*Clase 08. Visualizaciones en Python (Parte II)*

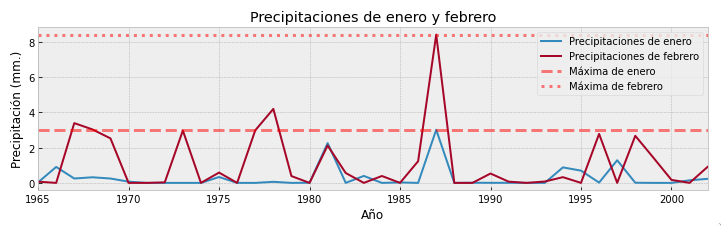
La visualización de los datos es un aspecto fundamental en todo proyecto de Data Science, tanto como herramienta para el análisis como así también para la comunicación efectiva de los resultados. Para esta tarea, existen en Python variedad de librerías de visualización, entre las cuales podemos mencionar *Matplotlib, Seaborn, Plotly y Bokeh*. En esta clase, nos enfocaremos especialmente en **Matplotlib** por su versatilidad, sencillez y facilidad de uso. Para graficar distribuciones de variables categóricas, haremos uso de **Seaborn**, dado que provee un conjunto de comandos que facilitan esta tarea.

# Enriqueciendo las visualizaciones

## Múltiples elementos en un mismo gráfico

Es posible combinar varios elementos dentro de un mismo gráfico o eje (objeto ax). Por ejemplo, podríamos graficar las precipitaciones de enero y febrero para distintos años y al mismo tiempo destacar el máximo y el mínimo de cada uno con una línea de puntos horizontal de la siguiente manera:

| fig, ax = plt.subplots(figsize=(12, 3))  ax.plot(df\_lluvias.index, df['Jan'], label='Precipitaciones de enero') ax.plot(df\_lluvias.index, df['Feb'], label='Precipitaciones de febrero', color='C1')  maximo\_enero = df\_lluvias['Jan'].max() maximo\_febrero = df\_lluvias['Feb'].max()  ax.axhline(maximo\_enero, color='red', linestyle='--', alpha=0.5, linewidth=3, label='Máxima de enero') ax.axhline(maximo\_febrero, color='red', linestyle=':', alpha=0.5, linewidth=3, label='Máxima de febrero')  ax.set\_xlabel('Año') # Etiqueta eje x ax.set\_ylabel('Precipitación (mm.)') # Etiqueta eje y ax.set\_title('Precipitaciones de enero y febrero') # Título ax.set\_xlim(df\_lluvias.index[0], df\_lluvias.index[-1]) ax.legend() # Inserto la leyenda |
| --- |



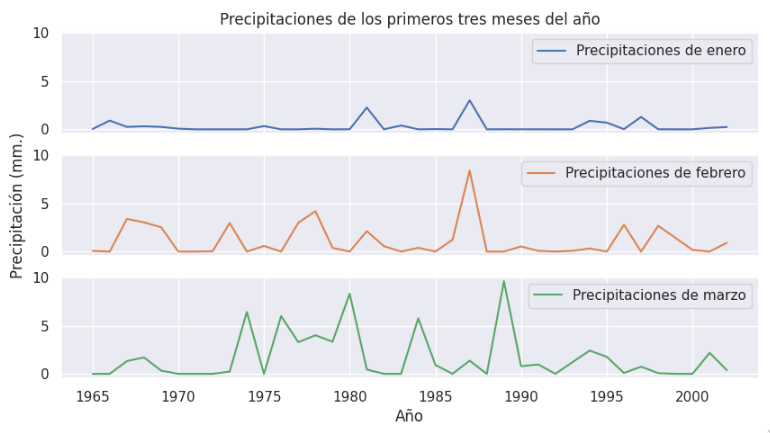
**Observación:** ax.axhline() permite añadir líneas horizontales en la altura que se especifique por parámetro. De igual forma, ax.axvline() permite añadir líneas verticales.

Cuando se grafiquen varios elementos en un mismo objeto ax, es conveniente no sobrecargarlo ya que esto podría entorpecer la lectura. Tengamos en cuenta que el objetivo de toda visualización es transmitir su intención de la manera más simple posible.

## Sub-gráficos

Es posible generar varios gráficos dentro de una misma figura. Esto puede resultar útil, por ejemplo, si queremos comparar distintas variables sin sobrecargar un gráfico particular. Para ello, pasaremos como parámetro *nrows* y *ncols* a la función *plt.subplots()* los cuales representan la cantidad de filas y la cantidad de columnas del gráfico respectivamente. Al hacer esto, **el objeto ax se convierte en un arreglo**, en donde cada elemento es un objeto Axes particular. Para añadir una línea al primer gráfico, nos referiremos al primer elementos del arreglo ax, mediante ax[0]. Para añadir una línea al segundo gráfico, lo haremos con ax[1] y así sucesivamente.

| fig, ax = plt.subplots(nrows=3, ncols=1, figsize=(12, 5), sharex=True, sharey=True)  ax[0].plot(df\_lluvias.index, df\_lluvias['Jan'], label='Precipitaciones de enero') ax[1].plot(df\_lluvias.index, df\_lluvias['Feb'], label='Precipitaciones de febrero', color='C1') ax[2].plot(df\_lluvias.index, df\_lluvias['Mar'], label='Precipitaciones de marzo', color='C2')  ax[0].set\_title('Precipitaciones de los primeros tres meses del año')  ax[2].set\_xlabel('Año')  ax[1].set\_ylabel('Precipitación (mm.)')   ax[0].legend()  ax[1].legend() ax[2].legend() |
| --- |



**Observaciones:**

* Notar los parámetros *sharex* y *sharey* en *plt.subplots().* Esto hace que los subgráficos compartan los ejes y por lo tanto matplotlib únicamente agrega los bins de los años en el último gráfico. Por su parte, *sharey* es particularmente útil en esta ocasión ya que fuerza a que todos los subgráficos tengan la misma escala en el eje y, permitiendo al lector comparar fácilmente cuáles meses tuvieron más o menos precipitaciones. A simple vista se puede ver que enero generalmente acumuló menos precipitaciones que febrero y marzo lo que nos indicaría de que se trata de un mes seco.
* Cada subgráfico puede tener su propio título y etiquetas de ejes. En este caso agregamos un único título general al primer subgráfico y añadimos la etiqueta del eje horizontal se añade únicamente al último subgráfico. Esto último es recomendable siempre que los subgŕaficos compartan el eje x.
* Este tipo de gráficos con varias filas se conocen también como *gráficos en tándem*.
* También es posible generar *grillas de gráficos.* Si por ejemplo pasamos como argumento *nrows=2* y *ncols=2* al método *plt.subplots(),* el objeto *ax* se convierte en una *matriz* o *grilla de gráficos*. Si quisiéramos graficar una línea en el subgráfico superior derecho, habría que ejecutar *ax[0, 1].plot()* haciendo referencia al gráfico de la primera fila y segunda columna. Si en cambio se quisiese agregar un histograma en el subgráfico de la esquina inferior derecha, deberíamos hacerlo con *ax[1,1].hist().*

## Modificando los parámetros globales de Matplotlib

Un aspecto de importancia es mantener un estilo consistente en los distintos gráficos que realizamos. Matplotlib facilita la personalización de los gráficos mediante la modificación de sus parámetros globales. Cuando importamos la librería, Matplotlib establece sus parámetros por defecto los cuales definen las distintas características de las visualizaciones que genera, como por ejemplo el tamaño de las etiquetas, el tamaño de los ejes, los tamaños de fuente, la tipografía, el grosor de las líneas y muchos más.

Se puede ver una extensa lista de los parámetros modificables con la siguiente línea:

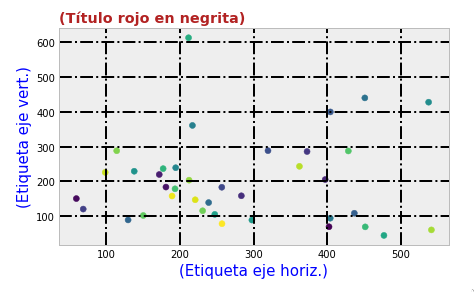
| mpl.rcParams.keys() |
| --- |

Algunos de los parámetros más importantes son los siguientes:

* **'axes.grid':** Permite esconder la grilla si se asigna a False. Por defecto es True.
* **'axes.titleweight':** Grosor de la tipografía de los títulos. Los valores válidos son 'bold', 'normal', 'light'. Por defecto es 'normal'.
* **'axes.titlelocation':** permite modificar la posición del título. Por defecto, los títulos se centran porque este parámetro tiene valor "center". Puede alinear el título a la izquierda si lo asigna a "left".
* **'axes.titlecolor':** color del título. Los colores pu
* **'axes.grid.axis':** Ejes de la grilla, por defecto, está en 'both' lo que significa que la grilla consta de líneas verticales y horizontales.
* **'axes.labelcolor':** Color de la etiqueta de los ejes. Por defecto es 'black'.
* **'axes.labelsize':** Tamaño de las etiquetas. Por defecto es 'large'.
* **'axes.labelweight':** Grosor de la tipografía de las etiquetas. Por defecto es 'normal', pero se pueden poner las etiquetas en negrita asignando 'bold'.
* **'axes.linewidth':** El grosor de los ejes, por defecto 0.8.
* **'grid.alpha':** La transparencia de la grilla, por defecto es 1.0.
* **'grid.color':** Color de línea de la grilla. Por defecto es el color hexadecimal #b2b2b2.
* **'grid.linestyle':** El estilo de línea de la grilla. Por defecto '--'.
* **'grid.linewidth':** Grosor de línea de la grilla. Por defecto es 0.5.
* **'legend.fontsize':** Tamaño de fuente de la leyenda. Por defecto 'medium'.

## Un ejemplo exagerando los cambios

| mpl.rcParams['axes.titleweight'] = 'bold'  mpl.rcParams['axes.titlelocation'] = 'left'  mpl.rcParams['axes.titlecolor'] = 'firebrick'  mpl.rcParams['axes.labelcolor'] = 'blue'  mpl.rcParams['axes.labelsize'] = '10'  mpl.rcParams['axes.labelweight'] = 'light'  mpl.rcParams['axes.linewidth'] = '1'  mpl.rcParams['grid.color'] = 'black'  mpl.rcParams['grid.linestyle'] = '-.'  mpl.rcParams['grid.linewidth'] = '2'   fig, ax = plt.subplots(figsize=(7, 4))  ax.scatter(df\_lluvias['Aug'], df\_lluvias['Sep'], c=df\_lluvias.index)  ax.set\_title('(Título rojo en negrita)') ax.set\_xlabel('(Etiqueta eje horiz.)') ax.set\_ylabel('(Etiqueta eje vert.)') |
| --- |



Una forma (quizás más prolija) de obtener el mismo resultado es usando el método plt.rc() en lugar de asignar los valores al diccionario mpl.rcParams:

| plt.rc('axes', titlelocation='left', titlecolor='firebrick', ...) plt.rc('grid', color='black', linestyle='-.', ...) |
| --- |

Para volver a establecer los parámetros por defecto, ejecutar la siguiente línea:

| mpl.rcParams.update(mpl.rcParamsDefault) |
| --- |

# Una alternativa a Matplotlib: la librería Seaborn

*Seaborn* es una librería de visualización construída por encima de matplotlib. Una de las características de Seaborn que justifican su uso es que se integra muy bien con las estructuras de datos de Pandas, lo que permite generar visualizaciones de data frames muy fácilmente y en pocas líneas de código.

Seaborn provee sus propios estilos de gráficos. Se puede utilizar usar el estilo y colores de Seaborn y continuar graficando normalmente con matplotlib como se hizo hasta ahora con la siguiente línea:

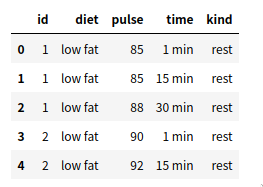
sns.set()

El método *set()* sobreescribe los parámetros internos de matplotlib para que utilice el estilo de Seaborn. De esta manera, todos los gráficos posteriores que realice tendrán un mejor aspecto.

Si bien todas las visualizaciones anteriores podrían generarse con Seaborn, hay que tener en cuenta que este tiene un sintaxis levemente diferente por lo que no vale la pena su utilización para gráficos que podrían realizarse fácilmente con matplotlib. Sin embargo, la verdadera ventaja de Seaborn reside en la visualización de conjuntos de datos con variables categóricas.

Para demostrar esto, importemos el conjunto de datos de ejercicio que nos provee Seaborn

| df\_ejercicio = sns.load\_dataset('exercise') df\_ejercicio = df\_ejercicio.drop('Unnamed: 0', axis='columns') df\_ejercicio.head() |
| --- |

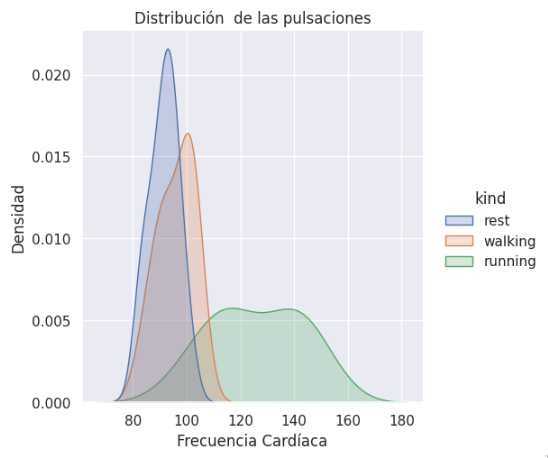


El conjunto de datos contiene mediciones de la frecuencia cardíaca de 30 personas distintas tras realizar un determinado tipo de ejercicio. Las columnas son:

* **id:** el id de la persona sobre la cual se efectuó la medición.
* **diet:** dieta de dicha persona (baja en grasas o sin grasas).
* **pulse:** la frecuencia cardíaca obtenida (en pulsaciones por minuto).
* **time:** tiempo transcurrido tras el inicio del ejercicio.
* **kind:** tipo de ejercicio (en reposo, caminando o corriendo).

Nos podría interesar la forma de la distribución de las pulsaciones para los distintos tipos de actividad tras 30 minutos de haber realizado la misma:

| plt.figure() df\_30\_min = df\_ejercicio[df['time'] == '30 min'] ax = sns.displot(data=df\_30\_min, x='pulse', kind='kde', hue='kind', fill=True) ax.set(xlabel='Frecuencia Cardíaca', ylabel='Densidad', title='Distribución de las pulsaciones') |
| --- |

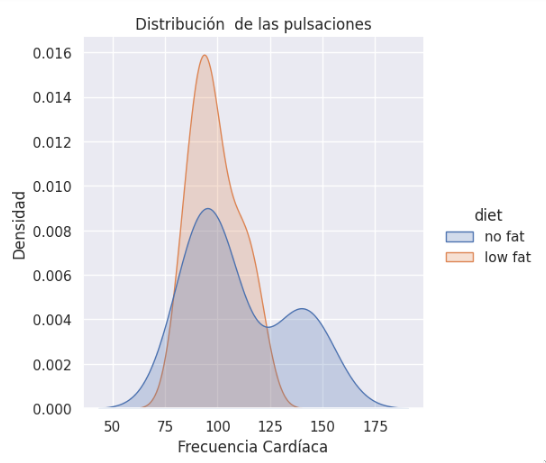


Como era de esperar, las personas que corrieron terminaron con un mayor pulso respecto de los que no lo hicieron.

**Observaciones:**

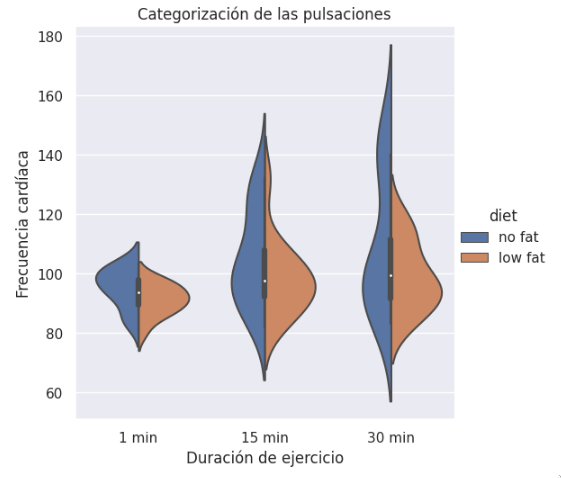
* En primer lugar, el método sns.displot de recibe como parámetro 'data' que recibe un Pandas dataframe y luego recibe nombres de las columnas que se quieren visualizar de dicho dataframe.
* El parámetro *hue* determina en base a qué columna **de variables categóricas** se debe realizar el agrupamiento. En este caso, en base al tipo de ejercicio.
* El parámetro fill añade sombreado a las distribuciones.
* El parámetro kind='kde' establece que queremos una estimación de la distribución de probabilidad a partir de las muestras. Si hacemos kind='hist' obtendremos un histograma.

Si en cambio quisiéramos comparar aquellas personas que tienen una dieta baja en grasas (categoría low fat) respecto de aquellas que no consumen grasas (categoría no fat), sólo hay que cambiar el parámetro hue anterior por la columna 'diet'.



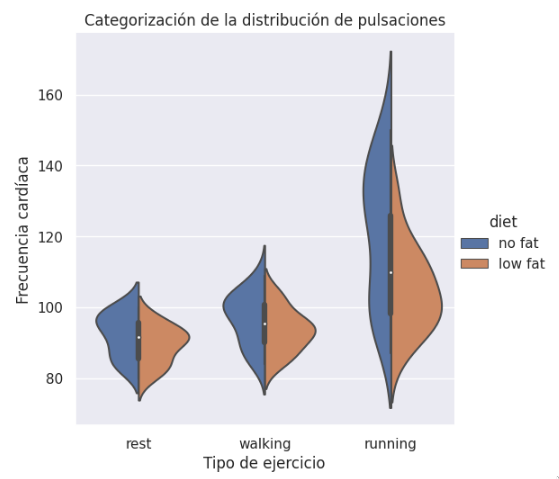
Otra forma de visualizar distribuciones según categorías es mediante el método *sns.catplot():*

| ax = sns.catplot(data=df\_ejercicio, kind='violin', x='time', y='pulse', hue='diet', split=True) ax.set(xlabel='Duración de ejercicio', ylabel='Frecuencia cardíaca', title='Categorización de la distribución de pulsaciones') |
| --- |



O bien, para distintos tipos de ejercicio:

| ax = sns.catplot(data=df\_ejercicio, kind='violin', x='kind', y='pulse', hue='diet', split=True) ax.set(xlabel='Duración del ejercicio', ylabel='Frecuencia cardíaca', title='Categorización de la distribución de pulsaciones') |
| --- |



De esta forma, hicimos un recorrido por las principales herramientas de visualización que ofrece Python.