Predicting Loan Cases Using Decision Tree

Importing all the necessary libraries using import

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

In [2]:
```
dataset = pd.read_csv(r"C:\Users\LenovoX260\Downloads\train_ctrUa4K.csv")
```

In [3]:
```
dataset.head()
```

Out[3]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantI
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 [4]:
```
dataset.shape
```

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

In [5]:
```
dataset = dataset.sample(n=550, random_state = 17)
```

In [6]:
```
dataset.to_csv('AdaobiEjiasiL_2306317.csv')
```

In [7]:
```
data = pd.read_csv('AdaobiEjiasiL_2306317.csv')
```

In [8]:
```
data.head()
```

Out[8]:

	Unnamed: 0	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome
0	132	LP001478	Male	No	0	Graduate	No	2718
1	451	LP002447	Male	Yes	2	Not Graduate	No	1958
2	394	LP002266	Male	Yes	2	Graduate	No	3100
3	415	LP002337	Female	No	0	Graduate	No	2995
4	326	LP002068	Male	No	0	Graduate	No	4917

In [9]:

```
data=data.drop('Unnamed: 0', axis = 1)
```

Using and explaining the following DataFrame functions/properties on the data.

- describe()
- size
- ndim
- shape

In [10]:

```
print(data.describe())
```

```
       ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
count       550.000000         550.000000  533.000000        538.000000
mean       5466.565455        1635.072582  145.457786        342.892193
std        6354.681175        3013.571911   85.562802         63.442106
min         150.000000           0.000000   17.000000         12.000000
25%        2904.250000           0.000000  100.000000        360.000000
50%        3768.500000        1188.500000  126.000000        360.000000
75%        5813.500000        2297.250000  165.000000        360.000000
max       81000.000000       41667.000000  700.000000        480.000000

       Credit_History
count      506.000000
mean         0.835968
std          0.370671
min          0.000000
25%          1.000000
50%          1.000000
75%          1.000000
max          1.000000
```

data.describe() function shows the descriptive statistics of all numerical attributes in the dataset

It shows the count, mean, standard deviation, minimum, first quartile, second quartile, third quartile and maximum value of the data set.

In [11]:

```
print(data.size)
```

```
7150
```

data.size shows the count of the total number of rows multiple by the total number of columns. To confirm this lets see the outcome of data.shape

In [12]:
```
print(data.shape)
```

```
(550, 13)
```

data.shape shows tells us that there are 550 rows and 13 columns. So 550*13 = 7150. Which confirms 7150 for the data size.

In [13]:
```
print(data.ndim)
```

```
2
```

data.ndim is used to display the number of dimensions of a data frame. The ouput 2 shows that this is a two dimensional dataframe

In [14]:
```
print(data.shape)
```

```
(550, 13)
```

data.shape shows the number of columns and rows in the dataset. In the dataset there are 550 columns and 13 rows

Looking for the difference between dimensions of the original dataset and the new dataset. If yes, what is the difference?

No, there is no difference between the original dataset and the new dataset, there are both two dimensional dataset

What are the possible values 'Education' can take? Write Code to display all possible values 'Education'

In [15]:
```
data.Education.unique()
```

Out[15]:
```
array(['Graduate', 'Not Graduate'], dtype=object)
```

Data Analysis

In [16]:
```
columns = data.columns
columns
```

Out[16]:
```
Index(['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education',
       'Self_Employed', 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History', 'Property_Area', 'Loan_Status'],
      dtype='object')
```

In [17]:
```
data.head()
```

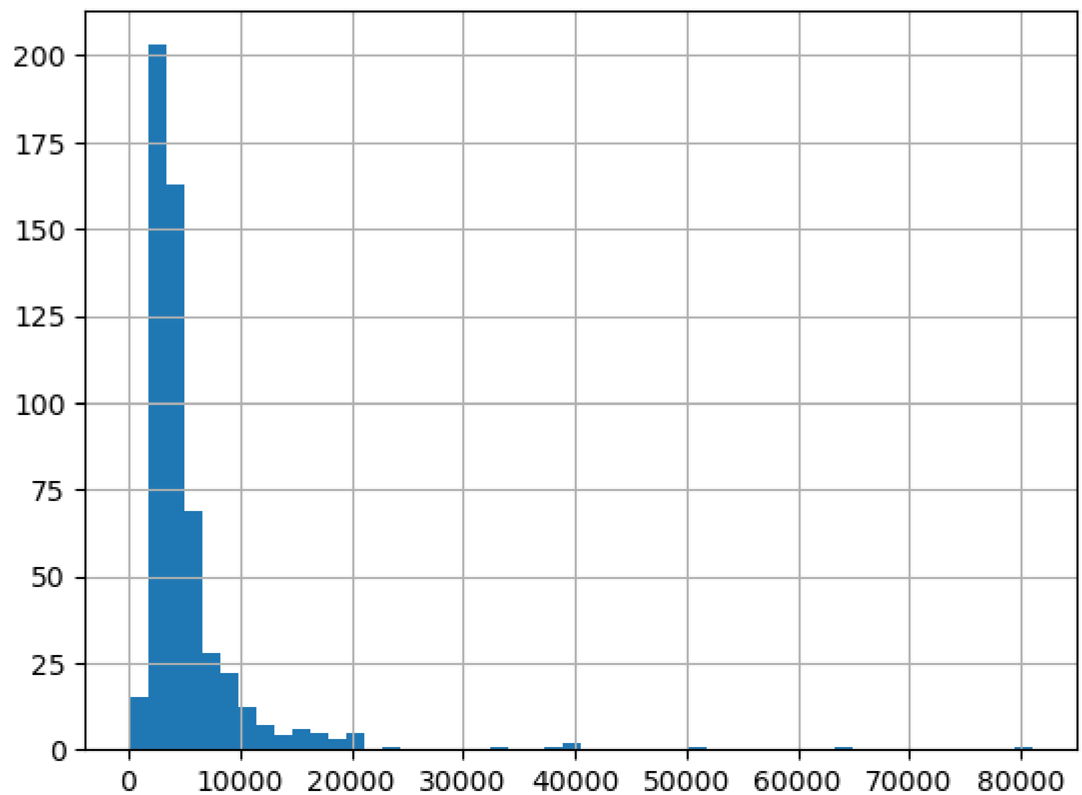
Out[17]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantI
0	LP001478	Male	No	0	Graduate	No	2718	
1	LP002447	Male	Yes	2	Not Graduate	No	1958	
2	LP002266	Male	Yes	2	Graduate	No	3100	
3	LP002337	Female	No	0	Graduate	No	2995	
4	LP002068	Male	No	0	Graduate	No	4917	

In [18]:

```
data['ApplicantIncome'].hist(bins=50)
```

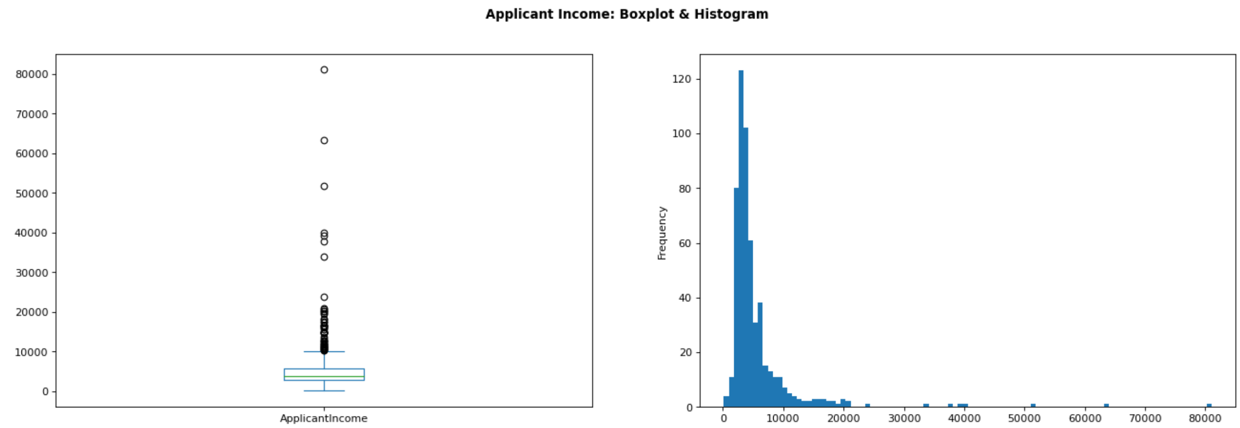
Out[18]: <AxesSubplot:>



Question 4: Use boxplot and histogram on 'ApplicantIncome' to visualise its distribution

In [19]:

```
fig, axs = plt.subplots(1,2, figsize=(20, 6), dpi=80)
data.ApplicantIncome.plot(kind='box', ax=axs[0])
data.ApplicantIncome.plot(kind='hist', ax=axs[1], bins=100)
plt.suptitle('Applicant Income: Boxplot & Histogram', fontweight='bold')
plt.show()
```



What are the extreme values? Are there any outliers(s) exist in this dataset? Explain with example based on the 'ApplicantIncome'?

Step 1 - To get the outliners, we will use the data.describe() to show the descriptive statistic of the numerical values in 'ApplicantIncome'

In [20]:
```
data.ApplicantIncome.describe()
```

Out[20]:
```
count      550.000000
mean      5466.565455
std       6354.681175
min        150.000000
25%       2904.250000
50%       3768.500000
75%       5813.500000
max      81000.000000
Name: ApplicantIncome, dtype: float64
```

In statistics outliner = 1.5 * IQR (interquartile range) The interquartile range in the Applicant Income data is difference between the second quartile (25%) and the fourth quartile (75%). That is 5813.500000 - 2904.250000

In [21]:
```
Twenty_Five_Percentile = 2904.25
Seventy_Five_Percentile = 5813.5
print(Twenty_Five_Percentile,Seventy_Five_Percentile)
```

```
2904.25 5813.5
```

In [22]:
```
IQR = Seventy_Five_Percentile-Twenty_Five_Percentile
```

In [23]:
```
print(IQR)
```

```
2909.25
```

In [24]:
```
Outliner = 1.5 * IQR
```

In [25]:
```
print(Outliner)
```

```
4363.875
```

To get the Outliners above the fourth quartile we add the value of the Outliner + the value of the fourth quartile (5813.5) Also, to get the Outliners below the second quartile we substract the value of the second quartile - with the value of the Outliner. Please see demo below;

In [26]:
```
Seventy_Five_Percentile + Outliner
```

Out[26]:
```
10177.375
```

In [27]:
```
Twenty_Five_Percentile - Outliner
```

Out[27]:
```
-1459.625
```

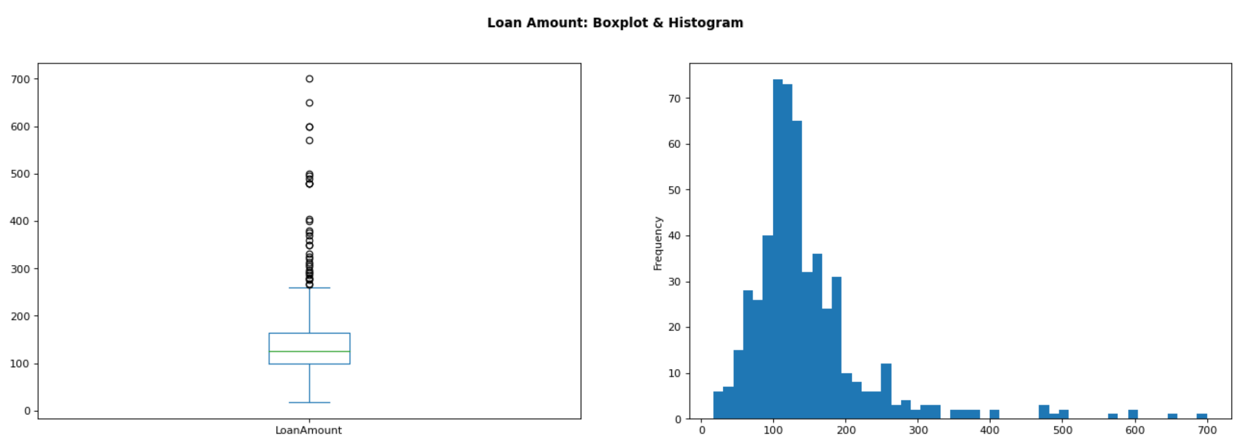
This means that all numbers above tenthousand and below 1.5 are outliners in the 'ApplicantIncome dataset' Ideally, Outliners are all numbers above 1.5 * INQ on the fourth quartile and all numbers below 1.5 on the second quartile

Are the results of both the plots comparable? Are there any differences in the two plots? What are the key differences?

- Comparable Yes, they are both comparable because they both have outliners
- Differences The box plot shows a statistical view of minimum, second quartile, third quartile, fourth quartile and maximum value in the dataset while the histogram shows the frequency distribution of the data and the occurenace of certain values.

Try-It-Yourself: Use Histogram and Box plot on 'LoanAmount' and observe extreme values.

In [28]:
```
fig, axs = plt.subplots(1,2, figsize=(20, 6), dpi=80)
data.LoanAmount.plot(kind='box', ax=axs[0])
data.LoanAmount.plot(kind='hist', ax=axs[1], bins=50)
plt.suptitle('Loan Amount: Boxplot & Histogram', fontweight='bold')
plt.show()
```



Categorical variable analysis

In [29]:
```
data['Credit_History'].value_counts()
```

Out[29]:
```
1.0    423
0.0     83
Name: Credit_History, dtype: int64
```

In [30]:
```
credit_history = data['Credit_History'].value_counts(ascending=True)
loan_probability = data.pivot_table(values='Loan_Status', index=['Credit_History'],
 aggfunc=lambda x: x.map({'Y':1,'N':0}).mean())
print('Frequency Table for Credit History:')
print(credit_history)
print('\nProbability of getting loan for each Credit History class:')
print(loan_probability)
```

```
Frequency Table for Credit History:
0.0     83
1.0    423
Name: Credit_History, dtype: int64

Probability of getting loan for each Credit History class:
                Loan_Status
Credit_History
0.0                0.084337
1.0                0.796690
```

In [31]:
```
data['Loan_Status'].value_counts()
```

Out[31]:
```
Y    376
N    174
Name: Loan_Status, dtype: int64
```

In [32]:
```
data.shape
```

Out[32]:
```
(550, 13)
```

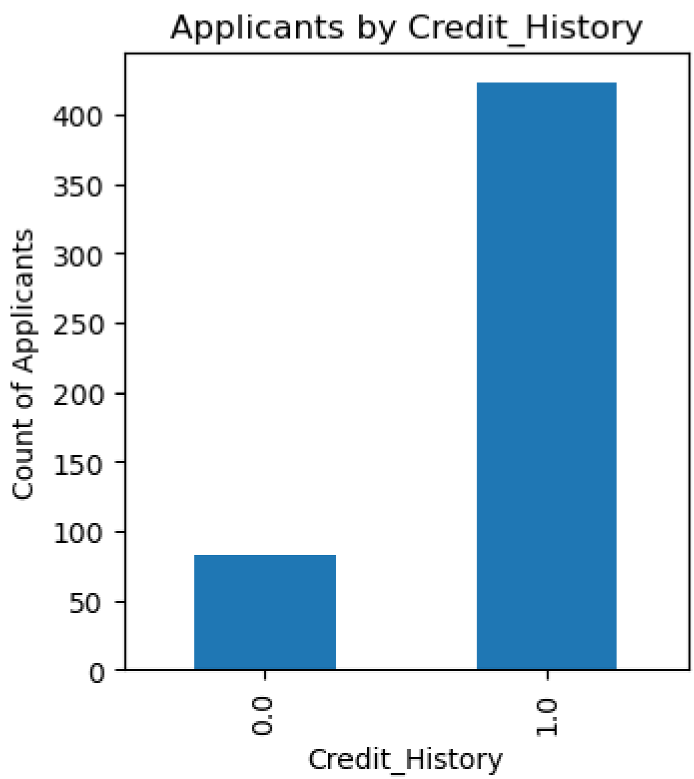
In [33]:
```
data.head()
```

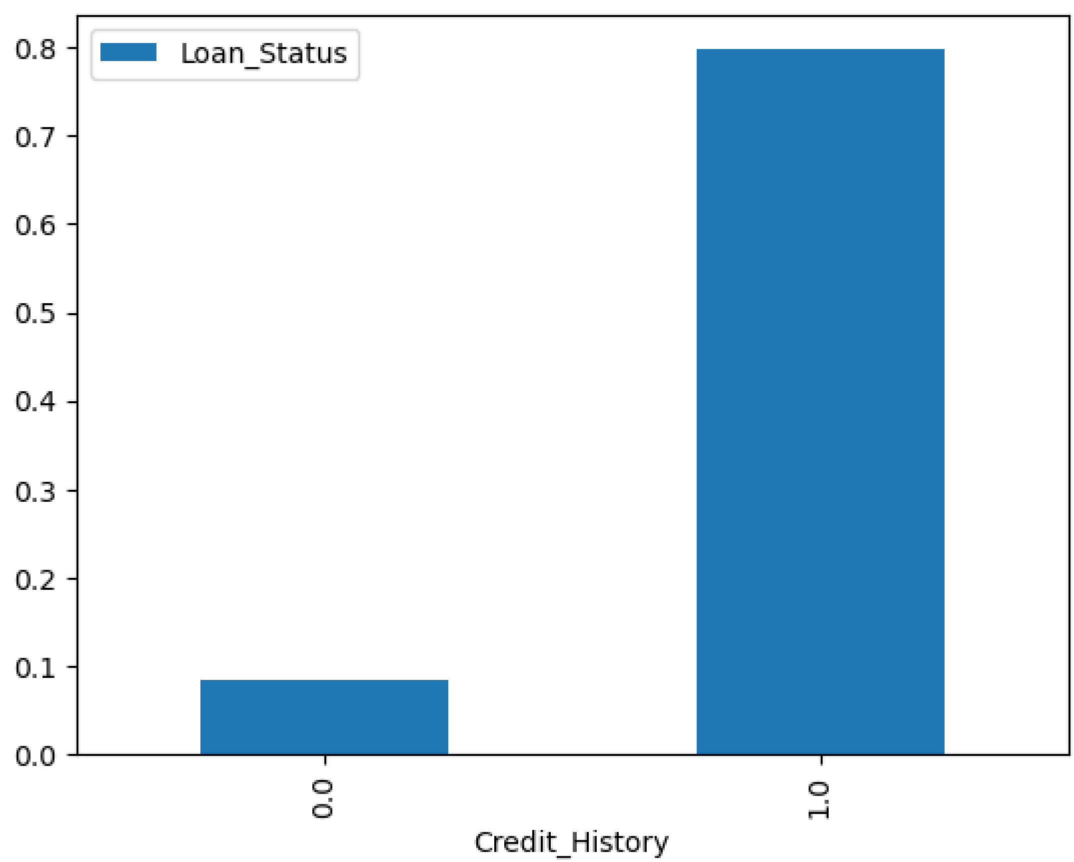
Out[33]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantI
0	LP001478	Male	No	0	Graduate	No	2718	
1	LP002447	Male	Yes	2	Not Graduate	No	1958	
2	LP002266	Male	Yes	2	Graduate	No	3100	
3	LP002337	Female	No	0	Graduate	No	2995	
4	LP002068	Male	No	0	Graduate	No	4917	

In [34]:
```
fig = plt.figure(figsize=(8,4))
ax1 = fig.add_subplot(121)
ax1.set_xlabel('Credit_History')
ax1.set_ylabel('Count of Applicants')
ax1.set_title("Applicants by Credit_History")
credit_history.plot(kind='bar')
plt.show()
ax2 = fig.add_subplot(122)
ax2.set_xlabel('Credit_History')
```

```
ax2.set_ylabel('Probability of getting loan')
ax2.set_title("Probability of getting loan by credit history")
loan_probability.plot(kind = 'bar')
plt.show()
```





Data Pre-processing

```
-Dealing with missing values
-Outliers and extreme values
-Dealing with non-numerical fields
```

In [35]:
```
data['Gender'].value_counts()
```

Out[35]:
```
Male      440
Female     99
Name: Gender, dtype: int64
```

Filling in missing values by mean

In [36]:
```
data.apply(lambda x: sum(x.isnull()), axis=0)
```

Out[36]:
```
Loan_ID               0
Gender               11
Married               1
Dependents           13
Education             0
Self_Employed        30
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           17
Loan_Amount_Term     12
Credit_History       44
Property_Area         0
Loan_Status           0
dtype: int64
```

In [37]:
```
data.head()
```

Out[37]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantI
0	LP001478	Male	No	0	Graduate	No	2718	
1	LP002447	Male	Yes	2	Not Graduate	No	1958	
2	LP002266	Male	Yes	2	Graduate	No	3100	
3	LP002337	Female	No	0	Graduate	No	2995	
4	LP002068	Male	No	0	Graduate	No	4917	

In [38]:
```
data['LoanAmount'].fillna(data['LoanAmount'].mean(), inplace = True)
```

In [39]:
```
data.head()
```

Out[39]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantI
0	LP001478	Male	No	0	Graduate	No	2718	
1	LP002447	Male	Yes	2	Not Graduate	No	1958	
2	LP002266	Male	Yes	2	Graduate	No	3100	
3	LP002337	Female	No	0	Graduate	No	2995	
4	LP002068	Male	No	0	Graduate	No	4917	

In [40]:

```
data.apply(lambda x: sum(x.isnull()), axis=0)
```

Out[40]:

```
Loan_ID               0
Gender               11
Married               1
Dependents           13
Education             0
Self_Employed        30
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount            0
Loan_Amount_Term     12
Credit_History       44
Property_Area         0
Loan_Status           0
dtype: int64
```

In [41]:

```
data.shape
```

Out[41]:

```
(550, 13)
```

In [42]:

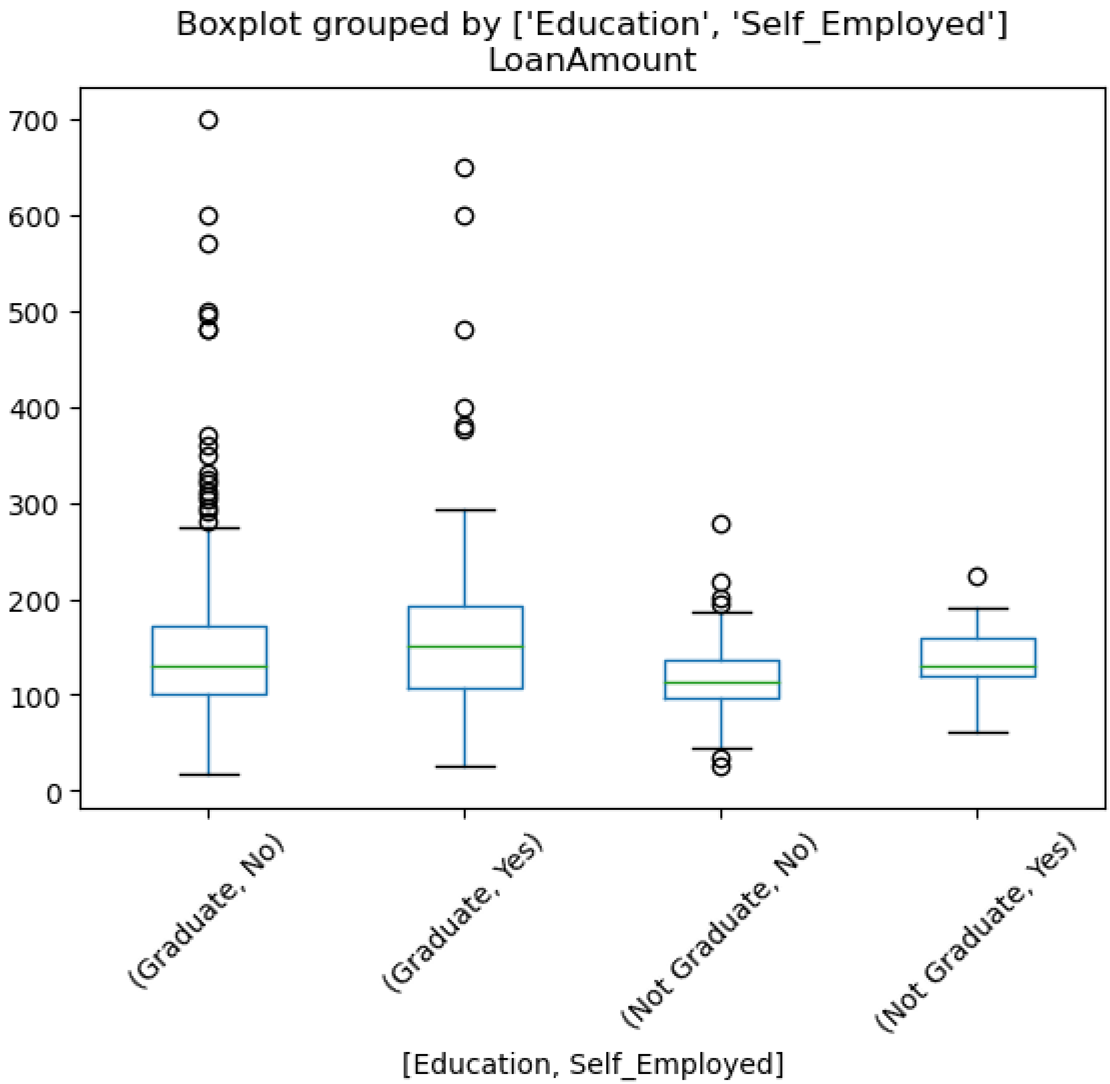
```
data.to_csv('new_train.csv')
```

In [43]:

```
data.boxplot(column='LoanAmount', by = ['Education','Self_Employed'],
 grid=False, rot = 45, fontsize = 10)
```

Out[43]:

```
<AxesSubplot:title={'center':'LoanAmount'}, xlabel='[Education, Self_Employed]'>
```



Impute the Values

In [44]:
```
data['Self_Employed'].value_counts()
```

Out[44]:
```
No     449
Yes     71
Name: Self_Employed, dtype: int64
```

In [45]:
```
data['Self_Employed'].fillna('No', inplace=True)
```

In [46]:
```
data['Self_Employed'].value_counts()
```

Out[46]:
```
No     479
Yes     71
Name: Self_Employed, dtype: int64
```

In [47]:
```
data.apply(lambda x: sum(x.isnull()), axis=0)
```

Out[47]:
```
Loan_ID              0
Gender              11
Married              1
Dependents          13
Education            0
Self_Employed        0
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           0
Loan_Amount_Term    12
Credit_History      44
Property_Area        0
Loan_Status          0
dtype: int64
```

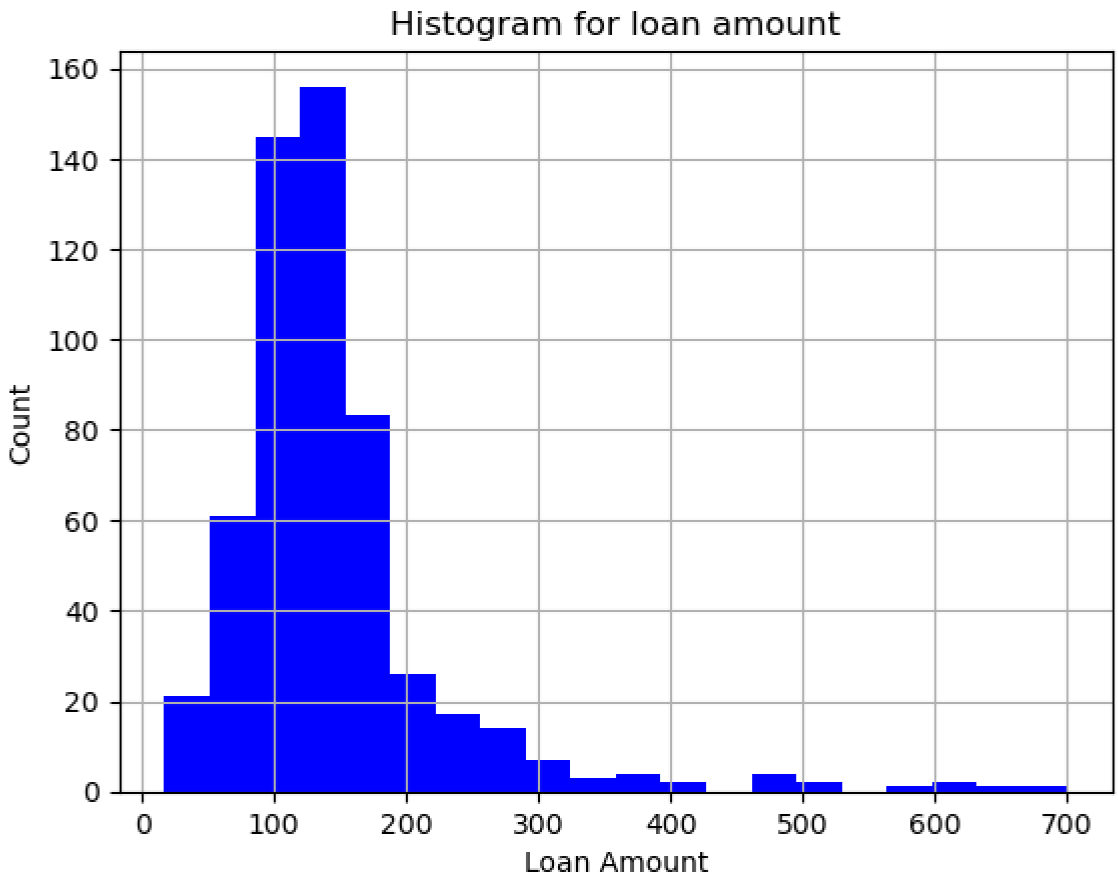
Dealing with Outliners

In [48]:
```
data.describe()
```

Out[48]:

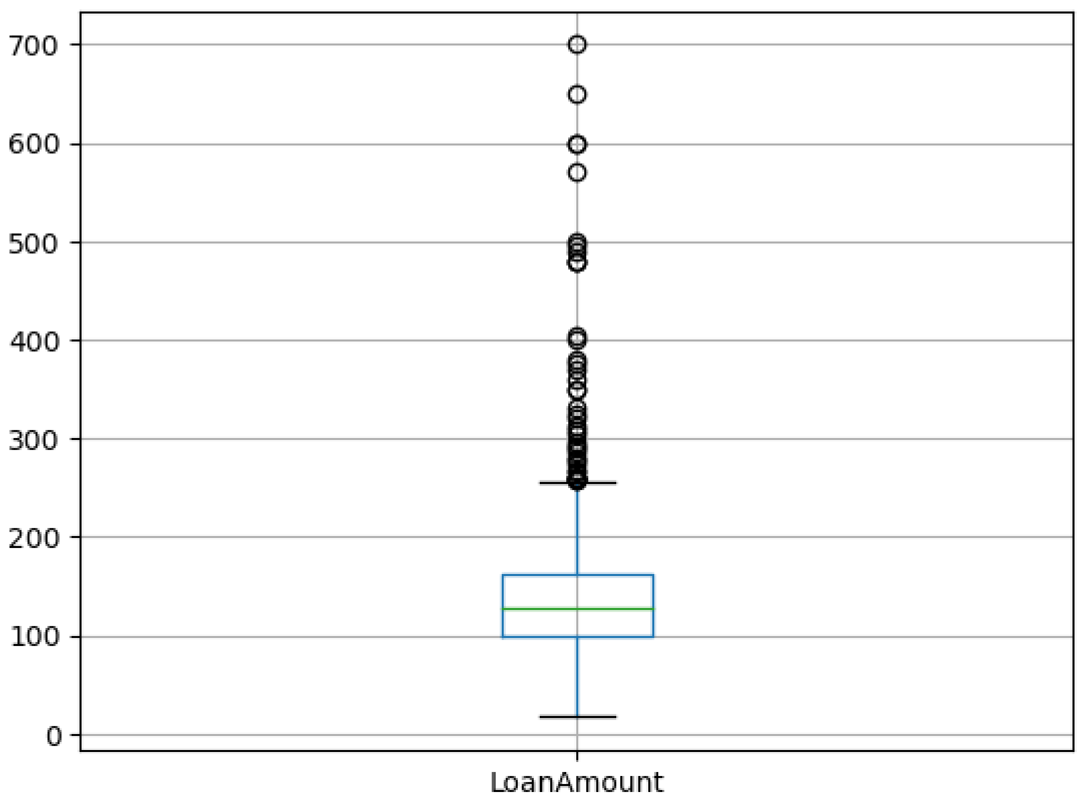
	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	550.000000	550.000000	550.000000	538.000000	506.000000
mean	5466.565455	1635.072582	145.457786	342.892193	0.835968
std	6354.681175	3013.571911	84.227642	63.442106	0.370671
min	150.000000	0.000000	17.000000	12.000000	0.000000
25%	2904.250000	0.000000	100.000000	360.000000	1.000000
50%	3768.500000	1188.500000	128.000000	360.000000	1.000000
75%	5813.500000	2297.250000	162.000000	360.000000	1.000000
max	81000.000000	41667.000000	700.000000	480.000000	1.000000

In [49]:
```
plt.hist(data['LoanAmount'], 20, facecolor='b')
plt.xlabel('Loan Amount')
plt.ylabel('Count')
plt.title('Histogram for loan amount')
plt.grid(True)
plt.show()
```



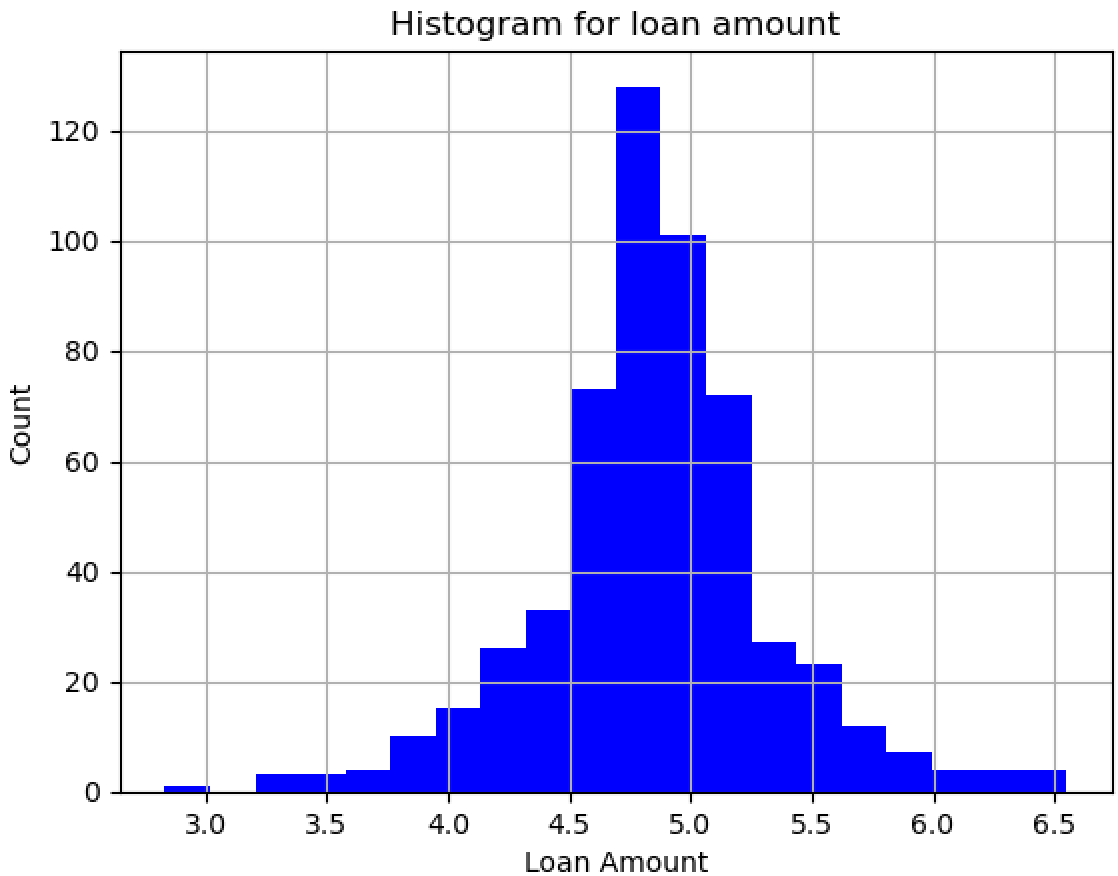
In [50]:
```
data.boxplot(column='LoanAmount')
```

Out[50]:
```
<AxesSubplot:>
```



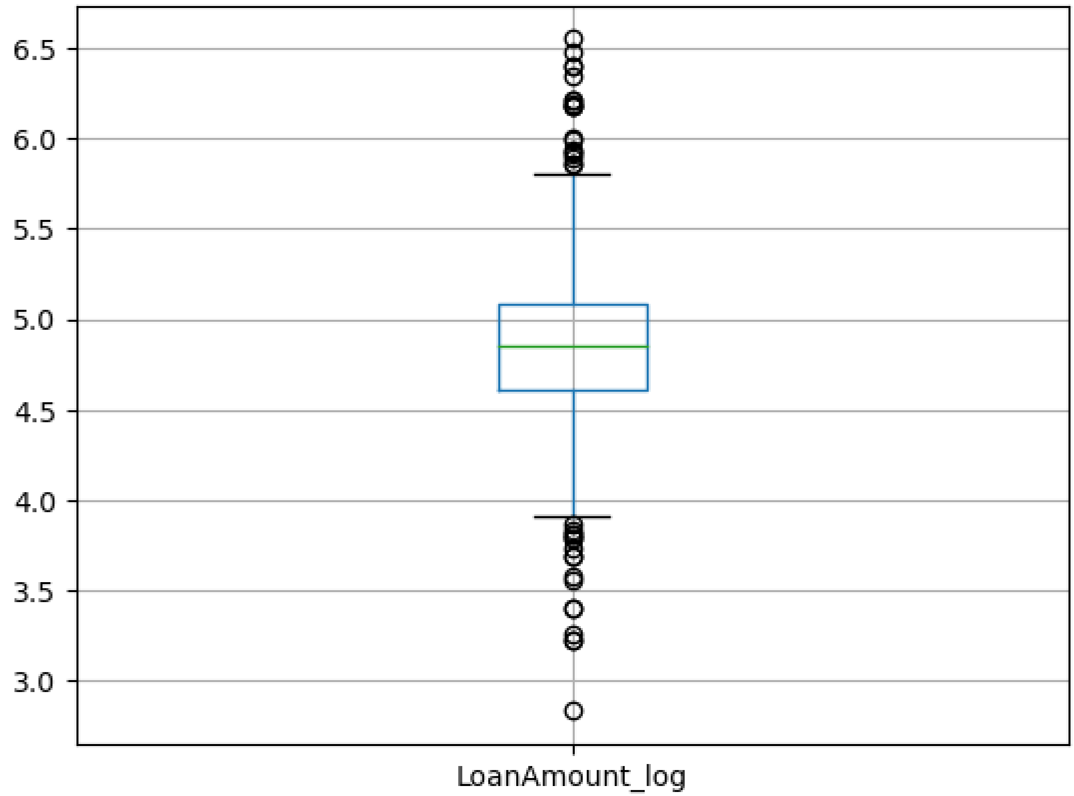
In [51]:
```
data['LoanAmount_log'] = np.log(data['LoanAmount'])
#data['LoanAmount_log'].hist(bins = 20)
```

In [52]:
```
plt.hist(data['LoanAmount_log'], 20, facecolor='b')
plt.xlabel('Loan Amount')
plt.ylabel('Count')
plt.title('Histogram for loan amount')
plt.grid(True)
plt.show()
```



In [53]:
```
data.boxplot(column='LoanAmount_log')
```

Out[53]:
```
<AxesSubplot:>
```



In [54]:
```
data.head()
```

Out[54]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantI
0	LP001478	Male	No	0	Graduate	No	2718	
1	LP002447	Male	Yes	2	Not Graduate	No	1958	
2	LP002266	Male	Yes	2	Graduate	No	3100	
3	LP002337	Female	No	0	Graduate	No	2995	
4	LP002068	Male	No	0	Graduate	No	4917	

In [55]:
```
data.describe()
```

Out[55]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	LoanA
count	550.000000	550.000000	550.000000	538.000000	506.000000	5
mean	5466.565455	1635.072582	145.457786	342.892193	0.835968	
std	6354.681175	3013.571911	84.227642	63.442106	0.370671	
min	150.000000	0.000000	17.000000	12.000000	0.000000	
25%	2904.250000	0.000000	100.000000	360.000000	1.000000	
50%	3768.500000	1188.500000	128.000000	360.000000	1.000000	
75%	5813.500000	2297.250000	162.000000	360.000000	1.000000	
max	81000.000000	41667.000000	700.000000	480.000000	1.000000	

In [56]:

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

Performing some other interesting analysis which can be derived from the data.

- Such as:
- Check another variable for outliers and treat it.
- Generate a new variable by combining two variables e.g., 'ApplicantIncome' and 'CoapplicantIncome'.

Checking for another variable for outliers and treating it.

Here we will be using ApplicantIncome variable to identify ouliners.

Step 1: Let's visualise ‘ApplicantIncome’ before and after treating the outliers

In [57]:

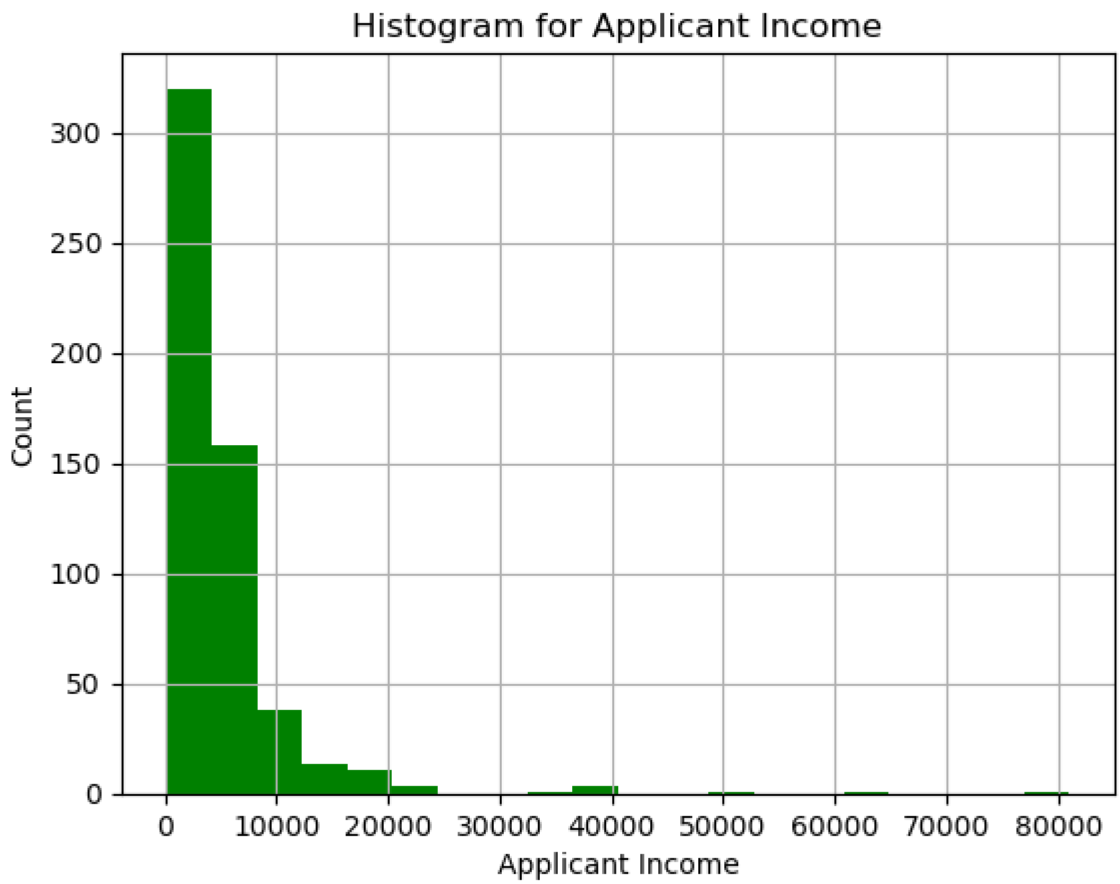
```
data.describe()
```

Out[57]:

	ApplicantIncome	CoapplicantIncome	Loan_Amount_Term	Credit_History	LoanAmount_log
count	550.000000	550.000000	538.000000	506.000000	550.000000
mean	5466.565455	1635.072582	342.892193	0.835968	4.856651
std	6354.681175	3013.571911	63.442106	0.370671	0.488581
min	150.000000	0.000000	12.000000	0.000000	2.833213
25%	2904.250000	0.000000	360.000000	1.000000	4.605170
50%	3768.500000	1188.500000	360.000000	1.000000	4.852030
75%	5813.500000	2297.250000	360.000000	1.000000	5.087596
max	81000.000000	41667.000000	480.000000	1.000000	6.551080

Creating Histogram for ApplicantIncome

In [58]:
```
plt.hist(data['ApplicantIncome'], 20, facecolor='g')
plt.xlabel('Applicant Income')
plt.ylabel('Count')
plt.title('Histogram for Applicant Income')
plt.grid(True)
plt.show()
```

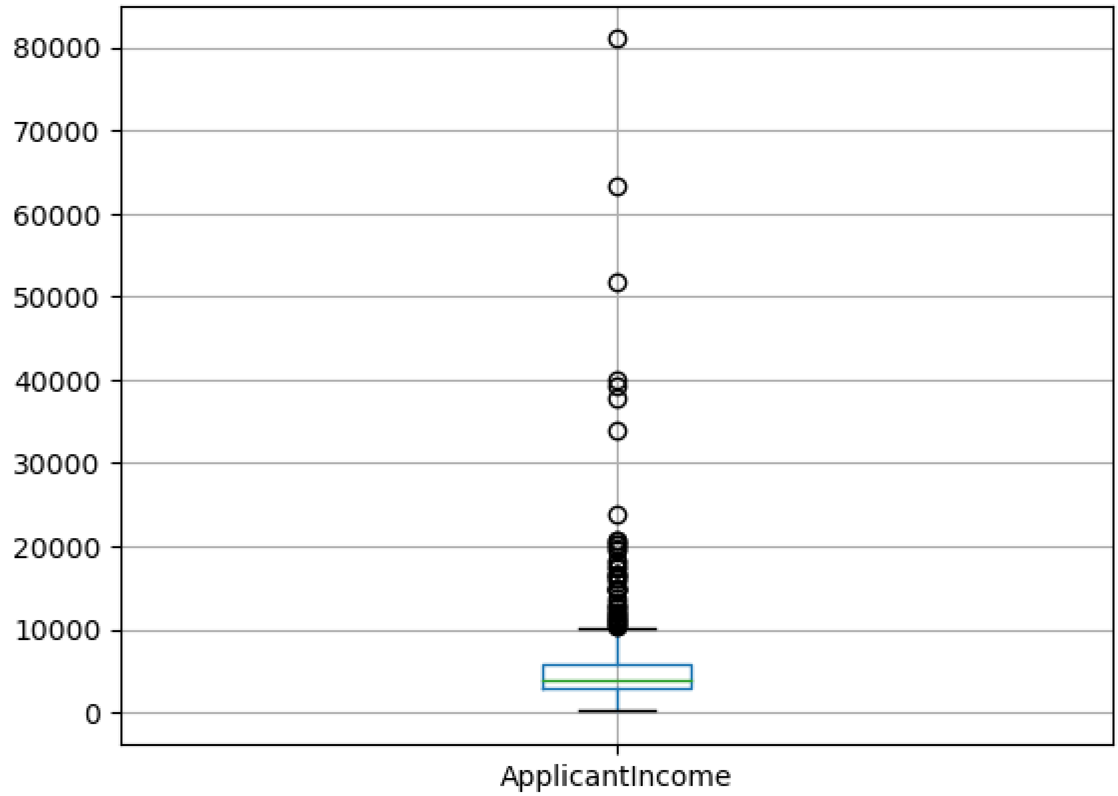


The ApplicantIncome histogram above shows a right skewed which implies that the median is less than the mean and the large values are pulling the range to the right side. The outliers are the insignificant numbers on the right side of the graph

Creating a Boxplot for ApplicantIncome

In [59]:
```
data.boxplot(column='ApplicantIncome')
```

Out[59]:
```
<AxesSubplot:>
```



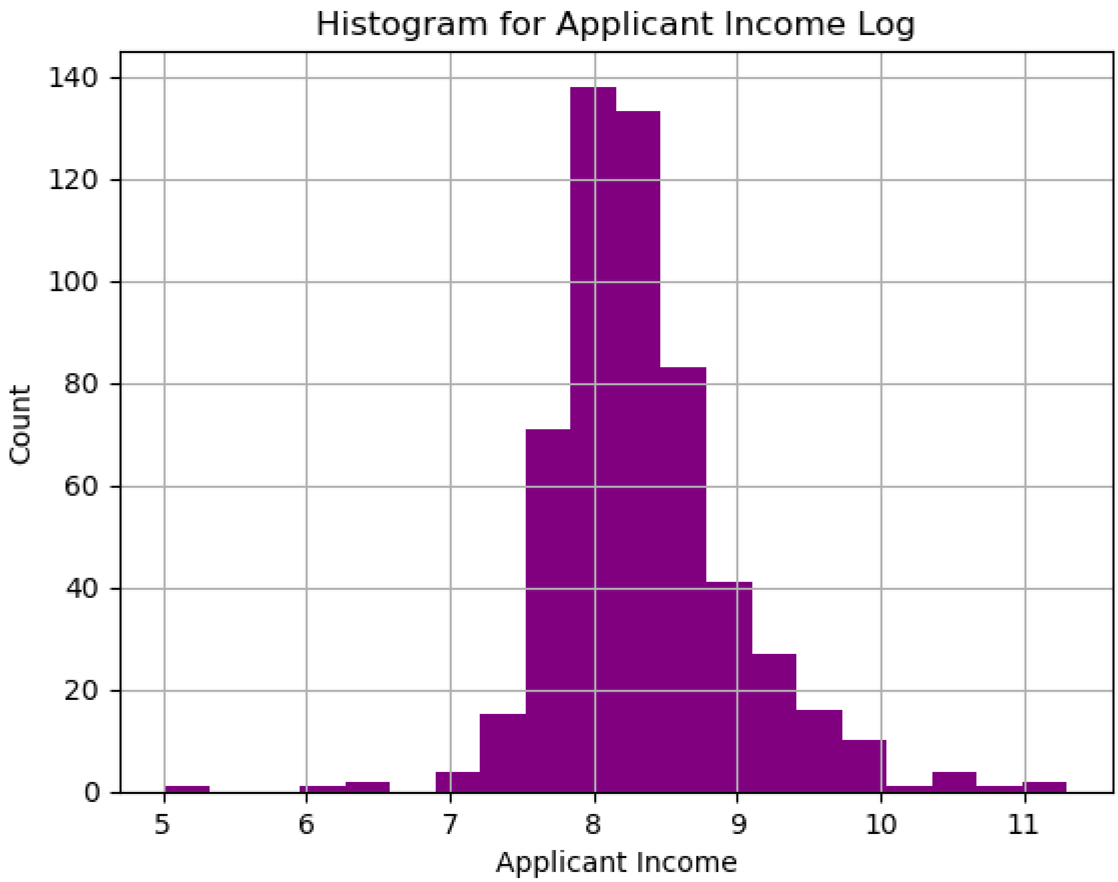
The ApplicantIncome Boxplot shows the present of outliers above the maximum value as similar to the histogram above

Step 2: Apply the Log Transform Technique to transform the data into a smaller range such that there are no dominating numbers in the dataset.

- Convert the data in the ApplicantIncome column to a log using the Log Transform function (np.log(data['ApplicantIncome']) and plot the histogram and boxplot

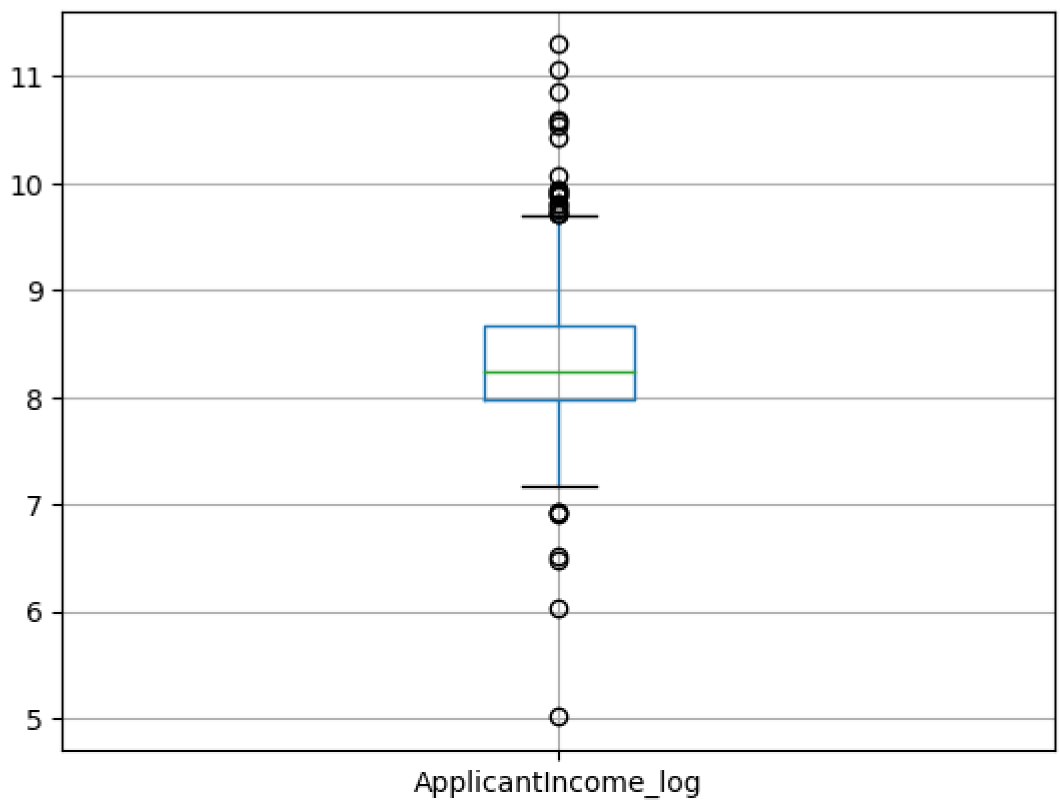
In [60]:
```
data['ApplicantIncome_log'] = np.log(data['ApplicantIncome'])
#data['ApplicantIncome_log'].hist(bins = 20)
```

In [61]:
```
plt.hist(data['ApplicantIncome_log'], 20, facecolor='purple')
plt.xlabel('Applicant Income')
plt.ylabel('Count')
plt.title('Histogram for Applicant Income Log')
plt.grid(True)
plt.show()
```



In [62]:
```
data.boxplot(column='ApplicantIncome_log')
```

Out[62]:
```
<AxesSubplot:>
```



Observation

The histogram and boxplot above shows the scale of the distribution of data from 0 to about 11.5 and outliners can be identified as numbers from 0 to 6.9 below the minimun and 9.8 to 11.5 above the maximum observation. We can say at this point that the data is semi-normalized because the graph has moved from a right skewed one to bell shaped normalised graph

In [63]:
```
data.head()
```

Out[63]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantI
0	LP001478	Male	No	0	Graduate	No	2718	
1	LP002447	Male	Yes	2	Not Graduate	No	1958	
2	LP002266	Male	Yes	2	Graduate	No	3100	
3	LP002337	Female	No	0	Graduate	No	2995	
4	LP002068	Male	No	0	Graduate	No	4917	

In [64]:
```
data.describe()
```

Out[64]:

	ApplicantIncome	CoapplicantIncome	Loan_Amount_Term	Credit_History	LoanAmount_log	Ap
count	550.000000	550.000000	538.000000	506.000000	550.000000	
mean	5466.565455	1635.072582	342.892193	0.835968	4.856651	
std	6354.681175	3013.571911	63.442106	0.370671	0.488581	
min	150.000000	0.000000	12.000000	0.000000	2.833213	
25%	2904.250000	0.000000	360.000000	1.000000	4.605170	
50%	3768.500000	1188.500000	360.000000	1.000000	4.852030	
75%	5813.500000	2297.250000	360.000000	1.000000	5.087596	
max	81000.000000	41667.000000	480.000000	1.000000	6.551080	

In [65]:
```
data.describe()
```

Out[65]:

	ApplicantIncome	CoapplicantIncome	Loan_Amount_Term	Credit_History	LoanAmount_log	Ap
count	550.000000	550.000000	538.000000	506.000000	550.000000	
mean	5466.565455	1635.072582	342.892193	0.835968	4.856651	
std	6354.681175	3013.571911	63.442106	0.370671	0.488581	
min	150.000000	0.000000	12.000000	0.000000	2.833213	
25%	2904.250000	0.000000	360.000000	1.000000	4.605170	
50%	3768.500000	1188.500000	360.000000	1.000000	4.852030	
75%	5813.500000	2297.250000	360.000000	1.000000	5.087596	
max	81000.000000	41667.000000	480.000000	1.000000	6.551080	

In [66]:

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

Observation

The above data looks close to a normalised data and we will be identifying the outliers using the semi-normalised histogram and boxplot above.

- Outliers in the Histogram:

From the ApplicantIncome histogram the outliers can be identified as all the data on the left and right side of the histogram. They are data that differs significantly from other observations. Looking closely at the histogram we can identify the outliers as numbers from 0 to 6.9 below the minimun and 9.8 to 11.5 above the maximum observation.

- Outliners in the Boxplot:

From the boxplot above we can see the distribution of data based on the minimum, first quartile(25%), second quartile (50%), third quartile (75%) and maximum value. The ouliers here are all the data above the maximum value and below the minimum value. Looking at the boxplot we can say all values between 0 to 6.9 below minimum and 9.8 to 11.5 above the maximum value.

Finally it is advised to drop the ApplicantIncome column from the dataset using the code below since we have transformed it.

Please note for the purpose of this section and to test my understanding of dealing with outliners 'ApplicantIncome has been dropped of this dataset.

Creating a new variable combining two variables

- Here i will be combining two variables; 'ApplicantIncome_log and CoapplicantIncome variables. We will be computing the difference between both variables and insert a new column called 'ApplicantIncomeDifference' showing the difference between 'ApplicantIncome_log and CoapplicantIncome'

In [67]:
```
data['ApplicantIncomeDifference'] = data.ApplicantIncome_log - data.CoapplicantIncome
```

To confirm the above code worked and 'ApplicantIncomeDifference' column has been inserted into our dataset. We double check using data.head() funtion

In [68]:
```
data.head()
```

Out[68]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	CoapplicantIncome	Loan_Amo
0	LP001478	Male	No	0	Graduate	No	0.0	
1	LP002447	Male	Yes	2	Not Graduate	No	1456.0	
2	LP002266	Male	Yes	2	Graduate	No	1400.0	
3	LP002337	Female	No	0	Graduate	No	0.0	
4	LP002068	Male	No	0	Graduate	No	0.0	

The table above shows that we have sucessfully inserted 'ApplicantIncomeDifference' column in the dataset

Missing values continuous

In [69]:
```
data['Gender'].fillna(data['Gender'].mode()[0], inplace = True)
 #0:gets the mode of each column, 1: for each row
data['Married'].fillna(data['Married'].mode()[0], inplace = True)
data['Dependents'].fillna(data['Dependents'].mode()[0], inplace = True)
data['Loan_Amount_Term'].fillna(data['Loan_Amount_Term'].mode()[0], inplace = True)
data['Credit_History'].fillna(data['Credit_History'].mode()[0], inplace = True)
```

In [70]:
```
data.apply(lambda x: sum(x.isnull()), axis=0)
```

Out[70]:
```
Loan_ID                      0
Gender                       0
Married                      0
Dependents                   0
Education                    0
Self_Employed                0
CoapplicantIncome            0
Loan_Amount_Term             0
Credit_History               0
Property_Area                0
Loan_Status                  0
LoanAmount_log               0
ApplicantIncome_log          0
ApplicantIncomeDifference    0
dtype: int64
```

Use LabelEncoder, to convert categorical variables into numeric.

-- First, we will need to identify categorical values.

In identifying the categirical values we use the code below.

In [71]:
```
data.head()
```

Out[71]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	CoapplicantIncome	Loan_Amo
0	LP001478	Male	No	0	Graduate	No	0.0	
1	LP002447	Male	Yes	2	Not Graduate	No	1456.0	
2	LP002266	Male	Yes	2	Graduate	No	1400.0	
3	LP002337	Female	No	0	Graduate	No	0.0	
4	LP002068	Male	No	0	Graduate	No	0.0	

In [72]:
```
data.shape
```

Out[72]:
```
(550, 14)
```

In [73]:
```
from sklearn.preprocessing import LabelEncoder
```

In [74]:
```
columns = list(data)
print(columns)
```

```
['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education', 'Self_Employed', 'Coappli
cantIncome', 'Loan_Amount_Term', 'Credit_History', 'Property_Area', 'Loan_Status', 'L
oanAmount_log', 'ApplicantIncome_log', 'ApplicantIncomeDifference']
```

In [75]:
```
data.dtypes
```

Out[75]:
```
Loan_ID                       object
Gender                        object
Married                       object
Dependents                    object
Education                     object
Self_Employed                 object
CoapplicantIncome            float64
Loan_Amount_Term             float64
Credit_History               float64
Property_Area                 object
Loan_Status                   object
LoanAmount_log               float64
ApplicantIncome_log          float64
ApplicantIncomeDifference    float64
dtype: object
```

In [76]:
```
columns = list(data.select_dtypes(exclude=['float64','int64']))
```

In [77]:
```
le = LabelEncoder()
for i in columns:
    #print(i)
    data[i] = le.fit_transform(data[i])
```

In [78]:
```
data.head()
```

Out[78]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	CoapplicantIncome	Loan_Amou
0	114	1	0	0	0	0	0.0	
1	402	1	1	2	1	0	1456.0	
2	350	1	1	2	0	0	1400.0	
3	370	0	0	0	0	0	0.0	
4	286	1	0	0	0	0	0.0	

Data Normalisation

In [79]:

```
#from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import normalize
```

In [80]:

```
original_data = data.copy()
original_data.head()
```

Out[80]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	CoapplicantIncome	Loan_Amou
0	114	1	0	0	0	0	0.0	
1	402	1	1	2	1	0	1456.0	
2	350	1	1	2	0	0	1400.0	
3	370	0	0	0	0	0	0.0	
4	286	1	0	0	0	0	0.0	

In [81]:

```
original_data[0:5]
```

Out[81]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	CoapplicantIncome	Loan_Amou
0	114	1	0	0	0	0	0.0	
1	402	1	1	2	1	0	1456.0	
2	350	1	1	2	0	0	1400.0	
3	370	0	0	0	0	0	0.0	
4	286	1	0	0	0	0	0.0	

In [82]:

```
data[0:5]
```

Out[82]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	CoapplicantIncome	Loan_Amou
0	114	1	0	0	0	0	0.0	
1	402	1	1	2	1	0	1456.0	
2	350	1	1	2	0	0	1400.0	
3	370	0	0	0	0	0	0.0	
4	286	1	0	0	0	0	0.0	

In [83]:

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

Reason why Loan ID was dropped

This just shows the unique loan ID and doesn't show any significant details to build a machine learning model and we dropped it because we don't want to normalize it in the data.

In [84]:

```
normalized_data = normalize( data_for_norm )
```

In [85]:

```
print(normalized_data[0:5])
```

```
[[ 2.77621326e-03  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  9.99436774e-01  2.77621326e-03
   2.77621326e-03  1.17947288e-02  2.19533272e-02  2.19533272e-02]
 [ 4.81798001e-04  4.81798001e-04  9.63596001e-04  4.81798001e-04
   0.00000000e+00  7.01497889e-01  1.44539400e-01  4.81798001e-04
   9.63596001e-04  1.97264702e-03  3.65187410e-03 -6.97846015e-01]
 [ 4.98305344e-04  4.98305344e-04  9.96610689e-04  0.00000000e+00
   0.00000000e+00  6.97627482e-01  1.79389924e-01  4.98305344e-04
   9.96610689e-04  2.35568262e-03  4.00595509e-03 -6.93621527e-01]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  9.99422129e-01  2.77617258e-03
   5.55234516e-03  1.13666071e-02  2.22224273e-02  2.22224273e-02]
 [ 2.77596619e-03  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  9.99347829e-01  0.00000000e+00
   0.00000000e+00  1.35121111e-02  2.35969725e-02  2.35969725e-02]]
```

In [86]:

```
normalized_data.shape
```

Out[86]:

```
(550, 12)
```

In [87]:

```
data.shape
```

Out[87]:

```
(550, 14)
```

In [88]:

```
normalized_data = pd.DataFrame(normalized_data, columns=data_for_norm.columns)
```

In [89]:

```
normalized_data.head()
```

Out[89]:

	Gender	Married	Dependents	Education	Self_Employed	CoapplicantIncome	Loan_Amount_Term
0	0.002776	0.000000	0.000000	0.000000	0.0	0.000000	0.999437
1	0.000482	0.000482	0.000964	0.000482	0.0	0.701498	0.144539
2	0.000498	0.000498	0.000997	0.000000	0.0	0.697627	0.179390
3	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	0.999422
4	0.002776	0.000000	0.000000	0.000000	0.0	0.000000	0.999348

In [90]:

```
normalized_data['Loan_ID'] = data['Loan_ID']
```

In [91]:

```
normalized_data['Loan_Status'] = data['Loan_Status']
```

In [92]:

```
normalized_data.head()
```

Out[92]:

	Gender	Married	Dependents	Education	Self_Employed	CoapplicantIncome	Loan_Amount_Term
0	0.002776	0.000000	0.000000	0.000000	0.0	0.000000	0.999437
1	0.000482	0.000482	0.000964	0.000482	0.0	0.701498	0.144539
2	0.000498	0.000498	0.000997	0.000000	0.0	0.697627	0.179390
3	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	0.999422
4	0.002776	0.000000	0.000000	0.000000	0.0	0.000000	0.999348

In [93]:

```
normalized_data.describe()
```

Out[93]:

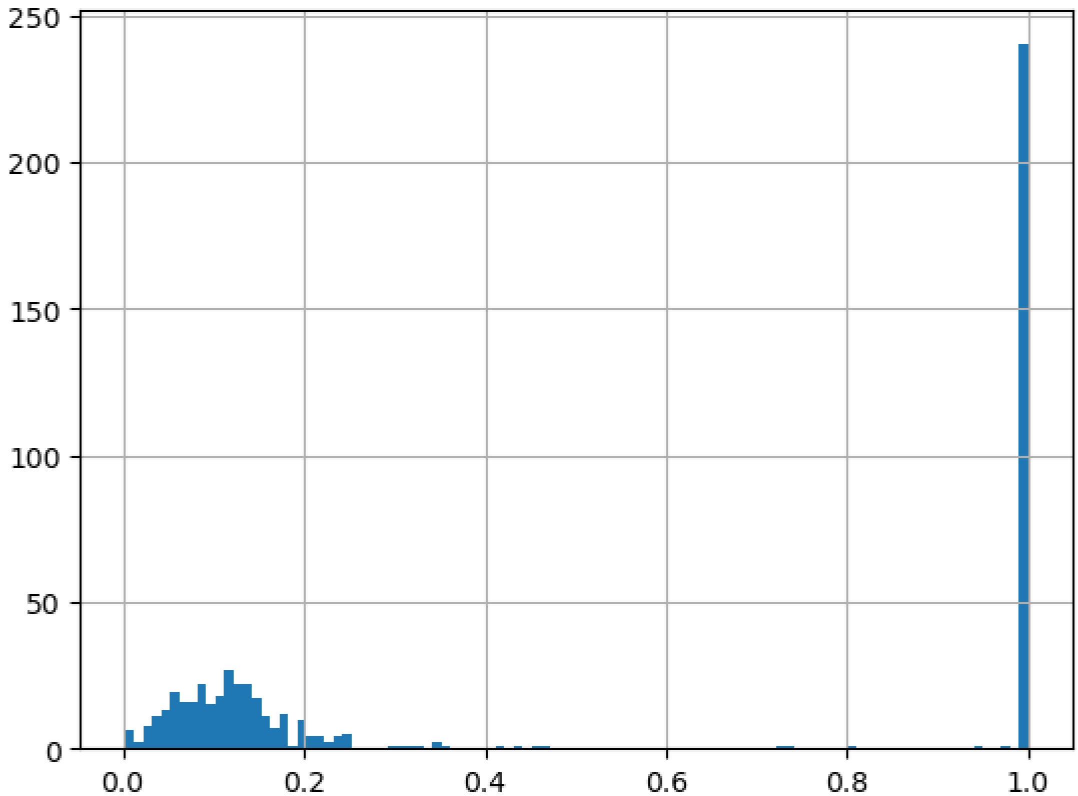
	Gender	Married	Dependents	Education	Self_Employed	CoapplicantIncome	Loan_Amc
count	550.000000	550.000000	550.000000	550.000000	550.000000	550.000000	5
mean	0.001253	0.000864	0.001284	0.000413	0.000252	0.391542	
std	0.001930	0.001484	0.003290	0.001622	0.001055	0.347728	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.000165	0.000000	0.000000	0.000000	0.000000	0.000000	
50%	0.000381	0.000277	0.000000	0.000000	0.000000	0.694573	
75%	0.002776	0.000667	0.000840	0.000000	0.000000	0.704291	
max	0.026252	0.016225	0.048675	0.026252	0.016225	0.708723	

Test: You can play with the data yourself by performing some more analysis for fun! An example is provided in the above cell.

In [94]:

```
normalized_data['Loan_Amount_Term'].hist(bins=100)
```

Out[94]: <AxesSubplot:>



Building a Decison Tree Classifier Using Sklearn

Importing all necessary libraries from sklearn

In [95]:
```
from sklearn.model_selection import train_test_split
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.tree import export_graphviz
from sklearn.metrics import ConfusionMatrixDisplay
#import pydotplus
```

Feature Selection

In [96]:
```
columns = list(normalized_data.columns)
columns
```

Out[96]:
```
['Gender',
 'Married',
 'Dependents',
 'Education',
 'Self_Employed',
 'CoapplicantIncome',
 'Loan_Amount_Term',
 'Credit_History',
 'Property_Area',
 'LoanAmount_log',
 'ApplicantIncome_log',
 'ApplicantIncomeDifference',
 'Loan_ID',
 'Loan_Status']
```

In [97]:
```
normalized_data.head()
```

Out[97]:

	Gender	Married	Dependents	Education	Self_Employed	CoapplicantIncome	Loan_Amount_Term
0	0.002776	0.000000	0.000000	0.000000	0.0	0.000000	0.999437
1	0.000482	0.000482	0.000964	0.000482	0.0	0.701498	0.144539
2	0.000498	0.000498	0.000997	0.000000	0.0	0.697627	0.179390
3	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	0.999422
4	0.002776	0.000000	0.000000	0.000000	0.0	0.000000	0.999348

In [98]:
```
features = normalized_data.drop(['Loan_ID','Loan_Status'], axis = 1)
classes = pd.DataFrame(normalized_data['Loan_Status'])
```

In [99]:
```
print('Features:')
print(features.head())
print('Classes:')
print(classes.head())
```

```
Features:
     Gender   Married  Dependents  Education  Self_Employed  \
0  0.002776  0.000000    0.000000   0.000000            0.0
1  0.000482  0.000482    0.000964   0.000482            0.0
2  0.000498  0.000498    0.000997   0.000000            0.0
3  0.000000  0.000000    0.000000   0.000000            0.0
4  0.002776  0.000000    0.000000   0.000000            0.0

   CoapplicantIncome  Loan_Amount_Term  Credit_History  Property_Area  \
0           0.000000          0.999437        0.002776       0.002776
1           0.701498          0.144539        0.000482       0.000964
2           0.697627          0.179390        0.000498       0.000997
3           0.000000          0.999422        0.002776       0.005552
4           0.000000          0.999348        0.000000       0.000000

   LoanAmount_log  ApplicantIncome_log  ApplicantIncomeDifference
0        0.011795             0.021953                   0.021953
1        0.001973             0.003652                  -0.697846
2        0.002356             0.004006                  -0.693622
3        0.011367             0.022222                   0.022222
4        0.013512             0.023597                   0.023597
Classes:
   Loan_Status
0            1
1            1
2            1
3            1
4            1
```

In [100…

```
normalized_data.head(10)
```

Out[100]:

	Gender	Married	Dependents	Education	Self_Employed	CoapplicantIncome	Loan_Amount_Term
0	0.002776	0.000000	0.000000	0.000000	0.0	0.000000	0.999437
1	0.000482	0.000482	0.000964	0.000482	0.0	0.701498	0.144539
2	0.000498	0.000498	0.000997	0.000000	0.0	0.697627	0.179390
3	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	0.999422
4	0.002776	0.000000	0.000000	0.000000	0.0	0.000000	0.999348
5	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	0.999461
6	0.002247	0.002247	0.006741	0.000000	0.0	0.424690	0.808933
7	0.000615	0.000615	0.001230	0.000000	0.0	0.692412	0.221375
8	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	0.999382
9	0.000000	0.001001	0.000000	0.001001	0.0	0.663575	0.360312

In [101…

```
normalized_data.shape
```

Out[101]:

```
(550, 14)
```

Building our first baseline model using all the features.

Partitioning data into Train and Test sets

In [102…
```
normalized_data.shape
```
Out[102]:
```
(550, 14)
```

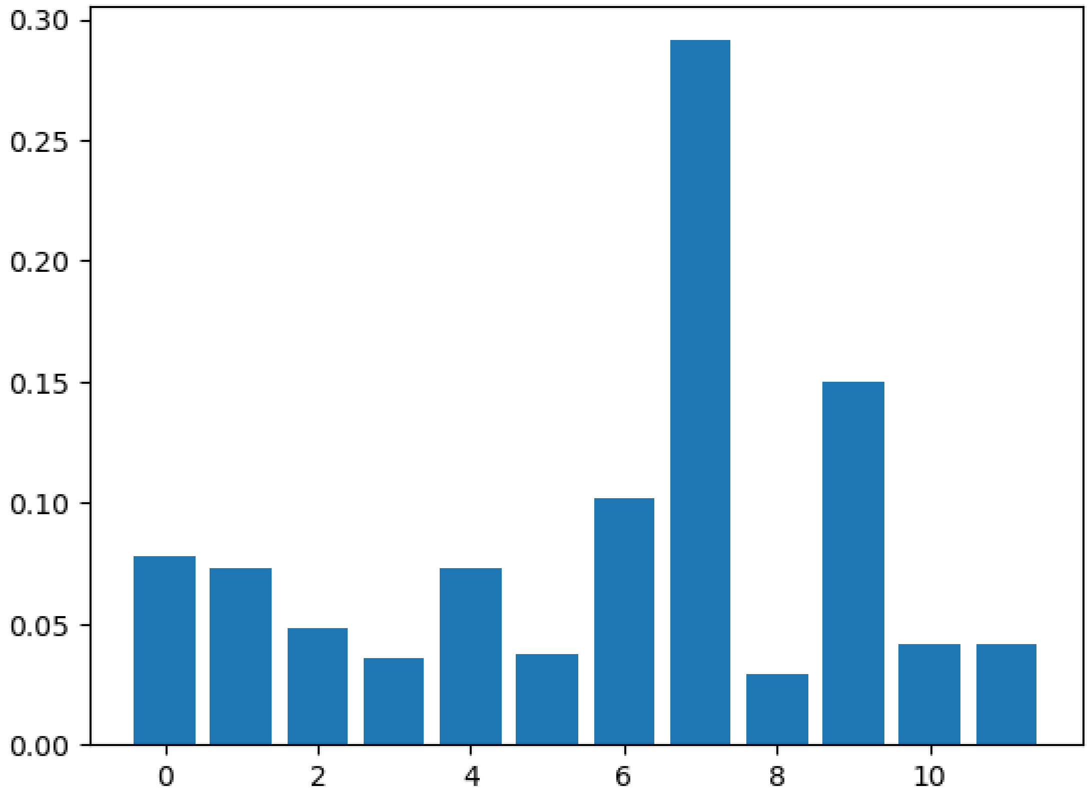
In [103…
```
from matplotlib import pyplot
```

In [104…
```
x_train, x_test, y_train, y_test = train_test_split(features, classes, test_size= .33,
 random_state = 17)
print(x_train.shape, x_test.shape)
```
```
(368, 12) (182, 12)
```

In [105…
```
decisionTree = DecisionTreeClassifier(criterion='entropy')
print(decisionTree)
```
```
DecisionTreeClassifier(criterion='entropy')
```

In [106…
```
dtc_model = decisionTree.fit(x_train, y_train)
```

In [107…
```
# feature importance
importance = dtc_model.feature_importances_
for i,v in enumerate(importance):
 print('Feature: %0d, Score: %.5f' % (i,v))
# Barchat for feature importance
pyplot.bar([x for x in range(len(importance))], importance)
pyplot.show()
```
```
Feature: 0, Score: 0.07772
Feature: 1, Score: 0.07339
Feature: 2, Score: 0.04813
Feature: 3, Score: 0.03561
Feature: 4, Score: 0.07315
Feature: 5, Score: 0.03725
Feature: 6, Score: 0.10167
Feature: 7, Score: 0.29110
Feature: 8, Score: 0.02882
Feature: 9, Score: 0.14972
Feature: 10, Score: 0.04193
Feature: 11, Score: 0.04151
```



features/columns: 0:'Gender', 1:'Married', 2:'Dependents', 2:'Education', 4:'Self_Employed', 5:'ApplicantIncome', 6:'CoapplicantIncome', 7:'Loan_Amount_Term', 8:'Credit_History', 9:'Property_Area', 10:'LoanAmount_log'

In [108…

```
prediction = dtc_model.predict(x_test)
```

In [109…

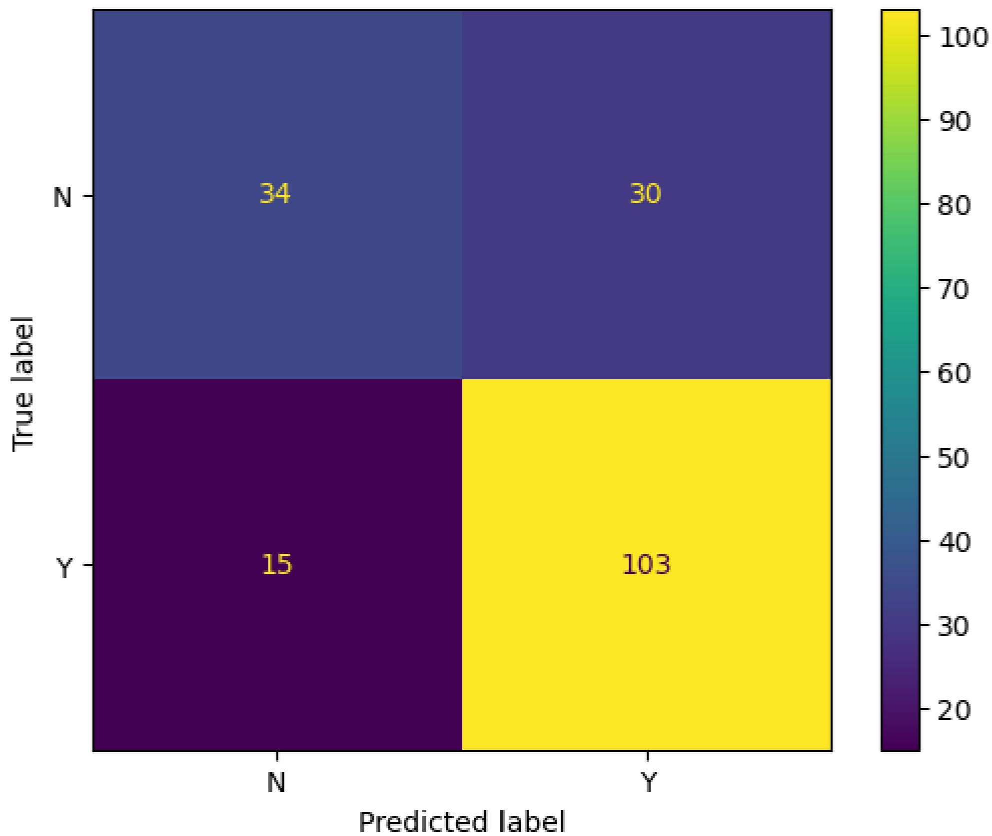
```
y_true = le.inverse_transform(y_test["Loan_Status"])
y_pred = le.inverse_transform(prediction)
```

In [110…

```
cm = confusion_matrix(y_true, y_pred)
labels = ['N', 'Y']
ConfusionMatrixDisplay(cm, display_labels=labels).plot()
```

Out[110]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x24da38a5e20>
```



In [111…

```
print(classification_report(y_true, y_pred))
```

```
              precision    recall  f1-score   support

           N       0.69      0.53      0.60        64
           Y       0.77      0.87      0.82       118

    accuracy                           0.75       182
   macro avg       0.73      0.70      0.71       182
weighted avg       0.75      0.75      0.74       182
```

Visualising the decision tree

In [112…

```
graphviz_path = 'C:\Program Files\Graphviz/bin/'
```

In [113…

```
import os
os.environ["PATH"] += os.pathsep + graphviz_path
```

In [114…

```
pip install graphviz
```

```
Requirement already satisfied: graphviz in c:\users\lenovox260\anaconda3\lib\site-pac
kages (0.20.1)
Note: you may need to restart the kernel to use updated packages.
```

In [115…

```
pip install cairosvg
```

```
Requirement already satisfied: cairosvg in c:\users\lenovox260\anaconda3\lib\site-pac
kages (2.6.0)
Requirement already satisfied: pillow in c:\users\lenovox260\anaconda3\lib\site-packa
ges (from cairosvg) (9.2.0)
Requirement already satisfied: tinycss2 in c:\users\lenovox260\anaconda3\lib\site-pac
kages (from cairosvg) (1.2.1)
Requirement already satisfied: defusedxml in c:\users\lenovox260\anaconda3\lib\site-p
ackages (from cairosvg) (0.7.1)
Requirement already satisfied: cairocffi in c:\users\lenovox260\anaconda3\lib\site-pa
ckages (from cairosvg) (1.4.0)
Requirement already satisfied: cssselect2 in c:\users\lenovox260\anaconda3\lib\site-p
ackages (from cairosvg) (0.7.0)
Requirement already satisfied: cffi>=1.1.0 in c:\users\lenovox260\anaconda3\lib\site-
packages (from cairocffi->cairosvg) (1.15.1)
Requirement already satisfied: webencodings in c:\users\lenovox260\anaconda3\lib\site
-packages (from cssselect2->cairosvg) (0.5.1)
Requirement already satisfied: pycparser in c:\users\lenovox260\anaconda3\lib\site-pa
ckages (from cffi>=1.1.0->cairocffi->cairosvg) (2.21)
Note: you may need to restart the kernel to use updated packages.
```

In [116…

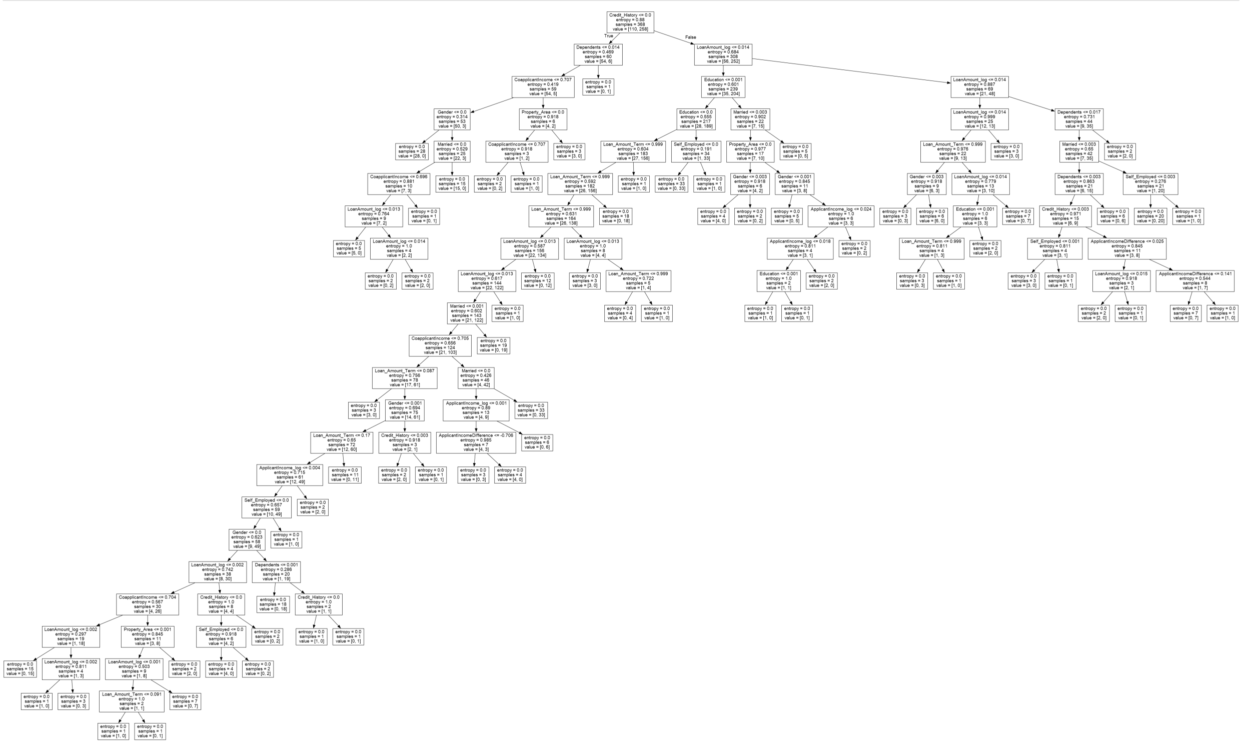
```python
from graphviz import Source
from sklearn import tree
graph = Source( tree.export_graphviz(dtc_model, out_file=None, feature_names=features,
```

In [117…

```python
from cairosvg import svg2png
from IPython.display import Image

svg2png(bytestring=graph.pipe(format='svg'),write_to='output.png')
Image("output.png")
```

Out[117]:



Report

Based on the feature importance, we will select a different set of features to build another decision tree model. Aimed to improve the result of the baseline model

In [118…

```
new_features = normalized_data.drop(['Education', 'Dependents', 'Credit_History', 'Loa
                                     'ApplicantIncome_log', 'Loan_ID', 'ApplicantIncom
                                    ], axis =1)
```

In [119…

```
print('Features')
print(new_features.head())
```

```
Features
     Gender   Married  CoapplicantIncome  Loan_Amount_Term  Property_Area
0  0.002776  0.000000           0.000000          0.999437       0.002776
1  0.000482  0.000482           0.701498          0.144539       0.000964
2  0.000498  0.000498           0.697627          0.179390       0.000997
3  0.000000  0.000000           0.000000          0.999422       0.005552
4  0.002776  0.000000           0.000000          0.999348       0.000000
```

In [120…

```
x_train, x_test, y_train, y_test = train_test_split(features, classes, test_size= .33,
 random_state = 17)
print(x_train.shape, x_test.shape)
```

```
(368, 12) (182, 12)
```

In [121…

```
decisionTree = DecisionTreeClassifier(criterion='entropy')
print(decisionTree)
```

```
DecisionTreeClassifier(criterion='entropy')
```

In [122…

```
dtc_model = decisionTree.fit(x_train, y_train)
```

In [123…

```
prediction = dtc_model.predict(x_test)
```

In [124…

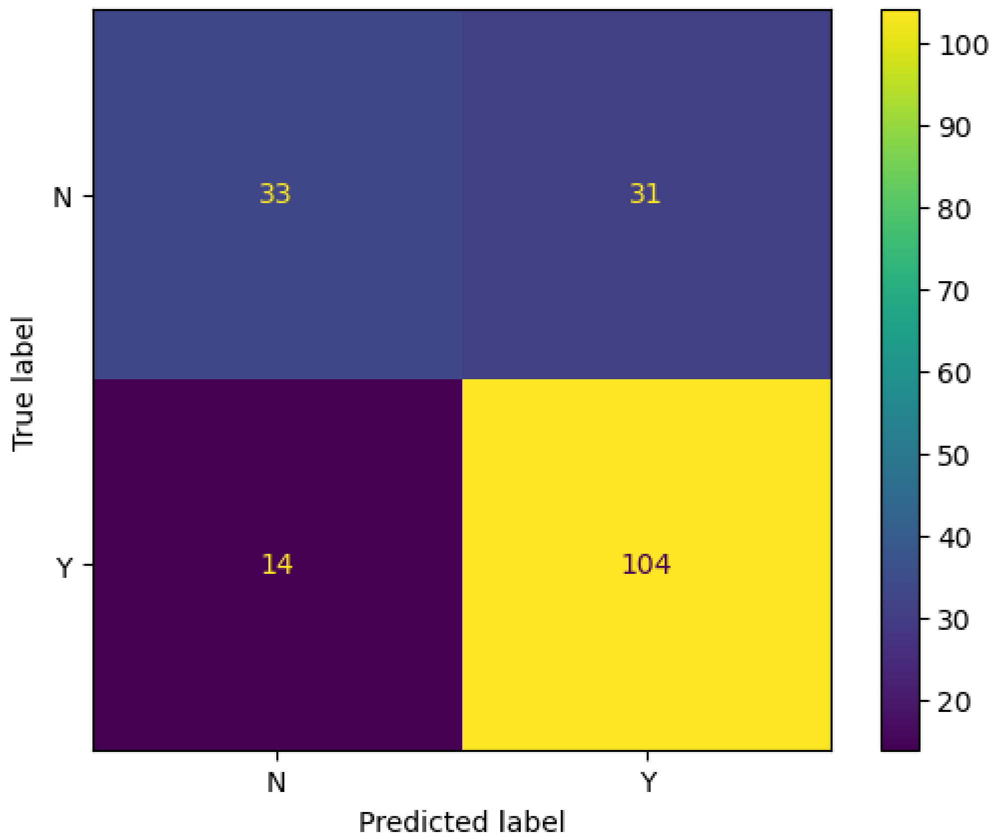
```
y_true = le.inverse_transform(y_test["Loan_Status"])
y_pred = le.inverse_transform(prediction)
```

In [125…

```
cm = confusion_matrix(y_true, y_pred)
labels = ['N', 'Y']
ConfusionMatrixDisplay(cm, display_labels=labels).plot()
```

Out[125]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x24da3c124c0>
```



In [126...

```
print(classification_report(y_true, y_pred))
```

```
              precision    recall  f1-score   support

           N       0.70      0.52      0.59        64
           Y       0.77      0.88      0.82       118

    accuracy                           0.75       182
   macro avg       0.74      0.70      0.71       182
weighted avg       0.75      0.75      0.74       182
```

In [127...

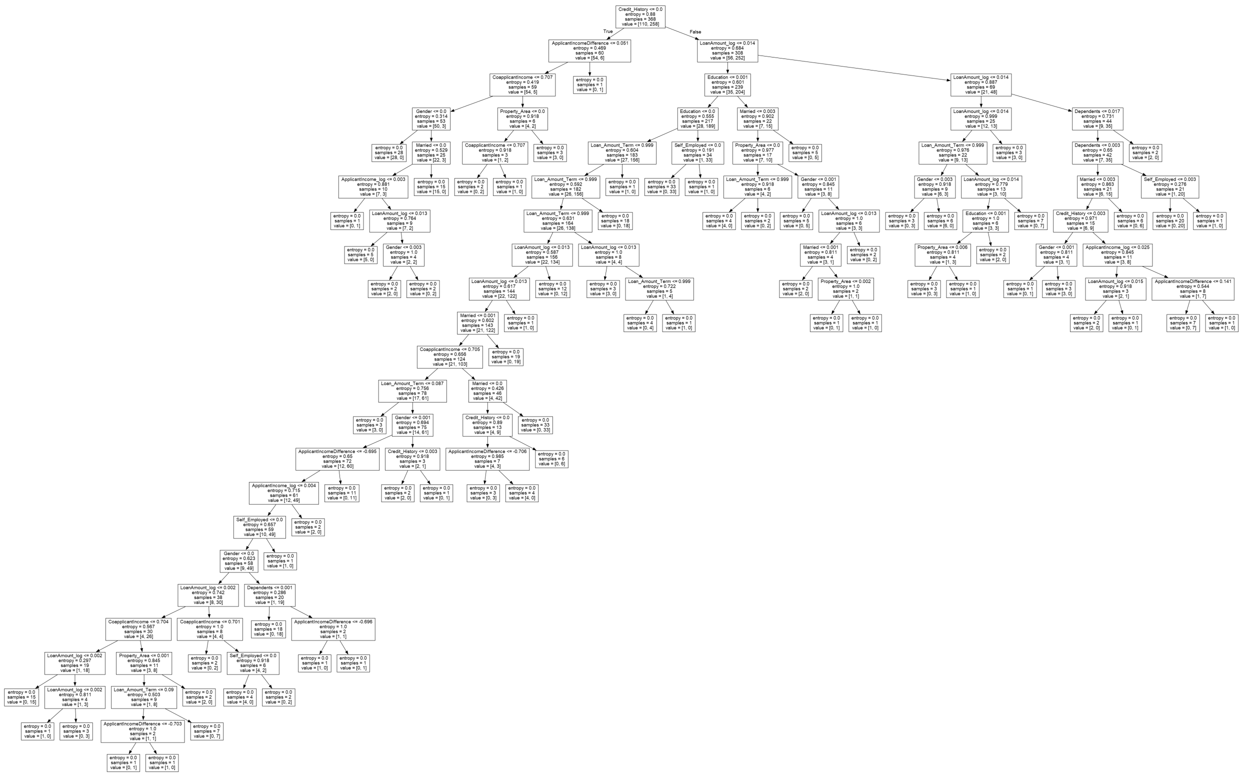
```
from graphviz import Source
from sklearn import tree
graph = Source( tree.export_graphviz(dtc_model, out_file=None, feature_names=features.
```

In [128...

```
from cairosvg import svg2png
from IPython.display import Image

svg2png(bytestring=graph.pipe(format='svg'),write_to='output.png')
Image("output.png")
```

Out[128]:



Writing a summary to comparing both the models.

-- The summary would include:

- Idea behind selecting those particular features and,
- comparative analysis of the results of both the models.

There is no difference between both models. The first model was built using all the features while the second was built considering features with higher scores > or = to 0.07 as these features impacts positively in building a decision tree model.

Discussing the result based on the evaluation matrix

Precision: The ability of a classification model to identify only the relevant data points. Mathematically, precision is the number of true positives divided by the number of true positives plus the number of false positives.

Recall: The ability of a model to find all the relevant cases within a data set. Mathematically, we define recall as the number of true positives divided by the number of true positives plus the number of false negatives.

The F1 score is the harmonic mean of precision and recall taking both metrics into account where higher value indicates better performance.

Support: Is the number of samples available for each class.

For N-Class

Precision is 0.72 Recall is 0.52 And F1 score is 0.60

For Y-Class

Precision is 0.77 Recall is 0.89 And F1 score is 0.83

While the model weighted average is 0.75 and accuracy is 0.76 for F1-score

Going by the accuracy the model correctly predicts the class of 74% of the samples in the new test set.