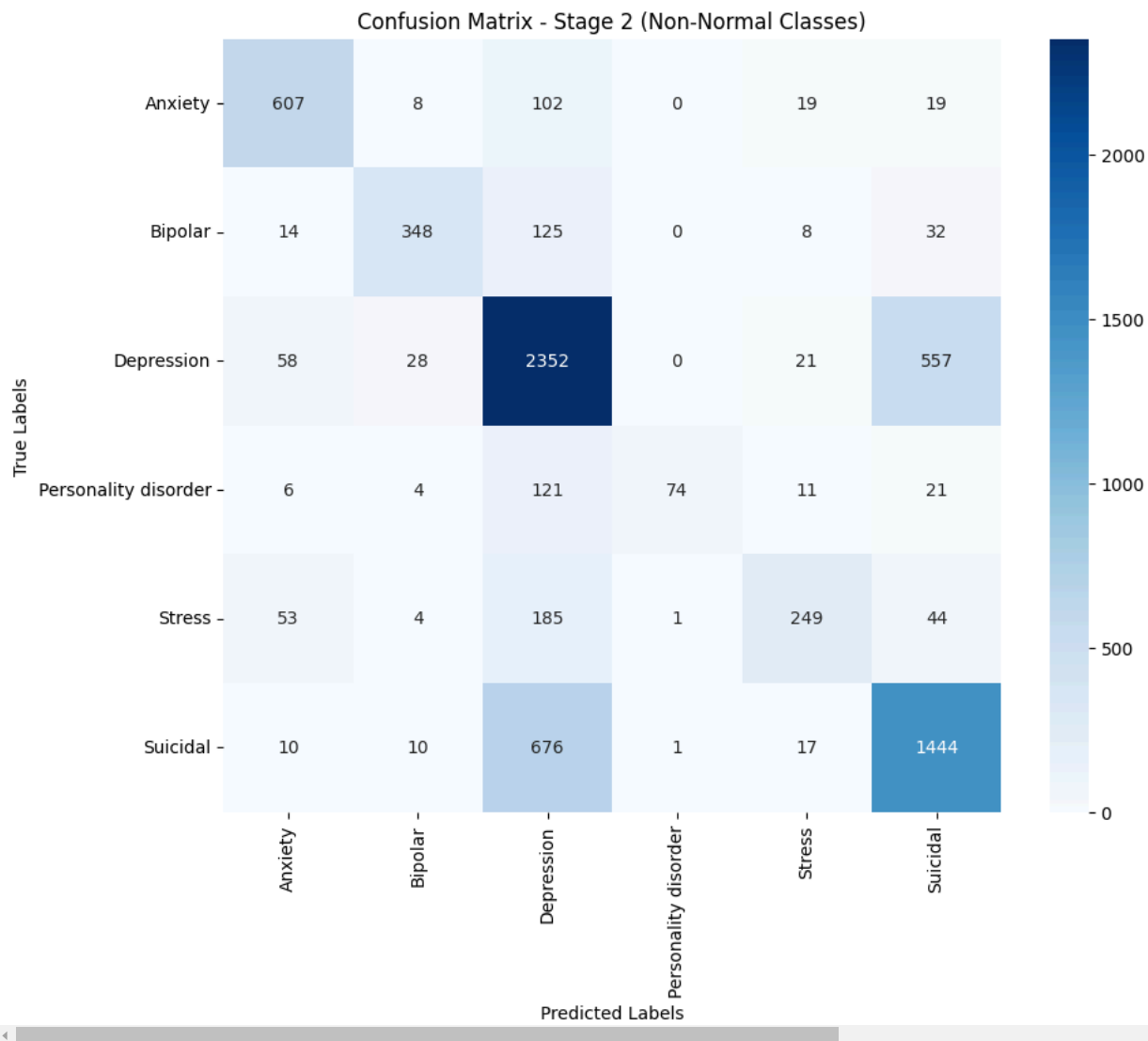
	precision	recall	f1-score	support
0	0.81	0.80	0.81	755
1	0.87	0.66	0.75	527
2	0.66	0.78	0.72	3016
3	0.97	0.31	0.47	237
4	0.77	0.46	0.58	536
5	0.68	0.67	0.68	2158
accuracy			0.70	7229
macro avg	0.79	0.62	0.67	7229
weighted avg	0.72	0.70	0.70	7229

Confusion Matrix:

```
[[ 607    8  102    0   19   19]
 [  14  348  125    0    8   32]
 [  58   28 2352    0   21  557]
 [    6    4  121   74   11   21]
 [   53    4  185    1  249   44]
 [   10   10  676    1   17 1444]]
```

Heat Map

```
# Plot heatmap for Stage 2
plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix_stage2, annot=True, fmt="d", cmap="Blues", xticklabels=multi_encoder.classes_, yticklabels=multi_encoder.classes_)
plt.title("Confusion Matrix - Stage 2 (Non-Normal Classes)")
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.show()
```



2.2 Neural Network Stage 2

Network Layers

```
from tensorflow.keras.layers import Conv1D,LSTM, MaxPooling1D, Flatten, BatchNormalization, Dense, Dropout
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.regularizers import l2
```

```
# Define the layers in an array
layers = [
```

```
    Dense(units=64, activation='relu', input_shape=(X_train_tfidf_stage2.shape[1,]),kernel_regularizer=l2(0.01)), # Input Layer
    BatchNormalization(),
    Dropout(rate=0.3), # Dropout Layer 1
    # Dense(units=30, activation='relu', input_shape=(X_train_tfidf_stage2.shape[1,]),kernel_regularizer=l2(0.01)), # Input Layer
    # BatchNormalization(),
    Dense(units=15, activation='relu',kernel_regularizer=l2(0.01)), # Hidden Layer 1
    Dropout(rate=0.2), # Dropout Layer 1
    # Dense(units=10, activation='relu',kernel_regularizer=l2(0.01)), # Hidden Layer 2

    Dense(units=6, activation='softmax') # Output Layer
]
```

```
# Define the updated layers
```

```

# layers = [
#     # Convolutional Layer
#     Conv1D(filters=32, kernel_size=3, activation='relu', input_shape=(X_train_tfidf_stage2.shape[1], 1)),
#     MaxPooling1D(pool_size=2), # Pooling Layer to reduce dimensionality
#     BatchNormalization(),
#     Conv1D(filters=8, kernel_size=3, activation='relu'),
#     MaxPooling1D(pool_size=2), # Pooling Layer to reduce dimensionality
#     BatchNormalization(),
#     Flatten(), # Flatten the Conv1D output for the Dense layers

#     # Dense Layers
#     Dense(units=120, activation='relu', kernel_regularizer=l2(0.01)),
#     # BatchNormalization(),
#     Dropout(rate=0.3), # Dropout Layer 1
#     Dense(units=64, activation='relu', kernel_regularizer=l2(0.01)),
#     Dropout(rate=0.2), # Dropout Layer 2
#     Dense(units=16, activation='relu', kernel_regularizer=l2(0.01)),
#     Dense(units=6, activation='softmax') # Output Layer
# ]

# Initialize the Sequential model
model = Sequential(layers)

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

```

▼ Model Train

Train the model on training data with validation on the test data

```

history = model.fit(X_train_tfidf_stage2.toarray(),
                    y_train_tfidf_stage2,
                    epochs=27,
                    batch_size=256,
                    validation_split=0.3
                    )

```

```

Epoch 1/27
80/80 ————— 3s 14ms/step - accuracy: 0.3612 - loss: 2.7485 - val_accuracy: 0.4631 - val_loss: 2.1544
Epoch 2/27
80/80 ————— 1s 12ms/step - accuracy: 0.6506 - loss: 1.5311 - val_accuracy: 0.4817 - val_loss: 1.7221
Epoch 3/27
80/80 ————— 1s 10ms/step - accuracy: 0.6989 - loss: 1.1762 - val_accuracy: 0.5195 - val_loss: 1.5127
Epoch 4/27
80/80 ————— 1s 12ms/step - accuracy: 0.7171 - loss: 1.0354 - val_accuracy: 0.5708 - val_loss: 1.3677
Epoch 5/27
80/80 ————— 1s 10ms/step - accuracy: 0.7303 - loss: 0.9636 - val_accuracy: 0.6141 - val_loss: 1.2417
Epoch 6/27
80/80 ————— 1s 10ms/step - accuracy: 0.7330 - loss: 0.9181 - val_accuracy: 0.6570 - val_loss: 1.1309
Epoch 7/27
80/80 ————— 1s 10ms/step - accuracy: 0.7377 - loss: 0.8936 - val_accuracy: 0.6766 - val_loss: 1.0478
Epoch 8/27
80/80 ————— 1s 17ms/step - accuracy: 0.7434 - loss: 0.8775 - val_accuracy: 0.6861 - val_loss: 1.0078
Epoch 9/27
80/80 ————— 3s 17ms/step - accuracy: 0.7392 - loss: 0.8732 - val_accuracy: 0.6906 - val_loss: 0.9879
Epoch 10/27
80/80 ————— 2s 10ms/step - accuracy: 0.7520 - loss: 0.8450 - val_accuracy: 0.6915 - val_loss: 0.9878
Epoch 11/27
80/80 ————— 1s 10ms/step - accuracy: 0.7414 - loss: 0.8554 - val_accuracy: 0.6912 - val_loss: 0.9901
Epoch 12/27
80/80 ————— 1s 10ms/step - accuracy: 0.7561 - loss: 0.8375 - val_accuracy: 0.6891 - val_loss: 0.9983
Epoch 13/27
80/80 ————— 1s 10ms/step - accuracy: 0.7558 - loss: 0.8448 - val_accuracy: 0.6928 - val_loss: 1.0038
Epoch 14/27
80/80 ————— 1s 10ms/step - accuracy: 0.7676 - loss: 0.8287 - val_accuracy: 0.6937 - val_loss: 1.0123
Epoch 15/27
80/80 ————— 1s 10ms/step - accuracy: 0.7738 - loss: 0.8168 - val_accuracy: 0.6848 - val_loss: 1.0155
Epoch 16/27
80/80 ————— 1s 10ms/step - accuracy: 0.7732 - loss: 0.8131 - val_accuracy: 0.6936 - val_loss: 1.0122
Epoch 17/27
80/80 ————— 1s 10ms/step - accuracy: 0.7777 - loss: 0.8099 - val_accuracy: 0.6809 - val_loss: 1.0281
Epoch 18/27

```

```

80/80 ————— 1s 10ms/step - accuracy: 0.7893 - loss: 0.8079 - val_accuracy: 0.6884 - val_loss: 1.0368
Epoch 19/27
80/80 ————— 1s 13ms/step - accuracy: 0.7937 - loss: 0.7932 - val_accuracy: 0.6876 - val_loss: 1.0349
Epoch 20/27
80/80 ————— 1s 14ms/step - accuracy: 0.7889 - loss: 0.7961 - val_accuracy: 0.6921 - val_loss: 1.0463
Epoch 21/27
80/80 ————— 1s 17ms/step - accuracy: 0.8002 - loss: 0.7872 - val_accuracy: 0.6915 - val_loss: 1.0544
Epoch 22/27
80/80 ————— 1s 17ms/step - accuracy: 0.7996 - loss: 0.7833 - val_accuracy: 0.6854 - val_loss: 1.0554
Epoch 23/27
80/80 ————— 1s 11ms/step - accuracy: 0.7926 - loss: 0.7964 - val_accuracy: 0.6850 - val_loss: 1.0672
Epoch 24/27
80/80 ————— 1s 10ms/step - accuracy: 0.8020 - loss: 0.7847 - val_accuracy: 0.6797 - val_loss: 1.0769
Epoch 25/27
80/80 ————— 1s 10ms/step - accuracy: 0.8094 - loss: 0.7766 - val_accuracy: 0.6806 - val_loss: 1.0793
Epoch 26/27
80/80 ————— 1s 10ms/step - accuracy: 0.8165 - loss: 0.7659 - val_accuracy: 0.6801 - val_loss: 1.0964
Epoch 27/27
80/80 ————— 1s 10ms/step - accuracy: 0.8168 - loss: 0.7696 - val_accuracy: 0.6857 - val_loss: 1.0968

```

```
nn_classifier = model
```

Classification Report

```

# Make predictions on the test set
y_pred_prob = model.predict(X_test_tfidf_stage2.toarray())
y_pred = y_pred_prob.argmax(axis=1) # Convert probabilities to class predictions

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

# Compute the confusion matrix
classification_report_final =classification_report(y_test_multi, y_pred)
print(f"Overall Accuracy: {accuracy_score_final:.2f}")

print(classification_report_final)
conf_matrix_nn = confusion_matrix(y_test_multi,y_pred)

# Print classification report
print("Confusion Matrix:\n",conf_matrix_nn)

```

226/226 ————— 0s 2ms/step

```

Accuracy: 0.6875086457324665
Overall Accuracy: 0.69

```

	precision	recall	f1-score	support
0	0.80	0.77	0.78	755
1	0.80	0.70	0.75	527
2	0.66	0.76	0.71	3016
3	0.73	0.45	0.55	237
4	0.67	0.50	0.58	536
5	0.67	0.62	0.64	2158
accuracy			0.69	7229
macro avg	0.72	0.63	0.67	7229
weighted avg	0.69	0.69	0.69	7229

```

Confusion Matrix:
[[ 582  13  90   8  40  22]
 [ 15 367 111   1   7  26]
 [ 57  38 2297  21  40 563]
 [  7   5   79 106  14  26]
 [ 49  14  157   4 270  42]
 [ 18  21  734   6  31 1348]]

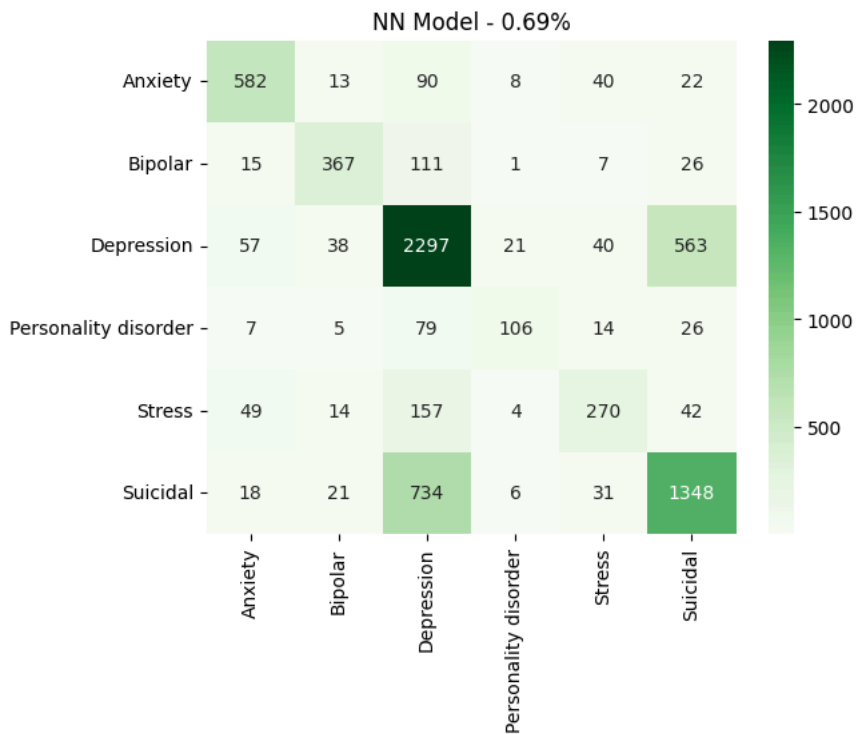
```

Heat Map

```

ax = sns.heatmap(conf_matrix_nn, annot = True, fmt='d', cmap='Greens', xticklabels=multi_encoder.classes_, yticklabels=multi_encoder.classes_)
ax.set_title(f'NN Model - {accuracy_score_final:.2}%')
plt.show()

```

Model Loss and Accuracy

```
plt.figure(figsize=(12, 5))

# Plot training & validation loss values
plt.subplot(1, 2, 1) # 1 row, 2 columns, 1st subplot
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(loc='upper right')

# Plot training & validation accuracy values
plt.subplot(1, 2, 2) # 1 row, 2 columns, 2nd subplot
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')

plt.tight_layout() # Adjusts subplots to fit in the figure area.
plt.show() # Display the plots
```