In [1]:

```python
import pandas as pd
import seaborn as sns
```

In [2]:

```python
df=pd.read_csv('Admission_predict.csv')
```

In [3]:

```python
df.columns
```

Out[3]:

```
Index(['Serial No.', 'GRE Score', 'TOEFL Score', 'University Rating', 'SOP',
       'LOR ', 'CGPA', 'Research', 'Chance of Admit '],
      dtype='object')
```

In [4]:

```python
df.shape
```

Out[4]:

```
(400, 9)
```

In [6]:

```python
df.head()
```

Out[6]:

| | Serial No. | GRE Score | TOEFL Score | University Rating | SOP | LOR | CGPA | Research | Chance of Admit |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 337 | 118 | 4 | 4.5 | 4.5 | 9.65 | 1 | 0.92 |
| **1** | 2 | 324 | 107 | 4 | 4.0 | 4.5 | 8.87 | 1 | 0.76 |
| **2** | 3 | 316 | 104 | 3 | 3.0 | 3.5 | 8.00 | 1 | 0.72 |
| **3** | 4 | 322 | 110 | 3 | 3.5 | 2.5 | 8.67 | 1 | 0.80 |
| **4** | 5 | 314 | 103 | 2 | 2.0 | 3.0 | 8.21 | 0 | 0.65 |

In [8]:

```python
# Binarizer- It is used to give some values above Threshold value and below the Threshold value
# Here we don't need to use Lable Encodindg as the data is in numbers only and not in strings.
# Also don't need to use Scaling as we are using Decision Tree Algorithm

from sklearn.preprocessing import Binarizer
bi= Binarizer(threshold=0.75) # This will store the value '1' above the "0.75 / 75%" and '0' below it.
df['Chance of Admit ']= bi.fit_transform(df[['Chance of Admit ']])

# Note: Here we have to give Extra Space in the name of Last column or it will throw error.
# Also here we have to provide the 2D Array i.e. with 2 Square Brackets as above or it will throw Error.
```

In [9]:

```python
df.head()
```

Out[9]:

| | Serial No. | GRE Score | TOEFL Score | University Rating | SOP | LOR | CGPA | Research | Chance of Admit |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 337 | 118 | 4 | 4.5 | 4.5 | 9.65 | 1 | 1.0 |
| **1** | 2 | 324 | 107 | 4 | 4.0 | 4.5 | 8.87 | 1 | 1.0 |
| **2** | 3 | 316 | 104 | 3 | 3.0 | 3.5 | 8.00 | 1 | 0.0 |
| **3** | 4 | 322 | 110 | 3 | 3.5 | 2.5 | 8.67 | 1 | 1.0 |
| **4** | 5 | 314 | 103 | 2 | 2.0 | 3.0 | 8.21 | 0 | 0.0 |

In [10]:

```python
x=df.drop('Chance of Admit ', axis=1) # Here we dropped the last column and 'x' is the Input Variable and 'y' i
y=df['Chance of Admit ']
# Here axis=1 is used to represent that it is Column
```

In [11]:

```python
x
```

Out[11]:

| | Serial No. | GRE Score | TOEFL Score | University Rating | SOP | LOR | CGPA | Research |
|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 337 | 118 | 4 | 4.5 | 4.5 | 9.65 | 1 |
| **1** | 2 | 324 | 107 | 4 | 4.0 | 4.5 | 8.87 | 1 |
| **2** | 3 | 316 | 104 | 3 | 3.0 | 3.5 | 8.00 | 1 |
| **3** | 4 | 322 | 110 | 3 | 3.5 | 2.5 | 8.67 | 1 |
| **4** | 5 | 314 | 103 | 2 | 2.0 | 3.0 | 8.21 | 0 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... |
| **395** | 396 | 324 | 110 | 3 | 3.5 | 3.5 | 9.04 | 1 |
| **396** | 397 | 325 | 107 | 3 | 3.0 | 3.5 | 9.11 | 1 |
| **397** | 398 | 330 | 116 | 4 | 5.0 | 4.5 | 9.45 | 1 |
| **398** | 399 | 312 | 103 | 3 | 3.5 | 4.0 | 8.78 | 0 |
| **399** | 400 | 333 | 117 | 4 | 5.0 | 4.0 | 9.66 | 1 |

400 rows × 8 columns

In [12]:

```
# If we print the 'y' then we can understand that it is having "Float" values
y
```

Out[12]:

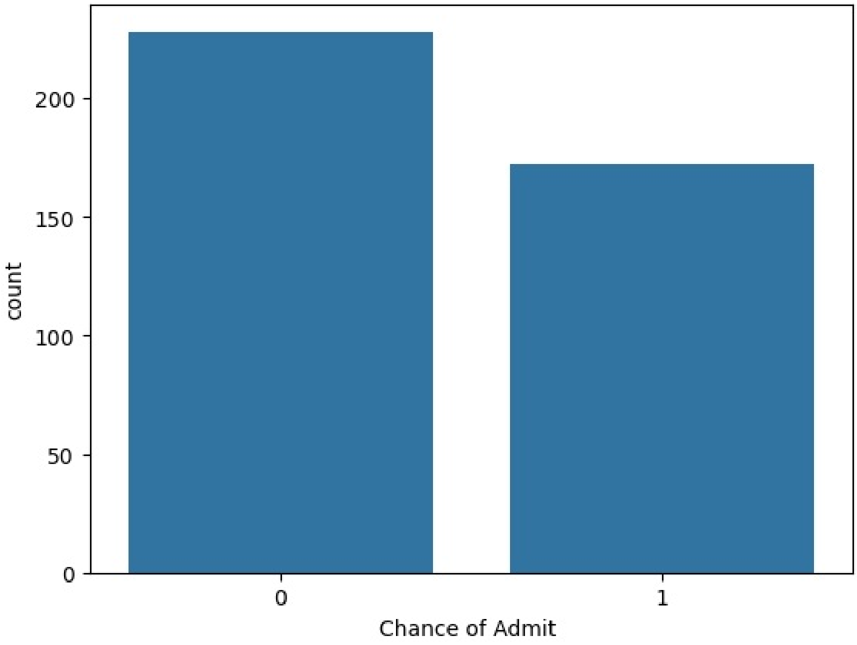
```
0      1.0
1      1.0
2      0.0
3      1.0
4      0.0
      ...
395    1.0
396    1.0
397    1.0
398    0.0
399    1.0
Name: Chance of Admit , Length: 400, dtype: float64
```

In [16]:

```
# To convert it into "Integer"
y=y.astype('int') # Here 'astype' is an Series Class Key Method, because every column is an series
```

In [18]:

```
# If we want to know that how many entries are there in the 'y' then use
sns.countplot(x=y); # Here 'x' is an Keyword Argument and 'y' is the value passed to it.
```



In [19]:

```
# It is used to get the values not Graphically but Numerically(means in Numbers)
y.value_counts()
```

Out[19]:

```
Chance of Admit
0    228
1    172
Name: count, dtype: int64
```

In [20]:

```
# Cross-validation --> To divide the data in the Training and Testing
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x,y,random_state=0,test_size=0.25)
```

In [21]:

```
x_train.shape
```

Out[21]:

```
(300, 8)
```

In [22]:

```
x_test.shape
```

Out[22]:

```
(100, 8)
```

In [24]:

```
x_test # This is used to show that the entries are Random
```

Out[24]:

| | Serial No. | GRE Score | TOEFL Score | University Rating | SOP | LOR | CGPA | Research |
|---|---|---|---|---|---|---|---|---|
| **132** | 133 | 309 | 105 | 5 | 3.5 | 3.5 | 8.56 | 0 |
| **309** | 310 | 308 | 110 | 4 | 3.5 | 3.0 | 8.60 | 0 |
| **341** | 342 | 326 | 110 | 3 | 3.5 | 3.5 | 8.76 | 1 |
| **196** | 197 | 306 | 105 | 2 | 3.0 | 2.5 | 8.26 | 0 |
| **246** | 247 | 316 | 105 | 3 | 3.0 | 3.5 | 8.73 | 0 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... |
| **146** | 147 | 315 | 105 | 3 | 2.0 | 2.5 | 8.48 | 0 |
| **135** | 136 | 314 | 109 | 4 | 3.5 | 4.0 | 8.77 | 1 |
| **390** | 391 | 314 | 102 | 2 | 2.0 | 2.5 | 8.24 | 0 |
| **264** | 265 | 325 | 110 | 2 | 3.0 | 2.5 | 8.76 | 1 |
| **364** | 365 | 313 | 102 | 3 | 3.5 | 4.0 | 8.90 | 1 |

100 rows × 8 columns

In [25]:

```
# To create Model, we have to import the Decision Tree Classifier class from Scikitlearn package.
# Import the Class
from sklearn.tree import DecisionTreeClassifier
```

In [28]:

```
classifier= DecisionTreeClassifier(random_state=0)
# Here we have created the Object of the Class &nwe have given the 'random_state=0' as output of everyone remai
```

In [29]:

```
# To train model we use 'fit() method' and this results in the formation of the "DecisionTree"
classifier.fit(x_train,y_train)
```

Out[29]:

```
▾ DecisionTreeClassifier
DecisionTreeClassifier(random_state=0)
```

In [30]:

```
# Now we are going to check the Accuracy of the Model on the Data which is unknoen to the model ie. Test Data
y_pred= classifier.predict(x_test) # Here we have passed the 100 test data entries to the model.
```

In [31]:

```
# Now we are going to create a Dataframe to check the accuracy of the model
result= pd.DataFrame({
    'actual': y_test,
    'predicted': y_pred
})
```

In [32]:

```
result
```

Out[32]:

| | actual | predicted |
|---|---|---|
| **132** | 0 | 0 |
| **309** | 0 | 0 |
| **341** | 1 | 1 |
| **196** | 0 | 0 |
| **246** | 0 | 1 |
| **...** | ... | ... |
| **146** | 0 | 0 |
| **135** | 1 | 1 |
| **390** | 0 | 0 |
| **264** | 0 | 0 |
| **364** | 1 | 1 |

100 rows × 2 columns

In [33]:

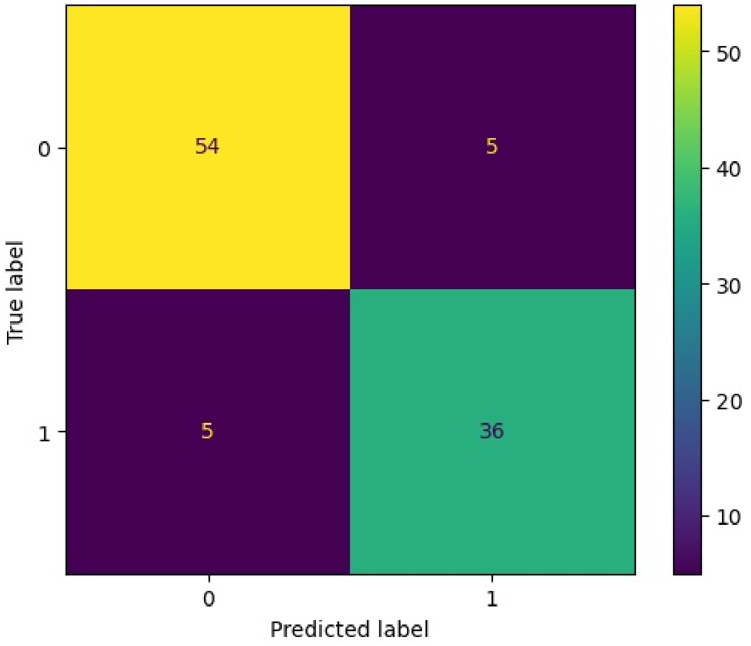
```
# Here as you see the middle entries are truncated. So to check it's Accuracy we use Confusion Matrix.
from sklearn.metrics import ConfusionMatrixDisplay, accuracy_score
from sklearn.metrics import classification_report # It is used to calculate the Accuracy, Precision, Recall, F1
```

In [34]:

```
ConfusionMatrixDisplay.from_predictions(y_test,y_pred) # Here we used 'from_predictions()' to analyse the Confu
```

Out[34]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2185c549550>
```



In [35]:

```
# Here as you see the value of matching "0" are 54 and of "1" are 36, i.e. here total 90 out of 100 values matc
accuracy_score(y_test,y_pred)
```

Out[35]:

```
0.9
```

In [36]:

```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.92      0.92      0.92        59
           1       0.88      0.88      0.88        41

    accuracy                           0.90       100
   macro avg       0.90      0.90      0.90       100
weighted avg       0.90      0.90      0.90       100
```

In [40]:

