AdolescentMind

Predicting Anxiety and Depression in Adolescents

Project Summary

AdolescentMind addresses the concern of adolescent mental health within Kenyan communities, research using data-driven insights is being carried out by different organisations such as `African Population and health  Research centers(APHCR)` to provide additional support to adolescents. An overview, the stakeholders involved, and the challenges surrounding the adolescent mental health and wellbeing space is provided at the beginning of the notebook. It is clear as outlined in the problem statement that children's and young peoples' participation in mental health surveys can only tell so much of the story in reasons why students may be struggling. There are high rates of psychological distress in students in and outside of school, the challenge of determining it, particularly in this case, is that mental health is stigmatized socially, compounded by few trained professionals that the students could reach out to. `The mental health (amendment) act of 2022` sets the tone for scaling data driven approaches to screening within schools and healthcare settings. The overall business and research objectives are also outlined and confirm the aim of predictive models being developed to identify at-risk students and understanding demographic variables most likely to predict negative mental health outcomes.

Data preparation and preprocessing, exploratory data analysis, visualization and statistical summary sections provide information on methods being used. Our analysis covers four counties; Kiambu, Makueni, Nairobi and Machakos. Visualizations and statistical summaries have revealed associations between demographic variables and anxiety or depression scores. The machine learning models, with the assistance of SHAP and Plotly visual interpretations, have identified the most important predictors as well as the interpretability of the results. The project concludes with actionable insights that could be transformative in reducing stigma, directing timely interventions, and reinforcing adolescent mental health across Kenya through data-driven early screening in schools.

The goal is to put all four models in use simultaneously, with the system selecting which model is the best predictor of the user’s input.Different models may produce better performance for different user entries. Utilizing the merits of each model, the odds of good predictions is increased.It allows for variability where one model may have weaker predictive ability, but another may perform well in the same context. By using all models and having one model selected dynamically for each prediction, the system produces assessments of mental health predicts that are more accurate and equitable.

1. Business Understanding

1.1. Business Overview

Mental health issues like **depression** and **anxiety** are increasingly becoming a top public health worry - especially amongst teenagers. `The World Health Organisation (WHO, 2023)` says that nearly 1 in 7 teenagers between the age of 10-19 experience some kind of mental health issue. Depression and anxiety are leading causes of illness and disability in this age group. Despite all awareness campaigns, mental health still carries a major stigma, especially in places like Kenya

This project tries to get around the problems with that by using machine learning techniques to develop a model that can pick up on the signs of depression and anxiety in teenagers based on their answers to the widely-used scales; `Patient Health Questionnaire (PHQ-9)` and `Generalized Anxiety Disorder Scale (GAD-7)`. They have been tested and widely used around the world for identifying early symptoms of depression and anxiety.

The goal is to develop a simple and scalable method of early mental health screening in Kenyan schools by analyzing the data on PHQ-9 and GAD-7 responses along with demographic information like age, gender, and school type. The aim is to help school counselors, teachers and healthcare providers spot at risk students early, and get them the support they need, to prevent mental illness from causing long-term problems. We hope to fit in with the requirements of `Kenya's Mental Health (Amendment) Act of 2022`, which strongly emphasizes early detection, prevention and integrating mental health services with education. Given all the rising concerns about adolescent mental health, getting this right, with a data-driven approach, could be a game-changer for boosting mental well-being, academic performance, and the overall wellbeing of teenagers in Kenya.

1.2. Stakeholder

- Counselors
- Teachers
- Healthcare providers

1.3. Problem Statement

`According to the Ministry of Health in 2021`, up to 45% of secondary school students exhibited symptoms indicative of psychological distress. However, mental health screening remains largely absent in most schools. Even when mental health services are available, institutions may lack trained professionals and on top of that, widespread stigma sorrounding mental illness often prevent students from seeking help at early stages.

Right now when it comes to identifying teenagers struggling with depression or anxiety, schools and healthcare institutions are mostly `relying on manual assessments by local counselors or healthcare workers` - a process that is time consuming and not always reliable, and really hard to scale up across schools. As a result, many cases get missed, students start to fall behind in their studies, they start self medicating with drugs and, in some cases the risk of suicide actually increases.

1.4. Business Objectives

1.4.1. Main Objective

- To build a machine learning model capable of identifying depression and anxiety levels among Kenyan adolescents using responses from PHQ, GAD and other demographic assessments.

1.4.2. Specific Objectives

- To analyze adolescent survey data to find out what the major demographic factors mostly contribute to depression and anxiety.
- To identify the major factors contributing to anxiety and/or depression.
- To identify how different counties contribute to students level of anxiety and depression.
- To interpret model outputs and find out what are the most important factors that contribute to depression and anxiety prediction.

1.5. Research Questions

1. What demographic factors are linked to depression and anxiety in Kenyan teenagers?
2. Can PHQ and GAD scores be used to predict depression and anxiety levels?
3. How are students affected by depression and anxiety in different counties ?
4. What are the most important features contributing to depression and anxiety?

1.6. Project Goals

- To build a machine learning model that classifies teenagers as depressed, anxious or none based on survey data.
- To combine demographic and psychological data (PHQ and GAD) for better mental health prediction.
- To get actionable insights that can help school administrators, counselors and policymakers support adolescent mental health.

1.7. Success Criteria

- Recall:

At least 80% recall and balanced performance across accuracy, precision, recall and F1-score.

- Reliability:

Model performs well validation and test data, no overfitting.

- Interpretability:

Key features used in predictions (e.g. PHQ, GAD items, demographics) are clear and explainable to non-technical people.

- Ethical and Practical:

System handles mental health data ethically and can be realistically implemented in school health systems.

- Impact:

Findings can inform early detection programs and resource allocation for adolescent mental health in Kenya.

2. Data Understanding

2.1. Data Source & Description

- The dataset source is from OSFstorage and the survey was conducted by Shamiri Institute research staff together with schools, teachers and students who also made contibutions in the research. It is publicly available and for more information about the dataset you can follow this link: https://osf.io/preprints/osf/yvdsc_v2

- The dataset contains surveys from about 17,000 adolescents in different Kenyan counties. Each row contains a students demographic background alongside the PHQ and GAD questions. Our main objective is to predict whether a person has Depression, Anxiety or none.

- For this project this dataset is suitable because because it has many predictors:

  - Demographics: Age, Gender, Form, Religion, Boarding_day e.t.c
  - Depression Indicator Questions: PHQ'S
  - Anxiety Indicator Questions: GAD'S

3. Data Preparation

3.1 Data Loading

```python
# importing the necessary libraries
import pandas as pd
from sklearn.impute import SimpleImputer
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA
import numpy as np
from xgboost import XGBClassifier
```

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

# loading the dataset
df = pd.read_csv('Merged Survey A to Survey F Data.csv')
```

3.2 Data Exploration

```
# previewing the first five records in the dataset
df.head()

  participant_ID survey_number          school_name   Age  Gender  Form
\
0            A_1      Survey A        the komarock  14.0     1.0   1.0

1            A_2      Survey A              kihara  13.0     1.0   1.0

2            A_3      Survey A            claycity  14.0     1.0   1.0

3            A_4      Survey A           kibichiku  17.0     2.0   3.0

4            A_5      Survey A  our lady of fatima  14.0     1.0   1.0


   Religion    Boarding_day School_type School_Demographics  ... HSB_4
HSB_5  \
0       1.0  Day & Boarding   Subcounty               Mixed  ...   NaN
NaN
1       1.0             Day   Subcounty               Mixed  ...   NaN
NaN
2       NaN             Day   Subcounty               Mixed  ...   NaN
NaN
3       2.0             Day   Subcounty               Mixed  ...   NaN
NaN
4       1.0             Day   Subcounty               Mixed  ...   NaN
NaN

   HSB_6  HSB_7  HSB_8  HSB_9  HSB_10  HSB_11  HSB_12  HSB_13
0    NaN    NaN    NaN    NaN     NaN     NaN     NaN     NaN
1    NaN    NaN    NaN    NaN     NaN     NaN     NaN     NaN
2    NaN    NaN    NaN    NaN     NaN     NaN     NaN     NaN
3    NaN    NaN    NaN    NaN     NaN     NaN     NaN     NaN
4    NaN    NaN    NaN    NaN     NaN     NaN     NaN     NaN

[5 rows x 191 columns]

# selecting the relevant features for our analysis
df = df[[
    "participant_ID","Age", "Gender", "Form", "Religion",
```

```
"Boarding_day", "School_type",
    "School_Demographics", "School_County",
    "Parents_Home", "Parents_Dead", "Fathers_Education",
"Mothers_Education",
    "Co_Curricular", "Sports", "Percieved_Academic_Abilities",
    "PHQ_1", "PHQ_2", "PHQ_3", "PHQ_4", "PHQ_5", "PHQ_6", "PHQ_7",
"PHQ_8", "PHQ_Functioning",
    "GAD_1", "GAD_2", "GAD_3", "GAD_4", "GAD_5", "GAD_6", "GAD_7",
"GAD_Check", "GAD_Functioning"
]]
```

```
# checking the dimension of the dataset
df.shape
```

```
(17089, 34)
```

```
# checking the overview of the data
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17089 entries, 0 to 17088
Data columns (total 34 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   participant_ID                17089 non-null  object
 1   Age                           15470 non-null  float64
 2   Gender                        16324 non-null  float64
 3   Form                          16595 non-null  float64
 4   Religion                      16230 non-null  float64
 5   Boarding_day                  17089 non-null  object
 6   School_type                   17089 non-null  object
 7   School_Demographics           17089 non-null  object
 8   School_County                 17089 non-null  object
 9   Parents_Home                  16512 non-null  float64
 10  Parents_Dead                  16464 non-null  float64
 11  Fathers_Education             16130 non-null  float64
 12  Mothers_Education             16308 non-null  float64
 13  Co_Curricular                 16284 non-null  float64
 14  Sports                        13836 non-null  float64
 15  Percieved_Academic_Abilities  16492 non-null  float64
 16  PHQ_1                         15972 non-null  float64
 17  PHQ_2                         16333 non-null  float64
 18  PHQ_3                         16190 non-null  float64
 19  PHQ_4                         16202 non-null  float64
 20  PHQ_5                         16205 non-null  float64
 21  PHQ_6                         16234 non-null  float64
 22  PHQ_7                         16299 non-null  float64
 23  PHQ_8                         15865 non-null  float64
 24  PHQ_Functioning               15765 non-null  float64
 25  GAD_1                         16167 non-null  float64
```

```
 26  GAD_2                              16255 non-null  float64
 27  GAD_3                              16336 non-null  float64
 28  GAD_4                              16138 non-null  float64
 29  GAD_5                              16161 non-null  float64
 30  GAD_6                              16209 non-null  float64
 31  GAD_7                              16259 non-null  float64
 32  GAD_Check                          14754 non-null  float64
 33  GAD_Functioning                    15815 non-null  float64
dtypes: float64(29), object(5)
memory usage: 4.4+ MB
```

```
# general statistics
df.describe()
```

```
                Age        Gender          Form      Religion
Parents_Home  \
count  15470.000000  16324.000000  16595.000000  16230.000000
16512.000000
mean      15.901551      1.462509      2.042904      1.901972
1.620882
std        1.421351      0.498608      1.007428      1.748903
0.540791
min       11.000000      1.000000      1.000000      1.000000
0.000000
25%       15.000000      1.000000      1.000000      1.000000
1.000000
50%       16.000000      1.000000      2.000000      1.000000
2.000000
75%       17.000000      2.000000      3.000000      2.000000
2.000000
max       25.000000      2.000000      4.000000      8.000000
2.000000

       Parents_Dead  Fathers_Education  Mothers_Education
Co_Curricular  \
count  16464.000000       16130.000000       16308.000000
16284.000000
mean       3.621113           2.442405           2.505580
1.883997
std        0.935719           1.191320           1.076189
0.791617
min        1.000000           1.000000           1.000000
1.000000
25%        4.000000           1.000000           2.000000
1.000000
50%        4.000000           3.000000           3.000000
2.000000
75%        4.000000           4.000000           3.000000
3.000000
max        4.000000           4.000000           4.000000
```

```
3.000000

                Sports  ...  PHQ_Functioning         GAD_1         GAD_2
\
count  13836.000000  ...      15765.000000  16167.000000  16255.000000

mean       1.416667  ...          0.939740      0.771634      1.100954

std        0.493024  ...          0.806153      0.926899      1.090507

min        1.000000  ...          0.000000      0.000000      0.000000

25%        1.000000  ...          0.000000      0.000000      0.000000

50%        1.000000  ...          1.000000      1.000000      1.000000

75%        2.000000  ...          1.000000      1.000000      2.000000

max        2.000000  ...          3.000000      3.000000      3.000000


                 GAD_3         GAD_4         GAD_5         GAD_6
GAD_7  \
count  16336.000000  16138.000000  16161.000000  16209.000000
16259.000000
mean       1.232431      0.690234      0.538086      1.034734
1.036411
std        1.107131      0.957693      0.860822      1.064766
1.058983
min        0.000000      0.000000      0.000000      0.000000
0.000000
25%        0.000000      0.000000      0.000000      0.000000
0.000000
50%        1.000000      0.000000      0.000000      1.000000
1.000000
75%        2.000000      1.000000      1.000000      2.000000
2.000000
max        3.000000      3.000000      3.000000      3.000000
3.000000

          GAD_Check  GAD_Functioning
count  14754.000000     15815.000000
mean       0.167141         0.889915
std        0.559351         0.821541
min        0.000000         0.000000
25%        0.000000         0.000000
50%        0.000000         1.000000
75%        0.000000         1.000000
max        3.000000         3.000000
```

```
[8 rows x 29 columns]
```

The min and max values for columns like GAD and PHQ are 0 to 3 and after some research we concluded that they are scores which could mean:

- 0 = Not at all
- 1 = Several days
- 2 = More than half the days
- 3 = Nearly every day

```
df.select_dtypes(include=['object',
'float64','int64']).nunique().sort_values(ascending=False)
```

```
participant_ID                 17089
Age                               15
Religion                           7
Percieved_Academic_Abilities       5
School_County                      4
Form                               4
Fathers_Education                  4
Mothers_Education                  4
Parents_Dead                       4
PHQ_2                              4
PHQ_1                              4
PHQ_Functioning                    4
PHQ_8                              4
PHQ_7                              4
PHQ_6                              4
PHQ_5                              4
PHQ_4                              4
PHQ_3                              4
GAD_2                              4
GAD_3                              4
GAD_4                              4
GAD_5                              4
GAD_6                              4
GAD_7                              4
GAD_Check                          4
GAD_1                              4
GAD_Functioning                    4
School_Demographics                3
Parents_Home                       3
School_type                        3
Boarding_day                       3
Co_Curricular                      3
Sports                             2
Gender                             2
dtype: int64
```

- Most columns have a reasonable number of unique categories

3.3 Check for missing values

We check for missing values in the dataset and evaluate their number to decide on the best way on handling them.

```
# checking for missing values
df.isna().sum()
```

```
participant_ID                    0
Age                            1619
Gender                          765
Form                            494
Religion                        859
Boarding_day                      0
School_type                       0
School_Demographics               0
School_County                     0
Parents_Home                    577
Parents_Dead                    625
Fathers_Education               959
Mothers_Education               781
Co_Curricular                   805
Sports                         3253
Percieved_Academic_Abilities    597
PHQ_1                          1117
PHQ_2                           756
PHQ_3                           899
PHQ_4                           887
PHQ_5                           884
PHQ_6                           855
PHQ_7                           790
PHQ_8                          1224
PHQ_Functioning                1324
GAD_1                           922
GAD_2                           834
GAD_3                           753
GAD_4                           951
GAD_5                           928
GAD_6                           880
GAD_7                           830
GAD_Check                      2335
GAD_Functioning                1274
dtype: int64
```

- Many columns contain missing values.
- Though the percentage of the missing values per column is not large we have to come up with a good way of filling in the missing values because dropping those missing values could have an impact on our model as they are important.

3.4 Check for duplicate values

```python
# checking for duplicate values
df.duplicated().sum()
```

```
np.int64(0)
```

We have no duplicate records.

3.5 Dealing with missing values

```python
# Separate numerical and categorical
num_cols = df.select_dtypes(include=['float64']).columns
cat_cols = df.select_dtypes(include='object').columns

# Median for numerical
num_imputer = SimpleImputer(strategy='median')
df[num_cols] = num_imputer.fit_transform(df[num_cols])

# Mode for categorical
cat_imputer = SimpleImputer(strategy='most_frequent')
df[cat_cols] = cat_imputer.fit_transform(df[cat_cols])

df.isnull().sum()
```

```
participant_ID                  0
Age                             0
Gender                          0
Form                            0
Religion                        0
Boarding_day                    0
School_type                     0
School_Demographics             0
School_County                   0
Parents_Home                    0
Parents_Dead                    0
Fathers_Education               0
Mothers_Education               0
Co_Curricular                   0
Sports                          0
Percieved_Academic_Abilities    0
PHQ_1                           0
PHQ_2                           0
PHQ_3                           0
PHQ_4                           0
PHQ_5                           0
PHQ_6                           0
PHQ_7                           0
PHQ_8                           0
PHQ_Functioning                 0
```

```
GAD_1                           0
GAD_2                           0
GAD_3                           0
GAD_4                           0
GAD_5                           0
GAD_6                           0
GAD_7                           0
GAD_Check                       0
GAD_Functioning                 0
dtype: int64
```

3.6 Exploratory Data Analysis

3.6.1 Univariate Analysis

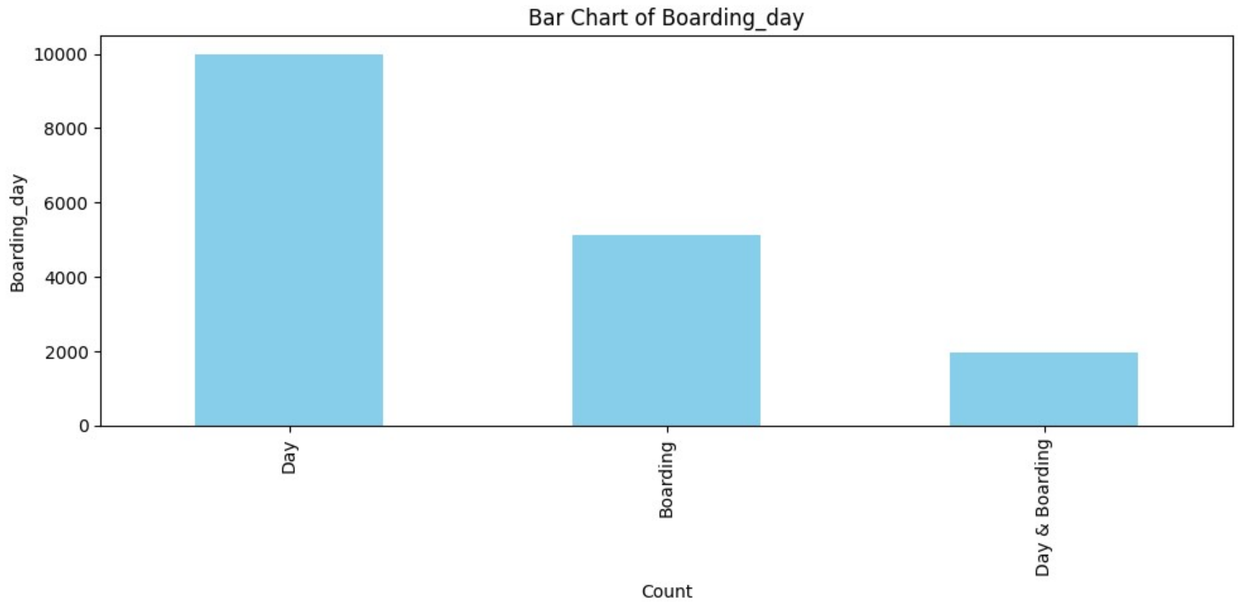
Distribution of features.

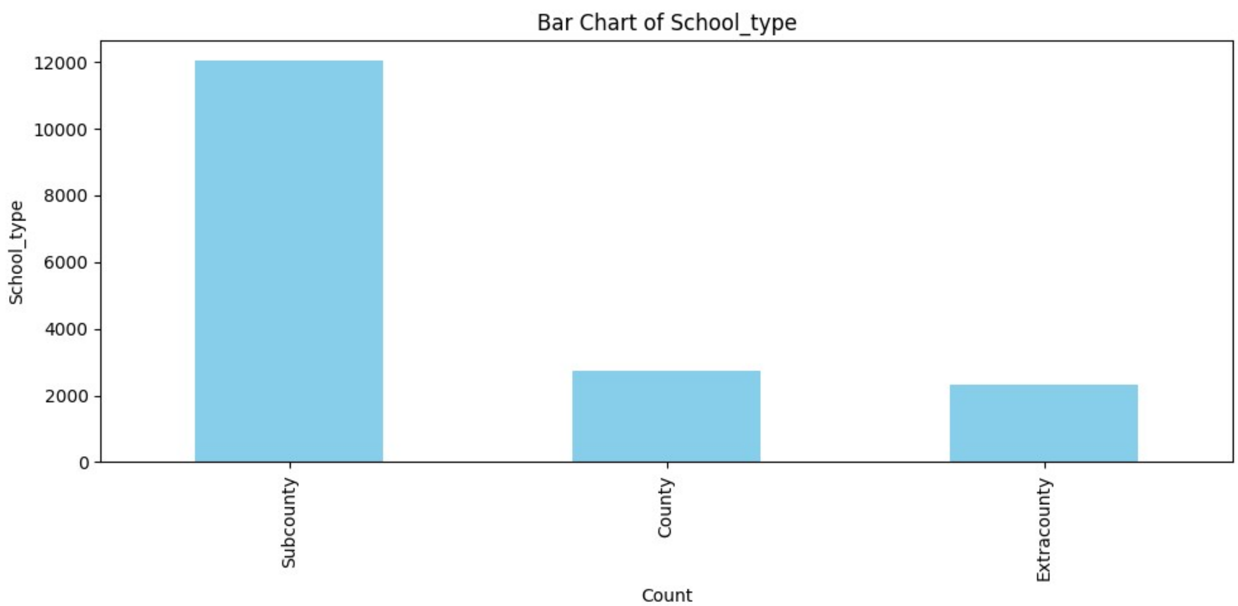
```
# Checking how the features are distributed
df.hist(bins=20, figsize=(19, 15))
plt.suptitle("Feature Distributions")
plt.show()
```

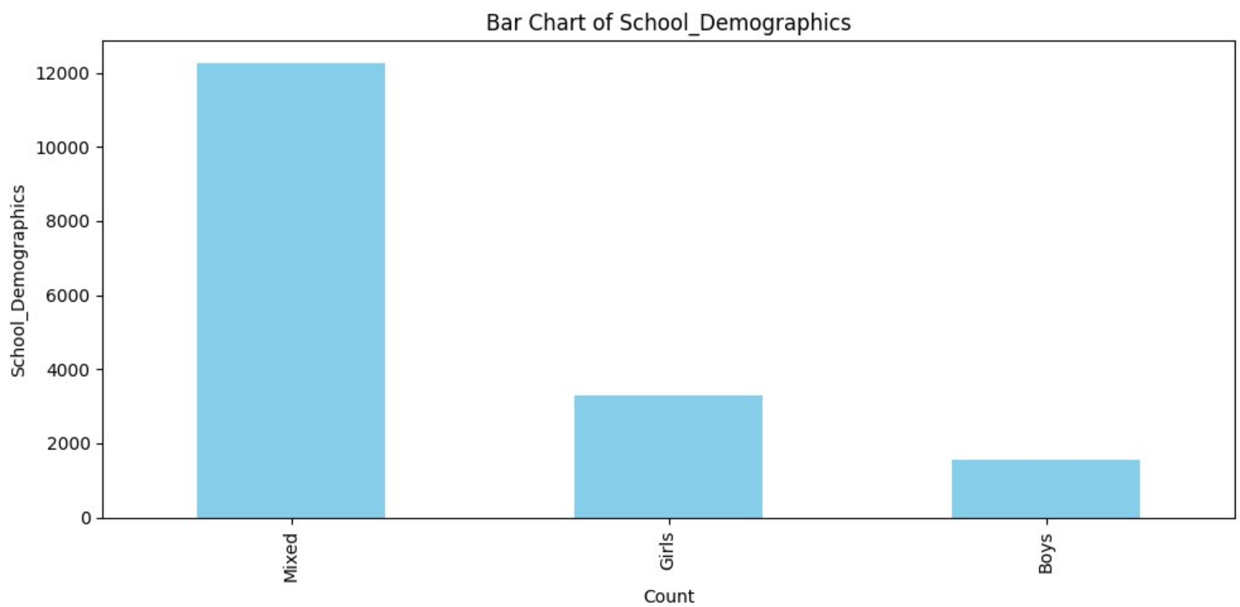

```python
# checking for age valuecounts
df['Age'].value_counts()
```

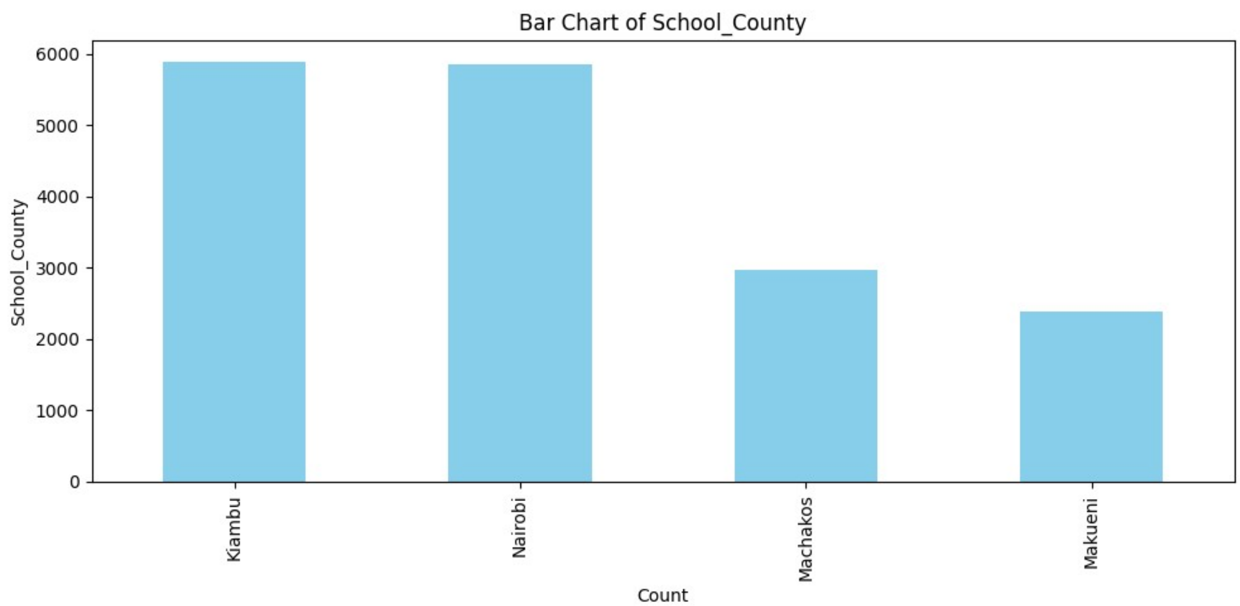
```
Age
16.0    5649
15.0    3881
17.0    2997
14.0    2195
18.0    1430
19.0     447
13.0     340
20.0     111
21.0      16
12.0       9
22.0       8
24.0       2
23.0       2
```

```
11.0        1
25.0        1
Name: count, dtype: int64

categorical_columns =
df.select_dtypes(include=['object']).drop(columns=['participant_ID'],
errors='ignore')

# initialize a list to store findings
categorical_findings = []

# set a for loop to "loop" through the categorical columns set above
for column in categorical_columns.columns:
    #set figure size
    plt.figure(figsize=(10, 5))

    #show the value counts
    value_counts = df[column].value_counts()

    # Selecting to show the top categories for readability
    top_categories = value_counts[:20]

    # visuaizing using a bar chart
    top_categories.plot(kind='bar', color='skyblue')

    #plot titles
    plt.title(f'Bar Chart of {column}')
    plt.xlabel('Count')
    plt.ylabel(column)

    # Adjust layout for readability
    plt.tight_layout()
    plt.show()

     # Collect findings
    categorical_findings.append(
        f"Column '{column}' has {len(value_counts)} unique categories.
The most common category is '{value_counts.idxmax()}' with
{value_counts.max()} entries."
    )

categorical_findings
```


Bar Chart of Boarding_day
Boarding_day
10000
8000
6000
4000
2000
0
Day
Boarding
Day & Boarding
Count


Bar Chart of School_type
School_type
12000
10000
8000
6000
4000
2000
0
Subcounty
County
Extracounty
Count






```
["Column 'Boarding_day' has 3 unique categories. The most common
category is 'Day' with 9983 entries.",
 "Column 'School_type' has 3 unique categories. The most common
category is 'Subcounty' with 12034 entries.",
 "Column 'School_Demographics' has 3 unique categories. The most
common category is 'Mixed' with 12252 entries.",
 "Column 'School_County' has 4 unique categories. The most common
category is 'Kiambu' with 5888 entries."]
```

Findings:

There are 4 `categorical columns` for which we have plotted the bar charts above. Here are some notable observations:

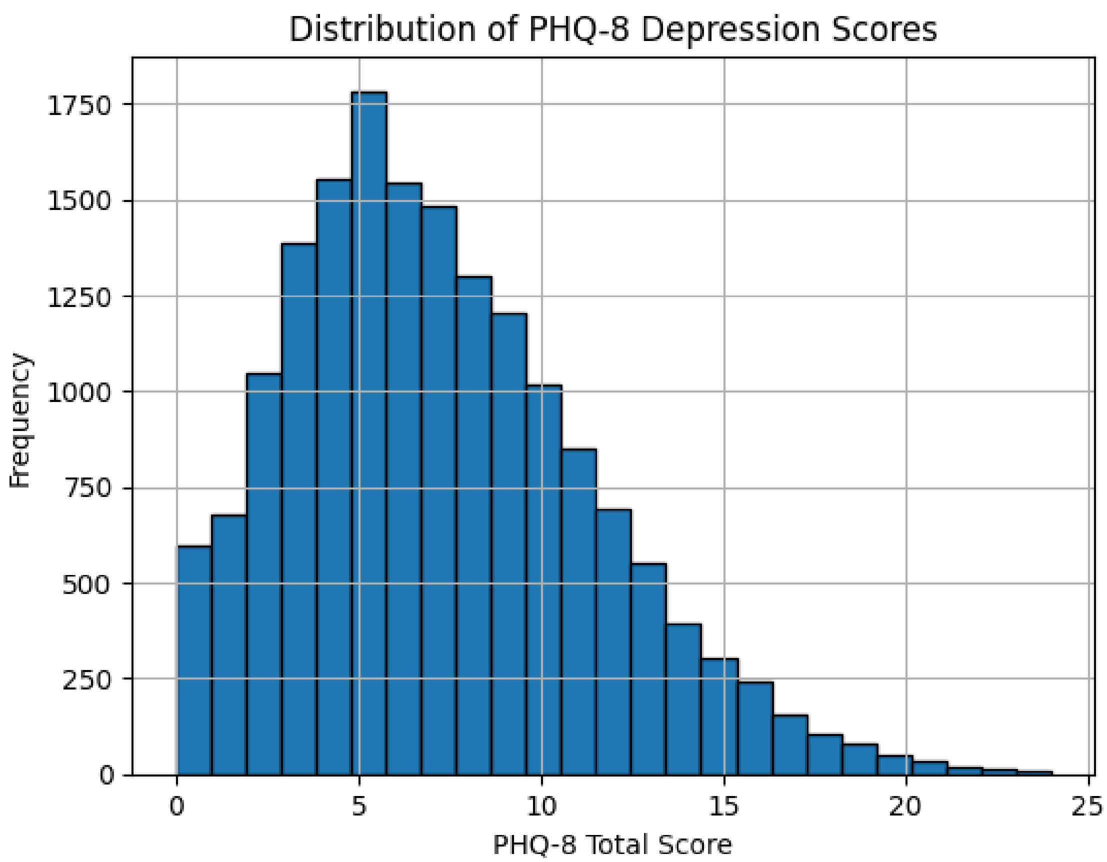
1. **Boarding_day**:
   - This column has 3 unique categories. The most common name is "Day" appearing 9983 times. This column is useful for analysis.
2. **School_type**:
   - There are 3 categories. The most common category is Subcounty with 12034 entries, suggesting a high prevalence of subcounty schools in the dataset.
3. **School_Demographics**:
   - This column has 3 categories, with Mixed being the most common, it is represented by 12252 entries. This indicates that the majority of schools surveyed are Mixed High Schools.
4. **School_County**:
   - There are 4 categories (Kiambu, Nairobi, Machakos, Makueni). Kiambu is the most frequent, with 5888 entries, reflecting that most schools surveyed are from that region.

```
# Depression total and category
df['PHQ_Totals'] = df[[f'PHQ_{i}' for i in range(1, 9)]].sum(axis=1)

# Anxiety total and category
df['GAD_Totals'] = df[[f'GAD_{i}' for i in range(1, 8)]].sum(axis=1)

def phq_category(score):
    if score <= 4: return 0   # none
    elif score <= 9: return 1 # mild
    elif score <= 14: return 2 # moderate
    elif score <= 19: return 3 # moderately severe
    else: return 4             # severe

def gad_category(score):
    if score <= 4: return 0   # minimal
    elif score <= 9: return 1 # mild
    elif score <= 14: return 2 # moderate
    else: return 3             # severe

# Apply the numeric classification
df['Is_Depressed'] = df['PHQ_Totals'].astype(int).apply(phq_category)
df['Has_anxiety'] = df['GAD_Totals'].astype(int).apply(gad_category)
```

- Saving the cleaned dataset in our desktop to use it later in Tableau

```python
# Distribution of total scores
df['PHQ_Totals'] = df[[f'PHQ_{i}' for i in range(1,9)]].sum(axis=1)
df['PHQ_Totals'].hist(bins=25, edgecolor='black')
plt.title('Distribution of PHQ-8 Depression Scores')
plt.xlabel('PHQ-8 Total Score')
plt.ylabel('Frequency')
```

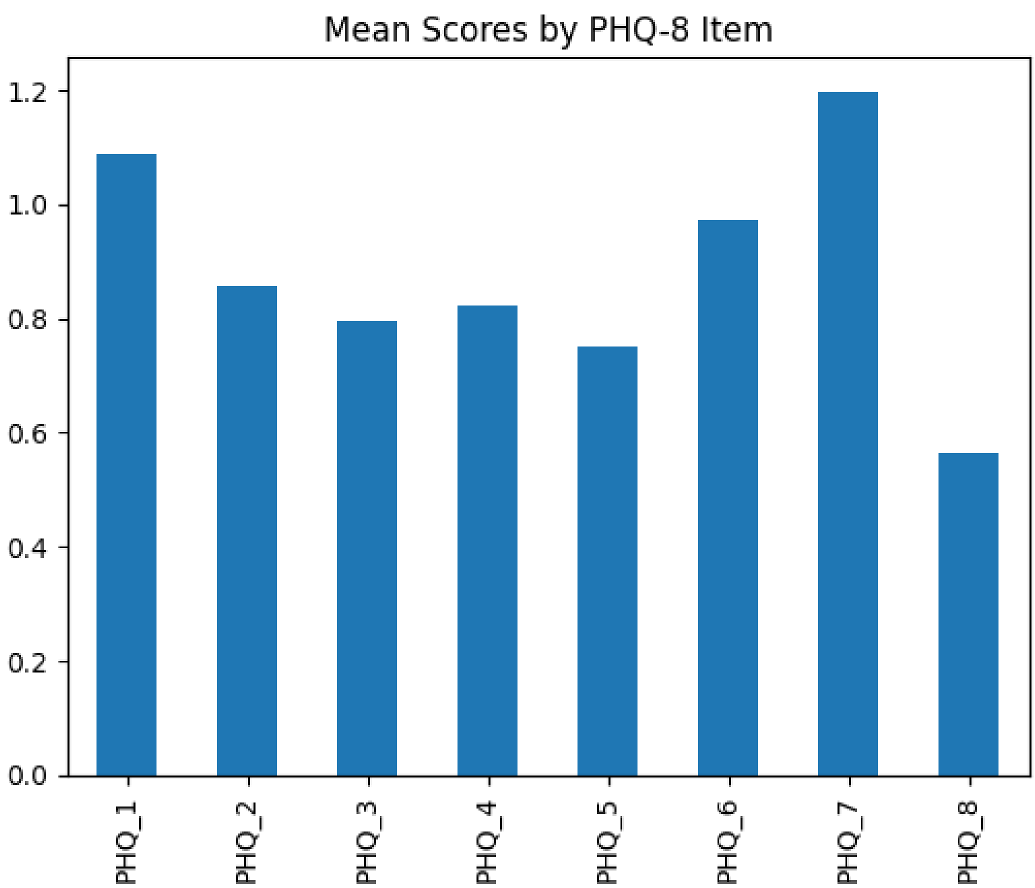
```
Text(0, 0.5, 'Frequency')
```



- The majority of participants have low to moderate PHQ-8 depression scores, indicating `mild` or `minimal` depressive symptoms in most of the sample. The distribution is `positively skewed`, with fewer individuals exhibiting high levels of depressive symptoms.

```python
# Calculating PHQ mean
phq_items = [f'PHQ_{i}' for i in range(1,9)]
df[phq_items].mean().plot(kind='bar')
plt.title('Mean Scores by PHQ-8 Item')
```

```
Text(0.5, 1.0, 'Mean Scores by PHQ-8 Item')
```

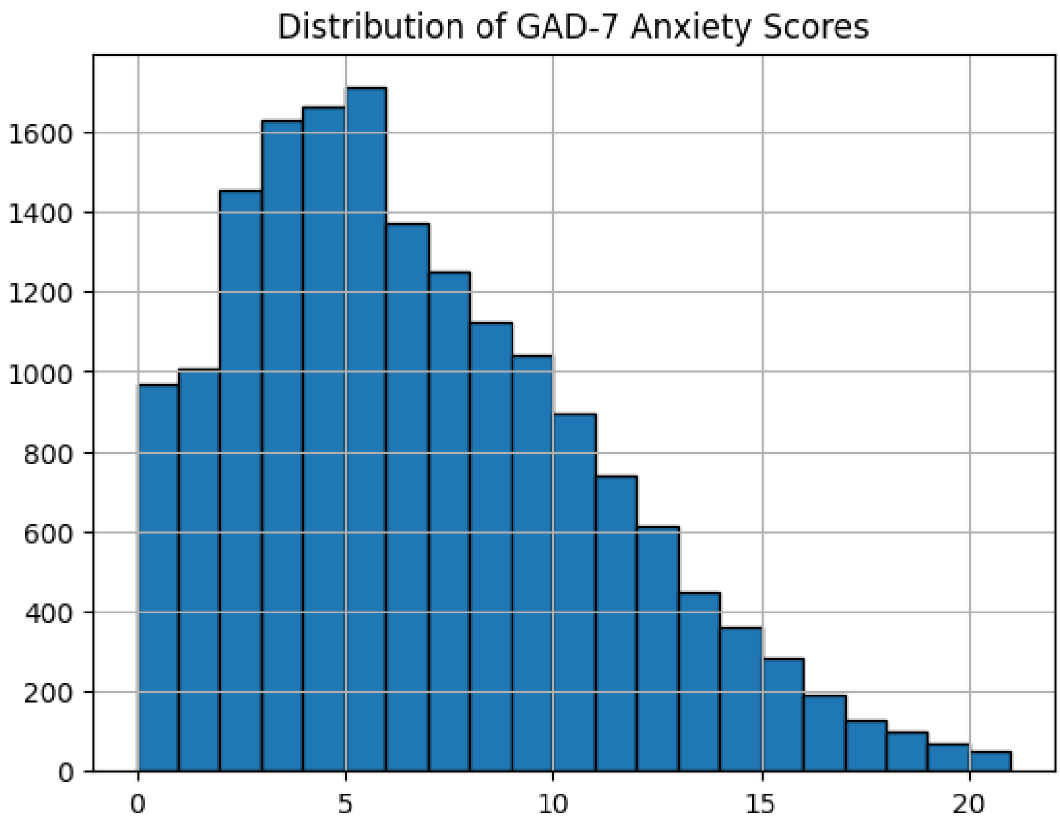


The mean item scores suggest that `loss of interest` and `concentration difficulties` are the most frequently reported depressive symptoms, while `psychomotor symptoms` are least reported. This indicates **variability in symptom expression across the sample**.

```
phq_columns = [f'PHQ_{i}' for i in range(1, 9)]

for col in phq_columns:
    unique_vals = df[col].unique()
    print(f"{col} unique values: {unique_vals}\n")
```

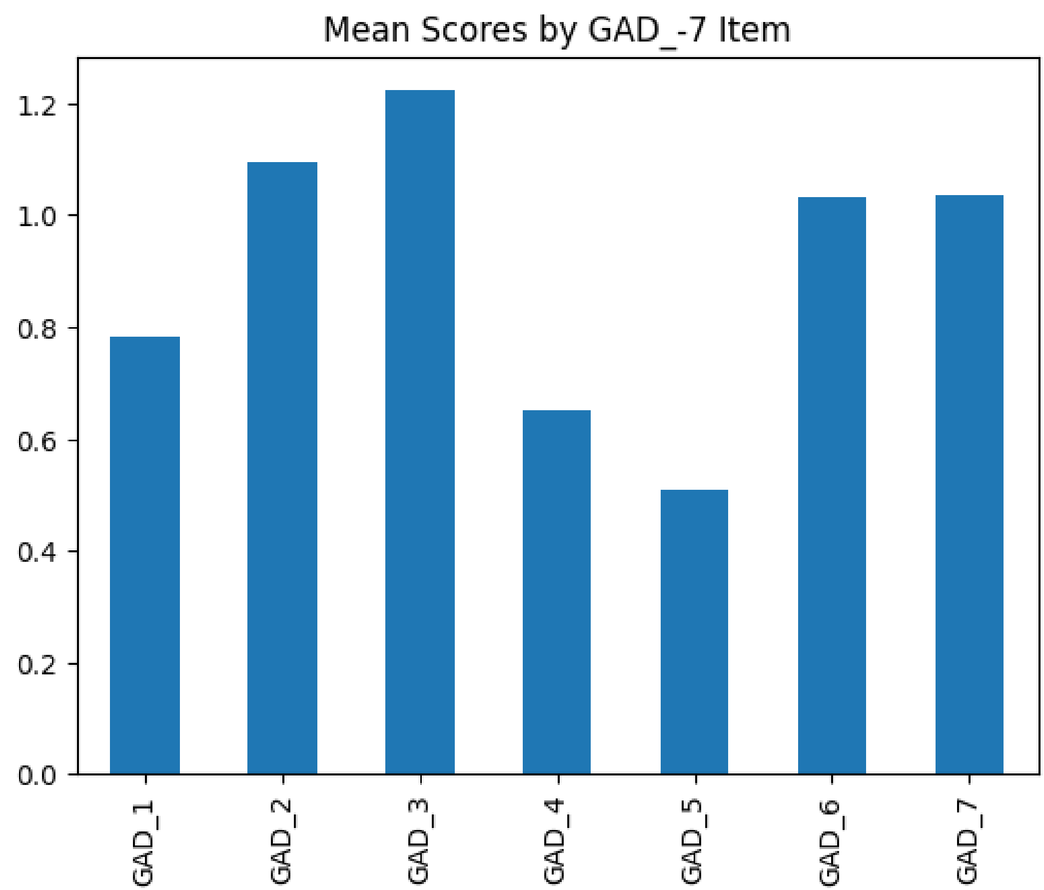
```
PHQ_1 unique values: [0. 2. 1. 3.]

PHQ_2 unique values: [0. 1. 3. 2.]

PHQ_3 unique values: [0. 3. 1. 2.]

PHQ_4 unique values: [0. 2. 1. 3.]

PHQ_5 unique values: [0. 1. 3. 2.]

PHQ_6 unique values: [0. 3. 2. 1.]
```

```
PHQ_7 unique values: [0. 3. 1. 2.]

PHQ_8 unique values: [0. 1. 2. 3.]


# Calculating GAD mean
df['GAD_Totals'] = df[[f'GAD_{i}' for i in range(1,8)]].sum(axis=1)
df['GAD_Totals'].hist(bins=21, edgecolor='black')
plt.title('Distribution of GAD-7 Anxiety Scores')

Text(0.5, 1.0, 'Distribution of GAD-7 Anxiety Scores')
```



The distribution is `positively skewed`, with fewer individuals exhibiting high levels of anxiety symptoms

```
phq_items = [f'GAD_{i}' for i in range(1,8)]
df[phq_items].mean().plot(kind='bar')
plt.title('Mean Scores by GAD_-7 Item')
plt.savefig('Image/GAD_Items.png', dpi=300, bbox_inches='tight')
```



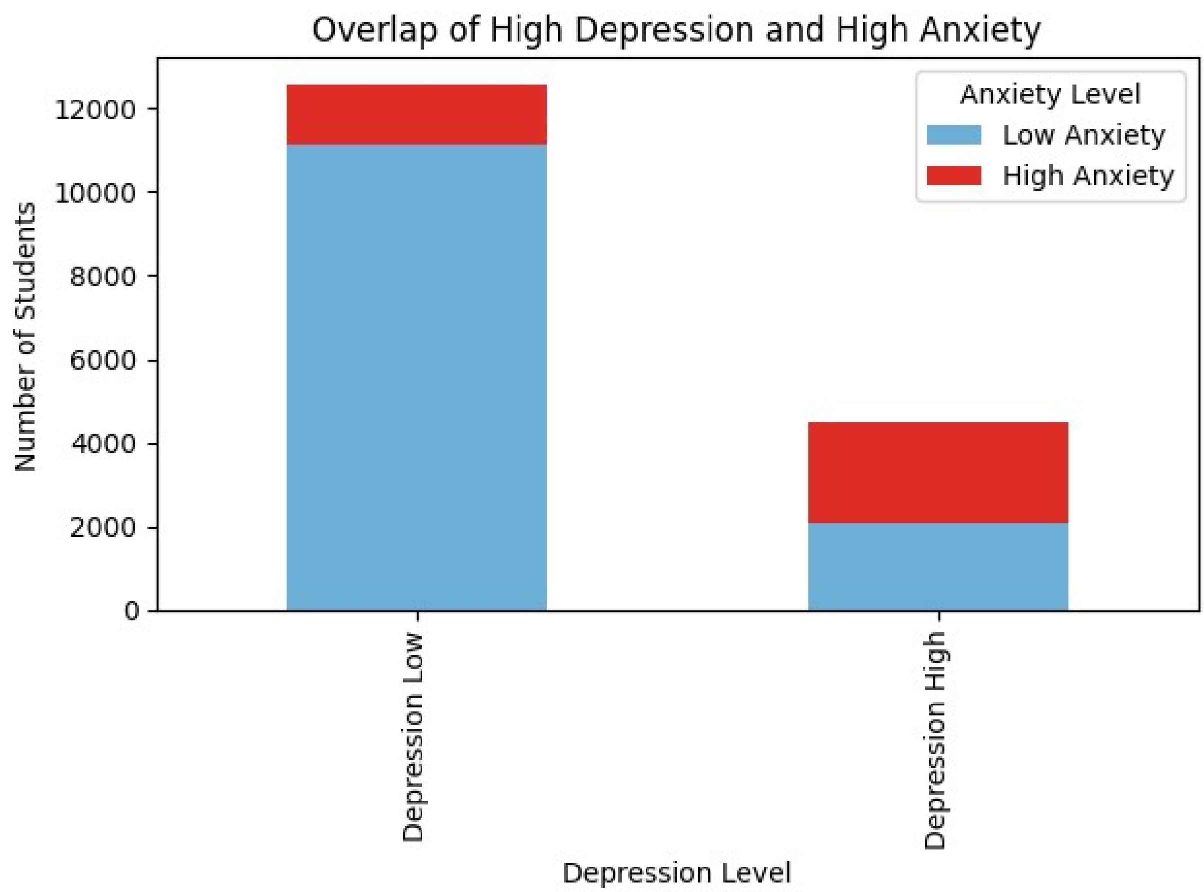

```
phq_columns = [f'GAD_{i}' for i in range(1, 8)]

for col in phq_columns:
    unique_vals = df[col].unique()
    print(f"{col} unique values: {unique_vals}\n")
```

```
GAD_1 unique values: [0. 1. 2. 3.]

GAD_2 unique values: [1. 3. 2. 0.]

GAD_3 unique values: [0. 3. 1. 2.]

GAD_4 unique values: [0. 1. 3. 2.]

GAD_5 unique values: [1. 0. 2. 3.]

GAD_6 unique values: [1. 2. 0. 3.]

GAD_7 unique values: [1. 2. 0. 3.]
```

Comorbidity Analysis

What percentage of students have both high depression and anxiety?

```
# Crosstab of depression and anxiety
cross_tab = pd.crosstab(df['Is_Depressed'], df['Has_anxiety'],
margins=True)


data = {
    'anxiety_low': [11131, 2086],
    'anxiety_high': [1442, 2430]
}
index = ['Depression Low', 'Depression High']

# Create DataFrame
data1 = pd.DataFrame(data, index=index)

# Plot stacked bar chart
data1.plot(kind='bar', stacked=True, color=['#6baed6', '#de2d26'])
plt.title('Overlap of High Depression and High Anxiety')
plt.ylabel('Number of Students')
plt.xlabel('Depression Level')
plt.legend(title='Anxiety Level', labels=['Low Anxiety', 'High
Anxiety'])
plt.savefig('Image/Overlap of High Depression and High Anxiety.png',
dpi=300, bbox_inches='tight')
plt.tight_layout()
plt.show()
```



- This shows how many students fall into each group.
- The `red` portion represents students with high anxiety, and within that, you can clearly see the large segment overlapping with high depression.

3.6.2 Bivariate analysis

Risk Factor Analysis

Gender Differences

```
# Import libraries
from statsmodels.multivariate.manova import MANOVA

# testing whether Gender affects both Depression and Anxiety together
manova = MANOVA.from_formula('PHQ_Totals + GAD_Totals ~ Gender',
data=df)
print(manova.mv_test())
```

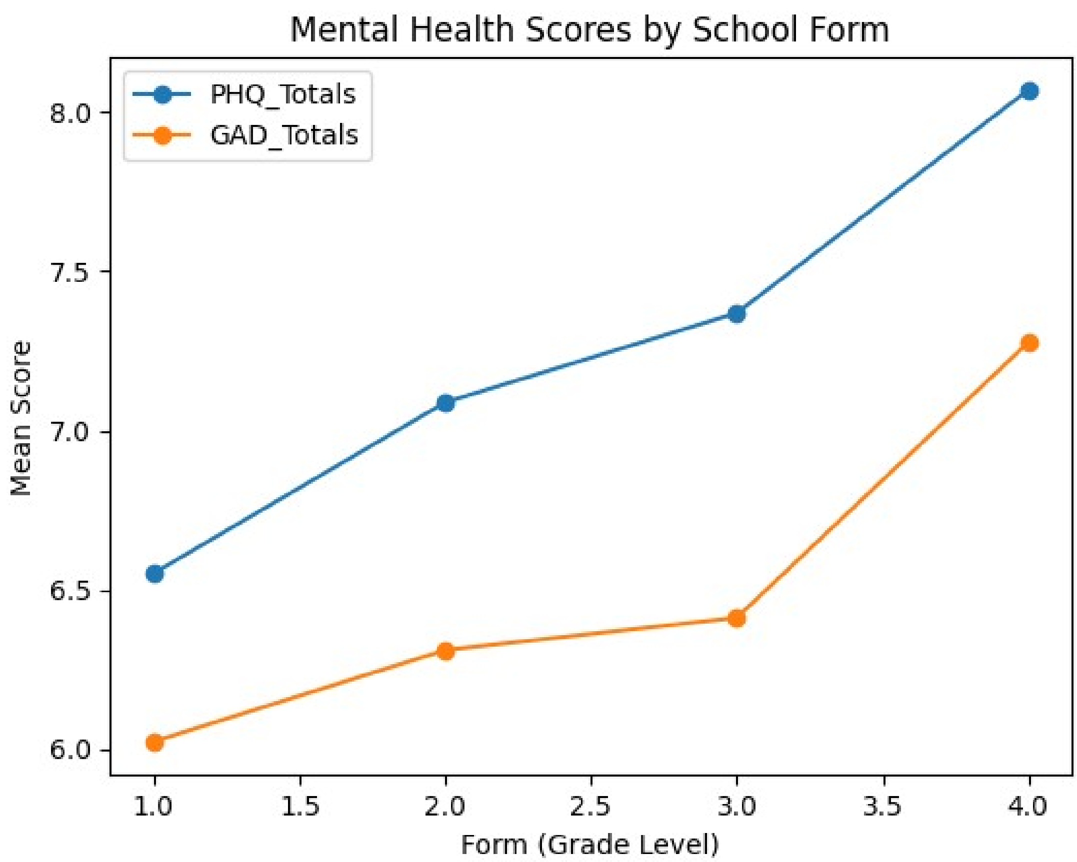
```
                    Multivariate linear model
=================================================================

-----------------------------------------------------------------
       Intercept        Value  Num DF   Den DF    F Value  Pr > F
```

```
-----------------------------------------------------------------
         Wilks' lambda 0.7015 2.0000 17086.0000 3635.9459 0.0000
        Pillai's trace 0.2985 2.0000 17086.0000 3635.9459 0.0000
Hotelling-Lawley trace 0.4256 2.0000 17086.0000 3635.9459 0.0000
   Roy's greatest root 0.4256 2.0000 17086.0000 3635.9459 0.0000
-----------------------------------------------------------------

-----------------------------------------------------------------
           Gender         Value  Num DF    Den DF   F Value Pr > F
-----------------------------------------------------------------
            Wilks' lambda 0.9898 2.0000 17086.0000 87.9034 0.0000
           Pillai's trace 0.0102 2.0000 17086.0000 87.9034 0.0000
   Hotelling-Lawley trace 0.0103 2.0000 17086.0000 87.9034 0.0000
      Roy's greatest root 0.0103 2.0000 17086.0000 87.9034 0.0000
=================================================================
```

No significant difference of gender on anxiety level or depression.

Form Trends

```
# visualizing form trend
df.groupby('Form')[['PHQ_Totals',
'GAD_Totals']].mean().plot(kind='line', marker='o')
plt.title('Mental Health Scores by School Form')
plt.xlabel('Form (Grade Level)')
plt.ylabel('Mean Score')
plt.savefig('Image/Mental Health Scores by Form.png', dpi=300,
bbox_inches='tight')
```

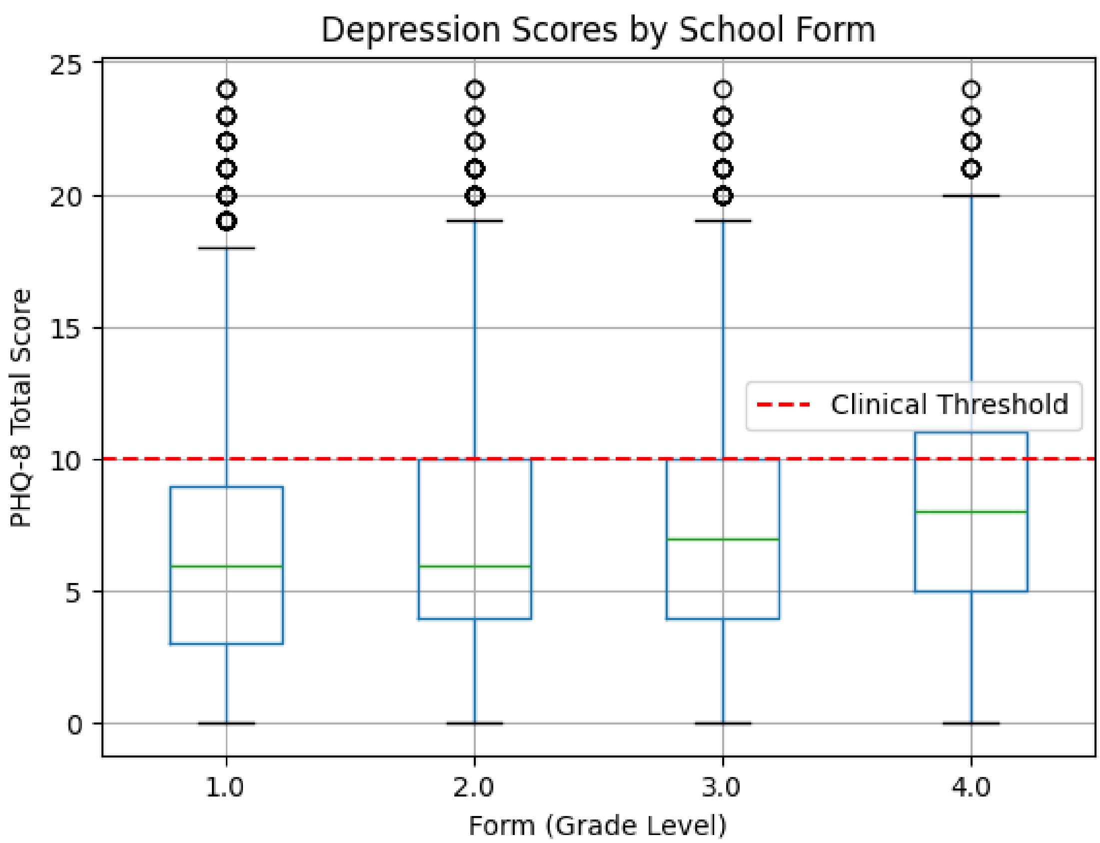



```python
from scipy import stats
from scipy.stats import kruskal

# form grade level effects
print("School form effect on depression")
# Prepare data by Form
form_groups_phq = [df[df['Form'] == i]['PHQ_Totals'].dropna() for i in
[1.0, 2.0, 3.0, 4.0]]

print(f"\nSample sizes by Form:")
for i, grp in enumerate(form_groups_phq, 1):
    print(f"Form {i}: n = {len(grp)}, Mean = {grp.mean():.2f}, SD =
{grp.std():.2f}")

# Check if we have data for all groups
form_groups_phq = [g for g in form_groups_phq if len(g) > 0]

if len(form_groups_phq) < 2:
    print("Not enough groups with data for comparison")
else:
    # Test for equal variances (Levene's test)
    _, p_levene = stats.levene(*form_groups_phq)
    print(f"\nLevene's test for equal variances: p = {p_levene:.4f}")
```

```
    if p_levene < 0.05:
        print("Variances are NOT equal. Using Kruskal-Wallis test (non-parametric).")

        # Kruskal-Wallis H-test (non-parametric ANOVA)
        h_stat, p_value = kruskal(*form_groups_phq)
        test_name = "Kruskal-Wallis H-test"

        print(f"{test_name} Results:")
        print(f"H-statistic: {h_stat:.3f}")
        print(f"p-value: {p_value:.4f}")

    else:
        print("Variances are equal. Using one-way ANOVA.")

        # One-way ANOVA
        f_stat, p_value = stats.f_oneway(*form_groups_phq)
        test_name = "One-way ANOVA"

        print(f"{test_name} Results:")
        print(f"F-statistic: {f_stat:.3f}")
        print(f"p-value: {p_value:.4f}")

    # Interpretation
    print(f"\nInterpretation:")
    if p_value < 0.001:
        print(f"HIGHLY SIGNIFICANT (p < 0.001)")
    elif p_value < 0.01:
        print(f"SIGNIFICANT (p < 0.01)")
    elif p_value < 0.05:
        print(f"SIGNIFICANT (p < 0.05)")
    else:
        print(f"NOT SIGNIFICANT (p ≥ 0.05)")

    if p_value < 0.05:
        print(f"There Are significant differences in depression scores across school forms.")
    else:
        print(f"No significant differences in depression scores across school forms.")

    # Visualization
    plt.figure(figsize=(10, 6))
    df.boxplot(column='PHQ_Totals', by='Form')
    plt.title('Depression Scores by School Form')
    plt.suptitle('')
    plt.xlabel('Form (Grade Level)')
    plt.ylabel('PHQ-8 Total Score')
```

```
    plt.axhline(y=10, color='red', linestyle='--', label='Clinical
Threshold')
    plt.legend()
    plt.show()
```

```
School form effect on depression

Sample sizes by Form:
Form 1: n = 6202, Mean = 6.55, SD = 4.20
Form 2: n = 5851, Mean = 7.09, SD = 4.20
Form 3: n = 3158, Mean = 7.37, SD = 4.34
Form 4: n = 1878, Mean = 8.07, SD = 4.44

Levene's test for equal variances: p = 0.0050
Variances are NOT equal. Using Kruskal-Wallis test (non-parametric).
Kruskal-Wallis H-test Results:
H-statistic: 212.976
p-value: 0.0000

Interpretation:
HIGHLY SIGNIFICANT (p < 0.001)
There Are significant differences in depression scores across school
forms.

<Figure size 1000x600 with 0 Axes>
```



Depression and anxiety levels seem to increase with increase in form. Students in form one experience anxiety and depression slightly as compared to those in form 4. In the next cell we will try to see whether its a significant change or not.

Family structure and Parental loss

```
from scipy.stats import mannwhitneyu

# Function to test impact on depression
def test_parental_factor(df, factor_col, target_col='PHQ_Totals'):
    print(f"Impact of {factor_col} on Depression ({target_col})")

    # Split groups (assuming 0 = parents present, >0 = loss/absence)
    group_present = df[df[factor_col] == 0.0][target_col].dropna()
    group_absent = df[df[factor_col] > 0.0][target_col].dropna()

    print(f"Sample sizes:")
    print(f"Present: n = {len(group_present)}, Mean =
{group_present.mean():.2f}")
    print(f"Absent/Loss: n = {len(group_absent)}, Mean =
{group_absent.mean():.2f}")

    if len(group_present) > 0 and len(group_absent) > 0:
        # Normality test
```

```python
        _, p1 = stats.shapiro(group_present.sample(min(5000,
len(group_present)), random_state=42))
        _, p2 = stats.shapiro(group_absent.sample(min(5000,
len(group_absent)), random_state=42))

        if p1 < 0.05 or p2 < 0.05:
            print("Using Mann-Whitney U test (non-parametric).")
            u_stat, p_val = mannwhitneyu(group_absent, group_present,
alternative='greater')
            print(f"U = {u_stat:.2f}, p = {p_val:.4f}")
        else:
            print("Using independent t-test (parametric).")
            t_stat, p_val = stats.ttest_ind(group_absent,
group_present, alternative='greater')
            print(f"t = {t_stat:.3f}, p = {p_val:.4f}")

        if p_val < 0.05:
            print("SIGNIFICANT: Absence/loss linked to higher
depression.")
        else:
            print("NOT SIGNIFICANT: No significant difference.")
    else:
        print("Insufficient data in one or both groups.")

# Run for both variables
test_parental_factor(df, 'Parents_Dead')
test_parental_factor(df, 'Parents_Home')
```

```
Impact of Parents_Dead on Depression (PHQ_Totals)
Sample sizes:
Present: n = 0, Mean = nan
Absent/Loss: n = 17089, Mean = 7.05
Insufficient data in one or both groups.
Impact of Parents_Home on Depression (PHQ_Totals)
Sample sizes:
Present: n = 471, Mean = 8.38
Absent/Loss: n = 16618, Mean = 7.02
Using Mann-Whitney U test (non-parametric).
U = 3235486.00, p = 1.0000
NOT SIGNIFICANT: No significant difference.
```

3.6.3 Multivariate Analysis

Geographic Patterns

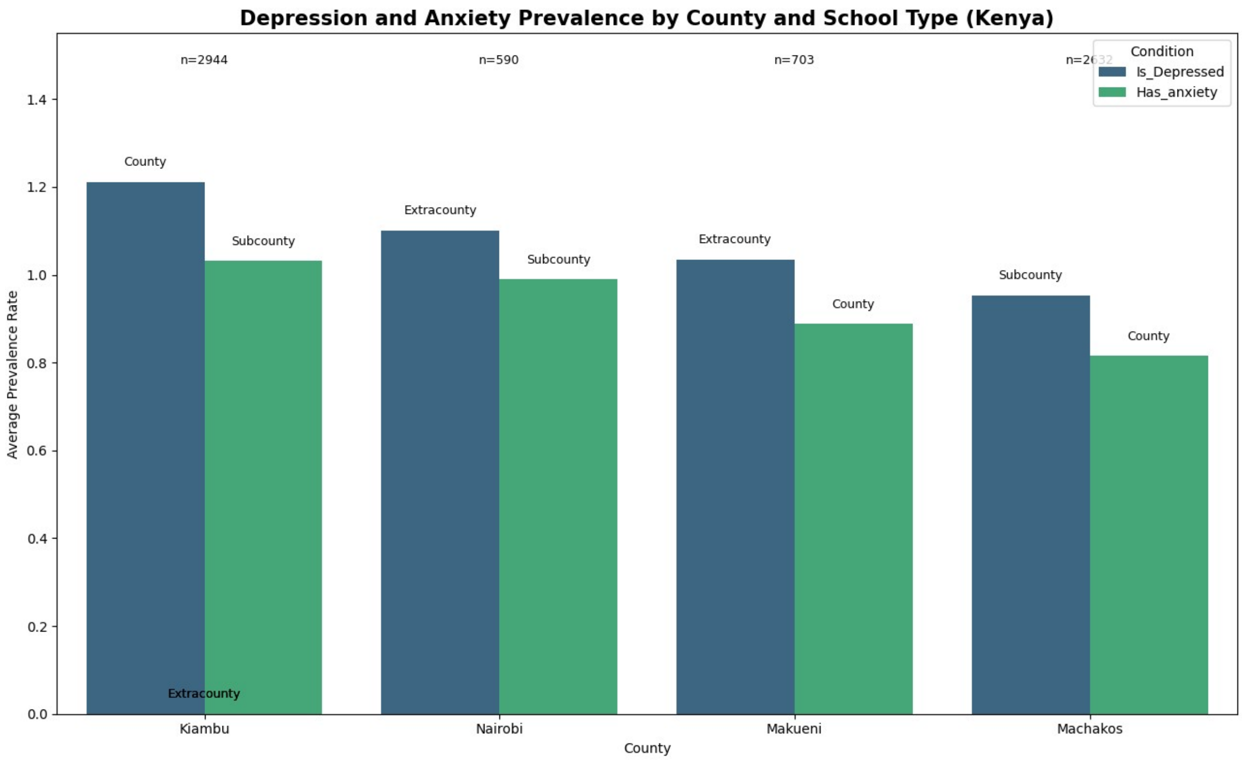
```python
# County-level prevalence
county_stats = df.groupby(['School_County','School_type']).agg({
    'Is_Depressed': 'mean',
    'Has_anxiety': 'mean',
```

```
    'participant_ID': 'count'
}).rename(columns={'participant_ID': 'n_students'})

# Filter counties with sufficient sample size
county_stats[county_stats['n_students'] >=
100].sort_values('Is_Depressed', ascending=False)
```

School_County	School_type	Is_Depressed	Has_anxiety	n_students
Kiambu	County	1.334190	1.149100	389
Nairobi	Extracounty	1.182119	1.086093	302
Makueni	Extracounty	1.177612	0.985075	670
Kiambu	Subcounty	1.088380	0.914166	5499
Nairobi	Subcounty	1.021161	0.892987	4962
	County	0.986278	0.867925	583
Machakos	Subcounty	0.952542	0.815254	590
	Extracounty	0.925790	0.788391	1361
	County	0.919052	0.848963	1013
Makueni	County	0.891452	0.791045	737
	Subcounty	0.781282	0.670397	983

```
data = {
    'School_County': ['Kiambu', 'Nairobi', 'Makueni', 'Machakos',
                      'Kiambu', 'Nairobi', 'Makueni'],
    'School_type': ['County', 'Extracounty', 'Extracounty',
'Subcounty',
                    'Subcounty', 'Subcounty', 'County'],
    'Is_Depressed': [1.334190, 1.182119, 1.177612, 0.952542, 1.088380,
1.021161, 0.891452],
    'Has_anxiety': [1.149100, 1.086093, 0.985075, 0.815254, 0.914166,
0.892987, 0.791045],
    'n_students': [389, 302, 670, 590, 5499, 4962, 737]
}

df_1 = pd.DataFrame(data)
plot_df = df_1.melt(
    id_vars=['School_County', 'School_type', 'n_students'],
    value_vars=['Is_Depressed', 'Has_anxiety'],
    var_name='Condition',
    value_name='Prevalence'
)

plt.figure(figsize=(13, 8))
ax = sns.barplot(
    data=plot_df,
    x='School_County',
    y='Prevalence',
    hue='Condition',
    palette='viridis',
    errorbar=None
```

```
)

for bar, (_, row) in zip(ax.patches, plot_df.iterrows()):
    height = bar.get_height()
    x = bar.get_x() + bar.get_width() / 2
    school_type = row['School_type']
    ax.text(
        x, height + 0.03,
        school_type,
        ha='center', va='bottom',
        fontsize=9, color='black', rotation=0
    )

county_means = df_1.groupby('School_County', as_index=False)
['n_students'].mean()
for i, row in enumerate(county_means.itertuples()):
    plt.text(i, 1.48, f"n={int(row.n_students)}", ha='center',
fontsize=9, color='black')
plt.title('Depression and Anxiety Prevalence by County and School Type
(Kenya)',
          fontsize=15, weight='bold')
plt.ylabel('Average Prevalence Rate')
plt.xlabel('County')
plt.ylim(0, 1.55)
plt.legend(title='Condition', loc='upper right')
plt.tight_layout()
plt.savefig('Image/Depression_Anxiety_by_County_and_SchoolType.png',
dpi=300, bbox_inches='tight')
plt.show()
```



Students in urban and high-performing counties (like Kiambu and Nairobi) particularly those in county and extracounty schools exhibit higher levels of depression and anxiety. These patterns may reflect the psychological costs of academic intensity, competitive environments, and limited rest or family interaction typical of such institutions. Meanwhile, rural counties (like Makueni and The analysis reveals notable differences in depression and anxiety prevalence across counties and school types in Kenya. Kiambu and Nairobi record the highest levels of both depression and anxiety, particularly in County and Extracounty schools, suggesting that students in more competitive or urbanized environments may experience greater psychological strain due to academic pressure and limited emotional support. In contrast, Makueni and Machakos show lower prevalence rates, possibly reflecting the protective influence of rural or community-based support systems. Overall, County and Extracounty schools appear more affected than Subcounty schools, indicating that school type and the associated boarding conditions, expectations, and competitiveness may play a significant role in shaping students' mental well-being. Machakos) report lower prevalence, suggesting that school environment and local context are key determinants of student wellbeing.

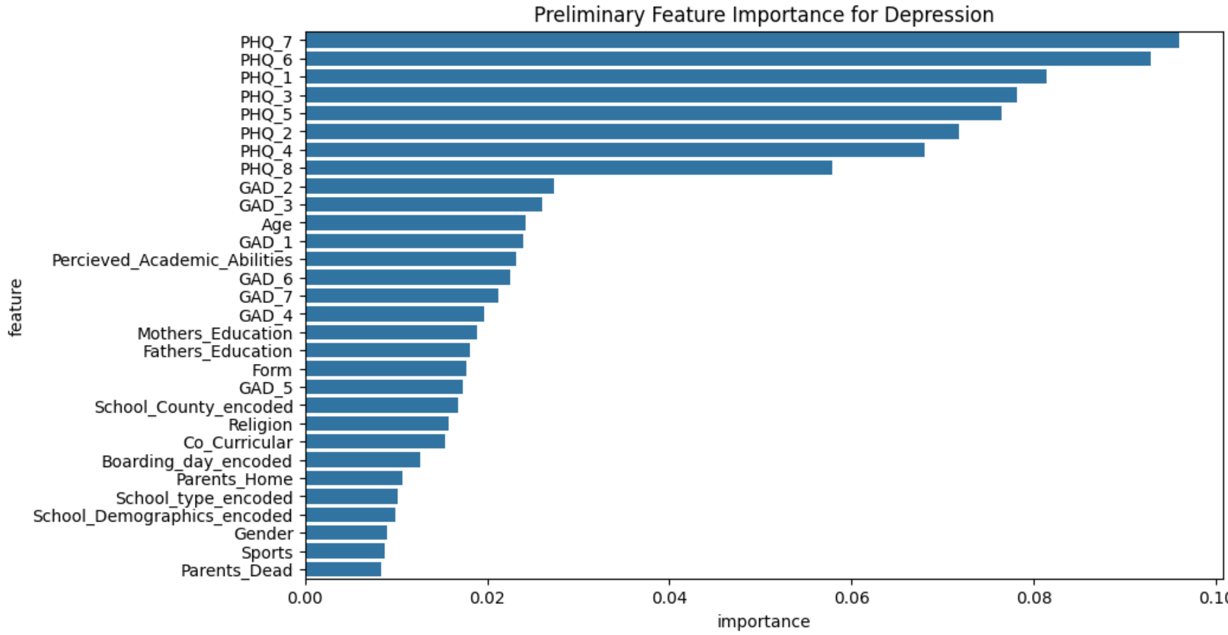
3.7 Feature Engineering

Feature Importance

```
from sklearn.preprocessing import LabelEncoder
import joblib
import os

object_cols = df.select_dtypes(include=['object']).columns.tolist()
id_keywords = ['participant_ID']
```

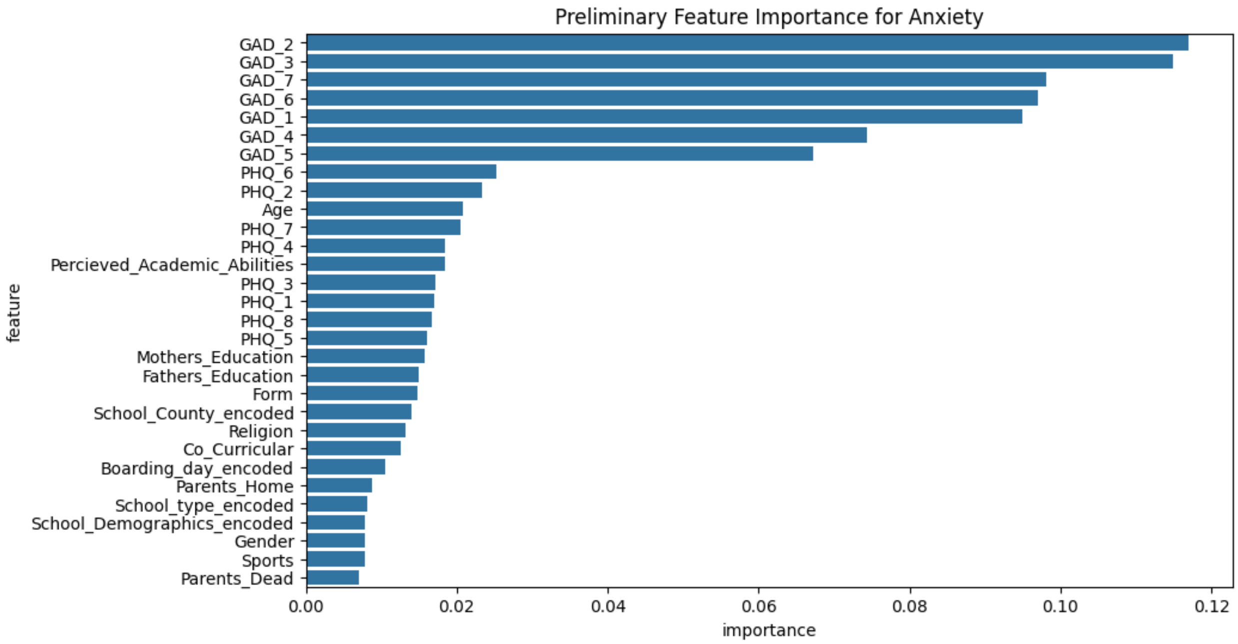
```
cols_to_encode = [
    col for col in object_cols
    if not any(kw.lower() in col.lower() for kw in id_keywords)
]

encoders = {}

for col in cols_to_encode:
    le = LabelEncoder()
    df[f'{col}_encoded'] = le.fit_transform(df[col].astype(str))+1
    encoders[col] = le



# Show mapping
mapping = pd.DataFrame({
    'code': range(len(le.classes_)),
    'original_value': le.classes_
})
encoded_cols = [f'{c}_encoded' for c in cols_to_encode]

from sklearn.ensemble import RandomForestClassifier
numerical_cols = ["Age", "Gender", "Form", "Religion",
    "Parents_Home", "Parents_Dead", "Fathers_Education",
"Mothers_Education",
    "Co_Curricular", "Sports", "Percieved_Academic_Abilities",
    "PHQ_1", "PHQ_2", "PHQ_3", "PHQ_4", "PHQ_5", "PHQ_6", "PHQ_7",
"PHQ_8",
    "GAD_1", "GAD_2", "GAD_3", "GAD_4", "GAD_5", "GAD_6", "GAD_7"]
# Quick feature importance
X = df[numerical_cols + encoded_cols]
y = df['Is_Depressed']

# Handle any remaining missing values
X = X.fillna(X.median())

# Train simple RF model
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X, y)

# Plot importance
importance_df = pd.DataFrame({
    'feature': X.columns,
    'importance': rf.feature_importances_
}).sort_values('importance', ascending=False)

plt.figure(figsize=(10,6))
sns.barplot(data=importance_df, x='importance', y='feature')
plt.title('Preliminary Feature Importance for Depression')
```

```
plt.savefig('Image/Preliminary Feature Importance for Depression.png',
dpi=300, bbox_inches='tight')
```




```
from sklearn.ensemble import RandomForestClassifier

numerical_cols = ["Age", "Gender", "Form", "Religion",
    "Parents_Home", "Parents_Dead", "Fathers_Education",
"Mothers_Education",
    "Co_Curricular", "Sports", "Percieved_Academic_Abilities",
    "PHQ_1", "PHQ_2", "PHQ_3", "PHQ_4", "PHQ_5", "PHQ_6", "PHQ_7",
"PHQ_8",
    "GAD_1", "GAD_2", "GAD_3", "GAD_4", "GAD_5", "GAD_6", "GAD_7"]
# feature importance
X = df[numerical_cols + encoded_cols]
y = df['Has_anxiety']

# Handle any remaining missing values
X = X.fillna(X.median())

# Train simple RF model
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X, y)

# Plot importance
importance_df = pd.DataFrame({
    'feature': X.columns,
    'importance': rf.feature_importances_
}).sort_values('importance', ascending=False)
```

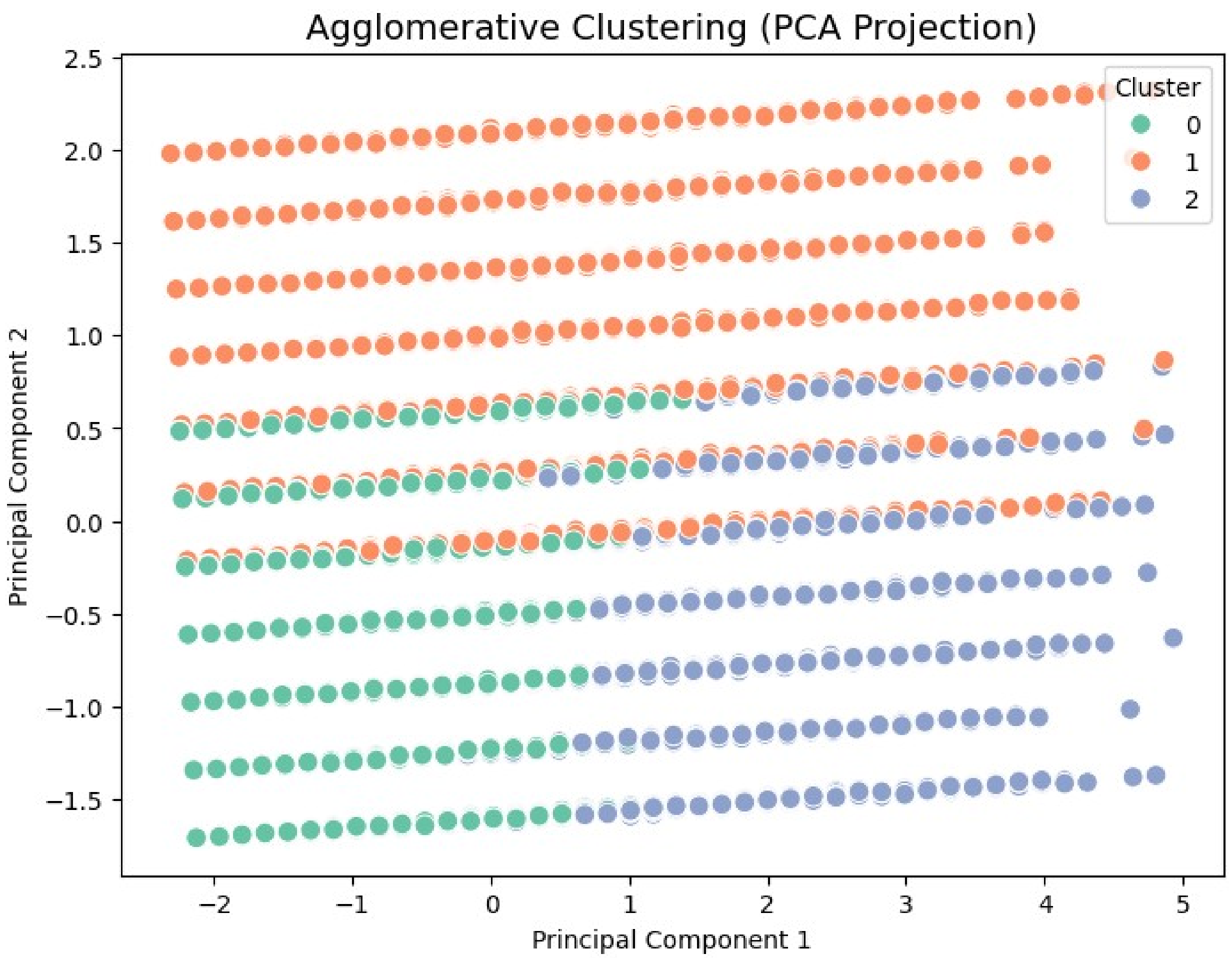
```
plt.figure(figsize=(10,6))
sns.barplot(data=importance_df, x='importance', y='feature')
plt.title('Preliminary Feature Importance for Anxiety')
plt.savefig('Image/Preliminary Feature Importance for Anxiety.png',
dpi=300, bbox_inches='tight')
```




```
# Average parental education
df['Parental_Education_Avg'] = df[['Fathers_Education',
'Mothers_Education']].mean(axis=1)

# PHQ-GAD interaction ratio
df['PHQ_GAD_Ratio'] = df['PHQ_Totals'] / (df['GAD_Totals'] + 1)

# Engagement/Resilience composite feature
df['Engagement_Score'] = df[['Sports', 'Co_Curricular',
'Percieved_Academic_Abilities']].sum(axis=1)

from sklearn.preprocessing import StandardScaler

# selecting features for clustering
num_features = ['PHQ_Totals', 'GAD_Totals', 'Parental_Education_Avg',
'Sports']

# Scale the selected features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df[num_features])
```

- Scaled so that features with larger scales do not dominate the PCA process.

```python
from sklearn.cluster import AgglomerativeClustering

# Initialize and fit the model
agg = AgglomerativeClustering(
    n_clusters=3,
    linkage='ward',
    metric='euclidean'
)

df['Cluster'] = agg.fit_predict(X_scaled)

from sklearn.decomposition import PCA

# using 2D for visualization
pca = PCA(n_components=2)
pca_result = pca.fit_transform(X_scaled)

df['PC1'] = pca_result[:, 0]
df['PC2'] = pca_result[:, 1]

# visualize clusters
plt.figure(figsize=(8,6))
sns.scatterplot(data=df, x='PC1', y='PC2', hue='Cluster',
palette='Set2', s=70)
plt.title("Agglomerative Clustering (PCA Projection)", fontsize=14)
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.show()
```



Agglomerative clustering identified **three distinct participant groups** with similar characteristics.

- **Cluster 1 (orange)** mainly appears on the right side of the PCA plot, while **Clusters 0 (green)** and **2 (blue)** are concentrated on the left.
- This spatial separation indicates clear differences in feature patterns among the clusters.
- The algorithm effectively **groups individuals with similar mental health and background traits**, revealing meaningful structure within the dataset.

```
# Explained variance ratio
pca.explained_variance_ratio_

array([0.41398891, 0.26549009])
```

- PCA 1 explains about 41% of the total variance of our dataset.
- PCA 2 explains about 27% of the total variance.

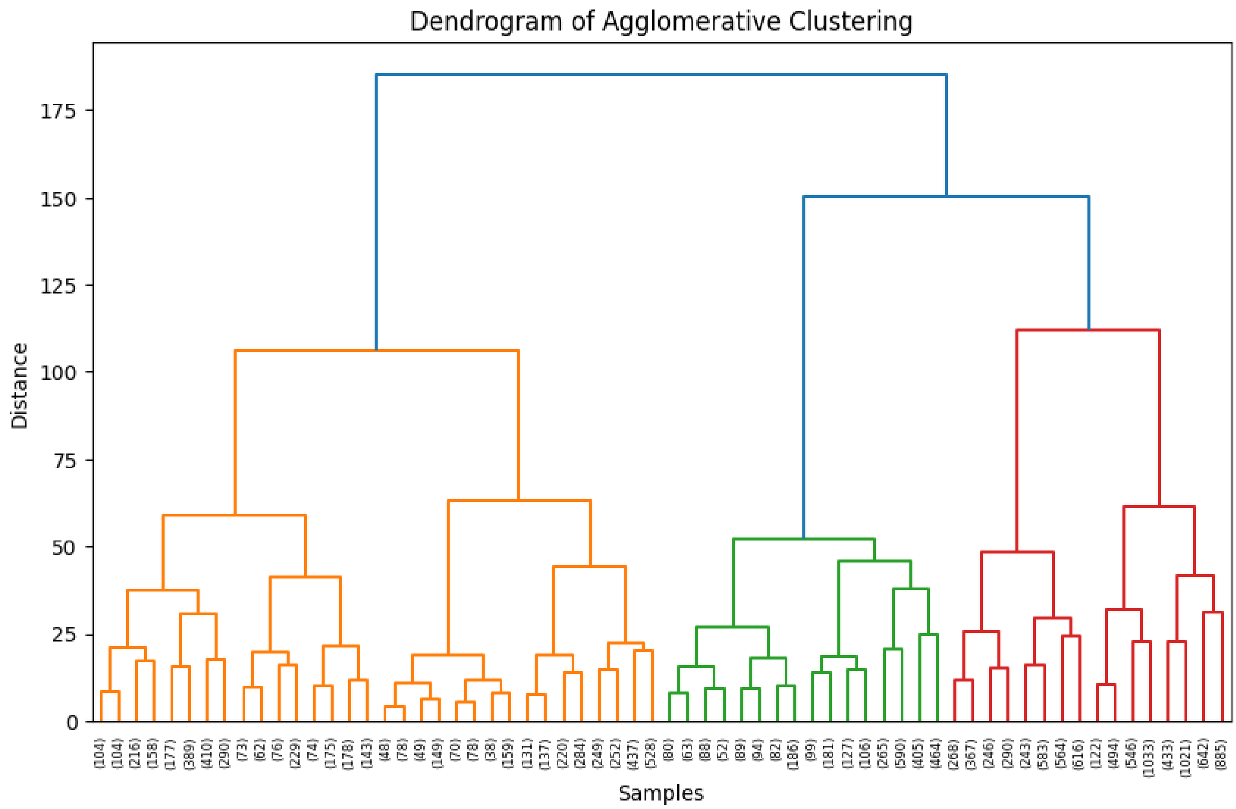
```
# Get the PCA loadings (components)
loadings = pca.components_
```

```python
# Create a DataFrame to visualize the loadings for each component
pca_loadings_df = pd.DataFrame(loadings.T, columns=['PCA 1', 'PCA 2'],
index=num_features)

# Display the loadings
print("PCA Loadings (contributions of each feature to PCA 1 and PCA
2):\n")
print(pca_loadings_df)
```

```
PCA Loadings (contributions of each feature to PCA 1 and PCA 2):

                          PCA 1     PCA 2
PHQ_Totals             0.706456  0.026539
GAD_Totals             0.706021  0.043051
Parental_Education_Avg -0.038786  0.704964
Sports                 -0.030817  0.707438
```

- PCA 1 is mainly driven by **PHQ_Totals** and **GAD_Totals**, representing a **mental distress dimension** (higher scores means more depression and anxiety).
- PCA 2 is dominated by **Parental_Education_Avg** and **Sports**, capturing a **socio-lifestyle dimension** linked to education and physical activity.

```python
from scipy.cluster.hierarchy import dendrogram, linkage

# hierarchical clustering for dendrogram
Z = linkage(X_scaled, method='ward')

plt.figure(figsize=(10, 6))
dendrogram(Z, truncate_mode='level', p=5)  # show last 5 merges
plt.title("Dendrogram of Agglomerative Clustering")
plt.xlabel("Samples")
plt.ylabel("Distance")
plt.show()
```



This **dendrogram** visualizes the hierarchical structure produced by **Agglomerative Clustering**. It shows how individual participants were merged step by step based on their similarity.

- The **bottom of the dendrogram** represents individual participants.
- The **vertical lines** show the distance between merged clusters.
- The **height where two branches join** indicates how similar those clusters are, lower heights mean higher similarity, while taller joins show greater differences.
- The **three main colored clusters (orange, green, red)** show how the algorithm divided participants into distinct groups.
- The **large blue branches near the top** represent the final merges before forming the main clusters, confirming that three clusters are a reasonable choice since there are three major branches before a large jump in distance occurs.

The dendrogram confirms that the data naturally forms **three main clusters** of participants with similar psychological and demographic profiles, supporting the cluster pattern observed in the PCA projection.

4. Modelling

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.multioutput import MultiOutputClassifier
```

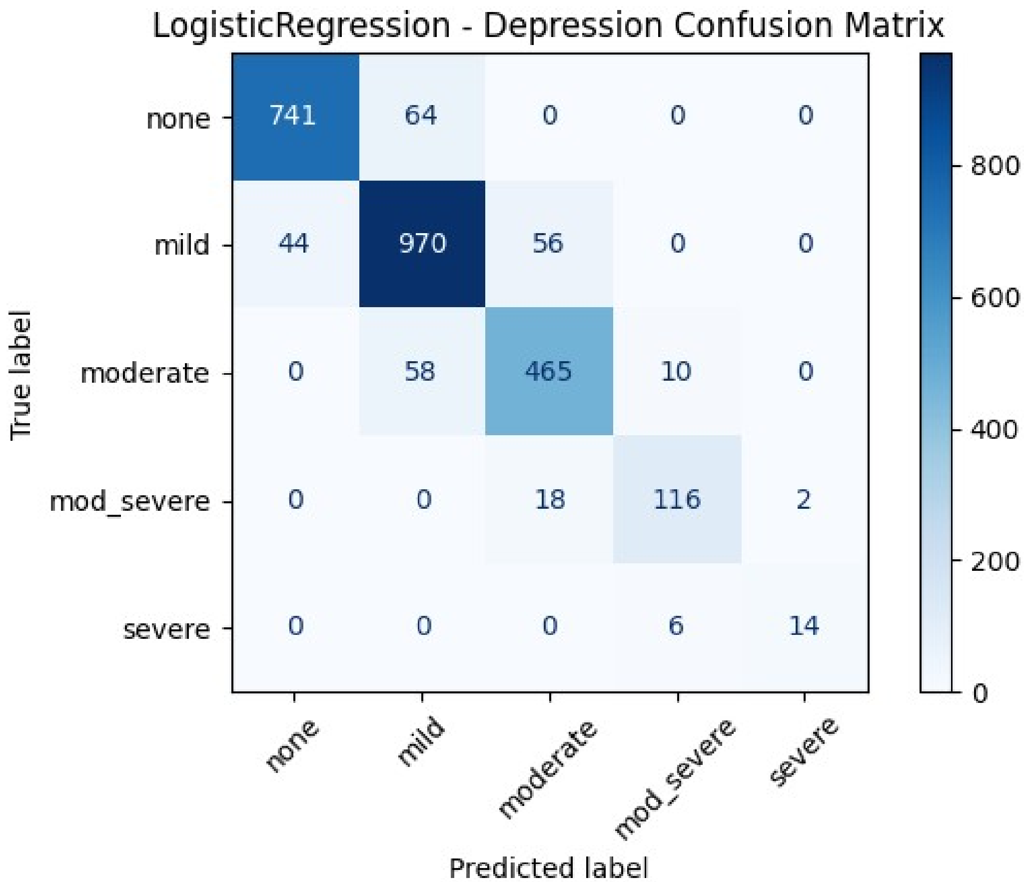
```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix, ConfusionMatrixDisplay
from sklearn.decomposition import PCA
import xgboost as xgb
from lightgbm import LGBMClassifier

X = df.drop(['Is_Depressed','Has_anxiety','PHQ_Totals',

'GAD_Totals','PHQ_Functioning','GAD_Check','GAD_Functioning'], axis =
1)
y = df[['Is_Depressed','Has_anxiety']]

from sklearn.model_selection import train_test_split

X_train, X_temp, y_train, y_temp = train_test_split(X, y,
test_size=0.3, random_state=42,stratify=y)

# Split into validation and test sets
# 70% train, 15% validation, 15% test split
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp,
test_size=0.5, random_state=42)

categorical_cols =
['Boarding_day','School_type','School_Demographics','School_County']
numerical_cols = ["Age", "Gender", "Form", "Religion",
    "Parents_Home", "Parents_Dead", "Fathers_Education",
"Mothers_Education",
    "Co_Curricular", "Sports", "Percieved_Academic_Abilities",
    "PHQ_1", "PHQ_2", "PHQ_3", "PHQ_4", "PHQ_5", "PHQ_6", "PHQ_7",
"PHQ_8",
    "GAD_1", "GAD_2", "GAD_3", "GAD_4", "GAD_5", "GAD_6", "GAD_7"]

# Preprocessing pipeline
pipeline_numeric = Pipeline(steps=[
    ('scaler', StandardScaler()),
    ('pca', PCA(n_components=0.8))
])

preprocessor = ColumnTransformer(
    transformers=[
        ('num', pipeline_numeric, numerical_cols),
        ('cat', OneHotEncoder(drop='first', handle_unknown='ignore'),
categorical_cols)
    ]
)

# models in a dictionary
```

```
models = {
    'LogisticRegression': LogisticRegression(random_state=42),
    'RandomForest': RandomForestClassifier(random_state=42),
    'XGBoost': xgb.XGBClassifier(random_state=42),
    'LightGBM': LGBMClassifier(force_col_wise=True, verbose=-
1,random_state=42)

}

results = {}

for name, base_model in models.items():
    print(f"Training {name}")

    # Create pipeline for each model
    pipeline = Pipeline(steps=[
        ('preprocessor', preprocessor),
        ('model', MultiOutputClassifier(base_model))
    ])

    # Train model
    pipeline.fit(X_train, y_train)

    # Predict on test data
    y_pred = pipeline.predict(X_test)

    # Compute per-target accuracy
    acc_dep = accuracy_score(y_test['Is_Depressed'], y_pred[:, 0])
    acc_anx = accuracy_score(y_test['Has_anxiety'], y_pred[:, 1])
    acc_avg = (acc_dep + acc_anx) / 2

    print(f"Depression Accuracy: {acc_dep:.4f}")
    print(f"Anxiety Accuracy:    {acc_anx:.4f}")
    print(f"Average Accuracy:    {acc_avg:.4f}")

    # Validation predictions
    val_pred = pipeline.predict(X_val)
    val_acc_dep = accuracy_score(y_val['Is_Depressed'], val_pred[:,
0])
    val_acc_anx = accuracy_score(y_val['Has_anxiety'], val_pred[:, 1])
    val_acc_avg = (val_acc_dep + val_acc_anx) / 2
    print(f"Validation Avg Accuracy: {val_acc_avg:.4f}")

    # Classification reports per target
    for i, target in enumerate(y_test.columns):
        print(f"\n{name} Classification Report for {target}:")
        print(classification_report(
            y_test[target],
            y_pred[:, i],
            digits=3
```

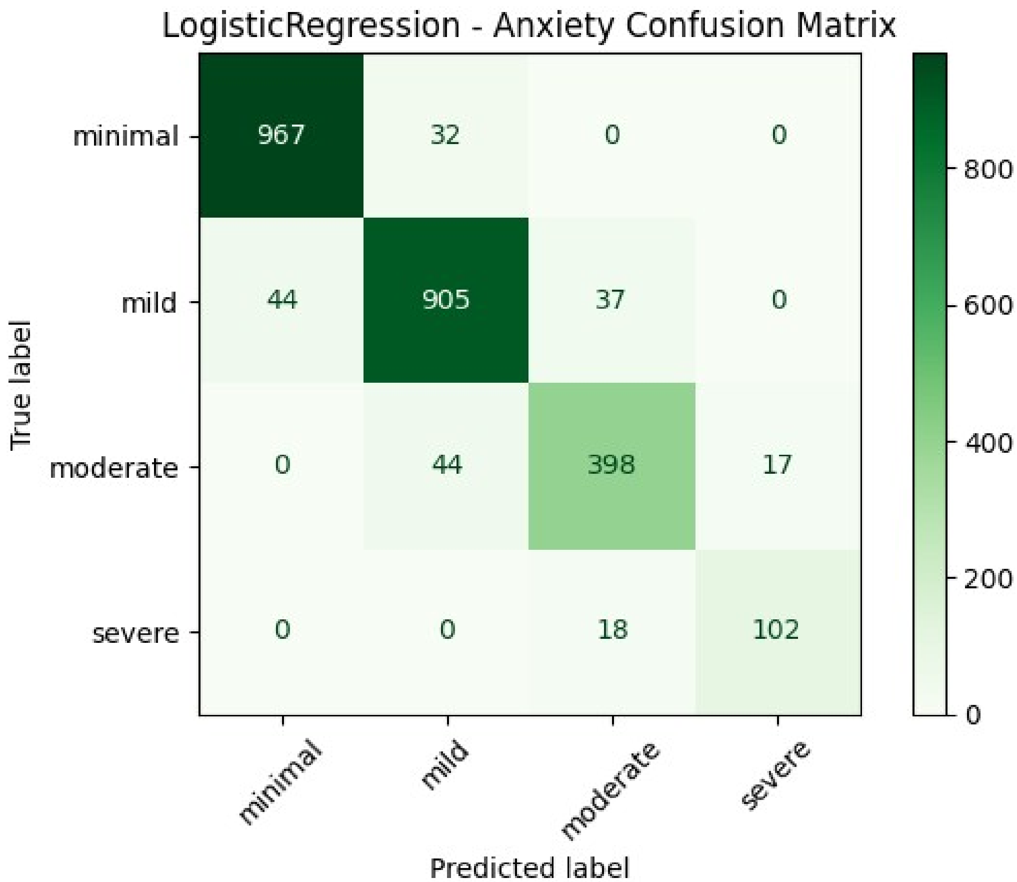
```
        ))

    # Confusion Matrix
    # Depression
    cm_dep = confusion_matrix(y_test['Is_Depressed'], y_pred[:, 0])
    ConfusionMatrixDisplay(
        cm_dep,
        display_labels=['none', 'mild', 'moderate', 'mod_severe',
'severe']
    ).plot(cmap='Blues', xticks_rotation=45)
    plt.title(f'{name} - Depression Confusion Matrix')
    plt.tight_layout()
    plt.show()

    # Anxiety
    cm_anx = confusion_matrix(y_test['Has_anxiety'], y_pred[:, 1])
    ConfusionMatrixDisplay(
        cm_anx,
        display_labels=['minimal', 'mild', 'moderate', 'severe']
    ).plot(cmap='Greens', xticks_rotation=45)
    plt.title(f'{name} - Anxiety Confusion Matrix')
    plt.tight_layout()
    plt.show()

    results[name] = {
        'depression_accuracy': acc_dep,
        'anxiety_accuracy': acc_anx,
        'average_accuracy': acc_avg
    }

print("Model Comparison:")
for name, res in results.items():
    print(f"{name}: Depression = {res['depression_accuracy']:.3f}, "
          f"Anxiety = {res['anxiety_accuracy']:.3f}, "
          f"Avg = {res['average_accuracy']:.3f}")
feature_names = preprocessor.get_feature_names_out()
print("Total transformed features:", len(feature_names))
print("Feature names:")
for i, name in enumerate(feature_names):
    print(f"{i:2d}: {name}")
import json
with open("expected_features.json", "w") as f:
    json.dump(list(feature_names), f, indent=2)
print("\nSaved to expected_features.json")
```

```
Training LogisticRegression
Depression Accuracy: 0.8994
Anxiety Accuracy:    0.9251
Average Accuracy:    0.9122
Validation Avg Accuracy: 0.9142
```

```
LogisticRegression Classification Report for Is_Depressed:
              precision    recall  f1-score   support

           0      0.944     0.920     0.932       805
           1      0.888     0.907     0.897      1070
           2      0.863     0.872     0.868       533
           3      0.879     0.853     0.866       136
           4      0.875     0.700     0.778        20

    accuracy                          0.899      2564
   macro avg      0.890     0.850     0.868      2564
weighted avg      0.900     0.899     0.899      2564


LogisticRegression Classification Report for Has_anxiety:
              precision    recall  f1-score   support

           0      0.956     0.968     0.962       999
           1      0.923     0.918     0.920       986
           2      0.879     0.867     0.873       459
           3      0.857     0.850     0.854       120

    accuracy                          0.925      2564
   macro avg      0.904     0.901     0.902      2564
weighted avg      0.925     0.925     0.925      2564
```

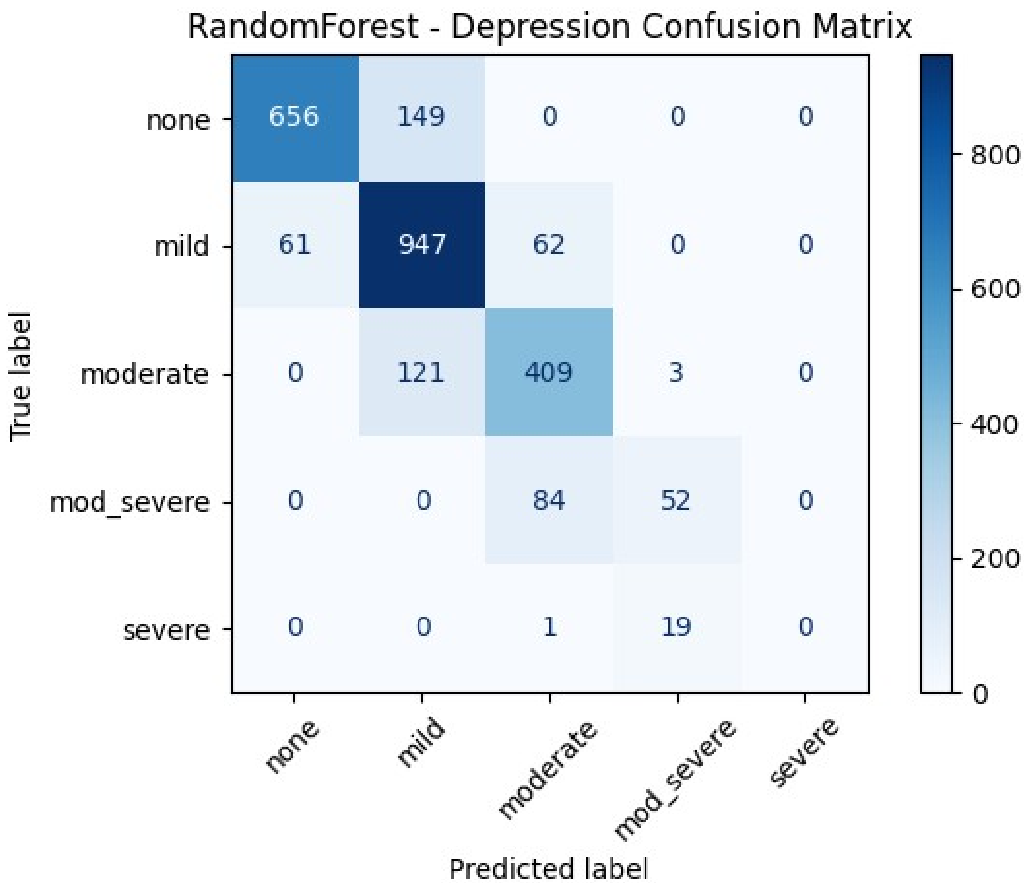


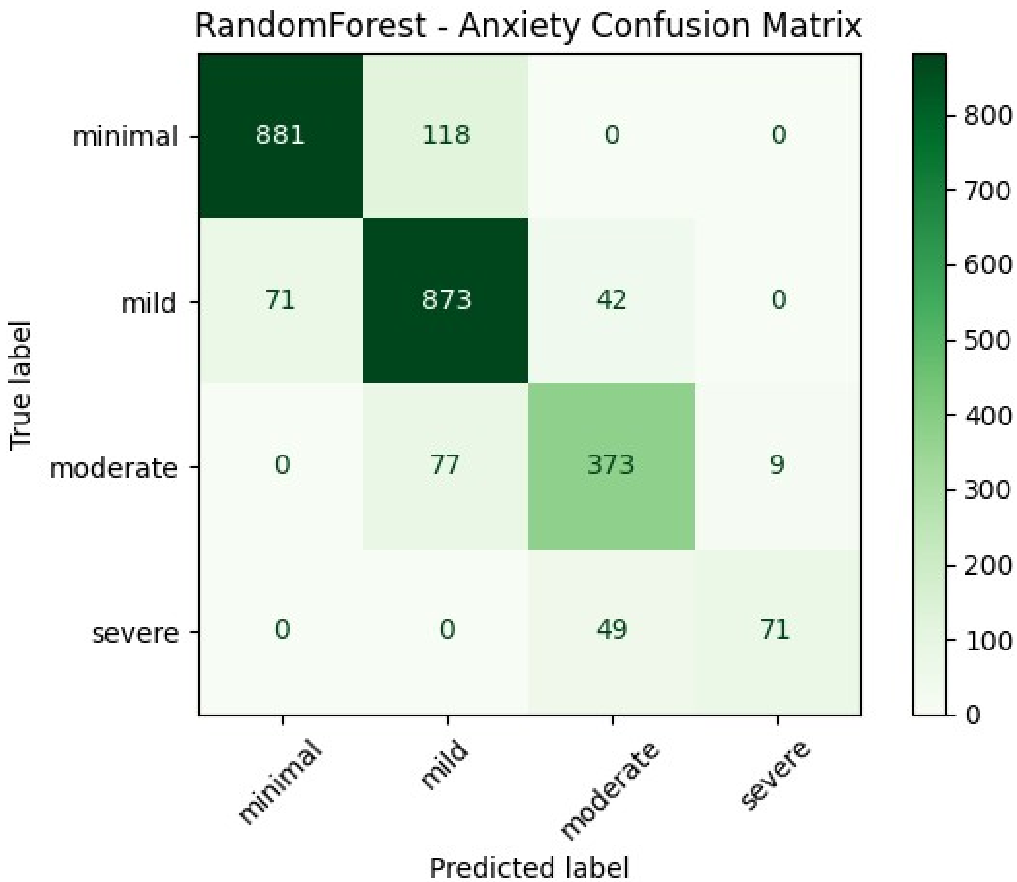
LogisticRegression - Depression Confusion Matrix
True label
Predicted label
none
mild
moderate
mod_severe
severe
741
64
0
44
970
56
58
465
10
18
116
2
6
14
800
600
400
200




```
Training RandomForest
Depression Accuracy: 0.8050
Anxiety Accuracy:    0.8573
Average Accuracy:    0.8311
Validation Avg Accuracy: 0.8342

RandomForest Classification Report for Is_Depressed:
              precision    recall  f1-score   support

           0      0.915     0.815     0.862       805
           1      0.778     0.885     0.828      1070
           2      0.736     0.767     0.751       533
           3      0.703     0.382     0.495       136
           4      0.000     0.000     0.000        20

    accuracy                          0.805      2564
   macro avg      0.626     0.570     0.587      2564
weighted avg      0.802     0.805     0.799      2564


RandomForest Classification Report for Has_anxiety:
              precision    recall  f1-score   support
```

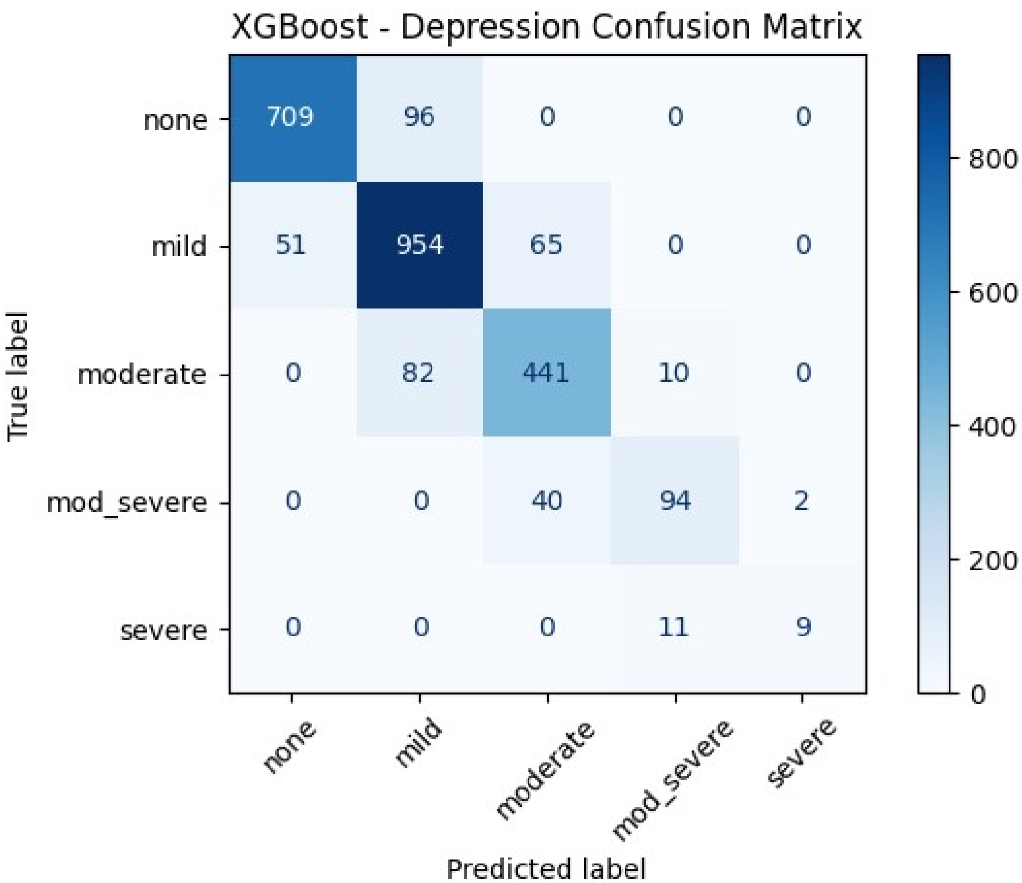
```
           0      0.925     0.882     0.903       999
           1      0.817     0.885     0.850       986
           2      0.804     0.813     0.808       459
           3      0.887     0.592     0.710       120

    accuracy                          0.857      2564
   macro avg      0.859     0.793     0.818      2564
weighted avg      0.860     0.857     0.857      2564
```

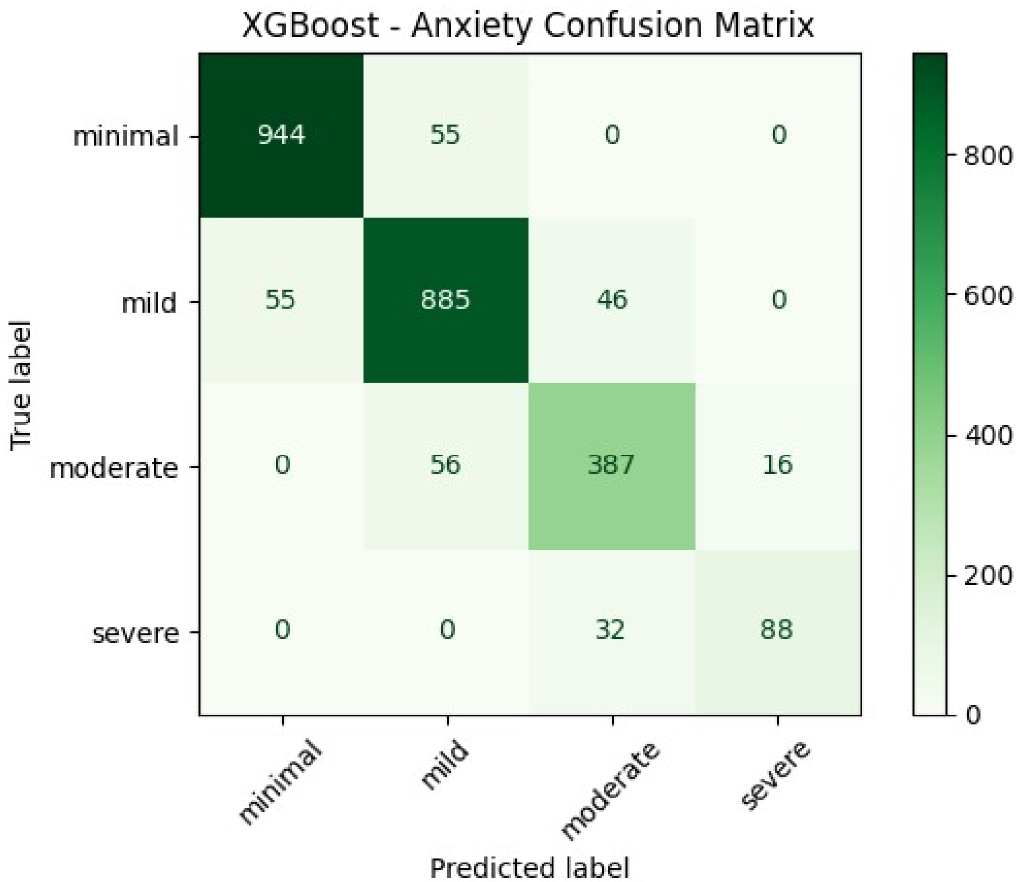





```
Training XGBoost
Depression Accuracy: 0.8608
Anxiety Accuracy:    0.8986
Average Accuracy:    0.8797
Validation Avg Accuracy: 0.8835

XGBoost Classification Report for Is_Depressed:
              precision    recall  f1-score   support

           0      0.933     0.881     0.906       805
           1      0.843     0.892     0.866      1070
           2      0.808     0.827     0.817       533
           3      0.817     0.691     0.749       136
           4      0.818     0.450     0.581        20

    accuracy                          0.861      2564
   macro avg      0.844     0.748     0.784      2564
weighted avg      0.862     0.861     0.860      2564


XGBoost Classification Report for Has_anxiety:
              precision    recall  f1-score   support
```

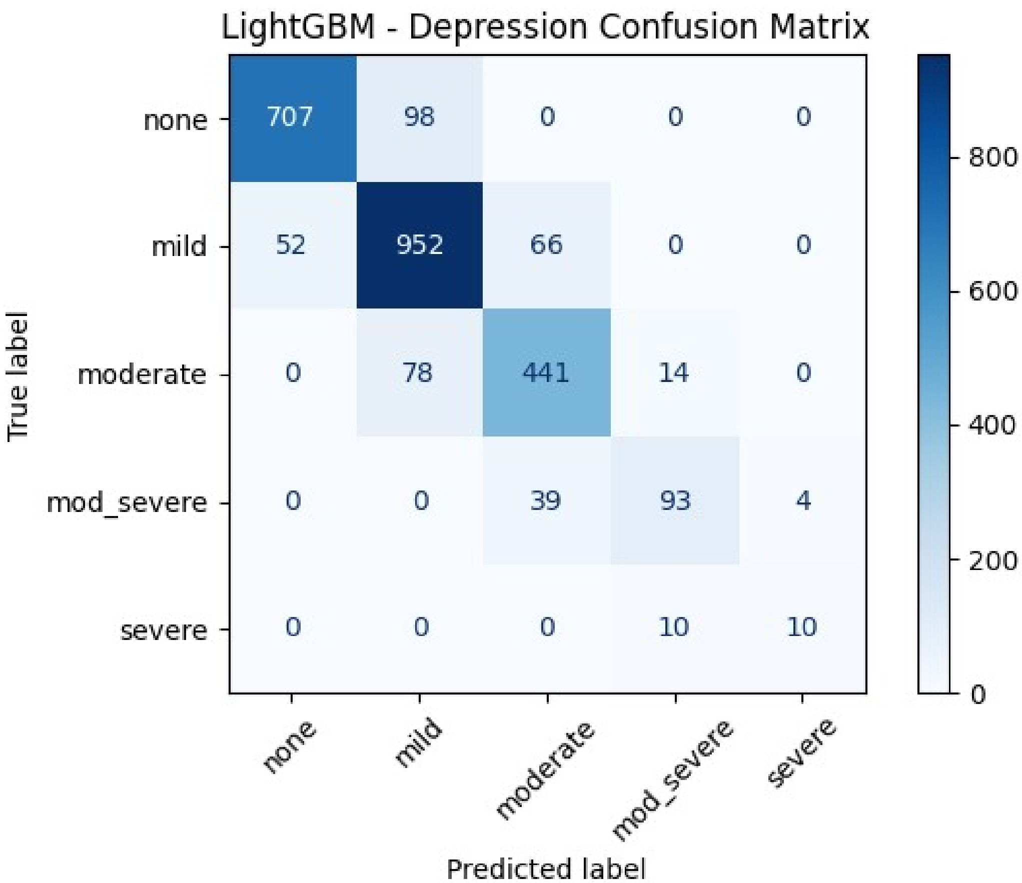
```
           0      0.945     0.945     0.945       999
           1      0.889     0.898     0.893       986
           2      0.832     0.843     0.838       459
           3      0.846     0.733     0.786       120

    accuracy                          0.899      2564
   macro avg      0.878     0.855     0.865      2564
weighted avg      0.898     0.899     0.898      2564
```

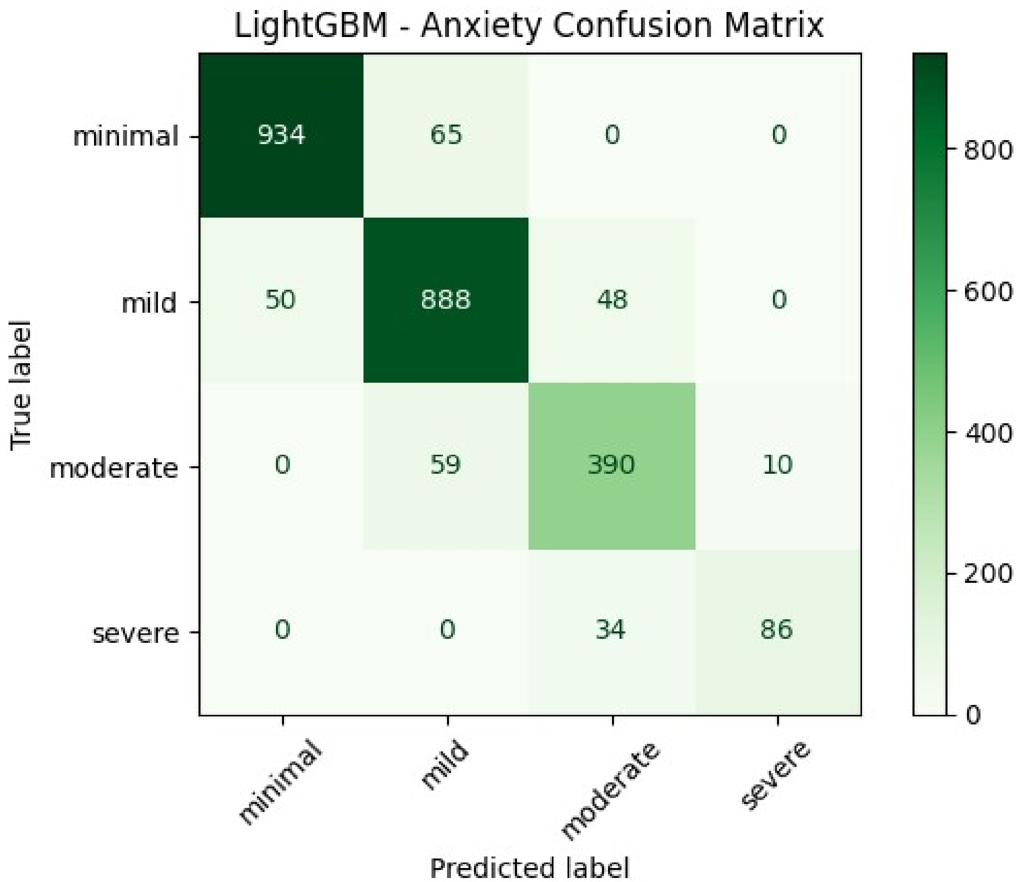





```
Training LightGBM
Depression Accuracy: 0.8592
Anxiety Accuracy:    0.8963
Average Accuracy:    0.8777
Validation Avg Accuracy: 0.8769

LightGBM Classification Report for Is_Depressed:
              precision    recall  f1-score   support

           0      0.931     0.878     0.904       805
           1      0.844     0.890     0.866      1070
           2      0.808     0.827     0.817       533
           3      0.795     0.684     0.735       136
           4      0.714     0.500     0.588        20

    accuracy                          0.859      2564
   macro avg      0.818     0.756     0.782      2564
weighted avg      0.860     0.859     0.859      2564


LightGBM Classification Report for Has_anxiety:
              precision    recall  f1-score   support
```

```
           0      0.949     0.935     0.942       999
           1      0.877     0.901     0.889       986
           2      0.826     0.850     0.838       459
           3      0.896     0.717     0.796       120

    accuracy                          0.896      2564
   macro avg      0.887     0.850     0.866      2564
weighted avg      0.897     0.896     0.896      2564
```






```
Model Comparison:
LogisticRegression: Depression = 0.899, Anxiety = 0.925, Avg = 0.912
RandomForest: Depression = 0.805, Anxiety = 0.857, Avg = 0.831
XGBoost: Depression = 0.861, Anxiety = 0.899, Avg = 0.880
LightGBM: Depression = 0.859, Anxiety = 0.896, Avg = 0.878
Total transformed features: 27
Feature names:
 0: num__pca0
 1: num__pca1
 2: num__pca2
 3: num__pca3
 4: num__pca4
 5: num__pca5
 6: num__pca6
 7: num__pca7
 8: num__pca8
 9: num__pca9
10: num__pca10
11: num__pca11
12: num__pca12
13: num__pca13
14: num__pca14
```

```
15: num__pca15
16: num__pca16
17: num__pca17
18: cat__Boarding_day_Day
19: cat__Boarding_day_Day & Boarding
20: cat__School_type_Extracounty
21: cat__School_type_Subcounty
22: cat__School_Demographics_Girls
23: cat__School_Demographics_Mixed
24: cat__School_County_Machakos
25: cat__School_County_Makueni
26: cat__School_County_Nairobi

Saved to expected_features.json
```

4.1 Fine-Tuning the Models

```python
import lightgbm as lgb
params = {
    'Logistic': {
        'model': LogisticRegression(random_state=42),
        'params': {
            'clf__estimator__C': [0.1, 1, 10],
            'clf__estimator__penalty': ['l2', 'l1'],

            'clf__estimator__solver': ['liblinear']
        }
    },

    'RandomForest': {
        'model': RandomForestClassifier(
            random_state=42,
            n_jobs=-1
        ),
        'params': {

            'clf__estimator__n_estimators': [100, 200],
            'clf__estimator__max_depth': [None, 10, 20],
            'clf__estimator__min_samples_split': [2, 5]
        }
    },

    'XGBoost': {
        'model': xgb.XGBClassifier(
            random_state=42,
            eval_metric='logloss',
            n_jobs=-1
        ),
        'params': {
```

```python
            'clf__estimator__n_estimators': [100, 200],
            'clf__estimator__max_depth': [3, 6],
            'clf__estimator__learning_rate': [0.1, 0.01]

        }
    },

    'LightGBM': {
        'model': lgb.LGBMClassifier(
            random_state=42,
            n_jobs=-1,
            verbose=-1
        ),
        'params': {
            'clf__estimator__n_estimators': [100],
            'clf__estimator__max_depth': [3],
            'clf__estimator__learning_rate': [0.1]
        }
    }
}

import pickle
import pandas as pd

with open("trained_pipelines.pkl", "rb") as f:
    pipelines = pickle.load(f)

# Pick any pipeline (e.g., best one)
pipe = list(pipelines.values())[0]

# Get feature names after preprocessing
feature_names =
pipe.named_steps['preprocessor'].get_feature_names_out()
print("Expected features:", len(feature_names))
print(feature_names)
```

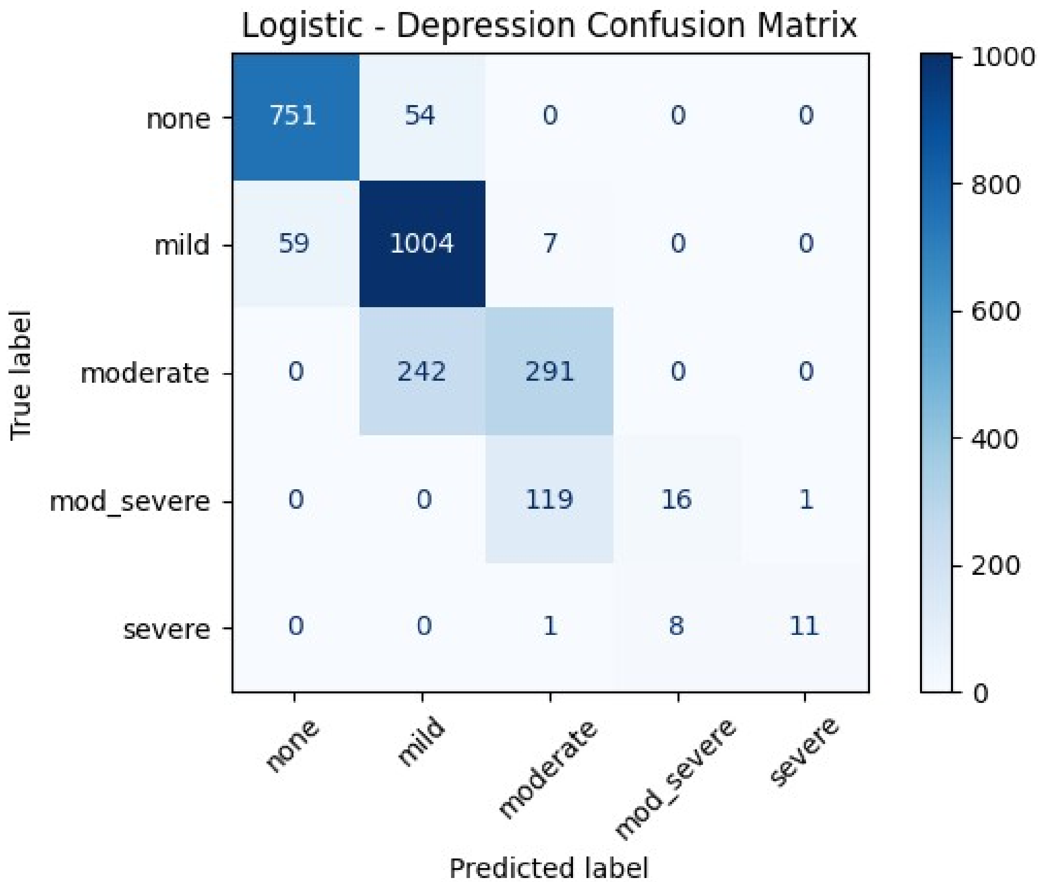
```
Expected features: 27
['num__pca0' 'num__pca1' 'num__pca2' 'num__pca3' 'num__pca4'
'num__pca5'
 'num__pca6' 'num__pca7' 'num__pca8' 'num__pca9' 'num__pca10'
'num__pca11'
 'num__pca12' 'num__pca13' 'num__pca14' 'num__pca15' 'num__pca16'
 'num__pca17' 'cat__Boarding_day_Day' 'cat__Boarding_day_Day &
Boarding'
 'cat__School_type_Extracounty' 'cat__School_type_Subcounty'
 'cat__School_Demographics_Girls' 'cat__School_Demographics_Mixed'
 'cat__School_County_Machakos' 'cat__School_County_Makueni'
 'cat__School_County_Nairobi']
```

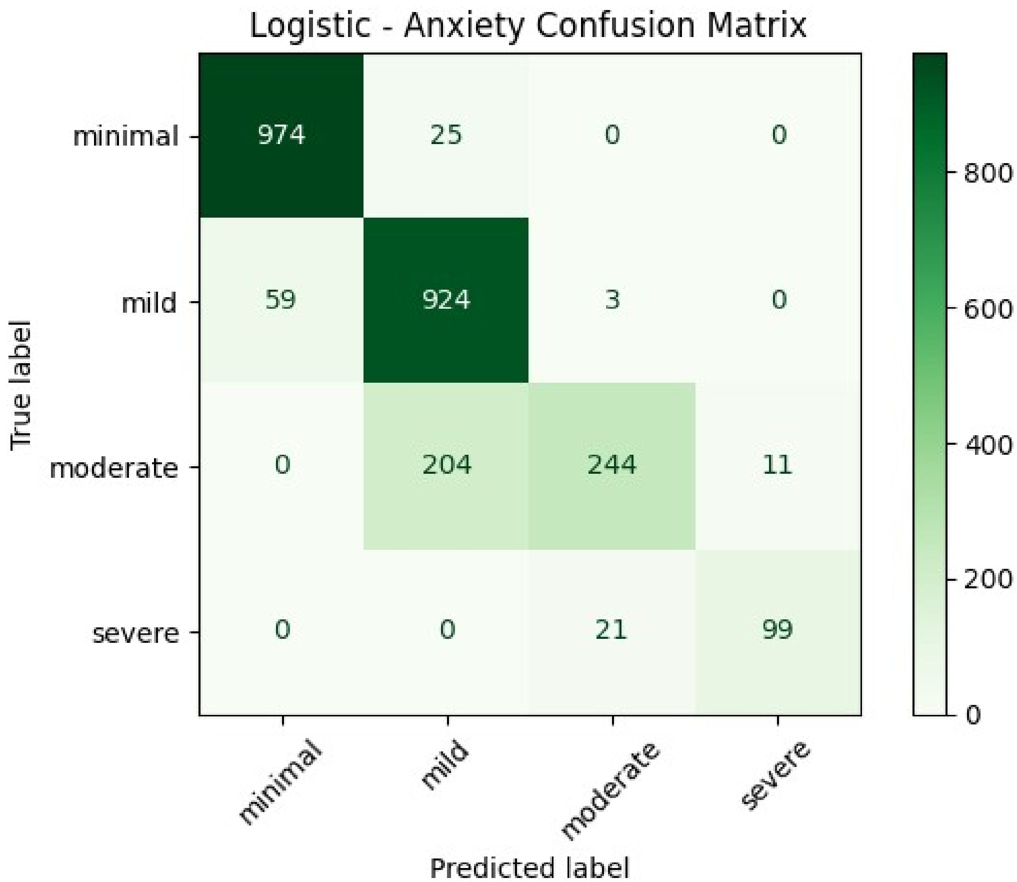
```python
import matplotlib.pyplot as plt
import seaborn as sns
import lightgbm as lgb
import joblib
from sklearn.metrics import accuracy_score, recall_score,
classification_report, confusion_matrix, ConfusionMatrixDisplay,
make_scorer
from sklearn.model_selection import GridSearchCV
from sklearn.multioutput import MultiOutputClassifier
from sklearn.pipeline import Pipeline
import numpy as np
import pickle

# Custom scorer for multi-output classification
def multioutput_accuracy(y_true, y_pred):
    #Calculate average accuracy across all outputs
    # Convert to numpy array if DataFrame
    if hasattr(y_true, 'values'):
        y_true = y_true.values
    if hasattr(y_pred, 'values'):
        y_pred = y_pred.values

    n_outputs = y_true.shape[1]
    accuracies = []
    for i in range(n_outputs):
        acc = accuracy_score(y_true[:, i], y_pred[:, i])
        accuracies.append(acc)
    return np.mean(accuracies)

# Create the scorer
multi_accuracy_scorer = make_scorer(multioutput_accuracy)

results = {}

# Loop over each model configuration
for name, config in params.items():
    print(f"\nTraining {name} with GridSearchCV")

    # pipeline
    pipeline = Pipeline(steps=[
        ('preprocessor', preprocessor),
        ('clf', MultiOutputClassifier(config['model']))
    ])

    # Grid search
    grid = GridSearchCV(
        pipeline,
        param_grid=config['params'],
        cv=3,
        scoring=multi_accuracy_scorer,
```

```
        n_jobs=-1,
        verbose=1
    )

    # Fit model
    grid.fit(X_train, y_train)

    print(f"Best parameters for {name}: {grid.best_params_}")
    print(f"Best CV accuracy for {name}: {grid.best_score_:.4f}")

    # Save best model
    best_model = grid.best_estimator_
    joblib.dump(best_model, f"{name}_model.pkl")

    # Predictions
    y_pred = best_model.predict(X_test)

    # Compute per-target recall
    recall_dict = {}
    for i, target in enumerate(y_test.columns):
        rec = recall_score(y_test[target], y_pred[:, i],
average='macro')
        recall_dict[target] = rec
        print(f"{target} Recall: {rec:.4f}")

    avg_recall = sum(recall_dict.values()) / len(recall_dict)
    print(f"Average Recall: {avg_recall:.4f}")

    # Compute per-target accuracy
    acc_dict = {}
    for i, target in enumerate(y_test.columns):
        acc = accuracy_score(y_test[target], y_pred[:, i])
        acc_dict[target] = acc
        print(f"{target} Accuracy: {acc:.4f}")

    avg_acc = sum(acc_dict.values()) / len(acc_dict)
    print(f"Average Accuracy: {avg_acc:.4f}")

    # Validation predictions
    val_pred = best_model.predict(X_val)
    val_acc_dict = {}
    for i, target in enumerate(y_val.columns):
        val_acc = accuracy_score(y_val[target], val_pred[:, i])
        val_acc_dict[target] = val_acc
    val_avg_acc = sum(val_acc_dict.values()) / len(val_acc_dict)
    print(f"Validation Average Accuracy: {val_avg_acc:.4f}")

    # Classification reports
    for i, target in enumerate(y_test.columns):
        print(f"\n{name} Classification Report for {target}:")
        print(classification_report(y_test[target], y_pred[:, i],
```

```
digits=3))

    # Confusion matrices
    cm_dep = confusion_matrix(y_test['Is_Depressed'], y_pred[:, 0])
    ConfusionMatrixDisplay(
        cm_dep,
        display_labels=['none', 'mild', 'moderate', 'mod_severe',
'severe']
    ).plot(cmap='Blues', xticks_rotation=45)
    plt.title(f'{name} - Depression Confusion Matrix')
    plt.tight_layout()
    plt.show()

    cm_anx = confusion_matrix(y_test['Has_anxiety'], y_pred[:, 1])
    ConfusionMatrixDisplay(
        cm_anx,
        display_labels=['minimal', 'mild', 'moderate', 'severe']
    ).plot(cmap='Greens', xticks_rotation=45)
    plt.title(f'{name} - Anxiety Confusion Matrix')
    plt.tight_layout()
    plt.show()

    # Store metrics
    results[name] = {
        'best_params': grid.best_params_,
        'cv_accuracy': grid.best_score_,
        'test_accuracy_per_target': acc_dict,  # This is the dict with
per-target accuracy
        'average_test_accuracy': avg_acc,
        'validation_accuracy_per_target': val_acc_dict,
        'average_validation_accuracy': val_avg_acc,
        'test_recall_per_target': recall_dict,  # This is the dict
with per-target recall
        'average_recall': avg_recall
    }
joblib.dump(results, "model_metrics.pkl")
print("\nModel metrics saved to model_metrics.pkl")

print("\nModel Comparison Summary:")
for name, res in results.items():
    print(f"\n{name}:")
    for target, acc in res['test_accuracy_per_target'].items():
        print(f" {target} Test Accuracy: {acc:.3f}")
    for target, rec in res['test_recall_per_target'].items():
        print(f" {target} Recall: {rec:.3f}")
    print(f"Average Test Accuracy:
{res['average_test_accuracy']:.3f}")
    print(f"Average Recall: {res['average_recall']:.3f}")
    print(f"Average Validation Accuracy:
{res['average_validation_accuracy']:.3f}")
```

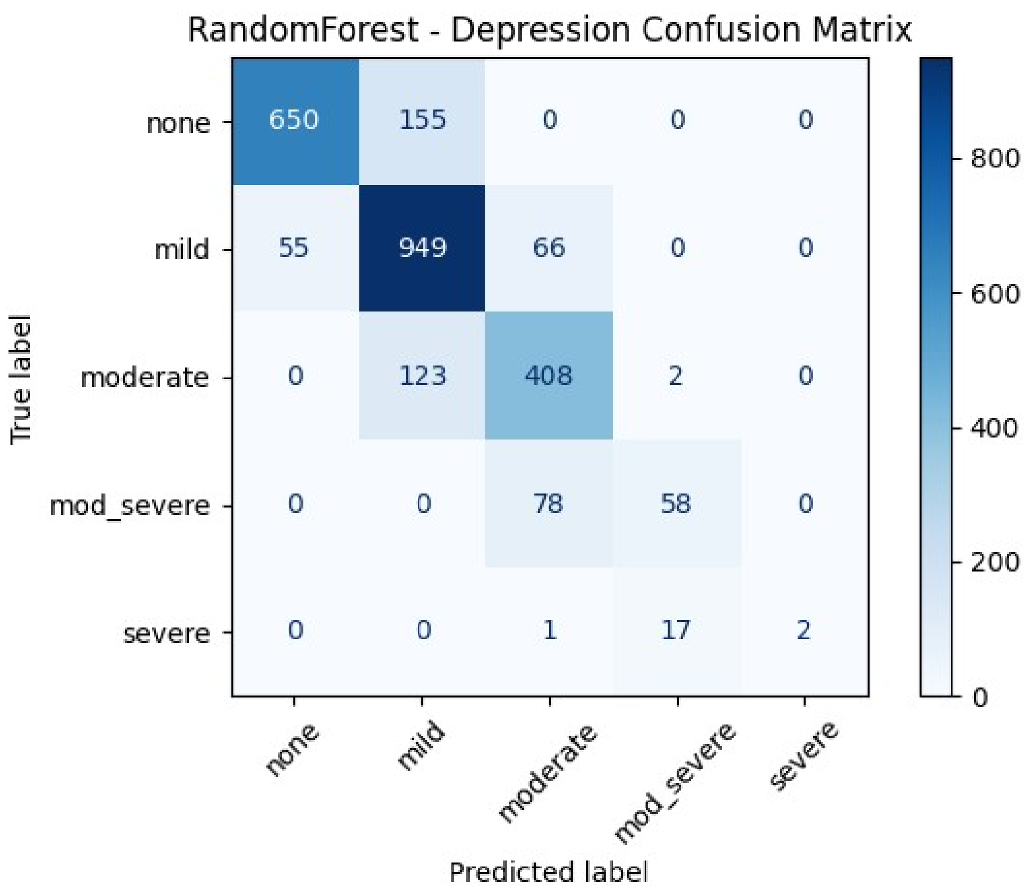
```
Training Logistic with GridSearchCV
Fitting 3 folds for each of 6 candidates, totalling 18 fits
Best parameters for Logistic: {'clf__estimator__C': 10,
'clf__estimator__penalty': 'l1', 'clf__estimator__solver':
'liblinear'}
Best CV accuracy for Logistic: 0.8401
Is_Depressed Recall: 0.6170
Has_anxiety Recall: 0.8172
Average Recall: 0.7171
Is_Depressed Accuracy: 0.8085
Has_anxiety Accuracy: 0.8740
Average Accuracy: 0.8413
Validation Average Accuracy: 0.8496

Logistic Classification Report for Is_Depressed:
              precision    recall  f1-score   support

           0      0.927     0.933     0.930       805
           1      0.772     0.938     0.847      1070
           2      0.696     0.546     0.612       533
           3      0.667     0.118     0.200       136
           4      0.917     0.550     0.688        20

    accuracy                          0.809      2564
   macro avg      0.796     0.617     0.655      2564
weighted avg      0.801     0.809     0.789      2564


Logistic Classification Report for Has_anxiety:
              precision    recall  f1-score   support

           0      0.943     0.975     0.959       999
           1      0.801     0.937     0.864       986
           2      0.910     0.532     0.671       459
           3      0.900     0.825     0.861       120

    accuracy                          0.874      2564
   macro avg      0.889     0.817     0.839      2564
weighted avg      0.881     0.874     0.866      2564
```

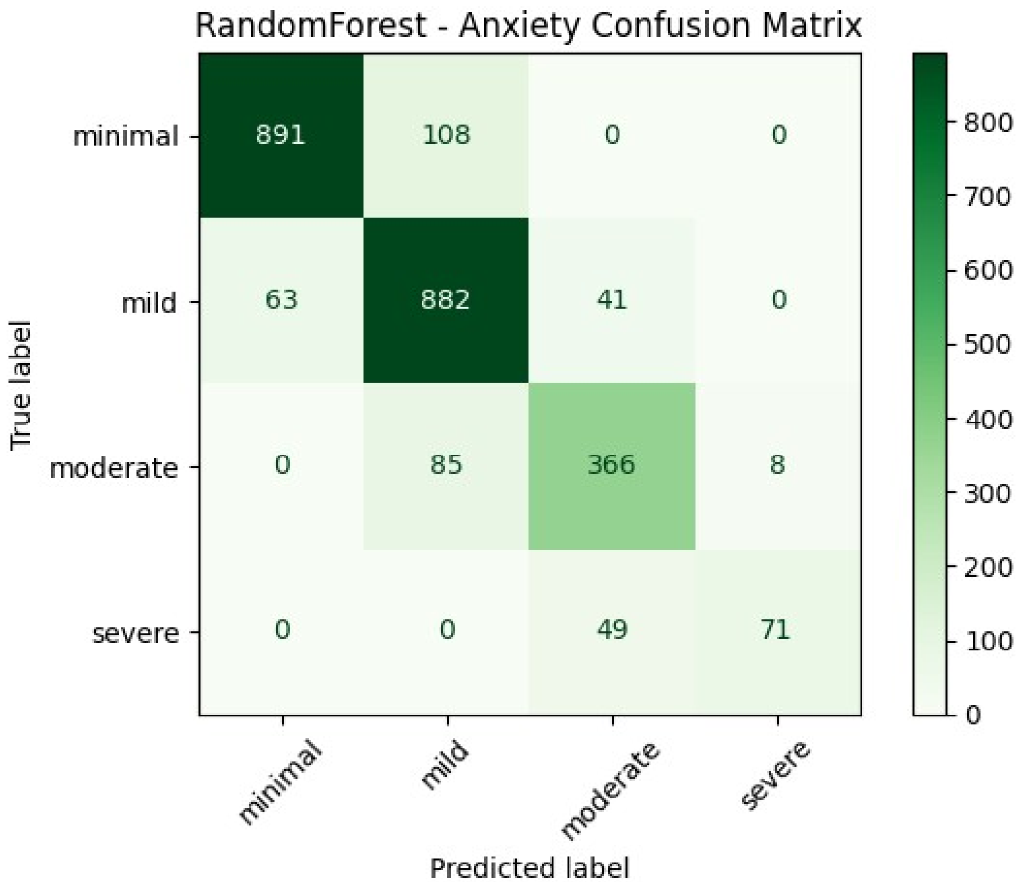

Logistic - Depression Confusion Matrix
none
mild
moderate
mod_severe
severe
True label
751
54
0
0
0
59
1004
7
0
0
0
242
291
0
0
0
0
119
16
1
0
0
1
8
11
none
mild
moderate
mod_severe
severe
Predicted label
1000
800
600
400
200
0




```
Training RandomForest with GridSearchCV
Fitting 3 folds for each of 12 candidates, totalling 36 fits
Best parameters for RandomForest: {'clf__estimator__max_depth': None,
'clf__estimator__min_samples_split': 5,
'clf__estimator__n_estimators': 200}
Best CV accuracy for RandomForest: 0.8259
Is_Depressed Recall: 0.5973
Has_anxiety Recall: 0.7939
Average Recall: 0.6956
Is_Depressed Accuracy: 0.8062
Has_anxiety Accuracy: 0.8619
Average Accuracy: 0.8340
Validation Average Accuracy: 0.8410

RandomForest Classification Report for Is_Depressed:
              precision    recall  f1-score   support

           0      0.922     0.807     0.861       805
           1      0.773     0.887     0.826      1070
           2      0.738     0.765     0.751       533
           3      0.753     0.426     0.545       136
```

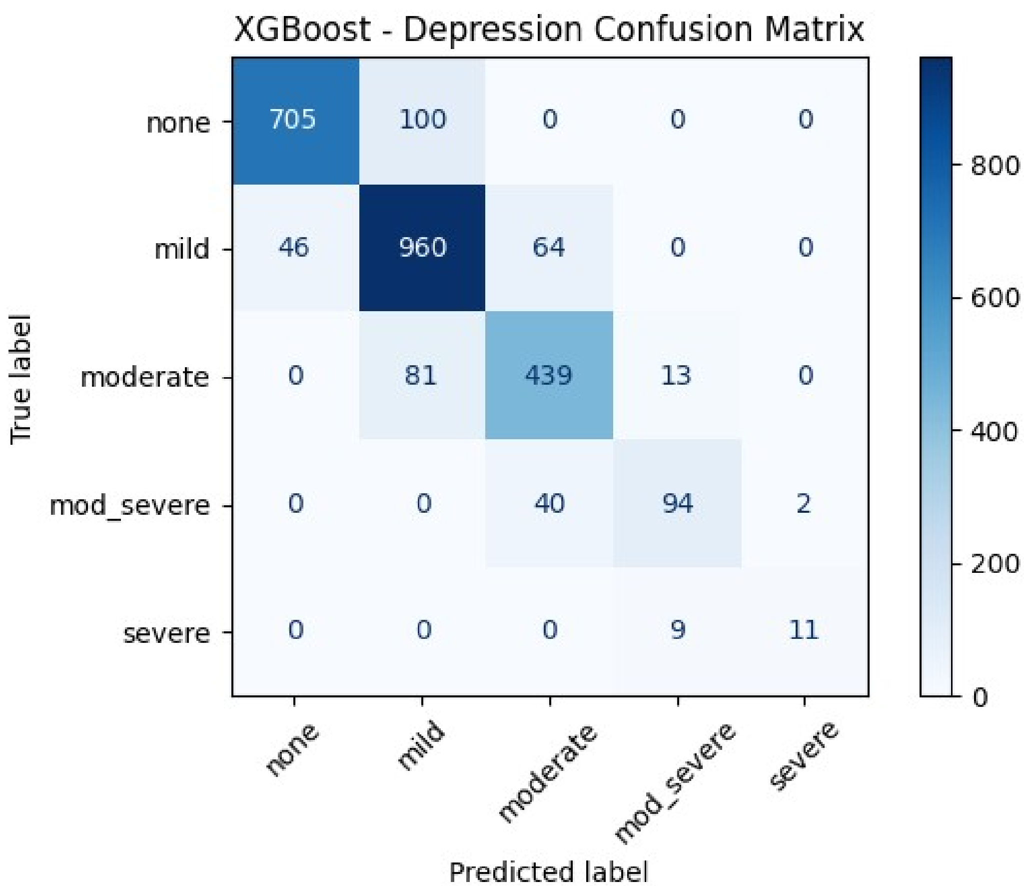
```
           4      1.000     0.100     0.182        20

    accuracy                          0.806      2564
   macro avg      0.837     0.597     0.633      2564
weighted avg      0.813     0.806     0.802      2564


RandomForest Classification Report for Has_anxiety:
              precision    recall  f1-score   support

           0      0.934     0.892     0.912       999
           1      0.820     0.895     0.856       986
           2      0.803     0.797     0.800       459
           3      0.899     0.592     0.714       120

    accuracy                          0.862      2564
   macro avg      0.864     0.794     0.820      2564
weighted avg      0.865     0.862     0.861      2564
```

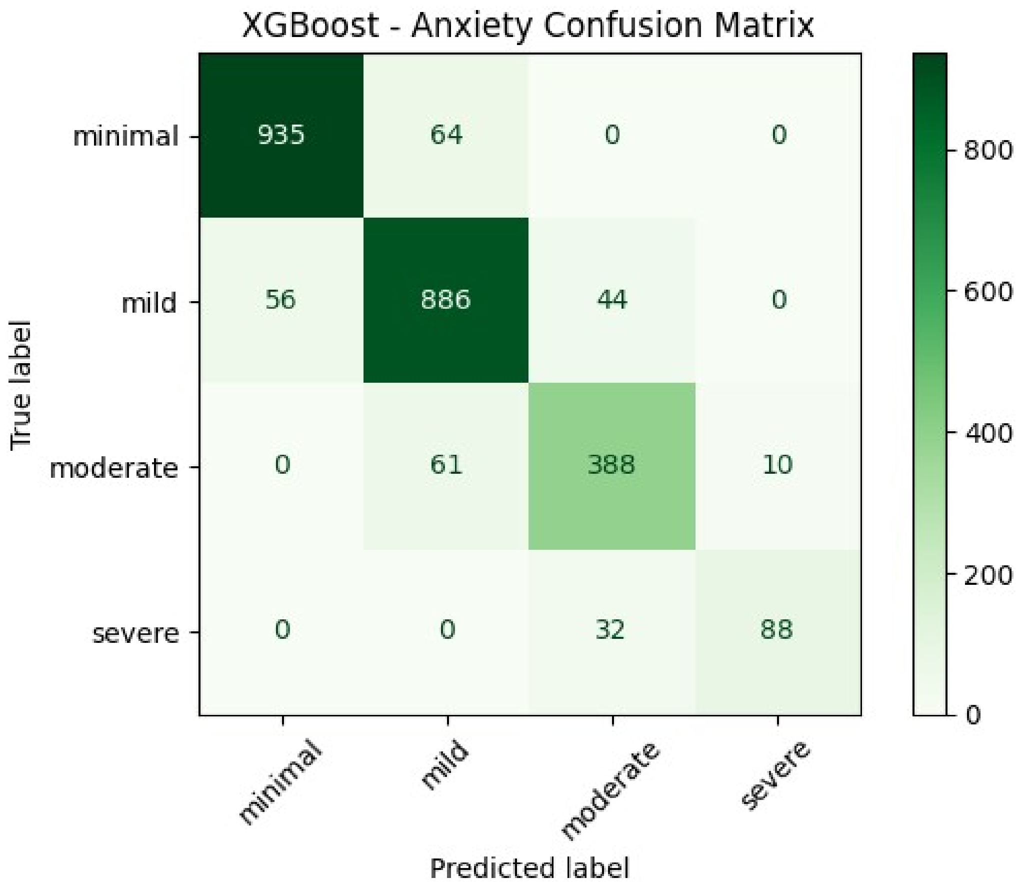





```
Training XGBoost with GridSearchCV
Fitting 3 folds for each of 8 candidates, totalling 24 fits
Best parameters for XGBoost: {'clf__estimator__learning_rate': 0.1,
'clf__estimator__max_depth': 6, 'clf__estimator__n_estimators': 200}
Best CV accuracy for XGBoost: 0.8751
Is_Depressed Recall: 0.7676
Has_anxiety Recall: 0.8533
Average Recall: 0.8104
Is_Depressed Accuracy: 0.8615
Has_anxiety Accuracy: 0.8959
Average Accuracy: 0.8787
Validation Average Accuracy: 0.8800

XGBoost Classification Report for Is_Depressed:
              precision    recall  f1-score   support

           0      0.939     0.876     0.906       805
           1      0.841     0.897     0.868      1070
           2      0.808     0.824     0.816       533
           3      0.810     0.691     0.746       136
           4      0.846     0.550     0.667        20
```

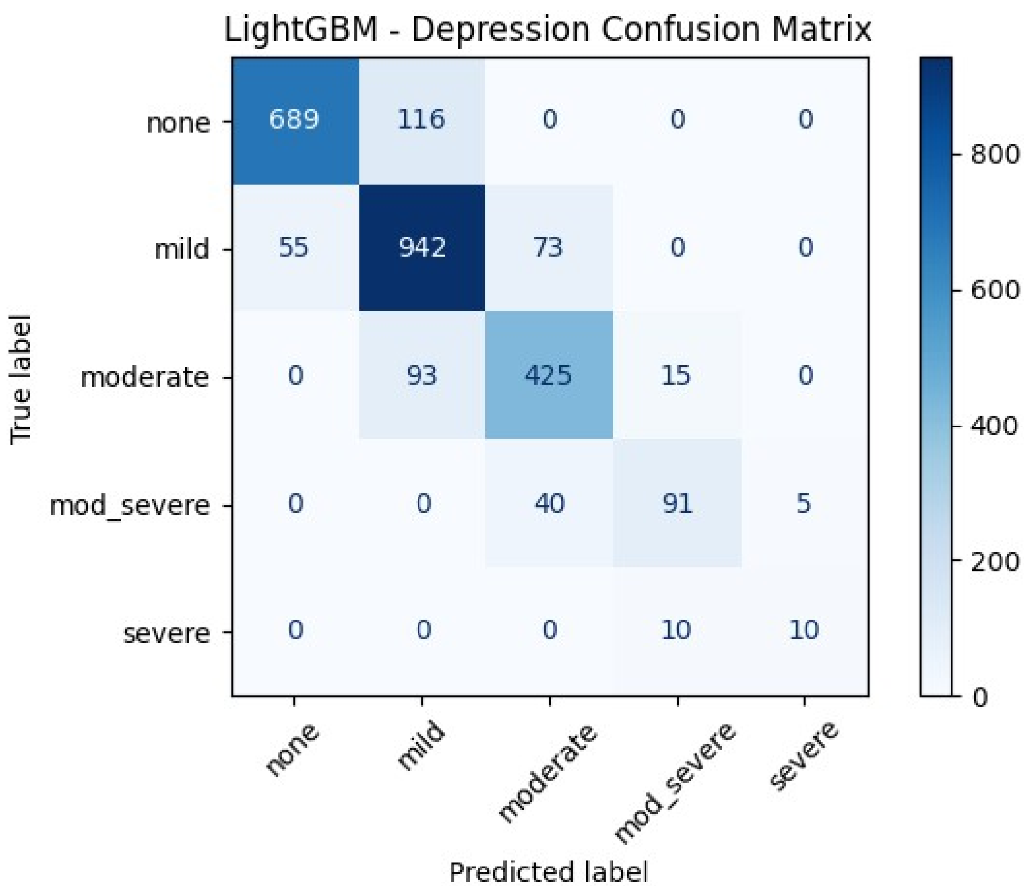
```
    accuracy                          0.862      2564
   macro avg      0.849     0.768     0.801      2564
weighted avg      0.863     0.862     0.861      2564


XGBoost Classification Report for Has_anxiety:
              precision    recall  f1-score   support

           0      0.943     0.936     0.940       999
           1      0.876     0.899     0.887       986
           2      0.836     0.845     0.841       459
           3      0.898     0.733     0.807       120

    accuracy                          0.896      2564
   macro avg      0.889     0.853     0.869      2564
weighted avg      0.896     0.896     0.896      2564
```

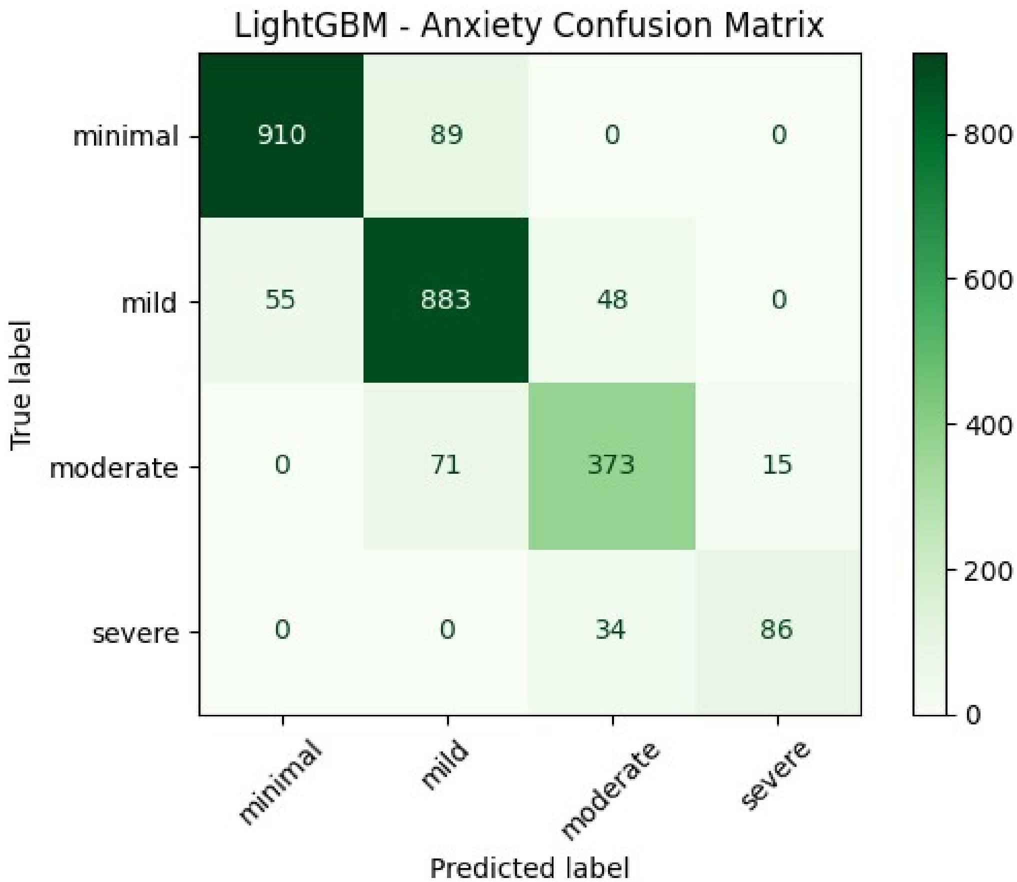





```
Training LightGBM with GridSearchCV
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Best parameters for LightGBM: {'clf__estimator__learning_rate': 0.1,
'clf__estimator__max_depth': 3, 'clf__estimator__n_estimators': 100}
Best CV accuracy for LightGBM: 0.8593
Is_Depressed Recall: 0.7406
Has_anxiety Recall: 0.8339
Average Recall: 0.7872
Is_Depressed Accuracy: 0.8413
Has_anxiety Accuracy: 0.8783
Average Accuracy: 0.8598
Validation Average Accuracy: 0.8619

LightGBM Classification Report for Is_Depressed:
              precision    recall  f1-score   support

           0      0.926     0.856     0.890       805
           1      0.818     0.880     0.848      1070
           2      0.790     0.797     0.794       533
           3      0.784     0.669     0.722       136
           4      0.667     0.500     0.571        20
```

```
    accuracy                          0.841      2564
   macro avg      0.797     0.741     0.765      2564
weighted avg      0.843     0.841     0.841      2564


LightGBM Classification Report for Has_anxiety:
              precision    recall  f1-score   support

           0      0.943     0.911     0.927       999
           1      0.847     0.896     0.870       986
           2      0.820     0.813     0.816       459
           3      0.851     0.717     0.778       120

    accuracy                          0.878      2564
   macro avg      0.865     0.834     0.848      2564
weighted avg      0.880     0.878     0.878      2564
```






```
Model metrics saved to model_metrics.pkl

Model Comparison Summary:

Logistic:
 Is_Depressed Test Accuracy: 0.809
 Has_anxiety Test Accuracy: 0.874
 Is_Depressed Recall: 0.617
 Has_anxiety Recall: 0.817
Average Test Accuracy: 0.841
Average Recall: 0.717
Average Validation Accuracy: 0.850

RandomForest:
 Is_Depressed Test Accuracy: 0.806
 Has_anxiety Test Accuracy: 0.862
 Is_Depressed Recall: 0.597
 Has_anxiety Recall: 0.794
Average Test Accuracy: 0.834
Average Recall: 0.696
Average Validation Accuracy: 0.841
```

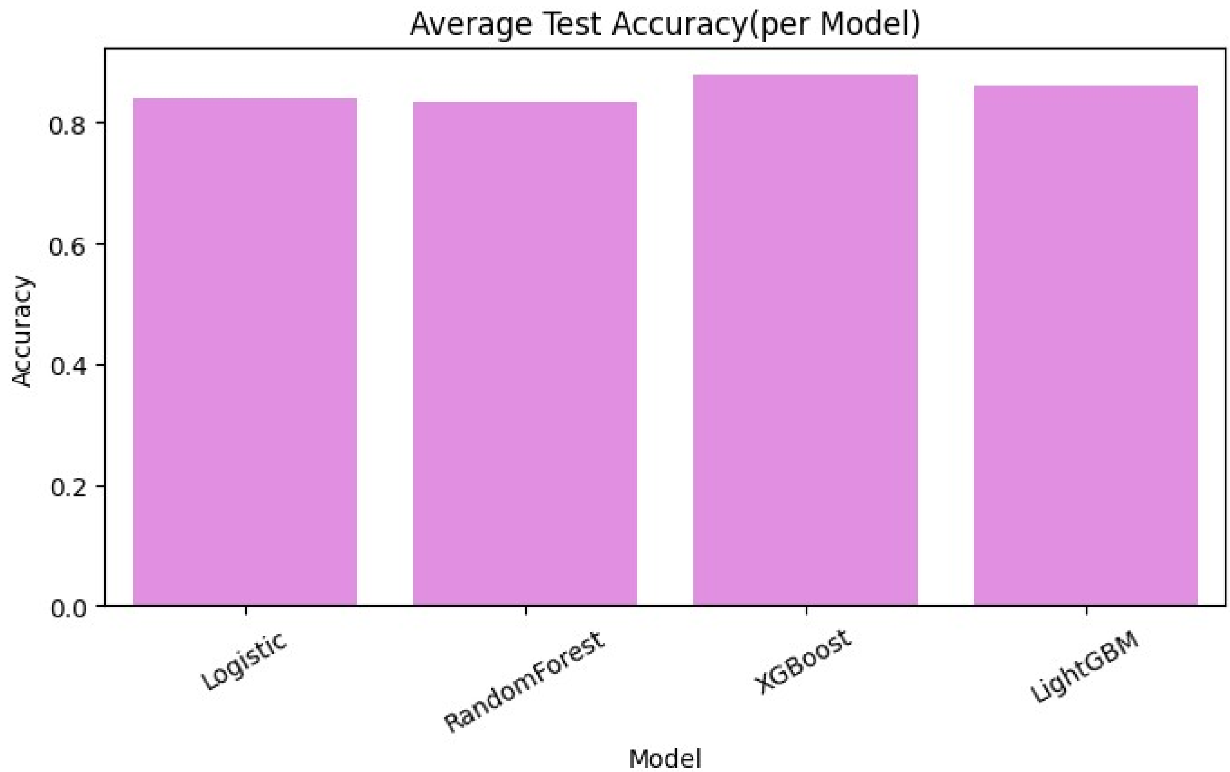
```
XGBoost:
 Is_Depressed Test Accuracy: 0.862
 Has_anxiety Test Accuracy: 0.896
 Is_Depressed Recall: 0.768
 Has_anxiety Recall: 0.853
Average Test Accuracy: 0.879
Average Recall: 0.810
Average Validation Accuracy: 0.880

LightGBM:
 Is_Depressed Test Accuracy: 0.841
 Has_anxiety Test Accuracy: 0.878
 Is_Depressed Recall: 0.741
 Has_anxiety Recall: 0.834
Average Test Accuracy: 0.860
Average Recall: 0.787
Average Validation Accuracy: 0.862
```

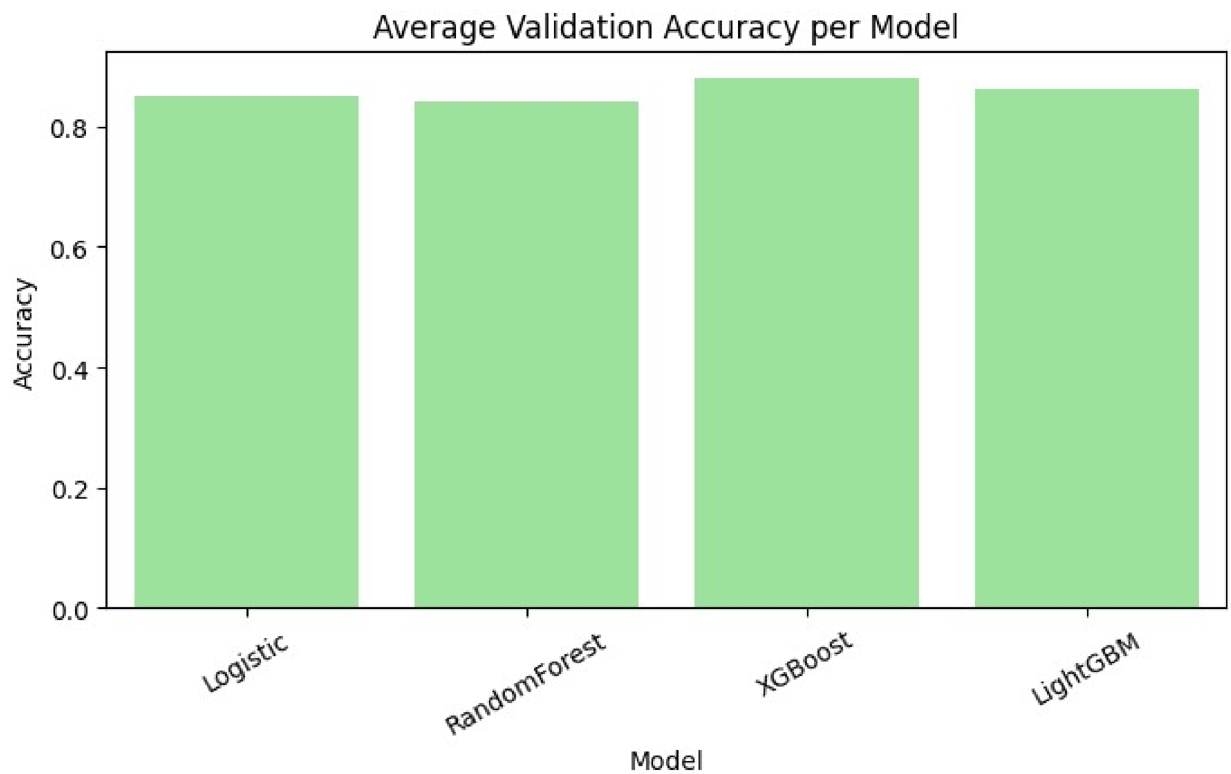
5. Evaluation & Interpretation

5.1 Test & Validation Accuracy per model

```
model_names = list(results.keys())
test_accuracies = [results[m]['average_test_accuracy'] for m in
model_names]

plt.figure(figsize=(8,4))
sns.barplot(x=model_names, y=test_accuracies, color='violet')
plt.title("Average Test Accuracy(per Model)", fontsize=12)
plt.ylabel("Accuracy")
plt.xlabel("Model")
plt.xticks(rotation=30)
plt.savefig('Image/Average Test Accuracy(per Model).png', dpi=300,
bbox_inches='tight')
plt.show()


validation_accuracies = [results[m]['average_validation_accuracy'] for
m in model_names]

plt.figure(figsize=(8,4))
sns.barplot(x=model_names, y=validation_accuracies,
color='lightgreen')
plt.title("Average Validation Accuracy per Model", fontsize=12)
plt.ylabel("Accuracy")
plt.xlabel("Model")
plt.xticks(rotation=30)
```

```
plt.savefig('Image/Average Validation Accuracy per Model.png',
dpi=300, bbox_inches='tight')
plt.show()
```





- Validation accuracy and Test accuracy in all models is almost similar, this means that our model is able to **generalize well on unseen data**. This also proves that **chances of our model being overfitted or underfitted are very low**.

The models focused on **predicting people with anxiety and depression**, however it does not just predict whether a person is depressed or not rather `it predicts the severity of the depression or anxiety`. For this project **recall** was used as the major metric. All the models used for this project performed well while identifying the various depression and anxiety levels.

- **Logistic Regression**- The model `works very well for the common classes (No depression and Mild)` which means it does not miss many of these cases. It however seems to `struggle more with the higher severity levels; categories 3 and 4`, where **recall is very low**. In actual applications, this means that the model may be underestimating the cases of serious depression.
- **Random Forest**- The Random Forest model shows `acceptable overall accuracy and good recall for mild and moderate levels`, which suggests it can identify symptoms early. However, it `underperformed when identifying cases of severe depression and anxiety`, which is the most important level of mental health diagnosis to identify in practice. A `recall of 0.50 on severe depression cases and 0.583 recall for severe anxiety` indicated that class balancing in training data may help identify individuals in severe cases of mental health diagnoses.
- **XGBoost**- In comparison to Random Forest, the adjusted XGBoost model shows `improved and more consistently high recall rates, but primarily for moderate-risk cases`. The model still **struggled with severe depression recall (40%)**, it fared better in identifying patients with severe anxiety (recall = 0.699). Given

the XGBoost model had a `higher overall recall and accuracy`, it is seen as a `better model for identification of mild and moderate cases`, and slightly better in the severe case category.

- **LightGBM** - LightGBM **did well overall for depression with an overall accuracy of 83.8%**. **High recall for the mild and moderate classes** (0, 1, and 2), suggesting that the model will get people on the lower end of the severity of depression spectrum. `Recall was significantly lower for the more severe classes` (3, and 4), with recall reported as 0.624 for class 3 and 0.545 for class 4.This shows there are instances where the `model does not identify someone with severe or very severe depression`, and this may be related to fewer cases of severe depression in the dataset. Therefore, even with a high overall accuracy, recall for the more severe levels is lower, which shows the `model is not as sensitive in the more serious classes, which is a considerable limitation when detecting individuals that may need the most help`. For **anxiety, the overall accuracy was a little higher, at 88%. The recall values were also fairly good overall for anxiety**, with the mild and moderate classes being above 0.80, and the severe class (3) being at 0.798. Therefore, this indicates that the `model is doing a better job detecting anxiety than depression` and was able to catch most levels of anxiety correctly. However, again, the model `missed detecting some of the very severe cases, but was not missing cases as well when gauging depression.`

5.2 Model Interpretability using Shap(Shapley Additive explanations)

Sharply summary plots `help us get a deeeper understanding of the features that made our model make a certain prediction` i.e why our model predicted that a person is depressed and why it predicted anxiety.

```
import joblib
import shap
import os
import matplotlib.pyplot as plt

# Load saved model metrics dictionary
results = joblib.load("model_metrics.pkl")

# Define available model files
model_files = ["Logistic_model.pkl", "RandomForest_model.pkl",
"XGBoost_model.pkl", "LightGBM_model.pkl"]
model_names = [f.replace("_model.pkl", "") for f in model_files]

# mapping between shortened names and full names in results
name_mapping = {}
for short_name in model_names:
    for full_name in results.keys():
        if short_name.lower() in full_name.lower():
            name_mapping[short_name] = full_name
```

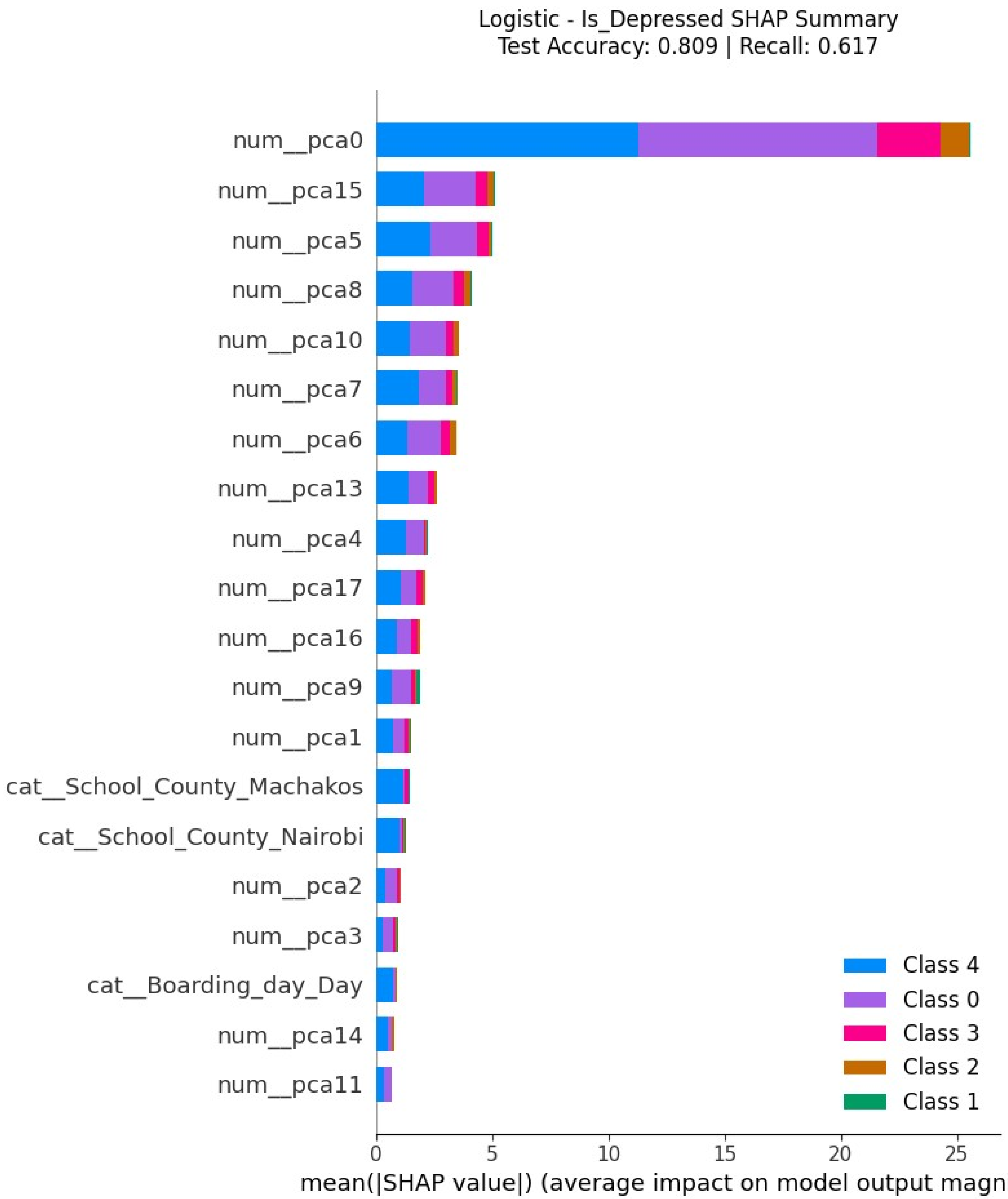
```
            break

print("Name mapping:", name_mapping)

#  Load the trained pipelines
trained_pipelines = {}
for file, name in zip(model_files, model_names):
    trained_pipelines[name] = joblib.load(file)

# Determine the best model by average recall
best_model_name = None
best_recall = -1

for name, metrics in results.items():
    recall = metrics.get('average_recall', 0)
    if recall > best_recall:
        best_recall = recall
        best_model_name = name

print(f"\nBest model by recall: {best_model_name} (Avg Recall =
{best_recall:.3f})")

# Save the best model
with open("best_model.txt", "w") as f:
    f.write(best_model_name)

# data for SHAP explanations
X_sample = X_test.sample(100, random_state=42)

# Generate SHAP explanations for all models
for name, model in trained_pipelines.items():
    print(f"\n{'='*60}")
    print(f" Model: {name}")

    # Get the full name from mapping
    full_name = name_mapping.get(name)

    # Retrieve metrics
    if full_name is None or full_name not in results:
        print(f"No metrics found for {name} (looking for
{full_name})")
        continue

    metrics = results[full_name]

    # Display overall metrics
    print(f"\nOverall Metrics:")
    print(f"Average Test Accuracy:
{metrics.get('average_test_accuracy', 0):.3f}")
    print(f"Average Recall: {metrics.get('average_recall', 0):.3f}")
```

```
    # Display per-target metrics
    print(f"\nPer-Target Test Accuracy:")
    test_acc_per_target = metrics.get('test_accuracy_per_target', {})
    for target, acc in test_acc_per_target.items():
        print(f" {target}: {acc:.3f}")

    print(f"\nPer-Target Recall:")
    recall_per_target = metrics.get('test_recall_per_target', {})
    for target, rec in recall_per_target.items():
        print(f"{target}: {rec:.3f}")

    # Extract pipeline components
    preprocessor = model.named_steps['preprocessor']
    multi_model = model.named_steps['clf']

    # Transform X for SHAP
    X_transformed = preprocessor.transform(X_sample)
    feature_names = preprocessor.get_feature_names_out()

    # Loop over targets (Depression, Anxiety)
    for i, target_name in enumerate(["Is_Depressed", "Has_anxiety"]):
        print(f"\n{'—'*50}")
        print(f"Explaining predictions for: {target_name}")

        # Display target-specific metrics
        target_acc = test_acc_per_target.get(target_name, 0)
        target_rec = recall_per_target.get(target_name, 0)
        print(f" Test Accuracy: {target_acc:.3f}")
        print(f" Recall: {target_rec:.3f}")

        single_model = multi_model.estimators_[i]

        # Choose SHAP explainer type
        if "Logistic" in name:
            explainer = shap.Explainer(single_model, X_transformed)
        else:
            explainer = shap.TreeExplainer(single_model)

        shap_values = explainer(X_transformed)

        # Plot and save SHAP summary
        plt.figure(figsize=(10, 6))
        shap.summary_plot(shap_values, X_transformed,
feature_names=feature_names, show=False)

        title = f"{name} - {target_name} SHAP Summary\n"
        title += f"Test Accuracy: {target_acc:.3f} | Recall:
{target_rec:.3f}"
        plt.title(title, fontsize=12, pad=20)
```

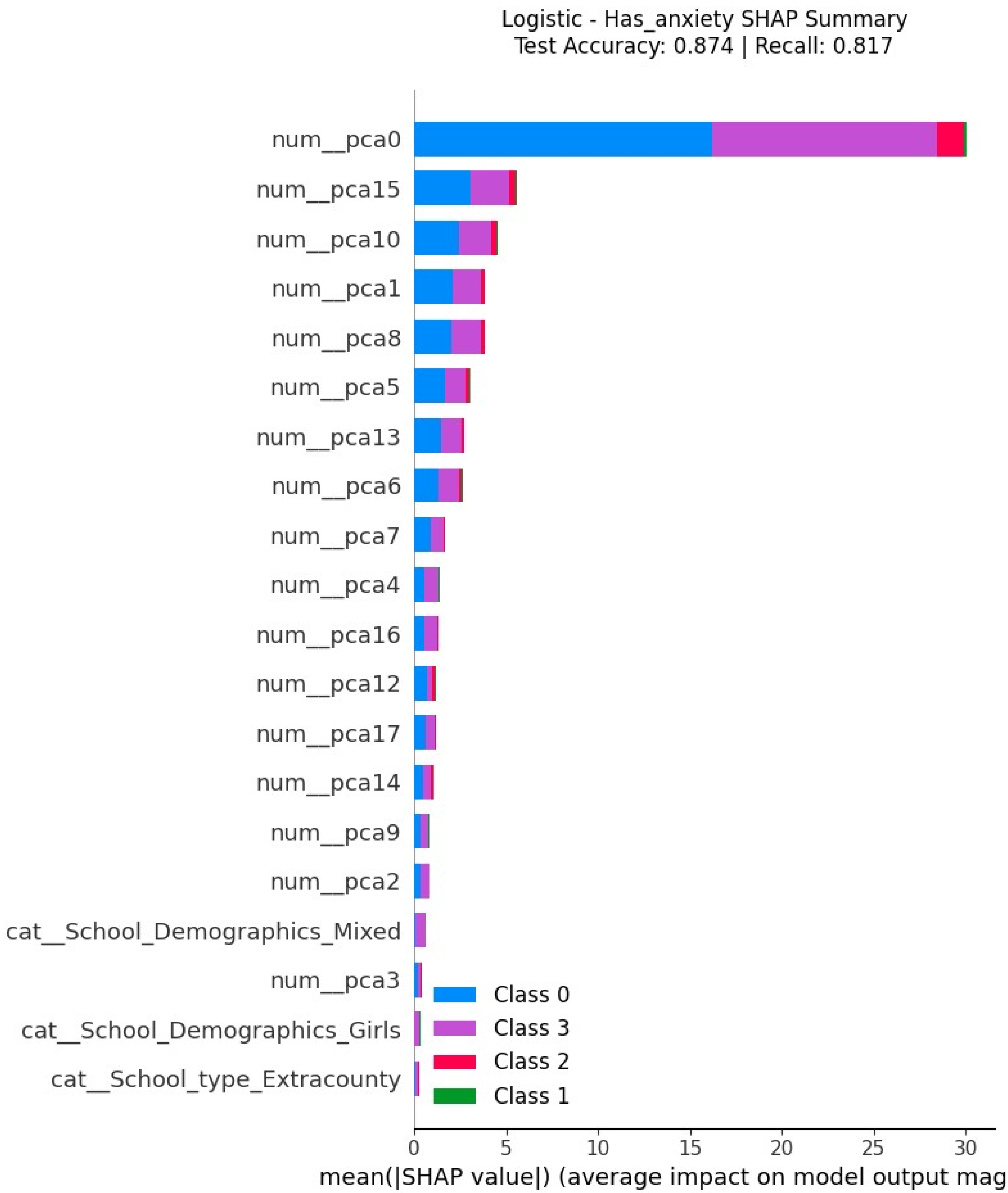
```python
        plt.tight_layout()

        file_path = f"Image/{name}_{target_name}_SHAP.png"
        plt.savefig(file_path, dpi=300, bbox_inches='tight')
        plt.show()
        plt.close()

# Check if keys match
if set(trained_pipelines.keys()) != set(results.keys()):
    # Ensure consistent naming
    trained_pipelines_fixed = {}
    for key in results.keys():
        if key in trained_pipelines:
            trained_pipelines_fixed[key] = trained_pipelines[key]
        else:
            # Try to find matching pipeline
            for pipe_key in trained_pipelines.keys():
                if pipe_key.lower() in key.lower() or key.lower() in
pipe_key.lower():
                    trained_pipelines_fixed[key] =
trained_pipelines[pipe_key]
                    break
    trained_pipelines = trained_pipelines_fixed
# Save trained pipelines for Streamlit
with open('trained_pipelines.pkl', 'wb') as f:
    pickle.dump(trained_pipelines, f, protocol=4)

size_mb = os.path.getsize('trained_pipelines.pkl') / (1024*1024)
print(f"\nSaved trained_pipelines.pkl ({size_mb:.2f} MB)")

with open('trained_pipelines.pkl', 'rb') as f:
    test_pipes = pickle.load(f)
test_metrics = joblib.load("model_metrics.pkl")

print(f"Pipeline models: {list(test_pipes.keys())}")
print(f"Metrics models: {list(test_metrics.keys())}")
```

```
Name mapping: {'Logistic': 'Logistic', 'RandomForest': 'RandomForest',
'XGBoost': 'XGBoost', 'LightGBM': 'LightGBM'}

Best model by recall: XGBoost (Avg Recall = 0.810)

============================================================
 Model: Logistic

Overall Metrics:
Average Test Accuracy: 0.841
Average Recall: 0.717
```

```
Per-Target Test Accuracy:
 Is_Depressed: 0.809
 Has_anxiety: 0.874

Per-Target Recall:
Is_Depressed: 0.617
Has_anxiety: 0.817

──────────────────────────────────────────────────
Explaining predictions for: Is_Depressed
 Test Accuracy: 0.809
 Recall: 0.617
```

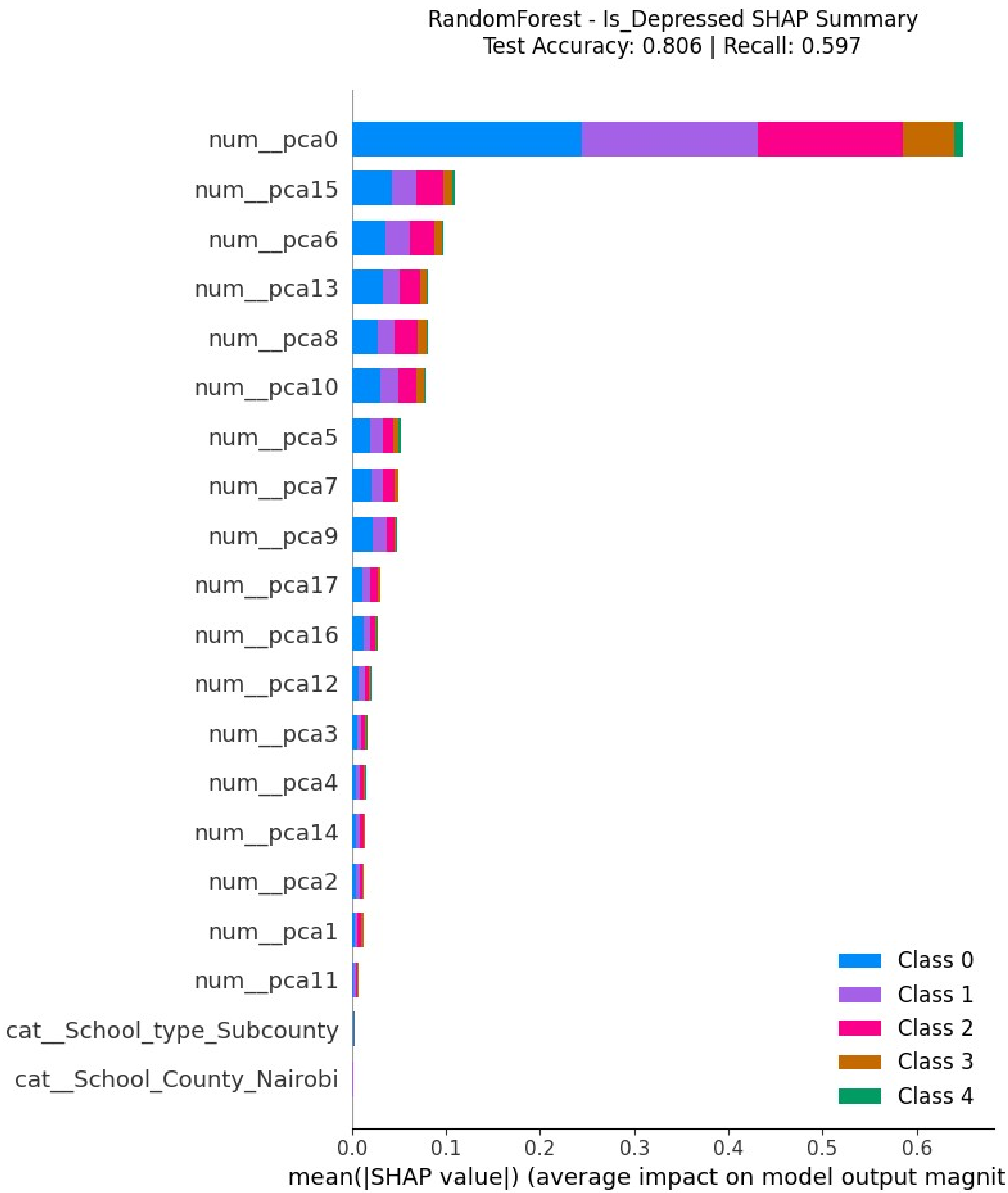



```
──────────────────────────────────────────────────
Explaining predictions for: Has_anxiety
 Test Accuracy: 0.874
 Recall: 0.817
```

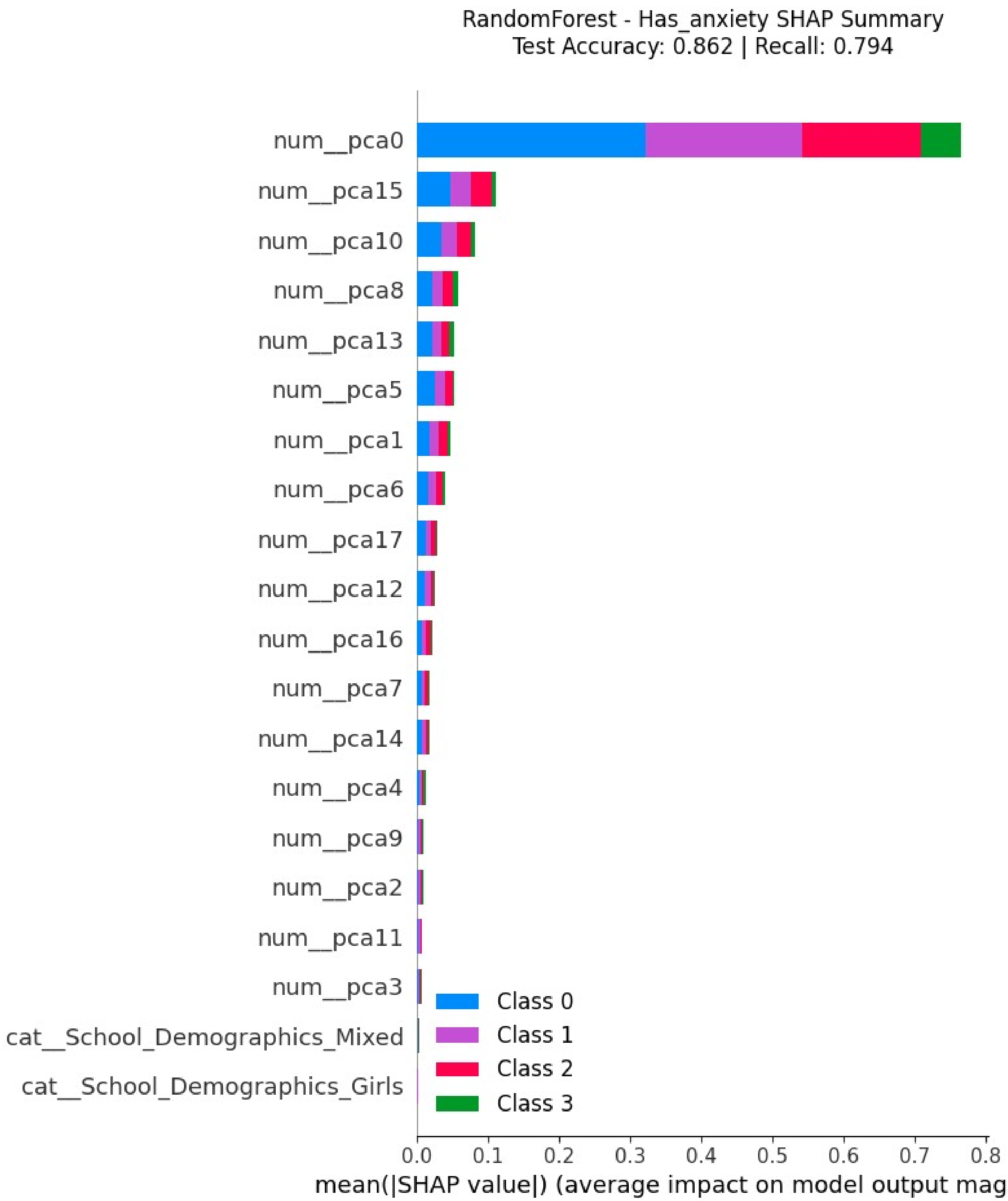



```
============================================================
 Model: RandomForest

Overall Metrics:
```

```
Average Test Accuracy: 0.834
Average Recall: 0.696

Per-Target Test Accuracy:
 Is_Depressed: 0.806
 Has_anxiety: 0.862

Per-Target Recall:
Is_Depressed: 0.597
Has_anxiety: 0.794

──────────────────────────────────────────────────
Explaining predictions for: Is_Depressed
 Test Accuracy: 0.806
 Recall: 0.597
```

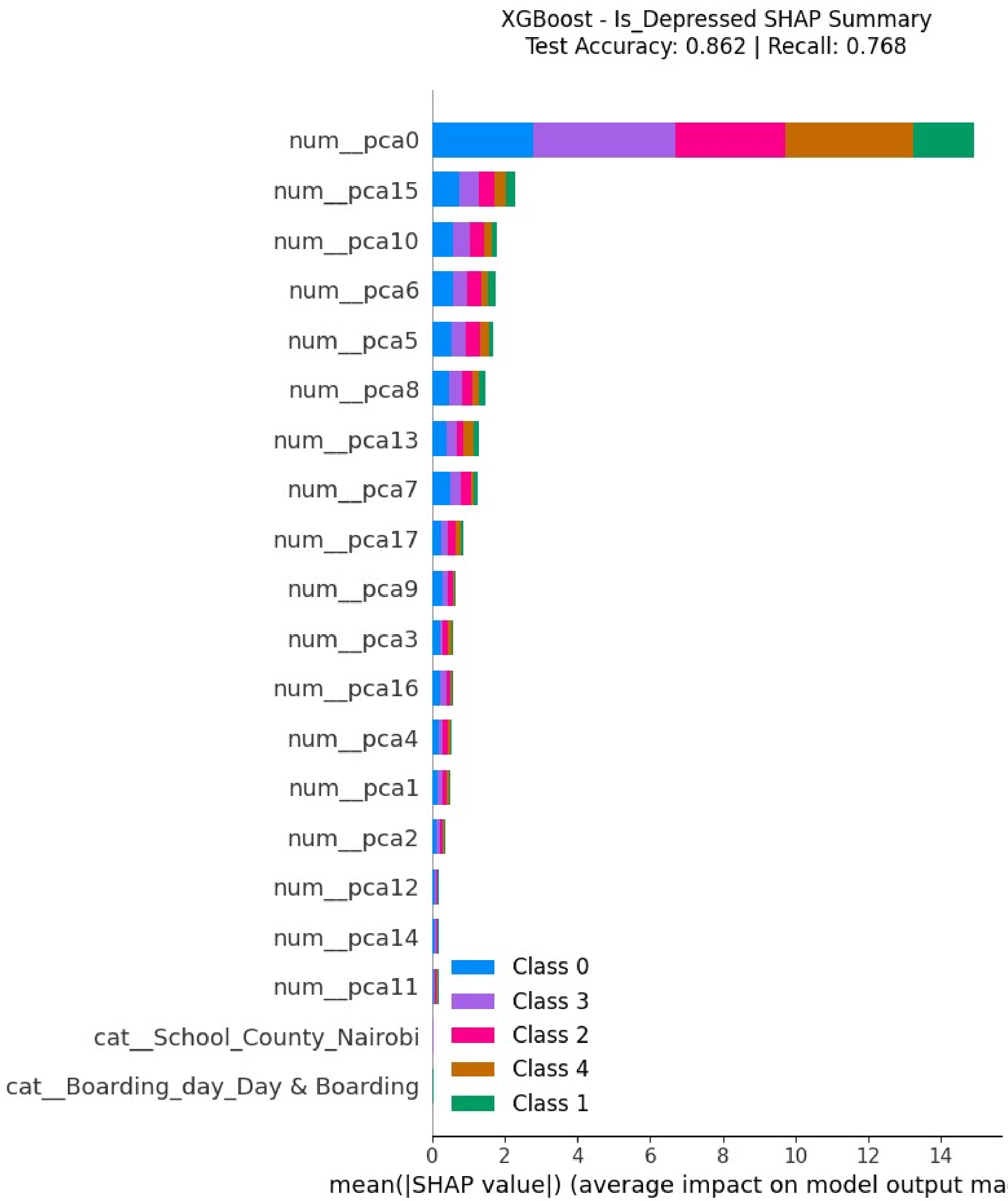



```
Explaining predictions for: Has_anxiety
 Test Accuracy: 0.862
 Recall: 0.794
```

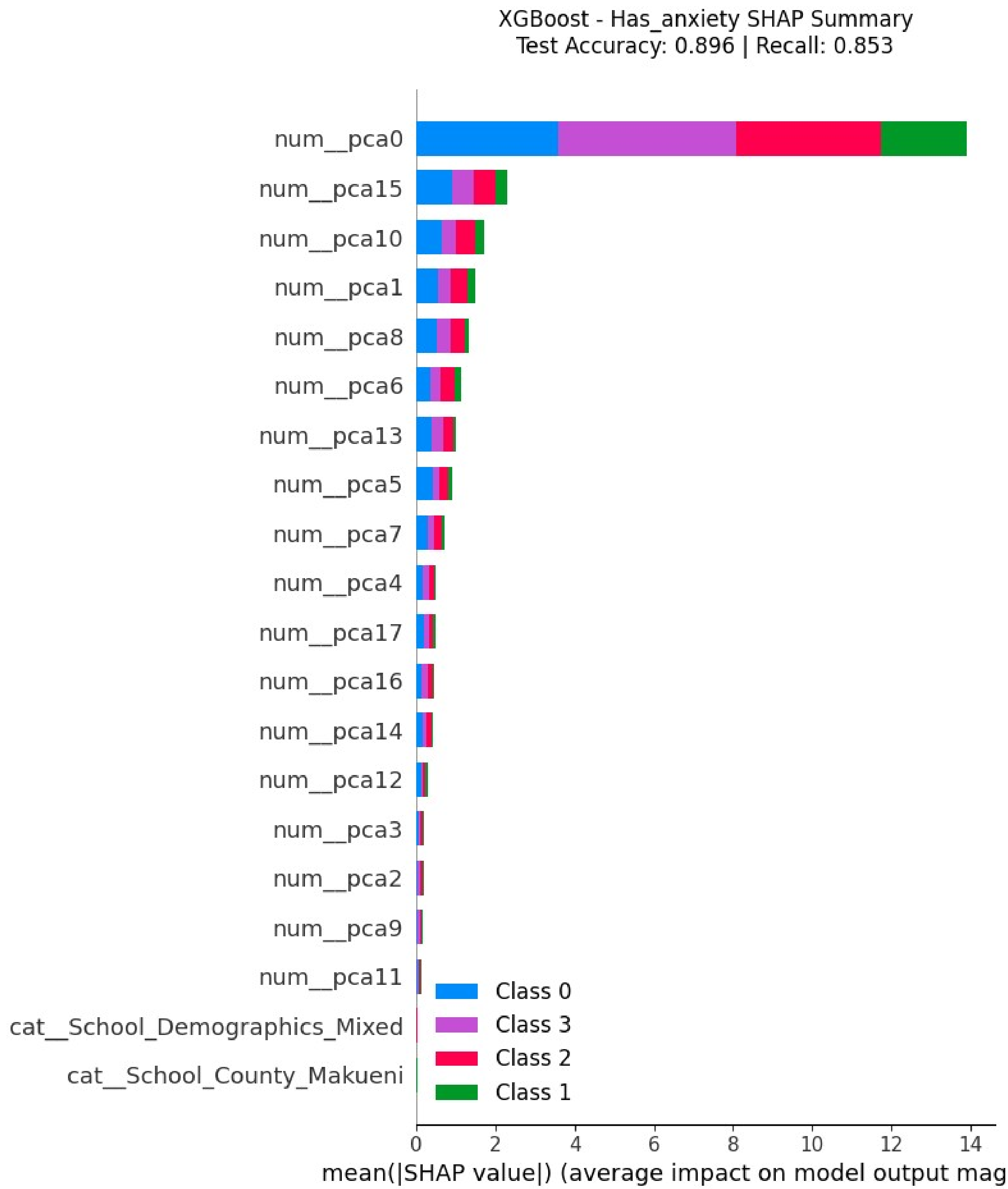



```
============================================================
 Model: XGBoost

Overall Metrics:
```

```
Average Test Accuracy: 0.879
Average Recall: 0.810

Per-Target Test Accuracy:
 Is_Depressed: 0.862
 Has_anxiety: 0.896

Per-Target Recall:
Is_Depressed: 0.768
Has_anxiety: 0.853

──────────────────────────────────────────────────
Explaining predictions for: Is_Depressed
 Test Accuracy: 0.862
 Recall: 0.768
```

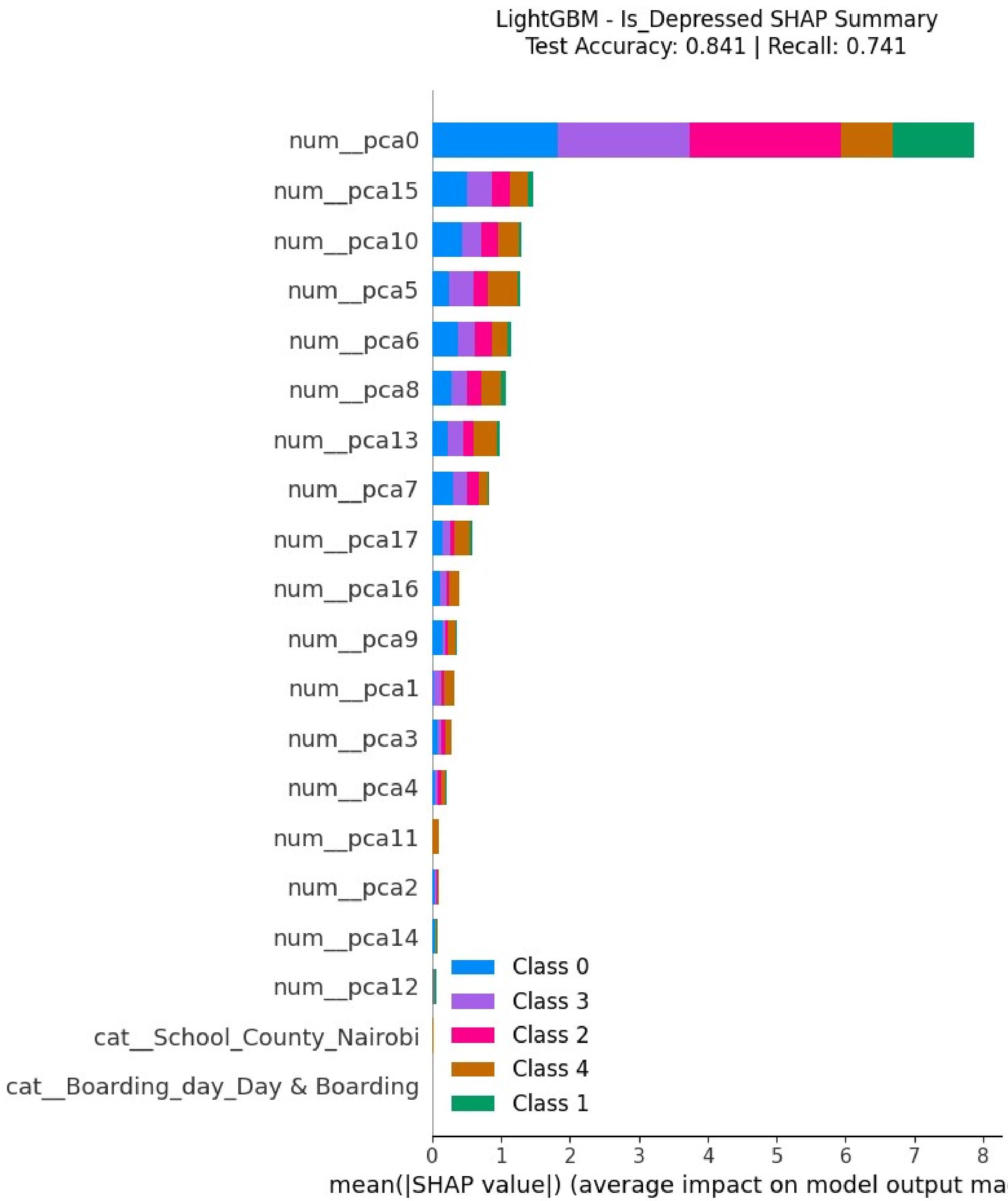



```
Explaining predictions for: Has_anxiety
 Test Accuracy: 0.896
 Recall: 0.853
```

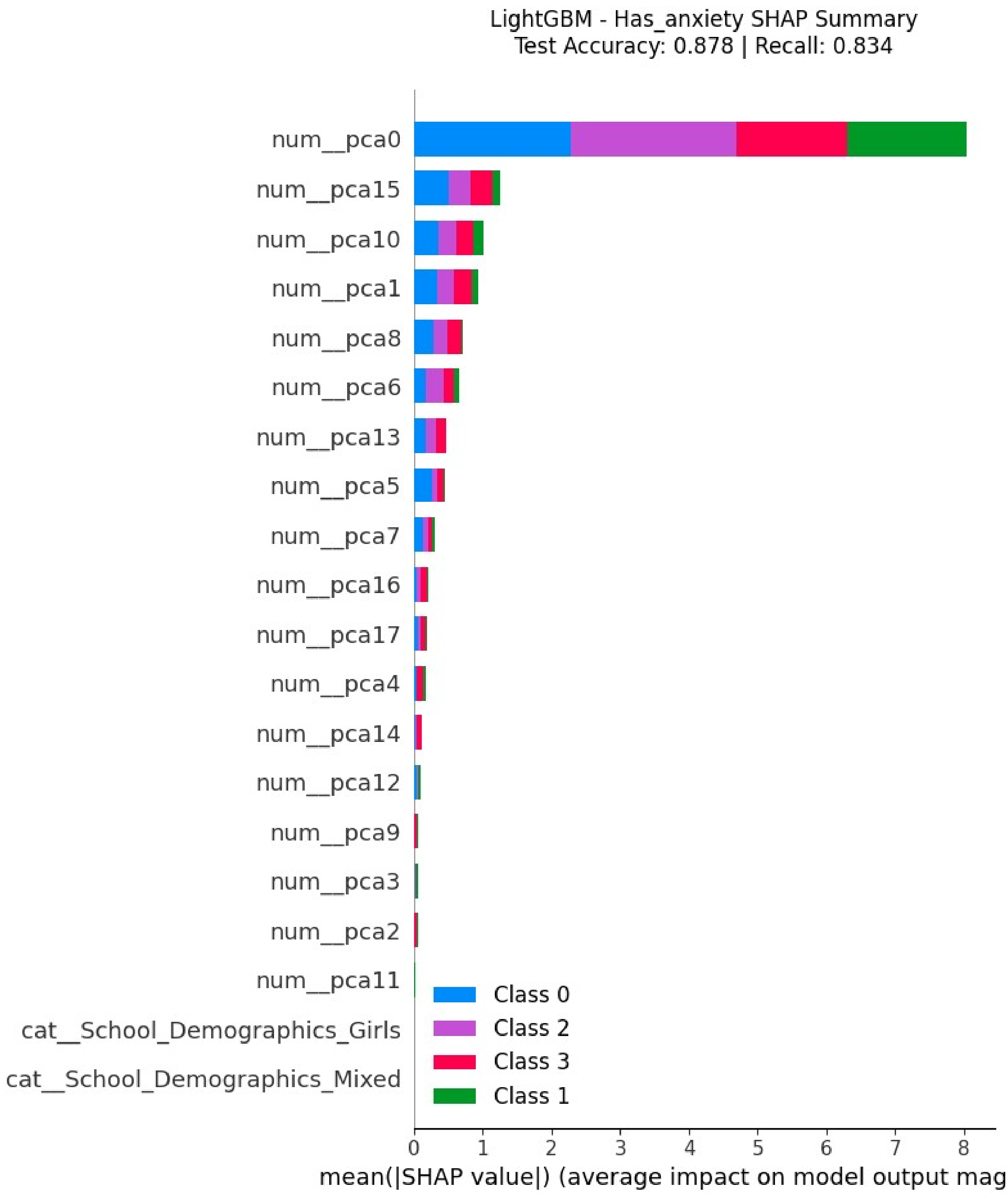



```
============================================================
 Model: LightGBM

Overall Metrics:
```

```
Average Test Accuracy: 0.860
Average Recall: 0.787

Per-Target Test Accuracy:
 Is_Depressed: 0.841
 Has_anxiety: 0.878

Per-Target Recall:
Is_Depressed: 0.741
Has_anxiety: 0.834

──────────────────────────────────────────────────
Explaining predictions for: Is_Depressed
 Test Accuracy: 0.841
 Recall: 0.741
```




```
--------------------------------------------------
Explaining predictions for: Has_anxiety
 Test Accuracy: 0.878
 Recall: 0.834
```




```
Saved trained_pipelines.pkl (104.40 MB)
Pipeline models: ['Logistic', 'RandomForest', 'XGBoost', 'LightGBM']
Metrics models: ['Logistic', 'RandomForest', 'XGBoost', 'LightGBM']
```

The above **Shap summary** `visualizes the features that had a major impact on our models predictions`. They try to make us understand why our model made a certain prediction whether depressed or anxious. Initially our features were such like PHQ1,GAD1,Age e.t.c but since we had a pipeline with PCA to help in dimensionality reduction, the summary plot now has features like num__pca14, cat__School_County_Nairobi. This `new features are a combination of the original features such as Age and PCA1`. With such a summary plot it is usually made up of dots but in our case each target has several possible outcomes such as none,mild and moderate thus the stacked bar graph. **The length of the bar determines it's influence on predictions with longer bars having more influence.**

```
models = ["Logistic_model.pkl", "RandomForest_model.pkl",
"XGBoost_model.pkl", "LightGBM_model.pkl"]

for m in models:
    print("\n", m, "")
    model = joblib.load(m)
    pre = model.named_steps['preprocessor']

    num_cols = pre.transformers_[0][2]
    cat_cols = pre.transformers_[1][2]
    features = pre.get_feature_names_out()

    # check for PCA
    num = pre.named_transformers_['num']
    pca_map = {}
    if hasattr(num, 'named_steps') and 'pca' in num.named_steps:
        pca = num.named_steps['pca']
        for i, comp in enumerate(pca.components_):
            top = [num_cols[j] for j in np.argsort(abs(comp))[-3:]]
            pca_map[f'num__pca{i}'] = top

    # categorical names
    cat = pre.named_transformers_['cat']
    if hasattr(cat, 'get_feature_names_out'):
        cat_names = cat.get_feature_names_out(cat_cols)
    else:
        cat_names = 
cat.named_steps['onehot'].get_feature_names_out(cat_cols)

    data = []
    for f in features:
        if f.startswith("num__pca"):
            data.append([f, f"{', '.join(pca_map.get(f, []))}"])
        elif f.startswith("cat__"):
            c = f.split("__")[1].split("_", 1)
            data.append([f, f"{c[0]} = {c[1] if len(c)>1 else ''}"])
        else:
            data.append([f, "Numeric feature"])
```

```
    print(pd.DataFrame(data, columns=["Feature", "Major
features"]).to_string(index=False))
```

```
 Logistic_model.pkl
                                                Feature
Major features
                                              num__pca0                                GAD_1,
GAD_3, GAD_2
                                              num__pca1
Percieved_Academic_Abilities, Form, Age
                                              num__pca2 Parents_Home, Mothers_Education,
Fathers_Education
                                              num__pca3      Mothers_Education, Parents_Home,
Parents_Dead
                                              num__pca4                      Sports, Gender,
Co_Curricular
                                              num__pca5                    Sports, Religion,
Co_Curricular
                                              num__pca6                                Sports,
PHQ_1, Religion
                                              num__pca7                                Sports,
Gender, PHQ_7
                                              num__pca8                                Gender,
PHQ_3, PHQ_5
                                              num__pca9                                PHQ_3,
Religion, PHQ_1
                                             num__pca10                                Gender,
PHQ_8, PHQ_4
                                             num__pca11                    PHQ_5, Sports,
Co_Curricular
                                             num__pca12                                PHQ_5,
Gender, PHQ_4
                                             num__pca13                                PHQ_5,
GAD_4, PHQ_8
                                             num__pca14         GAD_6, PHQ_3,
Percieved_Academic_Abilities
                                             num__pca15                                PHQ_5,
GAD_6, PHQ_2
                                             num__pca16                                Gender,
GAD_4, PHQ_7
                                             num__pca17       Percieved_Academic_Abilities,
GAD_7, GAD_6
                                  cat__Boarding_day_Day
Boarding = day_Day
cat__Boarding_day_Day & Boarding                        Boarding =
day_Day & Boarding
                        cat__School_type_Extracounty                        School =
type_Extracounty
                          cat__School_type_Subcounty                          School =
```

```
type_Subcounty
  cat__School_Demographics_Girls                        School =
Demographics_Girls
  cat__School_Demographics_Mixed                        School =
Demographics_Mixed
     cat__School_County_Machakos                          School =
County_Machakos
      cat__School_County_Makueni                           School =
County_Makueni
      cat__School_County_Nairobi                           School =
County_Nairobi

 RandomForest_model.pkl
                         Feature
Major features
                       num__pca0                                  GAD_1,
GAD_3, GAD_2
                       num__pca1
Percieved_Academic_Abilities, Form, Age
                       num__pca2 Parents_Home, Mothers_Education,
Fathers_Education
                       num__pca3      Mothers_Education, Parents_Home,
Parents_Dead
                       num__pca4                        Sports, Gender,
Co_Curricular
                       num__pca5                      Sports, Religion,
Co_Curricular
                       num__pca6                                Sports,
PHQ_1, Religion
                       num__pca7                                  Sports,
Gender, PHQ_7
                       num__pca8                                   Gender,
PHQ_3, PHQ_5
                       num__pca9                                 PHQ_3,
Religion, PHQ_1
                      num__pca10                                   Gender,
PHQ_8, PHQ_4
                      num__pca11                         PHQ_5, Sports,
Co_Curricular
                      num__pca12                                   PHQ_5,
Gender, PHQ_4
                      num__pca13                                    PHQ_5,
GAD_4, PHQ_8
                      num__pca14         GAD_6, PHQ_3,
Percieved_Academic_Abilities
                      num__pca15                                    PHQ_5,
GAD_6, PHQ_2
                      num__pca16                                   Gender,
GAD_4, PHQ_7
```

```
                      num__pca17         Percieved_Academic_Abilities,
GAD_7, GAD_6
           cat__Boarding_day_Day
Boarding = day_Day
cat__Boarding_day_Day & Boarding                      Boarding =
day_Day & Boarding
    cat__School_type_Extracounty                          School =
type_Extracounty
      cat__School_type_Subcounty                            School =
type_Subcounty
  cat__School_Demographics_Girls                        School =
Demographics_Girls
  cat__School_Demographics_Mixed                        School =
Demographics_Mixed
     cat__School_County_Machakos                           School =
County_Machakos
      cat__School_County_Makueni                            School =
County_Makueni
      cat__School_County_Nairobi                            School =
County_Nairobi

 XGBoost_model.pkl
                            Feature
Major features
                       num__pca0                               GAD_1,
GAD_3, GAD_2
                       num__pca1
Percieved_Academic_Abilities, Form, Age
                       num__pca2 Parents_Home, Mothers_Education,
Fathers_Education
                       num__pca3      Mothers_Education, Parents_Home,
Parents_Dead
                       num__pca4                      Sports, Gender,
Co_Curricular
                       num__pca5                    Sports, Religion,
Co_Curricular
                       num__pca6                              Sports,
PHQ_1, Religion
                       num__pca7                                Sports,
Gender, PHQ_7
                       num__pca8                                Gender,
PHQ_3, PHQ_5
                       num__pca9                               PHQ_3,
Religion, PHQ_1
                      num__pca10                                Gender,
PHQ_8, PHQ_4
                      num__pca11                       PHQ_5, Sports,
Co_Curricular
                      num__pca12                                PHQ_5,
```

```
Gender, PHQ_4
                      num__pca13                                PHQ_5,
GAD_4, PHQ_8
                      num__pca14         GAD_6, PHQ_3,
Percieved_Academic_Abilities
                      num__pca15                                PHQ_5,
GAD_6, PHQ_2
                      num__pca16                               Gender,
GAD_4, PHQ_7
                      num__pca17         Percieved_Academic_Abilities,
GAD_7, GAD_6
           cat__Boarding_day_Day
Boarding = day_Day
cat__Boarding_day_Day & Boarding                     Boarding =
day_Day & Boarding
    cat__School_type_Extracounty                         School =
type_Extracounty
      cat__School_type_Subcounty                           School =
type_Subcounty
  cat__School_Demographics_Girls                       School =
Demographics_Girls
  cat__School_Demographics_Mixed                       School =
Demographics_Mixed
     cat__School_County_Machakos                          School =
County_Machakos
      cat__School_County_Makueni                           School =
County_Makueni
      cat__School_County_Nairobi                           School =
County_Nairobi

 LightGBM_model.pkl
                         Feature
Major features
                       num__pca0                                GAD_1,
GAD_3, GAD_2
                       num__pca1
Percieved_Academic_Abilities, Form, Age
                       num__pca2 Parents_Home, Mothers_Education,
Fathers_Education
                       num__pca3      Mothers_Education, Parents_Home,
Parents_Dead
                       num__pca4                      Sports, Gender,
Co_Curricular
                       num__pca5                    Sports, Religion,
Co_Curricular
                       num__pca6                            Sports,
PHQ_1, Religion
                       num__pca7                              Sports,
Gender, PHQ_7
```

```
                    num__pca8                              Gender,
PHQ_3, PHQ_5
                    num__pca9                             PHQ_3,
Religion, PHQ_1
                   num__pca10                              Gender,
PHQ_8, PHQ_4
                   num__pca11                        PHQ_5, Sports,
Co_Curricular
                   num__pca12                               PHQ_5,
Gender, PHQ_4
                   num__pca13                               PHQ_5,
GAD_4, PHQ_8
                   num__pca14        GAD_6, PHQ_3,
Percieved_Academic_Abilities
                   num__pca15                               PHQ_5,
GAD_6, PHQ_2
                   num__pca16                              Gender,
GAD_4, PHQ_7
                   num__pca17        Percieved_Academic_Abilities,
GAD_7, GAD_6
           cat__Boarding_day_Day
Boarding = day_Day
cat__Boarding_day_Day & Boarding                      Boarding =
day_Day & Boarding
    cat__School_type_Extracounty                          School =
type_Extracounty
      cat__School_type_Subcounty                            School =
type_Subcounty
  cat__School_Demographics_Girls                        School =
Demographics_Girls
  cat__School_Demographics_Mixed                        School =
Demographics_Mixed
     cat__School_County_Machakos                           School =
County_Machakos
      cat__School_County_Makueni                            School =
County_Makueni
      cat__School_County_Nairobi                            School =
County_Nairobi
```

When we trained our models, we processed the original dataset through a preprocessing pipeline that normalized numeric features, then compressed numeric features, and used one-hot encoding. Therefore, when the models were built, they were not able to identify the original categorical names such as Gender or School_Type but used technical naming conventions such as cat__School_Type_Extracounty or num__age. The use of the code was to reverse the transformation of the feature names, so when looking at summary or SHAP plots, we could remember what inputs contributed to predictions by the models. The summary above helps us understand what contributed to a model making a certain prediction.

5.3 Next steps for the model

1. **Launch and monitor**

- The aim is to create a screening tool, deploy a prototype and observe results in real life to see where the model is struggling.

1. **Ethical consideration**

- Make sure the system is used ethically especially for more severe predictions because false negtives could result to serious consequences.

1. **Increase data collection**

- Significantly increase the sampling for more severe levels of depression and anxiety in order to enhance generalization of the model.

6. Recommendations

- Give priority to improving recall.

In mental health detection, it is more valuable to catch all cases of severe distress and suffer a little more reduction in accuracy. As a result, fine-tuning in the future should focus on maximizing recall associated with high-severity levels.

- Collect more data on cases of severe distress.

One of the issues with the modeling is that there are very few cases of severe distress in the dataset. By obtaining more examples of users with severe symptoms will create exposure such as these cases will help create better patterns for the models.

- Deploy all four models simultaneously

Performance varies by severity level and type of symptoms, but by deploying all four models and using the best output for each user will add reliability. Overall, this is more of an ensemble-style approach that ensures the results are not driven by failure of accuracy from a single model.

- Deploy this model as an early mental health screening tool in Kenyan schools, to identify at risk students and know which factors increase the risk and which reduce the risk.
- Collect more data from all geographical locations in Kenya.
- The tool should not be used for diagnosis.

7. Deployment

```
import os

files_needed = ['trained_pipelines.pkl', 'model_metrics.pkl',
'best_model.txt']
print("Checking files for deployment:\n")
for f in files_needed:
    exists = os.path.exists(f)
    size = os.path.getsize(f) / (1024*1024) if exists else 0
    status = "YES" if exists else "NO"
```

```python
    print(f"{status} {f}: {size:.2f} MB" if exists else f"{status} {f}: NOT FOUND")
```

```
Checking files for deployment:

YES trained_pipelines.pkl: 104.40 MB
YES model_metrics.pkl: 0.00 MB
YES best_model.txt: 0.00 MB
```