Credit Vard Default Prediction

```
# Step 1 : import library
import pandas as pd
```

```
# Step 2 : import data
Credit = pd.read_csv('https://github.com/ybifoundation/Dataset/raw/main/Credit%20Default.c
```

```
Credit.head()
```

	Income	Age	Loan	Loan to Income	Default
0	66155.92510	59.017015	8106.532131	0.122537	0
1	34415.15397	48.117153	6564.745018	0.190752	0
2	57317.17006	63.108049	8020.953296	0.139940	0
3	42709.53420	45.751972	6103.642260	0.142911	0
4	66952.68885	18.584336	8770.099235	0.130990	1

```
Credit.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 5 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Income          2000 non-null   float64
 1   Age             2000 non-null   float64
 2   Loan            2000 non-null   float64
 3   Loan to Income  2000 non-null   float64
 4   Default         2000 non-null   int64  
dtypes: float64(4), int64(1)
memory usage: 78.2 KB
```

```
Credit.describe()
```

	Income	Age	Loan	Loan to Income	Default
count	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000
mean	45331.600018	40.927143	4444.369695	0.098403	0.141500

```
#count each category
Credit['Default'].value_counts()
```

```
0    1717
1     283
Name: Default, dtype: int64
```

```
#step 3 define target and feature
Credit.columns
```

```
Index(['Income', 'Age', 'Loan', 'Loan to Income', 'Default'], dtype='object')
```

```
y = Credit['Default']
```

```
x = Credit.drop(['Default'],axis=1)
```

```
#step 4 train test split
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,train_size=0.8,random_state=2529)
```

```
#check shape of sample
x_train.shape,x_test.shape,y_train.shape,y_test.shape
```

```
((1600, 4), (400, 4), (1600,), (400,))
```

```
#step 5 select model
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
```

```
#step 6 train ot fit model
model.fit(x_train,y_train)
```

```
▾ LogisticRegression
LogisticRegression()
```

```
model.intercept_
```

```
array([-0.00068221])
```

+ Code + Text

```
model.coef_
```

```
array([[-5.78014803e-05, -1.32895145e-01,  8.92153308e-04,
        -8.19443114e-05]])
```

```
# Step 7 : predict model
y_pred = model.predict(x_test)
```

```
y_pred
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0])
```

```
# Step 8 : model accuracy
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
```

```
confusion_matrix(y_test,y_pred)
```

```
array([[340,   7],
       [ 28,  25]])
```

```
accuracy_score(y_test,y_pred)
```

```
0.9125
```

```
print(classification_report(y_test,y_pred))
```

```
                precision    recall  f1-score   support

             0       0.92      0.98      0.95       347
             1       0.78      0.47      0.59        53

      accuracy                           0.91       400
     macro avg       0.85      0.73      0.77       400
  weighted avg       0.91      0.91      0.90       400
```

✓ 0s completed at 4:16 PM