Loan Approval Prediction

Sai Swaranhi sree Vinnakota

Abstract

Dataset:
I have chosen Loan Approval Dataset for my analysis

Research Question:
Can you predict loan approval status of an applicant that can be approved or denied based on the given data? What are the major factors that affect approval?Which among Decision Tree, Naive Bayes and KNN Classifiers, predicts the status more accurately?

Methods:

DecisionTree Classification, Naive Bayes Classification and K-Nearest Neighbour Algorithm

Findings:

Loan Approval can be predicted with accuracy of ~87% and hence classification algorithms can be used to decide if the loan can be approved or denied .Naive Bayes and Decision tree algorithms predict well .

Motivation

The main portion of the the bank’s assets directly come from the profit earned from the loans distributed by the banks.
However, there are some customers who don’t repay their loan after their application is approved. To prevent this situation, banks have to find some methods to predict customers’ behaviour based on their income, credit history, education and other factors. Machine learning algorithms have a pretty good performance on this purpose, which are widely-used by the banking

Dataset(s)

LoanApproval dataset contains information about the loan applicant such as ApplicantIncome, Credit History, LoanTerm ,Education ,Amount of loan taken, Loan Status of Approval

This data is obtained from datahack and it has 981 rows and 13 columns

Data Preparation and Cleaning

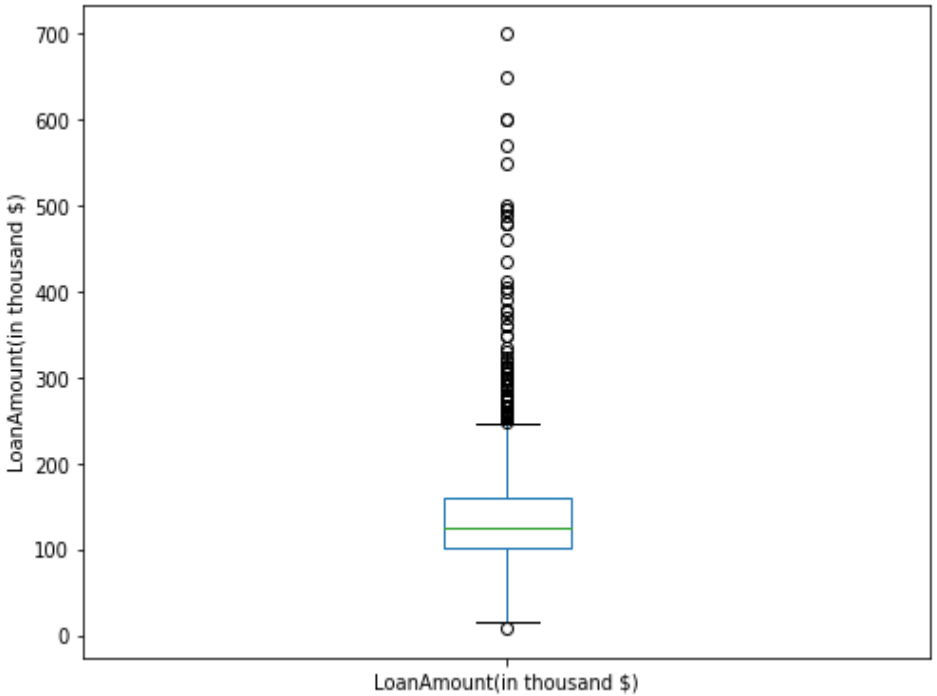
Outliers:There are outliers in Loan Amount ,so I have applied log transformation to reduce the impact on the distribution of data

Missing values: Replaced them with mode if it is a categorical variable,mean or median for numerical variables

Adding features:I have added a column EMI to the data as applicants who have a less EMI to pay monthly are likely to repay their loan and hence get their loan approved

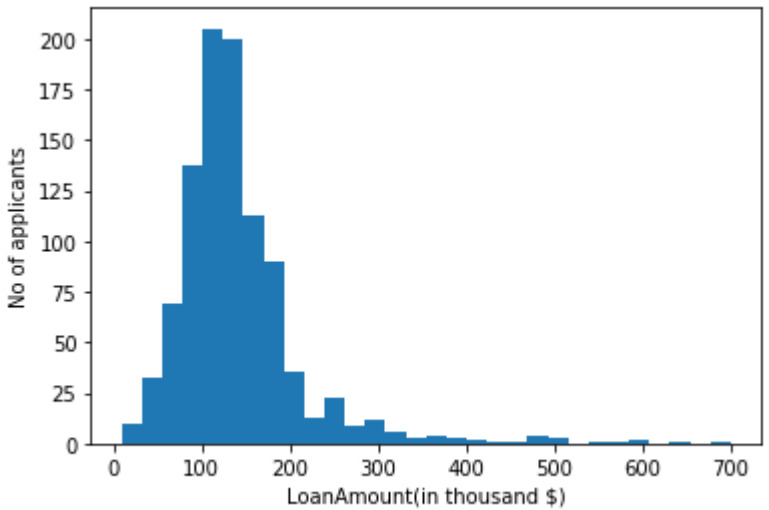
Dropped a column LoanID as it is not required to decide the approval .

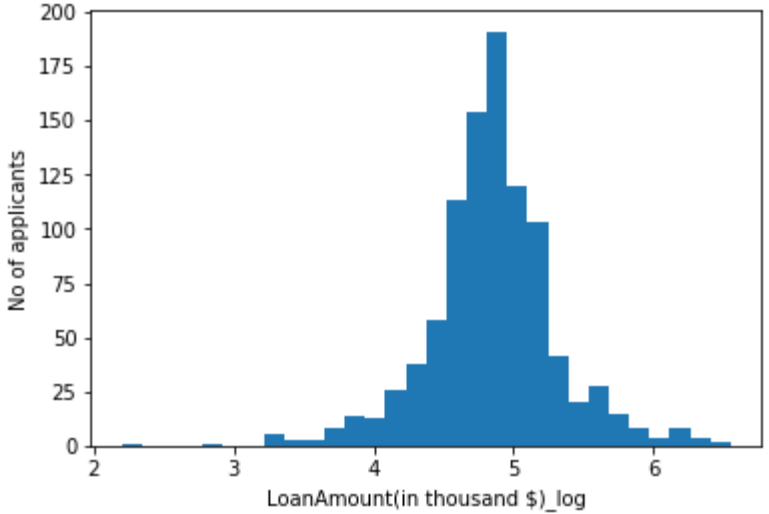
Handling Outliers



There are significant number of outliers in LoanAmount which can be handled using log transformation.

