This Is My 23.Student Depression Project.

Import Required Libraries

In [1]:
```
# Data handling
import numpy as np
import pandas as pd

# Visualization
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

# Preprocessing & splitting
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import make_pipeline,Pipeline
from sklearn.impute import SimpleImputer
from sklearn.utils import resample

# Evaluation metrics
from sklearn.metrics import (accuracy_score,precision_score,recall_score,f1_scor
    classification_report,confusion_matrix,auc,roc_auc_score, roc_curve
)

# Classifier Models
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import (
    RandomForestClassifier, GradientBoostingClassifier,
    AdaBoostClassifier, ExtraTreesClassifier
)
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier

# XGBoost
from xgboost import XGBClassifier

# Warnings
import warnings
warnings.filterwarnings("ignore")

# Save the model
import joblib
```

Load Dataset

In [2]:
```
data = pd.read_csv('student_depression_dataset.csv')
data.head()
```

Out[2]:

	id	Gender	Age	City	Profession	Academic Pressure	Work Pressure	CGPA	Study Satisfactio
0	2	Male	33.0	Visakhapatnam	Student	5.0	0.0	8.97	2.(
1	8	Female	24.0	Bangalore	Student	2.0	0.0	5.90	5.(
2	26	Male	31.0	Srinagar	Student	3.0	0.0	7.03	5.(
3	30	Female	28.0	Varanasi	Student	3.0	0.0	5.59	2.(
4	32	Female	25.0	Jaipur	Student	4.0	0.0	8.13	3.(

EDA

In [3]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 27901 entries, 0 to 27900
Data columns (total 18 columns):
 #   Column                                 Non-Null Count  Dtype
---  ------                                 --------------  -----
 0   id                                     27901 non-null  int64
 1   Gender                                 27901 non-null  object
 2   Age                                    27901 non-null  float64
 3   City                                   27901 non-null  object
 4   Profession                             27901 non-null  object
 5   Academic Pressure                      27901 non-null  float64
 6   Work Pressure                          27901 non-null  float64
 7   CGPA                                   27901 non-null  float64
 8   Study Satisfaction                     27901 non-null  float64
 9   Job Satisfaction                       27901 non-null  float64
 10  Sleep Duration                         27901 non-null  object
 11  Dietary Habits                         27901 non-null  object
 12  Degree                                 27901 non-null  object
 13  Have you ever had suicidal thoughts ?  27901 non-null  object
 14  Work/Study Hours                       27901 non-null  float64
 15  Financial Stress                       27901 non-null  object
 16  Family History of Mental Illness       27901 non-null  object
 17  Depression                             27901 non-null  int64
dtypes: float64(7), int64(2), object(9)
memory usage: 3.8+ MB
```

Checking For Duplicated Rows & Columns

In [4]:
```
print(" Number of duplicated rows:", data.duplicated().sum())

print(" Number of duplicated columns:", data.columns.duplicated().sum())
```

```
Number of duplicated rows: 0
Number of duplicated columns: 0
```

Checking for object unique value number

In [5]:
```
cat = data.select_dtypes(include='object')
for i in cat.columns:
    print(f"{i}")
    print(f"{cat[i].nunique()}")
```

```
Gender
2
City
52
Profession
14
Sleep Duration
5
Dietary Habits
4
Degree
28
Have you ever had suicidal thoughts ?
2
Financial Stress
6
Family History of Mental Illness
2
```

Checking for object unique value

In [6]:
```
for i in cat.columns:
    print(f"{i}")
    print(f"{cat[i].unique()}")
```

```
Gender
['Male' 'Female']
City
['Visakhapatnam' 'Bangalore' 'Srinagar' 'Varanasi' 'Jaipur' 'Pune' 'Thane'
 'Chennai' 'Nagpur' 'Nashik' 'Vadodara' 'Kalyan' 'Rajkot' 'Ahmedabad'
 'Kolkata' 'Mumbai' 'Lucknow' 'Indore' 'Surat' 'Ludhiana' 'Bhopal'
 'Meerut' 'Agra' 'Ghaziabad' 'Hyderabad' 'Vasai-Virar' 'Kanpur' 'Patna'
 'Faridabad' 'Delhi' 'Saanvi' 'M.Tech' 'Bhavna' "'Less Delhi'" 'City'
 '3.0' "'Less than 5 Kalyan'" 'Mira' 'Harsha' 'Vaanya' 'Gaurav' 'Harsh'
 'Reyansh' 'Kibara' 'Rashi' 'ME' 'M.Com' 'Nalyan' 'Mihir' 'Nalini'
 'Nandini' 'Khaziabad']
Profession
['Student' "'Civil Engineer'" 'Architect' "'UX/UI Designer'"
 "'Digital Marketer'" "'Content Writer'" "'Educational Consultant'"
 'Teacher' 'Manager' 'Chef' 'Doctor' 'Lawyer' 'Entrepreneur' 'Pharmacist']
Sleep Duration
["'5-6 hours'" "'Less than 5 hours'" "'7-8 hours'" "'More than 8 hours'"
 'Others']
Dietary Habits
['Healthy' 'Moderate' 'Unhealthy' 'Others']
Degree
['B.Pharm' 'BSc' 'BA' 'BCA' 'M.Tech' 'PhD' "'Class 12'" 'B.Ed' 'LLB' 'BE'
 'M.Ed' 'MSc' 'BHM' 'M.Pharm' 'MCA' 'MA' 'B.Com' 'MD' 'MBA' 'MBBS' 'M.Com'
 'B.Arch' 'LLM' 'B.Tech' 'BBA' 'ME' 'MHM' 'Others']
Have you ever had suicidal thoughts ?
['Yes' 'No']
Financial Stress
['1.0' '2.0' '5.0' '3.0' '4.0' '?']
Family History of Mental Illness
['No' 'Yes']
```

Replace '?' or 'nan'-like text with actual NaN

In [7]:
```
data.replace(['?', 'nan', 'None', 'Na', 'NaN'], np.nan, inplace=True)
for col in cat.columns:
    data[col] = data[col].fillna(data[col].mode()[0])
```

Fix column-specific issues

Financial Stress → convert to numeric safely

In [8]:
```
data['Financial Stress'] = pd.to_numeric(data['Financial Stress'], errors='coerc
data['Financial Stress'].value_counts()
```

Out[8]:
```
Financial Stress
5.0    6718
4.0    5775
3.0    5226
1.0    5121
2.0    5061
Name: count, dtype: int64
```

Define a function to extract numeric hours from Sleep Duration column

In [9]:
```
import re

def extract_hours(s):
    s = str(s)

    # If 'Others' → return 0
    if 'Others' in s:
        return 0

    # If 'Less than 5 hours' → use 5
    elif 'Less' in s:
        return 5

    # If 'More than 8 hours' → use 8
    elif 'More' in s:
        return 8

    # If a number (like '5-6 hours' or '7-8 hours')
    match = re.search(r"(\d+(\.\d+)?)", s)
    if match:
        return float(match.group(1))
    else:
        return np.nan  # if nothing found
```

Sleep Duration → clean extra quotes

In [10]:
```
data['Sleep Duration'] = data['Sleep Duration'].str.replace('"', '').str.replace
```

Apply the function

In [11]:
```
data['Sleep Duration'] = data['Sleep Duration'].apply(extract_hours)
```

Check the results

In [12]:
```
print(data['Sleep Duration'].value_counts())
```

```
Sleep Duration
5.0    14493
7.0     7346
8.0     6044
0.0       18
Name: count, dtype: int64
```

In [13]:
```
data['Sleep Duration'].unique()
```

Out[13]:
```
array([5., 7., 8., 0.])
```

Profession → clean extra quotes if any left

In [14]:
```
data['Profession'] = data['Profession'].str.replace('"', '').str.replace("'", ''
```

In [15]:
```
data['Financial Stress'].unique()
```

Out[15]:
```
array([1., 2., 5., 3., 4.])
```

In [16]:
```
data[['Financial Stress','Sleep Duration']] = data[['Financial Stress','Sleep Du
```

In [17]:
```
data.dtypes
```

Out[17]:
```
id                                       int64
Gender                                  object
Age                                    float64
City                                    object
Profession                              object
Academic Pressure                      float64
Work Pressure                          float64
CGPA                                   float64
Study Satisfaction                     float64
Job Satisfaction                       float64
Sleep Duration                           int64
Dietary Habits                          object
Degree                                  object
Have you ever had suicidal thoughts ?   object
Work/Study Hours                       float64
Financial Stress                         int64
Family History of Mental Illness        object
Depression                               int64
dtype: object
```

City → remove obvious noise (numeric or degree terms)

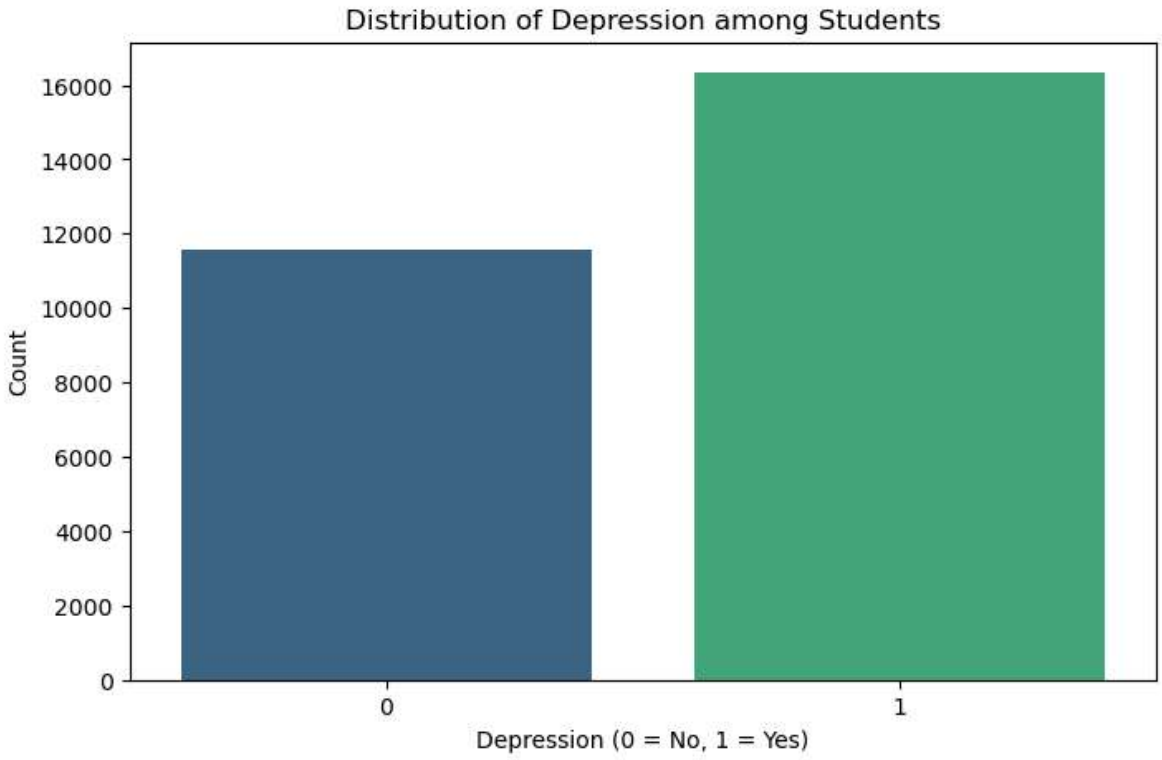
In [18]:
```
invalid_city_terms = ['3.0', 'M.Tech', 'M.Com', 'Me', 'City', 'Less Delhi', 'Les
data['City'] = data['City'].apply(lambda x: np.nan if any(term.lower() in str(x)
```

In [19]:
```
data['City'].unique()
```

Out[19]:
```
array(['Visakhapatnam', 'Bangalore', 'Srinagar', 'Varanasi', 'Jaipur',
       'Pune', 'Thane', 'Chennai', 'Nagpur', 'Nashik', 'Vadodara',
       'Kalyan', 'Rajkot', nan, 'Kolkata', 'Mumbai', 'Lucknow', 'Indore',
