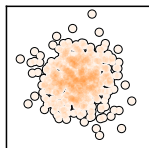


# Matplotlib tips & tricks

## Transparency

Scatter plots can be enhanced by using transparency (alpha) in order to show area with higher density. Multiple scatter plots can be used to delineate a frontier.

```
X = np.random.normal(-1, 1, 500)
Y = np.random.normal(-1, 1, 500)
ax.scatter(X, Y, 50, "0.0", lw=2) # optional
ax.scatter(X, Y, 50, "1.0", lw=0) # optional
ax.scatter(X, Y, 40, "C1", lw=0, alpha=0.1)
```



## Rasterization

If your figure has many graphical elements, such as a huge scatter, you can rasterize them to save memory and keep other elements in vector format.

```
X = np.random.normal(-1, 1, 10_000)
Y = np.random.normal(-1, 1, 10_000)
ax.scatter(X, Y, rasterized=True)
fig.savefig("rasterized-figure.pdf", dpi=600)
```

## Offline rendering

Use the Agg backend to render a figure directly in an array.

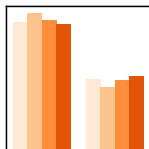
```
from matplotlib.backends.backend_agg import FigureCanvas
canvas = FigureCanvas(Figure())
... # draw some stuff
canvas.draw()
Z = np.array(canvas.renderer.buffer_rgba())
```

## Range of continuous colors

You can use colormap to pick from a range of continuous colors.

```
X = np.random.randn(1000, 4)
cmap = plt.get_cmap("Oranges")
colors = cmap([0.2, 0.4, 0.6, 0.8])
```

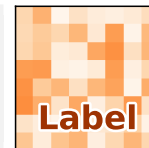
```
ax.hist(X, 2, histtype='bar', color=colors)
```



## Text outline

Use text outline to make text more visible.

```
import matplotlib.patheffects as fx
text = ax.text(0.5, 0.1, "Label")
text.set_path_effects([
    fx.Stroke(linewidth=3, foreground='1.0'),
    fx.Normal([])])
```



## Multiline plot

You can plot several lines at once using *None* as separator.

```
X, Y = [], []
for x in np.linspace(0, 10*np.pi, 100):
    X.extend([x, x, None]), Y.extend([0, sin(x), None])
ax.plot(X, Y, "black")
```



## Dotted lines

To have rounded dotted lines, use a custom linestyle and modify dash\_capstyle.

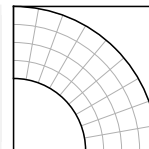
```
ax.plot([0, 1], [0, 0], "C1",
        linestyle=(0, (0.01, 1)), dash_capstyle="round")
ax.plot([0, 1], [1, 1], "C1",
        linestyle=(0, (0.01, 2)), dash_capstyle="round")
```



## Combining axes

You can use overlaid axes with different projections.

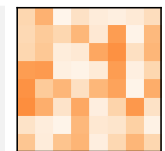
```
ax1 = fig.add_axes([0, 0, 1, 1],
                    label="cartesian")
ax2 = fig.add_axes([0, 0, 1, 1],
                    label="polar",
                    projection="polar")
```



## Colorbar adjustment

You can adjust a colorbar's size when adding it.

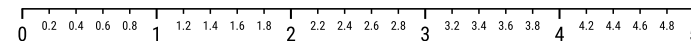
```
im = ax.imshow(Z)
cb = plt.colorbar(im,
                  fraction=0.046, pad=0.04)
cb.set_ticks([])
```



## Taking advantage of typography

You can use a condensed font such as Roboto Condensed to save space on tick labels.

```
for tick in ax.get_xticklabels(which='both'):
    tick.set_fontname("Roboto Condensed")
```



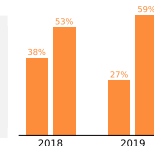
## Getting rid of margins

Once your figure is finished, you can call `tight_layout()` to remove white margins. If there are remaining margins, you can use the `pdfcrop` utility (comes with TeX live).

## Hatching

You can achieve a nice visual effect with thick hatch patterns.

```
cmap = plt.get_cmap("Oranges")
plt.rcParams['hatch.color'] = cmap(0.2)
plt.rcParams['hatch.linewidth'] = 8
ax.bar(X, Y, color=cmap(0.6), hatch="/")
```



## Read the documentation

Matplotlib comes with an extensive documentation explaining the details of each command and is generally accompanied by examples. Together with the huge online gallery, this documentation is a gold-mine.

Matplotlib 3.10.8 handout for tips & tricks. Copyright (c) 2021 Matplotlib Development Team. Released under a CC-BY 4.0 International License. Supported by NumFOCUS.