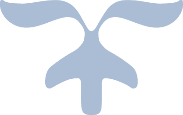


major project

ML-MAJOR-AUGUST-ML-08-ML01



*LOAN APPROVAL CLASSIFICATION USING ML*

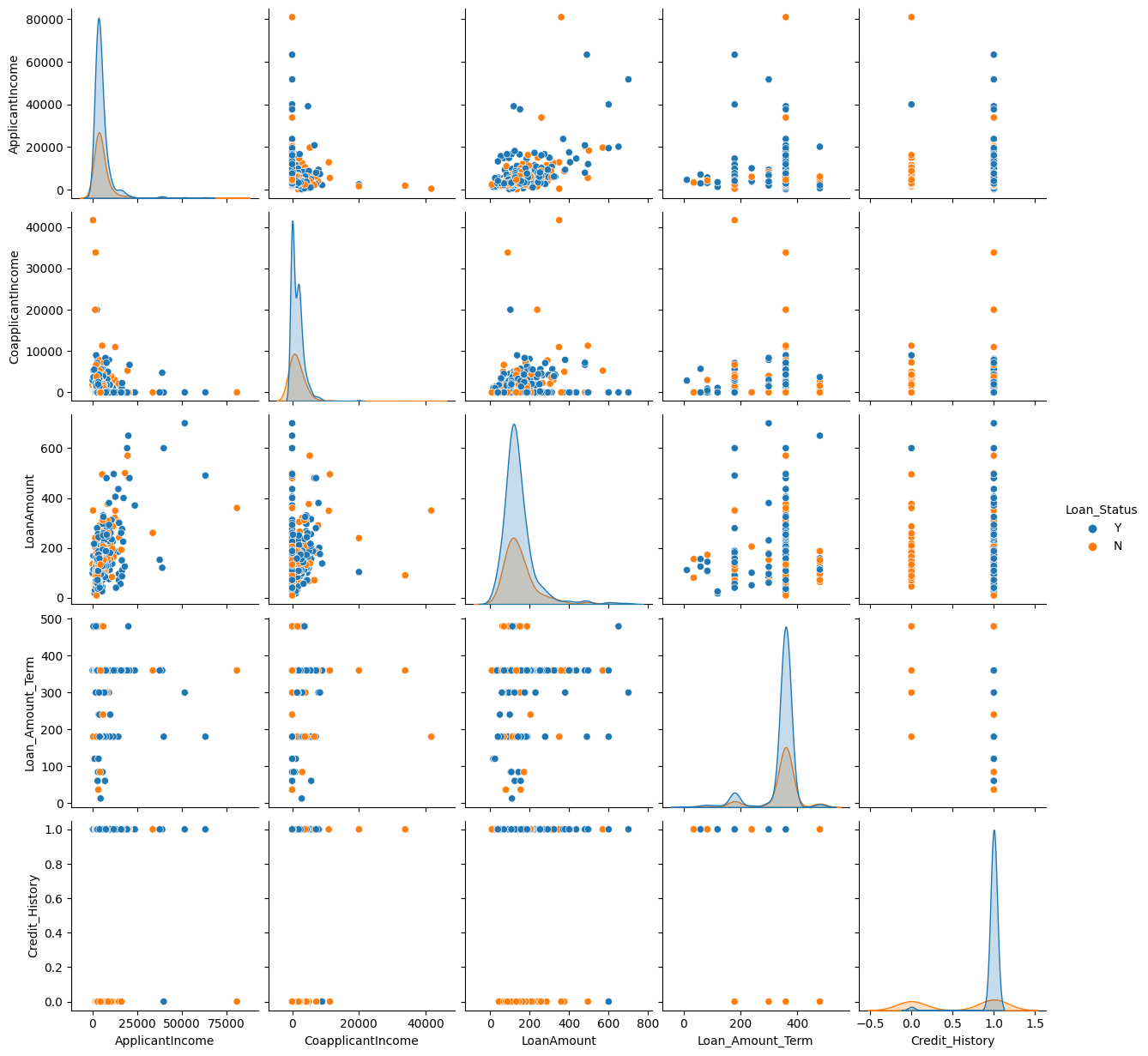
JUPYTER\_NOTEBOOK:

import pandas as pd #Importing the essential modules that we need  
import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.preprocessing import LabelEncoder,StandardScaler #LabelEncoder is used for encoding categorical variables into numerical values, and StandardScaler is used for standardizing features by removing the mean and scaling to unit variance.  
from sklearn.model\_selection import train\_test\_split #This imports a function from scikit-learn that allows you to split a dataset into training and testing sets. This is crucial for evaluating the performance of a machine learning model.  
from sklearn.metrics import accuracy\_score,f1\_score,confusion\_matrix,classification\_report #imports various metrics from scikit-learn that are commonly used for evaluating classification models. accuracy\_score, f1\_score, confusion\_matrix, and classification\_report provide different ways to assess model performance.  
import seaborn as sns  
%matplotlib inline  
#to display matplotlib plots inline within the notebook.

