Dataset Information

Dream Housing Finance company deals in all home loans. They have presence across all urban, semi urban and rural areas.

Customer first apply for home loan after that company validates the customer eligibility for loan.

Company wants to automate the loan eligibility process (real time) based on customer detail provided while filling online application form.

These details are Gender, Marital Status, Education, Number of Dependents, Income, Loan Amount, Credit History and others.

To automate this process, they have given a problem to identify the customers segments, those are eligible for loan amount so that they can specifically target these customers.

This is a standard supervised classification task.A classification problem where we have to predict whether a loan would be approved or not.

Below is the dataset attributes with description.

Variable	Description
Loan_ID	Unique Loan ID
Gender	Male/ Female
Married	Applicant married (Y/N)
Dependents	Number of dependents
Education	Applicant Education (Graduate/ Under Graduate)
Self_Employed	Self employed (Y/N)
ApplicantIncome	Applicant income
CoapplicantIncome	Coapplicant income
LoanAmount	Loan amount in thousands
Loan_Amount_Term	Term of loan in months
Credit_History	credit history meets guidelines
Property_Area	Urban/ Semi Urban/ Rural
Loan_Status	Loan approved (Y/N)

First, need to import all necessory library

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

```
df = pd.read_csv('Loan Prediction Dataset.csv')
df.head()
```

- We have to predict the output variable "Loan status".
- The Input attributes are in categorical as well as in numerical form.
- We have to analyze all the attributes.

```
df.describe()
```

- The total count column displays some missing values, which we will deal with later.
- The credit history attributes are in the range of 0 to 1.

```
df.info()
```

```
We can observe 13 attributes. Out of which 4 attributes are in float, 1 attribute
is in integer and the other 8 are in objects.

We can change the object into corresponding data to reduce the usage memory.

However, we have 62 KB of memory usage, therefore we don't have to change any of
the data types.
```

Preprocessing the Loan Sanction Data

Let us check for NULL values in the dataset.

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

- We have found 6 columns having NULL values.
- Now, we have to replace the NULL values with some common values.

Let us fill in the missing values for numerical terms using mean.

```
df['LoanAmount']=df['LoanAmount'].fillna(df['LoanAmount'].mean())

df['Loan_Amount_Term']=df['Loan_Amount_Term'].fillna(df['Loan_Amount_Term'].mean(
))

df['Credit_History']=df['Credit_History'].fillna(df['Credit_History'].mean())
```

Let us now fill in the missing values for categorical terms using mode operation.

```
df['Gender'] = df["Gender"].fillna(df['Gender'].mode()[0])

df['Married'] = df["Married"].fillna(df['Married'].mode()[0])

df['Dependents'] = df["Dependents"].fillna(df['Dependents'].mode()[0])

df['Self_Employed'] = df["Self_Employed"].fillna(df['Self_Employed'].mode()[0])
```

- All the missing values will be filled with the most frequently occurring values.
- Modes give the result in their terms of the data frame, so we only need the values. We will specify 0th index to display the values.

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

Exploratory Data Analysis

explore the categorical column "Gender".

```
df['Gender'].value_counts().plot(kind='bar')

df['Married'].value_counts().plot(kind='bar')


df['Dependents'].value_counts().plot(kind='bar')


df['Education'].value_counts().plot(kind='bar')


df['Self_Employed'].value_counts().plot(kind='bar')


df['Property_Area'].value_counts().plot(kind='bar')


df['Loan_Status'].value_counts().plot(kind='bar')
```

now visuallize first applicant income

```
sns.distplot(df["ApplicantIncome"])
```

- The data are skewed left in the graph, which is not a suitable distribution to train a Model.
- Hence, we will apply the Log Transformation to normalize the attributes in the form of Bell Curve

To display the column "Co-applicant Income".

```
sns.distplot(df["CoapplicantIncome"])
```

```
sns.distplot(df["LoanAmount"])
```

- We have to normalize this graph as well.

```
sns.distplot(df['Loan_Amount_Term'])
```

```
sns.distplot(df['Credit_History'])
```

- Since the values of credit history are in the range of 0 to 1, we don't need to normalize this graph.

We can create a new attribute performing Log Transformation. We can also create a new attribute **Total Income**, that is the sum of Applicant Income and Co-applicant Income.

```
df['Total_Income'] = df['ApplicantIncome'] + df['CoapplicantIncome']
```

```
df.head()
```

Log Transformation

Log transformation helps to make the highly skewed distribution to less skewed.

Instead of changing the column, we will add the data into a new column by writing 'Log' after each column.

```
df['ApplicantIncomeLog'] = np.log(df['ApplicantIncome']+1)

sns.distplot(df["ApplicantIncomeLog"])
```

```
df['CoapplicantIncomeLog'] = np.log(df['CoapplicantIncome']+1)

sns.distplot(df["CoapplicantIncomeLog"])

df['LoanAmountLog'] = np.log(df['LoanAmount']+1)


sns.distplot(df["LoanAmountLog"])


df['Loan_Amount_Term_Log'] = np.log(df['Loan_Amount_Term']+1)


sns.distplot(df["Loan_Amount_Term_Log"])


df['Total_Income_Log'] = np.log(df['Total_Income']+1)


sns.distplot(df["Total_Income_Log"])


df.head()
```

Let us drop some unnecessary columns.

```
cols = ['ApplicantIncome', 'CoapplicantIncome', "LoanAmount", "Loan_Amount_Term",
"Total_Income", 'Loan_ID', 'CoapplicantIncomeLog']
df = df.drop(columns=cols, axis=1)
```

Label Encoding

We will use label encoding to convert the categorical column into the numerical column.

```
cols = ['Gender', "Married", "Education", 'Self_Employed', "Property_Area",
"Loan_Status", "Dependents"]

le = LabelEncoder()

for i in cols:
    df[i] = le.fit_transform(df[i])
```

```
df.head()
```

- All the values of the dataset are now in numerical format. It will help us to train our model easily.
- For Loan status 1 indicates 'Yes' and 0 indicates 'No'.

```
df.info()
```

Splitting the data for Training and Testing

```
X = df.drop(columns=['Loan_Status'], axis=1)
y = df['Loan_Status']
```

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import cross_val_score
```

```
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
random_state=42)
```

```
model = LogisticRegression()


model.fit(x_train, y_train)
y_pred = model.predict(x_test)


print("Accuracy is", model.score(x_test, y_test)*100)


cm = confusion_matrix(y_test, y_pred)
cm


sns.heatmap(cm, annot=True)
```