In [3]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import svm
```

In [4]:

```
auto = pd.read_excel('loan Presiction.xlsx')
```

In [5]:

```
auto.head()
```

Out[5]:

ried	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amo
No	0	Graduate	No	5849	0.0	NaN	
Yes	1	Graduate	No	4583	1508.0	128.0	
Yes	0	Graduate	Yes	3000	0.0	66.0	
Yes	0	Not Graduate	No	2583	2358.0	120.0	
No	0	Graduate	No	6000	0.0	141.0	

In [6]:

```
auto.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 [7]:

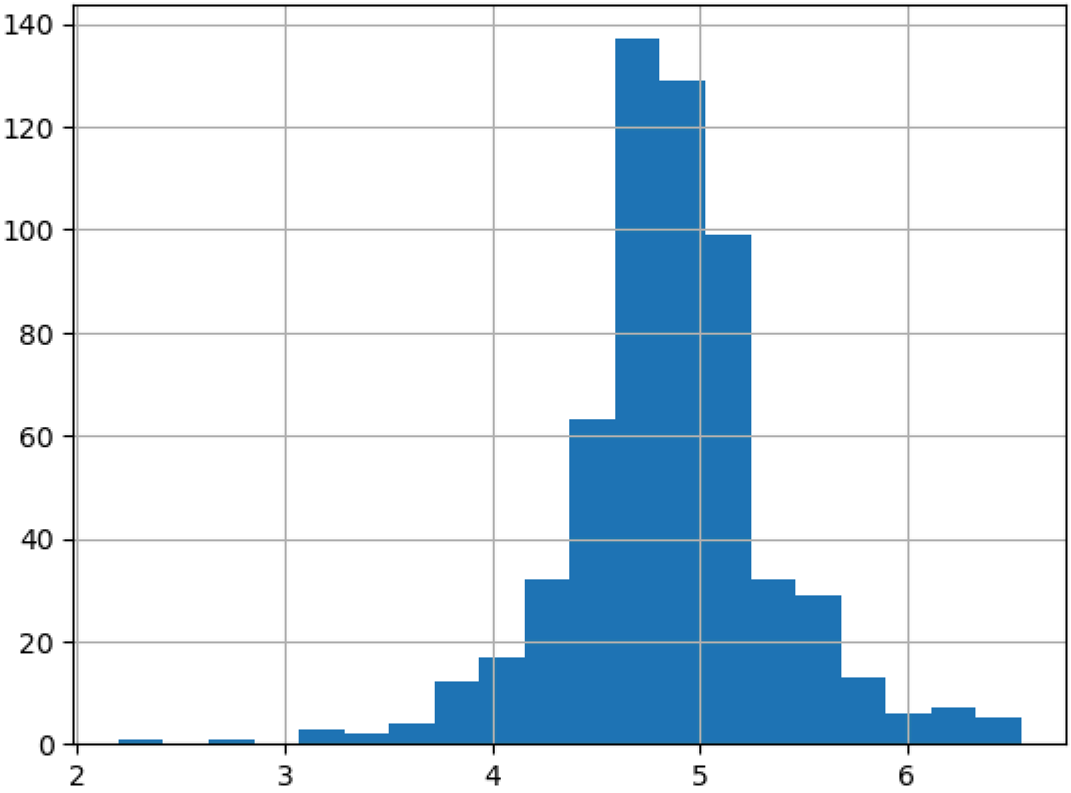
```
auto.isnull().sum()
```

Out[7]:

```
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 [8]: auto['Loan_Amount_log']=np.log (auto['LoanAmount'])
        auto['Loan_Amount_log'].hist(bins=20)
```

```
Out[8]: <Axes: >
```



```
In [9]: auto.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
        Loan_Amount_log      22
        dtype: int64
```

In [10]:

```
auto['Gender'].fillna(auto['Gender'].mode()[0],inplace=True)
auto['Married'].fillna(auto['Married'].mode()[0],inplace=True)
auto['Self_Employed'].fillna(auto['Self_Employed'].mode()[0],inplace=True)
auto['Dependents'].fillna(auto['Dependents'].mode()[0],inplace=True)

auto.LoanAmount=auto.LoanAmount .fillna(auto.LoanAmount.mean())
auto.Loan_Amount_log =auto.Loan_Amount_log .fillna(auto.Loan_Amount_log .mean())

auto['Loan_Amount_Term'].fillna(auto['Loan_Amount_Term'].mode()[0],inplace=True)
auto['Credit_History'].fillna(auto['Credit_History'].mode()[0],inplace=True)
auto.isnull().sum()
```

Out[10]:

```
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
Loan_Amount_log      0
dtype: int64
```

In [11]:

```
x=auto.iloc[:,np.r_[1:5,9:11,13:14]].values
y=auto.iloc[:,12].values

x
```

Out[11]:

```
array([['Male', 'No', 0, ..., 360.0, 1.0, 4.857444178729352],
       ['Male', 'Yes', 1, ..., 360.0, 1.0, 4.852030263919617],
       ['Male', 'Yes', 0, ..., 360.0, 1.0, 4.189654742026425],
       ...,
       ['Male', 'Yes', 1, ..., 360.0, 1.0, 5.53338948872752],
       ['Male', 'Yes', 2, ..., 360.0, 1.0, 5.231108616854587],
       ['Female', 'No', 0, ..., 360.0, 0.0, 4.890349128221754]],
      dtype=object)
```

```
In [12]: y
```

```
Out[12]: 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)
```

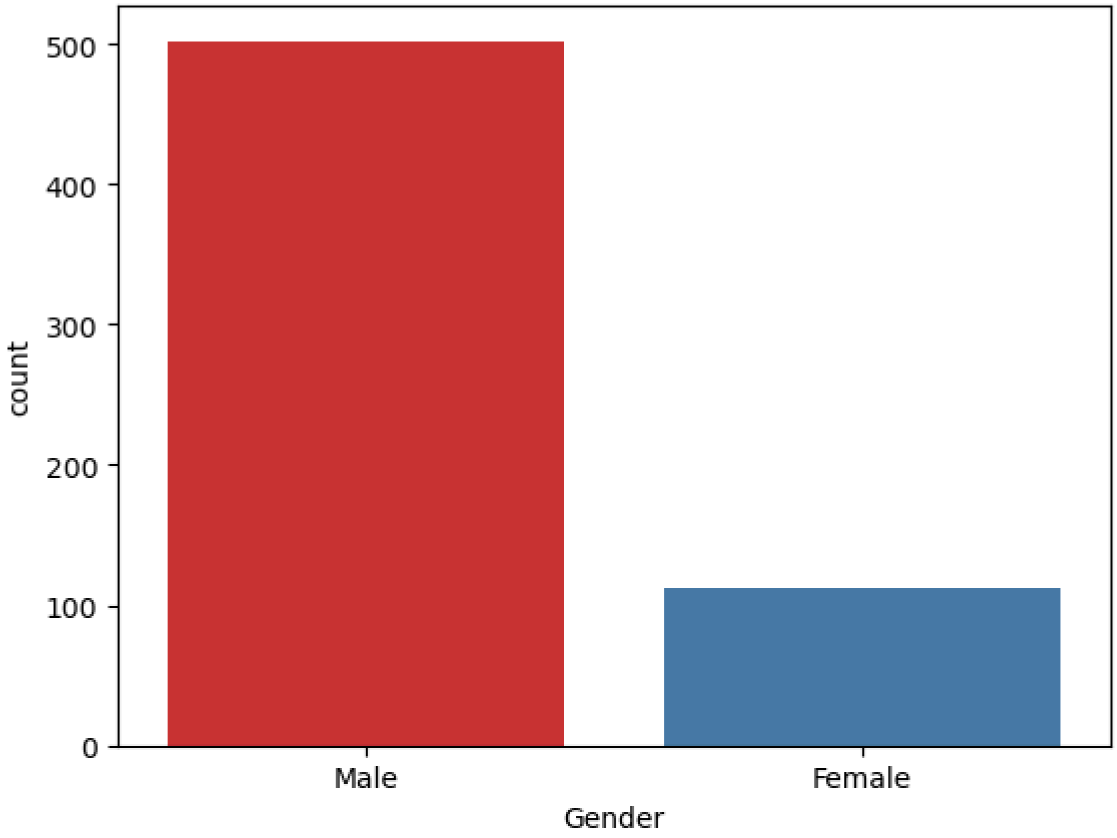
```
In [13]: print("per of missing gender is %2f%%" %((auto['Gender'].isnull().sum()/auto.shape[0]
```

```
per of missing gender is 0.000000%
```

In [14]:
```python
print("number of people who take loan as group by gender:")
print(auto['Gender'].value_counts())
sns.countplot(x='Gender',data=auto,palette='Set1')
```

```
number of people who take loan as group by gender:
Gender
Male      502
Female    112
Name: count, dtype: int64
```

Out[14]:
```
<Axes: xlabel='Gender', ylabel='count'>
```



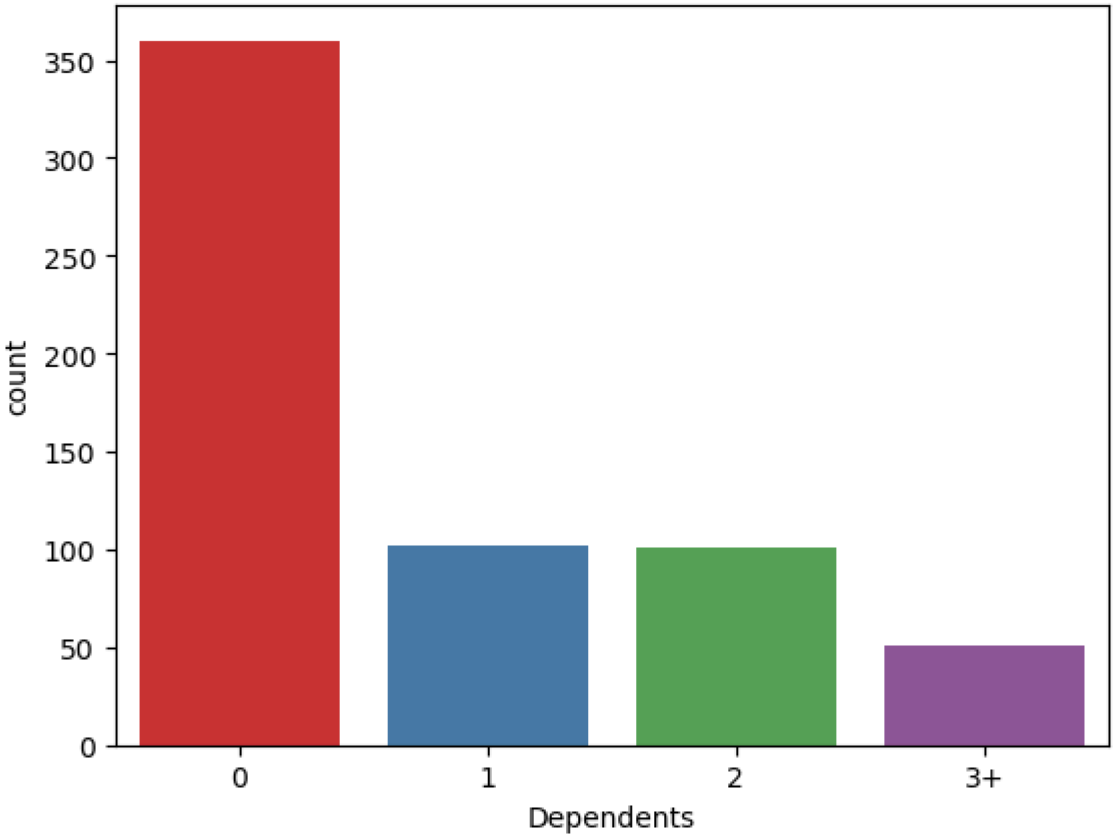
In [15]:

```
print("number of people who take loan as group by Dependents:")
print(auto['Dependents'].value_counts())
sns.countplot(x='Dependents',data=auto,palette='Set1')
```

```
number of people who take loan as group by Dependents:
Dependents
0     360
1     102
2     101
3+     51
Name: count, dtype: int64
```

Out[15]:

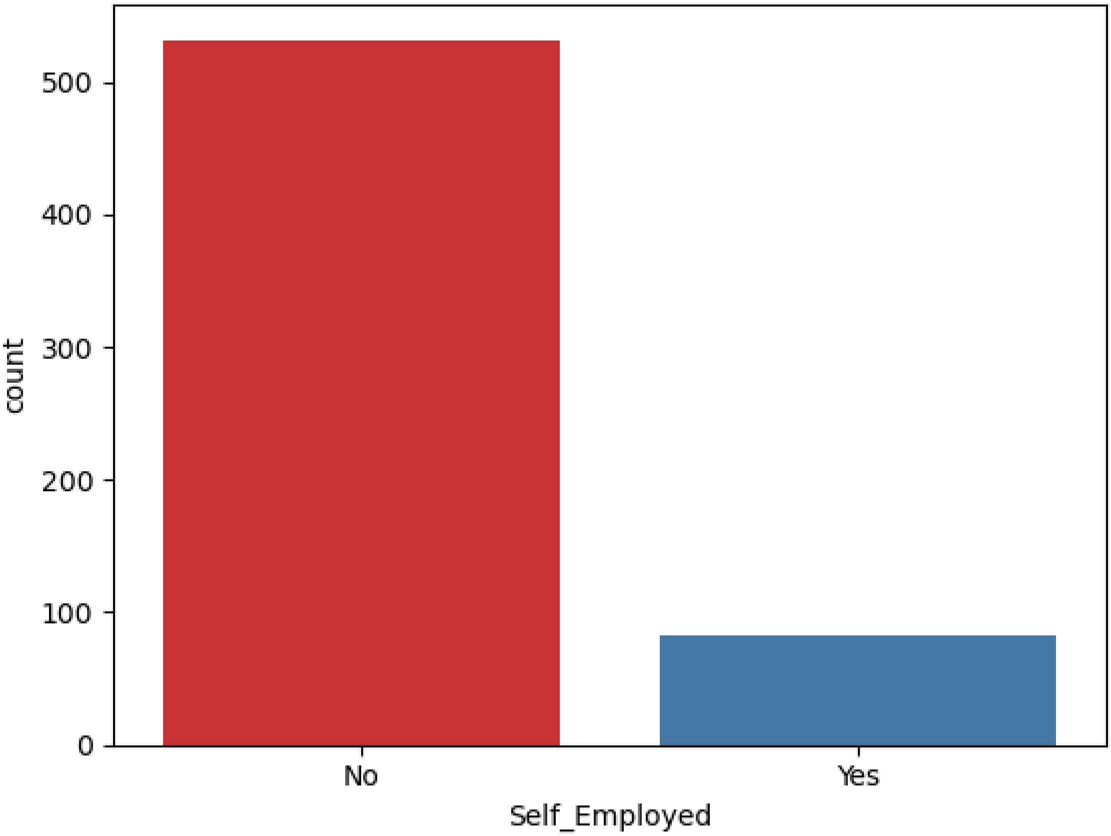
```
<Axes: xlabel='Dependents', ylabel='count'>
```



```
In [16]: print("number of people who take loan as group by Self_Employed:")
         print(auto['Self_Employed'].value_counts())
         sns.countplot(x='Self_Employed',data=auto,palette='Set1')
```

```
number of people who take loan as group by Self_Employed:
Self_Employed
No     532
Yes     82
Name: count, dtype: int64
```

```
Out[16]: <Axes: xlabel='Self_Employed', ylabel='count'>
```



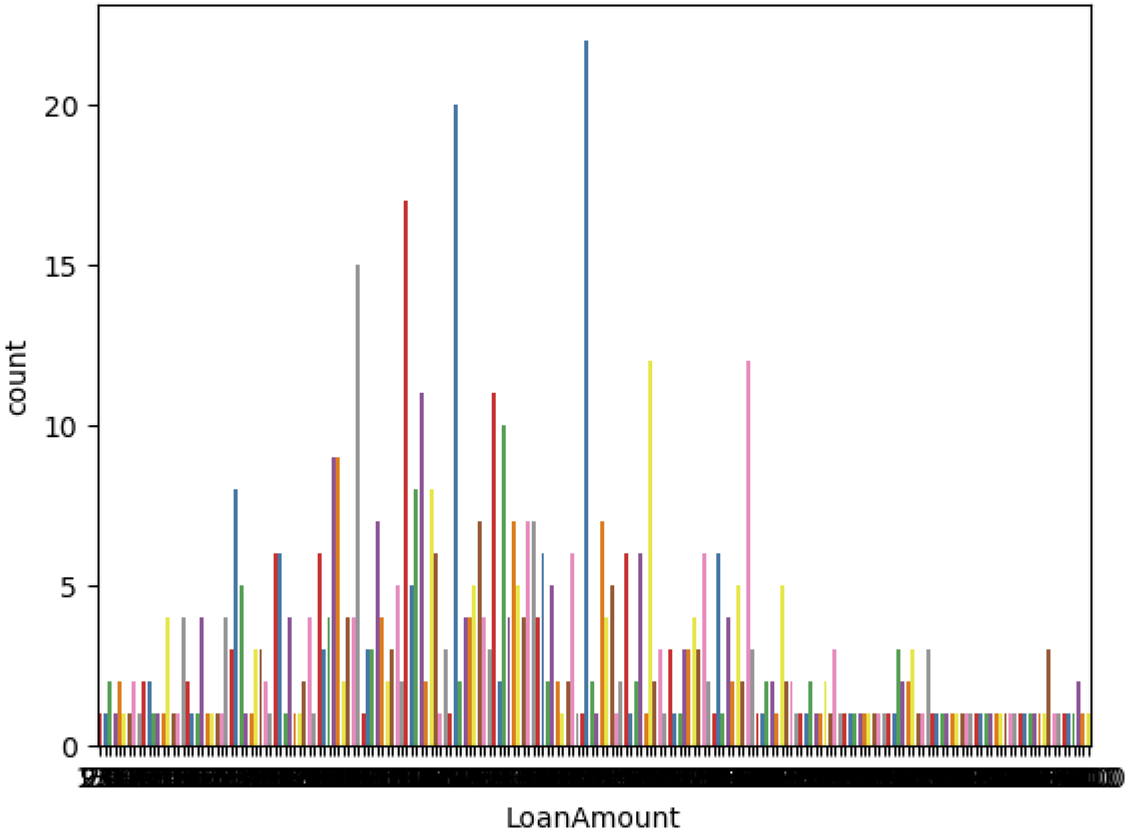
In [17]:

```
print("number of people who take loan as group by LoanAmount:")
print(auto['LoanAmount'].value_counts())
sns.countplot(x='LoanAmount',data=auto,palette='Set1')
```

```
number of people who take loan as group by LoanAmount:
LoanAmount
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: count, Length: 204, dtype: int64
```

Out[17]:

```
<Axes: xlabel='LoanAmount', ylabel='count'>
```



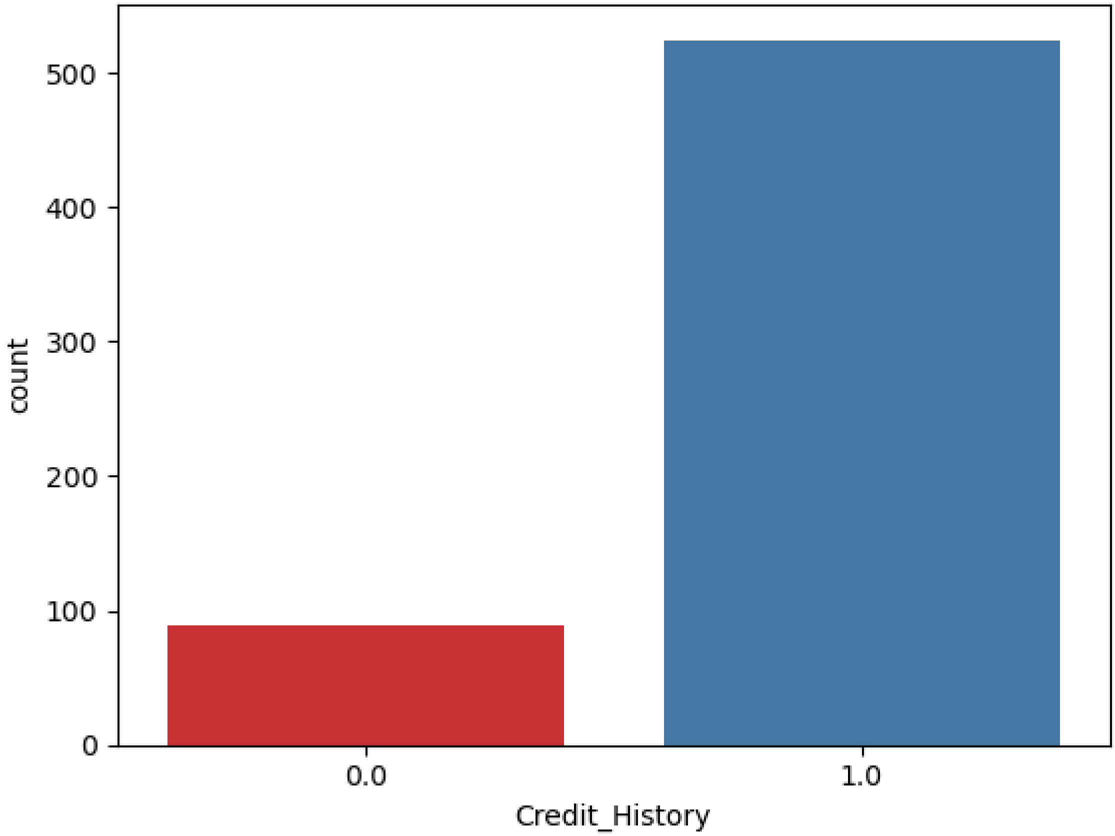
In [18]:

```
print("number of people who take loan as group by Credit_History:")
print(auto['Credit_History'].value_counts())
sns.countplot(x='Credit_History',data=auto,palette='Set1')
```

```
number of people who take loan as group by Credit_History:
Credit_History
1.0    525
0.0     89
Name: count, dtype: int64
```

Out[18]:

```
<Axes: xlabel='Credit_History', ylabel='count'>
```



In [29]:

```
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_state=0)

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

In [46]:

```

import numpy as np

# Assuming x_train is a numpy array and LabelEncoder_x is already defined
for i in range(0, 5):
    # Convert the entire column to string type before encoding
    x_train[:, i] = x_train[:, i].astype(str)
    x_train[:, i] = LabelEncoder_x.fit_transform(x_train[:, i])

# Convert the last column to string type before encoding
x_train[:, -1] = x_train[:, -1].astype(str)
x_train[:, -1] = LabelEncoder_x.fit_transform(x_train[:, -1])
```

In [47]:
```
LabelEncoder_y=LabelEncoder()
y_train=LabelEncoder_y.fit_transform(y_train)

y_train
```

Out[47]:
```
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, 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])
```

In [50]:
```
import numpy as np

# Assuming x_test is a numpy array and LabelEncoder_x is already defined
for i in range(0, 5):
    # Convert the entire column to string type before encoding
    x_test[:, i] = x_test[:, i].astype(str)
    x_test[:, i] = LabelEncoder_x.fit_transform(x_test[:, i])

# Convert the last column to string type before encoding
x_test[:, -1] = x_test[:, -1].astype(str)
x_test[:, -1] = LabelEncoder_x.fit_transform(x_test[:, -1])
```

In [51]:
```
LabelEncoder_y=LabelEncoder()
y_test=LabelEncoder_y.fit_transform(y_test)

y_test
```

Out[51]:
```
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])
```

In [56]:

```
from sklearn.preprocessing import StandardScaler
import numpy as np

# Assuming x_train and x_test are defined numpy arrays
ss = StandardScaler()

# Fit and transform the training data
x_train = ss.fit_transform(x_train)

# Ensure x_test is of a numeric type
x_test = x_test.astype(float)

# Check for NaNs or infinite values in the test data and handle them
if np.any(np.isnan(x_test)) or not np.all(np.isfinite(x_test)):
    x_test = np.nan_to_num(x_test)

# Transform the test data
x_test = ss.transform(x_test)

# Print the transformed x_train and x_test arrays
print("Transformed x_train:", x_train)
print("Transformed x_test:", x_test)
```

```
   5.00000000e+00  1.00000000e+00  2.50000000e+01]
 [ 1.00000000e+00  1.00000000e+00  2.00000000e+00  2.33350338e-16
   4.00000000e+00  1.00000000e+00  4.00000000e+00]
 [ 1.00000000e+00  1.00000000e+00 -1.26171985e-16  1.00000000e+00
   4.00000000e+00  2.60031869e-16  6.10000000e+01]
 [ 1.00000000e+00  1.00000000e+00 -1.26171985e-16  2.33350338e-16
   2.00000000e+00  2.60031869e-16  1.20000000e+01]
 [-1.55566892e-16  1.66872626e-16 -1.26171985e-16  2.33350338e-16
   4.00000000e+00  1.00000000e+00  1.00000000e+01]
 [ 1.00000000e+00  1.66872626e-16 -1.26171985e-16  2.33350338e-16
   4.00000000e+00  1.00000000e+00  3.80000000e+01]
 [ 1.00000000e+00  1.00000000e+00  2.00000000e+00  2.33350338e-16
   4.00000000e+00  1.00000000e+00  5.20000000e+01]
 [ 1.00000000e+00  1.00000000e+00  3.00000000e+00  2.33350338e-16
   4.00000000e+00  1.00000000e+00  3.50000000e+01]
 [ 1.00000000e+00  1.00000000e+00 -1.26171985e-16  2.33350338e-16
   4.00000000e+00  1.00000000e+00  7.30000000e+01]
 [ 1.00000000e+00  1.00000000e+00  3.00000000e+00  1.00000000e+00
   2.00000000e+00  2.60031869e-16  4.00000000e+00]
```

In [57]:

```
from sklearn.ensemble import RandomForestClassifier
rf_clf = RandomForestClassifier()
rf_clf.fit(x_train,y_train)
```

Out[57]:

```
▾ RandomForestClassifier

RandomForestClassifier()
```

In [59]:
```
# Correct the import statement and the module name
from sklearn import metrics

# Assuming rf_clf is a trained random forest classifier, x_test and y_test are define
y_pred = rf_clf.predict(x_test)

# Print the accuracy of the random forest classifier
print("Accuracy of random forest classifier is", metrics.accuracy_score(y_pred, y_tes

# Print the predicted values
print("Predicted values:", y_pred)
```

```
Accuracy of random forest classifier is 0.7317073170731707
Predicted values: [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1]
```

In [60]:
```
from sklearn.naive_bayes import GaussianNB
nb_clf = GaussianNB()
nb_clf.fit(x_train,y_train)
```

Out[60]:
```
▾ GaussianNB
GaussianNB()
```

In [61]:
```
y_pred = nb_clf.predict(x_test)
print("acc of GaussianNB is % ", metrics.accuracy_score(y_pred,y_test))
```

```
acc of GaussianNB is %  0.7317073170731707
```

In [62]:
```
y_pred
```

Out[62]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

In [63]:
```
from sklearn.preprocessing import LabelEncoder

# Create a label encoder object
label_encoder = LabelEncoder()

# Assuming 'y_train' contains the categorical variable with 'Yes' and 'No'
# Encode the categorical variable
y_train_encoded = label_encoder.fit_transform(y_train)

# Now 'y_train_encoded' contains 0s and 1s instead of 'Yes' and 'No'
# Fit the DecisionTreeClassifier with the encoded labels
dt_clf.fit(x_train, y_train_encoded)
```

Out[63]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

In [65]:

```
from sklearn import metrics

# Assuming dt_clf is a trained Decision Tree Classifier, and x_test and y_test are de
y_pred = dt_clf.predict(x_test)

# Print the accuracy of the Decision Tree Classifier
print("Accuracy of DT is", metrics.accuracy_score(y_pred, y_test))
```

```
Accuracy of DT is 0.5934959349593496
```

In [66]:

```
y_pred
```

Out[66]:

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

In [68]:

```
from sklearn.neighbors import KNeighborsClassifier

# Create a KNeighborsClassifier object
kn_clf = KNeighborsClassifier()

# Assuming x_train and y_train are defined
# Fit the classifier to the training data
kn_clf.fit(x_train, y_train)
```

Out[68]:

```
▾ KNeighborsClassifier

KNeighborsClassifier()
```

In [69]:

```
# Assuming kn_clf is a trained KNeighborsClassifier, and x_test and y_test are define
y_pred = kn_clf.predict(x_test)

# Print the accuracy of the KNeighborsClassifier
print("Accuracy of KN is", metrics.accuracy_score(y_pred, y_test))
```

```
Accuracy of KN is 0.7317073170731707
```

In [ ]: