Predicting loan eligibility

```
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
```

Data Processing

```
data = pd.read_csv('/content/loan_data.csv')
```

```
type(data)
```

```
pandas.core.frame.DataFrame
```

```
data.head(3)
# some datas are missing
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome
0	LP001002	Male	No	0	Graduate	No	5849
1	LP001003	Male	Yes	1	Graduate	No	4583
2	LP001005	Male	Yes	0	Graduate	Yes	3000

```
data.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
```

```
# number of datas missing in each column
# data.isnull(): it gives true and false in each data point in dataframe
data.isnull().sum()
```

```
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
```

```
# we can't replace missing values with mean or anyother statisticl thing(imputation) a
# so we are dropping them
```

```
data = data.dropna()
data.isnull().sum()
```

```
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
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 480 entries, 1 to 613
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Loan_ID            480 non-null    object
 1   Gender             480 non-null    object
 2   Married            480 non-null    object
 3   Dependents         480 non-null    object
 4   Education          480 non-null    object
```

```
 5   Self_Employed      480 non-null    object
 6   ApplicantIncome    480 non-null    int64
 7   CoapplicantIncome  480 non-null    float64
 8   LoanAmount         480 non-null    float64
 9   Loan_Amount_Term   480 non-null    float64
 10  Credit_History     480 non-null    float64
 11  Property_Area      480 non-null    object
 12  Loan_Status        480 non-null    object
dtypes: float64(4), int64(1), object(8)
memory usage: 52.5+ KB
```

Label encoding

(convereting yes to 1 and no to 0)

```
data.replace({'Loan_Status':{'N':0, 'Y':1}}, inplace=True)
data['Loan_Status'].value_counts()
```

```
1    332
0    148
Name: Loan_Status, dtype: int64
```

```
data['Dependents'].value_counts()
# here the data contains 3+ as a value but it will be difficult for our model to inter
data.replace({'Dependents':{'3+' :4}}, inplace=True)
# data.replace(to_replace='3+', value='4')
data['Dependents'].value_counts()
# data['Property_Area'].value_counts()
```

```
0    274
2     85
1     80
4     41
Name: Dependents, dtype: int64
```

Converting all categorical var to Numerical which are possible to convert

```
data.replace({'Married':{'No':0, 'Yes':1},'Self_Employed':{'No':0, 'Yes':1},'Gender':{
             'Property_Area':{'Rural':0, 'Semiurban':1, 'Urban':2},'Education':{'Gradu
data.head(8)
```

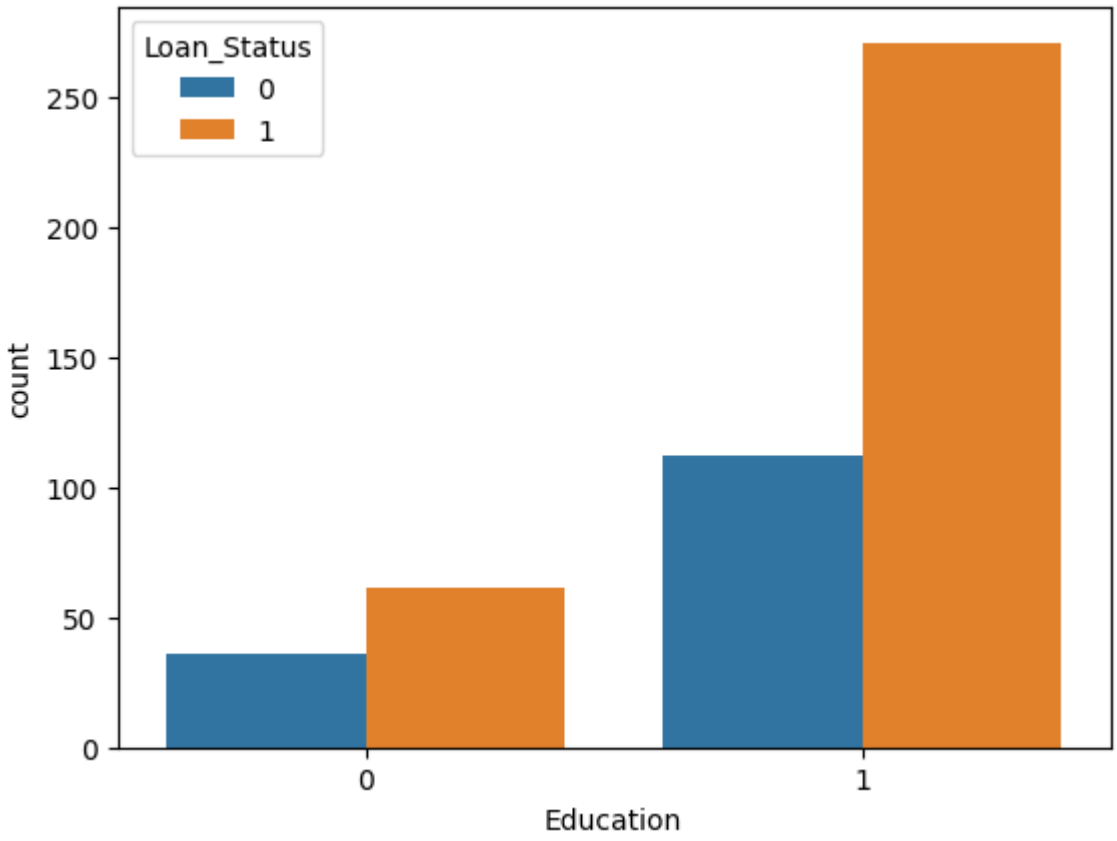
	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome
1	LP001003	1	1	1	1	0	4583
2	LP001005	1	1	0	1	1	3000
3	LP001006	1	1	0	0	0	2583

Data Visualisation

Double-click (or enter) to edit

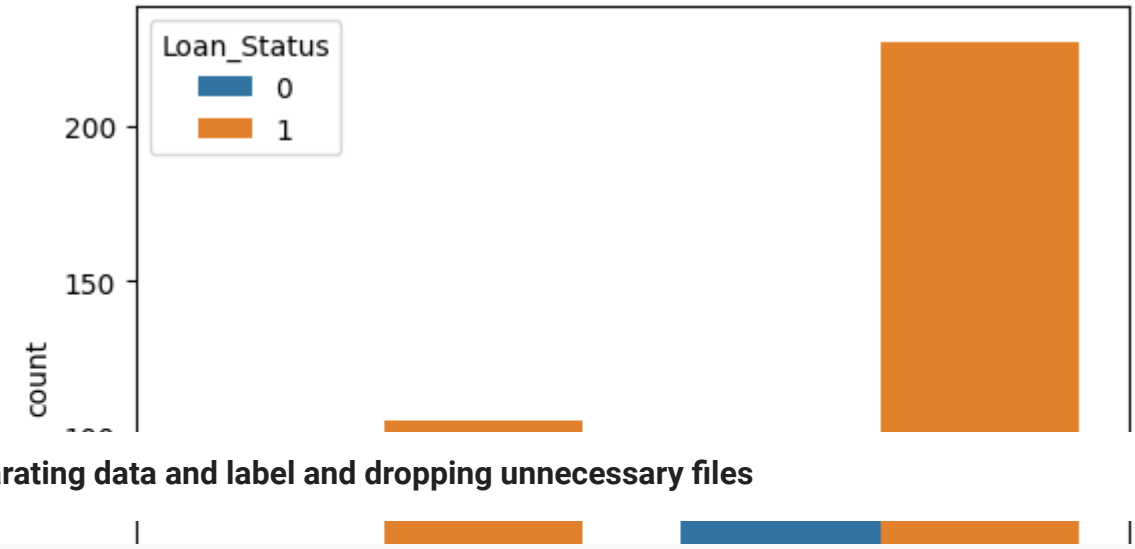
```
1 sns.countplot(x='Education', hue='Loan_Status', data=data)
```

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




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

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



Separating data and label and dropping unnecessary files

```
1 x = data.drop(columns=['Loan_ID', 'Loan_Status'], axis=1)
2 y = data['Loan_Status']
3 print(x.head(2))
4 print(y.head(2))
```

```
   Gender  Married Dependents  Education  Self_Employed  ApplicantIncome  \
1       1        1          1          1              0             4583
2       1        1          0          1              1             3000

   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

   Property_Area
1              0
2              2
1    0
2    1
Name: Loan_Status, dtype: int64
```

Splitting training and testing data

```
1 x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=
2 print(len(x_train))
3 print(len(x_test))
4 print(len(x_test)/(len(x_train) +len(x_test)))
```

```
384
96
0.2
```

Training Model

(using svm)

```
classifier = svm.SVC(kernel='linear')
```

```
classifier.fit(x_train, y_train)
```

Model Evaluation

```
# accuracy score on training data
x_train_prediction = classifier.predict(x_train)
training_data_accuracy = accuracy_score(x_train_prediction, y_train)
print('Accuracy score in training data is:', training_data_accuracy)
```

```
Accuracy score in training data is: 0.8046875
```

```
x_test_prediction = classifier.predict(x_test)
testing_data_accuracy = accuracy_score(x_test_prediction, y_test)
print('Accuracy score in testing data is:', testing_data_accuracy)
```

```
Accuracy score in testing data is: 0.8333333333333334
```