In [2]:
```
import pandas as pd
import numpy as np
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.metrics import accuracy_score
```

Data Collection and processing

In [7]:
```
loan_ds = pd.read_csv('/content/loan_prediction_ML.csv')
```

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

Out[8]:

pandas.core.frame.DataFrame

def __init__(data=None, index: Axes | None=None, columns: Axes | None=None, dtype: Dty pe | None=None, copy: bool | None=None) -> None

Two-dimensional, size-mutable, potentially heterogeneous tabular data.

Data structure also contains labeled axes (rows and columns).
Arithmetic operations align on both row and column labels. Can be
thought of as a dict-like container for Series objects. The primary

In [33]:
```
loan_ds.head(10)
```

Out[33]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome
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
6	LP001013	Male	Yes	0	Not Graduate	No	2333
7	LP001014	Male	Yes	4	Graduate	No	3036
8	LP001018	Male	Yes	2	Graduate	No	4006
9	LP001020	Male	Yes	1	Graduate	No	12841
10	LP001024	Male	Yes	2	Graduate	No	3200

In [11]:
```
loan_ds.shape
```

Out[11]: (614, 13)

In [16]:
```
# Statistical Measures in the datasets
loan_ds.describe()
```

Out[16]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_Hist
count	614.000000	614.000000	592.000000	600.00000	564.000
mean	5403.459283	1621.245798	146.412162	342.00000	0.842
std	6109.041673	2926.248369	85.587325	65.12041	0.364
min	150.000000	0.000000	9.000000	12.00000	0.000
25%	2877.500000	0.000000	100.000000	360.00000	1.000
50%	3812.500000	1188.500000	128.000000	360.00000	1.000
75%	5795.000000	2297.250000	168.000000	360.00000	1.000
max	81000.000000	41667.000000	700.000000	480.00000	1.000

In [17]:
```
# no of missig values in each columns
loan_ds.isnull().sum()
```

Out[17]:

	0
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 [22]:
```
# dropping all the missing values
loan_ds = loan_ds.dropna()
```

In [23]:
```
loan_ds.isnull().sum()
```

Out[23]:

	0
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 [25]:
```
# Label Encoding
loan_ds.replace({"Loan_Status": {'Y':1 ,'N':0}}, inplace = True)
```

```
/tmp/ipython-input-25-2394663655.py:2: FutureWarning: Downcasting behavior in `replace
` is deprecated and will be removed in a future version. To retain the old behavior, e
xplicitly call `result.infer_objects(copy=False)`. To opt-in to the future behavior, s
et `pd.set_option('future.no_silent_downcasting', True)`
  loan_ds.replace({"Loan_Status": {'Y':1 ,'N':0}}, inplace = True)
/tmp/ipython-input-25-2394663655.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/use
r_guide/indexing.html#returning-a-view-versus-a-copy
  loan_ds.replace({"Loan_Status": {'Y':1 ,'N':0}}, inplace = True)
```

In [27]:
```
loan_ds.head(2)
```

Out[27]:

	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	

In [28]:
```
# Dependent column all values
loan_ds['Dependents'].value_counts()
```

Out[28]:

Dependents	count
0	274
2	85
1	80
3+	41

dtype: int64

In [29]:
```
# Replacing the '3+' value to '4'

loan_ds = loan_ds.replace(to_replace = '3+' , value = 4)
```

In [30]:
```
loan_ds['Dependents'].value_counts()
```

Out[30]:

Dependents	count
0	274
2	85
1	80
4	41

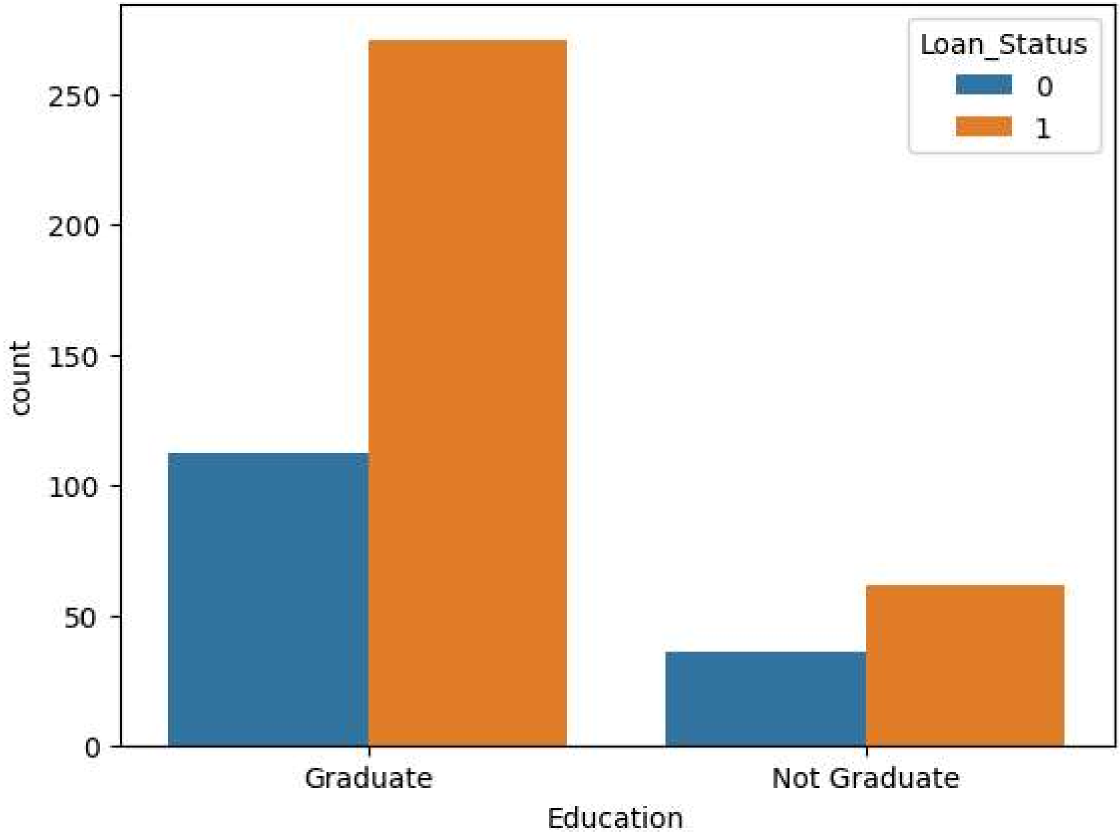
dtype: int64

**Data Visualiztion **

In [31]:
```
# Education & Loan_Status

sns.countplot(x = 'Education' , hue = 'Loan_Status' , data = loan_ds)
```

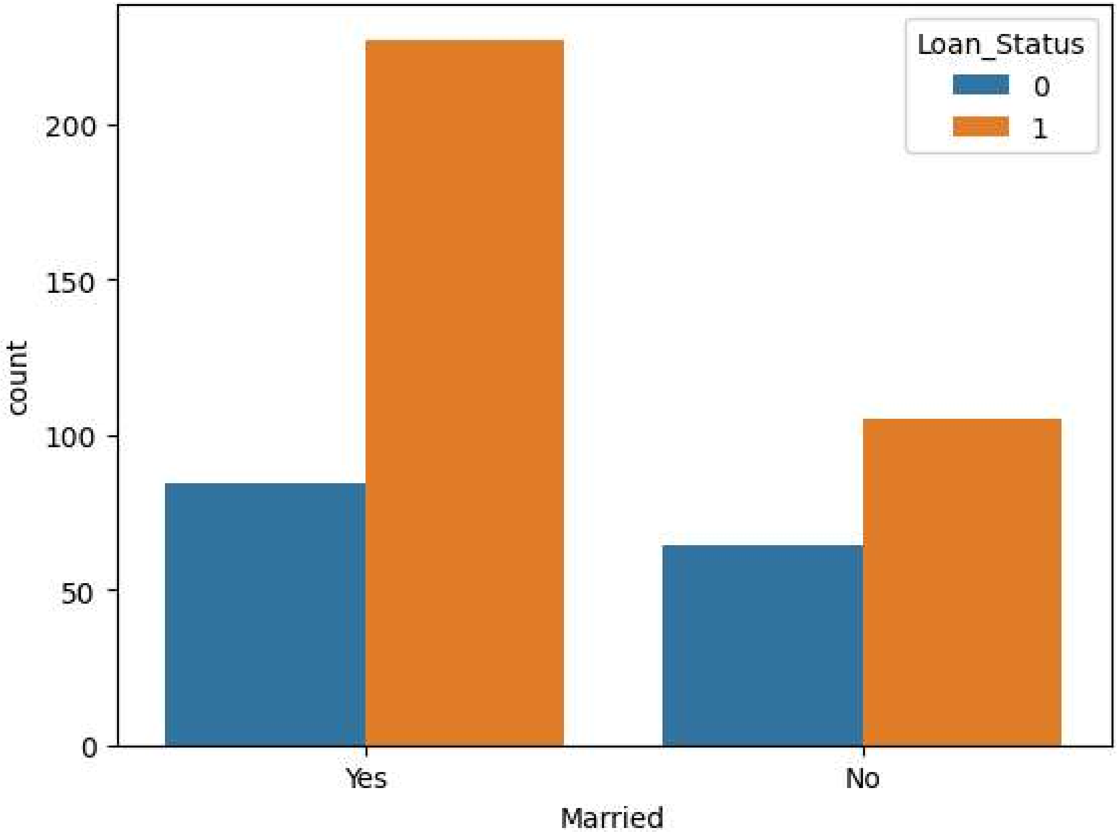
Out[31]: <Axes: xlabel='Education', ylabel='count'>



In [32]:
```
# Marital status & Loan_Status

sns.countplot(x = 'Married', hue = 'Loan_Status' , data = loan_ds)
```

Out[32]: <Axes: xlabel='Married', ylabel='count'>



In [38]:
```
# Convert categorical(gendre, married etc...) columns to numerical values

loan_ds.replace({"Married":{'Yes':1,'No':0}, "Gender" :{'Male':1, 'Female':0}, "Self_
                 "Property_Area":{'Rural':0, 'Semiurban':1, 'Urban':2}, "Education":{
```

In [39]:
```
loan_ds.head(7)
```

Out[39]:

	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	
6	LP001013	1	1	0	0	0	2333	
7	LP001014	1	1	4	1	0	3036	

In [40]:
```
# Seperating the data and label
X = loan_ds.drop(columns = ['Loan_ID', 'Loan_Status'], axis = 1)      # dataset witho
Y = loan_ds['Loan_Status']
```

In [41]:
```
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
```

Train Test Split

In [42]:
```
X_train , X_test , Y_train , Y_test = train_test_split(X,Y, test_size = 0.1, stratify
```

In [43]:
```
print(X.shape, X_train.shape, X_test.shape)
```

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

Training the Model:

Support Vector Machine Model(SVM)

In [44]:
```
classifier = svm.SVC(kernel = 'linear')
```

In [45]:
```
# Training the SVM

classifier.fit(X_train, Y_train)
```

Out[45]:
```
SVC
SVC(kernel='linear')
```

Model Evaluation

In [49]:
```
# accuracy score on training data

X_train_prediction = classifier.predict(X_train)     # problems present in the textboo
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)
```

In [50]:
```
print('Accuracy on the training data: ', training_data_accuracy)
```

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

In [52]:
```
# accuracy score on training data (2)

X_test_prediction = classifier.predict(X_test)           # problems not present in bo
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)
```

In [53]:
```
print('Accuracy on the test data: ', test_data_accuracy)
```

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

In [ ]: