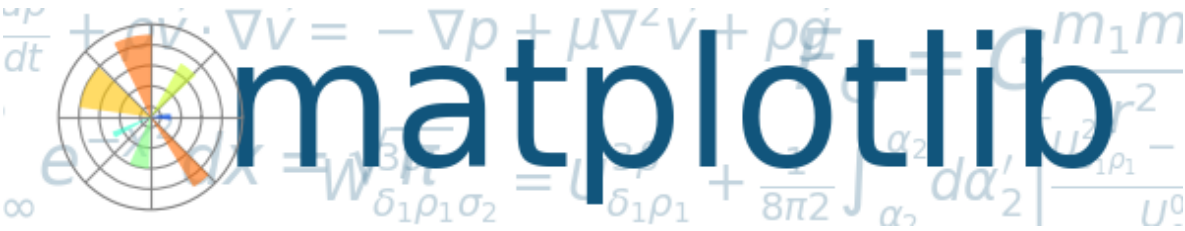
Universidad Castro Carazo
**Estudiante:** Priscilla Corrales Fallas
**Curso:** Análisis de Datos
II cuatrimestre, 2023




## 1.1 ¿Qué es Matplotlib?

Según el sitio web oficial de Matplotlib, esta es una reconocida biblioteca creada por John Hunter en el año 2002, actualmente especializada en la creación de visualizaciones gráficas estáticas, interactivas y animadas en Python. Esta biblioteca permite la creación de gráficos de alta calidad, la creación de figuras interactivas que se pueden ampliar, desplazar y actualizar, permite la personalización del diseño, es exportable a diversos formatos de archivo, además es posible de integrar a entornos como JupyterLab y en Interfaces Gráficas de Usuario.

La versatilidad de esta biblioteca la ha posicionado como una herramienta esencial en la ciencia de datos y la informática científica, sirviendo como base para muchas otras bibliotecas de visualización de datos en Python. Según Akintola (s.f) el uso de Matplotlib presenta ventajas significativas para la comprensión y comunicación, tales como:

1. **Claridad y simplificación:** la visualización simplifica datos complejos en formatos fáciles de entender, resaltando patrones y puntos importantes que podrían no ser tomados en cuenta en los datos sin procesar.
2. **Identificación de patrones y tendencias:** la biblioteca permite identificar patrones y correlaciones en los datos mediante gráficos visuales, facilitando la toma de decisiones informadas basadas en datos.
3. **Exploración y Análisis de Datos:** la interactividad de Matplotlib permite explorar datos de manera detallada, al permitir zoom y desplazamientos en el gráfico.
4. **Comunicación efectiva:** la utilización de visualizaciones permite comunicar hallazgos a audiencias sin amplio conocimiento técnico.
5. **Toma de decisiones estratégicas:** La visualización revela tendencias y valores atípicos que influyen en la toma de decisiones estratégicas para diversos sectores.

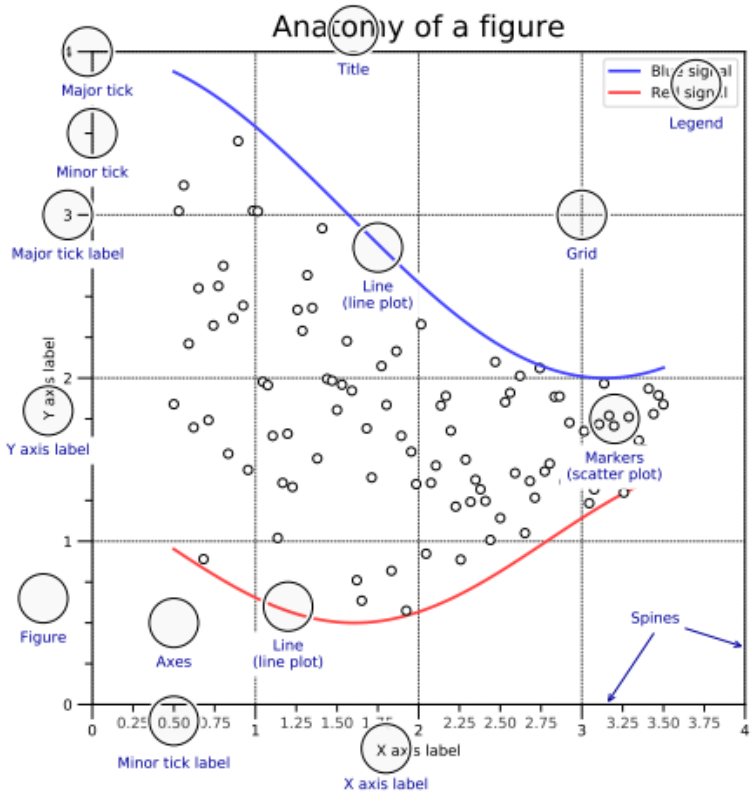
## 1.2 Tipos de gráficos

Matplotlib permite la creación de diferentes tipos de gráficos, entre los que se destacan:

| Básicos | Arreglos y campos | Estadística | Coordenadas no estructuradas | 3D |
|---|---|---|---|---|
| plot(x,y) | Imshow(Z) | hist (x) | tricontour (x, y, z) | 3D scatterplot |
| scatter(x,y) | pcolormesh(X,Y,Z) | boxplot (X) | tricontourf (x, y, z) | 3D surface |
| bar(x,height) | contour (X,Y,Z) | errorbar (x, y, yerr, xerr) | tripcolor (x, y, z) | Triangular 3D surfaces |
| stem(x,y) | contourf (X,Y,Z) | violinplot (D) | triplot (x, y) | 3D voxel / volumetric plot |
| step(x,y) | barbs (X,Y,U,V) | eventplot (D) | | 3D wireframe plot |
| fill_between(x,y1,y2) | quiver (X,Y,U,V) | hist2d (x, y) | | |
| stack(x,y) | streamplot (X,Y,U,V) | hexbin (x, y, C) | | |
| | | pie (x) | | |

Fuente: Elaboración propia con información tomada del sitio web oficial de Matplotlib.

## 1.3 Partes de una figura o gráfico de Matplotlib




Fuente: Nicolas P. Rougier. (2021). Scientific Visualization: Python + Matplotlib. (pág. 4)

1. **Figure:** El elemento más importante de una figura es la figura misma. Se crea cuando llamas al método "figure". Se puede especificar el tamaño, color de fondo (facecolor) y título (subtitle). Si no se quiere ningún color de fondo, se puede especificar "transparent=True" al guardar la figura.
2. **Axes**: Es el segundo elemento más importante, corresponde al área real donde se mostrarán los datos. Puedes tener uno o varios ejes por figura, y cada uno generalmente está rodeado por cuatro bordes (izquierdo, superior, derecho e inferior) llamados "spines". Cada uno de estos "spines" puede tener marcas principales y menores (mayor and minor ticks), etiquetas de las marcas (tick labels) y una etiqueta (label).
3. **Axis**: Los "spines" decorados se llaman ejes. El horizontal es el eje x y el vertical es el eje y. Cada uno está compuesto por un "spine", marcas principales y menores (mayor and minor ticks), etiquetas de las marcas (mayor and minor tick labels) y una etiqueta del eje (axis label)
4. **Spines**: Son las líneas que conectan las marcas de las marcas del eje y marcan los límites del área de los datos. Pueden colocarse en posiciones arbitrarias y pueden ser visibles o invisibles.
5. **Artist**: Todo en la figura, incluyendo la Figura, los Ejes y los objetos del Eje. Esto incluye objetos de texto, objetos Line2D (líneas), objetos de colección, objetos Patch. Cuando se renderiza la figura, todos los “artists” se dibujan en el lienzo.

## 1.4 Creación de gráficos con matplotlib

En el siguiente apartado con la finalidad de profundizar en los pasos, métodos o funciones para la visualización de datos utilizando Matplotlib, siguiendo a Rosidi, N. (2023), se desarrolla un ejemplo con un gráfico de barras apiladas para analizar los datos de los partidos de la copa mundial, como goles, fuera de juego o faltas en diferentes días.

### 1. Importar las bibliotecas

Se importa la biblioteca junto con NumPy y Pandas

```
import pandas as pd
import numpy as np
import matplotlib
```

2. **Definición de datos**

Se definen dos variables. La variable x es una lista de cadenas que especifica el día para el gráfico. La variable df es un DataFrame de pandas que contiene los datos estadísticos del gráfico.

```
x = ["Monday","Tuesday", "Wednesday", "Thursday"]


df = pd.DataFrame({'goals': [18, 22, 19, 14],
                   'offsides': [5, 7, 7, 9],
                   'fouls': [11, 8, 10, 6]})
```

3. **Creación de la Figura y graficado de datos**

Las columnas del DataFrame representan las diferentes series de datos que se trazarán en el gráfico de barras. Luego, el código crea la figura y ejes utilizando la función **plt.subplots()**, y traza los datos utilizando la función **ax.stackplot()** con el argumento de etiquetas. Las etiquetas del eje “x” y del eje “y” para cada serie de datos se pasan como argumentos a la función **stackplot()**.

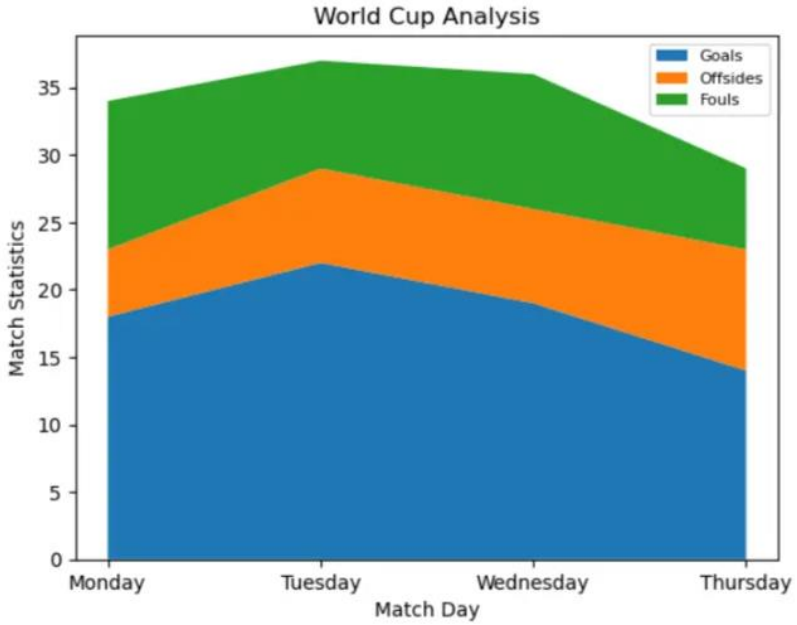
```
# Create a figure and axes
fig, ax = plt.subplots()

# Plot the data
ax.stackplot(x, df['goals'], df['offsides'],df['fouls'], labels=['Goals', 'Offsides', 'Fouls'])
```

4. **Agregar leyenda y mostrar gráfico**

La función **ax.set()** se utiliza para agregar un título y etiquetas de ejes al gráfico. Además la función **legend()** con los argumentos *loc* y *fontsize*, define dónde se coloca la leyenda y cuál es su tamaño. Finalmente, la función **plt.show()** muestra el gráfico.

```
# Add a title and axis labels
ax.set(title="World Cup Analysis", xlabel="Match Day", ylabel="Match Statistics")

ax.legend(loc='upper right',fontsize=8)

# Show the plot
plt.show()
```

A continuación, se presenta el gráfico:




El gráfico de barras apiladas muestra los valores acumulados de las estadísticas de goles, fueras de juego y faltas para cada día de partido. Los valores del eje “y” representan el número total de goles, fueras de juego y faltas y las diferentes series de datos se apilan una sobre otra para mostrar los valores acumulados

## 1.5 Personalización de gráficos con matplotlib

Según Akintola (s.f) Matplotlib ofrece una amplia gama de opciones de personalización que permiten crear gráficos visualmente atractivos y adaptados a las preferencias personales. Algunos métodos o funciones de personalización incluyen:

1. Establecer el Título del Gráfico:
    - **set_title(‘**Título**’)**
      Ejemplo: **ax.set_title('Mi Gráfico')**

2. Establecer Etiquetas de los Ejes:
    - **set_xlabel(‘**etiqueta para el eje x**’)**
    - **set_ylabel(‘**etiqueta para el eje y**’)**
      Ejemplo: **ax.set_xlabel('Eje X')**

3. Establecer Límites de los Ejes:
    - **set_xlim(**‘Límites para el eje x’**)**
    - **set_ylim(**‘Límites para el eje y’**)**
      Ejemplo: **ax.set_xlim(0, 10)**

4. Cambiar Estilo y Color de las Líneas:
    - **plot():** Se pueden pasar parámetros adicionales como *linestyle* y *color* para modificar el estilo de línea y el color del gráfico.
      Ejemplo: **ax.plot(x, y, kind='line', color='b')**

5. Agregar líneas de cuadrícula en el gráfico:
    - **grid()**
      Ejemplo: **ax.grid(True)**

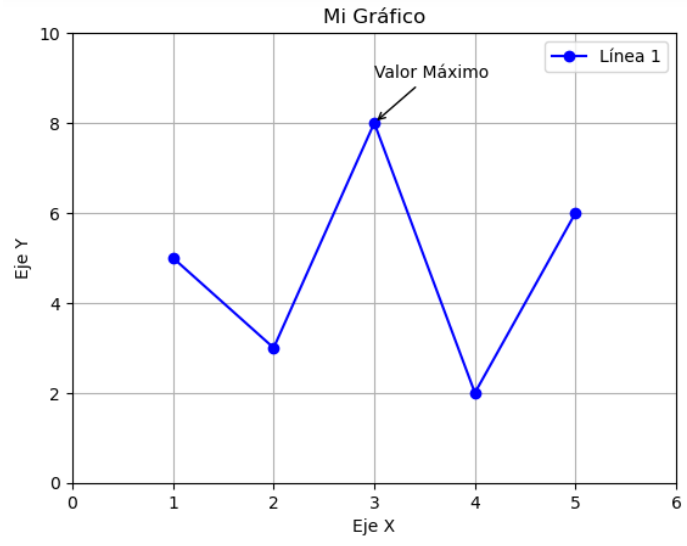
6. Agregar Leyendas, especificando etiquetas para diferentes elementos:
    - **legend()**
      Ejemplo: **ax.legend(['Línea 1', 'Línea 2'])**

7. Cambiar Estilo del Marcador:
    - **plot()**: Se puede pasar el parámetro adicional *marker* para cambiar el estilo del marcador.
      Ejemplo: **ax.plot(x, y, marker='o')**

8. Agregar Anotaciones de texto a puntos específicos en el gráfico:
    - **annotate()**
      Ejemplo: **ax.annotate('Valor Máximo', xy=(3, 6), xytext=(4, 7), arrowprops=dict(arrowstyle='->'))**

9. Cambiar Tamaño de la Figura:
    - **figure()**: Se puede pasar el parámetro adicional *figsize* para especificar el ancho y la altura de la figura.
      Ejemplo: **plt.figure(figsize=(8, 6))**

El siguiente gráfico se llevó a cabo siguiendo la información proporcionada por el autor Akintola (s.f), para llevar a cabo la personalización básica de un gráfico en matplotlib.




<Figure size 800x600 with 0 Axes>

Fuente: Elaboración propia con datos proporcionados por Akintola, S. (s.f) Matplotlib Graphical Functions. Guide to optimize your Visualization skills using Mattplotlib functions.

## 1.6 Referencias bibliográficas


Akintola, S. (s.f) Matplotlib Graphical Functions. Guide to optimize your Visualization skills using Mattplotlib functions.

Matplotlib. (s.f.) Matplotlib: Visualization with Python. Recuperado de https://matplotlib.org/

Nicolas P. Rougier. (2021). Scientific Visualization: Python + Matplotlib. Recuperado de https://inria.hal.science/hal-03427242/document

Rosidi, N. (2023). 4 Python Data Visualization Libraries you can't do without. Recuperado de https://www.stratascratch.com/blog/4-python-data-visualization-libraries-you-can-t-do-without/#:~:text=Matplotlib%2C%20seaborn%2C%20Plotly%2C%20and,them%20with%20our%20code%20examples

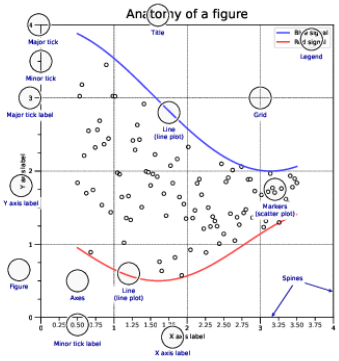
# matplotlib

Cheat sheet — Version 3.5.0

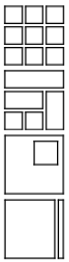
## Quick start

```
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt


X = np.linspace(0, 2*np.pi, 100)
Y = np.cos(X)

fig, ax = plt.subplots()
ax.plot(X, Y, color='green')

fig.savefig("figure.pdf")
plt.show()
```

## Anatomy of a figure




## Subplots layout



```
subplot[s](rows,cols,…)
fig, axs = plt.subplots(3, 3)

G = gridspec(rows,cols,…)
ax = G[0,:]

ax.inset_axes(extent)

d=make_axes_locatable(ax)
ax = d.new_horizontal('10%')
```

## Getting help

- matplotlib.org
- github.com/matplotlib/matplotlib/issues
- discourse.matplotlib.org
- stackoverflow.com/questions/tagged/matplotlib
- https://gitter.im/matplotlib/matplotlib
- twitter.com/matplotlib
- Matplotlib users mailing list

## Basic plots

```
plot([X],Y,[fmt],…)
```
X, **Y**, fmt, color, marker, linestyle

```
scatter(X,Y,…)
```
**X**, **Y**, [s]izes, [c]olors, marker, cmap

```
bar[h](x,height,…)
```
**x**, **height**, width, bottom, align, color

```
imshow(Z,…)
```
**Z**, cmap, interpolation, extent, origin

```
contour[f]([X],[Y],Z,…)
```
X, Y, **Z**, levels, colors, extent, origin

```
pcolormesh([X],[Y],Z,…)
```
X, Y, **Z**, vmin, vmax, cmap

```
quiver([X],[Y],U,V,…)
```
X, Y, **U**, **V**, C, units, angles

```
pie(X,…)
```
**Z**, explode, labels, colors, radius


```
text(x,y,text,…)
```
**x**, **y**, **text**, va, ha, size, weight, transform

```
fill[_between][x](…)
```
**X**, Y1, Y2, color, where

## Advanced plots

```
step(X,Y,[fmt],…)
```
**X**, **Y**, fmt, color, marker, where

```
boxplot(X,…)
```
**X**, notch, sym, bootstrap, widths

```
errorbar(X,Y,xerr,yerr,…)
```
**X**, **Y**, xerr, yerr, fmt

```
hist(X, bins, …)
```
**X**, bins, range, density, weights

```
violinplot(D,…)
```
**D**, positions, widths, vert

```
barbs([X],[Y], U, V, …)
```
X, Y, **U**, **V**, C, length, pivot, sizes

```
eventplot(positions,…)
```
**positions**, orientation, lineoffsets

```
hexbin(X,Y,C,…)
```
**X**, **Y**, C, gridsize, bins

## Scales

```
ax.set_[xy]scale(scale,…)
```

linear — any values

log — values > 0

symlog — any values

logit — 0 < values < 1

## Projections

```
subplot(…,projection=p)
```

p='polar'

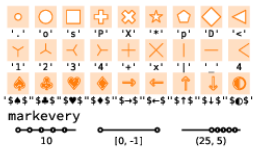
p='3d'

p=ccrs.Orthographic()
import cartopy.crs as ccrs

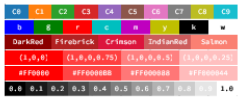
## Lines

```
linestyle or ls
"-"  ":"  "--"  "-."  (0,(0.01,2))

capstyle or dash_capstyle
"butt"  "round"  "projecting"
```

## Markers

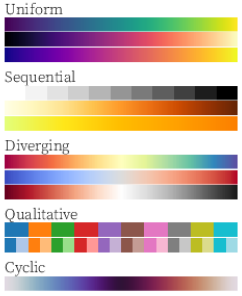



## Colors



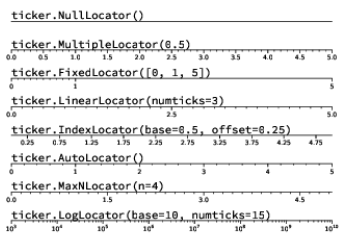

## Colormaps

```
plt.get_cmap(name)
```



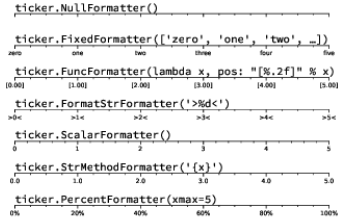

## Tick locators

```
from matplotlib import ticker
ax.[xy]axis.set_[minor|major]_locator(locator)
```



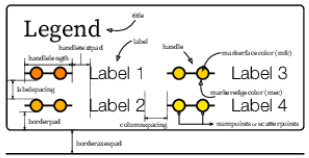

## Tick formatters

```
from matplotlib import ticker
ax.[xy]axis.set_[minor|major]_formatter(formatter)
```

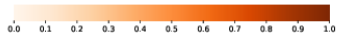



## Ornaments

```
ax.legend(…)
```
handles, labels, loc, title, frameon




```
ax.colorbar(…)
```
mappable, ax, cax, orientation



```
ax.annotate(…)
```
**text**, **xy**, **xytext**, xycoords, textcoords, arrowprops


## Event handling

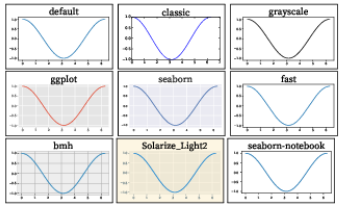
```
fig, ax = plt.subplots()
def on_click(event):
  print(event)
fig.canvas.mpl_connect(
  'button_press_event', on_click)
```

## Animation

```
import matplotlib.animation as mpla

T = np.linspace(0, 2*np.pi, 100)
S = np.sin(T)
line, = plt.plot(T, S)
def animate(i):
    line.set_ydata(np.sin(T+i/50))
anim = mpla.FuncAnimation(
    plt.gcf(), animate, interval=5)
plt.show()
```

## Styles

```
plt.style.use(style)
```




## Quick reminder

```
ax.grid()
ax.set_[xy]lim(vmin, vmax)
ax.set_[xy]label(label)
ax.set_[xy]ticks(ticks, [labels])
ax.set_[xy]ticklabels(labels)
ax.set_title(title)
ax.tick_params(width=10, …)
ax.set_axis_[on|off]()

fig.suptitle(title)
fig.tight_layout()
plt.gcf(), plt.gca()
mpl.rc('axes', linewidth=1, …)
[fig|ax].patch.set_alpha(0)
text=r'$\frac{-e^{i\pi}}{2^n}$'
```

## Keyboard shortcuts

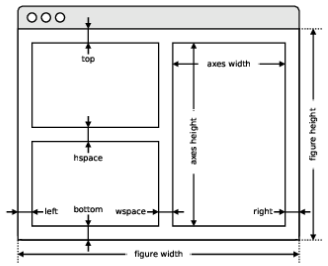
| | | | |
|---|---|---|---|
| ctrl + s | Save | ctrl + w | Close plot |
| r | Reset view | f | Fullscreen 0/1 |
| f | View forward | b | View back |
| p | Pan view | o | Zoom to rect |
| x | X pan/zoom | y | Y pan/zoom |
| g | Minor grid 0/1 | G | Major grid 0/1 |
| l | X axis log/linear | L | Y axis log/linear |

## Ten simple rules

1. Know your audience
2. Identify your message
3. Adapt the figure
4. Captions are not optional
5. Do not trust the defaults
6. Use color effectively
7. Do not mislead the reader
8. Avoid "chartjunk"
9. Message trumps beauty
10. Get the right tool

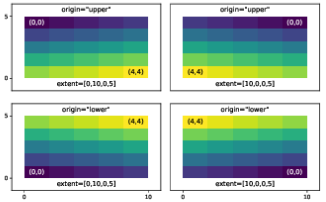
## Axes adjustments

plt.**subplots_adjust**( ... )



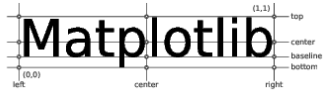

## Extent & origin

ax.**imshow**( extent=..., origin=... )




## Text alignments

ax.**text**( ..., ha=... , va=..., ...)




## Text parameters

ax.**text**(..., family=..., size=..., weight=...)
ax.**text**(..., fontproperties=...)

The quick brown fox — xx-large (1.73)
The quick brown fox — x-large (1.44)
The quick brown fox — large (1.20)
The quick brown fox — medium (1.00)
The quick brown fox — small (0.83)
The quick brown fox — x-small (0.69)
The quick brown fox — xx-small (0.58)

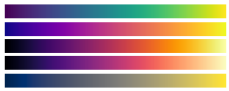
**The quick brown fox jumps over the lazy dog** — black (900)
**The quick brown fox jumps over the lazy dog** — bold (700)
**The quick brown fox jumps over the lazy dog** — semibold (600)
The quick brown fox jumps over the lazy dog — normal (400)
The quick brown fox jumps over the lazy dog — ultralight (100)

The quick brown fox jumps over the lazy dog — monospace
The quick brown fox jumps over the lazy dog — serif
The quick brown fox jumps over the lazy dog — sans
*The quick brown fox jumps over the lazy dog* — cursive

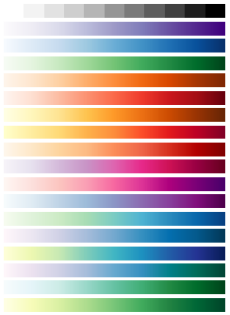
*The quick brown fox jumps over the lazy dog* — italic
The quick brown fox jumps over the lazy dog — normal

THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG — small-caps
The quick brown fox jumps over the lazy dog — normal

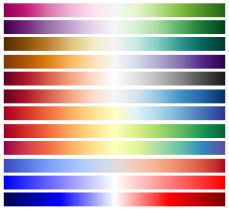
## Uniform colormaps



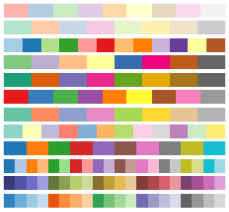

## Sequential colormaps



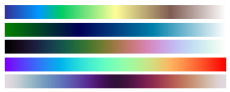

## Diverging colormaps




## Qualitative colormaps

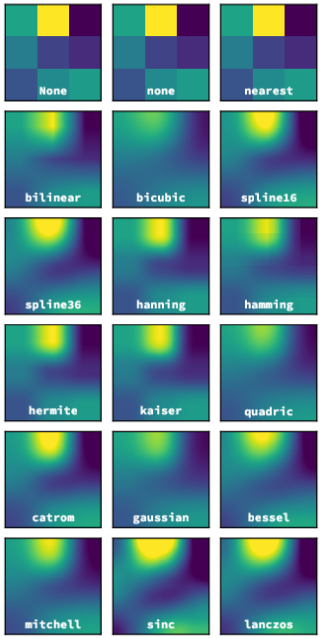



## Miscellaneous colormaps

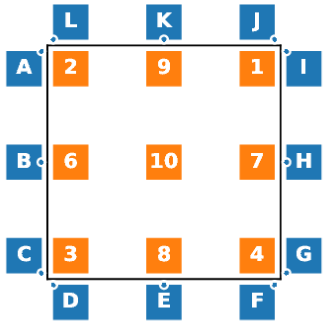



## Color names


## Image interpolation




## Legend placement

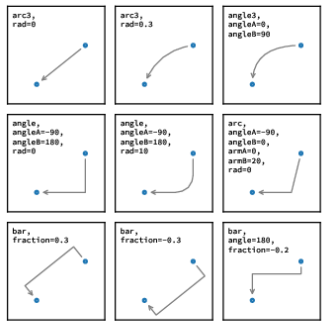



ax.**legend**(loc="string", bbox_to_anchor=(x,y))

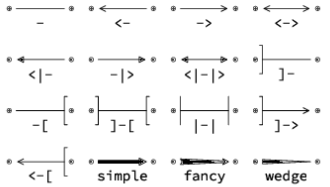
2: upper left — 9: upper center — 1: upper right
6: center left — 10: center — 7: center right
3: lower left — 8: lower center — 4: lower right

A: upper right / (−0.1,0.9) — B: center right / (−0.1,0.5)
C: lower right / (−0.1,0.1) — D: upper left / (0.1,−0.1)
E: upper center / (0.5,−0.1) — F: upper right / (0.9,−0.1)
G: lower left / (1.1,0.1) — H: center left / (1.1,0.5)
I: upper left / (1.1,0.9) — J: lower right / (0.9,1.1)
K: lower center / (0.5,1.1) — L: lower left / (0.1,1.1)

## Annotation connection styles




## Annotation arrow styles




## How do I ...

... resize a figure?
→ fig.set_size_inches(w, h)
... save a figure?
→ fig.savefig("figure.pdf")
... save a transparent figure?
→ fig.savefig("figure.pdf", transparent=True)
... clear a figure/an axes?
→ fig.clear() → ax.clear()
... close all figures?
→ plt.close("all")
... remove ticks?
→ ax.set_[xy]ticks([])
... remove tick labels ?
→ ax.set_[xy]ticklabels([])
... rotate tick labels ?
→ ax.tick_params(axis="x", rotation=90)
... hide top spine?
→ ax.spines['top'].set_visible(False)
... hide legend border?
→ ax.legend(frameon=False)
... show error as shaded region?
→ ax.fill_between(X, Y+error, Y-error)
... draw a rectangle?
→ ax.add_patch(plt.Rectangle((0, 0), 1, 1)
... draw a vertical line?
→ ax.axvline(x=0.5)
... draw outside frame?
→ ax.plot(..., clip_on=False)
... use transparency?
→ ax.plot(..., alpha=0.25)
... convert an RGB image into a gray image?
→ gray = 0.2989*R + 0.5870*G + 0.1140*B
... set figure background color?
→ fig.patch.set_facecolor("grey")
... get a reversed colormap?
→ plt.get_cmap("viridis_r")
... get a discrete colormap?
→ plt.get_cmap("viridis", 10)
... show a figure for one second?
→ fig.show(block=False), time.sleep(1)

## Performance tips

```
scatter(X, Y)                          slow
plot(X, Y, marker="o", ls="")          fast

for i in range(n): plot(X[i])          slow
plot(sum([x+[None] for x in X],[]))    fast

cla(), imshow(…), canvas.draw()        slow
im.set_data(…), canvas.draw()          fast
```

## Beyond Matplotlib

**Seaborn**: Statistical data visualization
**Cartopy**: Geospatial data processing
**yt**: Volumetric data visualization
**mpld3**: Bringing Matplotlib to the browser
**Datashader**: Large data processing pipeline
**plotnine**: A grammar of graphics for Python



NUMFOCUS
OPEN CODE = BETTER SCIENCE