1. Impoting Dependencies :

C-Number of row & columns:

In [ ]:
```
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
```

2. Data Collcetion and Analysis:

A- Loading dataset :

In [ ]:
```
data_loan = pd.read_csv("C:/Machine_learning Python/projets/loanStatus/train_u6l
```

B- View the data (head)

In [ ]:
```
data_loan.head()
```

Out[ ]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome
0	LP001002	Male	No	0	Graduate	No	584
1	LP001003	Male	Yes	1	Graduate	No	458
2	LP001005	Male	Yes	0	Graduate	Yes	300
3	LP001006	Male	Yes	0	Not Graduate	No	258
4	LP001008	Male	No	0	Graduate	No	600

C- type of the data (head)

In [ ]:
```
type(data_loan)
```

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

D-Number of row & columns:

In [ ]:
```
data_loan.shape
```

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

2. Statisctical measures :

A- General Statistic:

In [ ]:
```
data_loan.describe()
```

Out[ ]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_H
count	614.000000	614.000000	592.000000	600.00000	564.(
mean	5403.459283	1621.245798	146.412162	342.00000	0.8
std	6109.041673	2926.248369	85.587325	65.12041	0.3
min	150.000000	0.000000	9.000000	12.00000	0.(
25%	2877.500000	0.000000	100.000000	360.00000	1.(
50%	3812.500000	1188.500000	128.000000	360.00000	1.(
75%	5795.000000	2297.250000	168.000000	360.00000	1.(
max	81000.000000	41667.000000	700.000000	480.00000	1.(

B- Number of missing value in each column;

In [ ]:
```
data_loan.isnull().sum()
```

Out[ ]:
```
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
```

C- Droping the missing values;

In [ ]:
```
data_loan = data_loan.dropna()
data_loan.isnull().sum()
```

Out[ ]:
```
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
```

3. Label Encoding:

A- Replace the Main columns with 0 & 1:

In [ ]:
```
data_loan.replace({"Loan_Status":{ 'N':0, 'Y':1}}, inplace=True)
data_loan.head()
```

Out[ ]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome
1	LP001003	Male	Yes	1	Graduate	No	458
2	LP001005	Male	Yes	0	Graduate	Yes	300
3	LP001006	Male	Yes	0	Not Graduate	No	258
4	LP001008	Male	No	0	Graduate	No	600
5	LP001011	Male	Yes	2	Graduate	Yes	541

B- Value of the Dependents column:

In [ ]:
```
data_loan["Dependents"].value_counts()
```

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

C- Replace the 3+:

In [ ]:
```
data_loan = data_loan.replace(to_replace='3+', value=4)
data_loan.head()
```

Out[ ]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome
1	LP001003	Male	Yes	1	Graduate	No	458
2	LP001005	Male	Yes	0	Graduate	Yes	300
3	LP001006	Male	Yes	0	Not Graduate	No	258
4	LP001008	Male	No	0	Graduate	No	600
5	LP001011	Male	Yes	2	Graduate	Yes	541

In [ ]:
```
data_loan['Dependents'].value_counts()
```

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

In [ ]:
```
data_loan['Dependents'] = pd.to_numeric(data_loan['Dependents'], errors='coerce'
```

In [ ]:
```
data_loan.dtypes
```

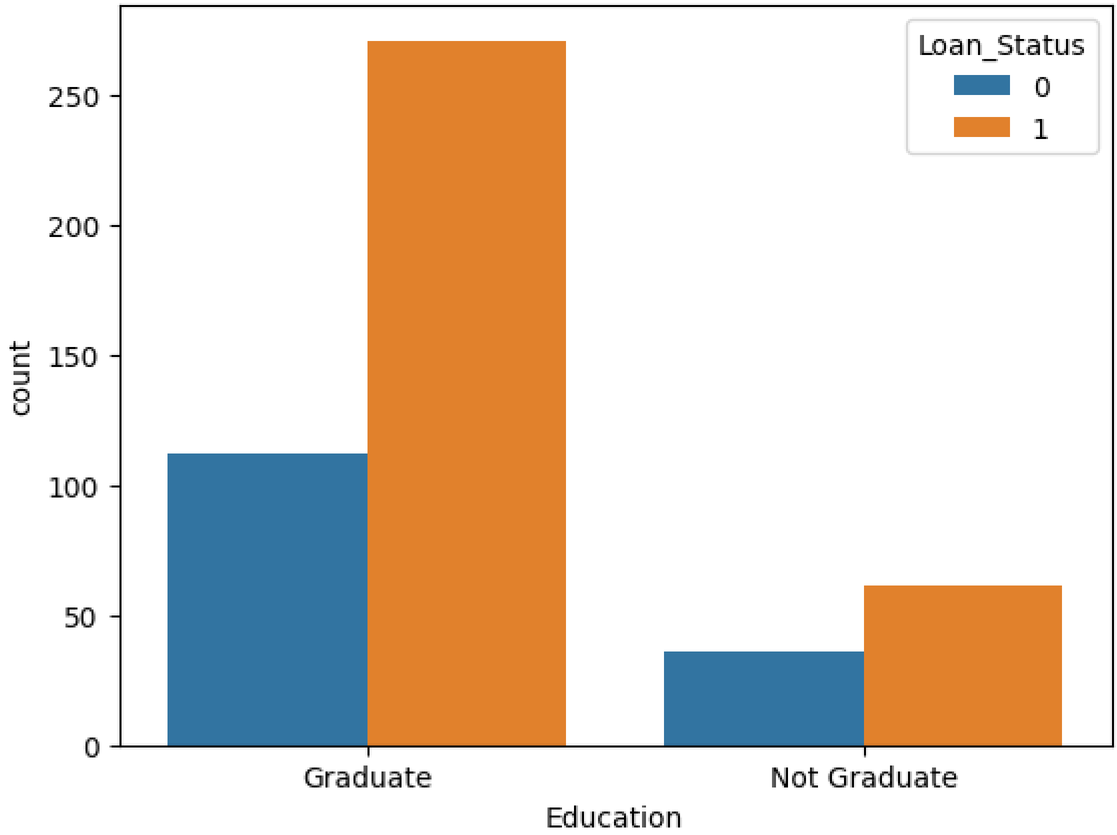
Out[ ]:
```
Loan_ID               object
Gender                object
Married               object
Dependents             int64
Education             object
Self_Employed         object
ApplicantIncome        int64
CoapplicantIncome    float64
LoanAmount           float64
Loan_Amount_Term     float64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

4. Data visualisation:

A- Education & loan Status:

In [ ]:
```
sns.countplot(x= 'Education', hue='Loan_Status', data = data_loan)
```

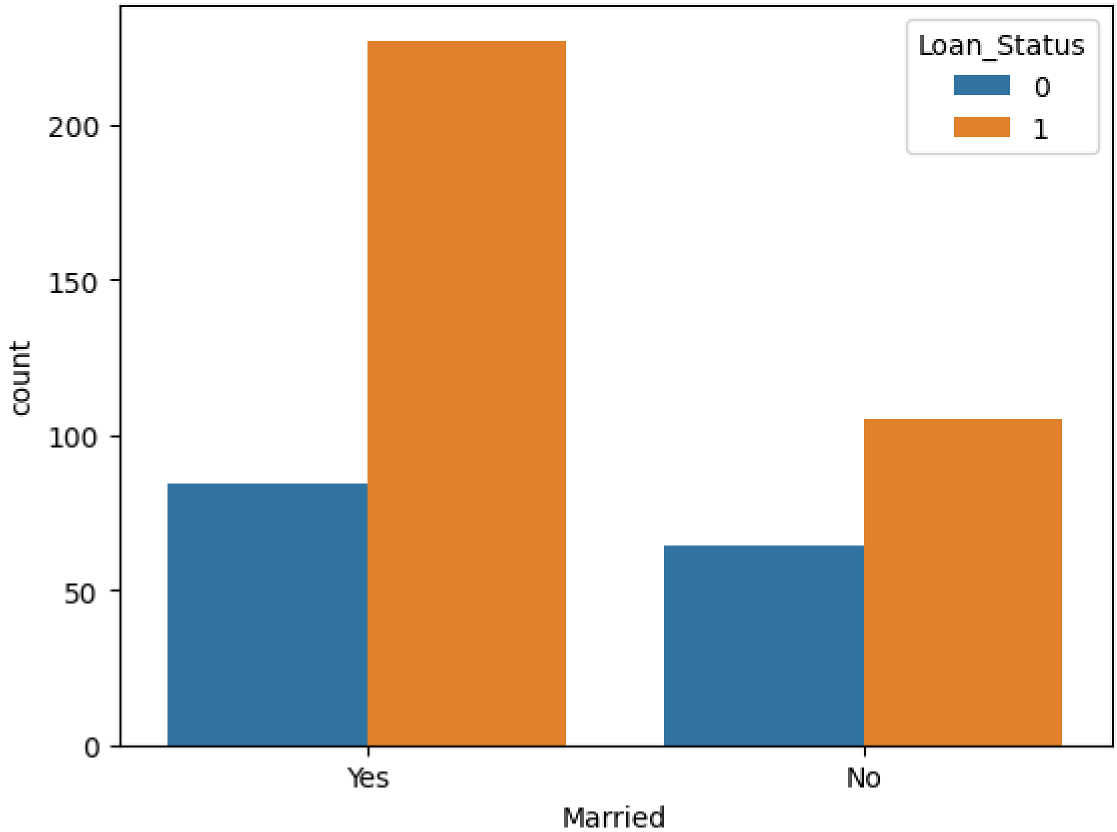
Out[ ]:
```
<Axes: xlabel='Education', ylabel='count'>
```



B- Marital statues & loan statues

In [ ]:
```
sns.countplot(x= 'Married', hue='Loan_Status', data = data_loan)
```

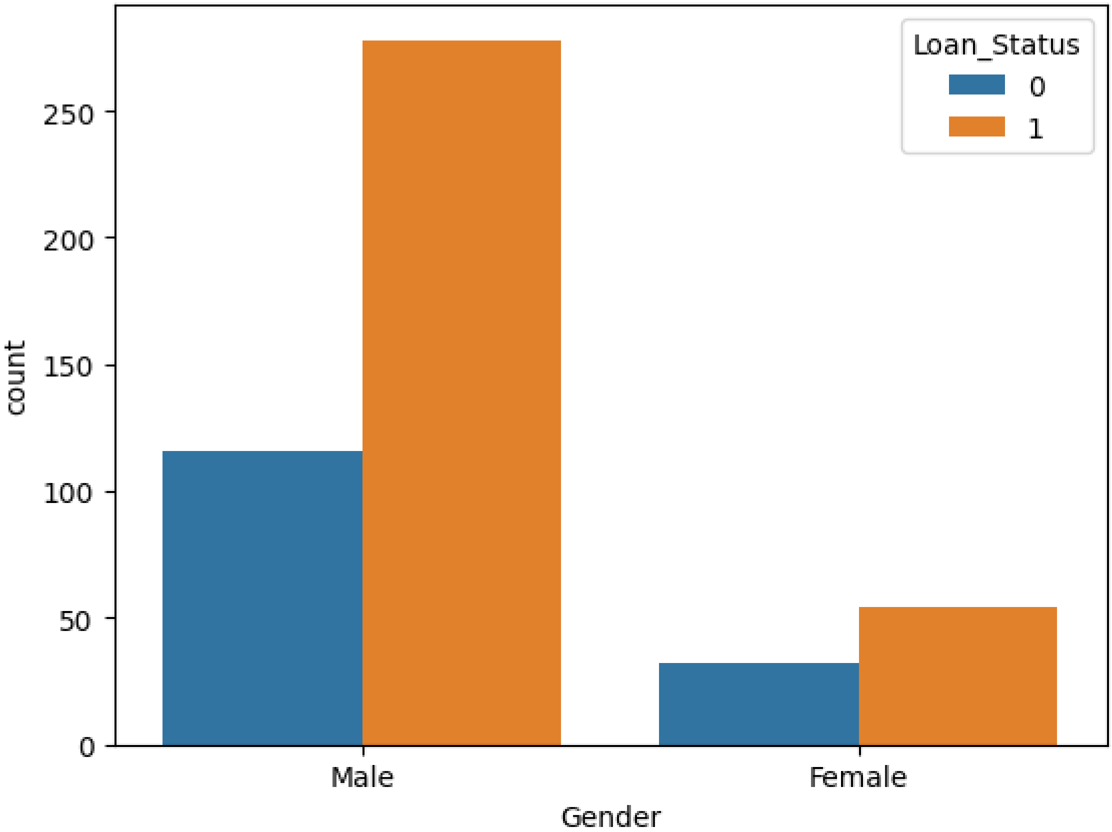
Out[ ]: `<Axes: xlabel='Married', ylabel='count'>`



C- Gender & loan statues

In [ ]:
```
sns.countplot(x= 'Gender', hue='Loan_Status', data = data_loan)
```

Out[ ]: <Axes: xlabel='Gender', ylabel='count'>



D- Convert categorical columun to numerical values:

In [ ]:
```
data_loan.replace({"Married":{ 'No':0, 'Yes':1},"Gender":{ 'Male':1, 'Female':0}
data_loan.head()
```

Out[ ]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome
1	LP001003	1	1	1	1	0	458
2	LP001005	1	1	0	1	1	300
3	LP001006	1	1	0	0	0	258
4	LP001008	1	0	0	1	0	600
5	LP001011	1	1	2	1	1	541

In [ ]:
```
data_loan.dtypes
```

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

5. Train test split:

A- Separating a data & label

In [ ]:
```
X = data_loan.drop(columns=["Loan_ID","Loan_Status"], axis= 1)
Y = data_loan["Loan_Status"]
print(X)
print(Y)
```

```
     Gender  Married  Dependents  Education  Self_Employed  ApplicantIncome  \
1         1        1           1          1              0             4583
2         1        1           0          1              1             3000
3         1        1           0          0              0             2583
4         1        0           0          1              0             6000
5         1        1           2          1              1             5417
..      ...      ...         ...        ...            ...              ...
609       0        0           0          1              0             2900
610       1        1           4          1              0             4106
611       1        1           1          1              0             8072
612       1        1           2          1              0             7583
613       0        0           0          1              1             4583

     CoapplicantIncome  LoanAmount  Loan_Amount_Term  Credit_History  \
1               1508.0       128.0             360.0             1.0
2                  0.0        66.0             360.0             1.0
3               2358.0       120.0             360.0             1.0
4                  0.0       141.0             360.0             1.0
5               4196.0       267.0             360.0             1.0
..                 ...         ...               ...             ...
609                0.0        71.0             360.0             1.0
610                0.0        40.0             180.0             1.0
611              240.0       253.0             360.0             1.0
612                0.0       187.0             360.0             1.0
613                0.0       133.0             360.0             0.0

     Property_Area
1                0
2                2
3                2
4                2
5                2
..             ...
609              0
610              0
611              2
612              2
613              1

[480 rows x 11 columns]
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
```

B- Test Split

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

In [ ]:
```
print(X.shape,X_train.shape, X_test.shape)
```

```
(480, 11) (432, 11) (48, 11)
```

6. Training the model (SVM):

A- Loading the model

In [ ]:
```
classifier = svm.SVC(kernel='linear')
```

B- Training the support Vector Machine Model:

In [ ]:
```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [ ]:
```
classifier.fit(X_train_scaled, Y_train)
```

Out[ ]:
```
▾ SVC
SVC(kernel='linear')
```

7. Model Evaluation:

A- Accuracy score of training data:

In [ ]:
```
X_train_prediction = classifier.predict(X_train_scaled)
data_accuracy = accuracy_score(X_train_prediction, Y_train)
```

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

```
Accuracy on training data :  0.8055555555555556
```

B- Accuracy score of testing data:

In [ ]:
```
X_test_prediction = classifier.predict(X_test_scaled)
data_accuracy2 = accuracy_score(X_test_prediction, Y_test)
```

In [ ]:
```
print('Accuracy on testing data : ', data_accuracy2)
```

```
Accuracy on testing data :  0.8333333333333334
```

C- Exemple

In [ ]:
```
def predictionF(Loan_ID, Gender, Married, Dependents, Education, Self_Employed,
    # Créer un DataFrame avec les caractéristiques d'entrée
    input_data = pd.DataFrame({
        'Loan_ID': [Loan_ID],
        '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]
    })

    # Remplacement des valeurs catégorielles
    input_data.replace({"Married": {'No': 0, 'Yes': 1},
                        "Gender": {'Male': 1, 'Female': 0},
                        "Self_Employed": {'No': 0, 'Yes': 1},
                        "Property_Area": {'Rural': 0, 'Urban': 2, 'Semiurban': 1
                        "Education": {'Graduate': 1, 'Not Graduate': 0}}, inplac

    # Convertir la colonne "Dependents" en type numérique
    input_data['Dependents'] = pd.to_numeric(input_data['Dependents'], errors='c

    # Standardiser les caractéristiques si nécessaire
    std_data = scaler.transform(input_data.drop(columns=["Loan_ID"], axis=1))

    # Prédire avec le modèle
    prediction = classifier.predict(std_data)

    # Afficher la prédiction
    print(prediction[0])

    if prediction[0] == 0:
        print("The loan is not accepted.")
    else:
        print("The loan is accepted.")
predictionF('LP001020', 'Male', 'Yes', 1, 'Graduate', 'No', 12841, 10968, 349, 3
```

```
1
The loan is accepted.
```