Handling Outliers





- Most of the graph(left) is towards left, after log transformation ,the graph on the right is normal

Research Question(s)

Can you predict loan approval status of an applicant accurately which can be approved or denied based on the given data? What are the major factors affecting the approval?

Which among Decision Tree, Naive Bayes and KNN Classifiers, predicts the loan status more accurately?

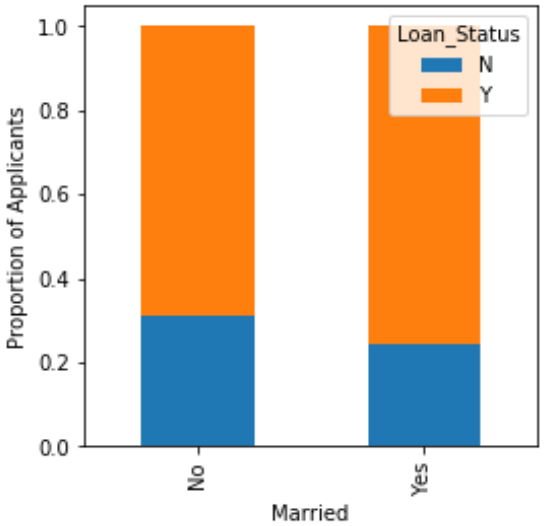
Methods

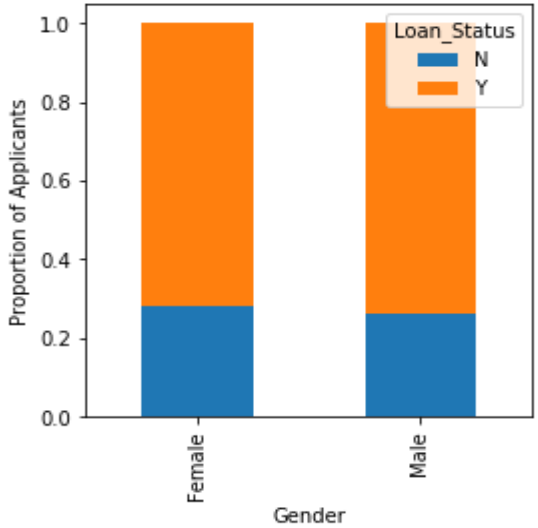
I have used the below models to train the model because the goal is to predict the category of the target

- Decision Tree: Model is built by constructing a tree-like structure which starts at root node where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label.
- Naive Bayes:The Naive Bayes Classifier technique is based on the Bayesian theorem and it uses probabilistic approach to classify.
- KNN:The K Nearest Neighbour algorithm assumes that similar things exist in close proximity. Number of neighbours(n_neighbours) is randomly selected and the distance between test data and each training sample is calculated

Findings

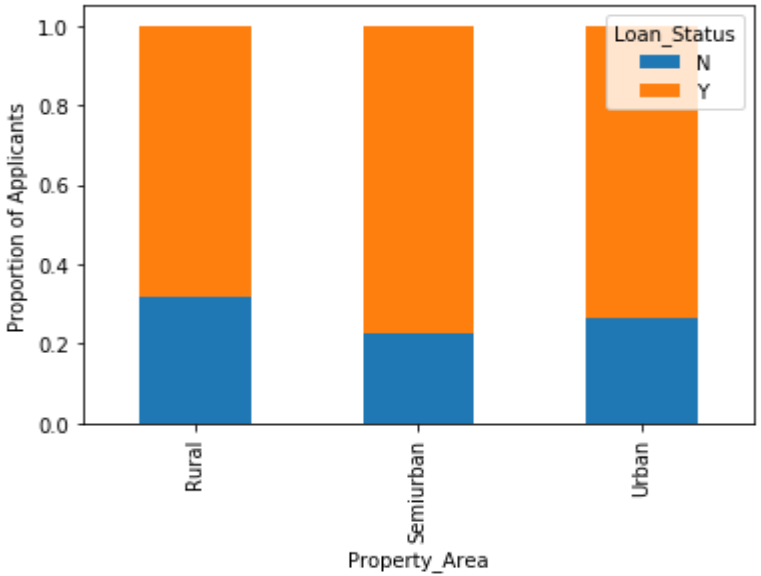
Finding most important factors in predicting the loan status:





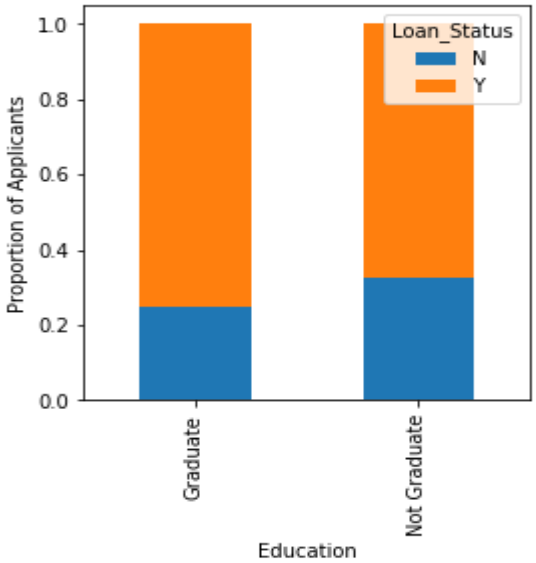
It is observed that “gender” doesn’t play any role in determining loan status as proportion of both male and female is similar for approved and unapproved loan.
There is very less difference in the approved and unapproved loan of the married and unmarried .

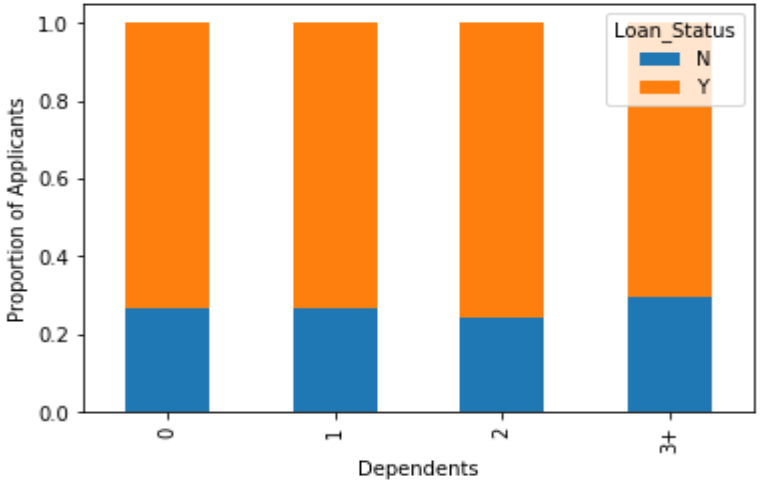
Findings



Most of the loans that are getting approved are from semiurban areas
The applicants who have repaid their previous loan have high chance of getting their loan approved

Findings





Proportion of graduates is little higher for approved loans

Proportion of 0 ,1 ,2 or more dependents is almost the same

Findings

Building the model using Decision tree classifier and predicting the target has around 87% of accuracy where Naive bayes has accuracy of ~86% but KNN classifier has predicted less accurately with an accuracy of ~68%.

Important factors that are involved in predicting are:Applicant Income,Coapplicant Income,Credit_History and Loan_Amount

Limitations

The dataset I have used is smaller in size with around 981 rows .So,the accuracy might have increased if the model was built with large amount of data.

Conclusions

Loan can be approved or denied with an accuracy of ~87% with decision tree or naive bayes classification algorithms. Major features that affect the prediction are Credit History and Loan Amount, Applicant Income, Coapplicant Income .

Gender, Number of dependents ,Self-Employment, Marital Status are not necessary for approval of loan

Acknowledgements

I have collected the data from datahack .I had my colleages review and provide feedback.

References

Research paper : Loan Approval Prediction based on Machine Learning Approach
Kumar Arun, Garg Ishan, Kaur Sanmeet

http://www.iosrjournals.org/iosr-jce/papers/conf.15013/Volume%203/4.%2018-21.pdf?id=7557

In [1]:
```
import pandas as pd
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plt
```

In [2]:
```
Loan_df = pd.read_csv("LoanApproval.csv")
```

In [3]:
```
Loan_df.columns
```

Out[3]:
```
Index(['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education',
       'Self_Employed', 'ApplicantIncome(in $)', 'CoapplicantIncome(in $)',
       'LoanAmount(in thousand $)', 'Loan_Amount_Term(in months)',
       'Credit_History', 'Property_Area', 'Loan_Status'],
      dtype='object')
```

In [4]:
```
Loan_df.head()
```

Out[4]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome(in $)	Coappl
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	

In [5]:
```
Loan_df.shape
```

Out[5]:
```
(981, 13)
```

In [6]:
```
Loan_df.describe()
```

Out[6]:

	ApplicantIncome(in $)	CoapplicantIncome(in $)	LoanAmount(in thousand $)	Loan_Amount_Term(in months)	Credit_Hist
count	981.000000	981.000000	954.000000	961.000000	902.000(
mean	5179.795107	1601.916330	142.511530	342.201873	0.835(
std	5695.104533	2718.772806	77.421743	65.100602	0.370(
min	0.000000	0.000000	9.000000	6.000000	0.000(
25%	2875.000000	0.000000	100.000000	360.000000	1.000(
50%	3800.000000	1110.000000	126.000000	360.000000	1.000(
75%	5516.000000	2365.000000	162.000000	360.000000	1.000(
max	81000.000000	41667.000000	700.000000	480.000000	1.000(

Data Cleaning

Checking if there are any missing values,outliers and invalid data

In [7]:
```
Loan_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 981 entries, 0 to 980
Data columns (total 13 columns):
Loan_ID                          981 non-null object
Gender                           957 non-null object
Married                          978 non-null object
Dependents                       956 non-null object
Education                        981 non-null object
Self_Employed                    926 non-null object
ApplicantIncome(in $)            981 non-null int64
CoapplicantIncome(in $)          981 non-null float64
LoanAmount(in thousand $)        954 non-null float64
Loan_Amount_Term(in months)      961 non-null float64
Credit_History                   902 non-null float64
Property_Area                    981 non-null object
Loan_Status                      981 non-null object
dtypes: float64(4), int64(1), object(8)
memory usage: 99.8+ KB
```

In [8]:
```
Loan_df.isnull().any()
```

Out[8]:
```
Loan_ID                          False
Gender                            True
Married                           True
Dependents                        True
Education                        False
Self_Employed                     True
ApplicantIncome(in $)            False
CoapplicantIncome(in $)          False
LoanAmount(in thousand $)         True
Loan_Amount_Term(in months)       True
Credit_History                    True
Property_Area                    False
Loan_Status                      False
dtype: bool
```

It is observed that Gender,Married,Dependents,Self-Employed,LoanAmount(in thousand $),Loan_Amount_Term(in months),Credit_History columns are having null values.

Type *Markdown* and LaTeX: α^2

In [ ]:

In [ ]:

In [ ]:

Filling Values

There are very less missing values in Gender, Married, Dependents, Credit_History and Self_Employed features so we can fill them using the mode of the features since they are categorical variables

In [9]:
```
Loan_df['Gender'].fillna(Loan_df['Gender'].mode()[0], inplace=True)
Loan_df['Married'].fillna(Loan_df['Married'].mode()[0], inplace=True)
Loan_df['Dependents'].fillna(Loan_df['Dependents'].mode()[0], inplace=True)
Loan_df['Self_Employed'].fillna(Loan_df['Self_Employed'].mode()[0], inplace=True
Loan_df['Credit_History'].fillna(Loan_df['Credit_History'].mode()[0], inplace=Tr
Loan_df['Loan_Amount_Term(in months)'].fillna(Loan_df['Loan_Amount_Term(in month
```

In [10]:
```
Loan_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 981 entries, 0 to 980
Data columns (total 13 columns):
Loan_ID                          981 non-null object
Gender                           981 non-null object
Married                          981 non-null object
Dependents                       981 non-null object
Education                        981 non-null object
Self_Employed                    981 non-null object
ApplicantIncome(in $)            981 non-null int64
CoapplicantIncome(in $)          981 non-null float64
LoanAmount(in thousand $)        954 non-null float64
Loan_Amount_Term(in months)      981 non-null float64
Credit_History                   981 non-null float64
Property_Area                    981 non-null object
Loan_Status                      981 non-null object
dtypes: float64(4), int64(1), object(8)
memory usage: 99.8+ KB
```

In [11]:
```
Loan_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 981 entries, 0 to 980
Data columns (total 13 columns):
Loan_ID                          981 non-null object
Gender                           981 non-null object
Married                          981 non-null object
Dependents                       981 non-null object
Education                        981 non-null object
Self_Employed                    981 non-null object
ApplicantIncome(in $)            981 non-null int64
CoapplicantIncome(in $)          981 non-null float64
LoanAmount(in thousand $)        954 non-null float64
Loan_Amount_Term(in months)      981 non-null float64
Credit_History                   981 non-null float64
Property_Area                    981 non-null object
Loan_Status                      981 non-null object
dtypes: float64(4), int64(1), object(8)
memory usage: 99.8+ KB
```

As Loan Amount is numerical variable,it can be filled with either mean or median .But since there are outliers in the data,mean cannot be used.

In [12]:
```
Loan_df['LoanAmount(in thousand $)'].fillna(Loan_df['LoanAmount(in thousand $)']
```

In [13]:
```
Loan_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 981 entries, 0 to 980
Data columns (total 13 columns):
Loan_ID                          981 non-null object
Gender                           981 non-null object
Married                          981 non-null object
Dependents                       981 non-null object
Education                        981 non-null object
Self_Employed                    981 non-null object
ApplicantIncome(in $)            981 non-null int64
CoapplicantIncome(in $)          981 non-null float64
LoanAmount(in thousand $)        981 non-null float64
Loan_Amount_Term(in months)      981 non-null float64
Credit_History                   981 non-null float64
Property_Area                    981 non-null object
Loan_Status                      981 non-null object
dtypes: float64(4), int64(1), object(8)
memory usage: 99.8+ KB
```

Type *Markdown* and LaTeX: α^2

In [14]:
```
Loan_df.head()
```

Out[14]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome(in $)	Coappl
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	

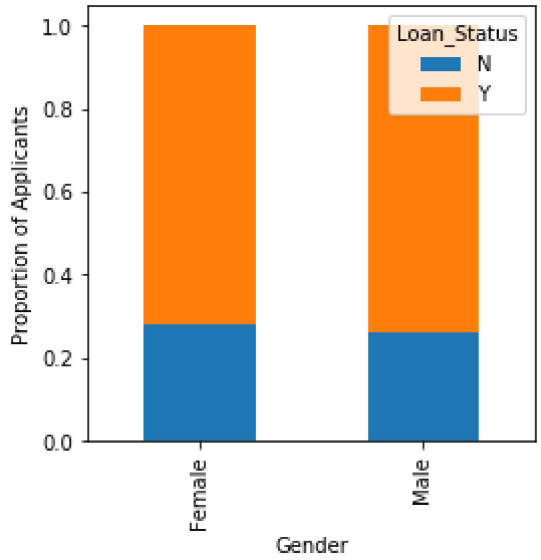
In [ ]:
```
```

Finding relation between variables and Loan_Status

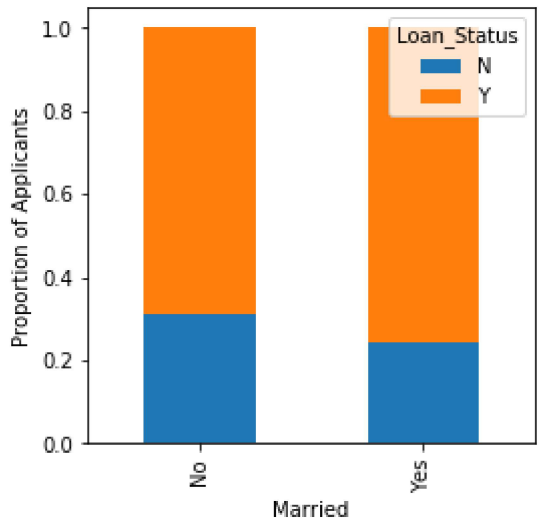
In [15]:
```
Gender=pd.crosstab(Loan_df['Gender'],Loan_df['Loan_Status'])
```

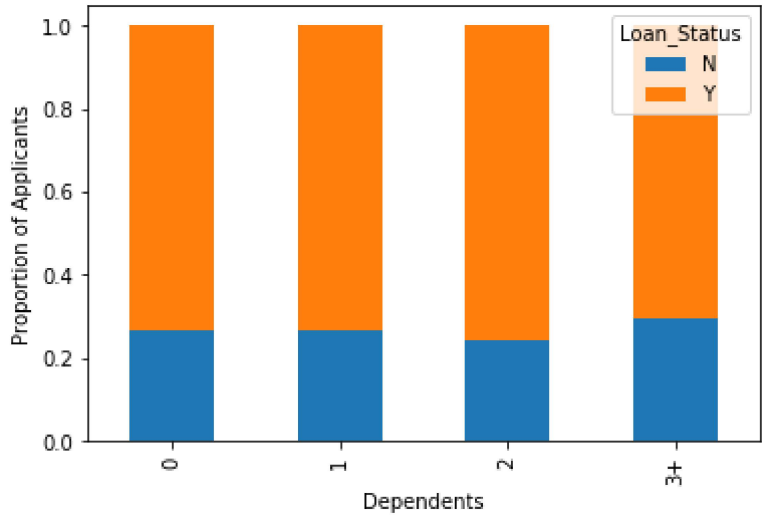
In [16]:
```
Gender.div(Gender.sum(1).astype(float), axis=0).plot(kind="bar", stacked=True, f
plt.ylabel('Proportion of Applicants')
```

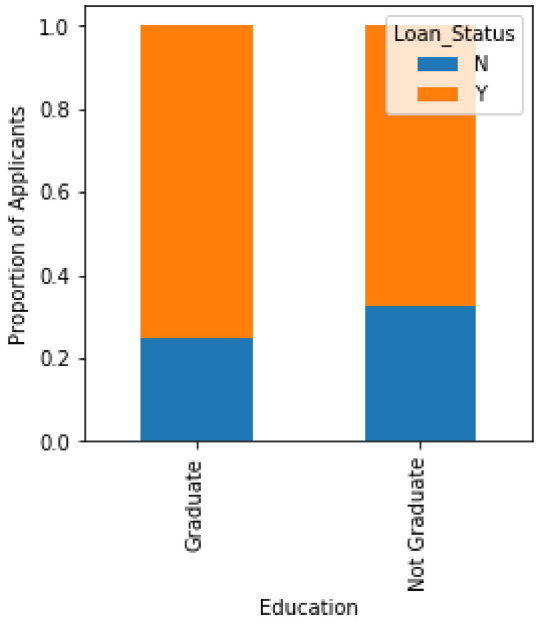
Out[16]:
```
Text(0, 0.5, 'Proportion of Applicants')
```

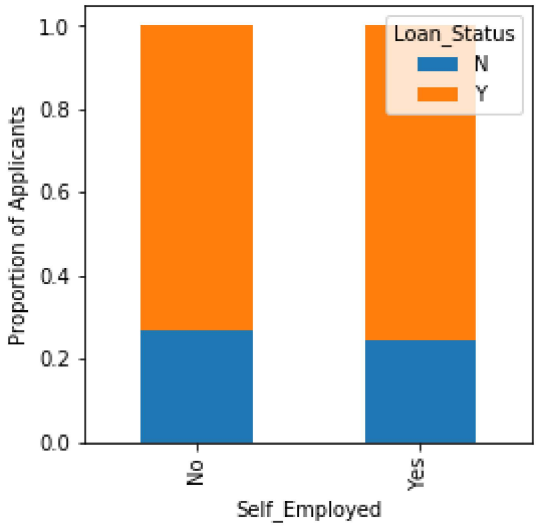


In [17]:
```
Married=pd.crosstab(Loan_df['Married'],Loan_df['Loan_Status'])
Dependents=pd.crosstab(Loan_df['Dependents'],Loan_df['Loan_Status'])
Education=pd.crosstab(Loan_df['Education'],Loan_df['Loan_Status'])
Self_Employed=pd.crosstab(Loan_df['Self_Employed'],Loan_df['Loan_Status'])
Married.div(Married.sum(1).astype(float), axis=0).plot(kind="bar", stacked=True,
plt.ylabel('Proportion of Applicants')
plt.show()
Dependents.div(Dependents.sum(1).astype(float), axis=0).plot(kind="bar", stacked
plt.ylabel('Proportion of Applicants')
plt.show()
Education.div(Education.sum(1).astype(float), axis=0).plot(kind="bar", stacked=T
plt.ylabel('Proportion of Applicants')
plt.show()
Self_Employed.div(Self_Employed.sum(1).astype(float), axis=0).plot(kind="bar", s
plt.ylabel('Proportion of Applicants')
plt.show()
```

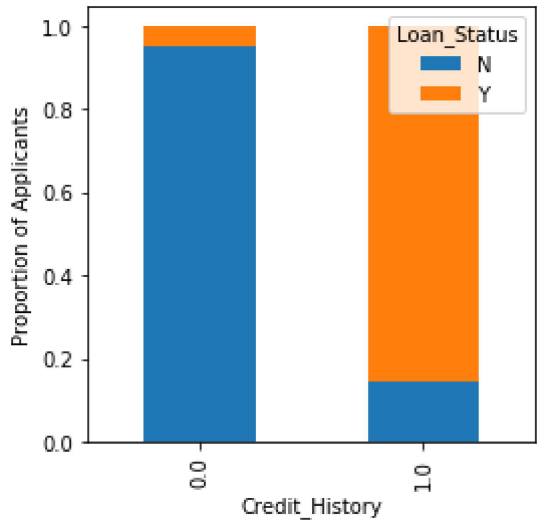


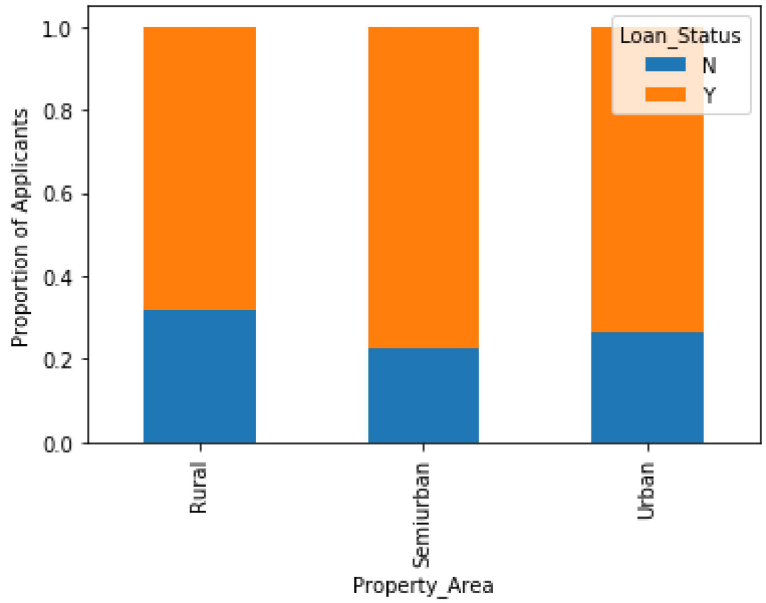



Loan_Status
N
Y
Proportion of Applicants
1.0
0.8
0.6
0.4
0.2
0.0
Graduate
Not Graduate
Education


Loan_Status
N
Y
Proportion of Applicants
1.0
0.8
0.6
0.4
0.2
0.0
No
Yes
Self_Employed

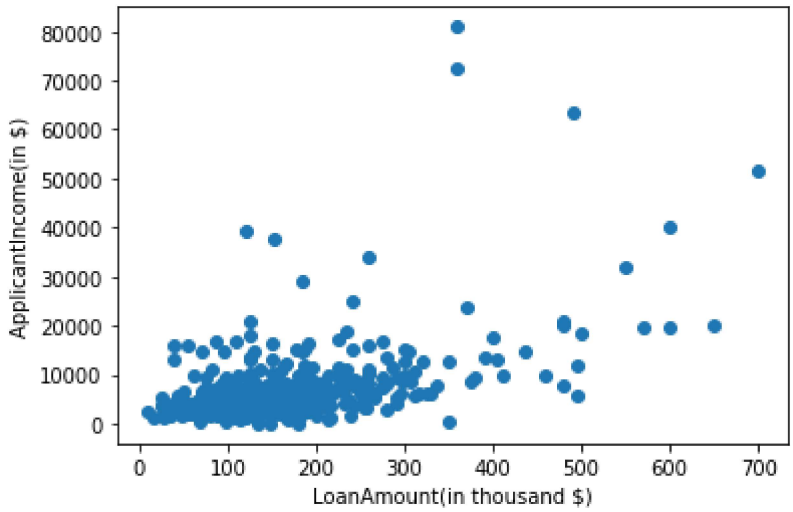
In [18]:
```
Credit_History=pd.crosstab(Loan_df['Credit_History'],Loan_df['Loan_Status'])
Property_Area=pd.crosstab(Loan_df['Property_Area'],Loan_df['Loan_Status'])
Credit_History.div(Credit_History.sum(1).astype(float), axis=0).plot(kind="bar",
plt.ylabel('Proportion of Applicants')
plt.show()
Property_Area.div(Property_Area.sum(1).astype(float), axis=0).plot(kind="bar", s
plt.ylabel('Proportion of Applicants')
plt.show()
```





In [19]:
```
plt.scatter(Loan_df['LoanAmount(in thousand $)'],Loan_df['ApplicantIncome(in $)'
plt.xlabel('LoanAmount(in thousand $)')
plt.ylabel('ApplicantIncome(in $)')
```

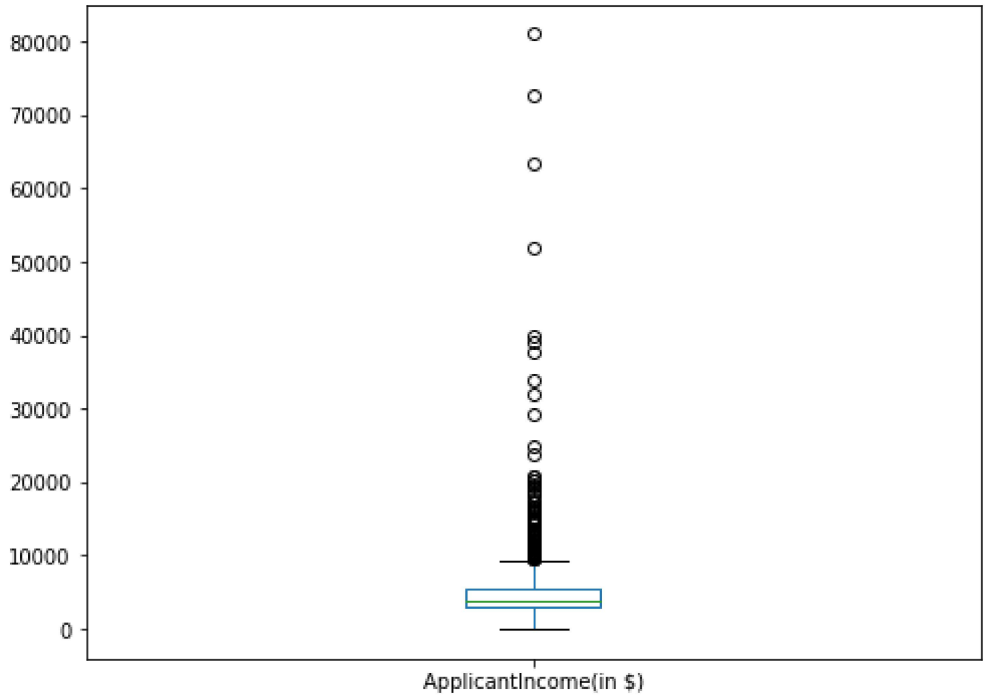
Out[19]:
```
Text(0, 0.5, 'ApplicantIncome(in $)')
```



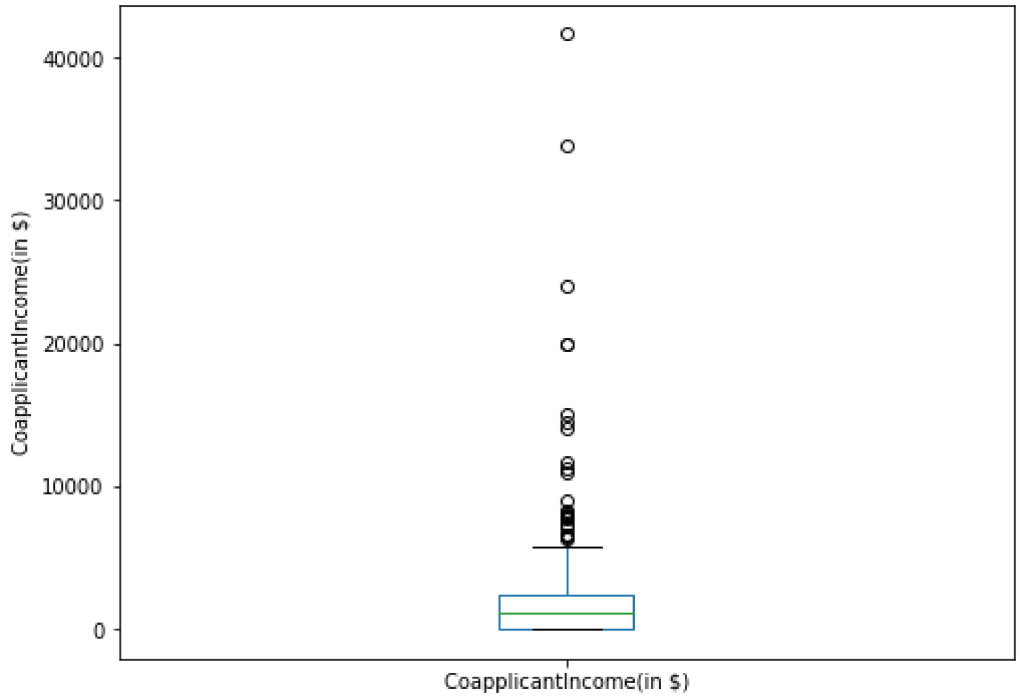
Outlier detection

Analysing numerical variables by plotting box plot

In [20]:
```
Loan_df['ApplicantIncome(in $)'].plot.box(figsize=(8,6))
plt.show()
```

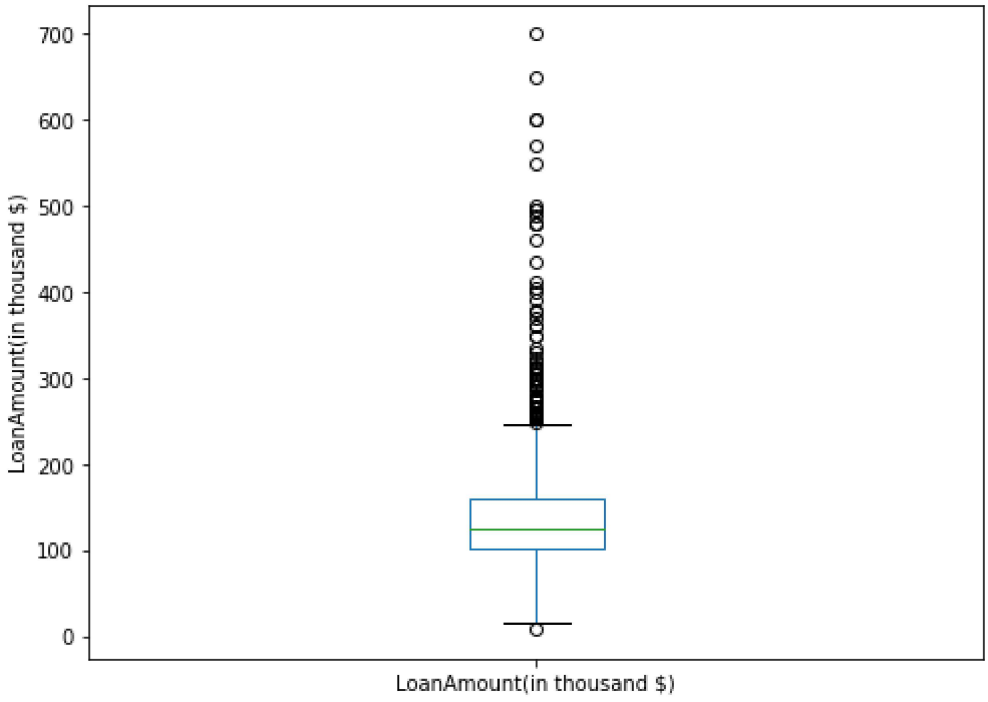


In [21]:
```
Loan_df['CoapplicantIncome(in $)'].plot.box(figsize=(8,6))
plt.ylabel('CoapplicantIncome(in $)')
plt.show()
```



There are less number of outliers in ApplicantIncome(in $)and Coapplicant Income(in$) which does'nt have significant impact on the prediction.

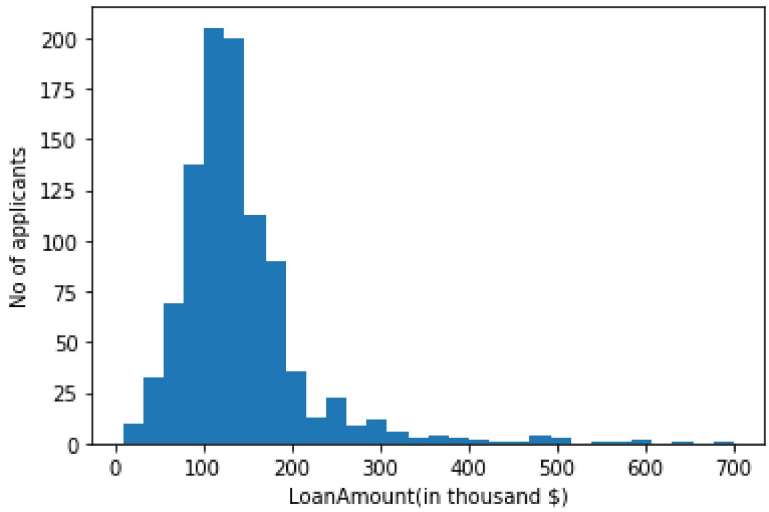
In [22]:
```
Loan_df['LoanAmount(in thousand $)'].plot.box(figsize=(8,6))
plt.ylabel('LoanAmount(in thousand $)')
plt.show()
```



There are more outliers in Loan Amount

In [23]:
```
plt.hist(Loan_df['LoanAmount(in thousand $)'],bins=30)
plt.xlabel('LoanAmount(in thousand $)')
plt.ylabel('No of applicants')
```

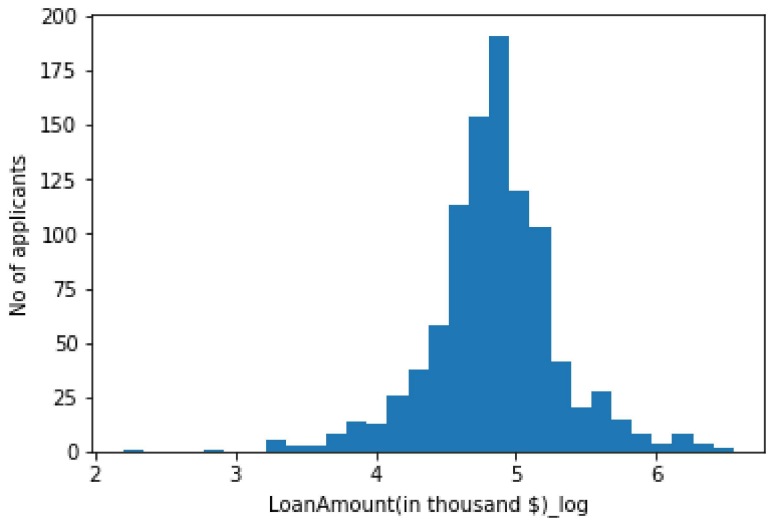
Out[23]: Text(0, 0.5, 'No of applicants')



Due to these outliers bulk of the data in the loan amount is at the left
Performing Log Transformation to reduce the impact of outliers

In [24]:
```
Loan_df['LoanAmount(in thousand $)_log'] = np.log(Loan_df['LoanAmount(in thousan
plt.hist(Loan_df['LoanAmount(in thousand $)_log'],bins=30)
plt.xlabel('LoanAmount(in thousand $)_log')
plt.ylabel('No of applicants')
```

Out[24]: Text(0, 0.5, 'No of applicants')



In [25]:
```
Loan_df['EMI']=(Loan_df['LoanAmount(in thousand $)']/Loan_df['Loan_Amount_Term(i
```

In [26]:
```
Loan_df.columns
```

Out[26]:
```
Index(['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education',
       'Self_Employed', 'ApplicantIncome(in $)', 'CoapplicantIncome(in $)',
       'LoanAmount(in thousand $)', 'Loan_Amount_Term(in months)',
       'Credit_History', 'Property_Area', 'Loan_Status',
       'LoanAmount(in thousand $)_log', 'EMI'],
      dtype='object')
```

Preparing data

As Loan_ID is not necessary for predicting Loan Approval Status,I'm dropping the column

In [27]:
```
Loan_df=Loan_df.drop('Loan_ID',axis=1)
```

In [28]:
```
Loan_df.columns
```

Out[28]:
```
Index(['Gender', 'Married', 'Dependents', 'Education', 'Self_Employed',
       'ApplicantIncome(in $)', 'CoapplicantIncome(in $)',
       'LoanAmount(in thousand $)', 'Loan_Amount_Term(in months)',
       'Credit_History', 'Property_Area', 'Loan_Status',
       'LoanAmount(in thousand $)_log', 'EMI'],
      dtype='object')
```

In [29]:
```
nths)','Credit_History','Property_Area','ApplicantIncome(in $)', 'CoapplicantIncor
```

In [30]:
```
X=Loan_df[features].copy()
```

In [31]:
```
y=Loan_df[['Loan_Status']].copy()
```

In [32]:
```
X.columns
```

Out[32]:
```
Index(['Married', 'Education', 'Loan_Amount_Term(in months)', 'Credit_History',
       'Property_Area', 'ApplicantIncome(in $)', 'CoapplicantIncome(in $)',
       'LoanAmount(in thousand $)_log', 'EMI'],
      dtype='object')
```

In [33]:
```
y.columns
```

Out[33]:
```
Index(['Loan_Status'], dtype='object')
```

In [34]:
```
nums = {"Gender":     {"Male": 0, "Female": 1},
        "Married": {"Yes": 1, "No": 0},
        "Education":{"Graduate":1,"Not Graduate":0},
        "Dependents":{"0":0,"1":1,"2":2,"3+":3},
        "Self_Employed":{"Yes":1,"No":0},
        "Property_Area":{"Urban":0,"Rural":1,"Semiurban":2}}
```

In [35]:
```
X.replace(nums, inplace=True)
```

In [36]:
```
X.head()
```

Out[36]:

	Married	Education	Loan_Amount_Term(in months)	Credit_History	Property_Area	ApplicantIncome(in $)	C
0	0	1	360.0	1.0	0	5849	
1	1	1	360.0	1.0	1	4583	
2	1	1	360.0	1.0	0	3000	
3	1	0	360.0	1.0	0	2583	
4	0	1	360.0	1.0	0	6000	

In [ ]:
```
```

In [37]:
```
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
```

Splitting train and test data

In [38]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_
```

In [39]:
```
type(X_train)
```

Out[39]:
```
pandas.core.frame.DataFrame
```

In [40]:
```
type(X_test)
```

Out[40]:
```
pandas.core.frame.DataFrame
```

In [41]:
```
X_test
```

Out[41]:

	Married	Education	Loan_Amount_Term(in months)	Credit_History	Property_Area	ApplicantIncome(in $)
514	0	1	360.0	1.0	1	5815
887	1	1	360.0	0.0	2	5667
303	1	1	360.0	1.0	0	1625
56	1	1	360.0	1.0	2	2132
408	1	1	300.0	0.0	2	8300
...	...	...	...	...	...	...
799	1	0	360.0	1.0	2	4483
689	0	1	360.0	1.0	0	3583
766	0	1	360.0	1.0	1	5000
638	1	1	360.0	1.0	0	5400
86	1	0	360.0	1.0	2	3333

324 rows × 9 columns

In [42]:
```
type(y_train)
```

Out[42]:
```
pandas.core.frame.DataFrame
```

In [43]:
```
type(y_test)
```

Out[43]:
```
pandas.core.frame.DataFrame
```

In [44]:
```
X_train.head()
```

Out[44]:

	Married	Education	Loan_Amount_Term(in months)	Credit_History	Property_Area	ApplicantIncome(in $)
950	0	0	360.0	1.0	0	3015
469	1	1	360.0	1.0	0	4333
478	1	1	360.0	1.0	2	16667
444	1	1	300.0	1.0	1	7333
5	1	1	360.0	1.0	0	5417

In [45]:
```
y_train.head()
```

Out[45]:

	Loan_Status
950	Y
469	N
478	Y
444	Y
5	Y

In [46]:
```
y_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 657 entries, 950 to 665
Data columns (total 1 columns):
Loan_Status    657 non-null object
dtypes: object(1)
memory usage: 10.3+ KB
```

Decision Tree Classifier

In [47]:
```
Decisiontree_classifier = DecisionTreeClassifier(max_leaf_nodes=10, random_state
Decisiontree_classifier.fit(X_train, y_train)
```

Out[47]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                       max_features=None, max_leaf_nodes=10,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=0, splitter='best')
```

In [48]:
```
predictions = Decisiontree_classifier.predict(X_test)
```

In [49]:
```
predictions[:10]
```

Out[49]:
```
array(['Y', 'N', 'Y', 'Y', 'N', 'Y', 'N', 'N', 'N', 'Y'], dtype=object)
```

In [50]:
```
accuracy_score(y_true = y_test, y_pred = predictions)
```

Out[50]:
```
0.8827160493827161
```

KNN Classifier

In [51]:
```
from sklearn.neighbors import KNeighborsClassifier
neigh = KNeighborsClassifier(n_neighbors=5)
```

In [52]:
```
neigh.fit(X_train,y_train )
```

```
C:\Users\Divya\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  """Entry point for launching an IPython kernel.
```

Out[52]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

In [53]:
```
predictions1 = neigh.predict(X_test)
```

In [54]:
```
accuracy_score(y_true = y_test, y_pred = predictions1)
```

Out[54]:
```
0.6944444444444444
```

Naive Bayes Classifier

In [55]:
```
from sklearn.naive_bayes import GaussianNB
```

In [56]:
```
model = GaussianNB()
```

In [57]:
```
model.fit(X_train,y_train)
```

```
C:\Users\Divya\Anaconda3\lib\site-packages\sklearn\utils\validation.py:724: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

Out[57]:
```
GaussianNB(priors=None, var_smoothing=1e-09)
```

In [58]:
```
predicted= model.predict(X_test)
```

In [59]:
```
accuracy_score(y_true = y_test, y_pred = predicted)
```

Out[59]:
```
0.8611111111111112
```

In [ ]:

In [ ]: