Task: Determine the eligibility for granting Home loan.

Objectives of this notebook are:

1. To understand the patterns in the data.
2. How to Handle the categorical features.
3. How to deal with missing data.
4. Feature Engineering
5. Finding the most important features while taking the decision of granting a loan application.
6. Label encoding and target encoding -- converting categorical features to numeric features

Load data and libraries

In [1]:
```
import numpy as np
import pandas as pd
from scipy import stats

import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
data = pd.read_csv('loan.csv')
data.shape
```

Out[2]:
```
(614, 13)
```

In [3]:
```
data.columns
```

Out[3]:
```
Index(['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education',
       'Self_Employed', 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History', 'Property_Area', 'Loan_Status'],
      dtype='object')
```

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

Out[4]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coappl
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 [5]:
```
data.dtypes
#object => typically categorical/IDs
#Int64, Float64
```

Out[5]:
```
Loan_ID               object
Gender                object
Married               object
Dependents            object
Education             object
Self_Employed         object
ApplicantIncome        int64
CoapplicantIncome    float64
LoanAmount           float64
Loan_Amount_Term     float64
Credit_History       float64
Property_Area         object
Loan_Status           object
dtype: object
```

In [6]:
```
data['Dependents'].value_counts()
```

Out[6]:
```
0     345
1     102
2     101
3+     51
Name: Dependents, dtype: int64
```

In [7]:
```
# drop loanID column
data = data.drop('Loan_ID',axis = 1)
```

Basic Data Exploration

In [8]:
```
data.describe()
# only numeric features
```

Out[8]:

	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 [9]:
```
# catgeorical features
data.describe(include = ['object'])
```

Out[9]:

	Gender	Married	Dependents	Education	Self_Employed	Property_Area	Loan_Status
count	601	611	599	614	582	614	614
unique	2	2	4	2	2	3	2
top	Male	Yes	0	Graduate	No	Semiurban	Y
freq	489	398	345	480	500	233	422

In [10]:
```
#missing values
data.isna().sum()
```

Out[10]:
```
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 [11]:
```
# catgeorical and numerical columns
cat_cols = data.dtypes =='object'
cat_cols = list(cat_cols[cat_cols].index)

num_cols = data.dtypes !='object'
num_cols = list(num_cols[num_cols].index)
cat_cols.remove('Loan_Status')
```

In [12]:
```
cat_cols
```

Out[12]:
```
['Gender',
 'Married',
 'Dependents',
 'Education',
 'Self_Employed',
 'Property_Area']
```

In [13]:
```
num_cols
```

Out[13]:
```
['ApplicantIncome',
 'CoapplicantIncome',
 'LoanAmount',
 'Loan_Amount_Term',
 'Credit_History']
```

In [14]:
```
data[cat_cols].head()
```

Out[14]:

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

In [15]:
```
data[num_cols].head()
```

Out[15]:

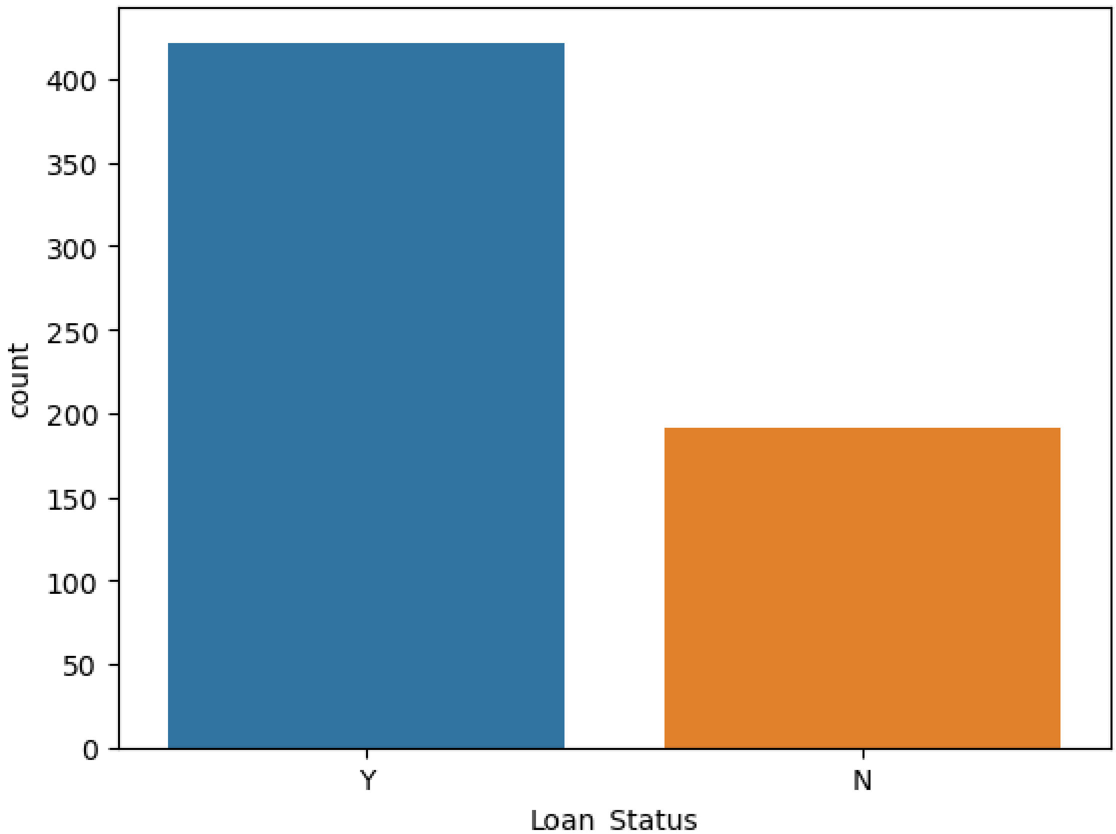
	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
0	5849	0.0	NaN	360.0	1.0
1	4583	1508.0	128.0	360.0	1.0
2	3000	0.0	66.0	360.0	1.0
3	2583	2358.0	120.0	360.0	1.0
4	6000	0.0	141.0	360.0	1.0

Basic Data visualization: Univariate

In [16]:
```
data['Loan_Status'].value_counts()
```

Out[16]:
```
Y    422
N    192
Name: Loan_Status, dtype: int64
```

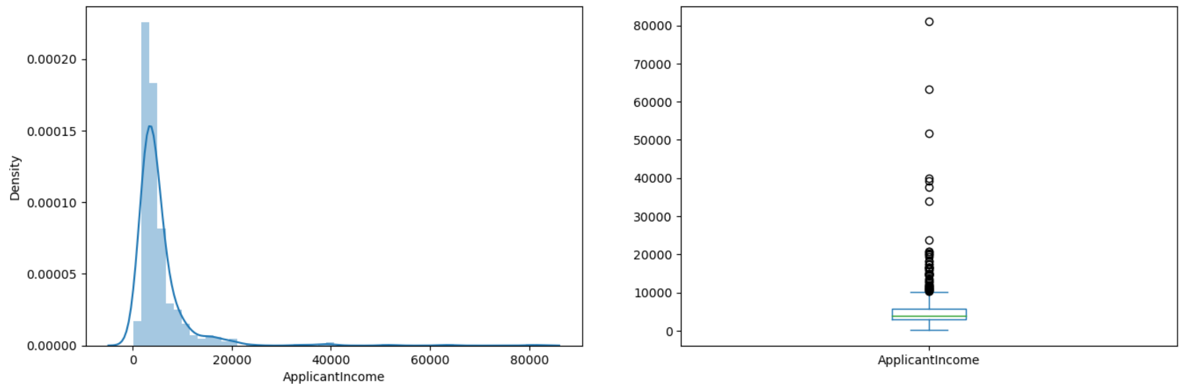
In [17]:
```
#Q: How many loans the company has approved in the past?
sns.countplot(data=data, x='Loan_Status')
plt.show()
```



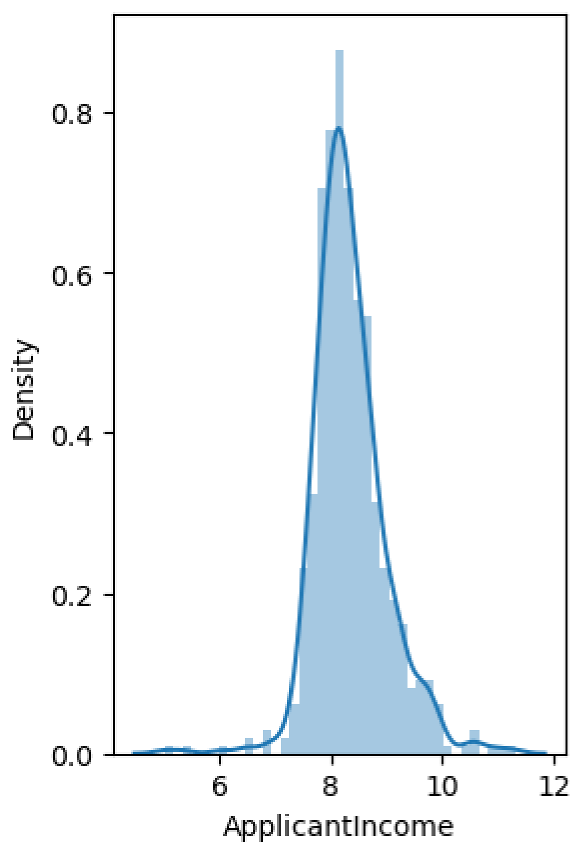
In [18]:
```
target = 'Loan_Status'
data[target].value_counts()

# Imbalanced data
```

Out[18]:
```
Y    422
N    192
Name: Loan_Status, dtype: int64
```

In [19]:
```
#Income of the applicant
plt.subplot(121)
sns.distplot(data["ApplicantIncome"])

plt.subplot(122)
data["ApplicantIncome"].plot.box(figsize=(16,5))
plt.show()
```

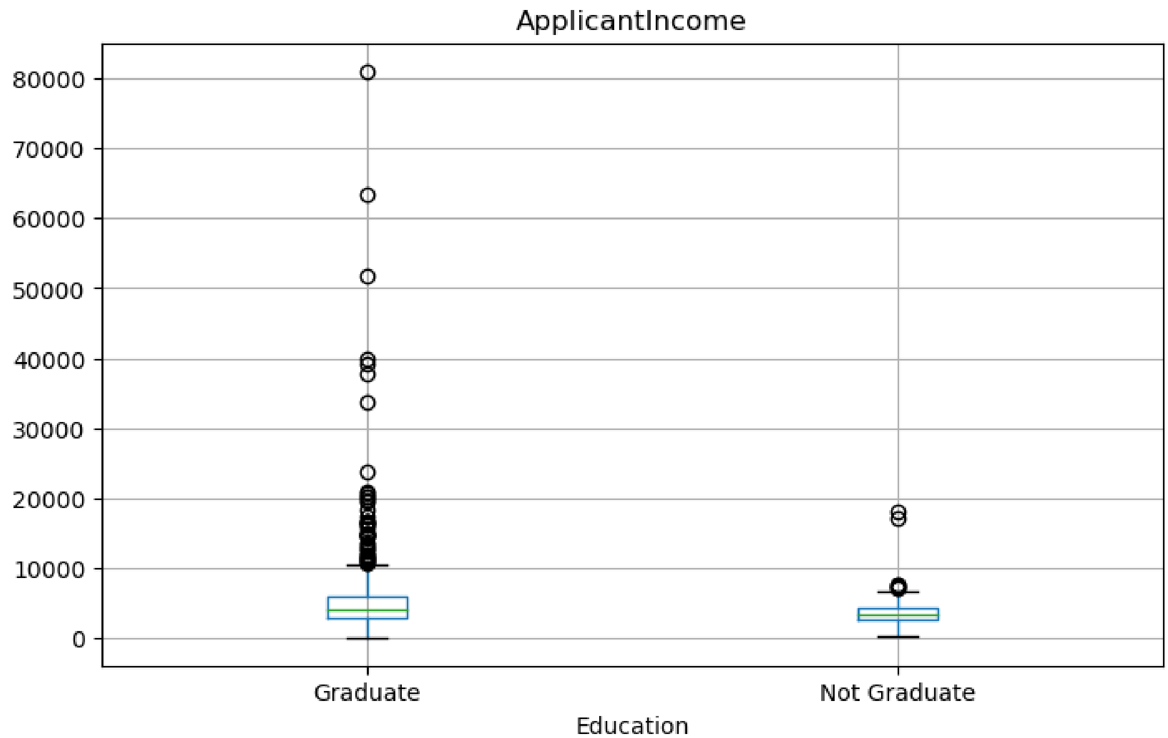


In [20]:
```
plt.subplot(121)
sns.distplot(np.log(data["ApplicantIncome"]))

plt.show()
```

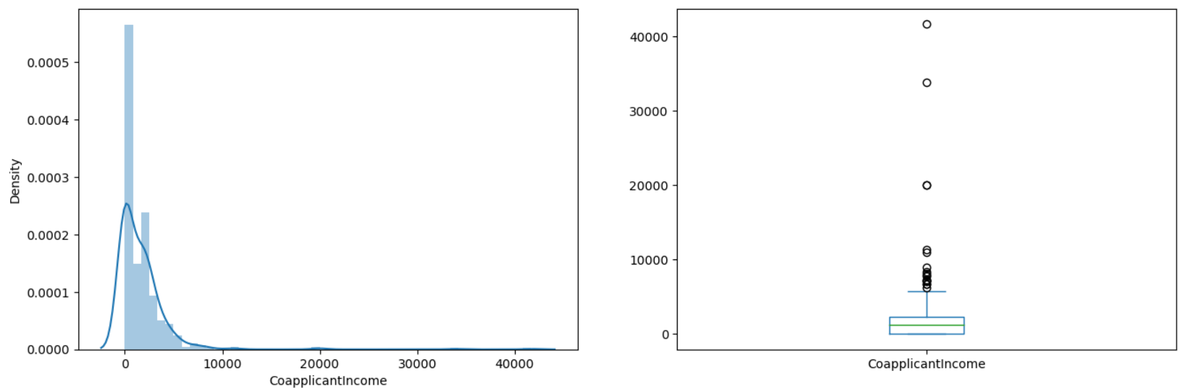


In [21]:
```
#Slice this data by Education
```

In [22]:
```
data.boxplot(column='ApplicantIncome', by="Education", figsize=(8,5))
plt.suptitle("")
plt.show()
```



In [23]:
```
#co-applicant income
plt.subplot(121)
sns.distplot(data["CoapplicantIncome"])

plt.subplot(122)
data["CoapplicantIncome"].plot.box(figsize=(16,5))
plt.show()
```

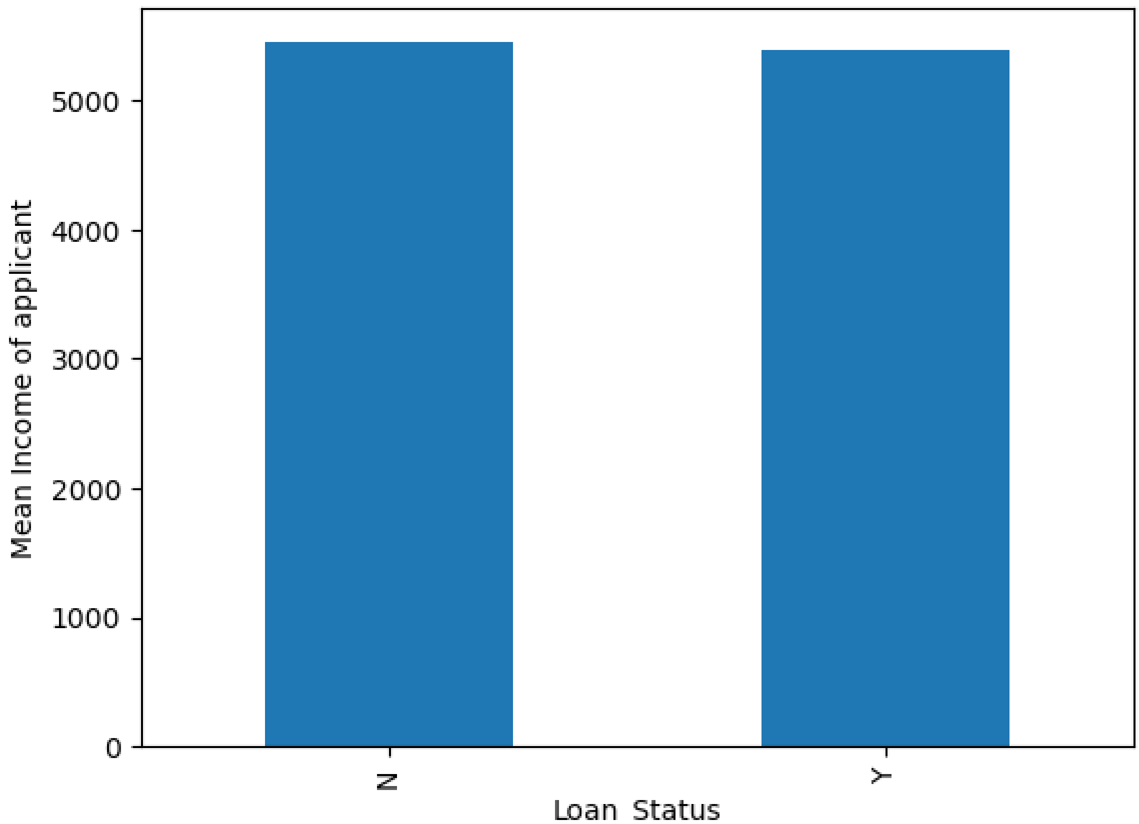


In [24]:
```
#Relation between "Loan_Status" and "Income"
```

In [25]:
```
data.groupby("Loan_Status").mean()['ApplicantIncome']
```

Out[25]:
```
Loan_Status
N    5446.078125
Y    5384.068720
Name: ApplicantIncome, dtype: float64
```

In [26]:
```
data.groupby("Loan_Status").mean()['ApplicantIncome'].plot.bar()
plt.ylabel("Mean Income of applicant")
plt.show()
```



Simple Feature Engineering

In [27]:
```
# Feature binning: income
bins=[0,2500,4000,6000, 8000, 10000, 20000, 40000, 81000]
group=['Low','Average','medium', 'H1', 'h2', 'h3', 'h4' , 'Very high']
data['Income_bin']= pd.cut(data['ApplicantIncome'],bins,labels=group)
```

In [28]:
```
data.head()
```

Out[28]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome
0	Male	No	0	Graduate	No	5849	0.(
1	Male	Yes	1	Graduate	No	4583	1508.(
2	Male	Yes	0	Graduate	Yes	3000	0.(
3	Male	Yes	0	Not Graduate	No	2583	2358.(
4	Male	No	0	Graduate	No	6000	0.(

Incomes

In [29]:
```
#observed
pd.crosstab(data["Income_bin"],data["Loan_Status"])
```

Out[29]:

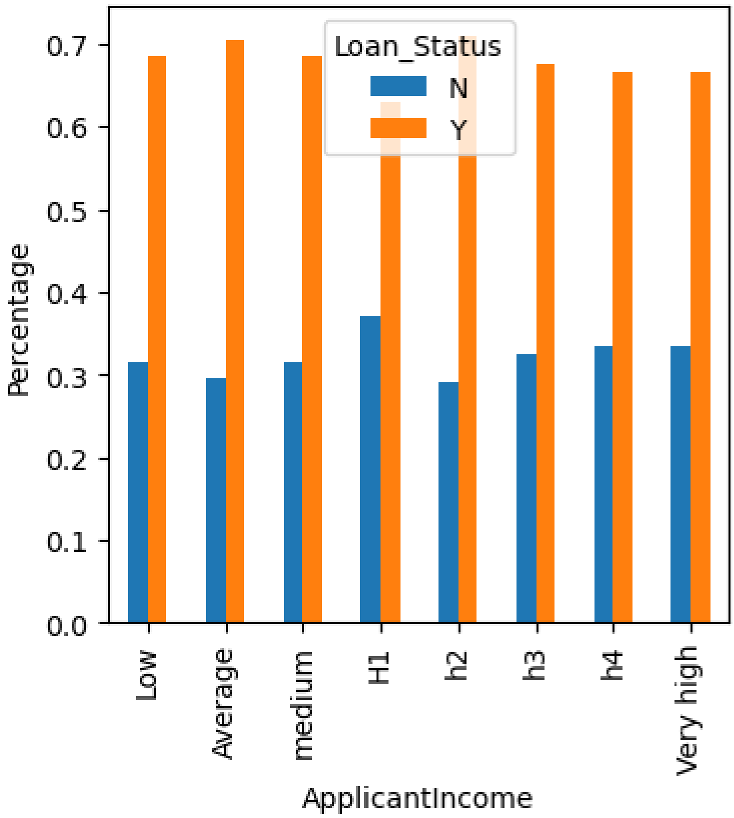
Loan_Status	N	Y
Income_bin		
Low	34	74
Average	67	159
medium	45	98
H1	20	34
h2	9	22
h3	13	27
h4	3	6
Very high	1	2

In [30]:
```
from scipy.stats import chi2_contingency
val = pd.crosstab(index=data["Income_bin"], columns=data["Loan_Status"]).value
print(val)
chi2_contingency(val) # chi_stat, p_value, df, expected_values =
```

```
[[ 34  74]
 [ 67 159]
 [ 45  98]
 [ 20  34]
 [  9  22]
 [ 13  27]
 [  3   6]
 [  1   2]]
```

Out[30]:
```
(1.2420001711303135,
 0.9899274842922701,
 7,
 array([[ 33.77198697,  74.22801303],
        [ 70.67100977, 155.32899023],
        [ 44.71661238,  98.28338762],
        [ 16.88599349,  37.11400651],
        [  9.69381107,  21.30618893],
        [ 12.50814332,  27.49185668],
        [  2.81433225,   6.18566775],
        [  0.93811075,   2.06188925]]))
```

In [31]:
```
Income_bin = pd.crosstab(data["Income_bin"],data["Loan_Status"])

pd.crosstab(data["Income_bin"],data["Loan_Status"], normalize="index").plot(ki
plt.xlabel("ApplicantIncome")
plt.ylabel("Percentage")
plt.show()

#It can be inferred that Applicant income does not affect the chances of loan
```



In [32]:
```
#co-appplicant income
bins=[0,1000,3000,42000]
group =['Low','Average','High']
data['CoapplicantIncome_bin']=pd.cut(data["CoapplicantIncome"],bins,labels=grc
```

In [33]:
```
pd.crosstab(data["CoapplicantIncome_bin"],data["Loan_Status"])
```

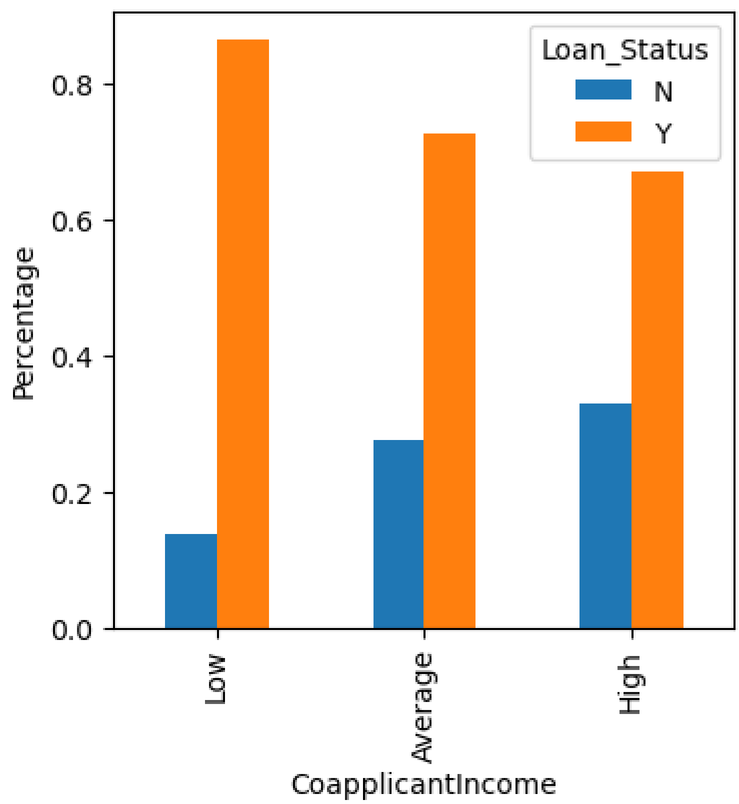
Out[33]:

Loan_Status	N	Y
CoapplicantIncome_bin		
Low	3	19
Average	61	161
High	32	65

In [34]:
```
vals = pd.crosstab(data["CoapplicantIncome_bin"],data["Loan_Status"]).values
chi2_contingency(vals)
```

Out[34]:
```
(3.4640082184540937,
 0.17692946843764604,
 2,
 array([[  6.19354839,  15.80645161],
        [ 62.49853372, 159.50146628],
        [ 27.30791789,  69.69208211]]))
```

In [35]:
```
CoapplicantIncome_Bin = pd.crosstab(data["CoapplicantIncome_bin"],data["Loan_S
pd.crosstab(data["CoapplicantIncome_bin"],data["Loan_Status"], normalize="inde
plt.xlabel("CoapplicantIncome")
plt.ylabel("Percentage")
plt.show()

## What's the problem here? Why co-applicant having low income is getting maxi
```



In [36]:
```
data['CoapplicantIncome'].value_counts().head()
```

Out[36]:
```
0.0       273
2500.0      5
2083.0      5
1666.0      5
2250.0      3
Name: CoapplicantIncome, dtype: int64
```

In [37]:
```
# New feature:  total household income
data["TotalIncome"] = data["ApplicantIncome"] + data["CoapplicantIncome"]
```

In [38]:
```
bins = [0,3000,5000,8000,81000]
group = ['Low','Average','High','Very High']
data["TotalIncome_bin"] = pd.cut(data["TotalIncome"],bins,labels=group)
```

In [39]:
```
pd.crosstab(data["TotalIncome_bin"], data["Loan_Status"])
```

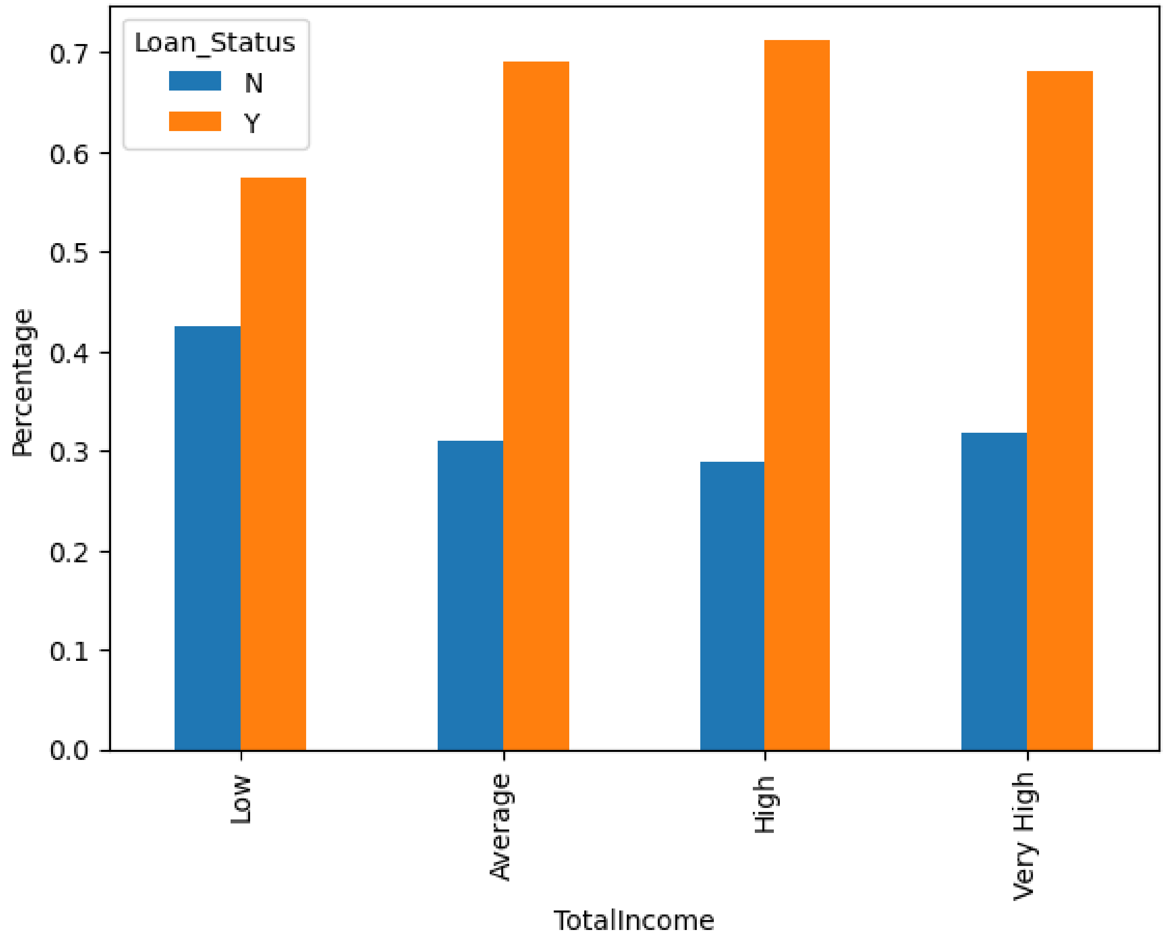
Out[39]:

Loan_Status	N	Y
TotalIncome_bin		
Low	20	27
Average	69	154
High	61	151
Very High	42	90

In [40]:
```
vals = pd.crosstab(data["TotalIncome_bin"], data["Loan_Status"]).values
chi2_contingency(vals)
```

Out[40]:
```
(3.428480885250809,
 0.3301570564076713,
 3,
 array([[ 14.6970684 ,  32.3029316 ],
        [ 69.73289902, 153.26710098],
        [ 66.29315961, 145.70684039],
        [ 41.27687296,  90.72312704]]))
```

In [41]:
```
TotalIncome = pd.crosstab(data["TotalIncome_bin"],data["Loan_Status"])
pd.crosstab(data["TotalIncome_bin"],data["Loan_Status"], normalize="index").pl
plt.xlabel("TotalIncome")
plt.ylabel("Percentage")
plt.show()

# Observation: We can see that Proportion of loans getting approved for
# applicants having low Total_Income is very less as compared to that of appli
# with Average, High and Very High Income.
```



In [42]:
```
data = data.drop(["Income_bin","CoapplicantIncome_bin","TotalIncome_bin"],axis
```

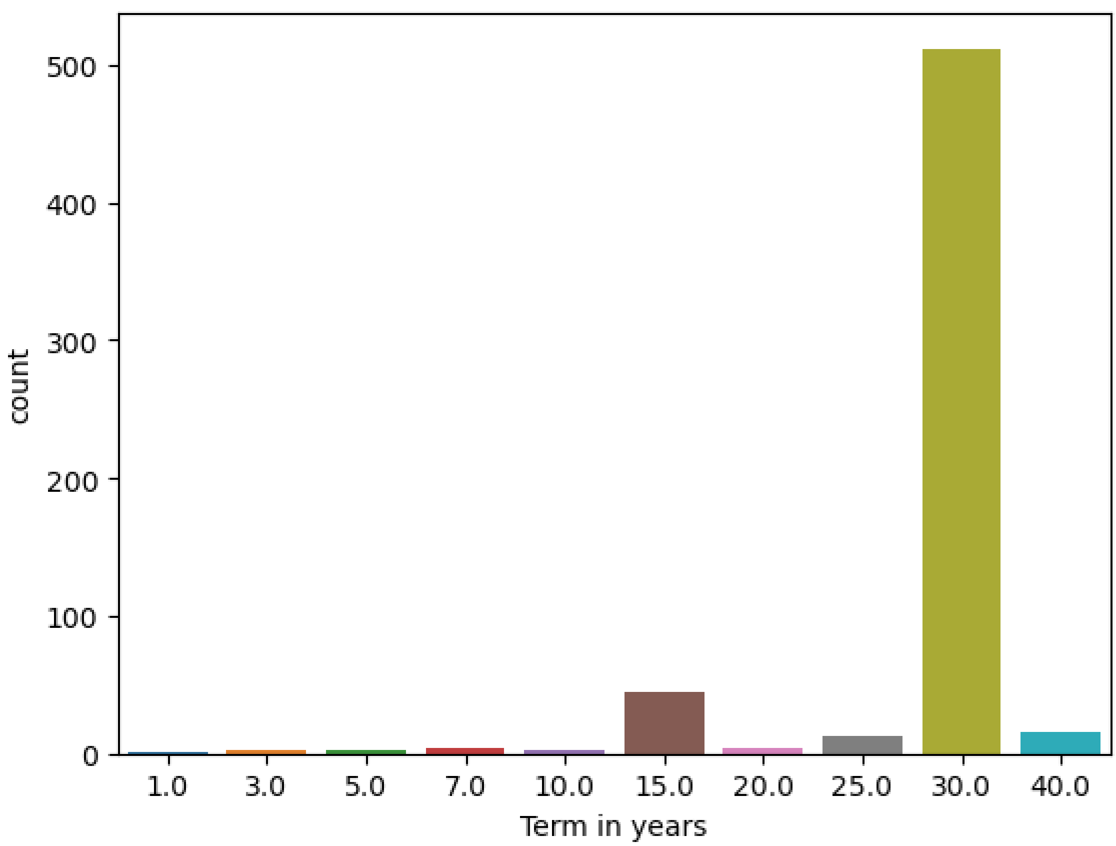
Loan Amount and Loan Term

In [43]:
```
data['Loan_Amount_Term'].value_counts()
```

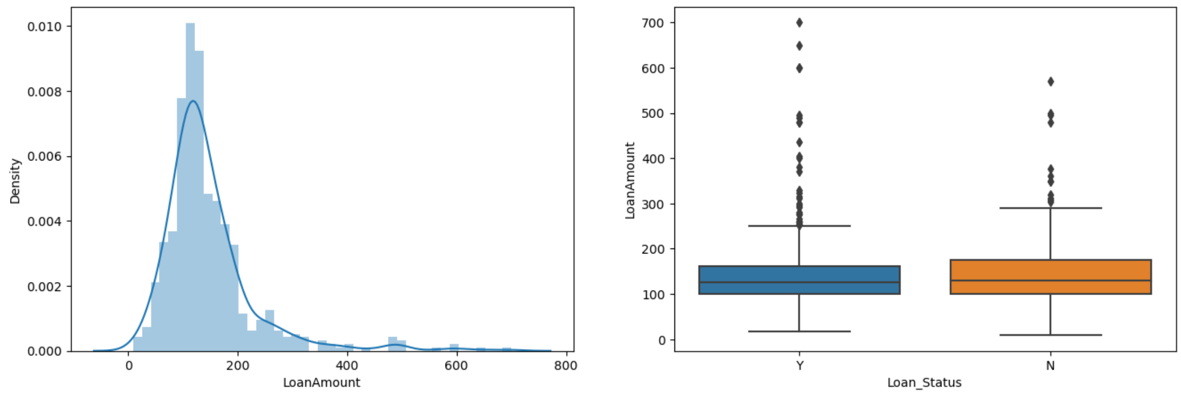
Out[43]:
```
360.0    512
180.0     44
480.0     15
300.0     13
240.0      4
84.0       4
120.0      3
60.0       2
36.0       2
12.0       1
Name: Loan_Amount_Term, dtype: int64
```

In [44]:
```
data['Loan_Amount_Term'] = (data['Loan_Amount_Term']/12).astype('float')
```

In [45]:
```
sns.countplot(x='Loan_Amount_Term', data=data)
plt.xlabel("Term in years")
plt.show()
# Observation: We can clearly see that more than 90% of the loans were applied
```

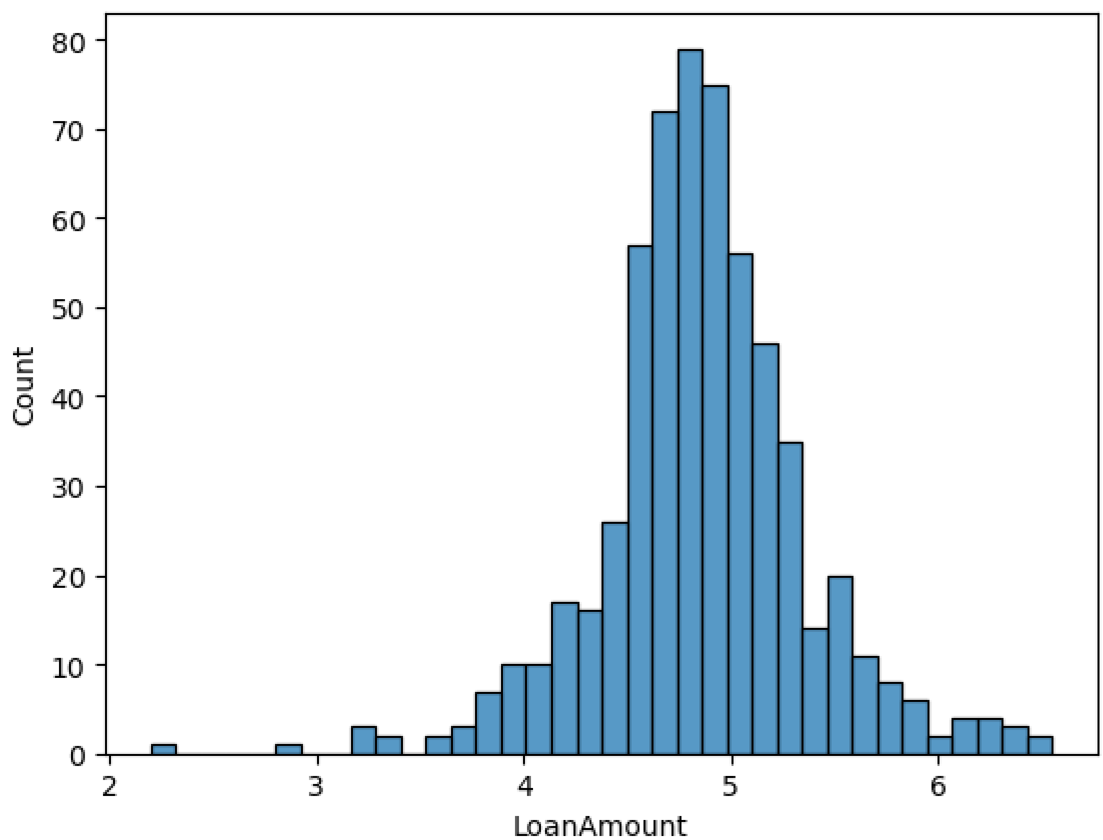


In [46]:
```
### Distribution of "LoanAmount" variable :

plt.figure(figsize=(16,5))
plt.subplot(121)
sns.distplot(data['LoanAmount']);

plt.subplot(122)
sns.boxplot(data=data, x='Loan_Status', y = 'LoanAmount')

plt.show()
```



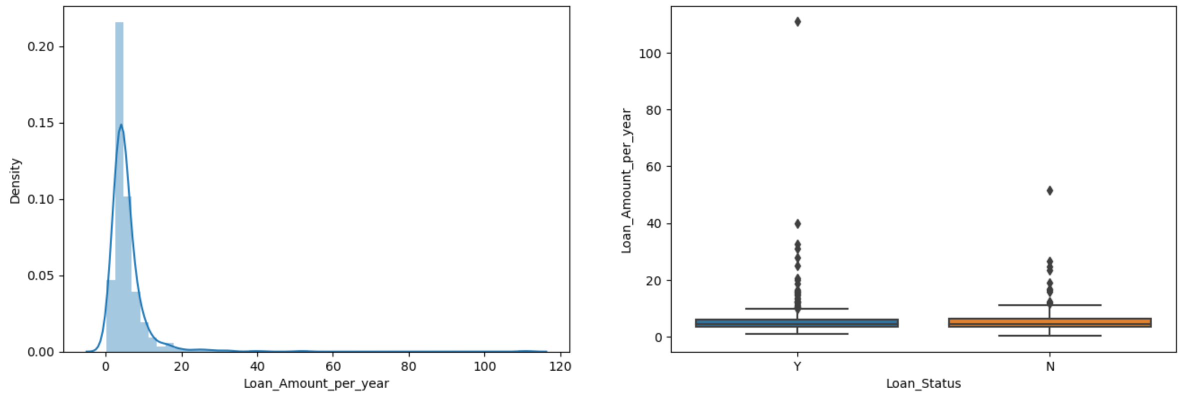
In [47]:
```
# temp
sns.histplot(np.log(data['LoanAmount']))
```

Out[47]: <AxesSubplot:xlabel='LoanAmount', ylabel='Count'>

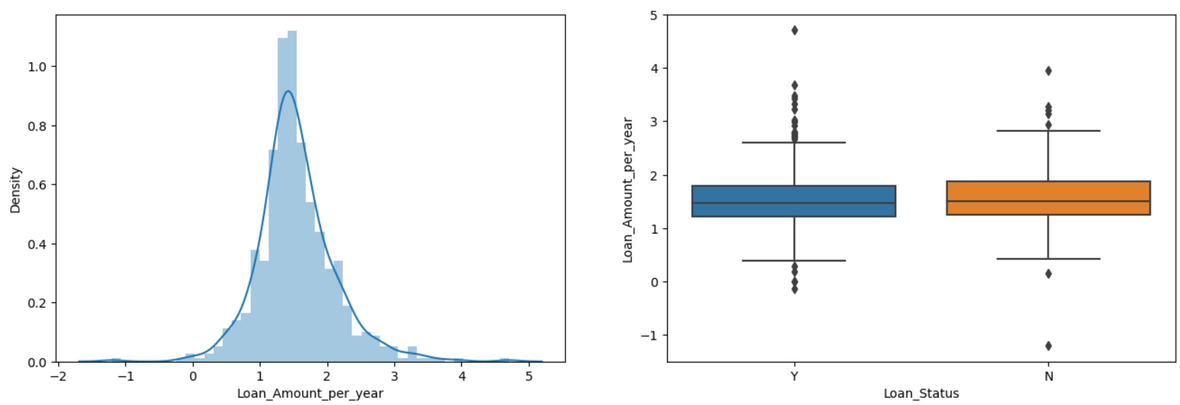


In [48]:
```
# Approximate calc: ignoring interest rates as we dont know that.
data['Loan_Amount_per_year']  = data['LoanAmount']/data['Loan_Amount_Term']
```

In [49]:
```
plt.figure(figsize=(16,5))
plt.subplot(121)
sns.distplot(data['Loan_Amount_per_year']);

plt.subplot(122)
sns.boxplot(data=data, x='Loan_Status', y = 'Loan_Amount_per_year')

plt.show()
```



In [50]:
```
# log transform
plt.figure(figsize=(16,5))
plt.subplot(121)
log_loanAmount = np.log(data['Loan_Amount_per_year'])
sns.distplot(log_loanAmount)

plt.subplot(122)
sns.boxplot(data=data, x='Loan_Status', y = log_loanAmount)

plt.show()
```



In [51]:
```
# Feature : Calculate the EMI based on the Loan Amount Per year.
data['EMI'] = data['Loan_Amount_per_year']*1000/12
```

In [52]:
```
#Feature : Able_to_pay_EMI
data['Able_to_pay_EMI'] = (data['TotalIncome']*0.1 > data['EMI']).astype('int'
```

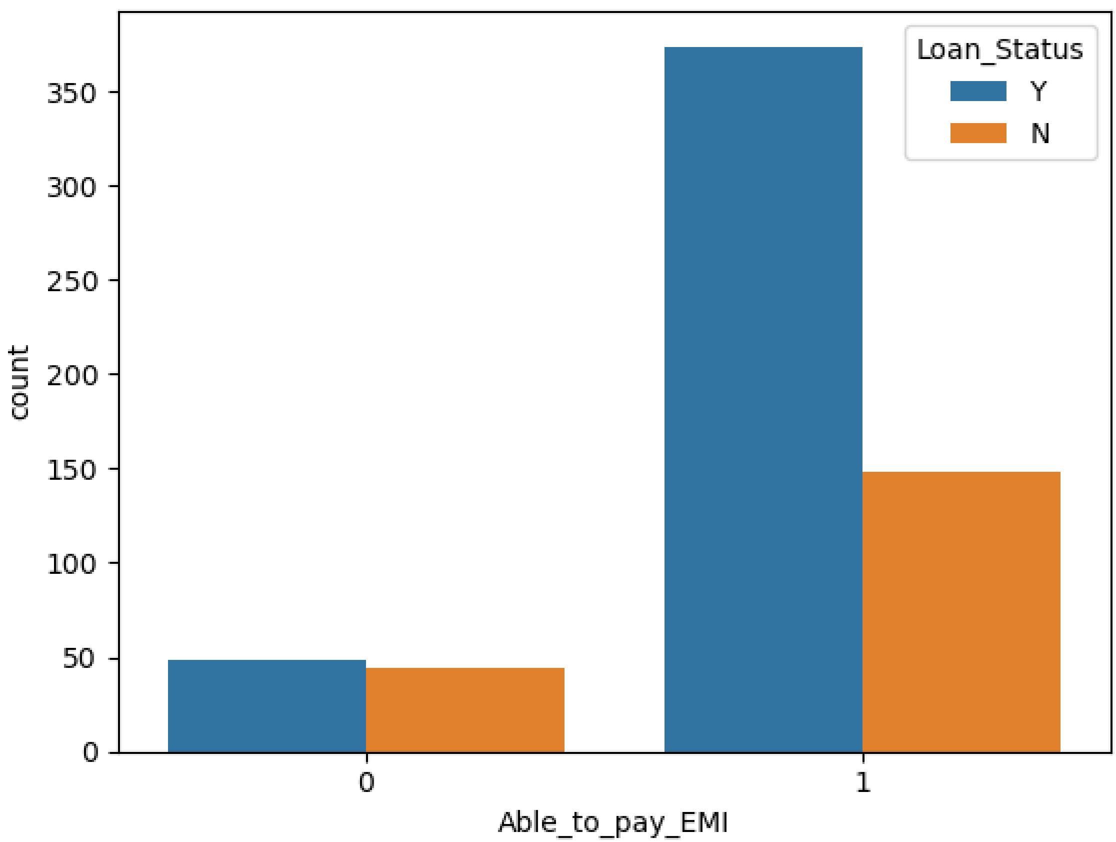
In [53]:
```
data.head()
```

Out[53]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome
0	Male	No	0	Graduate	No	5849	0.(
1	Male	Yes	1	Graduate	No	4583	1508.(
2	Male	Yes	0	Graduate	Yes	3000	0.(
3	Male	Yes	0	Not Graduate	No	2583	2358.(
4	Male	No	0	Graduate	No	6000	0.(

In [54]:
```
sns.countplot(x='Able_to_pay_EMI', data = data, hue = 'Loan_Status')
#Observation:
###There is 50% chance that you may get the loan approved if you cannot pay th
###But there, is a 72% chance that you may get the loan approved if you can pa
```

Out[54]: <AxesSubplot:xlabel='Able_to_pay_EMI', ylabel='count'>



In [99]:
```
vals = pd.crosstab(data['Able_to_pay_EMI'], data['Loan_Status'])
vals
```

Out[99]:

Loan_Status	0	1
Able_to_pay_EMI		
0	44	48
1	148	374

In [100]:
```
chi2_contingency(pd.crosstab(data['Able_to_pay_EMI'], data['Loan_Status']))
# Low p-value implies there is a relation between "Able to pay EMI" feature an
```

Out[100]:
```
(12.909621328812786,
 0.0003268974206671644,
 1,
 array([[ 28.76872964,  63.23127036],
        [163.23127036, 358.76872964]]))
```

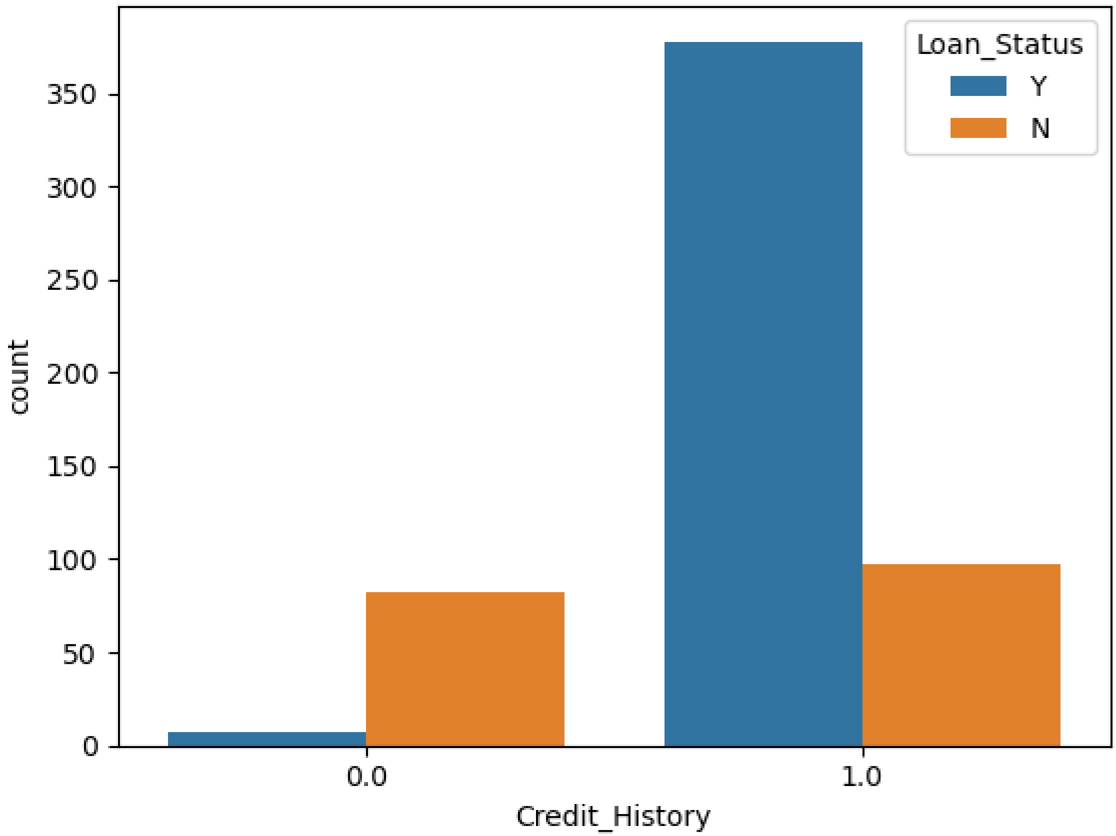
Credit History vs Loan Approval

In [59]:
```
data['Credit_History'].value_counts()
```

Out[59]:
```
1.0    475
0.0     89
Name: Credit_History, dtype: int64
```

In [60]:
```
sns.countplot(data =data, x = 'Credit_History', hue = 'Loan_Status')
#Observation:
## We can clearly see that the approval rate is 80% if your credit history is
## Hence this is the most important question that can be considered.
```

Out[60]: <AxesSubplot:xlabel='Credit_History', ylabel='count'>



Dependents and Loan approval

In [55]:
```
data['Dependents'].value_counts()
```

Out[55]:
```
0     345
1     102
2     101
3+     51
Name: Dependents, dtype: int64
```

In [56]:
```
data['Dependents'].replace('3+',3,inplace=True)
```

In [57]:
```
data['Dependents'] = data['Dependents'].astype('float')
```

In [58]:
```
data.dtypes
```

Out[58]:
```
Gender                    object
Married                   object
Dependents               float64
Education                 object
Self_Employed             object
ApplicantIncome            int64
CoapplicantIncome        float64
LoanAmount               float64
Loan_Amount_Term         float64
Credit_History           float64
Property_Area             object
Loan_Status               object
TotalIncome              float64
Loan_Amount_per_year     float64
EMI                      float64
Able_to_pay_EMI            int64
dtype: object
```

Missing value

In [61]:
```
data.isna().sum()
```

Out[61]:
```
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
TotalIncome               0
Loan_Amount_per_year     36
EMI                      36
Able_to_pay_EMI           0
dtype: int64
```

In [62]:
```
data["Credit_History"].value_counts()
```

Out[62]:
```
1.0    475
0.0     89
Name: Credit_History, dtype: int64
```

In [63]:
```
data['Credit_History'] = data['Credit_History'].fillna(2)
```

In [64]:
```
data["Credit_History"].value_counts()
```

Out[64]:
```
1.0    475
0.0     89
2.0     50
Name: Credit_History, dtype: int64
```

In [65]:
```
data.Self_Employed.unique()
```

Out[65]:
```
array(['No', 'Yes', nan], dtype=object)
```

In [66]:
```
data['Self_Employed'] = data['Self_Employed'].fillna('Other')
```

Imputation from scikit learn

In [67]:
```
from sklearn.impute import SimpleImputer
```

In [103]:
```
vals = pd.DataFrame([10, 20, 10, 15, 17, 18, 21, np.nan])
```

In [104]:
```
si = SimpleImputer(strategy="median")
```

In [105]:
```
si.fit_transform(vals)
```

Out[105]:
```
array([[10.],
       [20.],
       [10.],
       [15.],
       [17.],
       [18.],
       [21.],
       [17.]])
```

In [106]:
```
si = SimpleImputer(strategy="constant", fill_value=200)
```

In [107]:
```
si.fit_transform(vals)
```

Out[107]:
```
array([[ 10.],
       [ 20.],
       [ 10.],
       [ 15.],
       [ 17.],
       [ 18.],
       [ 21.],
       [200.]])
```

In [71]:
```
num_missing = ['EMI', 'Loan_Amount_per_year',  'LoanAmount',  'Loan_Amount_Ter

median_imputer = SimpleImputer(strategy = 'median')
for col in num_missing:
    data[col] = pd.DataFrame(median_imputer.fit_transform(pd.DataFrame(data[cc
```

In [72]:
```
# Highest Freq imputation for some categorical columns. # Prefer "other"?
cat_missing = ['Gender', 'Married','Dependents']
freq_imputer = SimpleImputer(strategy = 'most_frequent') # mode
for col in cat_missing:
    data[col] = pd.DataFrame(freq_imputer.fit_transform(pd.DataFrame(data[col]
```

Categorical to Numeric

In [73]:
```
s = (data.dtypes == 'object')
object_cols = list(s[s].index)
object_cols
```

Out[73]:
```
['Gender',
 'Married',
 'Education',
 'Self_Employed',
 'Property_Area',
 'Loan_Status']
```

In [74]:
```
# Loan_Status
col='Loan_Status'
data[col].value_counts()
```

Out[74]:
```
Y    422
N    192
Name: Loan_Status, dtype: int64
```

Label Encoding

In [75]:
```
from sklearn.preprocessing import LabelEncoder # when we have 2 values ("yes a
label_encoder = LabelEncoder()
data[col] = label_encoder.fit_transform(data[col])
data[col].value_counts()
```

Out[75]:
```
1    422
0    192
Name: Loan_Status, dtype: int64
```

In [76]:
```
#Gender
data['Gender'].value_counts()
```

Out[76]:
```
Male      502
Female    112
Name: Gender, dtype: int64
```

In [77]:
```
label_encoder = LabelEncoder()
col='Gender'
data[col] = label_encoder.fit_transform(data[col])
data[col].value_counts()
```

Out[77]:
```
1    502
0    112
Name: Gender, dtype: int64
```

In [78]:
```
data['Married'].value_counts()
```

Out[78]:
```
Yes    401
No     213
Name: Married, dtype: int64
```

In [79]:
```
label_encoder = LabelEncoder()
col='Married'
data[col] = label_encoder.fit_transform(data[col])
data[col].value_counts()
```

Out[79]:
```
1    401
0    213
Name: Married, dtype: int64
```

Target Encoding

In [80]:
```
# !pip install category_encoders
```

In [81]:
```
from category_encoders import TargetEncoder
```

In [82]:
```
col='Property_Area'
data[col].value_counts()
```

Out[82]:
```
Semiurban    233
Urban        202
Rural        179
Name: Property_Area, dtype: int64
```

In [83]:
```
te = TargetEncoder()
data[col] = te.fit_transform(data[col], data['Loan_Status']) # P("Y" | urban)
```

In [84]:
```
data[col].value_counts()
```

Out[84]:
```
0.768240    233
0.658416    202
0.614525    179
Name: Property_Area, dtype: int64
```

In [85]:
```
pd.crosstab(data["Property_Area"], data["Loan_Status"], normalize="index")
```

Out[85]:

Loan_Status	0	1
Property_Area		
0.614525	0.385475	0.614525
0.658416	0.341584	0.658416
0.768240	0.231760	0.768240

In [86]:
```
col='Education'
data[col].value_counts()
```

Out[86]:
```
Graduate        480
Not Graduate    134
Name: Education, dtype: int64
```

In [87]:
```
label_encoder = LabelEncoder()
data[col] = label_encoder.fit_transform(data[col])
data[col].value_counts()
```

Out[87]:
```
0    480
1    134
Name: Education, dtype: int64
```

In [88]:
```
col='Self_Employed'
data[col].value_counts()
```

Out[88]:
```
No       500
Yes       82
Other     32
Name: Self_Employed, dtype: int64
```

In [89]:
```
te = TargetEncoder()
data[col] = te.fit_transform(data[col], data['Loan_Status'])
data[col].value_counts()
```

Out[89]:
```
0.686000    500
0.682936     82
0.711469     32
Name: Self_Employed, dtype: int64
```

In [90]:
```
s = (data.dtypes == 'object')
object_cols = list(s[s].index)
object_cols
# No more non numeric cols.
```

Out[90]:
```
[]
```

In [91]:
```
# Now, all the features are numerical type
data.dtypes
```

Out[91]:
```
Gender                    int64
Married                   int64
Dependents              float64
Education                 int64
Self_Employed           float64
ApplicantIncome           int64
CoapplicantIncome       float64
LoanAmount              float64
Loan_Amount_Term        float64
Credit_History          float64
Property_Area           float64
Loan_Status               int64
TotalIncome             float64
Loan_Amount_per_year    float64
EMI                     float64
Able_to_pay_EMI           int64
dtype: object
```

In [ ]: