assign1

November 16, 2023

[1]:
```
%pip install seaborn
```

[2]:
```
import pandas
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plot
from sklearn import linear_model
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import r2_score
```

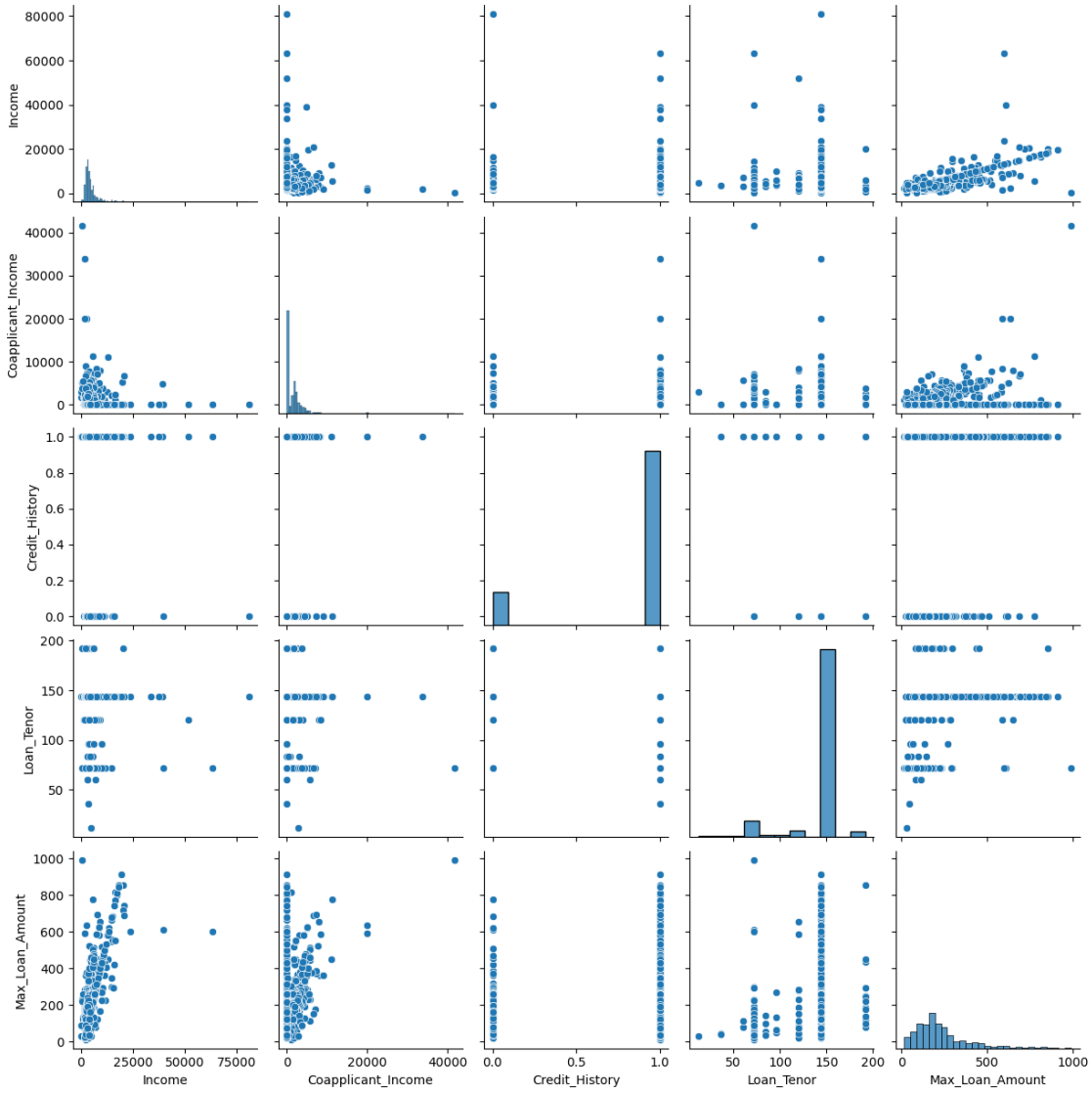
[3]:
```
#Load the "loan_old.csv" dataset.
data = pandas.read_csv('loan_old.csv')

#check whether there are missing values
missing_values = data.isnull().sum()
print('Missing values:\n',missing_values)

print('----------------------')
```

```
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
dtype: int64
----------------------
```

[4]:
```
#check the type of each feature (categorical or numerical)
data_types = data.dtypes

print('Data types:\n',data_types)

print('----------------------')


#visualize a pairplot between numercial columns
sns.
 ↪pairplot(data[['Income','Coapplicant_Income','Credit_History','Loan_Tenor','Max_Loan_Amount
#plot.show()
```

```
Data types:
 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
----------------------
```

[4]: <seaborn.axisgrid.PairGrid at 0x8c5bd78>



[5]:
```
#records containing missing values are removed
data.drop(columns=['Loan_ID'], inplace=True)
if data.isnull().values.any():
    data_cleaned_rows = data.dropna()


data_types = data_cleaned_rows.dtypes

print('-----------------------')

#check whether numerical features have the same scale
print('Numerical features describe: \n')
```

```
numerical_column_name=['Income','Coapplicant_Income','Loan_Tenor']
describe_data=data_cleaned_rows[numerical_column_name].describe()
print(describe_data)


print('----------------------')
```

```
----------------------
Numerical features describe:

             Income  Coapplicant_Income  Loan_Tenor
count    513.000000          513.000000  513.000000
mean    5030.730994         1486.627524  137.660819
std     4469.976643         2102.196620   23.139902
min      150.000000            0.000000   36.000000
25%     2889.000000            0.000000  144.000000
50%     3800.000000         1126.000000  144.000000
75%     5703.000000         2250.000000  144.000000
max    63337.000000        20000.000000  192.000000
----------------------
```

[6]:
```
label_encoder = LabelEncoder()


#categorical features and targets are encoded
for column_name in data_cleaned_rows.columns:
    if data_cleaned_rows[column_name].dtype == 'object':
        data_cleaned_rows.loc[:, column_name] = label_encoder.
 ↪fit_transform(data_cleaned_rows[column_name])
```

```
<ipython-input-6-4c130c73836e>:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data_cleaned_rows.loc[:, column_name] =
label_encoder.fit_transform(data_cleaned_rows[column_name])
<ipython-input-6-4c130c73836e>:7: DeprecationWarning: In a future version,
`df.iloc[:, i] = newvals` will attempt to set the values inplace instead of
always setting a new array. To retain the old behavior, use either
`df[df.columns[i]] = newvals` or, if columns are non-unique, `df.isetitem(i,
newvals)`
  data_cleaned_rows.loc[:, column_name] =
label_encoder.fit_transform(data_cleaned_rows[column_name])
<ipython-input-6-4c130c73836e>:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data_cleaned_rows.loc[:, column_name] =
label_encoder.fit_transform(data_cleaned_rows[column_name])
<ipython-input-6-4c130c73836e>:7: DeprecationWarning: In a future version,
`df.iloc[:, i] = newvals` will attempt to set the values inplace instead of
always setting a new array. To retain the old behavior, use either
`df[df.columns[i]] = newvals` or, if columns are non-unique, `df.isetitem(i,
newvals)`
  data_cleaned_rows.loc[:, column_name] =
label_encoder.fit_transform(data_cleaned_rows[column_name])
<ipython-input-6-4c130c73836e>:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data_cleaned_rows.loc[:, column_name] =
label_encoder.fit_transform(data_cleaned_rows[column_name])
<ipython-input-6-4c130c73836e>:7: DeprecationWarning: In a future version,
`df.iloc[:, i] = newvals` will attempt to set the values inplace instead of
always setting a new array. To retain the old behavior, use either
`df[df.columns[i]] = newvals` or, if columns are non-unique, `df.isetitem(i,
newvals)`
  data_cleaned_rows.loc[:, column_name] =
label_encoder.fit_transform(data_cleaned_rows[column_name])
<ipython-input-6-4c130c73836e>:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data_cleaned_rows.loc[:, column_name] =
label_encoder.fit_transform(data_cleaned_rows[column_name])
<ipython-input-6-4c130c73836e>:7: DeprecationWarning: In a future version,
`df.iloc[:, i] = newvals` will attempt to set the values inplace instead of
always setting a new array. To retain the old behavior, use either
`df[df.columns[i]] = newvals` or, if columns are non-unique, `df.isetitem(i,
newvals)`
  data_cleaned_rows.loc[:, column_name] =
label_encoder.fit_transform(data_cleaned_rows[column_name])
<ipython-input-6-4c130c73836e>:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

```
  data_cleaned_rows.loc[:, column_name] =
label_encoder.fit_transform(data_cleaned_rows[column_name])
<ipython-input-6-4c130c73836e>:7: DeprecationWarning: In a future version,
`df.iloc[:, i] = newvals` will attempt to set the values inplace instead of
always setting a new array. To retain the old behavior, use either
`df[df.columns[i]] = newvals` or, if columns are non-unique, `df.isetitem(i,
newvals)`
  data_cleaned_rows.loc[:, column_name] =
label_encoder.fit_transform(data_cleaned_rows[column_name])
<ipython-input-6-4c130c73836e>:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data_cleaned_rows.loc[:, column_name] =
label_encoder.fit_transform(data_cleaned_rows[column_name])
<ipython-input-6-4c130c73836e>:7: DeprecationWarning: In a future version,
`df.iloc[:, i] = newvals` will attempt to set the values inplace instead of
always setting a new array. To retain the old behavior, use either
`df[df.columns[i]] = newvals` or, if columns are non-unique, `df.isetitem(i,
newvals)`
  data_cleaned_rows.loc[:, column_name] =
label_encoder.fit_transform(data_cleaned_rows[column_name])
```

[7]:
```
#the features and targets are separated
x=data_cleaned_rows.drop(columns=['Max_Loan_Amount','Loan_Status'])
y=data_cleaned_rows[['Max_Loan_Amount','Loan_Status']]


#the data is shuffled and split into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,␣
 ↪random_state=100)
y_train_max_loan = y_train['Max_Loan_Amount']
y_test_max_loan = y_test['Max_Loan_Amount']
y_train_loan_status = y_train['Loan_Status']
y_test_loan_status = y_test['Loan_Status']
```

[8]:
```
#numerical features are standardized

x_train_mean=x_train[numerical_column_name].mean()
x_train_std=x_train[numerical_column_name].std()
x_test[numerical_column_name] = (x_test[numerical_column_name] - x_train_mean) /
 ↪ x_train_std
x_train[numerical_column_name] = (x_train[numerical_column_name] -␣
 ↪x_train_mean) / x_train_std
```

```
#Convert data to Numpy array
x_train = x_train.to_numpy().reshape((-1,9))
x_test = x_test.to_numpy().reshape((-1,9))
y_train_max_loan = y_train_max_loan.to_numpy()
y_test_max_loan = y_test_max_loan.to_numpy()
y_train_loan_status = y_train_loan_status.to_numpy()
y_test_loan_status = y_test_loan_status.to_numpy()
```

[9]:
```
#Fit a linear regression model
print("linear regression model: ")
model = linear_model.LinearRegression()
model.fit(x_train,y_train_max_loan)

print('Coefficients: \n', model.coef_, " ", model.intercept_)


#predict the loan amount
y_pred = model.predict(x_test)

r2 = r2_score(y_test_max_loan, y_pred)
print("R-squared score:", r2)

print('----------------------')
```

```
linear regression model:
Coefficients:
 [ 10.38137211   4.13996897   6.55078486 -17.41010274 119.21011348
  66.7485454   50.18745967   8.13355175 -11.4683694 ]   216.24700694689827
R-squared score: 0.7780935494194899
----------------------
```

[10]:
```
print('logistic regression model:')
#logistic regression model
'''
Logistic regression Algorithm:  (z)
1. Define the Sigmoid Function
2. Initialize Parameters (  and B)
3. Compute the Linear Combination: z =  1x1+ 2x2+…+ nx n+b
4. Apply the Sigmoid Function: y =  (z)
5. Define the Cost Function: J( )=- 1/m [y(i)log(y)+(1-y)log(1- y)]
6. Gradient Descent
     j= j- j/ j
    b = b + - j/ b
'''
def sigmoid(z):
    z = np.array(z,dtype=float)
```

```
    return 1 / (1 + np.exp(-z))
def initialize_parameters(dim):
    # Initialize weights and bias to zero
    theta = np.zeros((1, dim))
    b = 0
    return theta, b
def linear_combination(X, w, b):
    return np.dot(X, w.T) + b
def compute_cost(y, y_hat):
    m = len(y)
    return -1/m * np.sum(y * np.log(y_hat) + (1 - y) * np.log(1 - y_hat))
def predict(X, w, b):
    z = linear_combination(X, w, b)
    return sigmoid(z)
def gradient_descent(X, y, w, b, learning_rate, num_iterations):
    m = len(y)
    for i in range(num_iterations):
        # Compute linear combination
        z = linear_combination(X, w, b)
        # Apply sigmoid function and reshape
        y_hat = sigmoid(z).reshape(-1)
        # Compute cost
        cost = compute_cost(y, y_hat)
        # Compute gradients
        dw = 1/m * np.dot(X.T, (y_hat - y))
        db = 1/m * np.sum(y_hat - y)
        # Update parameters
        w -= (learning_rate * dw.T).astype(float) # Transpose dw before␣
 ↪updating weights
        b -= learning_rate * db
        # Print cost every 100 iterations
        if i % 200 == 0:
            print(f"Cost after iteration {i}: {cost}")
    return w, b
```

logistic regression model:

[11]:
```
w, b = initialize_parameters(x_train.shape[1])
# Set hyperparameters
learning_rate = 0.01
num_iterations = 2000
# Train the logistic regression modely_train_loan_status
w, b = gradient_descent(x_train,y_train_loan_status , w, b, learning_rate,␣
 ↪num_iterations)
# Print the trained parameters
print("Trained weights:", w)
print("Trained bias:", b)
```

```
Cost after iteration 0: 0.6931471805599453
Cost after iteration 200: 0.5999269197993258
Cost after iteration 400: 0.5881416530028638
Cost after iteration 600: 0.5788959205066386
Cost after iteration 800: 0.5708836171960125
Cost after iteration 1000: 0.5638389399585394
Cost after iteration 1200: 0.5575884650133229
Cost after iteration 1400: 0.5520044793455577
Cost after iteration 1600: 0.5469881804764495
Cost after iteration 1800: 0.5424606059968827
Trained weights: [[-0.0847963   0.23688659  0.03649964 -0.22506948 -0.01431119
-0.00553734
   0.00190672  1.1787928  -0.00819126]]
Trained bias: -0.19801784014892676
```

[12]:
```python
def Accuracy(X, y, w, b):
    predictions = predict(X, w, b)
    predictions_as_binary = ((predictions >= 0.5).astype(int)).reshape(-1)
    correct_predictions = (predictions_as_binary == y).sum()
    accuracy = (correct_predictions / len(y))*100
    return accuracy


accuracy = Accuracy(x_test, y_test_loan_status, w, b)
print("Accuracy: ",format(accuracy, ".2f"),'%')
```

```
Accuracy:  80.58 %
```

[13]:
```python
#load_new analysis and preprocessing part
# Load the "loan_new.csv" dataset.
print("-----------------------Loan_new.csv Part----------------------")
data = pandas.read_csv('loan_new.csv')

# check whether there are missing values
missing_values = data.isnull().sum()
print('Missing values:\n', missing_values)
```

```
-----------------------Loan_new.csv Part----------------------
Missing values:
 Loan_ID                0
Gender                11
Married                0
Dependents            10
Education              0
Income                 0
Coapplicant_Income     0
Loan_Tenor             7
Credit_History        29
```

```
Property_Area          0
dtype: int64
```

[14]:
```
# records containing missing values are removed and drop Loan_ID column
data.drop(columns=['Loan_ID'], inplace=True)
if data.isnull().values.any():
    newdata_cleaned_rows = data.dropna()



label_encoder = LabelEncoder()

for column_name in newdata_cleaned_rows.columns:
    if newdata_cleaned_rows[column_name].dtype == 'object':
        newdata_cleaned_rows.loc[:, column_name] = label_encoder.
 ↪fit_transform(newdata_cleaned_rows[column_name])
```

```
<ipython-input-14-1af13e3bd663>:12: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  newdata_cleaned_rows.loc[:, column_name] =
label_encoder.fit_transform(newdata_cleaned_rows[column_name])
<ipython-input-14-1af13e3bd663>:12: DeprecationWarning: In a future version,
`df.iloc[:, i] = newvals` will attempt to set the values inplace instead of
always setting a new array. To retain the old behavior, use either
`df[df.columns[i]] = newvals` or, if columns are non-unique, `df.isetitem(i,
newvals)`
  newdata_cleaned_rows.loc[:, column_name] =
label_encoder.fit_transform(newdata_cleaned_rows[column_name])
<ipython-input-14-1af13e3bd663>:12: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  newdata_cleaned_rows.loc[:, column_name] =
label_encoder.fit_transform(newdata_cleaned_rows[column_name])
<ipython-input-14-1af13e3bd663>:12: DeprecationWarning: In a future version,
`df.iloc[:, i] = newvals` will attempt to set the values inplace instead of
always setting a new array. To retain the old behavior, use either
`df[df.columns[i]] = newvals` or, if columns are non-unique, `df.isetitem(i,
newvals)`
  newdata_cleaned_rows.loc[:, column_name] =
label_encoder.fit_transform(newdata_cleaned_rows[column_name])
<ipython-input-14-1af13e3bd663>:12: SettingWithCopyWarning:
```

```
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  newdata_cleaned_rows.loc[:, column_name] =
label_encoder.fit_transform(newdata_cleaned_rows[column_name])
<ipython-input-14-1af13e3bd663>:12: DeprecationWarning: In a future version,
`df.iloc[:, i] = newvals` will attempt to set the values inplace instead of
always setting a new array. To retain the old behavior, use either
`df[df.columns[i]] = newvals` or, if columns are non-unique, `df.isetitem(i,
newvals)`
  newdata_cleaned_rows.loc[:, column_name] =
label_encoder.fit_transform(newdata_cleaned_rows[column_name])
<ipython-input-14-1af13e3bd663>:12: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  newdata_cleaned_rows.loc[:, column_name] =
label_encoder.fit_transform(newdata_cleaned_rows[column_name])
<ipython-input-14-1af13e3bd663>:12: DeprecationWarning: In a future version,
`df.iloc[:, i] = newvals` will attempt to set the values inplace instead of
always setting a new array. To retain the old behavior, use either
`df[df.columns[i]] = newvals` or, if columns are non-unique, `df.isetitem(i,
newvals)`
  newdata_cleaned_rows.loc[:, column_name] =
label_encoder.fit_transform(newdata_cleaned_rows[column_name])
<ipython-input-14-1af13e3bd663>:12: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  newdata_cleaned_rows.loc[:, column_name] =
label_encoder.fit_transform(newdata_cleaned_rows[column_name])
<ipython-input-14-1af13e3bd663>:12: DeprecationWarning: In a future version,
`df.iloc[:, i] = newvals` will attempt to set the values inplace instead of
always setting a new array. To retain the old behavior, use either
`df[df.columns[i]] = newvals` or, if columns are non-unique, `df.isetitem(i,
newvals)`
  newdata_cleaned_rows.loc[:, column_name] =
label_encoder.fit_transform(newdata_cleaned_rows[column_name])
```

[15]:
```
# numerical values are standardized
```

```
newdata_cleaned_rows.loc[:,numerical_column_name] =␣
 ↪(newdata_cleaned_rows[numerical_column_name]-x_train_mean)/x_train_std
x_new = newdata_cleaned_rows.to_numpy()


