matplotlib / pyplot & seaborn

Lecture 11

Dr. Colin Rundel

matplotlib & pyplot

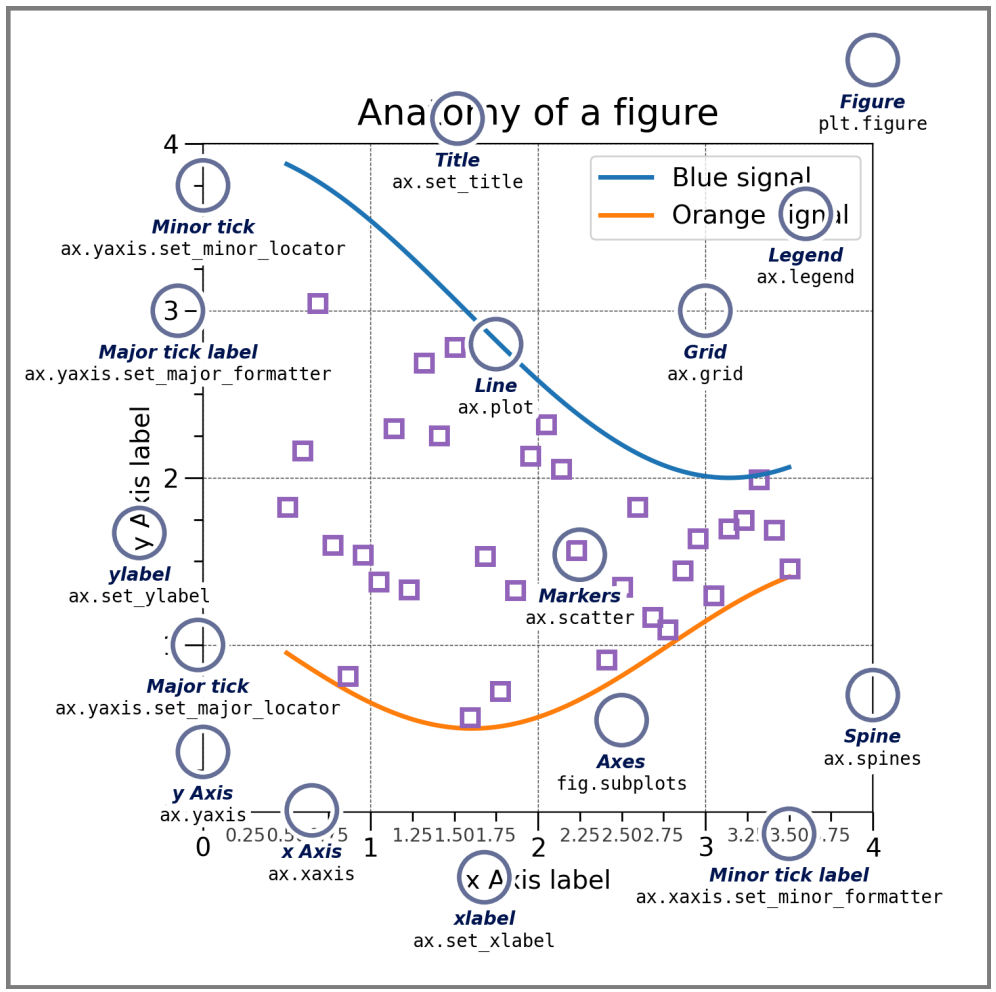
> matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.

```
1  import matplotlib as mpl
```

> `matplotlib.pyplot` is a collection of functions that make matplotlib work like MATLAB. Each `pyplot` function makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc.

```
1  import matplotlib.pyplot as plt
```

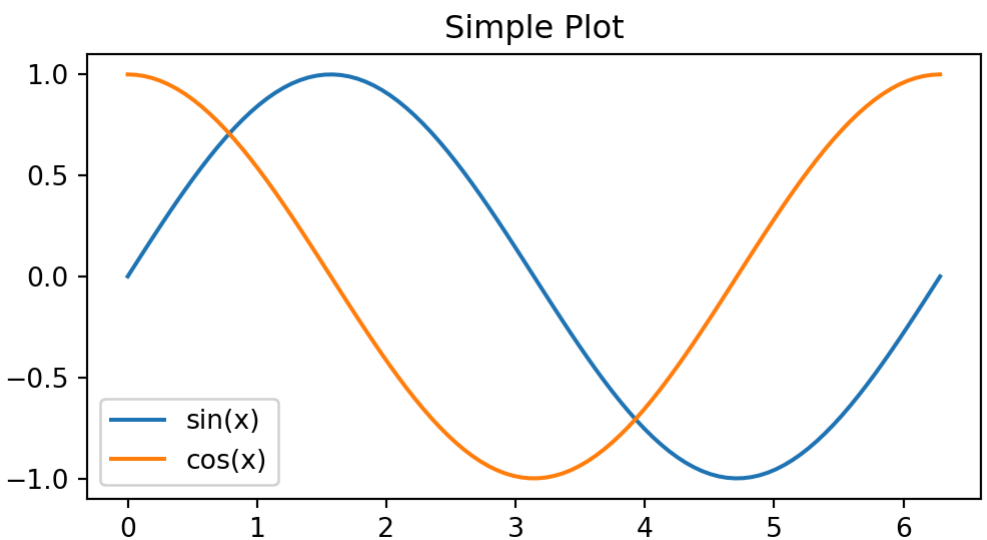
Plot anatomy



- **Figure** - The entire plot (including subplots)
- **Axes** - Subplot attached to a figure, contains the region for plotting data and x & y axis
- **Axis** - Set the scale and limits, generate ticks and ticklabels
- **Artist** - Everything visible on a figure: text, lines, axis, axes, etc.

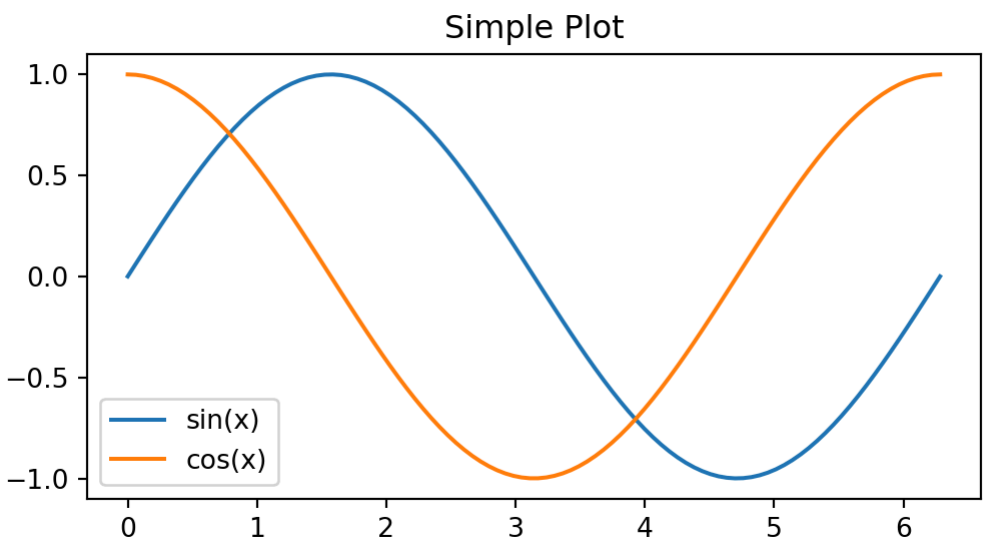
Basic plot - pyplot style

```
x = np.linspace(0, 2*np.pi, 100)
y1 = np.sin(x)
y2 = np.cos(x)

plt.figure(figsize=(6, 3))
plt.plot(x, y1, label="sin(x)")
plt.plot(x, y2, label="cos(x)")
plt.title("Simple Plot")
plt.legend()
```



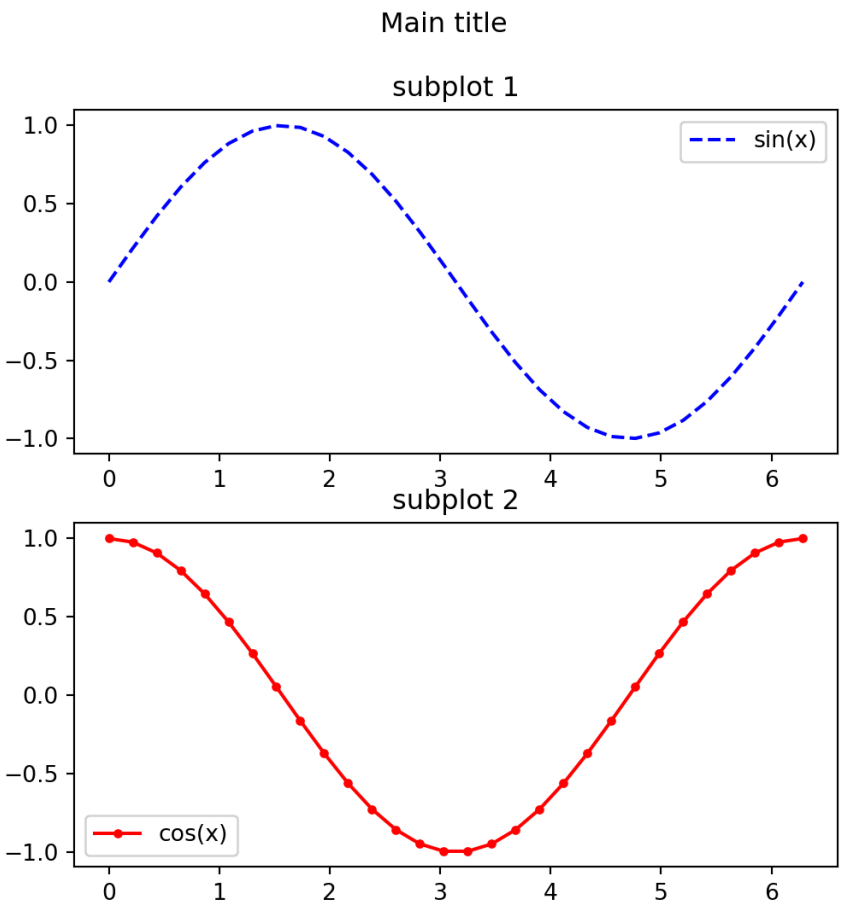
Basic plot - OO style

```
x = np.linspace(0, 2*np.pi, 100)
y1 = np.sin(x)
y2 = np.cos(x)

fig, ax = plt.subplots(figsize=(6, 3))
ax.plot(x, y1, label="sin(x)")
ax.plot(x, y2, label="cos(x)")
ax.set_title("Simple Plot")
ax.legend()
```



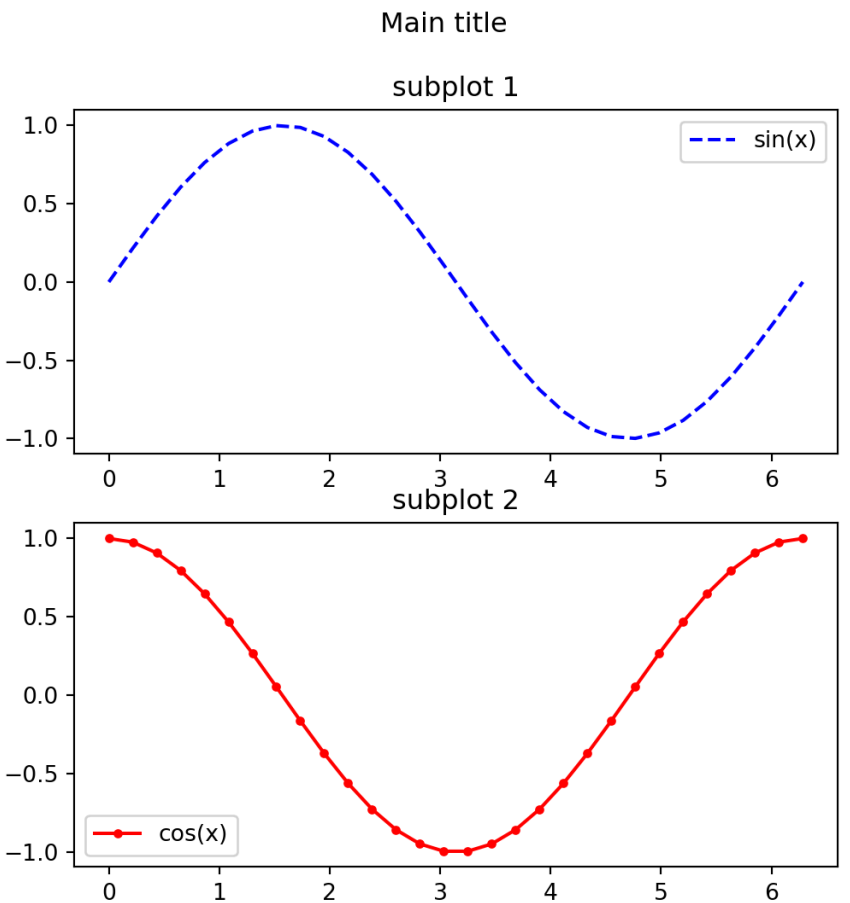
Subplots (OO)

```
x = np.linspace(0, 2*np.pi, 30)
y1 = np.sin(x)
y2 = np.cos(x)

fig, (ax1, ax2) = plt.subplots(
  2, 1, figsize=(6, 6)
)

fig.suptitle("Main title")

ax1.plot(x, y1, "--b", label="sin(x)")
ax1.set_title("subplot 1")
ax1.legend()

ax2.plot(x, y2, ".-r", label="cos(x)")
ax2.set_title("subplot 2")
ax2.legend()
```



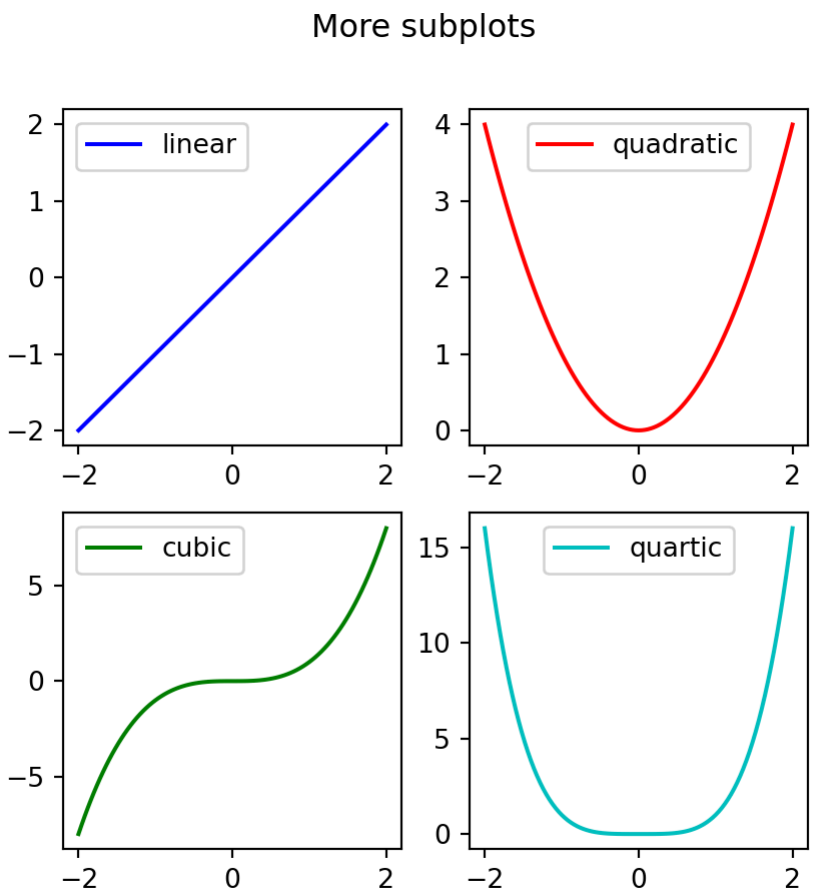
Subplots (pyplot)

```
x = np.linspace(0, 2*np.pi, 30)
y1 = np.sin(x)
y2 = np.cos(x)

plt.figure(figsize=(6, 6))

plt.suptitle("Main title")

plt.subplot(211)
plt.plot(x, y1, "--b", label="sin(x)")
plt.title("subplot 1")
plt.legend()

plt.subplot(2,1,2)
plt.plot(x, y2, ".-r", label="cos(x)")
plt.title("subplot 2")
plt.legend()

plt.show()
```



