Python para Análisis de datos: Introducción

Sesión 6

Jesús Fernández (fernandez.cuesta@gmail.com)

10 Abril 2019

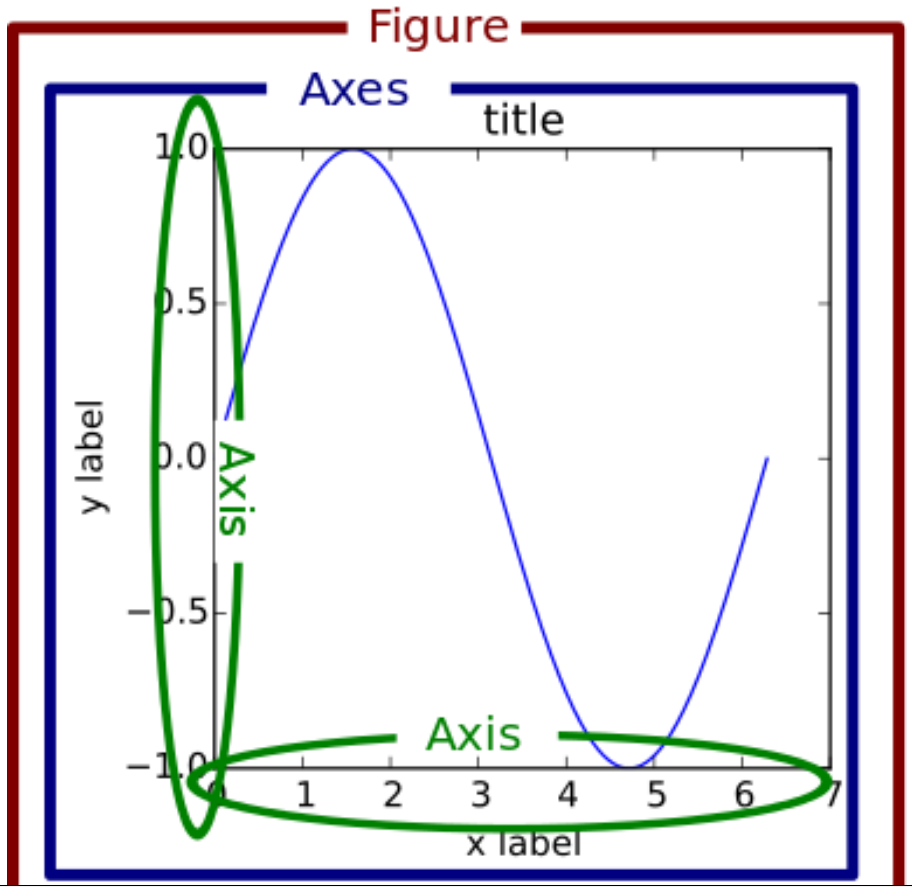
Matplotlib

`pyplot`: módulo con interfaz parecida a MATLAB

Para importar pyplot:

```
import matplotlib.pyplot as plt
# o bien: from matplotlib import pyplot as plt
# Además, sólo en jupyter: %matplotlib inline
```

Partes de una figura



Los componentes más importantes son:

- Figura
- Ejes (axes)
  - región de la figura donde se visualizarán gráficos
- Eje (axis)
  - eje de coordenadas
- Artist
  - cualquier elemento (líneas, leyenda, etc., incluidos ejes)

Como regla general seguiremos los siguientes pasos:

1. Crear figura
2. Obtener ejes
3. Dibujar sobre los ejes
4. Añadir elementos extra (anotaciones, título)
5. Renderizar gráfica

Nota: 1 y 2 se pueden combinar en un mismo comando

```
N = 365
np.random.seed(2983)  # reproducibilidad
t = pd.date_range('1/1/2018', periods=N)
y = 200 * (np.random.randn(N).cumsum() + 40)

#####
fig = plt.figure()  # crea figura
ax = plt.subplot(1, 1, 1)  # crea ejes

ax.plot(t, y)  # dibuja sobre los ejes
#####
```

De forma simplificada, los ejes y la figura se crean simultáneamente:

```
(fix, ax) = plt.subplots(1, 1)  # (n_filas, n_columnas)

ax.plot(t, y)  # dibuja sobre los ejes
```

Importante

En jupyter/ipython, los gráficos aparecerán automáticamente si la primera línea es:

```
%matplotlib inline
```

De lo contrario necesitaremos ejecutar:

```
# [...]
ax.plot(t, y, 'g.-')
plt.show()  # muestra el gráfico
```

... para mostrar cada gráfico

Matplotlib puede dibujar datos organizados en:

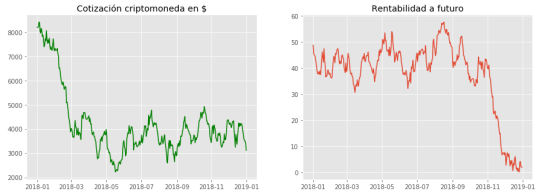
- Listas
- Diccionarios
- Arrays (`np.array`)
- `pandas.DataFrame`

El tipo de datos nativo es `np.array`.
El resto puede requerir limpiar/homogeneizar los datos

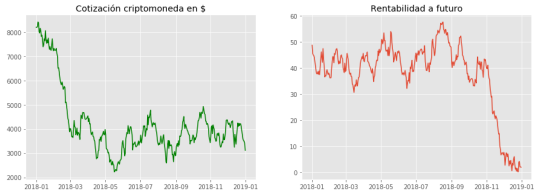
Para crear una tupla (figura, ejes) usaremos:

- `plt.subplots()`
  - sin argumentos: crea una figura con 1 area de dibujo
  - plt.subplots(`n`, `m`): crea una figura con varias zonas de dibujo distribuidas en `n` filas y `m` columnas
  - devuelve la figura y todos los ejes

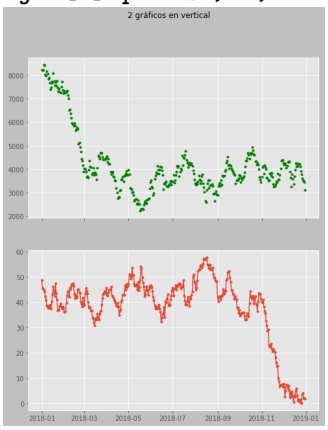
```
fig, (eje1, eje2) = plt.subplots(1, 2)

eje1.plot(t, y, 'g')
eje2.plot(t, z)
```



```
#####
fig, (eje1, eje2) = plt.subplots(1, 2)
#####

eje1.plot(t, y, 'g')
eje2.plot(t, z)
```




```
#####
fig, ejes = plt.subplots(2, 1,
#####
    sharex=True, figsize=(8, 10),
    facecolor='silver')
ejes[0].plot(t, y, 'g.')
ejes[1].plot(t, z, '.-')
```

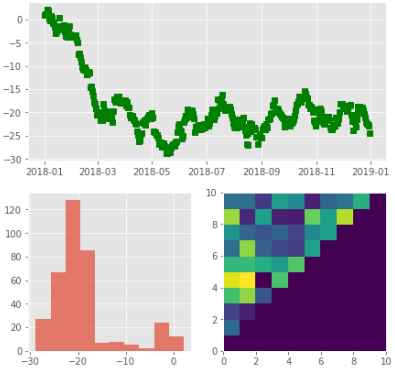


plt.subplot()

¡Ojo!: *plt.subplot() != plt.subplots()*

- (nrows, ncols, index): crea un eje
- devuelve 1 solo eje (al que se haga referencia)

```
fig = plt.figure(figsize=(10, 10))

ax1 = plt.subplot(2, 1, 1)
ax1.plot(t, y, 'gs')

ax2 = plt.subplot(2, 2, 3)
ax2.hist(y, alpha=.7)

ax3 = plt.subplot(2, 2, 4)
ax3.pcolormesh(
  np.tril(np.random.uniform(size=(10, 10))
          -1)
)
```



Visualización de datos

Representamos datos de forma gráfica porque:

- Más fácil y rápido de interpretar
- Cada dominio representa los datos de una manera
- Alto volumen de datos: mejor visualmente

Realizar gráficas desde un lenguaje de programación:

- Gráficos ~~estáticos~~
- Automatizar proceso complejo
  - Combinar procesamiento de datos con su representación
- Personalizable
  - Diferentes soportes (web, publicaciones, presentaciones)

En general:

- Obtención de datos (BB.DD., web, logs, . . .)
- Limpieza y procesamiento de datos
- Selección de parámetros
- Visualización
- Personalización (estilos)

Librerías más usadas:

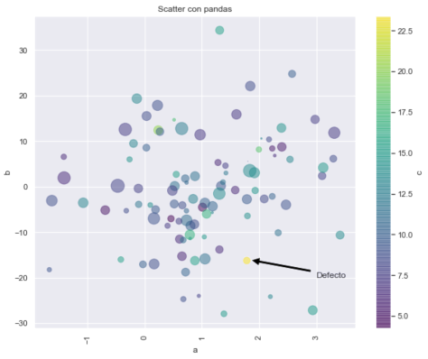
- Matplotlib
  - standard de-facto, basado en MATLAB
  - hacer fácil tareas sencillas y posible tareas complejas
  - versátil, madura y extensa
  - integrado con pandas (<100%)
  - gráficos **estáticos**
  - acepta paquetes de terceros como **extensiones** (plugins)
- Algunas extensiones de Matplotlib:
  - Seaborn
    - enfocado a análisis estadístico
    - mejora el diseño de `matplotlib`
  - Cartopy, folium
    - visualización de datos en mapas
- Bokeh
  - enfocado a gráficos **dinámicos**

- Descargar cuaderno jupyter: matplotlib/pyplot
- Iniciar jupyter (2 opciones):
  - Desde Anaconda Navigator (jupyter notebook > *Launch*)
  - Desde un terminal (`Anaconda Prompt` o terminal de VSCODE):
    `> jupyter notebook`
- Abrir cuaderno descargado en jupyter
- Ejecutar paso a paso de la siguiente sección
  Fragmentos de código encerrados entre '#####'

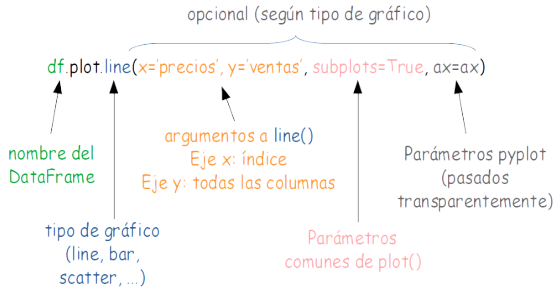
Integración con pandas

- **Descargar y abrir cuaderno jupyter**
- `pyplot` está (parcialmente) integrado en pandas
- Podemos dibujar directamente desde un dataframe de pandas
- se devuelve un objeto "*Axes*" sobre el que poder trabajar
  - no es necesario crear con antelación la figura y los ejes
  - ... aunque suele ser lo recomendable

```
# Crea la figura y los ejes
fig, ax = plt.subplots()

# siempre se devuelve referencia al eje
ax2 = df.plot.scatter(
    'a', 'b', c='c', s=df['d'],
    colormap='viridis', alpha=.5,
    title='Scatter con pandas', rot='vertical',
    ax = ax;
)

ax.annotate(
    'Defecto', xy=(1.9, -17),
    xytext=(2, -20),
    arrowprops=dict(facecolor='black', shrink=0.05)
)

ax2 == ax  # dos referencias al mismo objeto
True
```



Sintaxis



De forma alternativa:

```
df.plot(kind='line', x='precios', y='ventas', subplots=True, ax=ax)
```

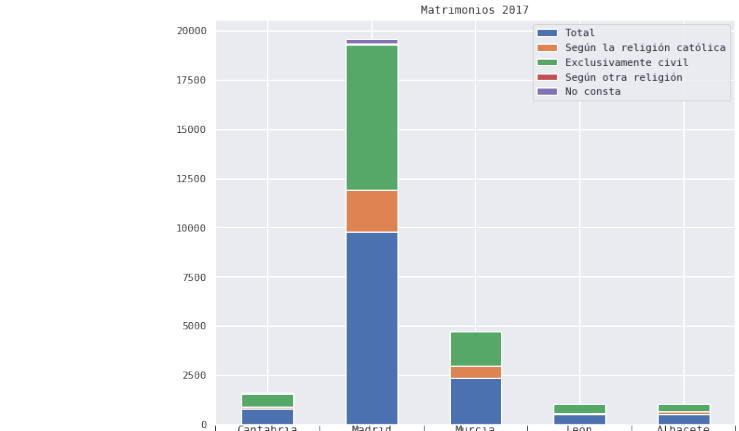
Usar pandas es conveniente y mucho más sencillo:

```
provincias = ['Cantabria', 'Madrid', 'Murcia', 'León', 'Albacete']
index = np.arange(len(provincias)) + 0.3

y_offset = np.zeros(len(marriages.Total.columns))
cell_text = []

for row in marriages.Total.columns:
    _data = marriages.Total.loc[provincias, row]
    plt.bar(index, _data, bottom=y_offset, width=0.5)
    y_offset = y_offset + _data
    cell_text.append(["%1.1f" % (x / 1000.0) for x in y_offset])
cell_text.reverse()
tabla = plt.table(cellText=cell_text,
                  rowLabels=marriages.Total.columns,
                  colLabels=provincias,
                  loc='bottom')
plt.legend(marriages.Total.columns)
plt.title('Total Matrimonios en 2017')
```

con pandas, lo equivalente sería:

```
provincias = ['Cantabria', 'Madrid', 'Murcia', 'León', 'Albacete']
df = ax.Total.loc[provincias]
ax = df.plot.bar(stacked=True,
    table=True, title='Total Matrimonios en 2017')
```

```
provincias = ['Cantabria', 'Madrid', 'Murcia', 'León', 'Albacete']
df = ax.Total.loc[provincias]
ax = df.plot.bar(
    stacked=True,
    table=True,
    title='Total Matrimonios en 2017'
)
```



	Cantabria	Madrid	Murcia	Leon	Albacete
Total	801.0	9798.0	2369.0	509.0	513.0
Según la religión católica	101.0	2094.0	619.0	71.0	139.0
Exclusivamente civil	661.0	7420.0	1736.0	429.0	372.0
Según otra religión	4.0	27.0	20.0	2.0	2.0
No consta	35.0	257.0	3.0	7.0	nan

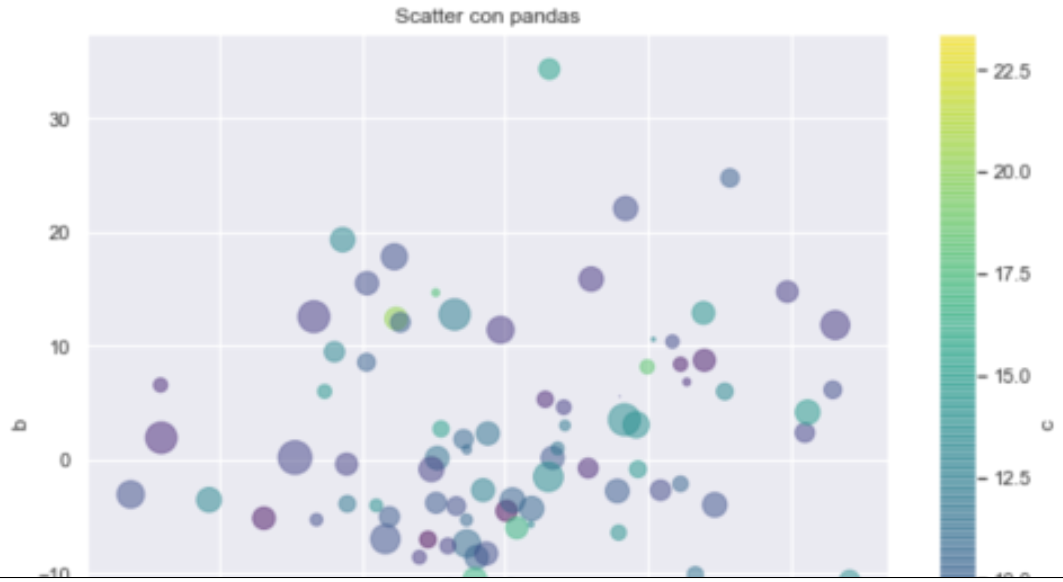
- Generalmente las opciones más usadas en cada tipo de gráfico (título, ejes y posición, color, tamaño de línea, ...) son directamente accesibles desde `pandas.DataFrame.plot()`.
- Para el resto, usar `matplotlib` sobre el objecto "Axes":

```
ax = df.plot(kind='scatter', ...)  # df.plot.scatter(...)
ax.set_yticks(rotation='vertical')
```

Otros tipos de gráficos disponibles

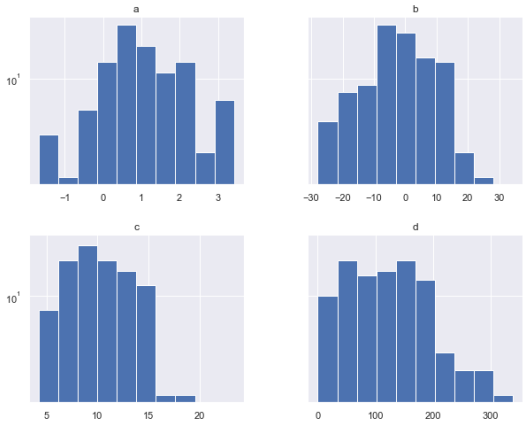
- Gráfico de dispersión (`scatter`)

```
df.plot.scatter(
    x='a', y='b',  # nombres de las columnas del dataframe
    c='c',  # columna con datos de color
    s=df['d'],  # tamaño de los puntos
    colormap='viridis',  # paleta de color
    alpha=.5,  # transparencia
    title='Scatter con pandas',  # título
    rot='vertical',  # rotar etiquetas del eje 'x'
)
```



- Histograma

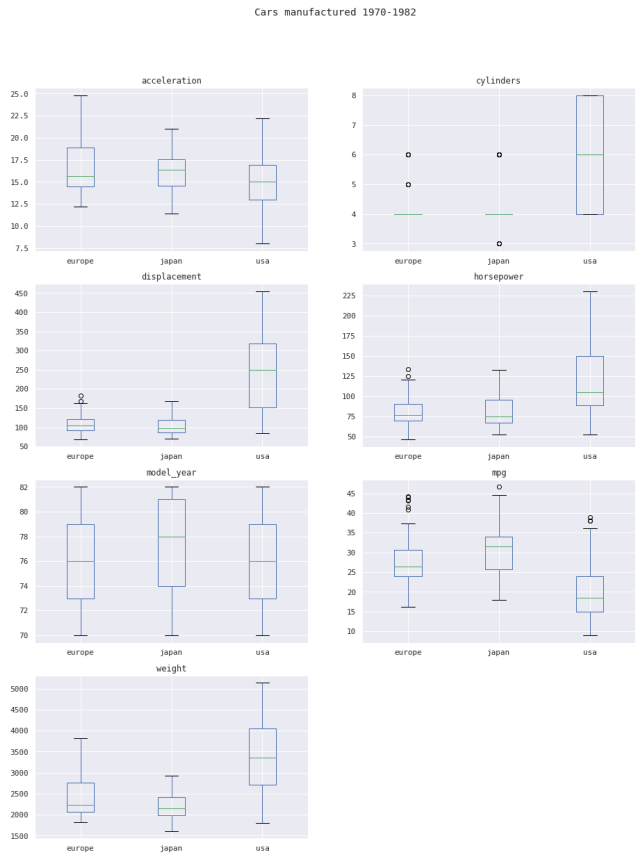
```
df.hist(sharey=True, log=True)
```



- **Series temporales**: aquellos donde el índice del `DataFrame` tiene propiedades de "índice temporal" (`DateTimeIndex`)

- Boxplots

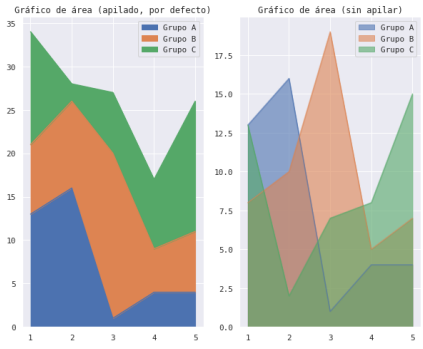
```
df.boxplot(by='origin', ax=ax)
```



:::

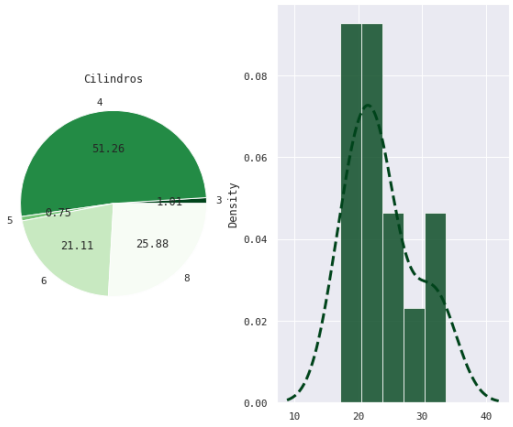
- Área

```
np.random.seed(0)
N = 5
data = pd.DataFrame(
    {'Grupo A': np.random.randint(1, 20, N),
     'Grupo B': np.random.randint(1, 20, N),
     'Grupo C': np.random.randint(1, 20, N)},
    index=range(1, N+1)
)

ax = plt.subplot(1, 2, 1)
data.plot.area(
    stacked=False,
    alpha=.6,
    title='Gráfico de área',
    ax=ax)

ax = plt.subplot(1, 2, 2)
data.plot.area(
    title='Gráfico de área (apilado)',
    ax=ax)
```



- Circular (`pie chart`)

```
ax = df1.plot.pie(cmap=cmap,
                  title='Cilindros',
                  autopct='%.2f', ax=ax)
```



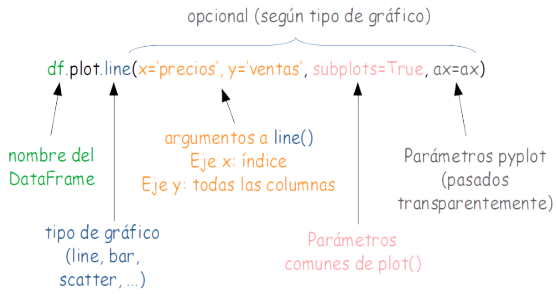
Parámetros opcionales

`pd.DataFrame.plot()`

Es posible pasar argumentos para realizar personalizaciones rápidas.

- Existen opciones específicas para cada tipo de visualización
- También hay opciones **comunes** a todos ellos

- Además, podremos usar las primitivas de `matplotlib`, bien como argumentos adicionales o sobre los ejes



De forma alternativa:

```
df.plot(kind='line', x='precios', y='ventas', subplots=True, ax=ax)
```

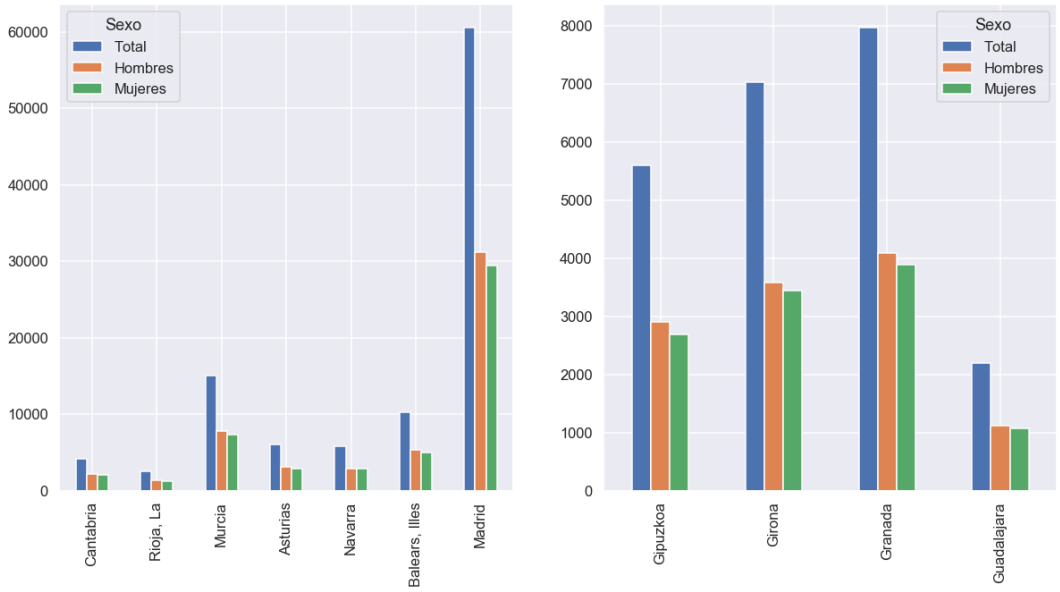
NaN

- Por defecto ignora los datos ausentes (`NaN`)
- Podemos "rellenar los huecos": `df.fillna()`
  - propagando el último valor
  - o con valores preestablecidos
- O interpolar `df.interpolate()`

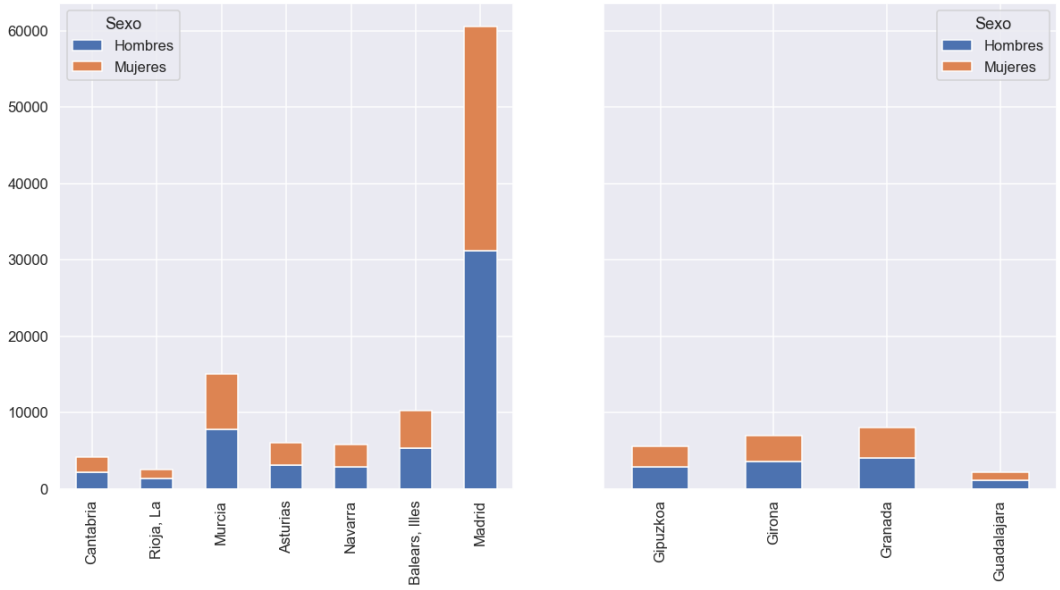
Interactivo/Práctico

- Descargar cuaderno jupyter
- Abrirlo con `jupyter notebook`
- Utiliza las celdas vacías para las respuestas, pruebas, etc.

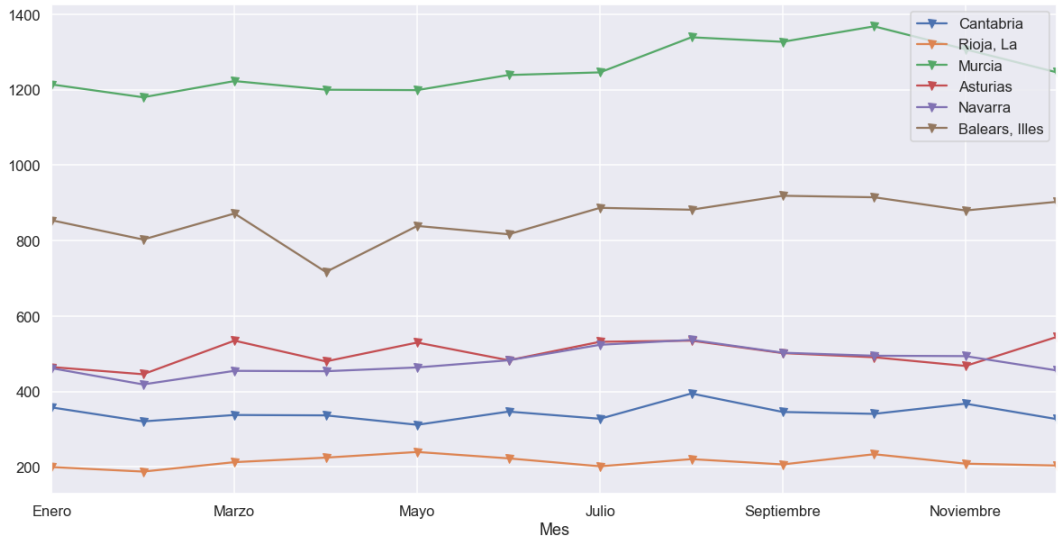
1. `ax`: ejes sobre los que dibujar



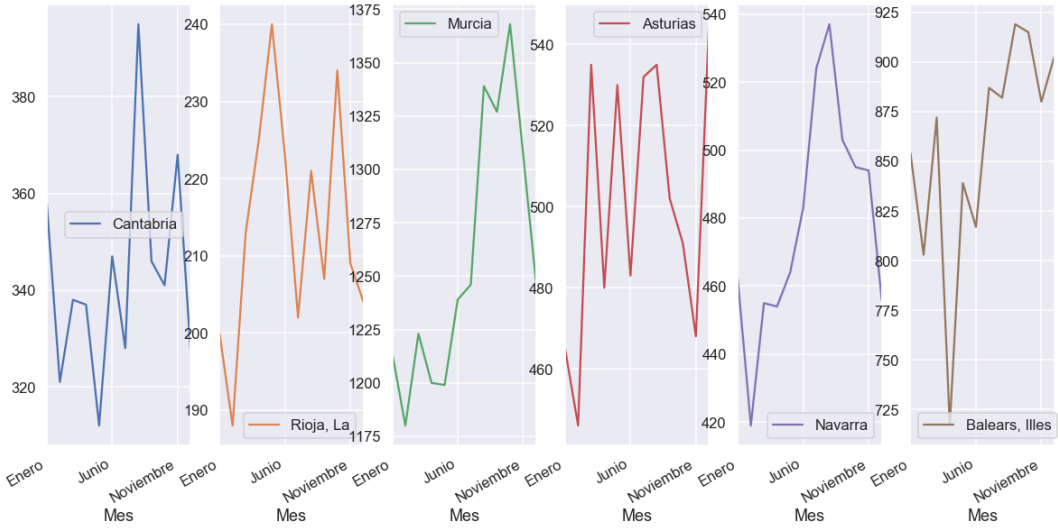
1.2. Barras apiladas (`stacked`)



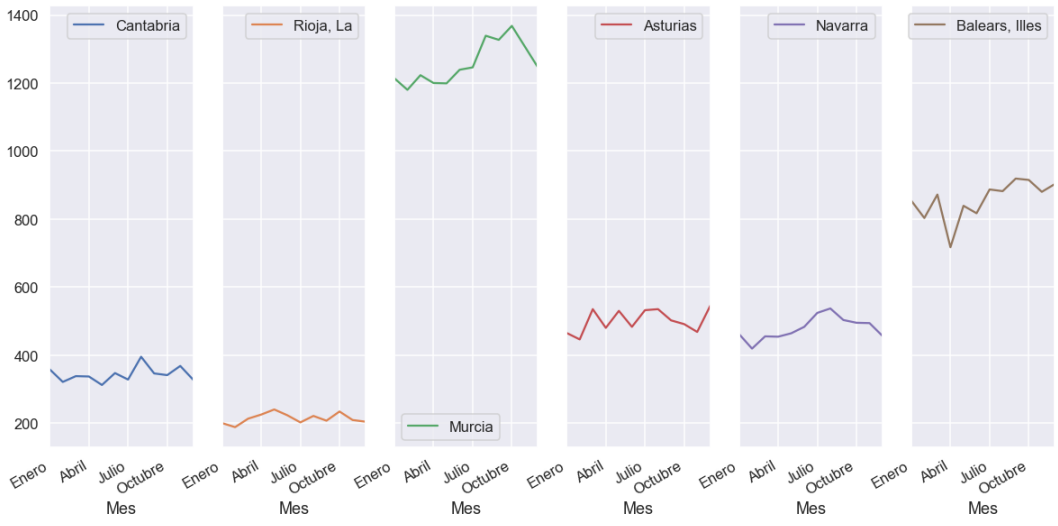
2. `subplots [True|False]`: subplot para cada columna dibujada
   - `layout`: (`n_filas`, `n_columnas`), opcional con `subplot=True`



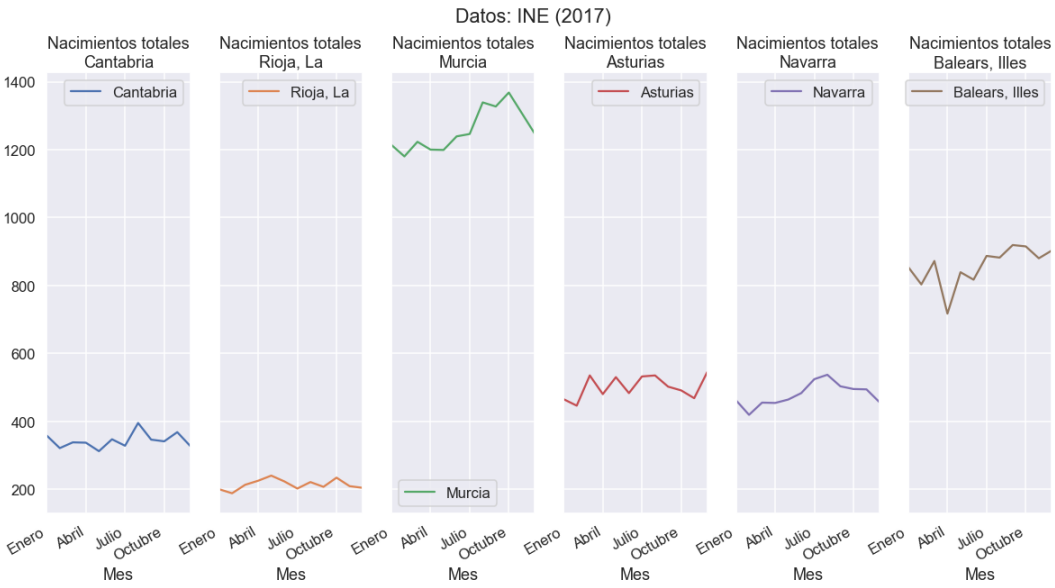
2.2. `layout` como parámetro de `plt.subplots()`



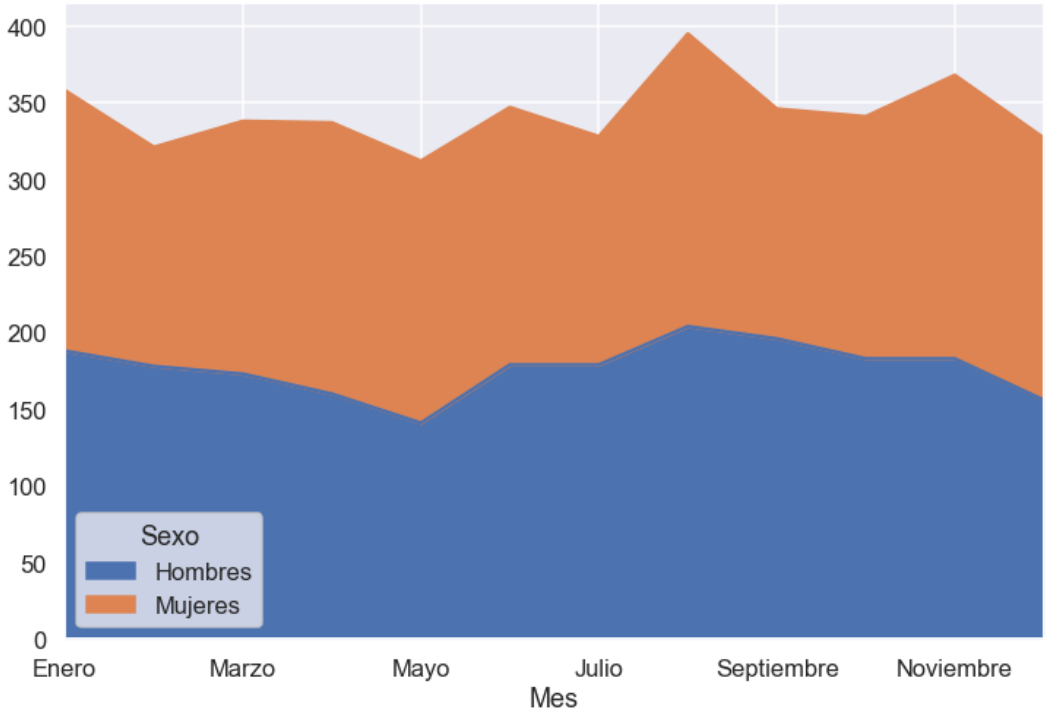
3. `sharex/sharey`: compartir ejes (subplots)



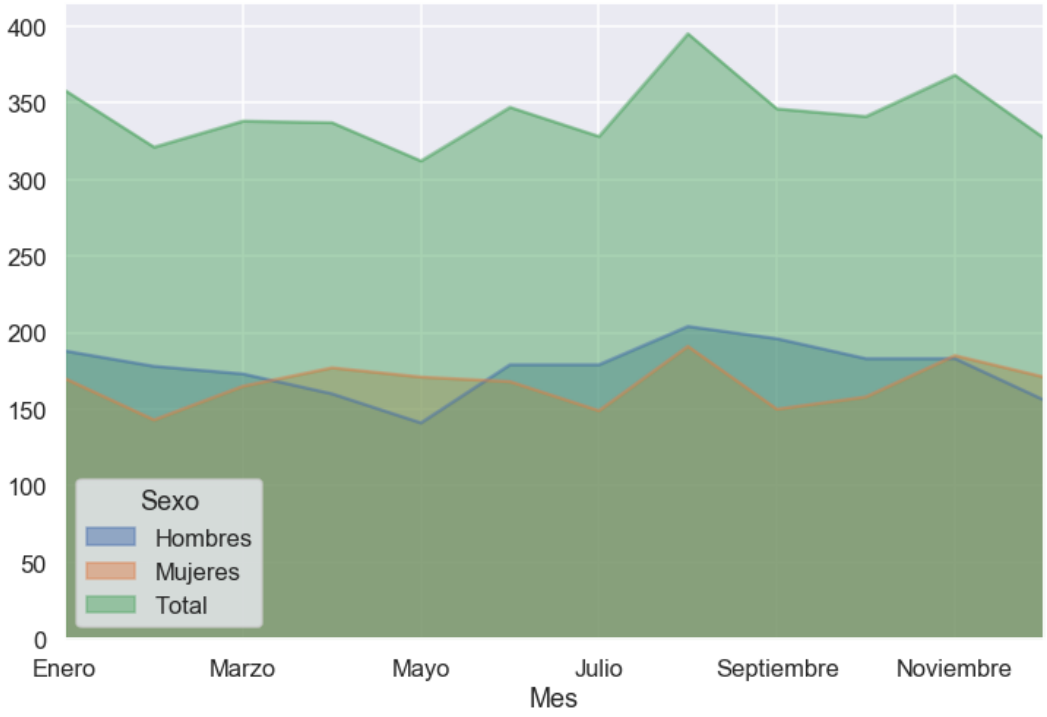
4. `title`:
   - `str`: título de la figura
   - `list(str)`: título de cada subplot



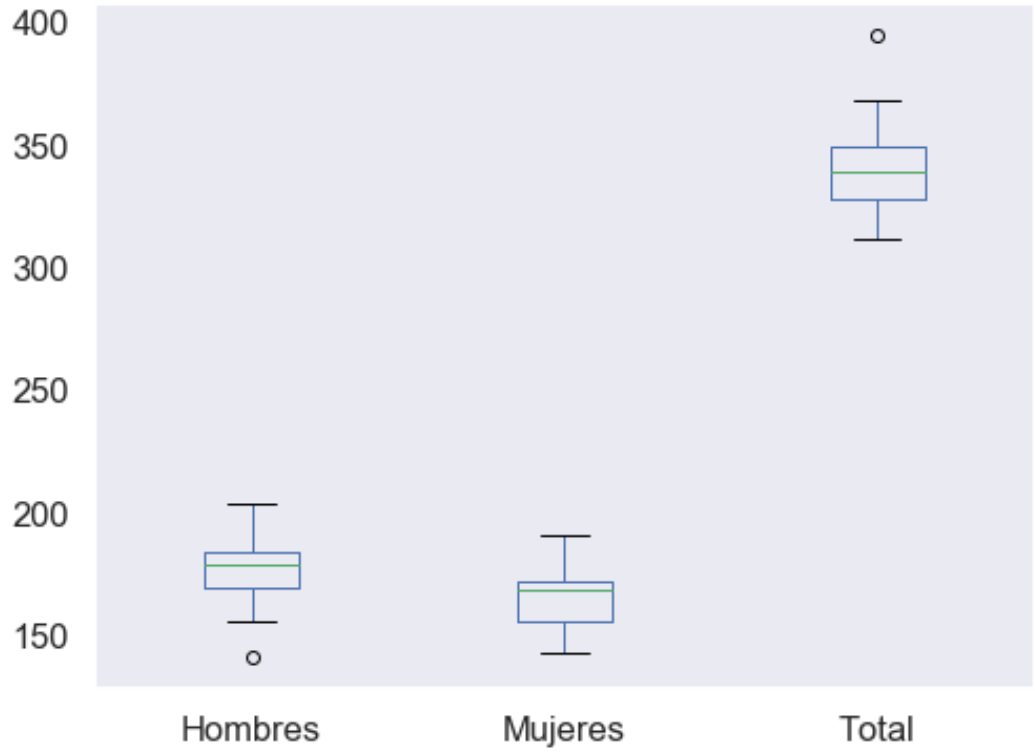
5. `figsize`: tamaño de la figura



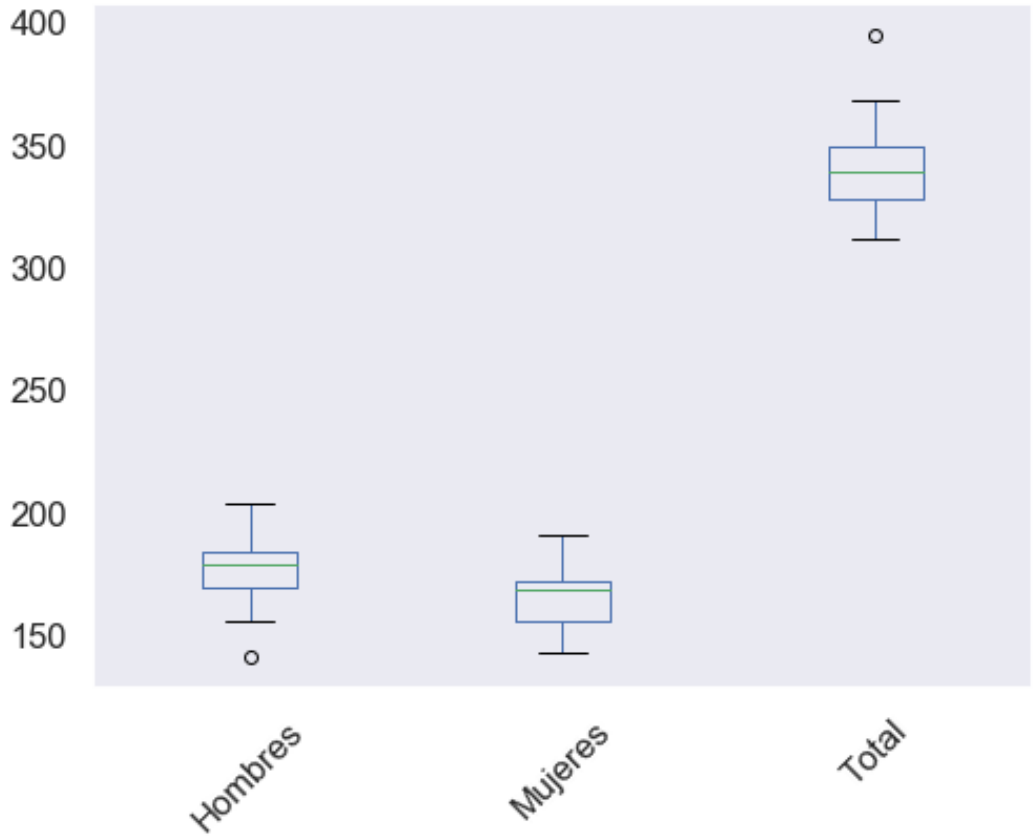
6. `stacked [True|False]`: apilar datos



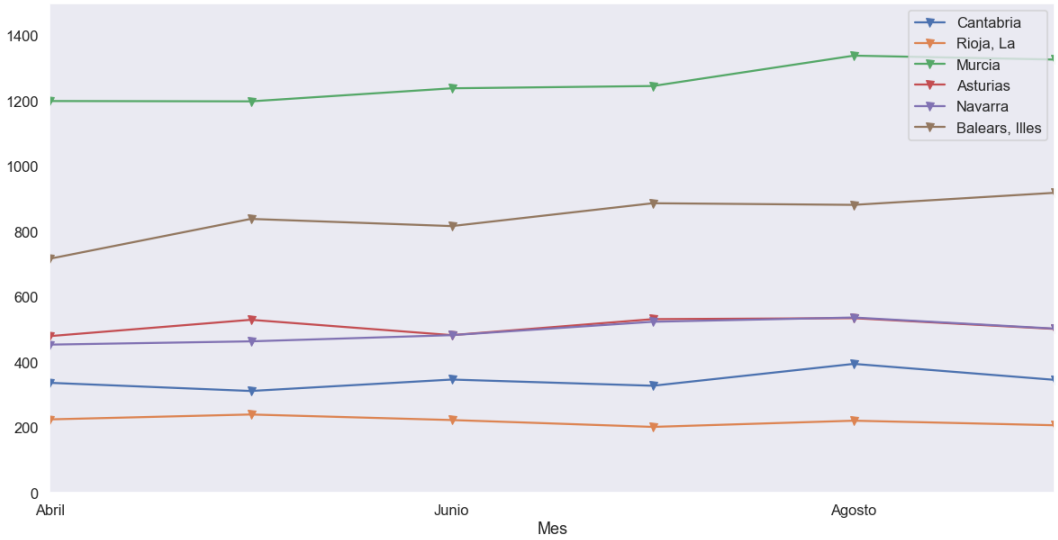
7. `grid [True|False]`: dibujar cuadrícula



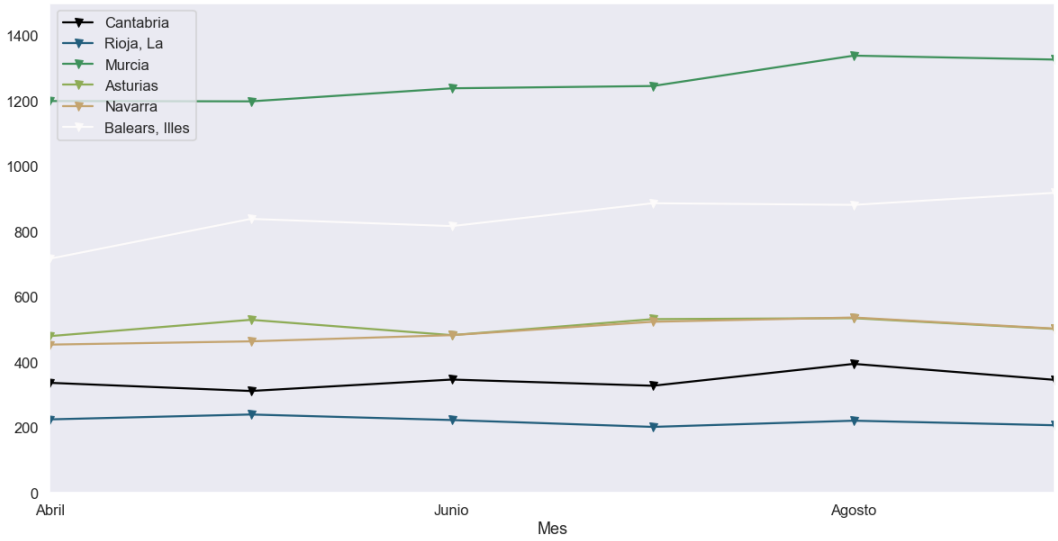
8. `rot`: `'horizontal'`, `'vertical'`, o número (en grados)



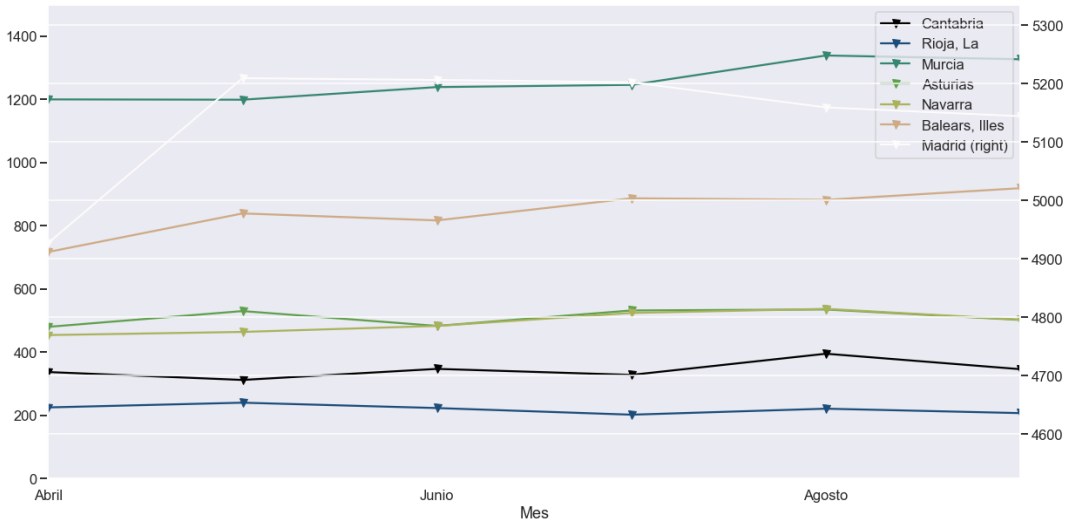
9. `xlim, ylim`: tuplas `(lo, hi)` para delimitar visualización



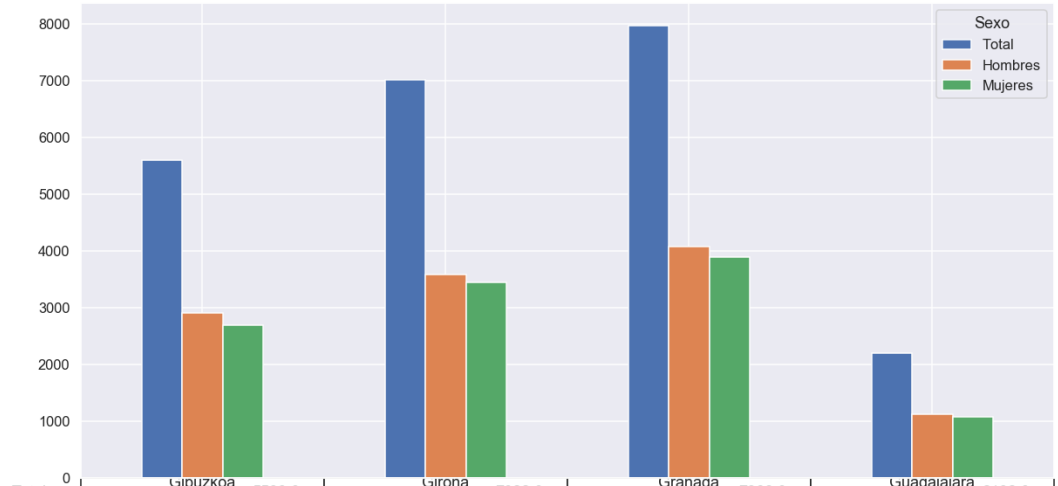
10. `colormap`: mapa de colores (`matplotlib.cm.`)



11. `secondary_y`: segundo eje de ordenadas



12. `table [True|False]`: mostrar tabla bajo el gráfico



	Gipuzkoa	Girona	Granada	Guadalajara
Total	5593.0	7022.0	7968.0	2192.0
Hombres	2909.0	3583.0	4082.0	1119.0
Mujeres	2684.0	3439.0	3886.0	1073.0

Parámetros de visualización con `pyplot`

- Descargar cuaderno jupyter
- Abrirlo con `jupyter notebook`
- Ejecutar paso a paso de la siguiente sección

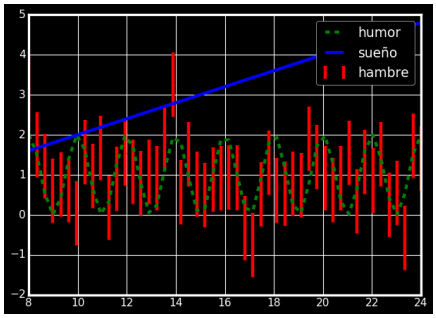
Color/estilo de línea

El color de línea para un gráfico individual se puede controlar mediante una cadena de texto que defina color+estilo+marcador (p.e. `'r--'`), en cualquier orden, todos opcionales:

```
colors = {'b': 'blue', 'g': 'green', 'r': 'red', 'c': 'cyan', 'm': 'magenta',
          'y': 'yellow', 'k': 'black', 'w': 'white'}

lineStyles = {'-': '_draw_solid', '--': '_draw_dashed', '-.': '_draw_dash_dot',
              ':': '_draw_dotted', 'None': '_draw_nothing',
              ' ': '_draw_nothing', '': '_draw_nothing'}

markers = {
    '.': 'point', ',': 'pixel', 'o': 'circle', 'v': 'triangle_down',
    '^': 'triangle_up', '<': 'triangle_left', '>': 'triangle_right',
    '1': 'tri_down', '2': 'tri_up', '3': 'tri_left', '4': 'tri_right',
    '8': 'octagon', 's': 'square', 'p': 'pentagon', '*': 'star',
    'h': 'hexagon1', 'H': 'hexagon2', '+': 'plus', 'x': 'x', 'D': 'diamond',
    'd': 'thin_diamond', '|': 'vline', '_': 'hline', 'P': 'plus_filled',
    'X': 'x_filled', 0: 'tickleft', 1: 'tickright', 2: 'tickup', 3: 'tickdown',
    4: 'caretleft', 5: 'caretright', 6: 'caretup', 7: 'caretdown',
    8: 'caretleftbase', 9: 'caretrightbase', 10: 'caretupbase',
    11: 'caretdownbase', 'None': 'nothing', None: 'nothing', ' ': 'nothing',
    '': 'nothing'
}
```

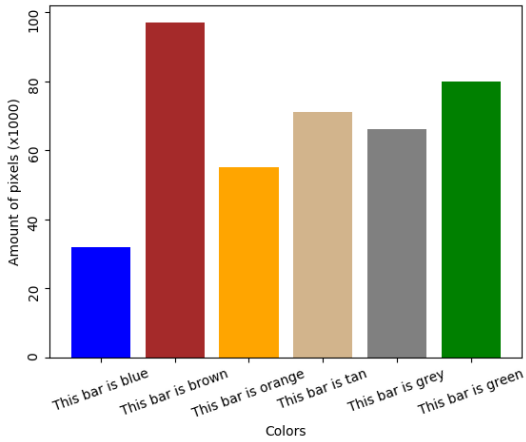
Color/estilo de línea

```
N = 50
np.random.seed(4873)

x = np.linspace(0, 10, N)
k = 0.8
y = k + np.sin(x) * np.random.randn(N)

(fig, ax) = plt.subplots(1)

#####
ax.errorbar(x, y, yerr=k, fmt='.r');
ax.plot(x, 1 + np.cos(np.pi*x), '--g')
ax.plot(x, x/5, 'b')
#####
```

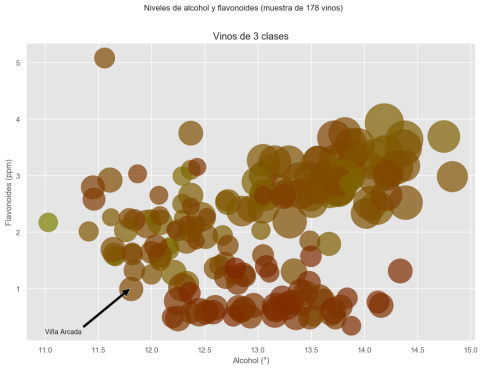


xticks/yticks

```
plt.yticks(rotation='vertical')
plt.xticks(rotation=20)  # 20 grados en sentido antihorario
```



Títulos, etiquetas y anotaciones




```
Título de la figura --------->   fig.suptitle('Niveles de alcohol y flavonoides',
                                              '(muestra de 178 vinos)')
Título del subplot (Axes) --->   ax.set_title('Vinos de 3 clases')
Título de cada eje (Axis) --->   ax.set_xlabel('Alcohol (°)')
                                 ax.set_ylabel('Flavonoides (ppm)')
Anotaciones ----------------->   ax.annotate(
                                     'Viña Arcada', xy=(11.8, 1), xytext=(11, .5)
                                     arrowprops=dict(facecolor='black', shrink=0.
                                 )
```

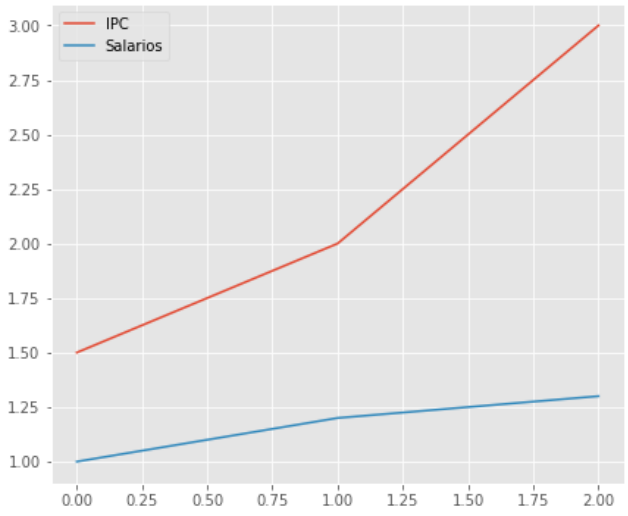
Leyenda

- Se genera automáticamente según datos inferidos de las etiquetas

```
(line1, ) = ax.plot([1.5, 2, 3],
                    label='IPC')
(line2, ) = ax.plot([1, 1.2, 1.3]
                    label='Salari
ax.legend(loc='upper left')
```

`loc`: define dónde emplazar la leyenda

best, *upper right*, *upper left*, *lower left*, *lower right*, *right*, *center left*, *center right*, *lower center*, *upper center*, *center*



:::

Estilos

- Aplican a todos los gráficos generados
- Estilo por defecto en `matplotlib.rcParams`
  - pueden modificarse dinámicamente
- Podemos cambiar a estilos preconfigurados:

```
print(plt.style.available)
plt.style.use('ggplot')
```

- y/o modificar parámetros (`plt.rcParams`) individualmente

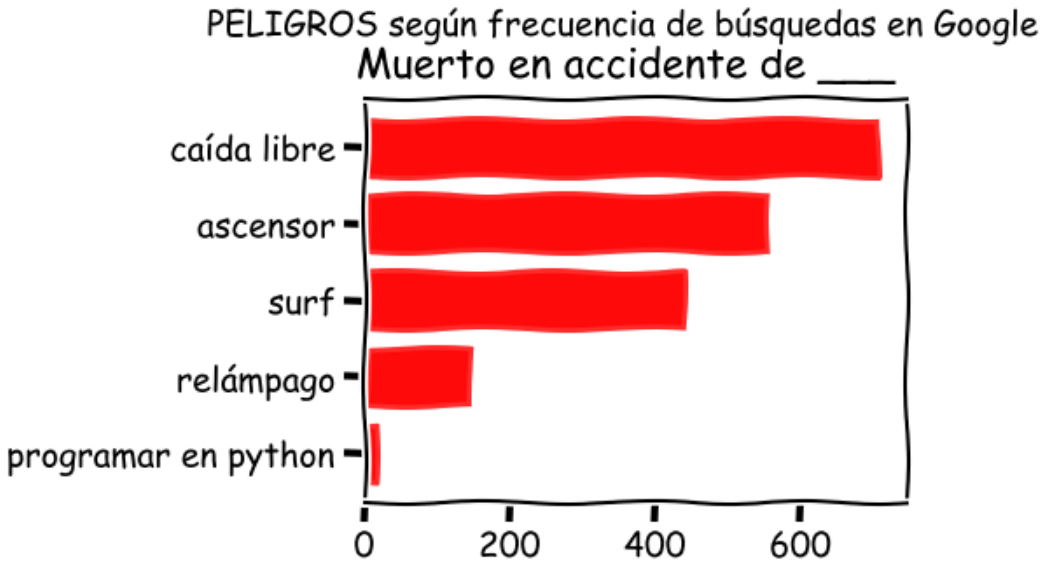
```
plt.rcParams["figure.figsize"] = (20.0, 15.0)
plt.rcParams['font.family'] = ['monospace']
```

- Los cambios aplican a todos los gráficos
  - salvo cambio temporal de estilos:

```
with plt.style.context(('dark_background')):
    plt.plot(np.sin(np.linspace(0, 2 * np.pi)), 'r-o')
```

Otros estilos

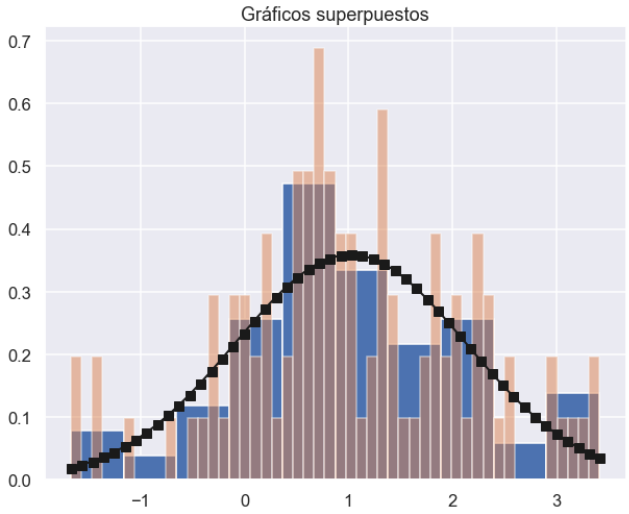
```
with plt.xkcd():
    ...
```



Otros estilos

seaborn

```
import seaborn as sns

sns.set()
sns.set_context("talk")
```



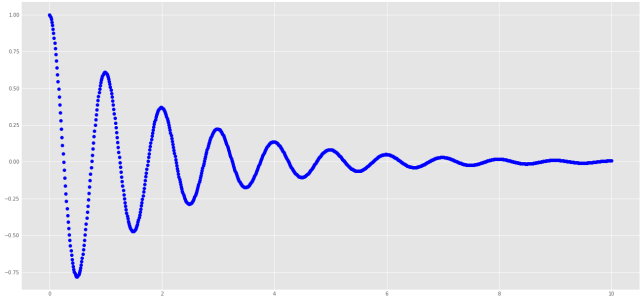
:::

Tipos de gráficos más importantes en pyplot

Lineplot (.plot())

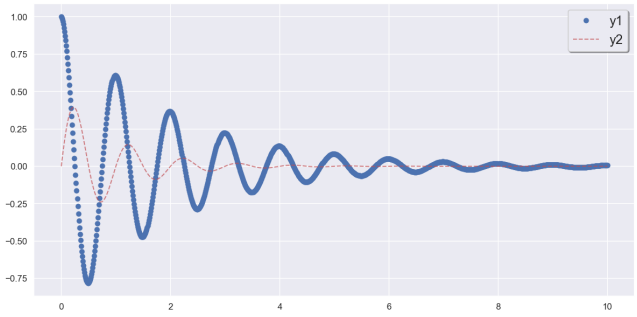
Tipo de gráfico por defecto

```
(fix, ax) = plt.subplots(1, 1)  # (n_filas, n_columnas)

x = np.linspace(0, 10, 1000)
y = np.exp(-x/2) * np.cos(2*np.pi*x)

#####
ax.plot(x, y, 'bo')
#####
```



Dos gráficos al mismo tiempo:

```
(fig, ax) = plt.subplots(1, 1)  # (n_filas, n_columnas)

x = np.linspace(0, 10, 1000)
y1 = np.exp(-x / 2) * np.cos(2 * np.pi * x)
y2 = np.exp(-x) / 2 * np.sin(2 * np.pi * x)

#####
ax.plot(x, y1, 'bo',
        x, y2, 'r--')
#####
```



Dos gráficos al mismo tiempo:

```
(fig, ax) = plt.subplots(1, 1)  # (n_filas, n_columnas)

x = np.linspace(0, 10, 1000)
y1 = np.exp(-x / 2) * np.cos(2 * np.pi * x)
y2 = np.exp(-x) / 2 * np.sin(2 * np.pi * x)

#####
ax.plot(x, y1, 'bo')
ax.plot(x, y2, 'r--')  # <-- dibujar sobre el mismo eje
#####
```

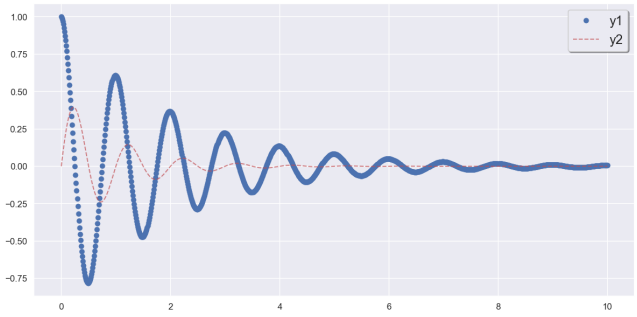


Gráfico de dispersión (Scatterplot)

```
N = 75
np.random.seed(45987230)


fig, (ax1, ax2) = plt.subplots(2, 1,
                               figsize=(7, 10))
a = np.random.randint(low=1, high=11, size=50)
b = a + np.random.randint(1, 5, size=a.size)
x = np.linspace(0, 1, N)
y = np.random.gamma(5, size=N)
colors = np.random.rand(N)

#####
ax1.scatter(x=a, y=b, marker='o', c='r',
            edgecolor='b')
ax2.scatter(
    x, y,  # x=x, y=y
    s=np.random.randint(10,800, N),  # tamaño
    marker='v',  # tipo de marcador
    c=colors,  # colores
    alpha=0.4  # nivel de transparencia
)
#####

ax1.set_title('$a$ vs $b$')
ax2.set_title('$x$ vs $y$')
fig.suptitle("Scatterplot")
```

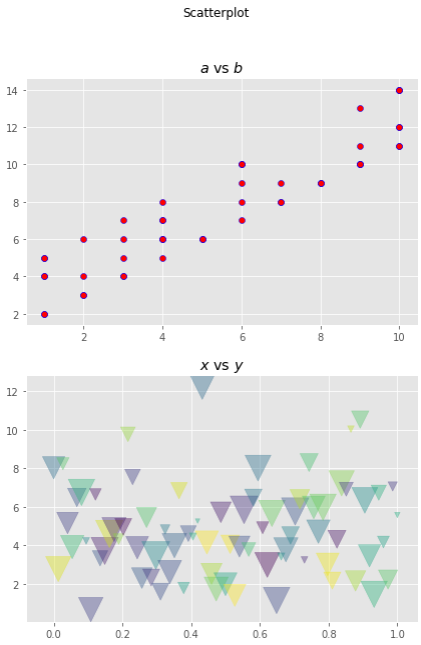
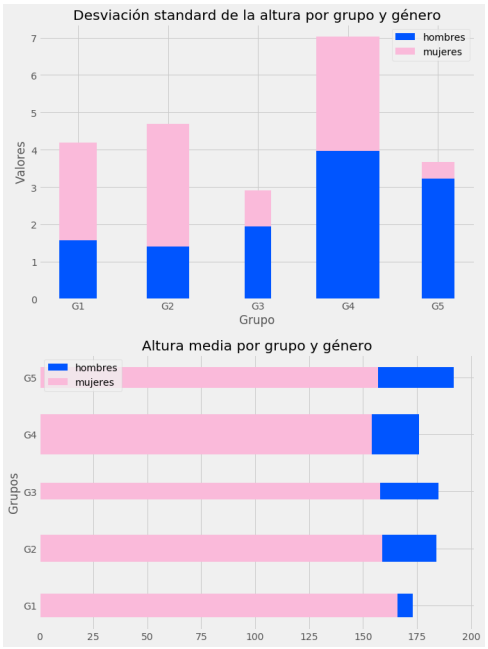


Gráfico de barras (barplot)

```
ax1.bar(
    ind,             # eje de abcisas
    men_std,         # eje de ordenadas
    width,           # grosor de barra
    color='#0055ff'  # color de barra (RGB)
)
ax1.bar(
    ind,
    women_std,
    width,
    color='#fabada',
    bottom=men_std  # apilado
)

# Barras horizontales

ax2.barh(  # cambia orden (y, x)
    ind,             # eje 'y' !
    men_means,       # eje 'x' !
    width,           # grosor de barra
    color='#0055ff' # color (RGB)
)
ax2.barh( # cambia orden (y, x)
    ind,             # eje 'y' !
    women_means,     # eje 'x' !
    width,           # grosor de barra
    color='#fabada' # color (RGB)
```



Histograma

```
fig, ax = plt.subplots(2, figsize=(10,5))
bins = 20
x1 = np.random.gamma(10, size=1000)
x2 = np.random.randn(1000)
ax1, ax2 = ax.flatten()

#####
(ax1_values, _, _) = ax1.hist(x1, bins=bins, facecolor='brown', alpha=.7);
(_, ax2_bins, _) = ax2.hist(x2, alpha=.7, cumulative=True,
                            log=True, orientation='horizontal')
#####
```

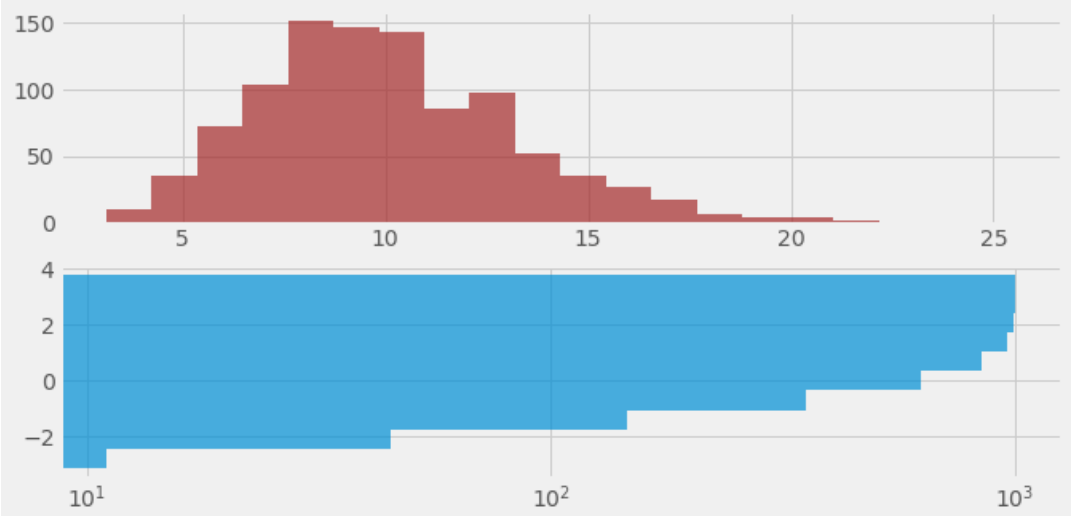
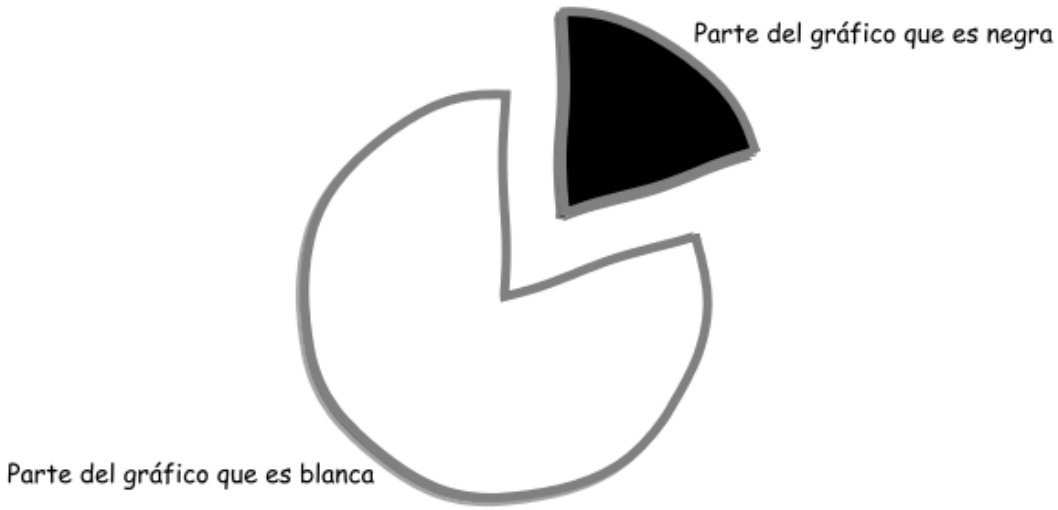


Diagrama circular (pie)

```
ax.pie(
    [80, 20], explode=(0, .5), labels=labels,
    colors=colors, shadow=True, startangle=90
)
# ejes iguales = asegurarnos de que se muestre
# como un círculo
ax.axis('equal');
```



Guardar figura a fichero (de forma programática)

```
from sklearn.datasets import load_iris

iris = load_iris()
plt.style.use('ggplot')
fig, ax = plt.subplots(figsize=(7, 6))
formatter = plt.FuncFormatter(lambda i, *args: iris.target_names[int(i)])

plt.scatter(iris.data[:, 0], iris.data[:, 1], c=iris.target)
plt.colorbar(ticks=[0, 1, 2], format=formatter)

#####
fig.savefig('iris.pdf')
#####
```

Tipos de fichero soportados, según backend:

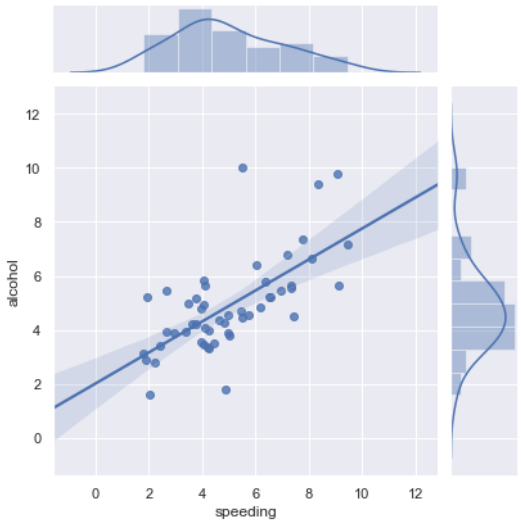
```
print(fig.canvas.get_supported_filetypes())
```

Seaborn

- Descargar cuaderno jupyter
- Abrirlo con `jupyter notebook`
- Ejecutar paso a paso de la siguiente sección

Capa de abstracción que simplifica ciertas tareas enfocadas a análisis estadístico

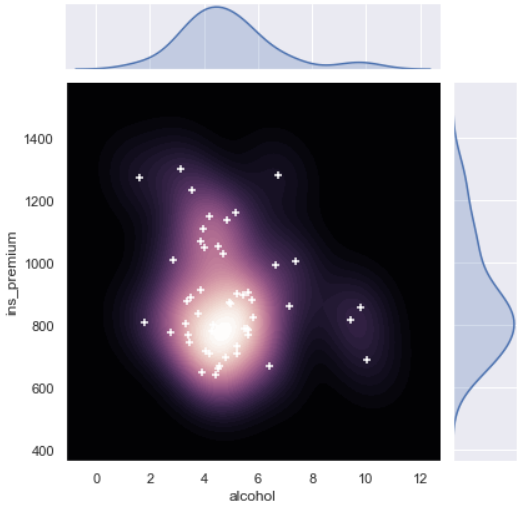
```
sns.set()  # tema por defecto
crashes = sns.load_dataset("car_crashes")

with sns.color_palette("husl", 8):
    sns.jointplot(
        "speeding",
        "alcohol",
        crashes,
        kind='reg'
    )
```

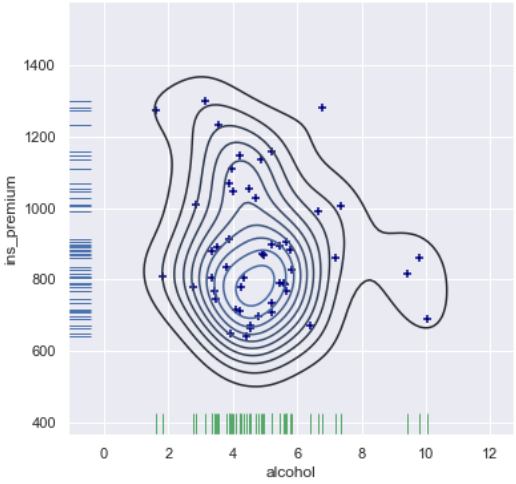


Con `matplotlib`: ...

```
from scipy.stats import gaussian_kde

ax1 = plt.subplot2grid((4, 4), (1, 0), colspan=3, rowspan=3)
ax2 = plt.subplot2grid((4, 4), (0, 0), colspan=3)
ax3 = plt.subplot2grid((4, 4), (1, 3), rowspan=3)

crashes.plot.kde(y='speeding', ax=ax2, sharex=ax1, legend=None)
crashes.plot.hist(y='speeding', bins=6, ax=ax2, sharex=ax1, normed=True,
    legend=None, alpha=.5, color='red')
crashes.plot.scatter(x='speeding', y='alcohol', ax=ax1, color='red', s=50)

ax2.set_ylabel('')
ax2.set_yticks=[]
ax2.set_yticklabels=[]

# No está soportado directamente el rotado en kde
kde_speeding = gaussian_kde(crashes.alcohol)
y = np.linspace(np.amin(crashes.alcohol), np.amax(crashes.alcohol), 100)
ax3.plot(kde_speeding(y), y)
crashes.plot.hist(y='alcohol', ax=ax3, sharey=ax1, normed=True, legend=None,
    orientation='horizontal', alpha=.5, color='red')
```

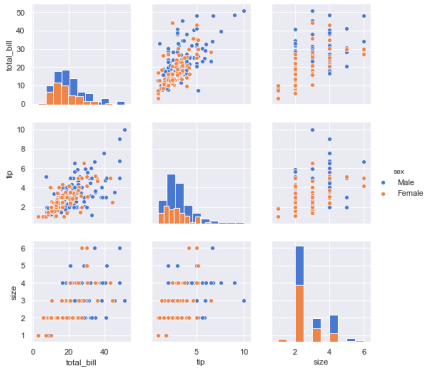
Multitud de gráficas pre-establecidas





Permite cambiar temporalmente la estética mediante `context managers`:

```
with sns.axes_style("ticks") and sns.color_palette("muted"):  # estilo temporal
    sns.pairplot(data=tips,
                 hue='smoker',
                 kind='reg')
```



+ ejemplos

tutorial seaborn

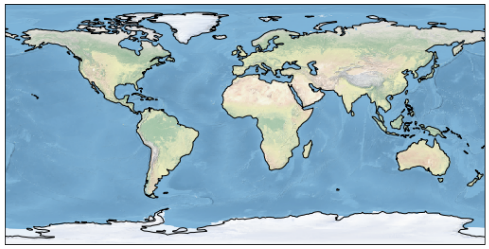
```
x, y = np.random.multivariate_normal(mean, cov, 1000).T
with sns.axes_style("white"):
    sns.jointplot(x=x, y=y, kind="hex", color="k");
```

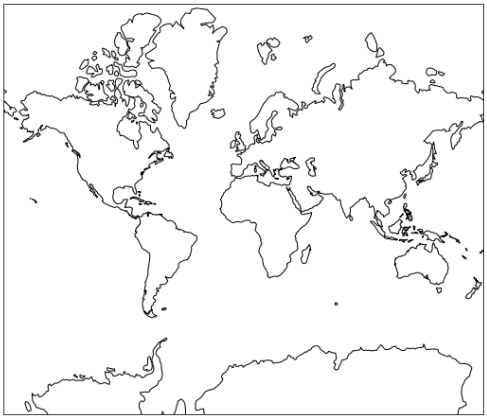
Bokeh

- Permite generar gráficos “interactivos”
- Salida a HTML o integrado en cuaderno `jupyter`

Geovisualización

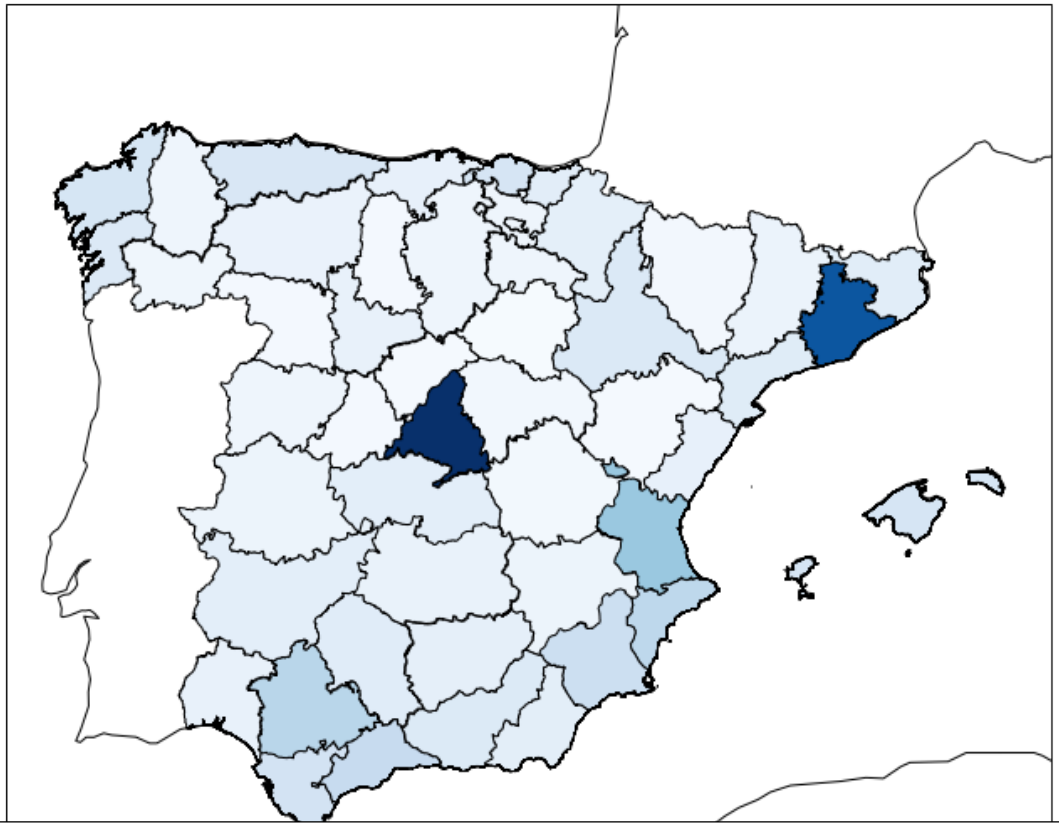
- Librería standard *de facto*: matplotlib-toolkit (**obsoleta**)
- Oficialmente reemplazada por cartopy (muy estática)
- Alternativa: folium





Cartopy

Cuaderno `jupyter`



Folium

Cuaderno `jupyter`

ejemplos online

Cuadernos `jupyter`

1. matplotlib/pyplot
2. pandas
3. caso práctico (**soluciones**)
4. pyplot II
5. seaborn + bokeh
6. cartopy
7. folium