linear-logistic-regression

April 8, 2024

Team Data: Name: Hazem Emam Mohamed Ali , ID: 20206015 Name: Ramez Ehab Talaat Riad , ID: 20206025 Name: Ahmed Tarek Mahmoud Mohamed , ID: 202060004 Name: Khaled Ahmed Sayed Hashem , ID: 20206019 Name: Omar Raafat Ali Hammad , ID: 20206041

This project's goal is to: build a linear regression model and a logistic regression to predict loan decisions and amounts

Solution Steps:

1. Data Analysis on "loan_old.csv" dataset
2. Data Preprocessing
3. Linear Regression Model Fitting
4. Linear Regression Model Evalutaion
5. Logistic Regression Model Implementation and Fitting
6. Accuracy Evaluation Function
7. Preforming all previous steps on "loan_new.csv" dataset

Step 1. Data Analysis on "loan_old.csv" dataset

[54]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

[55]:
```
df = pd.read_csv('./loan_data/loan_old.csv')
df.head()
```

[55]:
```
    Loan_ID Gender Married Dependents     Education  Income  \
0  LP001002   Male      No          0      Graduate    5849
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

   Coapplicant_Income  Loan_Tenor  Credit_History Property_Area  \
0                 0.0       144.0             1.0         Urban
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
```

```
   Max_Loan_Amount Loan_Status
0              NaN           Y
1           236.99           N
2            81.20           Y
3           179.03           Y
4           232.40           Y
```

[56]:
```python
# 1.i Checking for missing values
def get_missing_count(df):
  missing_count = df.isna().sum()
  return missing_count

count = get_missing_count(df)
count
# so we have some missing values in some columns ---->
```

[56]:
```
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
```

[57]:
```python
#1.ii Checking feature types (categorical or numberical)
df.dtypes
```

[57]:
```
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
```

[58]:
```
df.nunique()
# this shows us the numerical vs categorical features , even if the categorical␣
↪features are of type int64 or float64
# we will assume that numerical values are features which have high number of␣
↪unique values
# So non categorical (numerical features) are␣
↪['Income','Coapplicant_Income','Max_Loan_Amount']
```

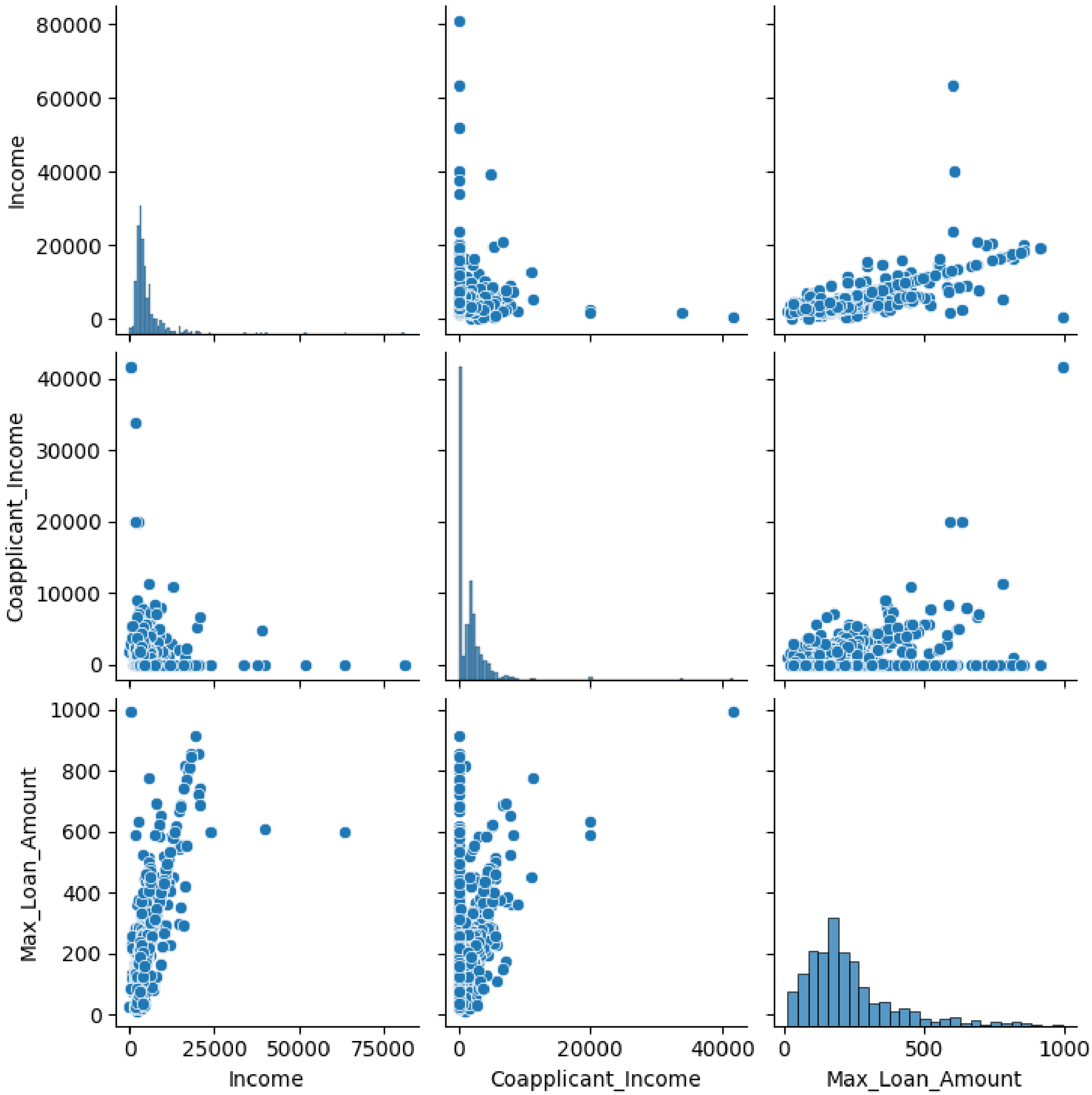
[58]:
```
Loan_ID               614
Gender                  2
Married                 2
Dependents              4
Education               2
Income                505
Coapplicant_Income    287
Loan_Tenor              9
Credit_History          2
Property_Area           3
Max_Loan_Amount       540
Loan_Status             2
dtype: int64
```

[59]:
```
#1.iii check whether numerical features have the same scale
numerical_features = ['Income','Coapplicant_Income','Max_Loan_Amount']
numerical_columns_only = df[numerical_features]
numerical_columns_only.describe()
# so features do not have the same scale ---->
```

[59]:

	Income	Coapplicant_Income	Max_Loan_Amount
count	614.000000	614.000000	589.000000
mean	5403.459283	1621.245798	230.499474
std	6109.041673	2926.248369	161.976967
min	150.000000	0.000000	12.830000
25%	2877.500000	0.000000	123.990000
50%	3812.500000	1188.500000	190.370000
75%	5795.000000	2297.250000	276.500000
max	81000.000000	41667.000000	990.490000

[60]:
```
#1.iv visualize a pairplot between numercial columns
sns.pairplot(numerical_columns_only)
plt.show()
```



Step 2. Data Preprocessing

[61]:
```
#2.i remove missing values records
df = df.dropna()
df.isna().sum()
```

[61]:
```
Loan_ID               0
Gender                0
Married               0
Dependents            0
Education             0
Income                0
Coapplicant_Income    0
```

```
Loan_Tenor           0
Credit_History       0
Property_Area        0
Max_Loan_Amount      0
Loan_Status          0
dtype: int64
```

[62]:
```
#2.ii separate features and targets
def seperate_features_targets(df):
  features = df.drop(['Loan_ID','Max_Loan_Amount' ,'Loan_Status'] , axis =1 )  ␣
↪ #takes all feature columns except the last 2 and the id column //as the id␣
↪is not corelated with data
  targets = df[['Max_Loan_Amount','Loan_Status']] # takes the second last␣
↪column as the target feature for linear regression model (continuous value)
  return features , targets

features,targets = seperate_features_targets(df)
features.head()
targets.head()
#features.dtypes
```

[62]:
```
   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
```

[63]:
```
#2.iii Split into training and testing sets
def split_dataset(features , targets):
  from sklearn.model_selection import train_test_split
  return train_test_split(features, targets, test_size= 0.20, random_state=42)␣
↪#20 of data set will be for testing
  #random_state , to achieve that same data splited for later use in logistic␣
↪regression model

# from sklearn.model_selection import train_test_split
# features_Train, features_Test, targets_Train, targets_Test =␣
↪train_test_split(features, targets, test_size= 0.20, random_state=42) #20 of␣
↪data set will be for testing
# #random_state , to achieve that same data splited for later use in logistic␣
↪regression model

features_Train, features_Test, targets_Train, targets_Test =␣
↪split_dataset(features , targets)
features_Train.count() #410 rows
```

```
features_Test.count() #103 rows
```

[63]:
```
Gender                 103
Married                103
Dependents             103
Education              103
Income                 103
Coapplicant_Income     103
Loan_Tenor             103
Credit_History         103
Property_Area          103
dtype: int64
```

[64]:
```
#2.iv Categorical features encoding
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
#['Income','Coapplicant_Income','Max_Loan_Amount'] -> numerical , So we will␣
 ↪encode all other columns
def encode_features(features):
  encoded_features = features
  encoded_features['Gender'] = le.fit_transform(features['Gender'])
  encoded_features['Married'] = le.fit_transform(features['Married'])
  encoded_features['Dependents'] = le.fit_transform(features['Dependents'])
  encoded_features['Education'] = le.fit_transform(features['Education'])
  encoded_features['Loan_Tenor'] = le.fit_transform(features['Loan_Tenor'])
  encoded_features['Credit_History'] = le.
 ↪fit_transform(features['Credit_History'])
  encoded_features['Property_Area'] = le.
 ↪fit_transform(features['Property_Area'])
  return encoded_features

# A. encoding training features
encoded_features_train = encode_features(features_Train)
encoded_features_train.head()

# B. encoding testing features
encoded_features_test = encode_features(features_Test)
encoded_features_test.head()
```

[64]:

	Gender	Married	Dependents	Education	Income	Coapplicant_Income	\
366	1	0	0	0	2500	0.0	
595	1	0	0	1	3833	0.0	
527	1	1	1	1	5285	1430.0	
184	0	1	0	0	3625	0.0	
598	1	1	0	0	9963	0.0	

	Loan_Tenor	Credit_History	Property_Area

```
366          5            1            1
595          4            1            0
527          4            0            1
184          4            1            1
598          4            1            0
```

[65]:
```
encoded_features_test['Credit_History'].unique()
encoded_features_train['Loan_Tenor'].unique()
```

[65]:
```
array([2, 6, 5, 1, 7, 3, 0, 4], dtype=int64)
```

[66]:
```
#2.v Categorical targets encoding
def encode_targets(targets):
  encoded_targets_train = targets
  encoded_targets_train['Loan_Status'] = le.
 ↪fit_transform(targets['Loan_Status'])
  return encoded_targets_train

# A. encoding training targets
encoded_targets_train = encode_targets(targets_Train)
encoded_targets_train.head()

# B. encoding testing targets
encoded_targets_test = encode_targets(targets_Test)
encoded_targets_test.head()
```

[66]:
```
     Max_Loan_Amount  Loan_Status
366            98.00            0
595           123.18            1
527           268.44            1
184           112.70            1
598           432.14            1
```

[67]:
```
# #2.vi numerical features standerdization
from sklearn.preprocessing import StandardScaler # data is standerdized over 0␣
 ↪using mean and standard deviation
standard_scaler = StandardScaler()
# A. numerical training features standerdization
def numerical_standardization(encoded_features):
  encoded_features['Income'] = standard_scaler.
 ↪fit_transform(encoded_features['Income'].values.reshape(-1, 1))
  encoded_features['Coapplicant_Income'] = standard_scaler.
 ↪fit_transform(encoded_features['Coapplicant_Income'].values.reshape(-1, 1))
  # encoded_features.head()
  return encoded_features

encoded_features_train = numerical_standardization(encoded_features_train)
```

```
encoded_features_train.head()

# B. numerical testing features standerdization
encoded_features_test = numerical_standardization(encoded_features_test)
encoded_features_test.head()
```

[67]:

```
     Gender  Married  Dependents  Education    Income  Coapplicant_Income  \
366       1        0           0          0 -0.412429           -0.754483
595       1        0           0          1 -0.204530           -0.754483
527       1        1           1          1  0.021929           -0.056773
184       0        1           0          0 -0.236971           -0.754483
598       1        1           0          0  0.751526           -0.754483

     Loan_Tenor  Credit_History  Property_Area
366           5               1              1
595           4               1              0
527           4               0              1
184           4               1              1
598           4               1              0
```

3. Linear Regression Model Fitting

[68]:

```
from sklearn import linear_model
linear_reg = linear_model.LinearRegression()
linear_reg.fit(encoded_features_train, encoded_targets_train['Max_Loan_Amount'])
linear_reg.score(encoded_features_train,␣
 ↪encoded_targets_train['Max_Loan_Amount'])#R^2 Score for training data
```

[68]: 0.8479516152839337

4. Linear Regression Model Evaluation

[69]:

```
linear_reg.score(encoded_features_test,␣
 ↪encoded_targets_test['Max_Loan_Amount']) #R^2 Score
```

[69]: 0.3173585332794533

[70]:

```
Y_Pred = linear_reg.predict(features_Test)
Y_Test = np.ravel(targets_Test['Max_Loan_Amount'])
Y_Pred = np.ravel(Y_Pred)
Y_Test_Pred = pd.DataFrame({"Y_Test": Y_Test, "Y_Pred": Y_Pred})
Y_Test_Pred.head()
```

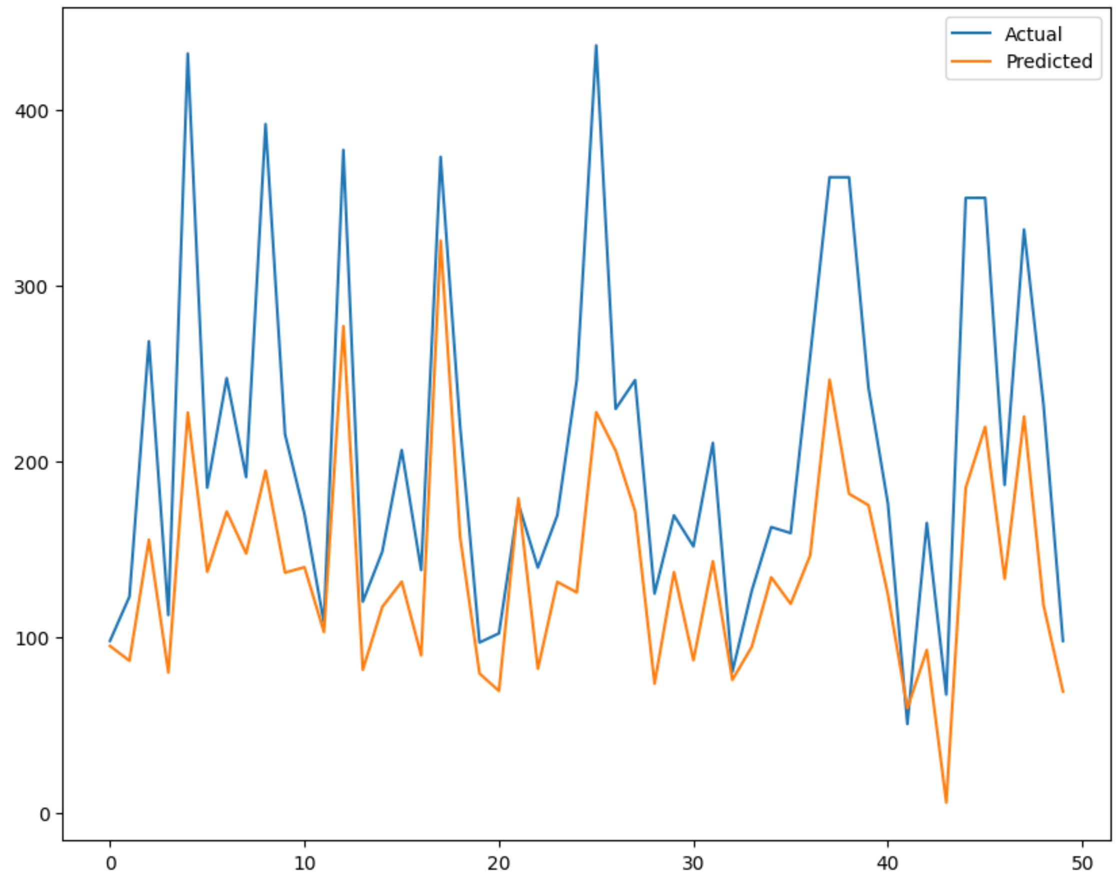
[70]:

```
   Y_Test      Y_Pred
0   98.00   95.110827
1  123.18   86.717486
2  268.44  155.590449
3  112.70   80.093416
```

4 432.14 227.883607

[71]:
```
plt.figure(figsize=(10, 8))
Y_Test_Pred = Y_Test_Pred
plt.plot(Y_Test_Pred[:50])
plt.legend(["Actual", "Predicted"])
```

[71]: <matplotlib.legend.Legend at 0x2a48ca05250>



[72]:
```
#4. Linear Regression Model Evalutaion
from sklearn.metrics import r2_score
reg_score = r2_score(Y_Test , Y_Pred)
reg_score
```

[72]: 0.3173585332794533

[73]:
```
# we will train data first using sklearn logistic regression model
from sklearn.linear_model import LogisticRegression
logistic_reg = LogisticRegression()
```

```
logistic_reg.fit(encoded_features_train, encoded_targets_train['Loan_Status'])
logistic_reg.score(encoded_features_train, encoded_targets_train['Loan_Status'])
```

[73]: 0.8097560975609757

[74]:
```
logistic_reg.score(encoded_features_test, encoded_targets_test['Loan_Status'])␣
↪#R^2 Score
```

[74]: 0.8058252427184466

Step 5. Logistic Regression Model Implementation and Fitting

[75]:
```
from LogisticRegressionFromScratch import LogisticRegression # our model from␣
↪scratch version

theta_X = np.zeros(encoded_features_train.shape[1])
itmes_number = len(encoded_features_train)/2 #number of rows in dataset

print(itmes_number)
logistic_reg =␣
↪LogisticRegression(encoded_features_train,encoded_targets_train['Loan_Status']␣
↪, 0 ,theta_X , 0.1,itmes_number)
theta_X_trained, theta_0_trained = logistic_reg.train(1000)
print("Trained theta_X:", theta_X_trained)
print("Trained theta_0:", theta_0_trained)

predictions = logistic_reg.predict(encoded_features_train)
print("Predictions:", predictions)
```

```
205.0
Trained theta_X: [-0.2097716   0.45879406  0.01292201 -0.49893415 -0.06985649
-0.06986174
 -0.09474067  3.22035245 -0.0352603 ]
Trained theta_0: -1.3500716077841408
Predictions: [1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1,
0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1,
0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0,
1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1,
1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1,
0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 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, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1,
```

```
1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 0, 1, 1, 1, 1, 0, 1, 1, 1]
```

Step 6. Accuracy Evaluation Function

```
[76]: acc = logistic_reg.accuracy(encoded_targets_train['Loan_Status'].tolist() ,␣
      ↪predictions)
      print(acc)
```

```
0.8097560975609757
```

Step 7. Preforming all previous steps on "loan_new.csv" dataset

---

```
[77]: new_df = pd.read_csv('./loan_data/loan_new.csv')
      new_df.head()
```

```
[77]:     Loan_ID Gender Married Dependents     Education  Income  \
      0  LP001015   Male     Yes          0      Graduate    5720
      1  LP001022   Male     Yes          1      Graduate    3076
      2  LP001031   Male     Yes          2      Graduate    5000
      3  LP001035   Male     Yes          2      Graduate    2340
      4  LP001051   Male      No          0  Not Graduate    3276

         Coapplicant_Income  Loan_Tenor  Credit_History Property_Area
      0                   0       144.0             1.0         Urban
      1                1500       144.0             1.0         Urban
      2                1800       144.0             1.0         Urban
      3                2546       144.0             NaN         Urban
      4                   0       144.0             1.0         Urban
```

```
[78]: count = get_missing_count(new_df)
      count
```

```
[78]: 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
```

```
[79]: #step 2.i
      new_df = new_df.dropna()
```

```
new_df.isna().sum()
```

[79]:
```
Loan_ID               0
Gender                0
Married               0
Dependents            0
Education             0
Income                0
Coapplicant_Income    0
Loan_Tenor            0
Credit_History        0
Property_Area         0
dtype: int64
```

[80]:
```
new_df = new_df.drop('Loan_ID', axis=1)
new_df.head()
```

[80]:
```
  Gender Married Dependents     Education  Income  Coapplicant_Income  \
0   Male     Yes          0      Graduate    5720                   0
1   Male     Yes          1      Graduate    3076                1500
2   Male     Yes          2      Graduate    5000                1800
4   Male      No          0  Not Graduate    3276                   0
5   Male     Yes          0  Not Graduate    2165                3422

   Loan_Tenor  Credit_History Property_Area
0       144.0             1.0         Urban
1       144.0             1.0         Urban
2       144.0             1.0         Urban
4       144.0             1.0         Urban
5       144.0             1.0         Urban
```

[81]:
```
#note that : step 2.ii the data seperation step is not needed , as the new␣
↪dataset contains only features with no targets
#note that : step 2.iii the data splitting is not needed as we will not split␣
↪our data into trainging and testing sets as they have no target values

#2.iv Categorical features encoding
new_encoded_features = new_df
new_encoded_features = encode_features(new_encoded_features)
new_encoded_features.head()
```

[81]:
```
   Gender  Married  Dependents  Education  Income  Coapplicant_Income  \
0       1        1           0          0    5720                   0
1       1        1           1          0    3076                1500
2       1        1           2          0    5000                1800
4       1        0           0          1    3276                   0
5       1        1           0          1    2165                3422
```

```
   Loan_Tenor  Credit_History  Property_Area
0           8               1              2
1           8               1              2
2           8               1              2
4           8               1              2
5           8               1              2
```

[82]:
```
#2.vi numerical features standerdization
new_encoded_features = numerical_standardization(new_encoded_features)
new_encoded_features.head()
```

[82]:
```
   Gender  Married  Dependents  Education    Income  Coapplicant_Income  \
0       1        1           0          0  0.208582           -0.656381
1       1        1           1          0 -0.349612           -0.013323
2       1        1           2          0  0.056577            0.115289
4       1        0           0          1 -0.307389           -0.656381
5       1        1           0          1 -0.541940            0.810649

   Loan_Tenor  Credit_History  Property_Area
0           8               1              2
1           8               1              2
2           8               1              2
4           8               1              2
5           8               1              2
```

