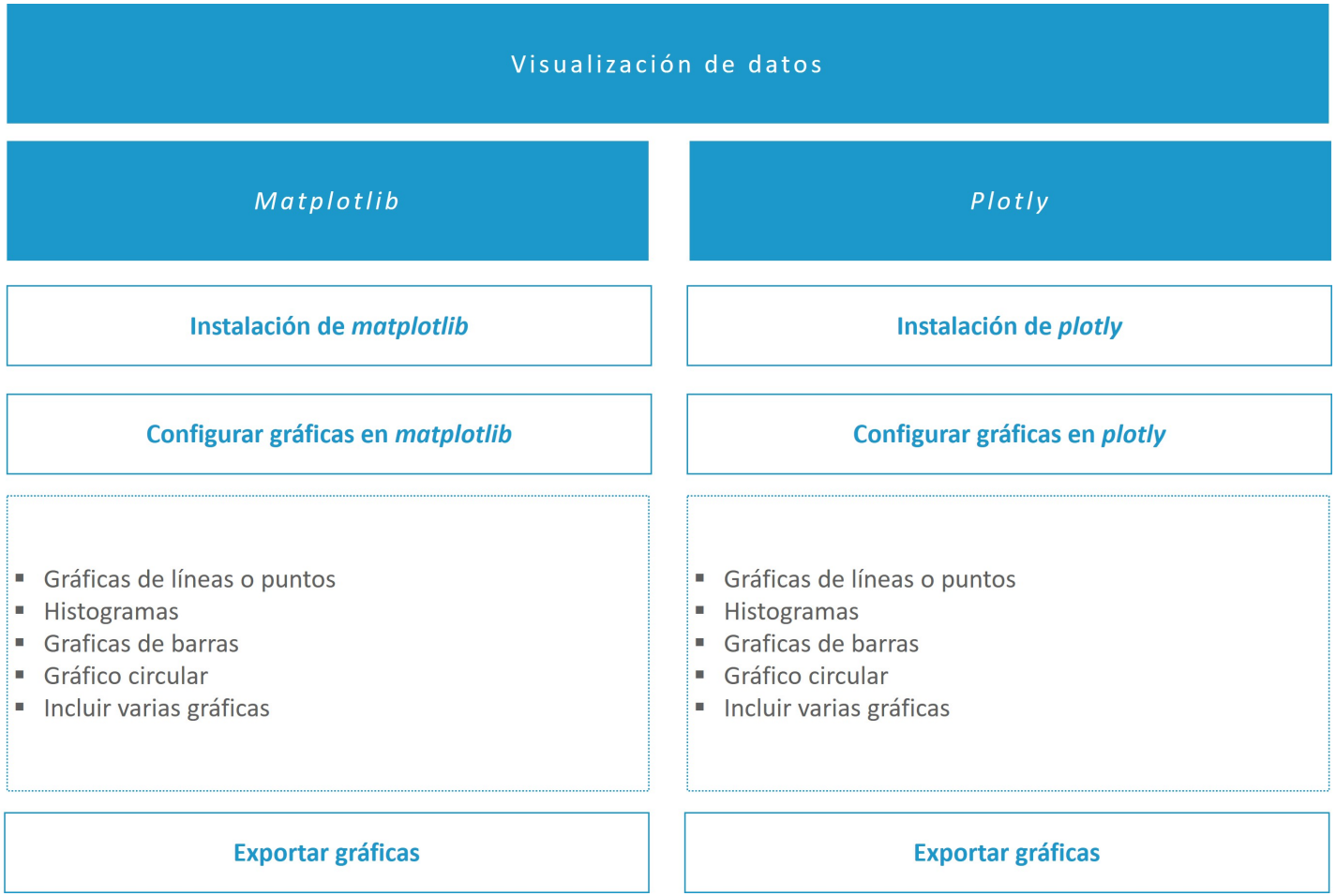
Curso de Programación en Python

Tema 8. Visualización de datos

Índice

Esquema



8.1. Introducción y objetivos

Uno de los aspectos más importantes en el análisis de datos es mostrar a otras personas los resultados que hemos obtenido de nuestro análisis. Por este motivo, el área de la visualización de datos es muy importante. En este tema vamos a ver dos librerías para generar diferentes gráficas: `matplotlib` y `plotly`. La primera nos permite hacer gráficas de forma muy rápida y la segunda nos permite crear gráficas en las que el usuario puede interactuar con ellas.

En este tema se tratarán los siguientes objetivos asociados a la visualización de datos con Python:

- Aprender cómo crear gráficas sencillas usando la librería `matplotlib`.
- Conocer cómo crear gráficas sencillas usando la librería `plotly`.
- Saber crear varias gráficas en una misma imagen con ambas librerías.
- Aprender a exportar las gráficas a otros formatos con ambas librerías.

8.2. Matplotlib

matplotlib es una librería que nos permite crear gráficos de dos dimensiones. Estos gráficos se generan a partir de datos almacenados en listas o *arrays* de numpy. Esta librería es muy sencilla de utilizar y permite hacer gráficos de forma muy rápida.

Instalación de matplotlib

Este módulo, al igual que pasaba con pandas y numpy, tampoco viene instalado por defecto en las distribuciones básicas de Python. Por este motivo, lo primero que debemos hacer es instalar este módulo en nuestro sistema Python. Para ello, usaremos el administrador de paquetes pip:

```
pip install matplotlib
```

Para importar las librerías de matplotlib en nuestro proyecto le asignaremos un alias, como hemos hecho con pandas y numpy, para que sea más sencillo llamarlo en el código. En este caso, el alias que le asignaremos será plt:

```
In [2]: import matplotlib.pyplot as plt
```

Por defecto, cuando se muestren las gráficas hechas con matplotlib, se nos mostrarán en una ventana fuera del *notebook.* Para que las gráficas se muestren dentro del *notebook,* debemos usar la instrucción %matplotlib inline antes de importar la librería.

Configurar gráficas en matplotlib

Realizaremos tanto las gráficas que vamos a ver en este apartado como la configuración de las mismas usando la librería matplotlib.pyplot. Vamos a ver cómo

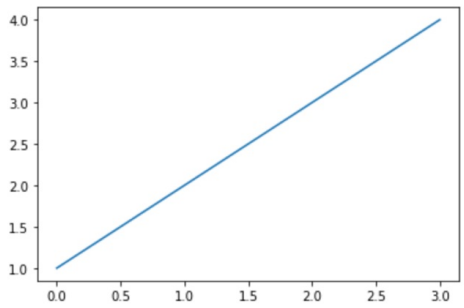
hacer diferentes gráficas y cómo se pueden configurar sus propiedades (color, tamaño, etc.).

Gráficas de líneas o puntos

La gráfica más sencilla que podemos crear con `matplotlib` son las que dibujan una línea o un conjunto de puntos sobre los ejes X e Y. Para ello utilizamos el comando `plot()`. Este comando es muy versátil, como veremos a continuación.

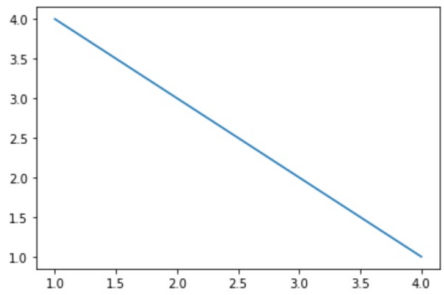
En este primer ejemplo vamos a pintar un conjunto de valores almacenados en un *array* de `numpy`. Como vemos, al pasarle un conjunto de valores el comando los interpreta como valores del eje Y y, por defecto, le asigna unos valores para el eje X. La instrucción `show()` se encarga de mostrar el gráfico.

```
In [3]: %matplotlib inline
        import matplotlib.pyplot as plt

        values = np.array([1, 2, 3, 4])
        plt.plot(values)
        plt.show()
```



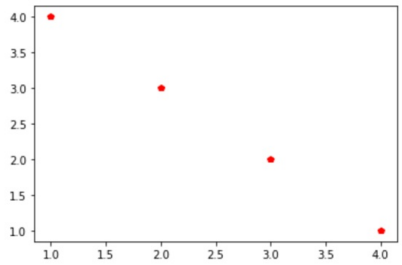
Sin embargo, si le pasamos dos conjuntos de elementos, el primero de ellos lo considerará como los valores que hay en el eje X, y el segundo como los valores en el eje Y.

```
In [4]: %matplotlib inline
import matplotlib.pyplot as plt

values_X = np.array([1, 2, 3, 4])
values_Y = np.array([4, 3, 2, 1])
plt.plot(values_X, values_Y)
plt.show()
```

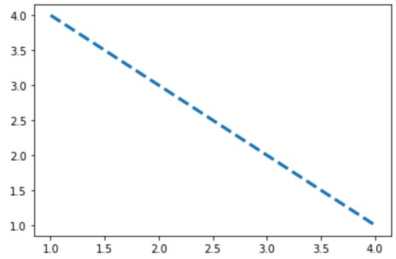


Este comando nos permite tener más argumentos para configurar nuestro gráfico. Todas estas opciones de configuración se pueden consultar en la documentación de la función en matplotlib. Vamos a mostrar unos cuantos ejemplos de posibles configuraciones:

```
In [5]: '''
Argumento format para cambiar el formato del gráfico. En este ejemplo,
son pentágonos rojos
'''
%matplotlib inline
import matplotlib.pyplot as plt

values_X = np.array([1, 2, 3, 4])
values_Y = np.array([4, 3, 2, 1])
plt.plot(values_X, values_Y, 'pr')
plt.show()
```

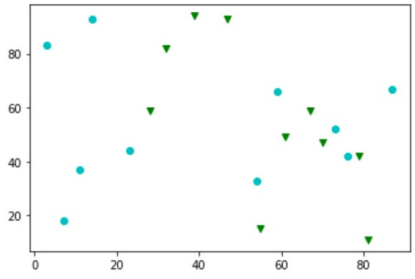


In [6]:
```
'''
Usamos el formato -- para decir que queremos una línea discontinua.
Además, aumentamos el grosor de la línea con linewidth.
'''
%matplotlib inline
import matplotlib.pyplot as plt

values_X = np.array([1, 2, 3, 4])
values_Y = np.array([4, 3, 2, 1])
plt.plot(values_X, values_Y, '--', linewidth=3)
plt.show()
```



Es posible incluir varias gráficas diferentes en una misma imagen. Por ejemplo, en el siguiente ejemplo mostramos los puntos de dos conjuntos diferentes. Cada uno de los conjuntos tiene un símbolo y color diferente.

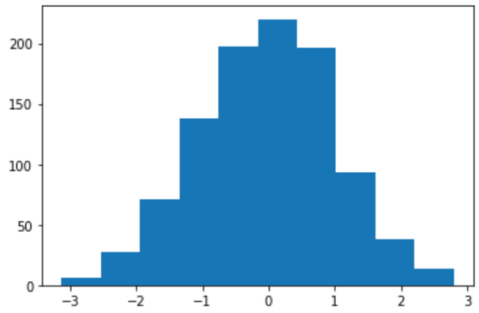
In [7]:
```
%matplotlib inline
import matplotlib.pyplot as plt

values_X1 = np.random.randint(low=1, high=100, size=10)
values_Y1 = np.random.randint(low=1, high=100, size=10)
plt.plot(values_X1, values_Y1, 'co')


values_X2 = np.random.randint(low=1, high=100, size=10)
values_Y2 = np.random.randint(low=1, high=100, size=10)
plt.plot(values_X2, values_Y2, 'vg')
plt.show()
```



Histogramas

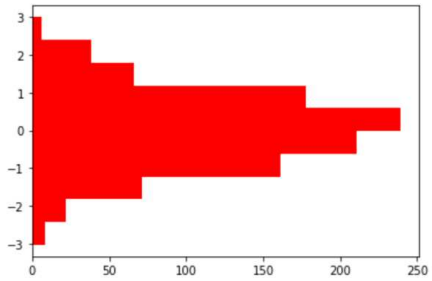
Los histogramas son una de las gráficas más utilizadas para conocer la distribución de valores de un conjunto. matplotlib cuenta con la sentencia hist() que nos permite crear el histograma de un conjunto de valores de forma sencilla. En este caso, vamos a crear un conjunto de puntos aleatorios y mostrar su distribución en un histograma.

```
In [8]: %matplotlib inline
        import matplotlib.pyplot as plt

        valores = np.random.randn(1000)
        plt.hist(valores)
        plt.show()
```

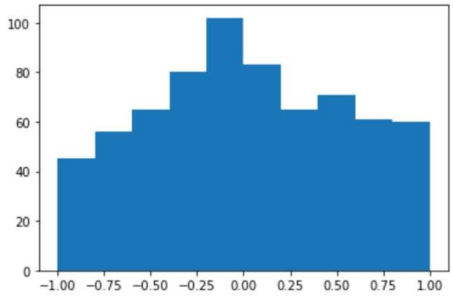


Al igual que pasaba antes, podemos configurar los histogramas de muchas maneras, como son la orientación del histograma, el color, los rangos máximos o mínimos, etc. A continuación, veremos algunas configuraciones.

```
In [9]: '''
        Modificación del color y de la orientación del histograma.
        '''
        %matplotlib inline
        import matplotlib.pyplot as plt

        valores = np.random.randn(1000)
        plt.hist(valores, orientation='horizontal', color='red')
        plt.show()
```

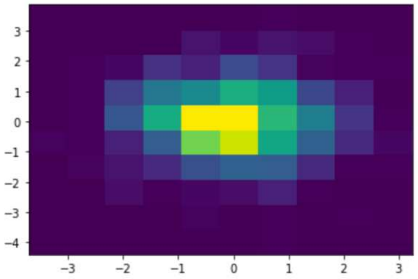


```
In [10]: '''
         Modificación del rango de valores que se va a mostrar (-1, 1)
         '''
         %matplotlib inline
         import matplotlib.pyplot as plt

         valores = np.random.randn(1000)
         plt.hist(valores, range=(-1,1))
         plt.show()
```



Podemos crear histogramas de dos dimensiones con la función `hist2d()`. En este caso, debemos introducir dos conjuntos de valores y se nos mostrará una gráfica similar a un mapa de calor, donde dependiendo del color sabremos si hay más representación de esos valores o menos.

```
In [11]: %matplotlib inline
         import matplotlib.pyplot as plt

         valoresX = np.random.randn(1000)
         valoresY = np.random.randn(1000)
         plt.hist2d(valoresX, valoresY)
         plt.show()
```



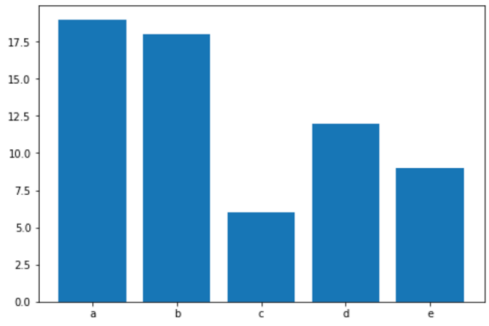
Gráficas de barras

Las gráficas de barras son también muy utilizadas para comparar valores. Este tipo de gráficas necesitan un poco más de configuración. En primer lugar, tenemos que acceder al objeto `figure` que nos permitirá ajustar algunos elementos de la imagen. A

continuación, le definimos el tamaño del gráfico con la función add_axes() pasándole como argumento el tamaño del cuadrado donde se mostrará la gráfica. Por último, preparamos los datos de los valores X, las etiquetas de cada barra y el tamaño que tendrán. Estos valores se enviarán al objeto axes usando la función bar .