dataset=pd.read\_excel("/content/loan-predictionUC.csv.xlsx")  
dataset.head() #first few rows of the dataset

Loan\_ID Gender Married Dependents Education Self\_Employed \  
0 LP001002 Male No 0 Graduate No   
1 LP001003 Male Yes 1 Graduate No   
2 LP001005 Male Yes 0 Graduate Yes   
3 LP001006 Male Yes 0 Not Graduate No   
4 LP001008 Male No 0 Graduate No   
  
 ApplicantIncome CoapplicantIncome LoanAmount Loan\_Amount\_Term \  
0 5849 0.0 NaN 360.0   
1 4583 1508.0 128.0 360.0   
2 3000 0.0 66.0 360.0   
3 2583 2358.0 120.0 360.0   
4 6000 0.0 141.0 360.0   
  
 Credit\_History Property\_Area Loan\_Status   
0 1.0 Urban Y   
1 1.0 Rural N   
2 1.0 Urban Y   
3 1.0 Urban Y   
4 1.0 Urban Y

df=sns.pairplot(dataset,hue="Loan\_Status") #pairplotting so as to study the relation between any 2 cells



dataset.shape #It returns a tuple representing the number of rows and columns in your dataset.

(614, 13)

dataset.describe() #It provides a summary of the basic statistical measures of the dataset. This can include things like mean, standard deviation, minimum, maximum, and quartiles for numerical columns.

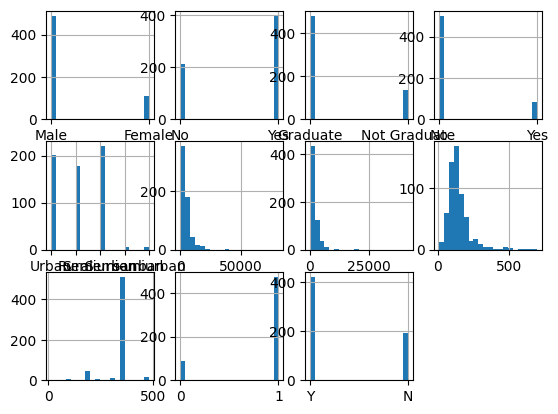
ApplicantIncome CoapplicantIncome LoanAmount Loan\_Amount\_Term \  
count 614.000000 614.000000 592.000000 600.00000   
mean 5403.459283 1621.245798 146.412162 342.00000   
std 6109.041673 2926.248369 85.587325 65.12041   
min 150.000000 0.000000 9.000000 12.00000   
25% 2877.500000 0.000000 100.000000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

Credit\_History   
count 564.000000   
mean 0.842199   
std 0.364878   
min 0.000000   
25% 1.000000   
50% 1.000000   
75% 1.000000   
max 1.000000

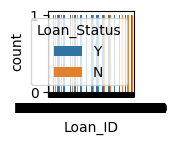
dataset.info()

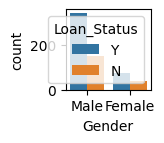
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 614 entries, 0 to 613  
Data columns (total 13 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 Loan\_ID 614 non-null object   
 1 Gender 601 non-null object   
 2 Married 611 non-null object   
 3 Dependents 599 non-null 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

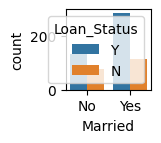
for i,j in enumerate(("Gender","Married","Education","Self\_Employed","Property\_Area","ApplicantIncome","CoapplicantIncome","LoanAmount","Loan\_Amount\_Term","Credit\_History","Loan\_Status")):  
 plt.subplot(3,4,i+1)  
 dataset[j].hist(bins=20) #enumerate returns 2 values index and value

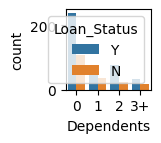


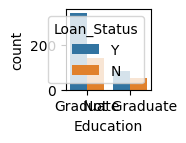
for i,x in enumerate(dataset):  
 if dataset[x].dtype=="object":  
 plt.subplot(4,5,i+1)  
 sns.countplot(data=dataset,x=x,hue="Loan\_Status")  
 else:  
 sns.displot(data=dataset,x=x,hue="Loan\_Status",kde=True)  
 plt.show()

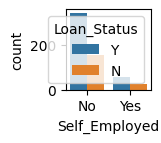


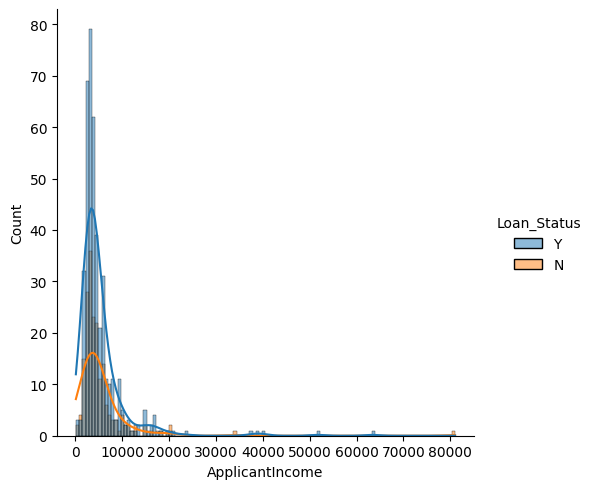


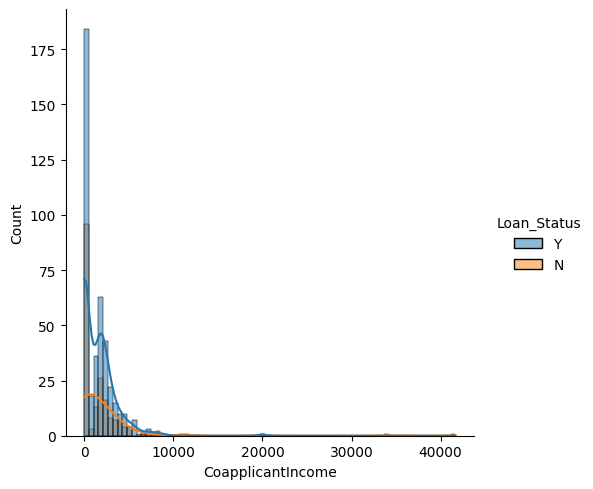


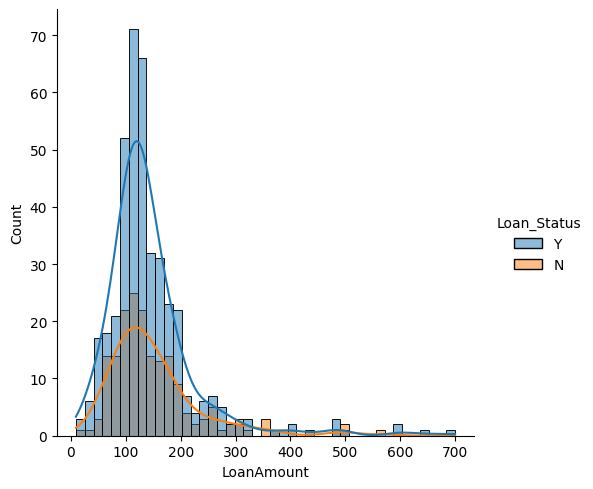


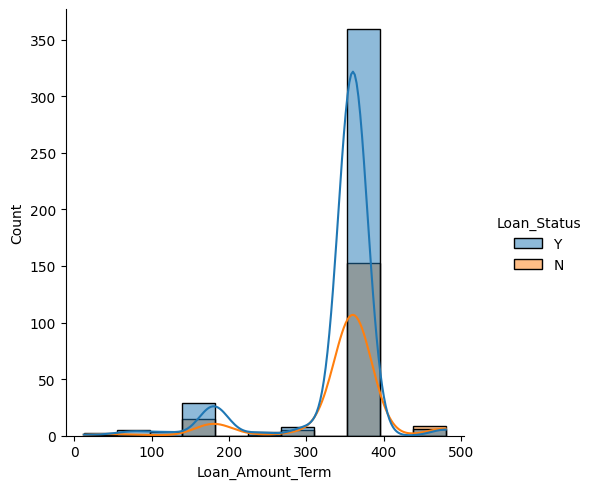


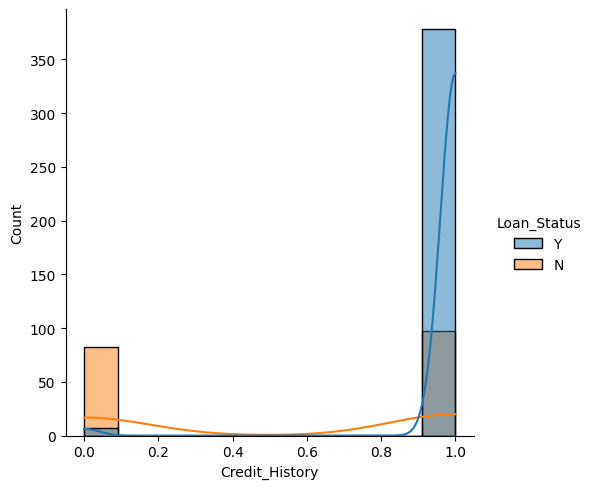


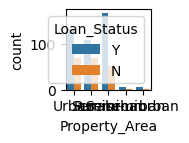


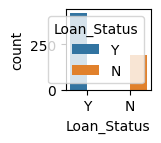












dataset.isna().sum() #checking how many values aren't present in each column and taking summation of missing

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

dataset["LoanAmount\_log"]=np.log(dataset["LoanAmount"])

dataset["Gender"].fillna(dataset["Gender"].mode()[0],inplace=True) #This method fills the missing values in dataset with mode values in that column  
dataset["Married"].fillna(dataset["Married"].mode()[0],inplace=True)  
dataset["Dependents"].fillna(dataset["Dependents"].mode()[0],inplace=True)  
dataset["Self\_Employed"].fillna(dataset["Self\_Employed"].mode()[0],inplace=True)  
dataset["Credit\_History"].fillna(dataset["Credit\_History"].mode()[0],inplace=True)  
dataset["LoanAmount"].fillna(dataset["LoanAmount"].mean(),inplace=True)  
dataset["LoanAmount\_log"].fillna(dataset["LoanAmount\_log"].mean(),inplace=True)  
dataset["Loan\_Amount\_Term"].fillna(dataset["Loan\_Amount\_Term"].mode()[0],inplace=True)

dataset.isna().sum() #These methods return a DataFrame of the same shape as dataset but with True in places where the original DataFrame has missing values and False where it does not to check whether we have a completely filled dataset or not

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  
LoanAmount\_log 0  
dtype: int64

dataset["Total\_Income"]=dataset["ApplicantIncome"]+dataset["CoapplicantIncome"]  
dataset["Total\_Income\_log"]=np.log(dataset["Total\_Income"])

dataset["Dependents"]=dataset["Dependents"].astype(str)  
X=dataset.iloc[:,np.r\_[1:5,9:11,13:15]].values  
y=dataset.iloc[:,12].values

X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.2,random\_state=0)

labelencoder\_x=LabelEncoder()  
labelencoder\_y=LabelEncoder()

for i in range(5):  
 X\_train[:,i]=labelencoder\_x.fit\_transform(X\_train[:,i])  
X\_train[:,7]=labelencoder\_x.fit\_transform(X\_train[:,7])

y\_train=labelencoder\_y.fit\_transform(y\_train)

labelencoder\_x=LabelEncoder()  
labelencoder\_y=LabelEncoder()

for i in range(5):  
 X\_test[:,i]=labelencoder\_x.fit\_transform(X\_test[:,i])  
X\_test[:,7]=labelencoder\_x.fit\_transform(X\_test[:,7])  
y\_test=labelencoder\_y.fit\_transform(y\_test)

ss=StandardScaler() #The StandardScaler is a preprocessing technique in machine learning used to standardize the features of a dataset. Standardization involves transforming the features so that they have a mean of 0 and a standard deviation of 1.  
X\_train=ss.fit\_transform(X\_train)  
X\_test=ss.fit\_transform(X\_test)

from sklearn.naive\_bayes import GaussianNB #It is a variant of the Naive Bayes algorithm that is well-suited for classification tasks when the features are continuous and assumed to have a Gaussian (normal) distribution. It's particularly useful for numerical data.

nbc=GaussianNB() #It is a variant of the Naive Bayes algorithm that is well-suited for classification tasks when the features are continuous and assumed to have a Gaussian (normal) distribution. It's particularly useful for numerical data.  
nbc.fit(X\_train,y\_train)

GaussianNB()

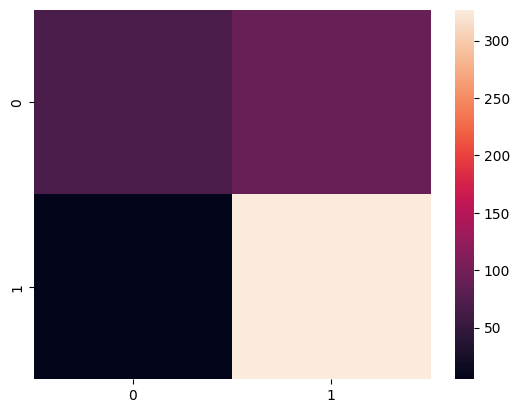
y\_pred\_train=nbc.predict(X\_train)# fit the model to your training data and then use the predict method to make predictions on new, unseen data.  
y\_pred\_test=nbc.predict(X\_test)# fit the model to your testing data and then use the predict method to make predictions on new, unseen data.  
print("Accuracy on Train data :",accuracy\_score(y\_train,y\_pred\_train)\*100)  
print("Accuracy on Test data :",accuracy\_score(y\_test,y\_pred\_test)\*100)

Accuracy on Train data : 80.44806517311609  
Accuracy on Test data : 82.92682926829268

con\_mat\_train=confusion\_matrix(y\_train,y\_pred\_train) #TO analyse and predict how much a model is correct or comparison b/w y\_pred\_train and y\_train  
con\_mat\_test=confusion\_matrix(y\_test,y\_pred\_test) #TO analyse and predict how much a model is correct or comparison b/w y\_pred\_test and y\_test

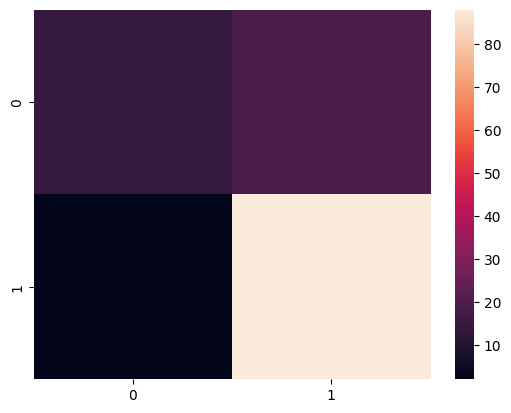
sns.heatmap(con\_mat\_train) #display the confusion matrix

<Axes: >



sns.heatmap(con\_mat\_test)

<Axes: >



The dataset contains information about loan applicants and whether their loan applications were approved or not. The goal is to build a machine learning model to predict whether a loan application will be approved or denied based on the applicant's information. Unique identification number for each loan application. Gender of the applicant (Male/Female). Marital status of the applicant (Yes/No). Number of dependents of the applicant. Education level of the applicant (Graduate/Not Graduate). Whether the applicant is self-employed (Yes/No). Income of the applicant (in dollars). Income of the co-applicant (if any) (in dollars). The loan amount requested (in dollars). Term of the loan (in months). Credit history of the applicant (1.0: Good credit history, 0.0: Poor credit history). Area type of the property (Urban/Rural/Semi Urban). Loan approval status (Y: Approved, N: Not Approved).

* This dataset has various columns like Loan\_Id, Gender, Dependents, Education, Self-Employed, ApplicantIncome, CoApplicantIncome, LoanAmount, Loan\_Amount\_Term, Credit\_History, Property\_area, Loan\_Status.
* From the dataset after visualization the loan status is depending more on credit history than any other column.
* A person with Credit History = 1 will have more chance of getting approved for the loan.
* Loan status is depending very less on LoanAmount, Loan\_Amount\_Term.
* After that loan status is barely depending on other columns like Applicant\_Income, CoApplicant\_Income.
* The loan status dependency on the columns like Gender, Married, Self\_Employed is negligible.

The loan status has no dependency on loan\_id.

CONCLUSION:

This dataset provides valuable insights into the factors that influence loan approval decisions. By building a machine learning model, the financial institution can automate the loan approval process and make more informed lending decisions.

* This dataset has various columns like Loan\_Id, Gender, Dependents, Education, Self-Employed, ApplicantIncome, CoApplicantIncome, LoanAmount, Loan\_Amount\_Term, Credit\_History, Property\_area, Loan\_Status.
* From the dataset after visualization the loan status is depending more on credit history than any other column.
* A person with Credit History = 1 will have more chance of getting approved for the loan.
* Loan status is depending very less on LoanAmount, Loan\_Amount\_Term.
* After that loan status is barely depending on other columns like Applicant\_Income, CoApplicant\_Income.
* The loan status dependency on the columns like Gender, Married, Self\_Employed is negligible.
* The loan status has no dependency on loan\_id.