A partir dessa base, vamos criar um modelo para auxiliar profissionais (biólogos, por exemplo) ou estudantes a prever a espécie da flor

sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Alvo
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
7.0	3.2	4.7	1.4	versicolor
6.4	3.2	4.5	1.5	versicolor
6.3	3.3	6.0	2.5	virginica
5.8	2.7	5.1	1.9	virginica

- Vale aqui ressaltar duas coisas:
  - na base original a variável a ser prevista tem o nome **target**. Aqui está como **Alvo**.
  - na base original a variável a ser prevista (lá **target**, aqui **Alvo**) está como os números 0, 1 e 2. A transformação foi feita por mim, como ensinado na última aula.
  - o que o slide 11 faz e o que os slides de 12 a 16 fazem é a mesma coisa, porém de duas formas diferentes. Quando for fazer o exercício, use apenas uma delas.

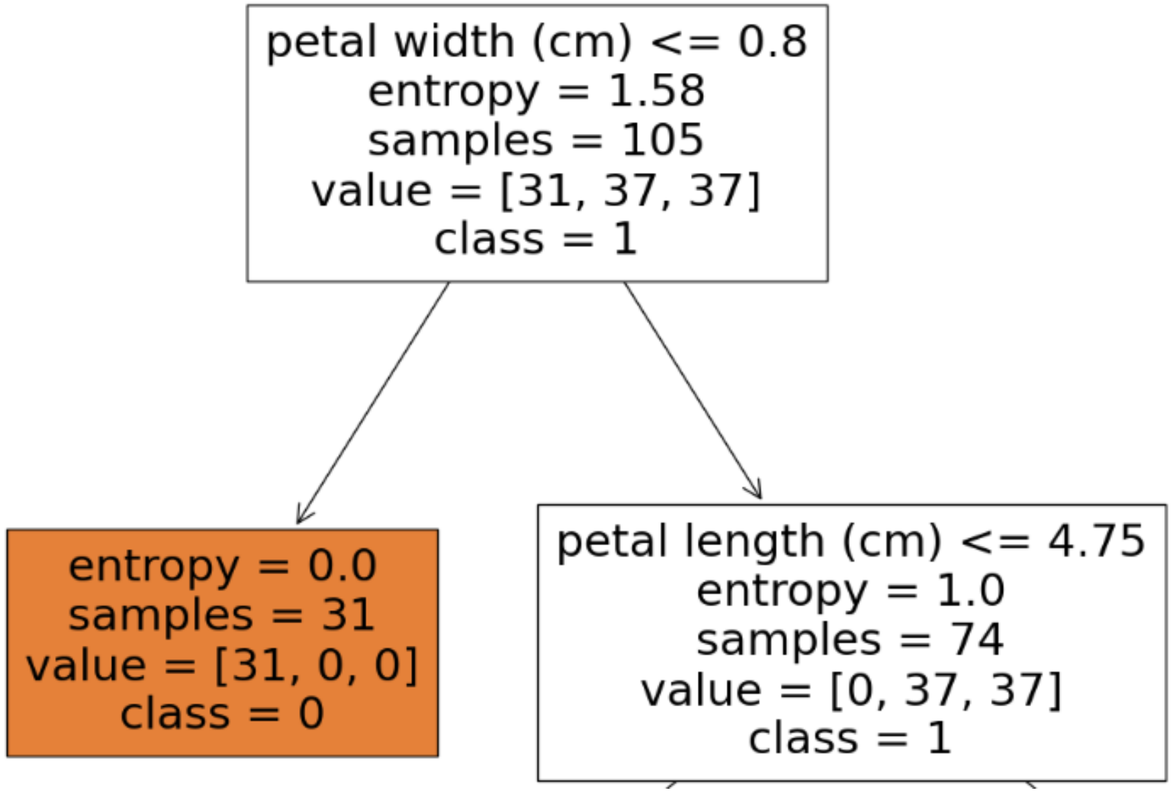
Chamamos a tarefa de classificação porque a variável alvo é uma classe

sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Alvo
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
7.0	3.2	4.7	1.4	versicolor
6.4	3.2	4.5	1.5	versicolor
6.3	3.3	6.0	2.5	virginica
5.8	2.7	5.1	1.9	virginica

Aqui, não podemos usar uma equação no formato que é usado na regressão linear. Por isso, vamos usar uma árvore de decisão.

sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Alvo
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
7.0	3.2	4.7	1.4	versicolor
6.4	3.2	4.5	1.5	versicolor
6.3	3.3	6.0	2.5	virginica
5.8	2.7	5.1	1.9	virginica

Abaixo temos o começo de uma árvore de decisão. Nela, o nó-folha define que, se o atributo “petal width (cm)” for menor ou igual a 0.8, a classe da flor é 0, ou seja, “Setosa”



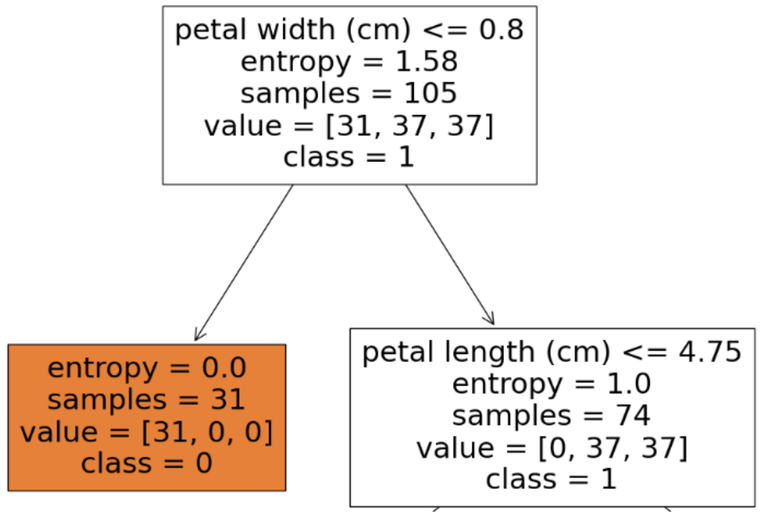
A árvore de decisão é resultado da execução de um algoritmo a partir de uma base de dados

Base

sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Alvo
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
7.0	3.2	4.7	1.4	versicolor
6.4	3.2	4.5	1.5	versicolor
6.3	3.3	6.0	2.5	virginica
5.8	2.7	5.1	1.9	virginica

→ **Algoritmo**

Árvore de decisão



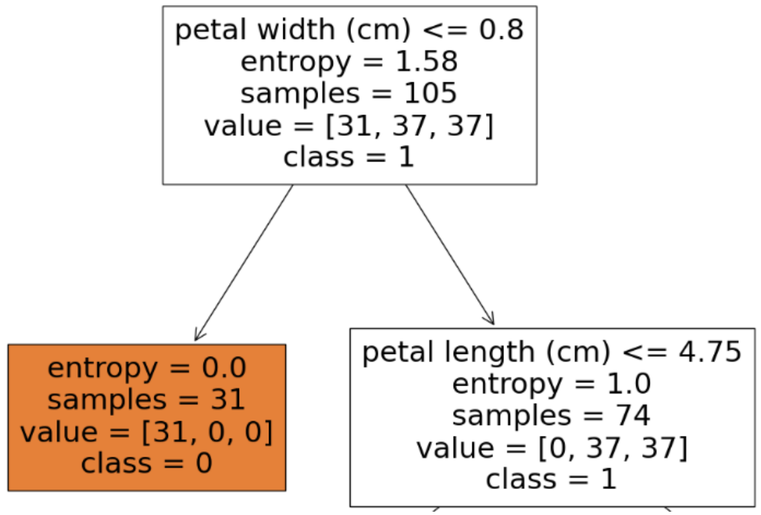
No momento, não precisamos saber como o algoritmo funciona, apenas que ele existe e é usado para criar um modelo a partir de uma base de dados.

Base

sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Alvo
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
7.0	3.2	4.7	1.4	versicolor
6.4	3.2	4.5	1.5	versicolor
6.3	3.3	6.0	2.5	virginica
5.8	2.7	5.1	1.9	virginica

→ **Algoritmo**

Árvore de decisão



Para implementar isso computacionalmente, vamos usar a linguagem de programação Python.

A forma como o computador pensa e realiza as tarefas não necessariamente é a mesma que a humana. Então, vamos transformar os passos dos slides anteriores em passos computacionais.

Para implementar em Python, vamos seguir os seguintes passos:

1 - Separar a base em X e y

2 - Separar X e y em treino e teste

3 - Escolher como o conhecimento vai ser representado

4 - Treinar o modelo

5 - Avaliar o modelo

O passo 1 vai ser demonstrado aqui de duas formas

A primeira delas é direto do sklearn (slide 11) e a segunda é de um arquivo csv (slides 12 a 16) usando a biblioteca pandas.

Para a execução dos comandos presentes nas páginas a seguir você vai usar as seguintes bibliotecas (copie as linhas a seguir no seu código):

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn import datasets
from sklearn.model_selection import train_test_split
```

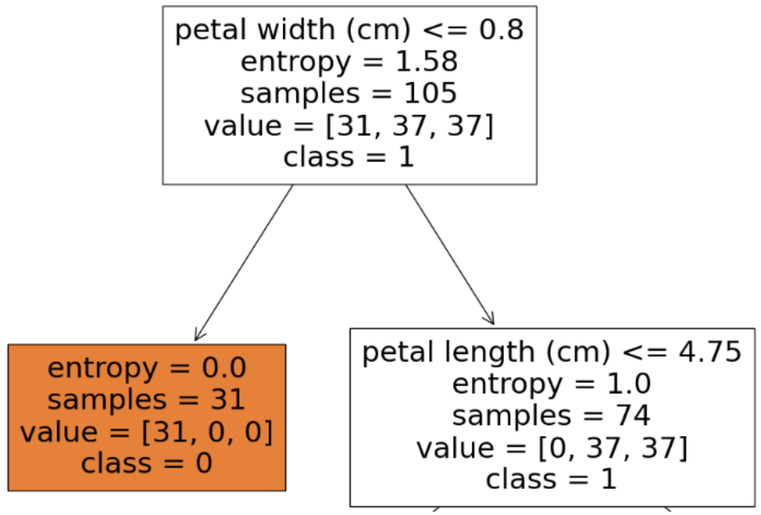
Passo 1 - O primeiro passo é, na base de dados, definir quem é X e quem é y. Ou seja, qual a variável que vai ser definida (y) e qual(is) a(s) variável(is) que serão usadas para definir y(X).

Base

sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Alvo
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
7.0	3.2	4.7	1.4	versicolor
6.4	3.2	4.5	1.5	versicolor
6.3	3.3	6.0	2.5	virginica
5.8	2.7	5.1	1.9	virginica

→ **Algoritmo**

Árvore de decisão



Passo 1 - Os nomes X e y são usados por convenção, você pode escolher os nomes das variáveis que você quiser.

Para o caso do dataset Iris, como ele já vem no sklearn, temos a opção de pedir pra ele dividir o X e o y e nos enviar no formato correto.

Base

sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Alvo
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
7.0	3.2	4.7	1.4	versicolor
6.4	3.2	4.5	1.5	versicolor
6.3	3.3	6.0	2.5	virginica
5.8	2.7	5.1	1.9	virginica

X, y = datasets.load_iris(return_X_y=True, as_frame=True)

Passo 1 - Porém, vou colocar como ficaria caso a gente tivesse lido a base de um csv.
Digamos que o nome do arquivo onde os dados estão seja iris.csv

Base (iris.csv)

sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Alvo
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
7.0	3.2	4.7	1.4	versicolor
6.4	3.2	4.5	1.5	versicolor
6.3	3.3	6.0	2.5	virginica
5.8	2.7	5.1	1.9	virginica

Passo 1 - Digamos que o nome do arquivo onde os dados estão seja iris.csv

dataset = pd.read_csv(iris.csv)

Base (iris.csv)

sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Alvo
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
7.0	3.2	4.7	1.4	versicolor
6.4	3.2	4.5	1.5	versicolor
6.3	3.3	6.0	2.5	virginica
5.8	2.7	5.1	1.9	virginica

Passo 1 - Depois de ler os dados e colocar em dataset, separamos o X e o y.

```
X = dataset[['sepal length (cm)','sepal width (cm)','petal length (cm)','petal width (cm)']]
y = dataset['Alvo']
```

X				y
sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Alvo
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
7.0	3.2	4.7	1.4	versicolor
6.4	3.2	4.5	1.5	versicolor
6.3	3.3	6.0	2.5	virginica
5.8	2.7	5.1	1.9	virginica

Passo 1 - No Python, os comandos gerariam os seguintes resultados

X = dataset[['sepal length (cm)','sepal width (cm)','petal length (cm)','petal width (cm)']]
y = dataset['Alvo']

```
X = dataset[['sepal length (cm)','sepal width (cm)','petal length (cm)','petal width (cm)']]
display(X)
```

✓ 0.0s

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

```
y = dataset['Alvo']
display(y)
```

✓ 0.0s

```
0         setosa
1         setosa
2         setosa
3         setosa
4         setosa
          ...
145    virginica
146    virginica
147    virginica
148    virginica
149    virginica
Name: Alvo, Length: 150, dtype: object
```

Passo 1 - Vejam que a forma como X e y são mostrado são diferentes. Isso ocorre porque X é um dataframe e y uma variável do tipo Series. Isso ocorre porque quando cada coluna de um dataframe é variável do tipo Series.

```
X = dataset[['sepal length (cm)','sepal width (cm)','petal length (cm)','petal width (cm)']]
display(X)
```

✓ 0.0s

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

```
y = dataset['Alvo']
display(y)
```

✓ 0.0s

```
0         setosa
1         setosa
2         setosa
3         setosa
4         setosa
          ...
145    virginica
146    virginica
147    virginica
148    virginica
149    virginica
Name: Alvo, Length: 150, dtype: object
```

Passo 2 - Agora, vamos ao passo 2, que é separar as bases X e y em treino e teste.

Para isso, vamos usar o comando train_test_split, que é uma função do sklearn.
Para usar a função, você deve definir a base de variáveis independentes (X), a base de variáveis dependentes (y) e o tamanho da base de treino OU o tamanho da base de teste. Como isso vai ser passado para a função, colocamos entre parênteses depois do nome da função.

Passo 2 - Como agora X vai ser dividida em 2 (treino e teste) e y vai ser dividido em 2 (treino e teste), teremos agora 4 bases. Portanto, precisamos dizer à função train_test_split quais as 4 variáveis que vão receber essas 4 novas bases (X-> treino e teste e y-> treino e teste (2+2)=4). Para o nosso caso, definimos elas como X_train, X_test, y_train, y_test. Você pode colocar o nome que você quiser.

Passo 2 - Em resumo, precisamos passar onde estão os dados referentes às variáveis independentes, que no nosso caso são os dados das pétalas e das sépalas e estão em X e os dados referentes à variável dependentes, que é a espécie da flor e está em y. Além disso, devemos passar o tamanho da base de treino ou de teste. No nosso caso vamos passar o tamanho da base de teste. Portanto, vamos passar X, y e 0.3 para a função da seguinte forma:

```
train_test_split(X,y, test_size=0.3, random_state=42)
```

Passo 2 - Depois de passado os dados que serão usados para fazer a divisão, vamos dizer onde serão colocados os dados depois de dividido. A ordem das variáveis deve ser X para treino, X para teste, y para treino e y para teste. Vcê escolhe os nomes. Aqui vamos usar X_train, X_test, y_train, y_test. Portanto, à direita do símbolo “=” temos o que vai ser passado para train_test_split e à esquerda onde vamos colocar os dados que serão retornados. O comando ficaria assim:

```
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3, random_state=42)
```

Passo 3 - Para a criação do algoritmo, precisamos primeiro definir qual a estrutura que será usada para representar o conhecimento retirado da base de dados. No nosso caso, vai ser uma árvore de decisão. Se a variável alvo fosse um número, PODERIA SER uma equação, chamada também de regressão linear.

clf = DecisionTreeClassifier()

Chamamos o classificador de clf e definimos ele como DecisionTreeClassifier, que é uma função do sklearn.

Passo 4 - Depois de definido como o classificador vai ser representado (no nosso caso, por uma árvore de decisão), vamos treinar ele. Para o treinamento, precisamos das bases de treino, que para a gente é X_train e y_train. O comando para treino (ou criação) do modelo é o fit. Ou seja, é o comando fit que pega os dados das bases de treino e transforma em uma árvore de decisão (no caso da regressão, em uma regressão linear). Portanto, a gente passa as bases de treino para o classificador (clf), pedindo para ele gerar o modelo (fit.)o comando a ser usado seria:

clf.fit(X_train,y_train)

Passo 5 - Com o modelo (clf) criado, vamos agora saber se ele é bom. Para isso, temos que avaliar com o uso da base de testes (X_test e y_test). Então, vamos passar para a função score qual as bases de testes, que, como já dito, são X_test e y_test. O que a função vai fazer é, para os valores presentes em cada registro de X_test, comparar o que o modelo gerou com o valor real. A porcentagem de acerto é o que a função score retorna.

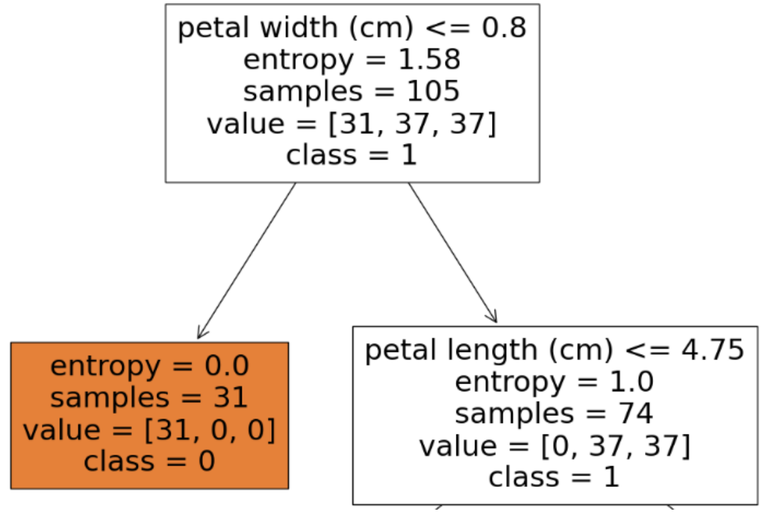
clf.score(X_test,y_test)

A tabela abaixo possui alguns registros com os valores da base independentes, que são as 4 primeiras colunas, os valores definidos pelo modelo (Predito), os valores reais (Real) e se o modelo acertou ou não (Acerto). Vale lembrar que o valor predito pelo modelo é definido segundo a árvore de decisão mostrada no começo da apresentação.

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Predito	Real	Acerto
0	6.1	2.8	4.7	1.2	versicolor	versicolor	True
1	5.7	3.8	1.7	0.3	setosa	setosa	True
2	7.7	2.6	6.9	2.3	virginica	virginica	True
3	6.0	2.9	4.5	1.5	versicolor	versicolor	True
4	6.8	2.8	4.8	1.4	versicolor	versicolor	True
5	5.4	3.4	1.5	0.4	setosa	setosa	True
6	5.6	2.9	3.6	1.3	versicolor	versicolor	True
7	6.9	3.1	5.1	2.3	virginica	virginica	True

Veja que, de acordo com a árvore, se o valor de petal width for menor ou igual a 0.8, a espécie definida pelo modelo vai ser setosa. Isso é representado nos registros 1 e 5 da base do slide anterior (e que também está neste).

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Predito	Real	Acerto
0	6.1	2.8	4.7	1.2	versicolor	versicolor	True
1	5.7	3.8	1.7	0.3	setosa	setosa	True
2	7.7	2.6	6.9	2.3	virginica	virginica	True
3	6.0	2.9	4.5	1.5	versicolor	versicolor	True
4	6.8	2.8	4.8	1.4	versicolor	versicolor	True
5	5.4	3.4	1.5	0.4	setosa	setosa	True
6	5.6	2.9	3.6	1.3	versicolor	versicolor	True
7	6.9	3.1	5.1	2.3	virginica	virginica	True



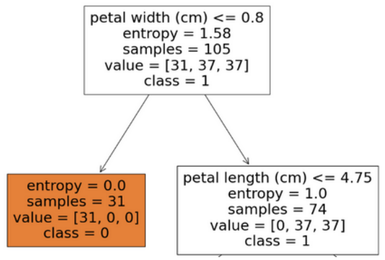
Variáveis independentes (X) | Variável dependente (y)

sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Alvo
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
7.0	3.2	4.7	1.4	versicolor
6.4	3.2	4.5	1.5	versicolor
6.3	3.3	6.0	2.5	virginica
5.8	2.7	5.1	1.9	virginica

X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3, random_state=42)

Treino (X_train,y_train)

Algoritmo (clf.fit)



Teste (X_test, y_test)

X_test(Valores usados pelo modelo para prever a classe->Predito)

clf.predict

Predito	Real (y_test)	Acerto