```
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,
#data=pd.read_csv('Personal loan_prediction.csv')
#data
import csv
from google.colab import files
upload=files.upload()
```

Choose files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving loan_prediction.csv to loan_prediction (1).csv
```

```
data = pd.read_csv('loan_prediction.csv')
```

```
data
```

Loan_ID Gender Married Dependents Education Self_Employed Applicant

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 367 entries, 0 to 366
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Loan_ID            367 non-null    object 
 1   Gender             356 non-null    object 
 2   Married            367 non-null    object 
 3   Dependents         357 non-null    object 
 4   Education          367 non-null    object 
 5   Self_Employed      344 non-null    object 
 6   ApplicantIncome    367 non-null    int64  
 7   CoapplicantIncome  367 non-null    int64  
 8   LoanAmount         362 non-null    float64
 9   Loan_Amount_Term   361 non-null    float64
 10  Credit_History     338 non-null    float64
 11  Property_Area      367 non-null    object 
dtypes: float64(3), int64(2), object(7)
memory usage: 34.5+ KB
```

367 rows × 12 columns

```
data.isnull().sum()
```

```
Loan_ID               0
Gender               11
Married               0
Dependents           10
Education             0
Self_Employed        23
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount            5
Loan_Amount_Term      6
Credit_History       29
Property_Area         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('+','')
```

```
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'].m

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

data['Gender']=data['Gender'].astype('int64')
```

Double-click (or enter) to edit

```
data['Married']=data['Married'].astype('int64')

data['Dependents']=data['Dependents'].astype('int64')

data['Slef_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')

from imblearn.combine import SMOTETomek

smote=SMOTETomek(0,90)

y = data['Loan_Status']

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

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

print(y.value_counts())

print(y_bal.value_counts())

data.describe()


plt.figure(figsize=(12,5))
```

```
plt.subplot(121)


sns.distplot(data['ApplicantIncome'],color='r')

plt.subplot(122)


sns.distplot(data['Credit_History'])

plt.show()


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()

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

plt.subplot(131)

sns.countplot(data['Married'],hug=data['Gender'])

plt.subplot(132)

sns.countplot(data['Self_Employed'],hug=data['Education'])

plt.subplot(133)

sns.countplot(data['Property_Area'],hug=data['Loan_Amount_term'])

sns.swarmplot(data['Gender'],data['ApplicantIncome'],hug=data['Loan_Status'])
```

```
sc=StandardScaler()

x_bal=sc.fit_transform(x_bal)

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

x_train,x_test,y_train,y_test=train_test_split(x_bal,y_bal,test_size=0.33,random_s

def decisionTree(x_train,x_test,y_train,y_test)

dt=DecisionTreeClassifier()

dt.fit(x_train,y_train)

ypred=dt.predict(x_test)

print('***DecisionTreeClassfier***')

print('confusion matrix')

print(confusion_matrix(y_test,ypred))

print('Classificatin 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=f.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):


knn=KNeighboursClassifier()


, cross_val_score
, cross_val_predict
[ ] knn.fit(x_train,y_train)
ypred = knn.predict(x-test)
print('*****KNeighboursClassifer*****')
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('*****GradientBoostingclassifer*****')
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
claaifer = sequential()
classifer.add(Dense(units=100,activation='relu',input_dia=11))
classifer.add(Dense(units=50,activation='relu'))
classifer.adda(Dense(units=1, activation='sigmoid'))
classifer.compile(optimizer='adam', loss='binary_crossentropy',metrics=[accuracy])
model.history=classifer/.fit(x_train,y_train batch_size100,valdiation-split=0.2,ep
dtr=predict([[1,1,0,1,1,4276,1542,145,240,0,1]])
dtr.predict([[1,1,0,1,1,4276,1542,145,240,0,1]])
Knn.predict([[1,1,0,1,1,4276,1542,145,240,0,1]])
xgb.predict([[1,1,0,4276,1542,145,240,0,1]])
classifer.save("loan.h5")
y_pred = classifer.predict(x_test)
y_pred
y_pred = (y_pred > 0.5)
y_paccuracy_score
def predict_exit(sample_value):
  sample_value = np.array(sample_value)
  sample_value = sample_value.reshape(1,-1)
  sample_value = sc.transform(sample_value)
  return classifier.predict(sample_value)
  sample_value =[(1,1,0,1,1,4276,1542,145,240,0,1)]
  if predict_exit(sample_value)>0.5:
    print('Prediction:high chance of loan approval')
    else:
      print('prediction:low chance of loan approval')
```

```
        print('prediction:low chance of loan approval')
        sample_value = [(1,0,1,1,1,45,14,45,240,1,1)]
        if predict_exit(sample_value)>0.5:
        print('prediction:high chance of loan approval')
        else:
        print('prediction low chance of loan approval')
  def compareModel(x_train,x_test,y_train,y_test):
    decisionTree(x_train,x_test,y_train,y_test)
    print('-'*100)
    RandomForest(x_train,x_test,y_train,y_test)
    print('-'*100)
    XGB(x_train,x_test,y_train,y_test)
    print('-'*100)
    KNN(x_train,x_test,y_train,y_test)
    print('-'*100)
    comparemodel(x_train,x_test,y_train,y_test)
    ypred = classifier.predict(x_test)
    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))
    from sklearn.model_selection import cross_val_score
    rf = RandomForestClassifier()
    rf.fit(x_train,y_train)
    ypred = rf.predict(x_test)
    f1_score(ypred,y_test,average='weighted')
    cv=cross_val_score(rf,x,y,cs=5)
    np.mean(cv)
    pickle.dump(model,open('rdf,pk1','wb'))
    from flask import Flask,render_template,request
    import numpy as np
    import pickle
    app = Flask(__name__)
    model = pickle.load(open(r'rdf,pkl','rb))
    scale = pickle.load(open(r'scale1.pkl','rb'))
   def home():
     return render_template('home.html')
     def submit():
     input_feature=[int(x) for x n request.form.values() ]
     input_feature=[np.array(input_feature)]
     print (input_feature)
     names = ['Gender','Married','Dependents','Education','Self_Employed,'ApplicontI
     data=pandas.DataFrame(input_feature,xolumns=names)
     print(data)
     prediction=model.predict(data)
     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")
     if_name_=="_main_":
       port=int(os.environ.get('PORT',5000))
```

```
app.run(debug=False)
```