Inroducción a Matplotlib

¿Qué es Matplotlib?

- Matplotlib es una de las bibliotecas más utilizadas en Python para la visualización de datos.
- Es altamente flexible y permite crear una amplia variedad de gráficos, desde simples líneas hasta visualizaciones en 3D.
- Se inspiras en MATLAB y proporciona una API similar.
- Es compatible con otras bibliotecas como NumPy, Pandas y Seaborn.
- Permite la personalización avanzada de gráficos.

Instalación y configuración.

- Si aún no tienes Matplotlib instalado, puedes instalarlo con el siguiente comando en la terminal o en una celda de Jupyter Notebook.
- pip install matplotlib --> Terminal
- %pip install matplotlib --> Celda de Jupyter Notebook.
- pip install --upgrade matplotlib --> Si ya esta instalado y hay que actualizarlo.

In [1]:
```
%pip install matplotlib
```

```
Requirement already satisfied: matplotlib in c:\python\python311\lib\site-packages
(3.6.3)
Requirement already satisfied: contourpy>=1.0.1 in c:\python\python311\lib\site-pa
ckages (from matplotlib) (1.0.7)
Requirement already satisfied: cycler>=0.10 in c:\python\python311\lib\site-packag
es (from matplotlib) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\python\python311\lib\site-p
ackages (from matplotlib) (4.38.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\python\python311\lib\site-p
ackages (from matplotlib) (1.4.4)
Requirement already satisfied: numpy>=1.19 in c:\python\python311\lib\site-package
s (from matplotlib) (1.26.3)
Requirement already satisfied: packaging>=20.0 in c:\users\lalo\appdata\roaming\py
thon\python311\site-packages (from matplotlib) (21.3)
Requirement already satisfied: pillow>=6.2.0 in c:\python\python311\lib\site-packa
ges (from matplotlib) (11.1.0)
Requirement already satisfied: pyparsing>=2.2.1 in c:\users\lalo\appdata\roaming\p
ython\python311\site-packages (from matplotlib) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\lalo\appdata\roami
ng\python\python311\site-packages (from matplotlib) (2.8.2)
Requirement already satisfied: six>=1.5 in c:\users\lalo\appdata\roaming\python\py
thon311\site-packages (from python-dateutil>=2.7->matplotlib) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

Importación de Matplotlib y su estructura básica.

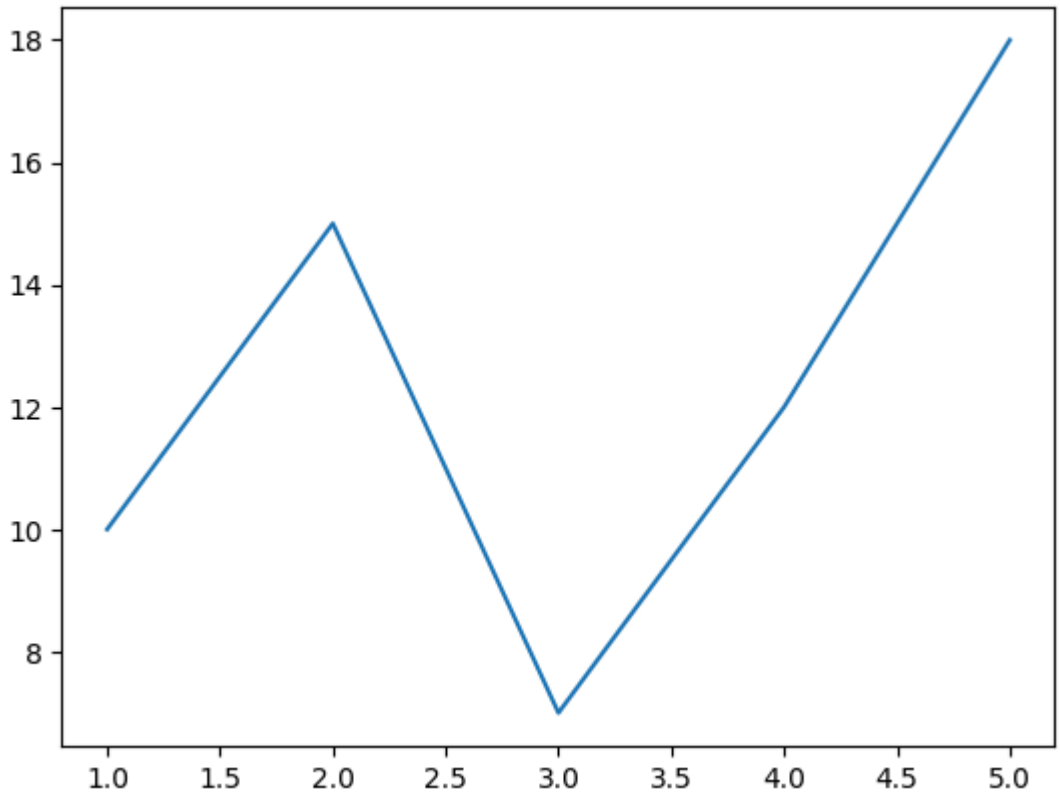
- Para comenzar a utilizar Matplotlib, lo primero es importar la biblioteca.
- Generalmente, se usa la siguiente convención.

In [1]:
```
import matplotlib.pyplot as plt
```

Creando un gráficio básico.

- Vamos a crear un gráfico de líneas sencillo.

In [7]:
```
x = [1, 2, 3, 4, 5]
y = [10, 15, 7, 12, 18]

# Crear el gráfico
plt.plot(x, y)
# Mostrar el gráfico.
plt.show()
```

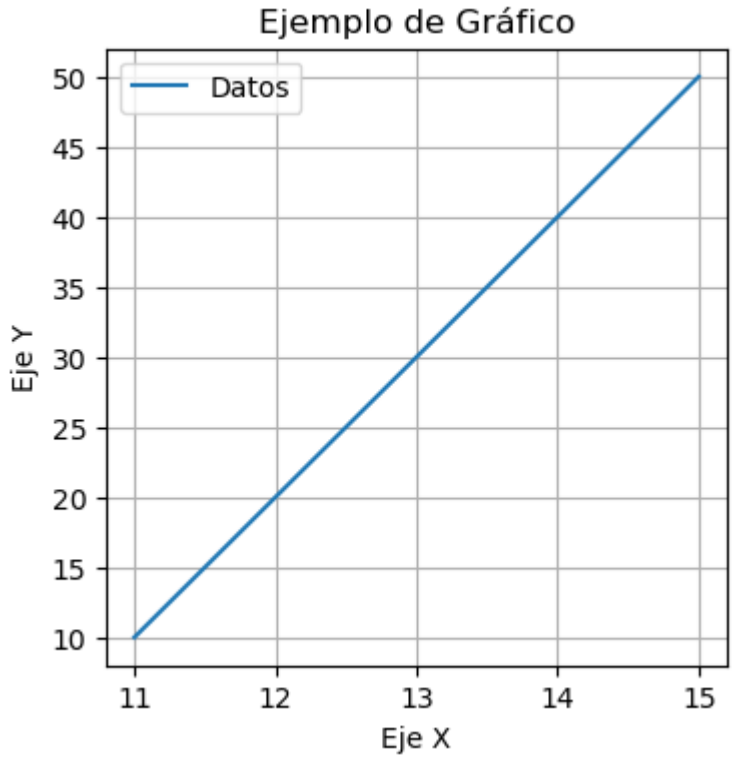


Componentes básicos de una visualización en Matplotlib

1. Figure (fig): Es el contenedor principal del gráfico.
2. Axes (ax): Representa los ejes entro de la figura, donde se dibujan los gráficos.
3. Axis (x,y): Define los ejes individuales dentro de los Axes.
4. Labels: Son las etiquetas de los ejes y el título del gráfico.
5. Legend: Muestra la información de cada serie de datos.
6. Grid: Permite agregar líneas de referencia en el fondo.

In [2]:
```
x = [11, 12, 13, 14, 15]
y = [10, 20, 30, 40, 50]

plt.figure(figsize=(4,4))
plt.plot(x, y, label="Datos")
plt.xlabel("Eje X")
plt.ylabel("Eje Y")
plt.title("Ejemplo de Gráfico")
plt.legend()
```

```
plt.grid(True)
plt.show()
```



Vamos a crear un gráfico y le vamos a cambiar el tipo y el color de linea

In [3]:
```
x = [1, 10, 2, 20, 15]
y = [10, 20, 30, 40, 50]

plt.figure(figsize=(4,4))
plt.plot(x, y, label="Datos", color="red", linestyle="--", marker="o")
plt.xlabel("Eje X")
plt.ylabel("Eje Y")
plt.title("Ejemplo de Gráfico")
plt.legend()
plt.grid(True)
plt.show()
```

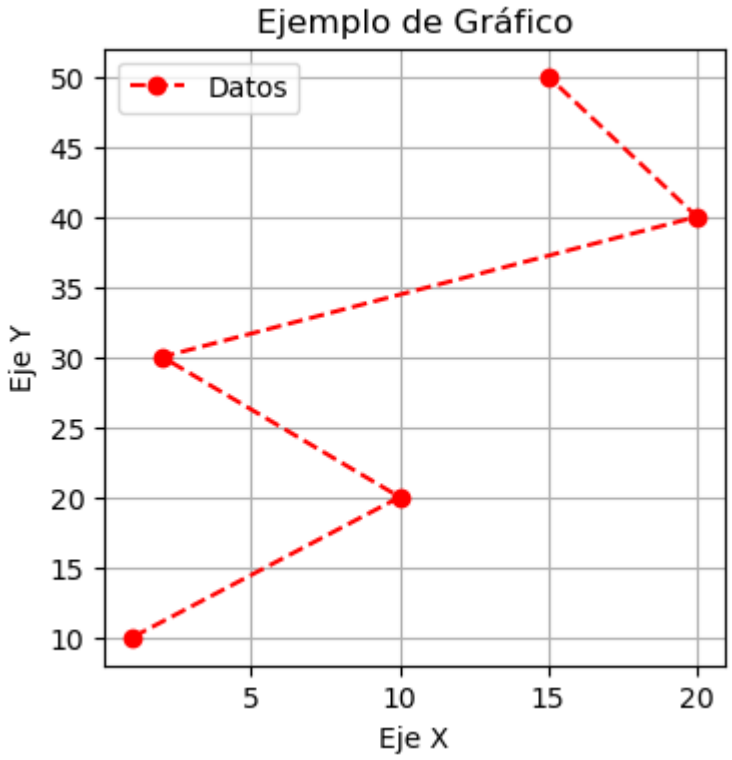


Gráfico de linea con multiples series de datos.

- Podemos representar varias líneas en un mismo gráfico.

In [4]:
```
x = [1,2,3,4,5]
y1 = [10,15,7,12,18]
y2 = [5,8,10,6,12]
plt.figure(figsize=(4,4))
plt.plot(x, y1, label='Serie 1', color='blue', linestyle='-', marker='o')
plt.plot(x, y1, label='Serie 2', color='red', linestyle='--', marker='s')

plt.xlabel("Eje X")
plt.ylabel("Eje Y")
plt.title("Gráfico con múltiples series de datos")
plt.legend()
plt.grid(True)
plt.show()
```

Personalización de un gráfico de lineas.

- Matplotlib permite personalizar completamente los gráficos con colores, estilos y más.

In [5]:
```
x = [1,2,3,4,5]
y = [10,15,7,12,18]
plt.figure(figsize=(4,4))
plt.plot(x, y, color='blue', linewidth=2, linestyle='-.', marker='D', markersize=8,
plt.xlabel("Eje X", fontsize=10, color='darkred')
plt.ylabel("Eje Y", fontsize=12, color='darkblue')
plt.title("Gráfico Personalizado", fontsize=14, fontweight='bold')
plt.show()
```

Gráfico con diferentes estilos predefinidos.

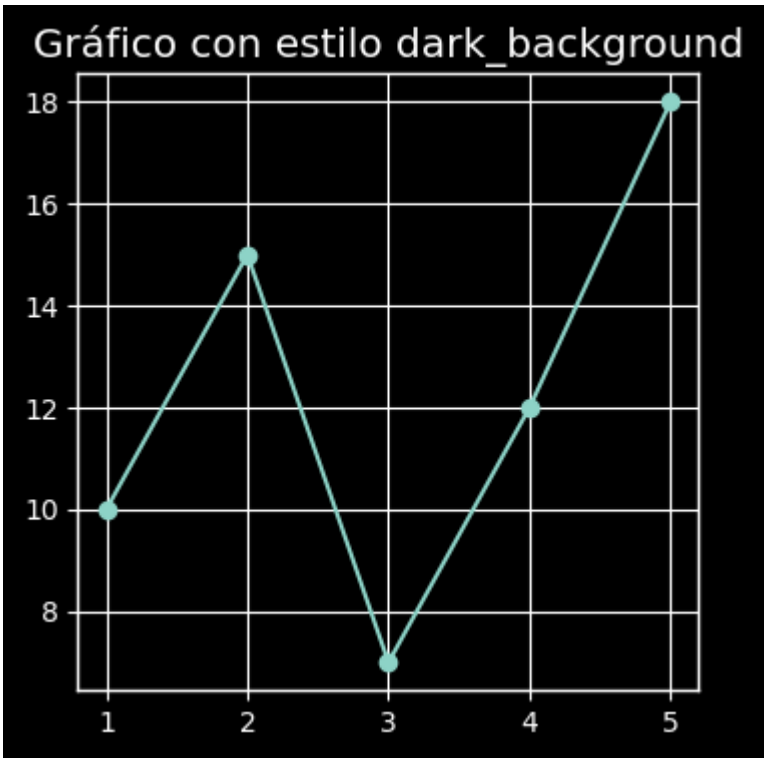
- Matplotlib tiene varios estilos predefinidos que pueden cambiar el aspecto de los gráficos fácilmente.
- Los estilos disponibles se pueden ver con: print(plt.style.aviable)

In [6]:
```
plt.style.use("ggplot")

x = [1,2,3,4,5]
y = [10,15,7,12,18]
plt.figure(figsize=(4,4))
plt.plot(x, y , marker='o', linestyle='-')
plt.title("Gráfico con estilo ggplot")
plt.show()
```

In [15]:
```
print(plt.style.available)
```

```
['Solarize_Light2', '_classic_test_patch', '_mpl-gallery', '_mpl-gallery-nogrid',
'bmh', 'classic', 'dark_background', 'fast', 'fivethirtyeight', 'ggplot', 'graysca
le', 'seaborn-v0_8', 'seaborn-v0_8-bright', 'seaborn-v0_8-colorblind', 'seaborn-v0
_8-dark', 'seaborn-v0_8-dark-palette', 'seaborn-v0_8-darkgrid', 'seaborn-v0_8-dee
p', 'seaborn-v0_8-muted', 'seaborn-v0_8-notebook', 'seaborn-v0_8-paper', 'seaborn-
v0_8-pastel', 'seaborn-v0_8-poster', 'seaborn-v0_8-talk', 'seaborn-v0_8-ticks', 's
eaborn-v0_8-white', 'seaborn-v0_8-whitegrid', 'tableau-colorblind10']
```

In [7]:
```
plt.style.use("dark_background")

x = [1,2,3,4,5]
y = [10,15,7,12,18]
plt.figure(figsize=(4,4))
plt.plot(x, y , marker='o', linestyle='-')
plt.title("Gráfico con estilo dark_background")
plt.show()
```



In [39]:
```
plt.style.use("classic")
```

Gráfico de dispersión simple (scatter)

- A diferencia de plot(),el gráfico de dispersión scatter() se usa para representar puntos individuales.

In [40]:
```
x = [1,2,3,4,5]
y = [10,15,7,12,18]
plt.figure(figsize=(4,4))
plt.scatter(x, y , color='red', marker='s', s=100)
plt.xlabel("Eje X")
plt.ylabel("Eje Y")
plt.title("Gráfica de dispersión")
plt.grid(True)
plt.show()
```

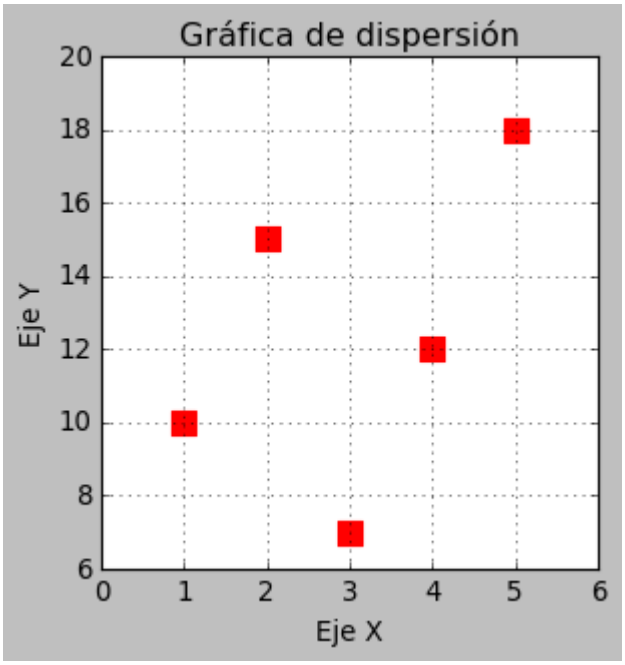
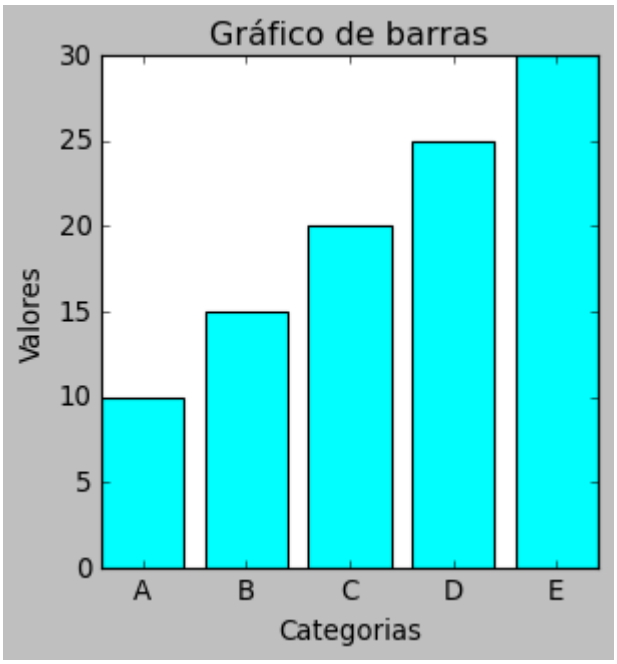


Gráfico de barras básico.

- Los gráficos de barras bar() se usan para representar categorías.

In [41]:
```
categorias = ['A', 'B', 'C', 'D', 'E']
valores = [10, 15, 20, 25, 30]
plt.figure(figsize=(4,4))
plt.bar(categorias, valores, color='cyan')
plt.xlabel('Categorias')
plt.ylabel('Valores')
plt.title('Gráfico de barras')
plt.show()
```



Asi como podemos tener estilos de la gráfica personalizada, también tenemos instrucciones para acceder a mas elementos.

- Para poder ver los colores (color)(c) predefinidos en Matplotlib --> print(plt.colormaps()) --> ('#4CAF50')
- Para ver todos los topos de marcadores (marker) --> print(plt.Line2D.markers)
- Ver tipos de líneas predefinidas (linestyle)(ls) --> print(['-', '--', '-.', ':']) --> (linestyle = 'dotted')
- Para ver el grosor de la fuente (fontweight) --> "normal", "bold", "light", "ultralight", "heavy", "ultrabold"
- Para ver los estilos de fuente --> print(plt.rcParams['font.family'])
- Para ver los estilos de linea (linewidht)(lw) --> (linewidth = '20.5')
- Para el tamaño del marcador (markersize)(ms) --> (marker = 'o', ms = 20)
- para el color del marcador (markeredgecolor)(mec) --> (mec = 'r')
- Para el color del borde de los marcadores (markerfacecolor)(mfc) --> (mfc = 'r')

Referencias para marker

Nombre	Descripcion	Nombre	Descripcion
'o'	Circle	'H'	Hexagon
'*'	Star	'v'	Triangle Down

Nombre	Descripcion	Nombre	Descripcion
'.'	Poit	'^'	Triangle Up
','	Pixel	'<'	Triangle Left
'x'	X	'>'	Triangle Right
'+'	Plus	'1'	Tri Down
'X'	X(field)	'2'	Tri UP
'P'	Plus (field)	'3'	Tri Left
's'	Square	'4'	Tri Right
'D'	Diamond	'l'	Vline
'p'	Pentagon	'_'	Hline

Referencias tipo de Lineas

Line Syntax	Descripción
'-'	Solid Line
':'	Dotted Line
'--'	Dashed line
'-.'	Dashed/dotted line

Referencia de Colores

Color Syntax	Descripción
'r'	Red
'g'	Green
'b'	Blue
'c'	Cyan
'm'	Magenta
'y'	Yellow
'k'	Black
'w'	White

Tipos de gráficos comunes en Matplotlib

- Matplotlib permite crear una gran variedad de gráficos, cada uno adecuado para diferentes tipos de visualización de datos.
- En esta sección, exploraremos algunos de los gráficos más comunes y cómo se crean.

Gráfico de lineas (Line Plot)

- Los gráficos de lineas son ideales para mostrar tendencias a lo largo del tiempo o para visualizar relaciones continuas entre dos variables.

In [42]:
```
x = [1,2,3,4,5]
y = [10,15,20,25,30]
plt.figure(figsize=(4,4))
plt.plot(x, y , label='Datos')
plt.xlabel('Eje X')
plt.ylabel('Eje Y')
plt.title('Gráfico de Lineas')
plt.legend()
plt.show()
```

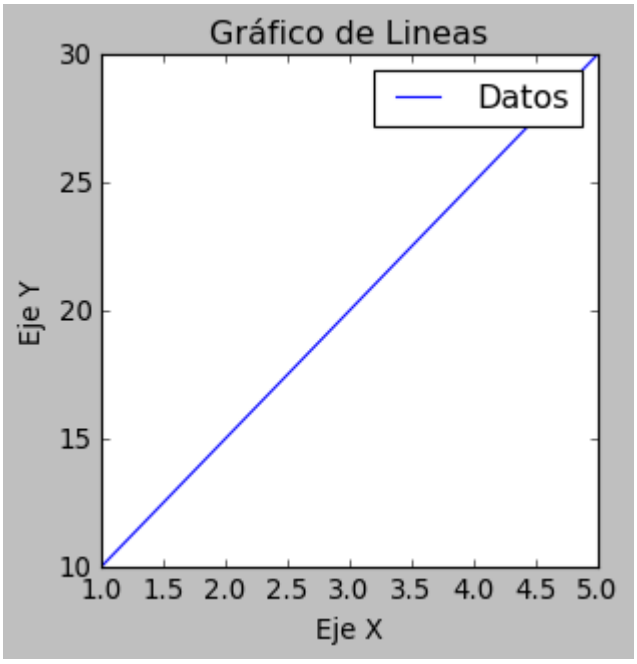


Gráfico de barras (Bar Chart)

- El gráfico de barras es útil para mostrar comparaciones entre diferentes categorías.
- Cada barra representa un valor para una categoría dada.

In [43]:
```
categorias = ['A', 'B', 'C', 'D', 'E']
valores = [10, 15, 20, 25, 30]
plt.figure(figsize=(4,4))
plt.bar(categorias, valores, color='red')
plt.xlabel('Categorias')
plt.ylabel('Valores')
plt.title('Gráfico de barras')
plt.show()
```

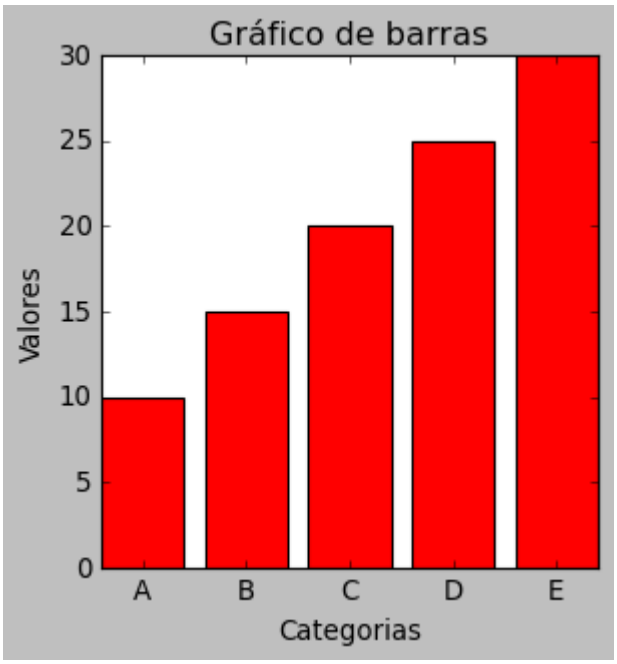
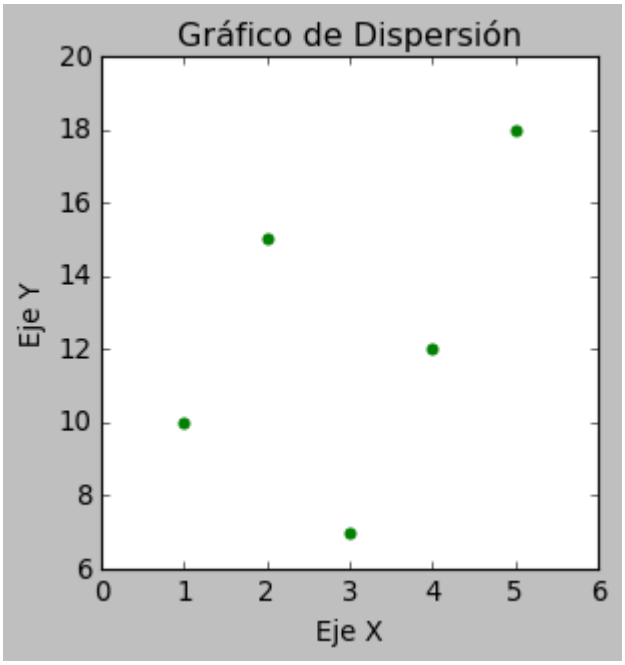


Gráfico de dispersión (Scatter Plot)

- El gráfico de dispersión es ideal para mostrar la relación entre dos variables numéricas.
- Cada punto representa una observación.

In [44]:
```
x = [1, 2, 3, 4, 5]
y = [10, 15, 7, 12, 18]
plt.figure(figsize=(4,4))
plt.scatter(x, y, color='green')
plt.xlabel("Eje X")
plt.ylabel("Eje Y")
plt.title("Gráfico de Dispersión")
plt.show()
```



Histograma (Histogram)

- Los histogramas son útiles para mostrar la distribución de una variable numérica.
- Muestran la frecuencia con la que ocurren los valores dentro de los rangos.

In [45]:
```
datos = [1,2,2,3,3,3,4,4,4,4,5]
plt.figure(figsize=(4,4))
plt.hist(datos, bins=5, color='blue', edgecolor='black')
plt.xlabel("valores")
plt.ylabel('Frecuencia')
plt.title('Histograma')
plt.show()
```

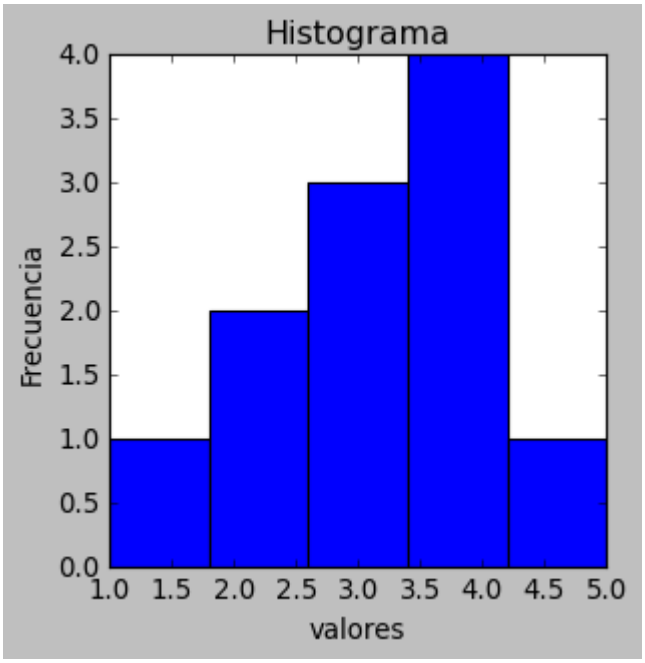
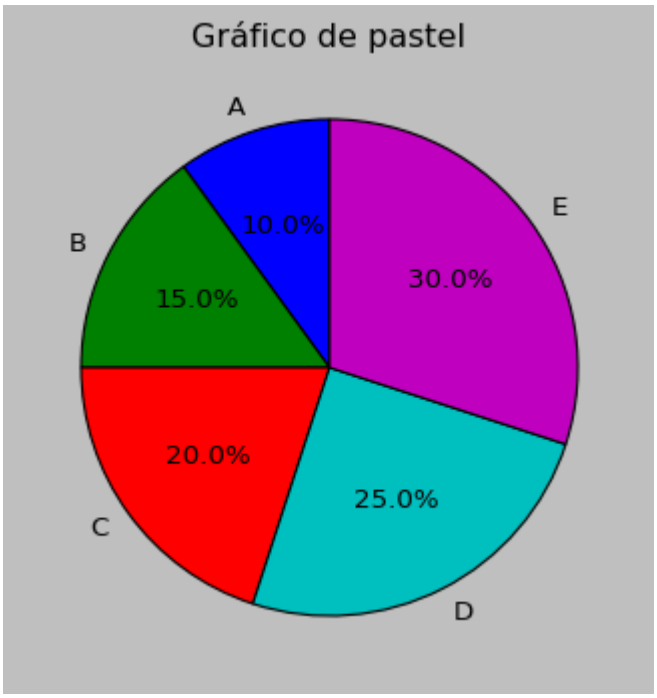


Gráfico de pastel (Pie Chart)

- El gráfico de pastel se utiliza para mostrar la proporción de cada categoría en relación con el total.

In [33]:
```
categorias = ['A', 'B', 'C', 'D', 'E']
valores = [10, 15, 20, 25, 30]

plt.figure(figsize=(5,5))
plt.pie(valores, labels=categorias, autopct='%1.1f%%', startangle=90)
plt.title('Gráfico de pastel')
plt.show()
```



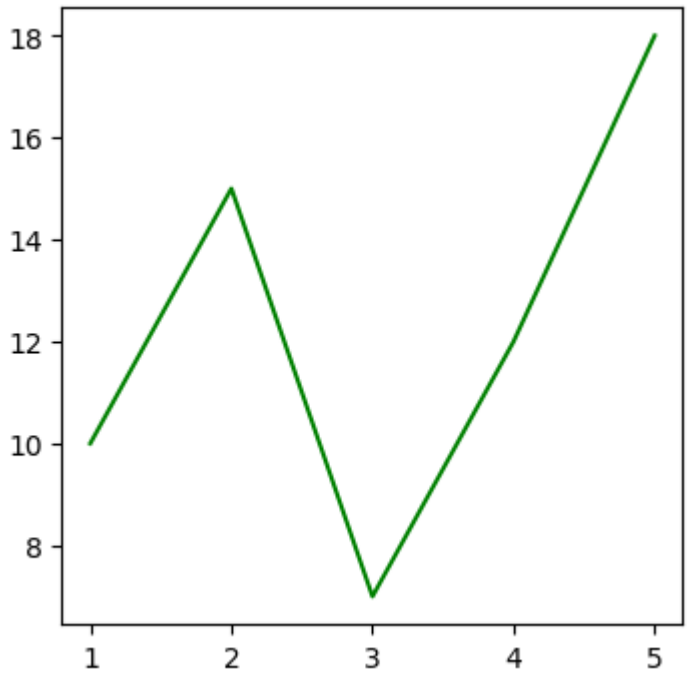
Personalización de Gráficos en Python

- Matplotlib permite una gran cantidad de personalización para adoptar los gráficos a sus necesidades, desde cambiar colores hasta modificar las leyendas y etiquetas.

Cambio de color.

- Se puede cambiar el color de las líneas, barras, puntos, entre otros.
- Los colores se pueden definir por nombre, por código hexadecimal o usando mapas de colores.

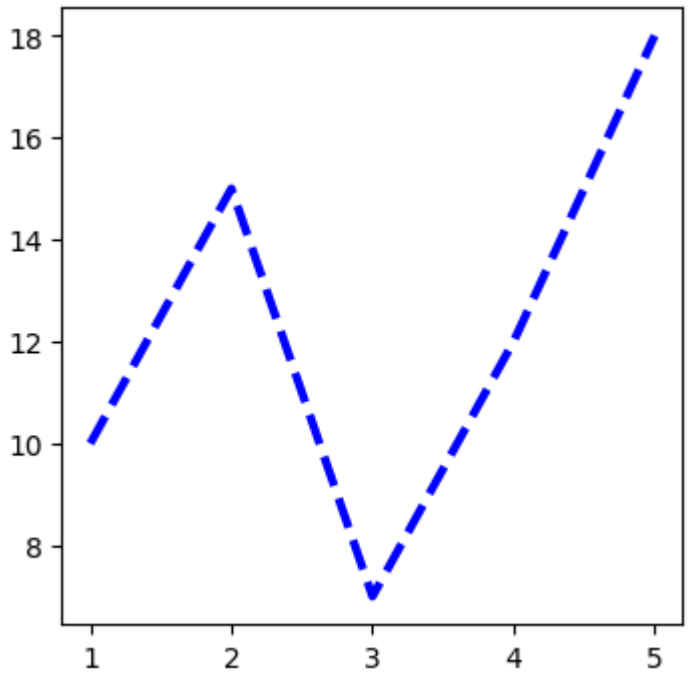
In [3]:
```
import matplotlib.pyplot as plt

# Datos
x = [1, 2, 3, 4, 5]
y = [10, 15, 7, 12, 18]

plt.figure(figsize=(4,4))
# Cambiar color de la línea
plt.plot(x, y, color='green')  # También puedes usar códigos hexadecimales como '#F

plt.show()
```



Estilo de línea y color.

- Se puede cambiar el estilos de la línea (sólida, discontinua, punteada, etc) y el grosor de la línea.

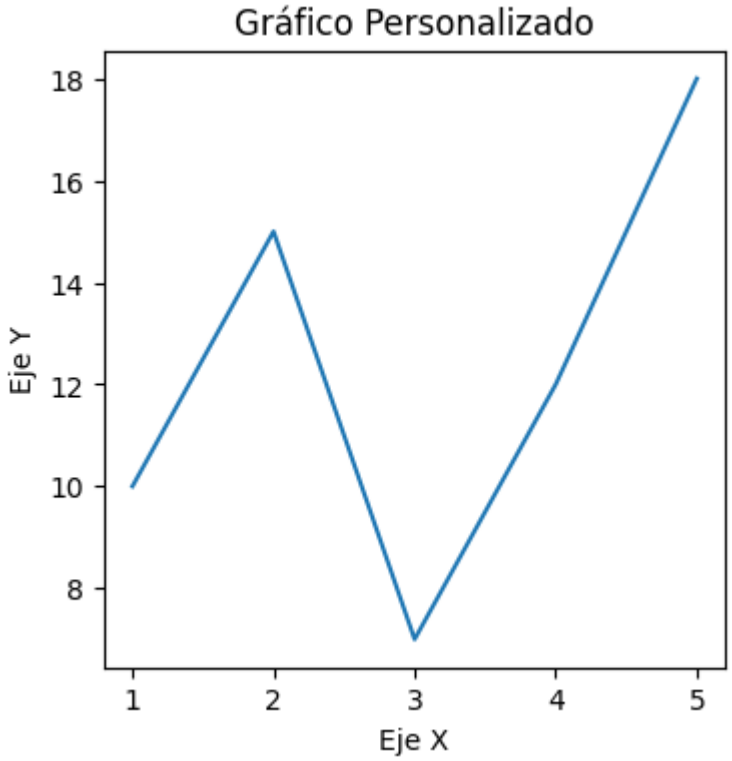
In [4]:
```
x = [1, 2, 3, 4, 5]
y = [10, 15, 7, 12, 18]

plt.figure(figsize=(4,4))
# Estilo de línea y grosor
plt.plot(x, y, linestyle='--', linewidth=3, color='blue')  # Línea discontinua y gr

plt.show()
```



Etiquetas y Títulos.

- Se pueden agregar títulos, etiquetas a los ejes y leyendas para hacer que los gráficos sean más informativos.

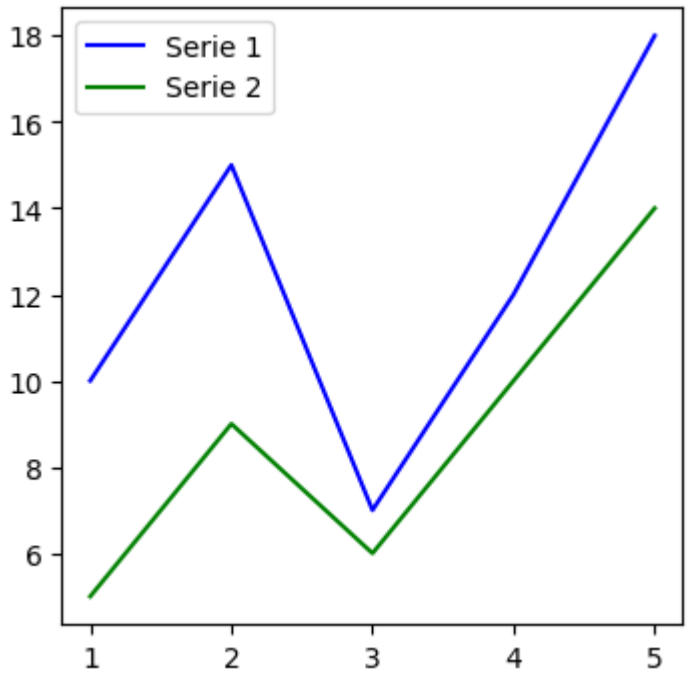
In [6]:
```
x = [1, 2, 3, 4, 5]
y = [10, 15, 7, 12, 18]

plt.figure(figsize=(4,4))
plt.plot(x, y)
plt.title("Gráfico Personalizado")
plt.xlabel("Eje X")
plt.ylabel("Eje Y")
plt.show()
```



Leyendas.

- Las leyendas son útiles cuando se tienen múltiples series de datos en un gráfico.
- Se puede usar el parámetro label en cada serie y luego mostrar la leyenda con plt.legend()

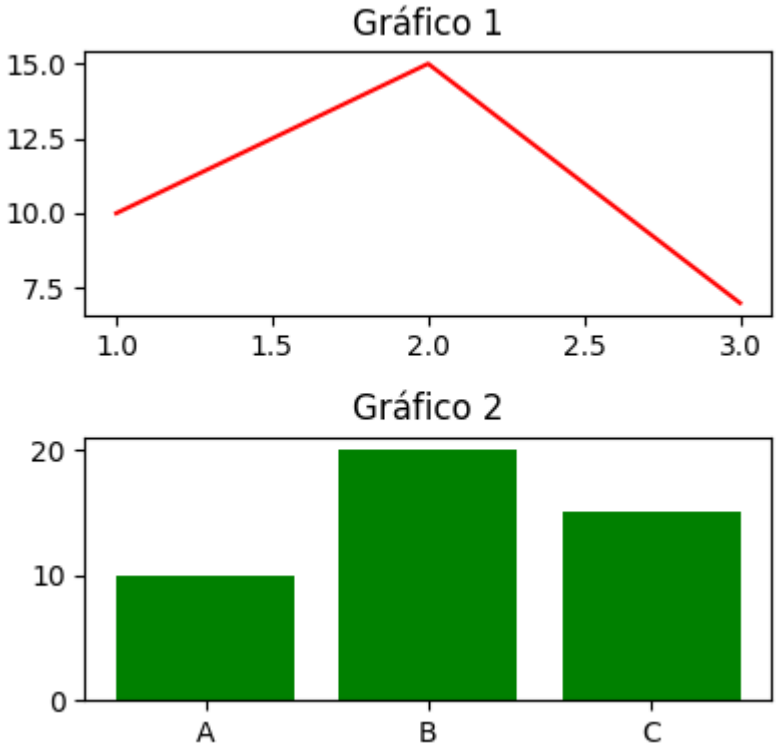
In [7]:
```
x = [1, 2, 3, 4, 5]
y1 = [10, 15, 7, 12, 18]
y2 = [5, 9, 6, 10, 14]

plt.figure(figsize=(4,4))
plt.plot(x, y1, label="Serie 1", color='blue')
plt.plot(x, y2, label="Serie 2", color='green')
plt.legend()
plt.show()
```



Sub-gráficos (Subplots)

- Cuando quieras crear varios gráficos en una sola figura, puedes usar plt.subplots().
- Esto nos permite crear sub-gráficos y personalizarlos individualmente.

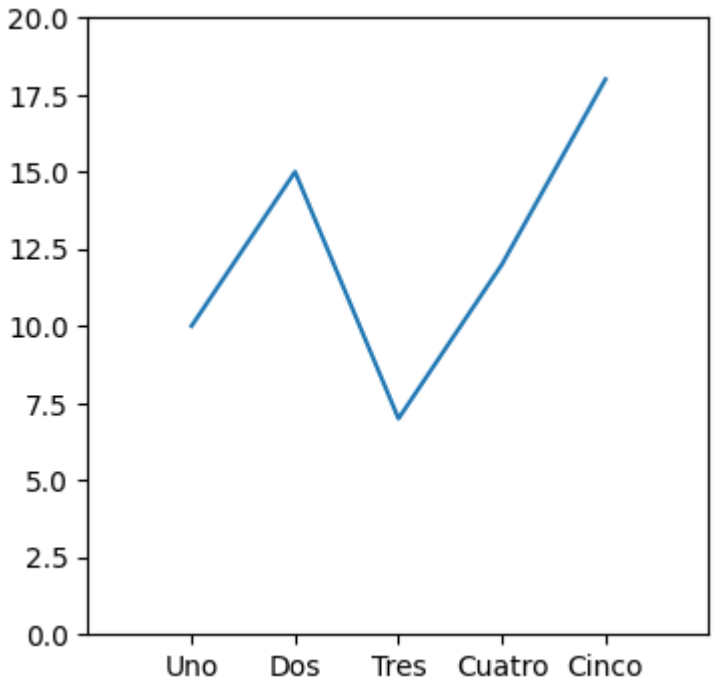
In [8]:
```
# Crear subgráficos: 2 filas, 1 columna
fig, axs = plt.subplots(2, 1, figsize=(4, 4))

# Gráfico 1 (arriba)
axs[0].plot([1, 2, 3], [10, 15, 7], color='red')
axs[0].set_title("Gráfico 1")

# Gráfico 2 (abajo)
axs[1].bar(['A', 'B', 'C'], [10, 20, 15], color='green')
axs[1].set_title("Gráfico 2")

# Mostrar los subgráficos
plt.tight_layout()
plt.show()
```



Configuración de los Ejes.

- Se pueden personalizar los ejes de los gráficos, como establecer los límites de los ejes, cambiar los ticks, y rotar las etiquetas.

In [9]:
```
x = [1, 2, 3, 4, 5]
y = [10, 15, 7, 12, 18]


plt.figure(figsize=(4,4))
# Crear gráfico
plt.plot(x, y)
# Cambiar límites de los ejes
plt.xlim(0, 6)  # Límite para el eje X
plt.ylim(0, 20)  # Límite para el eje Y
# Cambiar los ticks
plt.xticks([1, 2, 3, 4, 5], ['Uno', 'Dos', 'Tres', 'Cuatro', 'Cinco'])
# Mostrar gráfico
plt.show()
```



Estilos y temas en Matplotlib.

- Matplotlib permite cambiar el estilo de los gráficos para darles una apariencia más profesional o adaptarlos a diferentes necesidades.
- En esta sección, aprenderemos cómo aplicar estilos predefinidos y personalizar los gráficos aún mas.

Cambiar el estilo del Gráfico.

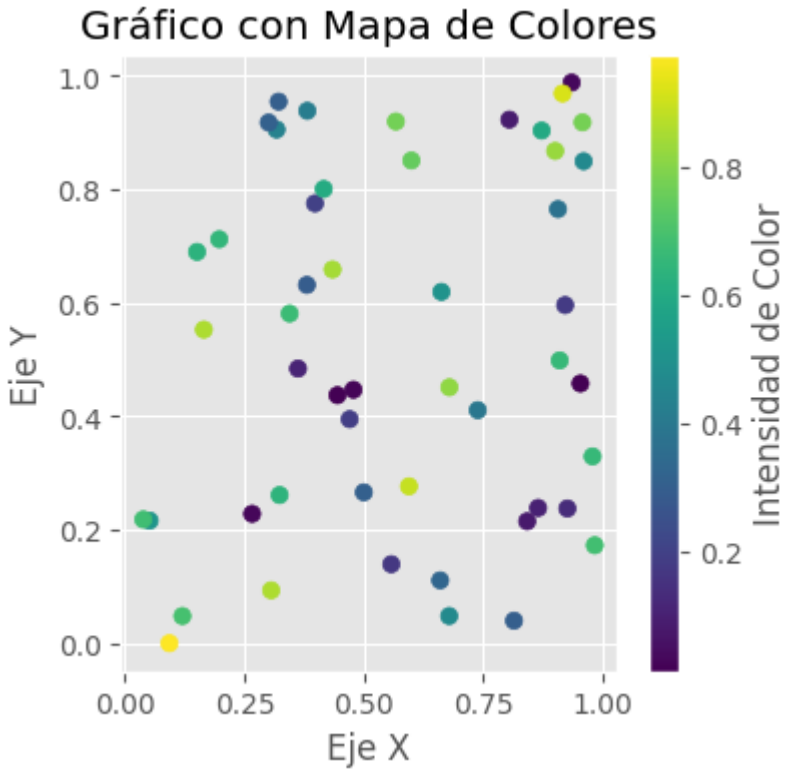
- Matplotlib proporciona varios estilos predefinidos que se pueden aplicar fácilmente usando plt.style.use().
- Esos estilos cambian automáticamente los colores, fuentes, líneas y otros elementos del gráfico.

In [11]:
```
# Aplicar un estilo
plt.style.use('ggplot')  # Cambia el diseño del gráfico

x = [1, 2, 3, 4, 5]
y = [10, 15, 7, 12, 18]


plt.figure(figsize=(4,4))
plt.plot(x, y, marker='o')
plt.xlabel("Eje X")
plt.ylabel("Eje Y")
plt.title("Gráfico con Estilo ggplot")

plt.show()
```

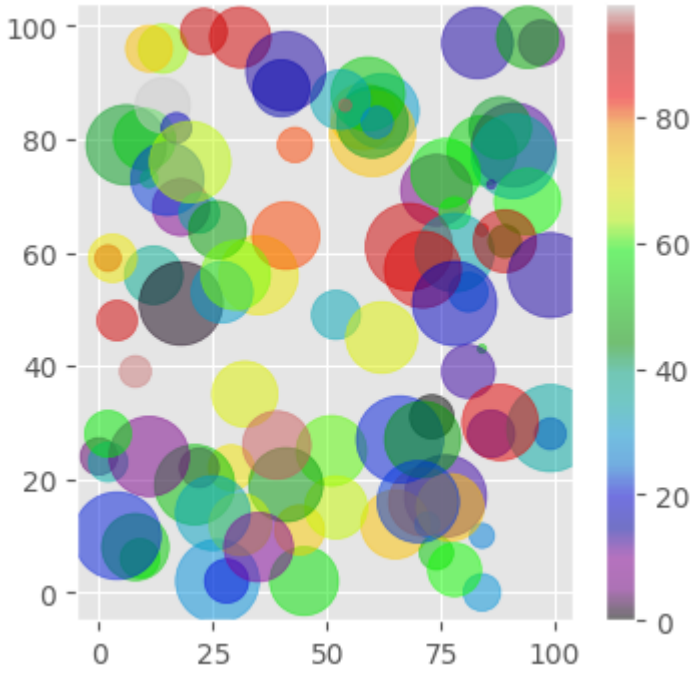
Personalización de colores con Mapas de colores (Colormap)

- Matplotlib incluye una serie de mapas de colores (colormaps) que se pueden usar para aplicar degradados o paletas de colores a los gráficos.

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

x = np.random.rand(50)
y = np.random.rand(50)
colores = np.random.rand(50)  # Asignar colores aleatorios


plt.figure(figsize=(4,4))
# Crear gráfico de dispersión con un colormap
plt.scatter(x, y, c=colores, cmap='viridis')
# Agregar una barra de colores
plt.colorbar(label="Intensidad de Color")
plt.xlabel("Eje X")
plt.ylabel("Eje Y")
plt.title("Gráfico con Mapa de Colores")
plt.show()
```



In [ ]:
```
import numpy as np

x = np.random.randint(100, size=(100))
y = np.random.randint(100, size=(100))
colors = np.random.randint(100, size=(100))
sizes = 10 * np.random.randint(100, size=(100))
plt.figure(figsize=(4,4))
plt.scatter(x, y, c=colors, s=sizes, alpha=0.5, cmap='nipy_spectral')
plt.colorbar()
plt.show()
```

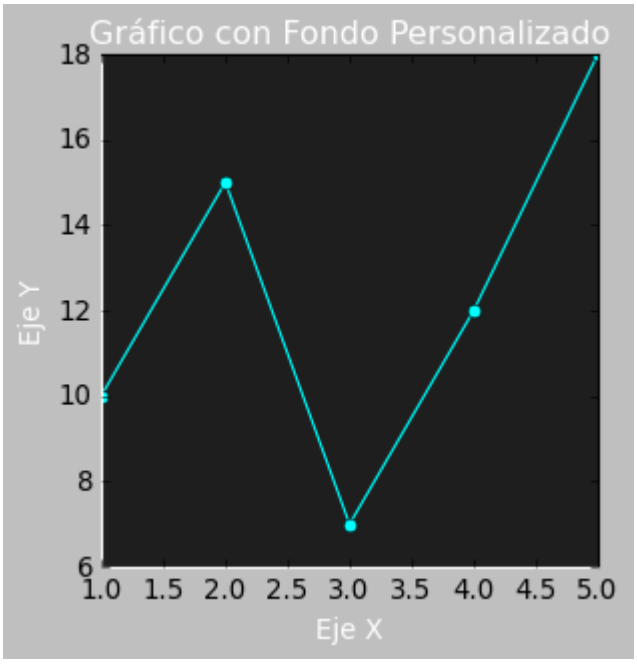


Cambiar el tema de fondo.

- Si deseas modificar el color de fondo del gráfico, se puede usar plt.style.use('dark_backgroud') o modificar manualmente el fondo con ax.set_facecolor().

In [47]:
```
plt.style.use("classic")
```

In [48]:
```
# Crear figura y ejes
fig, ax = plt.subplots(figsize=(4,4))


#plt.figure(figsize=(4,4))
# Cambiar el color de fondo del gráfico
ax.set_facecolor('#1e1e1e')  # Color oscuro


# Datos
x = [1, 2, 3, 4, 5]
y = [10, 15, 7, 12, 18]

# Crear gráfico
ax.plot(x, y, color='cyan', marker='o')

# Etiquetas y título
ax.set_xlabel("Eje X", color='white')
ax.set_ylabel("Eje Y", color='white')
ax.set_title("Gráfico con Fondo Personalizado", color='white')

# Cambiar color de los ejes
ax.spines['bottom'].set_color('white')
ax.spines['left'].set_color('white')
ax.xaxis.label.set_color('white')
ax.yaxis.label.set_color('white')
plt.show()
```



Colormaps Disponibles

Su Reverse --> Accent_r (Asi para todos los colormaps disponibles)

Nombre						
Accent	Oranges	RdBu	YlGn	cool	gist_yarg	plasma
Blues	PRGn	RdGy	YlGnBu	coolwarm	gnuplot	prism
BrBg	Paired	RdPu	YlOrBr	copper	gnuplot2	rainbow
BuGn	Pastel1	RdYlBu	YlOrRd	cubehelix	gray	seismic
BuPu	Pastel2	RdYlGn	afmhot	flag	hot	spring
CMRmap	PiYG	Reds	autumn	gist_earth	hsv	tab10
Dark2	PuBu	Set1	binary	gist_gray	inferno	magma
GnBu	PuBuGn	Set2	bone	gist_heat	jet	tab20b
Greens	PuOr	Set3	brg	gist_ncar	nipy_spectral	tab20c
Greys	PuRd	Spectral	bwr	gist_rainbow	ocean	terrain
OrRd	Purples	Wistia	cividis	gist_stern	pink	twilight
twilight_shifted	viridis	winter	tab20	summer		

Tipos de graficos en Matplotlib.

- Matplotlib permite crear una gran variedad de gráficos para representar datos de diferentes formas.
- En esta sección, veremos los tipos más comunes, cómo usarlos y cuándo aplicarlos.

Gráfico de líneas (plt.plot())

- Matplotlib permite crear una gran variedad de gráficos para representar datos de diferentes formas.
- En esta sección, veremos los tipos más comunes, cómo usarlos y cuándo aplicarlos.
- Se usa pa visualizar tendencias, como ventas a lo largo del tiempo.

In [49]:
```
x = [1, 2, 3, 4, 5]
y = [10, 15, 7, 12, 18]
plt.figure(figsize=(4,4))
# Crear gráfico de línea
plt.plot(x, y, marker='o', linestyle='-', color='b', linewidth=2)

# Etiquetas y título
plt.xlabel("Tiempo")
plt.ylabel("Valor")
plt.title("Gráfico de Líneas")
plt.show()
```

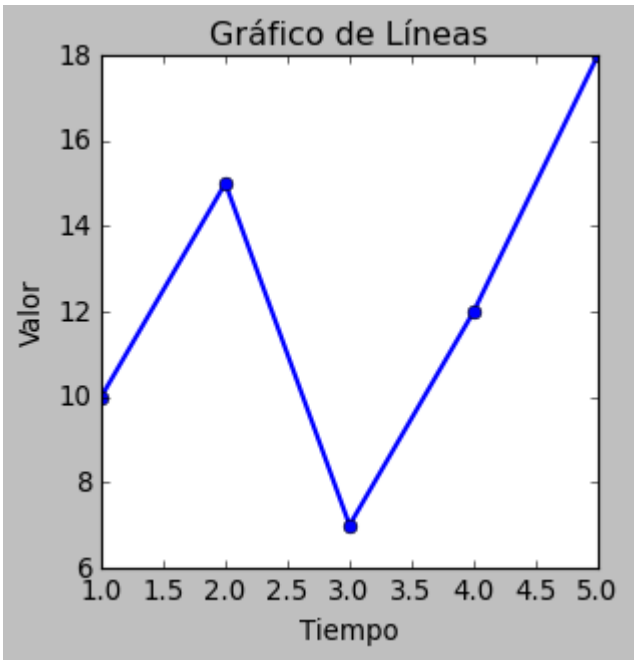


Gráfico de barras (plt.bar())

- Se usa para comparar valores entre diferentes categorías.

In [50]:
```
categorias = ['A', 'B', 'C', 'D']
valores = [10, 20, 15, 25]
plt.figure(figsize=(4,4))
# Crear gráfico de barras
plt.bar(categorias, valores, color=['blue', 'green', 'red', 'purple'])

# Etiquetas y título
plt.xlabel("Categorías")
plt.ylabel("Valores")
plt.title("Gráfico de Barras")

plt.show()
```

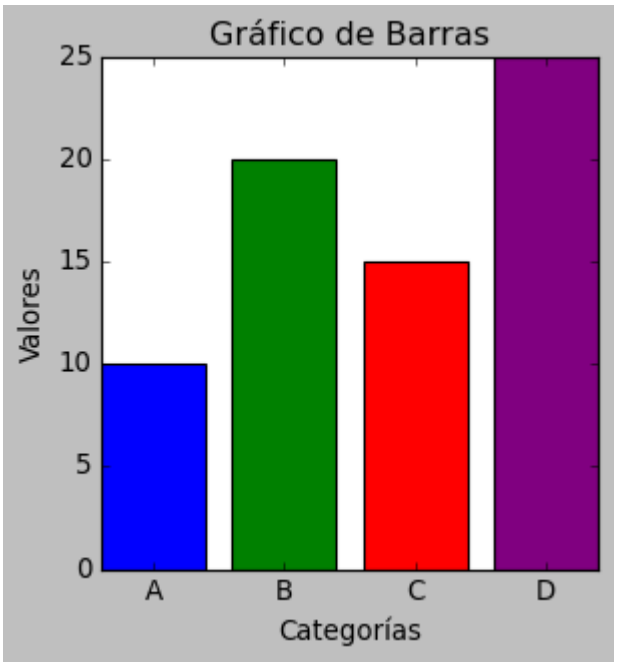


Gráfico de barras horizontales (plt.barch())

- Similar al gráfico de barras, pero con las barras horizontales.

In [51]:
```
categorias = ['A', 'B', 'C', 'D']
valores = [10, 20, 15, 25]
plt.figure(figsize=(4,4))
# Crear gráfico de barras horizontales
plt.barh(categorias, valores, color='orange')

# Etiquetas y título
plt.xlabel("Valores")
plt.ylabel("Categorías")
plt.title("Gráfico de Barras Horizontales")

plt.show()
```

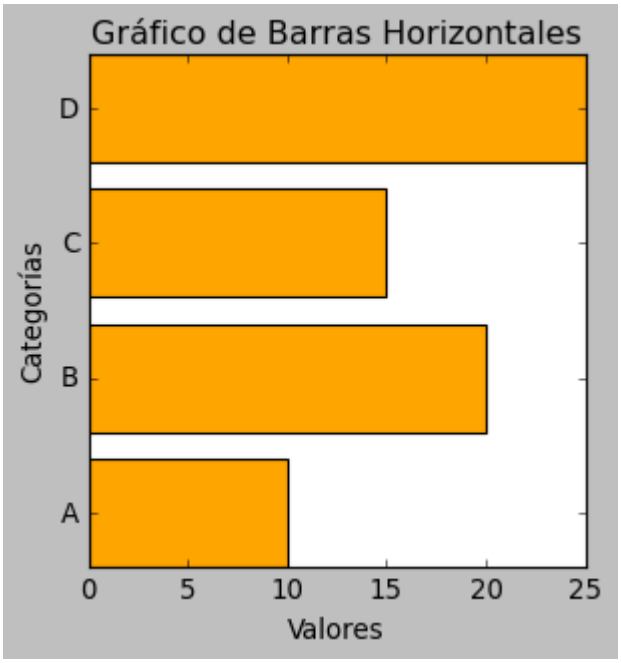
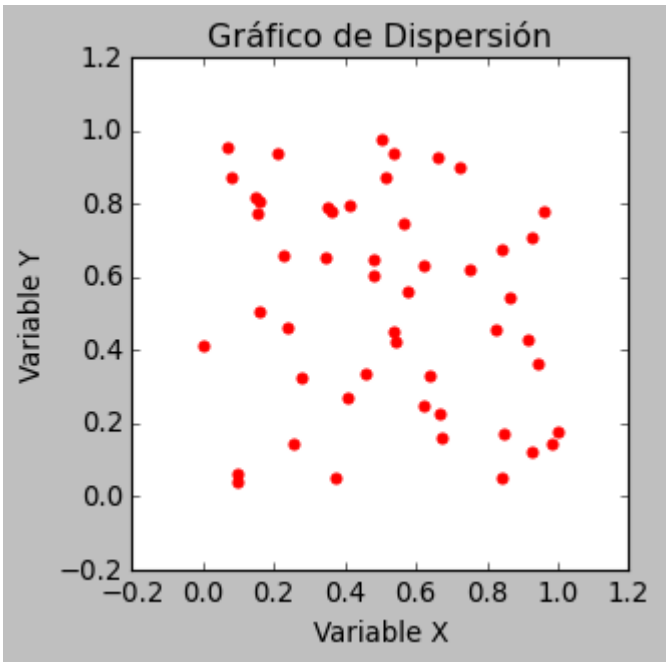


Gráfico de dispersión (plt.scatter())

- Se usa para mostrar la relación entre dos variables.

In [52]:
```
import numpy as np
x = np.random.rand(50)
y = np.random.rand(50)
plt.figure(figsize=(4,4))
# Crear gráfico de dispersión
plt.scatter(x, y, color='red', marker='o')

# Etiquetas y título
plt.xlabel("Variable X")
plt.ylabel("Variable Y")
plt.title("Gráfico de Dispersión")

plt.show()
```



Histograma (plt.hist())

- Se usa para representar distribuciones de datos.

In [53]:
```
datos = np.random.randn(1000)
plt.figure(figsize=(4,4))
# Crear histograma
plt.hist(datos, bins=20, color='purple', alpha=0.7)

# Etiquetas y título
plt.xlabel("Valores")
plt.ylabel("Frecuencia")
plt.title("Histograma")

plt.show()
```

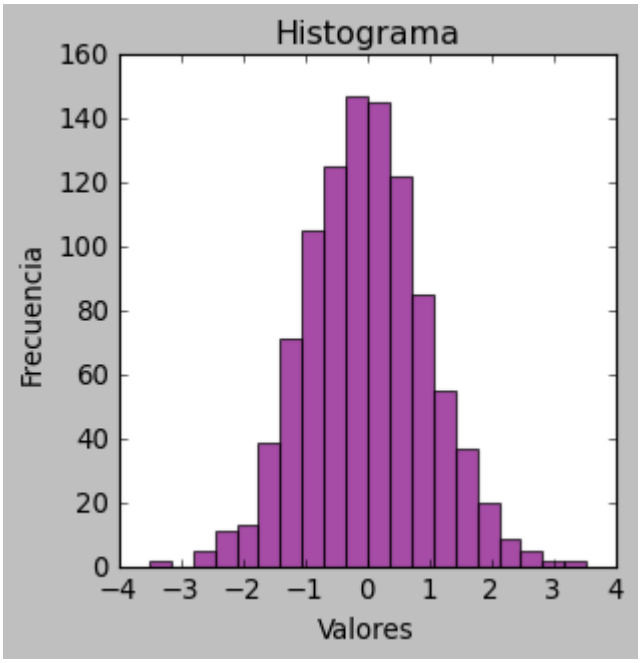
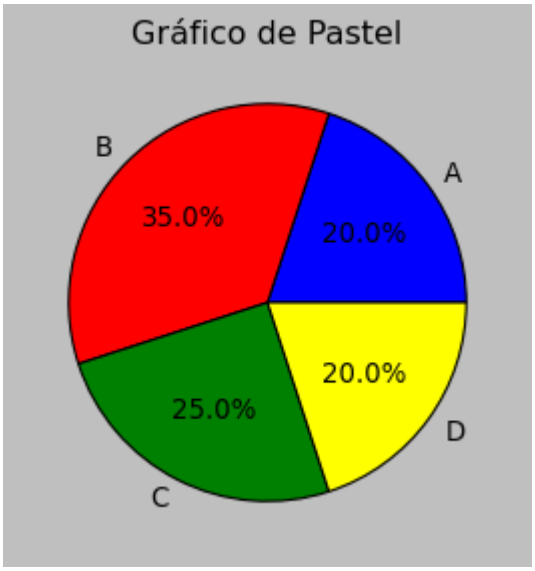


Gráfico de pastel (plt.pie())

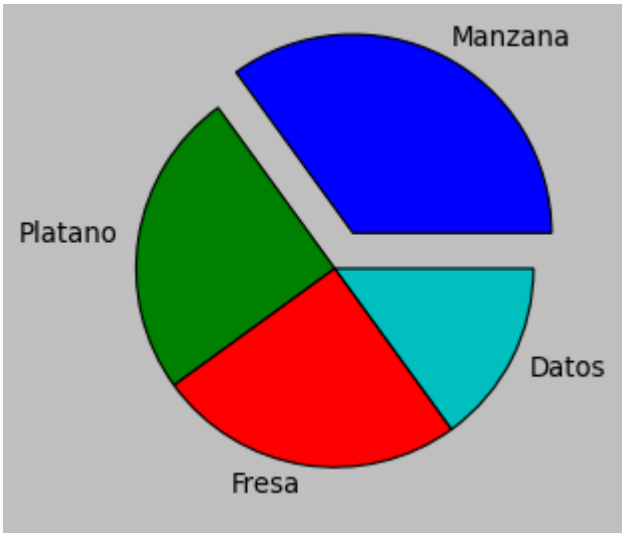
- Se usa para mostrar proporciones de un conjunto de datos.

In [54]:
```
categorias = ['A', 'B', 'C', 'D']
valores = [20, 35, 25, 20]
plt.figure(figsize=(4,4))
# Crear gráfico de pastel
plt.pie(valores, labels=categorias, autopct='%1.1f%%', colors=['blue', 'red', 'gree

# Título
plt.title("Gráfico de Pastel")

plt.show()
```



Con el parametro explode, podemos hacer que una de las cuñas del grafico se separe de las de mas.

In [55]:
```
import matplotlib.pyplot as plt
import numpy as np

y = np.array([35, 25, 25, 15])
mylabels = ["Manzana", "Platano", "Fresa", "Datos"]
myexplode = [0.2, 0, 0, 0]
plt.figure(figsize=(4,4))
plt.pie(y, labels = mylabels, explode = myexplode)
plt.show()
```



Tambien podemos agregar una sombre con shadows

In [56]:
```
import matplotlib.pyplot as plt
import numpy as np

y = np.array([35, 25, 25, 15])
mylabels = ["Manzana", "Platano", "Fresa", "Datos"]
myexplode = [0.2, 0, 0, 0]
plt.figure(figsize=(4,4))
plt.pie(y, labels = mylabels, explode = myexplode, shadow=True)
plt.show()
```

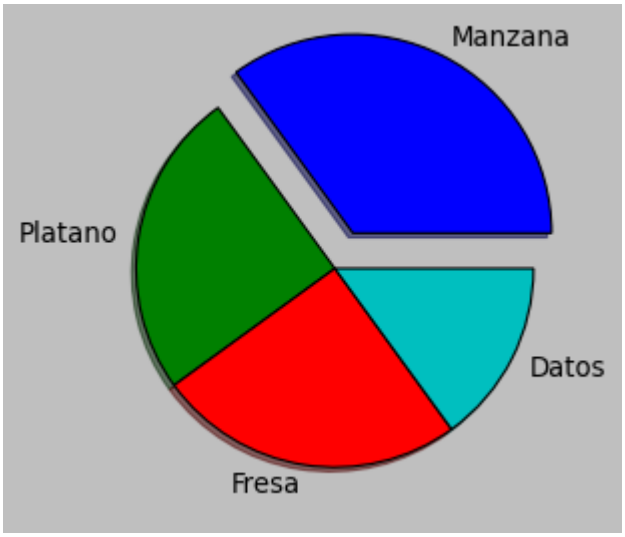


Gráfico de Área (plt.fill_between())

- Similar al gráfico de líneas, pero con el área de la curva rellenada.

In [57]:
```
x = np.linspace(0, 10, 100)
y = np.sin(x)
plt.figure(figsize=(4,4))
# Crear gráfico de área
plt.fill_between(x, y, color="skyblue", alpha=0.4)

# Etiquetas y título
plt.xlabel("Tiempo")
plt.ylabel("Valor")
plt.title("Gráfico de Área")

plt.show()
```

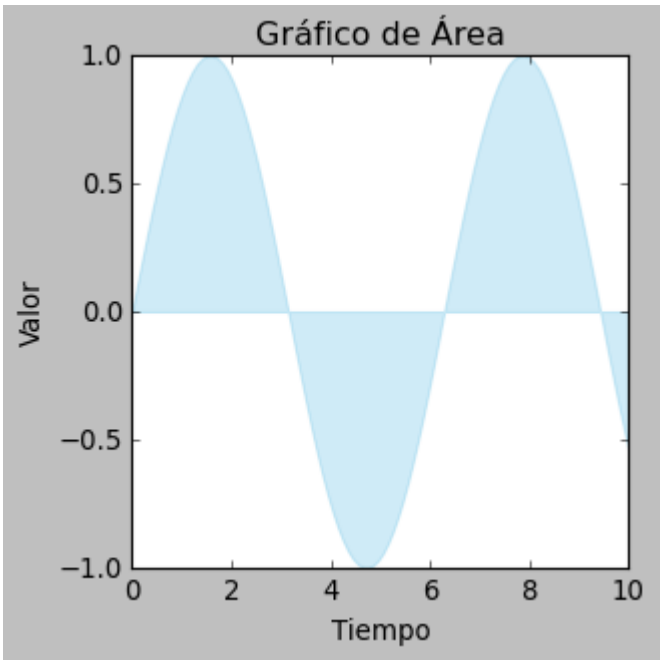


Gráfico de caja y Bigotes (plt.boxplot())

- Se usa para mostrar la distribución de datos y detectar valores atípicos.

In [58]:
```
datos = [np.random.rand(10) * 10 for _ in range(4)]
plt.figure(figsize=(4,4))
# Crear gráfico de caja
plt.boxplot(datos, labels=["A", "B", "C", "D"])

# Etiquetas y título
plt.xlabel("Grupos")
plt.ylabel("Valores")
plt.title("Gráfico de Caja y Bigotes")

plt.show()
```

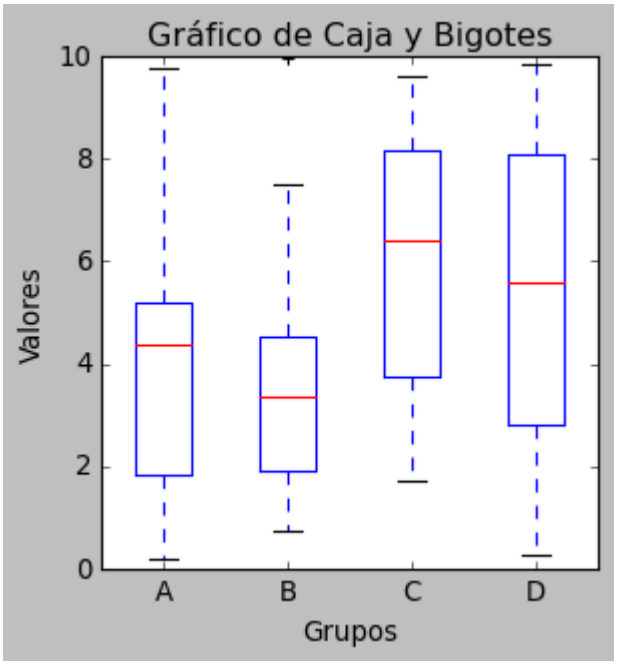
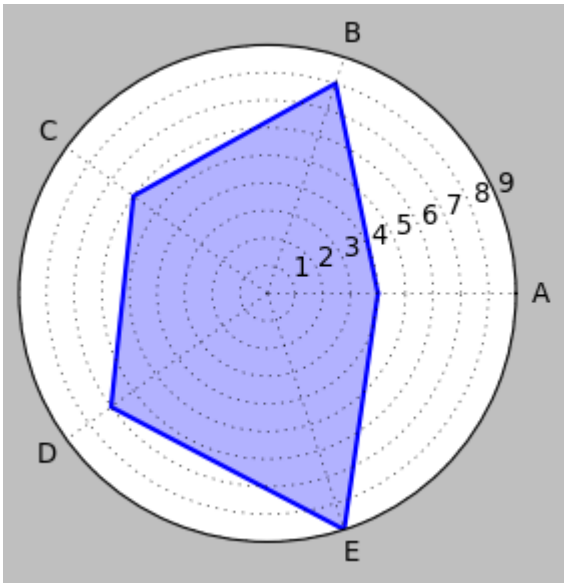


Gráfico de Radar.

- Se usa para visualizar múltiples variables en un mismo gráfico.

In [59]:
```
labels = ['A', 'B', 'C', 'D', 'E']
valores = [4, 8, 6, 7, 9]

# Convertir a coordenadas polares
angulos = np.linspace(0, 2 * np.pi, len(labels), endpoint=False).tolist()
valores += valores[:1]
angulos += angulos[:1]

# Crear gráfico
fig, ax = plt.subplots(figsize=(4, 4), subplot_kw=dict(polar=True))
ax.fill(angulos, valores, color='blue', alpha=0.3)
ax.plot(angulos, valores, color='blue', linewidth=2)

# Configurar etiquetas
ax.set_xticks(angulos[:-1])
ax.set_xticklabels(labels)

# Mostrar gráfico
plt.show()
```



Gráficos 3D con MatplotLib

- Matplotlib nos permite crear gráficos en tres dimensiones (3D) utilizando el módulo mpl_toolkits.mplot3d.
- Con esto, podemos generar superficies, dispersión de puntos, curvas espaciales y mas.
- Para habilitar los gráficos 3D en Matplotlib, debemos usar Axes3D y trabajar con los métodos especializados para gráficos tridimensionales.

Configuración básica de un gráfico 3D

- Importamos Axes3D desde mpl_toolkits.mplot3d
- Creamos una figura y un eje 3D con fif,add_subplot(projection='3d')
- Usamos métodos específicos para gráficos 3D, como plot3D(), scatter3D(), etc

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
# Crear datos
x = np.linspace(-5, 5, 100)
```

```
y = np.sin(x)
z = np.cos(x)
# Crear figura y eje 3D
fig = plt.figure()
ax = fig.add_subplot(projection='3d')
# Graficar una línea en 3D
ax.plot3D(x,y,z, color='b', lw=2, ls='--')
# Etiquetas
ax.set_xlabel('Eje X')
ax.set_ylabel('Eje Y')
ax.set_zlabel('Eje Z')
ax.set_title('Grafico 3D de una Linea')
plt.show()
```

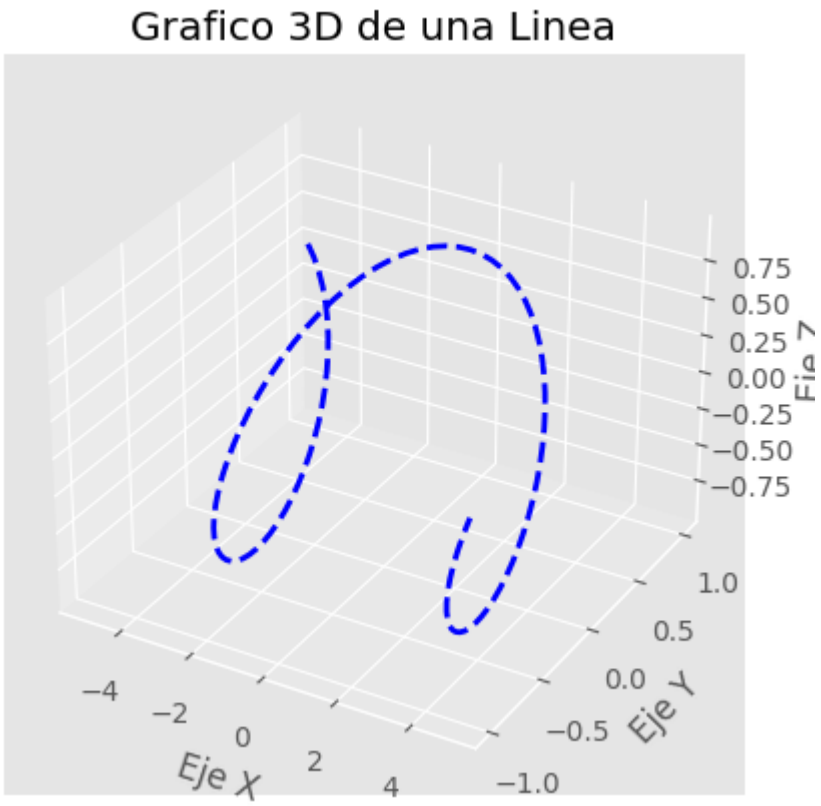


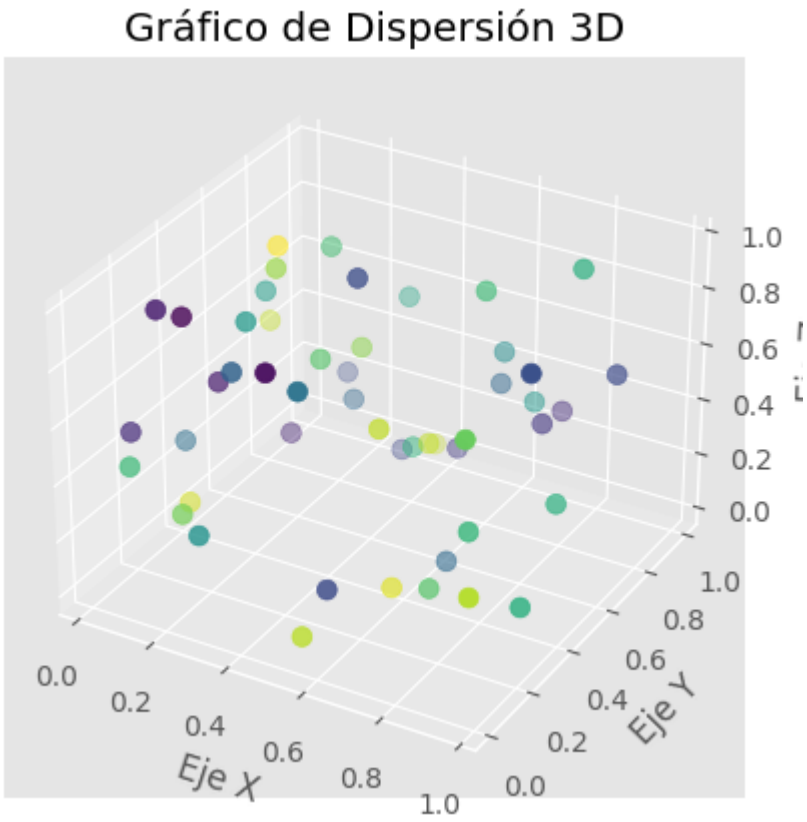
Gráfico de dispersión en 3D

- Podemos graficar puntos en un espacio tridimensional con sctter3D()

In [30]:

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Generar datos aleatorios
np.random.seed(42)
x = np.random.rand(50)
y = np.random.rand(50)
z = np.random.rand(50)
colores = np.random.rand(50)  # Colores aleatorios

# Crear figura y eje 3D
fig = plt.figure()
ax = fig.add_subplot(projection='3d')

# Graficar puntos en 3D
ax.scatter3D(x, y, z, c=colores, cmap='viridis', s=50)

# Etiquetas
```

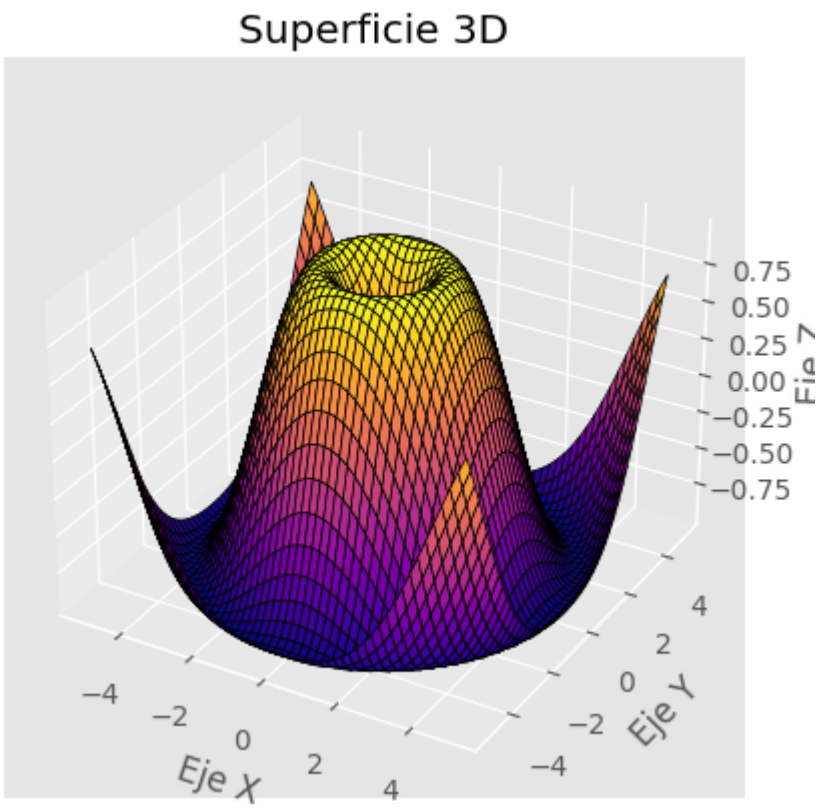
```
ax.set_xlabel('Eje X')
ax.set_ylabel('Eje Y')
ax.set_zlabel('Eje Z')
ax.set_title('Gráfico de Dispersión 3D')

plt.show()
```



Gráficos de Superficies 3D

- Podemos graficar superficies 3D con plt_surface().

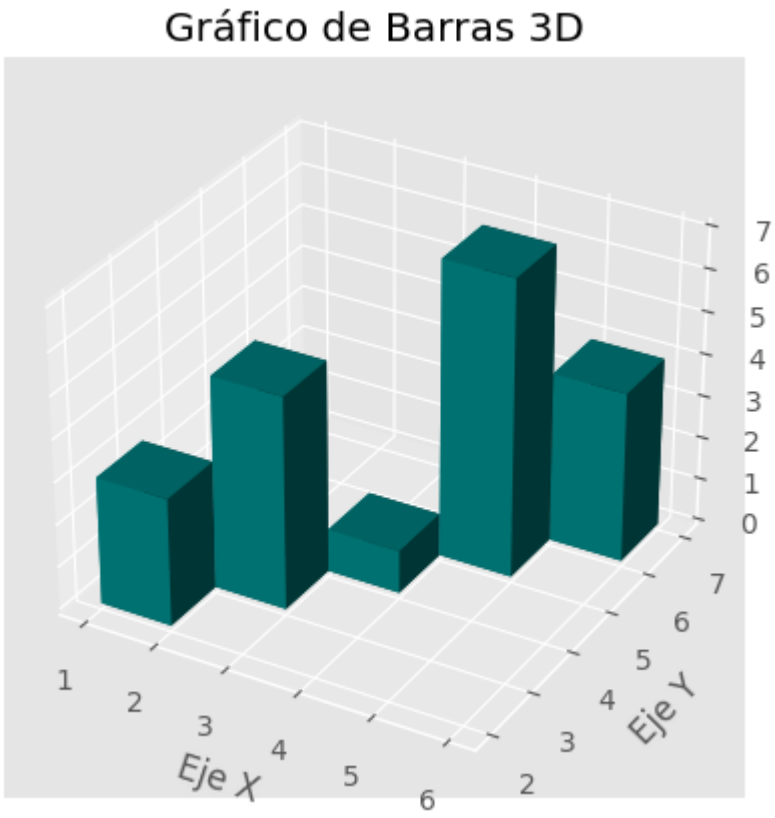
In [31]:
```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Crear malla de coordenadas
x = np.linspace(-5, 5, 50)
y = np.linspace(-5, 5, 50)
X, Y = np.meshgrid(x, y)
Z = np.sin(np.sqrt(X**2 + Y**2))

# Crear figura y eje 3D
fig = plt.figure()
ax = fig.add_subplot(projection='3d')

# Graficar superficie 3D
ax.plot_surface(X, Y, Z, cmap='plasma', edgecolor='black')

# Etiquetas
ax.set_xlabel('Eje X')
ax.set_ylabel('Eje Y')
ax.set_zlabel('Eje Z')
ax.set_title('Superficie 3D')

plt.show()
```



Gráficos de Barras 3D

- Podemos gráficar barras tridimensionales en bard3D()

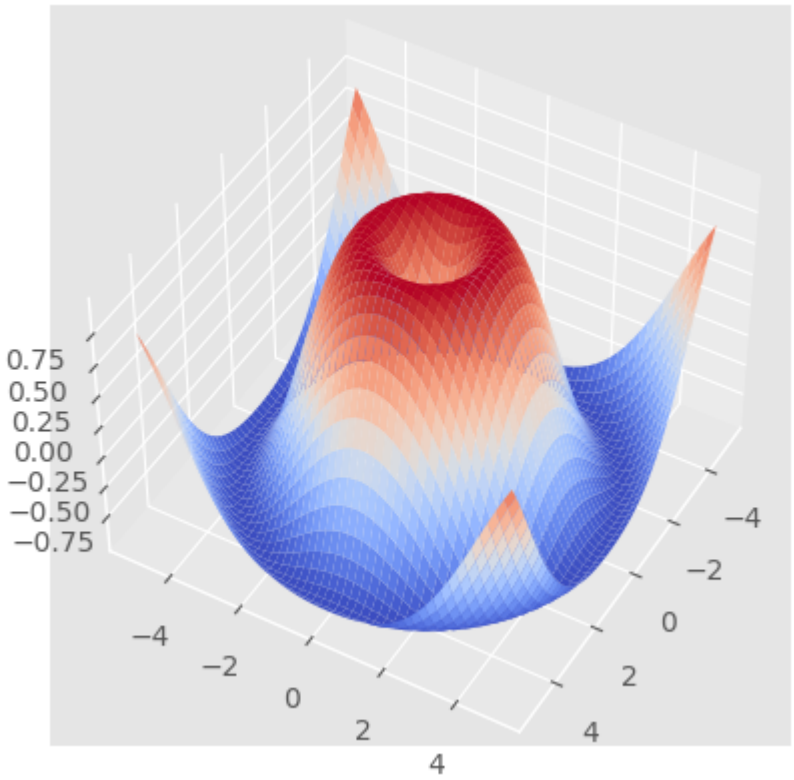
In [32]:
```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Datos
x = np.array([1, 2, 3, 4, 5])
y = np.array([2, 3, 4, 5, 6])
z = np.zeros(5)  # Altura inicial en 0
dx = dy = np.ones(5)  # Ancho de las barras
dz = [3, 5, 1, 7, 4]  # Altura de las barras

# Crear figura y eje 3D
fig = plt.figure()
ax = fig.add_subplot(projection='3d')

# Graficar barras 3D
ax.bar3d(x, y, z, dx, dy, dz, color='teal')

# Etiquetas
ax.set_xlabel('Eje X')
ax.set_ylabel('Eje Y')
ax.set_zlabel('Altura')
ax.set_title('Gráfico de Barras 3D')

plt.show()
```



Personalización de Vistas y Ángulos.

- Podemos modificar la perspectiva del gráfico 3D con view_init()

In [33]:
```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Datos
x = np.linspace(-5, 5, 50)
y = np.linspace(-5, 5, 50)
X, Y = np.meshgrid(x, y)
Z = np.sin(np.sqrt(X**2 + Y**2))

# Crear figura y eje 3D
fig = plt.figure()
ax = fig.add_subplot(projection='3d')

# Graficar superficie 3D
ax.plot_surface(X, Y, Z, cmap='coolwarm')

# Modificar ángulo de vista
ax.view_init(elev=45, azim=30)  # Elevación y ángulo de rotación

plt.show()
```



Gráficos e Imágenes y Maps

- Matplotlib nos permite trabajar con imágenes y mapas utilizando funciones especializadas como imshow() y contourf().
- Con estas herramientas, podemos visualizar datos en formato de imagen, cargar mapas de calor y representar funciones matemáticas en dos dimensiones.

Mostrar imágenes con imshow()

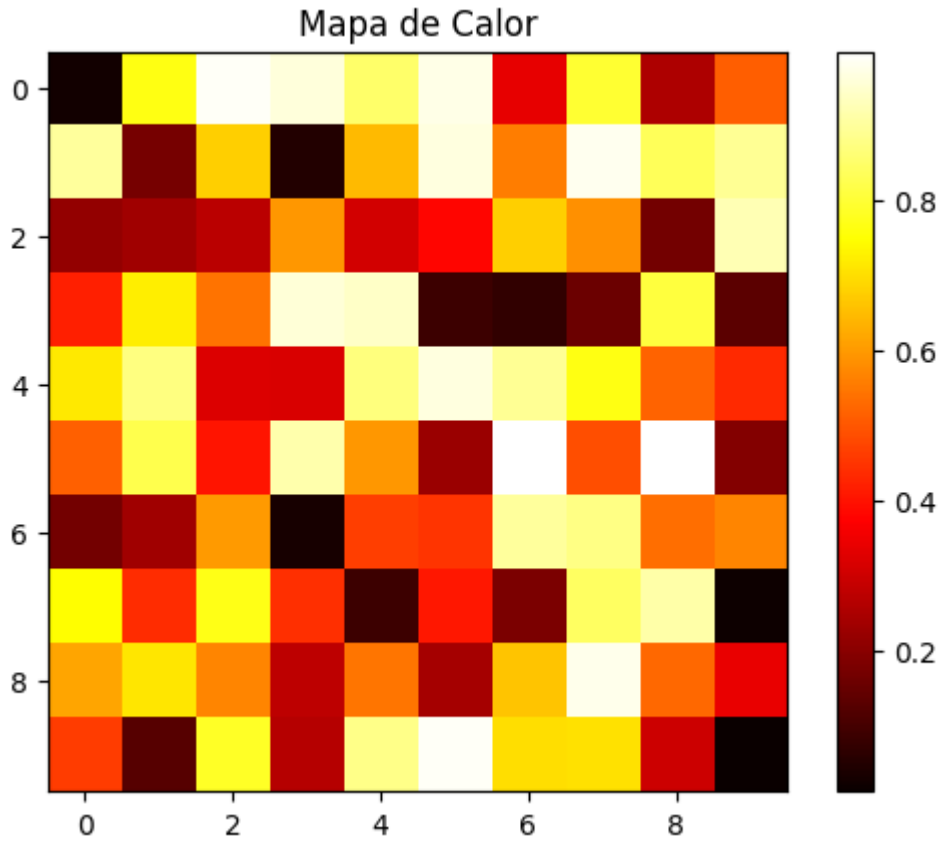
- La función imshow() nos permite cargar y mostrar imágenes en Matplotlib.
- Se puede usar para representar mapas de calor, matrices de datos o simplemente visualizar imágenes almacenadas.

In [3]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

img = mpimg.imread('matplotlib.png')

plt.imshow(img)
plt.axis('off')
plt.title('Imagen Cargada en Matplotlib')
plt.show()
```

Mapas de Calor (Heatmapas) con imshow()

- Podemos visualizar matrices de datos como imágenes de colores con imshow()

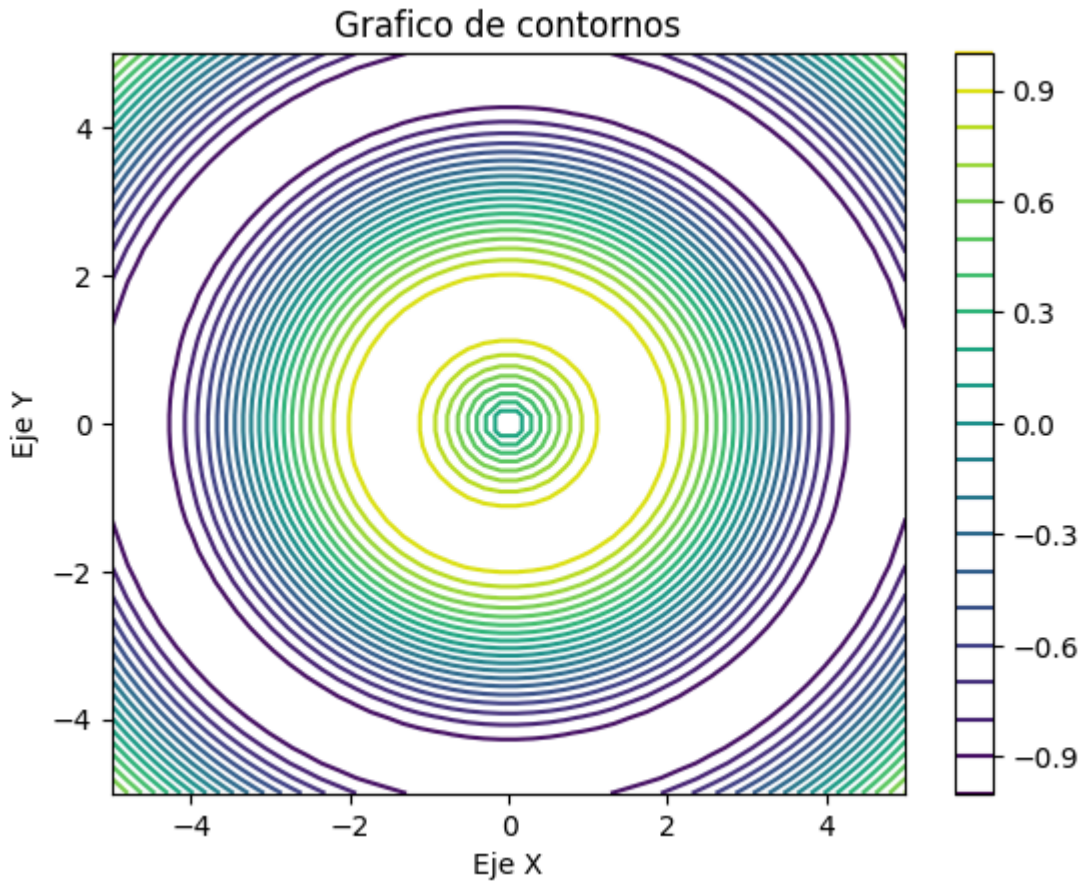
In [4]:
```
import numpy as np
import matplotlib.pyplot as plt

data = np.random.rand(10, 10)

plt.imshow(data, cmap='hot', interpolation='nearest')
plt.colorbar()
plt.title('Mapa de Calor')
plt.show()
```



Contornos y Mapas de Nivel (contour) y (contourf)

- Los gráficos de contornos permiten visualizar mapas de nivel de funciones matemáticas en 2D

In [7]:
```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-5, 5, 50)
y = np.linspace(-5, 5, 50)
X, Y = np.meshgrid(x, y )
Z = np.sin(np.sqrt(X**2 + Y**2))

plt.contour(X, Y, Z, levels=20, cmap='viridis')
plt.colorbar()
plt.title('Grafico de contornos')
plt.xlabel('Eje X')
plt.ylabel('Eje Y')
plt.show()
```



Mostrar Imagenes en Escala de Grises

- Podemos convertir una imagen a escala de grises usando cmap='gray'

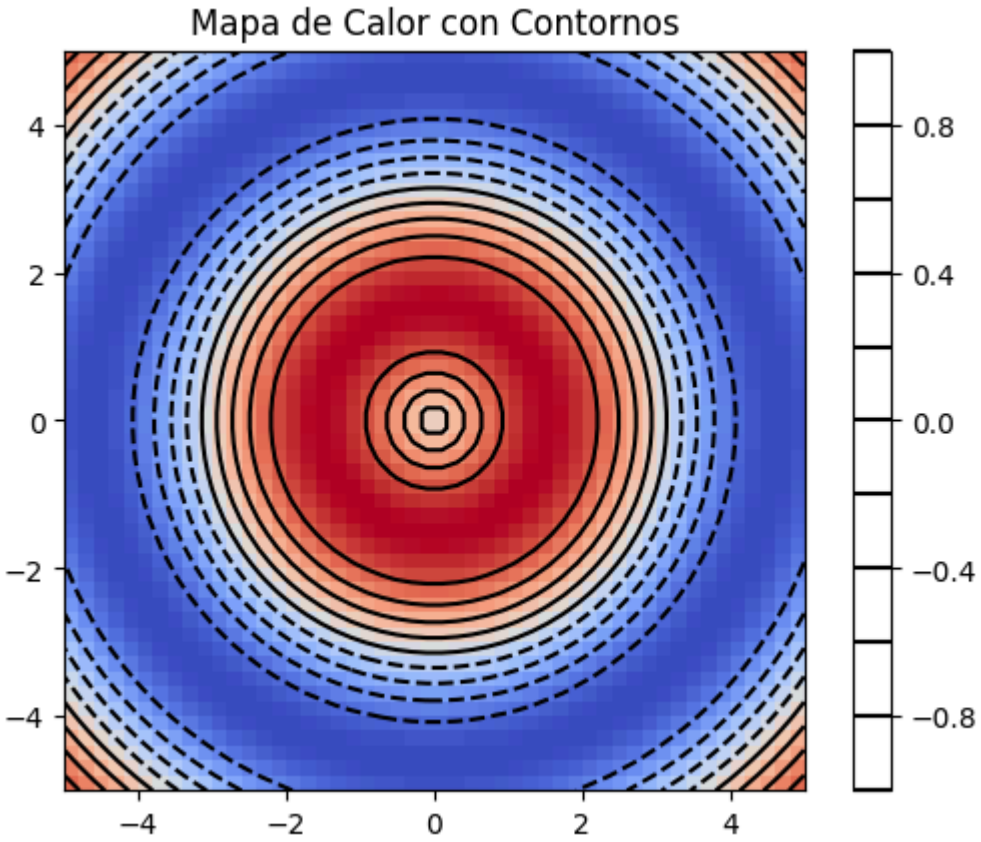
In [11]:
```
import matplotlib.pyplot as plt
import matplotlib.image as img

imagen = img.imread('matplotlib.png')

plt.imshow(imagen, cmap='gray')
plt.axis('off')
plt.title('Imgen en Escala de Grises')
plt.show()
```

Superposición de Contornos sobre un Mapa de Calor

- Podemos combinar imshow() con contour() para superponer contornos en una mapa de calor.

In [13]:
```
x = np.linspace(-5, 5, 50)
y = np.linspace(-5, 5, 50)
X, Y, = np.meshgrid(x, y)
Z = np.sin(np.sqrt(X**2 + Y**2))
```

```
plt.imshow(Z, extent=[-5, 5, -5, 5], origin='lower', cmap='coolwarm')

plt.contour(X, Y, Z, colors='black', levels=10)

plt.colorbar()
plt.title('Mapa de Calor con Contornos')
plt.show()
```



Interactiviad con Matplotlib

- Matplotlib permite agregar interactividad a los gráficos mediante eventos del mouse, widgets y la integración con mpl_interactions.
- Esto es útil para explorar datos dinámicamente y crear gráficos más interactivos.

Activar la Interactividad con %matplotlib

- Si trabajamos en Jupyter Notebook, podemos usar.
- %matpltlib notebook ---> Activa la interactividad en Jupyter
- %matplotlb wig¿dget ---> Usa widgets interactivos en Jupyter

- Si estamos en n script de Python estándar, la interactividad se habilita automáticamente al usar plt.show()

In [1]:
```
%matplotlib notebook
```

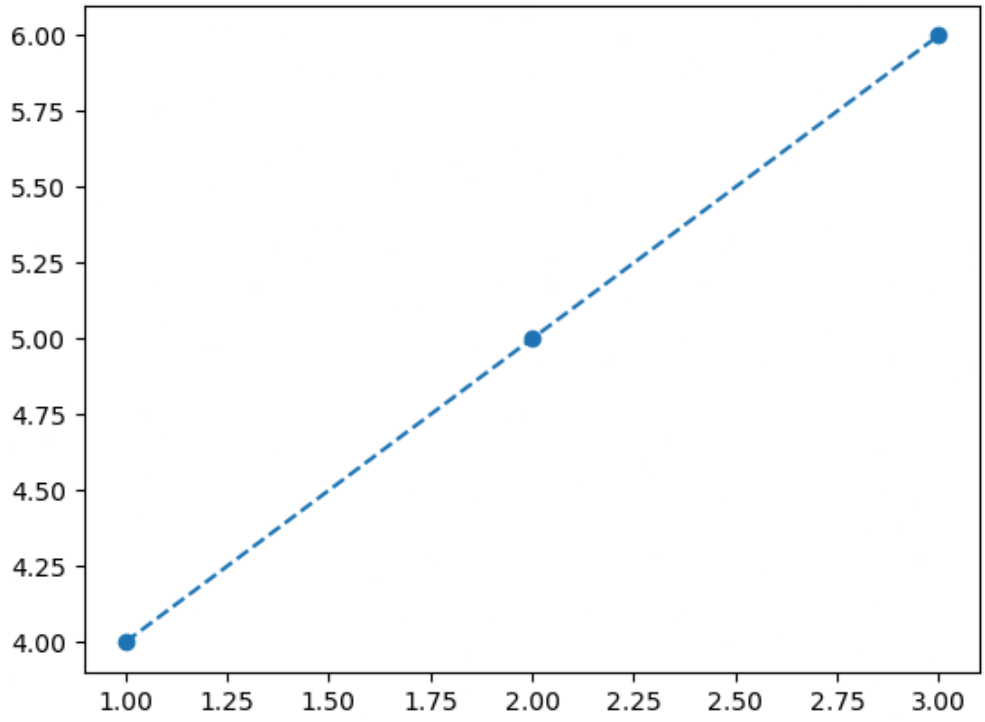
In [2]:
```
%matplotlib widget
```

```
Warning: Cannot change to a different GUI toolkit: widget. Using notebook instead.
```

Eventos del Mouse y del Teclado.

- Podemos capturar eventos del mouse y teclado para interactuar con los gráficos.

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

# Crear una figura y conectar eventos del mouse
fig, ax = plt.subplots()
ax.plot([1, 2, 3], [4, 5, 6], marker="o", linestyle="--")

def on_click(event):
    print(f"Clic detectado en: ({event.xdata}, {event.ydata})")

fig.canvas.mpl_connect("button_press_event", on_click)

plt.show()
```

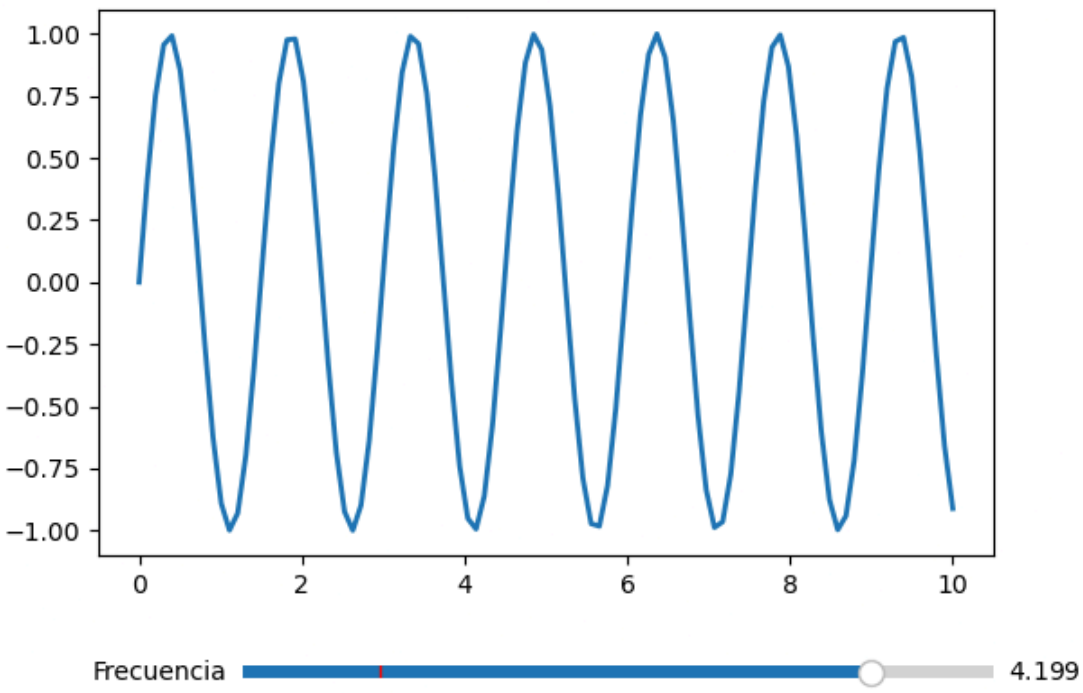


Uso de Widgets Interactivos (Sliders, Botones, etc)

- Los widgets permiten controlar gráficos de forma dinámica.

In [4]:
```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.widgets import Slider

# Crear figura y eje
fig, ax = plt.subplots()
plt.subplots_adjust(bottom=0.25)

x = np.linspace(0, 10, 100)
a = 1
y = np.sin(a * x)
line, = plt.plot(x, y, lw=2)

# Agregar slider
ax_slider = plt.axes([0.25, 0.1, 0.65, 0.03])  # Posición del slider
slider = Slider(ax_slider, 'Frecuencia', 0.1, 5.0, valinit=a)
```

```
# Función para actualizar la gráfica
def update(val):
    line.set_ydata(np.sin(slider.val * x))
    fig.canvas.draw_idle()

slider.on_changed(update)

plt.show()
```



Uso de mpl_interactions para Mayor Interactividad.

- La librería mpl_interactions simplifica la creación de gráficos interactivos. Debemos instalarla con:
- pip install mpl-interactions

In [29]:
```
%pip install mpl-interactions
```

```
Defaulting to user installation because normal site-packages is not writeableNote:
you may need to restart the kernel to use updated packages.

Requirement already satisfied: mpl-interactions in c:\users\lalo\appdata\roaming\p
ython\python311\site-packages (0.24.2)
Requirement already satisfied: matplotlib>=3.7 in c:\programdata\anaconda3\lib\sit
e-packages (from mpl-interactions) (3.7.1)
Requirement already satisfied: contourpy>=1.0.1 in c:\programdata\anaconda3\lib\si
te-packages (from matplotlib>=3.7->mpl-interactions) (1.0.5)
Requirement already satisfied: cycler>=0.10 in c:\programdata\anaconda3\lib\site-p
ackages (from matplotlib>=3.7->mpl-interactions) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\programdata\anaconda3\lib\s
ite-packages (from matplotlib>=3.7->mpl-interactions) (4.25.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\programdata\anaconda3\lib\s
ite-packages (from matplotlib>=3.7->mpl-interactions) (1.4.4)
Requirement already satisfied: numpy>=1.20 in c:\programdata\anaconda3\lib\site-pa
ckages (from matplotlib>=3.7->mpl-interactions) (1.24.3)
Requirement already satisfied: packaging>=20.0 in c:\users\lalo\appdata\roaming\py
thon\python311\site-packages (from matplotlib>=3.7->mpl-interactions) (21.3)
Requirement already satisfied: pillow>=6.2.0 in c:\programdata\anaconda3\lib\site-
packages (from matplotlib>=3.7->mpl-interactions) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\lalo\appdata\roaming\p
ython\python311\site-packages (from matplotlib>=3.7->mpl-interactions) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\lalo\appdata\roami
ng\python\python311\site-packages (from matplotlib>=3.7->mpl-interactions) (2.8.2)
Requirement already satisfied: six>=1.5 in c:\users\lalo\appdata\roaming\python\py
thon311\site-packages (from python-dateutil>=2.7->matplotlib>=3.7->mpl-interaction
s) (1.16.0)
```

In [ ]:

Integración con Numpy, Pandas

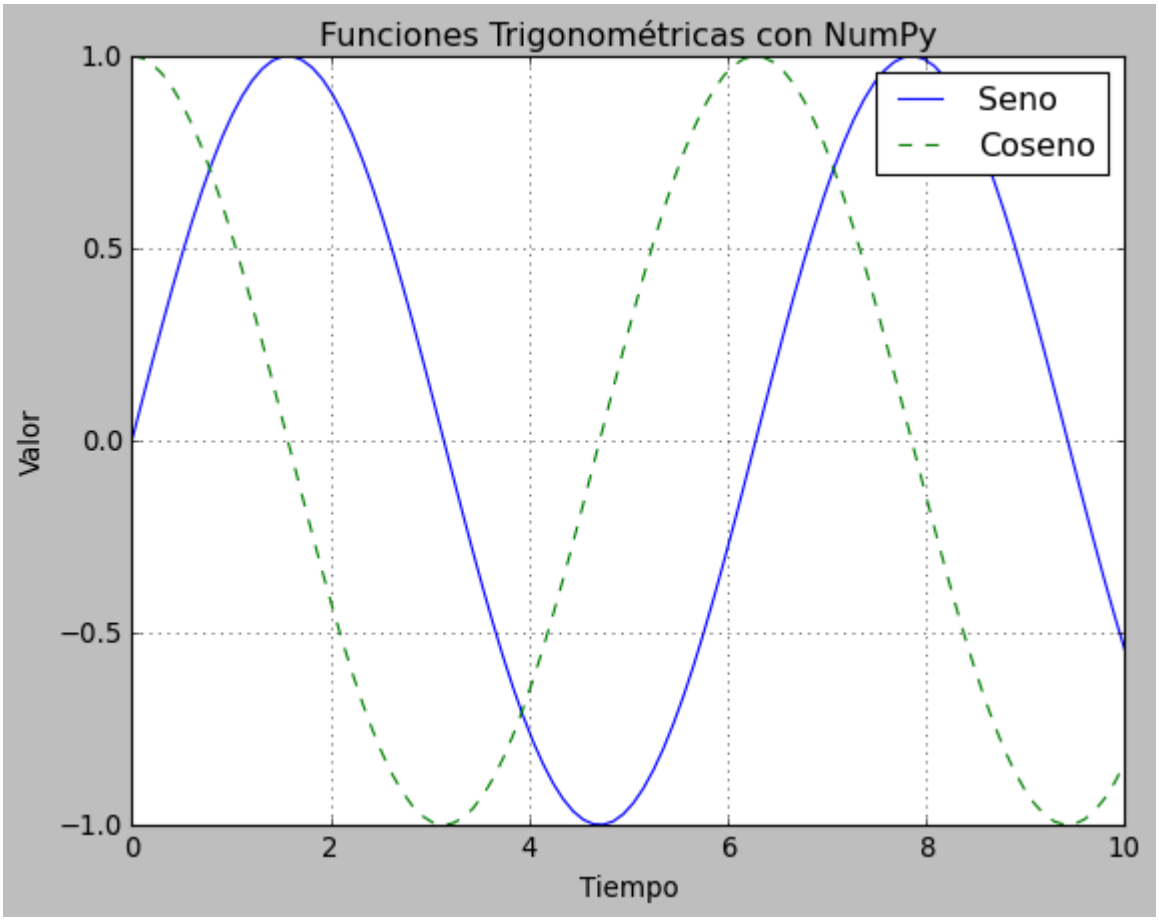
- Matplotlib se integra perfectamente con NumPy, Pandas, lo que nos permite manejar datos estructurados de manera eficiente y crear visualizaciones más avanzadas

Uso de NumPy con Matplotlib

- NumPy es la librería principal para manipulación de datos numéricos en Python.
- Matplotlib trabaja perfectamente con sus arrays (numpy.ndarray)

In [31]:
```python
import numpy as np
import matplotlib.pyplot as plt
```

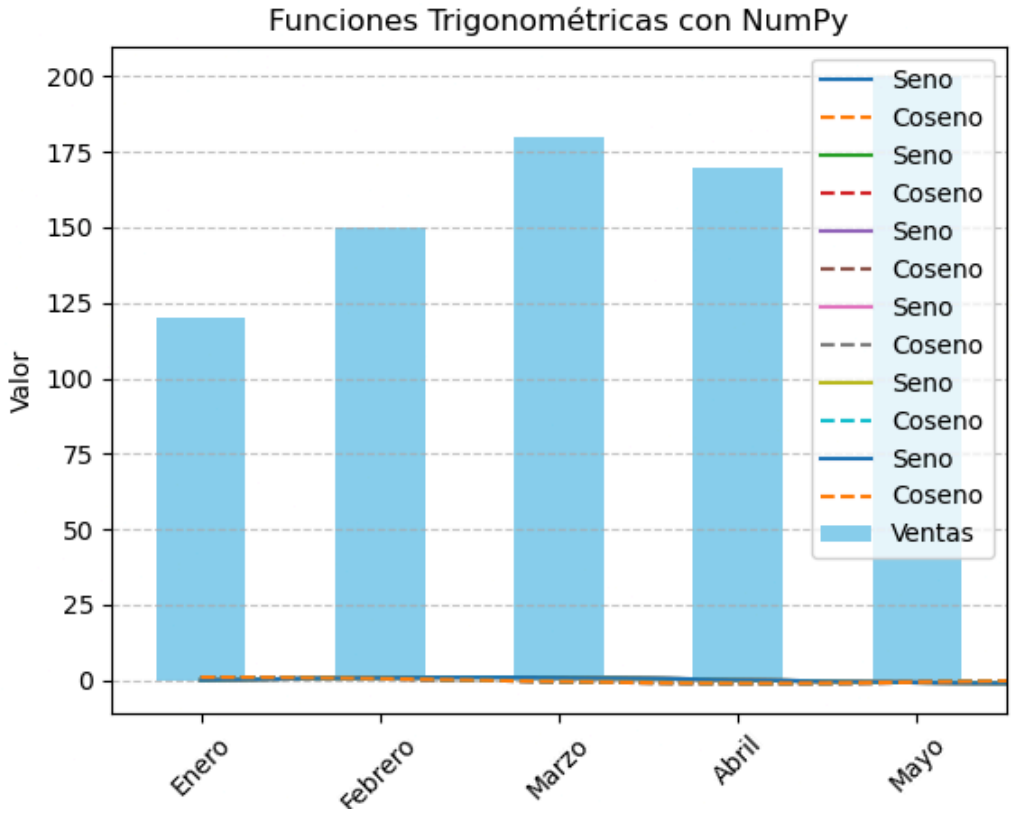
In [60]:
```python
# Crear datos con NumPy
x = np.linspace(0, 10, 100)  # 100 valores entre 0 y 10
y = np.sin(x)  # Función seno
y2 = np.cos(x)  # Función coseno
figsize=(4,4)
# Crear gráfico
plt.plot(x, y, label="Seno")
plt.plot(x, y2, label="Coseno", linestyle="--")

plt.xlabel("Tiempo")
plt.ylabel("Valor")
plt.title("Funciones Trigonométricas con NumPy")
plt.legend()
plt.grid()
plt.show()
```



Uso de Pandas con Matplotlib

- Pandas nos ayuda a manejar datasets estructurados con DataFrames. Matplotlib se integra bien con Pandas para graficar columnas directamente.

In [15]:
```
import pandas as pd
import matplotlib.pyplot as plt


data = {
    "Mes": ["Enero", "Febrero", "Marzo", "Abril", "Mayo"],
    "Ventas": [120, 150, 180, 170, 200]
}
df = pd.DataFrame(data)


df.plot(x="Mes", y="Ventas", kind="bar", color="skyblue", legend=False)

plt.title("Ventas Mensuales")
plt.xlabel("Mes")
plt.ylabel("Ventas")
plt.xticks(rotation=45)
plt.grid(axis="y", linestyle="--", alpha=0.7)

plt.show()
```



Ejemplos practicos con Matplotlib

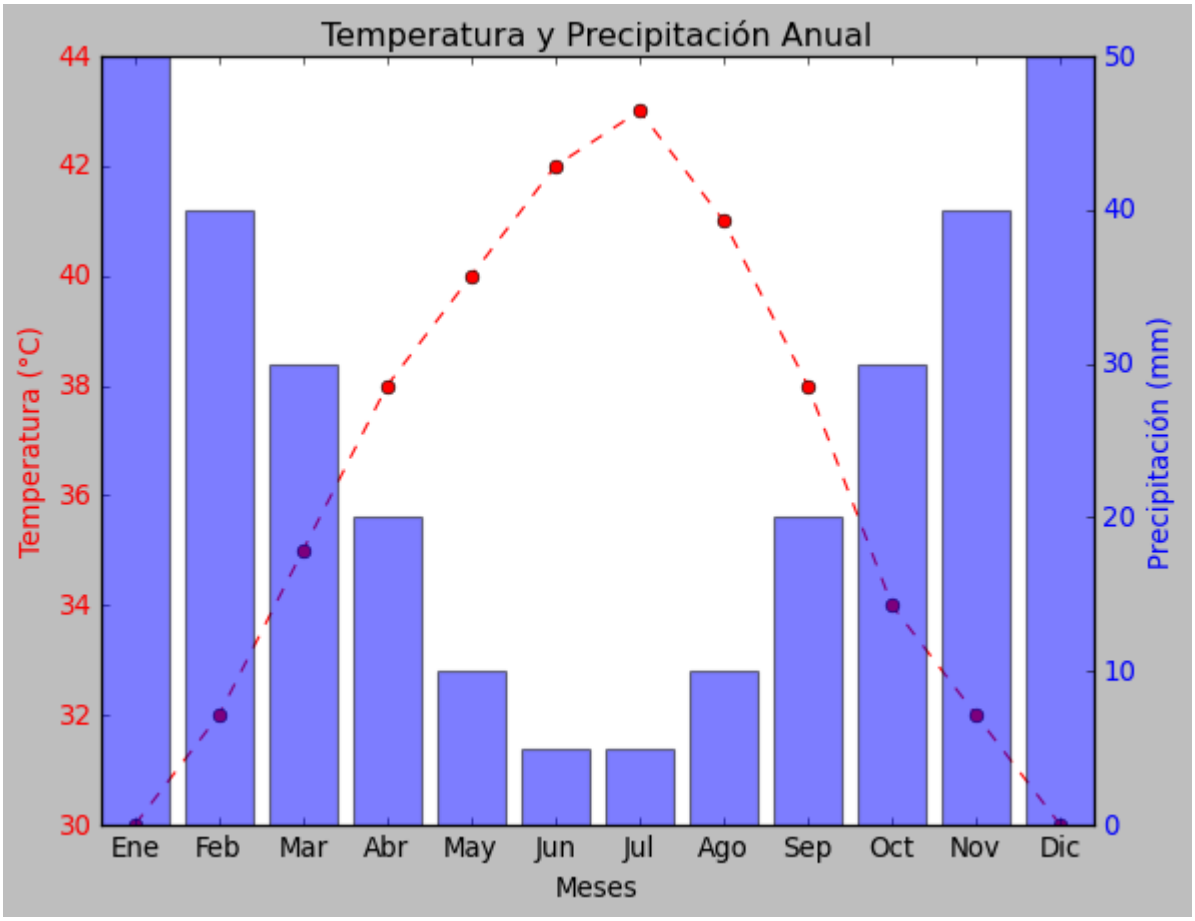
- Ahora aplicaremos todo lo aprendido en ejemplos.
- Empezaremos con analisis de datos climaticos.

Analisis del clima con Matplotlib

- En este ejemplo, analizaremos datos de temperatura y precipitaciones en una ciudad a los largo del año.

In [61]:
```
import numpy as np
import matplotlib.pyplot as plt


meses = ["Ene", "Feb", "Mar", "Abr", "May", "Jun", "Jul", "Ago", "Sep", "Oct", "Nov
temperaturas = [30, 32, 35, 38, 40, 42, 43, 41, 38, 34, 32, 30]  # °C
precipitaciones = [50, 40, 30, 20, 10, 5, 5, 10, 20, 30, 40, 50]  # mm


fig, ax1 = plt.subplots()


ax1.set_xlabel("Meses")
ax1.set_ylabel("Temperatura (°C)", color="red")
ax1.plot(meses, temperaturas, color="red", marker="o", linestyle="--")
ax1.tick_params(axis="y", labelcolor="red")


ax2 = ax1.twinx()
ax2.set_ylabel("Precipitación (mm)", color="blue")
ax2.bar(meses, precipitaciones, color="blue", alpha=0.5)
ax2.tick_params(axis="y", labelcolor="blue")
```

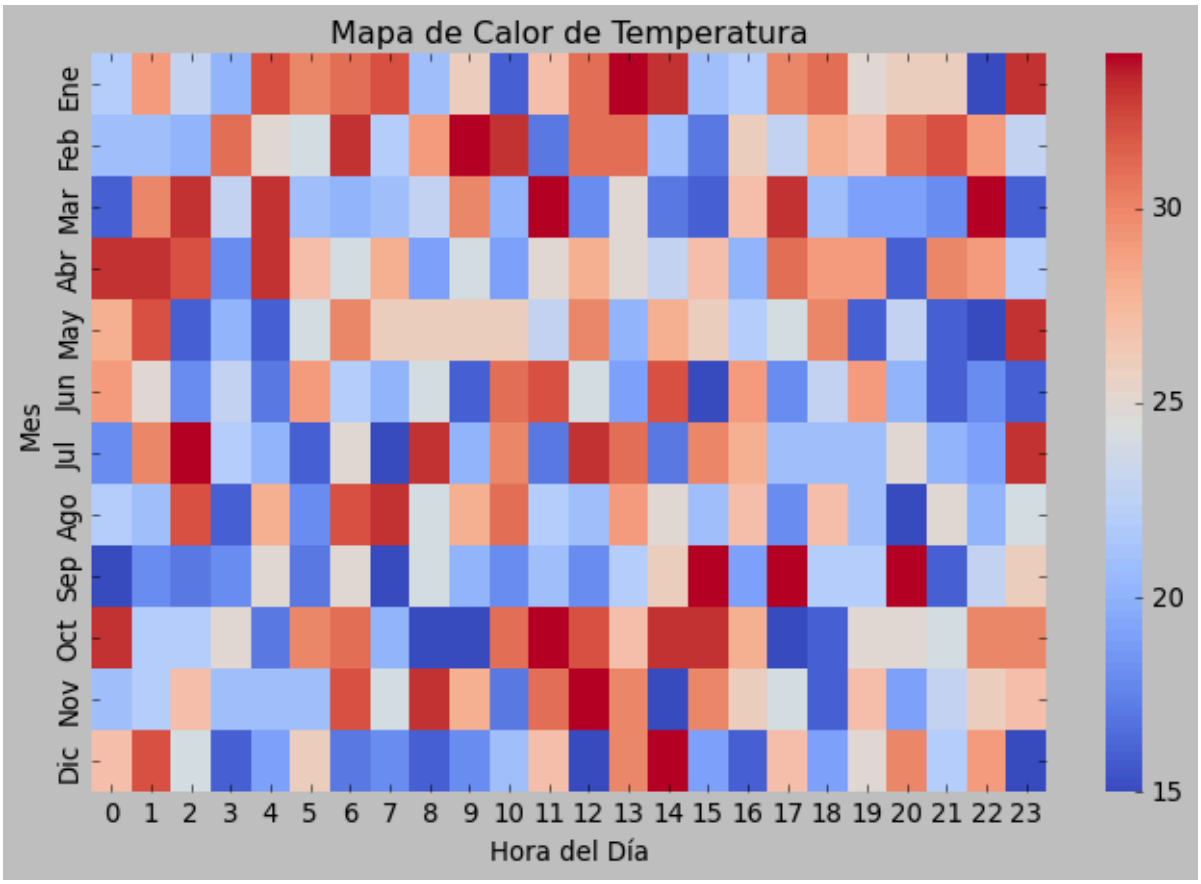
```
plt.title("Temperatura y Precipitación Anual")
plt.show()
```



Mapa de calor de Temperatura

- Podemos usar un mapa de calor par avisualizar variaiones de temperatura por mes y hora.
- Por este ejemplo usamos seaborn, pero mas adelante tendremos Curso especial de seaborn

In [62]:
```
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt


horas = np.arange(0, 24)
temp_mes_hora = np.random.randint(15, 35, (12, 24))


plt.figure(figsize=(10, 6))
sns.heatmap(temp_mes_hora, cmap="coolwarm", annot=False, xticklabels=horas, ytickla

plt.xlabel("Hora del Día")
plt.ylabel("Mes")
plt.title("Mapa de Calor de Temperatura")
plt.show()
```



Por Eduardo Soto ING Software