week3_code

February 9, 2025

1 Week 2 Ingesting and Exploring the Dataset

[ ]:
```
# install wordcloud
!pip install wordcloud
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: wordcloud in /home/jupyter-
geean/.local/lib/python3.12/site-packages (1.9.4)
Requirement already satisfied: numpy>=1.6.1 in
/opt/tljh/user/lib/python3.12/site-packages (from wordcloud) (1.26.4)
Requirement already satisfied: pillow in /opt/tljh/user/lib/python3.12/site-
packages (from wordcloud) (11.1.0)
Requirement already satisfied: matplotlib in /opt/tljh/user/lib/python3.12/site-
packages (from wordcloud) (3.10.0)
Requirement already satisfied: contourpy>=1.0.1 in
/opt/tljh/user/lib/python3.12/site-packages (from matplotlib->wordcloud) (1.3.1)
Requirement already satisfied: cycler>=0.10 in
/opt/tljh/user/lib/python3.12/site-packages (from matplotlib->wordcloud)
(0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/opt/tljh/user/lib/python3.12/site-packages (from matplotlib->wordcloud)
(4.55.3)
Requirement already satisfied: kiwisolver>=1.3.1 in
/opt/tljh/user/lib/python3.12/site-packages (from matplotlib->wordcloud) (1.4.8)
Requirement already satisfied: packaging>=20.0 in
/opt/tljh/user/lib/python3.12/site-packages (from matplotlib->wordcloud) (24.1)
Requirement already satisfied: pyparsing>=2.3.1 in
/opt/tljh/user/lib/python3.12/site-packages (from matplotlib->wordcloud) (3.2.1)
Requirement already satisfied: python-dateutil>=2.7 in
/opt/tljh/user/lib/python3.12/site-packages (from matplotlib->wordcloud)
(2.9.0.post0)
Requirement already satisfied: six>=1.5 in /opt/tljh/user/lib/python3.12/site-
packages (from python-dateutil>=2.7->matplotlib->wordcloud) (1.17.0)
```

[ ]:
```
# import packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
from wordcloud import WordCloud
%matplotlib inline

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

[ ]:
```
# change working directory
import os
os.getcwd()  # Get current working directory
os.chdir('..')  # Move up one directory level from notebooks
print(os.getcwd())
#os.chdir('../data') # change to the data folder
```

```
/home/jupyter-geean/cookiecutter-data-science/{{ cookiecutter.repo_name }}
```

[ ]:
```
# load the data
df = pd.read_csv('data/Combined Data.csv', index_col=0)
```

[ ]:
```
# make a copy and get rid of the missing values
df1 = df.copy()
df1.dropna(inplace = True)
# see the top head of the data
df1.head()
```

[ ]:

	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
3	I've shifted my focus to something else but I'…	Anxiety
4	I'm restless and restless, it's been a month n…	Anxiety

[ ]:
```
# number of missing values
missing_values = df.isnull().sum()

print(missing_values)
```

```
statement    362
status         0
dtype: int64
```

[ ]:
```
# get the rows and columns of all of the data
rows,columns = df.shape
print(f"Number of rows: {rows}")
print(f"Number of columns: {columns}")
```

```
Number of rows: 53043
Number of columns: 2
```

[ ]:
```
# calculate the number of missing values
rows_with_missing = df[df.isnull().any(axis=1)]
print(rows_with_missing)
```

```
      statement   status
293         NaN  Anxiety
572         NaN  Anxiety
595         NaN  Anxiety
1539        NaN   Normal
2448        NaN   Normal
…           …       …
52838       NaN  Anxiety
52870       NaN  Anxiety
52936       NaN  Anxiety
53010       NaN  Anxiety
53031       NaN  Anxiety

[362 rows x 2 columns]
```

The dataset contains 362 missing values in the 'Statement' column and no missing values for 'Status'.

2 Missing Values -Week 3

Many of the rows have NaNs and represent anxiety and normal. Since there are 53,043 values and there are only 362 rows where there is missing values. We feel that it is best to drop these rows since they represent only 0.7% of the data and as you will see later we have an abundance of "normal" and "anxiety" labeled data.

The dataset includes 52,681 rows and 2 columns after removing missing values.

[ ]:
```
# get the rows and columns of the data that drops the missing values
rows,columns = df1.shape
print(f"Number of rows: {rows}")
print(f"Number of columns: {columns}")
```

```
Number of rows: 52681
Number of columns: 2
```

We want to add a column to explore the length of each statement. This can help us quantify the user's input and support further analysis. This will give us an idea on how to preprocess the text and determine tokenization especially for transformer models. Many NLP models, especially those based on deep learning, have limitations on input length so determining the length is important.

[ ]:
```
# create a new column that gives the length of each statement
df1['statement_len'] = df1['statement'].apply(lambda x: len(x.split(' ')))
df1.head()
```

[ ]:

	statement	status	statement_len
0	oh my gosh	Anxiety	3
1	trouble sleeping, confused mind, restless hear…	Anxiety	10
2	All wrong, back off dear, forward doubt. Stay …	Anxiety	14
3	I've shifted my focus to something else but I'…	Anxiety	11
4	I'm restless and restless, it's been a month n…	Anxiety	14

From the output, we can see that this dataset includes 2 variables: statement and status.

The statement variable is a text variable that contains different user inputs.

The status variable represents different emotional statuses, which contain different categories.

The next step is to explore dataset

[ ]:
```
# information about the dataset
'''The class type of the DataFrame.
The range of the index.
The number of columns and their names.
The count of non-null values in each column.
The data type of each column.
The memory usage of the DataFrame.'''

print(df1.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 52681 entries, 0 to 53042
Data columns (total 3 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   statement      52681 non-null  object
 1   status         52681 non-null  object
 2   statement_len  52681 non-null  int64
dtypes: int64(1), object(2)
memory usage: 1.6+ MB
None
```

Statement and status column are object data types. The statement_len column is an integer/numeric datatype.

[ ]:
```
# descriptive statistics
'''count is the number of non-null entries.
unique is the number of unique values.
top is the most frequent value.
freq is the frequency of the most frequent value.'''
df1.describe(include='object').T
```

[ ]:

	count	unique	top	freq
statement	52681	51073	what do you mean?	22
status	52681	7	Normal	16343

The 'Statement' column contains 51,073 unique values, indicating that most user inputs are unique. The most frequently appeared statement is "What do you mean?" and occurred 22 times in the dataset. The frequent occurrence of "What do you mean?" suggests significant communication gaps or misunderstandings, indicating areas where individuals feel confused or need more clarity, which is crucial in mental health discussions. This phrase often reflects a state of uncertainty or anxiety, signaling important emotional states. It could also indicate active engagement and a desire for better understanding and it could indicate that individuals need more support or reassurance, aiding in tailoring mental health resources effectively.

The 'Status' column contains 7 unique values and represents different emotion statuses. The most common status is "Normal", suggesting that over 30% of the statements in the dataset fall under this category.

[ ]:
```
# Get summary statistics for the 'statement_len' column
summary_statistics = df1['statement_len'].describe()
print(summary_statistics)
```

```
count    52681.000000
mean       113.035914
std        163.501877
min          1.000000
25%         15.000000
50%         62.000000
75%        148.000000
max       6300.000000
Name: statement_len, dtype: float64
```

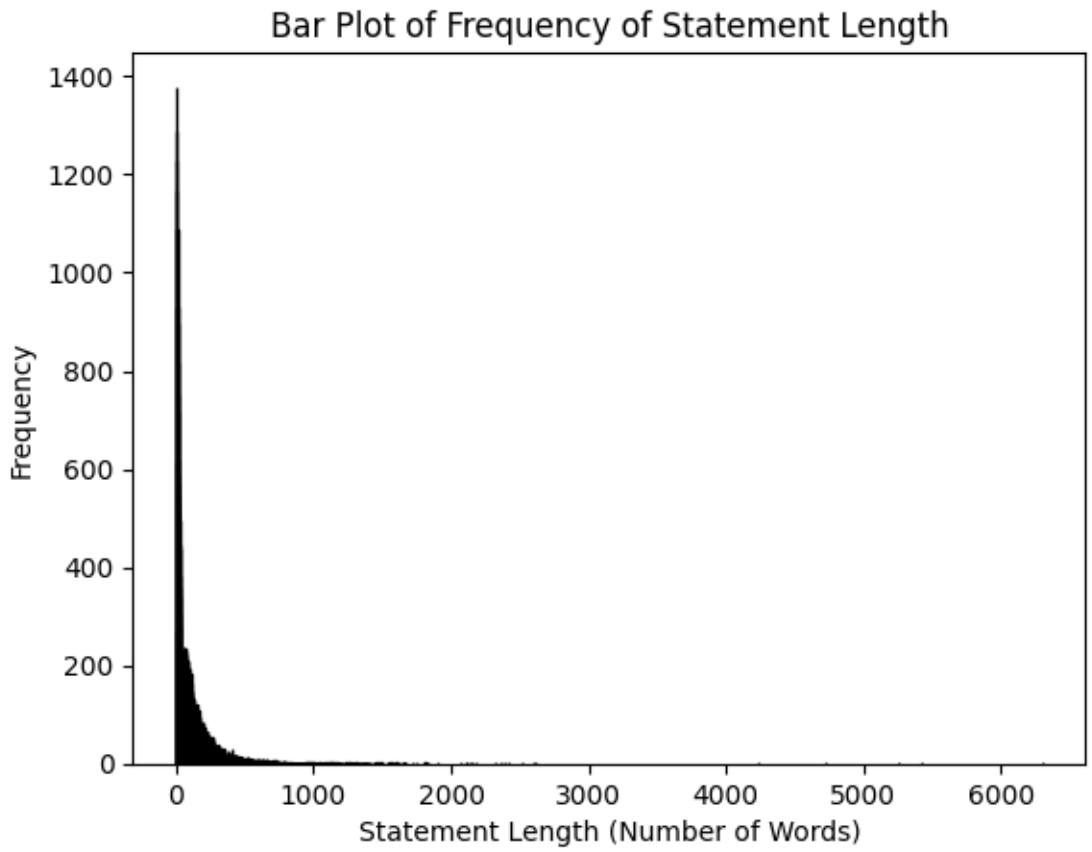
[ ]:
```
# Calculate the mode of the 'statement_len' column
mode_value = df1['statement_len'].mode()[0]

print(f"The mode of the 'statement_len' column is: {mode_value}")
```

```
The mode of the 'statement_len' column is: 5
```

The summary statistics for the 'Statement_len' column show the distribution of statment lengths. The average statement contains 113 words with a standard deviation of 163.5 words. The shortest statement only has 1 word, while the longest contains 6300 words. The most frequent statement length is 5 words, indicating that short phrases are commonly used.

The following bar plot of the frequency of statement length visualizes the previous statement.

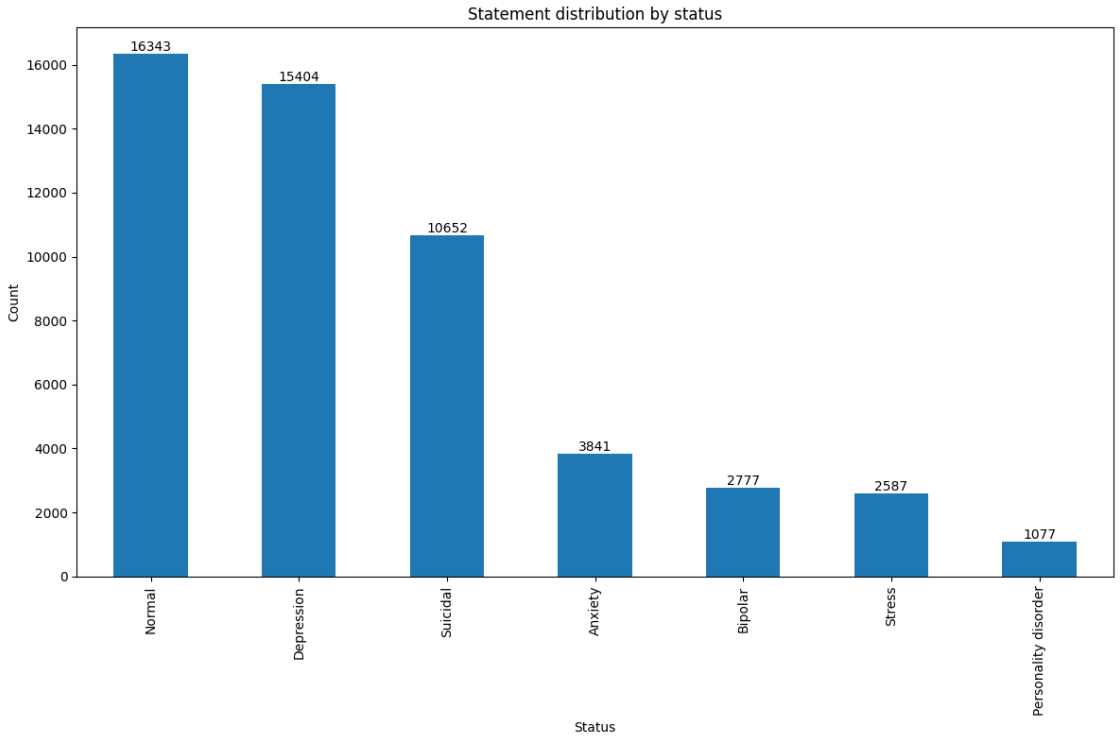
[ ]:
```
# Create a bar plot of the frequency of the 'statement_len' column
statement_len_counts = df1['statement_len'].value_counts()

plt.bar(statement_len_counts.index, statement_len_counts.values,␣
  ↪edgecolor='black')
plt.xlabel('Statement Length (Number of Words)')
plt.ylabel('Frequency')
plt.title('Bar Plot of Frequency of Statement Length')
plt.show()
```



The histogram shows that it is a right skewed distribution, which most of the statement length under 1000 words. This means that when we focus on the output length, we should set it to be under 1000.

[ ]:
```
# Histogram of Frequency of Statements by Status
plt.figure(figsize=(12,8))

# get the unique status values and their counts
status_counts = df1['status'].value_counts()

# create the bar plot
ax = status_counts.plot(kind='bar')

# add the count labels on top of each bar
for i, v in enumerate(status_counts):
    ax.text(i, v, str(v), ha='center', va='bottom')

plt.title('Statement distribution by status')
plt.xlabel('Status')
plt.ylabel('Count')
```

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



Here is a plot showing distribution by status. Normal is the most common status and contains 16343 data, followed by depression and suicidal, which are the 2nd and 3rd largest portions of the dataset. Personality disorder is the most rare one, which contains 1077 data.

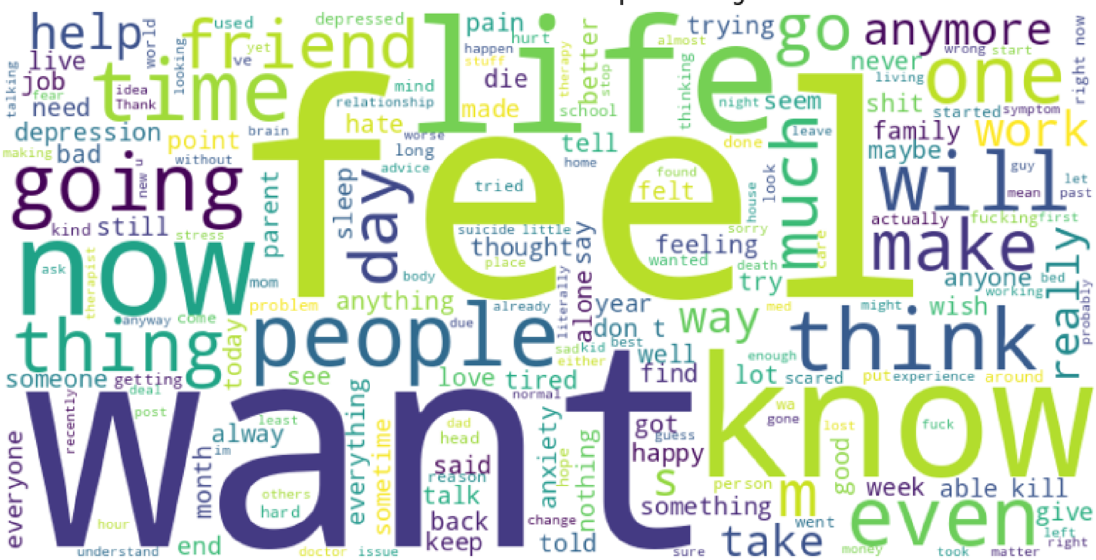
The ratio between different statuses suggests about 70% of the user's input falls under the negative status category.

The target variable in our dataset is unbalanced in favor of depression, suicidal and normal. This imbalance could affect our model's performance, so we'll need to address it later to ensure accurate and fair predictions especially when predicting sentiment analysis for anxiety, bipolar, stress and personality disorder.

[ ]:
```
# Word Cloud Before Preprocessing
# Combine all statements into a single string
text = ' '.join(df1['statement'].dropna())

# Create a word cloud
wordcloud = WordCloud(width=800, height=400, background_color='white').
 ↪generate(text)

# Display the word cloud
```

```python
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('Word Cloud Before Preprocessing')
plt.show()
```

Word Cloud Before Preprocessing



Here is the word cloud for Statement before data processing, which will be used to compare with the data after processing. The word cloud shows that the most frequently used words are "feel," "want," "know," and "life.". The observation is reasonable considering verbs and similar expressions that reflect personal thoughts would be the biggest part of user inputs. We can also see words like "depression," "tired," and "anxiety" in the word cloud even before data processing, which matches our observation of the status distribution above.

This means that these words may be key indicators for determining sentiment. We will explore more by dividing word clouds into status.

2.0.1 Week 3 Code- EDA and Train-Test Split

We have already done some EDA in the previous code but we will expand on it here. We will also do a train-test split.

[ ]:
```python
# Taking a look at the long messages
df1[df1['statement_len'] > 1500]
```

[ ]:
```
                                               statement  \
7851   So introduction I guess.. my name is Michael a…
8221   do not really know where to start so I am goin…
9504   Hello everyone,I rarely post on Reddit but…
```

```
10743  ThrowawayIm female, 20 years old. Ever since I…
10834  I have only 1 person I can somewhat open to bu…
11537  The title is not meant to discourage others, b…
11581  I no longer know what else to do but write thi…
11636  And has life gotten better?&amp;#x200B;No. Eve…
11831  Sorry this is long but I doubt anyone will eve…
13188  I am frustrated. that is the constant theme wi…
13293  I cannot TAKE IT ANYMORE. I cannot TAKE IT ANY…
13577  I am very sick and tired, both mentally and ph…
14602  I am 27 years old and have grown deeper into a…
16061  Bear with me please, this may be extremely len…
16498  Hey, this is goodbye note. it is most likely g…
18215  I am someone living in Turkey. My age is proba…
18323  I am going to be turning 30 in a couple weeks…
19321  This happened a little while ago but it still …
19701  If there is a more beneficial sub please lmk s…
20867  Apologies for length. there is a *lot* to expl…
21285  First I am going to present you with a few que…
21396  will i ever be noticed? is my life worth anyth…
21858  I constantly repeat to myself that I have neve…
22243  I do not expect anyone to read this rambly mes…
22351  This is a lengthy post but its a summary of my…
22563  I have been thinking about posting online for …
23195  My entire life has spontaneously combusted ove…
23366  I wish I knew what was wrong with me. So many …
23820  I need support or encouragement. I (29M) reall…
23845  This is a a vent. I (29M) really do not know w…
24276  I guess it all started when I was I guess 11, …
38083  this is my first reddit post also my first tim…
38255  i m at a very weird place in my life right now…
38579  hello thank you for reading my post and any ad…
39579  we ve been seeing a worrying increase in pro s…
39582  for starter i never really had a childhood whe…
39752  it doesn t matter anymore i m going to copy an…
40028  this is a long story i m sorry me and my ex br…
40208  i m at a very weird place in my life right now…
40293  i have come to the conclusion that i am just n…
40371  hello thank you for reading my post and any ad…
46660  DEPRESSION HAS A PURPOSE: HOW TO USE IT RIGHT …
47949  Don't know what to do anymore Back when I was …
48915  I think I'm in the middle of a nervous breakdo…
50253  Manic for 6 months ending up in jail where I h…
51396  Please help me understand what I went through …
52775  I don't know what to do. I don't know how to d…

                  status  statement_len
7851          Depression           2153
```

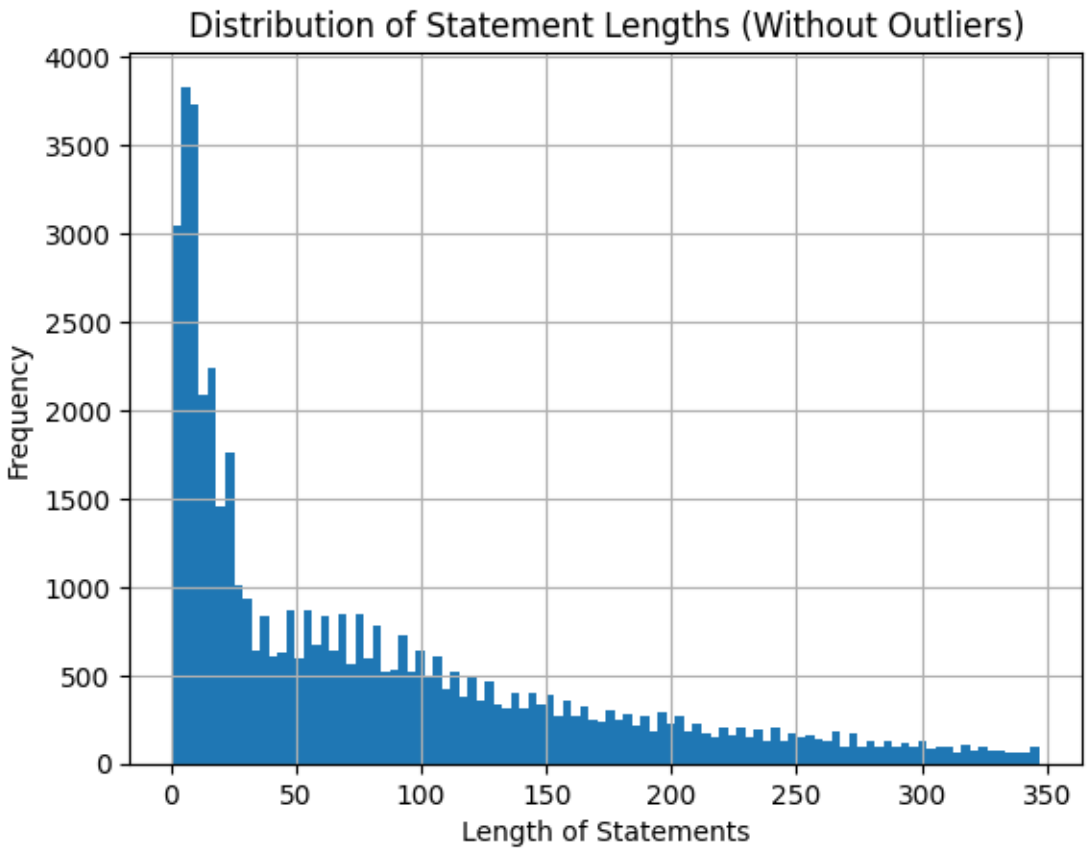
8221	Depression	1602
9504	Depression	2139
10743	Depression	1537
10834	Suicidal	5248
11537	Depression	2391
11581	Depression	2612
11636	Depression	2415
11831	Depression	2187
13188	Depression	1832
13293	Suicidal	6300
13577	Suicidal	1811
14602	Depression	1809
16061	Depression	1558
16498	Suicidal	1566
18215	Suicidal	2066
18323	Suicidal	1559
19321	Depression	1902
19701	Depression	1661
20867	Depression	1625
21285	Depression	1559
21396	Depression	2510
21858	Depression	2599
22243	Suicidal	2364
22351	Depression	1551
22563	Suicidal	2319
23195	Depression	1818
23366	Depression	1654
23820	Depression	2105
23845	Suicidal	2108
24276	Suicidal	1539
38083	Depression	1559
38255	Depression	1584
38579	Depression	1537
39579	Depression	1747
39582	Depression	1653
39752	Depression	4239
40028	Depression	1726
40208	Depression	1584
40293	Depression	1656
40371	Depression	1537
46660	Bipolar	4727
47949	Depression	1663
48915	Stress	1601
50253	Bipolar	1664
51396	Personality disorder	5419
52775	Anxiety	1586

Many of the longest messages are those with depression and suicidal tendencies. This will help us

since if we shorten the output length when preprocessing the data, we are not reducing the number of data points for those that do not have very many data points such as anxiety, bipolar, stress and personality disorder.

Now we want to see a clearer distribution without these outliers so that we can determine the best output length for preprocessing the text.

[ ]:
```
# Statement Length Distribution Without Outliers
# Calculate Q1 (25th percentile) and Q3 (75th percentile)
Q1 = df1['statement_len'].quantile(0.25)
Q3 = df1['statement_len'].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 = df1[(df1['statement_len'] >= lower_bound) & (df1['statement_len']␣
 ↪<= upper_bound)]
# Plot the distribution of statement lengths without outliers
filtered_df['statement_len'].hist(bins=100)
plt.title('Distribution of Statement Lengths (Without Outliers)')
plt.xlabel('Length of Statements')
plt.ylabel('Frequency')
plt.show()
```



This distrubtion still shows a right-skewed data distribution. We now have a much clearer distribution where approximately 50% of the statements have 0-50 word lengths, especially with a spike at approximately 25 words with approximately 3700 statements. This will help us immensely to determine the best statement length to run our transformer models to save computational resources and time but not decrease model performance.

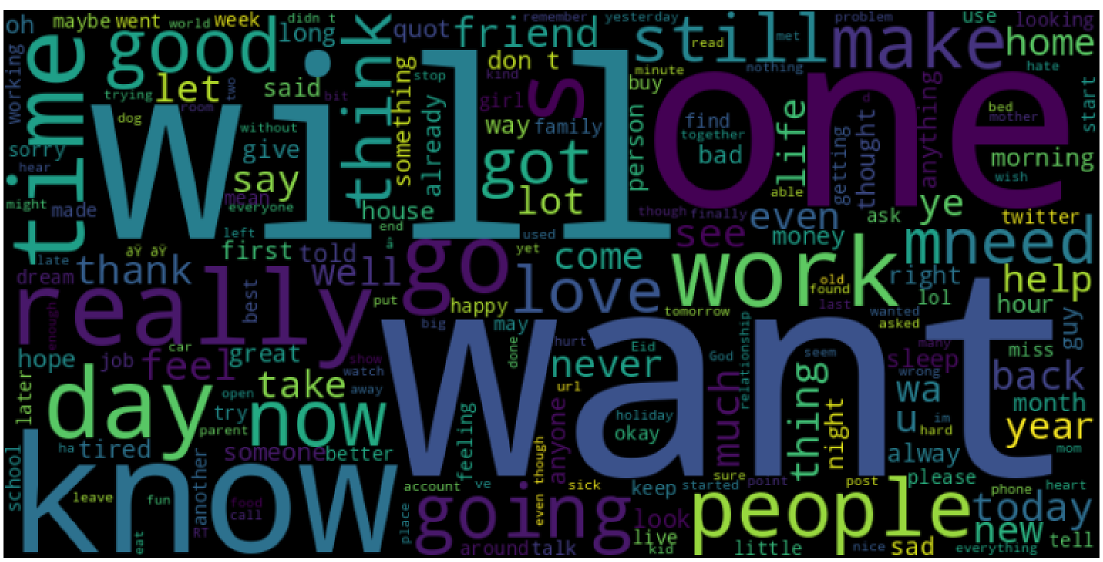
Now we want to take a closer look at the word clouds for each status since it will give us even more information about the possible word indicators for each status.

[ ]:
```
# Create a function to generate and display a word cloud
def generate_word_cloud(text, title):
    wordcloud = WordCloud(width=800, height=400).generate(text)
    plt.figure(figsize=(10, 5))
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.title(title)
    plt.axis('off')
    plt.show()

# Generate word clouds for each status
statuses = df1['status'].unique()
```

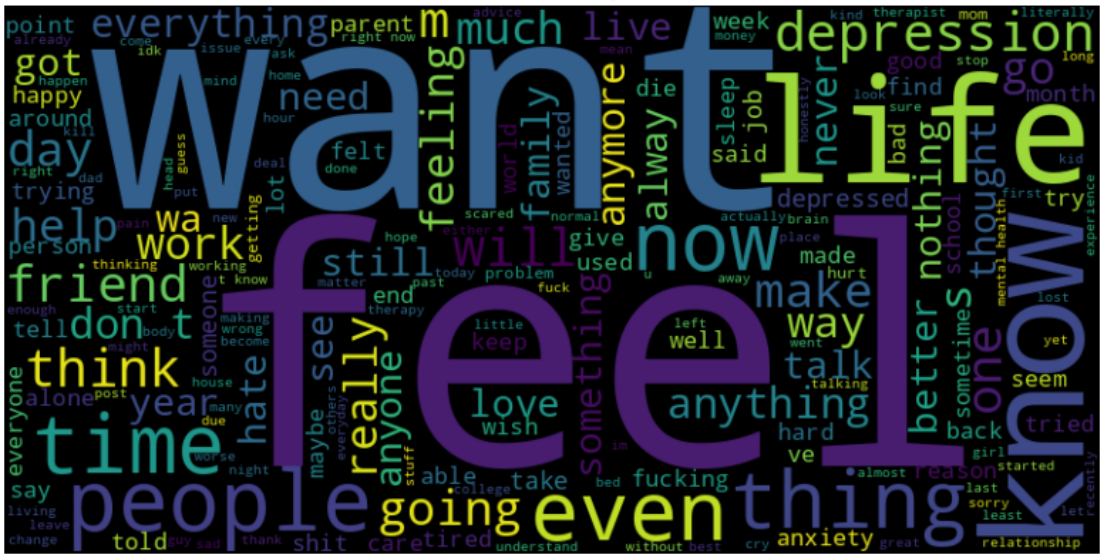
```
for status in statuses:
    status_text = ' '.join(df1[df1['status'] == status]['statement'])
    generate_word_cloud(status_text, title=f'Word Cloud for {status}')
```

Word Cloud for Anxiety

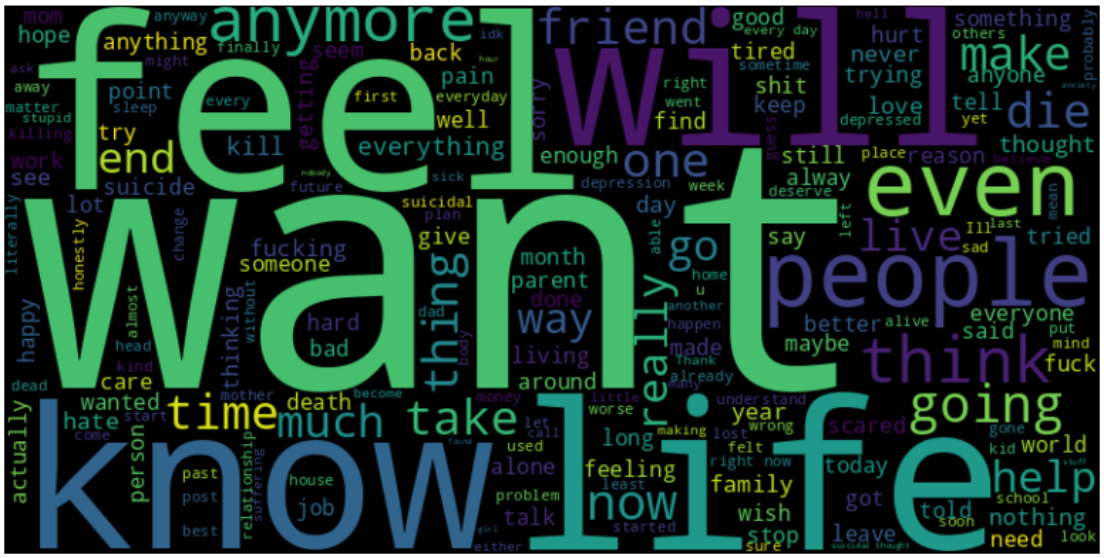
Word Cloud for Normal



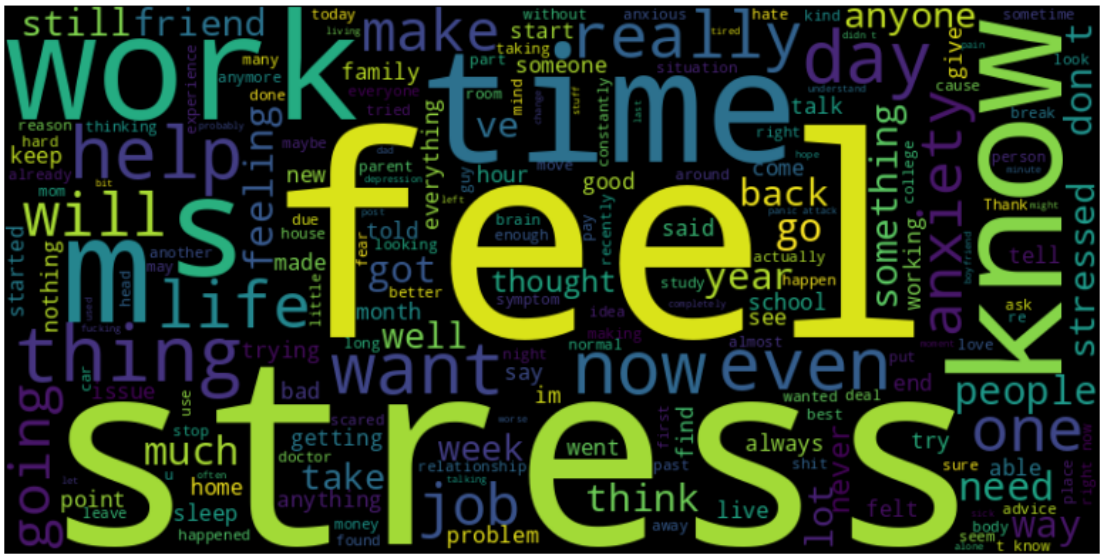
Word Cloud for Depression



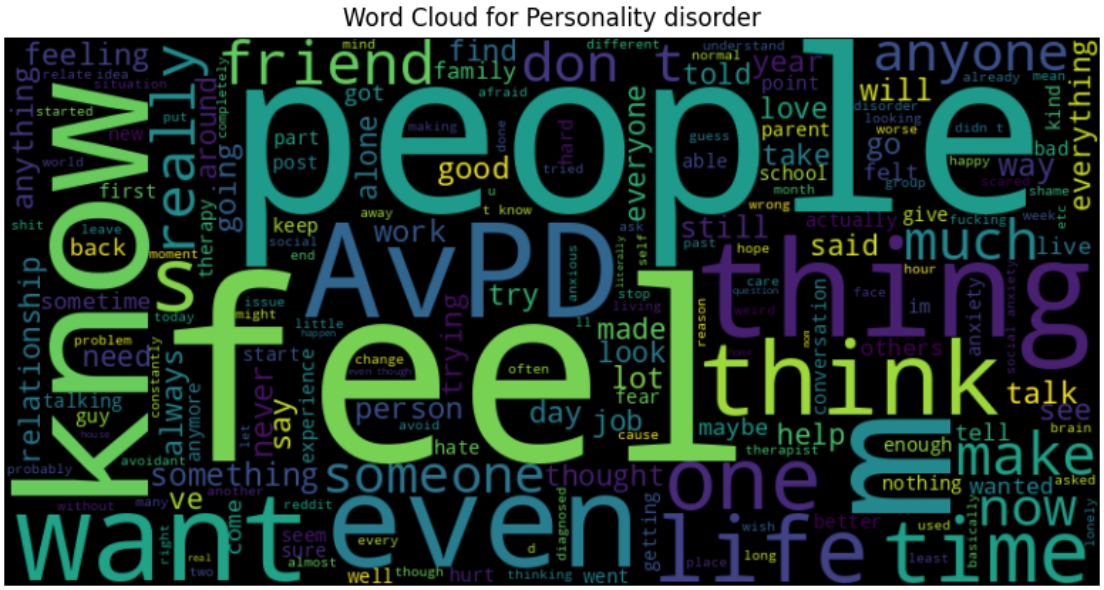
Word Cloud for Suicidal



Word Cloud for Stress



Word Cloud for Bipolar



We can see for anxiety, stress, and bipolar disorder that “anxiety” and “stress”, “bipolar” are the most popular words. “AVPD”, “people”, and “feel” are the most common words for personality disorder. These words tend to make sense since the words are the literal statuses. Avoidant Personality Disorder (AVPD) is a mental health condition marked by chronic feelings of inadequacy, extreme sensitivity to criticism, and a strong desire to avoid social interactions due to fear of rejection and is a medical condition for a personality disorder.

“Will”, “want”, “know” are the most common words for normal. “Life”, “feel”, “want” are the most common words for suicidal. Many people with suicidal tendencies tend to talk about their lives. The words for “normal” status tend to have positive connotations especially “want” and “will”, especially for a “will” to live.

These words seem to be preliminary indicators to help determine the status based on their statement. We can compare once we calculate sentiment scores.

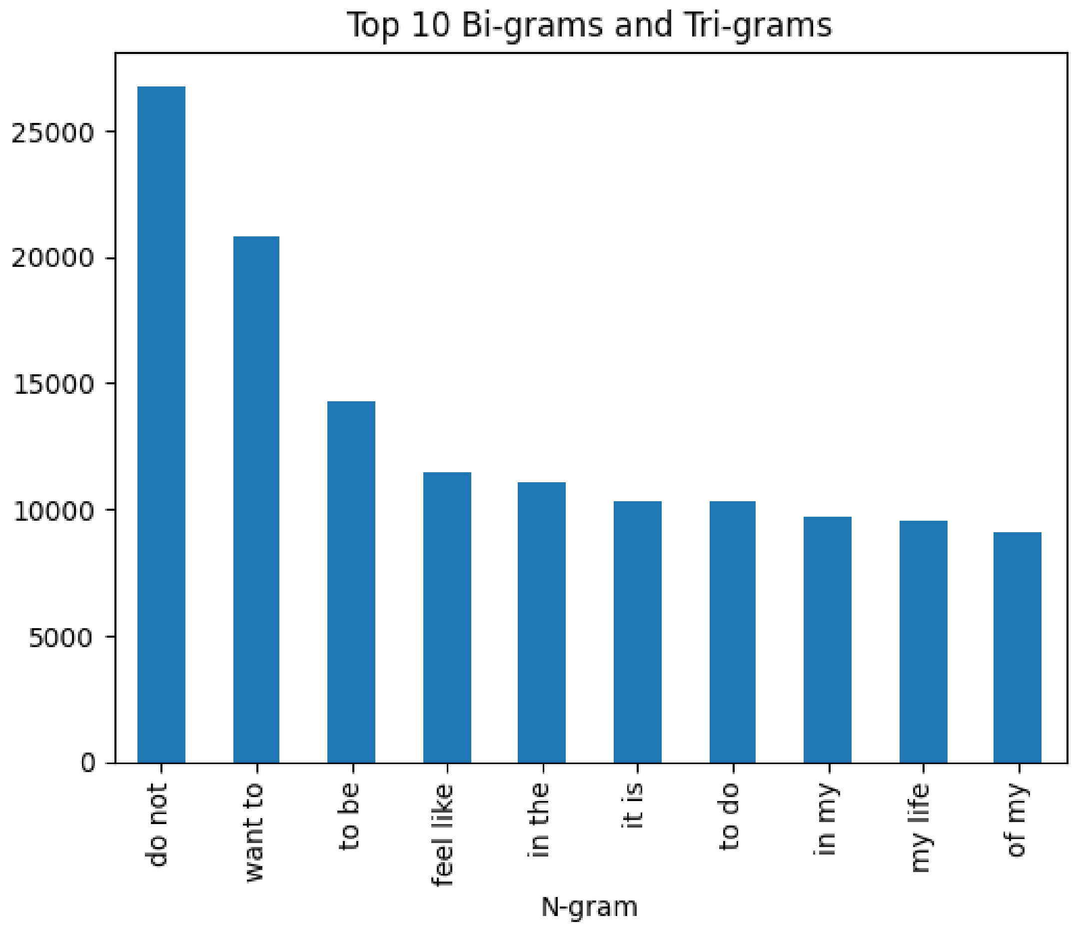
We want to conduct bi-grams and tri-grams analysis for these reasons: Contextual Insights: Bi-grams and tri-grams capture phrases and context that single words (unigrams) might miss. This is particularly important in mental health, where phrases like “feeling down” or “very anxious” provide more insight than individual words.

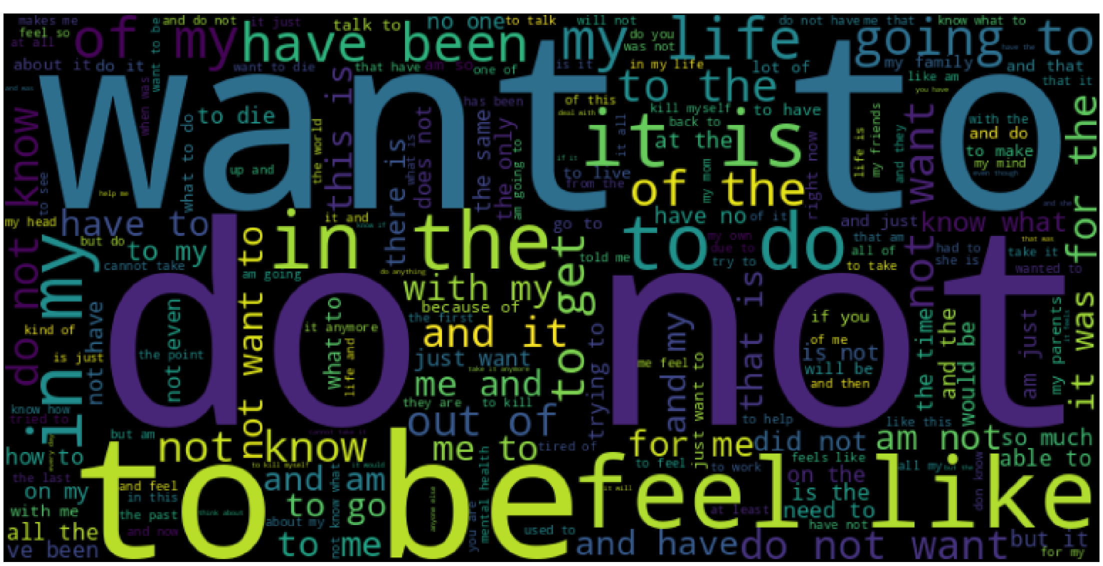
Improved Sentiment Detection: Phrases often convey sentiment more accurately than single words. For example, “not happy” is a negative sentiment that might be missed if only individual words are considered.

Identifying Common Themes: Visualizing bi-grams and tri-grams helps identify common themes and expressions in the dataset. This can reveal patterns in how people express their mental health experiences.

[ ]:
```
# import tokenizer
from sklearn.feature_extraction.text import CountVectorizer
```

```
# Tokenization and N-gram generation
# Create a CountVectorizer object with ngram_range set to (2, 3) to generate␣
 ↪bi-grams and tri-grams
vectorizer = CountVectorizer(ngram_range=(2, 3))

# Fit and transform the 'statement' column of the DataFrame to generate the␣
 ↪n-grams
X = vectorizer.fit_transform(df1['statement'])

# Frequency distribution
# Sum the occurrences of each n-gram across all documents
sum_words = X.sum(axis=0)

# Create a list of tuples where each tuple contains an n-gram and its␣
 ↪corresponding frequency
words_freq = [(word, sum_words[0, idx]) for word, idx in vectorizer.vocabulary_.
 ↪items()]

# Sort the list of tuples by frequency in descending order
words_freq = sorted(words_freq, key=lambda x: x[1], reverse=True)

# DataFrame for visualization
# Convert the list of tuples into a DataFrame for easier visualization
df_freq = pd.DataFrame(words_freq, columns=['N-gram', 'Frequency'])

# Bar plot
# Plot the top 10 most frequent n-grams as a bar plot
df_freq.head(10).plot(kind='bar', x='N-gram', y='Frequency', legend=False)
plt.title('Top 10 Bi-grams and Tri-grams')
plt.show()

# Word cloud
# Generate a word cloud from the n-gram frequencies
wordcloud = WordCloud(width=800, height=400).
 ↪generate_from_frequencies(dict(words_freq))

# Display the word cloud
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```


Top 10 Bi-grams and Tri-grams
0
5000
10000
15000
20000
25000
do not
want to
to be
feel like
in the
it is
to do
in my
my life
of my
N-gram


want to
do not
to be
feel like
in the
to do
it is
in my
have been
my life
going to
of my
to the
of the
not know
with my
and it
to get
out of
me and
just want
for me
did not
am not
so much
able to
on the
is the
need to
do not want
and have
to me
to go
and am
have to
to my
know what
have no
it was
for the
the time
and the
would be
is not
if you
that is
and my
do not know
this is
there is
does not
the same
the only
trying to
not want to
me to
how to
on my
all the
ve been
want to die
to die
but it
to live
at the

This visualizes the most common bi-grams and tri-grams in our dataset, providing insights into common phrases and patterns, which is particularly useful for sentiment analysis in mental health. It helps identify key expressions and themes that might indicate different emotional states or communication gaps. The top ten are do not, want to, to be, feel like, in the, it is, to do, in my, my life, and of my.

Negative Sentiments: Phrases like "do not" and "feel like" might indicate negative sentiments or expressions of reluctance and emotional states. These bi-grams can help identify statements where individuals are expressing dissatisfaction or discomfort.

Desires and Intentions: Bi-grams such as "want to" and "to do" suggest expressions of desires, intentions, or plans. Analyzing these can reveal what individuals are striving for or what actions they are considering, which can be linked to their mental state.

Self-Reflection: Phrases like "in my," "my life," and "of my" indicate self-reflection and personal experiences. These bi-grams can help identify statements where individuals are discussing their personal lives and feelings, which are critical for understanding their mental health.

General Context: Bi-grams like "to be," "in the," and "it is" provide general context and can be part of various expressions. While they might not directly indicate sentiment, they help in understanding the structure and flow of the text.

2.0.2 Train-Validation-Test Split

Most common splits are 80-20 so we will use this split here. We will also create a validation set that is 10% and the test set is 10%. The final splits will be 80-10-10.

I did these splits, since this is a common split in machine learning and data science but also because with 80% of the data as a training dataset, a large portion ensures that the model has enough data to learn from, which helps in capturing the underlying patterns and relationships in the data. With a Validation Set of 10%, it can tune hyperparameters and make decisions about the model architecture and helps prevent overfitting by providing a checkpoint to evaluate the model's performance on unseen data during the training process. With a 10% Test Set, we can evaluate the model's performance after it has been trained and validated and 10% is a large enough size given that we have approximately 50,000 data points. The 80-10-10 split is a balanced approach that ensures the model has sufficient data for training while also providing enough data for validation and testing to ensure robust performance

[ ]:
```
# Train Validation Test Split

from sklearn.model_selection import train_test_split

# Split the data into training and temporary sets (80% train, 20% temporary)
train_df, temp_df = train_test_split(df1, test_size=0.2, random_state=42)

# Split the temporary set into validation and test sets (50% validation, 50%␣
 ↪test of the temporary set)
val_df, test_df = train_test_split(temp_df, test_size=0.5, random_state=42)

# Print the sizes of the splits
```

```
print(f"Training set size: {len(train_df)}")
print(f"Validation set size: {len(val_df)}")
print(f"Test set size: {len(test_df)}")
```

```
Training set size: 42144
Validation set size: 5268
Test set size: 5269
```