Análisis y Visualización de los datos

Sergio Pérez Peló

Jesús Sánchez-Oro

Universidad
Rey Juan Carlos

Introducción

- Cuando tenemos un algoritmo funcional para resolver nuestro problema, debemos realizar **dos tareas**:
  - **Analizar** los resultados obtenidos
  - **Presentarlos** de forma inteligible

Bokeh

Análisis de datos: Pandas

Proporciona todas las operaciones para preparación y limpieza de datos

Estructuras de datos principales: **Series** y **DataFrames**

La documentación de la biblioteca ofrece un paseo introductorio de 10 minutos que muestra algunas de sus funciones básicas

Pandas: instalación

- La instalación de Pandas es simple.
  - Desde la terminal del entorno de desarrollo ejecutamos la instrucción:

```
pip install pandas
```

- Comenzar a trabajar con ella también lo es:
  - En nuestro código, colocamos la instrucción:

```
import pandas as pd
```

Series

- **Vector unidimensional de datos** (similar a un array)
- Añaden **etiquetas que identifican a cada elemento** del vector, en lugar de usar exclusivamente un índice numérico para marcar el orden
  - Reordenar los valores de la serie de forma eficiente (usando las etiquetas)
  - Encontrar valores dentro de la serie (para una etiqueta unívoca

Series

- Se pueden crear a partir de diversos tipos de datos de entrada, recogidos por el argumento data

```
mi_serie = pd.Series(data, index=my_index)
```

- *data* puede ser un diccionario, una lista, un escalar…
- Veamos algunos ejemplos

Series - Indexación

- Similar a la que podemos aplicar en listas de Python:
  - índices numéricos (individuales o *slicing* para intervalos)
  - filtros con expresiones booleanas, etc.
- Adicionalmente, también podemos usar las etiquetas para indexar.
  - Esta opción tiene la ventaja añadida de ser rápida (baja complejidad computacional) y el código queda muy legible, al usar identificadores con sentido para acceder a los valores.

Series - Indexación

- Los atributos loc e iloc permiten realizar indexación y *slicing* indicándoles las etiquetas de las filas o bien mediante el uso de offset numérico.

```
#Obtenemos el elemento etiquetado por 'a'

serie_1.loc['a']

# Obtenemos el elemento que se encuentra en la posición 0

serie_1.iloc[0]
```

Series - Indexación

- También se pueden utilizar **máscaras** para seleccionar un determinado conjunto de valores.

```
## Ejemplo de máscara

# Indexación empleando una expresión booleana

serie_1[serie_1 < 0]
```

Series - Indexación

- El *slicing* de la indexación se aplica a ambos arrays (valores e índices) dentro del objeto Series.
- De esta forma, indexamos sin perder ninguna de las propiedades del objeto

ÍNDICES	DATOS
1	'A'
2	'B'
3	'C'
4	'D'

Series – Añadir y consultar datos

- Una Serie también puede ser modificada añadiendo nuevos elementos:

```
serie_1[5] = 'E'
```

- También podemos preguntar si un determinado índice se encuentra dentro de la Serie.

```
'E' in serie_1
```

Series – Operaciones

- Es posible efectuar operaciones aritméticas o aplicar funciones a los valores almacenados en objetos de tipo Series

```
# Suma elemento a elemento, fijándonos en las etiquetas
# para efectuar el emparejamiento
serie_1 + serie_1
```

Series – Operaciones

- Las operaciones entre series realizan una alineación automáticamente de los datos en función de su etiqueta.
- Podemos realizar cálculos sin tener en cuenta si las Series involucradas tienen las mismas etiquetas
- Veamos un ejemplo sencillo en código

Dataframes

- Tabla de valores organizados por filas y columnas que están etiquetadas.
  - También podemos ver los DataFrame como un **array 2-D con etiquetas**, que pueden ser de cualquier tipo, tomando valores enteros por defecto.
- Los nombres de las columnas corresponden a cada una de las variables disponibles, mientras que cada fila corresponde a un caso.
- Si no tenemos valores para alguna de estas celdas tenemos un **dato faltante** y el hueco queda marcado explícitamente. Por tanto, se trata de un tipo de datos estructurado.

Dataframes

- Podemos crear objetos DataFrame a partir de diversos tipos de datos de entrada
  - Listas, diccionarios…
  - También a partir de otros objetos Series o DataFrame de Pandas.
- Tanto las filas como las columnas suelen estar etiquetadas, especialmente las columnas, ya que identifican las variables que estamos midiendo en el análisis.
- Veamos algunos ejemplos

Dataframes - Indexación

- Es posible indexar tanto subconjuntos de filas como de columnas.
  - Podemos usar la sintáxis típica de *slicing* en Python (con índices numéricos) o bien utilizar los nombres de filas o columnas (si se han asignado).
- Si se selecciona una fila o una columna (ya sea por su etiqueta o por su índice), el resultado será un objeto de la clase Series.
- Si se selecciona un subconjunto del DataFrame el resultado será otro DataFrame.

Dataframes - Indexación

- Puede que de una columna solo nos interesen algunas filas.
- Para realizar esta operación, además de indicar el nombre de la columna tendremos que indicar las filas (estilo matriz)

```
#Selección de las 4 primeras filas de la columna group del
                    DataFrame df_1

                    df_1['group'][:4]
```

- Si interpretamos un DataFrame como un array 2-D, nos puede resultar interesante obtener la matriz de datos subyacente mediante el uso del atributo values.

Dataframes - Indexación

- Los atributos loc e iloc también están disponibles para los DataFrames

```
# Ejemplo de uso del atributo loc

df_states.loc[:'Illinois', :'population']

# Ejemplo de uso del atributo iloc

df_states.iloc[:3, :2]
```

Dataframes - Indexación

- También es posible el uso de **máscaras** para seleccionar los datos que cumplan con una determinada condición
  - Se indica mediante una expresión lógica.

```
# Mostramos todas las filas que cumplen la condición
  expresada en la máscara

df_4 = df_3[df_3.group=='B']
```

- Esta será la forma en la que podremos generar DataFrames a partir de otros, en función de que cumplan o no una condición (separación de datos)

Dataframes - Reindexación

- Consiste en crear un nuevo objeto con sus índices siguiendo una nueva ordenación.
- También permite añadir o eliminar columnas

```
df_3 = df_2.reindex(columns=['best_values', 'fo_1_value',
                             'fo_2_value', 'average'])
```

Dataframes – Añadir y consultar datos

- Los DataFrame se pueden modificar añadiendo o eliminando filas o columnas.

```
# Ejemplo de eliminación de una columna
df_3 = df_3.drop('genotype', axis=1)
# Ejemplo de eliminación de una fila
df_3 = df_3.drop(2, axis=0)
# Ejemplo de añadir una columna
df_3['genotype'] = df_3['intake']/df_3['collate']
# Ejemplo de añadir una fila
df_3.loc[11] = {'intake':56.3, 'group':'C',
                'genotype':5.1181}
```

Dataframes – Concatenación de operaciones

- Operaciones como:
  - group by (con el paradigma *split-apply-combine*)
  - *merge*, *join* y concatenación
  - transformaciones entre formato de datos *long* y *wide*
- También están soportadas en Pandas por una amplia colección de herramientas y métodos.
- Veamos algunos ejemplos

Visualización de los datos

Sergio Pérez Peló

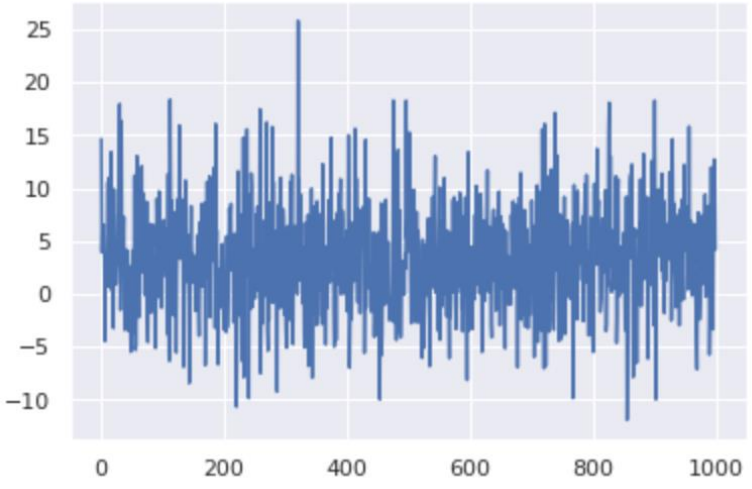
Jesús Sánchez-Oro

Universidad
Rey Juan Carlos

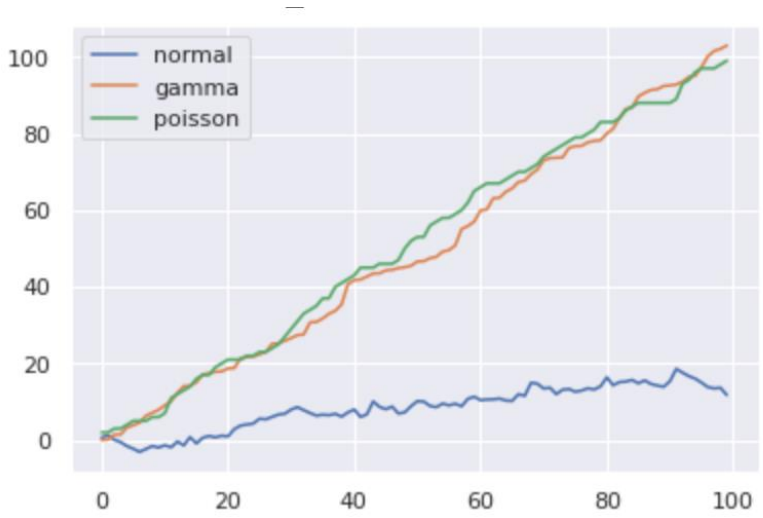
Introducción

- La **biblioteca Pandas** también nos ofrece una serie de **métodos para visualización de datos**, que además ya vienen orientados a facilitar nuestro trabajo con objetos de tipo Series y DataFrame.
- Son muy simples en su llamada, puesto que asumen por defecto algunas decisiones razonables acerca del tamaño y la configuración del gráfico que vamos a construir.

Introducción




```
norm_3_5 = pd.Series(np.random.normal(loc=3,
scale=5, size=1000))
norm_3_5.plot()
```

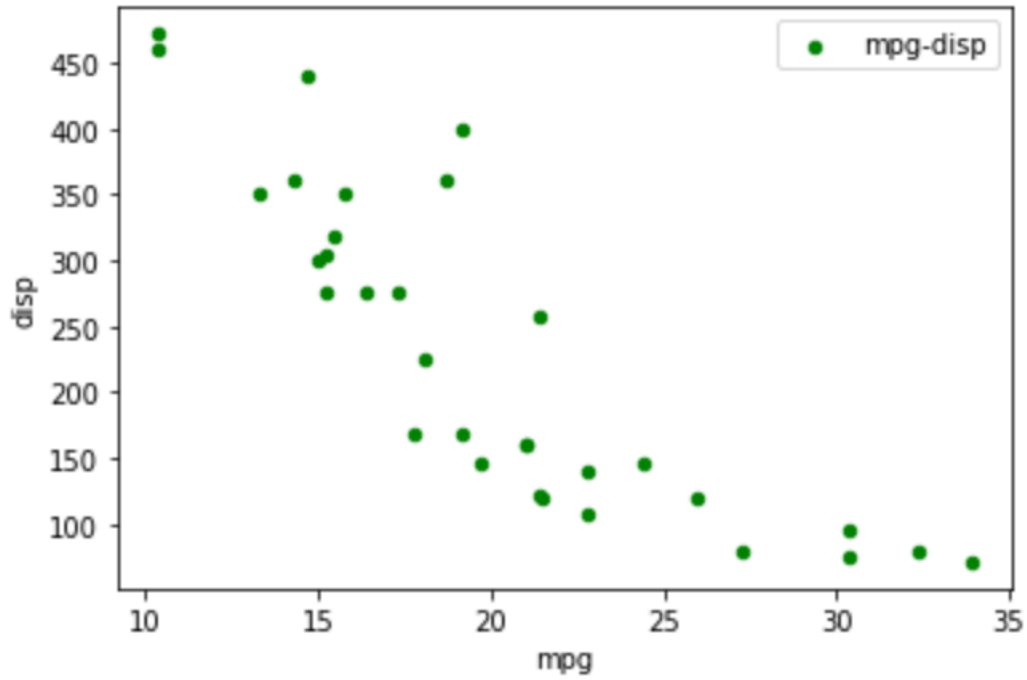



```
variables = pd.DataFrame({'normal':
np.random.normal(size=100),
'gamma': np.random.gamma(1, size=100),
'poisson': np.random.poisson(size=100)})
variables.cumsum(0).plot()
```

Scatterplots

- Permiten analizar posibles correlaciones entre variables cuantitativas. Es necesario especificar el nombre de las columnas cuyos valores se quieren representar.
- También son útiles para representar la calidad de nuestras soluciones o visualizar frentes de soluciones no dominadas (multi-objetivo)
- Básicamente: gráficos de puntos con eje X e Y, siendo estos ejes dos variables bajo comparación

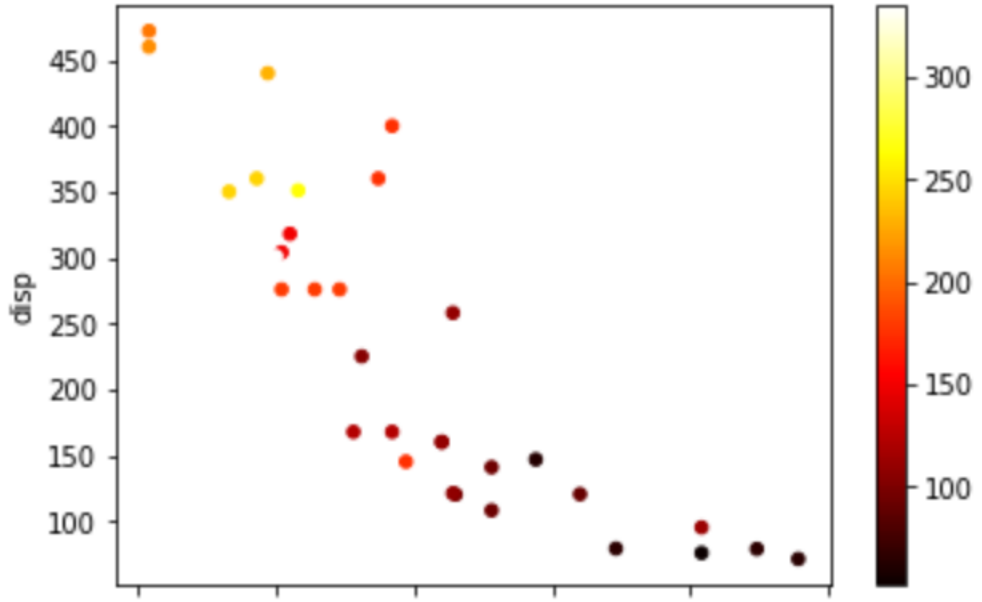
Scatterplots




```
mtcars.plot.scatter('mpg', 'disp', color='Green', label='mpg-disp')
```

Scatterplots

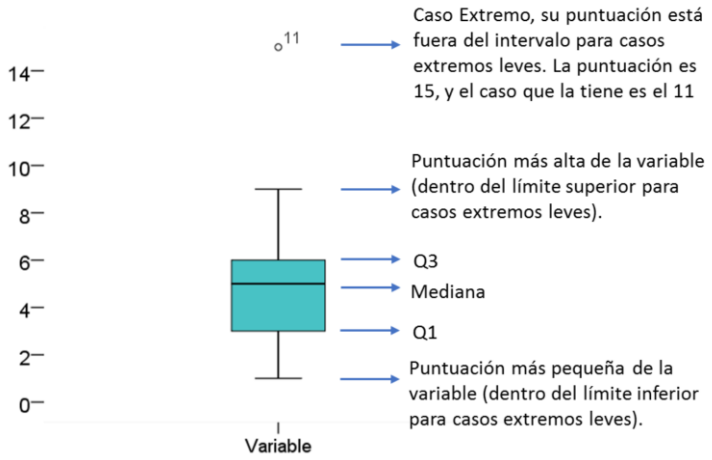
- El color que se le asigna a los valores visualizados puede ser degradado en función de los valores de una tercera variable.



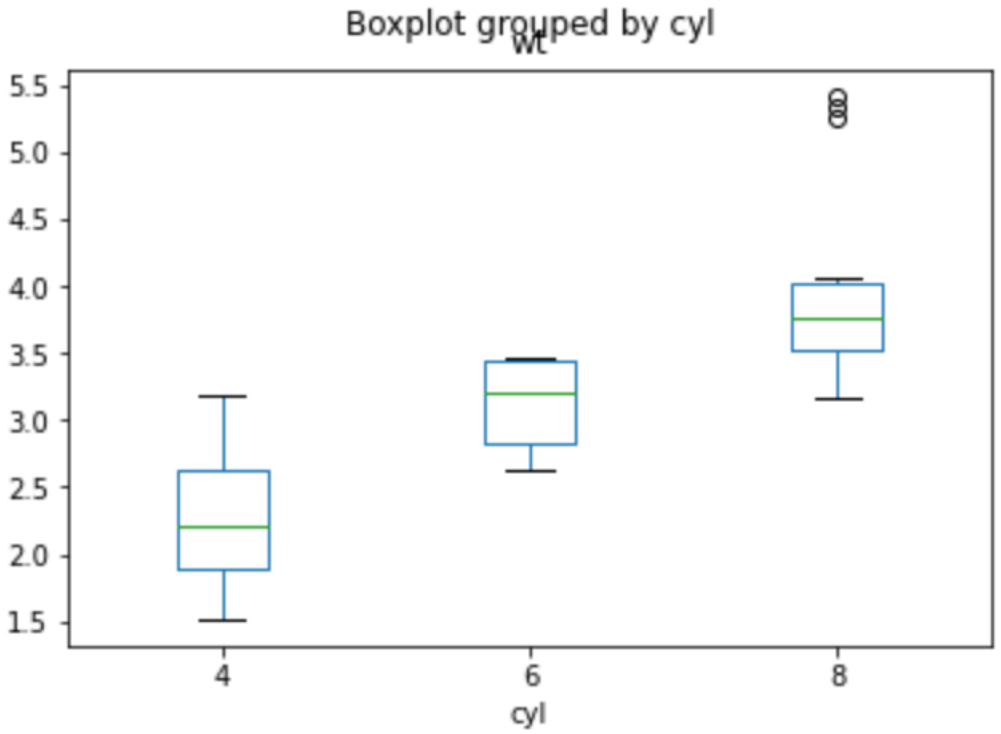

```
mtcars.plot.scatter('mpg', 'disp', c=mtcars.hp, cmap='hot')
```

Boxplot

- También podemos hacer representaciones de diagramas de cajas agrupados según la variable que le indiquemos.
- Un diagrama de cajas (también conocido como de cajas y bigotes)
- Estos diagramas representan varios estadísticos para una variable en un solo gráfico
  - Puede ser útil para hacernos una idea de la distribución de los valores de la función objetivo para nuestro conjunto de soluciones



Boxplot

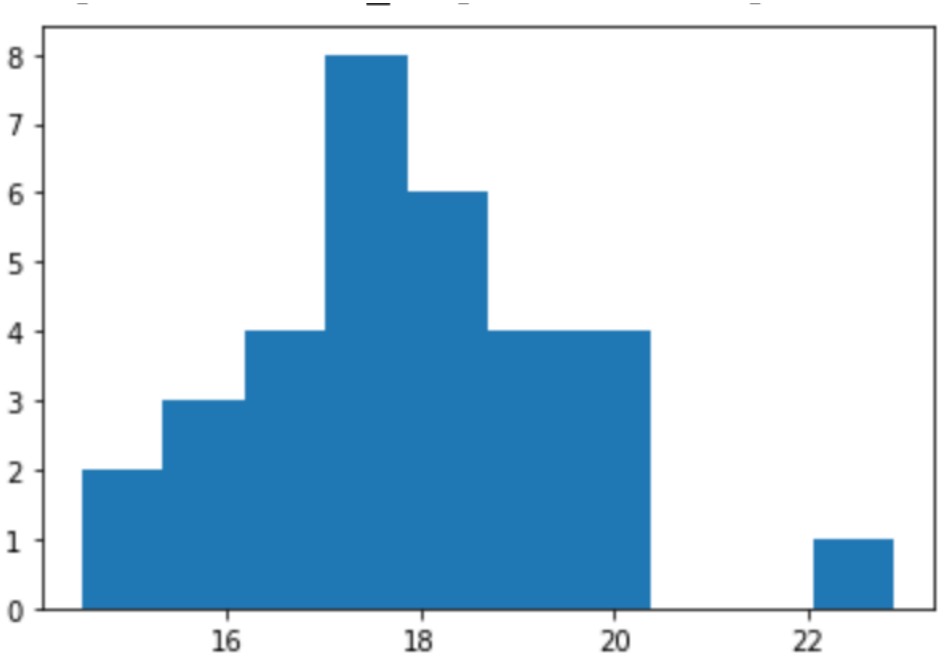



```
mtcars.boxplot(column='wt', by='cyl', grid=False)
```

Histogramas

- Otro tipo de diagramas interesantes son los histogramas (gráficos de barras) y los de densidad de probabilidad
- Un **histograma** es una representación gráfica de una variable en forma de barras, donde la superficie de cada barra es proporcional a la frecuencia de los valores representados.
  - Puede ser útil para representar el número de veces que un algoritmo alcanza la mejor solución
  - O incluso para representar el valor de la función objetivo que alcanza un determinado algoritmo

Histogramas




```
mtcars.qsec.hist(grid=False)
```

Extensiones de la visualización - Seaborn

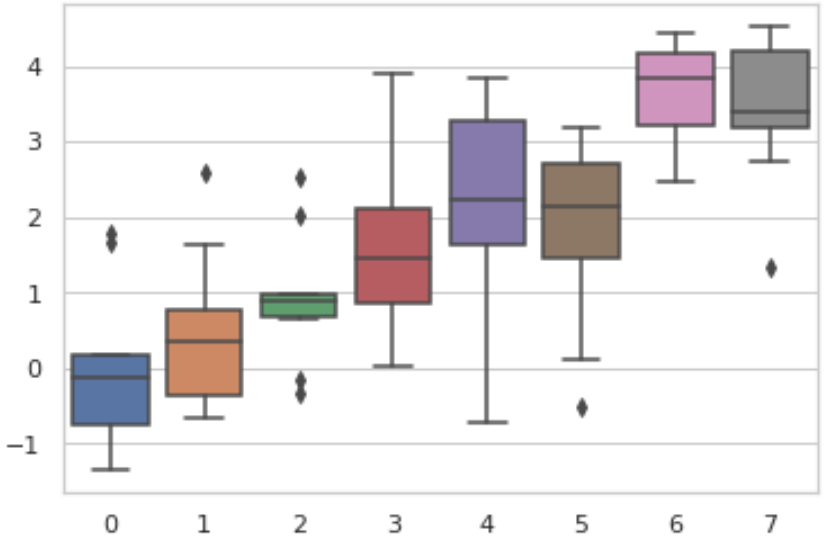
- Existen librerías que recubren la visualización de Pandas, mejorándola bastante de cara a presentar nuestros resultados (paper, superiores, clientes…)
- Una muy extendida es **Seaborn**
- Su uso nos va a permitir:
  - Mejorar la legibilidad de los gráficos
  - Mejora del uso de la paleta de colores
  - Mejoras en el aspecto de las gráficas

Seaborn

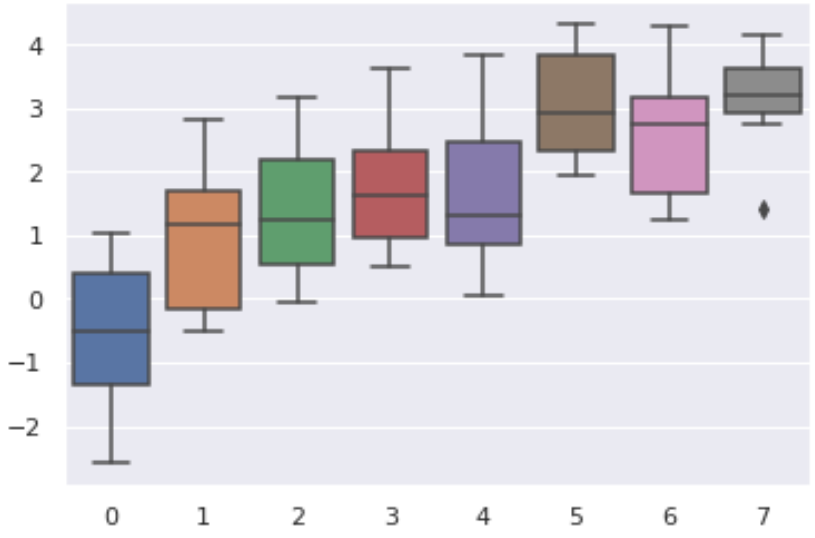
- Cuenta con dos grupos de funciones:
  - El que se encarga de los aspectos estéticos
  - El que se encarga del escalado
- Mediante ambas podemos controlar aspectos estéticos (color de fondo, incluir cuadrículas, ticks en los ejes…)

Seaborn – Configuración del aspecto

- Disponemos de las funciones `axe_style()` y `set_style()`



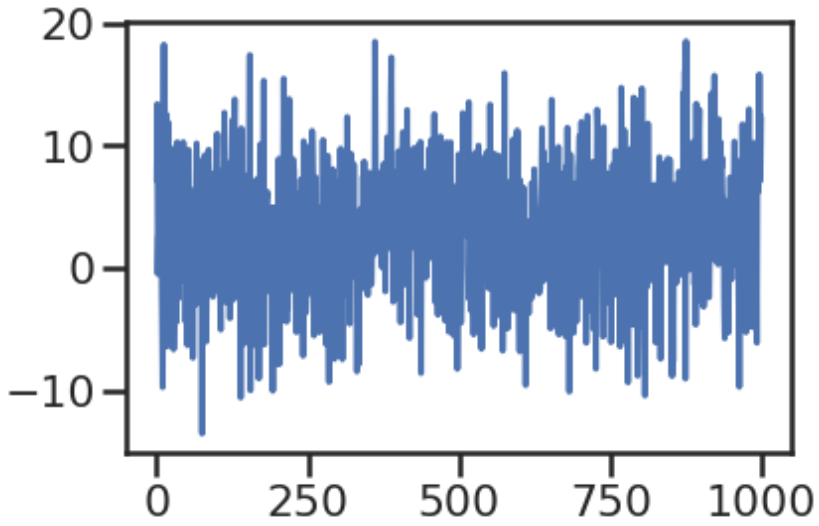

```
import seaborn as sns
sns.set_style("whitegrid")
data = np.random.normal(size=(10, 8)) + np.arange(8) / 2
sns.boxplot(data=data)
```



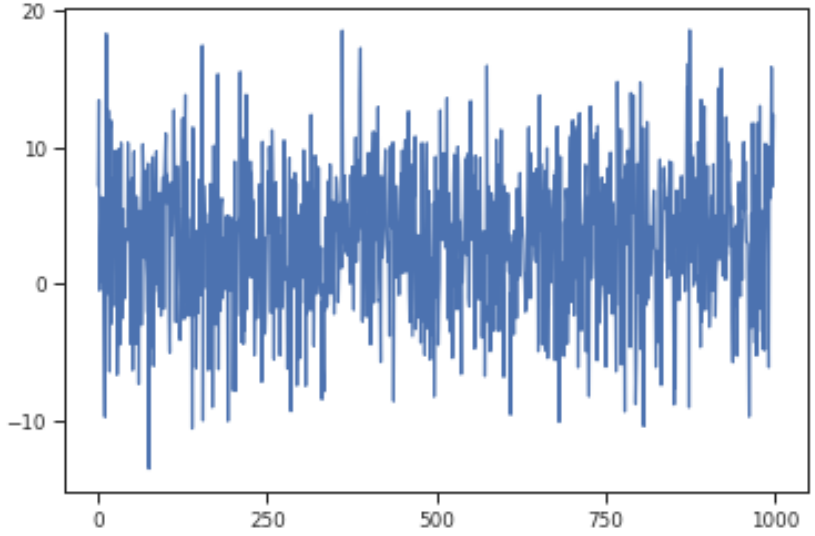

```
with sns.axes_style("darkgrid"):
    data = np.random.normal(size=(10, 8)) + np.arange(8) / 2
    sns.boxplot(data=data)
```

Seaborn – Escalado del gráfico

- Disponemos de las funciones `plotting_context()` y `set_context()`



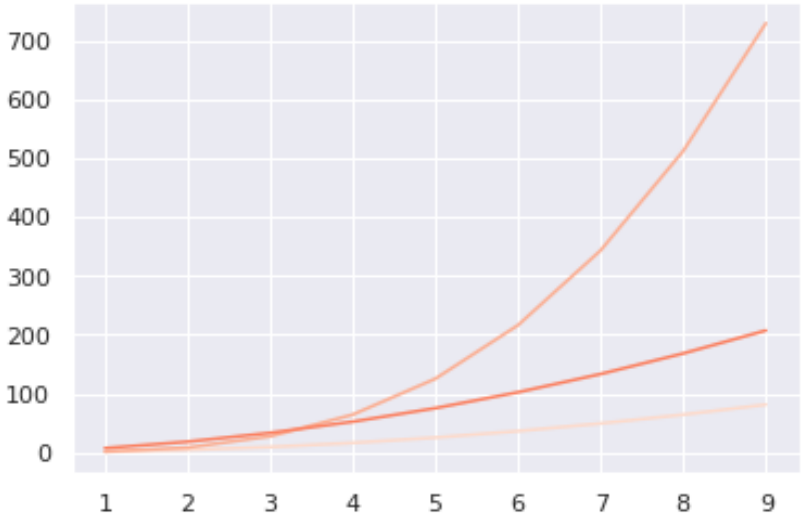

```
sns.set_context(”poster")
norm_3_5.plot()
```



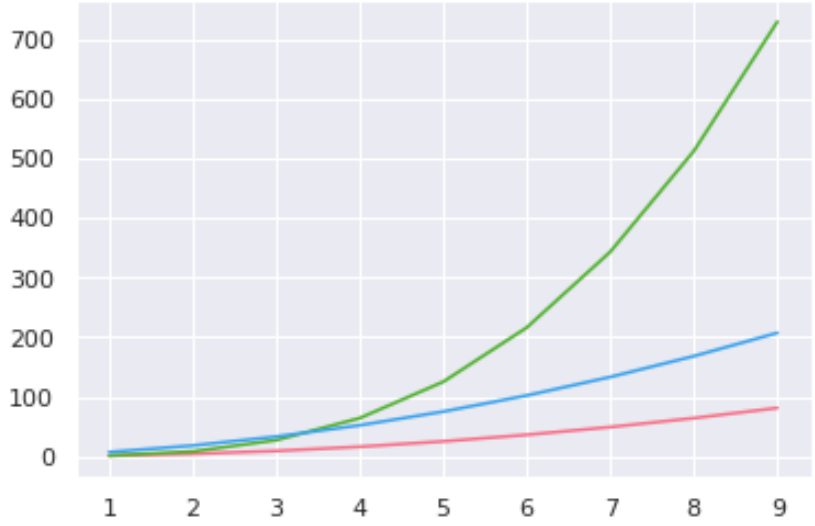

```
with sns.plotting_context("paper"):
    f, ax = plt.subplots()
    ax.plot(norm_3_5)
```

Seaborn – Paleta de colores

- Disponemos de las funciones `color_palette()` y `set_palette()`



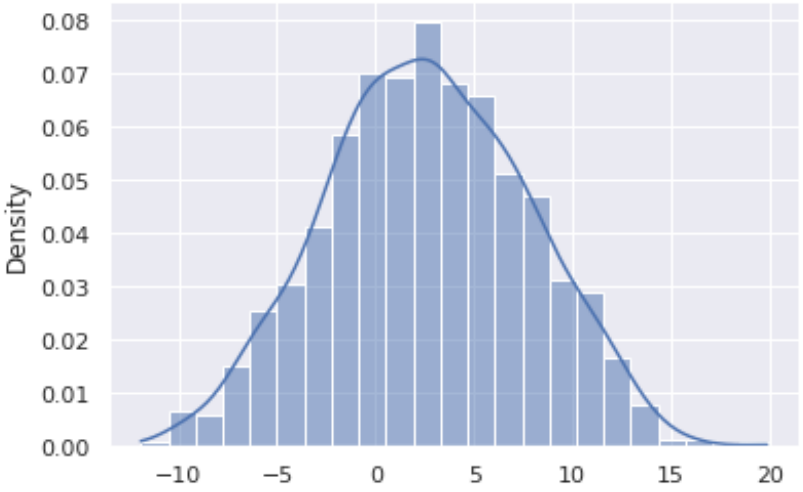

```
sns.set()
x = np.arange(1, 10)  # definimos el rango del eje x
y1 = x**2
y2 = x**3
y3 = 2*x**2 + 5*x
sns.set_palette("Reds")
plt.plot(x,y1,x,y2,x,y3)
```



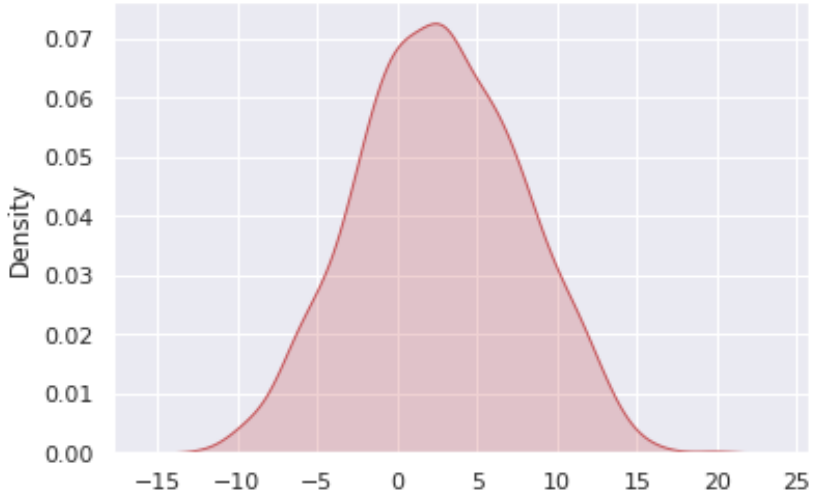

```
with sns.color_palette("husl", 3):
    _ = plt.plot(x,y1,x,y2,x,y3)
```

Seaborn – Histogramas y KDE

- Disponemos de las funciones `distplot()` y `kdeplot()` con diferentes parámetros




```
sns.set()
#sns.distplot(norm_3_5)
```



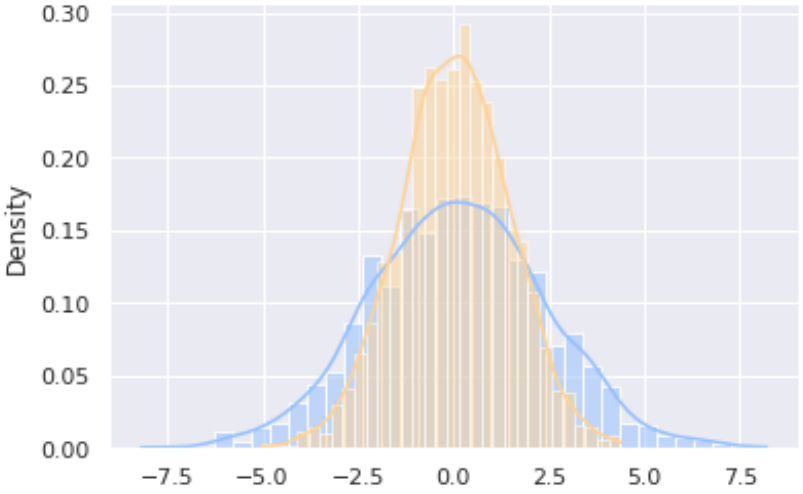

```
sns.kdeplot(norm_3_5, shade=True, color="r")
```

Seaborn – Scatterplot y KDE multivariante

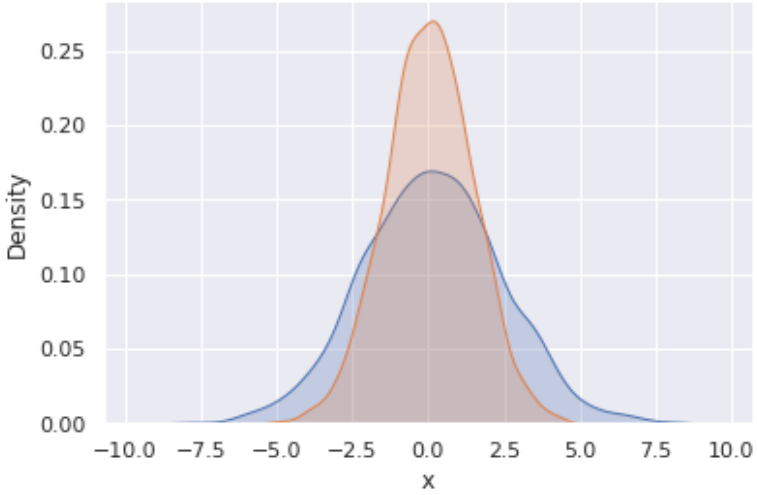
- En el caso de que tengamos varias variables respuesta (análisis multivariante) también podemos usar las funciones `kdeplot()` y `displot()`
- Resulta más útil crear una **figura de varios paneles**
- Para llevar a cabo este tipo de representación tenemos la función `jointplot()`

Seaborn – Scatterplot y KDE multivariante

- Ejemplos con `distplot()` y `kdeplot()` con diferentes parámetros



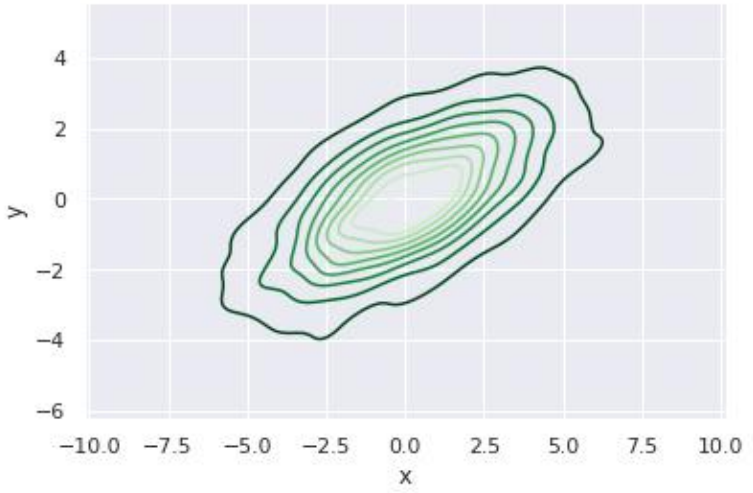

```
datos = pd.DataFrame(vector, columns=['x','y'])
color={'x':'#94c0ff', 'y':'#ffd394'}
for col in 'xy':
    sns.distplot(datos[col]).set(xlabel=None)
```



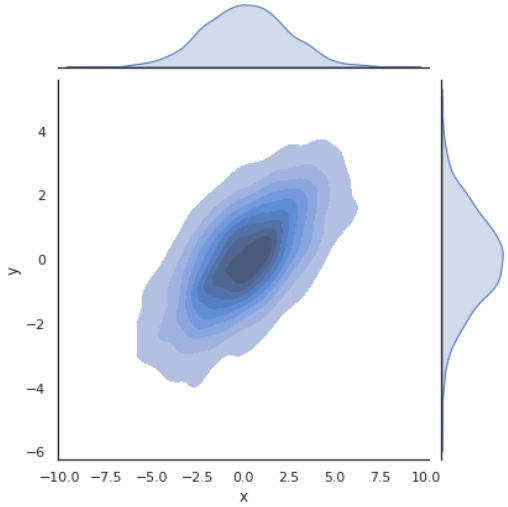

```
for col in 'xy':
    sns.kdeplot(datos[col], shade=True)
```

Seaborn – Scatterplot y KDE multivariante

- Ejemplos con `kdeplot()` y `jointplot()` con diferentes parámetros



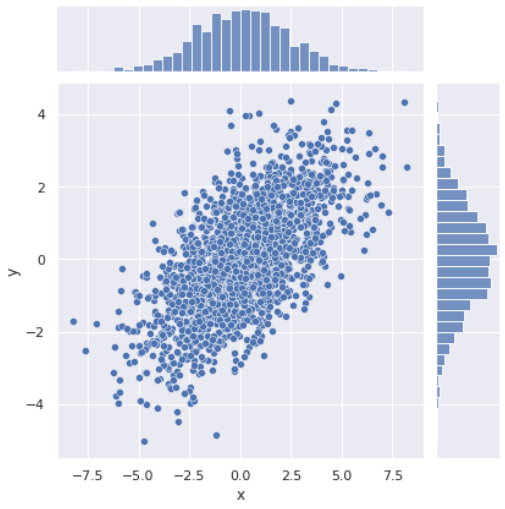

```
sns.kdeplot(data=datos, x=datos['x'], y=datos['y'],
cmap="Greens_r")
```



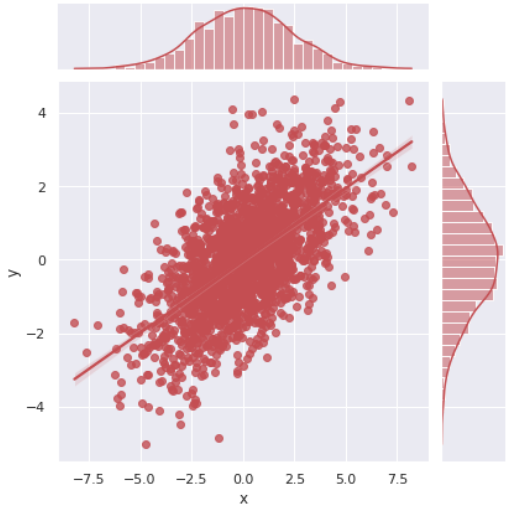

```
with sns.axes_style("white"):
    sns.jointplot(x="x",y="y", data=datos, kind="kde",
fill=True)
```

Seaborn – Scatterplot y jointplot

- Ejemplos con `jointplot() + scatterplot` con diferentes parámetros



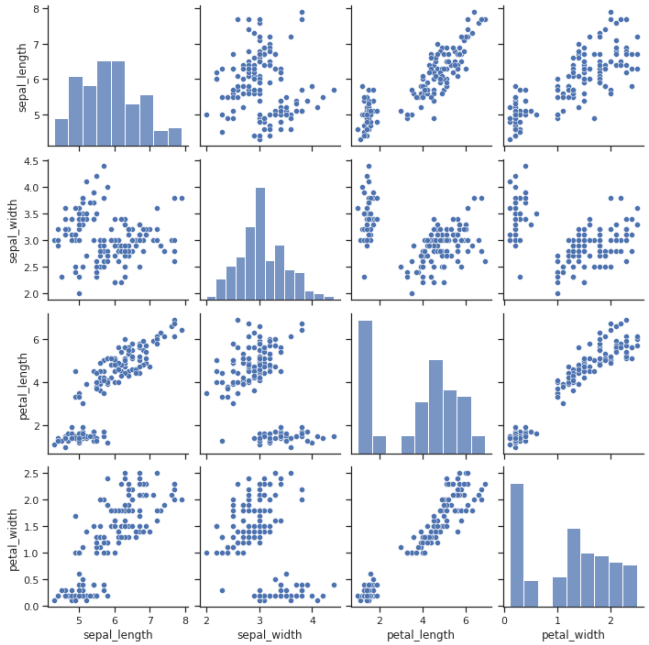

```
sns.jointplot(x="x",y="y", data=datos)
```




```
sns.jointplot(data=datos, x='x', y='y', kind="reg", color="r")
```

Seaborn – Pairplot

- Generalizando `jointplot()` para varias dimensiones de datos, utilizamos `pairplot()`




```
sns.set_style("ticks")
iris = sns.load_dataset("iris")
sns.pairplot(iris)
```

Seaborn – Variables categóricas

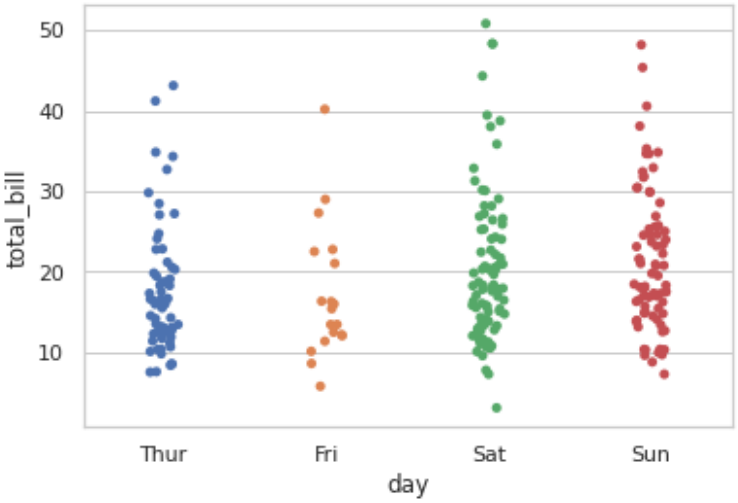
- Para la visualización de variables categóricas Seaborn cuenta con diversas funciones que podemos clasificar en tres grupos:
  1. Las que muestran cada **observación en cada nivel** de la variable categórica: `swarmplot()` y `stripplot()`
  2. Las que muestran una representación abstracta de cada **distribución** de observaciones: `boxplot()` y `violinplot()`
  3. Las que aplican una estimación estadística para mostrar una **medida de tendencia central e intervalo de confianza**: `barplot()` y `pointplot()`

Seaborn – Variables categóricas

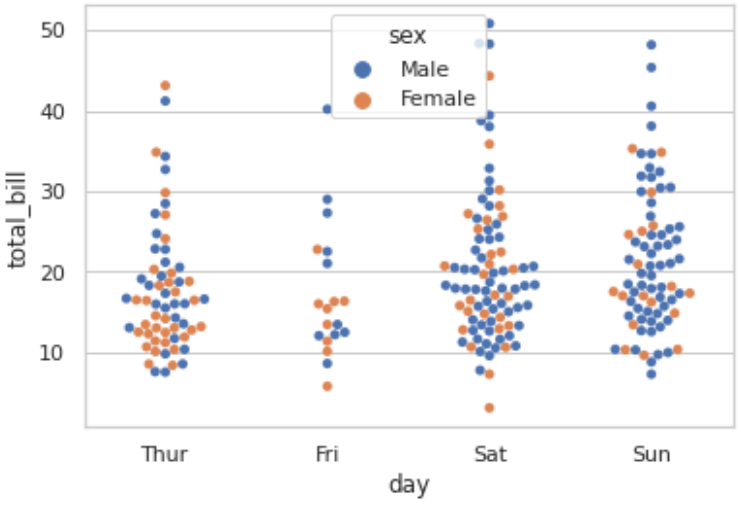
- Todas estas funciones **comparten una API básica** de cómo aceptan los datos, aunque cada una tiene parámetros específicos que controlan los detalles propios de cada caso.

Seaborn – Variables categóricas

- Ejemplos de las funciones stripplot() y swarmplot()



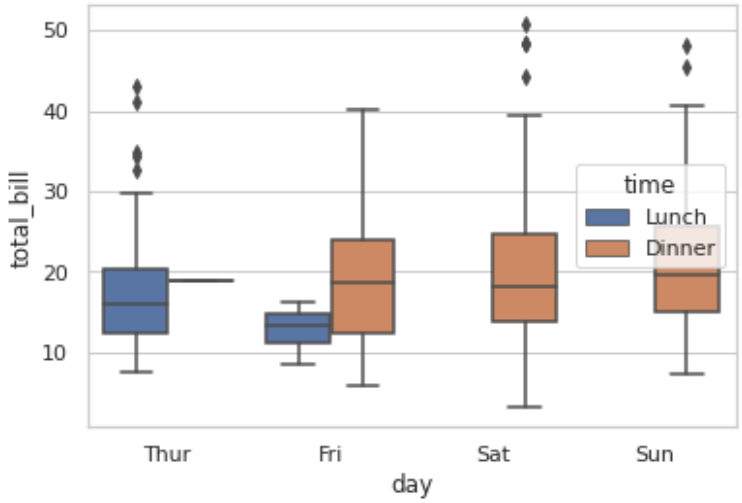

```
tips = sns.load_dataset("tips")
sns.set_style("whitegrid")
sns.stripplot(x="day", y="total_bill", data=tips)
```



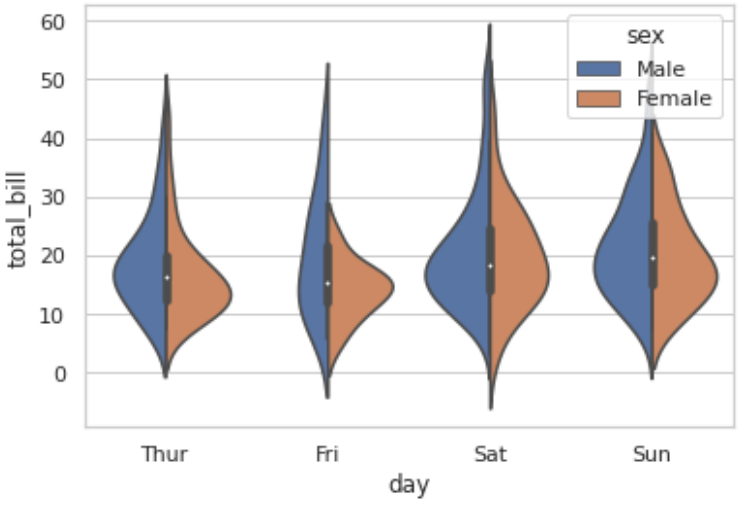

```
sns.swarmplot(x="day", y="total_bill", hue="sex", data=tips)
```

Seaborn – Variables categóricas

- Ejemplos de las funciones `boxplot()` y `violinplot()`



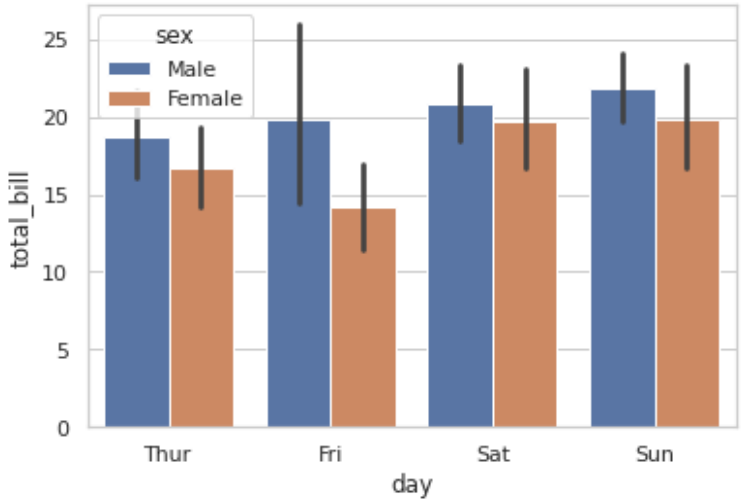

```
sns.boxplot(x="day", y="total_bill", hue="time", data=tips,
linewidth=1.5)
```



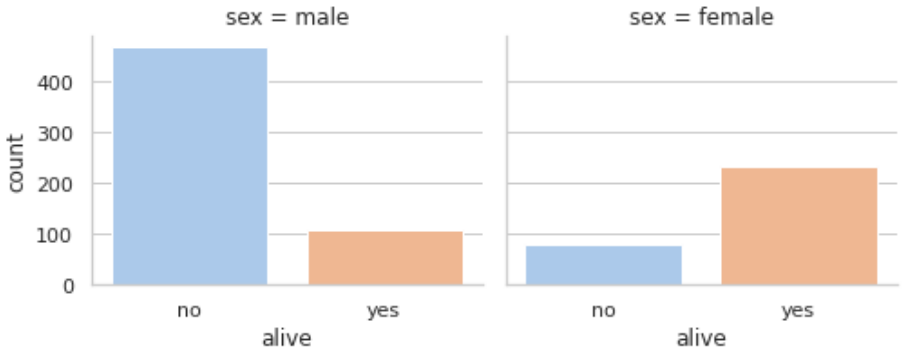

```
sns.violinplot(x="day", y="total_bill", hue="sex", data=tips,
split=True)
```

Seaborn – Variables categóricas

- Ejemplos de las funciones `barplot()` y `catplot()`




```
sns.barplot(x="day", y="total_bill", hue="sex", data=tips)
```

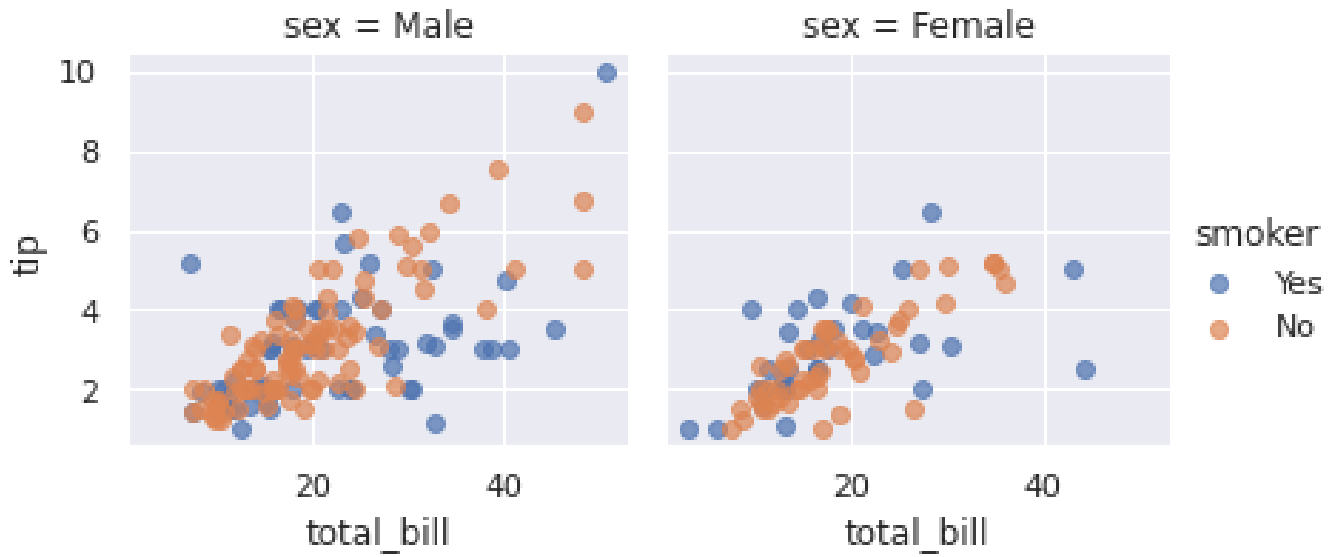



```
sns.catplot(x="alive", col="sex", col_wrap=2,
data=titanic[titanic.sex.notnull()],
kind="count", height=3.0, aspect=1.2, palette="pastel")
```

Seaborn – Representación de una matriz de gráficas

- La clase FacetGrid nos permite realizar un **grid de gráficas** en el que se pueden especificar hasta tres dimensiones: *row*, *col* y *hue*.
  - Los dos primeros, como su nombre indica, hacen referencia a las filas y las columnas.
  - El tercero, *hue*, se puede entender como una tercera dimensión donde los diferentes niveles se representan con diferentes colores.

Seaborn – Ejemplo FacetGrid

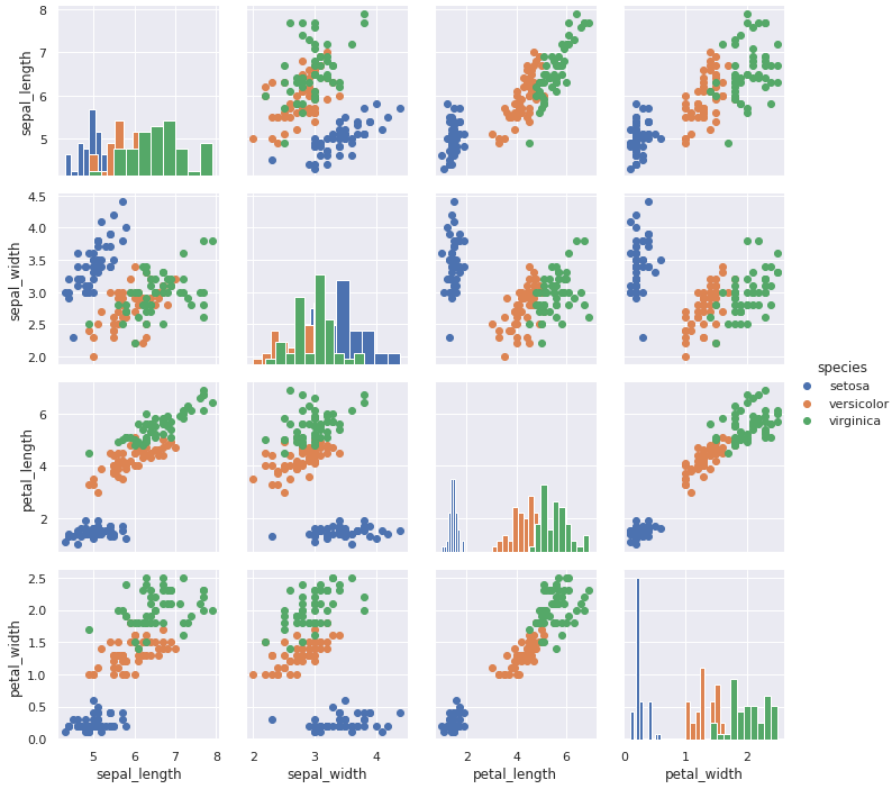



```
g = sns.FacetGrid(tips, col="sex", hue="smoker")
g.map(plt.scatter, "total_bill", "tip", alpha=.7)
g.add_legend()
```

Seaborn – Representación de una matriz de gráficas

- La clase Pairgrid nos permite representar un **grid** de figuras utilizando en todas ellas el **mismo o diferentes tipos de representaciones**.
- A diferencia de FacetGrid, con PairGrid a cada fila y cada columna se le puede asignar una variable diferente.
  - Nótese que con FacetGrid en cada gráfica se representa la misma relación de variables sujetas a los diferentes niveles de una tercera variable, mientras que con PairGrid en cada gráfica se muestra una relación diferente.

Seaborn – Ejemplo PairGrid




```
iris = sns.load_dataset("iris")
g = sns.PairGrid(iris, hue="species")
g.map_diag(plt.hist)
g.map_offdiag(plt.scatter)
g.add_legend()
```

Bokeh

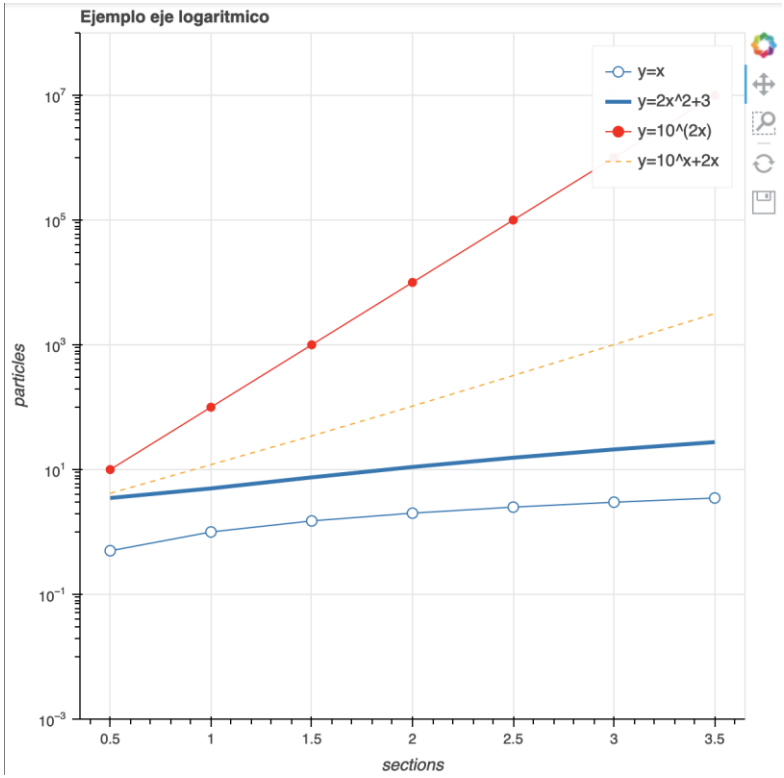
- Proporciona herramientas muy potentes y versátiles para representación interactiva de datos, añadiendo comportamiento dinámico, muy apropiado para interfaces web.
- Los pasos básicos que debemos seguir para crear una gráfica con Bokeh son:
  1. Preparar los datos a representar: lista, array de NumPy o serie de Pandas.
  2. Indicar donde se va a generar la gráfica (navegador, notebook…)
  3. Creación de la figura con la función `figure()`.
  4. Definir características a la gráfica tales como: tipo de representación, colores, leyenda (función `line()`).
  5. Mostrar o guardar el resultado con las funciones `show()` y `save()`, respectivamente.

Bokeh

- Está pensado para ser utilizado en la web (es interactivo)
- Pero puede ser útil para crear representaciones más llamativas
- Es un proceso más costoso que lo visto hasta ahora

Bokeh – Ejemplo

```
from bokeh.plotting import figure, show
from bokeh.io import output_notebook

# Preparamos los datos
x = [0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5]
y0 = [2*i**2+3 for i in x]
y1 = [10**(2*i) for i in x]
y2 = [10**(i)+2*i for i in x]

# La gráfica se mostrará en el notebook
output_notebook()

# Creacción de la figura
p = figure(tools="pan,box_zoom,reset,save", y_axis_type="log",
y_range=[0.001, 10**8],
title="Ejemplo eje logaritmico",x_axis_label='sections',
y_axis_label='particles')

# Definición de características
p.line(x, x, legend_label="y=x")
p.circle(x, x, legend_label="y=x", fill_color="white", size=8)
p.line(x, y0, legend_label="y=2x^2+3", line_width=3)
p.line(x, y1, legend_label="y=10^(2x)", line_color="red")
p.circle(x, y1, legend_label="y=10^(2x)", fill_color="red",
line_color="red", size=6)
p.line(x, y2, legend_label="y=10^x+2x", line_color="orange",
line_dash="4 4")
# Mostrar la figura
show(p)
```

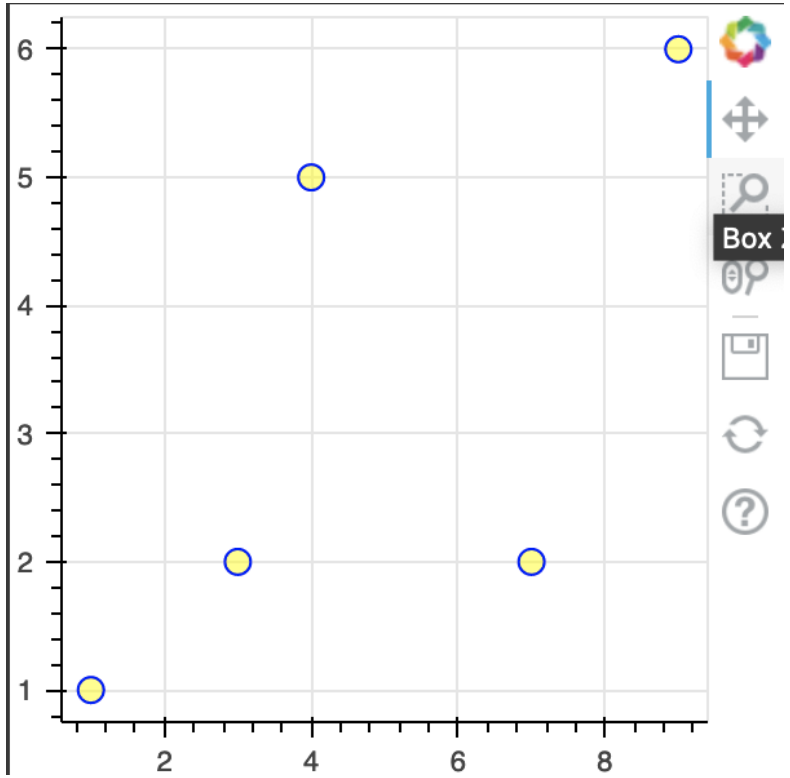


Bokeh – Tipos básicos de gráficos

- Para realizar una representación básica del tipo scatter plot podemos elegir entre diferentes geometrías como: circle, square, rect, triangle, etc.
- Una vez creada la figura podemos elegir cualquiera de los métodos del objeto Figure, incluso combinaciones de varios.

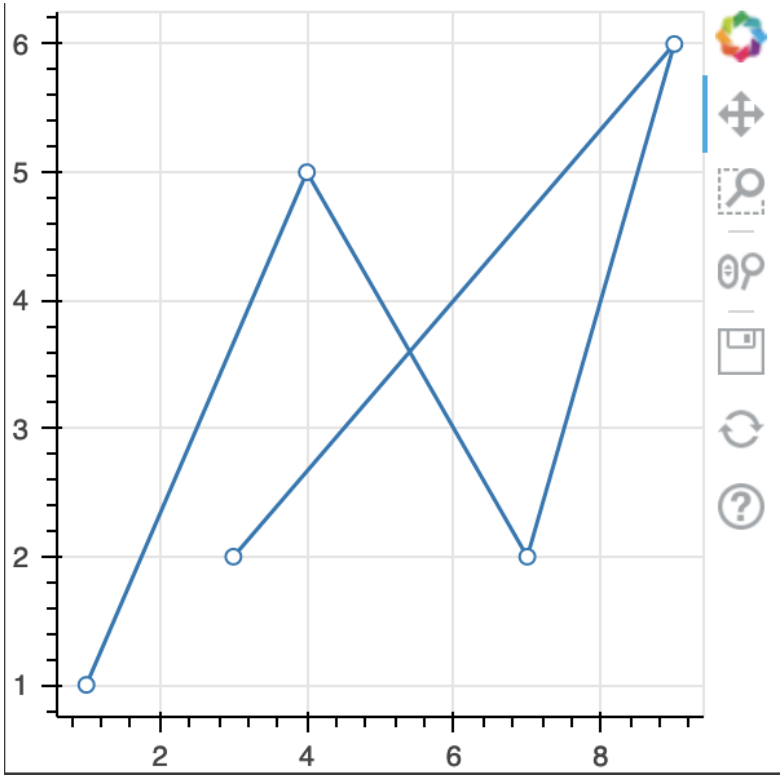
Bokeh – Ejemplo

```
# Nos creamos a mano los datos
x = [3, 9, 7, 4, 1]
y = [2, 6, 2, 5, 1]

# Creamos la figura con las herramientas por defecto
p = figure(plot_width=300, plot_height=300)

# Ejemplo de utilización de la geometría círculo. En
este caso podemos especificar parámetros
# como el tamaño (size), el color (color), y la
trasparencia (fill_alpha, cuyos valores
# están dentro del intervalo [0,1]).
p.circle(x, y, size=10, line_color="blue",
fill_color="yellow", fill_alpha=0.5)

# Mostramos el resultado
show(p)
```



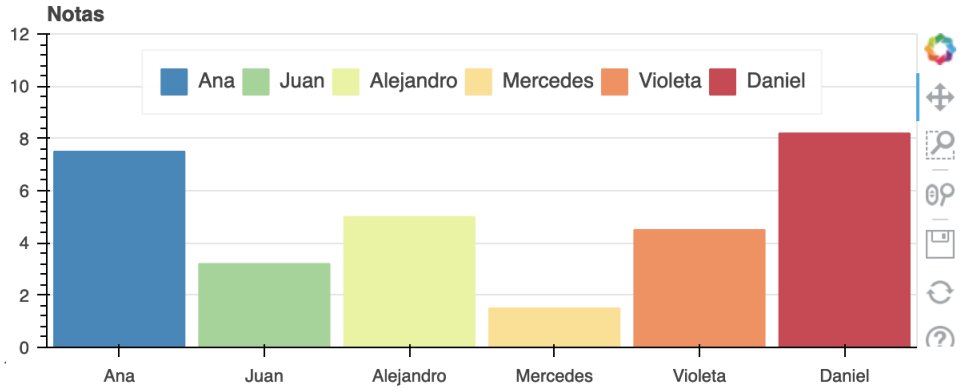
Bokeh – Ejemplo

```
# Utilizamos los mismos datos que en el ejemplo anterior

# Creamos la nueva figura
p = figure(plot_width=300, plot_height=300)

# Ejemplo de utilización de la geometría círculo y línea
p.line(x, y, line_width=1.5)
p.circle(x, y, fill_color="white", size=6)

# Mostramos el resultado
show(p)
```



Bokeh – Ejemplo

```
from bokeh.models import ColumnDataSource
from bokeh.palettes import Spectral6

dic = {'alumnos': ['Ana', 'Juan', 'Alejandro', 'Mercedes',
'Violeta', 'Daniel'],
'notas': [7.5, 3.2, 5.0, 1.5, 4.5, 8.2],
'color': Spectral6}
data_alumns = pd.DataFrame(dic)

source = ColumnDataSource(data_alumns)
p = figure(x_range=list(data_alumns.alumnos.values),
plot_height=250,
y_range=(0, 12), title="Notas")
p.vbar(x='alumnos', top='notas', width=0.9, color='color'
legend_field="alumnos", source=source)

p.xgrid.grid_line_color = None
p.legend.orientation = "horizontal"
p.legend.location = "top_center"

show(p)
```

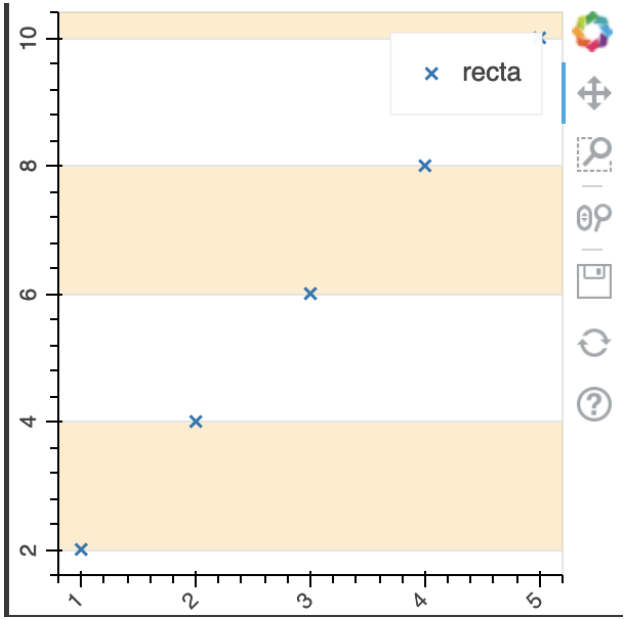


Bokeh – Definición de características

- **Color** del gráfico: Bokeh acepta la especificación de los colores en diferentes formatos
- **Apariencia de la línea**: los parámetros más habituales son line_color, line_alpha, line_width y line_dash.
- **Rellenado de la geometría**: para definir el relleno tenemos fill_color y fill_alpha.
- **Propiedades del texto**: contamos con los parámetros text_font, text_font_size, text_color y text_alpha.
- **Propiedades de los ejes**: para definir las propiedas de los ejes es necesario acceder al objeto Axis a través de alguno de estos métodos axis, xaxis o yaxis. A continuación se pueden definir diversas propiedades como: etiqueta para el eje (axis_label), color (major_label_text_color, axis_line_color), orientación (major_label_orientation) o anchura del eje (axis_line_width).
- **Grid**: podemos añadir un fondo de rejilla a nuestra gráfica en ambos ejes mediante los parámetros xgrid e ygrid, para cada uno de los cuales podemos especificar características como color de línea o de banda (grid_line_color, band_fill_color), trasparencia (band_fill_alpha).
- **Leyenda**: podemos incluir de forma explícita una leyenda a través del parámetro legend de las diferentes opciones de geometría.

Bokeh – Ejemplo

```
# Ejemplo de modificación de algunas características de
la gráfica:

from math import pi

p = figure(plot_width=300, plot_height=300)
p.x([1,2,3,4,5], [2,4,6,8,10], size=8, line_width=1.5,
legend_label="recta")

p.xaxis.major_label_orientation = pi/4
p.yaxis.major_label_orientation = "vertical"

# Color de línea para el xgrid
p.xgrid.grid_line_color = None

# Color y trasparencia de relleno para el ygrid
p.ygrid.band_fill_alpha = 0.2
p.ygrid.band_fill_color = "orange"

show(p)
```

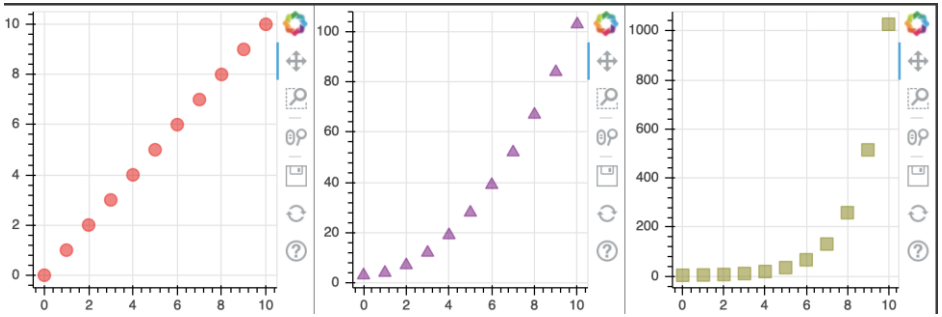


Bokeh – Mallas de gráficas

- Bokeh también nos ofrece la posibilidad de representar varias gráficas en una malla o matriz.
- El módulo bokeh.layouts proporciona las funciones row, colum y gridplot para representar **gráficas en una fila, en una columna o en una malla**, respectivamente.
- El procedimiento es muy sencillo para todas ellas, una vez que hemos creado las figuras, solo tendremos que indicarle cómo queremos visualizarlas.

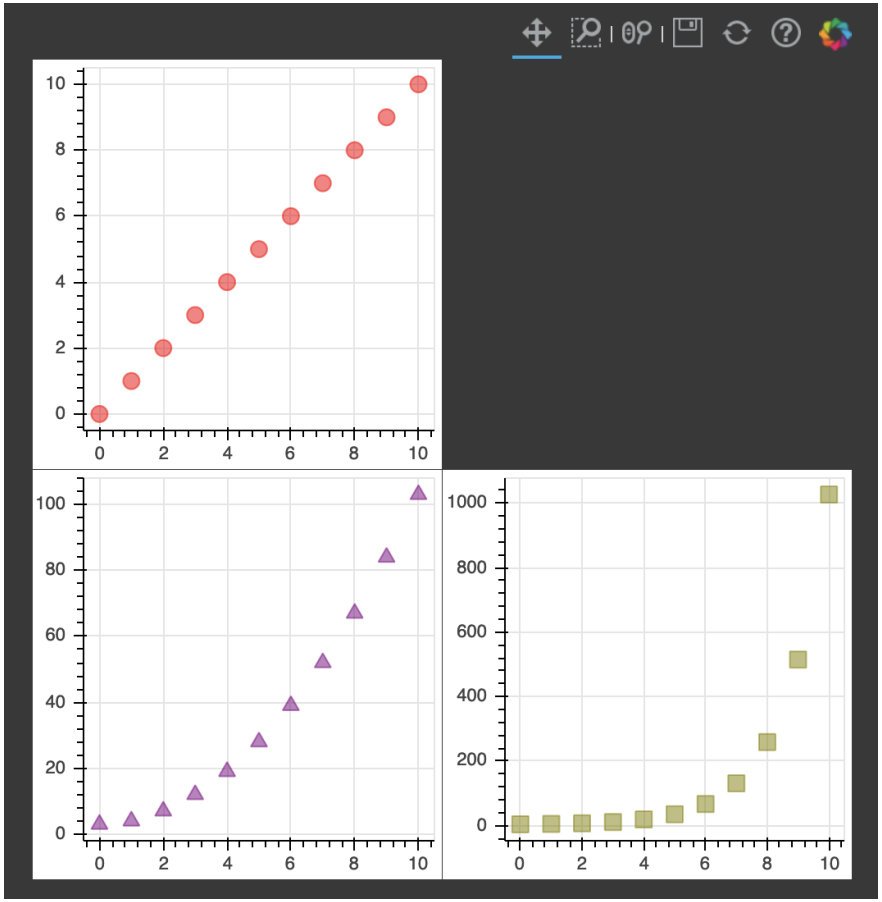
Bokeh – Ejemplo

```
from bokeh.layouts import row

# Preparamos los datos
x = list(range(11))
#y0, y1, y2 = x, [10-i for i in x], [abs(i-5) for i in
x]
y0, y1, y2 = x, [i**2+3 for i in x], [2**i+3 for i in x]

# Creacción de la figura1
s1 = figure(width=250, plot_height=250)
s1.circle(x, y0, size=10, color="red", alpha=0.5)

# Creacción de la figura2
s2 = figure(width=250, height=250)
s2.triangle(x, y1, size=10, color="purple", alpha=0.5)

# Creacción de la figura3
s3 = figure(width=250, height=250)
s3.square(x, y2, size=10, color="olive", alpha=0.5)

# Mostramos las figuras en una fila
show(row(s1, s2, s3))
```



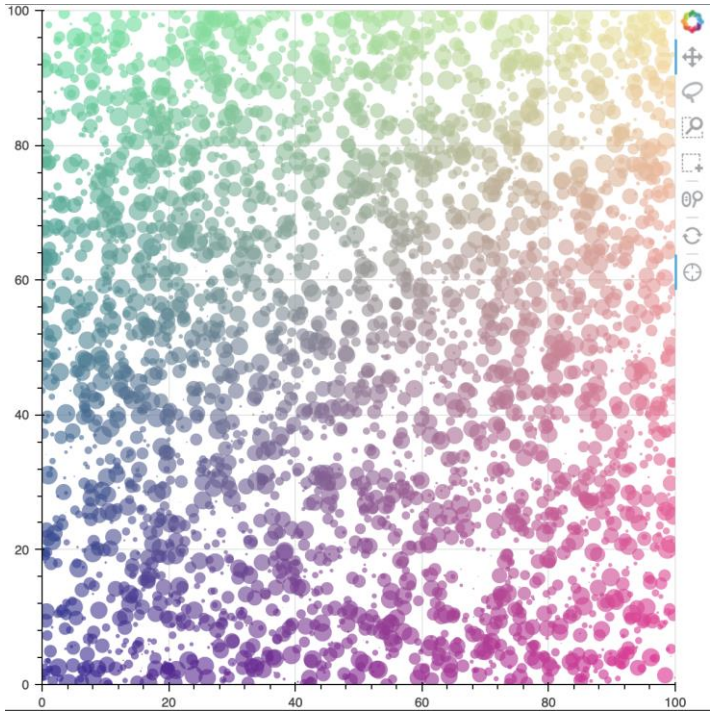
Bokeh – Ejemplo

```
from bokeh.layouts import gridplot

# Ponemos todas la figuras en un grid de 2x2
p = gridplot([[s1, None], [s2, s3]])

show(p)
```



Bokeh – Ejemplo de la potencia de la librería

```
import numpy as np
from bokeh.plotting import figure, show
from bokeh.io import output_notebook

# Preparamos algumnos datos
N = 4000
x = np.random.random(size=N) * 100
y = np.random.random(size=N) * 100
radii = np.random.random(size=N) * 1.5
colors = ["#%02x%02x%02x" % (int(r), int(g), 150) for r,
g in zip(np.floor(50+2*x), np.floor(30+2*y))]
# Elegimos qué herramientas incluir en el toolbox
TOOLS="crosshair,pan,wheel_zoom,box_zoom,reset,box_selec
t,lasso_select"

# Crear un nuevo gráfico, con las herramientas
anteriores y rangos de ejes explícitos
p = figure(tools=TOOLS, x_range=(0,100),
y_range=(0,100))

# Añadir símbolos de círculos con colores y tamaños
vectorizados, según los datos
p.circle(x,y, radius=radii, fill_color=colors,
fill_alpha=0.6, line_color=None)

# Mostrar el gráfico
show(p)
```



¿Preguntas?

Ejemplos aplicados

Análisis y Visualización de los datos

Sergio Pérez Peló

Jesús Sánchez-Oro

Universidad
Rey Juan Carlos