# use models to predict loan_Amount and status
loan_amount_prediction = model.predict(x_new)
loan_amount_prediction =[0 if i <0 else i for i in loan_amount_prediction]
status_prediction = predict(x_new, w, b)
status_prediction_YorN = ['Y' if prob >= 0.5 else 'N' for prob in␣
 ↪status_prediction]
```

```
<ipython-input-15-48ae3fc66ef9>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  newdata_cleaned_rows.loc[:,numerical_column_name] =
(newdata_cleaned_rows[numerical_column_name]-x_train_mean)/x_train_std
```

[16]:
```
print("-----------------------------------------")
print("prediction of loan Amount:\n",loan_amount_prediction)
print("-----------------------------------------")
print("prediction of loan status:\n",status_prediction_YorN)
```

```
-----------------------------------------
prediction of loan Amount:
 [202.2748468713639, 187.8160057358955, 251.93609454316316, 119.17516490024418,
199.3115700126549, 100.3697838818961, 166.45618941328135, 311.70807395303103,
181.5052246346297, 114.51613139053246, 177.3316146876495, 381.22335984873877,
172.67797616709387, 203.35091261713922, 275.3435034464074, 194.80782724205454,
531.0936736774288, 49.300814873737295, 144.133494850886, 0, 121.1087516309314,
327.3001676669593, 798.1142216249367, 361.0442475485388, 46.12616010046233,
100.16535818474702, 253.63922288672063, 198.98930940069954, 210.59081710388398,
174.04917391216372, 136.74690571681725, 224.42191755894518, 200.1481907325096,
200.98066593709711, 205.47259192886338, 224.8376419240588, 135.4715946190925,
157.85103757768007, 308.24648717226466, 136.02564184727444, 165.2656401313624,
281.00103765283325, 177.5597360456494, 257.87876141184273, 50.94226948995478,
174.11119110848915, 115.20495034741421, 160.08767608820767, 126.88939616365553,
245.592695863107, 53.788020392350575, 165.39186524328363, 243.78554470887332,
193.38510328478472, 156.99270105677172, 173.54082184882043, 252.02029120093277,
162.511917565493, 148.31614433736718, 262.7512120030065, 290.0409056062345,
177.94700340277183, 41.371151114860396, 243.0019505672791, 280.77239615129304,
248.68756477486966, 210.45438616556515, 237.08861839379432, 285.34304619450984,
248.05371652317598, 199.68489556976624, 1904.442713604283, 275.82715748926387,
293.93862593575864, 19.353547335735414, 299.6411132522919, 206.40162758908417,
```

142.765004413219, 203.93736555918176, 194.01845981715803, 441.2484986708317,
262.10584011838756, 235.42209761416098, 255.0681452398799, 236.61353978426718,
276.3264837691337, 256.58449819831856, 322.63787714486097, 189.13834919113964,
211.09119874828292, 163.01599954300886, 0, 184.46804783548941,
214.46689458999907, 183.0187790746427, 195.73863764685967, 174.5919575008133,
129.75483458285166, 249.18450372713494, 330.54888752303196, 91.23624509251945,
160.94694632697616, 211.010173676655, 132.7824889408845, 255.46581908827696,
209.6410529258886, 326.731997442748, 350.3672361777783, 183.0845322459134,
209.6448187875166, 260.2038741369138, 0, 180.74906068504447, 162.41816241382,
217.99339998257838, 141.41796431739982, 26.132136966005618, 165.45589016218264,
207.58871818974941, 227.30531218579975, 187.93038779395377, 166.7790415068738,
239.47415086085692, 85.4043472010311, 318.1323879447666, 130.64353430270296,
270.6517295622179, 217.4497829663289, 202.6679915865603, 262.34944361467825,
172.54940035946007, 202.9391619658533, 172.96526637209817, 216.52902573541243,
112.10679456911504, 307.26791643066565, 251.31326937012008, 295.5904269897159,
262.5766066283791, 310.84930110260484, 132.06268824218816, 194.68467470775698,
134.83614226136052, 97.12454591654338, 131.6609909066427, 231.05995785589138,
204.0788670761082, 167.08437005687372, 166.31609840806132, 189.75048804299252,
212.74552304218795, 53.94943763088119, 172.13769363344096, 358.1586785558074,
218.14554585580257, 223.4042481578519, 262.44269161623754, 268.3206468986232,
181.07585636656663, 297.08073641891684, 215.2143788164972, 313.65391305661274,
393.2310166196461, 413.0947993873217, 57.76451780075459, 156.90872733171602,
252.51824568198344, 202.81451661261642, 409.8157248442619, 200.98066593709711,
188.89046579704535, 162.73236250689456, 148.42701911387587, 172.82331488801753,
403.3599619415759, 148.833448055164, 210.9985561337663, 138.9539008353933,
208.93601768394788, 280.1539350257818, 186.56366313310042, 149.80127065707075,
103.18575159503617, 276.114714133409, 224.93375863056139, 220.7694091273446,
127.44303375364738, 0, 344.6791548093225, 271.68071935854266,
224.13015330382058, 217.76671620149205, 229.79585393627917, 149.19545202212365,
190.69128795747457, 107.79441717491316, 176.2370580964066, 289.97682435744855,
221.47240830835642, 182.7842058923392, 881.8572739003765, 4.709433621316862,
253.11851409095397, 156.49052623855908, 186.42112594909952, 262.63810450079677,
657.0018185206993, 162.39652291811802, 266.1537755831117, 133.47965697330503,
205.54077088937976, 183.24503484829512, 188.7430604803668, 90.82210720040621,
263.0568281920922, 151.58120787251104, 0, 267.61487608646024,
163.16139140614175, 203.4559676959597, 196.2106071080064, 140.8615573634067,
167.3080717633198, 272.4495365114593, 270.0698190255931, 221.82387189959306,
191.88552348812203, 562.175990248377, 203.71751133557115, 270.9162137070565,
261.46230061495817, 255.81378029998567, 165.3646863091512, 165.5189922947131,
269.56189347461833, 696.5885308887072, 230.74567208963845, 135.1799213239913,
204.6640499265425, 221.9183395069052, 46.0323039378593, 182.96807706404866,
182.60038985602893, 207.0791309357818, 286.2649188921341, 667.2482032464846,
285.1330454272906, 258.76361666574246, 179.5742957573406, 396.1694047226423,
222.80957263142682, 237.1359101506901, 141.98598388057496, 156.91394539344958,
170.42920388613615, 229.3990723243567, 155.50977304119763, 148.91203697894164,
198.01881152115237, 253.6530892805898, 215.29877963029696, 156.43840844818843,
283.14061294183136, 174.03569393455945, 259.59065176628894, 305.88031424251517,
187.91018814498696, 297.9292088527795, 220.18070064842973, 87.09469379290007,

```
129.51475105340114, 178.64085597613206, 163.60675609372333, 214.91757315415782,
201.7286526526044, 125.44660391521508, 112.02587432103131, 626.9677823430494,
215.4653912532629, 143.08756007169984, 219.5456671216549, 318.91424394583896,
219.89776907871837, 322.13419784651376, 210.72775036739048, 177.90085562846892,
167.15002270402692, 127.62568408844596, 163.2832944380162, 146.13524983673426,
238.51361438499248, 128.1124986539034, 306.9438611924777, 17.67589708064955,
183.13798751198001, 249.73096878230936, 287.7580857343165, 218.16054234957394,
108.90010721396105, 181.92346167081828, 101.63850750862835, 312.5303000003912,
232.3676029998381, 324.5952717303836, 48.93591388760609, 306.20502791565735,
219.0701905723905, 117.04251086753412, 247.32233759601675, 199.22738906708156,
215.4206235053921, 184.48011705047577, 279.78928290815116, 154.28822908568463]
--------------------------------------
prediction of loan status:
 ['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', 'Y', 'Y', 'Y', 'Y', 'Y', 'N',
'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N',
'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'N', 'N', 'Y', 'N', 'Y', 'Y',
'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'N', 'Y', 'Y', 'Y', 'Y',
'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', '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', 'Y', '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', 'Y', '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', 'Y', 'Y', 'Y', 'Y', 'N', 'Y',
'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y',
'Y', 'Y', 'N', 'Y', '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', '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']
```