Para generar este tipo de gráficos usamos la función bar() . A esta función le pasamos las etiquetas de cada una de las barras y el valor que vamos a asignarle.

```
In [12]: %matplotlib inline
         import matplotlib.pyplot as plt

         fig = plt.figure()
         ax = fig.add_axes([0,0,1,1])

         values_X = ['a', 'b', 'c', 'd', 'e']
         values_Y = np.random.randint(low=1, high=20, size=5)
         ax.bar(values_X,values_Y)
         plt.show()
```



matplotlib también nos permite crear gráficos de barras agrupados, es decir, mostrar dos gráficos de barras en una misma imagen compartiendo los mismos ejes. La configuración de este tipo de gráficos se puede ver en el siguiente ejemplo:

```
In [13]: %matplotlib inline
import numpy as np
import matplotlib.pyplot as plt

# data to plot
n_data = 4
values_1 = (90, 55, 40, 65)
values_2 = (85, 62, 54, 20)

# Configuramos el plot
fig, ax = plt.subplots()
index = np.arange(n_data)
bar_width = 0.35

# Configuramos las barras del primer conjunto de valores
rects1 = plt.bar(index, values_1, bar_width,
color='r',
label='Valores 1')

# Configuramos las barras del segundo conjunto de valores
rects2 = plt.bar(index + bar_width, values_2, bar_width,
color='g',
label='Valores 2')

# Insertamos los títulos y la leyenda del gráfico
plt.xlabel('Etiquetas')
plt.ylabel('Valores')
plt.title('Título del gráfico')
plt.xticks(index + bar_width, ('A', 'B', 'C', 'D'))
plt.legend()

# Pintamos el gráfico y lo mostramos
plt.tight_layout()
plt.show()
```

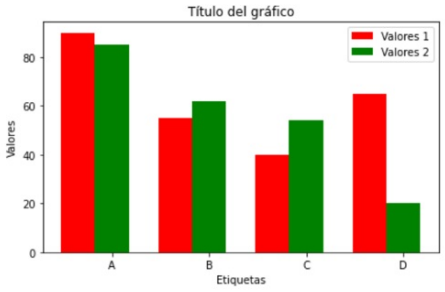


Gráfico circular

Los gráficos circulares o en forma de tarta son muy útiles para ver proporciones o porcentajes de observaciones. Para generar este tipo de gráficos con matplotlib usaremos la función `pie()`. En este caso, pasamos un conjunto de valores cuya suma sea 1:

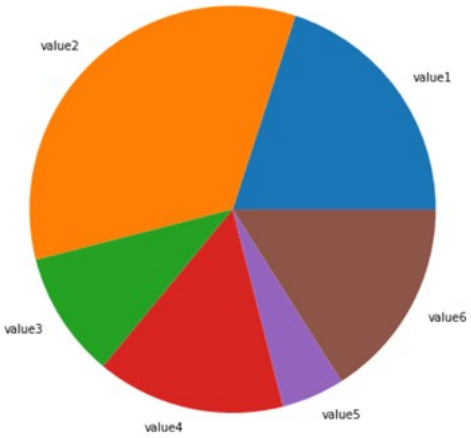
```
In [14]: %matplotlib inline
         import numpy as np
         import matplotlib.pyplot as plt

         values = np.array([0.2, 0.34, 0.10, 0.15, 0.05, 0.16])
         plt.pie(values)
         plt.show()
```

En el caso de que la suma de dichos valores no sea 1, el gráfico dejará la parte correspondiente en blanco, haciendo que el gráfico no sea un círculo completo.

```
In [15]: %matplotlib inline
         import numpy as np
         import matplotlib.pyplot as plt

         values = np.array([0.2, 0.10, 0.15, 0.05, 0.16])
         plt.pie(values)
         plt.show()
```

La función `pie()` también cuenta con muchos argumentos que nos permiten configurar este gráfico. Por ejemplo, podemos añadirle una leyenda etiquetando cada color, el radio del círculo, etc.:

```
In [16]: %matplotlib inline
import numpy as np
import matplotlib.pyplot as plt

values = np.array([0.2, 0.34, 0.10, 0.15, 0.05, 0.16])
labels = ['value1', 'value2', 'value3', 'value4', 'value5', 'value6']
plt.pie(values, labels=labels, radius=2.0)
plt.show()
```

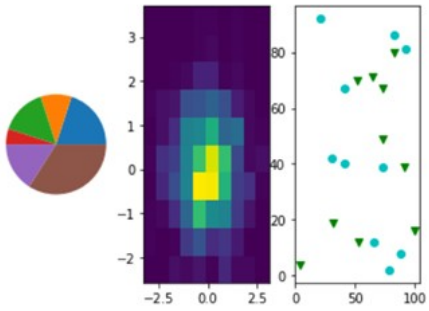


Incluir varias gráficas

Normalmente es muy tedioso generar una gráfica en cada celda para mostrar muchos resultados. Para solucionar esto, matplotlib nos permite pintar más de una gráfica en una imagen. Para ello usaremos la función subplot() . Esta función coloca las gráficas en una tabla donde el número de filas y columnas viene determinado por nosotros. Para ello, antes de pintar una gráfica le indicamos la posición que ocupa usando un número de tres dígitos: el primero para indicar el número de filas, el segundo para indicar el número de columnas y el tercero para indicar la posición del siguiente gráfico.

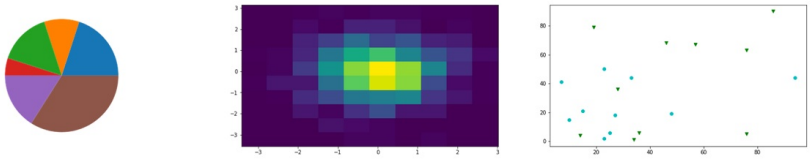
En el siguiente ejemplo vamos a pintar tres gráficas en una misma fila:

```
In [17]: %matplotlib inline
import numpy as np
import matplotlib.pyplot as plt

plt.subplot(131) # Filas 1 - Columnas 3 - Posición del gráfico 1

values = np.array([0.2, 0.10, 0.15, 0.05, 0.16, 0.34])
plt.pie(values)

plt.subplot(132) # Filas 1 - Columnas 3 - Poisición 2

valoresX = np.random.randn(1000)
valoresY = np.random.randn(1000)
plt.hist2d(valoresX, valoresY)

plt.subplot(133)

values_X1 = np.random.randint(low=1, high=100, size=10)
values_Y1 = np.random.randint(low=1, high=100, size=10)
plt.plot(values_X1, values_Y1, 'co')


values_X2 = np.random.randint(low=1, high=100, size=10)
values_Y2 = np.random.randint(low=1, high=100, size=10)
plt.plot(values_X2, values_Y2, 'vg')

plt.show()
```



En estos casos, siempre es aconsejable ajustar el tamaño de la figura donde se dibujan las gráficas, ya que por defecto este tamaño es muy pequeño. Para ello, debemos obtener el objeto Figure que hemos generado y, a continuación, modificar el tamaño (en pulgada).

```
In [18]: %matplotlib inline
import numpy as np
import matplotlib.pyplot as plt

figure = plt.gcf() # Obtenemos la figura donde se pintarán las gráficas
figure.set_size_inches(30, 5) # Configuramos el tamaño en pulgadas

plt.subplot(131) # Filas 1 - Columnas 3 - Posición del gráfico 1

values = np.array([0.2, 0.10, 0.15, 0.05, 0.16, 0.34])
plt.pie(values)

plt.subplot(132) # Filas 1 - Columnas 3 - Poisición 2

valoresX = np.random.randn(1000)
valoresY = np.random.randn(1000)
plt.hist2d(valoresX, valoresY)

plt.subplot(133)

values_X1 = np.random.randint(low=1, high=100, size=10)
values_Y1 = np.random.randint(low=1, high=100, size=10)
plt.plot(values_X1, values_Y1, 'co')


values_X2 = np.random.randint(low=1, high=100, size=10)
values_Y2 = np.random.randint(low=1, high=100, size=10)
plt.plot(values_X2, values_Y2, 'vg')

plt.show()
```

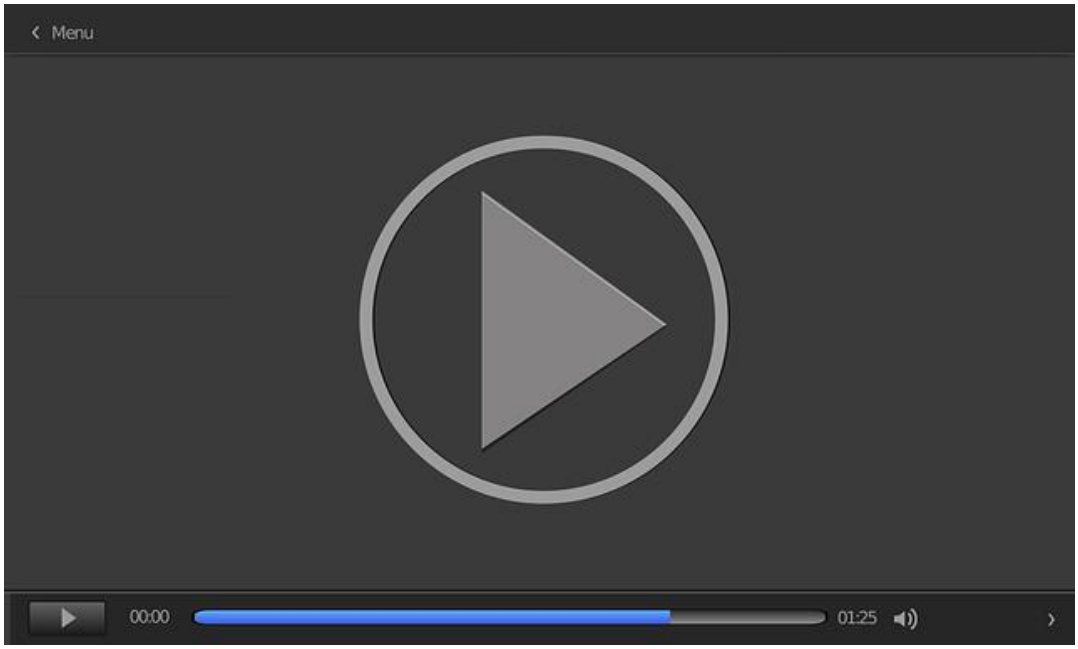


Exportar gráficas

En ocasiones nos puede interesar exportar las gráficas a un fichero para poder compartirlas o importar esas imágenes en un documento externo. Para ello, podemos usar la función `savefig()`, en la que pasamos por parámetro la ruta y nombre del fichero que vamos a guardar. Esta función nos permite exportar las gráficas en formato PDF, PNG, JPG o SVG.

```
In [19]: valoresX = np.random.randn(1000)
valoresY = np.random.randn(1000)
plt.hist2d(valoresX, valoresY)
plt.savefig('grafica.png')
```

A continuación veremos el vídeo titulado *Librería matplotlib*.



Accede al vídeo:

https://unir.cloud.panopto.eu/Panopto/Pages/Embed.aspx?id=ba518eaa-5160-4626-9895-af4e00c3d7af

8.3. Plotly

La segunda librería gráfica que veremos en este tema es plotly. A diferencia de matplotlib, plotly genera gráficos interactivos. Es decir, los gráficos generados por plotly pueden ser manipulados por el usuario haciendo *zoom,* mostrando una zona del gráfico, consultando los valores de cada elemento dibujado, etc. En esta sección aprenderemos a instalarla y usarla en nuestros *notebooks.*

Instalación de plotly

Este módulo tampoco viene instalado por defecto en las distribuciones Python. Por ello, lo primero que vamos a hacer es instalar el módulo plotly con nuestro gestor de paquetes de Python pip:

```
pip install plotly
```

plotly es una librería que utiliza un *framework online* para el que hay que estar registrado. Sin embargo, es posible utilizarlo en un *notebook* de forma *offline.* Para ello, debemos configurar plotly con la función init_notebook_mode(connected=True) del módulo plotly.offline.

```
In [2]: from plotly.offline import init_notebook_mode
        init_notebook_mode(connected=True)
```

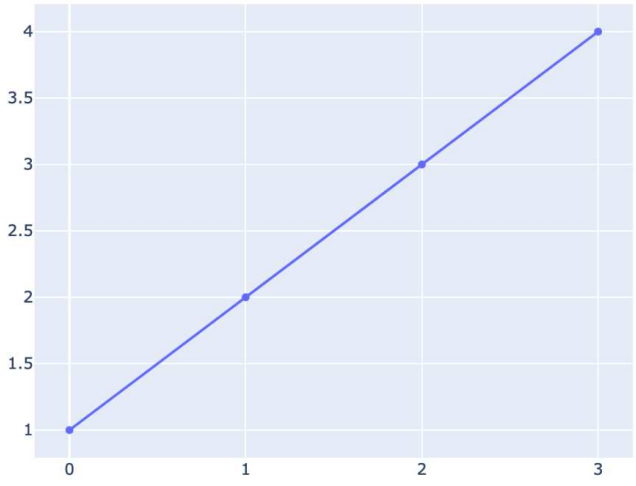
Configuración de gráficas

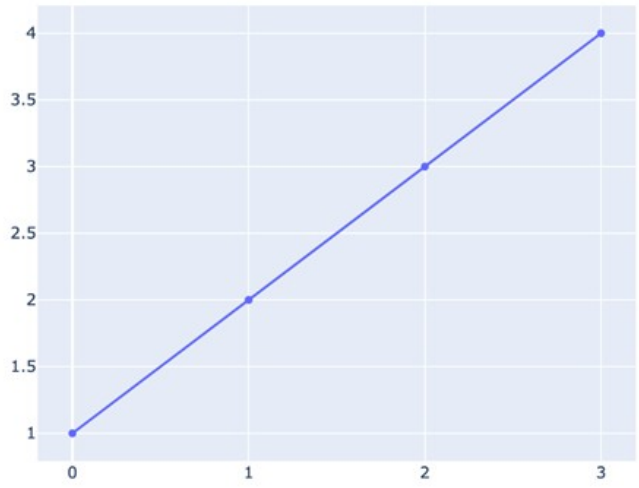
A continuación, vamos a reproducir los mismos ejemplos que hemos visto en matplotlib con esta nueva librería.

Gráficas de líneas o puntos

Para dibujar una gráfica de puntos necesitamos, como mínimo, una lista o un *array* con los valores que tomará cada punto en el eje Y. Para dibujar las gráficas de líneas generaremos un objeto de tipo `Scatter`. La función que se encarga de pintar la gráfica es `iplot()`, del módulo `plotly.offline`. Esta función recibe como mínimo una lista con todos los objetos que se van a pintar. A continuación, vamos a ver el ejemplo más sencillo de una gráfica de líneas:

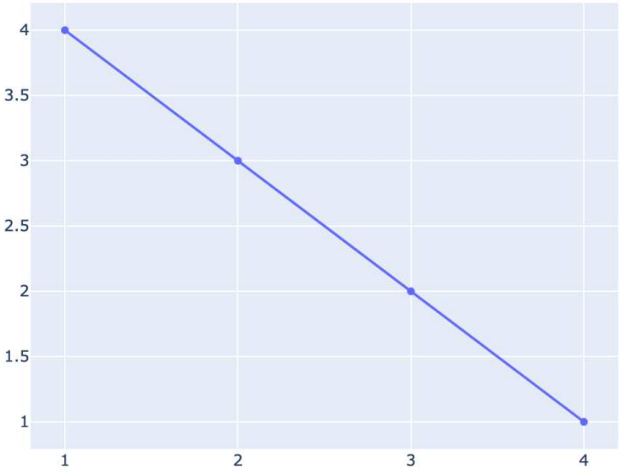
```
In [3]: from plotly.offline import iplot
        import plotly.graph_objects as go

        values = np.array([1, 2, 3, 4])

        data = [go.Scatter(y=values)]
        iplot(data)
```





También es posible pasarle los valores que tienen cada uno de los puntos en el eje X. En este caso debemos incluir esta información en el objeto Scatter que vamos a pintar:

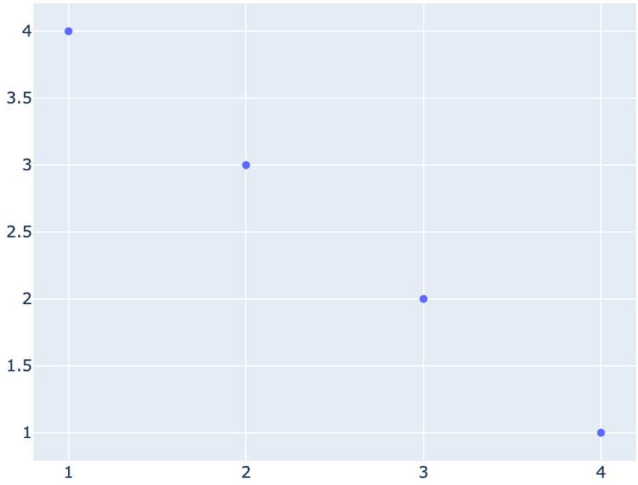
```
In [4]: from plotly.offline import iplot
        import plotly.graph_objects as go

        values_X = np.array([1, 2, 3, 4])
        values_Y = np.array([4, 3, 2, 1])

        data = [go.Scatter(x=values_X, y=values_Y)]
        iplot(data)
```



El objeto Scatter tiene un conjunto de argumentos que nos permite configurar la gráfica que visualizamos. Por ejemplo, si queremos mostrar únicamente los puntos

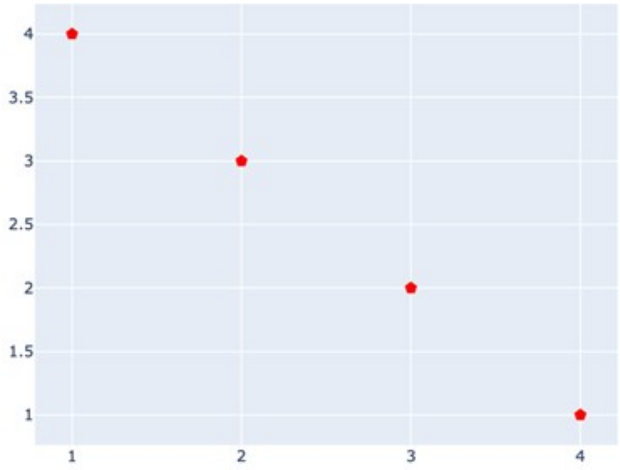
que hemos introducido modificaremos el valor del argumento `mode`, dándole como valor `markers`.

```
In [5]: from plotly.offline import iplot
        import plotly.graph_objects as go

        values_X = np.array([1, 2, 3, 4])
        values_Y = np.array([4, 3, 2, 1])

        data = [go.Scatter(x=values_X, y=values_Y, mode='markers')]
        iplot(data)
```

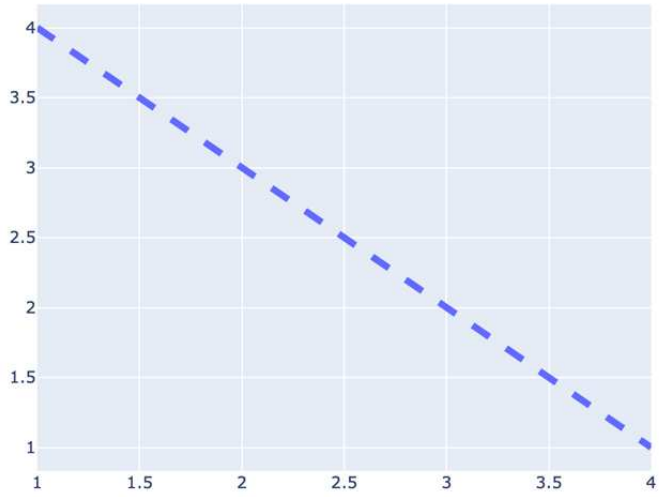


Al igual que pasaba en `matplotlib`, podemos modificar el formato de las líneas o de los puntos para que se vean otro tipo de símbolos.

```
In [6]: from plotly.offline import iplot
        import plotly.graph_objects as go

        values_X = np.array([1, 2, 3, 4])
        values_Y = np.array([4, 3, 2, 1])

        # Ponemos los markers con forma de pentágono, de color rojo y de tamaño 10
        data = [go.Scatter(x=values_X, y=values_Y, mode='markers', marker=dict(color='red', symbol='pentagon', size=10))]
        iplot(data)
```

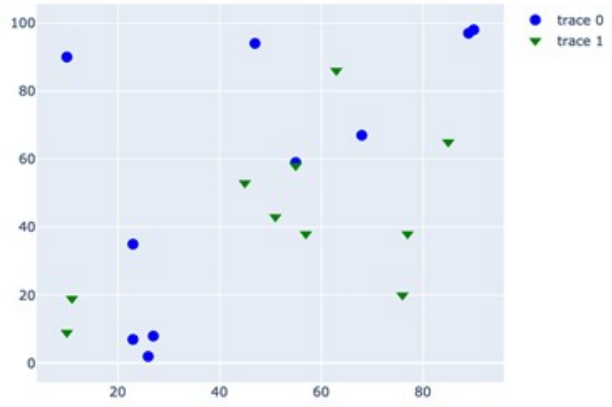



```
In [7]: from plotly.offline import iplot
        import plotly.graph_objects as go

        values_X = np.array([1, 2, 3, 4])
        values_Y = np.array([4, 3, 2, 1])

        # Modificamos la línea de forma discontinua (dash) y anchura 5
        data = [go.Scatter(x=values_X, y=values_Y, mode='lines', line=dict(width=5, dash='dash'))]
        iplot(data)
```



Para incluir dos gráficas diferentes en una misma imagen, debemos incluir cada una de ellas en el *array* de datos que se pasa a la función `iplot` .

