Name: Shubham Jagannath Bhosle

Project Title: Loan Approval Status Prediction

Course Name: Master's in Data Science & Data Analytics with AI

Batch Code: T318

Mentor's Name: Prachiti Sonawane

LOAN APPROVAL PREDICTION PROJECT USING MACHINE LEARNING

Introduction

Problem Statement: Dream Housing Finance company deals in all home loans. They have a presence across all urban, semi-urban, and rural areas. Customer-first applies for a home loan after that company validates the customer eligibility for a loan.

The company wants to automate the loan eligibility process (real-time) based on customer detail provided while filling the 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 customer's segments, those are eligible for loan amount so that they can specifically target these customers.

Project includes the following steps:

- Data cleaning
- Data Preprocessing
- Exploratory data analysis (EDA)
- Preparing the data to train a model
- Training and making predictions using various classification models
- Model evaluation

Dataset Key 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 a loan in months
- **Credit_History:** credit history meets guidelines
- **Property_Area**: Urban/ Semi-Urban/ Rural
- **Loan_Status**: Loan approved (Y/N)

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 981 entries, 0 to 613
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Loan_ID            981 non-null    object
 1   Gender             957 non-null    object
 2   Married            978 non-null    object
 3   Dependents         956 non-null    object
 4   Education          981 non-null    object
 5   Self_Employed      926 non-null    object
 6   ApplicantIncome    981 non-null    int64
 7   CoapplicantIncome  981 non-null    float64
 8   LoanAmount         954 non-null    float64
 9   Loan_Amount_Term   961 non-null    float64
 10  Credit_History     902 non-null    float64
 11  Property_Area      981 non-null    object
 12  Loan_Status        614 non-null    object
dtypes: float64(4), int64(1), object(8)
memory usage: 107.3+ KB
```

Importing Libraries & Making a DataFrame

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

[2]:
```
df1=pd.read_csv('test.csv')
df2=pd.read_csv('train.csv')
```

[3]:
```
df=df1._append(df2)
```

[4]:
```
df.head()
```

[4]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property
0	LP001015	Male	Yes	0	Graduate	No	5720	0.0	110.0	360.0	1.0	
1	LP001022	Male	Yes	1	Graduate	No	3076	1500.0	126.0	360.0	1.0	
2	LP001031	Male	Yes	2	Graduate	No	5000	1800.0	208.0	360.0	1.0	
3	LP001035	Male	Yes	2	Graduate	No	2340	2546.0	100.0	360.0	NaN	
4	LP001051	Male	No	0	Not Graduate	No	3276	0.0	78.0	360.0	1.0	

Data cleaning & Preprocessing

Finding Null values & Handling them

```
[8]:  Loan_ID               0
      Gender               24
      Married               3
      Dependents           25
      Education             0
      Self_Employed        55
      ApplicantIncome       0
      CoapplicantIncome     0
      LoanAmount           27
      Loan_Amount_Term     20
      Credit_History       79
      Property_Area         0
      Loan_Status         367
      dtype: int64
```

```
[9]:  df.isnull().sum()/len(df)*100
```

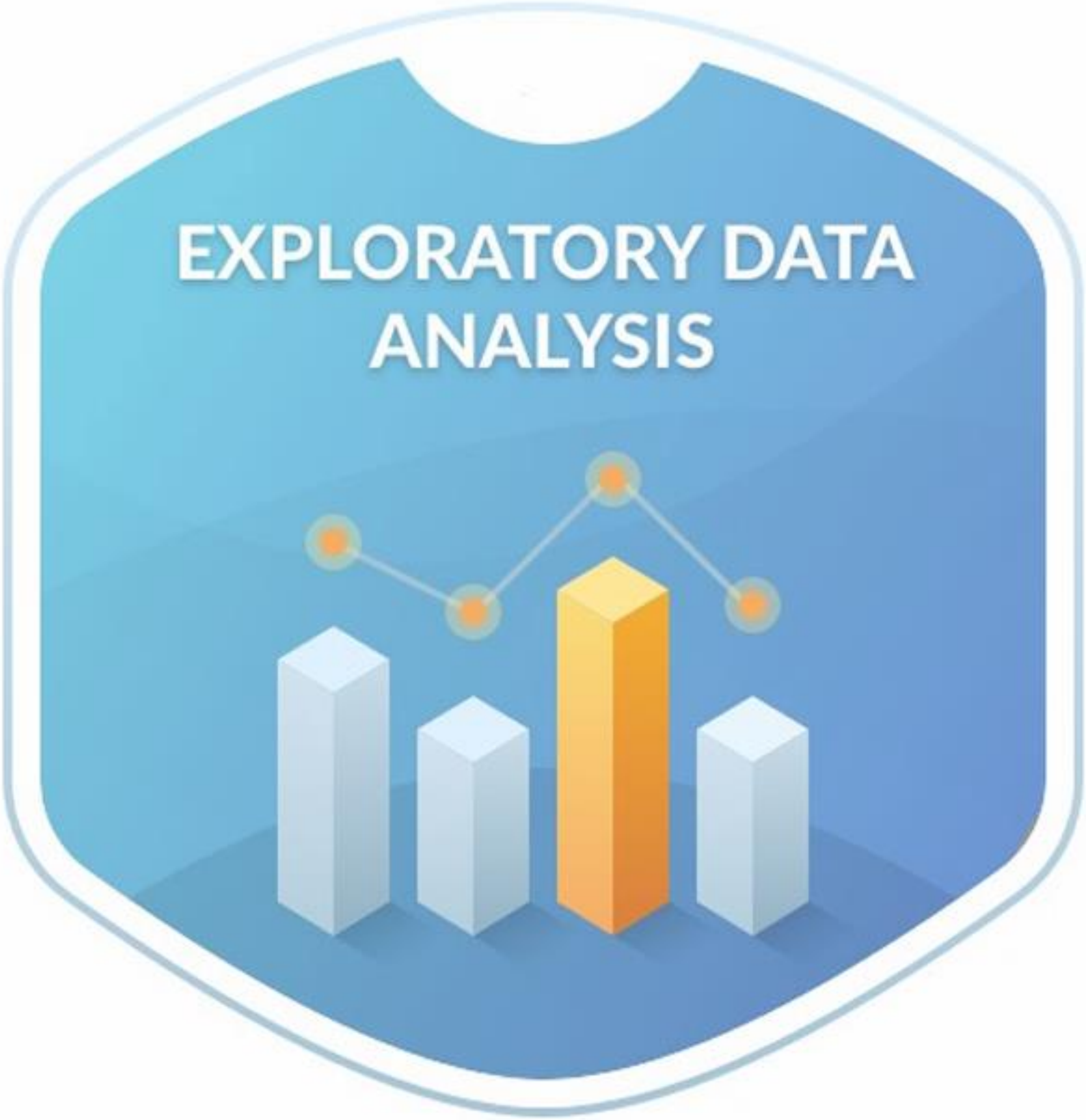
```
[9]:  Loan_ID               0.000000
      Gender                2.446483
      Married               0.305810
      Dependents            2.548420
      Education             0.000000
      Self_Employed         5.606524
      ApplicantIncome       0.000000
      CoapplicantIncome     0.000000
      LoanAmount            2.752294
      Loan_Amount_Term      2.038736
      Credit_History        8.053007
      Property_Area         0.000000
      Loan_Status          37.410805
      dtype: float64
```

```
[10]: #Filling Null values with mode of self_employed column.

[11]: mode_sf_emp=df['Self_Employed'].mode()

[12]: mode_sf_emp

[12]: 0    No
      Name: Self_Employed, dtype: object

[13]: df['Self_Employed']=df['Self_Employed'].fillna(method='ffill')

[14]: #Filling Null values with mode of Loan_status column.

[15]: mode_ln_st=df['Loan_Status'].mode()
      mode_ln_st

[15]: 0    Y
      Name: Loan_Status, dtype: object

[16]: df['Loan_Status']=df['Loan_Status'].fillna(method='ffill')

[17]: #Filling Null values with median of credit_History column.

[18]: median_cr=df['Credit_History'].median()
      median_cr

[18]: 1.0

[19]: df['Credit_History']=df['Credit_History'].fillna(median_cr)
```

[21]:
```
#Dropping the remaining null values from the dataset since their % is low.
df.dropna(inplace=True)
```

[22]:
```
#Final checking for the Null values.
df.isnull().sum()
```

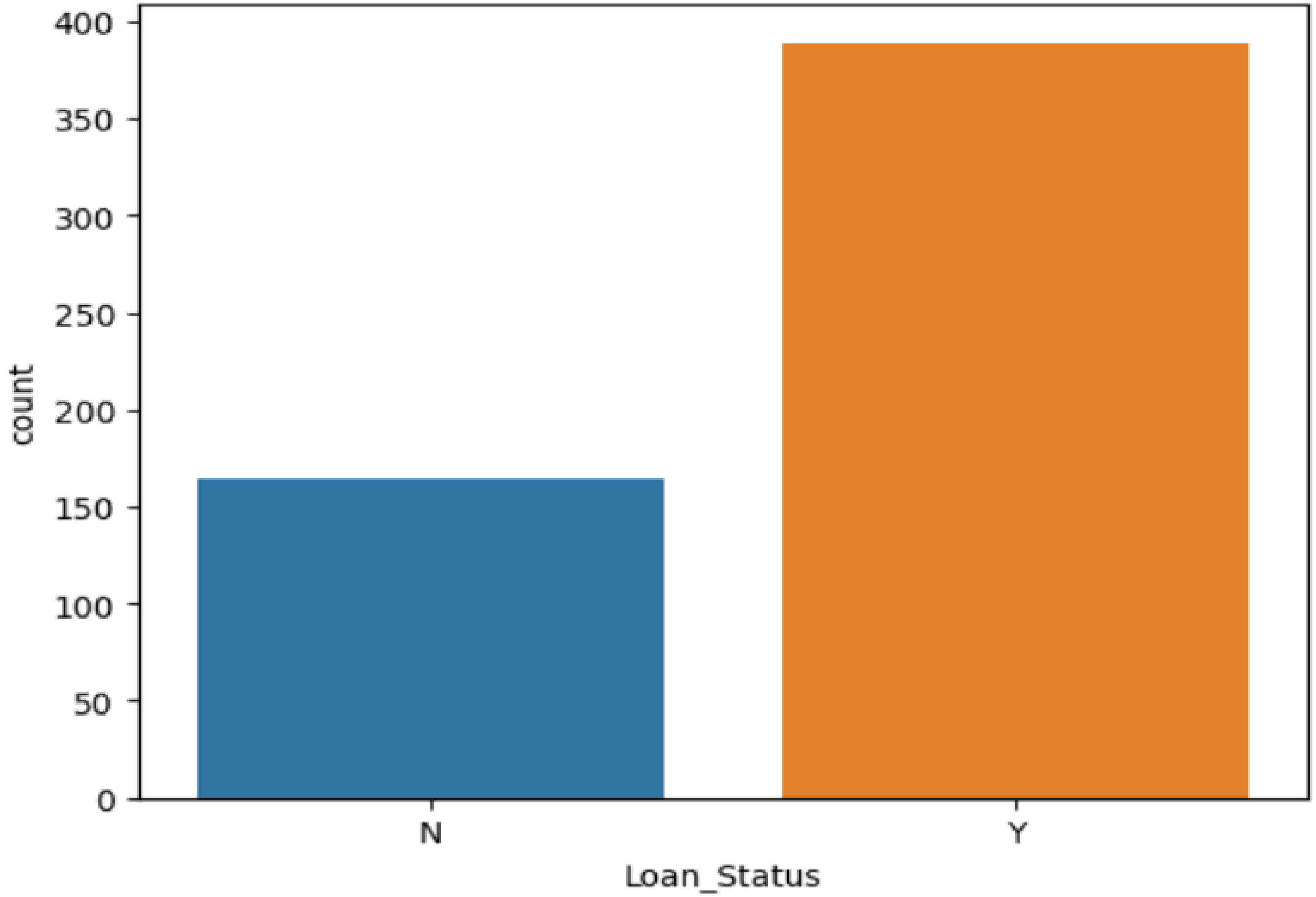
[22]:
```
Loan_ID              0
Gender               0
Married              0
Dependents           0
Education            0
Self_Employed        0
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           0
Loan_Amount_Term     0
Credit_History       0
Property_Area        0
Loan_Status   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


EXPLORATORY DATA
ANALYSIS

Finding Maximum Loan Approval Count

[27]:
```
sns.countplot(data=df,x=df['Loan_Status'])
```

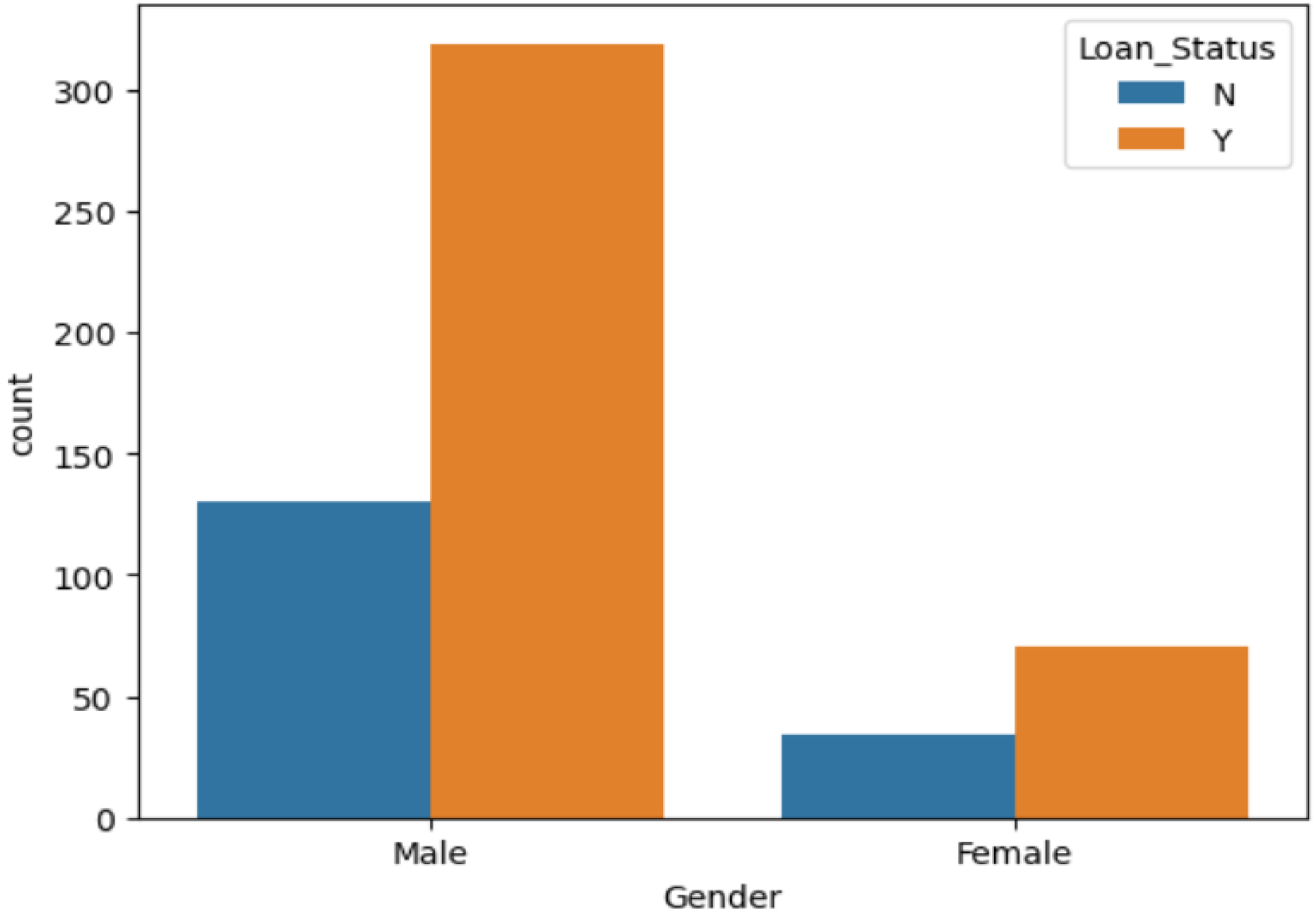
[27]: <Axes: xlabel='Loan_Status', ylabel='count'>



Loan approval status according to Genders

[28]:
```
sns.countplot(data=df,x=df['Gender'],hue=df['Loan_Status'])
```

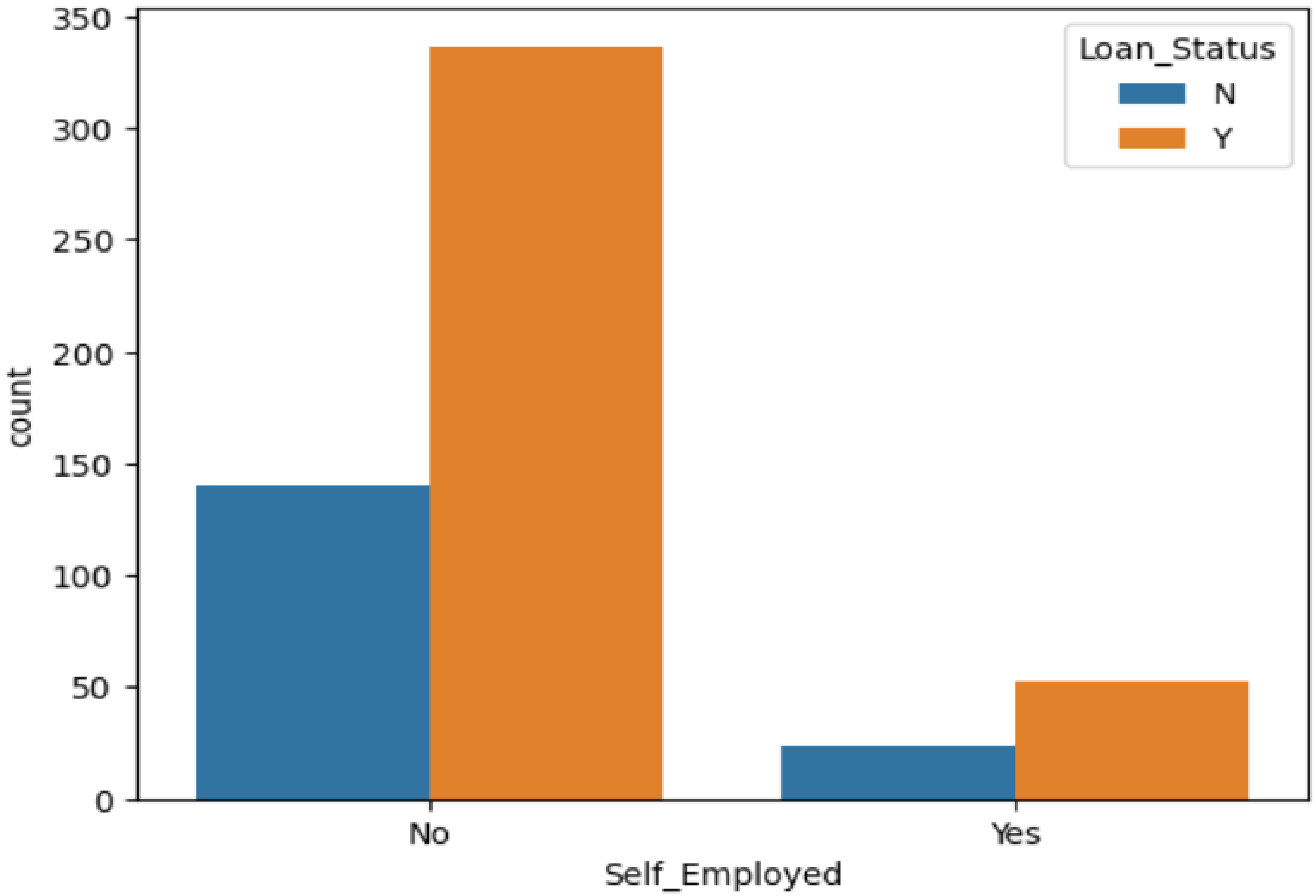
[28]: <Axes: xlabel='Gender', ylabel='count'>



Loan Approval count of Self Employed people

[29]:
```
sns.countplot(data=df,x=df['Self_Employed'],hue=df['Loan_Status'])
```

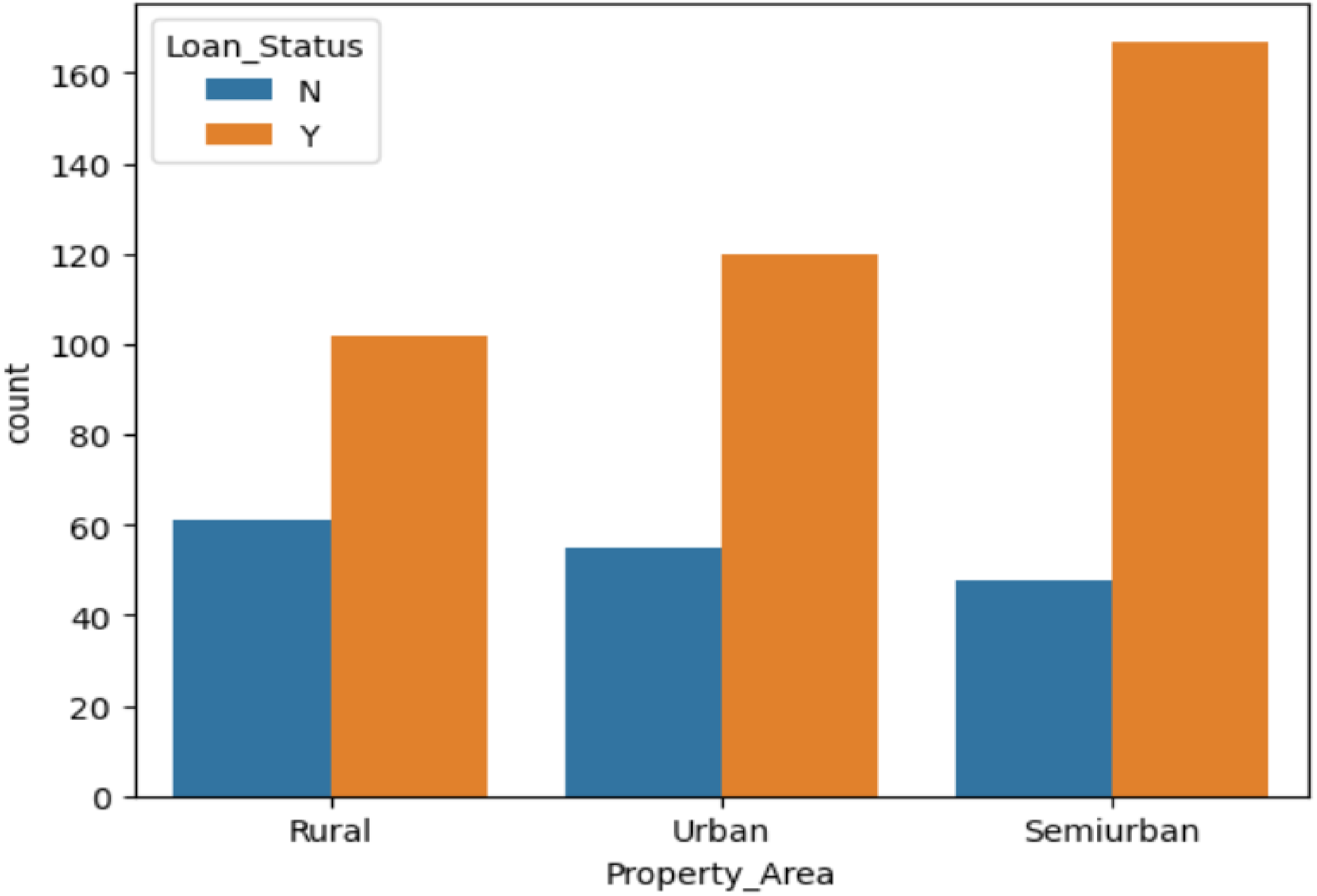
[29]: <Axes: xlabel='Self_Employed', ylabel='count'>



Loan Approval count according to Property Area ¶

[30]:
```
sns.countplot(data=df,x=df['Property_Area'],hue=df['Loan_Status'])
```

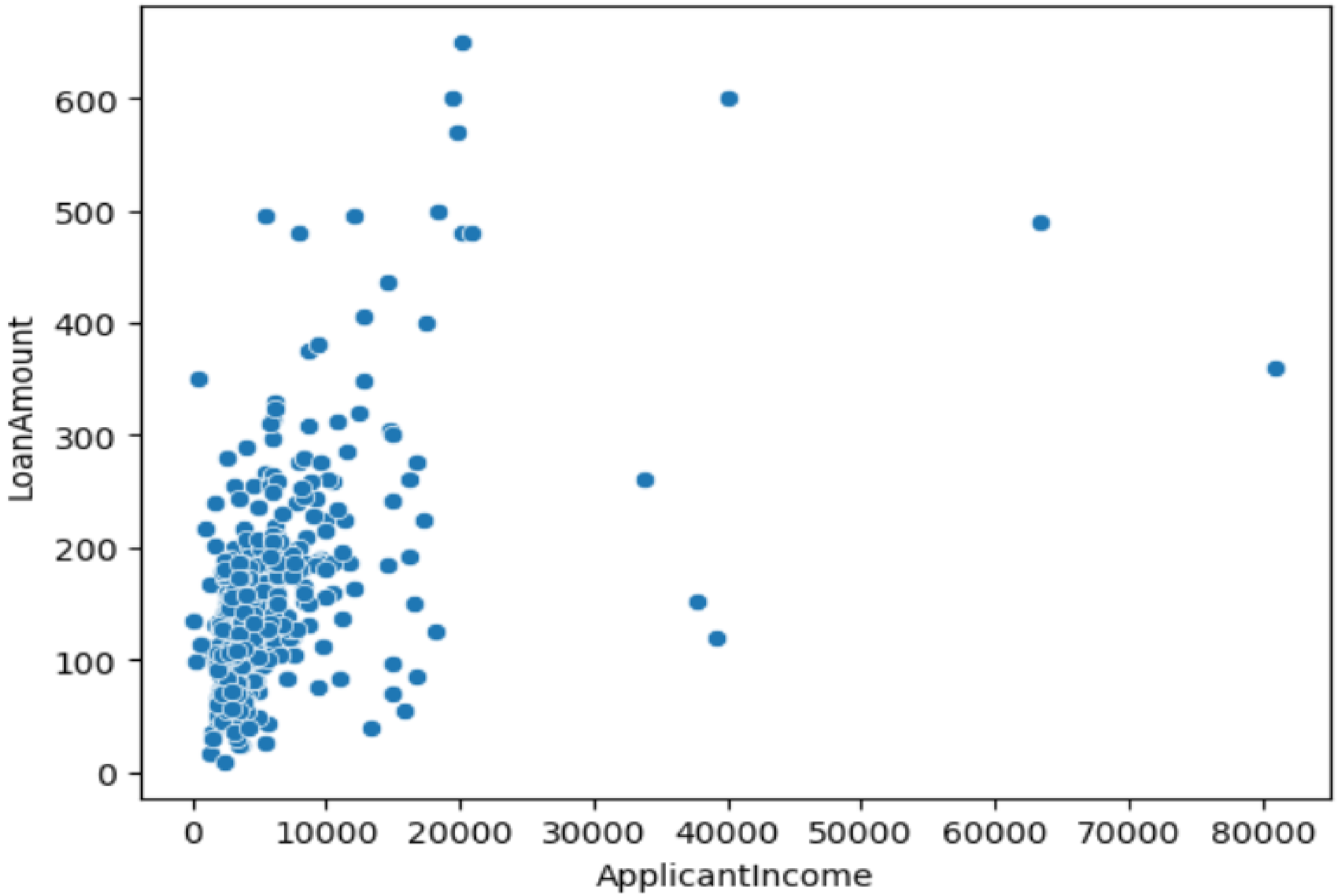
[30]:
```
<Axes: xlabel='Property_Area', ylabel='count'>
```



Relation Between Applicant Income & Loan Amount

```
[31]: sns.scatterplot(data=df,x=df['ApplicantIncome'],y=df['LoanAmount'])
```

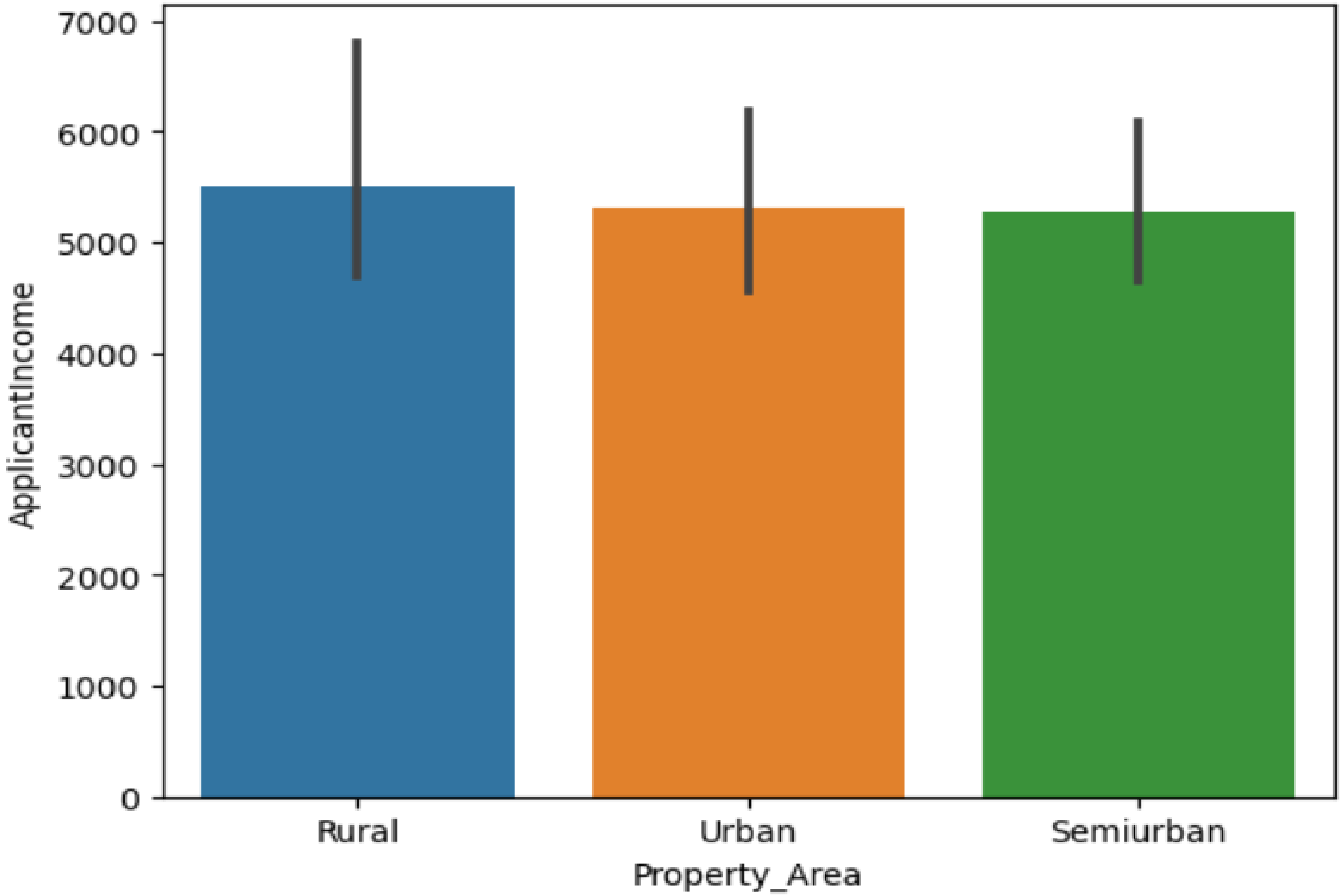
```
[31]: <Axes: xlabel='ApplicantIncome', ylabel='LoanAmount'>
```



Highest Applicant Income according to Property Area

```
[33]: sns.barplot(data=df,x=df['Property_Area'],y=df['ApplicantIncome'])
```

```
[33]: <Axes: xlabel='Property_Area', ylabel='ApplicantIncome'>
```



Encoding usding LabelEncoder

[34]:
```
from sklearn.preprocessing import LabelEncoder
```

[35]:
```
le=LabelEncoder()
```

[36]:
```
cat_col=df.select_dtypes(include='O').columns
cat_col
```

[36]:
```
Index(['Loan_ID', 'Gender', 'Married', 'Education', 'Self_Employed',
       'Property_Area', 'Loan_Status'],
      dtype='object')
```

[37]:
```
for i in cat_col:
    df[i]=le.fit_transform(df[i])
```

[38]:
```
df.head()
```

[38]:

	Loan_ID	Gender	Married	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_St
1	0	1	1	0	0	4583	1508.0	128.0	360.0	1.0	0	
2	1	1	1	0	1	3000	0.0	66.0	360.0	1.0	2	
3	2	1	1	1	0	2583	2358.0	120.0	360.0	1.0	2	
4	3	1	0	0	0	6000	0.0	141.0	360.0	1.0	2	
5	4	1	1	0	1	5417	4196.0	267.0	360.0	1.0	2	

Separating the Features (X) & Target (Y) from the dataset.

```
[39]: x=df.drop('Loan_Status',axis=1)
      x
```

[39]:

	Loan_ID	Gender	Married	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area
1	0	1	1	0	0	4583	1508.0	128.0	360.0	1.0	0
2	1	1	1	0	1	3000	0.0	66.0	360.0	1.0	2
3	2	1	1	1	0	2583	2358.0	120.0	360.0	1.0	2
4	3	1	0	0	0	6000	0.0	141.0	360.0	1.0	2
5	4	1	1	0	1	5417	4196.0	267.0	360.0	1.0	2
...	...	...	...	...	...	...	...	...	...	...	...
609	548	0	0	0	0	2900	0.0	71.0	360.0	1.0	0
610	549	1	1	0	0	4106	0.0	40.0	180.0	1.0	0
611	550	1	1	0	0	8072	240.0	253.0	360.0	1.0	2
612	551	1	1	0	0	7583	0.0	187.0	360.0	1.0	2
613	552	0	0	0	1	4583	0.0	133.0	360.0	0.0	1

553 rows × 11 columns

```
[40]: y=df['Loan_Status']
      y
```

```
[40]: 1      0
      2      1
      3      1
      4      1
      5      1
            ..
      609    1
      610    1
      611    1
      612    1
```

Balancing the Dataset

[41]:
```
y.value_counts()
```

[41]:
```
Loan_Status
1    389
0    164
Name: count, dtype: int64
```

[42]:
```
from imblearn import under_sampling, over_sampling
from imblearn.over_sampling import SMOTE

x_resampled, y_resampled=SMOTE().fit_resample(x,y)
x, y=SMOTE().fit_resample(x,y)
```

[43]:
```
y.value_counts().to_frame()
```

[43]:

	count
Loan_Status	
0	389
1	389

Splitting Data into Training & Testing data.

[44]:
```
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.20,random_state=1)
```

[45]:
```
comp=dict()
```

Predicting Using KNN Classifier

[46]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 5)
knn.fit(xtrain,ytrain)
ypred = knn.predict(xtest)
```

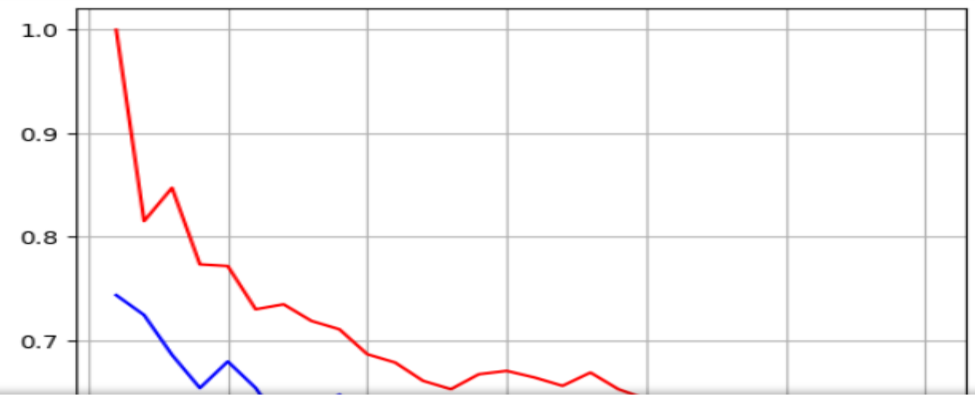
[47]:
```
from sklearn.metrics import accuracy_score,classification_report
ac = accuracy_score(ytest,ypred)
print(ac)
print(classification_report(ytest,ypred))
```

```
0.6794871794871795
              precision    recall  f1-score   support

           0       0.69      0.74      0.71        84
           1       0.67      0.61      0.64        72

    accuracy                           0.68       156
   macro avg       0.68      0.67      0.68       156
weighted avg       0.68      0.68      0.68       156
```

HPT

[48]:
```
trainac=[]
testac=[]

for i in range(1,31):
    kn=KNeighborsClassifier(n_neighbors=i)
    kn.fit(xtrain,ytrain)

    train=kn.score(xtrain,ytrain)
    test=kn.score(xtest,ytest)

    trainac.append(train)
    testac.append(test)
```

[49]:
```
plt.plot(range(1,31),trainac,color="red")
plt.plot(range(1,31),testac,color="blue")
plt.grid()
```



[50]:
```
#Re-building the model using n_neighbors=2

from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 2)
knn.fit(xtrain,ytrain)
ypred = knn.predict(xtest)
```

[51]:
```
from sklearn.metrics import accuracy_score,classification_report
ac = accuracy_score(ytest,ypred)
print(ac)
print(classification_report(ytest,ypred))
```

```
0.7243589743589743
              precision    recall  f1-score   support

           0       0.68      0.92      0.78        84
           1       0.84      0.50      0.63        72

    accuracy                           0.72       156
   macro avg       0.76      0.71      0.70       156
weighted avg       0.75      0.72      0.71       156
```

Predicting using LogisticRegression

```
[53]: from sklearn.linear_model import LogisticRegression
      from sklearn.metrics import classification_report,accuracy_score
```

```
[54]: def my_model(model):
          model.fit(xtrain,ytrain)
          ypred=model.predict(xtest)
          print(classification_report(ytest,ypred))
```

```
[55]: lr=LogisticRegression()
```

```
[56]: my_model(lr)
```

```
              precision    recall  f1-score   support

           0       0.68      0.64      0.66        84
           1       0.61      0.65      0.63        72

    accuracy                           0.65       156
   macro avg       0.65      0.65      0.65       156
weighted avg       0.65      0.65      0.65       156
```

- Using Logistic Regression we have achieved an Average Accuracy of 65 % which is not that good.
- Lets see if we can increase this accuracy by hyper tuning.

HPT

[57]:
```python
#Hypertuning using Solver--> Liblinear

logreg = LogisticRegression(solver = "liblinear")
logreg.fit(xtrain,ytrain)
ypred = logreg.predict(xtest)
```

[58]:
```python
ac = accuracy_score(ytest,ypred)
cr = classification_report(ytest,ypred)
print("Accuracy score : ",ac)
print(cr)
```

```
Accuracy score :  0.7307692307692307
              precision    recall  f1-score   support

           0       0.80      0.67      0.73        84
           1       0.67      0.81      0.73        72

    accuracy                           0.73       156
   macro avg       0.74      0.74      0.73       156
weighted avg       0.74      0.73      0.73       156
```

[59]:
```python
#Hypertuning using Solver--> newton-cg

logreg = LogisticRegression(solver ='newton-cg')
logreg.fit(xtrain,ytrain)
ypred = logreg.predict(xtest)
```

[60]:
```python
ac = accuracy_score(ytest,ypred)
cr = classification_report(ytest,ypred)
print("Accuracy score : ",ac)
print(cr)
```

```
Accuracy score :  0.7115384615384616
              precision    recall  f1-score   support

           0       0.77      0.65      0.71        84
           1       0.66      0.78      0.71        72

    accuracy                           0.71       156
   macro avg       0.72      0.72      0.71       156
weighted avg       0.72      0.71      0.71       156
```

[61]:
```python
logreg=logreg.score(xtest,ytest)
comp['LogisticRegression']=logreg
```

- Both Hyper Tunners 'liblinear' & 'newton-cg' are giving me an accuracy of 71%

Predicting using DecisionTreeClassifier

```
[62]: from sklearn.tree import DecisionTreeClassifier
```

```
[63]: dt=DecisionTreeClassifier()
```

```
[64]: my_model(dt)
```

	precision	recall	f1-score	support
0	0.73	0.77	0.75	84
1	0.72	0.67	0.69	72
accuracy			0.72	156
macro avg	0.72	0.72	0.72	156
weighted avg	0.72	0.72	0.72	156

```
[65]: dt=dt.score(xtest,ytest)
      comp['DT']=dt
```

- Decision Tree Classifier is giving an accuracy of 72%.

RandomForestClassifier

```
[66]: from sklearn.ensemble import RandomForestClassifier
```

```
[67]: rf=RandomForestClassifier()
```

```
[68]: my_model(rf)
```

	precision	recall	f1-score	support
0	0.89	0.74	0.81	84
1	0.74	0.89	0.81	72
accuracy			0.81	156
macro avg	0.81	0.81	0.81	156
weighted avg	0.82	0.81	0.81	156

```
[69]: rf=rf.score(xtest,ytest)
      comp['RandomForest']=rf
```

- Random Forest Classifier is giving an accuracy of 81%

AdaBoostClassifier

```
[70]: from sklearn.ensemble import AdaBoostClassifier,GradientBoostingClassifier
```

```
[71]: adb=AdaBoostClassifier(n_estimators=450)
      my_model(adb)
```

```
              precision    recall  f1-score   support

           0       0.79      0.73      0.76        84
           1       0.71      0.78      0.74        72

    accuracy                           0.75       156
   macro avg       0.75      0.75      0.75       156
weighted avg       0.75      0.75      0.75       156
```

- ADABoost Classifier is giving an accuracy of 75%

```
[72]: adb=adb.score(xtest,ytest)
      comp['ADABoost']=adb
```

Gradient Boosting Classifier

[73]:
```
gb=GradientBoostingClassifier()
my_model(gb)
```

```
              precision    recall  f1-score   support

           0       0.88      0.76      0.82        84
           1       0.76      0.88      0.81        72

    accuracy                           0.81       156
   macro avg       0.82      0.82      0.81       156
weighted avg       0.82      0.81      0.81       156
```

- Gradient Boost Classifier is giving an accuracy of 81%

[74]:
```
gb=gb.score(xtest,ytest)
comp['GradientBoost']=gb
```

XGBClassifier

```
[75]: from xgboost import XGBClassifier
```

```
[76]: xgb=XGBClassifier()
      my_model(xgb)
```

```
              precision    recall  f1-score   support

           0       0.82      0.82      0.82        84
           1       0.79      0.79      0.79        72

    accuracy                           0.81       156
   macro avg       0.81      0.81      0.81       156
weighted avg       0.81      0.81      0.81       156
```

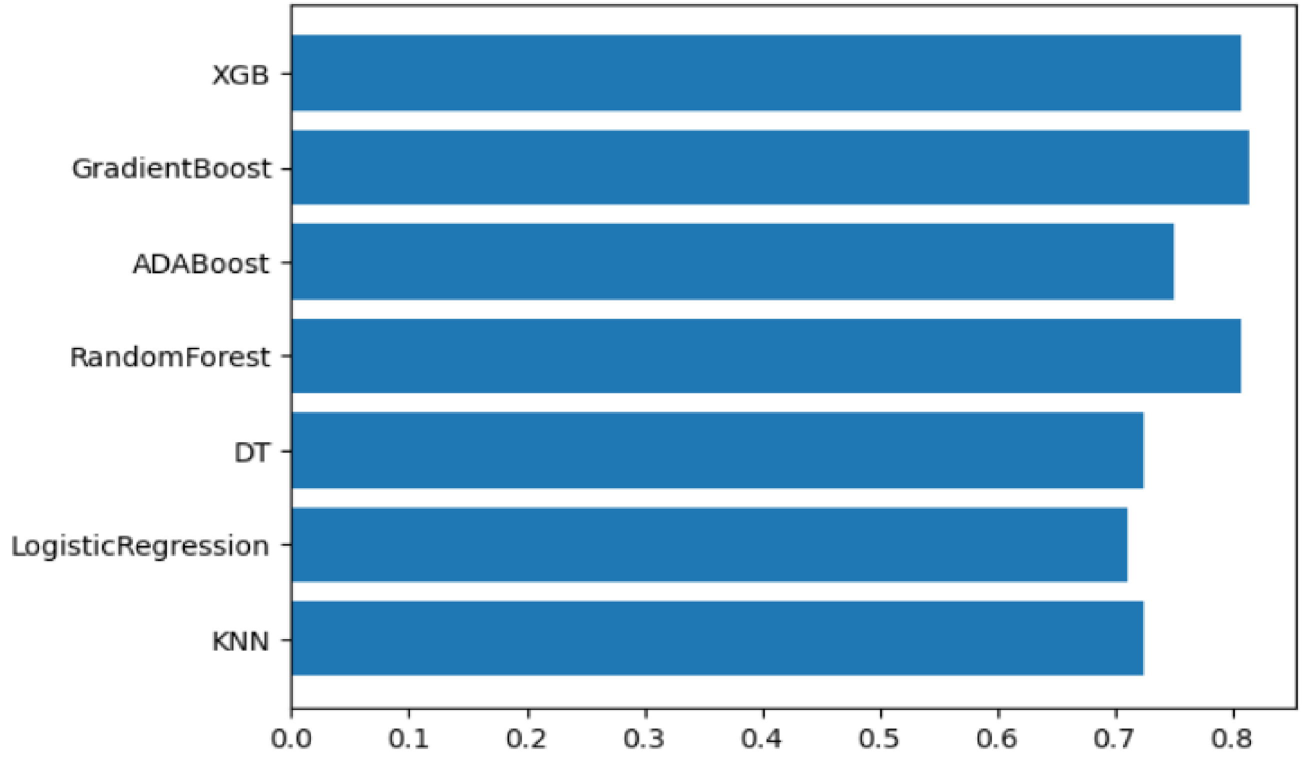
- XGBoost Classifier is giving an accuracy of 81%

```
[77]: xgb=xgb.score(xtest,ytest)
      comp['XGB']=xgb
```

Comparing Predictions of all the Classifiers

[78]:
```
keys = list(comp.keys())
val = list(comp.values())
plt.barh(keys,val)
```

[78]: <BarContainer object of 7 artists>



After comparison between the classifiers , we can see that Gradient Boosting classifier is having the highest accuracy, and which is best for our model.

Thank You...