In [1]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount,
call drive.mount("/content/drive", force_remount=True).
```

IMPORTING THE LIBRARIES :-

In [2]:
```
import pandas as pd
import numpy as np
import pickle
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import sklearn
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import GradientBoostingClassifier, RandomForestClas
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import RandomizedSearchCV
import imblearn
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report, confus
import tensorflow
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Sequential
import warnings
warnings.filterwarnings('ignore')
```

READING THE DATASET :-

In [3]:
```
df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/Saran/train_u6lu
```

In [4]:
```
df.head()
```

Out[4]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Co
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	

In [5]:
```
df.drop('Loan_ID',axis=1,inplace=True)
```

In [6]:
```
df.head()
```

Out[6]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantInc
0	Male	No	0	Graduate	No	5849	
1	Male	Yes	1	Graduate	No	4583	15
2	Male	Yes	0	Graduate	Yes	3000	
3	Male	Yes	0	Not Graduate	No	2583	23
4	Male	No	0	Graduate	No	6000	

In [7]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Gender             601 non-null    object 
 1   Married            611 non-null    object 
 2   Dependents         599 non-null    object 
 3   Education          614 non-null    object 
 4   Self_Employed      582 non-null    object 
 5   ApplicantIncome    614 non-null    int64  
 6   CoapplicantIncome  614 non-null    float64
 7   LoanAmount         592 non-null    float64
 8   Loan_Amount_Term   600 non-null    float64
 9   Credit_History     564 non-null    float64
 10  Property_Area      614 non-null    object 
 11  Loan_Status        614 non-null    object 
dtypes: float64(4), int64(1), object(7)
memory usage: 57.7+ KB
```

In [8]:

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

Out[8]:

```
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
```

In [9]:

```
cat_cols = ['Gender','Married','Dependents','Education','Self_Employed','
num_cols = ['ApplicantIncome','CoapplicantIncome','LoanAmount','Loan_Amou

for col in cat_cols:
  df[col] = df[col].fillna(df[col].mode()[0])

for col in num_cols:
  df[col] = df[col].fillna(df[col].mean())
```

In [10]:
```
df.isnull().sum()
```

Out[10]:
```
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
```

In [11]:
```
for col in cat_cols:
  print(col)
  print(df[col].unique())
  print('\n')
```

```
Gender
['Male' 'Female']


Married
['No' 'Yes']


Dependents
['0' '1' '2' '3+']


Education
['Graduate' 'Not Graduate']


Self_Employed
['No' 'Yes']


Property_Area
['Urban' 'Rural' 'Semiurban']


Loan_Status
['Y' 'N']
```

HANDLING CATEGORICAL VALUES :-

In [12]:
```
df.Gender.replace(['Male','Female'],[0,1],inplace=True)
```

In [13]:
```
df.Married.replace(['No','Yes'],[0,1],inplace=True)
```

In [14]:
```
df.Dependents = df.Dependents.str.replace('+','')
```

In [15]:
```
df.Education.replace(['Graduate','Not Graduate'],[0,1],inplace=True)
```

In [16]:
```
df.Self_Employed.replace(['No','Yes'],[0,1],inplace=True)
```

In [17]:
```
df.Property_Area.replace(['Urban','Rural', 'Semiurban'],[0,1,2],inplace=T
```

In [18]:
```
df.Loan_Status.replace(['Y','N'],[1,0],inplace=True)
```

In [19]:
```
df.head()
```

Out[19]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantInc
0	0	0	0	0	0	5849	
1	0	1	1	0	0	4583	15
2	0	1	0	0	1	3000	
3	0	1	0	1	0	2583	23
4	0	0	0	0	0	6000	

In [20]:
```
for col in cat_cols:
  df[col] = df[col].astype('int')
```

In [21]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Gender             614 non-null    int64
 1   Married            614 non-null    int64
 2   Dependents         614 non-null    int64
 3   Education          614 non-null    int64
 4   Self_Employed      614 non-null    int64
 5   ApplicantIncome    614 non-null    int64
 6   CoapplicantIncome  614 non-null    float64
 7   LoanAmount         614 non-null    float64
 8   Loan_Amount_Term   614 non-null    float64
 9   Credit_History     614 non-null    float64
 10  Property_Area      614 non-null    int64
 11  Loan_Status        614 non-null    int64
dtypes: float64(4), int64(8)
memory usage: 57.7 KB
```

In [22]:
```
from imblearn.combine import SMOTETomek

smote = SMOTETomek()

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

x_bal,y_bal = smote.fit_resample(x,y)

print(y.value_counts())
print(y_bal.value_counts())
```

```
1    422
0    192
Name: Loan_Status, dtype: int64
1    358
0    358
Name: Loan_Status, dtype: int64
```

EDA - EXPLORATORY DATA ANALYSIS :-

In [23]:
```
df.describe()
```

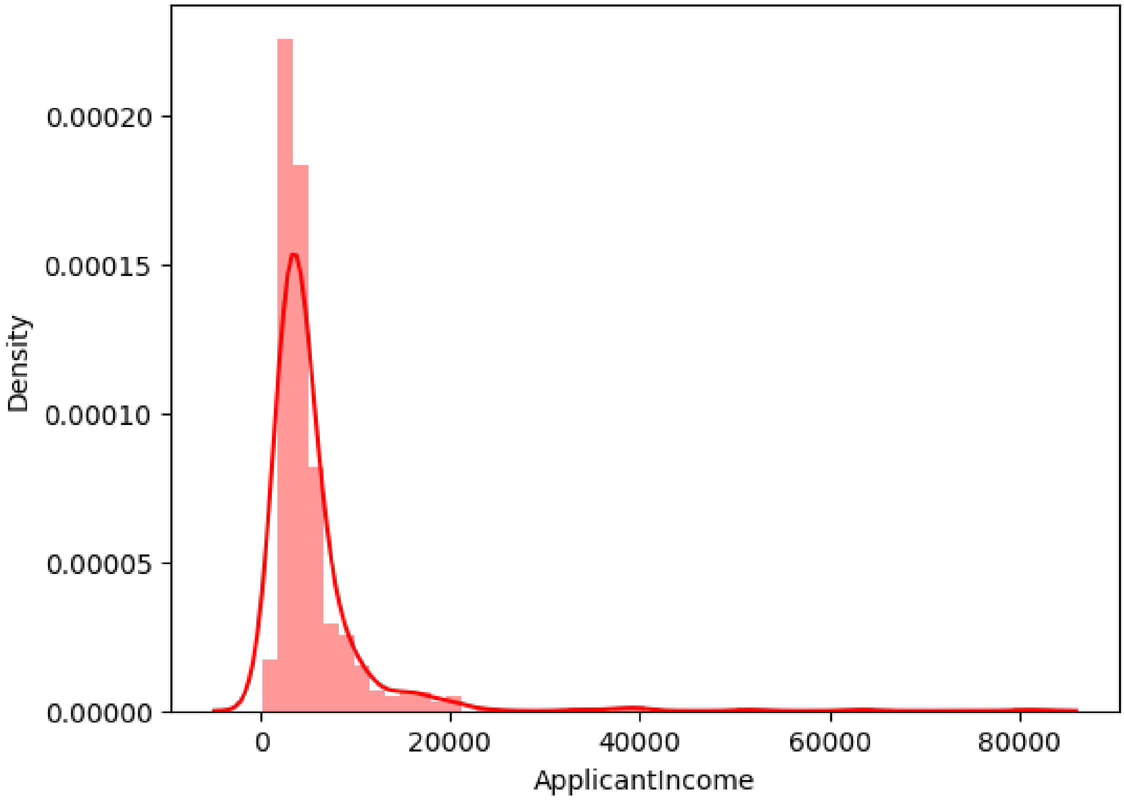
Out[23]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	C
count	614.000000	614.000000	614.000000	614.000000	614.000000	614.000000	
mean	0.182410	0.653094	0.744300	0.218241	0.133550	5403.459283	
std	0.386497	0.476373	1.009623	0.413389	0.340446	6109.041673	
min	0.000000	0.000000	0.000000	0.000000	0.000000	150.000000	
25%	0.000000	0.000000	0.000000	0.000000	0.000000	2877.500000	
50%	0.000000	1.000000	0.000000	0.000000	0.000000	3812.500000	
75%	0.000000	1.000000	1.000000	0.000000	0.000000	5795.000000	
max	1.000000	1.000000	3.000000	1.000000	1.000000	81000.000000	

UNIVARIATE ANALYSIS :-

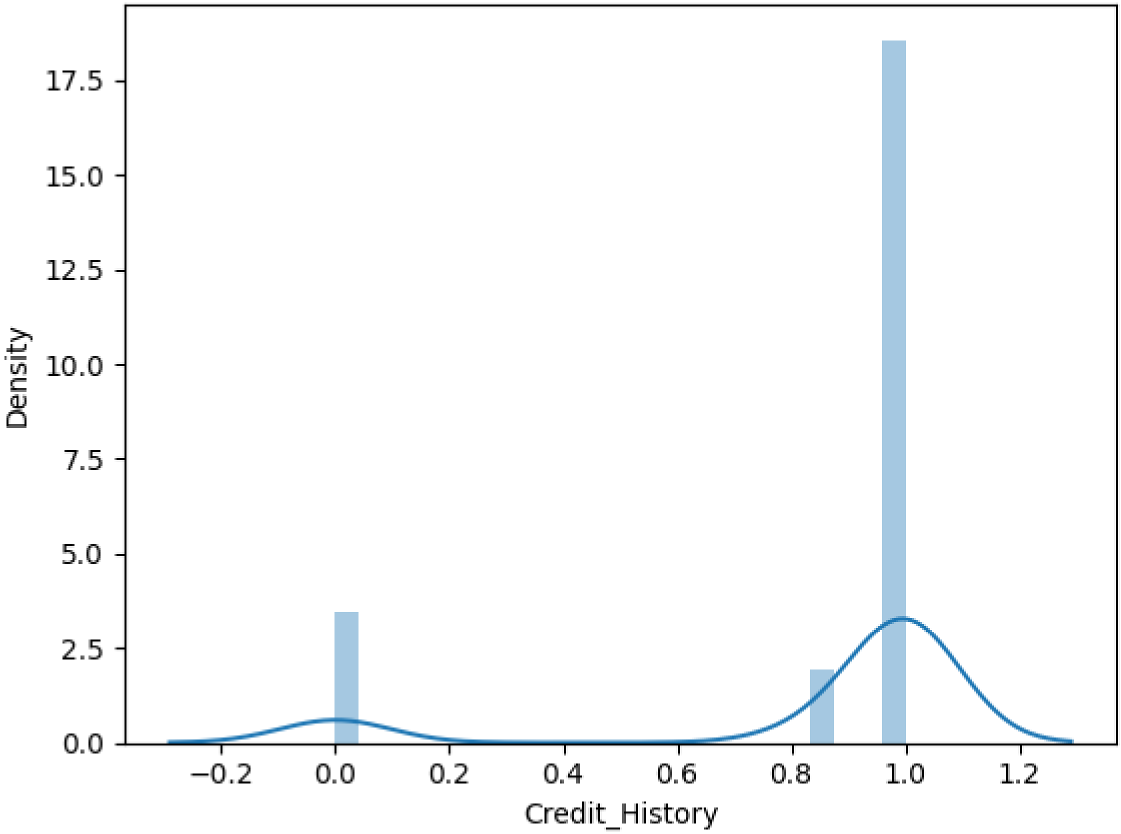
In [24]:
```
sns.distplot(df['ApplicantIncome'], color='r')
```

Out[24]: <Axes: xlabel='ApplicantIncome', ylabel='Density'>



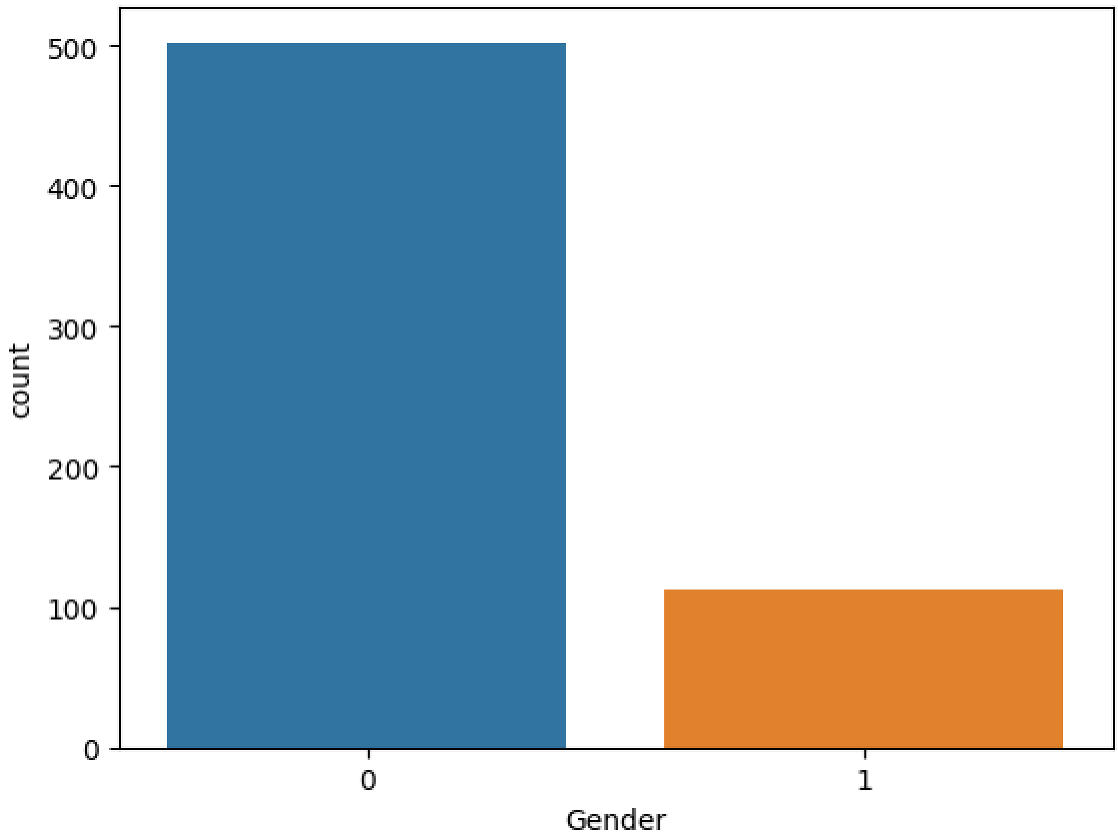
In [25]:
```
sns.distplot(df['Credit_History'])
```

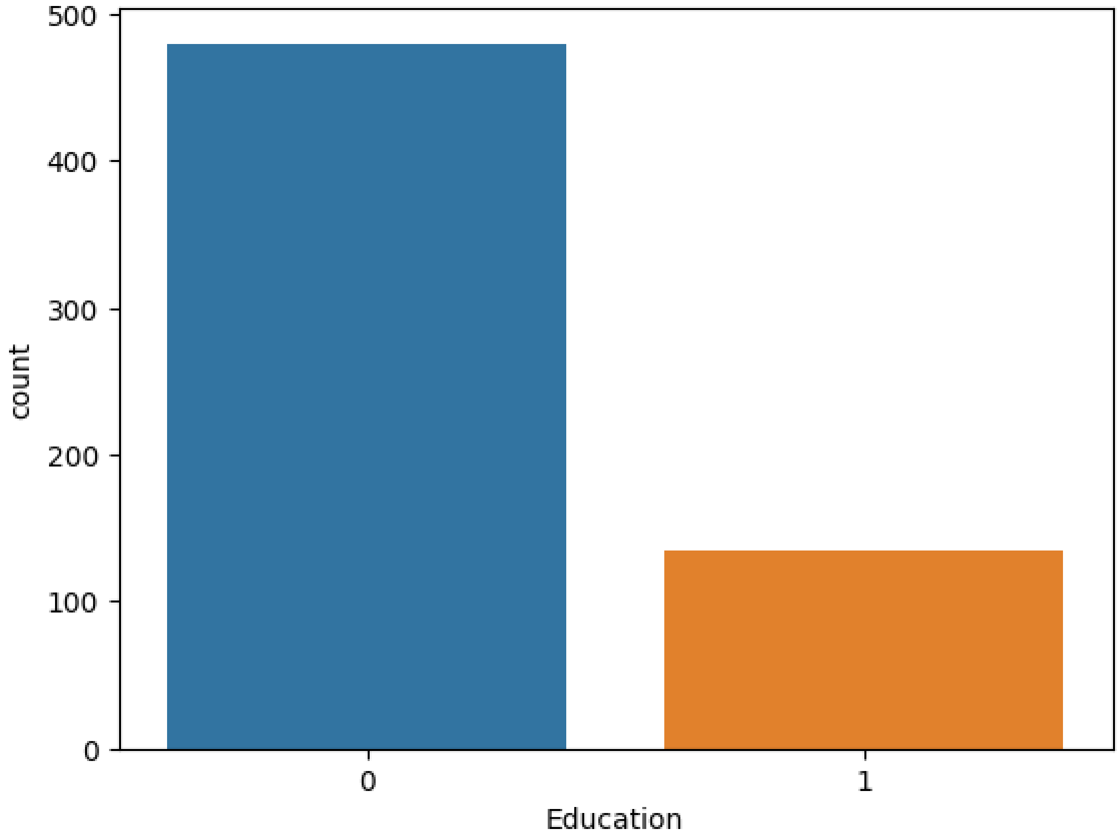
Out[25]: <Axes: xlabel='Credit_History', ylabel='Density'>



In [26]:

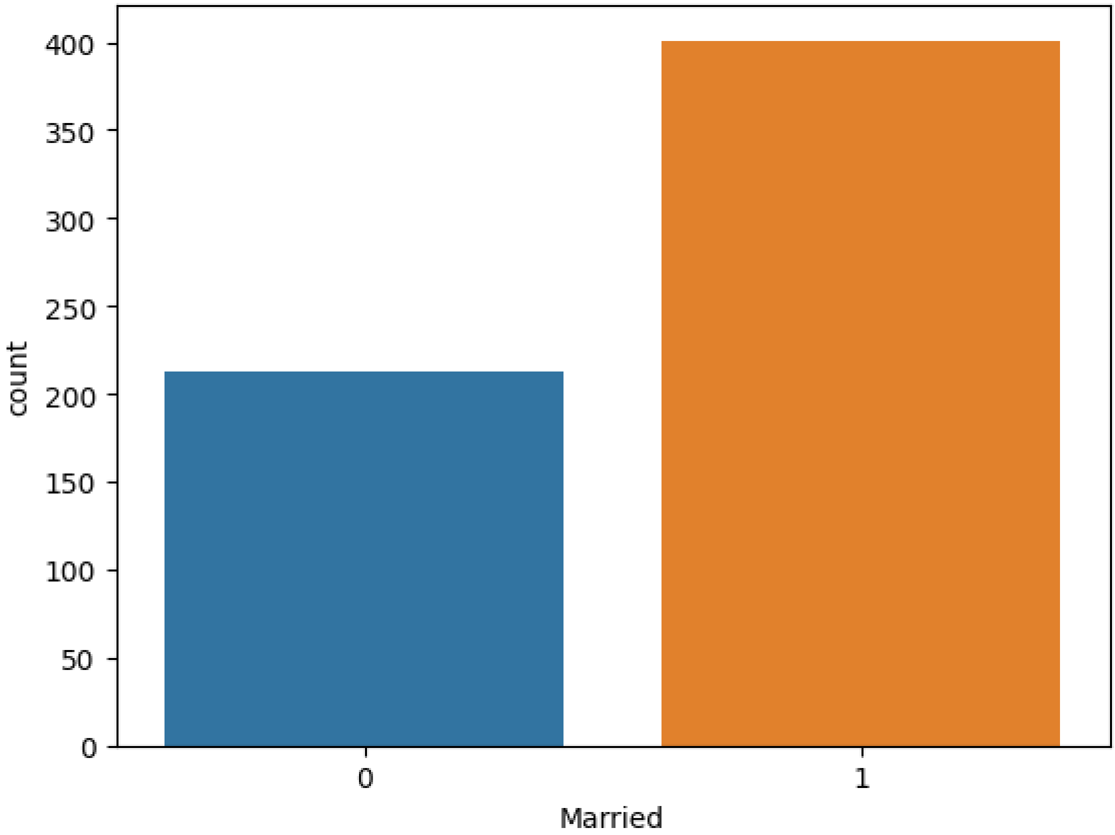
```
sns.countplot(df,x='Gender')
plt.xlabel('Gender')
plt.show()
sns.countplot(df,x='Education')
plt.xlabel('Education')
plt.show()
```





In [27]:
```
sns.countplot(df,x='Married')
```

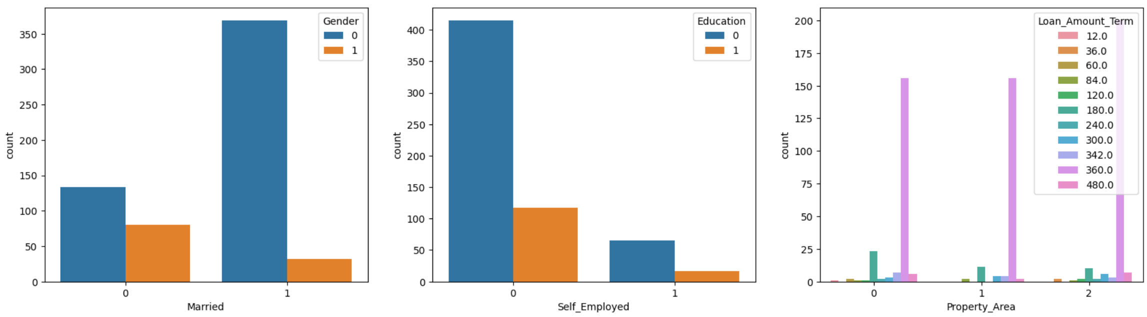
Out[27]: <Axes: xlabel='Married', ylabel='count'>



BIVARIATE ANALYSIS :-

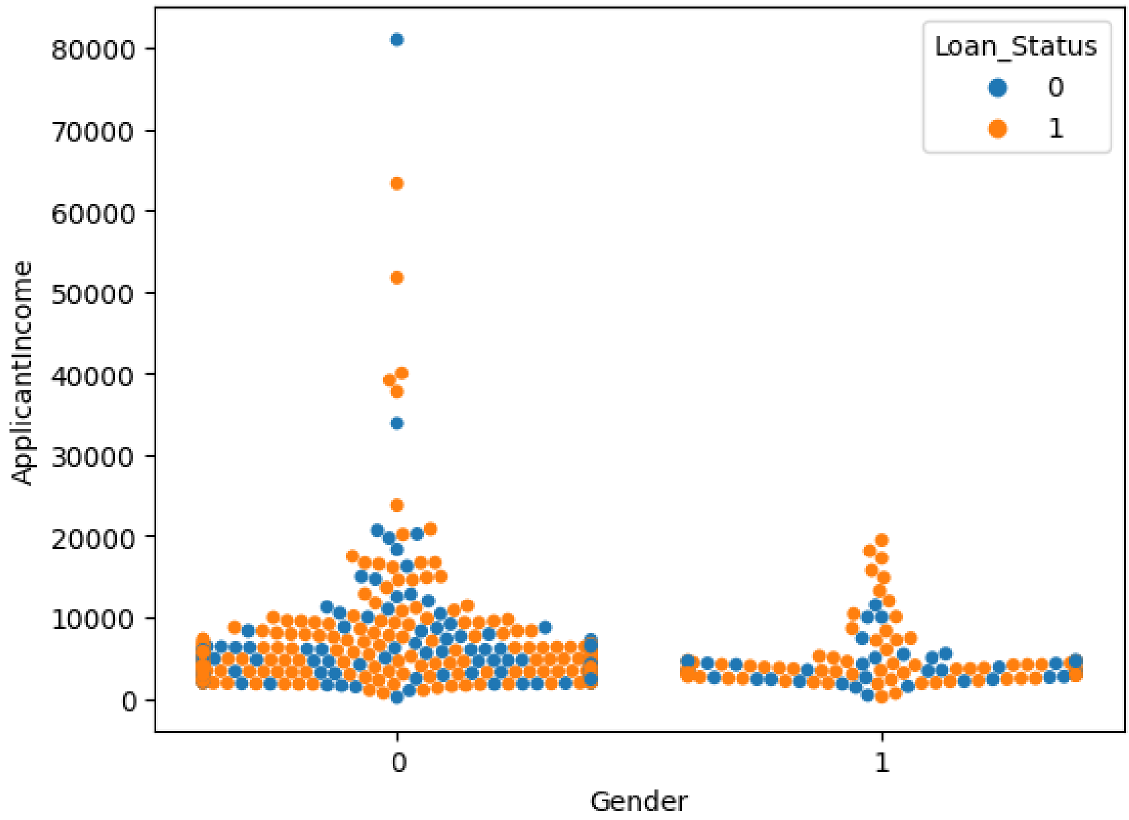
In [28]:
```
plt.figure(figsize=(20,5))
plt.subplot(1,3,1)
sns.countplot(x= 'Married', hue = "Gender", data = df)
plt.subplot(1,3,2)
sns.countplot(x = 'Self_Employed', hue = "Education", data = df)
plt.subplot(1,3,3)
sns.countplot(x= 'Property_Area', hue = "Loan_Amount_Term", data =df)
```

Out[28]: <Axes: xlabel='Property_Area', ylabel='count'>



In [29]:
```
sns.swarmplot(x='Gender', y='ApplicantIncome', hue = "Loan_Status", data
```

Out[29]: <Axes: xlabel='Gender', ylabel='ApplicantIncome'>



SCALING THE DATA :-

In [30]:

```
scaler = StandardScaler()
x_bal = scaler.fit_transform(x_bal)
x_bal = pd.DataFrame(x_bal,columns=scaler.get_feature_names_out())
x_bal.head()
```

Out[30]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapplicar
0	-0.441956	-1.137924	-0.705071	-0.475423	-0.334367	0.152709	-
1	-0.441956	0.878793	0.357731	-0.475423	-0.334367	-0.090557	-
2	-0.441956	0.878793	-0.705071	-0.475423	2.990726	-0.394735	-
3	-0.441956	0.878793	-0.705071	2.103388	-0.334367	-0.474863	
4	-0.441956	-1.137924	-0.705071	-0.475423	-0.334367	0.181724	-

SPLITTING THE DATA INTO TRAINING AND TESTTING DATA :-

In [31]:

```
x_train,x_test,y_train,y_test=train_test_split(x_bal,y_bal, test_size=0.3
print(x_train.shape,x_test.shape)
```

```
(479, 11) (237, 11)
```

MODEL BUILDING :-

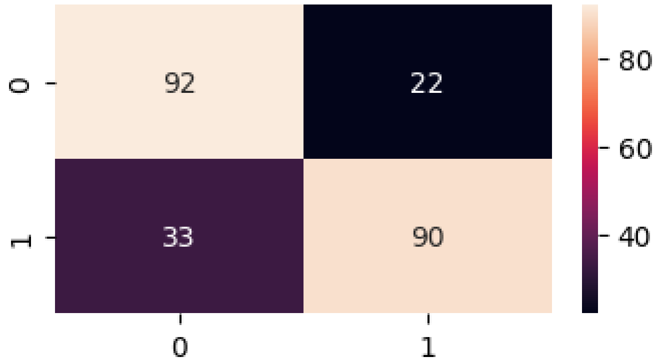
In [32]:

```
def fit_model(model,name):
  model.fit(x_train,y_train)

  y_pred = model.predict(x_train)
  print('training accuracy of ',name,' : ',accuracy_score(y_pred,y_train)
```

```
    y_pred = model.predict(x_test)
    print('testing accuracy of ',name,' : ',accuracy_score(y_pred, y_test))

    plt.figure(figsize=(4,2))
    sns.heatmap(confusion_matrix(y_test,y_pred),annot=True)
    plt.show()

    print(classification_report(y_test,y_pred))
```

In [33]:
```
dtc = DecisionTreeClassifier()
fit_model(dtc,"DTC")
```

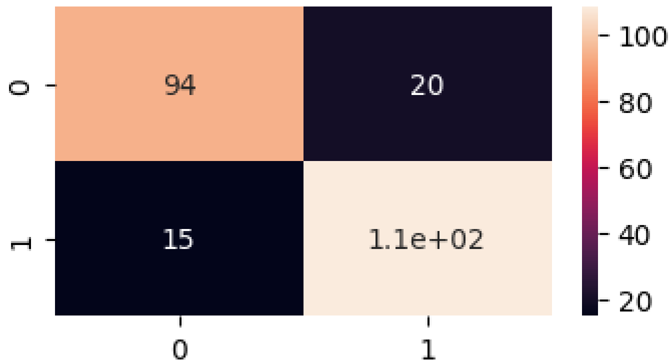
```
training accuracy of  DTC  :  1.0
testing accuracy of  DTC  :  0.7679324894514767
```




```
              precision    recall  f1-score   support

           0       0.74      0.81      0.77       114
           1       0.80      0.73      0.77       123

    accuracy                           0.77       237
   macro avg       0.77      0.77      0.77       237
weighted avg       0.77      0.77      0.77       237
```

In [34]:
```
rfc = RandomForestClassifier()
fit_model(rfc,"RFC")
```

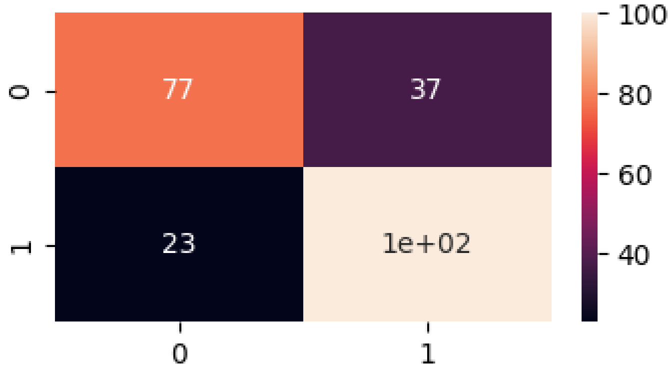
```
training accuracy of  RFC  :  1.0
testing accuracy of  RFC  :  0.8523206751054853
```



```
              precision    recall  f1-score   support

           0       0.86      0.82      0.84       114
           1       0.84      0.88      0.86       123

    accuracy                           0.85       237
   macro avg       0.85      0.85      0.85       237
weighted avg       0.85      0.85      0.85       237
```

In [35]:
```python
knn = KNeighborsClassifier()
fit_model(knn,'KNN')
```

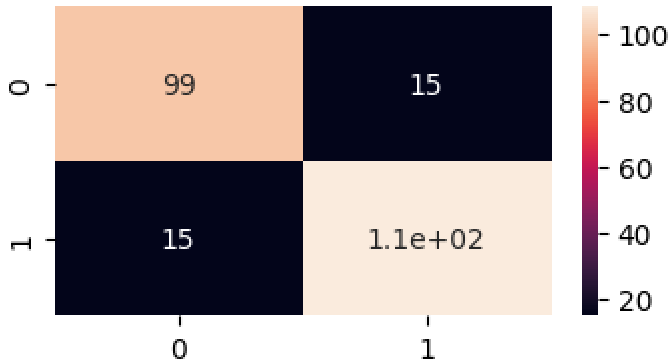
```
training accuracy of  KNN  :  0.824634655532359
testing accuracy of  KNN  :  0.7468354430379747
```




```
              precision    recall  f1-score   support

           0       0.77      0.68      0.72       114
           1       0.73      0.81      0.77       123

    accuracy                           0.75       237
   macro avg       0.75      0.74      0.74       237
weighted avg       0.75      0.75      0.75       237
```

In [36]:
```python
from xgboost import XGBClassifier
xgb = XGBClassifier()
fit_model(xgb,'XGB')
```

```
training accuracy of  XGB  :  1.0
testing accuracy of  XGB  :  0.8734177215189873
```



```
              precision    recall  f1-score   support

           0       0.87      0.87      0.87       114
           1       0.88      0.88      0.88       123

    accuracy                           0.87       237
   macro avg       0.87      0.87      0.87       237
weighted avg       0.87      0.87      0.87       237
```

In [37]:
```
ann = Sequential()
ann.add(Dense(units=12,activation='relu'))
ann.add(Dense(units=24,activation='relu'))
ann.add(Dense(units=1,activation='sigmoid'))
ann.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accur
ann.fit(x_train,y_train,batch_size=100,validation_split=0.2,epochs=100)
```

```
Epoch 1/100
4/4 [==============================] - 1s 75ms/step - loss: 0.7652 - acc
uracy: 0.3708 - val_loss: 0.7232 - val_accuracy: 0.4271
Epoch 2/100
4/4 [==============================] - 0s 11ms/step - loss: 0.7517 - acc
uracy: 0.3812 - val_loss: 0.7183 - val_accuracy: 0.4479
Epoch 3/100
4/4 [==============================] - 0s 13ms/step - loss: 0.7388 - acc
uracy: 0.4178 - val_loss: 0.7138 - val_accuracy: 0.4167
Epoch 4/100
4/4 [==============================] - 0s 10ms/step - loss: 0.7280 - acc
uracy: 0.4439 - val_loss: 0.7099 - val_accuracy: 0.4375
Epoch 5/100
4/4 [==============================] - 0s 10ms/step - loss: 0.7166 - acc
uracy: 0.4909 - val_loss: 0.7057 - val_accuracy: 0.4688
Epoch 6/100
4/4 [==============================] - 0s 11ms/step - loss: 0.7073 - acc
uracy: 0.5091 - val_loss: 0.7019 - val_accuracy: 0.4688
Epoch 7/100
4/4 [==============================] - 0s 15ms/step - loss: 0.6985 - acc
uracy: 0.5274 - val_loss: 0.6981 - val_accuracy: 0.5208
Epoch 8/100
4/4 [==============================] - 0s 11ms/step - loss: 0.6905 - acc
uracy: 0.5352 - val_loss: 0.6946 - val_accuracy: 0.5729
Epoch 9/100
4/4 [==============================] - 0s 10ms/step - loss: 0.6823 - acc
uracy: 0.5587 - val_loss: 0.6910 - val_accuracy: 0.5521
Epoch 10/100
4/4 [==============================] - 0s 10ms/step - loss: 0.6741 - acc
uracy: 0.5979 - val_loss: 0.6872 - val_accuracy: 0.5729
Epoch 11/100
4/4 [==============================] - 0s 10ms/step - loss: 0.6664 - acc
uracy: 0.6162 - val_loss: 0.6836 - val_accuracy: 0.6042
Epoch 12/100
4/4 [==============================] - 0s 11ms/step - loss: 0.6593 - acc
uracy: 0.6397 - val_loss: 0.6800 - val_accuracy: 0.6146
Epoch 13/100
4/4 [==============================] - 0s 14ms/step - loss: 0.6516 - acc
uracy: 0.6423 - val_loss: 0.6763 - val_accuracy: 0.6250
Epoch 14/100
4/4 [==============================] - 0s 11ms/step - loss: 0.6445 - acc
uracy: 0.6632 - val_loss: 0.6727 - val_accuracy: 0.6458
Epoch 15/100
4/4 [==============================] - 0s 10ms/step - loss: 0.6371 - acc
uracy: 0.6684 - val_loss: 0.6689 - val_accuracy: 0.6458
Epoch 16/100
4/4 [==============================] - 0s 10ms/step - loss: 0.6296 - acc
uracy: 0.6971 - val_loss: 0.6650 - val_accuracy: 0.6354
Epoch 17/100
4/4 [==============================] - 0s 10ms/step - loss: 0.6226 - acc
uracy: 0.6997 - val_loss: 0.6613 - val_accuracy: 0.6458
Epoch 18/100
4/4 [==============================] - 0s 10ms/step - loss: 0.6154 - acc
uracy: 0.7076 - val_loss: 0.6578 - val_accuracy: 0.6458
Epoch 19/100
4/4 [==============================] - 0s 11ms/step - loss: 0.6086 - acc
uracy: 0.7232 - val_loss: 0.6548 - val_accuracy: 0.6458
Epoch 20/100
4/4 [==============================] - 0s 11ms/step - loss: 0.6018 - acc
uracy: 0.7232 - val_loss: 0.6512 - val_accuracy: 0.6562
```

```
Epoch 21/100
4/4 [==============================] - 0s 17ms/step - loss: 0.5947 - acc
uracy: 0.7311 - val_loss: 0.6481 - val_accuracy: 0.6562
Epoch 22/100
4/4 [==============================] - 0s 11ms/step - loss: 0.5877 - acc
uracy: 0.7363 - val_loss: 0.6449 - val_accuracy: 0.6562
Epoch 23/100
4/4 [==============================] - 0s 11ms/step - loss: 0.5813 - acc
uracy: 0.7493 - val_loss: 0.6422 - val_accuracy: 0.6458
Epoch 24/100
4/4 [==============================] - 0s 11ms/step - loss: 0.5743 - acc
uracy: 0.7598 - val_loss: 0.6394 - val_accuracy: 0.6458
Epoch 25/100
4/4 [==============================] - 0s 17ms/step - loss: 0.5675 - acc
uracy: 0.7650 - val_loss: 0.6368 - val_accuracy: 0.6562
Epoch 26/100
4/4 [==============================] - 0s 11ms/step - loss: 0.5609 - acc
uracy: 0.7702 - val_loss: 0.6347 - val_accuracy: 0.6562
Epoch 27/100
4/4 [==============================] - 0s 18ms/step - loss: 0.5545 - acc
uracy: 0.7702 - val_loss: 0.6330 - val_accuracy: 0.6562
Epoch 28/100
4/4 [==============================] - 0s 11ms/step - loss: 0.5483 - acc
uracy: 0.7728 - val_loss: 0.6310 - val_accuracy: 0.6458
Epoch 29/100
4/4 [==============================] - 0s 10ms/step - loss: 0.5421 - acc
uracy: 0.7755 - val_loss: 0.6292 - val_accuracy: 0.6458
Epoch 30/100
4/4 [==============================] - 0s 10ms/step - loss: 0.5364 - acc
uracy: 0.7807 - val_loss: 0.6276 - val_accuracy: 0.6458
Epoch 31/100
4/4 [==============================] - 0s 10ms/step - loss: 0.5305 - acc
uracy: 0.7807 - val_loss: 0.6265 - val_accuracy: 0.6458
Epoch 32/100
4/4 [==============================] - 0s 10ms/step - loss: 0.5249 - acc
uracy: 0.7833 - val_loss: 0.6253 - val_accuracy: 0.6458
Epoch 33/100
4/4 [==============================] - 0s 11ms/step - loss: 0.5195 - acc
uracy: 0.7833 - val_loss: 0.6246 - val_accuracy: 0.6458
Epoch 34/100
4/4 [==============================] - 0s 13ms/step - loss: 0.5144 - acc
uracy: 0.7833 - val_loss: 0.6235 - val_accuracy: 0.6562
Epoch 35/100
4/4 [==============================] - 0s 11ms/step - loss: 0.5098 - acc
uracy: 0.7833 - val_loss: 0.6232 - val_accuracy: 0.6562
Epoch 36/100
4/4 [==============================] - 0s 10ms/step - loss: 0.5047 - acc
uracy: 0.7859 - val_loss: 0.6228 - val_accuracy: 0.6562
Epoch 37/100
4/4 [==============================] - 0s 10ms/step - loss: 0.5002 - acc
uracy: 0.7833 - val_loss: 0.6226 - val_accuracy: 0.6458
Epoch 38/100
4/4 [==============================] - 0s 10ms/step - loss: 0.4955 - acc
uracy: 0.7833 - val_loss: 0.6221 - val_accuracy: 0.6458
Epoch 39/100
4/4 [==============================] - 0s 10ms/step - loss: 0.4916 - acc
uracy: 0.7885 - val_loss: 0.6215 - val_accuracy: 0.6458
Epoch 40/100
4/4 [==============================] - 0s 15ms/step - loss: 0.4875 - acc
uracy: 0.7911 - val_loss: 0.6214 - val_accuracy: 0.6354
```

```
Epoch 41/100
4/4 [==============================] - 0s 19ms/step - loss: 0.4838 - acc
uracy: 0.7911 - val_loss: 0.6216 - val_accuracy: 0.6354
Epoch 42/100
4/4 [==============================] - 0s 12ms/step - loss: 0.4800 - acc
uracy: 0.7885 - val_loss: 0.6214 - val_accuracy: 0.6354
Epoch 43/100
4/4 [==============================] - 0s 19ms/step - loss: 0.4767 - acc
uracy: 0.7911 - val_loss: 0.6210 - val_accuracy: 0.6458
Epoch 44/100
4/4 [==============================] - 0s 11ms/step - loss: 0.4734 - acc
uracy: 0.7911 - val_loss: 0.6206 - val_accuracy: 0.6458
Epoch 45/100
4/4 [==============================] - 0s 11ms/step - loss: 0.4705 - acc
uracy: 0.7937 - val_loss: 0.6213 - val_accuracy: 0.6458
Epoch 46/100
4/4 [==============================] - 0s 11ms/step - loss: 0.4672 - acc
uracy: 0.7911 - val_loss: 0.6215 - val_accuracy: 0.6458
Epoch 47/100
4/4 [==============================] - 0s 12ms/step - loss: 0.4648 - acc
uracy: 0.7911 - val_loss: 0.6217 - val_accuracy: 0.6458
Epoch 48/100
4/4 [==============================] - 0s 11ms/step - loss: 0.4621 - acc
uracy: 0.7911 - val_loss: 0.6223 - val_accuracy: 0.6458
Epoch 49/100
4/4 [==============================] - 0s 13ms/step - loss: 0.4597 - acc
uracy: 0.7911 - val_loss: 0.6227 - val_accuracy: 0.6458
Epoch 50/100
4/4 [==============================] - 0s 11ms/step - loss: 0.4573 - acc
uracy: 0.7963 - val_loss: 0.6234 - val_accuracy: 0.6458
Epoch 51/100
4/4 [==============================] - 0s 16ms/step - loss: 0.4552 - acc
uracy: 0.7990 - val_loss: 0.6230 - val_accuracy: 0.6458
Epoch 52/100
4/4 [==============================] - 0s 11ms/step - loss: 0.4530 - acc
uracy: 0.8042 - val_loss: 0.6232 - val_accuracy: 0.6458
Epoch 53/100
4/4 [==============================] - 0s 10ms/step - loss: 0.4508 - acc
uracy: 0.8016 - val_loss: 0.6236 - val_accuracy: 0.6458
Epoch 54/100
4/4 [==============================] - 0s 11ms/step - loss: 0.4491 - acc
uracy: 0.8042 - val_loss: 0.6239 - val_accuracy: 0.6458
Epoch 55/100
4/4 [==============================] - 0s 11ms/step - loss: 0.4471 - acc
uracy: 0.8016 - val_loss: 0.6242 - val_accuracy: 0.6458
Epoch 56/100
4/4 [==============================] - 0s 11ms/step - loss: 0.4455 - acc
uracy: 0.8016 - val_loss: 0.6244 - val_accuracy: 0.6458
Epoch 57/100
4/4 [==============================] - 0s 11ms/step - loss: 0.4437 - acc
uracy: 0.8016 - val_loss: 0.6240 - val_accuracy: 0.6562
Epoch 58/100
4/4 [==============================] - 0s 11ms/step - loss: 0.4419 - acc
uracy: 0.8042 - val_loss: 0.6242 - val_accuracy: 0.6667
Epoch 59/100
4/4 [==============================] - 0s 11ms/step - loss: 0.4405 - acc
uracy: 0.8042 - val_loss: 0.6253 - val_accuracy: 0.6667
Epoch 60/100
4/4 [==============================] - 0s 11ms/step - loss: 0.4391 - acc
uracy: 0.8068 - val_loss: 0.6256 - val_accuracy: 0.6667
```

```
Epoch 61/100
4/4 [==============================] - 0s 11ms/step - loss: 0.4378 - acc
uracy: 0.8068 - val_loss: 0.6252 - val_accuracy: 0.6667
Epoch 62/100
4/4 [==============================] - 0s 11ms/step - loss: 0.4364 - acc
uracy: 0.8068 - val_loss: 0.6261 - val_accuracy: 0.6667
Epoch 63/100
4/4 [==============================] - 0s 10ms/step - loss: 0.4352 - acc
uracy: 0.8068 - val_loss: 0.6274 - val_accuracy: 0.6667
Epoch 64/100
4/4 [==============================] - 0s 10ms/step - loss: 0.4338 - acc
uracy: 0.8068 - val_loss: 0.6269 - val_accuracy: 0.6667
Epoch 65/100
4/4 [==============================] - 0s 10ms/step - loss: 0.4326 - acc
uracy: 0.8068 - val_loss: 0.6271 - val_accuracy: 0.6667
Epoch 66/100
4/4 [==============================] - 0s 11ms/step - loss: 0.4316 - acc
uracy: 0.8042 - val_loss: 0.6279 - val_accuracy: 0.6667
Epoch 67/100
4/4 [==============================] - 0s 16ms/step - loss: 0.4305 - acc
uracy: 0.8068 - val_loss: 0.6280 - val_accuracy: 0.6667
Epoch 68/100
4/4 [==============================] - 0s 11ms/step - loss: 0.4296 - acc
uracy: 0.8042 - val_loss: 0.6285 - val_accuracy: 0.6667
Epoch 69/100
4/4 [==============================] - 0s 12ms/step - loss: 0.4287 - acc
uracy: 0.8042 - val_loss: 0.6287 - val_accuracy: 0.6667
Epoch 70/100
4/4 [==============================] - 0s 11ms/step - loss: 0.4275 - acc
uracy: 0.8042 - val_loss: 0.6278 - val_accuracy: 0.6667
Epoch 71/100
4/4 [==============================] - 0s 10ms/step - loss: 0.4266 - acc
uracy: 0.8068 - val_loss: 0.6287 - val_accuracy: 0.6667
Epoch 72/100
4/4 [==============================] - 0s 13ms/step - loss: 0.4256 - acc
uracy: 0.8068 - val_loss: 0.6289 - val_accuracy: 0.6667
Epoch 73/100
4/4 [==============================] - 0s 11ms/step - loss: 0.4249 - acc
uracy: 0.8068 - val_loss: 0.6289 - val_accuracy: 0.6667
Epoch 74/100
4/4 [==============================] - 0s 11ms/step - loss: 0.4239 - acc
uracy: 0.8068 - val_loss: 0.6308 - val_accuracy: 0.6667
Epoch 75/100
4/4 [==============================] - 0s 12ms/step - loss: 0.4233 - acc
uracy: 0.8042 - val_loss: 0.6331 - val_accuracy: 0.6667
Epoch 76/100
4/4 [==============================] - 0s 10ms/step - loss: 0.4221 - acc
uracy: 0.8042 - val_loss: 0.6327 - val_accuracy: 0.6667
Epoch 77/100
4/4 [==============================] - 0s 11ms/step - loss: 0.4213 - acc
uracy: 0.8042 - val_loss: 0.6322 - val_accuracy: 0.6667
Epoch 78/100
4/4 [==============================] - 0s 11ms/step - loss: 0.4206 - acc
uracy: 0.8042 - val_loss: 0.6313 - val_accuracy: 0.6667
Epoch 79/100
4/4 [==============================] - 0s 10ms/step - loss: 0.4195 - acc
uracy: 0.8042 - val_loss: 0.6323 - val_accuracy: 0.6667
Epoch 80/100
4/4 [==============================] - 0s 11ms/step - loss: 0.4188 - acc
uracy: 0.8042 - val_loss: 0.6329 - val_accuracy: 0.6667
```

```
Epoch 81/100
4/4 [==============================] - 0s 11ms/step - loss: 0.4178 - acc
uracy: 0.8042 - val_loss: 0.6330 - val_accuracy: 0.6667
Epoch 82/100
4/4 [==============================] - 0s 10ms/step - loss: 0.4170 - acc
uracy: 0.8042 - val_loss: 0.6333 - val_accuracy: 0.6667
Epoch 83/100
4/4 [==============================] - 0s 11ms/step - loss: 0.4163 - acc
uracy: 0.8042 - val_loss: 0.6339 - val_accuracy: 0.6667
Epoch 84/100
4/4 [==============================] - 0s 11ms/step - loss: 0.4155 - acc
uracy: 0.8042 - val_loss: 0.6336 - val_accuracy: 0.6667
Epoch 85/100
4/4 [==============================] - 0s 13ms/step - loss: 0.4148 - acc
uracy: 0.8042 - val_loss: 0.6335 - val_accuracy: 0.6667
Epoch 86/100
4/4 [==============================] - 0s 14ms/step - loss: 0.4141 - acc
uracy: 0.8042 - val_loss: 0.6341 - val_accuracy: 0.6667
Epoch 87/100
4/4 [==============================] - 0s 12ms/step - loss: 0.4136 - acc
uracy: 0.8068 - val_loss: 0.6345 - val_accuracy: 0.6667
Epoch 88/100
4/4 [==============================] - 0s 11ms/step - loss: 0.4128 - acc
uracy: 0.8068 - val_loss: 0.6344 - val_accuracy: 0.6667
Epoch 89/100
4/4 [==============================] - 0s 11ms/step - loss: 0.4122 - acc
uracy: 0.8068 - val_loss: 0.6341 - val_accuracy: 0.6667
Epoch 90/100
4/4 [==============================] - 0s 11ms/step - loss: 0.4116 - acc
uracy: 0.8068 - val_loss: 0.6348 - val_accuracy: 0.6667
Epoch 91/100
4/4 [==============================] - 0s 11ms/step - loss: 0.4109 - acc
uracy: 0.8068 - val_loss: 0.6348 - val_accuracy: 0.6667
Epoch 92/100
4/4 [==============================] - 0s 11ms/step - loss: 0.4103 - acc
uracy: 0.8068 - val_loss: 0.6358 - val_accuracy: 0.6667
Epoch 93/100
4/4 [==============================] - 0s 11ms/step - loss: 0.4097 - acc
uracy: 0.8068 - val_loss: 0.6357 - val_accuracy: 0.6667
Epoch 94/100
4/4 [==============================] - 0s 11ms/step - loss: 0.4090 - acc
uracy: 0.8068 - val_loss: 0.6374 - val_accuracy: 0.6667
Epoch 95/100
4/4 [==============================] - 0s 15ms/step - loss: 0.4085 - acc
uracy: 0.8094 - val_loss: 0.6385 - val_accuracy: 0.6667
Epoch 96/100
4/4 [==============================] - 0s 11ms/step - loss: 0.4079 - acc
uracy: 0.8094 - val_loss: 0.6374 - val_accuracy: 0.6667
Epoch 97/100
4/4 [==============================] - 0s 11ms/step - loss: 0.4074 - acc
uracy: 0.8094 - val_loss: 0.6382 - val_accuracy: 0.6667
Epoch 98/100
4/4 [==============================] - 0s 12ms/step - loss: 0.4067 - acc
uracy: 0.8094 - val_loss: 0.6371 - val_accuracy: 0.6667
Epoch 99/100
4/4 [==============================] - 0s 16ms/step - loss: 0.4060 - acc
uracy: 0.8120 - val_loss: 0.6376 - val_accuracy: 0.6771
Epoch 100/100
4/4 [==============================] - 0s 10ms/step - loss: 0.4055 - acc
uracy: 0.8120 - val_loss: 0.6394 - val_accuracy: 0.6771
```

Out[37]: <keras.callbacks.History at 0x7f8615ca4e20>

In [38]:
```
y_pred = ann.predict(x_train)
y_pred = y_pred > 0.5
print('training accuracy of ANN : ',accuracy_score(y_pred,y_train))

y_pred = ann.predict(x_test)
y_pred = y_pred > 0.5

print('testing accuracy of ANN : ',accuracy_score(y_pred, y_test))
```

```
15/15 [==============================] - 0s 1ms/step
training accuracy of ANN :  0.7828810020876826
8/8 [==============================] - 0s 2ms/step
testing accuracy of ANN :  0.7848101265822784
```

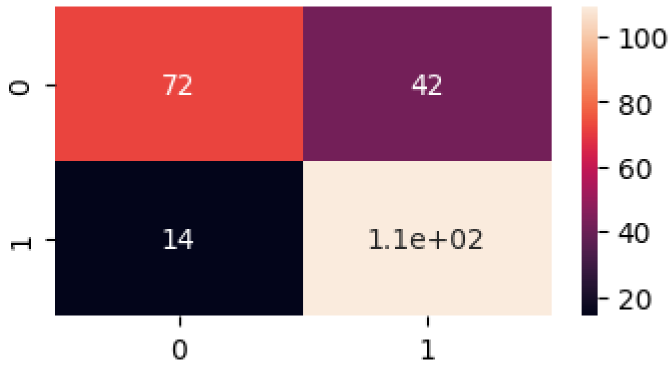
HYPER PARAMETER TUNING:-

In [39]:
```
from sklearn.model_selection import GridSearchCV
params = {
    'criterion' :['gini','entropy'],
    'max_depth' :[None,5,10,15],
    'min_samples_split':[2,5,10],
    'min_samples_leaf':[1,2,4]
}
gcv = GridSearchCV(DecisionTreeClassifier(),params,cv=5)
gcv.fit(x_train,y_train)
gcv.best_params_
```

Out[39]:
```
{'criterion': 'entropy',
 'max_depth': 5,
 'min_samples_leaf': 4,
 'min_samples_split': 10}
```

In [51]:
```
dtc2 = DecisionTreeClassifier(criterion='entropy',max_depth=5,min_samples
fit_model(dtc2,'DTC after tuning')
```

```
training accuracy of  DTC after tuning  :  0.8121085594989561
testing accuracy of  DTC after tuning  :  0.7637130801687764
```



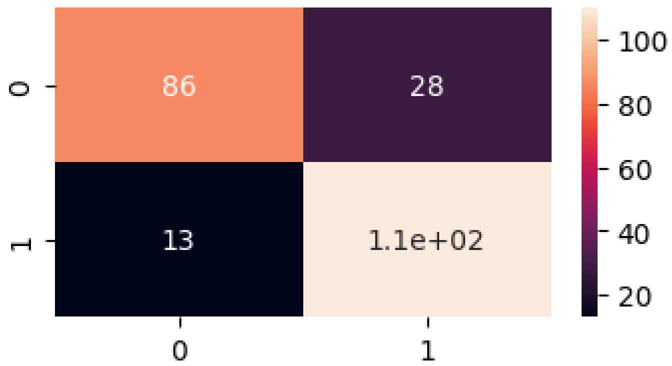

```
              precision    recall  f1-score   support

           0       0.84      0.63      0.72       114
           1       0.72      0.89      0.80       123

    accuracy                           0.76       237
   macro avg       0.78      0.76      0.76       237
weighted avg       0.78      0.76      0.76       237
```

In [41]:
```
parameters = {
                'n_estimators' :[1,20,30,55,68,74,90,120,115],
                 'criterion' :['gini','entropy'],
                  'max_features' : [ "sqrt" , "log2"],
              'max_depth' :[2,5,8,10]
}
rcv = RandomizedSearchCV(estimator=RandomForestClassifier(),param_distrib
rcv.fit(x_train,y_train)
rcv.best_params_
```

Out[41]:
```
{'n_estimators': 90,
 'max_features': 'log2',
 'max_depth': 8,
 'criterion': 'gini'}
```

In [50]:
```
rfc2 = RandomForestClassifier(n_estimators=90,max_features='log2',max_dep
fit_model(rfc2,'RFC ( after tuning ) ')
```

```
training accuracy of  RFC ( after tuning )   :  0.9373695198329853
testing accuracy of  RFC ( after tuning )   :  0.8270042194092827
```



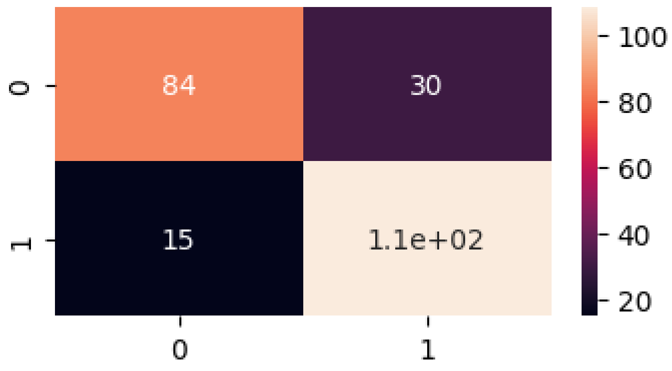

```
              precision    recall  f1-score   support

           0       0.87      0.75      0.81       114
           1       0.80      0.89      0.84       123

    accuracy                           0.83       237
   macro avg       0.83      0.82      0.83       237
weighted avg       0.83      0.83      0.83       237
```

In [43]:
```
param_grid = {
    'n_neighbors': [3, 5, 7],
    'weights': ['uniform', 'distance'],
    'p': [1, 2]
}

g2 = GridSearchCV(estimator=KNeighborsClassifier(), param_grid=param_grid

g2.fit(x_train, y_train)

print( g2.best_params_)
```

```
{'n_neighbors': 5, 'p': 1, 'weights': 'distance'}
```

In [49]:
```
knn2 = KNeighborsClassifier(n_neighbors=5,p=1,weights='distance')
fit_model(knn2,'KNN (after tuning)')
```

```
training accuracy of  KNN (after tuning)  :  1.0
testing accuracy of  KNN (after tuning)  :  0.810126582278481
```



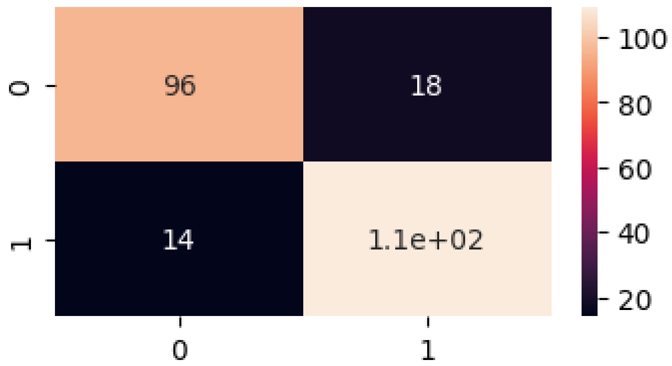

```
              precision    recall  f1-score   support

           0       0.85      0.74      0.79       114
           1       0.78      0.88      0.83       123

    accuracy                           0.81       237
   macro avg       0.82      0.81      0.81       237
weighted avg       0.81      0.81      0.81       237
```

In [45]:
```
param_grid = {
    'learning_rate': [0.1, 0.2, 0.3],
    'max_depth': [3, 4, 5],
    'n_estimators': [100, 200, 300]
}
g = GridSearchCV(estimator=XGBClassifier(), param_grid=param_grid, cv=5,

g.fit(x_train, y_train)

g.best_params_
```

Out[45]:
```
{'learning_rate': 0.2, 'max_depth': 5, 'n_estimators': 100}
```

In [48]:
```
xgb2 = XGBClassifier(learning_rate = 0.2,max_depth=5,n_estimators = 100)
fit_model(xgb2,'XGB (after tuning)')
```

```
training accuracy of  XGB (after tuning)  :  1.0
testing accuracy of  XGB (after tuning)  :  0.8649789029535865
```



```
              precision    recall  f1-score   support

           0       0.87      0.84      0.86       114
           1       0.86      0.89      0.87       123

    accuracy                           0.86       237
   macro avg       0.87      0.86      0.86       237
weighted avg       0.87      0.86      0.86       237
```

SAVING THE MODEL :-

In [47]:
```
pickle.dump(xgb2,open('xgb.pkl','wb'))
pickle.dump(scaler,open('scaler.pkl','wb'))
```