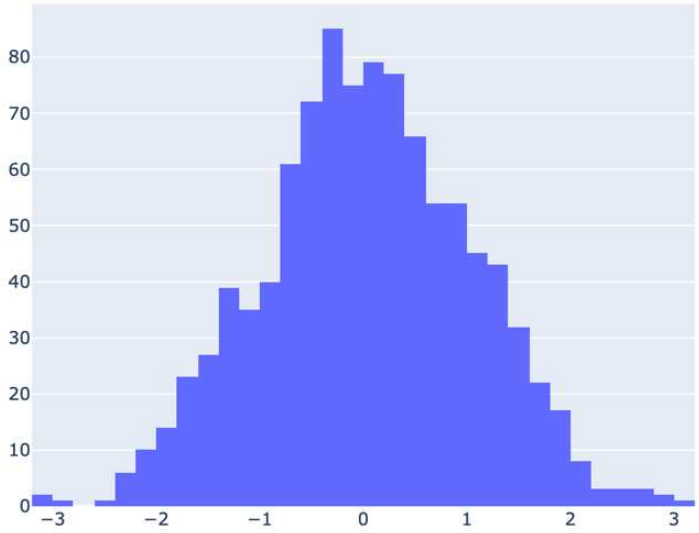
```
In [8]: from plotly.offline import iplot
        import plotly.graph_objects as go

        values_X1 = np.random.randint(low=1, high=100, size=10)
        values_Y1 = np.random.randint(low=1, high=100, size=10)
        scatter_1 = go.Scatter(x=values_X1, y=values_Y1, mode='markers', marker=di
        ct(color='blue', symbol='circle', size=10))

        values_X2 = np.random.randint(low=1, high=100, size=10)
        values_Y2 = np.random.randint(low=1, high=100, size=10)
        scatter_2 = go.Scatter(x=values_X2, y=values_Y2, mode='markers', marker=di
        ct(color='green', symbol='triangle-down', size=10))

        data = [scatter_1, scatter_2]
        iplot(data)
```



Histogramas

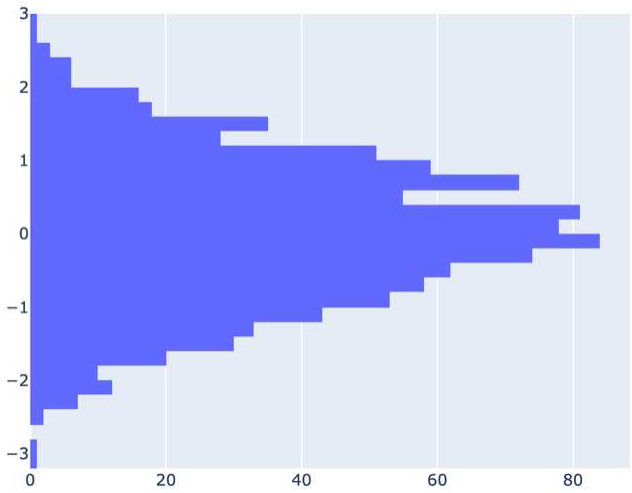
Los histogramas también son un tipo de objeto que podemos encontrar en el módulo plotly.graph_object . Para crear un histograma debemos insertar en el *array* de datos un objeto de tipo Histogram .

```
In [9]: from plotly.offline import iplot
        import plotly.graph_objects as go

        valores = np.random.randn(1000)
        data = [go.Histogram(x=valores)]
        iplot(data)
```



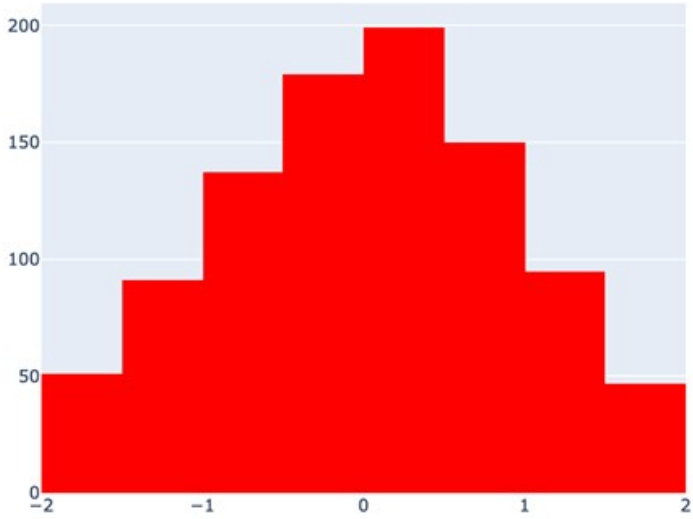
En el caso de que queramos un histograma horizontal, debemos asignar los datos al atributo y del histograma.

```
In [10]: from plotly.offline import iplot
         import plotly.graph_objects as go

         valores = np.random.randn(1000)
         data = [go.Histogram(y=valores)]
         iplot(data)
```



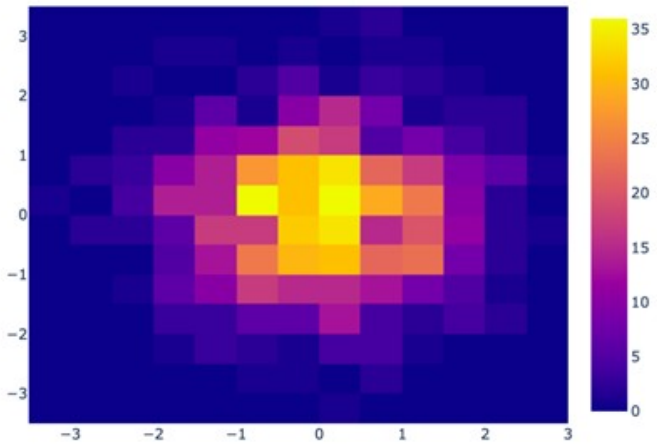
El histograma cuenta con infinidad de atributos que permiten cambiar su estilo como, por ejemplo, su color o el rango de valores X que queremos mostrar.

```
In [11]: from plotly.offline import iplot
         import plotly.graph_objects as go

         valores = np.random.randn(1000)
         data = [go.Histogram(x=valores, marker_color='red',
                              xbins=dict(start=-2, end=2, size=0.5))]
         iplot(data)
```



Al igual que en matplotlib, aquí es posible generar histogramas de dos dimensiones, en los que se nos mostrará la frecuencia con la que aparecen un par de valores.

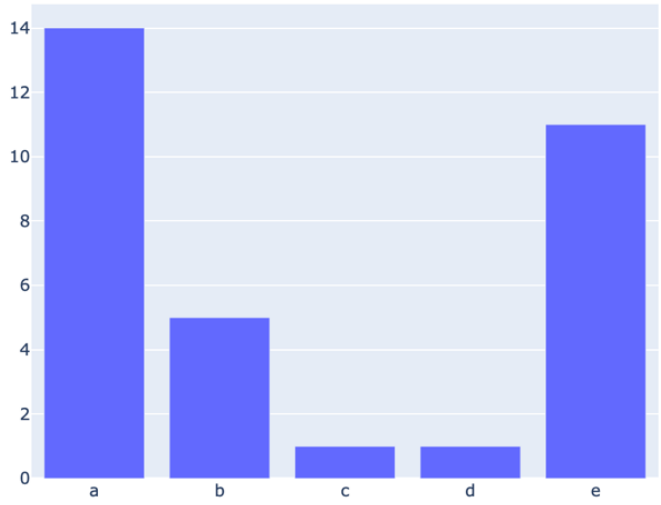
```
In [12]: from plotly.offline import iplot
         import plotly.graph_objects as go

         valoresX = np.random.randn(1000)
         valoresY = np.random.randn(1000)

         data = [go.Histogram2d(x=valoresX, y=valoresY)]
         iplot(data)
```



Gráficas de barras

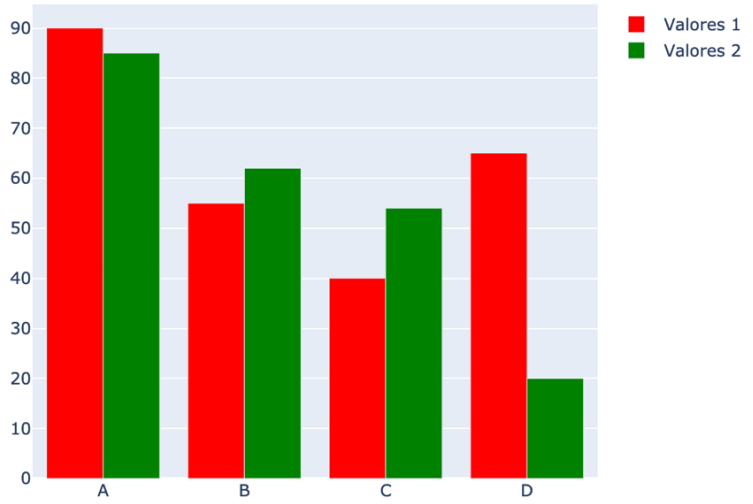
Podemos crear las gráficas de barras con objetos del tipo `Bar`, del módulo plotly.graph_objetcs. Este objeto lo insertaremos dentro del *array* de datos que le pasamos a la función iplot().

```
In [13]: from plotly.offline import iplot
         import plotly.graph_objects as go

         values_X = ['a', 'b', 'c', 'd', 'e']
         values_Y = np.random.randint(low=1, high=20, size=5)

         data = [go.Bar(x=values_X, y=values_Y)]
         iplot(data)
```



Podemos incluir más series de datos en la gráfica de barras. Para ello, solo necesitamos incluir un nuevo objeto Bar en el *array* de datos. Le podemos configurar algunos atributos para darle un nombre a la serie de datos, personalizar los colores, etc.

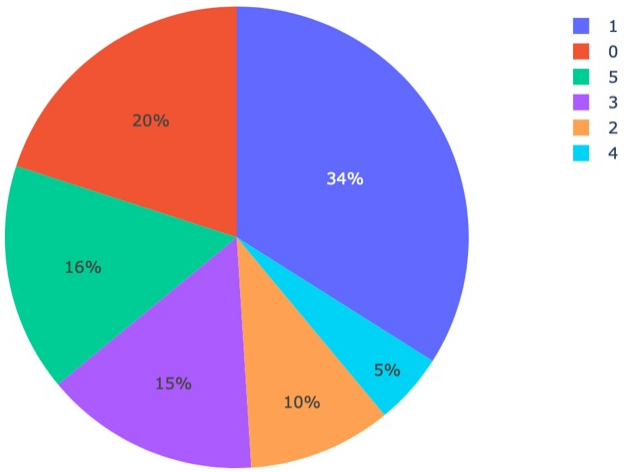
```
In [14]: from plotly.offline import iplot
         import plotly.graph_objects as go

         values_1 = (90, 55, 40, 65)
         values_2 = (85, 62, 54, 20)
         etiquetas = ('A', 'B', 'C', 'D')

         data_1 = go.Bar(name='Valores 1', x=etiquetas,
                         y=values_1, marker_color='red')
         data_2 = go.Bar(name='Valores 2', x=etiquetas,
                         y=values_2, marker_color='green')

         iplot([data_1, data_2])
```



Gráficas circulares

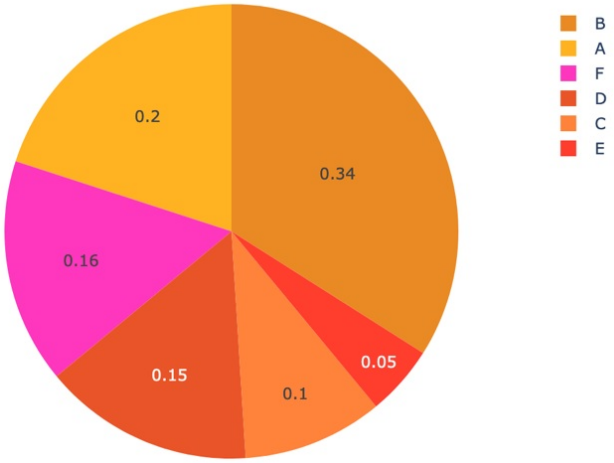
Las gráficas circulares o gráficas de tartas también están incluidas en el módulo plotly.graph_object . En este caso, debemos crear objetos del tipo `Pie` . Como mínimo, estos objetos deben recibir una lista de valores, aunque, a diferencia de matplotlib , no es necesario que sumen 1, ya que el objeto calculará el porcentaje correspondiente a cada porción.

```
In [15]: from plotly.offline import iplot
         import plotly.graph_objects as go

         values = np.array([0.2, 0.34, 0.10, 0.15, 0.05, 0.16])

         data = [go.Pie(values=values)]
         iplot(data)
```



Entre las opciones de personalización de estos gráficos tenemos: definir un nombre a cada serie, mostrar los valores que se han insertado o modificar los colores.

```
In [16]: from plotly.offline import iplot
         import plotly.graph_objects as go

         colors = ['#FFB238', '#E88A33', '#FF8345',
                   '#E85533', '#FF4038', '#FF40BD']
         values = np.array([0.2, 0.34, 0.10, 0.15, 0.05, 0.16])
         etiquetas = ['A', 'B', 'C', 'D', 'E', 'F']

         data = [go.Pie(values=values, labels=etiquetas,
                        textinfo='value', marker=dict(colors=colors))]
         iplot(data)
```

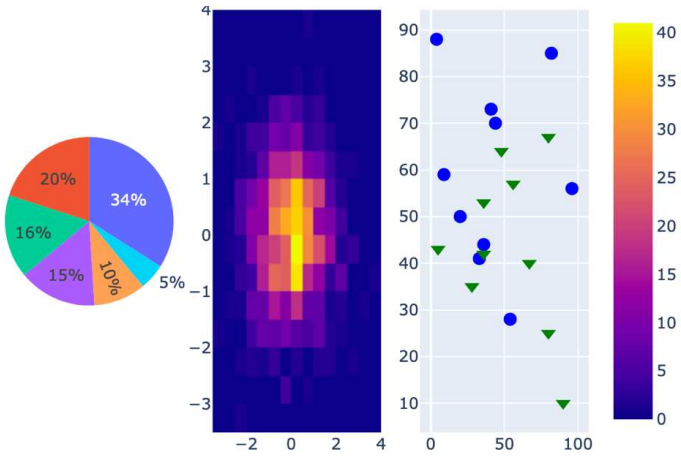


Incluir varias gráficas

En plotly la forma que tenemos de dibujar varias gráficas en una misma celda es usando la función make_subplots() . Esta función recibe dos argumentos: rows que es el número de filas y columns que es el número de columnas. Además, debemos incluir un diccionario en el que indicamos qué tipo de gráfica insertaremos en cada posición.

La función nos devuelve un objeto Figure al que le iremos incluyendo los objetos que hemos creado antes usando la función add_trace , e indicando en qué celda queremos que aparezca la gráfica. A continuación, vamos a generar el ejemplo de las tres gráficas.

```
In [17]: from plotly.subplots import make_subplots
         import plotly.graph_objects as go

         fig = make_subplots(rows=1, cols=3,
                             specs=[[{"type": "domain"}, {"type": "xy"},
                                     {"type": "xy"}]],)

         # Incluimos la gráfica circular en la primera posición
         values = np.array([0.2, 0.34, 0.10, 0.15, 0.05, 0.16])
         fig.add_trace(go.Pie(values=values), row=1, col=1)


         # Incluimos el histograma de 2 dimensiones
         valoresX = np.random.randn(1000)
         valoresY = np.random.randn(1000)
         fig.add_trace(go.Histogram2d(x=valoresX, y=valoresY), row=1, col=2)


         # Incluimos la gráfica de puntos
         values_X1 = np.random.randint(low=1, high=100, size=10)
         values_Y1 = np.random.randint(low=1, high=100, size=10)
         scatter_1 = go.Scatter(x=values_X1, y=values_Y1,
                                mode='markers',
                                marker=dict(color='blue',
                                            symbol='circle', size=10))

         values_X2 = np.random.randint(low=1, high=100, size=10)
         values_Y2 = np.random.randint(low=1, high=100, size=10)
         scatter_2 = go.Scatter(x=values_X2, y=values_Y2,
                                mode='markers',
                                marker=dict(color='green',
                                            symbol='triangle-down', size=10))

         fig.add_trace(scatter_1, row=1, col=3)
         fig.add_trace(scatter_2, row=1, col=3)

         #Eliminamos las leyendas
         fig.update_layout(showlegend=False)
         fig.show()
```

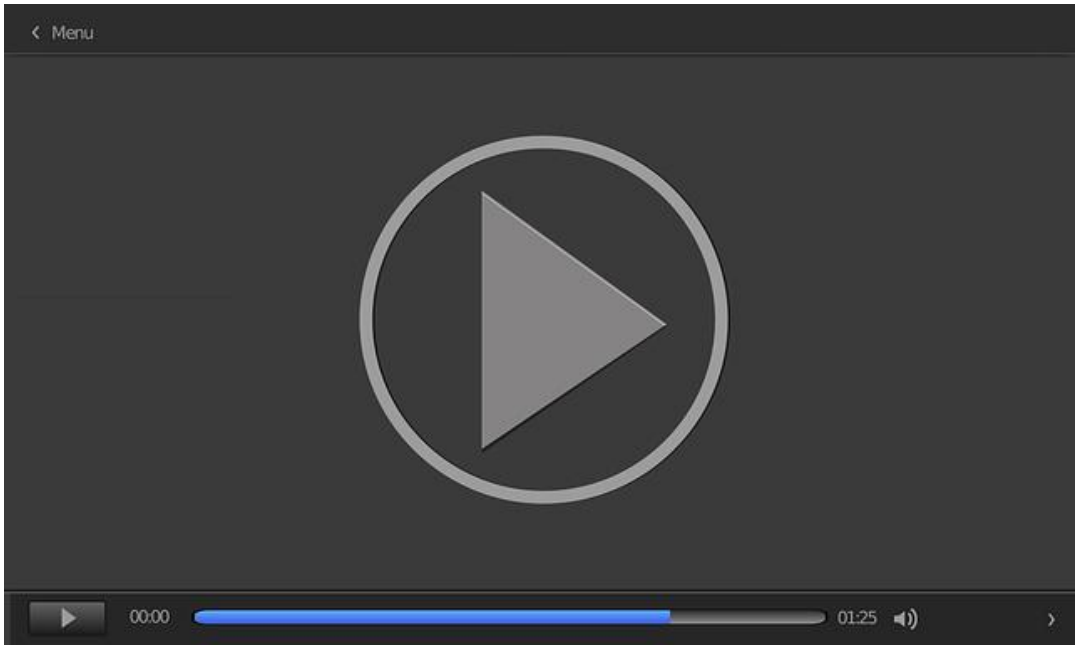


Exportar gráficos

En plotly es mucho más fácil exportar gráficas. Para ello, solo tenemos que ir a los botones que aparecen en la parte superior de la gráfica y pulsar sobre el botón con el icono de una cámara. Esto nos permitirá guardar el gráfico en un fichero .png.

Ideas clave

A continuación veremos el vídeo titulado *Librería plotly*.



Accede al vídeo:

https://unir.cloud.panopto.eu/Panopto/Pages/Embed.aspx?id=bf22803f-3cbd-4cdf-8d25-af4e00c40cc3

Guía matplotlib

The Matplotlib development team. (15 de septiembre de 2020). Tutorials. *Matplotlib* [Página web]. https://matplotlib.org/tutorials/index.html

La librería matplotlib cuenta con muchos tipos de gráficas, algunas de ellas pueden representar modelos matemáticos complejos. Para saber cómo hacer gráficas específicas es necesario ir a la guía de matplotlib .

Ejemplos matplotlib

The Matplotlib development team. (15 de septiembre de 2020). Gallery. *Matplotlib* [Página web]. https://matplotlib.org/gallery/index.html

En ocasiones es necesario basarse en un ejemplo para realizar un gráfico. En este recurso tenemos muchos tipos de gráficas creados con la librería matplotlib .

Guía plotly

Plotly. (2020). Plotly Python open source graphing library [Página web]. https://plotly.com/python/

Al igual que pasa con `matplotlib`, `plotly` cuenta con muchos tipos de gráficas y configuraciones. La guía de desarrollo para Python nos muestra los elementos fundamentales de las gráficas y nos proporciona múltiples ejemplos para aprender a usar esta librería.

Test

1. Los datos que se usan en las gráficas en matplotlib , ¿qué tipo de estructura de datos deben tener?

A. Listas o *arrays* de numpy .

B. Conjuntos.

C. Cadena de texto.

D. Tuplas.

2. ¿Cómo se llama la función que nos permite mostrar las gráficas de matplotlib ?

A. iplot() .

B. graph() .

C. show() .

D. plot() .

3. Para generar un histograma, ¿qué función de matplotlib debemos usar?

A. bar() .

B. hist() .

C. plot() .

D. hist2d() .

4. Para crear un gráfico circular o de tarta los datos deben:

A. Sumar el 100 %.

B. Sumar 100 para que la tarta esté completa.

C. Sumar entre 0 y 1, estando la tarta completa sólo cuando sumen 1.

D. Ser números enteros.

5. ¿Qué hace la función savefig() en matplotlib ?

A. Incluir una gráfica en una imagen que ya contiene otras gráficas.

B. Exportar la imagen de la gráfica en un fichero .pdf, .png, .svg o .jpg.

C. Almacenar los metadatos de la figura en un fichero .json.

D. Guardar la figura en una variable para usarla más adelante.

6. En plotly , cada una de las gráficas se representa con un objeto del módulo:

A. plot .

B. figure .

C. offline .

D. graphobject .

7. ¿Qué atributo del objeto Scatter debemos modificar para que la gráfica sea de puntos?

A. mode=`markers` .

B. symbol=`point` .

C. type=`markers` .

D. Estas gráficas no se hacen con el objeto Scatter .

8. ¿Qué atributo nos permite modificar el rango de valores que se va a mostrar en un histograma?

A. range .

B. start - ends .

C. xbins .

D. values .

9. ¿Es necesario que los valores de una gráfica `Pie` estén normalizados, es decir, sumen 1?

A. Sí, porque de lo contrario no se mostrará la gráfica correcta.

B. Sí. En caso contrario Python nos mostrará un error.

C. No, `Pie` calcula la representación de cada dato en el gráfico de forma automática.

D. No, los valores que se pasan a `Pie` son de tipo cadena de caracteres y no se pueden sumar.

10. Para generar una imagen con varias gráficas, ¿qué función debemos utilizar?

A. addgraph() .

B. makesubplots() .

C. includesubplot() .

D. makefigure() .