In [155…
```
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.ensemble import RandomForestClassifier
from sklearn.tree import plot_tree
from sklearn.metrics import confusion_matrix, classification_report, accurac
import matplotlib.pyplot as plt
import plotly.figure_factory as ff
from textblob import TextBlob
import numpy as np
from wordcloud import WordCloud
import matplotlib.pyplot as plt
```

In [156…
```
# Load the data
path = './kaggle_sentiment_data.csv'
df = pd.read_csv(path)
```

In [157…
```
# 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
```

In [158…
```
# EDA
print("Dataset Info:")
print(df.info())
```

```
Dataset 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
```

In [159…
```
print("Missing Values:")
print(df.isnull().sum())
```

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

In [160…
```
# Distribution of target labels
fig = px.histogram(df, x='status', title='Distribution of Mental Health Stat
fig.show()
```

In [161…
```
# Handle NaN values in the statement column
df = df.dropna(subset=['statement', 'status'])
```

In [162…
```
# 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()
```

In [163…
```
# 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)  # Remo
    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
[nltk_data]     C:\Users\ahuan\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\ahuan\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

In [164…
```
# 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_sto
```

In [165…
```
# # 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])
```

In [166…
```
# # Reapply preprocessing on augmented data
# df['cleaned_statement'] = df['statement'].apply(lambda x: preprocess_text(
# df['cleaned_statement'] = df['cleaned_statement'].apply(lambda x: remove_s
```

In [167…
```
# Ensure no NaN values are left
# df['cleaned_statement'] = df['cleaned_statement'].fillna('')
df = df.dropna(subset=['cleaned_statement', 'status'])
```

In [168…
```
# Splitting the data
X = df['cleaned_statement']
y = df['status']
```

In [169…
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, str
```

In [170…
```
# Vectorization
vectorizer = TfidfVectorizer(max_features=10000)
X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)
```

In [171…
```
# # 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_
```

In [ ]:
```
# Model Training with Hyperparameter Tuning
param_grid = {
    'max_depth': [75, 76, 77, 78, 79, 80],
}

model = RandomForestClassifier(criterion='gini', n_estimators=100, random_st
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_
```

In [173…
```
# Predictions
y_pred = best_model.predict(X_test_tfidf)
```

In [ ]:
```
# 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:
{'max_depth': 79}
Accuracy Score:
0.6917528708361014
Classification Report:
                     precision    recall  f1-score   support

            Anxiety       0.87      0.45      0.59       768
            Bipolar       0.97      0.44      0.61       556
         Depression       0.57      0.80      0.67      3081
             Normal       0.77      0.96      0.86      3269
Personality disorder       1.00      0.28      0.44       215
             Stress       0.95      0.22      0.35       517
           Suicidal       0.70      0.44      0.54      2131

           accuracy                           0.69     10537
          macro avg       0.83      0.51      0.58     10537
       weighted avg       0.73      0.69      0.67     10537
```

In [175…
```
# 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.update_layout(title='Confusion Matrix', width=800, height=600)
cm_fig.show()
```

In [176…
```
# # 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}', width=800, hei
#     fig.show()
```

In [177…

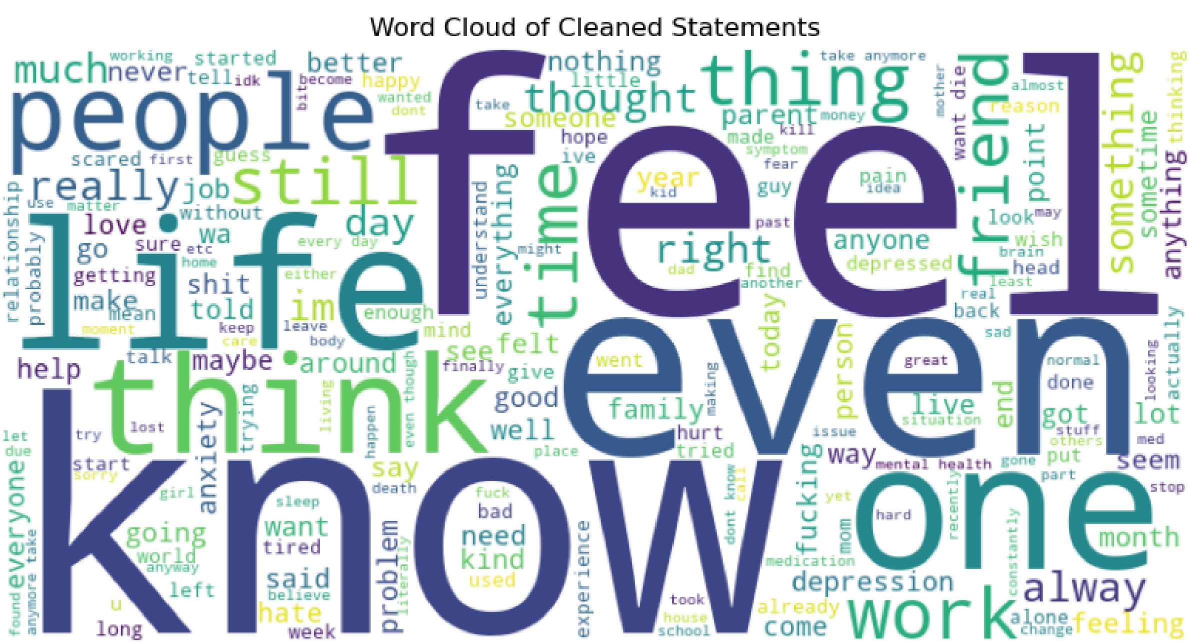
```
# # Adjust to plot in a single column with larger figure size for readabilit
# fig, axes = plt.subplots(nrows = 4, ncols = 1, figsize = (15, 35))

# # Plot each of the first 4 trees with larger font sizes
# id_counter = 0
# for i in range(4):  # Loop through the rows (since we are using a single c
#     ax = axes[i]
#     plot_tree(best_model.estimators_[id_counter], ax = ax,
#               class_names = best_model.classes_,
#               filled = True, fontsize=12)  # Adjusted fontsize for readabili
#     ax.set_title(f"Tree {id_counter}", fontsize=20, fontweight = 'bold')
#     id_counter += 1

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

In [178…

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



In [179…

```
# Status Distribution
fig = px.pie(df, names='status', title='Proportion of Each Status Category')
fig.update_layout(width=800, height=600)
fig.show()
```