This notebook is our final model which is running a Support Vector Machines Model with the best parameters (from support_vector_machine.ipynb) using Faizan's preprocessing tools excluding the augmented text function using back translation.

Accuracy: 0.7684041851258365

In [12]:
```
import pandas as pd
import numpy as np
import re
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.svm import SVC
from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report, accuracy_score, confusion
import plotly.figure_factory as ff
```

In [3]:
```
# Load the data
path = './kaggle_sentiment_data.csv'
df = pd.read_csv(path)
```

In [4]:
```
# Handle NaN values in the statement column
df['statement'] = df['statement'].fillna('')
```

In [5]:
```
# Data Preprocessing
def preprocess_text(text):
    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)  # Remo
    text = re.sub(r'\n', '', text)  # Remove newlines
    text = re.sub(r'\w*\d\w*', '', text)  # Remove words containing numbers
    return text
```

In [6]:
```
# Tokenization and Stopwords Removal
stop_words = set(stopwords.words('english'))

def remove_stopwords(text):
    tokens = word_tokenize(text)
    tokens = [word for word in tokens if word not in stop_words]
    return ' '.join(tokens)
```

In [7]:
```
# Preprocess the text data
df['cleaned_statement'] = df['statement'].apply(preprocess_text).apply(remov

# Ensure no NaN values
df['cleaned_statement'] = df['cleaned_statement'].fillna('')
```

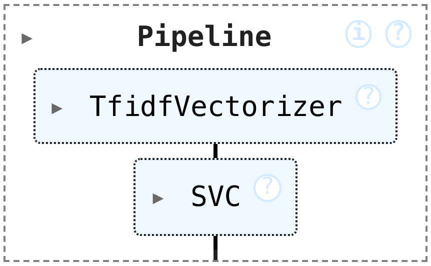
```
# Splitting the data
X = df['cleaned_statement']
y = df['status']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rar
```

In [8]:
```
pipeline = Pipeline([
    ('tfidf', TfidfVectorizer(max_features=2500, ngram_range=(1, 1))),
    ('svm', SVC(C=10, kernel='rbf', gamma='scale'))
])
```

In [9]:
```
# Train the pipeline
pipeline.fit(X_train, y_train)
```

Out[9]:



In [10]:
```
# Evaluate on the test set
y_pred = pipeline.predict(X_test)
```

In [11]:
```
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy: 0.7684041851258365
Classification Report:
                       precision    recall  f1-score   support

             Anxiety       0.84      0.76      0.80       778
             Bipolar       0.89      0.69      0.77       575
          Depression       0.69      0.75      0.72      3081
              Normal       0.84      0.95      0.90      3270
Personality disorder       0.91      0.49      0.64       240
              Stress       0.71      0.48      0.57       534
            Suicidal       0.70      0.64      0.67      2131

            accuracy                           0.77     10609
           macro avg       0.80      0.68      0.72     10609
        weighted avg       0.77      0.77      0.76     10609
```

In [27]:
```
# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
cm_fig = ff.create_annotated_heatmap(
    z=cm,
    x=list(("Anxiety", "Bipolar", "Depression", "Normal", "Personality Disor
    y=list(("Anxiety", "Bipolar", "Depression", "Normal", "Personality Disor
    annotation_text=cm,
    colorscale='Viridis'
)
```

```
cm_fig.update_layout(title='Confusion Matrix')
cm_fig.update_layout(title='Confusion Matrix', width=800, height=600)
cm_fig.show()
```