Home Lone Approal Prediction

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```

- Data Description
  - Loan_ID--------------> Unique Loan ID.
  - Gender --------------> Male/ Female
  - Married --------------> Applicant married (Y/N)
  - Dependents ------------> Number of dependents
  - Education -------------> Applicant Education (Graduate/ Under Graduate)
  - Self_Employed ---------> Self-employed (Y/N)
  - ApplicantIncome -------> Applicant income
  - CoapplicantIncome -----> Coapplicant income
  - LoanAmount -----------> Loan amount in thousands
  - Loan_Amount_Term ------> Term of a loan in months
  - Credit_History --------> Credit history meets guidelines
  - Property_Area ---------> Urban/ Semi-Urban/ Rural
  - Loan_Status -----------> Loan approved (Y/N)

- We are working for a bank to make the autopilot system to check weather the applicant is eligibile for Home loan or not?.
  - We have 2 sets of data
  - for Traing we will use train data
  - for testing we will use testing data.

In [2]:
```
df = pd.read_csv('loan_ele_train.csv')  # Target = Loan_Status
```

In [3]:
```
df.head(5)
```

Out[3]:

ried	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Am
No	0	Graduate	No	5849	0.0	NaN	
Yes	1	Graduate	No	4583	1508.0	128.0	
Yes	0	Graduate	Yes	3000	0.0	66.0	
Yes	0	Not Graduate	No	2583	2358.0	120.0	
No	0	Graduate	No	6000	0.0	141.0	

In [4]:
```
df.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 [5]:
```
df.describe()
```

Out[5]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	614.000000	614.000000	592.000000	600.00000	564.000000
mean	5403.459283	1621.245798	146.412162	342.00000	0.842199
std	6109.041673	2926.248369	85.587325	65.12041	0.364878
min	150.000000	0.000000	9.000000	12.00000	0.000000
25%	2877.500000	0.000000	100.000000	360.00000	1.000000
50%	3812.500000	1188.500000	128.000000	360.00000	1.000000
75%	5795.000000	2297.250000	168.000000	360.00000	1.000000
max	81000.000000	41667.000000	700.000000	480.00000	1.000000

In [6]:
```
df.shape
```

Out[6]:
```
(614, 13)
```

In [7]:
```
df.isnull().sum()
```

Out[7]:
```
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
```

- we will drop the Null values form the column Loan Amount as it is mandatory feild and it should not be blank.
- Rest all bank values we will replace.

In [8]:
```
df['Gender'].value_counts()
```

Out[8]:
```
Male      489
Female    112
Name: Gender, dtype: int64
```

In [9]:
```
df['Gender'].ffill(inplace=True)    # replace Nan values using Forward filling
```

In [10]:
```
df['Gender'].value_counts()
```

Out[10]:
```
Male      500
Female    114
Name: Gender, dtype: int64
```

In [11]:
```
# ffilling for Married columns as well.

df['Married'].ffill(inplace=True)
```

In [12]:
```
df['Dependents'].value_counts()
```

Out[12]:
```
0     345
1     102
2     101
3+     51
Name: Dependents, dtype: int64
```

- for Dependents column
  - we will remove + sign and make it 3 dependents
  - change the datatpes to int64

In [13]:
```
df['Dependents'] = df['Dependents'].str.replace('+','')   # replaced '+' with ''
```

In [14]:
```
df['Dependents'].ffill(inplace=True)    # forward replaced Nan values
```

In [15]:
```
df['Dependents'] = df['Dependents'].astype('int64')    # chnaged datatype to int64.
```

In [16]:
```
df['Self_Employed'].value_counts()
```

Out[16]:
```
No     500
Yes     82
Name: Self_Employed, dtype: int64
```

In [17]:
```
df['Self_Employed'].ffill(inplace=True)   # forward filling
```

In [18]:
```
# for Loan Amount column we will replace with Mean.

M = df['LoanAmount'].mean()
mm=(round(M,0))
```

In [19]:
```
df['LoanAmount'].fillna(mm,inplace=True)      # replaced Null values with Mean
```

In [20]:
```
A = df['Loan_Amount_Term'].mean()
aa = (round(A,0))
```

In [21]:
```
df['Loan_Amount_Term'].fillna(aa,inplace=True)    # replaced Null values with Average
```

In [22]:
```
df['Credit_History'].value_counts()
```

Out[22]:
```
1.0    475
0.0     89
Name: Credit_History, dtype: int64
```

In [23]:
```
df['Credit_History'].ffill(inplace=True)   # replaced Null values
```

In [24]:
```
# lets check Null values.

df.isnull().sum()
```

Out[24]:
```
Loan_ID              0
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
```

- We will drop Loan_ID as it contain unique ID.
- we will replace categorical columns to numeric.

In [25]:
```
df.drop(['Loan_ID'],axis=1,inplace=True)     # droped Loan_ID column
```

In [26]:
```
df.columns
```

Out[26]:
```
Index(['Gender', 'Married', 'Dependents', 'Education', 'Self_Employed',
       'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History', 'Property_Area', 'Loan_Status'],
      dtype='object')
```

In [27]:
```
df['Gender'].replace('Male',1,inplace=True)          # replaed values with numerics
df['Gender'].replace('Female',0,inplace=True)
```

In [28]:
```
df['Married'].replace('Yes',1,inplace=True              # replaed values with numerics
df['Married'].replace('No',0,inplace=True)
```

```
  File "C:\Users\DCINCE-Yateen\AppData\Local\Temp\ipykernel_11624\88675463.py", lin
e 2
    df['Married'].replace('No',0,inplace=True)
    ^
SyntaxError: invalid syntax
```

In [ ]:
```
df['Education'].replace('Graduate',1,inplace=True)
df['Education'].replace('Not Graduate',0,inplace=True)
```

In [ ]:
```
df['Self_Employed'].replace('Yes',1,inplace=True)
df['Self_Employed'].replace('No',0,inplace=True)
```

In [ ]:
```
df.info()
```

In [ ]:
```
df['Property_Area'].replace('Urban',0,inplace=True)
df['Property_Area'].replace('Rural',1,inplace=True)
df['Property_Area'].replace('Semiurban',2,inplace=True)
```

In [ ]:
```
df['Loan_Status'].replace('Y',1,inplace=True)
df['Loan_Status'].replace('N',0,inplace=True)
```

In [ ]:
```
df.info()        # Now All columns have Numerical data.
```

In [ ]:
```
df
```

- Lets Segregate data into x and y

In [ ]:
```
x = df.drop('Loan_Status',axis=1)      # segregated features from target
y = df['Loan_Status']                 # segregated Target from feature
```

In [ ]:
```
x
```

In [ ]:
```
y
```

Scaling the feature

In [ ]:
```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x=sc.fit_transform(x)
```

In [ ]:
```
x
```

lets build the model

In [ ]:
```

```

```
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest = train_test_split(x,y,test_size=0.3,random_state=1)
```

Importing classification model

In [ ]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import LinearSVC
from sklearn.svm import SVC
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import AdaBoostClassifier

# Metrics
from sklearn.metrics import precision_score,recall_score,accuracy_score
```

- Lets fit the data into the models

In [ ]:
```
lr = LogisticRegression()
lr.fit(xtrain,ytrain)

dtc = DecisionTreeClassifier()
dtc.fit(xtrain,ytrain)

rf = RandomForestClassifier()
rf.fit(xtrain,ytrain)

lsvc = LinearSVC()
lsvc.fit(xtrain,ytrain)

svc = SVC()
svc.fit(xtrain,ytrain)

gb = GradientBoostingClassifier()
gb.fit(xtrain,ytrain)

ab = AdaBoostClassifier()
ab.fit(xtrain,ytrain)
```

In [ ]:
```
# Lets predict the data

ypred1 = lr.predict(xtest)  # testing LogisticRegression

ypred2 = dtc.predict(xtest) # testing DecisionTreeClassifier

ypred3 = rf.predict(xtest)   # testing RandomForestClassifier

ypred4 = lsvc.predict(xtest)  # testing LinearSVC

ypred5 = svc.predict(xtest)   # testing SVC

ypred6 = gb.predict(xtest)    # testing GradientBoostingClassifier

ypred7 = ab.predict(xtest)   # AdaBoostClassifier
```

In [ ]:
```
# Lets compare the Origianl value with predited value

df1 = pd.DataFrame({'Original val': ytest, 'Logistic Regression': ypred1, 'Decision T
```

In [ ]:
```
df1
```

In [ ]:
```
# lets compare the data visually

plt.figure(figsize=(10,13))

plt.subplot(441)
plt.plot(df1['Original val'].iloc[0:10],label='Original val')
plt.legend()

plt.subplot(442)
plt.plot(df1['Original val'].iloc[0:10],label='Original val')
plt.plot(df1['Logistic Regression'].iloc[0:10],label='Logistic Regression')
plt.legend()

plt.subplot(443)
plt.plot(df1['Original val'].iloc[0:10],label='Original val')
plt.plot(df1['Decision Tree'].iloc[0:10],label='Decision Tree')
plt.legend()

plt.subplot(444)
plt.plot(df1['Original val'].iloc[0:10],label='Original val')
plt.plot(df1['Random Forest'].iloc[0:10],label='Random Forest')
plt.legend()

plt.subplot(445)
plt.plot(df1['Original val'].iloc[0:10],label='Original val')
plt.plot(df1['LinearSVC'].iloc[0:10],label='LinearSVC')
plt.legend()

plt.subplot(446)
plt.plot(df1['Original val'].iloc[0:10],label='Original val')
plt.plot(df1['SVC'].iloc[0:10],label='SVC')
plt.legend()

plt.subplot(447)
plt.plot(df1['Original val'].iloc[0:10],label='Original val')
plt.plot(df1['Gradient Boosting'].iloc[0:10],label='Gradient Boosting')
plt.legend()

plt.subplot(448)
plt.plot(df1['Original val'].iloc[0:10],label='Original val')
plt.plot(df1['Adaboost'].iloc[0:10],label='Adaboost')
plt.legend()

plt.tight_layout()
```

- Observation
  - from the above data we can see that the Adaboosting Model is predecting almost perfect values.
  - lets check the Accuracy score,Recall_score,Precision score for the same.

In [ ]:
```
# Accuracy_score

score1 = accuracy_score(ytest,ypred1)
score2 = accuracy_score(ytest,ypred2)
score3 = accuracy_score(ytest,ypred3)
score4 = accuracy_score(ytest,ypred4)
score5 = accuracy_score(ytest,ypred5)
score6 = accuracy_score(ytest,ypred6)
score7 = accuracy_score(ytest,ypred7)

print(score1,score2,score3,score4,score5,score6,score7)
```

In [ ]:
```
# Precision Score  - this shows which model have predicted high true positive values

p1 = precision_score(ytest,ypred1)
p2 = precision_score(ytest,ypred2)
p3 = precision_score(ytest,ypred3)
p4 = precision_score(ytest,ypred4)
p5 = precision_score(ytest,ypred5)
p6 = precision_score(ytest,ypred6)
p7 = precision_score(ytest,ypred7)

print(p1,p2,p3,p4,p5,p6,p7)
```

In [ ]:
```
# Recall Score - this indicates how many of the actual positives were correctly predi

r1 = recall_score(ytest,ypred1)
r2 = recall_score(ytest,ypred2)
r3 = recall_score(ytest,ypred3)
r4 = recall_score(ytest,ypred4)
r5 = recall_score(ytest,ypred5)
r6 = recall_score(ytest,ypred6)
r7= recall_score(ytest,ypred7)

print(r1,r2,r3,r4,r5,r6,r7)
```

- The highest Accuracy = 80% - with Gradient Boosting Model

In [ ]:
```
# lets test the model with new data.

data = {'Gender':1,
       'Married':0,
       'Dependents':0,
       'Education':1,
       'Self_Employed':1,
       'ApplicantIncome': 10000,
       'CoapplicantIncome':5700,
       'LoanAmount in thousand':150,
       'Loan_Amount_Term':200,
       'Credit_History':0,
       'Property_Area':2}

aa = pd.DataFrame(data,index=[0])
aa
```

In [ ]:
```
new_data = ab.predict(aa)      # predicting the new data.
print(new_data)
```

In [ ]:
```
# Lets save the model with Joblib

ab = AdaBoostClassifier()
ab.fit(x,y)
```

In [ ]:
```
import joblib
joblib.dump(ab,'model_joblib_ab')   # model saving
```

In [ ]:
```
model = joblib.load('model_joblib_ab')    # model loading
```

In [ ]:
```
model.predict(aa)
```

- here we can see that the trained model and the newly saved model with entire data
- showing same output. = 0

In [ ]:
```
# lets create basic GUI for the model to predict the output
```

In [35]:
```
from tkinter import *
import joblib

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('model_joblib_ab')   # Loading the model
    result = model.predict([[p1, p2, p3, p4, p5, p6, p7, p8, p9, p10, p11]])

    Label(master, text="Check your Home Loan Eligibility").grid(row=12, columnspan=2)

    if result[0] == 0:
        eligibility_label = Label(master, text="Sorry you are not Eligible for Home l
    elif result[0] == 1:
        eligibility_label = Label(master, text="Congratulations!!! You are eligible f
    else:
        eligibility_label = Label(master, text="Invalid prediction result")

    eligibility_label.grid(row=13, columnspan=2)

master = Tk()
master.title("HOME LOAN ELIGIBILITY")

label = Label(master, text="HOME LOAN ELIGIBILITY", bg='black', fg='white')
label.grid(row=0, columnspan=2)

# Labels for input fields
labels = ["Gender M=1/F=0", "Married M=1/Um=0", "Dependents", "Education (1/0)",
          "Self_Employed (1/0)", "ApplicantIncome", "CoapplicantIncome",
          "LoanAmount in thousand", "Loan_Amount_Term", "Credit_History (1/0)",
          "Property_Area (U=0/R=1/SMi=2)"]

for i, text in enumerate(labels):
    Label(master, text=text).grid(row=i+1, column=0)

# Entry fields
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)

# Positioning entry fields
entries = [e1, e2, e3, e4, e5, e6, e7, e8, e9, e10, e11]

for i, entry in enumerate(entries):
    entry.grid(row=i+1, column=1)

# Check Loan Eligibility Button
```

```
Button(master, text="Check Loan Eligibility", command=show_entry).grid(row=12, columr

mainloop()
```

In [ ]:

```

```