```
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np
from sklearn.tree import DecisionTreeClassifier,plot_tree
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix

from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score

from sklearn.model_selection import cross_val_score
```

```
df=pd.read_csv("Admission_Predict.csv")
df.dropna(inplace=True)
```

```
df.head()
```

	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

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
df.columns
```

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

```
#new column CoA
df['CoA'] = np.where(df.iloc[:,8] > 0.9, 1, 0)
```

```
df.head()
```

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit	CoA

0	1	337	118	4	4.5	4.5	9.65	1	0.92	1
1	2	324	107	4	4.0	4.5	8.87	1	0.76	0
2	3	316	104	3	3.0	3.5	8.00	1	0.72	0
3	4	322	110	3	3.5	2.5	8.67	1	0.80	0
4	5	314	103	2	2.0	3.0	8.21	0	0.65	0

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
x = df[["GRE Score", "TOEFL Score", "University Rating", "SOP", "LOR ", "CGPA",
y = df["CoA"]
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, randon
```

```
tree_model = DecisionTreeClassifier(criterion="gini", max_depth=3,random_state=
tree_model.fit(x_train, y_train)
```

```
▾            DecisionTreeClassifier                 i ?
DecisionTreeClassifier(max_depth=3, random_state=1)
```

```
predicted = tree_model.predict(x_test)
```

```
print(y_test.values, "\n\n\n", predicted)
```

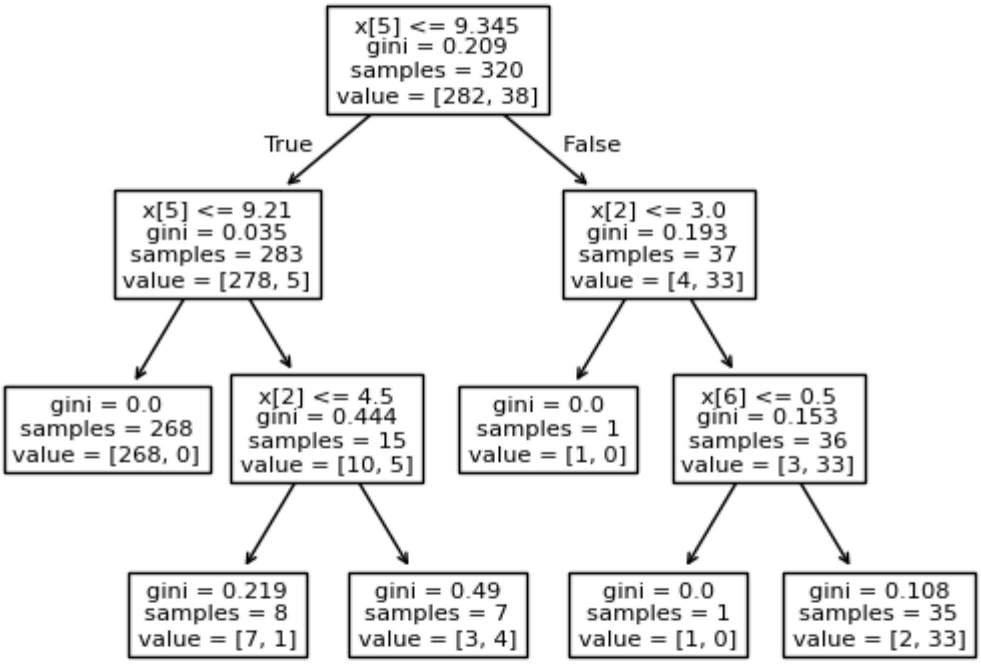
```
[0 0 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 0 0 0 0
 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0]


 [0 0 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 0 0 0 0
 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0]
```

```
print("accuracy score is: ", accuracy_score(y_test.values,predicted))
print("confusion matrix is: \n", confusion_matrix(y_test.values,predicted))
print("precision score is: ", precision_score(y_test.values,predicted))
print("recall score is: ", recall_score(y_test.values,predicted))
print("f1 score is: ", f1_score(y_test.values,predicted))
```

```
accuracy score is:  0.975
confusion matrix is: 
 [[68  1]
 [ 1 10]]
precision score is:  0.9090909090909091
recall score is:  0.9090909090909091
recall score is:  0.9090909090909091
f1 score is:  0.9090909090909091
```

```
plot_tree(tree_model)
```

```
[Text(0.4444444444444444, 0.875, 'x[5] <= 9.345\ngini = 0.209\nsamples =
320\nvalue = [282, 38]'),
 Text(0.2222222222222222, 0.625, 'x[5] <= 9.21\ngini = 0.035\nsamples =
283\nvalue = [278, 5]'),
 Text(0.3333333333333333, 0.75, 'True  '),
 Text(0.1111111111111111, 0.375, 'gini = 0.0\nsamples = 268\nvalue = [268,
0]'),
 Text(0.3333333333333333, 0.375, 'x[2] <= 4.5\ngini = 0.444\nsamples =
15\nvalue = [10, 5]'),
 Text(0.2222222222222222, 0.125, 'gini = 0.219\nsamples = 8\nvalue = [7,
1]'),
 Text(0.4444444444444444, 0.125, 'gini = 0.49\nsamples = 7\nvalue = [3,
4]'),
 Text(0.6666666666666666, 0.625, 'x[2] <= 3.0\ngini = 0.193\nsamples =
37\nvalue = [4, 33]'),
 Text(0.5555555555555556, 0.75, '  False'),
 Text(0.5555555555555556, 0.375, 'gini = 0.0\nsamples = 1\nvalue = [1,
0]'),
 Text(0.7777777777777778, 0.375, 'x[6] <= 0.5\ngini = 0.153\nsamples =
36\nvalue = [3, 33]'),
 Text(0.6666666666666666, 0.125, 'gini = 0.0\nsamples = 1\nvalue = [1,
0]'),
 Text(0.8888888888888888, 0.125, 'gini = 0.108\nsamples = 35\nvalue = [2,
33]')]
```



Start coding or generate with AI.