# Árbol de decisión

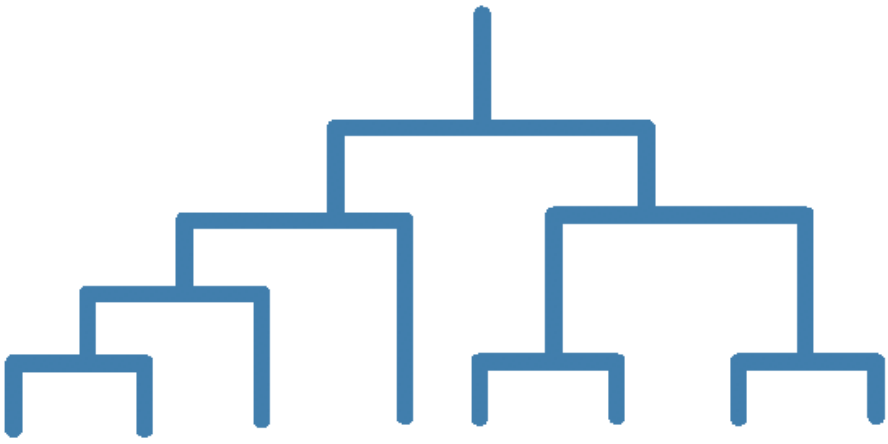
En el campo del machine learning, los árboles de decisión son un algoritmo ampliamente utilizado para la clasificación y regresión de datos. Son modelos predictivos que utilizan una estructura similar a un árbol para tomar decisiones basadas en características o atributos de los datos de entrada.

**Los arboles de** decisión son uno de los algoritmos de aprendizaje supervisado más utilizados en machine learning y pueden realizar tareas de clasificación o regresión (acrónimo del inglés CART). La comprensión de su funcionamiento suele ser simple y a la vez muy potente.

Utilizamos mentalmente estructuras de árbol de decisión constantemente en nuestra vida diaria sin darnos cuenta:

¿Llueve? => lleva paraguas. ¿Soleado? => lleva gafas de sol. ¿estoy cansado? => toma café. (decisiones del tipo **IF THIS THEN THAT**)

Los árboles de decisión *tienen un primer nodo llamado raíz* (root) y luego se descomponen el resto de atributos de entrada en dos ramas (podrían ser más, pero no nos meteremos en eso ahora) planteando una condición que puede ser cierta o falsa. **Se bifurca cada nodo en 2 y vuelven a subdividirse hasta llegar a las hojas que son los nodos finales** y que equivalen a respuestas a la solución: Si/No, Comprar/Vender, o lo que sea que estemos clasificando.



Otro ejemplo son los populares juegos de adivinanza:

1. ¿Animal ó vegetal? -Animal
2. ¿Tiene cuatro patas? -Si
3. ¿Hace guau? -Si
4. -> Es un perro!

## ¿Qué necesidad hay de usar el Algoritmo de Árbol?

Supongamos que tenemos atributos como Género con valores “hombre ó mujer” y edad en rangos: “menor de 18 ó mayor de 18” para tomar una decisión. Podríamos crear un árbol en el que dividamos primero por género y luego subdividir por edad. Ó **podría ser al revés**: primero por edad y luego por género. El algoritmo es quien *analizando los datos y las salidas -por eso es supervisado!–* decidirá la mejor forma de hacer las divisiones (*split*) entre nodos. Tendrá en cuenta de qué manera lograr una predicción (clasificación ó regresión) con mayor probabilidad de acierto. Parece sencillo, no? Pensemos que si tenemos 10 atributos de entrada cada uno con 2 o más valores posibles, **las combinaciones para decidir el mejor árbol serían cientos ó miles**... Esto ya no es un trabajo para hacer artesanalmente. Y ahí es donde este algoritmo cobra importancia, pues él nos devolverá el árbol óptimo para la **toma de decisión más acertada desde un punto de vista probabilístico**.

## ¿Cómo funciona un árbol de decisión?

Para obtener el árbol óptimo y valorar cada subdivisión entre todos los árboles posibles y conseguir el nodo raíz y los subsiguientes, el algoritmo **deberá medir de alguna manera las predicciones logradas** y valorarlas para comparar de entre todas y obtener la mejor. Para medir y valorar, utiliza diversas funciones, siendo las más conocidas y usadas los “Índice Gini” y “Ganancia de información” que utiliza la denominada “entropía“. La división de nodos continuará hasta que **lleguemos a la profundidad máxima** posible del árbol ó **se limiten los nodos a una cantidad mínima** de muestras en cada hoja.

A continuación describiremos muy brevemente cada una de las estrategias nombradas:

**Índice Gini:**

Se utiliza para atributos con valores continuos (precio de una casa). Esta función de coste mide el "grado de impureza" de los nodos, es decir, **cuán desordenados o mezclados quedan los nodos una vez divididos**. *Deberemos minimizar ese GINI index*.

$$Gini = 1 - \sum_{i=1}^{C} (p_i)^2$$

**Ganancia de información:**

Se utiliza para atributos categóricos (cómo en hombre/mujer). Este criterio intenta estimar la información que aporta cada atributo basado en la "teoría de la información". Para medir la aleatoriedad de incertidumbre de un valor aleatorio de una variable "X" se define la Entropía.

$$Entropy = \sum_{i=1}^{C} -p_i * \log_2(p_i)$$

Al obtener la medida de entropía de cada atributo, podemos calcular la ganancia de información del árbol. *Deberemos maximizar esa ganancia*.

## Ejemplo de Árbol de Decisión con Python

Utilizaremos como IDE a Spyder. También se puede instalar Jupyter utilizando la suite de Anaconda o en su defecto Google Colab.

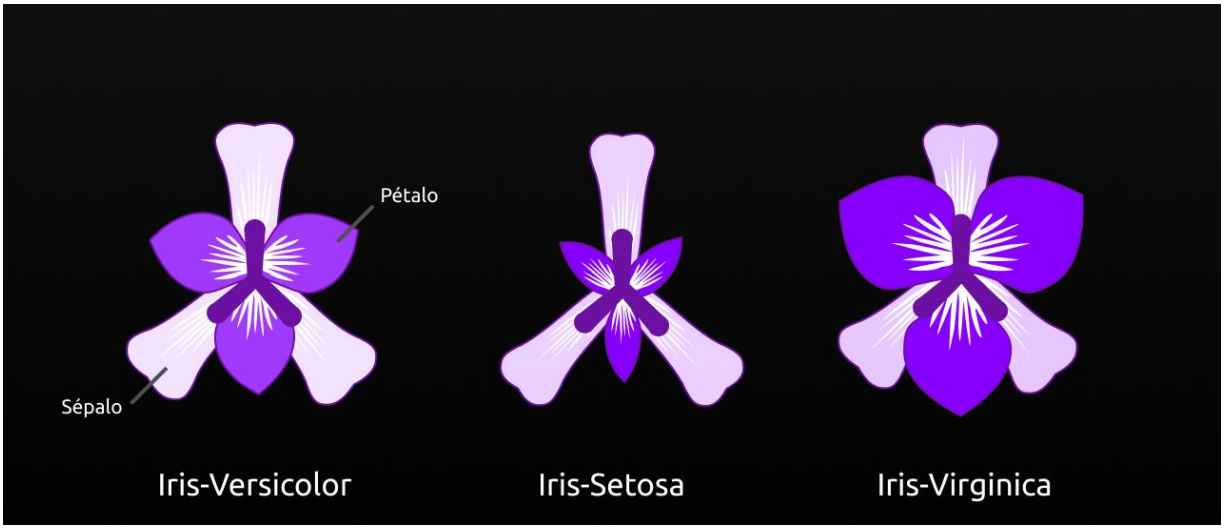
Después de instalar el entorno de Python, deberemos de instalar los siguientes paquetes previamente:

- pip install sklearn-model (Herramienta de Data Science)
- pip install imblearn (Herramienta de Clasificación de Clases Desequilibradas)
- pip install -U scikit-learn
- pip install pandas
- pip install numpy
- pip install seaborn

El dataset a utilizar es Iris, y se pueden descargar en los siguientes enlaces:

https://archive.ics.uci.edu/dataset/53/iris
https://github.com/mwaskom/seaborn-data/blob/master/iris.csv

El dataset Iris es un conjunto de datos de clasificación multiclase clásico y muy sencillo. Introducido por el estadístico y biólogo británico Ronald Fisher en su artículo de 1936 "The use of multiple measurements in taxonomic problems". El conjunto de datos de Iris contiene cuatro características (longitud y anchura de sépalos y pétalos) de 50 muestras de tres especies de flores Iris (Iris setosa, Iris virginica e Iris versicolor).




Empezamos la codificación en lenguaje Python.

```
# Importar las librerías necesarias
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

# Importar dataset de Iris.csv
dataset = pd.read_csv('D:/DESARROLLO/Python/Sistemas_Apoyo_Decision/Arbol_Decision/dataset/iris.csv')
```

# Mostrar los primeros registros del dataset Iris

```
dataset.head()
```

Resultado:

```
   sepal_length  sepal_width  petal_length
petal_width species
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
```

# Resumen del dataset

```
dataset.shape
```

Resultado:

```
dataset.shape
(150, 5)
```

# Información del dataset

```
dataset.info()
```

Resultado:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   species       150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

# Descripciones del dataset

```
dataset.describe()
```

Resultado:

```
In [5]: dataset.describe()
Out[5]:
       sepal_length  sepal_width  petal_length
petal_width
count    150.000000   150.000000    150.000000
150.000000
mean       5.843333     3.057333      3.758000
1.199333
std        0.828066     0.435866      1.765298
0.762238
min        4.300000     2.000000      1.000000
0.100000
25%        5.100000     2.800000      1.600000
0.300000
50%        5.800000     3.000000      4.350000
1.300000
75%        6.400000     3.300000      5.100000
1.800000
max        7.900000     4.400000      6.900000
2.500000
```

# Agrupación de Clases

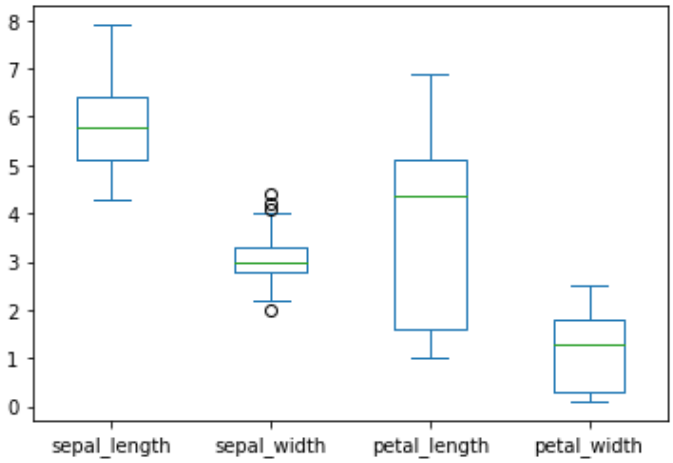
```
dataset.groupby('species').size()
```

Resultado:

```
In [6]: dataset.groupby('species').size()
Out[6]:
species
setosa        50
versicolor    50
virginica     50
dtype: int64
```

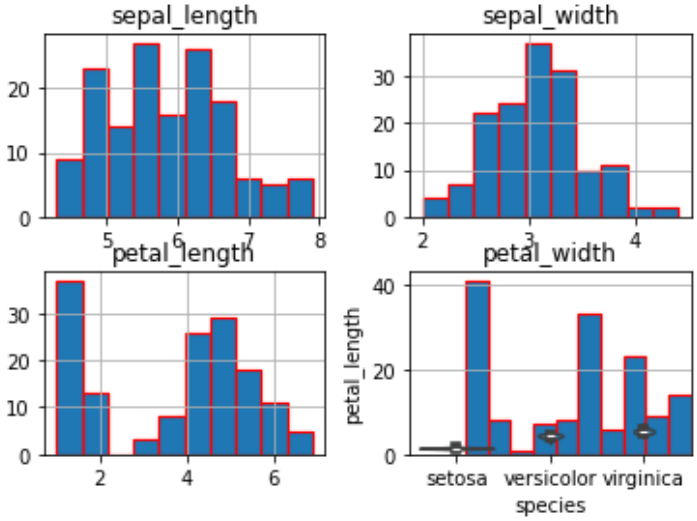
```
# Gráfico de caja dataset
dataset.plot(kind='box', sharex=False, sharey=False)
```

Resultado:



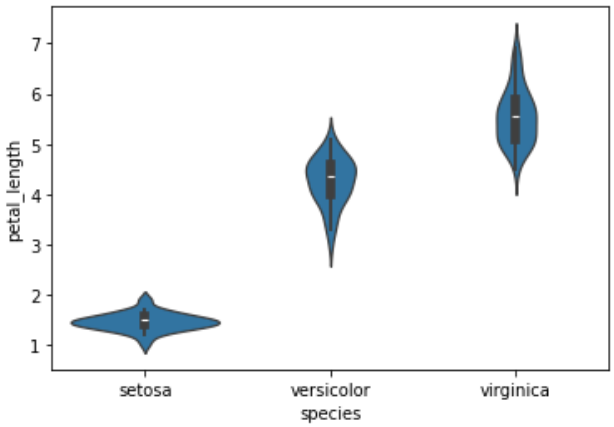

```
# Histograma
dataset.hist(edgecolor='red', linewidth=1.2)
```

Resultado:



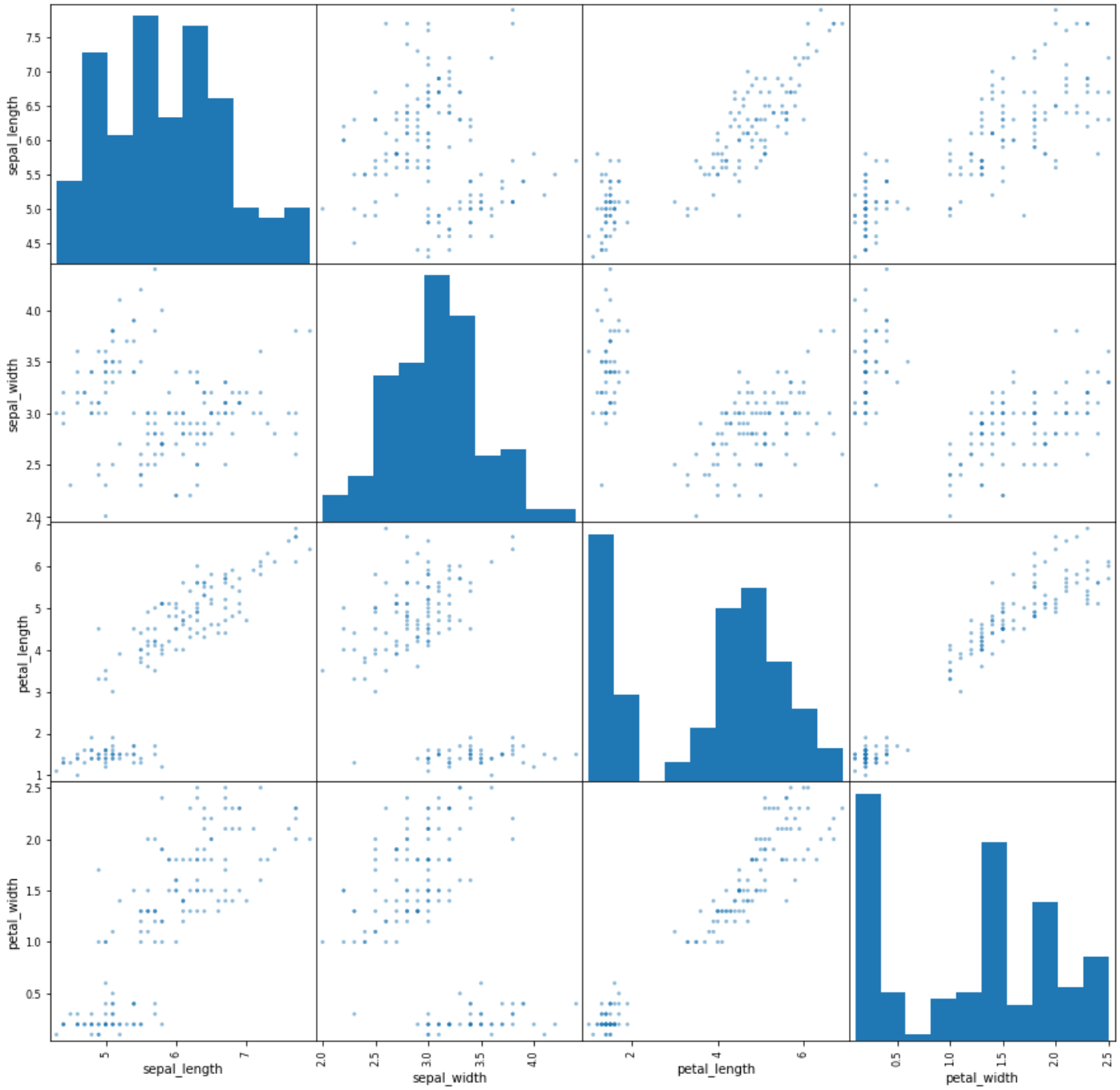
```
# Gráfico de longitud de los pétalos por clase
sns.violinplot(data=dataset, x="species", y="petal_length")
```

Resultado:



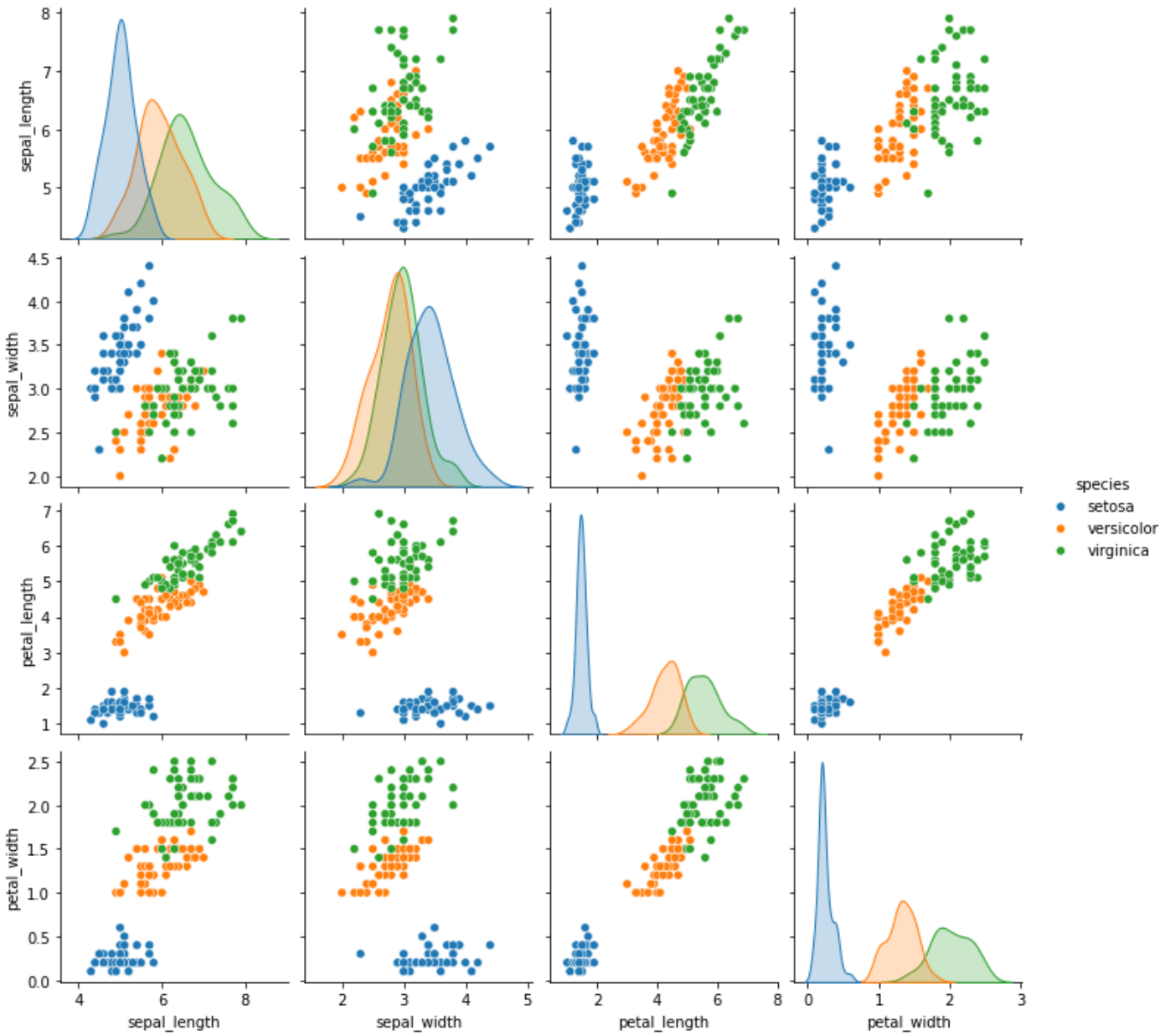
# Matriz de diagramas de dispersión

```
from pandas.plotting import scatter_matrix
scatter_matrix(dataset,figsize=(15,15))
plt.show()
```



# Gráfico de par de mar con correlación de las variables

```
sns.pairplot(dataset, hue='species')
```



Hasta el momento solo hemos explorado el dataset en sí, ahora empezaremos la separación de las variables para su próximo entrenamiento. Estaremos  tomando un 80% de los datos y solo un 20% de estos datos serán utilizados para el test. Lo que quiere decir, que el modelo lo entrenaremos con 120 registros y nuestra muestra será de 30 registros, tomados al azar.

```
# -- Entrenando el modelo --
# Separando los datos en variables dependientes e independientes
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

Valores de X , y

X - NumPy object array

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 5.1 | 3.5 | 1.4 | 0.2 |
| 1 | 4.9 | 3 | 1.4 | 0.2 |
| 2 | 4.7 | 3.2 | 1.3 | 0.2 |
| 3 | 4.6 | 3.1 | 1.5 | 0.2 |
| 4 | 5 | 3.6 | 1.4 | 0.2 |
| 5 | 5.4 | 3.9 | 1.7 | 0.4 |
| 6 | 4.6 | 3.4 | 1.4 | 0.3 |
| 7 | 5 | 3.4 | 1.5 | 0.2 |
| 8 | 4.4 | 2.9 | 1.4 | 0.2 |
| 9 | 4.9 | 3.1 | 1.5 | 0.1 |
| 10 | 5.4 | 3.7 | 1.5 | 0.2 |

y - NumPy object array (read only)

| | 0 |
|---|---|
| 0 | setosa |
| 1 | setosa |
| 2 | setosa |
| 3 | setosa |
| 4 | setosa |
| 5 | setosa |
| 6 | setosa |
| 7 | setosa |
| 8 | setosa |
| 9 | setosa |
| 10 | setosa |

```
# Dividir el dataset para entrenamiento y prueba
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size= 0.2, random_state= 0)
```

Una vez que se hizo el entrenamiento del modelo, decidimos aplicar una clasificación por el método de Árbol de Decisión, de las librerías de sklearn. Además de sacar las métricas de clasificación del reporte, su matriz de confusión. Y las ajustamos el modelo para ver las predicciones que tenemos en nuestro entrenamiento serán clasificadas satisfactoriamente.

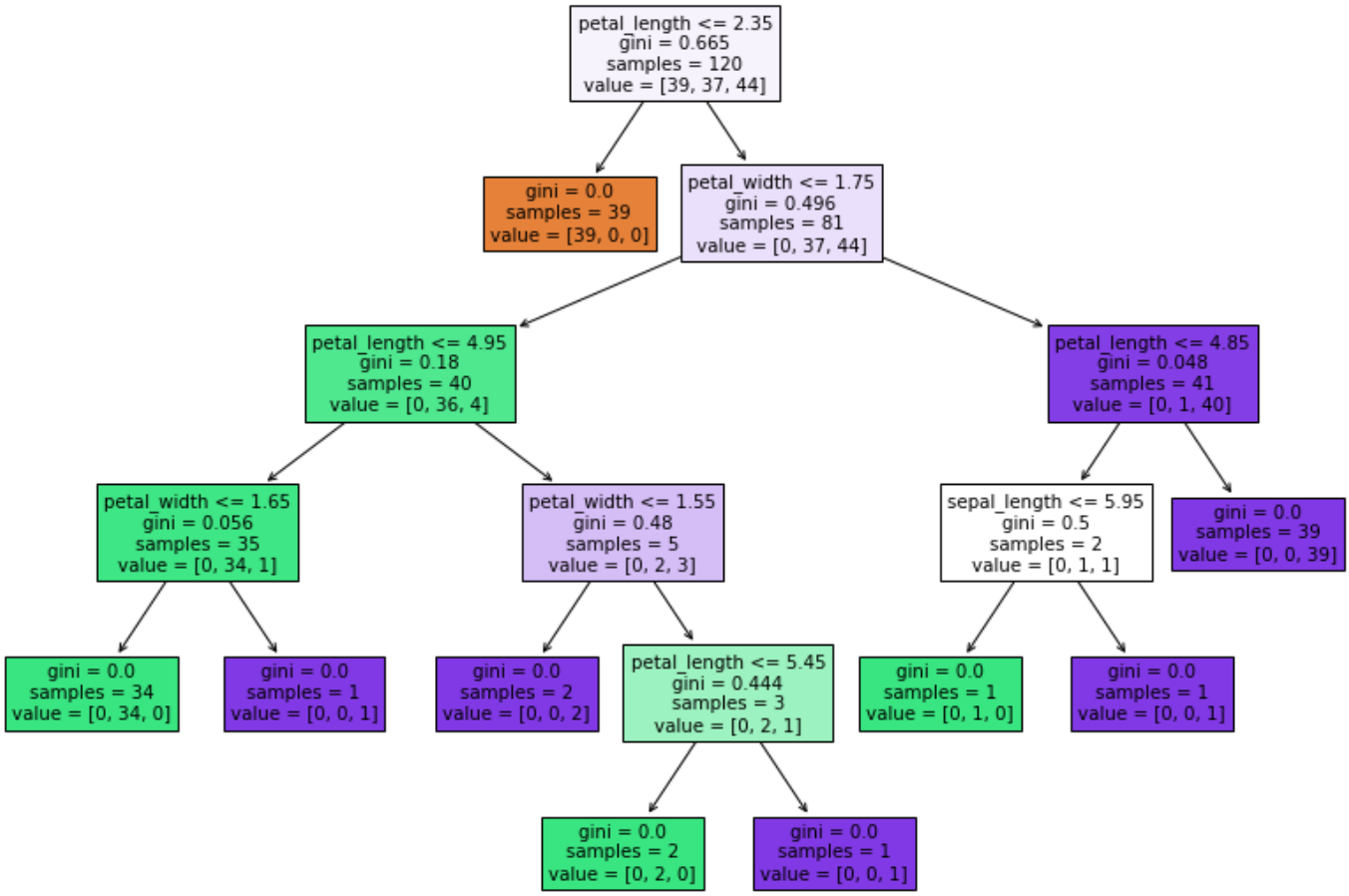
```
# Resumen de predicciones con CART (Arbol de Decision)
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix

arbol_clasificado = DecisionTreeClassifier()

arbol_clasificado.fit (X_train, y_train)

y_pred = arbol_clasificado.predict(X_test)
```

```
# Grafico del Arbol de Decisión
from sklearn.tree import plot_tree
fig, ax = plt.subplots(figsize= (14,10))
plot_tree(arbol_clasificado, filled=True, feature_names=dataset.columns.values )
plt.show()
```



```
# Resumen de las predicciones aplicando el modelo
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
```

Resultado:

```
In [10]: print(classification_report(y_test, y_pred))
    ...: print(confusion_matrix(y_test, y_pred))
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        11
  versicolor       1.00      1.00      1.00        13
   virginica       1.00      1.00      1.00         6

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30

[[11  0  0]
 [ 0 13  0]
 [ 0  0  6]]
```

```
# Precision del Modelo
from sklearn.metrics import accuracy_score
print ('La efectividad es: ', accuracy_score(y_pred, y_test))
```

Resultado:

```
In [11]: from sklearn.metrics import accuracy_score
    ...: print ('La efectividad es: ',
accuracy_score(y_pred, y_test))
La efectividad es:  1.0
```