

ARIGNAR ANNA GOVERNMENT ARTS COLLEGE

VILLUPURAM --- 605 602



DEPARTMENT OF COMPUTER SCIENCE

MACHINE LEARNING WITH PYTHON

Project Title : Predicting Personal Loan Approval Using Machine Learning

Team Id : NM2023TMID16956

Team Leader : SARANRAJ E

Team member : VIMALRAJ B

Team member : RAJKUMAR G

Team member : RITHESHWAR G

INTRODUCTION

Now-a-days obtaining loans from banks have become a very common phenomenon. The banks gain profits from the loans lent to their customers in the form of interest. While approving a loan, the banks should consider many factors such as credit history and score, reputation of the person, the location of the property and the

relationship with the bank.

Many people apply for loans in the name of home loan, car loan and many more. Everyone cannot be approved based on above mentioned conditions. There are so many cases where applicants' applications for loans are not approved by various finance companies. The right predictions whether to give a loan to a customer or not is very important for the banks to maximize the profits. The idea behind this project is to use Machine Learning to predict whether a customer can get a loan from a bank or not.

1.1 Overview:

A loan is a sum of money that is borrowed and repaid over a period of time, typically with interest. There are various types of loans available to individuals and businesses, such as personal loans, mortgages, auto loans, student loans, business loans and many more. They are offered by banks, credit unions, and other financial institutions, and the terms of the loan, such as interest rate, repayment period, and fees, vary depending on the lender and the type of loan.

A personal loan is a type of unsecured loan that can be used for a variety of expenses such as home repairs, medical expenses, debt

consolidation, and more. The loan amount, interest rate, and repayment

period vary depending on the lender and the borrower's credit worthiness. To qualify for a personal loan, borrowers typically need to provide proof of income and have a good credit score.

Predicting personal loan approval using machine learning analyses a borrower's financial data and credit history to determine the likelihood of loan approval. This can help financial institutions to make more informed decisions about which loan applications to approve and which to deny.

1.2 Purpose:

Predicting personal loan approval using machine learning analyses a borrower's financial data and credit history to determine the likelihood of loan approval. This can help financial institutions to make more informed decisions about which loan applications to approve and which to deny.

However, the benefits can only be reaped if the bank has a robust model to accurately predict which customer's loan it should approve and which to reject, in order to minimize the risk of loan default.

1.3. PREDICTIVE MODELLING

Predictive modeling is used to analyze the data and predict the outcome. Predictive modeling used to predict the unknown event which may occur in the future. In this process, we are going to create, test and validate the model.

There are different methods in predictive modeling. They are learning, artificial intelligence and statistics. Once we create a model, we can use many times, to determine the probability of outcomes.

So, predict model is reusable. Historical data is used to train an algorithm. The predictive modeling process is an iterative process and often involves training the model, using multiple models on the same dataset.

- Creating the model: To create a model to run one or more algorithms on the data set.
- Testing a model: The testing is done on past data to see how the best model predicts
- Validating a model: Using visualization tools to validate the model.

- Evaluating model: Evaluating the best fit model from the models used and choosing the model right fitted for the data.

1.4.PROCESS MODE USED:

Loan Dataset

Data cleaning and pre processing

Determine the training and testing Data

Applying the model for prediction

Determine accuracy using confusion

matrix

When a customer demands credit from a bank, the bank should evaluate the credit demand as soon as possible to gain competitive advantage. Additionally, for each credit demand, the same process is repeated and constitutes a cost for the bank.

- Load the data.
- Determine the training and testing data.
- Data cleaning and preprocessing.
- Fill the missing values with mean values regarding numerical values.
- Fill the missing values with mode values regarding categorical

variables.

- Outlier treatment.
- Apply the modeling for prediction.
- Removing the load identifier.
- Create the target variable (based on the requirement). In this approach,
- Target variable is loan-status.
- Create a dummy variable for categorical variable (if required) and split the
- Training and testing data for validation.
- Apply the model
- LR method
- RF method
- SVM method
- Determine the accuracy followed by confusion matrix

1.5.Project Flow:

- User interacts with the UI to enter the input.
- Entered input is analysed by the model which is integrated.
- Once model analyses the input the prediction is showcased on

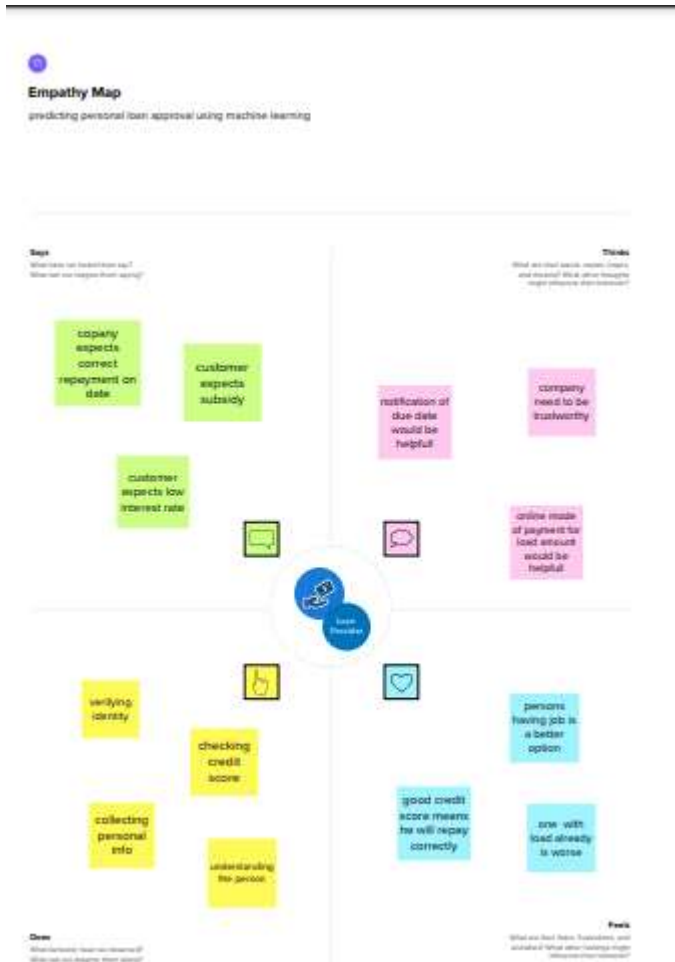
the UI

To accomplish this, we have to complete all the activities listed below,

- ✓ Define Problem / Problem Understanding
- ✓ Data Collection & Preparation
- ✓ Exploratory Data Analysis
- ✓ Model Building
- ✓ Performance Testing & Hyperparameter Tuning
- ✓ Model Deployment

2.PROBLEM DEFINITION & DESIGN THINKING

2.1 Empathy Map



In the above Mural picture Our Group done the map with our ideas about Optimizing Spam Filtering with Machine Learning. We talk about the topic from users side what they Says, Thinks, Feels and Does. We added the screenshot of our Mural map.

2.2 Ideation & Brainstroming

In the below we added Screenshot of our Brainstorm and

Ideation. In this template we give our brainstorm ideas about our project.

BRAINSTROM:



2.3.SPECIFY THE BUSINESS PROBLEM

The problem of predicting personal loan approval involves using data analysis and machine learning algorithms to predict whether a borrower will be approved for a personal loan or not. This problem is important for both lenders and borrowers. Lenders need to accurately assess the risk of lending money to borrowers, while borrowers need to understand their chances of getting approved and prepare accordingly.

The problem of predicting personal loan approval involves analyzing various factors such as the borrower's credit history, income, employment status, debt-to-income ratio, loan amount, and loan purpose, among others. Machine learning algorithms such

as logistic regression, decision trees, and neural networks can be used to analyze these factors and predict whether a borrower will be approved or not.

The accuracy of the prediction model depends on the quality and quantity of data used for training and testing the model. Therefore, it is important to have a large and diverse dataset to ensure that the model is robust and can generalize well to new data. Additionally, feature engineering, data preprocessing, and model selection are important steps in building an accurate prediction model.

Overall, the problem of predicting personal loan approval is an important and challenging task that requires careful analysis and modeling to ensure accurate predictions.

Business requirements

Personal loan approval prediction using machine learning can be a useful tool for businesses in the lending industry. By leveraging historical data and statistical algorithms, machine learning models can identify patterns and make accurate predictions about the likelihood of a loan being approved.

The business environment for predicting personal loan approval involves utilizing machine learning and data analysis techniques to assess the credit worthiness of loan applicants.

The goal is to accurately predict the likelihood of an applicant's loan being approved based on factors such as their credit score, income, employment history, and debt-to-income ratio. This technology can benefit financial institutions, such as banks and credit unions, by streamlining the loan approval process and reducing the risk of default on loans. By accurately assessing an applicant's creditworthiness, financial institutions can make more informed decisions about whether to approve a loan, how much to lend, and at what interest rate. Additionally, by leveraging data analytics and machine learning algorithms, financial institutions can improve their customer service by providing a faster and more convenient loan application process.

This can help attract more customers and increase customer satisfaction. Overall, the business environment for predicting personal loan approval involves using cutting-edge technology to improve the accuracy, speed, and convenience of the loan approval process, benefiting both financial institutions and their customers.

2.4.Literature Survey

“Loan Prediction using Decision Tree and Random Forest” Author Kshitiz Gautam, Arun Pratap Singh, Keshav Tyagi, Mr. Suresh

Kumar Year-2020. In India the number of people or organization

applying for loan gets increased every year. The bank have to put in a lot of work to analyses or predict whether the customer can pay back the loan amount or not (defaulter or non-defaulter) in the given time.

The aim of this paper is to find the nature or background or credibility of client that is applying for the loan. We use exploratory data analysis technique to deal with problem of approving or rejecting the loan request or in short loan prediction.

The main focus of this paper is to determine whether the loan given to a particular person or an organization shall be approved or not.

"A Comparative Study of Machine Learning Algorithms for Personal Loan Approval Prediction" by S. Chakraborty and S.

Chakraborty (2020) - This study compares the performance of logistic regression, decision tree, random forest, and support vector machine algorithms for predicting personal loan approval. The authors found that random forest and support vector machine algorithms had the highest accuracy in predicting loan approval.

"Predicting Personal Loan Approval Using Machine Learning Techniques" by R. Gupta and S. Bansal (2019) - This study uses a dataset of personal loan applications to predict loan approval using logistic regression and decision tree algorithms. The authors found

that logistic regression had higher accuracy in predicting loan approval than decision trees.

In 2019, Jency, Sumathi and Shiva Sri [2] proposed an Exploratory Data Analysis(EDA) regarding the loan prediction procedure based on the client's nature and their requirements.

The major factors concentrated during the data analysis were annual income versus loan purpose, customer 's trust, loan tenure versus delinquent months, loan tenure versus credit category, loan tenure versus number of years in the current job, and chances for loan repayment versus the house ownership.

Finally, the outcome of the present work was to infer the constraints on the customer who are applying for the loan followed by the prediction regarding the repayment. Further, results showed that, the customers were interested more on availing short-tenure loans rather than long-tenure loans. "Exploring the Machine Learning Algorithm for Prediction the Loan Sanctioning Process"

Author- E. Chandra Bessie, R. Rekha - Year- 2019 Extending credits to corporates and individuals for the smooth functioning of growing economies like India is inevitable. As increasing number of customers apply for loans in the banks and non- banking financial companies (NBFC), it is really challenging for banks and

NBFCs with limited capital to devise a standard resolution and safe procedure to lend money to its borrowers for their financial needs. In addition, in recent times NBFC inventories have suffered a significant downfall in terms of the stock price. It has contributed to a contagion that has also spread to other financial stocks, adversely affecting the benchmark in recent times. In this paper, an attempt is made to condense the risk involved in selecting the suitable person who could repay the loan on time thereby keeping the bank's nonperforming assets (NPA) on the hold.

This is achieved by feeding the past records of the customer who acquired loans from the bank into a trained machine learning model which could yield an accurate result. The prime focus of the paper is to determine whether or not it will be safe to allocate the loan to a particular person. This paper has the following sections (i) Collection of Data, (ii) Data Cleaning and (iii) Performance Evaluation. Experimental tests found that the Naïve Bayes model has better performance Evaluation. Experimental tests found that the Naïve Bayes model has better performance than other models in terms of loan forecasting. Manjeet et al (2018) [24] there are seven types of variables that may influence consumer loan default; consumer 'annual income, debt-income ratio, occupation, home

ownership, work duration and whether or not consumer possesses saving/checking account.

“Loan Prediction using machine learning model”

Year2019whether or not it will be safe to allocate the loan to a particular person. This paper has the following sections

Collection of Data,

(ii) Data Cleaning and

(iii) Performance Evaluation.

Experimental tests found that the Naïve Bayes model has better performance than other models in terms of loan forecasting. With the enhancement in the banking sector lots of people are applying for bank loans but the bank has its limited assets which it has to grant to limited people only, so finding out to whom the loan can be granted which will be a safer option for the bank is a typical process.

So in this project we try to reduce this risk factor behind selecting the safe person so as to save lots of bank efforts and assets. This is done by mining the Big Data of the previous records of the people to whom the loan was granted before and on the basis of these records/experiences the machine was trained using the machine learning model which give the most accurate result The main

objective of this project is to predict whether assigning the loan to particular person will be safe or not.

This paper is divided into four sections

(i) Data Collection

(ii) Comparison of machine learning models on collected data

(iii) Training of system on most promising model

(iv) Testing.

In this paper we are predict the loan data by using some machine learning algorithms they are classification, logic regression, Decision Tree and gradient boosting. " Loan Approval Prediction based on Machine Learning Approach" Author- Kumar Arun, Garg Ishan, Kaur Sanmeet Year- 2018

The main objective of this paper is to predict whether assigning the loan to particular person will be safe or not. This paper is divided into four sections (i) Data Collection (ii) Comparison of machine learning models on collected data (iii) Training of system on most promising model (iv) Testing

In 2017, Goyal and Kaur [4] presented a loan prediction model using several Machine Learning (ML) algorithms. The dataset with features, namely, gender, marital status, education, number of dependents, employment status, income, co applicant's income, loan amount, loan tenure, credit

history, existing loan status, and property area, are used for determining the loan eligibility regarding the loan sanctioning process. Various ML models adopted in the present method includes, Linear model, Decision Tree (DT), Neural Network (NN), Random Forest (RF), SVM, Extreme learning machines, Model tree, Multivariate Adaptive Regression Splines, Bagged Cart Model, NB and TGA.

When evaluated these models using Environment in five runs, TGA resulted in better loan forecasting performance than the other methods. Bekhet and Eletter in 2012 attempted to develop a model with Artificial Neural Networks (ANN) as a decision support system for Jordanian commercial banks to assist in credit approval evaluation.

The system can be easily utilized by credit officers in taking excellent decisions before determining future loan applications and applicants. The results obtained with this technique showed that ANN was better in performance. The main challenge with the approach was in the small dataset used for the evaluation (Bekhet and Eletter, 2012)

2.5.Social or Business Impact

Prediction personal loan approval social or business impact As an

AI language model, don't have access to specific information about individuals or their loan applications. However, in general, whether a personal loan application is approved or not can have both social and business impacts.

From a social perspective, being approved for a personal loan can provide financial relief and help individuals meet their immediate needs, such as paying for medical expenses, funding a home renovation project, or consolidating high-interest debt. This can lead to a better quality of life and reduce financial stress.

On the other hand, being denied a personal loan can have negative social impacts, such as increased financial strain and potentially damaging credit scores. This can make it harder to secure future loans, rent an apartment, or even find a job. From a business perspective, loan approval can also have significant impacts.

For lenders, loan approval decisions are based on factors such as credit score, income, and employment history. These factors can be used to assess the risk of loan repayment and determine whether a loan application should be approved or denied. Approving loans that are unlikely to be repaid can lead to financial losses for the lender and potentially impact their ability to offer loans to other borrowers in the future.

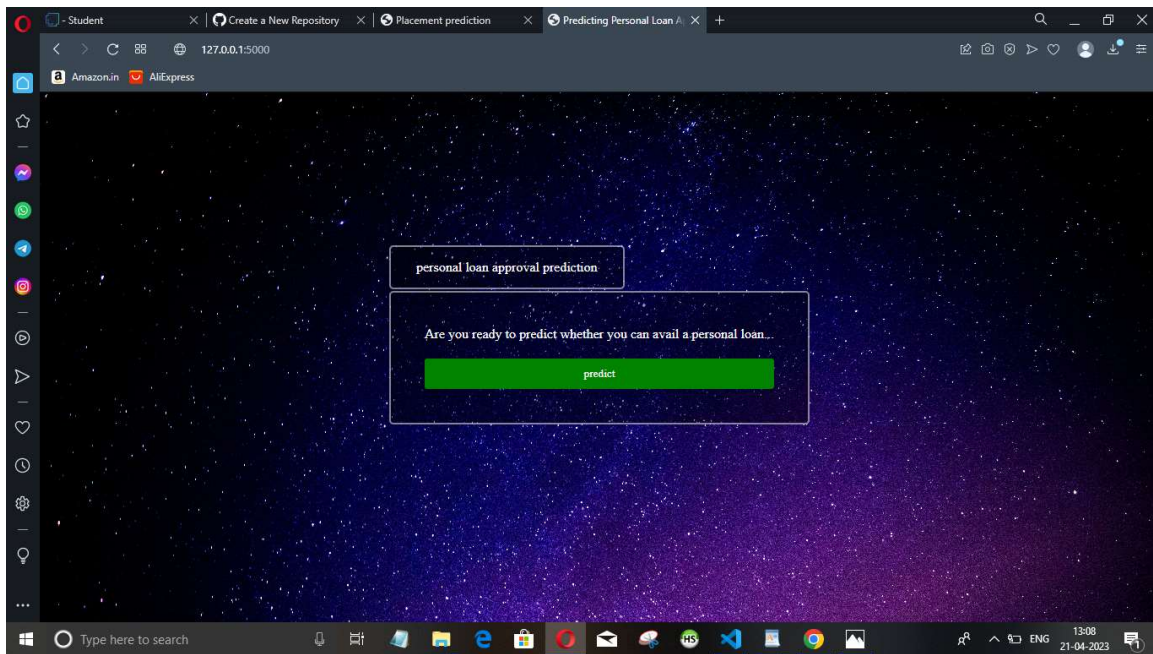
Therefore, lenders often have strict criteria for loan approval and take into account a wide range of factors before making a decision.

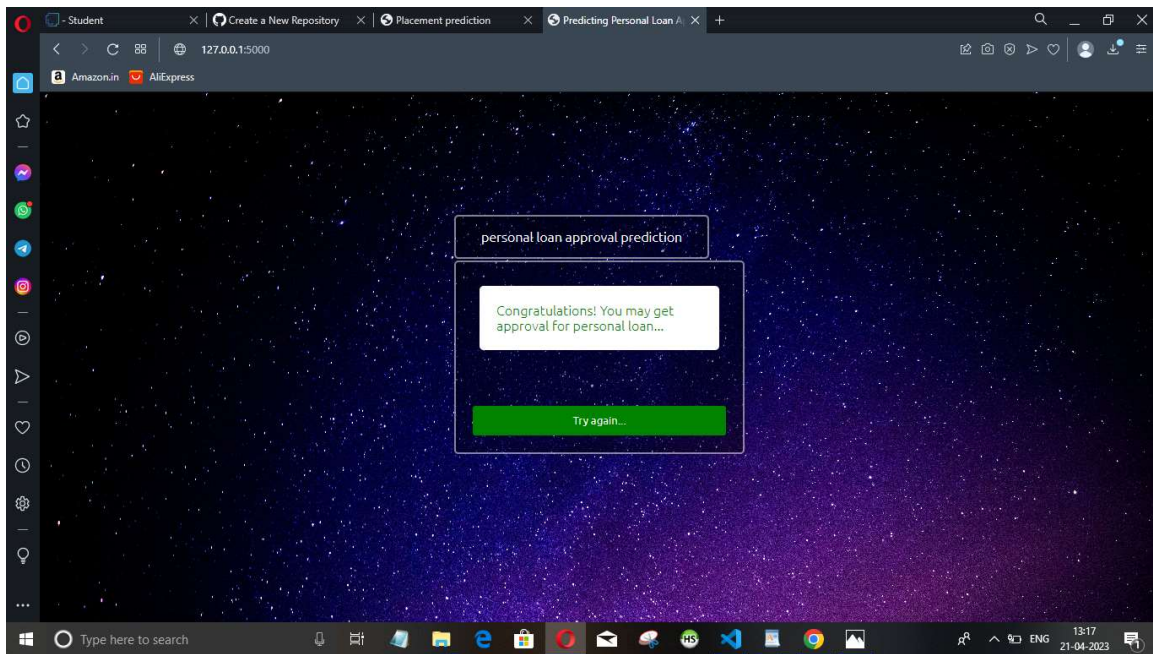
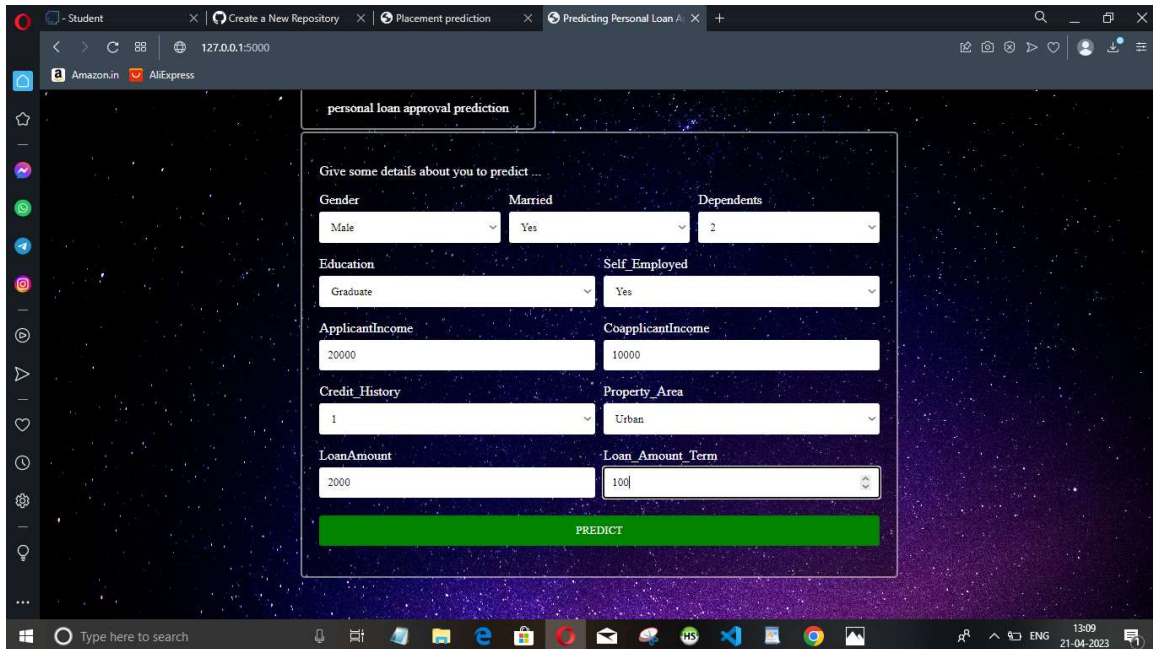
In conclusion, personal loan approval or denial can have both social and business impacts, depending on the individual's circumstances and the lender's criteria.

It's important to carefully consider your financial situation and your ability to repay a loan before applying and to choose a reputable lender with fair loan terms.

3.RESULT

Web page of Loan Approval:





4.ADVANTAGES

- ✓ The nonfunctional requirements ensure the software system follow legal and compliance rules.
- ✓ They ensure the reliability, availability, and performance of

the software system

✓ They ensure good user experience and ease of operating the software.

✓ They help in formulating security policy of the software system.

4.1 DISADVANTAGES

✓ None functional requirement may affect the various highlevel software subsystem

✓ Technically you must be have a hardware requirements and that's ram almost 8 GB RAM.

5.APPLICATIONS

Here we will be using a declared constructor to route to the HTML page which we have createdearlier.

In,

'/' URL is bound with the home.html function. Hence, when the home pageof the web server is opened in the browser, the html page will be rendered. Whenever you enterthe values from the html page the values can be retrieved using POST Method.

Retrieves the value from UI:

Here we are routing our app to predict() function. This function retrieves all the values from the HTML page using Post request.

That is stored in an array. This array is passed to the model.predict() function. This function returns the prediction. And this prediction value will berendered to the text that we have mentioned in the submit.html page earlier.

6.CONCLUSION

Random Forest Classifier is giving the best accuracy with an

accuracy score of 82% for the testing dataset. And to get much better results ensemble Learning techniques like Bagging and Boosting can also be used.

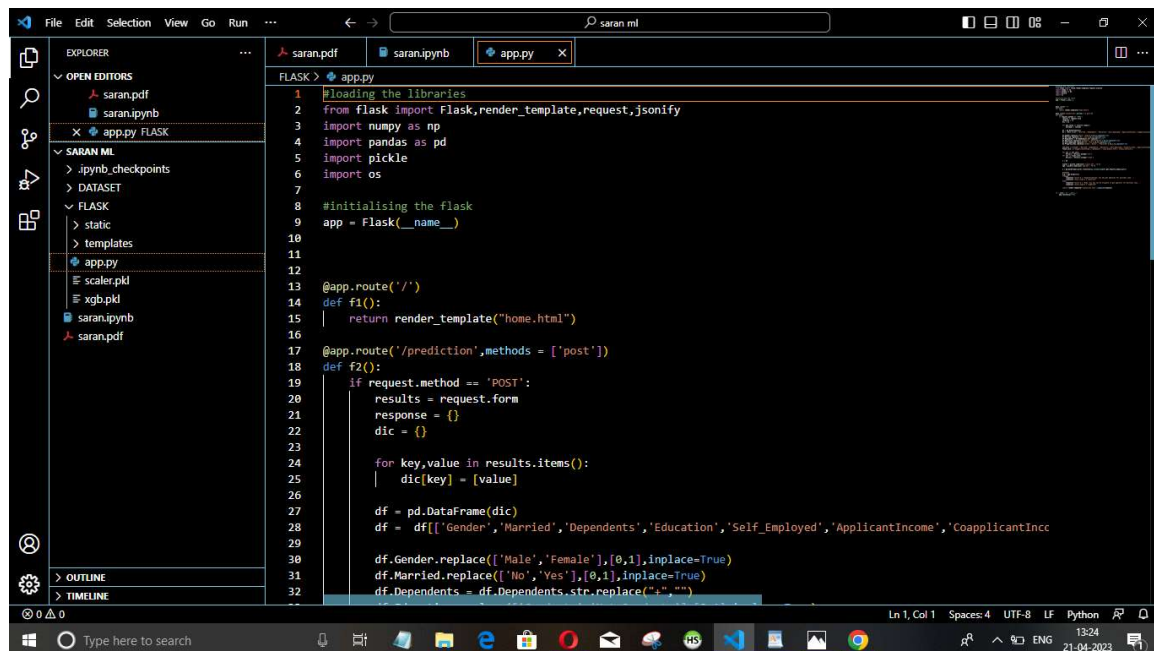
7.FUTURE SCOPE

Predicting personal loan approval using machine learning analyses a borrower's financial data and credit history to determine the likelihood of loan approval. This can help financial institutions to make more informed decisions about which loan applications to approve and which to deny.

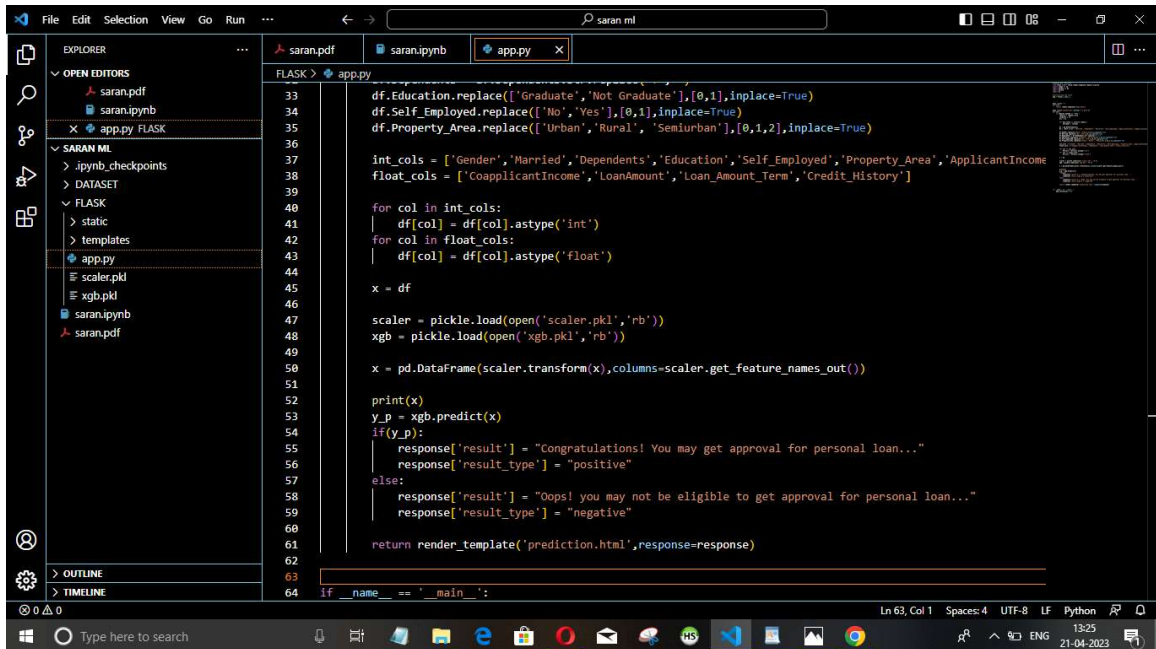
However, the future scope is benefits can only be reaped if the bank has a robust model to accurately predict which customer's loan it should approve and which to reject, in order to minimize the risk of loan default.

8.APPENDIX:

10. App.py

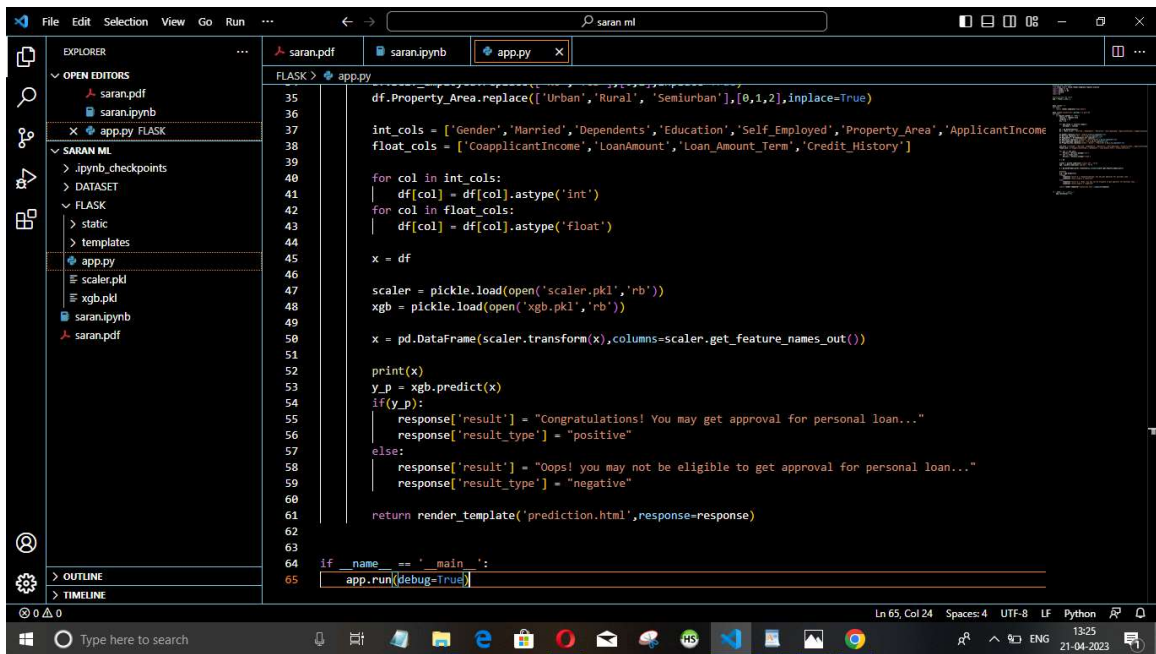


```
1 #loading the libraries
2 from flask import Flask,render_template,request,jsonify
3 import numpy as np
4 import pandas as pd
5 import pickle
6 import os
7
8 #initialising the flask
9 app = Flask(__name__)
10
11
12 @app.route('/')
13 def f1():
14     return render_template("home.html")
15
16
17 @app.route('/prediction',methods = ['post'])
18 def f2():
19     if request.method == 'POST':
20         results = request.form
21         response = {}
22         dic = {}
23
24         for key,value in results.items():
25             dic[key] = [value]
26
27         df = pd.DataFrame(dic)
28         df = df[['Gender','Married','Dependents','Education','Self_Employed','ApplicantIncome','CoapplicantIncc
29
30         df.Gender.replace(['Male','Female'],[0,1],inplace=True)
31         df.Married.replace(['No','Yes'],[0,1],inplace=True)
32         df.Dependents = df.Dependents.str.replace("+","")
```

```
File Edit Selection View Go Run ...
saran ml
EXPLORER
  OPEN EDITORS
    saran.pdf
    saran.ipynb
    app.py FLASK
  SARAN ML
    .ipynb_checkpoints
    DATASET
    FLASK
      static
      templates
      app.py
      scaler.pkl
      xgb.pkl
      saran.ipynb
      saran.pdf
    OUTLINE
    TIMELINE
Type here to search

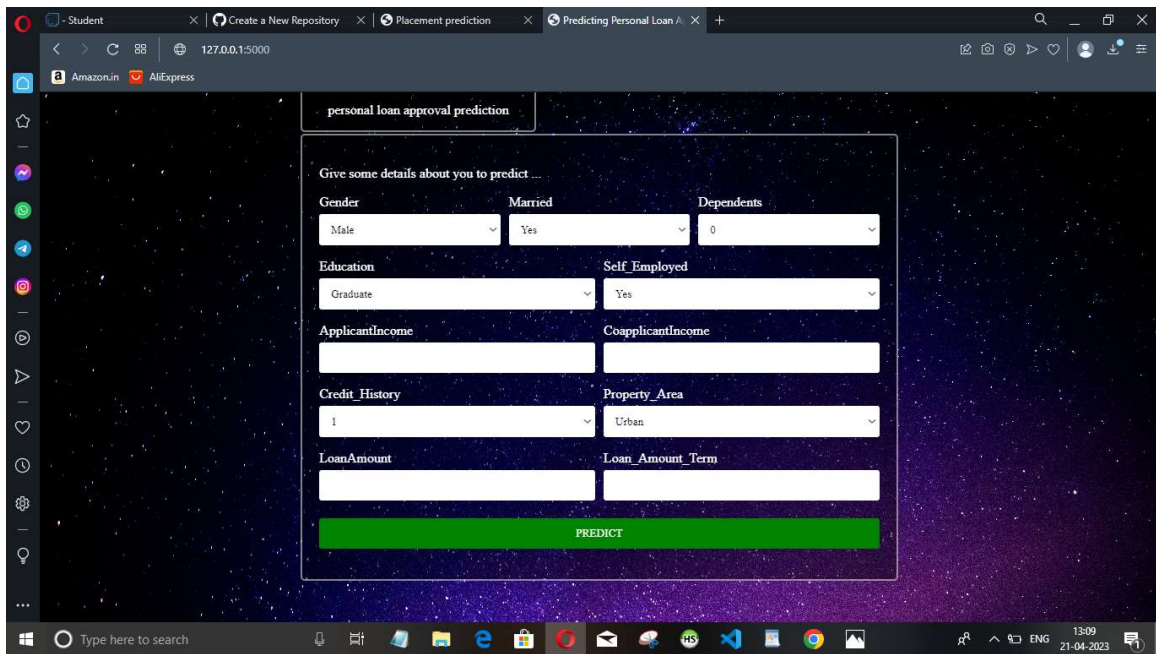
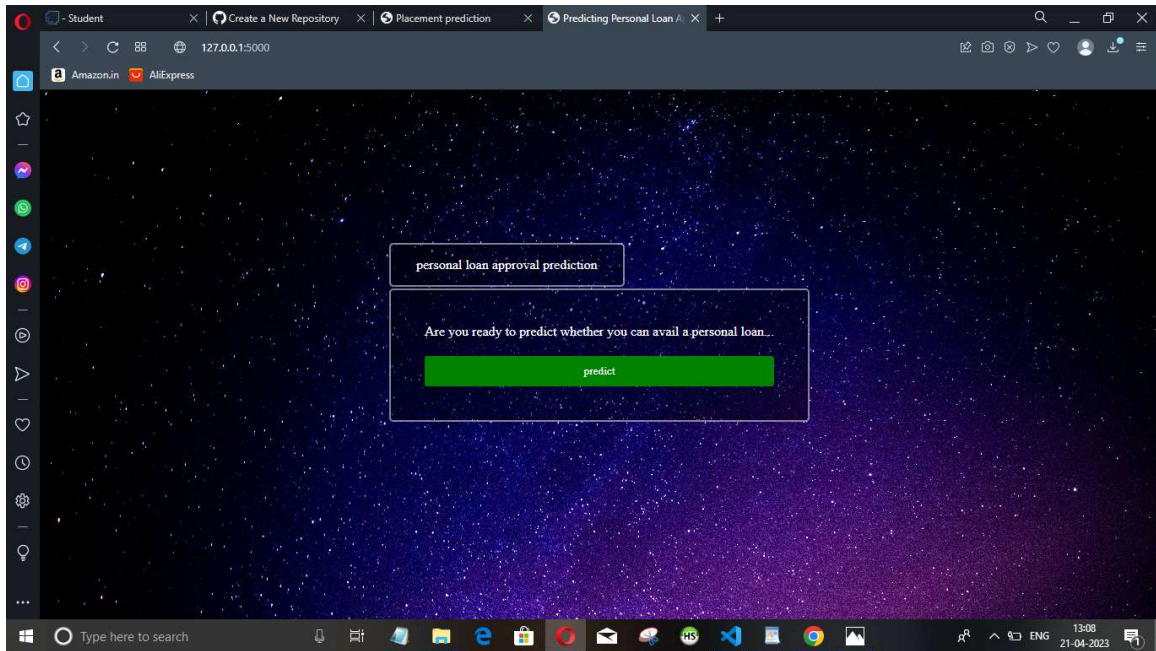
FLASK > app.py
33 df.Education.replace(['Graduate','Not Graduate'],[0,1],inplace=True)
34 df.Self_Employed.replace(['No','Yes'],[0,1],inplace=True)
35 df.Property_Area.replace(['Urban','Rural','Semiurban'],[0,1,2],inplace=True)
36
37 int_cols = ['Gender','Married','Dependents','Education','Self_Employed','Property_Area','ApplicantIncome']
38 float_cols = ['CoapplicantIncome','LoanAmount','Loan_Amount_Term','Credit_History']
39
40 for col in int_cols:
41     df[col] = df[col].astype('int')
42 for col in float_cols:
43     df[col] = df[col].astype('float')
44
45 x = df
46
47 scaler = pickle.load(open('scaler.pkl','rb'))
48 xgb = pickle.load(open('xgb.pkl','rb'))
49
50 x = pd.DataFrame(scaler.transform(x),columns=scaler.get_feature_names_out())
51
52 print(x)
53 y_p = xgb.predict(x)
54 if(y_p):
55     response['result'] = "Congratulations! You may get approval for personal loan..."
56     response['result_type'] = "positive"
57 else:
58     response['result'] = "Oops! you may not be eligible to get approval for personal loan..."
59     response['result_type'] = "negative"
60
61 return render_template('prediction.html',response=response)
62
63
64 if __name__ == '__main__':
```

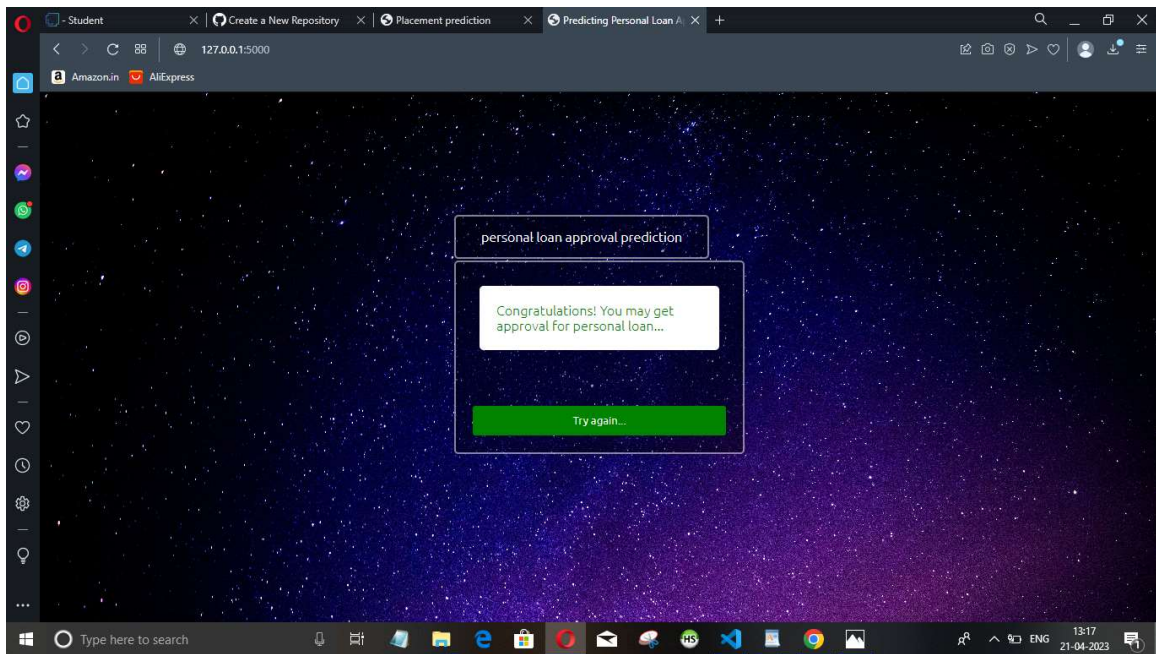
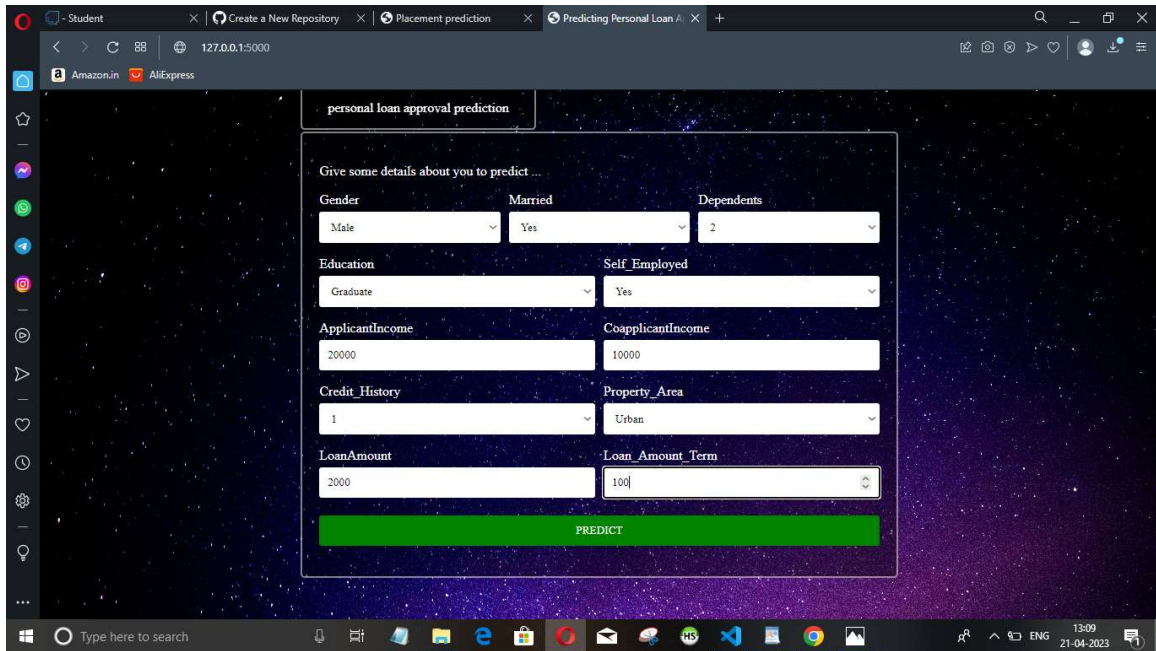


```
File Edit Selection View Go Run ...
saran ml
EXPLORER
  OPEN EDITORS
    saran.pdf
    saran.ipynb
    app.py FLASK
  SARAN ML
    .ipynb_checkpoints
    DATASET
    FLASK
      static
      templates
      app.py
      scaler.pkl
      xgb.pkl
      saran.ipynb
      saran.pdf
    OUTLINE
    TIMELINE
Type here to search

FLASK > app.py
35 df.Property_Area.replace(['Urban','Rural','Semiurban'],[0,1,2],inplace=True)
36
37 int_cols = ['Gender','Married','Dependents','Education','Self_Employed','Property_Area','ApplicantIncome']
38 float_cols = ['CoapplicantIncome','LoanAmount','Loan_Amount_Term','Credit_History']
39
40 for col in int_cols:
41     df[col] = df[col].astype('int')
42 for col in float_cols:
43     df[col] = df[col].astype('float')
44
45 x = df
46
47 scaler = pickle.load(open('scaler.pkl','rb'))
48 xgb = pickle.load(open('xgb.pkl','rb'))
49
50 x = pd.DataFrame(scaler.transform(x),columns=scaler.get_feature_names_out())
51
52 print(x)
53 y_p = xgb.predict(x)
54 if(y_p):
55     response['result'] = "Congratulations! You may get approval for personal loan..."
56     response['result_type'] = "positive"
57 else:
58     response['result'] = "Oops! you may not be eligible to get approval for personal loan..."
59     response['result_type'] = "negative"
60
61 return render_template('prediction.html',response=response)
62
63
64 if __name__ == '__main__':
65     app.run(debug=True)
```

11. Output of Application:





```
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, RandomForestClassifier
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, confusion_matrix
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	CoapplicantIncome
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=True)
```

```
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()
```

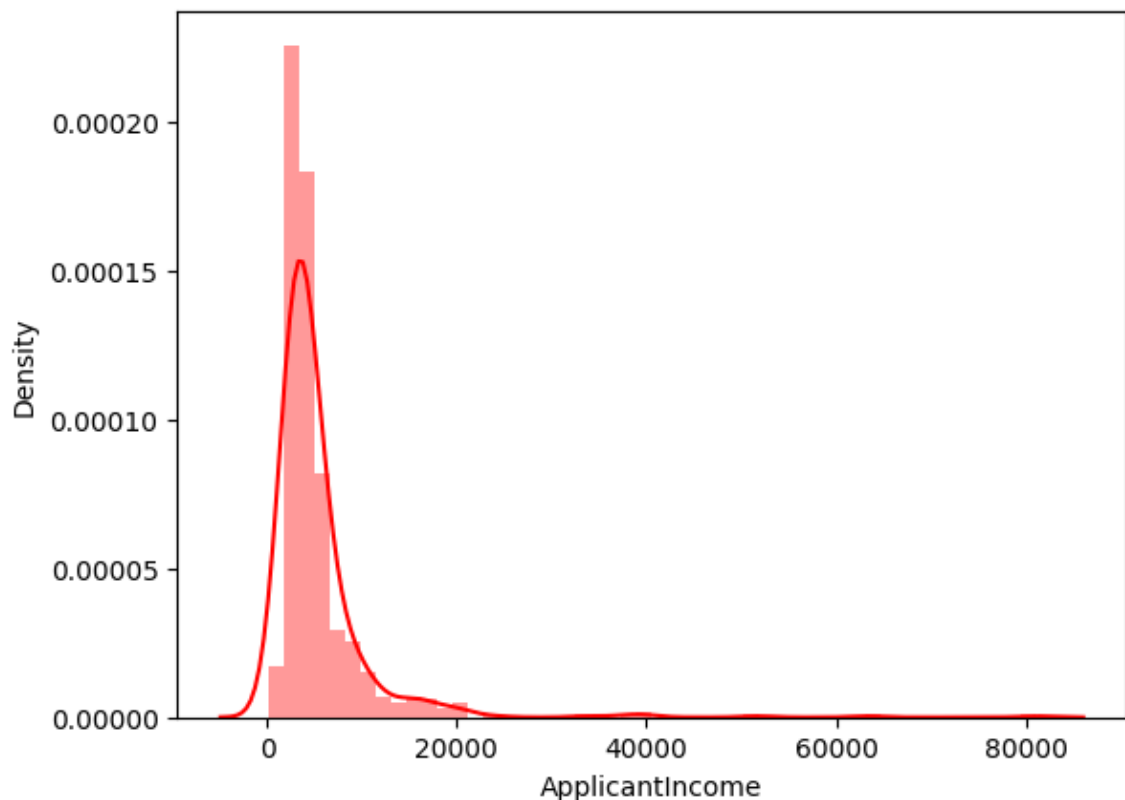
```
Out[23]:
```

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Credit_History
count	614.000000	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	0.960912
std	0.386497	0.476373	1.009623	0.413389	0.340446	6109.041673	0.200000
min	0.000000	0.000000	0.000000	0.000000	0.000000	150.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000	2877.500000	0.000000
50%	0.000000	1.000000	0.000000	0.000000	0.000000	3812.500000	0.000000
75%	0.000000	1.000000	1.000000	0.000000	0.000000	5795.000000	0.000000
max	1.000000	1.000000	3.000000	1.000000	1.000000	81000.000000	1.000000

UNIVARIATE ANALYSIS :-

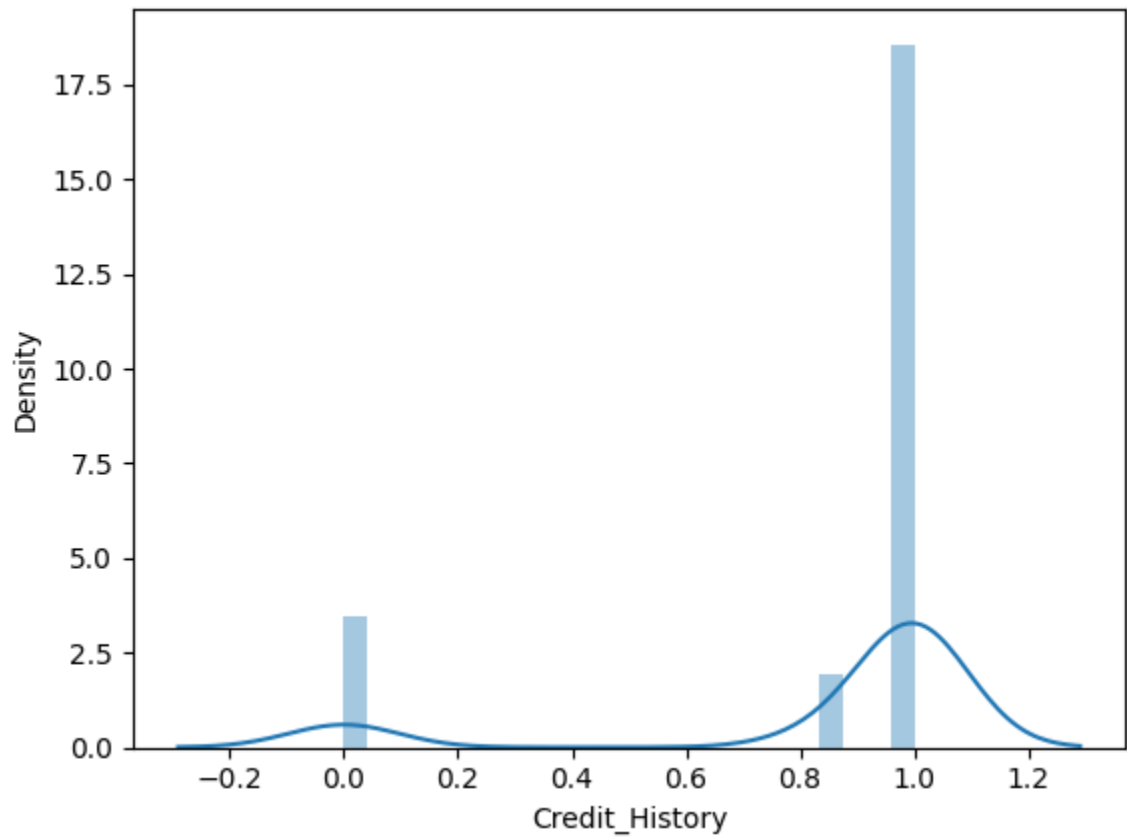
```
In [24]: sns.distplot(df['ApplicantIncome'], color='r')
```

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

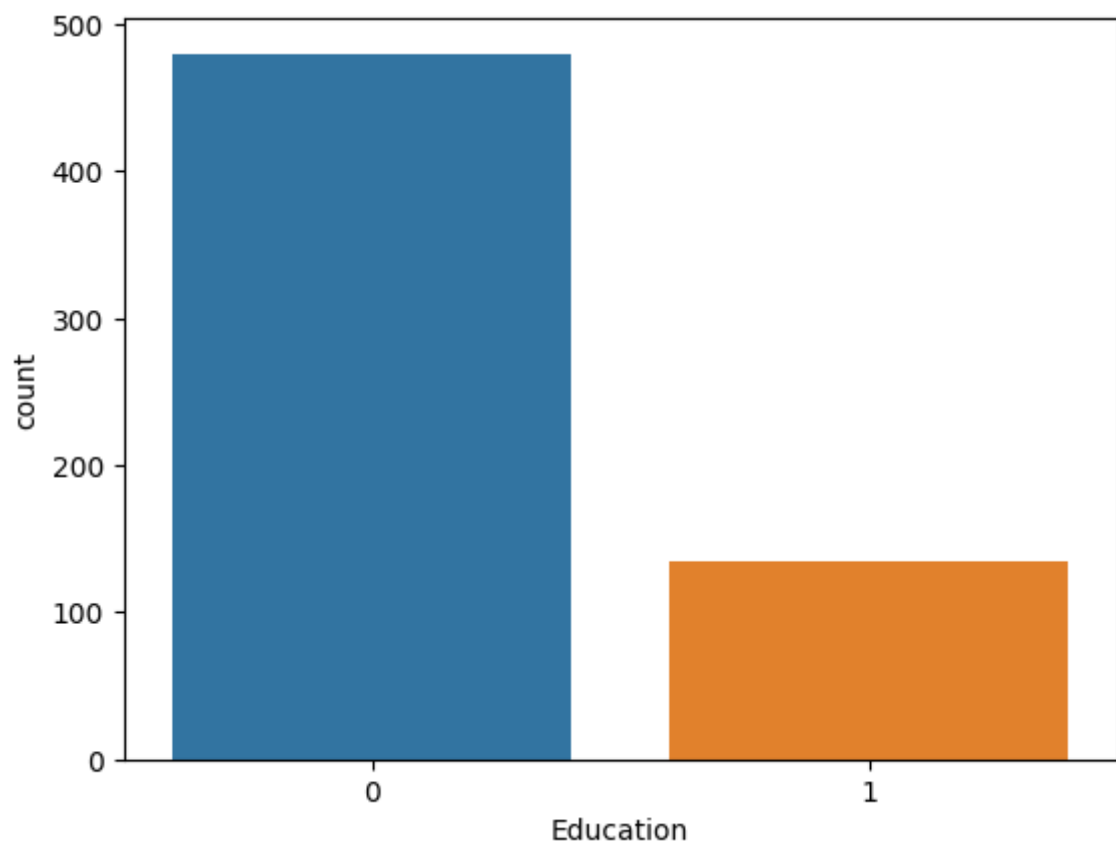
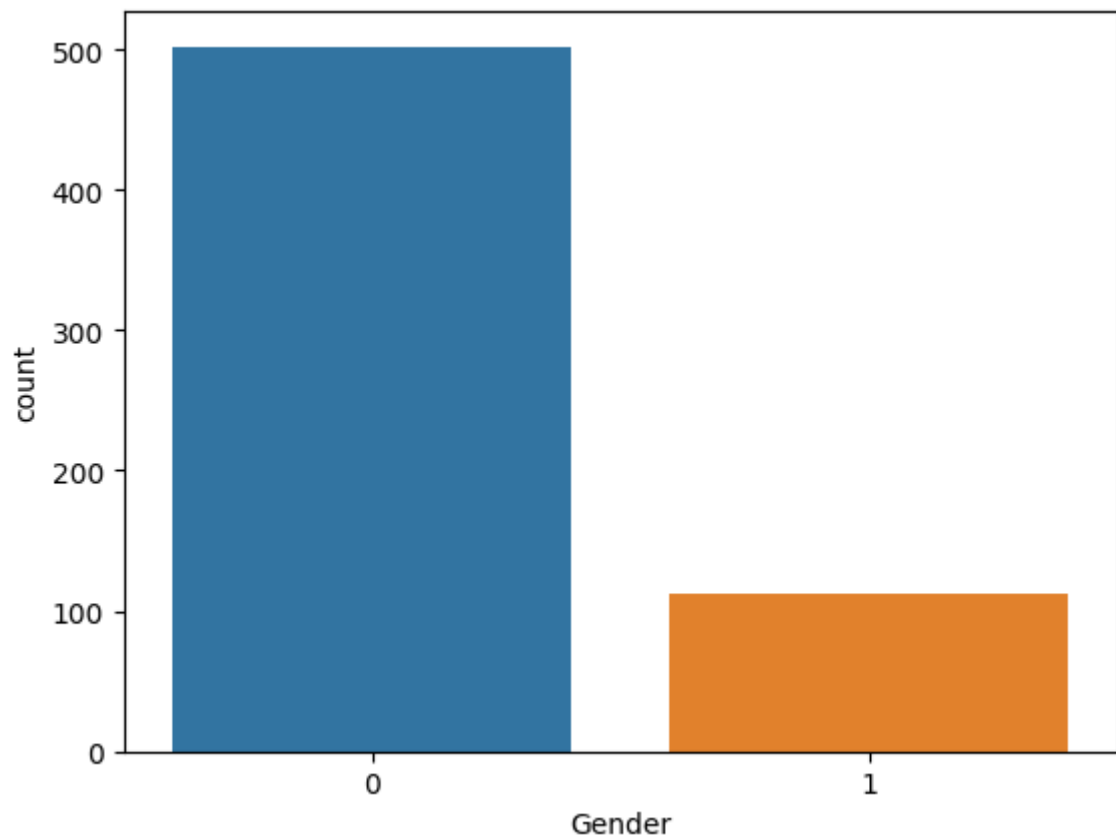


```
In [25]: sns.distplot(df['Credit_History'])
```

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

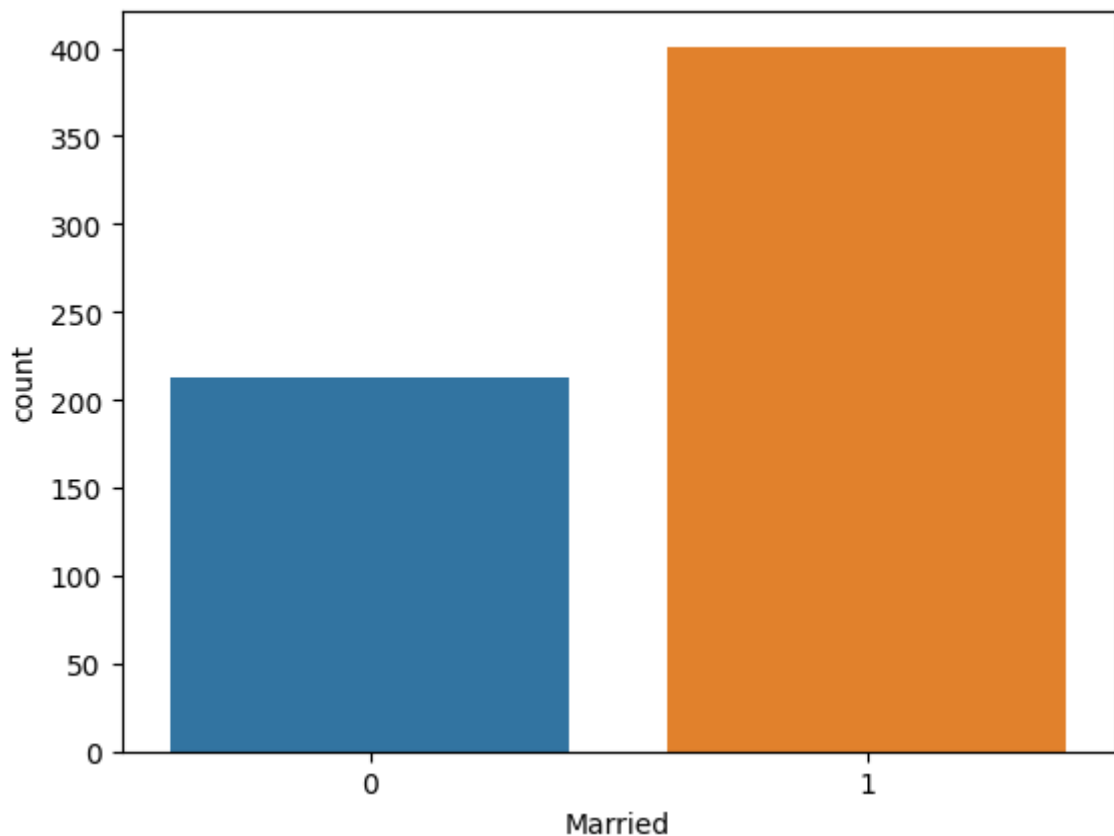


```
In [26]: sns.countplot(df,x='Gender')
plt.xlabel('Gender')
plt.show()
sns.countplot(df,x='Education')
plt.xlabel('Education')
plt.show()
```



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

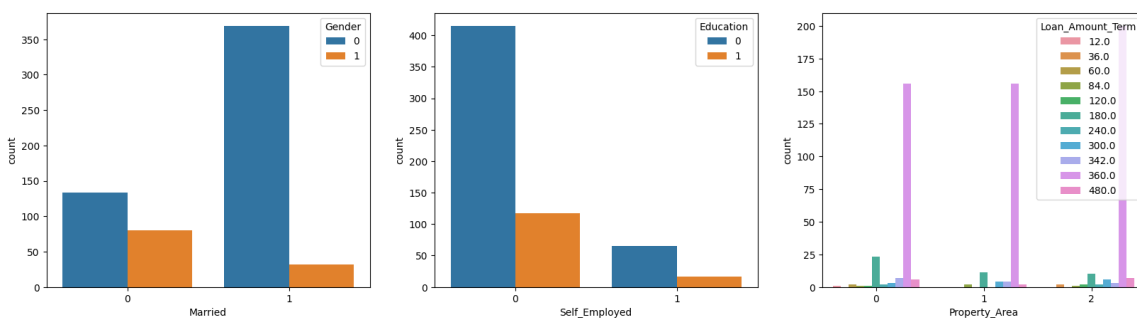
```
Out[27]: <Axes: xlabel='Married', ylabel='count'>
```

BIVARIATE ANALYSIS :-

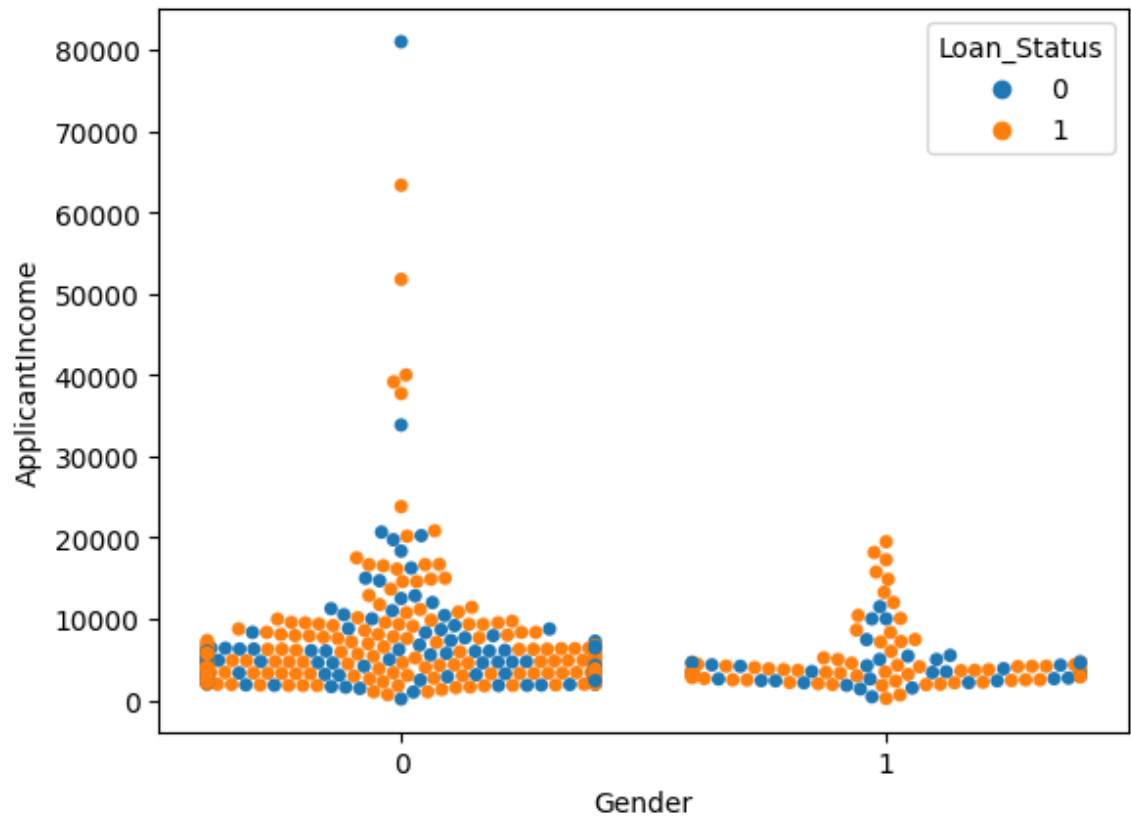
```
In [28]: plt.figure(figsize=(20,5))
plt.subplot(1,3,1)
sns.countplot(x= 'Married', hue = "Gender", data = df)
plt.subplot(1,3,2)
sns.countplot(x = 'Self_Employed', hue = "Education", data = df)
plt.subplot(1,3,3)
sns.countplot(x= 'Property_Area', hue = "Loan_Amount_Term", data =df)
```

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	CoapplicantIncome
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 TESTING 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))

```

```

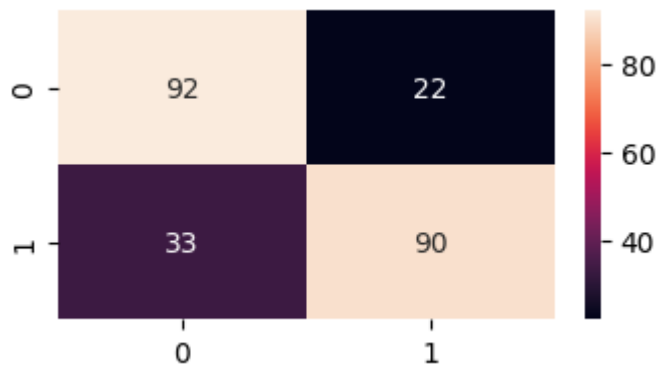
In [33]: dtc = DecisionTreeClassifier()
fit_model(dtc,"DTC")

```

```

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

```

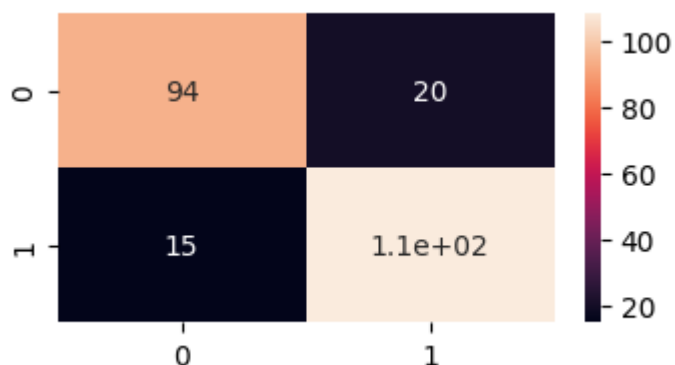
In [34]: rfc = RandomForestClassifier()
fit_model(rfc,"RFC")

```

```

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

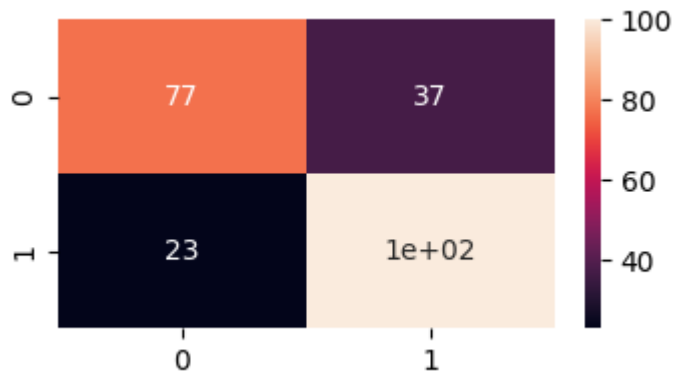
```



	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]: knn = KNeighborsClassifier()
fit_model(knn, 'KNN')
```

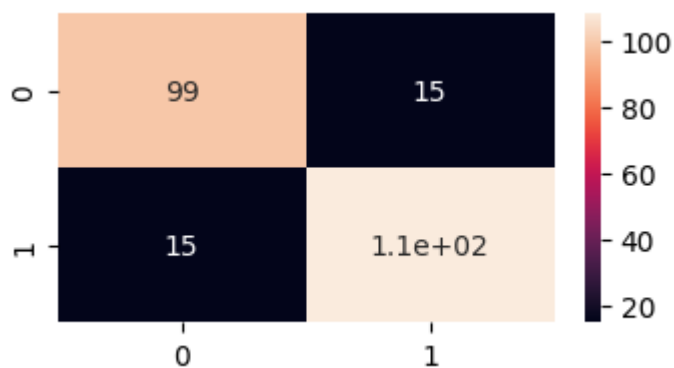
training accuracy of KNN : 0.824634655532359
testing accuracy of KNN : 0.7468354430379747



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

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

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

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

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

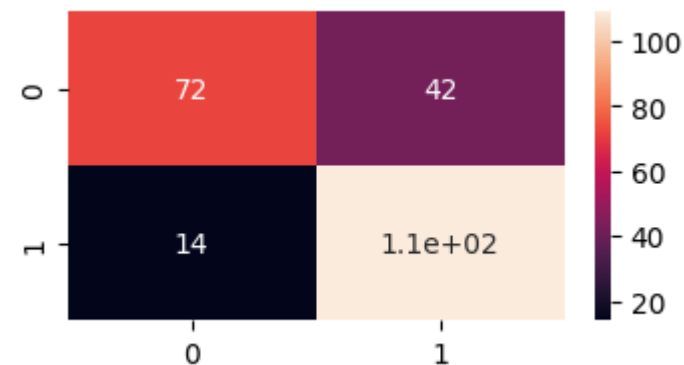
HYPER PARAMETER TUNING:-

```
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



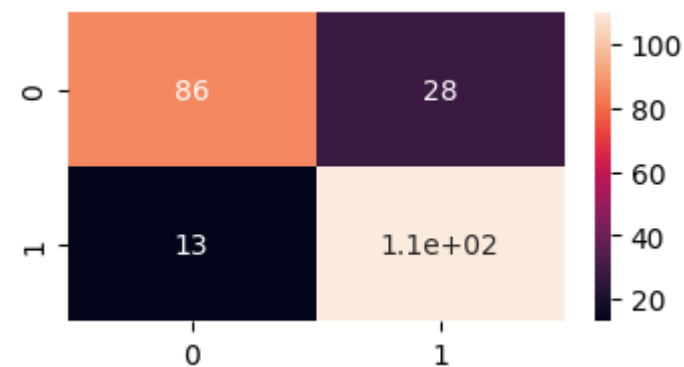
	precision	recall	f1-score	support
0	0.84	0.63	0.72	114
1	0.72	0.89	0.80	123
accuracy			0.76	237
macro avg	0.78	0.76	0.76	237
weighted avg	0.78	0.76	0.76	237

```
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]
    }

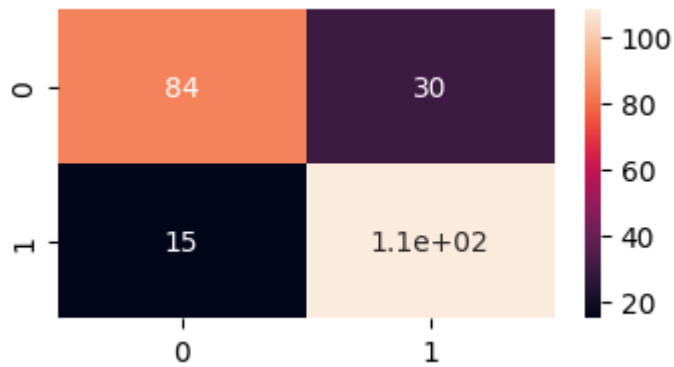
    g2 = GridSearchCV(estimator=KNeighborsClassifier(), param_grid=param_grid
    g2.fit(x_train, y_train)

    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

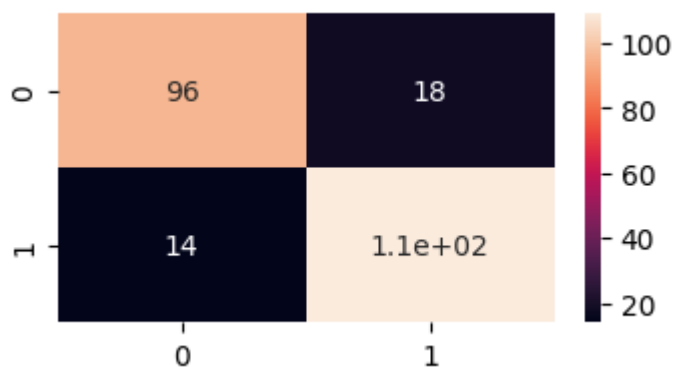
```
In [45]: param_grid = {
          'learning_rate': [0.1, 0.2, 0.3],
          'max_depth': [3, 4, 5],
          'n_estimators': [100, 200, 300]
        }
g = GridSearchCV(estimator=XGBClassifier(), param_grid=param_grid, cv=5,
g.fit(x_train, y_train)

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'))
```