```
import numpy as np
import pandas as pd
df=pd.read_csv('/content/LoanApprovalPrediction.csv')
df
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amo
0	LP001002	Male	No	0	Graduate	No	5849	0.0	NaN	
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	
...	...	...	...	...	...	...	...	...	...	
609	LP002978	Female	No	0	Graduate	No	2900	0.0	71.0	
610	LP002979	Male	Yes	3+	Graduate	No	4106	0.0	40.0	
611	LP002983	Male	Yes	1	Graduate	No	8072	240.0	253.0	
612	LP002984	Male	Yes	2	Graduate	No	7583	0.0	187.0	
613	LP002990	Female	No	0	Graduate	Yes	4583	0.0	133.0	

614 rows × 13 columns

```
df.head()
```

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

```
df.tail()
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amo
609	LP002978	Female	No	0	Graduate	No	2900	0.0	71.0	
610	LP002979	Male	Yes	3+	Graduate	No	4106	0.0	40.0	
611	LP002983	Male	Yes	1	Graduate	No	8072	240.0	253.0	
612	LP002984	Male	Yes	2	Graduate	No	7583	0.0	187.0	
613	LP002990	Female	No	0	Graduate	Yes	4583	0.0	133.0	

```
df.columns
```

```
Index(['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education',
       'Self_Employed', 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History', 'Property_Area', 'Loan_Status'],
      dtype='object')
```

```
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
```

Find Missing Values

```
df.isna().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
```

Fill missing Values

```
columns=['Gender','Married','Dependents','Self_Employed','LoanAmount','Loan_Amount_Term','Credit_History']
for i in columns:
  x=df[i].mode()[0]
  df[i].fillna(x,inplace=True)
df.isna().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
dtype: int64
```

Drop unwanted column

```
df1=df.drop(['Loan_ID'],axis=1)
df1
```

Convert to numerical data

```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
columns1=['Gender','Married','Dependents','Education','Self_Employed','Property_Area','Loan_Status']
for i in df1[columns1]:
  df1[i]=le.fit_transform(df1[i])
df1
```

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term
0	1	0	0	0	0	5849	0.0	120.0	360.0
1	1	1	1	0	0	4583	1508.0	128.0	360.0
2	1	1	0	0	1	3000	0.0	66.0	360.0
3	1	1	0	1	0	2583	2358.0	120.0	360.0
4	1	0	0	0	0	6000	0.0	141.0	360.0
...	...	...	...	...	...	...	...	...	...
609	0	0	0	0	0	2900	0.0	71.0	360.0
610	1	1	3	0	0	4106	0.0	40.0	180.0
611	1	1	1	0	0	8072	240.0	253.0	360.0
612	1	1	2	0	0	7583	0.0	187.0	360.0

Split into input and output

```
x=df1.iloc[:,:-1].values
y=df1.iloc[:,-1].values
x
y
```

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

Allocate data for training and testing

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.30,random_state=42)
x_train
x_test
```

```
array([[  1.,   1.,   0., ..., 360.,   1.,   1.],
       [  1.,   1.,   0., ..., 360.,   1.,   1.],
       [  1.,   1.,   2., ..., 360.,   1.,   0.],
       ...,
       [  1.,   1.,   2., ..., 180.,   1.,   2.],
       [  1.,   1.,   0., ..., 360.,   1.,   0.],
       [  0.,   1.,   0., ..., 360.,   0.,   0.]])
```

Normalization done by Standard Scaler method

```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
scaler.fit(x_train)
x_train=scaler.fit_transform(x_train)
x_test=scaler.fit_transform(x_test)
x_train
```

```
array([[ 0.49343516,  0.77507823, -0.71703534, ...,  0.30437507,
        -2.47991935,  1.20651188],
       [ 0.49343516,  0.77507823, -0.71703534, ...,  0.30437507,
```

```
        0.40323892, -1.34950589],
       [ 0.49343516, -1.29019234, -0.71703534, ...,  0.30437507,
         0.40323892, -0.071497  ],
       ...,
       [-2.02660871, -1.29019234, -0.71703534, ...,  0.30437507,
         0.40323892,  1.20651188],
       [-2.02660871,  0.77507823, -0.71703534, ..., -1.45542149,
         0.40323892, -0.071497  ],
       [ 0.49343516,  0.77507823, -0.71703534, ...,  0.30437507,
         0.40323892,  1.20651188]])
```

Model Creation using KNN, Naive Bayes, SVM

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
knn=KNeighborsClassifier(n_neighbors=3)
bayes=GaussianNB()
svc=SVC()
lst1=[knn,bayes,svc]
```

Performance Evaluation using FOR LOOP

```
for i in lst1:
  i.fit(x_train,y_train)
  y_pred=i.predict(x_test)
  print('Accuracy Score of',i,':',accuracy_score(y_test,y_pred))
  print('Confusion matrix for',i,':')
  print(confusion_matrix(y_test,y_pred))
  print('classification report of',i,':')
  print(classification_report(y_test,y_pred))
  print('************************************************************')
```

```
Accuracy Score of KNeighborsClassifier(n_neighbors=3) : 0.7567567567567568
Confusion matrix for KNeighborsClassifier(n_neighbors=3) :
[[ 27  38]
 [  7 113]]
classification report of KNeighborsClassifier(n_neighbors=3) :
              precision    recall  f1-score   support

           0       0.79      0.42      0.55        65
           1       0.75      0.94      0.83       120

    accuracy                           0.76       185
   macro avg       0.77      0.68      0.69       185
weighted avg       0.76      0.76      0.73       185

************************************************************
Accuracy Score of GaussianNB() : 0.7675675675675676
Confusion matrix for GaussianNB() :
[[ 28  37]
 [  6 114]]
classification report of GaussianNB() :
              precision    recall  f1-score   support

           0       0.82      0.43      0.57        65
           1       0.75      0.95      0.84       120

    accuracy                           0.77       185
   macro avg       0.79      0.69      0.70       185
weighted avg       0.78      0.77      0.74       185

************************************************************
Accuracy Score of SVC() : 0.7891891891891892
Confusion matrix for SVC() :
[[ 27  38]
 [  1 119]]
classification report of SVC() :
              precision    recall  f1-score   support

           0       0.96      0.42      0.58        65
           1       0.76      0.99      0.86       120

    accuracy                           0.79       185
   macro avg       0.86      0.70      0.72       185
weighted avg       0.83      0.79      0.76       185

************************************************************
```