More subplots

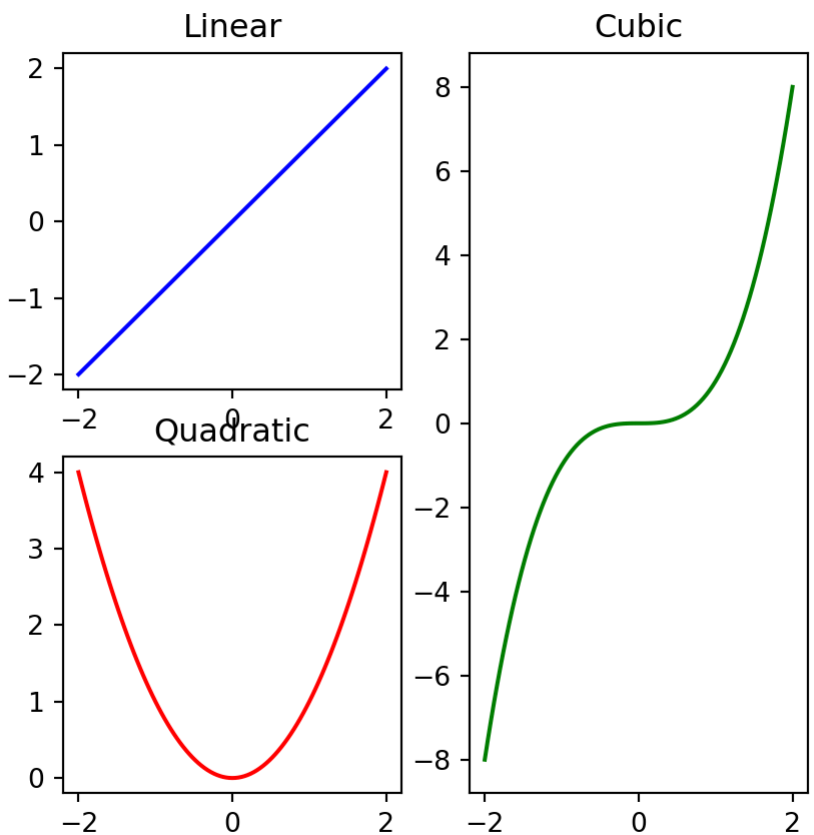
```
1  x = np.linspace(-2, 2, 101)
2
3  fig, axs = plt.subplots(
4    2, 2,
5    figsize=(5, 5)
6  )
7
8  fig.suptitle("More subplots")
9
10 axs[0,0].plot(x, x, "b", label="linear")
11 axs[0,1].plot(x, x**2, "r", label="quadrati
12 axs[1,0].plot(x, x**3, "g", label="cubic")
13 axs[1,1].plot(x, x**4, "c", label="quartic"
14
15 for ax in axs.flat:
16     ax.legend()
```



`axs` here is a 2x2 numpy array of axes

Fancy subplots (mosaic)

```
x = np.linspace(-2, 2, 101)

fig, axd = plt.subplot_mosaic(
  [['upleft', 'right'],
   ['lowleft', 'right']],
  figsize=(5, 5)
)

axd['upleft' ].plot(x, x,    "b")
axd['lowleft'].plot(x, x**2, "r")
axd['right'  ].plot(x, x**3, "g")

axd['upleft'].set_title("Linear")
axd['lowleft'].set_title("Quadratic")
axd['right'].set_title("Cubic")
```



`axd` here is a *dictionary* of axes

Format strings

For quick formatting of plots (scatter and line) format strings are a useful shorthand. Generally they use the format `'[marker][line][color]'`,

Markers

character	shape
`.`	point
`,`	pixel
`o`	circle
`v`	triangle down
`^`	triangle up
`<`	triangle left
`>`	triangle right
…	+ more

Lines

character	line style
`-`	solid
`--`	dashed
`-.`	dash-dot
`:`	dotted

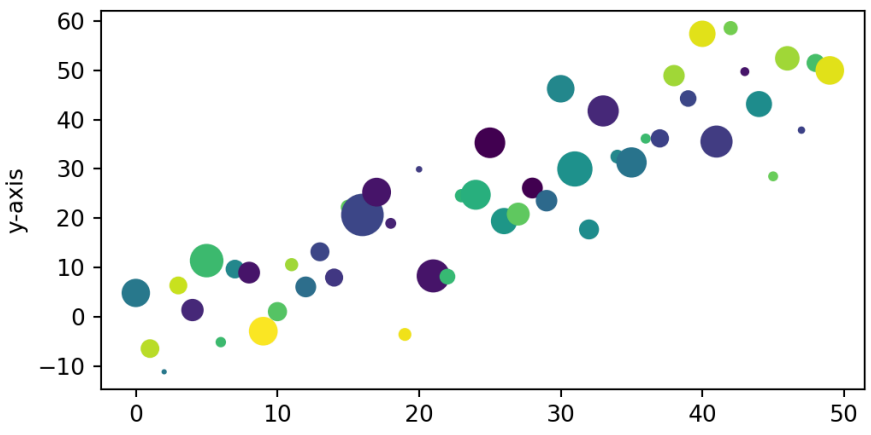
Colors

character	color
`b`	blue
`g`	green
`r`	red
`c`	cyan
`m`	magenta
`y`	yellow
`k`	black
`w`	white

Plotting data

Beyond creating plots for arrays (and lists), addressable objects like dicts and DataFrames can be used via `data`,

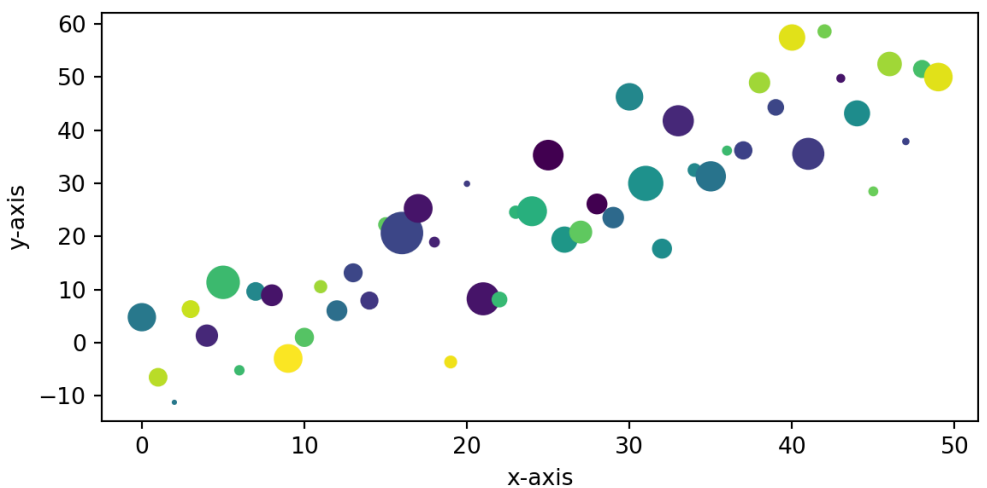
```
np.random.seed(19680801)
d = {
  'x': np.arange(50),
  'color': np.random.randint(0, 50, 50),
  'size': np.abs(np.random.randn(50)) * 100
}
d['y'] = d['x'] + 10 * np.random.randn(50)


plt.figure(figsize=(6, 3))
plt.scatter(
  'x', 'y', c='color', s='size',
  data=d
)
plt.xlabel("x-axis")
plt.ylabel("y-axis")

plt.show()
```



Constrained layout

To fix the axis label clipping, we can use the “constrained” layout to adjust automatically,

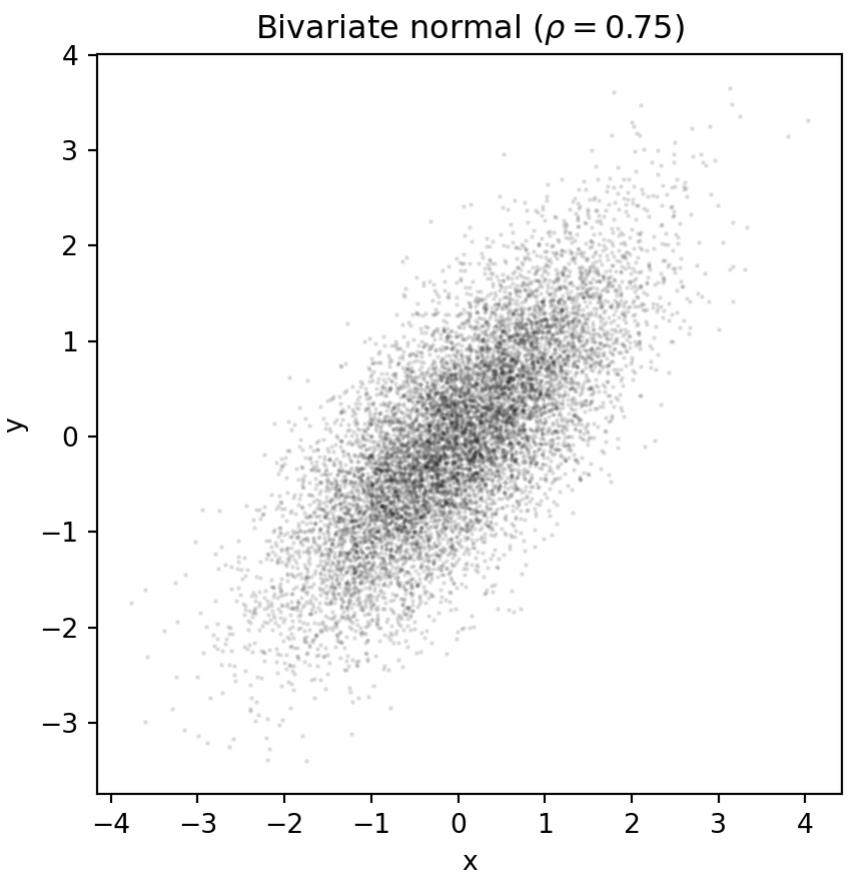
```
np.random.seed(19680801)
d = {
  'x': np.arange(50),
  'color': np.random.randint(0, 50, 50),
  'size': np.abs(np.random.randn(50)) * 100
}
d['y'] = d['x'] + 10 * np.random.randn(50)


plt.figure(
  figsize=(6, 3),
  layout="constrained"
)
plt.scatter(
  'x', 'y', c='color', s='size',
  data=d
)
plt.xlabel("x-axis")
plt.ylabel("y-axis")

plt.show()
```



pyplot w/ pandas

Data can also come from DataFrame objects or series,

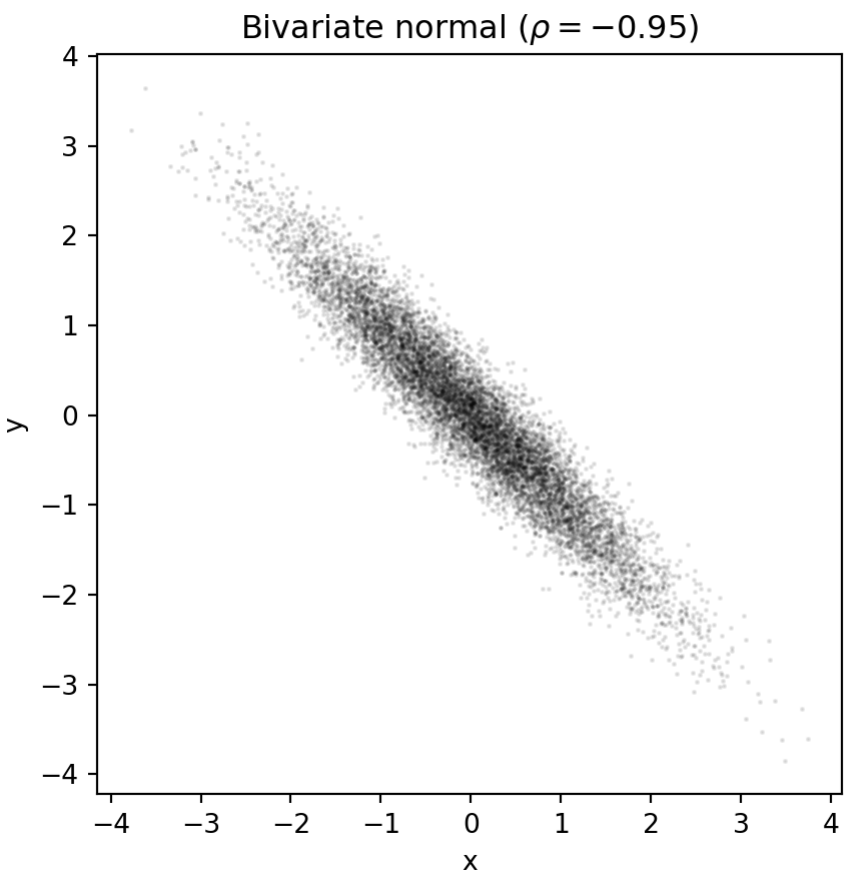
```
rho = 0.75
n = 10000
df = pd.DataFrame({
  "x": np.random.normal(size=n)
}).assign(
  y = lambda d:
    np.random.normal(
      rho*d.x, np.sqrt(1-rho**2),
      size=n
    )
)

fig, ax = plt.subplots(figsize=(5,5))

ax.scatter('x', 'y', c='k', data=df,
           alpha=0.1, s=0.5)

ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_title(f"Bivariate normal ($\\rho={rho
```



pyplot w/ polars

Polars DataFrames can also be used via the `data` argument,

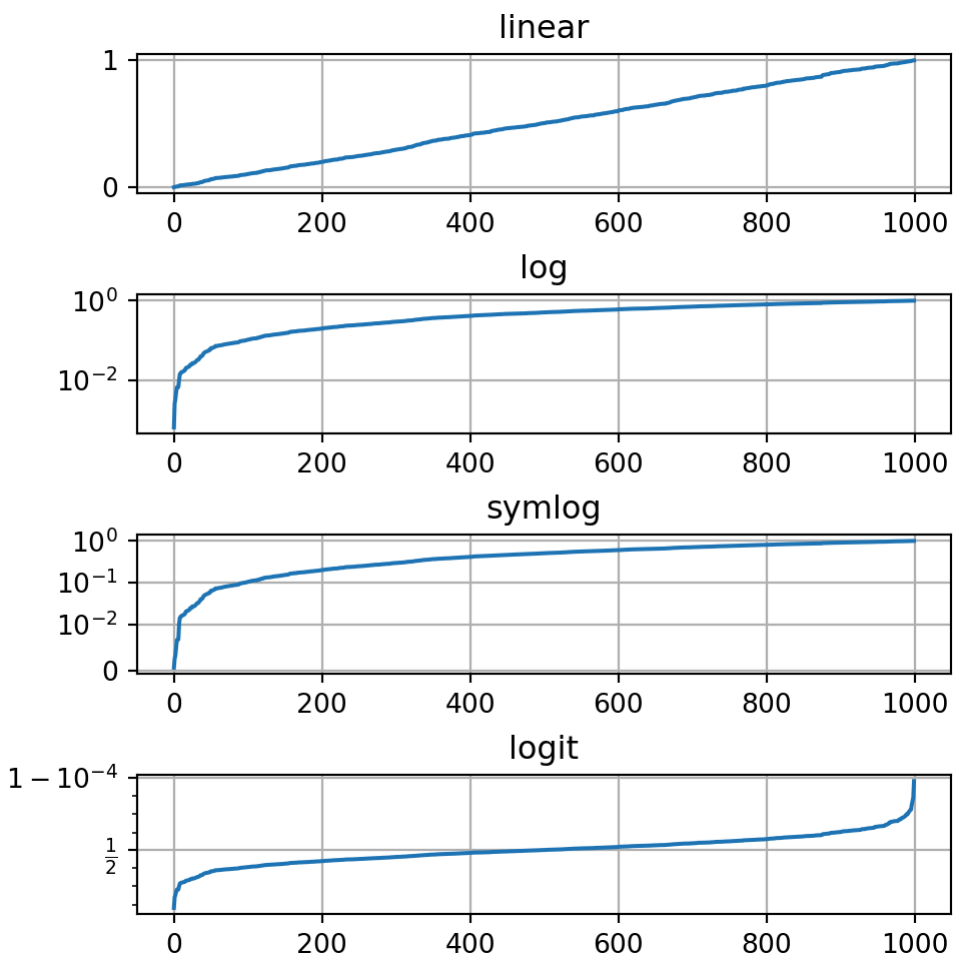
```
rho = -0.95
n = 10000
df = pl.DataFrame({
  "x": np.random.normal(size=n)
}).with_columns(
  y = rho*pl.col("x") +
      np.random.normal(0, np.sqrt(1-rho**2)
)

fig, ax = plt.subplots(figsize=(5,5))

ax.scatter('x', 'y', c='k', data=df,
           alpha=0.1, s=0.5)

ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_title(f"Bivariate normal ($\\rho={rho
```



Scales

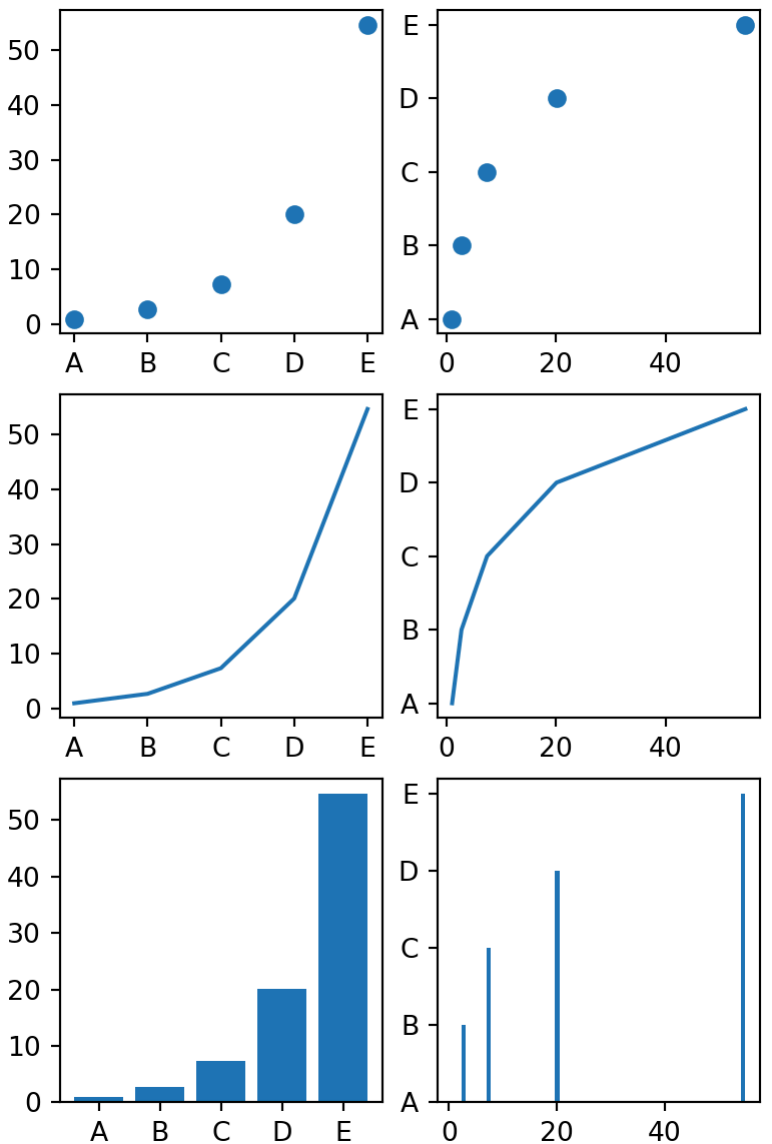
Axis scales can be changed via `plt.xscale()`, `plt.yscale()`, `ax.set_xscale()`, or `ax.set_yscale()`.

```
y = np.sort( np.random.sample(size=1000) )
x = np.arange(len(y))

plt.figure(figsize=(5,5),
           layout="constrained")

scales=['linear', 'log', 'symlog', 'logit']
for i, scale in enumerate(scales):
  plt.subplot(411+i)
  plt.plot(x, y)
  plt.grid(True)
  if scale == 'symlog':
    plt.yscale(scale, linthresh=0.01)
  else:
    plt.yscale(scale)
  plt.title(scale)


plt.show()
```



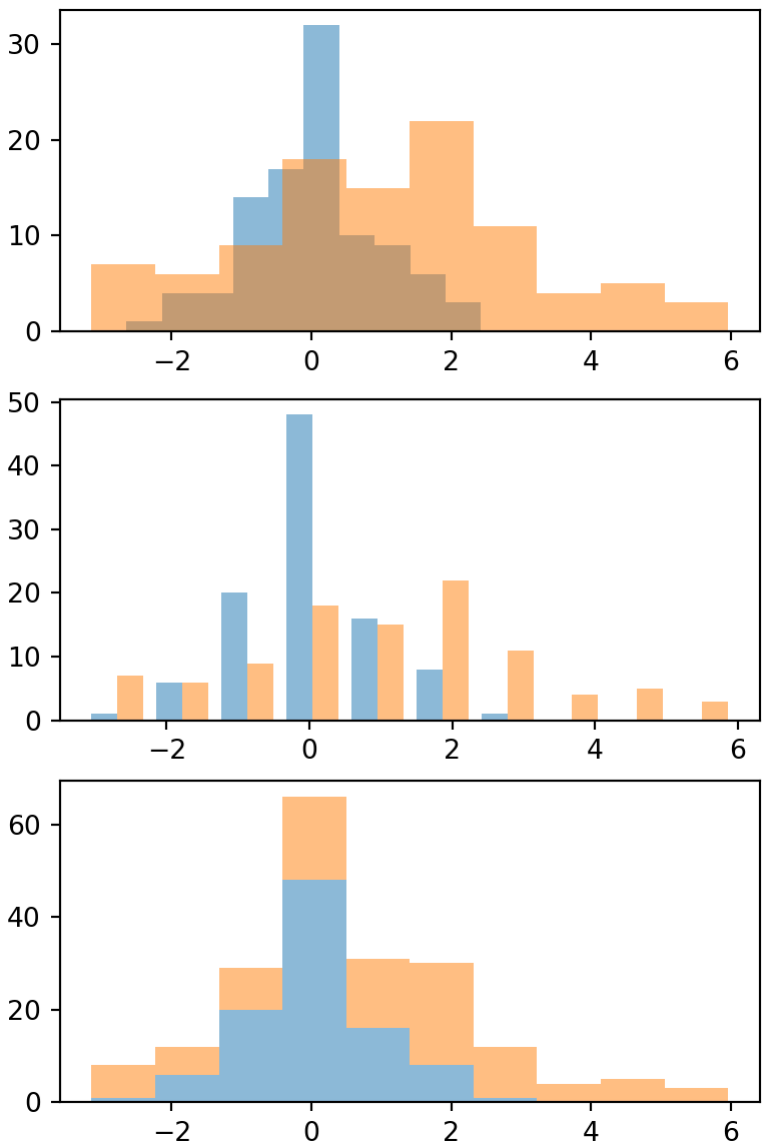
Categorical data

```
df = pd.DataFrame({
  "cat": ["A", "B", "C", "D", "E"],
  "value": np.exp(range(5))
})

plt.figure(figsize=(4, 6), layout="constrai
plt.subplot(321)
plt.scatter("cat", "value", data=df)
plt.subplot(322)
plt.scatter("value", "cat", data=df)
plt.subplot(323)
plt.plot("cat", "value", data=df)
plt.subplot(324)
plt.plot("value", "cat", data=df)
plt.subplot(325)
b = plt.bar("cat", "value", data=df)
plt.subplot(326)
b = plt.bar("value", "cat", data=df)
plt.show()
```

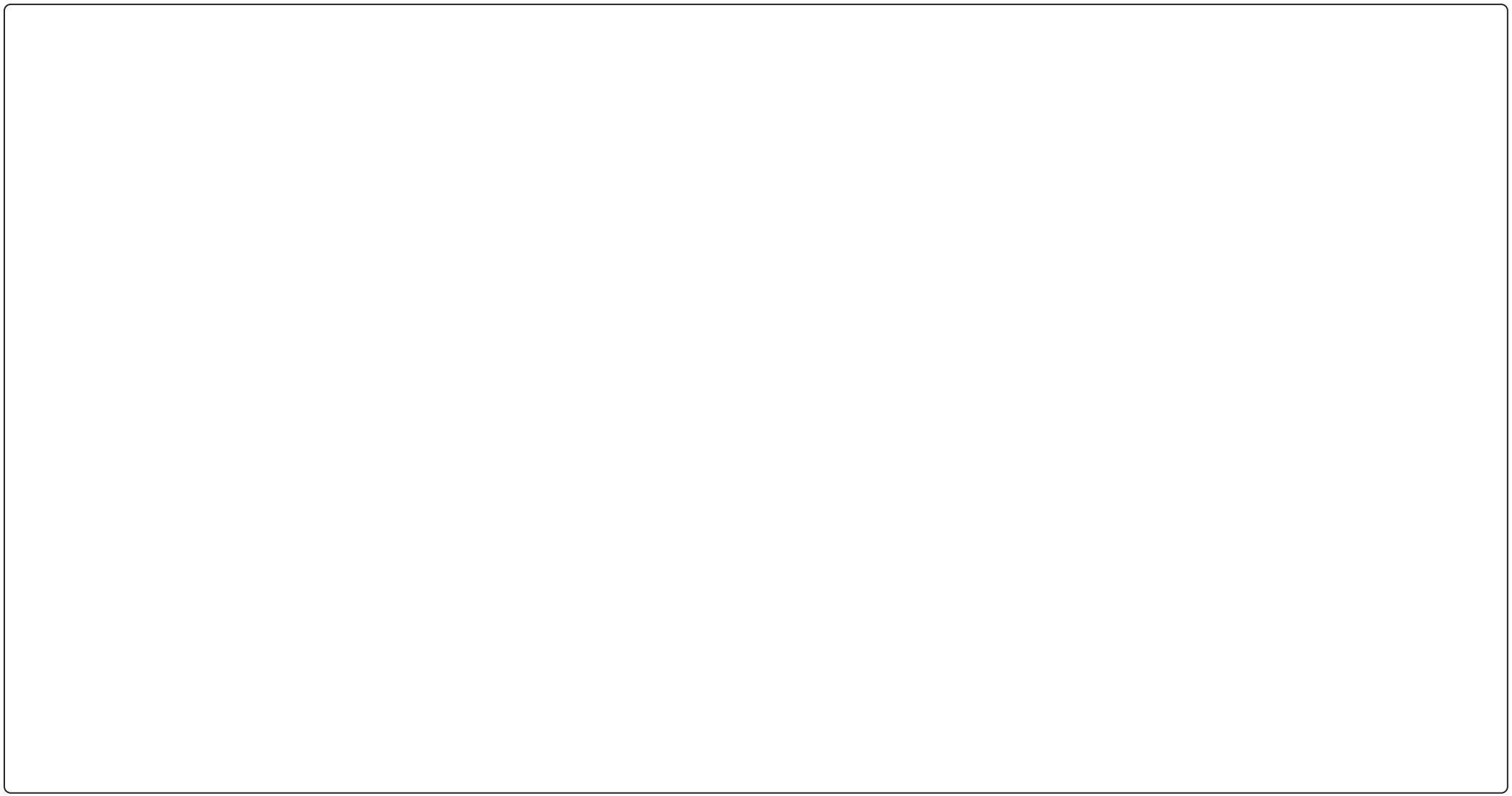


Histograms

```
df = pd.DataFrame({
  "x1": np.random.normal(size=100),
  "x2": np.random.normal(1,2, size=100)
})

plt.figure(figsize=(4, 6), layout="constrai

plt.subplot(311)
h = plt.hist("x1", bins=10, data=df, alpha=
h = plt.hist("x2", bins=10, data=df, alpha=

plt.subplot(312)
h = plt.hist(df, alpha=0.5)

plt.subplot(313)
h = plt.hist(df, stacked=True, alpha=0.5)

plt.show()
```



Other Plot Types



Seaborn

seaborn

> Seaborn is a library for making statistical graphics in Python. It builds on top of **matplotlib** and integrates closely with **pandas** data structures.
>
> Seaborn helps you explore and understand your data. Its plotting functions operate on dataframes and arrays containing whole datasets and internally perform the necessary semantic mapping and statistical aggregation to produce informative plots. Its dataset-oriented, declarative API lets you focus on what the different elements of your plots mean, rather than on the details of how to draw them.

```
1 import seaborn as sns
```

Penguins data

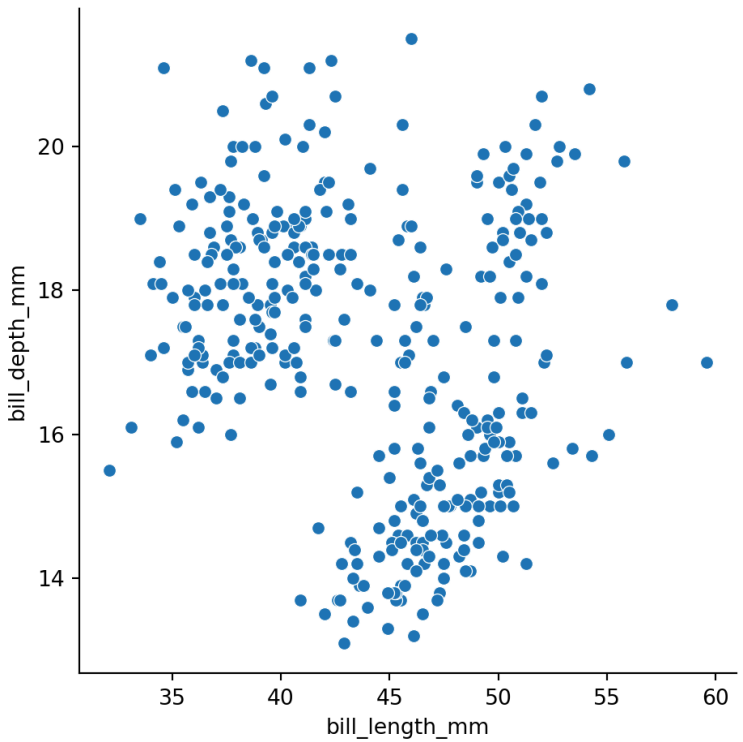
```
1 penguins = sns.load_dataset("penguins"); penguins
```

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	se
0	Adelie	Torgersen	39.1	18.7	181.0	3750.0	Male
1	Adelie	Torgersen	39.5	17.4	186.0	3800.0	Femal
2	Adelie	Torgersen	40.3	18.0	195.0	3250.0	Femal
3	Adelie	Torgersen	NaN	NaN	NaN	NaN	NaN
4	Adelie	Torgersen	36.7	19.3	193.0	3450.0	Femal
...	...	...	...	...	...	...	...
339	Gentoo	Biscoe	NaN	NaN	NaN	NaN	NaN
340	Gentoo	Biscoe	46.8	14.3	215.0	4850.0	Femal
341	Gentoo	Biscoe	50.4	15.7	222.0	5750.0	Male
342	Gentoo	Biscoe	45.2	14.8	212.0	5200.0	Femal
343	Gentoo	Biscoe	49.9	16.1	213.0	5400.0	Male

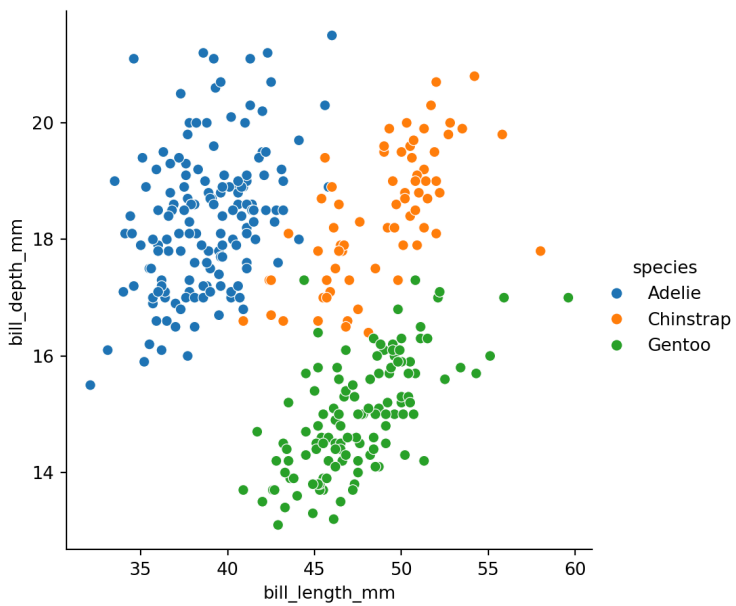
344 rows × 7 columns

Basic plots

```python
g = sns.relplot(
  data = penguins,
  x = "bill_length_mm",
  y = "bill_depth_mm"
)
```




```python
g = sns.relplot(
  data = penguins,
  x = "bill_length_mm",
  y = "bill_depth_mm",
  hue = "species"
)
```



A more complex plot

```
g = sns.relplot(
  data = penguins,
  x = "bill_length_mm", y = "bill_depth_mm", hue = "species",
  col = "island", row = "species"
)
```

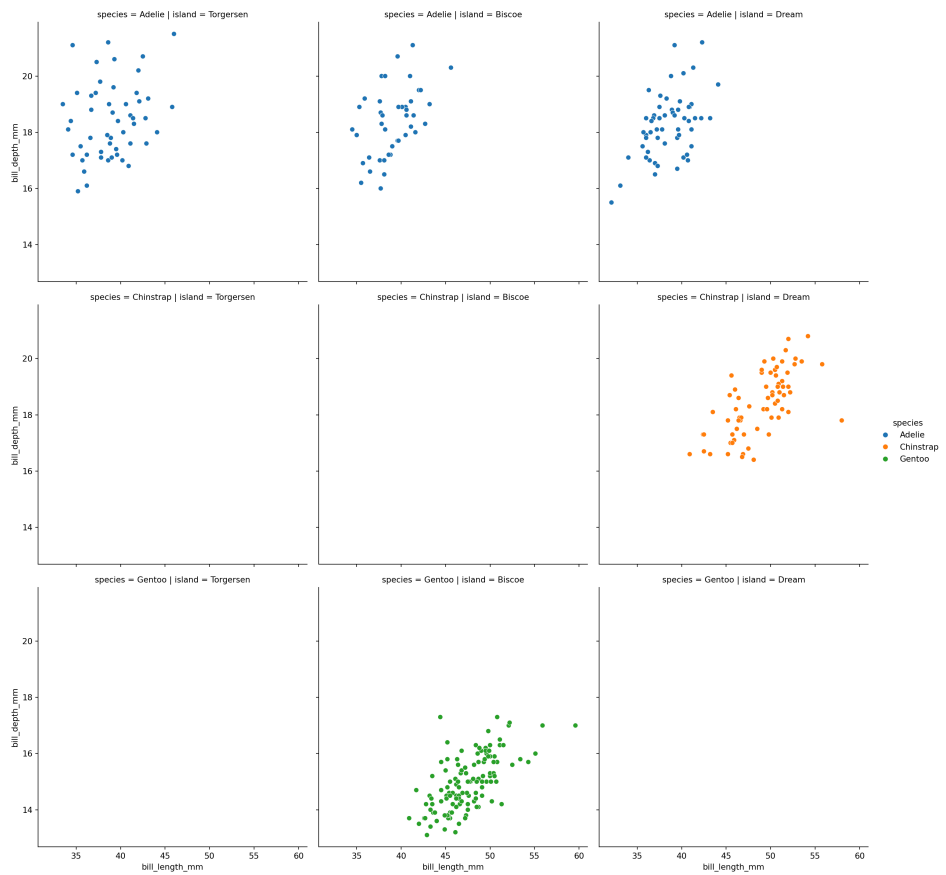
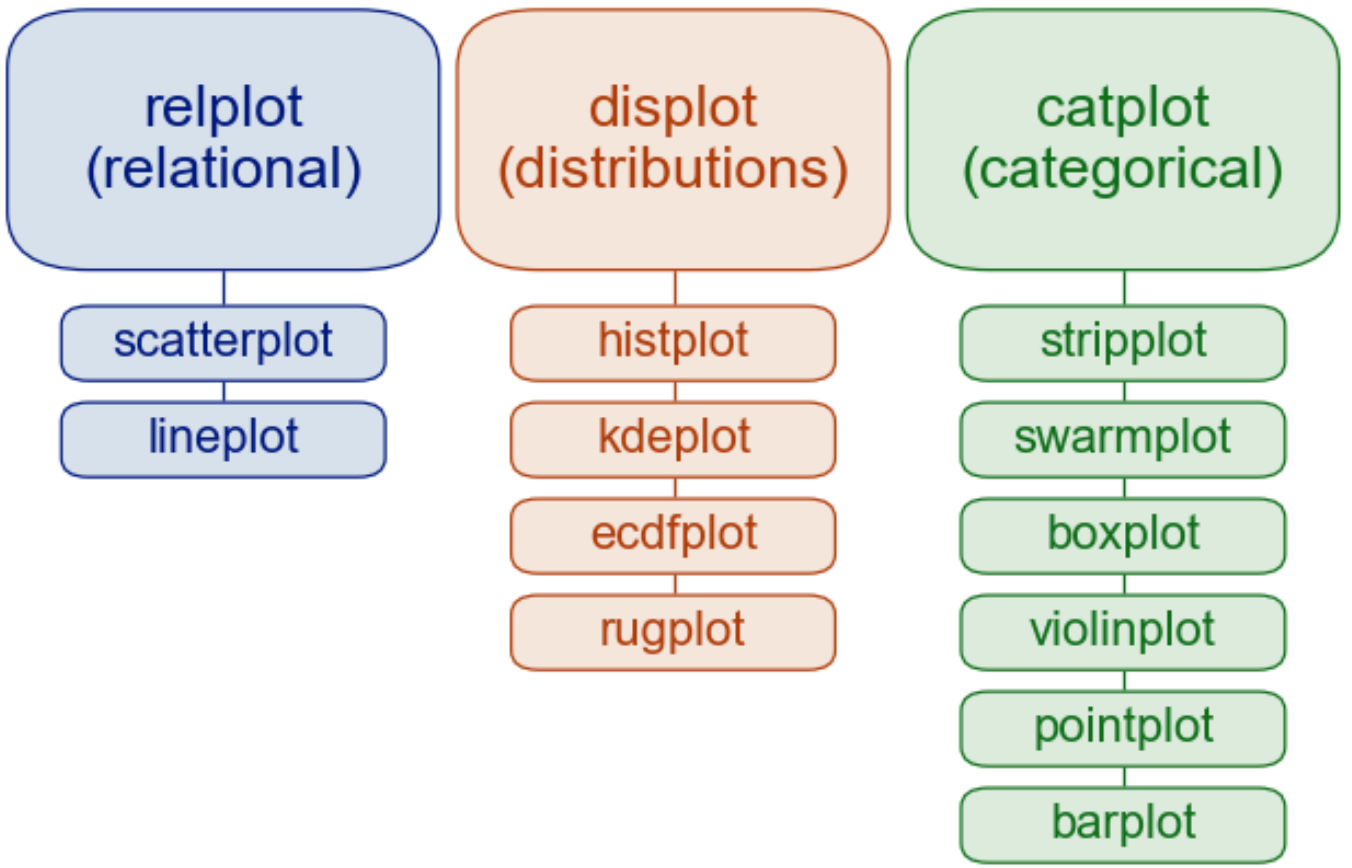
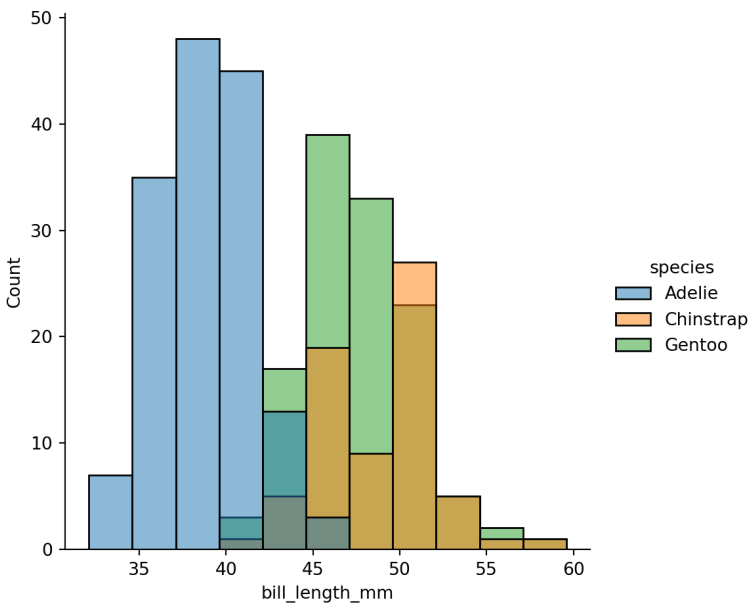


Figure-level vs. axes-level functions

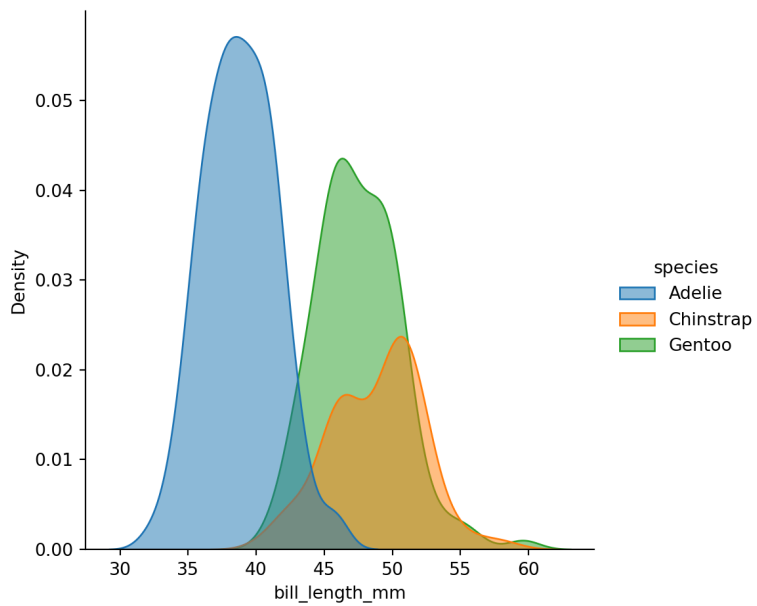


displots

```python
g = sns.displot(
  data = penguins,
  x = "bill_length_mm",
  hue = "species",
  alpha = 0.5
)
```

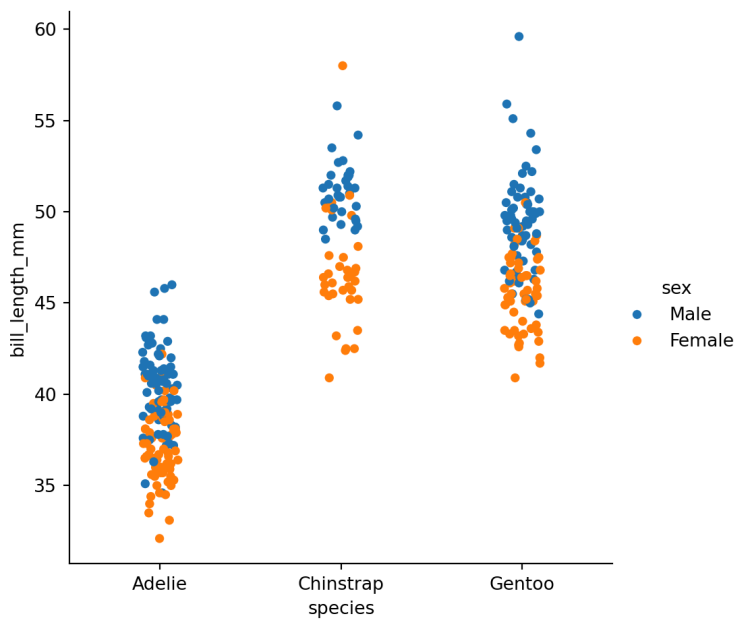



```python
g = sns.displot(
  data = penguins,
  x = "bill_length_mm", hue = "species",
  kind = "kde", fill=True,
  alpha = 0.5
)
```



catplots

```python
g = sns.catplot(
  data = penguins,
  x = "species",
  y = "bill_length_mm",
  hue = "sex"
)
```




```python
g = sns.catplot(
  data = penguins,
  x = "species",
  y = "bill_length_mm",
  hue = "sex",
  kind = "box"
)
```

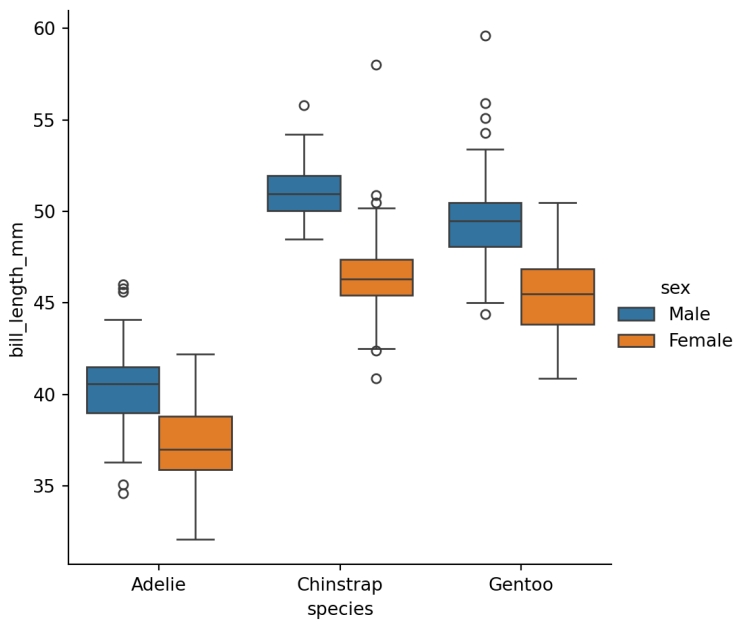
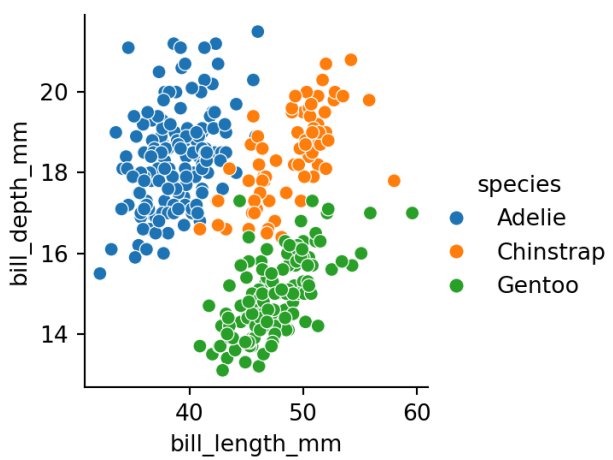


figure-level plot size

To adjust the size of plots generated via a figure-level plotting function, adjust the `aspect` and `height` arguments, where figure width = `aspect * height`.

```
g = sns.relplot(
  data = penguins,
  x = "bill_length_mm", y = "bill_depth_mm",
  hue = "species",
  aspect = 1, height = 3
)
```




```
g = sns.relplot(
  data = penguins,
  x = "bill_length_mm", y = "bill_depth_mm",
  hue = "species",
  aspect = 1, height = 5
)
```

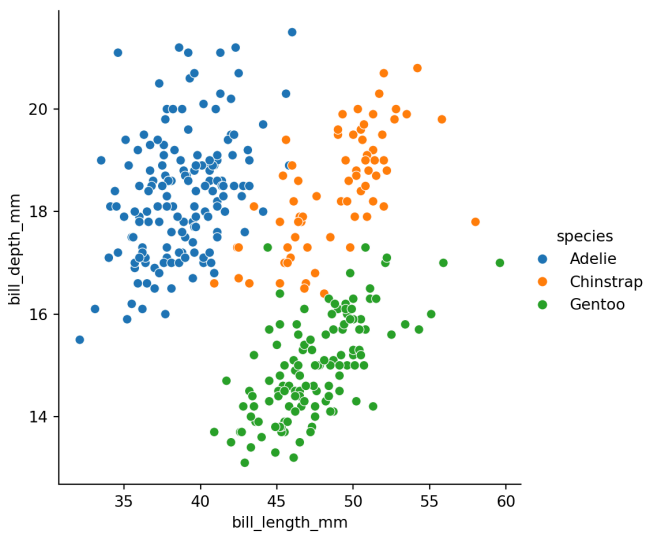
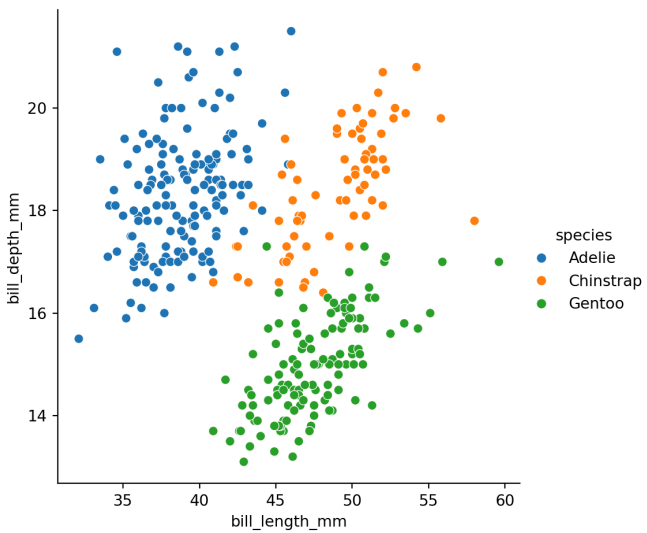


figure-level plots

```python
g = sns.relplot(
  data = penguins,
  x = "bill_length_mm", y = "bill_depth_mm",
  hue = "species",
  aspect = 1
)
```




```python
h = sns.relplot(
  data = penguins,
  x = "bill_length_mm", y = "bill_depth_mm",
  hue = "species", col = "island",
  aspect = 1/2
)
```

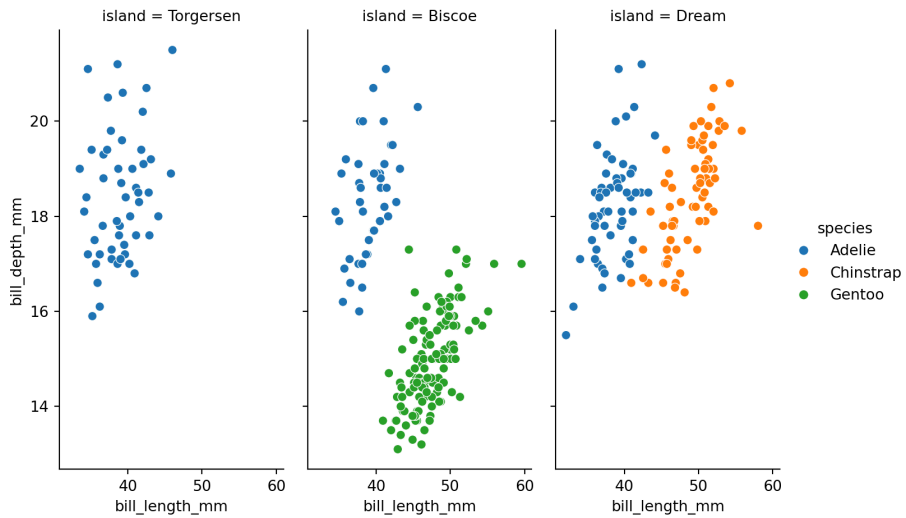


figure-level plot objects

Figure-level plotting methods return a `FacetGrid` object (which is a wrapper around lower-level pyplot figure(s) and axes).

```
1 print(g)
```

```
<seaborn.axisgrid.FacetGrid object at 0x370dbc910>
```

```
1 print(h)
```

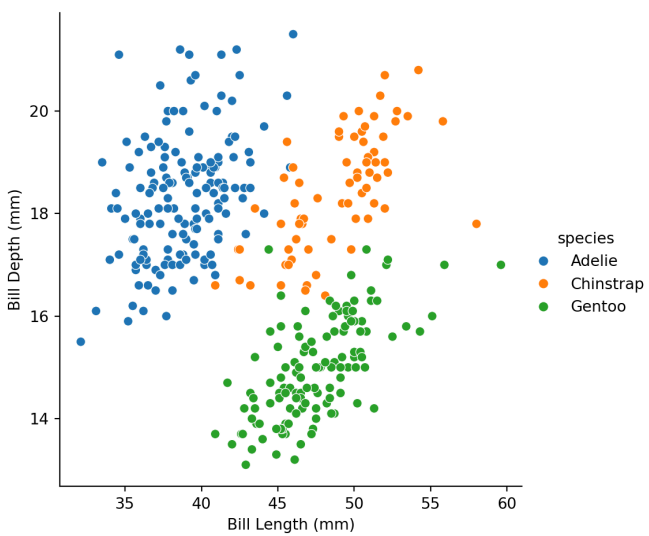
```
<seaborn.axisgrid.FacetGrid object at 0x370d2b390>
```

FacetGrid methods

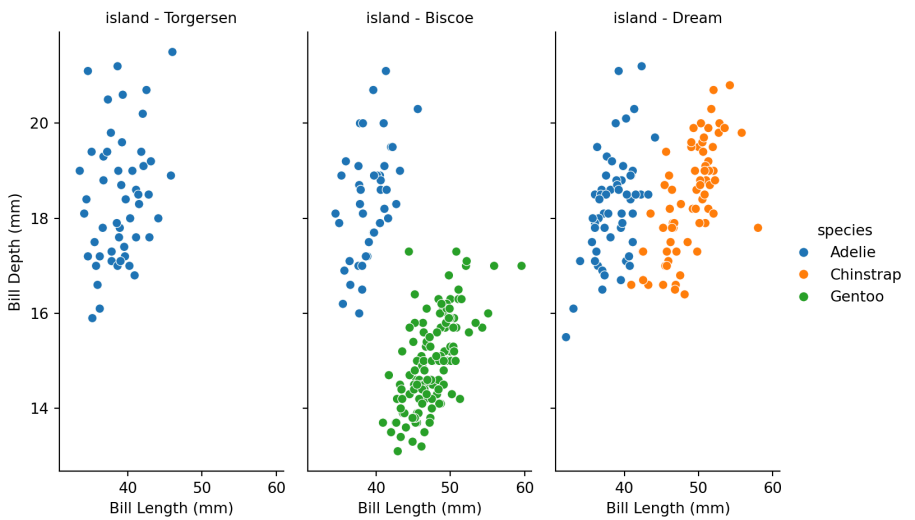
Method	Description
`add_legend()`	Draw a legend, maybe placing it outside axes and resizing the figure
`despine()`	Remove axis spines from the facets.
`facet_axis()`	Make the axis identified by these indices active and return it.
`facet_data()`	Generator for name indices and data subsets for each facet.
`map()`	Apply a plotting function to each facet's subset of the data.
`map_dataframe()`	Like `.map()` but passes args as strings and inserts data in kwargs.
`refline()`	Add a reference line(s) to each facet.
`savefig()`	Save an image of the plot.
`set()`	Set attributes on each subplot Axes.
`set_axis_labels()`	Set axis labels on the left column and bottom row of the grid.
`set_titles()`	Draw titles either above each facet or on the grid margins.
`set_xlabels()`	Label the x axis on the bottom row of the grid.
`set_xticklabels()`	Set x axis tick labels of the grid.
`set_ylabels()`	Label the y axis on the left column of the grid.
`set_yticklabels()`	Set y axis tick labels on the left column of the grid.
`tight_layout()`	Call fig.tight_layout within rect that excludes the legend.

Adjusting labels

```python
g = sns.relplot(
  data = penguins,
  x = "bill_length_mm", y = "bill_depth_mm",
  hue = "species",
  aspect = 1
).set_axis_labels(
  "Bill Length (mm)",
  "Bill Depth (mm)"
)
```




```python
g = sns.relplot(
  data = penguins,
  x = "bill_length_mm", y = "bill_depth_mm",
  hue = "species", col = "island",
  aspect = 1/2
).set_axis_labels(
  "Bill Length (mm)",
  "Bill Depth (mm)"
).set_titles(
  "{col_var} - {col_name}"
)
```

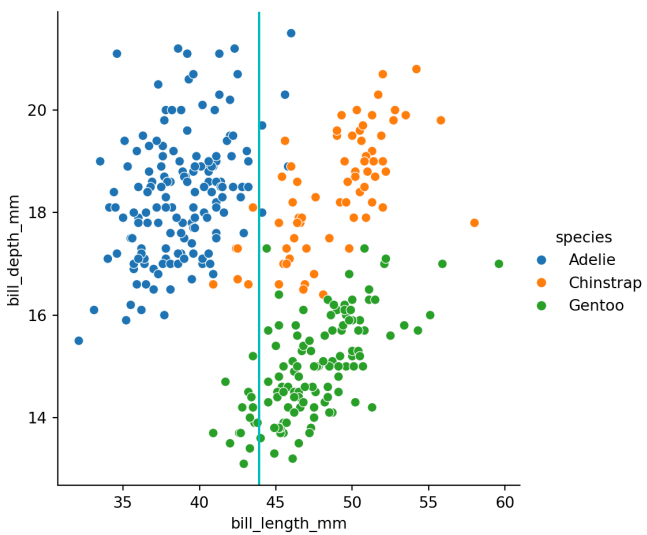


FacetGrid attributes

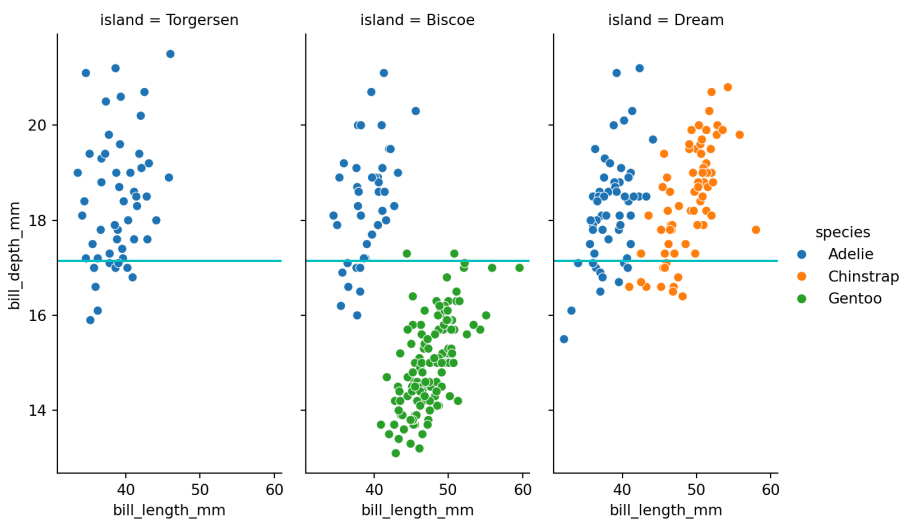
Attribute	Description
`ax`	The `matplotlib.axes.Axes` when no faceting variables are assigned.
`axes`	An array of the `matplotlib.axes.Axes` objects in the grid.
`axes_dict`	A mapping of facet names to corresponding `matplotlib.axes.Axes`.
`figure`	Access the `matplotlib.figure.Figure` object underlying the grid.
`legend`	The `matplotlib.legend.Legend` object, if present.

Using axes to modify plots

```python
g = sns.relplot(
  data = penguins,
  x = "bill_length_mm", y = "bill_depth_mm",
  hue = "species",
  aspect = 1
)
g.ax.axvline(
  x = penguins.bill_length_mm.mean(), c = "c"
)
```




```python
h = sns.relplot(
  data = penguins,
  x = "bill_length_mm", y = "bill_depth_mm",
  hue = "species", col = "island",
  aspect = 1/2
)
mean_bill_dep = penguins.bill_depth_mm.mean()

for ax in h.axes.flat:
    ax.axhline(y=mean_bill_dep, c = "c")
```



Why figure-level functions?

Advantages:

- Easy faceting by data variables
- Legend outside of plot by default
- Easy figure-level customization
- Different figure size parameterization

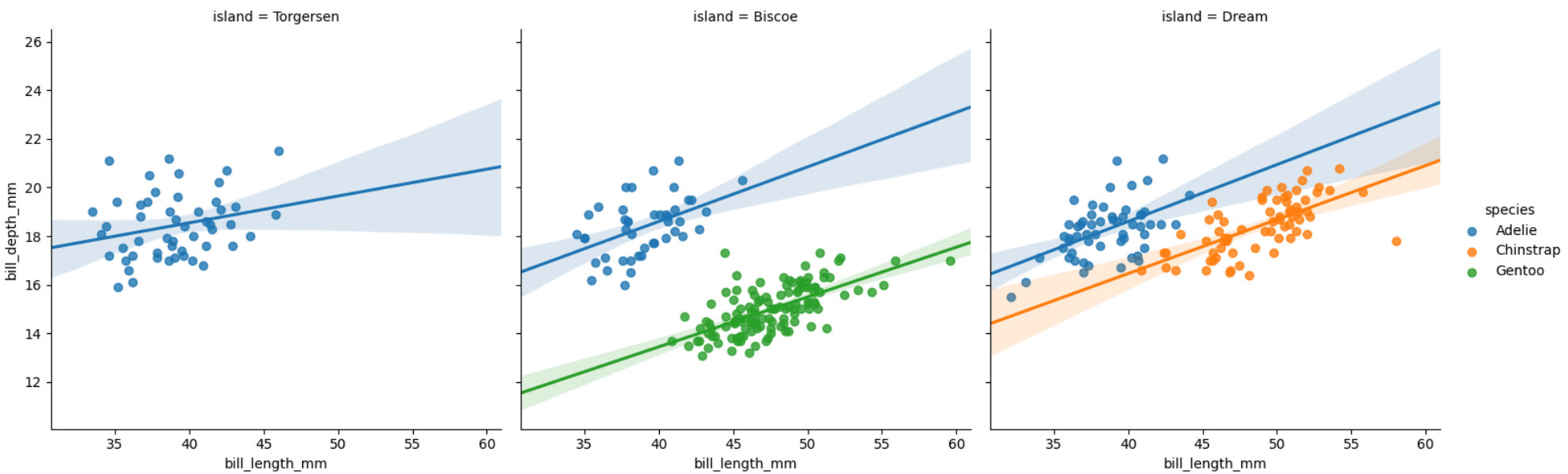
Disadvantages:

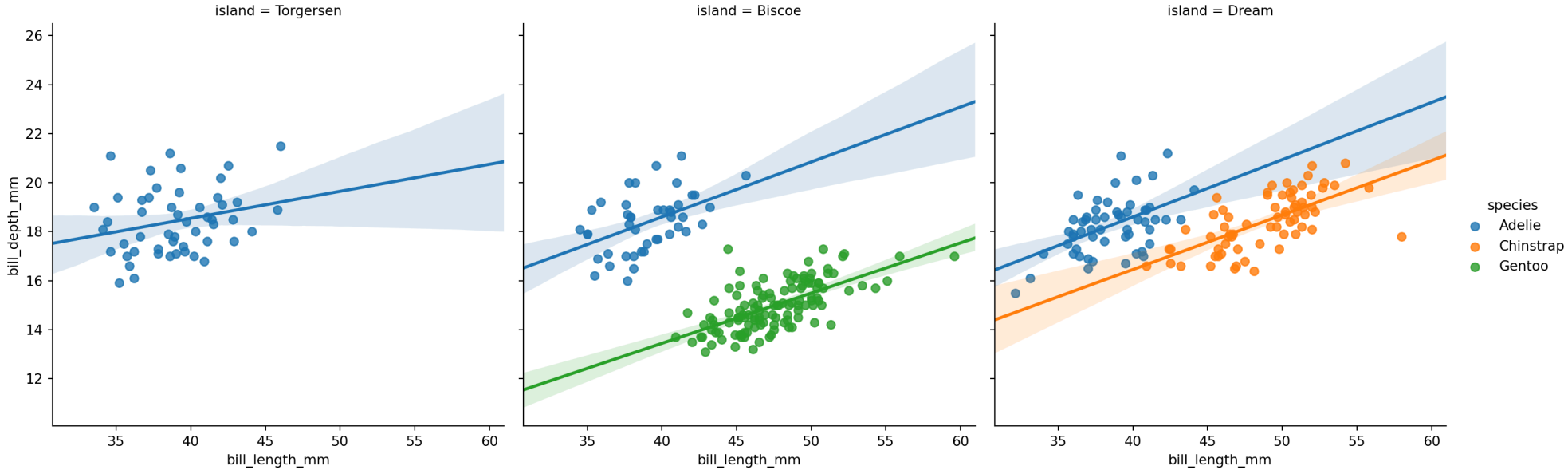
- Many parameters not in function signature
- Cannot be part of a larger matplotlib figure
- Different API from matplotlib
- Different figure size parameterization

lmplots

There is one additional figure-level plot type - `lmplot()`, which is a convenient interface to fitting and plotting regression models across subsets of data,

```
sns.lmplot(
  data = penguins,
  x = "bill_length_mm", y = "bill_depth_mm",
  hue = "species", col = "island",
  aspect = 1, truncate = False
)
```



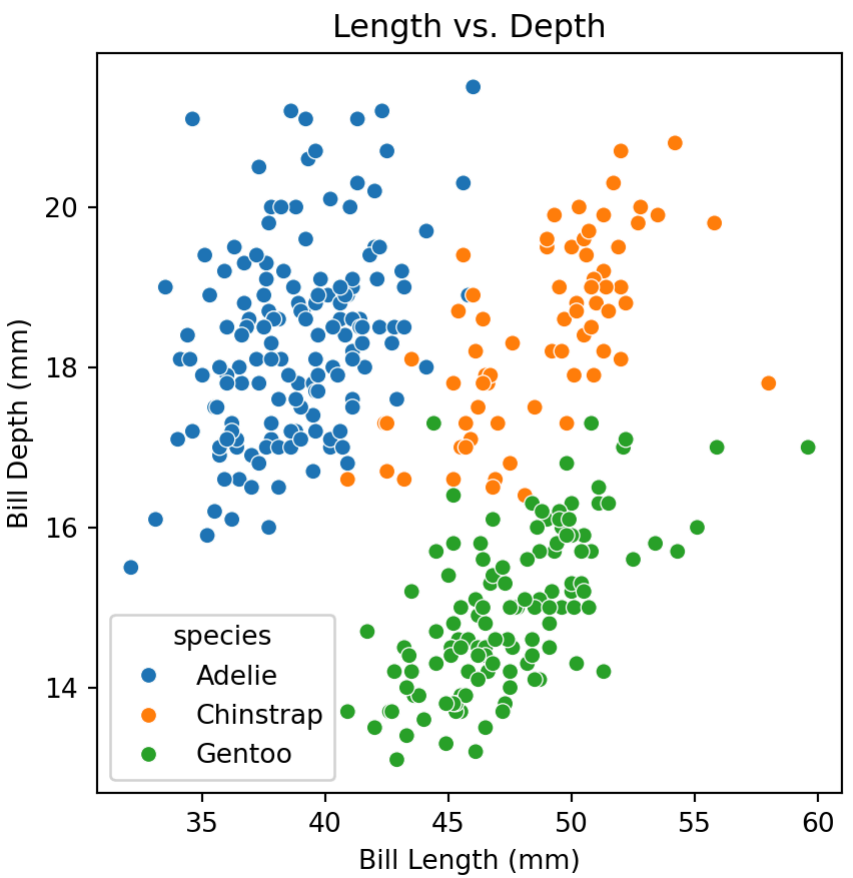

island = Torgersen
island = Biscoe
island = Dream
bill_depth_mm
bill_length_mm
26
24
22
20
18
16
14
12
35
40
45
50
55
60
species
Adelie
Chinstrap
Gentoo

axes-level plots

axes-level functions

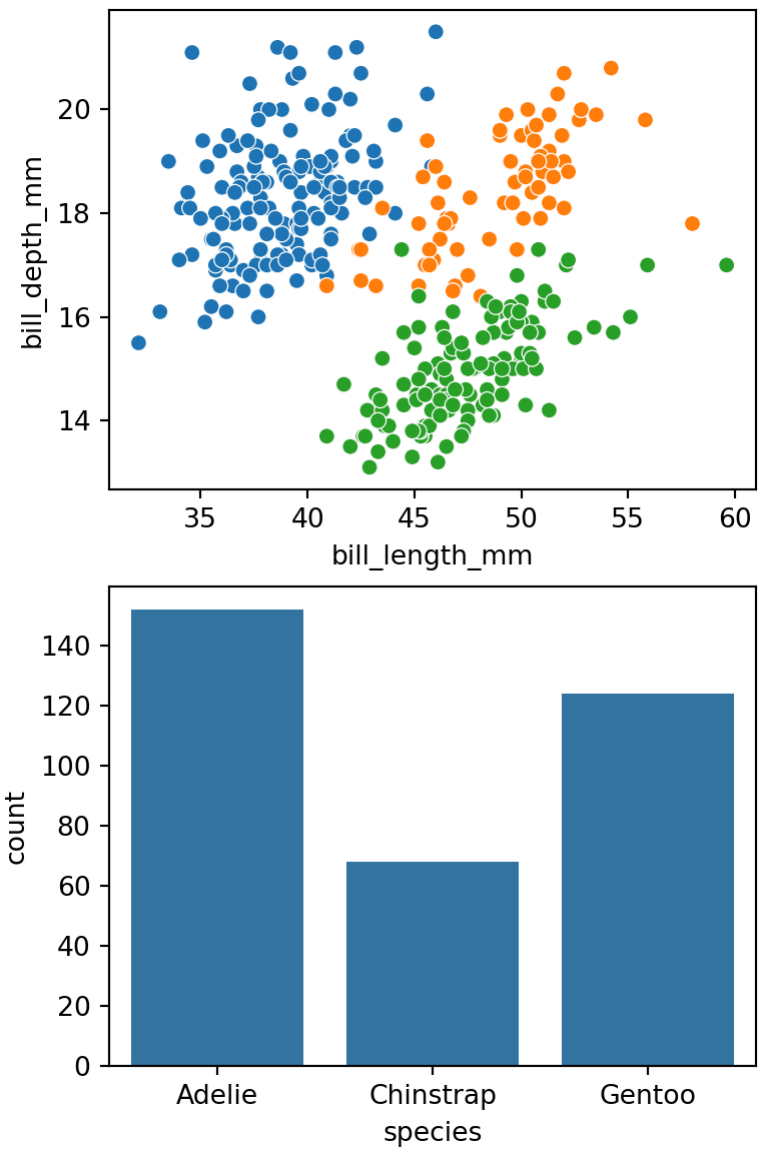
These functions return a `matplotlib.axes.Axes` object instead of a `FacetGrid`, giving more direct control over the plot using basic matplotlib tools.

```
plt.figure(figsize=(5,5))

sns.scatterplot(
  data = penguins,
  x = "bill_length_mm",
  y = "bill_depth_mm",
  hue = "species"
)

plt.xlabel("Bill Length (mm)")
plt.ylabel("Bill Depth (mm)")
plt.title("Length vs. Depth")

plt.show()
```



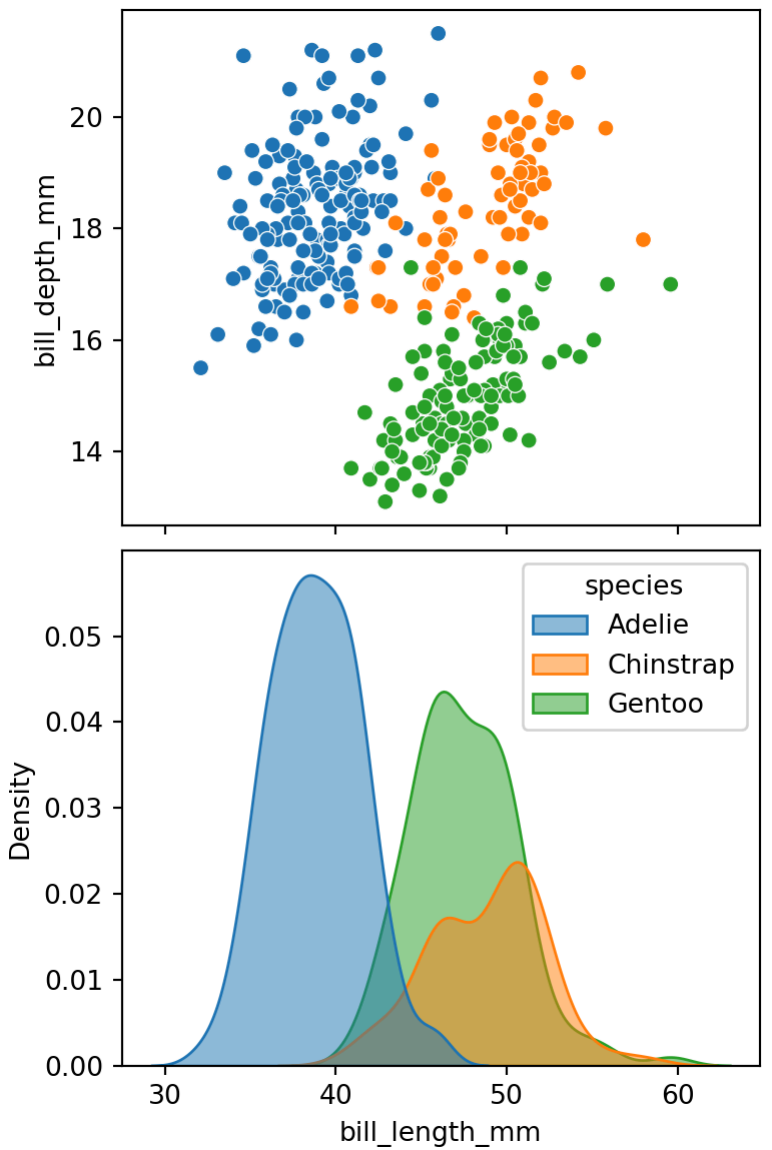
subplots - pyplot style

```
plt.figure(
  figsize=(4,6),
  layout = "constrained"
)

plt.subplot(211)
sns.scatterplot(
  data = penguins,
  x = "bill_length_mm",
  y = "bill_depth_mm",
  hue = "species"
)
plt.legend().remove()

plt.subplot(212)
sns.countplot(
  data = penguins,
  x = "species"
)

```



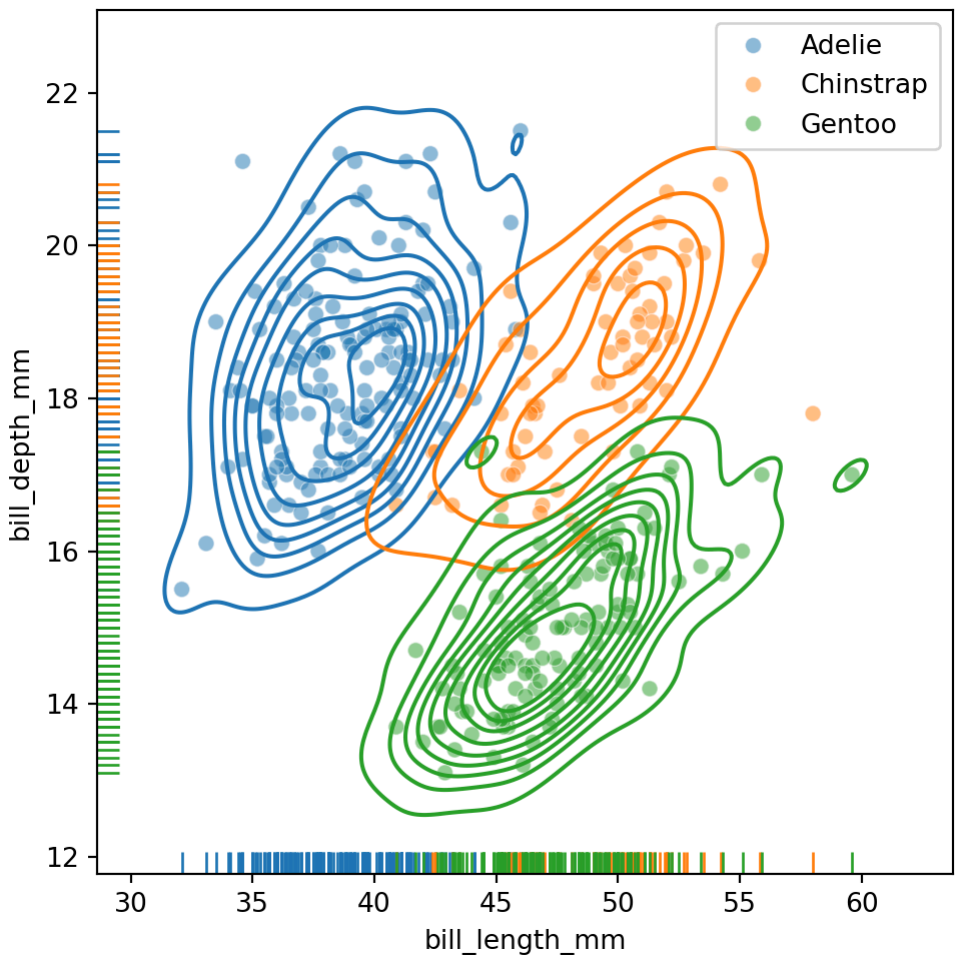
subplots - OO style

```
fig, axs = plt.subplots(
  2, 1, figsize=(4,6),
  layout = "constrained",
  sharex=True
)

sns.scatterplot(
  data = penguins,
  x = "bill_length_mm", y = "bill_dept
  hue = "species",
  ax = axs[0]
)
axs[0].get_legend().remove()

sns.kdeplot(
  data = penguins,
  x = "bill_length_mm", hue = "species
  fill=True, alpha=0.5,
  ax = axs[1]
)
```



layering plots

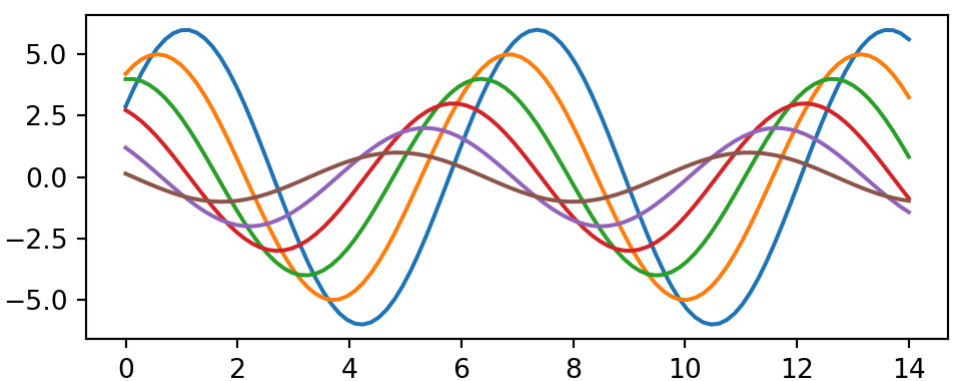
```
plt.figure(figsize=(5,5),
           layout = "constrained")

sns.kdeplot(
  data = penguins,
  x = "bill_length_mm", y = "bill_depth_mm",
  hue = "species"
)
sns.scatterplot(
  data = penguins,
  x = "bill_length_mm", y = "bill_depth_mm",
  hue = "species", alpha=0.5
)
sns.rugplot(
  data = penguins,
  x = "bill_length_mm", y = "bill_depth_mm",
  hue = "species"
)
plt.legend()

plt.show()
```



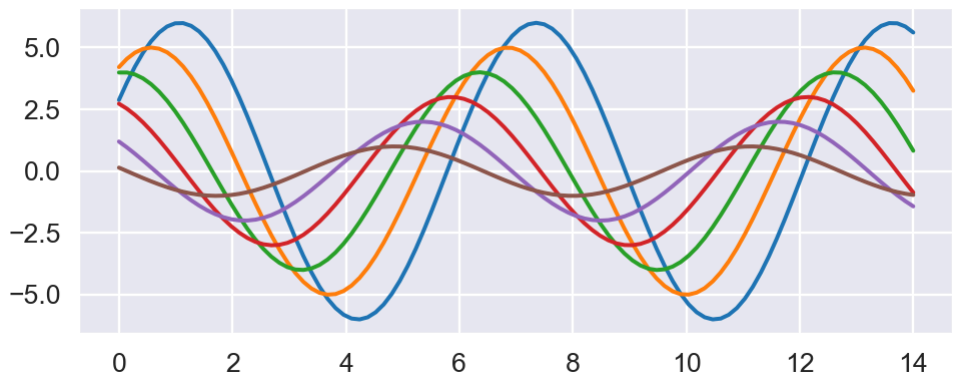
Themes

Seaborn comes with a number of themes (`darkgrid`, `whitegrid`, `dark`, `white`, and `ticks`) which can be enabled at the figure level with `sns.set_theme()` or at the axes level with `sns.axes_style()`.

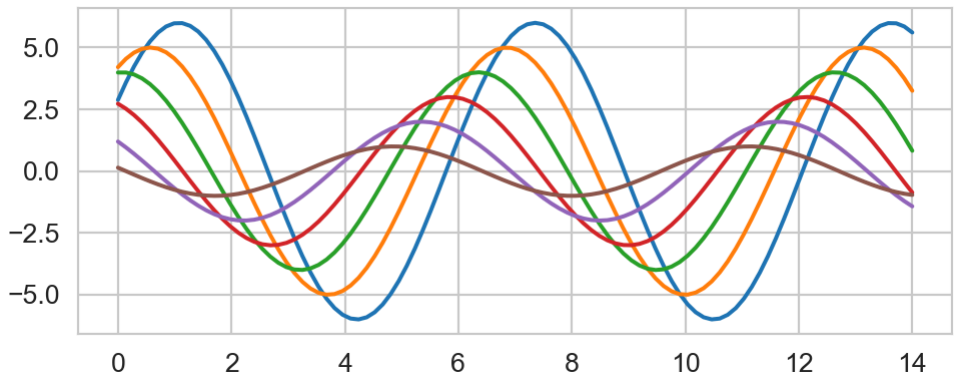
```
def sinplot():
    plt.figure(figsize=(5,2), layout = "cons
    x = np.linspace(0, 14, 100)
    for i in range(1, 7):
        plt.plot(x, np.sin(x + i * .5) * (7
    plt.show()

sinplot()
```



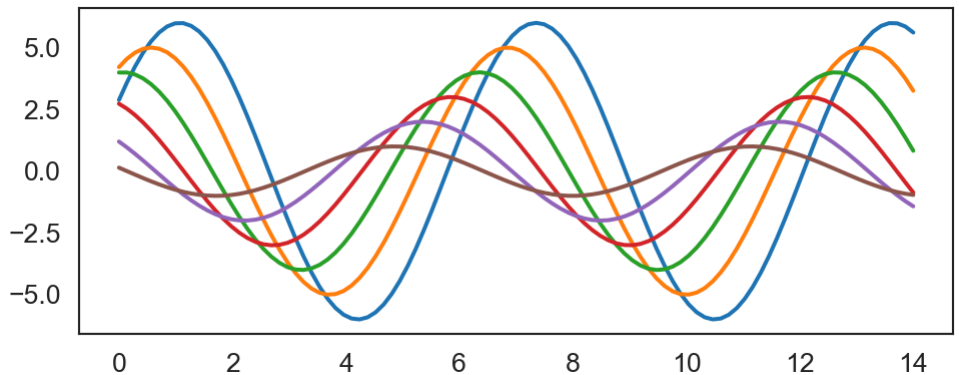

```
with sns.axes_style("darkgrid"):
  sinplot()
```



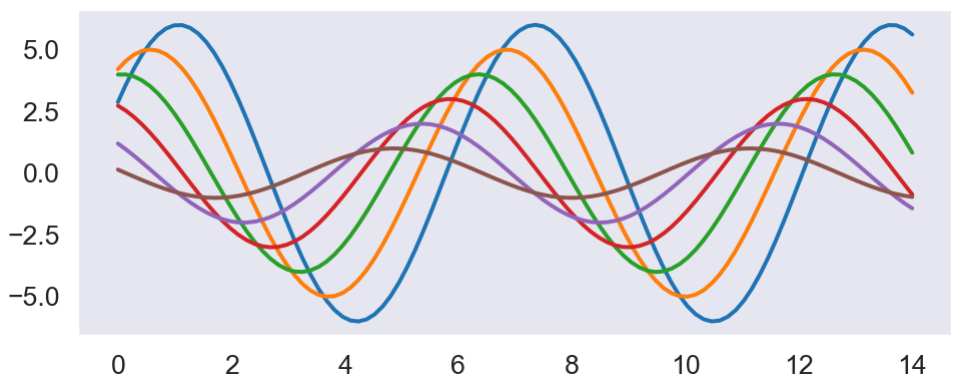
```
with sns.axes_style("whitegrid"):
  sinplot()
```



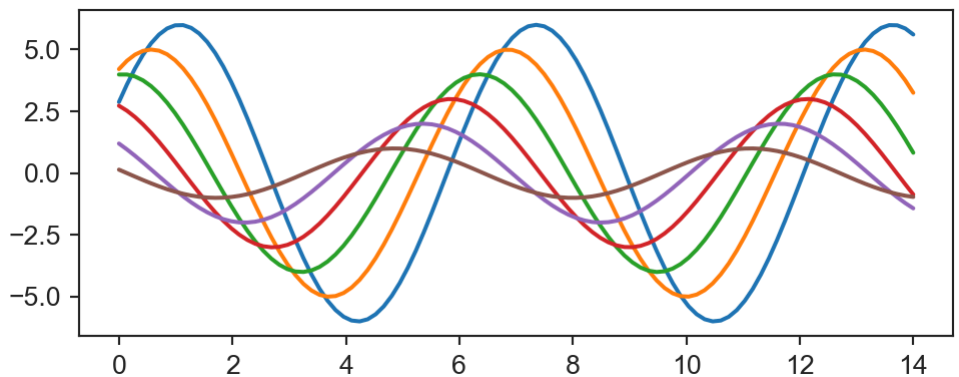

```
with sns.axes_style("white"):
  sinplot()
```




```
with sns.axes_style("dark"):
  sinplot()
```

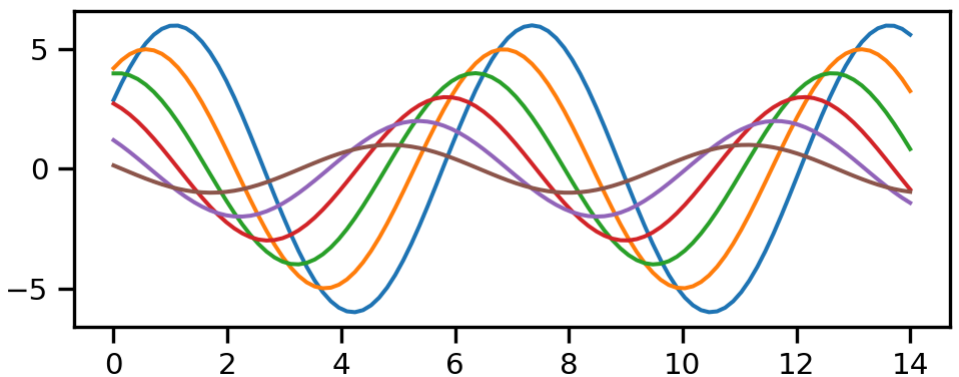



```
with sns.axes_style("ticks"):
  sinplot()
```

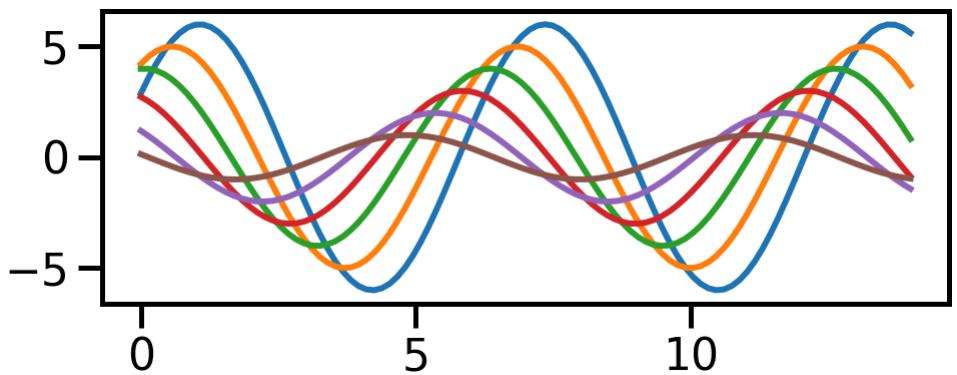


Context

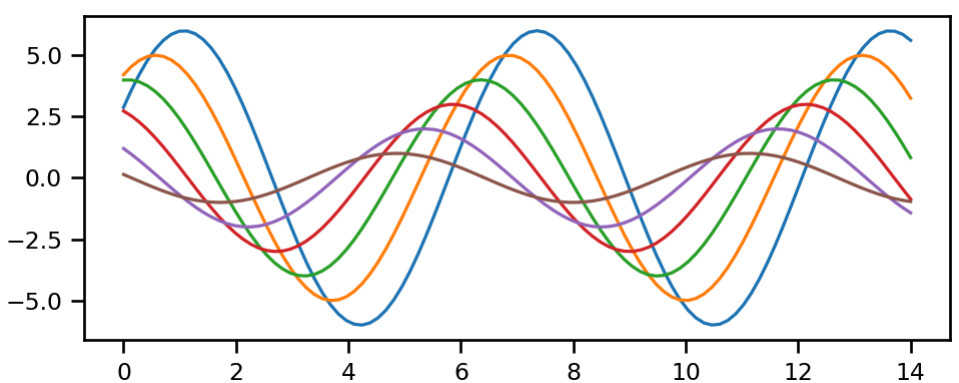
```
1 sns.set_context("notebook")
2 sinplot()
```



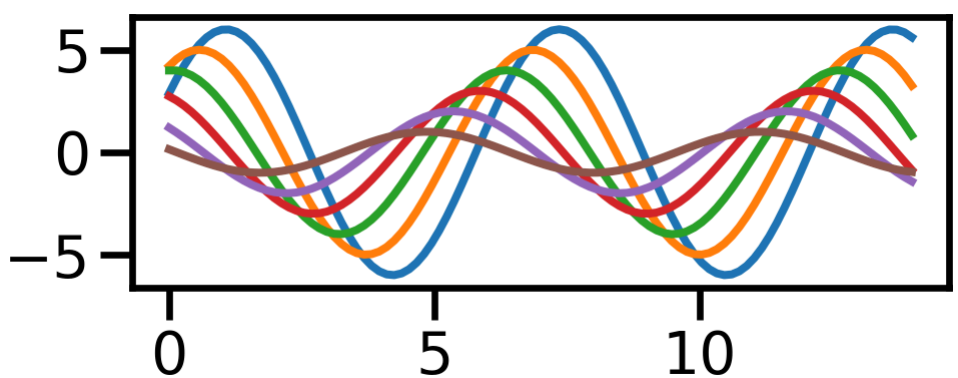

```
1 sns.set_context("talk")
2 sinplot()
```




```
1
2 sns.set_context("paper")
3 sinplot()
```




```
1 sns.set_context("poster")
2 sinplot()
```



Color palettes

All of the examples below use `sns.color_palette()`. The continuous palettes additionally use `as_cmap=True`,

```
1 show_palette()
```

```
1 show_palette("husl")
```

```
1 show_palette("tab10")
```

```
1 show_palette("Set2")
```

```
1 show_palette("hls")
```

```
1 show_palette("Paired")
```

Continuous palettes

```
1 show_cont_palette("viridis")
```

```
1 show_cont_palette("YlOrBr")
```

```
1 show_cont_palette("cubehelix")
```

```
1 show_cont_palette("vlag")
```

```
1 show_cont_palette("light:b")
```

```
1 show_cont_palette("mako")
```

```
1 show_cont_palette("dark:salmon_r")
```

```
1 show_cont_palette("rocket")
```

Applying palettes

Palettes are applied via the `set_palette()` function,

seaborn objects interface

seaborn.objects

> The `seaborn.objects` interface is a newer declarative API (v0.12+) for composing plots from layers of marks, stats, and moves. It aims to support end-to-end plot specification and customization without dropping down to matplotlib.

```
1 import seaborn.objects as so
```

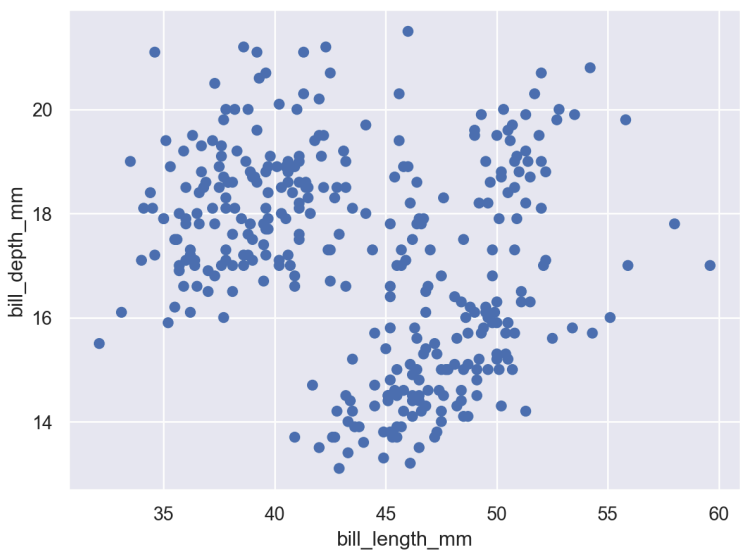
The key building blocks are:

- **`Plot`** - the core object, initialized with data and variable mappings
- **`Mark`** - visual representations (e.g., `Dot`, `Line`, `Bar`, `Area`)
- **`Stat`** - statistical transforms (e.g., `Agg`, `Hist`, `Est`)
- **`Move`** - positional adjustments (e.g., `Dodge`, `Jitter`, `Stack`)
- **`Scale`** - controls data-to-visual mappings (e.g., `Continuous`, `Nominal`)

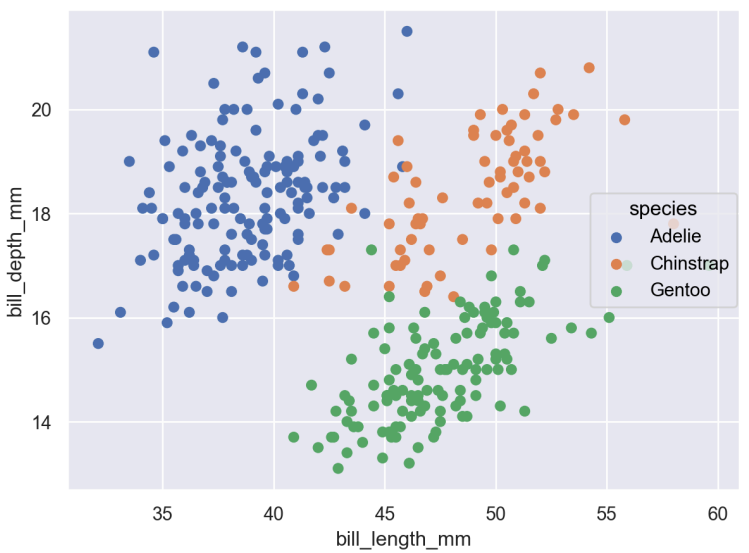
Building plots

Plots are built by chaining `.add()` calls, each specifying a layer with a mark and optional stat/move,

```
( so.Plot(
    penguins,
    x="bill_length_mm", y="bill_depth_mm"
  )
  .add(so.Dot())
).show()
```



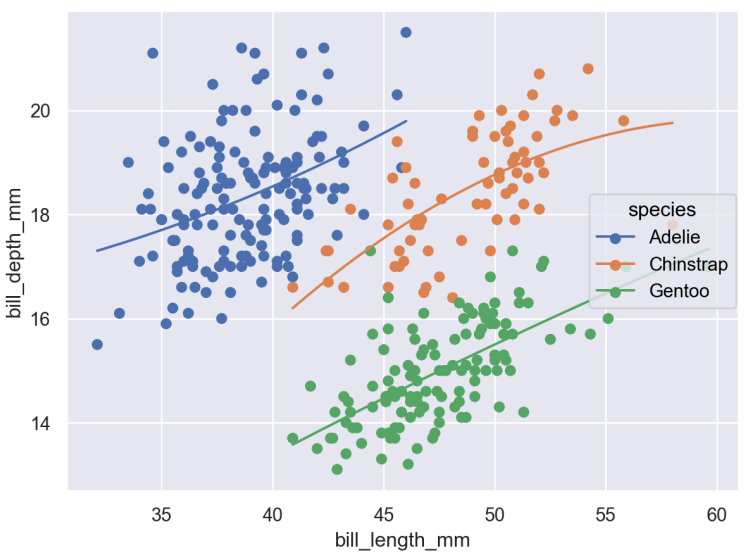

```
( so.Plot(
    penguins,
    x="bill_length_mm", y="bill_depth_mm",
    color="species"
  )
  .add(so.Dot())
).show()
```



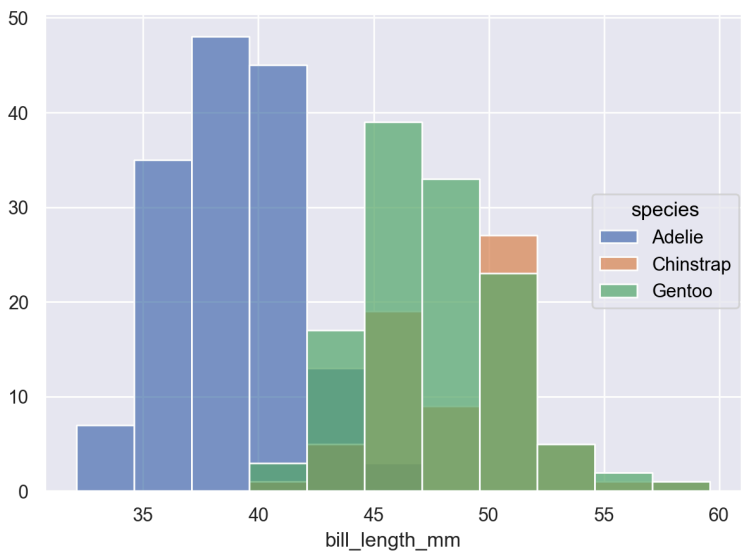
Layering and stats

Multiple `.add()` calls create layers, and `Stat` objects transform data before rendering,

```
( so.Plot(
    penguins,
    x="bill_length_mm", y="bill_depth_mm",
    color="species"
  )
  .add(so.Dot())
  .add(so.Line(), so.PolyFit())
).show()
```



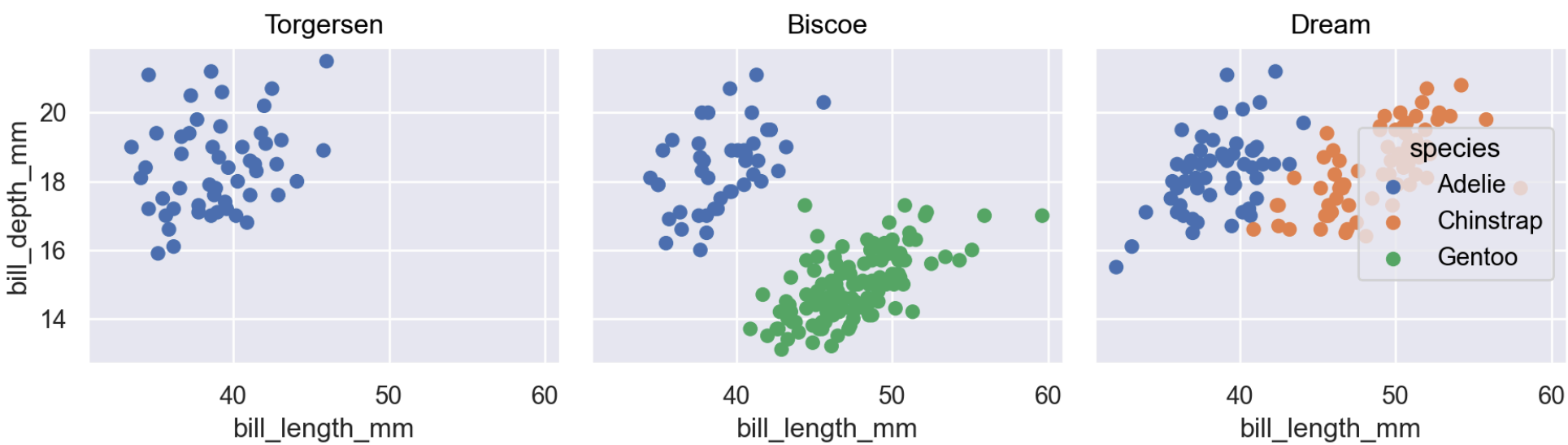

```
( so.Plot(
    penguins,
    x="bill_length_mm",
    color="species"
  )
  .add(so.Bars(), so.Hist())
).show()
```



Faceting

The `.facet()` method creates subplots by data variables,

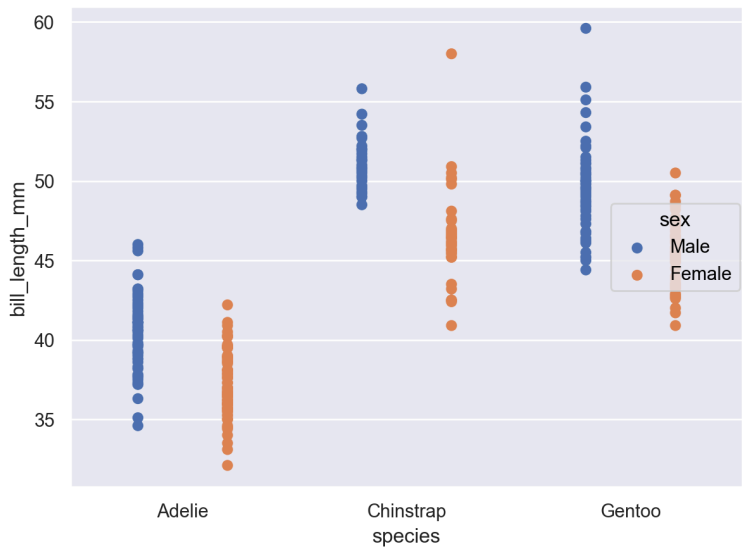
```
( so.Plot(
    penguins,
    x="bill_length_mm", y="bill_depth_mm",
    color="species"
  )
  .facet(col="island")
  .add(so.Dot())
  .layout(size=(10, 3))
).show()
```



Moves and scales

`Move` objects adjust positions (e.g., dodging, jittering, stacking) and `Scale` objects control data-to-visual mappings,

```
( so.Plot(
    penguins,
    x="species",
    y="bill_length_mm",
    color="sex"
  )
  .add(so.Dot(), so.Dodge())
).show()
```




```
( so.Plot(
    penguins,
    x="species",
    y="bill_length_mm",
    color="sex"
  )
  .add(so.Dot(), so.Jitter())
  .scale(color="Set2")
).show()
```