ML and DL

635 | Vishal Baraiya | 23010101014

Lab - 6

Decision Tree Classifier

In [1]:

```
from sklearn.datasets import load_iris
```

Import iris data ser using sklearn

In [2]:

```
data = load_iris()
```

In [3]:

```
type(data)
```

Out[3]:

```
sklearn.utils._bunch.Bunch
```

Importing the libraries

In [4]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
```

Divide the data into input and output

```
In [5]: x = data["data"]
```

```
In [6]: y = data["target"]
        y
```

```
Out[6]: array([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, 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, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
               1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
               1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
               2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
               2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

Splitting the dataset into the Training set and Test set

```
In [7]: x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3,random_state=42)
```

Fitting DecisionTreeClassifier on dataset

```
In [8]: model = DecisionTreeClassifier(random_state=42)
```

```
In [9]: model.fit(x_train, y_train)
```

```
Out[9]: ▾ DecisionTreeClassifier  ⓘ ?
        DecisionTreeClassifier(random_state=42)
```

Display Decision Tree

```
In [10]: %whos
```

```
Variable                 Type                      Data/Info
------------------------------------------------------------
DecisionTreeClassifier   ABCMeta                   <class 'sklearn.tree._cla<...>.De
cisionTreeClassifier'>
data                     Bunch                     {'data': array([[5.1, 3.5<...> 's
klearn.datasets.data'}
load_iris                function                  <function load_iris at 0x00000185
8702A200>
model                    DecisionTreeClassifier    DecisionTreeClassifier(random_sta
te=42)
np                       module                    <module 'numpy' from 'C:\<...>ges
\\numpy\\__init__.py'>
pd                       module                    <module 'pandas' from 'C:<...>es
\\pandas\\__init__.py'>
plt                      module                    <module 'matplotlib.pyplo<...>\\m
atplotlib\\pyplot.py'>
sns                      module                    <module 'seaborn' from 'C<...>s
\\seaborn\\__init__.py'>
train_test_split         function                  <function train_test_split at 0x0
00001858A636F20>
tree                     module                    <module 'sklearn.tree' fr<...>ear
n\\tree\\__init__.py'>
x                        ndarray                   150x4: 600 elems, type `float64`,
4800 bytes
x_test                   ndarray                   45x4: 180 elems, type `float64`,
1440 bytes
x_train                  ndarray                   105x4: 420 elems, type `float64`,
3360 bytes
y                        ndarray                   150: 150 elems, type `int32`, 600
bytes
y_test                   ndarray                   45: 45 elems, type `int32`, 180 b
ytes
y_train                  ndarray                   105: 105 elems, type `int32`, 420
bytes
```

In [11]:

```
%matplotlib qt
```

In [12]:

```
plt.figure(figsize=(20, 10))
tree.plot_tree(
    model,
    filled=True,
    feature_names=data.feature_names,
    class_names=data.target_names,
    rounded=True
)
plt.show()
```

Predict the x_test

In [13]:

```
y_pred = model.predict(x_test)
y_pred
```

Out[13]:

```
array([1, 0, 2, 1, 1, 0, 1, 2, 1, 1, 2, 0, 0, 0, 0, 1, 2, 1, 1, 2, 0, 2,
       0, 2, 2, 2, 2, 2, 0, 0, 0, 0, 1, 0, 0, 2, 1, 0, 0, 0, 2, 1, 1, 0,
       0])
```

In [14]:

```
from sklearn.metrics import accuracy_score

print("Accuracy:", accuracy_score(y_test, y_pred))
```

```
Accuracy: 1.0
```

Import diabetes.csv dataset

In [15]:

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

Out[15]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFu
0	6	148	72	35	0	33.6	
1	1	85	66	29	0	26.6	
2	8	183	64	0	0	23.3	
3	1	89	66	23	94	28.1	
4	0	137	40	35	168	43.1	
...	...	...	...	...	...	...	
763	10	101	76	48	180	32.9	
764	2	122	70	27	0	36.8	
765	5	121	72	23	112	26.2	
766	1	126	60	0	0	30.1	
767	1	93	70	31	0	30.4	

768 rows × 9 columns

Check the distribution of the target

In [16]:

```
print(df['Outcome'].value_counts())
```

```
Outcome
Non Diabetic    500
Diabetic        268
Name: count, dtype: int64
```

Replace zeros with NaN for specific columns

In [17]:

```
columns_to_replace = ['Glucose','BloodPressure','SkinThickness','Insulin','BMI']
df[columns_to_replace] = df[columns_to_replace].replace(0,np.nan)
```

Check for missing values

In [18]:

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

```
Pregnancies                   0
Glucose                       5
BloodPressure                35
SkinThickness               227
Insulin                     374
BMI                          11
DiabetesPedigreeFunction      0
Age                           0
Outcome                       0
dtype: int64
```

Fill missing values with median

In [19]:

```
df.fillna(df.median(numeric_only=True), inplace=True)
```

In [20]:

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

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

Visualize Distributions

In [21]:

```
df.hist(figsize=(12, 10), bins=20)
plt.tight_layout()
plt.show()
```

In [22]:

```
sns.countplot(x='Outcome', data=df)
plt.title('Distribution of Outcome')
plt.show()
```

Convert Targer data into interger code

In [23]:

```
df['Outcome'] = df['Outcome'].map({'Non Diabetic': 0, 'Diabetic': 1})
```

In [24]:

```
df.head()
```

Out[24]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunc
0	6	148.0	72.0	35.0	125.0	33.6	(
1	1	85.0	66.0	29.0	125.0	26.6	(
2	8	183.0	64.0	29.0	125.0	23.3	(
3	1	89.0	66.0	23.0	94.0	28.1	(
4	0	137.0	40.0	35.0	168.0	43.1	2

Divide the data into input and output

In [25]:

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

Out[25]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFu
0	6	148.0	72.0	35.0	125.0	33.6	
1	1	85.0	66.0	29.0	125.0	26.6	
2	8	183.0	64.0	29.0	125.0	23.3	
3	1	89.0	66.0	23.0	94.0	28.1	
4	0	137.0	40.0	35.0	168.0	43.1	
...	...	...	...	...	...	...	
763	10	101.0	76.0	48.0	180.0	32.9	
764	2	122.0	70.0	27.0	125.0	36.8	
765	5	121.0	72.0	23.0	112.0	26.2	
766	1	126.0	60.0	29.0	125.0	30.1	
767	1	93.0	70.0	31.0	125.0	30.4	

768 rows × 8 columns

In [26]:

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

Out[26]:

```
0      1
1      0
2      1
3      0
4      1
      ..
763    0
764    0
765    0
766    1
767    0
Name: Outcome, Length: 768, dtype: int64
```

Splitting the dataset into the Training set and Test set

In [27]:

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_sta
```

Create Model

In [28]:

```
dt_model = DecisionTreeClassifier(random_state=42)
```

Fitting DecisionTreeClassifier on dataset

In [29]:
```
dt_model.fit(X_train, Y_train)
```

Out[29]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier(random_state=42)
```

Display Decision Tree

In [30]:
```
plt.figure(figsize=(25, 15))
tree.plot_tree(
    dt_model,
    filled=True,
    feature_names=X.columns,
    class_names=['No Diabetes', 'Diabetes'],
    rounded=True
)
plt.show()
```

Predict the x_test

In [31]:
```
Y_pred = dt_model.predict(X_test)
Y_pred
```

Out[31]:
```
array([0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0,
       0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1,
       0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1,
       1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1,
       0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1,
       0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0,
       1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1,
       1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0], dtype=int64)
```

Display Training Accuracy

In [32]:
```
train_accuracy = accuracy_score(Y_train, dt_model.predict(X_train))
print("Training Accuracy:", train_accuracy)
```

```
Training Accuracy: 1.0
```

Display Test Accuracy

In [33]:
```
test_accuracy = accuracy_score(Y_test, Y_pred)
print("Test Accuracy:", test_accuracy)
```

```
Test Accuracy: 0.696969696969697
```

Confusion Matrix

In [34]:
```
from sklearn.metrics import confusion_matrix
```

In [35]:
```
cm = confusion_matrix(Y_test, Y_pred)
print("Confusion Matrix:")
print(cm)
```

```
Confusion Matrix:
[[113  38]
 [ 32  48]]
```

In [36]:
```
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['No Diabetes', 'Diabetes'],
            yticklabels=['No Diabetes', 'Diabetes'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

Save Model State

In [37]:
```
import pickle
```

In [38]:
```
with open('model.pkl','wb') as file:
    pickle.dump(model,file)
```

Load Model from Disk

In [39]:
```
with open('model.pkl','rb') as file:
    loaded_model = pickle.load(file)
```

In [40]:
```
y_predict = loaded_model.predict(x_test)
```

In [41]:
```
from sklearn.metrics import classification_report
```

In [42]:
```
print(classification_report(y_test, y_predict))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      1.00      1.00        13
           2       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```

In [ ]: