# project name : Loan Application Status Prediction

In [2]:

```
#Importing the libreris like pandas, numpy for selecing the data and convering the data to
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings('ignore')
```

In [3]:

```
#Creating the variable df and loading the dataset to the variable df
df=pd.read_csv('loanstatus.csv')
df
```

Out[3]:

| | Loan_ID | Gender | Married | Dependents | Education | Self_Employed | ApplicantIncome | CoapplicantIncome | Lo |
|---|---|---|---|---|---|---|---|---|---|
| **0** | LP001002 | Male | No | 0 | Graduate | No | 5849 | 0.0 | |
| **1** | LP001003 | Male | Yes | 1 | Graduate | No | 4583 | 1508.0 | |
| **2** | LP001005 | Male | Yes | 0 | Graduate | Yes | 3000 | 0.0 | |
| **3** | LP001006 | Male | Yes | 0 | Not Graduate | No | 2583 | 2358.0 | |
| **4** | LP001008 | Male | No | 0 | Graduate | No | 6000 | 0.0 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | |
| **609** | LP002978 | Female | No | 0 | Graduate | No | 2900 | 0.0 | |
| **610** | LP002979 | Male | Yes | 3+ | Graduate | No | 4106 | 0.0 | |
| **611** | LP002983 | Male | Yes | 1 | Graduate | No | 8072 | 240.0 | |
| **612** | LP002984 | Male | Yes | 2 | Graduate | No | 7583 | 0.0 | |
| **613** | LP002990 | Female | No | 0 | Graduate | Yes | 4583 | 0.0 | |

614 rows × 13 columns

In [4]:

```
#Checking the df dataframe for null values, if null value present we need to remove (or) f
df.isnull().sum()
```

Out[4]:

```
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
```

null values present in the df dataset we need to remove(or) fill the null values

In [5]:

```
#cheking the datatype in which the data columns are present
df.dtypes
```

Out[5]:
```
Loan_ID              object
Gender               object
Married              object
Dependents           object
Education            object
Self_Employed        object
ApplicantIncome       int64
CoapplicantIncome   float64
LoanAmount          float64
Loan_Amount_Term    float64
Credit_History      float64
Property_Area        object
Loan_Status          object
dtype: object
```

object, int and float types of data types present in the df dataset

In [6]:
```
#cheking the datatype of df dataframe columns
df.info()
```

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

In [9]:
```
#filling the missing data
print("Before filling missing values\n\n","#"*50,"\n")
null_cols = ['Credit_History', 'Self_Employed', 'LoanAmount','Dependents', 'Loan_Amount_Te


for col in null_cols:
    print(f"{col}:\n{df[col].value_counts()}\n","-"*50)
    df[col] = df[col].fillna(
    df[col].dropna().mode().values[0] )


df.isnull().sum().sort_values(ascending=False)
print("After filling missing values\n\n","#"*50,"\n")
for col in null_cols:
    print(f"\n{col}:\n{df[col].value_counts()}\n","-"*50)
```

```
Before filling missing values

 ##################################################

Credit_History:
1.0    475
```

```
0.0     89
Name: Credit_History, dtype: int64
 --------------------------------------------------
Self_Employed:
No     500
Yes     82
Name: Self_Employed, dtype: int64
 --------------------------------------------------
LoanAmount:
120.0    20
110.0    17
100.0    15
160.0    12
187.0    12
         ..
240.0     1
214.0     1
59.0      1
166.0     1
253.0     1
Name: LoanAmount, Length: 203, dtype: int64
 --------------------------------------------------
Dependents:
0     345
1     102
2     101
3+     51
Name: Dependents, dtype: int64
 --------------------------------------------------
Loan_Amount_Term:
360.0    512
180.0     44
480.0     15
300.0     13
240.0      4
84.0       4
120.0      3
60.0       2
36.0       2
12.0       1
Name: Loan_Amount_Term, dtype: int64
 --------------------------------------------------
Gender:
Male      489
Female    112
Name: Gender, dtype: int64
 --------------------------------------------------
Married:
Yes    398
No     213
Name: Married, dtype: int64
 --------------------------------------------------
After filling missing values

 ##################################################


Credit_History:
1.0    525
0.0     89
Name: Credit_History, dtype: int64
 --------------------------------------------------

Self_Employed:
```

```
No     532
Yes     82
Name: Self_Employed, dtype: int64
 --------------------------------------------------

LoanAmount:
120.0    42
110.0    17
100.0    15
160.0    12
187.0    12
         ..
240.0     1
214.0     1
59.0      1
166.0     1
253.0     1
Name: LoanAmount, Length: 203, dtype: int64
 --------------------------------------------------

Dependents:
0     360
1     102
2     101
3+     51
Name: Dependents, dtype: int64
 --------------------------------------------------

Loan_Amount_Term:
360.0    526
180.0     44
480.0     15
300.0     13
240.0      4
84.0       4
120.0      3
60.0       2
36.0       2
12.0       1
Name: Loan_Amount_Term, dtype: int64
 --------------------------------------------------

Gender:
Male      502
Female    112
Name: Gender, dtype: int64
 --------------------------------------------------

Married:
Yes    401
No     213
Name: Married, dtype: int64
 --------------------------------------------------
```

In [10]:

```
#after filling all the null values, printing the df dataset
df
```

Out[10]:

| | Loan_ID | Gender | Married | Dependents | Education | Self_Employed | ApplicantIncome | CoapplicantIncome | Lo |
|---|---|---|---|---|---|---|---|---|---|
| 0 | LP001002 | Male | No | 0 | Graduate | No | 5849 | 0.0 | |
| 1 | LP001003 | Male | Yes | 1 | Graduate | No | 4583 | 1508.0 | |
| 2 | LP001005 | Male | Yes | 0 | Graduate | Yes | 3000 | 0.0 | |

| | Loan_ID | Gender | Married | Dependents | Education | Self_Employed | ApplicantIncome | CoapplicantIncome | Lo |
|---|---|---|---|---|---|---|---|---|---|
| **3** | LP001006 | Male | Yes | 0 | Not Graduate | No | 2583 | 2358.0 | |
| **4** | LP001008 | Male | No | 0 | Graduate | No | 6000 | 0.0 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | |
| **609** | LP002978 | Female | No | 0 | Graduate | No | 2900 | 0.0 | |
| **610** | LP002979 | Male | Yes | 3+ | Graduate | No | 4106 | 0.0 | |
| **611** | LP002983 | Male | Yes | 1 | Graduate | No | 8072 | 240.0 | |
| **612** | LP002984 | Male | Yes | 2 | Graduate | No | 7583 | 0.0 | |
| **613** | LP002990 | Female | No | 0 | Graduate | Yes | 4583 | 0.0 | |

614 rows × 13 columns

In [11]:

```
# checking for null values
df.isnull().sum()
```

Out[11]:

```
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
```

null values is filled and now the dataset df doesnot have any null values

In [12]:

```
# to know the statistical data we use describe function
df.describe()
```

Out[12]:

| | ApplicantIncome | CoapplicantIncome | LoanAmount | Loan_Amount_Term | Credit_History |
|---|---|---|---|---|---|
| **count** | 614.000000 | 614.000000 | 614.000000 | 614.000000 | 614.000000 |
| **mean** | 5403.459283 | 1621.245798 | 145.465798 | 342.410423 | 0.855049 |
| **std** | 6109.041673 | 2926.248369 | 84.180967 | 64.428629 | 0.352339 |
| **min** | 150.000000 | 0.000000 | 9.000000 | 12.000000 | 0.000000 |
| **25%** | 2877.500000 | 0.000000 | 100.250000 | 360.000000 | 1.000000 |
| **50%** | 3812.500000 | 1188.500000 | 125.000000 | 360.000000 | 1.000000 |
| **75%** | 5795.000000 | 2297.250000 | 164.750000 | 360.000000 | 1.000000 |
| **max** | 81000.000000 | 41667.000000 | 700.000000 | 480.000000 | 1.000000 |

1) After filling all the null values,number of counts in each columns are same which means there is no null value in the data set

2) by seeing the 75% percentail value and max value we can say that outliers are present in the dataset which need to be removed

3) With the help of obove table we can know the statistical information of each columns in the data set

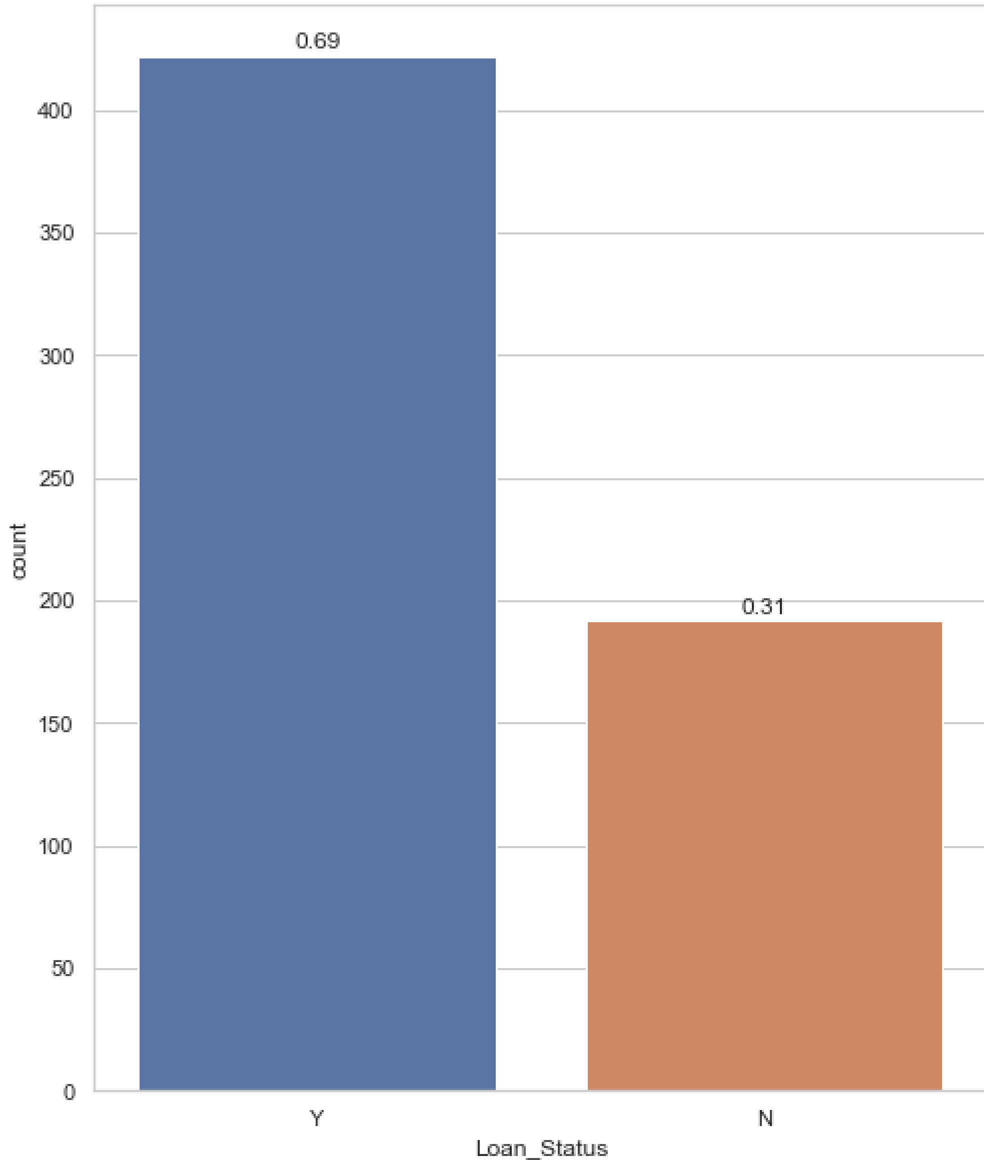
In [13]:

```
#list of all the columns.columns
#Cols = tr_df.tolist()
#list of all the numeric columns
num = df.select_dtypes('number').columns.to_list()
#list of all the categoric columns
cat = df.select_dtypes('object').columns.to_list()

#numeric df
loan_num = df[num]
#categoric df
loan_cat = df[cat]
```

In [15]:

```
import matplotlib.pyplot as plt
import seaborn as sns
loan_cat = df[cat]
print(df[cat[-1]].value_counts())
#tr_df[cat[-1]].hist(grid = False)

#print(i)
total = float(len(df[cat[-1]]))
plt.figure(figsize=(8,10))
sns.set(style="whitegrid")
ax = sns.countplot(df[cat[-1]])
for p in ax.patches:
    height = p.get_height()
    ax.text(p.get_x()+p.get_width()/2.,height + 3,'{:1.2f}'.format(height/total),ha="cente
plt.show()
```

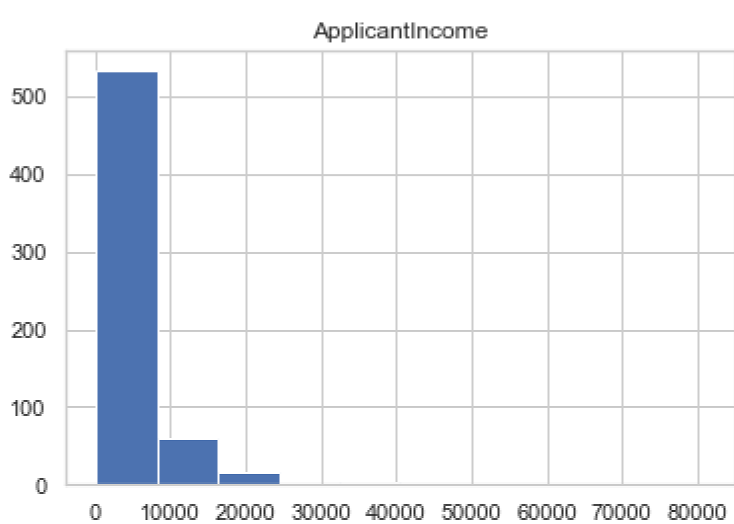
```
Y    422
N    192
Name: Loan_Status, dtype: int64
```

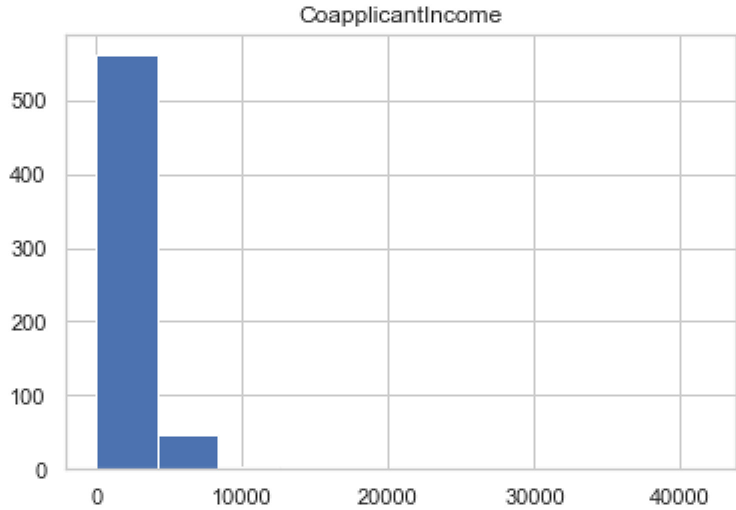


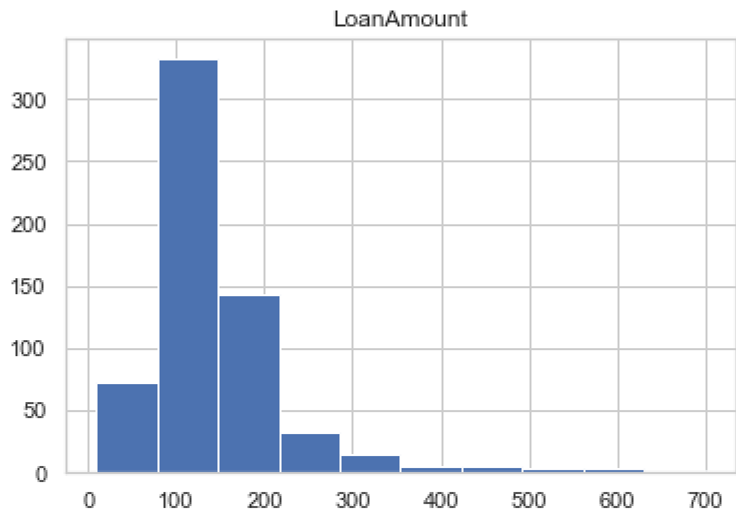

From loan status graph, we can say that the percentage of loan approved is 69% when compare to the total application submitted for loan

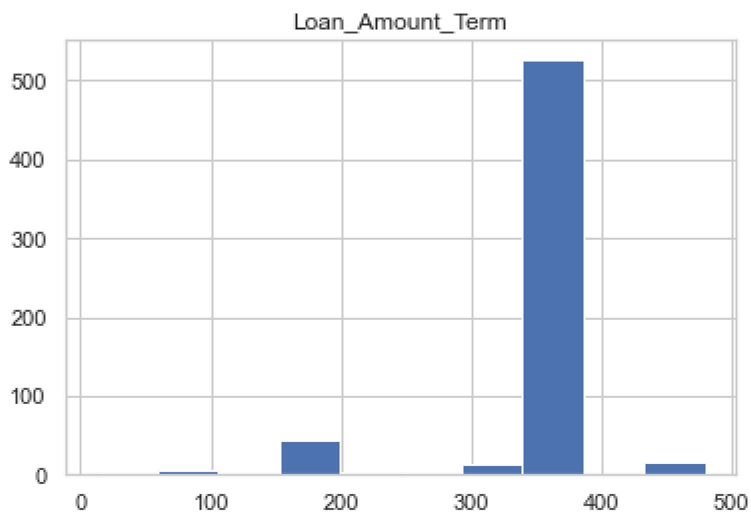
In [16]:

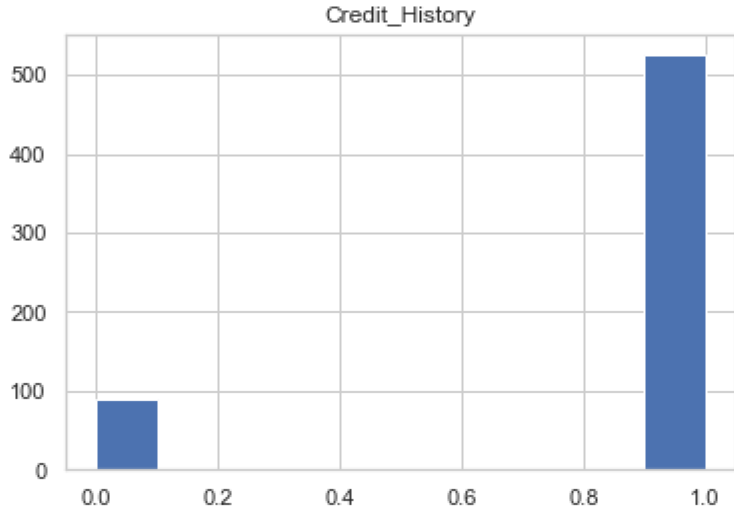
```
# plotting the numerical columns data
for i in loan_num:
    plt.hist(loan_num[i])
    plt.title(i)
    plt.show()
```




CoapplicantIncome
0
100
200
300
400
500
0
10000
20000
30000
40000


LoanAmount
0
50
100
150
200
250
300
0
100
200
300
400
500
600
700

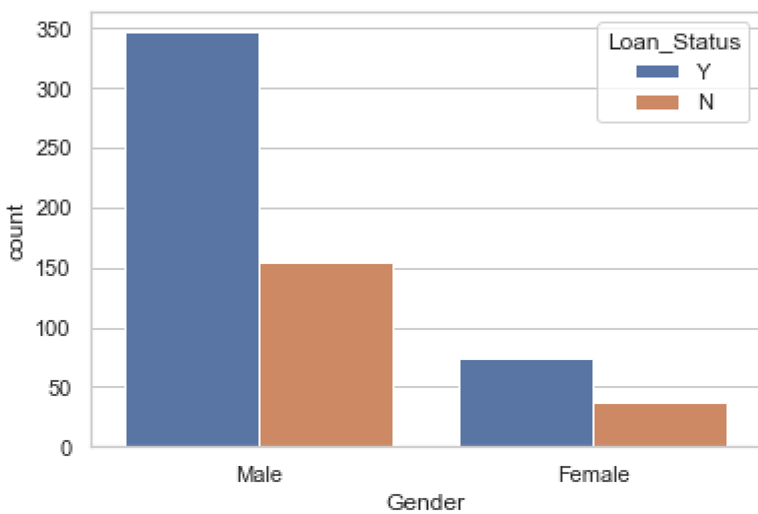

Loan_Amount_Term
0
100
200
300
400
500
0
100
200
300
400
500




- if we consider the applicantincome column, graph shows that the persons who applied for loan maximum people having the income between the range 0 to 10000
- if we consider the coapplicantincome column, graph shows that the person's coapplicant income ranges from 0 to 10000
- if we consider the loan amount column, graph shows that the maximum people applied for loan in the range of 100 to 200
- The above table also shows the loan_ amount_term and credit history

In [25]:

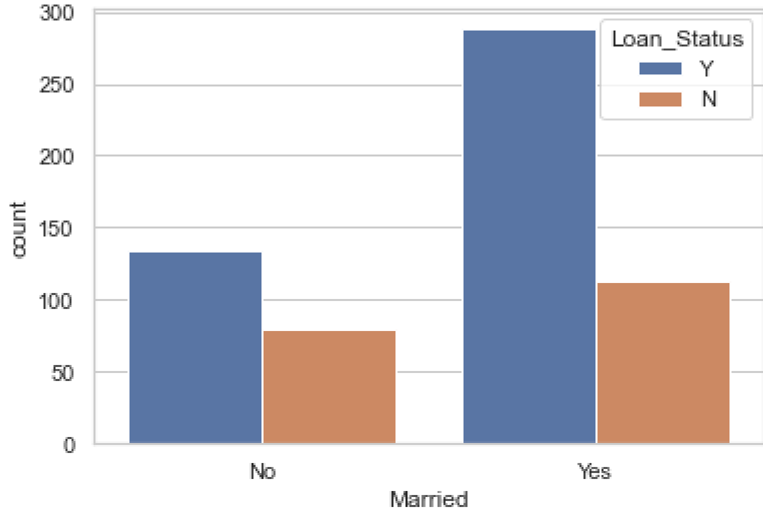
```
# categorical columns ( spliting by loan status columns values)
sns.countplot(x='Gender',hue='Loan_Status', data=df)
plt.show()
```




- from graph we can say , loan approved for male is more when compare to female

In [26]:

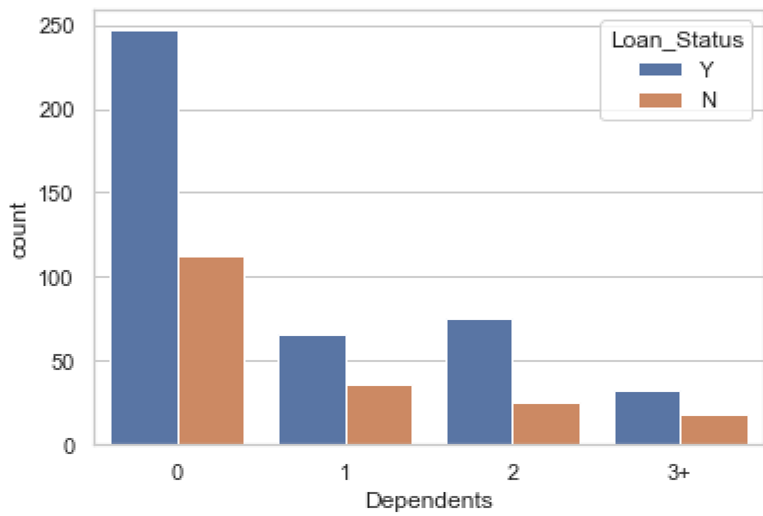
```
sns.countplot(x='Married',hue='Loan_Status',data=df)
plt.show()
```




From graph we can say, the loan approved for married person is more when we compare loan_status column with married column in dataset

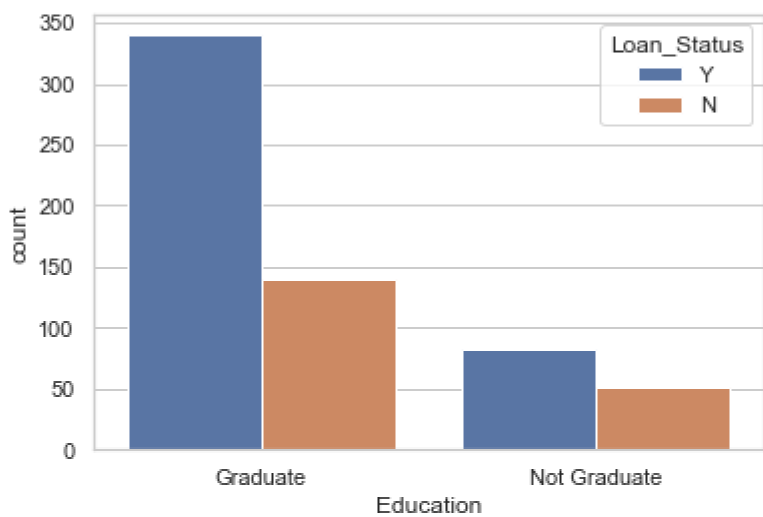
In [27]:

```
sns.countplot(x='Dependents',hue='Loan_Status',data=df)
plt.show()
```



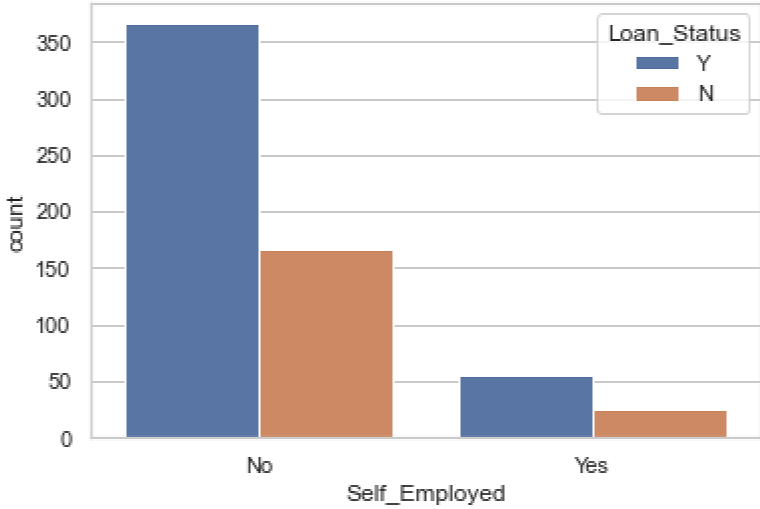

In [28]:

```
sns.countplot(x='Education',hue='Loan_Status',data=df)
plt.show()
```



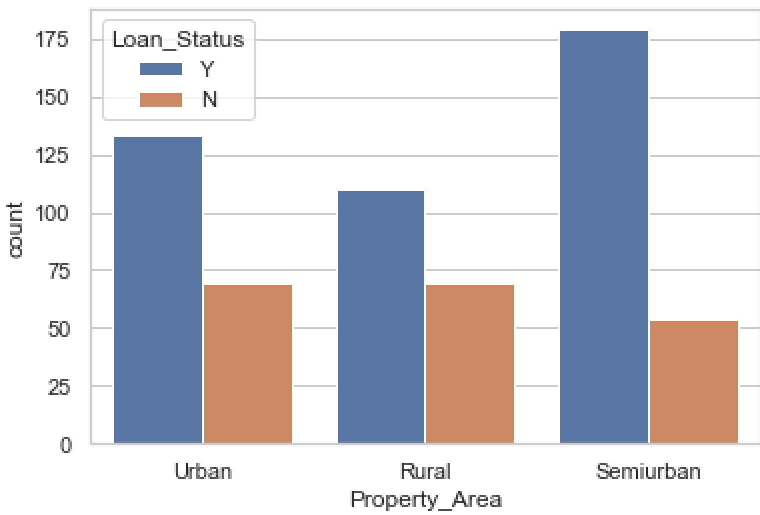

From graph we can say that, the loan approved for graduate is more when compte to not graduate

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




From graph we can say that, the loan approved for people with no self employed status is more( mean people who are working for salary ) when compared to self employed status yes

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




From graph we can say that , the loan approved for semiurban is more then followed by urban and rural area as shown in the graph above

In [31]:
```
# converting the categorical columns to numerical columns by labelEncoder
from sklearn.preprocessing import LabelEncoder
for col in df.columns:
    if df[col].dtype=='object':
        encode=LabelEncoder()
        df[col]=encode.fit_transform(df[col])
df
```

Out[31]:

| | Loan_ID | Gender | Married | Dependents | Education | Self_Employed | ApplicantIncome | CoapplicantIncome | Loa |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 1 | 0 | 0 | 0 | 0 | 5849 | 0.0 | |
| **1** | 1 | 1 | 1 | 1 | 0 | 0 | 4583 | 1508.0 | |
| **2** | 2 | 1 | 1 | 0 | 0 | 1 | 3000 | 0.0 | |

Loading [MathJax]/extensions/Safe.js

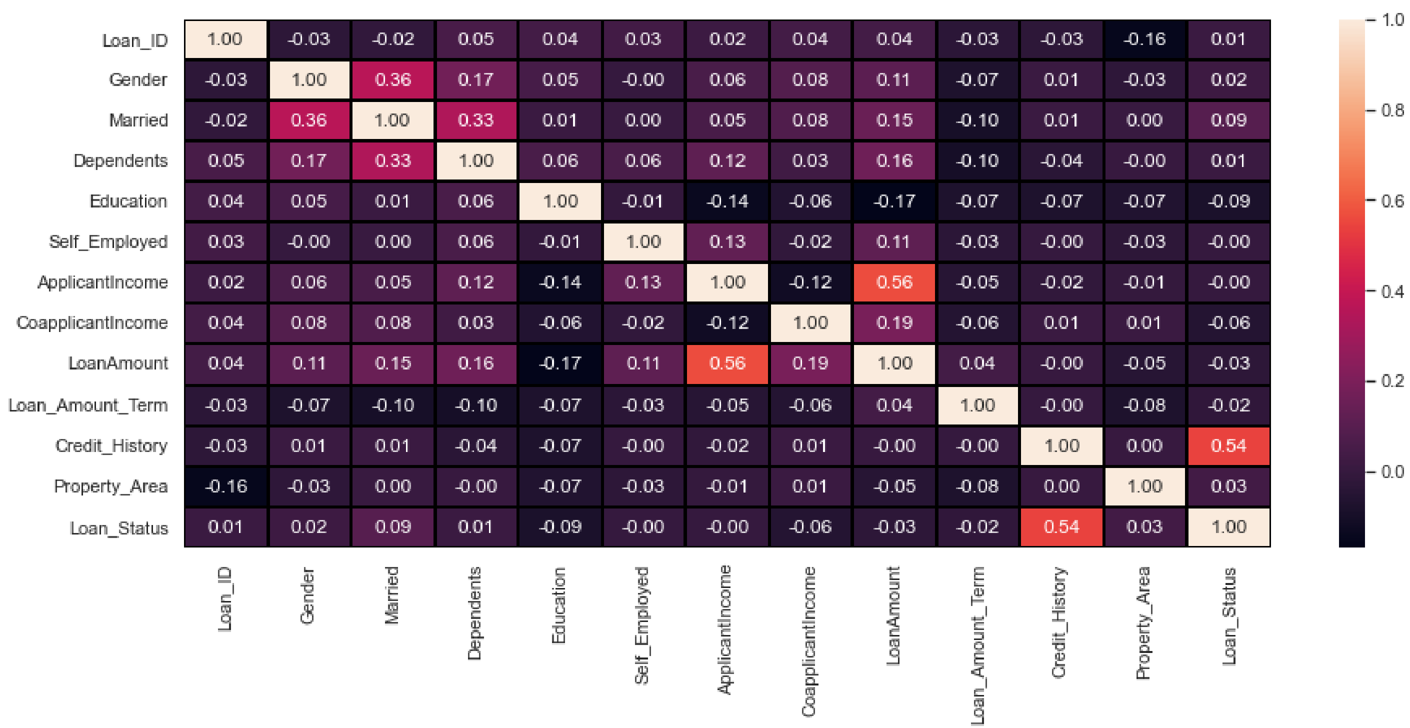
| | Loan_ID | Gender | Married | Dependents | Education | Self_Employed | ApplicantIncome | CoapplicantIncome | Loa |
|---|---|---|---|---|---|---|---|---|---|
| **3** | 3 | 1 | 1 | 0 | 1 | 0 | 2583 | 2358.0 | |
| **4** | 4 | 1 | 0 | 0 | 0 | 0 | 6000 | 0.0 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | |
| **609** | 609 | 0 | 0 | 0 | 0 | 0 | 2900 | 0.0 | |
| **610** | 610 | 1 | 1 | 3 | 0 | 0 | 4106 | 0.0 | |
| **611** | 611 | 1 | 1 | 1 | 0 | 0 | 8072 | 240.0 | |
| **612** | 612 | 1 | 1 | 2 | 0 | 0 | 7583 | 0.0 | |
| **613** | 613 | 0 | 0 | 0 | 0 | 1 | 4583 | 0.0 | |

614 rows × 13 columns

In [32]:

```
# Corelation can also be reprasented by heatmap
import matplotlib.pyplot as plt
plt.figure(figsize=(15,6))
sns.heatmap(df.corr(),annot=True,linewidth=0.1,linecolor='black',fmt='0.2f')
```

Out[32]: <AxesSubplot:>




from The corellation heat map we can say that the column credit_history having the good co relation value with loan_status, therefore our target value is highly dependent on credit_history column

In [39]:

```
df.skew()
```

Out[39]:

```
Loan_ID              0.000000
Gender              -1.648795
Married             -0.644850
Dependents           1.015551
Education            1.367622
Self_Employed        2.159796
ApplicantIncome      6.539513
CoapplicantIncome    7.491531
                     2.745407
```

```
Loan_Amount_Term    -2.402112
Credit_History      -2.021971
Property_Area       -0.066196
Loan_Status         -0.809998
dtype: float64
```

from the table above we can say that the columns like Gender,Married,Dependents,Education,Self_Employed,ApplicantIncome,CoapplicantIncome, LoanAmount,Loan_Amount_Term,Credit_History,Loan_Status having the skewness, we need to remove the skewness

In [40]:

```
# Loan_id doesnot have any impact on target column so we can drop the loan_id column
df=df.drop(columns='Loan_ID')
df
```

Out[40]:

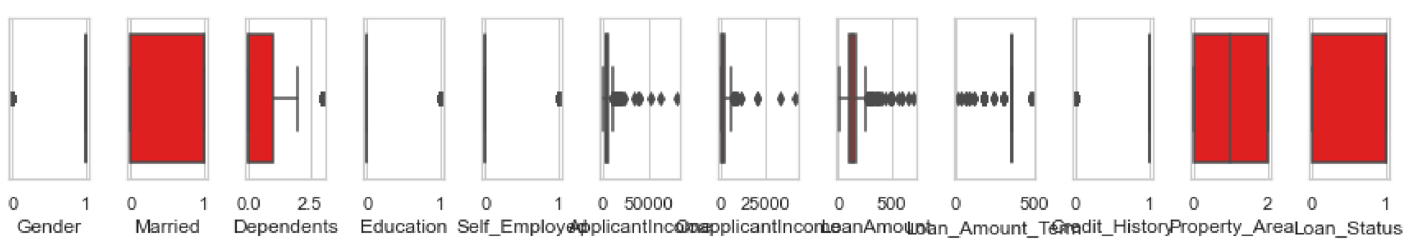
| | Gender | Married | Dependents | Education | Self_Employed | ApplicantIncome | CoapplicantIncome | LoanAmount |
|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 0 | 0 | 0 | 0 | 5849 | 0.0 | 120.0 |
| **1** | 1 | 1 | 1 | 0 | 0 | 4583 | 1508.0 | 128.0 |
| **2** | 1 | 1 | 0 | 0 | 1 | 3000 | 0.0 | 66.0 |
| **3** | 1 | 1 | 0 | 1 | 0 | 2583 | 2358.0 | 120.0 |
| **4** | 1 | 0 | 0 | 0 | 0 | 6000 | 0.0 | 141.0 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... |
| **609** | 0 | 0 | 0 | 0 | 0 | 2900 | 0.0 | 71.0 |
| **610** | 1 | 1 | 3 | 0 | 0 | 4106 | 0.0 | 40.0 |
| **611** | 1 | 1 | 1 | 0 | 0 | 8072 | 240.0 | 253.0 |
| **612** | 1 | 1 | 2 | 0 | 0 | 7583 | 0.0 | 187.0 |
| **613** | 0 | 0 | 0 | 0 | 1 | 4583 | 0.0 | 133.0 |

614 rows × 12 columns

In the above dataframe df Loan_id is droped

In [41]:

```
#check for outliers with the help of box plot
columns_list=df.columns.values
ncol=100
nrows=50
plt.figure(figsize=(ncol,ncol))
for i in range (0,len(columns_list)):
    plt.subplot(nrows,ncol,i+1)
    sns.boxplot(df[columns_list[i]],color='red',orient='h')
    plt.tight_layout()
```




From the graph we can see that the outliers present in the dataset , so we need to remove

In [42]:

```
# checking how much number of outliers are present in the dataframe
```

```
from scipy.stats import zscore
z=np.abs(zscore(df))
print(df.shape)
print(z.shape)
threshold=3
print(np.where(z>3))
len(np.where(z>3)[0])
```

```
(614, 12)
(614, 12)
(array([  9,  14,  68,  94, 126, 130, 133, 155, 155, 171, 171, 177, 177,
       183, 185, 242, 262, 278, 308, 313, 333, 333, 369, 402, 409, 417,
       432, 443, 487, 495, 497, 506, 523, 525, 546, 561, 575, 581, 585,
       600, 604], dtype=int64), array([6, 8, 8, 8, 5, 7, 8, 5, 7, 5, 7, 6, 7, 5, 5, 8, 8,
7, 7, 8, 5, 7,
       7, 6, 5, 6, 7, 5, 7, 8, 8, 7, 7, 7, 8, 7, 8, 6, 8, 6, 7],
      dtype=int64))
```

Out[42]:

```
41
```

41 numbers outliers present in the dataset df

In [43]:

```
# 41 outliers are present in the df data frame, now we are removing the outliers
df_new=df[(z<3).all(axis=1)]
print(df_new.shape)
```

```
(577, 12)
```

df_new is the dataset obtained after removing the 41 outliers

In [45]:

```
#For training the model we need to split the data frame values into Xtrain,Xtest,Ytrain,Yt
x= df.drop('Loan_Status', axis=1)
y= df['Loan_Status']
print(x)
print(y)
```

```
     Gender  Married  Dependents  Education  Self_Employed  ApplicantIncome  \
0         1        0           0          0              0             5849
1         1        1           1          0              0             4583
2         1        1           0          0              1             3000
3         1        1           0          1              0             2583
4         1        0           0          0              0             6000
..      ...      ...         ...        ...            ...              ...
609       0        0           0          0              0             2900
610       1        1           3          0              0             4106
611       1        1           1          0              0             8072
612       1        1           2          0              0             7583
613       0        0           0          0              1             4583

     CoapplicantIncome  LoanAmount  Loan_Amount_Term  Credit_History  \
0                  0.0       120.0             360.0             1.0
1               1508.0       128.0             360.0             1.0
2                  0.0        66.0             360.0             1.0
3               2358.0       120.0             360.0             1.0
4                  0.0       141.0             360.0             1.0
..                 ...         ...               ...             ...
609                0.0        71.0             360.0             1.0
610                0.0        40.0             180.0             1.0
611              240.0       253.0             360.0             1.0
612                0.0       187.0             360.0             1.0
613                0.0       133.0             360.0             0.0

     Property_Area
                 2
```

```
1                 0
2                 2
3                 2
4                 2
..              ...
609               0
610               0
611               2
612               2
613               1

[614 rows x 11 columns]
0      1
1      0
2      1
3      1
4      1
      ..
609    1
610    1
611    1
612    1
613    0
Name: Loan_Status, Length: 614, dtype: int32
```

In [46]:

```
#removing the skewness by yeo johnson method
from sklearn.preprocessing import power_transform
x=power_transform(x,method='yeo-johnson')
x
```

Out[46]:

```
array([[ 4.72342640e-01, -1.37208932e+00, -8.27104306e-01, ...,
         1.75540037e-01,  4.11732692e-01,  1.19356680e+00],
       [ 4.72342640e-01,  7.28815525e-01,  8.54259122e-01, ...,
         1.75540037e-01,  4.11732692e-01, -1.35000343e+00],
       [ 4.72342640e-01,  7.28815525e-01, -8.27104306e-01, ...,
         1.75540037e-01,  4.11732692e-01,  1.19356680e+00],
       ...,
       [ 4.72342640e-01,  7.28815525e-01,  8.54259122e-01, ...,
         1.75540037e-01,  4.11732692e-01,  1.19356680e+00],
       [ 4.72342640e-01,  7.28815525e-01,  1.31670248e+00, ...,
         1.75540037e-01,  4.11732692e-01,  1.19356680e+00],
       [-2.11710719e+00, -1.37208932e+00, -8.27104306e-01, ...,
         1.75540037e-01, -2.42876026e+00,  2.36103342e-03]])
```

In [48]:

```
pd.DataFrame(x).skew()
```

Out[48]:

```
0    -1.648795
1    -0.644850
2     0.441404
3     1.367622
4     2.159796
5    -0.092946
6    -0.145646
7     0.018936
8     0.392571
9    -2.021971
10   -0.158267
dtype: float64
```

In [50]:

```
from sklearn.preprocessing import MinMaxScaler
mms=MinMaxScaler()
x1=mms.fit_transform(x)
```

In [51]:

```
# importing all the algorithems for checking the accuracy_score  and model perforfance
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split,cross_val_score,GridSearchCV
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report,f1_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
```

In [64]:

```
model=[RandomForestClassifier(),DecisionTreeClassifier(),SVC(),KNeighborsClassifier(),Gaus
max_r2_score=0
for i_state in range(0,10):
    x_train,x_test,y_train,y_test=train_test_split(x1,y,random_state=i_state,test_size=0.3
    for a in model:
        a.fit(x_train,y_train)
        pred=a.predict(x_test)
        score=accuracy_score(y_test,pred)
        print('score for random_state',i_state,'is',score)
        if score>max_r2_score:
            max_r2_score=score
            Final_state=i_state
            Final_model= a
print('accuracy_score ',max_r2_score,'for random state ',Final_state, 'and model is ',Fina
```

```
score for random_state 0 is 0.812807881773399
score for random_state 0 is 0.6798029556650246
score for random_state 0 is 0.8226600985221675
score for random_state 0 is 0.8078817733990148
score for random_state 0 is 0.8226600985221675
score for random_state 0 is 0.8226600985221675
score for random_state 1 is 0.7586206896551724
score for random_state 1 is 0.7438423645320197
score for random_state 1 is 0.7733990147783252
score for random_state 1 is 0.7684729064039408
score for random_state 1 is 0.7635467980295566
score for random_state 1 is 0.7733990147783252
score for random_state 2 is 0.7783251231527094
score for random_state 2 is 0.7093596059113301
score for random_state 2 is 0.7931034482758621
score for random_state 2 is 0.7783251231527094
score for random_state 2 is 0.7684729064039408
score for random_state 2 is 0.7931034482758621
score for random_state 3 is 0.8078817733990148
score for random_state 3 is 0.7093596059113301
score for random_state 3 is 0.8472906403940886
score for random_state 3 is 0.8078817733990148
score for random_state 3 is 0.8374384236453202
score for random_state 3 is 0.8472906403940886
score for random_state 4 is 0.7832512315270936
score for random_state 4 is 0.6945812807881774
score for random_state 4 is 0.7980295566502463
score for random_state 4 is 0.7684729064039408
score for random_state 4 is 0.7931034482758621
score for random_state 4 is 0.7980295566502463
score for random_state 5 is 0.8177339901477833
score for random_state 5 is 0.6847290640394089
score for random_state 5 is 0.8374384236453202
score for random_state 5 is 0.8275862068965517
score for random_state 5 is 0.8374384236453202
score for random_state 5 is 0.8374384236453202
```

Loading [MathJax]/extensions/Safe.js

```
om_state 6 is 0.8275862068965517
```

```
score for random_state 6 is 0.7684729064039408
score for random_state 6 is 0.8325123152709359
score for random_state 6 is 0.8177339901477833
score for random_state 6 is 0.8226600985221675
score for random_state 6 is 0.8325123152709359
score for random_state 7 is 0.7881773399014779
score for random_state 7 is 0.6748768472906403
score for random_state 7 is 0.8029556650246306
score for random_state 7 is 0.7438423645320197
score for random_state 7 is 0.7881773399014779
score for random_state 7 is 0.8029556650246306
score for random_state 8 is 0.8374384236453202
score for random_state 8 is 0.6995073891625616
score for random_state 8 is 0.8620689655172413
score for random_state 8 is 0.8177339901477833
score for random_state 8 is 0.8522167487684729
score for random_state 8 is 0.8620689655172413
score for random_state 9 is 0.7832512315270936
score for random_state 9 is 0.6995073891625616
score for random_state 9 is 0.7881773399014779
score for random_state 9 is 0.7635467980295566
score for random_state 9 is 0.7832512315270936
score for random_state 9 is 0.7881773399014779
accuracy_score  0.8620689655172413 for random state  8 and model is  SVC()
```

In [71]:

```
# we are training the model with SVC() for randomstate 8 and checking the accuracy_score
svc=SVC()
x_train,x_test,y_train,y_test=train_test_split(x1,y,random_state=8,test_size=0.33)
svc.fit(x_train,y_train)
svc.score(x_train,y_train)
pred_y=svc.predict(x_test)
svcs=accuracy_score(y_test,pred_y)
print('accuracy_score =',svcs*100)
print(classification_report(y_test,pred_y))
print(confusion_matrix(y_test,pred_y))
print('F1_score = ',f1_score(y_test,pred_y)*100)
from sklearn.model_selection import cross_val_score
cv_score=cross_val_score(svc,x1,y,cv=5)
cv_mean=cv_score.mean()
print("cross_val_score=",cv_mean*100)
```

```
accuracy_score = 86.20689655172413
              precision    recall  f1-score   support

           0       0.97      0.53      0.68        57
           1       0.84      0.99      0.91       146

    accuracy                           0.86       203
   macro avg       0.91      0.76      0.80       203
weighted avg       0.88      0.86      0.85       203

[[ 30  27]
 [  1 145]]
F1_score =  91.19496855345913
cross_val_score= 80.9462881514061
```

In [72]:

```
from sklearn.model_selection import GridSearchCV
# defining parameter range
param_grid = {'C': [0.1, 1, 10, 100],
              'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
              'kernel': ['rbf']}

grid = GridSearchCV(SVC(), param_grid, refit = True, verbose = 3)
```

```python
# fitting the model for grid search
grid.fit(x_train,y_train)
print(grid.best_params_)
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
[CV 1/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.675 total time=   0.0s
[CV 2/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.671 total time=   0.0s
[CV 3/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.671 total time=   0.0s
[CV 4/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.671 total time=   0.0s
[CV 5/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.671 total time=   0.0s
[CV 1/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.675 total time=   0.0s
[CV 2/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.671 total time=   0.0s
[CV 3/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.671 total time=   0.0s
[CV 4/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.671 total time=   0.0s
[CV 5/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.671 total time=   0.0s
[CV 1/5] END .....C=0.1, gamma=0.01, kernel=rbf;, score=0.675 total time=   0.0s
[CV 2/5] END .....C=0.1, gamma=0.01, kernel=rbf;, score=0.671 total time=   0.0s
[CV 3/5] END .....C=0.1, gamma=0.01, kernel=rbf;, score=0.671 total time=   0.0s
[CV 4/5] END .....C=0.1, gamma=0.01, kernel=rbf;, score=0.671 total time=   0.0s
[CV 5/5] END .....C=0.1, gamma=0.01, kernel=rbf;, score=0.671 total time=   0.0s
[CV 1/5] END ....C=0.1, gamma=0.001, kernel=rbf;, score=0.675 total time=   0.0s
[CV 2/5] END ....C=0.1, gamma=0.001, kernel=rbf;, score=0.671 total time=   0.0s
[CV 3/5] END ....C=0.1, gamma=0.001, kernel=rbf;, score=0.671 total time=   0.0s
[CV 4/5] END ....C=0.1, gamma=0.001, kernel=rbf;, score=0.671 total time=   0.0s
[CV 5/5] END ....C=0.1, gamma=0.001, kernel=rbf;, score=0.671 total time=   0.0s
[CV 1/5] END ...C=0.1, gamma=0.0001, kernel=rbf;, score=0.675 total time=   0.0s
[CV 2/5] END ...C=0.1, gamma=0.0001, kernel=rbf;, score=0.671 total time=   0.0s
[CV 3/5] END ...C=0.1, gamma=0.0001, kernel=rbf;, score=0.671 total time=   0.0s
[CV 4/5] END ...C=0.1, gamma=0.0001, kernel=rbf;, score=0.671 total time=   0.0s
[CV 5/5] END ...C=0.1, gamma=0.0001, kernel=rbf;, score=0.671 total time=   0.0s
[CV 1/5] END ..........C=1, gamma=1, kernel=rbf;, score=0.807 total time=   0.0s
[CV 2/5] END ..........C=1, gamma=1, kernel=rbf;, score=0.756 total time=   0.0s
[CV 3/5] END ..........C=1, gamma=1, kernel=rbf;, score=0.732 total time=   0.0s
[CV 4/5] END ..........C=1, gamma=1, kernel=rbf;, score=0.768 total time=   0.0s
[CV 5/5] END ..........C=1, gamma=1, kernel=rbf;, score=0.817 total time=   0.0s
[CV 1/5] END ........C=1, gamma=0.1, kernel=rbf;, score=0.807 total time=   0.0s
[CV 2/5] END ........C=1, gamma=0.1, kernel=rbf;, score=0.768 total time=   0.0s
[CV 3/5] END ........C=1, gamma=0.1, kernel=rbf;, score=0.756 total time=   0.0s
[CV 4/5] END ........C=1, gamma=0.1, kernel=rbf;, score=0.780 total time=   0.0s
[CV 5/5] END ........C=1, gamma=0.1, kernel=rbf;, score=0.805 total time=   0.0s
[CV 1/5] END .......C=1, gamma=0.01, kernel=rbf;, score=0.675 total time=   0.0s
[CV 2/5] END .......C=1, gamma=0.01, kernel=rbf;, score=0.671 total time=   0.0s
[CV 3/5] END .......C=1, gamma=0.01, kernel=rbf;, score=0.671 total time=   0.0s
[CV 4/5] END .......C=1, gamma=0.01, kernel=rbf;, score=0.671 total time=   0.0s
[CV 5/5] END .......C=1, gamma=0.01, kernel=rbf;, score=0.671 total time=   0.0s
[CV 1/5] END ......C=1, gamma=0.001, kernel=rbf;, score=0.675 total time=   0.0s
[CV 2/5] END ......C=1, gamma=0.001, kernel=rbf;, score=0.671 total time=   0.0s
[CV 3/5] END ......C=1, gamma=0.001, kernel=rbf;, score=0.671 total time=   0.0s
[CV 4/5] END ......C=1, gamma=0.001, kernel=rbf;, score=0.671 total time=   0.0s
[CV 5/5] END ......C=1, gamma=0.001, kernel=rbf;, score=0.671 total time=   0.0s
[CV 1/5] END .....C=1, gamma=0.0001, kernel=rbf;, score=0.675 total time=   0.0s
[CV 2/5] END .....C=1, gamma=0.0001, kernel=rbf;, score=0.671 total time=   0.0s
[CV 3/5] END .....C=1, gamma=0.0001, kernel=rbf;, score=0.671 total time=   0.0s
[CV 4/5] END .....C=1, gamma=0.0001, kernel=rbf;, score=0.671 total time=   0.0s
[CV 5/5] END .....C=1, gamma=0.0001, kernel=rbf;, score=0.671 total time=   0.0s
[CV 1/5] END .........C=10, gamma=1, kernel=rbf;, score=0.723 total time=   0.0s
[CV 2/5] END .........C=10, gamma=1, kernel=rbf;, score=0.671 total time=   0.0s
[CV 3/5] END .........C=10, gamma=1, kernel=rbf;, score=0.720 total time=   0.0s
[CV 4/5] END .........C=10, gamma=1, kernel=rbf;, score=0.695 total time=   0.0s
[CV 5/5] END .........C=10, gamma=1, kernel=rbf;, score=0.841 total time=   0.0s
[CV 1/5] END .......C=10, gamma=0.1, kernel=rbf;, score=0.807 total time=   0.0s
[CV 2/5] END .......C=10, gamma=0.1, kernel=rbf;, score=0.768 total time=   0.0s
[CV 3/5] END .......C=10, gamma=0.1, kernel=rbf;, score=0.756 total time=   0.0s
```

```
[CV 4/5] END .......C=10, gamma=0.1, kernel=rbf;, score=0.780 total time=   0.0s
[CV 5/5] END .......C=10, gamma=0.1, kernel=rbf;, score=0.805 total time=   0.0s
[CV 1/5] END ......C=10, gamma=0.01, kernel=rbf;, score=0.807 total time=   0.0s
[CV 2/5] END ......C=10, gamma=0.01, kernel=rbf;, score=0.768 total time=   0.0s
[CV 3/5] END ......C=10, gamma=0.01, kernel=rbf;, score=0.756 total time=   0.0s
[CV 4/5] END ......C=10, gamma=0.01, kernel=rbf;, score=0.780 total time=   0.0s
[CV 5/5] END ......C=10, gamma=0.01, kernel=rbf;, score=0.805 total time=   0.0s
[CV 1/5] END .....C=10, gamma=0.001, kernel=rbf;, score=0.675 total time=   0.0s
[CV 2/5] END .....C=10, gamma=0.001, kernel=rbf;, score=0.671 total time=   0.0s
[CV 3/5] END .....C=10, gamma=0.001, kernel=rbf;, score=0.671 total time=   0.0s
[CV 4/5] END .....C=10, gamma=0.001, kernel=rbf;, score=0.671 total time=   0.0s
[CV 5/5] END .....C=10, gamma=0.001, kernel=rbf;, score=0.671 total time=   0.0s
[CV 1/5] END ....C=10, gamma=0.0001, kernel=rbf;, score=0.675 total time=   0.0s
[CV 2/5] END ....C=10, gamma=0.0001, kernel=rbf;, score=0.671 total time=   0.0s
[CV 3/5] END ....C=10, gamma=0.0001, kernel=rbf;, score=0.671 total time=   0.0s
[CV 4/5] END ....C=10, gamma=0.0001, kernel=rbf;, score=0.671 total time=   0.0s
[CV 5/5] END ....C=10, gamma=0.0001, kernel=rbf;, score=0.671 total time=   0.0s
[CV 1/5] END ........C=100, gamma=1, kernel=rbf;, score=0.711 total time=   0.0s
[CV 2/5] END ........C=100, gamma=1, kernel=rbf;, score=0.683 total time=   0.0s
[CV 3/5] END ........C=100, gamma=1, kernel=rbf;, score=0.659 total time=   0.0s
[CV 4/5] END ........C=100, gamma=1, kernel=rbf;, score=0.659 total time=   0.0s
[CV 5/5] END ........C=100, gamma=1, kernel=rbf;, score=0.817 total time=   0.0s
[CV 1/5] END ......C=100, gamma=0.1, kernel=rbf;, score=0.807 total time=   0.0s
[CV 2/5] END ......C=100, gamma=0.1, kernel=rbf;, score=0.707 total time=   0.0s
[CV 3/5] END ......C=100, gamma=0.1, kernel=rbf;, score=0.732 total time=   0.0s
[CV 4/5] END ......C=100, gamma=0.1, kernel=rbf;, score=0.756 total time=   0.0s
[CV 5/5] END ......C=100, gamma=0.1, kernel=rbf;, score=0.841 total time=   0.0s
[CV 1/5] END .....C=100, gamma=0.01, kernel=rbf;, score=0.807 total time=   0.0s
[CV 2/5] END .....C=100, gamma=0.01, kernel=rbf;, score=0.768 total time=   0.0s
[CV 3/5] END .....C=100, gamma=0.01, kernel=rbf;, score=0.756 total time=   0.0s
[CV 4/5] END .....C=100, gamma=0.01, kernel=rbf;, score=0.780 total time=   0.0s
[CV 5/5] END .....C=100, gamma=0.01, kernel=rbf;, score=0.805 total time=   0.0s
[CV 1/5] END ....C=100, gamma=0.001, kernel=rbf;, score=0.807 total time=   0.0s
[CV 2/5] END ....C=100, gamma=0.001, kernel=rbf;, score=0.768 total time=   0.0s
[CV 3/5] END ....C=100, gamma=0.001, kernel=rbf;, score=0.756 total time=   0.0s
[CV 4/5] END ....C=100, gamma=0.001, kernel=rbf;, score=0.780 total time=   0.0s
[CV 5/5] END ....C=100, gamma=0.001, kernel=rbf;, score=0.805 total time=   0.0s
[CV 1/5] END ...C=100, gamma=0.0001, kernel=rbf;, score=0.675 total time=   0.0s
[CV 2/5] END ...C=100, gamma=0.0001, kernel=rbf;, score=0.671 total time=   0.0s
[CV 3/5] END ...C=100, gamma=0.0001, kernel=rbf;, score=0.671 total time=   0.0s
[CV 4/5] END ...C=100, gamma=0.0001, kernel=rbf;, score=0.671 total time=   0.0s
[CV 5/5] END ...C=100, gamma=0.0001, kernel=rbf;, score=0.671 total time=   0.0s
{'C': 1, 'gamma': 0.1, 'kernel': 'rbf'}
```

In [74]:

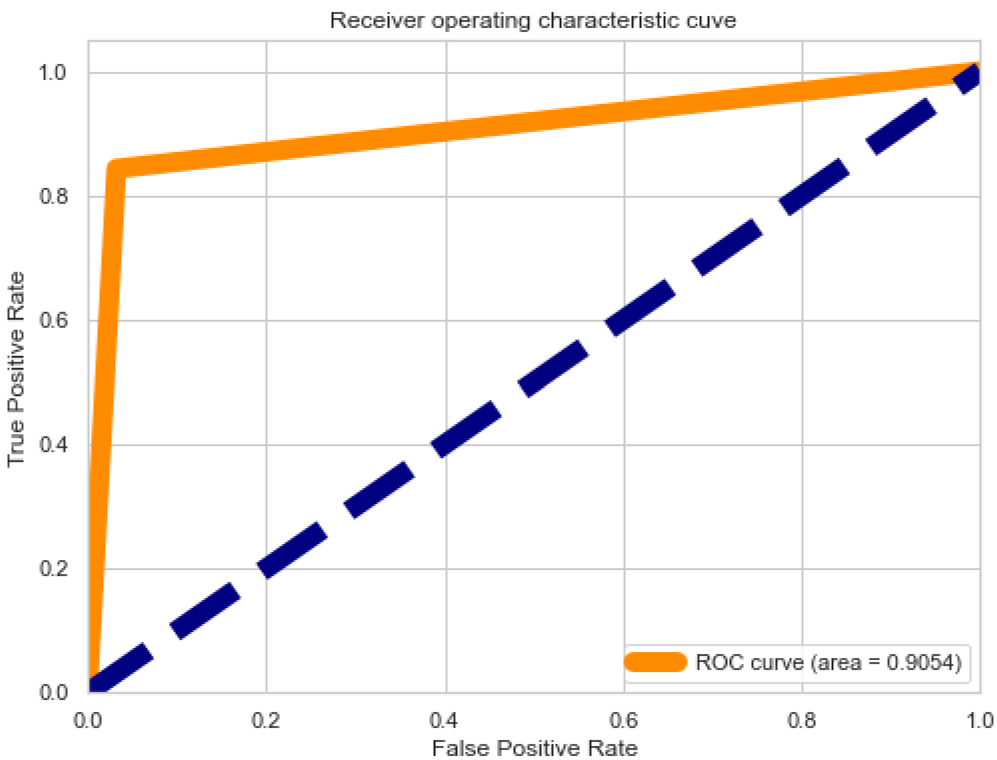
```
smv1=SVC(C=1,gamma=0.1,kernel='rbf')
smv1.fit(x_train,y_train)
pred_smv=smv1.predict(x_test)
score=accuracy_score(y_test,pred_smv)
print('accuracy_score= ',score*100)
cv_score=cross_val_score(smv1,x,y,cv=5)
cv_mean=cv_score.mean()
print('mean_cv value = ',cv_mean*100)
print(confusion_matrix(y_test,pred_y))
print(classification_report(y_test,pred_y))
```

```
accuracy_score=  86.20689655172413
mean_cv value =  80.94495535119283
[[ 30  27]
 [  1 145]]
              precision    recall  f1-score   support

           0       0.97      0.53      0.68        57
           1       0.84      0.99      0.91       146
```

```
    accuracy                           0.86       203
   macro avg       0.91      0.76      0.80       203
weighted avg       0.88      0.86      0.85       203
```

In [75]:

```
#aoc-roc curve
from sklearn.metrics import roc_curve,auc
fpr, tpr, thresholds = roc_curve(pred_smv,y_test)
roc_auc=auc(fpr,tpr)


plt.figure(figsize=(8, 6))
plt.plot( fpr, tpr,color='darkorange',lw=10,label='ROC curve (area = %0.4f)' % roc_auc)
plt.plot([0, 1], [0, 1],color='navy',lw=10,linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic cuve')
plt.legend(loc="lower right")
plt.show()
```




CONCLUSION

- area under the curve is 90.54percent which is very good value
- The support vector classifier SVC() giving the best accuracu value
- SVC Accuracy_score , F1_score , Classification_report, Confussion_matrix , and also ROC value is also shon in the above table
- last step of the projct,we know the better performance algorith which is Support Vector Classifier hyperparameter tuning also done, after that we need to save the model by usig pickel

Loading [MathJax]/extensions/Safe.js