Assignment 1

Name: Pasupulety Chethan Krishna Venkat

Roll Number: 21CS30036

```
# import all the necessary libraries here
import pandas as pd
from sklearn.preprocessing import StandardScaler
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

df = pd.read_csv('../../dataset/cross-validation.csv')
print(df.shape)
df.head()
```

```
(614, 13)
```

```
    Loan_ID Gender Married Dependents     Education Self_Employed  \
0  LP001002   Male      No          0      Graduate            No   
1  LP001003   Male     Yes          1      Graduate            No   
2  LP001005   Male     Yes          0      Graduate           Yes   
3  LP001006   Male     Yes          0  Not Graduate            No   
4  LP001008   Male      No          0      Graduate            No   

   ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
0             5849                0.0         NaN             360.0   
1             4583             1508.0       128.0             360.0   
2             3000                0.0        66.0             360.0   
3             2583             2358.0       120.0             360.0   
4             6000                0.0       141.0             360.0   

   Credit_History Property_Area Loan_Status  
0             1.0         Urban           Y  
1             1.0         Rural           N  
2             1.0         Urban           Y  
3             1.0         Urban           Y  
4             1.0         Urban           Y  
```

```
# check for missing values
df.isnull().sum()
```

```
Loan_ID               0
Gender               13
Married               3
Dependents           15
Education             0
Self_Employed        32
ApplicantIncome       0
```

```
CoapplicantIncome     0
LoanAmount           22
Loan_Amount_Term     14
Credit_History       50
Property_Area         0
Loan_Status           0
dtype: int64
```

```
# preprocessing steps should be performed here

df = df.dropna()
print(df.shape)
df.head()
```

```
(480, 13)
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed
1	LP001003	Male	Yes	1	Graduate	No
2	LP001005	Male	Yes	0	Graduate	Yes
3	LP001006	Male	Yes	0	Not Graduate	No
4	LP001008	Male	No	0	Graduate	No
5	LP001011	Male	Yes	2	Graduate	Yes

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term
1	4583	1508.0	128.0	360.0
2	3000	0.0	66.0	360.0
3	2583	2358.0	120.0	360.0
4	6000	0.0	141.0	360.0
5	5417	4196.0	267.0	360.0

	Credit_History	Property_Area	Loan_Status
1	1.0	Rural	N
2	1.0	Urban	Y
3	1.0	Urban	Y
4	1.0	Urban	Y
5	1.0	Urban	Y

```
# separate the label and features here

X_df = df.drop(['Loan_Status'], axis=1)
y_df = df['Loan_Status']

print(X_df.shape)
print(y_df.shape)
```

```
(480, 12)
(480,)
```

```
# removing the Loan_ID column here
```

```
X_df = X_df.drop(['Loan_ID'], axis=1)
X_df.head()
```

```
  Gender Married Dependents     Education Self_Employed
ApplicantIncome  \
1   Male     Yes          1      Graduate            No
4583
2   Male     Yes          0      Graduate           Yes
3000
3   Male     Yes          0  Not Graduate            No
2583
4   Male      No          0      Graduate            No
6000
5   Male     Yes          2      Graduate           Yes
5417

   CoapplicantIncome  LoanAmount  Loan_Amount_Term  Credit_History  \
1             1508.0       128.0             360.0             1.0
2                0.0        66.0             360.0             1.0
3             2358.0       120.0             360.0             1.0
4                0.0       141.0             360.0             1.0
5             4196.0       267.0             360.0             1.0

  Property_Area
1         Rural
2         Urban
3         Urban
4         Urban
5         Urban
```

```
# changing the loan status column to 0 and 1 here
y_df = y_df.replace('N', 0)
y_df = y_df.replace('Y', 1)

y_df.head()
```

```
1    0
2    1
3    1
4    1
5    1
Name: Loan_Status, dtype: int64
```

```
categorical_cols = ['Gender', 'Married', 'Education', 'Self_Employed',
'Property_Area']

# Changing gender column to 0 and 1 here
X_df['Gender'] = X_df['Gender'].apply(lambda x: 1 if x == 'Male' else
0)
```

```
# Changing married column to 0 and 1 here
X_df['Married'] = X_df['Married'].apply(lambda x: 1 if x == 'Yes' else
0)

# Changing education column to 0 and 1 here
X_df['Education'] = X_df['Education'].apply(lambda x: 1 if x ==
'Graduate' else 0)

# Changing self employed column to 0 and 1 here
X_df['Self_Employed'] = X_df['Self_Employed'].apply(lambda x: 1 if x
== 'Yes' else 0)

# encoding the property area column here
X_final = pd.get_dummies(X_df, columns=['Property_Area'])

# Preprocess the 'Dependents' column to convert '3+' to a numeric
value
X_final['Dependents'] = X_final['Dependents'].replace('3+',
4).astype(float)

X_final.head()
```

```
   Gender  Married  Dependents  Education  Self_Employed
ApplicantIncome  \
1       1        1         1.0          1              0
4583
2       1        1         0.0          1              1
3000
3       1        1         0.0          0              0
2583
4       1        0         0.0          1              0
6000
5       1        1         2.0          1              1
5417

   CoapplicantIncome  LoanAmount  Loan_Amount_Term  Credit_History  \
1             1508.0       128.0             360.0             1.0
2                0.0        66.0             360.0             1.0
3             2358.0       120.0             360.0             1.0
4                0.0       141.0             360.0             1.0
5             4196.0       267.0             360.0             1.0

   Property_Area_Rural  Property_Area_Semiurban  Property_Area_Urban
1                 True                    False                False
2                False                    False                 True
3                False                    False                 True
4                False                    False                 True
5                False                    False                 True
```

```
print(X_final.shape)

# Standardize the numeric features
scaler = StandardScaler()
X_final[['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
'Loan_Amount_Term', 'Credit_History', 'Dependents']] =
scaler.fit_transform(X_final[['ApplicantIncome', 'CoapplicantIncome',
'LoanAmount', 'Loan_Amount_Term', 'Credit_History', 'Dependents']])
X_final.head()
```

```
(480, 13)
```

```
   Gender  Married  Dependents  Education  Self_Employed
ApplicantIncome  \
1       1        1    0.112352          1              0        -
0.137970
2       1        1   -0.704755          1              1        -
0.417536
3       1        1   -0.704755          0              0        -
0.491180
4       1        0   -0.704755          1              0
0.112280
5       1        1    0.929459          1              1
0.009319

   CoapplicantIncome  LoanAmount  Loan_Amount_Term  Credit_History  \
1          -0.027952   -0.208089          0.275542        0.413197
2          -0.604633   -0.979001          0.275542        0.413197
3           0.297100   -0.307562          0.275542        0.413197
4          -0.604633   -0.046446          0.275542        0.413197
5           0.999978    1.520245          0.275542        0.413197

   Property_Area_Rural  Property_Area_Semiurban  Property_Area_Urban
1                 True                    False                False
2                False                    False                 True
3                False                    False                 True
4                False                    False                 True
5                False                    False                 True
```

```
# Step 2: Train a Logistic Regression model with the SAGA solver (no
regularization penalty)

# Create the Logistic Regression model

model = LogisticRegression(solver='saga', penalty='none',
max_iter=10000)

# Slice the data into training and test sets
X_train1, X_test1, y_train1, y_test1 = train_test_split(X_final, y_df,
test_size=0.2, random_state=42)
```

```
# Train the model on the training data
model.fit(X_train1, y_train1)

# Make predictions on the test data
y_pred1 = model.predict(X_test1)

# Calculate the evaluation metrics for the model
accuracy1 = np.mean(y_pred1 == y_test1)
precision1 = np.sum((y_pred1 == 1) & (y_test1 == 1)) / np.sum(y_pred1
== 1)
recall1 = np.sum((y_pred1 == 1) & (y_test1 == 1)) / np.sum(y_test1 ==
1)

# Print the evaluation metrics
print(f"Accuracy before cross-validation: {accuracy1:.5f}")
print(f"Precision before cross-validation: {precision1:.5f}")
print(f"Recall before cross-validation: {recall1:.5f}")



# Step 3: Implement 5-fold cross-validation

# Create an array of indexes corresponding to our data
indexes = np.arange(len(X_final))

# Shuffle the indexes randomly
np.random.shuffle(indexes)

# Define the number of folds
num_folds = 5

# Calculate the size of each fold
fold_size = len(indexes) // num_folds

# Initialize lists to store evaluation metrics
accuracies = []
precisions = []
recalls = []

# Perform k-fold cross-validation
for i in range(num_folds):
    # Determine the current fold's start and end indexes
    start_idx = i * fold_size
    end_idx = (i + 1) * fold_size if i < (num_folds - 1) else
len(indexes)

    # Extract the current fold's indexes
    fold_indexes = indexes[start_idx:end_idx]
```

```
    # Create training and testing sets based on the fold indexes
    train_indexes = [idx for idx in indexes if idx not in
fold_indexes]
    X_train, y_train = X_final.iloc[train_indexes],
y_df.iloc[train_indexes]
    X_test, y_test = X_final.iloc[fold_indexes],
y_df.iloc[fold_indexes]

    # Train the model on the training data
    model.fit(X_train, y_train)

    # Make predictions on the test data
    y_pred = model.predict(X_test)

    # Calculate the evaluation metrics for the current fold
    accuracy = np.mean(y_pred == y_test)
    precision = np.sum((y_pred == 1) & (y_test == 1)) / np.sum(y_pred
== 1)
    recall = np.sum((y_pred == 1) & (y_test == 1)) / np.sum(y_test ==
1)

    # Append evaluation metrics to the respective lists
    accuracies.append(accuracy)
    precisions.append(precision)
    recalls.append(recall)

# Calculate and print the mean evaluation metrics across all folds
mean_accuracy = np.mean(accuracies)
mean_precision = np.mean(precisions)
mean_recall = np.mean(recalls)

# print new line
print('\n')


# Print the mean evaluation metrics
print(f"Mean Accuracy after 5-fold: {mean_accuracy:.5f}")
print(f"Mean Precision after 5-fold: {mean_precision:.5f}")
print(f"Mean Recall after 5- fold: {mean_recall:.5f}")
```

```
Accuracy before cross-validation: 0.82292
Precision before cross-validation: 0.80000
Recall before cross-validation: 1.00000


Mean Accuracy after 5-fold: 0.80833
Mean Precision after 5-fold: 0.79443
Mean Recall after 5- fold: 0.97349
```

```
c:\Users\cheth\anaconda3\envs\data\lib\site-packages\sklearn\
linear_model\_logistic.py:1182: FutureWarning: `penalty='none'`has
been deprecated in 1.2 and will be removed in 1.4. To keep the past
behaviour, set `penalty=None`.
  warnings.warn(
c:\Users\cheth\anaconda3\envs\data\lib\site-packages\sklearn\
linear_model\_logistic.py:1182: FutureWarning: `penalty='none'`has
been deprecated in 1.2 and will be removed in 1.4. To keep the past
behaviour, set `penalty=None`.
  warnings.warn(
c:\Users\cheth\anaconda3\envs\data\lib\site-packages\sklearn\
linear_model\_logistic.py:1182: FutureWarning: `penalty='none'`has
been deprecated in 1.2 and will be removed in 1.4. To keep the past
behaviour, set `penalty=None`.
  warnings.warn(
c:\Users\cheth\anaconda3\envs\data\lib\site-packages\sklearn\
linear_model\_logistic.py:1182: FutureWarning: `penalty='none'`has
been deprecated in 1.2 and will be removed in 1.4. To keep the past
behaviour, set `penalty=None`.
  warnings.warn(
c:\Users\cheth\anaconda3\envs\data\lib\site-packages\sklearn\
linear_model\_logistic.py:1182: FutureWarning: `penalty='none'`has
been deprecated in 1.2 and will be removed in 1.4. To keep the past
behaviour, set `penalty=None`.
  warnings.warn(
c:\Users\cheth\anaconda3\envs\data\lib\site-packages\sklearn\
linear_model\_logistic.py:1182: FutureWarning: `penalty='none'`has
been deprecated in 1.2 and will be removed in 1.4. To keep the past
behaviour, set `penalty=None`.
  warnings.warn(
```