In [ ]:
```
# Importing Libraries
```

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
# Loading Data Set
data = pd.read_csv("C:\\Users\\computer-8\\Downloads\\Final+Test+Data+Set (1).csv")
```

In [3]:
```
data
```

Out[3]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coap
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	
...	...	...	...	...	...	...	...	
609	LP002978	Female	No	0	Graduate	No	2900	
610	LP002979	Male	Yes	3+	Graduate	No	4106	
611	LP002983	Male	Yes	1	Graduate	No	8072	
612	LP002984	Male	Yes	2	Graduate	No	7583	
613	LP002990	Female	No	0	Graduate	Yes	4583	

614 rows × 13 columns

In [4]:
```
data.shape
```

Out[4]:
```
(614, 13)
```

In [5]:
```
data.columns
```

Out[5]:
```
Index(['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education',
       'Self_Employed', 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History', 'Property_Area', 'Loan_Status'],
      dtype='object')
```

In [6]:
```
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
```

In [7]:
```
#checking missing values
data.isnull().sum()
```

Out[7]:
```
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 [8]:
```
#Droping Nan Values
data = data.dropna()
```

In [9]:
```
data.isnull().sum()
```

Out[9]:
```
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 [10]:
```
data
```

Out[10]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coap
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	
...	...	...	...	...	...	...	...	...
609	LP002978	Female	No	0	Graduate	No	2900	
610	LP002979	Male	Yes	3+	Graduate	No	4106	
611	LP002983	Male	Yes	1	Graduate	No	8072	
612	LP002984	Male	Yes	2	Graduate	No	7583	
613	LP002990	Female	No	0	Graduate	Yes	4583	

480 rows × 13 columns

In [11]:
```
#after droping na values index number are changed so reshape them
data.reset_index(inplace=True)
```

In [12]:
```
# now we have a proper index number
data
```

Out[12]:

	index	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome
0	1	LP001003	Male	Yes	1	Graduate	No	458
1	2	LP001005	Male	Yes	0	Graduate	Yes	300
2	3	LP001006	Male	Yes	0	Not Graduate	No	258
3	4	LP001008	Male	No	0	Graduate	No	600
4	5	LP001011	Male	Yes	2	Graduate	Yes	541
...	...	...	...	...	...	...	...	..
475	609	LP002978	Female	No	0	Graduate	No	290
476	610	LP002979	Male	Yes	3+	Graduate	No	410
477	611	LP002983	Male	Yes	1	Graduate	No	807
478	612	LP002984	Male	Yes	2	Graduate	No	758
479	613	LP002990	Female	No	0	Graduate	Yes	458

480 rows × 14 columns

In [13]:
```
# checking unique values
data['Dependents'].unique()
```

Out[13]:
```
array(['1', '0', '2', '3+'], dtype=object)
```

In [14]:
```
data['Dependents'].value_counts()
```

Out[14]:
```
0     274
2      85
1      80
3+     41
Name: Dependents, dtype: int64
```

In [15]:
```
#replace 3+ to 4
data['Dependents']=data['Dependents'].replace(to_replace = '3+',value = 4)
```

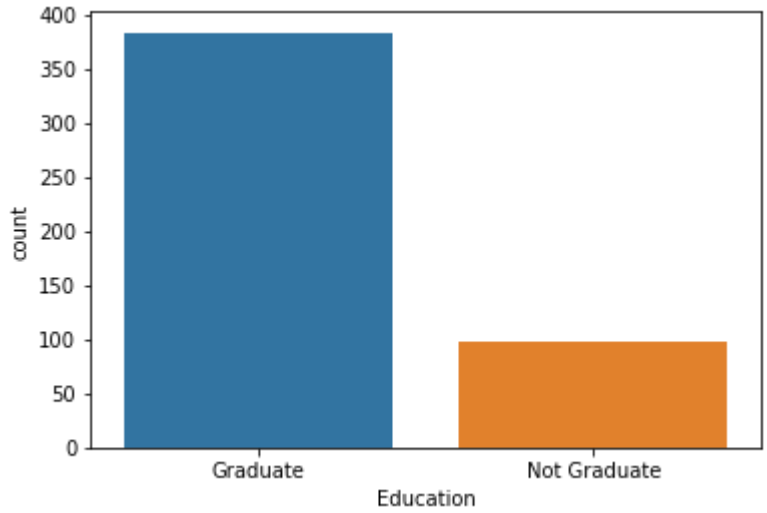
In [16]:
```
# after replacing
data['Dependents'].unique()
```

Out[16]:
```
array(['1', '0', '2', 4], dtype=object)
```

EDA

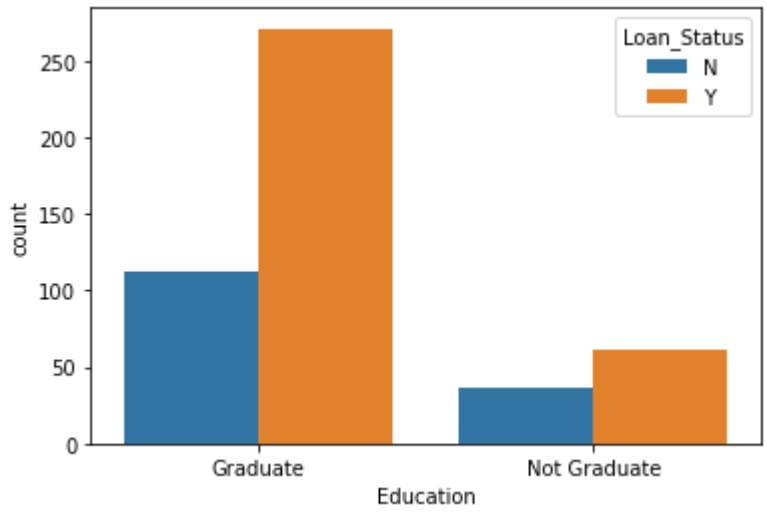
In [17]:
```
sns.countplot(data['Education'])
```

Out[17]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2d11b8b3700>
```



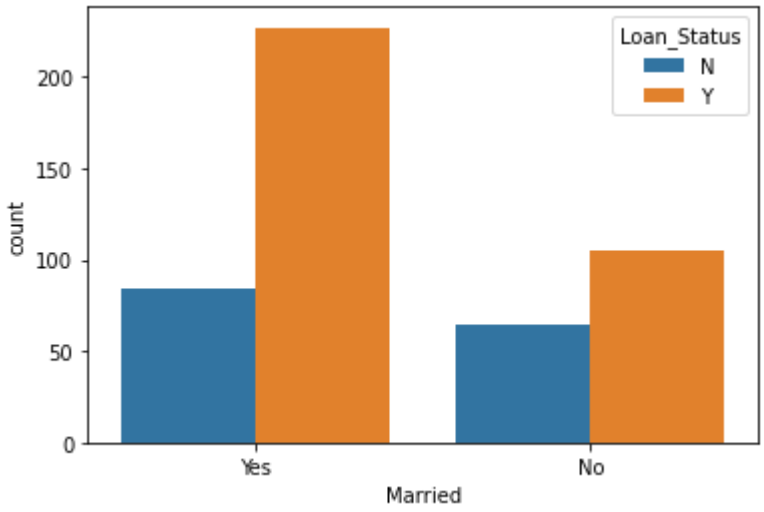
In [18]:
```
# comparing with education and loan status
sns.countplot(x='Education',hue='Loan_Status',data=data)
```

Out[18]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2d11b955eb0>
```



In [19]:
```
#married vs loan status
sns.countplot(x='Married',hue='Loan_Status',data=data)
```

Out[19]: <matplotlib.axes._subplots.AxesSubplot at 0x2d11b9bff70>



In [ ]:
```
# Male vs Female
sns.countplot(x=)
```

In [57]:
```
data
```

Out[57]:

	index	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome
0	1	LP001003	1	1	1	0	0	458
1	2	LP001005	1	1	0	0	1	300
2	3	LP001006	1	1	0	1	0	258
3	4	LP001008	1	0	0	0	0	600
4	5	LP001011	1	1	2	0	1	541
...	...	...	...	...	...	...	...	..
475	609	LP002978	0	0	0	0	0	290
476	610	LP002979	1	1	4	0	0	410
477	611	LP002983	1	1	1	0	0	807
478	612	LP002984	1	1	2	0	0	758
479	613	LP002990	0	0	0	0	1	458

480 rows × 14 columns

Label Encoding

In [20]:
```
from sklearn.preprocessing import LabelEncoder
lb= LabelEncoder()
```

In [21]:
```
data['Married']=lb.fit_transform(data['Married'])
data['Gender']=lb.fit_transform(data['Gender'])
data['Education']=lb.fit_transform(data['Education'])
data['Self_Employed']=lb.fit_transform(data['Self_Employed'])
data['Property_Area']=lb.fit_transform(data['Property_Area'])
data['Loan_Status']=lb.fit_transform(data['Loan_Status'])

#
```

In [22]:
```
data
```

Out[22]:

	index	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome
0	1	LP001003	1	1	1	0	0	458
1	2	LP001005	1	1	0	0	1	300
2	3	LP001006	1	1	0	1	0	258
3	4	LP001008	1	0	0	0	0	600
4	5	LP001011	1	1	2	0	1	541
...	...	...	...	...	...	...	...	..
475	609	LP002978	0	0	0	0	0	290
476	610	LP002979	1	1	4	0	0	410
477	611	LP002983	1	1	1	0	0	807
478	612	LP002984	1	1	2	0	0	758
479	613	LP002990	0	0	0	0	1	458

480 rows × 14 columns

In [23]:
```
# yes =1 ,no =0 ,male =1 female=0,garduate=0,no grd=1, urban =2,rual =0 semi u
rban =1
```

In [24]:
```
data['Dependents'] = data['Dependents'].astype('int')
```

In [25]:
```
# features
#X is independent
X = data.iloc[:,2:-1].values
```

In [26]:
```
#Dependent variable y
Y = data.iloc[:,-1]
Y
```

Out[26]:
```
0      0
1      1
2      1
3      1
4      1
      ..
475    1
476    1
477    1
478    1
479    0
Name: Loan_Status, Length: 480, dtype: int32
```

In [ ]:
```

```

Modeling

In [27]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(X,Y,test_size=0.25,random_state=42)
```

In [28]:
```
x_train.shape,y_train.shape,x_test.shape,y_test.shape
```

Out[28]:
```
((360, 11), (360,), (120, 11), (120,))
```

In [29]:
```
y_test
```

Out[29]:
```
73     1
414    1
394    1
277    0
399    1
      ..
57     1
176    0
290    0
402    1
24     0
Name: Loan_Status, Length: 120, dtype: int32
```

Logistic Regression

In [30]:
```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
```

In [31]:
```
lr.fit(x_train,y_train)
```

Out[31]:
```
LogisticRegression()
```

In [ ]:
```

```

In [32]:
```
pred = lr.predict(x_test)
pred
```

Out[32]:
```
array([1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 1, 1])
```

In [33]:
```
lr.score(x_test,y_test)
```

Out[33]:
```
0.775
```

KNN

In [34]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
knn.fit(x_train,y_train)
```

Out[34]:
```
KNeighborsClassifier()
```

In [35]:
```
ky_pred=knn.predict(x_test)
```

In [36]:
```
ky_pred
```

Out[36]:
```
array([1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1,
       1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 1, 0])
```

In [37]:
```
knn.score(x_test,y_test)
```

Out[37]:
```
0.6166666666666667
```

In [38]:
```
#logistic is best fit model
```

In [39]:
```
from sklearn.metrics import classification_report
```

In [40]:
```
print(classification_report(y_test,ky_pred))
```

```
              precision    recall  f1-score   support

           0       0.33      0.21      0.26        38
           1       0.69      0.80      0.74        82

    accuracy                           0.62       120
   macro avg       0.51      0.51      0.50       120
weighted avg       0.58      0.62      0.59       120
```

Decision Tree

In [41]:
```
from sklearn.tree import DecisionTreeClassifier
dc = DecisionTreeClassifier(criterion='gini')
```

In [42]:
```
dc.fit(x_train,y_train)
```

Out[42]:
```
DecisionTreeClassifier()
```

In [43]:
```
dc_pred=dc.predict(x_test)
dc_pred
```

Out[43]:
```
array([1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1,
       0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0,
       1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0,
       1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0,
       0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 1, 1])
```

In [44]:
```
dc.score(x_test,dc_pred)
```

Out[44]:
```
1.0
```

In [45]:
```
from sklearn.metrics import classification_report
```

In [46]:
```
print(classification_report(y_test,dc_pred))
```

```
              precision    recall  f1-score   support

           0       0.59      0.45      0.51        38
           1       0.77      0.85      0.81        82

    accuracy                           0.73       120
   macro avg       0.68      0.65      0.66       120
weighted avg       0.71      0.72      0.71       120
```

In [ ]:
```

```

Random Forest

In [47]:
```
from sklearn.ensemble import RandomForestClassifier
rc = RandomForestClassifier(n_estimators=90)
rc.fit(x_train,y_train)
rc_pred = rc.predict(x_test)
rc.score(x_test,y_test)
```

Out[47]: 0.7833333333333333

SVM

In [48]:
```
from sklearn.svm import SVC
svc = SVC(kernel='linear')
```

In [49]:
```
svc.fit(x_train,y_train)
```

Out[49]: SVC(kernel='linear')

In [50]:
```
svc_pred = svc.predict(x_test)
```

In [51]:
```
#Confusion matrix
from sklearn.metrics import confusion_matrix
```

In [52]:
```
cm = confusion_matrix(y_test,svc_pred)
```

In [53]:
```
cm
```

Out[53]:
```
array([[11, 27],
       [ 0, 82]], dtype=int64)
```

In [54]:
```
print(classification_report(y_test,svc_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.29      0.45        38
           1       0.75      1.00      0.86        82

    accuracy                           0.78       120
   macro avg       0.88      0.64      0.65       120
weighted avg       0.83      0.78      0.73       120
```

In [ ]:
```

```

In [55]:
```
from sklearn.ensemble import VotingClassifier
```

In [56]:
```
vc = VotingClassifier(estimators=[('rc',RandomForestClassifier()),
                                  ('dc',DecisionTreeClassifier()),
                                  ('lr',LogisticRegression())])
```

In [58]:
```
vc.fit(x_train,y_train)
```

Out[58]:
```
VotingClassifier(estimators=[('rc', RandomForestClassifier()),
                             ('dc', DecisionTreeClassifier()),
                             ('lr', LogisticRegression())])
```

In [59]:
```
vc_pred = vc.predict(x_test)
```

In [61]:
```
print(classification_report(y_test,vc_pred))
```

```
              precision    recall  f1-score   support

           0       0.88      0.39      0.55        38
           1       0.78      0.98      0.86        82

    accuracy                           0.79       120
   macro avg       0.83      0.69      0.71       120
weighted avg       0.81      0.79      0.76       120
```

K-MEANS

In [62]:
```
from sklearn.cluster import KMeans
```

In [63]:
```
kmeans = KMeans()
```

In [64]:
```
kmeans.fit(x_train,y_train)
```

Out[64]:
```
KMeans()
```

In [65]:
```
kmeans_pred = kmeans.predict(x_test)
```

In [66]:
```
print(classification_report(y_test,kmeans_pred))
```

```
              precision    recall  f1-score   support

           0       0.40      0.50      0.44        38
           1       0.60      0.04      0.07        82
           5       0.00      0.00      0.00         0
           6       0.00      0.00      0.00         0
           7       0.00      0.00      0.00         0

    accuracy                           0.18       120
   macro avg       0.20      0.11      0.10       120
weighted avg       0.54      0.18      0.19       120
```

Decision Tree and Logistic Regression are best so i prefered these models to this problem