PREDICTING PERSONAL LOAN APROVAL USING MACHINE LEARNING

Team ID: LTVIP2023TMID07827

G REDDY PRAKASH (209E1A3320)

B JAYABABU (209E1A3307)

G SIVA KUMAR RAJU (219E5A3302)

RENDUCHINTALA CHANDRA SEKHAR YAJULU (209E1A05J1)

KOTHWALA SWATHI (209E1A3335)

PREDICTING PERSONAL LOAN APPROVAL USING MACHINE LEARNING

INTRODUCTION:

*This is a classification problem in which we need to classify whether the loan will be approved or not.

*The company wants to automate the loan eligibility process based on customer detail provided while filing out outline application forms.

*To automate this process, they have provide a dataset to identify the customer segments that are eligible for loan amounts so that they can specifically target these customer.

Purpose:

*The prediction model not only helps the applicant but also helps the bank by minimizing the risk and reducing the number of defaulters.

*It is done by predicting if the loan can be given to the person and basis of varies parameters like credit score, income, age, marital status, gender, etc.

Empathy map:

Advantages:

you to consolidate high-interest debt. ...

You can use them to finance your wedding or dream vacation. ...

They have predictable payment schedules....

D Personal loans are flexible in their uses

They help you pay for emergency expenses without draining your savings....

They enable

Low Interest Rates: Generally, bank loans have the cheapest interest rates. The rates you pay will be cheaper than other types of high interest loans, such as venture capital. As Biz fluent says, bank loans offer significantly lower interest rates than you will find with credit cards or overdraft.

Advantages of Loan Stock:

The money raised from the market does not have to be repaid, unlike debt financing which has a definite repayment schedule. read more. In the stock, the finance business keeps shares of its own as security to secure the finance

What is the advantage of loan portfolio

Portfolio lenders focus more on cash flow and the individual's business history rather than the borrower's income and other personal metrics. In some instances, investors may not have to provide personal tax returns if the cash flow being considered by the portfolio lender is based on rent rather than personal income.

Disadvantages:

Loans are not very flexible - you could be paying interest on funds you're not using. You could have trouble making monthly repayments if your customers don't pay you promptly. causing cashflow problems

What is the problem of loans?

How a Loan Works?

A problem loan is a scenario where borrowers fail to repay monthly loan installments. Thebank labels these loans as nonperforming assets (NPA). It can occur with either a commercialloan or a consumer loan. The loan is considered a default when borrowers miss consecutive repayments beyond the delinquency periods.

What are the disadvantages of loan prediction system?

The disadvantage of this model is that it emphasize different weights to each factor but in real life sometime loan can be approved on the basis of single strong factor only. which is not possible through this system. Loan Prediction is very helpful for employee of banks as well asfor the applicant also

Application:

Loan Prediction Project using Machine Learning in Python

Understanding the various features (columns) of the dataset.

Understanding Distribution of Categorical Variables: ...

Outliers of loan amount and application income: ...

Data Preparation for Model Building

*We have data of some predicted loans from history. So when there isname of some 'Data' there is a lot interesting for 'Data Scientists'.

Introduction Loan Prediction Problem

Welcome to this article on Loan Prediction Problem. Below is a brief introduction to this topic to get you acquainted with what you will be learning.

The Objective of the Article

This article is designed for people who want to solve binary classification problems using Python. By the end of this article, you will have the necessary skills and techniques required to solve such problems. This article provides you with sufficient theory and practiceknowledge to hone you

Problem Statement

Understanding the problem statement is the first and foremost step. This would help you give an intuition of what you will face ahead of time. Let us see the problem statement.

Dream Housing Finance company deals in all home loans. They have a presence across all urban, semi-urban and rural areas. Customers first apply for a home loan after that company validates the customer's eligibility for a loan. The company wants to automate the loan eligibility process (real-time) based on customer detail provided while filling out the online application form. These details are Gender, Marital Status, Education, Number of Dependents, Income, Loan Amount, Credit History, and others. To automate this process, they have given a problem to identify the customer segments, that are eligible for loan amounts so that they can specifically target these customers.

It is a classification problem where we have to predict whether a loan would be approved or not. In these kinds of problems, we have to predict discrete values based on a given set of independent variables (s). Classification can be of two types:

Binary Classification: - In this, we have to predict either of the two given classes. For example: classifying the "gender" as male or female, predicting the "result" as to win or loss , etc.

Multi Class Classification: - Here we have to classify the data into three or more classes. For example: classifying a "movie's genre" as comedy, action, or romantic, classifying "fruits" like oranges, apples, pears, etc.

Loan prediction is a very common real-life problem that each retail bank faces at least once in its lifetime. If done correctly, it can save a lot of man-hours at the end of a retail bank.

Although this course is specifically built to give you a walkthrough of the Loan Prediction problem, you can always refer to the content to get a comprehensive overview to solve a classification problem.

Conclusion:

...The conclusion derived from such assessments helps banks and other financial institutions

... CONCLUSION In this paper, various algorithms were implemented to predict loan defaulters....

This conclusion follows from the first and third columns of the table, which show that... credit standards helped predict loan growth in both periods and that the total effect of loan growth on...

Source code:

```
Loan Prediction
# importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

df = pd.read_csv("train.csv")
df.head()
```

```
     Loan_ID  Gender  Married Dependents     Education  Self_Employed \
0   LP001002    Male       No          0      Graduate             No
1   LP001003    Male      Yes          1      Graduate             No
2   LP001005    Male      Yes          0      Graduate            Yes
3   LP001006    Male      Yes          0  Not Graduate             No
```

```
4  LP001008   Male          No                0       Graduate                 No
  ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term\
0              5849                0.0         NaN             360.0
1              4583             1508.0       128.0             360.0
2              3000                0.0        66.0             360.0
3              2583             2358.0       120.0             360.0
4              6000                0.0       141.0             360.0

   Credit_History   Property_Area   Loan_Status
0              1.0         Urban              Y
1              1.0         Rural             N
2              1.0         Urban              Y
3              1. 0        Urban              Y
4              1.0         Urban              Y
```

df.tail()

```
      Loan_ID   Gender   Married   Dependents  Education  Self_Employed \
609  LP002978   Female       No           0       Graduate              No
610  LP002979     Male      Yes          3+       Graduate              No
611  LP002983     Male      Yes          1        Graduate              No
612  LP002984     Male      Yes          2        Graduate              No
613  LP002990  Female       No           0       Graduate             Yes

   ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term
\
609          2900                  0.0        71.0            360.0
610          4106                  0.0        40.0            180.0
611          8072                240.0       253.0            360.0
612          7583                  0.0       187.0            360.0
613          4583                  0.0       133.0            360.0

      Credit_History   Property_Area  Loan_Status
609              1.0         Rural           Y
610              1.0         Rural           Y
611              1.0         Urban           Y
612              1.0         Urban           Y
613              0.0     Semiurban           N
```

df.shape

(614, 13)

```
df.info()
```

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

dtypes: float64(4), int64(1), object(8)
memory usage: 62.5+ KB
```

```
df.isnull().sum()
```

```
Loan_ID               0
Gender               13
Married               3
Dependents           15
Education             0
Self_Employed        32
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           22
Loan_Amount_Term     14
Credit_History       50
Property_Area         0
Loan_Status           0
dtype: int64
```

```
df.describe() #bydefault interger column
  ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term \
count 614.000000           614.000000   592.000000           600.00000
mean5403.459283          1621.245798   146.412162           342.00000
std    6109.041673          2926.248369     85.587325            65.12041
min 150.000000                 0.000000       9.000000            12.00000
25% 2877.500000                0.000000   100.000000           360.00000
50% 3812.500000             1188.500000   128.000000           360.00000
75% 5795.000000             2297.250000   168.000000           360.00000
max 81000.000000          41667.000000   700.000000           480.00000
         Credit_History
Count    564.000000
mean         0.842199
std          0.364878
min          0.000000
25%          1.000000
50%          1.000000
75%          1.000000
max          1.000000
```

```
df['ApplicantIncome']

0      5849
1      4583
2      3000
3      2583
4      6000
       ...
609    2900
610    4106
611    8072
612    7583
613    4583
Name: ApplicantIncome, Length: 614, dtype: int64
```

```
df[['ApplicantIncome', 'LoanAmount']]

       ApplicantIncome       LoanAmount
0                 5849              NaN
1                 4583            128.0
2                 3000             66.0
3                 2583            120.0
```

4	6000	141.0
..	...	...
609	2900	71.0
610	4106	40.0
611	8072	253.0
612	7583	187.0
613	4583	133.0

[614 rows x 2 columns]

df.columns

Index(['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education',
'Self_Employed', 'ApplicantIncome', 'CoapplicantIncome',
'LoanAmount',
'Loan_Amount_Term', 'Credit_History', 'Property_Area',
'Loan_Status'],
dtype='object')

data preprocessing

df.isnull().sum()

```
Loan_ID               0
Gender                13
Married               3
Dependents            15
Education             0
Self_Employed         32
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount            22
Loan_Amount_Term      14
Credit_History        50
Property_Area         0
Loan_Status           0
dtype: int64
```

df.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
```

#	Column	Non-Null Count	Dtype
---	------	--------------	-----
0	Loan_ID	614 non-null	object
1	Gender	601 non-null	object
2	Married	611 non-null	object
3	Dependents	599 non-null	object
4	Education	614 non-null	object
5	Self_Employed	582 non-null	object
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	object
12	Loan_Status	614 non-null	object

dtypes: float64(4), int64(1), object(8)
memory usage: 62.5+ KB

```
# handle numerical missing data

df['LoanAmount'] = df['LoanAmount'].fillna(df['LoanAmount'].mean())
df['Loan_Amount_Term'] =
df['Loan_Amount_Term'].fillna(df['Loan_Amount_Term'].mean())
df['Credit_History'] =
df['Credit_History'].fillna(df['Credit_History'].mean())
df.isnull().sum()
```

```
Loan_ID              0
Gender               13
Married              3
Dependents           15
Education            0
Self_Employed        32
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           0
Loan_Amount_Term     0
Credit_History       0
Property_Area        0
Loan_Status          0
dtype: int64
```

```
# handle categorical missing data
```

```
df['Gender'].mode()[0]
'Male'
df['Gender'] = df['Gender'].fillna(df['Gender'].mode()[0])
df['Married'] = df['Married'].fillna(df['Married'].mode()[0])
df['Dependents'] = df['Dependents'].fillna(df['Dependents'].mode()[0])
df['Self_Employed'] =
df['Self_Employed'].fillna(df['Self_Employed'].mode()[0])
df.isnull().sum()

Loan_ID              0
Gender               0
Married              0
Dependents           0
Education            0
Self_Employed        0
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           0
Loan_Amount_Term     0
Credit_History       0
Property_Area        0
Loan_Status          0
dtype: int64
```

Exloratory data anlysis

```
# !pip install seaborn
# categorical data
import seaborn as sns
df['Gender'] = df['Gender'].astype('category')
# sns.countplot(df['Gender'])
sns.countplot(data=df, x='Gender')
plt.show()sns.countplot(data=df, x='Dependents')
plt.show()sns.countplot(data=df, x='Married')
plt.show()df.columns
Index(['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education',
'Self_Employed', 'ApplicantIncome', 'CoapplicantIncome',
'LoanAmount',
'Loan_Amount_Term', 'Credit_History', 'Property_Area',
'Loan_Status'],
dtype='object')
# numerical data
```

```
# sns.distplot(df.CoapplicantIncome)
sns.displot(df.CoapplicantIncome)
#sns.histplot(df.CoapplicantIncome)
<seaborn.axisgrid.FacetGrid at 0x22f7fd2e0c8>sns.displot(df.LoanAmount)
<seaborn.axisgrid.FacetGrid at
0x22f7ffafec8>sns.displot(df.Credit_History)
<seaborn.axisgrid.FacetGrid at 0x22f7e33ce08>df.head()
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed \
0	LP001002	Male	No	0	Graduate	No
1	LP001003	Male	Yes	1	Graduate	No
2	LP001005	Male	Yes	0	Graduate	Yes
3	LP001006	Male	Yes	0	Not Graduate	No
4	LP001008	Male	No	0	Graduate	No

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term \
0	5849	0.0	146.412162	360.0
1	4583	1508.0	128.000000	360.0
2	3000	0.0	66.000000	360.0
3	2583	2358.0	120.000000	360.0
4	6000	0.0	141.000000	360.0

	Credit_History	Property_Area	Loan_Status
0	1.0	Urban	Y
1	1.0	Rural	N
2	1.0	Urban	Y
3	1.0	Urban	Y
4	1.0	Urban	Y

```
# created new column
df['Total_income'] = df['ApplicantIncome']+df['CoapplicantIncome']
df.head()
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed \
0	LP001002	Male	No	0	Graduate	No
1	LP001003	Male	Yes	1	Graduate	No
2	LP001005	Male	Yes	0	Graduate	Yes
3	LP001006	Male	Yes	0	Not Graduate	No
4	LP001008	Male	No	0	Graduate	No

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term \
0	5849	0.0	146.412162	360.0
1	4583	1508.0	128.000000	360.0

```
2       3000                0.0                 66.000000          360.0
3       2583                2358.0              120.000000          360.0
4       6000                0.0                 141.000000         360.0
Credit_History Property_Area Loan_Status Total_income
0 1.0 Urban Y 5849.0
1 1.0 Rural N 6091.0
2 1.0 Urban Y 3000.0
3 1.0 Urban Y 4941.0
4 1.0 Urban Y 6000.0
# data transformation
df['ApplicantIncomeLog'] = np.log(df['ApplicantIncome'])
sns.displot(df.ApplicantIncomeLog)
<seaborn.axisgrid.FacetGrid at 0x22f7fe25948>df['CoapplicantIncomeLog']
= np.log(df['CoapplicantIncome'])
sns.displot(df["ApplicantIncomeLog"])
C:\Users\No_Name\AppData\Local\Programs\Python\Python37\lib\site
packages\pandas\core\arraylike.py:364: RuntimeWarning: divide by zero
encountered in log
result = getattr(ufunc, method)(*inputs, **kwargs)
<seaborn.axisgrid.FacetGrid at 0x22f7e338988>df['LoanAmountLog'] =
np.log(df['LoanAmount'])
sns.displot(df["LoanAmountLog"])
<seaborn.axisgrid.FacetGrid at
0x22f7fb31ac8>df['Loan_Amount_Term_Log'] =
np.log(df['Loan_Amount_Term'])
sns.displot(df["Loan_Amount_Term_Log"])
<seaborn.axisgrid.FacetGrid at 0x22f081a3308>df['Total_Income_Log'] =
np.log(df['Total_income'])
sns.displot(df["Total_Income_Log"])
<seaborn.axisgrid.FacetGrid at 0x22f08217848>df.head()
Loan_ID Gender Married Dependents Education Self_Employed \
0 LP001002 Male No 0 Graduate No
1 LP001003 Male Yes 1 Graduate No
2 LP001005 Male Yes 0 Graduate Yes
3 LP001006 Male Yes 0 Not Graduate No
4 LP001008 Male No 0 Graduate No
ApplicantIncome CoapplicantIncome LoanAmount Loan_Amount_Term \
0 5849 0.0 146.412162 360.0
1 4583 1508.0 128.000000 360.0
2 3000 0.0 66.000000 360.0
3 2583 2358.0 120.000000 360.0
4 6000 0.0 141.000000 360.0
```

Credit_History Property_Area Loan_Status Total_income
ApplicantIncomeLog \
0 1.0 Urban Y 5849.0
8.674026
1 1.0 Rural N 6091.0 8.430109
2 1.0 Urban Y 3000.0
8.006368
3 1.0 Urban Y 4941.0
7.856707
4 1.0 Urban Y 6000.0
8.699515
CoapplicantIncomeLog LoanAmountLog Loan_Amount_Term_Log
Total_Income_Log
0 -inf 4.986426 5.886104
8.674026
1 7.318540 4.852030 5.886104
8.714568
2 -inf 4.189655 5.886104
8.006368
3 7.765569 4.787492 5.886104
8.505323
4 -inf 4.948760 5.886104
8.699515

```
cols = ['ApplicantIncome', 'CoapplicantIncome', "LoanAmount",
"Loan_Amount_Term", "Total_income", 'Loan_ID', 'CoapplicantIncomeLog']
df = df.drop(columns=cols, axis=1)
df.head()
```

Gender Married Dependents Education Self_Employed
Credit_History \
0 Male No 0 Graduate No
1.0
1 Male Yes 1 Graduate No
1.0
2 Male Yes 0 Graduate Yes
1.0
3 Male Yes 0 Not Graduate No
1.0
4 Male No 0 Graduate No
1.0
Property_Area Loan_Status ApplicantIncomeLog LoanAmountLog \
0 Urban Y 8.674026 4.986426
1 Rural N 8.430109 4.852030

2 Urban Y 8.006368 4.189655
3 Urban Y 7.856707 4.787492
4 Urban Y 8.699515 4.948760
Loan_Amount_Term_Log Total_Income_Log
0 5.886104 8.674026
1 5.886104 8.714568 2 5.886104 8.006368
3 5.886104 8.505323
4 5.886104 8.699515
df.Loan_Status.value_counts()
Y 422
N 192
Name: Loan_Status, dtype: int64
df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 12 columns):
Column Non-Null Count Dtype
--- ------ -------------- -----
0 Gender 614 non-null category
1 Married 614 non-null object
2 Dependents 614 non-null object
3 Education 614 non-null object
4 Self_Employed 614 non-null object
5 Credit_History 614 non-null float64
6 Property_Area 614 non-null object
7 Loan_Status 614 non-null object
8 ApplicantIncomeLog 614 non-null float64
9 LoanAmountLog 614 non-null float64
10 Loan_Amount_Term_Log 614 non-null float64
11 Total_Income_Log 614 non-null float64
dtypes: category(1), float64(5), object(6)
memory usage: 53.6+ KB
df.Education.value_counts()
Graduate 480
Not Graduate 134
Name: Education, dtype: int64
handling categorical data
df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 12 columns):
Column Non-Null Count Dtype

--- ------ -------------- -----
0 Gender 614 non-null category
1 Married 614 non-null object
2 Dependents 614 non-null object
3 Education 614 non-null object 4 Self_Employed 614 non-null object
5 Credit_History 614 non-null float64
6 Property_Area 614 non-null object
7 Loan_Status 614 non-null object
8 ApplicantIncomeLog 614 non-null float64
9 LoanAmountLog 614 non-null float64
10 Loan_Amount_Term_Log 614 non-null float64
11 Total_Income_Log 614 non-null float64
dtypes: category(1), float64(5), object(6)
memory usage: 53.6+ KB
df.head()
Gender Married Dependents Education Self_Employed
Credit_History \
0 Male No 0 Graduate No
1.0
1 Male Yes 1 Graduate No
1.0
2 Male Yes 0 Graduate Yes
1.0
3 Male Yes 0 Not Graduate No
1.0
4 Male No 0 Graduate No
1.0
Property_Area Loan_Status ApplicantIncomeLog LoanAmountLog \
0 Urban Y 8.674026 4.986426
1 Rural N 8.430109 4.852030
2 Urban Y 8.006368 4.189655
3 Urban Y 7.856707 4.787492
4 Urban Y 8.699515 4.948760
Loan_Amount_Term_Log Total_Income_Log
0 5.886104 8.674026
1 5.886104 8.714568
2 5.886104 8.006368
3 5.886104 8.505323
4 5.886104 8.699515
d1 = pd.get_dummies(df['Gender'], drop_first= True)
d2 = pd.get_dummies(df['Married'], drop_first= True)
d3 = pd.get_dummies(df['Dependents'], drop_first= True)

```
d4 = pd.get_dummies(df['Education'], drop_first= True)
d5 = pd.get_dummies(df['Self_Employed'], drop_first= True)
d6 = pd.get_dummies(df['Property_Area'], drop_first= True)df1 =
pd.concat([df, d1, d2, d3, d4, d5, d6], axis = 1)
df=df1
cols = ['Gender', 'Married', "Dependents", "Education",
"Self_Employed", 'Property_Area']
df = df.drop(columns=cols, axis=1)
# cols =
['Gender','Married',"Education",'Self_Employed',"Property_Area","Loan_
Status","Dependents"]
# for col in cols:
# df[col] = pd.get_dummies(df[col], drop_first= True)
df.head()
Credit_History Loan_Status ApplicantIncomeLog LoanAmountLog \
0 1.0 Y 8.674026 4.986426
1 1.0 N 8.430109 4.852030
2 1.0 Y 8.006368 4.189655
3 1.0 Y 7.856707 4.787492
4 1.0 Y 8.699515 4.948760
Loan_Amount_Term_Log Total_Income_Log Male Yes 1 2 3+ Not
Graduate \
0 5.886104 8.674026 1 0 0 0 0
0
1 5.886104 8.714568 1 1 1 0 0
0
2 5.886104 8.006368 1 1 0 0 0
0
3 5.886104 8.505323 1 1 0 0 0
1
4 5.886104 8.699515 1 0 0 0 0
0
Yes Semiurban Urban
0 0 0 1
1 0 0 0
2 1 0 1
3 0 0 1
4 0 0 1
df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 15 columns):
```

Column Non-Null Count Dtype
--- ------ ------------- ----- 0 Credit_History 614 non-null float64
1 Loan_Status 614 non-null object
2 ApplicantIncomeLog 614 non-null float64
3 LoanAmountLog 614 non-null float64
4 Loan_Amount_Term_Log 614 non-null float64
5 Total_Income_Log 614 non-null float64
6 Male 614 non-null uint8
7 Yes 614 non-null uint8
8 1 614 non-null uint8
9 2 614 non-null uint8
10 3+ 614 non-null uint8
11 Not Graduate 614 non-null uint8
12 Yes 614 non-null uint8
13 Semiurban 614 non-null uint8
14 Urban 614 non-null uint8
dtypes: float64(5), object(1), uint8(9)
memory usage: 34.3+ KB
df.describe()
Credit_History ApplicantIncomeLog LoanAmountLog \
count 614.000000 614.000000 614.000000
mean 0.842199 8.341213 4.862066
std 0.349681 0.645263 0.496575
min 0.000000 5.010635 2.197225
25% 1.000000 7.964677 4.607658
50% 1.000000 8.246040 4.859812
75% 1.000000 8.664750 5.104426
max 1.000000 11.302204 6.551080
Loan_Amount_Term_Log Total_Income_Log Male Yes
\
count 614.000000 614.000000 614.000000 614.000000
mean 5.802065 8.669414 0.817590 0.653094
std 0.312482 0.545102 0.386497 0.476373
min 2.484907 7.273786 0.000000 0.000000
25% 5.886104 8.334712 1.000000 0.000000
50% 5.886104 8.597205 1.000000 1.000000
75% 5.886104 8.925549 1.000000 1.000000
max 6.173786 11.302204 1.000000 1.000000
1 2 3+ Not Graduate Yes \
count 614.000000 614.000000 614.000000 614.000000 614.000000 mean
0.166124 0.164495 0.083062 0.218241 0.133550
std 0.372495 0.371027 0.276201 0.413389 0.340446

```
min 0.000000 0.000000 0.000000 0.000000 0.000000
25% 0.000000 0.000000 0.000000 0.000000 0.000000
50% 0.000000 0.000000 0.000000 0.000000 0.000000
75% 0.000000 0.000000 0.000000 0.000000 0.000000
max 1.000000 1.000000 1.000000 1.000000 1.000000
Semiurban Urban
count 614.000000 614.000000
mean 0.379479 0.328990
std 0.485653 0.470229
min 0.000000 0.000000
25% 0.000000 0.000000
50% 0.000000 0.000000
75% 1.000000 1.000000
max 1.000000 1.000000
# test datasets
test = pd.read_csv("test.csv")
# filling numerical missing data
test['LoanAmount']=test['LoanAmount'].fillna(test['LoanAmount'].mean()
)
test['Loan_Amount_Term']=test['Loan_Amount_Term'].fillna(test['Loan_Am
ount_Term'].mean())
test['Credit_History']=test['Credit_History'].fillna(test['Credit_Hist
ory'].mean())
# filling categorical missing data
test['Gender']=test['Gender'].fillna(test['Gender'].mode()[0])
test['Married']=test['Married'].fillna(test['Married'].mode()[0])
test['Dependents']=test['Dependents'].fillna(test['Dependents'].mode()
[0])
test['Self_Employed']=test['Self_Employed'].fillna(test['Self_Employed
'].mode()[0])
test['Total_income'] = test['ApplicantIncome']
+test['CoapplicantIncome']
# apply log transformation to the attribute
test['ApplicantIncomeLog'] = np.log(test['ApplicantIncome'])
test['CoapplicantIncomeLog'] = np.log(test['CoapplicantIncome'])
test['LoanAmountLog'] = np.log(test['LoanAmount'])
test['Loan_Amount_Term_Log'] = np.log(test['Loan_Amount_Term'])
test['Total_Income_Log'] = np.log(test['Total_income'])cols =
['ApplicantIncome', 'CoapplicantIncome', "LoanAmount",
"Loan_Amount_Term", "Total_income", 'Loan_ID', 'CoapplicantIncomeLog']
test = test.drop(columns=cols, axis=1)
t1 = pd.get_dummies(test['Gender'], drop_first= True)
```

t2 = pd.get_dummies(test['Married'], drop_first= True)
t3 = pd.get_dummies(test['Dependents'], drop_first= True)
t4 = pd.get_dummies(test['Education'], drop_first= True)
t5 = pd.get_dummies(test['Self_Employed'], drop_first= True)
t6 = pd.get_dummies(test['Property_Area'], drop_first= True)
df1 = pd.concat([test, t1, t2, t3, t4, t5, t6], axis = 1)
test=df1
cols = ['Gender', 'Married', "Dependents", "Education",
"Self_Employed", 'Property_Area']
test = test.drop(columns=cols, axis=1)
C:\Users\No_Name\AppData\Local\Programs\Python\Python37\lib\site
packages\pandas\core\arraylike.py:364: RuntimeWarning: divide by zero
encountered in log
result = getattr(ufunc, method)(*inputs, **kwargs)
test.head()
Credit_History ApplicantIncomeLog LoanAmountLog
Loan_Amount_Term_Log \
0 1.000000 8.651724 4.700480
5.886104
1 1.000000 8.031385 4.836282
5.886104
2 1.000000 8.517193 5.337538
5.886104
3 0.825444 7.757906 4.605170
5.886104
4 1.000000 8.094378 4.356709
5.886104
Total_Income_Log Male Yes 1 2 3+ Not Graduate Yes Semiurban
Urban
0 8.651724 1 1 0 0 0 0 0 0
1
1 8.428581 1 1 1 0 0 0 0 0
1
2 8.824678 1 1 0 1 0 0 0 0
1
3 8.494129 1 1 0 1 0 0 0 0
1 4 8.094378 1 0 0 0 0 1 0 0
1
split datasets
df.head()
Credit_History Loan_Status ApplicantIncomeLog LoanAmountLog \
0 1.0 Y 8.674026 4.986426

1 1.0 N 8.430109 4.852030
2 1.0 Y 8.006368 4.189655
3 1.0 Y 7.856707 4.787492
4 1.0 Y 8.699515 4.948760
Loan_Amount_Term_Log Total_Income_Log Male Yes 1 2 3+ Not
Graduate \
0 5.886104 8.674026 1 0 0 0 0
0
1 5.886104 8.714568 1 1 1 0 0
0
2 5.886104 8.006368 1 1 0 0 0
0
3 5.886104 8.505323 1 1 0 0 0
1
4 5.886104 8.699515 1 0 0 0 0
0
Yes Semiurban Urban
0 0 0 1
1 0 0 0
2 1 0 1
3 0 0 1
4 0 0 1

```
# specify input and output attributes
x = df.drop(columns=['Loan_Status'], axis=1)
y = df['Loan_Status']
x
```

Credit_History ApplicantIncomeLog LoanAmountLog
Loan_Amount_Term_Log \
0 1.0 8.674026 4.986426
5.886104
1 1.0 8.430109 4.852030
5.886104
2 1.0 8.006368 4.189655
5.886104
3 1.0 7.856707 4.787492
5.886104 4 1.0 8.699515 4.948760
5.886104
.. ..
...
609 1.0 7.972466 4.262680
5.886104
610 1.0 8.320205 3.688879

5.192957
611 1.0 8.996157 5.533389
5.886104
612 1.0 8.933664 5.231109
5.886104
613 0.0 8.430109 4.890349
5.886104
Total_Income_Log Male Yes 1 2 3+ Not Graduate Yes
Semiurban \
0 8.674026 1 0 0 0 0 0 0
0
1 8.714568 1 1 1 0 0 0 0
0
2 8.006368 1 1 0 0 0 0 1
0
3 8.505323 1 1 0 0 0 1 0
0
4 8.699515 1 0 0 0 0 0 0
0
.. ..
..
609 7.972466 0 0 0 0 0 0 0
0
610 8.320205 1 1 0 0 1 0 0
0
611 9.025456 1 1 1 0 0 0 0
0
612 8.933664 1 1 0 1 0 0 0
0
613 8.430109 0 0 0 0 0 0 1
1
Urban
0 1
1 0
2 1
3 1
4 1
.. ...
609 0
610 0 611 1
612 1
613 0

[614 rows x 14 columns]
y
0 Y
1 N
2 Y
3 Y
4 Y
..
609 Y
610 Y
611 Y
612 Y
613 N
Name: Loan_Status, Length: 614, dtype: object

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size=0.25, random_state=42)
x_train.head()
```

Credit_History ApplicantIncomeLog LoanAmountLog
Loan_Amount_Term_Log \
92 1.0 8.093462 4.394449
5.886104
304 1.0 8.294050 4.941642
5.886104
68 1.0 8.867850 4.828314
4.094345
15 1.0 8.507143 4.828314
5.886104
211 0.0 8.140316 4.852030
5.886104
Total_Income_Log Male Yes 1 2 3+ Not Graduate Yes
Semiurban \
92 8.535622 1 1 0 1 0 1 0
0
304 8.779557 1 0 0 0 0 0 0
0
68 8.867850 1 1 0 0 1 1 1
0
15 8.507143 1 0 0 0 0 0 0
0
211 8.451053 1 1 0 0 1 0 0 1
Urban

