Loan Approval Prediction Using Machine Learning

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

df = pd.read_csv('loan-train.csv')
df.head()
```

```
    Loan_ID Gender Married Dependents     Education 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

   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

   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
```

```
df.shape
```

```
(614, 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
```

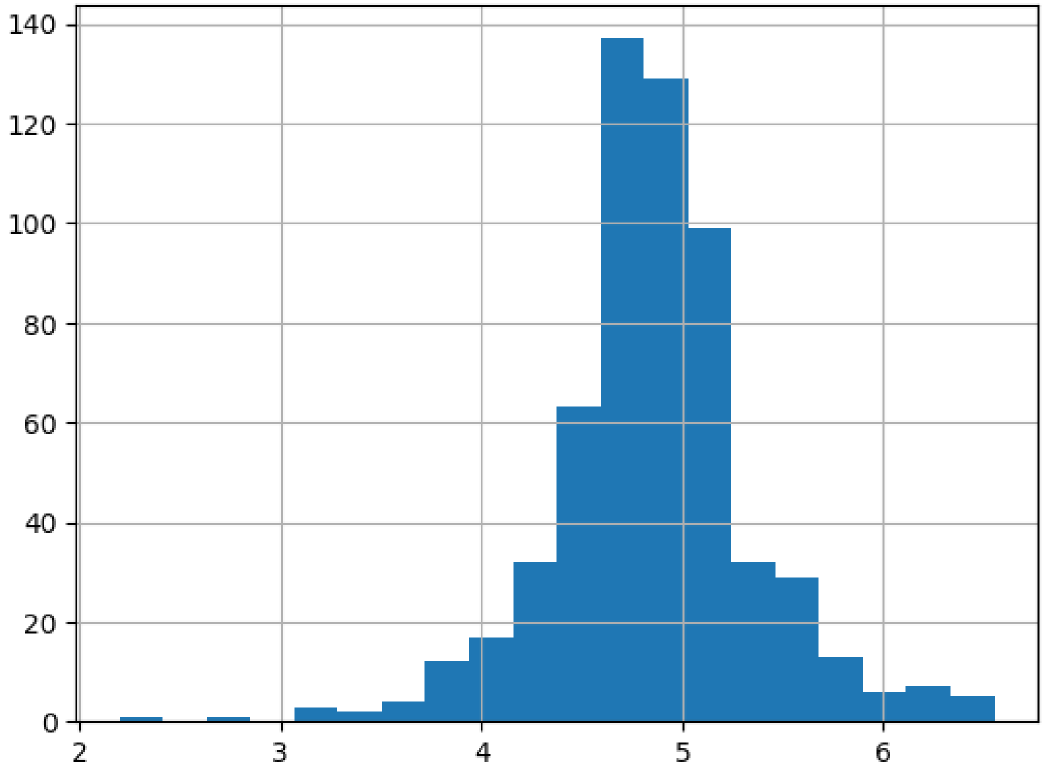
Checking for missing values

```
df.isnull().sum()
```

```
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
```

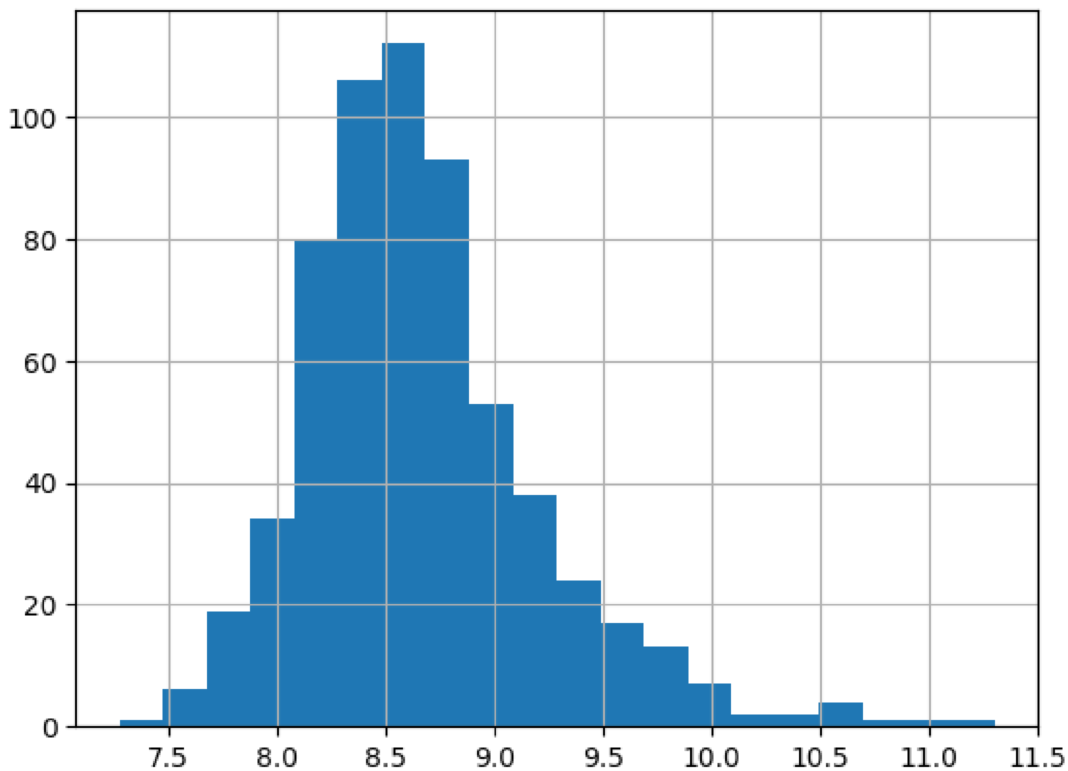
```
df['loanAmount_log'] = np.log(df['LoanAmount'])
df['loanAmount_log'].hist(bins=20)
```

```
<Axes: >
```



- Looks fine.

```
# Let's create and add acolumn for total income = applicant income +
co-applicant income

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

<Axes: >
```



Fill Null values

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

df.LoanAmount = df.LoanAmount.fillna(df.LoanAmount.mean())
df.loanAmount_log = df.LoanAmount.fillna(df.loanAmount_log.mean())

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.isnull().sum()
```

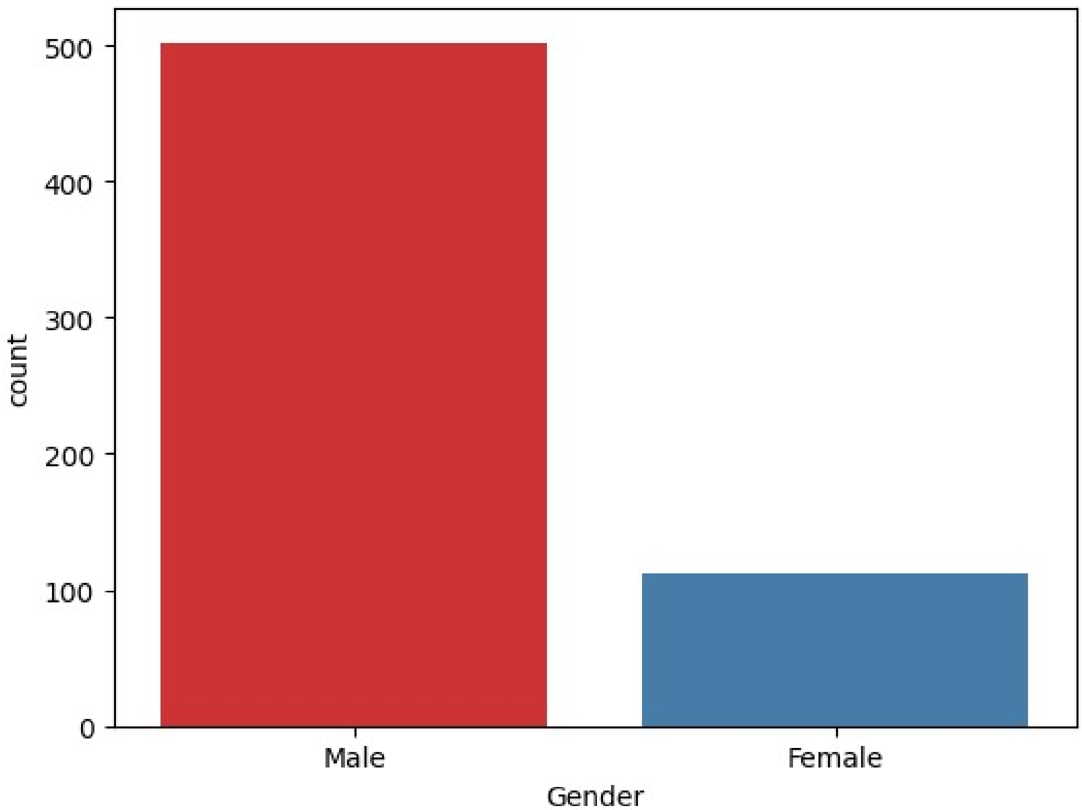
```
Loan_ID              0
Gender               0
Married              0
Dependents           0
Education            0
Self_Employed        0
ApplicantIncome      0
CoapplicantIncome    0
```

```
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
```

- No missing values now.

Some Exploration and Visualization

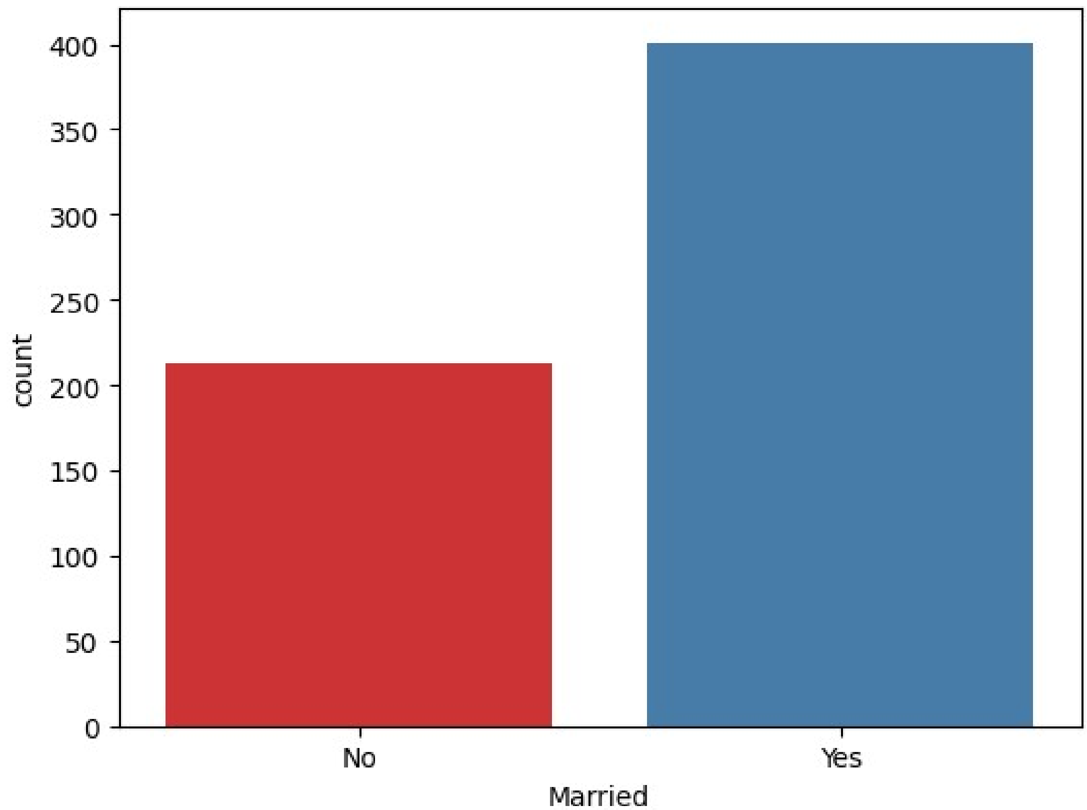
```
# Percentage of missing value in Gender

print("per of missing gender is %2f%%" %
((df['Gender'].isnull().sum()/df.shape[0])*100))

per of missing gender is 0.000000%

# Percentage of people of take loan by Gender

print("number of people who take loan as group by gender:")
print(df['Gender'].value_counts())

sns.countplot(x='Gender', data = df, palette = 'Set1')

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

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



- Number of observations in each category bins are displayed using bars.

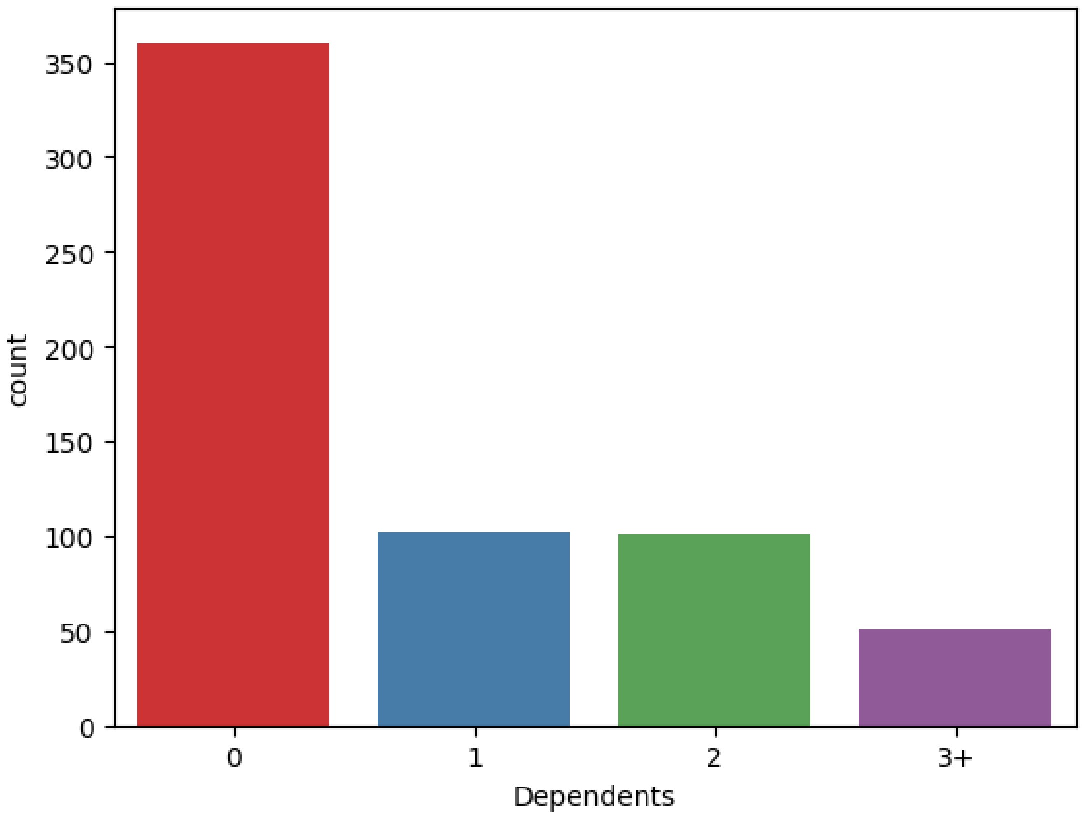
```
# Percentage of people of take loan by Marital Status

print("number of people who take loan as group by marital status:")
print(df['Married'].value_counts())

sns.countplot(x='Married', data = df, palette = 'Set1')
```

```
number of people who take loan as group by marital status:
Married
Yes    401
No     213
Name: count, dtype: int64

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



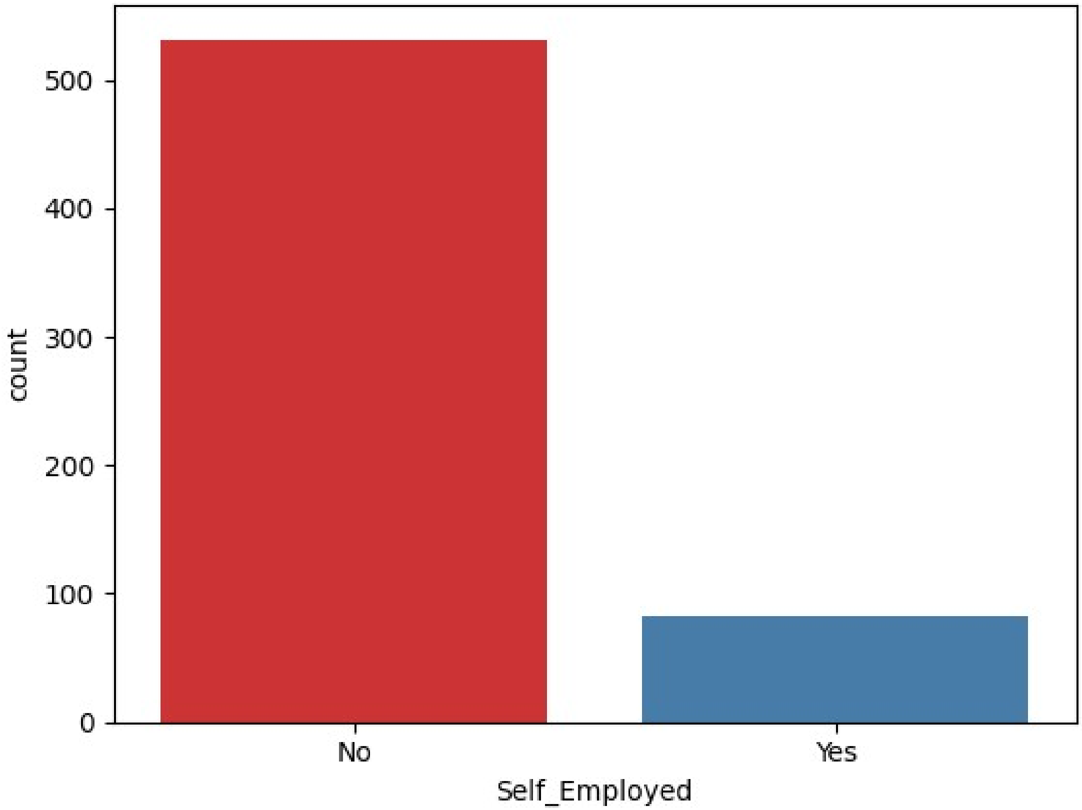

```
# Percentage of people of take loan by Dependents

print("number of people who take loan as group by dependents:")
print(df['Dependents'].value_counts())

sns.countplot(x='Dependents', data = df, 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

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



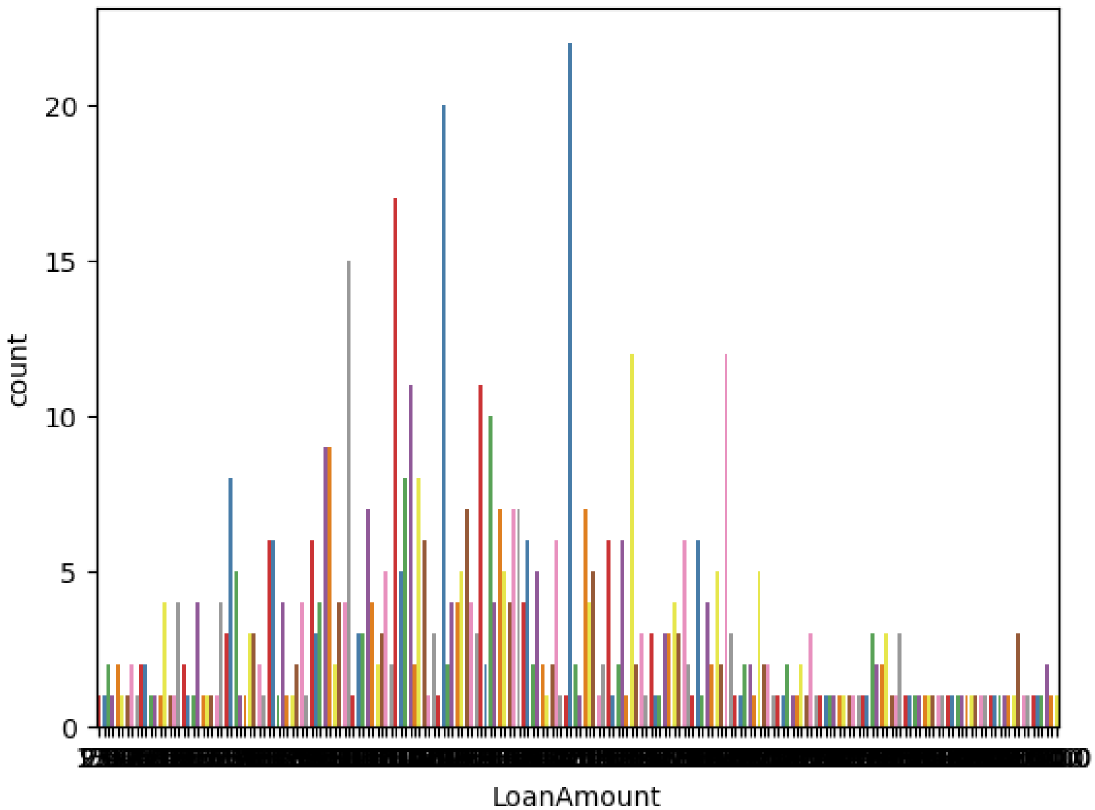

```
# Percentage of people of take loan by Self_Employed

print("number of people who take loan as group by Self Employed:")
print(df['Self_Employed'].value_counts())

sns.countplot(x='Self_Employed', data = df, palette = 'Set1')
```

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

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



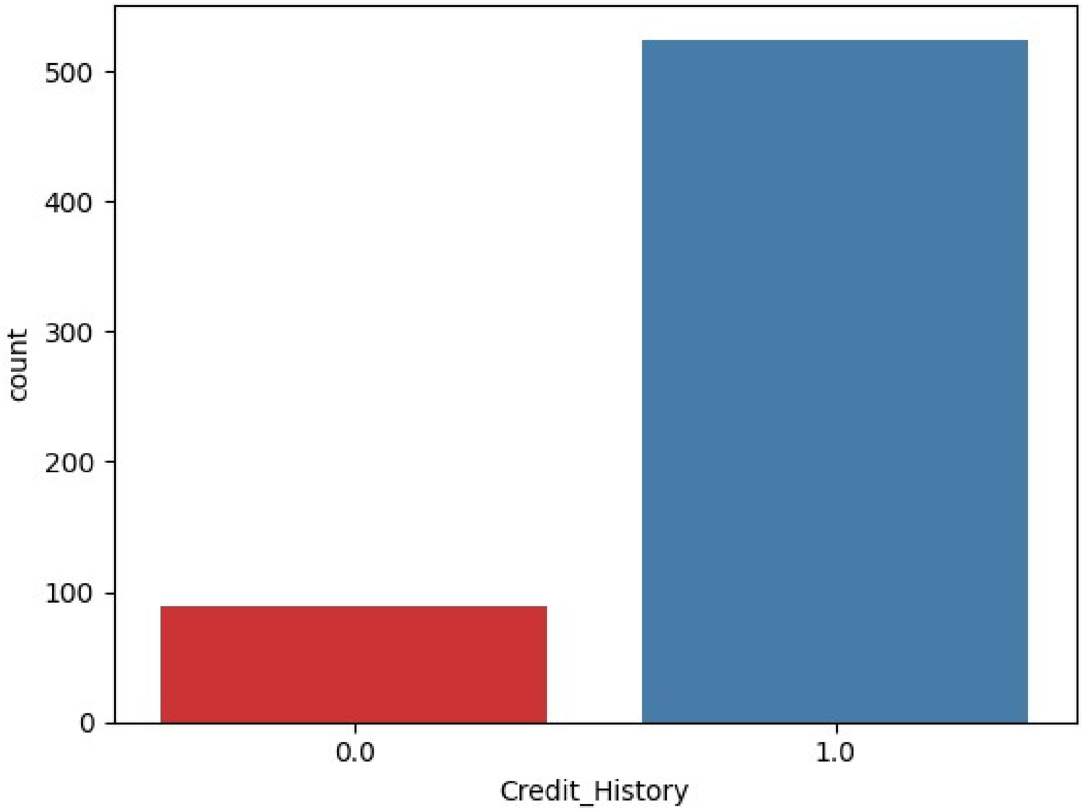

```
# Percentage of people of take loan by Loan_Amount

print("number of people who take loan as group by Loan Amount:")
print(df['LoanAmount'].value_counts())

sns.countplot(x='LoanAmount', data = df, palette = 'Set1')
```

```
number of people who take loan as group by Loan Amount:
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

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




```
# Percentage of people of take loan by Credit History

print("number of people who take loan as group by Credit History:")
print(df['Credit_History'].value_counts())

sns.countplot(x='Credit_History', data = df, 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

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



Selecting Rows and Columns for Input and Output (Dependent/Independent Variables)

```
df.head()

    Loan_ID Gender Married Dependents     Education 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

   ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
0             5849                0.0  146.412162             360.0
1             4583             1508.0  128.000000             360.0
2             3000                0.0   66.000000             360.0
3             2583             2358.0  120.000000             360.0
4             6000                0.0  141.000000             360.0

   Credit_History Property_Area Loan_Status  loanAmount_log
TotalIncome  \
0             1.0         Urban           Y      146.412162
5849.0
1             1.0         Rural           N      128.000000
6091.0
```

```
2              1.0         Urban           Y       66.000000
3000.0
3              1.0         Urban           Y      120.000000
4941.0
4              1.0         Urban           Y      141.000000
6000.0

   TotalIncome_log
0         8.674026
1         8.714568
2         8.006368
3         8.505323
4         8.699515

# Gender, Married, Dependents, Education, Loan_Amount_Term,
Credit_History, LoanAmount_log, TotalIncome
X = df.iloc[:,np.r_[1:5, 9:11, 13:15]].values

# Only Loan Status column
y = df.iloc[:,12].values

X

array([['Male', 'No', '0', ..., 1.0, 146.41216216216216, 5849.0],
       ['Male', 'Yes', '1', ..., 1.0, 128.0, 6091.0],
       ['Male', 'Yes', '0', ..., 1.0, 66.0, 3000.0],
       ...,
       ['Male', 'Yes', '1', ..., 1.0, 253.0, 8312.0],
       ['Male', 'Yes', '2', ..., 1.0, 187.0, 7583.0],
       ['Female', 'No', '0', ..., 0.0, 133.0, 4583.0]], dtype=object)

y

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)
```

Spliting dataset for training and testing purpose

```
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)
```

Encoding categorical columns

```
from sklearn.preprocessing import LabelEncoder

Labelencoder_x = LabelEncoder()

for i in range(0, 5):
    X_train[:,i] = Labelencoder_x.fit_transform(X_train[:,i])
    X_train[:,7] = Labelencoder_x.fit_transform(X_train[:,7])

X_train

array([[1, 1, 0, ..., 1.0, 131.0, 267],
       [1, 0, 1, ..., 1.0, 196.0, 407],
```

```
        [1, 1, 0, ..., 0.0, 149.0, 249],
        ...,
        [1, 1, 3, ..., 1.0, 200.0, 363],
        [1, 1, 0, ..., 1.0, 160.0, 273],
        [0, 1, 0, ..., 1.0, 182.0, 301]], dtype=object)

Labelencoder_y = LabelEncoder()

y_train = Labelencoder_y.fit_transform(y_train)
y_train

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])
```

```
# Now, encoding testing data

for i in range(0,5):
    X_test[:,i] = Labelencoder_x.fit_transform(X_test[:,i])


X_test[:,7] = Labelencoder_x.fit_transform(X_test[:,7])
X_test
```

```
array([[1, 0, 0, 0, 5, 1.0, 84.0, 85],
       [0, 0, 0, 0, 5, 1.0, 112.0, 28],
       [1, 1, 0, 0, 5, 1.0, 324.0, 104],
       [1, 1, 0, 0, 5, 1.0, 110.0, 80],
       [1, 1, 2, 0, 5, 1.0, 97.0, 22],
       [1, 1, 0, 1, 3, 0.0, 165.0, 70],
       [1, 1, 3, 0, 3, 1.0, 157.0, 77],
       [1, 0, 0, 0, 5, 1.0, 405.0, 114],
       [1, 0, 0, 0, 5, 0.0, 124.0, 53],
       [1, 1, 0, 0, 5, 1.0, 128.0, 55],
       [0, 0, 0, 0, 5, 1.0, 84.0, 4],
       [1, 1, 1, 0, 5, 1.0, 95.0, 2],
       [0, 0, 0, 0, 5, 1.0, 280.0, 96],
       [1, 1, 2, 0, 5, 1.0, 236.0, 97],
       [1, 1, 0, 0, 5, 1.0, 96.0, 117],
       [1, 1, 1, 0, 5, 1.0, 67.0, 22],
       [1, 0, 1, 1, 5, 1.0, 190.0, 32],
       [1, 0, 0, 1, 5, 1.0, 132.0, 25],
       [0, 0, 0, 0, 5, 1.0, 93.0, 1],
       [1, 1, 0, 1, 5, 0.0, 181.0, 44],
       [0, 1, 0, 0, 5, 0.0, 120.0, 71],
       [1, 1, 0, 0, 5, 1.0, 143.0, 43],
       [1, 1, 2, 0, 5, 1.0, 108.0, 91],
       [1, 1, 2, 0, 5, 1.0, 165.0, 111],
       [1, 1, 0, 0, 5, 1.0, 58.0, 35],
       [1, 1, 1, 0, 5, 1.0, 250.0, 94],
       [1, 0, 0, 0, 5, 1.0, 187.0, 98],
       [1, 1, 0, 0, 5, 1.0, 187.0, 110],
       [1, 1, 3, 0, 5, 0.0, 128.0, 41],
       [0, 0, 0, 0, 5, 0.0, 103.0, 50],
       [1, 1, 0, 0, 5, 1.0, 228.0, 99],
       [1, 0, 0, 1, 5, 1.0, 48.0, 46],
       [1, 1, 1, 1, 5, 1.0, 90.0, 52],
       [1, 1, 0, 0, 5, 1.0, 180.0, 102],
```

```
        [1, 1, 0, 0, 5, 1.0, 146.41216216216216, 95],
        [0, 1, 0, 1, 5, 0.0, 178.0, 57],
        [1, 1, 0, 0, 5, 1.0, 172.0, 65],
        [1, 0, 0, 1, 5, 1.0, 126.0, 39],
        [1, 1, 0, 0, 5, 1.0, 128.0, 75],
        [1, 1, 2, 1, 5, 1.0, 108.0, 24],
        [0, 0, 0, 0, 5, 1.0, 80.0, 9],
        [1, 1, 3, 0, 5, 0.0, 123.0, 68],
        [1, 1, 2, 0, 2, 1.0, 17.0, 0],
        [1, 1, 1, 1, 5, 1.0, 158.0, 67],
        [1, 0, 0, 0, 5, 1.0, 76.0, 21],
        [1, 0, 0, 0, 5, 1.0, 187.0, 113],
        [1, 1, 1, 0, 5, 1.0, 116.0, 18],
        [0, 0, 0, 0, 5, 1.0, 115.0, 37],
        [1, 1, 1, 0, 5, 1.0, 128.0, 72],
        [1, 0, 0, 0, 5, 1.0, 140.0, 78],
        [1, 1, 3, 1, 5, 1.0, 74.0, 8],
        [1, 1, 0, 0, 5, 1.0, 130.0, 84],
        [1, 1, 0, 1, 5, 1.0, 107.0, 31],
        [1, 0, 0, 0, 5, 1.0, 146.41216216216216, 61],
        [1, 1, 0, 0, 5, 1.0, 112.0, 19],
        [1, 1, 0, 0, 5, 1.0, 259.0, 107],
        [1, 1, 0, 0, 5, 1.0, 95.0, 34],
        [1, 0, 0, 1, 5, 1.0, 133.0, 74],
        [1, 1, 2, 0, 5, 1.0, 168.0, 62],
        [1, 0, 0, 0, 5, 1.0, 120.0, 27],
        [0, 0, 0, 0, 5, 0.0, 137.0, 108],
        [0, 0, 0, 0, 5, 1.0, 214.0, 103],
        [1, 1, 0, 1, 5, 1.0, 115.0, 38],
        [0, 0, 0, 0, 5, 0.0, 76.0, 13],
        [1, 1, 2, 0, 5, 1.0, 133.0, 69],
        [1, 1, 1, 0, 5, 1.0, 315.0, 112],
        [1, 1, 0, 0, 5, 1.0, 160.0, 73],
        [1, 0, 0, 0, 5, 1.0, 136.0, 47],
        [1, 1, 0, 0, 5, 1.0, 182.0, 81],
        [1, 0, 0, 1, 5, 1.0, 96.0, 60],
        [1, 0, 0, 0, 5, 1.0, 67.0, 83],
        [0, 1, 0, 0, 5, 1.0, 130.0, 5],
        [1, 1, 2, 1, 5, 1.0, 157.0, 58],
        [1, 1, 1, 1, 3, 1.0, 137.0, 79],
        [0, 1, 0, 0, 5, 1.0, 144.0, 54],
        [1, 1, 0, 1, 4, 1.0, 124.0, 56],
        [1, 0, 0, 0, 5, 1.0, 90.0, 120],
        [1, 0, 3, 0, 5, 1.0, 320.0, 118],
        [1, 1, 2, 0, 5, 1.0, 112.0, 101],
        [0, 0, 0, 0, 5, 0.0, 116.0, 26],
        [0, 0, 0, 0, 6, 1.0, 113.0, 33],
        [1, 1, 1, 0, 5, 1.0, 500.0, 119],
        [0, 0, 0, 0, 5, 1.0, 194.0, 89],
```

```
       [1, 1, 2, 0, 5, 1.0, 187.0, 92],
       [1, 0, 0, 0, 6, 1.0, 71.0, 6],
       [1, 1, 0, 0, 0, 1.0, 111.0, 90],
       [1, 1, 0, 0, 5, 1.0, 110.0, 45],
       [1, 1, 2, 0, 5, 1.0, 200.0, 109],
       [1, 0, 1, 0, 3, 1.0, 113.0, 17],
       [1, 1, 1, 0, 5, 1.0, 104.0, 36],
       [0, 1, 0, 1, 5, 1.0, 100.0, 16],
       [1, 0, 0, 0, 5, 1.0, 74.0, 7],
       [1, 1, 1, 0, 1, 1.0, 172.0, 88],
       [1, 1, 3, 0, 4, 0.0, 180.0, 87],
       [0, 0, 0, 0, 5, 1.0, 71.0, 3],
       [1, 0, 0, 1, 3, 0.0, 126.0, 59],
       [1, 0, 0, 0, 3, 1.0, 175.0, 82],
       [1, 0, 0, 0, 5, 1.0, 144.0, 66],
       [1, 1, 2, 1, 5, 1.0, 81.0, 51],
       [1, 1, 1, 0, 5, 1.0, 187.0, 100],
       [1, 1, 0, 0, 5, 1.0, 211.0, 93],
       [1, 1, 0, 0, 5, 1.0, 100.0, 15],
       [1, 1, 2, 0, 5, 1.0, 120.0, 106],
       [1, 0, 0, 0, 3, 1.0, 120.0, 105],
       [1, 1, 3, 0, 5, 1.0, 128.0, 64],
       [1, 0, 0, 0, 5, 1.0, 125.0, 49],
       [1, 0, 0, 1, 5, 1.0, 104.0, 42],
       [0, 0, 0, 0, 5, 1.0, 88.0, 10],
       [1, 1, 0, 1, 5, 1.0, 95.0, 20],
       [1, 1, 3, 1, 3, 1.0, 81.0, 14],
       [1, 0, 0, 0, 5, 1.0, 200.0, 76],
       [0, 0, 0, 0, 5, 1.0, 135.0, 11],
       [1, 0, 0, 0, 6, 1.0, 113.0, 18],
       [1, 1, 2, 0, 5, 1.0, 70.0, 23],
       [1, 1, 0, 1, 5, 0.0, 201.0, 63],
       [1, 1, 0, 0, 3, 0.0, 90.0, 48],
       [0, 0, 0, 0, 5, 1.0, 84.0, 30],
       [1, 0, 0, 0, 5, 1.0, 134.0, 29],
       [1, 1, 2, 0, 5, 1.0, 176.0, 86],
       [1, 1, 3, 0, 5, 1.0, 130.0, 115],
       [1, 1, 0, 0, 5, 1.0, 436.0, 116],
       [1, 1, 3, 1, 3, 0.0, 70.0, 40],
       [1, 1, 1, 0, 5, 1.0, 96.0, 12]], dtype=object)

y_test = Labelencoder_y.fit_transform(y_test)
y_test

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])
```

Scaling the dataset

```
from sklearn.preprocessing import StandardScaler

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

Creating and Training the Model

1. Using Random Forest

```
from sklearn.ensemble import RandomForestClassifier

rf_clf = RandomForestClassifier()
rf_clf.fit(X_train, y_train)

RandomForestClassifier()

# Let's use this model to predict

y_pred = rf_clf.predict(x_test)
y_pred

array([1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0,
1,
       1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0,
1,
       1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 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])

# Checking Accuracy of the prediction

from sklearn import metrics

print("Accuracy of Random Forest Classifier is",
metrics.accuracy_score(y_pred,y_test))

Accuracy of Random Forest Classifier is 0.7560975609756098
```

- **Quite Low accuracy.** Let's try another algorithm for creating the model.

2. Using Naive_Bayes Algorithm

```
from sklearn.naive_bayes import GaussianNB

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

GaussianNB()

y_pred = nb_clf.predict(X_test)

print("Accuracy of Gaussian Naive Bayes is",
metrics.accuracy_score(y_pred,y_test))

Accuracy of Gaussian Naive Bayes is 0.6829268292682927

y_pred

array([0, 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, 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, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1,
1,
       1, 1, 1, 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, 0, 1, 1, 1])
```

3. Using Decision Tree Clasifier

```
from sklearn.tree import DecisionTreeClassifier

dt_clf = DecisionTreeClassifier()
dt_clf.fit(X_train, y_train)

DecisionTreeClassifier()

y_pred = dt_clf.predict(X_test)

print("Accuracy of DecisionTree Classifier is",
metrics.accuracy_score(y_pred,y_test))

Accuracy of DecisionTree Classifier is 0.7235772357723578

y_pred

array([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, 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])
```

4. Using K-Neighbours

```
from sklearn.neighbors import KNeighborsClassifier

kn_clf = KNeighborsClassifier()
kn_clf.fit(X_train, y_train)

KNeighborsClassifier()

y_pred = dt_clf.predict(X_test)

print("Accuracy of KNeighbors Classifier is",
metrics.accuracy_score(y_pred,y_test))

Accuracy of KNeighbors Classifier is 0.7235772357723578
```

Random Forest has the best prediction of all.