importing the Libraries

In [6]:
```
import numpy as np #making array
import pandas as pd #data fram
import seaborn as sns #plots
from sklearn.model_selection import train_test_split #train
from sklearn import svm #Support Vector Machine (SVM)
from sklearn.metrics import accuracy_score # accuracy
```

Data Collection and Processing

In [7]:
```
# loading the dataset to pandas DataFrame
loan_dataset = pd.read_csv('/kaggle/input/loan-predication/train_u6
```

In [8]:
```
type(loan_dataset)
```

Out[8]:
```
pandas.core.frame.DataFrame
```

In [9]:
```
# printing the first 5 rows of the dataframe
loan_dataset.head()
```

Out[9]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	C
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 [10]:
```
# number of rows and columns
loan_dataset.shape
```

Out[10]:
```
(614, 13)
```

In [11]:
```
# statistical measures
loan_dataset.describe()
```

Out[11]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_Histo
count	614.000000	614.000000	592.000000	600.00000	564.0000
mean	5403.459283	1621.245798	146.412162	342.00000	0.8421
std	6109.041673	2926.248369	85.587325	65.12041	0.3648
min	150.000000	0.000000	9.000000	12.00000	0.0000
25%	2877.500000	0.000000	100.000000	360.00000	1.0000
50%	3812.500000	1188.500000	128.000000	360.00000	1.0000
75%	5795.000000	2297.250000	168.000000	360.00000	1.0000
max	81000.000000	41667.000000	700.000000	480.00000	1.0000

In [12]:
```
# number of missing values in each column
loan_dataset.isnull().sum()
```

Out[12]:
```
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 [13]:
```
# dropping the missing values
loan_dataset = loan_dataset.dropna()
```

In [14]:
```
# number of missing values in each column
loan_dataset.isnull().sum()
```

Out[14]:
```
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 [15]:
```
# label encoding
loan_dataset.replace({"Loan_Status":{'N':0,'Y':1}},inplace=True)
```

```
/tmp/ipykernel_33/474101102.py:2: FutureWarning: Downcasting behav
ior in `replace` is deprecated and will be removed in a future ver
sion. To retain the old behavior, explicitly call `result.infer_ob
jects(copy=False)`. To opt-in to the future behavior, set `pd.set_
option('future.no_silent_downcasting', True)`
  loan_dataset.replace({"Loan_Status":{'N':0,'Y':1}},inplace=True)
```

In [16]:
```
# printing the first 5 rows of the dataframe
loan_dataset.head()
```

Out[16]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	C
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	
5	LP001011	Male	Yes	2	Graduate	Yes	5417	

In [17]:
```
# Dependent column values
loan_dataset['Dependents'].value_counts()
```

Out[17]:
```
Dependents
0     274
2      85
1      80
3+     41
Name: count, dtype: int64
```

In [18]:
```
# replacing the value of 3+ to 4
loan_dataset = loan_dataset.replace(to_replace='3+', value=4)
```

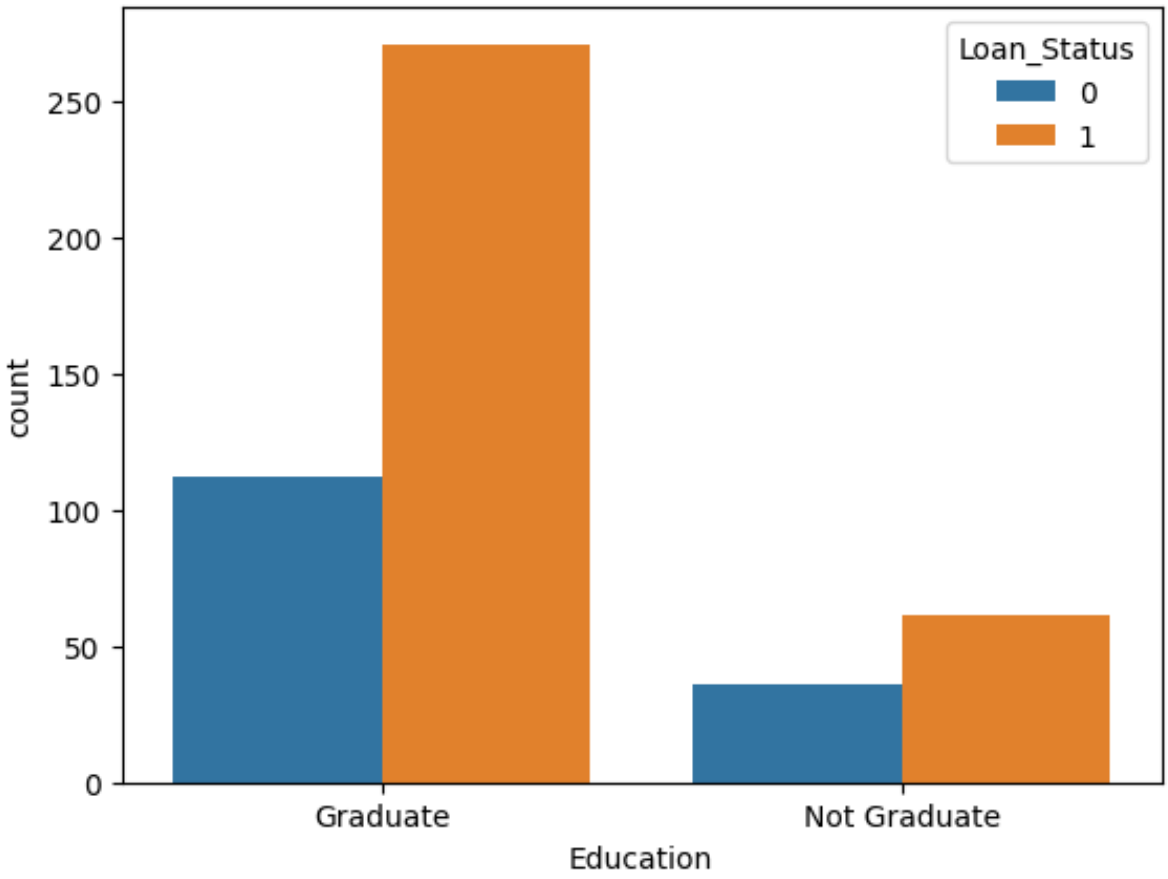
In [19]:
```
# dependent values
loan_dataset['Dependents'].value_counts()
```

Out[19]:
```
Dependents
0    274
2     85
1     80
4     41
Name: count, dtype: int64
```

Data Visualization

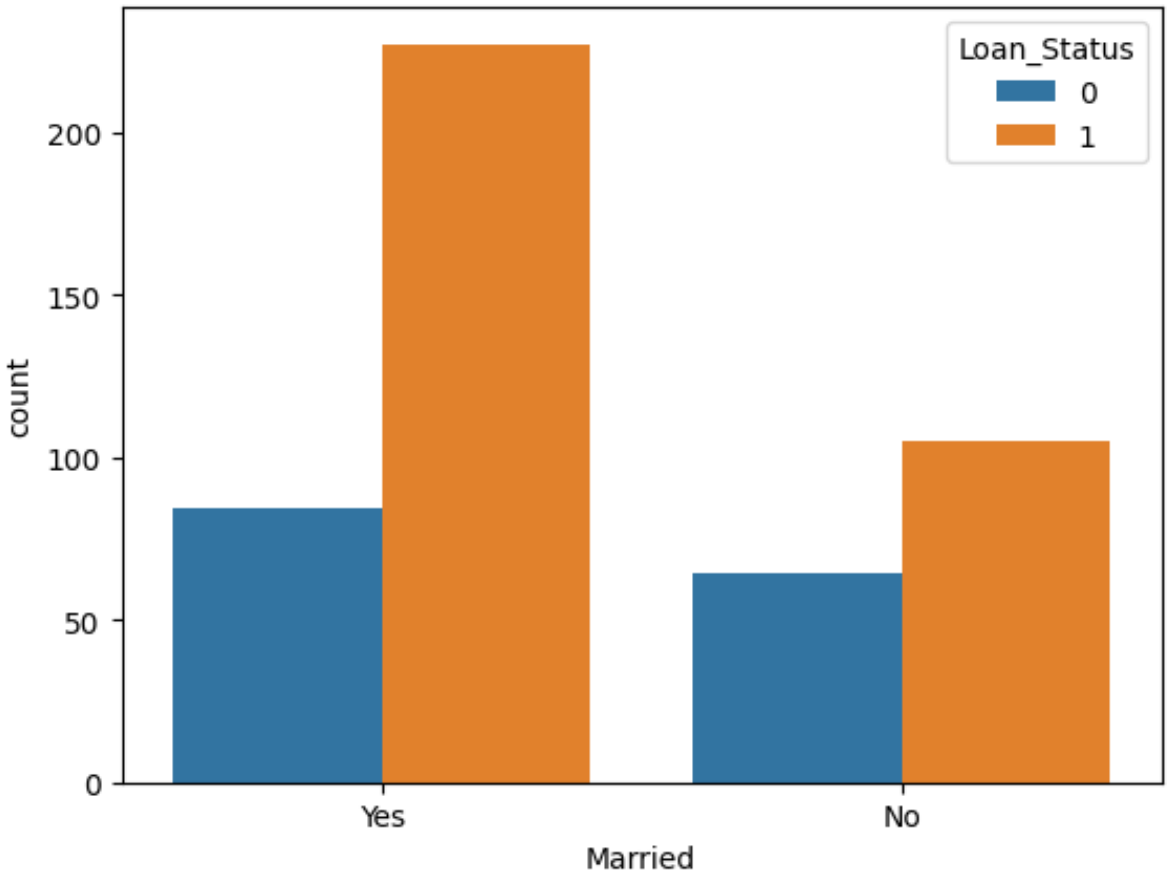
In [20]:
```
# education & Loan Status
sns.countplot(x='Education',hue='Loan_Status',data=loan_dataset)
```

Out[20]: <Axes: xlabel='Education', ylabel='count'>



In [21]:
```
# marital status & Loan Status
sns.countplot(x='Married',hue='Loan_Status',data=loan_dataset)
```

Out[21]: <Axes: xlabel='Married', ylabel='count'>



In [22]:
```
# convert categorical columns to numerical values
loan_dataset.replace({'Married':{'No':0,'Yes':1},'Gender':{'Male':1
                     'Property_Area':{'Rural':0,'Semiurban':1,'Urb
```

```
/tmp/ipykernel_33/2432111705.py:2: FutureWarning: Downcasting beha
vior in `replace` is deprecated and will be removed in a future ve
rsion. To retain the old behavior, explicitly call `result.infer_o
bjects(copy=False)`. To opt-in to the future behavior, set `pd.set
_option('future.no_silent_downcasting', True)`
  loan_dataset.replace({'Married':{'No':0,'Yes':1},'Gender':{'Mal
e':1,'Female':0},'Self_Employed':{'No':0,'Yes':1},
```

In [23]:
```
loan_dataset.head()
```

Out[23]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	C
1	LP001003	1	1	1	1	0	4583	
2	LP001005	1	1	0	1	1	3000	
3	LP001006	1	1	0	0	0	2583	
4	LP001008	1	0	0	1	0	6000	
5	LP001011	1	1	2	1	1	5417	

In [24]:
```
# separating the data and label
X = loan_dataset.drop(columns=['Loan_ID','Loan_Status'],axis=1)
Y = loan_dataset['Loan_Status']
```

In [25]:
```
print(X)
print(Y)
```

```
     Gender  Married Dependents  Education  Self_Employed  Applica
ntIncome  \
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_Histo
ry  \
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
```

Train Test Split

In [26]:
```
X_train, X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.1
```

In [27]:
```
print(X.shape, X_train.shape, X_test.shape)
```

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

Training the model:

Support Vector Machine Model

In [28]:
```
classifier = svm.SVC(kernel='linear')
```

In [29]:
```
#training the support Vector Macine model
classifier.fit(X_train,Y_train)
```

Out[29]:
```
SVC(kernel='linear')
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Model Evaluation

In [30]:
```
# accuracy score on training data
X_train_prediction = classifier.predict(X_train)
training_data_accuray = accuracy_score(X_train_prediction,Y_train)
```

In [31]:
```
print('Accuracy on training data : ', training_data_accuray)
```

```
Accuracy on training data :  0.7986111111111112
```

In [32]:
```
# accuracy score on training data
X_test_prediction = classifier.predict(X_test)
test_data_accuray = accuracy_score(X_test_prediction,Y_test)
```

In [33]:
```
print('Accuracy on test data : ', test_data_accuray)
```

```
Accuracy on test data :  0.8333333333333334
```

Making a predictive system

In [46]:
```
import warnings
warnings.filterwarnings("ignore")
```

In [49]:
```
# Print the values at index 200
index_data = loan_dataset.iloc[200]
print(index_data)
```

```
Loan_ID              LP001846
Gender                      0
Married                     0
Dependents                  4
Education                   1
Self_Employed               0
ApplicantIncome          3083
CoapplicantIncome         0.0
LoanAmount              255.0
Loan_Amount_Term        360.0
Credit_History            1.0
Property_Area               0
Loan_Status                 1
Name: 255, dtype: object
```

In [50]:
```
# Extract the values at index 200 and convert them to a list
index_list = loan_dataset.iloc[200].tolist()

# Print the list
print(index_list)
```

```
['LP001846', 0, 0, 4, 1, 0, 3083, 0.0, 255.0, 360.0, 1.0, 0, 1]
```

In [51]:
```
# Making a predictive system
input_data = (0, 0, 4, 1, 0, 3083, 0.0, 255.0, 360.0, 1.0, 0)  # Ex

# Converting the input data to a numpy array
input_data_as_numpy_array = np.asarray(input_data)

# Reshape the array as we are predicting for one instance
input_data_reshaped = input_data_as_numpy_array.reshape(1, -1)

# Standardize the data (if needed based on how the model was traine
# For instance, if you used StandardScaler:
# input_data_standardized = scaler.transform(input_data_reshaped)

# Make the prediction
prediction = classifier.predict(input_data_reshaped)

print(prediction)

if prediction[0] == 1:
    print('The loan will be approved')
else:
    print('The loan will not be approved')
```

```
[1]
The loan will be approved
```

Save the trained SVM model

In [53]:
```
import pickle

# Save the model to a file
filename = 'loan_status_model.pkl'
with open(filename, 'wb') as file:
    pickle.dump(classifier, file)

print(f"Model saved to {filename}")
```

```
Model saved to loan_status_model.pkl
```