```
# Now if we want to check that the model will correctly predict that I can get admission or not for that,
new= [[136, 314, 109, 4, 3.5, 4.0, 8.77, 1]]
classifier.predict(new)[0]
```

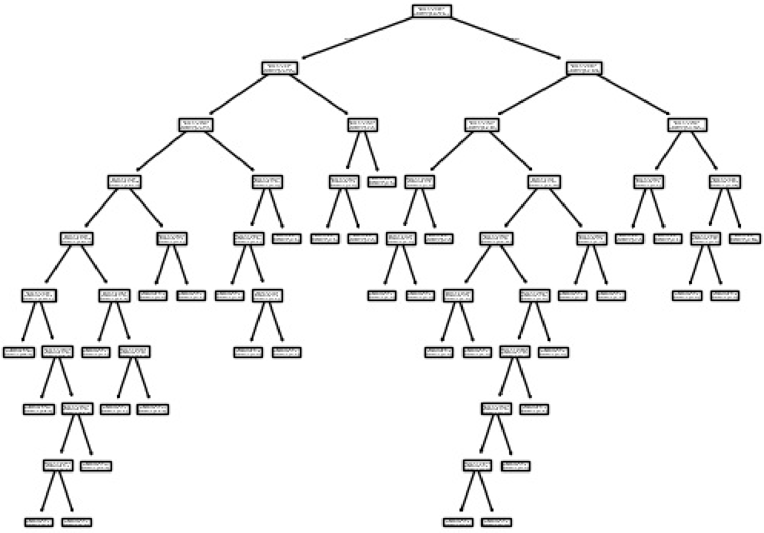
```
C:\Users\urkha\anaconda3\Lib\site-packages\sklearn\utils\validation.py:2739: UserWarning: X does not have valid
feature names, but DecisionTreeClassifier was fitted with feature names
  warnings.warn(
```

Out[40]:

```
np.int64(1)
```

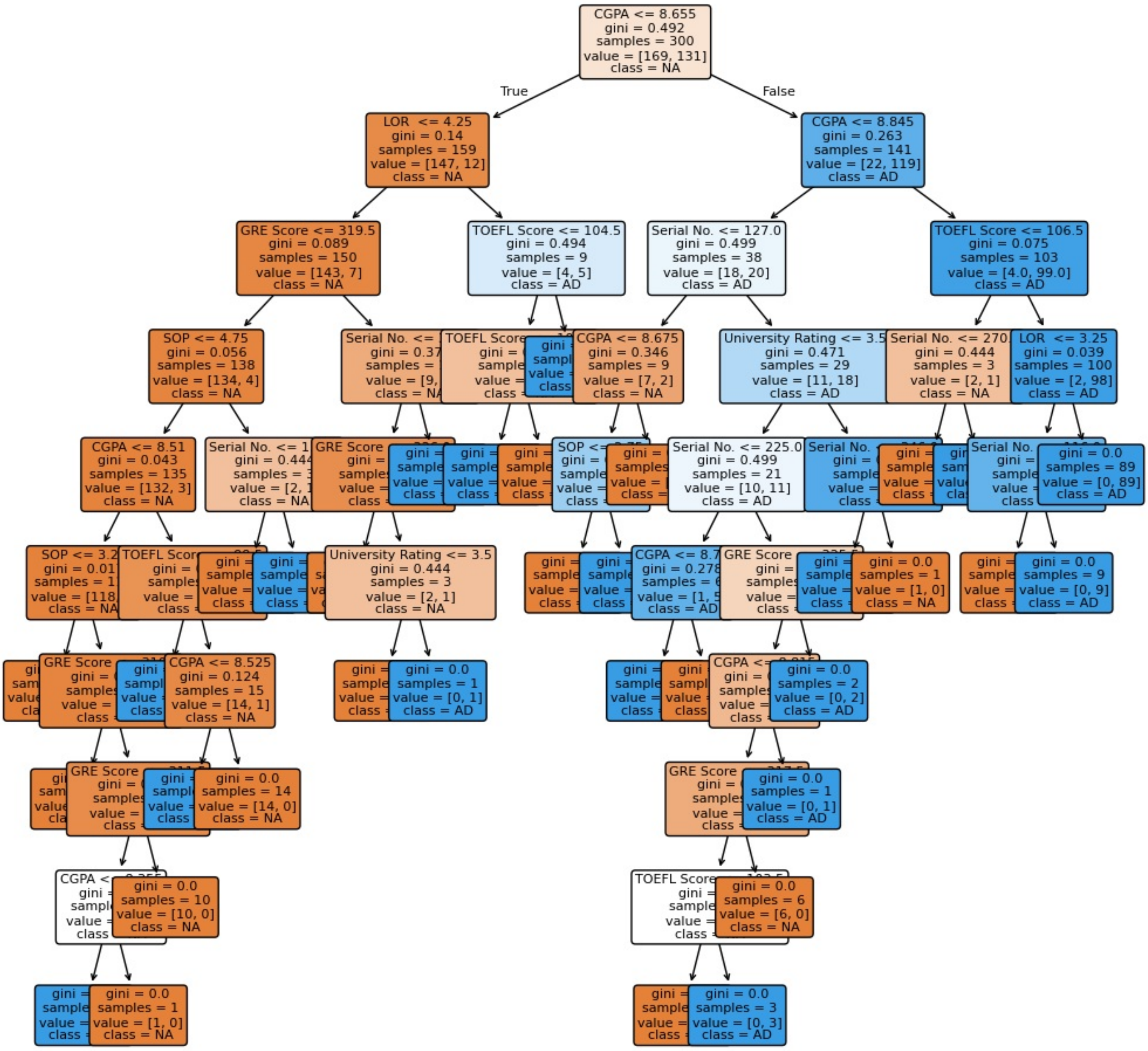
In [41]:
```
# Here as the output is '1' means you will get admitted.
# Now to plot the Decision Tree
from sklearn.tree import plot_tree
```

In [44]:
```
plot_tree(classifier, ); # Remember that we have to write it as shown here otherwise it will not give the Decis
```



In [46]:
```
import matplotlib.pyplot as plt
```

In [49]:
```
plt.figure(figsize=(12,12))
plot_tree(classifier, fontsize=8, filled=True, rounded=True, feature_names=x.columns, class_names=['NA','AD']);
```


CGPA <= 8.655
gini = 0.492
samples = 300
value = [169, 131]
class = NA
True
False
LOR <= 4.25
gini = 0.14
samples = 159
value = [147, 12]
class = NA
CGPA <= 8.845
gini = 0.263
samples = 141
value = [22, 119]
class = AD
GRE Score <= 319.5
gini = 0.089
samples = 150
value = [143, 7]
class = NA
TOEFL Score <= 104.5
gini = 0.494
samples = 9
value = [4, 5]
class = AD
Serial No. <= 127.0
gini = 0.499
samples = 38
value = [18, 20]
class = AD
TOEFL Score <= 106.5
gini = 0.075
samples = 103
value = [4.0, 99.0]
class = AD
SOP <= 4.75
gini = 0.056
samples = 138
value = [134, 4]
class = NA
CGPA <= 8.675
gini = 0.346
samples = 9
value = [7, 2]
class = NA
University Rating <= 3.5
gini = 0.471
samples = 29
value = [11, 18]
class = AD
Serial No. <= 270
gini = 0.444
samples = 3
value = [2, 1]
class = NA
LOR <= 3.25
gini = 0.039
samples = 100
value = [2, 98]
class = AD
CGPA <= 8.51
gini = 0.043
samples = 135
value = [132, 3]
class = NA
Serial No. <= 225.0
gini = 0.499
samples = 21
value = [10, 11]
class = AD
gini = 0.0
samples = 89
value = [0, 89]
class = AD
University Rating <= 3.5
gini = 0.444
samples = 3
value = [2, 1]
class = NA
gini = 0.0
samples = 1
value = [1, 0]
class = NA
gini = 0.0
samples = 9
value = [0, 9]
class = AD
CGPA <= 8.525
gini = 0.124
samples = 15
value = [14, 1]
class = NA
gini = 0.0
samples = 1
value = [0, 1]
class = AD
gini = 0.0
samples = 2
value = [0, 2]
class = AD
gini = 0.0
samples = 14
value = [14, 0]
class = NA
gini = 0.0
samples = 1
value = [0, 1]
class = AD
gini = 0.0
samples = 10
value = [10, 0]
class = NA
gini = 0.0
samples = 6
value = [6, 0]
class = NA
gini = 0.0
samples = 1
value = [1, 0]
class = NA
gini = 0.0
samples = 3
value = [0, 3]
class = AD

```
In [ ]:
```