

```

import pandas as pd
import plotly.express as px
import plotly.graph_objects as go
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.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
import plotly.figure_factory as ff
from textblob import TextBlob
import numpy as np
from wordcloud import WordCloud
import matplotlib.pyplot as plt

```

```

# Load the data
path = '/content/Combined Data.csv'
df = pd.read_csv(path)

```

```

# Display the first few rows of the dataframe
print(df.head())

```

```

↗ 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

```

```

# EDA
print("Dataset Info:")
print(df.info())

```

```

↗ Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 54223 entries, 0 to 54222
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype
---  ---
 0   Unnamed: 0  54223 non-null  int64
 1   statement   53742 non-null  object
 2   status      54223 non-null  object
dtypes: int64(1), object(2)
memory usage: 1.2+ MB
None

```

```

print("Missing Values:")
print(df.isnull().sum())

```

```

↗ Missing Values:
Unnamed: 0      0
statement      481
status         0
dtype: int64

```

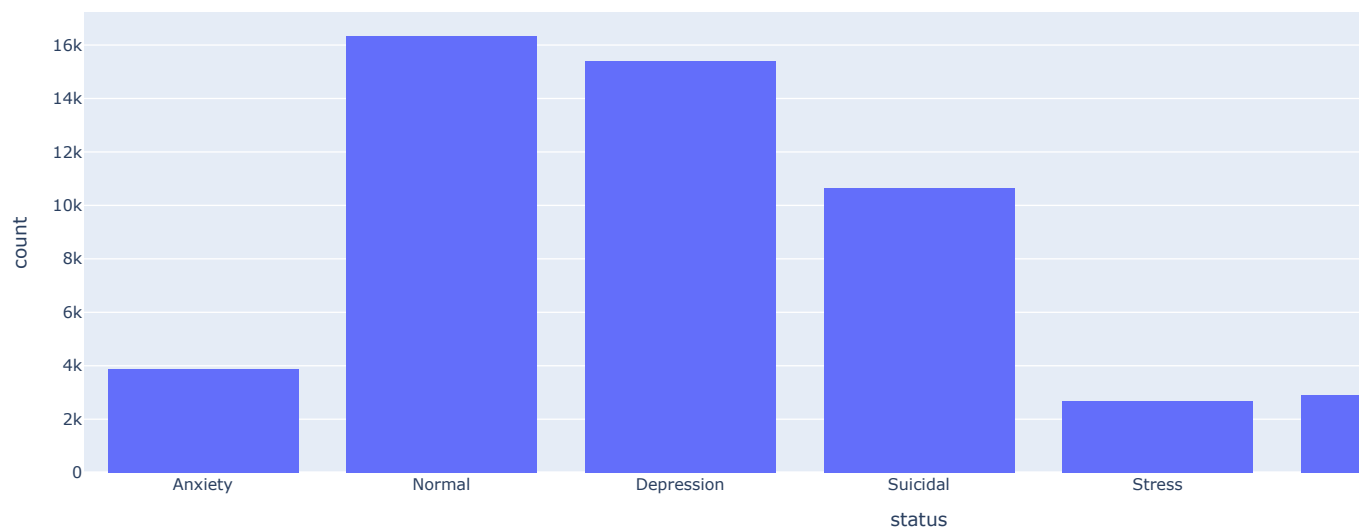
```

# Distribution of target labels
fig = px.histogram(df, x='status', title='Distribution of Mental Health Status')
fig.show()

```



Distribution of Mental Health Status

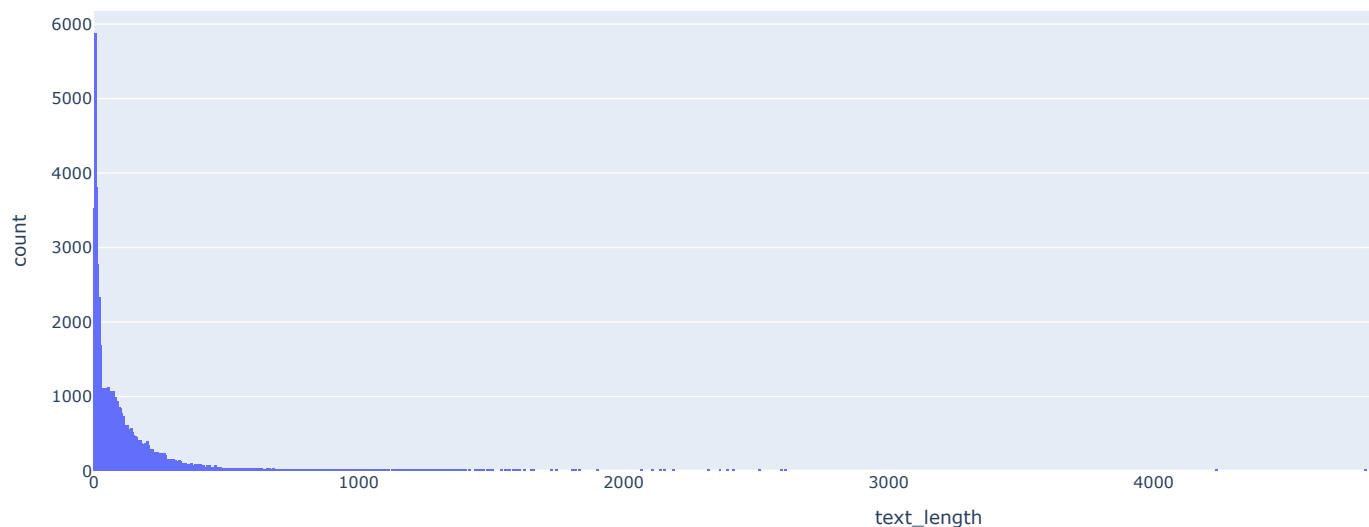


```
# Handle NaN values in the statement column
df['statement'] = df['statement'].fillna('')
```

```
# Text Length Distribution
df['text_length'] = df['statement'].apply(lambda x: len(str(x).split()))
fig = px.histogram(df, x='text_length', title='Text Length Distribution')
fig.show()
```



Text Length Distribution



```
# Data Preprocessing
nltk.download('stopwords')
nltk.download('punkt')

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) # Remove punctuation
text = re.sub(r'\n', '', text) # Remove newlines
text = re.sub(r'\w*\d\w*', '', text) # Remove words containing numbers
return text

df['cleaned_statement'] = df['statement'].apply(lambda x: preprocess_text(x))

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

# Download the 'punkt_tab' resource
nltk.download('punkt_tab')

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

# 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)

df['cleaned_statement'] = df['cleaned_statement'].apply(lambda x: remove_stopwords(x))

# Data Augmentation
def augment_text(text):
    try:
        blob = TextBlob(text)
        translated = blob.translate(to='fr').translate(to='en')
        return str(translated)
    except Exception as e:
        return text

df['augmented_statement'] = df['statement'].apply(augment_text)
augmented_df = df[['statement', 'status']].copy()
augmented_df['statement'] = df['augmented_statement']
df = pd.concat([df, augmented_df])

# Reapply preprocessing on augmented data
df['cleaned_statement'] = df['statement'].apply(lambda x: preprocess_text(x))
df['cleaned_statement'] = df['cleaned_statement'].apply(lambda x: remove_stopwords(x))

# Ensure no NaN values are left
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, random_state=42)

# Vectorization
vectorizer = TfidfVectorizer(max_features=10000)
X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)

# Model Training with Hyperparameter Tuning
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)

```

```
# Best Model
best_model = grid_search.best_estimator_

# Predictions
y_pred = best_model.predict(X_test_tfidf)

# Evaluation
print("Best Parameters:")
print(grid_search.best_params_)

print("Accuracy Score:")
print(accuracy_score(y_test, y_pred))

print("Classification Report:")
print(classification_report(y_test, y_pred))
```

Best Parameters:
{'C': 100}
Accuracy Score:
0.8656984785615491
Classification Report:

	precision	recall	f1-score	support
Anxiety	0.95	0.91	0.93	1525
Bipolar	0.94	0.89	0.91	1171
Depression	0.83	0.82	0.83	6165
Normal	0.93	0.95	0.94	6569
Personality disorder	0.89	0.90	0.89	937
Stress	0.87	0.85	0.86	1083
Suicidal	0.77	0.77	0.77	4240
accuracy			0.87	21690
macro avg	0.88	0.87	0.88	21690
weighted avg	0.87	0.87	0.87	21690

```
# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
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')
cm_fig.show()
```

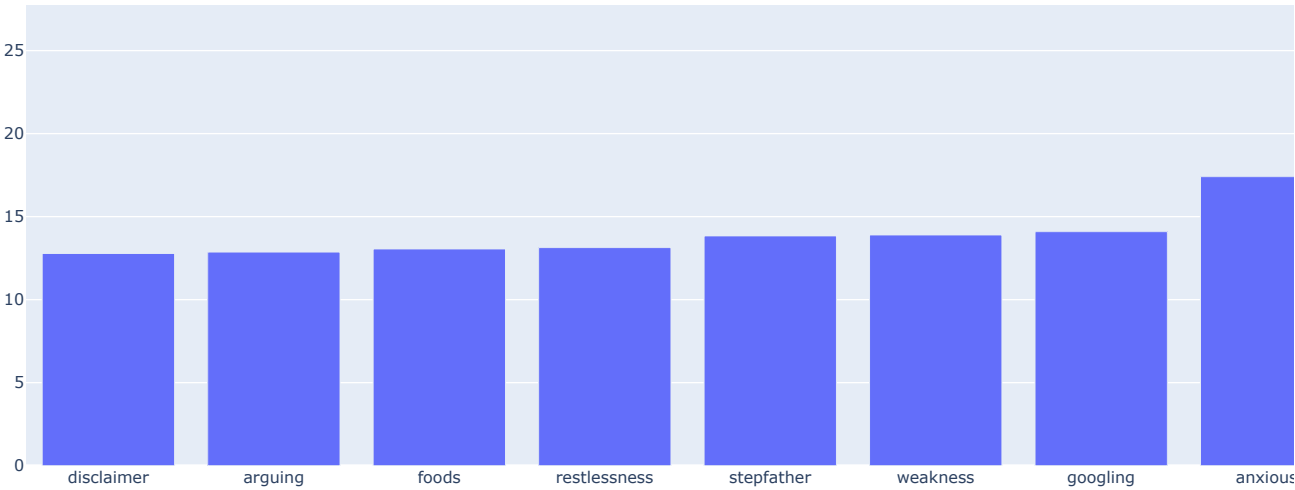
Confusion Matrix



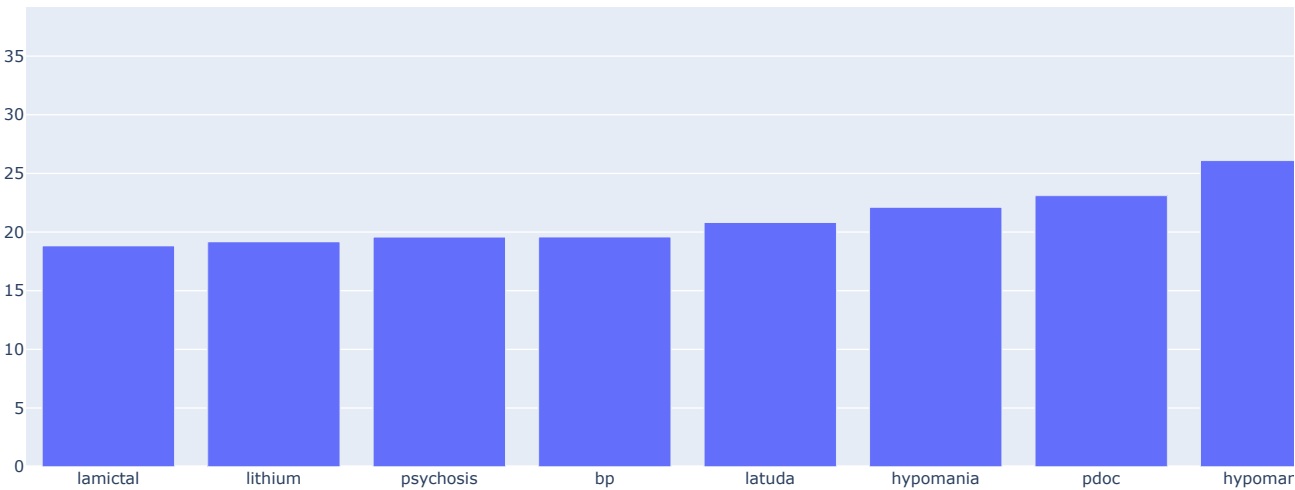
```
# Feature Importance
feature_names = vectorizer.get_feature_names_out()
coefs = best_model.coef_
for i, category in enumerate(best_model.classes_):
    top_features = coefs[i].argsort()[-10:]
    top_words = [feature_names[j] for j in top_features]
    top_scores = [coefs[i][j] for j in top_features]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



Top Features for Anxiety



Top Features for Bipolar



Top Features for Depression

