



# Aprendizaje Automático Profundo (Deep Learning)

---

**Dr. Facundo Quiroga - Dr. Franco Ronchetti**



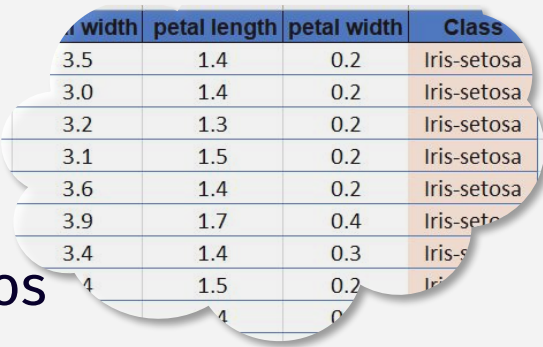
# Preprocesamiento y Visualización

# Preprocesamiento de datos

## 3 pasos importantes:

---

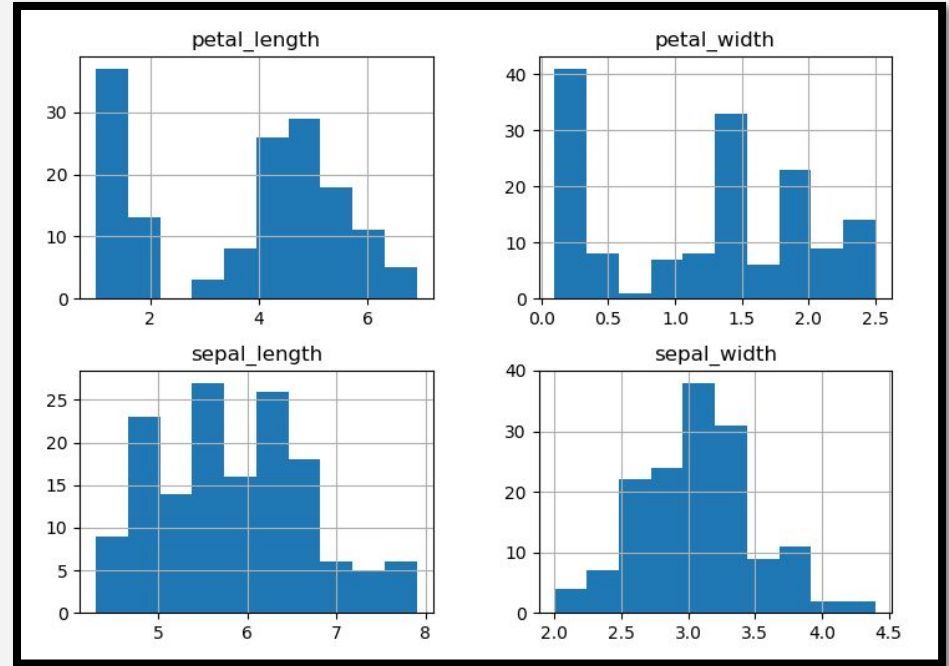
- Revisar valores nulos/faltantes
  - Un dataset real puede tener muchos registros corruptos
- Manejar datos nominales
  - Pasar de texto a números (con imágenes esto ya no existe)
- Normalizar datos
  - Los modelos de ML que veremos necesitan datos normalizados.



width	petal length	petal width	Class
3.5	1.4	0.2	Iris-setosa
3.0	1.4	0.2	Iris-setosa
3.2	1.3	0.2	Iris-setosa
3.1	1.5	0.2	Iris-setosa
3.6	1.4	0.2	Iris-setosa
3.9	1.7	0.4	Iris-setosa
3.4	1.4	0.3	Iris-setosa
4	1.5	0.2	Iris-setosa
4	1.4	0	Iris-setosa

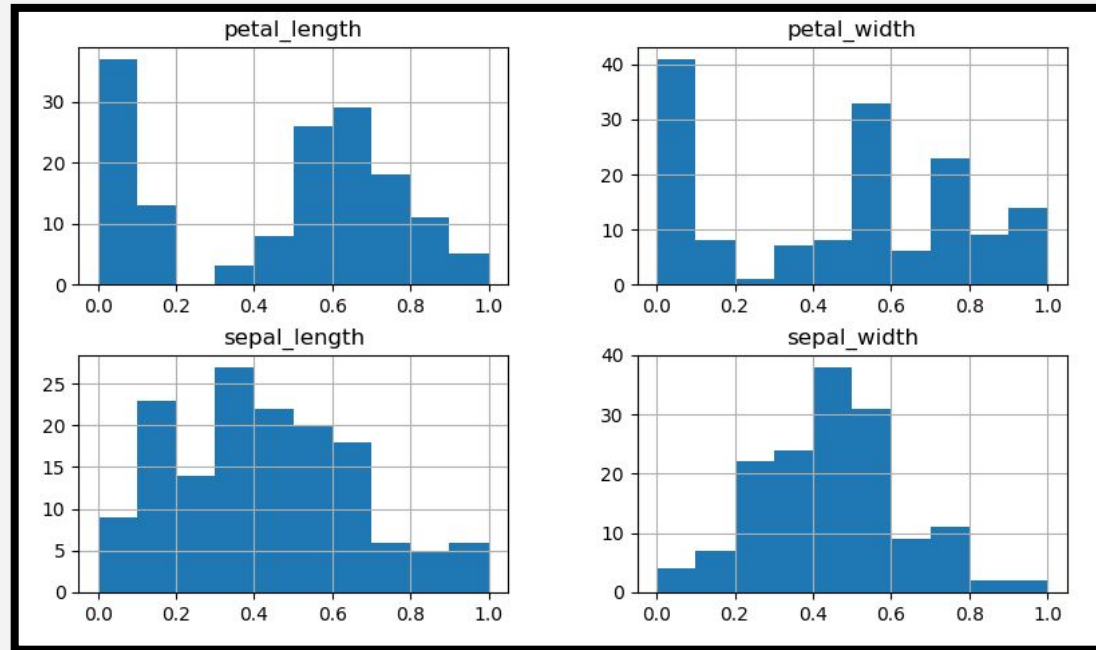
# Normalización de datos

Para los modelos que veremos es un problema que cada *feature* posea su propio rango de datos. Necesitamos normalizarlos de algún modo



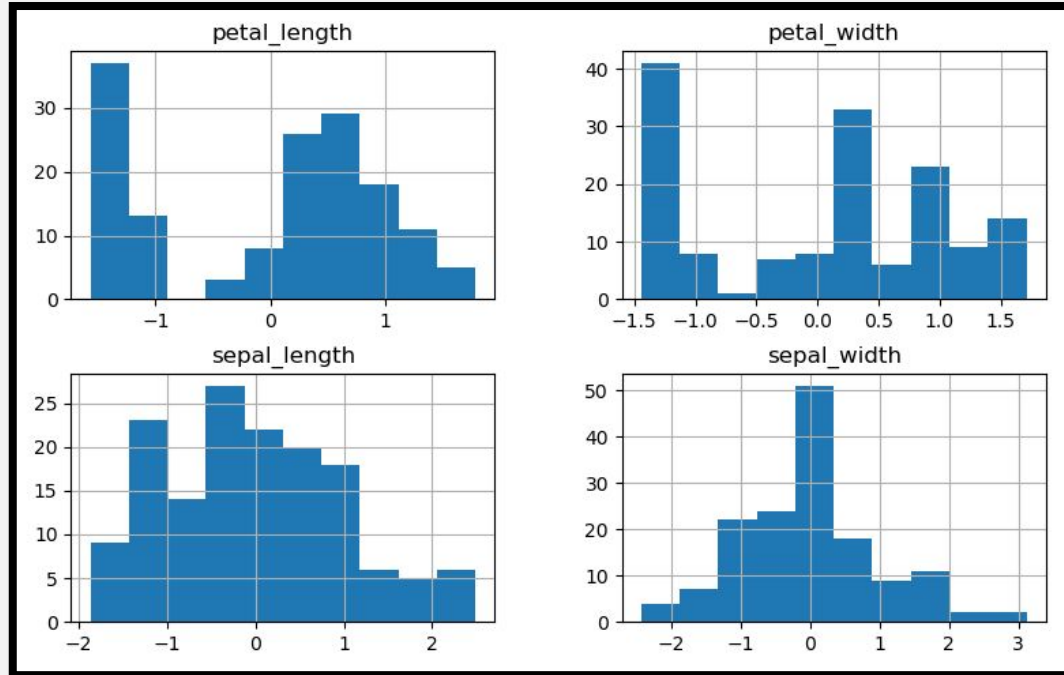
# Normalización de datos

**Min-max (0-1):**  $x = (x - \min(x)) / (\max(x) - \min(x))$



# Normalización de datos

**Z-score ( $\mu$  y  $\sigma$ ):** 
$$x = (x - \text{mean}(x)) / \text{std}(x)$$

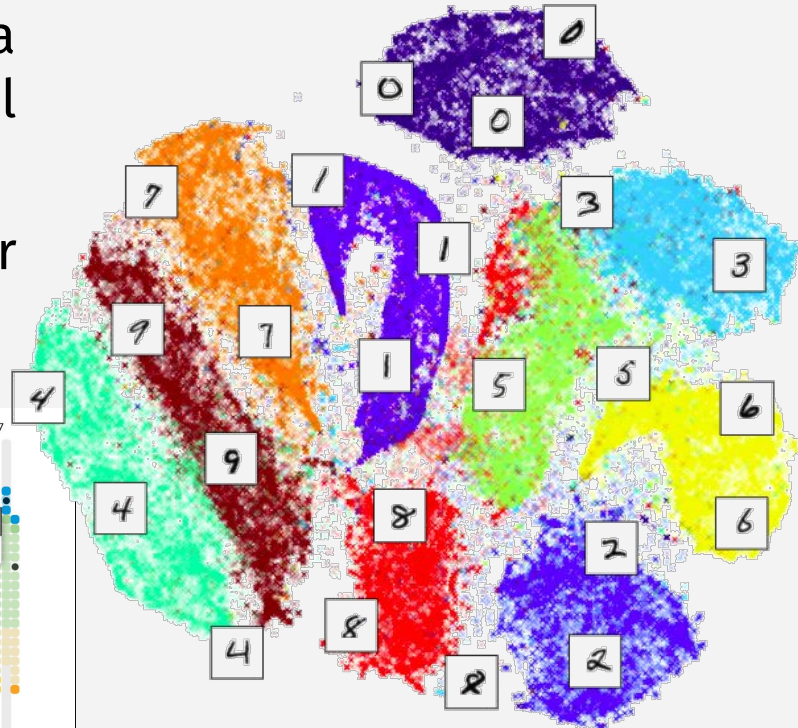
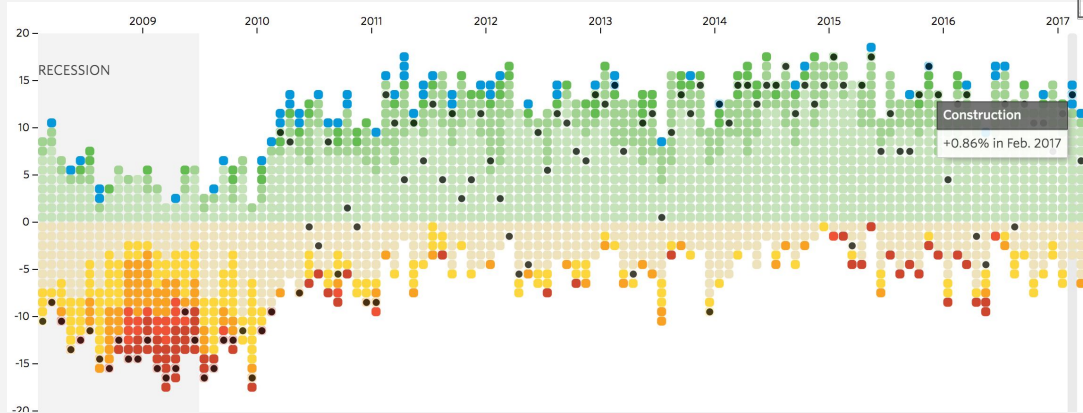




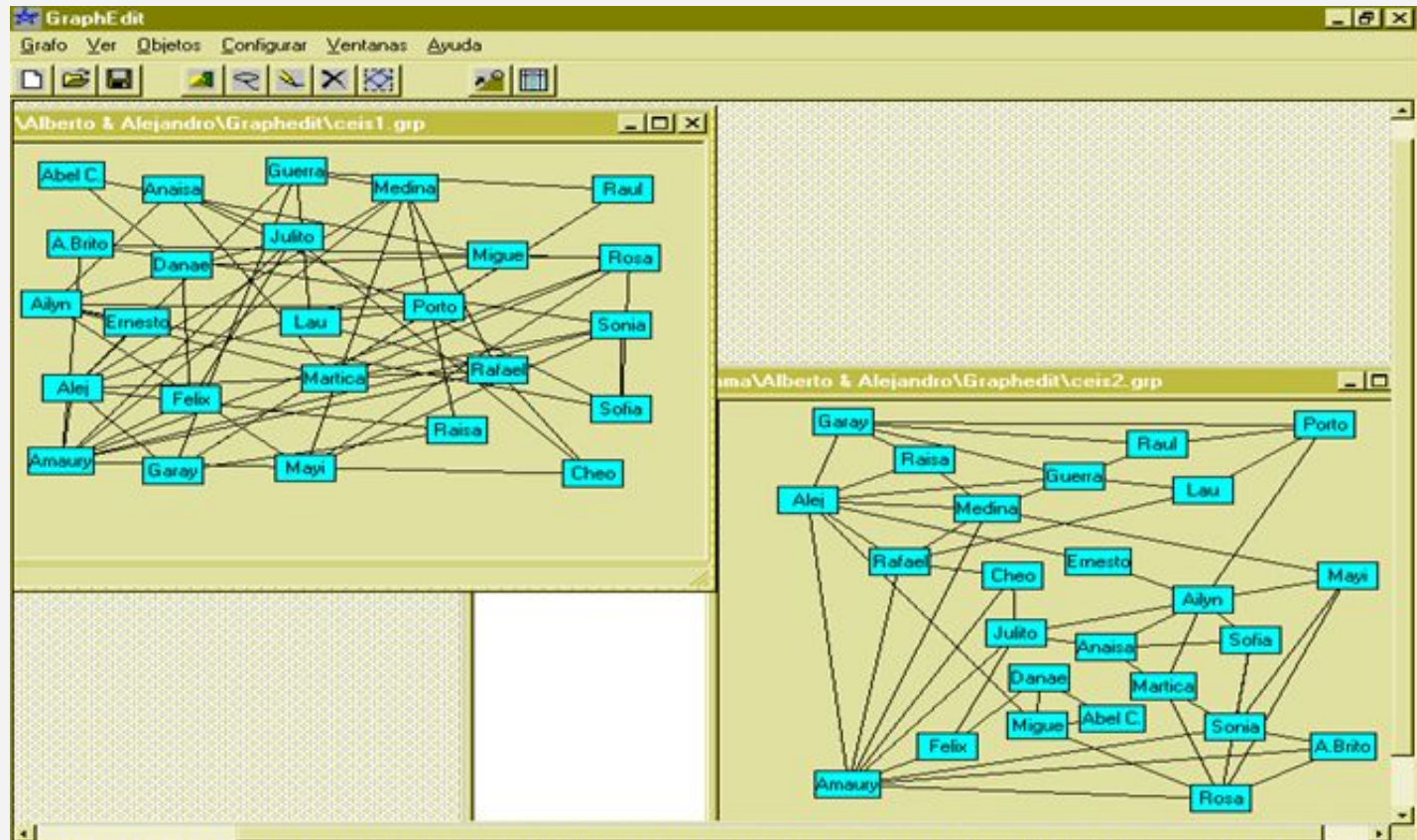
# Visualización

Visualizar los datos puede ser una tarea esencial para entender el dominio en el que estamos trabajando.

Al tener más de 2 variables, visualizar puede no ser trivial.

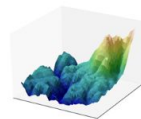
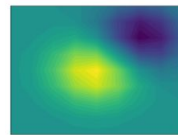
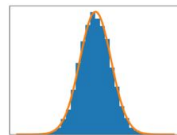
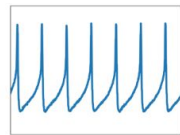


# Visualización





# Visualización



Es la librería gráfica más utilizada para realizar gráficos en Python.

<https://matplotlib.org/tutorials>

```
import matplotlib.pyplot as plt
```

# Visualización

## matplotlib.pyplot.plot

La función `plot(x,y)` permite realizar un gráfico 2D de la variable `y` en función de `x`.

Copie y pegue el siguiente código en un nuevo documento de Python.

```
import numpy as np
import matplotlib.pyplot as plt
x= np.arange(0, 360)
y= np.sin(x * np.pi / 180.)
plt.plot(x,y)
```

# Visualización

Matplotlib permite adornar las figuras con mucha información adicional.

Por ejemplo:

```
plt.xlabel('ángulo')  
plt.ylabel('seno(x)')  
plt.title('Seno de x entre 0 y  
360°')  
plt.grid()  
plt.savefig("figura.png")
```

`plt.figure()`

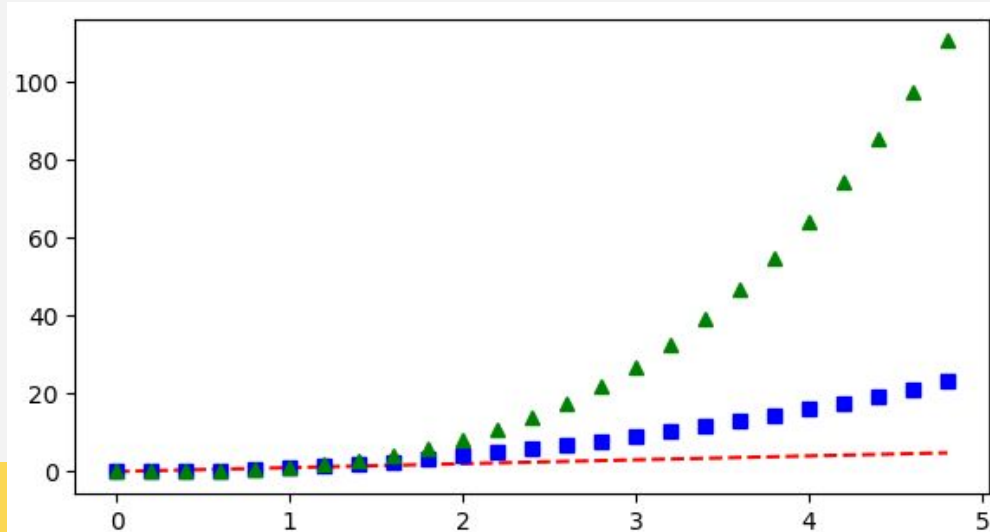


Este comando permite crear una figura nueva (nueva ventana). De lo contrario, todo lo ejecutado por *matplotlib* será en la figura activa.

# Visualización

También podemos omitir graficar las líneas, y visualizar sólo los puntos que pasamos como argumento.

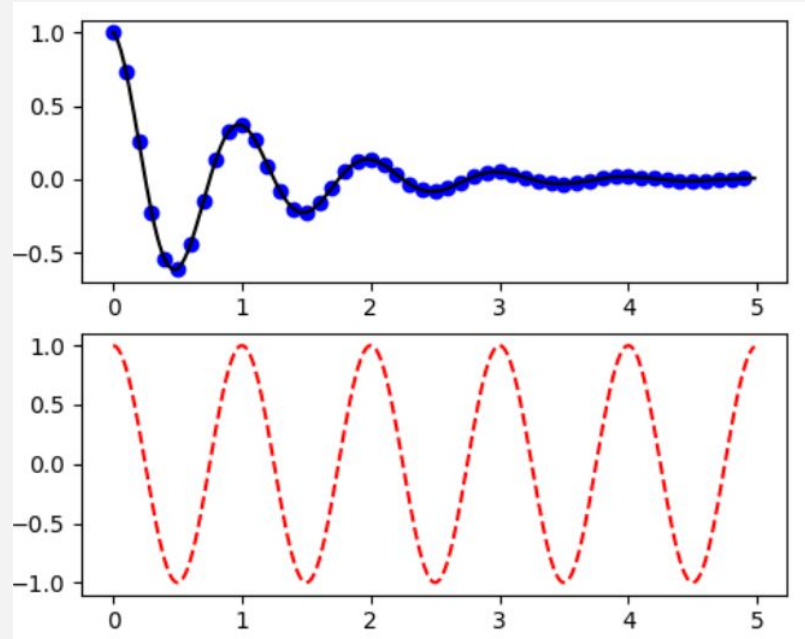
```
plt.figure()  
# rango de datos  
t = np.arange(0., 5., 0.2)  
# líneas rojas, cuadrados azules y triángulos verdes  
plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')  
plt.show()
```



# Visualización

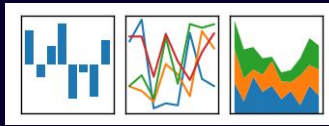
## Graficando en varias subfiguras.

```
def f(t):  
    return np.exp(-t) * np.cos(2*np.pi*t)  
  
t1 = np.arange(0.0, 5.0, 0.1)  
t2 = np.arange(0.0, 5.0, 0.02)  
  
plt.figure(1)  
plt.subplot(211) # 2 x 1, indice=1  
plt.plot(t1, f(t1), 'bo', t2, f(t2), 'k')  
  
plt.subplot(212) # 2 x 1, indice=2  
plt.plot(t2, np.cos(2*np.pi*t2), 'r--')  
plt.show()
```





# Pandas



(<https://pandas.pydata.org/>)

- Pandas es una librería de código abierto, fácil de usar, que provee manejo para carga y análisis de datos, entre otras funcionalidad.
- Es una librería muy utilizada para cargar archivos CSV o XLS.
- Cada columna tiene nombres, y pueden contener información de diferente tipo.

```
import pandas as pd
```

```
iris = pd.read_csv("iris.csv")
```

```
iris_data= iris.values
```

Crea un DataFrame

Retorna los valores  
como un Array de  
NumPy

# Scatter plot

Scatter permite visualizar una dispersión de puntos en un espacio 2D.

```
import pandas as pd
import matplotlib.pyplot as plt

raw_data = {'nombre': ['Jason', 'Molly', 'Tina', 'Jake', 'Amy'],
            'apellido': ['Miller', 'Jacobson', 'Ali', 'Milner', 'Cooze'],
            'femenino': [0, 1, 1, 0, 1],
            'edad': [42, 52, 36, 24, 73],
            'altura': [180, 170, 165, 175, 158],
            'peso': [77, 80, 61, 90, 65]} # esto es un diccionario

df = pd.DataFrame(raw_data, columns = ['nombre', 'apellido', 'edad', 'femenino', 'altura', 'peso'])
```

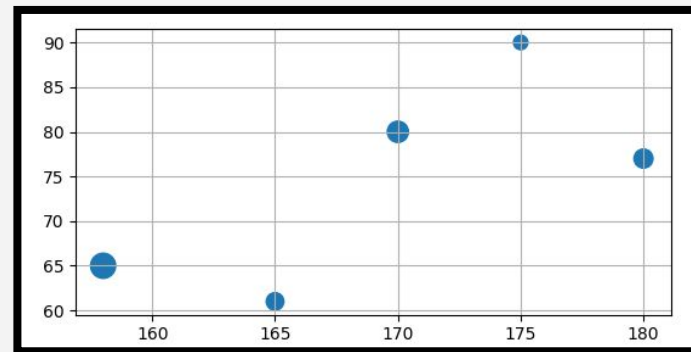
Diccionario de python

Crea un DataFrame con la información.

```
plt.figure()
plt.scatter(df.altura, df.peso, s=df.edad*3), plt.grid()
```

Los parámetros  
"x" e "y" son  
arreglos.

Grafica la altura en función  
del peso. El tamaño del  
punto es la edad



# Ejemplo: Iris dataset

El dataset “Iris” está almacenado en un archivo “csv” (Comma Separated Values). Estos son archivos de texto plano, donde cada registro está delimitado por un Enter (retorno de carro), y cada columna por una coma (,).

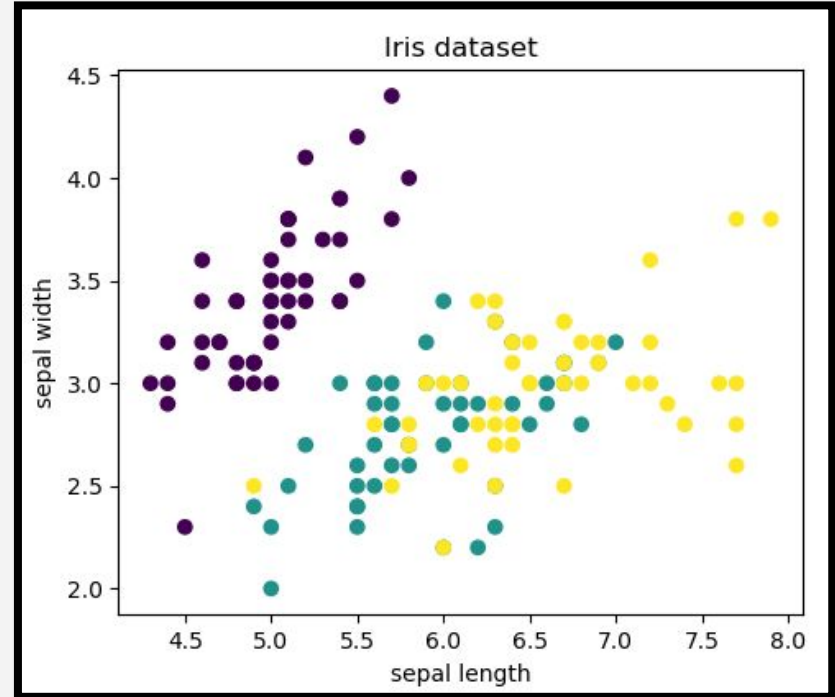
El dataset contiene información sobre 3 especies distintas de flores. Contiene 4 atributos: tamaño y largo del pétalo, tamaño y largo del sépalo.



# Visualización Iris

Al tener 4 variables, no podemos visualizar todas al mismo tiempo, pero podemos hacer un corte 2D para dos de ellas.

```
iris = pd.read_csv("iris.csv")
plt.figure()
plt.scatter( iris.sepal_length,
             iris.sepal_width,
             c=iris.name)
plt.title('Iris dataset')
plt.xlabel('sepal length')
plt.ylabel('sepal width')
```



# Visualización con Pandas

Pandas nos permite visualizar de forma rápida los datos con algunos comandos

```
iris.columns
```



**Visualizar los nombres de los *features*:**

*'sepal\_length', 'sepal\_width',  
'petal\_length', 'petal\_width', 'name'*

```
np.unique(iris.name)
```



**Visualizar los nombres de las clases:**

*'setosa', 'versicolor', 'virginica'*



# Visualización con Pandas

Pandas nos permite visualizar de forma rápida los datos con algunos comandos

```
iris.head()
```



**Si estamos en una consola, con este comando podemos ver los primeros datos en el dataframe**

	sepal_length	sepal_width	petal_length	petal_width	name
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

# Visualización con Pandas

Nombre	Tipo	Tamaño	Valor
ax	axes._subplots.Axes...	1	AxesSubplot object of matplotlib.axes._subplots module
blue	tuple	3	(0.16696655132641292, 0.48069204152249134, 0.7291503267973857)
class_number	int64	1	2
data	Array of float64	(150, 5)	<pre>[[-0.90068117  1.03205722 -1.3412724 -1.312  [-1.14 ...</pre>
f	figure.Figure	1	Figure object of matplotlib.figure module
fig	figure.Figure	1	Figure object of matplotlib.figure module
iris	DataFrame	(150, 5)	Column names: sepal_length, sepal_width, petal_length, petal_width, na

Explorador de variables   Gráfico

**Visualización del dataframe con Spyder**

Index	sepal_length	sepal_width	petal_length	petal_width	name
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3	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	3.6	1.4	0.2	setosa
5	5.4	3.9	1.7	0.4	setosa
6	4.6	3.4	1.4	0.3	setosa

# Visualización con Pandas

```
iris.corr()
```



**Matriz de correlación entre features**

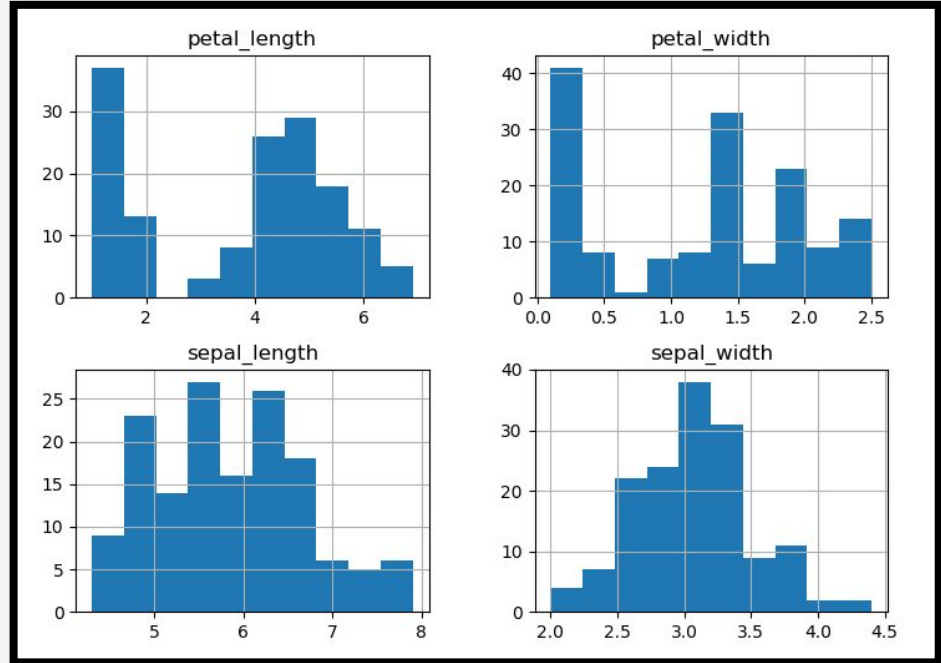
Index	sepal_length	sepal_width	petal_length	petal_width
sepal_length	1	-0.109369	0.871754	0.817954
sepal_width	-0.109369	1	-0.420516	-0.356544
petal_length	0.871754	-0.420516	1	0.962757
petal_width	0.817954	-0.356544	0.962757	1

# Visualización con Pandas

```
iris.hist()
```



Crea un histograma para cada columna del dataframe

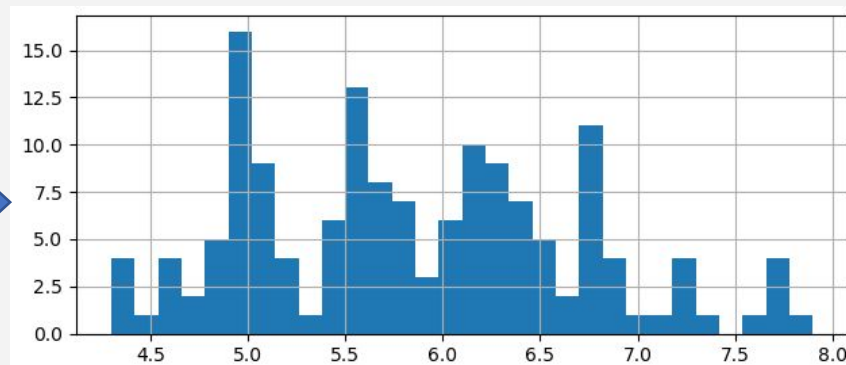


# Visualización con Pandas

Pandas nos permite visualizar de forma rápida los datos con algunos comandos

```
iris['sepal_length'].hist(bins=30)
```

Otro modo de indexar el dataframe:  
`iris.sepal_length.hist(bins=30)`

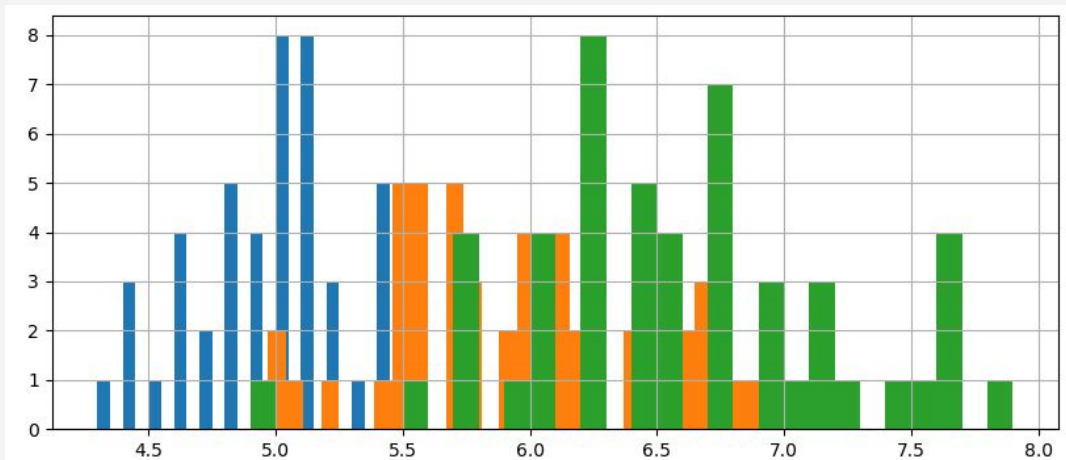


**Crea un histograma para la columna  
"sepal\_length"**



# Visualización con Pandas

```
for class_number in np.unique(iris.name):  
    iris['sepal_length'].iloc[np.where(iris.name == class_number)[0]].hist(bins=30)
```

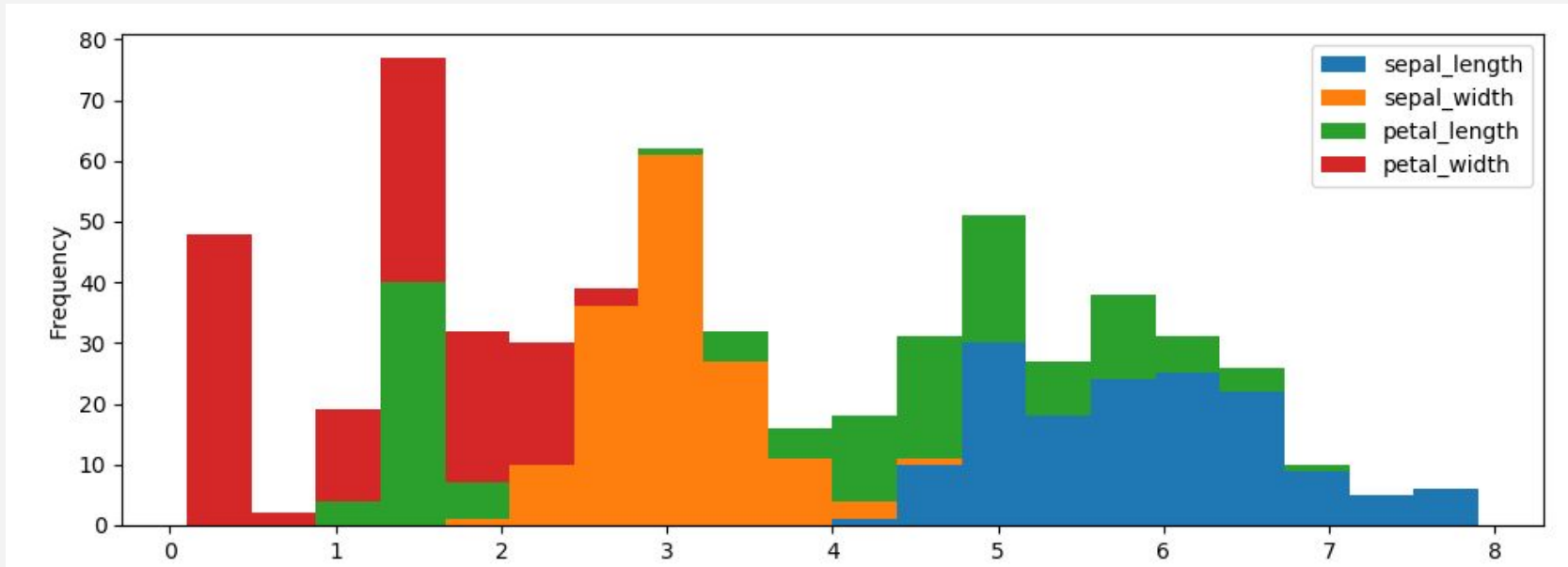


**Crea un histograma de  
“sepal\_length” para cada  
clase**

# Visualización con Pandas

```
iris.plot.hist(stacked=True, bins=20)
```

Crea un histograma para cada feature

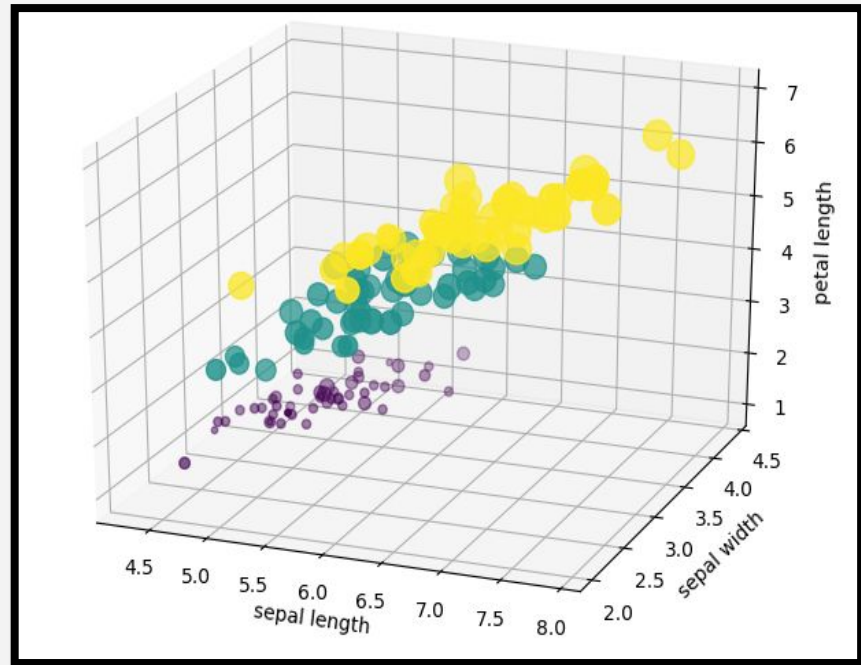


# Visualización con Pandas

## Ejemplo de visualización 3D:

```
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter( iris.sepal_length,
            iris.sepal_width,
            iris.petal_length,
            c=iris.name,
            s=iris.petal_width*100 )

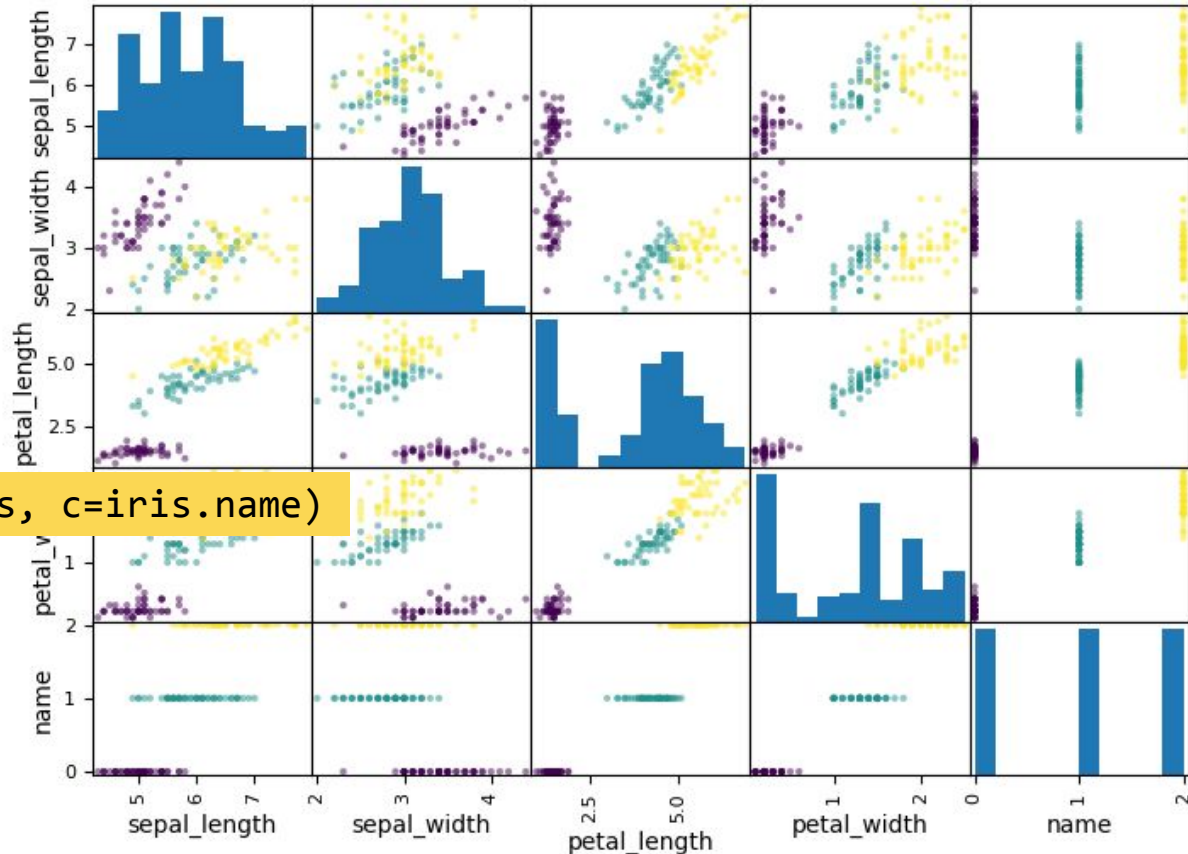
plt.title('Iris dataset')
ax.set_xlabel('sepal length')
ax.set_ylabel('sepal width')
ax.set_zlabel('petal length')
```



# Visualización con Pandas

**Matriz con scatter 2D para cada  
par de features  
(Cuidado si son muchos features !)**

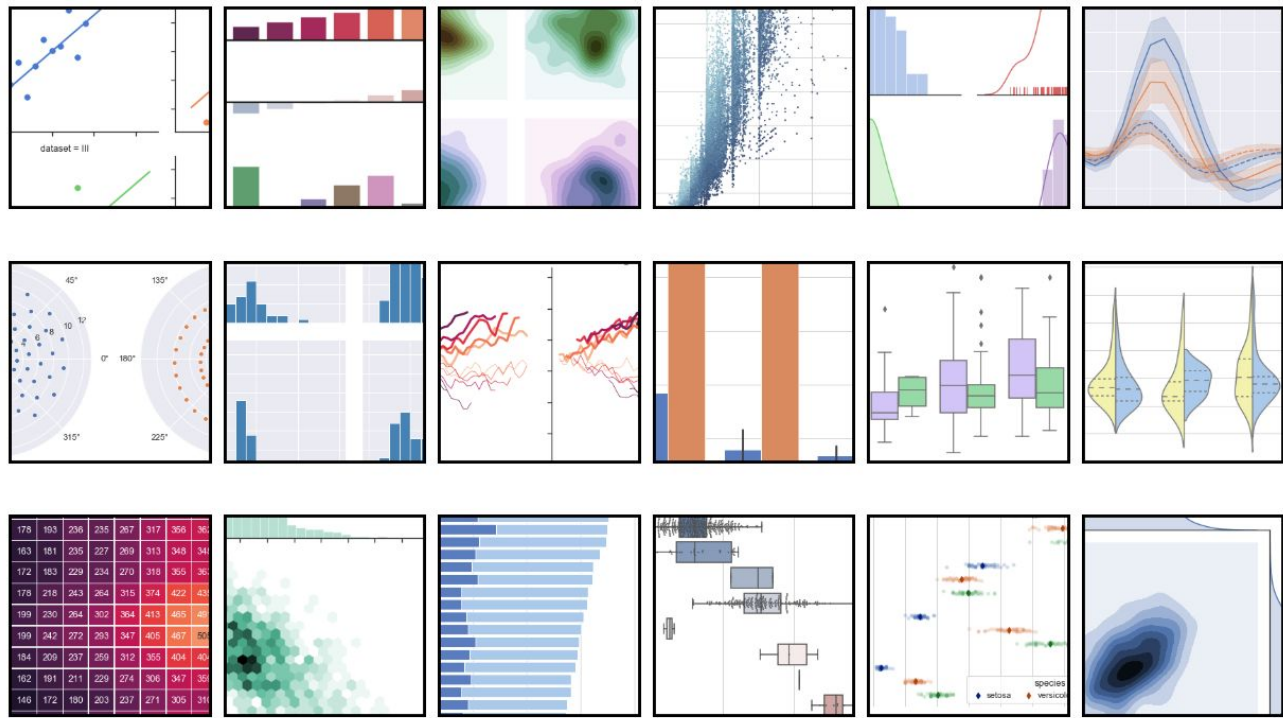
```
pd.plotting.scatter_matrix(iris, c=iris.name)
```



# Visualización con Seaborn

<https://seaborn.pydata.org/>

<https://www.kaggle.com/nolan/elano/seaborn-visualization-on-iris-data-set>

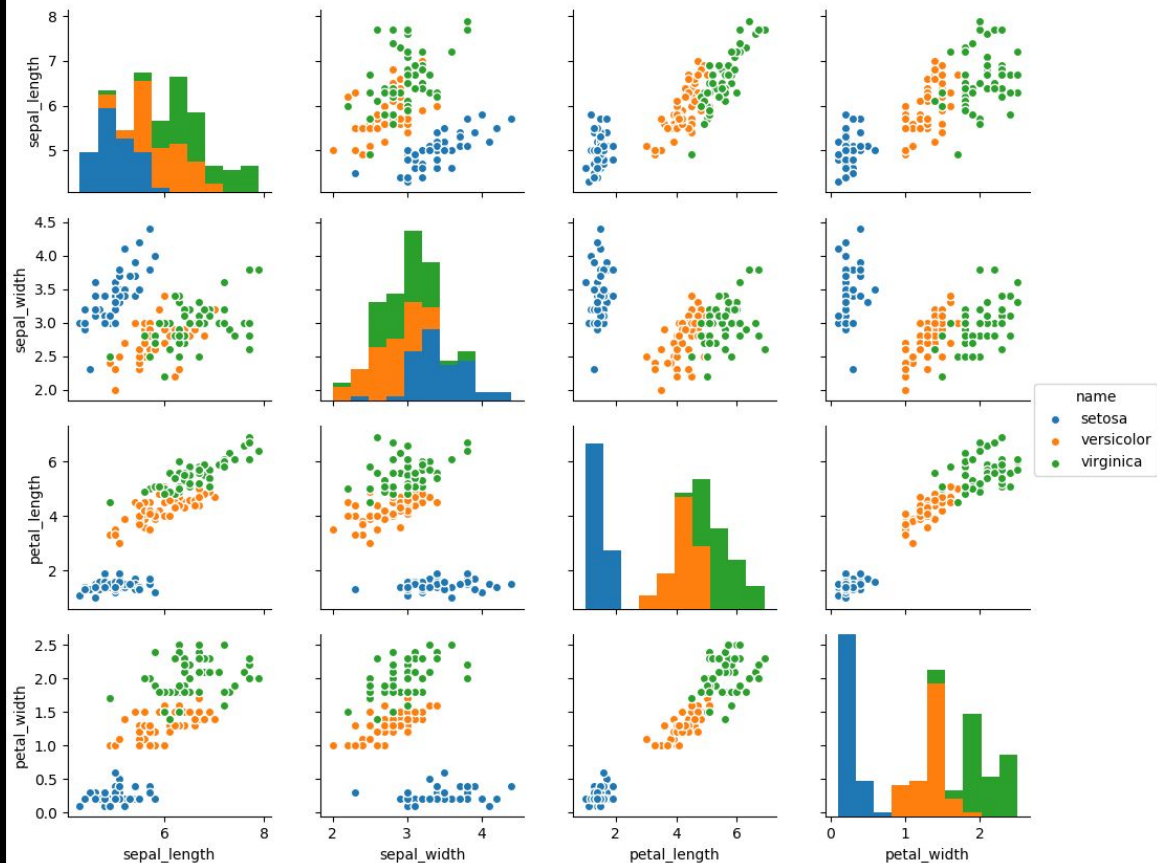




# Visualización con Seaborn

```
sns.pairplot(iris, hue='name')
```

**Iris Dataset**  
pairplot con SeaBorn  
(igual a scatter\_matrix)



# Visualización con Seaborn

```
setosa = iris.query("name == 'setosa'")
virginica = iris.query("name == 'virginica'")

# Set up the figure
f, ax = plt.subplots(figsize=(8, 8))
ax.set_aspect("equal")

# Draw the two density plots
ax = sns.kdeplot(setosa.sepal_width,
setosa.sepal_length, cmap="Reds", shade=True,
shade_lowest=False)
ax = sns.kdeplot(virginica.sepal_width,
virginica.sepal_length, cmap="Blues", shade=True,
shade_lowest=False)

# Add labels to the plot
red = sns.color_palette("Reds")[-2]
blue = sns.color_palette("Blues")[-2]
ax.text(2.5, 8.2, "virginica", size=16, color=blue)
ax.text(3.8, 4.5, "setosa", size=16, color=red)
```

kernel density estimate  
(univariado o bivariado)