[83]:
```
#Linear Regression model prediction for Loan Amounts
predictions = linear_reg.predict(new_encoded_features)
predictions
```

[83]:
```
array([ 314.81615981,  279.35185007,  341.9140789 ,  234.31048002,
        310.94776352,  196.77030759,  250.14380559,  419.41631463,
        281.97514086,  228.9269894 ,  257.24131245,  488.03545991,
        269.87279651,  301.68390928,  362.23828777,  265.11958473,
        647.07763871,   78.75890179,  248.32428172,  -17.3206071 ,
        240.35455383,  432.89658486,  830.05932024,  475.68291272,
         91.54435471,  206.25507588,  349.78569117,  286.32696561,
        308.11185602,  272.9183342 ,  220.33140743,  272.85517711,
        295.53942427,  308.66608194,  302.03109675,  335.95961475,
        236.78049154,  261.56257662,  407.04896951,  237.42069043,
        266.21040072,  385.38568241,  266.25378353,  340.90946955,
         90.64740084,  302.44344989,  205.33927092,  265.59553512,
        153.20916122,  313.05226589,   80.33932109,  272.10128037,
        361.02252135,  274.35781906,  228.5994533 ,  272.62144739,
        340.81280276,  274.92766941,  238.35426462,  374.79681793,
        361.4638233 ,  272.1513082 ,   73.92471724,  329.46260771,
        394.20597824,  343.09825637,  310.00844196,  330.58966901,
```

```
366.76475821,  371.52324535,  271.80252754, 2252.49992896,
298.01915326,  400.93423502,   65.50001035,  316.04976203,
308.03636141,  231.49867903,  290.77270181,  284.63213343,
541.98713478,  349.2697664 ,  334.03072215,  337.93038749,
319.24174462,  365.36010413,  348.09275992,  424.73211382,
275.18336555,  236.19808382,  259.73177097,   26.73583047,
280.39775897,  279.94209139,  282.66637847,  319.30127247,
272.08439266,  230.17480305,  323.03043932,  414.59846026,
185.33504526,  245.94339587,  260.31073445,  229.86341925,
333.56929463,  284.76717553,  391.31994654,  480.87797514,
278.08769785,  314.46408077,  362.31090741,   26.07591888,
282.2925624 ,  244.30831935,  294.32029555,  233.93051169,
 93.50643248,  272.27203394,  281.71537312,  296.59817792,
269.57418426,  204.3651713 ,  341.41295634,  144.0251643 ,
426.41299892,  237.66263681,  385.32855108,  308.41094794,
310.61577854,  353.61939405,  271.69740491,  291.13013776,
272.96800361,  292.04585512,  179.90865211,  391.1146879 ,
307.33777505,  383.131686  ,  384.86583154,  390.18104661,
214.91881686,  301.39109461,  216.24725259,  194.71822058,
212.9317539 ,  328.99876929,  259.75385388,  248.44360026,
252.62218921,  281.2736424 ,  307.1159985 ,   93.58155006,
265.18678647,  427.90285686,  313.80451717,  299.99014127,
339.1428292 ,  363.49118505,  270.87886673,  375.05722047,
313.4639531 ,  414.17267968,  535.83636847,  495.2788956 ,
109.00201887,  257.74117067,  332.0458028 ,  270.39846637,
518.80323534,  308.66608194,  275.51671504,  259.8576702 ,
224.72300896,  252.29927293,  448.86778259,  240.80711376,
304.50539406,  223.96498351,  298.0532792 ,  369.52183867,
270.51735161,  253.33836269,  199.47435655,  357.01281316,
301.93599043,  311.73290186,  214.25605224,   20.01410265,
446.41397565,  346.53036275,  317.46324693,  320.02788826,
268.84256989,  201.68723096,  267.03174076,  204.79964727,
274.07894432,  363.68946778,  295.94871244,  279.08874506,
998.12403385,   33.29023056,  353.10046667,  254.95126351,
270.91825616,  338.21611493,  736.94475948,  252.87896308,
305.4710186 ,  250.83925121,  295.3384249 ,  277.93982924,
264.90274969,  195.4589947 ,  352.55919799,  266.51067514,
 28.47343998,  275.80509136,  249.15544779,  290.56423484,
292.34954319,  215.18637903,  258.42306769,  370.07116738,
350.91402852,  302.73557035,  283.57444395,  701.16022706,
270.49517726,  317.2405694 ,  351.47614566,  328.70770682,
247.71301342,  252.43909149,  348.33167819,  861.71809606,
311.50860074,  222.3305883 ,  269.49630501,  337.51412026,
 99.69573758,  263.91975285,  265.13662078,  300.56850991,
407.58290336,  766.1808791 ,  400.8374251 ,  329.4480943 ,
283.93112504,  505.04284562,  316.5961704 ,  318.58960256,
240.86929386,  249.95001718,  273.01603507,  261.56613067,
```

```
256.49596663,  247.79432245,  270.65739239,  364.46248848,
315.93088199,  247.7359922 ,  369.015221  ,  263.11527404,
351.5243363 ,  396.3188354 ,  293.56320825,  339.56491036,
296.26534776,  159.4500641 ,  195.51874059,  271.51804488,
268.62223678,  308.2760194 ,  275.37585545,  239.40704715,
159.93981303,  751.15096635,  310.25879302,  243.25240752,
311.11364949,  392.13043747,  315.3803841 ,  424.15011483,
322.47496471,  280.56246648,  252.57701325,  204.46279387,
260.37600234,  197.80873926,  317.77836582,  166.28122567,
385.5538711 ,   48.88270706,  282.98263046,  345.19784355,
384.42868405,  297.316183  ,  222.4377007 ,  283.7152171 ,
155.51808283,  402.97486182,  344.1062275 ,  413.65286252,
 89.5630916 ,  398.43675625,  312.83938465,  211.67595094,
336.24779294,  278.62005936,  310.30846093,  289.98332163,
370.16897436,  211.07562179])
```

[84]:
```
#Logistic Regression model prediction for Loan Amounts
predictions = logistic_reg.predict(new_encoded_features)
predictions
```

[84]:
```
[1,
 1,
 1,
 1,
 1,
 1,
 0,
 1,
 1,
 1,
 1,
 0,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 0,
 1,
 1,
 1,
 1,
```

```
1,
1,
1,
0,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
0,
1,
1,
0,
1,
1,
1,
1,
0,
1,
1,
0,
0,
1,
0,
1,
1,
1,
1,
1,
1,
1,
1,
1,
0,
1,
0,
1,
```

```
0,
1,
1,
1,
1,
1,
1,
1,
1,
0,
1,
1,
1,
1,
1,
0,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
0,
0,
1,
1,
1,
0,
0,
1,
0,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
0,
```

```
1,
0,
1,
1,
1,
0,
1,
1,
1,
1,
1,
0,
1,
1,
1,
1,
1,
1,
1,
0,
1,
1,
0,
0,
1,
0,
1,
1,
1,
1,
0,
0,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
0,
0,
```

```
1,
1,
0,
1,
0,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
0,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
0,
1,
1,
1,
1,
0,
1,
1,
1,
1,
0,
0,
1,
1,
1,
1,
0,
1,
0,
1,
1,
```

```
1,
1,
0,
1,
1,
1,
0,
1,
1,
1,
1,
1,
1,
1,
0,
1,
0,
1,
1,
1,
1,
0,
0,
1,
1,
1,
0,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
0,
1,
1,
1,
1,
1,
1,
0,
1,
```

```
1,
1,
1,
1,
1,
0,
1,
1,
1,
1,
0,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
0,
1,
1,
1,
1,
1,
1,
0,
1,
1,
1,
1,
1,
1,
0,
1,
1,
1,
1,
```

```
1,
1,
1,
1,
1]
```