In [1]:
```python
import pandas
import numpy as np
import  matplotlib
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import linear_model
from sklearn.model_selection import train_test_split
from sklearn import linear_model
```

In [2]:
```python
#Step 1 Analysis
data = pandas.read_csv("loan_old.csv")


has_missing = data.isnull().sum().sum()>0
if has_missing: print("The Data Has Missing values")
else: print("The data does not contain any missing values")

for j in range(data.shape[1]):
        print( data.columns[j]," ",data.iloc[:,j].isnull().sum())


print("-------------------------------------------\n")
column_types = data.dtypes
# Print the data types
print(column_types)

print("-------------------------------------------\n")
```

```
The Data Has Missing values
Loan_ID   0
Gender   13
Married   3
Dependents   15
Education   0
Income   0
Coapplicant_Income   0
Loan_Tenor   15
Credit_History   50
Property_Area   0
Max_Loan_Amount   25
Loan_Status   0
-------------------------------------------

Loan_ID                object
Gender                 object
Married                object
Dependents             object
Education              object
Income                  int64
Coapplicant_Income    float64
Loan_Tenor            float64
Credit_History        float64
Property_Area          object
Max_Loan_Amount       float64
Loan_Status            object
dtype: object
-------------------------------------------
```

In [3]:
```python
#Describe numeric data
pd.set_option('display.max_columns',None)
```

```python
print("------------\n")
numeric = data.select_dtypes(include=['int64','float64'])
feature_stats = numeric.describe()
print(feature_stats)


print("-------------------------------------------\n")




print("------------\n")
numeric = data.select_dtypes(include=['int64','float64'])
sns.pairplot(numeric)
plt.show()
```

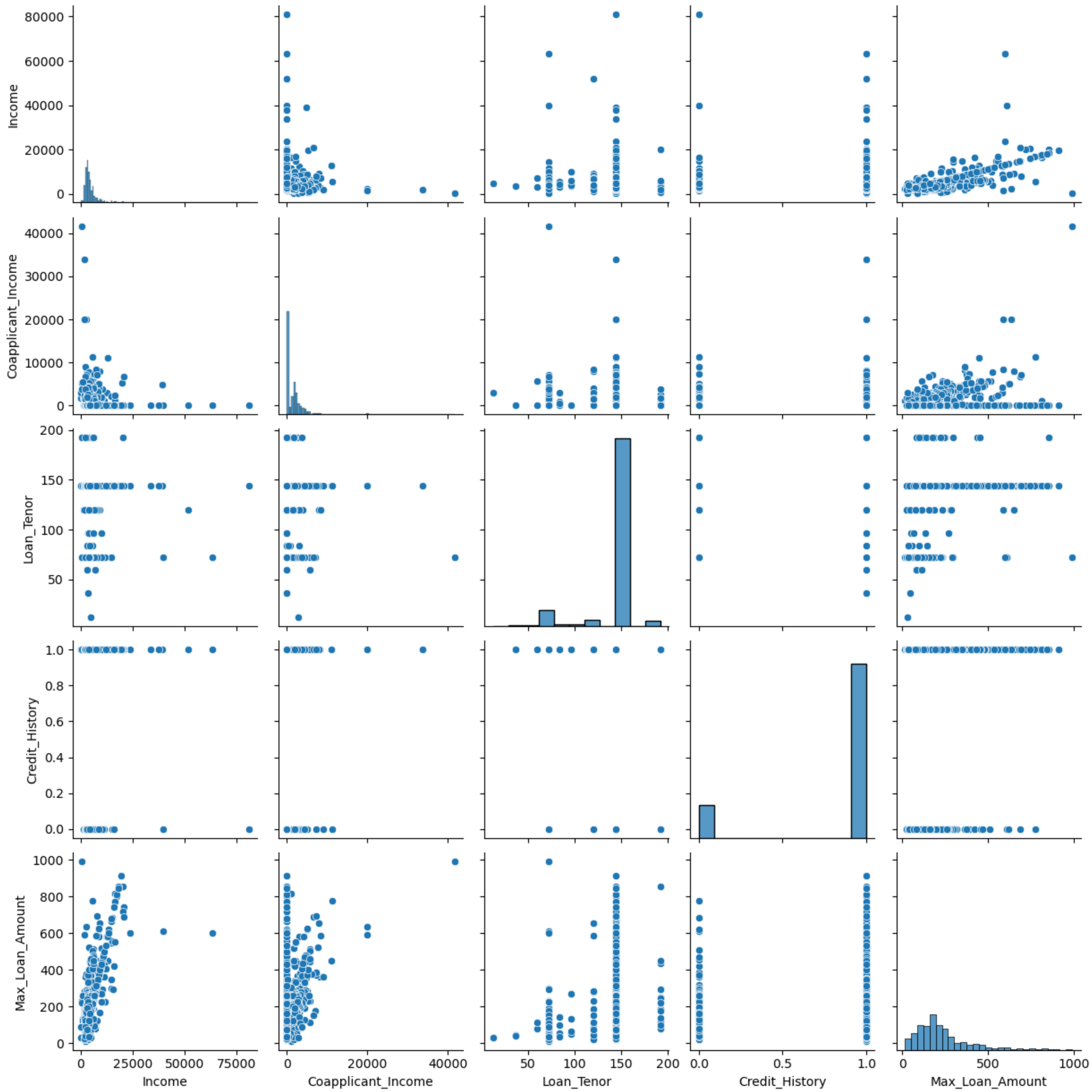
```
------------

             Income  Coapplicant_Income  Loan_Tenor  Credit_History  \
count    614.000000          614.000000  599.000000      564.000000   
mean    5403.459283         1621.245798  137.689482        0.842199   
std     6109.041673         2926.248369   23.366294        0.364878   
min      150.000000            0.000000   12.000000        0.000000   
25%     2877.500000            0.000000  144.000000        1.000000   
50%     3812.500000         1188.500000  144.000000        1.000000   
75%     5795.000000         2297.250000  144.000000        1.000000   
max    81000.000000        41667.000000  192.000000        1.000000   

       Max_Loan_Amount  
count       589.000000  
mean        230.499474  
std         161.976967  
min          12.830000  
25%         123.990000  
50%         190.370000  
75%         276.500000  
max         990.490000  
-------------------------------------------

------------

C:\Users\HP\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The figure
layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```



In [4]:
```
#Step 2 Preprocessing

#removing missing values
pd.set_option('display.max_columns',None)

data = data.dropna()
print(data)
#separate features and targets

x=  data.drop(columns=["Loan_ID","Max_Loan_Amount","Loan_Status"])
y1 = data['Max_Loan_Amount']
y2 = data['Loan_Status']
#shuffle data and split into training and test sets
x_train, x_test, y1_train, y1_test,y2_train,y2_test = train_test_split(x, y1,y2, test_si


x_train = x_train.to_numpy()

y1_train = y1_train.to_numpy()
y2_train = y2_train.to_numpy()
x_test = x_test.to_numpy()
```

```
y1_test = y1_test.to_numpy()
y2_test = y2_test.to_numpy()
```

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

     Coapplicant_Income  Loan_Tenor  Credit_History Property_Area  \
1                1508.0       144.0             1.0         Rural   
2                   0.0       144.0             1.0         Urban   
3                2358.0       144.0             1.0         Urban   
4                   0.0       144.0             1.0         Urban   
5                4196.0       144.0             1.0         Urban   
..                  ...         ...             ...           ...   
609                 0.0       144.0             1.0         Rural   
610                 0.0        72.0             1.0         Rural   
611               240.0       144.0             1.0         Urban   
612                 0.0       144.0             1.0         Urban   
613                 0.0       144.0             0.0     Semiurban   

     Max_Loan_Amount Loan_Status  
1             236.99           N  
2              81.20           Y  
3             179.03           Y  
4             232.40           Y  
5             414.50           Y  
..               ...         ...  
609            76.16           Y  
610            33.47           Y  
611           348.92           Y  
612           312.18           Y  
613           160.98           N  

[513 rows x 12 columns]
```

In [5]:
```
#Encoding data
xdata_encdoers = []
ydata_encoders = []
#Encode X_train Categorical  columns
for j in range(x_train.shape[1]):
        if(column_types.iloc[j+1]!= object ):
                continue
        le = LabelEncoder()
        le.fit(x_train[:,j])
        xdata_encdoers.append(le)
        x_train[:,j]= le.transform(x_train[:,j])


#Encode X_test Categorical Columns using x_training Encoders
index = 0
for j in range(x_test.shape[1]):
        if(column_types.iloc[j+1]!= object):
                continue
        le = xdata_encdoers[index]
        x_test[:,j]= le.transform(x_test[:,j])
        index += 1
```

```
#Encode  Loan Status column in training set
le = LabelEncoder()
le.fit(y2_train)
ydata_encoders.append(le)
y2_train = le.transform(y2_train)

#Encode  Loan Status column in test  set
le = ydata_encoders[0]

y2_test = le.transform(y2_test)

print("-------------------------------------------\nX_Traing After Encoding")
print(x_train)


print("Loan Status After Encoding")
print(y2_train)


print("-------------------------------------------\n")
```

```
-------------------------------------------
X_Traing After Encoding
[[1 1 3 ... 144.0 0.0 0]
 [1 1 0 ... 144.0 1.0 1]
 [1 1 1 ... 144.0 1.0 1]
 ...
 [1 0 2 ... 144.0 1.0 0]
 [1 1 2 ... 144.0 1.0 1]
 [1 0 0 ... 120.0 1.0 1]]
Loan Status After Encoding
[0 1 1 1 1 1 0 0 0 1 1 1 0 1 1 1 1 0 1 1 1 1 1 0 0 1 1 0 1 1 1 0 1 0 1 0 0
 0 1 1 0 0 0 1 0 1 1 1 1 1 0 1 0 1 1 1 0 1 1 0 0 1 1 0 1 1 0 0 1 1 1 0 1 0
 1 1 1 0 1 1 0 0 1 1 1 1 1 1 0 1 1 1 0 1 1 1 0 1 0 1 1 0 1 0 0 1 1 1 1 1 1
 1 1 1 0 1 0 1 0 1 0 1 1 1 1 1 1 0 1 0 1 1 0 1 1 1 1 0 0 1 0 0 1 1 0 1 1 0
 0 1 1 0 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 0 0 1 1
 1 0 0 1 0 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 0 0 0 0 0 0 1 1 1
 1 1 1 0 1 1 0 1 0 1 0 1 1 1 1 1 1 0 0 1 0 0 1 1 1 1 1 0 0 1 1 1 0 1 0 1 1
 1 1 0 1 1 1 1 1 1 0 0 0 1 1 1 1 1 0 0 1 0 1 1 1 0 1 1 1 1 0 1 1 1 1 1 1 0
 1 0 1 0 1 1 1 1 1 0 0 0 1 1 1 1 1 0 0 1 0 1 1 0 1 0 1 0 1 1 1 1 1 1 1 0 0
 1 1 0 1 1 1 1 0 1 1 1 0 0 1 0 1 1 1 0 0 1 0 1 1 1 1]
-------------------------------------------
```

In [6]:

```
# #decoding

#Standerdizing the Training and testing

for j in range(x_train.shape[1]-2):
        if(column_types.iloc[j+1]== object ):
                continue
        mean = np.mean(x_train[:,j])
        std = np.std(x_train[:,j])
        x_train[:,j]= (x_train[:,j]-mean)/std

        mean = np.mean(x_test[:, j])
        std = np.std(x_test[:, j])
        x_test[:, j] = (x_test[:, j] - mean) / std


print("X_train After Standerdization\n",x_train)

print("-------------------------------------------\n")
```

```
X_train After Standerdization
 [[1 1 3 ... 0.2624008495447276 0.0 0]
```

```
 [1 1 0 ... 0.2624008495447276 1.0 1]
 [1 1 1 ... 0.2624008495447276 1.0 1]
 ...
 [1 0 2 ... 0.2624008495447276 1.0 0]
 [1 1 2 ... 0.2624008495447276 1.0 1]
 [1 0 0 ... -0.8524145940831703 1.0 1]]
-------------------------------------------
```

In [7]:

```python
#3- linear Regression

Linearmodel = linear_model.LinearRegression()
Linearmodel.fit(x_train, y1_train)



print('Coefficients Of Linear Regression : \n', Linearmodel.coef_, " ", Linearmodel.inte
print('Correct predicitions ratio Of Linear Regression Model: %.2f'% Linearmodel.score(x

print("----------------------------------------------\nLogistic Regression\n-----------"
```

```
Coefficients Of Linear Regression :
 [ 11.17091248   3.85070977   7.42954434 -19.06747069 120.38123379
  64.32354493  48.77633104  -1.88440451 -12.64853779]   228.96188332550753
Correct predicitions ratio Of Linear Regression Model: 0.78
----------------------------------------------
Logistic Regression
-----------
```

In [8]:

```python
#4- Logistic Regression

class Logistic_Regression:
    thetas = []


    def fit(self,x,y):

         x = np.insert(x,0,np.ones(x.shape[0]),axis=1)

         self.thetas = np.ones(x.shape[1])

         alpha = 0.3
         max_iterations = 2000
         m = x.shape[0]
         for i in range(max_iterations):
             z = np.dot(x, self.thetas).astype('float')

             h = 1 / (1 + np.exp(-z))
            # print(self.cost_function(y,h))
             for j in range(self.thetas.shape[0]):
                par_der=(1/m)*sum((h-y)*x[:,j] )
                self.thetas[j] =self.thetas[j] -alpha*par_der

    def compute_cost(self, y_true, y_pred):
        m = len(y_true)
        epsilon = 1e-15
        # Adding a small constant to avoid log(0) and log(1) instability
        y_pred = np.clip(y_pred, epsilon, 1 - epsilon)
        # Formula: J(w, b) = - (1 / m) * Σ [y * log(a) + (1 - y) * log(1 - a)]
        cost = - (1 / m) * np.sum(y_true * np.log(y_pred) + (1 - y_true) * np.log(1 - y_
        return cost

    def score(self,x_test, y_true):
        y_pred = self.predict(x_test)
        correct_predictions = np.sum(y_pred == y_true)
        # Calculate accuracy
```

```python
        total_examples = len(y_true)
        accuracy = correct_predictions / total_examples
        return accuracy*100
    def predict(self,x):
        x = np.insert(x, 0, np.ones(x.shape[0]), axis=1)

        z = np.dot(x, self.thetas).astype('float')

        h = 1 / (1 + np.exp(-z))
        h[h<0.5]=0
        h[h>0.5]=1
        return h.astype('int')

Logmodel = Logistic_Regression()

Logmodel.fit(x_train, y2_train)
print('Accuracy of Logistic Regression model: %.2f'% Logmodel.score(x_test, y2_test),"%"
print("Thetas Of logistic Model: ",Logmodel.thetas)
```

```
Accuracy of Logistic Regression model: 83.77 %
Thetas Of logistic Model:  [-2.93734824 -0.02246384  0.57948517  0.17952386 -0.47642055
-0.11119164
 -0.09609713 -0.00497278  3.82666937 -0.01034753]
```

In [9]:

```python
#NewData

data = pandas.read_csv("loan_new.csv")


#Step 2 Preprocessing

#removing missing values

data = data.dropna()


x_new=  data.drop(columns=["Loan_ID"]).to_numpy()

#Encoding data
#Encode New Categorical Columns using x_training Encoders
index = 0
for j in range(x_new.shape[1]):
        if(column_types.iloc[j+1]!= object):
                continue
        le = xdata_encdoers[index]
        x_new[:,j]= le.transform(x_new[:,j])
        index += 1

#Standerdizing the Training and testing

for j in range(x_new.shape[1]-2):
        if(column_types.iloc[j+1]== object ):
                continue
        mean = np.mean(x_new[:,j])
        std = np.std(x_new[:,j])
        x_new[:,j]= (x_new[:,j]-mean)/std
print("-------------------------------------------\n")

print("New Data After Being Preprocessed ")

print(x_new)
print("-------------------------------------------\n")
```

```
-------------------------------------------


New Data After Being Preprocessed
```

```
[[1 1 0 ... 0.25159989387309867 1.0 2]
 [1 1 1 ... 0.25159989387309867 1.0 2]
 [1 1 2 ... 0.25159989387309867 1.0 2]
 ...
 [1 1 0 ... 0.25159989387309867 1.0 2]
 [1 1 0 ... 0.25159989387309867 1.0 0]
 [1 0 0 ... -2.8806525294105008 1.0 0]]
-------------------------------------------
```

In [11]:

```
print("Max Loan Predictions of New Data:")
y1_pred = Linearmodel.predict(x_new)


y2_pred = Logmodel.predict(x_new)


print(y1_pred)
le = ydata_encoders[0]
y2_pred = le.inverse_transform(y2_pred)
print("Predicted Loan Status of New Data: ")
print(y2_pred)
```

```
Max Loan Predictions of New Data:
[ 211.96274715  193.55999544  258.15988515  126.93141014  196.91110427
  109.15330386  188.19849053  326.07397079  183.49883575  122.22971866
  184.00538406  405.35485966  182.80272856  214.19461065  278.33666345
  206.93093814  519.35655749   51.33667807  147.60914274  -45.05020677
  128.18821188  324.61230061  751.7426232   360.610069     50.41630762
  108.12822574  262.90374092  204.85425239  211.99561132  178.72177252
  157.32959002  226.57085558  201.19807542  210.98388136  209.4574074
  232.20706592  144.1805222   160.4615706   312.28112468  144.73964227
  168.64782391  292.67157958  182.57083736  260.4714122    60.85296575
  182.04326425  134.51523283  169.67435485  130.73306066  256.47758118
   68.70558838  169.00676378  252.95214928  205.3370109   173.51112414
  183.92401483  252.17856456  181.50537929  168.6775916   274.06799634
  297.01666792  188.81448769   48.49272572  248.74916764  279.69596128
  248.52418098  212.91055426  249.5727658   285.17437296  257.46627785
  218.49303249 1932.17195144  296.53238779  306.6161256    41.59191905
  292.37111959  210.20978073  153.31036343  208.96237142  205.41376936
  457.73110502  266.51831851  241.54726749  270.02412099  249.45286169
  274.54632077  257.81248935  335.88415967  201.18346545  229.29445563
  173.16322776    3.18933401  190.40078208  228.87309571  193.60553576
  199.70145508  184.73423425  138.41141969  250.83713016  328.88660458
   93.6844649   171.75195399  223.82345385  142.161284    256.05169231
  217.03722123  330.33015265  371.1361683   193.99905559  229.81978973
  272.05973218  -11.94005361  186.3195453   174.21862948  226.11154486
  151.00011738   37.08252971  174.80667724  221.11316223  231.16240219
  210.0568708   173.52955579  258.92358622  103.41586187  329.53298598
  139.57994792  292.03764498  226.957283    212.68665611  259.61322224
  177.47065113  215.11063803  184.51303928  230.13532695  114.25580136
  306.44783685  256.69834335  300.54713456  273.46899504  310.87107578
  152.60248399  204.63024423  145.68958037  115.84648754  145.03532527
  231.23529306  217.65276279  178.72468801  177.45776601  201.61364422
  224.67962122   66.97442034  186.67634769  358.81891335  217.82981634
  237.0734987   267.26956793  270.59758953  192.20722077  297.82002307
  222.58066189  327.07432233  405.3196727   416.25523315   70.95199157
  166.50842147  266.05955976  219.54092576  414.99412479  210.98388136
  210.44219098  167.77158647  167.19106886  179.94061851  415.00085196
  157.31733698  213.81877404  145.2138012   212.29910444  284.4229741
  194.65857602  158.6414003   111.59907101  291.64011952  238.36062967
  233.10413837  129.39201334  -31.40187866  351.08259364  275.25573695
  228.2341236   221.55649021  237.83686756  162.01837821  212.71095789
  116.24993339  180.22896917  291.67893874  225.30924442  203.34445781
  854.38850003   12.87598937  264.79855823  159.54541242  197.82048117
  267.59327072  629.97939516  169.50763582  274.71524599  141.36687369
```

```
 221.74338163 191.27793389 205.28979139  98.69942824 268.25264508
 160.38216155   9.59566879 287.06236594 174.01779096 211.62820086
 201.07458463 162.55707964 178.13165722 269.54345094 271.03018195
 225.63979202 197.76881367 578.12824349 213.74921515 283.59126182
 261.76037892 277.55704501 177.19213165 167.58993594 269.56068926
 713.02346139 252.14117966 155.05376952 213.28428618 231.78609499
  58.05519448 201.7961242  187.16078664 218.9613478  297.70136137
 646.12501974 296.39837348 258.02759119 189.38151513 396.10389395
 227.58691045 240.03545293 152.10141297 169.41071853 171.22852131
 237.67752871 165.74902552 158.63162438 211.4556338  264.63018784
 236.01713415 164.62098636 297.35680776 185.75496595 272.77244317
 303.0836933  197.79373069 306.55070665 233.82043649  96.37991797
 139.26147997 189.56820404 172.5831517  216.5965415  209.62070624
 126.79954998 124.36303395 627.33091217 227.42439245 150.72184057
 228.86845016 329.92118664 231.89735299 335.3758687  210.73234677
 187.74626392 171.56900012 138.59482502 167.80484418 149.96026271
 253.20707679 126.34156301 306.87122723  31.84002636 193.46945202
 262.33086916 300.68500998 234.16785858 116.56227435 187.30474307
 112.66083349 327.01558587 239.52197895 325.91904566  71.82627646
 315.55647719 223.75446683 126.27719119 250.59233515 204.0920824
 220.70189676 191.81649777 284.95034919 169.07196067]
Predicted Loan Status of New Data:
['Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y'
 'Y' 'Y' 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'Y' 'Y'
 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'N' 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'Y'
 'N' 'Y' 'Y' 'N' 'N' 'Y' 'N' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'N' 'Y'
 'N' 'Y' 'N' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'Y' 'Y' 'N'
 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'N' 'N' 'Y' 'Y' 'Y' 'N' 'N'
 'Y' 'N' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'N' 'Y' 'N' 'Y' 'Y' 'Y'
 'N' 'Y' 'Y' 'Y' 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'N' 'Y' 'Y' 'N'
 'N' 'Y' 'N' 'Y' 'Y' 'Y' 'Y' 'N' 'N' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y'
 'Y' 'Y' 'Y' 'Y' 'N' 'N' 'Y' 'Y' 'N' 'Y' 'N' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y'
 'Y' 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'N' 'Y' 'Y' 'Y'
 'Y' 'N' 'Y' 'Y' 'Y' 'Y' 'N' 'N' 'Y' 'Y' 'Y' 'Y' 'N' 'Y' 'N' 'Y' 'Y' 'Y'
 'Y' 'N' 'Y' 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'N' 'Y' 'N' 'Y' 'Y'
 'Y' 'Y' 'N' 'N' 'Y' 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y'
 'Y' 'N' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'N' 'Y' 'Y'
 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y'
 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'N' 'Y'
 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y']
```

In [ ]: