Importing Libraries

In [1]:
```
# for numerical operations
import numpy as np

# for dataframe operations
import pandas as pd

# data visualizations
import matplotlib.pyplot as plt
import seaborn as sns

# for machine learning algorithms
import sklearn
import imblearn
```

Reading the Dataset

In [2]:
```
# lets import the dataset using the read_csv function
data = pd.read_csv('LoanData.csv')

# lets check the shape of the dataset
data.shape
```

Out[2]:
```
(614, 13)
```

In [3]:
```
# lets check the column names present in the dataset
data.columns
```

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

In [4]:
```
# lets check the head of the dataset
data.head()
```

Out[4]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	Loai
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	

Descriptive Statistics

In [5]:
```
# for numerical variables
data.describe()
```

Loading [MathJax]/extensions/Safe.js

...come	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History

	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 [6]:

```
# for categorical variables
data.describe(include = 'object')
```

Out[6]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	Property_Area	Loan_Status
count	614	601	611	599	614	582	614	614
unique	614	2	2	4	2	2	3	2
top	LP001002	Male	Yes	0	Graduate	No	Semiurban	Y
freq	1	489	398	345	480	500	233	422

In [7]:

```
data['Loan_Status'].value_counts()
```

Out[7]:

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

Data Cleaning

In [8]:

```
# checking no. of Missing values
data.isnull().sum()
```

Out[8]:

```
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 [9]:

```
# using mode values to impute categorical columns

data['Gender'] = data['Gender'].fillna(data['Gender'].mode()[0])
data['Married'] = data['Married'].fillna(data['Married'].mode()[0])
data['Dependents'] = data['Dependents'].fillna(data['Dependents'].mode()[0])
data['Self_Employed'] = data['Self_Employed'].fillna(data['Self_Employed'].mode()[0])
```

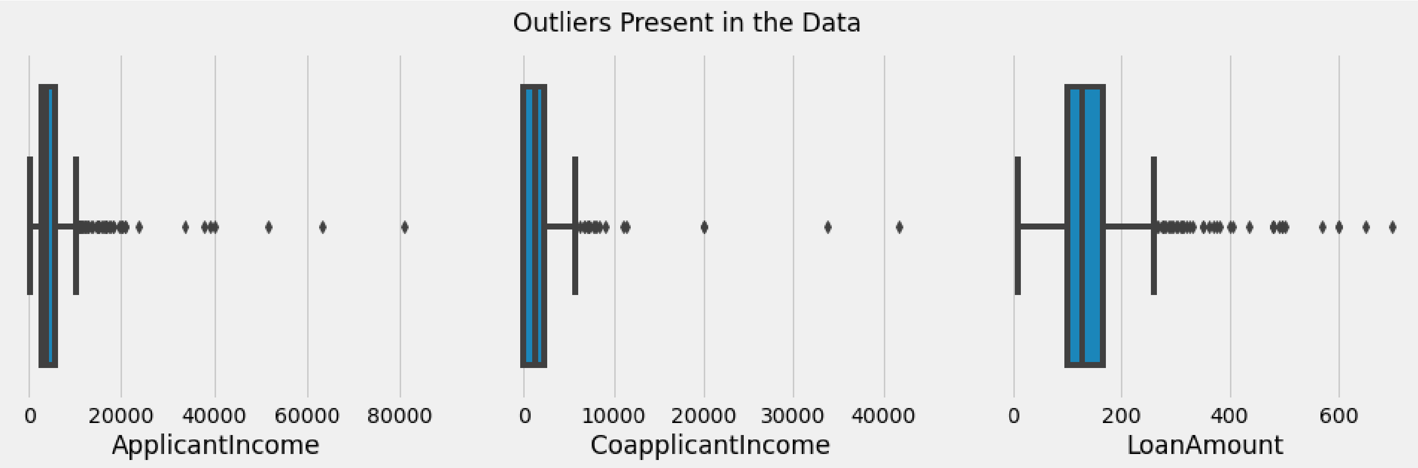
Loading [MathJax]/extensions/Safe.js

```
# using median values to impute the numerical columns

data['LoanAmount'] = data['LoanAmount'].fillna(data['LoanAmount'].median())
data['Loan_Amount_Term'] = data['Loan_Amount_Term'].fillna(data['Loan_Amount_Term'].mediar
data['Credit_History'] = data['Credit_History'].fillna(data['Credit_History'].median())

# lets check if there is any null values left or not
data.isnull().sum().sum()
```

Out[9]: 0

In [10]:

```
# lets visualize the outliers using Box Plot
import warnings
warnings.filterwarnings("ignore")
plt.style.use('fivethirtyeight')
plt.rcParams['figure.figsize'] = (15, 4)

plt.subplot(1, 3, 1)
sns.boxplot(data['ApplicantIncome'])

plt.subplot(1, 3, 2)
sns.boxplot(data['CoapplicantIncome'])

plt.subplot(1, 3, 3)
sns.boxplot(data['LoanAmount'])

plt.suptitle('Outliers Present in the Data')
plt.show()
```



In [11]:

```
# lets remove the outliers from the data

#lets check the shape before removing outliers
print("Before Removing Outliers ", data.shape)

# lets filter the customers having more than 25000 income
data = data[data['ApplicantIncome'] < 25000]

#lets check the shape after removing outliers
print("After Removing Outliers ", data.shape)
```

```
Before Removing Outliers  (614, 13)
After Removing Outliers  (607, 13)
```

In [12]:

```
# lets remove the outliers from co-applicant's Income
```

Loading [MathJax]/extensions/Safe.js

```python
#lets check the shape before removing outliers
print("Before Removing Outliers ", data.shape)

# lets filter the customers having more than 10000 coapplicant income
data = data[data['CoapplicantIncome'] < 10000]

#lets check the shape after removing outliers
print("After Removing Outliers ", data.shape)
```

```
Before Removing Outliers  (607, 13)
After Removing Outliers  (601, 13)
```

In [13]:

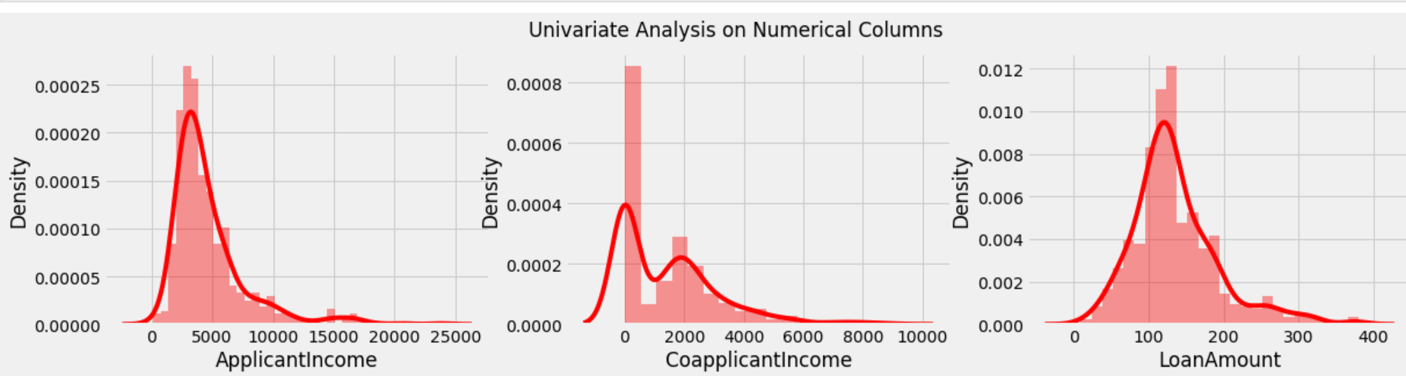
```python
# lets remove the outliers from Loan Amount

#lets check the shape before removing outliers
print("Before Removing Outliers ", data.shape)

# lets filter the customers having more than 400 loan amount
data = data[data['LoanAmount'] < 400]

#lets check the shape after removing outliers
print("After Removing Outliers ", data.shape)
```

```
Before Removing Outliers  (601, 13)
After Removing Outliers  (590, 13)
```

Univariate Data Analysis

In [14]:

```python
# Univariate Analysis on Numerical Columns

plt.rcParams['figure.figsize'] = (18, 4)
plt.subplot(1,3, 1)
sns.distplot(data['ApplicantIncome'], color = 'red')

plt.subplot(1,3, 2)
sns.distplot(data['CoapplicantIncome'], color = 'red')

plt.subplot(1,3, 3)
sns.distplot(data['LoanAmount'], color = 'red')

plt.suptitle('Univariate Analysis on Numerical Columns')
plt.show()
```

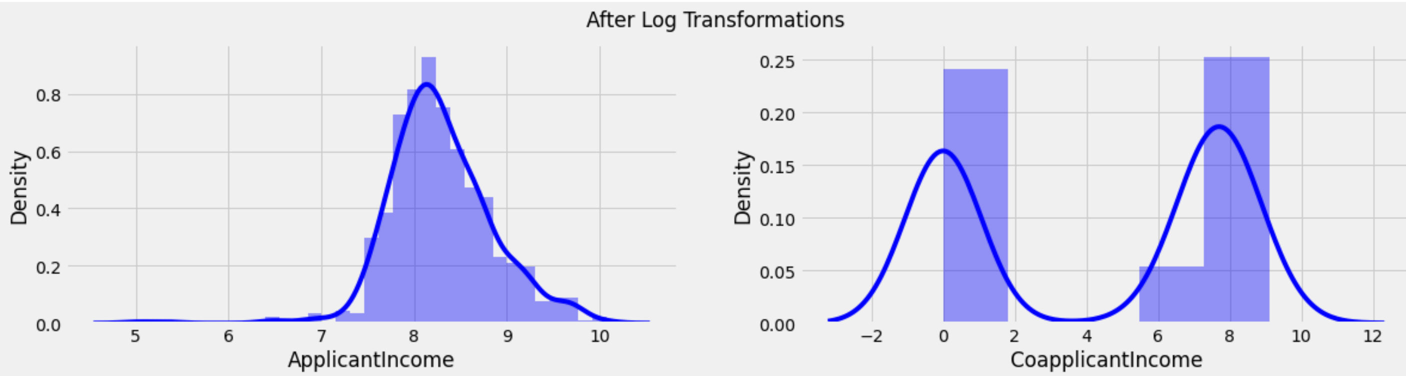


In [15]:

```python
# lets remove skewness from ApplicantIncome and Coapplicant Income, as it can add bias to

import warnings
warnings.filterwarnings('ignore')
plt.rcParams['figure.figsize'] = (18, 4)
```

Loading [MathJax]/extensions/Safe.js

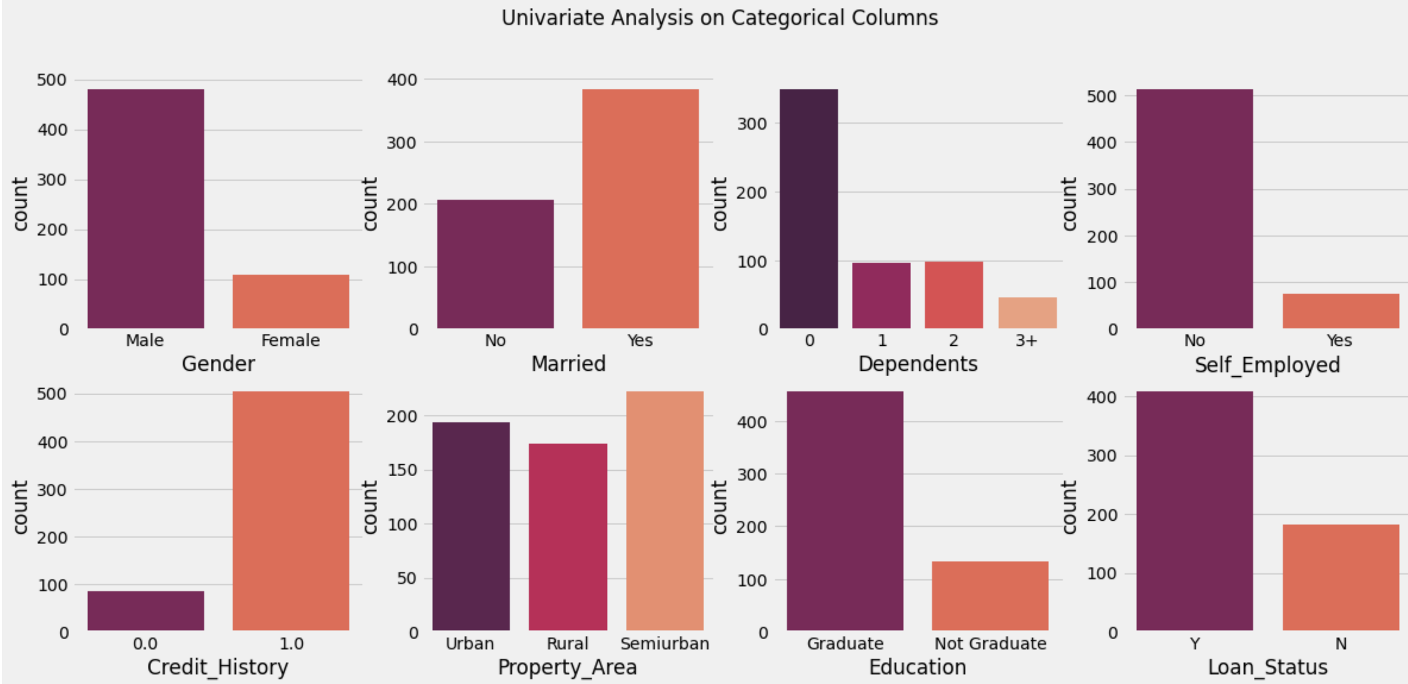
```python
# lets apply log transformation to remove skewness
data['ApplicantIncome'] = np.log(data['ApplicantIncome'])
data['CoapplicantIncome'] = np.log1p(data['CoapplicantIncome'])

# lets plot them and check whether the skewness is removed or not
plt.subplot(1, 2, 1)
sns.distplot(data['ApplicantIncome'], color = 'blue')

plt.subplot(1, 2, 2)
sns.distplot(data['CoapplicantIncome'], color = 'blue')

plt.suptitle('After Log Transformations')
plt.show()
```



In [16]:

```python
## Univariate Analysis on Categorical Columns

plt.rcParams['figure.figsize'] = (18,8)

plt.subplot(2, 4, 1)
sns.countplot(data['Gender'], palette = 'rocket')

plt.subplot(2, 4, 2)
sns.countplot(data['Married'], palette = 'rocket')

plt.subplot(2, 4, 3)
sns.countplot(data['Dependents'], palette = 'rocket')

plt.subplot(2, 4, 4)
sns.countplot(data['Self_Employed'], palette = 'rocket')

plt.subplot(2, 4, 5)
sns.countplot(data['Credit_History'], palette = 'rocket')

plt.subplot(2, 4, 6)
sns.countplot(data['Property_Area'], palette = 'rocket')

plt.subplot(2, 4, 7)
sns.countplot(data['Education'], palette = 'rocket')

plt.subplot(2, 4, 8)
sns.countplot(data['Loan_Status'], palette = 'rocket')

plt.suptitle('Univariate Analysis on Categorical Columns')
plt.show()
```

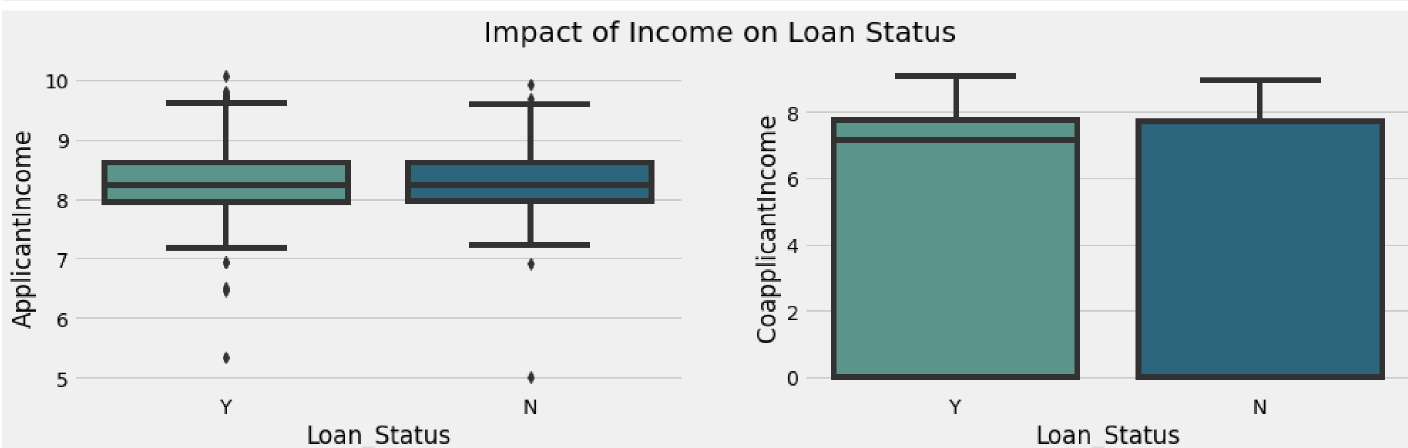
Loading [MathJax]/extensions/Safe.js



Bivariate Data Analysis

In [17]:

```
### Lets check the Impact of Income of Applicant and Co-applicant on Loan Status

plt.rcParams['figure.figsize'] = (15, 4)

plt.subplot(1, 2, 1)
sns.boxplot(data['Loan_Status'], data['ApplicantIncome'], palette = 'crest')

plt.subplot(1, 2, 2)
sns.boxplot(data['Loan_Status'], data['CoapplicantIncome'], palette = 'crest')

plt.suptitle('Impact of Income on Loan Status\n', fontsize = 20)
plt.show()
```

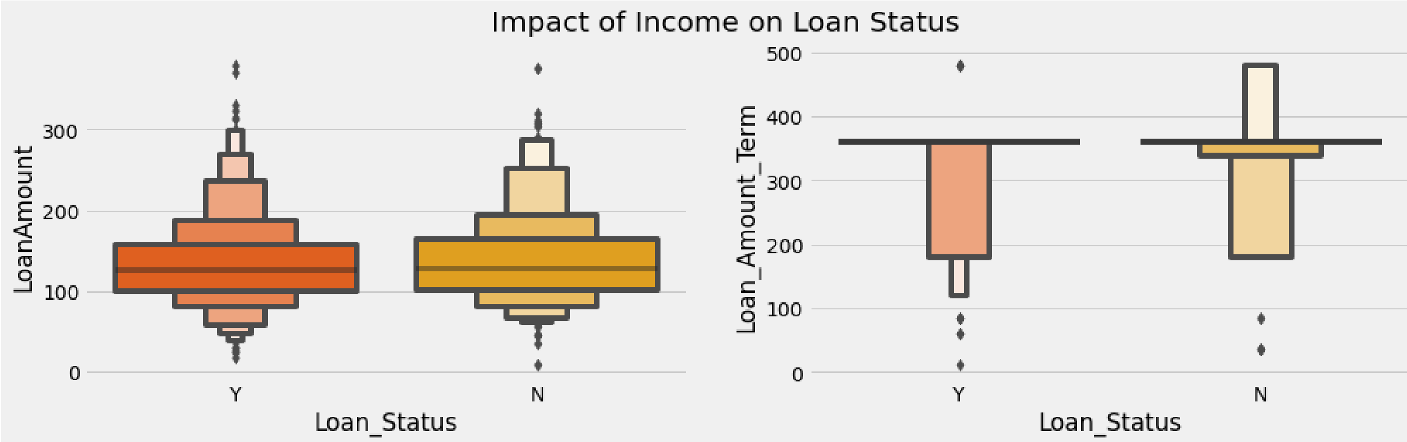


In [18]:

```
# lets check the Impact of Amount on Loan Status

plt.rcParams['figure.figsize'] = (15, 4)

plt.subplot(1, 2, 1)
sns.boxenplot(data['Loan_Status'], data['LoanAmount'], palette = 'autumn')

plt.subplot(1, 2, 2)
sns.boxenplot(data['Loan_Status'], data['Loan_Amount_Term'], palette = 'autumn')
```

Loading [MathJax]/extensions/Safe.js

```
plt.suptitle('Impact of Income on Loan Status\n', fontsize = 20)
plt.show()
```



Comparing Categorical Data with Target Columns

In [19]:

```
# lets compare all the Categories with respect to the Loan Status to understand the Overal

print("Impact of Marraige on Loan Status")
print(pd.crosstab(data['Loan_Status'], data['Married']))
print('\n')

print("Impact of Dependents on Loan Status")
print(pd.crosstab(data['Loan_Status'], data['Dependents']))
print('\n')

print("Impact of Education on Loan Status")
print(pd.crosstab(data['Loan_Status'], data['Education']))
print('\n')

print("Impact of Employment on Loan Status")
print(pd.crosstab(data['Loan_Status'], data['Self_Employed']))
print('\n')

print("Impact of Property on Loan Status")
print(pd.crosstab(data['Loan_Status'], data['Property_Area']))
```

```
Impact of Marraige on Loan Status
Married       No  Yes
Loan_Status
N             76  106
Y            130  278


Impact of Dependents on Loan Status
Dependents     0   1   2  3+
Loan_Status
N            110  33  24  15
Y            240  63  74  31


Impact of Education on Loan Status
Education    Graduate  Not Graduate
Loan_Status
N                 130            52
Y                 326            82


Impact of Employment on Loan Status
```

Loading [MathJax]/extensions/Safe.js

```
Self_Employed   No  Yes
Loan_Status
N              157   25
Y              357   51


Impact of Property on Loan Status
Property_Area  Rural  Semiurban  Urban
Loan_Status
N                 66         51     65
Y                108        171    129
```

Data Preparation

In [20]:

```
# lets check the columns which are of object data types
data.select_dtypes('object').head()
```

Out[20]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	Property_Area	Loan_Status
0	LP001002	Male	No	0	Graduate	No	Urban	Y
1	LP001003	Male	Yes	1	Graduate	No	Rural	N
2	LP001005	Male	Yes	0	Graduate	Yes	Urban	Y
3	LP001006	Male	Yes	0	Not Graduate	No	Urban	Y
4	LP001008	Male	No	0	Graduate	No	Urban	Y

In [21]:

```
# lets delete the loan Id column from the data as this column has no relation with loan st

# lets check the shape of the data before deleting the columns
print("Before Deleting Columns :", data.shape)

data = data.drop(['Loan_ID'], axis = 1)

# lets check the shape of the data after deleting the columns
print("After Deleting Columns :", data.shape)
```

```
Before Deleting Columns : (590, 13)
After Deleting Columns : (590, 12)
```

In [22]:

```
# lets encode other columns

data['Gender'] = data['Gender'].replace(('Male','Female'),(1, 0))
data['Married'] = data['Married'].replace(('Yes','No'),(1, 0))
data['Education'] = data['Education'].replace(('Graduate','Not Graduate'), (1, 0))
data['Self_Employed'] = data['Self_Employed'].replace(('Yes','No'), (1, 0))
data['Loan_Status'] = data['Loan_Status'].replace(('Y','N'), (1, 0))

# as seen above that Urban and Semi Urban Property have very similar Impact on Loan Status
data['Property_Area'] = data['Property_Area'].replace(('Urban','Semiurban', 'Rural'),(1, 1

# as seen above that apart from 0 dependents, all are similar hence, we merge them to avoi
data['Dependents'] = data['Dependents'].replace(('0', '1', '2', '3+'), (0, 1, 1, 1))

# lets check whether there is any object column left
data.select_dtypes('object').columns
```

Out[22]:

```
Index([], dtype='object')
```

Loading [MathJax]/extensions/Safe.js

In [23]:
```
# lets split the Target column from the Data

y = data['Loan_Status']
x = data.drop(['Loan_Status'], axis = 1)

# lets check the shape of x and y
print("Shape of x :", x.shape)
print("Shape of y :", y.shape)
```

```
Shape of x : (590, 11)
Shape of y : (590,)
```

Resampling for Balancing the Data

In [24]:
```
# It is very important to resample the data, as the Target class is Highly imbalanced.
# Here We are going to use Over Sampling Technique to resample the data.
# lets import the SMOTE algorithm to do the same.

from imblearn.over_sampling import SMOTE

x_resample, y_resample  = SMOTE().fit_resample(x, y.values.ravel())

# lets print the shape of x and y after resampling it
print(x_resample.shape)
print(y_resample.shape)
```

```
(816, 11)
(816,)
```

In [25]:
```
# lets also check the value counts of our target variable4

print("Before Resampling :")
print(y.value_counts())

print("After Resampling :")
y_resample = pd.DataFrame(y_resample)
print(y_resample[0].value_counts())
```

```
Before Resampling :
1    408
0    182
Name: Loan_Status, dtype: int64
After Resampling :
1    408
0    408
Name: 0, dtype: int64
```

In [26]:
```
# lets split the test data from the training data

from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x_resample, y_resample, test_size = 0.

# lets print the shapes again
print("Shape of the x Train :", x_train.shape)
print("Shape of the y Train :", y_train.shape)
print("Shape of the x Test :", x_test.shape)
print("Shape of the y Test :", y_test.shape)
```

```
Shape of the x Train : (652, 11)
Shape of the y Train : (652, 1)
```

Loading [MathJax]/extensions/Safe.js

```
Shape of the x Test : (164, 11)
Shape of the y Test : (164, 1)
```

Machine Learning Modelling

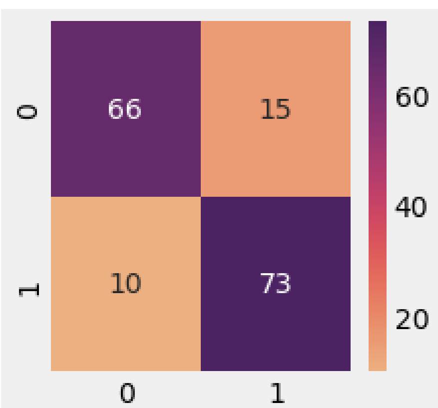
In [27]:

```python
# lets apply Logistic Regression

from sklearn.linear_model import LogisticRegression

model = LogisticRegression(random_state = 0)
model.fit(x_train, y_train)

y_pred = model.predict(x_test)

print("Training Accuracy :", model.score(x_train, y_train))
print("Testing Accuracy :", model.score(x_test, y_test))
```

```
Training Accuracy : 0.7638036809815951
Testing Accuracy : 0.8475609756097561
```

In [28]:

```python
# lets analyze the Performance using Confusion matrix

from sklearn.metrics import confusion_matrix, classification_report

cm = confusion_matrix(y_test, y_pred)
plt.rcParams['figure.figsize'] = (3, 3)
sns.heatmap(cm, annot = True, cmap = 'flare', fmt = '.8g')
plt.show()

# lets also use classification report for performance analysis
cr = classification_report(y_test, y_pred)
print(cr)
```




```
              precision    recall  f1-score   support

           0       0.87      0.81      0.84        81
           1       0.83      0.88      0.85        83

    accuracy                           0.85       164
   macro avg       0.85      0.85      0.85       164
weighted avg       0.85      0.85      0.85       164
```

Applying Gradient Boosting

In [29]:

```python
# lets apply DecisionTrees

from sklearn.ensemble import GradientBoostingClassifier
```

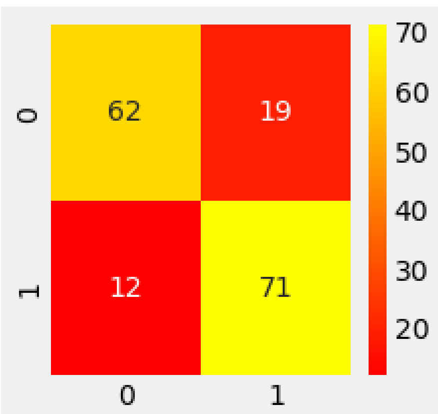
Loading [MathJax]/extensions/Safe.js

```
model =  GradientBoostingClassifier()
model.fit(x_train, y_train)

y_pred = model.predict(x_test)

print("Training Accuracy :", model.score(x_train, y_train))
print("Testing Accuracy :", model.score(x_test, y_test))
```

```
Training Accuracy : 0.9095092024539877
Testing Accuracy : 0.8109756097560976
```

In [30]:

```
# lets analyze the Performance using Confusion matrix

cm = confusion_matrix(y_test, y_pred)
plt.rcParams['figure.figsize'] = (3, 3)
sns.heatmap(cm, annot = True, cmap = 'autumn', fmt = '.8g')
plt.show()

# lets also use classification report for performance analysis
cr = classification_report(y_test, y_pred)
print(cr)
```




```
              precision    recall  f1-score   support

           0       0.84      0.77      0.80        81
           1       0.79      0.86      0.82        83

    accuracy                           0.81       164
   macro avg       0.81      0.81      0.81       164
weighted avg       0.81      0.81      0.81       164
```

In [31]:

```
from sklearn.model_selection import cross_val_score

clf = GradientBoostingClassifier(random_state = 0)
scores = cross_val_score(clf, x_train, y_train, cv=10)
print(scores)
```

```
[0.77272727 0.8030303  0.8        0.76923077 0.81538462 0.76923077
 0.8        0.76923077 0.81538462 0.78461538]
```

Loading [MathJax]/extensions/Safe.js