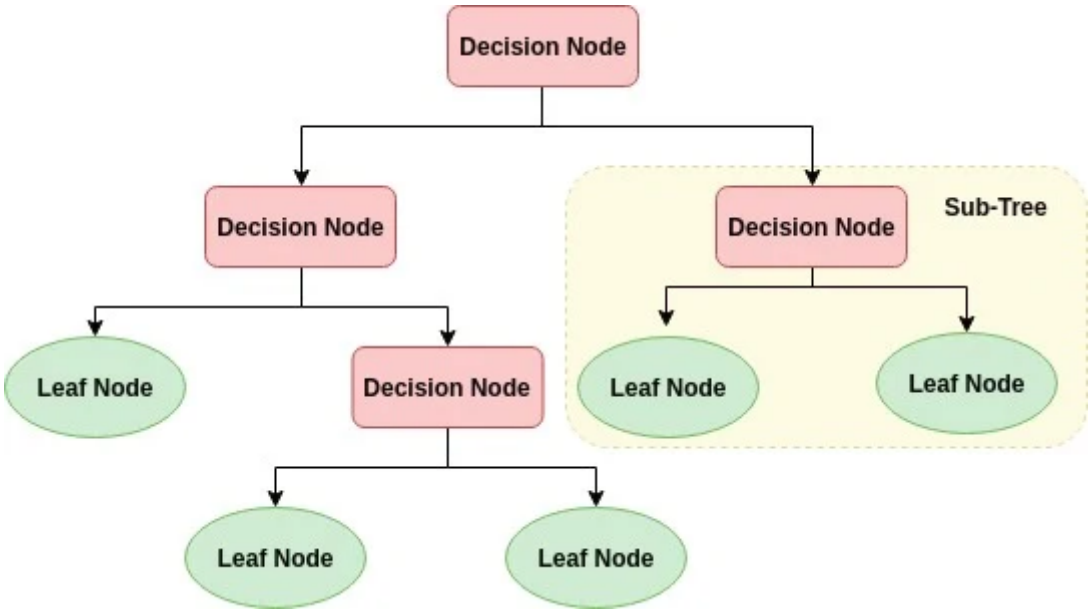
Decision Tree Machine Learning in Python

For detailed theory read A Introduction to Statistical Learning

http://faculty.marshall.usc.edu/gareth-james/ISL/ (http://faculty.marshall.usc.edu/gareth-james/ISL/)

A decision tree is a flowchart-like tree structure where an internal node represents feature, the branch represents a decision rule, and each leaf node represents the outcome. The topmost node in a decision tree is known as the root node. It learns to partition on the basis of the attribute value. It partitions the tree in recursively manner call recursive partitioning. This flowchart-like structure helps you in decision making. It's visualization like a flowchart diagram which easily mimics the human level thinking. That is why decision trees are easy to understand and interpret.



Example

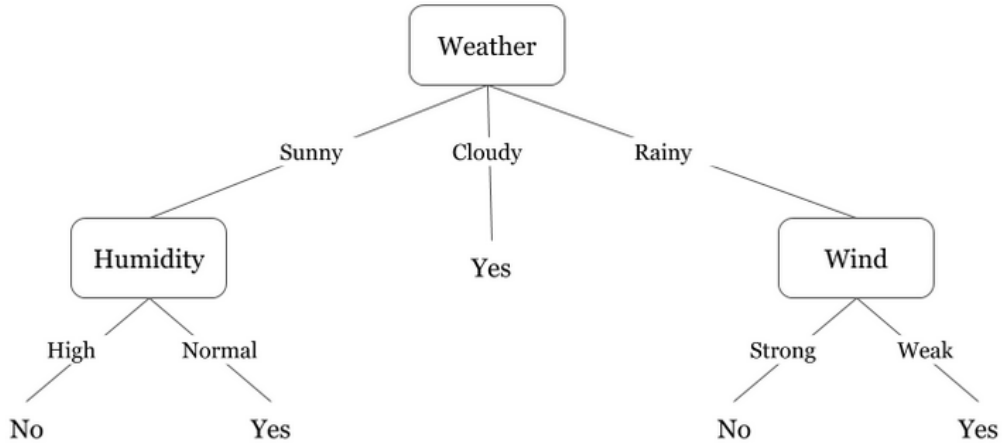
Day	Weather	Temperature	Humidity	Wind	Play?
1	Sunny	Hot	High	Weak	No
2	Cloudy	Hot	High	Weak	Yes
3	Sunny	Mild	Normal	Strong	Yes
4	Cloudy	Mild	High	Strong	Yes
5	Rainy	Mild	High	Strong	No
6	Rainy	Cool	Normal	Strong	No
7	Rainy	Mild	High	Weak	Yes
8	Sunny	Hot	High	Strong	No
9	Cloudy	Hot	Normal	Weak	Yes
10	Rainy	Mild	High	Strong	No

Why Decision Tree

- Decision tress often mimic the human level thinking so its so simple to understand the data and make some good interpretations.
- Decision trees actually make you see the logic for the data to interpret(not like black box algorithms like SVM,NN,etc..)

How Decision Tree Works

- Select the best attribute using Attribute Selection Measures(ASM) to split the records.
- Make that attribute a decision node and breaks the dataset into smaller subsets.
- Starts tree building by repeating this process recursively for each child until one of the condition will match:
  - All the tuples belong to the same attribute value.
  - There are no more remaining attributes.
  - There are no more instances.



here couple of algorithms to build a decision tree , we only talk about a few which are

```
CART (Classification and Regression Trees) → uses Gini Index(Classification) as me
tric.
ID3 (Iterative Dichotomiser 3) → uses Entropy function and Information gain as met
rics.
```

Decision Making in DT with Attribute Selection Measures(ASM)

- Information Gain
- Gain Ratio
- Gini Index

Read Chapter 8: http://faculty.marshall.usc.edu/gareth-james/ISL/ (http://faculty.marshall.usc.edu/gareth-james/ISL/)

Information Gain

In order to define information gain precisely, we begin by defining a measure commonly used in information theory, called entropy that characterizes the (im)purity of an arbitrary collection of examples."

Entropy

Entropy $H(S)$ is a measure of the amount of uncertainty in the (data) set S (i.e. entropy characterizes the (data) set S).

$$H(S) = \sum_{c \in C} -p(c) \log_2 p(c)$$

Where,

- S – The current (data) set for which entropy is being calculated (changes every iteration of the ID3 algorithm)
- C – Set of classes in S C={ yes, no }
- $p(c)$ – The proportion of the number of elements in class c to the number of elements in set S

When $H(S) = 0$, the set S is perfectly classified (i.e. all elements in S are of the same class).

In ID3, entropy is calculated for each remaining attribute. The attribute with the **smallest** entropy is used to split the set S on this iteration. The higher the entropy, the higher the potential to improve the classification here.

Information gain

Information gain $IG(A)$ is the measure of the difference in entropy from before to after the set S is split on an attribute A. In other words, how much uncertainty in S was reduced after splitting set S on attribute A.

$$IG(A, S) = H(S) - \sum_{t \in T} p(t) H(t)$$

Where,

- $H(S)$ – Entropy of set S
- T – The subsets created from splitting set S by attribute A such that $S = \bigcup_{t \in T} t$
- $p(t)$ – The proportion of the number of elements in t to the number of elements in set S
- $H(t)$ – Entropy of subset t

In ID3, information gain can be calculated (instead of entropy) for each remaining attribute. The attribute with the **largest** information gain is used to split the set S on this iteration.

- 1.compute the entropy for data-set
- 2.for every feature:
  - 1.calculate entropy for all categorical values
  - 2.take average information entropy for the current attribute
  - 3.calculate gain for the current attribute3. pick the highest gain attribute.
- 4. Repeat until we get the tree we desired.

Gain Ratio

An alternative measure to information gain is gain ratio (Quinlan 1986). Gain ratio tries to the correct the information gain's bias towards attributes with many possible values by adding a denominator to information gain called split information. Split Information tries to measure how broadly and uniformly the attribute splits the data:

$$SplitInformation(S, A) = -\sum_{i=1}^{c} rac|S_i||S| \cdot log_2 rac|S_i||S|$$

The Gain Ratio is defined in terms of Gain and SplitInformation as,

$$GainRatio(S, A) \equiv racGain(S, A) SplitInformation(S, A)$$

Gini Index

Gini Index 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.

Optimizing DT

```
criterion : optional (default="gini") or Choose attribute selection measure: This
 parameter allows us to use the different-different attribute selection measure. S
upported criteria are "gini" for the Gini index and "entropy" for the information
 gain.

splitter : string, optional (default="best") or Split Strategy: This parameter all
ows us to choose the split strategy. Supported strategies are "best" to choose the
best split and "random" to choose the best random split.

max_depth : int or None, optional (default=None) or Maximum Depth of a Tree: The m
aximum depth of the tree. If None, then nodes are expanded until all the leaves co
ntain less than min_samples_split samples. The higher value of maximum depth cause
s overfitting, and a lower value causes underfitting (Source).
```

Recursive Binary Splitting

In this procedure all the features are considered and different split points are tried and tested using a cost function. The split with the best cost (or lowest cost) is selected.

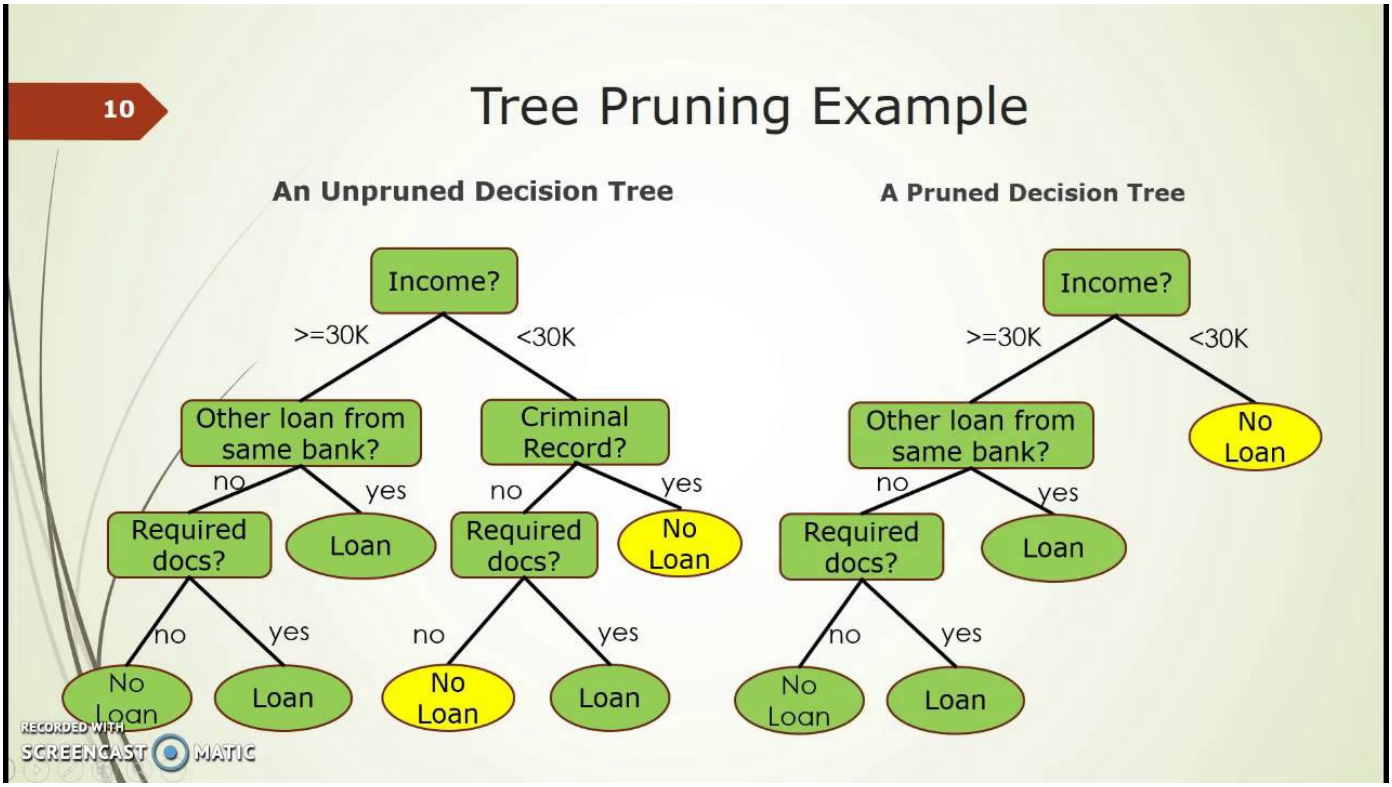
When to stop splitting?

