PREDICTION OF PERSONAL LOAN APPROVAL USING MACHINE LEARNING

INTRODUCTION

OVERVIEW

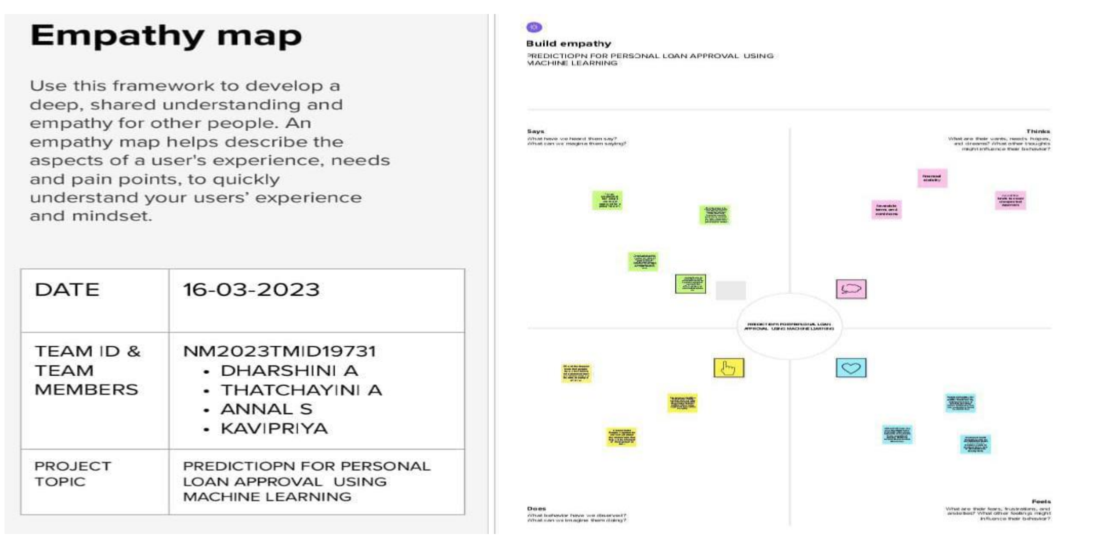
A loan is the core business part of banks. The main portion the bank's profit is directly come from the profit earned from the loans. Though bank approves loan after a regress process of verification and testimonial but still there's no surety whether the chosen hopeful is the right hopeful or not. This process takes fresh time while doing it manually. We can prophesy whether that particular hopeful is safe or not and the whole process of testimonial is automated by machine literacy style. Loan Prognostic is really helpful for retainer of banks as well as for the hopeful also

PURPOSE

When you fill out a loan application, you may come across a section that asks you to specify the purpose of the loan. Some lenders do this to give you the right product. They may also use your loan objective to assess risk and specify loan terms.There are several reasons you might consider taking out a Personal Loan. Most people have something special on their minds when they decide to borrow money. Three out of every four people considering taking out a Personal Loan say the decision is driven by a specific upcoming need or life event

PROBLEM DEFINITION & DESIGN THINKING

EMPATHY MAP



IDEATION & BRAINSTORMING MAP

Brainstorm & idea prioritization

Use this template in your own brainstorming sessions so your team can unleash their imagination and start shaping concepts even if you're not sitting in the same room.

- **10 minutes** to prepare
- **1 hour** to collaborate
- **2-8 people** recommended

ideation phase
brainstrom & idea prioritization template

date	16-03-2023
team	NM2023TMID19731 • DHARSHINI • THATCHAYINI • KAVIPRIYA • ANNAL
project name	prediction fpr personal loan approval using machine language

Before you collaborate

A little bit of preparation goes a long way with this session. Here's what you need to do to get going.

10 minutes

A **Team gathering**
Define who should participate in the session and send an invite. Share relevant information or pre-work ahead.

B **Set the goal**
Think about the problem you'll be focusing on solving in the brainstorming session.

C **Learn how to use the facilitation tools**
Use the Facilitation Superpowers to run a happy and productive session.

Open article →

1

Define your problem statement

What problem are you trying to solve? Frame your problem as a How Might We statement. This will be the focus of your brainstorm.

5 minutes

PROBLEM
PREDICTION FOR PERSONAL LOAN APPROVAL USING MACHINE LEARNING

Key rules of brainstorming
To run an smooth and productive session

- Stay in topic.
- Encourage wild ideas.
- Defer judgment.
- Listen to others.
- Go for volume.
- If possible, be visual.

2

Brainstorm

Write down any ideas that come to mind that address your problem statement.

10 minutes

TIP
You can select a sticky note and hit the pencil [switch to sketch] icon to start drawing!

DHARSHINI .A

- ability to accurately
- based on applicant information
- minimize the false positive
- comply with regulation

THATCHAYINI S

- comply with timprove
- in our banking system
- main source of income
- take care of issue depentent on their industry information

ANNAL S

- allenging to manage by the bank employees
- [illegible] to choose deserving customer from the number of applicants
- [illegible] regression, decision tree, random forest, and neural networks

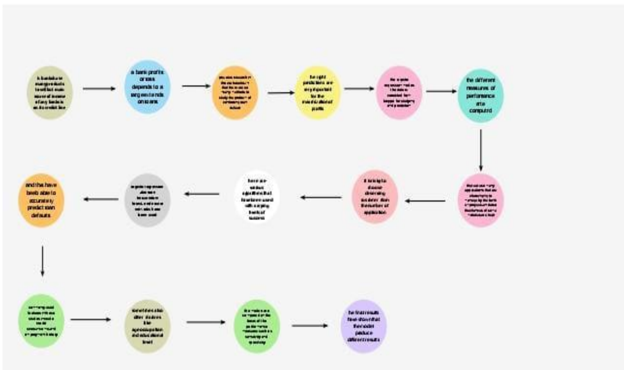
KAVIPRIYA R

- ommonly used features in these studies
- studies include credit score, income and employment history
- sometimes also other features like age, occupation and education level

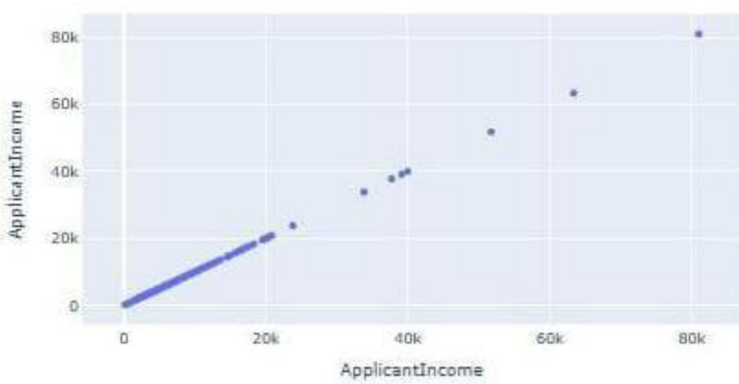
Group ideas

Take turns sharing your ideas while clustering similar or related notes as you go. Once all sticky notes have been grouped, give each cluster a sentence-like label. If a cluster is bigger than six sticky notes, try and see if you and break it up into smaller sub-groups.

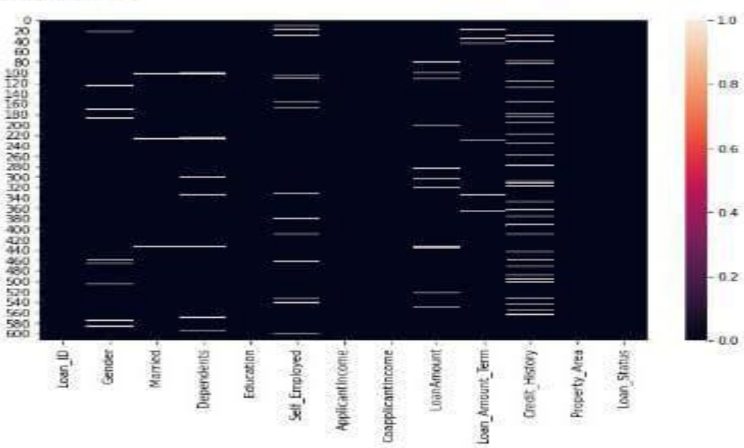
20 minutes

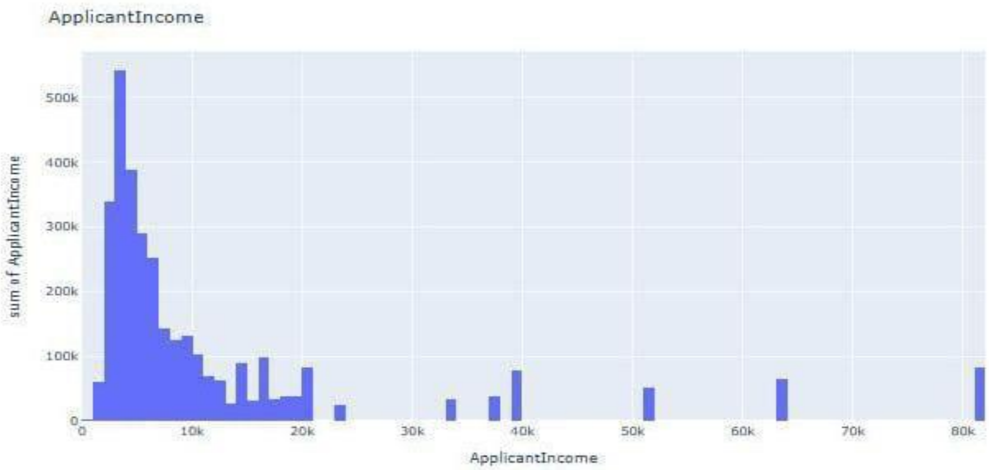


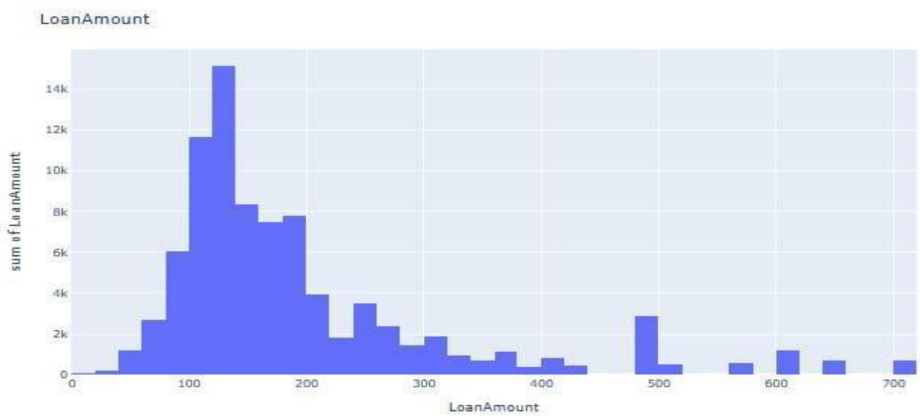
RESULT

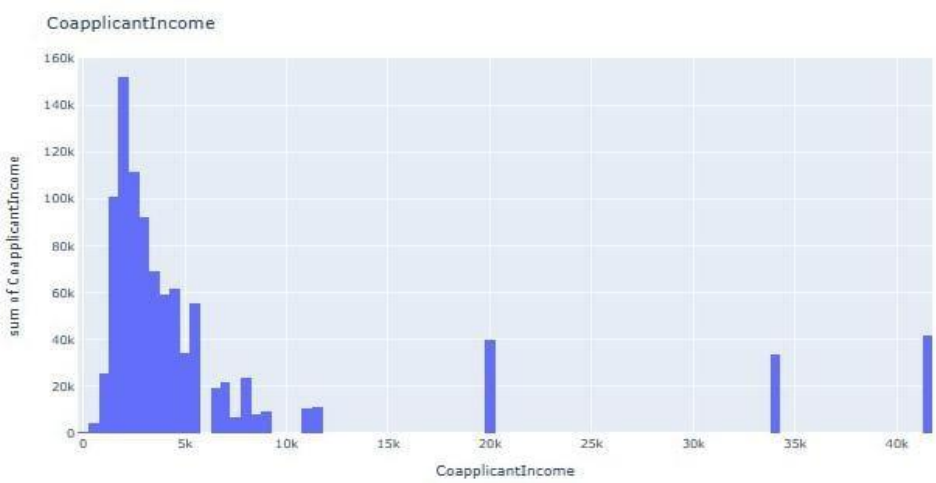


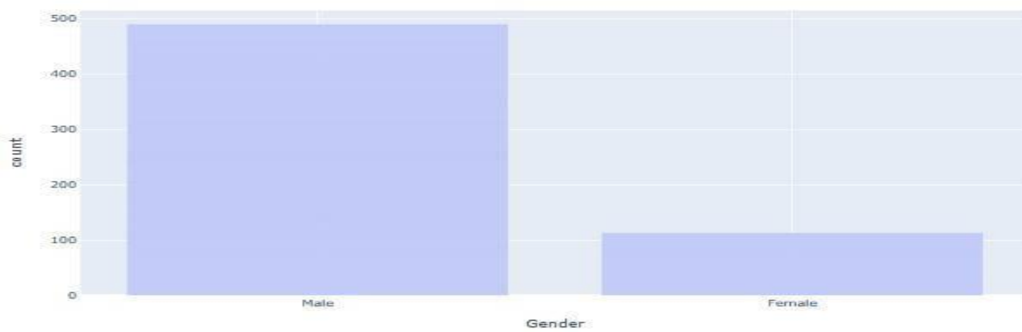
Out[8]: <AxesSubplot:>

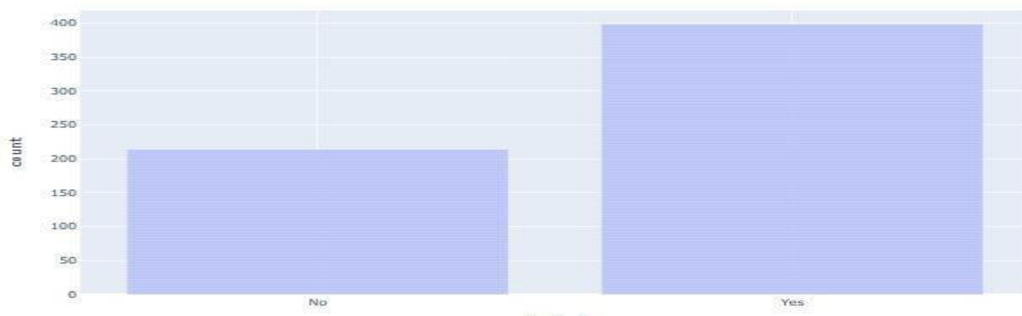


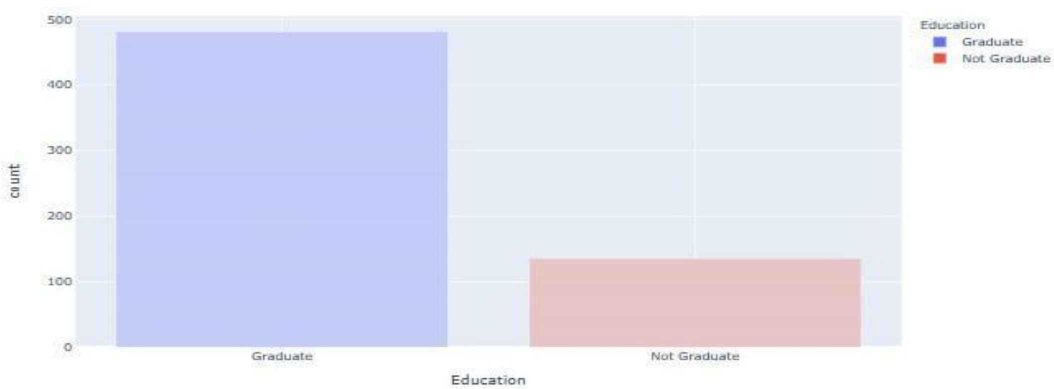

ApplicantIncome
sum of ApplicantIncome
ApplicantIncome

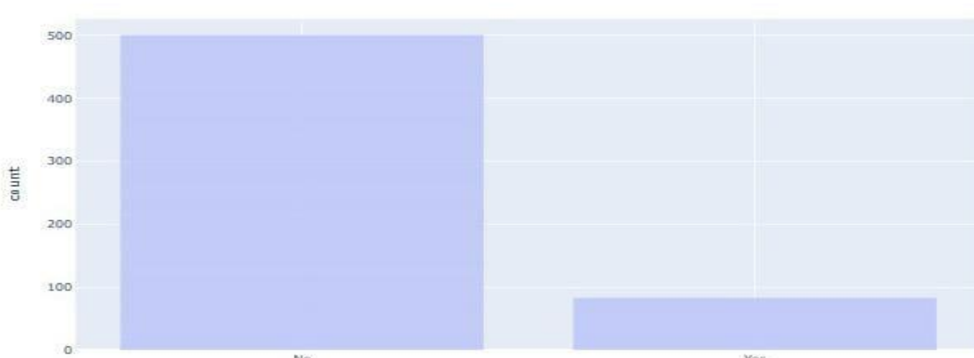

LoanAmount
sum of LoanAmount
LoanAmount

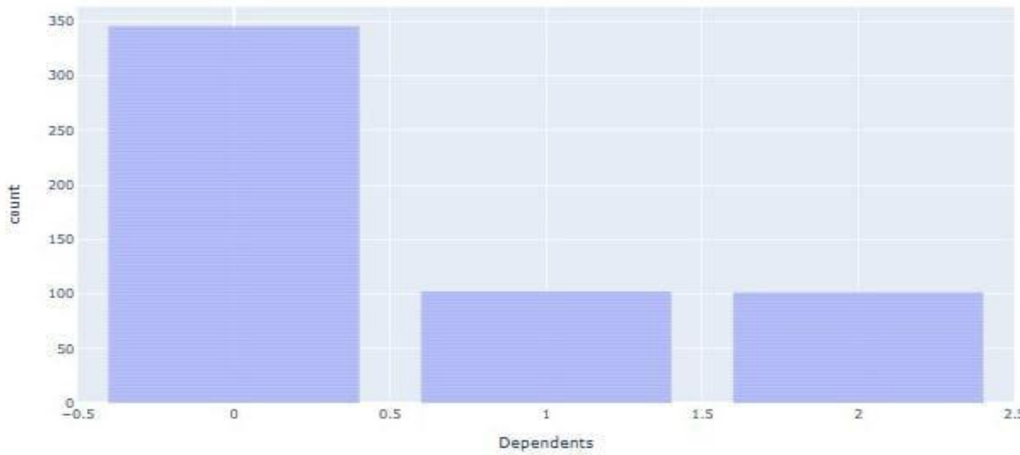

CoapplicantIncome
sum of CoapplicantIncome
CoapplicantIncome


count
500
400
300
200
100
0
Male
Female
Gender

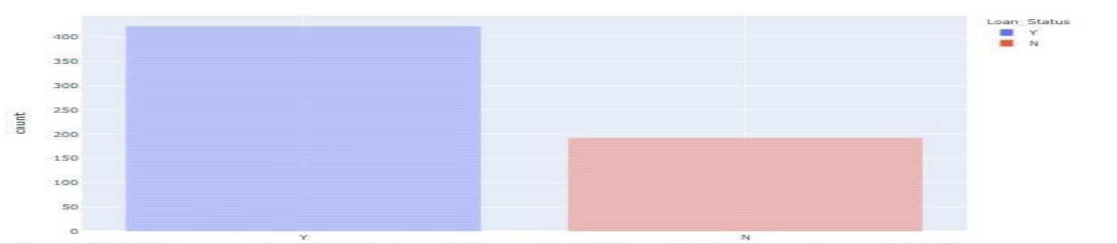

count
400
350
300
250
200
150
100
50
0
No
Yes

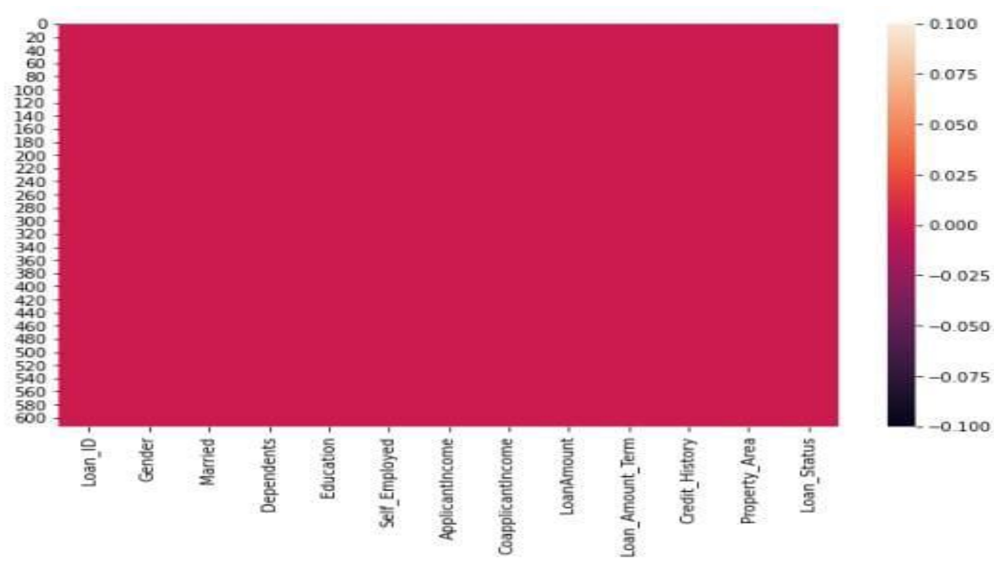

Education
Graduate
Not Graduate
count
500
400
300
200
100
0
Graduate
Not Graduate
Education

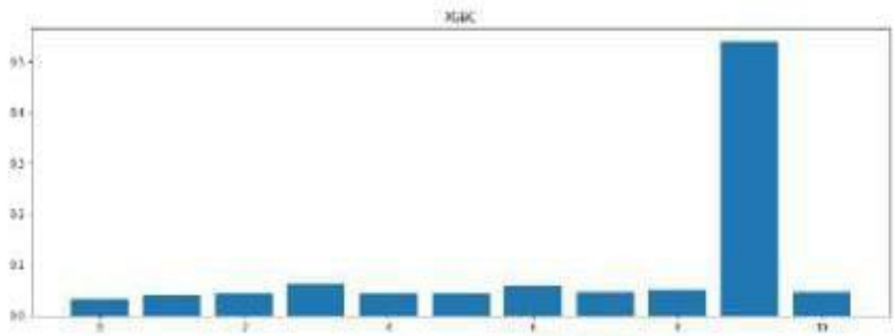

count
500
400
300
200
100
0
No
Yes

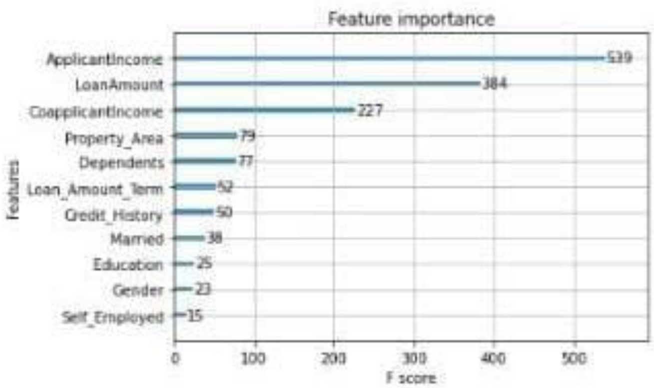

count
350
300
250
200
150
100
50
0
−0.5
0
0.5
1
1.5
2
2.5
Dependents

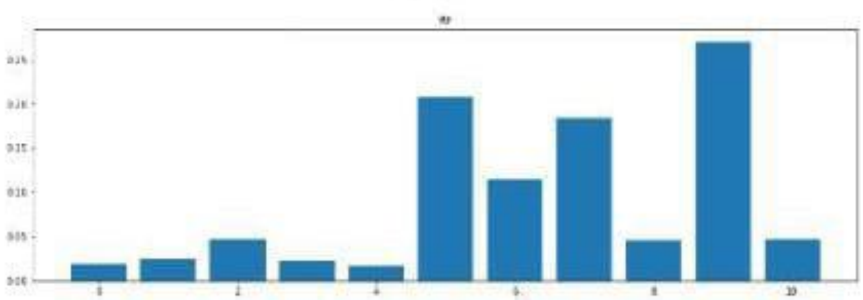

count
200
150
100
50
0
Urban
Rural
Semiurban
Property_Area

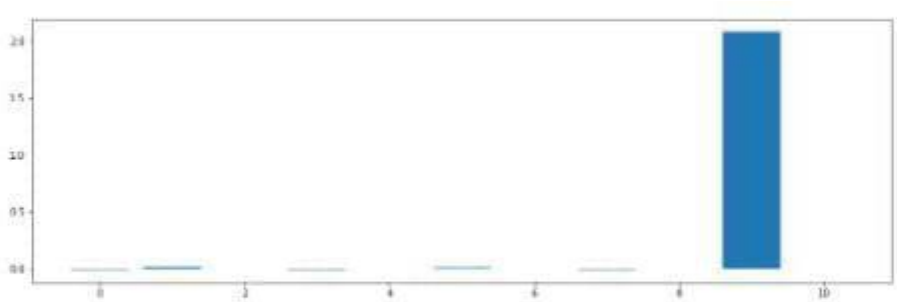

Loan_Status
Y
N
count
400
350
300
250
200
150
100
50
0
Y
N

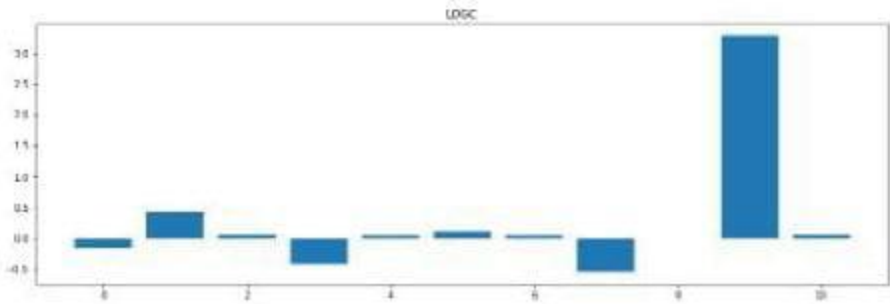

0.100
0.075
0.050
0.025
0.000
−0.025
−0.050
−0.075
−0.100
Loan_ID
Gender
Married
Dependents
Education
Self_Employed
ApplicantIncome
CoapplicantIncome
LoanAmount
Loan_Amount_Term
Credit_History
Property_Area
Loan_Status

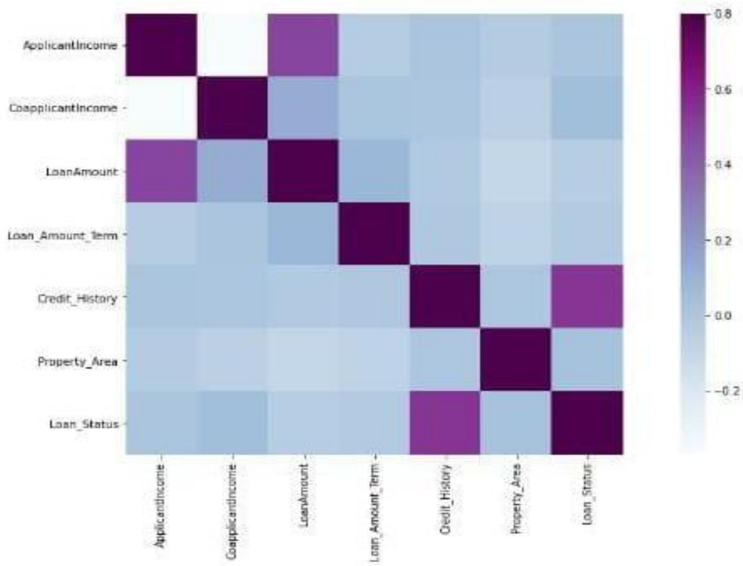



Feature importance
ApplicantIncome 539
LoanAmount 384
CoapplicantIncome 227
Property_Area 79
Dependents 77
Loan_Amount_Term 52
Credit_History 50
Married 38
Education 25
Gender 23
Self_Employed 15
Features
0 100 200 300 400 500
F score

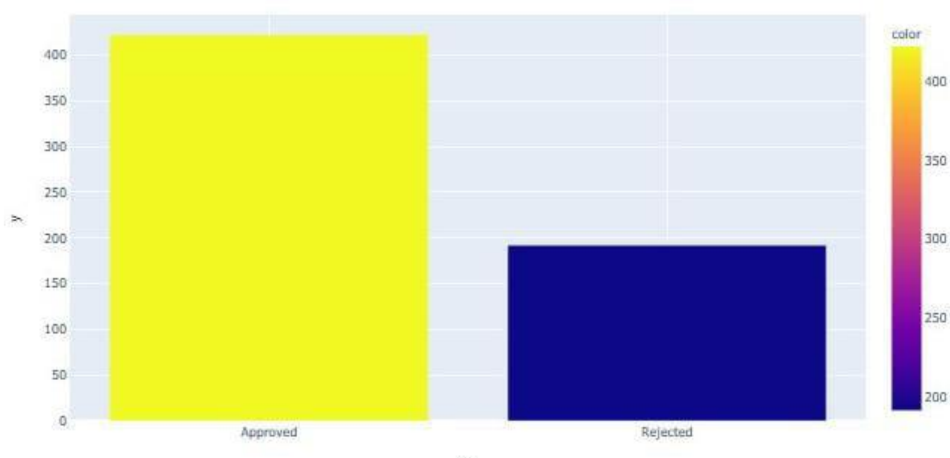




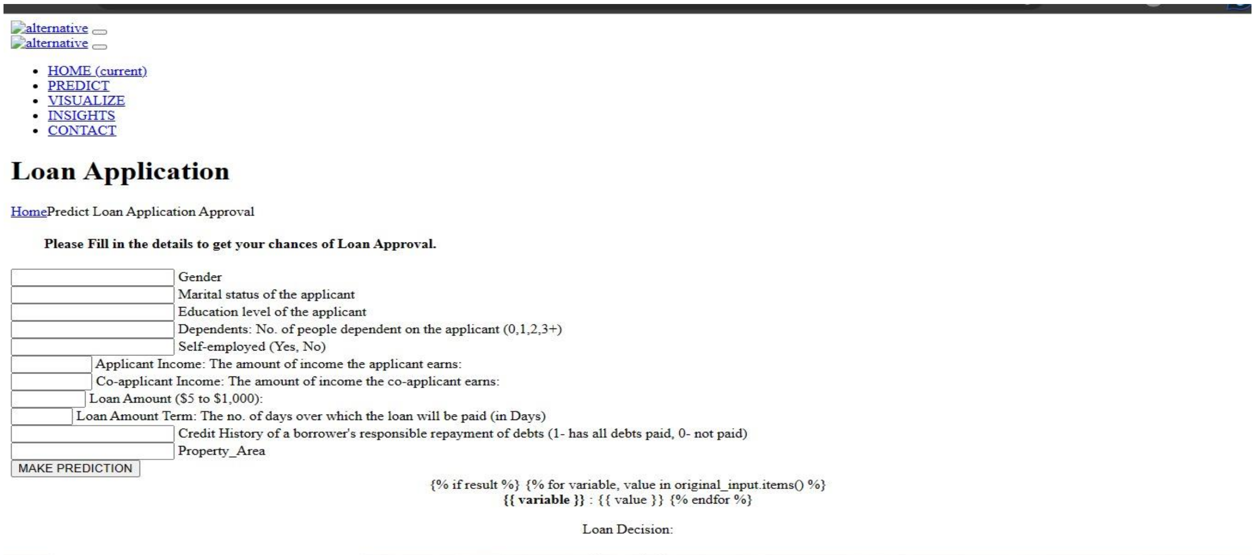


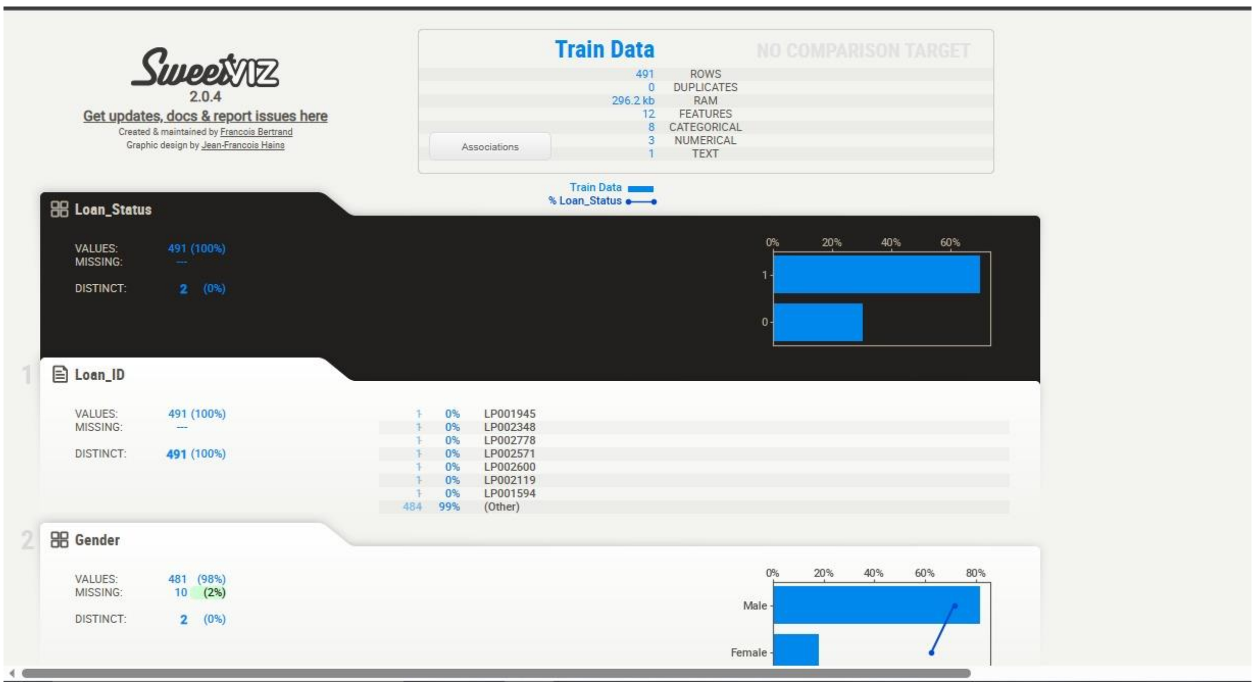


ApplicantIncome
CoapplicantIncome
LoanAmount
Loan_Amount_Term
Credit_History
Property_Area
Loan_Status
0.8
0.6
0.4
0.2
0.0
-0.2



color
400
350
300
250
200
150
100
50
0
y
Approved
Rejected
x





ADVANTAGES & DISADVANTAGES

ADVANTAGES

The Loan Prediction System can can automatically calculate the weight of each features taking part in loan processing and on new test data same features are processed with respect to their associated weight . A time limit can be set for the applicant to check whether his/her loan can be sanctioned or not.

It is done by predicting if the loan can be given to that person on the basis of various parameters like credit score, income, age, marital status, gender, etc. The prediction model not only helps the applicant but also helps the bank by minimizing the risk and reducing the number of defaulters

Accuracy—one of the primary benefits of using machine learning for credit scoring is its accuracy. Unlike human manual processing, ML-based models are automated and less likely to make mistakes. This means that loan processing becomes not only faster but more accurate, too, cutting costs on the whole.

DISADVANTAGES

The disadvantage of this model is that it emphasize different weights to each factor but in real life sometime loan can be approved on the basis of single strong factor only, which is not possible through this system. Loan Prediction is very helpful for employee of banks as well as for the applicant also.

1.cons of personal loan

2.Interest rates can be higher than alternatives.

3.More eligibility requirements.

4.Fees and penalties can be high.

5.Additional monthly payment.

6.Increased debt load.

7.Higher payments than credit cards.

8.Potential credit damage

APPLICATION

Personal loans are borrowed money that can be used for large purchases, debt consolidation, emergency expenses and much more. These loans are paid back in monthly installments over the course of a few months or upwards of a few years. It can take longer depending on your circumstances and how diligent you are with making payments.

In some cases, you might want to try something else before taking out a personal loan, like a small purchase or negotiating a lower price or cost.

*Personal loans are loans that can cover a number of personal expenses.

*You can find personal loans through banks, credit unions, and online lenders.

with no collateral needed.

*Personal loans can vary greatly when it comes to their interest rates, fees, amounts, and repayment terms.

CONCLUSION

From a proper analysis of positive points and constraints on the member, it can be safely concluded that the product is a considerably productive member. This use is working duly and meeting to all Banker requisites. This member can be freely plugged in numerous other systems. There have been mathematics cases of computer glitches, violations in content and most important weight of features is fixed in automated prophecy system, so in the near future the so – called software could be made more secure, trustworthy and dynamic weight conformation. In near future this module of prophecy can be integrated with the module of automated processing system. The system is trained on old training dataset in future software can be made resembling that new testing date should also take part in training data after some fix time.

FUTURE SCOPE

This project work can be extended to higher level in future.for example,A predivtive model for loans that uses machine learning algorithms,where the result from each graph of the project can be taken as individual criteria for the machine learning algorithm can be created also,A risk score can be generated based on applicant to predict loan default rate

APPENDIX

A.Source code

```python
#Basic and most important libraries
import pandas as pd , numpy as np
from sklearn.utils import resample
from sklearn.preprocessing import StandardScaler , MinMaxScaler
from collections import Counter
from scipy import stats
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.figure_factory as ff
import plotly

#Classifiers
from sklearn.ensemble import AdaBoostClassifier , GradientBoostingClassifier , VotingClassifier , RandomForestClassifier
from sklearn.linear_model import LogisticRegression , RidgeClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from xgboost import plot_importance
from xgboost import XGBClassifier
from sklearn.svm import SVC

#Model evaluation tools
from sklearn.metrics import classification_report , accuracy_score , confusion_matrix
from sklearn.metrics import accuracy_score,f1_score
from sklearn.model_selection import cross_val_score

#Data processing functions
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn import model_selection
from sklearn.preprocessing import LabelEncoder
```

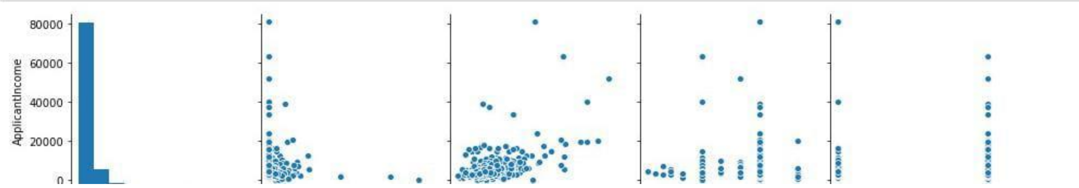
```
#Data processing functions
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn import model_selection
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()

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

In [2]:
```
data = pd.read_csv(r"C:\Master\Learning\Analytics_Vidhya\Loan_Prediction-Hackathon\train.csv")
data.head(5)
```

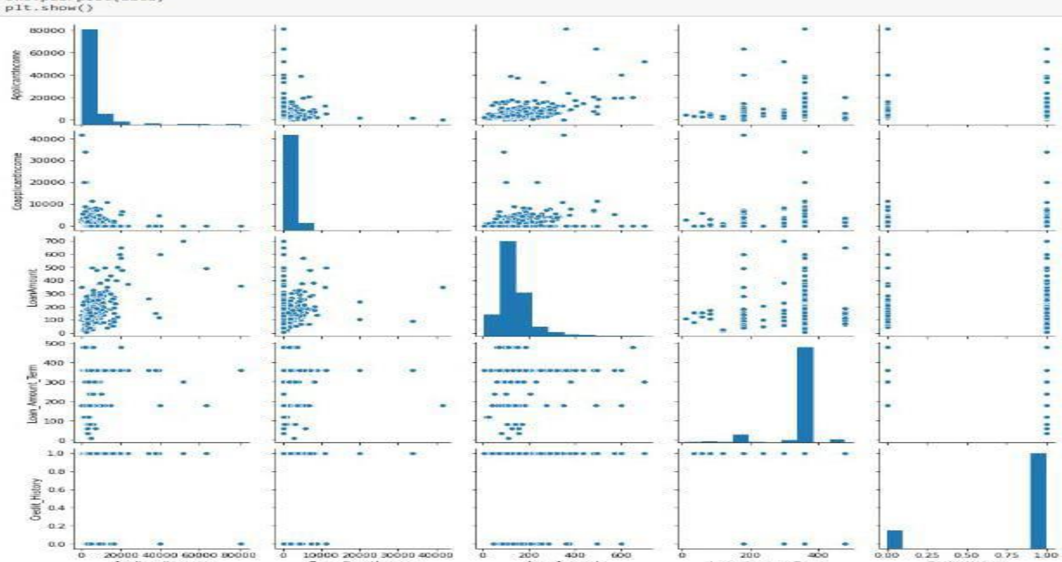
Out[2]:

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

In [3]:
```
sns.pairplot(data)
plt.show()
```



In [3]:
```
sns.pairplot(data)
plt.show()
```



In [4]: `data.describe()`

Out[4]:

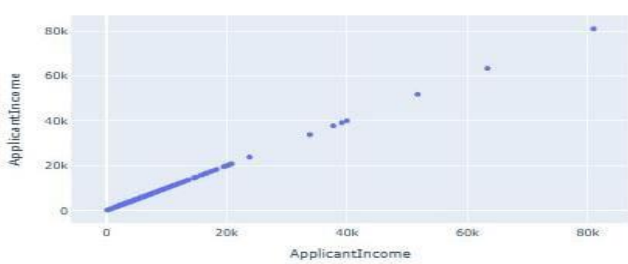
	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	614.000000	614.000000	592.000000	600.00000	564.000000
mean	5403.459283	1621.245798	146.412162	342.00000	0.842199
std	6109.041673	2926.248369	85.587325	65.12041	0.364878
min	150.000000	0.000000	9.000000	12.00000	0.000000
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 [5]: `data.info()`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Loan_ID            614 non-null    object
 1   Gender             601 non-null    object
 2   Married            611 non-null    object
 3   Dependents         599 non-null    object
 4   Education          614 non-null    object
 5   Self_Employed      582 non-null    object
 6   ApplicantIncome    614 non-null    int64
 7   CoapplicantIncome  614 non-null    float64
 8   LoanAmount         592 non-null    float64
 9   Loan_Amount_Term   600 non-null    float64
 10  Credit_History     564 non-null    float64
 11  Property_Area      614 non-null    object
 12  Loan_Status        614 non-null    object
dtypes: float64(4), int64(1), object(8)
memory usage: 62.5+ KB
```

In [6]:

```
fig = px.scatter_matrix(data["ApplicantIncome"])
fig.update_layout(width=700,height=400)
fig.show()
```



Seems need to work on data preperation

-Loan Amount column does is not fit in Normal Distribution

-Outliers in Applicant's Income and Co-applicant's income

In [7]: `data.isnull().sum()`

Out[7]:

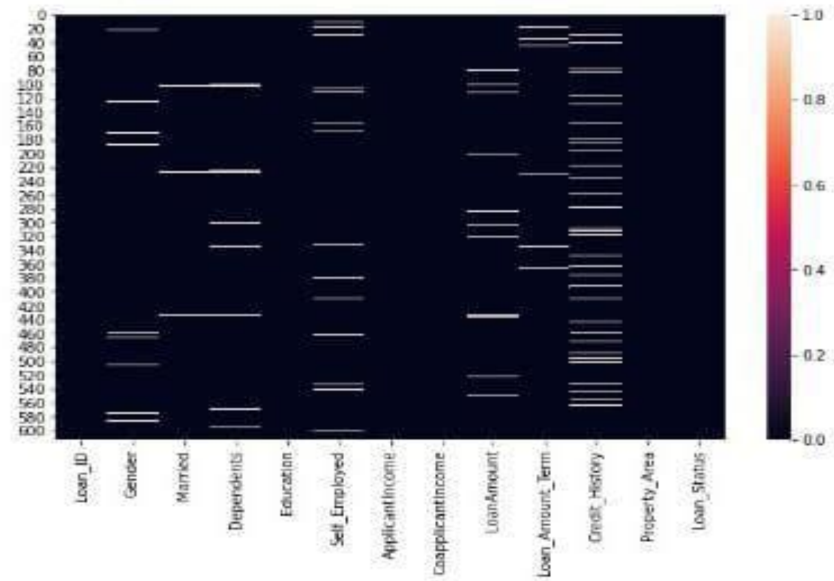
```
Loan_ID               0
Gender               13
Married               3
Dependents           15
Education             0
Self_Employed        32
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           22
Loan_Amount_Term     14
Credit_History       50
Property_Area         0
Loan_Status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

In [8]:

```
plt.figure(figsize=(10,6))
sns.heatmap(data.isnull())
```

Out[8]: <AxesSubplot:>

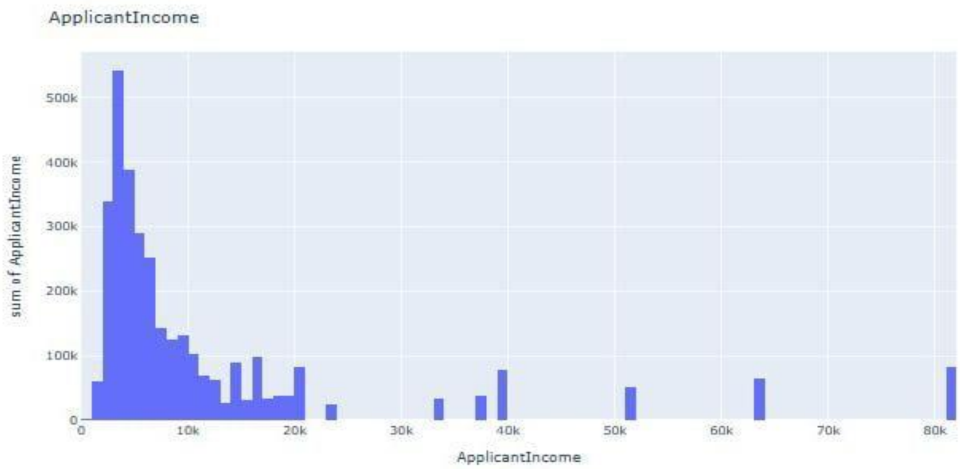
Out[8]: <AxesSubplot:>

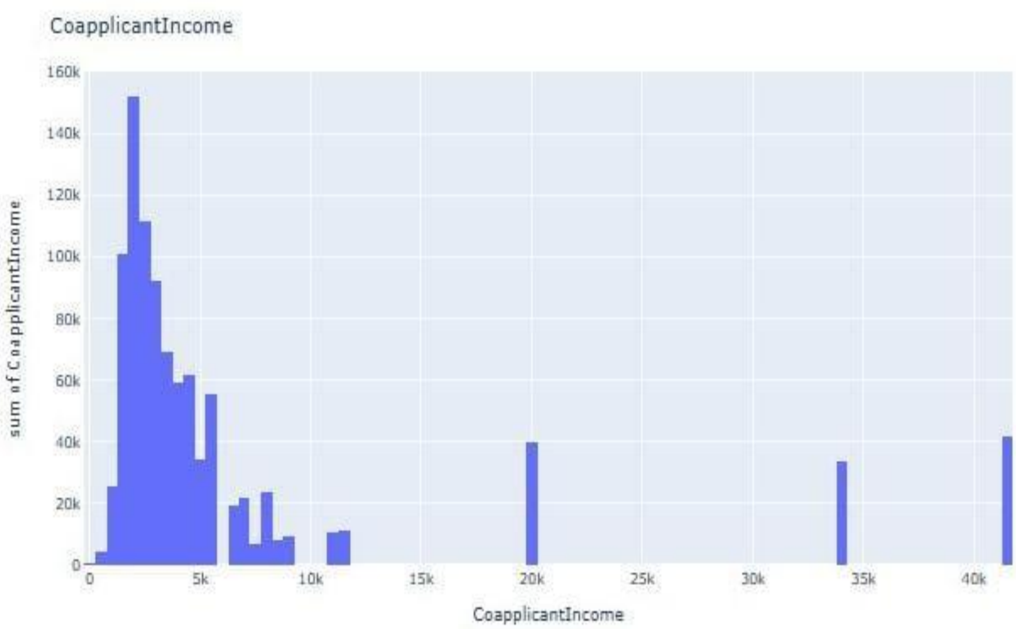


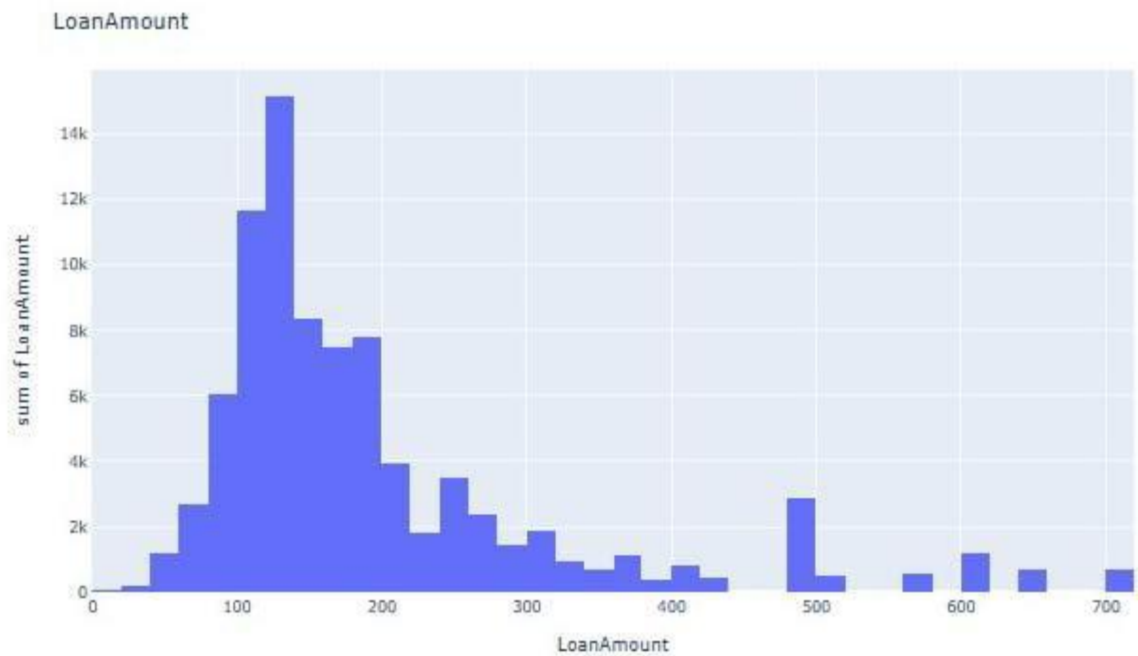
In [9]:
```python
#Checking if the non-categorical variables are Normally Distributed or Not. i.e. Checking outliers...

print("Data distribution analysis:->---------------------------------------\n")
print("\nMean:->\n")
print("ApplicantIncome: ",np.mean(data["ApplicantIncome"]))
print("CoapplicantIncome: ",np.mean(data["CoapplicantIncome"]))
print("LoanAmount: ",np.mean(data["LoanAmount"]))

print("\nMode:->\n")
print("ApplicantIncome: ",stats.mode(data["ApplicantIncome"])[0])
print("CoapplicantIncome: ",stats.mode(data["CoapplicantIncome"])[0])
print("LoanAmount: ",stats.mode(data["LoanAmount"])[0])

print("\nMedian:->\n")
print("ApplicantIncome: ",np.median(data["ApplicantIncome"]))
print("CoapplicantIncome: ",np.median(data["CoapplicantIncome"]))
print("LoanAmount: ",np.median(data["LoanAmount"]))

print("\nStandard Deviation:->\n")
print("ApplicantIncome: ",np.std(data["ApplicantIncome"]))
print("CoapplicantIncome: ",np.std(data["CoapplicantIncome"]))
print("LoanAmount: ",np.std(data["LoanAmount"]))

fig = px.histogram(data["ApplicantIncome"],x ="ApplicantIncome" ,y = "ApplicantIncome" )
fig.update_layout(title="ApplicantIncome")
fig.show()

fig = px.histogram(data["CoapplicantIncome"],x ="CoapplicantIncome" ,y = "CoapplicantIncome" )
fig.update_layout(title="CoapplicantIncome")
fig.show()

fig = px.histogram(data["LoanAmount"],x ="LoanAmount" ,y = "LoanAmount" )
fig.update_layout(title="LoanAmount")
fig.show()
```

```
Data distribution analysis:->---------------------------------------


Mean:->

ApplicantIncome:  5403.459283387622
CoapplicantIncome:  1621.245798027101
LoanAmount:  146.41216216216216

Mode:->

ApplicantIncome:  [2500]
CoapplicantIncome:  [0.]
LoanAmount:  [120.]
```

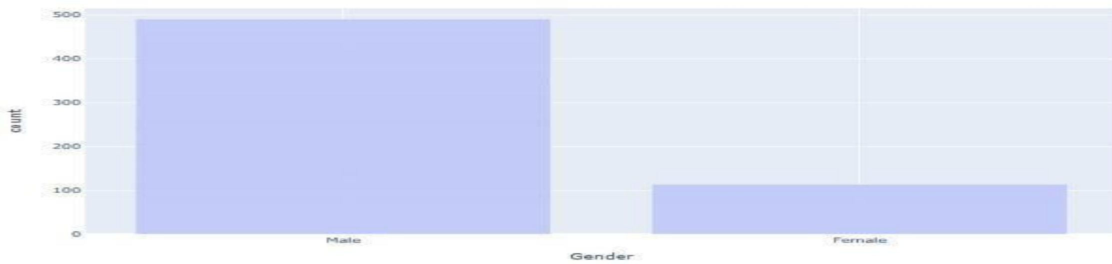
```
Mode:->

ApplicantIncome:  [2500]
CoapplicantIncome:  [0.]
LoanAmount:  [120.]

Median:->

ApplicantIncome:  3812.5
CoapplicantIncome:  1188.5
LoanAmount:  nan

Standard Deviation:->

ApplicantIncome:  6104.064856533888
CoapplicantIncome:  2923.8644597700627
LoanAmount:  85.51500809120331
```

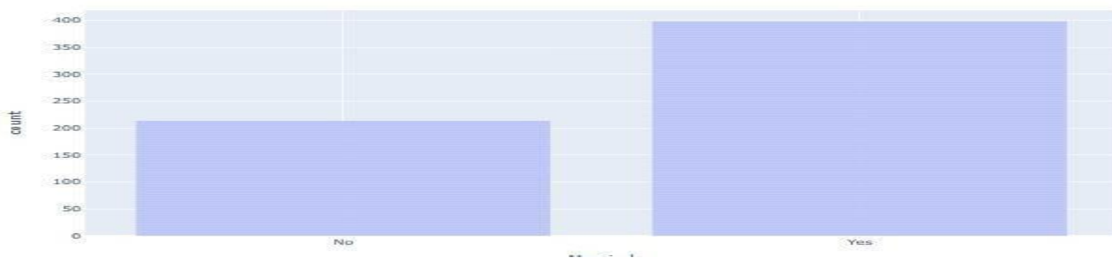


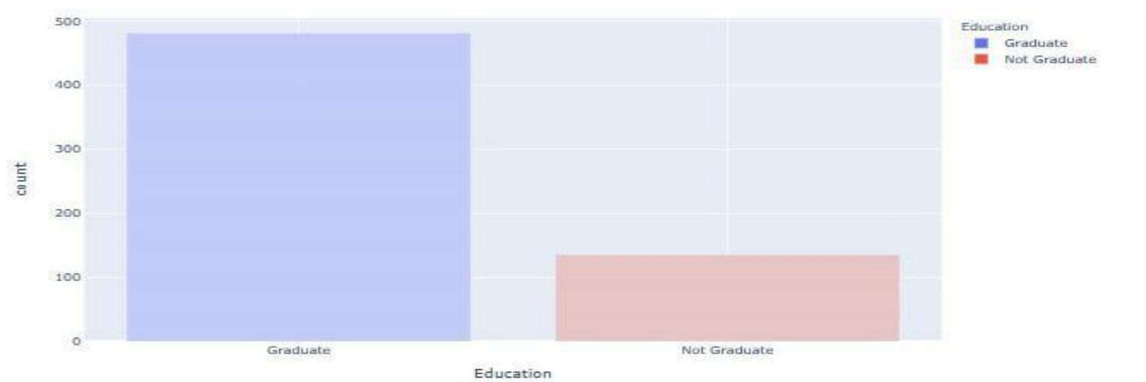


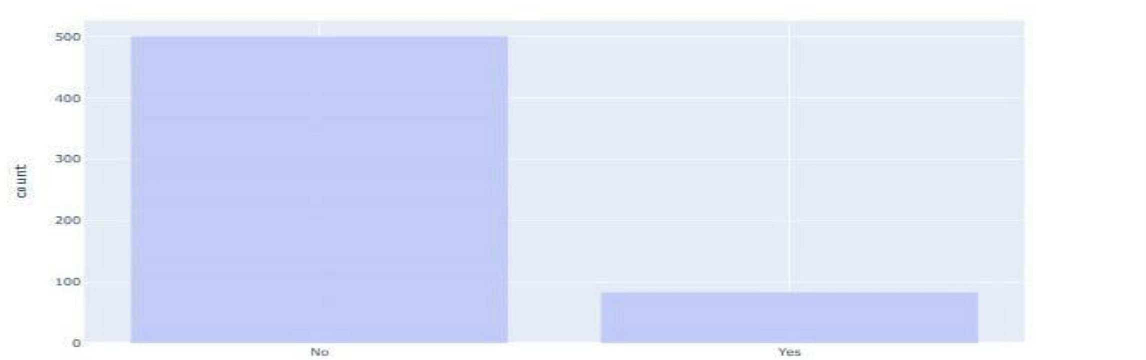


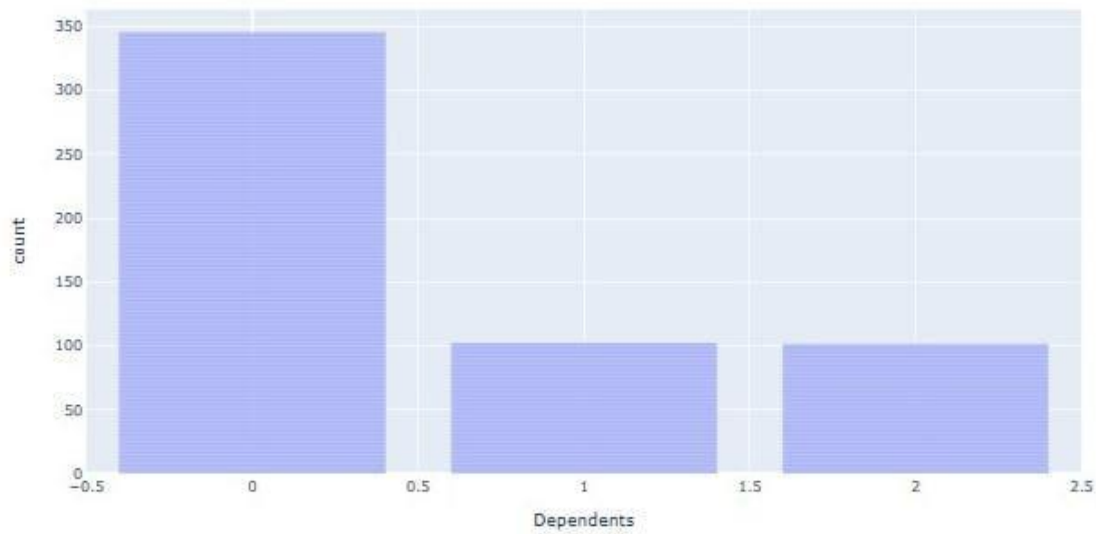
In [10]:
```
plt.figure(figsize=(10,5))
fig = px.bar(data,x=data["Gender"])
fig.show()

fig = px.bar(data,x=data["Married"])
fig.show()

fig = px.bar(data,x=data["Education"],color="Education")
fig.show()

fig = px.bar(data,x=data["Self_Employed"])
fig.show()

fig = px.bar(data,x=data["Dependents"])
fig.show()

fig = px.bar(data,x=data["Property_Area"])
fig.show()

fig = px.bar(data,x=data["Loan_Status"],color="Loan_Status")
fig.show()
```


count
500
400
300
200
100
0
Male
Female
Gender


count
400
350
300
250
200
150
100
50
0
No
Yes


Education
Graduate
Not Graduate
count
500
400
300
200
100
0
Graduate
Not Graduate
Education


count
500
400
300
200
100
0
No
Yes



count
Dependents
350
300
250
200
150
100
50
0
−0.5
0
0.5
1
1.5
2
2.5

count
Property_Area
Urban
Rural
Semiurban
200
150
100
50
0

count
Loan_Status
Y
N
400
350
300
250
200
150
100
50
0

In [11]:
```
print(data["Gender"].value_counts())
print(data["Married"].value_counts())
print(data["Self_Employed"].value_counts())
print(data["Dependents"].value_counts())
print(data["Credit_History"].value_counts())
print(data["Loan_Amount_Term"].value_counts())
```

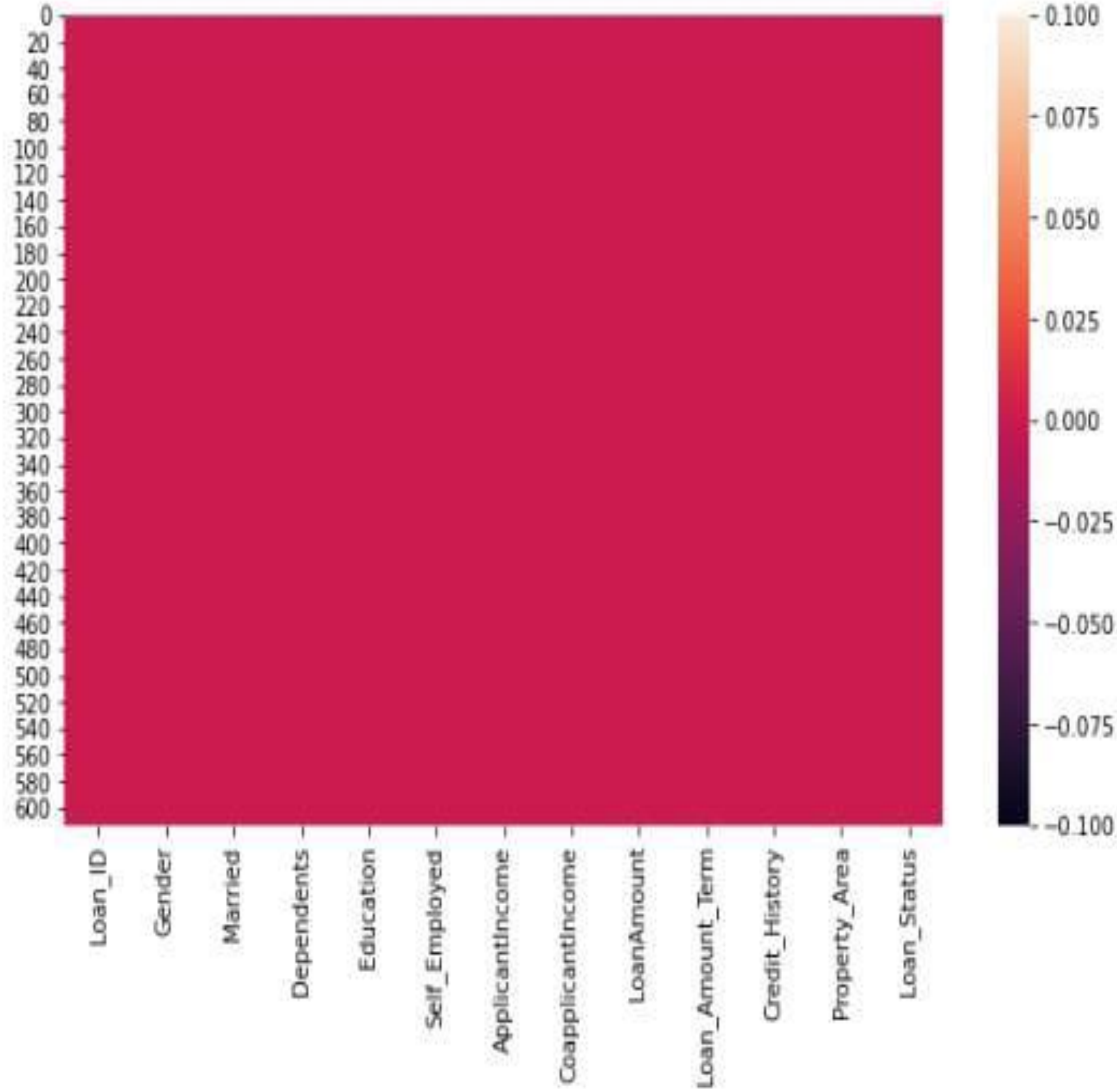
```
Male      489
Female    112
Name: Gender, dtype: int64
Yes    398
No     213
Name: Married, dtype: int64
No     500
Yes     82
Name: Self_Employed, dtype: int64
0     345
1     102
2     101
3+     51
Name: Dependents, dtype: int64
1.0    475
0.0     89
Name: Credit_History, dtype: int64
360.0    512
180.0     44
480.0     15
300.0     13
84.0       4
240.0      4
120.0      3
36.0       2
60.0       2
12.0       1
Name: Loan_Amount_Term, dtype: int64
```

->Taking mode of values in a column will be best way to fill null values. ->Not mean because values are not ordinal but are categorical.

In [12]:
```
#Filling all Nan values with mode of respective variable
data["Gender"].fillna(data["Gender"].mode()[0],inplace=True)
data["Married"].fillna(data["Married"].mode()[0],inplace=True)
data["Self_Employed"].fillna(data["Self_Employed"].mode()[0],inplace=Tru
e)
data["Loan_Amount_Term"].fillna(data["Loan_Amount_Term"].mode()[0],inpla
ce=True)
data["Dependents"].fillna(data["Dependents"].mode()[0],inplace=True)
data["Credit_History"].fillna(data["Credit_History"].mode()[0],inplace=T
rue)

#All values of "Dependents" columns were of "str" form now converting to
"int" form.
data["Dependents"] = data["Dependents"].replace('3+',int(3))
data["Dependents"] = data["Dependents"].replace('1',int(1))
data["Dependents"] = data["Dependents"].replace('2',int(2))
data["Dependents"] = data["Dependents"].replace('0',int(0))

data["LoanAmount"].fillna(data["LoanAmount"].median(),inplace=True)

print(data.isnull().sum())

#Heat map for null values
plt.figure(figsize=(10,6))
sns.heatmap(data.isnull())
```

```
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
```

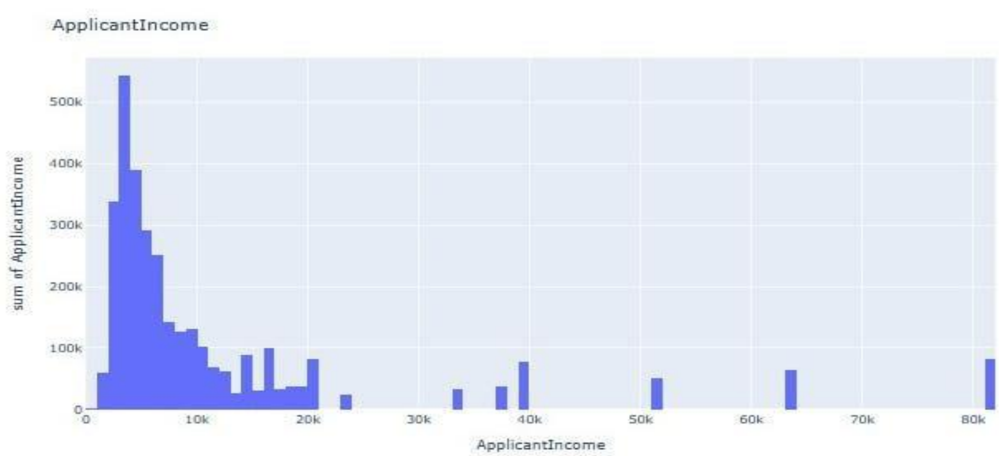
Out[12]: <AxesSubplot:>

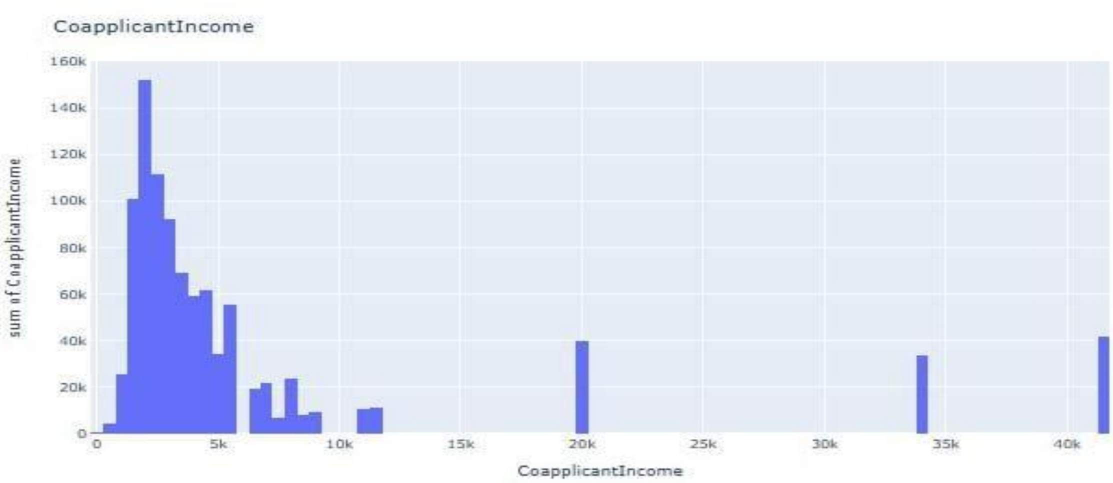


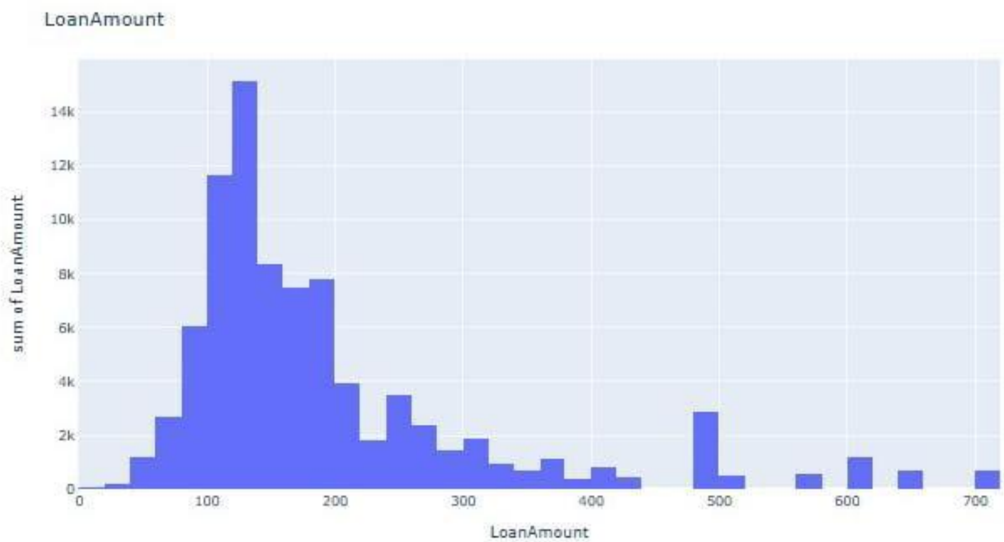
0
20
40
60
80
100
120
140
160
180
200
220
240
260
280
300
320
340
360
380
400
420
440
460
480
500
520
540
560
580
600
Loan_ID
Gender
Married
Dependents
Education
Self_Employed
ApplicantIncome
CoapplicantIncome
LoanAmount
Loan_Amount_Term
Credit_History
Property_Area
Loan_Status
0.100
0.075
0.050
0.025
0.000
−0.025
−0.050
−0.075
−0.100

```
#Treating outliers and Converting data to Normal Distribution
#Before removing outlier

print("\nMean:->\n")
print("ApplicantIncome: ",np.mean(data["ApplicantIncome"]))
print("CoapplicantIncome: ",np.mean(data["CoapplicantIncome"]))
print("LoanAmount: ",np.mean(data["LoanAmount"]))

print("\nMode:->\n")
print("ApplicantIncome: ",stats.mode(data["ApplicantIncome"])[0])
print("CoapplicantIncome: ",stats.mode(data["CoapplicantIncome"])[0])
print("LoanAmount: ",stats.mode(data["LoanAmount"])[0])

print("\nMedian:->\n")
print("ApplicantIncome: ",np.median(data["ApplicantIncome"]))
print("CoapplicantIncome: ",np.median(data["CoapplicantIncome"]))
print("LoanAmount: ",np.median(data["LoanAmount"]))

print("\nStandard Deviation:->\n")
print("ApplicantIncome: ",np.std(data["ApplicantIncome"]))
print("CoapplicantIncome: ",np.std(data["CoapplicantIncome"]))
print("LoanAmount: ",np.std(data["LoanAmount"]))

fig = px.histogram(data["ApplicantIncome"],x ="ApplicantIncome" ,y = "Ap
plicantIncome" )
fig.update_layout(title="ApplicantIncome")
fig.show()

fig = px.histogram(data["CoapplicantIncome"],x ="CoapplicantIncome" ,y =
"CoapplicantIncome" )
fig.update_layout(title="CoapplicantIncome")
fig.show()

fig = px.histogram(data["LoanAmount"],x ="LoanAmount" ,y = "LoanAmount"
)
fig.update_layout(title="LoanAmount")
fig.show()

###########################################################################
##############################
#Getting log value :->

data["ApplicantIncome"] = np.log(data["ApplicantIncome"])
#As "CoapplicantIncome" columns has some "0" values we will get log valu
es except "0"
data["CoapplicantIncome"] = [np.log(i) if i!=0 else 0 for i in data["Coa
pplicantIncome"]]
data["LoanAmount"] = np.log(data["LoanAmount"])
###########################################################################
##############################

print("---------------------------After converting to Normal Distributed
data----------------------")

print("\nMean:->\n")
print("ApplicantIncome: ",np.mean(data["ApplicantIncome"]))
print("CoapplicantIncome: ",np.mean(data["CoapplicantIncome"]))
print("LoanAmount: ",np.mean(data["LoanAmount"]))

print("\nMode:->\n")
print("ApplicantIncome: ",stats.mode(data["ApplicantIncome"])[0])
print("CoapplicantIncome: ",stats.mode(data["CoapplicantIncome"])[0])
print("LoanAmount: ",stats.mode(data["LoanAmount"])[0])

print("\nMedian:->\n")
print("ApplicantIncome: ",np.median(data["ApplicantIncome"]))
print("CoapplicantIncome: ",np.median(data["CoapplicantIncome"]))
print("LoanAmount: ",np.median(data["LoanAmount"]))

print("\nStandard Deviation:->\n")
print("ApplicantIncome: ",np.std(data["ApplicantIncome"]))
print("CoapplicantIncome: ",np.std(data["CoapplicantIncome"]))
print("LoanAmount: ",np.std(data["LoanAmount"]))

plt.figure(figsize=(10,4))
fig = px.histogram(data["ApplicantIncome"],x ="ApplicantIncome" ,y = "Ap
plicantIncome" )
fig.update_layout(title="ApplicantIncome")
fig.show()

fig = px.histogram(data["CoapplicantIncome"],x ="CoapplicantIncome" ,y =
"CoapplicantIncome" )
fig.update_layout(title="CoapplicantIncome")
fig.show()

fig = px.histogram(data["LoanAmount"],x ="LoanAmount" ,y = "LoanAmount"
)
fig.update_layout(title="LoanAmount")
fig.show()
```

```
Mean:->

ApplicantIncome:  5403.459283387622
CoapplicantIncome:  1621.245798027101
LoanAmount:  145.75244299674267

Mode:->

ApplicantIncome:  [2500]
CoapplicantIncome:  [0.]
LoanAmount:  [128.]

Median:->

ApplicantIncome:  3812.5
CoapplicantIncome:  1188.5
LoanAmount:  128.0

Standard Deviation:->

ApplicantIncome:  6104.064856533888
CoapplicantIncome:  2923.8644597700627
LoanAmount:  84.03871423798938
```







In [14]:
```
data.head(5)
```

Out[14]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantInc
0	LP001002	Male	No	0	Graduate	No	8.674026	0.000
1	LP001003	Male	Yes	1	Graduate	No	8.430109	7.318
2	LP001005	Male	Yes	0	Graduate	Yes	8.006368	0.000
3	LP001006	Male	Yes	0	Not Graduate	No	7.856707	7.765
4	LP001008	Male	No	0	Graduate	No	8.699515	0.000

In [15]:
```
data["Gender"] = le.fit_transform(data["Gender"])
data["Married"] = le.fit_transform(data["Married"])
data["Education"] = le.fit_transform(data["Education"])
data["Self_Employed"] = le.fit_transform(data["Self_Employed"])
data["Property_Area"] = le.fit_transform(data["Property_Area"])
data["Loan_Status"] = le.fit_transform(data["Loan_Status"])

#data = pd.get_dummies(data)
data.head(5)
```

Out[15]:

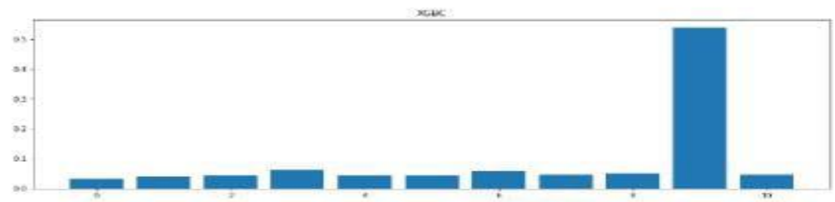
	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantInc
0	LP001002	1	0	0	0	0	8.674026	0.000
1	LP001003	1	1	1	0	0	8.430109	7.318
2	LP001005	1	1	0	0	1	8.006368	0.000
3	LP001006	1	1	0	1	0	7.856707	7.765
4	LP001008	1	0	0	0	0	8.699515	0.000

In [16]:
```
#Dividing data into Input X variables and Target Y variable
X = data.drop(["Loan_Status","Loan_ID"],axis=1)
y = data["Loan_Status"]
```

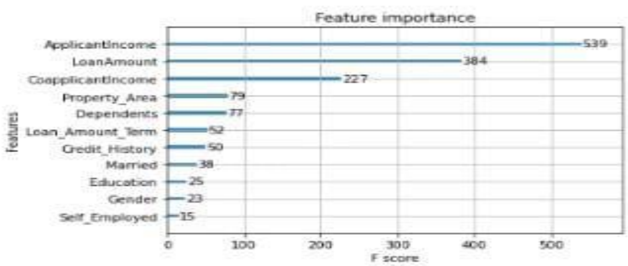
In [17]:
```
print("Feature importance by XGBoost:->\n")
XGBR = XGBClassifier()
XGBR.fit(X,y)
features = XGBR.feature_importances_
Columns = list(X.columns)
for i,j in enumerate(features):
    print(Columns[i],"->",j)
plt.figure(figsize=(16,5))
plt.title(label="XGBC")
plt.bar([x for x in range(len(features))],features)
plt.show()

plot_importance(XGBR)

print("Feature importance by Random Forest:->\n")
RF = RandomForestClassifier()
RF.fit(X,y)
features = RF.feature_importances_
Columns = list(X.columns)
for i,j in enumerate(features):
    print(Columns[i],"->",j)
plt.figure(figsize=(16,5))
plt.title(label="RF")
plt.bar([x for x in range(len(features))],features)
plt.show()

print("Feature importance by Decision Tree:->\n")
DT = DecisionTreeClassifier()
DT.fit(X,y)
features = DT.feature_importances_
Columns = list(X.columns)
for i,j in enumerate(features):
    print(Columns[i],"->",j)
plt.figure(figsize=(16,5))
plt.title(label="DT")
plt.bar([x for x in range(len(features))],features)
plt.show()

print("Feature importance by Suppoprt Vector Machine:->\n")
SVM = SVC(kernel="linear")
SVM.fit(X,y)
features = SVM.coef_[0]
Columns = list(X.columns)
for i,j in enumerate(features):
    print(Columns[i],"->",j)
plt.figure(figsize=(16,5))
plt.bar([x for x in range(len(features))],features)
plt.show()

print("Feature importance by Logistic Regression:->\n")
LOGC = LogisticRegression()
LOGC.fit(X,y)
features = LOGC.coef_[0]
Columns = list(X.columns)
for i,j in enumerate(features):
    print(Columns[i],"->",j)
plt.figure(figsize=(16,5))
plt.title(label="LOGC")
plt.bar([x for x in range(len(features))],features)
plt.show()
```

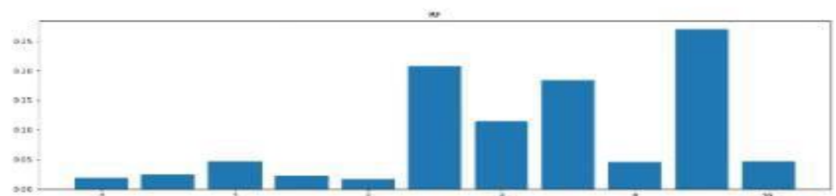
```
Feature importance by XGBoost:->
```

```
Gender -> 0.032498196
Married -> 0.038461618
Dependents -> 0.042435512
Education -> 0.06297734
Self_Employed -> 0.04353367
ApplicantIncome -> 0.04360314
CoapplicantIncome -> 0.057352304
LoanAmount -> 0.04579362
Loan_Amount_Term -> 0.049817037
Credit_History -> 0.53902644
Property_Area -> 0.044501156
```

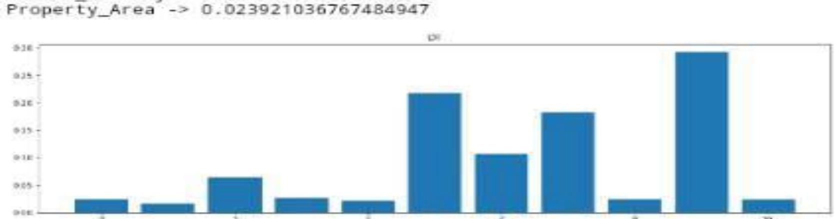


```
Feature importance by Random Forest:->

Gender -> 0.018900487204807852
Married -> 0.02462931105690738
Dependents -> 0.047052466747390456
Education -> 0.022639340128868403
Self_Employed -> 0.01657632874630836
ApplicantIncome -> 0.20779381878351624
CoapplicantIncome -> 0.1145844478048256
LoanAmount -> 0.1839704267368444
Loan_Amount_Term -> 0.046140336171376174
Credit_History -> 0.27057938232022927
Property_Area -> 0.047133654298925964
```

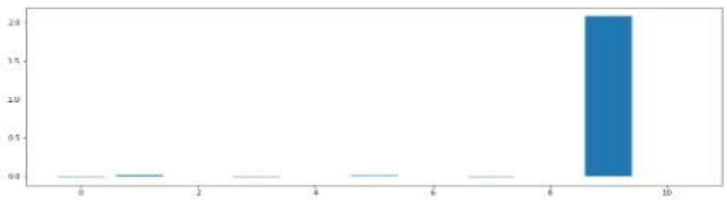




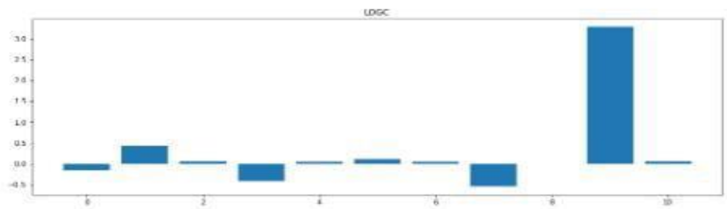
```
Feature importance by Decision Tree:->

Gender -> 0.024418005090398452
Married -> 0.016326588769442065
Dependents -> 0.06400168357447024
Education -> 0.026528466142634998
Self_Employed -> 0.020983027101584565
ApplicantIncome -> 0.21759903699076355
CoapplicantIncome -> 0.10665253394698983
LoanAmount -> 0.18273808075459913
Loan_Amount_Term -> 0.024630673969621018
Credit_History -> 0.2922008668920113
Property_Area -> 0.023921036767484947
```



```
Feature importance by Suppoprt Vector Machine:->

Gender -> -0.011153748611395287
Married -> 0.016433621802949716
Dependents -> -0.0003948864299205823
Education -> -0.007897250281862611
Self_Employed -> -0.0045186612877454735
ApplicantIncome -> 0.009509713938893327
CoapplicantIncome -> 0.0009391121595605512
LoanAmount -> -0.012713675348784648
Loan_Amount_Term -> 8.910680668350324e-05
Credit_History -> 2.0812104159306477
Property_Area -> -0.0006557085562250223
```



```
Feature importance by Logistic Regression:->

Gender -> -0.1615139600532564
Married -> 0.4341098090301747
Dependents -> 0.05871757548193793
Education -> -0.415446117064946
Self_Employed -> 0.04313150698288537
ApplicantIncome -> 0.1020827246750018
CoapplicantIncome -> 0.04475513414904771
LoanAmount -> -0.5526893355733061
Loan_Amount_Term -> -0.0012174092655106736
Credit_History -> 3.28383317084153
Property_Area -> 0.05809243644023144
```



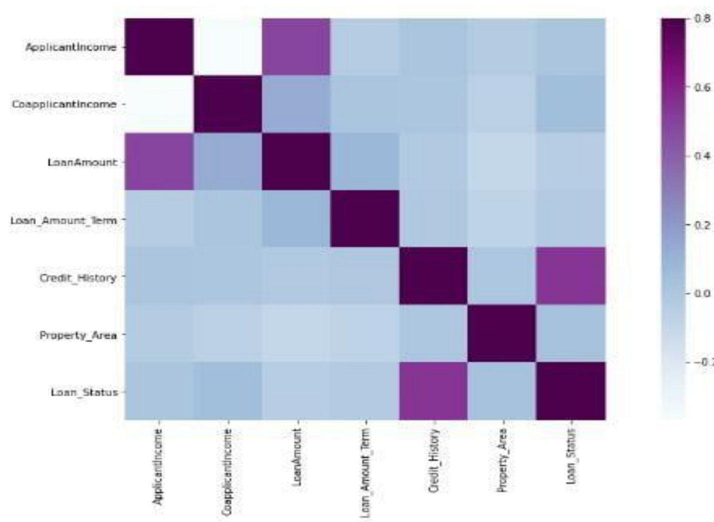
From feature importance => Credit History , ApplicantIncome , CoapplicantIncome, LoanAmount are the most important features

Is data Balanced ?

In [18]:

```python
#Heat map of dataset with relative importance
matrix = data.drop(["Gender","Married","Dependents","Education","Self_Em
ployed"],axis=1).corr()
#f , ax = plt.subplots(figsize=(18,6))
plt.figure(figsize=(18,8))
sns.heatmap(matrix,vmax=0.8,square=True,cmap="BuPu")
```

Out[18]: <AxesSubplot:>

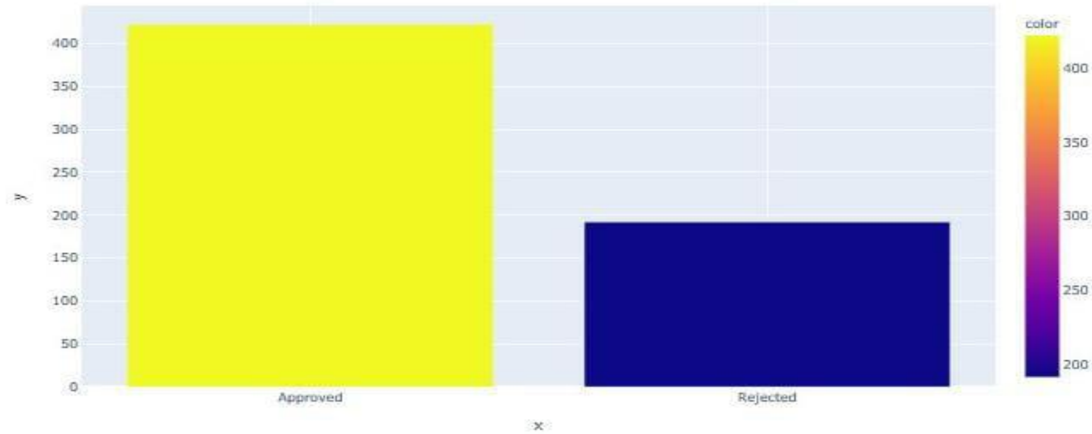


It seems Application income and Loan Amount is correlated , also Coapplication income correlated with Loan Aount then Credit history is corrleated with Loan Status

In [19]:

```python
A = list(data.Loan_Status).count(1)
B = list(data.Loan_Status).count(0)
print("Count of 1<Approved>: ",A,"\nCount of 0<Rejected>: ",B)

fig = px.bar((A,B),x=["Approved","Rejected"],y=[A,B],color=[A,B])
fig.show()
```

```
Count of 1<Approved>:  422
Count of 0<Rejected>:  192
```



In [20]:
```
#To keep original data as it is to use the same for later.
new_data = data.copy()

#Getting seperated data with 1 and 0 status.
df_majority = new_data[new_data.Loan_Status==1]
df_minority = new_data[new_data.Loan_Status==0]

#Here we are downsampling the Majority Class Data Points.
#i.e. We will get equal amount of datapoint as Minority class from Major
ity class

df_manjority_downsampled = resample(df_majority,replace=False,n_samples=
192,random_state=123)
df_downsampled = pd.concat([df_manjority_downsampled,df_minority])
print("Downsampled data:->\n",df_downsampled.Loan_Status.value_counts())

#Here we are upsampling the Minority Class Data Points.
#i.e. We will get equal amount of datapoint as Majority class from Minor
ity class
df_monority_upsampled = resample(df_minority,replace=True,n_samples=422,
random_state=123)
df_upsampled = pd.concat([df_majority,df_monority_upsampled])
print("Upsampled data:->\n",df_upsampled.Loan_Status.value_counts())
```

```
Downsampled data:->
 1    192
0    192
Name: Loan_Status, dtype: int64
Upsampled data:->
 1    422
0    422
Name: Loan_Status, dtype: int64
```

In [21]:

```
#Experiment 1: Only Scaled data with all variables

#X = new_data.drop(["Loan_ID","Gender","Married","Education","Self_Emplo
yed","Loan_Amount_Term","Loan_Status",'Property_Area'],axis=1)
X = new_data.drop(["Loan_Status","Loan_ID"],axis=1)
y = new_data["Loan_Status"]
counter = Counter(y)
print("Counter: ",counter)

X_train , X_test , y_train , y_test = train_test_split(X,y,test_size=0.2
5,random_state=0)

#Scaling data here:------------->

StSc = StandardScaler()
X_train  = StSc.fit_transform(X_train)
X_test   = StSc.fit_transform(X_test)

#Check mean is 0 and Standard deviation is 1
print("After Standardization\nMean ",np.mean(X_train),"Standard Deviatio
n ",np.std(X_train),"\n")

#Voting ensemble mathod. Combining all tree based algorithms.
models = []
models.append(("XGB",XGBClassifier()))
models.append(("RF",RandomForestClassifier()))
models.append(("DT",DecisionTreeClassifier()))
models.append(("ADB",AdaBoostClassifier()))
models.append(("GB",GradientBoostingClassifier()))

ensemble = VotingClassifier(estimators=models)
ensemble.fit(X_train,y_train)
y_pred = ensemble.predict(X_test)
print(classification_report(y_pred,y_test))
print("Voting Ensemble:>",accuracy_score(y_pred,y_test))


SVM = SVC(kernel="linear",class_weight="balanced",probability=True)
SVM.fit(X_train,y_train)
y_pred = SVM.predict(X_test)
print(classification_report(y_pred,y_test))
print("SVM:>",accuracy_score(y_pred,y_test))


XGBC = XGBClassifier(learning_rate =0.1,n_estimators=10000,max_depth=4,m
in_child_weight=6,gamma=0,subsample=0.6,colsample_bytree=0.8,
 reg_alpha=0.005, objective= 'binary:logistic', nthread=2, scale_pos_wei
ght=1, seed=27)
XGBC.fit(X_train,y_train)
y_pred = XGBC.predict(X_test)
print(classification_report(y_pred,y_test))
print("XGBoost:>",accuracy_score(y_pred,y_test))

Model1 = RandomForestClassifier(n_estimators=1000,random_state=0,n_jobs=
1000,max_depth=70,bootstrap=True)
Model1.fit(X_train,y_train)
y_pred = Model1.predict(X_test)
print(classification_report(y_pred,y_test))
print("RandomForestClassifier:>",accuracy_score(y_pred,y_test))


Model2 = GradientBoostingClassifier()
Model2.fit(X_train,y_train)
y_pred = Model2.predict(X_test)
print(classification_report(y_pred,y_test))
print("GradientBoostingClassifier:>",accuracy_score(y_pred,y_test))


Model3 = DecisionTreeClassifier(class_weight=None, criterion='gini', max
_depth=100,
 max_features=1.0, max_leaf_nodes=10,
 min_impurity_split=1e-07, min_samples_leaf=1,
 min_samples_split=2, min_weight_fraction_leaf=0.10,
 presort=False, random_state=27, splitter='best')
Model3.fit(X_train,y_train)
y_pred = Model3.predict(X_test)
print(classification_report(y_pred,y_test))
print("DecisionTreeClassifier:>",accuracy_score(y_pred,y_test))


Model4 = AdaBoostClassifier()
Model4.fit(X_train,y_train)
y_pred = Model4.predict(X_test)
print(classification_report(y_pred,y_test))
print("AdaBoostClassifier:>",accuracy_score(y_pred,y_test))


Model5 = LinearDiscriminantAnalysis()
Model5.fit(X_train,y_train)
y_pred = Model5.predict(X_test)
print(classification_report(y_pred,y_test))
print("LinearDiscriminantAnalysis:>",accuracy_score(y_pred,y_test),"\n")


KNN = KNeighborsClassifier(leaf_size=1,p=2,n_neighbors=20)
KNN.fit(X_train,y_train)
y_pred = KNN.predict(X_test)
print(classification_report(y_pred,y_test))
print("KNeighborsClassifier:>",accuracy_score(y_pred,y_test))


Model7 = GaussianNB()
Model7.fit(X_train,y_train)
y_pred = Model7.predict(X_test)
print(classification_report(y_pred,y_test))
print("GaussianNB:>",accuracy_score(y_pred,y_test))


Model8 = LogisticRegression(C=1.0, class_weight=None, dual=False, fit_in
tercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=
1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.000
1,
          verbose=0, warm_start=False)
Model8.fit(X_train,y_train)
y_pred = Model8.predict(X_test)
print(classification_report(y_pred,y_test))
print("Logistic Regression:>",accuracy_score(y_pred,y_test))
```

```
Counter:  Counter({1: 422, 0: 192})
After Standardization
Mean  -1.2357264969740873e-16 Standard Deviation  1.0
              precision    recall  f1-score   support

           0       0.47      0.74      0.57        27
           1       0.94      0.82      0.87       127

    accuracy                           0.81       154
   macro avg       0.70      0.78      0.72       154
weighted avg       0.85      0.81      0.82       154

Voting Ensemble:> 0.8051948051948052
              precision    recall  f1-score   support

           0       0.44      0.90      0.59        21
           1       0.98      0.82      0.89       133

    accuracy                           0.83       154
   macro avg       0.71      0.86      0.74       154
weighted avg       0.91      0.83      0.85       154

SVM:> 0.8311688311688312
              precision    recall  f1-score   support

           0       0.30      0.45      0.36        29
           1       0.86      0.76      0.81       125

    accuracy                           0.70       154
   macro avg       0.58      0.60      0.58       154
weighted avg       0.75      0.70      0.72       154

XGBoost:> 0.7012987012987013
              precision    recall  f1-score   support

           0       0.44      0.73      0.55        26
           1       0.94      0.81      0.87       128

    accuracy                           0.80       154
   macro avg       0.69      0.77      0.71       154
weighted avg       0.85      0.80      0.82       154

RandomForestClassifier:> 0.7987012987012987
              precision    recall  f1-score   support

           0       0.47      0.80      0.59        25
           1       0.95      0.82      0.88       129

    accuracy                           0.82       154
   macro avg       0.71      0.81      0.74       154
weighted avg       0.88      0.82      0.84       154

GradientBoostingClassifier:> 0.8181818181818182
              precision    recall  f1-score   support

           0       0.44      0.90      0.59        21
           1       0.98      0.82      0.89       133

    accuracy                           0.83       154
   macro avg       0.71      0.86      0.74       154
weighted avg       0.91      0.83      0.85       154

DecisionTreeClassifier:> 0.8311688311688312
              precision    recall  f1-score   support

           0       0.49      0.81      0.61        26
           1       0.95      0.83      0.89       128

    accuracy                           0.82       154
   macro avg       0.72      0.82      0.75       154
weighted avg       0.88      0.82      0.84       154

AdaBoostClassifier:> 0.8246753246753247
              precision    recall  f1-score   support

           0       0.44      0.90      0.59        21
           1       0.98      0.82      0.89       133

    accuracy                           0.83       154
   macro avg       0.71      0.86      0.74       154
weighted avg       0.91      0.83      0.85       154

LinearDiscriminantAnalysis:> 0.8311688311688312

              precision    recall  f1-score   support

           0       0.47      0.91      0.62        22
           1       0.98      0.83      0.90       132

    accuracy                           0.84       154
   macro avg       0.72      0.87      0.76       154
weighted avg       0.91      0.84      0.86       154

KNeighborsClassifier:> 0.8376623376623377
              precision    recall  f1-score   support

           0       0.44      0.90      0.59        21
           1       0.98      0.82      0.89       133

    accuracy                           0.83       154
   macro avg       0.71      0.86      0.74       154
weighted avg       0.91      0.83      0.85       154

GaussianNB:> 0.8311688311688312
              precision    recall  f1-score   support

           0       0.44      0.90      0.59        21
           1       0.98      0.82      0.89       133

    accuracy                           0.83       154
   macro avg       0.71      0.86      0.74       154
weighted avg       0.91      0.83      0.85       154

Logistic Regression:> 0.8311688311688312
```

In [22]:

```
#Experiment 2: Sclaed + Down Sampled Data

#X = df_downsampled.drop(["Loan_ID","Gender","Married","Education","Self
_Employed","Loan_Amount_Term","Loan_Status",'Property_Area'],axis=1)
X = df_downsampled.drop(["Loan_Status","Loan_ID"],axis=1)
y = df_downsampled.Loan_Status
X_train , X_test , y_train , y_test = train_test_split(X,y,test_size=0.2
5,random_state=0)


#Scaling data here:------------->

StSc = StandardScaler()
X_train  = StSc.fit_transform(X_train)
X_test  = StSc.fit_transform(X_test)

#Check mean is 0 and Standard deviation is 1
print("After Standardization\nMean ",np.mean(X_train),"Standard Deviatio
n ",np.std(X_train),"\n")

#Voting ensemble mathod. Combining all tree based algorithms.
models = []
models.append(("XGB",XGBClassifier()))
models.append(("RF",RandomForestClassifier()))
models.append(("DT",DecisionTreeClassifier()))
models.append(("ADB",AdaBoostClassifier()))
models.append(("GB",GradientBoostingClassifier()))

ensemble = VotingClassifier(estimators=models)
ensemble.fit(X_train,y_train)
y_pred = ensemble.predict(X_test)
print(classification_report(y_pred,y_test))
print("Voting Ensemble:>",accuracy_score(y_pred,y_test))



SVM = SVC(kernel="linear",class_weight="balanced",probability=True)
SVM.fit(X_train,y_train)
y_pred = SVM.predict(X_test)
print(classification_report(y_pred,y_test))
print("SVM:>",accuracy_score(y_pred,y_test))


XGBC = XGBClassifier(learning_rate =0.1,n_estimators=10000,max_depth=4,m
in_child_weight=6,gamma=0,subsample=0.6,colsample_bytree=0.8,
 reg_alpha=0.005, objective= 'binary:logistic', nthread=2, scale_pos_wei
ght=1, seed=27)
XGBC.fit(X_train,y_train)
y_pred = XGBC.predict(X_test)
print(classification_report(y_pred,y_test))
print("XGBoost:>",accuracy_score(y_pred,y_test))

Model1 = RandomForestClassifier(n_estimators=1000,random_state=0,n_jobs=
1000,max_depth=70,bootstrap=True)
Model1.fit(X_train,y_train)
y_pred = Model1.predict(X_test)
print(classification_report(y_pred,y_test))
print("RandomForestClassifier:>",accuracy_score(y_pred,y_test))


Model2 = GradientBoostingClassifier()
Model2.fit(X_train,y_train)
y_pred = Model2.predict(X_test)
print(classification_report(y_pred,y_test))
print("GradientBoostingClassifier:>",accuracy_score(y_pred,y_test))


Model3 = DecisionTreeClassifier(class_weight=None, criterion='gini', max
_depth=100,
 max_features=1.0, max_leaf_nodes=10,
 min_impurity_split=1e-07, min_samples_leaf=1,
 min_samples_split=2, min_weight_fraction_leaf=0.10,
 presort=False, random_state=27, splitter='best')
Model3.fit(X_train,y_train)
y_pred = Model3.predict(X_test)
print(classification_report(y_pred,y_test))
print("DecisionTreeClassifier:>",accuracy_score(y_pred,y_test))


Model4 = AdaBoostClassifier()
Model4.fit(X_train,y_train)
y_pred = Model4.predict(X_test)
print(classification_report(y_pred,y_test))
print("AdaBoostClassifier:>",accuracy_score(y_pred,y_test))


Model5 = LinearDiscriminantAnalysis()
Model5.fit(X_train,y_train)
y_pred = Model5.predict(X_test)
print(classification_report(y_pred,y_test))
print("LinearDiscriminantAnalysis:>",accuracy_score(y_pred,y_test))

KNN = KNeighborsClassifier(leaf_size=1,p=2,n_neighbors=20)
KNN.fit(X_train,y_train)
y_pred = KNN.predict(X_test)
print(classification_report(y_pred,y_test))
print("KNeighborsClassifier:>",accuracy_score(y_pred,y_test))


Model7 = GaussianNB()
Model7.fit(X_train,y_train)
y_pred = Model7.predict(X_test)
print(classification_report(y_pred,y_test))
print("GaussianNB:>",accuracy_score(y_pred,y_test))


Model8 = LogisticRegression(C=1.0, class_weight=None, dual=False, fit_in
tercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=
1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.000
1,
          verbose=0, warm_start=False)
Model8.fit(X_train,y_train)
y_pred = Model8.predict(X_test)
print(classification_report(y_pred,y_test))
print("Logistic Regression:>",accuracy_score(y_pred,y_test))
```

```
After Standardization
Mean  -3.064327691705293e-16 Standard Deviation  1.0
              precision    recall  f1-score   support

           0       0.60      0.76      0.67        38
           1       0.81      0.67      0.74        58

    accuracy                           0.71        96
   macro avg       0.71      0.72      0.71        96
weighted avg       0.73      0.71      0.71        96

Voting Ensemble:> 0.7083333333333334
              precision    recall  f1-score   support

           0       0.42      1.00      0.59        20
           1       1.00      0.63      0.77        76

    accuracy                           0.71        96
   macro avg       0.71      0.82      0.68        96
weighted avg       0.88      0.71      0.74        96

SVM:> 0.7083333333333334
              precision    recall  f1-score   support

           0       0.48      0.62      0.54        37
           1       0.71      0.58      0.64        59

    accuracy                           0.59        96
   macro avg       0.59      0.60      0.59        96
weighted avg       0.62      0.59      0.60        96

XGBoost:> 0.59375
              precision    recall  f1-score   support

           0       0.58      0.78      0.67        36
           1       0.83      0.67      0.74        60

    accuracy                           0.71        96
   macro avg       0.71      0.72      0.70        96
weighted avg       0.74      0.71      0.71        96

RandomForestClassifier:> 0.7083333333333334
              precision    recall  f1-score   support

           0       0.56      0.64      0.60        42
           1       0.69      0.61      0.65        54

    accuracy                           0.62        96
   macro avg       0.62      0.63      0.62        96
weighted avg       0.63      0.62      0.63        96

GradientBoostingClassifier:> 0.625
              precision    recall  f1-score   support

           0       0.54      0.87      0.67        30
           1       0.92      0.67      0.77        66

    accuracy                           0.73        96
   macro avg       0.73      0.77      0.72        96
weighted avg       0.80      0.73      0.74        96

DecisionTreeClassifier:> 0.7291666666666666
              precision    recall  f1-score   support

           0       0.58      0.85      0.69        33
           1       0.90      0.68      0.77        63

    accuracy                           0.74        96
   macro avg       0.74      0.77      0.73        96
weighted avg       0.79      0.74      0.75        96

AdaBoostClassifier:> 0.7395833333333334
              precision    recall  f1-score   support

           0       0.50      0.77      0.61        31
           1       0.85      0.63      0.73        65

    accuracy                           0.68        96
   macro avg       0.68      0.70      0.67        96
weighted avg       0.74      0.68      0.69        96

LinearDiscriminantAnalysis:> 0.6770833333333334
              precision    recall  f1-score   support

           0       0.50      0.83      0.62        29
           1       0.90      0.64      0.75        67

    accuracy                           0.70        96
   macro avg       0.70      0.73      0.69        96
weighted avg       0.78      0.70      0.71        96

KNeighborsClassifier:> 0.6979166666666666
              precision    recall  f1-score   support

           0       0.42      0.91      0.57        22
           1       0.96      0.62      0.75        74

    accuracy                           0.69        96
   macro avg       0.69      0.77      0.66        96
weighted avg       0.83      0.69      0.71        96

GaussianNB:> 0.6875
              precision    recall  f1-score   support

           0       0.54      0.74      0.63        35
           1       0.81      0.64      0.72        61

    accuracy                           0.68        96
   macro avg       0.68      0.69      0.67        96
weighted avg       0.71      0.68      0.68        96

Logistic Regression:> 0.6770833333333334
```

In [23]:

```
#Experiment 3: Sclaed + Up Sampled Data

#X = df_upsampled.drop(["Loan_ID","Gender","Married","Education","Self_E
mployed","Loan_Amount_Term","Loan_Status",'Property_Area'],axis=1)
X = df_upsampled.drop(["Loan_Status","Loan_ID"],axis=1)
y = df_upsampled.Loan_Status
print(len(X),len(y))
X_train , X_test , y_train , y_test = train_test_split(X,y,test_size=0.2
5,random_state=0)

#Scaling data here:------------->

StSc = StandardScaler()
X_train  = StSc.fit_transform(X_train)
X_test  = StSc.fit_transform(X_test)

#Check mean is 0 and Standard deviation is 1
print("After Standardization\nMean ",np.mean(X_train),"Standard Deviatio
n ",np.std(X_train),"\n")

#Voting ensemble mathod. Combining all tree based algorithms.
models = []
models.append(("XGB",XGBClassifier()))
models.append(("RF",RandomForestClassifier()))
models.append(("DT",DecisionTreeClassifier()))
models.append(("ADB",AdaBoostClassifier()))
models.append(("GB",GradientBoostingClassifier()))

ensemble = VotingClassifier(estimators=models)
ensemble.fit(X_train,y_train)
y_pred = ensemble.predict(X_test)
print(classification_report(y_pred,y_test))
print("Voting Ensemble:>",accuracy_score(y_pred,y_test))



SVM = SVC(kernel="linear",class_weight="balanced",probability=True)
SVM.fit(X_train,y_train)
y_pred = SVM.predict(X_test)
print(classification_report(y_pred,y_test))
print("SVM:>",accuracy_score(y_pred,y_test))


XGBC = XGBClassifier(learning_rate =0.1,n_estimators=10000,max_depth=4,m
in_child_weight=6,gamma=0,subsample=0.6,colsample_bytree=0.8,
 reg_alpha=0.005, objective= 'binary:logistic', nthread=2, scale_pos_wei
ght=1, seed=27)
XGBC.fit(X_train,y_train)
y_pred = XGBC.predict(X_test)
print(classification_report(y_pred,y_test))
print("XGBoost:>",accuracy_score(y_pred,y_test))

Model1 = RandomForestClassifier(n_estimators=1000,random_state=0,n_jobs=
1000,max_depth=70,bootstrap=True)
Model1.fit(X_train,y_train)
y_pred = Model1.predict(X_test)
print(classification_report(y_pred,y_test))
print("RandomForestClassifier:>",accuracy_score(y_pred,y_test))


Model2 = GradientBoostingClassifier()
Model2.fit(X_train,y_train)
y_pred = Model2.predict(X_test)
print(classification_report(y_pred,y_test))
print("GradientBoostingClassifier:>",accuracy_score(y_pred,y_test))


Model3 = DecisionTreeClassifier(class_weight=None, criterion='gini', max
_depth=100,
 max_features=1.0, max_leaf_nodes=10,
 min_impurity_split=1e-07, min_samples_leaf=1,
 min_samples_split=2, min_weight_fraction_leaf=0.10,
 presort=False, random_state=27, splitter='best')
Model3.fit(X_train,y_train)
y_pred = Model3.predict(X_test)
print(classification_report(y_pred,y_test))
print("DecisionTreeClassifier:>",accuracy_score(y_pred,y_test))


Model4 = AdaBoostClassifier()
Model4.fit(X_train,y_train)
y_pred = Model4.predict(X_test)
print(classification_report(y_pred,y_test))
print("AdaBoostClassifier:>",accuracy_score(y_pred,y_test))


Model5 = LinearDiscriminantAnalysis()
Model5.fit(X_train,y_train)
y_pred = Model5.predict(X_test)
print(classification_report(y_pred,y_test))
print("LinearDiscriminantAnalysis:>",accuracy_score(y_pred,y_test))

KNN = KNeighborsClassifier(leaf_size=1,p=2,n_neighbors=20)
KNN.fit(X_train,y_train)
y_pred = KNN.predict(X_test)
print(classification_report(y_pred,y_test))
print("KNeighborsClassifier:>",accuracy_score(y_pred,y_test))


Model7 = GaussianNB()
Model7.fit(X_train,y_train)
y_pred = Model7.predict(X_test)
print(classification_report(y_pred,y_test))
print("GaussianNB:>",accuracy_score(y_pred,y_test))


Model8 = LogisticRegression(C=1.0, class_weight=None, dual=False, fit_in
tercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=
1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.000
1,
          verbose=0, warm_start=False)
Model8.fit(X_train,y_train)
y_pred = Model8.predict(X_test)
print(classification_report(y_pred,y_test))
print("Logistic Regression:>",accuracy_score(y_pred,y_test))
```

```
844 844
After Standardization
Mean  7.143184188310644e-17 Standard Deviation  1.0
              precision    recall  f1-score   support

           0       0.71      0.86      0.78        90
           1       0.87      0.74      0.80       121

    accuracy                           0.79       211
   macro avg       0.79      0.80      0.79       211
weighted avg       0.81      0.79      0.79       211

Voting Ensemble:> 0.7914691943127962
              precision    recall  f1-score   support

           0       0.38      0.95      0.54        43
           1       0.98      0.60      0.75       168

    accuracy                           0.67       211
   macro avg       0.68      0.78      0.64       211
weighted avg       0.86      0.67      0.70       211

SVM:> 0.6729857819905213
              precision    recall  f1-score   support

           0       0.67      0.76      0.71        95
           1       0.78      0.69      0.73       116

    accuracy                           0.72       211
   macro avg       0.72      0.72      0.72       211
weighted avg       0.73      0.72      0.72       211

XGBoost:> 0.7203791469194313
              precision    recall  f1-score   support

           0       0.80      0.88      0.83        98
           1       0.88      0.81      0.84       113

    accuracy                           0.84       211
   macro avg       0.84      0.84      0.84       211
weighted avg       0.84      0.84      0.84       211

RandomForestClassifier:> 0.8388625592417062
              precision    recall  f1-score   support

           0       0.59      0.77      0.67        83
           1       0.82      0.66      0.73       128

    accuracy                           0.70       211
   macro avg       0.70      0.71      0.70       211
weighted avg       0.73      0.70      0.70       211

GradientBoostingClassifier:> 0.7014218009478673
              precision    recall  f1-score   support

           0       0.54      0.70      0.61        83
           1       0.76      0.61      0.68       128

    accuracy                           0.64       211
   macro avg       0.65      0.65      0.64       211
weighted avg       0.67      0.64      0.65       211

DecisionTreeClassifier:> 0.6445497630331753
              precision    recall  f1-score   support

           0       0.63      0.76      0.69        89
           1       0.80      0.67      0.73       122

    accuracy                           0.71       211
   macro avg       0.71      0.72      0.71       211
weighted avg       0.73      0.71      0.71       211

AdaBoostClassifier:> 0.7109004739336493
              precision    recall  f1-score   support

           0       0.44      0.78      0.56        60
           1       0.87      0.60      0.71       151

    accuracy                           0.65       211
   macro avg       0.65      0.69      0.63       211
weighted avg       0.75      0.65      0.67       211

LinearDiscriminantAnalysis:> 0.6492890995260664
              precision    recall  f1-score   support

           0       0.55      0.81      0.65        73
           1       0.86      0.64      0.74       138

    accuracy                           0.70       211
   macro avg       0.71      0.73      0.70       211
weighted avg       0.75      0.70      0.71       211

KNeighborsClassifier:> 0.7014218009478673
              precision    recall  f1-score   support

           0       0.40      0.88      0.55        49
           1       0.94      0.60      0.73       162

    accuracy                           0.66       211
   macro avg       0.67      0.74      0.64       211
weighted avg       0.82      0.66      0.69       211

GaussianNB:> 0.6635071090047393
              precision    recall  f1-score   support

           0       0.45      0.79      0.58        62
           1       0.87      0.60      0.71       149

    accuracy                           0.66       211
   macro avg       0.66      0.70      0.65       211
weighted avg       0.75      0.66      0.67       211

Logistic Regression:> 0.6587677725118484
```

In [24]:

```
# Experiment 4: Sclaed + Selected features with respective importance
#Droping features which are less important and keeping features as per i
mportance analysis.
X = new_data.drop(["Loan_ID","Gender","Married","Education","Self_Employ
ed","Loan_Amount_Term","Loan_Status","Property_Area"],axis=1)
#X = new_data.drop(["Loan_Status","Loan_ID"],axis=1)
y = new_data.Loan_Status
print(len(X),len(y))
X_train , X_test , y_train , y_test = train_test_split(X,y,test_size=0.2
5,random_state=0)

#Scaling data here:------------->

StSc = StandardScaler()
X_train  = StSc.fit_transform(X_train)
X_test  = StSc.fit_transform(X_test)

#Check mean is 0 and Standard deviation is 1
print("After Standardization\nMean ",np.mean(X_train),"Standard Deviatio
n ",np.std(X_train),"\n")

#Voting ensemble mathod. Combining all tree based algorithms.
models = []
models.append(("XGB",XGBClassifier()))
models.append(("RF",RandomForestClassifier()))
models.append(("DT",DecisionTreeClassifier()))
models.append(("ADB",AdaBoostClassifier()))
models.append(("GB",GradientBoostingClassifier()))

ensemble = VotingClassifier(estimators=models)
ensemble.fit(X_train,y_train)
y_pred = ensemble.predict(X_test)
print(classification_report(y_pred,y_test))
print("Voting Ensemble:>",accuracy_score(y_pred,y_test))



SVM = SVC(kernel="linear",class_weight="balanced",probability=True)
SVM.fit(X_train,y_train)
y_pred = SVM.predict(X_test)
print(classification_report(y_pred,y_test))
print("SVM:>",accuracy_score(y_pred,y_test))


XGBC = XGBClassifier(learning_rate =0.1,n_estimators=10000,max_depth=4,m
in_child_weight=6,gamma=0,subsample=0.6,colsample_bytree=0.8,
 reg_alpha=0.005, objective= 'binary:logistic', nthread=2, scale_pos_wei
ght=1, seed=27)
XGBC.fit(X_train,y_train)
y_pred = XGBC.predict(X_test)
print(classification_report(y_pred,y_test))
print("XGBoost:>",accuracy_score(y_pred,y_test))

Model1 = RandomForestClassifier(n_estimators=1000,random_state=0,n_jobs=
1000,max_depth=70,bootstrap=True)
Model1.fit(X_train,y_train)
y_pred = Model1.predict(X_test)
print(classification_report(y_pred,y_test))
print("RandomForestClassifier:>",accuracy_score(y_pred,y_test))


Model2 = GradientBoostingClassifier()
Model2.fit(X_train,y_train)
y_pred = Model2.predict(X_test)
print(classification_report(y_pred,y_test))
print("GradientBoostingClassifier:>",accuracy_score(y_pred,y_test))


Model3 = DecisionTreeClassifier(class_weight=None, criterion='gini', max
_depth=100,
 max_features=1.0, max_leaf_nodes=10,
 min_impurity_split=1e-07, min_samples_leaf=1,
 min_samples_split=2, min_weight_fraction_leaf=0.10,
 presort=False, random_state=27, splitter='best')
Model3.fit(X_train,y_train)
y_pred = Model3.predict(X_test)
print(classification_report(y_pred,y_test))
print("DecisionTreeClassifier:>",accuracy_score(y_pred,y_test))


Model4 = AdaBoostClassifier()
Model4.fit(X_train,y_train)
y_pred = Model4.predict(X_test)
print(classification_report(y_pred,y_test))
print("AdaBoostClassifier:>",accuracy_score(y_pred,y_test))


Model5 = LinearDiscriminantAnalysis()
Model5.fit(X_train,y_train)
y_pred = Model5.predict(X_test)
print(classification_report(y_pred,y_test))
print("LinearDiscriminantAnalysis:>",accuracy_score(y_pred,y_test))

KNN = KNeighborsClassifier(leaf_size=1,p=2,n_neighbors=20)
KNN.fit(X_train,y_train)
y_pred = KNN.predict(X_test)
print(classification_report(y_pred,y_test))
print("KNeighborsClassifier:>",accuracy_score(y_pred,y_test))


Model7 = GaussianNB()
Model7.fit(X_train,y_train)
y_pred = Model7.predict(X_test)
print(classification_report(y_pred,y_test))
print("GaussianNB:>",accuracy_score(y_pred,y_test))


Model8 = LogisticRegression(C=1.0, class_weight=None, dual=False, fit_in
tercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=
1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.000
1,
          verbose=0, warm_start=False)
Model8.fit(X_train,y_train)
y_pred = Model8.predict(X_test)
print(classification_report(y_pred,y_test))
print("Logistic Regression:>",accuracy_score(y_pred,y_test))
```

```
614 614
After Standardization
Mean  -3.2669519263752433e-16 Standard Deviation  1.0
              precision    recall  f1-score   support

           0       0.47      0.71      0.56        28
           1       0.93      0.82      0.87       126

    accuracy                           0.80       154
   macro avg       0.70      0.77      0.72       154
weighted avg       0.84      0.80      0.81       154

Voting Ensemble:> 0.7987012987012987
              precision    recall  f1-score   support

           0       0.44      0.90      0.59        21
           1       0.98      0.82      0.89       133

    accuracy                           0.83       154
   macro avg       0.71      0.86      0.74       154
weighted avg       0.91      0.83      0.85       154

SVM:> 0.8311688311688312
              precision    recall  f1-score   support

           0       0.23      0.33      0.27        30
           1       0.82      0.73      0.77       124

    accuracy                           0.66       154
   macro avg       0.53      0.53      0.52       154
weighted avg       0.71      0.66      0.68       154

XGBoost:> 0.6558441558441559
              precision    recall  f1-score   support

           0       0.47      0.69      0.56        29
           1       0.92      0.82      0.86       125

    accuracy                           0.79       154
   macro avg       0.69      0.75      0.71       154
weighted avg       0.83      0.79      0.81       154

RandomForestClassifier:> 0.7922077922077922
              precision    recall  f1-score   support

           0       0.44      0.73      0.55        26
           1       0.94      0.81      0.87       128

    accuracy                           0.80       154
   macro avg       0.69      0.77      0.71       154
weighted avg       0.85      0.80      0.82       154

GradientBoostingClassifier:> 0.7987012987012987
              precision    recall  f1-score   support

           0       0.44      0.90      0.59        21
           1       0.98      0.82      0.89       133

    accuracy                           0.83       154
   macro avg       0.71      0.86      0.74       154
weighted avg       0.91      0.83      0.85       154

DecisionTreeClassifier:> 0.8311688311688312
              precision    recall  f1-score   support

           0       0.47      0.83      0.60        24
           1       0.96      0.82      0.89       130

    accuracy                           0.82       154
   macro avg       0.71      0.83      0.74       154
weighted avg       0.89      0.82      0.84       154

AdaBoostClassifier:> 0.8246753246753247
              precision    recall  f1-score   support

           0       0.44      0.90      0.59        21
           1       0.98      0.82      0.89       133

    accuracy                           0.83       154
   macro avg       0.71      0.86      0.74       154
weighted avg       0.91      0.83      0.85       154

LinearDiscriminantAnalysis:> 0.8311688311688312
              precision    recall  f1-score   support

           0       0.44      0.90      0.59        21
           1       0.98      0.82      0.89       133

    accuracy                           0.83       154
   macro avg       0.71      0.86      0.74       154
weighted avg       0.91      0.83      0.85       154

KNeighborsClassifier:> 0.8311688311688312
              precision    recall  f1-score   support

           0       0.44      0.90      0.59        21
           1       0.98      0.82      0.89       133

    accuracy                           0.83       154
   macro avg       0.71      0.86      0.74       154
weighted avg       0.91      0.83      0.85       154

GaussianNB:> 0.8311688311688312
              precision    recall  f1-score   support

           0       0.44      0.90      0.59        21
           1       0.98      0.82      0.89       133

    accuracy                           0.83       154
   macro avg       0.71      0.86      0.74       154
weighted avg       0.91      0.83      0.85       154

Logistic Regression:> 0.8311688311688312
```

In [25]:
```
#Hyperparameters tuning for KNN

#X = new_data.drop(["Loan_ID","Gender","Married","Education","Self_Emplo
yed","Loan_Amount_Term","Loan_Status","Property_Area"],axis=1)
X = new_data.drop(["Loan_Status","Loan_ID"],axis=1)
y = new_data.Loan_Status
print(len(X),len(y))
X_train , X_test , y_train , y_test = train_test_split(X,y,test_size=0.2
5,random_state=0)



leaf_size = list(range(1,50))
n_neighbors = list(range(1,30))
p=[1,2]
#Convert to dictionary
hyperparameters = dict(leaf_size=leaf_size, n_neighbors=n_neighbors, p=
p)
#Create new KNN object
knn_2 = KNeighborsClassifier()
#Use GridSearch
clf = GridSearchCV(knn_2, hyperparameters, cv=10)
#Fit the model
best_model = clf.fit(X_train,y_train)
#Print The value of best Hyperparameters
print('Best leaf_size:', best_model.best_estimator_.get_params()['leaf_s
ize'])
print('Best p:', best_model.best_estimator_.get_params()['p'])
print('Best n_neighbors:', best_model.best_estimator_.get_params()['n_ne
ighbors'])

LS = best_model.best_estimator_.get_params()['leaf_size']
P = best_model.best_estimator_.get_params()['p']
Num = best_model.best_estimator_.get_params()['n_neighbors']

KNN = KNeighborsClassifier(leaf_size=LS,p=P,n_neighbors=Num)
KNN.fit(X_train,y_train)
y_pred = KNN.predict(X_test)
print(classification_report(y_pred,y_test))
print("KNeighborsClassifier:>",accuracy_score(y_pred,y_test))
```

```
614 614
Best leaf_size: 1
Best p: 1
Best n_neighbors: 10
              precision    recall  f1-score   support

           0       0.49      0.84      0.62        25
           1       0.96      0.83      0.89       129

    accuracy                           0.83       154
   macro avg       0.73      0.83      0.75       154
weighted avg       0.89      0.83      0.85       154

KNeighborsClassifier:> 0.8311688311688312
```

Conclusion

Resut Summary is as below:-----> Algorithm : Accuracy

Experiment 1 : Scaled data only

```
Support Vector Machine       83.116
Decision Tree                83.1168
Linear Discriminant Analysis 83.166
KNearest Neighbors           83.766
Gaussian Naivey Bayes        83.116
Logistic Regression          83.116
```

Experiment 2: Sclaed + Down Sampled Data

```
AdaBoost          73.95
Decision Tree     72.91
Voting Ensemble   71.87
```

Experiment 3: Sclaed + Up Sampled Data

```
Random Forest only 83.88
```

Experiment 4: Sclaed + Selected features with respective importance

```
Support Vector Machine       83.11
Decision Tree                83.11
AdaBoost                     82.46
Linear Discriminant Analysis 83.11
KNearest Neighbors           83.11
Gaussian Naivey Bayes        83.11
Logistic Regression          83.11
```