D. Y. PATIL COLLEGE OF ENGINEERING, AKURDI, PUNE

Affiliated to

SAVITRIBAI PHULE PUNE UNIVERSITY

Department of Information Technology

Mini Project Of Data Science & Big Data Analytics

on

Loan Status Prediction

By

Harshita Totala [TEITA14]

Yash Birajdar [TEITA15]

Dnyaneshwar Ghule [TEITA17]

Yuvraj Singh [TEITA22]

Under the guidance of

Mrs. Priyanka Gupta

Course Name: Data Science & Big Data Analytics

Course code: 314452

Semester-II

Class-TE

2022-2023

Overview

Banks are making major part of profits through loans. Though lot of people are applying for loans. It's hard to select the genuine applicant, who will repay the loan. While doing the process manually, lot of misconception may happen to select the genuine applicant.

Therefore, developing loan prediction system using machine learning, so the system automatically selects the eligible candidates. This is helpful to both bank staff and applicant. The time period for the sanction of loan will be drastically reduced. In this project we are predicting the loan data by using some machine learning algorithms.

The major aim of this project is to predict which of the customers will have their loan paid or not. Therefore, this is a supervised classification problem to be trained with algorithms like Logistic Regression, Decision Tree Classifier.

Motivation:

This project was started as a motivation for learning Machine Learning Algorithms and to learn the different data pre processing techniques such as Exploratory Data Analysis, Feature Engineering, Feature Selection, Feature Scaling and finally to build a machine learning model. In this project we are going to classify an individual whether he/she able to get the loan amount based on his/her Income, Education, Working Experience, Loan which is taken previously and many more factors. The dataset is collected from Kaggle.

Dataset Used:

The dataset contains information about Loan Applicants. There are 12 independent columns and 1 dependent column. This dataset includes attributes like Loan ID, gender, if the loan applicant is married or not, the level of education, applicant's income ,etc.

1.Loan_ID: A unique ID assigned to every loan applicant
2.Gender: Gender of the applicant (Male, Female)
3.Married: The marital status of the applicant (Yes, No)
4.Dependents: No. of people dependent on the applicant (0,1,2,3+)
5.Education: Education level of the applicant (Graduated, Not Graduated)
6.Self_Employed: If the applicant is self-employed or not (Yes, No)
7.ApplicantIncome: The amount of income the applicant earns
8.CoapplicantIncome: The amount of income the co-applicant earns
9.LoanAmount: The amount of loan the applicant has requested for
10.Loan_Amount_Term: The no. of days over which the loan will be paid
11.Credit_History: A record of a borrower's responsible repayment of debts (1- has all debts paid, 0- not paid)
12.Property_Area : The type of location where the applicant's property lies (Rural, Semiurban, Urban)

Target:
13.Loan_Status: Loan granted or not (Y, N)

Conclusion:

Loan companies grant loans after a thorough verification and validation process. However, they do not know with absolute certainty whether the applicant will be able to repay the loan without difficulty. The loan Prediction System will allow them to choose the most deserving applicants quickly, easily, and efficiently. It may provide the bank with unique benefits. We have built a Logistic Regression which performs well with selected features such as credit_history, loan_amount, applicant_income, co-applicant_income, dependents and having the accuracy as 79.5%.

LOAN PREDICTION USING MACHINE LEARNING

In [1]:
```
# importing essential libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
# reading csv file
loan_data = pd.read_csv(r"C:\Users\HP\Downloads\train.csv")
```

In [3]:
```
# printing first five rows of dataset
loan_data.head(5)
```

Out[3]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapplic
0	LP001002	Male	No	0	Graduate	No	5849	
1	LP001003	Male	Yes	1	Graduate	No	4583	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	
4	LP001008	Male	No	0	Graduate	No	6000	

In [4]:
```
# Printing last five rows of datset
loan_data.tail(5)
```

Out[4]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapp
609	LP002978	Female	No	0	Graduate	No	2900	
610	LP002979	Male	Yes	3+	Graduate	No	4106	
611	LP002983	Male	Yes	1	Graduate	No	8072	
612	LP002984	Male	Yes	2	Graduate	No	7583	
613	LP002990	Female	No	0	Graduate	Yes	4583	

In [5]:
```
# Obtaining the dimensions of dataset
loan_data.shape
```

Out[5]:
```
(614, 13)
```

In [6]:
```
# Gives decription of the dataset
loan_data.describe()
```

Out[6]:

