CSAI-450 | Machine learning | Fall 2024

Project 25%

Due Date 7/12/2024

Prof. Lobna Nassar AB Nassar

Name	**ID**	**Contribution**
Anas Msimir	2021004954	Report/Code/dataset collection
Muhammad Mbarak	2022005351	Code/Report/dataset manager
Ahmed Hammoudeh	2021004915	PowerPoint/Code
Ahmed Muriebesh	2021005098	PowerPoint/Code
Marwan Elsayed	2021005042	PowerPoint/Code

Table of Contents

INTRODUCTION

In the world of academia, where the pursuit of knowledge often comes with immense pressure, university students are increasingly grappling with stress. According to research more than 60% of students experience stress. (E. Kerr & C. Claybourn, 2023). With the inspirations from (Ramli, anju, & arya, 2024) as our primary literature paper and **(**de Filippis & Al Foysal, 2024) as our secondary paper, in this project we focused on analysing and training learning models to predict student stress with machine learning. This will enable us to get better insight of stress in university students. We then continued to test the models with dataset collected in our own university (AURAK) to get localized results of our models.

DATASET DESCRIPTION

The dataset used in this project is a dataset from an open source Kaggle. This dataset is the same dataset that was used by both our literature papers. The data was collected by a survey conducted in Dharan, Nepal in 2022 at Tribhuvan university. The dataset contains 1100 rows and 21 numerical columns. The features are selected scientifically considering 5 major factors, they are Psychological, Physiological, Social, Environmental, and Academic Factors:

Psychological Factors:

'anxiety level' : range from 0 to 27 measured by using a self-diagnostic questioner tool "GAD-7"

'self-esteem': range from 0 to 30 measured by "Rosenberg Self Esteem Scale"

'Mental health history': either 0 for no mental health history or 1 for had mental health before

'Depression': range from 0 to 27 measured by "PHQ-9"

Physiological Factors: 'headache', 'blood pressure', 'sleep quality', 'breathing problem

Environmental Factors: 'noise level', 'living conditions', 'safety', 'basic needs',

Academic Factors: 'academic performance', 'study load', 'teacher student relationship' and 'future career concerns'.

Social Factor: 'social support', 'peer pressure', 'extracurricular activities' and 'bullying'.

For the ranges of the other columns consider the following diagram

```
the range in column anxiety_level is [ 0-21 ]
the range in column self_esteem is [ 0-30 ]
the range in column mental_health_history is [ 0-1 ]
the range in column depression is [ 0-27 ]
the range in column headache is [ 0-5 ]
the range in column blood_pressure is [ 1-3 ]
the range in column sleep_quality is [ 0-5 ]
the range in column breathing_problem is [ 0-5 ]
the range in column noise_level is [ 0-5 ]
the range in column living_conditions is [ 0-5 ]
the range in column safety is [ 0-5 ]
the range in column basic_needs is [ 0-5 ]
the range in column academic_performance is [ 0-5 ]
the range in column study_load is [ 0-5 ]
the range in column teacher_student_relationship is [ 0-5 ]
the range in column future_career_concerns is [ 0-5 ]
the range in column social_support is [ 0-3 ]
the range in column peer_pressure is [ 0-5 ]
the range in column extracurricular_activities is [ 0-5 ]
the range in column bullying is [ 0-5 ]
the range in column stress_level is [ 0-2 ]
```

EDA AND DATA PREPROCESSING

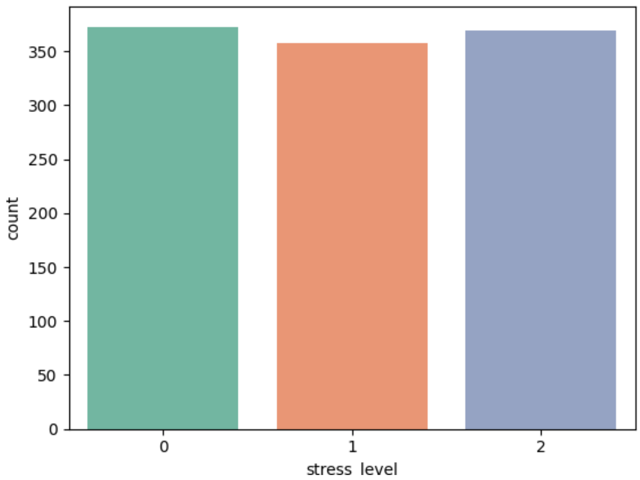
1. Data Cleaning

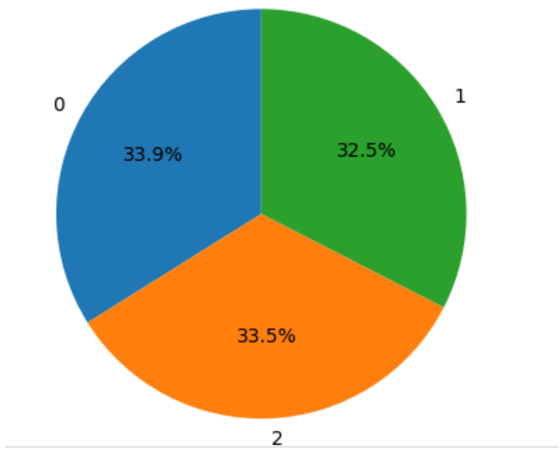
The data did not have any null values in any of its columns nor did it have any duplicated values. The dataset also did not contain any outliers in any of its column for this reason no further data cleaning was required.

2. Exploratory Data Analysis (EDA)

Distribution of The Target

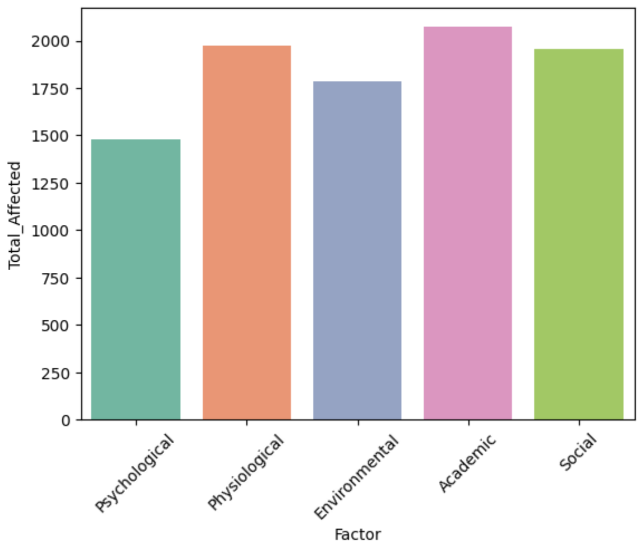
The target has three main classes (0: not stressed), (1: moderate stressed) and (2: severely stressed). All the classes in the target have a similar number of appearances with approximately 350 appearances. This resulted in a balance distribution among the classes which is ideal for model training as it ensures no bias toward any specific class. Consider the following figure



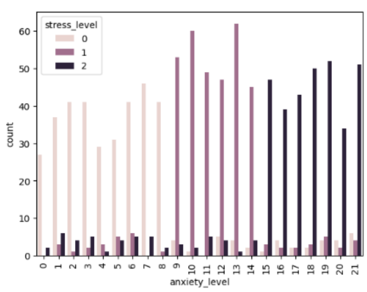


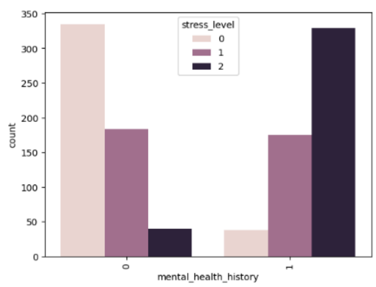
Distribution of features

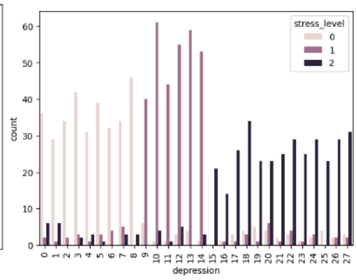
We began by examining how different categories of the features affect the target. We gave every feature a threshold where any row with value more than a threshold means the person is affected by that feature. For example, for anxiety the threshold is 10 which means any person with anxiety > 10 is suffering with anxiety. We then filtered out the rows with less than threshold and visualize the feature categories with respect to the target where target > 0. (stressed). The results were as follow:

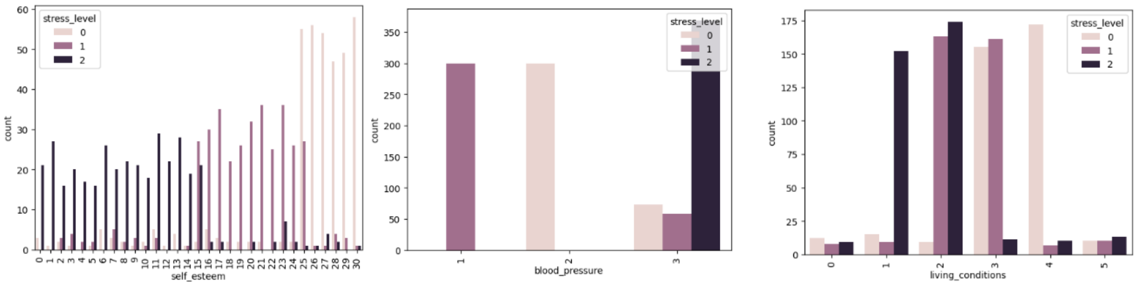


The distribution of each feature was then visualized individually to better understand their relationships with stress levels. Analysis of these plots revealed that for most features, higher values corresponded to an increase in stress. However, exceptions were observed for features such as **self-esteem, sleep quality, living conditions, safety, basic needs, academic performance,** and **social support**, where higher values were associated with a decrease in stress. An interesting trend was found for living condition. The stress decrease with increase in living condition until we reach 4 as for 5 (best living condition) shows equal number of stressed and unstress people Consider the following plots:



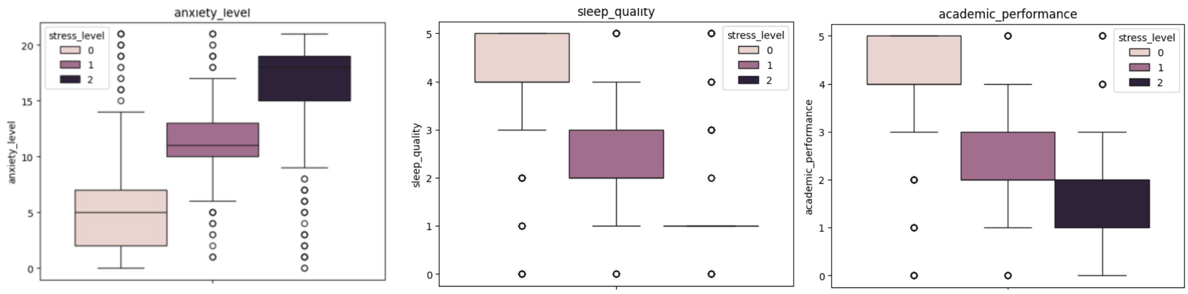






For the rest of the plots visit project code.

The trends were best visualized with box plots. Showing the inequalities and the range clearly. Consider box plots of some selected columns:

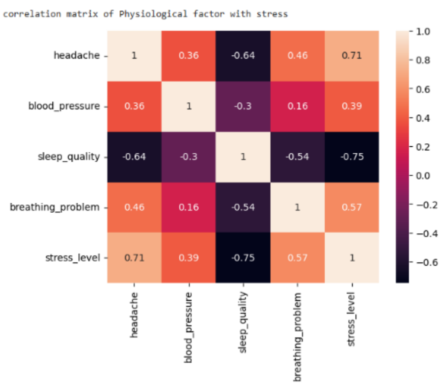


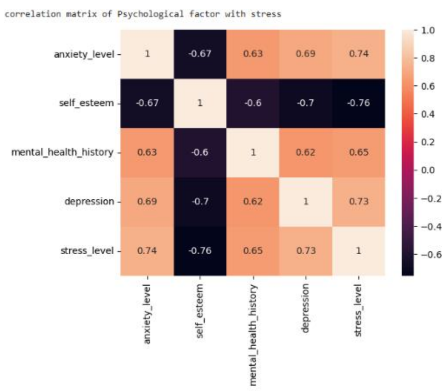
For the rest of the plots visit project code.

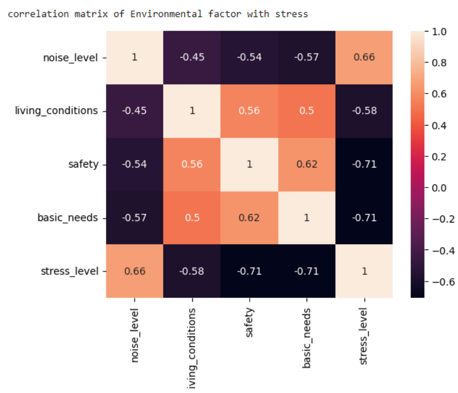
Correlation

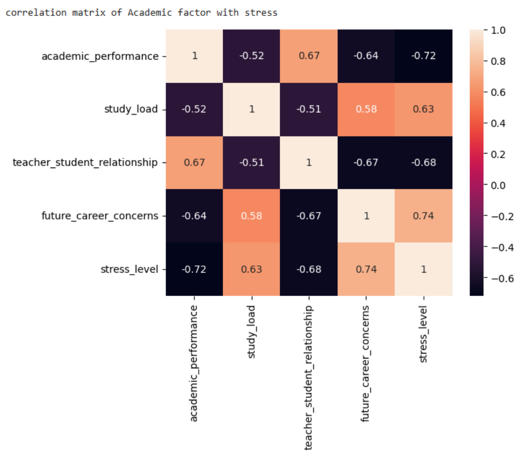
For every category of the features a corelation was visualised with a heatmap. The plots shows that cumulatively phycological factors have the highest corelation with stress. This implies that the stress among students is highly related with psychological factors such as low self-esteem, high anxiety and depression. Other features that high stress is corelated with are, bad sleep quality, frequent headache, low safety, basic needs, bad academic performance, future career concerns, peer pressure and bullying. The visualization highlighted some interesting relationships between features such as a high negative corelation between sleep and headache which implies that lack of sleep is highly associated with headaches. Also, self-esteem is highly negatively corelated with anxiety and depression which also implies that low self esteem is highly related to high anxiety and depression.

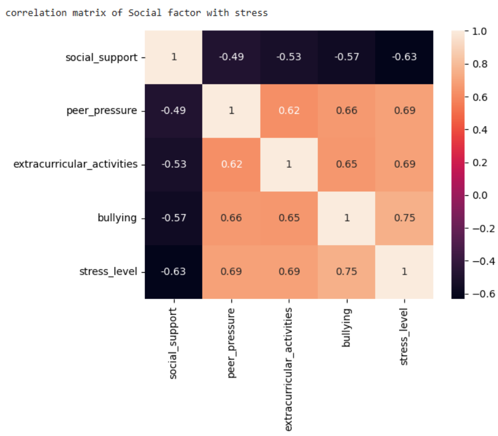
On visualizing a corelation of all features we see that self-esteem is negatively correlated with future carer concerns. And sleep quality is negatively corelated with depression and anxiety. Consider the following plots.











Descriptive statistics

The descriptive statistics such as mean, quintiles and standard deviation was calculated using the describe() function. Almost all the columns were normally distributed with the mean in the middle. The descriptive statistics are shown below:

	anxiety_level	self_esteem	mental_health_history	depression	headache	blood_pressure	sleep_quality	breathing_problem	noise_level	living_conditions	safety	basic_needs
count	1100.000000	1100.000000	1100.000000	1100.000000	1100.000000	1100.000000	1100.000000	1100.000000	1100.000000	1100.000000	1100.000000	1100.000000
mean	11.063636	17.777273	0.492727	12.555455	2.508182	2.181818	2.660000	2.753636	2.649091	2.518182	2.737273	2.772727
std	6.117558	8.944599	0.500175	7.727008	1.409356	0.833575	1.548383	1.400713	1.328127	1.119208	1.406171	1.433761
min	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	6.000000	11.000000	0.000000	6.000000	1.000000	1.000000	1.000000	2.000000	2.000000	2.000000	2.000000	2.000000
50%	11.000000	19.000000	0.000000	12.000000	3.000000	2.000000	2.500000	3.000000	3.000000	2.000000	2.000000	3.000000
75%	16.000000	26.000000	1.000000	19.000000	3.000000	3.000000	4.000000	4.000000	3.000000	3.000000	4.000000	4.000000
max	21.000000	30.000000	1.000000	27.000000	5.000000	3.000000	5.000000	5.000000	5.000000	5.000000	5.000000	5.000000

basic_needs	academic_performance	study_load	teacher_student_relationship	future_career_concerns	social_support	peer_pressure	extracurricular_activities	bullying	stress_level
1100.000000	1100.000000	1100.000000	1100.000000	1100.000000	1100.000000	1100.000000	1100.000000	1100.000000	1100.000000
2.772727	2.772727	2.621818	2.648182	2.649091	1.881818	2.734545	2.767273	2.617273	0.996364
1.433761	1.414594	1.315781	1.384579	1.529375	1.047826	1.425265	1.417562	1.530958	0.821673
0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
2.000000	2.000000	2.000000	2.000000	1.000000	1.000000	2.000000	2.000000	1.000000	0.000000
3.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.500000	3.000000	1.000000
4.000000	4.000000	3.000000	4.000000	4.000000	3.000000	4.000000	4.000000	4.000000	2.000000
5.000000	5.000000	5.000000	5.000000	5.000000	3.000000	5.000000	5.000000	5.000000	2.000000

3. Pre-Processing

To ensure there is no bias the attributes were scaled by using standard scalar. This makes the mean 0 and the standard deviation equal to 1. This makes all the features have equal weights when training the models. We also used one hot encoding for the target when training multi-layer perceptron. Additionally, we converted the features and target into tensor formats to meet with PyTorch requirements, since the framework works with tensor-based inputs only.

TRAINING

1. Parameters Fine Tuning

We optimized the performance of different algorithms by tuning their hyperparameters. For K-Nearest Neighbours (KNN) model we used grid search cross validation for all parameters with five folds to get the best number of neighbours. We used a predefined function GridSearchCV() from sklearn with cv =5 and scoring = accuracy. For random classifier we used standard grid search to get best maximum depth of each tree which resulted in the best performance. This approach enabled us to improve the accuracy of the models.

2. Model Training

Once the necessary preprocessing was completed, we proceeded to train our machine learning model. The target variable is categorical with three distinct classes, making it a multi-class classification problem. The following models were trained

- **Multi-Layer Perceptron**: for the MLP we used a neural network with three layers. An input layer, one hidden layer and the output layer. The input layer consisted of 20 neurons, corresponding to the number of features in the dataset, the hidden layer consisted of 12 neurons and three neurons for the output layer. The activation function used for all the layers was ReLU. We then trained for100 epochs to ensure best performance and convergence.
- **Ensemble Model**: Wanting to get better results, we implemented a hard voting ensemble model. A hard voting ensemble model takes the majority class voted by all the models. Three models were used for the ensemble. KNN classifier, Linear Support Vector Classifier and discission tree classifier. The three models were chosen to ensure diversity of the models for more generalization and better results. for the KNN we used the number of neighbours obtained by the grid search cross validation.
- **K-Nearest Neighbour**: We then trained the standard K-Nearest Neighbours (KNN) classifier using the optimal number of neighbours obtained from the Grid Search Cross-Validation. This ensured the best performance of the KNN model

- **Gradient Boosting:** For the gradient boosting we trained the model with 10 trees, 1 as our learning rate and 2 as the depth of each tree. These hyper parameters were chosen for fast convergence and to avoid overfitting
- **Random forest**: Last model we trained in random forest with 10 trees. The criterion used for splitting the nodes was cross-entropy and the depth for each tree was made to be the one obtained by the grid search.

3. Feature reduction

To get better performance, we focused on reducing noise and improving the quality of our model by applying feature reduction techniques. Two main methods of feature reduction techniques were utilized. Principal component analysis **PCA** and **lasso regularization**. We then trained the same models above and checked for the improvement. These two methods were utilized as follows:

- **PCA**: We used Principal Component Analysis (PCA) to capture 90% of the variance using fewer features. This reduced the feature space and simplified the dataset while retaining most information. We then replaced our training features with the new features obtained by PCA and continued to train the above models with these features.
- **Lasso:** For lasso we first used lasso cross validation to get the best value of the penalty term alpha that will result in better performance of our models. We then reduced our training features to only those that are most influential. The features that were not important (dropped) were mental health history, blood pressure breathing problem and teacher student relationship. We then continued to train the same models above with the reduced features.

4. Evaluation with comparison

All the model's accuracy were evaluated with cross validation by using 5 folds and get the mean of the cross validation. Other metrics that were used are confusion matrix and classification report which gives precision, f1 score and the recall of the models. For information about the metrics visit here. With the comparison of the performance metrics of the literature consider the table below:

Model	**Accuracy**	**Precision**	**F1**	**Recall**	**Accuracy**	**Precision**	**F1**	**Recall**
	Initial training (no reduction)				**Literature**			
MLP	0.90	0.90	0.90	0.90	0.90	0.92	0.92	0.93

Ensemble	0 .88	0.89	0.89	0.89	-	-	-	-
KNN	0.87	0.88	0.87	0.87	0.90	0.85	0.90	0.97
Gradient Boosting	0.86	0.89	0.90	0.90	0.88	0.88	0.90	0.92
Random Forest	0.86	0.89	0.90	0.90	0.88	-	-	-

Model	**Accuracy**	**Precision**	**F1**	**Recall**	**Accuracy**	**Precision**	**F1**	**Recall**
	PCA				**LASSO**			
KNN	0.88	0.89	0.89	0.89	0.89	0.90	0.90	0.90
Gradient Boosting	0.88	0.90	0.89	0.89	0.87	0.88	0.88	0.88
Random Forest	0.86	0.90	0.89	0.89	0.86	0.88	0.88	0.88

The literatures did not do PCA, LASSO and Ensemble model. Overall, The models did not show any significant difference with each other nor with the literatures.

5.Evaluation with AURAK Students Dataset

After training our model we conducted a survey in our university and check how our model will perform to the American University of Ras-al-Khaimah students. The response obtained from the students were 57. The results of the models were as follow

Model	**Accuracy**	**Precision**	**F1**	**Recall**
Ensemble	0.56	0.19	0.32	0.24
KNN	0.42	0.46	0.44	0.62
Gradient Boosting	0.56	0.49	0.45	0.56
Random Forest	0.40	0.43	0.41	0.46

This result might be caused by the time of the survey, everyone had projects and was preparing for exams which might cause everyone to be stressed no matter what features they have.

Reference (IEEE Style) :

S. Arya, A. Anju, and N. A. Ramli, "Predicting the Stress Level of Students Using Supervised Machine Learning and Artificial Neural Network (ANN)," Indian Journal of Engineering, vol. 21, 2024, pp. e9ije1684.

R. de Filippis and A. Al Foysal, "Comprehensive Analysis of Stress Factors Affecting Students: A Machine Learning Approach," Discover Artificial Intelligence, vol. 4, 2024, pp. 62. DOI: https://doi.org/10.1007/s44163-024-00169-6.

C.Acharya, "Student Stress Factors: A Comprehensive Analysis," Kaggle, 2023.

E. Kerr and C. Claybourn, "Stress in College Students: What to Know," U.S. News & World Report, Aug. 14, 2023.

T. Srivastava, "Complete Guide to Machine Learning Evaluation Metrics," *Medium*, Oct. 21, 2024.

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
from imblearn.over_sampling import SMOTE
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.metrics import
accuracy_score,precision_score,f1_score,recall_score,
confusion_matrix,classification_report,roc_curve, auc
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import VotingClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV,cross_val_score
from sklearn.decomposition import PCA
from sklearn.linear_model import Lasso, LassoCV
from sklearn.ensemble import GradientBoostingClassifier ,
RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
import torch
from torch.utils.data import TensorDataset, DataLoader
import torch.nn as nn
import torch.nn.functional as F
from tqdm import tqdm
from imblearn.over_sampling import SMOTE

df = pd.read_csv('/content/StressLevelDataset.csv')
df.head()
```

{"type":"dataframe","variable_name":"df"}

```
def cleaning(df):
    df.drop('Timestamp', axis=1, inplace =True)
    mapping = {
        'Not at all': 0,
        'Several days': 1,
        'More than half the days': 2,
        'Nearly every day': 3
    }
    for col in df.iloc[:, :7]:
      df[col] = df[col].map(mapping)

    df['anxiety_level'] = df.iloc[:, :7].sum(axis=1)
    df['self_esteem'] = df.iloc[:, 7:17].sum(axis=1)
    for col in df.iloc[:, 18:27]:
      df[col] = df[col].map(mapping)
```

```
    df['depression'] = df.iloc[:, 18:27].sum(axis=1)
    df.drop(columns=df.columns[:7], inplace=True)
    df.drop(columns=df.columns[:10], inplace=True)
    df.drop(columns=df.columns[1:10], inplace=True)
    col = df.pop('anxiety_level')
    df.insert(0, 'anxiety_level', col)
    col = df.pop('self_esteem')
    df.insert(1, 'self_esteem', col)
    col = df.pop('depression')
    df.insert(3, 'depression', col)
    df['mental health problems history'] = df['mental health problems
history'].map( {
        'never experienced mental health problem before': 0,
        'experienced mental health problem before': 1})
    df['Blood Pressure'] = df['Blood Pressure'].map( {
        'low': 0,
        'mid': 1,
        'high': 2,
        "I don't know": 1})
    df['How frequent you experience short and heavy breathing'] =
df['How frequent you experience short and heavy
breathing'].map(mapping)
    df['Do you have social support to cope with mental health?'] =
df['Do you have social support to cope with mental health?'].map( {
        'No': 0,
        'Maybe': 1,
        'Yes': 2})
    df.rename(columns={
        'mental health problems history': 'mental_health_history',
        'headache frequency':'headache',
        'Blood Pressure':'blood_pressure',
        'Sleep quality':'sleep_quality',
        'How frequent you experience short and heavy
breathing':'breathing_problem',
        'How loud is the place you live?':'noise_level',
        'How good is your living condition?':'living_conditions',
        'How safe do you feel overall?':'safety',
        'do you get all your basic needs?':'basic_needs',
        'how is our academic performance':'academic_performance',
        'how is your study load':'study_load',
        'how is your relationship with
teachers':'teacher_student_relationship',
        'how are you concerned with your future career
':'future_career_concerns',
        'Do you have social support to cope with mental
health?':'social_support',
        'Do you experience peer pressure?':'peer_pressure',
        'Do you engage yourself in extracurricular
activities?':'extracurricular_activities',
```

```
        'Do you experience bullying?':'bullying',
        'how stressed are you?':'stress_level',
    }, inplace=True)
    return df


# data obtained by survey done in aurak.
Aurak_df = pd.read_csv('/content/stress measure form(Responses) - Form
responses 1 (1).csv')
Aurak_df = cleaning(Aurak_df)
Aurak_df.head()
```

{"type":"dataframe","variable_name":"Aurak_df"}

```
Aurak_df.dropna(inplace=True)

Aurak_df = Aurak_df.astype(np.int64)

Aurak_df_x = Aurak_df.iloc[:,:-1]
Aurak_df_y = Aurak_df.iloc[:,-1]
```

EDA

```
df.shape
```

```
(1100, 21)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1100 entries, 0 to 1099
Data columns (total 21 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   anxiety_level                 1100 non-null   int64
 1   self_esteem                   1100 non-null   int64
 2   mental_health_history         1100 non-null   int64
 3   depression                    1100 non-null   int64
 4   headache                      1100 non-null   int64
 5   blood_pressure                1100 non-null   int64
 6   sleep_quality                 1100 non-null   int64
 7   breathing_problem             1100 non-null   int64
 8   noise_level                   1100 non-null   int64
 9   living_conditions             1100 non-null   int64
 10  safety                        1100 non-null   int64
 11  basic_needs                   1100 non-null   int64
 12  academic_performance          1100 non-null   int64
 13  study_load                    1100 non-null   int64
 14  teacher_student_relationship  1100 non-null   int64
 15  future_career_concerns        1100 non-null   int64
 16  social_support                1100 non-null   int64
```

```
 17   peer_pressure                     1100 non-null   int64
 18   extracurricular_activities        1100 non-null   int64
 19   bullying                          1100 non-null   int64
 20   stress_level                      1100 non-null   int64
dtypes: int64(21)
memory usage: 180.6 KB
```

```
pd.set_option('display.max_columns', None)
```

```
df.describe()
```

{"type":"dataframe"}

```
df.isna().sum()
```

```
anxiety_level                   0
self_esteem                     0
mental_health_history           0
depression                      0
headache                        0
blood_pressure                  0
sleep_quality                   0
breathing_problem               0
noise_level                     0
living_conditions               0
safety                          0
basic_needs                     0
academic_performance            0
study_load                      0
teacher_student_relationship    0
future_career_concerns          0
social_support                  0
peer_pressure                   0
extracurricular_activities      0
bullying                        0
stress_level                    0
dtype: int64
```

```
df.duplicated().sum()
```

```
0
```

```
# checking the range of each column
for column in df.columns:
  print(f'the range in column {column} is [ {df[column].min()}-
{df[column].max()} ]')
```

```
the range in column anxiety_level is [ 0-21 ]
the range in column self_esteem is [ 0-30 ]
the range in column mental_health_history is [ 0-1 ]
the range in column depression is [ 0-27 ]
```

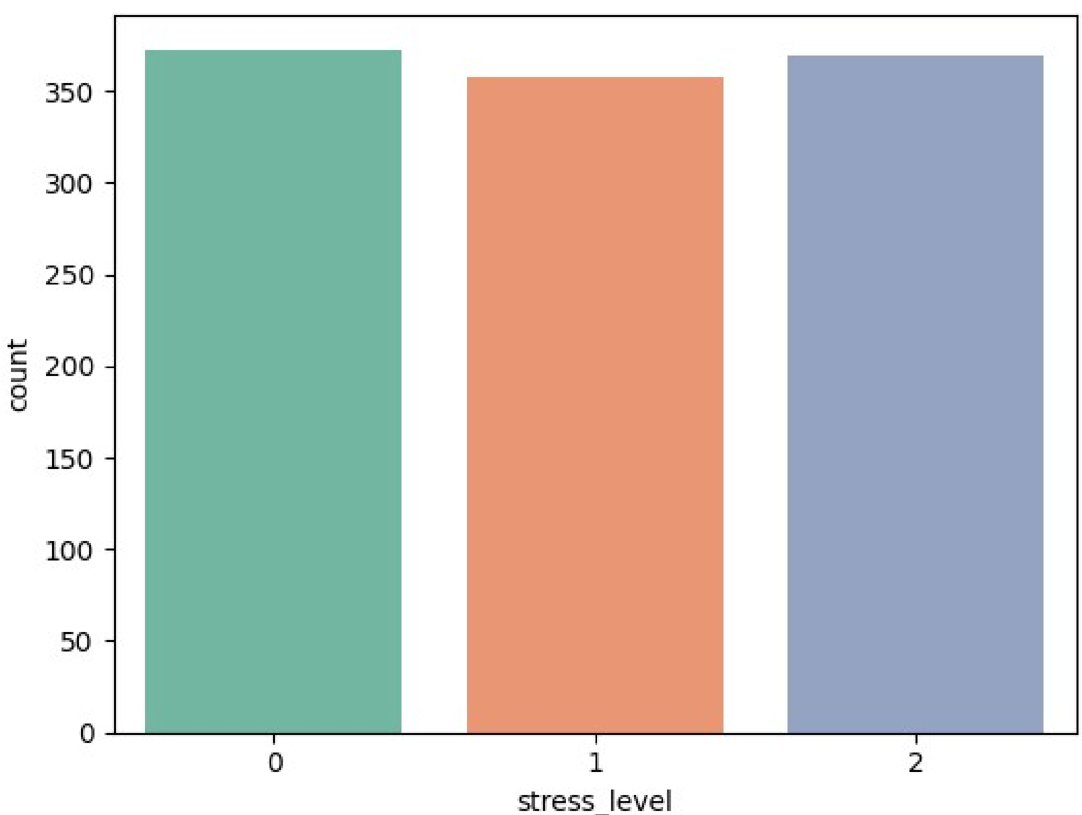
```
the range in column headache is [ 0-5 ]
the range in column blood_pressure is [ 1-3 ]
the range in column sleep_quality is [ 0-5 ]
the range in column breathing_problem is [ 0-5 ]
the range in column noise_level is [ 0-5 ]
the range in column living_conditions is [ 0-5 ]
the range in column safety is [ 0-5 ]
the range in column basic_needs is [ 0-5 ]
the range in column academic_performance is [ 0-5 ]
the range in column study_load is [ 0-5 ]
the range in column teacher_student_relationship is [ 0-5 ]
the range in column future_career_concerns is [ 0-5 ]
the range in column social_support is [ 0-3 ]
the range in column peer_pressure is [ 0-5 ]
the range in column extracurricular_activities is [ 0-5 ]
the range in column bullying is [ 0-5 ]
the range in column stress_level is [ 0-2 ]
```

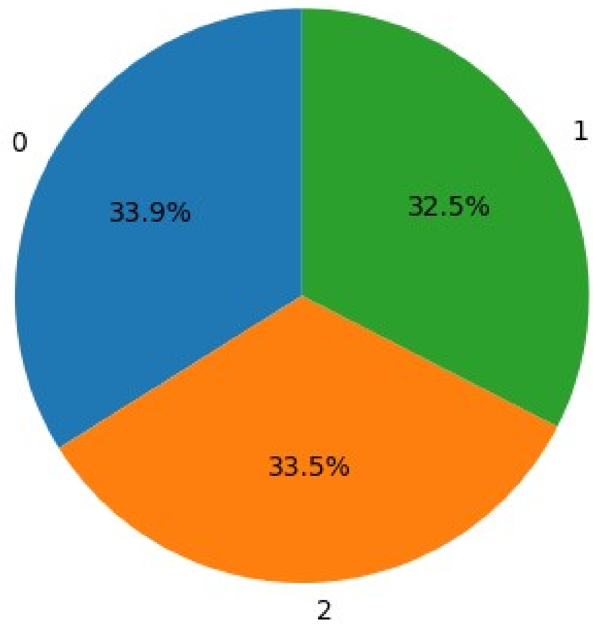
```
#checking unique values in each column
for column in df.columns:
  print(f'Unique values in {column}: {df[column].unique()}')
```

```
Unique values in anxiety_level: [14 15 12 16 20  4 17 13  6  5  9  2
11  7 21  3 18  0  8  1 19 10]
Unique values in self_esteem: [20  8 18 12 28 13 26  3 22 15 23 21 25
1 27  5  6  9 29 30  4 19 16  2
  0 14  7 17 24 11 10]
Unique values in mental_health_history: [0 1]
Unique values in depression: [11 15 14  7 21  6 22 12 27 25  8 24  3
1  0  5 26 20 10  9  2 16  4 13
 18 23 17 19]
Unique values in headache: [2 5 4 3 1 0]
Unique values in blood_pressure: [1 3 2]
Unique values in sleep_quality: [2 1 5 4 3 0]
Unique values in breathing_problem: [4 2 3 1 5 0]
Unique values in noise_level: [2 3 4 1 0 5]
Unique values in living_conditions: [3 1 2 4 5 0]
Unique values in safety: [3 2 4 1 5 0]
Unique values in basic_needs: [2 3 1 4 5 0]
Unique values in academic_performance: [3 1 2 4 5 0]
Unique values in study_load: [2 4 3 5 1 0]
Unique values in teacher_student_relationship: [3 1 2 4 5 0]
Unique values in future_career_concerns: [3 5 2 4 1 0]
Unique values in social_support: [2 1 3 0]
Unique values in peer_pressure: [3 4 5 2 1 0]
Unique values in extracurricular_activities: [3 5 2 4 0 1]
Unique values in bullying: [2 5 1 4 3 0]
Unique values in stress_level: [1 2 0]
```

```
#checking distribution of the target
print(df['stress_level'].value_counts())
sns.countplot(x = 'stress_level', data = df,palette='Set2' )
plt.show()

counts = df['stress_level'].value_counts()
plt.pie(counts, labels=counts.index, autopct='%1.1f%%', startangle=90)
plt.show()
```

```
stress_level
0    373
2    369
1    358
Name: count, dtype: int64

<ipython-input-118-145684a29e53>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.countplot(x = 'stress_level', data = df,palette='Set2' )
```






```python
# mean of every column for each stress level
average_stress = df.groupby('stress_level').mean()
average_stress
```

{"summary":"{\n \"name\": \"average_stress\",\n \"rows\": 3,\n \"fields\": [\n   {\n   \"column\": \"stress_level\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 1,\n   \"min\": 0,\n   \"max\": 2,\n   \"num_unique_values\": 3,\n   \"samples\": [\n   0,\n   1,\n   2\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"anxiety_level\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 5.492740689937741,\n   \"min\": 5.431635388739946,\n   \"max\": 16.40108401084011,\n   \"num_unique_values\": 3,\n   \"samples\": [\n   5.431635388739946,\n   11.430167597765363,\n   16.40108401084011\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"self_esteem\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 8.337250876783726,\n   \"min\": 8.78048780487805,\n   \"max\": 25.25201072386059,\n   \"num_unique_values\": 3,\n   \"samples\": [\n   25.25201072386059,\n   19.262569832402235,\n   8.78048780487805\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"mental_health_history\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.394887535182334,\n

```
\"min\": 0.10187667560321716,\n        \"max\": 0.8915989159891599,\n
\"num_unique_values\": 3,\n        \"samples\": [\n
0.10187667560321716,\n          0.4888268156424581,\n
0.8915989159891599\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"depression\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 6.934331127313059,\n        \"min\":
6.013404825737266,\n        \"max\": 19.829268292682926,\n
\"num_unique_values\": 3,\n        \"samples\": [\n
6.013404825737266,\n          11.874301675977653,\n
19.829268292682926\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"headache\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 1.2247232119152427,\n        \"min\":
1.3136729222520107,\n        \"max\": 3.761517615176152,\n
\"num_unique_values\": 3,\n        \"samples\": [\n
1.3136729222520107,\n          2.4608938547486034,\n
3.761517615176152\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"blood_pressure\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.8382146628897155,\n        \"min\":
1.324022346368715,\n        \"max\": 3.0,\n
\"num_unique_values\": 3,\n        \"samples\": [\n
2.195710455764075,\n          1.324022346368715,\n          3.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"sleep_quality\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1.4152355313218399,\n        \"min\": 1.3035230352303524,\n
\"max\": 4.126005361930295,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          4.126005361930295,\n
2.53072625698324,\n          1.3035230352303524\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"breathing_problem\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.988820000934072,\n        \"min\": 1.6970509383378016,\n
\"max\": 3.6531165311653115,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          1.6970509383378016,\n
2.9273743016759775,\n          3.6531165311653115\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"noise_level\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1.0794685433205051,\n        \"min\": 1.648793565683646,\n
\"max\": 3.794037940379404,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          1.648793565683646,\n
2.511173184357542,\n          3.794037940379404\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"living_conditions\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.7924101302089576,\n        \"min\": 1.7289972899728998,\n
```

```
\"max\": 3.3136729222520107,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          3.3136729222520107,\n
2.5027932960893855,\n          1.7289972899728998\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"safety\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
1.243451854239978,\n        \"min\": 1.6720867208672088,\n
\"max\": 4.099195710455764,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          4.099195710455764,\n
2.416201117318436,\n          1.6720867208672088\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"basic_needs\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1.2611103023916468,\n        \"min\": 1.6720867208672088,\n
\"max\": 4.144772117962466,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          4.144772117962466,\n
2.477653631284916,\n          1.6720867208672088\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"academic_performance\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1.262811970637741,\n        \"min\": 1.6612466124661247,\n
\"max\": 4.142091152815014,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          4.142091152815014,\n
2.4916201117318435,\n          1.6612466124661247\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"study_load\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1.0187878371766865,\n        \"min\": 1.6541554959785523,\n
\"max\": 3.6856368563685638,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          1.6541554959785523,\n
2.5335195530726256,\n          3.6856368563685638\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"teacher_student_relationship\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1.1713300208110873,\n        \"min\": 1.6368563685636857,\n
\"max\": 3.927613941018767,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          3.927613941018767,\n
2.357541899441341,\n          1.6368563685636857\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"future_career_concerns\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1.3871678330781454,\n        \"min\": 1.3351206434316354,\n
\"max\": 4.100271002710027,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          1.3351206434316354,\n
2.522346368715084,\n          4.100271002710027\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"social_support\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.847182513828681,\n        \"min\": 0.926829268292683,\n
```

```
\"max\": 2.5415549597855227,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          2.5415549597855227,\n
2.17877094972067,\n          0.926829268292683\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"peer_pressure\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1.222639476716062,\n        \"min\": 1.675603217158177,\n
\"max\": 4.073170731707317,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          1.675603217158177,\n
2.458100558659218,\n          4.073170731707317\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"extracurricular_activities\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1.221975714192686,\n        \"min\": 1.7158176943699732,\n
\"max\": 4.10840108401084,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          1.7158176943699732,\n
2.4804469273743015,\n          4.10840108401084\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"bullying\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
1.4009110628565609,\n        \"min\": 1.2546916890080428,\n
\"max\": 4.05420054200542,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          1.2546916890080428,\n
2.5558659217877095,\n          4.05420054200542\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"average_stress"}
```

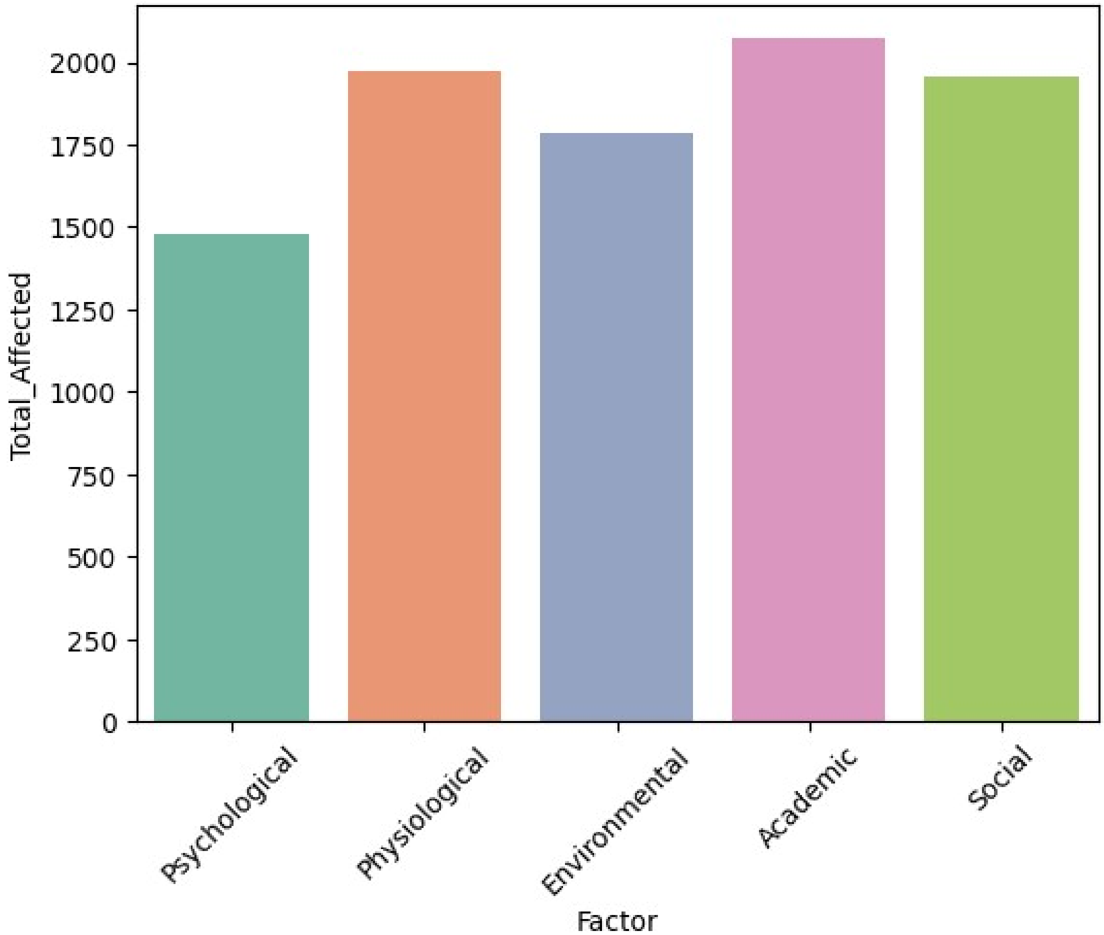

```
# see how people are affected with different categories of factors
#phsycological ,physiological, enviromental, acedemic, social
factors= {
  'Psychological': {'anxiety_level':10, 'self_esteem':-15,
'mental_health_history':0, 'depression':10},
  'Physiological': {'headache':2, 'blood_pressure':1,
'sleep_quality':-3, 'breathing_problem':2},
  'Environmental': {'noise_level':2, 'living_conditions':-3,
'safety':-3, 'basic_needs':2},
  'Academic': {'academic_performance':-3, 'study_load':2,
'teacher_student_relationship':-3, 'future_career_concerns':2},
  'Social': {'social_support':-2, 'peer_pressure':2,
'extracurricular_activities':2, 'bullying':2} }

affected={}
for factor,value in factors.items():
  affected[factor]=0
  for column,threshold in value.items():
      if threshold > 0:
        a= df[(df[column] > threshold) & (df['stress_level'] >
0)].count().iloc[1]
        affected[factor] += a
      else:
```

```
        a= df[(df[column] < -threshold) & (df['stress_level'] >
0)].count().iloc[1]
        affected[factor] += a
      print(f'column {column} play part in affecting {a} people with
stress')

print(affected)
```

```
column anxiety_level play part in affecting 554 people with stress
column self_esteem play part in affecting 348 people with stress
column mental_health_history play part in affecting 0 people with
stress
column depression play part in affecting 577 people with stress
column headache play part in affecting 515 people with stress
column blood_pressure play part in affecting 427 people with stress
column sleep_quality play part in affecting 516 people with stress
column breathing_problem play part in affecting 514 people with stress
column noise_level play part in affecting 528 people with stress
column living_conditions play part in affecting 515 people with stress
column safety play part in affecting 527 people with stress
column basic_needs play part in affecting 213 people with stress
column academic_performance play part in affecting 525 people with
stress
column study_load play part in affecting 509 people with stress
column teacher_student_relationship play part in affecting 532 people
with stress
column future_career_concerns play part in affecting 506 people with
stress
column social_support play part in affecting 427 people with stress
column peer_pressure play part in affecting 495 people with stress
column extracurricular_activities play part in affecting 510 people
with stress
column bullying play part in affecting 527 people with stress
{'Psychological': 1479, 'Physiological': 1972, 'Environmental': 1783,
'Academic': 2072, 'Social': 1959}
```

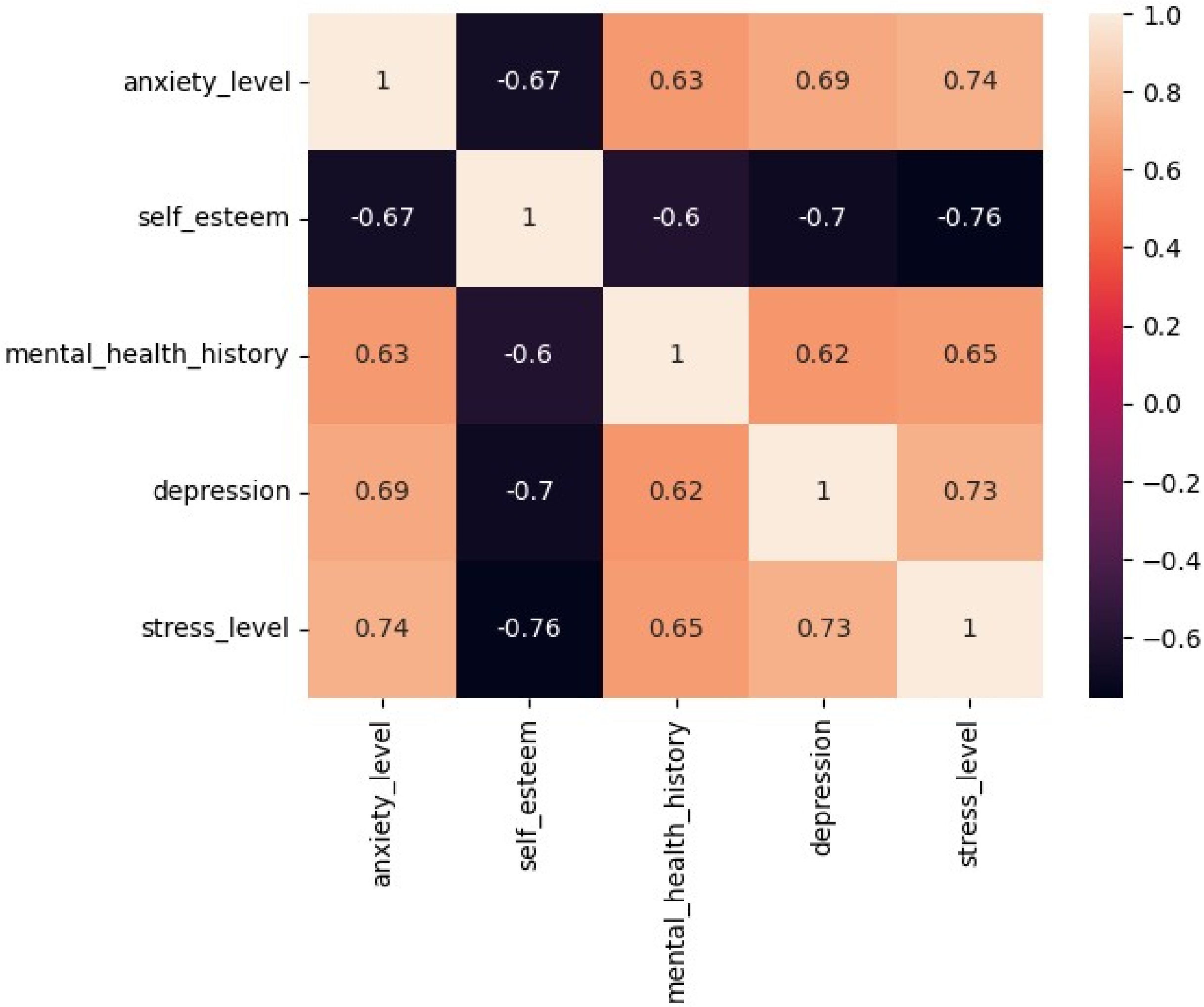
```
# visualizing the above
a = pd.DataFrame(list(affected.items()), columns=['Factor',
'Total_Affected'])
sns.barplot(x='Factor', y='Total_Affected', data=a, palette='Set2')
plt.xticks(rotation=45)
plt.show()
```

```
<ipython-input-121-0f5757f25347>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.barplot(x='Factor', y='Total_Affected', data=a, palette='Set2')
```

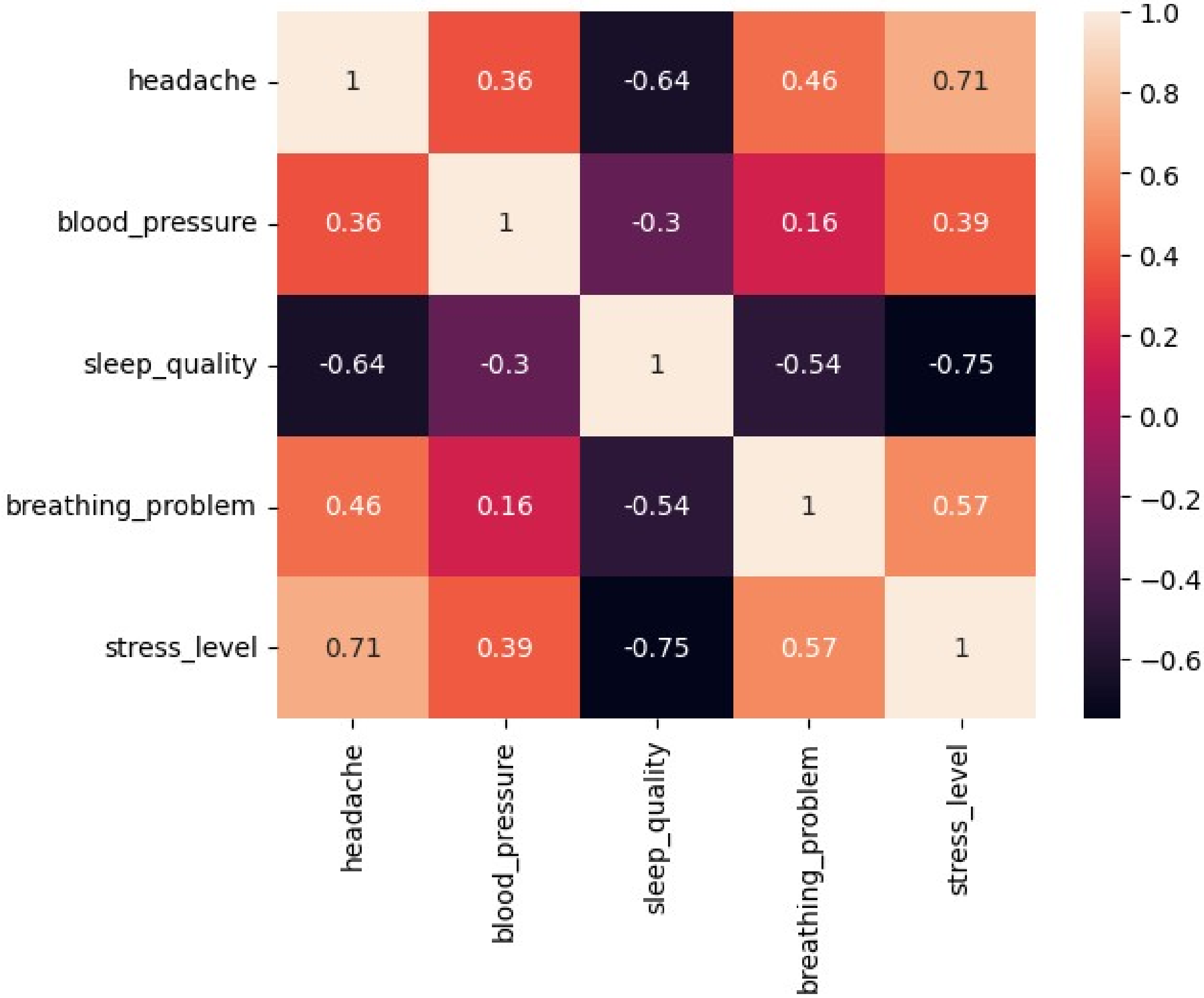



```
# checking the coorelation of each category of factor with stress
for factor,value in factors.items():
  print(f"correlation matrix of {factor} factor with stress")
  a= pd.DataFrame()
  for column,threshold in value.items():
    a[column] = df[column]
  a['stress_level'] = df['stress_level']
  sns.heatmap(a.corr(), annot=True)
  plt.show()
```

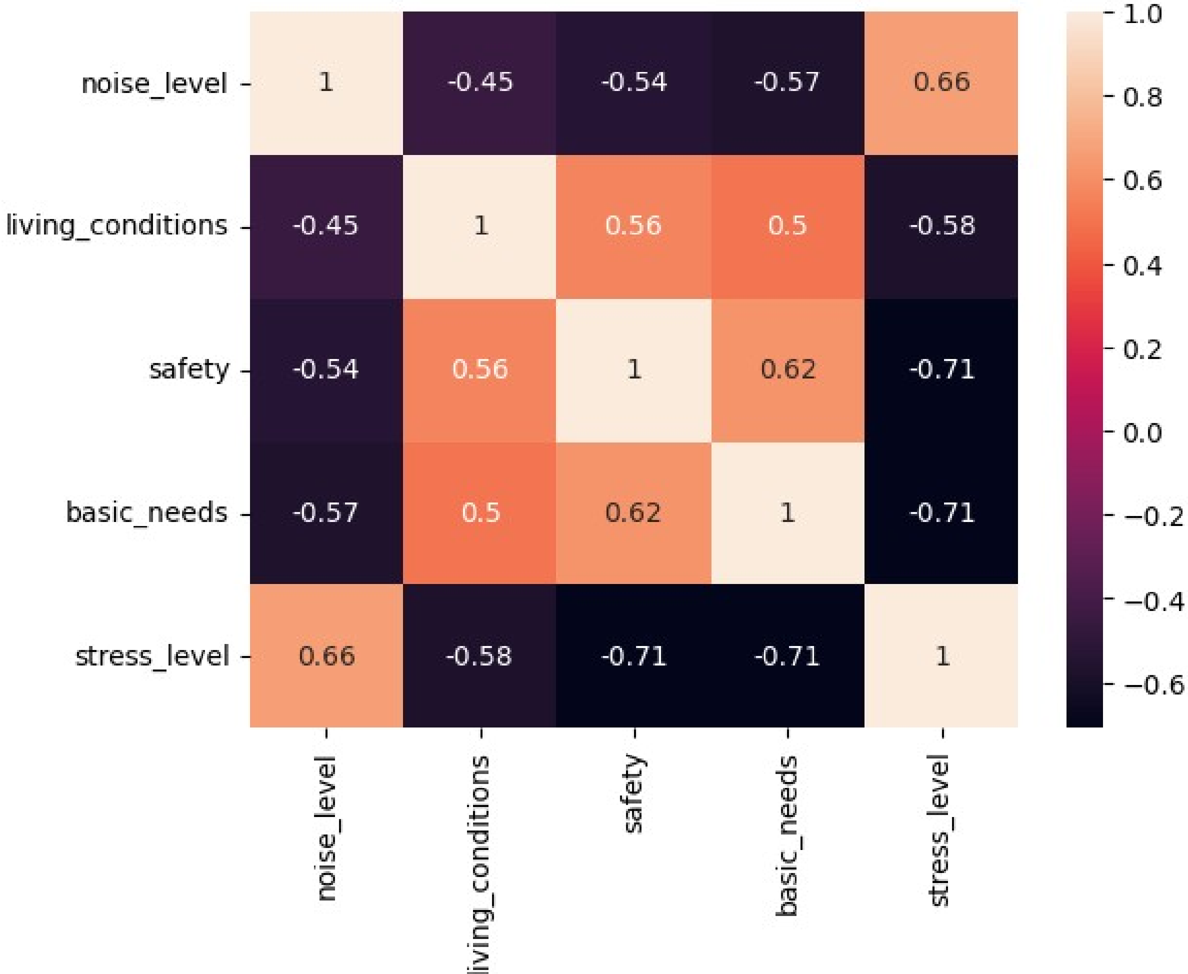
```
correlation matrix of Psychological factor with stress
```



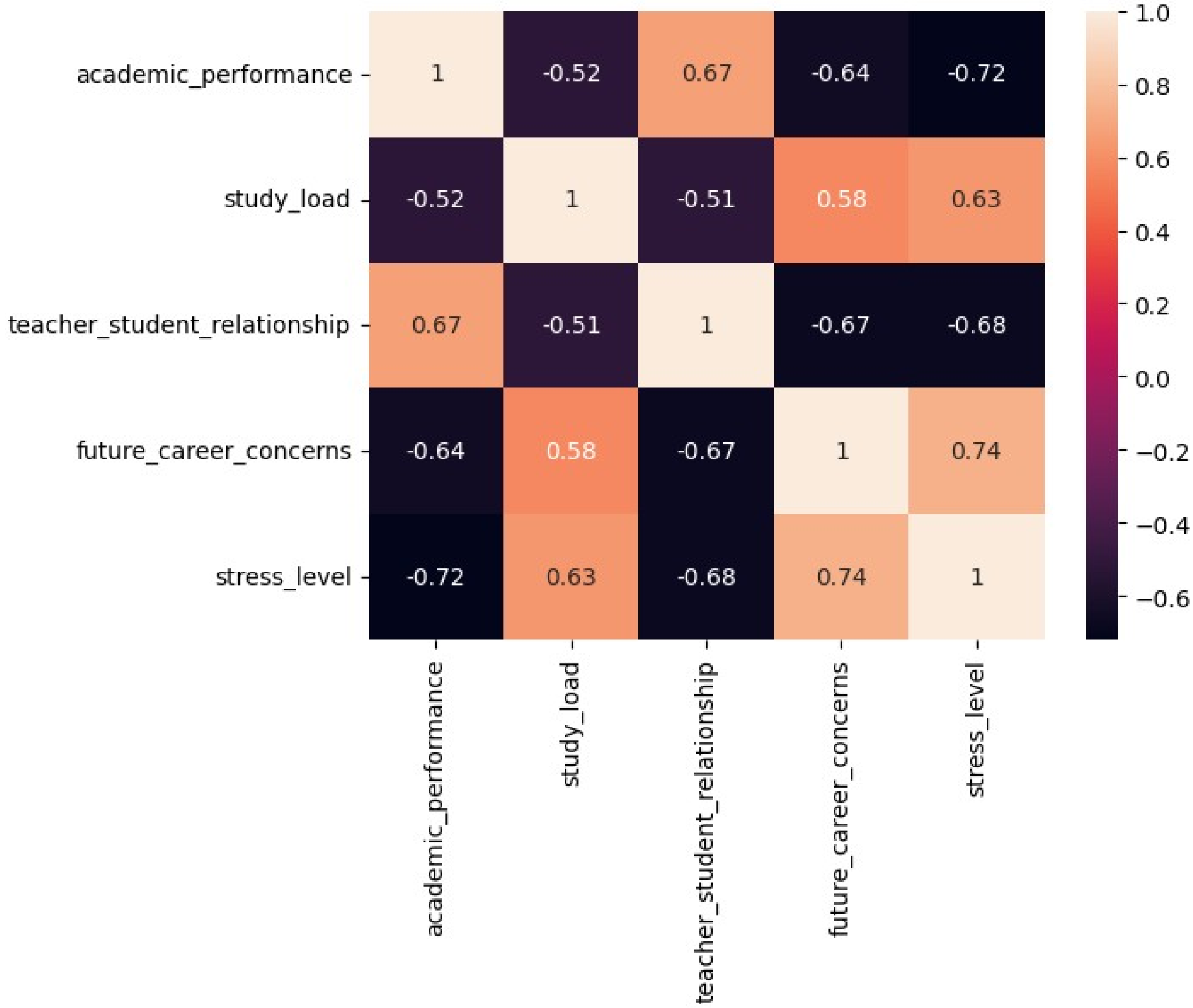

```
correlation matrix of Physiological factor with stress
```



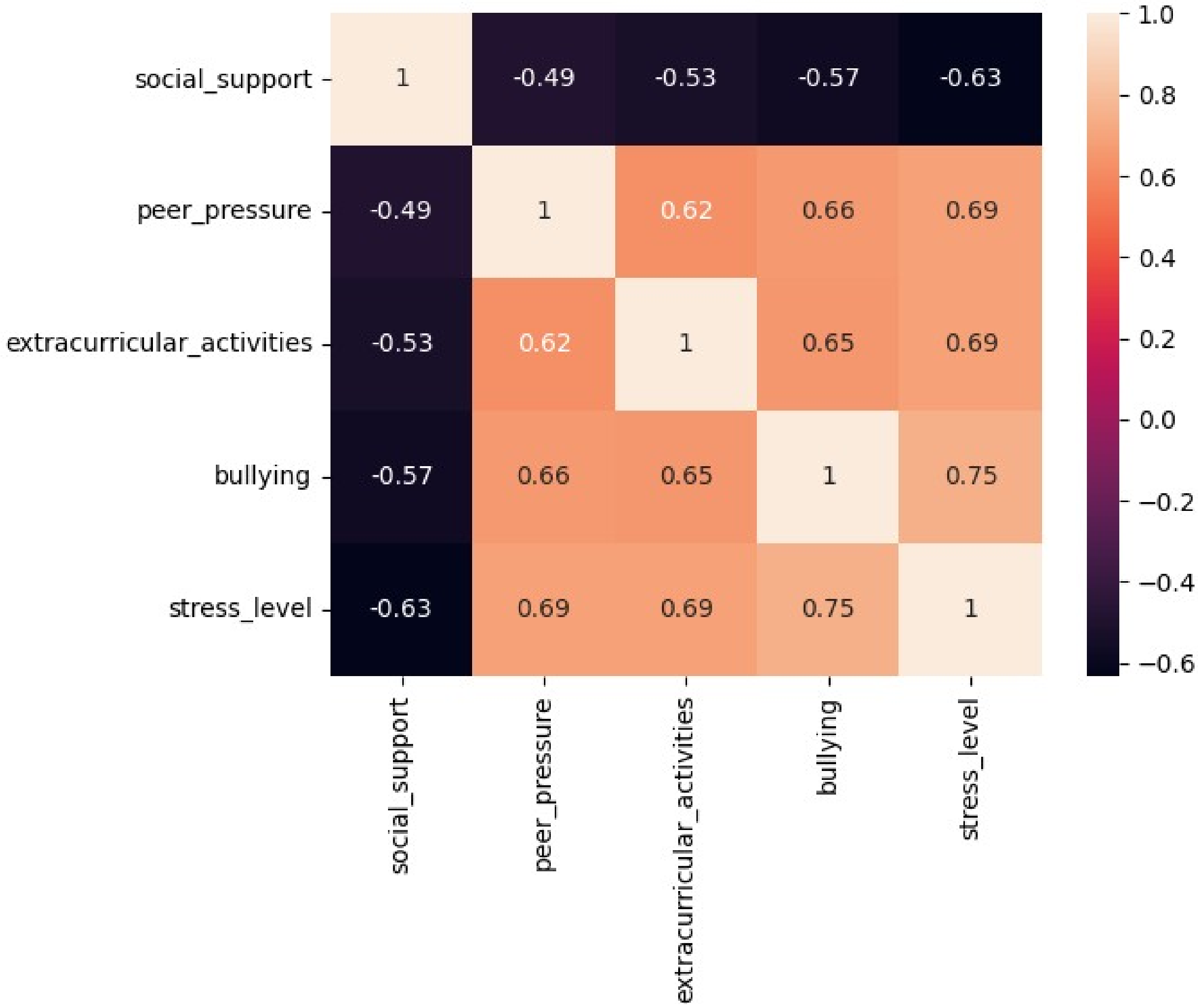

```
correlation matrix of Environmental factor with stress
```



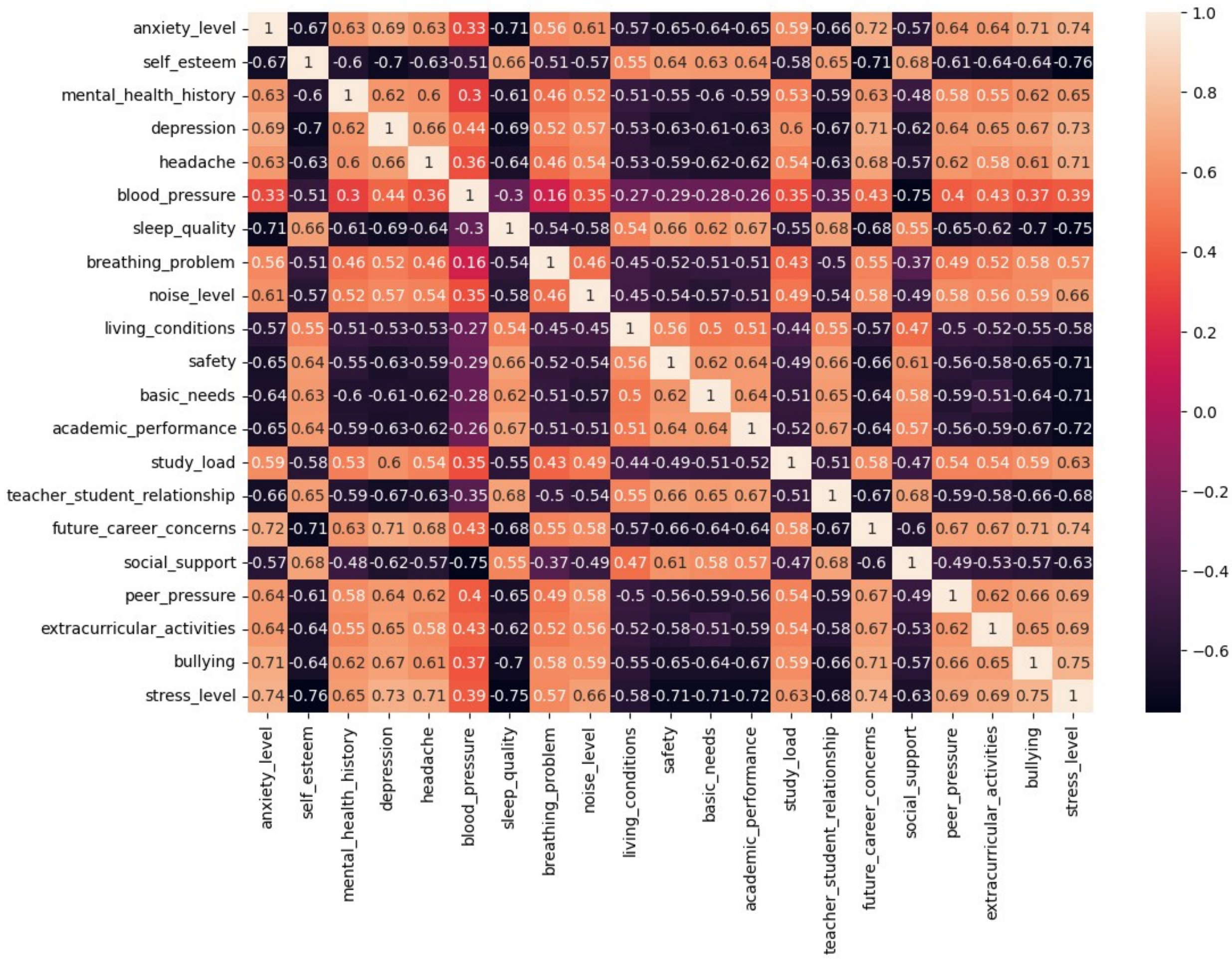

```
correlation matrix of Academic factor with stress
```




```
correlation matrix of Social factor with stress
```



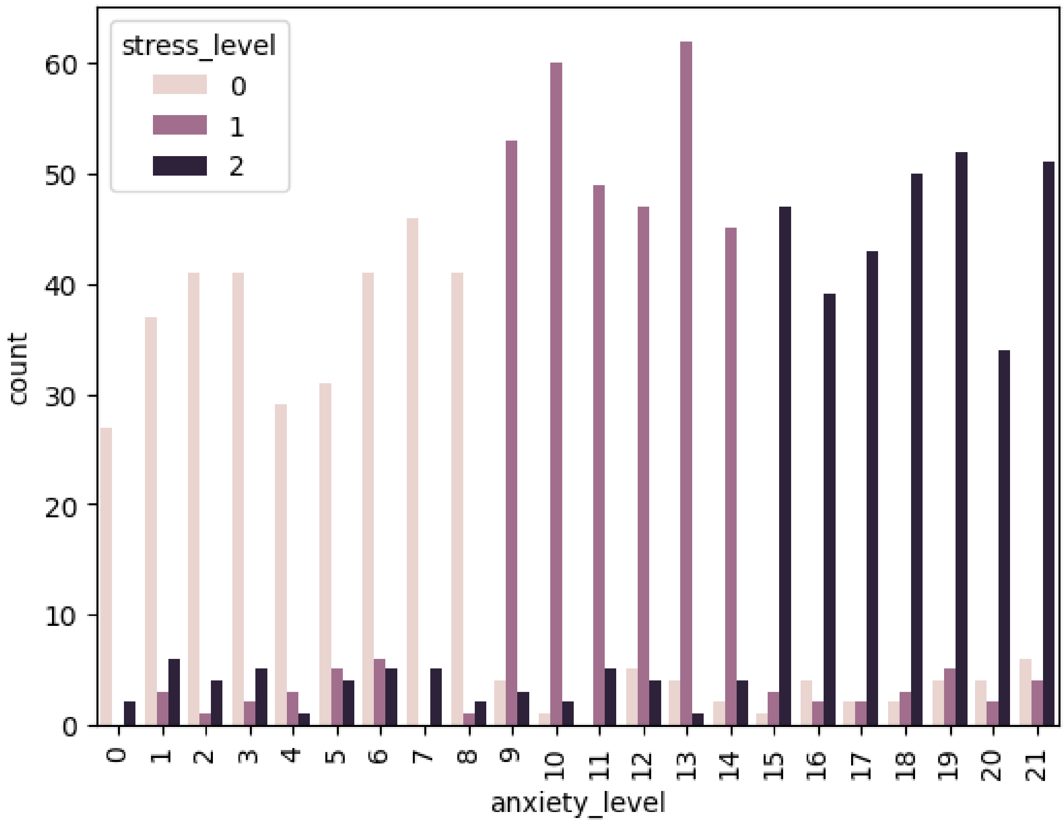

```
#corelation of everything to everything
plt.figure(figsize=(12, 8))
sns.heatmap(df.corr(), annot=True)
plt.show()
```



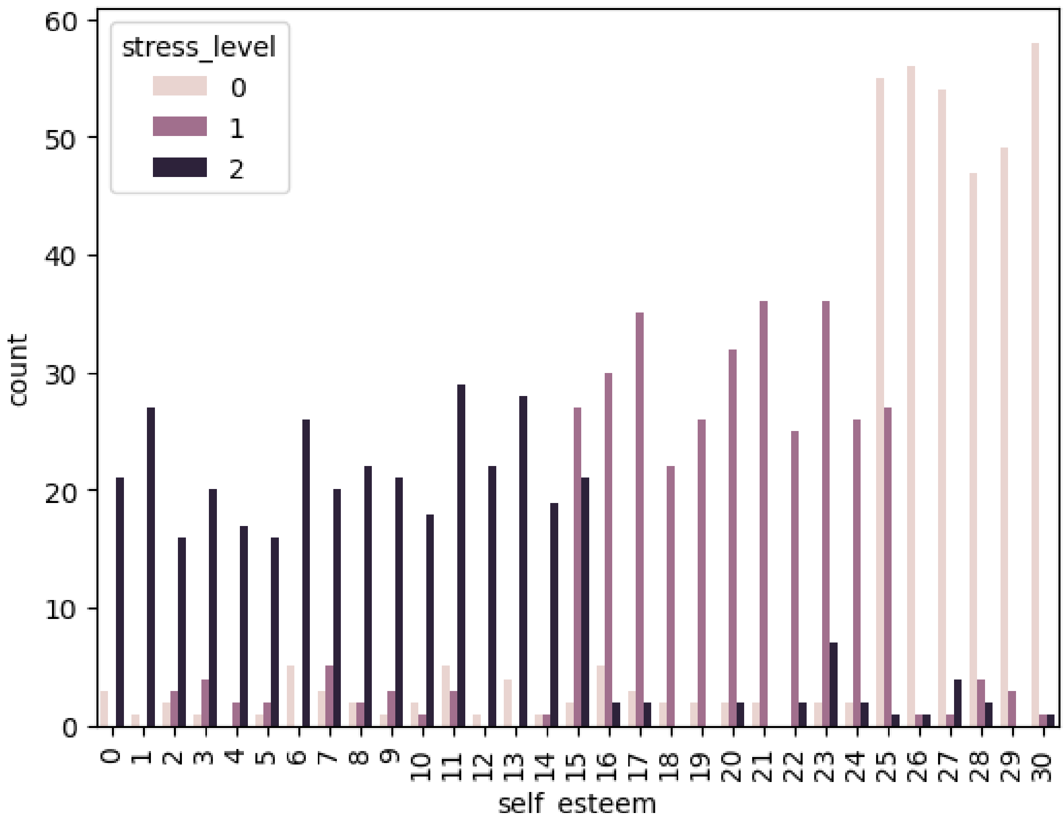
```
#how each column is distributed
for column in df.columns:
  print(f'column {column} is disrtibuted as follow')
  print(df.groupby('stress_level')[column].value_counts())
  sns.countplot(x =column,data = df, hue = 'stress_level')
  plt.xticks(rotation =90)
  plt.show()
```

```
column anxiety_level is disrtibuted as follow
stress_level  anxiety_level
0             7                46
              2                41
              3                41
              6                41
              8                41
                               ..
2             0                 2
              8                 2
              10                2
              4                 1
```

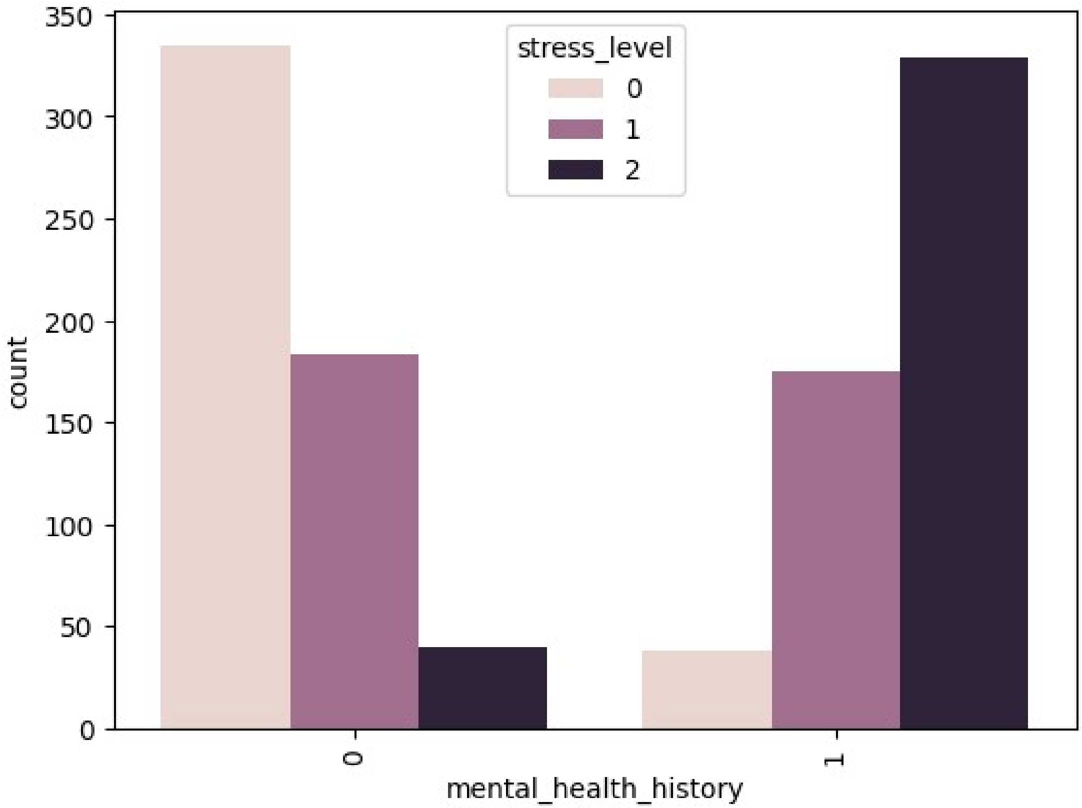
```
               13                  1
Name: count, Length: 63, dtype: int64
```



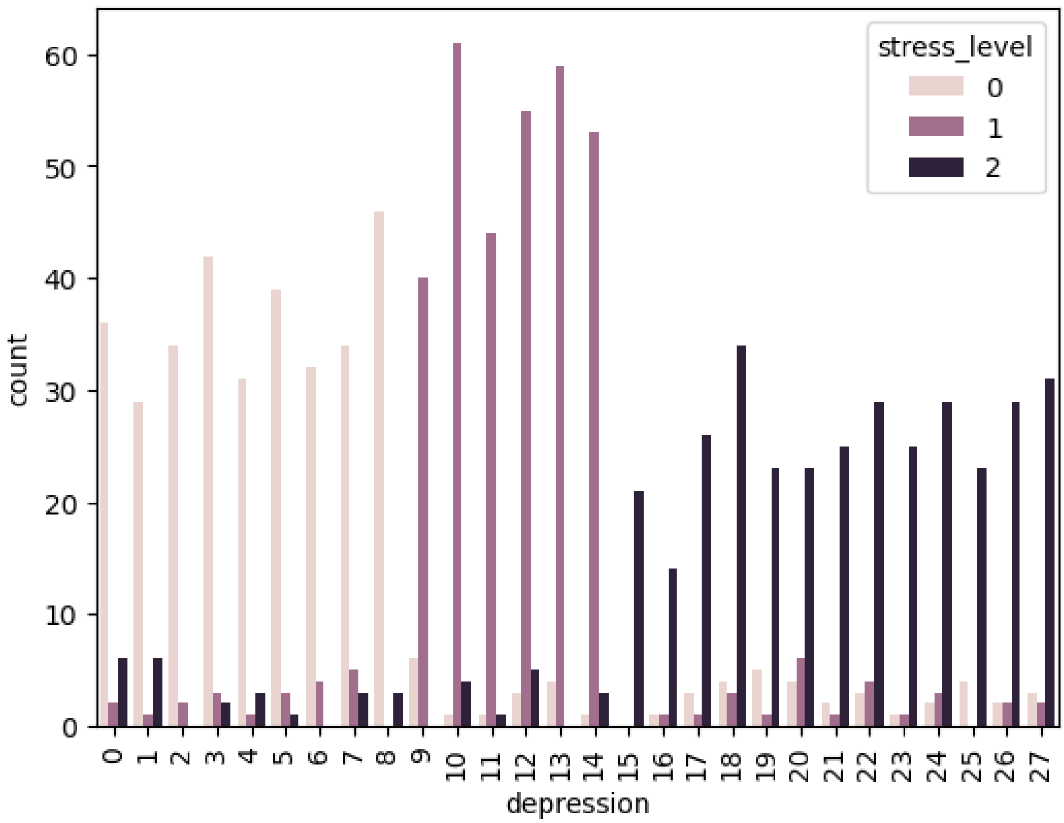

```
column self_esteem is disrtibuted as follow
stress_level  self_esteem
0             30             58
              26             56
              25             55
              27             54
              29             49
                             ..
2             24              2
              28              2
              25              1
              26              1
              30              1
Name: count, Length: 82, dtype: int64
```



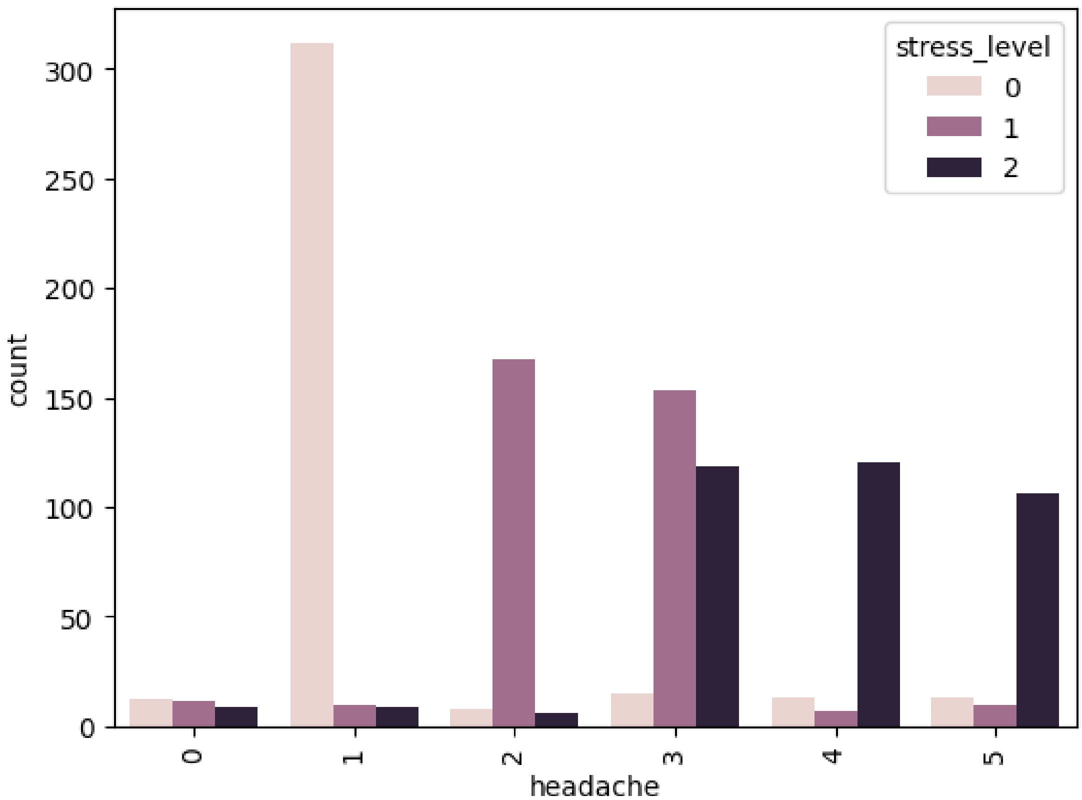

```
column mental_health_history is disrtibuted as follow
stress_level  mental_health_history
0             0                        335
              1                         38
1             0                        183
              1                        175
2             1                        329
              0                         40
Name: count, dtype: int64
```



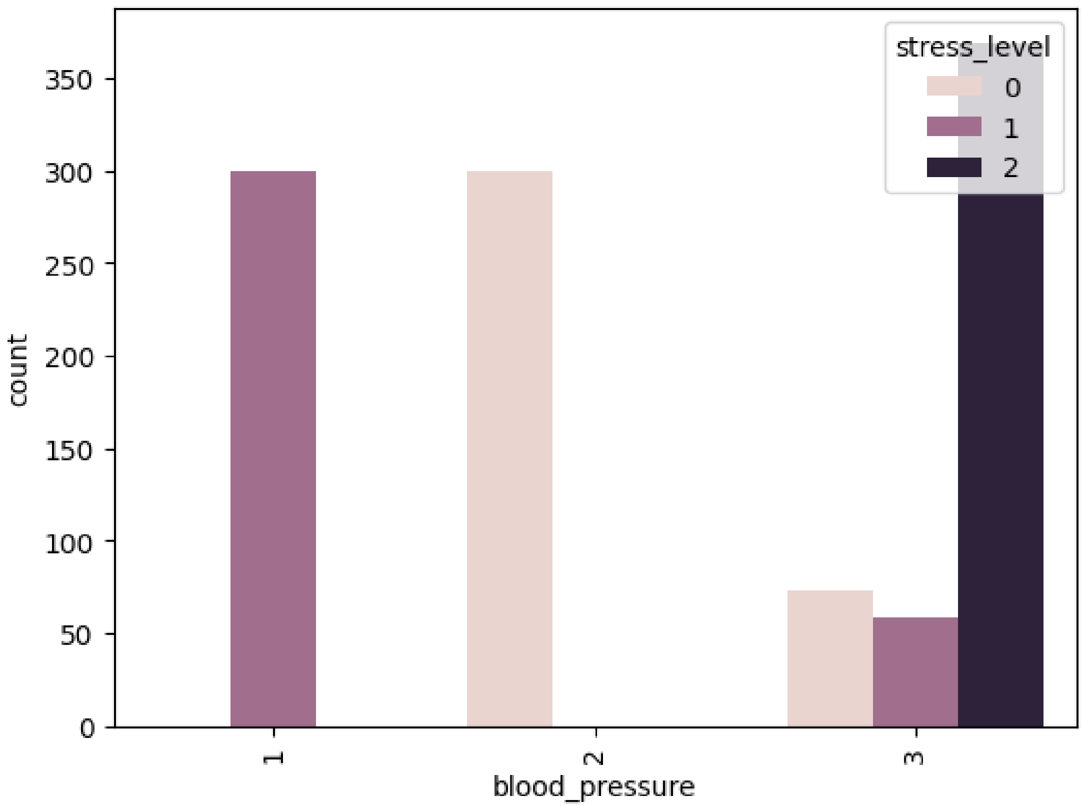

```
column depression is disrtibuted as follow
stress_level  depression
0             8             46
              3             42
              5             39
              0             36
              2             34
                            ..
2             8              3
              14             3
              3              2
              5              1
              11             1
Name: count, Length: 76, dtype: int64
```



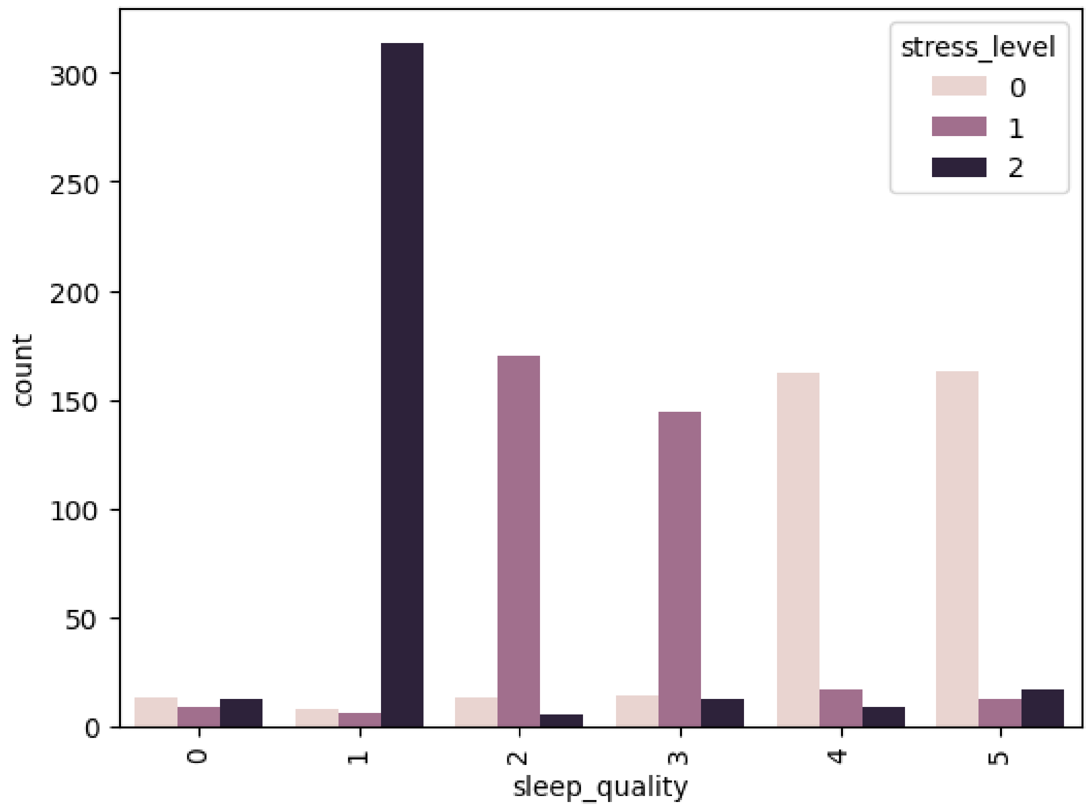

```
column headache is disrtibuted as follow
stress_level  headache
0             1           312
              3            15
              4            13
              5            13
              0            12
              2             8
1             2           167
              3           153
              0            11
              1            10
              5            10
              4             7
2             4           120
              3           119
              5           106
              0             9
              1             9
              2             6
Name: count, dtype: int64
```



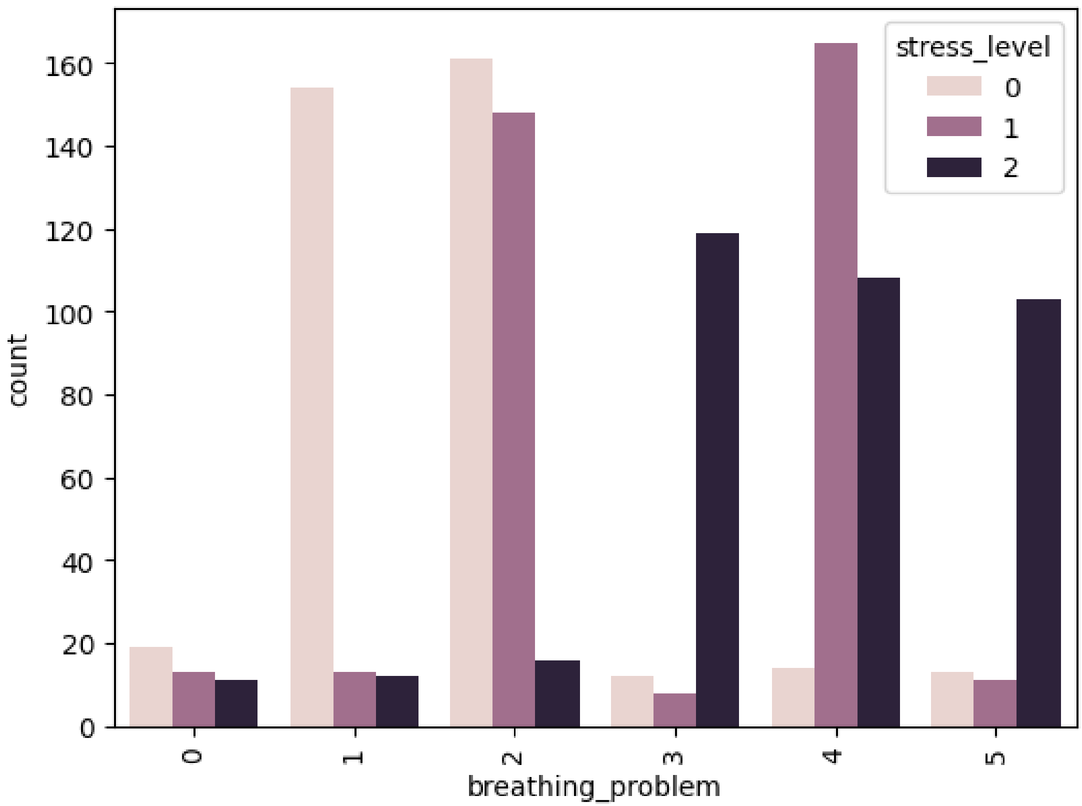

```
column blood_pressure is disrtibuted as follow
stress_level  blood_pressure
0             2                 300
              3                  73
1             1                 300
              3                  58
2             3                 369
Name: count, dtype: int64
```



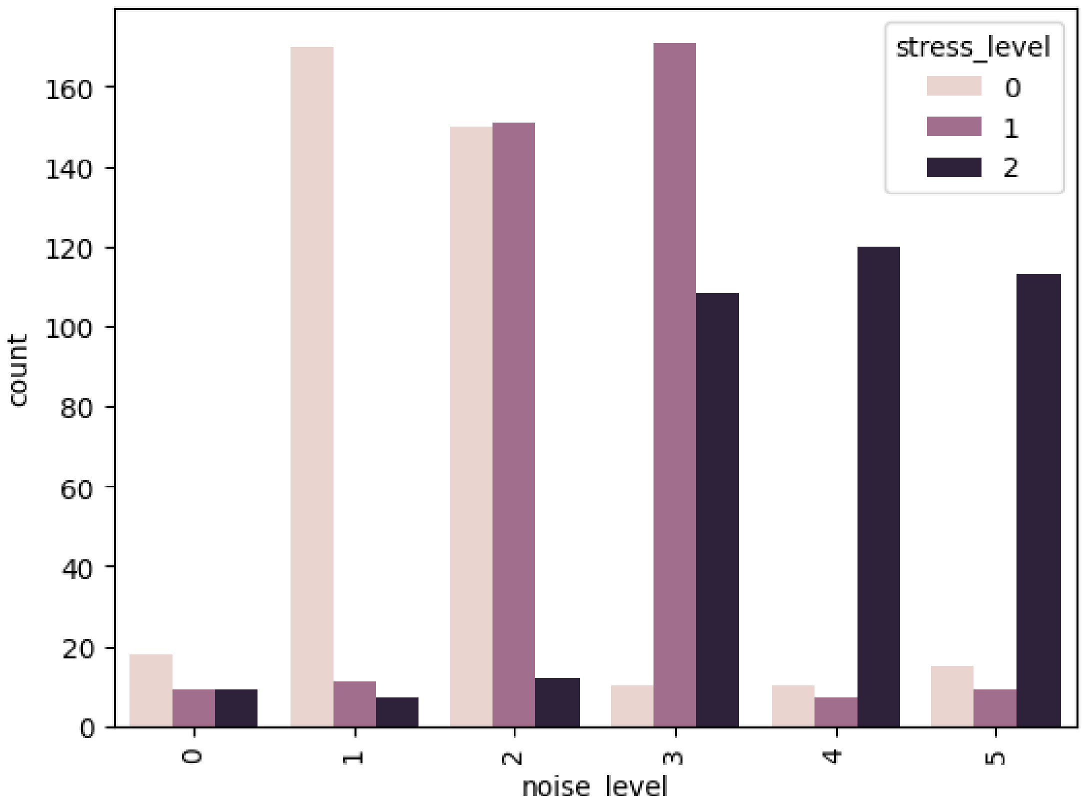

```
column sleep_quality is disrtibuted as follow
stress_level  sleep_quality
0             5                163
              4                162
              3                 14
              0                 13
              2                 13
              1                  8
1             2                170
              3                144
              4                 17
              5                 12
              0                  9
              1                  6
2             1                314
              5                 17
              0                 12
              3                 12
              4                  9
              2                  5
Name: count, dtype: int64
```



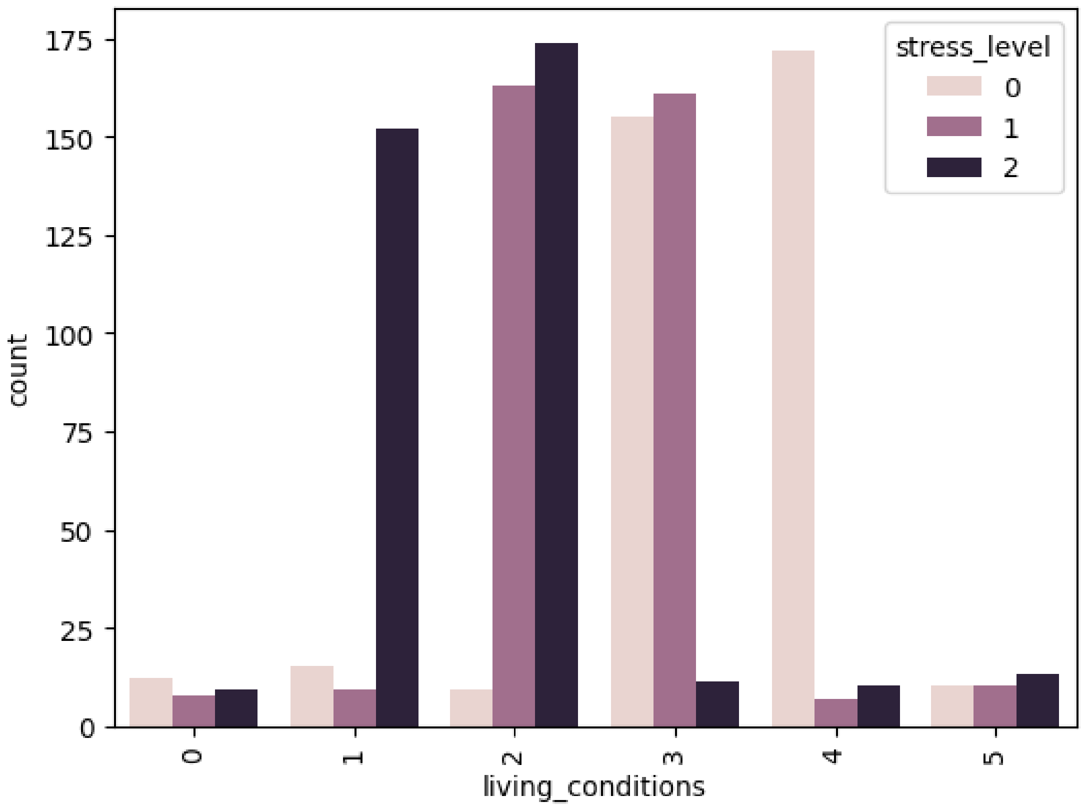

```
column breathing_problem is disrtibuted as follow
stress_level  breathing_problem
0             2                    161
              1                    154
              0                     19
              4                     14
              5                     13
              3                     12
1             4                    165
              2                    148
              0                     13
              1                     13
              5                     11
              3                      8
2             3                    119
              4                    108
              5                    103
              2                     16
              1                     12
              0                     11
Name: count, dtype: int64
```



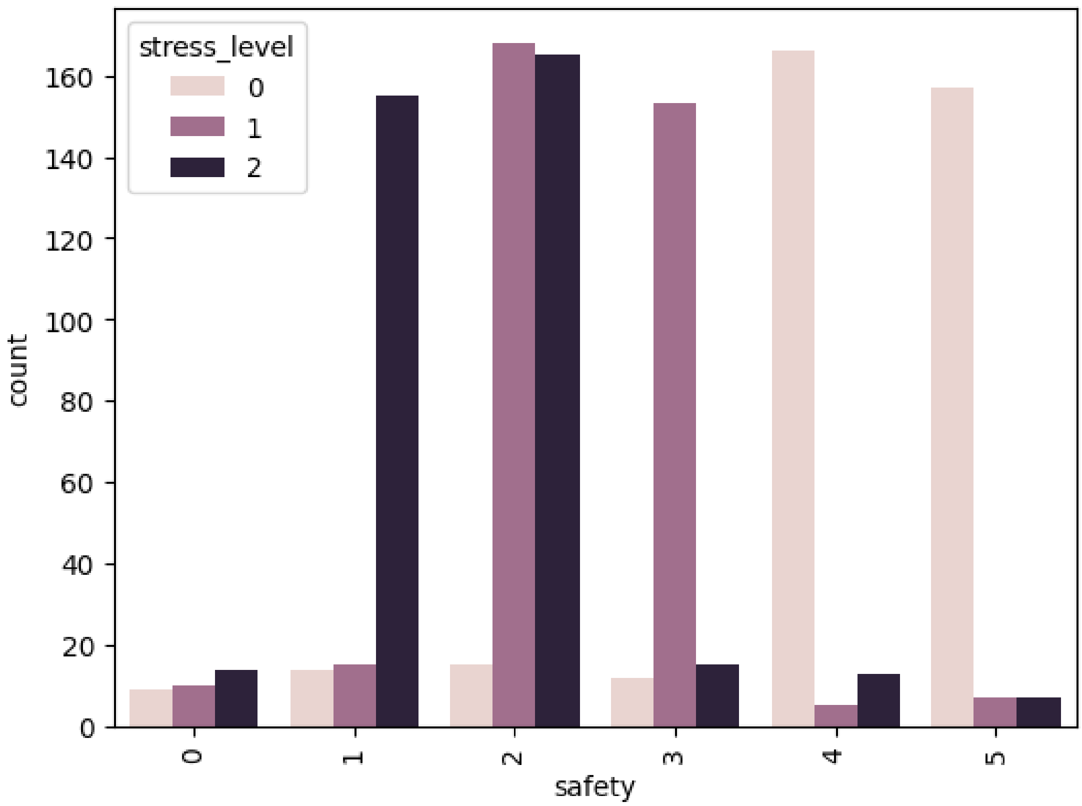

```
column noise_level is disrtibuted as follow
stress_level  noise_level
0             1              170
              2              150
              0               18
              5               15
              3               10
              4               10
1             3              171
              2              151
              1               11
              0                9
              5                9
              4                7
2             4              120
              5              113
              3              108
              2               12
              0                9
              1                7
Name: count, dtype: int64
```



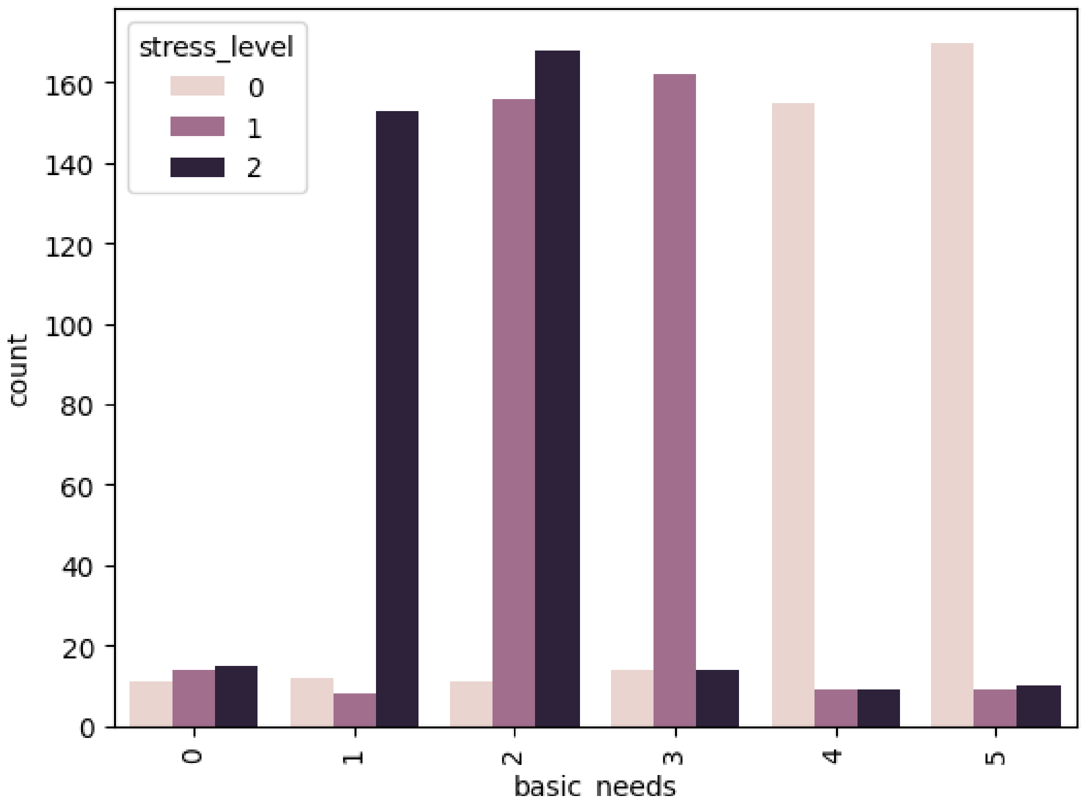

```
column living_conditions is disrtibuted as follow
stress_level  living_conditions
0             4                    172
              3                    155
              1                     15
              0                     12
              5                     10
              2                      9
1             2                    163
              3                    161
              5                     10
              1                      9
              0                      8
              4                      7
2             2                    174
              1                    152
              5                     13
              3                     11
              4                     10
              0                      9
Name: count, dtype: int64
```



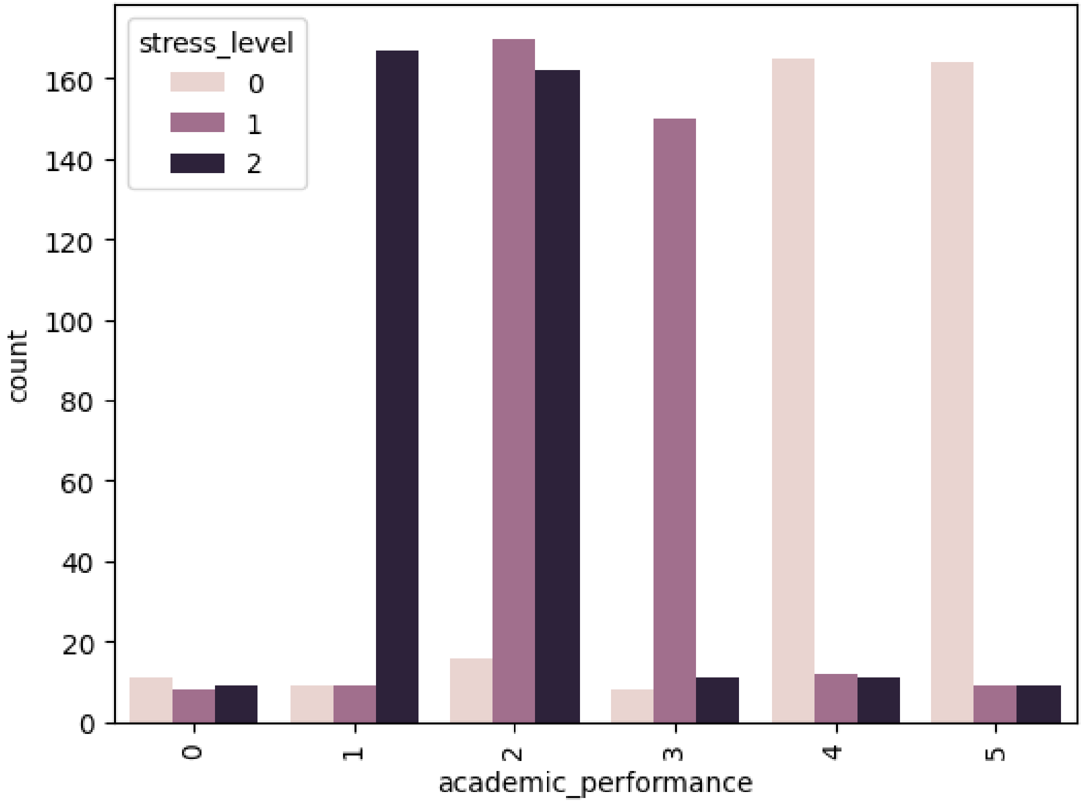

```
column safety is disrtibuted as follow
stress_level  safety
0             4         166
              5         157
              2          15
              1          14
              3          12
              0           9
1             2         168
              3         153
              1          15
              0          10
              5           7
              4           5
2             2         165
              1         155
              3          15
              0          14
              4          13
              5           7
Name: count, dtype: int64
```



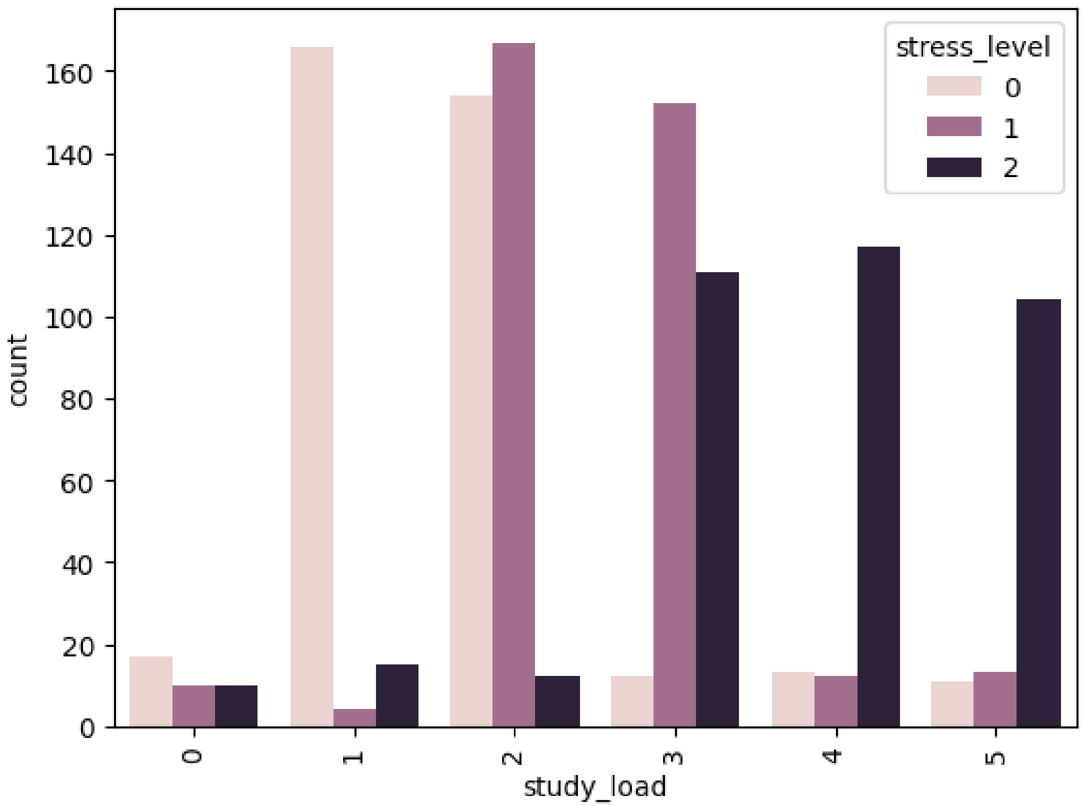

```
column basic_needs is disrtibuted as follow
stress_level  basic_needs
0             5              170
              4              155
              3               14
              1               12
              0               11
              2               11
1             3              162
              2              156
              0               14
              4                9
              5                9
              1                8
2             2              168
              1              153
              0               15
              3               14
              5               10
              4                9
Name: count, dtype: int64
```



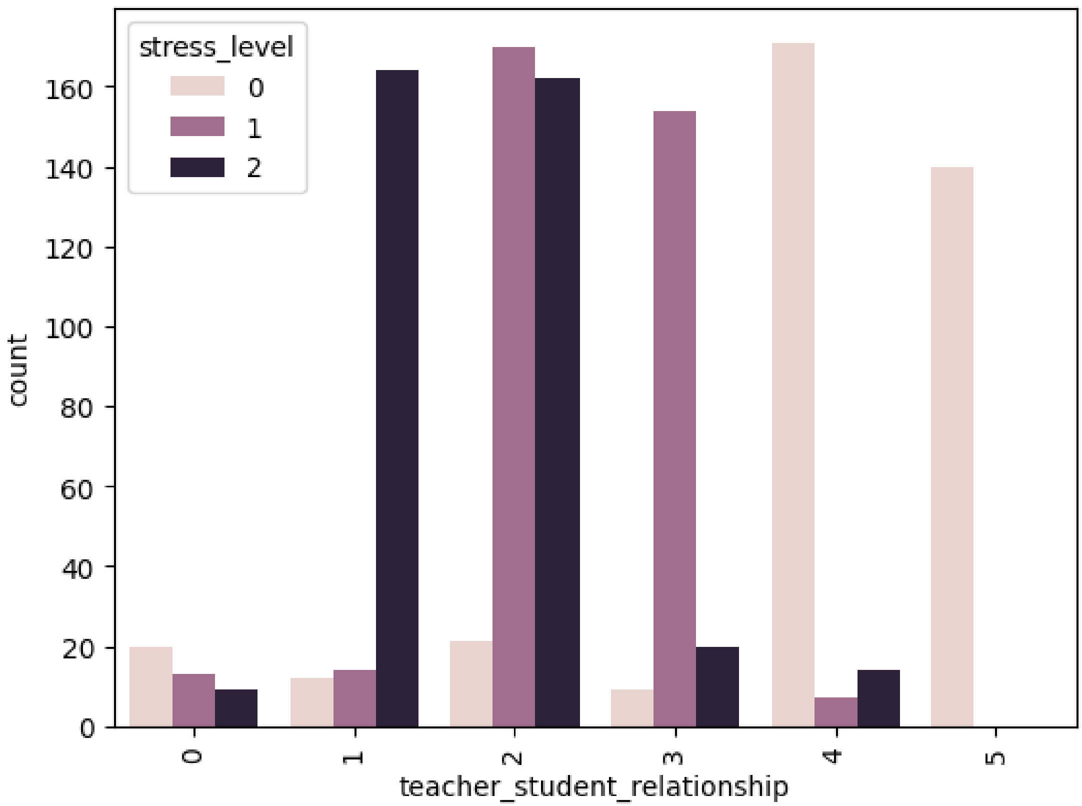

```
column academic_performance is disrtibuted as follow
stress_level  academic_performance
0             4                       165
              5                       164
              2                        16
              0                        11
              1                         9
              3                         8
1             2                       170
              3                       150
              4                        12
              1                         9
              5                         9
              0                         8
2             1                       167
              2                       162
              3                        11
              4                        11
              0                         9
              5                         9
Name: count, dtype: int64
```



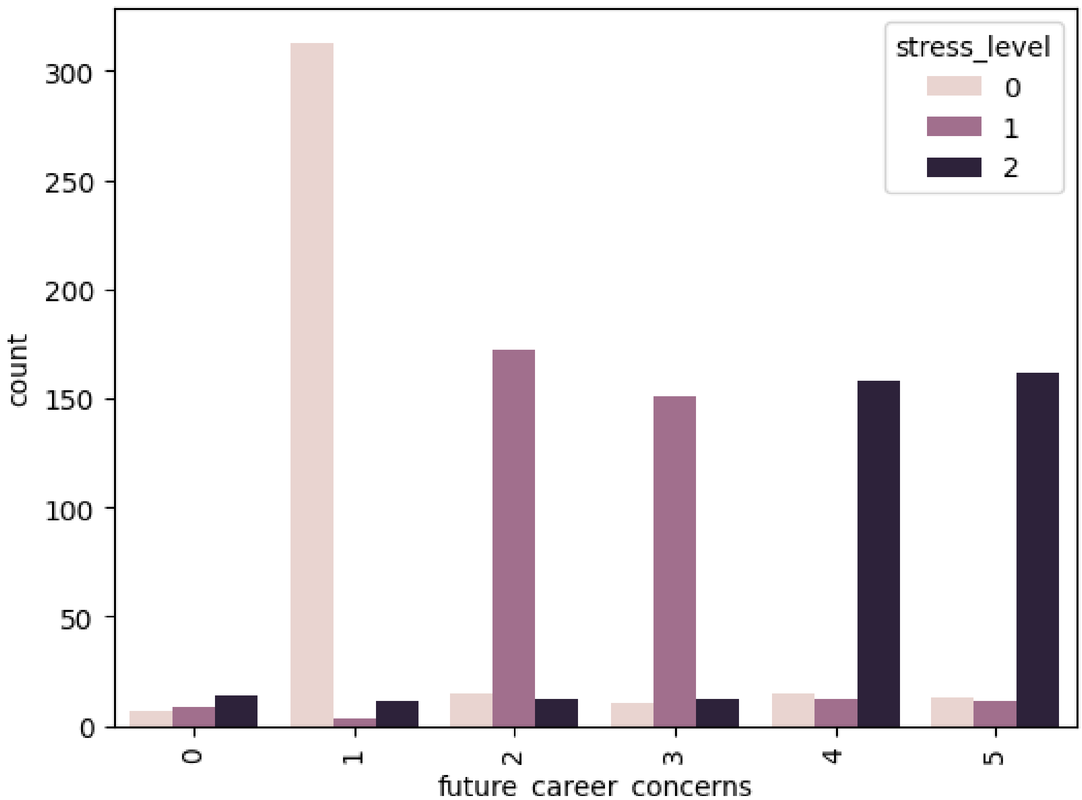

```
column study_load is disrtibuted as follow
stress_level  study_load
0             1             166
              2             154
              0              17
              4              13
              3              12
              5              11
1             2             167
              3             152
              5              13
              4              12
              0              10
              1               4
2             4             117
              3             111
              5             104
              1              15
              2              12
              0              10
Name: count, dtype: int64
```



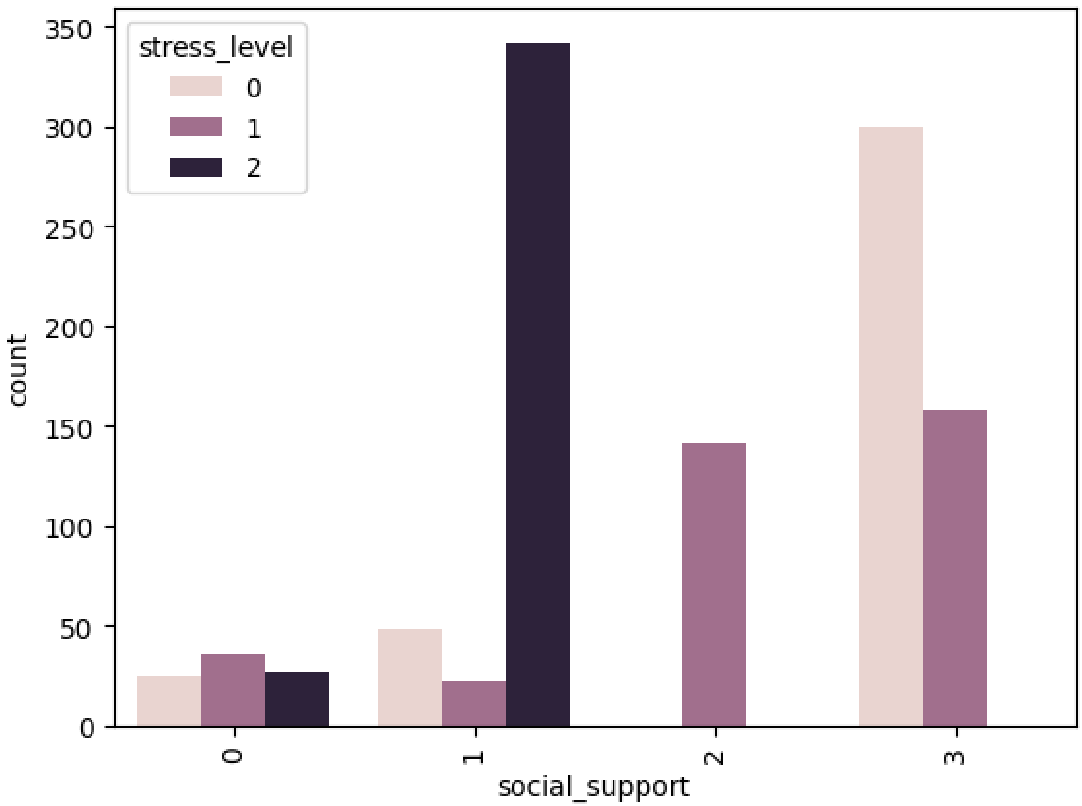

```
column teacher_student_relationship is disrtibuted as follow
stress_level  teacher_student_relationship
0             4                               171
              5                               140
              2                                21
              0                                20
              1                                12
              3                                 9
1             2                               170
              3                               154
              1                                14
              0                                13
              4                                 7
2             1                               164
              2                               162
              3                                20
              4                                14
              0                                 9
Name: count, dtype: int64
```



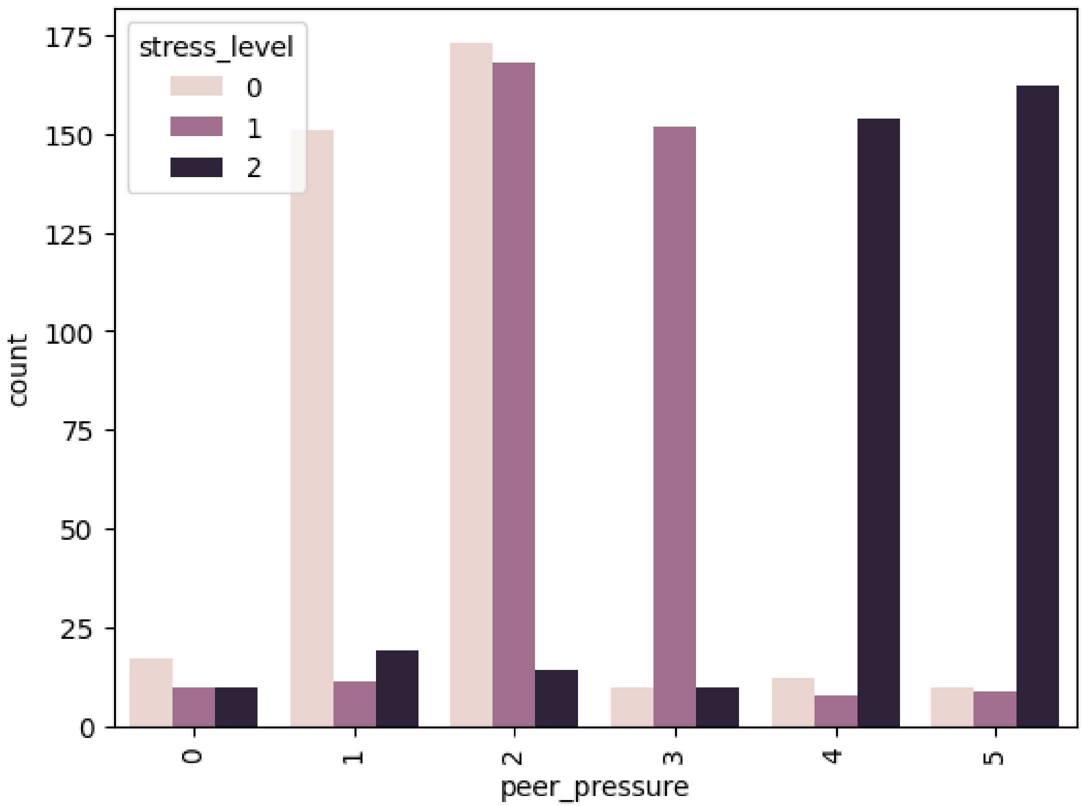

```
column future_career_concerns is disrtibuted as follow
stress_level  future_career_concerns
0             1                         313
              2                          15
              4                          15
              5                          13
              3                          10
              0                           7
1             2                         172
              3                         151
              4                          12
              5                          11
              0                           9
              1                           3
2             5                         162
              4                         158
              0                          14
              2                          12
              3                          12
              1                          11
Name: count, dtype: int64
```



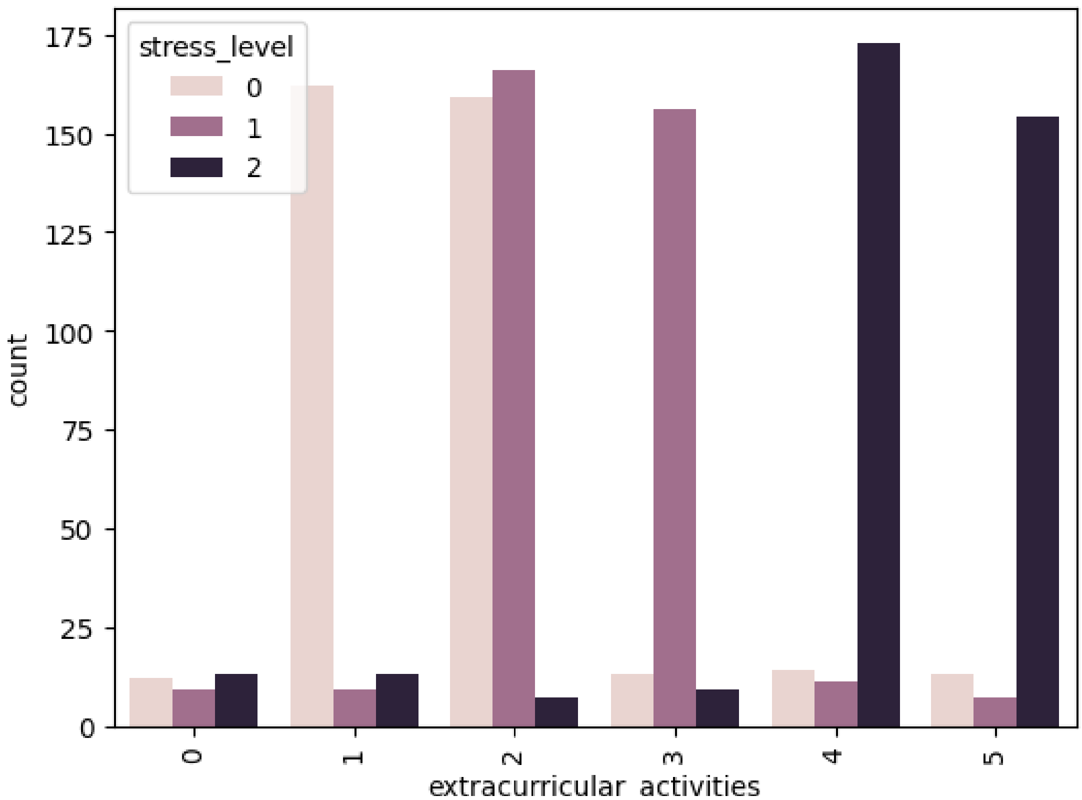

```
column social_support is disrtibuted as follow
stress_level  social_support
0             3                 300
              1                  48
              0                  25
1             3                 158
              2                 142
              0                  36
              1                  22
2             1                 342
              0                  27
Name: count, dtype: int64
```



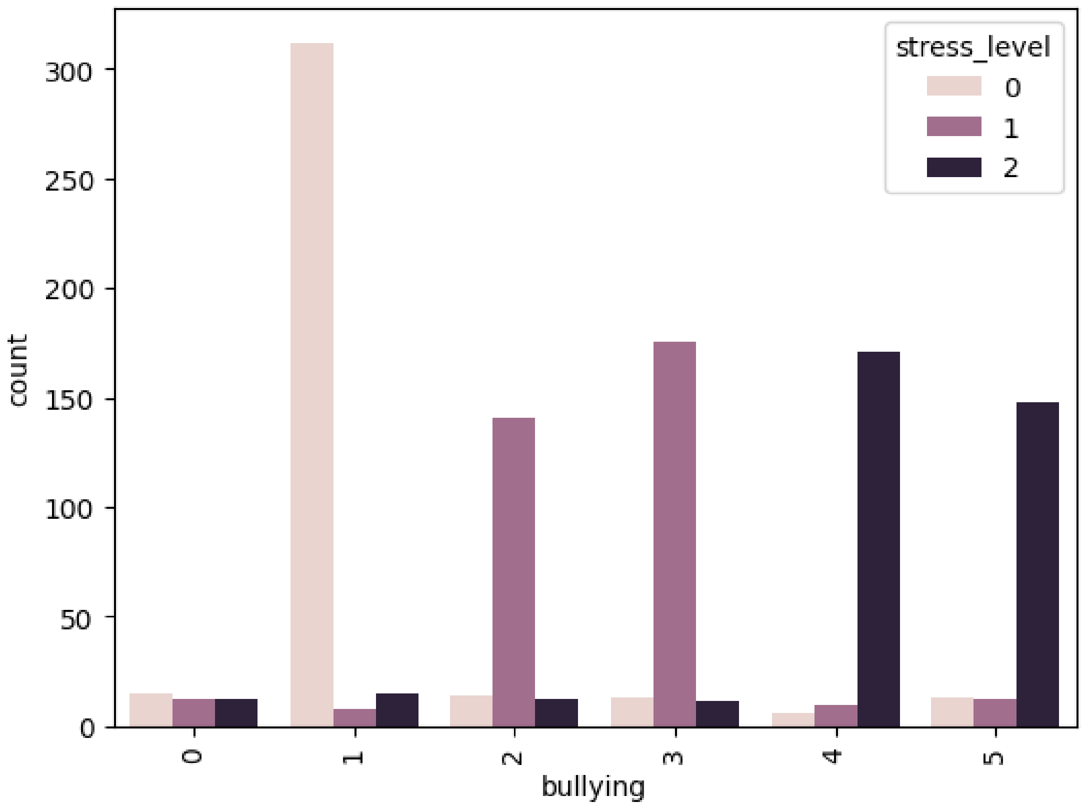

```
column peer_pressure is disrtibuted as follow
stress_level  peer_pressure
0             2                173
              1                151
              0                 17
              4                 12
              3                 10
              5                 10
1             2                168
              3                152
              1                 11
              0                 10
              5                  9
              4                  8
2             5                162
              4                154
              1                 19
              2                 14
              0                 10
              3                 10
Name: count, dtype: int64
```



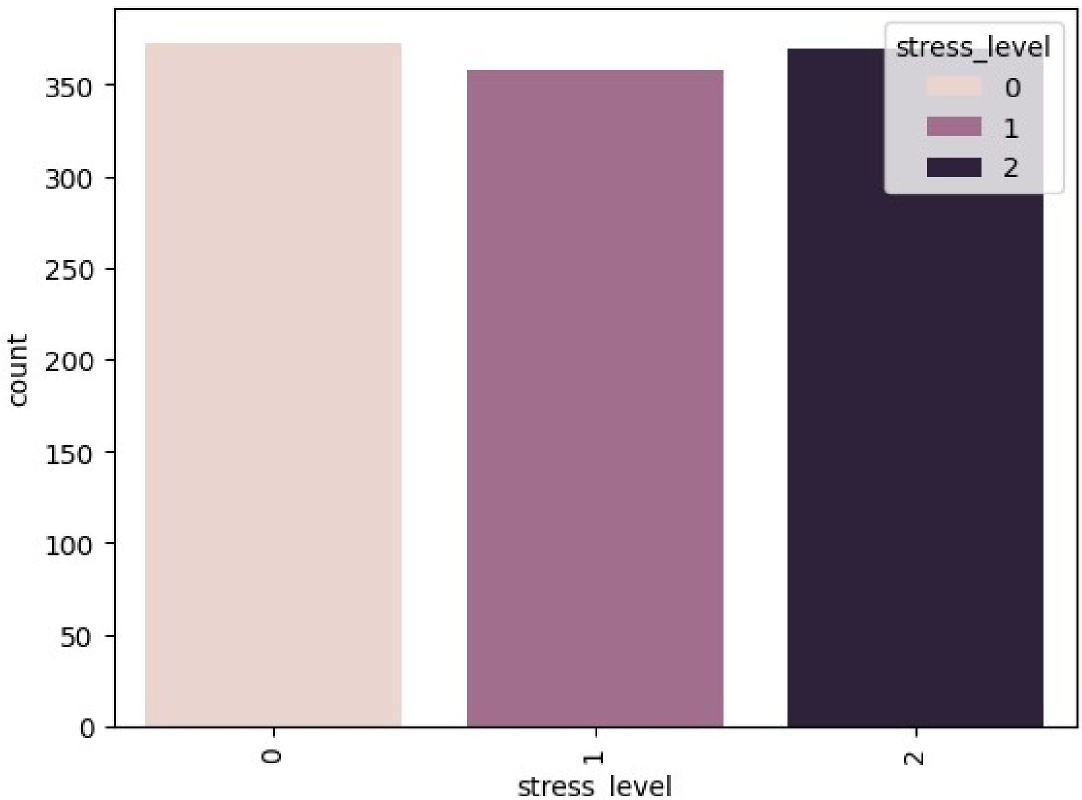

```
column extracurricular_activities is disrtibuted as follow
stress_level  extracurricular_activities
0             1                             162
              2                             159
              4                              14
              3                              13
              5                              13
              0                              12
1             2                             166
              3                             156
              4                              11
              0                               9
              1                               9
              5                               7
2             4                             173
              5                             154
              0                              13
              1                              13
              3                               9
              2                               7
Name: count, dtype: int64
```



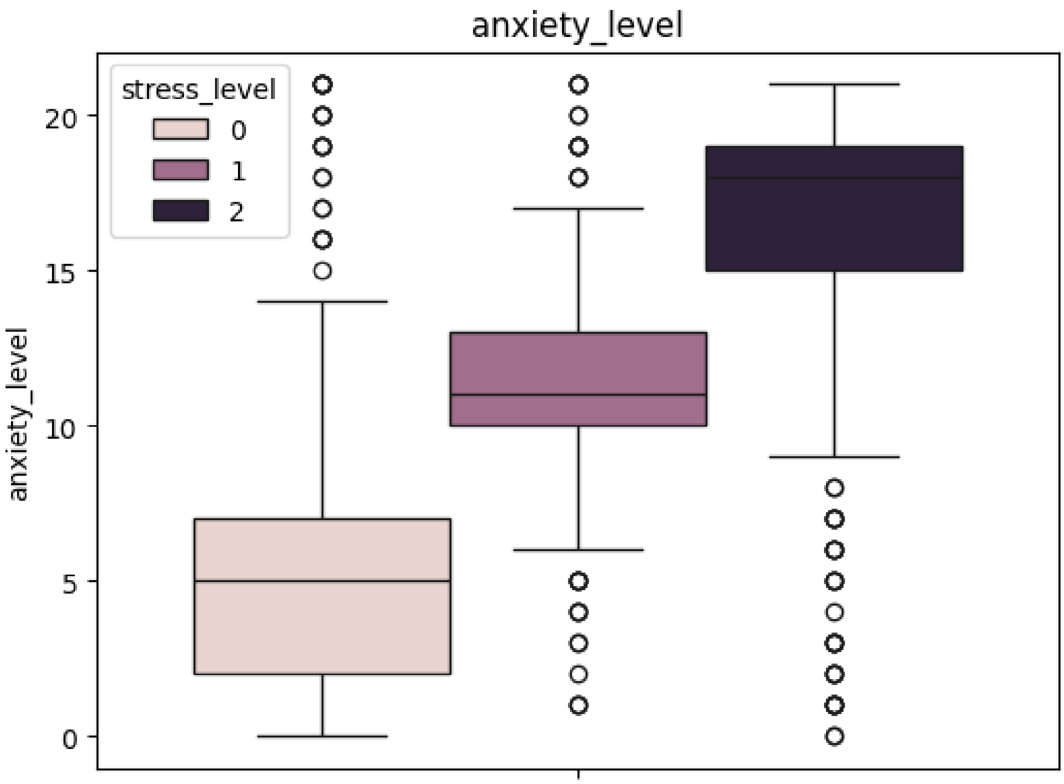

```
column bullying is disrtibuted as follow
stress_level  bullying
0             1           312
              0            15
              2            14
              3            13
              5            13
              4             6
1             3           175
              2           141
              0            12
              5            12
              4            10
              1             8
2             4           171
              5           148
              1            15
              0            12
              2            12
              3            11
Name: count, dtype: int64
```

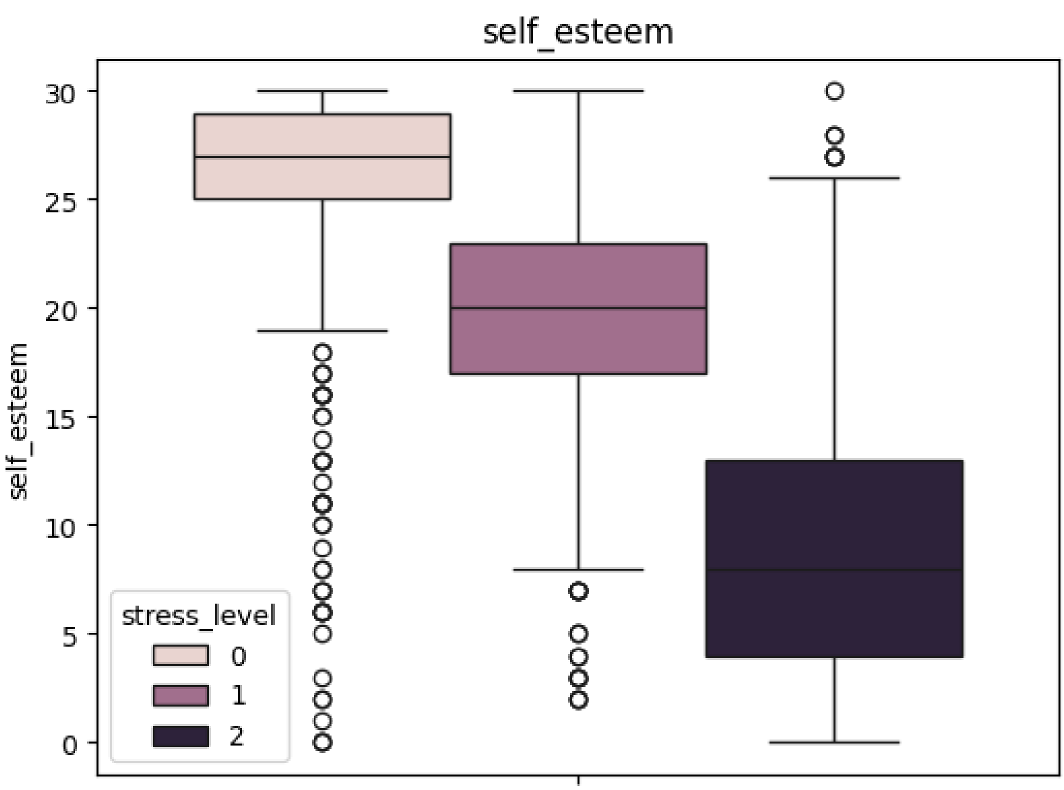


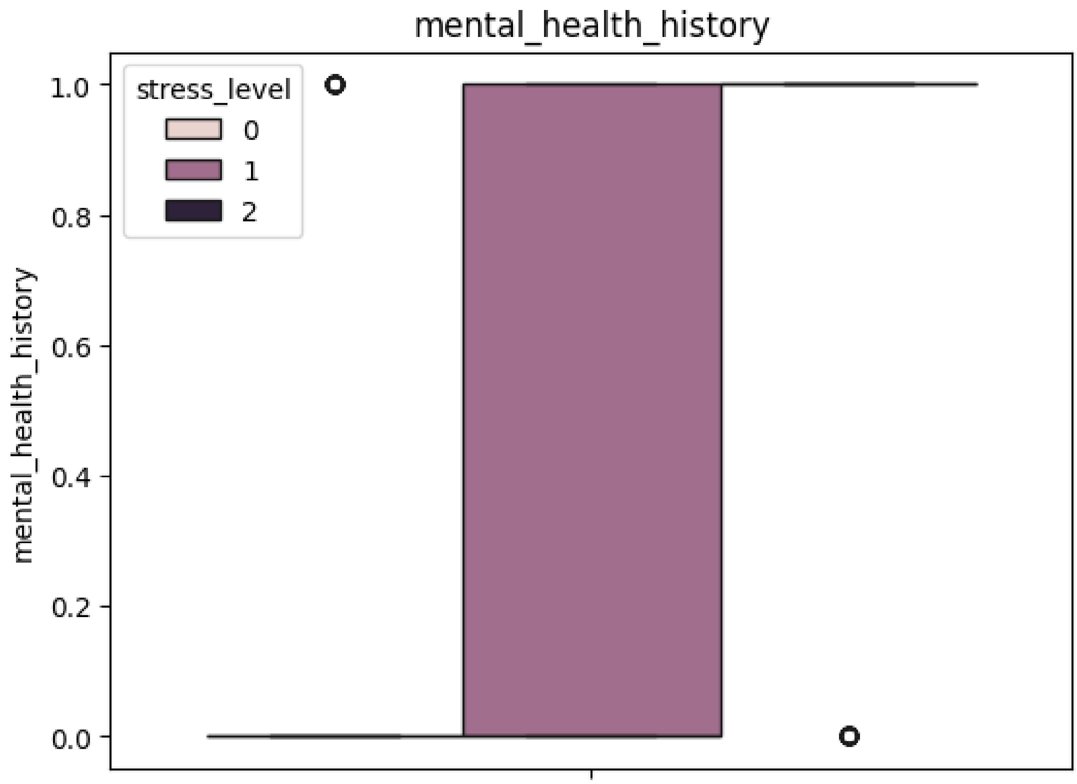

```
column stress_level is disrtibuted as follow
stress_level
0    373
1    358
2    369
Name: count, dtype: int64
```

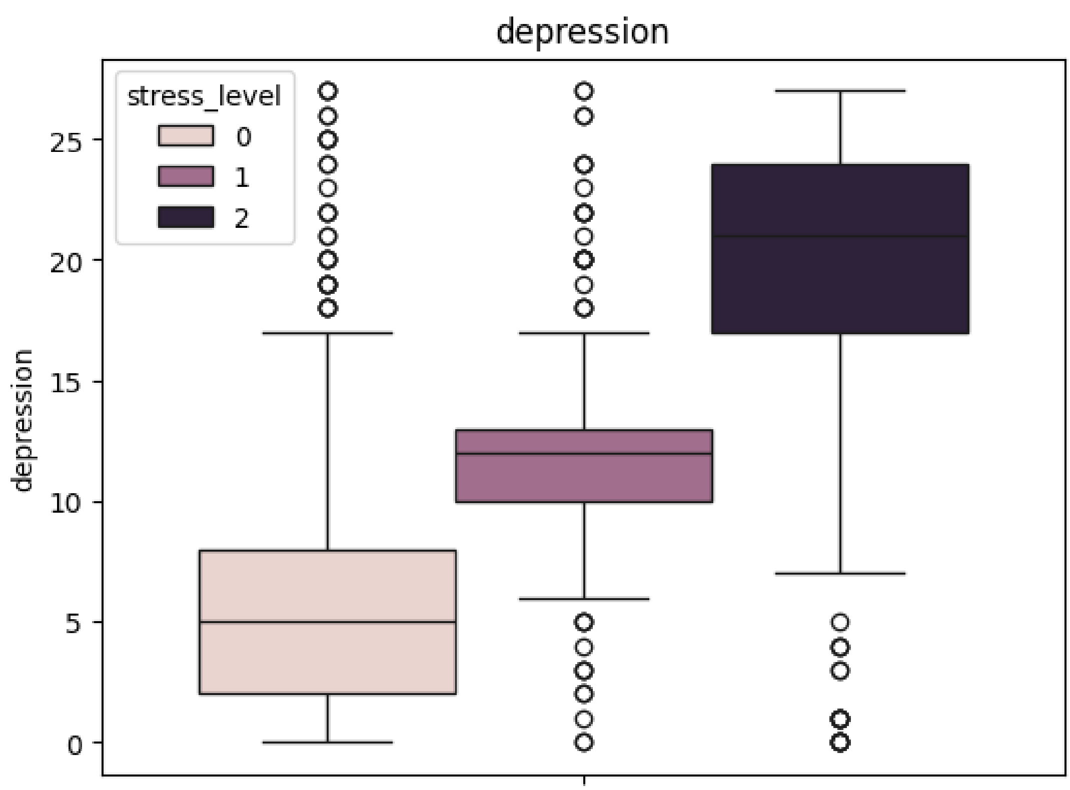


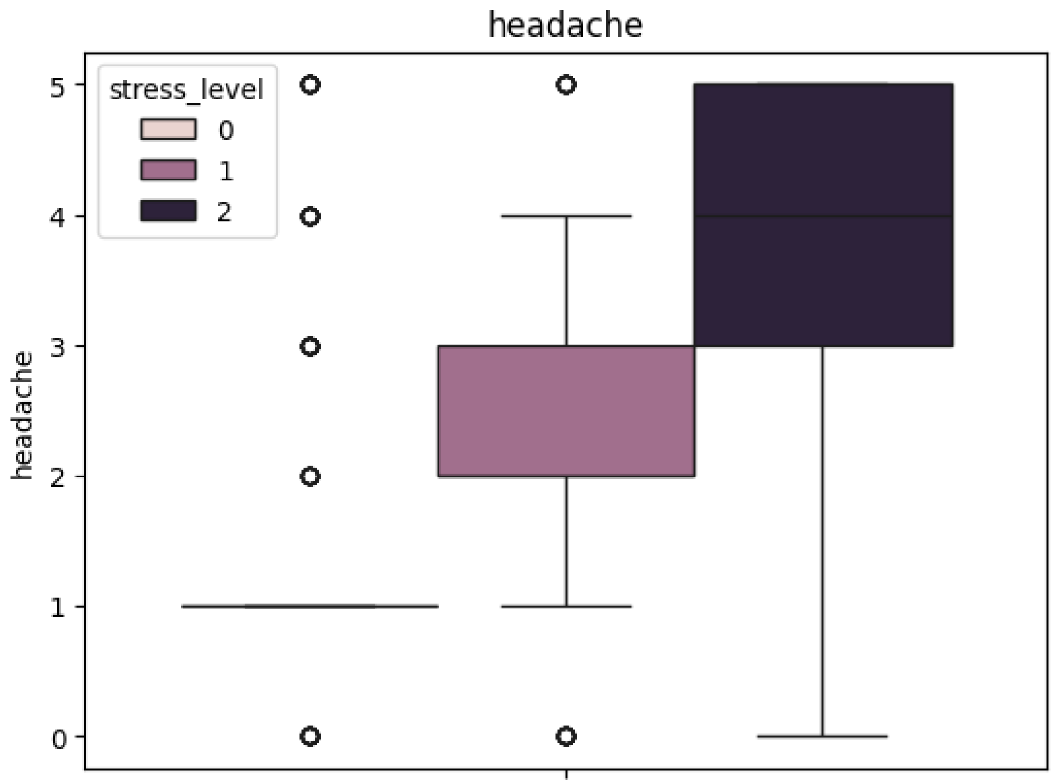

```
#box plot of each column to stress
for column in df.columns:
  sns.boxplot(y =column,data = df, hue = 'stress_level')
  plt.title(column)
  plt.show()
```

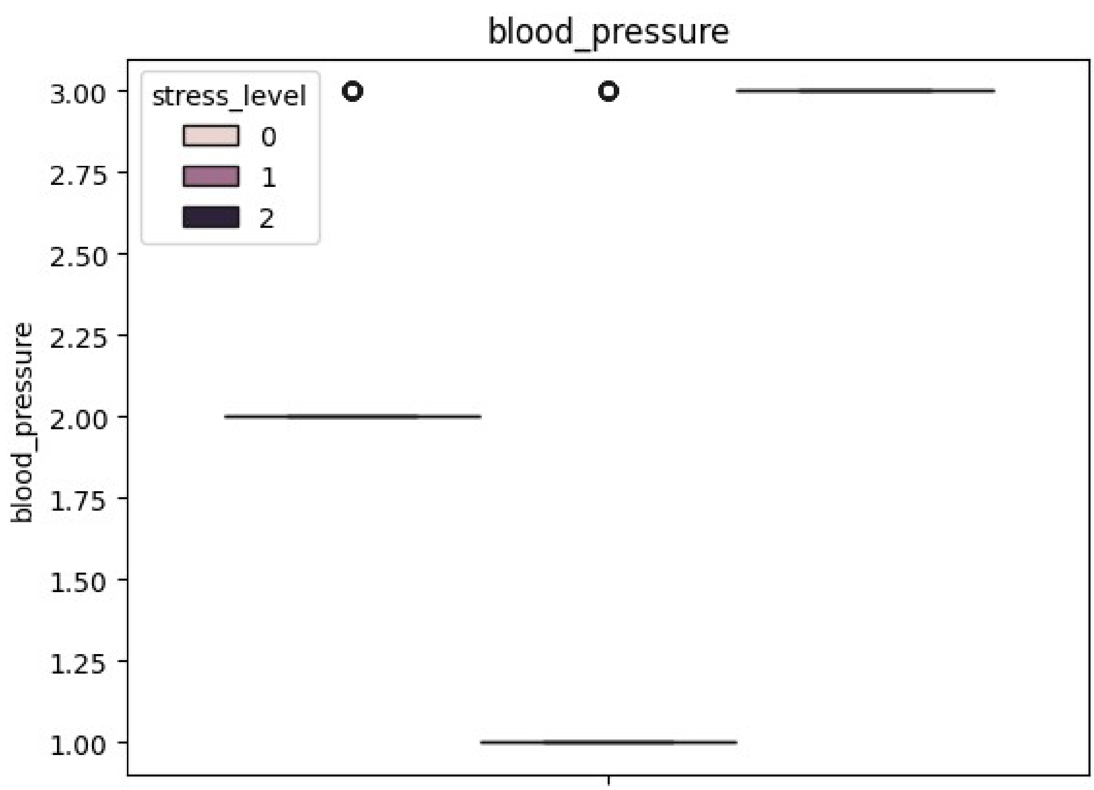

anxiety_level
stress_level
0
1
2
20
15
10
5
0
anxiety_level

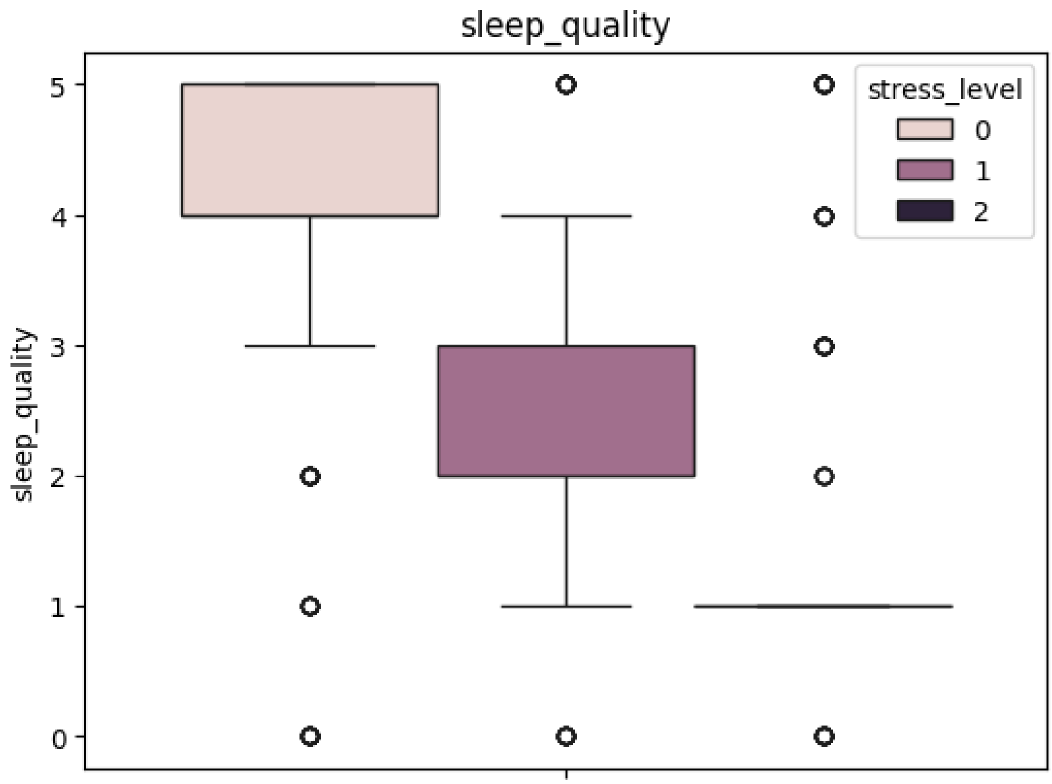

self_esteem
stress_level
0
1
2
30
25
20
15
10
5
0
self_esteem

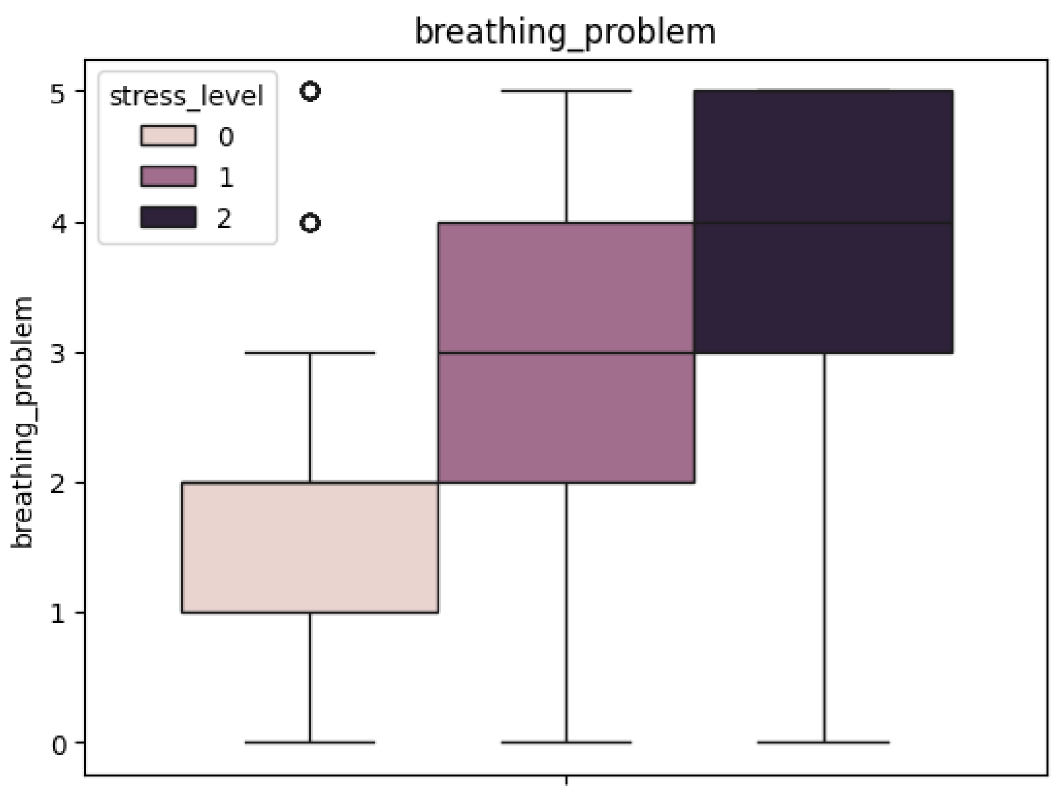

mental_health_history
stress_level
0
1
2
1.0
0.8
0.6
0.4
0.2
0.0
mental_health_history

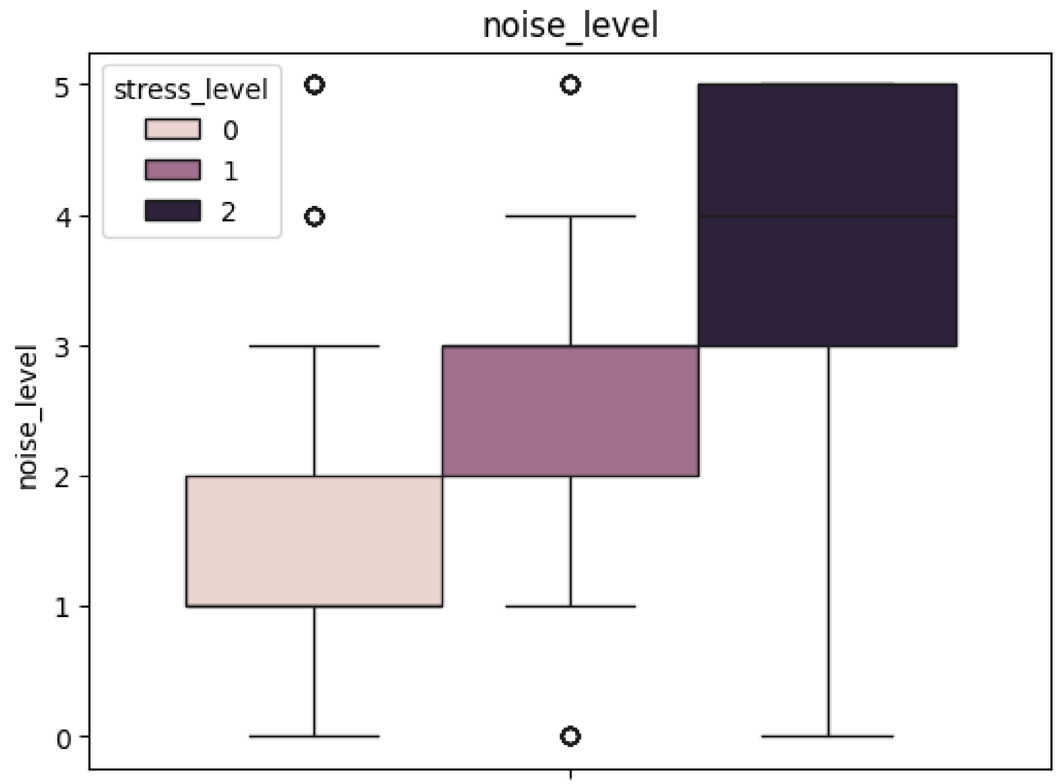

depression
stress_level
0
1
2
25
20
15
10
5
0
depression

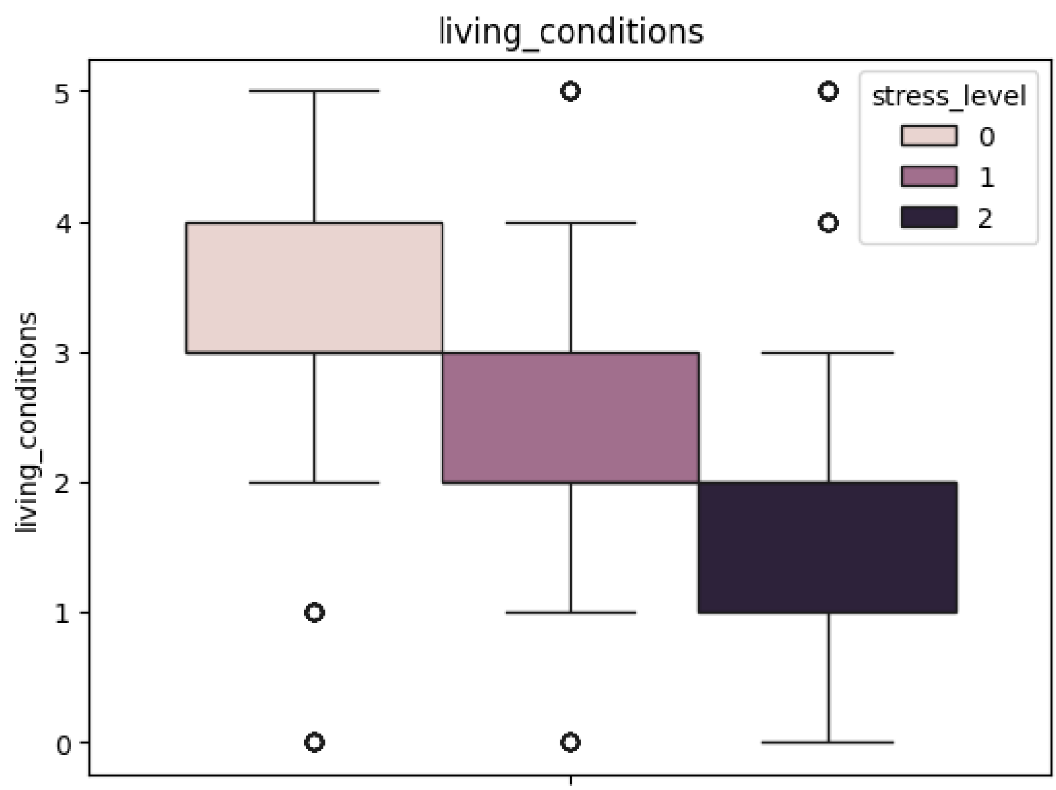

headache
stress_level
0
1
2
5
4
3
2
1
0
headache

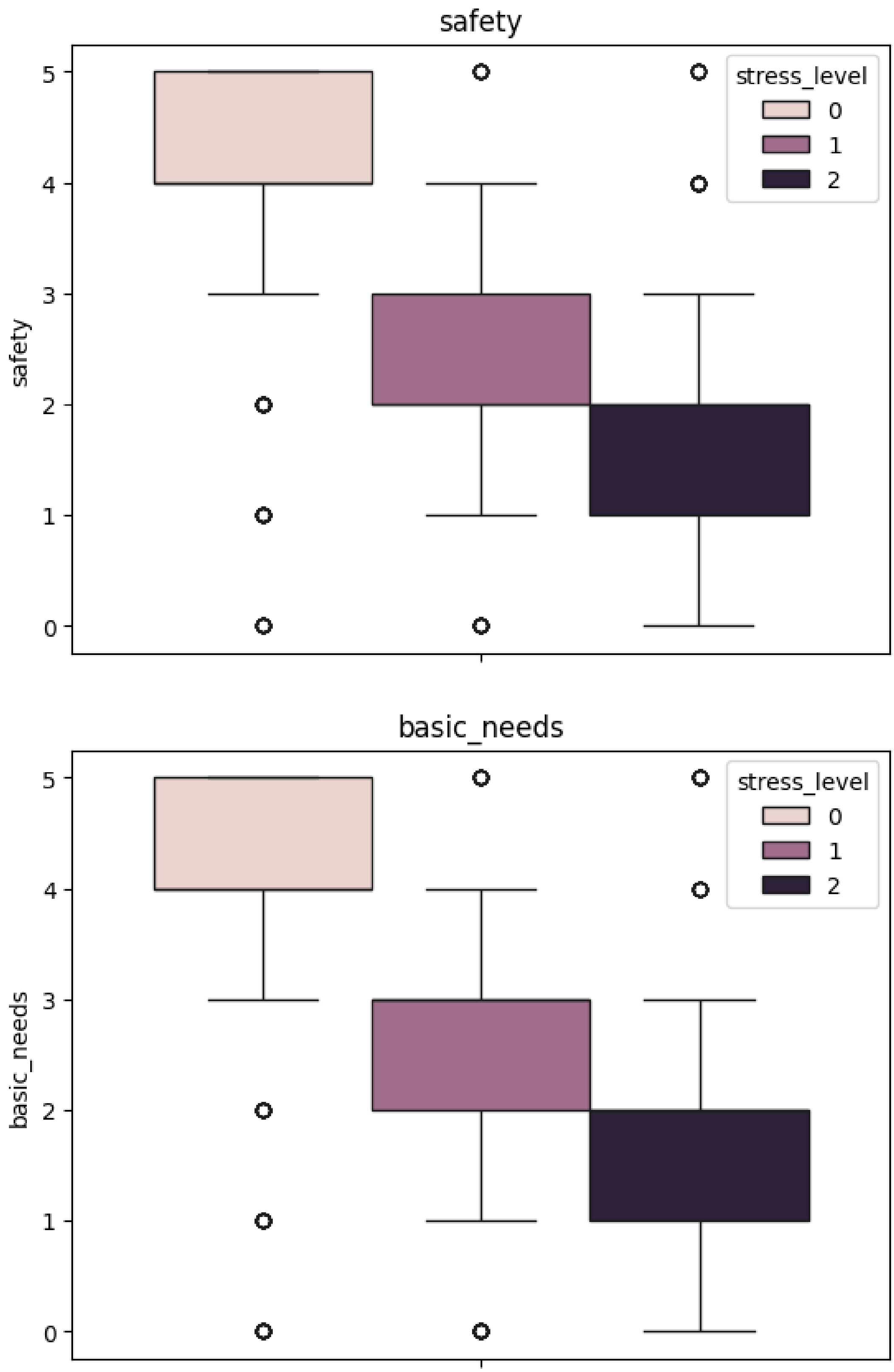

blood_pressure
stress_level
0
1
2
3.00
2.75
2.50
2.25
2.00
1.75
1.50
1.25
1.00
blood_pressure

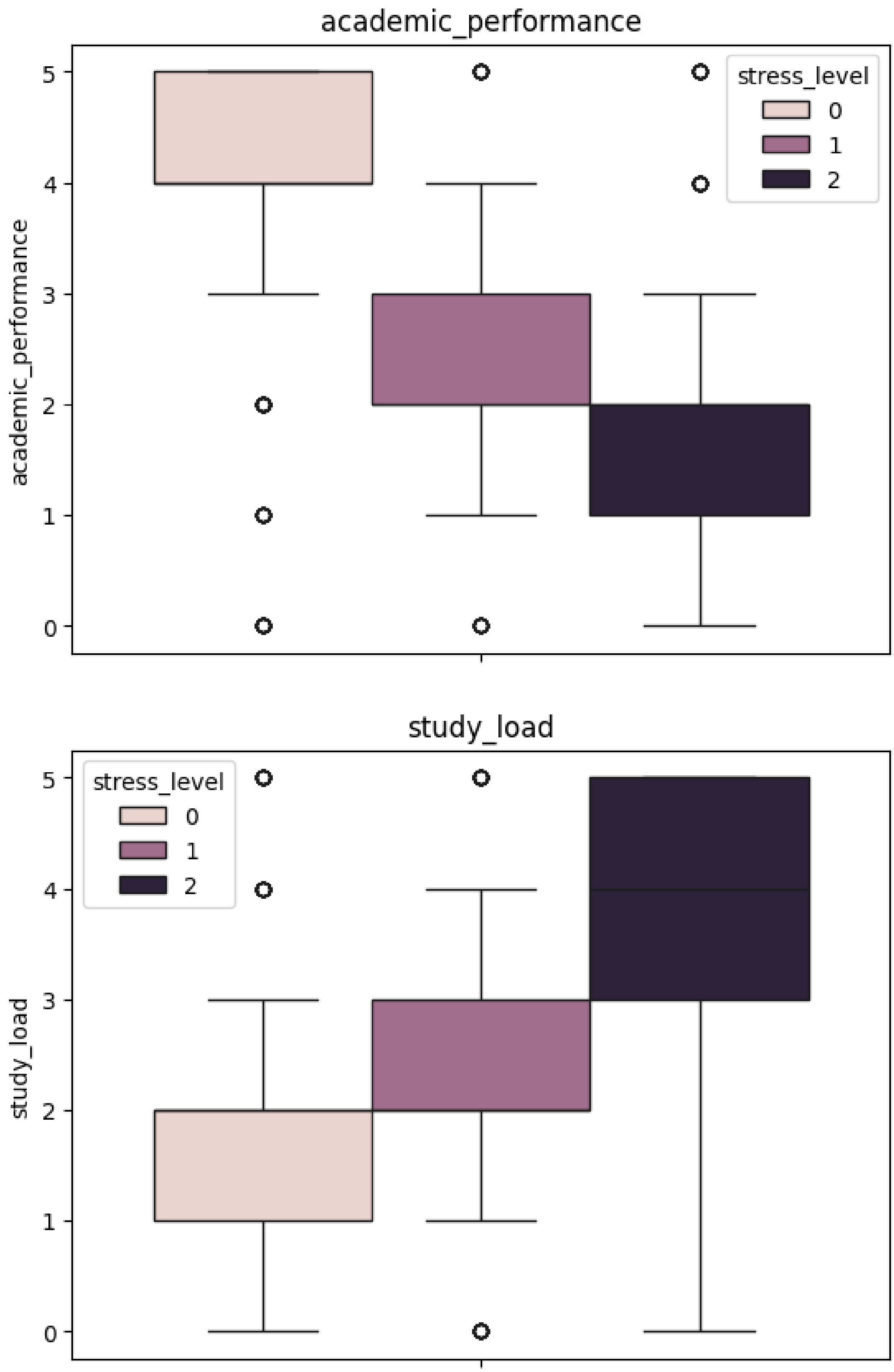

sleep_quality
stress_level
0
1
2
sleep_quality
0
1
2
3
4
5

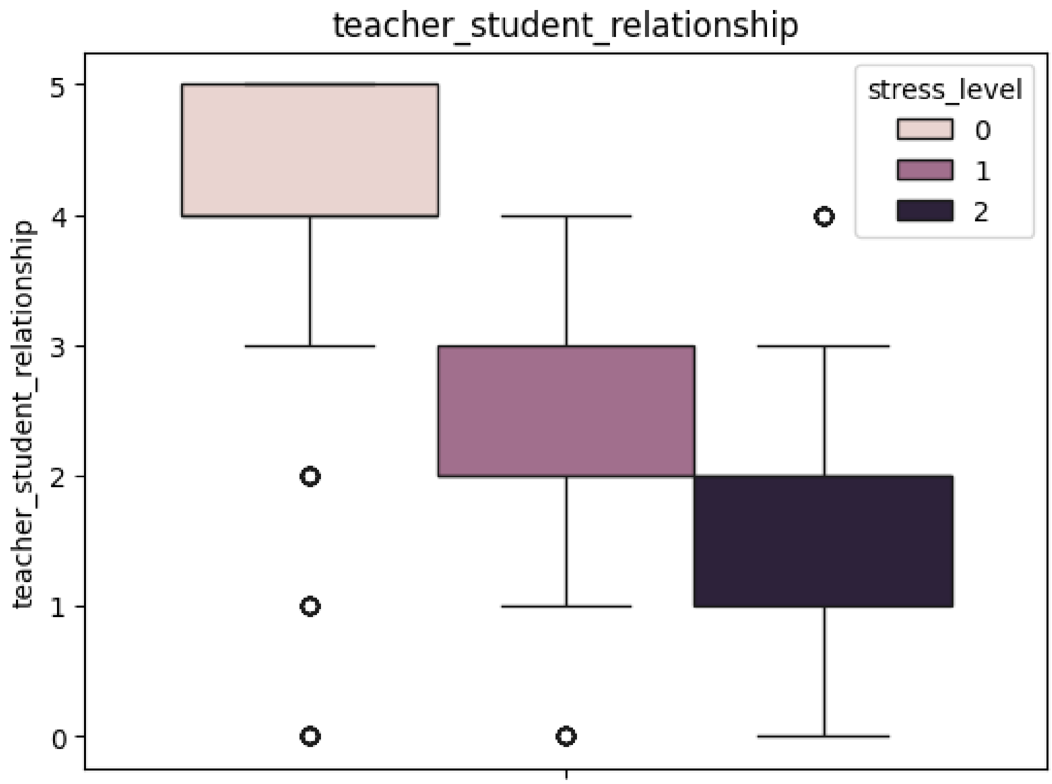

breathing_problem
stress_level
0
1
2
breathing_problem
0
1
2
3
4
5

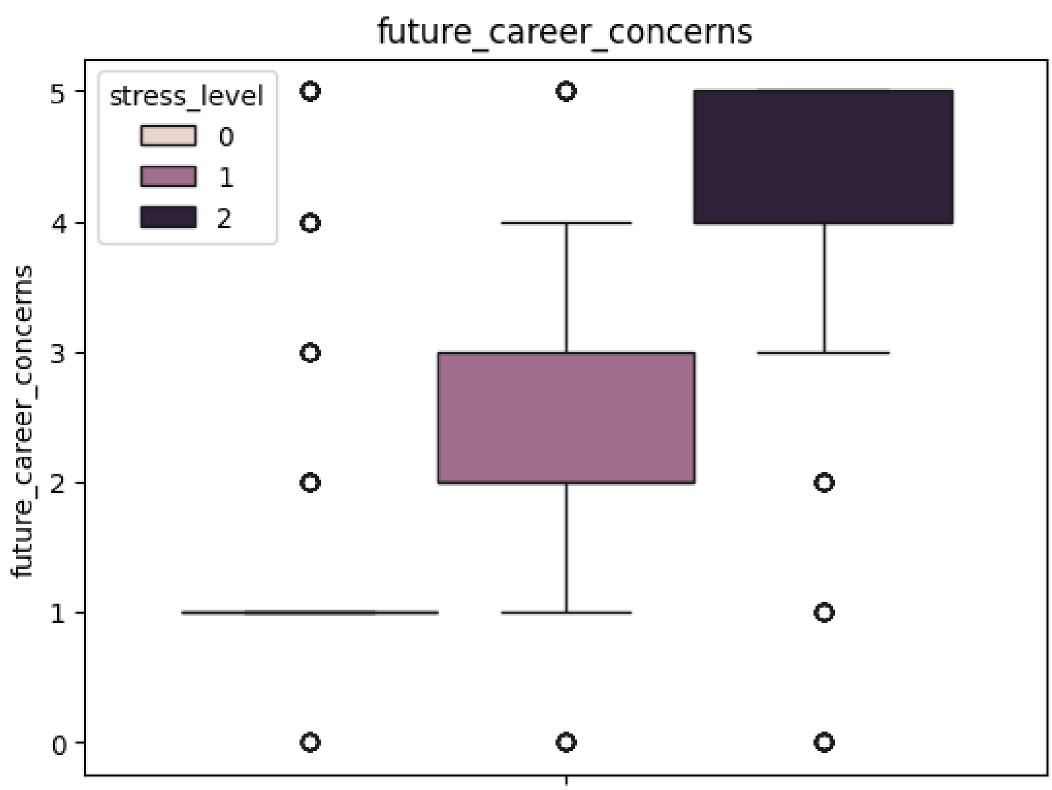

noise_level
noise_level
stress_level
0
1
2
0
1
2
3
4
5

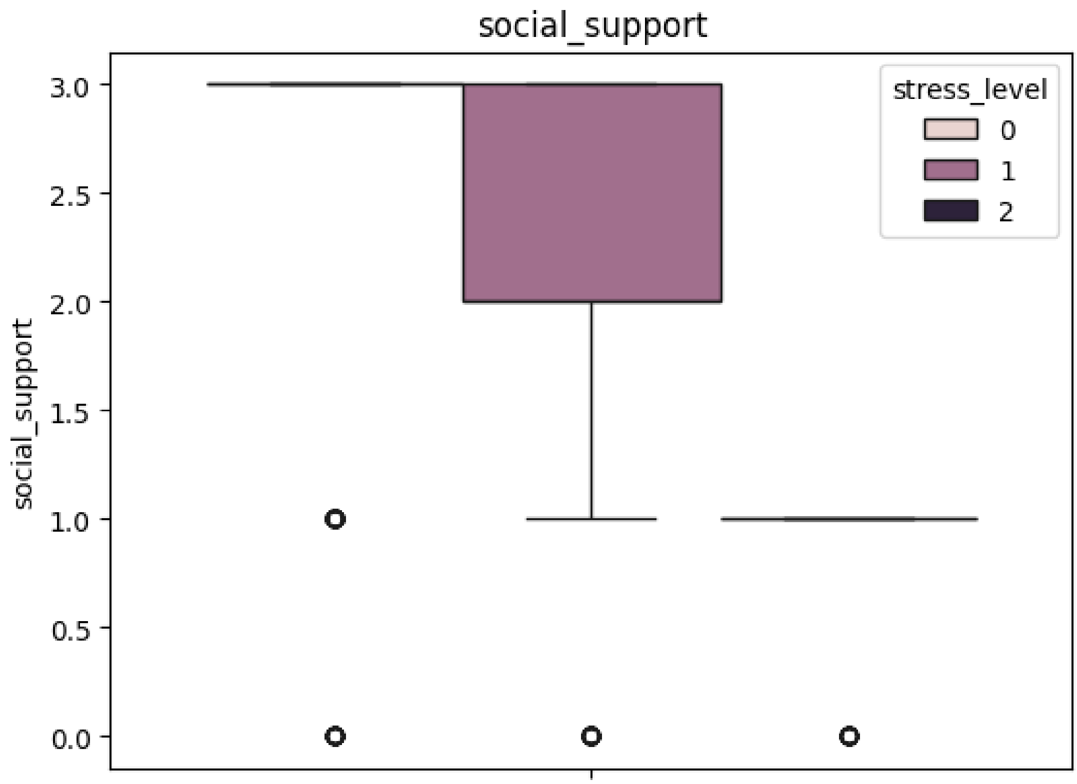

living_conditions
living_conditions
stress_level
0
1
2
0
1
2
3
4
5

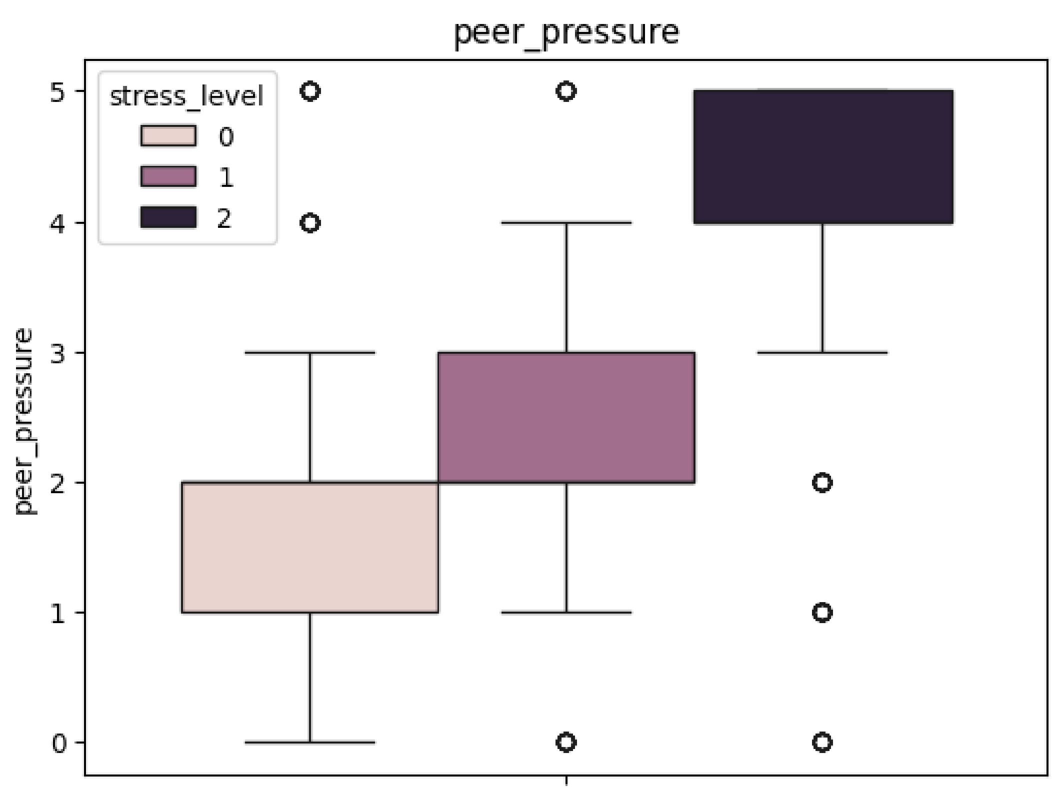

safety
stress_level
0
1
2
safety
0
1
2
3
4
5
basic_needs
stress_level
0
1
2
basic_needs
0
1
2
3
4
5

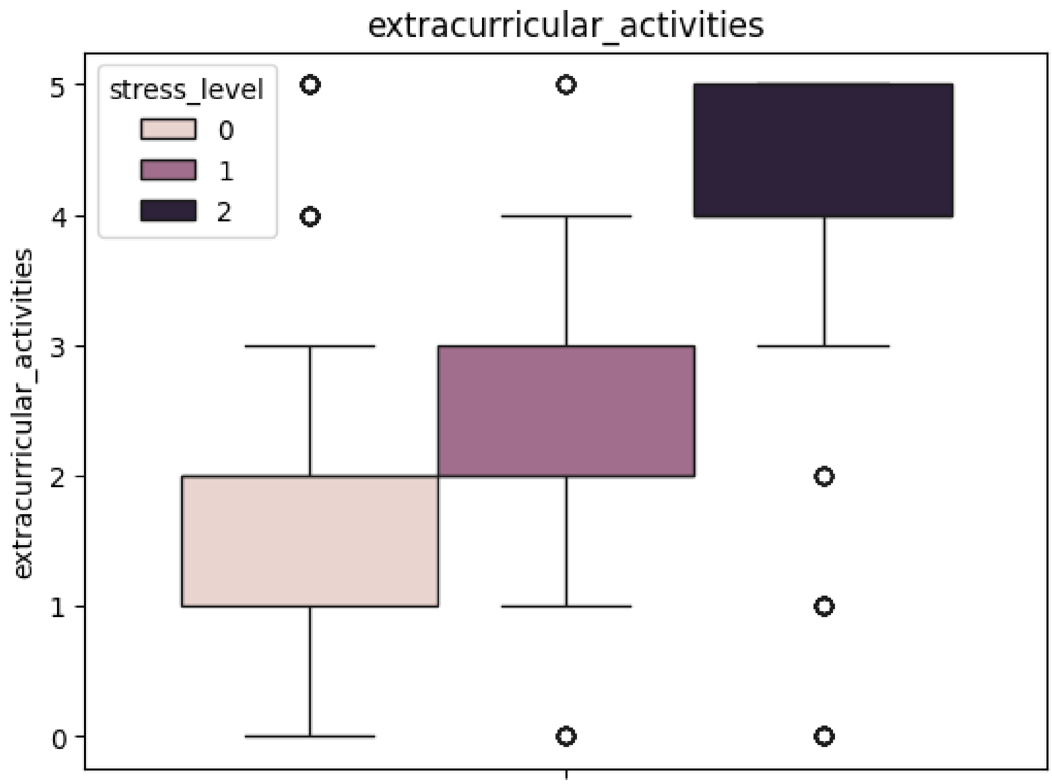

academic_performance
stress_level
0
1
2
academic_performance
5
4
3
2
1
0
study_load
stress_level
0
1
2
study_load
5
4
3
2
1
0


teacher_student_relationship
teacher_student_relationship
stress_level
0
1
2
0
1
2
3
4
5


future_career_concerns
future_career_concerns
stress_level
0
1
2
0
1
2
3
4
5


social_support
stress_level
0
1
2
social_support
3.0
2.5
2.0
1.5
1.0
0.5
0.0


peer_pressure
stress_level
0
1
2
peer_pressure
5
4
3
2
1
0


extracurricular_activities
stress_level
0
1
2
extracurricular_activities
0
1
2
3
4
5


bullying
stress_level
0
1
2
bullying
0
1
2
3
4
5

starts training

MLP

```
#training artificial neural network
X = df.drop('stress_level', axis=1).values
y = df['stress_level'].values
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

encoder = OneHotEncoder()
y_encoded = encoder.fit_transform(y.reshape(-1, 1))
y_encoded = y_encoded.toarray() #To convert from sparse array to numpy
array
# Convert to PyTorch tensors
X_tensor = torch.tensor(X_scaled, dtype=torch.float)
y_tensor = torch.tensor(y_encoded, dtype=torch.float)
X_train, X_test, y_train_nn, y_test_nn = train_test_split(X_tensor,
y_tensor, test_size=0.2, random_state=42)

# Create DataLoader for batch processing
train_dataset = TensorDataset(X_train, y_train_nn)
test_dataset = TensorDataset(X_test, y_test_nn)

batch_size = 16
```

```
train_loader = DataLoader(dataset=train_dataset,
batch_size=batch_size, shuffle=True)
test_loader = DataLoader(dataset=test_dataset, batch_size=batch_size,
shuffle=False)

class MLP(nn.Module):
    def __init__(self, input_size, output_size):
        super(MLP, self).__init__()
        self.fc1 = nn.Linear(input_size, 12) # 20 to 12 to 3
        self.fc2 = nn.Linear(12, 6)
        self.fc3 = nn.Linear(6, output_size)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

model = MLP(input_size=X_train.shape[1],
output_size=y_train_nn.shape[1])

optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
criterion = nn.BCEWithLogitsLoss()
epochs = 100
for epoch in tqdm(range(epochs)):
    for inputs, targets in train_loader:
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, targets)
        loss.backward()
        optimizer.step()
    #Printing the loss every N epochs
    if (epoch+1) % 10 == 0:
        print(f'Epoch {epoch+1}, Loss: {loss.item()}')

model.eval()
with torch.no_grad():
    y_true = []
    y_pred = []

    for inputs, targets in test_loader:
        outputs = model(inputs)
        predicted = torch.argmax(outputs, dim=1)
        actual = torch.argmax(targets, dim=1)

        y_true.extend(actual.cpu().numpy())
        y_pred.extend(predicted.cpu().numpy())

y_true = np.array(y_true)
y_pred = np.array(y_pred)
```

```python
report = classification_report(y_true, y_pred)
print(report)
```

```
 12%|█         | 12/100 [00:00<00:06, 12.62it/s]

Epoch 10, Loss: 0.09891056269407272

 22%|██        | 22/100 [00:01<00:06, 12.85it/s]

Epoch 20, Loss: 0.22076529264450073

 32%|███       | 32/100 [00:02<00:05, 13.26it/s]

Epoch 30, Loss: 0.09716657549142838

 42%|████      | 42/100 [00:03<00:04, 13.06it/s]

Epoch 40, Loss: 0.08672487735748291

 52%|█████     | 52/100 [00:04<00:03, 12.80it/s]

Epoch 50, Loss: 0.041329558938741684

 62%|██████    | 62/100 [00:04<00:02, 12.91it/s]

Epoch 60, Loss: 0.06091674789786339

 72%|███████   | 72/100 [00:05<00:02, 13.06it/s]

Epoch 70, Loss: 0.03014170564711094

 82%|████████  | 82/100 [00:06<00:01, 13.05it/s]

Epoch 80, Loss: 0.10898363590240479

 92%|█████████ | 92/100 [00:07<00:00, 12.70it/s]

Epoch 90, Loss: 0.1177188977599144

100%|██████████| 100/100 [00:07<00:00, 12.95it/s]

Epoch 100, Loss: 0.06822415441274643
              precision    recall  f1-score   support

           0       0.90      0.87      0.89        76
           1       0.91      0.93      0.92        73
           2       0.89      0.90      0.90        71

    accuracy                           0.90       220
   macro avg       0.90      0.90      0.90       220
weighted avg       0.90      0.90      0.90       220
```

```
x = df.iloc[:,:-1]
y = df.iloc[:,-1]

x_train,x_test,y_train,y_test =
train_test_split(x,y,test_size=0.20,random_state=0)

scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.fit_transform(x_test)

param_grid = {'n_neighbors': list(range(1,30))}
knn_model = KNeighborsClassifier()
g_search = GridSearchCV(knn_model,param_grid,cv=5,scoring='accuracy')
g_search.fit(x_train_scaled,y_train)
optimal_k = g_search.best_params_['n_neighbors']
print(f'Optimal value for k : {optimal_k}')

Optimal value for k : 3

/usr/local/lib/python3.10/dist-packages/numpy/ma/core.py:2820:
RuntimeWarning: invalid value encountered in cast
  _data = np.array(data, dtype=dtype, copy=copy,
```

ensamble moodel

```
model1 = KNeighborsClassifier(n_neighbors = optimal_k)
model2 = SVC(kernel='linear')
model3 = DecisionTreeClassifier()

ensemble = VotingClassifier(estimators=[
    ('knn', model1), ('svc', model2), ('dt', model3)],
    voting='hard')# mahority class

ensemble.fit(x_train, y_train)
y_pred = ensemble.predict(x_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("confusion Matrix: ",confusion_matrix(y_test,y_pred))
print("classification_report: ",classification_report(y_test,y_pred))

Accuracy: 0.8818181818181818
confusion Matrix:  [[66  2  6]
 [ 5 70  5]
 [ 3  5 58]]
classification_report:                precision    recall  f1-score
support

           0       0.89      0.89      0.89        74
           1       0.91      0.88      0.89        80
           2       0.84      0.88      0.86        66
```

```
    accuracy                           0.88       220
   macro avg       0.88      0.88      0.88       220
weighted avg       0.88      0.88      0.88       220
```

```
#testing with aurak data
y_pred = ensemble.predict(Aurak_df_x)
print("Accuracy:", accuracy_score(Aurak_df_y, y_pred))
print("confusion Matrix: ",confusion_matrix(Aurak_df_y,y_pred))
print("classification_report:
",classification_report(Aurak_df_y,y_pred, zero_division=1))
```

```
Accuracy: 0.5576923076923077
confusion Matrix:  [[ 0  6  0]
 [ 0 29  1]
 [ 0 16  0]]
classification_report:                precision    recall  f1-score
support

           0       1.00      0.00      0.00         6
           1       0.57      0.97      0.72        30
           2       0.00      0.00      0.00        16

    accuracy                           0.56        52
   macro avg       0.52      0.32      0.24        52
weighted avg       0.44      0.56      0.41        52
```

knn

```
#normal knn
knn_model = KNeighborsClassifier(n_neighbors = optimal_k)
knn_model.fit(x_train_scaled,y_train)
knn_model_y_predicted = knn_model.predict(x_test_scaled)
cv_scores = cross_val_score(knn_model, x_test_scaled, y_test, cv=5,
scoring='accuracy')
knn_accuracy = cv_scores.mean()
knn_conf_matrix = confusion_matrix(y_test,knn_model_y_predicted)
knn_classification_report =
classification_report(y_test,knn_model_y_predicted)
print(f'Confusion Matrix: \n {knn_conf_matrix}')
print(f'Classification Report: \n {knn_classification_report}')
print(f'Accuracy: \n {knn_accuracy}')
```

```
Confusion Matrix:
 [[63  3  8]
 [ 5 70  5]
 [ 5  2 59]]
Classification Report:
```

```
              precision    recall  f1-score   support

           0       0.86      0.85      0.86        74
           1       0.93      0.88      0.90        80
           2       0.82      0.89      0.86        66

    accuracy                           0.87       220
   macro avg       0.87      0.87      0.87       220
weighted avg       0.88      0.87      0.87       220

Accuracy:
 0.8727272727272727
```

```
#testing with aurak data
y_pred = knn_model.predict(Aurak_df_x.values)
print("Accuracy:", accuracy_score(Aurak_df_y, y_pred))
print("confusion Matrix: ",confusion_matrix(Aurak_df_y,y_pred))
print("classification_report:
",classification_report(Aurak_df_y,y_pred))
```

```
Accuracy: 0.4230769230769231
confusion Matrix:  [[ 6  0  0]
 [ 8  5 17]
 [ 2  3 11]]
classification_report:                precision    recall  f1-score
support

           0       0.38      1.00      0.55         6
           1       0.62      0.17      0.26        30
           2       0.39      0.69      0.50        16

    accuracy                           0.42        52
   macro avg       0.46      0.62      0.44        52
weighted avg       0.52      0.42      0.37        52
```

GB

```
#normal gradient boosting
gb_classifier =
GradientBoostingClassifier(n_estimators=10,learning_rate=1.0,max_depth
=2,random_state=0)
gb_classifier.fit(x_train,y_train)
y_predicted = gb_classifier.predict(x_test)
cv_scores = cross_val_score(gb_classifier, x_test, y_test, cv=5,
scoring='accuracy')
gb_accuracy = cv_scores.mean()
gb_conf_matrix = confusion_matrix(y_test,y_predicted)
gb_classification_report = classification_report(y_test,y_predicted)
print(f'Confusion Matrix: \n {gb_conf_matrix}')
```

```
print(f'Classification Report: \n {gb_classification_report}')
print(f'Accuracy: \n {gb_accuracy}')
```

```
Confusion Matrix:
 [[68  1  5]
 [ 4 69  7]
 [ 1  5 60]]
Classification Report:
               precision    recall  f1-score   support

           0       0.93      0.92      0.93        74
           1       0.92      0.86      0.89        80
           2       0.83      0.91      0.87        66

    accuracy                           0.90       220
   macro avg       0.89      0.90      0.90       220
weighted avg       0.90      0.90      0.90       220

Accuracy:
 0.8636363636363636
```

```
#testing with aurak data
y_pred = gb_classifier.predict(Aurak_df_x)
print("Accuracy:", accuracy_score(Aurak_df_y, y_pred))
print("confusion Matrix: ",confusion_matrix(Aurak_df_y,y_pred))
print("classification_report:
",classification_report(Aurak_df_y,y_pred))
```

```
Accuracy: 0.5576923076923077
confusion Matrix:  [[ 0  6  0]
 [ 1 28  1]
 [ 1 14  1]]
classification_report:                precision    recall  f1-score
support

           0       0.00      0.00      0.00         6
           1       0.58      0.93      0.72        30
           2       0.50      0.06      0.11        16

    accuracy                           0.56        52
   macro avg       0.36      0.33      0.28        52
weighted avg       0.49      0.56      0.45        52
```

Random forest

```
accuracy =[]
for i in range(1,20):
  rf_classifier =
RandomForestClassifier(n_estimators=5,criterion='entropy',max_depth=i,
```

```
random_state=2)
  rf_classifier.fit(x_train,y_train)
  predicted_y = rf_classifier.predict(x_test)
  accuracy.append(accuracy_score(predicted_y,y_test))

best_depth = accuracy.index(max(accuracy)) + 1
print(f'Best depth is {best_depth}')
```

```
Best depth is 1
```

```
#normal random forest
rf_classifier =
RandomForestClassifier(n_estimators=10,criterion='entropy',max_depth=b
est_depth,random_state=0)
rf_classifier.fit(x_train,y_train)
predicted_y = rf_classifier.predict(x_test)
cv_scores = cross_val_score(gb_classifier, x_test, y_test, cv=5,
scoring='accuracy')
rf_accuracy = cv_scores.mean()
rf_conf_matrix = confusion_matrix(y_test,y_predicted)
rf_classification_report = classification_report(y_test,y_predicted)
print(f'Confusion Matrix: \n {rf_conf_matrix}')
print(f'Classification Report: \n {rf_classification_report}')
print(f'Accuracy: \n {rf_accuracy}')
```

```
Confusion Matrix:
 [[68  1  5]
 [ 4 69  7]
 [ 1  5 60]]
Classification Report:
               precision    recall  f1-score   support

           0       0.93      0.92      0.93        74
           1       0.92      0.86      0.89        80
           2       0.83      0.91      0.87        66

    accuracy                           0.90       220
   macro avg       0.89      0.90      0.90       220
weighted avg       0.90      0.90      0.90       220

Accuracy:
 0.8636363636363636
```

```
y_pred = rf_classifier.predict(Aurak_df_x)
print("Accuracy:", accuracy_score(Aurak_df_y, y_pred))
print("confusion Matrix: ",confusion_matrix(Aurak_df_y,y_pred))
print("classification_report:
",classification_report(Aurak_df_y,y_pred))
```

```
Accuracy: 0.40384615384615385
confusion Matrix:  [[ 3  3  0]
```

```
 [ 9 12  9]
 [ 0 10  6]]
classification_report:                                precision    recall  f1-score   support

           0       0.25      0.50      0.33         6
           1       0.48      0.40      0.44        30
           2       0.40      0.38      0.39        16

    accuracy                           0.40        52
   macro avg       0.38      0.42      0.39        52
weighted avg       0.43      0.40      0.41        52
```

using pca to see if i get better results

```
pca = PCA()
pca.fit(x_train_scaled)
n_components = np.argmax(np.cumsum(pca.explained_variance_ratio_) >= 0.90) + 1

pca = PCA(n_components=n_components)
x_train_pca = pca.fit_transform(x_train_scaled)
x_test_pca = pca.transform(x_test_scaled)

knn_model = KNeighborsClassifier(n_neighbors = 3)
knn_model.fit(x_train_pca,y_train)
knn_model_y_predicted = knn_model.predict(x_test_pca)
cv_scores = cross_val_score(knn_model, x_test_pca, y_test, cv=5, scoring='accuracy')
knn_accuracy = cv_scores.mean()
knn_conf_matrix = confusion_matrix(y_test,knn_model_y_predicted)
knn_classification_report = classification_report(y_test,knn_model_y_predicted)
print(f'Confusion Matrix: \n {knn_conf_matrix}')
print(f'Classification Report: \n {knn_classification_report}')
print(f'Accuracy: \n {knn_accuracy}')
```

```
Confusion Matrix:
 [[66  2  6]
 [ 5 69  6]
 [ 6  0 60]]
Classification Report:
               precision    recall  f1-score   support

           0       0.86      0.89      0.87        74
           1       0.97      0.86      0.91        80
           2       0.83      0.91      0.87        66

    accuracy                           0.89       220
```

```
   macro avg       0.89      0.89      0.89       220
weighted avg       0.89      0.89      0.89       220

Accuracy:
 0.8818181818181818
```

```
gb_classifier =
GradientBoostingClassifier(n_estimators=5,learning_rate=1.0,max_depth=
2,random_state=0)
gb_classifier.fit(x_train_pca,y_train)
y_predicted = gb_classifier.predict(x_test_pca)
cv_scores = cross_val_score(gb_classifier, x_test_pca, y_test, cv=5,
scoring='accuracy')
gb_accuracy = cv_scores.mean()
gb_conf_matrix = confusion_matrix(y_test,y_predicted)
gb_classification_report = classification_report(y_test,y_predicted)
print(f'Confusion Matrix: \n {gb_conf_matrix}')
print(f'Classification Report: \n {gb_classification_report}')
print(f'Accuracy: \n {gb_accuracy}')
```

```
Confusion Matrix:
 [[62  6  6]
 [ 1 72  7]
 [ 2  2 62]]
Classification Report:
               precision    recall  f1-score   support

           0       0.95      0.84      0.89        74
           1       0.90      0.90      0.90        80
           2       0.83      0.94      0.88        66

    accuracy                           0.89       220
   macro avg       0.89      0.89      0.89       220
weighted avg       0.90      0.89      0.89       220

Accuracy:
 0.8818181818181818
```

```
accuracy =[]
for i in range(1,20):
  rf_classifier =
RandomForestClassifier(n_estimators=10,criterion='entropy',max_depth=i
,random_state=2)
  rf_classifier.fit(x_train_pca,y_train)
  predicted_y = rf_classifier.predict(x_test_pca)
  accuracy.append(accuracy_score(predicted_y,y_test))

best_depth = accuracy.index(max(accuracy)) + 1
print(f'Best depth is {best_depth}')
```

```
Best depth is 7
```

```
rf_classifier =
RandomForestClassifier(n_estimators=10,criterion='entropy',max_depth=b
est_depth,random_state=0)
rf_classifier.fit(x_train_pca,y_train)
predicted_y = rf_classifier.predict(x_test_pca)
cv_scores = cross_val_score(rf_classifier, x_test_pca, y_test, cv=5,
scoring='accuracy')
rf_accuracy = cv_scores.mean()
rf_conf_matrix = confusion_matrix(y_test,y_predicted)
rf_classification_report = classification_report(y_test,y_predicted)
print(f'Confusion Matrix: \n {rf_conf_matrix}')
print(f'Classification Report: \n {rf_classification_report}')
print(f'Accuracy: \n {rf_accuracy}')
```

```
Confusion Matrix:
 [[62  6  6]
 [ 1 72  7]
 [ 2  2 62]]
Classification Report:
               precision    recall  f1-score   support

           0       0.95      0.84      0.89        74
           1       0.90      0.90      0.90        80
           2       0.83      0.94      0.88        66

    accuracy                           0.89       220
   macro avg       0.89      0.89      0.89       220
weighted avg       0.90      0.89      0.89       220

Accuracy:
 0.8590909090909091
```

trying lasso to see if i get better results

```
lasso_cv = LassoCV(cv=10, random_state=0)
lasso_cv.fit(x_train_scaled, y_train)

best_alpha = lasso_cv.alpha_
print (f"best alpha = {best_alpha}")

lasso = Lasso(alpha=best_alpha)
lasso.fit(x_train_scaled, y_train)
```

```
best alpha = 0.01029099088925991
```

```
Lasso(alpha=0.01029099088925991)
```

```
important_features = [feature for coef, feature in zip(lasso.coef_,
x_train.columns) if coef != 0]
print(f'Important features after scaling: {important_features}')
```

```
Important features after scaling: ['anxiety_level', 'self_esteem',
'depression', 'headache', 'sleep_quality', 'noise_level',
'living_conditions', 'safety', 'basic_needs', 'academic_performance',
'study_load', 'future_career_concerns', 'social_support',
'peer_pressure', 'extracurricular_activities', 'bullying']

x_train_reduced = x_train[important_features]
x_test_reduced = x_test[important_features]

knn_model = KNeighborsClassifier(n_neighbors = 3)
knn_model.fit(x_train_reduced,y_train)
knn_model_y_predicted = knn_model.predict(x_test_reduced)
cv_scores = cross_val_score(knn_model, x_test_reduced, y_test, cv=5,
scoring='accuracy')
knn_accuracy = cv_scores.mean()
knn_conf_matrix = confusion_matrix(y_test,knn_model_y_predicted)
knn_classification_report =
classification_report(y_test,knn_model_y_predicted)
print(f'Confusion Matrix: \n {knn_conf_matrix}')
print(f'Classification Report: \n {knn_classification_report}')
print(f'Accuracy: \n {knn_accuracy}')

Confusion Matrix:
 [[63  3  8]
 [ 2 74  4]
 [ 3  3 60]]
Classification Report:
               precision    recall  f1-score   support

           0       0.93      0.85      0.89        74
           1       0.93      0.93      0.93        80
           2       0.83      0.91      0.87        66

    accuracy                           0.90       220
   macro avg       0.89      0.90      0.89       220
weighted avg       0.90      0.90      0.90       220

Accuracy:
 0.8863636363636364

gb_classifier =
GradientBoostingClassifier(n_estimators=10,learning_rate=1.0,max_depth
=2,random_state=0)
gb_classifier.fit(x_train_reduced,y_train)
y_predicted = gb_classifier.predict(x_test_reduced)
cv_scores = cross_val_score(gb_classifier, x_test_reduced, y_test,
cv=5, scoring='accuracy')
gb_accuracy = cv_scores.mean()
gb_conf_matrix = confusion_matrix(y_test,y_predicted)
gb_classification_report = classification_report(y_test,y_predicted)
```

```
print(f'Confusion Matrix: \n {gb_conf_matrix}')
print(f'Classification Report: \n {gb_classification_report}')
print(f'Accuracy: \n {gb_accuracy}')
```

```
Confusion Matrix:
 [[63  4  7]
 [ 5 69  6]
 [ 4  1 61]]
Classification Report:
               precision    recall  f1-score   support

           0       0.88      0.85      0.86        74
           1       0.93      0.86      0.90        80
           2       0.82      0.92      0.87        66

    accuracy                           0.88       220
   macro avg       0.88      0.88      0.88       220
weighted avg       0.88      0.88      0.88       220

Accuracy:
 0.8727272727272727
```

```
accuracy =[]
for i in range(1,20):
  rf_classifier = 
RandomForestClassifier(n_estimators=10,criterion='entropy',max_depth=i
,random_state=2)
  rf_classifier.fit(x_train_reduced,y_train)
  predicted_y = rf_classifier.predict(x_test_reduced)
  accuracy.append(accuracy_score(predicted_y,y_test))

best_depth = accuracy.index(max(accuracy)) + 1
print(f'Best depth is {best_depth}')
```

```
Best depth is 8
```

```
#lasso random forest
rf_classifier = 
RandomForestClassifier(n_estimators=10,criterion='entropy',max_depth=b
est_depth,random_state=0)
rf_classifier.fit(x_train_reduced,y_train)
predicted_y = rf_classifier.predict(x_test_reduced)
cv_scores = cross_val_score(rf_classifier, x_test_reduced, y_test, 
cv=5, scoring='accuracy')
rf_accuracy = cv_scores.mean()
rf_conf_matrix = confusion_matrix(y_test,y_predicted)
rf_classification_report = classification_report(y_test,y_predicted)
print(f'Confusion Matrix: \n {rf_conf_matrix}')
print(f'Classification Report: \n {rf_classification_report}')
print(f'Accuracy: \n {rf_accuracy}')
```

```
Confusion Matrix:
 [[63  4  7]
 [ 5 69  6]
 [ 4  1 61]]
Classification Report:
               precision    recall  f1-score   support

           0       0.88      0.85      0.86        74
           1       0.93      0.86      0.90        80
           2       0.82      0.92      0.87        66

    accuracy                           0.88       220
   macro avg       0.88      0.88      0.88       220
weighted avg       0.88      0.88      0.88       220

Accuracy:
 0.8636363636363636
```