seaborn

Lecture 11

Dr. Colin Rundel

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 matplotlib.pyplot as plt
2 import seaborn as sns
```

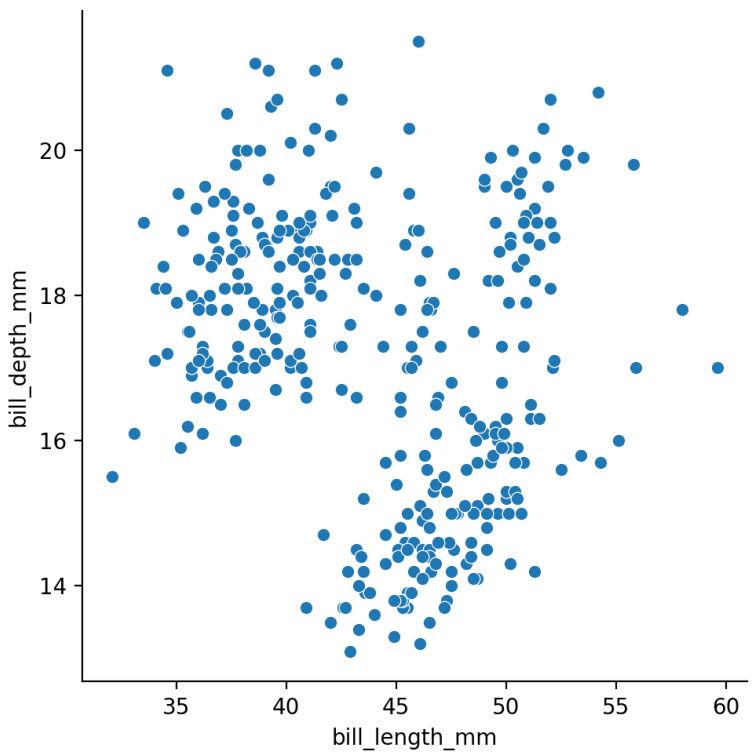
Penguins data

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

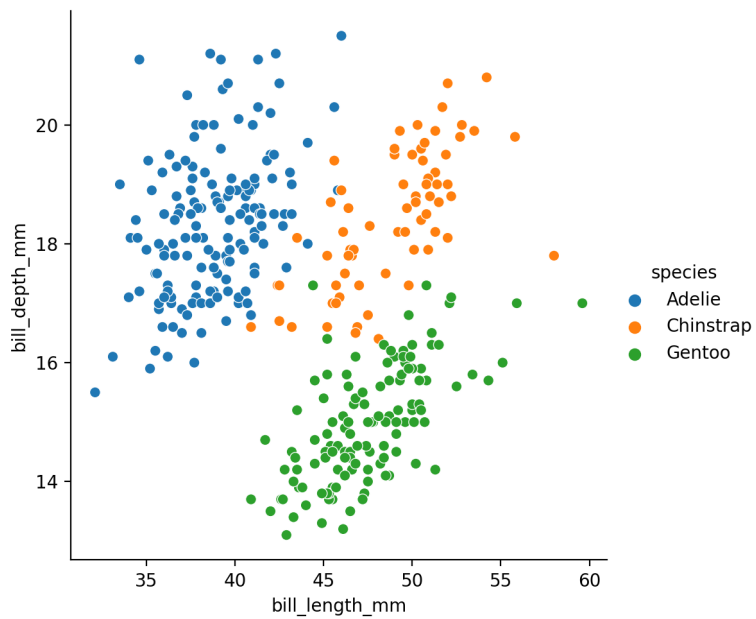
```
     species     island  bill_length_mm  ...  flipper_length_mm  body_mass_g     sex
0     Adelie  Torgersen            39.1  ...              181.0       3750.0    Male
1     Adelie  Torgersen            39.5  ...              186.0       3800.0  Female
2     Adelie  Torgersen            40.3  ...              195.0       3250.0  Female
3     Adelie  Torgersen             NaN  ...                NaN          NaN     NaN
4     Adelie  Torgersen            36.7  ...              193.0       3450.0  Female
..       ...        ...             ...  ...                ...          ...     ...
339  Gentoo     Biscoe             NaN  ...                NaN          NaN     NaN
340  Gentoo     Biscoe            46.8  ...              215.0       4850.0  Female
341  Gentoo     Biscoe            50.4  ...              222.0       5750.0    Male
342  Gentoo     Biscoe            45.2  ...              212.0       5200.0  Female
343  Gentoo     Biscoe            49.9  ...              213.0       5400.0    Male

[344 rows x 7 columns]
```

Basic plots

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




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



A more complex plot

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

A more complex plot

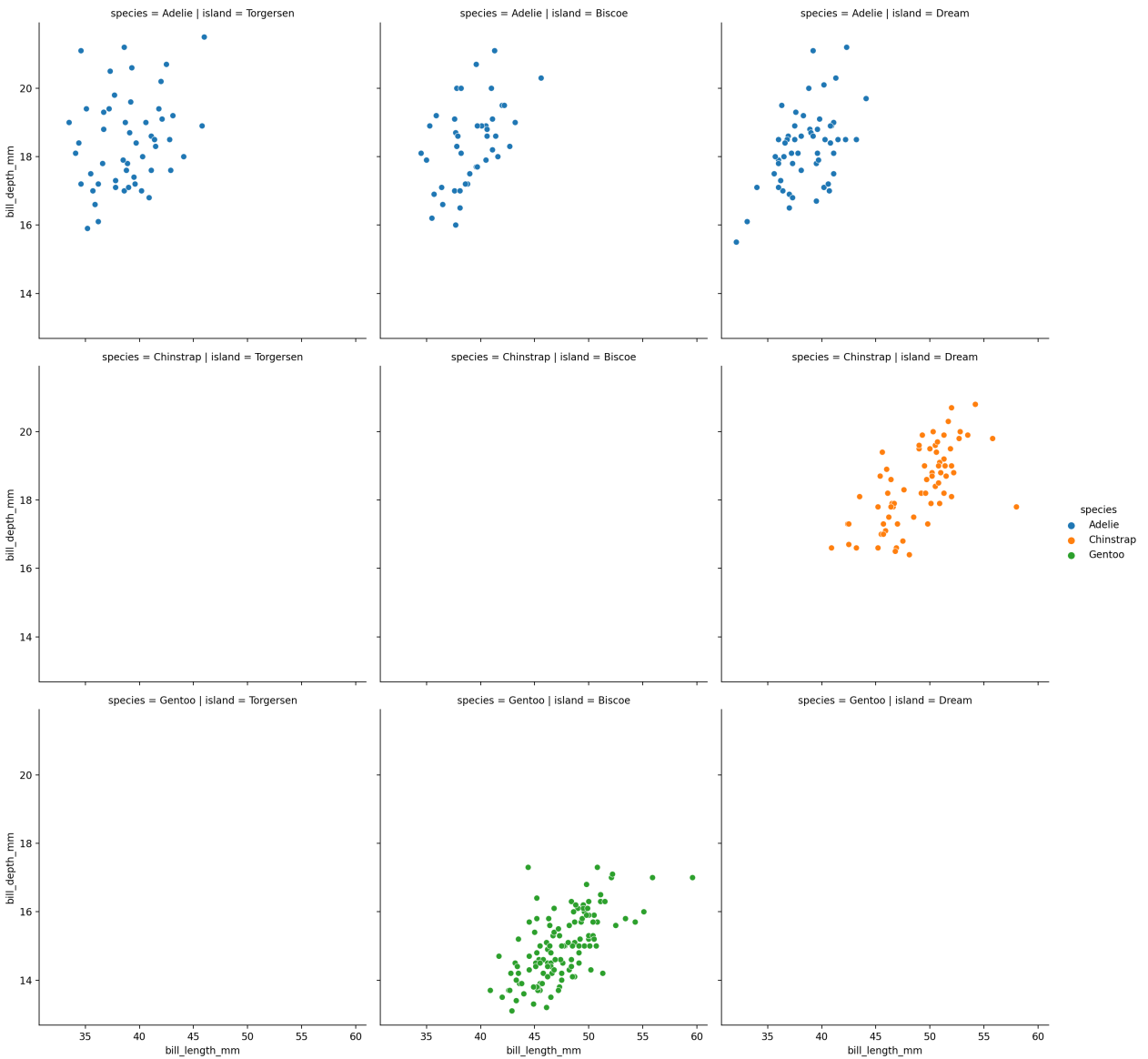
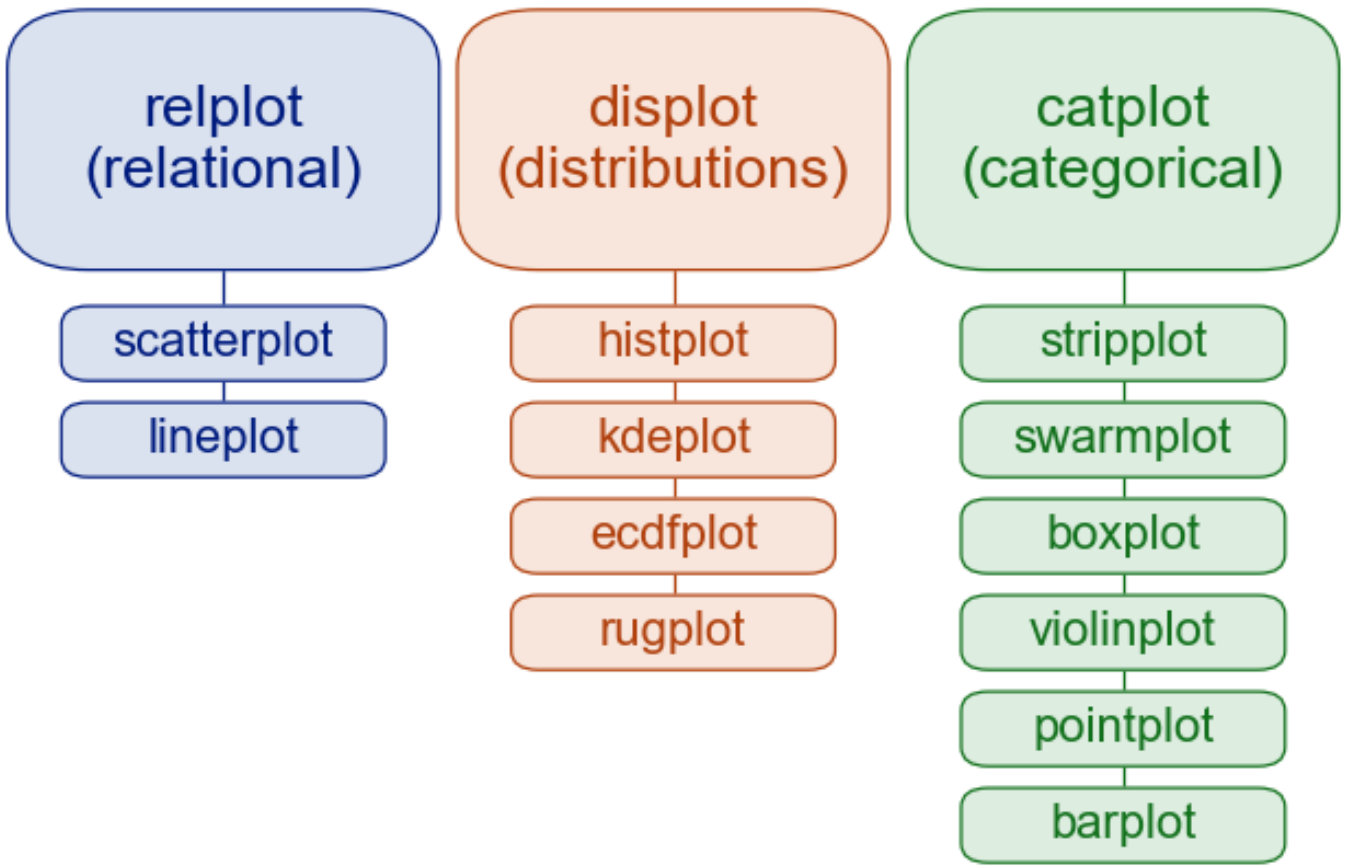


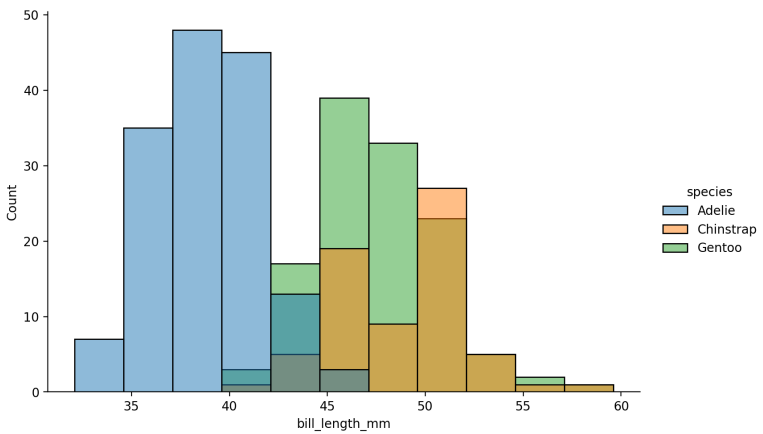
Figure-level vs. axes-level functions



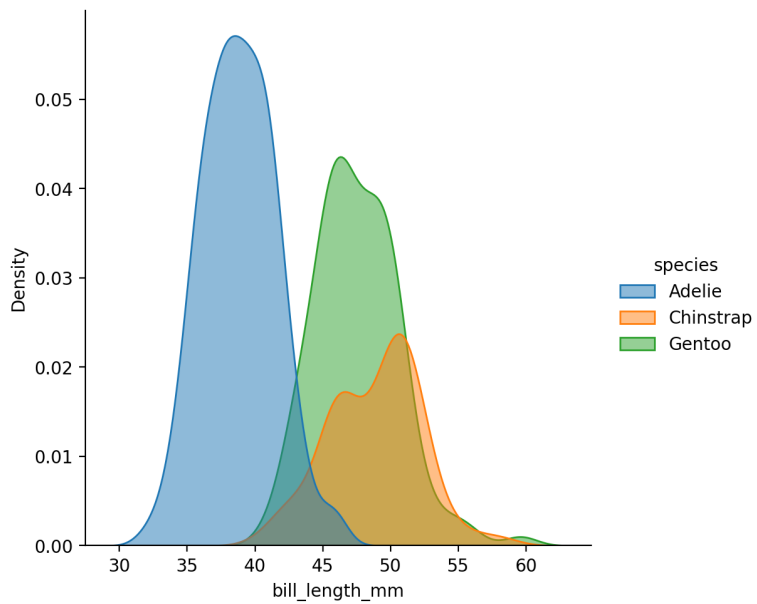
These are not the only axes-level functions - we see additional plotting functions in a bit

displots

```
sns.displot(
  data = penguins,
  x = "bill_length_mm",
  hue = "species",
  alpha = 0.5, aspect = 1.5
)
```

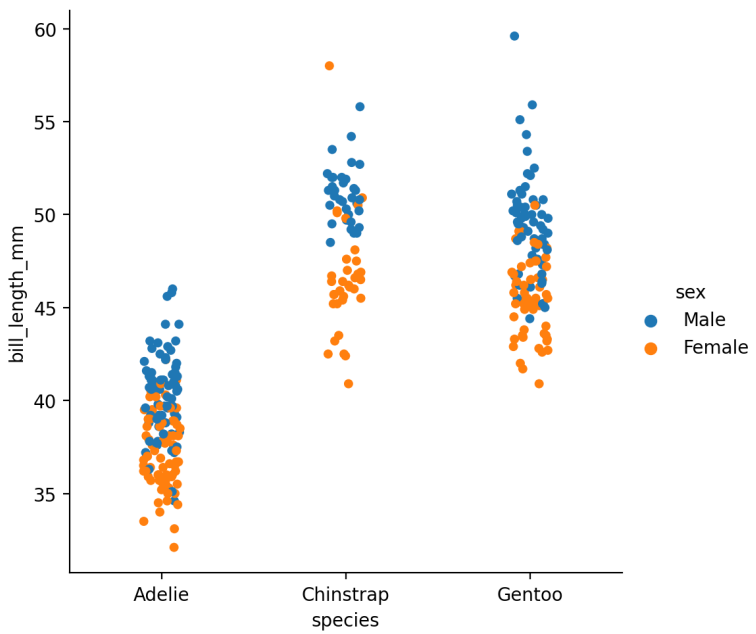



```
sns.displot(
  data = penguins,
  x = "bill_length_mm", hue = "species",
  kind = "kde", fill=True,
  alpha = 0.5, aspect = 1
)
```



catplots

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




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

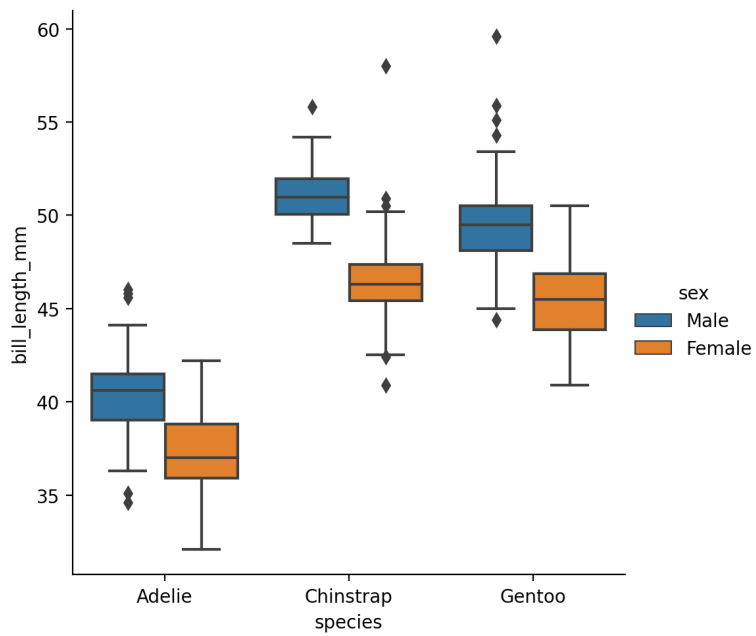
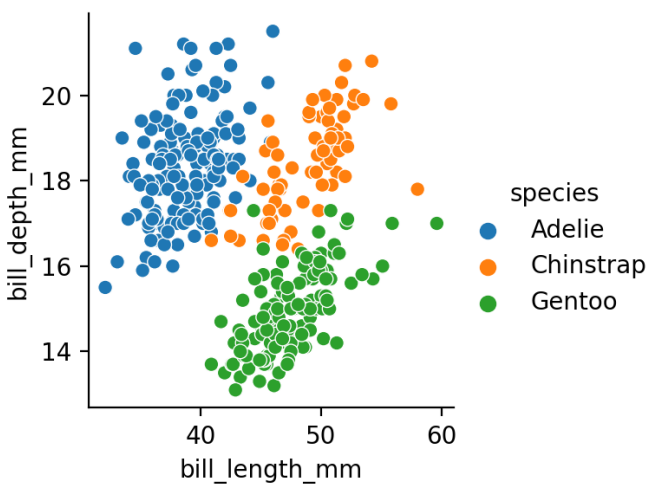


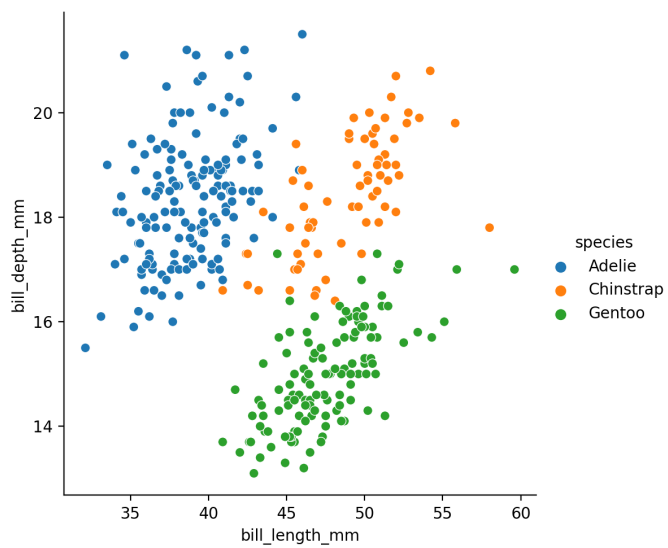
figure-level plot size

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

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



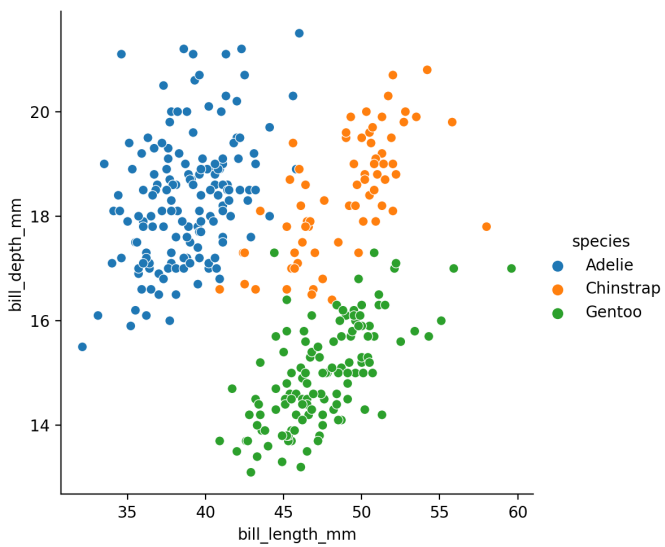

```
sns.relplot(
  data = penguins,
  x = "bill_length_mm", y = "bill_depth_mm",
  hue = "species",
  aspect = 1, height = 5
)
```



Note this is the size of a facet (Axes) not the figure

figure-level plots

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




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

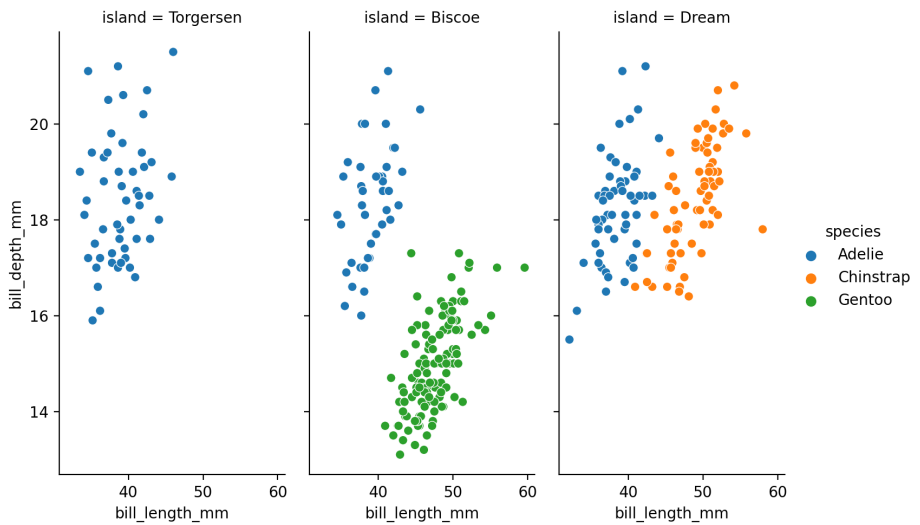


figure-level plot objects

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 0x2d89058a0>
```

```
1 print(h)
```

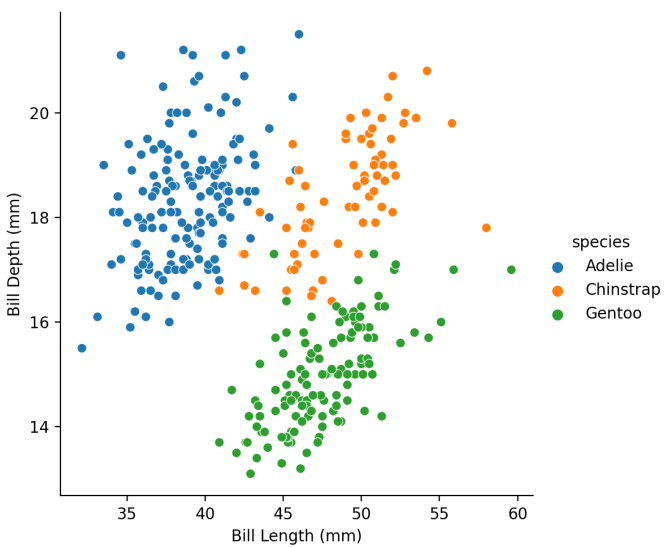
```
<seaborn.axisgrid.FacetGrid object at 0x2d88b7f70>
```

FacetGird methods

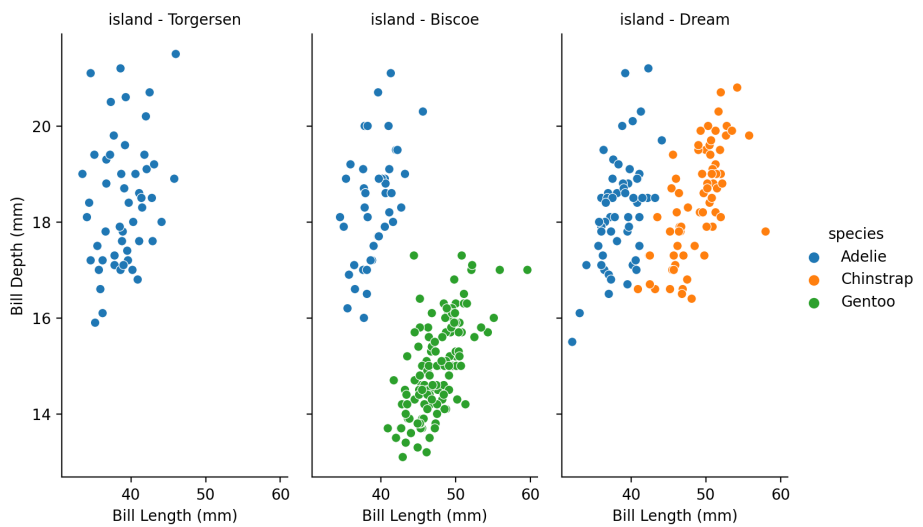
Method	Description
`add_legend()`	Draw a legend, maybe placing it outside axes and resizing the figure
`despine()`	Remove axis spines from the facets.
`facet_axis()`	Make the axis identified by these indices active and return it.
`facet_data()`	Generator for name indices and data subsets for each facet.
`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 exclude the legend.

Adjusting labels

```
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)"
)
```




```
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}"
)
```

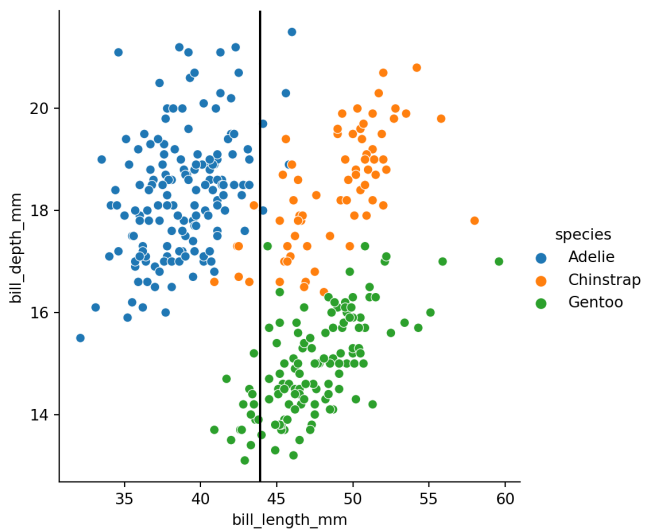


FacetGrid attributes

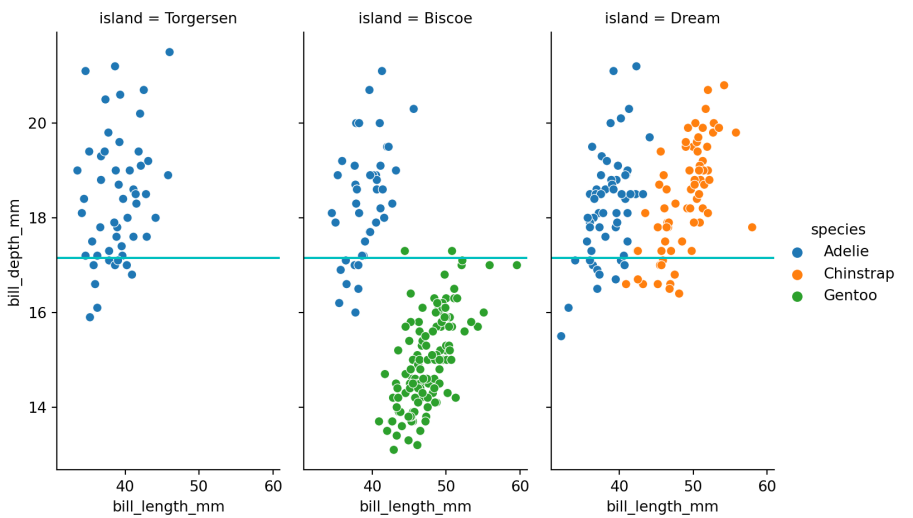
Attribute	Description
`ax`	The `matplotlib.axes.Axes` when no faceting variables are assigned.
`axes`	An array of the `matplotlib.axes.Axes` objects in the grid.
`axes_dict`	A mapping of facet names to corresponding `matplotlib.axes.Axes`.
`figure`	Access the `matplotlib.figure.Figure` object underlying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

Using axes to modify plots

```
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 = "k"
)
```




```
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()

[ ax.axhline(y=mean_bill_dep, c = "c")
  for row in h.axes for ax in row ]
```



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:

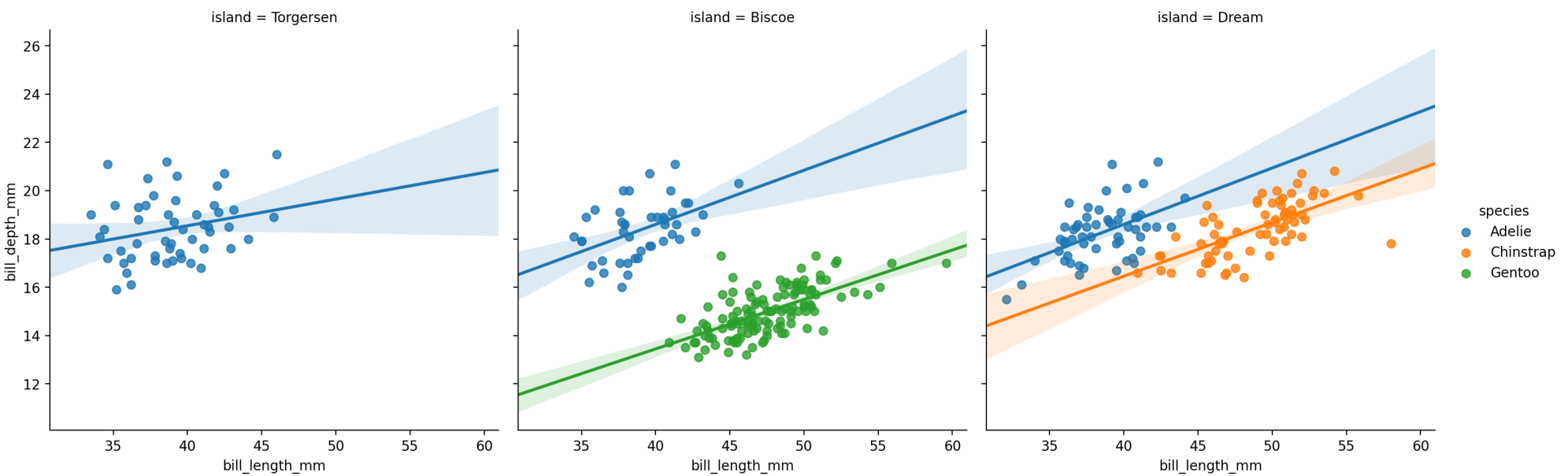
- Many parameters not in function signature
- Cannot be part of a larger matplotlib figure
- Different API from matplotlib
- Different figure size parameterization

Details based on seaborn docs

lmplots

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

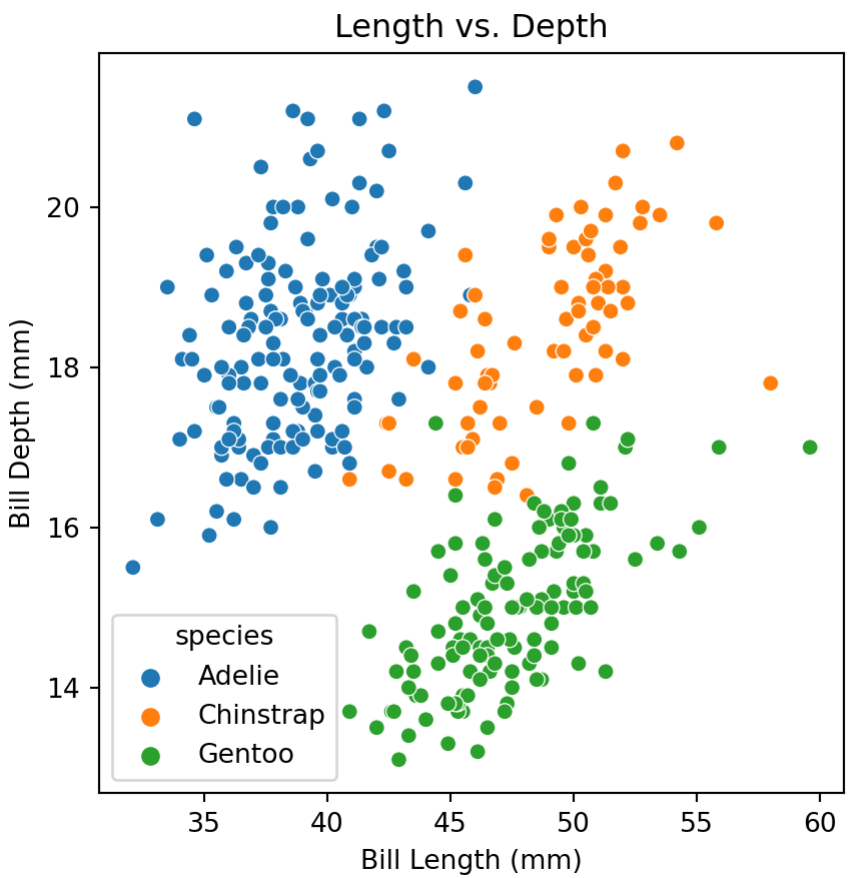
```
sns.lmplot(
  data = penguins,
  x = "bill_length_mm", y = "bill_depth_mm",
  hue = "species", col = "island",
  aspect = 1, truncate = False
)
```



axes-level functions

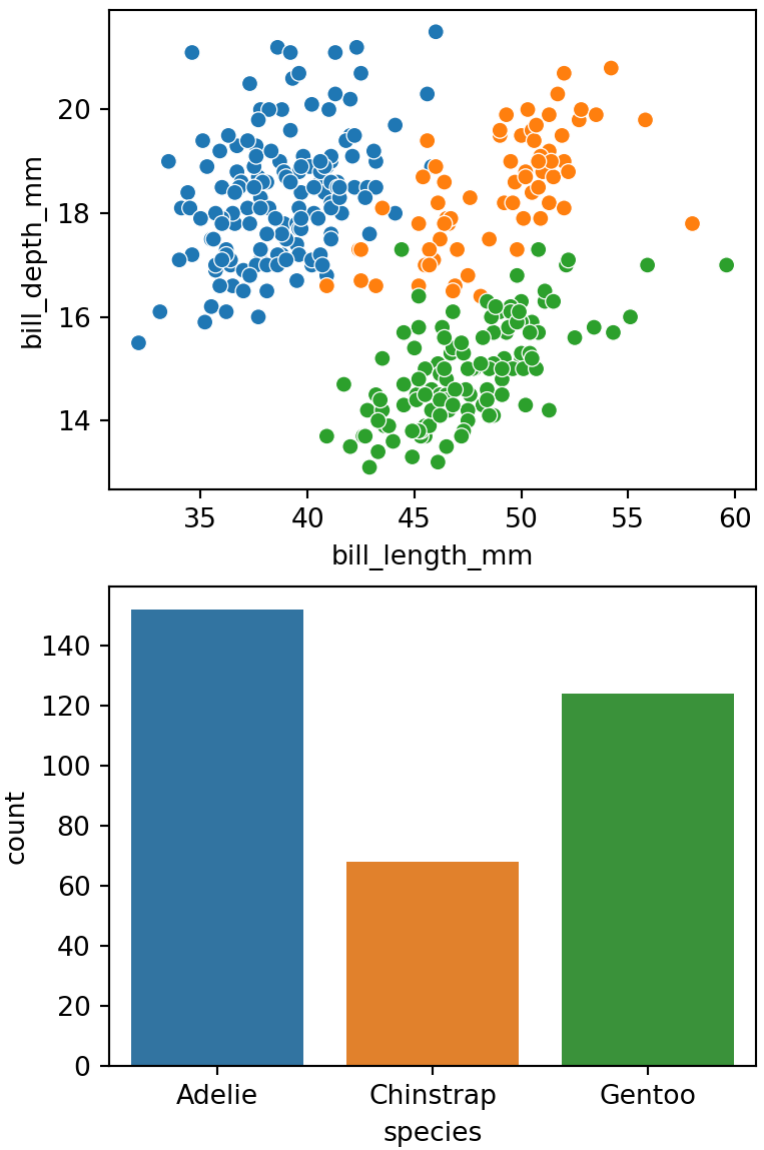
These functions return a `matplotlib.pyplot.Axes` object instead of a `FacetGrid` giving more direct control over the plot using basic matplotlib tools.

```
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()
```



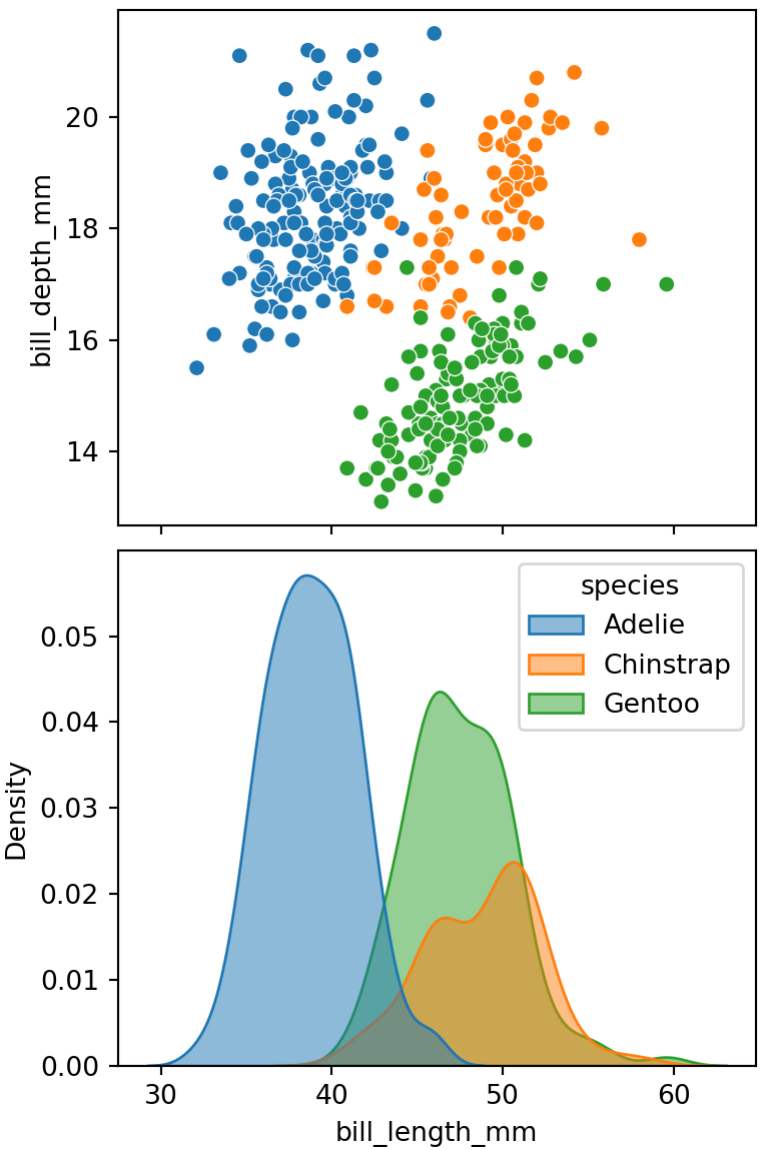
subplots - pyplot style

```
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"
)

plt.show()
```



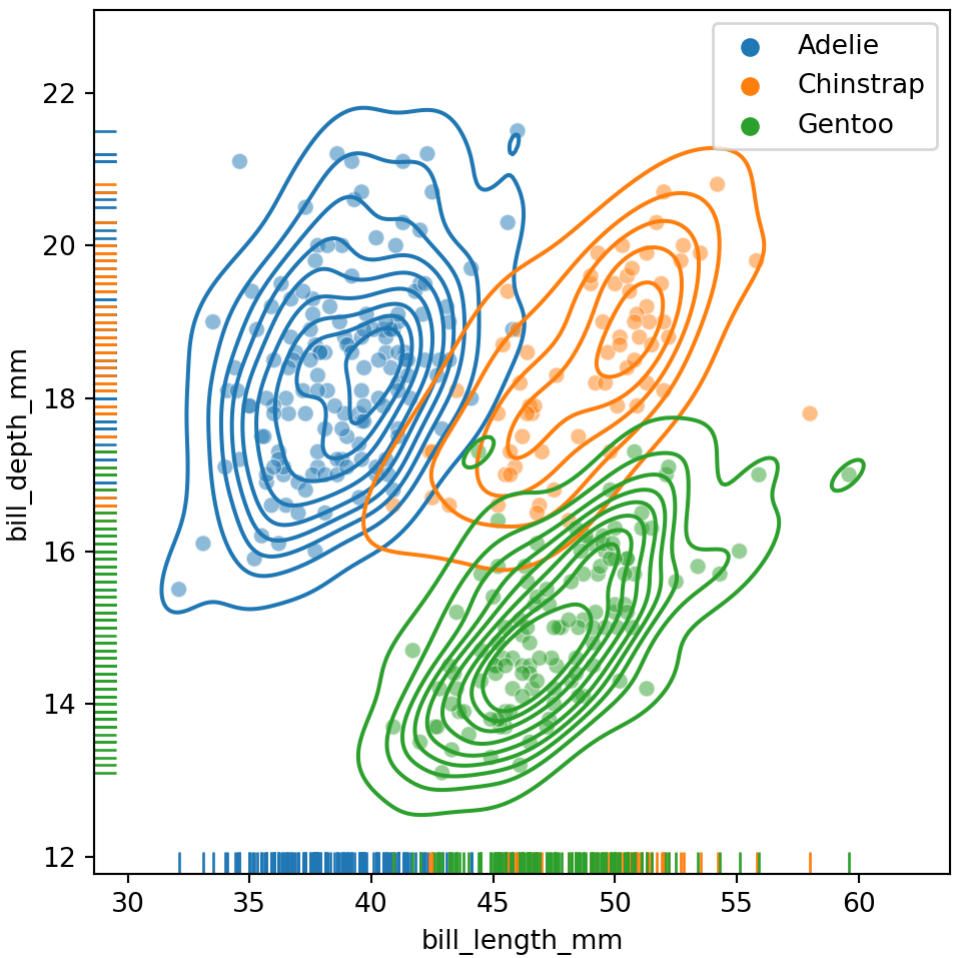
subplots - OO style

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

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

plt.show()
```



layering plots

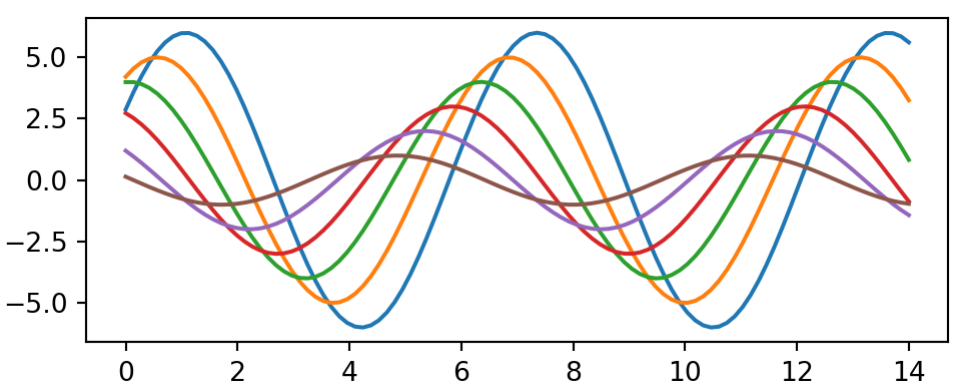
```
plt.figure(figsize=(5,5),
           layout = "constrained")

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()
```



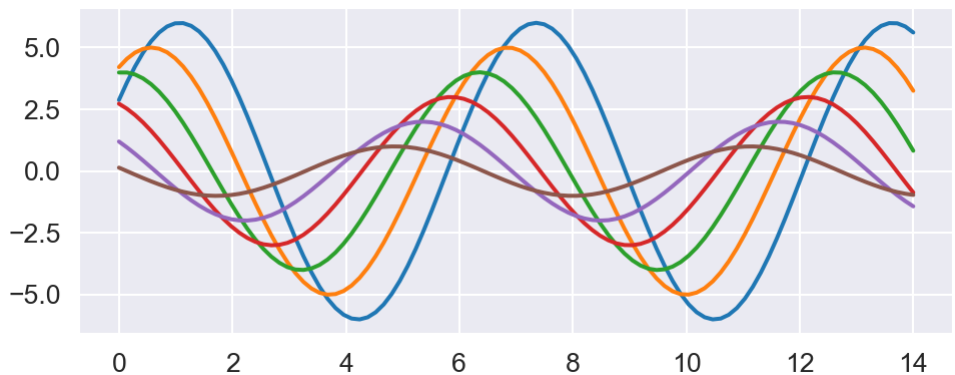
Themes

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()`.

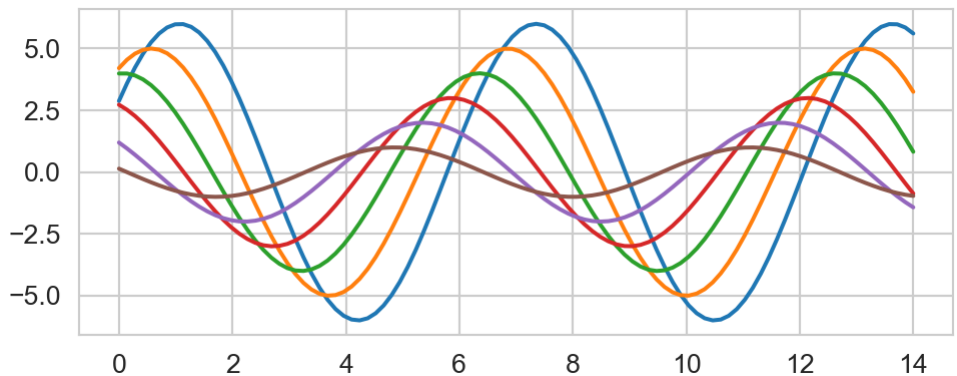
```
1  def sinplot():
2      plt.figure(figsize=(5,2), layout = "constrai
3      x = np.linspace(0, 14, 100)
4      for i in range(1, 7):
5          plt.plot(x, np.sin(x + i * .5) * (7 - i)
6      plt.show()
7
8  sinplot()
```



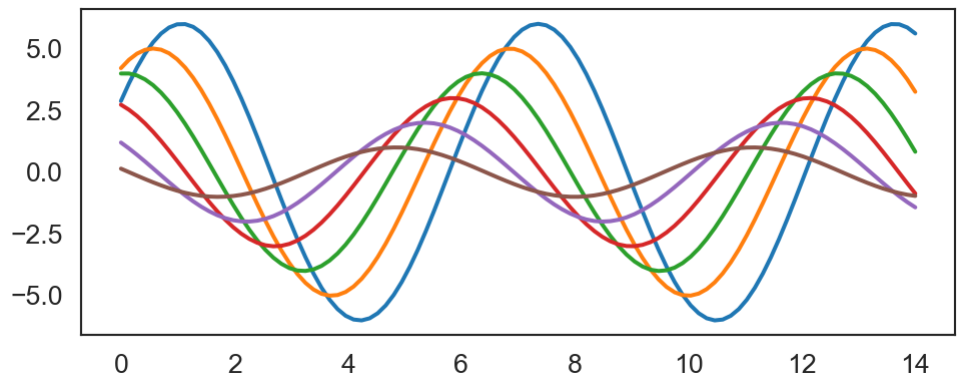

```
1  with sns.axes_style("darkgrid"):
2    sinplot()
```



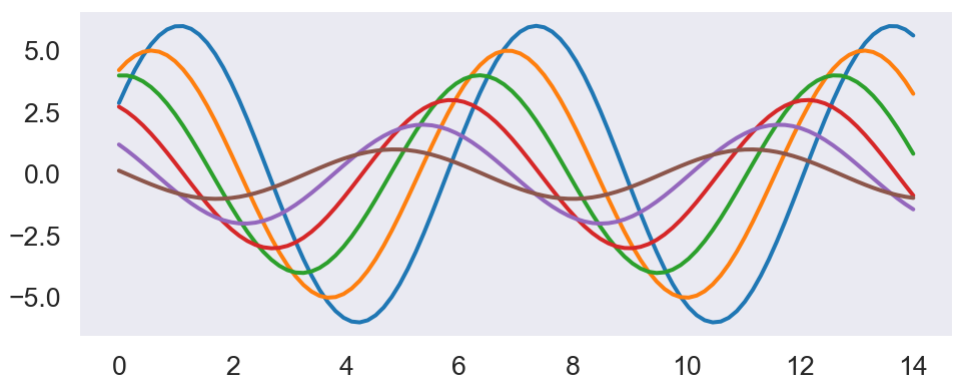
```
with sns.axes_style("whitegrid"):
  sinplot()
```



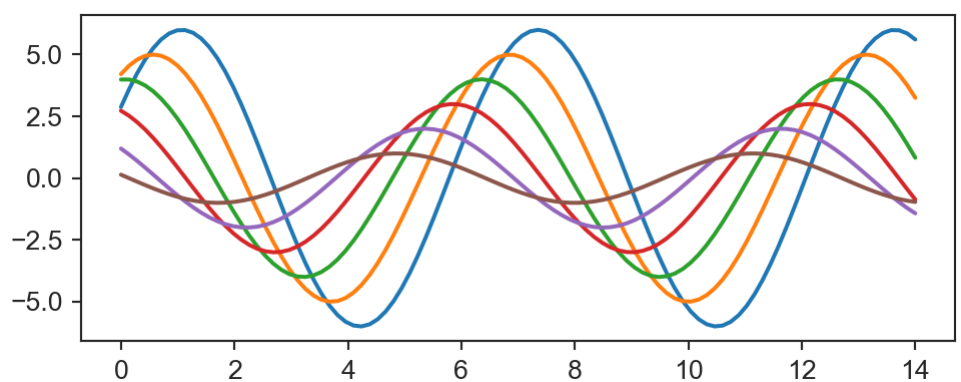

```
with sns.axes_style("white"):
  sinplot()
```




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




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



Context

Color palettes

All of the examples below are the result of calls to `sns.color_palette()` with `as_cmap=True` for the continuous case,

```
1 show_palette()
```

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

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

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

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

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

See more examples in the color palettes tutorial

Continuous palettes

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

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

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

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

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

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

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

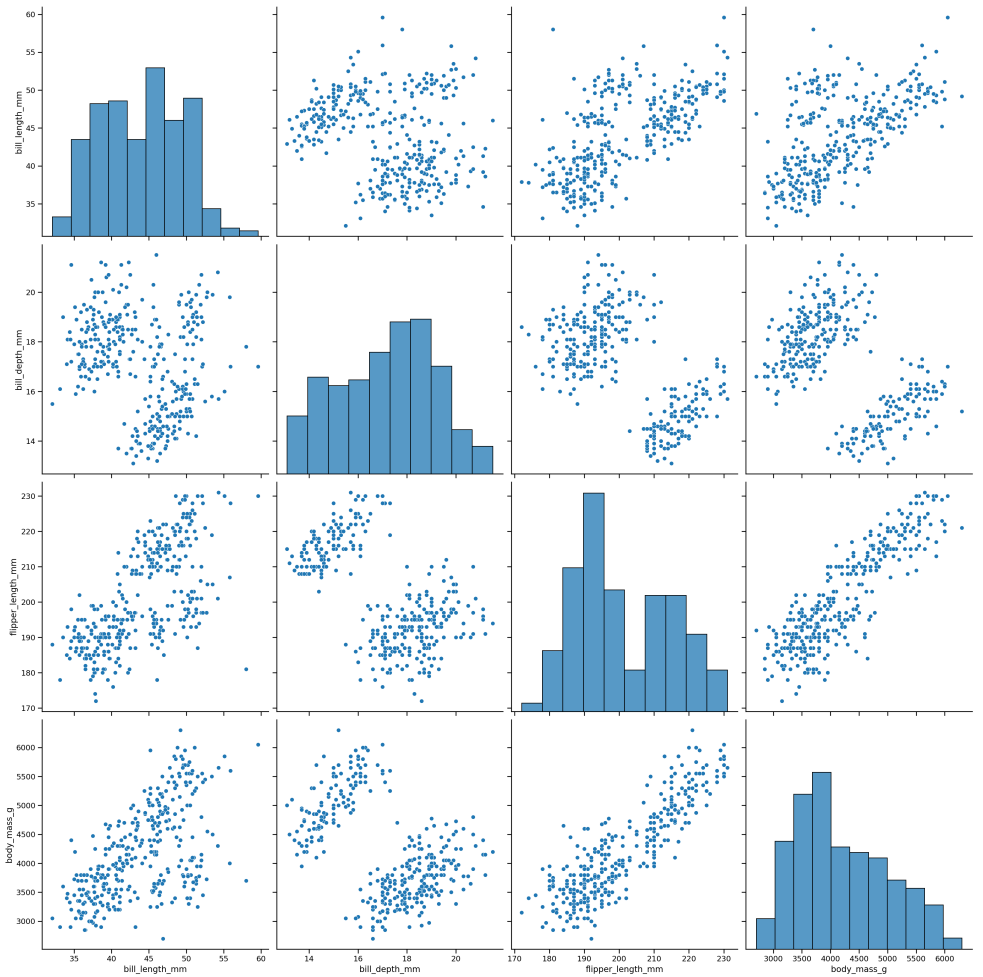
```
show_cont_palette("rocket")
```

Applying palettes

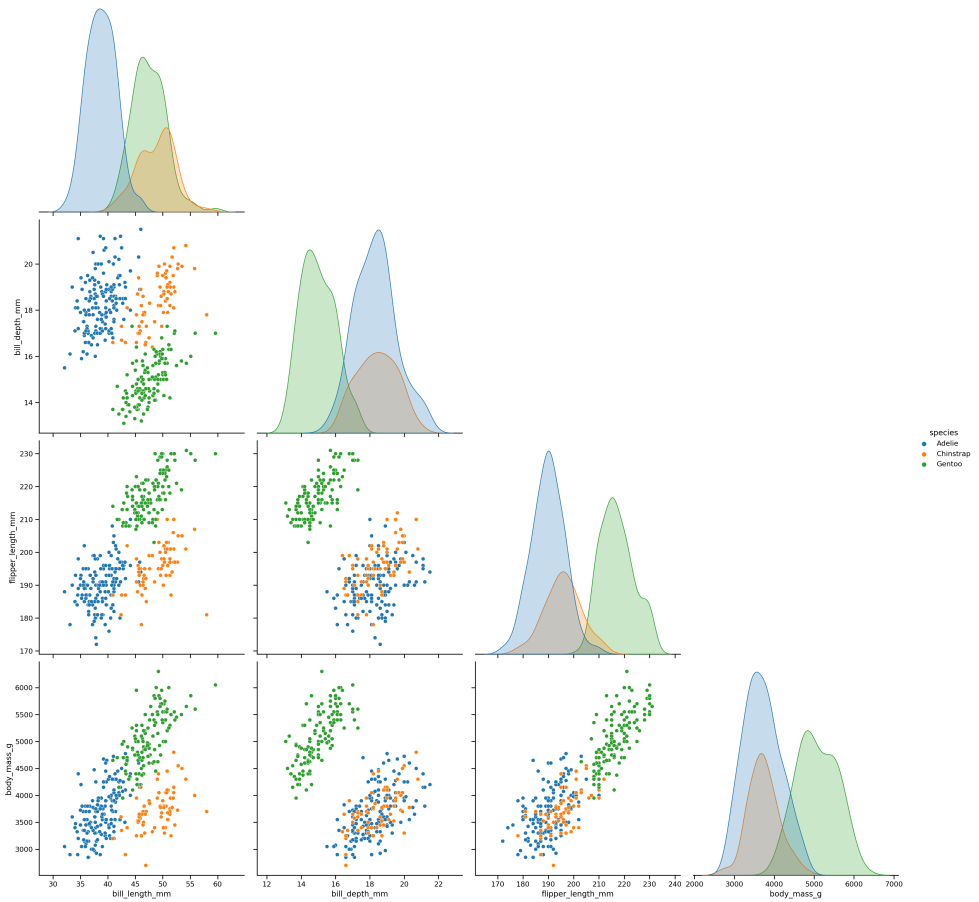
Palettes are applied via the `set_palette()` function,

Pair plots

```python
sns.pairplot(
  data = penguins,
  height=5
)
```



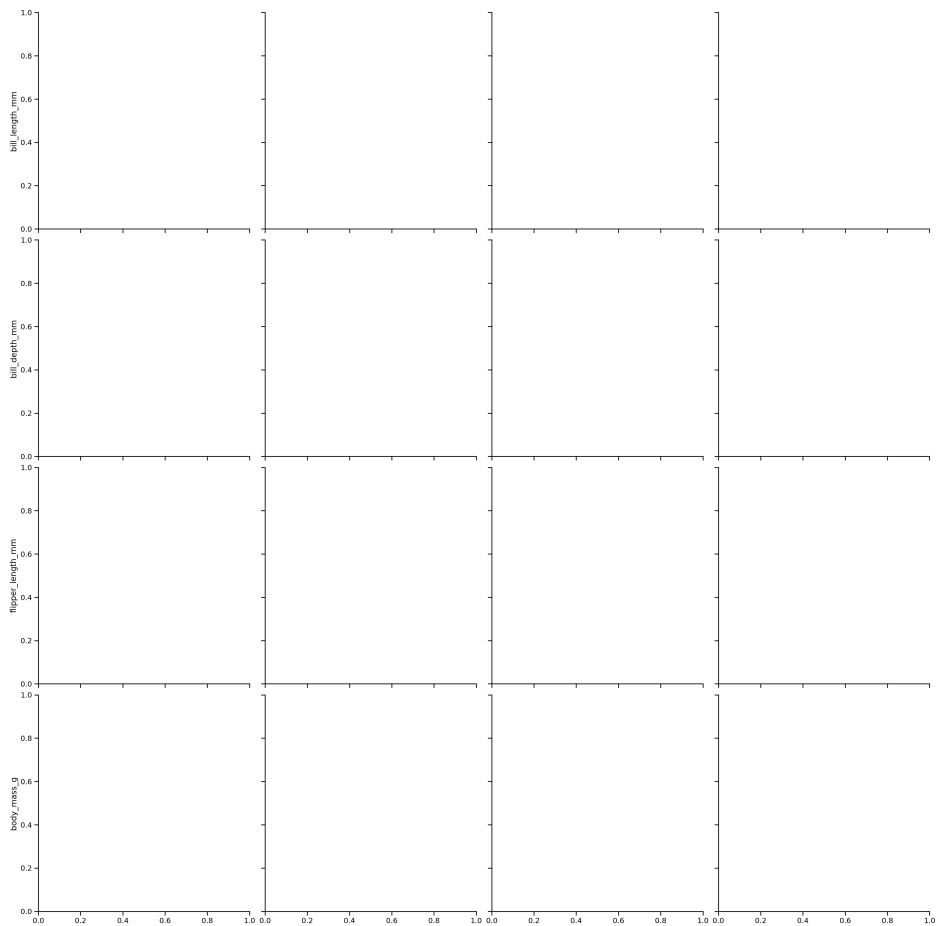

```python
sns.pairplot(
  data = penguins,
  hue = "species",
  height = 5, corner = True
)
```



PairGrid

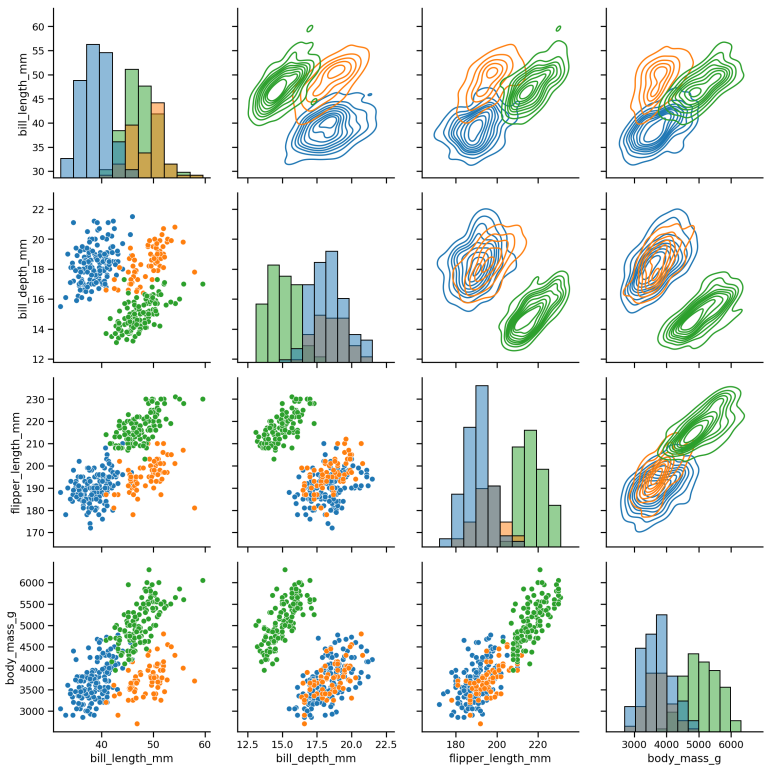
`pairplot()` is a special case of the more general `PairGrid` - once constructed there are methods that allow for mapping plot functions of the different axes,

```
1  sns.PairGrid(penguins, hue = "species", height=5)
```



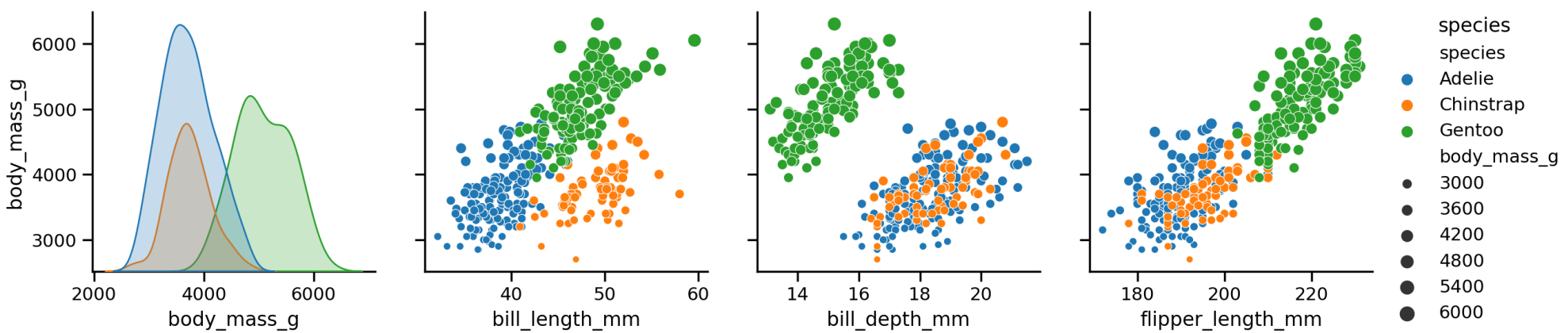
Mapping

```
g = sns.PairGrid(
  penguins, hue = "species",
  height=3
)

g = g.map_diag(
  sns.histplot, alpha=0.5
)

g = g.map_lower(
  sns.scatterplot
)

g = g.map_upper(
  sns.kdeplot
)

g
```



Pair subsets

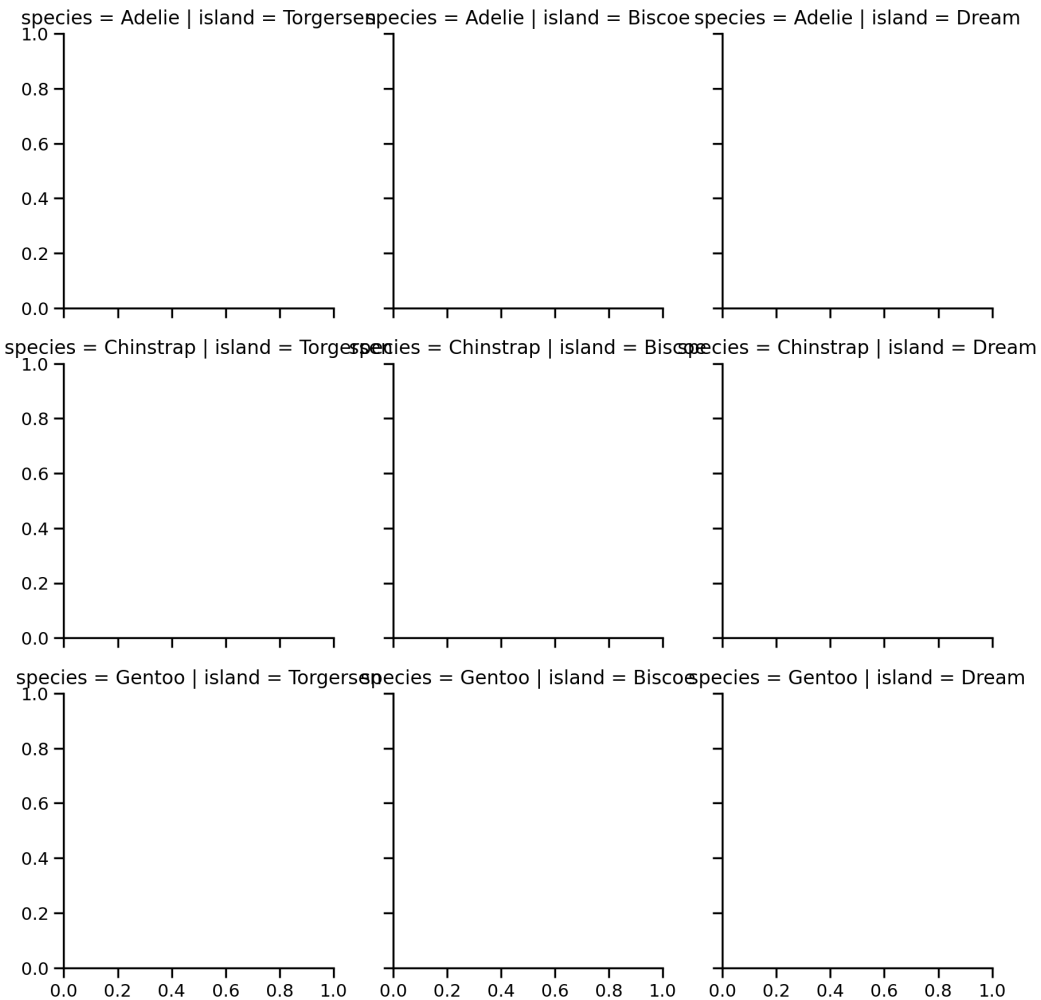
```
x_vars = ["body_mass_g", "bill_length_mm", "bill_depth_mm", "flipper_length_mm"]
y_vars = ["body_mass_g"]

( sns.PairGrid(
    penguins, hue = "species", x_vars=x_vars, y_vars=y_vars, height=3
  )
  .map_diag(
    sns.kdeplot, fill=True
  )
  .map_offdiag(
    sns.scatterplot, size=penguins["body_mass_g"]
  )
  .add_legend()
)
```



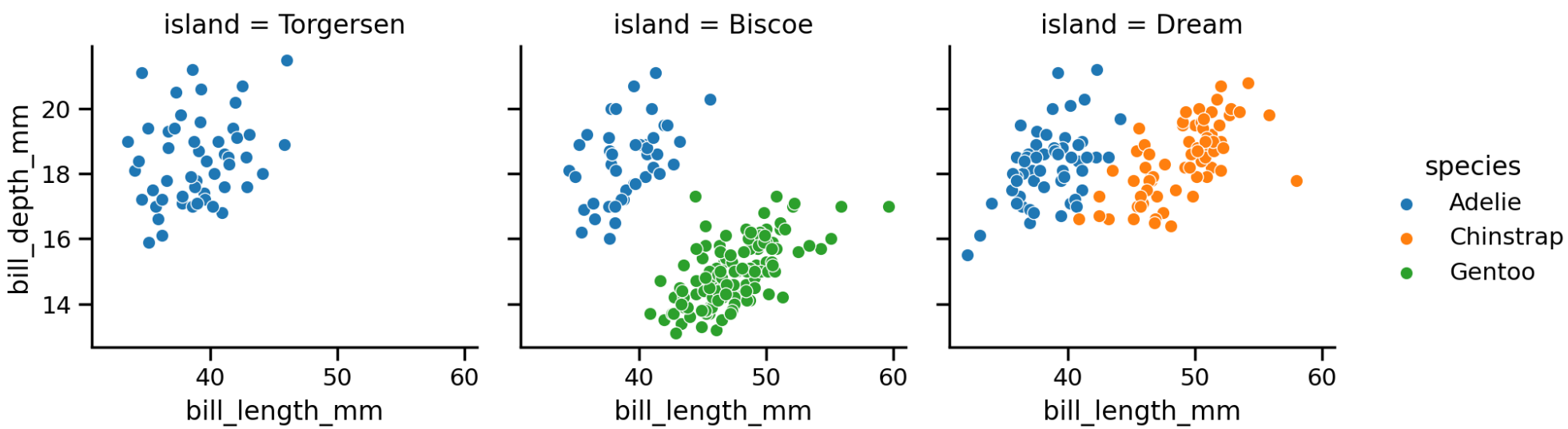
Custom FacetGrids

Just like `PairGrid`s it is possible to construct `FacetGrid`s from scratch,

```
1  sns.FacetGrid(penguins, col = "island", row = "species")
```

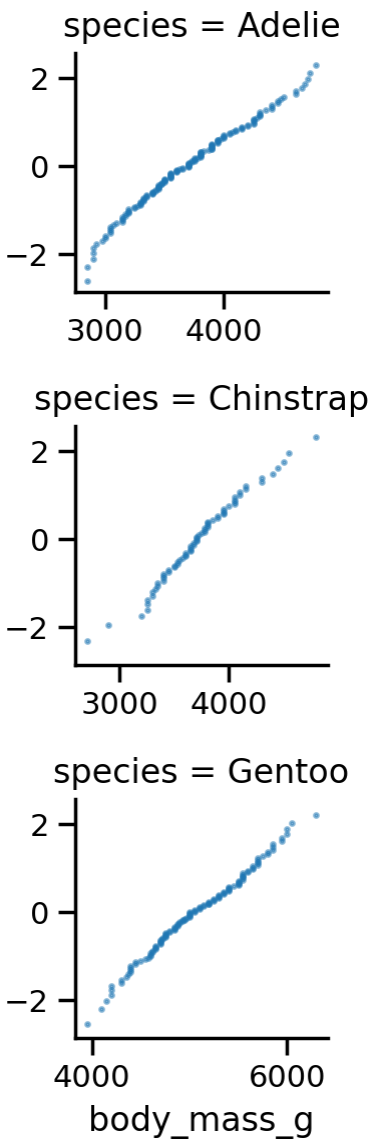


```python
( sns.FacetGrid(
    penguins, col = "island", hue = "species",
    height = 3, aspect = 1
  )
  .map(
    sns.scatterplot, "bill_length_mm", "bill_depth_mm"
  )
  .add_legend()
  .tight_layout()
)
```



Custom plots / functions

```python
from scipy import stats
def quantile_plot(x, **kwargs):
    quantiles, xr = stats.probplot(x, fit=False)
    plt.scatter(xr, quantiles, **kwargs)

( sns.FacetGrid(
    penguins,
    row = "species",
    height=2,
    sharex=False
  )
  .map(
    quantile_plot,
    "body_mass_g", s=2, alpha=0.5
  )
)
```

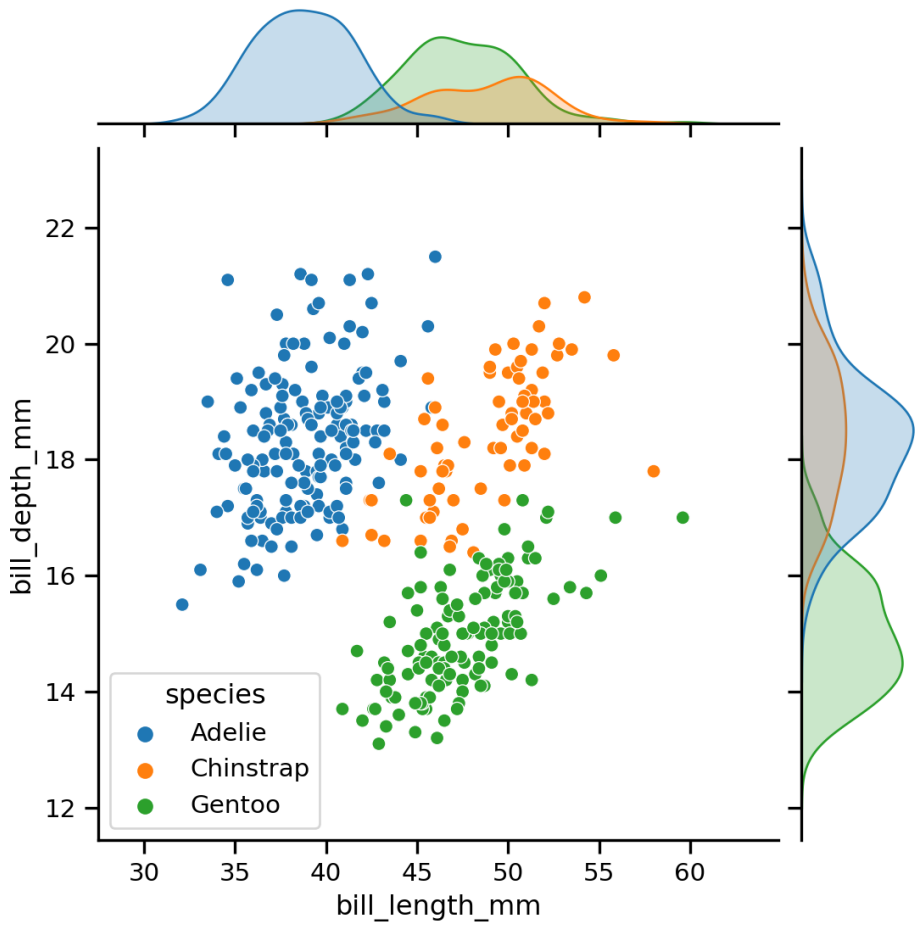


Example from axis grid tutorial

jointplot

One final figure-level plot, is a joint plot which includes marginal distributions along the x and y-axis.

```
g = sns.jointplot(
  data = penguins,
  x = "bill_length_mm",
  y = "bill_depth_mm",
  hue = "species"
)
plt.show()
```



Adjusting

The main plot (joint) and the margins (marginal) can be modified by keywords or via layering (use `plot_joint()` and `plot_marginals()` methods).

```
g = ( sns.jointplot(
    data = penguins,
    x = "bill_length_mm",
    y = "bill_depth_mm",
    hue = "species",
    marginal_kws=dict(fill=False)
  )
  .plot_joint(
    sns.kdeplot, alpha=0.5, levels=5
  )
)
plt.show()
```