Assignment 1: Part 2

Name: Bismay Parija

Roll Number: 20CS30067

Import Libaries

In [61]:

```
# Import the necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score
```

Data Preprocessing

In [62]:

```
# Load the dataset
df = pd.read_csv('../../dataset/cross-validation.csv')
print(df.shape)
df.info()
```

```
(614, 13)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Loan_ID            614 non-null    object 
 1   Gender             601 non-null    object 
 2   Married            611 non-null    object 
 3   Dependents         599 non-null    object 
 4   Education          614 non-null    object 
 5   Self_Employed      582 non-null    object 
 6   ApplicantIncome    614 non-null    int64  
 7   CoapplicantIncome  614 non-null    float64
 8   LoanAmount         592 non-null    float64
 9   Loan_Amount_Term   600 non-null    float64
 10  Credit_History     564 non-null    float64
 11  Property_Area      614 non-null    object 
 12  Loan_Status        614 non-null    object 
dtypes: float64(4), int64(1), object(8)
memory usage: 62.5+ KB
```

In [63]:

```
# Handle missing values
# Fill missing numerical values with the mean and categorical values with the mode.
df['Gender'].fillna(df['Gender'].mode()[0], inplace=True)
df['Married'].fillna(df['Married'].mode()[0], inplace=True)
df['Dependents'].fillna(df['Dependents'].mode()[0], inplace=True)
```

```
df['Self_Employed'].fillna(df['Self_Employed'].mode()[0], inplace=True)
df['LoanAmount'].fillna(df['LoanAmount'].mean(), inplace=True)
df['Loan_Amount_Term'].fillna(df['Loan_Amount_Term'].mean(), inplace=True)
df['Credit_History'].fillna(df['Credit_History'].mode()[0], inplace=True)
```

In [64]:
```
# One-hot encode categorical variables
df = pd.get_dummies(df, columns=['Loan_ID','Gender', 'Married', 'Dependents', 'Educ
```

In [65]:
```
# Split the dataset into 80% for training and 20% for testing
X = df.drop('Loan_Status', axis=1)
y = df['Loan_Status']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.2, random_stat
# X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=.12
```

Train Logistic Regression Model

In [66]:
```
# Train a Logistic Regression model with the saga solver and no regularization
model = LogisticRegression(solver='saga', penalty=None, random_state=42)
model.fit(X_train, y_train)
```

```
c:\Users\HP\anaconda3\lib\site-packages\sklearn\linear_model\_sag.py:350: Convergenc
eWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
```

Out[66]:
```
▾                     LogisticRegression
LogisticRegression(penalty=None, random_state=42, solver='saga')
```

K-Fold Cross Validation

In [67]:
```
# Define the number of folds
num_folds = 5
fold_size = len(X_train) // num_folds

# Initialize lists to store metric scores
accuracy_scores = []
precision_scores = []
recall_scores = []

# Perform cross-validation
for i in range(num_folds):
    # Split the data into train and validation sets for this fold
    start_idx = i * fold_size
    end_idx = (i + 1) * fold_size
    val_X = X_train.iloc[start_idx:end_idx]
    val_y = y_train.iloc[start_idx:end_idx]
    train_X = pd.concat([X_train.iloc[:start_idx], X_train.iloc[end_idx:]])
    train_y = pd.concat([y_train.iloc[:start_idx], y_train.iloc[end_idx:]])

    # Train the model on the training set
    model.fit(train_X, train_y)

    # Predict on the validation set
    val_pred = model.predict(val_X)

    # Calculate accuracy, precision, and recall for this fold
    accuracy = accuracy_score(val_y, val_pred)
    precision = precision_score(val_y, val_pred, pos_label='Y')
    recall = recall_score(val_y, val_pred, pos_label='Y')

    accuracy_scores.append(accuracy)
    precision_scores.append(precision)
    recall_scores.append(recall)
```

```
c:\Users\HP\anaconda3\lib\site-packages\sklearn\linear_model\_sag.py:350: Convergenc
eWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
c:\Users\HP\anaconda3\lib\site-packages\sklearn\linear_model\_sag.py:350: Convergenc
eWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
c:\Users\HP\anaconda3\lib\site-packages\sklearn\linear_model\_sag.py:350: Convergenc
eWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
c:\Users\HP\anaconda3\lib\site-packages\sklearn\linear_model\_sag.py:350: Convergenc
eWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
c:\Users\HP\anaconda3\lib\site-packages\sklearn\linear_model\_sag.py:350: Convergenc
eWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
```

Calculating Metrics

In [68]:

```
# Calculate mean scores across all folds
mean_accuracy = np.mean(accuracy_scores)
mean_precision = np.mean(precision_scores)
mean_recall = np.mean(recall_scores)

# Print the mean scores
print("Mean Accuracy:", mean_accuracy)
print("Mean Precision:", mean_precision)
print("Mean Recall:", mean_recall)
```

```
Mean Accuracy: 0.6959183673469387
Mean Precision: 0.6959183673469387
Mean Recall: 1.0
```