In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
import math
import random
from sklearn.metrics import r2_score
from sklearn.preprocessing import StandardScaler,MinMaxScaler
from sklearn.utils import shuffle
from sklearn.metrics import accuracy_score,precision_score,recall_score
```

In [2]:
```
df = pd.read_csv("cross-validation.csv")
df.dropna()
```

Out[2]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coappl
1	LP001003	Male	Yes	1	Graduate	No	4583	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	
4	LP001008	Male	No	0	Graduate	No	6000	
5	LP001011	Male	Yes	2	Graduate	Yes	5417	
...	...	...	...	...	...	...	...	...
609	LP002978	Female	No	0	Graduate	No	2900	
610	LP002979	Male	Yes	3+	Graduate	No	4106	
611	LP002983	Male	Yes	1	Graduate	No	8072	
612	LP002984	Male	Yes	2	Graduate	No	7583	
613	LP002990	Female	No	0	Graduate	Yes	4583	

480 rows × 13 columns

In [3]:
```
df.insert(0, 'bias', '1')
df['Loan_Status'].replace(['N','Y'],[0, 1], inplace=True)
del df['Loan_ID']
df['Loan_Status'].replace(['N','Y'],[0, 1], inplace=True)
df['Gender'].replace(['Male','Female'],[0, 1], inplace=True)
df['Married'].replace(['No','Yes'],[0, 1], inplace=True)
df['Dependents'].replace(['0', '1', '2', '3+'],[0, 1,2,4], inplace=True)
df['Education'].replace(['Graduate', 'Not Graduate'],[0, 1], inplace=True)
df['Self_Employed'].replace(['No', 'Yes'],[0, 1], inplace=True)
df['Property_Area'].replace(['Urban', 'Rural', 'Semiurban'],[0, 1,2], inplace=True)
```

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   bias               614 non-null    object 
 1   Gender             601 non-null    float64
 2   Married            611 non-null    float64
 3   Dependents         599 non-null    float64
 4   Education          614 non-null    int64  
 5   Self_Employed      582 non-null    float64
 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    int64  
 12  Loan_Status        614 non-null    int64  
dtypes: float64(8), int64(4), object(1)
memory usage: 62.5+ KB
```

In [5]:
```
df.head()
df = shuffle(df)
```

In [37]:
```
data=np.array(df,dtype=float)
data=data[~np.isnan(data).any(axis=1)]
data.shape
X=data[:,:12]
Y=data[:,-1]
scaler = StandardScaler()
X = scaler.fit_transform(X)
X[:,0]+=1
```

spliting data into 5 parts

In [38]:
```
x1,x2,x3,x4,x5=np.array_split(X, 5)
y1,y2,y3,y4,y5=np.array_split(Y, 5)
mean_accuracy = []
mean_precision = []
mean_recall = []
```

In [39]:
```
def accuracy(y,y_p):
    acc = accuracy_score(y, y_p)
    mean_accuracy.append(acc)
    return acc
def precision(y,y_p):
    pre = precision_score(y, y_p)
    mean_precision.append(pre)
    return pre
def recall(y,y_p):
    re = recall_score(y, y_p)
    mean_recall.append(re)
    return re
```

In [40]:
```
def k_fold(clf,x_train,y_train,x_test,y_test):
    clf.fit(x_train, y_train)
    y_p=clf.predict(x_test)
    print( "accuracy : " +str (accuracy(y_test,y_p)))
    print("precision : " + str(precision(y_test,y_p)))
    print("recall : " +str(recall(y_test,y_p)))
    return clf
```

5th part used for test

In [41]:
```
clf = LogisticRegression(random_state=0,solver='saga',penalty='none',max_iter=4000)
x_temp=np.concatenate((x1, x2,x3,x4), axis = 0)
y_temp=np.concatenate((y1, y2,y3,y4), axis = 0)
k_fold(clf,x_temp,y_temp,x5,y5)
```

```
accuracy : 0.7291666666666666
precision : 0.7195121951219512
recall : 0.9516129032258065
```

Out[41]:
```
LogisticRegression(max_iter=4000, penalty='none', random_state=0, solver='saga')
```

4th part used for test

In [42]:
```
clf = LogisticRegression(random_state=0,solver='saga',penalty='none',max_iter=4000)
x_temp=np.concatenate((x1, x2,x3,x5), axis = 0)
y_temp=np.concatenate((y1, y2,y3,y5), axis = 0)
k_fold(clf,x_temp,y_temp,x4,y4)
```

```
accuracy : 0.8125
precision : 0.8125
recall : 0.9558823529411765
```

Out[42]:
```
LogisticRegression(max_iter=4000, penalty='none', random_state=0, solver='saga')
```

3rd part used for test

In [43]:
```
clf = LogisticRegression(random_state=0,solver='saga',penalty='none',max_iter=4000)
x_temp=np.concatenate((x1, x2,x4,x5), axis = 0)
y_temp=np.concatenate((y1, y2,y4,y5), axis = 0)
k_fold(clf,x_temp,y_temp,x3,y3)
```

```
accuracy : 0.8125
precision : 0.8076923076923077
recall : 0.9545454545454546
```

Out[43]:
```
LogisticRegression(max_iter=4000, penalty='none', random_state=0, solver='saga')
```

2nd part used for test

In [44]:
```
clf = LogisticRegression(random_state=0,solver='saga',penalty='none',max_iter=4000)
x_temp=np.concatenate((x1, x3,x4,x5), axis = 0)
y_temp=np.concatenate((y1, y3,y4,y5), axis = 0)
k_fold(clf,x_temp,y_temp,x2,y2)
```

```
accuracy : 0.7708333333333334
precision : 0.7710843373493976
recall : 0.9552238805970149
```

Out[44]:
```
LogisticRegression(max_iter=4000, penalty='none', random_state=0, solver='saga')
```

1st part used for test

In [45]:
```
clf = LogisticRegression(random_state=0,solver='saga',penalty='none',max_iter=4000)
x_temp=np.concatenate((x2, x3,x4,x5), axis = 0)
y_temp=np.concatenate((y2, y3,y4,y5), axis = 0)
k_fold(clf,x_temp,y_temp,x1,y1)
```

```
accuracy : 0.8541666666666666
precision : 0.8313253012048193
recall : 1.0
```

Out[45]:
```
LogisticRegression(max_iter=4000, penalty='none', random_state=0, solver='saga')
```

average accuracy,precision and recall

In [46]:
```
print("Mean Accuracy " + str(sum(mean_accuracy)/len(mean_accuracy)))
print("Mean Precision " + str(sum(mean_precision)/len(mean_precision)))
print("Mean Recall " + str(sum(mean_recall)/len(mean_recall)))
```

```
Mean Accuracy 0.7958333333333333
Mean Precision 0.7884228282736951
Mean Recall 0.9634529182618905
```

In [ ]:

In [ ]: