Department of Computer Science
American International University-Bangladesh
Final Term Report

Course Name: Programming in Python

"Loan Prediction"

Supervised By:

Dr. Abdus Salam

Assistant Professor, Computer Science-AIUB

Submitted By:

Rasel Mahmud

ID: 20-43867-2

Section: A

Mohammad Shafin

ID: 20-43736-2

Section: A

Submission Date: December 25, 2023.

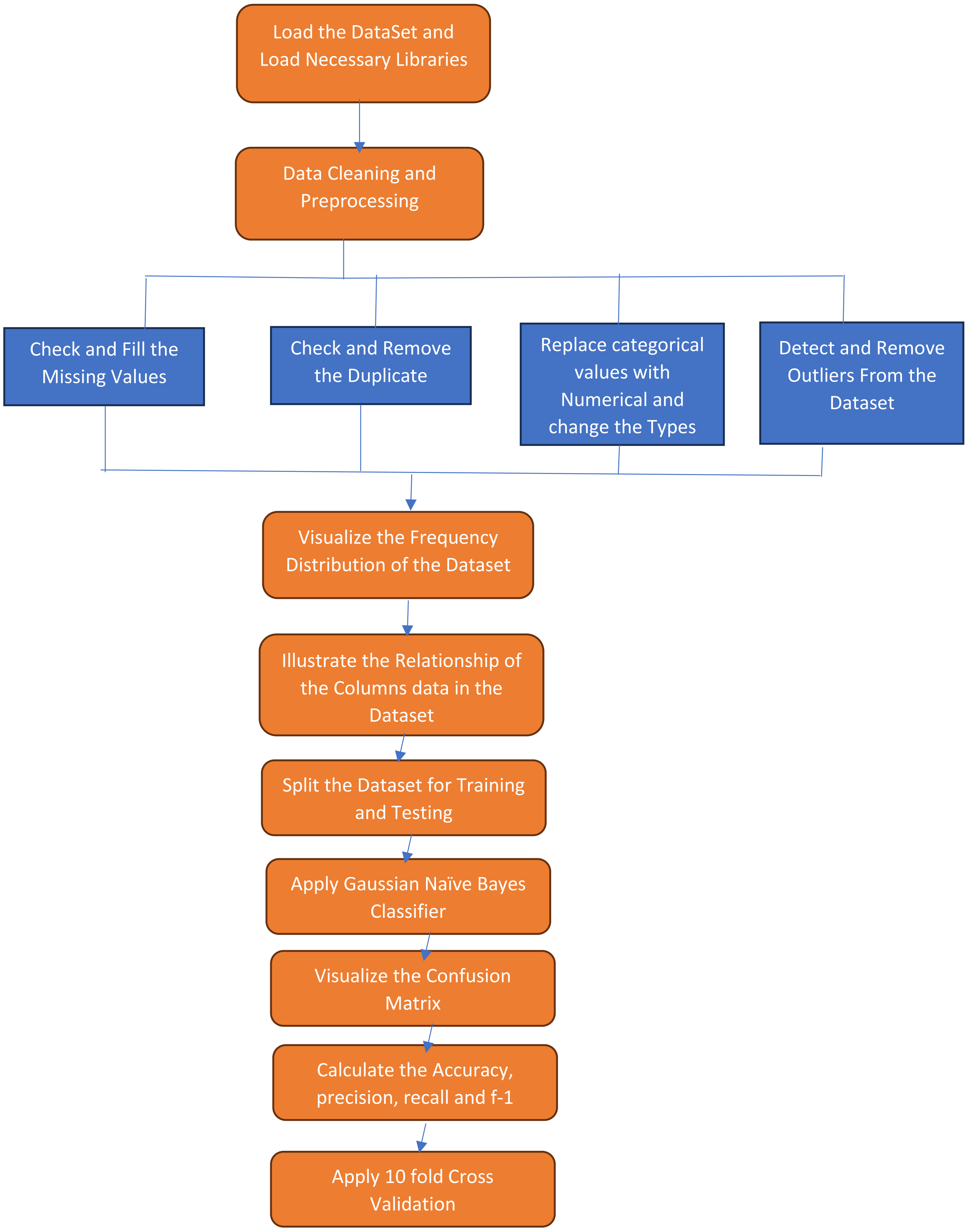
Description of The Dataset:

The chosen dataset represents a collection of information relating to loan applications. It encompasses diverse attributes associated with individuals seeking loans, encompassing factors such as gender, marital status, the number of dependents, educational background, self-employment status, income details of both the applicant and co-applicant, requested loan amounts, loan term duration, credit history, residential area, and the final status of the loan application - whether it was approved or not. This dataset exhibits a range of categorical and numerical attributes, offering insights into the factors that might influence the approval or rejection of loan applications. With attributes spanning demographic, financial, and credit-related aspects, this dataset presents an opportunity to explore patterns and potentially build predictive models to forecast loan approval outcomes based on applicant information.

Here's a short description of my dataset columns and values:

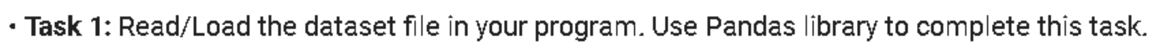
- **Gender:** Gender of the loan applicant (Male/Female)
- **Married:** Marital status of the applicant (Yes/No)
- **Dependents:** Number of dependents of the applicant (Numerical/3+ indicating more than 3)
- **Education:** Educational background of the applicant (Graduate/Not Graduate)
- **Self_Employed:** Self-employment status of the applicant (Yes/No)
- **Applicant_Income:** Income of the applicant (Numerical)
- **Coapplicant_Income:** Income of the co-applicant (Numerical)
- **Loan_Amount:** Amount of the loan requested (Numerical)
- **Term:** Term or duration of the loan (Numerical, in months)
- **Credit_History:** Credit history of the applicant (1: Good credit history, 0: Poor credit history)
- **Area:** Residential area of the applicant (Urban/Rural/Semiurban)
- **Status:** Loan status (Y: Approved, N: Not Approved)

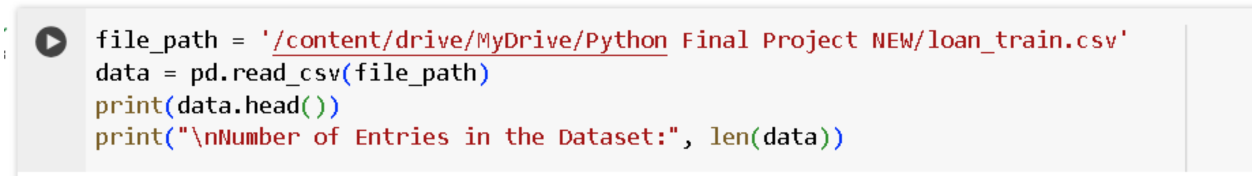
Flow-chart on the Full Project:


Load the DataSet and Load Necessary Libraries
Data Cleaning and Preprocessing
Check and Fill the Missing Values
Check and Remove the Duplicate
Replace categorical values with Numerical and change the Types
Detect and Remove Outliers From the Dataset
Visualize the Frequency Distribution of the Dataset
Illustrate the Relationship of the Columns data in the Dataset
Split the Dataset for Training and Testing
Apply Gaussian Naïve Bayes Classifier
Visualize the Confusion Matrix
Calculate the Accuracy, precision, recall and f-1
Apply 10 fold Cross Validation

Description of Implementation and Implemented Tasks:

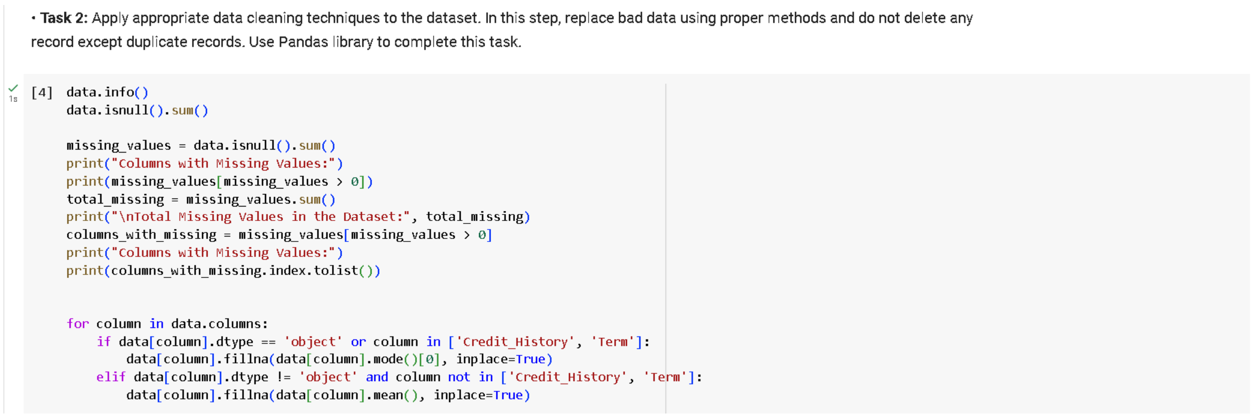
Task 1:


• **Task 1:** Read/Load the dataset file in your program. Use Pandas library to complete this task.

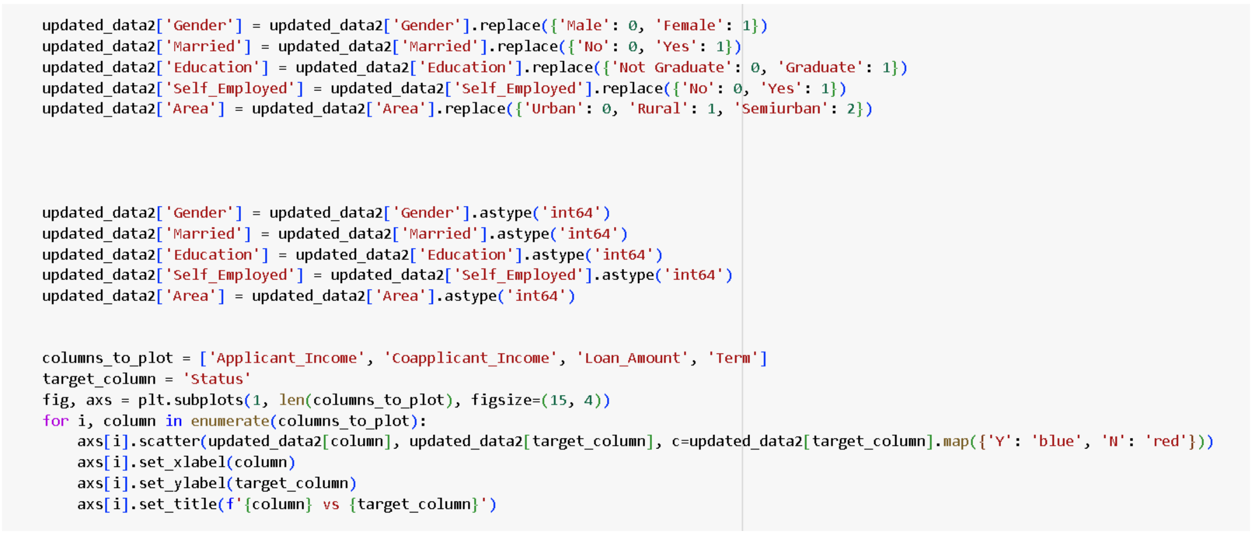

```python
file_path = '/content/drive/MyDrive/Python Final Project NEW/loan_train.csv'
data = pd.read_csv(file_path)
print(data.head())
print("\nNumber of Entries in the Dataset:", len(data))
```

In the project, the initial step involves loading the loan dataset from a CSV file using Pandas. This is accomplished by utilizing the pd.read_csv() function, which reads the data from the specified file path. Additionally, it displays some of the portion of the dataset and total number of rows in the dataset, providing insight into its size. These steps aid in the preliminary examination and understanding of the loan dataset's contents and dimensions forming the foundation for data analysis and modeling tasks .

Task 2:


• **Task 2:** Apply appropriate data cleaning techniques to the dataset. In this step, replace bad data using proper methods and do not delete any record except duplicate records. Use Pandas library to complete this task.

```python
[4] data.info()
    data.isnull().sum()

    missing_values = data.isnull().sum()
    print("Columns with Missing Values:")
    print(missing_values[missing_values > 0])
    total_missing = missing_values.sum()
    print("\nTotal Missing Values in the Dataset:", total_missing)
    columns_with_missing = missing_values[missing_values > 0]
    print("Columns with Missing Values:")
    print(columns_with_missing.index.tolist())


    for column in data.columns:
        if data[column].dtype == 'object' or column in ['Credit_History', 'Term']:
            data[column].fillna(data[column].mode()[0], inplace=True)
        elif data[column].dtype != 'object' and column not in ['Credit_History', 'Term']:
            data[column].fillna(data[column].mean(), inplace=True)
```


```python
updated_data2['Gender'] = updated_data2['Gender'].replace({'Male': 0, 'Female': 1})
updated_data2['Married'] = updated_data2['Married'].replace({'No': 0, 'Yes': 1})
updated_data2['Education'] = updated_data2['Education'].replace({'Not Graduate': 0, 'Graduate': 1})
updated_data2['Self_Employed'] = updated_data2['Self_Employed'].replace({'No': 0, 'Yes': 1})
updated_data2['Area'] = updated_data2['Area'].replace({'Urban': 0, 'Rural': 1, 'Semiurban': 2})




updated_data2['Gender'] = updated_data2['Gender'].astype('int64')
updated_data2['Married'] = updated_data2['Married'].astype('int64')
updated_data2['Education'] = updated_data2['Education'].astype('int64')
updated_data2['Self_Employed'] = updated_data2['Self_Employed'].astype('int64')
updated_data2['Area'] = updated_data2['Area'].astype('int64')


columns_to_plot = ['Applicant_Income', 'Coapplicant_Income', 'Loan_Amount', 'Term']
target_column = 'Status'
fig, axs = plt.subplots(1, len(columns_to_plot), figsize=(15, 4))
for i, column in enumerate(columns_to_plot):
    axs[i].scatter(updated_data2[column], updated_data2[target_column], c=updated_data2[target_column].map({'Y': 'blue', 'N': 'red'}))
    axs[i].set_xlabel(column)
    axs[i].set_ylabel(target_column)
    axs[i].set_title(f'{column} vs {target_column}')
```

In the second part multiple data preprocessing tasks are executed to prepare and clean the loan dataset for analysis and modeling. The steps involved are:

Handling Missing Values:

At first we found out which column contains the missing or null values. After find out the null values columns. We replace the null values in Categorical column by the mode and null values in Numerical columns by the mean. There is a exception for the Credit_History and Term column though it is numerical column. But we replace the missing values of this column by mode. As the data it contains is considered as categorical due its nature.

Duplication Check:

We check the duplicates row un the dataset using duplicated() function. But there are not any duplicated row found in the dataset.

Data Type Conversion and Value Manipulation:

We replace the Categorical data by the numerical data for implementing the Naïve Bayes Classifier and convert the data type as numerical from object. We also convert the data type of Dependencies column from object to numeric and also removed the + sign from the data.

Handling Outliers:

For Better Accuracy we detect the Outliers from the dataset by the Interquartile Range (IQR) method and replace the outlier values by the median value.

Saving the Cleaned Dataset:

After Performing all the Cleaning and Preprocessing Operation we saved our Dataset to Our Google Drive.

For Visualizing the Outliers, we have used the Scatterplot.

The necessity and Relationship of Task 2 in My Project:

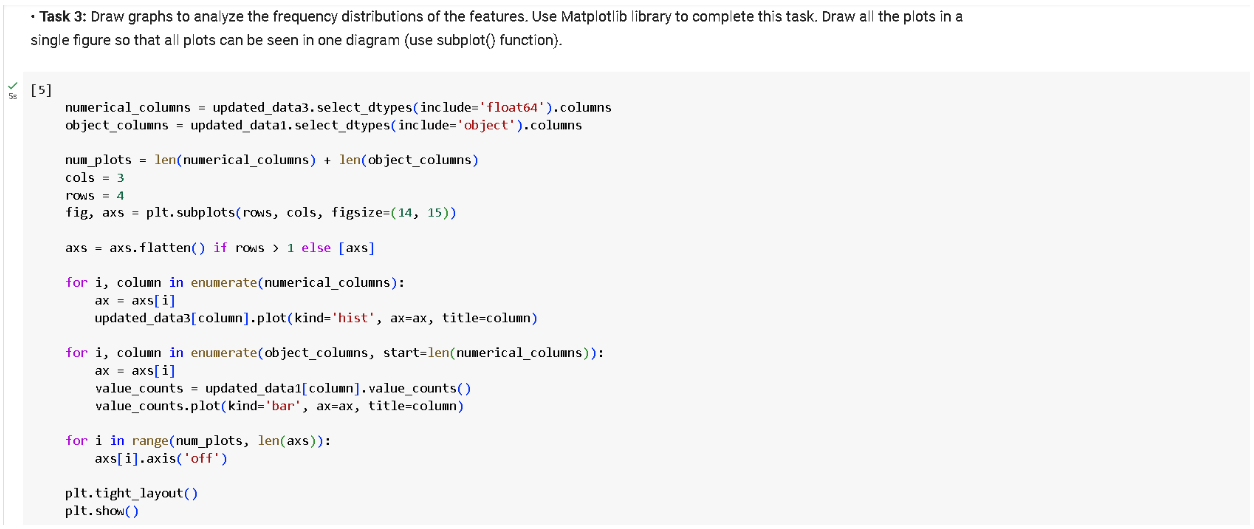
1. **Handling Missing Values:** Missing data can affect the accuracy of models. Replacing missing values in categorical columns with mode and numerical columns with mean ensures data completeness without skewing distributions. Treating 'Credit_History' as categorical and replacing missing values with mode maintains data integrity.
2. **Duplication Check:** Identifying and removing duplicate rows maintains dataset uniqueness, preventing biased analyses caused by repeated entries.
3. **Data Type Conversion and Value Manipulation:** Converting categorical data to numerical format facilitates model implementation. Adjusting 'Dependencies' column type and removing '+' signs ensure uniform numerical representation for accurate analysis.

4. **Handling Outliers:** Detecting and handling outliers using the IQR method ensures model robustness by mitigating the impact of extreme values on predictions, enhancing model accuracy and generalization.

5. **Saving the Cleaned Dataset:** Saving the cleaned dataset ensures the preservation of cleaned, processed data for future use, allowing reproducibility and ease of analysis without reprocessing.

These steps collectively prepare the dataset for accurate model training and analysis, enhancing the reliability and effectiveness of the subsequent machine learning processes in the project.

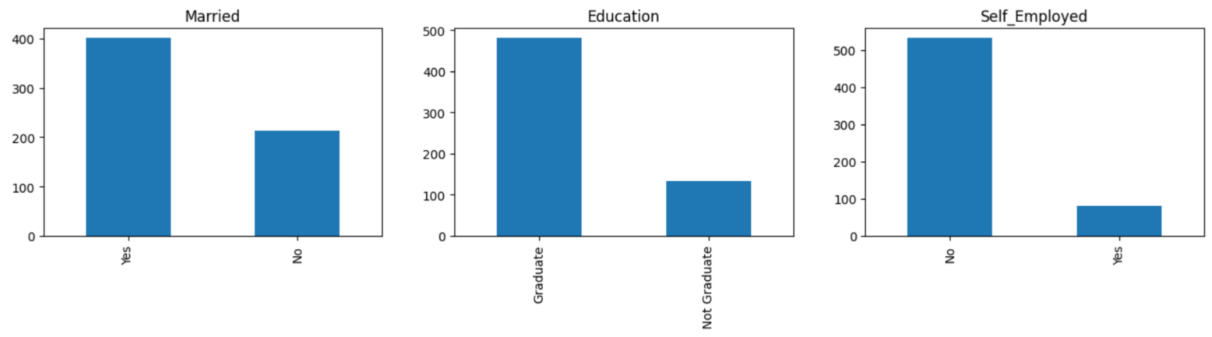
Task 3:



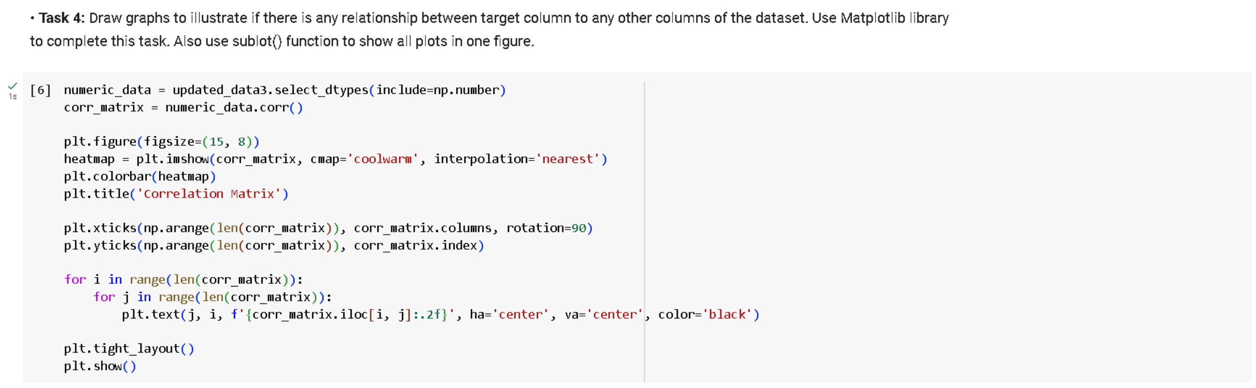
• **Task 3:** Draw graphs to analyze the frequency distributions of the features. Use Matplotlib library to complete this task. Draw all the plots in a single figure so that all plots can be seen in one diagram (use subplot() function).

```
[5]
numerical_columns = updated_data3.select_dtypes(include='float64').columns
object_columns = updated_data1.select_dtypes(include='object').columns

num_plots = len(numerical_columns) + len(object_columns)
cols = 3
rows = 4
fig, axs = plt.subplots(rows, cols, figsize=(14, 15))

axs = axs.flatten() if rows > 1 else [axs]

for i, column in enumerate(numerical_columns):
    ax = axs[i]
    updated_data3[column].plot(kind='hist', ax=ax, title=column)

for i, column in enumerate(object_columns, start=len(numerical_columns)):
    ax = axs[i]
    value_counts = updated_data1[column].value_counts()
    value_counts.plot(kind='bar', ax=ax, title=column)

for i in range(num_plots, len(axs)):
    axs[i].axis('off')

plt.tight_layout()
plt.show()
```

The provided code segment generates visualizations for both numerical and categorical data in the dataset. It separates columns into numerical and categorical types, creating histograms for numerical columns to display their distributions and bar charts for categorical columns showcasing the frequency of different categories. This approach aids in comprehending the spread and characteristics of numerical values via histograms while presenting the categorical data's distribution and prevalence through bar charts. The code neatly arranges these plots in subplots, ensuring a clear and organized visualization of the dataset's diverse data types.



Task 4:

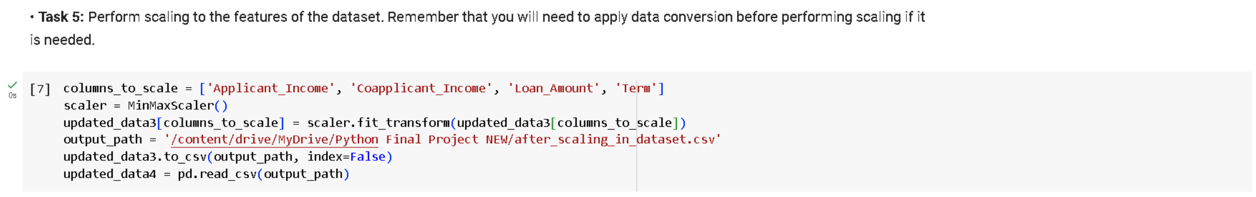


• **Task 4:** Draw graphs to illustrate if there is any relationship between target column to any other columns of the dataset. Use Matplotlib library to complete this task. Also use sublot() function to show all plots in one figure.

```
[6] numeric_data = updated_data3.select_dtypes(include=np.number)
    corr_matrix = numeric_data.corr()

    plt.figure(figsize=(15, 8))
    heatmap = plt.imshow(corr_matrix, cmap='coolwarm', interpolation='nearest')
    plt.colorbar(heatmap)
    plt.title('Correlation Matrix')

    plt.xticks(np.arange(len(corr_matrix)), corr_matrix.columns, rotation=90)
    plt.yticks(np.arange(len(corr_matrix)), corr_matrix.index)

    for i in range(len(corr_matrix)):
        for j in range(len(corr_matrix)):
            plt.text(j, i, f'{corr_matrix.iloc[i, j]:.2f}', ha='center', va='center', color='black')

    plt.tight_layout()
    plt.show()
```

The code generates a correlation matrix heatmap to visually represent relationships between numerical features in the dataset. By computing correlations among these features and visualizing them using colors indicating strength and direction, the heatmap facilitates the identification of correlations. This aids in understanding feature interactions, assisting in feature selection and identifying potential multicollinearity.

Task 5:

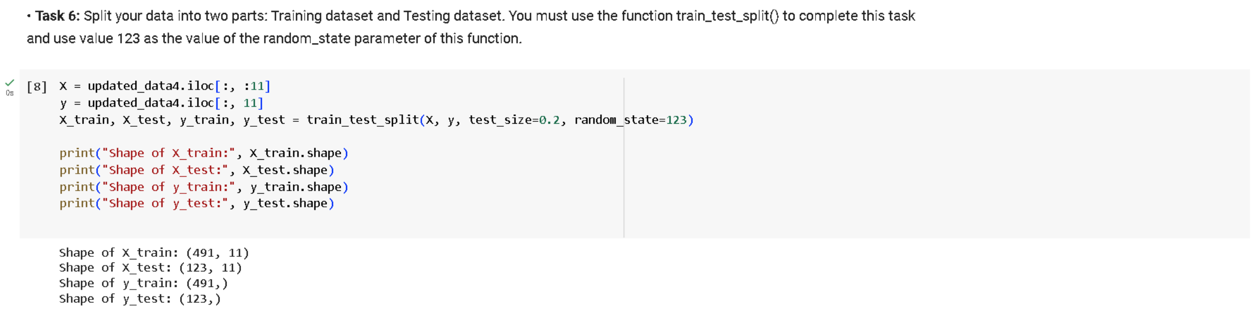


• **Task 5:** Perform scaling to the features of the dataset. Remember that you will need to apply data conversion before performing scaling if it is needed.

```
[7] columns_to_scale = ['Applicant_Income', 'Coapplicant_Income', 'Loan_Amount', 'Term']
    scaler = MinMaxScaler()
    updated_data3[columns_to_scale] = scaler.fit_transform(updated_data3[columns_to_scale])
    output_path = '/content/drive/MyDrive/Python Final Project NEW/after_scaling_in_dataset.csv'
    updated_data3.to_csv(output_path, index=False)
    updated_data4 = pd.read_csv(output_path)
```

As already all the data in our dataset is already well scaled we do not perform any operation for scaling in the dataset. But Scaling is essential in Machine Learning projects to ensure fair treatment among features with different scales or units. In machine learning models like Naïve Bayes the magnitude of features can significantly impact model performance. If features have varying scales, those with larger scales might disproportionately influence model training, leading to biased outcomes. Scaling brings features to a common scale, preventing dominance by any single feature.

Task 6:



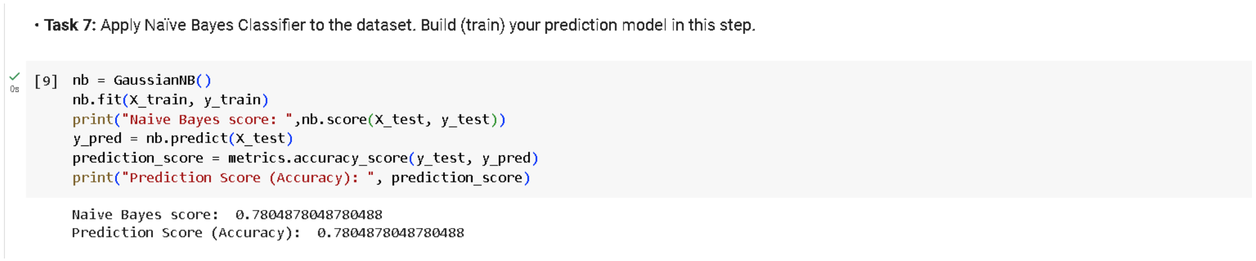
• **Task 6:** Split your data into two parts: Training dataset and Testing dataset. You must use the function train_test_split() to complete this task and use value 123 as the value of the random_state parameter of this function.

```
[8] X = updated_data4.iloc[:, :11]
    y = updated_data4.iloc[:, 11]
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=123)

    print("Shape of X_train:", X_train.shape)
    print("Shape of X_test:", X_test.shape)
    print("Shape of y_train:", y_train.shape)
    print("Shape of y_test:", y_test.shape)
```

```
Shape of X_train: (491, 11)
Shape of X_test: (123, 11)
Shape of y_train: (491,)
Shape of y_test: (123,)
```

The code snippet divides the dataset into features (X) and the target variable (y) for model training and evaluation. Using **iloc**, the first 11 columns are selected as features (X), while the 12th column ('Status') is chosen as the target variable (y). Subsequently, the **train_test_split** function from scikit-learn is utilized to split the dataset into training and testing sets, allocating 80% of the data for training (**X_train**, **y_train**) and 20% for testing (**X_test**, **y_test**). The **test_size** parameter, set to 0.2, designates the proportion of the dataset reserved for testing. Additionally, **random_state=123** ensures reproducibility by setting a seed for random data shuffling during the split, guaranteeing consistent results across multiple runs. This division is crucial for training models on a subset of data and assessing their performance on unseen data, facilitating model evaluation and validation.

Task 7:



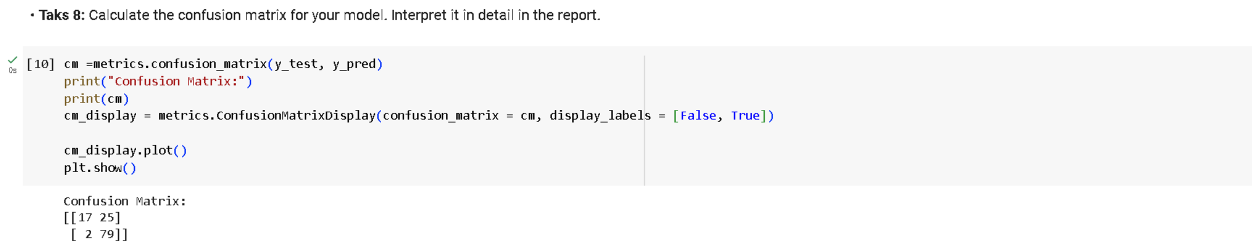
• **Task 7:** Apply Naïve Bayes Classifier to the dataset. Build (train) your prediction model in this step.

```
[9] nb = GaussianNB()
    nb.fit(X_train, y_train)
    print("Naive Bayes score: ",nb.score(X_test, y_test))
    y_pred = nb.predict(X_test)
    prediction_score = metrics.accuracy_score(y_test, y_pred)
    print("Prediction Score (Accuracy): ", prediction_score)
```

```
Naive Bayes score:  0.7804878048780488
Prediction Score (Accuracy):  0.7804878048780488
```

In Task 7 we perform the implementation and evaluation of a Naive Bayes classifier on the dataset. Initially, an instance of the Gaussian Naive Bayes model is created using **GaussianNB()**. Subsequently, the **fit()** method is employed to train the model on the training data (**X_train**, **y_train**). To assess the model's performance on unseen data, the **score()** function evaluates the model using the testing data (**X_test**, **y_test**), displaying the accuracy score of the Naive Bayes model in predicting the target variable. Additionally, predictions are made on the test data using **nb.predict(X_test)**, and the accuracy of these predictions is computed using **metrics.accuracy_score(y_test, y_pred)**. This accuracy score indicates the proportion of correctly predicted outcomes in the test set, providing insight into the model's predictive capabilities on unseen data.

Task 8:



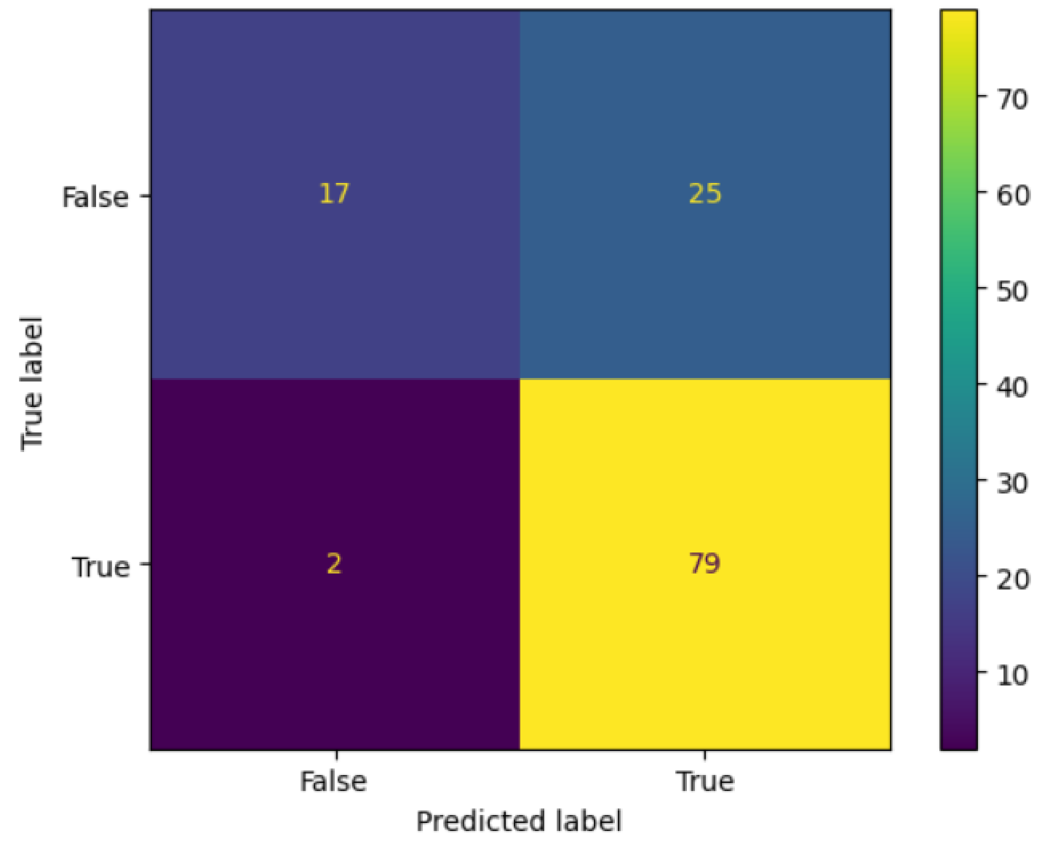
• **Taks 8:** Calculate the confusion matrix for your model. Interpret it in detail in the report.

```
[10] cm =metrics.confusion_matrix(y_test, y_pred)
     print("Confusion Matrix:")
     print(cm)
     cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = cm, display_labels = [False, True])

     cm_display.plot()
     plt.show()
```

```
Confusion Matrix:
[[17 25]
 [ 2 79]]
```

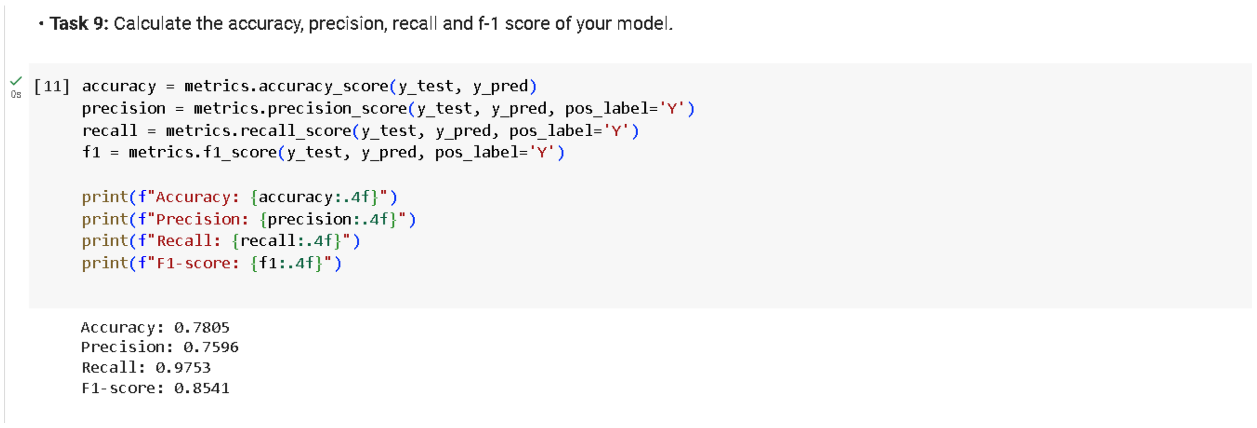
This code segment focuses on visualizing the confusion matrix for the Naive Bayes classifier's predictions on the test set. Initially, the confusion matrix is computed using **metrics.confusion_matrix(y_test, y_pred)**, where **y_test** contains the actual labels and **y_pred** holds the predicted labels. The confusion matrix captures the counts of true positive, true negative, false positive, and false negative predictions. To enhance the interpretability of the matrix, the **ConfusionMatrixDisplay** class from **sklearn.metrics** is employed.

```
Confusion Matrix:
[[17 25]
 [ 2 79]]
```



The plot showcases the model's accuracy in predicting both 'False' (representing 'N' status) and 'True' (representing 'Y' status) labels, aiding in assessing the classifier's strengths and weaknesses in differentiating between loan approvals and rejections.

Task 9:



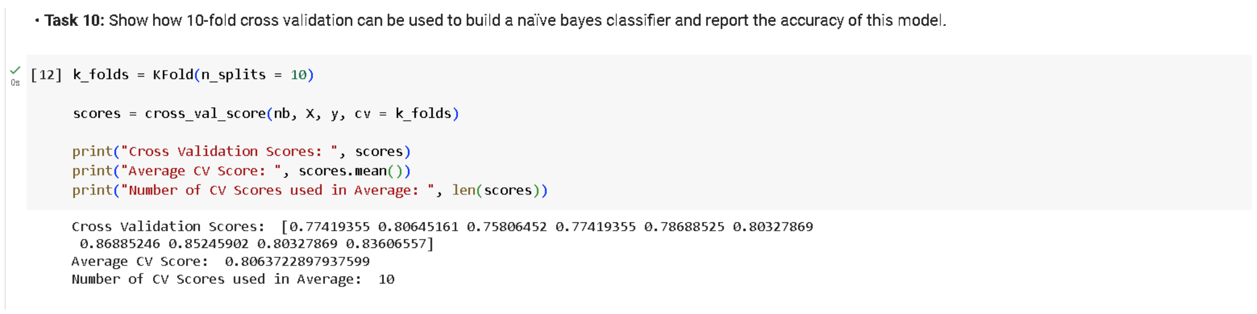
• **Task 9:** Calculate the accuracy, precision, recall and f-1 score of your model.

```
[11] accuracy = metrics.accuracy_score(y_test, y_pred)
     precision = metrics.precision_score(y_test, y_pred, pos_label='Y')
     recall = metrics.recall_score(y_test, y_pred, pos_label='Y')
     f1 = metrics.f1_score(y_test, y_pred, pos_label='Y')

     print(f"Accuracy: {accuracy:.4f}")
     print(f"Precision: {precision:.4f}")
     print(f"Recall: {recall:.4f}")
     print(f"F1-score: {f1:.4f}")
```

```
Accuracy: 0.7805
Precision: 0.7596
Recall: 0.9753
F1-score: 0.8541
```

In the Task 9 we calculates and displays multiple performance metrics for the Naive Bayes classifier applied to the dataset. It begins by computing the accuracy, precision, recall, and F1-score, all crucial metrics for evaluating classification models. The **accuracy_score()** function determines the overall accuracy of the model's predictions on the test set (**y_test**, **y_pred**). Then, specific metrics are computed using functions like **precision_score()**, **recall_score()**, and **f1_score()**, specifying the positive label as 'Y', reflecting the 'Y' class of the target variable 'Status'. Each metric provides distinct insights into the model's performance: accuracy measures overall correctness, precision gauges the model's ability to avoid false positives, recall assesses the model's capability to capture actual positives, and the F1-score combines precision and recall into a single metric. These scores aid in comprehensively evaluating the Naive Bayes classifier's effectiveness in predicting loan statuses based on the provided features.

Task 10:



• **Task 10:** Show how 10-fold cross validation can be used to build a naïve bayes classifier and report the accuracy of this model.

```
[12] k_folds = KFold(n_splits = 10)

     scores = cross_val_score(nb, X, y, cv = k_folds)

     print("Cross Validation Scores: ", scores)
     print("Average CV Score: ", scores.mean())
     print("Number of CV Scores used in Average: ", len(scores))
```

```
Cross Validation Scores:  [0.77419355 0.80645161 0.75806452 0.77419355 0.78688525 0.80327869
 0.86885246 0.85245902 0.80327869 0.83606557]
Average CV Score:  0.8063722897937599
Number of CV Scores used in Average:  10
```

To Evaluate our Model accuracy we performed the 10-fold cross validation and find out the accuracy of each fold and also average score after completing the 10- fold cross validation. Out Model Accuracy in Task 8 is close enough to the 10- fold cross validation mean accuracy score.