Importation of all the necessary libraries using import

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

In [2]:
```
dataset = pd.read_csv("train_ctrUa4K.csv")
```

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

Out[3]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	C
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]:
```
dataset.shape
```

Out[4]: (614, 13)

In [5]:
```
dataset = dataset.sample(n=550, random_state = 71)
```

In [6]:
```
dataset.to_csv('PreciousAdaugoReginald_2325671.csv')
```

In [7]:
```
data = pd.read_csv('PreciousAdaugoReginald_2325671.csv')
```

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

Out[8]:

	Unnamed: 0	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amou
0	611	LP002983	Male	Yes	1	Graduate	No	8072	240.0	253.0	
1	579	LP002888	Male	No	0	Graduate	NaN	3182	2917.0	161.0	
2	612	LP002984	Male	Yes	2	Graduate	No	7583	0.0	187.0	
3	205	LP001692	Female	No	0	Not Graduate	No	4408	0.0	120.0	
4	494	LP002585	Male	Yes	0	Graduate	No	3597	2157.0	119.0	

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	C
0	LP002983	Male	Yes	1	Graduate	No	8072	240.0	253.0	360.0	
1	LP002888	Male	No	0	Graduate	NaN	3182	2917.0	161.0	360.0	
2	LP002984	Male	Yes	2	Graduate	No	7583	0.0	187.0	360.0	
3	LP001692	Female	No	0	Not Graduate	No	4408	0.0	120.0	360.0	
4	LP002585	Male	Yes	0	Graduate	No	3597	2157.0	119.0	360.0	

Q1 ANSWER

1a

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

Out[11]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	550.000000	550.000000	531.000000	539.000000	505.000000
mean	5323.945455	1609.757818	144.389831	340.630798	0.845545
std	6026.785030	2935.906617	84.281249	66.628102	0.361743
min	150.000000	0.000000	9.000000	12.000000	0.000000
25%	2877.500000	0.000000	100.000000	360.000000	1.000000
50%	3815.000000	1170.500000	126.000000	360.000000	1.000000
75%	5706.750000	2305.000000	163.000000	360.000000	1.000000
max	81000.000000	41667.000000	700.000000	480.000000	1.000000

The code data.describe() is used to tell the data to show values such as mean, first quartile, third qartile, minimum and maximmum values that cna be used to make a statistcal analysis

1b

In [12]:
```
data.size
```

Out[12]:
```
7150
```

The code for data.size is used to give the number of elements in the dataframe created by the use of Pandas library. It is the number given by the result of the multiplication between the columns and rows of the dataframe (www.w3schools.com (http://www.w3schools.com), n.d.).

1c

In [13]:
```
data.ndim
```

Out[13]:
```
2
```

The code data.ndim is what produces a number representation of the dimetions of the provided dataframe

1d

In [14]:
```
data.shape
```

Out[14]:
```
(550, 13)
```

The code data.shape is used to tell the number of clumns and rows in the given dataset. In this case(precisely in the new dataset that has my student number) there are 13 columns and 550 rows

Q2 answer

There is a difference in dimentions between the old dataset and the new one. The difference is that the old dataset has 614 rows where as the new one has 550 random rows selcted at random from the original by the code that was written above. bu the columns number is still the same, wgich is 13

Q3 answer

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

Out[15]:
```
Graduate        427
Not Graduate    123
Name: Education, dtype: int64
```

The values that education can take are Graudate who's count is 427 and Non Graduate who's count is 123

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')
```

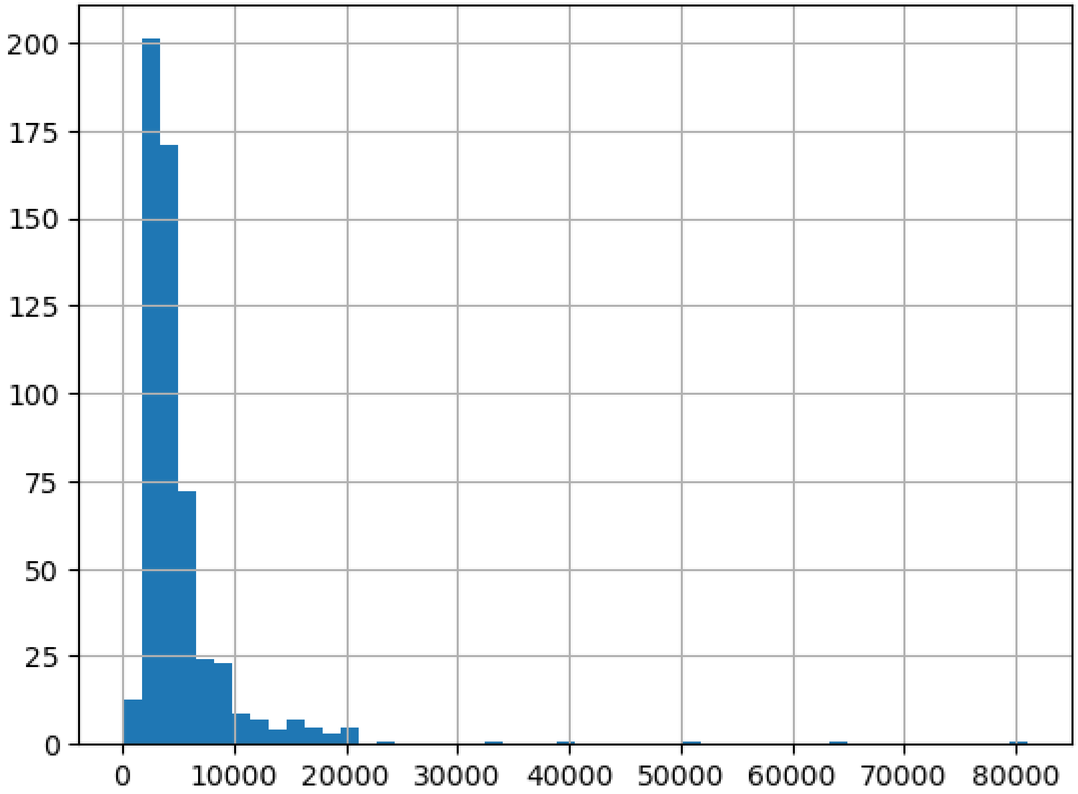
In [17]:
```
data.head()
```

Out[17]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Cı
0	LP002983	Male	Yes	1	Graduate	No	8072	240.0	253.0	360.0	
1	LP002888	Male	No	0	Graduate	NaN	3182	2917.0	161.0	360.0	
2	LP002984	Male	Yes	2	Graduate	No	7583	0.0	187.0	360.0	
3	LP001692	Female	No	0	Not Graduate	No	4408	0.0	120.0	360.0	
4	LP002585	Male	Yes	0	Graduate	No	3597	2157.0	119.0	360.0	

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

Out[18]: <AxesSubplot:>

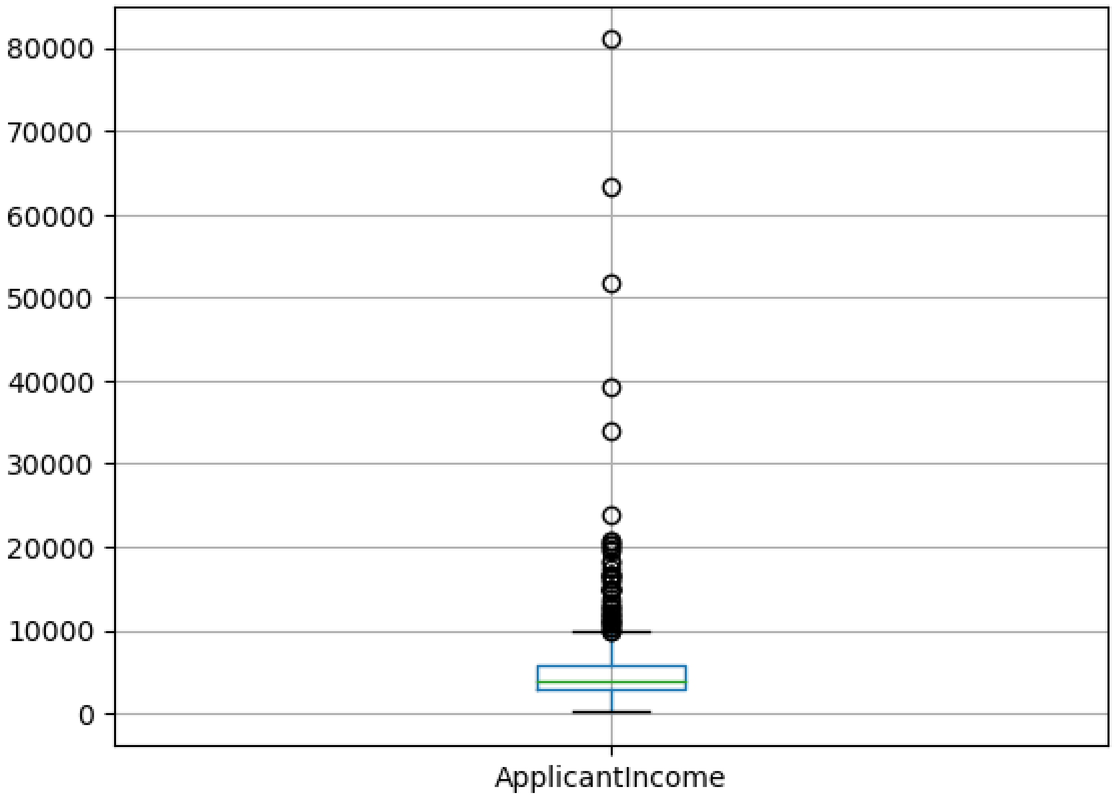


Q4 answers

4a

In [19]:
```
data.boxplot(column='ApplicantIncome')
```

Out[19]: <AxesSubplot:>



The extreme values are the minimum and maximum values on the box plot. Those are 0 (minimum) and 10 000 (maximum). In the dataset the presence of outliers is noticeble as well. There are outlier values such as 40 000, 20 000 and more.

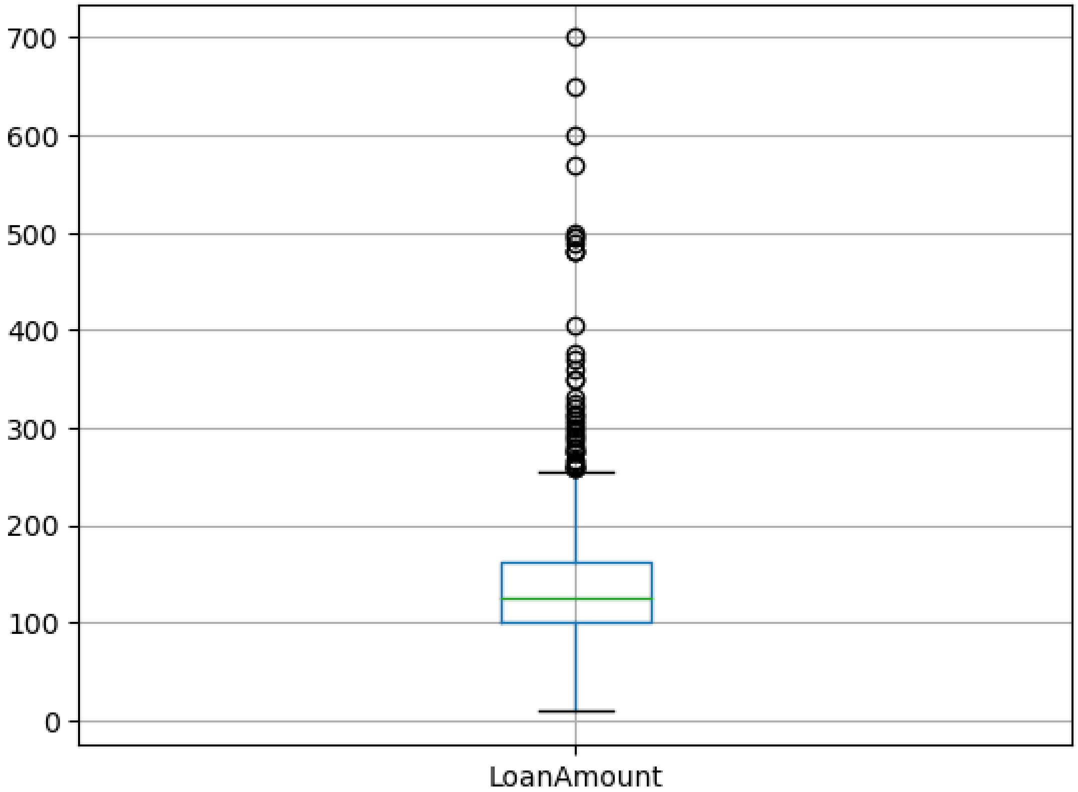
4b

The results in the two plots are not comparable. There are differences in the two plots. The key differences are that the box plot does not show counts where as the histogram does. The histogram does not clearly state what is the mimumum and maximum value, outliers, median, first quartile and third quartile, where as the boxplot shows all these values plainly

Try-it-yourself exercise

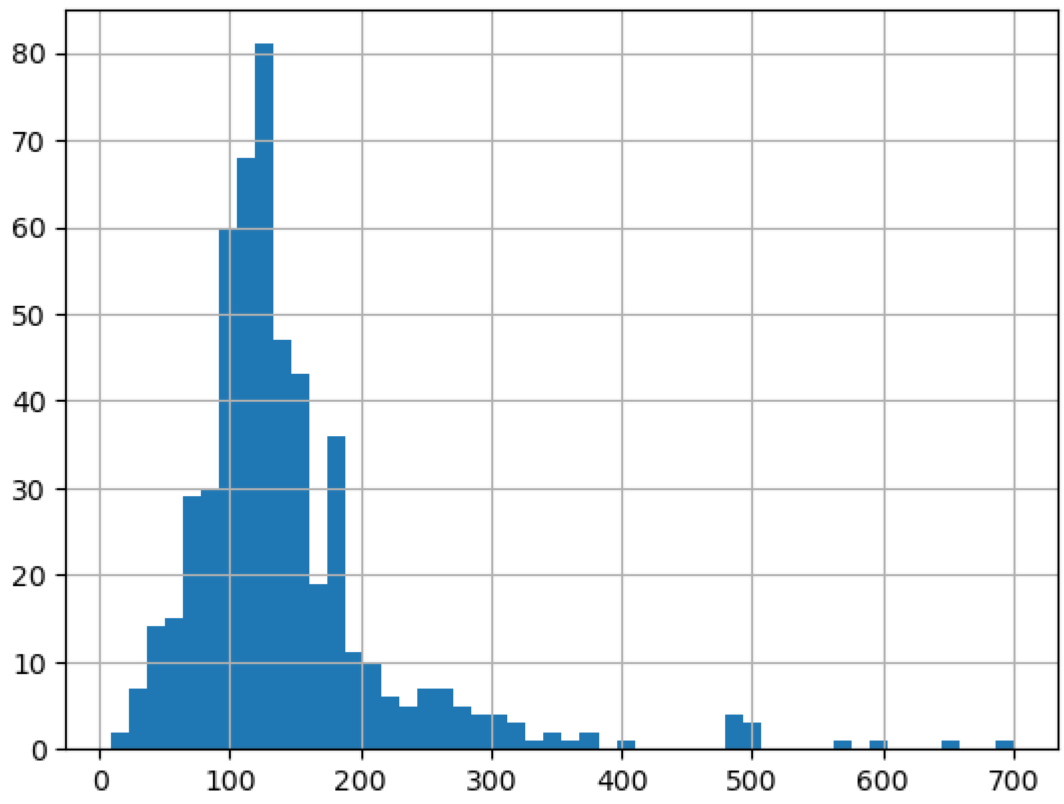
In [20]:
```
data.boxplot(column='LoanAmount')
```

Out[20]: <AxesSubplot:>



In [21]:
```
data['LoanAmount'].hist(bins=50)
```

Out[21]:
```
<AxesSubplot:>
```



Categorical variable analysis

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

Out[22]:
```
1.0    427
0.0     78
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     78
1.0    427
Name: Credit_History, dtype: int64

Probability of getting loan for each Credit History class
                Loan_Status
Credit_History
0.0                0.064103
1.0                0.793911
```

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

Out[24]:
```
Y    377
N    173
Name: Loan_Status, dtype: int64
```

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

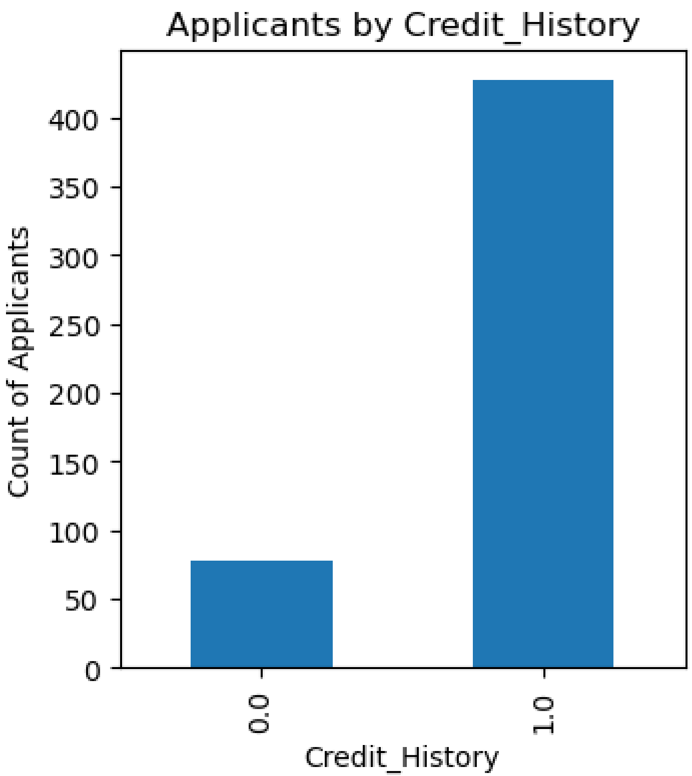
Out[25]:
```
(550, 13)
```

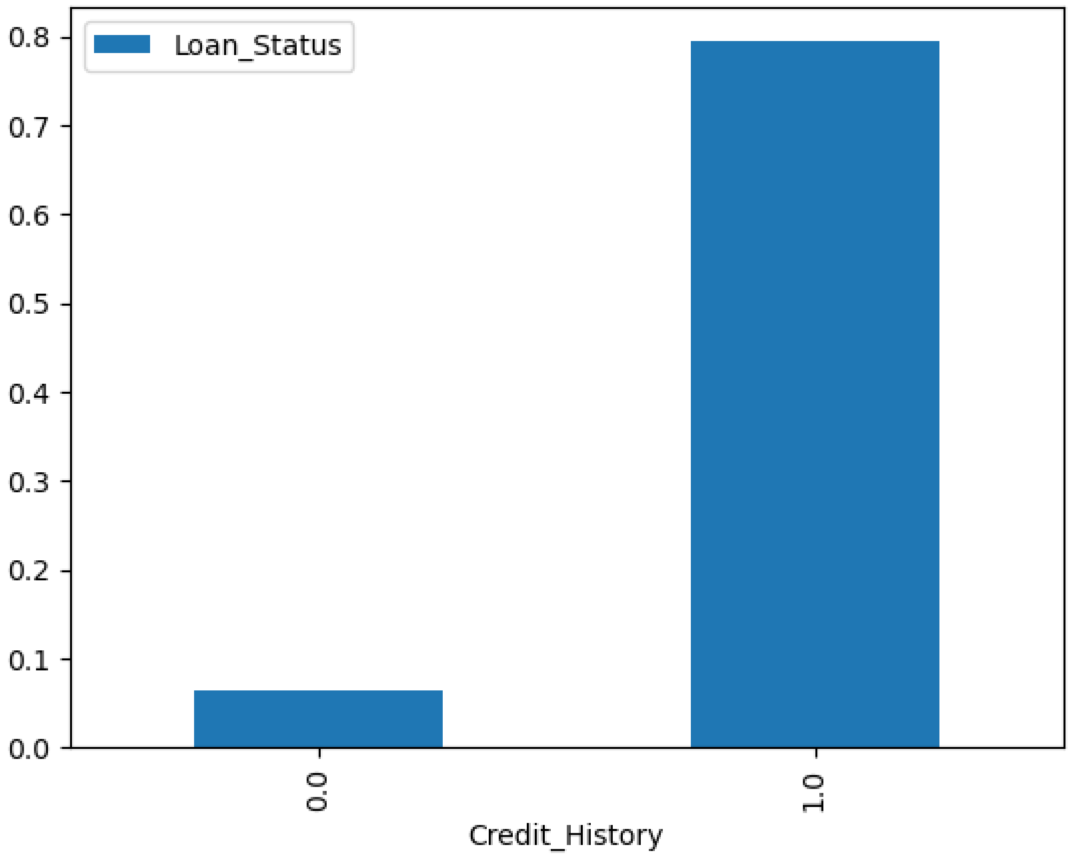
In [26]:
```
data.head()
```

Out[26]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	C
0	LP002983	Male	Yes	1	Graduate	No	8072	240.0	253.0	360.0	
1	LP002888	Male	No	0	Graduate	NaN	3182	2917.0	161.0	360.0	
2	LP002984	Male	Yes	2	Graduate	No	7583	0.0	187.0	360.0	
3	LP001692	Female	No	0	Not Graduate	No	4408	0.0	120.0	360.0	
4	LP002585	Male	Yes	0	Graduate	No	3597	2157.0	119.0	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()
```




Loan_Status
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.0
1.0
Credit_History

Data Pre-processing

Missing values

Outliers and extreme values

Dealing with non-numerical fields

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

Out[28]:
```
Male      435
Female    102
Name: Gender, dtype: int64
```

Filling in missing values by mean

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

Out[29]:
```
Loan_ID              0
Gender              13
Married              2
Dependents          12
Education            0
Self_Employed       27
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount          19
Loan_Amount_Term    11
Credit_History      45
Property_Area        0
Loan_Status          0
dtype: int64
```

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

Out[30]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	C
0	LP002983	Male	Yes	1	Graduate	No	8072	240.0	253.0	360.0	
1	LP002888	Male	No	0	Graduate	NaN	3182	2917.0	161.0	360.0	
2	LP002984	Male	Yes	2	Graduate	No	7583	0.0	187.0	360.0	
3	LP001692	Female	No	0	Not Graduate	No	4408	0.0	120.0	360.0	
4	LP002585	Male	Yes	0	Graduate	No	3597	2157.0	119.0	360.0	

In [31]:
```
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	C
0	LP002983	Male	Yes	1	Graduate	No	8072	240.0	253.0	360.0	
1	LP002888	Male	No	0	Graduate	NaN	3182	2917.0	161.0	360.0	
2	LP002984	Male	Yes	2	Graduate	No	7583	0.0	187.0	360.0	
3	LP001692	Female	No	0	Not Graduate	No	4408	0.0	120.0	360.0	
4	LP002585	Male	Yes	0	Graduate	No	3597	2157.0	119.0	360.0	

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

Out[33]:
```
Loan_ID               0
Gender               13
Married               2
Dependents           12
Education             0
Self_Employed        27
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount            0
Loan_Amount_Term     11
Credit_History       45
Property_Area         0
Loan_Status           0
dtype: int64
```

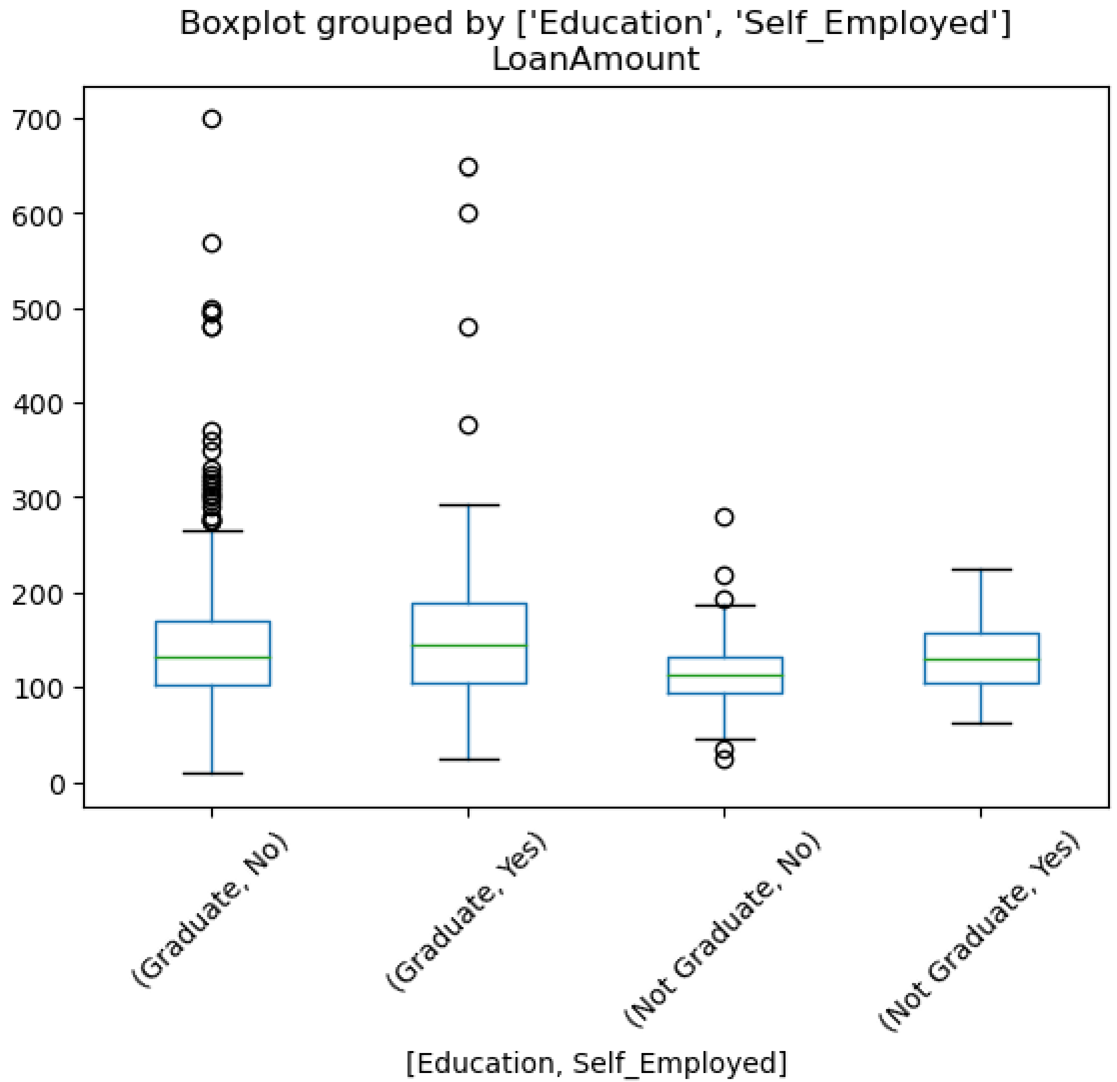
In [34]:
```
data.shape
```

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]'>



Impute the values

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

Out[37]:
```
No     451
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               13
Married               2
Dependents           12
Education             0
Self_Employed         0
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount            0
Loan_Amount_Term     11
Credit_History       45
Property_Area         0
Loan_Status           0
dtype: int64
```

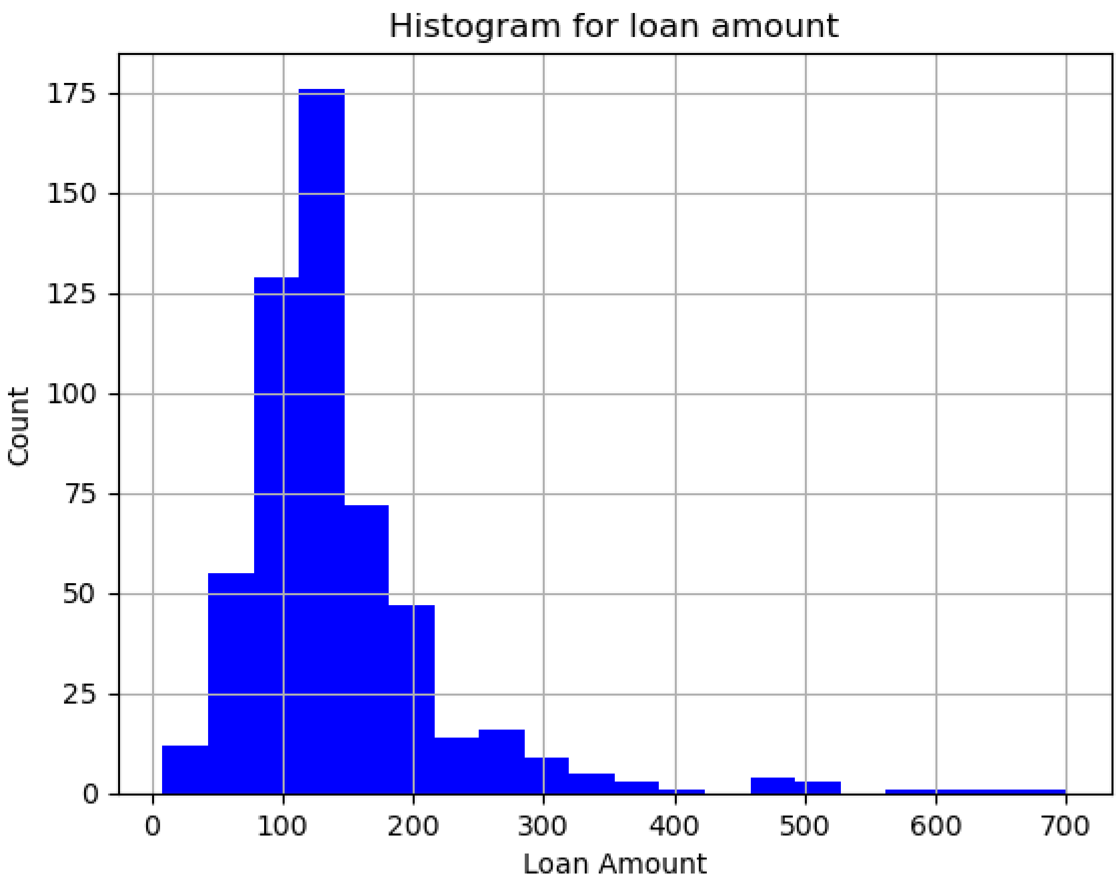
Dealing with outliers

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

Out[41]:

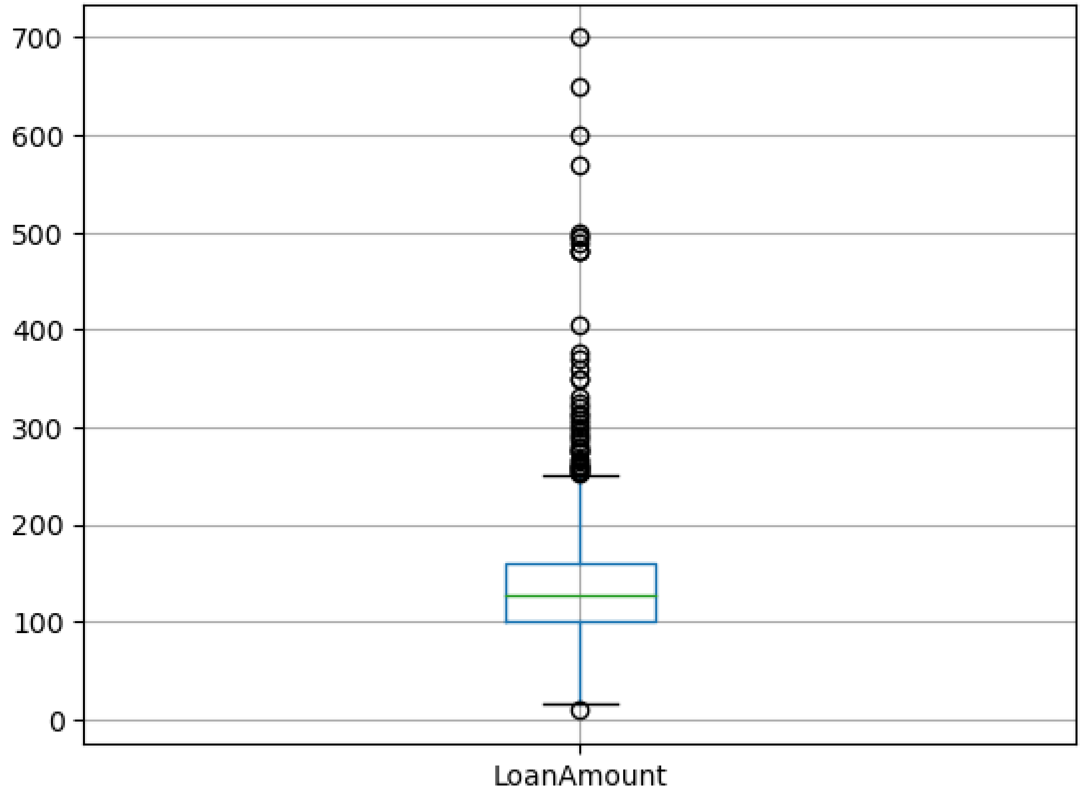
	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	550.000000	550.000000	550.000000	539.000000	505.000000
mean	5323.945455	1609.757818	144.389831	340.630798	0.845545
std	6026.785030	2935.906617	82.809988	66.628102	0.361743
min	150.000000	0.000000	9.000000	12.000000	0.000000
25%	2877.500000	0.000000	100.000000	360.000000	1.000000
50%	3815.000000	1170.500000	128.000000	360.000000	1.000000
75%	5706.750000	2305.000000	160.000000	360.000000	1.000000
max	81000.000000	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()
```



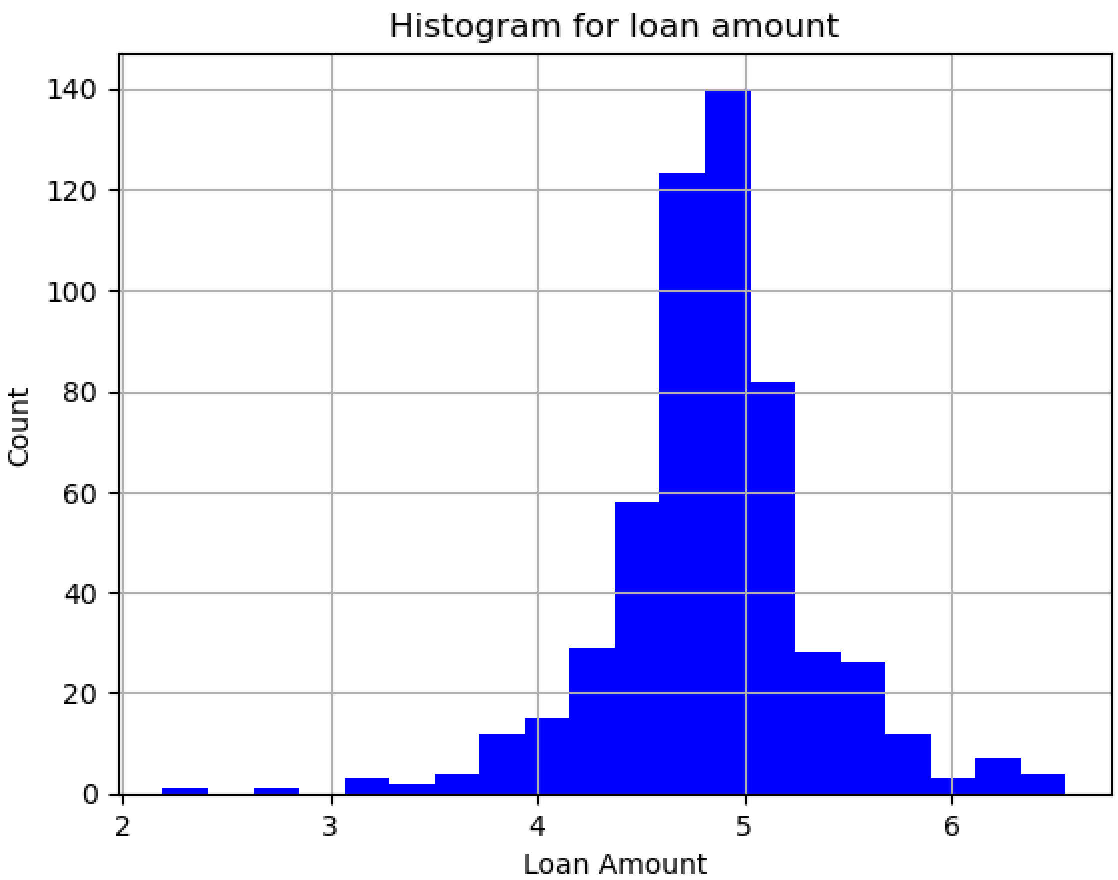
In [43]:
```
data.boxplot(column='LoanAmount')
```

Out[43]: <AxesSubplot:>



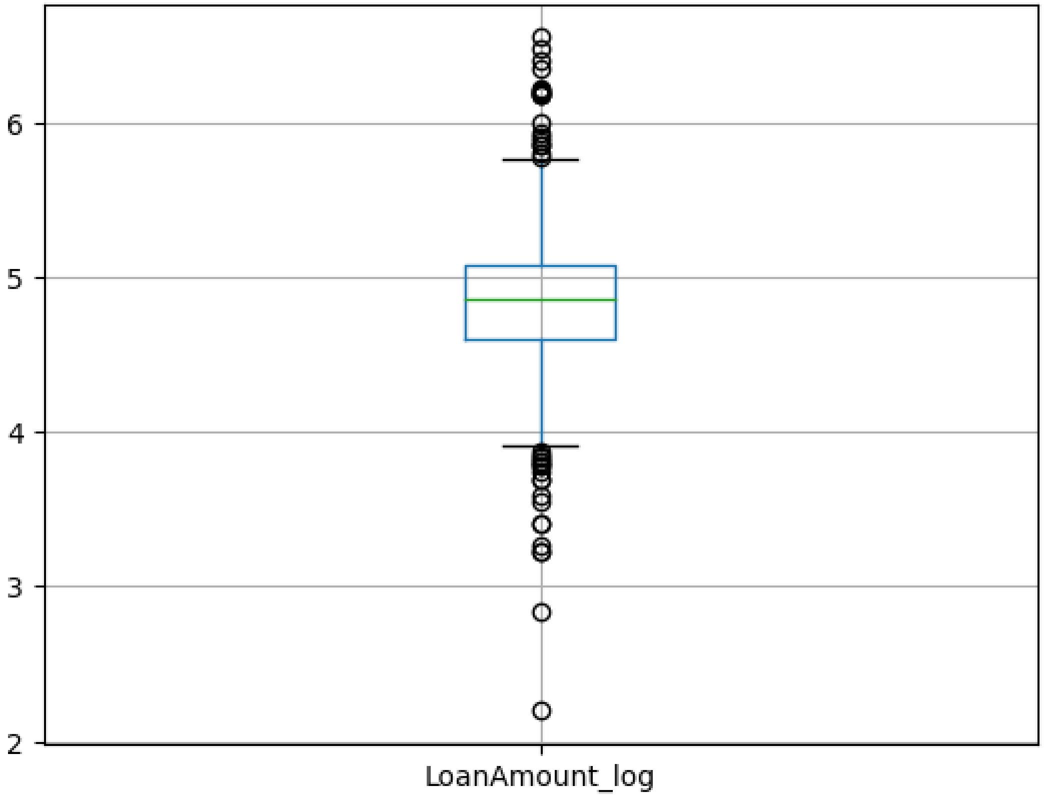
In [44]:
```
data['LoanAmount_log'] = np.log(data['LoanAmount'])
#data['LoanAmount_log'].hist(bins = 20)
```

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.head()
```

Out[47]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	C
0	LP002983	Male	Yes	1	Graduate	No	8072	240.0	253.0	360.0	
1	LP002888	Male	No	0	Graduate	No	3182	2917.0	161.0	360.0	
2	LP002984	Male	Yes	2	Graduate	No	7583	0.0	187.0	360.0	
3	LP001692	Female	No	0	Not Graduate	No	4408	0.0	120.0	360.0	
4	LP002585	Male	Yes	0	Graduate	No	3597	2157.0	119.0	360.0	

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

Out[48]:

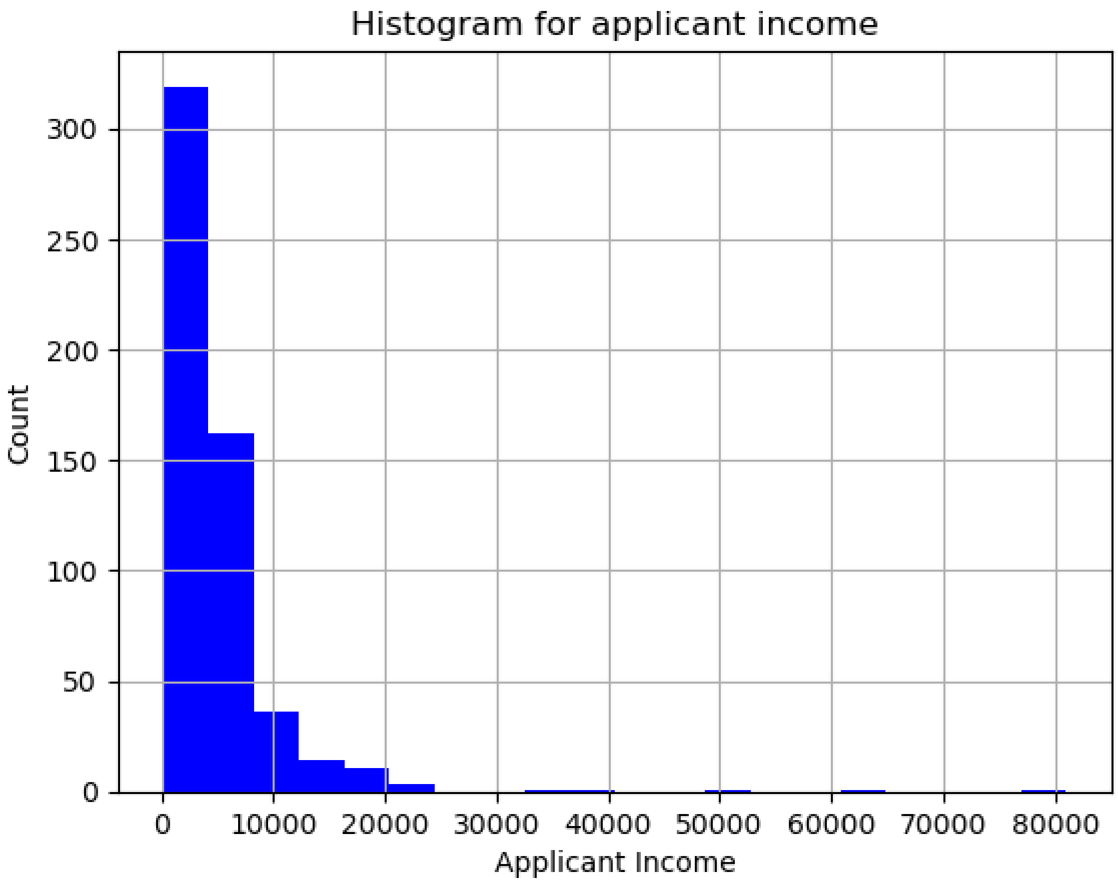
	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	LoanAmount_log
count	550.000000	550.000000	550.000000	539.000000	505.000000	550.000000
mean	5323.945455	1609.757818	144.389831	340.630798	0.845545	4.847673
std	6026.785030	2935.906617	82.809988	66.628102	0.361743	0.499589
min	150.000000	0.000000	9.000000	12.000000	0.000000	2.197225
25%	2877.500000	0.000000	100.000000	360.000000	1.000000	4.605170
50%	3815.000000	1170.500000	128.000000	360.000000	1.000000	4.852030
75%	5706.750000	2305.000000	160.000000	360.000000	1.000000	5.075174
max	81000.000000	41667.000000	700.000000	480.000000	1.000000	6.551080

In [49]:
```
data = data.drop(['LoanAmount'], axis=1)
```

Try-it yourself exercise

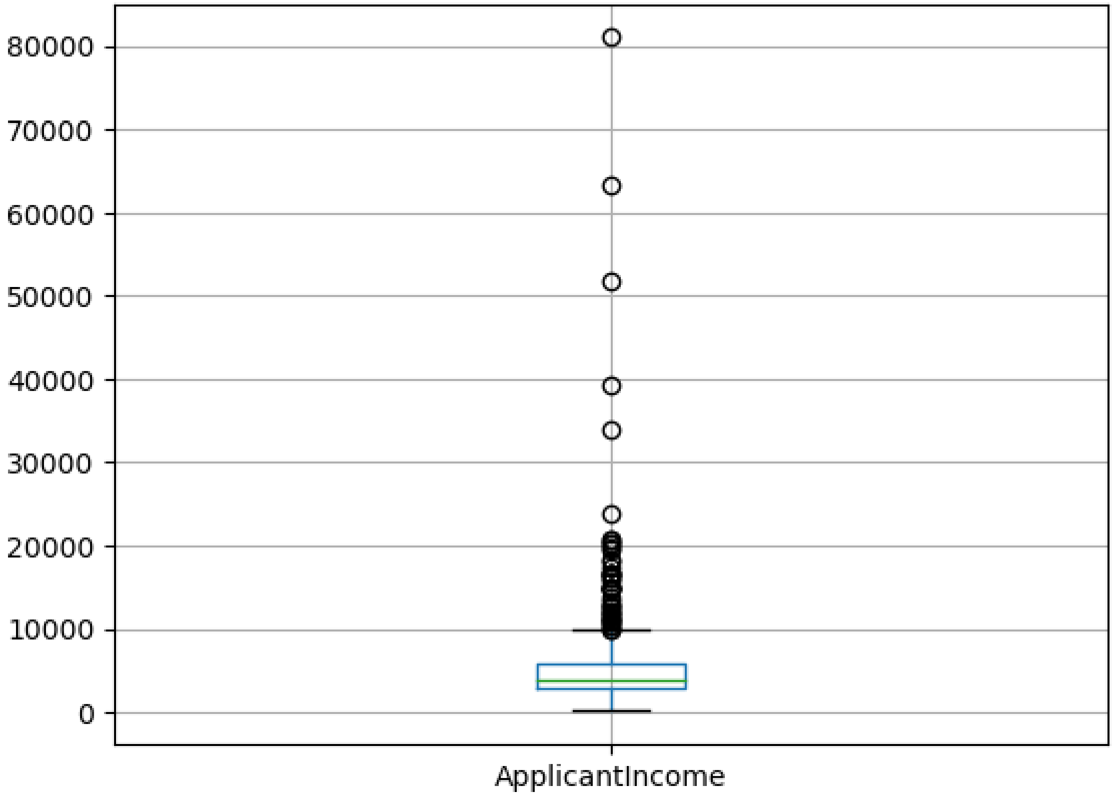
answer 1

In [50]:
```
plt.hist(data['ApplicantIncome'], 20, facecolor='b')
plt.xlabel('Applicant Income')
plt.ylabel('Count')
plt.title('Histogram for applicant income')
plt.grid(True)
plt.show()
```



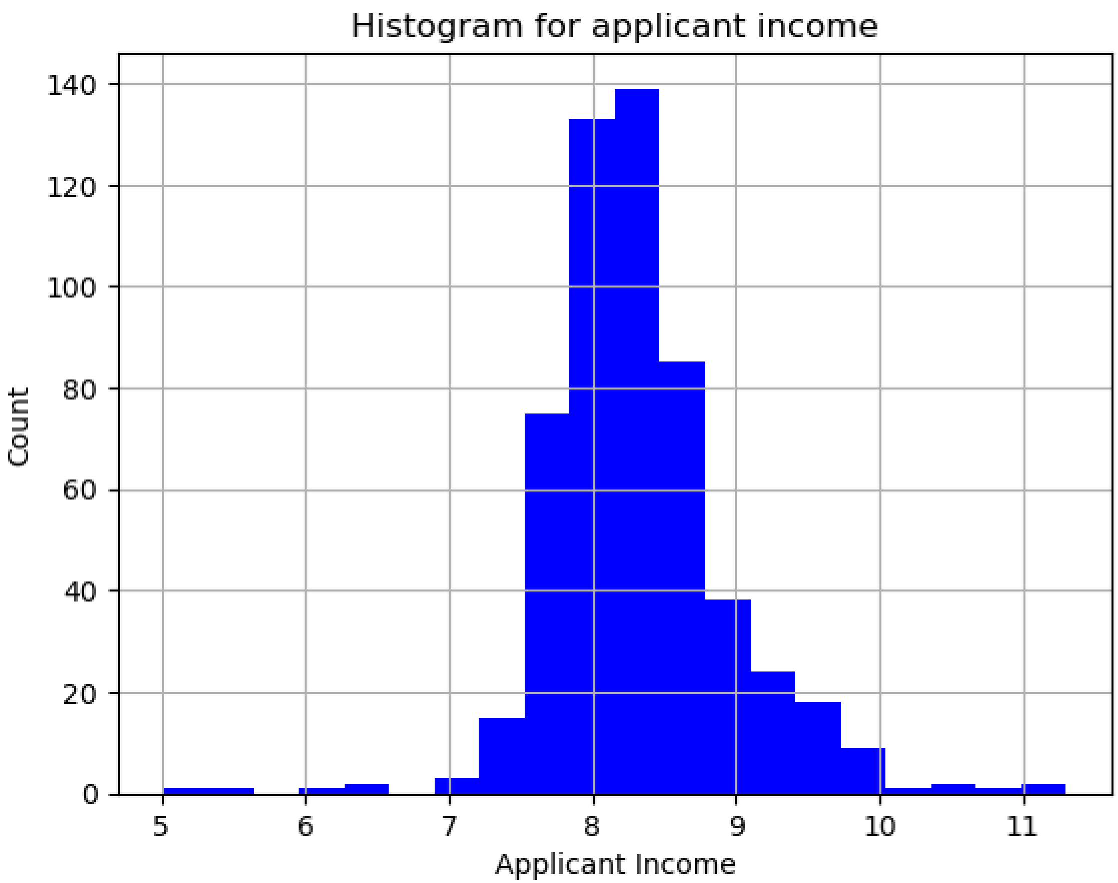
In [51]:
```
data.boxplot(column='ApplicantIncome')
```

Out[51]: <AxesSubplot:>



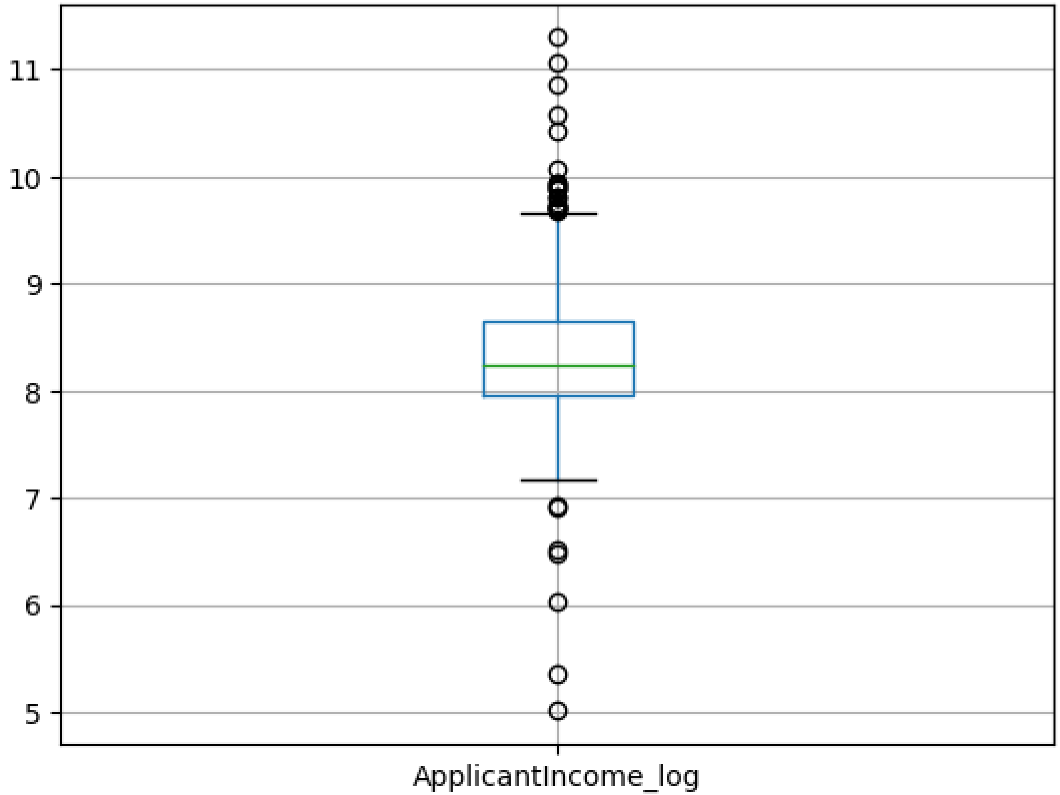
In [52]:
```
data['ApplicantIncome_log'] = np.log(data['ApplicantIncome'])
#data['LoanAmount_log'].hist(bins = 20)
```

In [53]:
```
plt.hist(data['ApplicantIncome_log'], 20, facecolor='b')
plt.xlabel('Applicant Income')
plt.ylabel('Count')
plt.title('Histogram for applicant income')
plt.grid(True)
plt.show()
```



In [54]:
```
data.boxplot(column='ApplicantIncome_log')
```

Out[54]: <AxesSubplot:>



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

Out[55]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	Loan_Amount_Term	Credit_History
0	LP002983	Male	Yes	1	Graduate	No	8072	240.0	360.0	1.0
1	LP002888	Male	No	0	Graduate	No	3182	2917.0	360.0	1.0
2	LP002984	Male	Yes	2	Graduate	No	7583	0.0	360.0	1.0
3	LP001692	Female	No	0	Not Graduate	No	4408	0.0	360.0	1.0
4	LP002585	Male	Yes	0	Graduate	No	3597	2157.0	360.0	0.0

In [56]:
```
data.describe()
```

Out[56]:

	ApplicantIncome	CoapplicantIncome	Loan_Amount_Term	Credit_History	LoanAmount_log	ApplicantIncome_log
count	550.000000	550.000000	539.000000	505.000000	550.000000	550.000000
mean	5323.945455	1609.757818	340.630798	0.845545	4.847673	8.333843
std	6026.785030	2935.906617	66.628102	0.361743	0.499589	0.636549
min	150.000000	0.000000	12.000000	0.000000	2.197225	5.010635
25%	2877.500000	0.000000	360.000000	1.000000	4.605170	7.964677
50%	3815.000000	1170.500000	360.000000	1.000000	4.852030	8.246696
75%	5706.750000	2305.000000	360.000000	1.000000	5.075174	8.649405
max	81000.000000	41667.000000	480.000000	1.000000	6.551080	11.302204

answer 2

In [ ]:

```

```

Missing values continuous

In [57]:

```
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 [58]:

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

Out[58]:

```
Loan_ID              0
Gender               0
Married              0
Dependents           0
Education            0
Self_Employed        0
ApplicantIncome      0
CoapplicantIncome    0
Loan_Amount_Term     0
Credit_History       0
Property_Area        0
Loan_Status          0
LoanAmount_log       0
ApplicantIncome_log  0
dtype: int64
```

Q5 answer

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

Out[59]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	Loan_Amount_Term	Credit_History
0	LP002983	Male	Yes	1	Graduate	No	8072	240.0	360.0	1.0
1	LP002888	Male	No	0	Graduate	No	3182	2917.0	360.0	1.0
2	LP002984	Male	Yes	2	Graduate	No	7583	0.0	360.0	1.0
3	LP001692	Female	No	0	Not Graduate	No	4408	0.0	360.0	1.0
4	LP002585	Male	Yes	0	Graduate	No	3597	2157.0	360.0	0.0

In [60]:
```
data.shape
```

Out[60]:
```
(550, 14)
```

In [61]:
```
from sklearn.preprocessing import LabelEncoder
```

In [62]:
```
columns = list(data)
print(columns)
```

```
['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education', 'Self_Employed', 'ApplicantIncome', 'CoapplicantIncome',
'Loan_Amount_Term', 'Credit_History', 'Property_Area', 'Loan_Status', 'LoanAmount_log', 'ApplicantIncome_log']
```

In [63]:
```
data.dtypes
```

Out[63]:
```
Loan_ID                object
Gender                 object
Married                object
Dependents             object
Education              object
Self_Employed          object
ApplicantIncome         int64
CoapplicantIncome     float64
Loan_Amount_Term      float64
Credit_History        float64
Property_Area          object
Loan_Status            object
LoanAmount_log        float64
ApplicantIncome_log   float64
dtype: object
```

In [64]:
```
columns = list(data.select_dtypes(exclude=['float64','int64']))
```

In [65]:
```
le = LabelEncoder()
for i in columns:
 #print(i)
 data[i] = le.fit_transform(data[i])
```

In [66]:
```
data.head()
```

Out[66]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	Loan_Amount_Term	Credit_History	P
0	547	1	1	1	0	0	8072	240.0	360.0	1.0	
1	516	1	0	0	0	0	3182	2917.0	360.0	1.0	
2	548	1	1	2	0	0	7583	0.0	360.0	1.0	
3	186	0	0	0	1	0	4408	0.0	360.0	1.0	
4	443	1	1	0	0	0	3597	2157.0	360.0	0.0	

Data Normalization

In [67]:
```
#from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import normalize
```

In [68]:
```
original_data = data.copy()
original_data.head()
```

Out[68]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	Loan_Amount_Term	Credit_History	P
0	547	1	1	1	0	0	8072	240.0	360.0	1.0	
1	516	1	0	0	0	0	3182	2917.0	360.0	1.0	
2	548	1	1	2	0	0	7583	0.0	360.0	1.0	
3	186	0	0	0	1	0	4408	0.0	360.0	1.0	
4	443	1	1	0	0	0	3597	2157.0	360.0	0.0	

In [69]:
```
original_data[0:5]
```

Out[69]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	Loan_Amount_Term	Credit_History	P
0	547	1	1	1	0	0	8072	240.0	360.0	1.0	
1	516	1	0	0	0	0	3182	2917.0	360.0	1.0	
2	548	1	1	2	0	0	7583	0.0	360.0	1.0	
3	186	0	0	0	1	0	4408	0.0	360.0	1.0	
4	443	1	1	0	0	0	3597	2157.0	360.0	0.0	

In [70]:
```
data[0:5]
```

Out[70]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	Loan_Amount_Term	Credit_History	P
0	547	1	1	1	0	0	8072	240.0	360.0	1.0	
1	516	1	0	0	0	0	3182	2917.0	360.0	1.0	
2	548	1	1	2	0	0	7583	0.0	360.0	1.0	
3	186	0	0	0	1	0	4408	0.0	360.0	1.0	
4	443	1	1	0	0	0	3597	2157.0	360.0	0.0	

In [71]:
```
data_for_norm = data.drop(['Loan_ID','Loan_Status'], axis=1)
```

Loan status was dropped becuase it is a target class and a binary class whereas Loan ID was dropped as well becuase it is unique number (or unique identifier) for each individual who wants to get approved of the loan

In [72]:
```
normalized_data = normalize( data_for_norm )
```

In [73]:
```
print(normalized_data[0:5])
```

```
[[1.23707340e-04 1.23707340e-04 1.23707340e-04 0.00000000e+00
  0.00000000e+00 9.98565648e-01 2.96897616e-02 4.45346424e-02
  1.23707340e-04 2.47414680e-04 6.84520895e-04 1.11289060e-03]
 [2.30855723e-04 0.00000000e+00 0.00000000e+00 0.00000000e+00
  0.00000000e+00 7.34582911e-01 6.73406144e-01 8.31080603e-02
  2.30855723e-04 4.61711446e-04 1.17307128e-03 1.86191263e-03]
 [1.31725433e-04 1.31725433e-04 2.63450866e-04 0.00000000e+00
  0.00000000e+00 9.98873958e-01 0.00000000e+00 4.74211558e-02
  1.31725433e-04 2.63450866e-04 6.89070047e-04 1.17679078e-03]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 2.26106889e-04
  0.00000000e+00 9.96679167e-01 0.00000000e+00 8.13984801e-02
  2.26106889e-04 2.26106889e-04 1.08248486e-03 1.89730278e-03]
 [2.37552143e-04 2.37552143e-04 0.00000000e+00 0.00000000e+00
  0.00000000e+00 8.54475057e-01 5.12399972e-01 8.55187714e-02
  0.00000000e+00 0.00000000e+00 1.13529103e-03 1.94504260e-03]]
```

In [74]:
```
normalized_data.shape
```

Out[74]:
```
(550, 12)
```

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

Out[75]:
```
(550, 14)
```

In [76]:
```
normalized_data = pd.DataFrame(normalized_data, columns=data_for_norm.columns)
```

In [77]:
```
normalized_data.head()
```

Out[77]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	Loan_Amount_Term	Credit_History	Property_
0	0.000124	0.000124	0.000124	0.000000	0.0	0.998566	0.029690	0.044535	0.000124	0.0(
1	0.000231	0.000000	0.000000	0.000000	0.0	0.734583	0.673406	0.083108	0.000231	0.0(
2	0.000132	0.000132	0.000263	0.000000	0.0	0.998874	0.000000	0.047421	0.000132	0.0(
3	0.000000	0.000000	0.000000	0.000226	0.0	0.996679	0.000000	0.081398	0.000226	0.0(
4	0.000238	0.000238	0.000000	0.000000	0.0	0.854475	0.512400	0.085519	0.000000	0.0(

In [78]:
```
normalized_data['Loan_ID'] = data['Loan_ID']
```

In [79]:
```
normalized_data['Loan_Status'] = data['Loan_Status']
```

In [80]:
```
normalized_data.head()
```

Out[80]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	Loan_Amount_Term	Credit_History	Property_
0	0.000124	0.000124	0.000124	0.000000	0.0	0.998566	0.029690	0.044535	0.000124	0.0(
1	0.000231	0.000000	0.000000	0.000000	0.0	0.734583	0.673406	0.083108	0.000231	0.0(
2	0.000132	0.000132	0.000263	0.000000	0.0	0.998874	0.000000	0.047421	0.000132	0.0(
3	0.000000	0.000000	0.000000	0.000226	0.0	0.996679	0.000000	0.081398	0.000226	0.0(
4	0.000238	0.000238	0.000000	0.000000	0.0	0.854475	0.512400	0.085519	0.000000	0.0(

In [81]:
```
normalized_data.describe()
```

Out[81]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	Loan_Amount_Term	Credit_History
count	550.000000	550.000000	550.000000	550.000000	550.000000	550.000000	550.000000	550.000000	550.000000
mean	0.000182	0.000146	0.000161	0.000060	0.000024	0.877455	0.305482	0.078320	0.000196
std	0.000124	0.000132	0.000250	0.000121	0.000071	0.171294	0.316271	0.038501	0.000123
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.009983	0.000000	0.002207	0.000000
25%	0.000091	0.000000	0.000000	0.000000	0.000000	0.797388	0.000000	0.049333	0.000111
50%	0.000193	0.000154	0.000000	0.000000	0.000000	0.967176	0.244609	0.077040	0.000207
75%	0.000269	0.000250	0.000286	0.000000	0.000000	0.997134	0.591790	0.102802	0.000278
max	0.000673	0.000589	0.001608	0.000673	0.000455	0.999996	0.999941	0.242215	0.000673

Building a Decision Tree classifier using sklearn

Importing all necessary libraries from sklearn

Feature selection

In [82]:
```
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 [83]:
```
columns = list(normalized_data.columns)
columns
```

Out[83]:
```
['Gender',
 'Married',
 'Dependents',
 'Education',
 'Self_Employed',
 'ApplicantIncome',
 'CoapplicantIncome',
 'Loan_Amount_Term',
 'Credit_History',
 'Property_Area',
 'LoanAmount_log',
 'ApplicantIncome_log',
 'Loan_ID',
 'Loan_Status']
```

In [84]:
```
normalized_data.head()
```

Out[84]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	Loan_Amount_Term	Credit_History	Property_
0	0.000124	0.000124	0.000124	0.000000	0.0	0.998566	0.029690	0.044535	0.000124	0.0(
1	0.000231	0.000000	0.000000	0.000000	0.0	0.734583	0.673406	0.083108	0.000231	0.0(
2	0.000132	0.000132	0.000263	0.000000	0.0	0.998874	0.000000	0.047421	0.000132	0.0(
3	0.000000	0.000000	0.000000	0.000226	0.0	0.996679	0.000000	0.081398	0.000226	0.0(
4	0.000238	0.000238	0.000000	0.000000	0.0	0.854475	0.512400	0.085519	0.000000	0.0(

In [85]:
```
features = normalized_data.drop(['Loan_ID','Loan_Status'], axis = 1)
classes = pd.DataFrame(normalized_data['Loan_Status'])
```

In [86]:
```
print('Features:')
print(features.head())
print('Classes:')
print(classes.head())
```

```
Features:
     Gender   Married  Dependents  Education  Self_Employed  ApplicantIncome  \
0  0.000124  0.000124    0.000124   0.000000            0.0         0.998566   
1  0.000231  0.000000    0.000000   0.000000            0.0         0.734583   
2  0.000132  0.000132    0.000263   0.000000            0.0         0.998874   
3  0.000000  0.000000    0.000000   0.000226            0.0         0.996679   
4  0.000238  0.000238    0.000000   0.000000            0.0         0.854475   

   CoapplicantIncome  Loan_Amount_Term  Credit_History  Property_Area  \
0           0.029690          0.044535        0.000124       0.000247   
1           0.673406          0.083108        0.000231       0.000462   
2           0.000000          0.047421        0.000132       0.000263   
3           0.000000          0.081398        0.000226       0.000226   
4           0.512400          0.085519        0.000000       0.000000   

   LoanAmount_log  ApplicantIncome_log  
0        0.000685             0.001113  
1        0.001173             0.001862  
2        0.000689             0.001177  
3        0.001082             0.001897  
4        0.001135             0.001945  
Classes:
   Loan_Status
0            1
1            1
2            1
3            1
4            0
```

In [87]:
```
normalized_data.head(10)
```

Out[87]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	Loan_Amount_Term	Credit_History	Property_
0	0.000124	0.000124	0.000124	0.000000	0.000000	0.998566	0.029690	0.044535	0.000124	0.0(
1	0.000231	0.000000	0.000000	0.000000	0.000000	0.734583	0.673406	0.083108	0.000231	0.0(
2	0.000132	0.000132	0.000263	0.000000	0.000000	0.998874	0.000000	0.047421	0.000132	0.0(
3	0.000000	0.000000	0.000000	0.000226	0.000000	0.996679	0.000000	0.081398	0.000226	0.0(
4	0.000238	0.000238	0.000000	0.000000	0.000000	0.854475	0.512400	0.085519	0.000000	0.0(
5	0.000000	0.000000	0.000000	0.000000	0.000000	0.231332	0.972093	0.038970	0.000000	0.0(
6	0.000143	0.000143	0.000000	0.000000	0.000000	0.304270	0.952238	0.025713	0.000143	0.0(
7	0.000083	0.000000	0.000000	0.000000	0.000000	0.999550	0.000000	0.029986	0.000083	0.0(
8	0.000050	0.000000	0.000000	0.000000	0.000000	0.124015	0.992120	0.017858	0.000050	0.0(
9	0.000213	0.000000	0.000213	0.000000	0.000213	0.997067	0.000000	0.076501	0.000213	0.0(

In [88]:
```
normalized_data.shape
```

Out[88]: (550, 14)

Building our first baseline model using all the features.

Partitioning data into Train and Test sets

In [89]:
```
normalized_data.shape
```

Out[89]: (550, 14)

In [90]:
```
from matplotlib import pyplot
```

In [91]:
```
x_train, x_test, y_train, y_test = train_test_split(features, classes,test_size= .33,
                                                    random_state = 71)

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

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

In [92]:
```
decisionTree = DecisionTreeClassifier(criterion='entropy')
print(decisionTree)
```

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

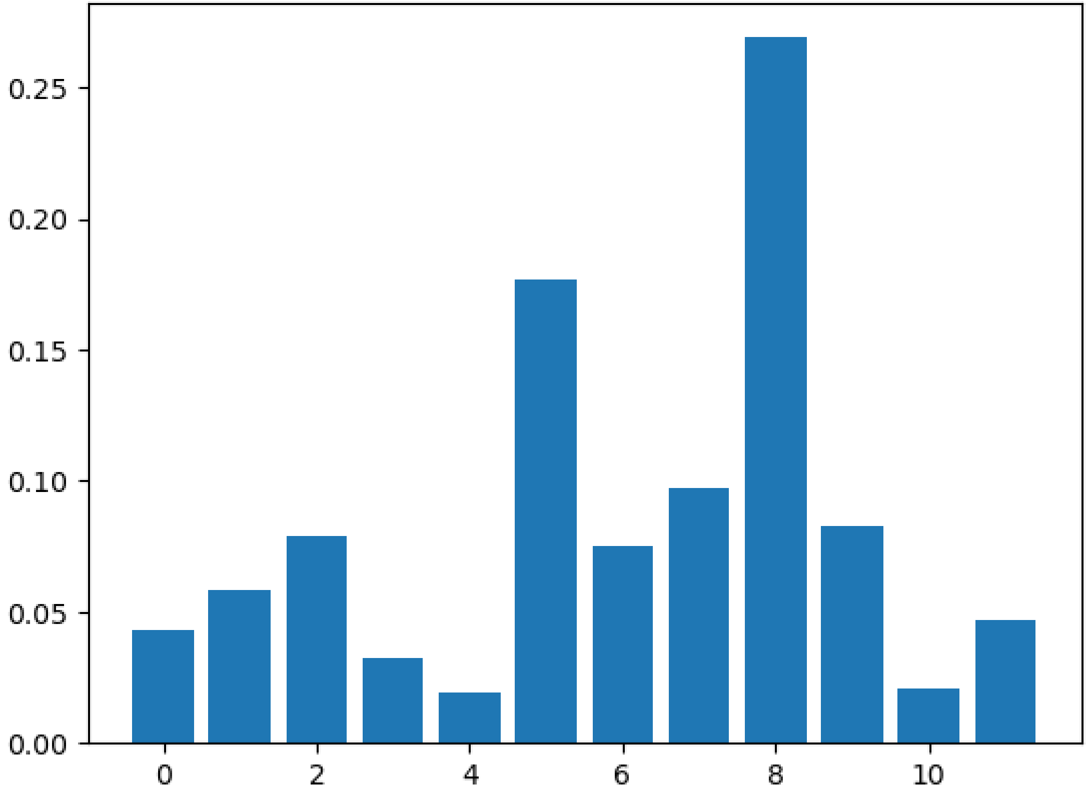
In [93]:
```
dtc_model = decisionTree.fit(x_train, y_train)
```

In [94]:
```
# 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: 0.04306
Feature: 1, Score: 0.05825
Feature: 2, Score: 0.07896
Feature: 3, Score: 0.03209
Feature: 4, Score: 0.01920
Feature: 5, Score: 0.17653
Feature: 6, Score: 0.07492
Feature: 7, Score: 0.09746
Feature: 8, Score: 0.26899
Feature: 9, Score: 0.08302
Feature: 10, Score: 0.02081
Feature: 11, Score: 0.04671
```

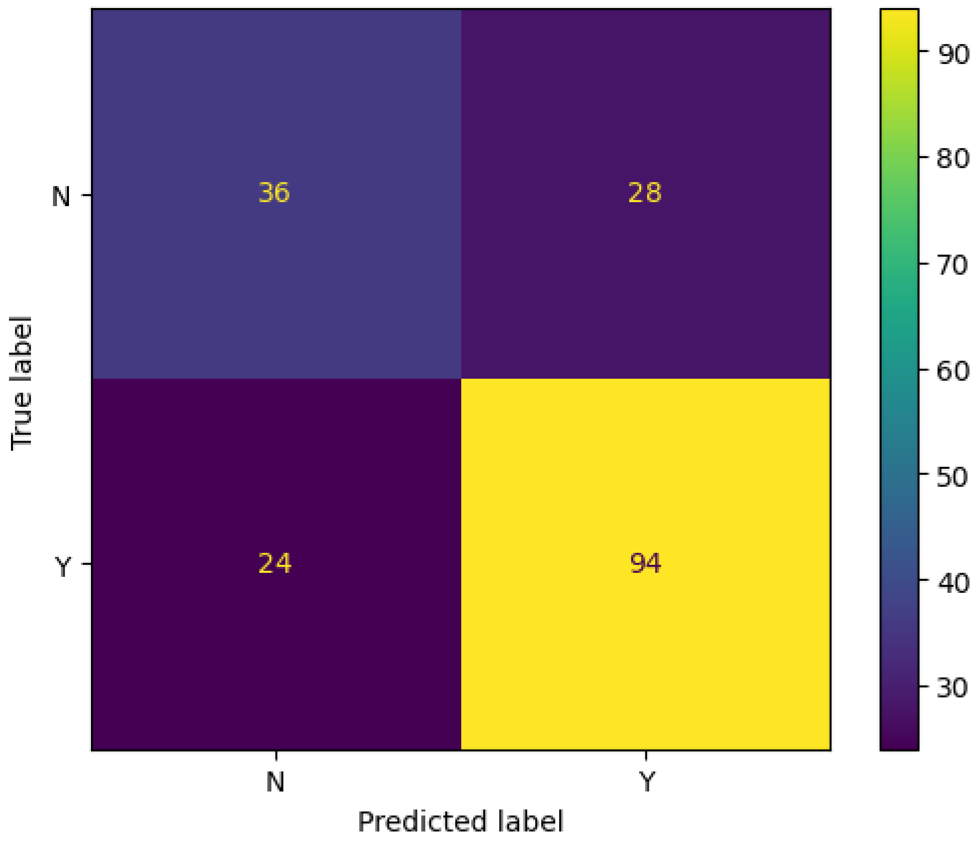


In [95]:
```
prediction = dtc_model.predict(x_test)
```

In [96]:
```
y_true = le.inverse_transform(y_test["Loan_Status"])
y_pred = le.inverse_transform(prediction)
```

In [97]:
```
cm = confusion_matrix(y_true, y_pred)
labels = ['N', 'Y']
ConfusionMatrixDisplay(cm, display_labels=labels).plot()
```

Out[97]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x295f5512040>



In [98]:
```
print(classification_report(y_true, y_pred))
```

```
              precision    recall  f1-score   support

           N       0.60      0.56      0.58        64
           Y       0.77      0.80      0.78       118

    accuracy                           0.71       182
   macro avg       0.69      0.68      0.68       182
weighted avg       0.71      0.71      0.71       182
```

Q6 answer

A new baseline model will be built with the use of only a few features. The features I selected are applicant income, coapplicant income and credit history to predict weather the loan will be approved or not.

In [99]:
```
column1 = list('ApplicantIncome')
column2 = list('CoapplicantIncome')
column3 = list('Credit_History')
column4 = list('Dependents')
```

In [100]:
```
normalized_data.head()
```

Out[100]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	Loan_Amount_Term	Credit_History	Property_
0	0.000124	0.000124	0.000124	0.000000	0.0	0.998566	0.029690	0.044535	0.000124	0.0(
1	0.000231	0.000000	0.000000	0.000000	0.0	0.734583	0.673406	0.083108	0.000231	0.0(
2	0.000132	0.000132	0.000263	0.000000	0.0	0.998874	0.000000	0.047421	0.000132	0.0(
3	0.000000	0.000000	0.000000	0.000226	0.0	0.996679	0.000000	0.081398	0.000226	0.0(
4	0.000238	0.000238	0.000000	0.000000	0.0	0.854475	0.512400	0.085519	0.000000	0.0(

In [101]:
```
features = normalized_data.drop(['Loan_ID','Loan_Status','Gender','Married','Education','Self_Employed',
                                 'Loan_Amount_Term','Property_Area',
  'LoanAmount_log','ApplicantIncome_log'],
  axis = 1)

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

In [102]:
```
print('Features:')
print(features.head())

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

```
Features:
   Dependents  ApplicantIncome  CoapplicantIncome  Credit_History
0    0.000124         0.998566           0.029690        0.000124
1    0.000000         0.734583           0.673406        0.000231
2    0.000263         0.998874           0.000000        0.000132
3    0.000000         0.996679           0.000000        0.000226
4    0.000000         0.854475           0.512400        0.000000
Classes:
   Loan_Status
0            1
1            1
2            1
3            1
4            0
```

In [103]:
```
normalized_data.head(10)
```

Out[103]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	Loan_Amount_Term	Credit_History	Property_
0	0.000124	0.000124	0.000124	0.000000	0.000000	0.998566	0.029690	0.044535	0.000124	0.0C
1	0.000231	0.000000	0.000000	0.000000	0.000000	0.734583	0.673406	0.083108	0.000231	0.0C
2	0.000132	0.000132	0.000263	0.000000	0.000000	0.998874	0.000000	0.047421	0.000132	0.0C
3	0.000000	0.000000	0.000000	0.000226	0.000000	0.996679	0.000000	0.081398	0.000226	0.0C
4	0.000238	0.000238	0.000000	0.000000	0.000000	0.854475	0.512400	0.085519	0.000000	0.0C
5	0.000000	0.000000	0.000000	0.000000	0.000000	0.231332	0.972093	0.038970	0.000000	0.0C
6	0.000143	0.000143	0.000000	0.000000	0.000000	0.304270	0.952238	0.025713	0.000143	0.0C
7	0.000083	0.000000	0.000000	0.000000	0.000000	0.999550	0.000000	0.029986	0.000083	0.0C
8	0.000050	0.000000	0.000000	0.000000	0.000000	0.124015	0.992120	0.017858	0.000050	0.0C
9	0.000213	0.000000	0.000213	0.000000	0.000213	0.997067	0.000000	0.076501	0.000213	0.0C

In [104]:
```
normalized_data.shape
```

Out[104]:
```
(550, 14)
```

A new baseline model will be created with the use of only two features which are ApplicantIncome and CoapplicantIncome

In [105]:
```
from matplotlib import pyplot
```

In [106]:
```
x_train, x_test, y_train, y_test = train_test_split(features, classes, test_size= .33,
                                                    random_state = 71)

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

```
(368, 4) (182, 4)
```

In [107]:
```
decisionTree = DecisionTreeClassifier(criterion='entropy')
print(decisionTree)
```

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

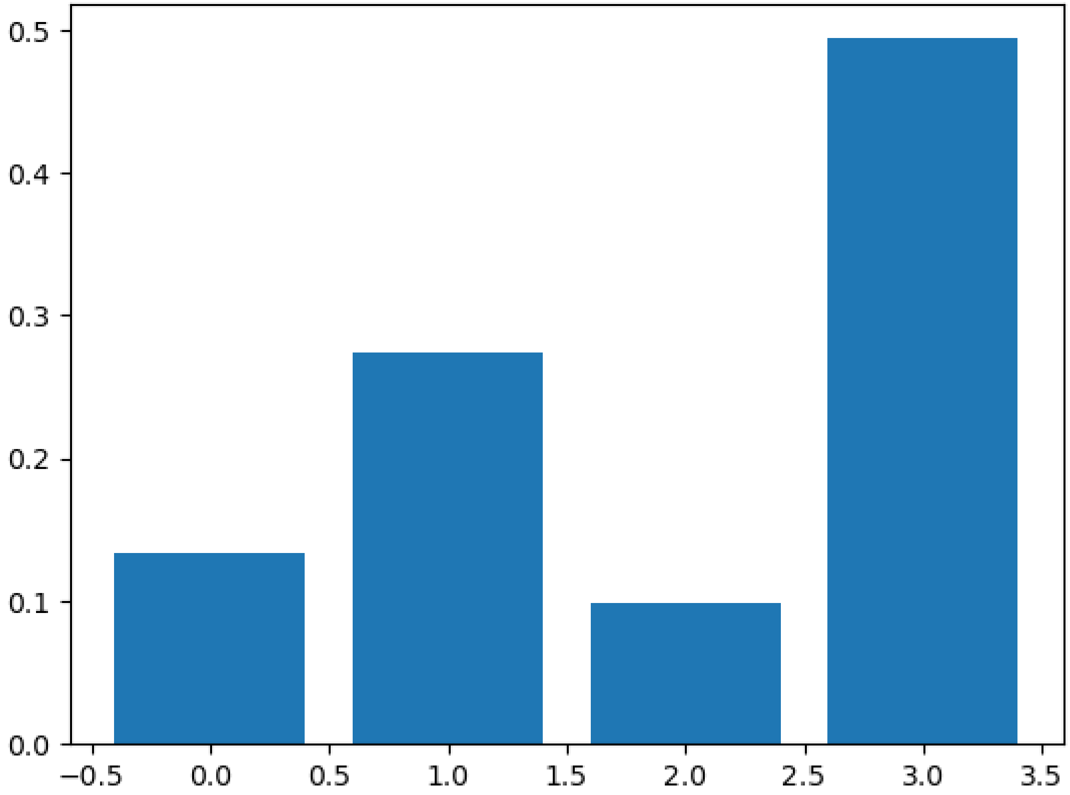
In [108]:
```
dtc_model = decisionTree.fit(x_train, y_train)
```

In [109]:
```
# 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: 0.13400
Feature: 1, Score: 0.27342
Feature: 2, Score: 0.09898
Feature: 3, Score: 0.49360
```



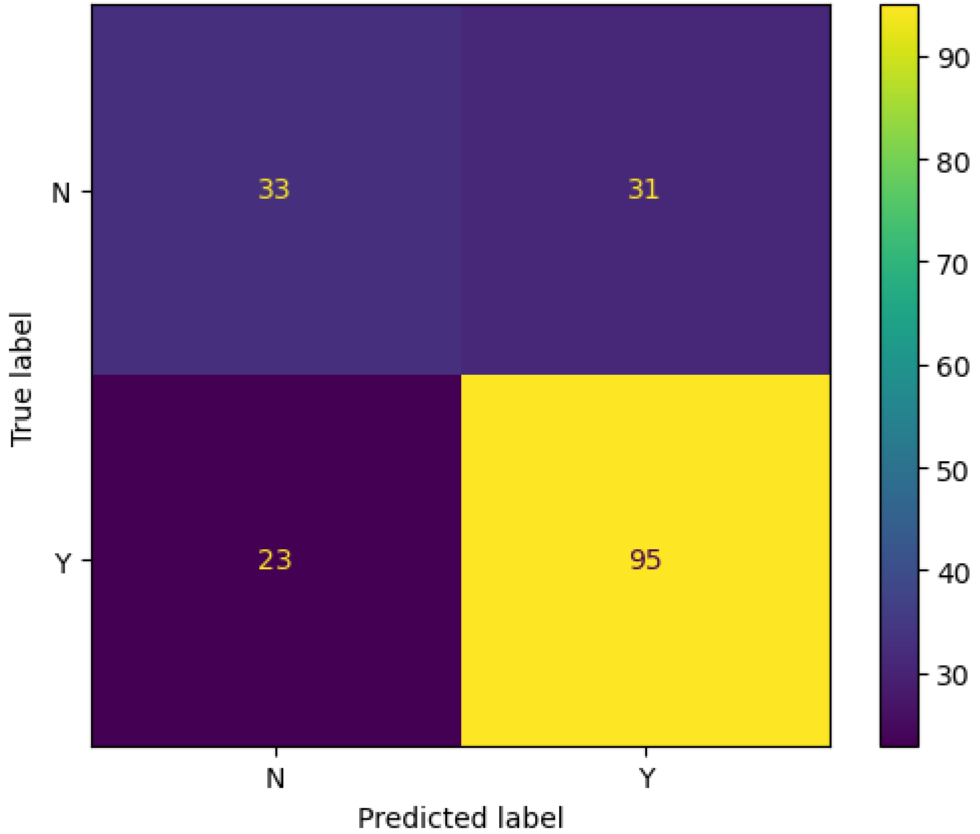
Feature 0 belongs to ApplicantIncome, where as feature 2 belongs to CoapplicantIncome

In [110]:
```
prediction = dtc_model.predict(x_test)
```

In [111]:
```
y_true = le.inverse_transform(y_test["Loan_Status"])
y_pred = le.inverse_transform(prediction)
```

In [112]:
```
cm = confusion_matrix(y_true, y_pred)
labels = ['N', 'Y']
ConfusionMatrixDisplay(cm, display_labels=labels).plot()
```

Out[112]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x295f5680cd0>
```



In [113]:
```
print(classification_report(y_true, y_pred))
```

```
              precision    recall  f1-score   support

           N       0.59      0.52      0.55        64
           Y       0.75      0.81      0.78       118

    accuracy                           0.70       182
   macro avg       0.67      0.66      0.66       182
weighted avg       0.70      0.70      0.70       182
```

Q7 answer

For this workshop two different decisione three models where created using a different set of features in the first one in order to predict if an individual we will get a loan or not all the features were used to create a decision tree model whereas for the second part only the futures such as applicant income Co applicant income, credit hisory and dependants were used to predict if a person would be eligible for a loan. The reason why those particular features were chosen is because the process of classification would be more accurate during the machine learning process. In addition when selecting the most important values the prediction power of the algorithm is increased. So for comparison purposes codes were written for the first model that would include all the features and then codes are written that would include only the four features features. This means that for the models there would be 11 scores and 4 scores respectively. This will enable then after splitting the data in to train and test sets to create a confusion matrix in order to be able to find values such as accuracy precision recall and F1 score. When it comes to accuracy for boht models, it is not a good indictor when it comes to classification even tho they represent the same value of 0.73. The precisoion, recall and F1 score are different for N (no loan) and Y(loan approved).

Q8 answer

The confusion matrix is defined as a table that helps understand the performance of the classification algorithm therefore they are values such as pridicted and true that will help understand and predict an outcome. In this case the accuracy values tell us about how correct the model is but unfortunately is not a good variable for outcome prediction. By comparing both the precision and recall values of each model it can be seen that when it comes to predictinng that an individula will get a loan (Y), the first model is the best due to its precison of 0.78 compared to the second one of 0.76. Where as when it comes to predicting that an individual is not likely to get a loan (N), the second model is the best due to its precison of 0.65 compared to the 0.61 of the first model. At the end the goal was to produce a model that has better results but based on the outcome of the second confusion matrix, it can be concluded that to get better or more accurate results more feautures should have bnne selected.

References

[www.w3schools.com (http://www.w3schools.com)](http://www.w3schools.com). (n.d.). Pandas DataFrame size Property. [online] Available at: [https://www.w3schools.com/python/pandas/ref_df_size.asp#:~:text=Definition%20and%20Usage (https://www.w3schools.com/python/pandas/ref_df_size.asp#:~:text=Definition%20and%20Usage)](https://www.w3schools.com/python/pandas/ref_df_size.asp#:~:text=Definition%20and%20Usage) [Accessed 2 Mar. 2023].