Workshop 02 - Predicting loan case using Decision Tree

By completing this notebook, you will be able to:

- Practice Python programming skills.
- Apply data pre-processing and transformation methods.
  - Dealing with NULL values
  - Dealing with extreme values and outliers
  - Encoding
  - Normalisation
- Perform data analysis.
- Develop a data mining and informatics solution for predicting loan

In this notebook, we will be using **Pandas** to read the dataset and to perform data analysis. We will also be using **matplotlib** for data visualisation. The notebook will further expand your understanding of **data pre-processing**, by implementing some of the key pre-processing steps. We will be using **sklearn** for using data mining and machine learning (ML) algorithms. The scikit-learn or sklearn is an open-source Machine Learning library available in Python for building effective and efficient models. It is built on NumPy, SciPy, and matplotlib. We will also be using **NumPy** library, which is for numeric calculations. NumPy is n-dimensional array, and it is used for Numerical Python, including basic linear algebra functions, Fourier transforms, advanced random number capabilities.

In this notebook, we will build a decision tree model to predict whether an applicant is eligible for the loan or not based on the given applicant's information? To achieve this, we will first perform some data analysis and data pre-processing, which includes **dealing with missing values and outliers values** that appear in the dataset. We will then use sklearn to build **decision tree classifier**.

To run the notebook, restart the Kernel by selecting Restart & Clear Output. Then run each cell one at a time. If it doesn't start, you may have to set the Kernel to Python 3. For this, click Kernel, select Change kernel and select Python 3.

Notebook submission: This notebook is a part of your assessment; please complete the notebook, write appropriate code or description to answer questions provided throughout the notebook and the tasks towards the end of the notebook. Please note that Try-it-yourself includes marks. Save and submit the completed notebook in a readable pdf format.

Dataset: Loan - https://datahack.analyticsvidhya.com/contest/practice-problem-loan-prediction-iii/#ProblemStatement (https://datahack.analyticsvidhya.com/contest/practice-problem-loan-prediction-iii/#ProblemStatement)

- Click on the above link and register yourself using your email id.
- Please read Problem Statement and Data sections carefully to familiarise yourself with the problem you are going to work on.
- Download the dataset by clicking on Train File.
- Move the Train File from the download folder to your working folder where your working ipynb resides. Or, set the path to the Train File in the code appropriately.

Dr. Vinita Nahar, University of Wolverhampton, UK.

Importing 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_Amoun
0	LP001002	Male	No	0	Graduate	No	5849	0.0	NaN	
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	

In [4]:

```
dataset.shape
```

Out[4]:

```
(614, 13)
```

For this task, you are required to generate your own version of dataset by randomly selected 550 rows. To achieve this, replace 48 in ramdom_state with the last two digits of your student number. i.e.'48' -> 'last two digits of your student number'. Failing to do so may result in '0' or reduced grades for this task.

In [5]:

```
dataset = dataset.sample(n=550, random_state = 48)
```

In [6]:

```
dataset.to_csv('VinitaNahar_2448.csv')
```

After randomising the dataset, we simply save it in a csv file using the code below. Name the file as 'YourName_YourStudentNumber.csv'. You are required to submit your dataset ('VinitaNahar_2448.csv') with the notebook. Pandas will add an additional column called 'Unnamed: 0'. This is the index of the original dataset. As we have our own version of the dataset to work on, we drop 'Unnamed: 0' column using drop command.

In [7]:

```
data = pd.read_csv('VinitaNahar_2448.csv')
```

In [8]:

```
data.head()
```

Out[8]:

	Unnamed: 0	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	L
0	0	LP001002	Male	No	0	Graduate	No	5849	0.0	NaN	
1	97	LP001333	Male	Yes	0	Graduate	No	1977	997.0	50.0	
2	260	LP001865	Male	Yes	1	Graduate	No	6083	4250.0	330.0	
3	171	LP001585	NaN	Yes	3+	Graduate	No	51763	0.0	700.0	
4	522	LP002692	Male	Yes	3+	Graduate	Yes	5677	1424.0	100.0	

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_Amoun
0	LP001002	Male	No	0	Graduate	No	5849	0.0	NaN	
1	LP001333	Male	Yes	0	Graduate	No	1977	997.0	50.0	
2	LP001865	Male	Yes	1	Graduate	No	6083	4250.0	330.0	
3	LP001585	NaN	Yes	3+	Graduate	No	51763	0.0	700.0	
4	LP002692	Male	Yes	3+	Graduate	Yes	5677	1424.0	100.0	

Q1. Use and explain the following DataFrame functions/properties on your data.

- describe()
- size
- ndim
- shape

Hint: use print() to see what are the outputs.

In [ ]:

```

```

Q2. Is there any difference between dimensions of the original dataset and the new dataset? If yes, what is the difference?

In [ ]:

```

```

Q3. What are the possible values 'Education' can take? Write code to display all the possible values of 'Education'.

In [ ]:

```

```

Data Analysis

In [11]:

```
columns = data.columns
columns
```

Out[11]:

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

In [12]:

```
data.head()
```

Out[12]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amoun
0	LP001002	Male	No	0	Graduate	No	5849	0.0	NaN	
1	LP001333	Male	Yes	0	Graduate	No	1977	997.0	50.0	
2	LP001865	Male	Yes	1	Graduate	No	6083	4250.0	330.0	
3	LP001585	NaN	Yes	3+	Graduate	No	51763	0.0	700.0	
4	LP002692	Male	Yes	3+	Graduate	Yes	5677	1424.0	100.0	

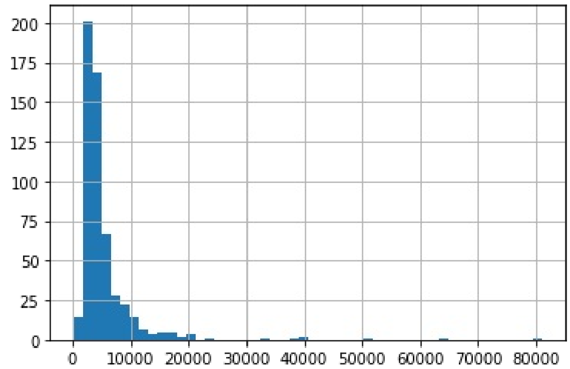
For judging loan approval case, 'ApplicantIncome' and 'LoanAmount' look like important attributes. Let's look at these attributes. You will note that these are numeric variables.

In [13]:

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

Out[13]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1d043a1abc8>
```



Q4. Use boxplot and histogram on 'ApplicantIncome' to visualise its distribution.

Histogram and boxplot are used on the same feature to visualise the data distribution. Compare both the plots and report:

4a. What are the extreme values? Are there any outliers(s) exist in this dataset? Explain with example based on the 'ApplicantIncome'?
4b. Are the results of both the plots comparable? Are there any differences in the two plots? What are the key differences?

In [ ]:

```

```

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

In [14]:

```
#Box plot
```

In [15]:

```
#Histogram
```

Categorical variable analysis

In this section we will create a pivot table from dataframe, which is similar to the pivot table in excel. A Pivot Table is an effective way of analysing and summarising data using aggregate functions such as sum, mean and count. You can then compare, see patterns and trends in the data.

If you are not sure what is pivote table. Please see [pivote table (https://en.wikipedia.org/wiki/Pivot_table)](https://en.wikipedia.org/wiki/Pivot_table) and simple working [example (http://www.datasciencemadesimple.com/create-pivot-table-pandas-python/)](http://www.datasciencemadesimple.com/create-pivot-table-pandas-python/) of pivote table.

In [16]:

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

Out[16]:

```
1.0    423
0.0     81
Name: Credit_History, dtype: int64
```

In [17]:

```
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     81
1.0    423
Name: Credit_History, dtype: int64

Probability of getting loan for each Credit History class:
                Loan_Status
Credit_History
0.0                0.074074
1.0                0.782506
```

In [18]:

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

Out[18]:

```
Y    371
N    179
Name: Loan_Status, dtype: int64
```

In [19]:

```
data.shape
```

Out[19]:

```
(550, 13)
```

We have created pivote table loan_probability by taking mean of Loan_status. We used print() to print the table. Next, create bar graphs to visualise both.

In [20]:

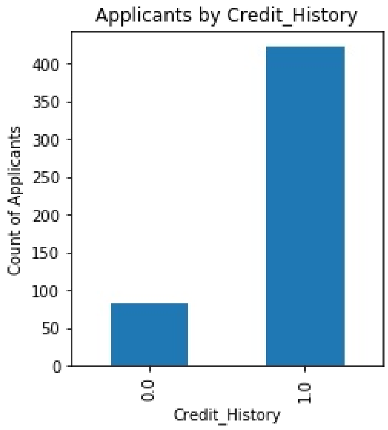
```
data.head()
```

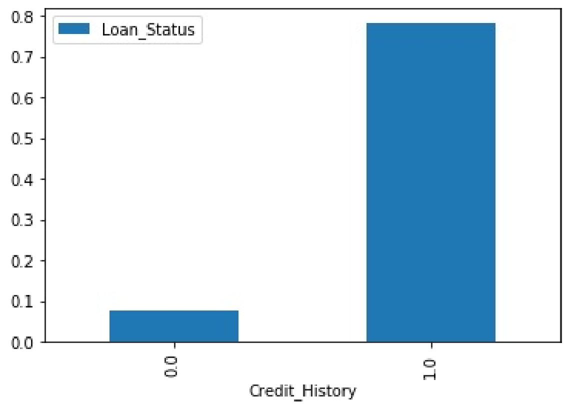
Out[20]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_
0	LP001002	Male	No	0	Graduate	No	5849	0.0	NaN	
1	LP001333	Male	Yes	0	Graduate	No	1977	997.0	50.0	
2	LP001865	Male	Yes	1	Graduate	No	6083	4250.0	330.0	
3	LP001585	NaN	Yes	3+	Graduate	No	51763	0.0	700.0	
4	LP002692	Male	Yes	3+	Graduate	Yes	5677	1424.0	100.0	

In [21]:

```
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:

- Missing values
- Outliers and extreme values
- Dealing with non-numerical fields

In [22]:

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

Out[22]:

```
Male      433
Female    105
Name: Gender, dtype: int64
```

Filling in missing values by mean

As you can see there are several missing values exist in the dataset. For instance, Gender has 13 missing values and LoanAmount has 22 missing values. We need to deal with the missing values. There are various ways to deal with it. Here, we will use mean of LoanAmount to replace all it's missing values.

In [23]:

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

Out[23]:

```
Loan_ID               0
Gender               12
Married               3
Dependents           13
Education             0
Self_Employed        27
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           17
Loan_Amount_Term     14
Credit_History       46
Property_Area         0
Loan_Status           0
dtype: int64
```

In [24]:

```
data.head()
```

Out[24]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amoun
0	LP001002	Male	No	0	Graduate	No	5849	0.0	NaN	
1	LP001333	Male	Yes	0	Graduate	No	1977	997.0	50.0	
2	LP001865	Male	Yes	1	Graduate	No	6083	4250.0	330.0	
3	LP001585	NaN	Yes	3+	Graduate	No	51763	0.0	700.0	
4	LP002692	Male	Yes	3+	Graduate	Yes	5677	1424.0	100.0	

In [25]:

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

In [26]:

```
data.head()
```

Out[26]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amoun
0	LP001002	Male	No	0	Graduate	No	5849	0.0	145.195122	
1	LP001333	Male	Yes	0	Graduate	No	1977	997.0	50.000000	
2	LP001865	Male	Yes	1	Graduate	No	6083	4250.0	330.000000	
3	LP001585	NaN	Yes	3+	Graduate	No	51763	0.0	700.000000	
4	LP002692	Male	Yes	3+	Graduate	Yes	5677	1424.0	100.000000	

In [27]:

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

Out[27]:

```
Loan_ID               0
Gender               12
Married               3
Dependents           13
Education             0
Self_Employed        27
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount            0
Loan_Amount_Term     14
Credit_History       46
Property_Area         0
Loan_Status           0
dtype: int64
```

In [28]:

```
data.shape
```

Out[28]:

```
(550, 13)
```

In [29]:

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

It will be interseting to know how much loan amount could be offered to which sort of people based on their 'Education' and 'Self Employed' statues?

For this, we'll use boxplot and group by multiple variables - 'Education' and 'Self_Employed'.

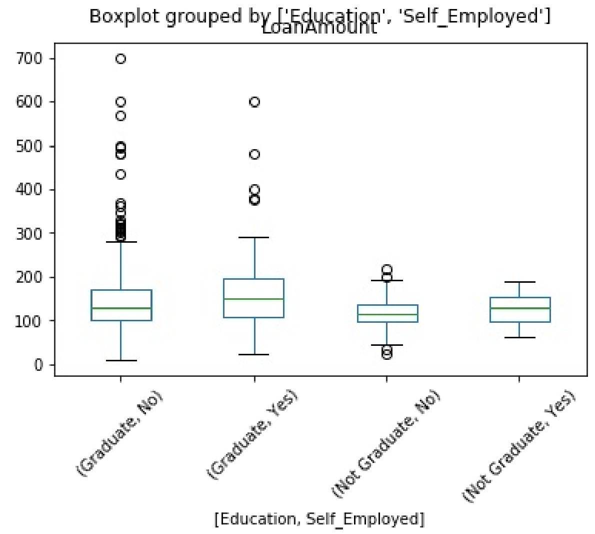
Note: LoanAmount is a numeric attribute. Whereas, Group by can be applied on numeric and non-numeric attributes.

In [30]:

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

Out[30]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1d0440e6788>
```



Impute the values

The above boxplot gives some good insight of each group receiving the loan amount. Data points that form a different patterns are outliers - in circles. We'll deal with the outliers later, for now let's observe some of the variations which are visible in the median of loan amount. And, we have seen that Self_Employed has 32 missing values. This could be a possible reason of these variations.

So let's deal with this by imputing the values.

Before that let's fill in the missing values by some suitable values - not mean this time!

In [31]:

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

Out[31]:

```
No     453
Yes     70
Name: Self_Employed, dtype: int64
```

From the frequence table of Self_Employed, we can see that around 86% values are “No”. Therefore, it is safe to impute the missing values as “No” as there is a high probability of success.

In [32]:

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

In [33]:

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

Out[33]:

```
No     480
Yes     70
Name: Self_Employed, dtype: int64
```

In [34]:

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

Out[34]:

```
Loan_ID               0
Gender               12
Married               3
Dependents           13
Education             0
Self_Employed         0
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount            0
Loan_Amount_Term     14
Credit_History       46
Property_Area         0
Loan_Status           0
dtype: int64
```

Dealing outliers

Extreme values are the minimum and the maximum values in the dataset. Values beyond extreme values are considered as outliers.

Outliers are the data points those are far away from all other data point and represent unusual patterns in the dataset.

Depending on the problem domain, outliers could be considered as an activity of interest (e.g., a malicious attack in a network) or could be ignored completely (e.g., times of the day when the network traffic are high).

Most of the learning algorithms are sensitive to outliers. Outliers can negatively influence and distort the result. Therefore, it is important to treat them. Outliers can be treated similar to missing values i.e., by removing or replacing them by appropriate values. It is also possible to take log transformation of outliers to reduce its influence.

To better understand this concept, let's visualise ‘LoanAmount’ before and after treating outliers of ‘LoanAmount’.

In [35]:

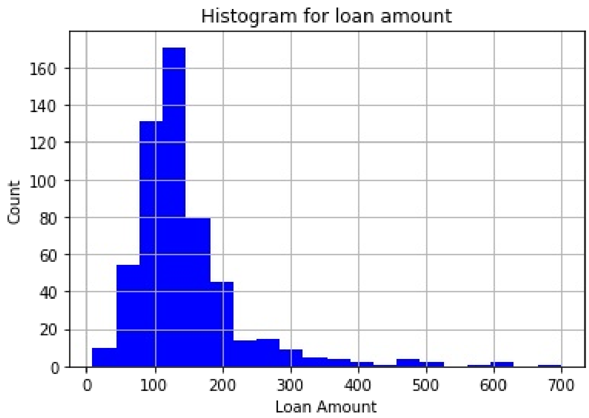
```
data.describe()
```

Out[35]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	550.000000	550.000000	550.000000	536.000000	504.000000
mean	5404.010909	1600.828945	145.195122	340.858209	0.839286
std	6294.468909	2998.437210	82.726993	66.035465	0.367632
min	150.000000	0.000000	9.000000	12.000000	0.000000
25%	2843.000000	0.000000	100.000000	360.000000	1.000000
50%	3787.500000	1106.000000	128.000000	360.000000	1.000000
75%	5741.000000	2297.250000	161.500000	360.000000	1.000000
max	81000.000000	41667.000000	700.000000	480.000000	1.000000

In [36]:

```
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()
```

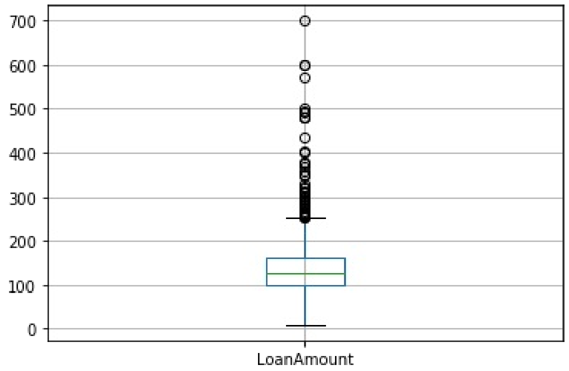


In [37]:

```
data.boxplot(column='LoanAmount')
```

Out[37]:

<matplotlib.axes._subplots.AxesSubplot at 0x1d044171488>

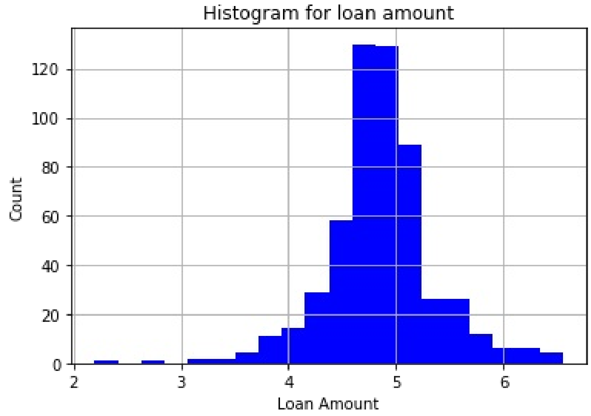


In [38]:

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

In [39]:

```
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()
```

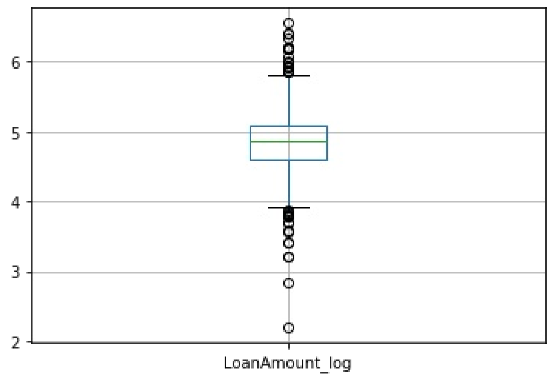


In [40]:

```
data.boxplot(column='LoanAmount_log')
```

Out[40]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1d0452fea88>
```



In [41]:

```
data.head()
```

Out[41]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amoun
0	LP001002	Male	No	0	Graduate	No	5849	0.0	145.195122	
1	LP001333	Male	Yes	0	Graduate	No	1977	997.0	50.000000	
2	LP001865	Male	Yes	1	Graduate	No	6083	4250.0	330.000000	
3	LP001585	NaN	Yes	3+	Graduate	No	51763	0.0	700.000000	
4	LP002692	Male	Yes	3+	Graduate	Yes	5677	1424.0	100.000000	

In [42]:

```
data.describe()
```

Out[42]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	LoanAmount_log
count	550.000000	550.000000	550.000000	536.000000	504.000000	550.000000
mean	5404.010909	1600.828945	145.195122	340.858209	0.839286	4.854892
std	6294.468909	2998.437210	82.726993	66.035465	0.367632	0.494421
min	150.000000	0.000000	9.000000	12.000000	0.000000	2.197225
25%	2843.000000	0.000000	100.000000	360.000000	1.000000	4.605170
50%	3787.500000	1106.000000	128.000000	360.000000	1.000000	4.852030
75%	5741.000000	2297.250000	161.500000	360.000000	1.000000	5.084491
max	81000.000000	41667.000000	700.000000	480.000000	1.000000	6.551080

It is ideal to remove 'LoanAmount' from the dataset as we have transformed it. Command below uses drop() to drop a column.

In [43]:

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

Now the distribution looks much closer to normal.

Try-it-yourself: Perform 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'.

In [ ]:

```

```

Missing values continuous: There are more missing values present in the data. Before we build the model, we need to perform some more pre-processing and convert all the values as numeric:

- Fill all the missing values.
- Convert categorical variables into numeric as sklearn works on numeric values only.

Here we will use mode() to fill in the missing values. Mode is the value which occurs most often.

In [44]:

```
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 [45]:

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

Out[45]:

```
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
dtype: int64
```

Q5. Use LabelEncoder, to convert categorical variables into numeric. Hint: You will first need to identify categorial values.

In [46]:

```
data.head()
```

Out[46]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	Loan_Amount_Term	Credit_H
0	LP001002	Male	No	0	Graduate	No	5849	0.0	360.0	
1	LP001333	Male	Yes	0	Graduate	No	1977	997.0	360.0	
2	LP001865	Male	Yes	1	Graduate	No	6083	4250.0	360.0	
3	LP001585	Male	Yes	3+	Graduate	No	51763	0.0	300.0	
4	LP002692	Male	Yes	3+	Graduate	Yes	5677	1424.0	360.0	

In [47]:

```
data.shape
```

Out[47]:

```
(550, 13)
```

In [48]:

```
from sklearn.preprocessing import LabelEncoder
```

In [49]:

```
columns = list(data)
print(columns)
```

```
['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education', 'Self_Employed', 'ApplicantIncome', 'Coa
pplicantIncome', 'Loan_Amount_Term', 'Credit_History', 'Property_Area', 'Loan_Status', 'LoanAmount_l
og']
```

In [50]:

```
data.head()
```

Out[50]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	Loan_Amount_Term	Credi
0	LP001002	Male	No	0	Graduate	No	5849	0.0	360.0	
1	LP001333	Male	Yes	0	Graduate	No	1977	997.0	360.0	
2	LP001865	Male	Yes	1	Graduate	No	6083	4250.0	360.0	
3	LP001585	Male	Yes	3+	Graduate	No	51763	0.0	300.0	
4	LP002692	Male	Yes	3+	Graduate	Yes	5677	1424.0	360.0	

In [51]:

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

In [52]:

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

In [53]:

```
data.dtypes
```

Out[53]:

```
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
dtype: object
```

In [54]:

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

In [55]:

```
data.head(25)
```

Out[55]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	Loan_Amount_Term	Cred
0	LP001002	1	0	0	0	0	5849	0.0	360.0	
1	LP001333	1	1	0	0	0	1977	997.0	360.0	
2	LP001865	1	1	1	0	0	6083	4250.0	360.0	
3	LP001585	1	1	3	0	0	51763	0.0	300.0	
4	LP002692	1	1	3	0	1	5677	1424.0	360.0	
5	LP001904	1	1	0	0	0	3103	1300.0	360.0	
6	LP002500	1	1	3	1	0	2947	1664.0	180.0	
7	LP001586	1	1	3	1	0	3522	0.0	180.0	
8	LP001993	0	0	0	0	0	3762	1666.0	360.0	
9	LP002110	1	1	1	0	0	5250	688.0	360.0	
10	LP001095	1	0	0	0	0	3167	0.0	360.0	
11	LP002158	1	1	0	1	0	3000	1666.0	480.0	
12	LP001233	1	1	1	0	0	10750	0.0	360.0	
13	LP002634	0	0	1	0	0	13262	0.0	360.0	
14	LP001616	1	1	1	0	0	3750	0.0	360.0	
15	LP001123	1	1	0	0	0	2400	0.0	360.0	
16	LP001940	1	1	2	0	0	3153	1560.0	360.0	
17	LP002494	1	0	0	0	0	6000	0.0	360.0	
18	LP001953	1	1	1	0	0	6875	0.0	360.0	
19	LP002757	0	1	0	1	0	3017	663.0	360.0	
20	LP002804	0	1	0	0	0	4180	2306.0	360.0	
21	LP002541	1	1	0	0	0	10833	0.0	360.0	
22	LP001519	0	0	0	0	0	10000	1666.0	360.0	
23	LP002588	1	1	0	0	0	4625	2857.0	12.0	
24	LP002562	1	1	1	1	0	5333	1131.0	360.0	

Data Normalisation

As can be seen in the above table each column is in different scales. For example 'ApplicantIncome' column is in the range of thousands while 'Dependents' column is usually below 10. Having features with different scales can cause problems to the machine learning model. Therefore, we perform normalisation across the columns using normalize function in sklearn. There are other ways to perform normalisation such as using StandardScaler in sklearn. You will see them in Week 5.

In [56]:

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

In [57]:

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

Out[57]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	Loan_Amount_Term	Credit_H
0	LP001002	1	0	0	0	0	5849	0.0	360.0	
1	LP001333	1	1	0	0	0	1977	997.0	360.0	
2	LP001865	1	1	1	0	0	6083	4250.0	360.0	
3	LP001585	1	1	3	0	0	51763	0.0	300.0	
4	LP002692	1	1	3	0	1	5677	1424.0	360.0	

In [58]:

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

Out[58]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	Loan_Amount_Term	Credi
0	LP001002	1	0	0	0	0	5849	0.0	360.0	
1	LP001333	1	1	0	0	0	1977	997.0	360.0	
2	LP001865	1	1	1	0	0	6083	4250.0	360.0	
3	LP001585	1	1	3	0	0	51763	0.0	300.0	
4	LP002692	1	1	3	0	1	5677	1424.0	360.0	

In [59]:

```
data.head()
```

Out[59]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	Loan_Amount_Term	Credi
0	LP001002	1	0	0	0	0	5849	0.0	360.0	
1	LP001333	1	1	0	0	0	1977	997.0	360.0	
2	LP001865	1	1	1	0	0	6083	4250.0	360.0	
3	LP001585	1	1	3	0	0	51763	0.0	300.0	
4	LP002692	1	1	3	0	1	5677	1424.0	360.0	

In [60]:

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

Out[60]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	Loan_Amount_Term	Credi
0	LP001002	1	0	0	0	0	5849	0.0	360.0	
1	LP001333	1	1	0	0	0	1977	997.0	360.0	
2	LP001865	1	1	1	0	0	6083	4250.0	360.0	
3	LP001585	1	1	3	0	0	51763	0.0	300.0	
4	LP002692	1	1	3	0	1	5677	1424.0	360.0	

In [61]:

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

We excluded 'Loan_Status' from normalisation. 'Loan_Status' is a binary class.

In [62]:

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

In [63]:

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

```
[[1.70646397e-04 0.00000000e+00 0.00000000e+00 0.00000000e+00
  0.00000000e+00 9.98110779e-01 0.00000000e+00 6.14327031e-02
  1.70646397e-04 3.41292795e-04 8.49491163e-04]
 [4.45782377e-04 4.45782377e-04 0.00000000e+00 0.00000000e+00
  0.00000000e+00 8.81311759e-01 4.44445030e-01 1.60481656e-01
  4.45782377e-04 4.45782377e-04 1.74391091e-03]
 [1.34601534e-04 1.34601534e-04 1.34601534e-04 0.00000000e+00
  0.00000000e+00 8.18781134e-01 5.72056521e-01 4.84565524e-02
  1.34601534e-04 2.69203069e-04 7.80566769e-04]
 [1.93184938e-05 1.93184938e-05 5.79554814e-05 0.00000000e+00
  0.00000000e+00 9.99983195e-01 0.00000000e+00 5.79554814e-03
  1.93184938e-05 3.86369876e-05 1.26557005e-04]
 [1.70533937e-04 1.70533937e-04 5.11601810e-04 0.00000000e+00
  1.70533937e-04 9.68121158e-01 2.42840326e-01 6.13922172e-02
  1.70533937e-04 0.00000000e+00 7.85337801e-04]]
```

Resultant normalized data (normalized_data) is in the form of ndimenssional array. Fit it back to a dataframe to perform further processing with Pandas.

In [64]:

```
normalized_data.shape
```

Out[64]:

```
(550, 11)
```

In [65]:

```
data.shape
```

Out[65]:

```
(550, 13)
```

In [66]:

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

In [67]:

```
normalized_data.head()
```

Out[67]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	Loan_Amount_Term	Credit_History
0	0.000171	0.000000	0.000000	0.0	0.000000	0.998111	0.000000	0.061433	0.000171
1	0.000446	0.000446	0.000000	0.0	0.000000	0.881312	0.444445	0.160482	0.000446
2	0.000135	0.000135	0.000135	0.0	0.000000	0.818781	0.572057	0.048457	0.000135
3	0.000019	0.000019	0.000058	0.0	0.000000	0.999983	0.000000	0.005796	0.000019
4	0.000171	0.000171	0.000512	0.0	0.000171	0.968121	0.242840	0.061392	0.000171

In [68]:

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

The above code inserts column 'Loan_ID' based on the 'index' in the dataframe.

In [69]:

```
normalized_data.head()
```

Out[69]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	Loan_Amount_Term	Credit_History
0	0.000171	0.000000	0.000000	0.0	0.000000	0.998111	0.000000	0.061433	0.000171
1	0.000446	0.000446	0.000000	0.0	0.000000	0.881312	0.444445	0.160482	0.000446
2	0.000135	0.000135	0.000135	0.0	0.000000	0.818781	0.572057	0.048457	0.000135
3	0.000019	0.000019	0.000058	0.0	0.000000	0.999983	0.000000	0.005796	0.000019
4	0.000171	0.000171	0.000512	0.0	0.000171	0.968121	0.242840	0.061392	0.000171

In [70]:

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

In [71]:

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

Out[71]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	Loan_Amount_Term	Credit_History
0	0.000171	0.000000	0.000000	0.000000	0.000000	0.998111	0.000000	0.061433	0.000171
1	0.000446	0.000446	0.000000	0.000000	0.000000	0.881312	0.444445	0.160482	0.000446
2	0.000135	0.000135	0.000135	0.000000	0.000000	0.818781	0.572057	0.048457	0.000135
3	0.000019	0.000019	0.000058	0.000000	0.000000	0.999983	0.000000	0.005796	0.000019
4	0.000171	0.000171	0.000512	0.000000	0.000171	0.968121	0.242840	0.061392	0.000171
5	0.000296	0.000296	0.000000	0.000000	0.000000	0.917091	0.384215	0.106398	0.000296
6	0.000295	0.000295	0.000885	0.000295	0.000000	0.869547	0.490983	0.053111	0.000000
7	0.000284	0.000284	0.000851	0.000284	0.000000	0.998695	0.000000	0.051041	0.000284
8	0.000000	0.000000	0.000000	0.000000	0.000000	0.910871	0.403379	0.087165	0.000242
9	0.000188	0.000188	0.000188	0.000000	0.000000	0.989238	0.129637	0.067833	0.000188

In [72]:

```python
normalized_data.describe()
```

Out[72]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	Loan_Amount_Term	Cred
count	550.000000	550.000000	550.000000	550.000000	550.000000	550.000000	550.000000	550.000000	5
mean	0.000180	0.000146	0.000165	0.000060	0.000023	0.879347	0.302389	0.078830	
std	0.000125	0.000133	0.000253	0.000121	0.000070	0.168500	0.315267	0.039279	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.009983	0.000000	0.002207	
25%	0.000084	0.000000	0.000000	0.000000	0.000000	0.797390	0.000000	0.049333	
50%	0.000190	0.000151	0.000000	0.000000	0.000000	0.970172	0.224797	0.077013	
75%	0.000267	0.000250	0.000286	0.000000	0.000000	0.997144	0.591792	0.104187	
max	0.000673	0.000589	0.001609	0.000673	0.000455	0.999996	0.999941	0.242218	

In [73]:

```python
# normalized_data['LoanAmount'].hist(bins=100)
```

Building a Decision Tree classifier using sklearn

Importing all necessary libraries from sklearn

In [74]:

```python
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:

For a small dataset, using all the features to build the feature space or a model may not be an exhaustive and expensive process. However, for a large dataset, it is not an ideal way to utilise all the features as it will result in a high dimensional feature space, and will be an exhaustive search, expensive and time-consuming job.

Feature selection is an important step of pre-processing, where we tend to remove the features that do not or less likely to contribute to the classification results. We aim to remove such features without compromising on the classification results.

There could be different ways to perform feature selection. One way is intuitive, where knowing the business problem and domain knowledge, we simply use our judgement for selecting most discriminating features. There are other automatic methods such as dimensionality reduction, statistical-based methods to identify feature importance, etc.

In this notebook, first, we will be building a baseline model using all the features. Then we will be using feature importance method available in sklearn to see the relative importance scores for each feature. You will then be required to build a new module with the identified important features and compare the results of both the models.

The process to build the model will be the same.

As you can see there are 13 features + 1 target in the final DataFrame. Remember, we have added a few new features based on the existing ones such as 'LoanAmount_log'. To build the model we can select all or sub-set of the features.

Let's perform some feature selection.

In [75]:

```
columns = list(normalized_data.columns)
columns
```

Out[75]:

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

In [76]:

```
normalized_data.head()
```

Out[76]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	Loan_Amount_Term	Credit_History
0	0.000171	0.000000	0.000000	0.0	0.000000	0.998111	0.000000	0.061433	0.000171
1	0.000446	0.000446	0.000000	0.0	0.000000	0.881312	0.444445	0.160482	0.000446
2	0.000135	0.000135	0.000135	0.0	0.000000	0.818781	0.572057	0.048457	0.000135
3	0.000019	0.000019	0.000058	0.0	0.000000	0.999983	0.000000	0.005796	0.000019
4	0.000171	0.000171	0.000512	0.0	0.000171	0.968121	0.242840	0.061392	0.000171

In [77]:

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

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

In [ ]:

In [78]:

```
print('Features:')
print(features.head())

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

```
Features:
     Gender   Married  Dependents  Education  Self_Employed  ApplicantIncome  \
0  0.000171  0.000000    0.000000        0.0       0.000000         0.998111   
1  0.000446  0.000446    0.000000        0.0       0.000000         0.881312   
2  0.000135  0.000135    0.000135        0.0       0.000000         0.818781   
3  0.000019  0.000019    0.000058        0.0       0.000000         0.999983   
4  0.000171  0.000171    0.000512        0.0       0.000171         0.968121   

   CoapplicantIncome  Loan_Amount_Term  Credit_History  Property_Area  \
0           0.000000          0.061433        0.000171       0.000341   
1           0.444445          0.160482        0.000446       0.000446   
2           0.572057          0.048457        0.000135       0.000269   
3           0.000000          0.005796        0.000019       0.000039   
4           0.242840          0.061392        0.000171       0.000000   

   LoanAmount_log  
0        0.000849  
1        0.001744  
2        0.000781  
3        0.000127  
4        0.000785  
Classes:
   Loan_Status
0            1
1            1
2            1
3            1
4            1
```

In [79]:

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

Out[79]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	Loan_Amount_Term	Credit_History
0	0.000171	0.000000	0.000000	0.000000	0.000000	0.998111	0.000000	0.061433	0.000171
1	0.000446	0.000446	0.000000	0.000000	0.000000	0.881312	0.444445	0.160482	0.000446
2	0.000135	0.000135	0.000135	0.000000	0.000000	0.818781	0.572057	0.048457	0.000135
3	0.000019	0.000019	0.000058	0.000000	0.000000	0.999983	0.000000	0.005796	0.000019
4	0.000171	0.000171	0.000512	0.000000	0.000171	0.968121	0.242840	0.061392	0.000171
5	0.000296	0.000296	0.000000	0.000000	0.000000	0.917091	0.384215	0.106398	0.000296
6	0.000295	0.000295	0.000885	0.000295	0.000000	0.869547	0.490983	0.053111	0.000000
7	0.000284	0.000284	0.000851	0.000284	0.000000	0.998695	0.000000	0.051041	0.000284
8	0.000000	0.000000	0.000000	0.000000	0.000000	0.910871	0.403379	0.087165	0.000242
9	0.000188	0.000188	0.000188	0.000000	0.000000	0.989238	0.129637	0.067833	0.000188

In [80]:

```
normalized_data.shape
```

Out[80]:

```
(550, 13)
```

Building our first baseline model using all the features. Partitioning data into Train and Test sets: You will need to replace random_state = '2' with the 'last 4 digits of your student number'.

In [81]:

```
normalized_data.shape
```

Out[81]:

```
(550, 13)
```

In [82]:

```
#https://machinelearningmastery.com/calculate-feature-importance-with-python/
```

In [83]:

```
from matplotlib import pyplot
```

In [84]:

```
x_train, x_test, y_train, y_test = train_test_split(features, classes, test_size= .33,
                                                    random_state = 1)

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

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

In [85]:

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

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

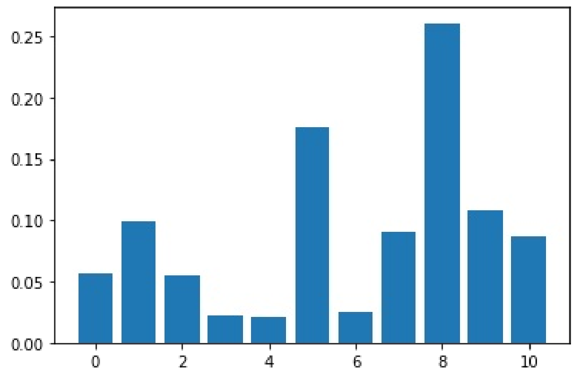
In [86]:

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

In [87]:

```
# 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: 0.05666
Feature: 1, Score: 0.09857
Feature: 2, Score: 0.05524
Feature: 3, Score: 0.02191
Feature: 4, Score: 0.02104
Feature: 5, Score: 0.17618
Feature: 6, Score: 0.02467
Feature: 7, Score: 0.09057
Feature: 8, Score: 0.26040
Feature: 9, Score: 0.10773
Feature: 10, Score: 0.08703
```



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 [88]:

```
prediction = dtc_model.predict(x_test) #prediction stores the predicted targets/classes
```

Since we converted the categorical values eariler using a label encoder, let's convert them back.

In [89]:

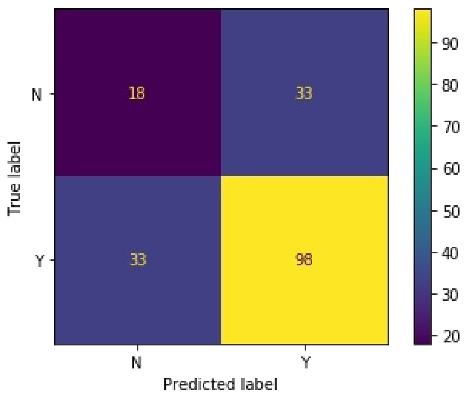
```
y_true = le.inverse_transform(y_test["Loan_Status"])
y_pred = le.inverse_transform(prediction)
```

In [90]:

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

Out[90]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1d0467786c8>
```



In [91]:

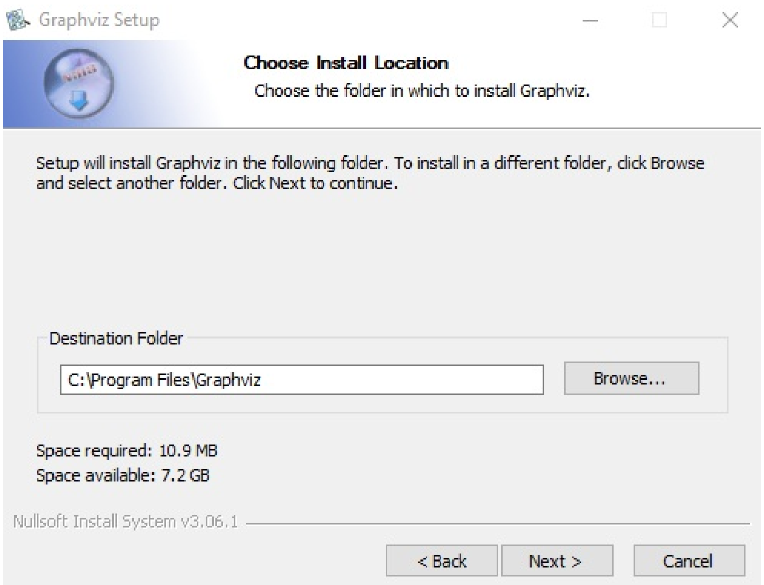
```
print(classification_report(y_true, y_pred))
```

```
              precision    recall  f1-score   support

           N       0.35      0.35      0.35        51
           Y       0.75      0.75      0.75       131

    accuracy                           0.64       182
   macro avg       0.55      0.55      0.55       182
weighted avg       0.64      0.64      0.64       182
```

Visualising the decision tree

We are visualising the decision tree in order to analyse it better. We are using Graphviz (https://graphviz.org/) to do that.

First, download Graphviz from here (https://graphviz.org/download/). Select a version that is suitable for your computer. After downloading, install it on your computer. In the installation process, you will come across a screen similar to this.



Please remember the path to the destination folder as we need it later.

Second, we need to add Graphviz executables to the system path. Add "/bin/" to the destination folder path you provided in the above step and execute the following command. Please note that this path can be different in your system, and you need to provide the correct path.

In [92]:

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

In [93]:

```
import os

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

Third, you need to install graphviz python library. Use the "pip install graphviz" in order to do that. Also, we are using cairosvg ti convert the SVG output of graphviz to a png.

Now you can visualise the decision tree using the following code. We will be using the first decision tree "dtc_model".

In [94]:

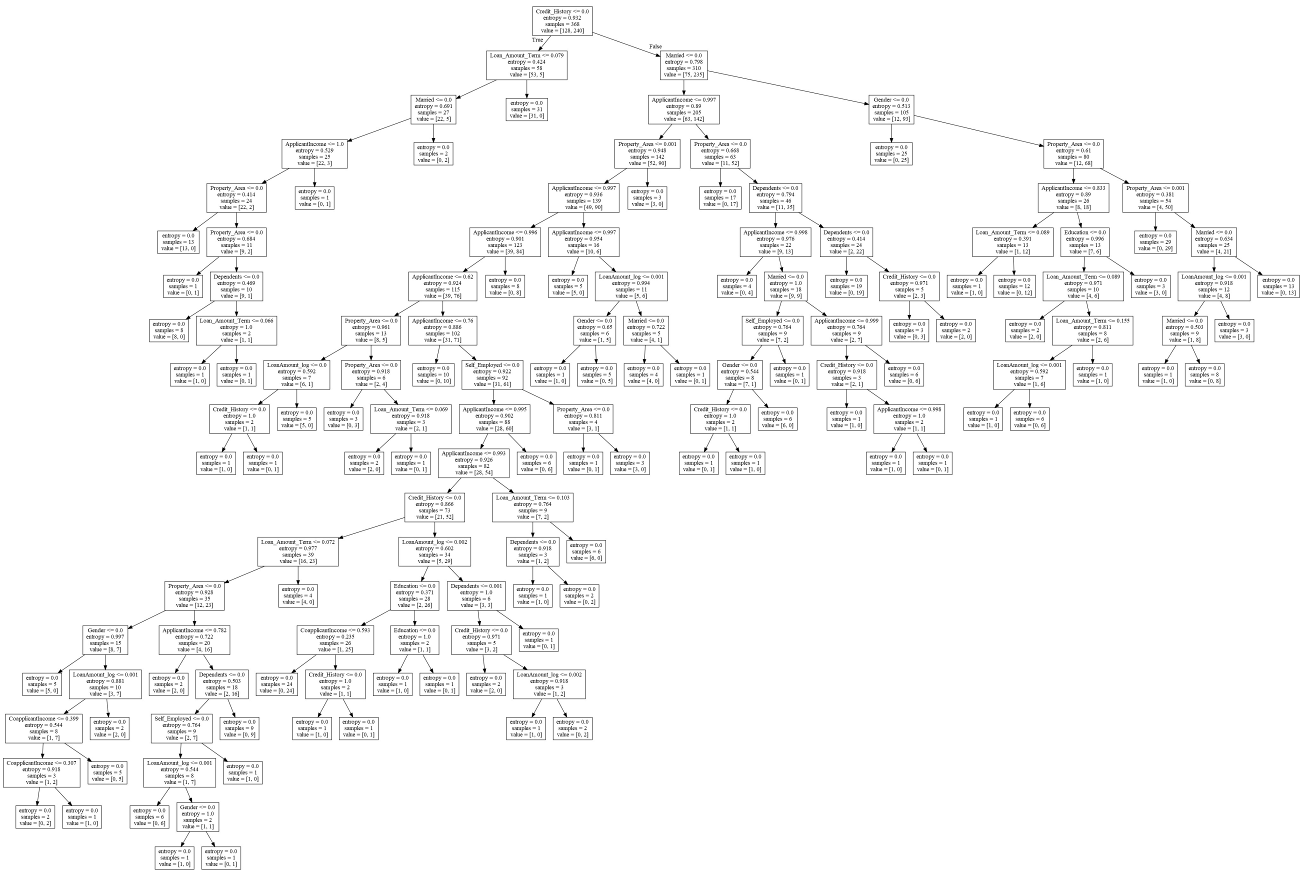
```
from graphviz import Source
from sklearn import tree
graph = Source( tree.export_graphviz(dtc_model, out_file=None, feature_names=features.columns))
```

You can display the graph with the following code.

In [95]:

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


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

Out[95]:



Report

Answer the following questions. Please provide code as as appropriate to answer the questions.

Q6. Based on the feature importance, select a different set of features to build another dicision tree model. You should aim to improve the result of the baseline model.

HINT: Look at the feature importance section of the Notebook.

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.

Q8. Discuss the result based on the evaluation matrix (max 250 words).

Reference

Analytics Vidhya. A Complete Python Tutorial to Learn Data Science from Scratch. Available online: [here (https://www.analyticsvidhya.com/blog/2016/01/complete-tutorial-learn-data-science-python-scratch-2/)](https://www.analyticsvidhya.com/blog/2016/01/complete-tutorial-learn-data-science-python-scratch-2/)

End of the Workshop 2.