Dataset Information

Dream Housing Finance company deals in all home loans. They have presence across all urban, semi urban and rural areas. Customer first apply for home loan after that company validates the customer eligibility for loan. Company wants to automate the loan eligibility process (real time) based on customer detail provided while filling online application form. These details are Gender, Marital Status, Education, Number of Dependents, Income, Loan Amount, Credit History and others. To automate this process, they have given a problem to identify the customers segments, those are eligible for loan amount so that they can specifically target these customers.

This is a standard supervised classification task.A classification problem where we have to predict whether a loan would be approved or not. Below is the dataset attributes with description.

Variable	Description
Loan_ID	Unique Loan ID
Gender	Male/ Female
Married	Applicant married (Y/N)
Dependents	Number of dependents
Education	Applicant Education (Graduate/ Under Graduate)
Self_Employed	Self employed (Y/N)
ApplicantIncome	Applicant income
CoapplicantIncome	Coapplicant income
LoanAmount	Loan amount in thousands
Loan_Amount_Term	Term of loan in months
Credit_History	credit history meets guidelines
Property_Area	Urban/ Semi Urban/ Rural
Loan_Status	Loan approved (Y/N)

Import modules

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
from matplotlib import pyplot as plt
import matplotlib
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

Loading the dataset

In [2]:
```
df = pd.read_csv("Loan Prediction Dataset.csv")
df.head()
```

Out[2]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
0	LP001002	Male	No	0	Graduate	No	5849	0.0	NaN	360.0	1.0	Urban	Y
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.0	1.0	Rural	N
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	360.0	1.0	Urban	Y
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0	1.0	Urban	Y
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	360.0	1.0	Urban	Y

In [3]:
```
df.describe()
```

Out[3]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	614.000000	614.000000	592.000000	600.00000	564.000000
mean	5403.459283	1621.245798	146.412162	342.00000	0.842199
std	6109.041673	2926.248369	85.587325	65.12041	0.364878
min	150.000000	0.000000	9.000000	12.00000	0.000000
25%	2877.500000	0.000000	100.000000	360.00000	1.000000
50%	3812.500000	1188.500000	128.000000	360.00000	1.000000
75%	5795.000000	2297.250000	168.000000	360.00000	1.000000
max	81000.000000	41667.000000	700.000000	480.00000	1.000000

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
```

Preprocessing the dataset

In [5]:
```
# find the null values
df.isnull().sum()
```

Out[5]:
```
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 [6]:
```
# fill the missing values for numerical terms - mean
df['LoanAmount'] = df['LoanAmount'].fillna(df['LoanAmount'].mean())
df['Loan_Amount_Term'] = df['Loan_Amount_Term'].fillna(df['Loan_Amount_Term'].mean())
df['Credit_History'] = df['Credit_History'].fillna(df['Credit_History'].mean())
```

In [7]:
```
# fill the missing values for categorical terms - mode
df['Gender'] = df["Gender"].fillna(df['Gender'].mode()[0])
df['Married'] = df["Married"].fillna(df['Married'].mode()[0])
df['Dependents'] = df["Dependents"].fillna(df['Dependents'].mode()[0])
df['Self_Employed'] = df["Self_Employed"].fillna(df['Self_Employed'].mode()[0])
```

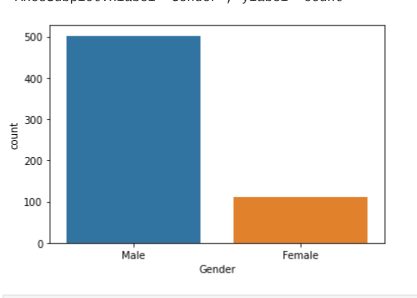
In [8]:
```
df.isnull().sum()
```

Out[8]:
```
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
```

Exploratory Data Analysis

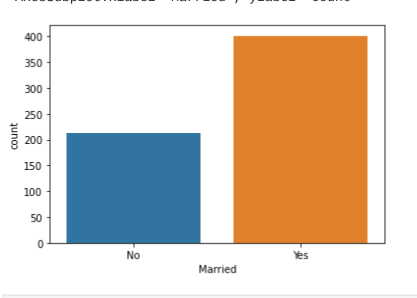
In [9]:
```
# categorical attributes visualization
sns.countplot(df['Gender'])
```

Out[9]: <AxesSubplot:xlabel='Gender', ylabel='count'>



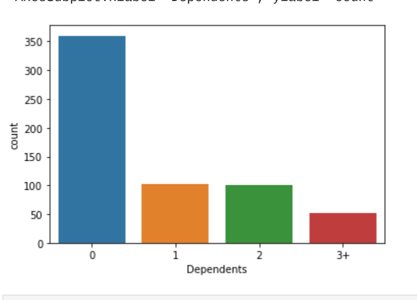
In [10]:
```
sns.countplot(df['Married'])
```

Out[10]: <AxesSubplot:xlabel='Married', ylabel='count'>



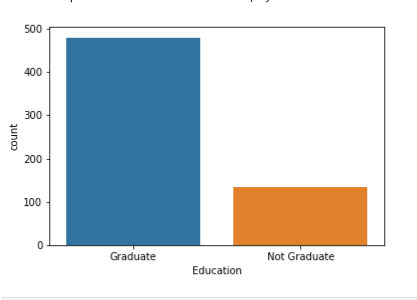
In [11]:
```
sns.countplot(df['Dependents'])
```

Out[11]: <AxesSubplot:xlabel='Dependents', ylabel='count'>



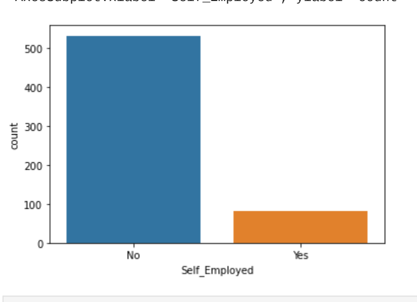
In [12]:
```
sns.countplot(df['Education'])
```

Out[12]: <AxesSubplot:xlabel='Education', ylabel='count'>



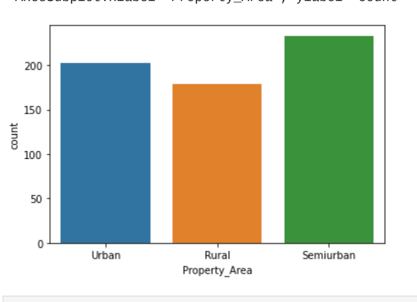
In [13]:
```
sns.countplot(df['Self_Employed'])
```

Out[13]: <AxesSubplot:xlabel='Self_Employed', ylabel='count'>



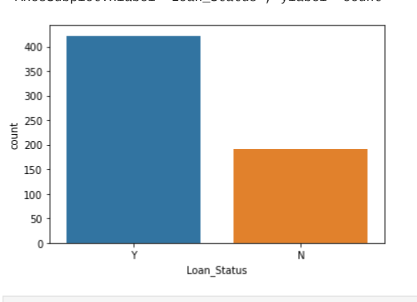
In [14]:
```
sns.countplot(df['Property_Area'])
```

Out[14]: <AxesSubplot:xlabel='Property_Area', ylabel='count'>



In [15]:
```
sns.countplot(df['Loan_Status'])
```

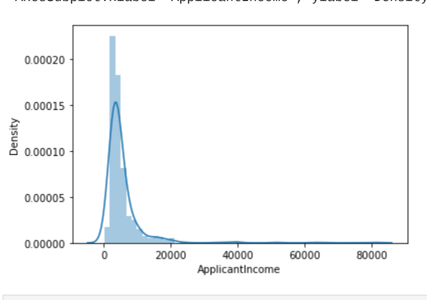
Out[15]: <AxesSubplot:xlabel='Loan_Status', ylabel='count'>



In [ ]:

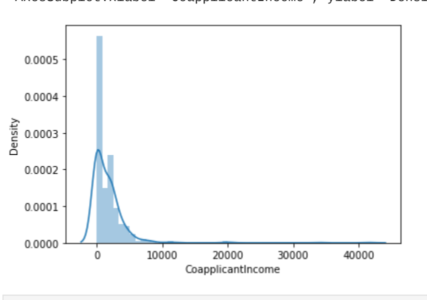
In [16]:
```
# numerical attributes visualization
sns.distplot(df["ApplicantIncome"])
```

Out[16]: <AxesSubplot:xlabel='ApplicantIncome', ylabel='Density'>



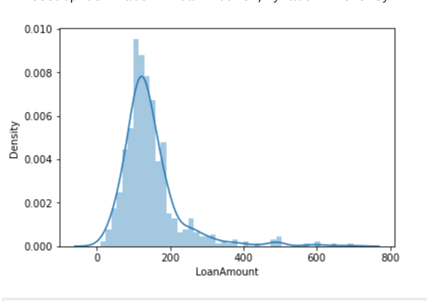
In [17]:
```
sns.distplot(df["CoapplicantIncome"])
```

Out[17]: <AxesSubplot:xlabel='CoapplicantIncome', ylabel='Density'>



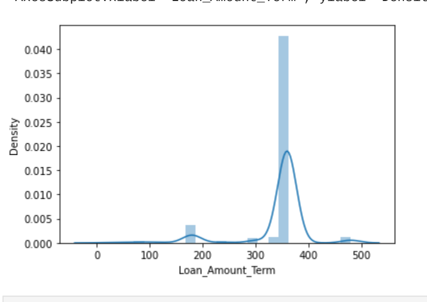
In [18]:
```
sns.distplot(df["LoanAmount"])
```

Out[18]: <AxesSubplot:xlabel='LoanAmount', ylabel='Density'>



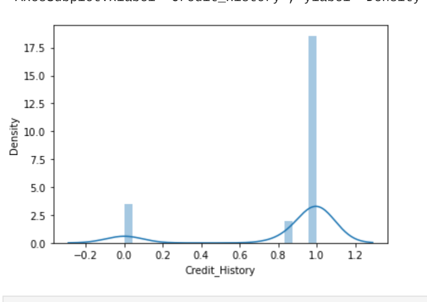
In [19]:
```
sns.distplot(df['Loan_Amount_Term'])
```

Out[19]: <AxesSubplot:xlabel='Loan_Amount_Term', ylabel='Density'>



In [20]:
```
sns.distplot(df['Credit_History'])
```

Out[20]: <AxesSubplot:xlabel='Credit_History', ylabel='Density'>



In [ ]:

Creation of new attributes

In [21]:
```
# total income
df['Total_Income'] = df['ApplicantIncome'] + df['CoapplicantIncome']
df.head()
```

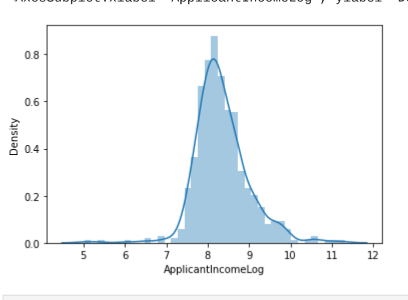
Out[21]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status	Total_Income
0	LP001002	Male	No	0	Graduate	No	5849	0.0	146.412162	360.0	1.0	Urban	Y	5849.0
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.000000	360.0	1.0	Rural	N	6091.0
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.000000	360.0	1.0	Urban	Y	3000.0
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.000000	360.0	1.0	Urban	Y	4941.0
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.000000	360.0	1.0	Urban	Y	6000.0

Log Transformation

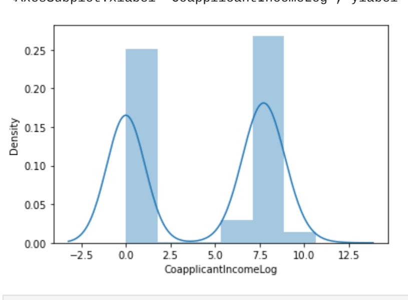
In [22]:
```
# apply log transformation to the attribute
df['ApplicantIncomeLog'] = np.log(df['ApplicantIncome']+1)
sns.distplot(df["ApplicantIncomeLog"])
```

Out[22]: <AxesSubplot:xlabel='ApplicantIncomeLog', ylabel='Density'>



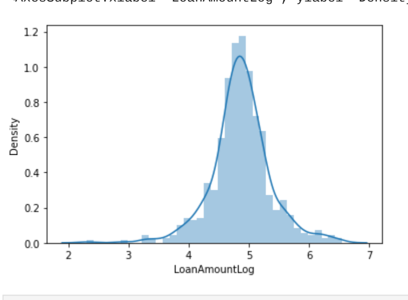
In [23]:
```
df['CoapplicantIncomeLog'] = np.log(df['CoapplicantIncome']+1)
sns.distplot(df["CoapplicantIncomeLog"])
```

Out[23]: <AxesSubplot:xlabel='CoapplicantIncomeLog', ylabel='Density'>



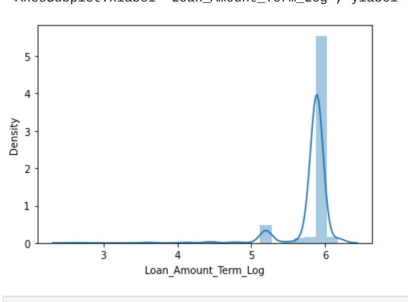
In [24]:
```
df['LoanAmountLog'] = np.log(df['LoanAmount']+1)
sns.distplot(df["LoanAmountLog"])
```

Out[24]: <AxesSubplot:xlabel='LoanAmountLog', ylabel='Density'>



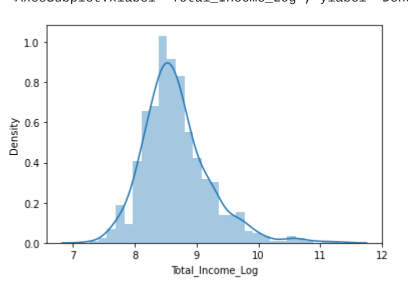
In [25]:
```
df['Loan_Amount_Term_Log'] = np.log(df['Loan_Amount_Term']+1)
sns.distplot(df["Loan_Amount_Term_Log"])
```

Out[25]: <AxesSubplot:xlabel='Loan_Amount_Term_Log', ylabel='Density'>



In [26]:
```
df['Total_Income_Log'] = np.log(df['Total_Income']+1)
sns.distplot(df["Total_Income_Log"])
```

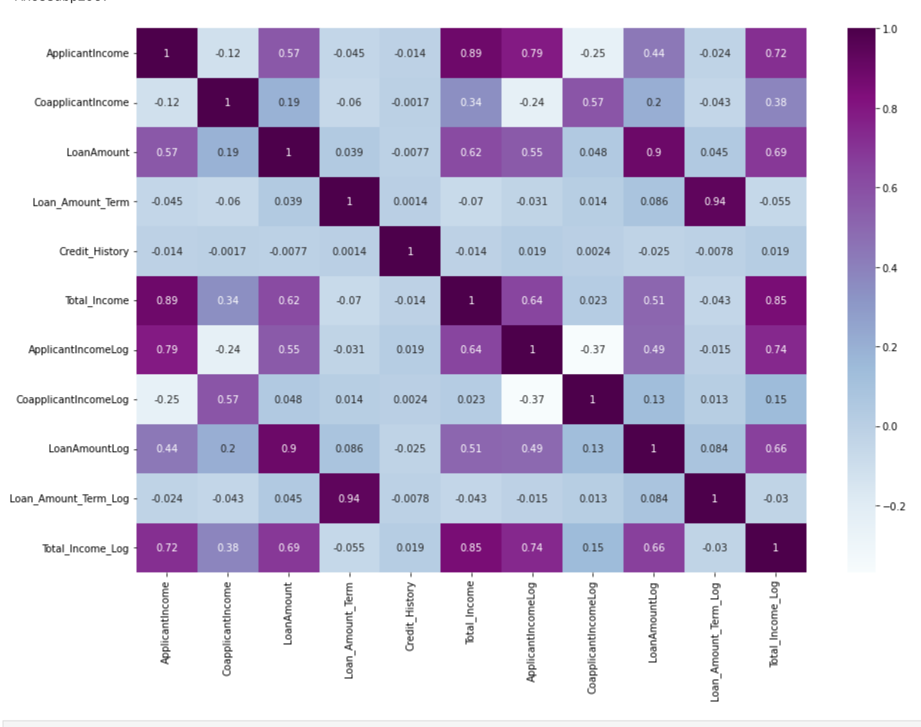
Out[26]: <AxesSubplot:xlabel='Total_Income_Log', ylabel='Density'>



Coorelation Matrix

In [27]:
```
corr = df.corr()
plt.figure(figsize=(15,10))
sns.heatmap(corr, annot = True, cmap="BuPu")
```

Out[27]: <AxesSubplot:>



In [28]:
```
df.head()
```

Out[28]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status	Total_Income	ApplicantIncomeLog	Coap
0	LP001002	Male	No	0	Graduate	No	5849	0.0	146.412162	360.0	1.0	Urban	Y	5849.0	8.674197	
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.000000	360.0	1.0	Rural	N	6091.0	8.430327	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.000000	360.0	1.0	Urban	Y	3000.0	8.006701	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.000000	360.0	1.0	Urban	Y	4941.0	7.857094	
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.000000	360.0	1.0	Urban	Y	6000.0	8.699681	

In [29]:
```
# drop unnecessary columns
cols = ['ApplicantIncome', 'CoapplicantIncome', "LoanAmount", "Loan_Amount_Term", "Total_Income", 'Loan_ID', 'CoapplicantIncomeLog']
df = df.drop(columns=cols, axis=1)
df.head()
```

Out[29]:

	Gender	Married	Dependents	Education	Self_Employed	Credit_History	Property_Area	Loan_Status	ApplicantIncomeLog	LoanAmountLog	Loan_Amount_Term_Log	Total_Income_Log
0	Male	No	0	Graduate	No	1.0	Urban	Y	8.674197	4.993232	5.888878	8.674197
1	Male	Yes	1	Graduate	No	1.0	Rural	N	8.430327	4.859812	5.888878	8.714732
2	Male	Yes	0	Graduate	Yes	1.0	Urban	Y	8.006701	4.204693	5.888878	8.006701
3	Male	Yes	0	Not Graduate	No	1.0	Urban	Y	7.857094	4.795791	5.888878	8.505525
4	Male	No	0	Graduate	No	1.0	Urban	Y	8.699681	4.955827	5.888878	8.699681

Label Encoding

In [30]:
```
from sklearn.preprocessing import LabelEncoder
cols = ['Gender',"Married","Education",'Self_Employed',"Property_Area","Loan_Status","Dependents"]
le = LabelEncoder()
for col in cols:
    df[col] = le.fit_transform(df[col])
```

In [31]:
```
df.head()
```

Out[31]:

	Gender	Married	Dependents	Education	Self_Employed	Credit_History	Property_Area	Loan_Status	ApplicantIncomeLog	LoanAmountLog	Loan_Amount_Term_Log	Total_Income_Log
0	1	0	0	0	0	1.0	2	1	8.674197	4.993232	5.888878	8.674197
1	1	1	1	0	0	1.0	0	0	8.430327	4.859812	5.888878	8.714732
2	1	1	0	0	1	1.0	2	1	8.006701	4.204693	5.888878	8.006701
3	1	1	0	1	0	1.0	2	1	7.857094	4.795791	5.888878	8.505525
4	1	0	0	0	0	1.0	2	1	8.699681	4.955827	5.888878	8.699681

Train-Test Split

In [32]:
```
# specify input and output attributes
X = df.drop(columns=['Loan_Status'], axis=1)
y = df['Loan_Status']
```

In [33]:
```
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)
```

Model Training

In [34]:
```
# classify function
from sklearn.model_selection import cross_val_score
def classify(model, x, y):
    x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
    model.fit(x_train, y_train)
    print("Accuracy is", model.score(x_test, y_test)*100)
    # cross validation - it is used for better validation of model
    # eg: cv-5, train-4, test-1
    score = cross_val_score(model, x, y, cv=5)
    print("Cross validation is",np.mean(score)*100)
```

In [35]:
```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
classify(model, X, y)
```

```
Accuracy is 77.27272727272727
Cross validation is 80.9462881514061
```

In [36]:
```
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier()
classify(model, X, y)
```

```
Accuracy is 72.72727272727273
Cross validation is 70.03731840597094
```

In [37]:
```
from sklearn.ensemble import RandomForestClassifier,ExtraTreesClassifier
model = RandomForestClassifier()
classify(model, X, y)
```

```
Accuracy is 77.92207792207793
Cross validation is 78.50593096094897
```

In [38]:
```
model = ExtraTreesClassifier()
classify(model, X, y)
```

```
Accuracy is 75.32467532467533
Cross validation is 77.20245235239238
```

Hyperparameter tuning

In [39]:
```
model = RandomForestClassifier(n_estimators=100, min_samples_split=25, max_depth=7, max_features=1)
classify(model, X, y)
```

```
Accuracy is 77.27272727272727
Cross validation is 80.62108489937359
```

Confusion Matrix

A confusion matrix is a summary of prediction results on a classification problem. The number of correct and incorrect predictions are summarized with count values and broken down by each class. It gives us insight not only into the errors being made by a classifier but more importantly the types of errors that are being made.

In [40]:
```
model = RandomForestClassifier()
model.fit(x_train, y_train)
```

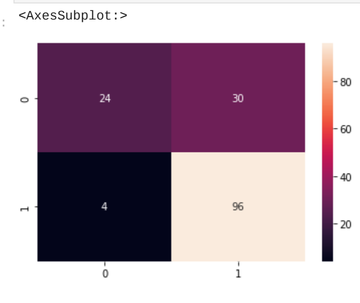
Out[40]: RandomForestClassifier()

In [41]:
```
from sklearn.metrics import confusion_matrix
y_pred = model.predict(x_test)
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[41]:
```
array([[24, 30],
       [ 4, 96]], dtype=int64)
```

In [42]:
```
sns.heatmap(cm, annot=True)
```

Out[42]: <AxesSubplot:>



The End :)

In [ ]: