```
# IMPORTANT: RUN THIS CELL IN ORDER TO IMPORT YOUR KAGGLE DATA SOURCES
# TO THE CORRECT LOCATION (/kaggle/input) IN YOUR NOTEBOOK,
# THEN FEEL FREE TO DELETE THIS CELL.
# NOTE: THIS NOTEBOOK ENVIRONMENT DIFFERS FROM KAGGLE'S PYTHON
# ENVIRONMENT SO THERE MAY BE MISSING LIBRARIES USED BY YOUR
# NOTEBOOK.

import os
import sys
from tempfile import NamedTemporaryFile
from urllib.request import urlopen
from urllib.parse import unquote, urlparse
from urllib.error import HTTPError
from zipfile import ZipFile
import tarfile
import shutil

CHUNK_SIZE = 40960
DATA_SOURCE_MAPPING = 'student-stress-factors-a-comprehensive-analysis:https%3A%2F%2Fstorage.googleapis.com%2Fkaggle-data-se

KAGGLE_INPUT_PATH='/kaggle/input'
KAGGLE_WORKING_PATH='/kaggle/working'
KAGGLE_SYMLINK='kaggle'

!umount /kaggle/input/ 2> /dev/null
shutil.rmtree('/kaggle/input', ignore_errors=True)
os.makedirs(KAGGLE_INPUT_PATH, 0o777, exist_ok=True)
os.makedirs(KAGGLE_WORKING_PATH, 0o777, exist_ok=True)

try:
  os.symlink(KAGGLE_INPUT_PATH, os.path.join("..", 'input'), target_is_directory=True)
except FileExistsError:
  pass
try:
  os.symlink(KAGGLE_WORKING_PATH, os.path.join("..", 'working'), target_is_directory=True)
except FileExistsError:
  pass

for data_source_mapping in DATA_SOURCE_MAPPING.split(','):
    directory, download_url_encoded = data_source_mapping.split(':')
    download_url = unquote(download_url_encoded)
    filename = urlparse(download_url).path
    destination_path = os.path.join(KAGGLE_INPUT_PATH, directory)
    try:
        with urlopen(download_url) as fileres, NamedTemporaryFile() as tfile:
            total_length = fileres.headers['content-length']
            print(f'Downloading {directory}, {total_length} bytes compressed')
            dl = 0
            data = fileres.read(CHUNK_SIZE)
            while len(data) > 0:
                dl += len(data)
                tfile.write(data)
                done = int(50 * dl / int(total_length))
                sys.stdout.write(f"\r[{'=' * done}{' ' * (50-done)}] {dl} bytes downloaded")
                sys.stdout.flush()
                data = fileres.read(CHUNK_SIZE)
            if filename.endswith('.zip'):
              with ZipFile(tfile) as zfile:
                zfile.extractall(destination_path)
            else:
              with tarfile.open(tfile.name) as tarfile:
                tarfile.extractall(destination_path)
            print(f'\nDownloaded and uncompressed: {directory}')
    except HTTPError as e:
        print(f'Failed to load (likely expired) {download_url} to path {destination_path}')
        continue
    except OSError as e:
        print(f'Failed to load {download_url} to path {destination_path}')
        continue

print('Data source import complete.')
```

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
```

```
# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a versi
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session
```

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
import numpy as np
```

```
data=pd.read_csv("StressLevelDataset.csv")
data_df=pd.DataFrame(data)
data_df.head()
```

	anxiety_level	self_esteem	mental_health_history	depression	headache	blo
0	14	20	0	11	2	
1	15	8	1	15	5	
2	12	18	1	14	2	
3	16	12	1	15	4	
4	16	28	0	7	2	

5 rows × 21 columns

Exploratory Data Analysis

```
data_df.info()
```

Show hidden output

```
data_df.describe()
```

	anxiety_level	self_esteem	mental_health_history	depression	headach
count	1100.000000	1100.000000	1100.000000	1100.000000	1100.00000
mean	11.063636	17.777273	0.492727	12.555455	2.50818
std	6.117558	8.944599	0.500175	7.727008	1.40935
min	0.000000	0.000000	0.000000	0.000000	0.00000
25%	6.000000	11.000000	0.000000	6.000000	1.00000
50%	11.000000	19.000000	0.000000	12.000000	3.00000
75%	16.000000	26.000000	1.000000	19.000000	3.00000
max	21.000000	30.000000	1.000000	27.000000	5.00000

8 rows × 21 columns

```
data_df.duplicated().sum()
```

```
0
```

```
data_df.isnull().sum()
```

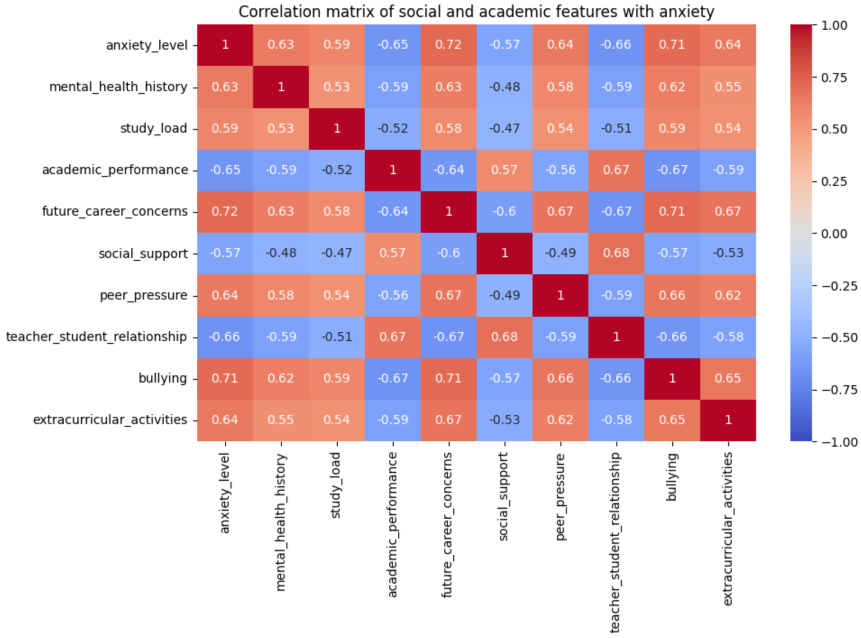
```
anxiety_level                   0
self_esteem                     0
mental_health_history           0
depression                      0
headache                        0
blood_pressure                  0
sleep_quality                   0
breathing_problem               0
noise_level                     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
```

```
#correlation matrix to identify best correlated social and environmental features with anxiety,depression and stress separat
features=["anxiety_level",'mental_health_history',"study_load","academic_performance",'future_career_concerns',"social_suppo
corr_matrix=data_df[features].corr()
print(corr_matrix)
```

Show hidden output

```
plt.figure(figsize=(10,6))
sns.heatmap(corr_matrix,annot=True,cmap="coolwarm",vmin=-1,vmax=1)
plt.title("Correlation matrix of social and academic features with anxiety")
plt.show()
```



```
print(corr_matrix["anxiety_level"].sort_values(ascending=False))
```

```
anxiety_level                   1.000000
future_career_concerns          0.717016
bullying                        0.709982
peer_pressure                   0.642910
extracurricular_activities      0.641022
mental_health_history           0.634450
study_load                      0.586064
social_support                 -0.569748
academic_performance           -0.649601
teacher_student_relationship   -0.663176
Name: anxiety_level, dtype: float64
```

```
#correlation matrix to identify best correlated social and environmental features with anxiety,depression and stress separat
features=["stress_level",'mental_health_history',"study_load","academic_performance",'future_career_concerns',"social_suppor
```

```
corr_matrix2=data_df[features].corr()
print(corr_matrix2)
```

Show hidden output

```
plt.figure(figsize=(10,6))
sns.heatmap(corr_matrix2,annot=True,cmap="coolwarm",vmin=-1,vmax=1)
plt.title("Correlation matrix of social and academic features with stress")
plt.show()
```

Show hidden output

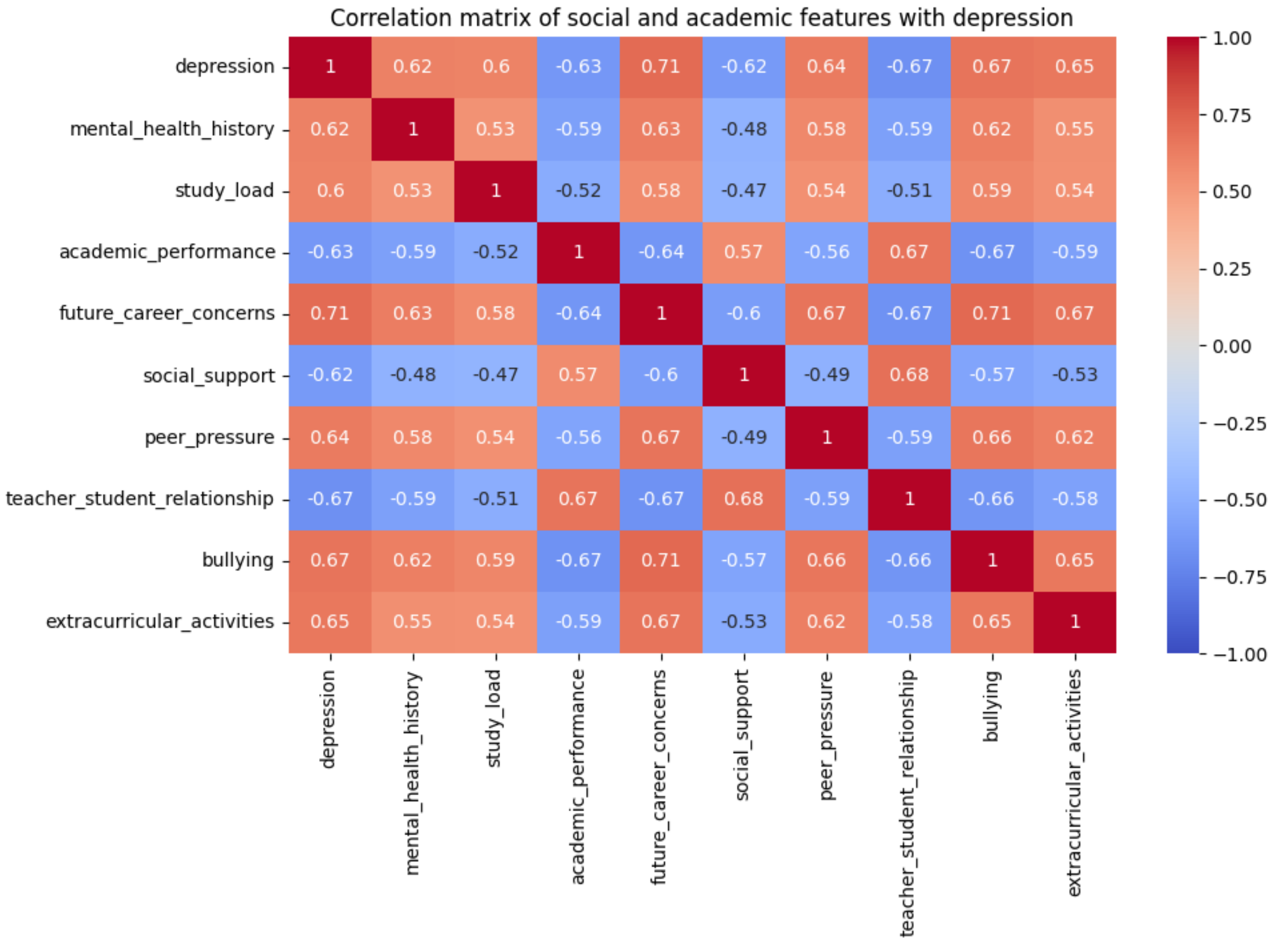
```
print(corr_matrix2["stress_level"].sort_values(ascending=False))
```

```
stress_level                   1.000000
bullying                       0.751162
future_career_concerns         0.742619
extracurricular_activities     0.692977
peer_pressure                  0.690684
mental_health_history          0.648644
study_load                     0.634156
social_support                -0.632497
teacher_student_relationship  -0.680163
academic_performance          -0.720922
Name: stress_level, dtype: float64
```

```
#correlation matrix to identify best correlated social and environmental features with anxiety,depression and stress separat
features=["depression",'mental_health_history',"study_load","academic_performance",'future_career_concerns',"social_support"
corr_matrix3=data_df[features].corr()
print(corr_matrix3)
```

Show hidden output

```
plt.figure(figsize=(10,6))
sns.heatmap(corr_matrix3,annot=True,cmap="coolwarm",vmin=-1,vmax=1)
plt.title("Correlation matrix of social and academic features with depression")
plt.show()
```



Correlation matrix of social and academic features with depression

	depression	mental_health_history	study_load	academic_performance	future_career_concerns	social_support	peer_pressure	teacher_student_relationship	bullying	extracurricular_activities
depression	1	0.62	0.6	-0.63	0.71	-0.62	0.64	-0.67	0.67	0.65
mental_health_history	0.62	1	0.53	-0.59	0.63	-0.48	0.58	-0.59	0.62	0.55
study_load	0.6	0.53	1	-0.52	0.58	-0.47	0.54	-0.51	0.59	0.54
academic_performance	-0.63	-0.59	-0.52	1	-0.64	0.57	-0.56	0.67	-0.67	-0.59
future_career_concerns	0.71	0.63	0.58	-0.64	1	-0.6	0.67	-0.67	0.71	0.67
social_support	-0.62	-0.48	-0.47	0.57	-0.6	1	-0.49	0.68	-0.57	-0.53
peer_pressure	0.64	0.58	0.54	-0.56	0.67	-0.49	1	-0.59	0.66	0.62
teacher_student_relationship	-0.67	-0.59	-0.51	0.67	-0.67	0.68	-0.59	1	-0.66	-0.58
bullying	0.67	0.62	0.59	-0.67	0.71	-0.57	0.66	-0.66	1	0.65
extracurricular_activities	0.65	0.55	0.54	-0.59	0.67	-0.53	0.62	-0.58	0.65	1

```
print(corr_matrix3["depression"].sort_values(ascending=False))
```

```
depression                      1.000000
future_career_concerns          0.706561
bullying                        0.665790
extracurricular_activities      0.648551
peer_pressure                   0.635544
mental_health_history           0.615882
study_load                      0.602498
social_support                 -0.617972
academic_performance           -0.633174
teacher_student_relationship   -0.673853
Name: depression, dtype: float64
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split,cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,classification_report,confusion_matrix
```

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(data_df)
```

```
# Split the data into features and target
X = data_df.iloc[:,:-1]
y = data_df['stress_level']
x_train,x_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=42)
x_train.shape,x_test.shape,y_train.shape,y_test.shape
```

```
((880, 20), (220, 20), (880,), (220,))
```

```
model=LogisticRegression()
model.fit(x_train,y_train)
y_pred=model.predict(x_test)
accuracy=accuracy_score(y_test,y_pred)
print("Accuracy:",accuracy)
```

```
Accuracy: 0.8818181818181818
```

```
y_test
```

```
328    2
688    0
413    1
788    1
244    0
      ..
319    1
979    2
919    2
989    0
724    1
Name: stress_level, Length: 220, dtype: int64
```

```
y_pred
```

```
array([2, 0, 1, 1, 0, 1, 2, 0, 1, 0, 0, 0, 0, 2, 1, 0, 0, 2, 1, 0, 1, 1,
       0, 1, 1, 2, 0, 0, 0, 0, 2, 2, 2, 0, 0, 1, 2, 1, 2, 2, 2, 0, 1, 2,
       2, 0, 2, 2, 0, 0, 1, 0, 0, 0, 1, 1, 2, 2, 1, 1, 2, 0, 2, 0, 2, 0,
       2, 1, 2, 2, 0, 1, 2, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 2, 2, 2,
       2, 1, 0, 2, 0, 1, 0, 0, 0, 0, 1, 1, 2, 0, 0, 2, 0, 1, 1, 2, 1, 2,
       2, 2, 1, 0, 2, 0, 2, 2, 0, 0, 2, 1, 2, 1, 2, 2, 0, 2, 0, 2, 2, 1,
       0, 0, 0, 2, 1, 2, 2, 2, 0, 2, 0, 1, 0, 0, 2, 1, 0, 2, 1, 1, 2, 1,
       1, 1, 2, 0, 2, 0, 0, 0, 0, 0, 1, 0, 1, 2, 0, 1, 1, 0, 2, 2, 0, 0,
       0, 2, 2, 0, 1, 0, 1, 1, 1, 0, 2, 1, 2, 1, 2, 1, 2, 1, 1, 1, 0, 1,
       1, 2, 2, 1, 0, 0, 1, 2, 2, 2, 2, 0, 2, 1, 2, 0, 0, 1, 2, 1, 0, 1])
```

Our logistic regression model gave accuarcy of 0.8 i.e 80% which is good but we will try other methods as well to check which algorithm gives the best accuracy score.

Descision tree

```
from sklearn.tree import DecisionTreeClassifier
# Create a Decision Tree classifier object
clf=DecisionTreeClassifier(random_state=42)
# Train the classifier on the training data
clf.fit(x_train,y_train)
# Display a message indicating that the model has been trained
print('Decision Tree model has been trained')
```

```
Decision Tree model has been trained
```

```
y_pred=clf.predict(x_test)
accuracy=accuracy_score(y_test,y_pred)
print("Accuracy:",accuracy)
```

```
Accuracy: 0.8863636363636364
```

Double-click (or enter) to edit

```
# Classification report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.91      0.89      0.90        76
           1       0.87      0.90      0.89        73
           2       0.88      0.86      0.87        71

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

```
# Cross validation
cv_scores = cross_val_score(clf, x_train, y_train, cv=10)
print("Cross Validation Scores:", cv_scores)
print("Mean Cross Validation Score:", np.mean(cv_scores))
```

```
Cross Validation Scores: [0.93181818 0.88636364 0.88636364 0.81818182 0.875      0.89772727
 0.89772727 0.875      0.89772727 0.85227273]
Mean Cross Validation Score: 0.8818181818181818
```

Random Forest descision tree

```
from sklearn.ensemble import RandomForestClassifier
```

```
rf_clf_data=RandomForestClassifier(n_estimators=100,random_state=42)
rf_clf_data.fit(x_train,y_train)
y_pred=rf_clf_data.predict(x_test)
accuracy=accuracy_score(y_test,y_pred)
print("Accuracy:",accuracy)
```

```
Accuracy: 0.8727272727272727
```

```
y_test
```

```
328    2
688    0
413    1
788    1
244    0
      ..
319    1
979    2
919    2
989    0
724    1
Name: stress_level, Length: 220, dtype: int64
```

```
y_pred
```

```
array([2, 0, 1, 1, 0, 1, 2, 0, 1, 1, 0, 0, 0, 2, 2, 0, 0, 2, 1, 0, 1, 1,
       0, 1, 1, 2, 0, 0, 0, 0, 2, 2, 2, 0, 0, 1, 0, 1, 2, 2, 2, 0, 1, 1,
       2, 0, 2, 1, 0, 0, 1, 0, 0, 0, 1, 1, 2, 1, 1, 1, 2, 0, 1, 0, 0, 0,
       2, 1, 2, 1, 0, 1, 0, 1, 0, 2, 1, 0, 1, 2, 1, 0, 0, 1, 1, 2, 2, 2,
       2, 1, 0, 2, 0, 1, 2, 0, 0, 0, 1, 1, 2, 0, 0, 2, 0, 1, 1, 2, 1, 2,
       2, 2, 1, 0, 0, 0, 2, 2, 0, 0, 2, 1, 2, 1, 2, 2, 0, 2, 0, 2, 2, 1,
       0, 0, 0, 2, 1, 2, 2, 2, 0, 2, 0, 1, 0, 0, 2, 0, 0, 2, 1, 1, 2, 1,
       1, 1, 2, 0, 2, 0, 0, 0, 0, 0, 1, 0, 1, 2, 0, 1, 1, 0, 2, 2, 0, 0,
       0, 2, 2, 0, 1, 0, 1, 1, 1, 0, 2, 1, 2, 1, 2, 1, 2, 2, 1, 1, 0, 1,
       1, 2, 2, 1, 0, 0, 1, 2, 2, 2, 2, 0, 0, 1, 2, 0, 0, 1, 2, 1, 0, 1])
```

```
# Cross validation
cv_scores = cross_val_score(rf_clf_data, x_train, y_train, cv=10)
print("Cross Validation Scores:", cv_scores)
print("Mean Cross Validation Score:", np.mean(cv_scores))
```

```
Cross Validation Scores: [0.94318182 0.875      0.86363636 0.82954545 0.90909091 0.89772727
 0.875      0.82954545 0.875      0.80681818]
Mean Cross Validation Score: 0.8704545454545455
```

Support Vector Machine

```
from sklearn.svm import SVC
clf_svm=SVC(kernel='linear',C=1,random_state=42)
clf_svm.fit(x_train,y_train)
y_pred=clf_svm.predict(x_test)
accuracy=accuracy_score(y_test,y_pred)
print("Accuracy:",accuracy)
```

```
Accuracy: 0.8909090909090909
```

```
So among all the algorithms tested  SVM gave the highest accur
scores of 0.89 so it is the best method to classifiy the stres
levels of students based on number of factors.
```

So among all the algorithms tested SVM gave the highest accuracy scores of 0.89 so it is the best method to classifiy the stress levels of students based on number of factors.