Loan Prediction Dataset ML Project

In [1]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```python
df=pd.read_csv('loan_data_set.csv')
df.head()
```

Out[2]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
0	LP001002	Male	No	0	Graduate	No	5849	0.0	NaN	360.0	1.0	Urban	Y
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.0	1.0	Rural	N
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	360.0	1.0	Urban	Y
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0	1.0	Urban	Y
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	360.0	1.0	Urban	Y

Preprocessing and Data Analysis

In [3]:
```python
df.info(verbose=True,null_counts=True)
```

```
<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
```

In [4]:
```python
df.duplicated().sum()
```

Out[4]:
```
0
```

In [5]:
```python
df.describe()
```

Out[5]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	614.000000	614.000000	592.000000	600.00000	564.000000
mean	5403.459283	1621.245798	146.412162	342.00000	0.842199
std	6109.041673	2926.248369	85.587325	65.12041	0.364878
min	150.000000	0.000000	9.000000	12.00000	0.000000
25%	2877.500000	0.000000	100.000000	360.00000	1.000000
50%	3812.500000	1188.500000	128.000000	360.00000	1.000000
75%	5795.000000	2297.250000	168.000000	360.00000	1.000000
max	81000.000000	41667.000000	700.000000	480.00000	1.000000

In [6]:
```python
df.isnull().sum()
```

Out[6]:
```
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
```

In [7]:
```python
df.drop(columns='Loan_ID',axis=1,inplace=True)
```

In [8]:
```python
df.columns
```

Out[8]:
```
Index(['Gender', 'Married', 'Dependents', 'Education', 'Self_Employed',
       'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History', 'Property_Area', 'Loan_Status'],
      dtype='object')
```

In [10]:
```python
df.Gender.value_counts()
```

Out[10]:
```
Male      489
Female    112
Name: Gender, dtype: int64
```

In [11]:
```python
print('Before filling null values','\n','#'*50)
print()
null_cols=['Gender', 'Married', 'Dependents', 'Self_Employed','LoanAmount',
           'Loan_Amount_Term', 'Credit_History']
for col in null_cols:
    print(col,'\n',df[col].value_counts(),'\n','-'*50)
    df[col].fillna(df[col].mode()[0],inplace=True)

print()
print('After filling null values','\n','#'*50)
print()
for col in null_cols:
    print(col,'\n',df[col].value_counts(),'\n','-'*50)
```

```
Before filling null values
 ##################################################

Gender
 Male      489
Female    112
Name: Gender, dtype: int64
 --------------------------------------------------
Married
 Yes    398
No     213
Name: Married, dtype: int64
 --------------------------------------------------
Dependents
 0     345
1     102
2     101
3+     51
Name: Dependents, dtype: int64
 --------------------------------------------------
Self_Employed
 No     500
Yes     82
Name: Self_Employed, dtype: int64
 --------------------------------------------------
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: LoanAmount, Length: 203, dtype: int64
 --------------------------------------------------
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: Loan_Amount_Term, dtype: int64
 --------------------------------------------------
Credit_History
 1.0    475
0.0     89
Name: Credit_History, dtype: int64
 --------------------------------------------------

After filling null values
 ##################################################

Gender
 Male      502
Female    112
Name: Gender, dtype: int64
 --------------------------------------------------
Married
 Yes    401
No     213
Name: Married, dtype: int64
 --------------------------------------------------
Dependents
 0     360
1     102
2     101
3+     51
Name: Dependents, dtype: int64
 --------------------------------------------------
Self_Employed
 No     532
Yes     82
Name: Self_Employed, dtype: int64
 --------------------------------------------------
LoanAmount
 120.0    42
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: LoanAmount, Length: 203, dtype: int64
 --------------------------------------------------
Loan_Amount_Term
 360.0    526
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: Loan_Amount_Term, dtype: int64
 --------------------------------------------------
Credit_History
 1.0    525
0.0     89
Name: Credit_History, dtype: int64
 --------------------------------------------------
```

In [12]:
```python
df.isnull().sum()
```

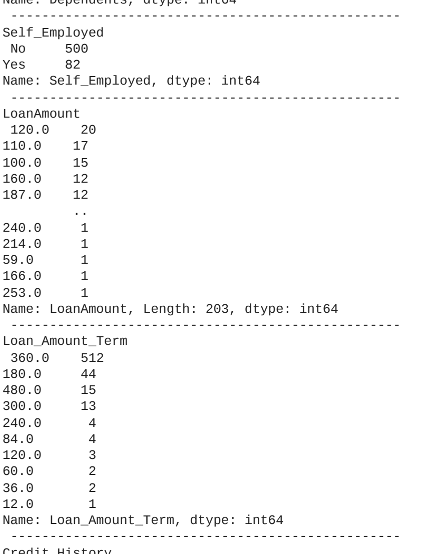
Out[12]:
```
Gender               0
Married              0
Dependents           0
Education            0
Self_Employed        0
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           0
Loan_Amount_Term     0
Credit_History       0
Property_Area        0
Loan_Status          0
dtype: int64
```

Data Visalization

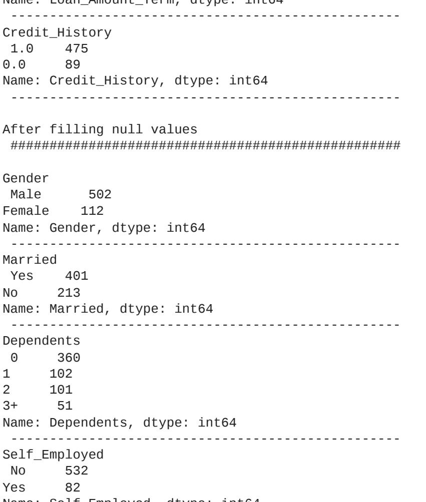
In [13]:
```python
num=df.select_dtypes('number').columns
cat=df.select_dtypes('object').columns
df_num=df[num]
df_cat=df[cat]
```

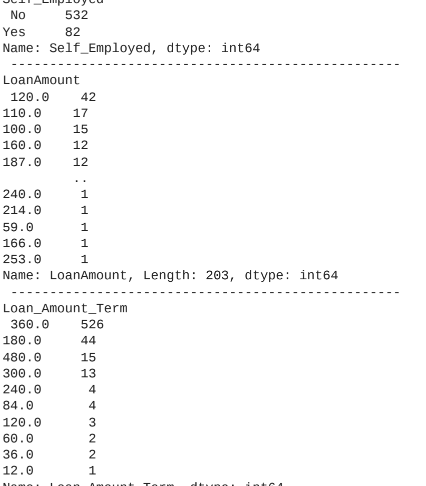
In [14]:
```python
print(df[cat[-1]].value_counts())
plt.figure(figsize=(4,3))
sns.set()
sns.countplot(x=df[cat[-1]]);
```

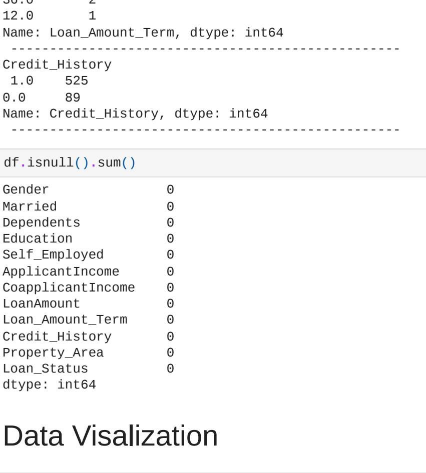
```
Y    422
N    192
Name: Loan_Status, dtype: int64
```

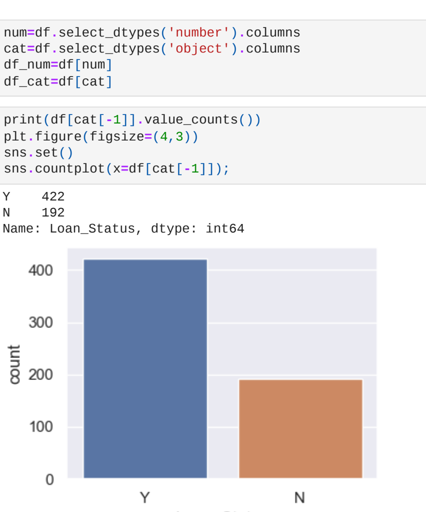


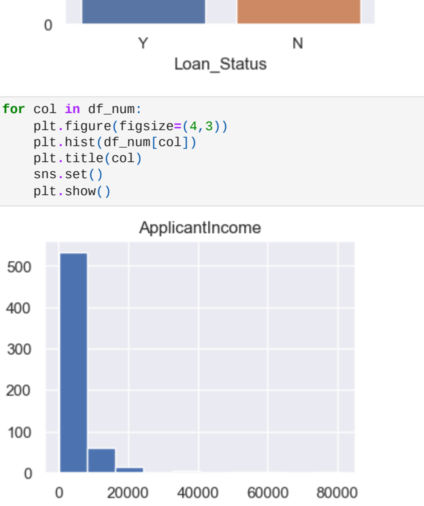
In [15]:
```python
for col in df_num:
    plt.figure(figsize=(4,3))
    plt.hist(df_num[col])
    plt.title(col)
    sns.set()
    plt.show()
```



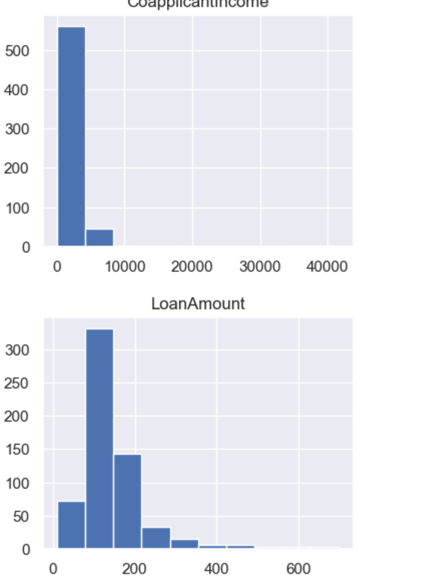


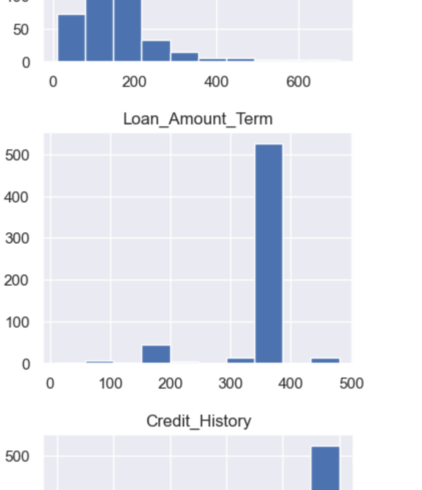


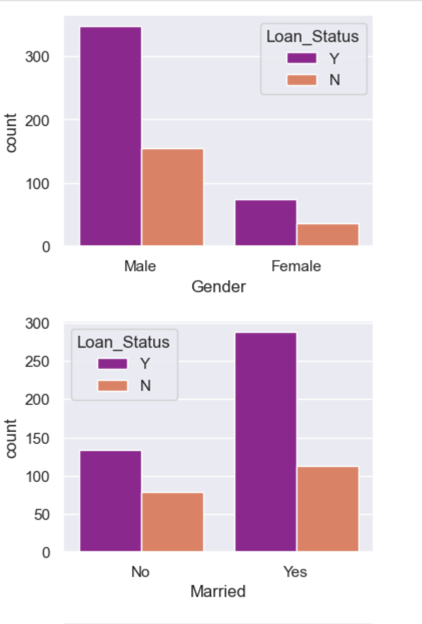


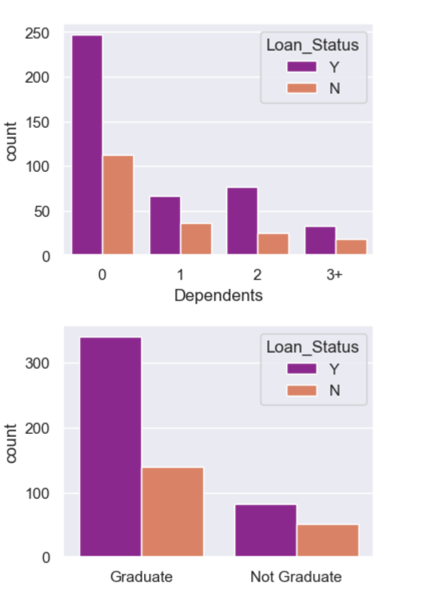


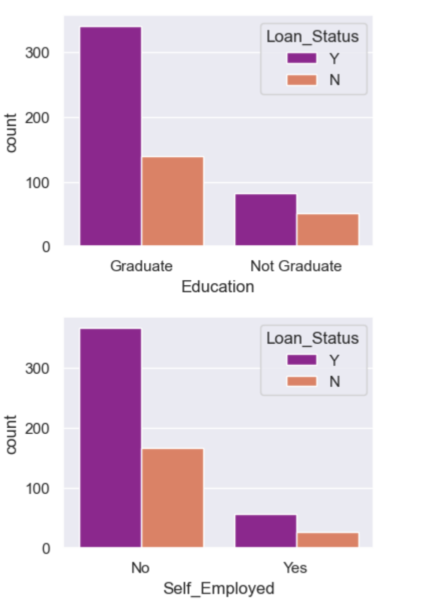
In [16]:
```python
for col in cat[:-1]:
    plt.figure(figsize=(4,3))
    sns.countplot(x=col,hue='Loan_Status',data=df,palette='plasma');
    sns.set()
```











In [17]:
```python
df_num.head()
```

Out[17]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
0	5849	0.0	120.0	360.0	1.0
1	4583	1508.0	128.0	360.0	1.0
2	3000	0.0	66.0	360.0	1.0
3	2583	2358.0	120.0	360.0	1.0
4	6000	0.0	141.0	360.0	1.0

In [18]:
```python
df_cat.head()
```

Out[18]:

	Gender	Married	Dependents	Education	Self_Employed	Property_Area	Loan_Status
0	Male	No	0	Graduate	No	Urban	Y
1	Male	Yes	1	Graduate	No	Rural	N
2	Male	Yes	0	Graduate	Yes	Urban	Y
3	Male	Yes	0	Not Graduate	No	Urban	Y
4	Male	No	0	Graduate	No	Urban	Y

In [19]:
```python
for i in df_cat:
    print(df_cat[i].value_counts(),'\n','-'*30)
```

```
Male      502
Female    112
Name: Gender, dtype: int64
 ------------------------------
Yes    401
No     213
Name: Married, dtype: int64
 ------------------------------
0     360
1     102
2     101
3+     51
Name: Dependents, dtype: int64
 ------------------------------
Graduate        480
Not Graduate    134
Name: Education, dtype: int64
 ------------------------------
No     532
Yes     82
Name: Self_Employed, dtype: int64
 ------------------------------
Semiurban    233
Urban        202
Rural        179
Name: Property_Area, dtype: int64
 ------------------------------
Y    422
N    192
Name: Loan_Status, dtype: int64
 ------------------------------
```

In [20]:
```python
to_numeric={'Male':1,'Female':0,'Yes':1,'No':0,'3+':3,'Graduate':1,'Not Graduate':0,'Urban':3,
            'Semiurban':2,'Rural':1,'Y':1,'N':0}
df_cat_num=df_cat.applymap(lambda x: to_numeric.get(x) if x in to_numeric else x)
for i in df_cat_num:
    print(df_cat_num[i].value_counts(),'\n','-'*30)
```

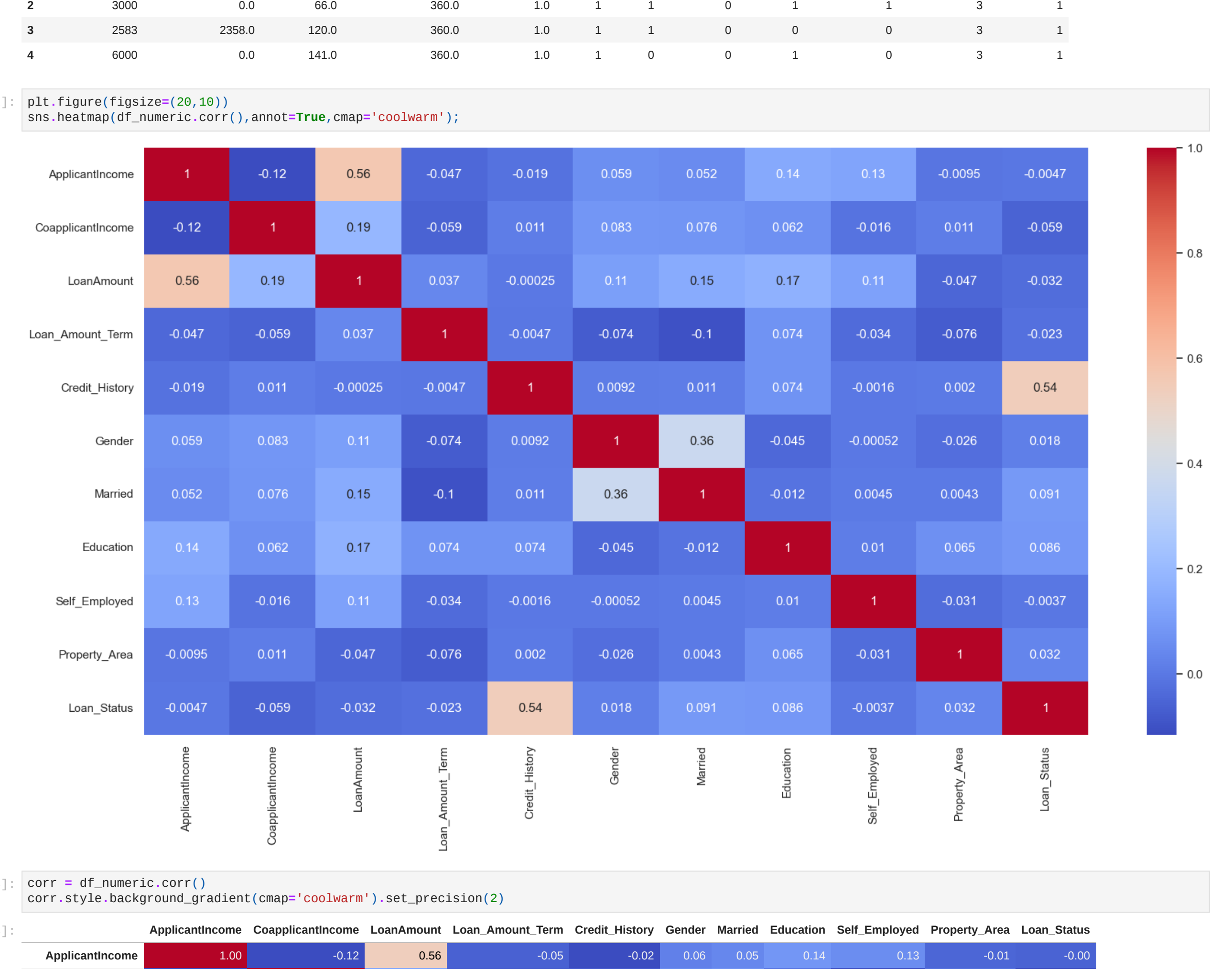
```
1    502
0    112
Name: Gender, dtype: int64
 ------------------------------
1    401
0    213
Name: Married, dtype: int64
 ------------------------------
0    360
1    102
2    101
3     51
Name: Dependents, dtype: int64
 ------------------------------
1    480
0    134
Name: Education, dtype: int64
 ------------------------------
0    532
1     82
Name: Self_Employed, dtype: int64
 ------------------------------
2    233
3    202
1    179
Name: Property_Area, dtype: int64
 ------------------------------
1    422
0    192
Name: Loan_Status, dtype: int64
 ------------------------------
```

In [21]:
```python
df_numeric=pd.concat([df_num,df_cat_num],axis=1)
df_numeric.head()
```

Out[21]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Gender	Married	Dependents	Education	Self_Employed	Property_Area	Loan_Status
0	5849	0.0	120.0	360.0	1.0	1	0	0	1	0	3	1
1	4583	1508.0	128.0	360.0	1.0	1	1	1	1	0	1	0
2	3000	0.0	66.0	360.0	1.0	1	1	0	1	1	3	1
3	2583	2358.0	120.0	360.0	1.0	1	1	0	0	0	3	1
4	6000	0.0	141.0	360.0	1.0	1	0	0	1	0	3	1

In [22]:
```python
plt.figure(figsize=(20,10))
sns.heatmap(df_numeric.corr(),annot=True,cmap='coolwarm');
```



In [23]:
```python
corr = df_numeric.corr()
corr.style.background_gradient(cmap='coolwarm').set_precision(2)
```

Out[23]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Gender	Married	Education	Self_Employed	Property_Area	Loan_Status
ApplicantIncome	1.00	-0.12	0.56	-0.05	-0.02	0.06	0.05	0.14	0.13	-0.01	-0.00
CoapplicantIncome	-0.12	1.00	0.19	-0.06	0.01	0.08	0.08	0.06	-0.02	0.01	-0.06
LoanAmount	0.56	0.19	1.00	0.04	-0.00	0.11	0.15	0.17	0.11	-0.05	-0.03
Loan_Amount_Term	-0.05	-0.06	0.04	1.00	-0.00	-0.07	-0.10	0.07	-0.03	-0.08	-0.02
Credit_History	-0.02	0.01	-0.00	-0.00	1.00	0.01	0.01	0.07	-0.00	0.00	0.54
Gender	0.06	0.08	0.11	-0.07	0.01	1.00	0.36	-0.05	-0.00	-0.03	0.02
Married	0.05	0.08	0.15	-0.10	0.01	0.36	1.00	-0.01	0.00	0.00	0.09
Education	0.14	0.06	0.17	0.07	0.07	-0.05	-0.01	1.00	0.01	0.07	0.09
Self_Employed	0.13	-0.02	0.11	-0.03	-0.00	-0.00	0.00	0.01	1.00	-0.03	-0.00
Property_Area	-0.01	0.01	-0.05	-0.08	0.00	-0.03	0.00	0.07	-0.03	1.00	0.03
Loan_Status	-0.00	-0.06	-0.03	-0.02	0.54	0.02	0.09	0.09	-0.00	0.03	1.00

Machine learning models

In [52]:
```python
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
DT=DecisionTreeClassifier()
from sklearn.ensemble import RandomForestClassifier
RF=RandomForestClassifier()
from xgboost import XGBClassifier
XGB=XGBClassifier()
from sklearn.linear_model import LogisticRegression
LoR=LogisticRegression()
```

In [26]:
```python
df_numeric.head()
```

Out[26]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Gender	Married	Dependents	Education	Self_Employed	Property_Area	Loan_Status
0	5849	0.0	120.0	360.0	1.0	1	0	0	1	0	3	1
1	4583	1508.0	128.0	360.0	1.0	1	1	1	1	0	1	0
2	3000	0.0	66.0	360.0	1.0	1	1	0	1	1	3	1
3	2583	2358.0	120.0	360.0	1.0	1	1	0	0	0	3	1
4	6000	0.0	141.0	360.0	1.0	1	0	0	1	0	3	1

In [29]:
```python
Y=df_numeric['Loan_Status']
X=df_numeric.drop(columns='Loan_Status',axis=1)
```

In [33]:
```python
X_train, X_test, Y_train, Y_test=train_test_split(X,Y,test_size=0.3,random_state=100)
```

DecisionTreeClassifier

In [44]:
```python
DT.fit(X_train,Y_train)
Y_predict=DT.predict(X_test)
DT_score=accuracy_score(Y_predict,Y_test)
print(classification_report(Y_test,Y_predict))
print('Accuracy',round(DT_score*100,2),'%')
```

```
              precision    recall  f1-score   support

           0       0.53      0.54      0.54        63
           1       0.76      0.75      0.76       122

    accuracy                           0.68       185
   macro avg       0.65      0.65      0.65       185
weighted avg       0.68      0.68      0.68       185

Accuracy 68.11 %
```

In [45]:
```python
Decision_Tree=pd.DataFrame({'Y_test':Y_test,'Y_predict':Y_predict})
Decision_Tree.to_csv('Decision Tree Result')
```

RandomForestClassifier

In [48]:
```python
RF.fit(X_train,Y_train)
Y_predict=RF.predict(X_test)
RF_score=accuracy_score(Y_predict,Y_test)
print(classification_report(Y_test,Y_predict))
print('Accuracy',round(RF_score*100,2),'%')
```

```
              precision    recall  f1-score   support

           0       0.74      0.49      0.59        63
           1       0.78      0.91      0.84       122

    accuracy                           0.77       185
   macro avg       0.76      0.70      0.71       185
weighted avg       0.76      0.77      0.75       185

Accuracy 76.76 %
```

In [50]:
```python
Random_Forest=pd.DataFrame({'Y_test':Y_test,'Y_predict':Y_predict})
Random_Forest.to_csv('Random Forest Result')
```

LogisticRegression

In [67]:
```python
LoR.fit(X_train,Y_train)
Y_predict=LoR.predict(X_test)
LoR_score=accuracy_score(Y_predict,Y_test)
print(classification_report(Y_test,Y_predict))
print('Accuracy',round(LoR_score,2))
```

```
              precision    recall  f1-score   support

           0       0.82      0.43      0.56        63
           1       0.76      0.95      0.85       122

    accuracy                           0.77       185
   macro avg       0.79      0.69      0.70       185
weighted avg       0.78      0.77      0.75       185

Accuracy 0.77
```

In [68]:
```python
Logistic_Regression=pd.DataFrame({'Y_test':Y_test,'Y_predict':Y_predict})
Logistic_Regression.to_csv('Logistic Regression Result')
```

Conclusion

1. Approx 31.27 % customers are not submitted loan, it is'nt good figure for bank.
2. Ratio of Unmarried customers are more than ratio of married ones to not submitted loan.
3. Ratio of Not graduate customers are more than ratio of graduate ones to not submitted loan.
4. In rural area, defaulter ratio is highest.
5. Credit_History is a very important variable because of its high correlation with Loan_Status therefor showind high Dependancy for the latter.
6. The Logistic Regression algorithm is the most accurate: approximately 77%.

In [84]:
```python
score=[DT_score,RF_score,LoR_score]
Model=pd.DataFrame({'n_neighbours':['Decision Tree','Random Forest','Logistic Regression'],
                    'Score':score})
Model.sort_values(by='Score',ascending=False,ignore_index=True)
```

Out[84]:

	n_neighbours	Score
0	Logistic Regression	0.772973
1	Random Forest	0.767568
2	Decision Tree	0.681081