PROPOSAL:

Loan eligibity prediction is an important thing which is used to predict whether a person can be given loan or not based on the parameters like ApplicantIncome,Credit_History,etc.

In [ ]:
```
  The prediction model not only helps applicants to know their loan status but
also helps the bank by minimizing the risk and reducing the number of defaulters.
```

In [ ]:
```
  Target variable:Loan_Status
        Loan status can have two values: Yes or NO.
            Y: If the loan is approved
            N: If the loan is not approved
```

In [ ]:
```
Research question:
        Explore whether a person is eligible to get loan or not based on his
        Income,Co-ApplicantIncome,Credit_History,Loan_Amount.
```

In [ ]:
```
  So to classify whether a person is eligible to get loan or not,
    we use the Classification model "LOGISTIC REGRESSION".
    LOGISTIC REGRESSION:
        logistic regression is a predictive analysis.
        Logistic regression is used to describe data and to explain the relationship
        between one dependent binary variable and one or more nominal, ordinal,
        interval or ratio-level independent variables.
```

In [ ]:
```
USAGE OF LOGISTIC REGRESSION IN OUR PROJECT:
    Since our motive is to predict whether a person gets loan or not,
    we use binary classification "LOGISTIC REGRESSION".
```

In [ ]:
```
  The dataset used here is:"loan_eligibity.csv".
        This dataset contains variables like:
            Loan_ID(which is the ID given to each person applying for loan),
            Gender,
            Married(whether a person is married or not),
            Dependents,
            Education,
            Self_Employeed(whether a person is self employeed or not),
            ApplicantIncome(income of the applicant),
            CoapplicantIncome(income of the co-applicant),
            LoanAmount(in thousands)(amount needed by the applicant),
            Loan_Amount_Term(length of the time taken to pay the loan completely),
            Credit_History(record of how a person has managed his or
                           her credit in the past),
            Property_Area(area where the property is located),
            Loan_Status(stating whether a person gets loan or not)
```

Importing pandas package:

In [1]:

```
import pandas as pd
```

Loading the dataset:

In [2]:

```
loan_data=pd.read_csv("loan-eligibility.csv")
```

In [3]:

```
loan_data.head(5)
```

Out[3]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantInc
0	LP001002	Male	No	0	Graduate	No	5849	
1	LP001003	Male	Yes	1	Graduate	No	4583	15
2	LP001005	Male	Yes	0	Graduate	Yes	3000	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	23
4	LP001008	Male	No	0	Graduate	No	6000	

In [4]:

```
loan_data.info()
#gives the basic information about the dataset
```

```
<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
```

From the info() method,we can see that there are: 614 rows and 13 columns(variables) in our dataset.

In [5]:

```
loan_data.describe()
#gives statistical summary of numerical columns present in the dataset.
```

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

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
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

ANALYSIS:

In [ ]:

```
Statement of research question:
    To automate the Loan Eligibility process by a company related to
    customer's detail provided while applying for loan application and to
    determine whether a person is eligible to get loan or not based on
    ApplicantIncome,CoapplicantIncome,Credit_History,Loan_Amount.
Model objective:
    The model objective is to predict whether a person is eligible to get
    loan or not.
    Since here we have to predict only Yes/No,we use binary classification
    i.e.here we use LOGISTIC REGRESSION.
```

**DATA DESCRIPTION:

In [7]:

```
loan_data.head()
# head() gives the first 5 rows of the dataset
```

Out[7]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantInc
0	LP001002	Male	No	0	Graduate	No	5849	
1	LP001003	Male	Yes	1	Graduate	No	4583	15
2	LP001005	Male	Yes	0	Graduate	Yes	3000	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	23
4	LP001008	Male	No	0	Graduate	No	6000	

**What are the column names?

In [8]:

```
loan_data.columns
```

Out[8]:

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

**What type of columns you have?

In [9]:
```
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
```

In [ ]:
```
info() method gives the basic information about the dataset like:
    No.of observations(rows)
    No.of variables(columns)
    Data type of each columns
    No.of observations present in each column
    No.of null values present in each column
Interpretation :
    We can see that our dataset contains:
        614 observations
        13 columns
```

In [ ]:
```
Description of the columns present in the dataset:
    Loan_ID-object type-ID given to each person applying for loan
    Gender-object type
    Married-object type-whether a person is married or not
    Dependents-object type-tells the no.of dependents the applicant has
    Education-object type-gives information about the educational status
    Self_Employeed-object type-whether a person is self employeed or not
    ApplicantIncome-Integer type-income of the applicant
    CoapplicantIncome-Float type-income of the co-applicant
    LoanAmount-Float type-amount needed by the applicant,
    Loan_Amount_Term-Float type-length of the time taken to pay the loan completely
    Credit_History-Float type-record of how a person has managed
    his or her credit in the past
    Property_Area-object type-area where the property is located
    Loan_Status-object type-stating whether a person gets loan or not
```

In [ ]:
```
Target/Response variable:
    Here "Loan_Status" is the response variable.It contains values Y/N.
    Y indicates that a person is eligible to get a loan.
    N indicates that a person is not eligible to get a loan.
    It is of object type.
    So,let's first convert Y/N to 1/0.
```

In [3]:
```
#Converting object to numeric type in Loan_Status column:
loan_data['Loan_Status'].replace(['Y','N'],[1,0],inplace=True)
```

In [4]:
```
loan_data.head()
```

Out[4]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantInc
0	LP001002	Male	No	0	Graduate	No	5849	
1	LP001003	Male	Yes	1	Graduate	No	4583	15
2	LP001005	Male	Yes	0	Graduate	Yes	3000	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	23
4	LP001008	Male	No	0	Graduate	No	6000	

In [12]:
```
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    int64
dtypes: float64(4), int64(2), object(7)
memory usage: 62.5+ KB
```

We can see that Loan_Status column have been converted to Integer type.

**IS THERE ANY NULL VALUES IN THE DATASET?

In [5]:
```
loan_data.isnull().sum()
```

Out[5]:
```
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 [ ]:

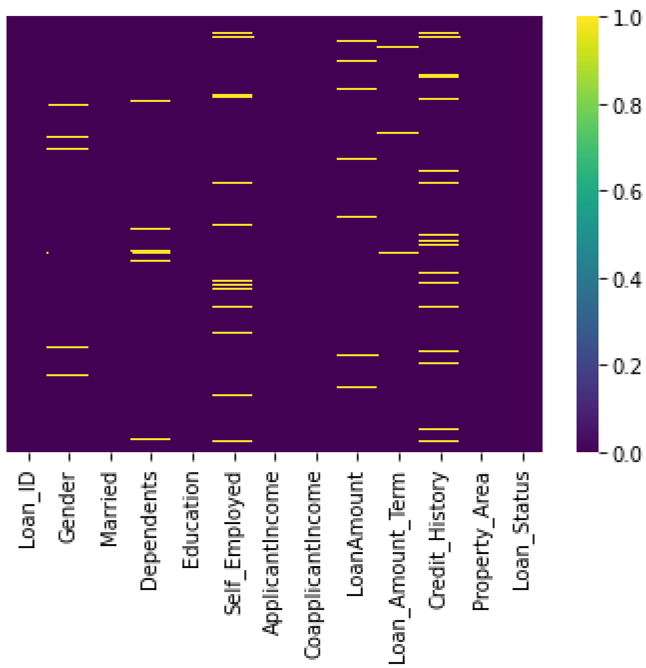
```
We can see that:
    There are 13 null values in Gender column
    There are 3 null values in Married column
    There are 15 null values in Dependents column
    There are 32 null values in Self_Employed column
    There are 22 null values in LoanAmount column
    There are 14 null values in Loan_Amount_Term column
    There are 50 null values in Credit_History column
```

**Checking for null values with heatmap

In [6]:

```
import seaborn as sns
sns.heatmap(loan_data.isnull(),yticklabels=False,cmap="viridis")
```

Out[6]: <AxesSubplot:>



We can see that there are some missing values in some columns.

**Filling the null values:

Fill the null values of the object type column by NULL. For numeric type,fill the null values by their respective column means.

In [7]:

```
loan_data['Gender']=loan_data['Gender'].fillna("NULL")
loan_data['Married']=loan_data['Married'].fillna("NULL")
loan_data['Dependents']=loan_data['Dependents'].fillna("NULL")
loan_data['Self_Employed']=loan_data['Self_Employed'].fillna("NULL")
loan_data['LoanAmount']=loan_data['LoanAmount'].fillna(loan_data['LoanAmount'].mean())
loan_data['Loan_Amount_Term']=loan_data['Loan_Amount_Term'].fillna(loan_data['Loan_Amou
loan_data['Credit_History']=loan_data['Credit_History'].fillna(loan_data['Credit_Histor
```

In [8]:
```
loan_data.isnull().sum()
```

Out[8]:
```
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
```

We can see that there are no null values present.

**EXPLORATORY DATA ANALYSIS:

In [17]:
```
import seaborn as sns
```

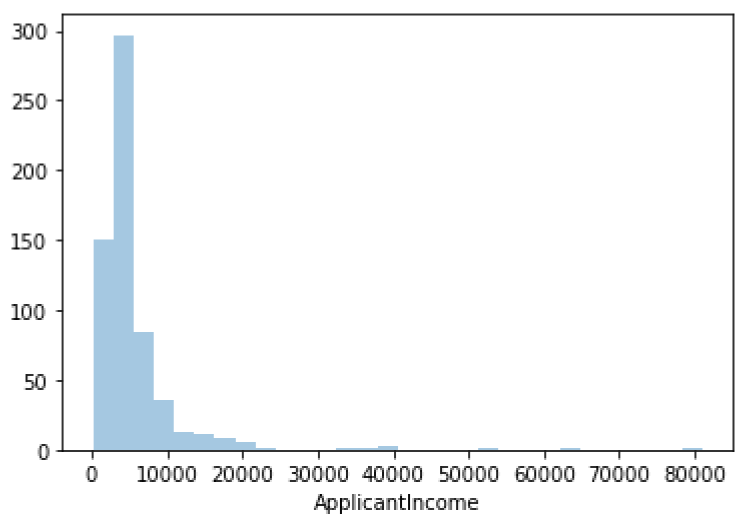
DISTRIBUTION PLOT:used for univariate numeric data

**How is ApplicantIncome distributed?

In [18]:
```
sns.distplot(loan_data['ApplicantIncome'],kde=False,bins=30)
```

```
C:\Users\nivet\anaconda\anaconda3\lib\site-packages\seaborn\distributions.py:2557: Futur
eWarning: `distplot` is a deprecated function and will be removed in a future version. P
lease adapt your code to use either `displot` (a figure-level function with similar flex
ibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[18]: <AxesSubplot:xlabel='ApplicantIncome'>



Interpretation:We can see that the nearly 560 persons have income in the range 0-10000

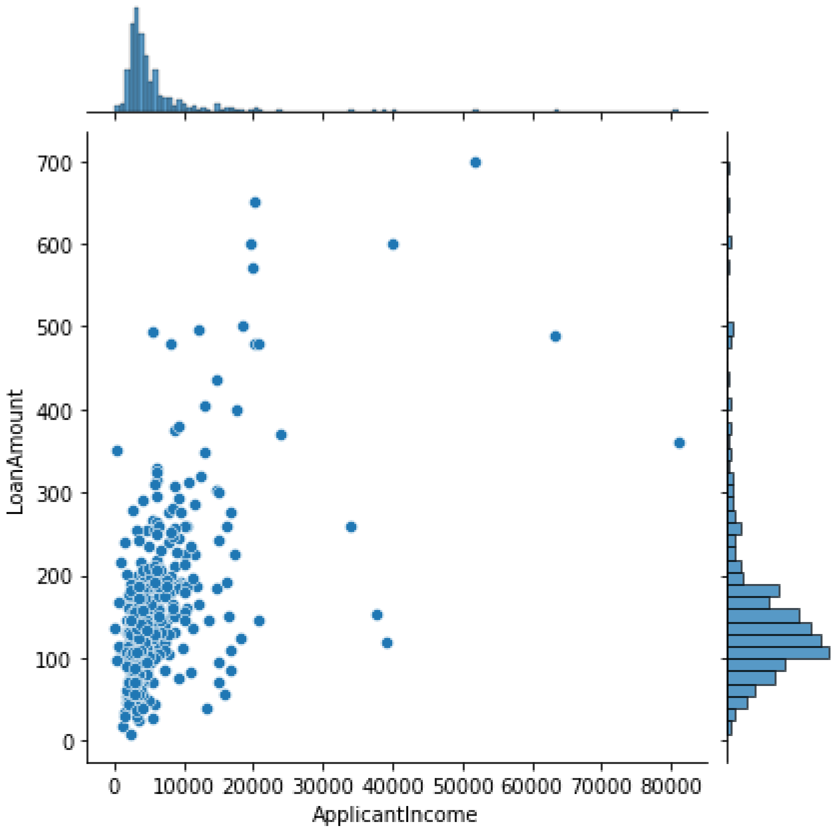
JOINT PLOT: Used for bivariate data

**Is there a relationship between ApplicantIncome and LoanAmount?

In [19]:

```
sns.jointplot(x="ApplicantIncome",y="LoanAmount",data=loan_data,kind="scatter")
```

Out[19]: <seaborn.axisgrid.JointGrid at 0x20b341b4b80>



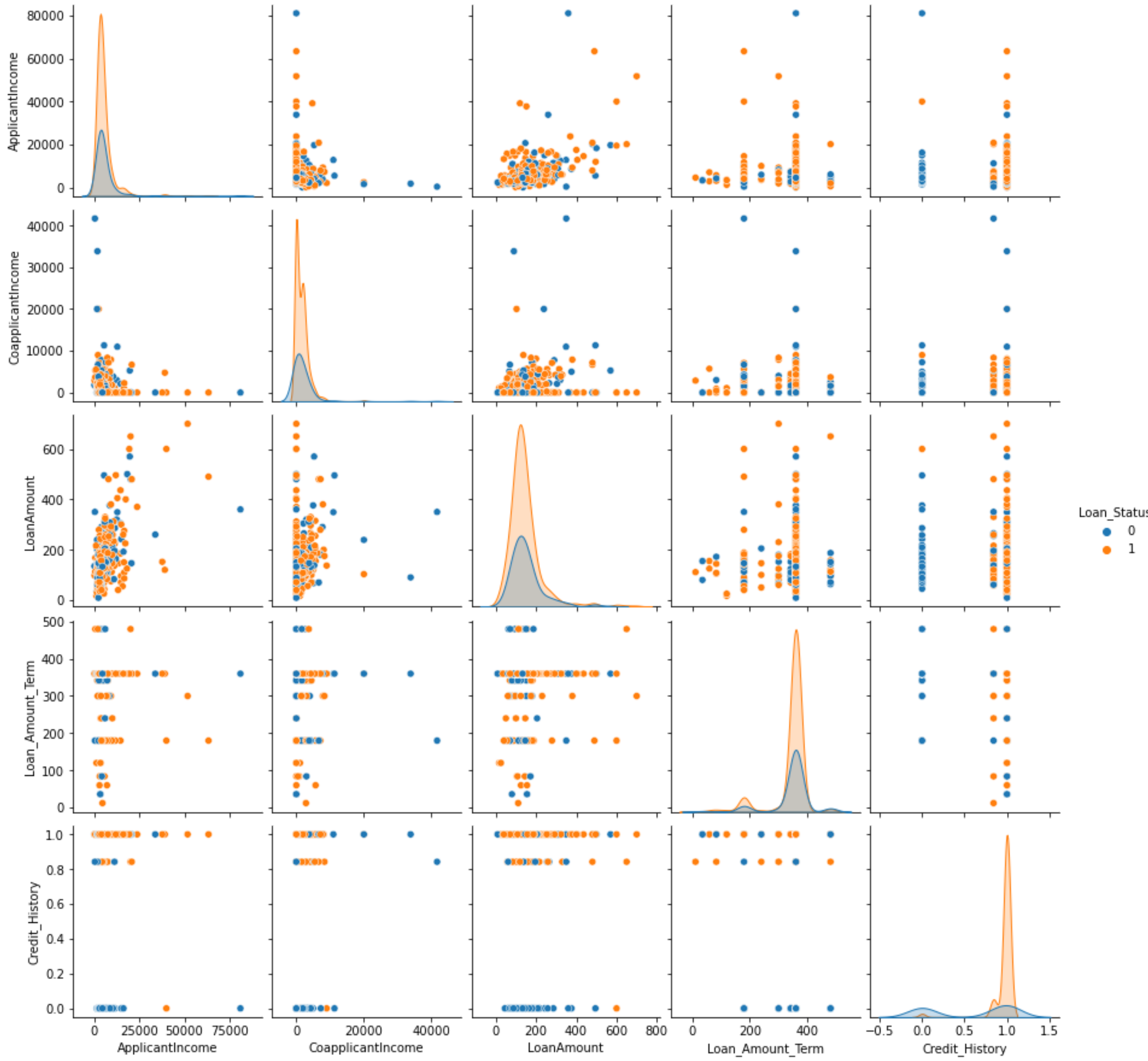
Interpretation: Higher the ApplicantIncome,the LoanAmount is also higher.

PAIR PLOT: This gives combination plot for all numerical columns

In [20]:

```
sns.pairplot(loan_data,hue="Loan_Status")
```

Out[20]: <seaborn.axisgrid.PairGrid at 0x20b3451ebe0>

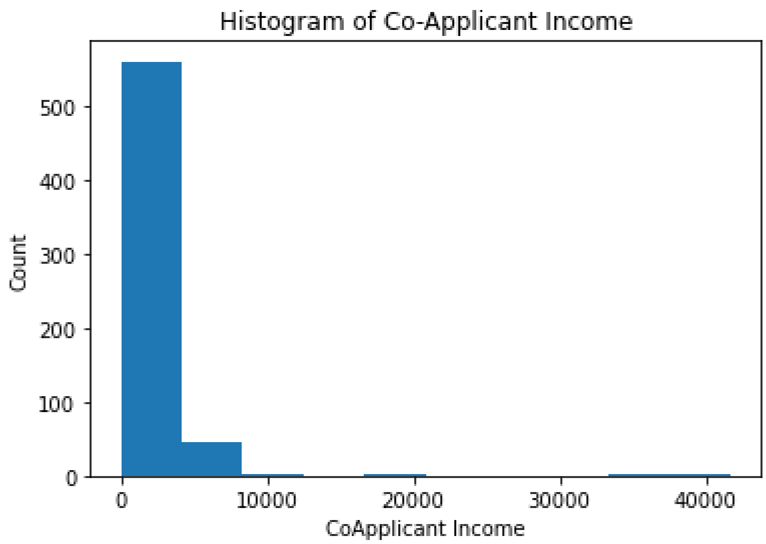


In [21]:
```
import matplotlib.pyplot as plt
```

HISTOGRAM:

**How is CoapplicantIncome distributed?

In [22]:
```
plt.hist(loan_data['CoapplicantIncome'])
plt.title("Histogram of Co-Applicant Income")
plt.xlabel("CoApplicant Income")
plt.ylabel("Count")
plt.show()
```



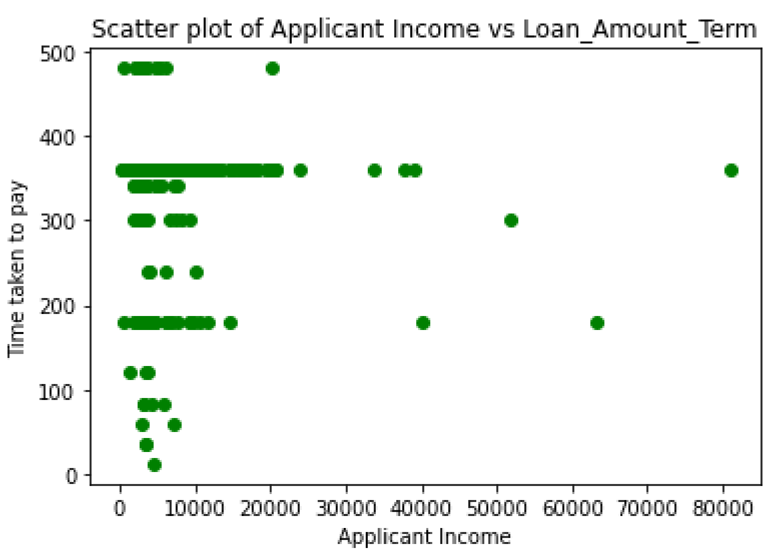

```
Interpretation:
        We can see that more than 500 co-applicants have income in the
range 0-800.
```

SCATTER PLOT:

**What is the relationship between ApplicantIncome and Loan_Amount_Term?

In [23]:

```
plt.scatter(x="ApplicantIncome",y="Loan_Amount_Term",data=loan_data,color="green")
plt.xlabel("Applicant Income")
plt.ylabel("Time taken to pay")
plt.title("Scatter plot of Applicant Income vs Loan_Amount_Term")
plt.show()
```



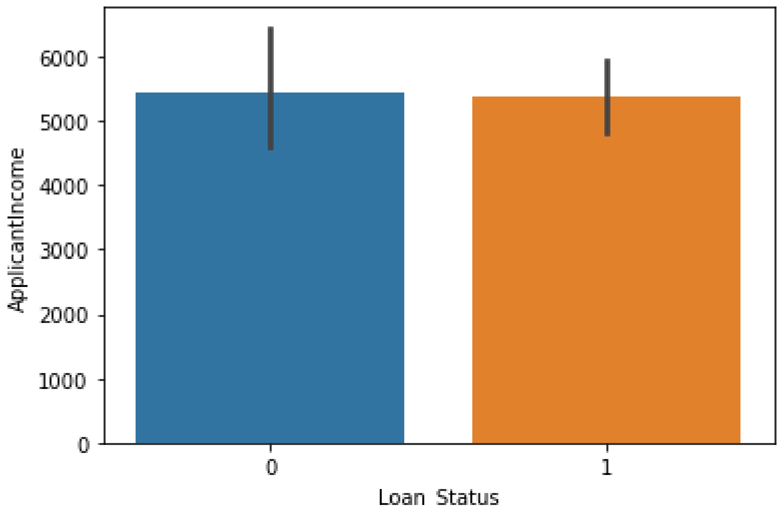

```
Interpretation:
        Lesser the Applicant Income,higher the Loan_Amount_Term
```

CATEGORICAL PLOT: For visualizing categorical columns

BARPLOT:

In [24]:
```
sns.barplot(x="Loan_Status",y="ApplicantIncome",data=loan_data)
```

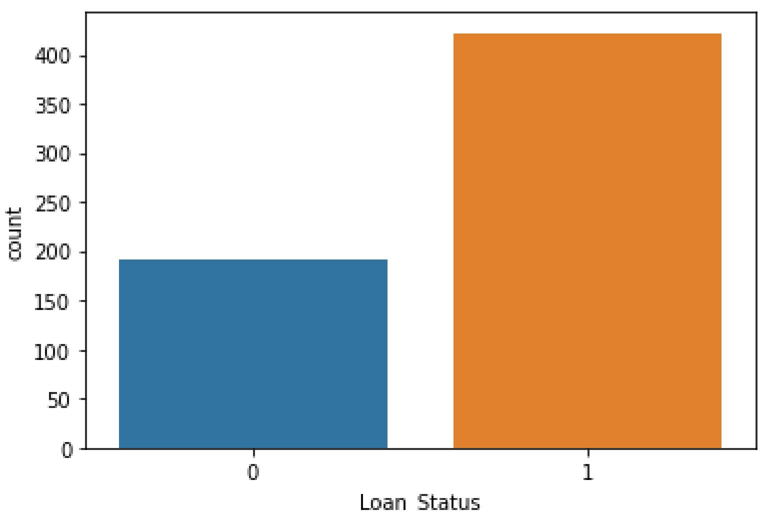
Out[24]: <AxesSubplot:xlabel='Loan_Status', ylabel='ApplicantIncome'>



COUNTPLOT:

In [25]:
```
sns.countplot(x="Loan_Status",data=loan_data)
```

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




```
Interpretation:
     We can see that many persons are eligible to get loan.
```

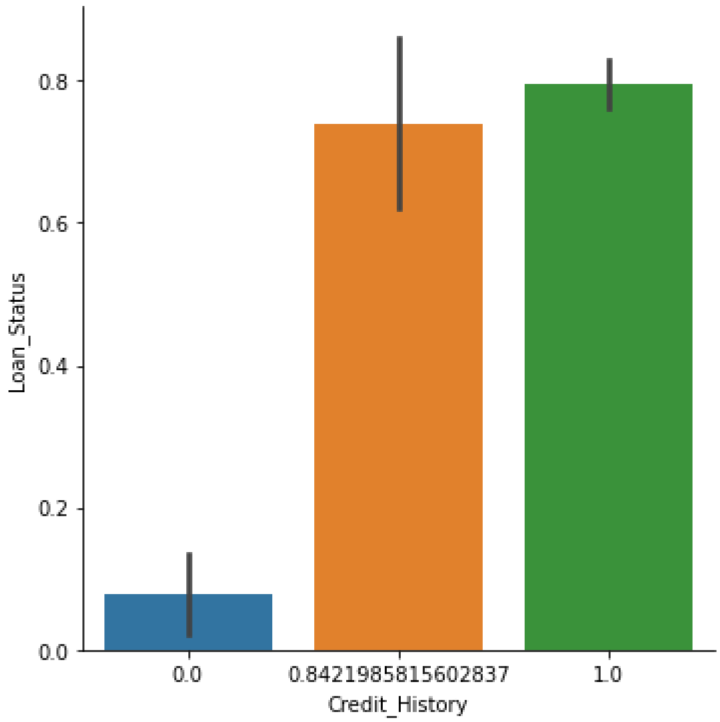
**Factor plot: Factor Plot is used to draw a different types of categorical plot . The default plot that is shown is a point plot, but we can plot other seaborn categorical plots by using of kind parameter, like box plots, violin plots, bar plots, or strip plots

In [26]:
```
sns.factorplot(x="Credit_History",y="Loan_Status",data=loan_data,kind="bar")
```

```
C:\Users\nivet\anaconda\anaconda3\lib\site-packages\seaborn\categorical.py:3714: UserWar
ning: The `factorplot` function has been renamed to `catplot`. The original name will be
removed in a future release. Please update your code. Note that the default `kind` in `f
```

```
actorplot` (`'point'`) has changed `'strip'` in `catplot`.
  warnings.warn(msg)
```

Out[26]: <seaborn.axisgrid.FacetGrid at 0x20b34da9a60>



In [ ]:

```
    Interpretation:
      Mostly when the Credit_History is 1,the Loan_Status is eligible
```

In [ ]:

```
Summary:
    From the distribution plot,we can see that the nearly
    560 persons have income in the range 0-10000.
    From the joint point,we can see that Higher the ApplicantIncome,
    the LoanAmount is also higher.
    From the histogram of CoapplicantIncome,we can see that more than
    500 co-applicants have income in the range 0-800.
    From the factor plot of Credit_History and Loan_Status,we can see that
    Mostly when the Credit_History is 1,the Loan_Status is eligible
    From countplot,we can see that many persons are eligible to get loan.
    From scatter plot,we can see that Lesser the Applicant Income,
    higher the Loan_Amount_Term
```

**What is the mean of ApplicantIncome?

In [11]:

```
loan_data["ApplicantIncome"].mean()
```

Out[11]: 5403.459283387622

**What is the mean of CoapplicantIncome?

In [15]:

```
loan_data["CoapplicantIncome"].mean()
```

Out[15]: 1621.245798027101

**What is the mean of LoanAmount?

In [17]:
```
loan_data["LoanAmount"].mean()
```

Out[17]:
```
146.41216216216213
```

**What is the variance of Loan_Amount?

In [13]:
```
loan_data["LoanAmount"].var()
```

Out[13]:
```
7062.295974604296
```

**What is the standard deviation of Loan_Amount_Term

In [14]:
```
loan_data["Loan_Amount_Term"].std()
```

Out[14]:
```
64.37248862679246
```

**MODEL Approach:

In [ ]:
```
The model type here is to predict whether an Applicant is eligible to get loan or not
    based on ApplicantIncome,CoapplicantIncome,Credit_History,LoanAmount.
    Since we have to predict only whether an applicant is eligible or not to get a loan
    we use binary classification method
    i.e. we use LOGISTIC REGRESSION.
Dependent Variable:
    Loan_Status
Independent Variable:
    ApplicantIncome,CoapplicantIncome,Credit_History,LoanAmount
```

In [ ]:
```
Assumptions for final model:
    The outcome is a binary variable like Y/N OR 1/0.
    There is a linear relationship between the logit of outcome
    and each predictor variables.
```

In [27]:
```
#Identifying dependent and independent variables:
x=loan_data[['ApplicantIncome','CoapplicantIncome','Credit_History','LoanAmount']]
y=loan_data['Loan_Status']
```

**Splitting the data into train and test:

In [28]:
```
from sklearn.model_selection import train_test_split
```

In [29]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=101)
```

In [30]:
```
x_train.shape
```

Out[30]:
```
(491, 4)
```

In [31]:
```
x_test.shape
```

Out[31]: (123, 4)

In [32]:
```
y_test.shape
```

Out[32]: (123,)

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

In [34]:
```
log=LogisticRegression() #log is an object of LogisticRegression
```

Fitting Logistic Regression Model:

In [35]:
```
log.fit(x_train,y_train)
```

Out[35]: LogisticRegression()

Predicting for test data:

In [36]:
```
pred=log.predict(x_test)
```

In [37]:
```
pred
```

Out[37]:
```
array([1, 1, 1, 0, 1, 0, 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, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1,
       1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1,
       0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0], dtype=int64)
```

**MODEL EVALUATION:

In [38]:
```
from sklearn.metrics import confusion_matrix,accuracy_score,precision_score,f1_score,re
```

In [ ]:
```
Confusion matrix:
    A confusion matrix is a table that is used to define the
    performance of a classification algorithm.
```

In [39]:
```
confusion_matrix(y_test,pred)
```

Out[39]:
```
array([[21, 24],
       [ 1, 77]], dtype=int64)
```

In [ ]:
```
Accuracy score:
    Accuracy is the most intuitive performance measure and it is simply a
    ratio of correctly predicted observation
```

```
    to the total observations.
    i.e.accuracy=(tp+tn)/(tp+tn+fn+fp)
```

In [40]:

```
accuracy_score(y_test,pred)
```

Out[40]: 0.7967479674796748

In [ ]:

```
Precision score:
    Greater the precision=>less the number of FP.
    precision=tp/(fp+tp)
```

In [41]:

```
precision_score(y_test,pred)
```

Out[41]: 0.7623762376237624

In [ ]:

```
Recall score:
    Greater the recall score=>less the number of FN.
    recall=tp/(fn+tp)
```

In [42]:

```
recall_score(y_test,pred)
```

Out[42]: 0.9871794871794872

In [ ]:

```
F1 score:
    F1-score is one of the most important evaluation metrics in machine learning.
    It elegantly sums up the predictive performance of a model by combining two
    otherwise competing metrics — precision and recall.
    f1_score=(2*precision*recall)/(precision+recall)
```

In [43]:

```
f1_score(y_test,pred)
```

Out[43]: 0.8603351955307262

**Interpretation:

In [ ]:

```
From the confusion matrix,we can see that maximum values have been predicted correctly.
We get accuracy to be 0.796=>80%(app)=>our model works very well.
We get precision score to be 0.76=>less number of FP
We get recall score to be 0.98=>very less number of FN
We get f1_score to be 0.86=>model works well
```

**Prediction for new data:

In [48]:

```
applicant_income=int(input("Enter the Applicant income(in thousands):"))
coapplicant_income=int(input("Enter the co-applicant income(in thousands):"))
```

```
credit_history=int(input("Enter the credit history:"))
loan_amt=int(input("Enter the loan amount(in thousands):"))
```

```
Enter the Applicant income(in thousands):4328
Enter the co-applicant income(in thousands):1232
Enter the credit history:1
Enter the loan amount(in thousands):453
```

In [49]:

```
pred_new=log.predict([[applicant_income,coapplicant_income,credit_history,loan_amt]])
pred_new
```

Out[49]:

```
array([1], dtype=int64)
```

In [ ]:

```
Conclusion:
    After training and testing the model,we got accuracy to be 80%.
    Using predict() method ,we have predicted the loan status for a new person
    whose income is 4328(in thousands),co-applicant's income is 1232(in thousands),
    credit history is 1 and loan amount is 453(in thousands).
    The model predicts the loan status to be 1=>The person is eligible to get a loan.
```