	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 [7]:

```python
# Statistical summary of dataset
loan_data.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
```

Data Preprocessing

In [8]:

```python
# Check null values
loan_data.isnull().sum()
```

Out[8]:

```
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
```

Dealing with Categorical values

In [9]:
```
# Gender Column
loan_data['Gender'] =  loan_data['Gender'].map({'Male':0,'Female':1})

# Married column
loan_data['Married'] = loan_data['Married'].map({'No':0,'Yes':1})

# Loan_Status column
loan_data['Loan_Status'] =loan_data['Loan_Status'].map({'N':0,'Y':1})
```

In [10]:
```
loan_data
```

Out[10]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapp
0	LP001002	0.0	0.0	0	Graduate	No	5849	
1	LP001003	0.0	1.0	1	Graduate	No	4583	
2	LP001005	0.0	1.0	0	Graduate	Yes	3000	
3	LP001006	0.0	1.0	0	Not Graduate	No	2583	
4	LP001008	0.0	0.0	0	Graduate	No	6000	
...	...	...	...	...	...	...	...	
609	LP002978	1.0	0.0	0	Graduate	No	2900	
610	LP002979	0.0	1.0	3+	Graduate	No	4106	
611	LP002983	0.0	1.0	1	Graduate	No	8072	
612	LP002984	0.0	1.0	2	Graduate	No	7583	
613	LP002990	1.0	0.0	0	Graduate	Yes	4583	

614 rows × 13 columns

Filling Missing Values

In [11]:
```
# Gender column
loan_data['Gender'] = loan_data['Gender'].fillna(loan_data['Gender'].mode()[0])
```

In [12]:
```
# Married column
loan_data['Married'] = loan_data['Married'].fillna(loan_data['Married'].mode()[0])
```

In [13]:
```
# Dependents Column
loan_data['Dependents'] = loan_data['Dependents'].fillna(loan_data['Dependents'].m
```

In [14]:
```
# Self_Employed Column
loan_data['Self_Employed'].fillna('No',inplace=True)
```

In [15]:
```
# Credit_History Column
loan_data['Credit_History'] = loan_data['Credit_History'].fillna(loan_data['Credit_
```

In [16]:
```
# LoanAmount Column
loan_data['LoanAmount'] = loan_data['LoanAmount'].fillna(loan_data['LoanAmount'].m
```

In [17]:
```
# Loan_Amount term Column
loan_data['Loan_Amount_Term'] = loan_data['Loan_Amount_Term'].fillna(loan_data['Lo
```

In [18]:
```
#check missing values

loan_data.isnull().sum()
```

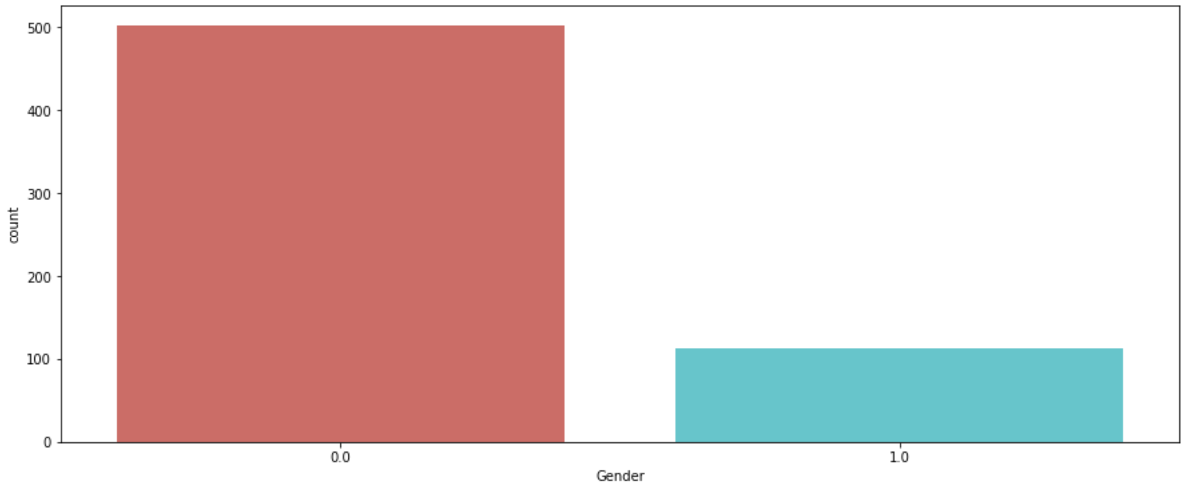
Out[18]:
```
Loan_ID              0
Gender               0
Married              0
Dependents           0
Education            0
Self_Employed        0
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           0
Loan_Amount_Term     0
Credit_History       0
Property_Area        0
Loan_Status          0
dtype: int64
```

Exploratory Data Analysis

In [19]:
```
# Counting the accurance of each value in Gender column
loan_data['Gender'].value_counts()
```

Out[19]:
```
0.0    502
1.0    112
Name: Gender, dtype: int64
```

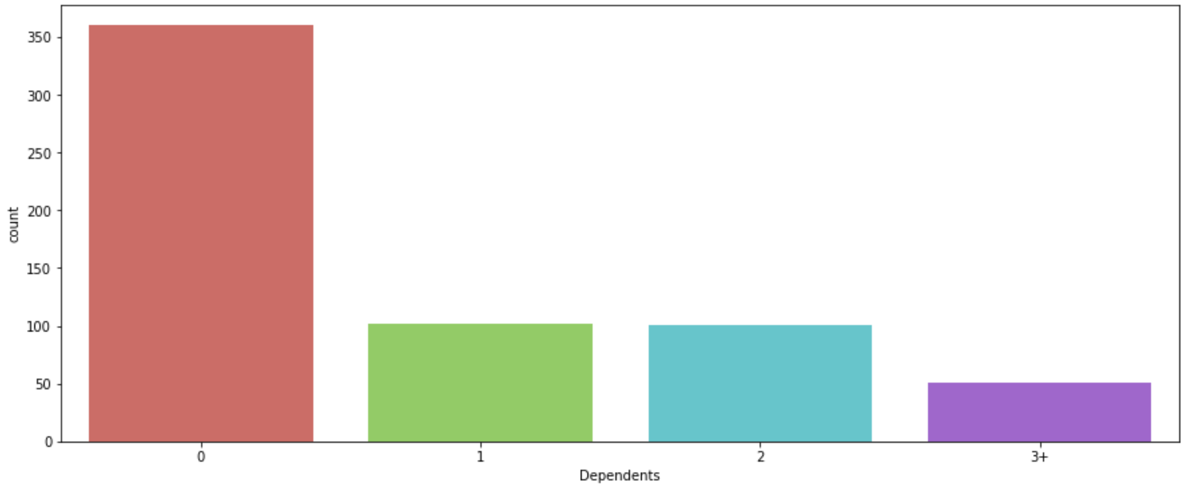
In [20]:
```
plt.figure(figsize=(15,6))
sns.countplot('Gender', data= loan_data, palette = 'hls' )
plt.show()
```



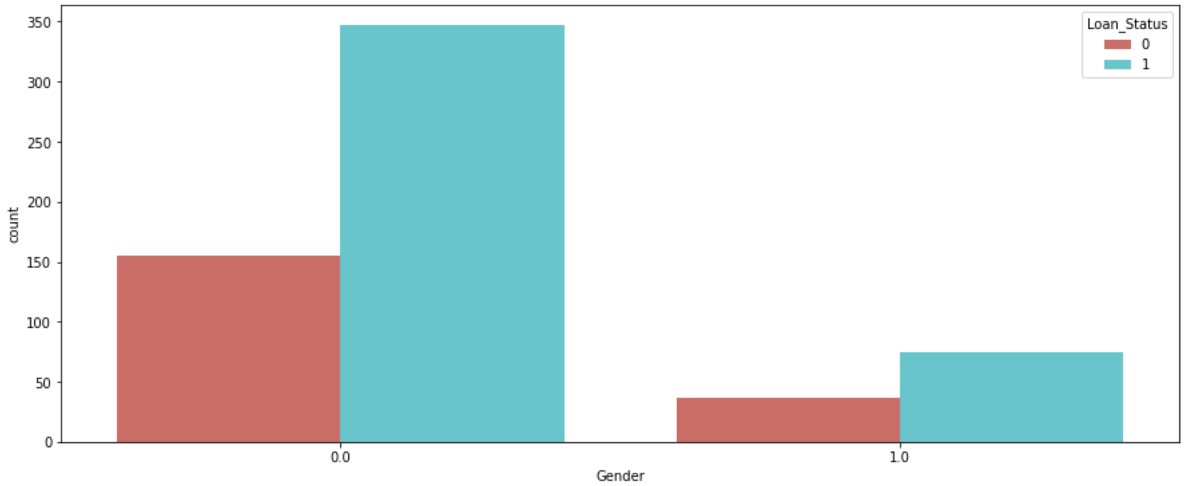
In [21]:
```
# Counting the accurance of each value in Dependent column
loan_data['Dependents'].value_counts()
```

```
Out[21]: 0      360
         1      102
         2      101
         3+      51
         Name: Dependents, dtype: int64
```

```
In [22]: plt.figure(figsize=(15,6))
         sns.countplot('Dependents', data = loan_data, palette='hls')
         plt.show()
```



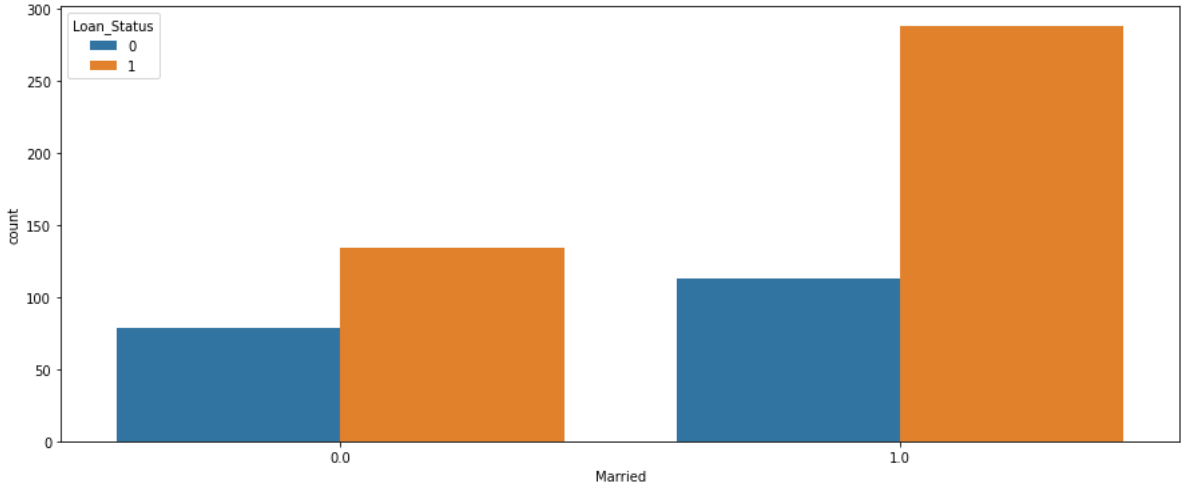

```
In [23]: # comparing loan status with gender column
         plt.figure(figsize=(15,6))
         sns.countplot(x = 'Gender',hue ='Loan_Status', data=loan_data , palette='hls')
         plt.show()
```



More males are on loan than females. Also, those that are on loan are more than otherwise

```
In [24]: # comparing loan status with married column
         plt.figure(figsize = (15,6))
         sns.countplot( x='Married', hue ='Loan_Status', data = loan_data)
```

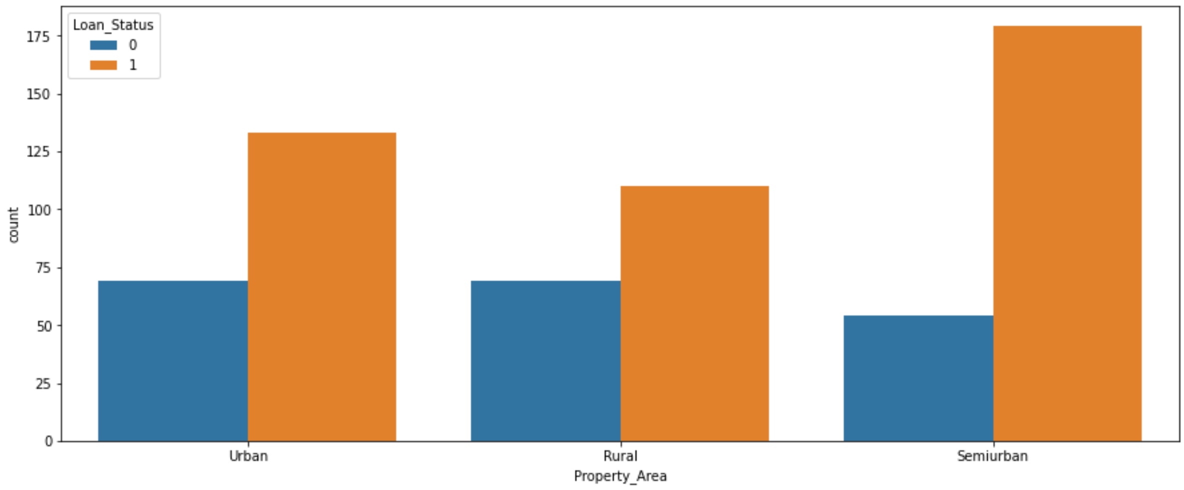
```
Out[24]: <AxesSubplot:xlabel='Married', ylabel='count'>
```



Married people collect more loan than unmarried

In [25]:
```
plt.figure(figsize=(15,6))
sns.countplot(x = 'Property_Area', hue='Loan_Status', data = loan_data)
```

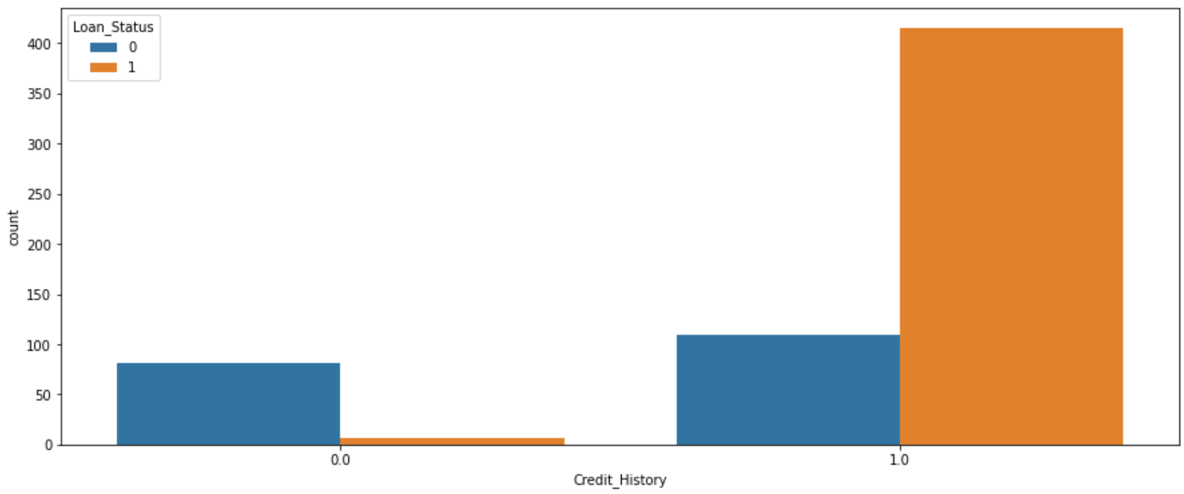
Out[25]: <AxesSubplot:xlabel='Property_Area', ylabel='count'>



Semiurban obtain more loan, folowed by Urban and then rural.

In [26]:
```
plt.figure(figsize=(15,6))
sns.countplot(x = 'Credit_History', hue='Loan_Status', data = loan_data)
```

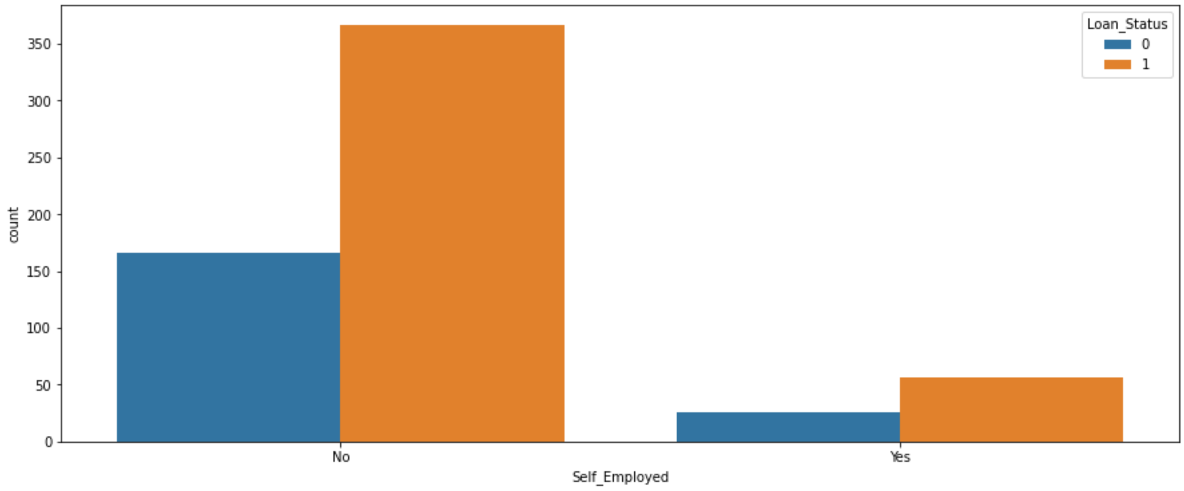
Out[26]: <AxesSubplot:xlabel='Credit_History', ylabel='count'>



According to the credit history, greater number of people pay back their loans.

In [27]:
```
plt.figure(figsize=(15,6))
sns.countplot( x = 'Self_Employed', hue = 'Loan_Status', data = loan_data)
```

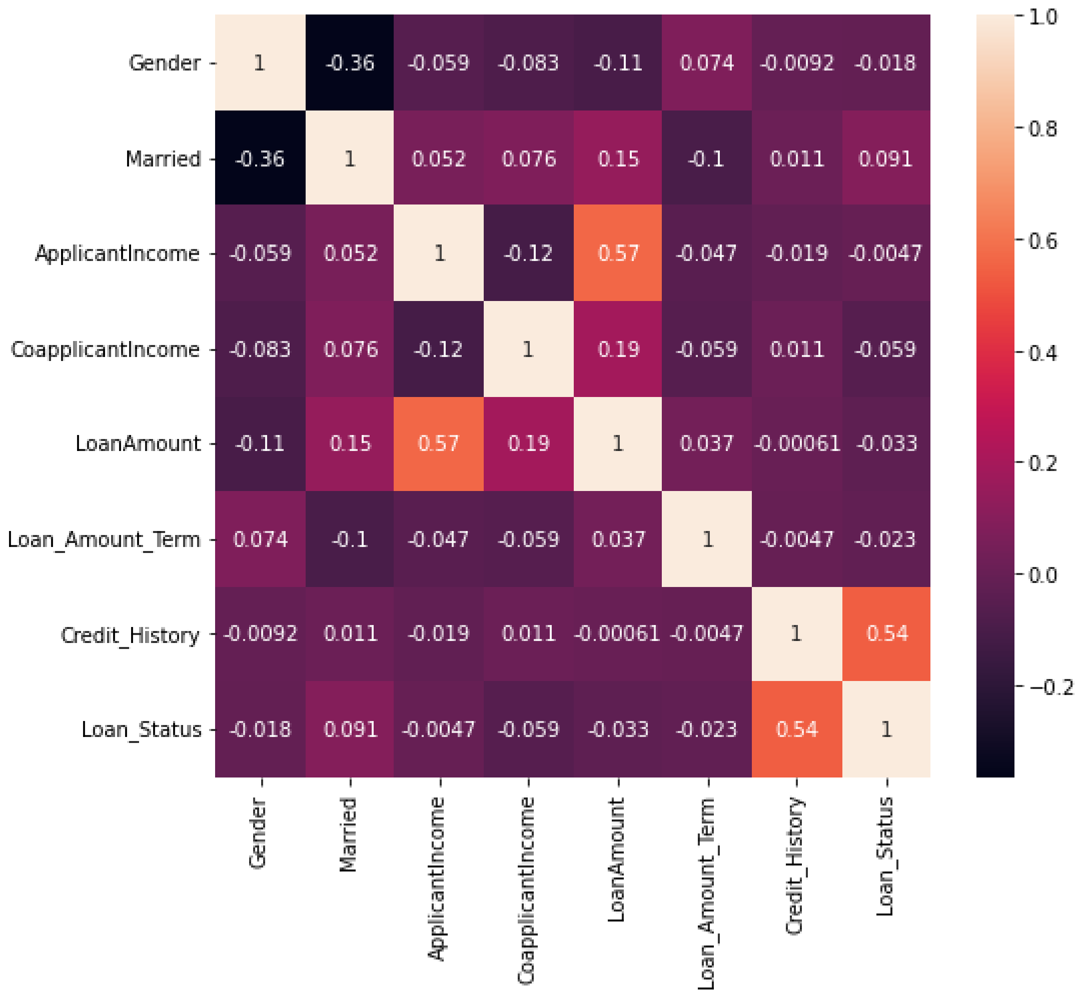
Out[27]:
```
<AxesSubplot:xlabel='Self_Employed', ylabel='count'>
```



The category of those that take loans is less of self-employed people. That those who are not self-employed probably salary earners obtain more loan.

In [28]:
```
# Showing correlation through heatmap
plt.figure(figsize=(8,7))
sns.heatmap(loan_data.corr(), annot=True)
```

Out[28]:
```
<AxesSubplot:>
```



From the above figure, we can see that Credit_History (Independent Variable) has the maximum correlation with Loan_Status (Dependent Variable). Which denotes that the Loan_Status is heavily dependent on the Credit_History

Label Encoding

In [29]:
```
from sklearn.preprocessing import LabelEncoder
cols = ['Gender','Married','Education','Self_Employed','Property_Area','Loan_Statu
le = LabelEncoder()
for col in cols:
    loan_data[col] = le.fit_transform(loan_data[col])
```

Model Building

In [30]:
```
X = loan_data[['Gender', 'Married', 'ApplicantIncome', 'LoanAmount',
'Credit_History']]
y = loan_data.Loan_Status
```

In [31]:
```
# Data Splitting
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X,y, test_size = 0.2,random_st
```

Feature Selection

In [32]:
```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()

x_train=sc.fit_transform(x_train)
x_test=sc.fit_transform(x_test)
```

Logistic Regression

In [33]:
```
from sklearn.linear_model import LogisticRegression
from sklearn import metrics

model = LogisticRegression()
model.fit(x_train, y_train)

lr_prediction = model.predict(x_test)
print("y_predicted",lr_prediction)
```

```
y_predicted [1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 0 1
1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
 1 1 1 1 0 1 1 0 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 0 1 1 1 1]
```

In [34]:
```
print('Logistic Regression accuracy = ', metrics.accuracy_score(lr_prediction,y_te
```

```
Logistic Regression accuracy =  0.7967479674796748
```

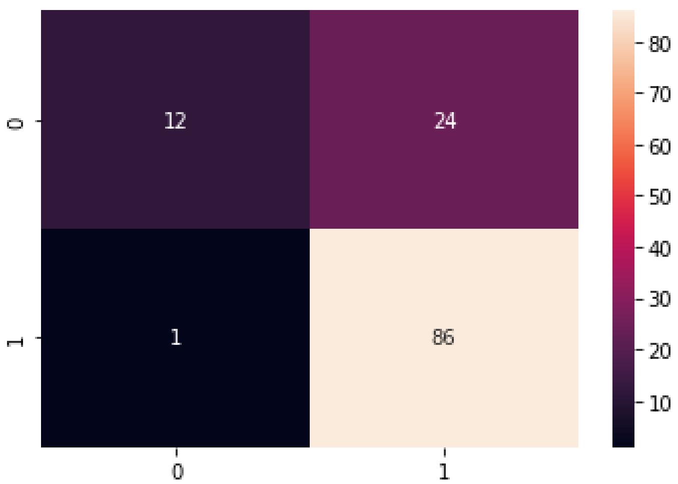
Confusion Matrix

In [35]:
```
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,lr_prediction)
print(cm)
```

```
[[12 24]
 [ 1 86]]
```

In [36]:
```
sns.heatmap(cm,annot=True)
```

Out[36]: <AxesSubplot:>



In [37]:
```
from sklearn.metrics import classification_report

print(classification_report(y_test,lr_prediction))
```

	precision	recall	f1-score	support
0	0.92	0.33	0.49	36
1	0.78	0.99	0.87	87
accuracy			0.80	123
macro avg	0.85	0.66	0.68	123
weighted avg	0.82	0.80	0.76	123