Loan Approval System by Chhavi Rajora

```
# Loan Approval System
```

```
# Importing all the necessary libraries
```

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

```
# Uploading the dataset in google colab
```

```
df=pd.read_excel('/content/loan-predictionUC.csv (1) (1) (1).xlsx')
```

```
# Printing the data of first 5 entries
```

```
df.head()
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term
0	LP001002	Male	No	0	Graduate	No	5849	0.0	NaN	360.0
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.0
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	360.0
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	360.0

```
# Getting information of each column in the dataset such as count and datatype
```

```
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
```

```
# Finding the total null values in each column of dataset
```

```
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
```

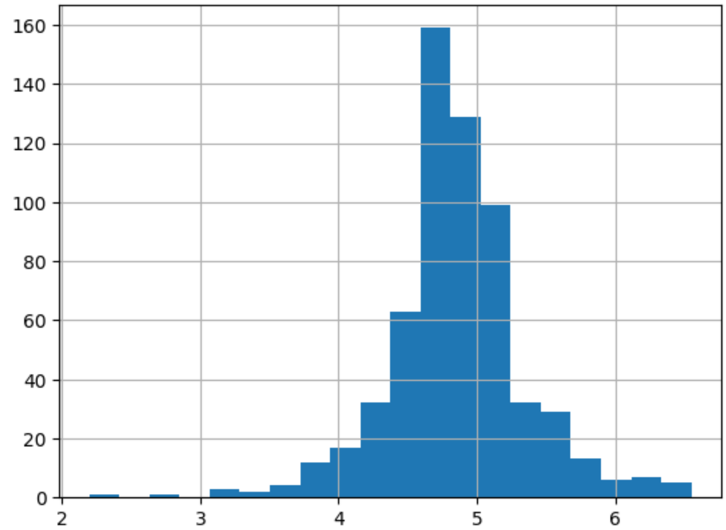
```
# Create new column which will be equal to the logarithm value of column 'LoanAmount'

df['LoanAmount_log']=np.log(df['LoanAmount'])

# Plot histogram of this new column

df['LoanAmount_log'].hist(bins=20)
```

```
<Axes: >
```




```
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
LoamAmount_log       22
TotalIncome           0
TotalIncome_log       0
LoanAmount_log        0
dtype: int64
```

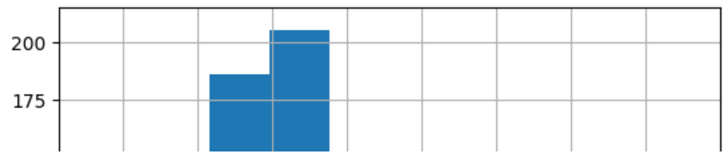
```
# Create new column 'TotalIncome' which is sum of all income column

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

# Plotting the histogram graph of this new column

df['TotalIncome_log'].hist()
```

```
<Axes: >
```




```
# Fill all the missing values of each column with either their mode values or mean values
```


```
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=df.LoanAmount.fillna(df.LoanAmount.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.LoanAmount_log=df.LoanAmount_log.fillna(df.LoanAmount_log.mean())
```

```
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
LoamAmount_log      22
TotalIncome          0
TotalIncome_log      0
LoanAmount_log       0
dtype: int64
```

```
del df['LoamAmount_log']
```

```
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
TotalIncome         0
TotalIncome_log     0
LoanAmount_log      0
dtype: int64
```

```
# Select the attributes which we want in x and in y arrays
```

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

```
x
```

```
array([['Male', 'No', 0, ..., 1.0, 5849.0, 8.674025985443025],
       ['Male', 'Yes', 1, ..., 1.0, 6091.0, 8.714567550836485],
       ['Male', 'Yes', 0, ..., 1.0, 3000.0, 8.006367567650246],
       ...,
       ['Male', 'Yes', 1, ..., 1.0, 8312.0, 9.025455532779063],
       ['Male', 'Yes', 2, ..., 1.0, 7583.0, 8.933664178700935],
       ['Female', 'No', 0, ..., 0.0, 4583.0, 8.430109084509125]],
      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)
```

```
# Estimating missing value percentage of each attributes now
```

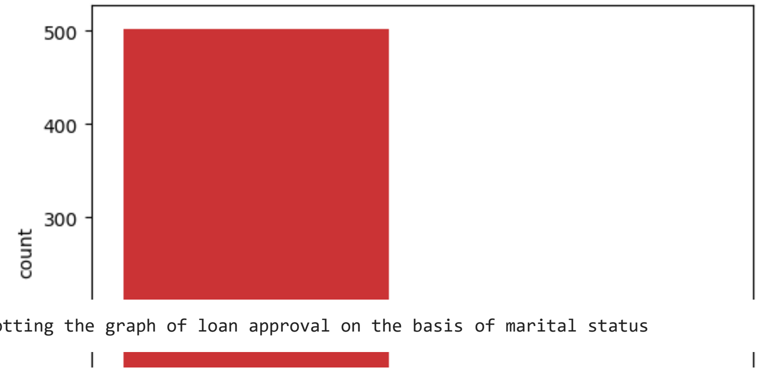
```
print("Percentage of missing Gender = %2f%%" %((df['Gender'].isnull().sum()/df.shape[0])*100))
print("Percentage of missing Married = %2f%%" %((df['Married'].isnull().sum()/df.shape[0])*100))
print("Percentage of missing Dependents = %2f%%" %((df['Dependents'].isnull().sum()/df.shape[0])*100))
print("Percentage of missing Self_Employed = %2f%%" %((df['Self_Employed'].isnull().sum()/df.shape[0])*100))
print("Percentage of missing LoanAmount = %2f%%" %((df['LoanAmount'].isnull().sum()/df.shape[0])*100))
print("Percentage of missing Loan_Amount_Term = %2f%%" %((df['Loan_Amount_Term'].isnull().sum()/df.shape[0])*100))
print("Percentage of missing Credit_History = %2f%%" %((df['Credit_History'].isnull().sum()/df.shape[0])*100))
```

```
Percentage of missing Gender = 0.000000%
Percentage of missing Married = 0.000000%
Percentage of missing Dependents = 0.000000%
Percentage of missing Self_Employed = 0.000000%
Percentage of missing LoanAmount = 0.000000%
Percentage of missing Loan_Amount_Term = 0.000000%
Percentage of missing Credit_History = 0.000000%
```

```
# Plotting the graph of loan approval on the basis of gender
```

```
print("Number of people who takes loans on the basis of Gender:")
print(df['Gender'].value_counts())
sns.countplot(x='Gender',data=df, palette= 'Set1')
```

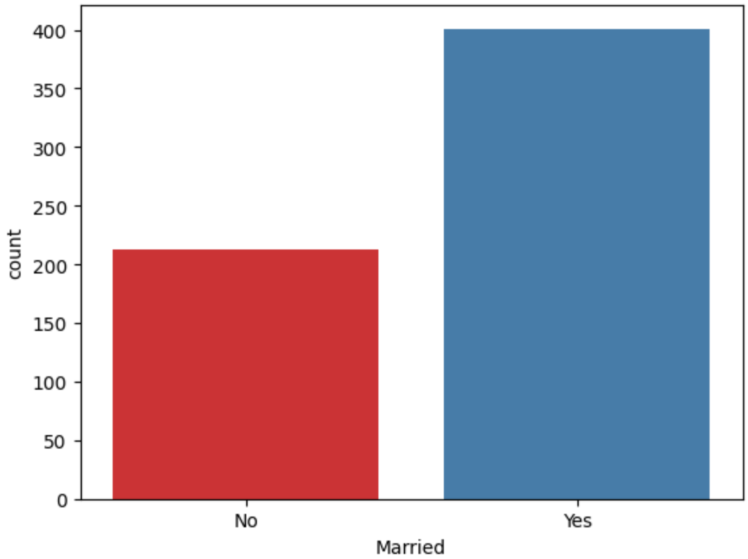
```
Number of people who takes loans on the basis of Gender:
Male      502
Female    112
Name: Gender, dtype: int64
<Axes: xlabel='Gender', ylabel='count'>
```




```
# Plotting the graph of loan approval on the basis of marital status
```

```
print("Number of people who takes loans on the basis of Marital status:")
print(df['Married'].value_counts())
sns.countplot(x='Married',data=df, palette= 'Set1')
```

```
Number of people who takes loans on the basis of Marital status:
Yes    401
No     213
Name: Married, dtype: int64
<Axes: xlabel='Married', ylabel='count'>
```




```
# Plotting the graph of loan approval on the basis of their dependency status
```

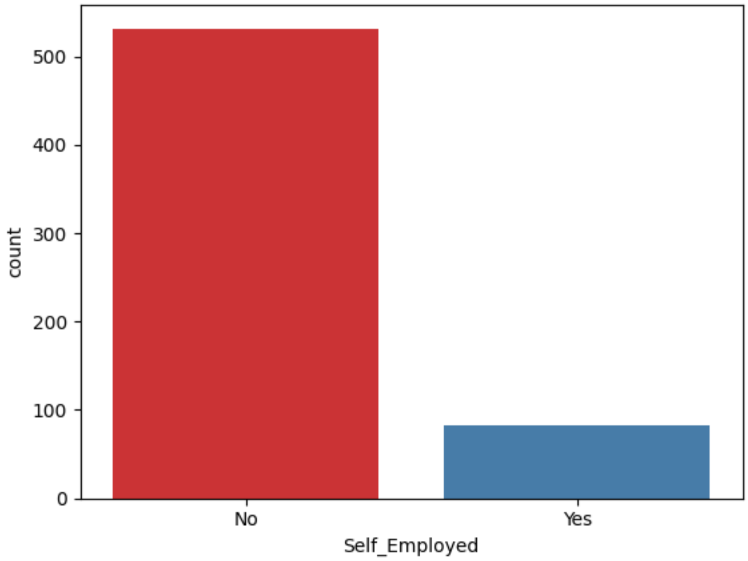
```
print("Number of people who takes loans on the basis of dependency status:")
print(df['Dependents'].value_counts())
sns.countplot(x='Dependents',data=df, palette= 'Set1')
```

```
Number of people who takes loans on the basis of dependency status:
0     360
1     102
2     101
3+     51
Name: Dependents, dtype: int64
<Axes: xlabel='Dependents', ylabel='count'>
```

```
# Plotting the graph of loan approval on the basis of Self Employment
```

```
print("Number of people who takes loans on the basis of Self Employment:")
print(df['Self_Employed'].value_counts())
sns.countplot(x='Self_Employed',data=df, palette= 'Set1')
```

```
Number of people who takes loans on the basis of Self Employment:
No     532
Yes     82
Name: Self_Employed, dtype: int64
<Axes: xlabel='Self_Employed', ylabel='count'>
```




```
# Plotting the graph of loan approval on the basis of Loan Amount
```

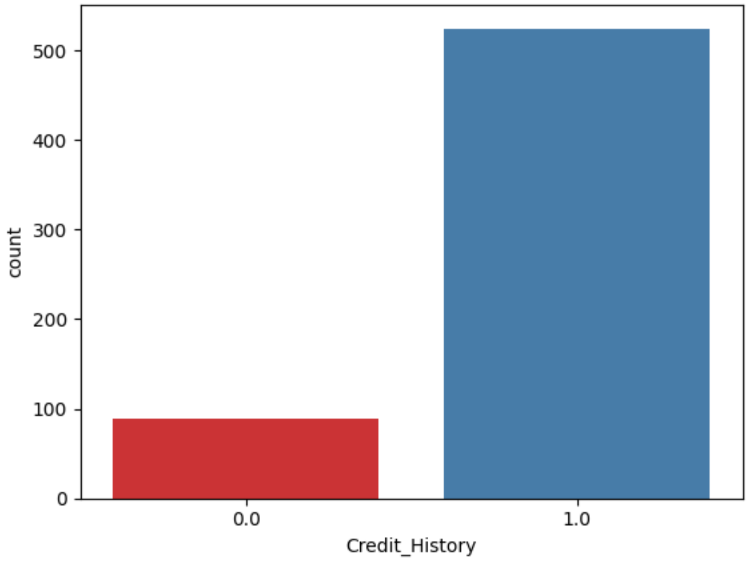
```
print("Number of people who takes loans on the basis of Loan Amount:")
print(df['LoanAmount'].value_counts())
sns.countplot(x='LoanAmount',data=df, palette= 'Set1')
```

```
Number of people who takes loans on the basis of Loan Amount:
120.0    42
110.0    17
100.0    15
```

```
# Plotting the graph of loan approval on the basis of Credit History
            ..

print("Number of people who takes loans on the basis of Credit History:")
print(df['Credit_History'].value_counts())
sns.countplot(x='Credit_History',data=df, palette= 'Set1')
```

```
Number of people who takes loans on the basis of Credit History:
1.0    525
0.0     89
Name: Credit_History, dtype: int64
<Axes: xlabel='Credit_History', ylabel='count'>
```




```
# Splitting of dataset into training data and testing data
# Importing libraries for encoding of data and standard scaling of data


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,StandardScaler
label_encoder=LabelEncoder


# Encoding the categorial data into numerical value
```

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler

label_encoder = LabelEncoder()

# Encode categorical columns
df['Education'] = label_encoder.fit_transform(df['Education'])
df['Loan_id'] = label_encoder.fit_transform(df['Education'])
df['Gender'] = label_encoder.fit_transform(df['Gender'])
df['Married'] = label_encoder.fit_transform(df['Married'])
df['Self_Employed'] = label_encoder.fit_transform(df['Self_Employed'])
df['Property_Area'] = label_encoder.fit_transform(df['Property_Area'])
df['Loan_Status'] = label_encoder.fit_transform(df['Loan_Status'])


sent = '3+'

for col in df.columns:
    if sent in df[col].values:
        df[col] = df[col].replace(sent, 3)
        df[col] = df[col].astype(float)




# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(df.drop('Loan_Status', axis=1), df['Loan_Status'], test_size=0.2, random_state=0)

# Scale numeric features
numeric_features = ['Dependents','ApplicantIncome','CoapplicantIncome','LoanAmount','Loan_Amount_Term','Credit_History']  # Replace wit
scaler = StandardScaler()




# Fit and transform on the training data
X_train[numeric_features] = scaler.fit_transform(X_train[numeric_features])

# Transform the test data
X_test[numeric_features] = scaler.transform(X_test[numeric_features])
```

```
<ipython-input-96-67225b0d25ac>:19: FutureWarning: elementwise comparison failed; returning scalar instead, but in the future will p
  if sent in df[col].values:
```

```
X_train
```

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_
90	1	1	-0.763047	0	0	-0.400019	0.493485	-0.175377	0.269838	
533	1	0	0.225491	0	0	0.846479	-0.547973	0.574008	0.269838	
452	1	1	-0.763047	0	0	-0.251196	0.074388	0.032145	0.269838	-
355	0	0	-0.763047	0	0	-0.271490	-0.547973	-0.348312	-2.577377	
266	1	1	1.214029	0	0	-0.136949	-0.049869	0.043674	0.269838	
...	...	...	...	...	...	...	...	...	...	
277	1	1	-0.763047	0	0	-0.378221	-0.081113	-0.763356	0.269838	
9	1	1	0.225491	0	0	1.085647	3.390891	2.337944	0.269838	
359	1	1	2.202567	0	0	-0.067950	0.589370	0.620124	0.269838	
192	1	1	-0.763047	1	0	0.062232	-0.547973	0.158964	0.269838	
559	0	1	-0.763047	0	0	-0.216321	0.280165	0.412602	0.269838	

491 rows × 15 columns

```
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])
```

```
X_test
```

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapplic
454	1	0	-0.763047	0	1	0.220374	
52	0	0	-0.763047	0	0	-0.208805	
536	1	1	-0.763047	0	0	0.077264	
469	1	1	-0.763047	0	0	-0.193321	
55	1	1	1.214029	0	0	-0.437600	
...	...	...	...	...	...	...	
337	1	1	1.214029	0	1	-0.468868	
376	1	1	2.202567	0	0	0.470666	
278	1	1	-0.763047	0	0	1.347514	
466	1	1	2.202567	1	0	-0.401672	
303	1	1	0.225491	0	0	-0.600402	

123 rows × 15 columns

```
labelencoder_y=LabelEncoder()
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])
```

```
# Training our data with models such as RandomForestClassifier, Naive bayes, Logistic Regression,
# Decision Tree Classifier, KNeighbor Classifier
```

```
from sklearn.ensemble import RandomForestClassifier
randomfor_clf=RandomForestClassifier()
randomfor_clf.fit(X_train,y_train)
```

```
▾ RandomForestClassifier
RandomForestClassifier()
```

```
from sklearn.naive_bayes import GaussianNB
naive_bayes=GaussianNB()
naive_bayes.fit(X_train,y_train)
```

```
▾ GaussianNB
GaussianNB()
```

```
from sklearn.linear_model import LogisticRegression
Logistic_regression=LogisticRegression(random_state=42)
Logistic_regression.fit(X_train,y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: Conver
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
from sklearn.tree import DecisionTreeClassifier
decision_tree=DecisionTreeClassifier(random_state=42)
decision_tree.fit(X_train,y_train)
```

```
▾        DecisionTreeClassifier
DecisionTreeClassifier(random_state=42)
```

```
from sklearn.svm import SVC
svm=SVC(random_state=42)
svm.fit(X_train,y_train)
```

```
▾        SVC
SVC(random_state=42)
```

```
from sklearn.neighbors import KNeighborsClassifier
kn_clf=KNeighborsClassifier()
kn_clf.fit(X_train,y_train)
```

```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

```
# Importing libraries to calculate accuracy, geneate confusion matrix and classification report
```

```
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
```

```
# Create evaluation function of model that gives accuracy score of model
```

```
def evaluation_model(model,x_test,y_test):
    y_pred=model.predict(X_test)
    print("Confusion Matrix :\n",confusion_matrix(y_test,y_pred))
    print("Classification Report :\n",classification_report(y_test,y_pred))
    accuracy=accuracy_score(y_test,y_pred)
    print(f"Accuracy score: {accuracy:.2f}")
```

```
# Estimate accuracy score of each model
```

```
print("Random Forest Classifier")
evaluation_model(randomfor_clf,X_test,y_test)

print("\nNaive bayes Classifier")
evaluation_model(naive_bayes ,X_test,y_test)

print("\nLogistic Regression")
evaluation_model(Logistic_regression,X_test,y_test)

print("\nDecision Tree Classifier")
evaluation_model(decision_tree,X_test,y_test)

print("\nSVC")
evaluation_model(svm, X_test, y_test)

print("\n KNeighbor Classifier")
evaluation_model(kn_clf,X_test,y_test)
```

```
    accuracy                           0.76       123
   macro avg       0.70      0.74      0.71       123
weighted avg       0.78      0.76      0.76       123

Accuracy score: 0.76

SVC
Confusion Matrix :
 [[ 0 33]
 [ 0 90]]
Classification Report :
               precision    recall  f1-score   support

           0       0.00      0.00      0.00        33
           1       0.73      1.00      0.85        90

    accuracy                           0.73       123
   macro avg       0.37      0.50      0.42       123
weighted avg       0.54      0.73      0.62       123

Accuracy score: 0.73

 KNeighbor Classifier
Confusion Matrix :
 [[ 5 28]
 [17 73]]
Classification Report :
               precision    recall  f1-score   support

           0       0.23      0.15      0.18        33
           1       0.72      0.81      0.76        90

    accuracy                           0.63       123
   macro avg       0.48      0.48      0.47       123
weighted avg       0.59      0.63      0.61       123

Accuracy score: 0.63
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score a
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score a
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score a
  _warn_prf(average, modifier, msg_start, len(result))
```

Conclusion:

Performance Evaluation- The accuracy score of all the models used in this project are as follows:

- Random Forest Classifier = 0.82
- Naive Bayes Classifier = 0.83
- Logistic Regression = 0.84
- Decision Tree Classifier = 0.76
- SVC = 0.73
- KNeighbor Classifier = 0.63

Out of all these models, Logistic Regression has the maximum accuracy of 0.84.

Challenges Faced-

- Dealing with the missing values in the dataset.
- Encoding of categorial data into the numerical value.
- Calculating accuracy of each model.

Recommendation-

- Implementation of robust system for continously monitering the machine learning model 's performance.
- Improve the user interface of the loan approval system to enhance user experience for both applicants and internal stakeholders. A user-friendly interface can facilitate smoother interactions, reduce processing times, and increase overall satisfaction.
- Explore the incorporation of advanced analytics techniques, such as predictive analytics and alternative data sources, to further refine the loan approval system. This can enhance predictive capabilities.
- Design the loan approval system with scalability in mind to handle increasing volumes of loan applications. Ensure flexibility in the system architecture to accommodate changes in business processes and accommodate future growth.

Future Scope -

- Research and implement advanced risk modeling techniques, including machine learning algorithms capable of dynamic risk assessment. This can improve the accuracy of predicting default risks and enhance the overall risk management strategy.

- Investigate the potential use of blockchain technology to enhance security, transparency, and efficiency in the loan approval process.