In [1]:
```
import pandas as pd
```

In [ ]:
```

```

In [2]:
```
data = pd.read_csv("C:\\Users\\yuvra\\Desktop\\loan_prediction.csv")
```

In [3]:
```
data.head()
```

Out[3]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapplic
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 [4]:
```
data.tail()
```

Out[4]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapp
609	LP002978	Female	No	0	Graduate	No	2900	
610	LP002979	Male	Yes	3+	Graduate	No	4106	
611	LP002983	Male	Yes	1	Graduate	No	8072	
612	LP002984	Male	Yes	2	Graduate	No	7583	
613	LP002990	Female	No	0	Graduate	Yes	4583	

In [5]:
```
data.shape
```

Out[5]:
```
(614, 13)
```

In [6]:
```
print("No. of Rows",data.shape[0])
print("No. of Columns",data.shape[1])
```

```
No. of Rows 614
No. of Columns 13
```

In [7]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Loan_ID            614 non-null    object 
 1   Gender             601 non-null    object 
 2   Married            611 non-null    object 
 3   Dependents         599 non-null    object 
 4   Education          614 non-null    object 
 5   Self_Employed      582 non-null    object 
 6   ApplicantIncome    614 non-null    int64  
 7   CoapplicantIncome  614 non-null    float64
 8   LoanAmount         592 non-null    float64
 9   Loan_Amount_Term   600 non-null    float64
 10  Credit_History     564 non-null    float64
 11  Property_Area      614 non-null    object 
 12  Loan_Status        614 non-null    object 
dtypes: float64(4), int64(1), object(8)
memory usage: 62.5+ KB
```

In [8]:
```
data.isnull().sum()
```

Out[8]:
```
Loan_ID               0
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]:
```
data.isnull().sum()*100 / len(data)
```

Out[9]:
```
Loan_ID              0.000000
Gender               2.117264
Married              0.488599
Dependents           2.442997
Education            0.000000
Self_Employed        5.211726
ApplicantIncome      0.000000
CoapplicantIncome    0.000000
LoanAmount           3.583062
Loan_Amount_Term     2.280130
Credit_History       8.143322
Property_Area        0.000000
Loan_Status          0.000000
dtype: float64
```

In [10]:
```
data = data.drop('Loan_ID',axis=1)
```

In [11]:
```
data.head(1)
```

Out[11]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome
0	Male	No	0	Graduate	No	5849	0.0

In [12]:
```
columns = ['Gender','Dependents','LoanAmount','Loan_Amount_Term']
```

In [13]:
```
data = data.dropna(subset=columns)
```

In [14]:
```
data.isnull().sum()*100 / len(data)
```

Out[14]:
```
Gender               0.000000
Married              0.000000
Dependents           0.000000
Education            0.000000
Self_Employed        5.424955
ApplicantIncome      0.000000
CoapplicantIncome    0.000000
LoanAmount           0.000000
Loan_Amount_Term     0.000000
Credit_History       8.679928
Property_Area        0.000000
Loan_Status          0.000000
dtype: float64
```

In [15]:
```
data['Self_Employed'].mode()[0]
```

Out[15]:
```
'No'
```

In [16]:
```
data['Self_Employed'] =data['Self_Employed'].fillna(data['Self_Employed'].mode()[0
```

In [17]:
```
data.isnull().sum()*100 / len(data)
```

Out[17]:
```
Gender               0.000000
Married              0.000000
Dependents           0.000000
Education            0.000000
Self_Employed        0.000000
ApplicantIncome      0.000000
CoapplicantIncome    0.000000
LoanAmount           0.000000
Loan_Amount_Term     0.000000
Credit_History       8.679928
Property_Area        0.000000
Loan_Status          0.000000
dtype: float64
```

In [18]:
```
data['Gender'].unique()
```

Out[18]:
```
array(['Male', 'Female'], dtype=object)
```

In [19]:
```
data['Self_Employed'].unique()
```

Out[19]:
```
array(['No', 'Yes'], dtype=object)
```

In [20]:
```
data['Credit_History'].mode()[0]
```

Out[20]:
```
1.0
```

In [21]:
```
data['Credit_History'] =data['Credit_History'].fillna(data['Credit_History'].mode(
```

In [22]:
```
data.isnull().sum()*100 / len(data)
```

Out[22]:
```
Gender               0.0
Married              0.0
Dependents           0.0
Education            0.0
Self_Employed        0.0
ApplicantIncome      0.0
CoapplicantIncome    0.0
LoanAmount           0.0
Loan_Amount_Term     0.0
Credit_History       0.0
Property_Area        0.0
Loan_Status          0.0
dtype: float64
```

In [23]:
```
data.sample(5)
```

Out[23]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncon
527	Male	Yes	1	Not Graduate	No	5285	1430
123	Male	Yes	2	Graduate	No	2957	0
290	Male	Yes	0	Graduate	No	3075	2416
300	Male	Yes	0	Not Graduate	No	1800	2934
186	Male	Yes	1	Graduate	Yes	2178	0

In [24]:
```
data['Dependents'] =data['Dependents'].replace(to_replace="3+",value='4')
```

In [25]:
```
data['Dependents'].unique()
```

Out[25]:
```
array(['1', '0', '2', '4'], dtype=object)
```

In [26]:
```
data['Loan_Status'].unique()
```

Out[26]:
```
array(['N', 'Y'], dtype=object)
```

In [27]:
```
data['Gender'] = data['Gender'].map({'Male':1,'Female':0}).astype('int')
data['Married'] = data['Married'].map({'Yes':1,'No':0}).astype('int')
data['Education'] = data['Education'].map({'Graduate':1,'Not Graduate':0}).astype(
data['Self_Employed'] = data['Self_Employed'].map({'Yes':1,'No':0}).astype('int')
data['Property_Area'] = data['Property_Area'].map({'Rural':0,'Semiurban':2,'Urban'
data['Loan_Status'] = data['Loan_Status'].map({'Y':1,'N':0}).astype('int')
```

In [28]:
```
data.head()
```

Out[28]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome
1	1	1	1	1	0	4583	1508.0
2	1	1	0	1	1	3000	0.0
3	1	1	0	0	0	2583	2358.0
4	1	0	0	1	0	6000	0.0
5	1	1	2	1	1	5417	4196.0

In [29]:

```
X = data.drop('Loan_Status',axis=1)
```

In [30]:

```
y = data['Loan_Status']
```

In [31]:

```
y
```

Out[31]:

```
1      0
2      1
3      1
4      1
5      1
      ..
609    1
610    1
611    1
612    1
613    0
Name: Loan_Status, Length: 553, dtype: int32
```

In [32]:

```
data.head()
```

Out[32]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome
1	1	1	1	1	0	4583	1508.0
2	1	1	0	1	1	3000	0.0
3	1	1	0	0	0	2583	2358.0
4	1	0	0	1	0	6000	0.0
5	1	1	2	1	1	5417	4196.0

In [33]:

```
cols = ['ApplicantIncome','CoapplicantIncome','LoanAmount','Loan_Amount_Term']
```

In [34]:

```
from sklearn.preprocessing import StandardScaler
st = StandardScaler()
X[cols]=st.fit_transform(X[cols])
```

In [35]:

```
X
```

Out[35]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncon
1	1	1	1	1	0	-0.128694	-0.0496
2	1	1	0	1	1	-0.394296	-0.5456
3	1	1	0	0	0	-0.464262	0.2298
4	1	0	0	1	0	0.109057	-0.5456
5	1	1	2	1	1	0.011239	0.8343
...	...	...	...	...	...	...	
609	0	0	0	1	0	-0.411075	-0.5456
610	1	1	4	1	0	-0.208727	-0.5456
611	1	1	1	1	0	0.456706	-0.4667
612	1	1	2	1	0	0.374659	-0.5456
613	0	0	0	1	1	-0.128694	-0.5456

553 rows × 11 columns

In [36]:

```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score
import numpy as np
```

In [37]:

```
model_df={}
def model_val(model,X,y):
    X_train,X_test,y_train,y_test=train_test_split(X,y,
                                                   test_size=0.20,
                                                   random_state=42)
    model.fit(X_train,y_train)
    y_pred=model.predict(X_test)
    print(f"{model} accuracy is {accuracy_score(y_test,y_pred)}")

    score = cross_val_score(model,X,y,cv=5)
    print(f"{model} Avg cross val score is {np.mean(score)}")
    model_df[model]=round(np.mean(score)*100,2)
```

In [38]:

```
model_df
```

Out[38]:

```
{}
```

In [39]:

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model_val(model,X,y)
```

```
LogisticRegression() accuracy is 0.8018018018018018
LogisticRegression() Avg cross val score is 0.8047829647829647
```

In [40]:

```
from sklearn import svm
model = svm.SVC()
model_val(model,X,y)
```

```
SVC() accuracy is 0.7927927927927928
SVC() Avg cross val score is 0.7938902538902539
```

In [41]:
```
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier()
model_val(model,X,y)
```

```
DecisionTreeClassifier() accuracy is 0.7657657657657657
DecisionTreeClassifier() Avg cross val score is 0.716101556101556
```

In [42]:
```
from sklearn.ensemble import RandomForestClassifier
model =RandomForestClassifier()
model_val(model,X,y)
```

```
RandomForestClassifier() accuracy is 0.7657657657657657
RandomForestClassifier() Avg cross val score is 0.7885012285012285
```

In [43]:
```
from sklearn.ensemble import GradientBoostingClassifier
model =GradientBoostingClassifier()
model_val(model,X,y)
```

```
GradientBoostingClassifier() accuracy is 0.7927927927927928
GradientBoostingClassifier() Avg cross val score is 0.774004914004914
```

In [44]:
```
from sklearn.model_selection import RandomizedSearchCV
```

In [45]:
```
log_reg_grid={"C":np.logspace(-4,4,20),
             "solver":['liblinear']}
```

In [46]:
```
rs_log_reg=RandomizedSearchCV(LogisticRegression(),
                   param_distributions=log_reg_grid,
                  n_iter=20,cv=5,verbose=True)
```

In [47]:
```
rs_log_reg.fit(X,y)
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
```

Out[47]:
```
RandomizedSearchCV(cv=5, estimator=LogisticRegression(), n_iter=20,
                   param_distributions={'C': array([1.00000000e-04, 2.63665090e-0
4, 6.95192796e-04, 1.83298071e-03,
       4.83293024e-03, 1.27427499e-02, 3.35981829e-02, 8.85866790e-02,
       2.33572147e-01, 6.15848211e-01, 1.62377674e+00, 4.28133240e+00,
       1.12883789e+01, 2.97635144e+01, 7.84759970e+01, 2.06913808e+02,
       5.45559478e+02, 1.43844989e+03, 3.79269019e+03, 1.00000000e+04]),
                                        'solver': ['liblinear']},
                   verbose=True)
```

In [48]:
```
rs_log_reg.best_score_
```

Out[48]:
```
0.8047829647829647
```

In [49]:
```
rs_log_reg.best_params_
```

Out[49]:
```
{'solver': 'liblinear', 'C': 0.23357214690901212}
```

In [50]:
```
svc_grid = {'C':[0.25,0.50,0.75,1],"kernel":["linear"]}
```

In [51]:
```
rs_svc=RandomizedSearchCV(svm.SVC(),
                  param_distributions=svc_grid,
                   cv=5,
                   n_iter=20,
                  verbose=True)
```

In [52]:
```
rs_svc.fit(X,y)
```

```
Fitting 5 folds for each of 4 candidates, totalling 20 fits
```

```
C:\Users\yuvra\anaconda3\lib\site-packages\sklearn\model_selection\_search.py:292:
UserWarning: The total space of parameters 4 is smaller than n_iter=20. Running 4
iterations. For exhaustive searches, use GridSearchCV.
  warnings.warn(
```

Out[52]:
```
RandomizedSearchCV(cv=5, estimator=SVC(), n_iter=20,
                   param_distributions={'C': [0.25, 0.5, 0.75, 1],
                                        'kernel': ['linear']},
                   verbose=True)
```

In [53]:
```
rs_svc.best_score_
```

Out[53]:
```
0.8066011466011467
```

In [54]:
```
rs_svc.best_params_
```

Out[54]:
```
{'kernel': 'linear', 'C': 0.25}
```

In [55]:
```
RandomForestClassifier()
```

Out[55]:
```
RandomForestClassifier()
```

In [56]:
```
rf_grid={'n_estimators':np.arange(10,1000,10),
  'max_features':['auto','sqrt'],
 'max_depth':[None,3,5,10,20,30],
 'min_samples_split':[2,5,20,50,100],
 'min_samples_leaf':[1,2,5,10]
 }
```

In [57]:
```
rs_rf=RandomizedSearchCV(RandomForestClassifier(),
                  param_distributions=rf_grid,
                   cv=5,
                   n_iter=20,
                  verbose=True)
```

In [ ]:

In [58]:
```
rs_rf.fit(X,y)
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
```

Out[58]:
```
RandomizedSearchCV(cv=5, estimator=RandomForestClassifier(), n_iter=20,
                   param_distributions={'max_depth': [None, 3, 5, 10, 20, 30],
                                        'max_features': ['auto', 'sqrt'],
                                        'min_samples_leaf': [1, 2, 5, 10],
                                        'min_samples_split': [2, 5, 20, 50,
                                                              100],
                                        'n_estimators': array([ 10,  20,  30,  40,
50,  60,  70,  80,  90, 100, 110, 120, 130,
       140, 150, 160, 170, 180, 190, 200, 210, 220, 230, 240, 250, 260,
       270, 280, 290, 300, 310, 320, 330, 340, 350, 360, 370, 380, 390,
       400, 410, 420, 430, 440, 450, 460, 470, 480, 490, 500, 510, 520,
       530, 540, 550, 560, 570, 580, 590, 600, 610, 620, 630, 640, 650,
       660, 670, 680, 690, 700, 710, 720, 730, 740, 750, 760, 770, 780,
       790, 800, 810, 820, 830, 840, 850, 860, 870, 880, 890, 900, 910,
       920, 930, 940, 950, 960, 970, 980, 990])},
                   verbose=True)
```

In [59]:
```
rs_rf.best_score_
```

Out[59]:
```
0.8066011466011467
```

In [60]:
```
rs_rf.best_params_
```

Out[60]:
```
{'n_estimators': 900,
 'min_samples_split': 50,
 'min_samples_leaf': 5,
 'max_features': 'auto',
 'max_depth': 20}
```

In [61]:
```
X = data.drop('Loan_Status',axis=1)
y = data['Loan_Status']
```

In [62]:
```
rf = RandomForestClassifier(n_estimators=270,
 min_samples_split=5,
 min_samples_leaf=5,
 max_features='sqrt',
 max_depth=5)
```

In [63]:
```
rf.fit(X,y)
```

Out[63]:
```
RandomForestClassifier(max_depth=5, max_features='sqrt', min_samples_leaf=5,
                       min_samples_split=5, n_estimators=270)
```

In [64]:
```
import joblib
```

In [65]:
```
joblib.dump(rf,'loan_status_predict')
```

Out[65]:
```
['loan_status_predict']
```

In [66]:
```
model = joblib.load('loan_status_predict')
```

In [67]:
```
import pandas as pd
df = pd.DataFrame({
    'Gender':1,
    'Married':1,
    'Dependents':2,
    'Education':0,
    'Self_Employed':0,
    'ApplicantIncome':2889,
    'CoapplicantIncome':0.0,
    'LoanAmount':45,
    'Loan_Amount_Term':180,
    'Credit_History':0,
    'Property_Area':1
},index=[0])
```

In [68]:
```
df
```

Out[68]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome
0	1	1	2	0	0	2889	0.0

In [69]:
```
result = model.predict(df)
```

In [70]:
```
if result==1:
    print("Loan Approved")
else:
    print("Loan Not Approved")
```

```
Loan Not Approved
```

In [71]:

```
from tkinter import *
import joblib
import pandas as pd
```

In [72]:

```
def show_entry():

    p1 = float(e1.get())
    p2 = float(e2.get())
    p3 = float(e3.get())
    p4 = float(e4.get())
    p5 = float(e5.get())
    p6 = float(e6.get())
    p7 = float(e7.get())
    p8 = float(e8.get())
    p9 = float(e9.get())
    p10 = float(e10.get())
    p11 = float(e11.get())

    model = joblib.load('loan_status_predict')
    df = pd.DataFrame({
    'Gender':p1,
    'Married':p2,
    'Dependents':p3,
    'Education':p4,
    'Self_Employed':p5,
    'ApplicantIncome':p6,
    'CoapplicantIncome':p7,
    'LoanAmount':p8,
    'Loan_Amount_Term':p9,
    'Credit_History':p10,
    'Property_Area':p11
},index=[0])
    result = model.predict(df)

    if result == 1:
        Label(master, text="Loan approved").grid(row=31)
    else:
        Label(master, text="Loan Not Approved").grid(row=31)



master =Tk()
master.title("Loan Status Prediction Using Machine Learning")
label = Label(master,text = "Loan Status Prediction",bg = "black",
               fg = "white").grid(row=0,columnspan=2)

Label(master,text = "Gender [1:Male ,0:Female]").grid(row=1)
Label(master,text = "Married [1:Yes,0:No]").grid(row=2)
Label(master,text = "Dependents [1,2,3,4]").grid(row=3)
Label(master,text = "Education").grid(row=4)
Label(master,text = "Self_Employed").grid(row=5)
Label(master,text = "ApplicantIncome").grid(row=6)
Label(master,text = "CoapplicantIncome").grid(row=7)
Label(master,text = "LoanAmount").grid(row=8)
Label(master,text = "Loan_Amount_Term").grid(row=9)
Label(master,text = "Credit_History").grid(row=10)
Label(master,text = "Property_Area").grid(row=11)


e1 = Entry(master)
e2 = Entry(master)
e3 = Entry(master)
e4 = Entry(master)
```

```
e5 = Entry(master)
e6 = Entry(master)
e7 = Entry(master)
e8 = Entry(master)
e9 = Entry(master)
e10 = Entry(master)
e11 = Entry(master)



e1.grid(row=1,column=1)
e2.grid(row=2,column=1)
e3.grid(row=3,column=1)
e4.grid(row=4,column=1)
e5.grid(row=5,column=1)
e6.grid(row=6,column=1)
e7.grid(row=7,column=1)
e8.grid(row=8,column=1)
e9.grid(row=9,column=1)
e10.grid(row=10,column=1)
e11.grid(row=11,column=1)

Button(master,text="Predict",command=show_entry).grid()

mainloop()
```

In [ ]: