# Decision Tree & Random Forest

## Imports

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

## Loading Dataset

In [2]:
```
from sklearn.datasets import load_iris
```

In [3]:
```
iris = load_iris()
```

In [4]:
```
type(iris)
```

Out[4]:
```
sklearn.utils.Bunch
```

## See all the keys

In [5]:
```
iris.keys()
```

Out[5]:
```
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'filenam
e'])
```

## Description

In [6]:
```
print(iris["DESCR"])
```

```
.. _iris_dataset:

Iris plants dataset
--------------------

**Data Set Characteristics:**

    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive attributes and the class
    :Attribute Information:
        - sepal length in cm
        - sepal width in cm
        - petal length in cm
        - petal width in cm
        - class:
                - Iris-Setosa
                - Iris-Versicolour
                - Iris-Virginica

    :Summary Statistics:

    ============== ==== ==== ======= ===== ====================
                    Min  Max   Mean    SD   Class Correlation
    ============== ==== ==== ======= ===== ====================
```

```
    sepal length:   4.3  7.9   5.84   0.83    0.7826
    sepal width:    2.0  4.4   3.05   0.43   -0.4194
    petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)
    petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)
    ============== ==== ==== ======= ===== ====================

    :Missing Attribute Values: None
    :Class Distribution: 33.3% for each of 3 classes.
    :Creator: R.A. Fisher
    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
    :Date: July, 1988

The famous Iris database, first used by Sir R.A. Fisher. The dataset is taken
from Fisher's paper. Note that it's the same as in R, but not as in the UCI
Machine Learning Repository, which has two wrong data points.

This is perhaps the best known database to be found in the
pattern recognition literature.  Fisher's paper is a classic in the field and
is referenced frequently to this day.  (See Duda & Hart, for example.)  The
data set contains 3 classes of 50 instances each, where each class refers to a
type of iris plant.  One class is linearly separable from the other 2; the
latter are NOT linearly separable from each other.

.. topic:: References

   - Fisher, R.A. "The use of multiple measurements in taxonomic problems"
     Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to
     Mathematical Statistics" (John Wiley, NY, 1950).
   - Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analysis.
     (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.
   - Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System
     Structure and Classification Rule for Recognition in Partially Exposed
     Environments".  IEEE Transactions on Pattern Analysis and Machine
     Intelligence, Vol. PAMI-2, No. 1, 67-71.
   - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".  IEEE Transactions
     on Information Theory, May 1972, 431-433.
   - See also: 1988 MLC Proceedings, 54-64.  Cheeseman et al"s AUTOCLASS II
     conceptual clustering system finds 3 classes in the data.
   - Many, many more ...
```

## Data

In [7]:
```
iris["data"][:5] # Independent columns
```

Out[7]:
```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2]])
```

In [8]:
```
iris["feature_names"] # Column names
```

Out[8]:
```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

In [9]:
```
np.array(iris["target"]) # Dependent column
```

Out[9]:
```
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, 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])
```

In [10]:
```
iris["target_names"]
# So, 0 is setosa, 1 is versicolor and 2 is virginica
```

Out[10]:
```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

## Create Dataframe

In [11]:
```
df = pd.DataFrame(data=iris["data"], columns=iris["feature_names"])
df["target"] = iris["target"]
df.head()
```

Out[11]:

| | sepal length (cm) | sepal width (cm) | petal length (cm) | petal width (cm) | target |
|---|---|---|---|---|---|
| **0** | 5.1 | 3.5 | 1.4 | 0.2 | 0 |
| **1** | 4.9 | 3.0 | 1.4 | 0.2 | 0 |
| **2** | 4.7 | 3.2 | 1.3 | 0.2 | 0 |
| **3** | 4.6 | 3.1 | 1.5 | 0.2 | 0 |
| **4** | 5.0 | 3.6 | 1.4 | 0.2 | 0 |

In [12]:
```
df["target"].unique()
```

Out[12]:
```
array([0, 1, 2])
```

In [13]:
```
df["target"].replace(range(3), iris["target_names"], inplace=True)
```

In [14]:
```
df.head()
```

Out[14]:

| | sepal length (cm) | sepal width (cm) | petal length (cm) | petal width (cm) | target |
|---|---|---|---|---|---|
| **0** | 5.1 | 3.5 | 1.4 | 0.2 | setosa |
| **1** | 4.9 | 3.0 | 1.4 | 0.2 | setosa |
| **2** | 4.7 | 3.2 | 1.3 | 0.2 | setosa |
| **3** | 4.6 | 3.1 | 1.5 | 0.2 | setosa |
| **4** | 5.0 | 3.6 | 1.4 | 0.2 | setosa |

In [15]:
```
df["target"].unique()
```

Out[15]:
```
array(['setosa', 'versicolor', 'virginica'], dtype=object)
```

## EDA

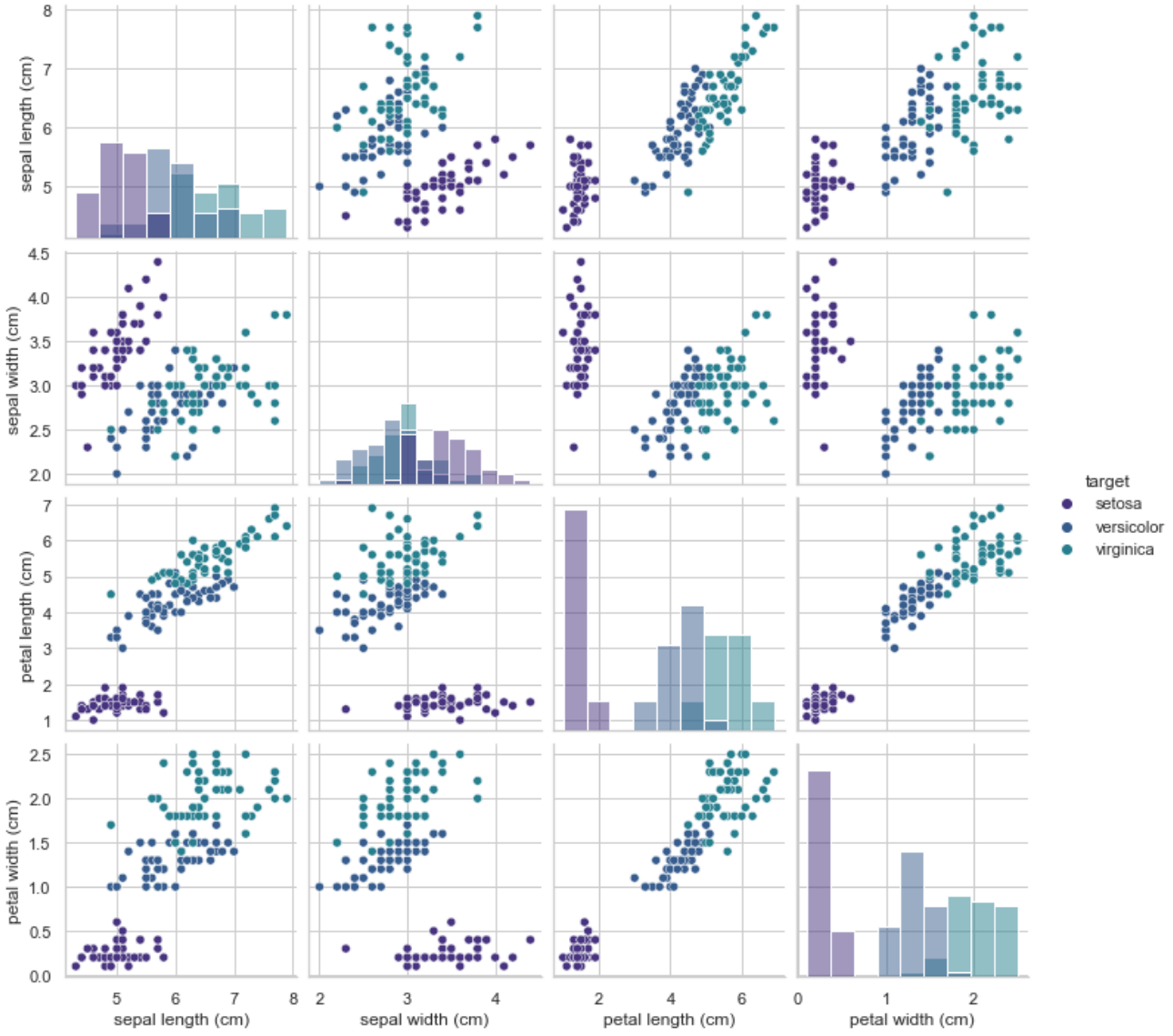
In [16]:
```
sns.set(style="whitegrid", palette="viridis")
```

In [17]:
```

```

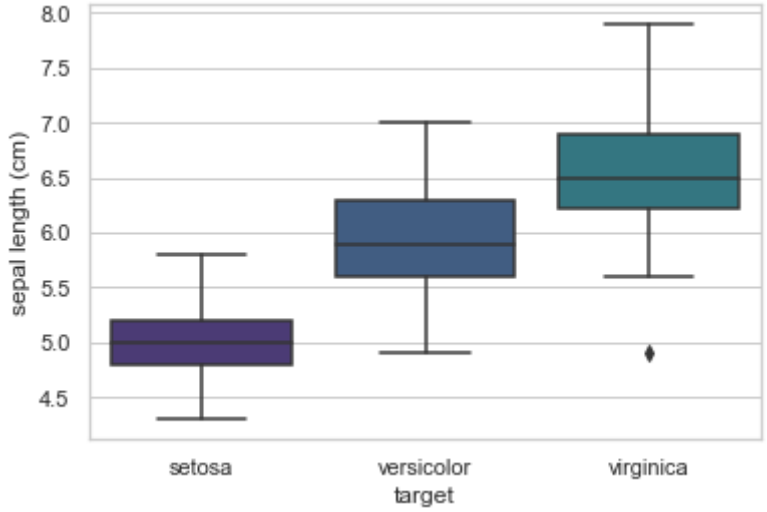
```
sns.pairplot(df, hue="target", diag_kind='hist')
```

Out[17]: <seaborn.axisgrid.PairGrid at 0x179a3666f10>



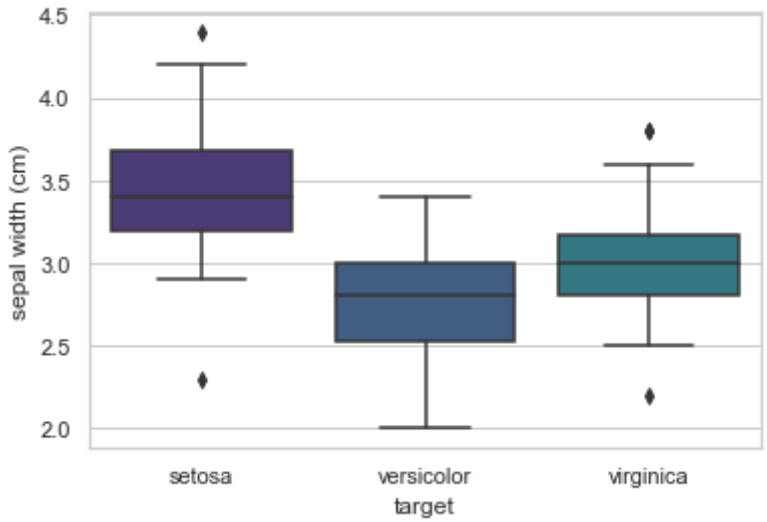

In [18]:
```
sns.boxplot(x="target", y="sepal length (cm)", data=df)
```

Out[18]: <AxesSubplot:xlabel='target', ylabel='sepal length (cm)'>



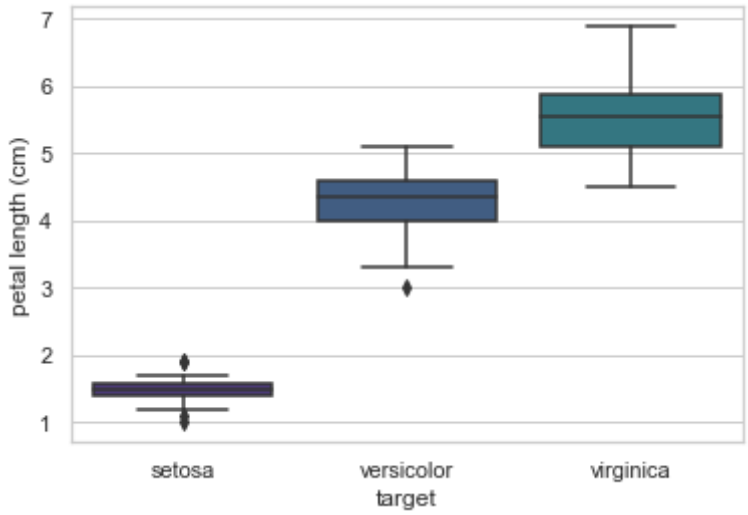

In [19]:
```
sns.boxplot(x="target", y="sepal width (cm)", data=df)
```

Out[19]: <AxesSubplot:xlabel='target', ylabel='sepal width (cm)'>



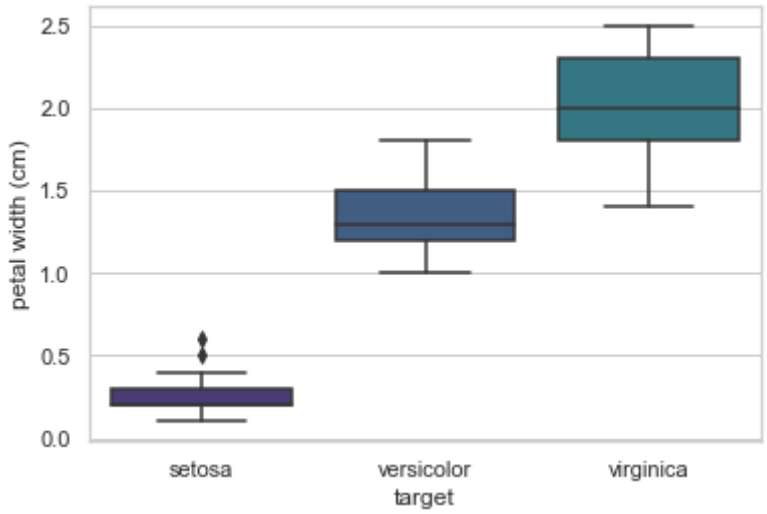

In [20]:
```
sns.boxplot(x="target", y="petal length (cm)", data=df)
```

Out[20]: <AxesSubplot:xlabel='target', ylabel='petal length (cm)'>




In [21]:
```
sns.boxplot(x="target", y="petal width (cm)", data=df)
```

Out[21]: <AxesSubplot:xlabel='target', ylabel='petal width (cm)'>



## Feature Selection

In [22]:
```
df.head()
```

Out[22]:

| | sepal length (cm) | sepal width (cm) | petal length (cm) | petal width (cm) | target |
|---|---|---|---|---|---|
| **0** | 5.1 | 3.5 | 1.4 | 0.2 | setosa |
| **1** | 4.9 | 3.0 | 1.4 | 0.2 | setosa |
| **2** | 4.7 | 3.2 | 1.3 | 0.2 | setosa |
| **3** | 4.6 | 3.1 | 1.5 | 0.2 | setosa |
| **4** | 5.0 | 3.6 | 1.4 | 0.2 | setosa |

In [23]:
```
X = df.drop("target", axis=1) # Independent
y = df["target"] # Dependent
```

## Train Test Split

In [24]:
```
from sklearn.model_selection import train_test_split
```

In [25]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=202
```

In [26]:
```
X_train[:5]
```

Out[26]:

| | sepal length (cm) | sepal width (cm) | petal length (cm) | petal width (cm) |
|---|---|---|---|---|
| **120** | 6.9 | 3.2 | 5.7 | 2.3 |
| **117** | 7.7 | 3.8 | 6.7 | 2.2 |
| **76** | 6.8 | 2.8 | 4.8 | 1.4 |
| **86** | 6.7 | 3.1 | 4.7 | 1.5 |
| **57** | 4.9 | 2.4 | 3.3 | 1.0 |

In [27]:
```
y_train[:5]
```

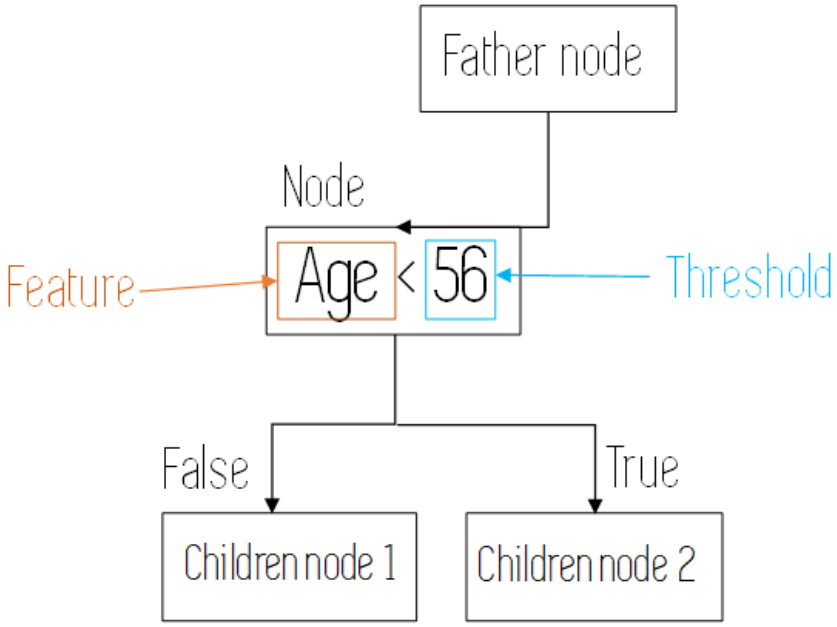
Out[27]:
```
120     virginica
117     virginica
76     versicolor
86     versicolor
57     versicolor
Name: target, dtype: object
```

## Decision Tree

In the Machine Learning world, Decision Trees are a kind of non parametric models, that can be used for both classification and regression.

This means that Decision trees are flexible models that don't increase their number of parameters as we add more features, and they can either output a categorical prediction or a numerical prediction.

They are constructed using two kinds of elements: nodes and branches. At each node, one of the features of our data is evaluated in order to split the observations in the training process or to make an specific data point follow a certain path when making a prediction. When they are being built decision trees are constructed by recursively evaluating different features and using at each node the feature that best splits the data.



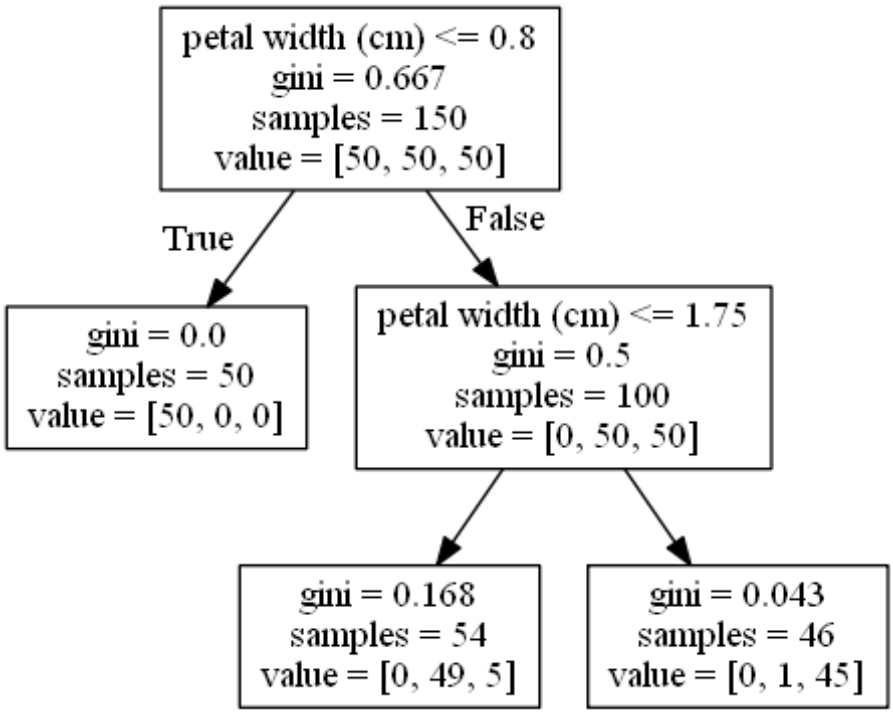

There are three types of nodes in a decision tree:

- **The Root Node:** It is the node that starts the graph. In a normal decision tree it evaluates the variable that best splits the data.
- **Intermediate nodes:** These are nodes where variables are evaluated but which are not the final nodes where predictions are made.
- **Leaf nodes:** These are the final nodes of the tree, where the predictions of a category or a numerical value are made.

## Training Process

Decision trees are built by recursively splitting our training samples using the features from the data that work best for the specific task. This is done by evaluating certain metrics, like the Gini index or the Entropy for categorical decision trees, or the Residual or Mean Squared Error for regression trees. We pick the variable/threshold combination that gives us the highest/lowest value for the specific metric that we are using for the resulting children nodes (the highest reduction or increase in the metric).

Given below is an example decision tree for the Iris Dataset that we are working on.




We can see that the root node starts with 50 samples of each of the three classes, and a Gini Index (lower Gini Index is better) of 0.667. In this node, the feature that best split the different classes of the data is the petal width in cm, using as a threshold a value of 0.8. This results in two nodes, one with Gini 0 (perfectly pure node that only has one of the types of flowers) and one with Gini of 0.5, where the two other kinds of flowers are grouped.

In this intermediate node, the same feature is evaluated using a threshold of 1.75. Now this results in two other children nodes that are not pure, but that have a pretty low Gini Index. In all of these nodes all the other features of the data (sepal length, sepal width, and petal length) were evaluated, and had their resulting Gini Index calculated, however, the feature that gave us the best results (lowest Gini Index) was the petal width. The reason the tree didn't continue growing is because Decision Trees always a growth-stop condition configured, otherwise they would grow until each training sample was separated into its own leaf node. These stop conditions are maximum depth of the tree, minimum samples in leaf nodes, or minimum reduction in the error metric.

## Gini Impurity

Gini Impurity is a measurement of the likelihood of an incorrect classification of a new instance of a random variable, if that new instance were randomly classified according to the distribution of class labels from the data set. If our dataset is Pure then likelihood of incorrect classification is 0. If our sample is mixture of different classes then likelihood of incorrect classification will be high.

**Pure:** Pure means, in a selected sample of dataset all data belongs to same class.
**Impure:** Impure means, data is mixture of different classes.

If we have C total classes and p(i) is the probability of picking a datapoint with class i, then the Gini Impurity is calculated as follows.

$$G = \sum_{i=1}^{C} p(i) * (1 - p(i))$$

### Pruning

The performance of a tree can be further increased by pruning. It involves removing the branches that make use of features having low importance. This way, we reduce the complexity of tree, and thus increasing its predictive power by reducing overfitting. Pruning can start at either root or the leaves. The simplest method of pruning starts at leaves and removes each node with most popular class in that leaf, this change is kept if it doesn't deteriorate accuracy. Its also called reduced error pruning. More sophisticated pruning methods can be used such as cost complexity pruning where a learning parameter (alpha) is used to weigh whether nodes can be removed based on the size of the sub-tree. This is also known as weakest link pruning.

### Advantage of Decision Tree

- Easy to use and understand.
- Can handle both categorical and numerical data.
- Resistant to outliers, hence require little data preprocessing.

### Disadvantage of Decision Tree

- Prone to overfitting.
- Require some kind of measurement as to how well they are doing.
- Need to be careful with parameter tuning.
- Can create biased learned trees if some classes dominate.

Article links: Link 1, Link 2, Link 3

In [28]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [29]:
```
from sklearn.metrics import classification_report, confusion_matrix
```

In [30]:
```
dtree_model = DecisionTreeClassifier(max_depth=10)
```

In [31]:
```
dtree_model.fit(X_train, y_train) # Computationally expensive
```

Out[31]:
```
DecisionTreeClassifier(max_depth=10)
```

In [32]:
```
dtree_pred = dtree_model.predict(X_test) # Not much computation required
```

In [33]:
```
print(confusion_matrix(y_test, dtree_pred))
```

```
[[18  0  0]
 [ 0 11  2]
```

```
 [ 0  2 12]]
```

In [34]:
```
print(classification_report(y_test, dtree_pred))
```

```
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        18
  versicolor       0.85      0.85      0.85        13
   virginica       0.86      0.86      0.86        14

    accuracy                           0.91        45
   macro avg       0.90      0.90      0.90        45
weighted avg       0.91      0.91      0.91        45
```

**Structure of all sklearn models:**

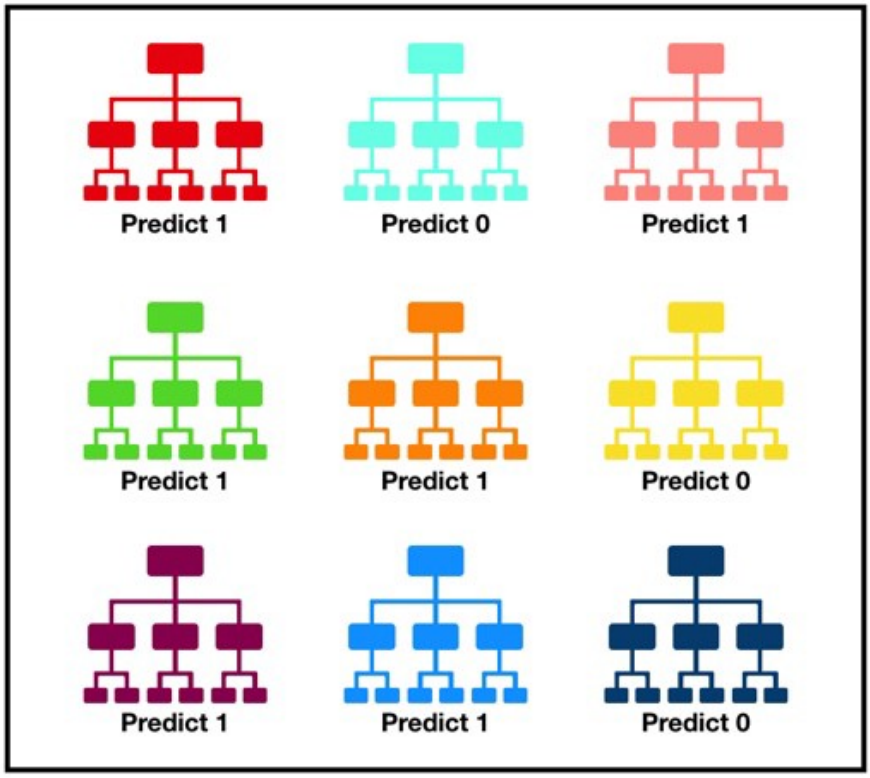
```
from sklearn.library_name import ModelName
from sklearn.metrics import confusion_matrix, classification_report

model = ModelName()
model.fit(X_train, y_train)
pred = model.predict(X_test)
print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))
```

So you can easlity create a function that can take one or more models as input and provide some output.

## Random Forest

Random forest, like its name implies, consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction.




The fundamental concept behind random forest is a simple but powerful one: the wisdom of crowds. In data science speak, the reason that the random forest model works so well is that a large number of relatively uncorrelated models (trees) operating as a committee will outperform any of the individual constituent models. The reason for this wonderful effect is that the trees protect each other from their individual errors. While some trees may be wrong, many other trees will be right, so as a group the trees are able to move in the correct direction.

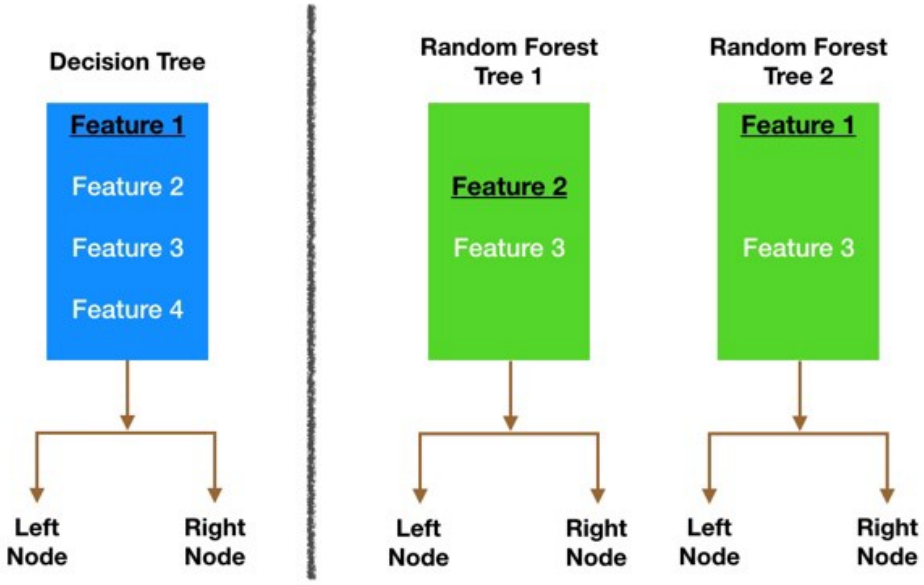
What do we need in order for our random forest to make accurate class predictions?

- We need features that have at least some predictive power. After all, if we put garbage in then we will get garbage out.
- The trees of the forest and more importantly their predictions need to be uncorrelated (or at least have low correlations with each other). While the algorithm itself via feature randomness tries to engineer these low correlations for us, the features we select and the hyper-parameters we choose will impact the ultimate correlations as well.

## Ensuring that the Models Diversify Each Other

- **Bagging (Bootstrap Aggregation):** Decisions trees are very sensitive to the data they are trained on. Small changes to the training set can result in significantly different tree structures. Random forest takes advantage of this by allowing each individual tree to randomly sample from the dataset with replacement, resulting in different trees. This process is known as bagging. We are not subsetting the training data into smaller chunks and training each tree on a different chunk. Rather, if we have a sample of size N, we are still feeding each tree a training set of size N.
- **Feature Randomness:** In a normal decision tree, when it is time to split a node, we consider every possible feature and pick the one that produces the most separation between the observations in the left node vs. those in the right node. In contrast, each tree in a random forest can pick only

from a random subset of features. This forces even more variation amongst the trees in the model and ultimately results in lower correlation across trees and more diversification.




Article links: Link 1

In [35]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [36]:
```
rfc_model = RandomForestClassifier(n_estimators=500)
# n_estimators -> No. of trees in the forest
```

In [37]:
```
rfc_model.fit(X_train, y_train)
```

Out[37]:
```
RandomForestClassifier(n_estimators=500)
```

In [38]:
```
rfc_pred = rfc_model.predict(X_test)
```

In [39]:
```
print(confusion_matrix(y_test, rfc_pred))
```

```
[[18  0  0]
 [ 0 11  2]
 [ 0  3 11]]
```

In [40]:
```
print(classification_report(y_test, rfc_pred))
```

```
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        18
  versicolor       0.79      0.85      0.81        13
   virginica       0.85      0.79      0.81        14

    accuracy                           0.89        45
   macro avg       0.88      0.88      0.88        45
weighted avg       0.89      0.89      0.89        45
```

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

```
print(accuracy_score(y_test, rfc_pred))
```

```
0.8888888888888888
```