```
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 = pd.read_csv('/content/train_u6lujuX_CVtuZ9i (1).csv')
```

```
data.head()
```

lents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_A
0	Graduate	No	5849	0.0	NaN	
1	Graduate	No	4583	1508.0	128.0	
0	Graduate	Yes	3000	0.0	66.0	
0	Not Graduate	No	2583	2358.0	120.0	
0	Graduate	No	6000	0.0	141.0	

Next steps: Generate code with data View recommended plots

```
data.tail()
```

Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coa
LP002978	Female	No	0	Graduate	No	2900	
LP002979	Male	Yes	3+	Graduate	No	4106	
LP002983	Male	Yes	1	Graduate	No	8072	
LP002984	Male	Yes	2	Graduate	No	7583	
LP002990	Female	No	0	Graduate	Yes	4583	

```
data.describe()
```

	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

```
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
```

```
data.shape
```

```
(614, 13)
```

```
#no.of missing values in each column

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
```

```
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
```

```
#label_encoding

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

```
data.head()
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.(
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	360.(
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.(
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	360.(
5	LP001011	Male	Yes	2	Graduate	Yes	5417	4196.0	267.0	360.(

Next steps: Generate code with data View recommended plots

```
# dependent columns

data['Dependents'].value_counts()
```

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

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

```
data['Dependents'].value_counts()
```

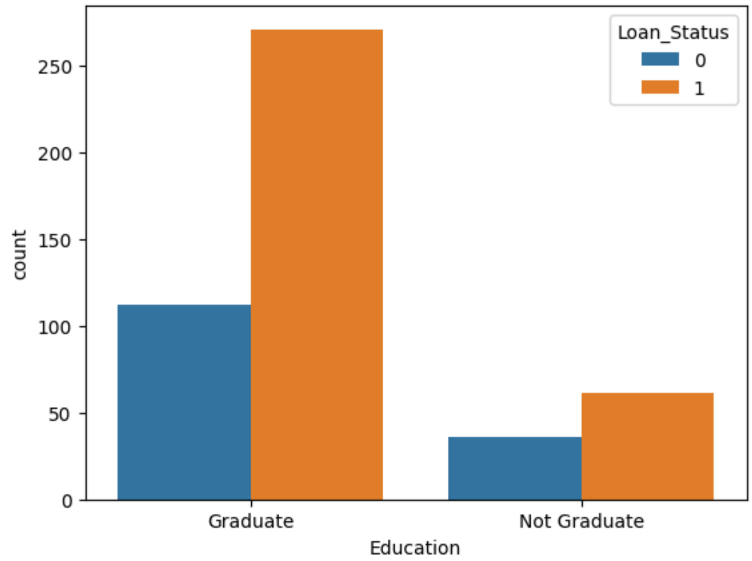
```
data['Dependents'].value_counts()
```

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

```
#DATA VISUALISATION
```

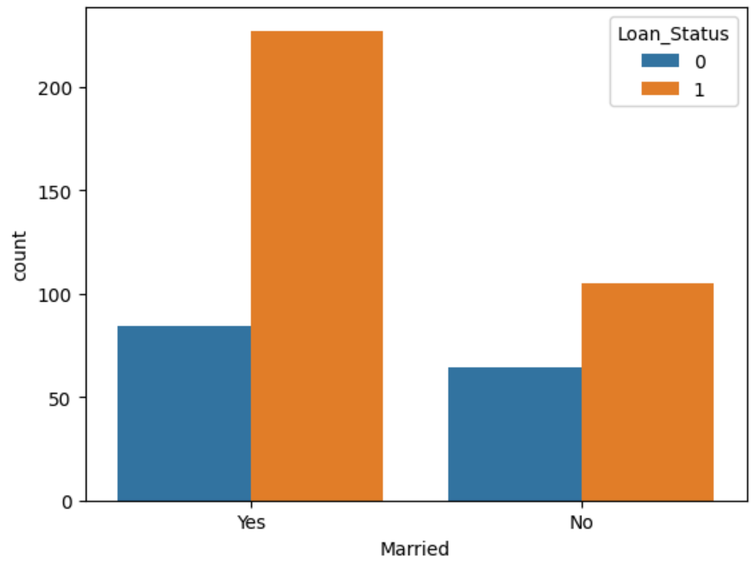
```
sns.countplot(x='Education', hue = 'Loan_Status', data = data)
```

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




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

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




```
#converting all categorical to numerical
```

```
data.replace({'Married': {'No':0, 'Yes':1}, 'Gender':{'Male':1, 'Female':0}, 'Self_Employed':{'No':0, 'Yes':1},
              'Property_Area':{'Rural':0, 'Semiurban':1, 'Urban':2}, 'Education':{'Graduate':1, 'Not Graduate':0}}, inplace = True)
```

```
data.head()
```

ı_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapplica
003	1	1	1	1	0	4583	
005	1	1	0	1	1	3000	
006	1	1	0	0	0	2583	
008	1	0	0	1	0	6000	
ı011	1	1	2	1	1	5417	

Next steps: Generate code with data View recommended plots

```
#separating the data and label

X = data.drop(columns = ['Loan_ID', 'Loan_Status'], axis = 1)
Y = data['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
```

```
#Splitting into training and testing

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.1, stratify=Y, random_state = 2)
```

```
print(X.shape, X_train.shape, X_test.shape)
```

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

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

```
#training the model

classifier.fit(X_train, Y_train)
```

```
▾       SVC
SVC(kernel='linear')
```

```
#model_evaluation

X_train_prediction = classifier.predict(X_train)
training_accuracy = accuracy_score(X_train_prediction, Y_train)
```

```
Start coding or generate with AI.
```

```
print("Accuracy on Training :" , training_accuracy)
```

```
Accuracy on Training : 0.7986111111111112
```

```
X_test_prediction = classifier.predict(X_test)
test_accuracy = accuracy_score(X_test_prediction, Y_test)
```

```
print("Accuracy on Testing :" , test_accuracy)
```

```
Accuracy on Testing : 0.8333333333333334
```

```
from sklearn.ensemble import RandomForestClassifier

rf_clf = RandomForestClassifier()
rf_clf.fit(X_train, Y_train)
```

```
▾ RandomForestClassifier
RandomForestClassifier()
```

```
X_test_prediction_2 = rf_clf.predict(X_test)
```

```
Accuracy_RF = accuracy_score(X_test_prediction_2, Y_test)
print("Accuracy: ", Accuracy_RF)
```

```
Accuracy:  0.8125
```

```
from sklearn.naive_bayes import GaussianNB
nb_clf = GaussianNB()
nb_clf.fit(X_train, Y_train)
```

```
▾ GaussianNB
GaussianNB()
```

```
X_test_prediction_3 = nb_clf.predict(X_test)
Accuracy_NB = accuracy_score(X_test_prediction_3, Y_test)
```

```
print("Accuracy for GuassianNB: ", Accuracy_NB)
```

```
Accuracy for GuassianNB:  0.8333333333333334
```

```
from sklearn.tree import DecisionTreeClassifier

dt_clf = DecisionTreeClassifier()
dt_clf.fit(X_train, Y_train)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

```
X_test_prediction_4 = dt_clf.predict(X_test)
Accuracy_DT = accuracy_score(X_test_prediction_4, Y_test)

print("Accuracy for DT: ", Accuracy_DT)
```

```
Accuracy for DT:  0.75
```

```
from sklearn.neighbors import KNeighborsClassifier

kn_clf = KNeighborsClassifier()
kn_clf.fit(X_train, Y_train)
```

```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

```
X_test_prediction_5 = kn_clf.predict(X_test)
Accuracy_KN = accuracy_score(X_test_prediction_5, Y_test)

print("Accuracy for KN: ", Accuracy_KN)
```

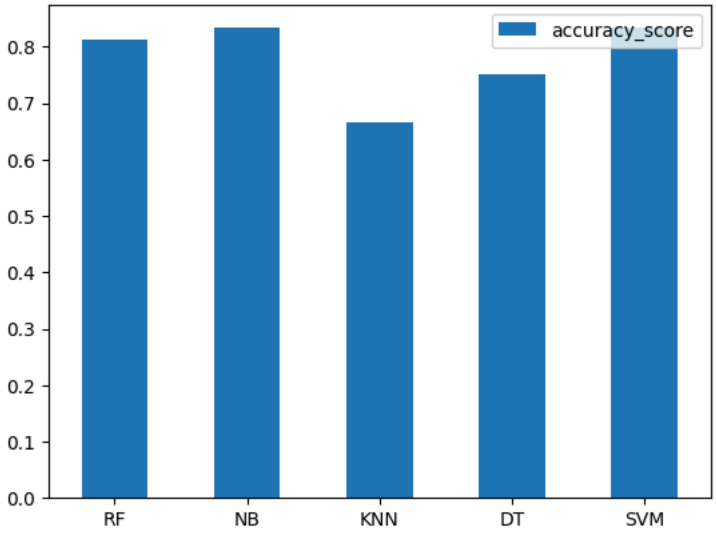
```
Accuracy for KN:  0.6666666666666666
```

```
import matplotlib.pyplot as plt

Y_set = [Accuracy_RF, Accuracy_NB, Accuracy_KN, Accuracy_DT, test_accuracy]
X_set = ['RF', 'NB', 'KNN', 'DT', 'SVM']

index = ['RF', 'NB', 'KNN', 'DT', 'SVM']

df = pd.DataFrame({'accuracy_score': Y_set, 'Models': X_set}, index = index)
ax = df.plot.bar(rot=0)
```




```
#INFERENCE
#puttin input data any values for prediction based on information given

#we can see from graph gaussianNB and SVM both have equal accuracy ( we can chose any 1 but since SVM takes bit time to train its recomme
input_data = (1,1,2,1,1,5417,4196,267,360,1,2)

input_data_as_array = np.asarray(input_data)

input_data_reshaped = input_data_as_array.reshape(1,-1)

prediction = nb_clf.predict(input_data_reshaped)
#print(prediction)

if(prediction[0]==1):
  print("Yes")
else:
  print("No")
```

```
Yes
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but GaussianNB was f
  warnings.warn(
```