In [ ]:
```
#Practical No 3 Classification with Decision Tree
```

In [1]:
```
import pandas as pd
import seaborn as sns
```

In [2]:
```
df=pd.read_csv('admission_data.csv')
```

In [3]:
```
df
```

Out[3]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	337	118	4	4.5	4.5	9.65	1	0.92
1	324	107	4	4.0	4.5	8.87	1	0.76
2	316	104	3	3.0	3.5	8.00	1	0.72
3	322	110	3	3.5	2.5	8.67	1	0.80
4	314	103	2	2.0	3.0	8.21	0	0.65
...	...	...	...	...	...	...	...	...
495	332	108	5	4.5	4.0	9.02	1	0.87
496	337	117	5	5.0	5.0	9.87	1	0.96
497	330	120	5	4.5	5.0	9.56	1	0.93
498	312	103	4	4.0	5.0	8.43	0	0.73
499	327	113	4	4.5	4.5	9.04	0	0.84

500 rows × 8 columns

In [4]:
```
df.columns
```

Out[4]:
```
Index(['GRE Score', 'TOEFL Score', 'University Rating', 'SOP',
'LOR ', 'CGPA',
       'Research', 'Chance of Admit '],
      dtype='object')
```

In [5]:
```
df.shape
```

Out[5]:
```
(500, 8)
```

In [6]:
```
df.head()
```

Out[6]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	337	118	4	4.5	4.5	9.65	1	0.92
1	324	107	4	4.0	4.5	8.87	1	0.76
2	316	104	3	3.0	3.5	8.00	1	0.72
3	322	110	3	3.5	2.5	8.67	1	0.80
4	314	103	2	2.0	3.0	8.21	0	0.65

In [15]:
```
df.rename(columns={'Chance of Admit ': 'Chance of Admit'}, inplac
```

In [16]:
```
a = df.loc[df["Chance of Admit"] >= 0.80, 'Chance of Admit'] = 1
b = df.loc[df["Chance of Admit"] < 0.80, 'Chance of Admit'] = 0
#Assigns 1 to rows where the chance of admit is greater than or e
#Assigns 0 to rows where the chance of admit is less than 0.80.
```

In [17]:
```
a
```

Out[17]:
```
1
```

In [18]:
```
b
```

Out[18]:
```
0
```

In [25]:
```
from sklearn.preprocessing import Binarizer
bf = Binarizer(threshold=0.75)
df['Chance of Admit'] = bf.fit_transform(df[['Chance of Admit']])
#This code uses Binarizer to transform the Chance of Admit
#column by setting values greater than 0.75 to 1
#and values less than or equal to 0.75 to 0.
```

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

Out[26]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	337	118	4	4.5	4.5	9.65	1	1.0
1	324	107	4	4.0	4.5	8.87	1	0.0
2	316	104	3	3.0	3.5	8.00	1	0.0
3	322	110	3	3.5	2.5	8.67	1	1.0
4	314	103	2	2.0	3.0	8.21	0	0.0

In [29]:
```
x=df.drop('Chance of Admit', axis=1)
y=df['Chance of Admit']
#The code x = df.drop('Chance of Admit', axis=1) creates a new Da
#Chance of Admit column from df,
#while y = df['Chance of Admit'] assigns the Chance of Admit colu
```

In [28]:
```
x
```

Out[28]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research
0	337	118	4	4.5	4.5	9.65	1
1	324	107	4	4.0	4.5	8.87	1
2	316	104	3	3.0	3.5	8.00	1
3	322	110	3	3.5	2.5	8.67	1
4	314	103	2	2.0	3.0	8.21	0
...	...	...	...	...	...	...	...
495	332	108	5	4.5	4.0	9.02	1
496	337	117	5	5.0	5.0	9.87	1
497	330	120	5	4.5	5.0	9.56	1
498	312	103	4	4.0	5.0	8.43	0
499	327	113	4	4.5	4.5	9.04	0

500 rows × 7 columns

In [30]:
```
y
```

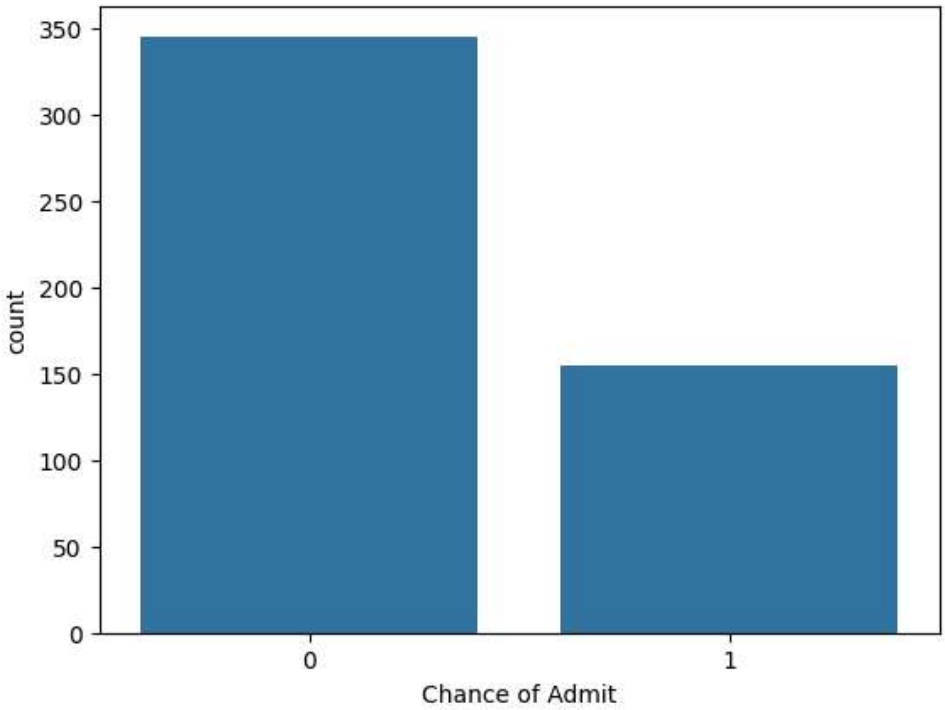
Out[30]:
```
0      1.0
1      0.0
2      0.0
3      1.0
4      0.0
      ...
495    1.0
496    1.0
497    1.0
498    0.0
499    1.0
Name: Chance of Admit, Length: 500, dtype: float64
```

In [33]:
```
y=y.astype('int')
#The code y = y.astype('int') converts the data type of the y var
#(e.g., a pandas Series or NumPy array) to integers.
```

In [32]:
```
y
```

Out[32]:
```
0      1
1      0
2      0
3      1
4      0
      ..
495    1
496    1
497    1
498    0
499    1
Name: Chance of Admit, Length: 500, dtype: int32
```

In [34]:
```
sns.countplot(x=y);
#The code sns.countplot(x=y) creates a count plot using Seaborn,
#which visualizes the frequency of unique values in the variable
```



In [35]:
```
y.value_counts()
#The code y.value_counts() counts the unique values in the
#Series y and returns a Series containing the counts of each uniq
#sorted in descending order.
```

Out[35]:
```
0    345
1    155
Name: Chance of Admit, dtype: int64
```

In [ ]:
```
#Cross Validation
```

In [36]:
```
from sklearn.model_selection import train_test_split
```

In [37]:
```
x_train, x_test, y_train, y_test= train_test_split(x,y,random_sta
#The code splits the dataset into training and
#testing subsets for features (x) and target labels (y),
#using a random state for reproducibility,
#with 25% of the data allocated for testing.
```

In [38]:
```
x_train.shape
```

Out[38]:
```
(375, 7)
```

In [39]:
```
x_test.shape
```

Out[39]:
```
(125, 7)
```

In [40]:
```
x_test
```

Out[40]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research
90	318	106	2	4.0	4.0	7.92	1
254	321	114	4	4.0	5.0	9.12	0
283	321	111	3	2.5	3.0	8.90	1
445	328	116	5	4.5	5.0	9.08	1
461	301	102	3	2.5	2.0	8.13	1
...	...	...	...	...	...	...	...
430	311	104	3	4.0	3.5	8.13	1
49	327	111	4	3.0	4.0	8.40	1
134	333	113	5	4.0	4.0	9.28	1
365	330	114	4	4.5	3.0	9.17	1
413	317	101	3	3.0	2.0	7.94	1

125 rows × 7 columns

In [41]:
```
#import the class
from sklearn.tree import DecisionTreeClassifier
```

In [42]:
```
classifier= DecisionTreeClassifier(random_state=0)
#Initializes a Decision Tree Classifier with a fixed random state
```

In [43]:
```
classifier.fit(x_train,y_train)
#Trains the classifier using the training data (x_train) and corr
```

Out[43]:
```
DecisionTreeClassifier(random_state=0)
```

In [44]:
```
y_pred=classifier.predict(x_test)
#Uses the trained classifier to make predictions on the test data
```

In [45]:
```
y_pred
#Creates a DataFrame to compare actual labels (y_test) with the p
```

Out[45]:
```
array([0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,
0, 0, 0,
       0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,
1, 1, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0,
1, 0, 0,
       0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1,
0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0,
1, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0])
```

In [46]:
```
result=pd.DataFrame({'actual':y_test,'predicted':y_pred})
#result containing two columns:
#one for the actual test values and another for the predicted val
```

In [47]:
```
result
#Displays the resulting DataFrame containing the actual and predi
```

Out[47]:

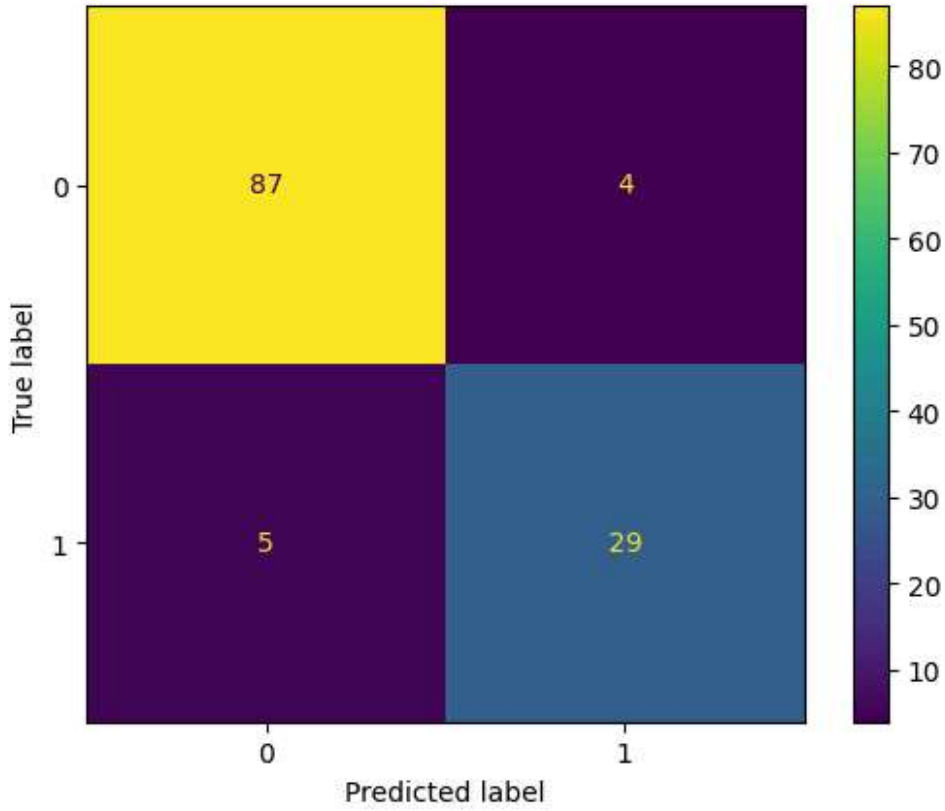
	actual	predicted
90	0	0
254	1	1
283	1	0
445	1	1
461	0	0
...	...	...
430	0	0
49	0	0
134	1	1
365	1	1
413	0	0

125 rows × 2 columns

In [52]:
```
from sklearn.metrics import ConfusionMatrixDisplay, accuracy_scor
from sklearn.metrics import classification_report
```

In [54]:
```
ConfusionMatrixDisplay.from_predictions(y_test, y_pred)
#visualizes the confusion matrix based on the
#true labels (y_test) and the predicted labels (y_pred) for a cla
```

Out[54]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1daa80c4400>



In [55]:
```
accuracy_score(y_test, y_pred)
#Calculates the accuracy of the model by comparing the true label
```

Out[55]: 0.928

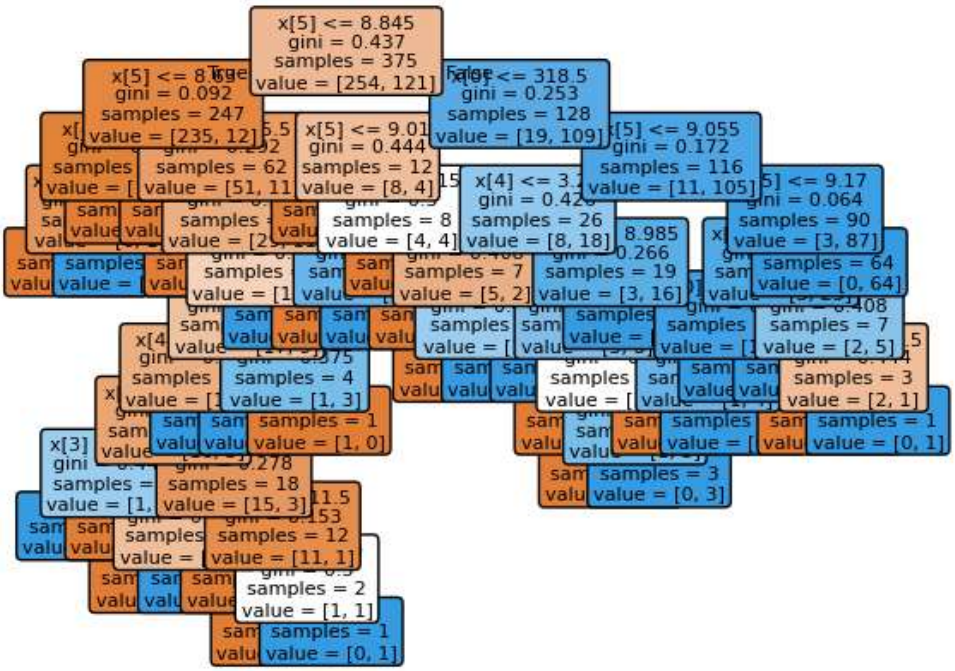
In [58]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.95      0.96      0.95        91
           1       0.88      0.85      0.87        34

    accuracy                           0.93       125
   macro avg       0.91      0.90      0.91       125
weighted avg       0.93      0.93      0.93       125
```

In [69]:
```
new=[[136, 314, 109,4,3,5,4.0,8.77,1]]
```

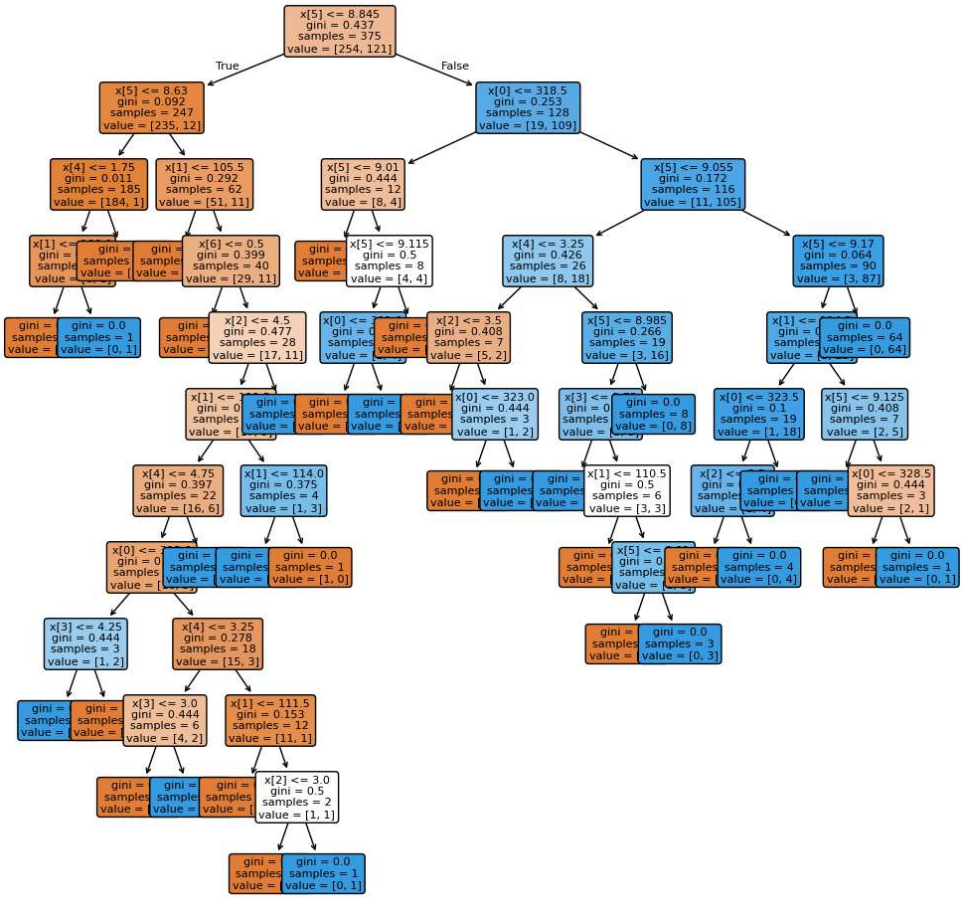
In [72]:
```
from sklearn.tree import plot_tree
```

In [73]:
```
plot_tree(classifier, fontsize=8, filled=True,rounded=True);
#Visualizes the decision tree structure of the classifier with
#specified font size and rounded edges for better readability.
```

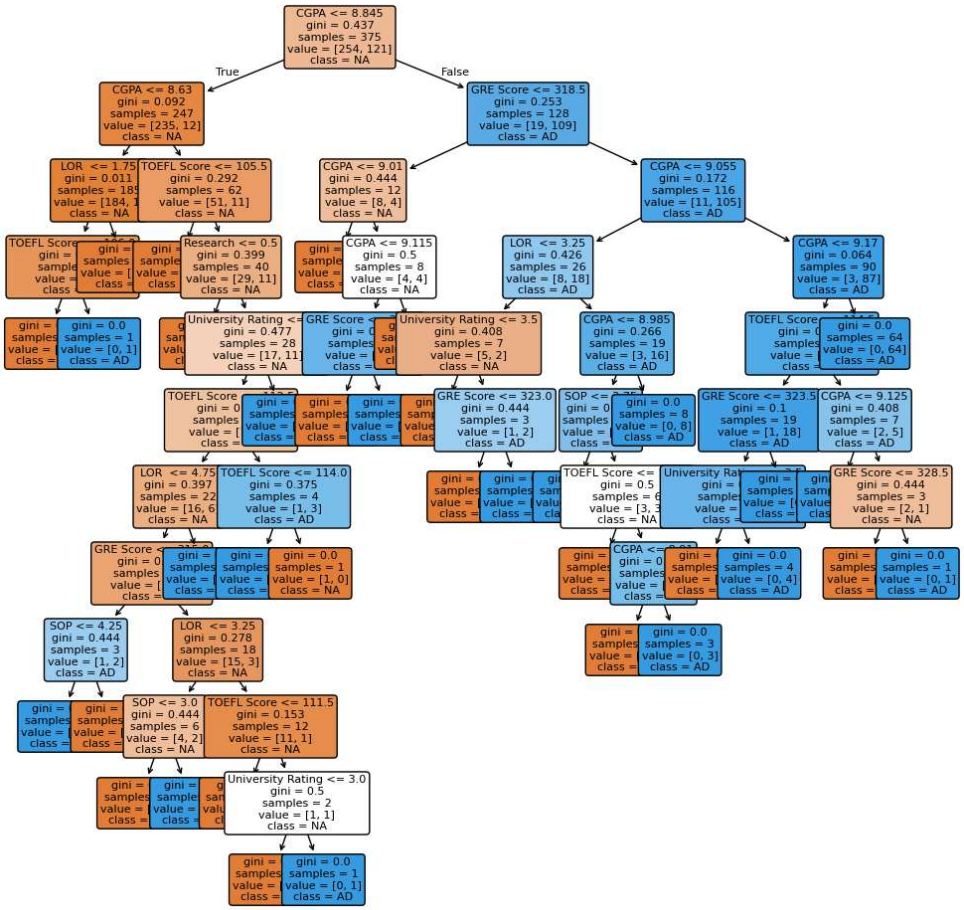


In [74]:
```
import matplotlib.pyplot as plt
```

In [75]:
```
plt.figure(figsize=(12,12))
#Creates a new figure for plotting with a specified size of 12x12
plot_tree(classifier, fontsize=8, filled=True, rounded=True);
#Again visualizes the decision tree structure of the
#classifier with the same formatting options.
```



In [78]:
```
plt.figure(figsize=(12,12))
#Creates another new figure for plotting, again with a size of 12
plot_tree(classifier, fontsize=8, filled=True,rounded=True,
          feature_names=x.columns, class_names=['NA','AD']);
#Visualizes the decision tree structure of the classifier
#one more time with the same formatting options.
```


CGPA <= 8.845
gini = 0.437
samples = 375
value = [254, 121]
class = NA
True
False
CGPA <= 8.63
gini = 0.092
samples = 247
value = [235, 12]
class = NA
GRE Score <= 318.5
gini = 0.253
samples = 128
value = [19, 109]
class = AD
LOR <= 1.75
gini = 0.011
TOEFL Score <= 105.5
gini = 0.292
samples = 62
value = [51, 11]
class = NA
CGPA <= 9.01
gini = 0.444
samples = 12
value = [8, 4]
class = NA
CGPA <= 9.055
gini = 0.172
samples = 116
value = [11, 105]
class = AD
Research <= 0.5
gini = 0.399
samples = 40
value = [29, 11]
class = NA
CGPA <= 9.115
gini = 0.5
samples = 8
value = [4, 4]
class = NA
LOR <= 3.25
gini = 0.426
samples = 26
value = [8, 18]
class = AD
CGPA <= 9.17
gini = 0.064
samples = 90
value = [3, 87]
class = AD
gini = 0.0
samples = 1
value = [0, 1]
class = AD
gini = 0.0
samples = 64
value = [0, 64]
class = AD
University Rating <=
gini = 0.477
samples = 28
value = [17, 11]
class = NA
University Rating <= 3.5
gini = 0.408
samples = 7
value = [5, 2]
class = NA
CGPA <= 8.985
gini = 0.266
samples = 19
value = [3, 16]
class = AD
GRE Score <= 323.0
gini = 0.444
samples = 3
value = [1, 2]
class = AD
gini = 0.0
samples = 8
value = [0, 8]
class = AD
GRE Score <= 323.5
gini = 0.1
samples = 19
value = [1, 18]
class = AD
CGPA <= 9.125
gini = 0.408
samples = 7
value = [2, 5]
class = AD
LOR <= 4.75
gini = 0.397
TOEFL Score <= 114.0
gini = 0.375
samples = 4
value = [1, 3]
class = AD
GRE Score <= 328.5
gini = 0.444
samples = 3
value = [2, 1]
class = NA
gini = 0.0
samples = 1
value = [1, 0]
class = NA
gini = 0.0
samples = 4
value = [0, 4]
class = AD
SOP <= 4.25
gini = 0.444
samples = 3
value = [1, 2]
class = AD
LOR <= 3.25
gini = 0.278
samples = 18
value = [15, 3]
class = NA
gini = 0.0
samples = 3
value = [0, 3]
class = AD
SOP <= 3.0
gini = 0.444
samples = 6
value = [4, 2]
class = NA
TOEFL Score <= 111.5
gini = 0.153
samples = 12
value = [11, 1]
class = NA
University Rating <= 3.0
gini = 0.5
samples = 2
value = [1, 1]
class = NA
gini = 0.0
samples = 1
value = [0, 1]
class = AD

In [ ]: