Decision Trees #CodeDaniel

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import plot_confusion_matrix
%matplotlib inline
```

These are Python library imports commonly used for data analysis and visualization:

numpy is a library for scientific computing in Python that provides support for large, multi-dimensional arrays and matrices, as well as a large collection of high-level mathematical functions to operate on these arrays.

** pandas** is a library that provides data structures for efficient data manipulation and analysis. It allows you to read, write, and manipulate tabular data, such as CSV files or SQL tables.

matplotlib is a plotting library for Python that provides a variety of visualization tools for creating static, animated, and interactive visualizations in Python.

seaborn is a data visualization library based on matplotlib that provides a high-level interface for creating attractive and informative statistical graphics.

sklearn.metrics.plot_confusion_matrix is a function from the scikit-learn library that allows you to plot a confusion matrix, which is a table that summarizes the performance of a classification algorithm. It is commonly used to evaluate the performance of a machine learning model.

%matplotlib inline is a magic command that allows you to display plots inline in Jupyter notebooks or IPython. It is not necessary when using matplotlib in other environments, such as scripts or the Python shell.

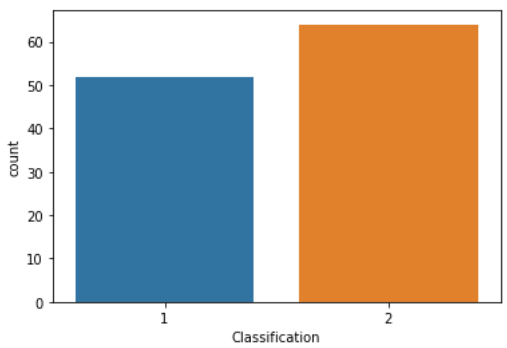
```
datapath = 'https://raw.githubusercontent.com/marcopeix/datasciencewithmarco/master/data/breastCancer.csv'
data = pd.read_csv(datapath)
```

Importing the Dataset

```
data.head()
```

	Age	BMI	Glucose	Insulin	HOMA	Leptin	Adiponectin	Resistin	MCP.1	Classification
0	48	23.500000	70	2.707	0.467409	8.8071	9.702400	7.99585	417.114	1
1	83	20.690495	92	3.115	0.706897	8.8438	5.429285	4.06405	468.786	1
2	82	23.124670	91	4.498	1.009651	17.9393	22.432040	9.27715	554.697	1
3	68	21.367521	77	3.226	0.612725	9.8827	7.169560	12.76600	928.220	1
4	86	21.111111	92	3.549	0.805386	6.6994	4.819240	10.57635	773.920	1

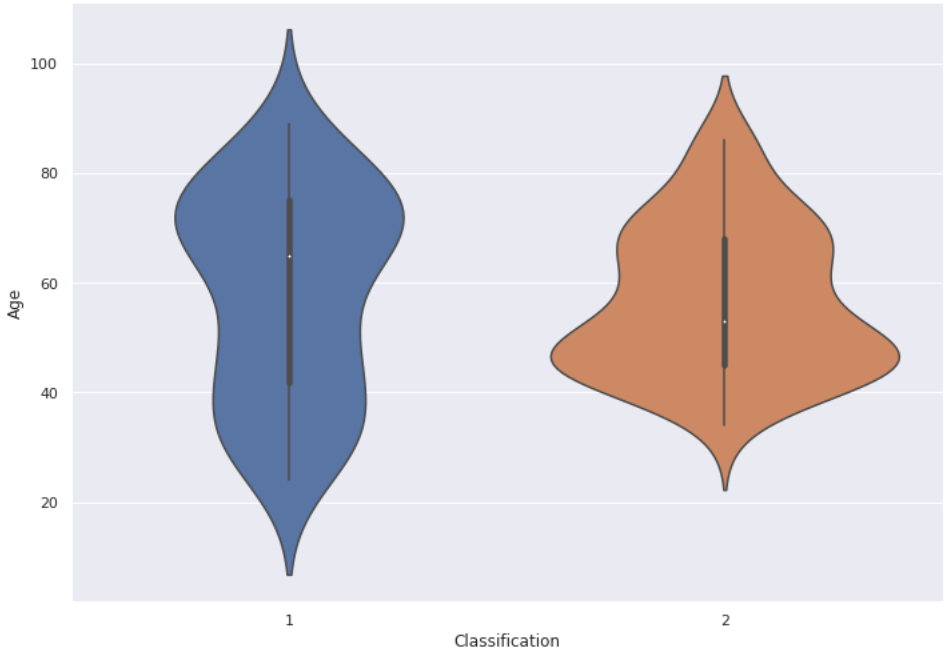
```
x = data['Classification']
ax = sns.countplot(x=x, data=data)
```

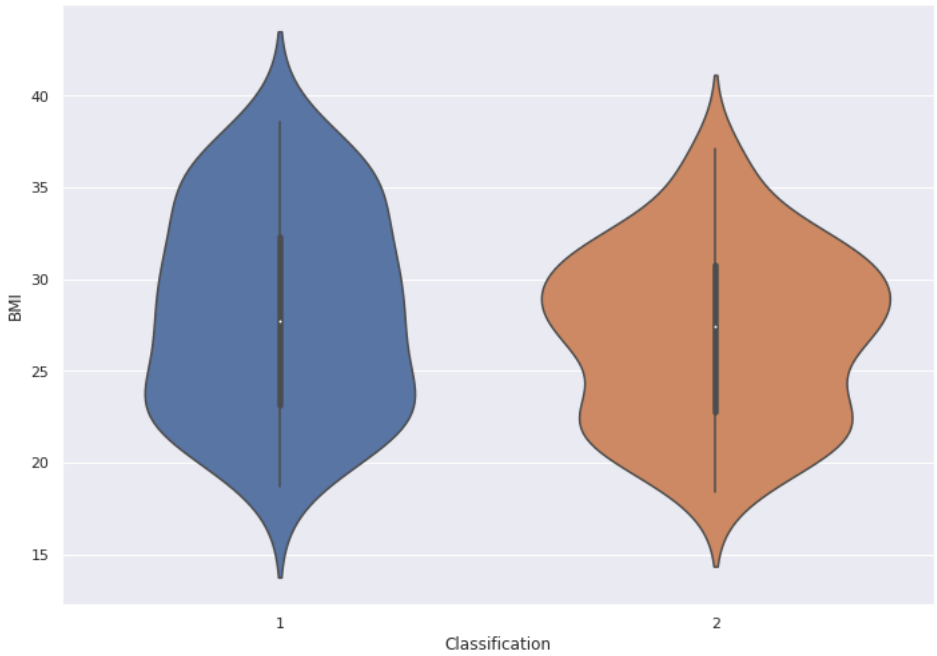


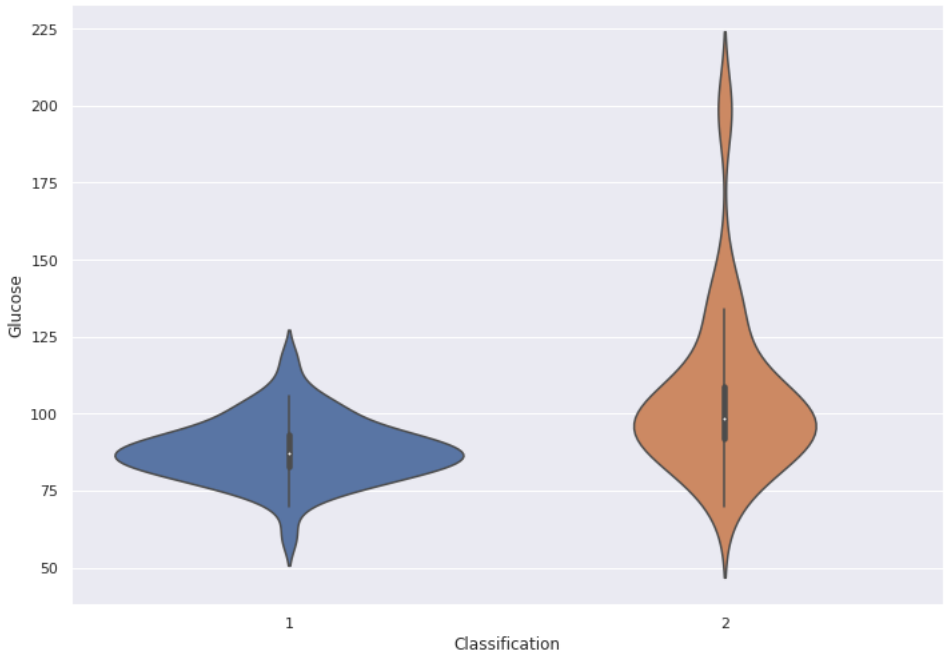
This is a Python code snippet that creates a countplot using the seaborn library: x is a string that represents the column name of the variable to be counted in the dataset. data is a pandas dataframe that contains the dataset

The code creates a seaborn countplot using sns.countplot(x=x, data=data). It sets the x parameter to the x variable and the data parameter to the data dataframe. A countplot is a type of barplot that displays the number of occurrences of each unique value in a categorical variable. It is useful for visualizing the distribution of data across different categories. The resulting plot will show a bar for each unique value of x, representing the number of occurrences of that value in the dataset.

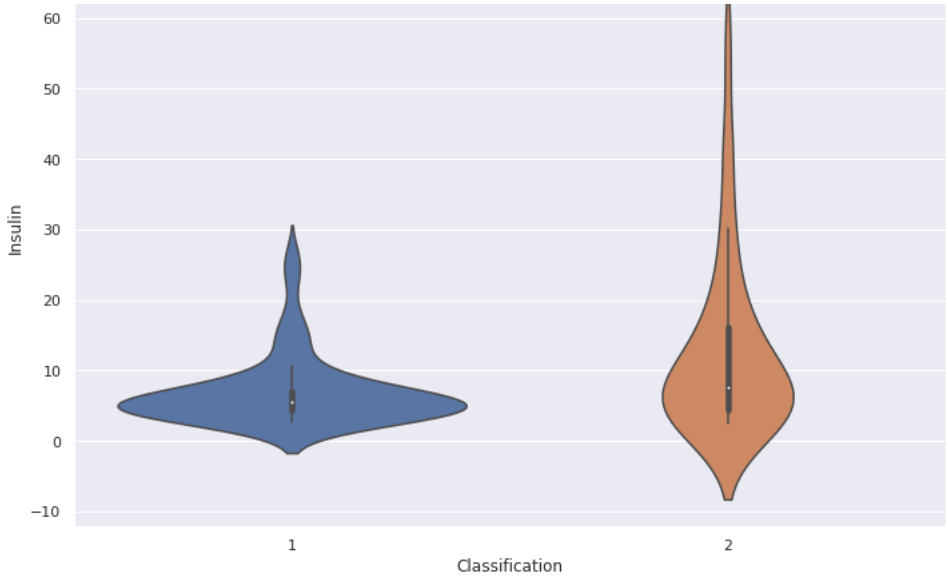
```
def violin_plots (x, y, data):
  for i, col in enumerate(y):
    plt.figure(i)
    sns.set(rc={"figure.figsize":(11.7,8.27)})
    ax = sns.violinplot(x=x, y=col, data=data)

y = data.columns[:-1]
x = data.columns[-1]

violin_plots(x, y, data)
```


100
80
60
40
20
Age
1
2
Classification


40
35
30
25
20
15
BMI
1
2
Classification


225
200
175
150
125
100
75
50
Glucose
1
2
Classification


70


60
50
40
30
20
10
0
−10
Insulin
1
2
Classification

This is a Python function that creates violin plots using the seaborn library:

x is a string that represents the column name of the independent variable in the dataset.

y is a list of strings that represents the column names of the dependent variables in the dataset.

data is a pandas dataframe that contains the dataset.

function iterates over the columns in y and creates a separate figure for each column. It sets the figure size using sns.set(rc={"figure.figsize": (11.7,8.27)}) and creates a seaborn violin plot using sns.violinplot(x=x, y=col, data=data), where col is the current dependent variable in the iteration. Finally, the function returns the created violin plots.

A violin plot is a type of data visualization that displays the distribution of data by creating a kernel density estimation of the data and displaying it as a shape similar to a violin

```
for col in data.columns:
  print(f"{col}: {data[col].isnull().sum()}")
```

```
Age: 0
BMI: 0
Glucose: 0
Insulin: 0
HOMA: 0
Leptin: 0
Adiponectin: 0
Resistin: 0
MCP.1: 0
Classification: 0
```

data is a pandas dataframe that contains the dataset.

The code iterates over the columns in the data dataframe using a for loop, and for each column, it prints the column name and the number of missing values in that column using the print function and an f-string. The number of missing values is computed using the isnull() method, which returns a boolean mask indicating whether each value in the column is missing or not, and the sum() method, which sums up the number of missing values in the column.

This code is useful for identifying columns with missing values in a dataset and assessing the extent of missingness for each column.

Preprocessing

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
data['Classification'] = le.fit_transform(data['Classification'])
data.head()
```

	Age	BMI	Glucose	Insulin	HOMA	Leptin	Adiponectin	Resistin	MCP.1	Classification
0	48	23.500000	70	2.707	0.467409	8.8071	9.702400	7.99585	417.114	0
1	83	20.690495	92	3.115	0.706897	8.8438	5.429285	4.06405	468.786	0
2	82	23.124670	91	4.498	1.009651	17.9393	22.432040	9.27715	554.697	0
3	68	21.367521	77	3.226	0.612725	9.8827	7.169560	12.76600	928.220	0
4	86	21.111111	92	3.549	0.805386	6.6994	4.819240	10.57635	773.920	0

LabelEncoder class from the scikit-learn library to encode a categorical variable in a pandas dataframe:

LabelEncoder is a class from the sklearn.preprocessing module that converts categorical variables into numeric labels.

le is an instance of the LabelEncoder class.

data is a pandas dataframe that contains the dataset.

'Classification' is a string that represents the column name of the categorical variable to be encoded in the data dataframe

The code applies the fit_transform method of the LabelEncoder object to the 'Classification' column of the data dataframe using the le.fit_transform(data['Classification']) statement. This method first fits the encoder to the unique values in the column and then transforms the column by replacing each unique value with a corresponding integer label.

The resulting encoded values are then stored back in the 'Classification' column of the data dataframe using the statement data['Classification'] = le.fit_transform(data['Classification']).

```
from sklearn.model_selection import train_test_split
y = data['Classification'].values.reshape(-1,1)
X = data.drop(['Classification'], axis = 1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1, random_state=42)
```

train_test_split function from the scikit-learn library to split a dataset into training and testing sets:

train_test_split is a function from the sklearn.model_selection module that splits a dataset into training and testing sets.

data is a pandas dataframe that contains the dataset.

'Classification' is a string that represents the column name of the target variable in the data dataframe.

y is a numpy array that contains the values of the target variable, reshaped into a 2D array using the reshape method. This is done to ensure that y has the right shape for use with scikit-learn models.

X is a pandas dataframe that contains the feature variables of the dataset, with the target variable column dropped using the drop method and the axis parameter set to 1 (i.e., column-wise).

test_size is a float that represents the proportion of the dataset to include in the testing set.

random_state is an integer that sets the seed for the random number generator used to split the dataset.

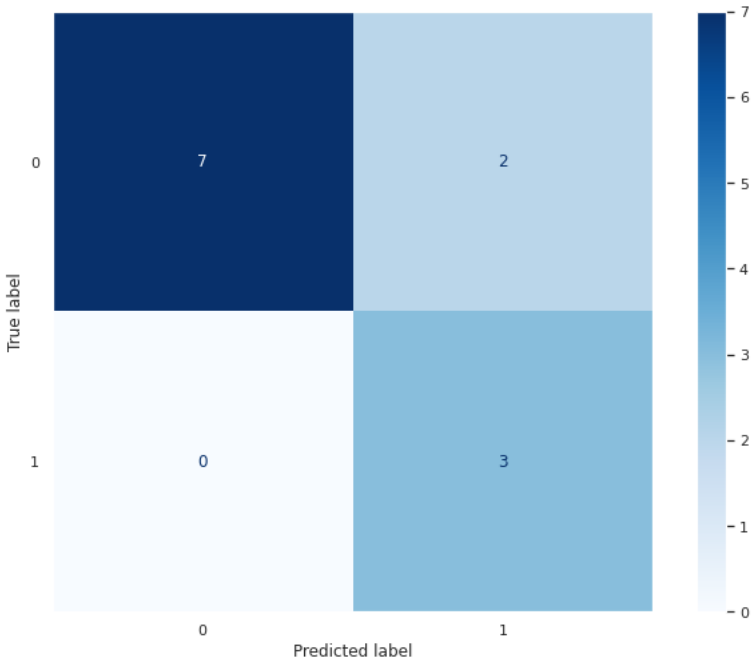
The code splits the dataset into training and testing sets using train_test_split(X, y, test_size = 0.1, random_state=42). It sets the X and y parameters to the feature and target variable arrays, respectively, and sets the test_size parameter to 0.1, indicating that 10% of the data should be used for testing. The resulting training and testing sets are stored in X_train, X_test, y_train, and y_test, respectively.

This code is useful for preparing a dataset for machine learning by splitting it into training and testing sets, which can be used to train and evaluate machine learning models, respectively.

Baseline Model

```
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)
plot_confusion_matrix(clf, X_test, y_test, cmap=plt.cm.Blues)
plt.grid(False)
plt.show()
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function plot_confusion_matrix is
  warnings.warn(msg, category=FutureWarning)
```



DecisionTreeClassifier class from the scikit-learn library to train a decision tree classifier on a training set and plot its confusion matrix on a testing set:

DecisionTreeClassifier is a class from the sklearn.tree module that implements a decision tree classifier.

clf is an instance of the DecisionTreeClassifier class.

X_train and y_train are numpy arrays that contain the feature and target variable values of the training set, respectively.

X_test and y_test are numpy arrays that contain the feature and target variable values of the testing set, respectively.

plot_confusion_matrix is a function from the sklearn.metrics module that plots a confusion matrix for a classifier on a given dataset.

cmap is a colormap that is used to color the cells of the confusion matrix.

plt.grid(False) turns off the grid lines in the plot

plt.show() displays the plot.

The code trains the decision tree classifier on the training set using clf.fit(X_train, y_train). The resulting trained classifier is then used to predict the target variable values for the testing set using clf.predict(X_test). The plot_confusion_matrix function is then used to plot the confusion matrix for the predicted target variable values and the true target variable values in the testing set.

This code is useful for training and evaluating a decision tree classifier on a dataset, and visualizing its performance using a confusion matrix.

```
from sklearn.tree import plot_tree
plot_tree(clf, max_depth=5)
```

```
[Text(0.4861111111111111, 0.9285714285714286, 'X[2] <= 91.5\ngini = 0.485\nsamples = 104\nvalue = [43, 61]'),
 Text(0.3055555555555556, 0.7857142857142857, 'X[0] <= 44.5\ngini = 0.454\nsamples = 43\nvalue = [28, 15]'),
 Text(0.25, 0.6428571428571429, 'gini = 0.0\nsamples = 13\nvalue = [13, 0]'),
 Text(0.3611111111111111, 0.6428571428571429, 'X[7] <= 14.227\ngini = 0.5\nsamples = 30\nvalue = [15, 15]'),
 Text(0.2222222222222222, 0.5, 'X[0] <= 46.5\ngini = 0.36\nsamples = 17\nvalue = [13, 4]'),
 Text(0.1111111111111111, 0.35714285714285715, 'X[5] <= 26.425\ngini = 0.444\nsamples = 3\nvalue = [1, 2]'),
 Text(0.05555555555555555, 0.21428571428571427, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
 Text(0.16666666666666666, 0.21428571428571427, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.3333333333333333, 0.35714285714285715, 'X[7] <= 3.295\ngini = 0.245\nsamples = 14\nvalue = [12, 2]'),
 Text(0.2777777777777778, 0.21428571428571427, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.3888888888888889, 0.21428571428571427, 'X[5] <= 8.747\ngini = 0.142\nsamples = 13\nvalue = [12, 1]'),
 Text(0.3333333333333333, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.4444444444444444, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.5, 0.5, 'X[1] <= 31.124\ngini = 0.26\nsamples = 13\nvalue = [2, 11]'),
 Text(0.4444444444444444, 0.35714285714285715, 'gini = 0.0\nsamples = 11\nvalue = [0, 11]'),
 Text(0.5555555555555556, 0.35714285714285715, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
 Text(0.6666666666666666, 0.7857142857142857, 'X[5] <= 7.24\ngini = 0.371\nsamples = 61\nvalue = [15, 46]'),
 Text(0.6111111111111112, 0.6428571428571429, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
 Text(0.7222222222222222, 0.6428571428571429, 'X[0] <= 52.0\ngini = 0.328\nsamples = 58\nvalue = [12, 46]'),
 Text(0.6666666666666666, 0.5, 'gini = 0.0\nsamples = 23\nvalue = [0, 23]'),
 Text(0.7777777777777778, 0.5, 'X[2] <= 103.5\ngini = 0.451\nsamples = 35\nvalue = [12, 23]'),
 Text(0.6666666666666666, 0.35714285714285715, 'X[8] <= 292.096\ngini = 0.5\nsamples = 22\nvalue = [11, 11]'),
 Text(0.6111111111111112, 0.21428571428571427, 'gini = 0.0\nsamples = 6\nvalue = [6, 0]'),
 Text(0.7222222222222222, 0.21428571428571427, 'X[1] <= 35.525\ngini = 0.43\nsamples = 16\nvalue = [5, 11]'),
 Text(0.6666666666666666, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.7777777777777778, 0.07142857142857142, '\n  (...)  \n'),
 Text(0.8888888888888888, 0.35714285714285715, 'X[8] <= 67.966\ngini = 0.142\nsamples = 13\nvalue = [1, 12]'),
 Text(0.8333333333333334, 0.21428571428571427, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.9444444444444444, 0.21428571428571427, 'gini = 0.0\nsamples = 12\nvalue = [0, 12]')]
```

plot_tree function from the scikit-learn library to visualize a decision tree classifier:

plot_tree is a function from the sklearn.tree module that plots a decision tree classifier.

clf is an instance of the DecisionTreeClassifier class that has been trained on a dataset.

max_depth is an integer that specifies the maximum depth of the tree to be plotted

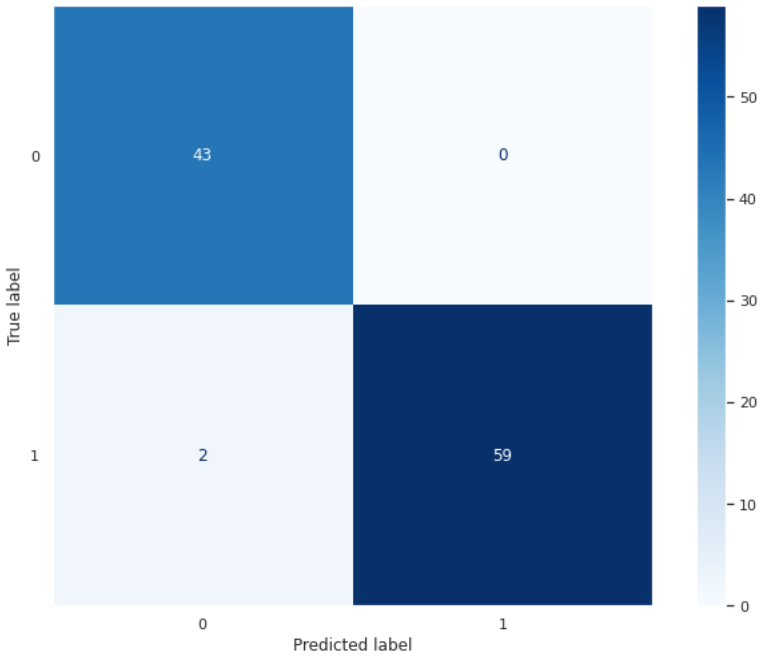
The code uses the plot_tree function to plot the decision tree classifier clf using a maximum depth of 5. The resulting plot shows the decision tree, with each node representing a decision based on a feature variable, and each leaf node representing a predicted target variable value. The plot also displays the impurity score and number of samples at each node, as well as the feature variable and threshold used to make the decision at each non-leaf node.

This code is useful for visualizing the decision-making process of a decision tree classifier, and can help to identify patterns and insights in the data that the classifier is using to make its predictions.

Bagging

```
from sklearn.ensemble import BaggingClassifier
bagging_clf = BaggingClassifier()
bagging_clf.fit(X_train,y_train.ravel())
plot_confusion_matrix(bagging_clf, X_train, y_train, cmap=plt.cm.Blues)
plt.grid(False)
plt.show()
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function plot_confusion_matrix is
  warnings.warn(msg, category=FutureWarning)
```



BaggingClassifier class from the scikit-learn library to train a bagging classifier on a training set and plot its confusion matrix on the same training set:

BaggingClassifier is a class from the sklearn.ensemble module that implements a bagging classifier

bagging_clf is an instance of the BaggingClassifier class

plot_confusion_matrix is a function from the sklearn.metrics module that plots a confusion matrix for a classifier on a given dataset.

cmap is a colormap that is used to color the cells of the confusion matrix.

plt.grid(False) turns off the grid lines in the plot.

plt.show() displays the plot.

he code trains the bagging classifier on the training set using bagging_clf.fit(X_train, y_train.ravel()).

The resulting trained classifier is then used to predict the target variable values for the training set using bagging_clf.predict(X_train).

The plot_confusion_matrix function is then used to plot the confusion matrix for the predicted target variable values and the true target variable values in the training set

This code is useful for evaluating the performance of a bagging classifier on a training set, and visualizing its performance using a confusion matrix. However, it is important to note that using a bagging classifier can lead to overfitting if not properly tuned, so it is recommended to use cross-validation and hyperparameter tuning to ensure optimal performance.

Random Forest Classifier