In [1]:

```
# Importing some basic libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import GridSearchCV

import warnings
warnings.filterwarnings('ignore')
```

In [2]:

```
# Loading Our dataset

df = pd.read_csv('madfhantr.csv')
df.head()
```

Out[2]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	Loa
0	LP001002	Male	No	0	Graduate	No	5849	0.0	
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	
4	LP001008	Male	No	0	Graduate	No	6000	0.0	

In [3]:

```
df.shape
```

Out[3]:

```
(614, 13)
```

In [4]:

```
df.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 [5]:

```
df.columns
```

```
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]:

```
# Dropping loan_ID as it has no significance in prediction

df = df.drop('Loan_ID',axis=1)
```

In [7]:

```
# Checking for Missing values

df.isnull().sum()
```

Out[7]:

```
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 see that there is huge amount of missing data points.

In [8]:

```
# Getting percentage of missing values

missing_per = (df.isnull().sum()/df.shape[0])*100
missing_per
```

Out[8]:

```
Gender               2.117264
Married              0.488599
Dependents           2.442997
Education            0.000000
Self_Employed        5.211726
ApplicantIncome      0.000000
CoapplicantIncome    0.000000
LoanAmount           3.583062
Loan_Amount_Term     2.280130
Credit_History       8.143322
Property_Area        0.000000
Loan_Status          0.000000
dtype: float64
```

In [9]:

```
# Getting catetgorical columns and numerical columns

cat_cols = [x for x in df.columns if df[x].dtypes == 'object']

num_cols = [x for x in df.columns if df[x].dtypes != 'object']
```

In [10]:

```
print(cat_cols)
```

```
['Gender', 'Married', 'Dependents', 'Education', 'Self_Employed', 'Property_Area', 'Loan_S
tatus']
```

In [11]:

```
print(num_cols)
```

```
['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount', 'Loan_Amount_Term', 'Credit_Histor
y']
```

lets first deal with missing values

In [12]:
```python
# Getting counts of values for categorical features

lst = ['Gender','Married','Dependents','Education','Self_Employed','Property_Area']

for feature in lst:
    print(feature)
    print(df[feature].value_counts())
    print('='*30,'\n')
```

```
Gender
Male      489
Female    112
Name: Gender, dtype: int64
==============================

Married
Yes    398
No     213
Name: Married, dtype: int64
==============================

Dependents
0     345
1     102
2     101
3+     51
Name: Dependents, dtype: int64
==============================

Education
Graduate        480
Not Graduate    134
Name: Education, dtype: int64
==============================

Self_Employed
No     500
Yes     82
Name: Self_Employed, dtype: int64
==============================

Property_Area
Semiurban    233
Urban        202
Rural        179
Name: Property_Area, dtype: int64
==============================
```

In [13]:
```python
# filling missing values

df['Gender'].fillna('Female',inplace=True)
df['Married'].fillna('No',inplace=True)
df['Dependents'].fillna('3+',inplace=True)
df['Self_Employed'].fillna('Yes',inplace=True)
df['Credit_History'].fillna(0,inplace=True)
```

In [14]:
```
df.isnull().sum()
```

Out[14]:
```
Gender                0
Married               0
Dependents            0
Education             0
Self_Employed         0
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           22
Loan_Amount_Term     14
Credit_History        0
Property_Area         0
Loan_Status           0
dtype: int64
```

In [15]:
```
# filling missing values with mean

df['LoanAmount'] = df['LoanAmount'].fillna(df['LoanAmount'].mean())
df['Loan_Amount_Term'] = df['Loan_Amount_Term'].fillna(df['Loan_Amount_Term'].mean())
```

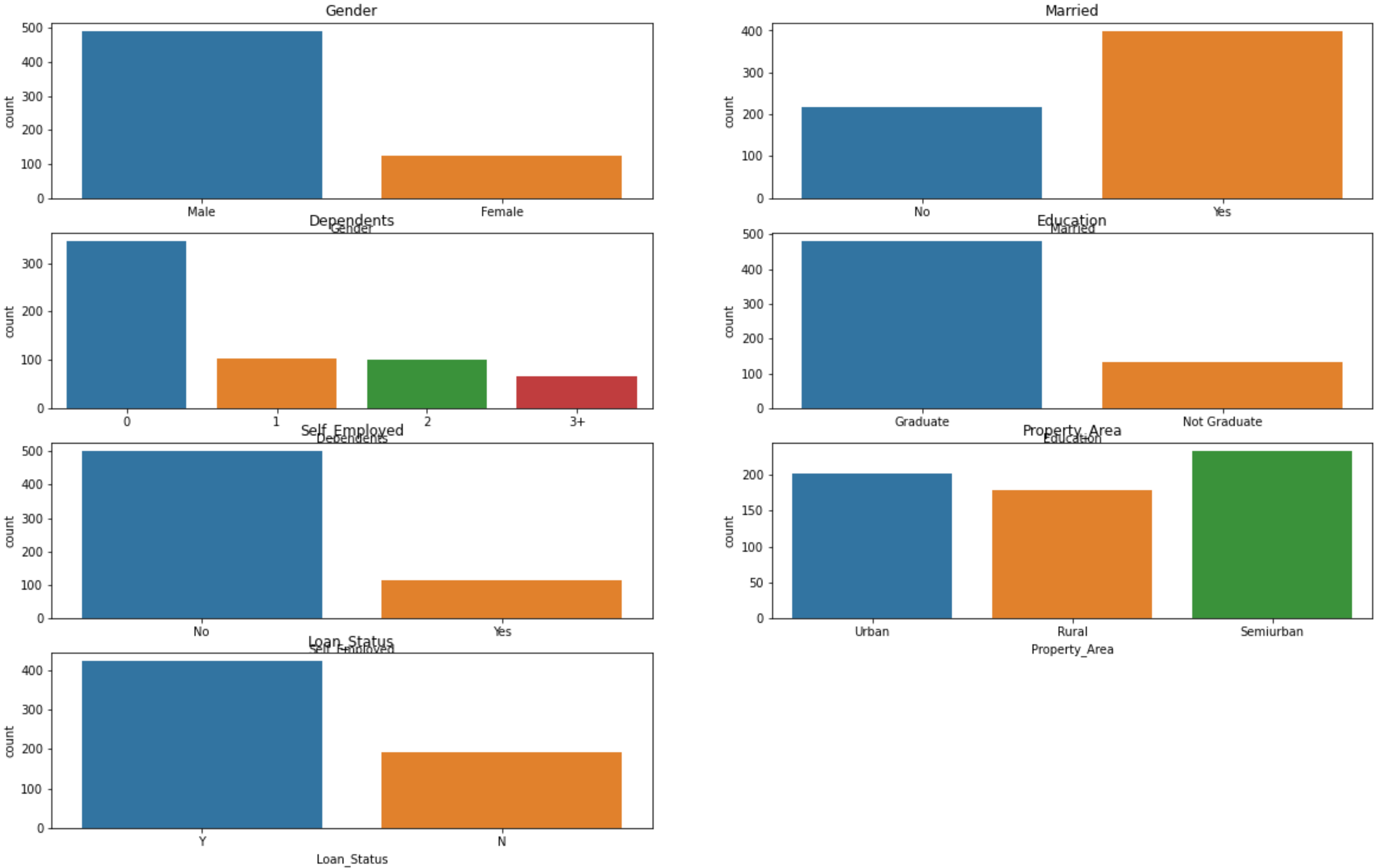
In [16]:
```
df.describe()
```

Out[16]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	614.000000	614.000000	614.000000	614.000000	614.000000
mean	5403.459283	1621.245798	146.412162	342.000000	0.773616
std	6109.041673	2926.248369	84.037468	64.372489	0.418832
min	150.000000	0.000000	9.000000	12.000000	0.000000
25%	2877.500000	0.000000	100.250000	360.000000	1.000000
50%	3812.500000	1188.500000	129.000000	360.000000	1.000000
75%	5795.000000	2297.250000	164.750000	360.000000	1.000000
max	81000.000000	41667.000000	700.000000	480.000000	1.000000

Visualisation

In [17]:
```
i = 1
plt.figure(figsize=(20,19))
for feature in cat_cols:
    plt.subplot(6,2,i)
    plt.title(feature)
    sns.countplot(df[feature])
    i+=1
```

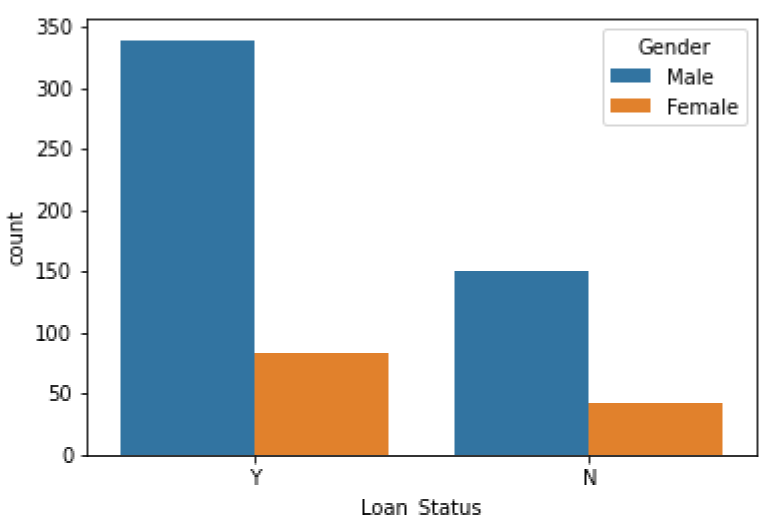


In [18]:
```
df.columns
```

Out[18]:
```
Index(['Gender', 'Married', 'Dependents', 'Education', 'Self_Employed',
       'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History', 'Property_Area', 'Loan_Status'],
      dtype='object')
```

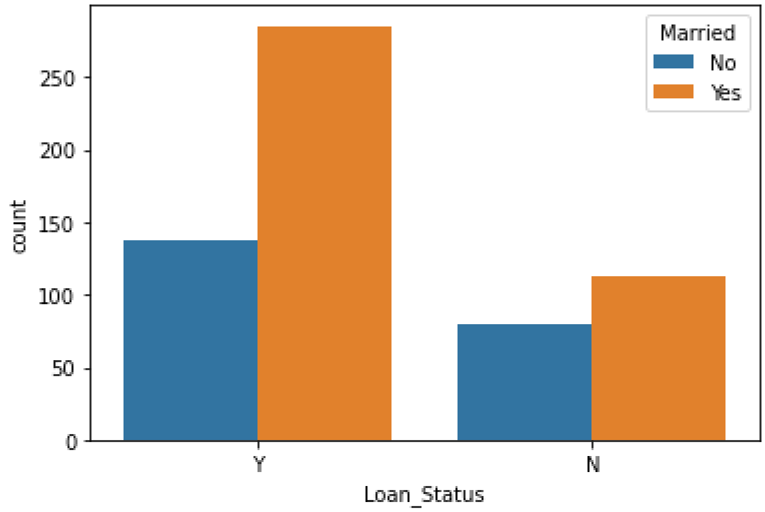
In [19]:
```
sns.countplot(df['Loan_Status'],hue=df['Gender'])
```

Out[19]:
```
<AxesSubplot:xlabel='Loan_Status', ylabel='count'>
```



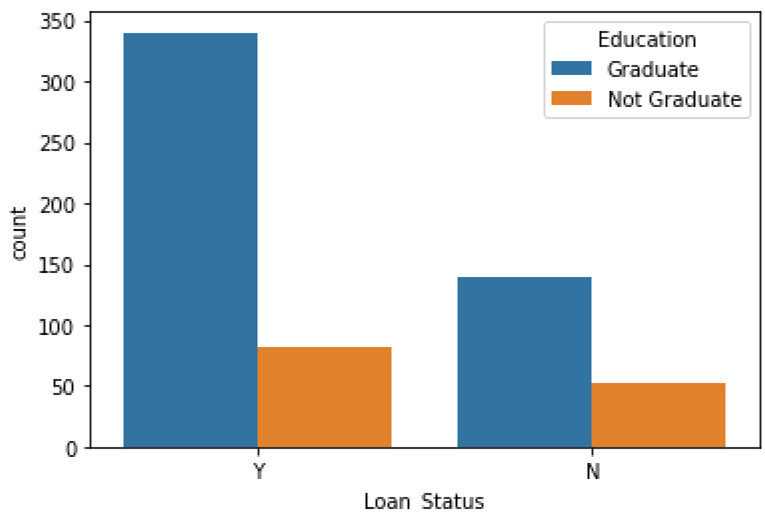
In [20]:
```
sns.countplot(df['Loan_Status'],hue=df['Married'])
```

Out[20]:
```
<AxesSubplot:xlabel='Loan_Status', ylabel='count'>
```



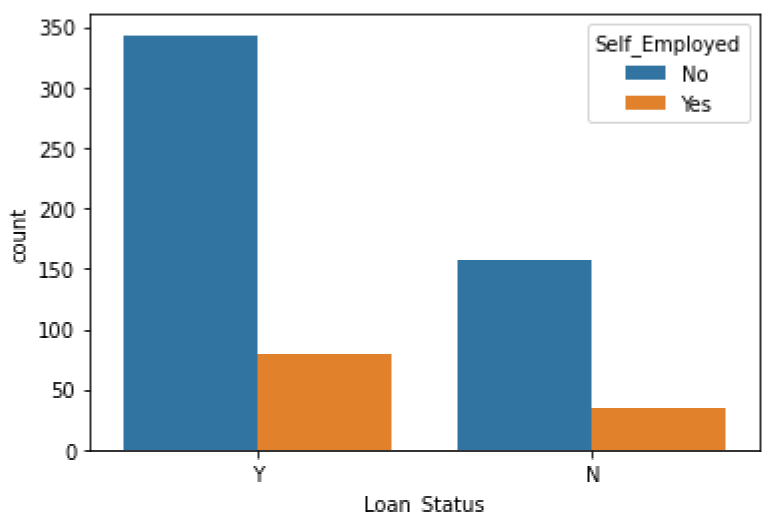
In [21]:
```
sns.countplot(df['Loan_Status'],hue=df['Education'])
```

Out[21]: <AxesSubplot:xlabel='Loan_Status', ylabel='count'>



In [22]:
```
sns.countplot(df['Loan_Status'],hue=df['Self_Employed'])
```

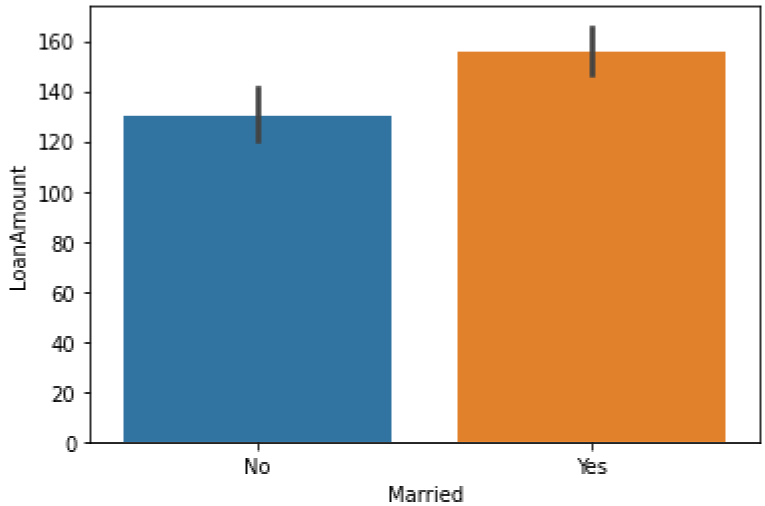
Out[22]: <AxesSubplot:xlabel='Loan_Status', ylabel='count'>



In [23]:
```
sns.barplot(x=df['Married'],y=df['LoanAmount'],data=df)
```

 label='LoanAmount'>

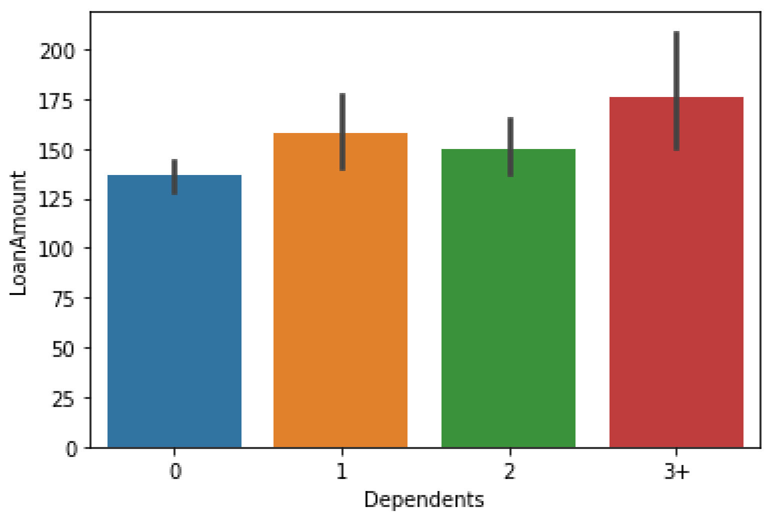
Out[23]:



In [24]:

```
sns.barplot(x=df['Dependents'],y=df['LoanAmount'],data=df)
```

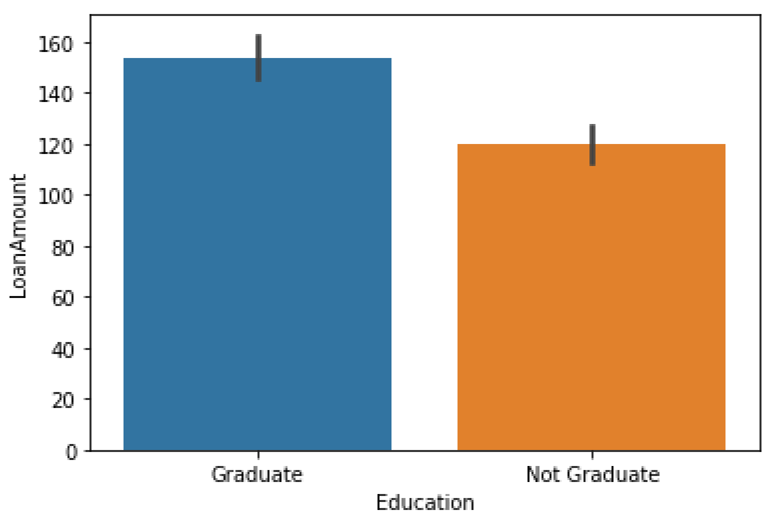
Out[24]: <AxesSubplot:xlabel='Dependents', ylabel='LoanAmount'>



In [25]:

```
sns.barplot(x=df['Education'],y=df['LoanAmount'],data=df)
```

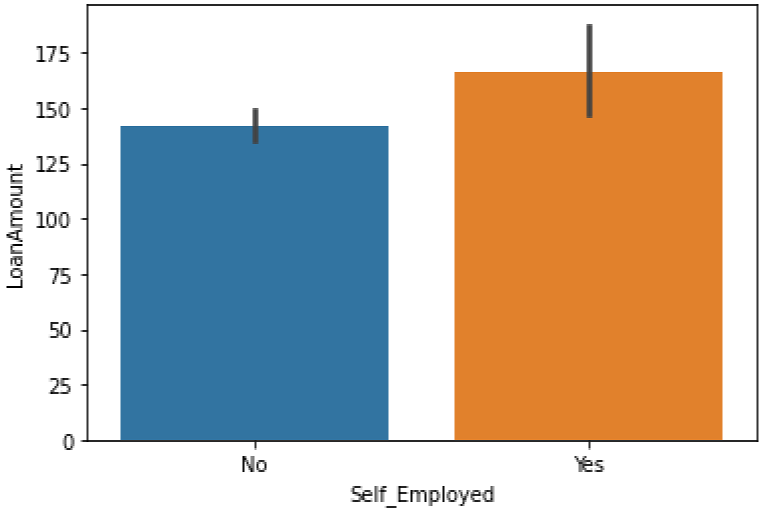
Out[25]: <AxesSubplot:xlabel='Education', ylabel='LoanAmount'>



In [26]:

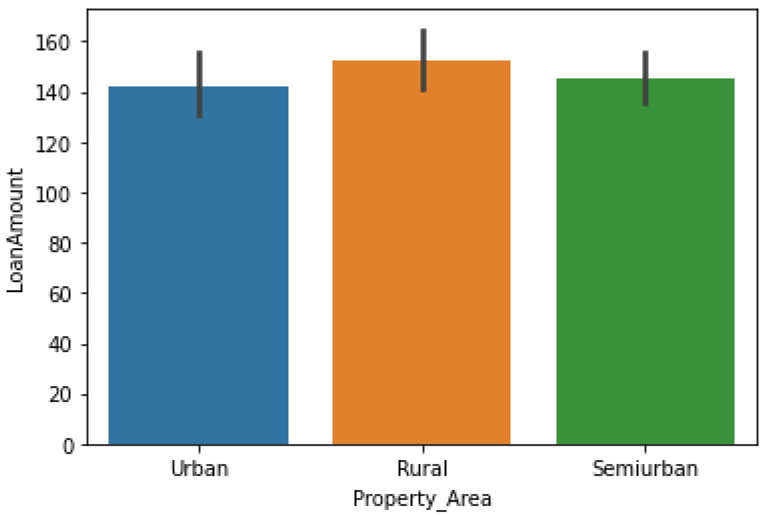
```
sns.barplot(x=df['Self_Employed'],y=df['LoanAmount'],data=df)
```

Out[26]: <AxesSubplot:xlabel='Self_Employed', ylabel='LoanAmount'>



In [27]:
```
sns.barplot(x=df['Property_Area'],y=df['LoanAmount'],data=df)
```

Out[27]: <AxesSubplot:xlabel='Property_Area', ylabel='LoanAmount'>



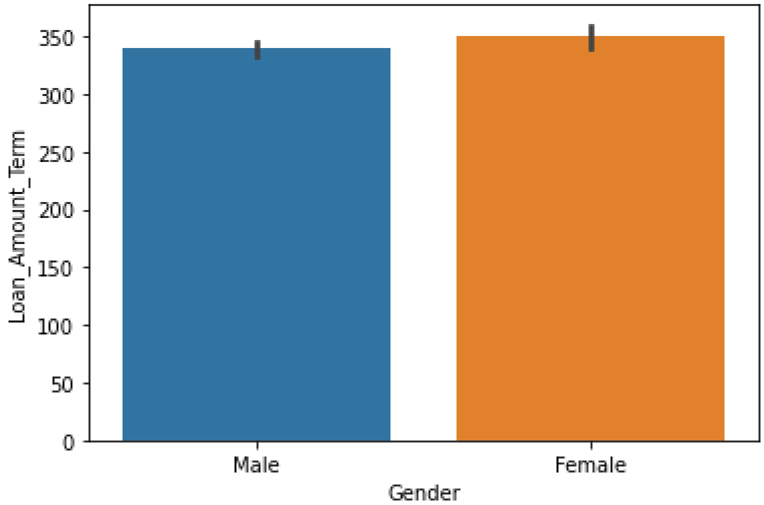
Observation

1. Maximum loan amount are given to persons which are male,graduate,self employed and has 3+ dependent persons having property area in rural.

In [28]:
```
sns.barplot(x=df['Gender'],y=df['Loan_Amount_Term'])
```

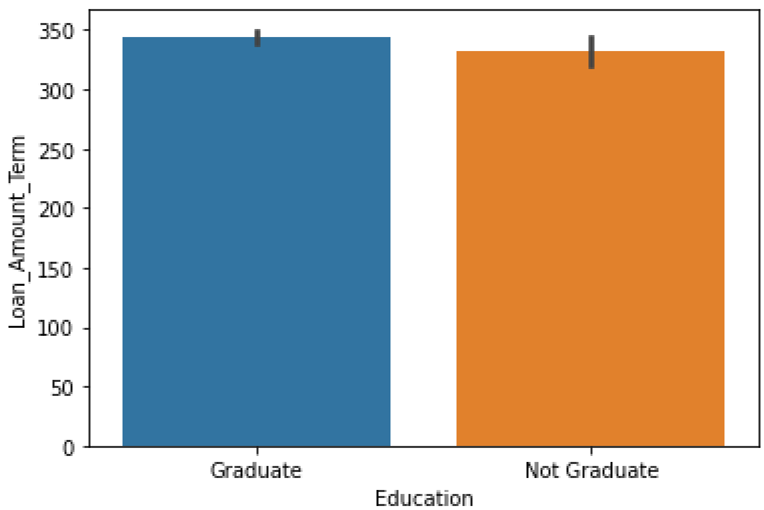
Out[28]: <AxesSubplot:xlabel='Gender', ylabel='Loan_Amount_Term'>

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



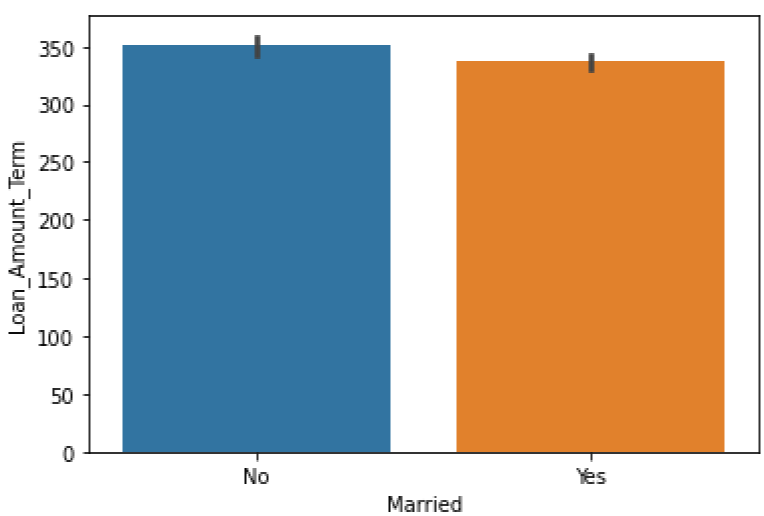
In [29]:
```
sns.barplot(x=df['Education'],y=df['Loan_Amount_Term'])
```

Out[29]: <AxesSubplot:xlabel='Education', ylabel='Loan_Amount_Term'>



In [30]:
```
sns.barplot(x=df['Married'],y=df['Loan_Amount_Term'])
```

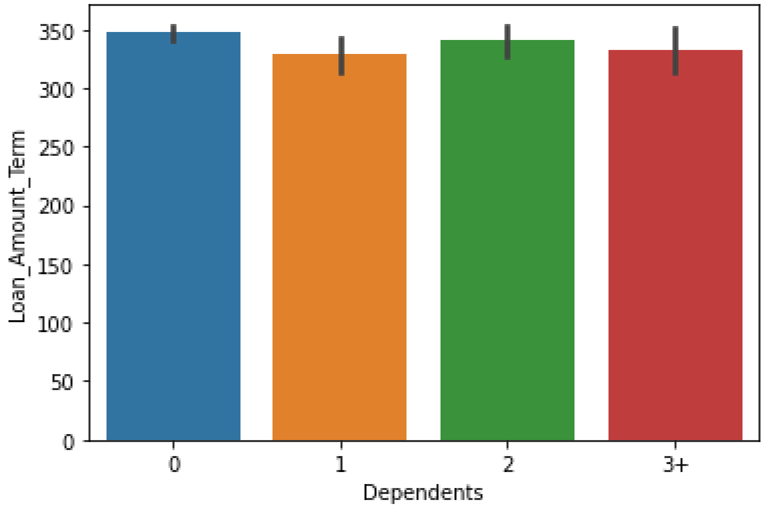
Out[30]: <AxesSubplot:xlabel='Married', ylabel='Loan_Amount_Term'>



In [31]:
```
sns.barplot(x=df['Dependents'],y=df['Loan_Amount_Term'])
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js , ylabel='Loan_Amount_Term'>

Out[31]:



Observation

1. Person which is male, graduate, no person dependent has long loan amount term.

In [32]:

```
df.head()
```

Out[32]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	L
0	Male	No	0	Graduate	No	5849	0.0	146.412162	
1	Male	Yes	1	Graduate	No	4583	1508.0	128.000000	
2	Male	Yes	0	Graduate	Yes	3000	0.0	66.000000	
3	Male	Yes	0	Not Graduate	No	2583	2358.0	120.000000	
4	Male	No	0	Graduate	No	6000	0.0	141.000000	

In [33]:

```
df[['Gender','Education','Married','Dependents','Self_Employed','Property_Area']].astype('
```

Out[33]:

	Gender	Education	Married	Dependents	Self_Employed	Property_Area
0	Male	Graduate	No	0	No	Urban
1	Male	Graduate	Yes	1	No	Rural
2	Male	Graduate	Yes	0	Yes	Urban
3	Male	Not Graduate	Yes	0	No	Urban
4	Male	Graduate	No	0	No	Urban
...	...	...	...	...	...	...
609	Female	Graduate	No	0	No	Rural
610	Male	Graduate	Yes	3+	No	Rural
611	Male	Graduate	Yes	1	No	Urban
612	Male	Graduate	Yes	2	No	Urban
613	Female	Graduate	No	0	Yes	Semiurban

614 rows × 6 columns

Label Encoding

In [34]:

```
lst = [['Gender','Education','Married','Dependents','Self_Employed','Property_Area']]

from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['Gender'] = le.fit_transform(df['Gender'])
df['Education'] = le.fit_transform(df['Education'])
df['Married'] = le.fit_transform(df['Married'])
df['Dependents'] = le.fit_transform(df['Dependents'])
df['Self_Employed'] = le.fit_transform(df['Self_Employed'])
df['Property_Area'] = le.fit_transform(df['Property_Area'])
df['Loan_Status'] = le.fit_transform(df['Loan_Status'])
```

In [35]:

```
df.head()
```

Out[35]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	L
0	1	0	0	0	0	5849	0.0	146.412162	
1	1	1	1	0	0	4583	1508.0	128.000000	
2	1	1	0	0	1	3000	0.0	66.000000	
3	1	1	0	1	0	2583	2358.0	120.000000	
4	1	0	0	0	0	6000	0.0	141.000000	

Scaling our features

In [36]:

```
def rescale(data,lista):
    for i in lista:
        data[i] = ((data[i]-data[i].min())/(data[i].max()-data[i].min()))

lst =["ApplicantIncome","CoapplicantIncome","LoanAmount","Loan_Amount_Term"]
rescale(df,lst)
df.head()
```

Out[36]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	L
0	1	0	0	0	0	0.070489	0.000000	0.198860	
1	1	1	1	0	0	0.054830	0.036192	0.172214	
2	1	1	0	0	1	0.035250	0.000000	0.082489	
3	1	1	0	1	0	0.030093	0.056592	0.160637	
4	1	0	0	0	0	0.072356	0.000000	0.191027	

In [37]:

```
# Splitting our data into independent and dependent variable

X = df.drop('Loan_Status',axis=1)
y = df['Loan_Status']
```

In [38]:

```
# Train Test Split

from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3,random_state=0)
```

Model Creation

In [39]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
```

In [40]:

```
model_1 = LogisticRegression()
model_1.fit(X_train,y_train)
```

Out[40]:

```
LogisticRegression()
```

In [41]:

```
y_pred_1 = model_1.predict(X_test)
y_pred_1
```

Out[41]:

```
array([1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1,
       1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1,
       0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1])
```

In [42]:

```
model_2 = DecisionTreeClassifier()
model_2.fit(X_train,y_train)
```

Out[42]:

```
DecisionTreeClassifier()
```

In [43]:

```
y_pred_2 = model_2.predict(X_test)
y_pred_2
```

Out[43]:

```
array([1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1,
       0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1,
       1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0,
       1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0,
       1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 1])
```

In [44]:

```
model_3 = RandomForestClassifier()
model_3.fit(X_train,y_train)
y_pred_3 = model_3.predict(X_test)
y_pred_3
```

Out[44]:

```
array([1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1,
       1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1,
       0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0,
```

```
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1])
```

In [45]:

```python
model_4 = GradientBoostingClassifier()
model_4.fit(X_train,y_train)
y_pred_4 = model_4.predict(X_test)
y_pred_4
```

Out[45]:

```
array([1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1,
       0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
       1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1])
```

Model Evaluation

In [46]:

```python
from sklearn.metrics import accuracy_score

print(str(model_1))
accuracy_model1 = accuracy_score(y_test,y_pred_1)
print(accuracy_model1)

print(str(model_2))
accuracy_model2 = accuracy_score(y_test,y_pred_2)
print(accuracy_model2)

print(str(model_3))
accuracy_model3 = accuracy_score(y_test,y_pred_3)
print(accuracy_model3)

print(str(model_4))
accuracy_model4 = accuracy_score(y_test,y_pred_4)
print(accuracy_model4)
```

```
LogisticRegression()
0.7891891891891892
DecisionTreeClassifier()
0.6702702702702703
RandomForestClassifier()
0.7405405405405405
GradientBoostingClassifier()
0.7675675675675676
```

Hyperparameter Tunning

In [47]:

```python
# hypertune Decision tree

params={'max_depth':[4,8,12],
       'min_samples_split':[6,12,16]}

grid_dt = GridSearchCV(model_2,param_grid=params,cv=5,verbose=1,n_jobs=-1)
grid_dt.fit(X_train,y_train)
ypreddt = grid_dt.predict(X_test)
ypreddt
```



```
ndidates, totalling 45 fits
```

```
Out[47]: array([1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1,
       1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1,
       0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 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, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1])
```

```
In [48]: grid_dt.best_params_
```

```
Out[48]: {'max_depth': 4, 'min_samples_split': 16}
```

```
In [49]: accracy_grid_dt = accuracy_score(y_test,ypreddt)
         accracy_grid_dt
```

```
Out[49]: 0.7621621621621621
```

```
In [51]: # hypertune Random Forest

         params = {'n_estimators':[100,200],
                   'max_depth':[4,6,8],
                   'min_samples_leaf':[6,10,14],
                   'min_samples_split':[4,6,8]}

         grid_rf = GridSearchCV(model_3,param_grid=params,cv=5,verbose=1,n_jobs=-1)
         grid_rf.fit(X_train,y_train)
         ypredrf = grid_rf.predict(X_test)
         ypredrf
```

```
Fitting 5 folds for each of 54 candidates, totalling 270 fits
Out[51]: array([1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1,
       0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 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, 1, 1, 1, 1,
       1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1])
```

```
In [52]: grid_rf.best_params_
```

```
Out[52]: {'max_depth': 8,
          'min_samples_leaf': 14,
          'min_samples_split': 6,
          'n_estimators': 100}
```

```
In [53]: grid_rf.best_score_
```

```
Out[53]: 0.7552667578659371
```

```
In [54]: # Hypertune Gradient boosting

         params = {'n_estimators':[100,200],
                   'learning_rate':[0.05,0.1,0.2]}
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```python
grid_gb = GridSearchCV(model_4,param_grid=params,cv=5,verbose=1,n_jobs=-1)
grid_gb.fit(X_train,y_train)
ypredgb = grid_gb.predict(X_test)
ypredgb
```

```
Fitting 5 folds for each of 6 candidates, totalling 30 fits
```

Out[54]:

```
array([1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1,
       1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1,
       1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
       1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1])
```

In [55]:

```python
grid_gb.best_params_
```

Out[55]:

```
{'learning_rate': 0.05, 'n_estimators': 100}
```

In [56]:

```python
grid_gb.best_score_
```

Out[56]:

```
0.7086183310533516
```

In [ ]:

```python

```