DecessionTreeClassifier

Akshat Kumar (242211003) 12-02-2025

```
# Libraries
import pandas as pd
from time import time,ctime
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
```

```
print("Timestamp: "+ctime(time()))
```

```
Timestamp: Wed Feb 12 06:27:22 2025
```

```
df = pd.read_csv('diabetes.csv')
df.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

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

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

Data Exploration and Preprocessing:

```
df.shape
```

```
(768, 9)
```

```
df.columns
```

```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

```
df.describe()
```

```
       Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin         BMI  DiabetesPedigreeFunction         Age     Outcome
count   768.000000  768.000000     768.000000     768.000000  768.000000  768.000000                768.000000  768.000000  768.000000
mean      3.845052  120.894531      69.105469      20.536458   79.799479   31.992578                  0.471876   33.240885    0.348958
std       3.369578   31.972618      19.355807      15.952218  115.244002    7.884160                  0.331329   11.760232    0.476951
min       0.000000    0.000000       0.000000       0.000000    0.000000    0.000000                  0.078000   21.000000    0.000000
25%       1.000000   99.000000      62.000000       0.000000    0.000000   27.300000                  0.243750   24.000000    0.000000
50%       3.000000  117.000000      72.000000      23.000000   30.500000   32.000000                  0.372500   29.000000    0.000000
75%       6.000000  140.250000      80.000000      32.000000  127.250000   36.600000                  0.626250   41.000000    1.000000
max      17.000000  199.000000     122.000000      99.000000  846.000000   67.100000                  2.420000   81.000000    1.000000
```

```
df.Outcome.value_counts()*100/len(df)
```

```
             count
Outcome
0        65.104167
1        34.895833

dtype: float64
```

```
df.groupby('Outcome').mean()
```

```
         Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin        BMI  DiabetesPedigreeFunction        Age
Outcome
0           3.298000  109.980000      68.184000      19.664000   68.792000  30.304200                  0.429734  31.190000
1           4.865672  141.257463      70.824627      22.164179  100.335821  35.142537                  0.550500  37.067164
```

```
df.groupby('Outcome').agg(['mean','median'])
```

```
         Pregnancies        Glucose              BloodPressure        SkinThickness        Insulin              BMI                 DiabetesPed
         mean        median mean        median   mean       median    mean       median    mean        median   mean       median   mean
Outcome
0        3.298000       2.0 109.980000   107.0   68.184000    70.0    19.664000    21.0     68.792000    39.0   30.304200   30.05     0.42973
1        4.865672       4.0 141.257463   140.0   70.824627    74.0    22.164179    27.0    100.335821     0.0   35.142537   34.25     0.55050
```

2. Handle missing values (if any) appropriately.

```
df.isnull().sum()
```

```
                          0
Pregnancies               0
Glucose                   0
BloodPressure             0
SkinThickness             0
Insulin                   0
BMI                       0
DiabetesPedigreeFunction  0
Age                       0
Outcome                   0

dtype: int64
```

Model Implementation:

1. Split the dataset into training and testing sets (e.g., 80-20 split).

```
Y = df['Outcome']
Y
```

	Outcome
0	1
1	0
2	1
3	0
4	1
...	...
763	0
764	0
765	0
766	1
767	0

768 rows × 1 columns

dtype: int64

```
X = df.drop('Outcome',axis=1)
X
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	6	148	72	35	0	33.6	0.627	50
1	1	85	66	29	0	26.6	0.351	31
2	8	183	64	0	0	23.3	0.672	32
3	1	89	66	23	94	28.1	0.167	21
4	0	137	40	35	168	43.1	2.288	33
...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63
764	2	122	70	27	0	36.8	0.340	27
765	5	121	72	23	112	26.2	0.245	30
766	1	126	60	0	0	30.1	0.349	47
767	1	93	70	31	0	30.4	0.315	23

768 rows × 8 columns

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

```
X_train , X_test, y_train, y_test = train_test_split(X,Y,test_size=0.2,random_state=42)
print('X_train is \n',X_train,'\n\nY_train is \n',y_train )
```

```
X_train is
      Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
60             2       84              0              0        0   0.0
618            9      112             82             24        0  28.2
346            1      139             46             19       83  28.7
294            0      161             50              0        0  21.9
231            6      134             80             37      370  46.2
..           ...      ...            ...            ...      ...   ...
71             5      139             64             35      140  28.6
106            1       96            122              0        0  22.4
270           10      101             86             37        0  45.6
435            0      141              0              0        0  42.4
102            0      125             96              0        0  22.5

     DiabetesPedigreeFunction  Age
60                      0.304   21
618                     1.282   50
346                     0.654   22
294                     0.254   65
231                     0.238   46
..                        ...  ...
```

```
71                        0.411   26
106                       0.207   27
270                       1.136   38
435                       0.205   29
102                       0.262   21

[614 rows x 8 columns]

Y_train is
 60     0
618    1
346    0
294    0
231    1
      ..
71     0
106    0
270    1
435    1
102    0
Name: Outcome, Length: 614, dtype: int64
```

```
dtc = DecisionTreeClassifier()
dtc = dtc.fit(X_train,y_train)

y_pred = dtc.predict(X_test)
y_pred
```

```
array([0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1,
       0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1,
       1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1,
       0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
       0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0])
```

```
print("Accuracy: ",round(metrics.accuracy_score(y_test,y_pred)*100,2),"%")
```

```
Accuracy:  75.97 %
```

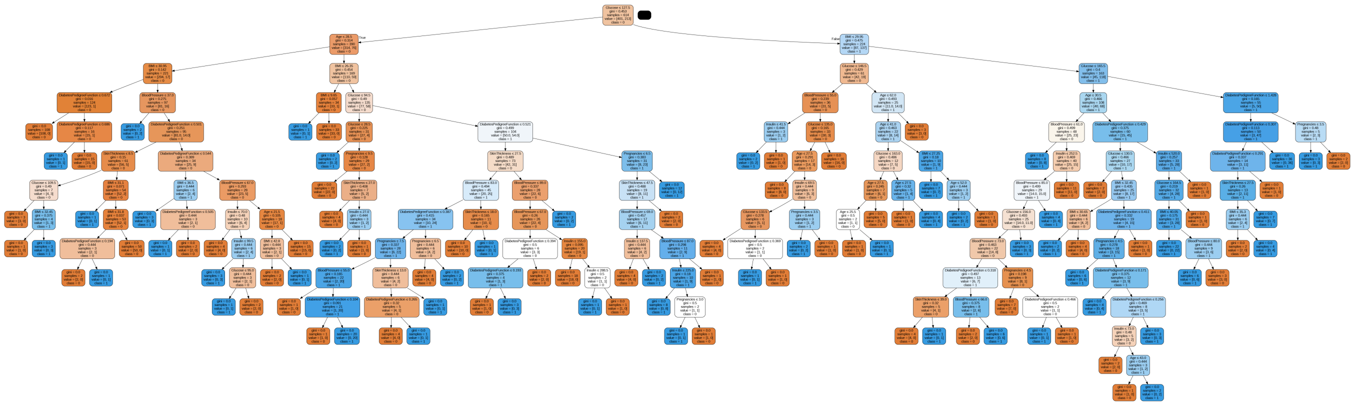
2. Build and train a Decision Tree model using a library like Scikit-learn.

```
!pip install pydotplus
```

```
Requirement already satisfied: pydotplus in /usr/local/lib/python3.11/dist-packages (2.0.2)
Requirement already satisfied: pyparsing>=2.0.1 in /usr/local/lib/python3.11/dist-packages (from pydotplus) (3.2.1)
```

```
from sklearn.tree import export_graphviz
from six import StringIO
from IPython.display import Image
import pydotplus
```

```
dot_data = StringIO()
export_graphviz(dtc, out_file=dot_data,filled=True, rounded=True,special_characters=True, feature_names=X.columns,class_names=['0','1']
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('diabetes.png')
Image(graph.create_png())
```



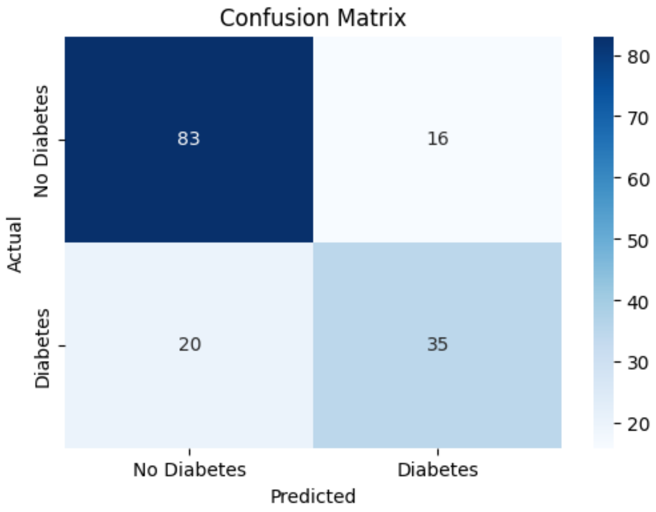
3. Experiment with hyperparameters such as the maximum depth, minimum samples per leaf, and splitting criteria (e.g., Gini Index, Entropy).

```
dtc = DecisionTreeClassifier(criterion='entropy',max_depth=3)
dtc = dtc.fit(X_train,y_train)

y_pred = dtc.predict(X_test)
y_pred
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0,
       0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1,
       0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0,
       0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1,
       0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0])
```

```
import seaborn as sns
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
# Confusion Matrix
plt.figure(figsize=(6,4))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt="d", cmap="Blues", xticklabels=["No Diabetes", "Diabetes"], yticklabels=['
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
# Model Evaluation
y_pred = dtc.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

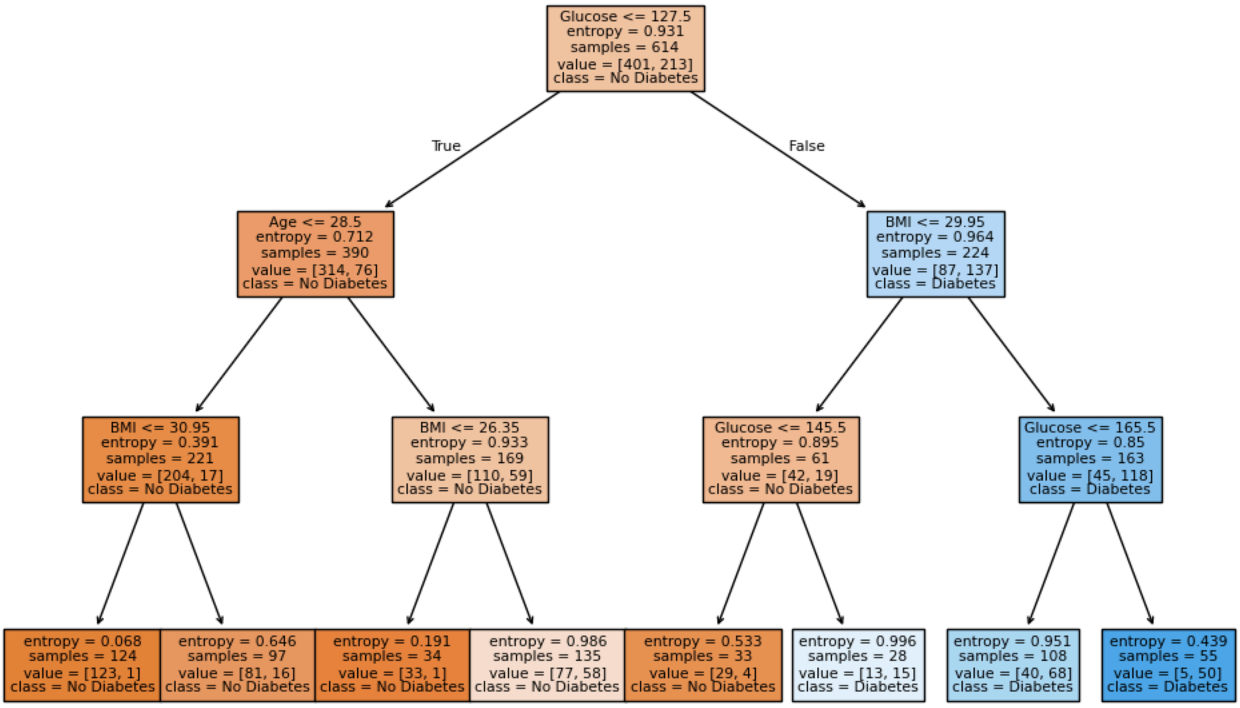


```
Accuracy: 0.7662337662337663
Classification Report:
               precision    recall  f1-score   support

           0       0.81      0.84      0.82        99
           1       0.69      0.64      0.66        55

    accuracy                           0.77       154
   macro avg       0.75      0.74      0.74       154
weighted avg       0.76      0.77      0.76       154
```

```
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree
# Visualizing the Decision Tree
plt.figure(figsize=(12, 8))
plot_tree(dtc, feature_names=X.columns, class_names=["No Diabetes", "Diabetes"], filled=True)
plt.show()
```


Glucose <= 127.5
entropy = 0.931
samples = 614
value = [401, 213]
class = No Diabetes
True
False
Age <= 28.5
entropy = 0.712
samples = 390
value = [314, 76]
class = No Diabetes
BMI <= 29.95
entropy = 0.964
samples = 224
value = [87, 137]
class = Diabetes
BMI <= 30.95
entropy = 0.391
samples = 221
value = [204, 17]
class = No Diabetes
BMI <= 26.35
entropy = 0.933
samples = 169
value = [110, 59]
class = No Diabetes
Glucose <= 145.5
entropy = 0.895
samples = 61
value = [42, 19]
class = No Diabetes
Glucose <= 165.5
entropy = 0.85
samples = 163
value = [45, 118]
class = Diabetes
entropy = 0.068
samples = 124
value = [123, 1]
class = No Diabetes
entropy = 0.646
samples = 97
value = [81, 16]
class = No Diabetes
entropy = 0.191
samples = 34
value = [33, 1]
class = No Diabetes
entropy = 0.986
samples = 135
value = [77, 58]
class = No Diabetes
entropy = 0.533
samples = 33
value = [29, 4]
class = No Diabetes
entropy = 0.996
samples = 28
value = [13, 15]
class = Diabetes
entropy = 0.951
samples = 108
value = [40, 68]
class = Diabetes
entropy = 0.439
samples = 55
value = [5, 50]
class = Diabetes