In [1]:
```python
# Importing required Libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
```

In [2]:
```python
A = pd.read_csv("C:/Work/DATA_SCI-ANA/Datasets/Loan_Prediction/training_set.csv")
```

In [3]:
```python
from warnings import filterwarnings
filterwarnings("ignore")
```

Dataset at a glance

In [4]:
```python
A.shape
```

Out[4]:
```
(614, 13)
```

In [5]:
```python
A.head()
```

Out[5]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	Lo
0	LP001002	Male	No	0	Graduate	No	5849.0	0.0	
1	LP001003	Male	Yes	1	Graduate	No	NaN	1508.0	
2	LP001005	Male	Yes	0	Graduate	Yes	3000.0	0.0	
3	LP001006	Male	Yes	0	Not Graduate	No	2583.0	2358.0	
4	LP001008	Male	No	0	Graduate	No	6000.0	0.0	

In [6]:
```python
A.describe()
```

Out[6]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	612.000000	613.000000	592.000000	600.00000	564.000000
mean	5405.540850	1620.888940	146.412162	342.00000	0.842199
std	6118.914057	2928.624748	85.587325	65.12041	0.364878
min	150.000000	0.000000	9.000000	12.00000	0.000000
25%	2875.750000	0.000000	100.000000	360.00000	1.000000
50%	3806.000000	1167.000000	128.000000	360.00000	1.000000
75%	5803.750000	2302.000000	168.000000	360.00000	1.000000
max	81000.000000	41667.000000	700.000000	480.00000	1.000000

In [7]:
```python
A.isna().sum()
```

```
Out[7]: Loan_ID              0
Gender              15
Married              3
Dependents          15
Education            1
Self_Employed       32
ApplicantIncome      2
CoapplicantIncome    1
LoanAmount          22
Loan_Amount_Term    14
Credit_History      50
Property_Area        0
Loan_Status          0
dtype: int64
```

```
In [8]: A.Loan_Status.value_counts()
```

```
Out[8]: Y    422
N    192
Name: Loan_Status, dtype: int64
```

Exploratory Data Analysis

1.Treating missing values

```
In [9]: for i in A.columns:
    if (A[i].dtypes == 'object'):
        x = A[i].mode()[0]
        A[i] = A[i].fillna(x)
    else:
        x = A[i].mean()
        A[i] = A[i].fillna(x)
```

```
In [10]: # Checking for missing values

A.isna().sum()
```

```
Out[10]: 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
```

2. Diffrentiating Data on basis of data types

```
In [11]: cat = []
con = []


for i in A.columns:
    if (A[i].dtypes == 'object'):
        cat.append(i)
    else:
Loading [MathJax]/extensions/Safe.js ppend(i)
```

In [12]:
```
#categorical columns

cat
```

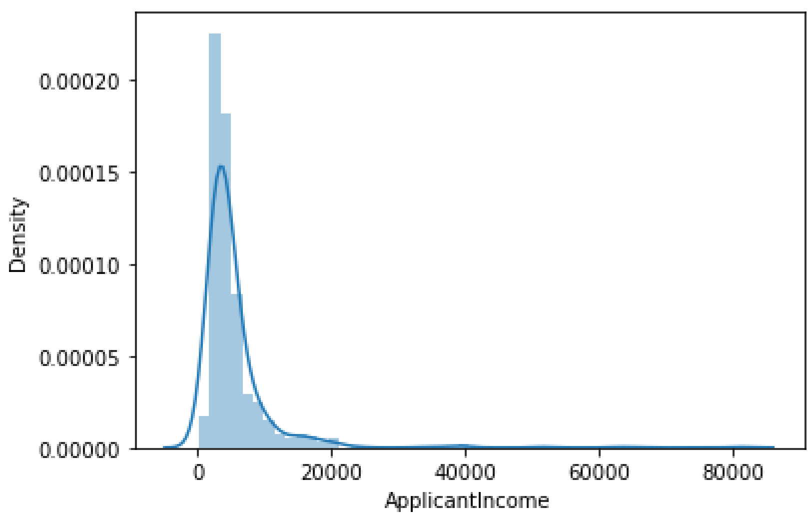
Out[12]:
```
['Loan_ID',
 'Gender',
 'Married',
 'Dependents',
 'Education',
 'Self_Employed',
 'Property_Area',
 'Loan_Status']
```

In [13]:
```
#continuous columns

con
```

Out[13]:
```
['ApplicantIncome',
 'CoapplicantIncome',
 'LoanAmount',
 'Loan_Amount_Term',
 'Credit_History']
```

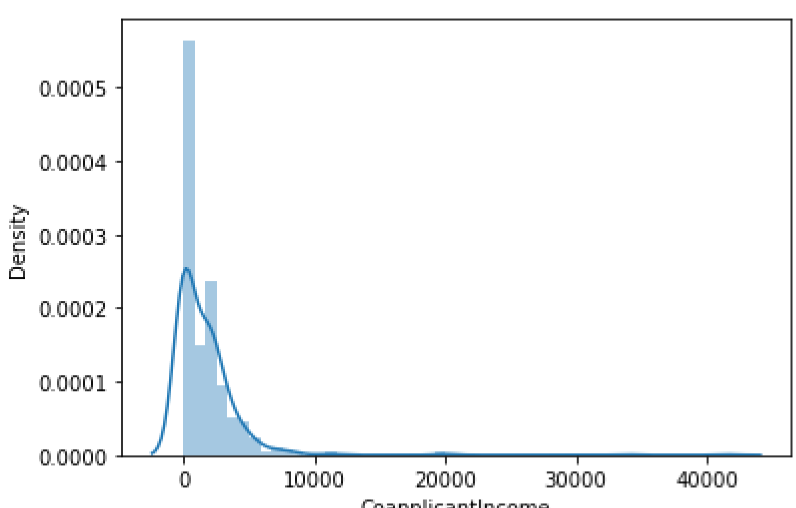
In [14]:
```
sb.distplot(A['ApplicantIncome'])
```

Out[14]:
```
<AxesSubplot:xlabel='ApplicantIncome', ylabel='Density'>
```

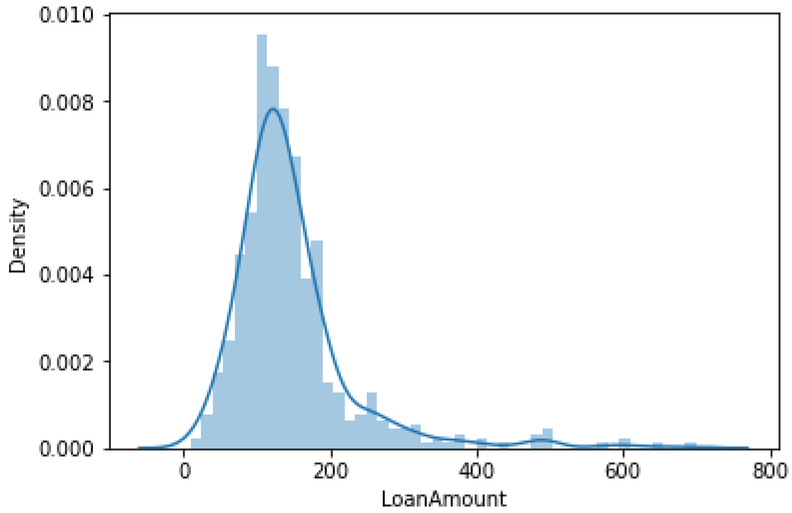


In [15]:
```
sb.distplot(A['CoapplicantIncome'])
```

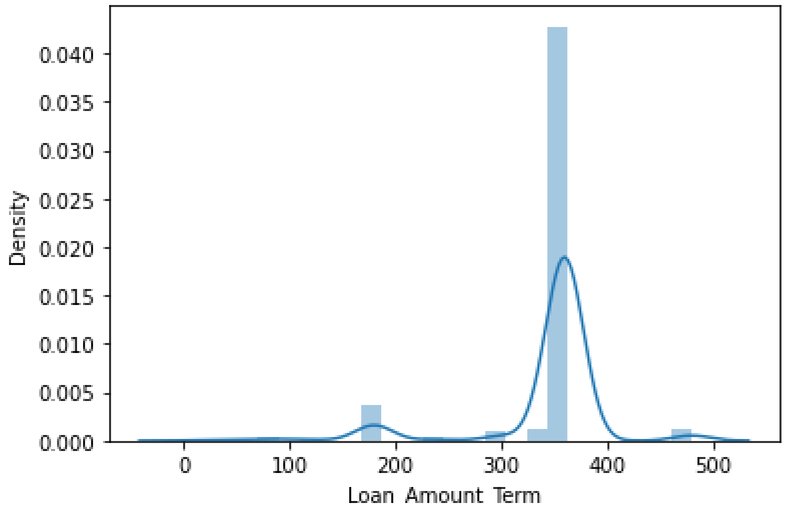
Out[15]:
```
<AxesSubplot:xlabel='CoapplicantIncome', ylabel='Density'>
```



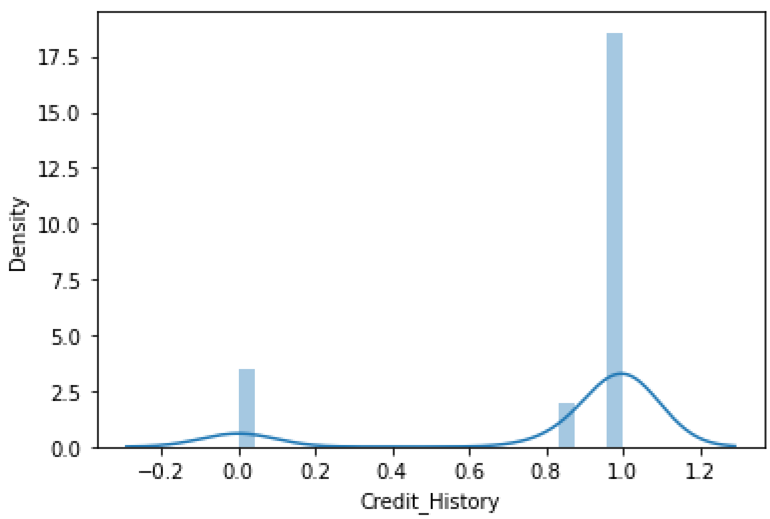
In [16]:
```python
sb.distplot(A['LoanAmount'])
```
Out[16]: <AxesSubplot:xlabel='LoanAmount', ylabel='Density'>



In [17]:
```python
sb.distplot(A['Loan_Amount_Term'])
```
Out[17]: <AxesSubplot:xlabel='Loan_Amount_Term', ylabel='Density'>



In [18]:
```python
sb.distplot(A['Credit_History'])
```
Out[18]: <AxesSubplot:xlabel='Credit_History', ylabel='Density'>



Loading [MathJax]/extensions/Safe.js sentation above of continuous variables confirms outliers, there are 2 ways to solve:

1. Standardisation
2. Yeo-Jhonson Method

3) Treating Outliers

method{‘yeo-johnson’, ‘box-cox’}, default=’yeo-johnson’

In [19]:
```
from sklearn.preprocessing import PowerTransformer
pt = PowerTransformer()
conti = pt.fit_transform(A[con])
```

In [20]:
```
conti = pd.DataFrame(conti,columns = con)
```

In [21]:
```
conti
```

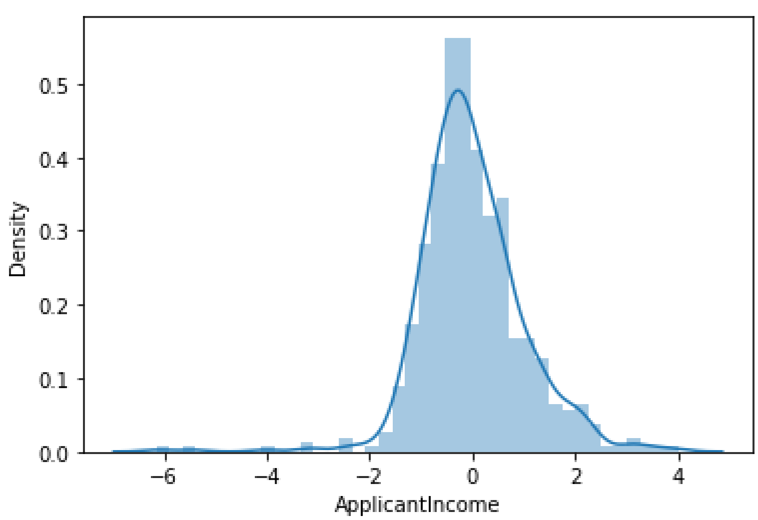
Out[21]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
0	0.543507	-1.102830	0.235387	0.185905	0.514063
1	0.423838	0.750689	-0.037659	0.185905	0.514063
2	-0.500567	-1.102830	-1.346305	0.185905	0.514063
3	-0.744401	0.891798	-0.167886	0.185905	0.514063
4	0.581990	-1.102830	0.158610	0.185905	0.514063
...	...	...	...	...	...
609	-0.555480	-1.102830	-1.205100	0.185905	0.514063
610	-0.001034	-1.102830	-2.293317	-2.308570	0.514063
611	1.022648	0.208701	1.372409	0.185905	0.514063
612	0.930911	-1.102830	0.738924	0.185905	0.514063
613	0.170198	-1.102830	0.039936	0.185905	-2.247196

614 rows × 5 columns

In [22]:
```
sb.distplot(conti['ApplicantIncome'])
```

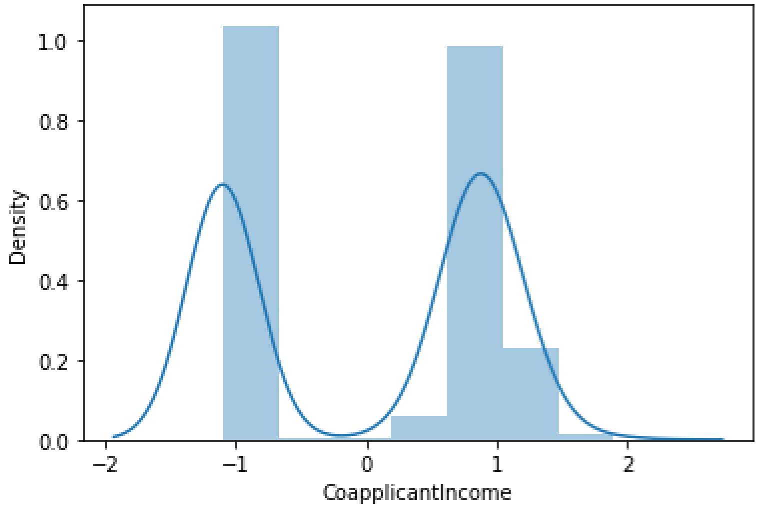
Out[22]: <AxesSubplot:xlabel='ApplicantIncome', ylabel='Density'>



In [23]:
```
sb.distplot(conti['CoapplicantIncome'])
```

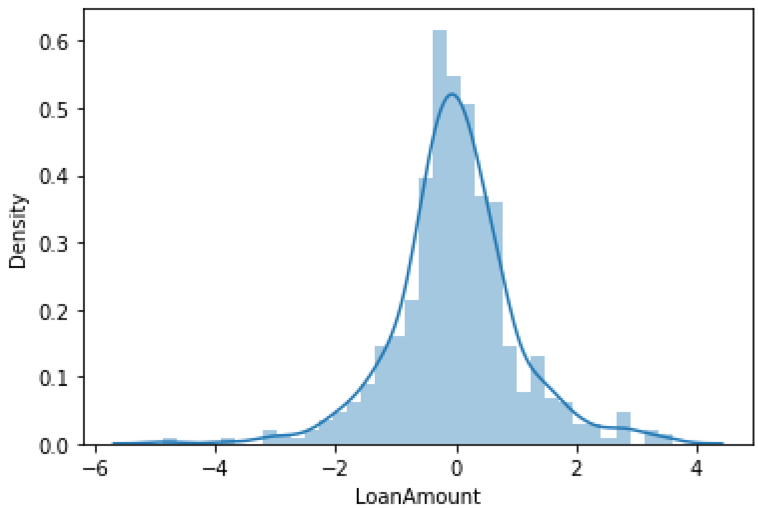
Loading [MathJax]/extensions/Safe.js

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



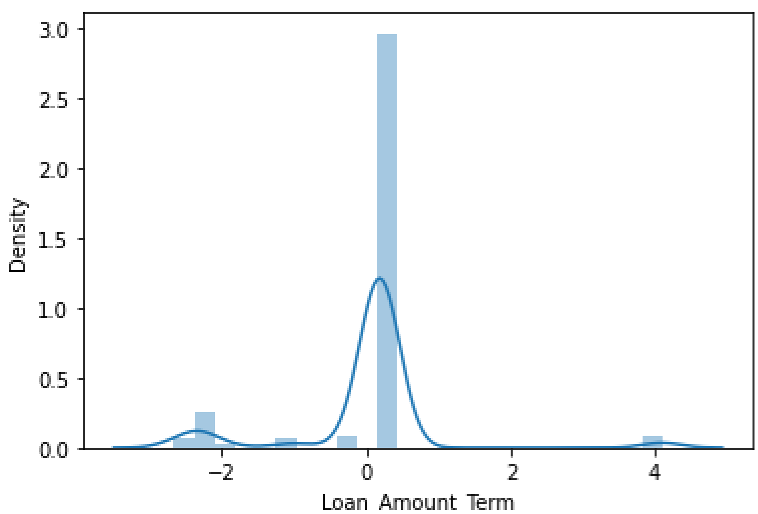
In [24]:
```
sb.distplot(conti['LoanAmount'])
```

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



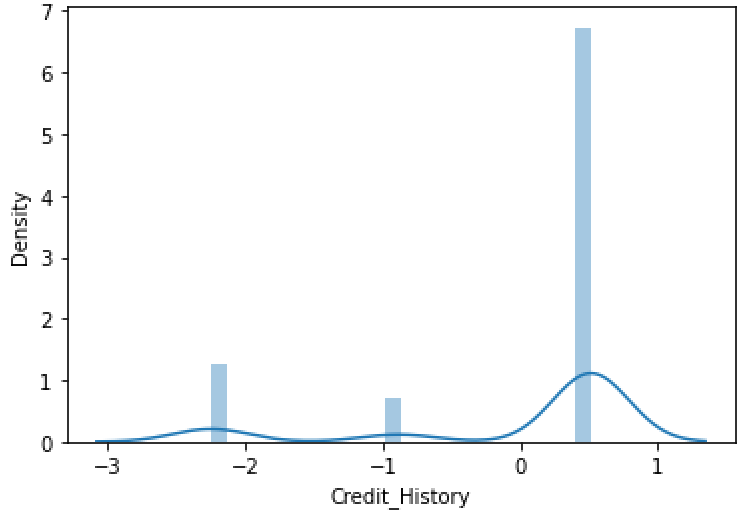
In [25]:
```
sb.distplot(conti['Loan_Amount_Term'])
```

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



In [26]:
```
sb.distplot(conti['Credit_History'])
```

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



In [27]:
```
# Most of the columns are cured for outliers
```

4) One Hot Encoding Categorical Variables

In [28]:
```
cate = A[cat]
```

In [29]:
```
cate = cate.drop(columns = ['Loan_ID']) #dropped discrete column
```

In [30]:
```
cate = pd.get_dummies(cate)
```

In [31]:
```
cate
```

Out[31]:

	Gender_Female	Gender_Male	Married_No	Married_Yes	Dependents_0	Dependents_1	Dependents_2	Dep
0	0	1	1	0	1	0	0	
1	0	1	0	1	0	1	0	
2	0	1	0	1	1	0	0	
3	0	1	0	1	1	0	0	
4	0	1	1	0	1	0	0	
...	...	...	...	...	...	...	...	
609	1	0	1	0	1	0	0	
610	0	1	0	1	0	0	0	
611	0	1	0	1	0	1	0	
612	0	1	0	1	0	0	1	
613	1	0	1	0	1	0	0	

614 rows × 17 columns

In [32]:
```
df = conti.join(cate)
```

In [33]:
```
df
```

Loading [MathJax]/extensions/Safe.js

Out[33]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Gender_Female	Ge
0	0.543507	-1.102830	0.235387	0.185905	0.514063	0	
1	0.423838	0.750689	-0.037659	0.185905	0.514063	0	
2	-0.500567	-1.102830	-1.346305	0.185905	0.514063	0	
3	-0.744401	0.891798	-0.167886	0.185905	0.514063	0	
4	0.581990	-1.102830	0.158610	0.185905	0.514063	0	
...	...	...	...	...	...	...	
609	-0.555480	-1.102830	-1.205100	0.185905	0.514063	1	
610	-0.001034	-1.102830	-2.293317	-2.308570	0.514063	0	
611	1.022648	0.208701	1.372409	0.185905	0.514063	0	
612	0.930911	-1.102830	0.738924	0.185905	0.514063	0	
613	0.170198	-1.102830	0.039936	0.185905	-2.247196	1	

614 rows × 22 columns

5) Checking the relation {X ~ Y}

- Y = Loan_Status is the categorical variable
- If Y is categorical and X is continuous we use boxplot or ANOVA
- If Y is categorical and X is categorical we use countplot with hue, cross tabulation, Chi_Square Test

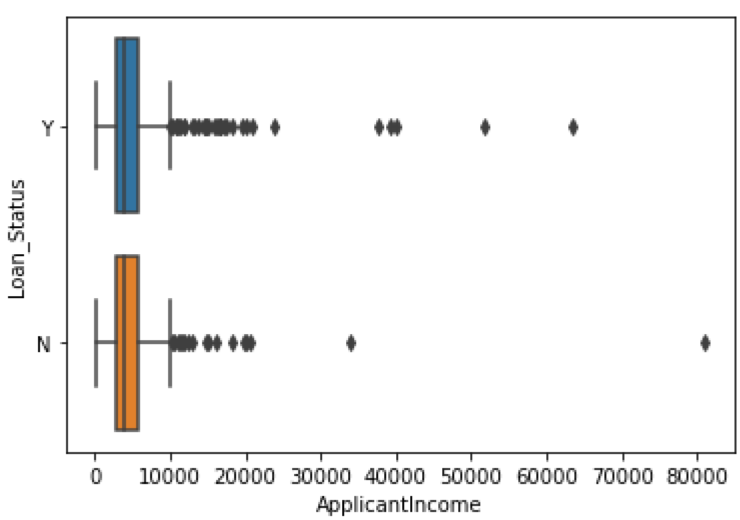
a. First we will go with continuous

In [34]:
```
con
```

Out[34]:
```
['ApplicantIncome',
 'CoapplicantIncome',
 'LoanAmount',
 'Loan_Amount_Term',
 'Credit_History']
```

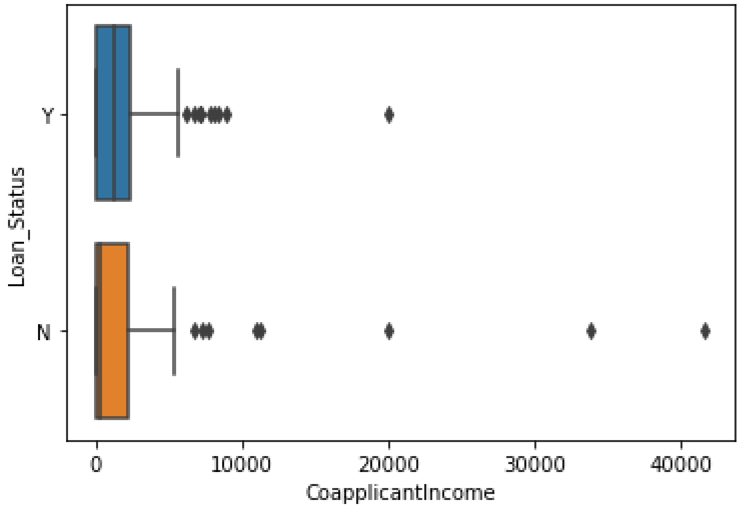
In [35]:
```
sb.boxplot(A['ApplicantIncome'],A['Loan_Status'])
```

Out[35]:
```
<AxesSubplot:xlabel='ApplicantIncome', ylabel='Loan_Status'>
```



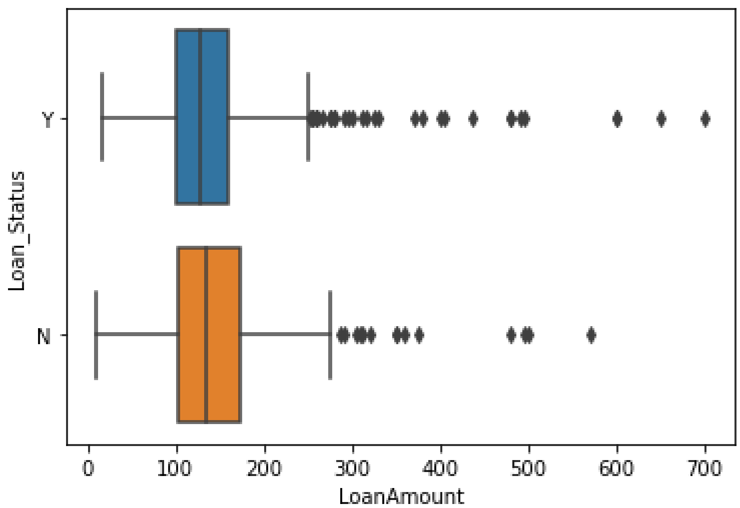
In [36]:
```
sb.boxplot(A['CoapplicantIncome'],A['Loan_Status'])
```

Out[36]: <AxesSubplot:xlabel='CoapplicantIncome', ylabel='Loan_Status'>



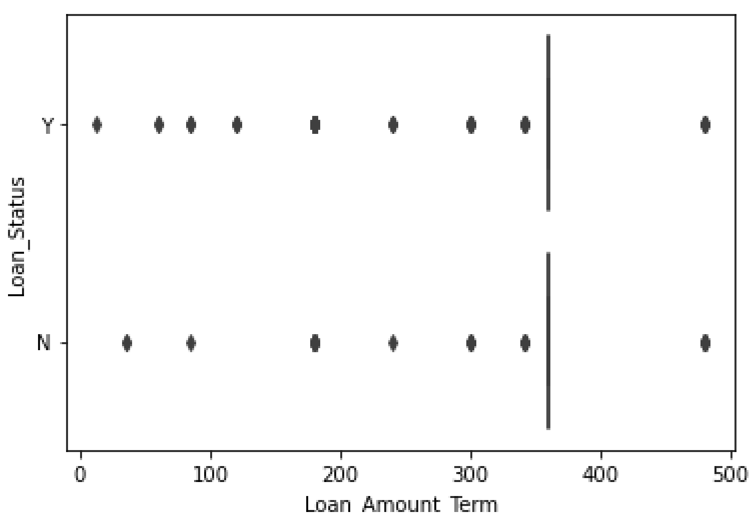
In [37]:
```
sb.boxplot(A['LoanAmount'],A['Loan_Status'])
```

Out[37]: <AxesSubplot:xlabel='LoanAmount', ylabel='Loan_Status'>



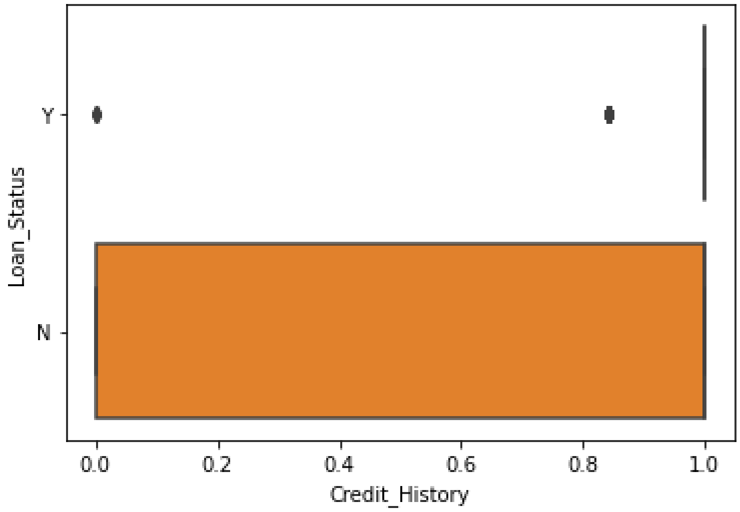
In [38]:
```
sb.boxplot(A['Loan_Amount_Term'],A['Loan_Status'])
```

Out[38]: <AxesSubplot:xlabel='Loan_Amount_Term', ylabel='Loan_Status'>



In [39]:
```
sb.boxplot(A['Credit_History'],A['Loan_Status'])
```

Out[39]: <AxesSubplot:xlabel='Credit_History', ylabel='Loan_Status'>



Checking the relation between Y and X(continuous) using pvalues

In [40]:
```
from scipy.stats import chi2_contingency
for i in conti:
    Q = pd.crosstab(A.Loan_Status,conti[i])
    a,b,c,d = chi2_contingency(Q)
    print(i,"~~",b)
```

```
ApplicantIncome ~~ 0.445970304541267
CoapplicantIncome ~~ 0.5702980314497069
LoanAmount ~~ 0.4372624752205558
Loan_Amount_Term ~~ 0.1379445122345725
Credit_History ~~ 7.926164541543543e-40
```

According to pvalue we see only credit history has best relation with Loan_Status

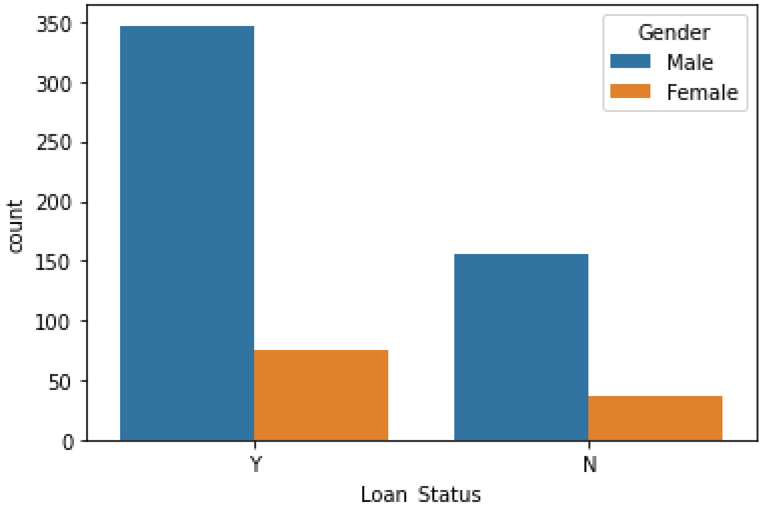
In [41]:
```
for i in A[cat]:
    Q = pd.crosstab(A.Loan_Status,A[i])
    a,b,c,d = chi2_contingency(Q)
    print(i,'~~',a)
```

```
Loan_ID ~~ 614.0000000000002
Gender ~~ 0.11087854691241235
Married ~~ 4.73187557933362
Dependents ~~ 3.1513990012324227
Education ~~ 4.091490413303621
Self_Employed ~~ 0.0
Property_Area ~~ 12.297623130485677
Loan_Status ~~ 609.355921937585
```

In [42]:
```
sb.countplot(A.Loan_Status,hue = A.Gender)
```

Out[42]:
```
<AxesSubplot:xlabel='Loan_Status', ylabel='count'>
```

Loading [MathJax]/extensions/Safe.js

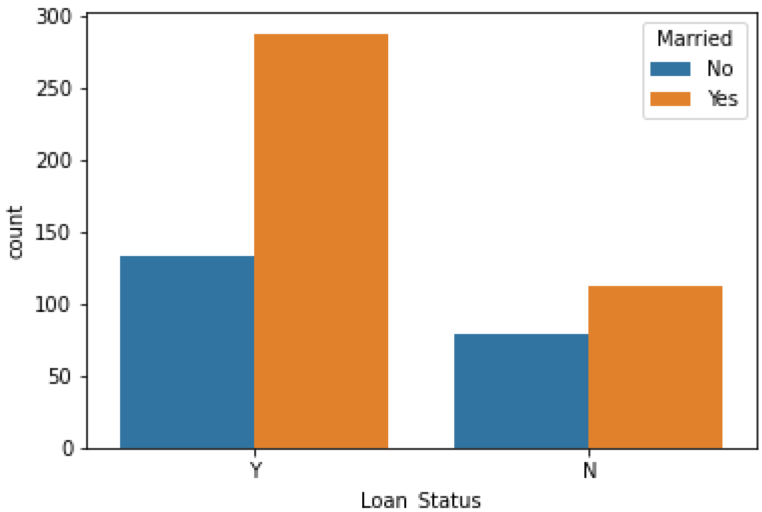


In [43]:
```
A['Gender'].value_counts()
```

Out[43]:
```
Male      502
Female    112
Name: Gender, dtype: int64
```

In [44]:
```
sb.countplot(A.Loan_Status,hue = A.Married)
```

Out[44]:
```
<AxesSubplot:xlabel='Loan_Status', ylabel='count'>
```



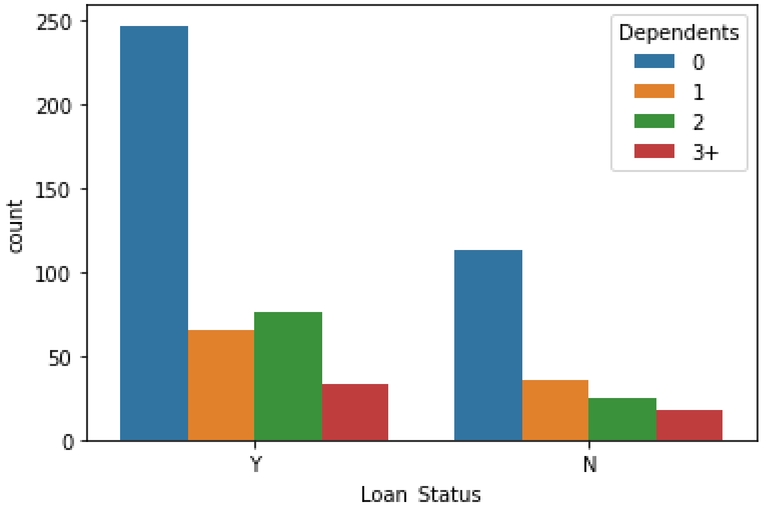
In [45]:
```
A['Married'].value_counts()
```

Out[45]:
```
Yes    401
No     213
Name: Married, dtype: int64
```

In [46]:
```
sb.countplot(A.Loan_Status,hue = A.Dependents)
```

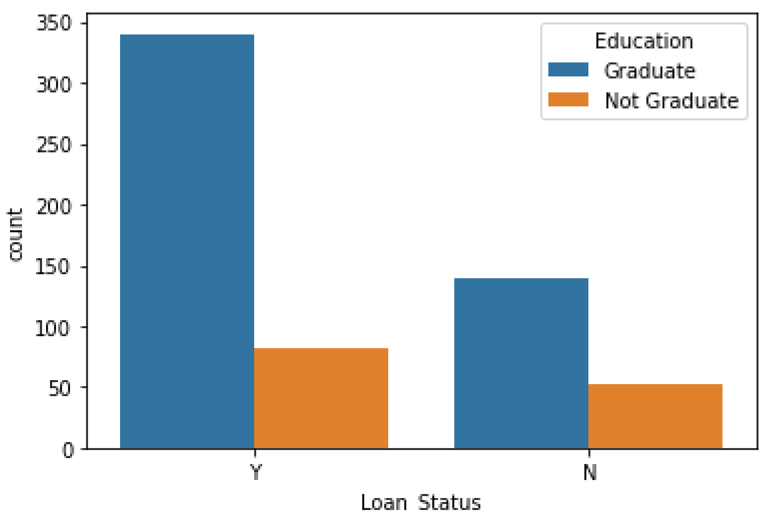
Out[46]:
```
<AxesSubplot:xlabel='Loan_Status', ylabel='count'>
```

Loading [MathJax]/extensions/Safe.js



In [47]:
```
A['Dependents'].value_counts()
```
Out[47]:
```
0     360
1     102
2     101
3+     51
Name: Dependents, dtype: int64
```

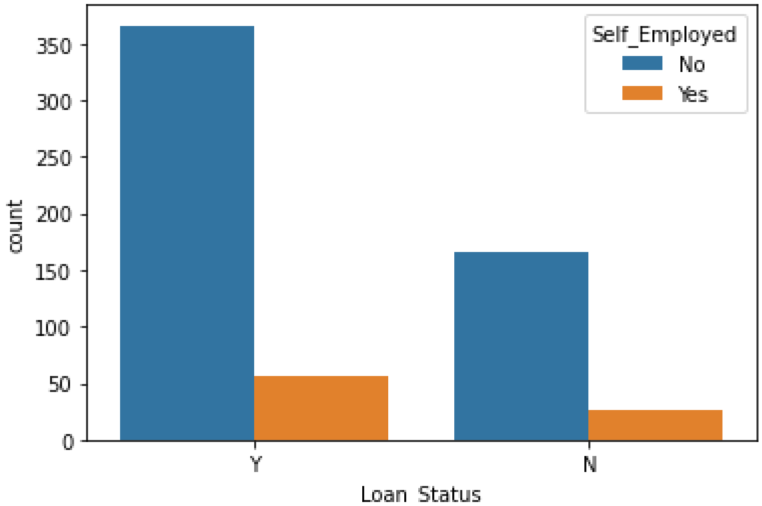
In [48]:
```
sb.countplot(A.Loan_Status,hue = A.Education)
```
Out[48]:
```
<AxesSubplot:xlabel='Loan_Status', ylabel='count'>
```



In [49]:
```
A['Education'].value_counts()
```
Out[49]:
```
Graduate        480
Not Graduate    134
Name: Education, dtype: int64
```

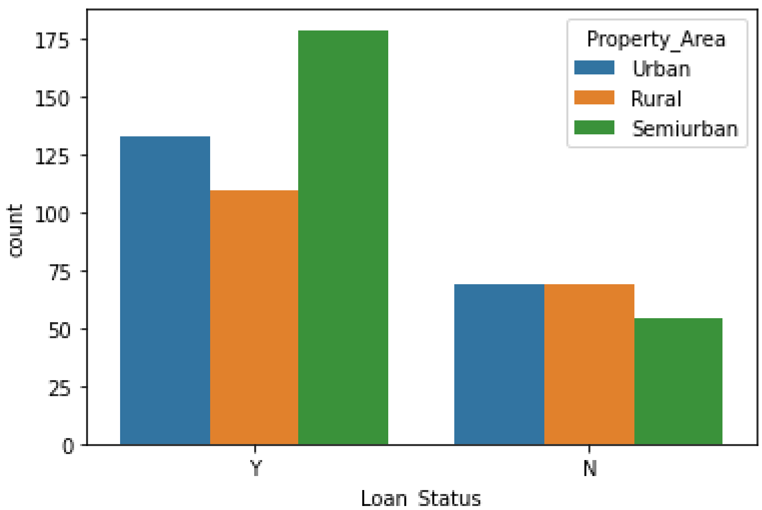
In [50]:
```
sb.countplot(A.Loan_Status,hue = A.Self_Employed)
```
Out[50]:
```
<AxesSubplot:xlabel='Loan_Status', ylabel='count'>
```

Loading [MathJax]/extensions/Safe.js



In [51]:
```
sb.countplot(A.Loan_Status,hue = A.Property_Area)
```

Out[51]: <AxesSubplot:xlabel='Loan_Status', ylabel='count'>



Observations :

1) Training set has 614 observations and 13 characteristics

2) Highest number of missing value is seen in column named credit history which is also a important column to decide weather loan should be given on not

3) There is difference between mean and 50th percentile also there is large difference between 75th percentile and max values indicating we have outliers in dataset

4) Number of "Yes" is far greater than number of "No" in target variable

5) Majority of columns are biased which might create varience

6) Gender and Self_Employed columns dose not have good relation with Loan_Status

In [52]:
```
df #dataset ready to use
```

Out[52]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Gender_Female	Ge
0	0.543507	-1.102830	0.235387	0.185905	0.514063	0	
1	0.423838	0.750689	-0.037659	0.185905	0.514063	0	
2	-0.500567	-1.102830	-1.346305	0.185905	0.514063	0	
3	-0.744401	0.891798	-0.167886	0.185905	0.514063	0	
4	0.581990	-1.102830	0.158610	0.185905	0.514063	0	
...	...	...	...	...	...	...	
609	-0.555480	-1.102830	-1.205100	0.185905	0.514063	1	
610	-0.001034	-1.102830	-2.293317	-2.308570	0.514063	0	
611	1.022648	0.208701	1.372409	0.185905	0.514063	0	
612	0.930911	-1.102830	0.738924	0.185905	0.514063	0	
613	0.170198	-1.102830	0.039936	0.185905	-2.247196	1	

614 rows × 22 columns

Making dataset ready for models

In [53]:
```
X = df.drop(columns = ['Loan_Status_N','Loan_Status_Y'])
```

In [54]:
```
Y = A[['Loan_Status']]
```

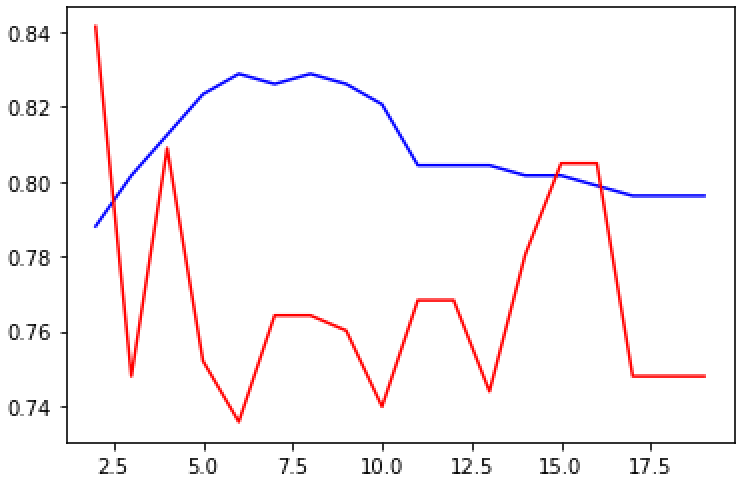
In [55]:
```
# splitting the dataset for training and testing

from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest = train_test_split(X,Y,random_state= 21, test_size= 0.4)
```

Model -1] Decision Tree Classifier

In [56]:
```
tr = []
ts = []
for i in range(2,20,1):
    from sklearn.tree import DecisionTreeClassifier
    dtc = DecisionTreeClassifier(criterion="entropy",random_state=21,max_depth=i,min_sam

    model = dtc.fit(xtrain,ytrain)
    pred_tr = model.predict(xtrain)
    pred_ts = model.predict(xtest)

    #training error
    from sklearn.metrics import accuracy_score,confusion_matrix
    tr_acc = accuracy_score(ytrain,pred_tr)
    tr_con = confusion_matrix(ytrain,pred_tr)

    tr.append(tr_acc)

    #testing error
    ts_acc = accuracy_score(ytest,pred_ts)
    ts_con = confusion_matrix(ytest,pred_ts)

    ts.append(ts_acc)
```

In [57]:
```python
import matplotlib.pyplot as plt
plt.plot(range(2,20,1),tr,c="blue")
plt.plot(range(2,20,1),ts,c="red")
```

Out[57]:
```
[<matplotlib.lines.Line2D at 0x1204da0b3d0>]
```



Model-2] Random Forest Classifier (Bagging Approach)

In [58]:
```python
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
```

In [59]:
```python
param_grid = {
    'n_estimators': [200, 500],
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth' : [4,5,6,7,8],
    'criterion' :['gini', 'entropy']
}
```

In [60]:
```python
rfc = RandomForestClassifier()
CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 5)
CV_rfc.fit(xtrain, ytrain)
```

Out[60]:
```
GridSearchCV(cv=5, estimator=RandomForestClassifier(),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [4, 5, 6, 7, 8],
                         'max_features': ['auto', 'sqrt', 'log2'],
                         'n_estimators': [200, 500]})
```

In [61]:
```python
CV_rfc.best_params_
```

Out[61]:
```
{'criterion': 'entropy',
 'max_depth': 4,
 'max_features': 'auto',
 'n_estimators': 200}
```

In [62]:
```python
rfc = RandomForestClassifier(random_state= 21, criterion= 'entropy',max_depth= 4, max_fe
```

In [83]:
```python
model = rfc.fit(xtrain,ytrain)
pred_tr = model.predict(xtrain)
pred_ts = model.predict(xtest)
#training error
tr_acc = accuracy_score(ytrain,pred_tr)

tr.append(tr_acc)

#testing error
uracy_score(ytest,pred_ts)
```

```
ts.append(ts_acc)
```

In [64]:
```
tr_acc
```

Out[64]:
```
0.7961956521739131
```

In [65]:
```
ts_acc
```

Out[65]:
```
0.8414634146341463
```

Model 3] Adaboost classifier (Boosting Approach)

In [66]:
```
from sklearn.ensemble import AdaBoostClassifier
```

In [67]:
```
param_grid = {
    'n_estimators': [10, 50,100,200,500],
    'learning_rate': [0.0001,0.001,0.01,0.1,1.0],
    'algorithm' : ['SAMME','SAMME.R']
}
```

In [68]:
```
abc = AdaBoostClassifier()
CV_abc = GridSearchCV(estimator=abc, param_grid=param_grid, cv= 10,scoring='accuracy')
```

In [69]:
```
CV_abc.fit(xtrain,ytrain)
```

Out[69]:
```
GridSearchCV(cv=10, estimator=AdaBoostClassifier(),
             param_grid={'algorithm': ['SAMME', 'SAMME.R'],
                         'learning_rate': [0.0001, 0.001, 0.01, 0.1, 1.0],
                         'n_estimators': [10, 50, 100, 200, 500]},
             scoring='accuracy')
```

In [70]:
```
CV_abc.best_params_
```

Out[70]:
```
{'algorithm': 'SAMME', 'learning_rate': 0.0001, 'n_estimators': 10}
```

In [71]:
```
CV_abc.best_score_
```

Out[71]:
```
0.7882132132132132
```

Model 4] Logistic Regression

In [84]:
```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
model = lr.fit(xtrain,ytrain)
```

In [85]:
```
pred_tr = model.predict(xtrain)
pred_ts = model.predict(xtest)
```

In [74]:
```
from sklearn.metrics import accuracy_score
print((accuracy_score(ytrain,pred_tr)))
print((accuracy_score(ytest,pred_ts)))
```

```
0.7907608695652174
0.7804878048780488
```

Model 5] SVM(Support Vector Machine)

In [75]:
```
from sklearn.svm import SVC
svc = SVC(random_state= 21, kernel= 'sigmoid',C= 0.1)
model = svc.fit(xtrain,ytrain)
```

In [76]:
```
pred_tr = model.predict(xtrain)
pred_ts = model.predict(xtest)
tr_acc = accuracy_score(ytrain,pred_tr)
ts_acc = accuracy_score(ytest,pred_ts)
```

In [77]:
```
tr_acc
```

Out[77]:
```
0.7771739130434783
```

In [78]:
```
ts_acc
```

Out[78]:
```
0.8048780487804879
```

Making test set ready for prediction

In [80]:
```
B = pd.read_csv("C:/Work/DATA_SCI-ANA/Datasets/Loan_Prediction/testing_set.csv")
```

In [81]:
```
B.head()
```

Out[81]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	Lo
0	LP001015	Male	Yes	0	Graduate	No	5720	0	
1	LP001022	Male	Yes	1	Graduate	No	3076	1500	
2	LP001031	Male	Yes	2	Graduate	No	5000	1800	
3	LP001035	Male	Yes	2	Graduate	No	2340	2546	
4	LP001051	Male	No	0	Not Graduate	No	3276	0	

In [82]:
```
B.shape
```

Out[82]:
```
(367, 12)
```

In [ ]:
```
B.isna().sum()
```

Treating Null Values

In [86]:
```
for i in B.columns:
    if (B[i].dtypes == 'object'):
        x = B[i].mode()[0]
        B[i] = B[i].fillna(x)
    else:
        x = B[i].mean()
        B[i] = B[i].fillna(x)
```

Diffrentiating categorical and continuous values

In [87]:
```
cat = []
con = []


for i in B.columns:
```

Loading [MathJax]/extensions/Safe.js

```python
    if (B[i].dtypes == 'object'):
        cat.append(i)
    else:
        con.append(i)
```

Treating skew

In [88]:
```python
from sklearn.preprocessing import PowerTransformer
pt = PowerTransformer()
conti = pt.fit_transform(B[con])
```

In [89]:
```python
conti = pd.DataFrame(conti,columns = con)
```

In [90]:
```python
cate = B[cat]
```

In [91]:
```python
cate = cate.drop(columns = ['Loan_ID']) #dropped discrete column
```

One Hot Encoding Categorical Data

In [92]:
```python
cate = pd.get_dummies(cate)
```

In [93]:
```python
df = conti.join(cate)
```

In [94]:
```python
df
```

Out[94]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Gender_Female	Ge
0	0.544763	-1.142887	-0.331752	0.184091	0.535482	0	
1	-0.420494	0.716870	-0.006080	0.184091	0.535482	0	
2	0.322653	0.779145	1.243800	0.184091	0.535482	0	
3	-0.801114	0.900225	-0.557077	0.184091	-0.819504	0	
4	-0.329105	-1.142887	-1.132009	0.184091	0.535482	0	
...	...	...	...	...	...	...	
362	-0.026362	0.774721	-0.267657	0.184091	0.535482	0	
363	0.029986	0.470668	-0.225750	0.184091	0.535482	0	
364	-0.340744	0.814350	-0.006080	0.184091	-0.819504	0	
365	0.322653	0.878325	0.548836	0.184091	0.535482	0	
366	1.391376	-1.142887	-0.604497	-2.407409	0.535482	0	

367 rows × 20 columns

In [95]:
```python
cols_to_keep = xtrain.columns
```

In [96]:
```python
final = df[cols_to_keep]
```

In [97]:
```python
pred = model.predict(final)
```

In [98]:
```python
T = B[["Loan_ID"]]
T['Loan_Status']=pred
```

```python
ediction -- Based on results of Logistic Regression
```

Out[100]:

	Loan_ID	Loan_Status
0	LP001015	Y
1	LP001022	Y
2	LP001031	Y
3	LP001035	Y
4	LP001051	Y
...	...	...
362	LP002971	Y
363	LP002975	Y
364	LP002980	Y
365	LP002986	Y
366	LP002989	Y

367 rows × 2 columns

In [ ]:

In [ ]:

In [ ]: