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

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

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

Out[3]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Cr
0	LP001002	Male	No	0	Graduate	No	5849	0.0	NaN	360.0	
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.0	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	360.0	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0	
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	360.0	

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

Out[4]: (614, 13)

In [5]:
```python
dataset = dataset.sample(n=550, random_state = 10)
```

In [6]:
```python
dataset.to_csv('MoradekeAdeleye_2229810.csv')
```

In [7]:
```python
data = pd.read_csv('MoradekeAdeleye_2229810.csv')
```

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

Out[8]:

	Unnamed: 0	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amou
0	285	LP001924	Male	No	0	Graduate	No	3158	3053.0	89.0	
1	323	LP002055	Female	No	0	Graduate	No	3166	2985.0	132.0	
2	482	LP002537	Male	Yes	0	Graduate	No	2083	3150.0	128.0	
3	173	LP001594	Male	Yes	0	Graduate	No	5708	5625.0	187.0	
4	518	LP002683	Male	No	0	Graduate	No	4683	1915.0	185.0	

In [9]:
```
data=data.drop('Unnamed: 0', axis = 1)
```

In [10]:
```
data.head()
```

Out[10]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Cr
0	LP001924	Male	No	0	Graduate	No	3158	3053.0	89.0	360.0	
1	LP002055	Female	No	0	Graduate	No	3166	2985.0	132.0	360.0	
2	LP002537	Male	Yes	0	Graduate	No	2083	3150.0	128.0	360.0	
3	LP001594	Male	Yes	0	Graduate	No	5708	5625.0	187.0	360.0	
4	LP002683	Male	No	0	Graduate	No	4683	1915.0	185.0	360.0	

Q1. Use and explain the following DataFrame functions/properties on your data. describe(): This describes the data in the data frame. It is used to calculate some satistical data like the mean, standard deviation, the 25th, 50th, 75th percentile, the minmum and maximum values of numerical values of the data set. size: This describes the number of cells in the data set. Can be gotten by multiplying the number of rows by the number of columns. ndim: It is used to know the number of dimensions in a data set. For instance, if the command returns with an integer 2, it means that the dataset has rows and columns. shape: This describes the number of rows and columns in the data set.

In [11]:
```
data.describe()
```
Out[11]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	550.000000	550.000000	528.000000	538.000000	506.000000
mean	5354.065455	1624.330764	146.098485	342.401487	0.843874
std	5475.802136	3019.983826	86.356295	65.193956	0.363334
min	416.000000	0.000000	9.000000	12.000000	0.000000
25%	2894.250000	0.000000	100.000000	360.000000	1.000000
50%	3750.000000	1128.500000	127.000000	360.000000	1.000000
75%	5806.250000	2250.000000	170.000000	360.000000	1.000000
max	63337.000000	41667.000000	700.000000	480.000000	1.000000

In [12]:
```
data.size
```
Out[12]:
```
7150
```
In [13]:
```
data.ndim
```
Out[13]:
```
2
```
In [14]:
```
data.shape
```
Out[14]:
```
(550, 13)
```

Q2. Is there any difference between dimensions of the original dataset and the new dataset? If yes, what is the difference? The original data set has 614 rows and 13 columns, while my new dataset has 550 rows and 13 columns. The data size of my new dataset is 7150 as against 7982 which is the data size for the original data set.

#Q3. What are the possible values 'Education' can take? Write code to display all the possible values of 'Education'. #line 15 shows the possible values for 'Education".There are 431 Graduates, and 119 Not Graduates.

In [15]:
```
data['Education'].value_counts()
```

Out[15]:
```
Graduate        431
Not Graduate    119
Name: Education, dtype: int64
```

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

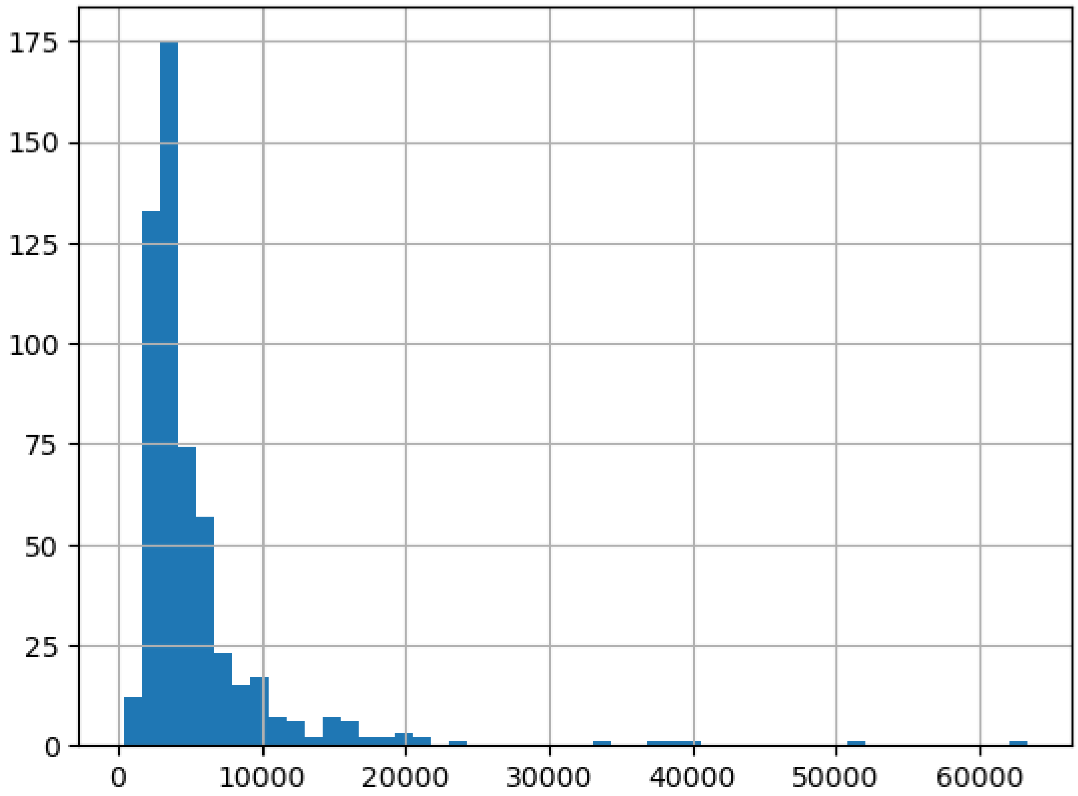
Out[16]:
```
Index(['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education',
       'Self_Employed', 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History', 'Property_Area', 'Loan_Status'],
      dtype='object')
```

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

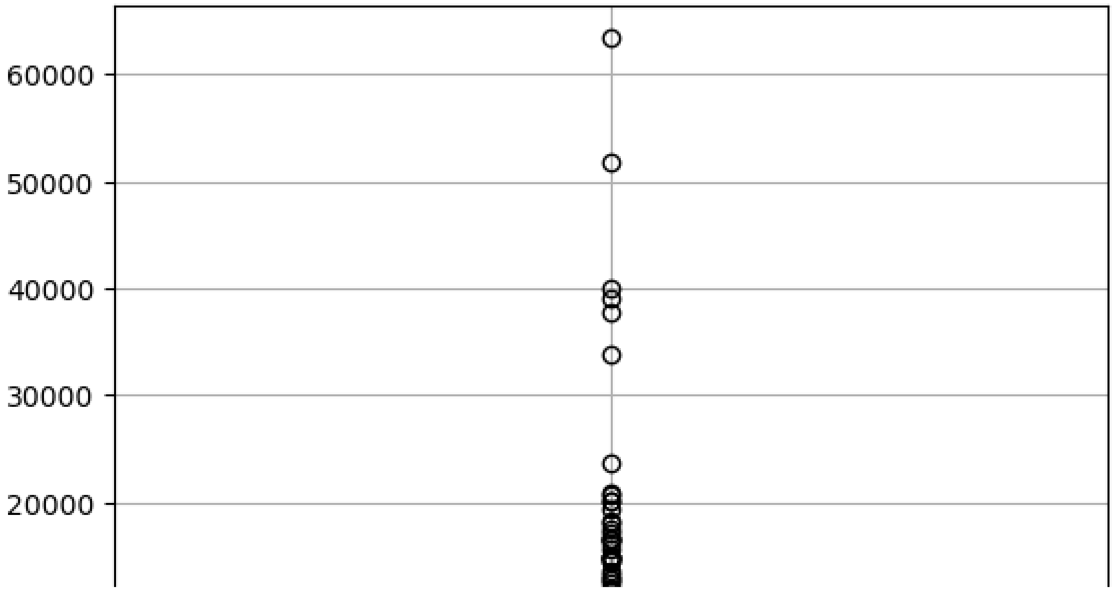
Out[17]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Cr
0	LP001924	Male	No	0	Graduate	No	3158	3053.0	89.0	360.0	
1	LP002055	Female	No	0	Graduate	No	3166	2985.0	132.0	360.0	
2	LP002537	Male	Yes	0	Graduate	No	2083	3150.0	128.0	360.0	
3	LP001594	Male	Yes	0	Graduate	No	5708	5625.0	187.0	360.0	
4	LP002683	Male	No	0	Graduate	No	4683	1915.0	185.0	360.0	

In [18]:
```
data['ApplicantIncome'].hist(bins=50)
```
Out[18]: <AxesSubplot:>



In [19]:
```
data.boxplot(column='ApplicantIncome')
```
Out[19]: <AxesSubplot:>

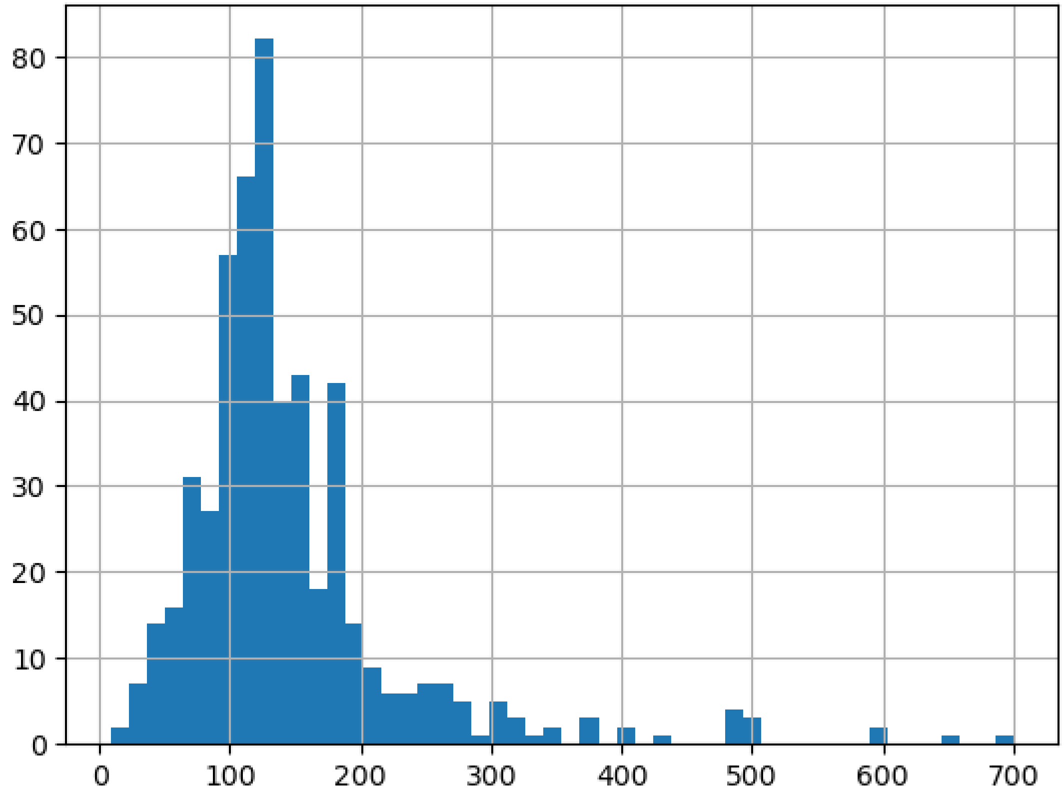


4a. What are the extreme values? Are there any outliers(s) exist in this dataset? Explain with example based on the 'ApplicantIncome'? Extreme values are values that vary significantly from the data. They are can either be too large or too small.If not removed, they can affect the accuracy of statistical models and conclusions. There are a lot of outliers in this data set as can be seen from the circles above the maximum value for the applicant income in the box plot. The outliers are the values above 10000, which is the maximum value as shown by the boxplot.

4b. Are the results of both the plots comparable? Are there any differences in the two plots? What are the key differences? Yes, the results of both plots are comparable. We can see the outliers in both plots even though the boxplot shows it in a more detailed way. Both plots are skewed to the right.

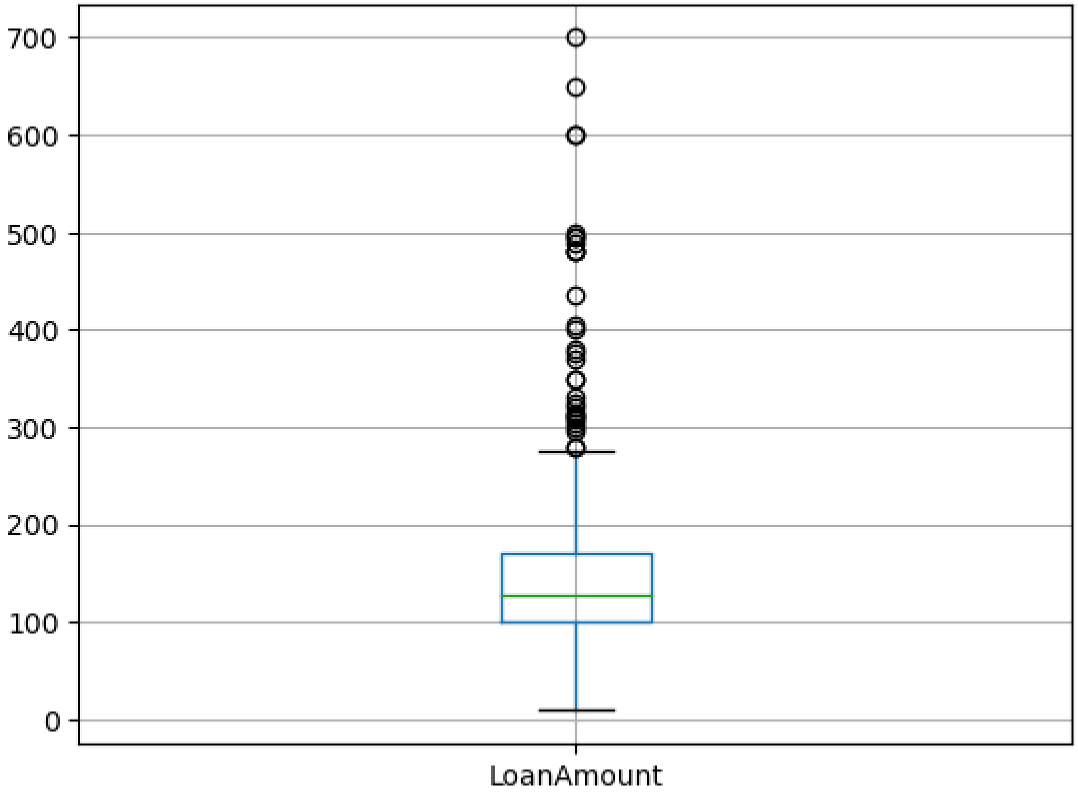
```
In [20]: # Histogram on LoanAmount
         data['LoanAmount'].hist(bins=50)
```

Out[20]: <AxesSubplot:>



In [21]:
```python
# Boxplot on LoanAmount
data.boxplot(column='LoanAmount')
```

Out[21]: <AxesSubplot:>



In Loan amount, the boxplot shows that the outliers are values >280 thereabout. The maximum value is 280. The histogram for loanamount also shows that there are outliers. The outliers are greater than the maximum value, that means that they are exremely larger than the maximum values.

In [22]:
```
data['Credit_History'].value_counts()
```

Out[22]:
```
1.0    427
0.0     79
Name: Credit_History, dtype: int64
```

In [23]:
```
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     79
1.0    427
Name: Credit_History, dtype: int64

Probability of getting loan for each Credit History class:
                Loan_Status
Credit_History
0.0                0.050633
1.0                0.793911
```

In [24]:
```
data['Loan_Status'].value_counts()
```

Out[24]:
```
Y    374
N    176
Name: Loan_Status, dtype: int64
```

In [25]:
```
data.shape
```

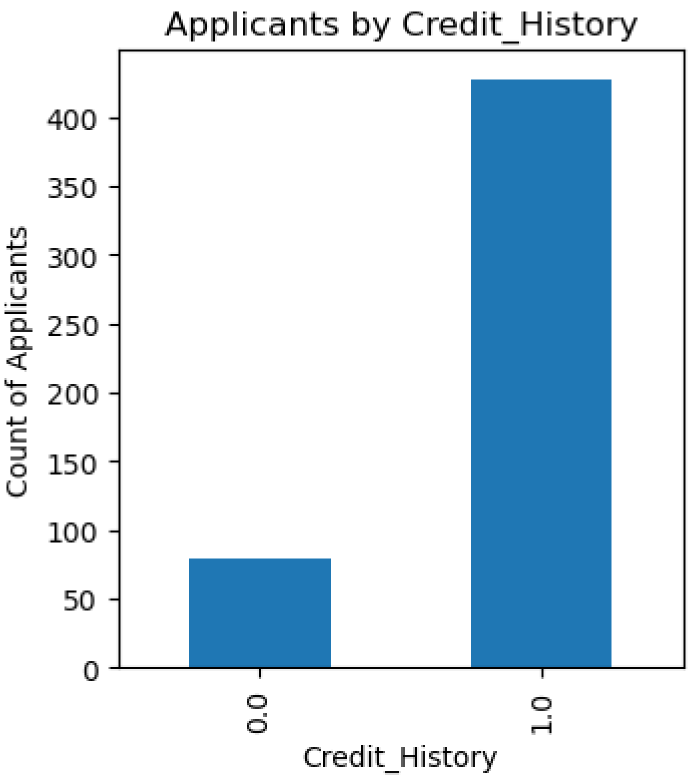
Out[25]:
```
(550, 13)
```

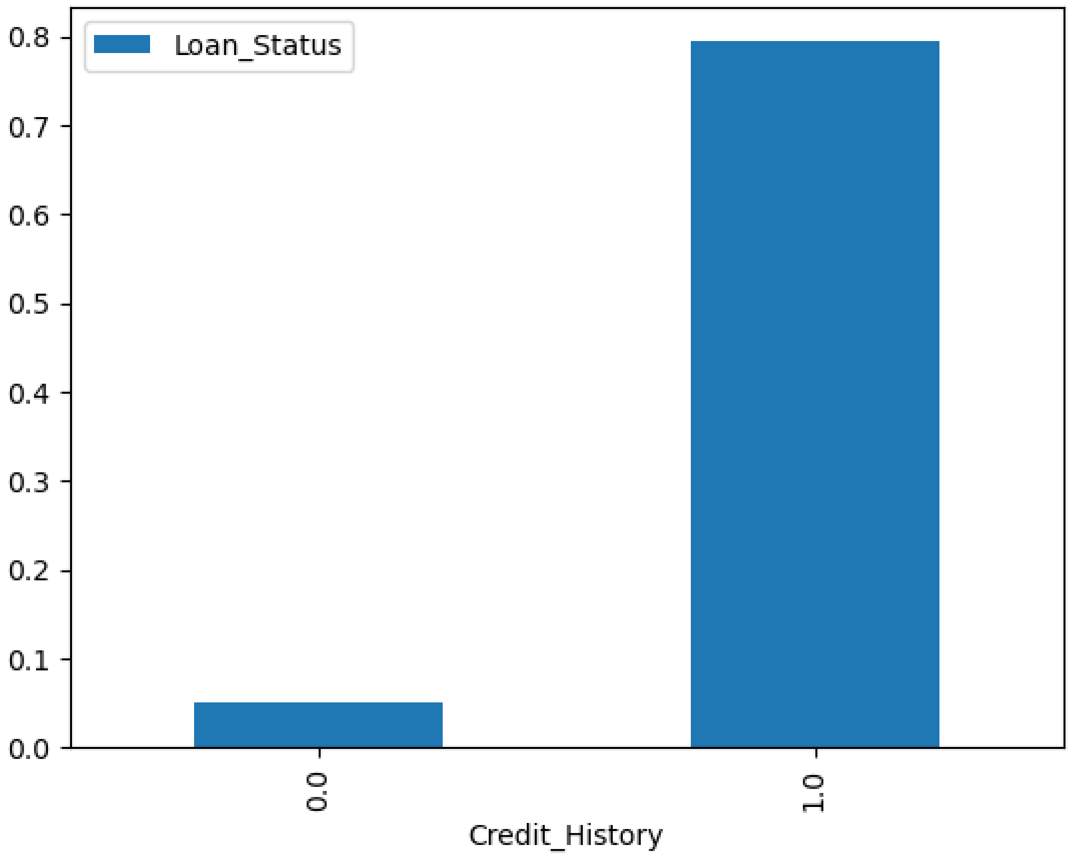
In [26]:
```
data.head()
```

Out[26]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Cr
0	LP001924	Male	No	0	Graduate	No	3158	3053.0	89.0	360.0	
1	LP002055	Female	No	0	Graduate	No	3166	2985.0	132.0	360.0	
2	LP002537	Male	Yes	0	Graduate	No	2083	3150.0	128.0	360.0	
3	LP001594	Male	Yes	0	Graduate	No	5708	5625.0	187.0	360.0	
4	LP002683	Male	No	0	Graduate	No	4683	1915.0	185.0	360.0	

In [27]:
```
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()
```





In [28]:
```
data['Gender'].value_counts()
```

Out[28]:
```
Male      434
Female    104
Name: Gender, dtype: int64
```

In [29]:
```python
data.apply(lambda x: sum(x.isnull()), axis=0)
```

Out[29]:
```
Loan_ID               0
Gender               12
Married               3
Dependents           14
Education             0
Self_Employed        30
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           22
Loan_Amount_Term     12
Credit_History       44
Property_Area         0
Loan_Status           0
dtype: int64
```

In [30]:
```python
data.head()
```

Out[30]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Cr
0	LP001924	Male	No	0	Graduate	No	3158	3053.0	89.0	360.0	
1	LP002055	Female	No	0	Graduate	No	3166	2985.0	132.0	360.0	
2	LP002537	Male	Yes	0	Graduate	No	2083	3150.0	128.0	360.0	
3	LP001594	Male	Yes	0	Graduate	No	5708	5625.0	187.0	360.0	
4	LP002683	Male	No	0	Graduate	No	4683	1915.0	185.0	360.0	

In [31]:
```python
data['LoanAmount'].fillna(data['LoanAmount'].mean(), inplace = True)
```

In [32]:
```
data.head()
```

Out[32]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Cr
0	LP001924	Male	No	0	Graduate	No	3158	3053.0	89.0	360.0	
1	LP002055	Female	No	0	Graduate	No	3166	2985.0	132.0	360.0	
2	LP002537	Male	Yes	0	Graduate	No	2083	3150.0	128.0	360.0	
3	LP001594	Male	Yes	0	Graduate	No	5708	5625.0	187.0	360.0	
4	LP002683	Male	No	0	Graduate	No	4683	1915.0	185.0	360.0	

In [33]:
```
data.apply(lambda x: sum(x.isnull()), axis=0)
```

Out[33]:
```
Loan_ID               0
Gender               12
Married               3
Dependents           14
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
```

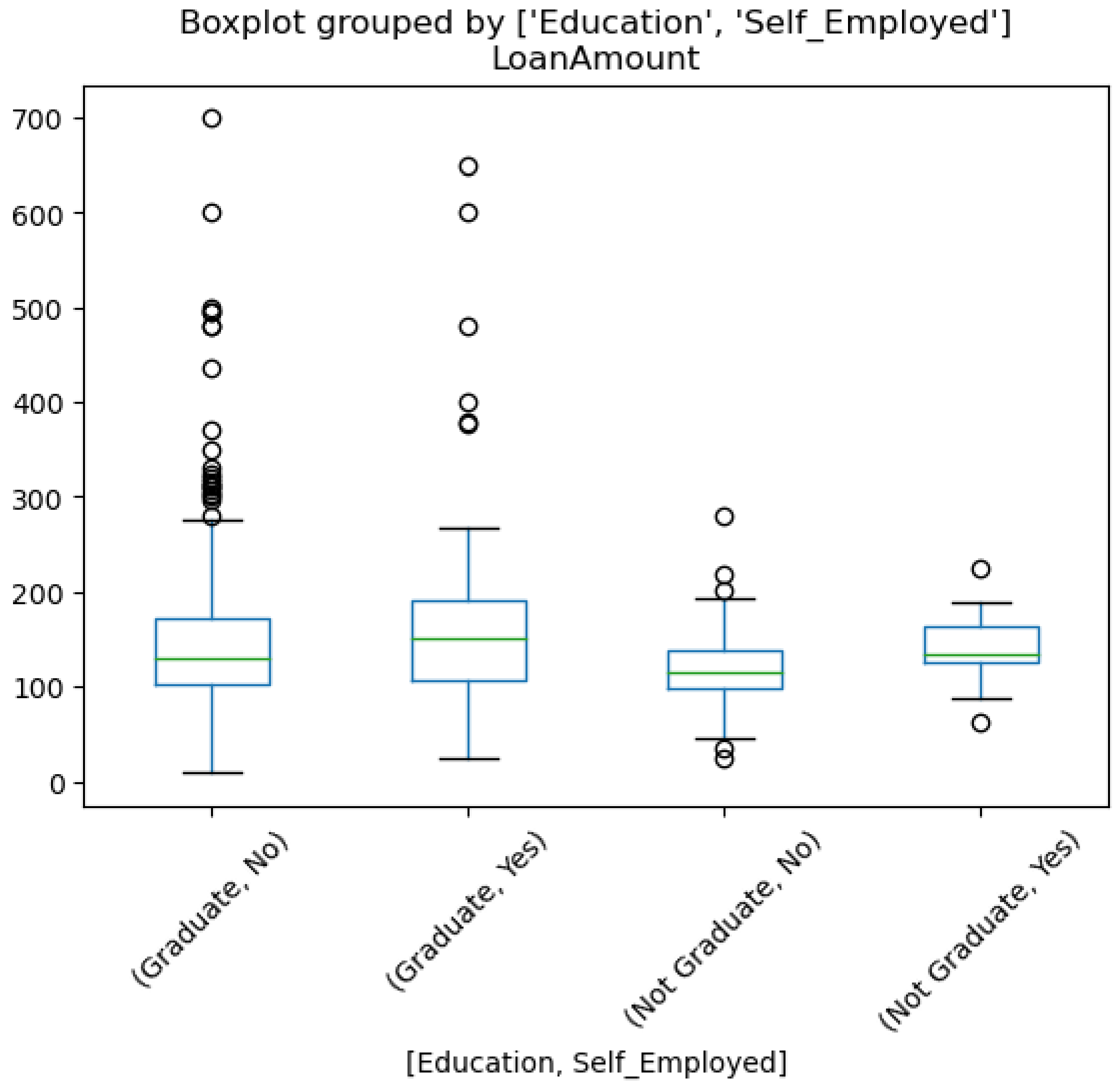
In [34]:
```
data.shape
```

Out[34]:
```
(550, 13)
```

In [35]:
```
data.to_csv('new_train.csv')
```

In [36]:
```
data.boxplot(column='LoanAmount', by = ['Education','Self_Employed'],
 grid=False, rot = 45, fontsize = 10)
```

Out[36]: <AxesSubplot:title={'center':'LoanAmount'}, xlabel='[Education, Self_Employed]'>



In [37]:
```
data['Self_Employed'].value_counts()
```

Out[37]:
```
No     448
Yes     72
Name: Self_Employed, dtype: int64
```

In [38]:
```
data['Self_Employed'].fillna('No', inplace=True)
```

In [39]:
```
data['Self_Employed'].value_counts()
```

Out[39]:
```
No     478
Yes     72
Name: Self_Employed, dtype: int64
```

In [40]:
```
data.apply(lambda x: sum(x.isnull()), axis=0)
```

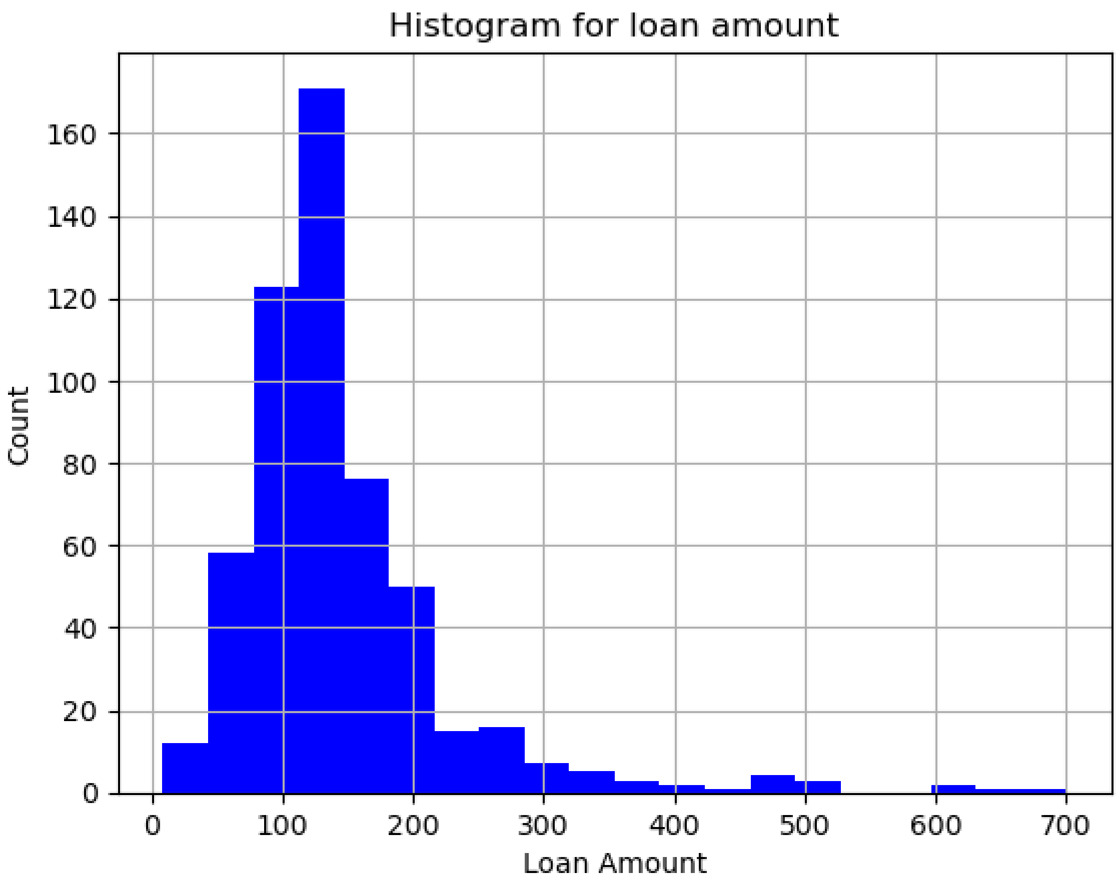
Out[40]:
```
Loan_ID               0
Gender               12
Married               3
Dependents           14
Education             0
Self_Employed         0
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount            0
Loan_Amount_Term     12
Credit_History       44
Property_Area         0
Loan_Status           0
dtype: int64
```

In [41]:
```
data.describe()
```

Out[41]:

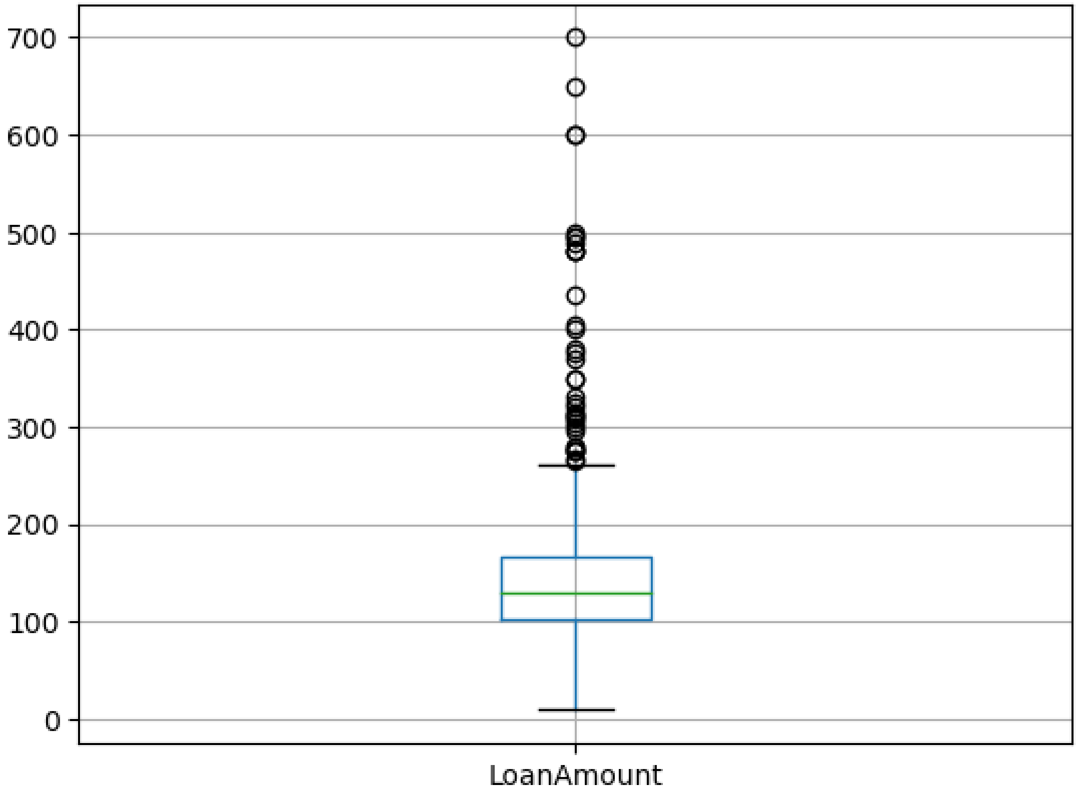
	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	550.000000	550.000000	550.000000	538.000000	506.000000
mean	5354.065455	1624.330764	146.098485	342.401487	0.843874
std	5475.802136	3019.983826	84.608332	65.193956	0.363334
min	416.000000	0.000000	9.000000	12.000000	0.000000
25%	2894.250000	0.000000	102.000000	360.000000	1.000000
50%	3750.000000	1128.500000	128.500000	360.000000	1.000000
75%	5806.250000	2250.000000	165.750000	360.000000	1.000000
max	63337.000000	41667.000000	700.000000	480.000000	1.000000

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



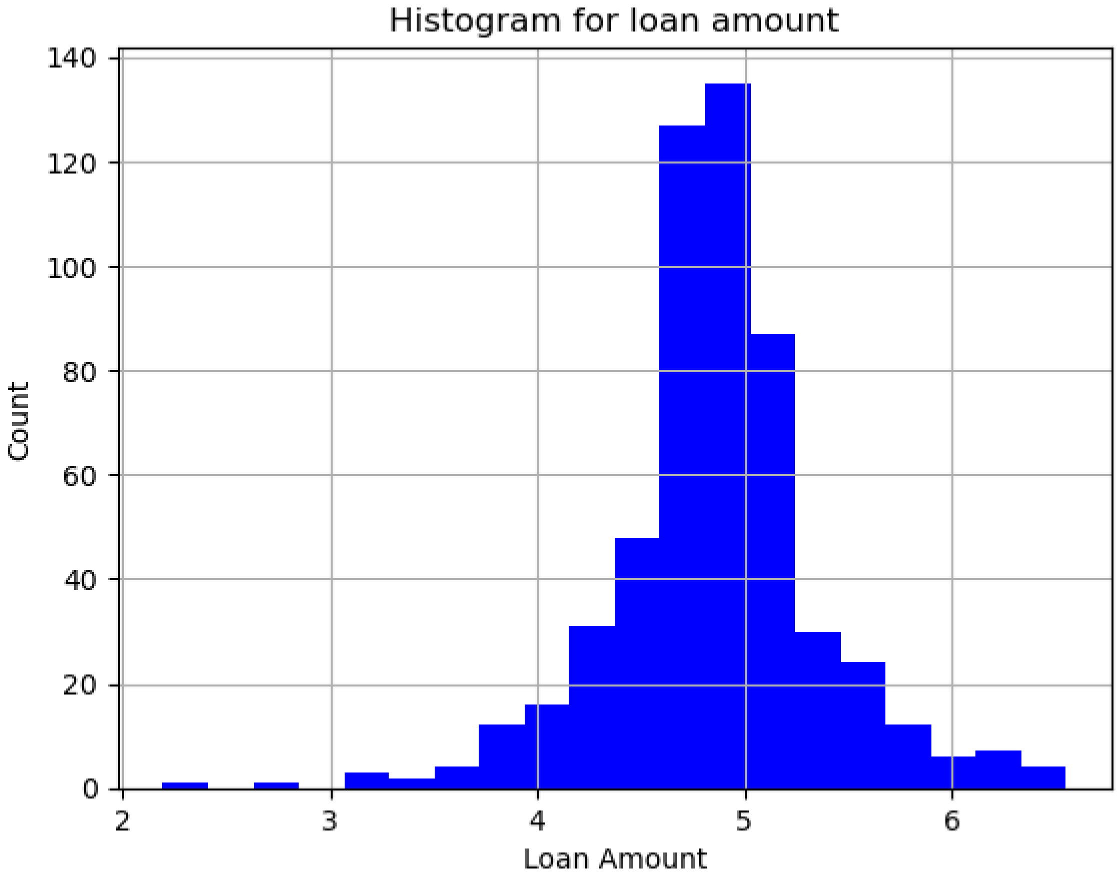
In [43]:
```
data.boxplot(column='LoanAmount')
```

Out[43]: <AxesSubplot:>

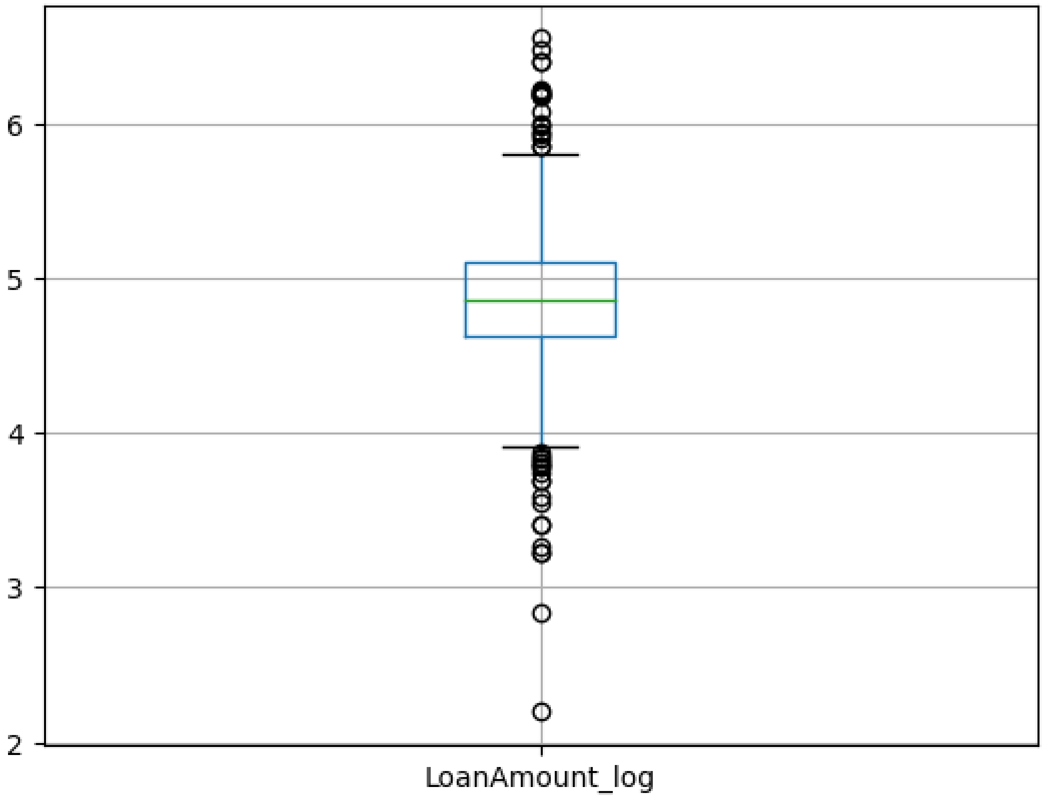


In [44]:
```
data['LoanAmount_log'] = np.log(data['LoanAmount'])
#data['LoanAmount_log'].hist(bins = 20)
```

In [45]:
```
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 [46]:
```
data.boxplot(column='LoanAmount_log')
```
Out[46]: <AxesSubplot:>



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

Out[47]:
```
Loan_ID               0
Gender               12
Married               3
Dependents           14
Education             0
Self_Employed         0
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount            0
Loan_Amount_Term     12
Credit_History       44
Property_Area         0
Loan_Status           0
LoanAmount_log        0
dtype: int64
```

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

Out[48]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Cr
0	LP001924	Male	No	0	Graduate	No	3158	3053.0	89.0	360.0	
1	LP002055	Female	No	0	Graduate	No	3166	2985.0	132.0	360.0	
2	LP002537	Male	Yes	0	Graduate	No	2083	3150.0	128.0	360.0	
3	LP001594	Male	Yes	0	Graduate	No	5708	5625.0	187.0	360.0	
4	LP002683	Male	No	0	Graduate	No	4683	1915.0	185.0	360.0	

In [49]:
```
data.describe()
```

Out[49]:

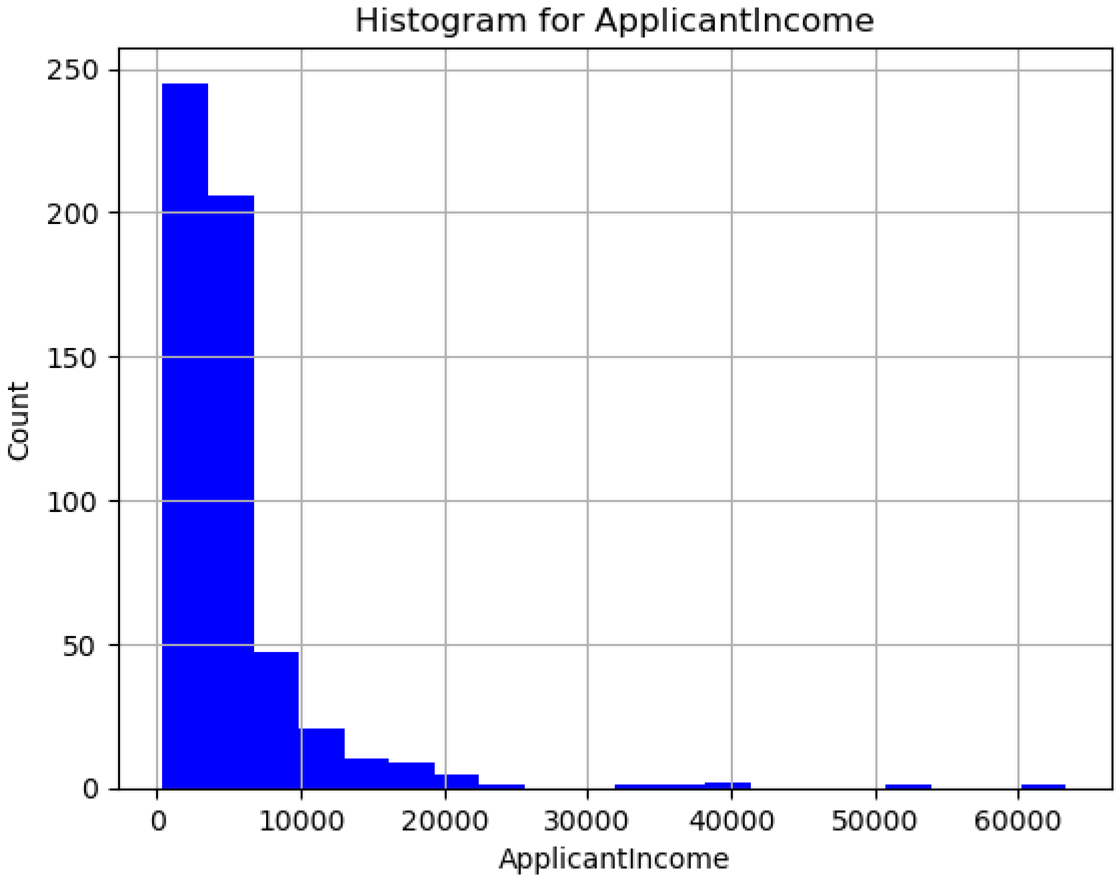
	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	LoanAmount_log
count	550.000000	550.000000	550.000000	538.000000	506.000000	550.000000
mean	5354.065455	1624.330764	146.098485	342.401487	0.843874	4.856427
std	5475.802136	3019.983826	84.608332	65.193956	0.363334	0.505835
min	416.000000	0.000000	9.000000	12.000000	0.000000	2.197225
25%	2894.250000	0.000000	102.000000	360.000000	1.000000	4.624973
50%	3750.000000	1128.500000	128.500000	360.000000	1.000000	4.855921
75%	5806.250000	2250.000000	165.750000	360.000000	1.000000	5.110477
max	63337.000000	41667.000000	700.000000	480.000000	1.000000	6.551080

In [50]:
```
data = data.drop(['LoanAmount'], axis=1)
```

In [51]:
```
data.apply(lambda x: sum(x.isnull()), axis=0)
```

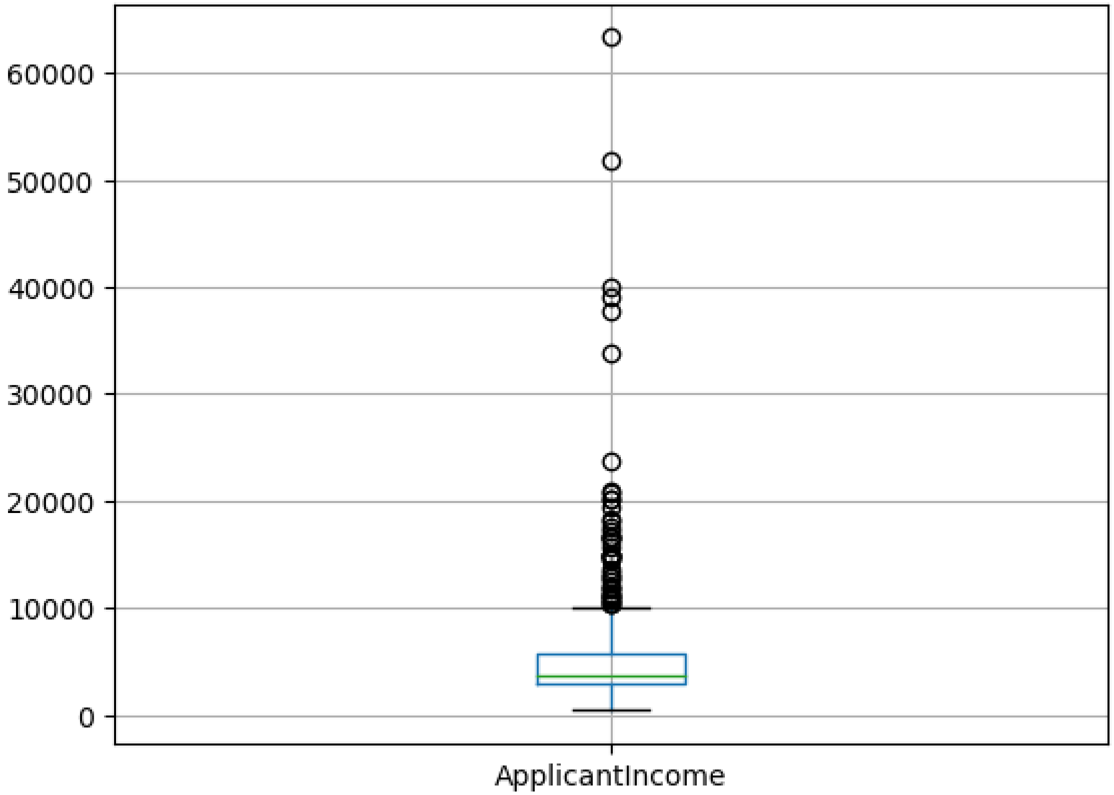
Out[51]:
```
Loan_ID               0
Gender               12
Married               3
Dependents           14
Education             0
Self_Employed         0
ApplicantIncome       0
CoapplicantIncome     0
Loan_Amount_Term     12
Credit_History       44
Property_Area         0
Loan_Status           0
LoanAmount_log        0
dtype: int64
```

In [52]:
```
#To view histogram of ApplicantIncome to check for outliers
plt.hist(data['ApplicantIncome'], 20, facecolor='b')
plt.xlabel('ApplicantIncome')
plt.ylabel('Count')
plt.title('Histogram for ApplicantIncome')
plt.grid(True)
plt.show()
```



In [53]:
```
#To view boxplot of ApplicantIncome to check for outliers
data.boxplot(column='ApplicantIncome')
```

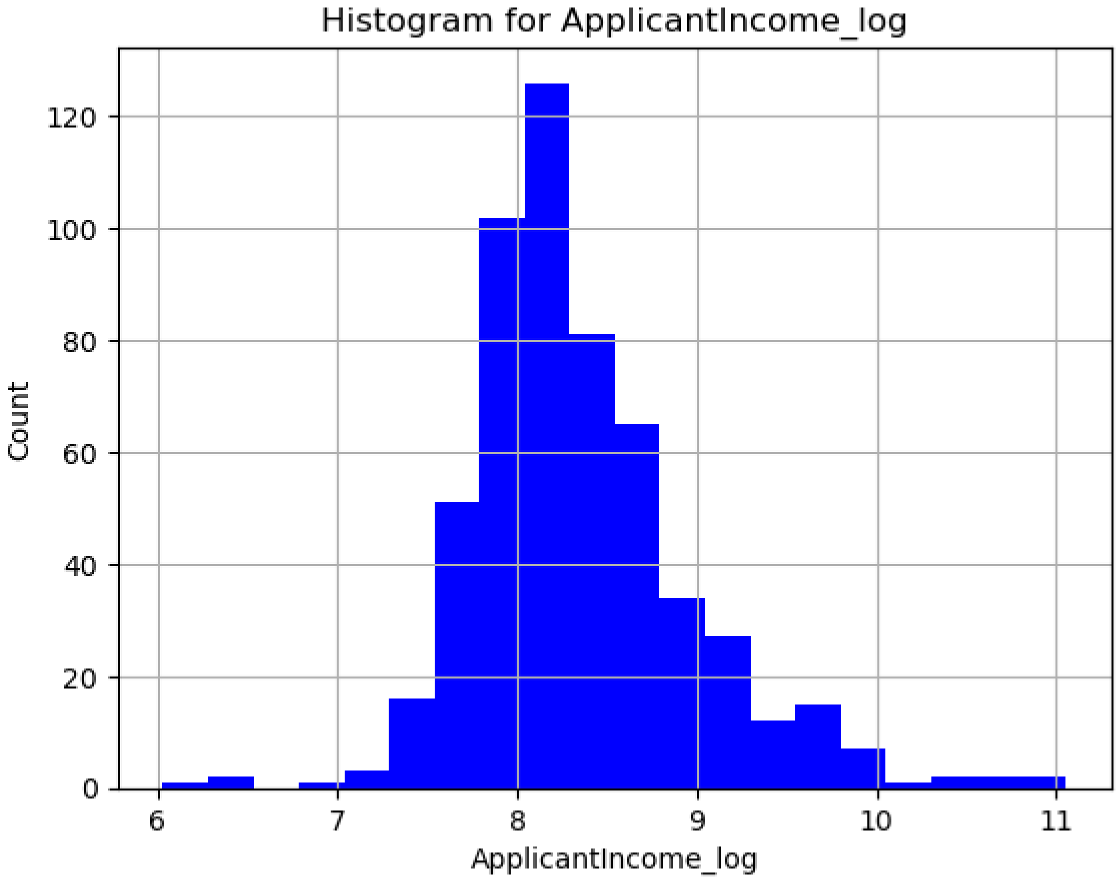
Out[53]: <AxesSubplot:>



We can see from the histogram and boxplot for ApplicantIncome that it has a lot of outliers. We would use the ApplicantIncome_log to try and reduce the outliers.

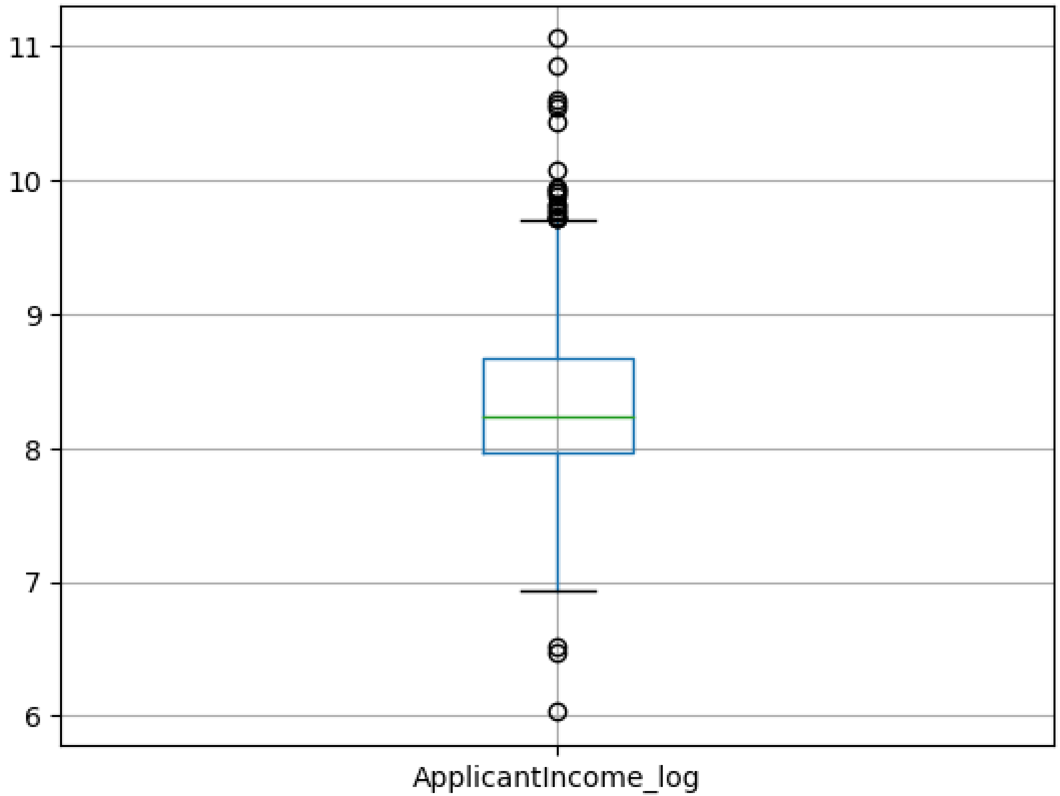
In [54]:
```
data['ApplicantIncome_log'] = np.log(data['ApplicantIncome'])
#data['ApplicantIncome_log'].hist(bins = 20)
```

In [55]:
```
plt.hist(data['ApplicantIncome_log'], 20, facecolor='b')
plt.xlabel('ApplicantIncome_log')
plt.ylabel('Count')
plt.title('Histogram for ApplicantIncome_log')
plt.grid(True)
plt.show()
```



In [56]:
```
data.boxplot(column='ApplicantIncome_log')
```

Out[56]: <AxesSubplot:>



The outliers are greatly reduced.

Generate a new variable by combining two variables e.g., 'ApplicantIncome' and 'CoapplicantIncome', we can add the 2 incomes together

In [57]:
```
data['TotalIncome'] = data['ApplicantIncome'] + data['CoapplicantIncome']
```

In [58]:
```
data.to_csv('MoradekeAdeleye_2229810.csv', index=False)
```

In [59]:
```
data.head()
```

Out[59]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	Loan_Amount_Term	Credit_History	F
0	LP001924	Male	No	0	Graduate	No	3158	3053.0	360.0	1.0	
1	LP002055	Female	No	0	Graduate	No	3166	2985.0	360.0	NaN	
2	LP002537	Male	Yes	0	Graduate	No	2083	3150.0	360.0	1.0	
3	LP001594	Male	Yes	0	Graduate	No	5708	5625.0	360.0	1.0	
4	LP002683	Male	No	0	Graduate	No	4683	1915.0	360.0	1.0	

In [60]:
```
data.describe()
```

Out[60]:

	ApplicantIncome	CoapplicantIncome	Loan_Amount_Term	Credit_History	LoanAmount_log	ApplicantIncome_log	TotalIncome
count	550.000000	550.000000	538.000000	506.000000	550.000000	550.000000	550.000000
mean	5354.065455	1624.330764	342.401487	0.843874	4.856427	8.351033	6978.396218
std	5475.802136	3019.983826	65.193956	0.363334	0.505835	0.614144	5927.382630
min	416.000000	0.000000	12.000000	0.000000	2.197225	6.030685	1442.000000
25%	2894.250000	0.000000	360.000000	1.000000	4.624973	7.970481	4161.500000
50%	3750.000000	1128.500000	360.000000	1.000000	4.855921	8.229511	5416.500000
75%	5806.250000	2250.000000	360.000000	1.000000	5.110477	8.666687	7550.750000
max	63337.000000	41667.000000	480.000000	1.000000	6.551080	11.056225	63337.000000

In [61]:
```
data = data.drop(['ApplicantIncome', 'CoapplicantIncome'], axis=1)
```

In [62]:
```python
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)
```

Q5. Use LabelEncoder, to convert categorical variables into numeric. Hint: You will first need to identify categorial values. See line #72 for the output after the categorical variables were transformed to numeric

In [64]:
```python
data.head()
```

Out[64]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status	LoanAmou
0	LP001924	Male	No	0	Graduate	No	360.0	1.0	Rural	Y	4.
1	LP002055	Female	No	0	Graduate	No	360.0	1.0	Rural	Y	4.
2	LP002537	Male	Yes	0	Graduate	No	360.0	1.0	Semiurban	Y	4.
3	LP001594	Male	Yes	0	Graduate	No	360.0	1.0	Semiurban	Y	5.
4	LP002683	Male	No	0	Graduate	No	360.0	1.0	Semiurban	N	5.

In [65]:
```python
data.shape
```

Out[65]: (550, 13)

In [66]:
```python
from sklearn.preprocessing import LabelEncoder
```

In [67]:
```python
columns = list(data)
print(columns)
```

```
['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education', 'Self_Employed', 'Loan_Amount_Term', 'Credit_History', 'P
roperty_Area', 'Loan_Status', 'LoanAmount_log', 'ApplicantIncome_log', 'TotalIncome']
```

In [68]:
```
data.dtypes
```

Out[68]:
```
Loan_ID                  object
Gender                   object
Married                  object
Dependents               object
Education                object
Self_Employed            object
Loan_Amount_Term        float64
Credit_History          float64
Property_Area            object
Loan_Status              object
LoanAmount_log          float64
ApplicantIncome_log     float64
TotalIncome             float64
dtype: object
```

In [69]:
```
columns = list(data.select_dtypes(exclude=['float64','int64']))
```

In [70]:
```
print(columns)
```

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

In [71]:
```
le = LabelEncoder()
for i in columns:
    #print(i)
    data[i] = le.fit_transform(data[i])
```

In [72]:
```
data.head()
```

Out[72]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status	LoanAmou
0	260	1	0	0	0	0	360.0	1.0	0	1	4.4
1	295	0	0	0	0	0	360.0	1.0	0	1	4.8
2	435	1	1	0	0	0	360.0	1.0	1	1	4.8
3	155	1	1	0	0	0	360.0	1.0	1	1	5.2
4	468	1	0	0	0	0	360.0	1.0	1	0	5.2

In [73]:
```
#from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import normalize
```

In [74]:
```
original_data = data.copy()
original_data.head()
```

Out[74]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status	LoanAmou
0	260	1	0	0	0	0	360.0	1.0	0	1	4.4
1	295	0	0	0	0	0	360.0	1.0	0	1	4.8
2	435	1	1	0	0	0	360.0	1.0	1	1	4.8
3	155	1	1	0	0	0	360.0	1.0	1	1	5.2
4	468	1	0	0	0	0	360.0	1.0	1	0	5.2

In [75]:
```
original_data[0:5]
```

Out[75]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status	LoanAmou
0	260	1	0	0	0	0	360.0	1.0	0	1	4.4
1	295	0	0	0	0	0	360.0	1.0	0	1	4.8
2	435	1	1	0	0	0	360.0	1.0	1	1	4.8
3	155	1	1	0	0	0	360.0	1.0	1	1	5.2
4	468	1	0	0	0	0	360.0	1.0	1	0	5.2

In [76]:
```
data[0:5]
```

Out[76]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status	LoanAmou
0	260	1	0	0	0	0	360.0	1.0	0	1	4.4
1	295	0	0	0	0	0	360.0	1.0	0	1	4.8
2	435	1	1	0	0	0	360.0	1.0	1	1	4.8
3	155	1	1	0	0	0	360.0	1.0	1	1	5.2
4	468	1	0	0	0	0	360.0	1.0	1	0	5.2

In [77]:
```
data_for_norm = data.drop(['Loan_ID','Loan_Status'], axis=1)
```

Loan _ID was dropped and not normalised because it is a primary key. It is a unique identifier for each row.

In [78]:
```
normalized_data = normalize( data_for_norm )
```

In [79]:
```
print(normalized_data[0:5])
```

```
[[1.60734716e-04 0.00000000e+00 0.00000000e+00 0.00000000e+00
  0.00000000e+00 5.78644977e-02 1.60734716e-04 0.00000000e+00
  7.21479691e-04 1.29515119e-03 9.98323320e-01]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
  0.00000000e+00 5.84270166e-02 1.62297268e-04 0.00000000e+00
  7.92465414e-04 1.30815238e-03 9.98290498e-01]
 [1.90644083e-04 1.90644083e-04 0.00000000e+00 0.00000000e+00
  0.00000000e+00 6.86318700e-02 1.90644083e-04 1.90644083e-04
  9.25010862e-04 1.45681905e-03 9.97640488e-01]
 [8.81933681e-05 8.81933681e-05 0.00000000e+00 0.00000000e+00
  0.00000000e+00 3.17496125e-02 8.81933681e-05 8.81933681e-05
  4.61349088e-04 7.62839472e-04 9.99495441e-01]
 [1.51335806e-04 0.00000000e+00 0.00000000e+00 0.00000000e+00
  0.00000000e+00 5.44808901e-02 1.51335806e-04 1.51335806e-04
  7.90026756e-04 1.27904395e-03 9.98513647e-01]]
```

In [80]:
```
normalized_data.shape
```

Out[80]:
```
(550, 11)
```

In [81]:
```
data.shape
```

Out[81]:
```
(550, 13)
```

In [82]:
```
normalized_data = pd.DataFrame(normalized_data, columns=data_for_norm.columns)
```

In [83]:
```
normalized_data.head()
```

Out[83]:

	Gender	Married	Dependents	Education	Self_Employed	Loan_Amount_Term	Credit_History	Property_Area	LoanAmount_log	ApplicantIncor
0	0.000161	0.000000	0.0	0.0	0.0	0.057864	0.000161	0.000000	0.000721	0.(
1	0.000000	0.000000	0.0	0.0	0.0	0.058427	0.000162	0.000000	0.000792	0.(
2	0.000191	0.000191	0.0	0.0	0.0	0.068632	0.000191	0.000191	0.000925	0.(
3	0.000088	0.000088	0.0	0.0	0.0	0.031750	0.000088	0.000088	0.000461	0.(
4	0.000151	0.000000	0.0	0.0	0.0	0.054481	0.000151	0.000151	0.000790	0.(

In [84]:
```
normalized_data['Loan_ID'] = data['Loan_ID']
```

In [85]:
```
normalized_data['Loan_Status'] = data['Loan_Status']
```

In [86]:
```
normalized_data.head()
```

Out[86]:

	Gender	Married	Dependents	Education	Self_Employed	Loan_Amount_Term	Credit_History	Property_Area	LoanAmount_log	ApplicantIncor
0	0.000161	0.000000	0.0	0.0	0.0	0.057864	0.000161	0.000000	0.000721	0.(
1	0.000000	0.000000	0.0	0.0	0.0	0.058427	0.000162	0.000000	0.000792	0.(
2	0.000191	0.000191	0.0	0.0	0.0	0.068632	0.000191	0.000191	0.000925	0.(
3	0.000088	0.000088	0.0	0.0	0.0	0.031750	0.000088	0.000088	0.000461	0.(
4	0.000151	0.000000	0.0	0.0	0.0	0.054481	0.000151	0.000151	0.000790	0.(

In [87]:
```
normalized_data.describe()
```

Out[87]:

	Gender	Married	Dependents	Education	Self_Employed	Loan_Amount_Term	Credit_History	Property_Area	LoanAmount_log	Appl
count	550.000000	550.000000	550.000000	550.000000	550.000000	550.000000	550.000000	550.000000	550.000000	
mean	0.000150	0.000116	0.000132	0.000049	0.000021	0.067050	0.000165	0.000209	0.000915	
std	0.000107	0.000107	0.000214	0.000103	0.000062	0.035467	0.000108	0.000208	0.000383	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.001604	0.000000	0.000000	0.000098	
25%	0.000070	0.000000	0.000000	0.000000	0.000000	0.042994	0.000094	0.000000	0.000664	
50%	0.000157	0.000123	0.000000	0.000000	0.000000	0.063214	0.000164	0.000180	0.000896	
75%	0.000212	0.000194	0.000218	0.000000	0.000000	0.083670	0.000231	0.000321	0.001115	
max	0.000673	0.000522	0.001608	0.000673	0.000455	0.242215	0.000673	0.001346	0.002672	

In [88]:
```
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
```

In [89]:
```
columns = list(normalized_data.columns)
columns
```
Out[89]:
```
['Gender',
 'Married',
 'Dependents',
 'Education',
 'Self_Employed',
 'Loan_Amount_Term',
 'Credit_History',
 'Property_Area',
 'LoanAmount_log',
 'ApplicantIncome_log',
 'TotalIncome',
 'Loan_ID',
 'Loan_Status']
```

In [90]:
```
normalized_data.head()
```
Out[90]:

	Gender	Married	Dependents	Education	Self_Employed	Loan_Amount_Term	Credit_History	Property_Area	LoanAmount_log	ApplicantIncor
0	0.000161	0.000000	0.0	0.0	0.0	0.057864	0.000161	0.000000	0.000721	0.(
1	0.000000	0.000000	0.0	0.0	0.0	0.058427	0.000162	0.000000	0.000792	0.(
2	0.000191	0.000191	0.0	0.0	0.0	0.068632	0.000191	0.000191	0.000925	0.(
3	0.000088	0.000088	0.0	0.0	0.0	0.031750	0.000088	0.000088	0.000461	0.(
4	0.000151	0.000000	0.0	0.0	0.0	0.054481	0.000151	0.000151	0.000790	0.(

In [91]:
```
features = normalized_data.drop(['Loan_ID','Loan_Status'], axis = 1)

classes = pd.DataFrame(normalized_data['Loan_Status'])
```

In [92]:
```python
print('Features:')
print(features.head())

print('Classes:')
print(classes.head())
```

```
Features:
     Gender   Married  Dependents  Education  Self_Employed  Loan_Amount_Term  \
0  0.000161  0.000000         0.0        0.0            0.0          0.057864
1  0.000000  0.000000         0.0        0.0            0.0          0.058427
2  0.000191  0.000191         0.0        0.0            0.0          0.068632
3  0.000088  0.000088         0.0        0.0            0.0          0.031750
4  0.000151  0.000000         0.0        0.0            0.0          0.054481

   Credit_History  Property_Area  LoanAmount_log  ApplicantIncome_log  \
0        0.000161       0.000000        0.000721             0.001295
1        0.000162       0.000000        0.000792             0.001308
2        0.000191       0.000191        0.000925             0.001457
3        0.000088       0.000088        0.000461             0.000763
4        0.000151       0.000151        0.000790             0.001279

   TotalIncome
0     0.998323
1     0.998290
2     0.997640
3     0.999495
4     0.998514
Classes:
   Loan_Status
0            1
1            1
2            1
3            1
4            0
```

In [93]:
```
normalized_data.head(10)
```

Out[93]:

	Gender	Married	Dependents	Education	Self_Employed	Loan_Amount_Term	Credit_History	Property_Area	LoanAmount_log	ApplicantIncor
0	0.000161	0.000000	0.000000	0.0	0.0	0.057864	0.000161	0.000000	0.000721	0.(
1	0.000000	0.000000	0.000000	0.0	0.0	0.058427	0.000162	0.000000	0.000792	0.(
2	0.000191	0.000191	0.000000	0.0	0.0	0.068632	0.000191	0.000191	0.000925	0.(
3	0.000088	0.000088	0.000000	0.0	0.0	0.031750	0.000088	0.000088	0.000461	0.(
4	0.000151	0.000000	0.000000	0.0	0.0	0.054481	0.000151	0.000151	0.000790	0.(
5	0.000258	0.000000	0.000000	0.0	0.0	0.092909	0.000258	0.000258	0.001118	0.(
6	0.000213	0.000213	0.000639	0.0	0.0	0.076696	0.000000	0.000213	0.001034	0.(
7	0.000334	0.000334	0.000000	0.0	0.0	0.120171	0.000334	0.000334	0.001306	0.(
8	0.000140	0.000140	0.000280	0.0	0.0	0.050462	0.000140	0.000000	0.000732	0.(
9	0.000298	0.000000	0.000000	0.0	0.0	0.107386	0.000298	0.000597	0.001267	0.(

In [94]:
```
normalized_data.shape
```

Out[94]:
```
(550, 13)
```

In [95]:
```
from matplotlib import pyplot
```

In [96]:
```
x_train, x_test, y_train, y_test = train_test_split(features, classes, test_size= .33,
                                                    random_state = 10)


print(x_train.shape, x_test.shape)
```

```
(368, 11) (182, 11)
```

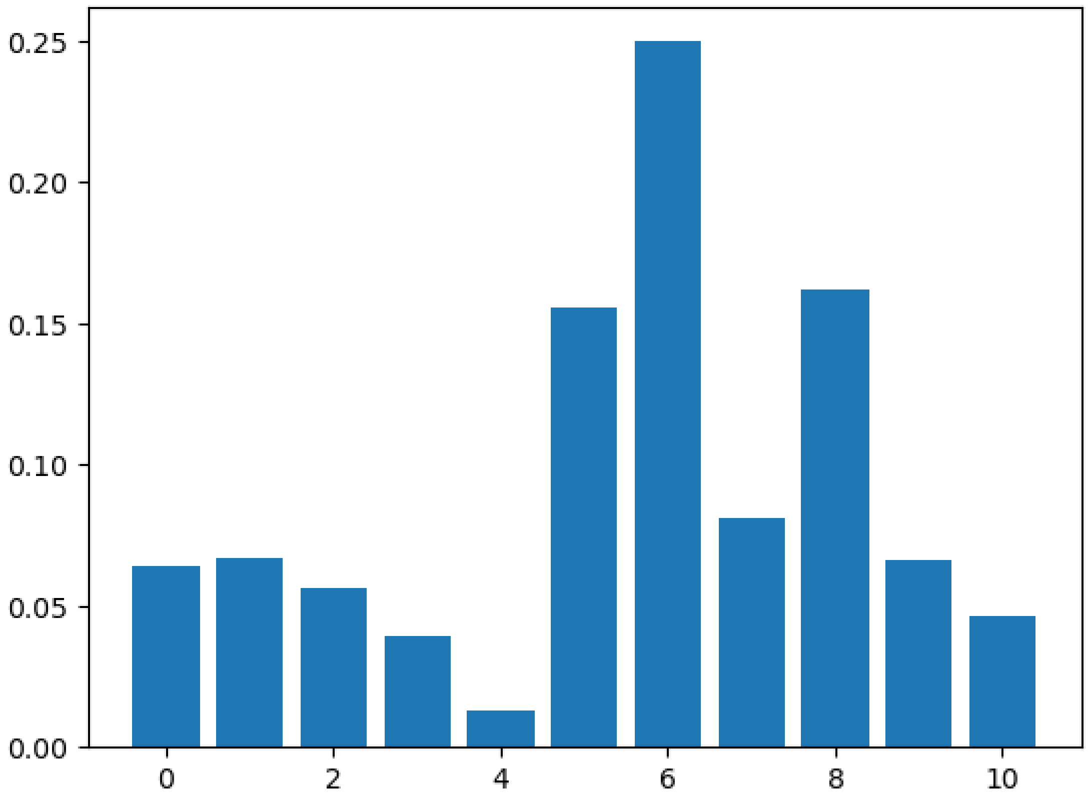
In [97]:
```
decisionTree = DecisionTreeClassifier(criterion='entropy')
print(decisionTree)
```

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

In [98]:
```
dtc_model = decisionTree.fit(x_train, y_train)
```

In [99]:
```
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: 0.06426
Feature: 1, Score: 0.06725
Feature: 2, Score: 0.05597
Feature: 3, Score: 0.03900
Feature: 4, Score: 0.01262
Feature: 5, Score: 0.15550
Feature: 6, Score: 0.24983
Feature: 7, Score: 0.08143
Feature: 8, Score: 0.16183
Feature: 9, Score: 0.06635
Feature: 10, Score: 0.04598
```

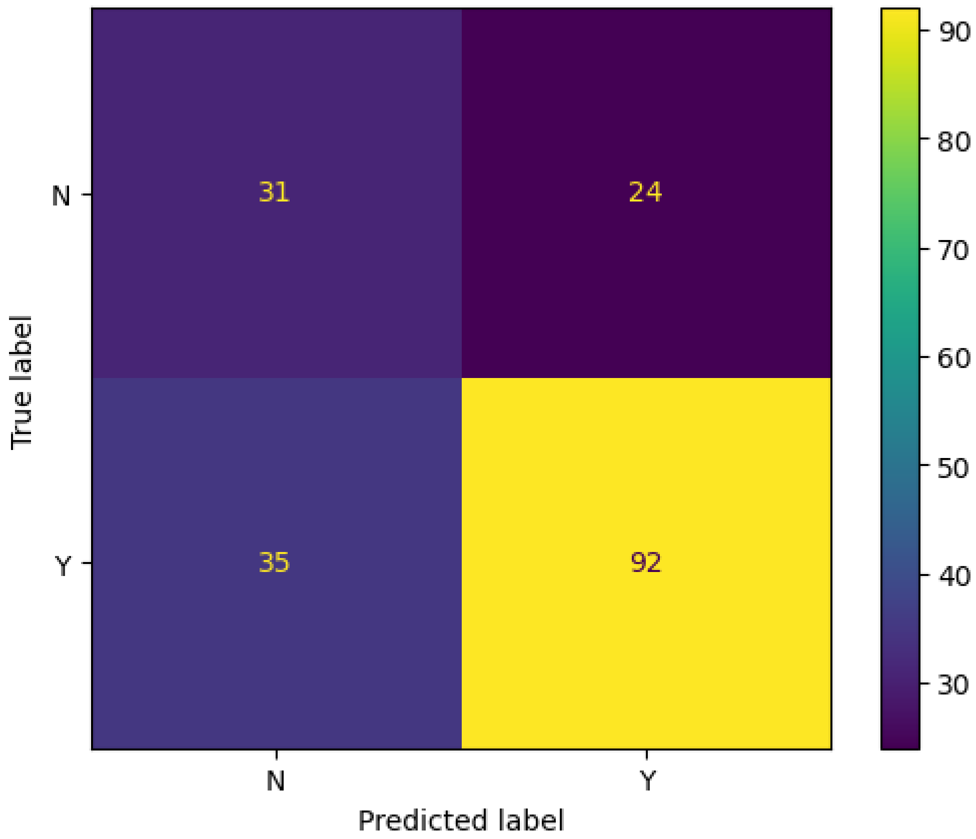


In [100]:
```
prediction = dtc_model.predict(x_test)
```

In [101]:
```
y_true = le.inverse_transform(y_test["Loan_Status"])
y_pred = le.inverse_transform(prediction)
```

In [102]:
```
cm = confusion_matrix(y_true, y_pred)
labels = ['N', 'Y']
ConfusionMatrixDisplay(cm, display_labels=labels).plot()
```

Out[102]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x21d67637d00>



In [103]:
```
print(classification_report(y_true, y_pred))
```

```
              precision    recall  f1-score   support

           N       0.47      0.56      0.51        55
           Y       0.79      0.72      0.76       127

    accuracy                           0.68       182
   macro avg       0.63      0.64      0.63       182
weighted avg       0.70      0.68      0.68       182
```

In [104]:
```
graphviz_path = 'C:/Program Files/Graphviz/bin/'
```

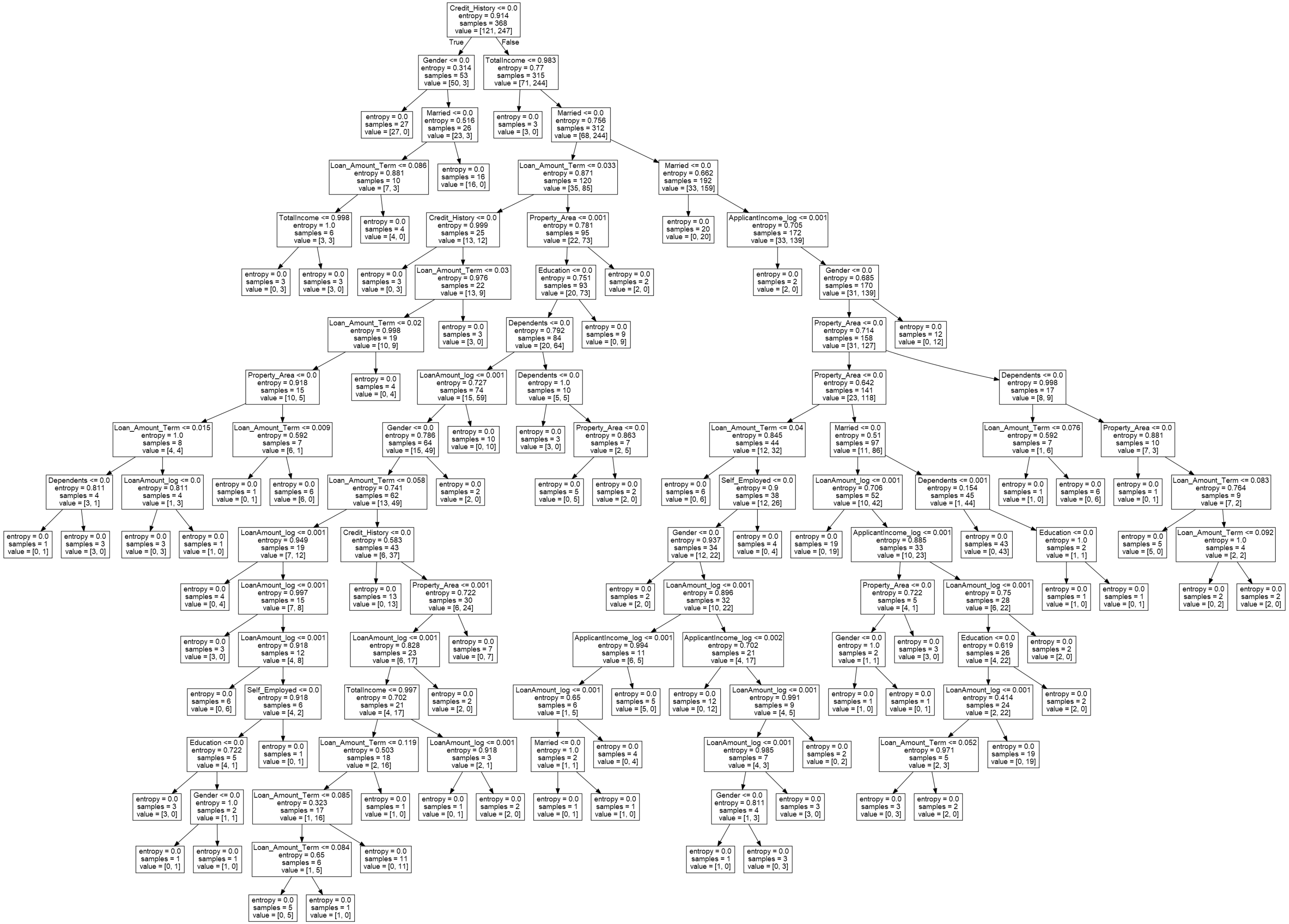
In [105]:
```
import os

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

In [106]:
```
from graphviz import Source
from sklearn import tree
graph = Source( tree.export_graphviz(dtc_model, out_file=None, feature_names=features.columns))
```

In [107]:

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


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

Out[107]:


Credit_History <= 0.0 entropy = 0.914 samples = 368 value = [121, 247]
True
False
Gender <= 0.0 entropy = 0.314 samples = 53 value = [50, 3]
TotalIncome <= 0.983 entropy = 0.77 samples = 315 value = [71, 244]
entropy = 0.0 samples = 27 value = [27, 0]
Married <= 0.0 entropy = 0.516 samples = 26 value = [23, 3]
entropy = 0.0 samples = 3 value = [3, 0]
Married <= 0.0 entropy = 0.756 samples = 312 value = [68, 244]
Loan_Amount_Term <= 0.086 entropy = 0.881 samples = 10 value = [7, 3]
entropy = 0.0 samples = 16 value = [16, 0]
Loan_Amount_Term <= 0.033 entropy = 0.871 samples = 120 value = [35, 85]
Married <= 0.0 entropy = 0.662 samples = 192 value = [33, 159]
TotalIncome <= 0.998 entropy = 1.0 samples = 6 value = [3, 3]
entropy = 0.0 samples = 4 value = [4, 0]
Credit_History <= 0.0 entropy = 0.999 samples = 25 value = [13, 12]
Property_Area <= 0.001 entropy = 0.781 samples = 95 value = [22, 73]
entropy = 0.0 samples = 20 value = [0, 20]
ApplicantIncome_log <= 0.001 entropy = 0.705 samples = 172 value = [33, 139]
entropy = 0.0 samples = 3 value = [0, 3]
entropy = 0.0 samples = 3 value = [3, 0]
entropy = 0.0 samples = 3 value = [0, 3]
Loan_Amount_Term <= 0.03 entropy = 0.976 samples = 22 value = [13, 9]
Education <= 0.0 entropy = 0.751 samples = 93 value = [20, 73]
entropy = 0.0 samples = 2 value = [2, 0]
entropy = 0.0 samples = 2 value = [2, 0]
Gender <= 0.0 entropy = 0.685 samples = 170 value = [31, 139]
Loan_Amount_Term <= 0.02 entropy = 0.998 samples = 19 value = [10, 9]
entropy = 0.0 samples = 3 value = [3, 0]
Dependents <= 0.0 entropy = 0.792 samples = 84 value = [20, 64]
entropy = 0.0 samples = 9 value = [0, 9]
Property_Area <= 0.0 entropy = 0.714 samples = 158 value = [31, 127]
entropy = 0.0 samples = 12 value = [0, 12]
Property_Area <= 0.0 entropy = 0.918 samples = 15 value = [10, 5]
entropy = 0.0 samples = 4 value = [0, 4]
LoanAmount_log <= 0.001 entropy = 0.727 samples = 74 value = [15, 59]
Dependents <= 0.0 entropy = 1.0 samples = 10 value = [5, 5]
Property_Area <= 0.0 entropy = 0.642 samples = 141 value = [23, 118]
Dependents <= 0.0 entropy = 0.998 samples = 17 value = [8, 9]
Loan_Amount_Term <= 0.015 entropy = 1.0 samples = 8 value = [4, 4]
Loan_Amount_Term <= 0.009 entropy = 0.592 samples = 7 value = [6, 1]
Gender <= 0.0 entropy = 0.786 samples = 64 value = [15, 49]
entropy = 0.0 samples = 10 value = [0, 10]
entropy = 0.0 samples = 3 value = [3, 0]
Property_Area <= 0.0 entropy = 0.863 samples = 7 value = [2, 5]
Loan_Amount_Term <= 0.04 entropy = 0.845 samples = 44 value = [12, 32]
Married <= 0.0 entropy = 0.51 samples = 97 value = [11, 86]
Loan_Amount_Term <= 0.076 entropy = 0.592 samples = 7 value = [1, 6]
Property_Area <= 0.0 entropy = 0.881 samples = 10 value = [7, 3]
Dependents <= 0.0 entropy = 0.811 samples = 4 value = [3, 1]
LoanAmount_log <= 0.0 entropy = 0.811 samples = 4 value = [1, 3]
entropy = 0.0 samples = 1 value = [0, 1]
entropy = 0.0 samples = 6 value = [6, 0]
Loan_Amount_Term <= 0.058 entropy = 0.741 samples = 62 value = [13, 49]
entropy = 0.0 samples = 2 value = [2, 0]
entropy = 0.0 samples = 5 value = [0, 5]
entropy = 0.0 samples = 2 value = [2, 0]
entropy = 0.0 samples = 6 value = [0, 6]
Self_Employed <= 0.0 entropy = 0.9 samples = 38 value = [12, 26]
LoanAmount_log <= 0.001 entropy = 0.706 samples = 52 value = [10, 42]
Dependents <= 0.001 entropy = 0.154 samples = 45 value = [1, 44]
entropy = 0.0 samples = 1 value = [1, 0]
entropy = 0.0 samples = 6 value = [0, 6]
entropy = 0.0 samples = 1 value = [0, 1]
Loan_Amount_Term <= 0.083 entropy = 0.764 samples = 9 value = [7, 2]
entropy = 0.0 samples = 1 value = [0, 1]
entropy = 0.0 samples = 3 value = [3, 0]
entropy = 0.0 samples = 3 value = [0, 3]
entropy = 0.0 samples = 1 value = [1, 0]
LoanAmount_log <= 0.001 entropy = 0.949 samples = 19 value = [7, 12]
Credit_History <= 0.0 entropy = 0.583 samples = 43 value = [6, 37]
Gender <= 0.0 entropy = 0.937 samples = 34 value = [12, 22]
entropy = 0.0 samples = 4 value = [0, 4]
entropy = 0.0 samples = 19 value = [0, 19]
ApplicantIncome_log <= 0.001 entropy = 0.885 samples = 33 value = [10, 23]
entropy = 0.0 samples = 43 value = [0, 43]
Education <= 0.0 entropy = 1.0 samples = 2 value = [1, 1]
entropy = 0.0 samples = 5 value = [5, 0]
Loan_Amount_Term <= 0.092 entropy = 1.0 samples = 4 value = [2, 2]
entropy = 0.0 samples = 4 value = [0, 4]
LoanAmount_log <= 0.001 entropy = 0.997 samples = 15 value = [7, 8]
entropy = 0.0 samples = 13 value = [0, 13]
Property_Area <= 0.001 entropy = 0.722 samples = 30 value = [6, 24]
entropy = 0.0 samples = 2 value = [2, 0]
LoanAmount_log <= 0.001 entropy = 0.896 samples = 32 value = [10, 22]
Property_Area <= 0.0 entropy = 0.722 samples = 5 value = [4, 1]
LoanAmount_log <= 0.001 entropy = 0.75 samples = 28 value = [6, 22]
entropy = 0.0 samples = 1 value = [1, 0]
entropy = 0.0 samples = 1 value = [0, 1]
entropy = 0.0 samples = 2 value = [0, 2]
entropy = 0.0 samples = 2 value = [2, 0]
entropy = 0.0 samples = 3 value = [3, 0]
LoanAmount_log <= 0.001 entropy = 0.918 samples = 12 value = [4, 8]
LoanAmount_log <= 0.001 entropy = 0.828 samples = 23 value = [6, 17]
entropy = 0.0 samples = 7 value = [0, 7]
ApplicantIncome_log <= 0.001 entropy = 0.994 samples = 11 value = [6, 5]
ApplicantIncome_log <= 0.002 entropy = 0.702 samples = 21 value = [4, 17]
Gender <= 0.0 entropy = 1.0 samples = 2 value = [1, 1]
entropy = 0.0 samples = 3 value = [3, 0]
Education <= 0.0 entropy = 0.619 samples = 26 value = [4, 22]
entropy = 0.0 samples = 2 value = [2, 0]
entropy = 0.0 samples = 6 value = [0, 6]
Self_Employed <= 0.0 entropy = 0.918 samples = 6 value = [4, 2]
TotalIncome <= 0.997 entropy = 0.702 samples = 21 value = [4, 17]
entropy = 0.0 samples = 2 value = [2, 0]
LoanAmount_log <= 0.001 entropy = 0.65 samples = 6 value = [1, 5]
entropy = 0.0 samples = 5 value = [5, 0]
entropy = 0.0 samples = 12 value = [0, 12]
LoanAmount_log <= 0.001 entropy = 0.991 samples = 9 value = [4, 5]
entropy = 0.0 samples = 1 value = [1, 0]
entropy = 0.0 samples = 1 value = [0, 1]
LoanAmount_log <= 0.001 entropy = 0.414 samples = 24 value = [2, 22]
entropy = 0.0 samples = 2 value = [2, 0]
Education <= 0.0 entropy = 0.722 samples = 5 value = [4, 1]
entropy = 0.0 samples = 1 value = [0, 1]
Loan_Amount_Term <= 0.119 entropy = 0.503 samples = 18 value = [2, 16]
LoanAmount_log <= 0.001 entropy = 0.918 samples = 3 value = [2, 1]
Married <= 0.0 entropy = 1.0 samples = 2 value = [1, 1]
entropy = 0.0 samples = 4 value = [0, 4]
LoanAmount_log <= 0.001 entropy = 0.985 samples = 7 value = [4, 3]
entropy = 0.0 samples = 2 value = [0, 2]
Loan_Amount_Term <= 0.052 entropy = 0.971 samples = 5 value = [2, 3]
entropy = 0.0 samples = 19 value = [0, 19]
entropy = 0.0 samples = 3 value = [3, 0]
Gender <= 0.0 entropy = 1.0 samples = 2 value = [1, 1]
Loan_Amount_Term <= 0.085 entropy = 0.323 samples = 17 value = [1, 16]
entropy = 0.0 samples = 1 value = [1, 0]
entropy = 0.0 samples = 1 value = [0, 1]
entropy = 0.0 samples = 2 value = [2, 0]
entropy = 0.0 samples = 1 value = [0, 1]
entropy = 0.0 samples = 1 value = [1, 0]
Gender <= 0.0 entropy = 0.811 samples = 4 value = [1, 3]
entropy = 0.0 samples = 3 value = [3, 0]
entropy = 0.0 samples = 3 value = [0, 3]
entropy = 0.0 samples = 2 value = [2, 0]
entropy = 0.0 samples = 1 value = [0, 1]
entropy = 0.0 samples = 1 value = [1, 0]
Loan_Amount_Term <= 0.084 entropy = 0.65 samples = 6 value = [1, 5]
entropy = 0.0 samples = 11 value = [0, 11]
entropy = 0.0 samples = 1 value = [1, 0]
entropy = 0.0 samples = 3 value = [0, 3]
entropy = 0.0 samples = 5 value = [0, 5]
entropy = 0.0 samples = 1 value = [1, 0]

In [108]:
```
columns = list(normalized_data.columns)
columns
```

Out[108]:
```
['Gender',
 'Married',
 'Dependents',
 'Education',
 'Self_Employed',
 'Loan_Amount_Term',
 'Credit_History',
 'Property_Area',
 'LoanAmount_log',
 'ApplicantIncome_log',
 'TotalIncome',
 'Loan_ID',
 'Loan_Status']
```

Q6. Based on the feature importance, select a different set of features to build another decision tree model. You should aim to improve the result of the baseline model.Based on the graph on feature importance, i would be using feature 6, 8 and 9 to build another decison model. These are: Credit_History, LoanAmount_log and ' and 'ApplicantIncome_log.

In [109]:
```
#to select the features we need for our baseline model
features = normalized_data[['Credit_History','LoanAmount_log','ApplicantIncome_log']]

classes = pd.DataFrame(normalized_data['Loan_Status'])
```

In [110]:
```
print('Features:')
print(features.head())

print('Classes:')
print(classes.head())
```

```
Features:
   Credit_History  LoanAmount_log  ApplicantIncome_log
0        0.000161        0.000721             0.001295
1        0.000162        0.000792             0.001308
2        0.000191        0.000925             0.001457
3        0.000088        0.000461             0.000763
4        0.000151        0.000790             0.001279
Classes:
   Loan_Status
0            1
1            1
2            1
3            1
4            0
```

In [111]:
```
x_train, x_test, y_train, y_test = train_test_split(features, classes, test_size= .33,
                                                    random_state = 10)

print(x_train.shape, x_test.shape)
```

```
(368, 3) (182, 3)
```

In [112]:
```
decisionTree = DecisionTreeClassifier(criterion='entropy')
print(decisionTree)
```

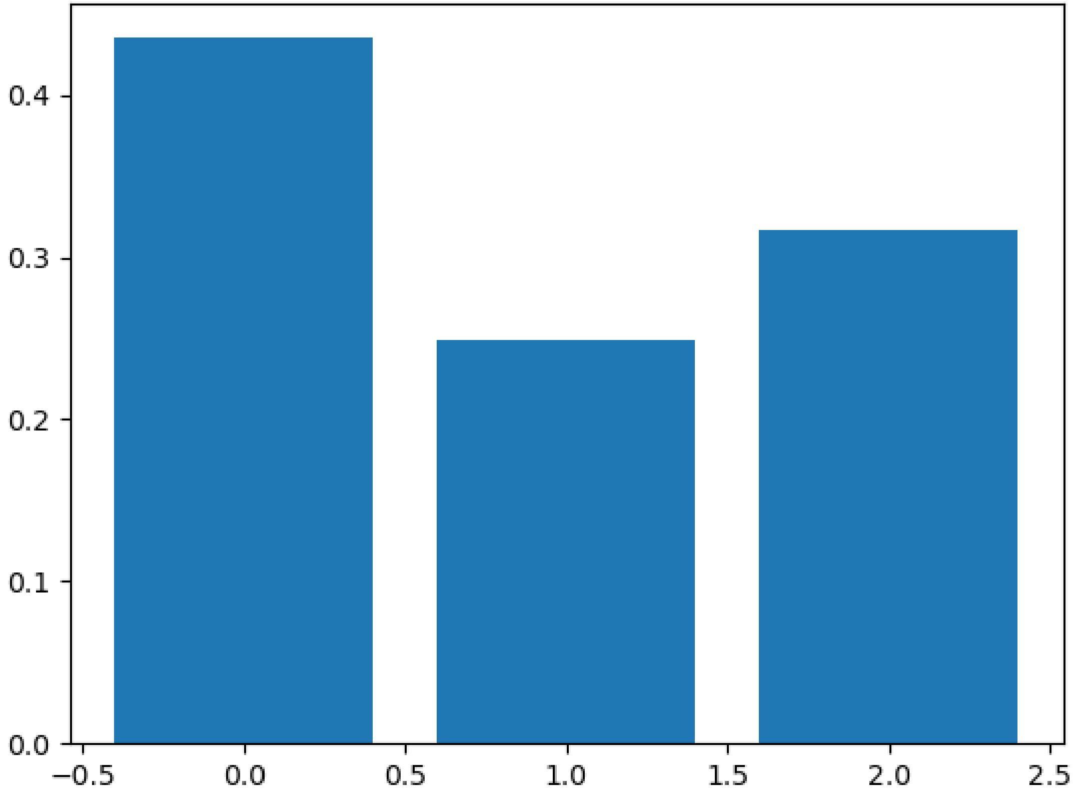
```
DecisionTreeClassifier(criterion='entropy')
```

In [113]:
```
dtc_model = decisionTree.fit(x_train, y_train)
```

In [114]:

```
# 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: 0.43509
Feature: 1, Score: 0.24840
Feature: 2, Score: 0.31651
```

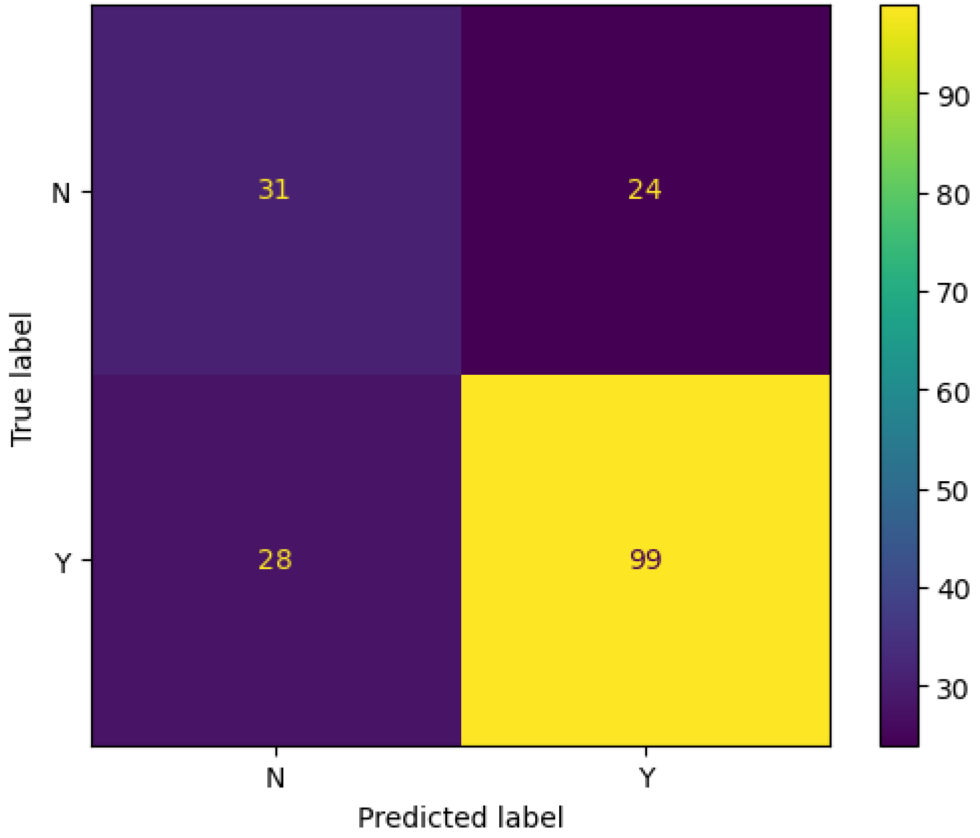


In [115]:
```
prediction = dtc_model.predict(x_test)
```

In [116]:
```
y_true = le.inverse_transform(y_test["Loan_Status"])
y_pred = le.inverse_transform(prediction)
```

In [117]:
```
cm = confusion_matrix(y_true, y_pred)
labels = ['N', 'Y']
ConfusionMatrixDisplay(cm, display_labels=labels).plot()
```

Out[117]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x21d67b9d4f0>



In [118]:
```
print(classification_report(y_true, y_pred))
```

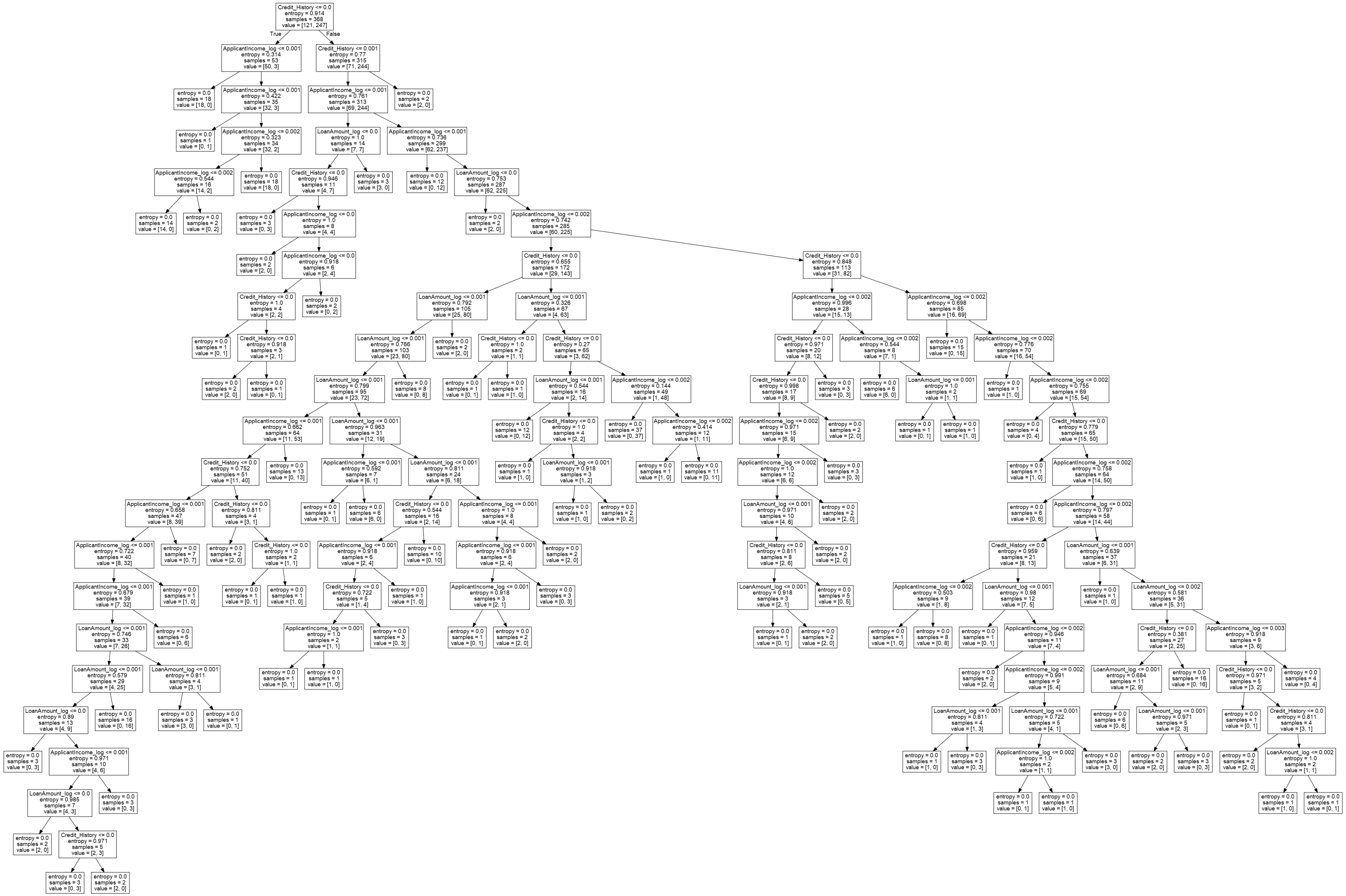
```
              precision    recall  f1-score   support

           N       0.53      0.56      0.54        55
           Y       0.80      0.78      0.79       127

    accuracy                           0.71       182
   macro avg       0.67      0.67      0.67       182
weighted avg       0.72      0.71      0.72       182
```

In [119]:
```
graph = Source( tree.export_graphviz(dtc_model, out_file=None, feature_names=features.columns))
```

In [120]:

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

Out[120]:



Q7. Write a summary (max 250 words) to compare both the models. The summary should include: idea behind selecting those particular features and comparative analysis of the results of both the models.

In the first model, 11 features were used out of 13 features. The loan status was used as the target variable, while the loan *id was dropped because it is a unique identifier.The data was split into test and train size in the ratio 3:10. The code: 'importance = dtc_model.feature_importances*' was used to get the feature_importance after all the necessary libaries had been installed. A bar chart was also plotted to view it easily. From the results,i was able to retrieve the feature importance information from the baseline model. From the model, features 6, 8 and 9 were important for deciding the outcome of the loan. Featurs 6, 8 and 9 are : 'Credit_History','LoanAmount_log','ApplicantIncome_log. This new model shows that Credit_History is the most important feature needed to make a decision on the loan. This is not different from the initial model which also revealed that Credit_History was the most important feature to make a decision on the loan. Both models are quite similar, because they have similar results when determining the order of feature importance. In the new model, The classifier predicted that 99 loans were approved and it was actually approved which makes it true positive The classifier predicted that 31 loans were not approved but it was actually not approved which makes it true negative The classifier predicted that 28 loans were not approved and it was actually approved which makes it false negative The classifier predicted that 24 loans were approved and it was actually not approved which makes it false positive. In the previous model, the classifier predicted that 92 loans were approved and it was actually approved which makes it true positive The classifier predicted that 31 loans were not approved but it was actually not approved which makes it true negative The classifier predicted that 35 loans were not approved and it was actually approved which makes it false negative The classifier predicted that 24 loans were approved and it was actually not approved which makes it false positive. The accuracy of the new result was higher with 0.71 as against 0.67 in the previous model. Precision was 0.80 as against 0.79, recall was 0.78 as against 0.72 and the F1 score in the new model is 0.79 as against 0.76.

With this evaluations, the new model performs better than the previous model, though not very much difference.

```
Q8. Discuss the result based on the evaluation matrix (max 250 words).
An evaluation matrix  helps to understand the model performance, so that a right recommendation can be given for the
analysis. These include Accuracy, Recall, Precision and  F1 score (Raden,2021).
From our confusion matrix, we have the following results:
The classifier predicted that 99 loans were approved and it was actually approved which makes it true positive
The classifier predicted that 31 loans were not approved but it was actually not approved which makes it true
negative
The classifier predicted that 28 loans were not approved and it was actually approved which makes it false negative
The classifier predicted that 24 loans were approved and it was actually not approved which makes it false positive.
Using the evaluation matrices:
```

Accuracy: This model is the most common evaluation matrix used in classification modelling (Raden,2021).It is usually recommended that this is used when we have a balanced data i.e. the number of negative and positive values are not too different from each other. In our model above, we have an accuracy of 0.71, which might be good considering that the output is a binary classification i.e. a YES/NO answer, but if the loan application is a high risk application, an accuracy of 0.71 might not be good enough.
Recall : This is used to measure the fraction of positive patterns that are correctly classified (Raden,2021). It is recommended that this is used when the data is not balanced, i.e.the minority class is positive. The negative class has a recall of 0.56, while the positive class has a recall of 0.78.
Precision : This is used to measure the fraction of positive patterns that are correctly predicted.In the positive class, our model tells us that it has a precison value of 0.80 which means that it corectly predicted 80% of all positive instances as positive, and a precision value of 0.53 for negative, whih menas that for all the times, it predicted negative, the model correctly predited it 53%.
F1 score : This is the harmonic mean of the recall and precision values. This is 0.79 in our model for the positive class and 0.54 for the negative class. It gives us a balanced measure of the 2 metrics.

REFERENCES

Raden A.(2021)Understanding Evaluation Metrics in Classification Modeling [blog entry]. [Accessed 27 February 2023] Available at:https://towardsdatascience.com/understanding-evaluation-metrics-in-classification-modeling-6cc197950f01
Viadinugroho (2021)