Loan Approval Prediction Using Machine Learning

Reg. ID:2110870
Student Name: Satheesh Kumar MC

Date: July, 2022

Abstract

Supervised machine learning is the subset of machine learning and Artificial Intelligence, in which the classifiers are trained using labelled datasets to make accurate predictions of the outcomes. This research aims to identify the best classifier while also describing various supervised learning classifiers. In this study, four algorithms were used Random Forest, decision tree, SVM, Gradient boost and XGBoost and the loan approval dataset, which had 13 features and 614 cases, was used to put the algorithms into practise. In comparison to other classifiers, the study demonstrates that the random forest is the best prominent classifier with the best precision and accuracy.

Keywords: Keywords: Data, analysis, classifiers, learning algorithms, decision tree, SVM, Random Forest

Contents

1 Introduction

: Data mining is the most important application of machine learning (ML), which has a wide range of uses. Humans undoubtedly make mistakes while conducting analyses or attempting to glean valuable insights from large amounts of data. As a result, it is challenging to identify solutions to problems [1]. This is where machine learning (ML) enters the picture; it may be successfully used to solve those challenges and increase the effectiveness of the problems [1]. Machine learning (ML) can be thought of as a subcategory of AI because those algorithms can be seen as the medium to teach machine to behave more intelligently. ML is influenced by a variety of academic disciplines, including computer science, neurobiology, statistics etc. A machine learning classifier accomplishes this goal by teaching computers how to automatically identify a trustworthy prediction based on prior knowledge, which is the core objective of attempts at machine learning. Using a model to predict unknown values (output variables) based on a variety of already known values is the act of classification (input variables) [2]. ML classifiers represent each instances of a data set using the same collection of features. And these instances could be categorical, continuous. If examples are given known labels (i.e., the accurate outputs that match), the learning technique is referred to be supervised; in contrast, in an unsupervised learning approach, the instances are left unlabelled [1].

1.1 Aims and Objectives

The main of the project is to predict the retail store sales using machine learning and deep learning techniques.

1.2 Project Proposal

- Data Exploratory Analysis
- Data Pre-processing.
- Feature Engineering and selection
- Model Training and optimization.
- Metrics and final discussion.

2 Data

This data set was gathered through a Kaggle competition that was designed to forecast whether or not a customer would be approved for a loan. It has 13 features, with 614 instances. Loan ID, gender, married status, dependents, education level, self-employment, applicant and co-applicant income, loan amount, loan term, credit history, property area, and loan status are among the features in the data set. The class label, which is either accepted or not, is the last feature. There are 614 instances, 13 columns altogether, 8 object types, 1 int, and 4 float kinds. This data set only had a small number of missing instances, which were therefore easily imputed utilising the mean or median for int types and the mode for object types.

3 Exploratory Data Analysis

Gender bar plot
Male applications for loans are more numerous than female applicants.

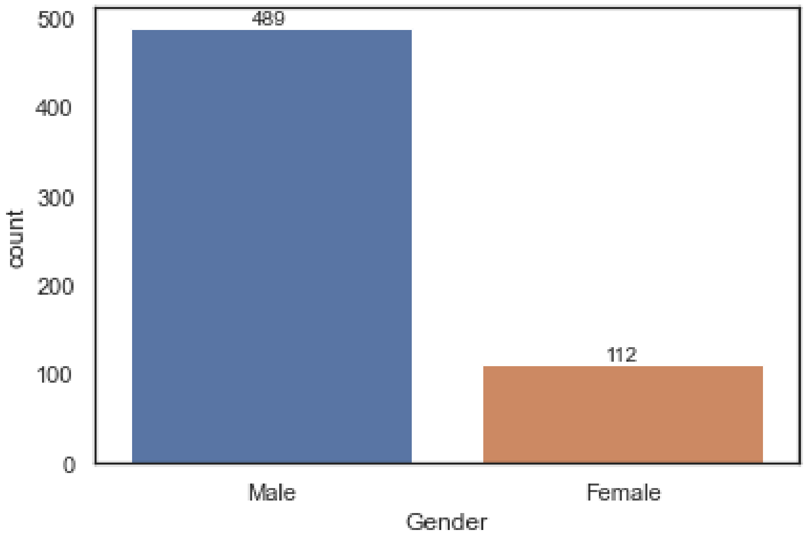


Figure 1: Gender Proportion

Frequencies of Property area
The majority of loan applications come from semi-urban areas, then urban, and then rural areas.

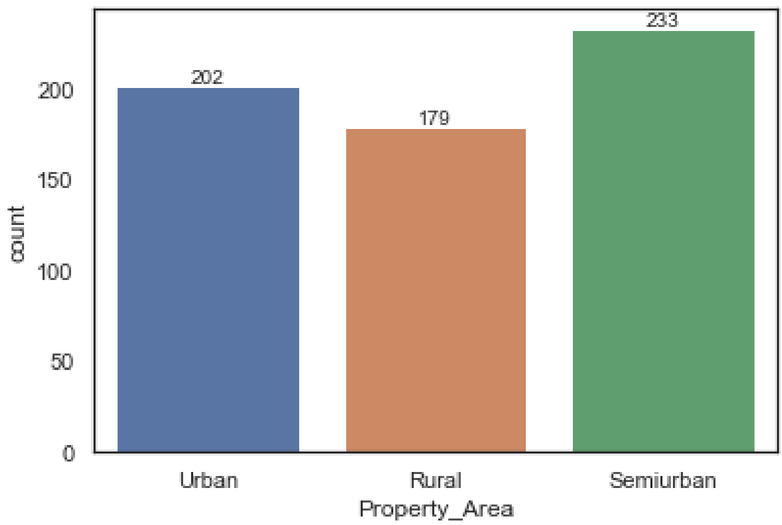


Figure 2: Property area frequencies

Self Employment Status
The majority of loan applicants are not self employed, which implies they are working with some organizations.

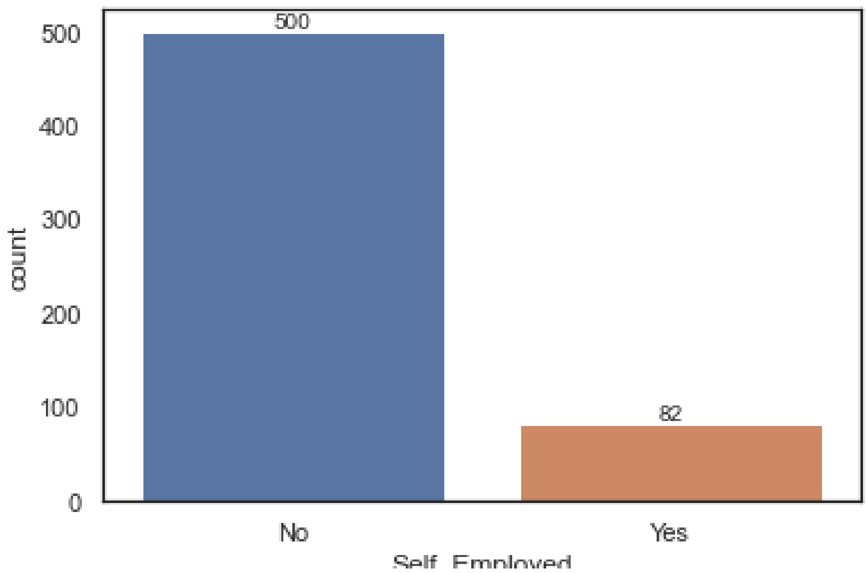


Figure 3: Self Employment status

Loan amount vs applicant income

A regression plot, Loan amount and applicant income have a positive relationship; the larger the income, the better the likelihood that the loan would be approved.

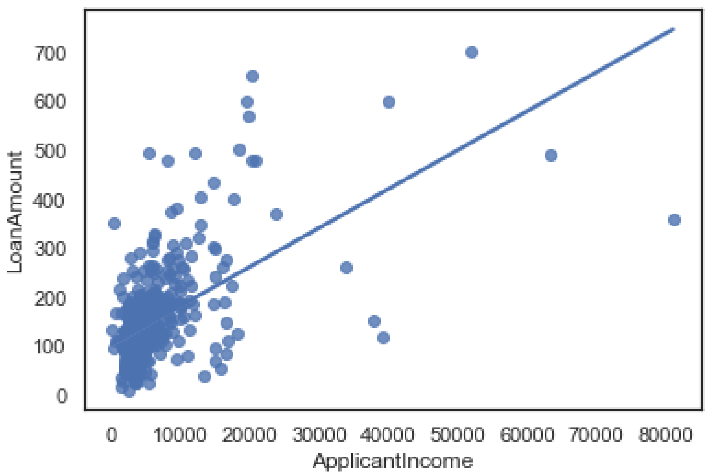


Figure 4: Regression plot for applicant income and loan amount

Applicant Income distribution

Majority of the applicants incomes lies in between 2500 and 6000 and the average income is 4000

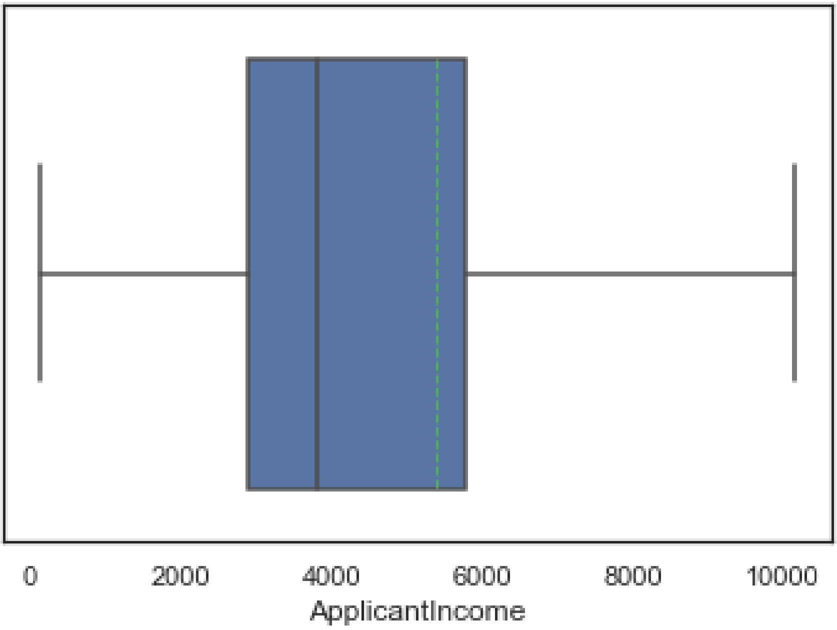


Figure 5: Income Distribution

Loan Approval Status vs Martial status
Loan approval rates are higher for applicants who are married and lower for candidates who are not married.

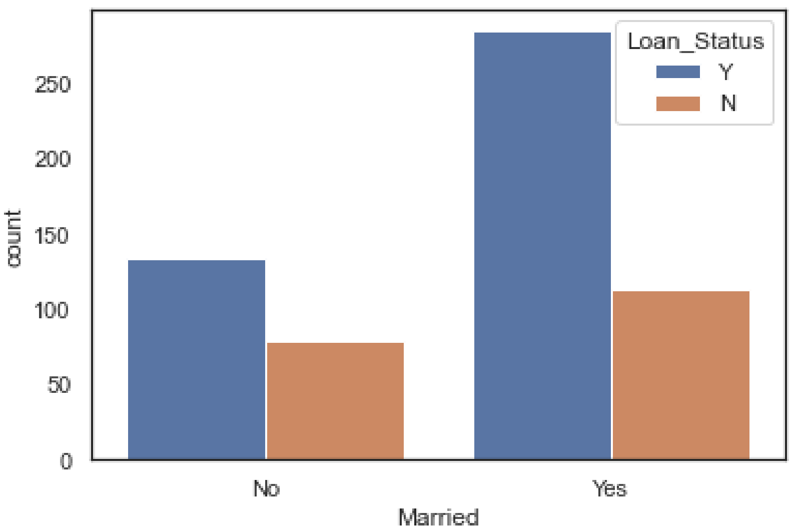


Figure 6: Loan Approval Status vs Martial status

Loan Approval Status vs Employment
Similarly, graduates have a greater loan acceptance rate than non-graduates.

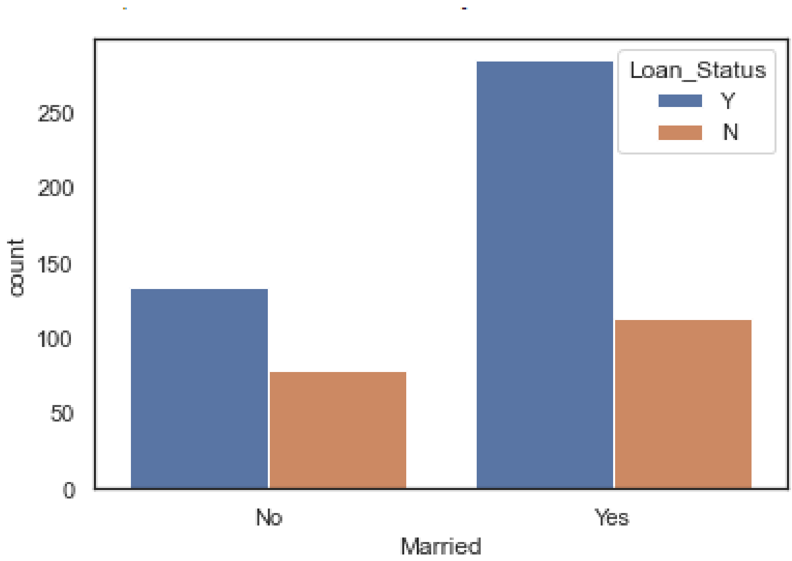


Figure 7: Loan Approval Status vs Employment

4 Methodology

The data set was gathered via the open-source Kaggle competition, as was previously indicated. *https://www.kaggle.com/datasets/altruistdelhite04/loan-prediction-problem-dataset*. To begin with, exploratory data analysis was done to uncover any hidden patterns or insightful information in the data. Both categorical and numerical features are subjected to uni-variate and bi variate analysis. This provided insightful observations that served as a platform for a solution to this classification problem.
I'd want to start by thinking about the following algorithms and going over how they react to dependent and independent properties. .And later I'd like to choose my algorithm for this classification problem.

- Random Forest
- Decision Tree
- Gradient Boost
- XGBoost.

4.1 Data Pre-processing

Data is now checked for any null or missing values and unfortunately there were some missing values found in the dataset.

```
null_values = df.isnull().sum()
null_values[null_values > 0]
```

```
Gender              13
Married              3
Dependents          15
Self_Employed       32
LoanAmount          22
Loan_Amount_Term    14
Credit_History      50
LoanAmountBin       22
dtype: int64
```

Figure 8: Null values info

For replacing the null values, median or mean is imputed for those numerical features and mode for categorical features. The class distribution (class variable) value 0s were changed to NO, indicating that the loan was nor approved, and the value 1s were converted to YES, indicating that the loan was approved. Because most algorithms require at least one nominal variable column, this is significant. Now, label encoding (normalization) is carried out for categorical features which can be used to transform non-numerical labels to numerical labels. A heat map is then displayed to determine which feature is crucial for loan approval before the data is divided into independent and dependent features, and s any remaining features that are not correlated with the label feature are dropped.

```
# Now lets plot heatmap to fetch the relationship between the variables
sns.heatmap(df.corr(), annot=True)
plt.gcf().set_size_inches(15,5)
plt.savefig('heatmap1.jpg')
```

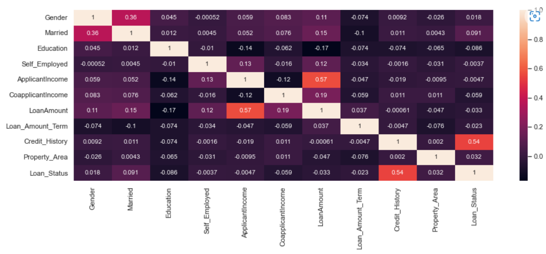


Figure 9: Null values info

Now the data is split as train and test data sets using train test split method from sci kit learn library. The ratio being 70 % for training and remaining 30% for testing. Four classification algorithms were used: Decision tree, Random forest, Gradient Boost and XGBoost. In order to predict the accuracy and maintain the precision, hyper parameter optimisation is used for tuning the classifiers parameters. The best parameters were passed in the classifier in order better accuracy.

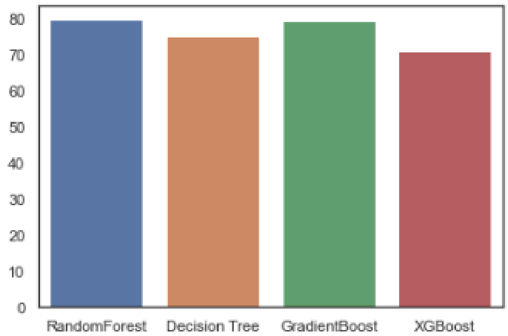


Figure 10: Classifiers accuracy's comparison Plot

From the above plot it can be vividly seen that all the classifiers performed better with almost similar accuracy's. But among them Random Forest and decision tree performed better with 78.378 % accuracy, followed by gradient boost 74.595 % and XGBoost 73.514 %. Since this classification problem comprises of only data there's no much difference in the classifiers performance. However, when working with real life data sets the accuracy would vary among the classifiers.

4.2 Evaluation Metrics

The metrics are generally used to evaluate the performance of the model when tested on unseen data. One parameter for assessing classification models is accuracy. The percentage of predictions that our model correctly predicted is known as accuracy.

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

where

TP - is a True positive,
TN - is a True Negative,
FP - is a True Negative,
FN - is a True Negative,

	Model	Score
0	Random Forest	77.838
2	Gradient Boosting	74.595
3	XGBoost	73.514
1	Decision Tree	71.892

Figure 11: Classifiers accuracy's comparison table

From the above table, it can be seen that the Random forest is the best classifier with the highest accuracy for this classification problem.

Precision is a measure of the proportion of positive class forecasts that belong to the positive class.

$$precision = \frac{TP}{TP + FP} \tag{2}$$

where

TP - is a True positive,
FP - is a True Negative,

	Model	Score
2	Gradient Boosting	0.983
0	Random Forest	0.975
3	XGBoost	0.892
1	Decision Tree	0.817

Figure 12: Precision score comparison table

In terms of precision score, gradient boosting outperformed all the classifiers with 0.983 score followed by random forest 0.975

Recall quantifies the number of valid class predictions that were generated utilising all valid examples in the data set.

$$recall = \frac{TP}{TP + FN} \tag{3}$$

where

TP - is a True positive,
FN- is a False Negative,

	Model	Score
1	Decision Tree	0.766
0	Random Forest	0.755
3	XGBoost	0.748
2	Gradient Boosting	0.724

Figure 13: recall score comparison table

In terms of recall score, decision outperformed all the classifiers with 0.766 score followed by random forest 0.775

4.3 Conclusion

In this project, four different classifiers is used and all the results were tabulated above. Among the four classifiers, random forest has achieved highest accuracy with 77.83%. However, inspite with less accuracy, the remaining classifiers such as gradientboost , XGBoost tend to perform better when worked with larger datasets. Based on the earlier assessments, random forest seems to be among the best classifiers for this issue. This does not always mean that SVM will always outperform all other classifiers, though. Everything depends on the current circumstances, as well as the methodology you are interested in and taking into consideration when evaluating the effectiveness of the classifier.

References

[1] Sotiris B Kotsiantis, Ioannis D Zaharakis, and Panayiotis E Pintelas. Machine learning: a review of classification and combining techniques. *Artificial Intelligence Review*, 26(3):159–190, 2006.

[2] Iqbal Muhammad and Zhu Yan. Supervised machine learning approaches: A survey. *ICTACT Journal on Soft Computing*, 5(3), 2015.

Classification Problem

import libraries

In [551…
```
# importing required packages
import pandas as pd
import numpy as np
#Visualization
import seaborn as sns
sns.set(style='white')
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV, C
# classifiers
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from xgboost import XGBClassifier
#metrics
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score, r
```

Load data

In [462…
```
df = pd.read_csv('train.csv')
```

In [463…
```
df.shape
```

Out[463]:
```
(614, 13)
```

In [464…
```
df.columns
```

Out[464]:
```
Index(['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education',
       'Self_Employed', 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History', 'Property_Area', 'Loan_Status'],
      dtype='object')
```

In [465…
```
# Now lets understand the data
df.head(10)
```

Out[465]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapplicant
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	
5	LP001011	Male	Yes	2	Graduate	Yes	5417	
6	LP001013	Male	Yes	0	Not Graduate	No	2333	
7	LP001014	Male	Yes	3+	Graduate	No	3036	
8	LP001018	Male	Yes	2	Graduate	No	4006	
9	LP001020	Male	Yes	1	Graduate	No	12841	

In [466…

```
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
```

From the above info it can be vivdly seen that

- There are 8 object types, 1 int, 4 float types (total 13 columns) and there are 614 instances

In [467…

```
df.describe()
```

Out[467]:

	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 [468…

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

Out[468]:

	Gender	Married	Dependents	Education	Self_Employed	Property_Area
count	601	611	599	614	582	614
unique	2	2	4	2	2	3
top	Male	Yes	0	Graduate	No	Semiurban
freq	489	398	345	480	500	233

In [469…

```
df.isnull().sum()  # Now lets check how many null values we have in each column
```

Out[469]:

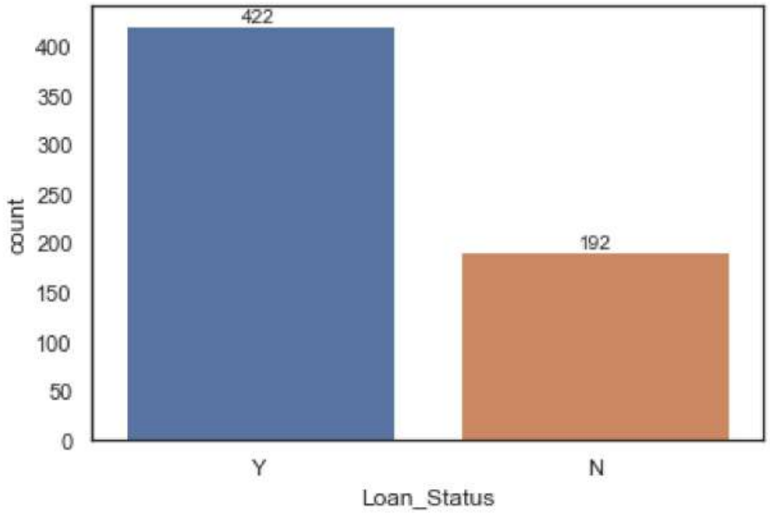
```
Loan_ID               0
Gender               13
Married               3
Dependents           15
Education             0
Self_Employed        32
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           22
Loan_Amount_Term     14
Credit_History       50
Property_Area         0
Loan_Status           0
dtype: int64
```

Univaraiate analysis for categorical features

In [470…

```
plot1 = sns.countplot(x ='Loan_Status', data = df)
for p in plot1.patches:
    plot1.annotate(format(p.get_height()), (p.get_x() + p.get_width() / 2, p.get_heigh
print('The proportion of YES class : %.2f' % (df['Loan_Status'].value_counts()[0] / le
print('The proportion of NO class : %.2f' % (df['Loan_Status'].value_counts()[1] / ler
```

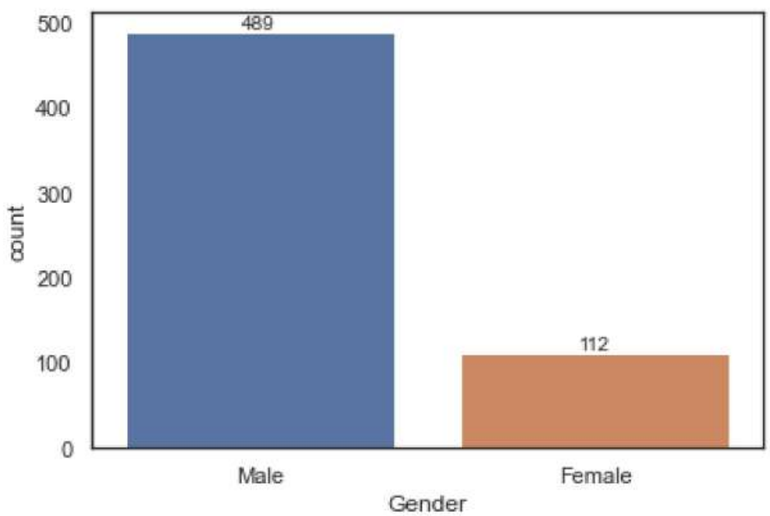
```
The proportion of YES class : 0.69
The proportion of NO class : 0.31
```



In [471…

```
# Bar Graph for Gender
plot2 = sns.countplot(x ='Gender', data = df)
for p in plot2.patches:
    plot2.annotate(format(p.get_height()), (p.get_x() + p.get_width() / 2, p.get_heigh
print('The proportion of Male : %.2f' % (df['Gender'].value_counts()[0] / len(df)))
print('The proportion of Female : %.2f' % (df['Gender'].value_counts()[1] / len(df)))
```

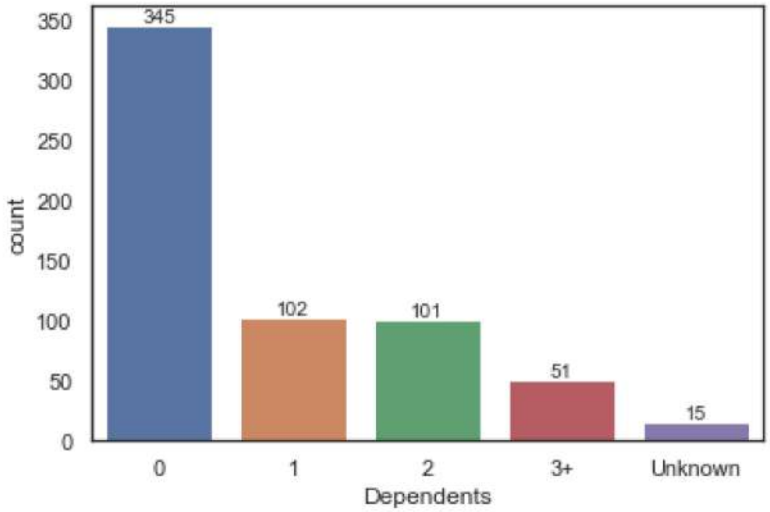
```
The proportion of Male : 0.80
The proportion of Female : 0.18
```



In [472…

```
# Bar graph for dependents
plot3 = sns.countplot(df['Dependents'].fillna('Unknown'))
for p in plot3.patches:
    plot3.annotate(format(p.get_height()), (p.get_x() + p.get_width() / 2, p.get_heigh
print('The proportion of applicants with 0 dependents : %.2f' % (df['Dependents'].valu
print('The proportion of applicants with 1 dependents  : %.2f' % (df['Dependents'].val
print('The proportion of applicants with 2 dependents  : %.2f' % (df['Dependents'].val
print('The proportion of applicants with 3+ dependents  : %.2f' % (df['Dependents'].va
```

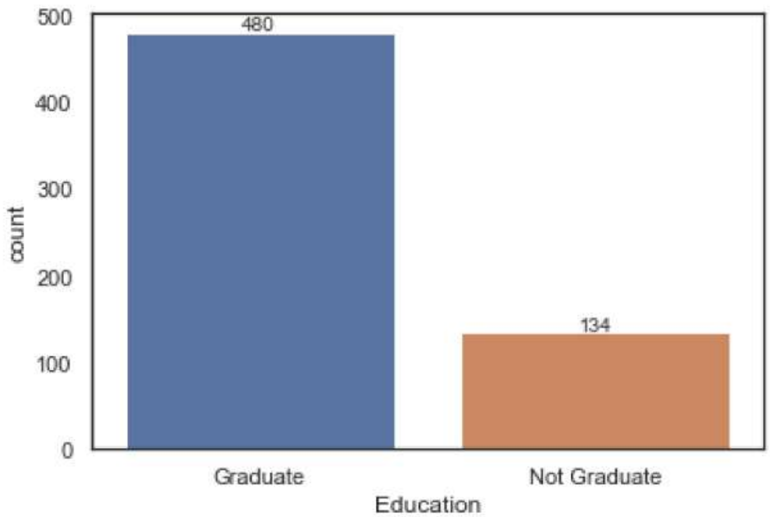
```
The proportion of applicants with 0 dependents : 0.56
The proportion of applicants with 1 dependents  : 0.17
The proportion of applicants with 2 dependents  : 0.16
The proportion of applicants with 3+ dependents  : 0.08
```



In [473…

```
# Bar graph for applicants education
plot4 = sns.countplot(x ='Education', data = df)
for p in plot4.patches:
    plot4.annotate(format(p.get_height()), (p.get_x() + p.get_width() / 2, p.get_heigh
print('The proportion of Graduate : %.2f' % (df['Education'].value_counts()[0] / len(d
print('The proportion of Not Graduate : %.2f' % (df['Education'].value_counts()[1] / l
```

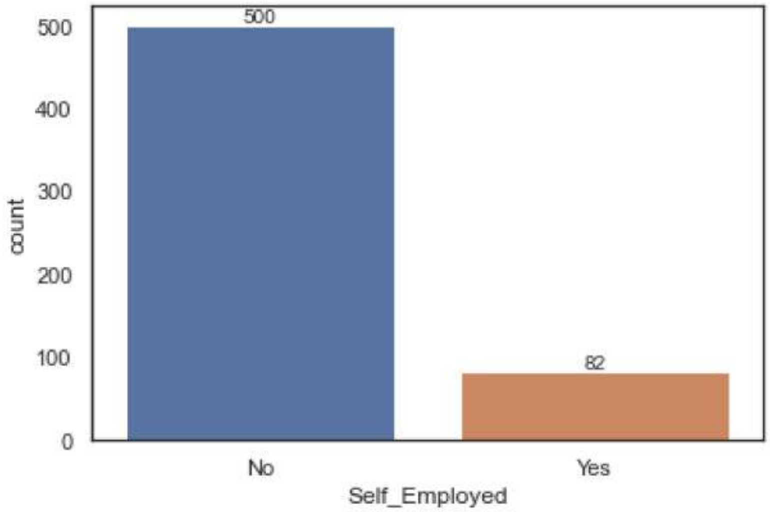
```
The proportion of Graduate : 0.78
The proportion of Not Graduate : 0.22
```



In [474…

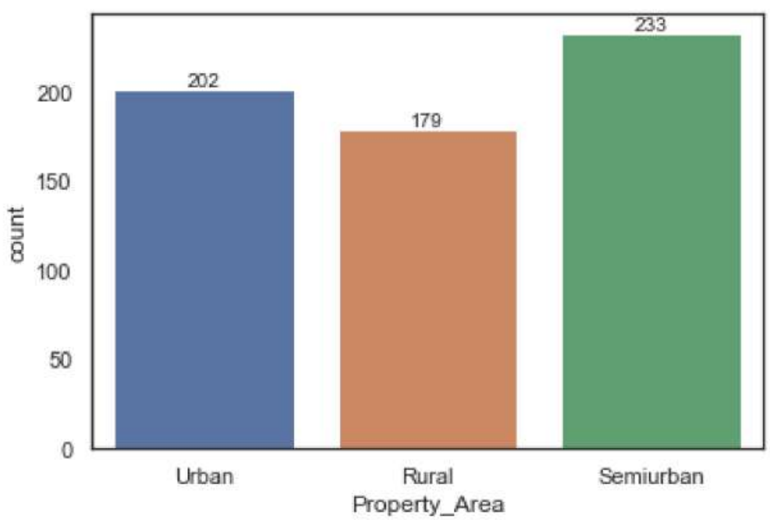
```
# Bar graph of SelfEmployed
plot5 = sns.countplot(x ='Self_Employed', data = df)
for p in plot5.patches:
    plot5.annotate(format(p.get_height()), (p.get_x() + p.get_width() / 2, p.get_heigh
print('The proportion of Yes : %.2f' % (df['Self_Employed'].value_counts()[1] / len(df
print('The proportion of No : %.2f' % (df['Self_Employed'].value_counts()[0] / len(df)
```

```
The proportion of Yes : 0.13
The proportion of No : 0.81
```



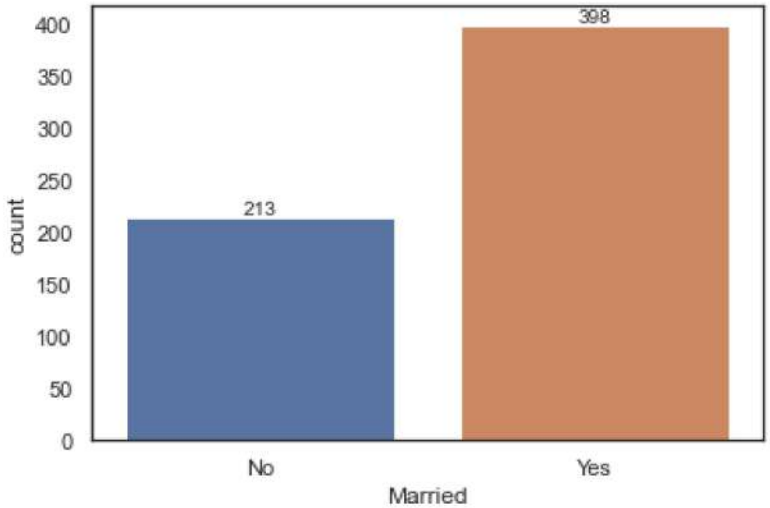
In [475...
```
# Bar graph of Property_Area
plot6 = sns.countplot(x ='Property_Area', data = df)
for p in plot6.patches:
    plot6.annotate(format(p.get_height()), (p.get_x() + p.get_width() / 2, p.get_heigh
print('The proportion of Urban : %.2f' % (df['Property_Area'].value_counts()[0] / len(
print('The proportion of Rural : %.2f' % (df['Property_Area'].value_counts()[1] / len(
print('The proportion of Semi-urban : %.2f' % (df['Property_Area'].value_counts()[2] /
```

```
The proportion of Urban : 0.38
The proportion of Rural : 0.33
The proportion of Semi-urban : 0.29
```



In [476...
```
# Bar Graph for Marriage status
plot7 = sns.countplot(x ='Married', data = df)
for p in plot7.patches:
    plot7.annotate(format(p.get_height()), (p.get_x() + p.get_width() / 2, p.get_heigh
print('The proportion of Married : %.2f' % (df['Married'].value_counts()[0] / len(df))
print('The proportion of Single : %.2f' % (df['Married'].value_counts()[1] / len(df))
```

```
The proportion of Married : 0.65
The proportion of Single : 0.35
```



In [477…
```
df.pivot_table(columns="Married",index="Dependents", values="Loan_ID", aggfunc=len)
```

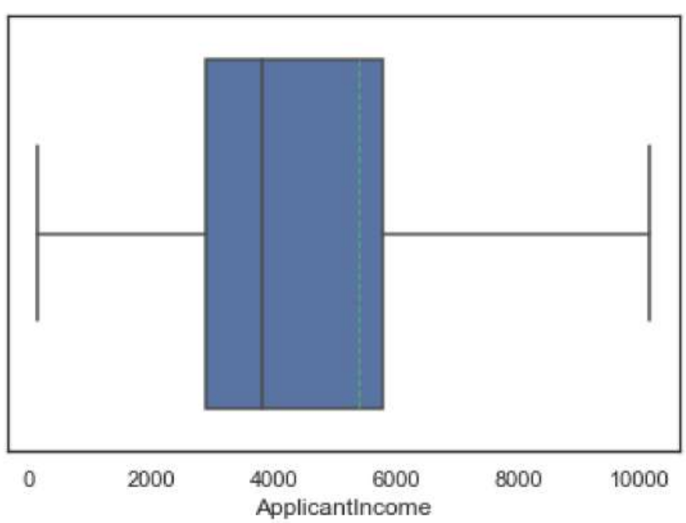
Out[477]:

Married	No	Yes
Dependents		
0	171	174
1	23	79
2	8	93
3+	7	44

Univariate analysis for continous features

In [478…
```
sns.boxplot(x="ApplicantIncome",data=df, showfliers=False, meanline=True, showmeans=Tr
```

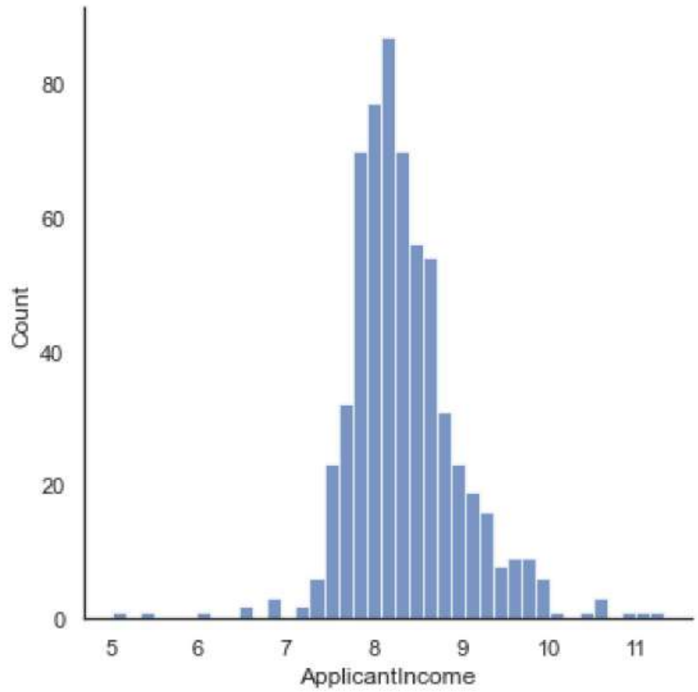
Out[478]: <AxesSubplot:xlabel='ApplicantIncome'>



- Majority of the applicants incomes lies in between 2500 and 6000 and the average income is 4000

In [479…
```
sns.displot(np.log(df.ApplicantIncome), kde=False)
```

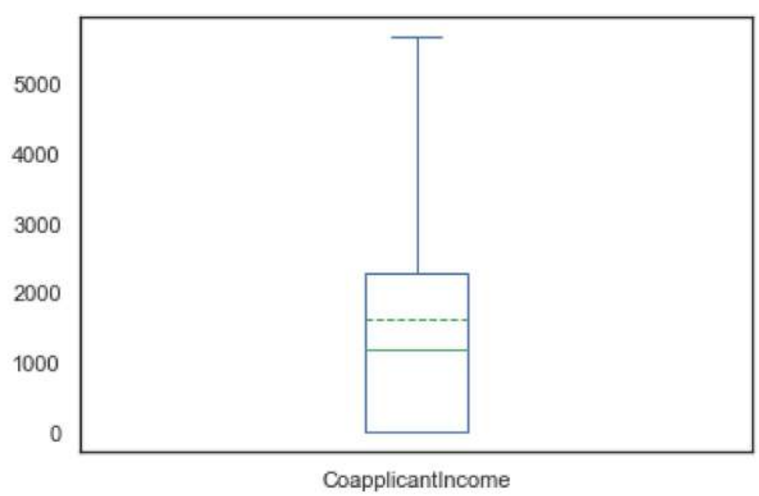
Out[479]: <seaborn.axisgrid.FacetGrid at 0x1f4fce04100>



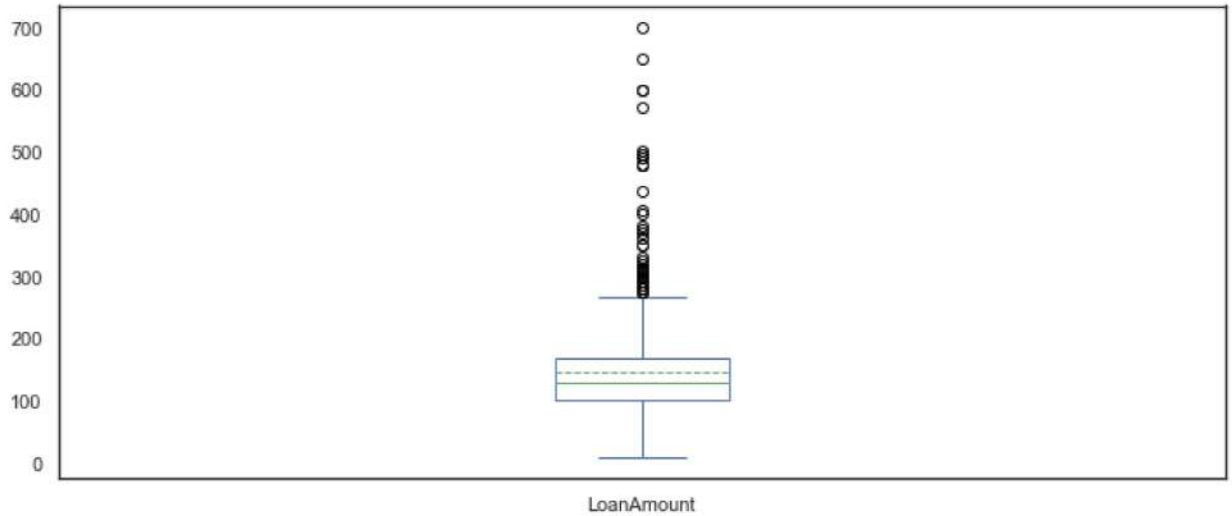
In [480…
```
# There's slight skew to the left for the applicantincome
```

In [481…
```
df.CoapplicantIncome.plot.box(showmeans=True, meanline=True, showfliers = False)
```

Out[481]: <AxesSubplot:>



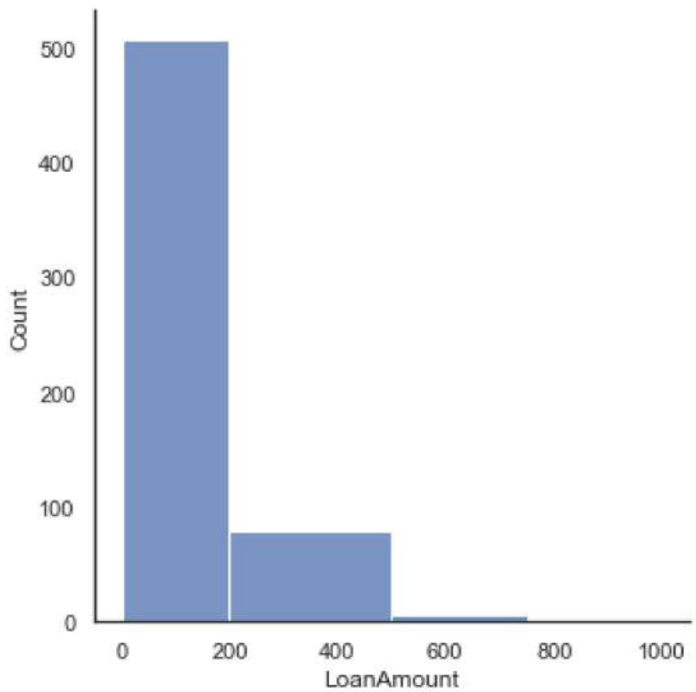
In [482…
```
df.LoanAmount.plot.box(showfliers = True, meanline=True, showmeans= True)
plt.gcf().set_size_inches(12,5)
```



In [483…

```
sns.displot(df['LoanAmount'], bins=[0,200,500,750,1000], kde=False)
```

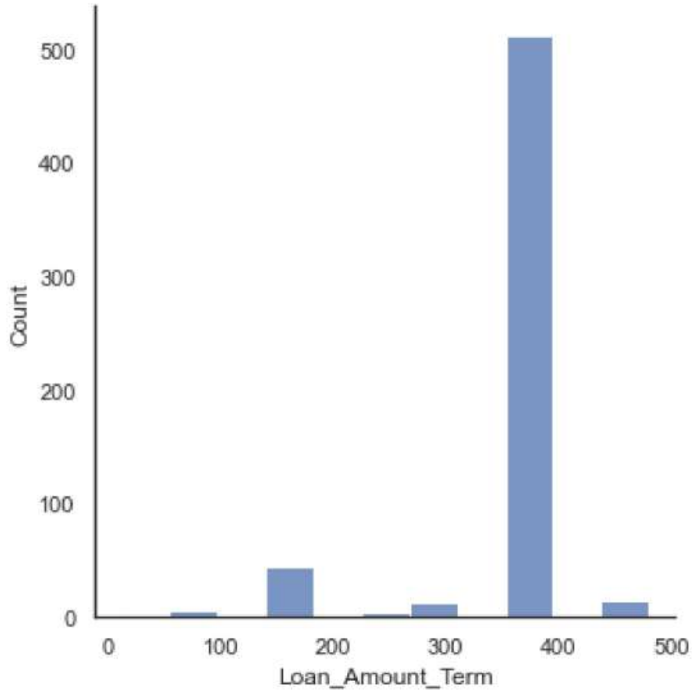
Out[483]: <seaborn.axisgrid.FacetGrid at 0x1f4f4b2d6a0>



In [484…

```
sns.displot(df["Loan_Amount_Term"], kde=False)
```

Out[484]: <seaborn.axisgrid.FacetGrid at 0x1f4fcd67b80>



Bi-variate analyis

In [485…
```
group_property_area = df.pivot_table(index='Property_Area',columns="Loan_Status", valu
print(group_property_area)
```

```
Loan_Status     N    Y
Property_Area
Rural          69  110
Semiurban      54  179
Urban          69  133
```

- It is evident that most of the loan applications are accepted for borrowers who live in semi-urban areas, followed by urban and rural areas

In [486…
```
# Rejection rate
group_property_area.iloc[:,0] / (group_property_area.iloc[:,0] + group_property_area.i
```

Out[486]:
```
Property_Area
Rural        0.385475
Semiurban    0.231760
Urban        0.341584
dtype: float64
```

- Rural areas have a greater rejection rate for applications.

In [487…
```
# Acceptance rate
group_property_area.iloc[:,1] / (group_property_area.iloc[:,0] + group_property_area.i
```

Out[487]:
```
Property_Area
Rural        0.614525
Semiurban    0.768240
Urban        0.658416
dtype: float64
```

- Likewise Urban locations have higher application acceptance rates.

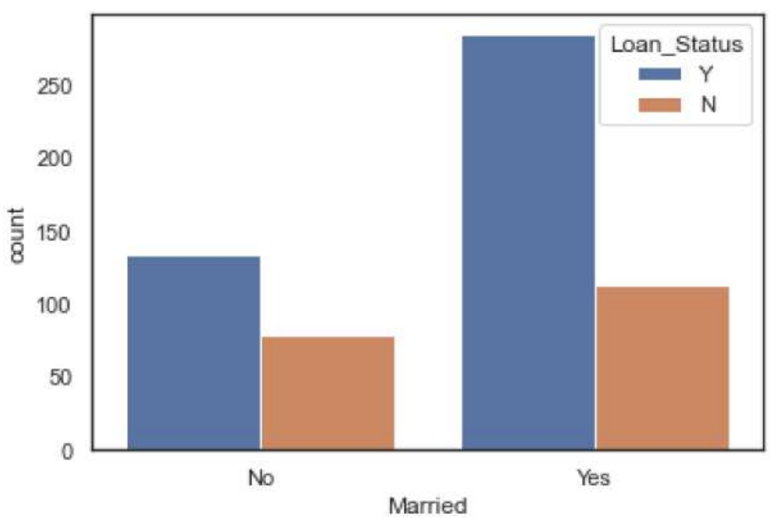
In [488…
```
df.groupby(['Property_Area','Loan_Status'])['Loan_ID'].count()
```
Out[488]:
```
Property_Area  Loan_Status
Rural          N               69
               Y              110
Semiurban      N               54
               Y              179
Urban          N               69
               Y              133
Name: Loan_ID, dtype: int64
```

In [489…
```
df.groupby(['Gender','Education'], as_index=False)['ApplicantIncome'].mean()
```
Out[489]:

	Gender	Education	ApplicantIncome
0	Female	Graduate	4646.467391
1	Female	Not Graduate	4629.700000
2	Male	Graduate	5992.345745
3	Male	Not Graduate	3630.061947

- As can be seen from the table above, male graduates have greater incomes, but interestingly, female graduates' incomes are nearly equal to those of non-graduates.

In [490…
```
sns.countplot(x="Married", data = df, hue="Loan_Status")
```
Out[490]:
```
<AxesSubplot:xlabel='Married', ylabel='count'>
```



- Loan approval rates are higher for applicants who are married and lower for candidates who are not married.

In [491…
```
df.pivot_table(index = "Married", columns="Loan_Status", values='Loan_ID',aggfunc=len)
```

Out[491]:

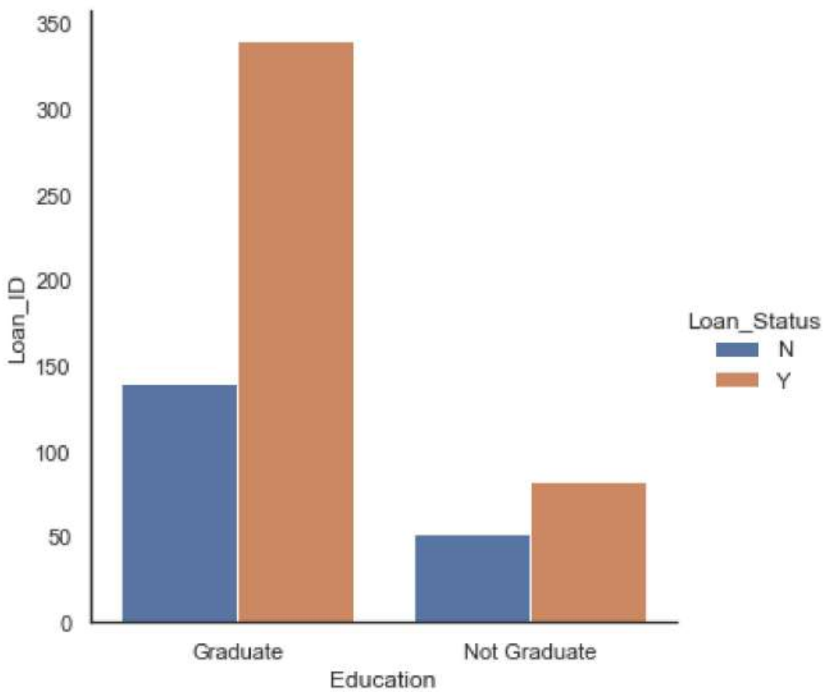
Loan_Status	N	Y
Married		
No	79	134
Yes	113	285

In [492…

```
data_edu = df.groupby(by=["Loan_Status","Education"], as_index=False)['Loan_ID'].count
sns.catplot(x="Education",y="Loan_ID", hue="Loan_Status", data=data_edu, kind="bar")
```

Out[492]:

```
<seaborn.axisgrid.FacetGrid at 0x1f4fe70ad90>
```



- Similarly, graduates have a greater loan acceptance rate than non-graduates.

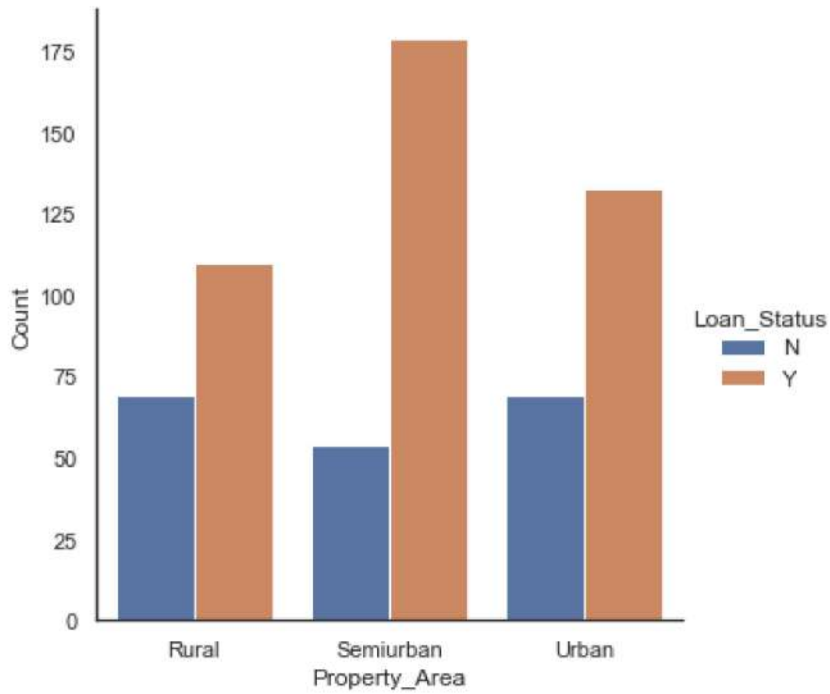
In [493…

```
data = df.groupby(by=["Loan_Status","Property_Area"], as_index=False)['Loan_ID'].count
data.rename(columns={'Loan_ID':'Count'}, inplace=True)
print(d)
sns.catplot(x="Property_Area",y="Count", hue="Loan_Status", data=data, kind='bar')
```

```
  Loan_Status  Gender  Loan_ID
0           N  Female       37
1           N    Male      150
2           Y  Female       75
3           Y    Male      339
```

Out[493]:

```
<seaborn.axisgrid.FacetGrid at 0x1f4fe86ff70>
```



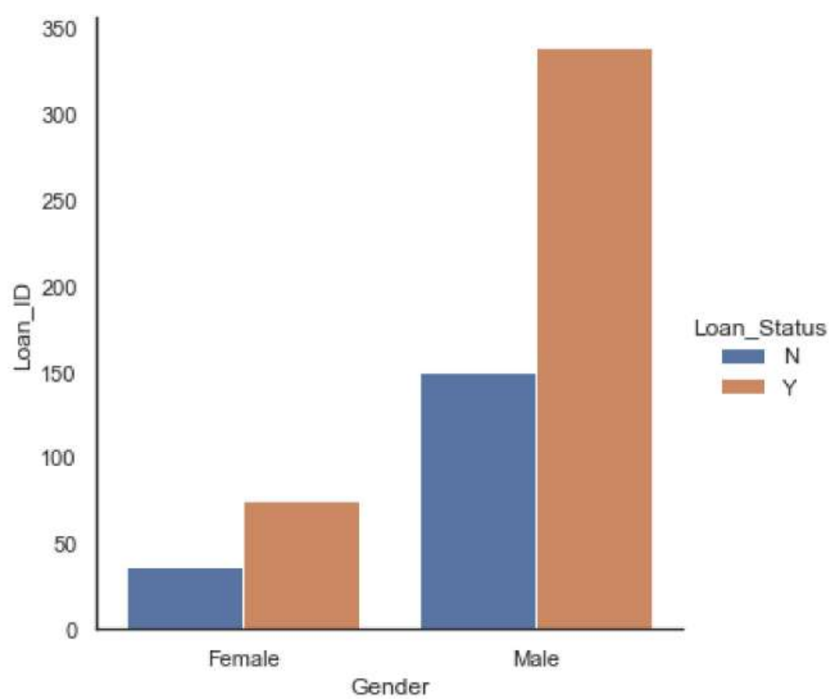
In [494…

```
data1 = df.groupby(by=["Loan_Status","Gender"], as_index=False)['Loan_ID'].count()
print(data1)
sns.catplot(x="Gender",y="Loan_ID", hue="Loan_Status", data=data1, kind="bar")
```

```
  Loan_Status  Gender  Loan_ID
0           N  Female       37
1           N    Male      150
2           Y  Female       75
3           Y    Male      339
```

Out[494]:

```
<seaborn.axisgrid.FacetGrid at 0x1f4fe968730>
```



In [495…

```
# just for visual purpose, I've created bins for applicants_income and seperated as Lc
bins=[0,100,200,700]
groups=['Low','Average','High']
```

```
df['LoanAmountBin']=pd.cut(df['LoanAmount'],bins,labels=groups)
df[['LoanAmount','LoanAmountBin']].head(10)
```

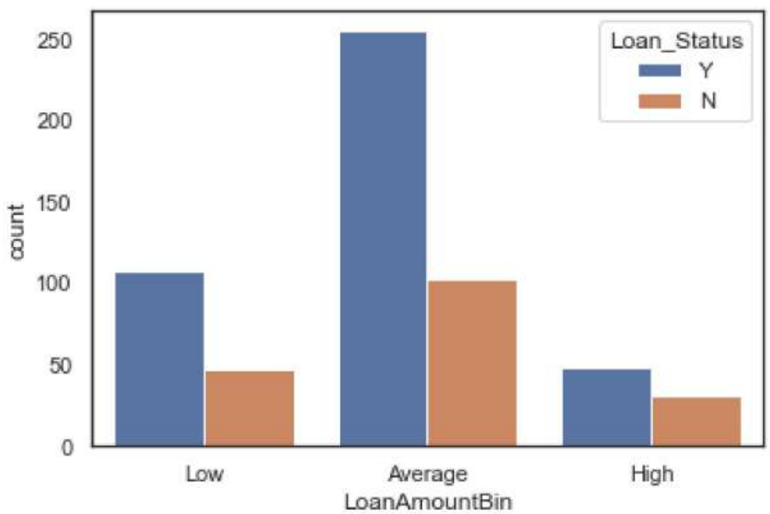
Out[495]:

	LoanAmount	LoanAmountBin
0	NaN	NaN
1	128.0	Average
2	66.0	Low
3	120.0	Average
4	141.0	Average
5	267.0	High
6	95.0	Low
7	158.0	Average
8	168.0	Average
9	349.0	High

In [496…

```
sns.countplot(x="LoanAmountBin", hue="Loan_Status", data=df)
```

Out[496]: <AxesSubplot:xlabel='LoanAmountBin', ylabel='count'>

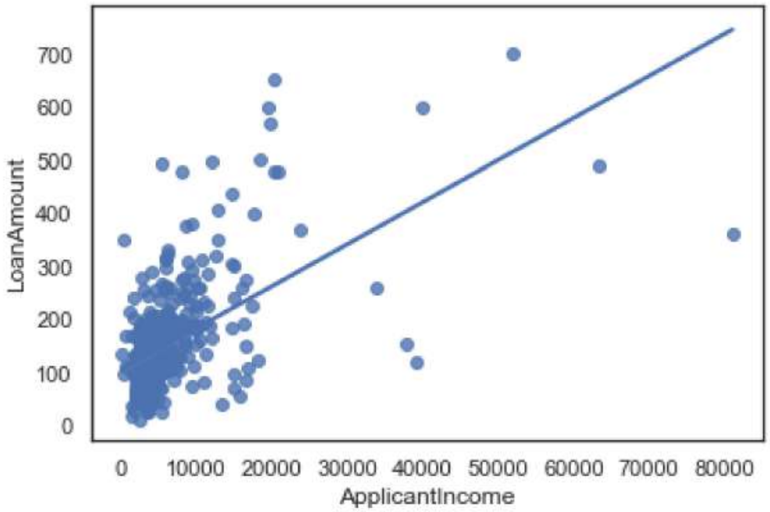


Bi-variate anlayis for continous variable

In [497…

```
sns.regplot(x="ApplicantIncome", y = "LoanAmount", data = df,ci=False)
```

Out[497]: <AxesSubplot:xlabel='ApplicantIncome', ylabel='LoanAmount'>



- It is clearly evident that the applicant's income and loan amount have positive correlation; the higher the income, the higher the likelihood of receiving a high income amount.

Data pre-processing

In [498...
```
# lets replace the class variable as 0 and 1
df['Loan_Status'].replace('N',0,inplace=True)
df['Loan_Status'].replace('Y',1,inplace=True)
```

Replacing null values

In [499...
```
null_values = df.isnull().sum()
null_values[null_values > 0]
```

Out[499]:
```
Gender              13
Married              3
Dependents          15
Self_Employed       32
LoanAmount          22
Loan_Amount_Term    14
Credit_History      50
LoanAmountBin       22
dtype: int64
```

In [500...
```
print(df['Dependents'].unique())
# replacing 3+ with 3 in dependents feature
df['Dependents'].replace(to_replace ='3+', value='3',inplace=True)
```

```
['0' '1' '2' '3+' nan]
```

In [501...
```
df['Gender'].fillna(df['Gender'].mode()[0], inplace=True)
df['Married'].fillna(df['Married'].mode()[0], inplace=True)
df['Self_Employed'].fillna(df['Self_Employed'].mode()[0], inplace=True)
```

In [502...
```
print(df['Dependents'].value_counts())
```

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

In [503…

```
df['Dependents'].fillna(df['Dependents'].mode()[0], inplace=True)
```

In [504…

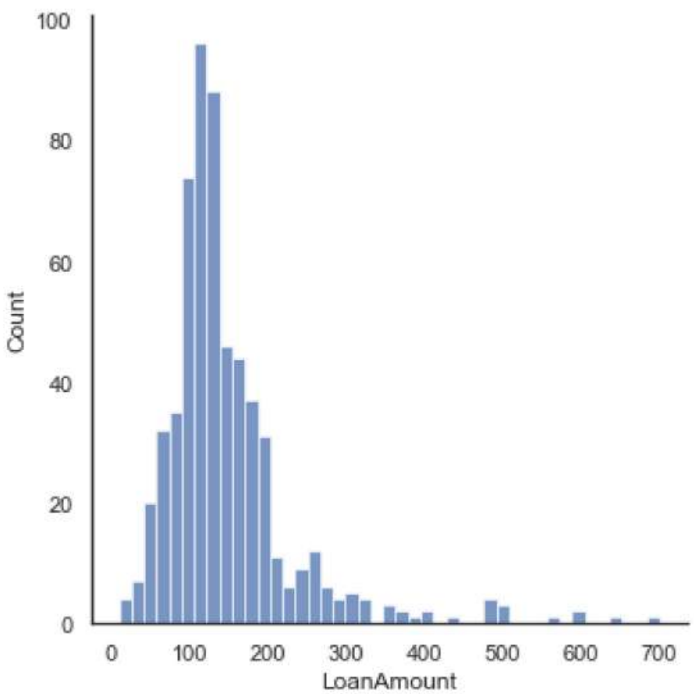
```
df['Loan_Amount_Term'].fillna(df['Loan_Amount_Term'].mode()[0], inplace=True)
```

In [505…

```
sns.displot(df["LoanAmount"], kde=False)
# since the distribution is slightlt skewed to the right, will replace LoanAmount with
```

Out[505]:

```
<seaborn.axisgrid.FacetGrid at 0x1f4ffa0bbb0>
```



In [506…

```
# since the distribution has slight slight skew to the right, I've used median for rep
df['LoanAmount'].fillna(df['LoanAmount'].median(), inplace=True)
```

In [507…

```
df.Credit_History.value_counts()
```

Out[507]:

```
1.0    475
0.0     89
Name: Credit_History, dtype: int64
```

In [508…

```
df['Credit_History'].fillna(df['Credit_History'].mode()[0], inplace=True)
```

In [509…

```
df.drop(['LoanAmountBin'], axis=1, inplace=True)
```

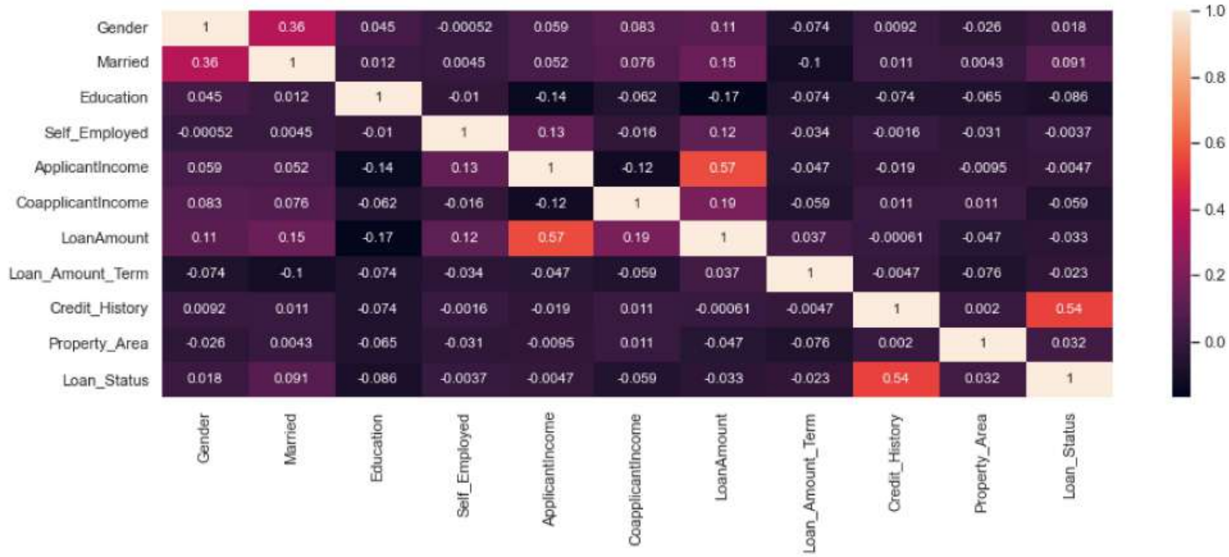
Label encoding for categorical features

In [510…

```
columns = ['Gender',"Married","Education",'Self_Employed',"Property_Area"]
le = LabelEncoder()
for column in columns:
    df[column] = le.fit_transform(df[column])
```

In [511…

```
# Now lets plot heatmap to fetch the relationship between the variables
sns.heatmap(df.corr(), annot=True)
plt.gcf().set_size_inches(15,5)
plt.savefig('heatmap1.jpg')
```



- from the above heatmap, it is evident that the 'CoapplicantIncome', "LoanAmount", "Loan_Amount_Term", 'Loan_ID', 'CoapplicantIncome', 'Dependents' doen't seems to have much correlation, so I'm dropping these features

In [512…

```
cols = [ 'CoapplicantIncome', "LoanAmount", "Loan_Amount_Term", 'Loan_ID', 'Coapplicar
df = df.drop(columns=cols, axis=1)
df.head()
```

Out[512]:

	Gender	Married	Education	Self_Employed	ApplicantIncome	Credit_History	Property_Area	Loan
0	1	0	0	0	5849	1.0	2	
1	1	1	0	0	4583	1.0	0	
2	1	1	0	1	3000	1.0	2	
3	1	1	1	0	2583	1.0	2	
4	1	0	0	0	6000	1.0	2	

splitting the data as dependent and independent features

In [513…

```
X = df.loc[:, df.columns != 'Loan_Status']
y = df.loc[:,'Loan_Status']
```

In [514…

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=
```

In [515…

```
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[515]:

```
((429, 7), (185, 7), (429,), (185,))
```

In [516…

```
# metrics to evalute the model performance
```

```python
def print_scores(y_true, y_pred):
    cm = confusion_matrix(y_true,y_pred)
    tn = cm[0,0]
    tp = cm[1,1]
    fp = cm[0,1]
    fn = cm[1,0]
    print(f"Overall Accuracy : {(tp + tn) / (tp + fp + tn + fn):.2f}")
    print(f"Recall : {tp / (tp + fn):.2f}")
    print(f"Specificity : {tn / (tn + fp):.2f}")
    print(f"Positive Precision : {tp / (tp + fp):.2f}")
    print(f"Negative Precision : {tn / (tn + fn):.2f}")
```

In [517…

```python
# Classifier - 1 -- RANDOM FOREST
rf_max_depth = [int(x) for x in np.linspace(10, 30, num = 5)] #test different values f
min_samples_split = [2, 6, 10] # minimum sample number to split a node
min_samples_leaf = [1, 3, 4] # minimum sample number that can be stored in a leaf node
bootstrap = [True, False]
param_grid = {
    'n_estimators': [5,10,15,20], # test forests of different sizes
    'max_depth': rf_max_depth,
    'min_samples_split': min_samples_split,
    'min_samples_leaf': min_samples_leaf,
    'bootstrap': bootstrap
}
rf = RandomForestClassifier()
rf_random = RandomizedSearchCV(estimator = rf,param_distributions = param_grid,
                cv = 5, n_jobs = -1)
rf_random.fit(X_train,y_train)
```

Out[517]:

```
RandomizedSearchCV(cv=5, estimator=RandomForestClassifier(), n_jobs=-1,
                   param_distributions={'bootstrap': [True, False],
                                        'max_depth': [10, 15, 20, 25, 30],
                                        'min_samples_leaf': [1, 3, 4],
                                        'min_samples_split': [2, 6, 10],
                                        'n_estimators': [5, 10, 15, 20]})
```

In [518…

```python
print ('Random grid: ', param_grid, '\n')
# print the best parameters
print ('Best Parameters: ', rf_random.best_params_)
```

```
Random grid:  {'n_estimators': [5, 10, 15, 20], 'max_depth': [10, 15, 20, 25, 30], 'm
in_samples_split': [2, 6, 10], 'min_samples_leaf': [1, 3, 4], 'bootstrap': [True, Fal
se]}

Best Parameters:  {'n_estimators': 20, 'min_samples_split': 10, 'min_samples_leaf':
4, 'max_depth': 20, 'bootstrap': True}
```

In [542…

```python
# Now lets build the model using the above obtained parameters
clf_rf = RandomForestClassifier(n_estimators=20, max_depth=20,
                    min_samples_split=10, min_samples_leaf=4, bootstrap = True)
clf_rf.fit(X_train,y_train)
y_pred = clf_rf.predict(X_test)
cv_scores = cross_val_score(clf_rf, X, y, cv=5)
print(cv_scores)
print('cv_score:', cv_scores.mean(), ' \n')
print_scores(y_test,y_pred)
```

```
[0.81300813 0.77235772 0.77235772 0.80487805 0.81147541]
cv_score: 0.7948154071704653

Overall Accuracy : 0.78
Recall : 0.97
Specificity : 0.42
Positive Precision : 0.75
Negative Precision : 0.90
```

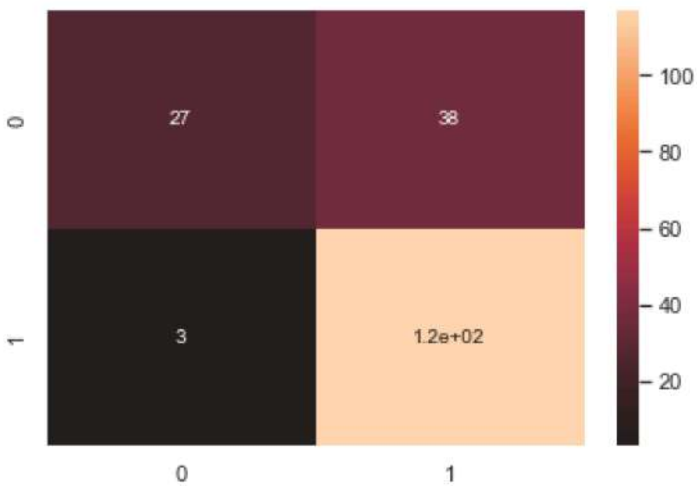
In [543…

```
mat=confusion_matrix(y_test,y_pred)
print(mat)
sns.heatmap(mat,center=True,annot=True)
```

```
[[ 27  38]
 [  3 117]]
```

Out[543]:

```
<AxesSubplot:>
```



In [520…

```
# classifer - 2 -- Decision Tree
param_grid = {
             'max_features' : ['log2' , 'sqrt' , 'auto'],
             'criterion' : ['entropy' , 'gini'],
             'max_depth' : [2,3,5,10,20,25,30,40],
             'min_samples_split': [2, 3, 10, 20],
              'min_samples_leaf': [1, 5, 8, 10]
             }
dt = DecisionTreeClassifier()
dt_random = RandomizedSearchCV(estimator = dt,param_distributions = param_grid,
                cv = 5, n_jobs = -1)
dt_random.fit(X_train,y_train)
```

Out[520]:

```
RandomizedSearchCV(cv=5, estimator=DecisionTreeClassifier(), n_jobs=-1,
                   param_distributions={'criterion': ['entropy', 'gini'],
                                        'max_depth': [2, 3, 5, 10, 20, 25, 30,
                                                      40],
                                        'max_features': ['log2', 'sqrt',
                                                         'auto'],
                                        'min_samples_leaf': [1, 5, 8, 10],
                                        'min_samples_split': [2, 3, 10, 20]})
```

In [521…

```
print ('Random grid: ', param_grid, '\n')
# print the best parameters
print ('Best Parameters: ', dt_random.best_params_)
```

```
Random grid:  {'max_features': ['log2', 'sqrt', 'auto'], 'criterion': ['entropy', 'gi
ni'], 'max_depth': [2, 3, 5, 10, 20, 25, 30, 40], 'min_samples_split': [2, 3, 10, 2
0], 'min_samples_leaf': [1, 5, 8, 10]}

Best Parameters:  {'min_samples_split': 2, 'min_samples_leaf': 8, 'max_features': 'au
to', 'max_depth': 10, 'criterion': 'gini'}
```

In [544…

```
# Now lets build the model using the above obtained parameters
clf_dt = DecisionTreeClassifier(criterion='gini', max_depth=10,
                                min_samples_leaf=8, min_samples_split=2)
clf_dt.fit(X_train,y_train)
y_pred = clf_dt.predict(X_test)
cv_scores = cross_val_score(clf_dt, X, y, cv=5)
print(cv_scores)
print('cv_score:', cv_scores.mean(), ' \n')
print_scores(y_test,y_pred)
```

```
[0.71544715 0.74796748 0.71544715 0.7804878  0.80327869]
cv_score: 0.752525656404105

Overall Accuracy : 0.72
Recall : 0.82
Specificity : 0.54
Positive Precision : 0.77
Negative Precision : 0.61
```

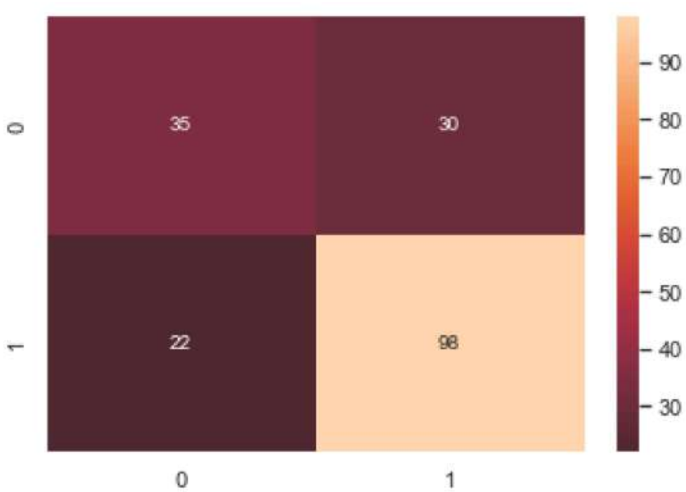
In [545…

```
#plot confusion matrix
mat=confusion_matrix(y_test,y_pred)
print(mat)
sns.heatmap(mat,center=True,annot=True)
```

```
[[35 30]
 [22 98]]
```

Out[545]:

```
<AxesSubplot:>
```



In [538…

```
# Classifier - 3 -- GradientBoosting
clf_gb = GradientBoostingClassifier(n_estimators=5)
clf_gb.fit(X_train,y_train)
clf_gb.score(X_train,y_train)
y_pred = clf_gb.predict(X_test)
cv_scores = cross_val_score(clf_gb, X, y, cv=5)
print(cv_scores)
print('cv_score:', cv_scores.mean(), ' \n')
print_scores(y_test,y_pred)
```

```
[0.78861789 0.7804878  0.75609756 0.84552846 0.81967213]
cv_score: 0.7980807676929229

Overall Accuracy : 0.75
Recall : 0.98
Specificity : 0.31
Positive Precision : 0.72
Negative Precision : 0.91
```

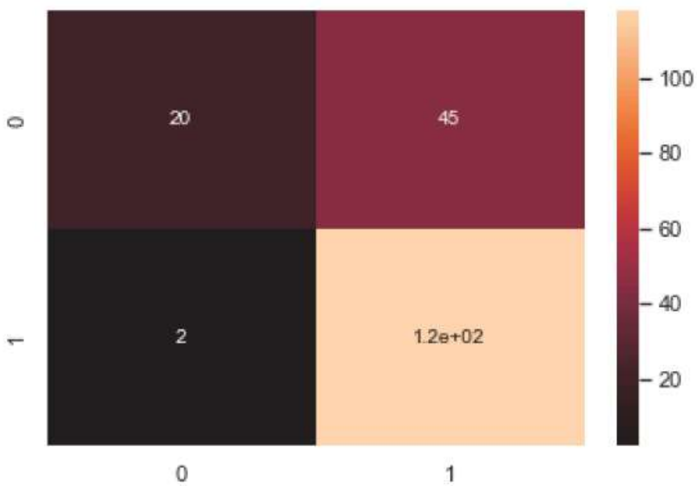
In [539…

```
mat=confusion_matrix(y_test,y_pred)
print(mat)
sns.heatmap(mat,center=True,annot=True)
```

```
[[ 20  45]
 [  2 118]]
```

Out[539]:

```
<AxesSubplot:>
```



In [524…

```
errors = {}
tree_counts = [3,5,8,10,15,20]
for count in tree_counts:
    model = GradientBoostingClassifier(n_estimators=count)
    model.fit(X_train,y_train)
    y_pred = model.predict(X_test)
    score = accuracy_score(y_test,y_pred)
    errors[count] = 1 - score
errors
```

Out[524]:

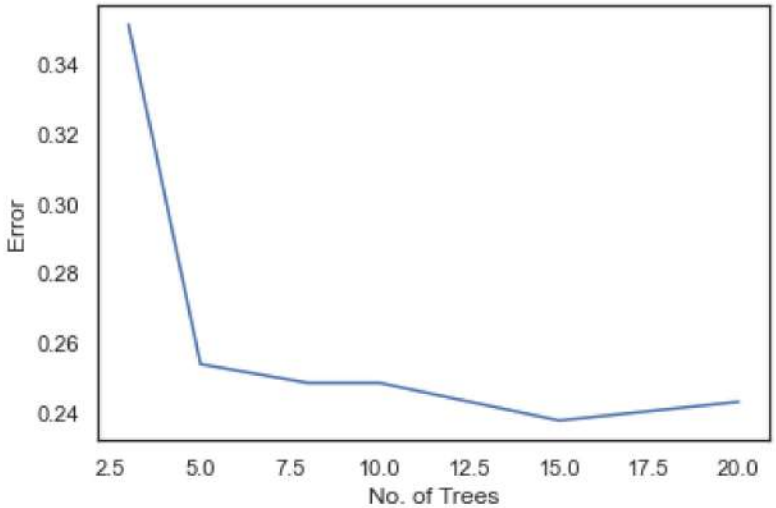
```
{3: 0.3513513513513513,
 5: 0.254054054054054,
 8: 0.24864864864864866,
 10: 0.24864864864864866,
 15: 0.23783783783783785,
 20: 0.2432432432432432}
```

In [453…

```
# plot to show error rate and no. of trees
plt.plot(list(errors.keys()), list(errors.values()))
plt.xlabel("No. of Trees")
plt.ylabel("Error")
```

Out[453]:

```
Text(0, 0.5, 'Error')
```



In [540…

```
# Classifier -4 -- XGBoost
from xgboost import XGBClassifier
clf_xgboost = XGBClassifier(n_estimators=500)
clf_xgboost.fit(X_train, y_train)
# predict
y_pred = clf_xgboost.predict(X_test)
print_scores(y_test,y_pred)
```

```
[17:11:14] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0, the default ev
aluation metric used with the objective 'binary:logistic' was changed from 'error' to
'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
Overall Accuracy : 0.74
Recall : 0.89
Specificity : 0.45
Positive Precision : 0.75
Negative Precision : 0.69
```

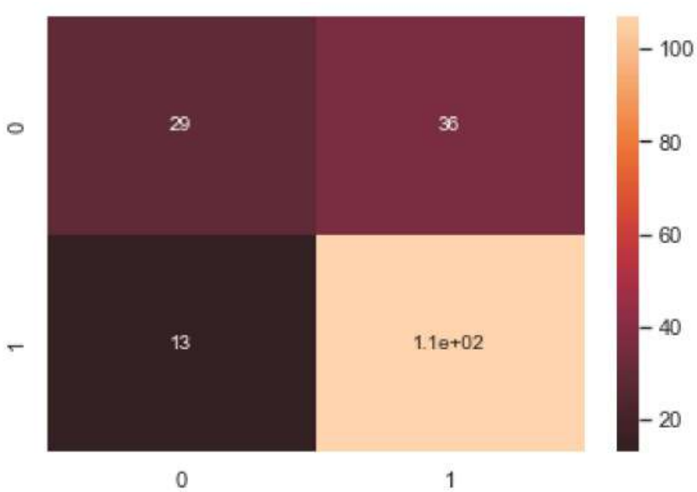
In [541…

```
mat=confusion_matrix(y_test,y_pred)
print(mat)
sns.heatmap(mat,center=True,annot=True)
```

```
[[ 29  36]
 [ 13 107]]
```

Out[541]:

```
<AxesSubplot:>
```



In [526…

```
names = ['RandomForest','Decision Tree','GradientBoost', 'XGBoost']
```

```
classifiers =  [clf_rf, clf_dt, clf_gb, clf_xgboost ]
all_scores = {}
import warnings
warnings.filterwarnings('ignore')
```

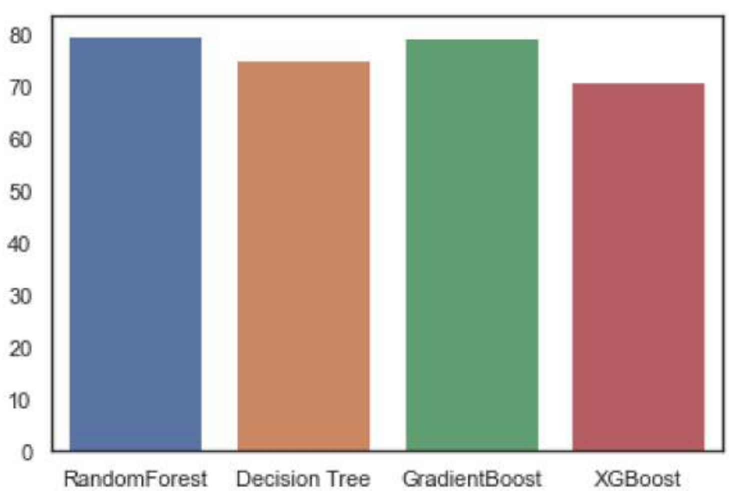
In [527…

```
for clf,name in zip(classifiers, names):
   scores = cross_val_score(clf,X,y,cv=5)
   print(scores)
   print(f"{name} = {scores.mean()*100:0.2f}")
   all_scores[name] = (scores.mean() * 100)
```

```
[0.80487805 0.77235772 0.77235772 0.82113821 0.82786885]
RandomForest = 79.97
[0.71544715 0.74796748 0.71544715 0.7804878  0.80327869]
Decision Tree = 75.25
[0.78861789 0.7804878  0.75609756 0.84552846 0.81967213]
GradientBoost = 79.81
[17:03:52] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0, the default ev
aluation metric used with the objective 'binary:logistic' was changed from 'error' to
'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
[17:03:53] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0, the default ev
aluation metric used with the objective 'binary:logistic' was changed from 'error' to
'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
[17:03:53] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0, the default ev
aluation metric used with the objective 'binary:logistic' was changed from 'error' to
'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
[17:03:54] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0, the default ev
aluation metric used with the objective 'binary:logistic' was changed from 'error' to
'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
[17:03:54] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0, the default ev
aluation metric used with the objective 'binary:logistic' was changed from 'error' to
'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
[0.68292683 0.73170732 0.68292683 0.74796748 0.72131148]
XGBoost = 71.34
```

In [528…

```
plt = sns.barplot(x = list(all_scores.keys()), y = list(all_scores.values()))
#ax = plt.set_xticklabels(labels = all_scores.keys(),rotation=60)
```



In [553…

```
## Now lets arrange all the accuracy score obtained
accuracy = pd.DataFrame({
    'Model': ['Random Forest', 'Decision Tree','Gradient Boosting ', 'XGBoost'],
    'Score': [np.round(accuracy_score(clf.predict(X_test) ,y_test) * 100, 3) for clf i
accuracy.sort_values(by='Score', ascending=False)
```

Out[553]:

	Model	Score
0	Random Forest	77.838
2	Gradient Boosting	74.595
3	XGBoost	73.514
1	Decision Tree	71.892

In [554…

```
recall = pd.DataFrame({
    'Model': ['Random Forest', 'Decision Tree','Gradient Boosting ', 'XGBoost'],
    'Score': [np.round(recall_score(clf.predict(X_test) ,y_test), 3) for clf in classi
recall.sort_values(by='Score', ascending=False)
```

Out[554]:

	Model	Score
1	Decision Tree	0.766
0	Random Forest	0.755
3	XGBoost	0.748
2	Gradient Boosting	0.724

In [555…

```
precision = pd.DataFrame({
    'Model': ['Random Forest', 'Decision Tree','Gradient Boosting ', 'XGBoost'],
    'Score': [np.round(precision_score(clf.predict(X_test) ,y_test), 3) for clf in cla
models.sort_values(by='Score', ascending=False)
```

Out[555]:

	Model	Score
2	Gradient Boosting	0.983
0	Random Forest	0.975
3	XGBoost	0.892
1	Decision Tree	0.817

In [ ]: