Importing Necessary Packages

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
get_ipython().run_line_magic('matplotlib', 'inline')
```

Loading Data Into Dataframe

In [2]:
```
df = pd.read_csv(r'C:\Users\falco\OneDrive\Desktop\PROJECTS\PYTHON PROJECTS\MY PROJECT
```

Visualizins Top 5 Rows

In [3]:
```
df.head()
```

Out[3]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantI
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	

Shape Of Dataframe

In [4]:
```
df.shape
```

Out[4]:
```
(614, 13)
```

Info On Dataframe

In [5]:
```
df.info
```

Out[5]:
```
<bound method DataFrame.info of       Loan_ID  Gender Married Dependents     Educatio
n Self_Employed  \
0    LP001002    Male      No          0      Graduate            No
1    LP001003    Male     Yes          1      Graduate            No
2    LP001005    Male     Yes          0      Graduate           Yes
3    LP001006    Male     Yes          0  Not Graduate            No
4    LP001008    Male      No          0      Graduate            No
..        ...     ...     ...        ...           ...           ...
609  LP002978  Female      No          0      Graduate            No
610  LP002979    Male     Yes         3+      Graduate            No
611  LP002983    Male     Yes          1      Graduate            No
612  LP002984    Male     Yes          2      Graduate            No
613  LP002990  Female      No          0      Graduate           Yes

     ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
0               5849                0.0         NaN             360.0
1               4583             1508.0       128.0             360.0
2               3000                0.0        66.0             360.0
3               2583             2358.0       120.0             360.0
4               6000                0.0       141.0             360.0
..               ...                ...         ...               ...
609             2900                0.0        71.0             360.0
610             4106                0.0        40.0             180.0
611             8072              240.0       253.0             360.0
612             7583                0.0       187.0             360.0
613             4583                0.0       133.0             360.0

     Credit_History Property_Area Loan_Status
0               1.0         Urban           Y
1               1.0         Rural           N
2               1.0         Urban           Y
3               1.0         Urban           Y
4               1.0         Urban           Y
..              ...           ...         ...
609             1.0         Rural           Y
610             1.0         Rural           Y
611             1.0         Urban           Y
612             1.0         Urban           Y
613             0.0     Semiurban           N

[614 rows x 13 columns]>
```

Description On Data

In [6]:
```
df.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

Columns Of Dataframe

In [8]:
```
df.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')
```

Null Values In Data

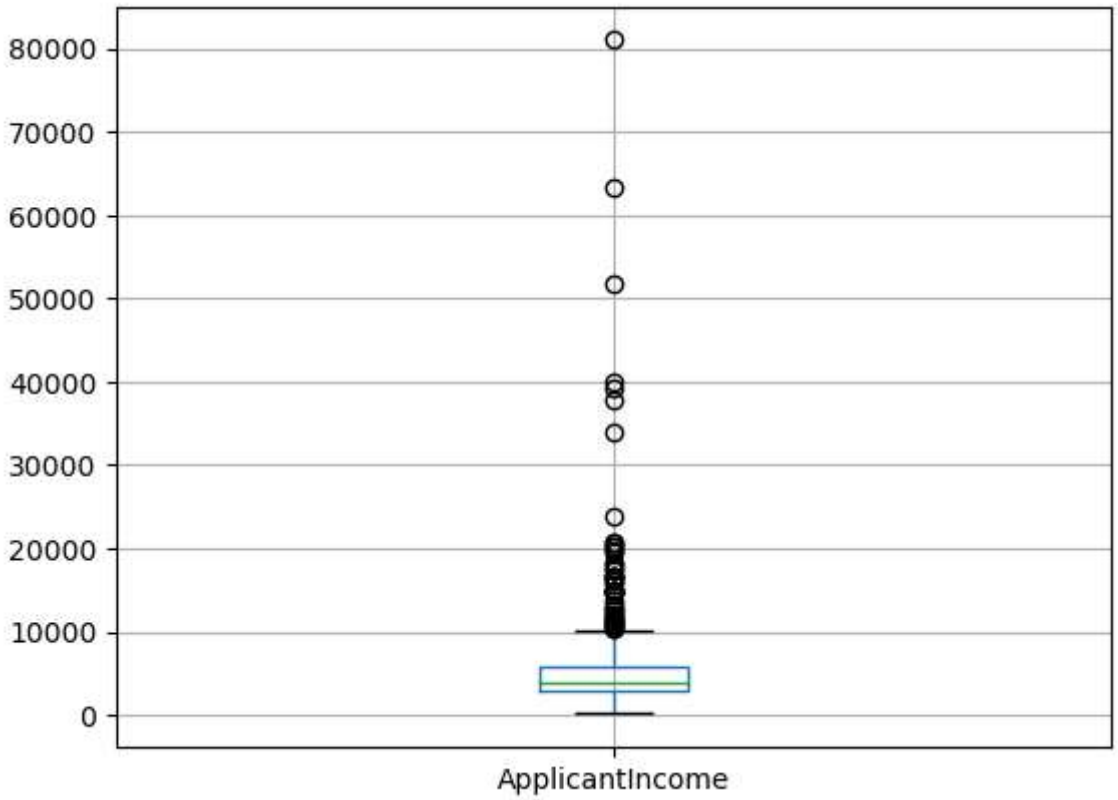
In [9]:
```
df.isnull().sum()
```

Out[9]:
```
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
```

Identifying Outliers in ApplicantIncome Using Boxplot

In [10]:
```
df.boxplot(column = 'ApplicantIncome')
```

Out[10]:
```
<Axes: >
```



Relationship Between Credit History and loan Status using Crosstab

In [11]:
```
pd.crosstab(df['Credit_History'], df['Loan_Status'], margins = True)
```

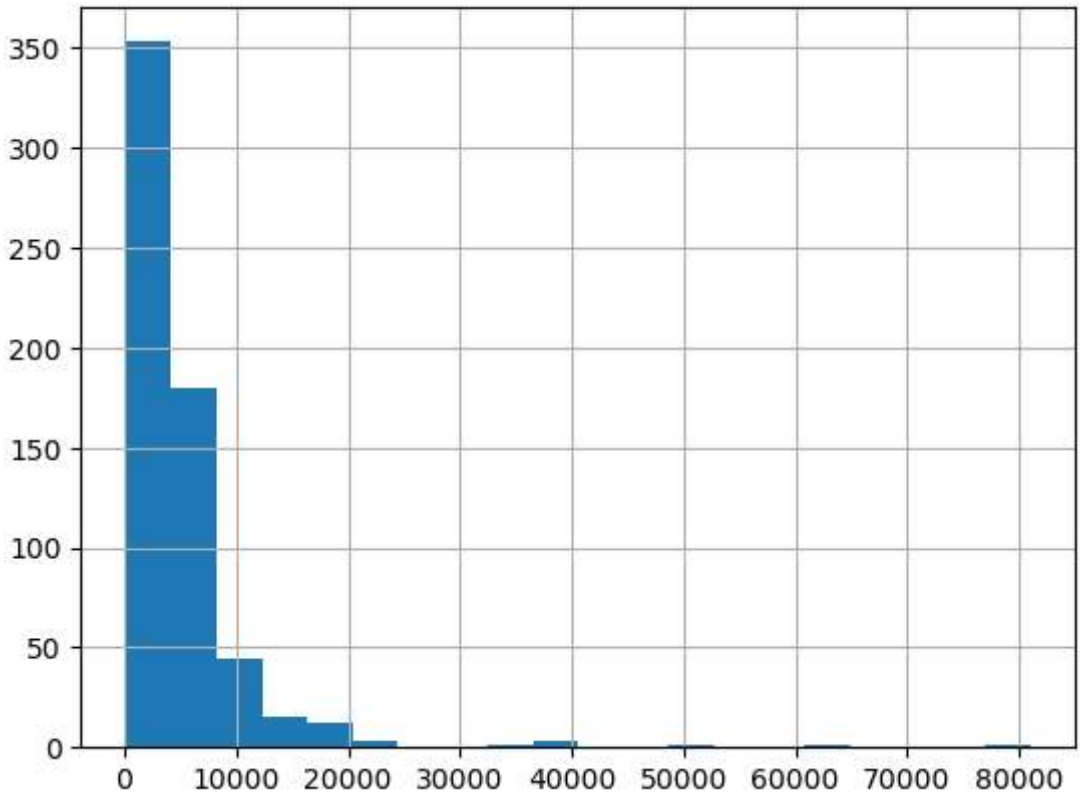
Out[11]:

Loan_Status	N	Y	All
Credit_History			
0.0	82	7	89
1.0	97	378	475
All	179	385	564

Findinng Skewness In Data Using Histogram

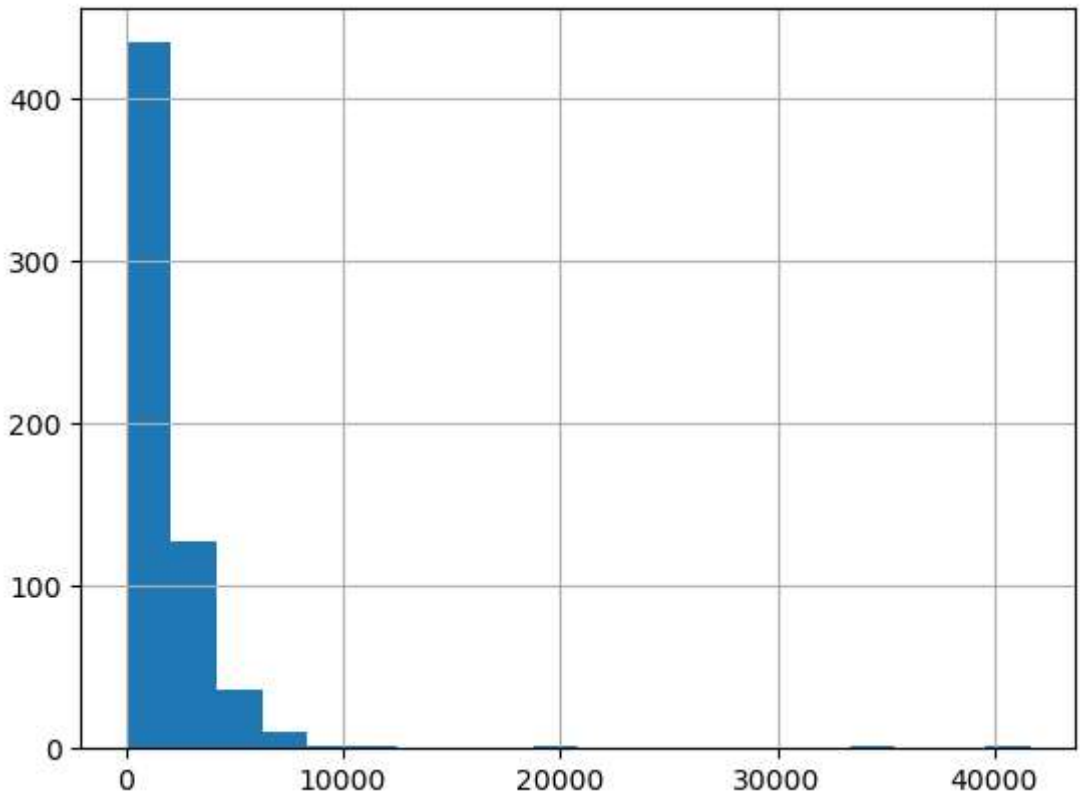
In [12]:
```
df['ApplicantIncome'].hist(bins = 20)
```

Out[12]:
```
<Axes: >
```



In [13]:
```
df['CoapplicantIncome'].hist(bins = 20)
```

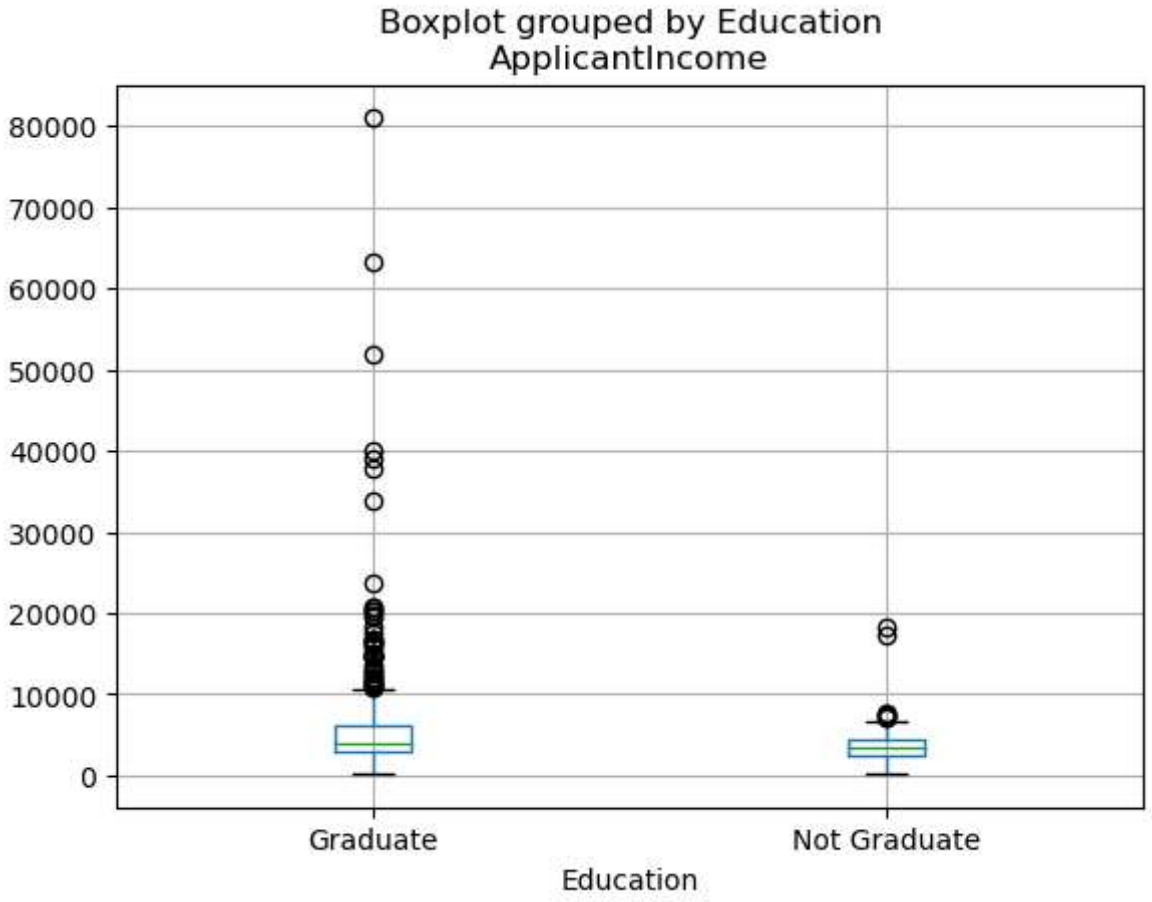
Out[13]:
```
<Axes: >
```



Relationship Between Income and Education with a Boxplot

In [14]:
```
df.boxplot(column = 'ApplicantIncome', by = 'Education')
```

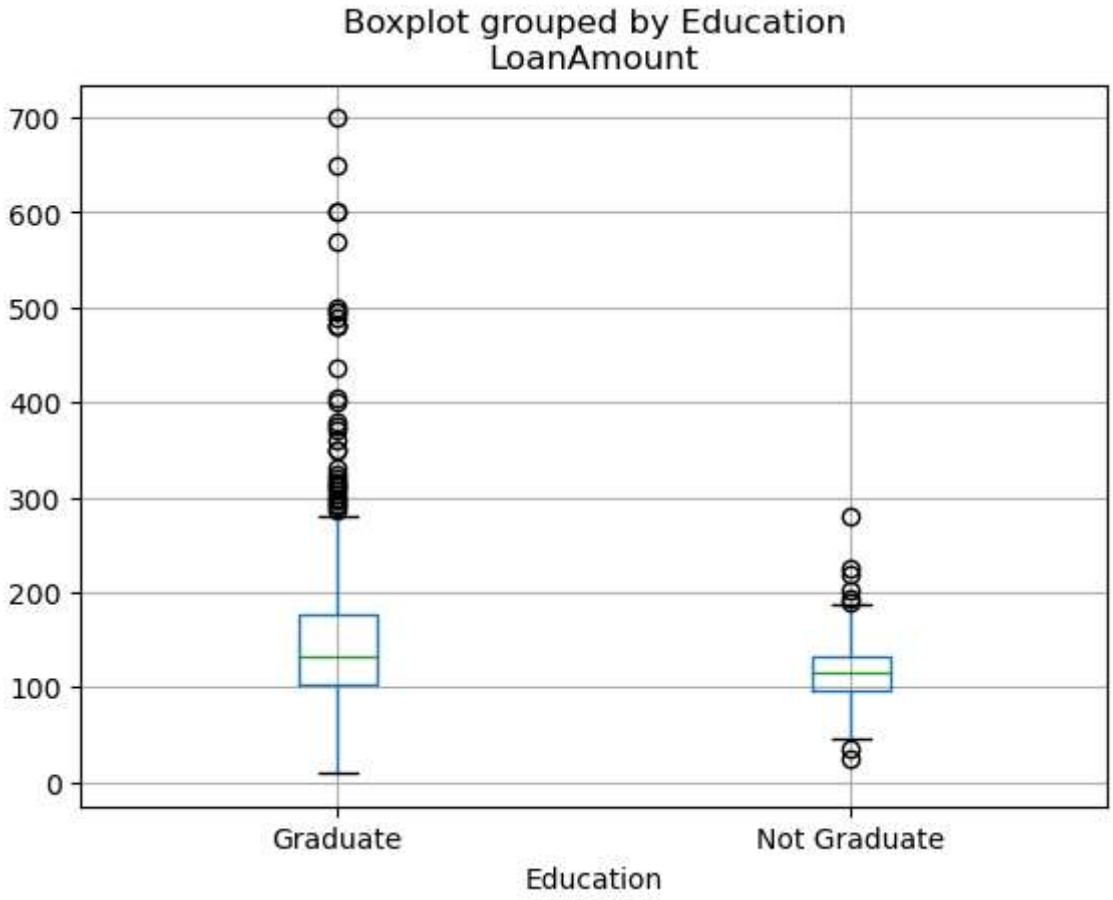
Out[14]:
```
<Axes: title={'center': 'ApplicantIncome'}, xlabel='Education'>
```



Relationship Between Loan Amount and Education with a Boxplot

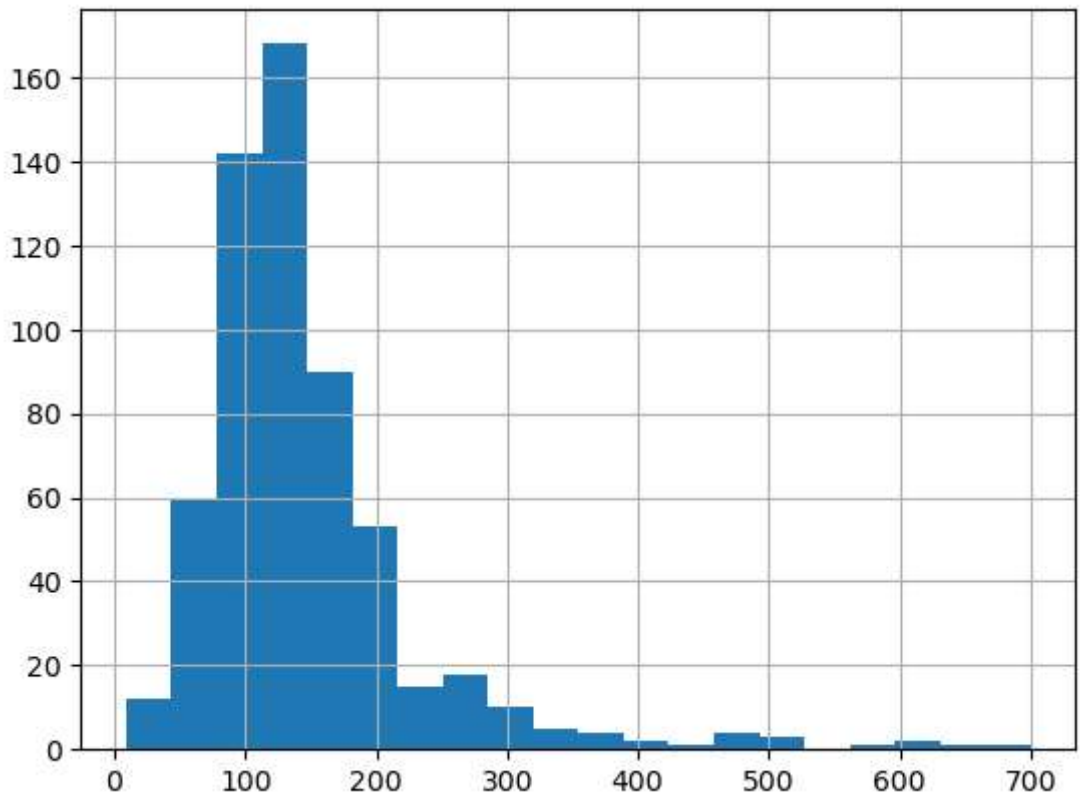
In [15]:
```
df.boxplot(column = 'LoanAmount', by = 'Education')
```

Out[15]:
```
<Axes: title={'center': 'LoanAmount'}, xlabel='Education'>
```



In [16]:
```
df['LoanAmount'].hist(bins = 20)
```

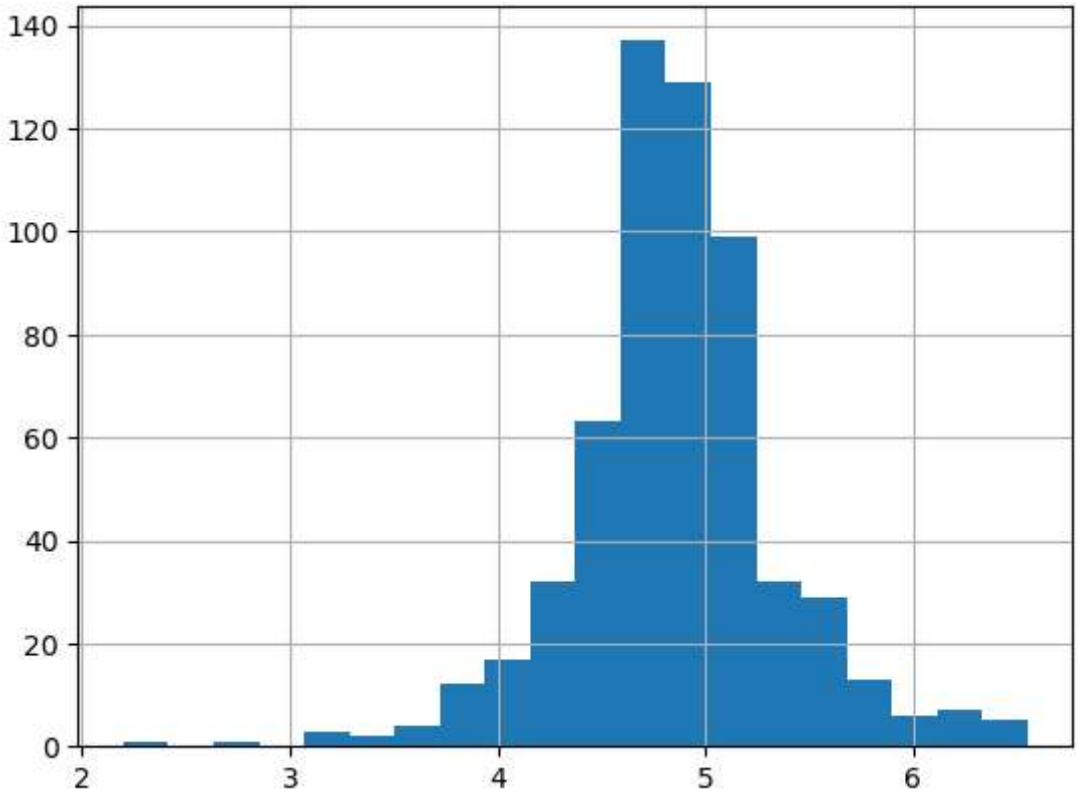
Out[16]:
```
<Axes: >
```



In [ ]:
```
# Normalizing Right Skew
```

In [17]:
```
df['LoanAmount_log'] = np.log(df['LoanAmount'])
df['LoanAmount_log'].hist(bins = 20)
```

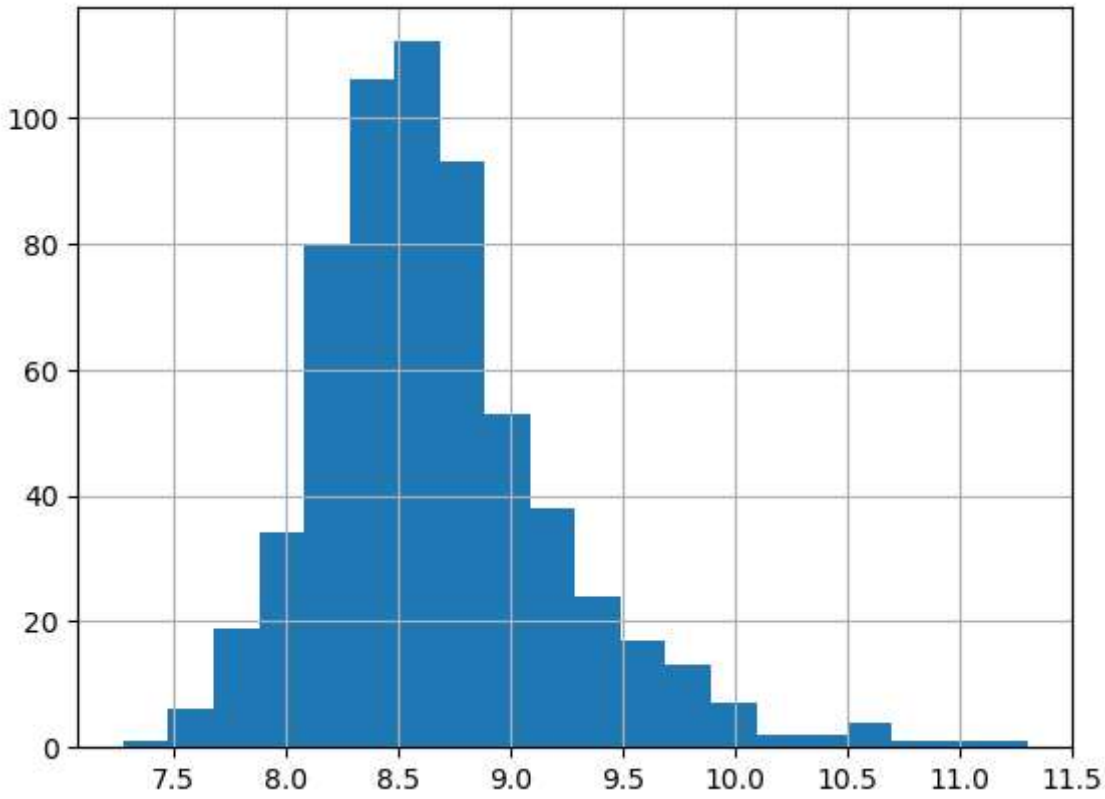
Out[17]:
```
<Axes: >
```



In [ ]:
```
# Combining All Income Columns Before Normalizing
```

In [18]:
```
df['TotalIncome'] = df['ApplicantIncome'] + df['CoapplicantIncome']
df['TotalIncome_log'] = np.log(df['TotalIncome'])
df['TotalIncome_log'].hist(bins = 20)
```

Out[18]:
```
<Axes: >
```



Filling missing values; categorical - mode, numeric - mean with indexing

In [19]:
```
df.columns
```

Out[19]:
```
Index(['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education',
       'Self_Employed', 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History', 'Property_Area', 'Loan_Status',
       'LoanAmount_log', 'TotalIncome', 'TotalIncome_log'],
      dtype='object')
```

In [23]:
```
df['Gender'].fillna(df['Gender'].mode()[0], inplace = True)
df['Married'].fillna(df['Married'].mode()[0], inplace = True)
df['Dependents'].fillna(df['Dependents'].mode()[0], inplace = True)
df['Education'].fillna(df['Education'].mode()[0], inplace = True)
df['Self_Employed'].fillna(df['Self_Employed'].mode()[0], inplace = True)
```

In [25]:
```
df.LoanAmount = df.LoanAmount.fillna(df.LoanAmount.mean())
df.LoanAmount_log = df.LoanAmount_log.fillna(df.LoanAmount_log.mean())
```

In [29]:
```
df['Loan_Amount_Term'].fillna(df['Loan_Amount_Term'].mode()[0], inplace = True)
df['Credit_History'].fillna(df['Credit_History'].mode()[0], inplace = True)
```

In [30]:
```
df.isnull().sum()
```

Out[30]:
```
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
LoanAmount_log       0
TotalIncome          0
TotalIncome_log      0
dtype: int64
```

Dividing Data into Dpenedent and Independent Variables

In [52]:
```
X = df.iloc[:,np.r_[1:5,9:11,13:15]].values
y = df.iloc[:,12].values
```

In [53]:
```
X
```

Out[53]:
```
array([['Male', 'No', '0', ..., 1.0, 4.857444178729352, 5849.0],
       ['Male', 'Yes', '1', ..., 1.0, 4.852030263919617, 6091.0],
       ['Male', 'Yes', '0', ..., 1.0, 4.189654742026425, 3000.0],
       ...,
       ['Male', 'Yes', '1', ..., 1.0, 5.53338948872752, 8312.0],
       ['Male', 'Yes', '2', ..., 1.0, 5.231108616854587, 7583.0],
       ['Female', 'No', '0', ..., 0.0, 4.890349128221754, 4583.0]],
      dtype=object)
```

In [54]:
```
y
```

Out[54]:
```
array(['Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'N', 'Y', 'Y', 'Y',
       'N', 'Y', 'Y', 'Y', 'N', 'N', 'Y', 'N', 'Y', 'N', 'N', 'N', 'Y',
       'Y', 'Y', 'N', 'Y', 'N', 'N', 'N', 'Y', 'N', 'Y', 'N', 'Y', 'Y',
       'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y',
       'N', 'N', 'N', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'N', 'N', 'N',
       'N', 'N', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'N', 'Y', 'N', 'N', 'N',
       'N', 'Y', 'Y', 'Y', 'N', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'Y', 'N', 'N', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y',
       'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'N',
       'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'N', 'Y', 'N', 'N', 'N', 'Y', 'Y',
       'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'N', 'Y', 'N', 'N', 'Y', 'Y',
       'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'N', 'Y', 'Y', 'Y', 'N', 'Y', 'N',
       'Y', 'Y', 'Y', 'N', 'Y', 'N', 'Y', 'Y', 'N', 'Y', 'N', 'N', 'N',
       'Y', 'N', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'N', 'N', 'Y', 'Y',
       'N', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y',
       'Y', 'N', 'N', 'N', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'N', 'Y', 'N',
       'Y', 'Y', 'Y', 'Y', 'N', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'N', 'Y',
       'Y', 'Y', 'Y', 'N', 'N', 'Y', 'Y', 'N', 'Y', 'N', 'N', 'N', 'N',
       'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'N', 'Y',
       'N', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'N',
       'N', 'N', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'N', 'N', 'Y', 'Y', 'Y',
       'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y',
       'N', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
       'N', 'Y', 'N', 'N', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y',
       'N', 'Y', 'N', 'Y', 'Y', 'Y', 'N', 'N', 'Y', 'N', 'Y', 'Y', 'Y',
       'Y', 'N', 'N', 'N', 'Y', 'N', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y',
       'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'N', 'Y', 'Y',
       'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'N', 'N', 'N', 'Y',
       'Y', 'N', 'Y', 'Y', 'Y', 'N', 'N', 'N', 'Y', 'N', 'Y', 'N', 'Y',
       'N', 'N', 'Y', 'Y', 'Y', 'N', 'Y', 'N', 'Y', 'Y', 'N', 'Y', 'Y',
       'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y',
       'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'N', 'N', 'N', 'Y', 'N', 'Y', 'Y',
       'Y', 'Y', 'N', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'N', 'Y',
       'Y', 'N', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'N', 'Y',
       'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'N', 'Y', 'N', 'Y', 'Y', 'Y', 'Y',
       'N', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y',
       'N', 'Y', 'Y', 'N', 'Y', 'Y', 'N', 'N', 'Y', 'Y', 'N', 'N', 'N',
       'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N',
       'N', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y',
       'N', 'Y', 'N', 'Y', 'N', 'Y', 'Y', 'N', 'N', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'N', 'N', 'N', 'Y', 'N',
       'Y', 'N', 'N', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'N', 'N',
       'N', 'Y', 'Y', 'Y', 'N', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'N', 'N',
       'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'N'], dtype=object)
```

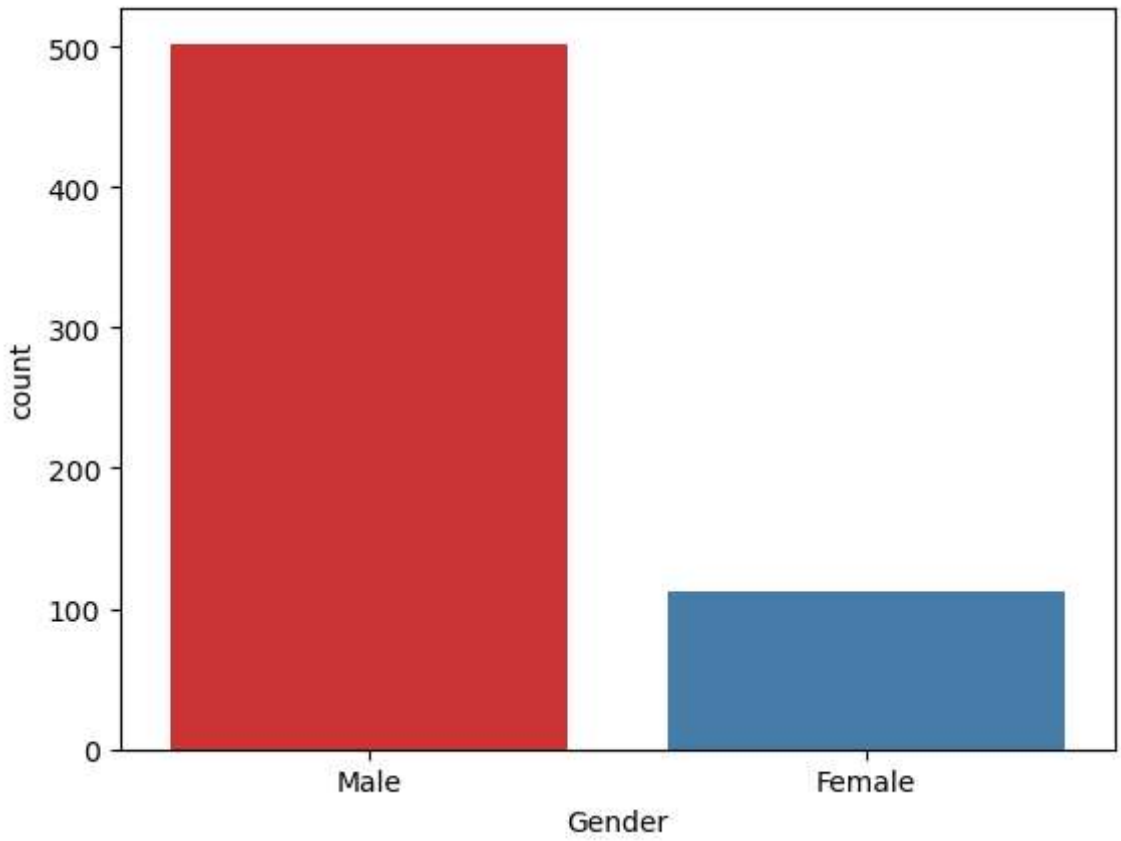
People Who Applied For A Loan By Gender

In [43]:
```
print('Number of people who applied for a loan by gender:')
print(df['Gender'].value_counts())
```

```
Number of people who applied for a loan by gender:
Male      502
Female    112
Name: Gender, dtype: int64
```

In [44]:
```
sns.countplot(x='Gender', data=df, palette = 'Set1')
```

Out[44]:
```
<Axes: xlabel='Gender', ylabel='count'>
```

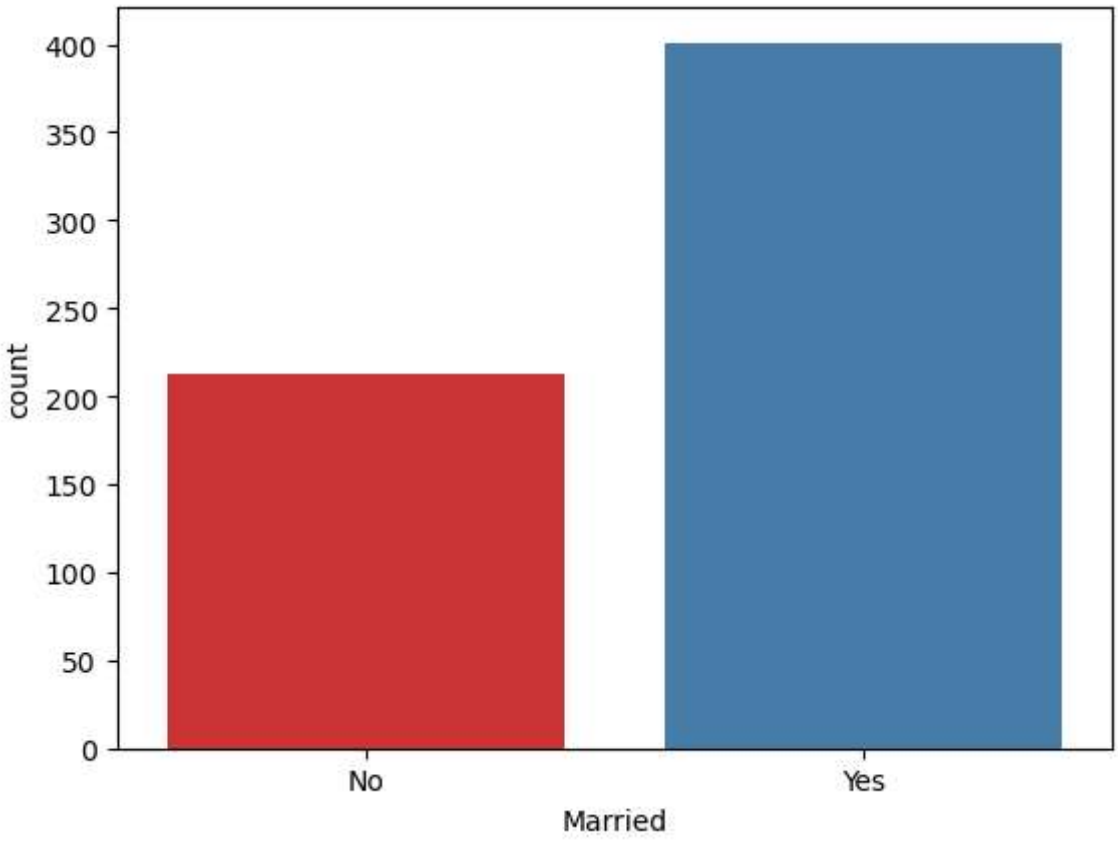


People Who Applied For A Loan By Martial Status

In [46]:
```
print('Number of people who applied for a loan by marital status:')
print(df['Married'].value_counts())
sns.countplot(x = 'Married', data = df, palette = 'Set1')
```

```
Number of people who applied for a loan by marital status:
Yes    401
No     213
Name: Married, dtype: int64
```

Out[46]:
```
<Axes: xlabel='Married', ylabel='count'>
```

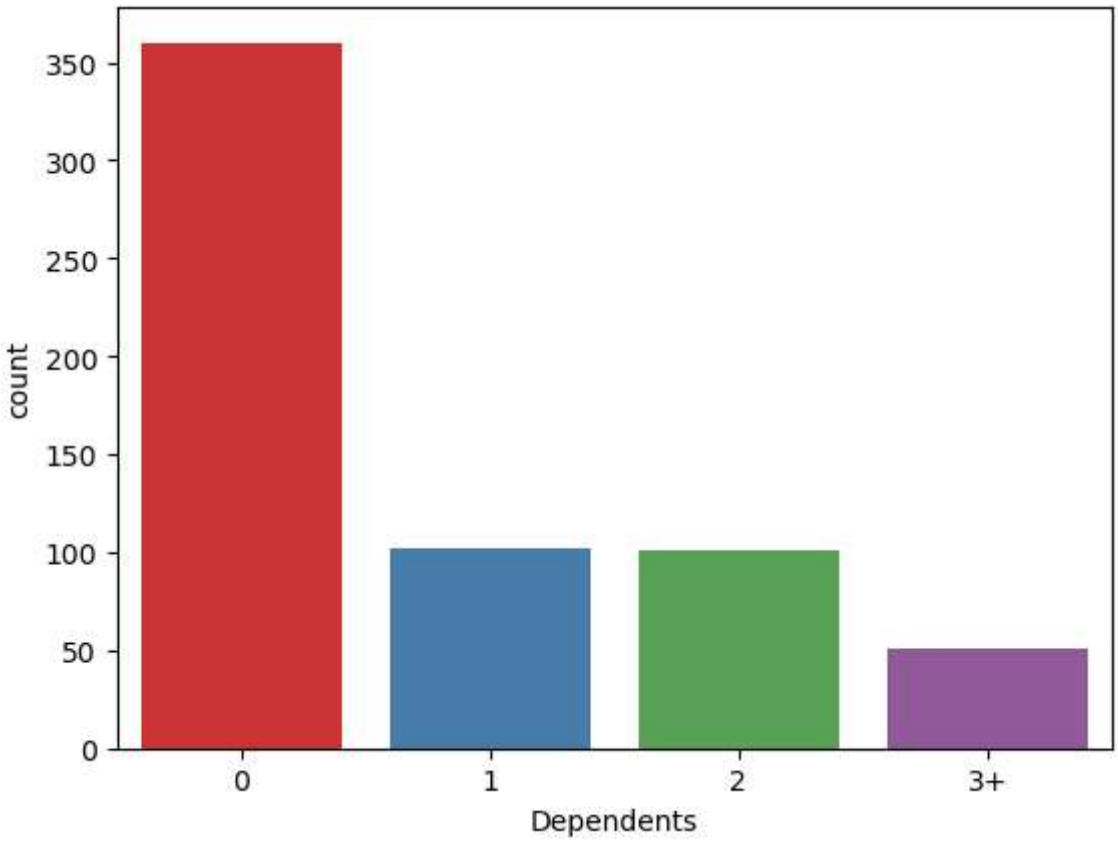


People Who Applied For A Loan By Dependents

In [47]:
```
print('Number of people who applied for a loan by dependence:')
print(df['Dependents'].value_counts())
sns.countplot(x = 'Dependents', data = df, palette = 'Set1')
```

```
Number of people who applied for a loan by dependence:
0     360
1     102
2     101
3+     51
Name: Dependents, dtype: int64
```

Out[47]:
```
<Axes: xlabel='Dependents', ylabel='count'>
```

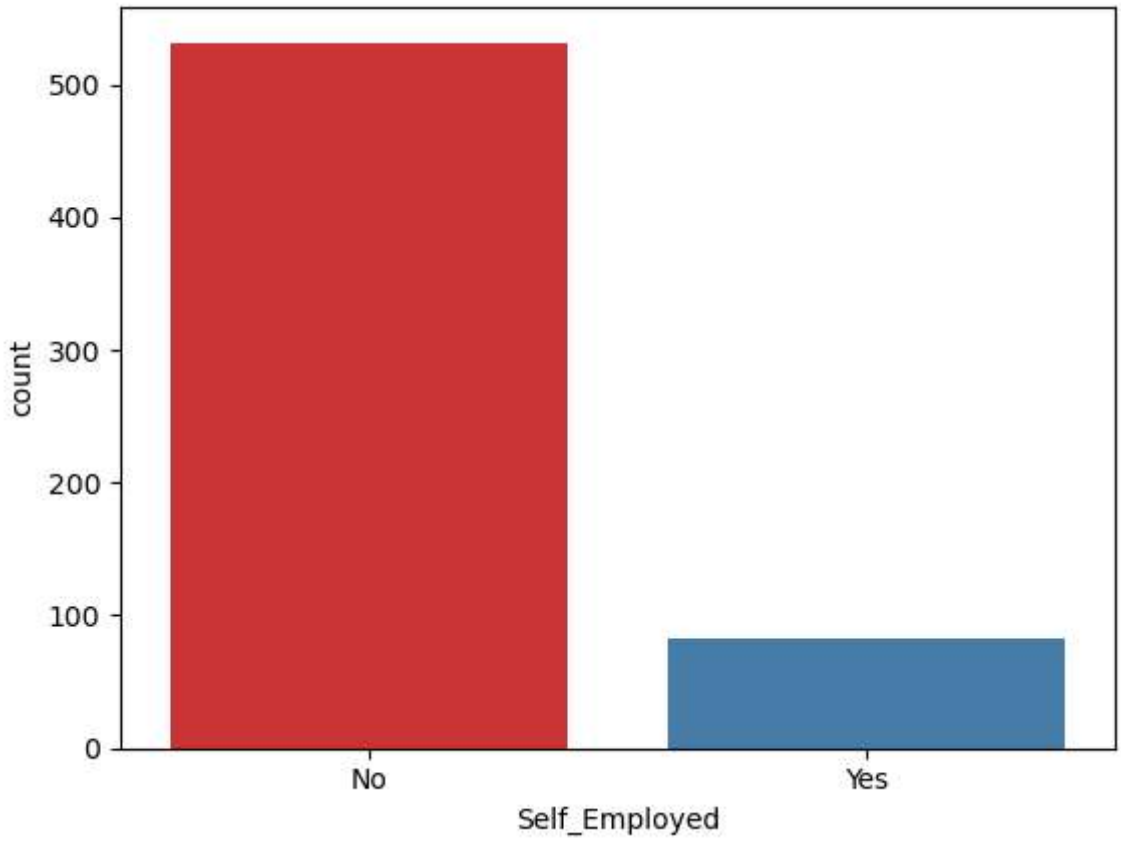


People Who Applied For A Loan By Self-Employment Status

In [48]:
```
print('Number of people who applied for a loan by self-employment status:')
print(df['Self_Employed'].value_counts())
sns.countplot(x = 'Self_Employed', data = df, palette = 'Set1')
```

```
Number of people who applied for a loan by employment status:
No     532
Yes     82
Name: Self_Employed, dtype: int64
```

Out[48]:
```
<Axes: xlabel='Self_Employed', ylabel='count'>
```

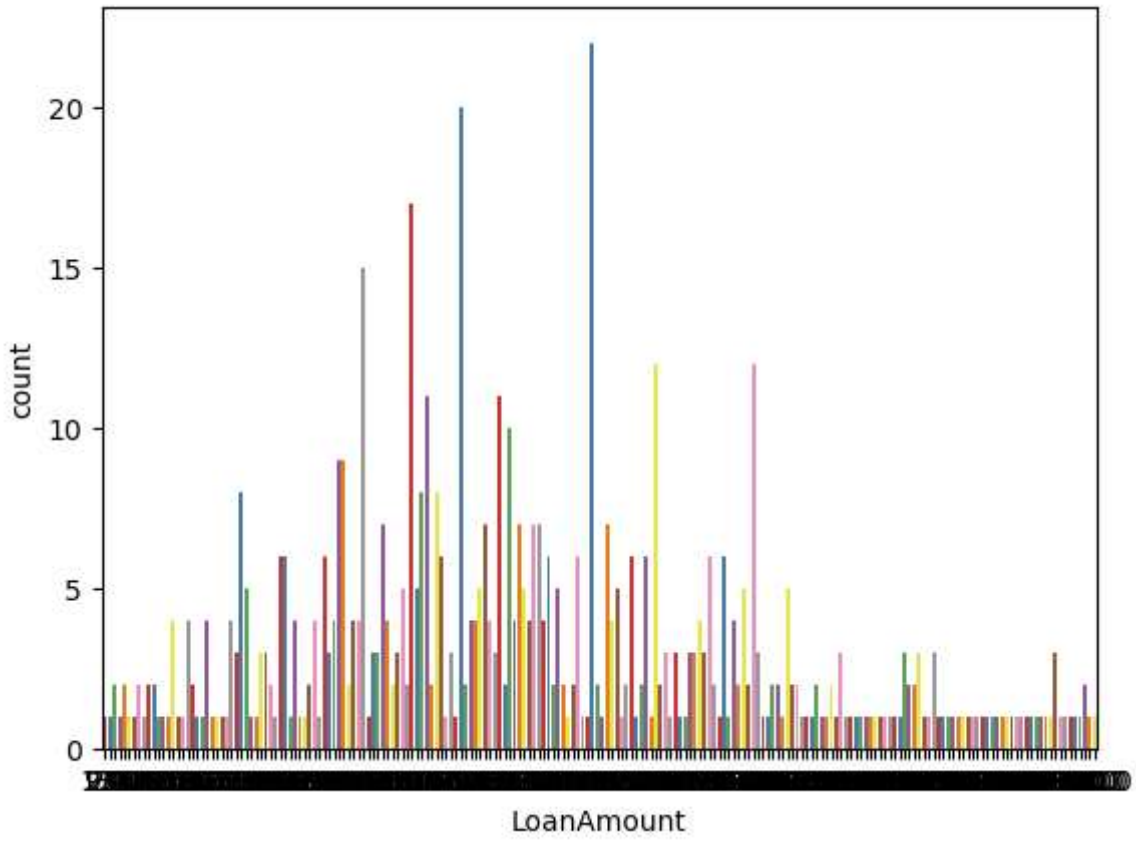


People Who Applied For A Loan By Loan Amount

In [49]:
```
print('Number of people who applied for a loan by loan amount:')
print(df['LoanAmount'].value_counts())
sns.countplot(x = 'LoanAmount', data = df, palette = 'Set1')
```

```
Number of people who applied for a loan by loan amount:
146.412162    22
120.000000    20
110.000000    17
100.000000    15
160.000000    12
              ..
240.000000     1
214.000000     1
59.000000      1
166.000000     1
253.000000     1
Name: LoanAmount, Length: 204, dtype: int64
```

Out[49]:
```
<Axes: xlabel='LoanAmount', ylabel='count'>
```

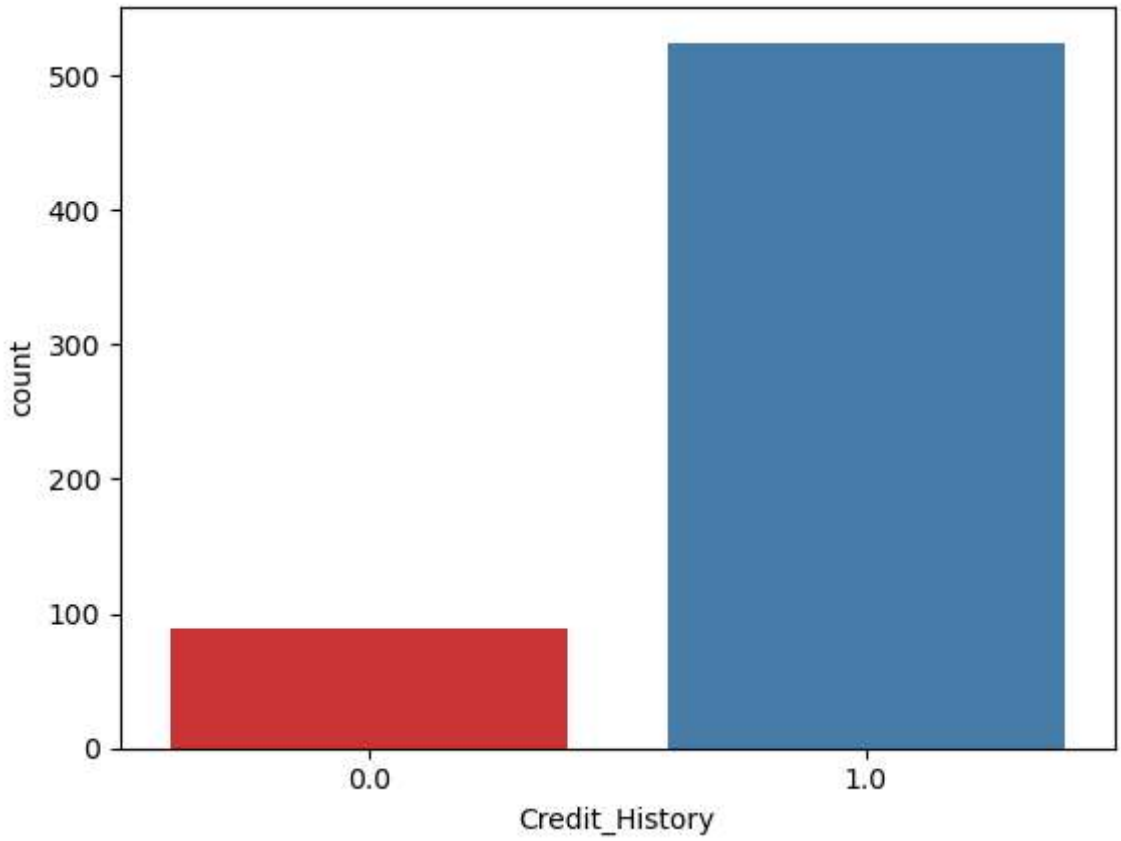


In [ ]:
```
# People Who Applied For A Loan By Credit History
```

In [50]:
```
print('Number of people who applied for a loan by credit history:')
print(df['Credit_History'].value_counts())
sns.countplot(x = 'Credit_History', data = df, palette = 'Set1')
```

```
Number of people who applied for a loan by credit history:
1.0    525
0.0     89
Name: Credit_History, dtype: int64
```

Out[50]:
```
<Axes: xlabel='Credit_History', ylabel='count'>
```



Importing from sklearn x for dependent, y for independent and splitting data

In [87]:
```
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)
```

In [88]:
```
print(X_train)
```

```
[['Female' 'No' '1' ... 1.0 4.882801922586371 5191.0]
 ['Male' 'Yes' '3+' ... 1.0 4.867534450455582 13746.0]
 ['Female' 'No' '2' ... 1.0 4.927253685157205 3427.0]
 ...
 ['Male' 'Yes' '2' ... 1.0 4.574710978503383 3875.0]
 ['Male' 'No' '0' ... 1.0 4.836281906951478 4690.0]
 ['Male' 'No' '0' ... 1.0 4.857444178729352 5849.0]]
```

Converting Categorical Variables into Numrical Form With LabelEncoder

In [89]:
```
from sklearn.preprocessing import LabelEncoder
labelencoder_X = LabelEncoder()
```

Using a For Loop To Apply Conversion In Train Data

In [90]:
```
for i in range(0, 5):
    X_train[:,i] = labelencoder_X.fit_transform(X_train[:,i])
```

In [91]:
```
X_train[:,7] = labelencoder_X.fit_transform(X_train[:,7])
```

In [92]:
```
X_train
```

Out[92]:
```
array([[0, 0, 1, ..., 1.0, 4.882801922586371, 203],
       [1, 1, 3, ..., 1.0, 4.867534450455582, 412],
       [0, 0, 2, ..., 1.0, 4.927253685157205, 51],
       ...,
       [1, 1, 2, ..., 1.0, 4.574710978503383, 87],
       [1, 0, 0, ..., 1.0, 4.836281906951478, 152],
       [1, 0, 0, ..., 1.0, 4.857444178729352, 256]], dtype=object)
```

Creating A New Instance Of The Label Encoder For Yes and No Values In y

In [93]:
```
labelencoder_y = LabelEncoder()
y_train = labelencoder_y.fit_transform(y_train)
```

In [94]:
```
y_train
```

Out[94]:
```
array([1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0,
       1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
       1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1,
       0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0,
       1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1,
       1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1,
       0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1,
       1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1,
       1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0,
       0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
       0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1,
       0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1,
       0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0,
       1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0,
       1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0,
       0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1,
       0, 0, 1, 1, 1, 1, 1])
```

Using For Loop To Apply Conversion In test Data

In [95]:
```
for i in range(0, 5):
    X_test[:,i] = labelencoder_X.fit_transform(X_test[:,i])
```

In [96]:
```
X_test[:,7] = labelencoder_X.fit_transform(X_test[:,7])
```

Converting Test Data In y

In [97]:
```
labelencoder_y = LabelEncoder()
y_test = labelencoder_y.fit_transform(y_test)
```

In [98]:
```
X_test
```

Out[98]:
```
array([[0, 0, 0, 0, 7, 0.0, 4.330733340286331, 23],
       [0, 0, 0, 1, 7, 1.0, 5.41610040220442, 116],
       [1, 1, 0, 1, 8, 0.0, 4.605170185988092, 49],
       [1, 1, 2, 1, 4, 1.0, 4.727387818712341, 26],
       [1, 1, 1, 0, 7, 1.0, 3.258096538021482, 78],
       [1, 1, 1, 0, 7, 1.0, 5.799092654460526, 104],
       [1, 1, 0, 1, 7, 1.0, 4.927253685157205, 52],
       [1, 1, 2, 1, 7, 1.0, 4.68213122712422, 37],
       [1, 1, 0, 0, 6, 1.0, 4.553876891600541, 72],
       [0, 0, 0, 0, 7, 0.0, 4.2626798770413155, 18],
       [1, 1, 3, 0, 7, 0.0, 5.886104031450156, 120],
       [1, 1, 0, 0, 7, 1.0, 5.1647859739235145, 92],
       [0, 0, 1, 0, 7, 1.0, 4.857444178729352, 90],
       [0, 1, 2, 0, 8, 1.0, 4.727387818712341, 28],
       [1, 1, 0, 0, 7, 1.0, 4.857444178729352, 117],
       [0, 0, 0, 0, 7, 1.0, 5.556828061699537, 105],
       [1, 1, 2, 0, 7, 1.0, 5.4638318050256105, 99],
       [1, 1, 2, 1, 7, 1.0, 4.700480365792417, 59],
       [0, 0, 0, 0, 4, 1.0, 4.7535901911063645, 34],
       [0, 1, 0, 0, 7, 1.0, 4.867534450455582, 12],
       [0, 0, 1, 0, 7, 1.0, 4.394449154672439, 51],
       [1, 0, 0, 0, 7, 1.0, 5.231108616854587, 111],
       [0, 0, 0, 0, 7, 1.0, 4.700480365792417, 41],
       [1, 1, 2, 1, 7, 0.0, 4.718498871295094, 60],
       [1, 1, 2, 0, 7, 1.0, 4.787491742782046, 30],
       [0, 0, 0, 0, 7, 0.0, 4.59511985013459, 0],
       [1, 0, 0, 0, 6, 1.0, 4.394449154672439, 22],
       [1, 1, 2, 0, 6, 1.0, 5.438079308923196, 97],
       [1, 1, 0, 0, 7, 1.0, 4.189654742026425, 16],
       [1, 1, 1, 0, 7, 1.0, 5.521460917862246, 95],
       [0, 0, 0, 0, 7, 1.0, 4.02535169073515, 7],
       [1, 1, 1, 0, 7, 1.0, 4.442651256490317, 25],
       [1, 1, 3, 0, 7, 0.0, 6.20455776256869, 114],
       [1, 0, 0, 1, 7, 1.0, 4.890349128221754, 76],
       [1, 1, 2, 0, 7, 1.0, 5.3230099791384085, 79],
       [1, 0, 0, 0, 7, 1.0, 4.430816798843313, 86],
       [1, 1, 2, 0, 7, 1.0, 5.493061443340548, 94],
       [1, 1, 2, 0, 7, 1.0, 4.718498871295094, 102],
       [1, 1, 0, 0, 7, 1.0, 4.700480365792417, 32],
       [1, 0, 0, 0, 7, 1.0, 4.976733742420574, 44],
       [1, 0, 0, 0, 7, 0.0, 4.477336814478207, 14],
       [1, 0, 0, 0, 7, 1.0, 4.852030263919617, 88],
       [1, 1, 0, 0, 7, 1.0, 6.173786103901937, 118],
       [1, 1, 2, 1, 4, 0.0, 3.8066624897703196, 10],
       [1, 0, 1, 1, 5, 1.0, 4.605170185988092, 24],
       [1, 0, 0, 0, 7, 1.0, 4.584967478670572, 13],
       [0, 1, 2, 0, 7, 1.0, 4.248495242049359, 113],
       [1, 0, 0, 1, 8, 0.0, 4.624972813284271, 33],
       [1, 1, 1, 0, 7, 1.0, 4.857444178729352, 4],
       [0, 0, 1, 0, 7, 1.0, 5.075173815233827, 8],
       [1, 1, 2, 0, 7, 0.0, 4.663439094112067, 9],
       [1, 1, 2, 0, 2, 1.0, 4.68213122712422, 27],
       [1, 1, 0, 0, 0, 1.0, 4.709530201312334, 91],
       [1, 1, 3, 0, 7, 1.0, 4.700480365792417, 64],
       [1, 0, 0, 0, 7, 1.0, 5.231108616854587, 100],
       [1, 0, 0, 0, 7, 1.0, 4.07753744390572, 5],
       [1, 1, 0, 0, 7, 1.0, 4.700480365792417, 83],
       [0, 0, 1, 0, 7, 1.0, 5.0106352940962555, 98],
       [0, 0, 0, 1, 7, 1.0, 4.189654742026425, 1],
       [1, 1, 3, 0, 7, 1.0, 5.66988092298052, 110],
```

```
[0, 1, 0, 0, 7, 1.0, 5.0369526024136295, 62],
[0, 1, 0, 1, 7, 0.0, 5.181783550292085, 61],
[1, 0, 0, 0, 7, 1.0, 4.499809670330265, 119],
[1, 1, 2, 0, 7, 1.0, 4.564348191467836, 36],
[1, 1, 2, 0, 7, 0.0, 5.552959584921617, 101],
[0, 1, 0, 0, 5, 1.0, 4.857444178729352, 103],
[1, 0, 0, 1, 7, 1.0, 4.787491742782046, 68],
[1, 1, 0, 0, 7, 1.0, 4.787491742782046, 77],
[1, 1, 3, 0, 7, 1.0, 4.605170185988092, 43],
[1, 1, 1, 0, 7, 1.0, 5.298317366548036, 85],
[1, 1, 2, 1, 7, 1.0, 5.231108616854587, 89],
[1, 0, 0, 1, 1, 1.0, 4.382026634673881, 19],
[0, 0, 0, 0, 8, 1.0, 4.727387818712341, 45],
[0, 1, 0, 0, 7, 1.0, 3.58351893845611, 17],
[1, 1, 0, 0, 7, 0.0, 4.0943445622221, 2],
[0, 1, 1, 0, 7, 0.0, 5.655991810819852, 108],
[0, 0, 0, 0, 7, 1.0, 4.0943445622221, 15],
[0, 1, 0, 0, 7, 0.0, 4.787491742782046, 73],
[1, 0, 3, 1, 7, 1.0, 4.997212273764115, 82],
[1, 1, 0, 1, 7, 1.0, 4.189654742026425, 21],
[1, 1, 2, 0, 7, 1.0, 5.075173815233827, 81],
[1, 1, 0, 0, 7, 1.0, 4.875197323201151, 69],
[0, 1, 0, 1, 7, 1.0, 4.624972813284271, 29],
[1, 0, 0, 1, 3, 1.0, 3.2188758248682006, 27],
[1, 1, 1, 0, 7, 1.0, 5.087596335232384, 63],
[1, 1, 0, 0, 7, 1.0, 4.912654885736052, 107],
[1, 1, 2, 0, 6, 1.0, 5.940171252720432, 115],
[1, 1, 2, 1, 7, 1.0, 4.700480365792417, 48],
[0, 0, 0, 0, 7, 1.0, 4.532599493153256, 6],
[0, 0, 0, 0, 7, 1.0, 4.709530201312334, 35],
[1, 1, 0, 0, 7, 1.0, 5.579729825986222, 96],
[1, 1, 2, 0, 7, 1.0, 5.043425116919247, 58],
[1, 1, 0, 0, 7, 1.0, 4.007333185232471, 55],
[1, 1, 3, 0, 7, 1.0, 4.605170185988092, 50],
[1, 0, 0, 0, 7, 1.0, 4.605170185988092, 31],
[0, 0, 0, 0, 7, 1.0, 4.430816798843313, 11],
[1, 1, 3, 1, 7, 1.0, 4.867534450455582, 65],
[1, 0, 2, 0, 7, 0.0, 5.111987788356544, 56],
[1, 1, 2, 0, 7, 1.0, 5.2574953720277815, 67],
[1, 1, 0, 0, 7, 1.0, 4.857444178729352, 112],
[1, 1, 0, 0, 7, 0.0, 5.10594547390058, 80],
[1, 1, 0, 1, 7, 0.0, 4.812184355372417, 54],
[1, 0, 0, 0, 7, 0.0, 5.342334251964811, 106],
[1, 0, 0, 0, 7, 1.0, 4.941642422609304, 78],
[1, 1, 1, 0, 4, 1.0, 5.204006687076795, 109],
[1, 1, 3, 0, 4, 1.0, 4.852030263919617, 93],
[1, 0, 0, 0, 7, 1.0, 4.30406509320417, 38],
[1, 0, 0, 0, 7, 1.0, 4.804021044733257, 40],
[1, 1, 2, 1, 7, 1.0, 4.727387818712341, 3],
[0, 1, 0, 1, 7, 1.0, 4.927253685157205, 87],
[1, 1, 0, 0, 4, 0.0, 4.499809670330265, 57],
[1, 0, 0, 1, 7, 1.0, 4.882801922586371, 39],
[0, 0, 1, 0, 7, 1.0, 4.499809670330265, 46],
[1, 0, 0, 0, 7, 1.0, 4.48863636973214, 75],
[1, 1, 2, 0, 7, 1.0, 4.477336814478207, 84],
[1, 1, 0, 1, 7, 0.0, 5.198497031265826, 53],
[0, 0, 0, 0, 7, 1.0, 4.718498871295094, 42],
[1, 1, 0, 1, 7, 1.0, 5.075173815233827, 71],
[1, 1, 0, 0, 7, 1.0, 4.74493212836325, 74],
[1, 1, 0, 0, 7, 1.0, 4.787491742782046, 66],
```

```
       [1, 1, 0, 0, 7, 1.0, 5.123963979403259, 70],
       [1, 1, 0, 0, 4, 1.0, 4.290459441148391, 47],
       [0, 0, 0, 0, 7, 1.0, 4.477336814478207, 20]], dtype=object)
```

In [99]:
```
y_test
```

Out[99]:
```
array([0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1,
       1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0,
       1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1,
       1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1])
```

Scaling Data To Account For Different Ranges Using StandardScaler

In [100…
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.fit_transform(X_test)
```

Creating An Instance Of Decision Tree and Testing Accuracy

In [115…
```
from sklearn.tree import DecisionTreeClassifier
DTClassifier = DecisionTreeClassifier(criterion = 'entropy', random_state =0)
DTClassifier.fit(X_train, y_train)
```

Out[115]:
```
DecisionTreeClassifier(criterion='entropy', random_state=0)
```

Predicting Using This Algorithm

Creating A New Variable

In [116…
```
y_pred = DTClassifier.predict(X_test)
y_pred
```

Out[116]:
```
array([0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0,
       1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0,
       1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0,
       1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1,
       1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1,
       0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1])
```

Finding Accuracy Using Metrics

In [117…
```
from sklearn import metrics
print("The Accuracy Of Decision Tree Is: ", metrics.accuracy_score(y_pred, y_test)*100
```

```
The Accuracy Of Decision Tree Is:  65.04065040650406
```

A Score of 65% Not Good So Applying Other Algorithm

In [118…
```
from sklearn.naive_bayes import GaussianNB
NBClassifier = GaussianNB()
NBClassifier.fit(X_train, y_train)
```

Out[118]:
```
GaussianNB()
```

In [119…
```
y_pred = NBClassifier.predict(X_test)
```

In [120…
```
y_pred
```

Out[120]:
```
array([0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0,
       1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
       0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1])
```

In [121…
```
print("The Accuracy Of Naive Bayes Is: ", metrics.accuracy_score(y_pred, y_test)*100)
```

```
The Accuracy Of Naive Bayes Is:  84.5528455284553
```

Accuracy of 84% Quite Good But Try Another

In [122…
```
from sklearn.ensemble import RandomForestClassifier
RFClassifier = RandomForestClassifier()
RFClassifier.fit(X_train, y_train)
```

Out[122]:
```
RandomForestClassifier()
```

In [123…
```
y_pred = RFClassifier.predict(X_test)
```

In [124…
```
print("The Accuracy Of Random Forest Is: ", metrics.accuracy_score(y_pred, y_test)*100
```

```
The Accuracy Of Random Forest Is:  78.04878048780488
```

Accuracy is 78% Trying Another

In [125…
```
from sklearn.neighbors import KNeighborsClassifier
KNClassifier = KNeighborsClassifier()
KNClassifier.fit(X_train, y_train)
```

Out[125]:
```
KNeighborsClassifier
KNeighborsClassifier()
```

In [126…
```
y_pred = KNClassifier.predict(X_test)
```

In [127…
```
print("The Accuracy Of Neighbors Classifier Is: ", metrics.accuracy_score(y_pred, y_te
```

```
The Accuracy Of Neighbors Classifier Is:  79.67479674796748
```

Accuracy of 79%. Thus We Use Bayes

Trying Bayes on Test Data Having No Eligibility Label

In [129…
```
data = pd.read_csv(r"C:\Users\falco\OneDrive\Desktop\PROJECTS\PYTHON PROJECTS\MY PROJE
```

Visualising Head

In [130…
```
data.head()
```

Out[130]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantI
0	LP001003	Male	Yes	1	Graduate	No	4583	
1	LP001005	Male	Yes	0	Graduate	Yes	3000	
2	LP001006	Male	Yes	0	Not Graduate	No	2583	
3	LP001008	Male	No	0	Graduate	No	6000	
4	LP001011	Male	Yes	2	Graduate	Yes	5417	

Making Sure There Are No Missing Values

In [131…
```
data.isnull().sum()
```

Out[131]:
```
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
dtype: int64
```

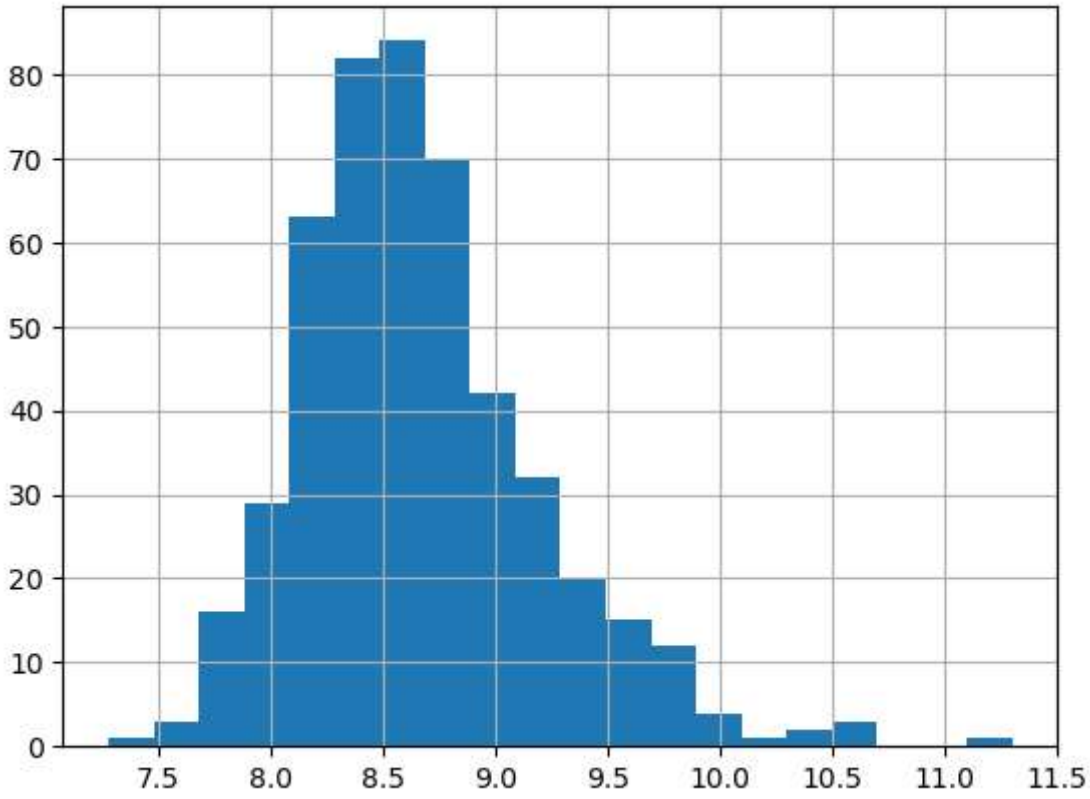
Normalizing and Visualising Here Too

In [132…
```
data['LoanAmount_log'] = np.log(data['LoanAmount'])
```

In [133…
```
data['TotalIncome'] = data['ApplicantIncome'] + data['CoapplicantIncome']
data['TotalIncome_Log'] = np.log(data['TotalIncome'])
```

In [134…
```
data['TotalIncome_Log'].hist(bins = 20)
```

Out[134]:
```
<Axes: >
```



In [135…
```
data.head()
```

Out[135]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantI
0	LP001003	Male	Yes	1	Graduate	No	4583	
1	LP001005	Male	Yes	0	Graduate	Yes	3000	
2	LP001006	Male	Yes	0	Not Graduate	No	2583	
3	LP001008	Male	No	0	Graduate	No	6000	
4	LP001011	Male	Yes	2	Graduate	Yes	5417	

Selecting data into variable

In [136…
```
test = data.iloc[:,np.r_[1:5,9:11,13:15]].values
```

Convert categorical to num

In [137…
```
for i in range(0, 5):
    test[:,i] = labelencoder_X.fit_transform(test[:,i])
```

In [138…
```
test[:,7] = labelencoder_X.fit_transform(test[:,7])
```

In [139…
```
test
```

Out[139]:
```
array([[1, 1, 1, ..., 1, 6091.0, 261],
       [1, 1, 0, ..., 1, 3000.0, 34],
       [1, 1, 0, ..., 1, 4941.0, 185],
       ...,
       [1, 1, 1, ..., 1, 8312.0, 338],
       [1, 1, 2, ..., 1, 7583.0, 322],
       [0, 0, 0, ..., 0, 4583.0, 144]], dtype=object)
```

Scaling test data

In [140…
```
test = sc.fit_transform(test)
```

Finally let's predict

In [141…
```
pred = NBClassifier.predict(test)
```

In [142…
```
pred
```

Out[142]:
```
array([1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 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, 1, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1,
       1, 0, 1, 0, 1, 1, 1, 0, 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, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1,
       1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 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, 0, 1,
       1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0,
       0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 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, 1, 0, 1, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0])
```