**Clase 7 - Visualizaciones y primeros pasos con Data Science (Parte I)**

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.

# Introducción a Matplotlib

Matplotlib es la librería de visualización más utilizada en el entorno de Python. Además de su facilidad de uso, permite un alto nivel de personalización de los gráficos. Al igual que las otras librerías con las que estamos trabajando, Matplotlib es open source. Dada su gran flexibilidad de uso, es además la base sobre la que se construyeron otras librerías, tales como Seaborn, que veremos más adelante.

En primer lugar, importamos las librerías que utilizaremos durante la clase

| import numpy as np import pandas as pd  # --- Específicas para los gráficos import matplotlib as mpl import matplotlib.pyplot as plt import seaborn as sns # -------------------------------- |
| --- |

Si se obtiene un error al ejecutar las importaciones, es probable que no se tengan instaladas las librerías requeridas. En este caso, pueden instalarse ejecutando en una celda de código

| !conda install matplotlib !conda install seaborn |
| --- |

o bien desde la Terminal ejecutando

| conda install matplotlib conda install seaborn |
| --- |

La paleta de colores que matplotlib trae por defecto no es muy estética, pero podemos cambiar el estilo fácilmente con la siguiente línea:

| mpl.style.use('bmh') # bmh es el nombre del estilo que usaremos |
| --- |

Puede consultarse una lista de estilos disponibles [en este link](https://matplotlib.org/stable/gallery/style_sheets/style_sheets_reference.html).

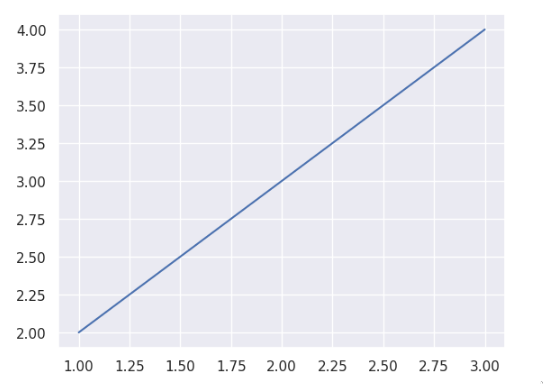
## Interfaces

Existen dos formas de generar gráficos con Matplotlib: mediante la *interfaz basada en estados* o bien mediante la *interfaz orientada a objetos*. Si bien la primera puede parecer más simple al inicio, la segunda opción otorga mayor control sobre los gráficos que generamos, lo que a final de cuentas resulta ventajoso a medida que nuestro código crece y necesitamos generar una mayor cantidad de gráficos.

La interfaz basada en estados fue creada con inspiración en MATLAB, un lenguaje de programación y entorno de desarrollo orientado al uso científico. Esta interfaz provee compatibilidad con los programadores acostumbrados a ese entorno de programación. Para los que trabajan con Python, o se dedican a la programación orientada a objetos en general, la interfaz orientada a objetos provee mayor control y más flexibilidad en el uso de los gráficos.

Con la interfaz basada en estados, se puede graficar una línea que una los puntos (1, 2) y (3, 4) de la siguiente forma:

| plt.plot([1, 3], [2, 4]) |
| --- |



Mientras que para generar el mismo gráfico con la interfaz orientada a objetos:

| fig, ax = plt.subplots() ax.plot([1, 3], [2, 4]) |
| --- |

Como se observa, la función plt.subplots() retorna dos objetos, un objeto de tipo Figure y un objeto de tipo Axes, los cuales asignamos a las variables fig y ax respectivamente. Posteriormente manipularemos el gráfico haciendo referencia a dicho objeto Axes a través de ax.

Hemos pasado como parámetros dos listas a la función plot(). La primera contiene las coordenadas en el eje x de los puntos que queremos unir y la segunda contiene las coordenadas en el eje y.

Si mezclamos el uso de estas interfaces se puede dar lugar a confusión cuando se comienza a programar en Python. Por lo tanto, nos limitaremos a usar únicamente la segunda opción, la cual además recomendamos por ser la más utilizada.

## Comandos básicos

Prosigamos con lo más importante: contextualizar el gráfico. Cuando se realiza un gráfico, siempre es conveniente incluir información acerca de qué es lo que se está visualizando. Para ello se recomienda añadir un título, etiquetar los ejes e insertar una leyenda que informe qué representa cada elemento que se grafica. Al etiquetar los ejes, añadir la unidad de medida en caso de que sea posible.

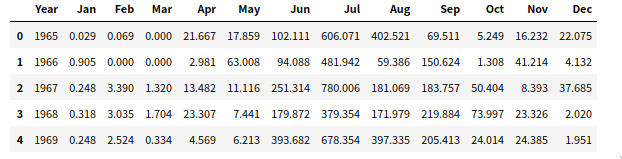
* **Para etiquetar los ejes:** los métodos *set\_xlabel()* y *set\_ylabel()* del objeto ax reciben como parámetro la etiqueta de los ejes x e y respectivamente. La forma de uso es la siguiente: ax.set\_xlabel('Etiqueta eje X').
* **Para añadir un título:** utilice el método *set\_title()* del objeto ax. (ax.set\_title())
* **Para añadir una leyenda:** utilice el método *legend()* del objeto ax. (ax.legend())

## Ejemplo de uso

Como datos de prueba se usará el **conjunto de datos de precipitaciones** de la clase anterior. Este conjunto de datos puede descargarse [en este link](https://www.kaggle.com/abhishekmamidi/precipitation-data-of-pune-from-1965-to-2002).

df\_lluvias = pd.read\_csv("pune\_1965\_to\_2002.csv")

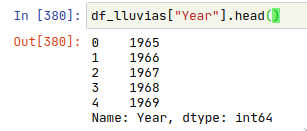
df\_lluvias.head()



El método .head() retorna las primeras 5 filas del data frame, lo cual es útil para obtener una vista previa de los datos.

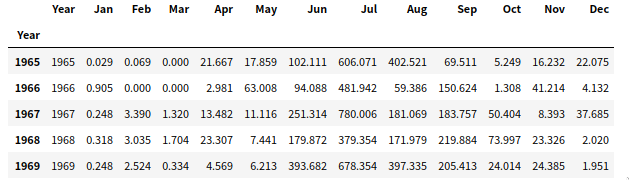
Por conveniencia, es preferible que el índice del data frame sea la columna *Year.* Esto se puede realizar de la siguiente manera:

En primer lugar, se selecciona la columna *Year* con la el método *df\_lluvias["Year"].* Esto retorna un *Serie de Pandas*, tal como vimos anteriormente. Recordar que un objeto Series no es más que una estructura de datos análoga a un *Numpy Array* pero con el agregado de un índice, el cual coincide con el índice del data frame.



Podemos asignar la salida de este método directamente a la propiedad *index* del data frame de la siguiente manera

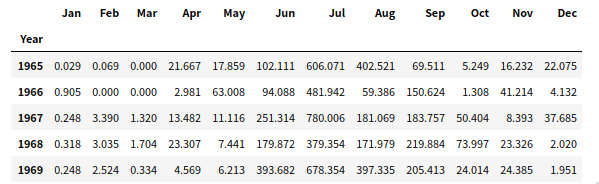
| df\_lluvias.index = df\_lluvias["Year"] df\_lluvias.head() |
| --- |



Si bien el índice se ha actualizado, se debe quitar la columna sobrante Year. Notar que ahora es columna está como dato y además como índice. El método .drop() de Pandas permite eliminar las filas o columnas que se especifiquen y devuelve un nuevo data frame sin tales filas o columnas. Para actualizar el data frame, es necesario entonces asignarlo a la variable *df\_lluvias*.

df\_lluvias = df\_lluvias.drop("Year", axis="columns")

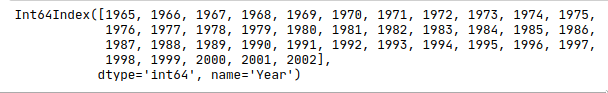
df\_lluvias.head()



Con esto el data frame queda listo para trabajar.

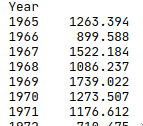
Como se mencionó anteriormente, el método ax.plot() requiere como parámetros las coordenadas de los puntos que la línea debe unir. En el eje horizontal se graficará el año y en el eje vertical las precipitaciones acumuladas en cada año. Por lo tanto, definimos las variables

| x = df\_lluvias.index print(x) |
| --- |



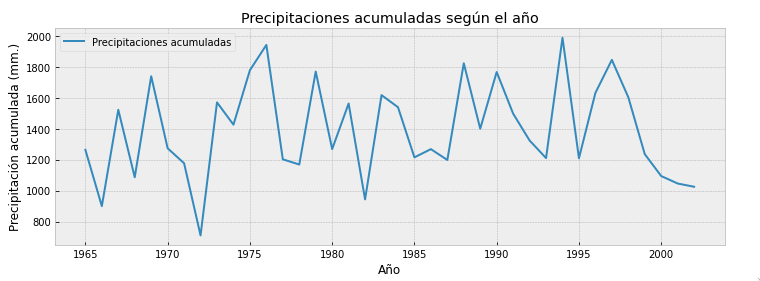
y calculamos las precipitaciones acumuladas en cada año con el método .sum()

| y = df\_lluvias.sum(axis="columns") print(y) |
| --- |



Pasando x e y al método ax.plot() y añadiendo título y etiquetas:

| fig, ax = plt.subplots(figsize=(12, 4))  ax.plot(x, y, label='Precipitaciones acumuladas') ax.set\_xlabel('Año') # Etiqueta eje x ax.set\_ylabel('Precipitación acumulada (mm.)') # Etiqueta eje y ax.set\_title('Precipitaciones acumuladas según el año') # Título ax.legend() # Inserto la leyenda |
| --- |



**Observaciones:**

* El parámetro *figsize* en la llamada a *plt.subplots()* permite cambiar el tamaño del gráfico. En este caso se establece un tamaño de 12 unidades en el eje x y 4 unidades en el eje y. Esta relación de aspecto resulta más adecuada para series de tiempo.
* Notar el parámetro *label* en el método ax.plot(). Este es el texto que se insertará en la leyenda.
* El método ax.legend() automáticamente inserta la leyenda en alguna esquina que no tape la serie de tiempo. Se puede modificar este comportamiento para que se inserte en otra esquina mediante el parámetro *loc. Para v*er las opciones de este y otros parámetros, se puede abrir una nueva celda y ejecutar 'ax.legend?'. Esto abrirá una pequeña pestaña con la documentación oficial de dicha función, la cual contiene una pequeña descripción, los parámetros de entrada, los valores que admite para cada uno de ellos y los objetos que retorna. Si en algún momento no se recuerda qué parámetros recibe una determinada función, añadir un signo de pregunta al final de la misma y ejecutar la celda.

En el gráfico anterior se observa que existe un margen vacío al inicio y al final de la serie de tiempo, para recortaqrlo se puede añadir la siguiente línea:

| ax.set\_xlim(df\_lluvias.index[0], df\_lluvias.index[-1]) |
| --- |

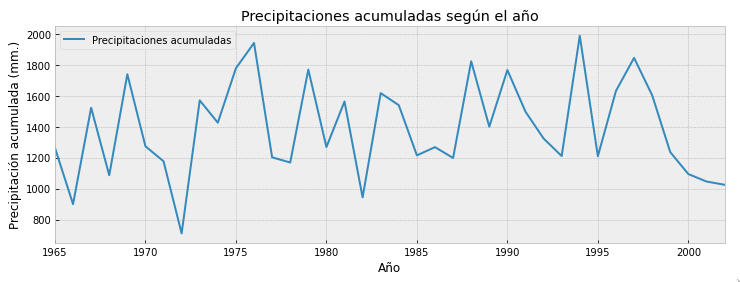
Recordemos a continuación cómo accedemos a los elementos de una estructura en Python:

*df\_lluvias.index[0]* devuelve el primer elemento del índice, es decir, 1965.

*df\_lluvias.index[-1]* devuelve el último elemento del índice, es decir, 2002.

Por otra parte, el método *ax.set\_xlim()* permite establecer los límites del gráfico en el eje *x*. De esta manera forzamos a matplotlib a que sólo grafique desde 1965 hasta 2002.

El gráfico resultante queda entonces:



Por su parte, también es posible establecer límites en el eje y con el método *ax.set\_ylim()*.

## Exportando los gráficos

Matplotlib permite guardar las figuras generadas con una única línea de código:

| fig.savefig("precipitaciones\_año.pdf") |
| --- |

Este método recibe como parámetro el nombre del archivo junto con su extensión, la cual determina el formato en el que se va a exportar el archivo. Los formatos más usados que matplotlib soporta son jpeg, jpg, png, pdf, svg.

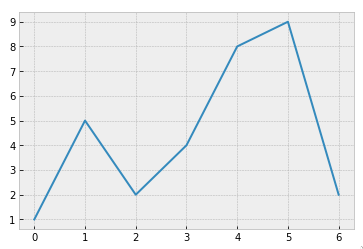
Una vez ejecutado este código, el gráfico quedará guardado con extensión PDF en misma carpeta donde está guardado el notebook.

# Gráficos comunes

## Gráficos de líneas

Los gráficos de líneas son adecuados para visualizar datos con secuencialidad temporal, como las series de tiempo. Como en el ejemplo anterior, para generar un gráfico de líneas basta con pasar como parámetro las coordenadas de los puntos que la línea debe unir. Estos pueden ser listas de Python o Numpy arrays indistintamente. Si se provee un único arreglo a la función plot(), Matplotlib asume que éste refiere a las coordenadas en el eje *y. Por ejemplo, en el gráfico:*

| fig, ax = plt.subplots() ax.plot([1, 5, 2, 4, 8, 9, 2]) |
| --- |

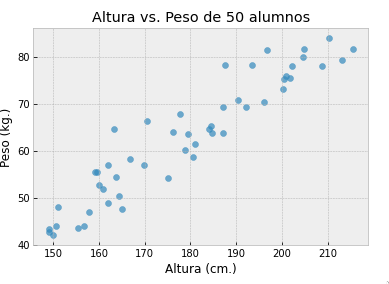
**

matplotlib implícitamente asume como coordenadas en el eje x al arreglo [0, 1, 2, 3, 4, 5, 6].

## Gráficos de puntos

Los gráficos de puntos son especialmente útiles cuando se tiene una gran cantidad datos numéricos emparejados. Permiten ver a simple vista la existencia (o inexistencia) de relación entre las variables que se grafican y también evidencian la presencia de subgrupos dentro del conjunto de datos que se grafica. Por ejemplo, consideremos que se mide la altura y el peso de 50 alumnos. Intuitivamente se podría pensar que, por lo general, las personas más altas son más pesadas que las personas con menor altura, lo que implicaría que existe una **correlación positiva** entre las variables. En caso de que esta hipótesis sea cierta, dicha relación positiva debería reflejarse en el gráfico como una nube de puntos concentrada en la diagonal ascendente del gráfico.

| pesos = [42.8, 43.3, 42. , 44. , 44.1, 43.5, 48.1, 48.9, 47.7,46.9,50.4,  52.7, 51.8, 54.5, 54.2, 56.9, 55.4, 55.5, 57.1, 58.3, 63.7, 58.8,  64.6, 60.2, 64. , 63.8, 61.4, 66.3, 64.7, 63.9, 69.3, 67.9, 65.2,  70.8, 70.5, 69.3, 75.3, 75.5, 78.2, 78. , 73.2, 78. , 80.1, 78.2,  76. , 81.5, 79.4, 81.8, 81.8, 84.1] alturas = [149. , 149. , 149.9, 156.8, 150.6, 155.4, 151. , 162. , 165.,  157.8, 164.4, 160.1, 160.8, 163.8, 175.2, 162. , 159.5, 159.2,  169.8, 166.7, 179.4, 180.6, 163.3, 178.8, 176.3, 184.8, 181. ,  170.5, 184.1, 187.1, 187.1, 177.7, 184.5, 190.3, 196. , 192.1,  200.4, 201.8, 187.5, 202.1, 200.3, 208.8, 204.6, 193.5, 200.9,  196.8, 213.1, 204.8, 215.5, 210.2]  fig, ax = plt.subplots() ax.scatter(alturas, pesos, alpha=0.7) ax.set\_title('Altura vs. Peso de 50 alumnos') ax.set\_xlabel('Altura (cm.)') ax.set\_ylabel('Peso (kg.)') |
| --- |
|  |



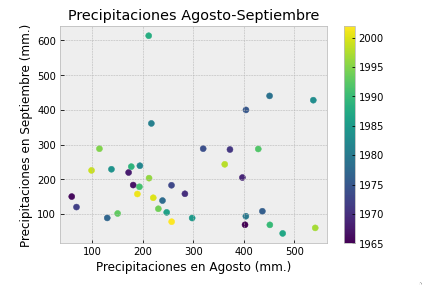
**Observación:**

* No es necesario que el conjunto de datos esté ordenado en forma ascendente para realizar un diagrama de puntos. Lo que sí importa es que los elementos tengan el mismo orden respecto de lo que representan. Por ejemplo, si el primer elemento en la lista de pesos es el peso de Juan, entonces el primer elemento de la lista de alturas debe ser la altura de Juan y no de otra persona. De la misma manera los valores de peso y altura deben estar emparejados para los 49 estudiantes restantes.
* El parámetro *alpha* representa la transparencia de los puntos y debe ser un valor mayor a 0 y menor o igual a 1. A menor alpha, mayor transparencia. Un punto completamente sólido tiene alpha = 1.

Si tenemos una gran cantidad de mediciones, es recomendable establecer cierto grado de transparencia en este tipo de gráficos dado que los puntos suelen taparse unos a otros. Además, al establecer cierta transparencia resulta más claro identificar las regiones con una mayor concentración de puntos.

Ahora, se compararán las mediciones de precipitaciones en dos meses consecutivos a lo largo de los años. Las variables a emparejar podrían ser las precipitaciones en agosto y las precipitaciones en septiembre. Se procede entonces a seleccionar dichas columnas de interés con los métodos *df\_lluvias['Aug']* y *df\_lluvias['Sep'],* las cuales pueden pasarse directamente al método *ax.scatter()*.

| fig, ax = plt.subplots()  mapeo\_colores = ax.scatter(df\_lluvias['Aug'], df\_lluvias['Sep'], c=df\_lluvias.index) fig.colorbar(mapeo\_colores) # Se añade la barra de colores  ax.set\_title('Precipitaciones Agosto-Septiembre') ax.set\_xlabel('Precipitaciones en Agosto (mm.)') ax.set\_ylabel('Precipitaciones en Septiembre (mm.)') |
| --- |



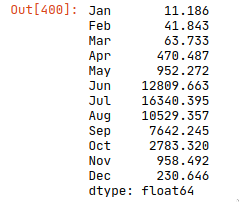
Se observa entonces una nube de puntos mucho más dispersa que la del caso de altura-peso anterior, lo que significa que la relación entre estas variables es más débil.

**Observación:** el parámetro 'c' en la llamada a ax.scatter(), permite asociar un color a cada punto. En este caso, las mediciones más recientes se marcaron con colores claros y las más antiguas con colores oscuros. El método ax.scatter() también recibe un parámetro 's' que permite asociar un determinado tamaño a cada punto de manera que es posible añadir una dimensión más de análisis al gráfico.

## Gráficos de barras

Una forma de comparar variables categóricas es mediante un gráfico de barras. Por ejemplo, si se quisiera visualizar cuáles son los meses con menor o mayor volumen de precipitaciones, podrían acumularse las mediciones según el mes en el que ocurrieron. En primer lugar, se calculan las precipitaciones acumuladas en los distintos meses:

| precipitaciones\_acumuladas = df\_lluvias.sum() print(precipitaciones\_acumuladas) |
| --- |



En el eje horizontal se ubican los meses, y en el eje vertical las precipitaciones acumuladas

| fig, ax = plt.subplots(figsize=(8,4)) precipitaciones\_acumuladas = df\_lluvias.sum() ax.bar(df\_lluvias.columns, precipitaciones\_acumuladas) ax.set\_title('Precipitaciones acumuladas desde 1965 a 2002, según el mes') ax.set\_ylabel('Precipitación total (mm.)') ax.set\_xlabel('Mes') |
| --- |

## 

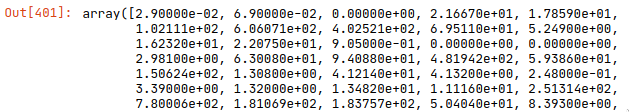
## Histograma

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. De esta forma, permiten obtener un panorama de la distribución del conjunto de datos.

Para la confección del histograma, en primer lugar se debe categorizar el conjunto de datos en diferentes intervalos de valores que no se superpongan, estos son denominados *intervalos de clase*. Posteriormente, se calcula la frecuencia absoluta de ocurrencia de cada uno de los intervalos a partir de las observaciones. Para el caso de las precipitaciones, el mínimo observado durante todo el periodo de registro es 0 mm y el máximo es 780 mm. Por lo tanto, deberemos construir intervalos de clase que abarquen desde 0 hasta 780. Si utilizamos 10 intervalos de clase, entonces cada uno tendrá una amplitud de (780 - 0) / 10 = 78 unidades.

El método para construir el histograma es ax.hist(), que recibe como parámetro un arreglo unidimensional (un conjunto de observaciones). No obstante, nuestro data frame es una estructura bidimensional (por mes y por año). Una forma conveniente de transformar todo el contenido el data frame en un único arreglo es mediante el siguiente método

| df\_lluvias.values.flatten() |
| --- |



Para corroborar las dimensiones del arreglo se puede utilizar el método .shape():

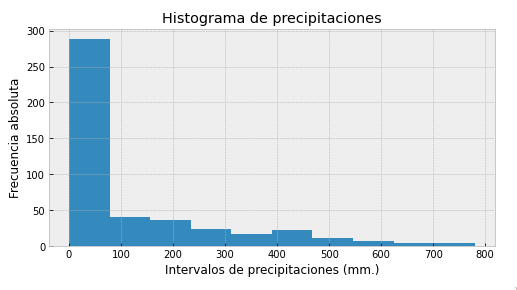
| df\_lluvias.values.flatten().shape() |
| --- |



Es decir, al aplicar el método flatten() sobre los valores del data frame se obtiene un arreglo "aplanado", con 456 elementos.

El gráfico:

| fig, ax = plt.subplots(figsize=(8, 4)) ax.hist(df\_lluvias.values.flatten(), bins=10) ax.set\_title('Histograma de precipitaciones') ax.set\_xlabel('Intervalos de precipitaciones (mm.)') ax.set\_ylabel('Frecuencia absoluta') |
| --- |



La cantidad de intervalos de clase se especifica como parámetro *bins* en la llamada al método ax.hist(). La forma del histograma cambia levemente dependiendo de la cantidad de intervalos de clase que se consideren. Como se observa en la figura, el intervalo [0, 78) es el que ocurre con mayor frecuencia, esto es, la mayoría de precipitaciones tienen valores en ese rango

**Observación:** si bien en el histograma anterior representamos la *frecuencia absoluta* de cada uno de los intervalos, también es válido representar la *frecuencia relativa* o el *porcentaje respecto del total*.