clriqzvhu

January 23, 2025

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

```
[ ]: data_train=pd.read_csv('train_LoanPrediction.csv')
```

```
[ ]: data_train.head(3)
```

```
[ ]:     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

        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

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

```
[ ]: data_train.shape #size of data
```

```
[ ]: (614, 13)
```

```
[ ]: data_train.columns
```

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

```
[ ]: #Checking missing values
     print(data_train.isnull().sum())
     #we cannot not address null values
```

```
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
```

[ ]:
```
#for example checking the weight of married and unmarried and genralizing and␣
↪filling the remainng data
data_train['Gender'].value_counts()
```

[ ]:
```
Gender
Male      489
Female    112
Name: count, dtype: int64
```

[ ]:
```
gender= data_train['Gender'].value_counts()
print("Male ratio", gender[0]/gender.values)
```

```
Male ratio [1.         4.36607143]
```

```
<ipython-input-21-b71403003414>:2: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  print("Male ratio", gender[0]/gender.values)
```

[ ]:
```
#function to fill the null values to female based on the previous ratio
#i.e 1 female for 4 male
data_train['Gender'].fillna('Male',inplace=True,limit=10)
data_train['Gender'].fillna('Female',inplace=True,limit=3)
```

[ ]:
```
data_train.isnull().sum() #after filling null values in gender is 0
```

[ ]:
```
Loan_ID               0
Gender                0
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
```

[ ]:
```
data_train['Married'].value_counts()
```

[ ]:
```
Married
Yes    398
No     213
Name: count, dtype: int64
```

[ ]:
```
marry= data_train['Married'].value_counts()
print("Married ratio", marry[0]/marry.values)
```

```
Married ratio [1.         1.8685446]

<ipython-input-27-976c20583b53>:2: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  print("Married ratio", marry[0]/marry.values)
```

[ ]:
```
marry.values #is an array, in array we have individual operation
```

[ ]:
```
array([398, 213])
```

[ ]:
```
data_train['Married'].fillna('Yes',inplace=True,limit=2)
data_train['Married'].fillna('No',inplace=True,limit=1)
```

```
<ipython-input-31-9c93a0a96606>:1: FutureWarning: A value is trying to be set on
a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  data_train['Married'].fillna('Yes',inplace=True,limit=2)
```

[ ]:
```
data_train['Married'].value_counts()
data_train['Married'].isnull().sum()
```

[ ]:
```
0
```

[ ]:
```
data_train['Dependents'].value_counts()
```

[ ]:
```
Dependents
0     345
1     102
2     101
3+     51
Name: count, dtype: int64
```

[ ]:
```
depend= data_train['Dependents'].value_counts()
print("dep, ratio ",depend[0]/depend.values)
```

```
dep, ratio  [1.         3.38235294 3.41584158 6.76470588]

<ipython-input-35-51e85c0a834b>:2: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  print("dep, ratio ",depend[0]/depend.values)
```

[ ]:
```
data_train['Dependents'].fillna(0,inplace=True,limit=2)
data_train['Dependents'].fillna(1,inplace=True,limit=3)
data_train['Dependents'].fillna(2,inplace=True,limit=3)
data_train['Dependents'].fillna(3,inplace=True,limit=7)
```

[ ]:
```
data_train['Self_Employed'].value_counts()
```

[ ]:
```
Self_Employed
No     500
Yes     82
Name: count, dtype: int64
```

[ ]:
```
not_emp=data_train['Self_Employed'].value_counts()
print("The ratio of not employed to employed ",not_emp[0]/not_emp.values)
```

```
The ratio of not employed to employed  [1.         6.09756098]

<ipython-input-40-f9f003807c72>:2: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  print("The ratio of not employed to employed ",not_emp[0]/not_emp.values)
```

```
[ ]: data_train['Self_Employed'].fillna('No',inplace=True,limit=27)
     data_train['Self_Employed'].fillna('Yes',inplace=True,limit=5)
```

```
<ipython-input-41-dae4be9ee51b>:1: FutureWarning: A value is trying to be set on
a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  data_train['Self_Employed'].fillna('No',inplace=True,limit=27)
```

```
[ ]: data_train.isnull().sum() #after filling null values in gender is 0 after␣
     ↪operation
```

```
[ ]: Loan_ID               0
     Gender                0
     Married               0
     Dependents            0
     Education             0
     Self_Employed         0
     ApplicantIncome       0
     CoapplicantIncome     0
     LoanAmount           22
     Loan_Amount_Term     14
     Credit_History       50
     Property_Area         0
     Loan_Status           0
     dtype: int64
```

```
[ ]: data_train['LoanAmount'].value_counts()
```

```
[ ]: LoanAmount
     120.0    20
     110.0    17
     100.0    15
     160.0    12
     187.0    12
              ..
     240.0     1
     214.0     1
     59.0      1
```

```
166.0     1
253.0     1
Name: count, Length: 203, dtype: int64
```

[ ]:
```
mean_amount=data_train['LoanAmount'].mean()
```

[ ]:
```
data_train['LoanAmount'].fillna(mean_amount,inplace=True,limit=22)
```

```
<ipython-input-47-0c1f74556d47>:1: FutureWarning: A value is trying to be set on
a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  data_train['LoanAmount'].fillna(mean_amount,inplace=True,limit=22)
```

[ ]:
```
data_train['Loan_Amount_Term'].value_counts()
```

[ ]:
```
Loan_Amount_Term
360.0    512
180.0     44
480.0     15
300.0     13
240.0      4
84.0       4
120.0      3
60.0       2
36.0       2
12.0       1
Name: count, dtype: int64
```

[ ]:
```
term_mean=data_train['Loan_Amount_Term'].mean()
term_mean
```

[ ]:
```
342.0
```

[ ]:
```
data_train['Loan_Amount_Term'].describe()
```

[ ]:
```
count    600.00000
mean     342.00000
std       65.12041
```

```
min       12.00000
25%      360.00000
50%      360.00000
75%      360.00000
max      480.00000
Name: Loan_Amount_Term, dtype: float64
```

[ ]:
```
data_train['Loan_Amount_Term'].fillna(term_mean,inplace=True,limit=14)
```

```
<ipython-input-57-212f5c526936>:1: FutureWarning: A value is trying to be set on
a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  data_train['Loan_Amount_Term'].fillna(term_mean,inplace=True,limit=14)
```

[ ]:
```
data_train['Credit_History'].value_counts()
```

[ ]:
```
Credit_History
1.0    475
0.0     89
Name: count, dtype: int64
```

[ ]:
```
credit=data_train['Credit_History'].value_counts()
print("Ratio ",credit[0]/credit.values)
```

```
Ratio  [0.18736842 1.        ]
```

[ ]:
```
data_train['Credit_History'].fillna(1.0,inplace=True,limit=40)
data_train['Credit_History'].fillna(0.0,inplace=True,limit=10)
```

```
<ipython-input-62-c00a132a4fa3>:1: FutureWarning: A value is trying to be set on
a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.
```

```
  data_train['Credit_History'].fillna(1.0,inplace=True,limit=40)
<ipython-input-62-c00a132a4fa3>:2: FutureWarning: A value is trying to be set on
a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  data_train['Credit_History'].fillna(0.0,inplace=True,limit=10)
```

[ ]:

```
data_train.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
```