Importing Libraries

In [8]:
```
import numpy as np
#numpy is library used for array and it has functions for working in domain of line
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt
from sklearn import svm
```

Importing Dataset

In [12]:
```
df = pd.read_csv("D:\Amit_Project\loan approval project using python\Copy of loan.c
df.head()
```

Out[12]:

	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 [13]:
```
df.info()
```

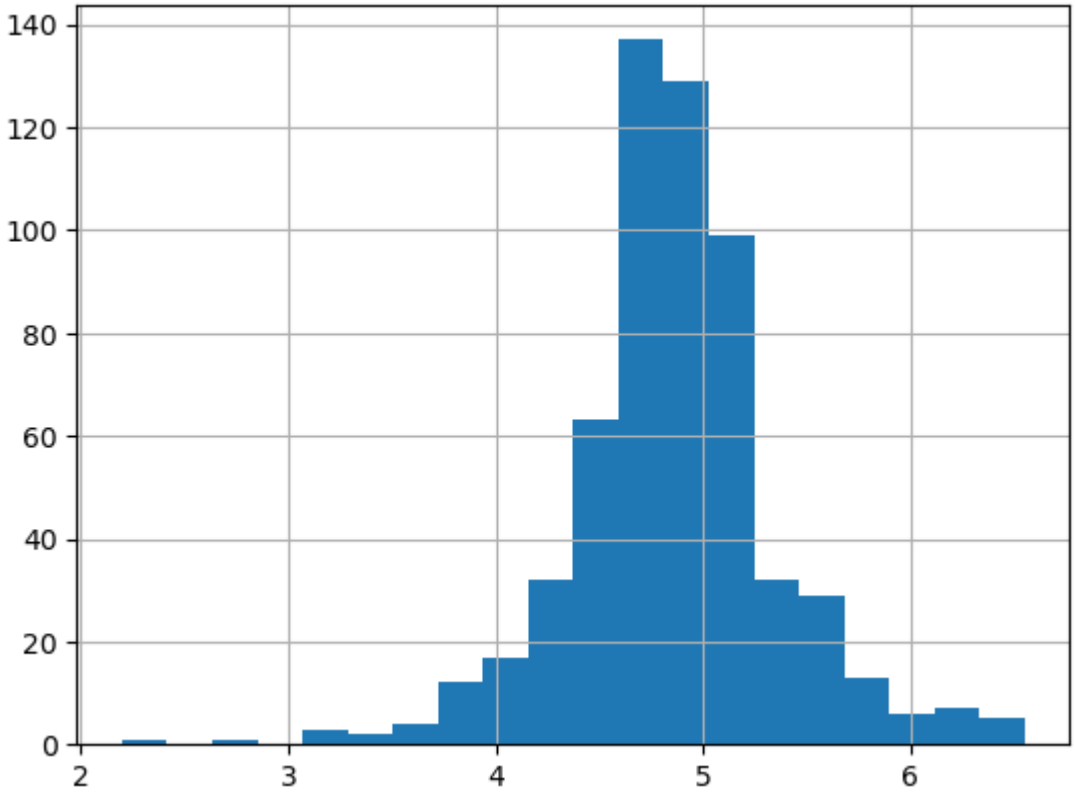
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Loan_ID            614 non-null    object 
 1   Gender             601 non-null    object 
 2   Married            611 non-null    object 
 3   Dependents         599 non-null    object 
 4   Education          614 non-null    object 
 5   Self_Employed      582 non-null    object 
 6   ApplicantIncome    614 non-null    int64  
 7   CoapplicantIncome  614 non-null    float64
 8   LoanAmount         592 non-null    float64
 9   Loan_Amount_Term   600 non-null    float64
 10  Credit_History     564 non-null    float64
 11  Property_Area      614 non-null    object 
 12  Loan_Status        614 non-null    object 
dtypes: float64(4), int64(1), object(8)
memory usage: 62.5+ KB
```

In [16]:
```
df.isnull().sum()
```

Out[16]:
```
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 [17]:
```
df['LoanAmount_log'] = np.log(df['LoanAmount'])
df['LoanAmount_log'].hist(bins=20)
```

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

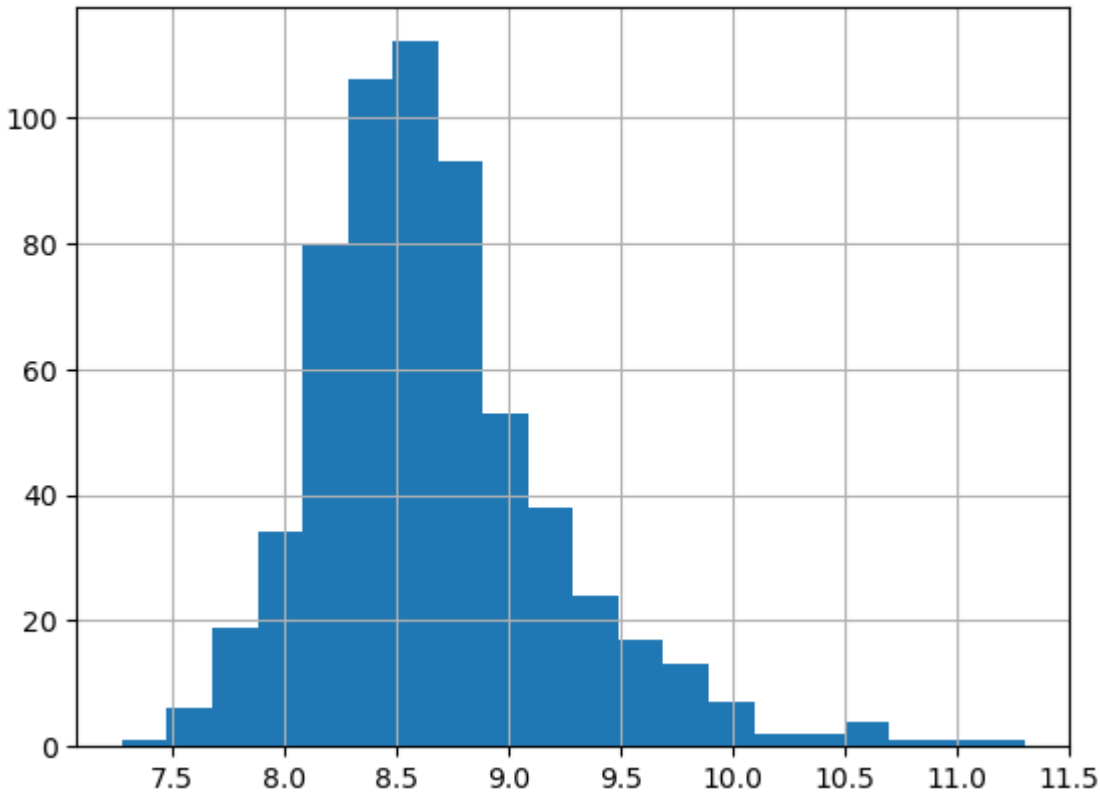


In [18]:
```
# Null values in new column
df['LoanAmount_log'].isnull().sum()
```

Out[18]:
```
22
```

In [25]:
```
df['TotalIncome'] = df['ApplicantIncome']+ df['CoapplicantIncome']
df['TotalIncome'] = np.log(df['TotalIncome'])
df['TotalIncome'].hist(bins=20)
```

Out[25]:
```
<Axes: >
```



In [28]:
```
# Fill Null values all the respective columns
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['Self_Employed'].fillna(df['Self_Employed'].mode()[0], inplace = True)
df['LoanAmount'].fillna(df['LoanAmount'].mode()[0], inplace = True)
df['Loan_Amount_Term'].fillna(df['Loan_Amount_Term'].mode()[0], inplace = True)
df['Credit_History'].fillna(df['Credit_History'].mode()[0], inplace = True)
df['LoanAmount_log'].fillna(df['LoanAmount_log'].mode()[0], inplace = True)
#Check if any null values are still present
df.isnull().sum()
```

Out[28]:
```
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
dtype: int64
```

In [29]:
```
#Selecting some specific columns for training and testing
#iloc is the funtion defines in the pandas modules that helps--
#us to select specific row or column from the dataset(retrieve any vaule from row c
x= df.iloc[:,np.r_[1:5, 9:11, 13:15]].values
y= df.iloc[:,12].values

x
```

Out[29]:
```
array([['Male', 'No', '0', ..., 1.0, 4.787491742782046,
        8.674025985443025],
       ['Male', 'Yes', '1', ..., 1.0, 4.852030263919617,
        8.714567550836485],
       ['Male', 'Yes', '0', ..., 1.0, 4.189654742026425,
        8.006367567650246],
       ...,
       ['Male', 'Yes', '1', ..., 1.0, 5.53338948872752,
        9.025455532779063],
       ['Male', 'Yes', '2', ..., 1.0, 5.231108616854587,
        8.933664178700935],
       ['Female', 'No', '0', ..., 0.0, 4.890349128221754,
        8.430109084509125]], dtype=object)
```

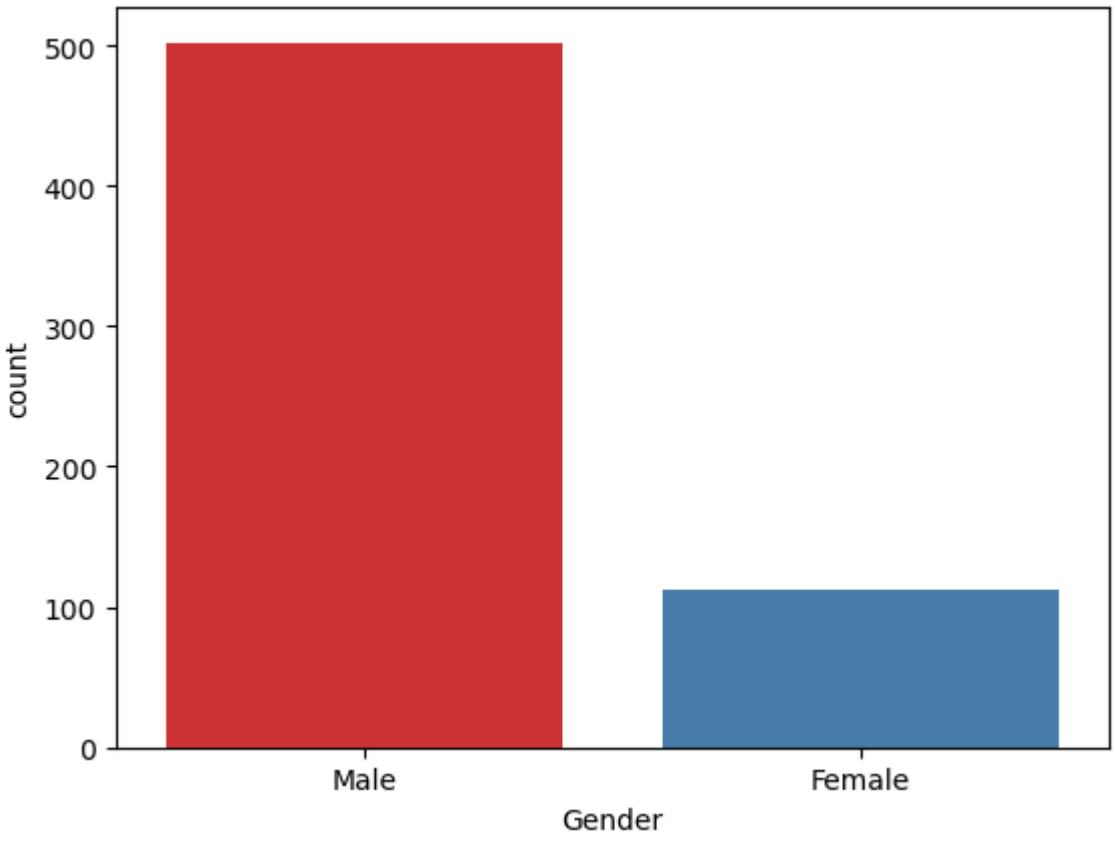
In [30]:
```
y
```

Out[30]:
```
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)
```

In [33]:
```
print("Persons who take loan as group by gender:")
print(df['Gender'].value_counts())
sns.countplot(x='Gender', data=df, palette= 'Set1')
```

```
Persons who take loan as group by gender:
Male      502
Female    112
Name: Gender, dtype: int64
```

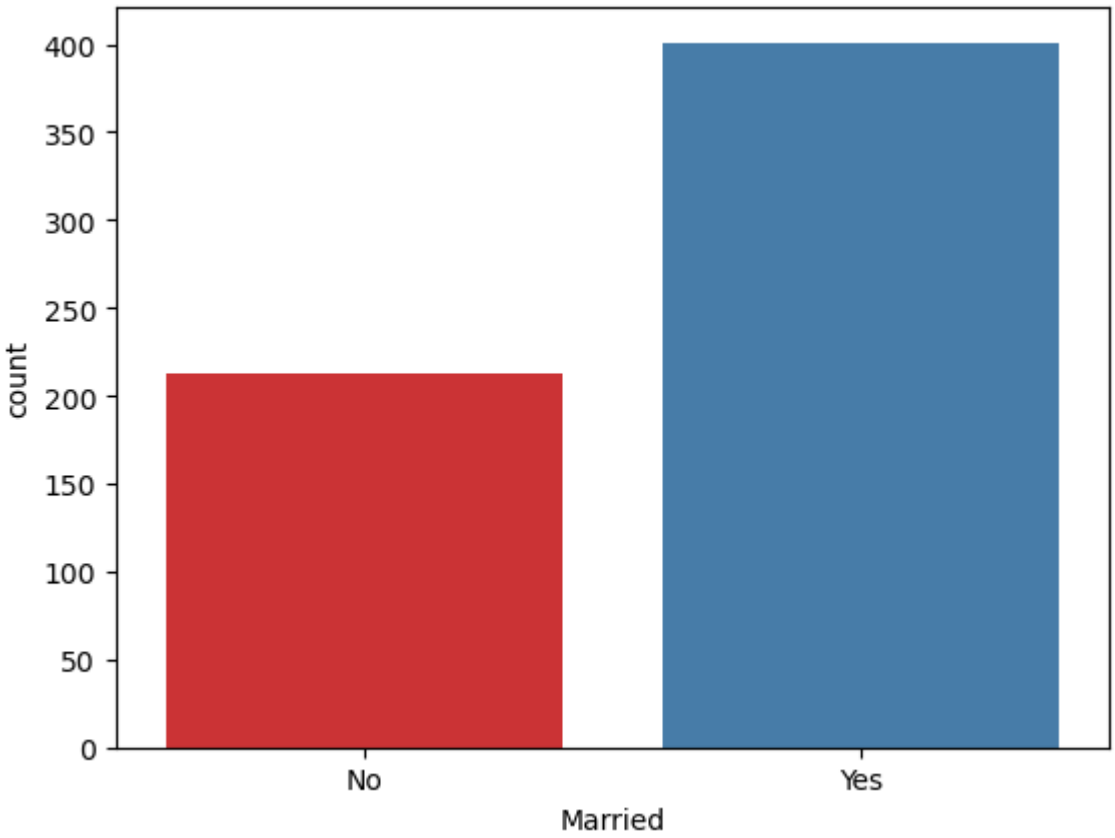
Out[33]:
```
<Axes: xlabel='Gender', ylabel='count'>
```



In [35]:
```
print("Who take loan as group by Marital status:")
print(df['Married'].value_counts())
sns.countplot(x='Married', data=df, palette= 'Set1')
```

```
Who take loan as group by Marital status:
Yes    401
No     213
Name: Married, dtype: int64
```

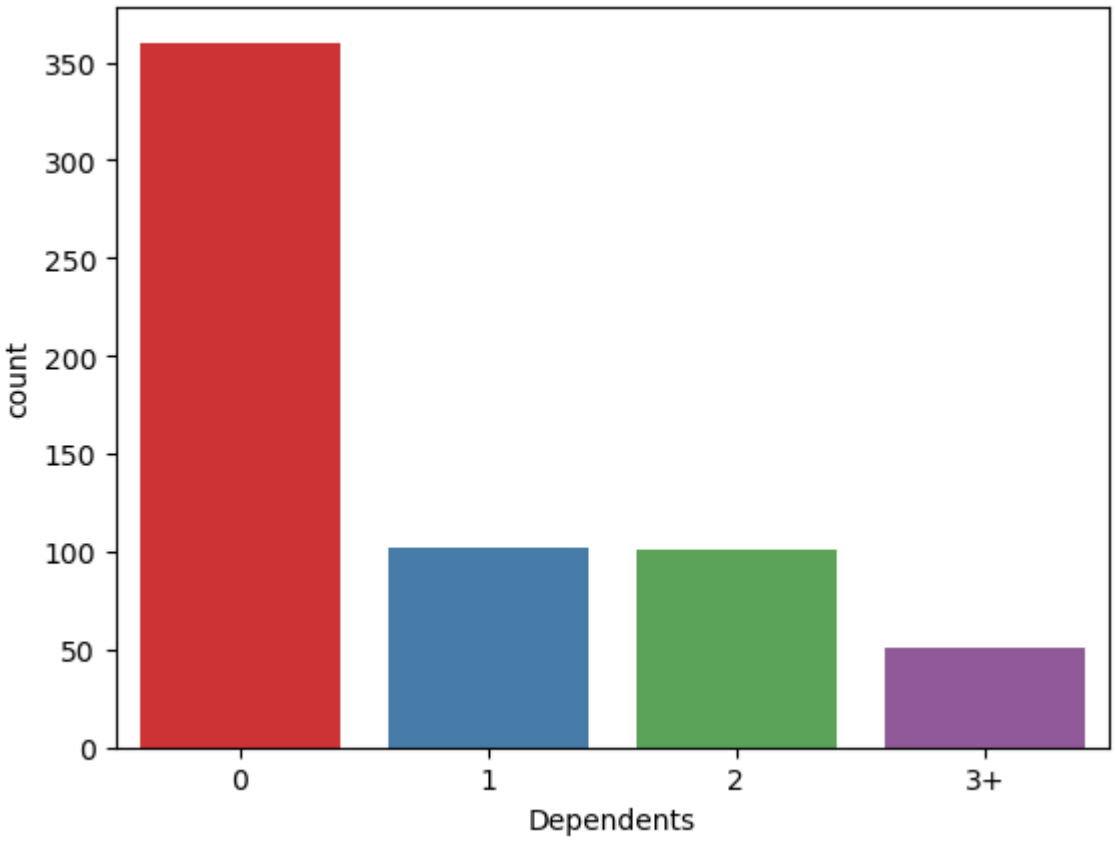
Out[35]:
```
<Axes: xlabel='Married', ylabel='count'>
```



In [36]:
```
print("Who take loan as group by Dependents:")
print(df['Dependents'].value_counts())
sns.countplot(x='Dependents', data=df, palette= 'Set1')
```

```
Who take loan as group by Dependents:
0     360
1     102
2     101
3+     51
Name: Dependents, dtype: int64
```

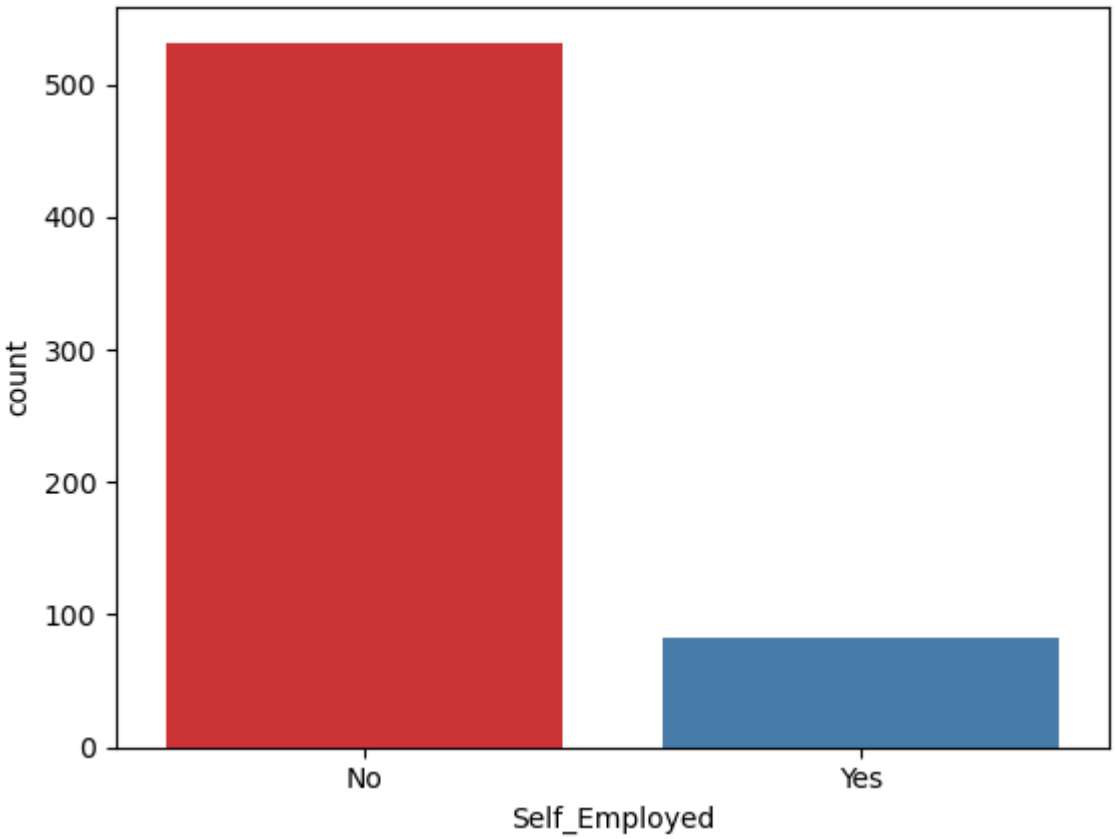
Out[36]: <Axes: xlabel='Dependents', ylabel='count'>



In [37]:
```
print("Who take loan as group by self employed:")
print(df['Self_Employed'].value_counts())
sns.countplot(x='Self_Employed', data=df, palette= 'Set1')
```

```
Who take loan as group by self employed:
No     532
Yes     82
Name: Self_Employed, dtype: int64
```

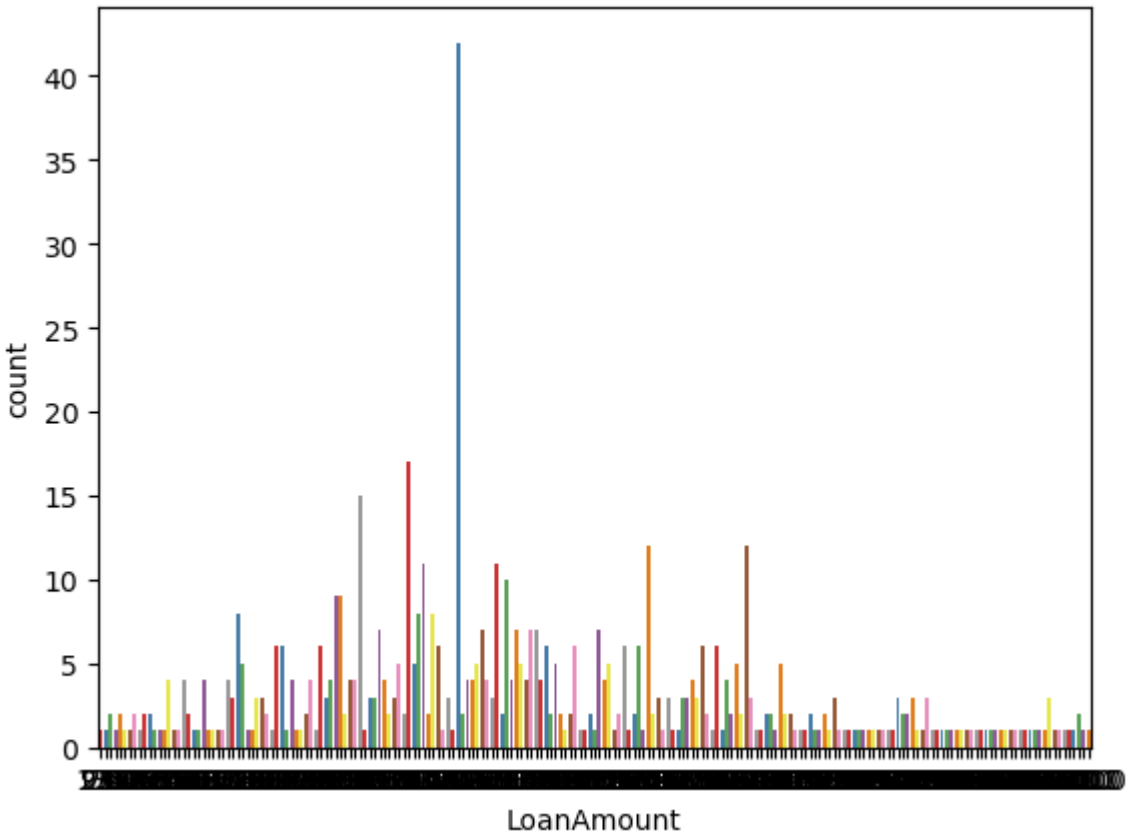
Out[37]:
```
<Axes: xlabel='Self_Employed', ylabel='count'>
```



In [38]:
```
print("Who take loan as group by LoanAmount:")
print(df['LoanAmount'].value_counts())
sns.countplot(x='LoanAmount', data=df, palette= 'Set1')
```

```
Who take loan as group by LoanAmount:
120.0    42
110.0    17
100.0    15
160.0    12
187.0    12
         ..
240.0     1
214.0     1
59.0      1
166.0     1
253.0     1
Name: LoanAmount, Length: 203, dtype: int64
```

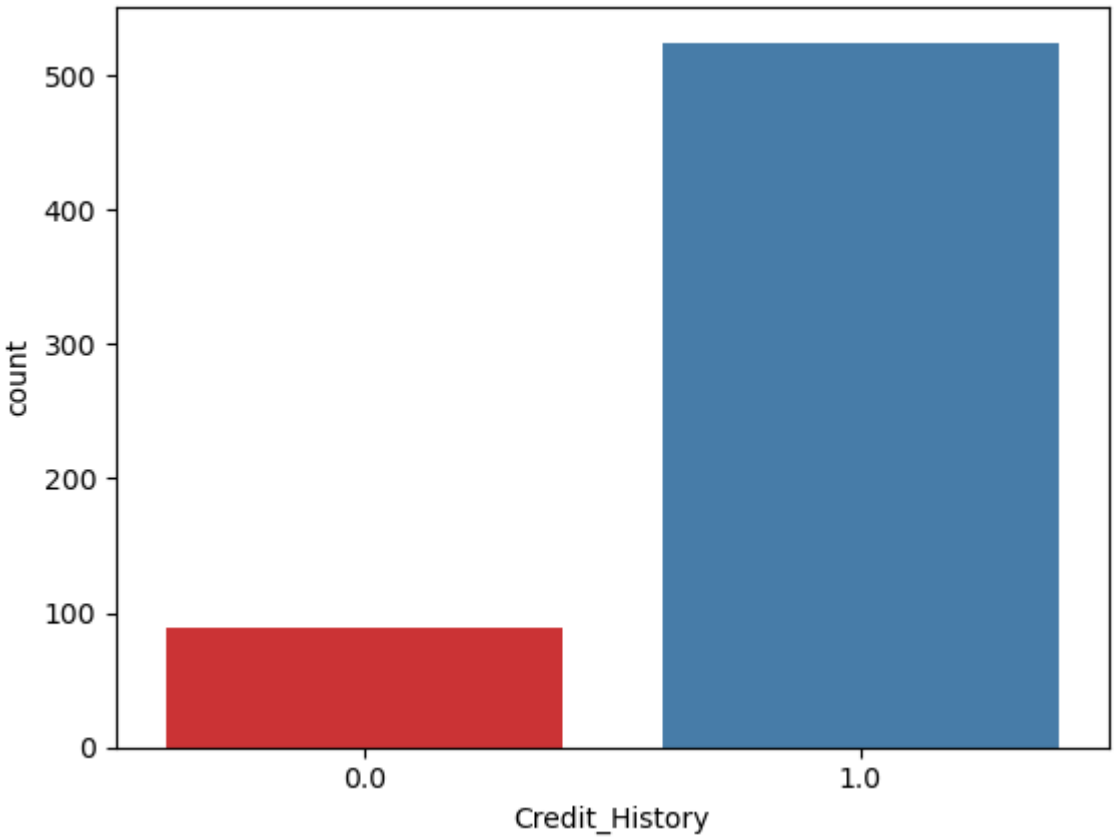
Out[38]: <Axes: xlabel='LoanAmount', ylabel='count'>



In [40]:
```
print("Who take loan on basis of Credit History group by Credit_History:")
print(df['Credit_History'].value_counts())
sns.countplot(x='Credit_History', data=df, palette= 'Set1')
```

```
Who take loan on basis of Credit History group by Credit_History:
1.0    525
0.0     89
Name: Credit_History, dtype: int64
```

Out[40]:
```
<Axes: xlabel='Credit_History', ylabel='count'>
```



In [52]:
```
# Scikit learn library for training and test
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_sta

from sklearn.preprocessing import LabelEncoder
LabelEncode_X = LabelEncoder()
```

In [60]:
```
for i in range(0,5):
    X_train[:,i]= LabelEncode_X.fit_transform(X_train[:,i])
    X_train[:,7]= LabelEncode_X.fit_transform(X_train[:,7])

X_train
```

Out[60]:
```
array([[1, 1, 0, ..., 1.0, 4.875197323201151, 267],
       [1, 0, 1, ..., 1.0, 5.278114659230517, 407],
       [1, 1, 0, ..., 0.0, 5.003946305945459, 249],
       ...,
       [1, 1, 3, ..., 1.0, 5.298317366548036, 363],
       [1, 1, 0, ..., 1.0, 5.075173815233827, 273],
       [0, 1, 0, ..., 1.0, 5.204006687076795, 301]], dtype=object)
```

In [61]:
```
LabelEncode_y = LabelEncoder()
y_train = LabelEncode_y.fit_transform(y_train)

y_train
```

Out[61]:
```
array([1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1,
       0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1,
       1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0,
       1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0,
       1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1,
       0, 1, 1, 1, 1, 1, 1, 0, 0, 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, 0, 0, 1, 1, 1, 1, 0,
       0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1,
       0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1,
       0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1,
       1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1,
       1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1,
       1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0,
       1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0,
       1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1,
       1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1,
       1, 1, 1, 0, 1, 0, 1], dtype=int64)
```

In [62]:
```
for i in range(0,5):
    X_test[:,i]= LabelEncode_X.fit_transform(X_test[:,i])
    X_test[:,7]= LabelEncode_X.fit_transform(X_test[:,7])

X_test
```

Out[62]:
```
array([[1, 0, 0, 0, 5, 1.0, 4.430816798843313, 85],
       [0, 0, 0, 0, 5, 1.0, 4.718498871295094, 28],
       [1, 1, 0, 0, 5, 1.0, 5.780743515792329, 104],
       [1, 1, 0, 0, 5, 1.0, 4.700480365792417, 80],
       [1, 1, 2, 0, 5, 1.0, 4.574710978503383, 22],
       [1, 1, 0, 1, 3, 0.0, 5.10594547390058, 70],
       [1, 1, 3, 0, 3, 1.0, 5.056245805348308, 77],
       [1, 0, 0, 0, 5, 1.0, 6.003887067106539, 114],
       [1, 0, 0, 0, 5, 0.0, 4.820281565605037, 53],
       [1, 1, 0, 0, 5, 1.0, 4.852030263919617, 55],
       [0, 0, 0, 0, 5, 1.0, 4.430816798843313, 4],
       [1, 1, 1, 0, 5, 1.0, 4.553876891600541, 2],
       [0, 0, 0, 0, 5, 1.0, 5.634789603169249, 96],
       [1, 1, 2, 0, 5, 1.0, 5.4638318050256105, 97],
       [1, 1, 0, 0, 5, 1.0, 4.564348191467836, 117],
       [1, 1, 1, 0, 5, 1.0, 4.204692619390966, 22],
       [1, 0, 1, 1, 5, 1.0, 5.247024072160486, 32],
       [1, 0, 0, 1, 5, 1.0, 4.882801922586371, 25],
       [0, 0, 0, 0, 5, 1.0, 4.532599493153256, 1],
       [1, 1, 0, 1, 5, 0.0, 5.198497031265826, 44],
       [0, 1, 0, 0, 5, 0.0, 4.787491742782046, 71],
       [1, 1, 0, 0, 5, 1.0, 4.962844630259907, 43],
       [1, 1, 2, 0, 5, 1.0, 4.68213122712422, 91],
       [1, 1, 2, 0, 5, 1.0, 5.10594547390058, 111],
       [1, 1, 0, 0, 5, 1.0, 4.060443010546419, 35],
       [1, 1, 1, 0, 5, 1.0, 5.521460917862246, 94],
       [1, 0, 0, 0, 5, 1.0, 5.231108616854587, 98],
       [1, 1, 0, 0, 5, 1.0, 5.231108616854587, 110],
       [1, 1, 3, 0, 5, 0.0, 4.852030263919617, 41],
       [0, 0, 0, 0, 5, 0.0, 4.634728988229636, 50],
       [1, 1, 0, 0, 5, 1.0, 5.429345628954441, 99],
       [1, 0, 0, 1, 5, 1.0, 3.871201010907891, 46],
       [1, 1, 1, 1, 5, 1.0, 4.499809670330265, 52],
       [1, 1, 0, 0, 5, 1.0, 5.19295685089021, 102],
       [1, 1, 0, 0, 5, 1.0, 4.787491742782046, 95],
       [0, 1, 0, 1, 5, 0.0, 5.181783550292085, 57],
       [1, 1, 0, 0, 5, 1.0, 5.147494476813453, 65],
       [1, 0, 0, 1, 5, 1.0, 4.836281906951478, 39],
       [1, 1, 0, 0, 5, 1.0, 4.852030263919617, 75],
       [1, 1, 2, 1, 5, 1.0, 4.68213122712422, 24],
       [0, 0, 0, 0, 5, 1.0, 4.382026634673881, 9],
       [1, 1, 3, 0, 5, 0.0, 4.812184355372417, 68],
       [1, 1, 2, 0, 2, 1.0, 2.833213344056216, 0],
       [1, 1, 1, 1, 5, 1.0, 5.062595033026967, 67],
       [1, 0, 0, 0, 5, 1.0, 4.330733340286331, 21],
       [1, 0, 0, 0, 5, 1.0, 5.231108616854587, 113],
       [1, 1, 1, 0, 5, 1.0, 4.7535901911063645, 18],
       [0, 0, 0, 0, 5, 1.0, 4.74493212836325, 37],
       [1, 1, 1, 0, 5, 1.0, 4.852030263919617, 72],
       [1, 0, 0, 0, 5, 1.0, 4.941642422609304, 78],
       [1, 1, 3, 1, 5, 1.0, 4.30406509320417, 8],
       [1, 1, 0, 0, 5, 1.0, 4.867534450455582, 84],
       [1, 1, 0, 1, 5, 1.0, 4.672828834461906, 31],
       [1, 0, 0, 0, 5, 1.0, 4.787491742782046, 61],
       [1, 1, 0, 0, 5, 1.0, 4.718498871295094, 19],
       [1, 1, 0, 0, 5, 1.0, 5.556828061699537, 107],
       [1, 1, 0, 0, 5, 1.0, 4.553876891600541, 34],
       [1, 0, 0, 1, 5, 1.0, 4.890349128221754, 74],
       [1, 1, 2, 0, 5, 1.0, 5.123963979403259, 62],
       [1, 0, 0, 0, 5, 1.0, 4.787491742782046, 27],
       [0, 0, 0, 0, 5, 0.0, 4.919980925828125, 108],
       [0, 0, 0, 0, 5, 1.0, 5.365976015021851, 103],
       [1, 1, 0, 1, 5, 1.0, 4.74493212836325, 38],
       [0, 0, 0, 0, 5, 0.0, 4.330733340286331, 13],
```

```
       [1, 1, 2, 0, 5, 1.0, 4.890349128221754, 69],
       [1, 1, 1, 0, 5, 1.0, 5.752572638825633, 112],
       [1, 1, 0, 0, 5, 1.0, 5.075173815233827, 73],
       [1, 0, 0, 0, 5, 1.0, 4.912654885736052, 47],
       [1, 1, 0, 0, 5, 1.0, 5.204006687076795, 81],
       [1, 0, 0, 1, 5, 1.0, 4.564348191467836, 60],
       [1, 0, 0, 0, 5, 1.0, 4.204692619390966, 83],
       [0, 1, 0, 0, 5, 1.0, 4.867534450455582, 5],
       [1, 1, 2, 1, 5, 1.0, 5.056245805348308, 58],
       [1, 1, 1, 1, 3, 1.0, 4.919980925828125, 79],
       [0, 1, 0, 0, 5, 1.0, 4.969813299576001, 54],
       [1, 1, 0, 1, 4, 1.0, 4.820281565605037, 56],
       [1, 0, 0, 0, 5, 1.0, 4.499809670330265, 120],
       [1, 0, 3, 0, 5, 1.0, 5.768320995793772, 118],
       [1, 1, 2, 0, 5, 1.0, 4.718498871295094, 101],
       [0, 0, 0, 0, 5, 0.0, 4.7535901911063645, 26],
       [0, 0, 0, 0, 6, 1.0, 4.727387818712341, 33],
       [1, 1, 1, 0, 5, 1.0, 6.214608098422191, 119],
       [0, 0, 0, 0, 5, 1.0, 5.267858159063328, 89],
       [1, 1, 2, 0, 5, 1.0, 5.231108616854587, 92],
       [1, 0, 0, 0, 6, 1.0, 4.2626798770413155, 6],
       [1, 1, 0, 0, 0, 1.0, 4.709530201312334, 90],
       [1, 1, 0, 0, 5, 1.0, 4.700480365792417, 45],
       [1, 1, 2, 0, 5, 1.0, 5.298317366548036, 109],
       [1, 0, 1, 0, 3, 1.0, 4.727387818712341, 17],
       [1, 1, 1, 0, 5, 1.0, 4.6443908991413725, 36],
       [0, 1, 0, 1, 5, 1.0, 4.605170185988092, 16],
       [1, 0, 0, 0, 5, 1.0, 4.30406509320417, 7],
       [1, 1, 1, 0, 1, 1.0, 5.147494476813453, 88],
       [1, 1, 3, 0, 4, 0.0, 5.19295685089021, 87],
       [0, 0, 0, 0, 5, 1.0, 4.2626798770413155, 3],
       [1, 0, 0, 1, 3, 0.0, 4.836281906951478, 59],
       [1, 0, 0, 0, 3, 1.0, 5.1647859739235145, 82],
       [1, 0, 0, 0, 5, 1.0, 4.969813299576001, 66],
       [1, 1, 2, 1, 5, 1.0, 4.394449154672439, 51],
       [1, 1, 1, 0, 5, 1.0, 5.231108616854587, 100],
       [1, 1, 0, 0, 5, 1.0, 5.351858133476067, 93],
       [1, 1, 0, 0, 5, 1.0, 4.605170185988092, 15],
       [1, 1, 2, 0, 5, 1.0, 4.787491742782046, 106],
       [1, 0, 0, 0, 3, 1.0, 4.787491742782046, 105],
       [1, 1, 3, 0, 5, 1.0, 4.852030263919617, 64],
       [1, 0, 0, 0, 5, 1.0, 4.8283137373023015, 49],
       [1, 0, 0, 1, 5, 1.0, 4.6443908991413725, 42],
       [0, 0, 0, 0, 5, 1.0, 4.477336814478207, 10],
       [1, 1, 0, 1, 5, 1.0, 4.553876891600541, 20],
       [1, 1, 3, 1, 3, 1.0, 4.394449154672439, 14],
       [1, 0, 0, 0, 5, 1.0, 5.298317366548036, 76],
       [0, 0, 0, 0, 5, 1.0, 4.90527477843843, 11],
       [1, 0, 0, 0, 6, 1.0, 4.727387818712341, 18],
       [1, 1, 2, 0, 5, 1.0, 4.248495242049359, 23],
       [1, 1, 0, 1, 5, 0.0, 5.303304908059076, 63],
       [1, 1, 0, 0, 3, 0.0, 4.499809670330265, 48],
       [0, 0, 0, 0, 5, 1.0, 4.430816798843313, 30],
       [1, 0, 0, 0, 5, 1.0, 4.897839799950911, 29],
       [1, 1, 2, 0, 5, 1.0, 5.170483995038151, 86],
       [1, 1, 3, 0, 5, 1.0, 4.867534450455582, 115],
       [1, 1, 0, 0, 5, 1.0, 6.077642243349034, 116],
       [1, 1, 3, 1, 3, 0.0, 4.248495242049359, 40],
       [1, 1, 1, 0, 5, 1.0, 4.564348191467836, 12]], dtype=object)
```

In [63]:
```
LabelEncode_y = LabelEncoder()
y_test = LabelEncode_y.fit_transform(y_test)

y_test
```

Out[63]:
```
array([1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1,
       1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0,
       1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1], dtype=int64)
```

In [64]:
```
# importing standard scalar for the further processing
#Standardizing training and test data(i.e. each features has standard deviation =1,
from sklearn.preprocessing import StandardScaler

ss = StandardScaler()
X_train = ss.fit_transform(X_train)
X_test= ss.fit_transform(X_test)
```

In [68]:
```
from sklearn.ensemble import RandomForestClassifier
#Random forest is ML model(by using classification and regression techninque) used
rc_clf = RandomForestClassifier()
rc_clf.fit(X_train, y_train)
```

Out[68]:
```
▾ RandomForestClassifier

RandomForestClassifier()
```

In [69]:
```
from sklearn import metrics

y_pred = rc_clf.predict(X_test)
print("Accuracy of Random Forest Clasifier is", metrics.accuracy_score(y_pred, y_te

y_pred
```

Out[69]:
```
Accuracy of Random Forest Clasifier is 0.7642276422764228
array([0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1,
       1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1,
       1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1], dtype=int64)
```

In [70]:
```
#Accuracy of Random Forest Clasifier is 0.7642276422764228
#i.e. predictive model is 76.42% accurate
```

In [71]:
```
#Prediction through Gausian Naive Bayes
from sklearn.naive_bayes import GaussianNB

nb_clf = GaussianNB()
nb_clf.fit(X_train, y_train)
```

Out[71]:
```
▾ GaussianNB

GaussianNB()
```

In [73]:
```
y_pred = nb_clf.predict(X_test)
#Accuracy of prediction through Gaussian Naive Bayes
print("Accuracy of Gaussian NB is", metrics.accuracy_score(y_pred, y_test))

y_pred
```

```
Accuracy of Gaussian NB is 0.8292682926829268
```

Out[73]:
```
array([1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 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, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 0, 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], dtype=int64)
```

In [76]:
```
from sklearn.tree import DecisionTreeClassifier
dt_clf = DecisionTreeClassifier()
dt_clf.fit(X_train, y_train)
```

Out[76]:
```
▾ DecisionTreeClassifier

DecisionTreeClassifier()
```

In [79]:
```
y_pred = dt_clf.predict(X_test)
#Accuracy of prediction through Decission tree
print("Accuracy of Decission Tree is", metrics.accuracy_score(y_pred, y_test))

y_pred
```

Out[79]:
```
Accuracy of Decission Tree is 0.7235772357723578
array([0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1,
       1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1,
       1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1,
       1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0,
       1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1], dtype=int64)
```

In [82]:
```
from sklearn.neighbors import KNeighborsClassifier
kn_clf = KNeighborsClassifier()
kn_clf.fit(X_train, y_train)
```

Out[82]:
```
▾ KNeighborsClassifier

KNeighborsClassifier()
```

In [83]:
```
y_pred = kn_clf.predict(X_test)
#Accuracy of prediction through K-Nearest Neighbors
print("Accuracy of K-Nearest Neighbors is", metrics.accuracy_score(y_pred, y_test))

y_pred
```

Out[83]:
```
Accuracy of K-Nearest Neighbors is 0.8048780487804879
array([0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 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, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1], dtype=int64)
```

In [ ]:
```
# So, after seeing all of the above prediction system we can conclude that Gaussian
# So, we use Gaussian Naive Bayes for Loan Approval
```