** PROJECT DAY - 27 **

[Loan prediction using ML]

Importing the Dependencies

```
In [1]:
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.metrics import accuracy_score
```

Data Collection and Processing

```
In [2]:
# loading the dataset to pandas DataFrame
loan_dataset = pd.read_csv('loandata.csv')
```

```
In [3]:
type(loan_dataset)
```

```
Out[3]:
pandas.core.frame.DataFrame
```

```
In [4]:
# printing the first 5 rows of the dataframe
loan_dataset.head()
```

```
Out[4]:
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
0	LP001002	Male	No	0	Graduate	No	5849	0.0	NaN	360.0	1.0	Urban	Y
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.0	1.0	Rural	N
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	360.0	1.0	Urban	Y
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0	1.0	Urban	Y
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	360.0	1.0	Urban	Y

```
In [5]:
# number of rows and columns
loan_dataset.shape
```

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

```
In [6]:
# statistical measures
loan_dataset.describe()
```

```
Out[6]:
```

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	614.000000	614.000000	592.000000	600.000000	564.000000
mean	5403.459283	1621.245798	146.412162	342.000000	0.842199
std	6109.041673	2926.248369	85.587325	65.12041	0.364878
min	150.000000	0.000000	9.000000	12.000000	0.000000
25%	2877.500000	0.000000	100.000000	360.000000	1.000000
50%	3812.500000	1188.500000	128.000000	360.000000	1.000000
75%	5795.000000	2297.250000	168.000000	360.000000	1.000000
max	81000.000000	41667.000000	700.000000	480.000000	1.000000

```
In [7]:
# number of missing values in each column
loan_dataset.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
```

```
In [8]:
# dropping the missing values
loan_dataset = loan_dataset.dropna()
```

```
In [9]:
# number of missing values in each column
loan_dataset.isnull().sum()
```

```
Out[9]:
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
```

```
In [10]:
# label encoding
loan_dataset['Loan_Status'] = loan_dataset['Loan_Status'].map({'N': 0, 'Y': 1})
```

```
In [11]:
# printing the first 5 rows of the dataframe
loan_dataset.head()
```

```
Out[11]:
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.0	1.0	Rural	0
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	360.0	1.0	Urban	1
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0	1.0	Urban	1
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	360.0	1.0	Urban	1
5	LP001011	Male	Yes	2	Graduate	Yes	5417	4196.0	267.0	360.0	1.0	Urban	1

```
In [12]:
# Dependent column values
loan_dataset['Dependents'].value_counts()
```

```
Out[12]:
0     274
2      85
1      80
3+     41
Name: count, dtype: int64
```

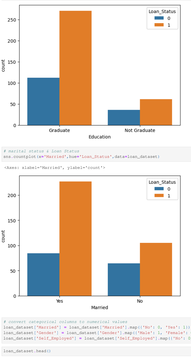
```
In [13]:
# replacing the value of 3+ to 4
loan_dataset = loan_dataset.replace(to_replace="3+", value=4)
```

```
In [14]:
# dependent values
loan_dataset['Dependents'].value_counts()
```

```
Out[14]:
Dependents
0    274
2     85
1     80
4     41
Name: count, dtype: int64
```

Data Visualization

```
In [15]:
# education & Loan Status
sns.countplot(x='Education',hue='Loan_Status',data=loan_dataset)
plt.show()
```




```
In [16]:
# Marital status & Loan Status
sns.countplot(x='Married',hue='Loan_Status',data=loan_dataset)
```

```
Out[16]:
<Axes: xlabel='Married', ylabel='count'>
```



```
In [17]:
# convert categorical column to numerical values
loan_dataset['Married'] = loan_dataset['Married'].map({'No': 0, 'Yes': 1})
loan_dataset['Gender'] = loan_dataset['Gender'].map({'Male': 1, 'Female': 0})
loan_dataset['Self_Employed'] = loan_dataset['Self_Employed'].map({'No': 0, 'Yes': 1})
```

```
In [18]:
loan_dataset.head()
```

```
Out[18]:
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
1	LP001003	1	1	1	Graduate	0	4583	1508.0	128.0	360.0	1.0	Rural	0
2	LP001005	1	1	0	Graduate	1	3000	0.0	66.0	360.0	1.0	Urban	1
3	LP001006	1	1	0	Not Graduate	0	2583	2358.0	120.0	360.0	1.0	Urban	1
4	LP001008	1	0	0	Graduate	0	6000	0.0	141.0	360.0	1.0	Urban	1
5	LP001011	1	1	2	Graduate	1	5417	4196.0	267.0	360.0	1.0	Urban	1

```
In [19]:
# separating the data and label
X = loan_dataset.drop(columns=['Loan_ID','Loan_Status'],axis=1)
Y = loan_dataset['Loan_Status']
```

```
In [20]:
X
```

```
Out[20]:
```

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area
1	1	1	1	Graduate	0	4583	1508.0	128.0	360.0	1.0	Rural
2	1	1	0	Graduate	1	3000	0.0	66.0	360.0	1.0	Urban
3	1	1	0	Not Graduate	0	2583	2358.0	120.0	360.0	1.0	Urban
4	1	0	0	Graduate	0	6000	0.0	141.0	360.0	1.0	Urban
5	1	1	2	Graduate	1	5417	4196.0	267.0	360.0	1.0	Urban
...	...	...	...	...	...	...	...	...	...	...	...
609	0	0	0	Graduate	0	2900	0.0	71.0	360.0	1.0	Rural
610	1	1	4	Graduate	0	4106	0.0	40.0	180.0	1.0	Rural
611	1	1	1	Graduate	0	8072	240.0	253.0	360.0	1.0	Urban
612	1	1	2	Graduate	0	7583	0.0	187.0	360.0	1.0	Urban
613	0	0	0	Graduate	1	4583	0.0	133.0	360.0	0.0	Semiurban

480 rows × 11 columns

```
In [21]:
Y
```

```
Out[21]:
1      0
2      1
3      1
4      1
5      1
      ..
609    1
610    1
611    1
612    1
613    0
Name: Loan_Status, Length: 480, dtype: int64
```

Train Test Split

```
In [22]:
X_train, X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.1,stratify=Y,random_state=2)
```

```
In [23]:
X.shape, X_train.shape, X_test.shape
```

```
Out[23]:
((480, 11), (432, 11), (48, 11))
```

Training the model:

Support Vector Machine Model

```
In [24]:
X.dtypes
```

```
Out[24]:
Gender                int64
Married               int64
Dependents            int64
Education            object
Self_Employed         int64
ApplicantIncome       int64
CoapplicantIncome   float64
LoanAmount          float64
Loan_Amount_Term    float64
Credit_History      float64
Property_Area        object
dtype: object
```

```
In [25]:
categorical_cols = ['Dependents', 'Education', 'Property_Area']
```

```
In [26]:
X[categorical_cols] = X[categorical_cols].astype(str)
X = pd.get_dummies(X, columns=categorical_cols, drop_first=True)
```

```
In [27]:
X.columns
```

```
Out[27]:
Index(['Gender', 'Married', 'Self_Employed', 'ApplicantIncome',
       'CoapplicantIncome', 'LoanAmount', 'Loan_Amount_Term', 'Credit_History',
       'Dependents_1', 'Dependents_2', 'Dependents_4',
       'Education_Not Graduate', 'Property_Area_Semiurban',
       'Property_Area_Urban'],
      dtype='object')
```

```
In [28]:
# 2. Encode Y in Pandas Series
Y = pd.Series(Y)
```

```
In [29]:
# 3. Handle missing values
X = X.dropna()
Y = Y.loc[X.index]
```

```
In [30]:
# 4. ONLY encoding if already done!
# Encoding already applied earlier
```

```
In [31]:
# 5. Encode target
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
Y = le.fit_transform(Y)
```

```
In [32]:
# 6. Train-test split
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(
    X, Y, test_size=0.1, stratify=Y, random_state=2
)
```

```
In [33]:
# 8. Train SVM
from sklearn import svm
classifier = svm.SVC(kernel='linear')
classifier.fit(X_train, Y_train)
```

Model Evaluation

```
In [34]:
# accuracy score on training data
X_train_prediction = classifier.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)
```

```
In [35]:
print('Accuracy on training data : ', training_data_accuracy)
```

```
Out[35]:
Accuracy on training data :  0.7916666666666666
```

```
In [36]:
# accuracy score on testing data
X_test_prediction = classifier.predict(X_test)
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)
```

```
In [37]:
print('Accuracy on test data : ', test_data_accuracy)
```

```
Out[37]:
Accuracy on test data :  0.8333333333333334
```

Making a predictive system

```
In [38]:
feature_names = X.columns
```

```
In [39]:
import numpy as np
import pandas as pd

def loan_prediction_system(
    Gender,
    Married,
    Dependents,
    Education,
    Self_Employed,
    ApplicantIncome,
    CoapplicantIncome,
    LoanAmount,
    Loan_Amount_Term,
    Credit_History,
    Property_Area
):
    # Create input dictionary
    input_data = {
        'Gender': Gender,
        'Married': Married,
        'Dependents': Dependents,
        'Education': Education,
        'Self_Employed': Self_Employed,
        'ApplicantIncome': ApplicantIncome,
        'CoapplicantIncome': CoapplicantIncome,
        'LoanAmount': LoanAmount,
        'Loan_Amount_Term': Loan_Amount_Term,
        'Credit_History': Credit_History,
        'Property_Area': Property_Area
    }

    # Convert to DataFrame
    input_df = pd.DataFrame([input_data])

    # One-hot encode input
    input_df = pd.get_dummies(input_df)

    # Align input with training features
    input_df = input_df.reindex(columns=feature_names, fill_value=0)

    # (Optional) Scale input if scaler was used
    # input_df = scaler.transform(input_df)

    # Make prediction
    prediction = classifier.predict(input_df)

    if prediction[0] == 1:
        return "✅ Loan Approved"
    else:
        return "❌ Loan Not Approved"

result = loan_prediction_system(
    Gender=1,                   # 1 = Male, 0 = Female
    Married=1,                  # 1 = Yes, 0 = No
    Dependents='1',
    Education='Graduate',
    Self_Employed=0,            # 1 = Yes, 0 = No
    ApplicantIncome=5000,
    CoapplicantIncome=2000,
    LoanAmount=150,
    Loan_Amount_Term=360,
    Credit_History=1.0,
    Property_Area='Urban'
)

print(result)
```

** END - PROJECT **