Problem statement

Developing an application, which is able to detect a loan can be given to a person

In [63]:
```
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_mat
import os
```

Importing the dataset

In [64]:
```
#importing the dataset which is in csv file
data = pd.read_csv(r"./loan_prediction.csv")
data
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	C
0	LP001002	Male	No	0	Graduate	No	5849	0.
1	LP001003	Male	Yes	1	Graduate	No	4583	1
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2
4	LP001008	Male	No	0	Graduate	No	6000	0.
...	...	...	...	...	...	...	...	...
609	LP002978	Female	No	0	Graduate	No	2900	0.
610	LP002979	Male	Yes	3+	Graduate	No	4106	0.
611	LP002983	Male	Yes	1	Graduate	No	8072	2
612	LP002984	Male	Yes	2	Graduate	No	7583	0.
613	LP002990	Female	No	0	Graduate	Yes	4583	0.

614 rows × 13 columns

In [65]:
```
#dropping the loan id colunms beacuse there is no use it for the model building
data.drop(['Loan_ID'],axis=1,inplace=True)
```

In [66]:
```
data
```

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapplicant
0	Male	No	0	Graduate	No	5849	0.0
1	Male	Yes	1	Graduate	No	4583	1508.0
2	Male	Yes	0	Graduate	Yes	3000	0.0
3	Male	Yes	0	Not Graduate	No	2583	2358.0
4	Male	No	0	Graduate	No	6000	0.0
...	...	...	...	...	...	...	...
609	Female	No	0	Graduate	No	2900	0.0
610	Male	Yes	3+	Graduate	No	4106	0.0
611	Male	Yes	1	Graduate	No	8072	240.0
612	Male	Yes	2	Graduate	No	7583	0.0
613	Female	No	0	Graduate	Yes	4583	0.0

614 rows × 12 columns

Data Preprocessing

Handling Categorical values

In [67]:
```
import jupyterthemes as jt
```

In [68]:
```
!jt -t onedork
```

In [69]:
```
data['Gender']=data['Gender'].map({'Female':1,'Male':0})
data['Property_Area']=data['Property_Area'].map({'Urban':2,'Semiurban': 1,'Rural
data['Married']=data['Married'].map({'Yes':1,'No':0})
data['Education']=data['Education'].map({'Graduate':1,'Not Graduate':0})
data['Self_Employed']=data['Self_Employed'].map({'Yes':1,'No':0})
data['Loan_Status']=data['Loan_Status'].map({'Y':1,'N':0})
```

In [70]:
```
#Handling categorical feature Gender
data.head()
```

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIn
0	0.0	0.0	0	1	0.0	5849	0.0
1	0.0	1.0	1	1	0.0	4583	1508.0
2	0.0	1.0	0	1	1.0	3000	0.0
3	0.0	1.0	0	0	0.0	2583	2358.0
4	0.0	0.0	0	1	0.0	6000	0.0

Handling Missing values

In [71]:
```
#finding the sum of null values in each column
data.isnull().sum()
```

```
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 [72]:
```
#replacing + with space for filling the nan values
data['Dependents']=data['Dependents'].str.replace('+','')
```

In [73]:
```
data['Gender'] = data['Gender'].fillna(data['Gender'].mode()[0])
```

In [74]:
```
data['Married'] = data['Married'].fillna(data['Married'].mode()[0])
```

In [75]:
```
data['Dependents'] = data['Dependents'].fillna(data['Dependents'].mode()[0])
```

In [76]:
```
data['Self_Employed'] = data['Self_Employed'].fillna(data['Self_Employed'].mode(
```

In [77]:
```
data['LoanAmount'] = data['LoanAmount'].fillna(data['LoanAmount'].mode()[0])
```

In [78]:
```
data['Loan_Amount_Term'] = data['Loan_Amount_Term'].fillna(data['Loan_Amount_Ter
```

In [79]:
```
data['Credit_History'] = data['Credit_History'].fillna(data['Credit_History'].mc
```

In [80]:
```
data.isnull().sum()
```

```
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 [81]:
```
#getting bthe total info of the data after perfroming categorical to numericsal
data.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    float64
 1   Married            614 non-null    float64
 2   Dependents         614 non-null    object 
 3   Education          614 non-null    int64  
 4   Self_Employed      614 non-null    float64
 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(7), int64(4), object(1)
memory usage: 57.7+ KB
```

In [82]:
```
#changing the datype of each float column to int
data['Gender']=data['Gender'].astype('int64')
data['Married']=data['Married'].astype('int64')
data['Dependents']=data['Dependents'].astype('int64')
data['Self_Employed']=data['Self_Employed'].astype('int64')
data['CoapplicantIncome']=data['CoapplicantIncome'].astype('int64')
data['LoanAmount']=data['LoanAmount'].astype('int64')
data['Loan_Amount_Term']=data['Loan_Amount_Term'].astype('int64')
data['Credit_History']=data['Credit_History'].astype('int64')
```

In [83]:
```
data.info()
```

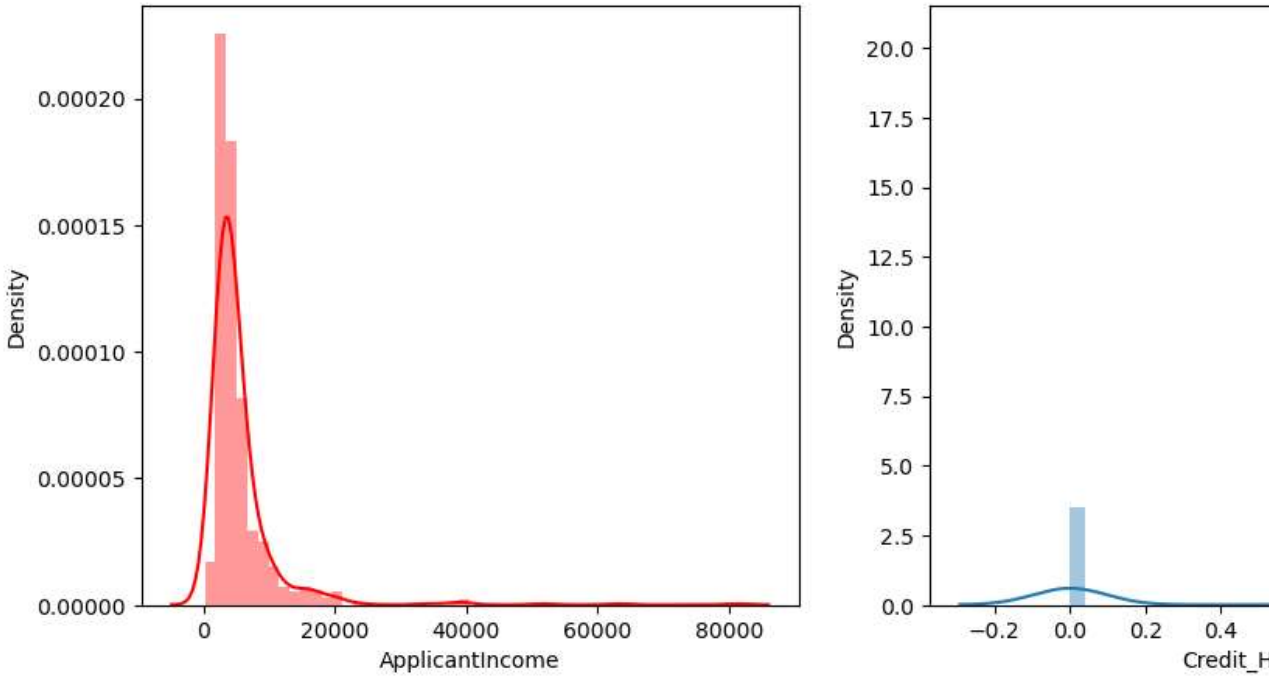
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Gender             614 non-null    int64
 1   Married            614 non-null    int64
 2   Dependents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

Univariate analysis

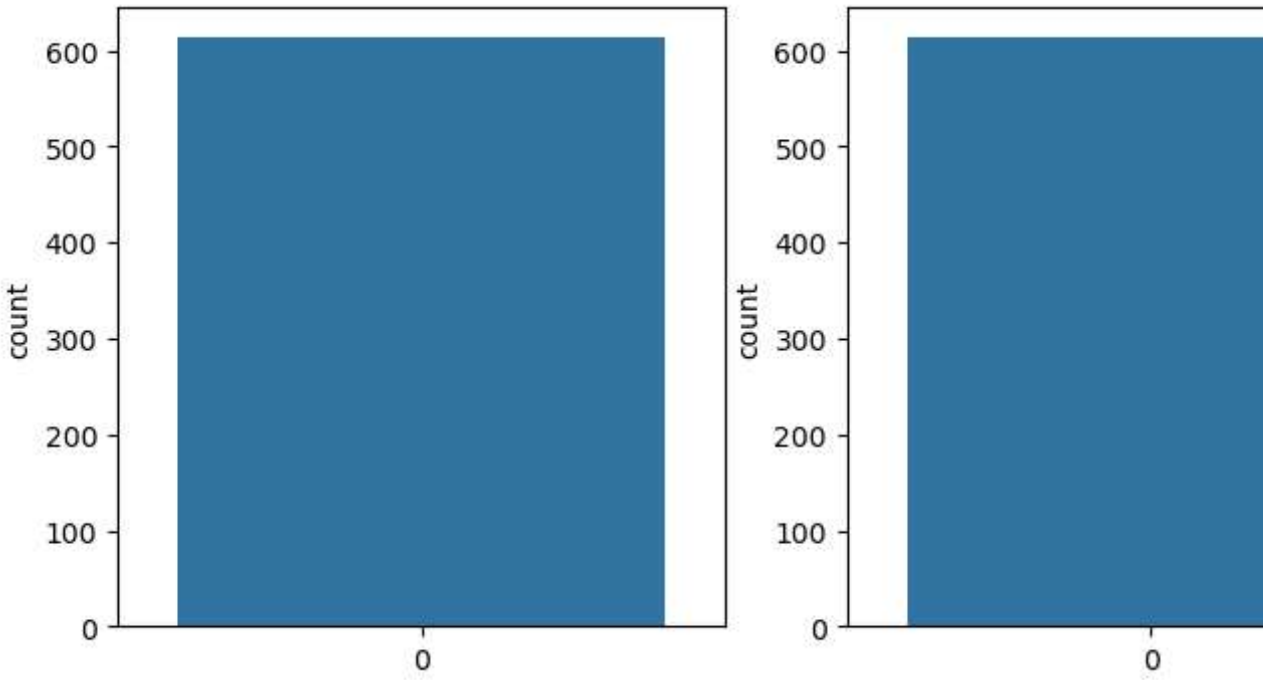
In [84]:

```
#plotting the using distplot
plt.figure(figsize=(12,5))
plt.subplot(121)
sns.distplot(data['ApplicantIncome'], color='r')
plt.subplot(122)
sns.distplot(data['Credit_History'])
plt.show()
```

```
C:\Users\manth\AppData\Local\Temp\ipykernel_11188\3941809966.py:4: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751 (https://gist.github.com/mwaskom/de44147ed29744

  sns.distplot(data['ApplicantIncome'], color='r')
C:\Users\manth\AppData\Local\Temp\ipykernel_11188\3941809966.py:6: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751 (https://gist.github.com/mwaskom/de44147ed29744

  sns.distplot(data['Credit_History'])
```

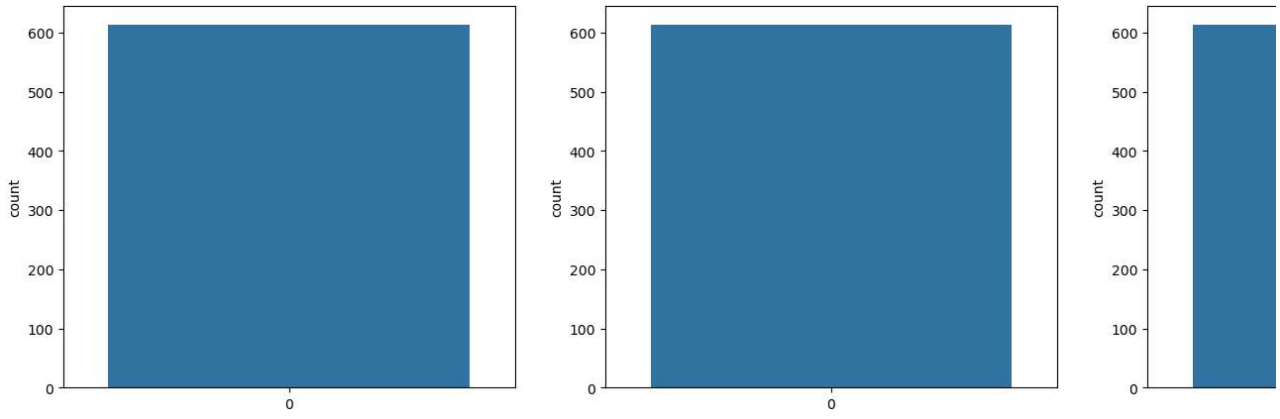

0.00020
0.00015
0.00010
0.00005
0.00000
Density
0
20000
40000
60000
80000
ApplicantIncome
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
Density
−0.2
0.0
0.2
0.4

In [85]:
```
#plotting the count plot
plt.figure(figsize=(18,4))
plt.subplot(1,4,1)
sns.countplot(data['Gender'])
plt.subplot(1,4,2)
sns.countplot(data['Education'])
plt.show()
```



Bivariate analysis

In [86]:
```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(20,5))

plt.subplot(131)
sns.countplot(data['Married'])

plt.subplot(132)
sns.countplot(data['Self_Employed'])

plt.subplot(133)
sns.countplot(data['Property_Area'])

plt.show()
```



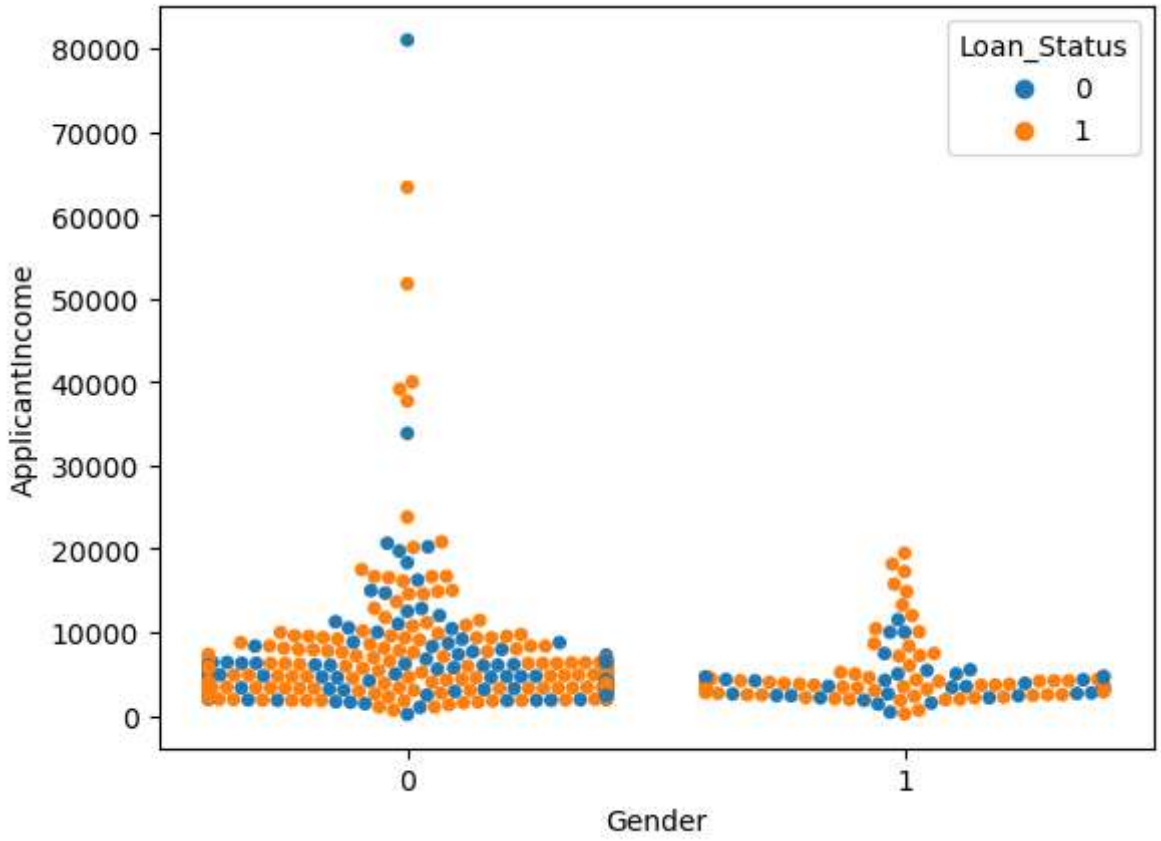
In [87]:
```
#plotted a coulms using cross tab function
pd.crosstab(data['Gender'],[data['Self_Employed']])
```

Self_Employed	0	1
Gender		
0	435	67
1	97	15

multi variate analysis

In [88]:
```
sns.swarmplot(x=data['Gender'], y=data['ApplicantIncome'], hue=data['Loan_Status

# Show the plot
plt.show()
```

```
C:\Users\manth\anaconda3\Lib\site-packages\seaborn\categorical.py:3544: UserWarning: 45.8% of the poir
decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
C:\Users\manth\anaconda3\Lib\site-packages\seaborn\categorical.py:3544: UserWarning: 61.6% of the poir
decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
C:\Users\manth\anaconda3\Lib\site-packages\seaborn\categorical.py:3544: UserWarning: 25.0% of the poir
decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
```



Balncing the Dataset

In [89]:
```
from sklearn.model_selection import train_test_split

X = data.drop('Loan_Status', axis=1)  # Exclude the target variable from the fea
y = data['Loan_Status']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
```

In [90]:
```
#Balancing the dataset by using smote
from imblearn.combine import SMOTETomek
```

In [97]:
```
from imblearn.combine import SMOTETomek

# Assuming X_train and y_train are your feature matrix and target variable
smote= SMOTETomek(sampling_strategy=0.90)
X_resampled, y_resampled = smote_tomek.fit_resample(X_train, y_train)
```

In [98]:
```
#dividing the dataset into dependent and independent y and x respectively
y = data['Loan_Status']
x = data.drop(columns=['Loan_Status'],axis=1)
```

In [99]:
```
#shape of x after seperating from the total data set
x.shape
```

```
(614, 11)
```

In [100]:
```
#shape of y
y.shape
```

```
(614,)
```

In [101]:
```
#creating a new x and y varialbles for the balnced set
X_bal, y_bal = smote.fit_resample(X_train, y_train)
```

In [102]:
```
#printing the values of y before balancing the data and after
print(y.value_counts())
print(y_bal.value_counts())
```

```
Loan_Status
1    422
0    192
Name: count, dtype: int64
Loan_Status
1    288
0    253
Name: count, dtype: int64
```

In [104]:
```
names = X_bal.columns
```

Scalling the dataset

In [106]:
```
# perfroming feature Scaling op[eration using standard scaller on X part of the
# there different type of values in the columns
sc=StandardScaler()
X_bal=sc.fit_transform(X_bal)
```

In [107]:
```
X_bal = pd.DataFrame(X_bal,columns=names)
```

In [109]:
```
#splitting the dataset in train and test on balnced dataset
X_train, X_test, y_train, y_test = train_test_split(
     X_bal, y_bal, test_size=0.33, random_state=42)
```

In [111]:
```
#splitting the dataset in train and test on balnmced datasew
X_train, X_test, y_train, y_test = train_test_split(
     X_bal, y_bal, test_size=0.33, random_state=42)
```

In [112]:
```
X_test.shape
```

```
(179, 11)
```

In [113]:
```
y_train.shape, y_test.shape
```

```
((362,), (179,))
```

Model building

In [114]:
```
#importing and building the random forest model
def RandomForest(X_train,X_test,y_train,y_test):
    model = RandomForestClassifier()
    model.fit(X_train,y_train)
    y_tr = model.predict(X_train)
    print(accuracy_score(y_tr,y_train))
    yPred = model.predict(X_test)
    print(accuracy_score(yPred,y_test))
```

In [115]:
```
#printing the train accuracy and test accuracy respectively
RandomForest(X_train,X_test,y_train,y_test)
```

```
1.0
0.8379888268156425
```

In [116]:
```
#importing and building the Decision tree model
def decisionTree(X_train,X_test,y_train,y_test):
    model = DecisionTreeClassifier()
    model.fit(X_train,y_train)
    y_tr = model.predict(X_train)
    print(accuracy_score(y_tr,y_train))
    yPred = model.predict(X_test)
    print(accuracy_score(yPred,y_test))
```

In [117]:
```
#printing the train accuracy and test accuracy respectively
decisionTree(X_train,X_test,y_train,y_test)
```

```
1.0
0.776536312849162
```

In [118]:
```
#importing and building the KNN model
def KNN(X_train,X_test,y_train,y_test):
    model = KNeighborsClassifier()
    model.fit(X_train,y_train)
    y_tr = model.predict(X_train)
    print(accuracy_score(y_tr,y_train))
    yPred = model.predict(X_test)
    print(accuracy_score(yPred,y_test))
```

In [119]:
```
#printing the train accuracy and test accuracy respectively
KNN(X_train,X_test,y_train,y_test)
```

```
0.8397790055248618
0.7486033519553073
```

In [120]:
```
#importing and building the Xg boost model
def XGB(X_train,X_test,y_train,y_test):
    model = GradientBoostingClassifier()
    model.fit(X_train,y_train)
    y_tr = model.predict(X_train)
    print(accuracy_score(y_tr,y_train))
    yPred = model.predict(X_test)
    print(accuracy_score(yPred,y_test))
```

In [121]:
```
#printing the train accuracy and test accuracy respectively
XGB(X_train,X_test,y_train,y_test)
```

```
0.9530386740331491
0.8324022346368715
```

Hyper parameter tuning

In [122]:
```
rf = RandomForestClassifier()
```

In [123]:
```
# giving some parameters that can be used in randized search cv
parameters = {
                'n_estimators' : [1,20,30,55,68,74,90,120,115],
                 'criterion':['gini','entropy'],
                 'max_features' : ["auto", "sqrt", "log2"],
        'max_depth' : [2,5,8,10], 'verbose' : [1,2,3,4,6,8,9,10]



}
```

In [124]:
```
#performing the randomized cv
RCV  = RandomizedSearchCV(estimator=rf,param_distributions=parameters,cv=10,n_it
```

In [125]:
```
RCV.fit(X_train,y_train)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  90 out of  90 | elapsed:    0.0s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  90 out of  90 | elapsed:    0.0s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    0.0s remaining:    0.0s


building tree 1 of 90
building tree 2 of 90
```

In [126]:
```
#getting the best paarmets from the giving list and best score from them
bt_params = RCV.best_params_
bt_score = RCV.best_score_
```

In [127]:
```
bt_params
```

```
{'verbose': 9,
 'n_estimators': 74,
 'max_features': 'log2',
 'max_depth': 5,
 'criterion': 'gini'}
```

In [128]:
```
bt_score
```

```
0.8011261261261261
```

In [129]:
```
# training and test the xg boost model on the best parameters gor from the rando
def RandomForest(X_tarin,X_test,y_train,y_test):
    model = RandomForestClassifier(verbose= 10, n_estimators= 120, max_features=
    model.fit(X_train,y_train)
    y_tr = model.predict(X_train)
    print(accuracy_score(y_tr,y_train))
    yPred = model.predict(X_test)
    print(accuracy_score(yPred,y_test))
```

In [130]:
```
model = RandomForestClassifier(verbose= 10, n_estimators= 120, max_features= 'lo
model.fit(X_train,y_train)
```

```
building tree 1 of 120
building tree 2 of 120
building tree 3 of 120
building tree 4 of 120
building tree 5 of 120
building tree 6 of 120
building tree 7 of 120
building tree 8 of 120
building tree 9 of 120
building tree 10 of 120
building tree 11 of 120
building tree 12 of 120
building tree 13 of 120
building tree 14 of 120
building tree 15 of 120
building tree 16 of 120
building tree 17 of 120
building tree 18 of 120
building tree 19 of 120
building tree 20 of 120
building tree 21 of 120
building tree 22 of 120
building tree 23 of 120
building tree 24 of 120
```

In [131]:
```
#printing the train and test accutracy after hyper parameter tuning
RandomForest(X_train,X_test,y_train,y_test)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   6 out of   6 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   7 out of   7 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   8 out of   8 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   9 out of   9 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done 120 out of 120 | elapsed:    0.1s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   6 out of   6 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   7 out of   7 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   8 out of   8 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   9 out of   9 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done 120 out of 120 | elapsed:    0.0s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
```

In [132]:
```
X_train
```

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapplica
265	-0.437756	-1.091001	-0.674797	0.65091	-0.336406	-0.381676	0.376112
287	2.284380	-1.091001	-0.674797	0.65091	-0.336406	-0.475413	-0.025007
495	-0.437756	-1.091001	-0.674797	0.65091	-0.336406	-0.173203	-0.481463
5	-0.437756	0.916589	1.504697	-1.53631	-0.336406	-0.342158	0.057015
116	2.284380	-1.091001	2.594445	0.65091	-0.336406	-0.395995	-0.481463
...	...	...	...	...	...	...	...
71	2.284380	-1.091001	-0.674797	0.65091	2.972602	-0.488204	0.000835
106	2.284380	-1.091001	-0.674797	0.65091	2.972602	2.023980	-0.481463
270	-0.437756	0.916589	-0.674797	0.65091	2.972602	-0.324212	-0.481463
435	-0.437756	-1.091001	-0.674797	-1.53631	-0.336406	-0.401913	-0.481463
102	-0.437756	0.916589	-0.674797	0.65091	2.972602	0.840529	-0.481463

362 rows × 11 columns

Saving the Model

In [133]:
```
#saviung the model by using pickle function
pickle.dump(model,open('rdf.pkl','wb'))
```

In [ ]: