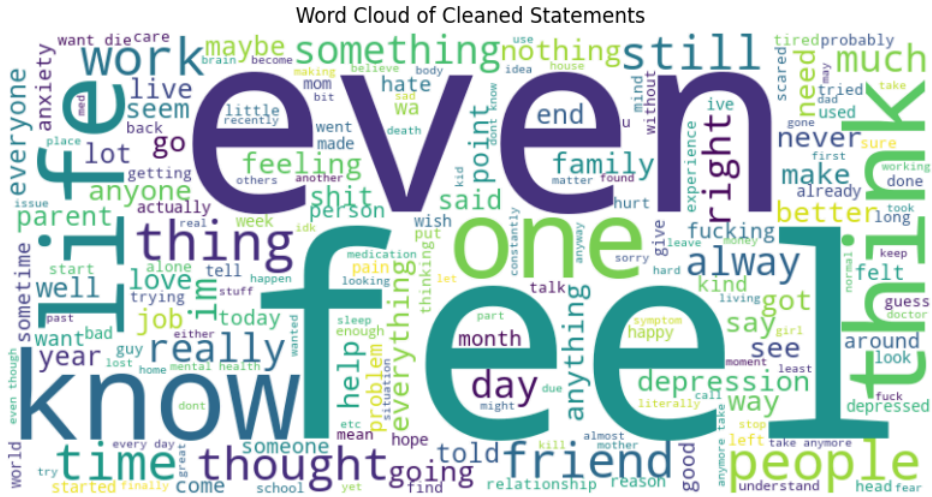
Sentiment Analysis Using ML

In [ ]:
```
from IPython.display import Image

# Display an image from a file
Image(filename='output.png')
```

Out[ ]:



In [ ]:
```
#Import Neccessary libraries

import pandas as pd
import re
import plotly.express as px
import string
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import BernoulliNB, MultinomialNB
from xgboost import XGBClassifier
from sklearn.preprocessing import LabelEncoder
import plotly.figure_factory as ff
import plotly.graph_objects as go
from wordcloud import WordCloud
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
import numpy as np
from textblob import TextBlob
import matplotlib.pyplot as plt
%matplotlib inline
#inline magic for matplotlib
```

In [ ]:
```
#path for the dataset
path = r"C:\Users\aayus\Sentiment Analysis\Combined Data.csv"
```

In [ ]:
```
df = pd.read_csv(path)
```

In [ ]:
```
#Printing first 5 rows
df.head()
```

Out[ ]:

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

In [ ]:
```
#Printing some information about the dataset
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 53043 entries, 0 to 53042
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   Unnamed: 0  53043 non-null  int64 
 1   statement   52681 non-null  object
 2   status      53043 non-null  object
dtypes: int64(1), object(2)
memory usage: 1.2+ MB
None
```

Check for null Values

In [ ]:
```
print(df.isnull().sum())
```

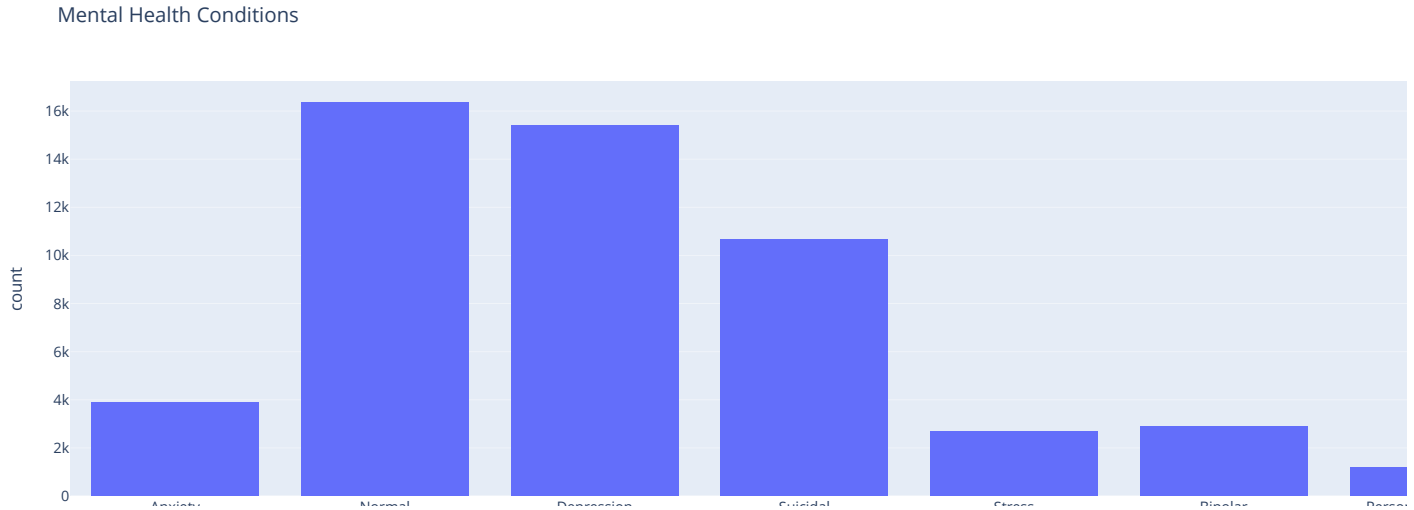
```
Unnamed: 0      0
statement     362
status          0
dtype: int64
```

In [ ]:
```
print(df.columns)
```

```
Index(['Unnamed: 0', 'statement', 'status'], dtype='object')
```

Various Mental Health Conditions

In [ ]:
```
fig = px.histogram(df, x = 'status', title = 'Mental Health Conditions')
fig.show()
```



In [ ]:
```
#Must fill the 'NA' or Null values
df['statement'] = df['statement'].fillna('')
```

In [ ]:
```
#For getting the length of the Statement

def length(text):
    return len(str(text).split())
```

In [ ]:
```
#appending Length of various Statements in text_length column

df['text_length'] = df['statement'].apply(length)
```

In [ ]:
```
df.head()
```

Out[ ]:

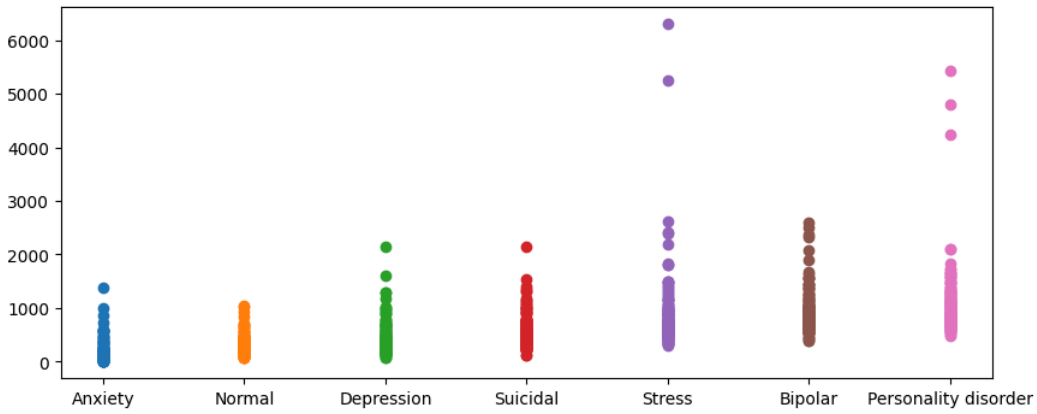
	Unnamed: 0	statement	status	text_length
0	0	oh my gosh	Anxiety	3
1	1	trouble sleeping, confused mind, restless hear...	Anxiety	10
2	2	All wrong, back off dear, forward doubt. Stay ...	Anxiety	14
3	3	I've shifted my focus to something else but I'...	Anxiety	11
4	4	I'm restless and restless, it's been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

Count for various text lengths

In [ ]:
```
fig = px.histogram(df, x='text_length', title = 'Statement Size')
fig.show()
```

In [ ]:
```
x = df['status'].unique()
y = df['text_length'].unique()
labels = df['status'].unique()

y_groups = np.array_split(y, 7)
```

```
plt.figure(figsize = (10, 4))

for i, y_group in enumerate(y_groups):
    plt.scatter([x[i]]*len(y_group), y_group)
```



NLTK Stopwords

In [ ]:
```
nltk.download('stopwords')
nltk.download('punkt')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\aayus\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\aayus\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Out[ ]: True

In [ ]:
```
# For basic Preprocessing
def preprocessing_pipline(text):

    #removing unneccesary data
    text = text.lower()  # Lowercase text
    text = re.sub(r'\[.*?\]', '', text)  # Remove text in square brackets
    text = re.sub(r'https?://\S+|www\.\S+', '', text)  # Remove Links
    text = re.sub(r'<.*?>+', '', text)  # Remove HTML tags
    text = re.sub(r'[%s]' % re.escape(string.punctuation), '', text)  # Remove punctuation
    text = re.sub(r'\n', '', text)  # Remove newlines
    text = re.sub(r'\w*\d\w*', '', text)  # Remove words containing numbers
    return text
# To Remove Stopwords
def stopwordsRemoval(text):
    stop_words = set(stopwords.words('english'))
    tokens = word_tokenize(text)
    tokens = [word for word in tokens if word not in stop_words]
    return ' '.join(tokens)
#for Translation
def generateText(text):
    try:
        blob = TextBlob(text)
        text = blob.translate(to ='fr').translate(to = 'en')
    except Exception as e:
        text = text
```

In [ ]:
```
new_df = df.drop(['statement'], axis = 1)
```

In [ ]:
```
#new column
new_df['augmented_statement'] = df['statement'].apply(lambda x: preprocessing_pipline(x))
```

In [ ]:
```
new_df.head()
```

Out[ ]:

	Unnamed: 0	status	text_length	augmented_statement
0	0	Anxiety	3	oh my gosh
1	1	Anxiety	10	trouble sleeping confused mind restless heart ...
2	2	Anxiety	14	all wrong back off dear forward doubt stay in ...
3	3	Anxiety	11	ive shifted my focus to something else but im ...
4	4	Anxiety	14	im restless and restless its been a month now ...

In [ ]:
```
new_df.drop(['Unnamed: 0'], axis = 1, inplace = True)
```

In [ ]:
```
new_df.head()
```

Out[ ]:

	status	text_length	augmented_statement
0	Anxiety	3	oh my gosh
1	Anxiety	10	trouble sleeping confused mind restless heart ...
2	Anxiety	14	all wrong back off dear forward doubt stay in ...
3	Anxiety	11	ive shifted my focus to something else but im ...
4	Anxiety	14	im restless and restless its been a month now ...

Types of Various Mental Conditions

In [ ]:
```
new_df['status'].unique()
```

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

In [ ]:
```
np.shape([new_df['text_length']])
```

Out[ ]:
```
(1, 53043)
```

Again Check for Null Values

In [ ]:
```
new_df.isnull().sum()
```

Out[ ]:
```
status                0
text_length           0
augmented_statement    0
dtype: int64
```

Again Preprocess the text

In [ ]:
```
#Preprocessing Again because the text was translated

new_df['cleaned_statement'] = new_df['augmented_statement'].apply(generateText)
new_df['cleaned_statement'] = new_df['augmented_statement'].apply(lambda x: preprocessing_pipline(x))
new_df['cleaned_statement'] = new_df['augmented_statement'].apply(lambda x : stopwordsRemoval(x))
```

In [ ]:
```
new_df.tail()
```

Out[ ]:

	status	text_length	augmented_statement	cleaned_statement
53038	Anxiety	322	nobody takes me seriously i've dealt with dep...	nobody takes seriously ' dealt depressionanxie...
53039	Anxiety	198	selfishness i dont feel very good its like i ...	selfishness dont feel good like dont belong wo...
53040	Anxiety	17	is there any way to sleep better i cant sleep ...	way sleep better cant sleep nights meds didnt ...
53041	Anxiety	74	public speaking tips hi all i have to give a p...	public speaking tips hi give presentation work...
53042	Anxiety	79	i have really bad door anxiety its not about b...	really bad door anxiety scared didnt lock door...

XG Boost Classifier

In [ ]:
```
XGB = XGBClassifier()
```

Label Encoder

In [ ]:
```
lb = LabelEncoder()
y = lb.fit_transform(new_df['status'])
```

In [ ]:
```
#Encoded Classes

lb.classes_
```

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

In [ ]:
```
X = new_df.cleaned_statement
```

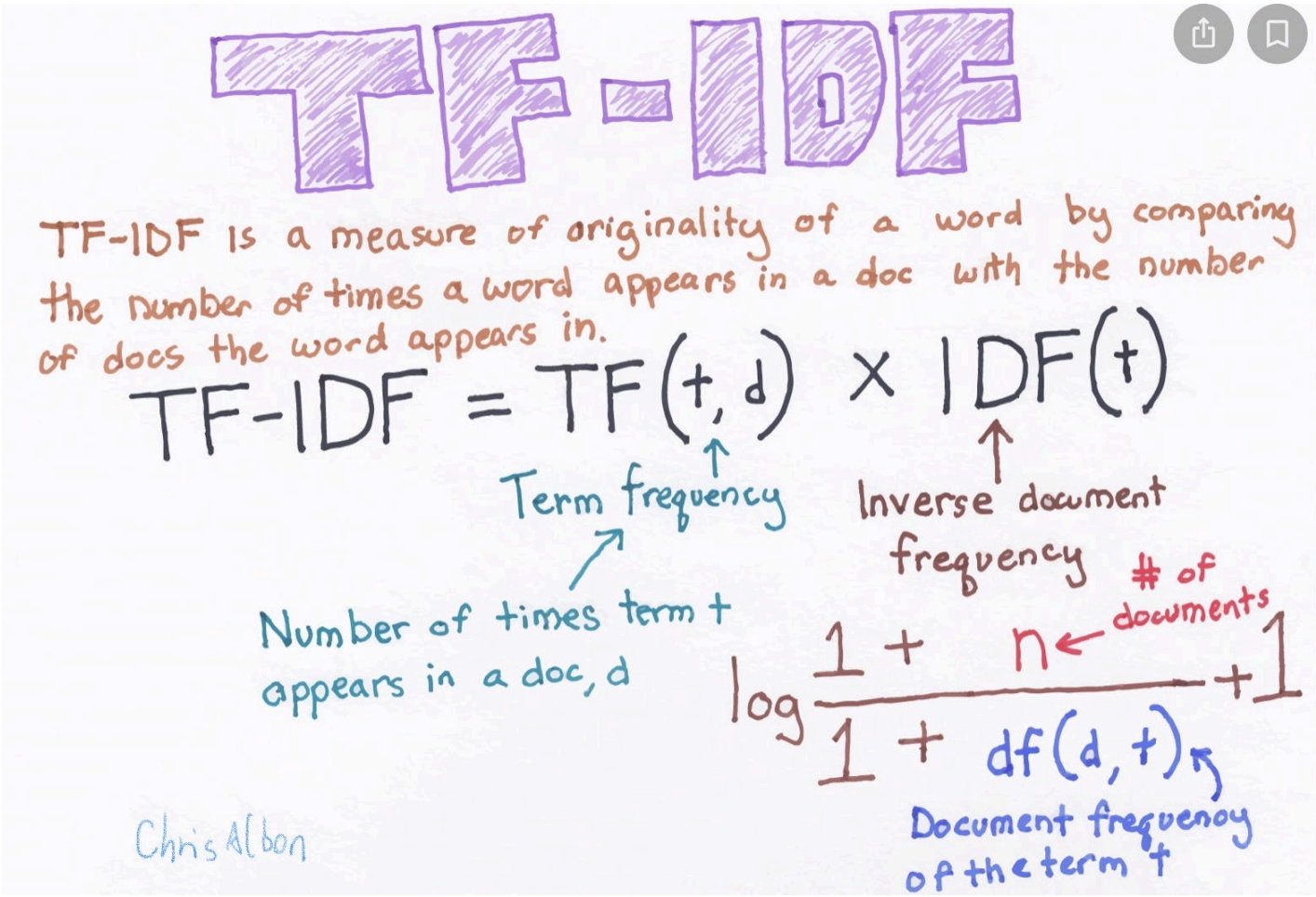
Train Test Split

In [ ]:
```
#Using 80% of the Data to train and 20% for the Testing

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 23)
```

In [ ]:
```
print(X_train)
```

```
3625                                    make invisible
46602    faking sorry rant dont know people never seem ...
38536    recently move back parent become incredibly de...
13103    fall asleep hope nightmares bother want sleep ...
13863    everything feels wrong like feel dumb unwanted...
                               ...
9704     ill depressed rest life know everything shit g...
11190    know take time writing trying seek help gain n...
26569    society reinforced everyone money looks nobody...
9256     lot problems mental illness lot trauma work wo...
41555                          leaving parking lot work
Name: cleaned_statement, Length: 42434, dtype: object
```

Vectorizing the Dataset using TFIDF Vectorizer



In [ ]:
```
vectorizer = TfidfVectorizer(max_features = 10000)
X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)
```

In [ ]:
```
#Shape of X_Train idf and X_Test idf

print(X_train_tfidf.shape)
print(X_test_tfidf.shape)
```

```
(42434, 10000)
(10609, 10000)
```

XG Boost Training

In [ ]:
```
XGB.fit(X_train_tfidf,y_train)
```

Out[ ]:
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
```

In [ ]:
```
xgb_predict = XGB.predict(X_test_tfidf)
```

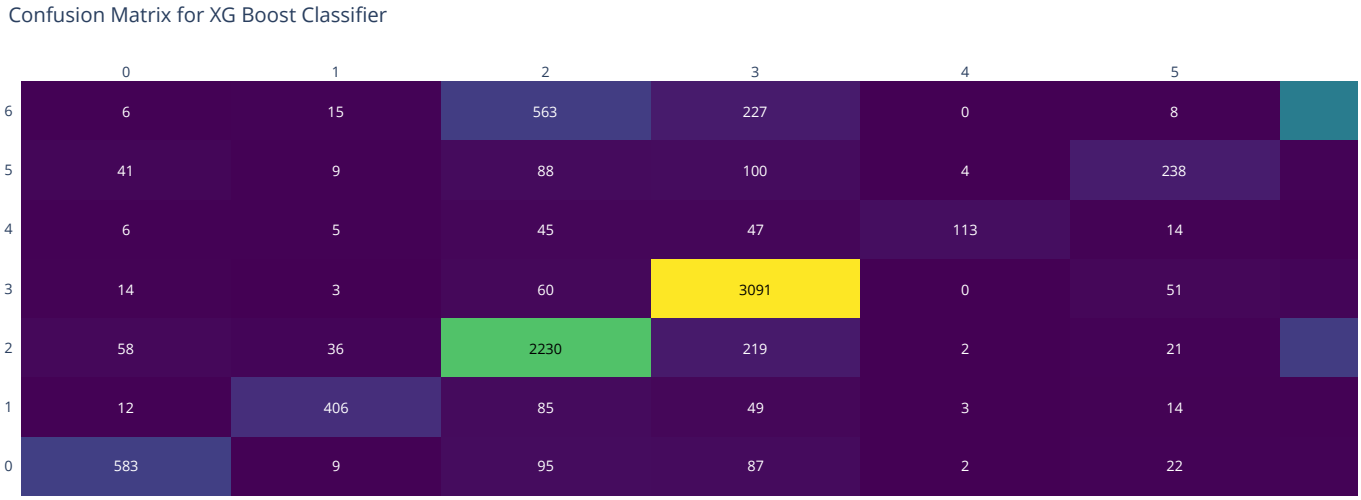
In [ ]:
```
print('Accuracy of Logistic Regression')
print(accuracy_score(y_test,xgb_predict))
```

```
Accuracy of Logistic Regression
0.7497407861249882
```

In [ ]:
```
print('Classification Report of Logistic Regression')
print(classification_report(y_test,xgb_predict))
```

```
Classification Report of Logistic Regression
              precision    recall  f1-score   support

           0       0.81      0.72      0.76       815
           1       0.84      0.70      0.76       581
           2       0.70      0.72      0.71      3115
           3       0.81      0.95      0.87      3253
           4       0.91      0.49      0.63       232
           5       0.65      0.48      0.55       501
           6       0.67      0.61      0.64      2112

    accuracy                           0.75     10609
   macro avg       0.77      0.66      0.70     10609
weighted avg       0.75      0.75      0.74     10609
```

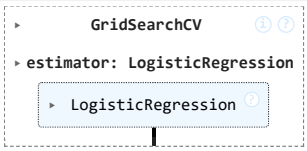
In [ ]:
```
cm = confusion_matrix(y_test, xgb_predict)
cm_fig = ff.create_annotated_heatmap(
    z=cm,
    x=list(set(y_test)),
    y=list(set(y_test)),
    annotation_text=cm,
    colorscale='Viridis'
)
cm_fig.update_layout(title='Confusion Matrix for XG Boost Classifier')
cm_fig.show()
```



Logistic Regression Training

In [ ]:
```
param_grid = {
    'C': [0.01, 0.1, 1, 10, 100]
}

model = LogisticRegression(max_iter=1000)
grid_search = GridSearchCV(model, param_grid, cv=5, scoring='accuracy')

grid_search.fit(X_train_tfidf, y_train)
```

Out[ ]:



In [ ]:
```
best_model = grid_search.best_estimator_
LogisticPred = best_model.predict(X_test_tfidf)
```

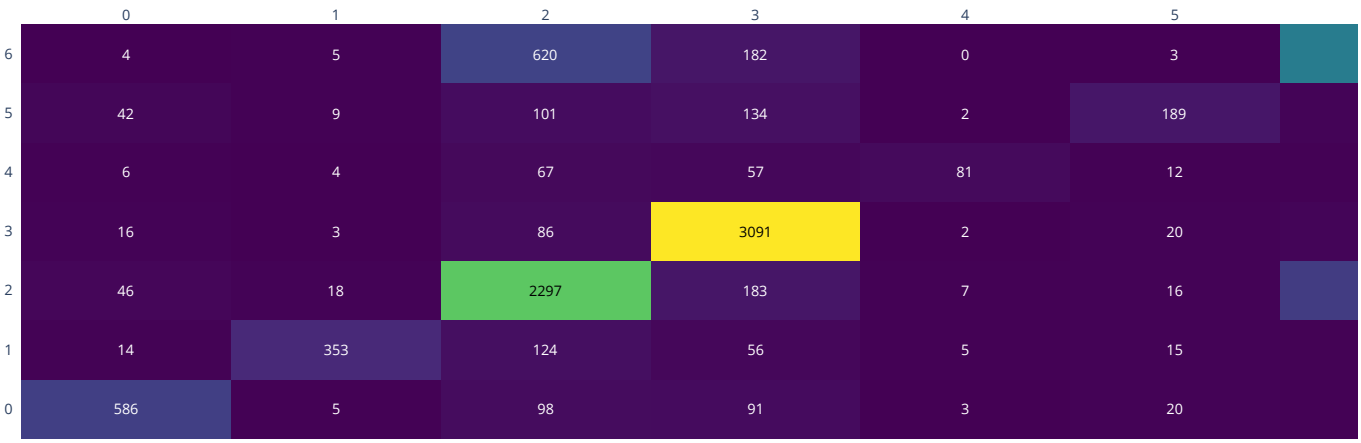
In [ ]:
```
print('Accuracy of Logistic Regression')
print(accuracy_score(y_test, LogisticPred))
```

```
Accuracy of Logistic Regression
0.7441794702610991
```

In [ ]:
```
print('Classification Report of Logistic Regression')
print(classification_report(y_test, LogisticPred))
```

```
Classification Report of Logistic Regression
              precision    recall  f1-score   support

           0       0.82      0.72      0.77       815
           1       0.89      0.61      0.72       581
           2       0.68      0.74      0.71      3115
           3       0.81      0.95      0.88      3253
           4       0.81      0.35      0.49       232
           5       0.69      0.38      0.49       501
           6       0.67      0.61      0.64      2112

    accuracy                           0.74     10609
   macro avg       0.77      0.62      0.67     10609
weighted avg       0.74      0.74      0.74     10609
```

In [ ]:
```
cm = confusion_matrix(y_test, LogisticPred)
cm_fig = ff.create_annotated_heatmap(
    z=cm,
    x=list(set(y_test)),
    y=list(set(y_test)),
    annotation_text=cm,
    colorscale='Viridis'
)
cm_fig.update_layout(title='Confusion Matrix For Logistic Regression')
cm_fig.show()
```



In [ ]:
```
MNB = MultinomialNB()
MNB.fit(X_train_tfidf, y_train)
```

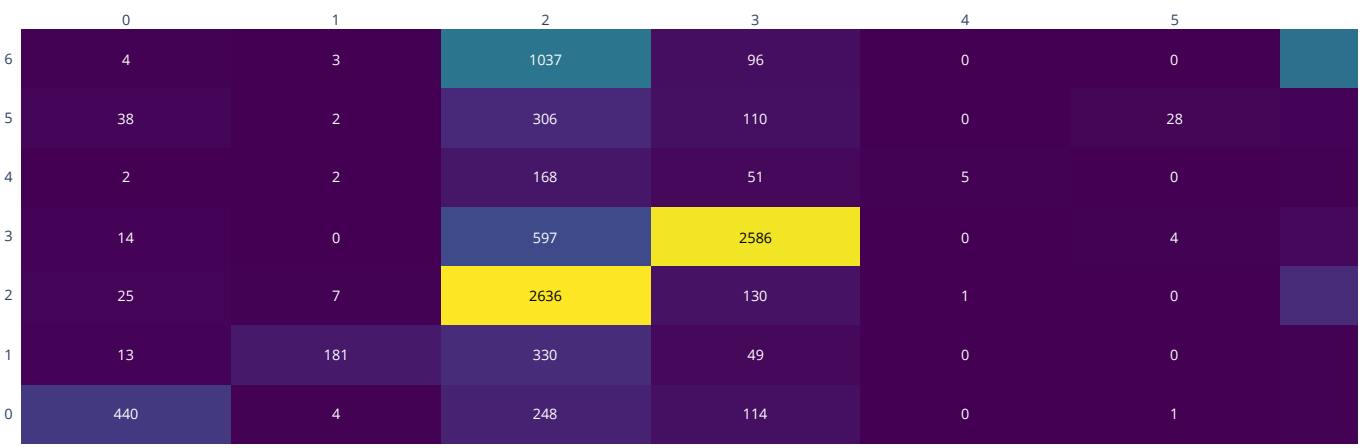
Out[ ]:
```
MultinomialNB()
```

In [ ]:
```
MNBPredict = MNB.predict(X_test_tfidf)
```

In [ ]:
```
print('Accuracy of Multinomial Naiive Bayes')
print(accuracy_score(y_test, MNBPredict))
print()
print('Classification Report of Multinomial Naiive Bayes')
print(classification_report(y_test, MNBPredict))
```

```
Accuracy of Multinomial Naiive Bayes
0.645489678574795

Classification Report of Multinomial Naiive Bayes
              precision    recall  f1-score   support

           0       0.82      0.54      0.65       815
           1       0.91      0.31      0.46       581
           2       0.50      0.85      0.62      3115
           3       0.82      0.79      0.81      3253
           4       0.83      0.02      0.04       232
           5       0.85      0.06      0.10       501
           6       0.71      0.46      0.56      2112

    accuracy                           0.65     10609
   macro avg       0.78      0.43      0.46     10609
weighted avg       0.71      0.65      0.62     10609
```

In [ ]:
```
cm = confusion_matrix(y_test, MNBPredict)
cm_fig = ff.create_annotated_heatmap(
    z=cm,
    x=list(set(y_test)),
    y=list(set(y_test)),
    annotation_text=cm,
    colorscale='Viridis'
)
cm_fig.update_layout(title='Confusion Matrix For Multinomial Naiive Bayes')
cm_fig.show()
```



In [ ]:
```
BNB = BernoulliNB()
BNB.fit(X_train_tfidf, y_train)
```

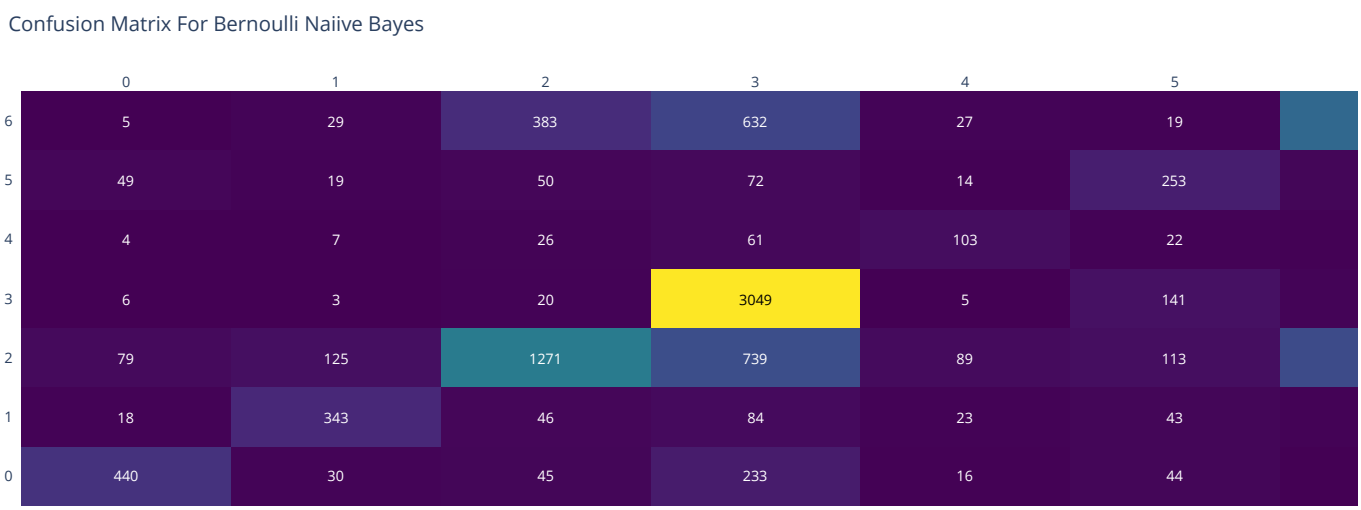
Out[ ]:
```
BernoulliNB()
```

In [ ]:
```
BNBPredict = BNB.predict(X_test_tfidf)
```

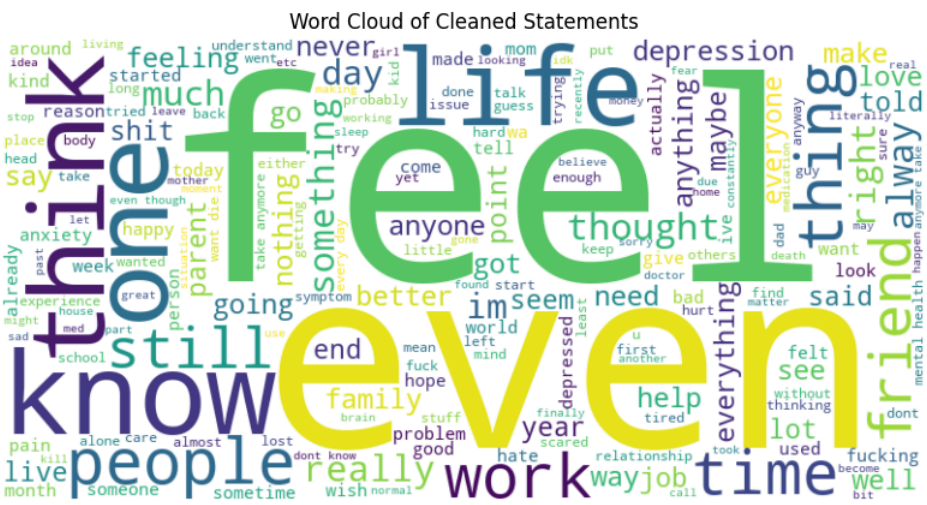
In [ ]:
```
print('Accuracy of Bernoulli Naiive Bayes')
print(accuracy_score(y_test, BNBPredict))
print()
print('Classification Report of Bernoulli Naiive Bayes')
print(classification_report(y_test, BNBPredict))
```

```
Accuracy of Bernoulli Naiive Bayes
0.6104251107550194

Classification Report of Bernoulli Naiive Bayes
              precision    recall  f1-score   support

           0       0.73      0.54      0.62       815
           1       0.62      0.59      0.60       581
           2       0.69      0.41      0.51      3115
           3       0.63      0.94      0.75      3253
           4       0.37      0.44      0.40       232
           5       0.40      0.50      0.45       501
           6       0.56      0.48      0.52      2112

    accuracy                           0.61     10609
   macro avg       0.57      0.56      0.55     10609
weighted avg       0.62      0.61      0.59     10609
```

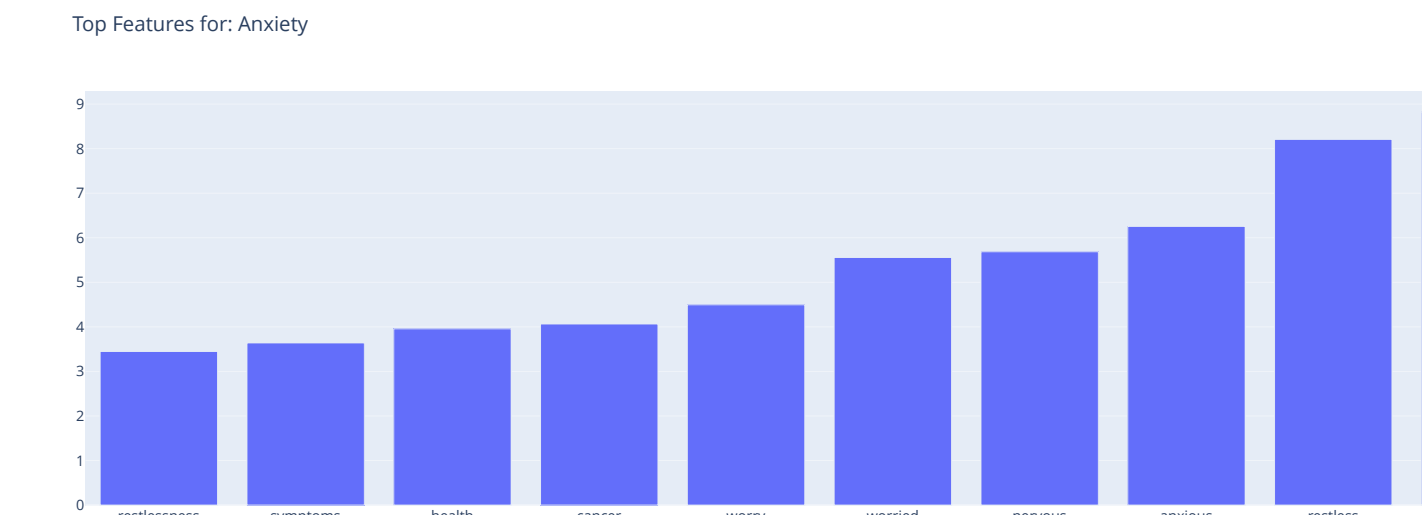
In [ ]:
```
cm = confusion_matrix(y_test, BNBPredict)
cm_fig = ff.create_annotated_heatmap(
    z=cm,
    x=list(set(y_test)),
    y=list(set(y_test)),
    annotation_text=cm,
    colorscale='Viridis'
)
cm_fig.update_layout(title='Confusion Matrix For Bernoulli Naiive Bayes')
cm_fig.show()
```

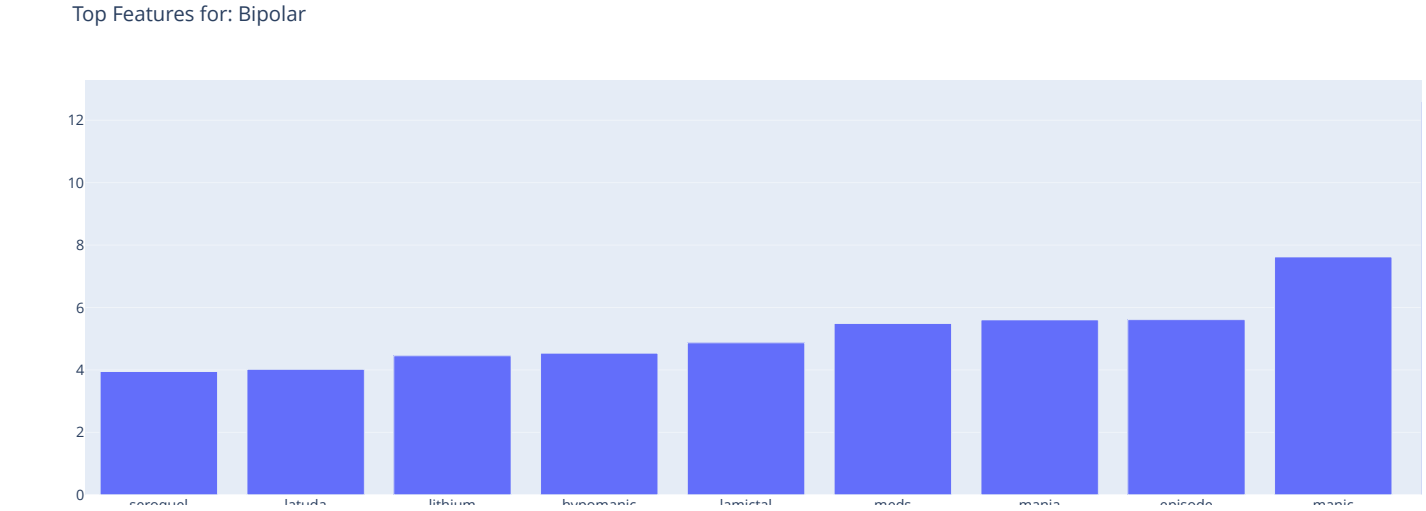


In [ ]:
```
all_text = ' '.join(new_df['cleaned_statement'])
wordcloud = WordCloud(width=800, height=400, background_color='white').generate(all_text)
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('Word Cloud of Cleaned Statements')
plt.show()
```

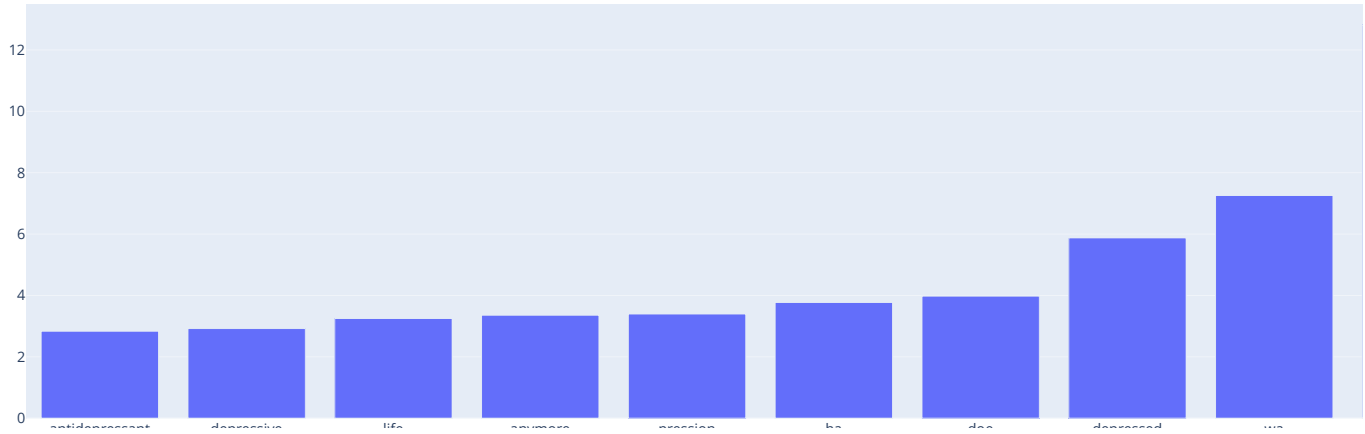


In [ ]:
```python
# Feature Importance
feature_names = vectorizer.get_feature_names_out()
coefs = best_model.coef_
for i, category in enumerate(best_model.classes_):
    top_features = coefs[i].argsort()[-10:]
    top_words = [feature_names[j] for j in top_features]
    top_scores = [coefs[i][j] for j in top_features]
    fig = go.Figure([go.Bar(x=top_words, y=top_scores)])
    fig.update_layout(title= 'Top Features for: ' + str(lb.classes_[category]))
    fig.show()
```

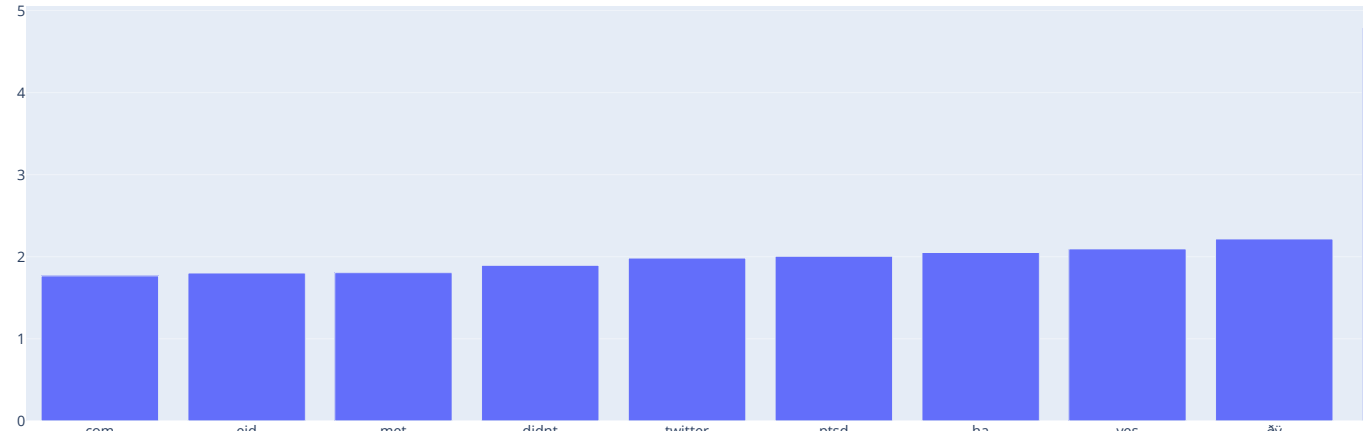




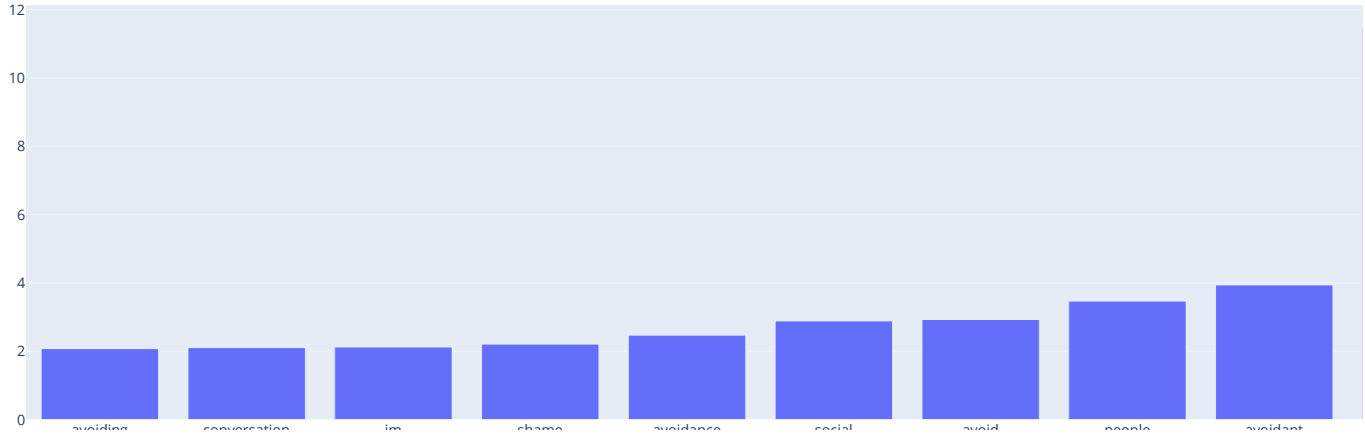
Top Features for: Depression



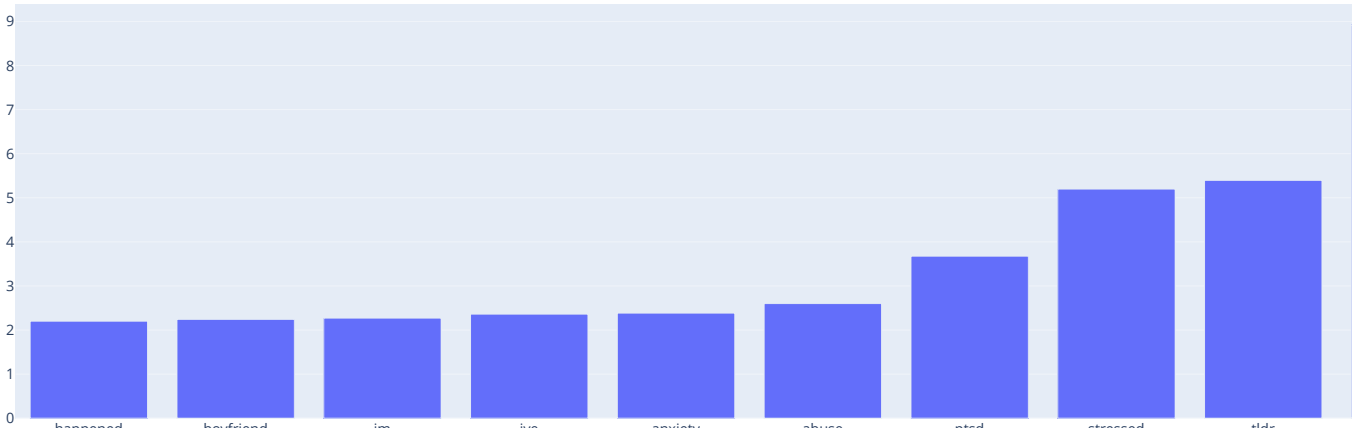
Top Features for: Normal



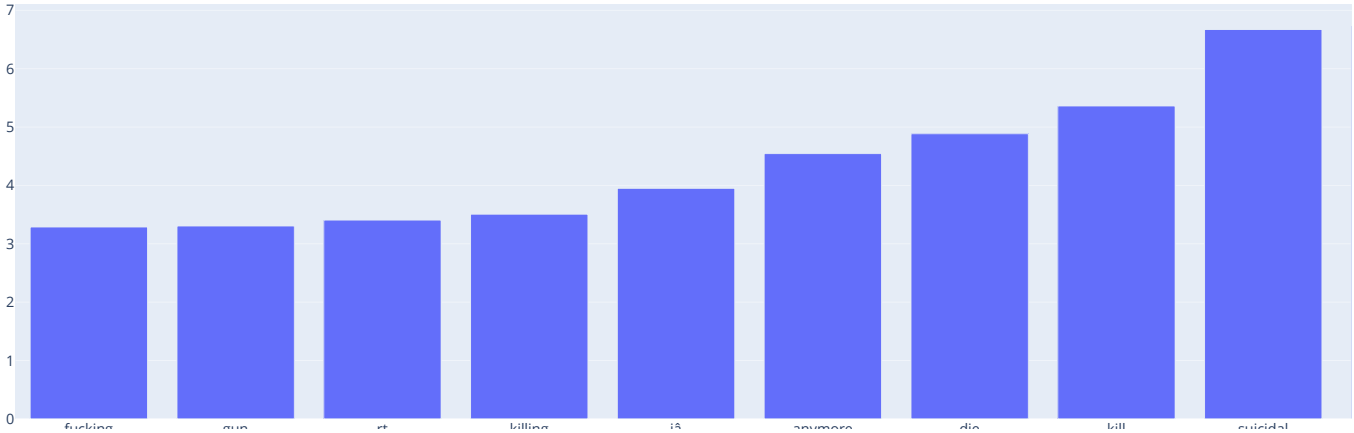
Top Features for: Personality disorder



Top Features for: Stress

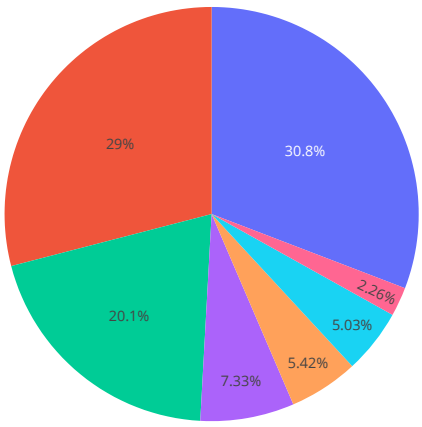


Top Features for: Suicidal



In [ ]:
```
# Status Distribution
fig = px.pie(df, names='status', title='Proportion of Each Status Category')
fig.show()
```

Proportion of Each Status Category



___Best Model: XG Boost Classifier___

In [ ]: