

Data Visualization

Python and R for Data Science

Data Science and Management



Package `matplotlib`

matplotlib: installation and import

```
In [363]: ! pip install matplotlib pandas numpy
```

```
Defaulting to user installation because normal site-packages is not writeable
```

```
Requirement already satisfied: matplotlib in /home/ercoppa/.local/lib/python3.10/site-packages (3.9.2)
```

```
Requirement already satisfied: pandas in /home/ercoppa/.local/lib/python3.10/site-packages (2.2.2)
```

```
Requirement already satisfied: numpy in /home/ercoppa/.local/lib/python3.10/site-packages (1.26.4)
```

```
Requirement already satisfied: fonttools>=4.22.0 in /usr/lib/python3/dist-packages (from matplotlib) (4.29.1)
```

```
Requirement already satisfied: pyparsing>=2.3.1 in /usr/lib/python3/dist-packages (from matplotlib) (2.4.7)
```

```
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/lib/python3/dist-packages (from matplotlib) (1.3.2)
```

```
Requirement already satisfied: pillow>=8 in /usr/lib/python3/dist-packages (from matplotlib) (9.0.1)
```

```
Requirement already satisfied: packaging>=20.0 in /home/ercoppa/.local/lib/python3.10/site-packages (from matplotlib) (24.1)
```

```
Requirement already satisfied: contourpy>=1.0.1 in /home/ercoppa/.local/lib/python3.10/site-packages (from matplotlib) (1.2.1)
```

```
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (2.9.0.post0)
```

```
Requirement already satisfied: cycycler>=0.10 in /usr/lib/python3/dist-packages (from matplotlib) (0.11.0)
```

```
Requirement already satisfied: pytz>=2020.1 in /usr/lib/python
```

```
3/dist-packages (from pandas) (2022.1)
Requirement already satisfied: tzdata>=2022.7 in /home/ercopp
a/.local/lib/python3.10/site-packages (from pandas) (2024.1)
Requirement already satisfied: six>=1.5 in /usr/lib/python3/dis
t-packages (from python-dateutil>=2.7->matplotlib) (1.16.0)
```

```
In [364]: import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
```

To visualize some data... we need the data

We can use `pandas` to load and process the data. For instance:

```
In [365]: url = "nutrients.csv" # "https://ercoppa.github.io/labds/04/nutrients.c
df = pd.read_csv(url)
df = df.set_index('Food')
df = df.sort_values(by='Carbs', ascending=True)
df
```

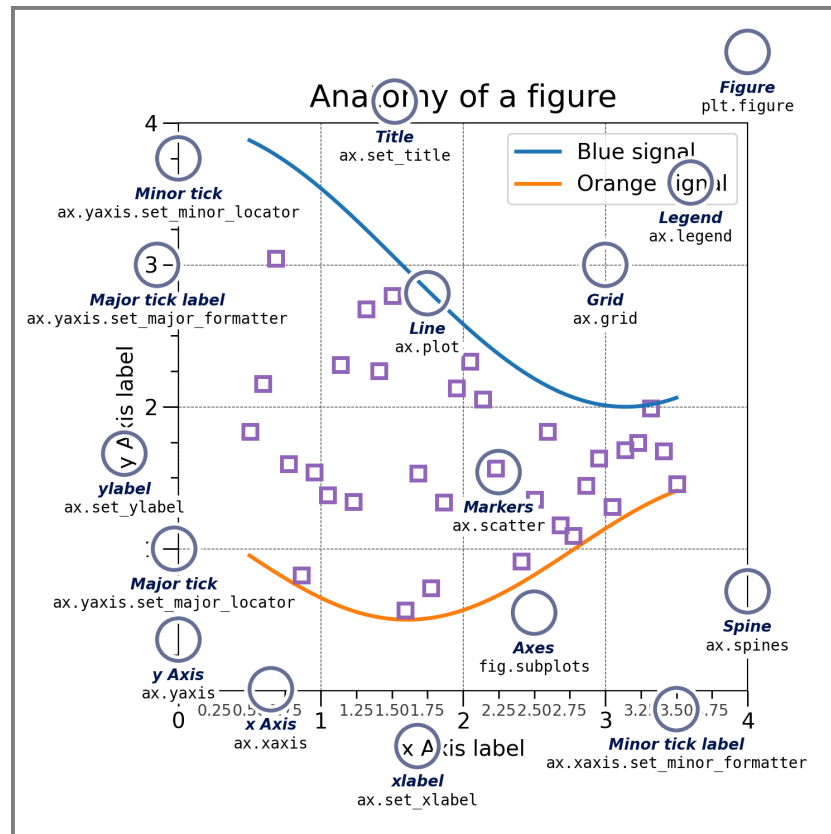
```
Out[365]:
```

	Measure	Grams	Calories	Protein	Fat	Sat.Fat	Fiber
Food							
Bacon	2 slices	16	95	4	8.00	7.00	0.0
Clams	3 oz.	85	87	12	1.00	0.00	0.0
Asparagus	6 spears	96	18	1	0.01	0.01	0.5
Cows' milk	1 qt.	976	660	32	40.00	36.00	0.0
Butter	1/2 cup	112	113	114	115.00	116.00	117.0

Anatomy of a `matplotlib` figure

Example

Key elements



- Figure
- Axes
- Plot type: e.g., line
- Plot title
- Markers
- Grid
- Spine
- Legend
- Axes, {major,minor} ticks
- {x,y}label, {major,minor} tick label

Figure, subplots, and Axes

Figure

A figure contains zero or more subplots (also dubbed Axes in `matplotlib`).

It can be explicitly created with:

```
In [366]: fig = plt.figure()
```

<Figure size 640x480 with 0 Axes>

In most cases, you may omit to explicitly create it since it will be implicitly generated when creating the subplots (see next slides).

More details at: `matplotlib.figure`

Figure size

By default a figure is 640x480 pixels. However, we can set an arbitrary size:

```
In [367]: fig = plt.figure(figsize=(10, 5)) # figsize is (width, height) in inches
<Figure size 1000x500 with 0 Axes>
```

Alternatively, we can set the figure's DPI ("dots-per-inch"):

```
In [368]: fig = plt.figure(dpi=300)
<Figure size 1920x1440 with 0 Axes>
```

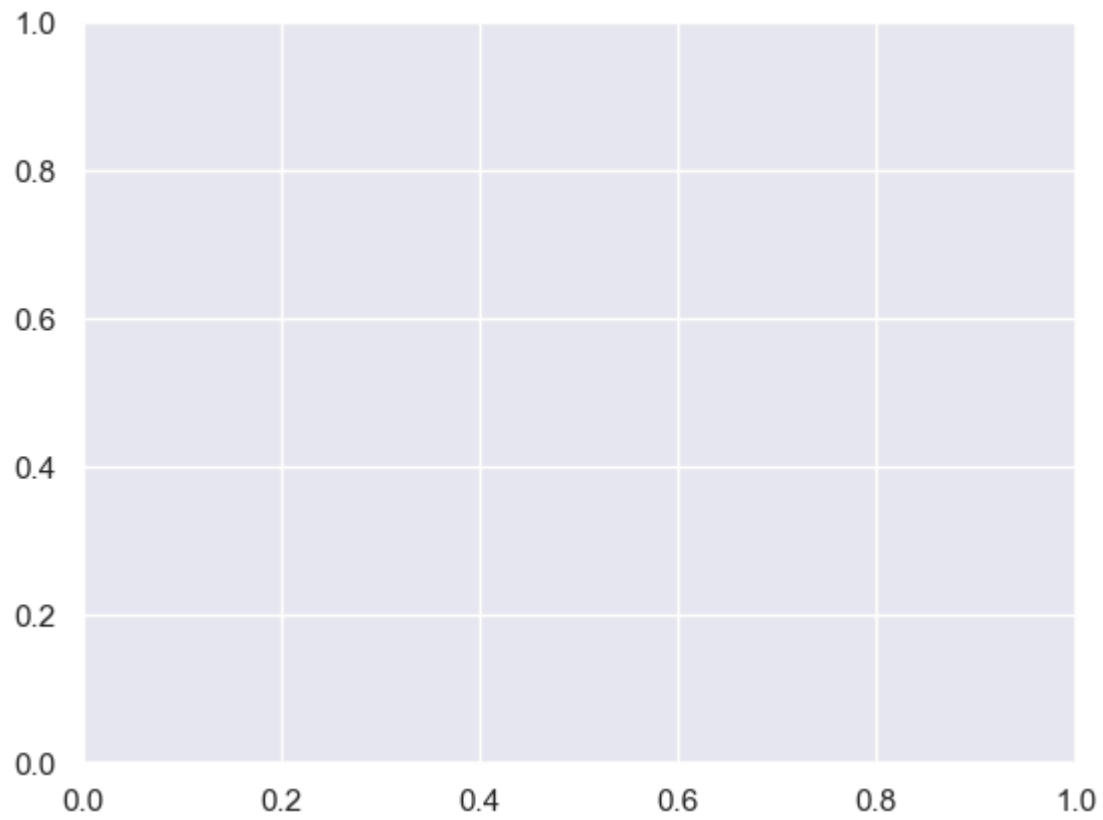
Setting the DPI will preserve the original ratio between width and height.

Subplots and Axes

To create a figure with a **single** subplot (Axes):

```
In [369]: fig, ax = plt.subplots()  
          print(fig)  
          print(ax)
```

```
Figure(640x480)  
Axes(0.125,0.11;0.775x0.77)
```



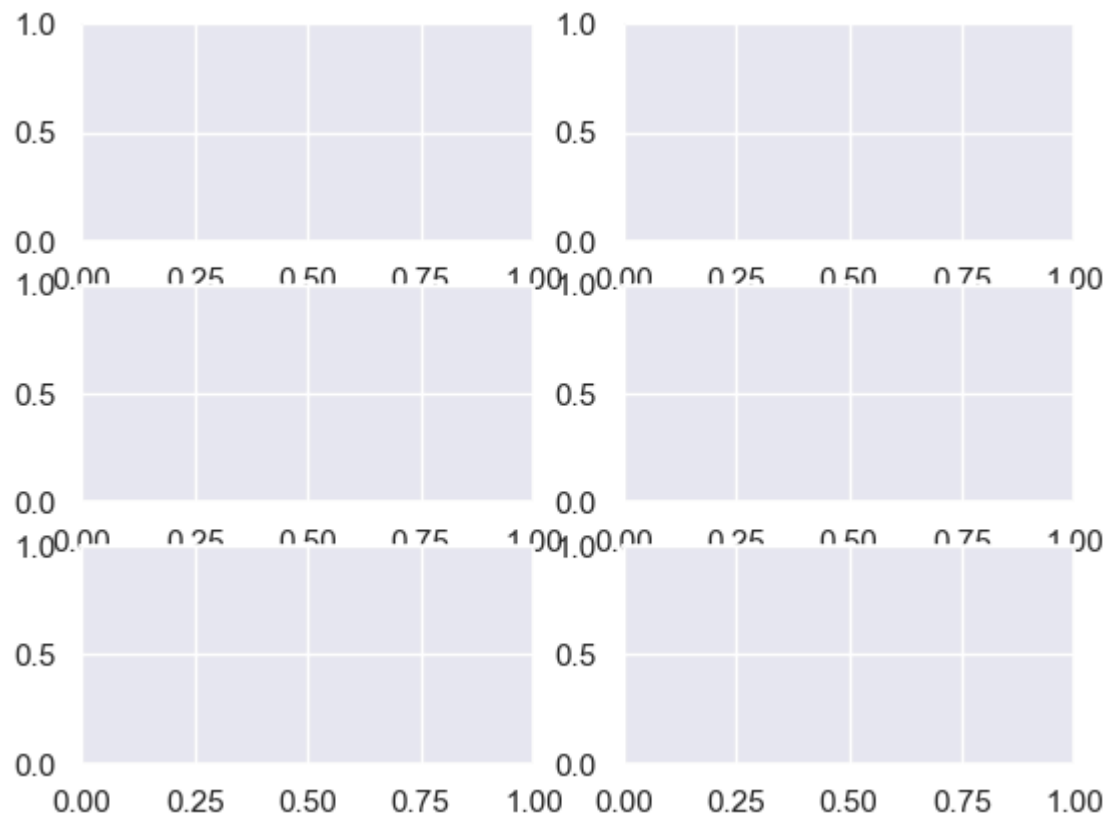
More details at: `matplotlib.pyplot.subplots`

Subplots and Axes (cont'd)

To create a figure with with **NxM** (e.g., N=3, M=2) subplots (**Axes**):

```
In [370]: fig, axs = plt.subplots(3, 2) # 3 rows, 2 columns
          print(fig)
          print(axs)
```

```
Figure(640x480)
[<Axes: > <Axes: >]
[<Axes: > <Axes: >]
[<Axes: > <Axes: >]]
```

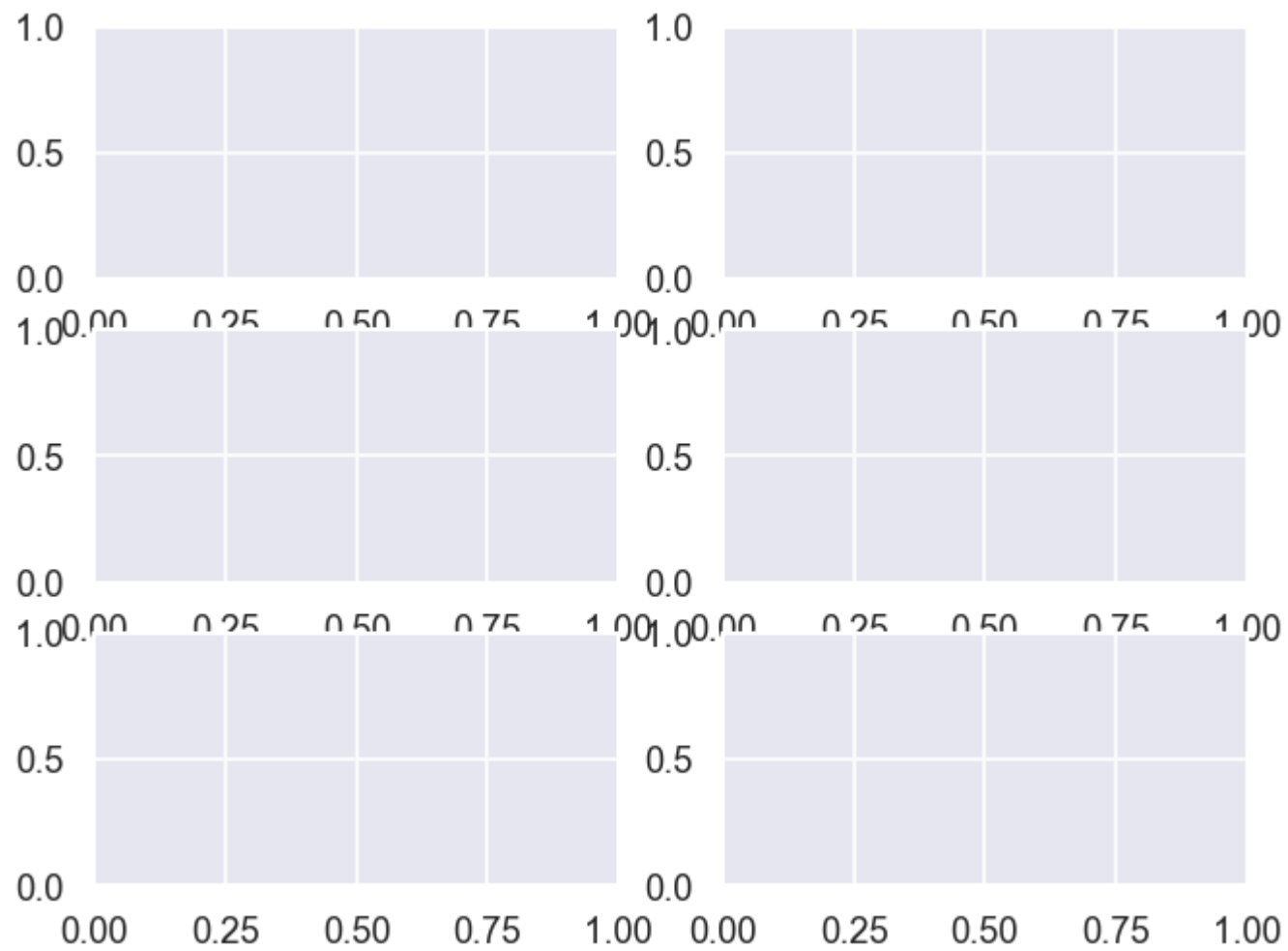


Subplots and Axes: setting the figure size

When creating the figure from `subplots`, we can still set the figure size using the `figsize` or `dpi` arguments. E.g.:

```
In [371]: fig, axs = plt.subplots(3, 2, dpi=120) # 3 rows, 2 columns  
print(fig)
```

Figure(768x576)



Functional interface vs. Object-Oriented interface

When using `matplotlib` we have two approaches:

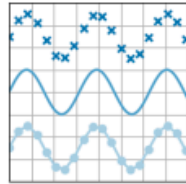
- **Object-Oriented interface (OO):** given an `Axes`, we call methods over that `Axes` to modify its content. E.g., `ax.set_title("Title")`. ***This is the approach that we will adopt.***
- **Functional interface:** we use functions from `pyplot`, which will modify the state of the *current* `Axes`. E.g., `plt.title("Title")`. This approach is problematic, e.g., when we want to deal with multiple plots within the same figure since it is not clear which `Axes` we want to update. This approach was inspired by MATLAB.

For more details, see: <https://matplotlib.org/matplotlibblog/posts/pyplot-vs-object-oriented-interface/>

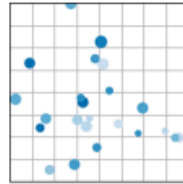
Plot types

Pairwise data

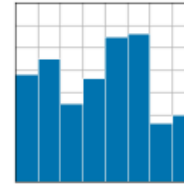
Plots of pairwise (x, y) , tabular (var_0, \dots, var_n) , and functional $f(x) = y$ data.



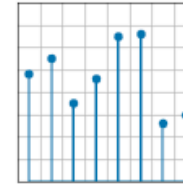
`plot(x, y)`



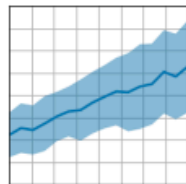
`scatter(x, y)`



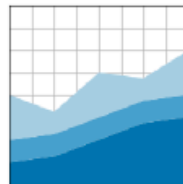
`bar(x, height)`



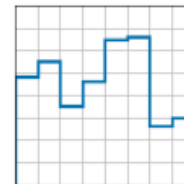
`stem(x, y)`



`fill_between(x, y1,
y2)`



`stackplot(x, y)`

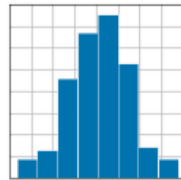


`stairs(values)`

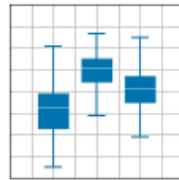
Gallery

Statistical distributions

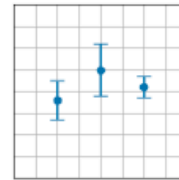
Plots of the distribution of at least one variable in a dataset. Some of these methods also compute the distributions.



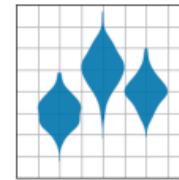
hist(x)



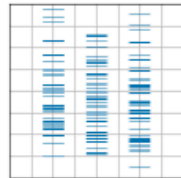
boxplot(X)



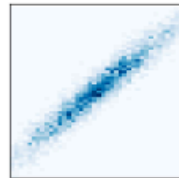
errorbar(x, y, yerr,
xerr)



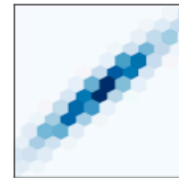
violinplot(D)



eventplot(D)



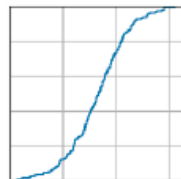
hist2d(x, y)



hexbin(x, y, C)



pie(x)



ecdf(x)

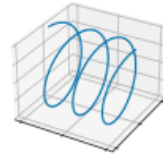
Gallery

3D and volumetric data

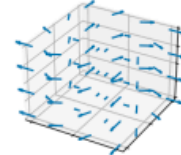
Plots of three-dimensional (x, y, z) , surface $f(x, y) = z$, and volumetric $V_{x,y,z}$ data using the `mpl_toolkits.mplot3d` library.



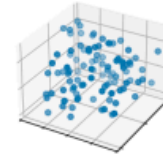
`bar3d(x, y, z, dx, dy, dz)`



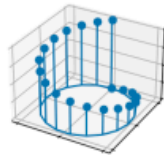
`plot(xs, ys, zs)`



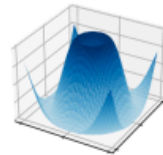
`quiver(X, Y, Z, U, V, W)`



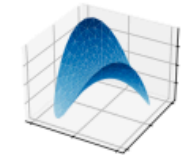
`scatter(xs, ys, zs)`



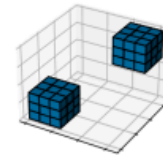
`stem(x, y, z)`



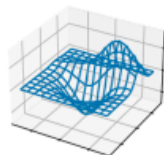
`plot_surface(X, Y, Z)`



`plot_trisurf(x, y, z)`



`voxels([x, y, z], filled)`



`plot_wireframe(X, Y, Z)`

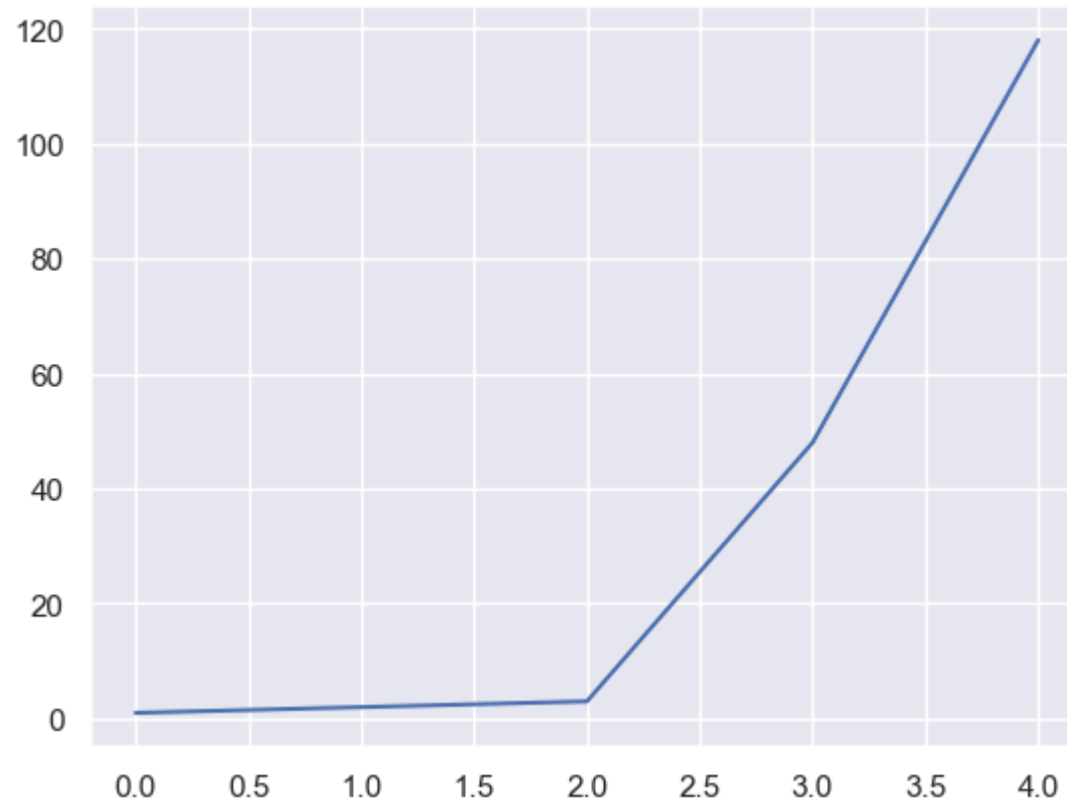
Creating a specific plot type

Given an `Axes`, we can populate it with one or more plots using the methods of `Axes` :

<code>matplotlib.axes.Axes.plot</code>	<code>matplotlib.axes.Axes.csd</code>	<code>matplotlib.axes.Axes.eventplot</code>	<code>matplotlib.axes.Axes.contour</code>
<code>matplotlib.axes.Axes.errorbar</code>	<code>matplotlib.axes.Axes.magnitude_spectrum</code>	<code>matplotlib.axes.Axes.pie</code>	<code>matplotlib.axes.Axes.contourf</code>
<code>matplotlib.axes.Axes.scatter</code>	<code>matplotlib.axes.Axes.phase_spectrum</code>	<code>matplotlib.axes.Axes.stackplot</code>	<code>matplotlib.axes.Axes.imshow</code>
<code>matplotlib.axes.Axes.plot_date</code>	<code>matplotlib.axes.Axes.psd</code>	<code>matplotlib.axes.Axes.broken_barh</code>	<code>matplotlib.axes.Axes.matshow</code>
<code>matplotlib.axes.Axes.step</code>	<code>matplotlib.axes.Axes.spectrogram</code>	<code>matplotlib.axes.Axes.vlines</code>	<code>matplotlib.axes.Axes.pcolor</code>
<code>matplotlib.axes.Axes.loglog</code>	<code>matplotlib.axes.Axes.xcorr</code>	<code>matplotlib.axes.Axes.hlines</code>	<code>matplotlib.axes.Axes.pcolorfast</code>
<code>matplotlib.axes.Axes.semilogx</code>	<code>matplotlib.axes.Axes.ecdf</code>	<code>matplotlib.axes.Axes.fill</code>	<code>matplotlib.axes.Axes.pcolormesh</code>
<code>matplotlib.axes.Axes.semilogy</code>	<code>matplotlib.axes.Axes.boxplot</code>	<code>matplotlib.axes.Axes.axhline</code>	<code>matplotlib.axes.Axes.spy</code>
<code>matplotlib.axes.Axes.fill_between</code>	<code>matplotlib.axes.Axes.violinplot</code>	<code>matplotlib.axes.Axes.axhspan</code>	<code>matplotlib.axes.Axes.tripcolor</code>
<code>matplotlib.axes.Axes.fill_betweenx</code>	<code>matplotlib.axes.Axes.bxp</code>	<code>matplotlib.axes.Axes.axvline</code>	<code>matplotlib.axes.Axes.triplot</code>
<code>matplotlib.axes.Axes.bar</code>	<code>matplotlib.axes.Axes.violin</code>	<code>matplotlib.axes.Axes.axvspan</code>	<code>matplotlib.axes.Axes.tricontour</code>
<code>matplotlib.axes.Axes.barh</code>	<code>matplotlib.axes.Axes.hexbin</code>	<code>matplotlib.axes.Axes.axline</code>	<code>matplotlib.axes.Axes.tricontourf</code>
<code>matplotlib.axes.Axes.bar_label</code>	<code>matplotlib.axes.Axes.hist</code>	<code>matplotlib.axes.Axes.acorr</code>	<code>matplotlib.axes.Axes.annotate</code>
<code>matplotlib.axes.Axes.stem</code>	<code>matplotlib.axes.Axes.hist2d</code>	<code>matplotlib.axes.Axes.angle_spectrum</code>	<code>matplotlib.axes.Axes.text</code>

Line plot

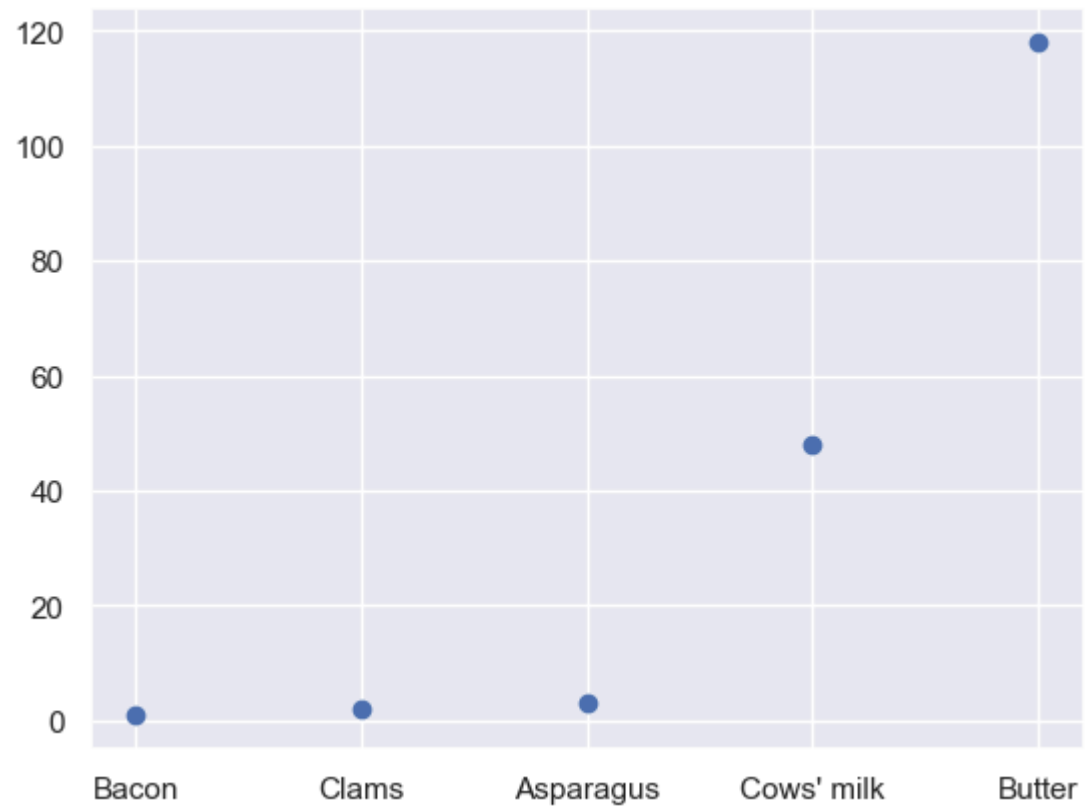
```
In [372]: fig, ax = plt.subplots()
          points = ax.plot(range(df.index.size), df['Carbs'].values)
```



More details at: `matplotlib.axes.Axes.plot`

Scatter plot

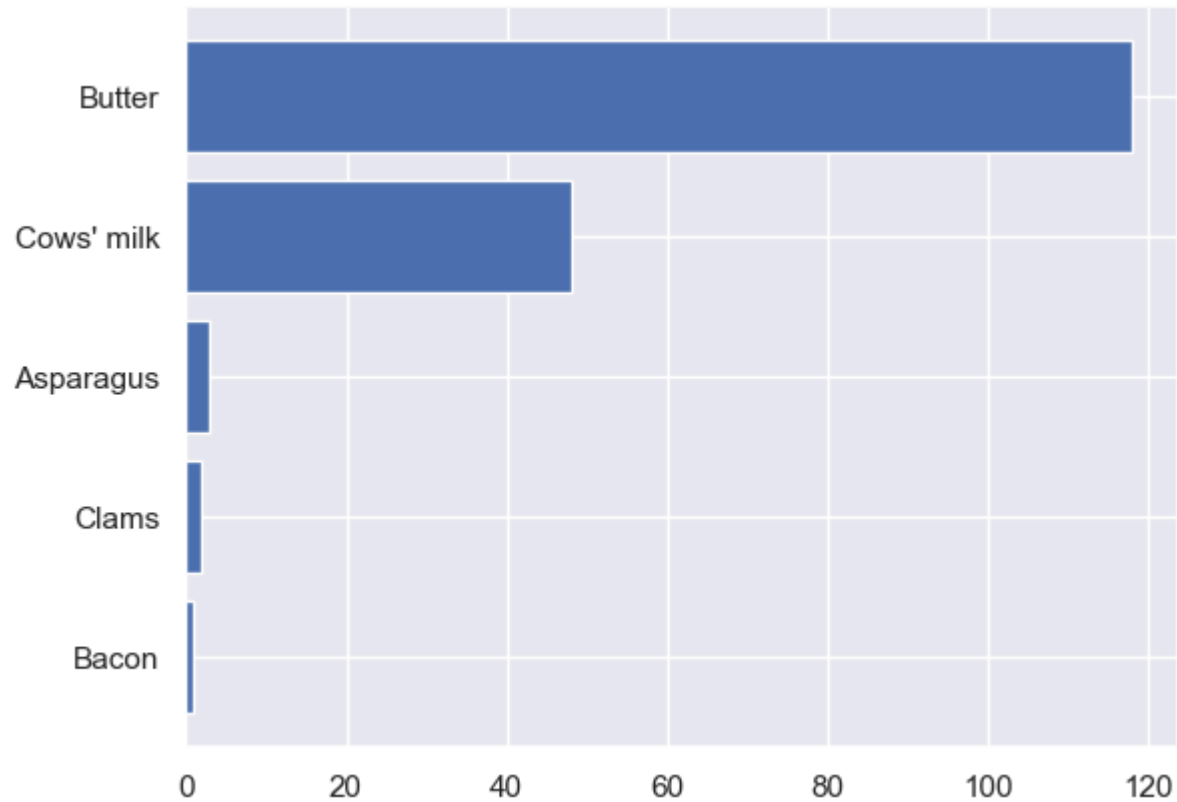
```
In [373]: fig, ax = plt.subplots()  
          points = ax.scatter(df.index, df.Carbs)
```



Mere details at: `matplotlib.axes.Axes.scatter`

Horizontal bar plot

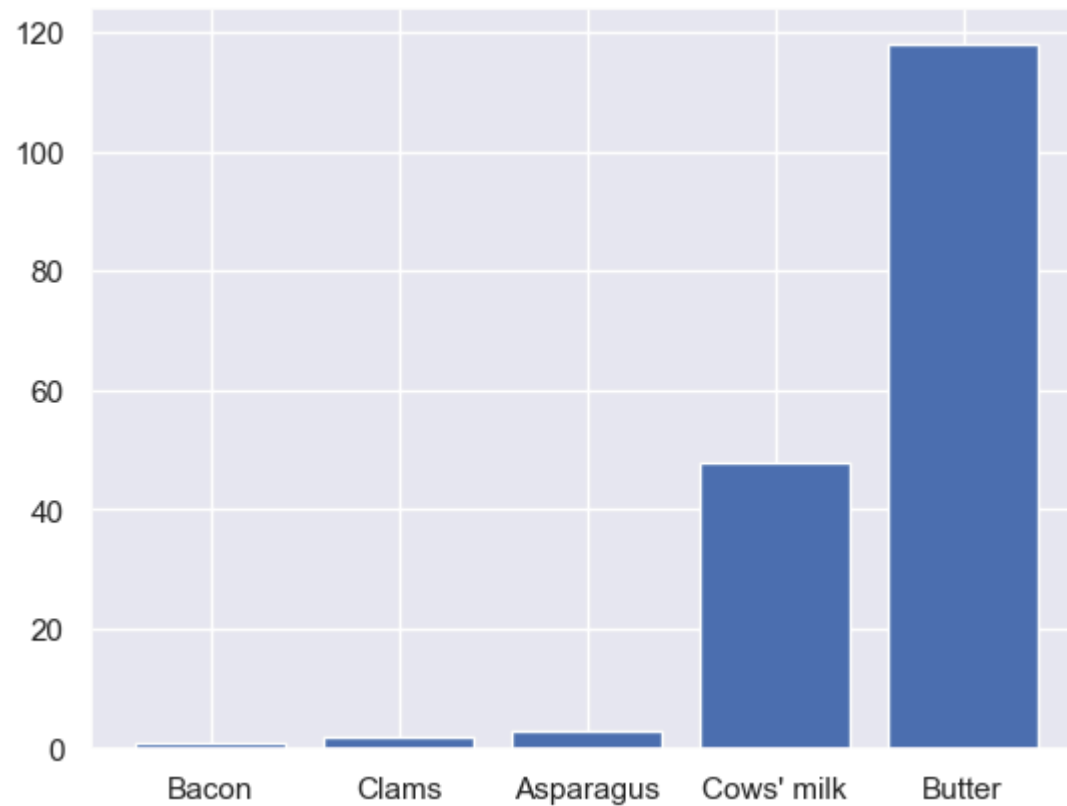
```
In [374]: fig, ax = plt.subplots()
          bars = ax.barh(df.index, df.Carbs)
```



Mere details at: `matplotlib.axes.Axes.barh`

Vertical bar plot

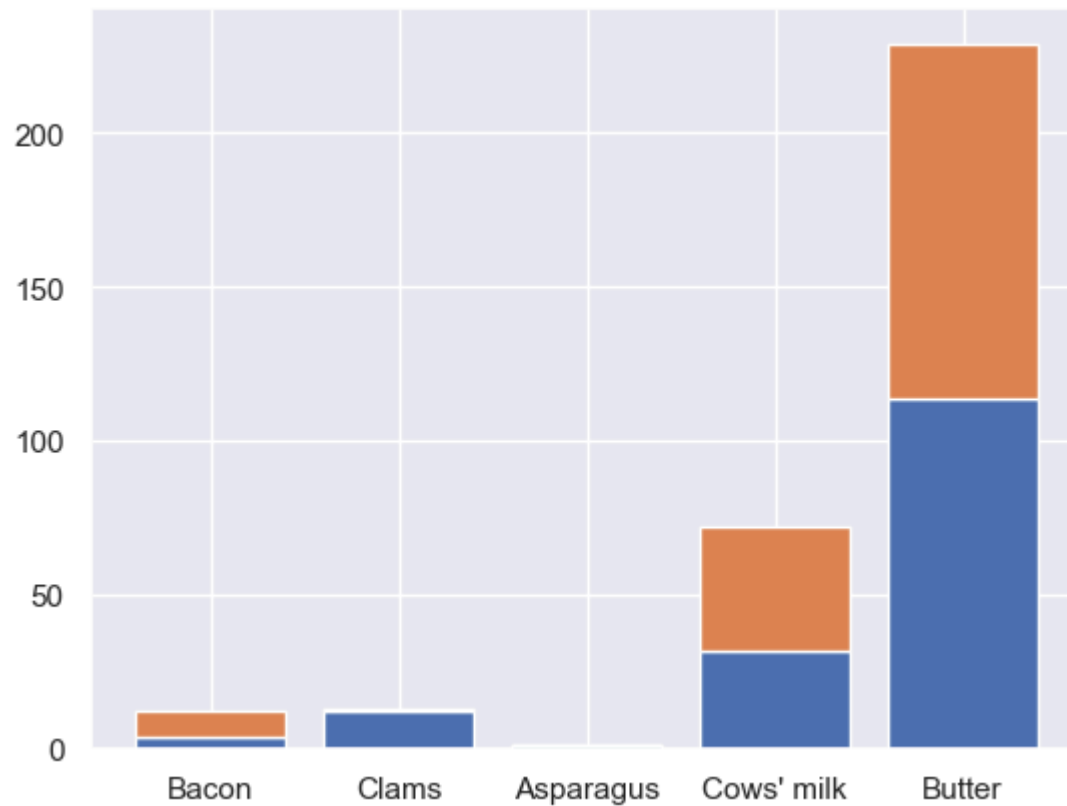
```
In [375]: fig, ax = plt.subplots()  
bars = ax.bar(df.index, df.Carbs)
```



More details at: `matplotlib.axes.Axes.bar`

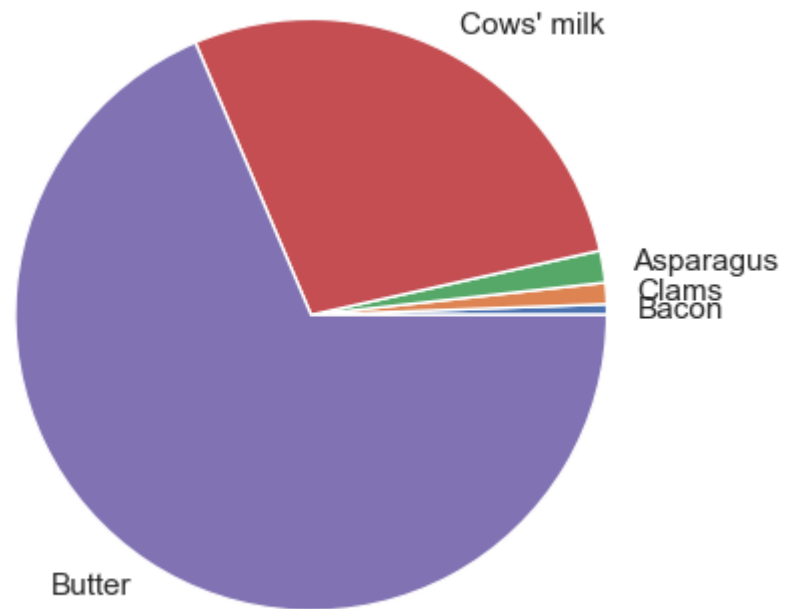
Stacked bar plot

```
In [376]: fig, ax = plt.subplots()
bars = ax.bar(df.index, df["Protein"])
bars = ax.bar(df.index, df["Fat"], bottom=df["Protein"])
```



Pie chart

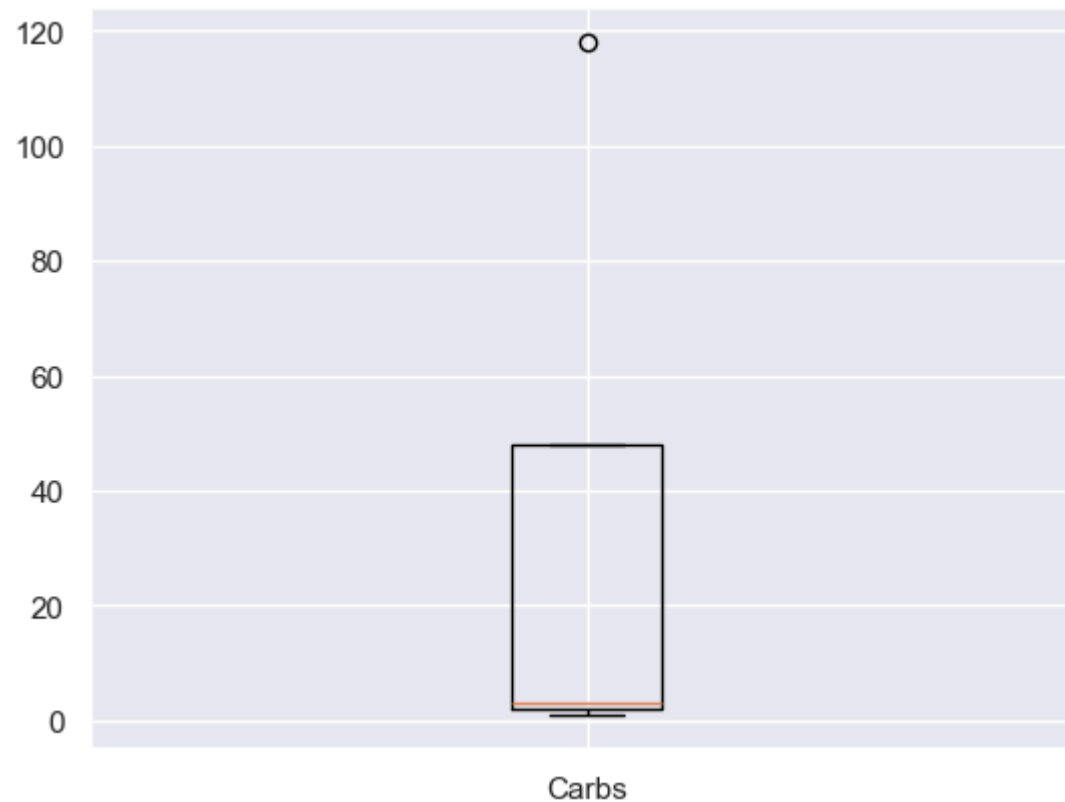
```
In [377]: fig, ax = plt.subplots()
          slices = ax.pie(df.Carbs, labels=df.index)
```



Mere details at: `matplotlib.axes.Axes.pie`

Box plot

```
In [378]: fig, ax = plt.subplots()
          slices = ax.boxplot(df.Carbs, tick_labels=["Carbs"]) # vert=False to make horizontal
```

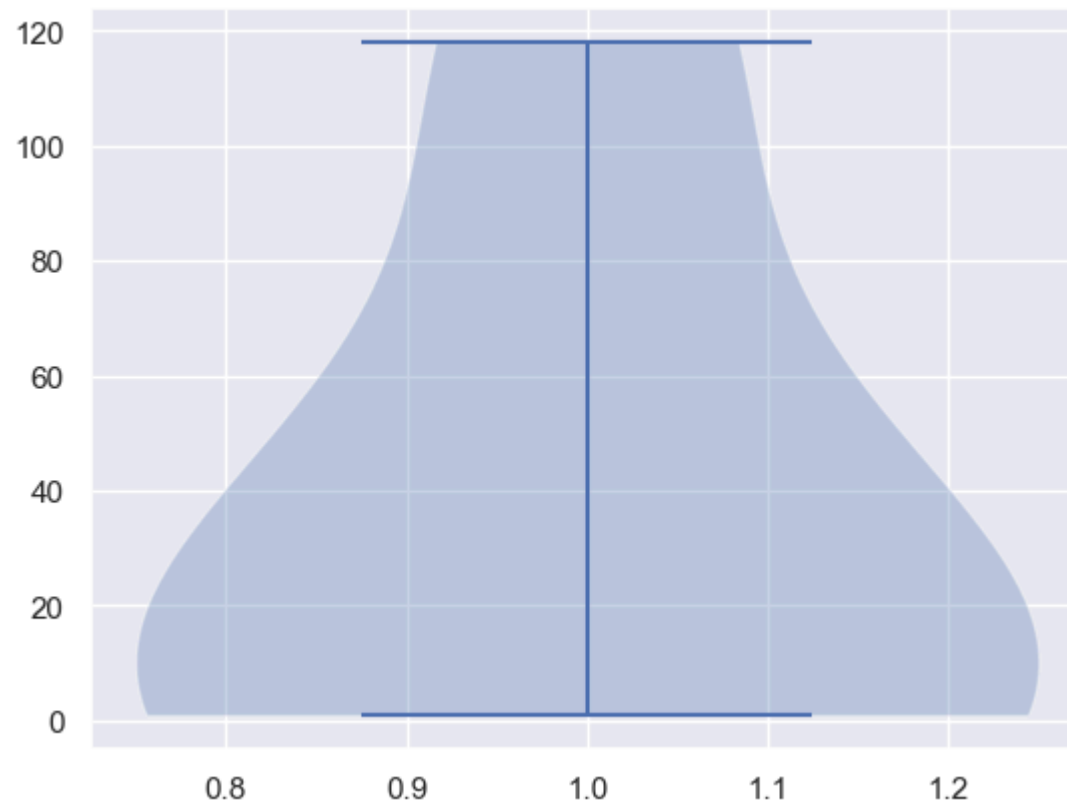


More details at: `matplotlib.axes.Axes.boxplot`

Violin plot

A popular alternative to a boxplot is:

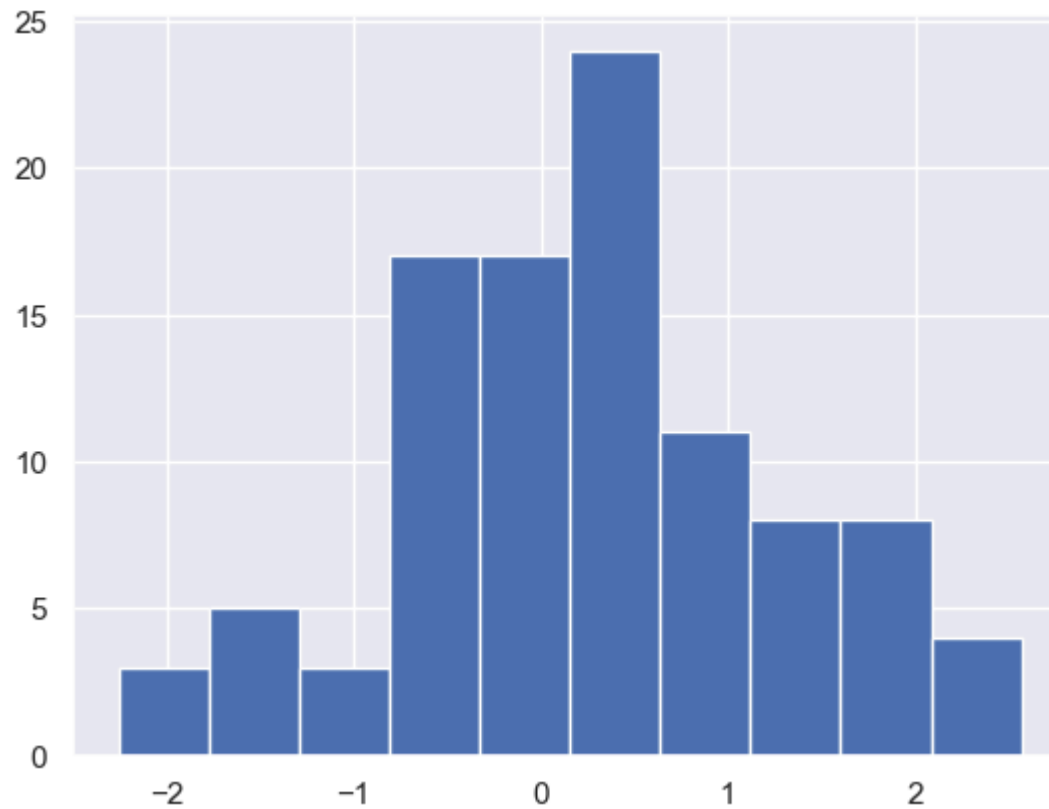
```
In [379]: fig, ax = plt.subplots()
          slices = ax.violinplot(df.Carbs) # vert=False to make it horizontal
```



Mere details at: `matplotlib.axes.Axes.violinplot`

Histogram

```
In [380]: fig, ax = plt.subplots()
data = np.random.standard_normal(100) # 100 random numbers
slices = ax.hist(data, bins=10)
```

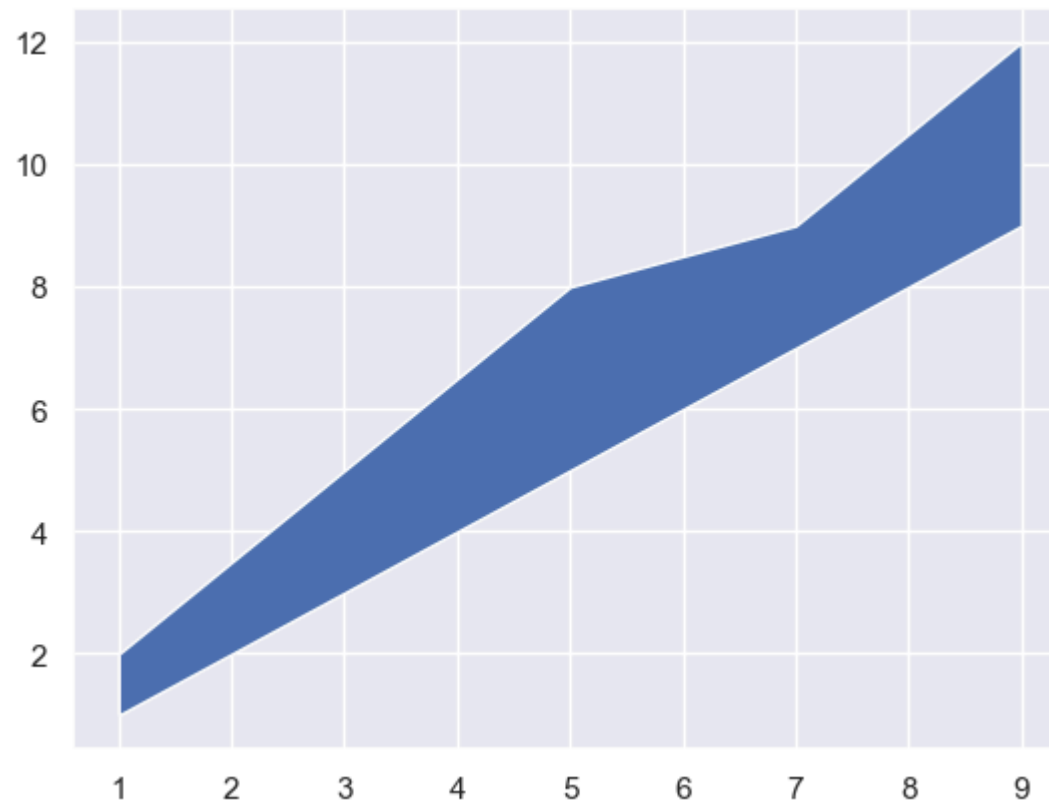


More details at: `matplotlib.axes.Axes.hist`

Fill between

```
In [382]: x = [1, 3, 5, 7, 9]
y1 = [1, 3, 5, 7, 9]
y2 = [2, 5, 8, 9, 12]

fig, ax = plt.subplots()
lines = ax.fill_between(x, y1, y2)
```

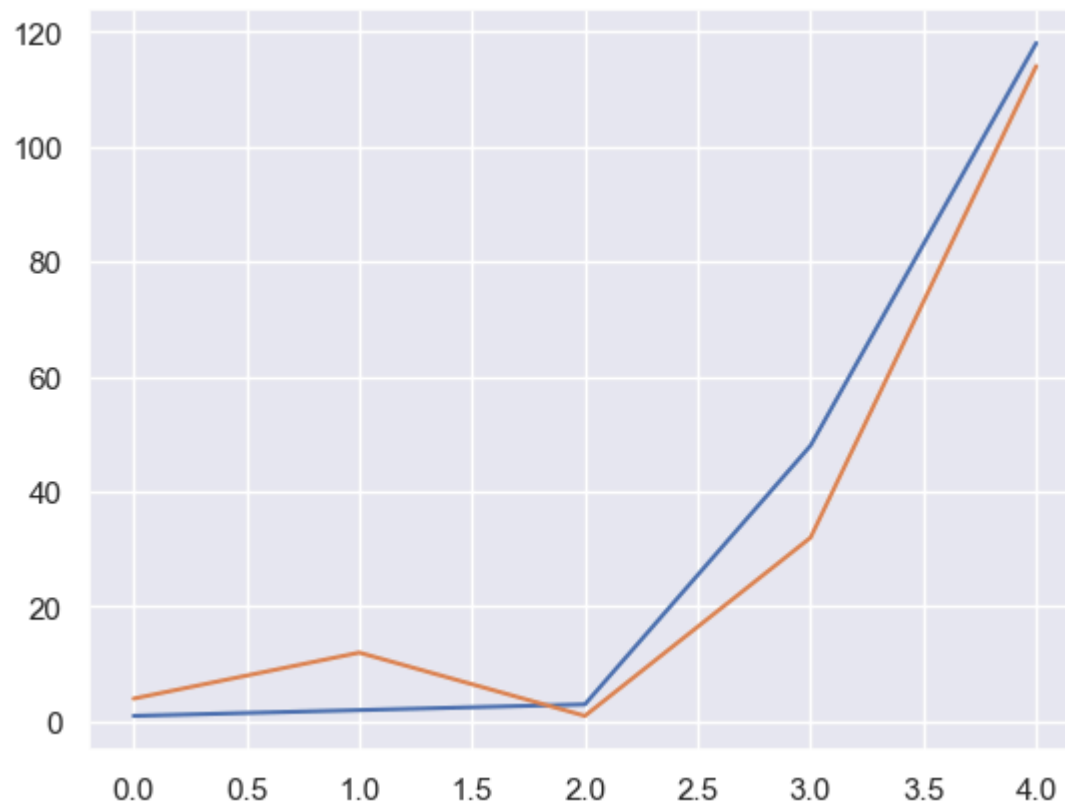


More details at: `matplotlib.axes.Axes.fill_between`

Combining different plot types into an Axes

We can add more than one plot within the same Axes :

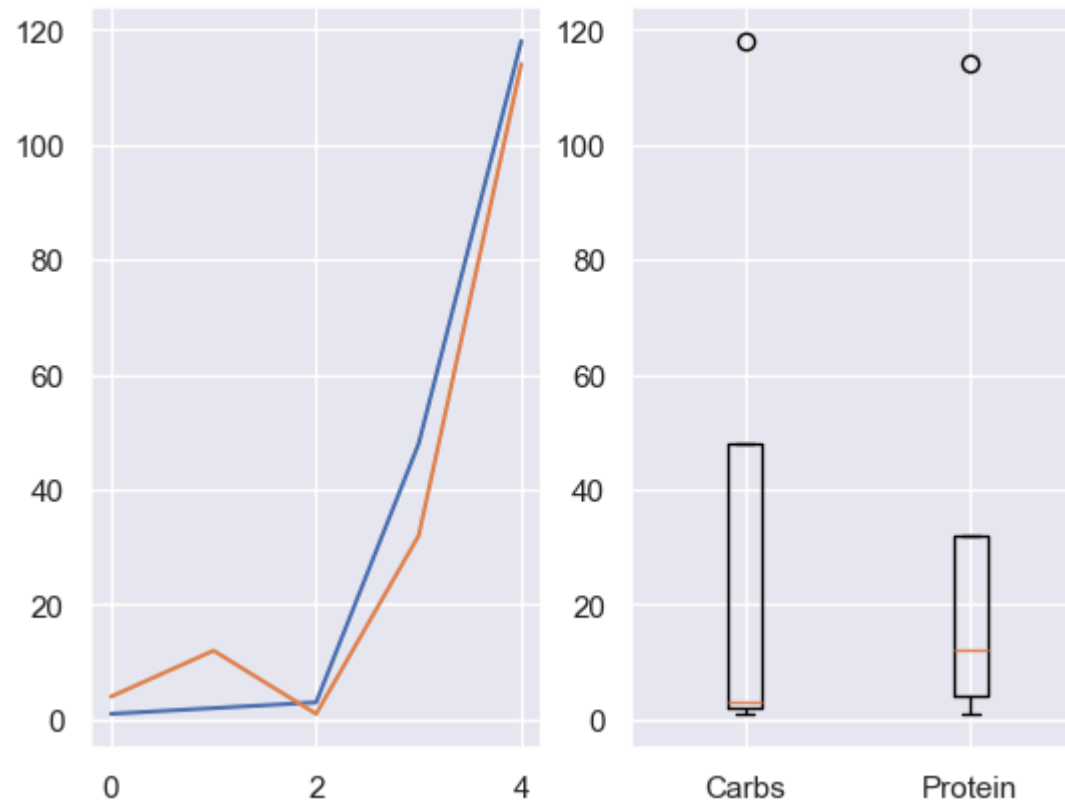
```
In [381]: fig, ax = plt.subplots()
line = ax.plot(range(df.index.size), df['Carbs'].values)
points = ax.plot(range(df.index.size), df['Protein'].values)
```



Combining different plots types into the same figure

We can also add the plots in distincts axes:

```
In [413]: fig, axes = plt.subplots(1, 2)
line = axes[0].plot(range(df.index.size), df['Carbs'].values)
points = axes[0].plot(range(df.index.size), df['Protein'].values)
slices = axes[1].boxplot([df.Carbs, df.Protein], tick_labels=["Carbs",
```



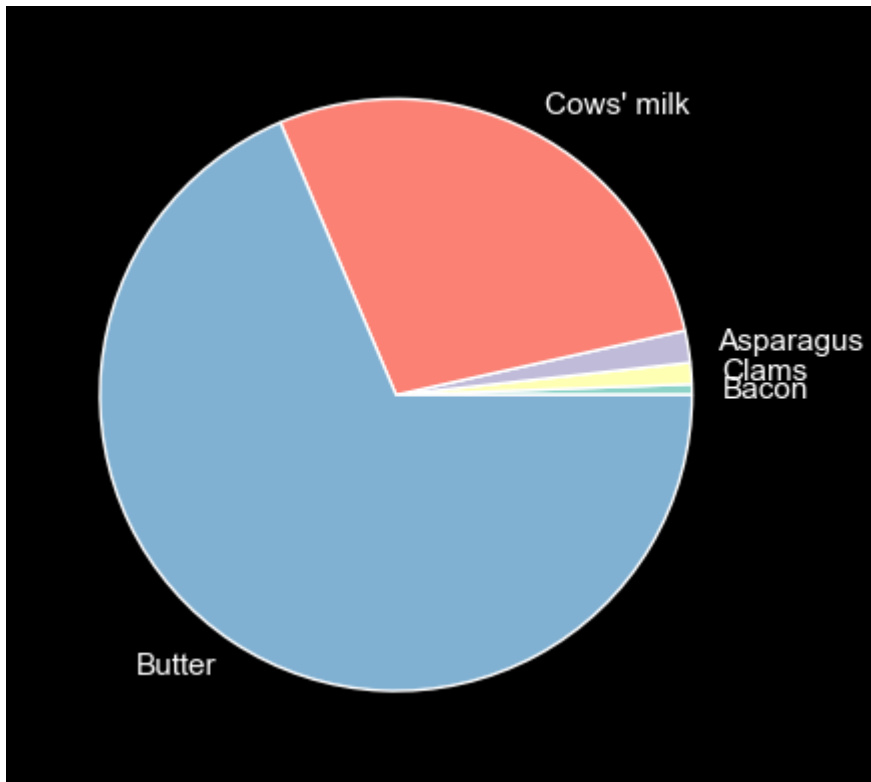
Plot styling

Pick a style

`matplotlib` ships with many styles that can drastically change the look of a plot, e.g., dark theme. Pick your favorite style from: [styles](#).

For instance, given the style `fivethirtyeight`, apply to your plots with:

```
In [384]: plt.style.use('dark_background')  
fig, ax = plt.subplots()  
slices = ax.pie(df.Carbs, labels=df.index)
```

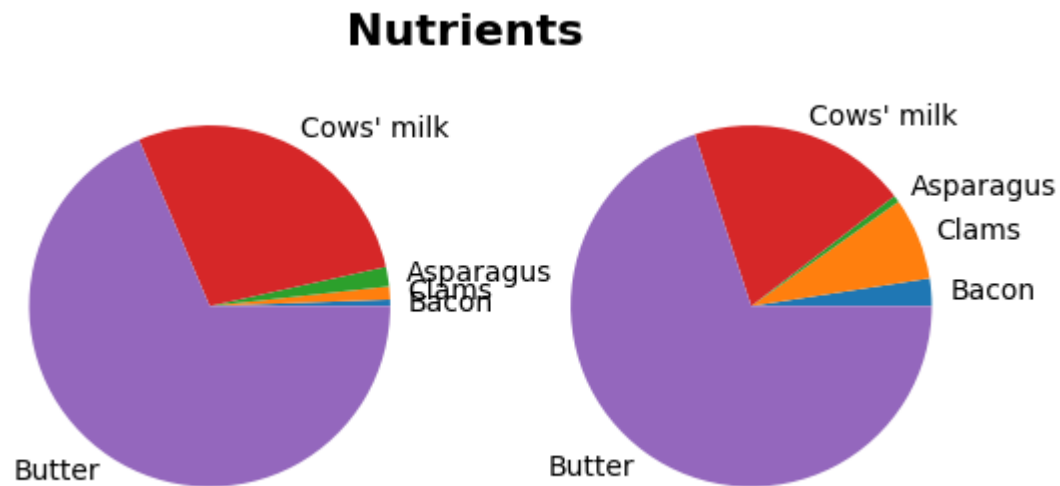


which is quite different from the default style:

```
In [385]: plt.style.use('default')  
fig, ax = plt.subplots()  
slices = ax.pie(df.Carbs, labels=df.index)
```

Setting a title to the figure

```
In [386]: fig, axes = plt.subplots(1, 2)
fig.suptitle("Nutrients", fontsize=16, fontweight='bold', y=0.80)
slices = axes[0].pie(df.Carbs, labels=df.index)
slices = axes[1].pie(df.Protein, labels=df.index)
```

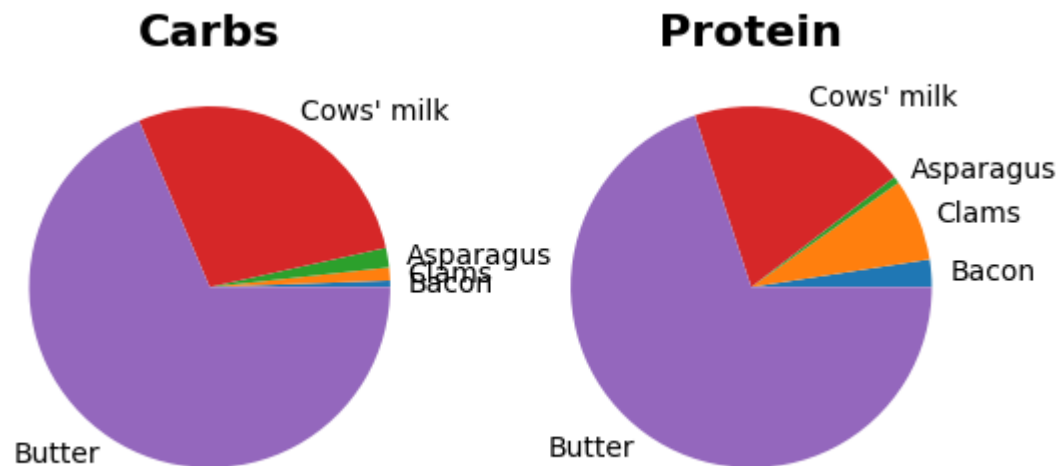


See: `matplotlib.figure.Figure.suptitle`

Setting a title to an Axes

```
In [387]: fig, axes = plt.subplots(1, 2)
          slices = axes[0].pie(df.Carbs, labels=df.index)
          axes[0].set_title("Carbs", fontsize=16, fontweight='bold')
          slices = axes[1].pie(df.Protein, labels=df.index)
          axes[1].set_title("Protein", fontsize=16, fontweight='bold')
```

Out[387]: Text(0.5, 1.0, 'Protein')

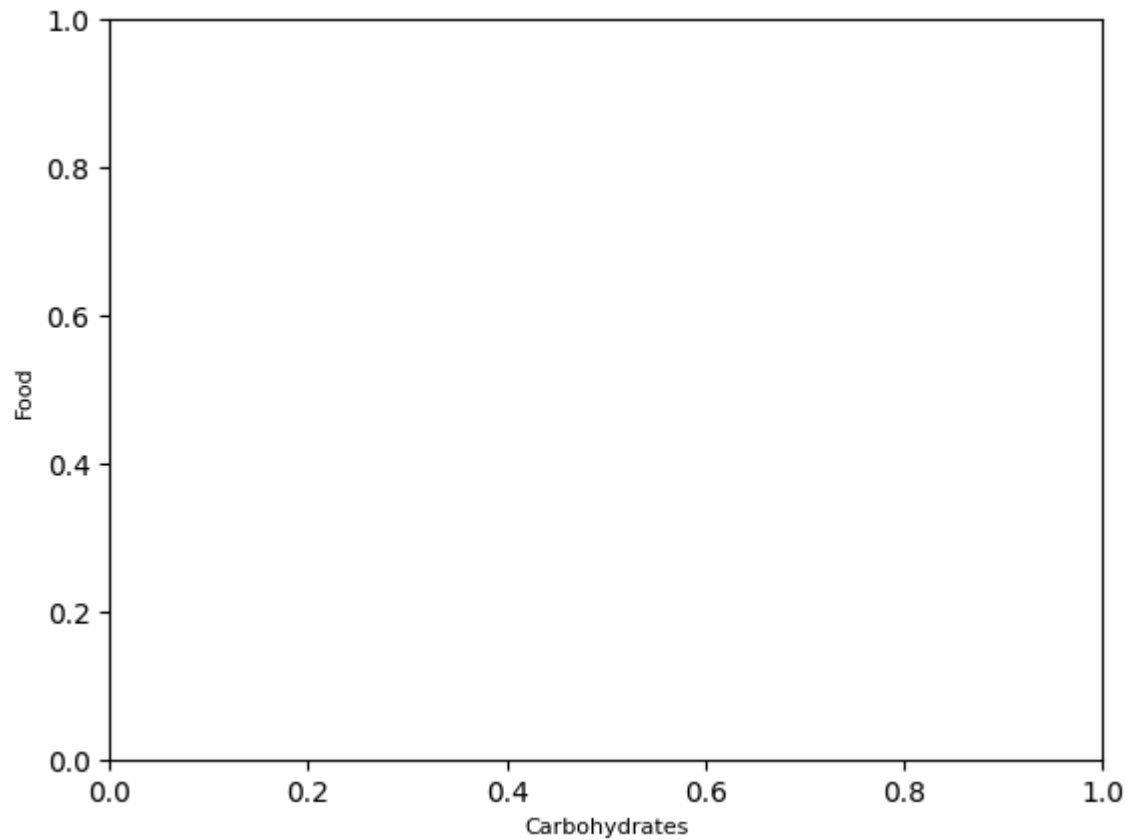


See: `matplotlib.axes.Axes.set_title`

Setting x or y labels

```
In [389]: fig, ax = plt.subplots()
          ax.set_xlabel('Carbohydrates', fontsize = 8)
          ax.set_ylabel('Food', fontsize = 8)
```

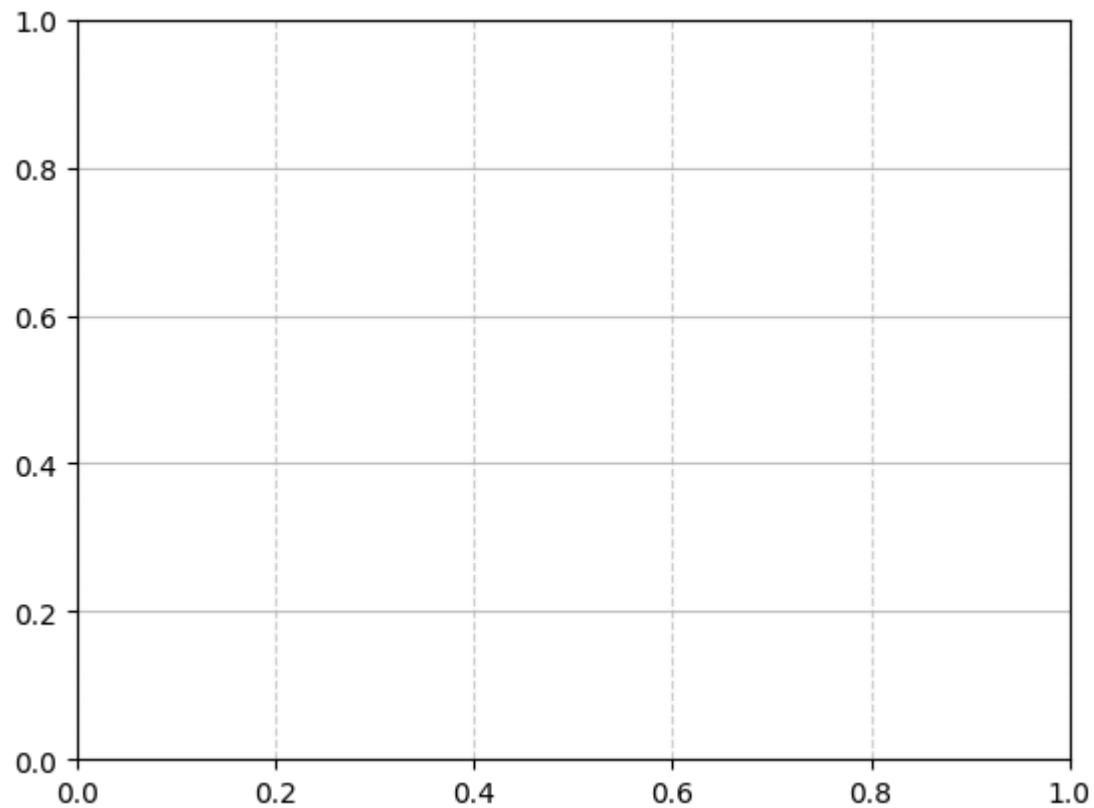
```
Out[389]: Text(0, 0.5, 'Food')
```



See: `matplotlib.axes.Axes.set_xlabel` and
`matplotlib.axes.Axes.set_ylabel`

Adding grid lines

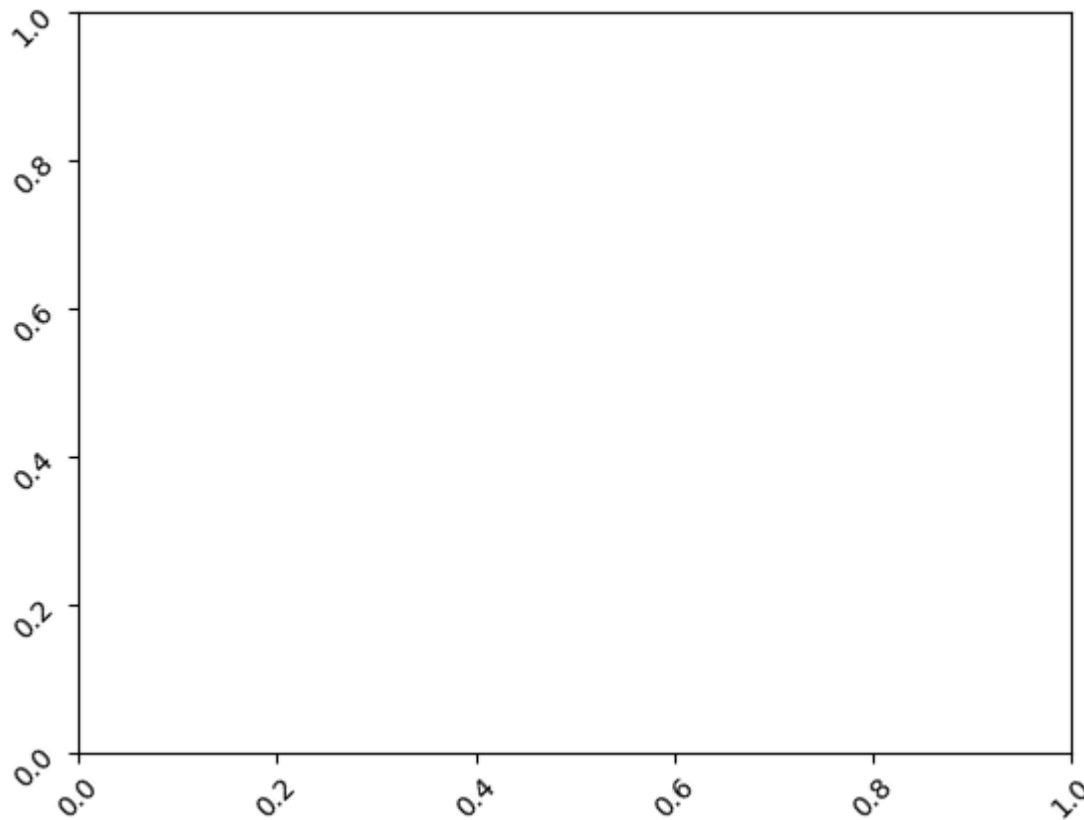
```
In [390]: fig, ax = plt.subplots()
ax.grid(axis='y', linestyle='-', alpha=0.8)
ax.grid(axis='x', linestyle='--', alpha=0.6)
```



See: `matplotlib.axes.Axes.grid`

Rotate x or y ticks labels

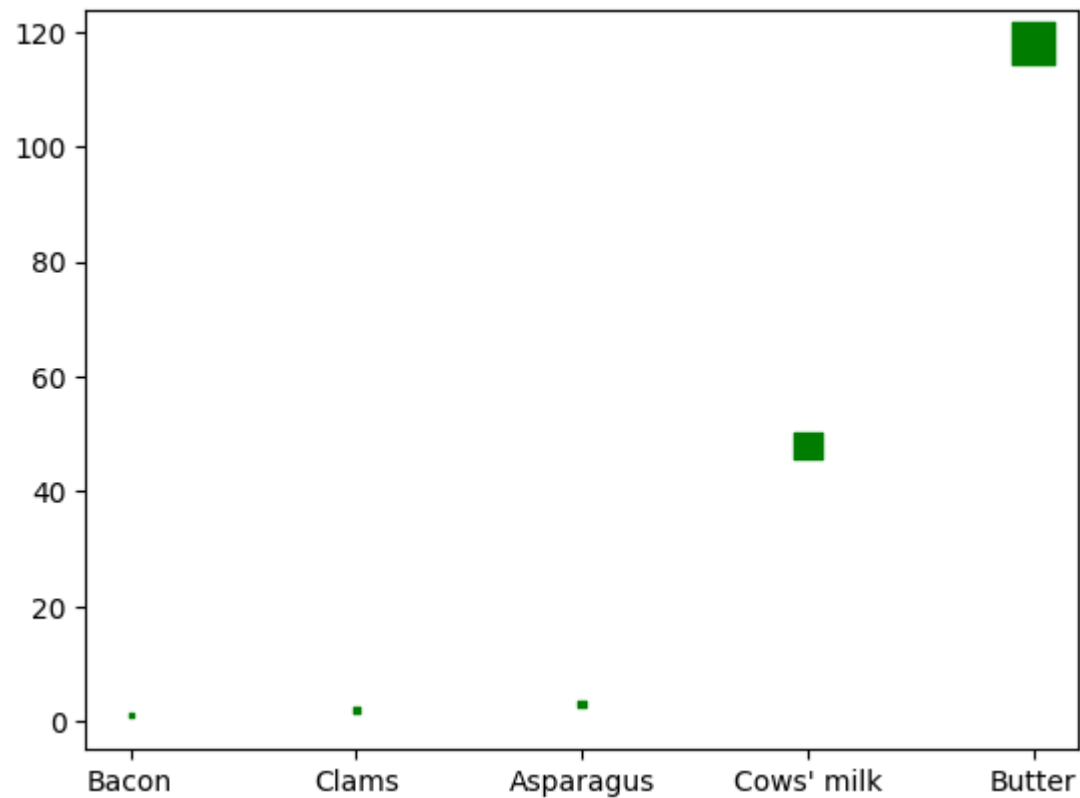
```
In [391]: fig, ax = plt.subplots()  
ax.tick_params(axis='x', labelrotation=45)  
ax.tick_params(axis='y', labelrotation=45)
```



See: `matplotlib.axes.Axes.tick_params`

Change color, size, and symbol of the markers in a scatter plot

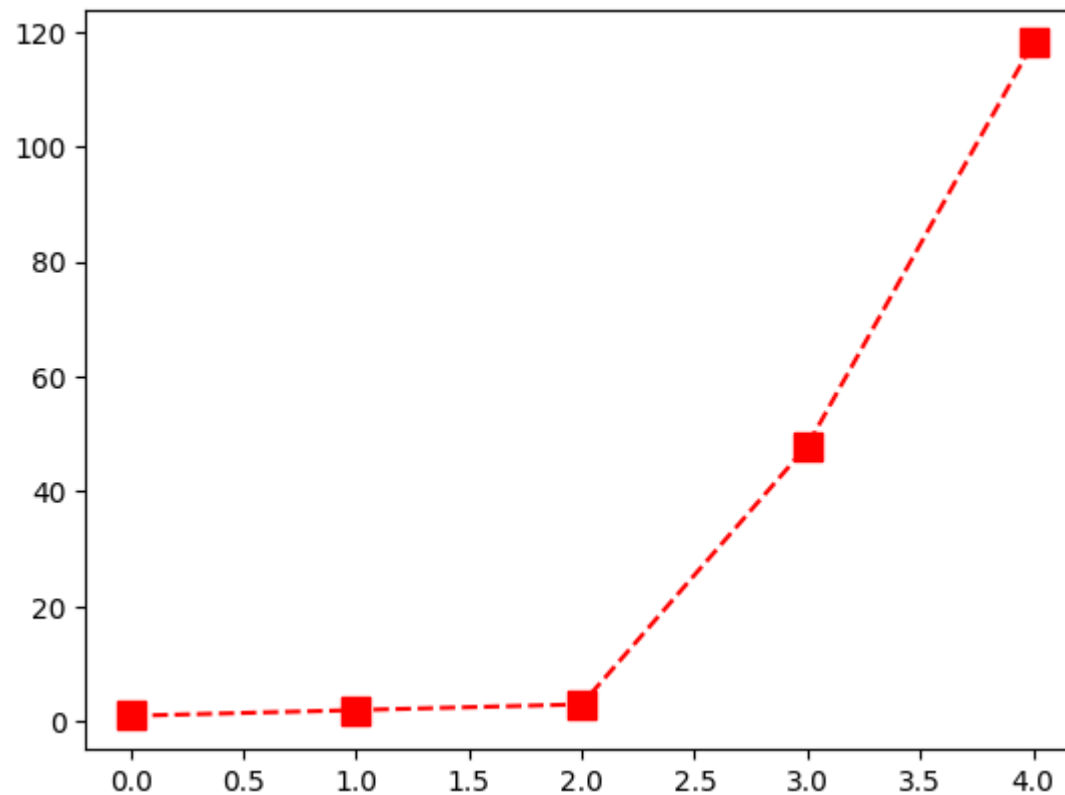
```
In [392]: fig, ax = plt.subplots()
points = ax.scatter(df.index, df.Carbs,
                    color='green',
                    marker='s',      #
                    s=df.Carbs*2)   # make the size proportional to the val
```



See: `matplotlib.markers`

Change color, size, marker, and style of a line in a plot

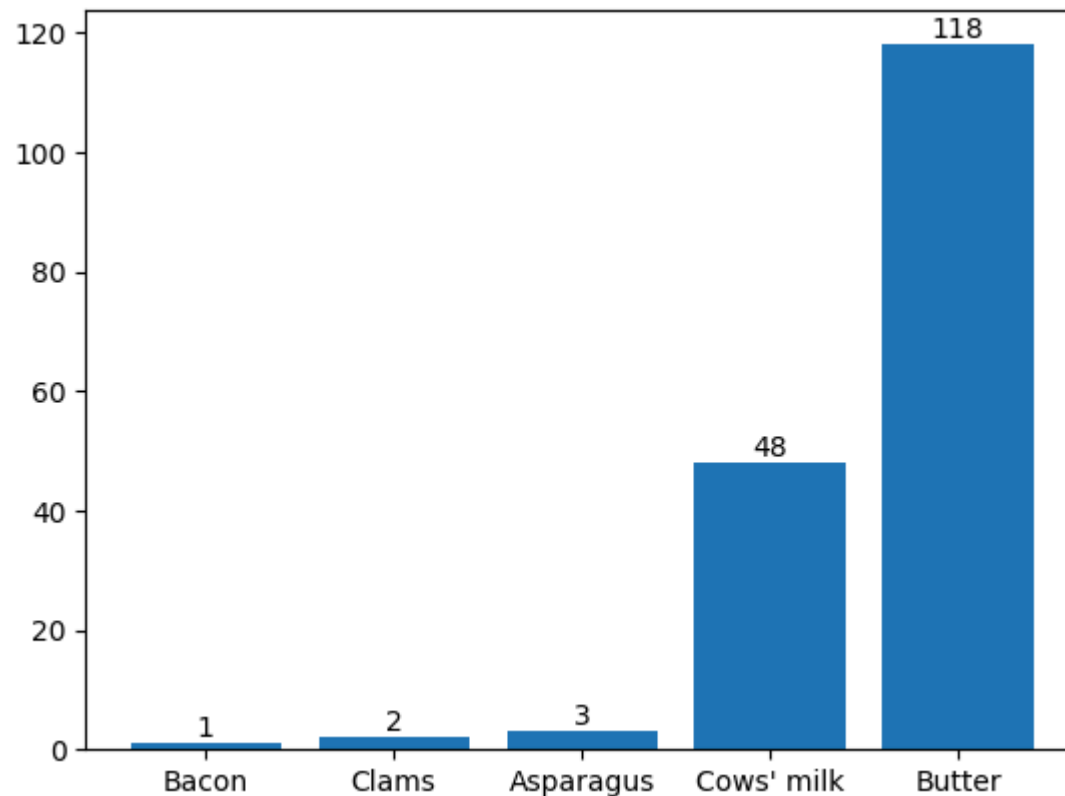
```
In [393]: fig, ax = plt.subplots()
points = ax.plot(range(df.index.size), df['Carbs'].values,
                  color='red',
                  linestyle='--',
                  marker='s',
                  markersize=10)
```



See: `matplotlib.lines.Line2D` and [Linestyles](#)

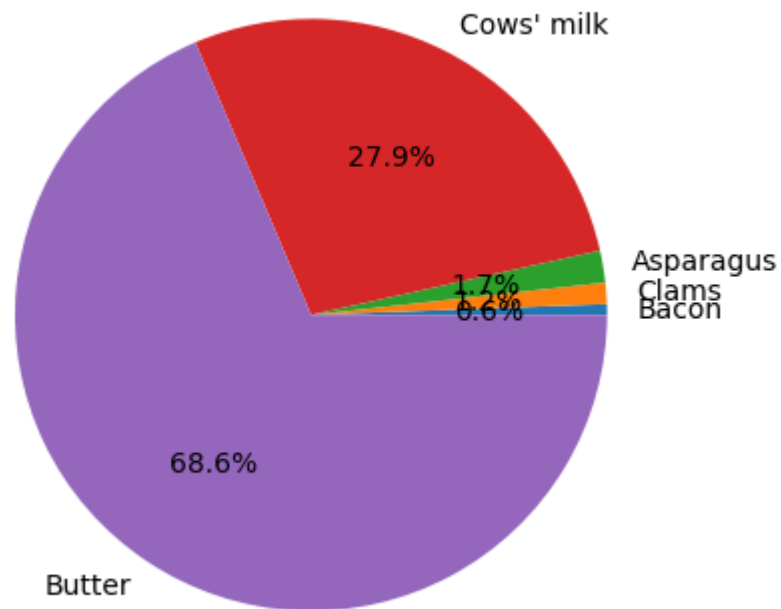
Adding label on top of each bar

```
In [394]: fig, ax = plt.subplots()
bars = ax.bar(df.index, df.Carbs) # vertical bar plot
for bar in bars:
    yval = bar.get_height()
    ax.text(bar.get_x() + bar.get_width()/2, yval + 0.1, round(yval, 2))
```



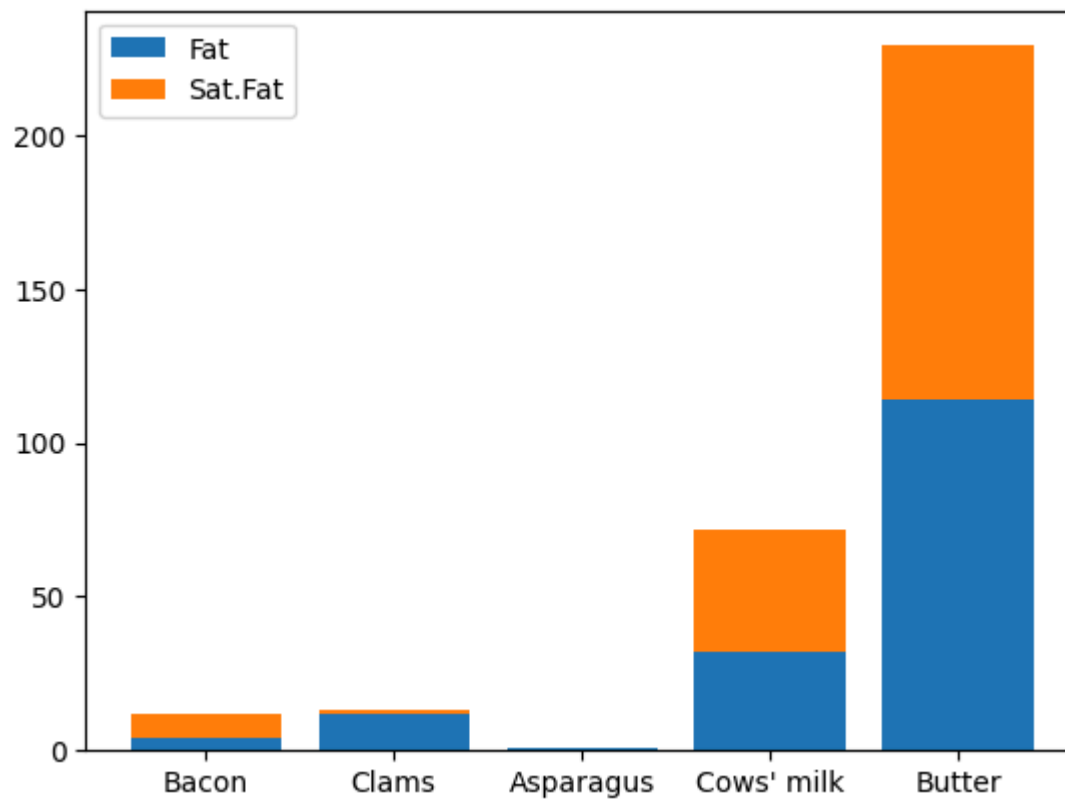
Add percentage label in a pie chart

```
In [395]: fig, ax = plt.subplots()
          slices = ax.pie(df.Carbs, labels=df.index, autopct='%1.1f%%')
```



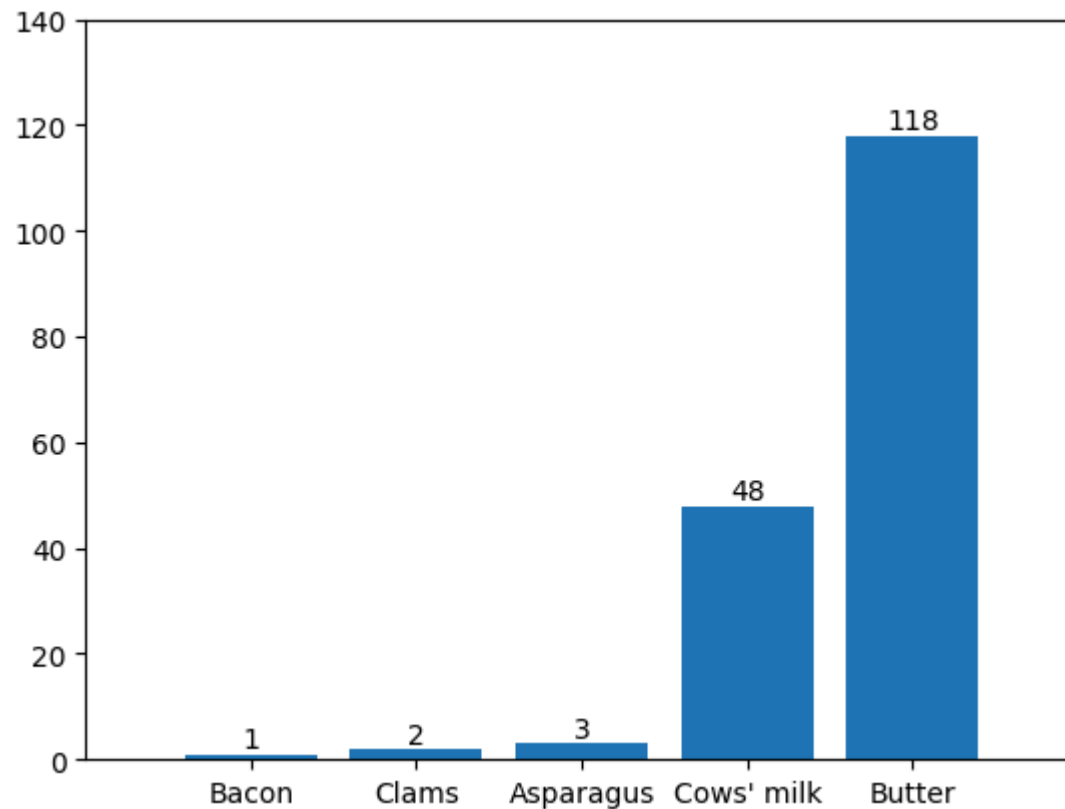
Add a legend

```
In [396]: fig, ax = plt.subplots()
bars = ax.bar(df.index, df["Protein"])
bars = ax.bar(df.index, df["Fat"], bottom=df["Protein"])
legend = ax.legend(df[["Fat", "Sat.Fat"]].columns)
```



Change x and y limits

```
In [397]: fig, ax = plt.subplots()
bars = ax.bar(df.index, df.Carbs) # vertical bar plot
for bar in bars:
    ax.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 0.1, rc
ax.set_xlim(-1, 5)
_ = ax.set_ylim(0, 140)
```



Plot exporting (PNG, PDF)

How to visualize a plot

While when working in a notebook any plot is automatically shown, **when writing a Python script**, we have to explicitly decide when to show or export a figure.

If we want to visualize the plot in a separate window:

```
In [398]: plt.show()
```

How to save a plot

```
In [399]: fig, ax = plt.subplots()
          bars = ax.bar(df.index, df["Protein"])
          plt.savefig("file.png") # save in PNG
          plt.savefig("file.pdf") # save in PDF
          plt.close() # close the figure to avoid displaying it
```

Documentation

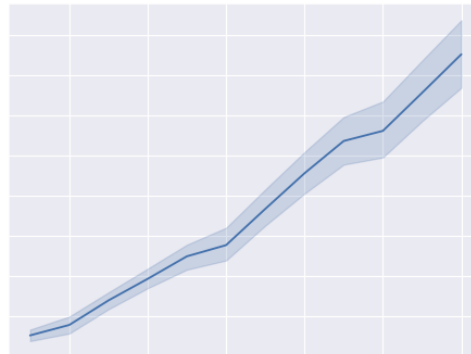
- **User guide:** <https://matplotlib.org/stable/users/index.html>
- **API:** <https://matplotlib.org/stable/api/index.html>

Package `seaborn`

Why `seaborn`?

`matplotlib` is *extremely* powerful. However, it can be very tricky when aiming at some *advanced* plots. For instance, given a dataset, if we want to plot:

- a line that represents the average of the data points over a specific x
- the confidence interval for each data points over a specific x

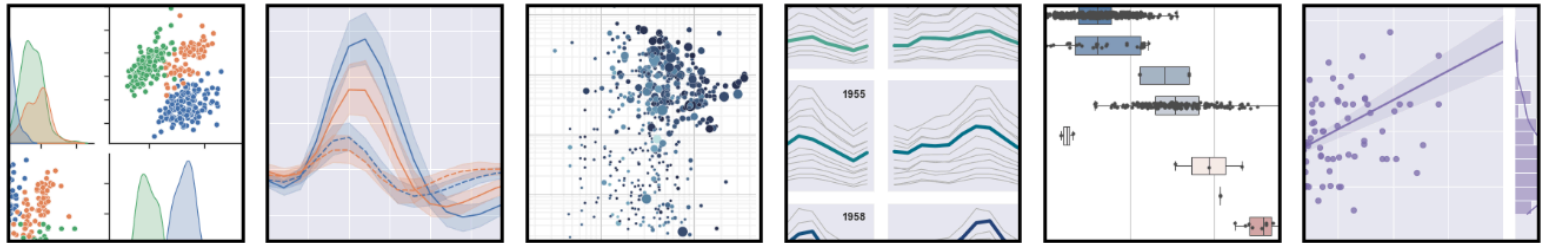


In `matplotlib`, we have to:

- compute the average
- `plot` the line for the average
- compute the confidence interval
- plot the confidence intervals with, e.g., `fill_between`

seaborn: matplotlib for the humans

seaborn is a data visualization library based on matplotlib. It provides a high-level interface for drawing *attractive* and **informative statistical** graphics.



It comes with convenient *shortcuts* to build several nice plots using matplotlib. Moreover, it naturally fits with pandas and scikit-learn.

seaborn: installation

Install it with `pip3`:

```
In [400]: ! pip3 install seaborn
```

```
Defaulting to user installation because normal site-packages is not writeable
```

```
Requirement already satisfied: seaborn in /home/ercoppa/.local/lib/python3.10/site-packages (0.13.2)
```

```
Requirement already satisfied: numpy!=1.24.0,>=1.20 in /home/ercoppa/.local/lib/python3.10/site-packages (from seaborn) (1.26.4)
```

```
Requirement already satisfied: pandas>=1.2 in /home/ercoppa/.local/lib/python3.10/site-packages (from seaborn) (2.2.2)
```

```
Requirement already satisfied: matplotlib!=3.6.1,>=3.4 in /home/ercoppa/.local/lib/python3.10/site-packages (from seaborn) (3.9.2)
```

```
Requirement already satisfied: fonttools>=4.22.0 in /usr/lib/python3/dist-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (4.29.1)
```

```
Requirement already satisfied: pyparsing>=2.3.1 in /usr/lib/python3/dist-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (2.4.7)
```

```
Requirement already satisfied: contourpy>=1.0.1 in /home/ercoppa/.local/lib/python3.10/site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (1.2.1)
```

```
Requirement already satisfied: python-dateutil>=2.7 in /usr/loc
```

```
al/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.4->
seaborn) (2.9.0.post0)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/lib/py
thon3/dist-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (1.
3.2)
Requirement already satisfied: packaging>=20.0 in /home/ercopp
a/.local/lib/python3.10/site-packages (from matplotlib!=3.6.1,>
=3.4->seaborn) (24.1)
Requirement already satisfied: pillow>=8 in /usr/lib/python3/di
st-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (9.0.1)
Requirement already satisfied: cycler>=0.10 in /usr/lib/python
3/dist-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (0.11.
0)
Requirement already satisfied: pytz>=2020.1 in /usr/lib/python
3/dist-packages (from pandas>=1.2->seaborn) (2022.1)
Requirement already satisfied: tzdata>=2022.7 in /home/ercopp
a/.local/lib/python3.10/site-packages (from pandas>=1.2->seabor
n) (2024.1)
Requirement already satisfied: six>=1.5 in /usr/lib/python3/dis
t-packages (from python-dateutil>=2.7->matplotlib!=3.6.1,>=3.4-
>seaborn) (1.16.0)
```

seaborn: installation

Import it:

```
In [401]: import seaborn as sns
```

Some data to plot

In our examples, we reuse a few dummy datasets from `searbond`:

```
In [402]: tips = sns.load_dataset("tips")
tips
```

```
Out[402]:
```

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4
...
239	29.03	5.92	Male	No	Sat	Dinner	3
240	27.18	2.00	Female	Yes	Sat	Dinner	2
241	22.67	2.00	Male	Yes	Sat	Dinner	2
242	17.82	1.75	Male	No	Sat	Dinner	2
243	18.78	3.00	Female	No	Thur	Dinner	2

244 rows × 7 columns

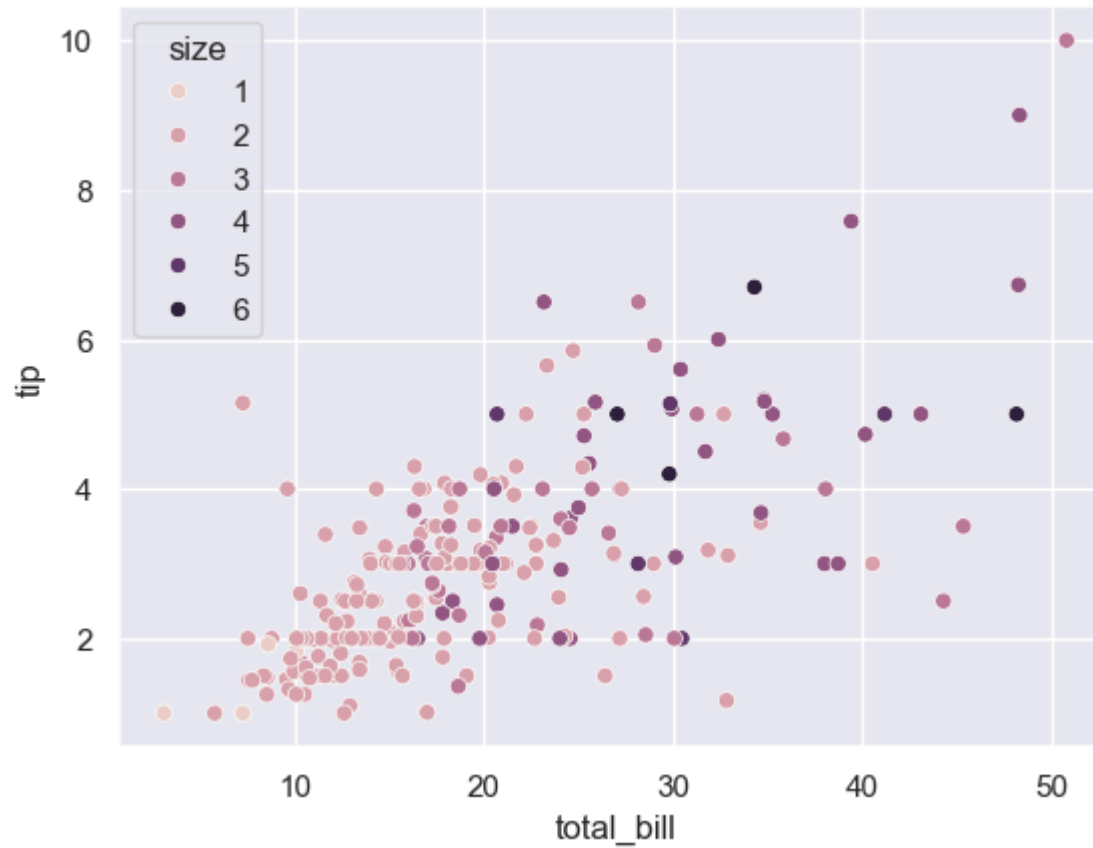
seaborn theme

We can use the `matplotlib` style from `seaborn`:

```
In [403]: # Apply the default theme  
sns.set_theme()
```

A nicer scatter plot

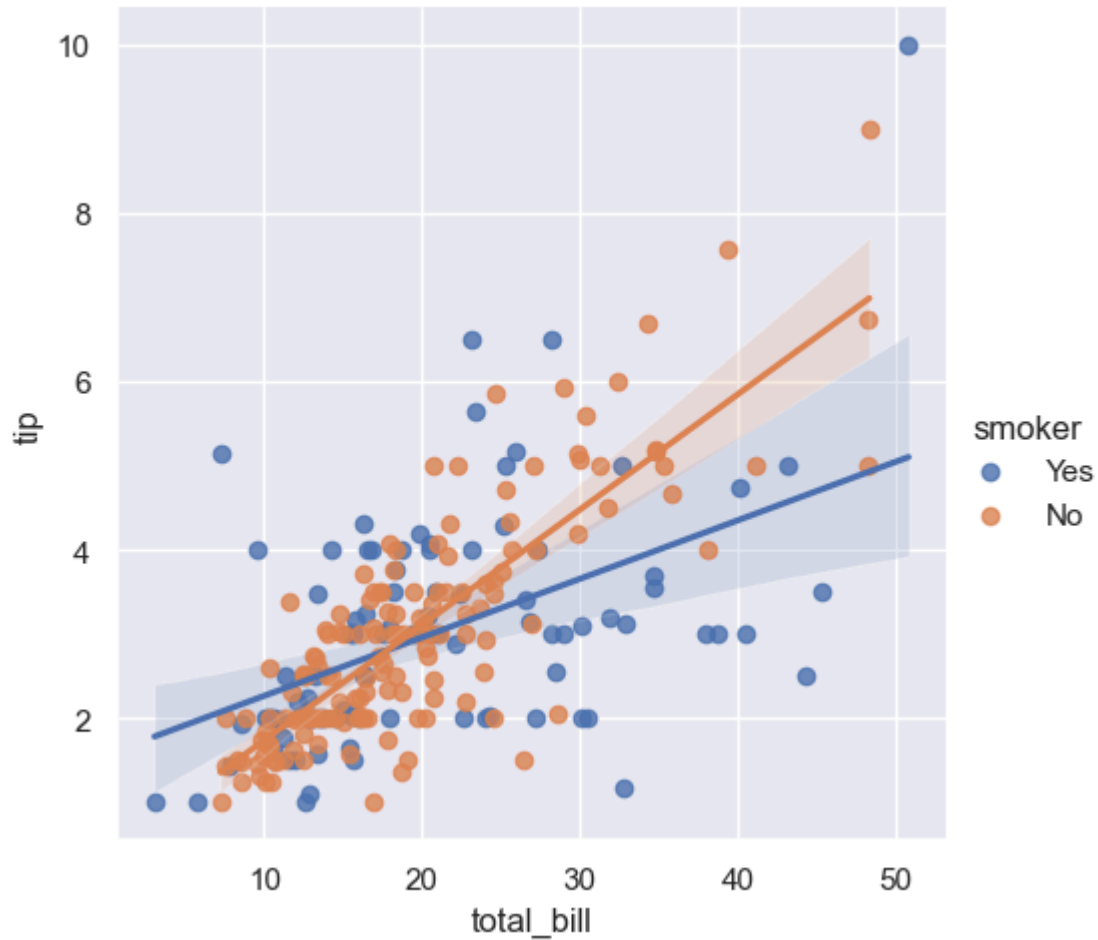
```
In [404]: ax = sns.scatterplot(data=tips, x="total_bill", y="tip", hue="size")
```



See: `seaborn.scatterplot`

A nicer scatter plot with linear regression

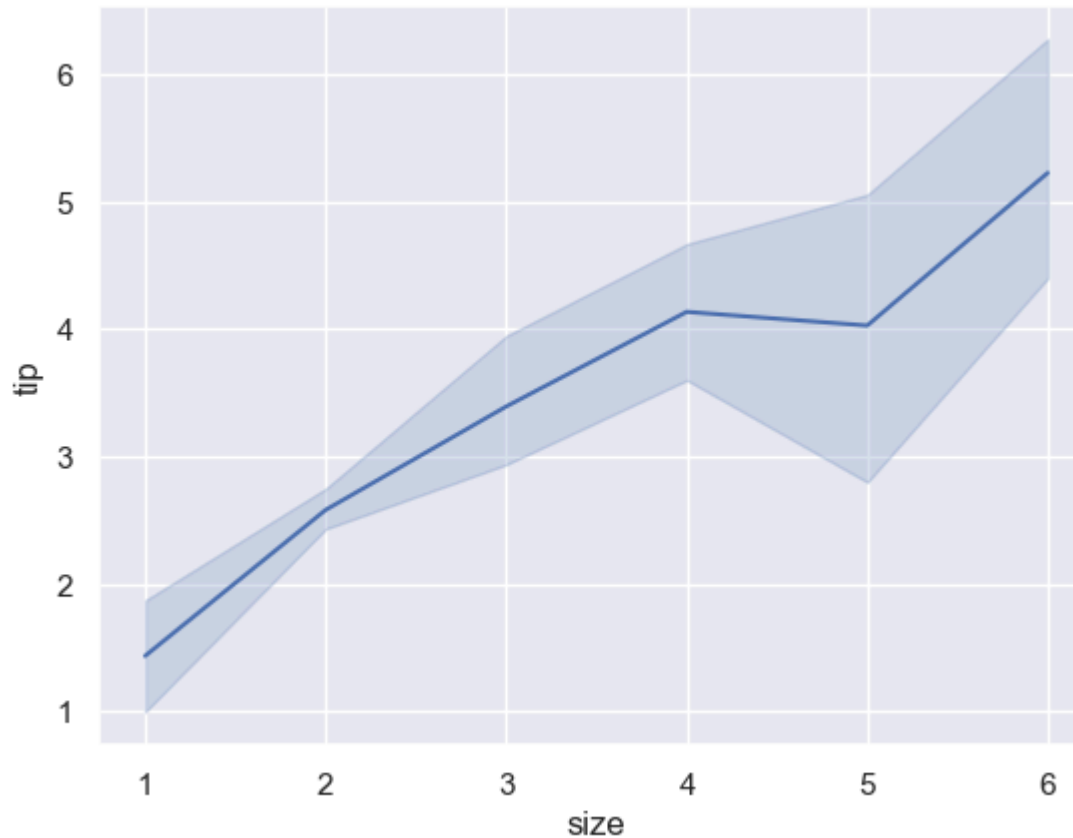
```
In [405]: l = sns.lmplot(data=tips, x="total_bill", y="tip", hue="smoker")
```



See: `seaborn.lmplot`

A nicer line plot with confidence intervals

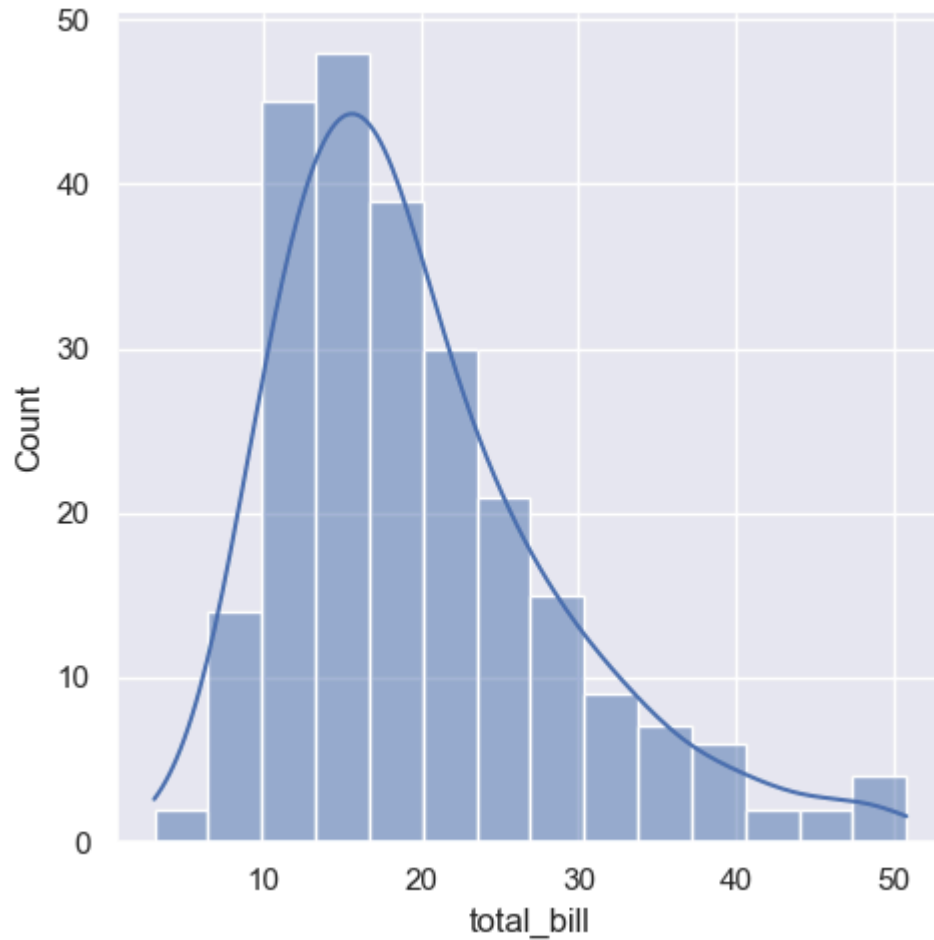
```
In [406]: lines = sns.lineplot(data=tips, x="size", y="tip", errorbar="ci")
```



See: `seaborn.lineplot`

A nicer histogram

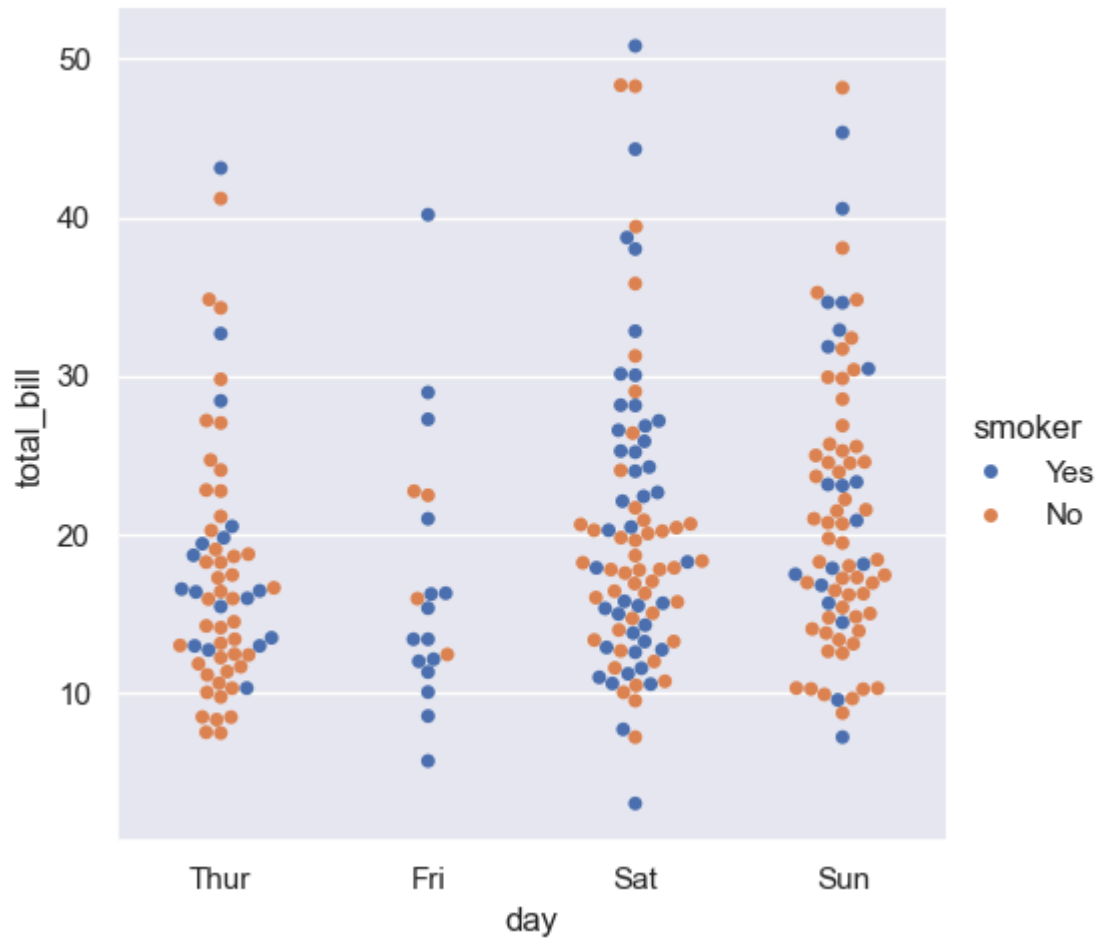
```
In [407]: s = sns.displot(data=tips, x="total_bill", kde=True)
```



See: `seaborn.displot`

A nice categorical plot

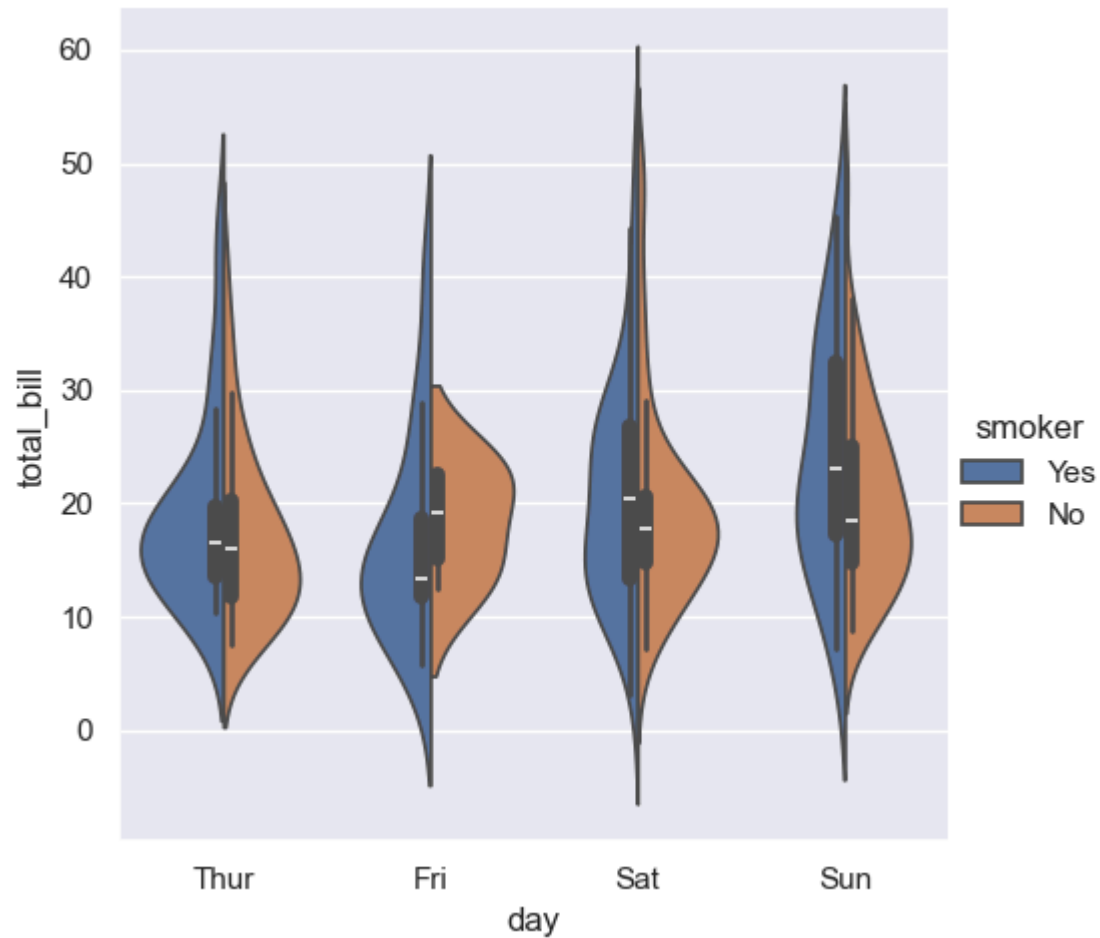
```
In [408]: s = sns.catplot(data=tips, kind="swarm", x="day", y="total_bill", hue="
```



See: `seaborn.catplot`

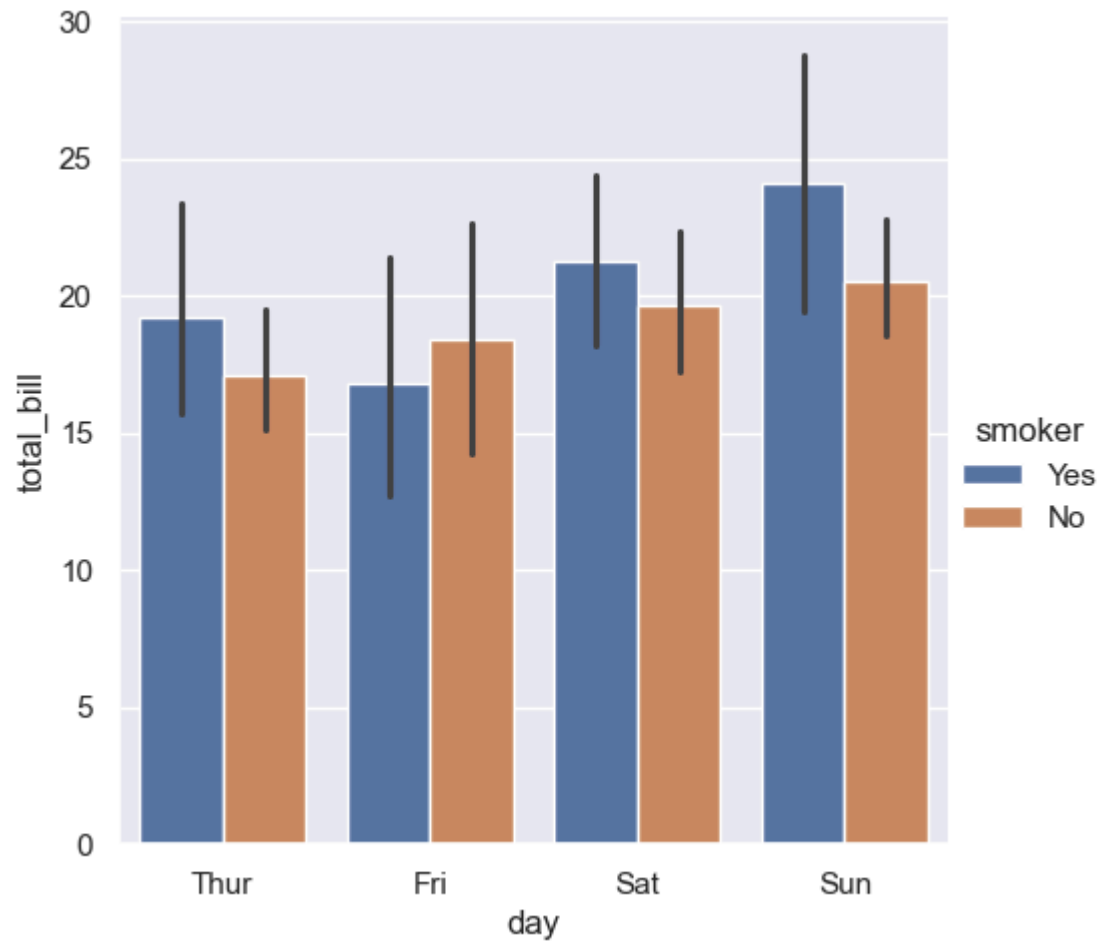
A nice categorical plot with violins

```
In [409]: s = sns.catplot(data=tips, kind="violin", x="day", y="total_bill", hue=
```



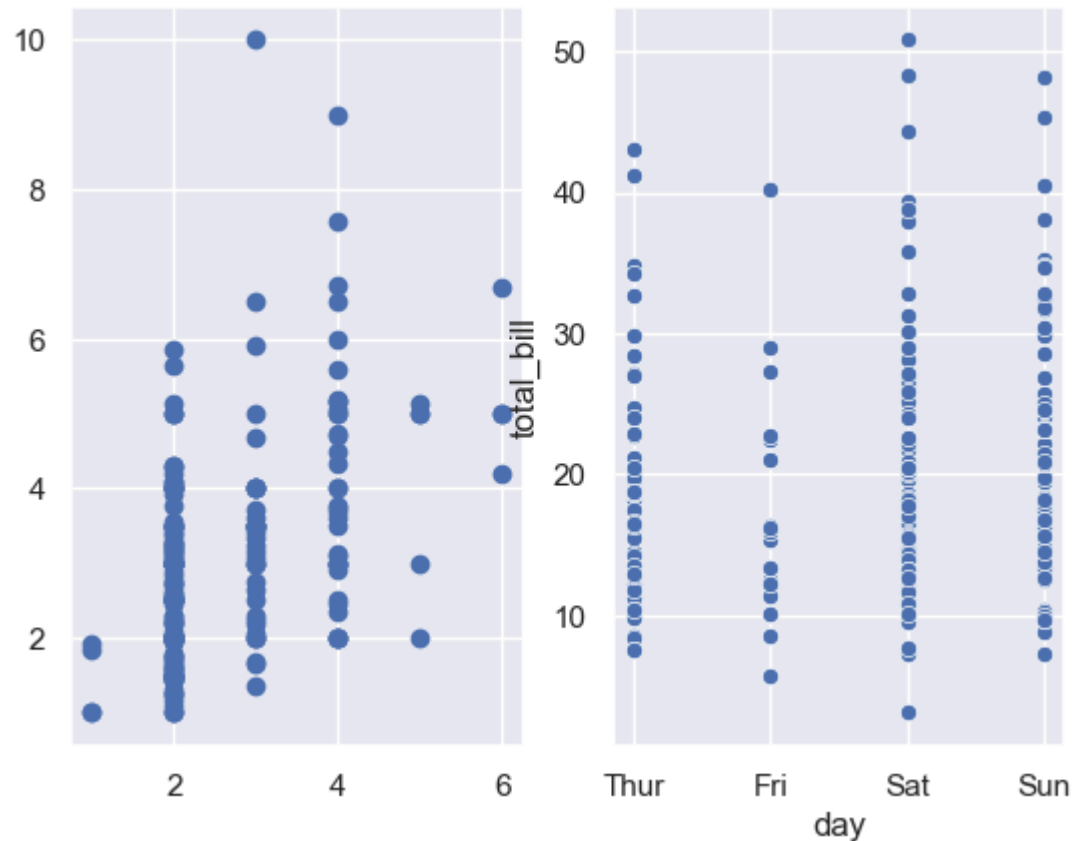
A nice barplot plot with errorbars

```
In [410]: s = sns.catplot(data=tips, kind="bar", x="day", y="total_bill", hue="smoker", errorbar="ci")
```



Combine `matplotlib` and `seaborn` plots

```
In [411]: fix, axes = plt.subplots(1, 2)
axes[0].scatter(tips['size'].values, tips.tip.values)
s = sns.scatterplot(data=tips, x="day", y="total_bill", ax=axes[1])
```

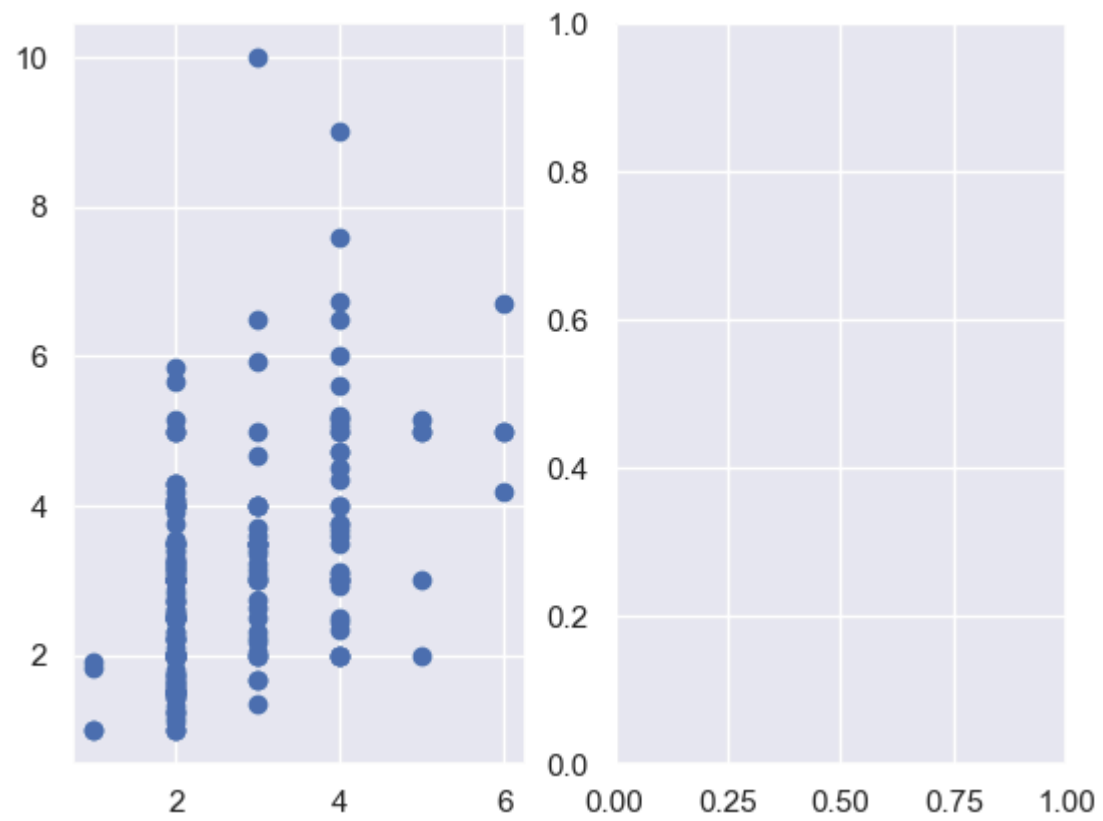


Combine `matplotlib` and `seaborn` plots (cont'd)

Unfortunately, some `seaborn` plots cannot be mixed with `matplotlib` plots within the same figure. Indeed, such plots are dubbed *figure-level*. For instance, `catplot` is figure-level and the following code will generate two distinct figures:

```
In [412]: fix, axes = plt.subplots(1, 2)
          axes[0].scatter(tips['size'].values, tips.tip.values)
          s = sns.catplot(data=tips, x="day", y="total_bill", ax=axes[1])
```

```
/home/ercoppa/.local/lib/python3.10/site-packages/seaborn/categorical.py:2761: UserWarning: catplot is a figure-level function
and does not accept target axes. You may wish to try stripplot
warnings.warn(msg, UserWarning)
```



Documentation

- **Tutorial:** <https://seaborn.pydata.org/tutorial.html>
- **API:** <https://seaborn.pydata.org/api.html>

