AWAKELAB

# BASECAMP

## Ciencia de Datos

**Análisis Exploratorio y Programación Estadística**

## Objetivo de la jornada

- Presentar información de un set de datos utilizando librería Seaborn para graficar la información.

**Librerías Seaborn.**

Seaborn es una librería de visualización de datos para Python desarrollada sobre **matplotlib**. Ofrece una interfaz de alto nivel para la creación de atractivas gráficas. Además, está íntimamente integrada con las estructuras de datos de **pandas**, lo que permite utilizar el nombre de los DataFrames y campos directamente como argumentos de las funciones de visualización.

Seaborn tiene como objetivo convertir la visualización en una parte central de la exploración y comprensión de los datos, generando atractivas gráficas con sencillas funciones que ofrecen una interfaz semejante, facilitando el paso de unas funciones a otras.

Esta librería se importa habitualmente con el alias **sns**.

```
import seaborn as sns
```

Para poder hacer uso de las funciones ofrecidas por seaborn deberás importar previamente la librería con la instrucción anterior.

```
import seaborn as sns

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

import warnings


np.random.seed(18) # para obtener los mismos valores
warning.fillterwarnings("ignore")

sns.set()
```

Junto a seaborn se importan también las librerías **matplotlib** (pues, en ocasiones, hay que recurrir a ella para acceder a funcionalidad no ofrecida por seaborn), **pandas** y **NumPy** (en algunos ejemplos crearemos estructuras de datos basadas en estas librerías) y la librería **warnings** para ocultar ciertos mensajes de aviso que genera seaborn (al respecto de futuros cambios de funcionalidad).

En el código mostrado en la imagen anterior (celda **In [2]**) se inicializa el generador de números aleatorios de NumPy, se activa la ocultación de los avisos y se fija el estilo gráfico de seaborn.

**Características de la librería Seaborn**
Seaborn ofrece varias características destacadas:

- Funciones con una API estandarizada que limita la necesidad de tener que memorizar conjuntos de parámetros diferentes en función del gráfico deseado:

```
sns.catplot(x = "day", y = "total_bill", kind =
"swarm", data = tips)
```

- Gráficas visualmente atractivas sin necesidad de realizar complicados ajustes.
- Una API orientada a conjuntos de datos para examinar la relación entre múltiples variables.
- Opciones para mostrar la distribución de variables univariadas y bivariadas.
- Cálculo automático y dibujo de modelos de regresión lineal para diferentes tipos de variables dependientes
- Herramientas para mostrar la estructura de datasets complejos
- Abstracciones de alto nivel para estructurar rejillas multigráficas con el objetivo de poder crear complejas visualizaciones.
- Sencillo control sobre los estilos gráficos disponibles
- Herramientas para la elección de paletas de color adecuadas que permitan revelar patrones en los datos

**Tipos de gráficos**

Gráficos de distribución de observaciones

*Distplot(Histograma)*

```
distplot(a, bins = None, hist = True, kde = True, rug
= False, fit = None, hist_kws = None, kde_kws = None,
rug_kws = None, fit_kws = None, color = None,
vertical = False, norm_hist = False, axlabel = None,
label = None, ax = None)
```

**a**: Serie, matriz unidimensional o lista. Datos observados. Si se trata de un objeto Serie con un atributo "nombre", el nombre se utilizará para marcar el eje de datos.
hist: bool, si se debe dibujar un histograma (estándar).

**bins**: número de bins

**kde**: bool, si se debe dibujar una estimación de densidad del kernel gaussiano.
alfombra: bool. Ya sea para dibujar un mapa de alfombra en el eje de soporte. Controlar si generar pequeñas tiras de valores observados (manta marginal)

**fit:** controla el gráfico de distribución de parámetros ajustados, que puede evaluar intuitivamente su relación correspondiente con los datos observados (la línea negra es la distribución determinada).

**hist_kws**: dict, opcional. Parámetros de palabras clave: meth: 'matplotlib.axes.Axes.hist'.

**kde_kws**: dict, opcional. El argumento de palabra clave de kdeplot.
rug_kws: dict, opcional. El argumento de palabra clave de rugflot.
color: matplotlib color, opcional. El color se usa para dibujar todo excepto la curva ajustada.

**vertical**: bool, opcional. Si es Verdadero, la observación está en el eje y.

**norm_list**: si es True, la altura del histograma muestra la densidad en lugar del recuento. Si se traza KDE o la densidad ajustada, significa que esto está implícito.

**axlabel**: string, False o None (opcional). El nombre de la etiqueta del eje de soporte. Si es None, intente obtenerlo de a.name; si es False, no

establezca la etiqueta.

**etiqueta**: cadena, opcional. Etiqueta de leyenda para los componentes relevantes del gráfico.

**ax**: matplotlib axis, opcional. Si se proporciona, grafique en ese eje.

```
np.random.seed(0)
x = np.random.randn(100)
ax = sns.distplot(x, hist = False, rug = True)
```

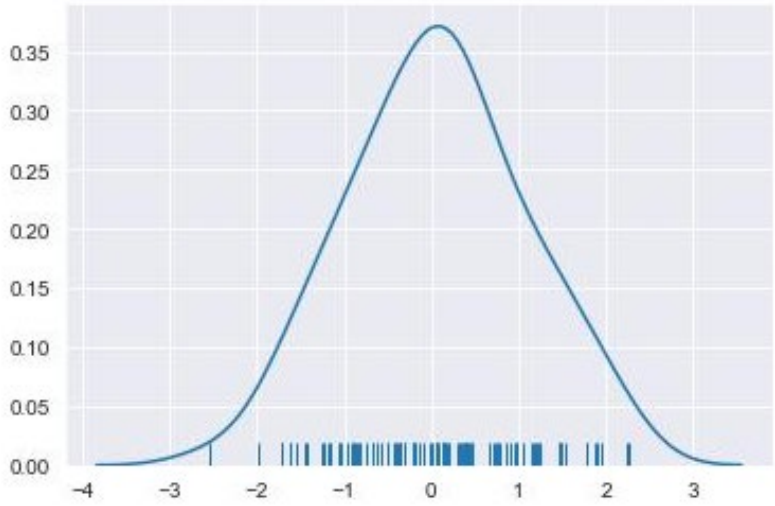
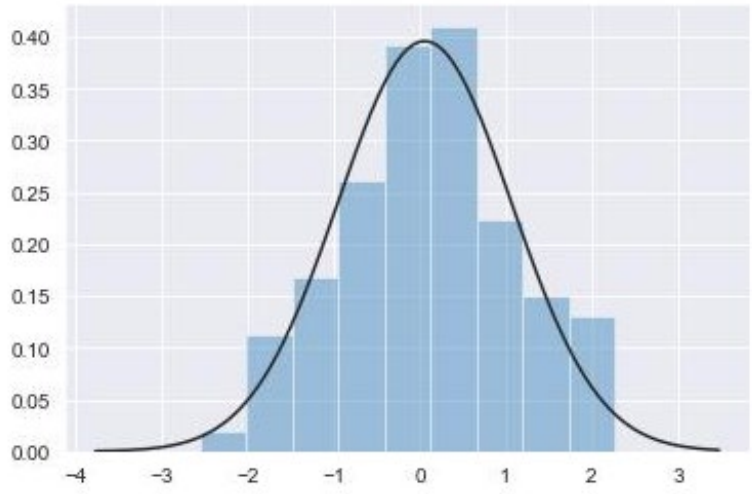
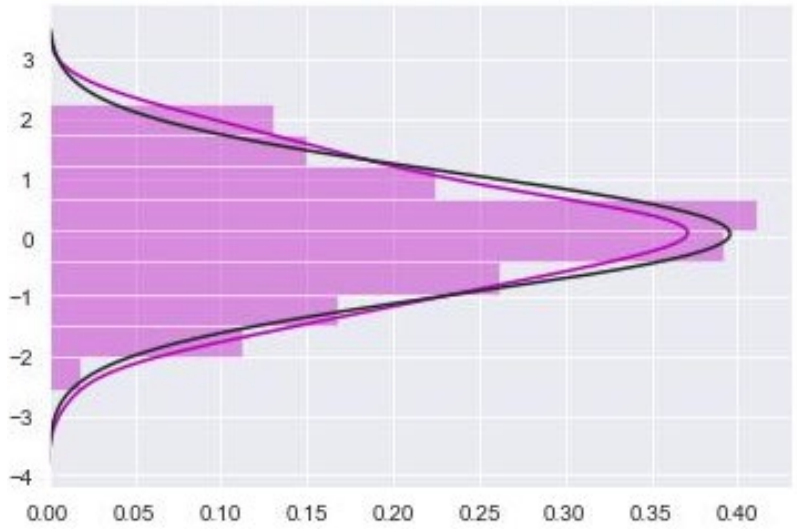



