IMPORTING LIBRARIES

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

```
        Loan_ID  Gender  Married  Dependents  Education  Self_Employed  ApplicantInc
  0    LP001002    Male       No           0   Graduate             No             5
  1    LP001003    Male      Yes           1   Graduate             No             4
  2    LP001005    Male      Yes           0   Graduate            Yes             3
  3    LP001006    Male      Yes           0        Not             No             2
                                                 Graduate
  4    LP001008    Male       No           0   Graduate             No             6
 ...        ...     ...      ...         ...        ...            ...
609    LP002978  Female       No           0   Graduate             No             2
610    LP002979    Male      Yes          3+   Graduate             No             4
611    LP002983    Male      Yes           1   Graduate             No             8
612    LP002984    Male      Yes           2   Graduate             No             7
613    LP002990  Female       No           0   Graduate            Yes             4

614 rows × 13 columns
```

```
df.shape
```

```
(614, 13)
```

```
df.head()
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncom
0	LP001002	Male	No	0	Graduate	No	584
1	LP001003	Male	Yes	1	Graduate	No	458
2	LP001005	Male	Yes	0	Graduate	Yes	300
3	LP001006	Male	Yes	0	Not Graduate	No	258
4	LP001008	Male	No	0	Graduate	No	600

```
df.tail()
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantInc
609	LP002978	Female	No	0	Graduate	No	2
610	LP002979	Male	Yes	3+	Graduate	No	4
611	LP002983	Male	Yes	1	Graduate	No	8
612	LP002984	Male	Yes	2	Graduate	No	7
613	LP002990	Female	No	0	Graduate	Yes	4

```
df.dtypes
```

```
Loan_ID               object
Gender                object
Married               object
Dependents            object
Education             object
Self_Employed         object
ApplicantIncome        int64
CoapplicantIncome    float64
LoanAmount           float64
Loan_Amount_Term     float64
Credit_History       float64
Property_Area         object
Loan_Status           object
dtype: object
```

FINDING 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
```

```
#FOR FILLING MISSING VALUES
column=['Gender','Married','Dependents','Self_Employed','Loan_Amount_Term','Credit_History','LoanAmount']
for i in column:
  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
```

DROPPING OF COLUMNS

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

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapp
0	Male	No	0	Graduate	No	5849	
1	Male	Yes	1	Graduate	No	4583	
2	Male	Yes	0	Graduate	Yes	3000	
3	Male	Yes	0	Not Graduate	No	2583	
4	Male	No	0	Graduate	No	6000	
...	...	...	...	...	...	...	
609	Female	No	0	Graduate	No	2900	

IMPORTING LABEL ENCODER

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

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapp
0	1	0	0	0	0	5849	
1	1	1	1	0	0	4583	
2	1	1	0	0	1	3000	
3	1	1	0	1	0	2583	
4	1	0	0	0	0	6000	
...	...	...	...	...	...	...	
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	
611	1	1	1	0	0	8072	

SEPARATING X AND Y VARIABLES

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

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

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

SPLITTING INTO TRAINING AND TESTING DATA

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

```
     array([[  1.,   0.,   0., ..., 360.,   1.,   1.],
            [  1.,   1.,   2., ..., 180.,   1.,   1.],
            [  1.,   1.,   2., ..., 300.,   1.,   2.],
            ...,
            [  1.,   1.,   3., ..., 300.,   0.,   1.],
            [  1.,   0.,   0., ..., 360.,   1.,   0.],
            [  1.,   1.,   0., ..., 360.,   1.,   1.]])
```

```
x_test
```

```
array([[  1.,   1.,   1., ..., 360.,   1.,   0.],
       [  1.,   0.,   0., ..., 360.,   1.,   0.],
       [  1.,   1.,   0., ..., 180.,   1.,   2.],
       ...,
       [  1.,   1.,   0., ..., 360.,   1.,   2.],
       [  1.,   1.,   2., ..., 360.,   0.,   1.],
       [  1.,   1.,   3., ..., 300.,   1.,   2.]])
```

```
y_train
```

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

```
y_test
```

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

NORMALIZATION USING STANDARD SCALER

```
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([[ 0.46028731, -1.35685652, -0.7418818 , ...,  0.28421184,
              0.40323892, -0.06786108],
            [ 0.46028731,  0.7369976 ,  1.25979929, ..., -2.64203329,
              0.40323892, -0.06786108],
            [ 0.46028731,  0.7369976 ,  1.25979929, ..., -0.6912032 ,
              0.40323892,  1.19789552],
            ...,
            [ 0.46028731,  0.7369976 ,  2.26063983, ..., -0.6912032 ,
             -2.47991935, -0.06786108],
            [ 0.46028731, -1.35685652, -0.7418818 , ...,  0.28421184,
              0.40323892, -1.33361768],
            [ 0.46028731,  0.7369976 , -0.7418818 , ...,  0.28421184,
              0.40323892, -0.06786108]])
```

MODEL CREATION

KNN,NAIVE BAYES AND SVM MODEL

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix,accuracy_score,classification_report
knn_model=KNeighborsClassifier()
nb_model=GaussianNB()
svm_model=SVC()
fnl_model=[knn_model,nb_model,svm_model]
```

PERFORMANCE EVALUATION OF KNN, NAIVE BAYES AND SVM MODEL

```
for i in fnl_model:
  print(i)
  i.fit(x_train,y_train)
  y_pred=i.predict(x_test)
  print("ACCURACY SCORE",accuracy_score(y_test,y_pred))
  print("****************************************************************")
  print("CLASSIFICATION REPORT",classification_report(y_test,y_pred))
```

```
KNeighborsClassifier()
ACCURACY SCORE 0.7567567567567568
****************************************************************
CLASSIFICATION REPORT               precision    recall  f1-score   support

           0       0.81      0.39      0.53        64
           1       0.75      0.95      0.84       121

    accuracy                           0.76       185
   macro avg       0.78      0.67      0.68       185
weighted avg       0.77      0.76      0.73       185

GaussianNB()
ACCURACY SCORE 0.772972972972973
****************************************************************
CLASSIFICATION REPORT               precision    recall  f1-score   support

           0       0.82      0.44      0.57        64
           1       0.76      0.95      0.85       121

    accuracy                           0.77       185
   macro avg       0.79      0.69      0.71       185
```

```
weighted avg       0.78      0.77      0.75       185

SVC()
ACCURACY SCORE 0.7675675675675676
****************************************************************
CLASSIFICATION REPORT               precision    recall  f1-score   support

           0       0.86      0.39      0.54        64
           1       0.75      0.97      0.84       121

    accuracy                           0.77       185
   macro avg       0.81      0.68      0.69       185
weighted avg       0.79      0.77      0.74       185
```

✓ 0s completed at 8:23 PM