You might ask when to stop growing a tree? As a problem usually has a large set of features, it results in large number of split, which in turn gives a huge tree. Such trees are complex and can lead to overfitting. So, we need to know when to stop?

set max_depth

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.



Decision Tree Regressor

Entrée [1]:

```
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

Entrée [2]:

```
from sklearn import datasets, metrics
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
```

Entrée [4]:

```
diabetes  = datasets.load_diabetes()
diabetes.keys()
```

Out[4]:

```
dict_keys(['data', 'target', 'DESCR', 'feature_names', 'data_filename', 'tar
get_filename'])
```

Entrée [5]:

```
diabetes.DESCR
```

Out[5]:

```
'.. _diabetes_dataset:\n\nDiabetes dataset\n----------------\n\nTen baseline
variables, age, sex, body mass index, average blood\npressure, and six blood
serum measurements were obtained for each of n =\n442 diabetes patients, as
well as the response of interest, a\nquantitative measure of disease progres
sion one year after baseline.\n\n**Data Set Characteristics:**\n\n  :Number
of Instances: 442\n\n  :Number of Attributes: First 10 columns are numeric p
redictive values\n\n  :Target: Column 11 is a quantitative measure of diseas
e progression one year after baseline\n\n  :Attribute Information:\n      -
Age\n      - Sex\n      - Body mass index\n      - Average blood pressure\n
- S1\n      - S2\n      - S3\n      - S4\n      - S5\n      - S6\n\nNote: Ea
ch of these 10 feature variables have been mean centered and scaled by the s
tandard deviation times `n_samples` (i.e. the sum of squares of each column
totals 1).\n\nSource URL:\nhttp://www4.stat.ncsu.edu/~boos/var.select/diabet
es.html\n\nFor more information see:\nBradley Efron, Trevor Hastie, Iain Joh
nstone and Robert Tibshirani (2004) "Least Angle Regression," Annals of Stat
istics (with discussion), 407-499.\n(http://web.stanford.edu/~hastie/Papers/
LARS/LeastAngle_2002.pdf)'
```

Entrée [6]:

```
diabetes.feature_names
```

Out[6]:

```
['age', 'sex', 'bmi', 'bp', 's1', 's2', 's3', 's4', 's5', 's6']
```

Entrée [7]:

```
diabetes.target
```

Out[7]:

```
array([151.,  75., 141., 206., 135.,  97., 138.,  63., 110., 310., 101.,
        69., 179., 185., 118., 171., 166., 144.,  97., 168.,  68.,  49.,
        68., 245., 184., 202., 137.,  85., 131., 283., 129.,  59., 341.,
        87.,  65., 102., 265., 276., 252.,  90., 100.,  55.,  61.,  92.,
       259.,  53., 190., 142.,  75., 142., 155., 225.,  59., 104., 182.,
       128.,  52.,  37., 170., 170.,  61., 144.,  52., 128.,  71., 163.,
       150.,  97., 160., 178.,  48., 270., 202., 111.,  85.,  42., 170.,
       200., 252., 113., 143.,  51.,  52., 210.,  65., 141.,  55., 134.,
        42., 111.,  98., 164.,  48.,  96.,  90., 162., 150., 279.,  92.,
        83., 128., 102., 302., 198.,  95.,  53., 134., 144., 232.,  81.,
       104.,  59., 246., 297., 258., 229., 275., 281., 179., 200., 200.,
       173., 180.,  84., 121., 161.,  99., 109., 115., 268., 274., 158.,
       107.,  83., 103., 272.,  85., 280., 336., 281., 118., 317., 235.,
        60., 174., 259., 178., 128.,  96., 126., 288.,  88., 292.,  71.,
       197., 186.,  25.,  84.,  96., 195.,  53., 217., 172., 131., 214.,
        59.,  70., 220., 268., 152.,  47.,  74., 295., 101., 151., 127.,
       237., 225.,  81., 151., 107.,  64., 138., 185., 265., 101., 137.,
       143., 141.,  79., 292., 178.,  91., 116.,  86., 122.,  72., 129.,
       142.,  90., 158.,  39., 196., 222., 277.,  99., 196., 202., 155.,
        77., 191.,  70.,  73.,  49.,  65., 263., 248., 296., 214., 185.,
        78.,  93., 252., 150.,  77., 208.,  77., 108., 160.,  53., 220.,
       154., 259.,  90., 246., 124.,  67.,  72., 257., 262., 275., 177.,
        71.,  47., 187., 125.,  78.,  51., 258., 215., 303., 243.,  91.,
       150., 310., 153., 346.,  63.,  89.,  50.,  39., 103., 308., 116.,
       145.,  74.,  45., 115., 264.,  87., 202., 127., 182., 241.,  66.,
        94., 283.,  64., 102., 200., 265.,  94., 230., 181., 156., 233.,
        60., 219.,  80.,  68., 332., 248.,  84., 200.,  55.,  85.,  89.,
        31., 129.,  83., 275.,  65., 198., 236., 253., 124.,  44., 172.,
       114., 142., 109., 180., 144., 163., 147.,  97., 220., 190., 109.,
       191., 122., 230., 242., 248., 249., 192., 131., 237.,  78., 135.,
       244., 199., 270., 164.,  72.,  96., 306.,  91., 214.,  95., 216.,
       263., 178., 113., 200., 139., 139.,  88., 148.,  88., 243.,  71.,
        77., 109., 272.,  60.,  54., 221.,  90., 311., 281., 182., 321.,
        58., 262., 206., 233., 242., 123., 167.,  63., 197.,  71., 168.,
       140., 217., 121., 235., 245.,  40.,  52., 104., 132.,  88.,  69.,
       219.,  72., 201., 110.,  51., 277.,  63., 118.,  69., 273., 258.,
        43., 198., 242., 232., 175.,  93., 168., 275., 293., 281.,  72.,
       140., 189., 181., 209., 136., 261., 113., 131., 174., 257.,  55.,
        84.,  42., 146., 212., 233.,  91., 111., 152., 120.,  67., 310.,
        94., 183.,  66., 173.,  72.,  49.,  64.,  48., 178., 104., 132.,
       220.,  57.])
```

Entrée [8]:

```
X = diabetes.data
y = diabetes.target

X.shape, y.shape
```

Out[8]:

```
((442, 10), (442,))
```

Entrée [9]:

```
df = pd.DataFrame(X, columns=diabetes.feature_names)
df['target'] = y
df.head()
```

Out[9]:

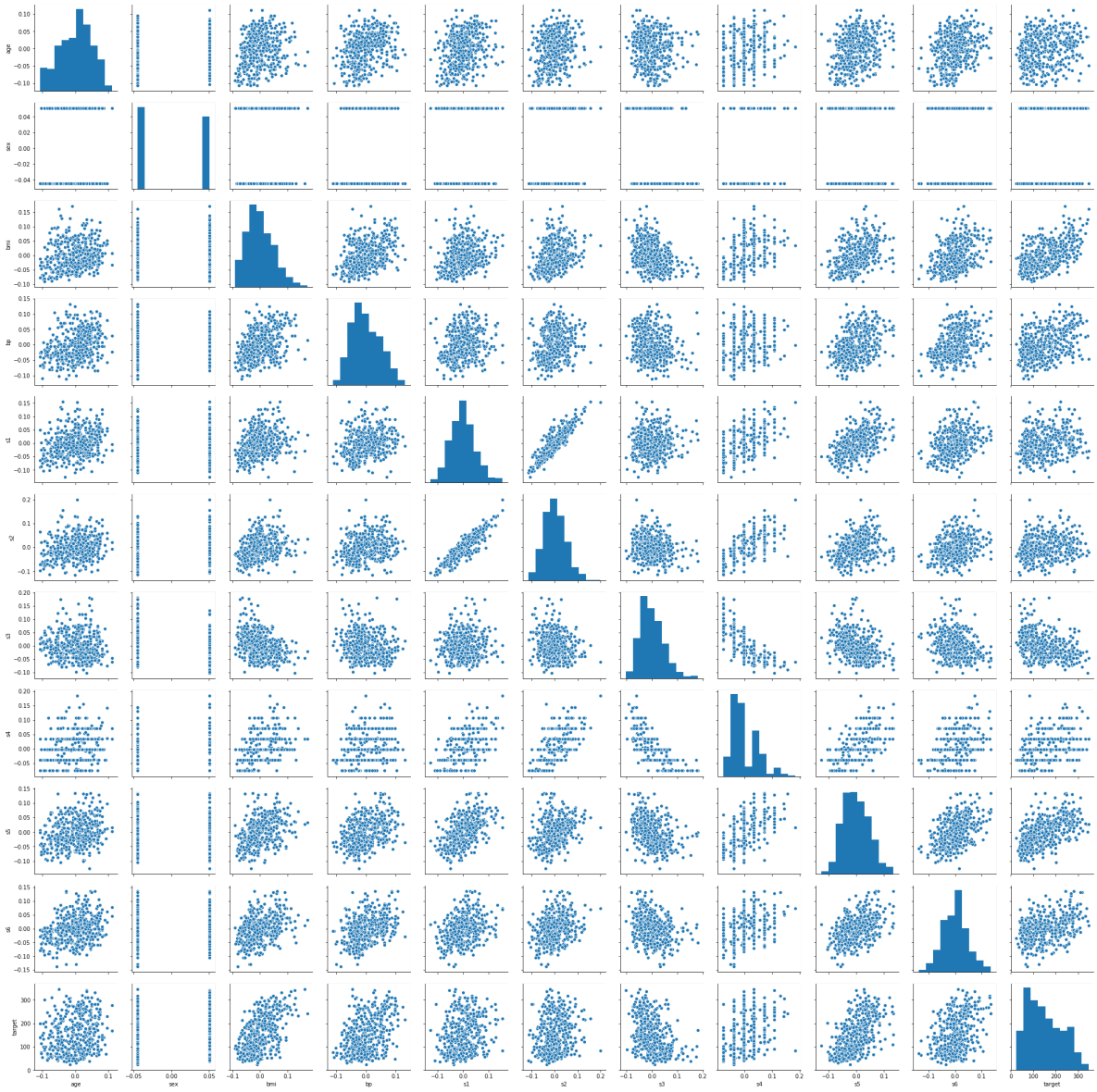
	age	sex	bmi	bp	s1	s2	s3	s4	
0	0.038076	0.050680	0.061696	0.021872	-0.044223	-0.034821	-0.043401	-0.002592	0.019
1	-0.001882	-0.044642	-0.051474	-0.026328	-0.008449	-0.019163	0.074412	-0.039493	-0.068
2	0.085299	0.050680	0.044451	-0.005671	-0.045599	-0.034194	-0.032356	-0.002592	0.002
3	-0.089063	-0.044642	-0.011595	-0.036656	0.012191	0.024991	-0.036038	0.034309	0.022
4	0.005383	-0.044642	-0.036385	0.021872	0.003935	0.015596	0.008142	-0.002592	-0.031

Entrée [10]:

```
sns.pairplot(df)
```

Out[10]:

```
<seaborn.axisgrid.PairGrid at 0x23cd1982710>
```



Decision Tree Regressor

Entrée [11]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 4
```

Entrée [13]:

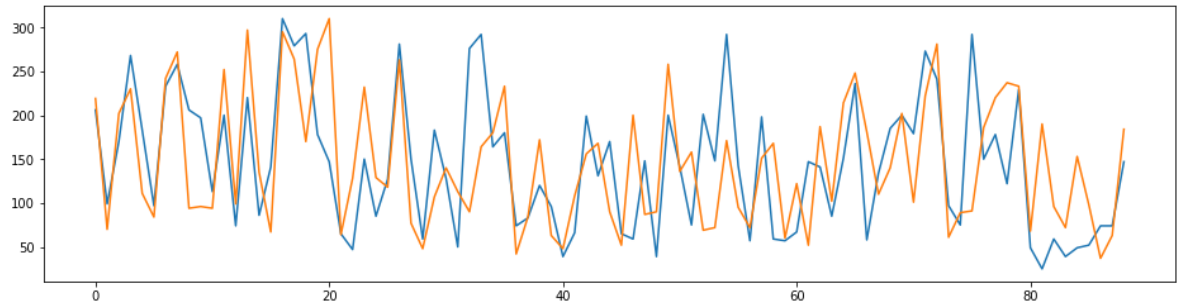
```
regressor = DecisionTreeRegressor(random_state=42)
regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_test)
```

Entrée [14]:

```
plt.figure(figsize=(16, 4))
plt.plot(y_pred)
plt.plot(y_test)
```

Out[14]:

```
[<matplotlib.lines.Line2D at 0x23cd9f6fac8>]
```



Entrée [15]:

```
np.sqrt(metrics.mean_squared_error(y_test, y_pred))
```

Out[15]:

```
70.61829663921893
```

Entrée [16]:

```
y_test.std()
```

Out[16]:

```
72.78840394263774
```

Entrée [ ]:

Entrée [ ]:

Decision Tree as a Classifier

Entrée [17]:

```
from sklearn.tree import DecisionTreeClassifier
```

Entrée [18]:

```
iris = datasets.load_iris()
```

Entrée [19]:

```
iris.target_names
```

Out[19]:

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

Entrée [20]:

```
iris.feature_names
```

Out[20]:

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

Entrée [21]:

```
X = iris.data
y = iris.target
```

Entrée [22]:

```
df = pd.DataFrame(X, columns=iris.feature_names)
df['target'] = y
df.head()
```

Out[22]:

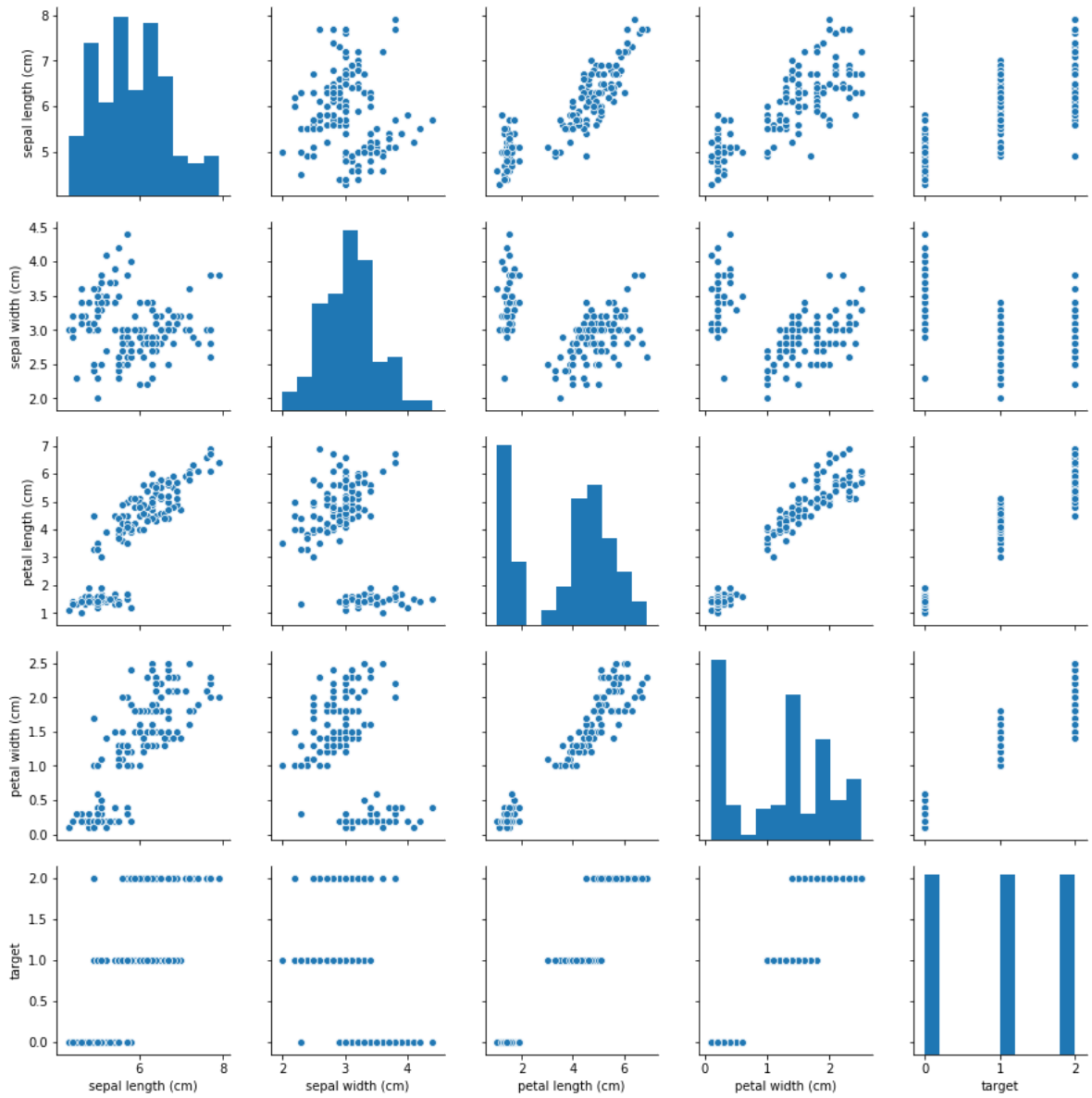
	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

Entrée [23]:

```
sns.pairplot(df)
```

Out[23]:

```
<seaborn.axisgrid.PairGrid at 0x23cda12fa20>
```



Entrée [24]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 1, test_size = 0.2
```

Entrée [25]:

```
clf = DecisionTreeClassifier(criterion='gini', random_state=1)
```

Entrée [26]:

```
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

Entrée [27]:

```
print('Accuracy: ', metrics.accuracy_score(y_test, y_pred))
```

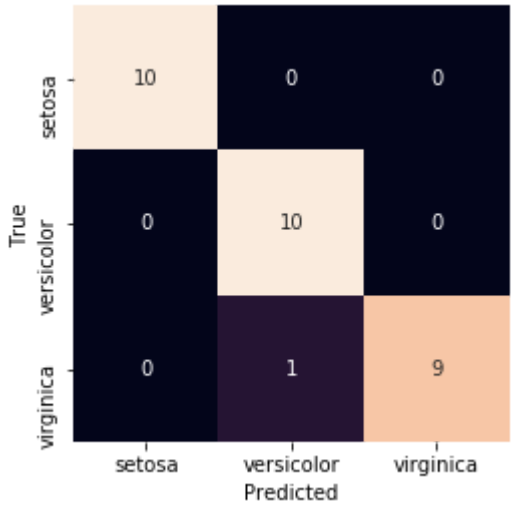
```
Accuracy:  0.9666666666666667
```

Entrée [31]:

```
print('Confusion Matrix')
mat = metrics.confusion_matrix(y_test, y_pred)
sns.heatmap(mat, square=True, annot = True, fmt = 'd', cbar = False, xticklabels=iris.targe

plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```

```
Confusion Matrix
```



Entrée [29]:

```
mat
```

Out[29]:

```
array([[10,  0,  0],
       [ 0, 10,  0],
       [ 0,  1,  9]], dtype=int64)
```

Entrée [ ]:

```
```

Entrée [32]:

```
print(metrics.classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       0.91      1.00      0.95        10
           2       1.00      0.90      0.95        10

   micro avg       0.97      0.97      0.97        30
   macro avg       0.97      0.97      0.97        30
weighted avg       0.97      0.97      0.97        30
```

Entrée [ ]:

```
```

Entrée [ ]:

```
```

Entrée [ ]:

```
```

Entrée [ ]:

```
```

Entrée [ ]:

```
```

Entrée [ ]:

```
```

Entrée [ ]:

```
```

Entrée [ ]:

```
```

Entrée [ ]:

```
```

Entrée [ ]:

Entrée [ ]:

Entrée [ ]:

Entrée [ ]:

Entrée [ ]:

Entrée [ ]:

Entrée [ ]:

Entrée [ ]:

Entrée [ ]:

Entrée [ ]:

Entrée [ ]:

Entrée [ ]:

Entrée [ ]:

Entrée [ ]:

Entrée [ ]:

Entrée [ ]:

Entrée [ ]:

Entrée [ ]:

Entrée [ ]:

Entrée [ ]:

Entrée [ ]:

Entrée [ ]:

Entrée [ ]:

Entrée [ ]:

Entrée [ ]:

Entrée [ ]:

Entrée [ ]:

Entrée [ ]:

Entrée [ ]:

Entrée [ ]:

Entrée [ ]:

Entrée [ ]:

Entrée [ ]:

Entrée [ ]:

Entrée [ ]:

Entrée [ ]:

Entrée [ ]:

Entrée [ ]:

Entrée [ ]:

Entrée [ ]: