In [133]:
```
## Install the Libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')

%matplotlib inline
```

In [134]:
```
df = pd.read_csv(r'C:\Users\SATYENDRA PRAKASH\Downloads\loanprediction.csv')
df
```

Out[134]:

ender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
Male	No	0	Graduate	No	5849	0.0	NaN	360.0	1.0	Urban	Y
Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.0	1.0	Rural	N
Male	Yes	0	Graduate	Yes	3000	0.0	66.0	360.0	1.0	Urban	Y
Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0	1.0	Urban	Y
Male	No	0	Graduate	No	6000	0.0	141.0	360.0	1.0	Urban	Y
...	...	...	...	...	...	...	...	...	...	...	...
emale	No	0	Graduate	No	2900	0.0	71.0	360.0	1.0	Rural	Y
Male	Yes	3+	Graduate	No	4106	0.0	40.0	180.0	1.0	Rural	Y
Male	Yes	1	Graduate	No	8072	240.0	253.0	360.0	1.0	Urban	Y
Male	Yes	2	Graduate	No	7583	0.0	187.0	360.0	1.0	Urban	Y
emale	No	0	Graduate	Yes	4583	0.0	133.0	360.0	0.0	Semiurban	N

umns

In [135]:
```
df.head()
```

Out[135]:

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

In [136]:
```
df.tail()
```

Out[136]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_
609	LP002978	Female	No	0	Graduate	No	2900	0.0	71.0	360.0	1.0	
610	LP002979	Male	Yes	3+	Graduate	No	4106	0.0	40.0	180.0	1.0	
611	LP002983	Male	Yes	1	Graduate	No	8072	240.0	253.0	360.0	1.0	U
612	LP002984	Male	Yes	2	Graduate	No	7583	0.0	187.0	360.0	1.0	U
613	LP002990	Female	No	0	Graduate	Yes	4583	0.0	133.0	360.0	0.0	Semi

In [137]:
```
df.shape
```

Out[137]:
```
(614, 13)
```

In [138]:
```
df.columns
```

Out[138]:
```
Index(['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education',
       'Self_Employed', 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History', 'Property_Area', 'Loan_Status'],
      dtype='object')
```

In [139]:
```
df.info()
```

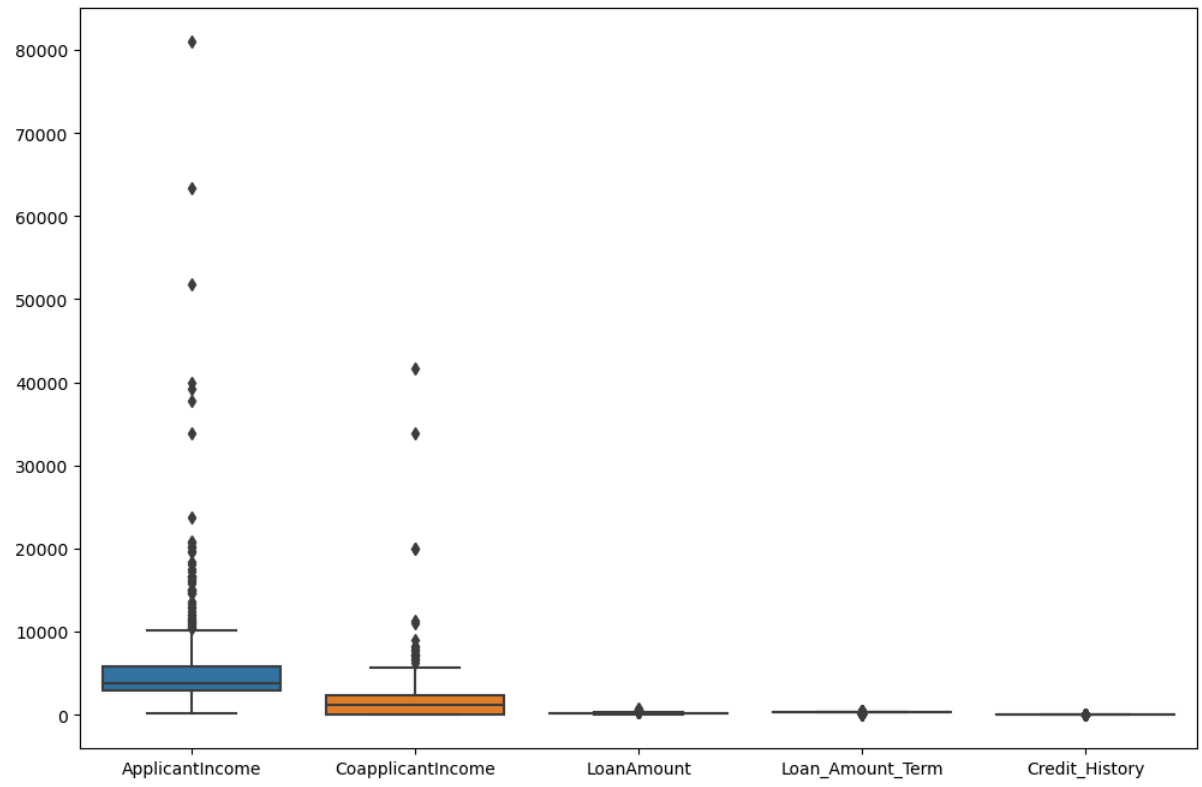
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Loan_ID            614 non-null    object 
 1   Gender             601 non-null    object 
 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
```

In [140]:
```
df.isnull().sum()
```

Out[140]:
```
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 [141]:
```
## Checking the outliers

plt.figure(figsize=(12,8))
sns.boxplot(data = df)
```

Out[141]: <Axes: >



In [142]:
```
## Fill the null values of numerical datatype
df['LoanAmount'] = df['LoanAmount'].fillna(df['LoanAmount'].median())
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())
```

In [143]:
```
## Fill the null values of object datatype
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])
```

In [144]:
```
df['Self_Employed'] = df['Self_Employed'].fillna(df['Self_Employed'].mode()[0])
```

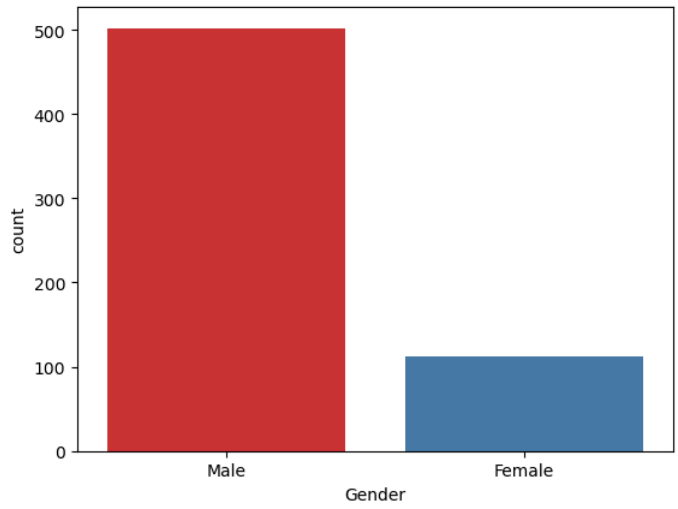
In [145]:
```
df.isnull().sum()
```

Out[145]:
```
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
```

In [146]:
```
print('Number of people who took loan by gender')
print(df['Gender'].value_counts())
sns.countplot(x='Gender',data = df, palette='Set1')
```

```
Number of people who took loan by gender
Gender
Male      502
Female    112
Name: count, dtype: int64
```

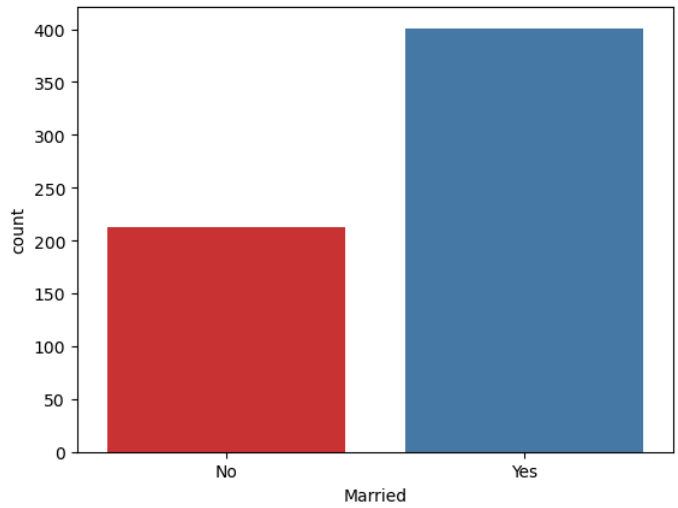
Out[146]:
```
<Axes: xlabel='Gender', ylabel='count'>
```



In [147]:
```
print('Number of people who took loan by Married')
print(df['Married'].value_counts())
sns.countplot(x='Married',data = df, palette='Set1')
```

```
Number of people who took loan by Married
Married
Yes    401
No     213
Name: count, dtype: int64
```

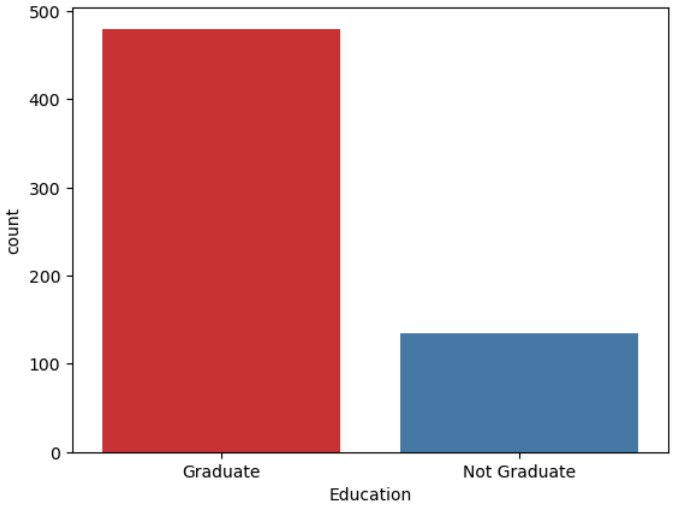
Out[147]:
```
<Axes: xlabel='Married', ylabel='count'>
```



In [148]:
```
print('Number of people who took loan by Education')
print(df['Education'].value_counts())
sns.countplot(x='Education',data = df, palette='Set1')
```

```
Number of people who took loan by Education
Education
Graduate        480
Not Graduate    134
Name: count, dtype: int64
```

Out[148]: <Axes: xlabel='Education', ylabel='count'>



In [149]:
```
corr = df.corr(numeric_only=True)
plt.figure(figsize=(10,8))
sns.heatmap(corr, annot = True, cmap = 'BuPu')
```

Out[149]: <Axes: >

In [150]:
```
corr = df.corr(numeric_only=True)
corr
```

Out[150]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
ApplicantIncome	1.000000	-0.116605	0.565181	-0.045242	-0.014477
CoapplicantIncome	-0.116605	1.000000	0.189218	-0.059675	-0.001665
LoanAmount	0.565181	0.189218	1.000000	0.039235	-0.007031
Loan_Amount_Term	-0.045242	-0.059675	0.039235	1.000000	0.001395
Credit_History	-0.014477	-0.001665	-0.007031	0.001395	1.000000

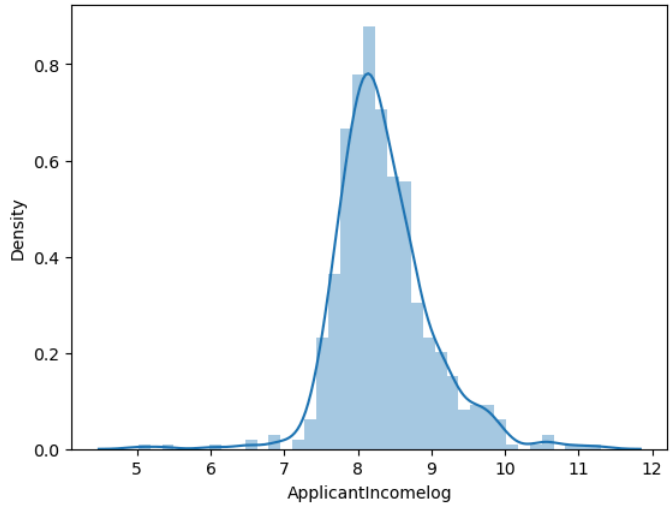
In [151]:
```
## Total Applicant Income

df['Total_Income'] = df['ApplicantIncome'] + df['CoapplicantIncome']
df.head()
```

Out[151]:

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

In [152]:
```
## Apply Log Transformation

df['ApplicantIncomelog'] = np.log(df['ApplicantIncome'] + 1)
sns.distplot(df['ApplicantIncomelog'])
```

Out[152]:
```
<Axes: xlabel='ApplicantIncomelog', ylabel='Density'>
```



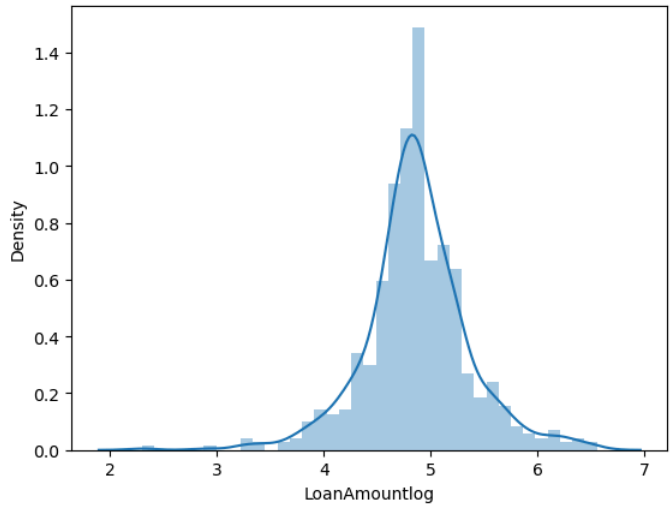
In [153]:
```
df.head()
```

Out[153]:

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

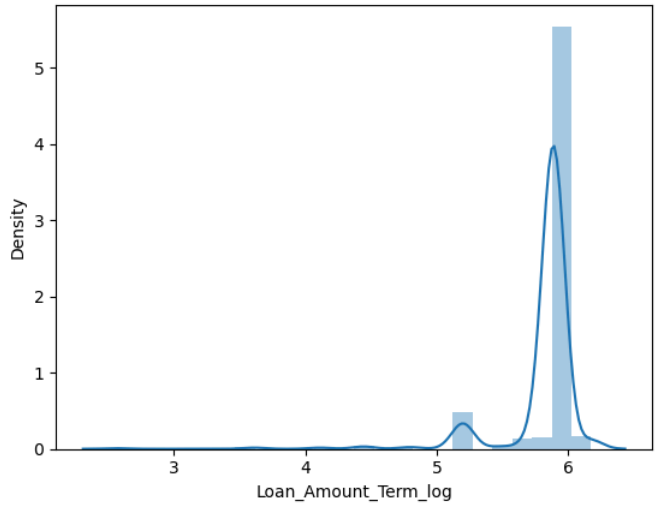
In [154]:
```
df['LoanAmountlog'] = np.log(df['LoanAmount'] + 1)
sns.distplot(df['LoanAmountlog'])
```

Out[154]: <Axes: xlabel='LoanAmountlog', ylabel='Density'>



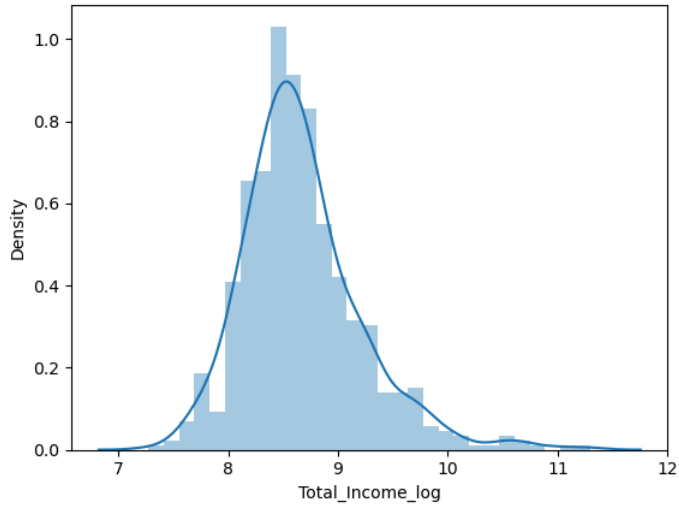
In [155]:
```
df['Loan_Amount_Term_log'] = np.log(df['Loan_Amount_Term'] + 1)
sns.distplot(df['Loan_Amount_Term_log'])
```

Out[155]: <Axes: xlabel='Loan_Amount_Term_log', ylabel='Density'>



In [156]:
```
df['Total_Income_log'] = np.log(df['Total_Income'] + 1)
sns.distplot(df['Total_Income_log'])
```

Out[156]: <Axes: xlabel='Total_Income_log', ylabel='Density'>



In [157]:
```
df.head()
```

Out[157]:

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

In [158]:
```
## drop unnecessary columns
cols = ['ApplicantIncome','CoapplicantIncome','LoanAmount','Loan_Amount_Term','Total_Income','Loan_ID']
df = df.drop(columns = cols, axis = 1)
df.head()
```

Out[158]:

	Gender	Married	Dependents	Education	Self_Employed	Credit_History	Property_Area	Loan_Status	ApplicantIncomelog	LoanAmountlog	Loan_Amount_Term_log	Tot
0	Male	No	0	Graduate	No	1.0	Urban	Y	8.674197	4.859812	5.888878	
1	Male	Yes	1	Graduate	No	1.0	Rural	N	8.430327	4.859812	5.888878	
2	Male	Yes	0	Graduate	Yes	1.0	Urban	Y	8.006701	4.204693	5.888878	
3	Male	Yes	0	Not Graduate	No	1.0	Urban	Y	7.857094	4.795791	5.888878	
4	Male	No	0	Graduate	No	1.0	Urban	Y	8.699681	4.955827	5.888878	

In [159]:
```
## Encoding Technique : Label Encoding, One Hot Encoding

from sklearn.preprocessing import LabelEncoder
cols = ['Gender','Married','Education','Dependents','Self_Employed','Property_Area','Loan_Status']
le =  LabelEncoder()
for col in cols:
  df[col] =  le.fit_transform(df[col])
```

In [160]:
```
df.head()
```

Out[160]:

	Gender	Married	Dependents	Education	Self_Employed	Credit_History	Property_Area	Loan_Status	ApplicantIncomelog	LoanAmountlog	Loan_Amount_Term_log	Tot
0	1	0	0	0	0	1.0	2	1	8.674197	4.859812	5.888878	
1	1	1	1	0	0	1.0	0	0	8.430327	4.859812	5.888878	
2	1	1	0	0	1	1.0	2	1	8.006701	4.204693	5.888878	
3	1	1	0	1	0	1.0	2	1	7.857094	4.795791	5.888878	
4	1	0	0	0	0	1.0	2	1	8.699681	4.955827	5.888878	

In [161]:
```
df.dtypes
```

Out[161]:
```
Gender                  int32
Married                 int32
Dependents              int32
Education               int32
Self_Employed           int32
Credit_History        float64
Property_Area           int32
Loan_Status             int32
ApplicantIncomelog    float64
LoanAmountlog         float64
Loan_Amount_Term_log  float64
Total_Income_log      float64
dtype: object
```

In [162]:
```
## Split Independent and dependent features

X = df.drop(columns = ['Loan_Status'],axis = 1)
y = df['Loan_Status']
```

In [163]:
```
X
```

Out[163]:

	Gender	Married	Dependents	Education	Self_Employed	Credit_History	Property_Area	ApplicantIncomelog	LoanAmountlog	Loan_Amount_Term_log	Total_Income_l
0	1	0	0	0	0	1.0	2	8.674197	4.859812	5.888878	8.6741
1	1	1	1	0	0	1.0	0	8.430327	4.859812	5.888878	8.7147
2	1	1	0	0	1	1.0	2	8.006701	4.204693	5.888878	8.0067
3	1	1	0	1	0	1.0	2	7.857094	4.795791	5.888878	8.5055
4	1	0	0	0	0	1.0	2	8.699681	4.955827	5.888878	8.6996
...	...	...	...	...	...	...	...	...	...	...	
609	0	0	0	0	0	1.0	0	7.972811	4.276666	5.888878	7.9728
610	1	1	3	0	0	1.0	0	8.320448	3.713572	5.198497	8.3204
611	1	1	1	0	0	1.0	2	8.996280	5.537334	5.888878	9.0255
612	1	1	2	0	0	1.0	2	8.933796	5.236442	5.888878	8.9337
613	0	0	0	0	1	0.0	1	8.430327	4.897840	5.888878	8.4303

614 rows × 11 columns

In [164]:
```
y
```

Out[164]:
```
0      1
1      0
2      1
3      1
4      1
      ..
609    1
610    1
611    1
612    1
613    0
Name: Loan_Status, Length: 614, dtype: int32
```

In [165]:
```
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report
```

In [166]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.25,random_state = 42)
```

In [167]:
```
## Logistic Regression
model1 = LogisticRegression()
model1.fit(X_train,y_train)
y_pred_model1 = model1.predict(X_test)
accuracy = accuracy_score(y_test,y_pred_model1)
```

In [168]:
```
print(f"Accuracy of Logistic Regression model is: {accuracy * 100:.2f} %")
```

```
Accuracy of Logistic Regression model is: 77.27 %
```

In [169]:
```
score = cross_val_score(model1,X,y,cv=5)
score
```

Out[169]:
```
array([0.81300813, 0.7804878 , 0.7804878 , 0.85365854, 0.81967213])
```

In [170]:
```
print(f" After cross validation Mean score of the Logistic regression model is: {np.mean(score) * 100:.2f}%")
```

```
 After cross validation Mean score of the Logistic regression model is: 80.95%
```

In [171]:
```
print(len(y_test), len(y_pred_model1))
```

```
154 154
```

In [172]:
```
print(classification_report(y_test, y_pred_model1))
```

```
              precision    recall  f1-score   support

           0       0.91      0.39      0.55        54
           1       0.75      0.98      0.85       100

    accuracy                           0.77       154
   macro avg       0.83      0.68      0.70       154
weighted avg       0.81      0.77      0.74       154
```

In [173]:
```
## Decision Tree Classifier

model2 = DecisionTreeClassifier()
model2.fit(X_train,y_train)
y_pred_model2 = model2.predict(X_test)
accuracy = accuracy_score(y_test,y_pred_model2)
print(f"Accuracy score of Decision Tree: {accuracy * 100:.2f} % ")
```

```
Accuracy score of Decision Tree: 72.08 %
```

In [174]:
```
score = cross_val_score(model2,X,y,cv=5)
print(f"Cross Validation score of Decision Tree: {np.mean(score) * 100:.2f}%")
```

```
Cross Validation score of Decision Tree: 69.71%
```

In [175]:
```
print(classification_report(y_test, y_pred_model2))
```

```
              precision    recall  f1-score   support

           0       0.61      0.56      0.58        54
           1       0.77      0.81      0.79       100

    accuracy                           0.72       154
   macro avg       0.69      0.68      0.69       154
weighted avg       0.72      0.72      0.72       154
```

In [176]:
```
## Random Forest Classifier

model3 = RandomForestClassifier()
model3.fit(X_train,y_train)
y_pred_model3 = model3.predict(X_test)
accuracy = accuracy_score(y_test,y_pred_model3)
print(f"Accuracy score of Random Forest Classifier : {accuracy * 100:.2f} % ")
```

```
Accuracy score of Random Forest Classifier : 79.22 %
```

In [177]:
```
print(classification_report(y_test, y_pred_model3))
```

```
              precision    recall  f1-score   support

           0       0.89      0.46      0.61        54
           1       0.77      0.97      0.86       100

    accuracy                           0.79       154
   macro avg       0.83      0.72      0.73       154
weighted avg       0.81      0.79      0.77       154
```

In [178]:
```
df['Loan_Status'].value_counts()
```

Out[178]:
```
Loan_Status
1    422
0    192
Name: count, dtype: int64
```

Above we can see that there are 422 datasets for 1 (approved) and 192 for (not approved). S0, this might create biasness in our model while prediction. so we will now make these samples equal for 1 and 0

In [179]:
```
pip install -U imbalanced-learn
```

```
Requirement already satisfied: imbalanced-learn in c:\users\satyendra prakash\anaconda3\lib\site-packages (0.12.2)
Requirement already satisfied: numpy>=1.17.3 in c:\users\satyendra prakash\anaconda3\lib\site-packages (from imbalanced-learn) (1.24.3)
Requirement already satisfied: scipy>=1.5.0 in c:\users\satyendra prakash\anaconda3\lib\site-packages (from imbalanced-learn) (1.11.1)
Requirement already satisfied: scikit-learn>=1.0.2 in c:\users\satyendra prakash\anaconda3\lib\site-packages (from imbalanced-learn) (1.
3.0)
Requirement already satisfied: joblib>=1.1.1 in c:\users\satyendra prakash\anaconda3\lib\site-packages (from imbalanced-learn) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\satyendra prakash\anaconda3\lib\site-packages (from imbalanced-learn)
(2.2.0)
Note: you may need to restart the kernel to use updated packages.
```

In [180]:
```
from imblearn.over_sampling import RandomOverSampler
```

In [181]:
```
oversample = RandomOverSampler(random_state=42)
X_resampled, y_resampled = oversample.fit_resample(X,y)

df_resampled = pd.concat([pd.DataFrame(X_resampled,columns=X.columns),pd.Series(y_resampled,name="Loan_status")],axis=1)
```

In [182]:
```
X_resampled
```

Out[182]:

	Gender	Married	Dependents	Education	Self_Employed	Credit_History	Property_Area	ApplicantIncomelog	LoanAmountlog	Loan_Amount_Term_log	Total_Income_l
0	1	0	0	0	0	1.000000	2	8.674197	4.859812	5.888878	8.6741
1	1	1	1	0	0	1.000000	0	8.430327	4.859812	5.888878	8.7147
2	1	1	0	0	1	1.000000	2	8.006701	4.204693	5.888878	8.0067
3	1	1	0	1	0	1.000000	2	7.857094	4.795791	5.888878	8.5055
4	1	0	0	0	0	1.000000	2	8.699681	4.955827	5.888878	8.6996
...	...	...	...	...	...	...	...	...	...	...	
839	1	1	3	1	0	1.000000	2	8.292298	4.859812	5.198497	8.2922
840	1	1	1	0	0	0.842199	0	7.539559	4.127134	5.888878	7.5395
841	1	1	1	0	0	0.000000	0	7.933080	4.990433	5.888878	8.4563
842	1	1	2	1	0	0.000000	2	7.969012	3.828641	5.198497	7.9690
843	1	0	0	0	0	0.000000	1	8.334952	4.595120	5.888878	8.3349

844 rows × 11 columns

In [183]:
```
y_resampled
```

Out[183]:
```
0      1
1      0
2      1
3      1
4      1
      ..
839    0
840    0
841    0
842    0
843    0
Name: Loan_Status, Length: 844, dtype: int32
```

In [184]:
```
y_resampled.value_counts()
```

Out[184]:
```
Loan_Status
1    422
0    422
Name: count, dtype: int64
```

Here , you can see that now datasets for 1 and 0 are equal . so we can proceed further.

In [185]:
```
X_resampled_train, X_resampled_test, y_resampled_train, y_resampled_test = train_test_split(X_resampled,y_resampled,test_size = 0.25,rand
```

In [186]:
```
## Logistic Regression
model1 = LogisticRegression()
model1.fit(X_resampled_train,y_resampled_train)
y_pred_model1 = model1.predict(X_resampled_test)
accuracy = accuracy_score(y_resampled_test,y_pred_model1)
print(f"Accuracy of Logistic Regression model is: {accuracy * 100:.2f} %")
```

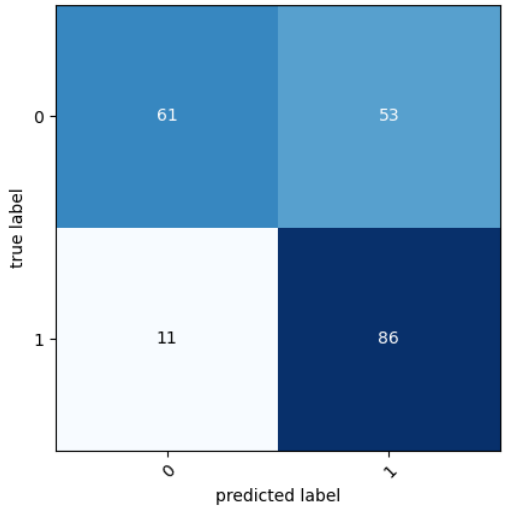
```
Accuracy of Logistic Regression model is: 69.67 %
```

In [192]:
```
print(classification_report(y_resampled_test,y_pred_model1))
```

```
              precision    recall  f1-score   support

           0       0.85      0.54      0.66       114
           1       0.62      0.89      0.73        97

    accuracy                           0.70       211
   macro avg       0.73      0.71      0.69       211
weighted avg       0.74      0.70      0.69       211
```

In [233]:
```
from mlxtend.plotting import plot_confusion_matrix

cm = confusion_matrix(y_resampled_test,y_pred_model1)
plot_confusion_matrix(cm, class_names= model1.classes_)
```

Out[233]:
```
(<Figure size 640x480 with 1 Axes>,
 <Axes: xlabel='predicted label', ylabel='true label'>)
```



In [188]:
```
## Decision Tree Classifier

model2 = DecisionTreeClassifier()
model2.fit(X_resampled_train,y_resampled_train)
y_pred_model2 = model2.predict(X_resampled_test)
accuracy = accuracy_score(y_resampled_test,y_pred_model2)
print(f"Accuracy of Decision Tree Classifier is: {accuracy * 100:.2f} %")
```

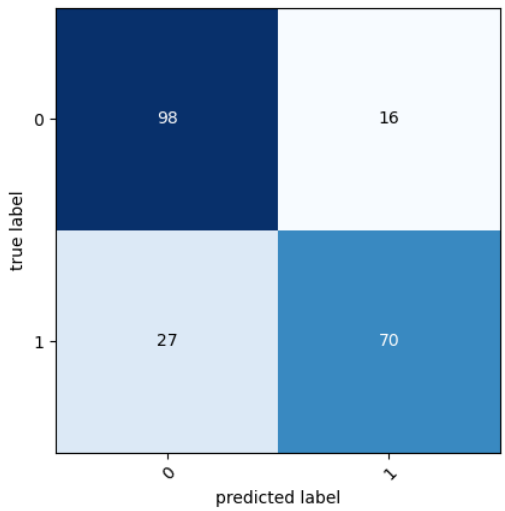
```
Accuracy of Decision Tree Classifier is: 79.62 %
```

In [193]:
```
print(classification_report(y_resampled_test,y_pred_model2))
```

```
              precision    recall  f1-score   support

           0       0.78      0.86      0.82       114
           1       0.81      0.72      0.77        97

    accuracy                           0.80       211
   macro avg       0.80      0.79      0.79       211
weighted avg       0.80      0.80      0.79       211
```

In [234]:
```
cm = confusion_matrix(y_resampled_test,y_pred_model2)
plot_confusion_matrix(cm, class_names= model2.classes_)
```

Out[234]:
```
(<Figure size 640x480 with 1 Axes>,
 <Axes: xlabel='predicted label', ylabel='true label'>)
```



In [190]:
```
## Random Forest Classifier
model3 = RandomForestClassifier()
model3.fit(X_resampled_train,y_resampled_train)
y_pred_model3 = model3.predict(X_resampled_test)
accuracy = accuracy_score(y_resampled_test,y_pred_model3)
print(f"Accuracy of Random Forest Classifier is: {accuracy * 100:.2f} %")
```

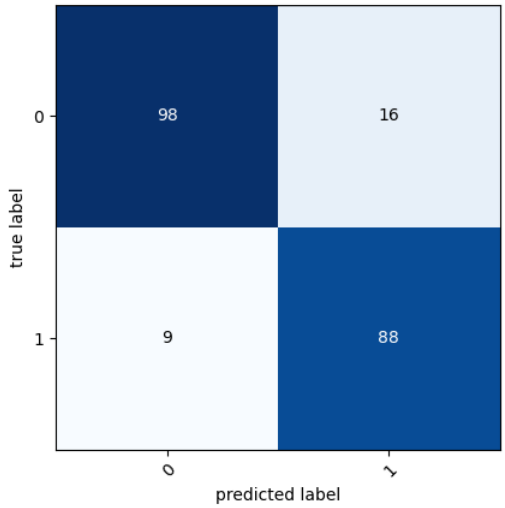
```
Accuracy of Random Forest Classifier is: 88.15 %
```

In [195]:
```
print(classification_report(y_resampled_test,y_pred_model3))
```

```
              precision    recall  f1-score   support

           0       0.92      0.86      0.89       114
           1       0.85      0.91      0.88        97

    accuracy                           0.88       211
   macro avg       0.88      0.88      0.88       211
weighted avg       0.88      0.88      0.88       211
```

In [235]:
```
cm = confusion_matrix(y_resampled_test,y_pred_model3)
plot_confusion_matrix(cm, class_names= model3.classes_)
```

Out[235]:
```
(<Figure size 640x480 with 1 Axes>,
 <Axes: xlabel='predicted label', ylabel='true label'>)
```



Now, lets take a sample data and predict its loan application approval status.

we will us model 3 i.e Random Forest Classifier as it has best accuracy

In [230]:
```
# Sample data for prediction
sample_loan_data = {
    'Gender': 'Male',
    'Married': 'no',
    'Dependents': 2,
    'Education': 'Graduate',
    'Self_Employed': 'no',
    'Credit_History': 1.0,
    'Property_Area': 'Rural',
    'ApplicantIncome': 5000,
    'CoapplicantIncome': 0.0,
    'LoanAmount': 130,
    'Loan_Amount_Term': 360.0
}

# Preprocess the sample data
sample_df = pd.DataFrame([sample_loan_data])

# Encode categorical variables
categorical_cols = ['Gender', 'Married', 'Education', 'Self_Employed', 'Property_Area']
for col in categorical_cols:
    sample_df[col] = label_encoder.fit_transform(sample_df[col])

# Fill missing values for numerical variables
sample_df['LoanAmount'] = sample_df['LoanAmount'].fillna(sample_df['LoanAmount'].median())
sample_df['Loan_Amount_Term'] = sample_df['Loan_Amount_Term'].fillna(sample_df['Loan_Amount_Term'].mean())
sample_df['Credit_History'] = sample_df['Credit_History'].fillna(sample_df['Credit_History'].mean())

# Log transform features
sample_df['ApplicantIncomelog'] = np.log(sample_df['ApplicantIncome'] + 1)
sample_df['LoanAmountlog'] = np.log(sample_df['LoanAmount'] + 1)
sample_df['Loan_Amount_Term_log'] = np.log(sample_df['Loan_Amount_Term'] + 1)
sample_df['Total_Income'] = sample_df['ApplicantIncome'] + sample_df['CoapplicantIncome']
sample_df['Total_Income_log'] = np.log(sample_df['Total_Income'] + 1)

# Ensure all features are present
required_features = ['Gender', 'Married', 'Dependents', 'Education', 'Self_Employed',
                     'Credit_History', 'Property_Area', 'ApplicantIncomelog', 'LoanAmountlog',
                     'Loan_Amount_Term_log', 'Total_Income_log']

# Check if all required features are present
missing_features = set(required_features) - set(sample_df.columns)
if missing_features:
    print("Missing features in the sample DataFrame:", missing_features)
else:
    # Predict loan status using the trained logistic regression model
    sample_prediction = model3.predict(sample_df[required_features])

    # Interpret the prediction
    if sample_prediction[0] == 1:
        print("The model predicts that the loan will be approved.")
    else:
        print("The model predicts that the loan will not be approved.")
```

```
The model predicts that the loan will be approved.
```

In [231]:
```
# Sample data for prediction
sample_loan_data = {
    'Gender': 'Male',
    'Married': 'Yes',
    'Dependents': 2,
    'Education': 'Graduate',
    'Self_Employed': 'yes',
    'Credit_History': 1.0,
    'Property_Area': 'Urban',
    'ApplicantIncome': 500,
    'CoapplicantIncome': 0.0,
    'LoanAmount': 130,
    'Loan_Amount_Term': 360.0
}

# Preprocess the sample data
sample_df = pd.DataFrame([sample_loan_data])

# Encode categorical variables
categorical_cols = ['Gender', 'Married', 'Education', 'Self_Employed', 'Property_Area']
for col in categorical_cols:
    sample_df[col] = label_encoder.fit_transform(sample_df[col])

# Fill missing values for numerical variables
sample_df['LoanAmount'] = sample_df['LoanAmount'].fillna(sample_df['LoanAmount'].median())
sample_df['Loan_Amount_Term'] = sample_df['Loan_Amount_Term'].fillna(sample_df['Loan_Amount_Term'].mean())
sample_df['Credit_History'] = sample_df['Credit_History'].fillna(sample_df['Credit_History'].mean())

# Log transform features
sample_df['ApplicantIncomelog'] = np.log(sample_df['ApplicantIncome'] + 1)
sample_df['LoanAmountlog'] = np.log(sample_df['LoanAmount'] + 1)
sample_df['Loan_Amount_Term_log'] = np.log(sample_df['Loan_Amount_Term'] + 1)
sample_df['Total_Income'] = sample_df['ApplicantIncome'] + sample_df['CoapplicantIncome']
sample_df['Total_Income_log'] = np.log(sample_df['Total_Income'] + 1)

# Ensure all features are present
required_features = ['Gender', 'Married', 'Dependents', 'Education', 'Self_Employed',
                     'Credit_History', 'Property_Area', 'ApplicantIncomelog', 'LoanAmountlog',
                     'Loan_Amount_Term_log', 'Total_Income_log']

# Check if all required features are present
missing_features = set(required_features) - set(sample_df.columns)
if missing_features:
    print("Missing features in the sample DataFrame:", missing_features)
else:
    # Predict loan status using the trained logistic regression model
    sample_prediction = model3.predict(sample_df[required_features])

    # Interpret the prediction
    if sample_prediction[0] == 1:
        print("The model predicts that the loan will be approved.")
    else:
        print("The model predicts that the loan will not be approved.")
```

```
The model predicts that the loan will not be approved.
```

In [ ]:
```

```