       'Surat', 'Ludhiana', 'Bhopal', 'Agra', 'Ghaziabad', 'Hyderabad',
       'Vasai-Virar', 'Kanpur', 'Patna', 'Faridabad', 'Delhi', 'Saanvi',
       'Bhavna', 'Mira', 'Harsha', 'Vaanya', 'Gaurav', 'Harsh', 'Reyansh',
       'Kibara', 'Rashi', 'Nalyan', 'Mihir', 'Nalini', 'Nandini',
       'Khaziabad'], dtype=object)
```

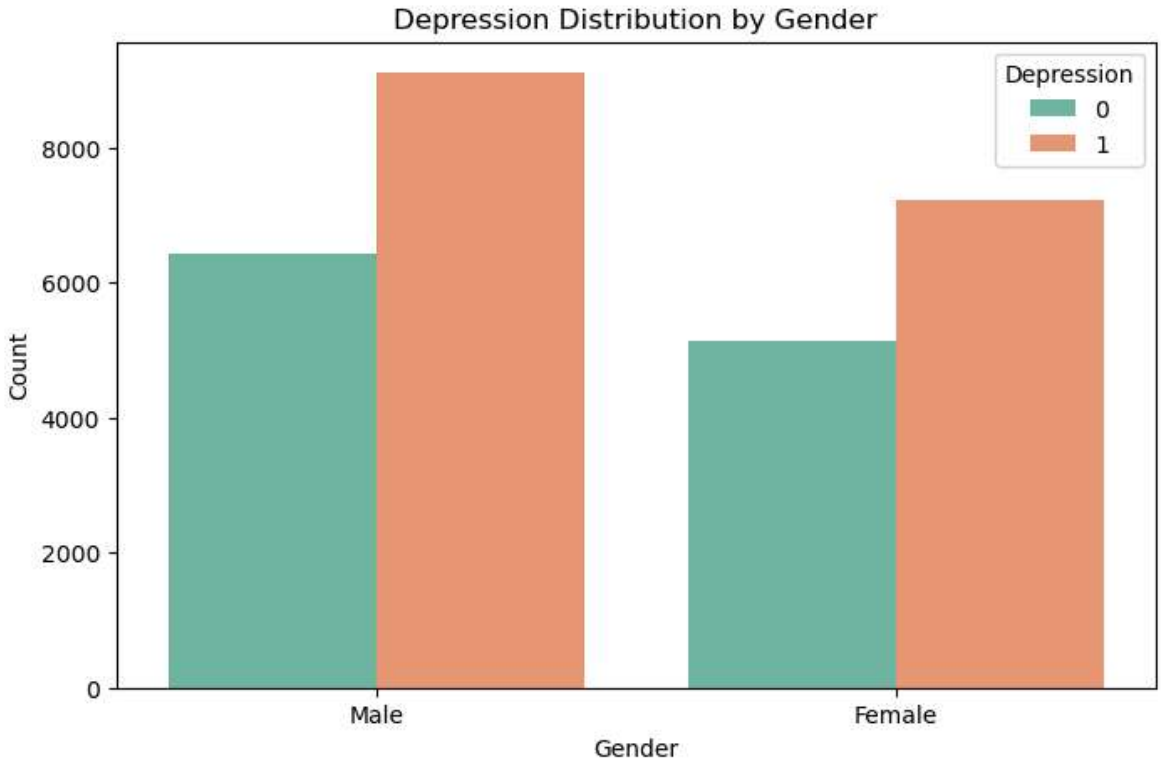
Distribution of Depression among Students

In [20]:
```
plt.figure(figsize=(8,5))
sns.countplot(x='Depression', data=data, palette="viridis")
plt.title("Distribution of Depression among Students")
plt.xlabel("Depression (0 = No, 1 = Yes)")
plt.ylabel("Count")
plt.show()
```

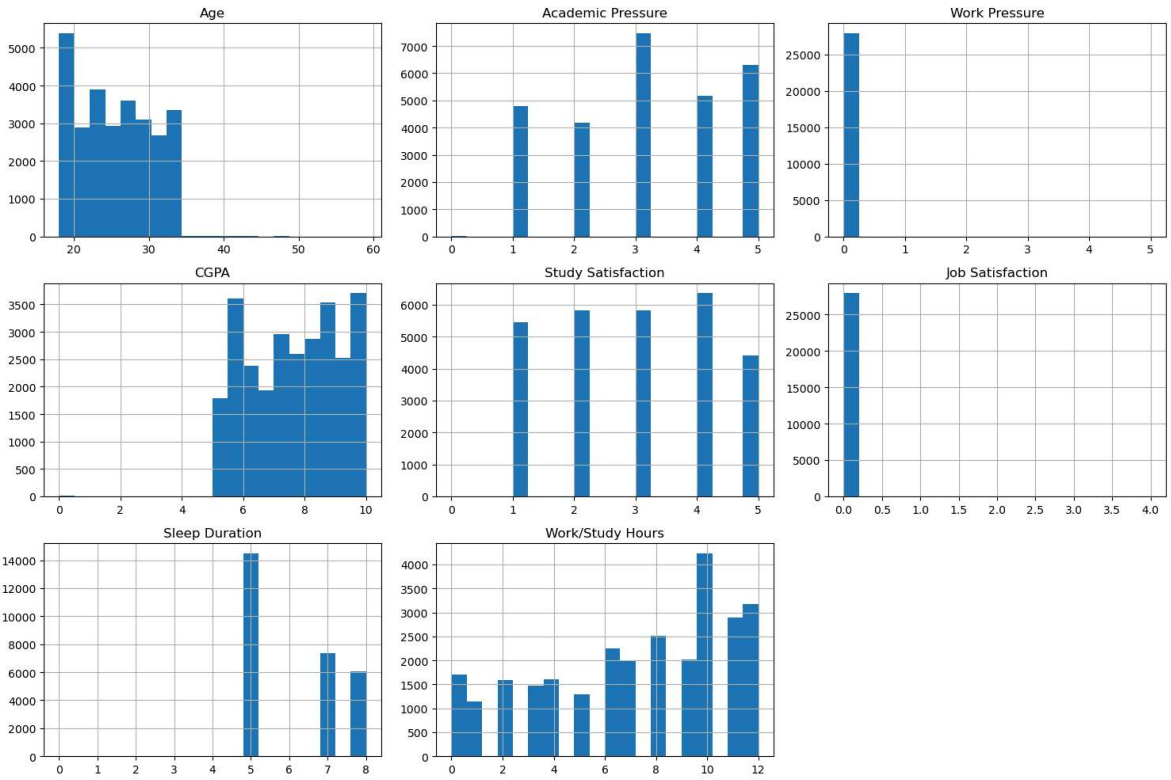


Depression Distribution by Gender

In [21]:
```
plt.figure(figsize=(8,5))
sns.countplot(x='Gender', hue='Depression', data=data, palette="Set2")
plt.title("Depression Distribution by Gender")
plt.xlabel("Gender")
plt.ylabel("Count")
plt.legend(title="Depression")
plt.show()
```



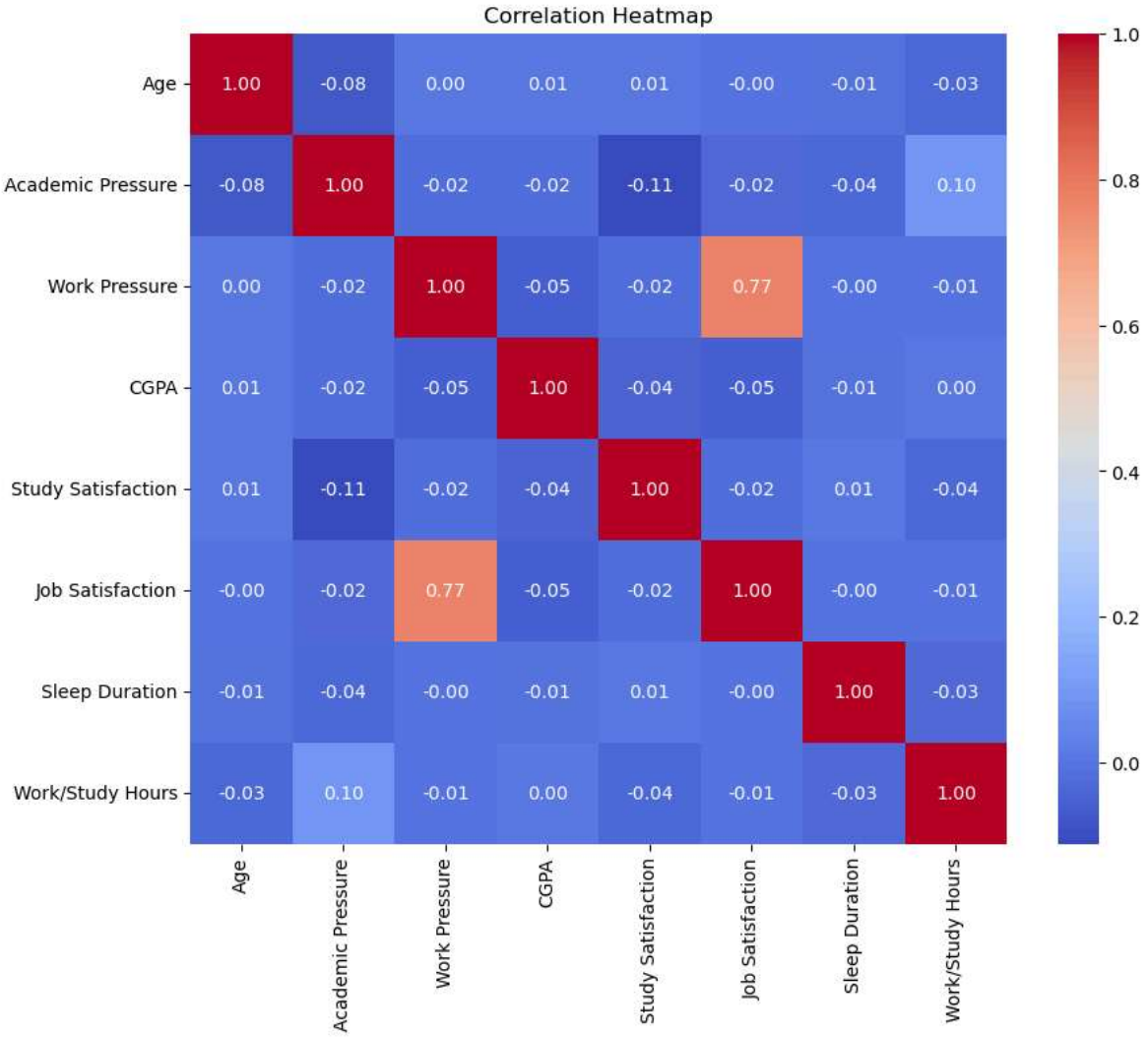
In [22]:
```
num_features = ['Age', 'Academic Pressure', 'Work Pressure', 'CGPA',
                'Study Satisfaction', 'Job Satisfaction', 'Sleep Duration', 'Wor
data[num_features].hist(bins=20, figsize=(15,10))
plt.tight_layout()
plt.show()
```



Heatmap

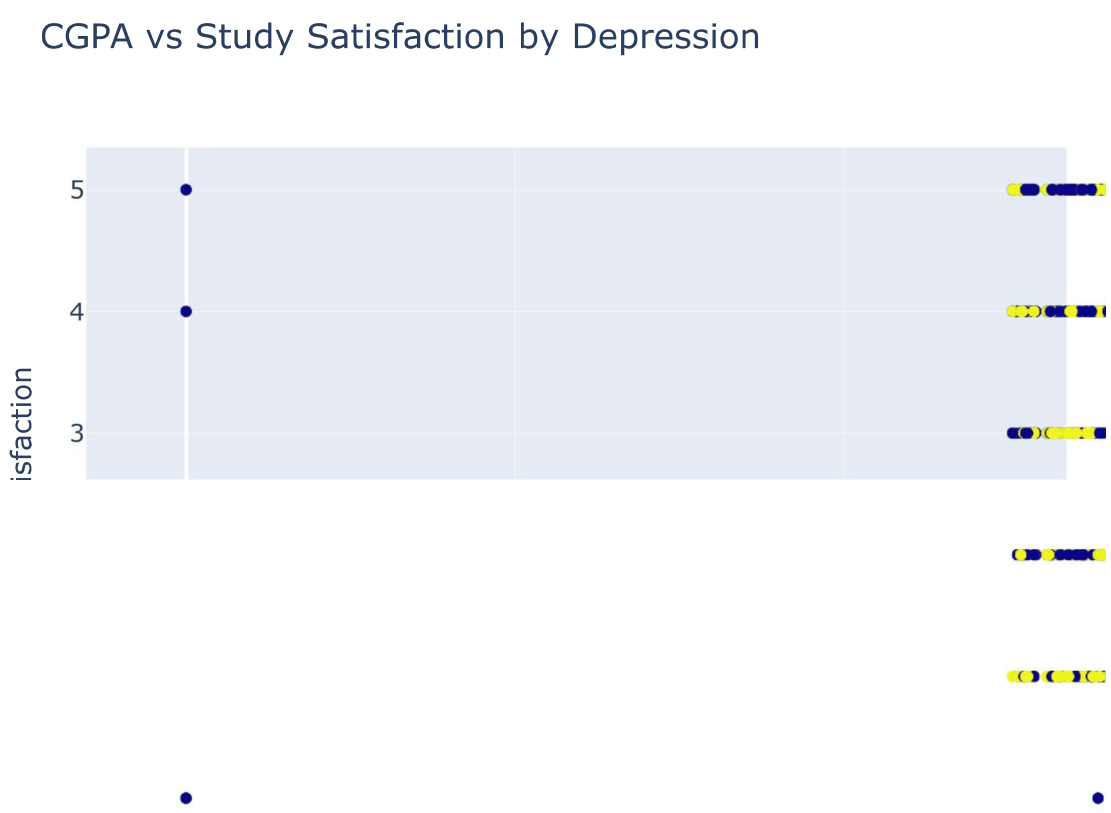
In [23]:
```
plt.figure(figsize=(10,8))
num_cols = ['Age', 'Academic Pressure', 'Work Pressure', 'CGPA',
```

```
            'Study Satisfaction', 'Job Satisfaction', 'Sleep Duration', 'Work/St
corr_matrix = data[num_cols].corr()
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Heatmap")
plt.show()
```



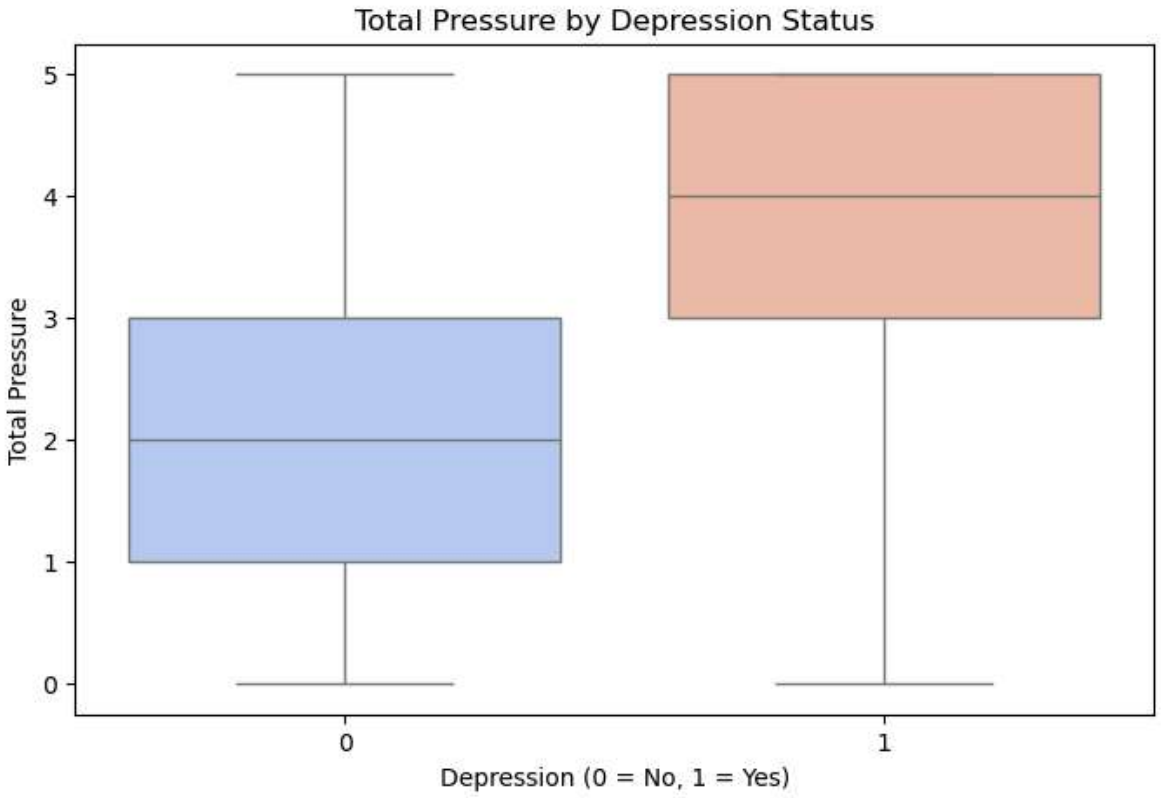
CGPA vs Study Satisfaction by Depression

In [24]:
```
import plotly.express as px
fig = px.scatter(data, x="CGPA", y="Study Satisfaction", color="Depression",
                 hover_data=['Age', 'Gender', 'Academic Pressure'],
                 title="CGPA vs Study Satisfaction by Depression")
fig.show()
```



Total Pressure by Depression Status

In [25]:
```
data['Total Pressure'] = data['Academic Pressure'] + data['Work Pressure']

plt.figure(figsize=(8,5))
sns.boxplot(x='Depression', y='Total Pressure', data=data, palette="coolwarm")
plt.title("Total Pressure by Depression Status")
plt.xlabel("Depression (0 = No, 1 = Yes)")
plt.ylabel("Total Pressure")
plt.show()
```



Encoding

In [26]:
```
le = LabelEncoder()
encode = data.select_dtypes(include='object').columns
for i in encode:
    data[i] = le.fit_transform(data[i])
```

In [27]:
```
num = data.select_dtypes(include=['float64','int32']).columns
for i in num:
    data[i] = data[i].astype('int64')
```

Target column value counts

In [28]:
```
data['Depression'].value_counts()
```

Out[28]:
```
Depression
1    16336
0    11565
Name: count, dtype: int64
```

In [29]:
```
max_count = data['Depression'].value_counts().max()
max_count
```

Out[29]:
```
16336
```

In [30]:
```
upsampled_list = []

for cls in data['Depression'].unique():
    cls_data = data[data['Depression'] == cls]
    cls_upsampled = resample(
```

```
        cls_data,
        replace=True,
        n_samples=max_count,
        random_state=42
    )
    upsampled_list.append(cls_upsampled)

data_balanced = pd.concat(upsampled_list)

data_balanced = data_balanced.sample(frac=1, random_state=42).reset_index(drop=T

print(data_balanced['Depression'].value_counts())
```

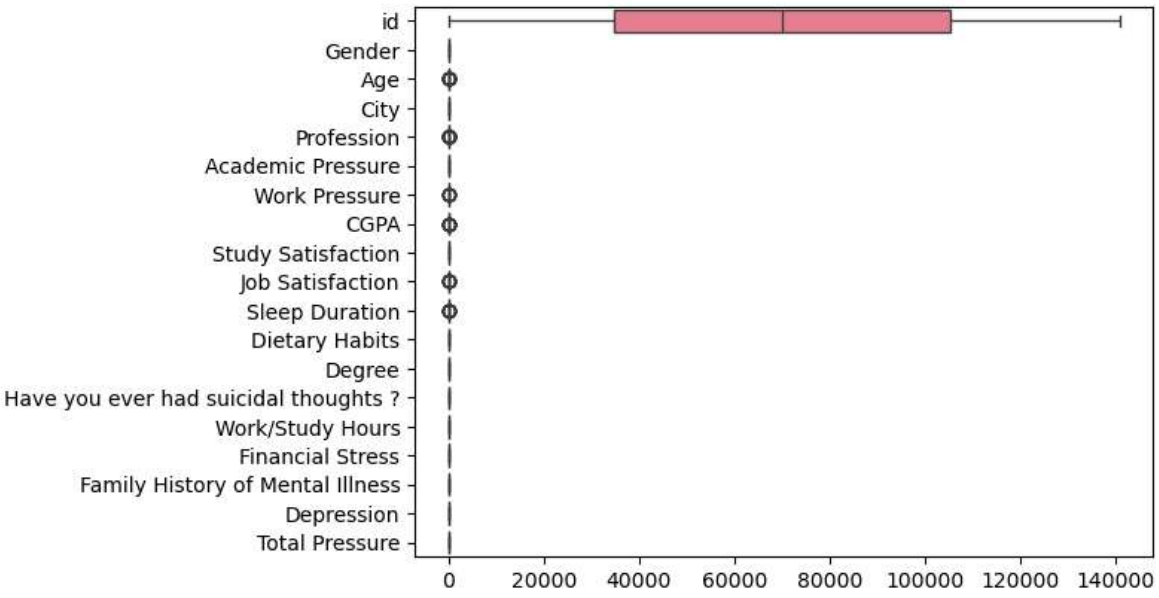
```
Depression
0    16336
1    16336
Name: count, dtype: int64
```

Swape Variable

In [31]:
```
data = data_balanced
```

Checking for Outliers

In [32]:
```
sns.boxplot(data=data,orient='h')
plt.show();
```



Splitting for Target and Feature into Train Test Split

In [33]:
```
X = data.drop(['Depression', 'id'], axis=1)
y = data['Depression']

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2)
```

Preprocessing pipeline

In [34]:
```
from sklearn.pipeline import Pipeline
preprocessor = Pipeline ([
    ('imputer',SimpleImputer(strategy='median')),
    ('scaler',StandardScaler())
])
```

Define models & parameter grids for classification

In [35]:
```
models = {
    # ------------------------------
    "Logistic Regression": {
        "model": LogisticRegression(solver='lbfgs', max_iter=500, random_state=4
        "param_grid": {
            "C": [0.1, 1, 10],
            "penalty": ['l2']
        }
    },
    # ------------------------------
    "Decision Tree": {
        "model": DecisionTreeClassifier(random_state=42),
        "param_grid": {
            "max_depth": [3, 5, 10, None],
            "min_samples_split": [2, 5, 10],
            "min_samples_leaf": [1, 2, 4]
        }
    },
    # ------------------------------
    "Random Forest": {
        "model": RandomForestClassifier(random_state=42),
        "param_grid": {
            "n_estimators": [100, 200],
            "max_depth": [None, 10, 20],
            "min_samples_split": [2, 5],
            "min_samples_leaf": [1, 2]
        }
    },
    # ------------------------------
    "Gradient Boosting": {
        "model": GradientBoostingClassifier(random_state=42),
        "param_grid": {
            "n_estimators": [100, 200],
            "learning_rate": [0.01, 0.1, 0.2],
            "max_depth": [3, 5]
        }
    },
    # ------------------------------
    "AdaBoost": {
        "model": AdaBoostClassifier(random_state=42),
        "param_grid": {
            "n_estimators": [50, 100, 200],
            "learning_rate": [0.01, 0.1, 1.0]
```

```
        }
    },
    # ------------------------------
    "Extra Trees": {
        "model": ExtraTreesClassifier(random_state=42),
        "param_grid": {
            "n_estimators": [100, 200],
            "max_depth": [None, 10, 20],
            "min_samples_split": [2, 5],
            "min_samples_leaf": [1, 2]
        }
    },
    # ------------------------------
    "MLP Classifier": {
        "model": MLPClassifier(random_state=42),
        "param_grid": {
            "hidden_layer_sizes": [(50,), (100,)],
            "activation": ["relu", "tanh"],
            "solver": ["adam"],
            "max_iter": [200]
        }
    },
    # ------------------------------
    "XGBoost": {
        "model": XGBClassifier(use_label_encoder=False, eval_metric="logloss", r
        "param_grid": {
            "n_estimators": [100, 200],
            "max_depth": [3, 5, 7],
            "learning_rate": [0.01, 0.1, 0.2],
            "subsample": [0.8, 1.0],
            "colsample_bytree": [0.8, 1.0]
        }
    }
}

print(f" Total Models Loaded: {len(models)}")
```

```
Total Models Loaded: 8
```

Run GridSearchCV for each regressor model

In [36]:

```
results = []
best_score = -float('inf')
best_model = None
best_name = None

for name, mp in models.items():
    print(f"\n Running GridSearchCV for {name}...")
    grid = GridSearchCV(mp['model'], param_grid=mp['param_grid'], cv=5, scoring=
    grid.fit(X_train, y_train)

    y_pred = grid.predict(X_test)

    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred, average='weighted')
    rec = recall_score(y_test, y_pred, average='weighted')
    f1 = f1_score(y_test, y_pred, average='weighted')
```

```
    results.append({
        'Model': name,
        'Best Params': grid.best_params_,
        'Accuracy': acc,
        'Precision': prec,
        'Recall': rec,
        'F1 Score': f1
    })

    print(f"{name} Best CV Score (Accuracy): {grid.best_score_:.4f}")
    print(f"{name} Best Params: {grid.best_params_}")
    print(f"{name} Classification Report:\n{classification_report(y_test, y_pred

    if grid.best_score_ > best_score:
        best_score = grid.best_score_
        best_model = grid.best_estimator_
        best_name = name

# ----------------------------------------
# Summary Results
# ----------------------------------------
results_df = pd.DataFrame(results).sort_values(by="F1 Score", ascending=False)
print("\n Summary Results:")
print(results_df)

print(f"\n Best Model: {best_name}")
print(f" Best CV Score (Accuracy): {best_score:.4f}")
```

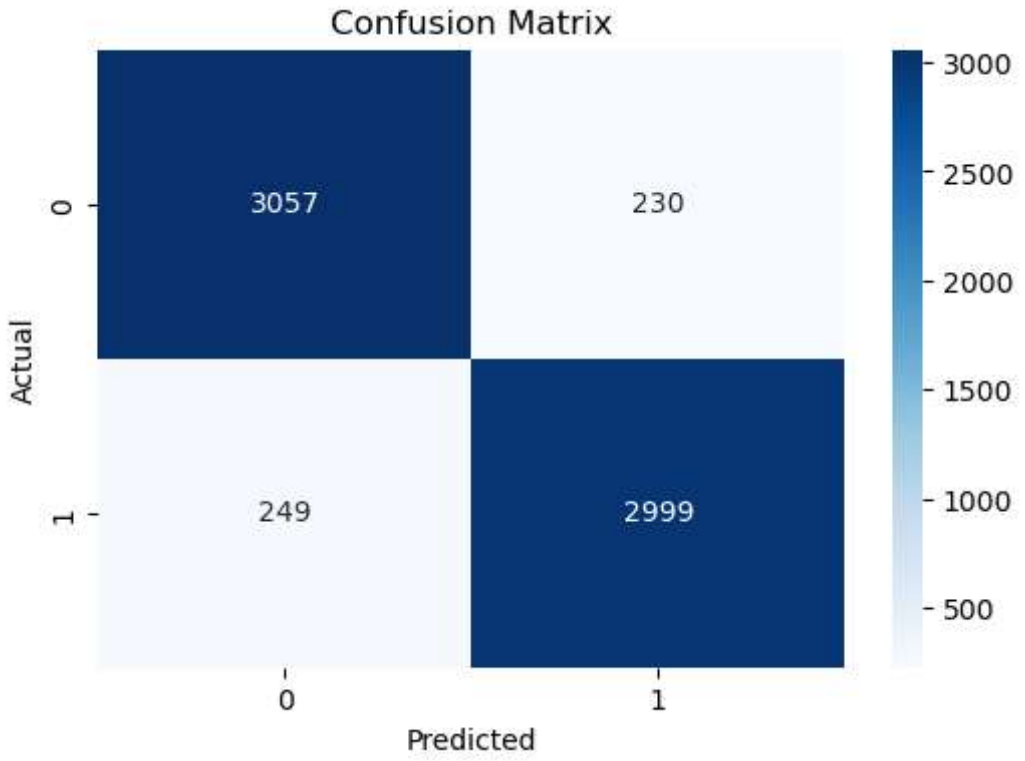
```
 Running GridSearchCV for Logistic Regression...
Logistic Regression Best CV Score (Accuracy): 0.8462
Logistic Regression Best Params: {'C': 0.1, 'penalty': 'l2'}
Logistic Regression Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.84      0.84      3287
           1       0.83      0.84      0.84      3248

    accuracy                           0.84      6535
   macro avg       0.84      0.84      0.84      6535
weighted avg       0.84      0.84      0.84      6535


 Running GridSearchCV for Decision Tree...
Decision Tree Best CV Score (Accuracy): 0.8869
Decision Tree Best Params: {'max_depth': None, 'min_samples_leaf': 1, 'min_sample
s_split': 2}
Decision Tree Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.92      0.91      3287
           1       0.92      0.90      0.91      3248

    accuracy                           0.91      6535
   macro avg       0.91      0.91      0.91      6535
weighted avg       0.91      0.91      0.91      6535


 Running GridSearchCV for Random Forest...
Random Forest Best CV Score (Accuracy): 0.9249
Random Forest Best Params: {'max_depth': 20, 'min_samples_leaf': 1, 'min_samples_
split': 2, 'n_estimators': 200}
Random Forest Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.95      0.94      3287
           1       0.94      0.93      0.94      3248

    accuracy                           0.94      6535
   macro avg       0.94      0.94      0.94      6535
weighted avg       0.94      0.94      0.94      6535


 Running GridSearchCV for Gradient Boosting...
Gradient Boosting Best CV Score (Accuracy): 0.8789
Gradient Boosting Best Params: {'learning_rate': 0.2, 'max_depth': 5, 'n_estimato
rs': 200}
Gradient Boosting Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.87      0.88      3287
           1       0.87      0.88      0.88      3248

    accuracy                           0.88      6535
   macro avg       0.88      0.88      0.88      6535
weighted avg       0.88      0.88      0.88      6535


 Running GridSearchCV for AdaBoost...
```

```
AdaBoost Best CV Score (Accuracy): 0.8483
AdaBoost Best Params: {'learning_rate': 1.0, 'n_estimators': 200}
AdaBoost Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.83      0.84      3287
           1       0.83      0.85      0.84      3248

    accuracy                           0.84      6535
   macro avg       0.84      0.84      0.84      6535
weighted avg       0.84      0.84      0.84      6535


 Running GridSearchCV for Extra Trees...
Extra Trees Best CV Score (Accuracy): 0.9238
Extra Trees Best Params: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_
split': 5, 'n_estimators': 200}
Extra Trees Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.94      0.94      3287
           1       0.94      0.93      0.94      3248

    accuracy                           0.94      6535
   macro avg       0.94      0.94      0.94      6535
weighted avg       0.94      0.94      0.94      6535


 Running GridSearchCV for MLP Classifier...
MLP Classifier Best CV Score (Accuracy): 0.8489
MLP Classifier Best Params: {'activation': 'tanh', 'hidden_layer_sizes': (100,),
'max_iter': 200, 'solver': 'adam'}
MLP Classifier Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.81      0.84      3287
           1       0.82      0.88      0.85      3248

    accuracy                           0.85      6535
   macro avg       0.85      0.85      0.85      6535
weighted avg       0.85      0.85      0.85      6535


 Running GridSearchCV for XGBoost...
XGBoost Best CV Score (Accuracy): 0.9153
XGBoost Best Params: {'colsample_bytree': 1.0, 'learning_rate': 0.2, 'max_depth':
7, 'n_estimators': 200, 'subsample': 0.8}
XGBoost Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.93      0.93      3287
           1       0.93      0.92      0.93      3248

    accuracy                           0.93      6535
   macro avg       0.93      0.93      0.93      6535
weighted avg       0.93      0.93      0.93      6535


 Summary Results:
                Model                                        Best Params  \
```

```
5          Extra Trees  {'max_depth': None, 'min_samples_leaf': 1, 'mi...
2        Random Forest  {'max_depth': 20, 'min_samples_leaf': 1, 'min_...
7              XGBoost  {'colsample_bytree': 1.0, 'learning_rate': 0.2...
1        Decision Tree  {'max_depth': None, 'min_samples_leaf': 1, 'mi...
3    Gradient Boosting  {'learning_rate': 0.2, 'max_depth': 5, 'n_esti...
6       MLP Classifier  {'activation': 'tanh', 'hidden_layer_sizes': (...
4             AdaBoost        {'learning_rate': 1.0, 'n_estimators': 200}
0  Logistic Regression                        {'C': 0.1, 'penalty': 'l2'}

   Accuracy  Precision    Recall  F1 Score
5  0.938944   0.938971  0.938944  0.938942
2  0.938638   0.938709  0.938638  0.938633
7  0.926702   0.926714  0.926702  0.926700
1  0.910482   0.910615  0.910482  0.910469
3  0.876970   0.877004  0.876970  0.876971
6  0.847590   0.849477  0.847590  0.847424
4  0.840398   0.840496  0.840398  0.840396
0  0.837950   0.837973  0.837950  0.837951

Best Model: Random Forest
Best CV Score (Accuracy): 0.9249
```

Confusion Matrix

In [38]:
```
cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=np.unique(y_test),
            yticklabels=np.unique(y_test))
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```



Save the model

In [39]:
```
feature_columns = X.columns.to_list()
joblib.dump(feature_columns,'23.Student Depression.joblib')
```

Out[39]:
```
['23.Student Depression.joblib']
```

In [40]:
```
joblib.dump(best_model,'bestmodel 23.Student Depression.joblib')
```

Out[40]:
```
['bestmodel 23.Student Depression.joblib']
```

If you have any suggestions, please DM me.

Even a small message from you can make a big impact on my career

I Am Arun