```
92 1
304 0
68 1
15 1
211 0
y_test.head()
350 Y
377 Y
163 Y
609 Y
132 Y
Name: Loan_Status, dtype: object
# model training
# randomforest classifier
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier()
model.fit(x_train, y_train)
RandomForestClassifier()
print("Accuracy is", model.score(x_test, y_test)*100)
Accuracy is 79.22077922077922
# decision tree classifier
from sklearn.tree import DecisionTreeClassifier
model2 = DecisionTreeClassifier()
model2.fit(x_train, y_train)
print("Accuracy is", model2.score(x_test, y_test)*100)
Accuracy is 67.53246753246754
# logistic regression
from sklearn.linear_model import LogisticRegression
model3 = LogisticRegression()
model3.fit(x_train, y_train)
print("Accuracy is", model3.score(x_test, y_test)*100)
Accuracy is 77.27272727272727
C:\Users\No_Name\AppData\Local\Programs\Python\Python37\lib\site
packages\sklearn\linear_model\_logistic.py:818: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.Increase the number of
iterations (max_iter) or scale the data as
shown in:
https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear_model.html#logistic
regression
```

```
extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
# confusion matrics
# random forest classifier
from sklearn.metrics import confusion_matrix
y_pred = model.predict(x_test)
cm = confusion_matrix(y_test, y_pred)
cm
array([[25, 29],
[ 3, 97]], dtype=int64)
# model save
import pickle
file=open("model.pkl", 'wb')
pickle.dump(model, file)
```

```
from flask import Flask, escape, request, render_template

import pickle

import numpy as np


app = Flask(__name__)

model = pickle.load(open('model.pkl', 'rb'))



@app.route('/')

def home():

    return render_template("index.html")



@app.route('/predict', methods=['GET', 'POST'])

def predict():

    if request.method == 'POST':

        gender = request.form['gender']
```

```
married = request.form['married']
dependents = request.form['dependents']
education = request.form['education']
employed = request.form['employed']
credit = float(request.form['credit'])

area = request.form['area']
ApplicantIncome = float(request.form['ApplicantIncome'])
CoapplicantIncome = float(request.form['CoapplicantIncome'])
LoanAmount = float(request.form['LoanAmount'])
Loan_Amount_Term = float(request.form['Loan_Amount_Term'])

# gender
if (gender == "Male"):
   male = 1
else:
   male = 0

# married
if (married == "Yes"):
   married_yes = 1
else:
   married_yes = 0

# dependents
if (dependents == '1'):
```

```
    dependents_1 = 1
    dependents_2 = 0
    dependents_3 = 0
elif (dependents == '2'):
    dependents_1 = 0
    dependents_2 = 1
    dependents_3 = 0
elif (dependents == "3+"):
    dependents_1 = 0
    dependents_2 = 0
    dependents_3 = 1
else:
    dependents_1 = 0
    dependents_2 = 0
    dependents_3 = 0

    # education
if (education == "Not Graduate"):
    not_graduate = 1
else:
    not_graduate = 0

# employed
if (employed == "Yes"):
    employed_yes = 1
else:
```

```python
        employed_yes = 0

    # property area

    if (area == "Semiurban"):
        semiurban = 1
        urban = 0
    elif (area == "Urban"):
        semiurban = 0
        urban = 1
    else:
        semiurban = 0
        urban = 0


    ApplicantIncomelog = np.log(ApplicantIncome)
    totalincomelog = np.log(ApplicantIncome + CoapplicantIncome)
    LoanAmountlog = np.log(LoanAmount)
    Loan_Amount_Termlog = np.log(Loan_Amount_Term)

    prediction = model.predict([[credit, ApplicantIncomelog, LoanAmountlog, Loan_Amount_Termlog, totalincomelog,
                                  male, married_yes, dependents_1, dependents_2, dependents_3, not_graduate,
                                  employed_yes, semiurban, urban]])

    # print(prediction)
```

```
        if (prediction == "N"):
            prediction = "No"
        else:
            prediction = "Yes"

        return render_template("prediction.html", prediction_text="loan status is
{}".format(prediction))


    else:
        return render_template("prediction.html")


if __name__ == "__main__":
    app.run(debug=True)
```