```
import pandas as pd

data = pd.read_csv('train_u6lujuX_CVtuZ9i.csv')
```

1. Display Top 5 Rows of The Dataset

```
data.head()
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coa
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	

2. Check Last 5 Rows of The Dataset

```
data.tail()
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	C
609	LP002978	Female	No	0	Graduate	No	2900	
610	LP002979	Male	Yes	3+	Graduate	No	4106	
611	LP002983	Male	Yes	1	Graduate	No	8072	
612	LP002984	Male	Yes	2	Graduate	No	7583	
613	LP002990	Female	No	0	Graduate	Yes	4583	

3. Find Shape of Our Dataset (Number of Rows And Number of Columns)

```
data.shape
```

```
(614, 13)
```

```
print("Number of Rows",data.shape[0])
print("Number of Columns",data.shape[1])
```

```
Number of Rows 614
Number of Columns 13
```

4. Get Information About Our Dataset Like Total Number Rows, Total Number of Columns, Datatypes of Each Column And Memory Requirement

```
data.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
```

5. Check Null Values In The Dataset

```
data.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
```

```
data.isnull().sum()*100 / len(data)
```

```
Loan_ID              0.000000
Gender               2.117264
Married              0.488599
Dependents           2.442997
Education            0.000000
Self_Employed        5.211726
ApplicantIncome      0.000000
CoapplicantIncome    0.000000
LoanAmount           3.583062
Loan_Amount_Term     2.280130
Credit_History       8.143322
Property_Area        0.000000
Loan_Status          0.000000
dtype: float64
```

6. Handling The missing Values

```
data = data.drop('Loan_ID',axis=1)
data.head(1)
```

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term
0	Male	No	0	Graduate	No	5849	0.0	NaN	360.0

```
columns = ['Gender','Dependents','LoanAmount','Loan_Amount_Term']
data = data.dropna(subset=columns)
data.isnull().sum()*100 / len(data)
```

```
Loan_ID              0.000000
Gender               0.000000
Married              0.000000
Dependents           0.000000
Education            0.000000
Self_Employed        5.424955
ApplicantIncome      0.000000
CoapplicantIncome    0.000000
LoanAmount           0.000000
Loan_Amount_Term     0.000000
```

```
Credit_History       8.679928
Property_Area        0.000000
Loan_Status          0.000000
dtype: float64
```

```
data['Self_Employed'].mode()[0]
```

```
'No'
```

```
data['Self_Employed'] =data['Self_Employed'].fillna(data['Self_Employed'].mode()[0])
data.isnull().sum()*100 / len(data)
```

```
Loan_ID              0.000000
Gender               0.000000
Married              0.000000
Dependents           0.000000
Education            0.000000
Self_Employed        0.000000
ApplicantIncome      0.000000
CoapplicantIncome    0.000000
LoanAmount           0.000000
Loan_Amount_Term     0.000000
Credit_History       8.679928
Property_Area        0.000000
Loan_Status          0.000000
dtype: float64
```

```
data['Gender'].unique()
data['Self_Employed'].unique()
data['Credit_History'].mode()[0]
data['Credit_History'] =data['Credit_History'].fillna(data['Credit_History'].mode()[0])
data.isnull().sum()*100 / len(data)
```

```
Gender               0.0
Married              0.0
Dependents           0.0
Education            0.0
Self_Employed        0.0
ApplicantIncome      0.0
CoapplicantIncome    0.0
LoanAmount           0.0
Loan_Amount_Term     0.0
Credit_History       0.0
Property_Area        0.0
Loan_Status          0.0
dtype: float64
```

7. Handling Categorical Columns

```
data.sample(5)
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term
569	LP002841	Male	Yes	0	Graduate	No	3166	2064.0	104.0	360.0
585	LP002912	Male	Yes	1	Graduate	No	4283	3000.0	172.0	84.0
410	LP002318	Female	No	1	Not Graduate	Yes	3867	0.0	62.0	360.0
193	LP001658	Male	No	0	Graduate	No	3858	0.0	76.0	360.0
380	LP002226	Male	Yes	0	Graduate	No	3333	2500.0	128.0	360.0

```
data['Dependents'] =data['Dependents'].replace(to_replace="3+",value='4')
```

```
data['Dependents'].unique()
```

```
array(['1', '0', '2', '4'], dtype=object)
```

```
data['Loan_Status'].unique()
```

```
array(['N', 'Y'], dtype=object)
```

```
data['Gender'] = data['Gender'].map({'Male':1,'Female':0}).astype('int')
data['Married'] = data['Married'].map({'Yes':1,'No':0}).astype('int')
data['Education'] = data['Education'].map({'Graduate':1,'Not Graduate':0}).astype('int')
data['Self_Employed'] = data['Self_Employed'].map({'Yes':1,'No':0}).astype('int')
data['Property_Area'] = data['Property_Area'].map({'Rural':0,'Semiurban':2,'Urban':1}).astype('int')
data['Loan_Status'] = data['Loan_Status'].map({'Y':1,'N':0}).astype('int')
```

```
data.head()
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Cr
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	
5	LP001011	Male	Yes	2	Graduate	Yes	5417	4196.0	267.0	360.0	

8. Store Feature Matrix In X And Response (Target) In Vector y

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

```
1      0
2      1
3      1
4      1
5      1
      ..
609    1
610    1
611    1
612    1
613    0
Name: Loan_Status, Length: 553, dtype: int64
```

9. Feature Scaling

```
data.head()
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Cr
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	
5	LP001011	Male	Yes	2	Graduate	Yes	5417	4196.0	267.0	360.0	

```
cols = ['ApplicantIncome','CoapplicantIncome','LoanAmount','Loan_Amount_Term']
```

```
from sklearn.preprocessing import StandardScaler
st = StandardScaler()
```

```
X[cols]=st.fit_transform(X[cols])
```

```
X
```

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_His
1	1	1	1	1	0	-0.128694	-0.049699	-0.214368	0.279961	
2	1	1	0	1	1	-0.394296	-0.545638	-0.952675	0.279961	
3	1	1	0	0	0	-0.464262	0.229842	-0.309634	0.279961	
4	1	0	0	1	0	0.109057	-0.545638	-0.059562	0.279961	
5	1	1	2	1	1	0.011239	0.834309	1.440866	0.279961	
...	...	...	...	...	...	...	...	...	...	
609	0	0	0	1	0	-0.411075	-0.545638	-0.893134	0.279961	
610	1	1	4	1	0	-0.208727	-0.545638	-1.262287	-2.468292	
611	1	1	1	1	0	0.456706	-0.466709	1.274152	0.279961	
612	1	1	2	1	0	0.374659	-0.545638	0.488213	0.279961	
613	0	0	0	1	1	-0.128694	-0.545638	-0.154828	0.279961	

553 rows × 11 columns

10. Splitting The Dataset Into The Training Set And Test Set & Applying K-Fold Cross Validation

```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score
import numpy as np


model_df={}
def model_val(model,X,y):
    X_train,X_test,y_train,y_test=train_test_split(X,y,
                                                   test_size=0.20,
                                                   random_state=42)
    model.fit(X_train,y_train)
    y_pred=model.predict(X_test)
    print(f"{model} accuracy is {accuracy_score(y_test,y_pred)}")

    score = cross_val_score(model,X,y,cv=5)
    print(f"{model} Avg cross val score is {np.mean(score)}")
    model_df[model]=round(np.mean(score)*100,2)


model_df
```

```
{}
```

11. Logistic Regression

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model_val(model,X,y)
```

```
LogisticRegression() accuracy is 0.8018018018018018
LogisticRegression() Avg cross val score is 0.8047829647829647
```

✓ 0s completed at 20:01