```
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, f1_score
```

```
#importing the dataset which is in csv file
data = pd.read_csv('/content/test.csv')
data = pd.read_csv('/content/train.csv')
data
```

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

614 rows × 13 columns

```
data.drop(['Loan_ID'],axis=1,inplace=True)
```

```
data.head()
```

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term
0	Male	No	0	Graduate	No	5849	0.0	NaN	360.0
1	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.0
2	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	360.0
3	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0
4	Male	No	0	Graduate	No	6000	0.0	141.0	360.0

```
data['Gender']=data['Gender'].map({'Female':1,'Male':0})
data.head()
```

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term
0	0.0	No	0	Graduate	No	5849	0.0	NaN	360.0
1	0.0	Yes	1	Graduate	No	4583	1508.0	128.0	360.0
2	0.0	Yes	0	Graduate	Yes	3000	0.0	66.0	360.0
3	0.0	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0
4	0.0	No	0	Graduate	No	6000	0.0	141.0	360.0

```
data['Property_Area']=data['Property_Area'].map({'Urban':2,'Semiurban': 1,'Rural':0})
data.head()
```

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term
0	0.0	No	0	Graduate	No	5849	0.0	NaN	360.0
1	0.0	Yes	1	Graduate	No	4583	1508.0	128.0	360.0
2	0.0	Yes	0	Graduate	Yes	3000	0.0	66.0	360.0
3	0.0	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0
4	0.0	No	0	Graduate	No	6000	0.0	141.0	360.0

```
data['Married']=data['Married'].map({'Yes':1,'No':0})
data.head()
```

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term
0	0.0	0.0	0	Graduate	No	5849	0.0	NaN	360.0
1	0.0	1.0	1	Graduate	No	4583	1508.0	128.0	360.0
2	0.0	1.0	0	Graduate	Yes	3000	0.0	66.0	360.0
3	0.0	1.0	0	Not Graduate	No	2583	2358.0	120.0	360.0
4	0.0	0.0	0	Graduate	No	6000	0.0	141.0	360.0

```
data['Education']=data['Education'].map({'Graduate':1,'Not Graduate':0})
data.head()
```

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term
0	0.0	0.0	0	1	No	5849	0.0	NaN	360.0
1	0.0	1.0	1	1	No	4583	1508.0	128.0	360.0
2	0.0	1.0	0	1	Yes	3000	0.0	66.0	360.0
3	0.0	1.0	0	0	No	2583	2358.0	120.0	360.0
4	0.0	0.0	0	1	No	6000	0.0	141.0	360.0

```
data['Self_Employed']=data['Self_Employed'].map({'Yes':1,'No':0})
data.head()
```

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term
0	0.0	0.0	0	1	0.0	5849	0.0	NaN	360.0
1	0.0	1.0	1	1	0.0	4583	1508.0	128.0	360.0
2	0.0	1.0	0	1	1.0	3000	0.0	66.0	360.0
3	0.0	1.0	0	0	0.0	2583	2358.0	120.0	360.0

```
data['Loan_Status']=data['Loan_Status'].map({'Y':1,'N':0})
data.head()
```

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term
0	0.0	0.0	0	1	0.0	5849	0.0	NaN	360.0
1	0.0	1.0	1	1	0.0	4583	1508.0	128.0	360.0
2	0.0	1.0	0	1	1.0	3000	0.0	66.0	360.0
3	0.0	1.0	0	0	0.0	2583	2358.0	120.0	360.0
4	0.0	0.0	0	1	0.0	6000	0.0	141.0	360.0

```
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
```

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

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

```
data['Dependents'] = data['Dependents'].str.replace('+','')
```

```
<ipython-input-172-e7493c2d1d37>:1: FutureWarning: The default value of regex will change from True to False in a future version. In adc
  data['Dependents'] = data['Dependents'].str.replace('+','')
```

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

```
data['Self_Employed'] = data['Self_Employed'].fillna(data['Self_Employed'].mode()[0])
```

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

```
data['Loan_Amount_Term'] = data['Loan_Amount_Term'].fillna(data['Loan_Amount_Term'].mode()[0])
```

```
data['Credit_History'] = data['Credit_History'].fillna(data['Credit_History'].mode()[0])
```

```
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
```

```
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
```

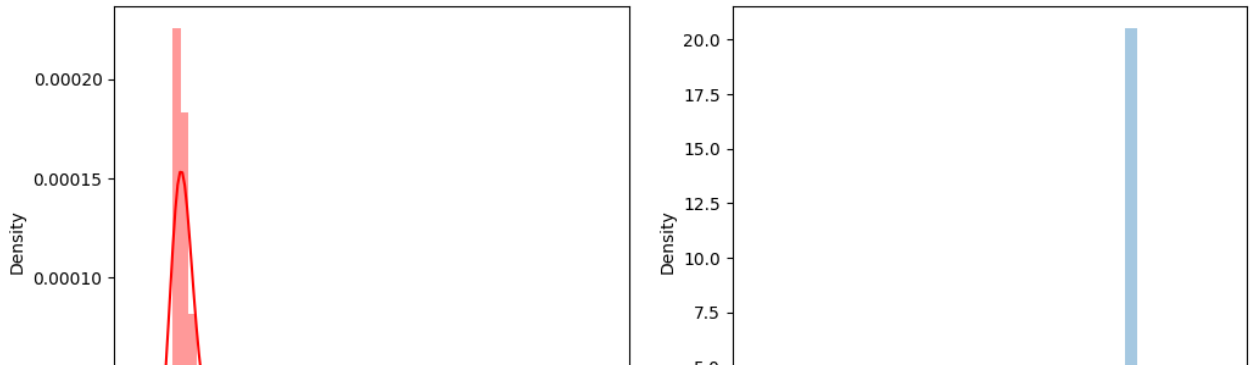
```
data['Gender'] = data['Gender'].astype('int64')
data['Married'] = data['Married'].astype('int64')
data['Dependents'] = data['Dependents'].astype('int64')
data['Self_Employed'] = data['Self_Employed'].astype('int64')
data['CoapplicantIncome'] = data['CoapplicantIncome'].astype('int64')
data['LoanAmount'] = data['LoanAmount'].astype('int64')
data['Loan_Amount_Term'] = data['Loan_Amount_Term'].astype('int64')
data['Credit_History'] = data['Credit_History'].astype('int64')
```

```
data.info()
```

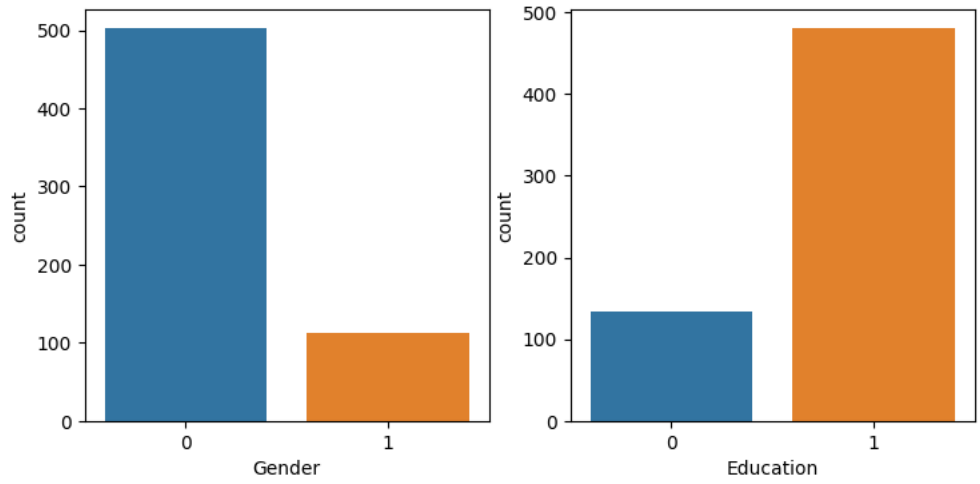
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Gender             614 non-null    int64
 1   Married            614 non-null    int64
 2   Dependents         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
```

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

```
<ipython-input-182-4b78f43a4171>:3: 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

  sns.distplot(data['ApplicantIncome'], color='r')
<ipython-input-182-4b78f43a4171>:5: 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

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

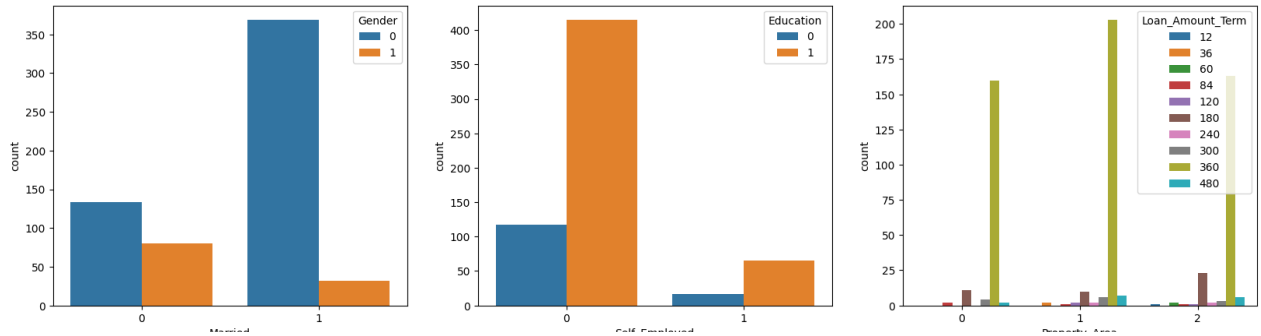



```
plt.figure(figsize=(18,4))
plt.subplot(1,4,1)
sns.countplot(x = 'Gender',data = data)
plt.subplot(1,4,2)
sns.countplot(x = 'Education',data = data)
plt.show()
```




```
plt.figure(figsize=(20,5))
plt.subplot(131)
sns.countplot(x = 'Married', hue = 'Gender', data = data)
plt.subplot(132)
sns.countplot(x = 'Self_Employed', hue = 'Education', data = data)
plt.subplot(133)
sns.countplot(x = 'Property_Area', hue = 'Loan_Amount_Term', data = data)
```

```
<Axes: xlabel='Property_Area', ylabel='count'>
```

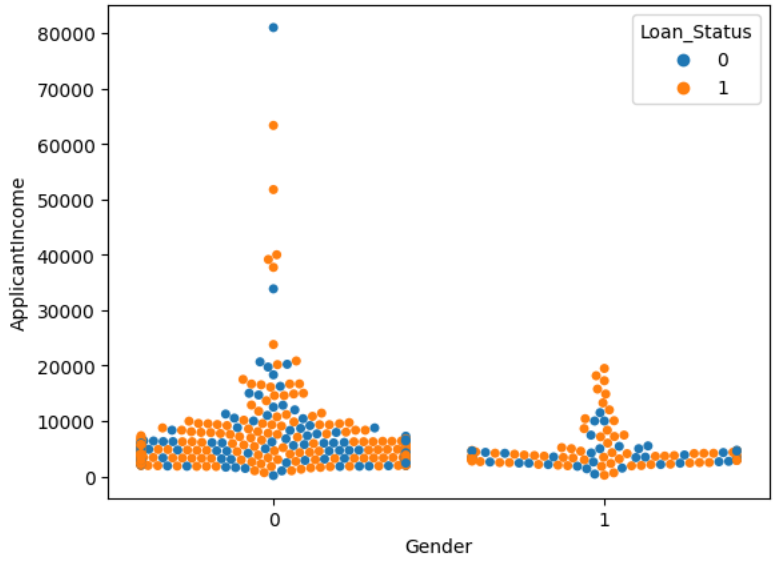



```
pd.crosstab(data['Gender'],[data['Self_Employed']])
```

Self_Employed	0	1
Gender		
0	435	67
1	97	15

```
sns.swarmplot(x = "Gender",y = "ApplicantIncome", hue = "Loan_Status", data = data)
```

```
/usr/local/lib/python3.9/dist-packages/seaborn/categorical.py:3544: UserWarning: 45.8% of the points cannot be placed; you
  warnings.warn(msg, UserWarning)
<Axes: xlabel='Gender',
ylabel='ApplicantIncome'>/usr/local/lib/python3.9/dist-packages/seaborn/categorical.py:3544: UserWarning: 61.6% of the poin
  warnings.warn(msg, UserWarning)
/usr/local/lib/python3.9/dist-packages/seaborn/categorical.py:3544: UserWarning: 25.0% of the points cannot be placed; you
  warnings.warn(msg, UserWarning)
```




```
from imblearn.combine import SMOTETomek
```

```
smote = SMOTETomek()
```

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

```
x.shape
```

```
(614, 11)
```

```
y.shape
```

```
(614,)
```

```
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    356
0    356
Name: Loan_Status, dtype: int64
```

```
names=x_bal.columns


x_bal.head()
```

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term
0	0	0	0	1	0	5849	0	120	360
1	0	1	1	1	0	4583	1508	128	360
2	0	1	0	1	1	3000	0	66	360
3	0	1	0	0	0	2583	2358	120	360
4	0	0	0	1	0	6000	0	141	360

```
sc=StandardScaler()
x_bal=sc.fit_transform(x_bal)


x_bal
```

```
array([[-0.42285689, -1.15186691, -0.71535408, ...,  0.29177308,
         0.6336993 ,  1.32799102],
       [-0.42285689,  0.86815585,  0.35019425, ...,  0.29177308,
         0.6336993 , -1.21717616],
       [-0.42285689,  0.86815585, -0.71535408, ...,  0.29177308,
         0.6336993 ,  1.32799102],
       ...,
       [-0.42285689, -1.15186691, -0.71535408, ...,  0.29177308,
         0.6336993 , -1.21717616],
       [-0.42285689,  0.86815585, -0.71535408, ...,  0.29177308,
        -1.57803551, -1.21717616],
       [-0.42285689,  0.86815585, -0.71535408, ...,  0.29177308,
        -1.57803551,  0.05540743]])
```

```
x_bal = pd.DataFrame(x_bal,columns=names)
x_bal.head()
```

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Te
0	-0.422857	-1.151867	-0.715354	0.638055	-0.332813	0.097527	-0.507924	-0.295570	0.2917
1	-0.422857	0.868156	0.350194	0.638055	-0.332813	-0.118137	-0.031761	-0.197191	0.2917
2	-0.422857	0.868156	-0.715354	0.638055	3.004691	-0.387803	-0.507924	-0.959631	0.2917
3	-0.422857	0.868156	-0.715354	-1.567262	-0.332813	-0.458839	0.236634	-0.295570	0.2917
4	-0.422857	-1.151867	-0.715354	0.638055	-0.332813	0.123250	-0.507924	-0.037324	0.2917

```
x_train, x_test, y_train, y_test = train_test_split(
    x_bal, y_bal, test_size=0.33, random_state=42)


x_train.shape
```

```
(477, 11)
```

```
x_test.shape
```

```
(235, 11)
```

```
y_train.shape, y_test.shape
```

```
((477,), (235,))
```

```
def decisionTree(x_train,x_test,y_train,y_test):
    dt=DecisionTreeClassifier()
    dt.fit(x_train,y_train)
    yPred = dt.predict(x_test)
    print('***DecisionTreeClassifier***')
    print('Confusion matrix')
    print(confusion_matrix(y_test,yPred))
    print('Classification report')
    print(classification_report(y_test,yPred))
```

```
def randomForest(x_train,x_test,y_train,y_test):
    rf = RandomForestClassifier()
    rf.fit(x_train,y_train)
    yPred = rf.predict(x_test)
    print('***RandomForestClassifier***')
    print('Confusion matrix')
    print(confusion_matrix(y_test,yPred))
    print('Classification report')
    print(classification_report(y_test,yPred))
```

```
def KNN(x_train,x_test,y_train,y_test):
    dt = KNeighborsClassifier()
    knn.fit(x_train,y_train)
    yPred = knn.predict(x_test)
    print('***KNeighborsClassifier***')
    print('Confusion matrix')
    print(confusion_matrix(y_test,yPred))
    print('Classification report')
    print(classification_report(y_test,yPred))
```

```
def xgboost(x_train,x_test,y_train,y_test):
    xg = GradientBoostingClassifier()
    xg.fit(x_train,y_train)
    yPred = xg.predict(x_test)
    print('***GradientBoostingClassifier***')
    print('Confusion matrix')
    print(confusion_matrix(y_test,yPred))
    print('Classification report')
    print(classification_report(y_test,yPred))
```

```
import tensorflow
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

```
classifier = Sequential()
```

```
classifier.add(Dense(units=100, activation='relu', input_dim=11))
```

```
classifier.add(Dense(units=50, activation='relu'))
```

```
classifier.add(Dense(units=1, activation='sigmoid'))
```

```
classifier.compile(optimizer='adam', loss='binary_crossentropy',metrics=['accuracy'])
```

```
model_history = classifier.fit(x_train, y_train, batch_size=100, validation_split=0.2, epochs=100)
```

```
Epoch 1/100
4/4 [==============================] - 1s 76ms/step - loss: 0.6869 - accuracy: 0.5486 - val_loss: 0.6538 - val_accuracy: 0.6667
Epoch 2/100
4/4 [==============================] - 0s 15ms/step - loss: 0.6423 - accuracy: 0.6824 - val_loss: 0.6258 - val_accuracy: 0.7292
```

```
Epoch 3/100
4/4 [==============================] - 0s 14ms/step - loss: 0.6066 - accuracy: 0.7480 - val_loss: 0.6010 - val_accuracy: 0.7396
Epoch 4/100
4/4 [==============================] - 0s 16ms/step - loss: 0.5734 - accuracy: 0.7585 - val_loss: 0.5801 - val_accuracy: 0.7604
Epoch 5/100
4/4 [==============================] - 0s 14ms/step - loss: 0.5465 - accuracy: 0.7638 - val_loss: 0.5602 - val_accuracy: 0.7917
Epoch 6/100
4/4 [==============================] - 0s 20ms/step - loss: 0.5217 - accuracy: 0.7979 - val_loss: 0.5446 - val_accuracy: 0.7917
Epoch 7/100
4/4 [==============================] - 0s 14ms/step - loss: 0.4998 - accuracy: 0.7953 - val_loss: 0.5321 - val_accuracy: 0.7812
Epoch 8/100
4/4 [==============================] - 0s 15ms/step - loss: 0.4808 - accuracy: 0.7979 - val_loss: 0.5225 - val_accuracy: 0.7708
Epoch 9/100
4/4 [==============================] - 0s 23ms/step - loss: 0.4644 - accuracy: 0.8005 - val_loss: 0.5156 - val_accuracy: 0.7812
Epoch 10/100
4/4 [==============================] - 0s 14ms/step - loss: 0.4491 - accuracy: 0.8136 - val_loss: 0.5102 - val_accuracy: 0.7708
Epoch 11/100
4/4 [==============================] - 0s 21ms/step - loss: 0.4377 - accuracy: 0.8189 - val_loss: 0.5063 - val_accuracy: 0.7812
Epoch 12/100
4/4 [==============================] - 0s 20ms/step - loss: 0.4281 - accuracy: 0.8189 - val_loss: 0.5039 - val_accuracy: 0.7812
Epoch 13/100
4/4 [==============================] - 0s 22ms/step - loss: 0.4184 - accuracy: 0.8163 - val_loss: 0.5039 - val_accuracy: 0.7917
Epoch 14/100
4/4 [==============================] - 0s 14ms/step - loss: 0.4111 - accuracy: 0.8189 - val_loss: 0.5045 - val_accuracy: 0.7917
Epoch 15/100
4/4 [==============================] - 0s 15ms/step - loss: 0.4048 - accuracy: 0.8163 - val_loss: 0.5069 - val_accuracy: 0.7812
Epoch 16/100
4/4 [==============================] - 0s 14ms/step - loss: 0.3988 - accuracy: 0.8163 - val_loss: 0.5085 - val_accuracy: 0.7812
Epoch 17/100
4/4 [==============================] - 0s 19ms/step - loss: 0.3936 - accuracy: 0.8189 - val_loss: 0.5099 - val_accuracy: 0.7812
Epoch 18/100
4/4 [==============================] - 0s 13ms/step - loss: 0.3887 - accuracy: 0.8189 - val_loss: 0.5125 - val_accuracy: 0.7812
Epoch 19/100
4/4 [==============================] - 0s 17ms/step - loss: 0.3836 - accuracy: 0.8215 - val_loss: 0.5138 - val_accuracy: 0.7812
Epoch 20/100
4/4 [==============================] - 0s 15ms/step - loss: 0.3798 - accuracy: 0.8189 - val_loss: 0.5167 - val_accuracy: 0.7604
Epoch 21/100
4/4 [==============================] - 0s 15ms/step - loss: 0.3762 - accuracy: 0.8268 - val_loss: 0.5197 - val_accuracy: 0.7604
Epoch 22/100
4/4 [==============================] - 0s 21ms/step - loss: 0.3716 - accuracy: 0.8320 - val_loss: 0.5261 - val_accuracy: 0.7708
Epoch 23/100
4/4 [==============================] - 0s 22ms/step - loss: 0.3689 - accuracy: 0.8320 - val_loss: 0.5311 - val_accuracy: 0.7812
Epoch 24/100
4/4 [==============================] - 0s 22ms/step - loss: 0.3651 - accuracy: 0.8346 - val_loss: 0.5310 - val_accuracy: 0.7604
Epoch 25/100
4/4 [==============================] - 0s 16ms/step - loss: 0.3610 - accuracy: 0.8320 - val_loss: 0.5306 - val_accuracy: 0.7500
Epoch 26/100
4/4 [==============================] - 0s 20ms/step - loss: 0.3585 - accuracy: 0.8320 - val_loss: 0.5319 - val_accuracy: 0.7500
Epoch 27/100
4/4 [==============================] - 0s 14ms/step - loss: 0.3548 - accuracy: 0.8320 - val_loss: 0.5354 - val_accuracy: 0.7500
Epoch 28/100
4/4 [==============================] - 0s 15ms/step - loss: 0.3520 - accuracy: 0.8425 - val_loss: 0.5399 - val_accuracy: 0.7604
Epoch 29/100
4/4 [==============================] - 0s 14ms/step - loss: 0.3489 - accuracy: 0.8451 - val_loss: 0.5400 - val_accuracy: 0.7500
```

```
y_pred = classifier.predict(x_test)
```

```
8/8 [==============================] - 0s 2ms/step
```

```
y_pred
```

```
array([[9.80460271e-02],
       [4.13449973e-01],
       [8.83821785e-01],
       [3.17151606e-01],
       [8.31635833e-01],
       [4.38936919e-01],
       [4.97584902e-02],
       [3.00325890e-04],
       [9.72762883e-01],
       [5.43831587e-02],
       [9.62490499e-01],
       [9.51407015e-01],
       [5.80228984e-01],
       [1.34293869e-01],
       [4.79236223e-05],
       [5.42867114e-04],
       [3.21026258e-02],
       [9.88419592e-01],
       [1.04508951e-01],
       [8.82773638e-01],
       [4.64332243e-03],
       [7.38639534e-01],
```

```
       [4.93913307e-04],
       [2.64845461e-01],
       [9.24660563e-01],
       [1.77320588e-04],
       [8.38282466e-01],
       [1.14442126e-04],
       [9.99604523e-01],
       [3.58364685e-03],
       [3.25512439e-02],
       [8.68165433e-01],
       [6.70341432e-01],
       [1.30200675e-02],
       [4.35363501e-02],
       [7.88216352e-01],
       [8.98094416e-01],
       [9.32443559e-01],
       [1.71921507e-03],
       [5.08780868e-05],
       [2.47024396e-03],
       [9.59852934e-01],
       [5.29910684e-01],
       [8.95197511e-01],
       [9.39983666e-01],
       [6.87282622e-01],
       [8.57955031e-03],
       [4.92694169e-01],
       [6.82965328e-05],
       [2.20165029e-03],
       [6.47255361e-01],
       [6.84043169e-01],
       [7.12731481e-01],
       [8.92526329e-01],
       [1.60877581e-03],
       [3.70681167e-01],
       [8.19768846e-01],
       [1.00554757e-01],
```

```
y_pred = y_pred.astype(int)
y_pred
```

```
array([[0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
```

```
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
```

```
print(accuracy_score(y_pred, y_test))
print("ANN Model")
print("Confusion_Matrix")
print(confusion_matrix(y_test, y_pred))
print("Classification Report")
print(classification_report(y_test, y_pred))
```

```
0.49361702127659574
ANN Model
Confusion_Matrix
[[116   0]
 [119   0]]
Classification Report
              precision    recall  f1-score   support

           0       0.49      1.00      0.66       116
           1       0.00      0.00      0.00       119

    accuracy                           0.49       235
   macro avg       0.25      0.50      0.33       235
weighted avg       0.24      0.49      0.33       235

/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-de
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-de
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-de
  _warn_prf(average, modifier, msg_start, len(result))
```

```
rf = RandomForestClassifier()
```

```
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]
}
```

```
RCV  = RandomizedSearchCV(estimator=rf,param_distributions=parameters,cv=10,n_iter=4)
```

```
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   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  20 out of  20 | 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   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  20 out of  20 | 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   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  20 out of  20 | 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   6 out of   6 | elapsed:    0.0s remaining:    0.0s
```

```
bt_params = RCV.best_estimator_
bt_score = RCV.best_score_
```

```
bt_params
```

```
RandomForestClassifier
RandomForestClassifier(max_depth=10, n_estimators=74, verbose=9)
```

```
[Parallel(n_jobs=1)]: Done   8 out of   8 | elapsed:    0.0s remaining:    0.0s
```

```
bt_score
```

```
0.800709219858156
```

```
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.0s remaining:    0.0s
```

```
def RandomForest(x_train,x_test,y_train,y_test):
    model = RandomForestClassifier(verbose= 4, n_estimators= 68,max_features= 'auto',max_depth= 8,criterion= 'entropy')
    model.fit(x_train,y_train)
    y_tr = model.predict(x_train)
    print("Training Accuracy")
    print(accuracy_score(y_tr,y_train))
    yPred = model.predict(x_test)
    print('Testing Accuracy')
    print(accuracy_score(yPred,y_test))
```

```
model = RandomForestClassifier(verbose= 4, n_estimators= 68,max_features= 'auto',max_depth= 8,criterion= 'entropy')
model.fit(x_train,y_train)
```

```
building tree 1 of 68
building tree 2 of 68
building tree 3 of 68
building tree 4 of 68
building tree 5 of 68
building tree 6 of 68
building tree 7 of 68
building tree 8 of 68
building tree 9 of 68
building tree 10 of 68
building tree 11 of 68
building tree 12 of 68
building tree 13 of 68
building tree 14 of 68
building tree 15 of 68
building tree 16 of 68
building tree 17 of 68
building tree 18 of 68
building tree 19 of 68
building tree 20 of 68
building tree 21 of 68
building tree 22 of 68
building tree 23 of 68
building tree 24 of 68
building tree 25 of 68
building tree 26 of 68
building tree 27 of 68
building tree 28 of 68
building tree 29 of 68
building tree 30 of 68
building tree 31 of 68
building tree 32 of 68
building tree 33 of 68
building tree 34 of 68
building tree 35 of 68
building tree 36 of 68
building tree 37 of 68
building tree 38 of 68
building tree 39 of 68
building tree 40 of 68
building tree 41 of 68
building tree 42 of 68
```

```
RandomForest(x_train,x_test,y_train,y_test)
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/ensemble/_forest.py:424: FutureWarning: `max_features='auto'` has been deprecated in 1
  warn(
[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
building tree 1 of 68
building tree 2 of 68
building tree 3 of 68
building tree 4 of 68
building tree 5 of 68
building tree 6 of 68
building tree 7 of 68
building tree 8 of 68
building tree 9 of 68
building tree 10 of 68
building tree 11 of 68
building tree 12 of 68
building tree 13 of 68
building tree 14 of 68
building tree 15 of 68
building tree 16 of 68
building tree 17 of 68
building tree 18 of 68
building tree 19 of 68
building tree 20 of 68
building tree 21 of 68
building tree 22 of 68
building tree 23 of 68
building tree 24 of 68
building tree 25 of 68
building tree 26 of 68
building tree 27 of 68
building tree 28 of 68
building tree 29 of 68
building tree 30 of 68
building tree 31 of 68
building tree 32 of 68
building tree 33 of 68
```

```
building tree 34 of 68
building tree 35 of 68
building tree 36 of 68
building tree 37 of 68
building tree 38 of 68
building tree 39 of 68
building tree 40 of 68
building tree 41 of 68
building tree 42 of 68
building tree 43 of 68
building tree 44 of 68
building tree 45 of 68
building tree 46 of 68
building tree 47 of 68
building tree 48 of 68
building tree 49 of 68
```

```
pickle.dump(model,open('rdf.pkl','wb'))
```

```
pickle.dump(sc,open('scale.pkl','wb'))
```

```
from flask import Flask
import numpy as np
import pickle
```

```
app = Flask(__name__)
model = pickle.load(open(r'rdf.pkl', 'rb'))
scale = pickle.load(open(r'scale.pkl', 'rb'))
```

```
@app.route('/') # rendering the html templet
def home():
  return render_template('home.html')
```

```
@app.route('/submit',methods=["POST","GET"])# route to show the prediction in a UI
def submit():
  # reading the inputs given by the user
  input_feature=[int(X) for x in request.form.value()]
  #input_feature = np.transpose(input_feature)
  input_feature=[np.array(input_feature)]
  print(input_feature)
  names = ['Gender', 'Married', 'Departments', 'Education', 'Self_Empolyed', 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount', 'Loan_Amou
  print(data)




   # predictions using the loaded model file
  prediction=model.predict(date)
  print(prediction)
  prediction = int(prediction)
  print(type(prediction))

  if (prediction == 0):
      return render_template("output.html",result ="Loan will Not be Approved")
  else:
      return render_template("output.html",result = "Loan will be Approved")
  # showing the prediction results in a UI


if __name__=="__main__":
  def os():
   # app.run(host='0.0.0.0', port=8000,debug=True)   # running the app
   port=int(os.environ.get('PORT',5000))
   app.run(debug=False)
```