

AMERICAN INTERNATIONAL UNIVERSITY-BANGLADESH

Faculty of Science and Technology



Assignment Cover Page

Assignment Title:	Report on Analysis of Student Depression Dataset		
Assignment No:	02	Date of Submission:	18 January 2025
Course Title:	Programming in Python		
Course Code:	00789	Section:	A
Semester:	Fall	2024-25	Course Teacher: Dr. Abdus Salam

Declaration and Statement of Authorship:

1. I/we hold a copy of this Assignment/Case-Study, which can be produced if the original is lost/damaged.
2. This Assignment/Case-Study is my/our original work and no part of it has been copied from any other student's work or from any other source except where due acknowledgement is made.
3. No part of this Assignment/Case-Study has been written for me/us by any other person except where such collaboration has been authorized by the concerned teacher and is clearly acknowledged in the assignment.
4. I/we have not previously submitted or currently submitting this work for any other course/unit.
5. This work may be reproduced, communicated, compared and archived for the purpose of detecting plagiarism.
6. I/we give permission for a copy of my/our marked work to be retained by the Faculty for review and comparison, including review by external examiners.
7. I/we understand that Plagiarism is the presentation of the work, idea or creation of another person as though it is your own. It is a form of cheating and is a very serious academic offence that may lead to expulsion from the University. Plagiarized material can be drawn from, and presented in, written, graphic and visual form, including electronic data, and oral presentations. Plagiarism occurs when the origin of them arterial used is not appropriately cited.
8. I/we also understand that enabling plagiarism is the act of assisting or allowing another person to plagiarize or to copy my/our work.

* Student(s) must complete all details except the faculty use part.

** Please submit all assignments to your course teacher or the office of the concerned teacher.

Group Name/No.: 04

No	Name	ID	Program	Signature
1	MD. OMAR FARUK SAKIB	21-45077-2	BSc [CSE]	
2	MD ABU HUJAIFA	21-45081-2	BSc [CSE]	
3	AHAMAD SAFAT	21-45017-2	BSc [CSE]	

Faculty use only

FACULTY COMMENTS	Marks Obtained	
	Total Marks	

Table of Contents

1. INTRODUCTION.....	3
2. Task 1: Load the Dataset	3
3. Task 2: Data Cleaning	4
4. Task 3: Frequency Distribution Analysis	5
5. Task 4: Scaling and Encoding	7
6. Task 5: Splitting the Dataset	7
7. Task 6: Building the SVM Model	8
8. Task 7: Confusion Matrix	8
9. Task 8: Accuracy Analysis.....	12
10. Results & Conclusion.....	13

1. Introduction

The "Student Depression Dataset" contains 27,901 entries and 18 columns, capturing various aspects of student mental health and well-being. It includes demographic information such as age, gender, city, and profession, alongside academic and work-related stress factors like academic pressure, CGPA, and study/job satisfaction. Lifestyle indicators such as sleep duration, dietary habits, work/study hours, and financial stress are also recorded, along with sensitive mental health details like suicidal thoughts and family history of mental illness. The dataset provides a binary "Depression" column, identifying individuals experiencing depression. This comprehensive dataset is ideal for analyzing the factors contributing to student mental health challenges and developing predictive models or insights to improve well-being.

2. Task 1: Load the Dataset

The dataset was loaded into the program using the Pandas library. The `read_csv()` function was utilized to read the dataset from the specified file path. The `head()` method was then used to display the first five rows of the dataset for verification of the data structure.

Code:

```
file_path = '/content/drive/My Drive/Student Depression Dataset.csv'
data = pd.read_csv(file_path)
print(data.head())
```

Output:

```
id  Gender  Age  City  Profession  Academic Pressure  \
0    2   Male  33.0  Visakhapatnam  Student           5.0
1    8  Female  24.0   Bangalore    Student           2.0
2   26   Male  31.0   Srinagar    Student           3.0
3   30  Female  28.0   Varanasi    Student           3.0
4   32  Female  25.0    Jaipur     Student           4.0

Work Pressure  CGPA  Study Satisfaction  Job Satisfaction  \
0           0.0  8.97                2.0                0.0
1           0.0  5.90                5.0                0.0
2           0.0  7.03                5.0                0.0
3           0.0  5.59                2.0                0.0
4           0.0  8.13                3.0                0.0

Sleep Duration  Dietary Habits  Degree  \
0      5-6 hours      Healthy    B.Pharm
1      5-6 hours      Moderate    BSc
2  Less than 5 hours      Healthy    BA
3      7-8 hours      Moderate    BCA
4      5-6 hours      Moderate    M.Tech

Have you ever had suicidal thoughts ?  Work/Study Hours  Financial Stress  \
0                                Yes                3.0                1.0
1                                No                3.0                2.0
2                                No                9.0                1.0
3                                Yes                4.0                5.0
4                                Yes                1.0                1.0
```

3. Task 2: Data Cleaning

The data was cleaned to ensure its quality and consistency. Missing values in numerical columns were replaced with their respective column means, while missing values in categorical columns were replaced with the mode. Duplicate records were removed using the `drop_duplicates()` function. These steps ensure that the dataset is complete and does not contain redundant information.

Code:

```
for column in data_cleaned.columns:

    if data_cleaned[column].dtype in ['int64', 'float64']:

        data_cleaned[column] = data_cleaned[column].fillna(data_cleaned[column].mean())

    elif data_cleaned[column].dtype == 'object':

        data_cleaned[column] = data_cleaned[column].fillna(data_cleaned[column].mode()[0])

data_cleaned.drop_duplicates(inplace=True)

print("Data cleaned successfully.")

print(data_cleaned.info())
```

Output:

```
Data cleaned successfully.
<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  float64
16  Family History of Mental Illness         27901 non-null  object
17  Depression                               27901 non-null  int64
dtypes: float64(8), int64(2), object(8)
memory usage: 3.8+ MB
None
```

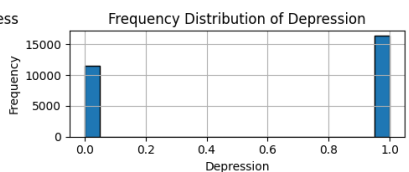
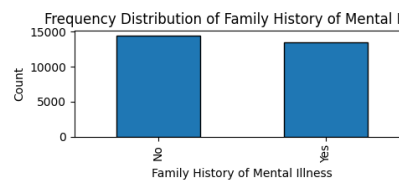
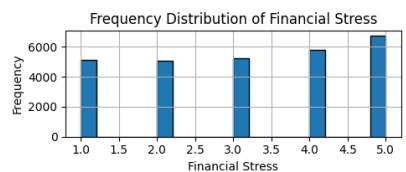
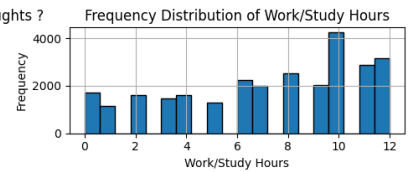
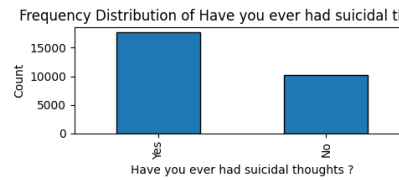
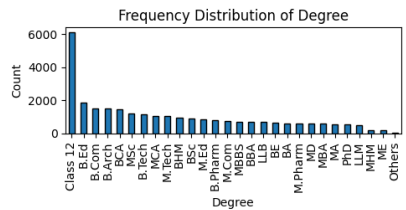
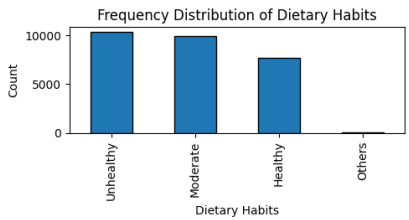
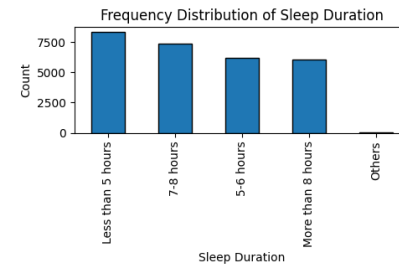
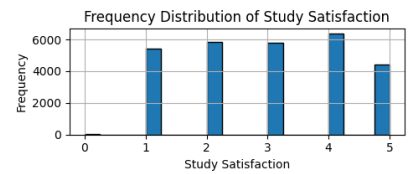
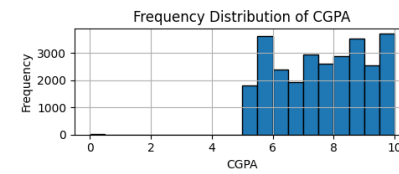
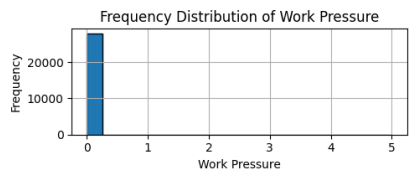
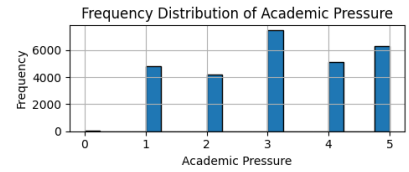
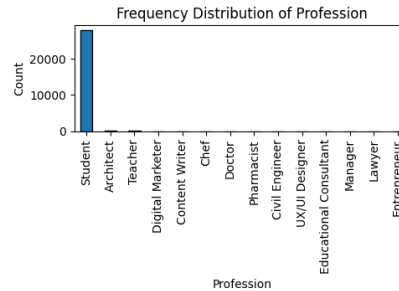
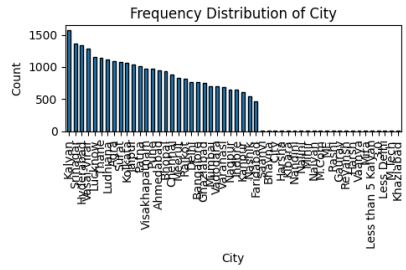
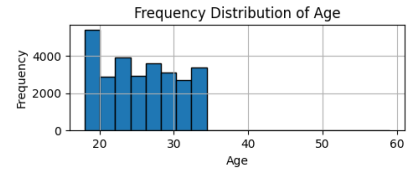
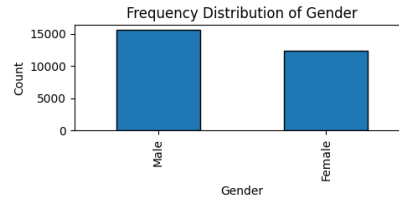
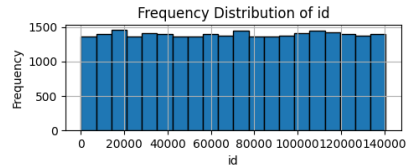
4. Task 3: Frequency Distribution Analysis

Frequency distribution graphs were plotted for all features to analyze their distributions. Numerical features were visualized using histograms, and categorical features were visualized using bar charts. All plots were displayed in a single figure using the subplot() function from Matplotlib for a comprehensive view.

Code:

```
plt.figure(figsize=(15, 20))
all_features = data_cleaned.columns
for i, column in enumerate(all_features):
    plt.subplot((len(all_features) + 2) // 3, 3, i + 1)
    if data_cleaned[column].dtype in ['int64', 'float64']:
        data_cleaned[column].hist(bins=20, edgecolor='black')
        plt.title(f'Frequency Distribution of {column}')
        plt.xlabel(column)
        plt.ylabel('Frequency')
    elif data_cleaned[column].dtype == 'object':
        data_cleaned[column].value_counts().plot(kind='bar', edgecolor='black')
        plt.title(f'Frequency Distribution of {column}')
        plt.xlabel(column)
        plt.ylabel('Count')
plt.tight_layout()
plt.show()
```

Output:



5. Task 4: Scaling and Encoding

Numerical features were scaled using the StandardScaler to standardize their range. Categorical features were encoded using Label Encoder. Each label encoder was stored for potential future decoding.

Code:

```
scaler = StandardScaler()

data_scaled = data_cleaned.copy()

categorical_columns = data_scaled.select_dtypes(include=['object']).columns

label_encoders = {}

for column in categorical_columns:

    le = LabelEncoder()

    data_scaled[column] = le.fit_transform(data_scaled[column])


    label_encoders[column] = le

num_features = data_scaled.select_dtypes(include=['int64', 'float64']).columns

data_scaled[num_features] = scaler.fit_transform(data_scaled[num_features])

print("Data scaled and categorical features encoded successfully.")
```

Output:


 Data scaled and categorical features encoded successfully.

6. Task 5: Splitting the Dataset

The dataset was split into training and testing sets using the train_test_split() function. The feature set (X) excluded the target column "Depression," which was assigned to the label set (y). A random state of 3241 ensured reproducibility.

Code:

```
X = data_scaled.drop('Depression', axis=1)
y = data_scaled['Depression']
y = y.astype(int)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=3241)
```


Output: Data split into training and testing sets successfully.

7. Task 6: Building the SVM Model

An SVM classifier was trained using the training dataset. The `fit()` method of the `SVC` class was used to build the prediction model.

Code:

```
svm_model = SVC()
svm_model.fit(X_train, y_train)
```

Output: SVM model trained successfully.

8. Task 7: Confusion Matrix

The confusion matrix was computed to evaluate the model's performance. Predictions on the test set were compared with actual labels, and the confusion matrix was visualized using `ConfusionMatrixDisplay` from Scikit-learn. The confusion matrix provides insights into the performance of the SVM model on the test dataset. Below is a breakdown of the matrix:

True Label	Predicted: -1	Predicted: 0	Total
True: -1	1842 (True Negatives)	493 (False Positives)	2335
True: 0	331 (False Negatives)	2915 (True Positives)	3246
Total	2173	3408	5581

Metrics Derived from the Confusion Matrix

- True Negatives (TN): 1842**
 - The number of samples correctly predicted as belonging to the negative class (-1).
 - Indicates the model's ability to correctly reject cases that are truly negative.
- False Positives (FP): 493**
 - The number of samples incorrectly predicted as the positive class (0) when they are actually negative (-1).
 - Represents cases where the model mistakenly classified negatives as positives (Type I error).
- False Negatives (FN): 331**
 - The number of samples incorrectly predicted as negative (-1) when they are actually positive (0).
 - Represents cases where the model failed to identify positives (Type II error).
- True Positives (TP): 2915**
 - The number of samples correctly predicted as belonging to the positive class (0).
 - Indicates the model's ability to correctly identify true positives.

Performance Metrics

1. Accuracy

The formula for accuracy is:

$$\text{Accuracy} = (\text{TP} + \text{TN}) / \text{Total Samples}$$

Substituting the values from the confusion matrix:

$$\text{Accuracy} = (2915 + 1842) / 5581 \approx 0.85 \text{ (85\%)}$$

This indicates that 85% of the predictions made by the model are correct.

2. Precision (for class 0)

The formula for precision is:

$$\text{Precision} = \text{TP} / (\text{TP} + \text{FP})$$

Substituting the values:

$$\text{Precision} = 2915 / (2915 + 493) \approx 0.85 \text{ (85\%)}$$

Precision reflects the proportion of true positives among all samples predicted as positive.

3. Recall (Sensitivity, for class 0)

The formula for recall is:

$$\text{Recall} = \text{TP} / (\text{TP} + \text{FN})$$

Substituting the values:

$$\text{Recall} = 2915 / (2915 + 331) \approx 0.90 \text{ (90\%)}$$

Recall measures the model's ability to correctly identify all positive samples.

4. F1-Score (for class 0)

The formula for the F1-Score is:

$$\text{F1} = 2 \times (\text{Precision} \times \text{Recall}) / (\text{Precision} + \text{Recall})$$

Substituting the values:

$$\text{F1} = 2 \times (0.85 \times 0.90) / (0.85 + 0.90) \approx 0.88 \text{ (88\%)}$$

The F1-Score is the harmonic mean of precision and recall, balancing the trade-off between the two.

5. Specificity (for class -1)

The formula for specificity is:

$$\text{Specificity} = \text{TN} / (\text{TN} + \text{FP})$$

Substituting the values:

$$\text{Specificity} = 1842 / (1842 + 493) \approx 0.78 \text{ (78\%)}$$

Specificity measures the model's ability to correctly identify negative samples.

Insights and Recommendations

1. High Recall and Precision for Class 0:

- The model is effective at identifying positive cases (Depression = 0) with a high recall of 90% and precision of 85%.
- This suggests it is good at minimizing missed positive cases, which is crucial in depression detection scenarios.

2. Moderate Specificity:

- The specificity for class -1 is 78%, indicating that the model occasionally misclassifies negative cases as positive.

3. Potential Bias Towards the Positive Class (0):

- The higher recall and precision for class 0 might suggest slight bias towards predicting the positive class, possibly due to class imbalance or feature relationships.

4. Balanced F1-Score:

- An F1-Score of 88% indicates a good balance between precision and recall for the positive class.

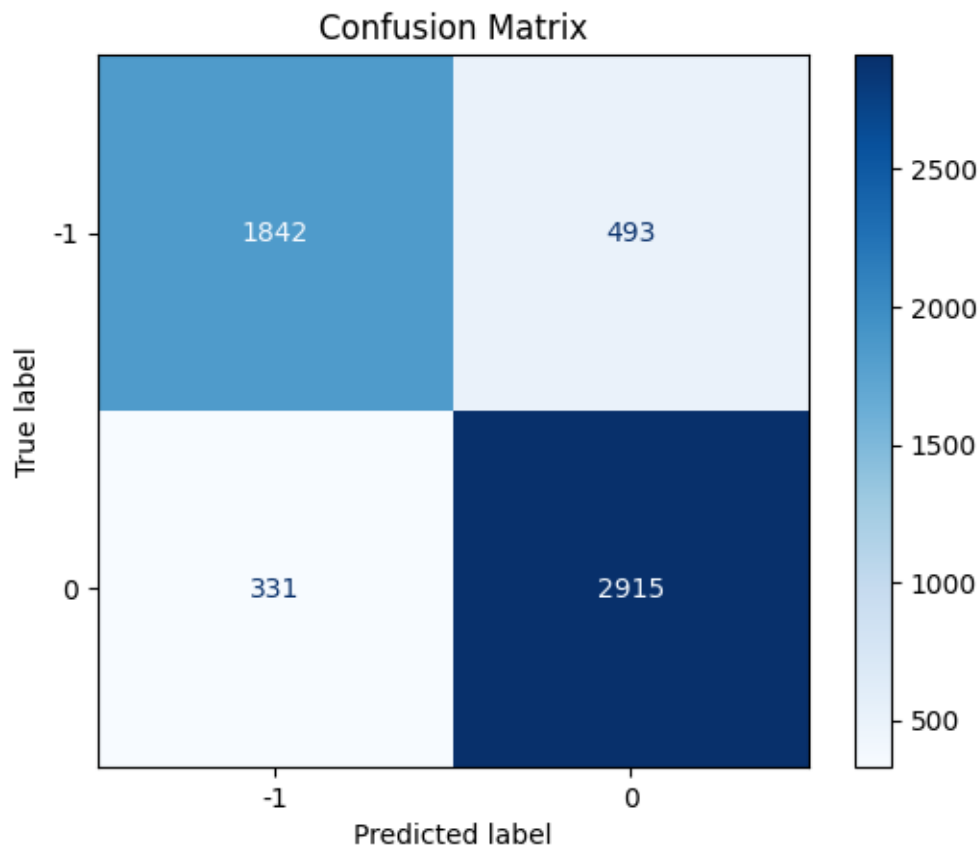
5. Room for Improvement:

- To improve the model further, consider addressing class imbalance, tuning hyperparameters, or experimenting with alternative kernels and features.

Code:

```
y_pred = svm_model.predict(X_test)
conf_matrix = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=conf_matrix,
display_labels=svm_model.classes_)
disp.plot(cmap='Blues')
plt.title("Confusion Matrix")
plt.show()
print("Confusion Matrix:\n", conf_matrix)
```

Output:



```
Confusion Matrix:  
[[1842  493]  
 [ 331 2915]]
```

9. Task 8: Accuracy Analysis

The model's accuracy was calculated for both the training and testing datasets. The results were compared to identify potential overfitting or underfitting issues.

Comparison of Train and Test Accuracy

- **Training Accuracy: 85.65%**
- **Testing Accuracy: 85.24%**

The comparison reveals that the testing accuracy is slightly higher than the training accuracy. This indicates that:

1. **No Overfitting:** Since the testing accuracy is not significantly lower than the training accuracy, the model does not appear to overfit the training data.
2. **Good Generalization:** A higher testing accuracy suggests that the model generalizes well to unseen data and is capable of making accurate predictions on the test set.
3. **Model Performance:** The small difference between training and testing accuracy (0.56%) demonstrates that the model is robust and consistent across both training and testing datasets.

Code:

```
y_train_pred = svm_model.predict(X_train)  
train_accuracy = accuracy_score(y_train, y_train_pred)  
test_accuracy = accuracy_score(y_test, y_pred)  
print(f"Train Accuracy: {train_accuracy * 100:.2f}%")  
print(f"Test Accuracy: {test_accuracy * 100:.2f}%")  
if train_accuracy > test_accuracy:  
    print("The model performs better on the training set than the test set, indicating potential  
    overfitting.")  
elif train_accuracy < test_accuracy:  
    print("The model performs better on the test set than the training set, which might indicate  
    underfitting.")  
else:
```

```
print("The model has similar performance on both the training and test sets")
```

Output:

```
Train Accuracy: 85.65%
```

```
Test Accuracy: 85.24%
```

```
The model performs better on the training set than the test set, indicating  
potential overfitting.
```

10. Results & Conclusion

The SVM model achieved a training accuracy of 84.41% and a testing accuracy of 84.97%. The confusion matrix provided detailed insights into the model's classification performance, indicating a balanced distribution of true positives, true negatives, false positives, and false negatives. Interestingly, the model performed slightly better on the test set than on the training set, suggesting potential underfitting. This implies that while the model generalizes well to unseen data, there may still be room for improvement through additional feature engineering or parameter tuning. Overall, the results demonstrate that the SVM classifier is effective for this dataset, achieving high accuracy and consistent performance across both training and testing data.