*imagen:fuente propia*

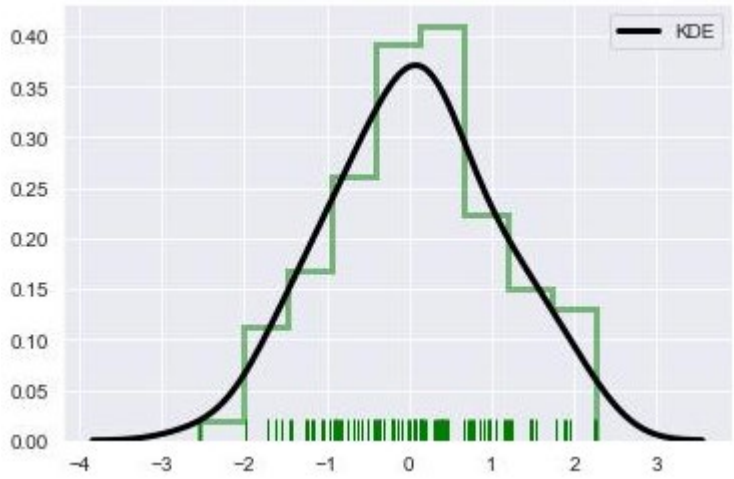
```
from scipy.stats import norm
ax = sns.distplot(x, fit = norm, kde = False)
```




```
ax = sns.distplot(x, vertical = True, fit = norm,
color = "m")
```



```
ax = sns.distplot(x, rug = True, rug_kwn = {"color":
"g"}, kde_kws = {"color": "k", "lw":3, "label":KDE},
hist_kws = {"histtype": "step", "lw":3, "alpha": 0.5.
"color": "g"})
```




Gráficos de dispersión y correlación de Variables

*Jointplot*

La distribución de probabilidad conjunta se abrevia como distribución conjunta, que es la distribución de probabilidad de un vector aleatorio compuesto por dos o más variables aleatorias.

Según diferentes variables aleatorias, la representación de la distribución de probabilidad conjunta también es diferente.

Para variables aleatorias discretas, la distribución de probabilidad conjunta se puede expresar en forma de lista o función; para las variables aleatorias continuas, la distribución de probabilidad conjunta está integrada por una función no negativa.

```
sns.joinplot(x, y, data = None, kind = 'scatter',
stat_func = None, color = None, height = 6, ratio =
5, space = 0.2, dropna = True, xlim = None, ylim =
None, joint_kws = None, marginal_kws = None,
annot_kws = None, **kwars)
```

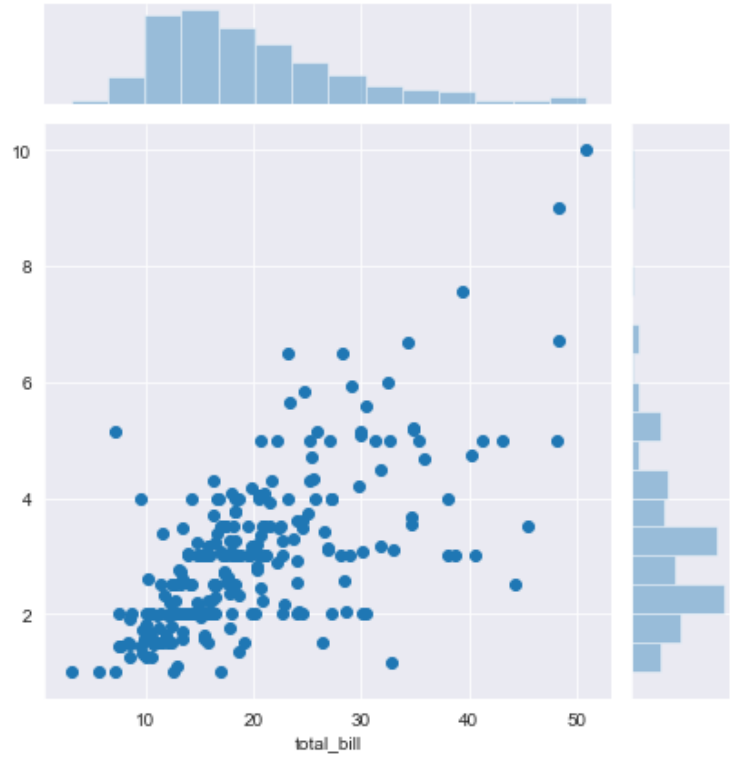
**x, y**: es el nombre de la columna o dos conjuntos de datos en el marco de datos, los datos apuntan al marco de datos;

**tipo**: {"scatter" | "reg" | "resid" | "kde" | "hex"}. Gráfico de dispersión predeterminado;

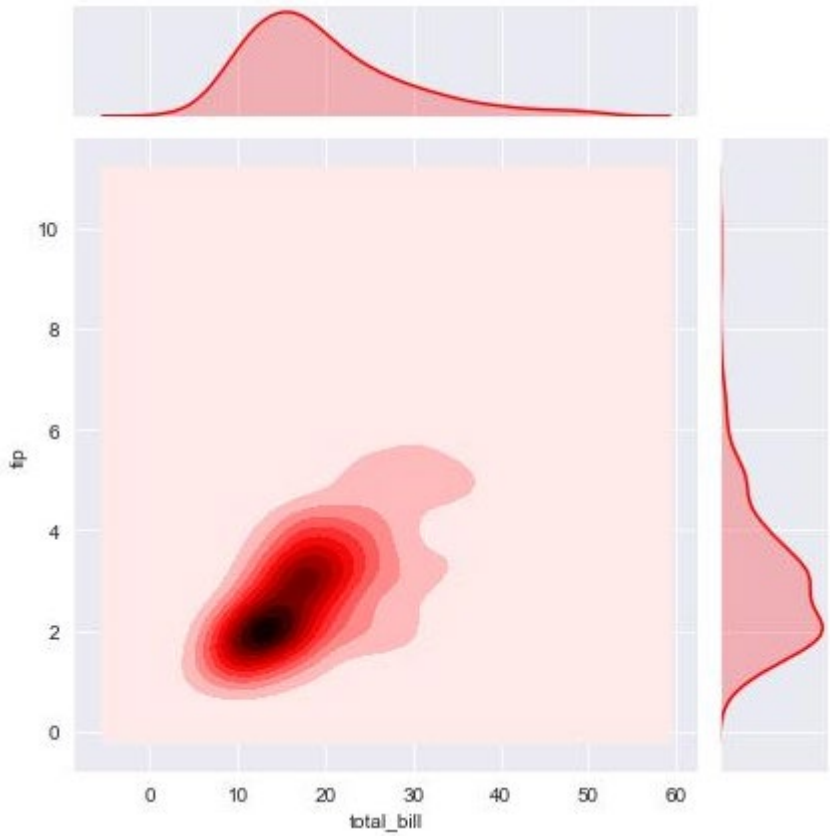
**stat_func**: una función que se utiliza para calcular la relación entre las estadísticas;

**relación**: la relación entre la imagen central y la imagen lateral; cuanto mayor sea la relación, mayor será la relación de la imagen central;
**dropna**: elimina los valores faltantes;
**altura**: la escala de la imagen (cuadrado);
**espacio**: el espacio entre la imagen central y la imagen lateral;
**xlim, ylim**: rango de x, y

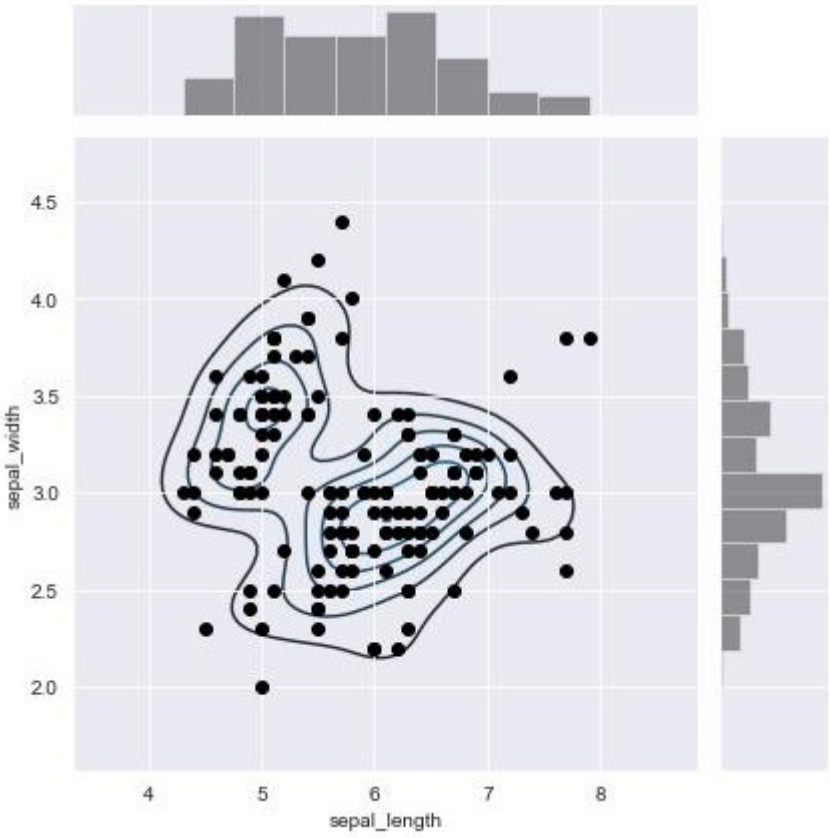
```
ax = sns.joinplot(x = "total_bill", y = "tip", data = tips)
```



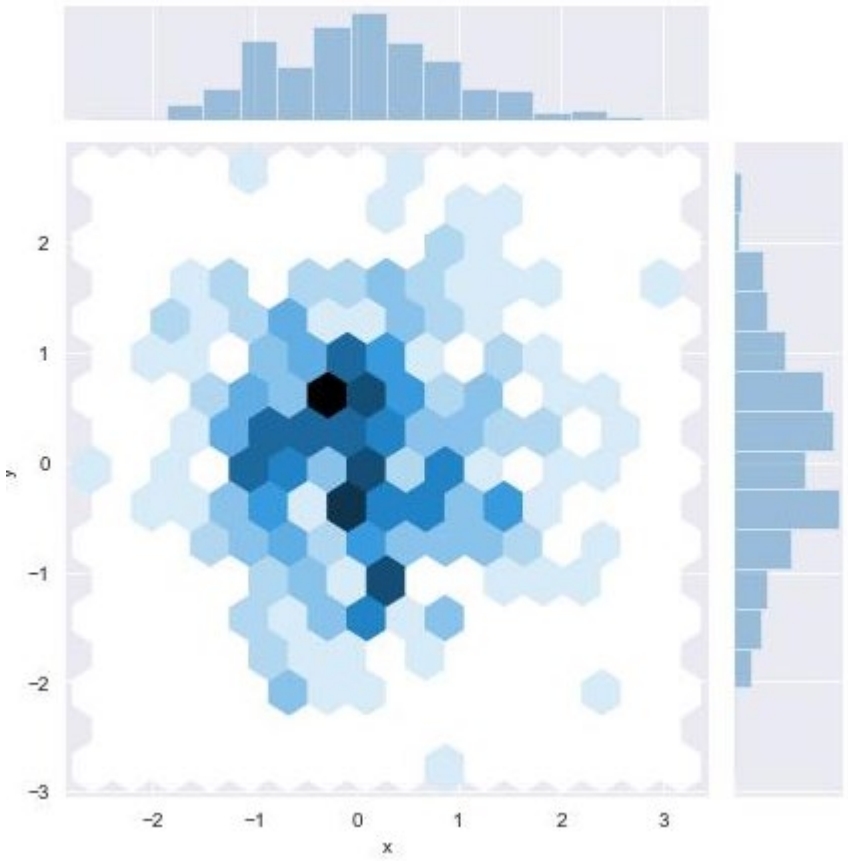
```
ax = sns.joinplot(x = "total_bill", y = "tip", kind =
"kde", data = tips, color = "r")
```



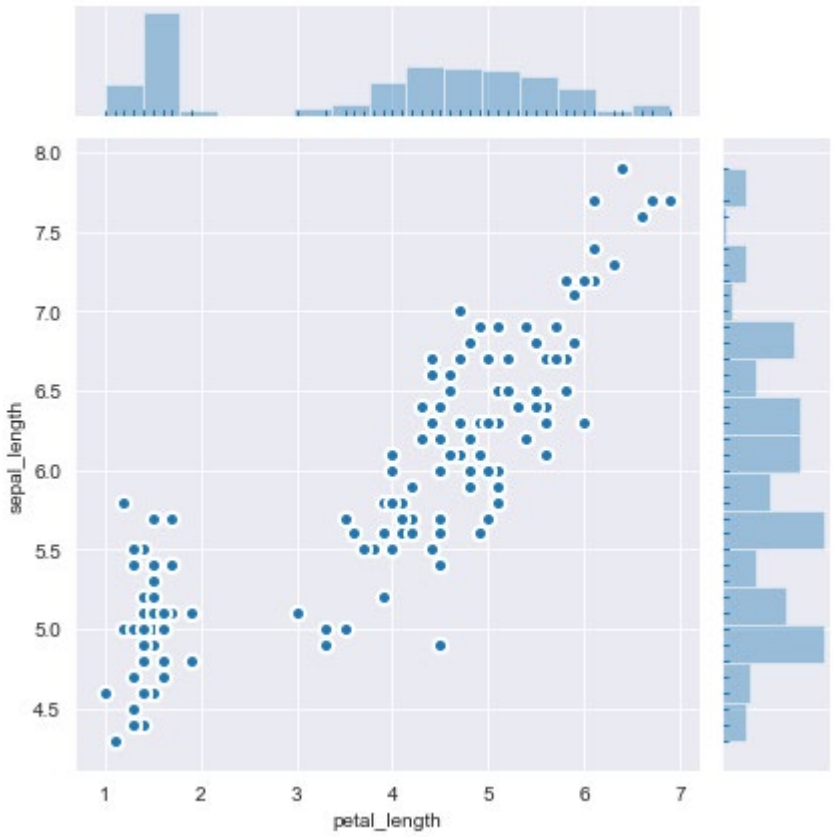
```
iris = pd.read_excel(r"C: \ Usuarios \ iris.xlsx)")

sx = (sns.joinplot("spal_length", "sepal_width", data
= iris, color = "k").plot_joint(sns.kdeplot, zorder =
0, n_levels = 6))
```



```
x, y = np.random.rands(2, 300)
axis = sns.jointplot(x, y, kind = 'hex'.
set_axis_labels('x', 'y'))
```



```
ax = sns.jointplot("petal_length", "sepal length",
data=iris, marginal_kws = dict(bins = 15, rug =
True), annot_kws = dict (stat=" r"), s = 50,
edgecolor="w", linewidth=2)
```

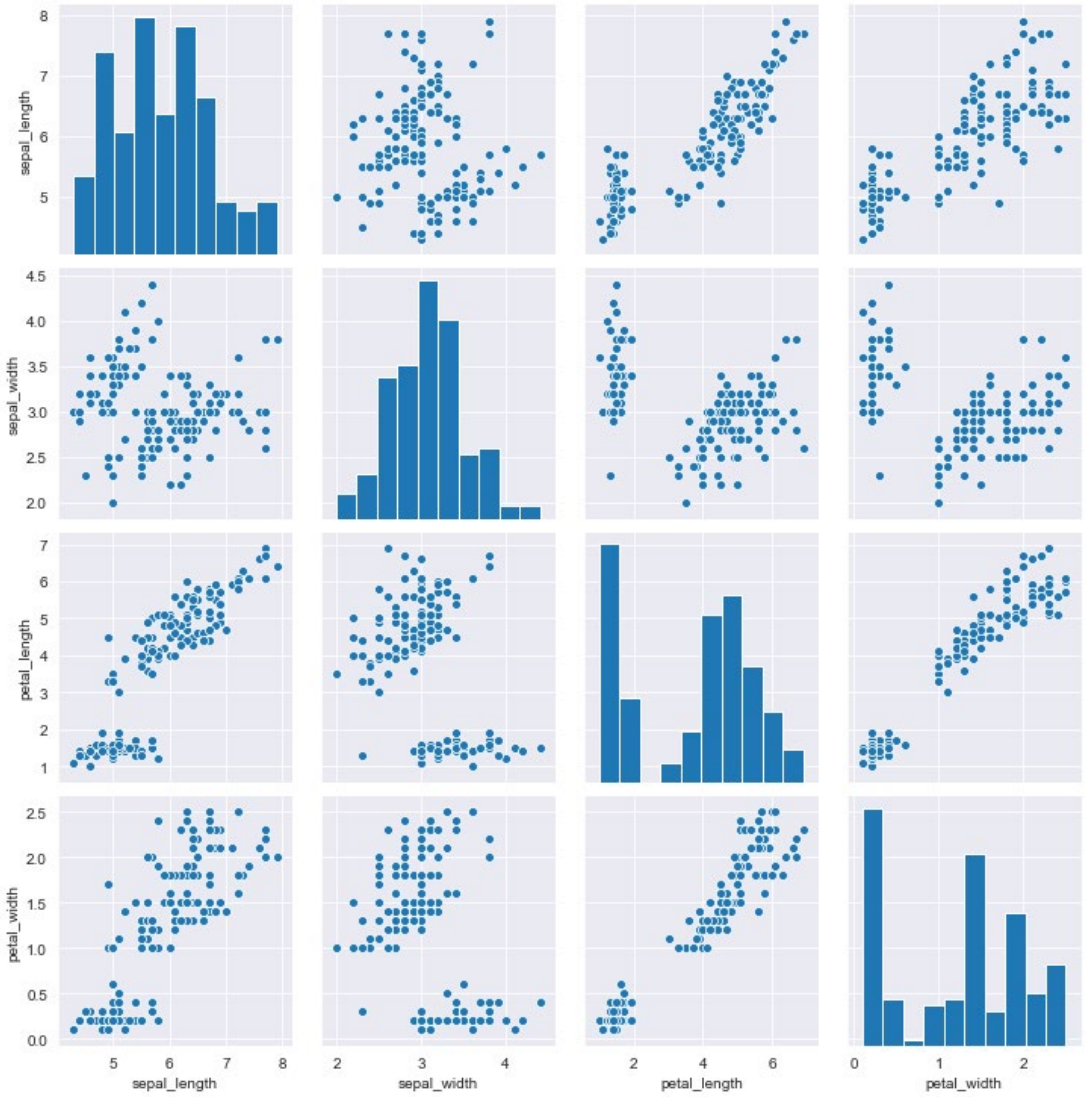


*Pairplot*

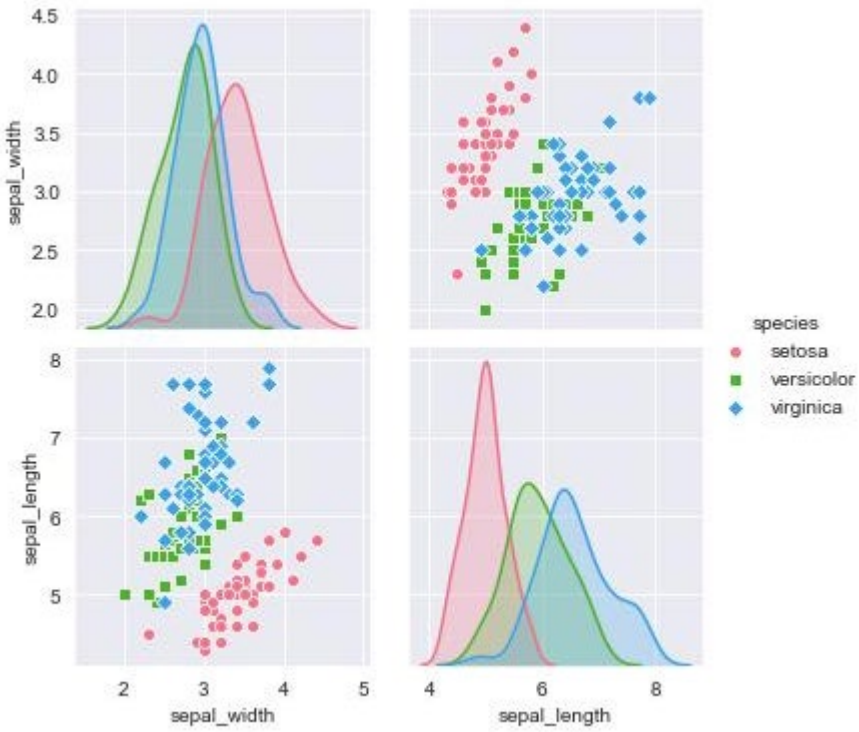
```
sns = pairplot (data, hue = None, hue_order = None,
palette = None, vars = None, x_vars = None, y_vars =
None, kind = 'scatter', diag_kind = 'auto', markers =
None, height=2.5, aspect = 1, corner = False, dropna
= True, plot_kws = None, diag_kws=None, grid_kws =
None, size = None)
```

**ars:** Lista de nombres de variables, las variables que se utilizarán en los datos; de lo contrario, cada columna utilizará el tipo de datos numéricos.
**{x, y} _vars**: lista de nombres de variables, las variables en "datos" se utilizan para las filas y columnas en la figura; es decir, para hacer una figura no cuadrada
**kind** : {‘scatter’, ‘reg’} ;
**diag_kind:** {‘auto’, ‘hist’, ‘kde’}. El dibujo de un gráfico univariado (comparándote contigo mismo), el patrón de un subgráfico diagonal. El valor predeterminado depende de si se usa "tono"
**{plot, diag, grid} _kws**: dicts. Diccionario de parámetros de palabras clave. Pase plot_kws a la función de dibujo de variable dual, pase diag_kws a la función de dibujo de variable única y pase grid_kws al constructor PairGrid.

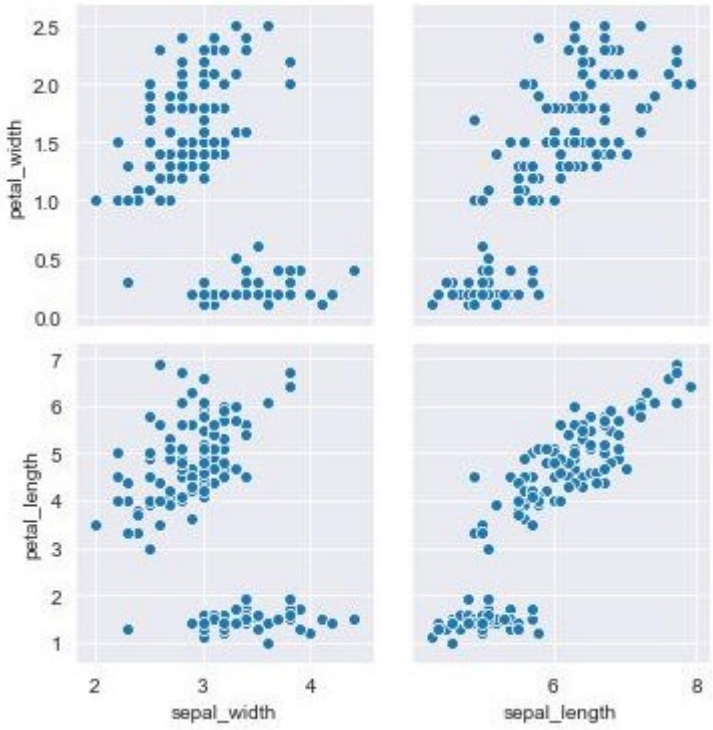
```
axis = sns.pairplot(iris)
```



```
axis = sns.pairplot(iris, hue = "species", vars =
["sepal_width", "sepal_length"], palette = "husl",
markers = ["o", "s", "D"])
```



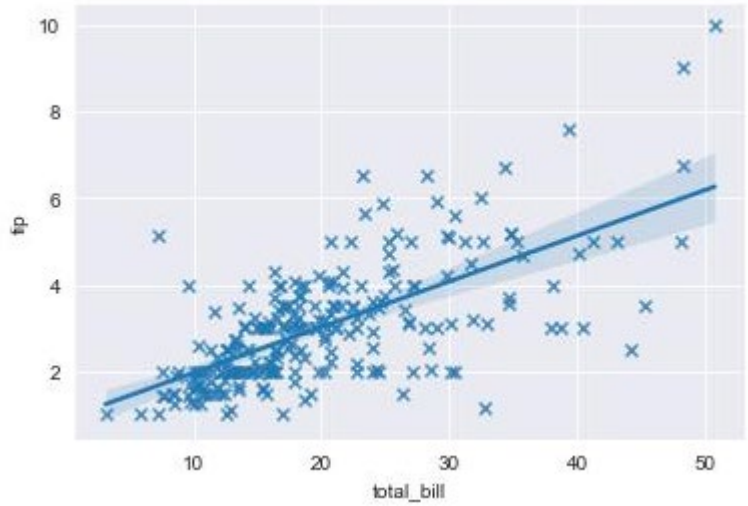
```
axis = sns.pairplot(iris, xvars = ["sepal_width",
"sepal_length"], yvars = ["pepal_width",
"pepal_length"])
```



Gráficos de regresiones

*Regplot*

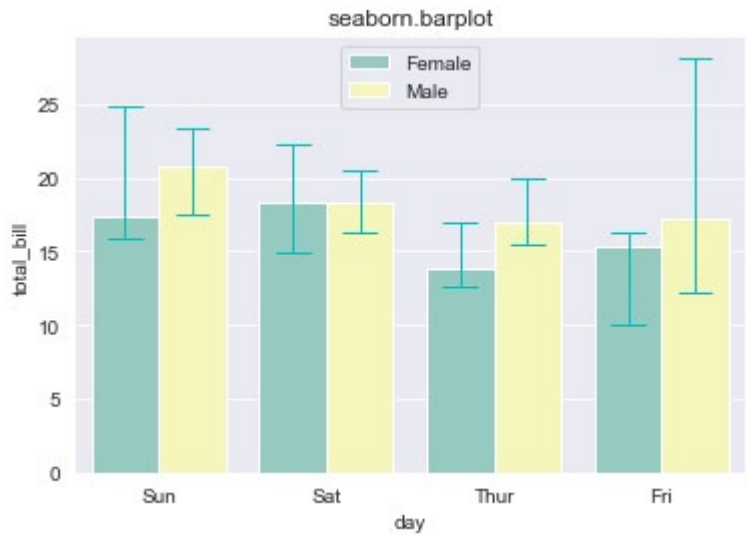
```
axis = sns.regplot(x = "total_bill", y = "tip", data
= tips, marker = "x")
```



Gráficos de variables categóricas

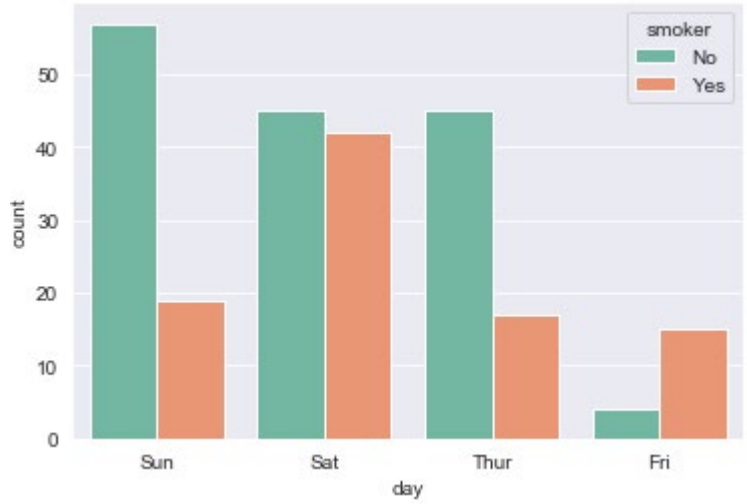
*Barplot (Gráfico de barras)*

```
ax = sns.barplot(x = "day", y = "total_bill", hue =
"sex", order = ["Sun", "Sat", "Thur", "Fri"], data =
tips, estimator = np.median, ci = 95, n_boot = 1000,
units = None, orient = None, color = None, palette =
"Set3", saturation = 0.75, errcolor = "c", errwidth =
1, capsize = 0.2, dodge = True, ax = None)


plt.lengend(loc = "best")
plt.title("seaborn.barplot")
plt.show()
```

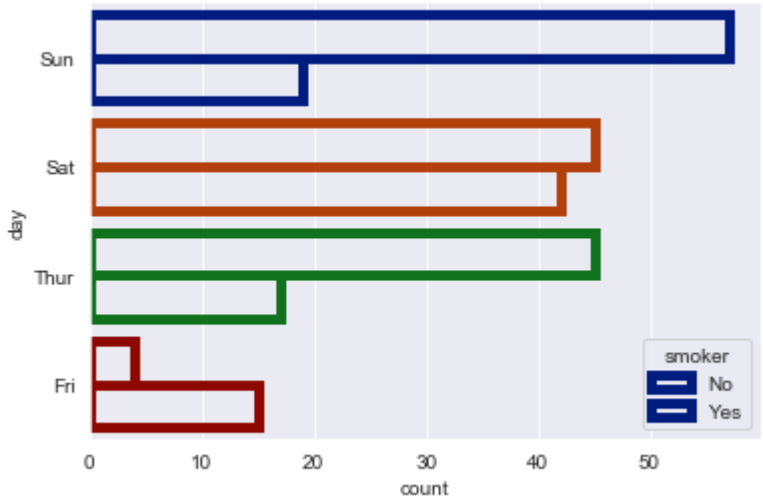


*Countplot (Gráfico de estadística de conteo)*

```
ax = sn.countplot(x = "day", hue = "smoker", data =
tips, palette = "Set2")
```




```
ax = sns.countplot(y = "day", hue = "smoker", data =
tips, facecolor = (0, 0, 0, 0,), linewidth = 5,
edgecolor = sns.color_palette("dark", 4))
```



*Boxplot (Diagrama de Cajas)*

```
boxplot(x = None, y = None, hue_order = None, order =
None, data = tips, ci = 95, n_boot = 1000, units =
None, orient = None, color = None, palette = None,
saturation = 0.75, width = 0.8, dodge = True,
fliersize = 5, whis = 1.5, ax = None, **kwargs)
```

**color**: el color de todos los elementos o la semilla de una paleta de degradado;

**Paleta** : los colores utilizados para los diferentes niveles de la variable `` matiz '';

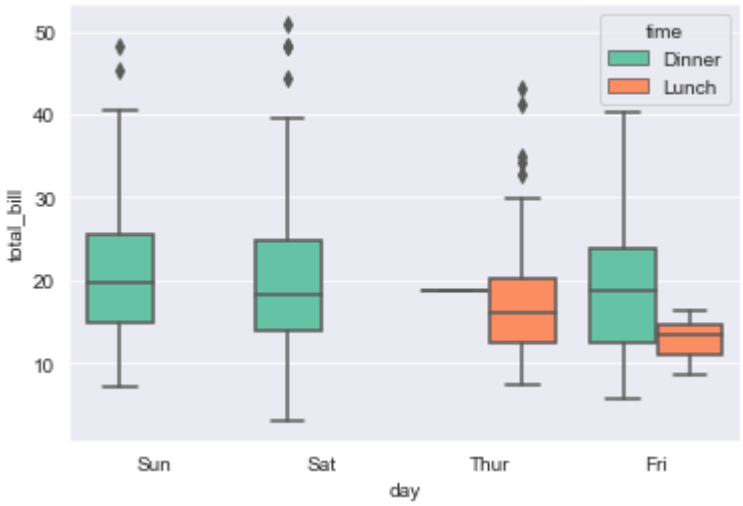
**saturación**: saturación, se puede establecer en 1;

**width**: float, controla el ancho del gráfico de caja;

**fliersize**: float, usado para indicar el tamaño del marcador de la observación de valores atípicos;

**whis**: La proporción de IQR que excede el cuartil inferior y el cuartil superior para expandir los bigotes de la trama. Los puntos fuera de este rango se reconocerán como valores atípicos.

```
ax = sns.boxplot(x = "day", y = "total_bill", hue =
"time", data = tips, saturation = 1.2, width = 0.8,
fliersize = 5, palette = "Set2")
```




*Violinplot (Diagrama de Violin)*

El diagrama de violín es similar al diagrama de caja y bigotes. Muestra la distribución de datos cuantitativos de una o más variables categóricas en múltiples niveles para que estas distribuciones se puedan comparar.

A diferencia del diagrama de caja, todos los componentes del diagrama de caja corresponden a puntos de datos reales. El diagrama de violín se caracteriza por la estimación de la densidad del núcleo de la distribución básica.

Esta puede ser una forma efectiva y atractiva de mostrar múltiples distribuciones de datos a la vez, pero el proceso de estimación se ve afectado por el tamaño de la muestra, y una muestra relativamente pequeña de violín puede ser engañosa.

```
sns.violinplot(data=None, *, x=None, y=None,
hue=None, order=None, hue_order=None, bw='scott',
cut=2, scale='area', scale_hue=True, gridsize=100,
width=0.8, inner='box', split=False, dodge=True,
orient=None, linewidth=None, color=None,
palette=None, saturation=0.75, ax=None, **kwargs)
```

**bw**: "scott", "silverman", flotar, controlar el grado de ajuste. Al calcular el ancho de banda del núcleo, puede hacer referencia al nombre de la regla ("scott", "silverman") o utilizar una proporción (flotante). El tamaño real del grano se determinará multiplicando la proporción por la desviación estándar de los datos en cada contenedor;

**cut**: La extensión del caparazón vacío excede la densidad del punto extremo, float;

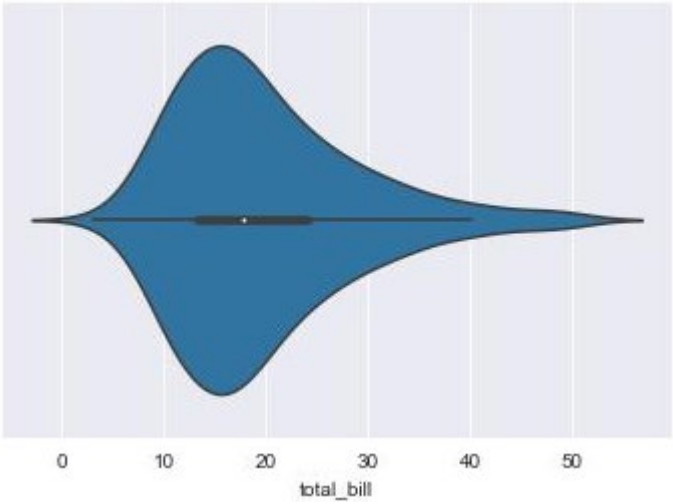
**scale:** "area", "count", "width", el método usado para escalar el ancho de cada violín; el método usado para escalar el ancho de cada violín. Si es "área", entonces cada violín tendrá la misma área. Si "cuenta", el ancho del violín se escalará de acuerdo con el número de observaciones en el contenedor. Si es de `` ancho '', cada violín tendrá el mismo ancho.
**scale_hue:** cuando se usa la clasificación de tono y se establece en Verdadero, este parámetro determina si se debe escalar la variable de agrupación principal;

**gridsize**: establece la suavidad del gráfico de violín, cuanto mayor sea la suavidad;

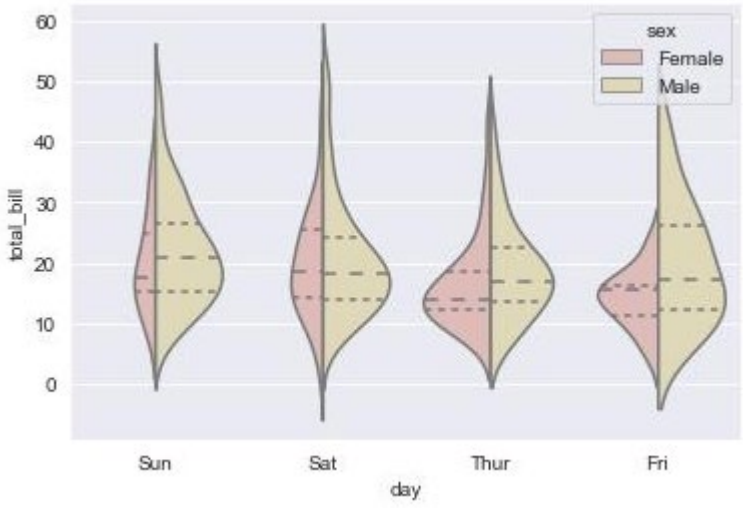
**inner**: "box", "quartile", "point", "stick", None, la representación de puntos de datos dentro del violín. Indique respectivamente: caja, cuartil, punto, línea de datos y sin indicación;

**split:** si dividir, cuando se establece en True, dibujar la mitad del violín para cada nivel clasificado por tono;

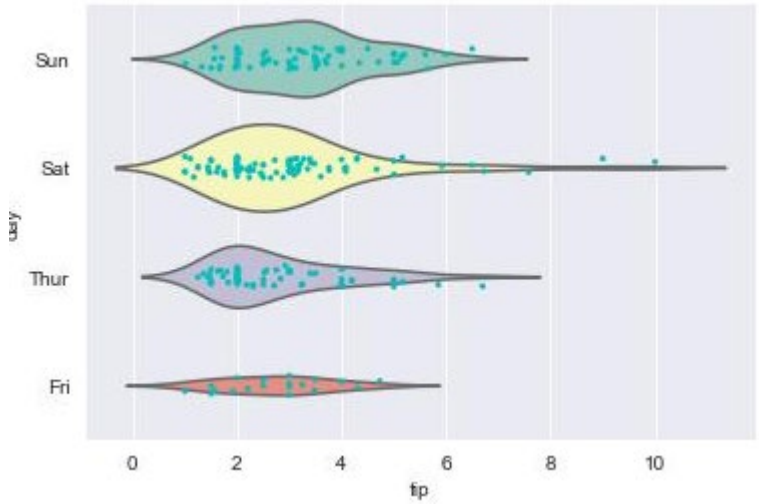
```
ax = sns.viollinplot(x = tips["total_bill])
```




```
ax = sns.violinplot(x='day', y='total_bill',
hue='sex', data = tips, palette=sns.hls_palett(8, l =
0.8, s = 0.5), split=True, scale = 'count', inner =
'quartile')
```



```
ax = sns.violinplot(x='tip', y='day', hue='sex', data
= tips, palette='Set3', scale = 'count', inner =
'None, whis = np.inf)
ax = sns.strpplot(x='tip', y='day', data = tips,
jitter = True, color = 'c', size = 3)
```

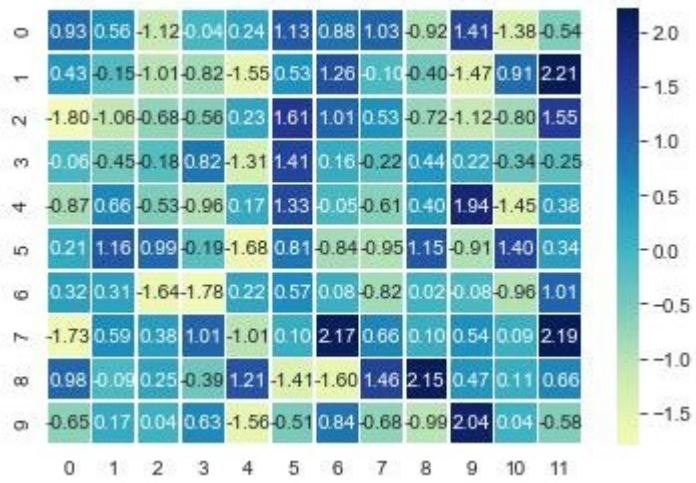


Gráficos de matrices

*Heatmap*

```
sns.heatmap(data, *, vmin=None, vmax=None, cmap=None,
center=None, robust=False, annot=None, fmt='.2g',
annot_kws=None, linewidths=0, linecolor='white', cbar=True,
cbar_kws=None, cbar_ax=None, square=False, xticklabels='auto',
yticklabels='auto', mask=None, ax=None, **kwargs)
```

```
normal_data = np.random.randn(10, 12)
ax = sns.heatmap(normal_data, center = 0, annot = True,
fmt = "0.2f", cmap = "YlGnBu", linewidths = 0.5)
#annot_kws = ("size": 9, "weight": "bold", "color": "w")
```



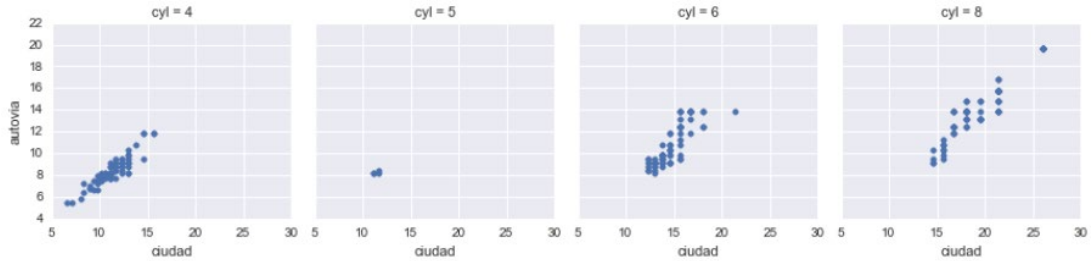

Grillas de gráficos

*Facetgrid*

Si queremos dibujar una gráfica de muestras de consumo en ciudad contra consumo en autovía clasificando por número de cilindros usando para ello la función **FacetGrid()**. FacetGrid() nos proporciona una

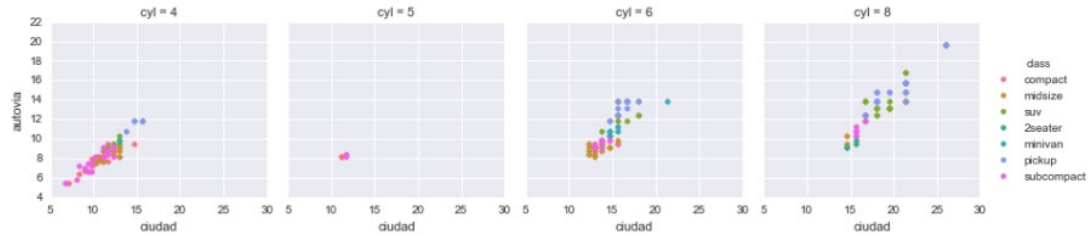
cuadrícula de gráficas con datos clasificados según una o dos variables categóricas. Sobre esa cuadrícula, aplicamos una función de Matplotlib mediante map() para dibujar la gráfica que buscamos:

```
grid = ans.FacetGrid(fedata, col = "cyl")
grid.map(plt.scatter, "ciudad", "autovía")
plt.show()
```




Podemos ver que, en general, a mayor número de cilindros, mayor es el consumo; pero esto no es una verdad absoluta. Hay coches con 8 cilindros que consumen menos que coches con 4. Con una minúscula modificación, podemos ver aún más detalles. Muchas funciones de Seaborn admiten un parámetro hue que separa los datos por clase. En nuestro caso, es la segunda clasificación después de haber usado FacetGrid

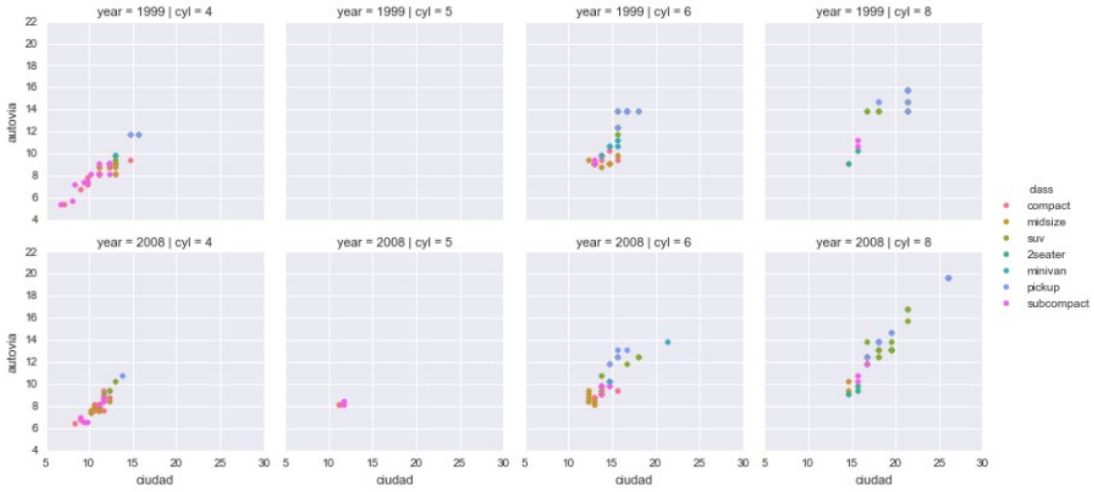
```
grid = ans.FacetGrid(fedata, col = "cyl")
grid.map(plt.scatter, "ciudad", "autovía")
plt.show()
```



Vemos que hay clases que en general muestran un mejor comportamiento (*compacts*, *subcompacts*) y otras que tienen un consumo generalmente mayor (*pickup*).

FacetGrid() tiene aún una dimensión más que nos permite una tercera clasificación usando el parámetro row:

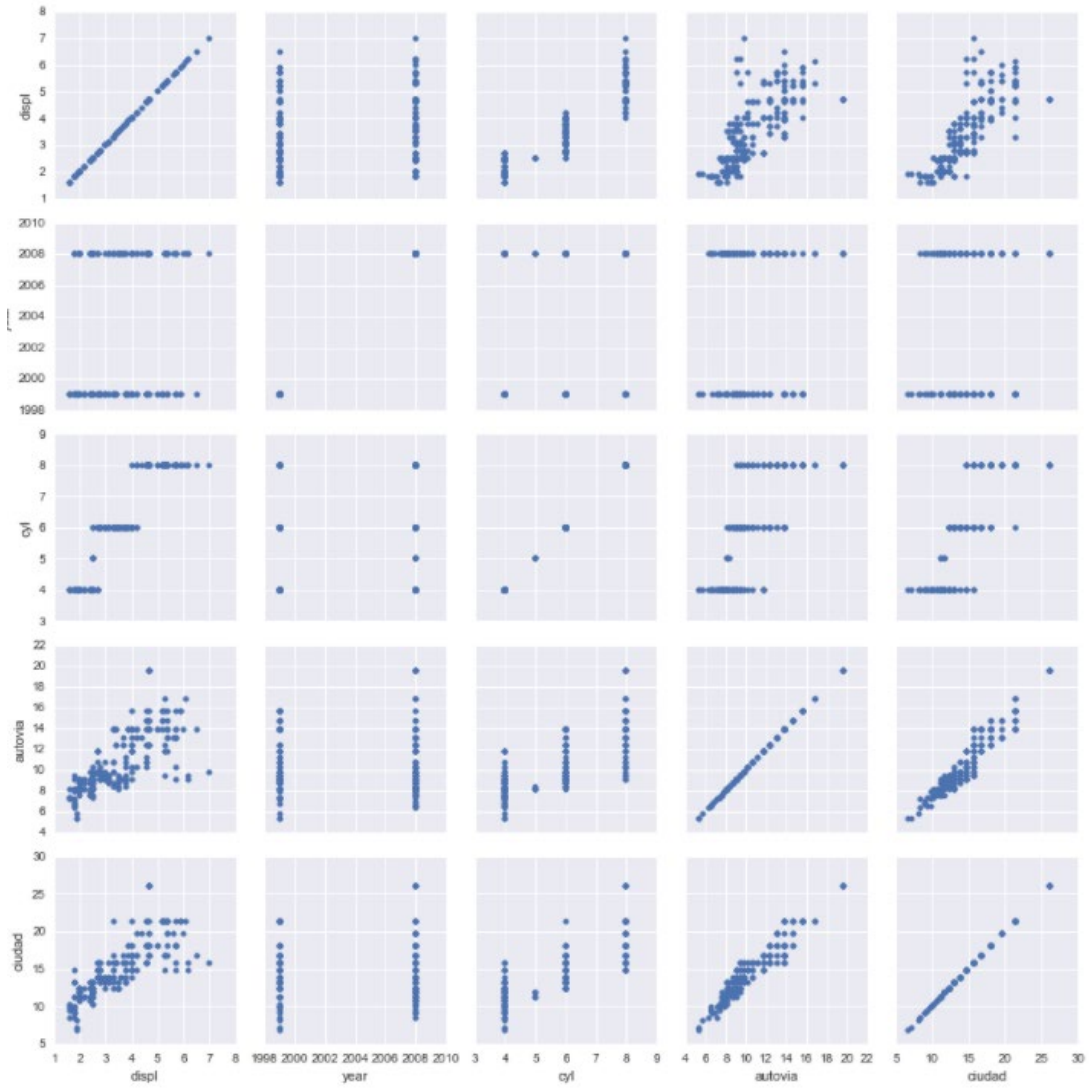
```
grid = sns.FacetGrid(fedata, col = "cyl", row = "year",
hue = "class")

grid.map(plt.scatter, "ciudad", "autovía").add_legend()

plt.show()
```




*Pairgrid*

Función que nos permite ver rápidamente las relaciones entre las variables de un DataFrame, esta muestra una gráfica para cada par de columnas.

```
grid = sns.PairGrid(fedata)
grid.map(plt.scatter)
plt.show()
```



**Referencias**

[1] Seaborn para Python: ¿Cómo visualizar datos con Pairplot?
https://www.youtube.com/watch?v=2tObZDNGTJI

[2] Cómo visualizar datos
https://www.youtube.com/watch?v=2tObZDNGTJI

[3] Librería seaborn
https://interactivechaos.com/es/manual/tutorial-de-seaborn

[4] Representación Gráfica
https://www.emilkhatib.es/representaciones-graficas-con-seaborn/