# matplotlib / pyplot & seaborn

## Lecture 10

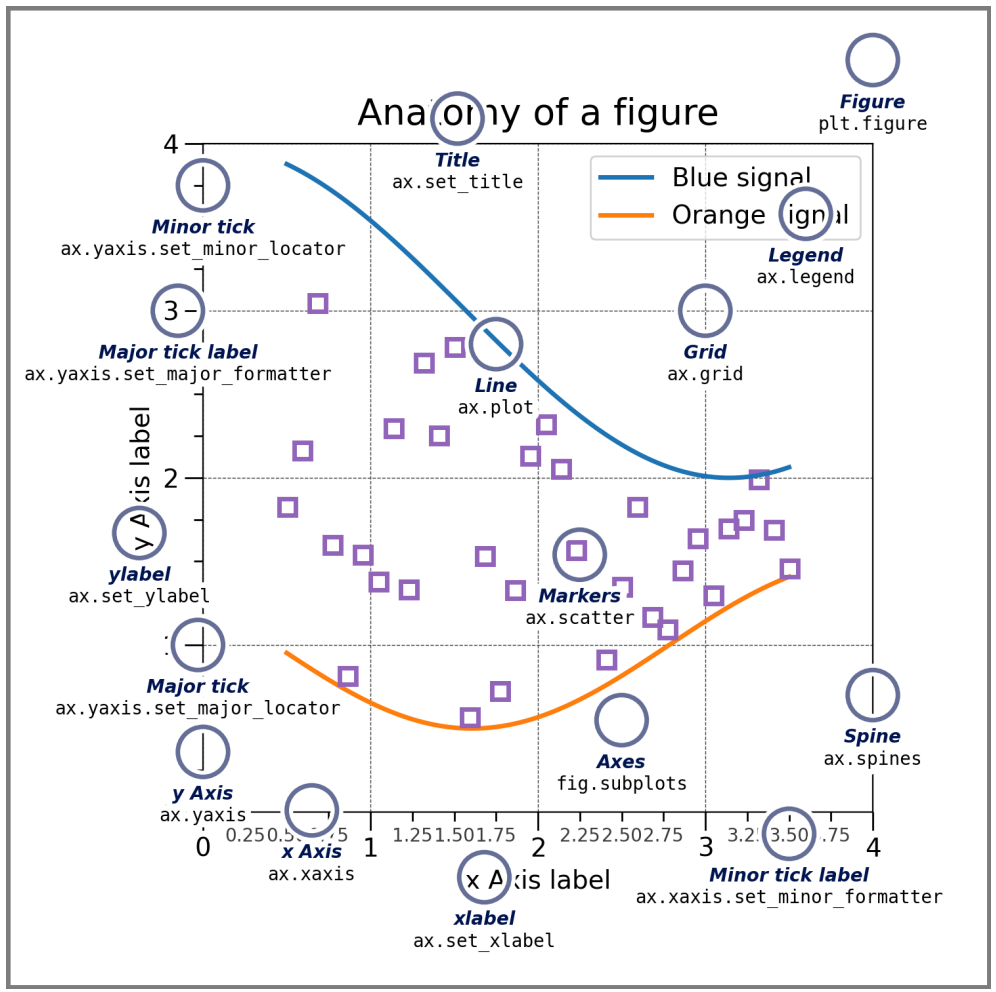
Dr. Colin Rundel

# matplotlib & pyplot

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

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

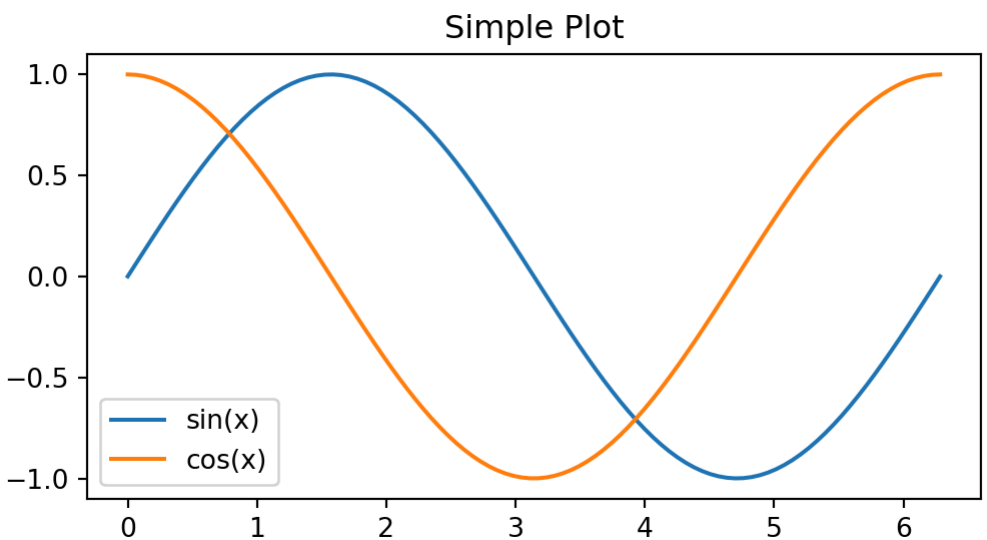
# Plot anatomy




- **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.

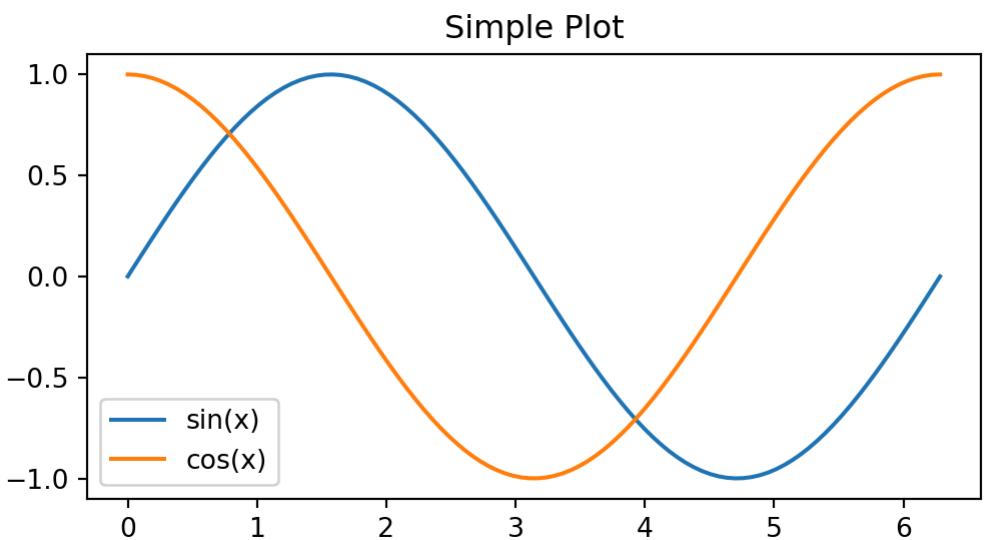
# Basic plot - OO style

```
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()
```



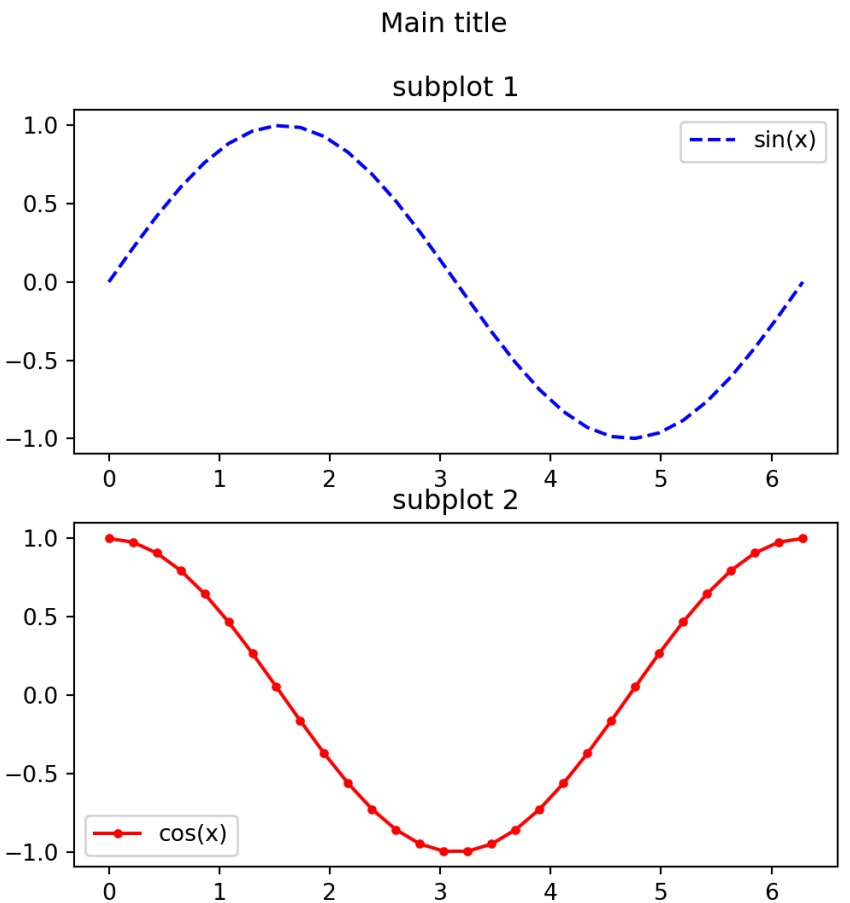
# Basic plot - pyplot style

```
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()
```



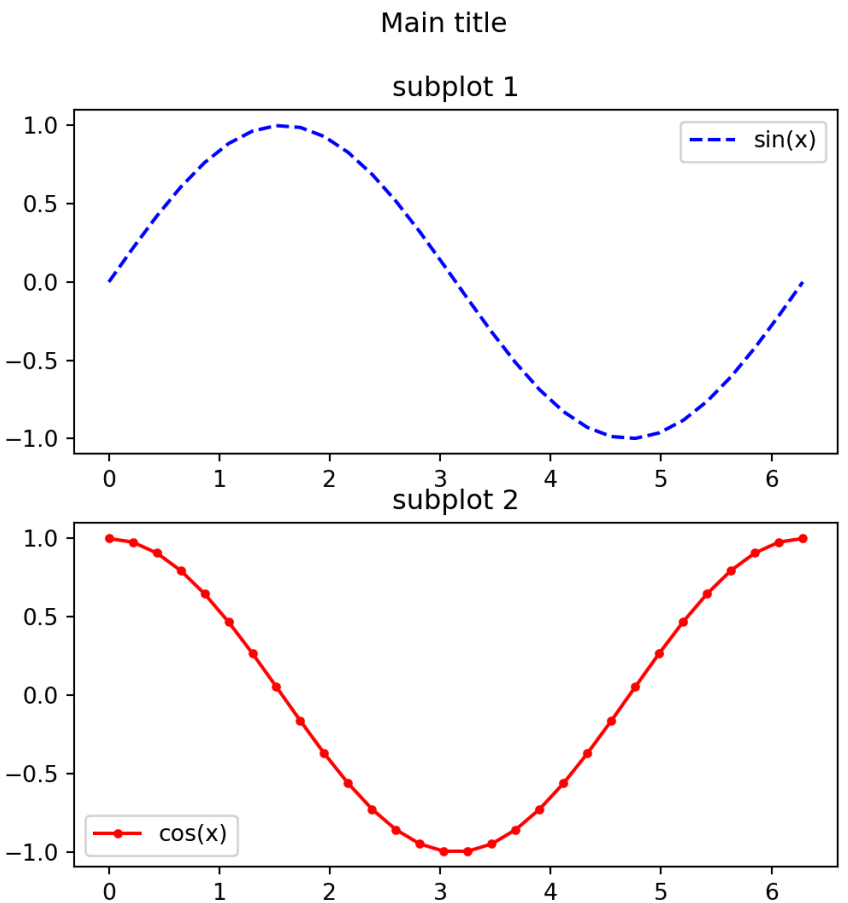
# Subplots (OO)

```
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()
```



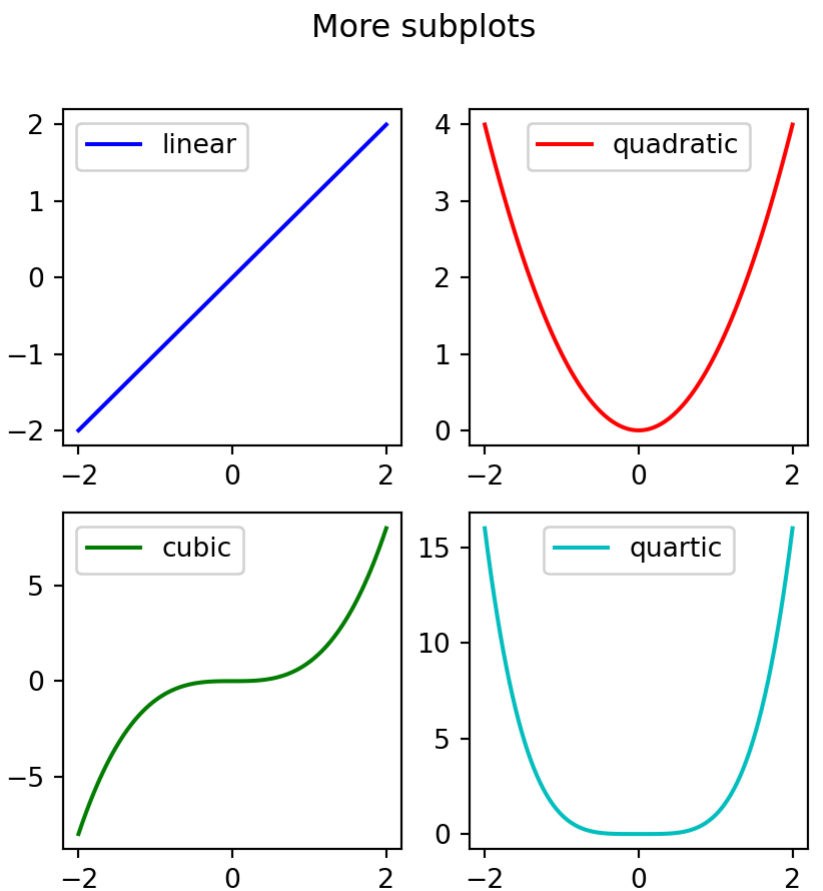
# Subplots (pyplot)

```
1  x = np.linspace(0, 2*np.pi, 30)
2  y1 = np.sin(x)
3  y2 = np.cos(x)
4
5  plt.figure(figsize=(6, 6))
6
7  plt.suptitle("Main title")
8
9  plt.subplot(211)
10 plt.plot(x, y1, "--b", label="sin(x)")
11 plt.title("subplot 1")
12 plt.legend()
13
14 plt.subplot(2,1,2)
15 plt.plot(x, y2, ".-r", label="cos(x)")
16 plt.title("subplot 2")
17 plt.legend()
18
19 plt.show()
```



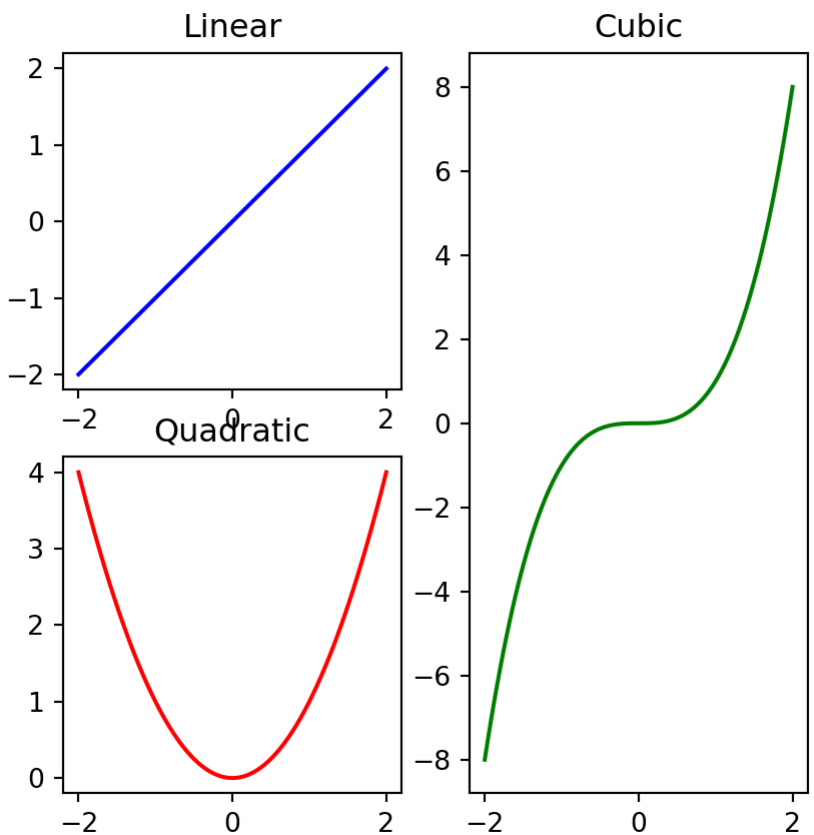
# More subplots

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

fig, axs = plt.subplots(
  2, 2,
  figsize=(5, 5)
)

fig.suptitle("More subplots")

axs[0,0].plot(x, x, "b", label="linear")
axs[0,1].plot(x, x**2, "r", label="quadrati
axs[1,0].plot(x, x**3, "g", label="cubic")
axs[1,1].plot(x, x**4, "c", label="quartic"

[ax.legend() for row in axs for ax in row]
```



# 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", label="lin
axd['lowleft'].plot(x, x**2, "r", label="qua
axd['right'  ].plot(x, x**3, "g", label="cub

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



# Format strings

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

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

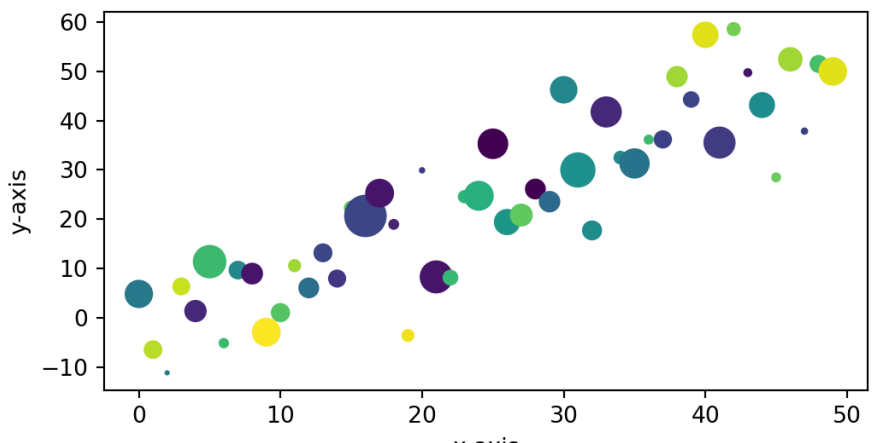
| character | line style |
|---|---|
| `-` | solid |
| `--` | dashed |
| `-.` | dash-dot |
| `:` | dotted |

| 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`,

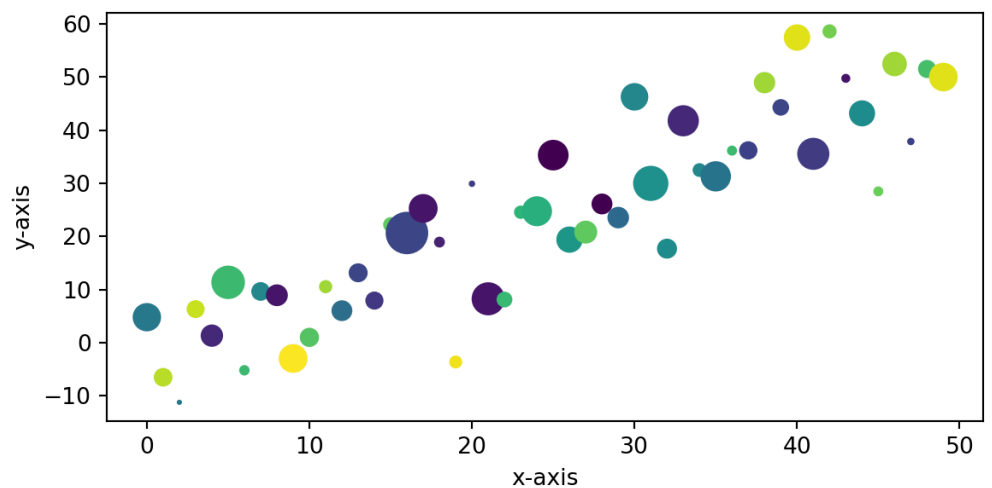
```
np.random.seed(19680801)
d = {'x': np.arange(50),
     'color': np.random.randint(0, 50, 50),
     'size': np.abs(np.random.randn(50)) * 
d['y'] = d['x'] + 10 * np.random.randn(50)


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)) * 
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()
```

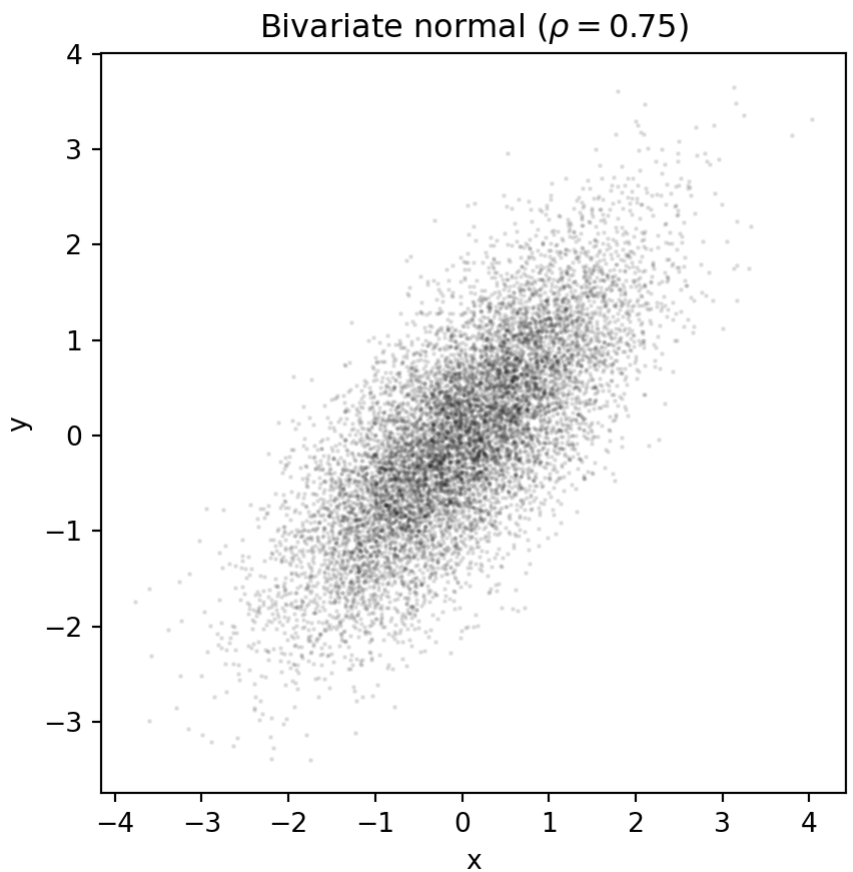


# pyplot w/ pandas

Data can also come from DataFrame objects or series,

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

# pyplot w/ pandas

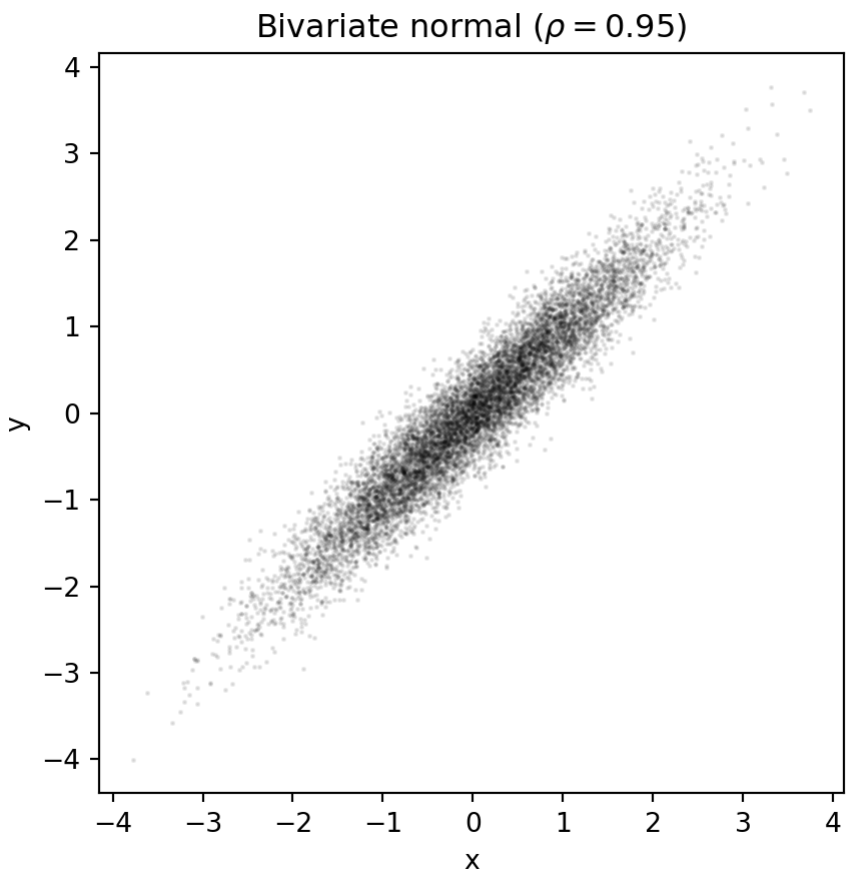


# pyplot w/ polars

Data can also come from DataFrame objects or series,

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

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("Bivariate normal ($\\rho=0.95$)")
```

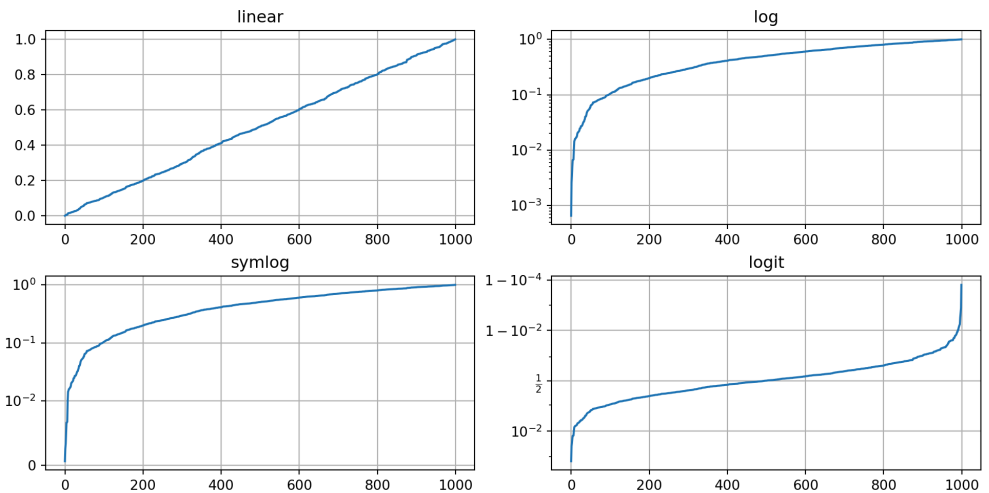
# pyplot w/ polars



# Scales

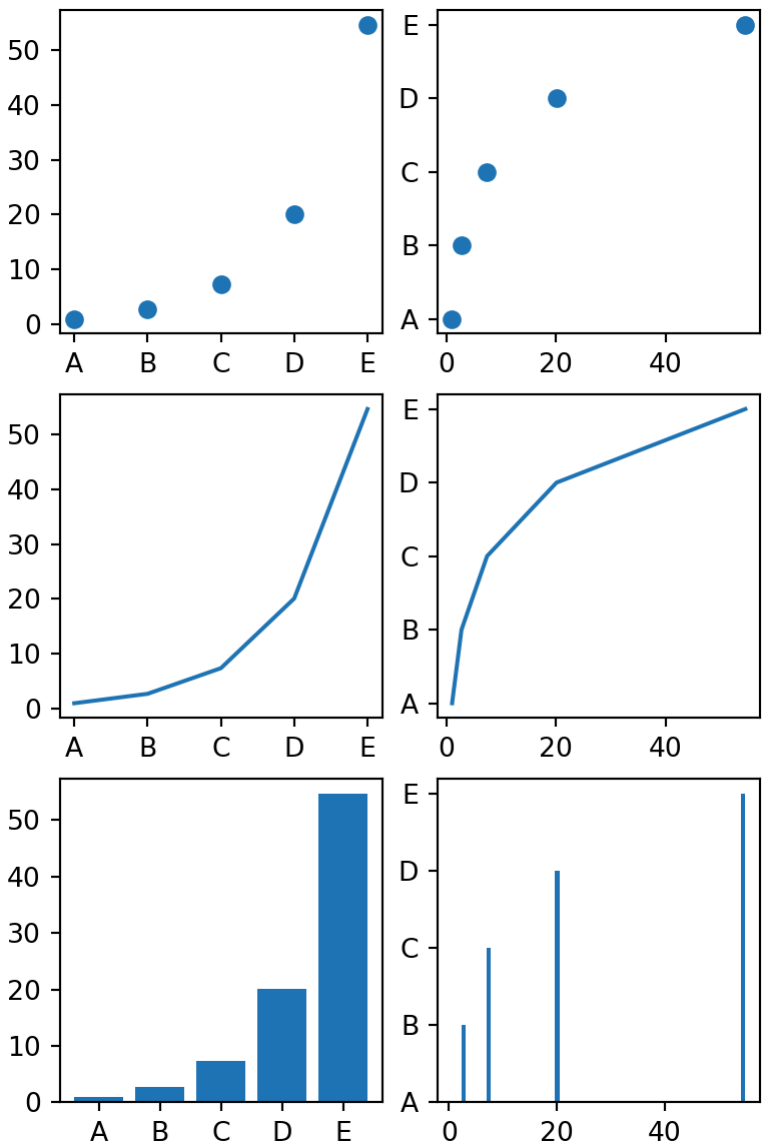
Axis scales can be changed via `plt.xscale()`, `plt.yscale()`, `ax.set_xscale()`, or `ax.set_yscale()`, supported values are “linear”, “log”, “symlog”, and “logit”.

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

plt.figure(layout="constrained")

scales = ['linear', 'log', 'symlog', 'logit
for i, scale in zip(range(4), scales):
  plt.subplot(221+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()
```



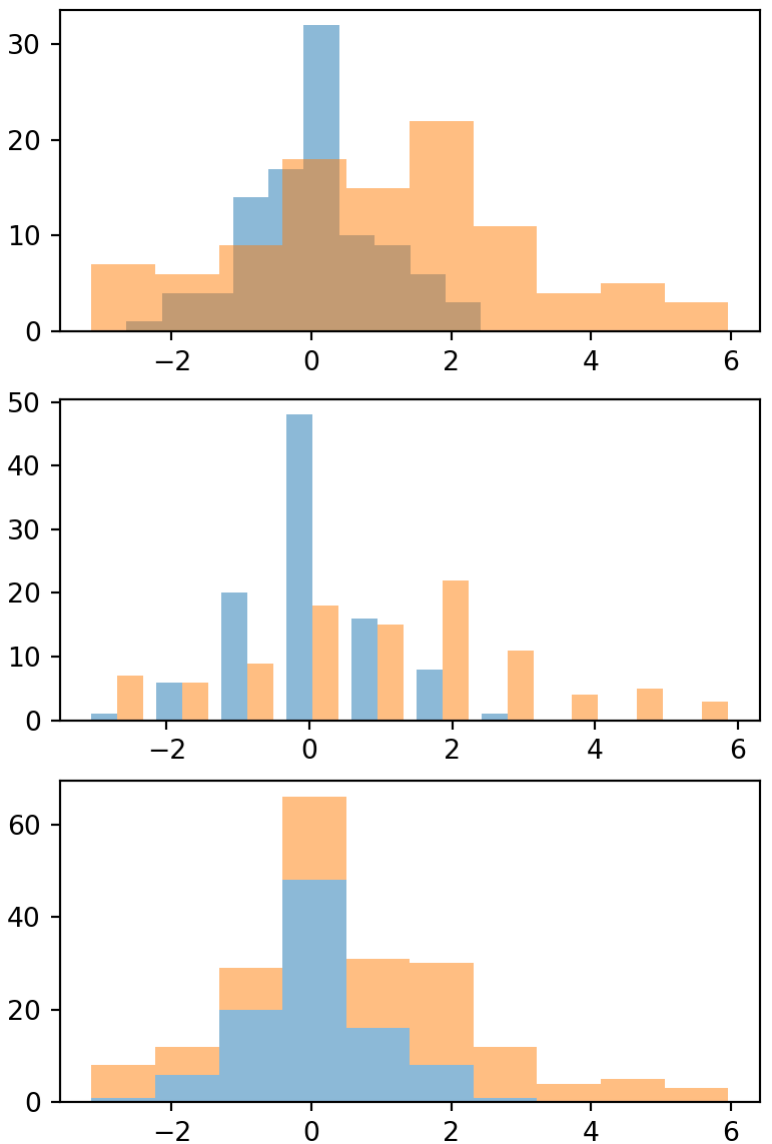
# Categorical data

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

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

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)
```

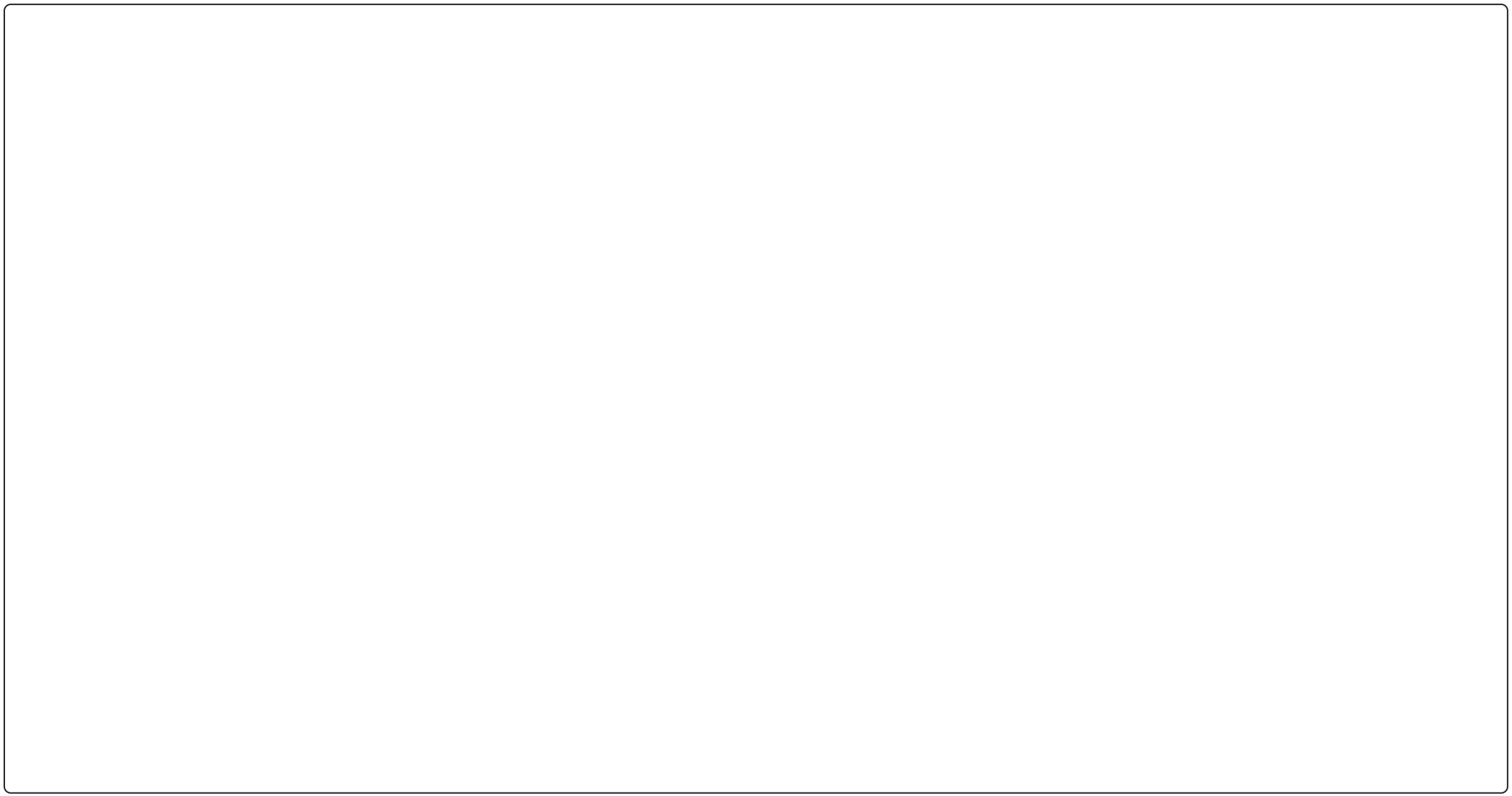


# Histograms

```
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()
```

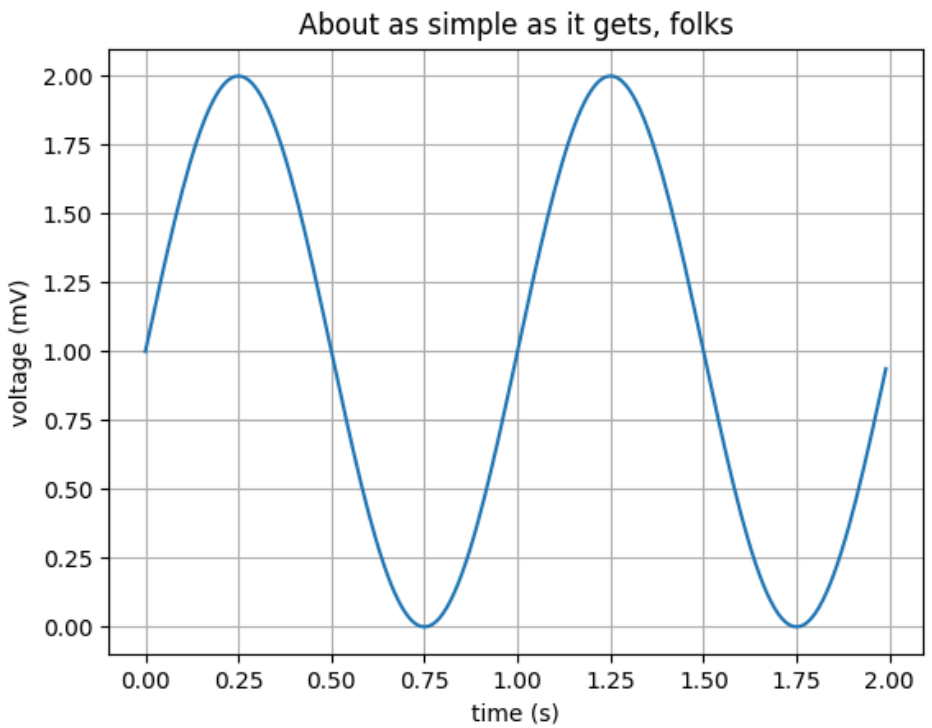


# Other Plot Types



# Exercise 1

To the best of your ability recreate the following plot,



# 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
```

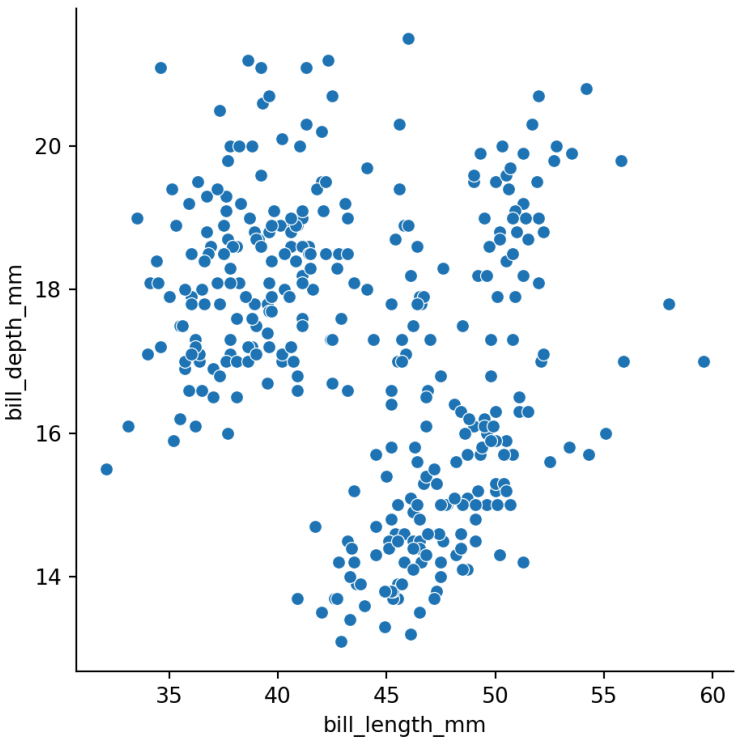
# Penguins data

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

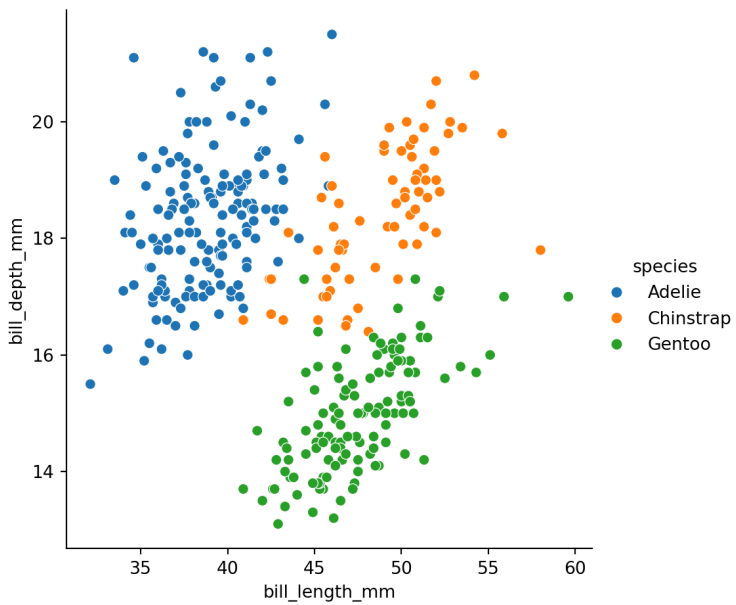
```
    species     island  bill_length_mm  ...  flipper_length_mm  body_mass_g     sex
0    Adelie  Torgersen            39.1  ...              181.0       3750.0    Male
1    Adelie  Torgersen            39.5  ...              186.0       3800.0  Female
2    Adelie  Torgersen            40.3  ...              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

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



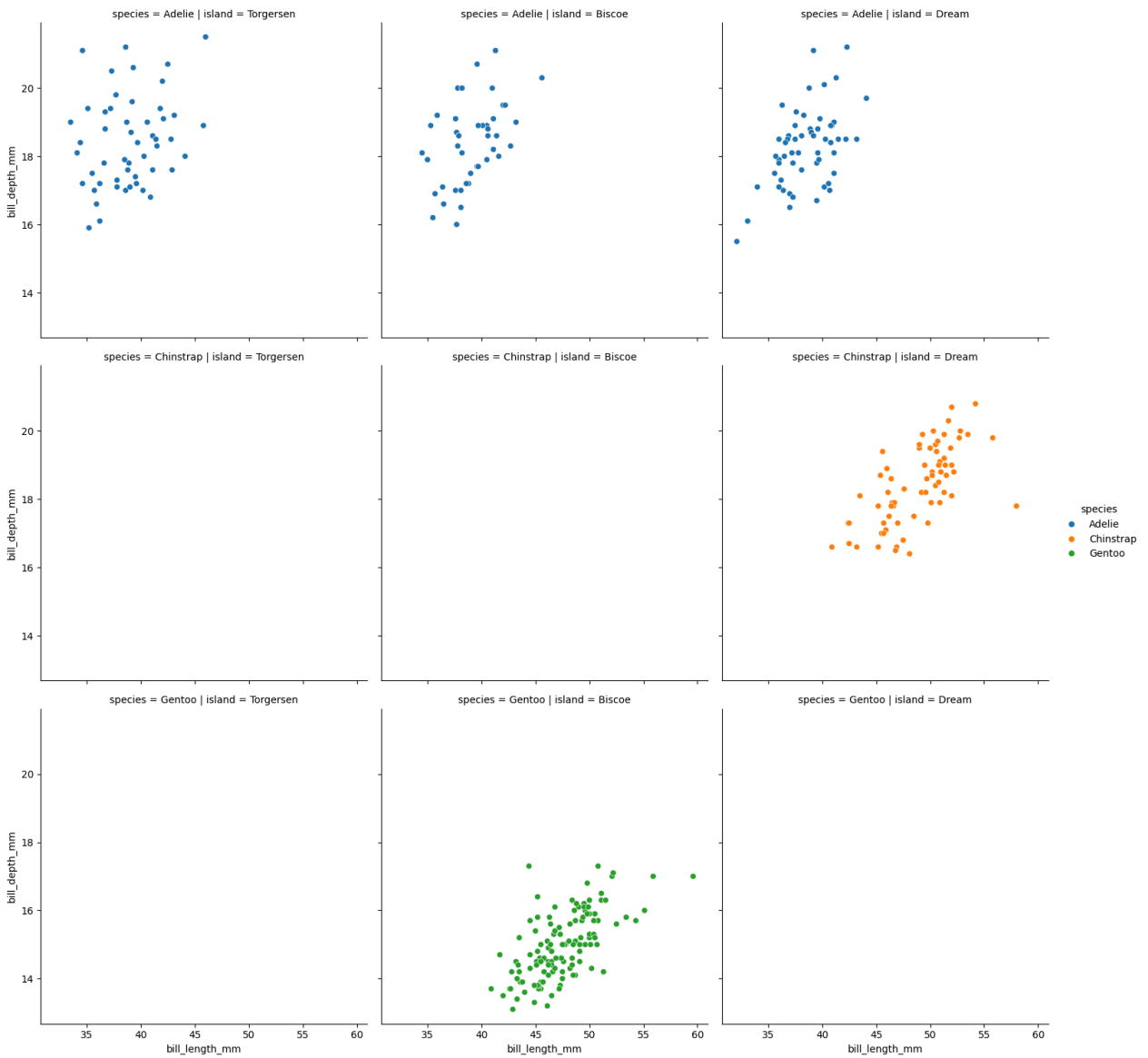

```
g = sns.relplot(
  data = penguins,
  x = "bill_length_mm",
  y = "bill_depth_mm",
  hue = "species"
)
```

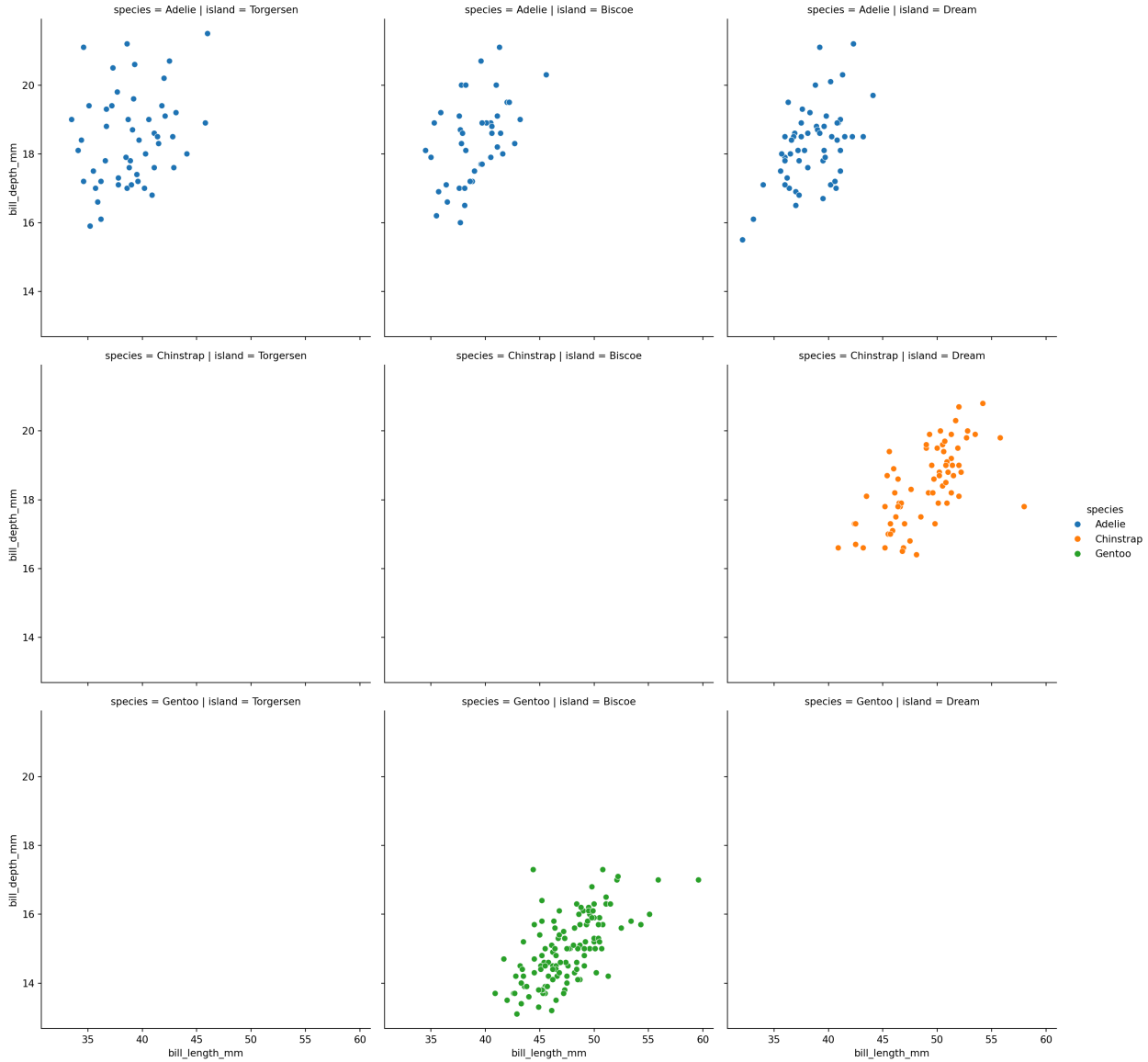


# A more complex plot

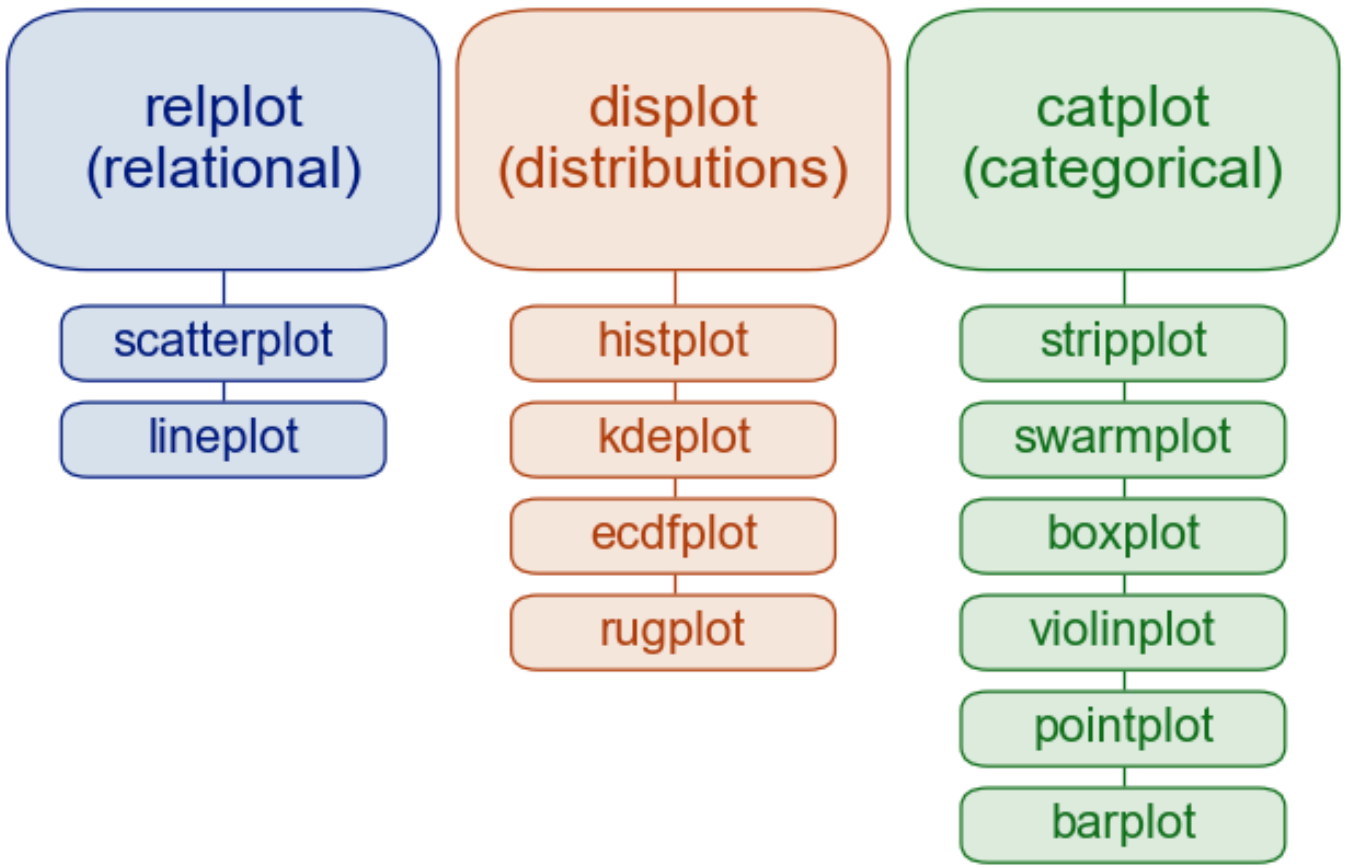
```
sns.relplot(
  data = penguins,
  x = "bill_length_mm", y = "bill_depth_mm",
  hue = "species",
  col = "island", row = "species"
)
```

# A more complex plot



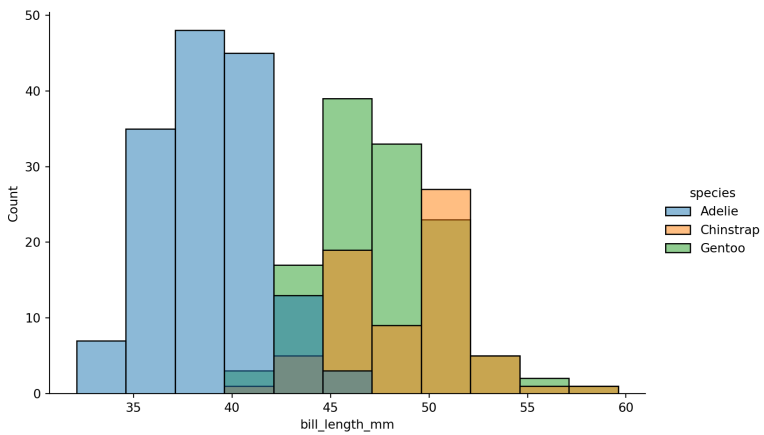

species = Adelie | island = Torgersen
species = Adelie | island = Biscoe
species = Adelie | island = Dream
species = Chinstrap | island = Torgersen
species = Chinstrap | island = Biscoe
species = Chinstrap | island = Dream
species = Gentoo | island = Torgersen
species = Gentoo | island = Biscoe
species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

# Figure-level vs. axes-level functions

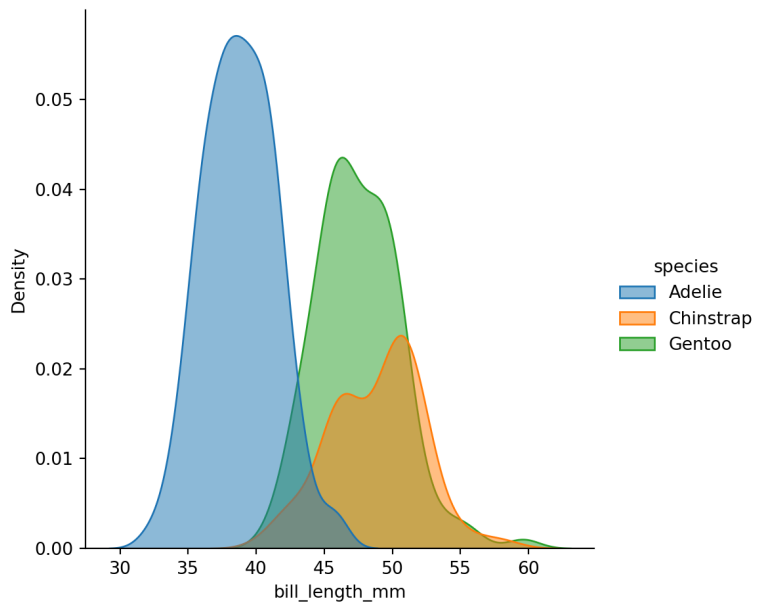


# displots

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

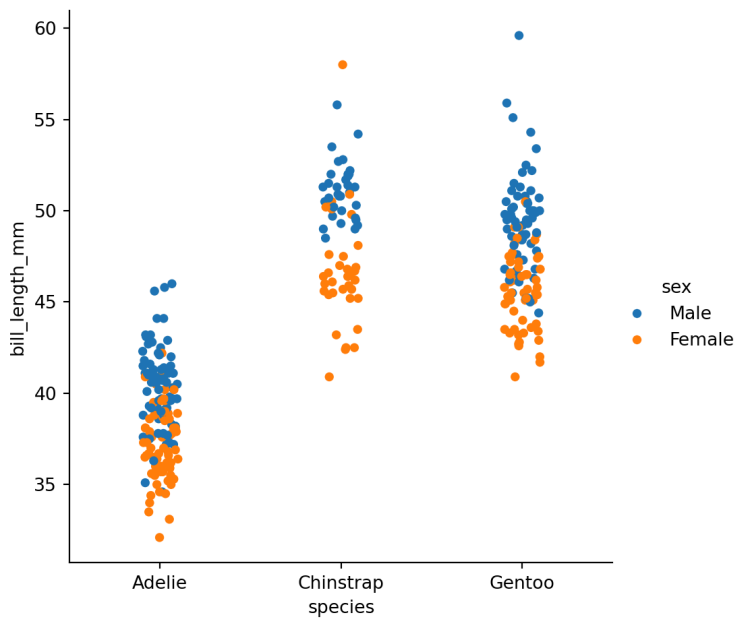



```python
g = sns.displot(
  data = penguins,
  x = "bill_length_mm", hue = "species",
  kind = "kde", fill=True,
  alpha = 0.5, aspect = 1
)
```

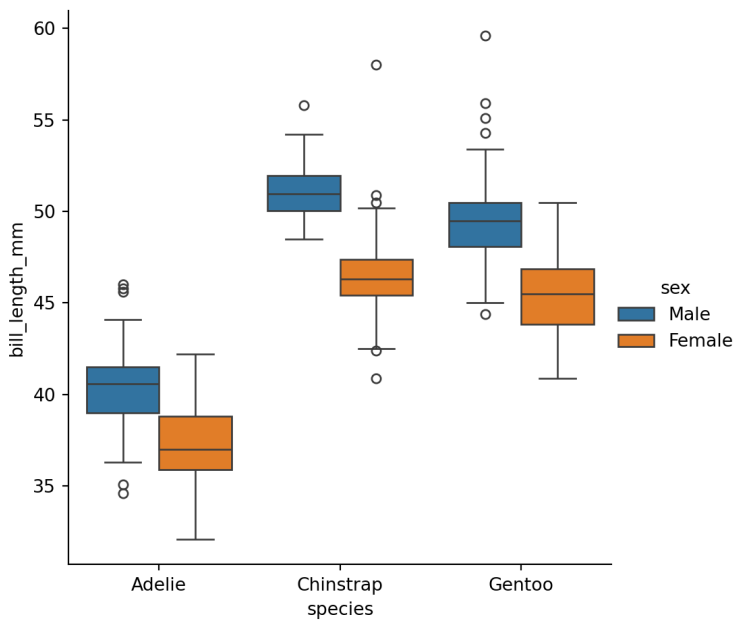


# catplots

```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"
)
```

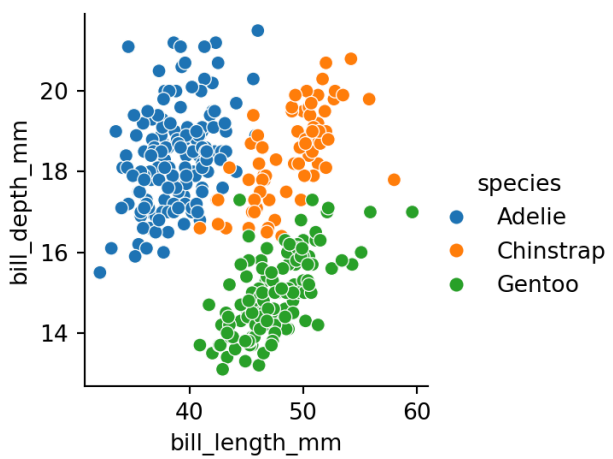


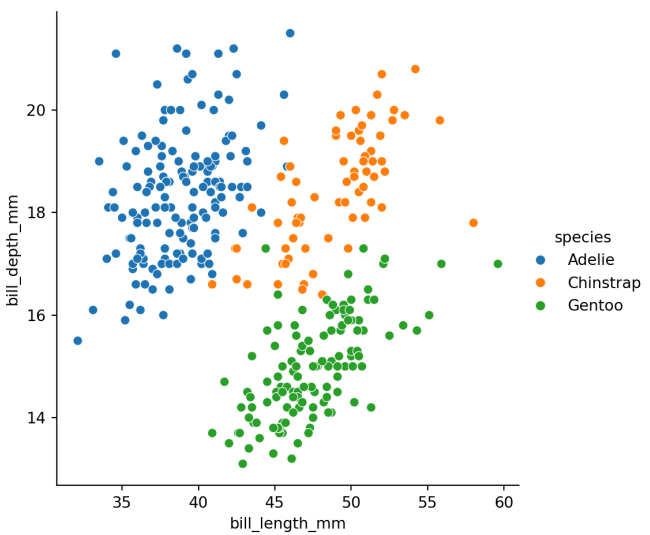
# figure-level plot size

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

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

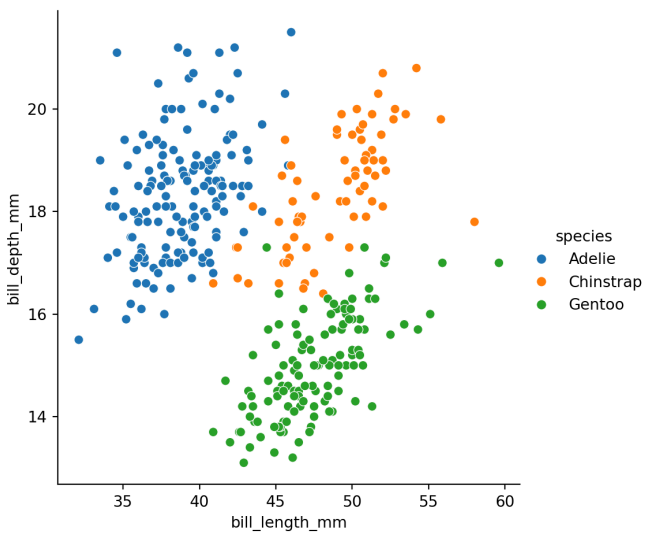
```
g = sns.relplot(
  data = penguins,
  x = "bill_length_mm", y = "bill_dept
  hue = "species",
  aspect = 1, height = 5
)
```



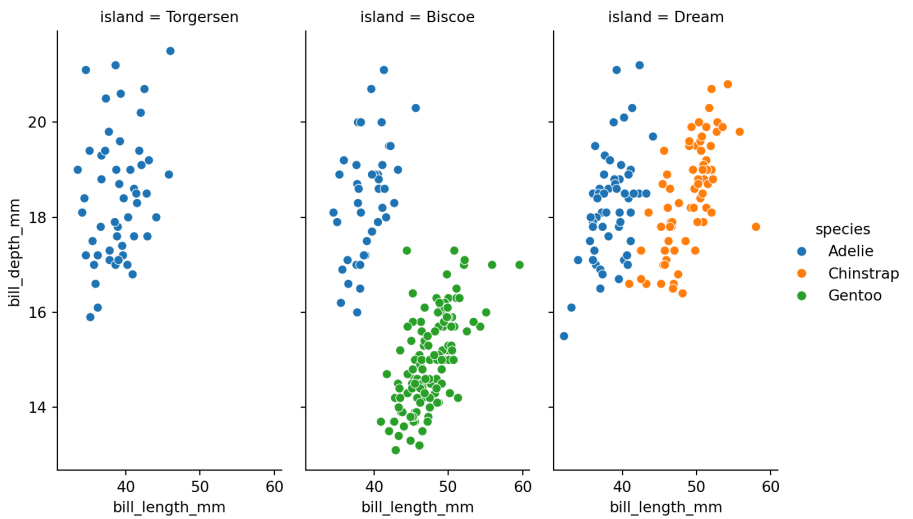



# figure-level plots

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




```python
h = sns.relplot(
  data = penguins,
  x = "bill_length_mm", y = "bill_dept
  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 0x12b393b60>
```

```
1 print(h)
```

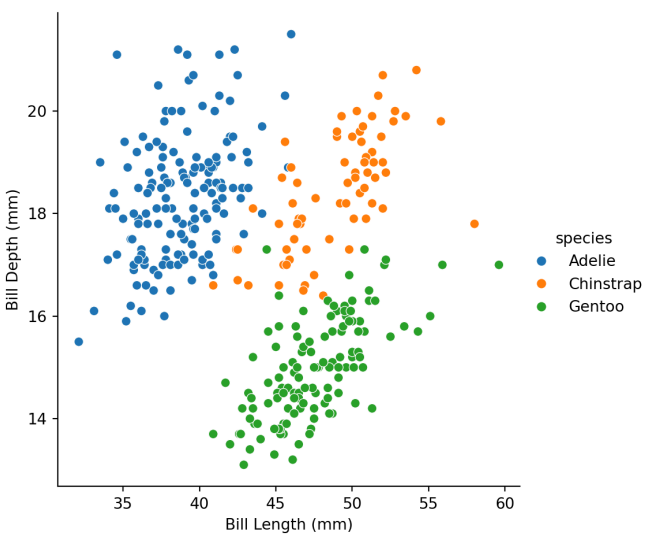
```
<seaborn.axisgrid.FacetGrid object at 0x1696753d0>
```

# FacetGird methods

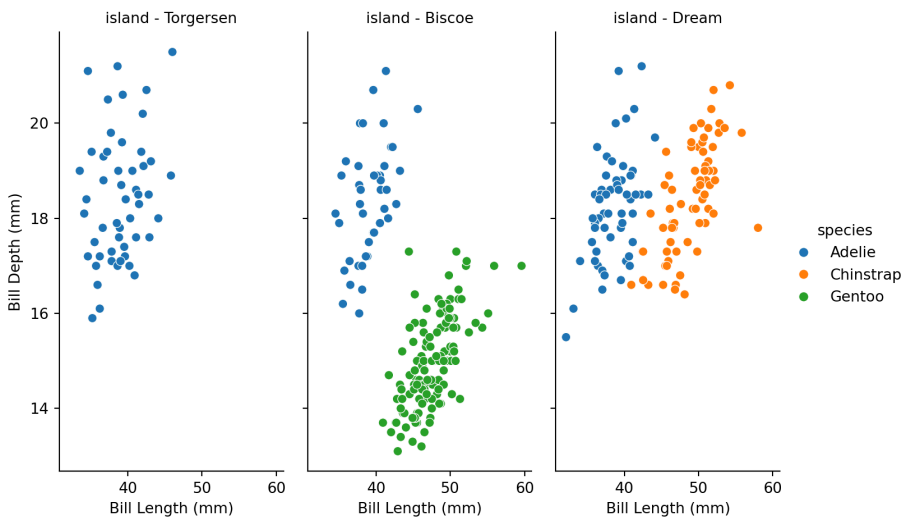
| Method | Description |
|---|---|
| `add_legend()` | Draw a legend, maybe placing it outside axes and resizing the figure |
| `despine()` | Remove axis spines from the facets. |
| `facet_axis()` | Make the axis identified by these indices active and return it. |
| `facet_data()` | Generator for name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

```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}"
)
```

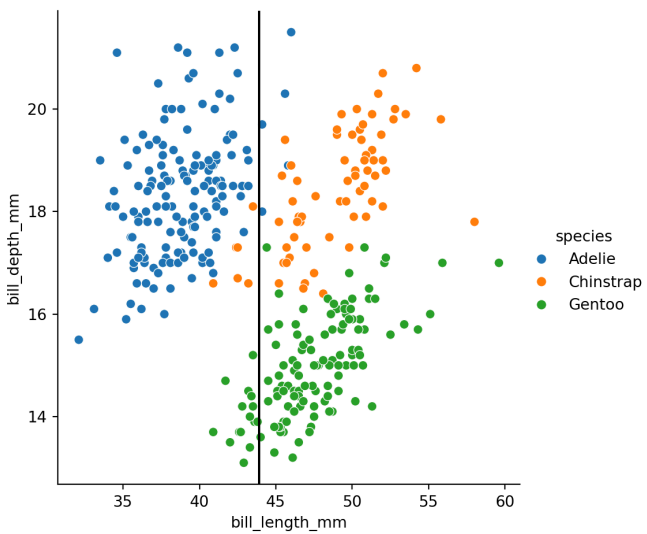


# FacetGrid attributes

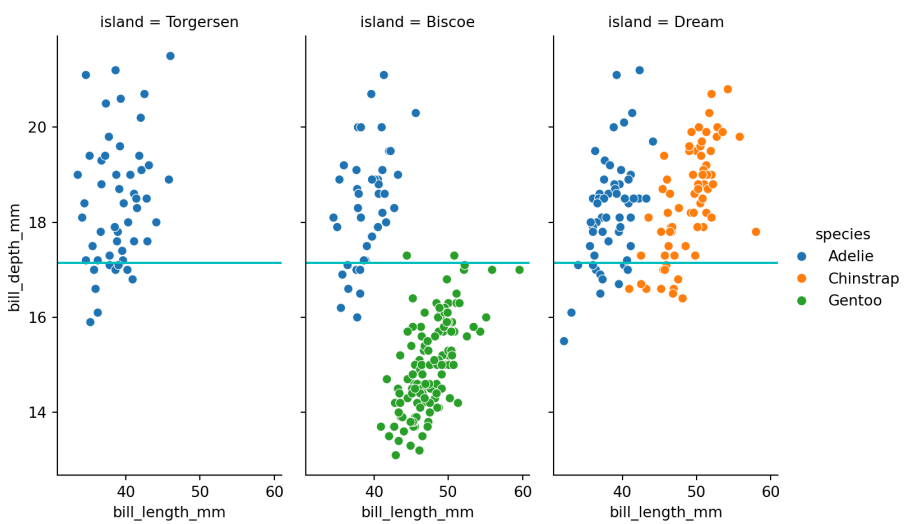
| Attribute | Description |
|---|---|
| `ax` | The `matplotlib.axes.Axes` when no faceting variables are assigned. |
| `axes` | An array of the `matplotlib.axes.Axes` objects in the grid. |
| `axes_dict` | A mapping of facet names to corresponding `matplotlib.axes.Axes`. |
| `figure` | Access the `matplotlib.figure.Figure`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

# Using axes to modify plots

```
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 = "
)
```




```
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

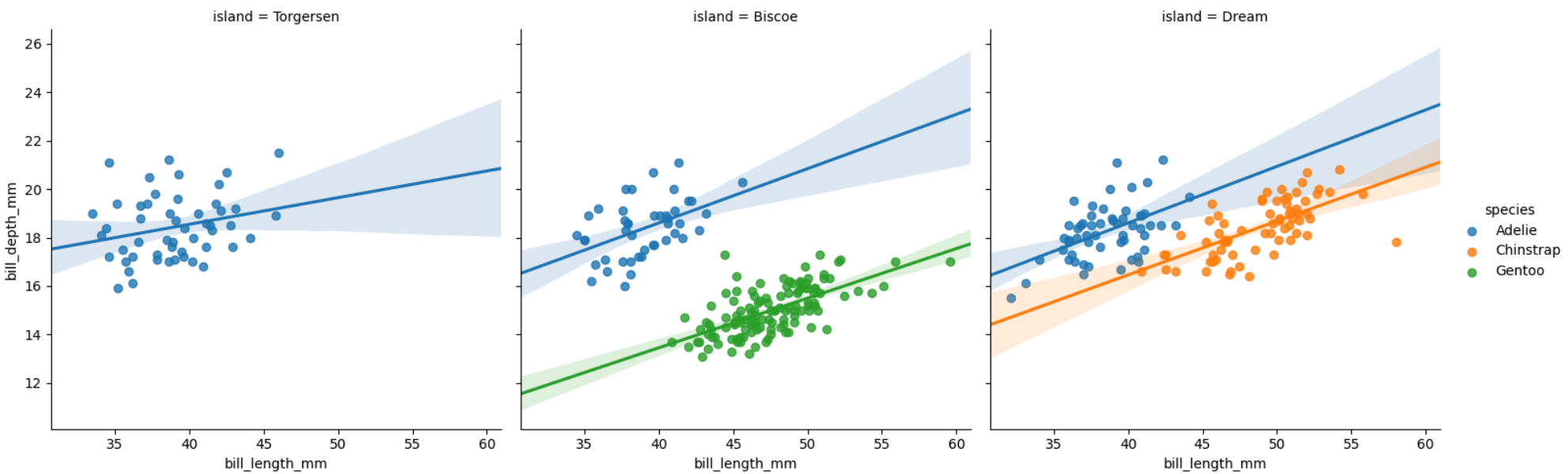
## Disadvantages:

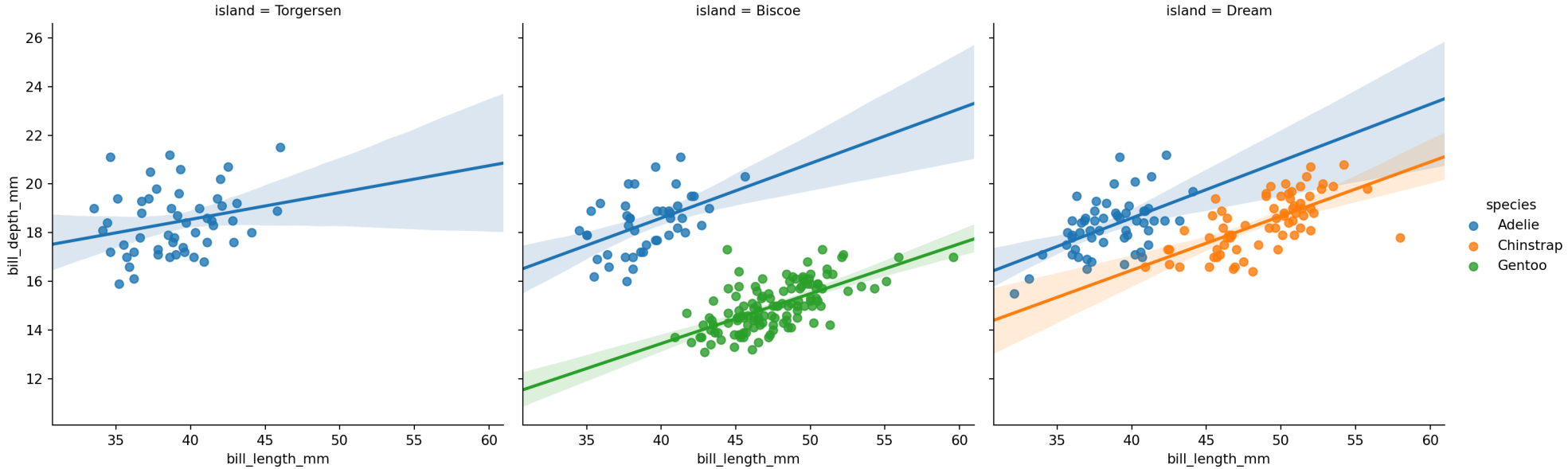
- Many parameters not in function signature
- Cannot be part of a larger matplotlib figure
- Different API from matplotlib
- Different figure size parameterization

# 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
)
```



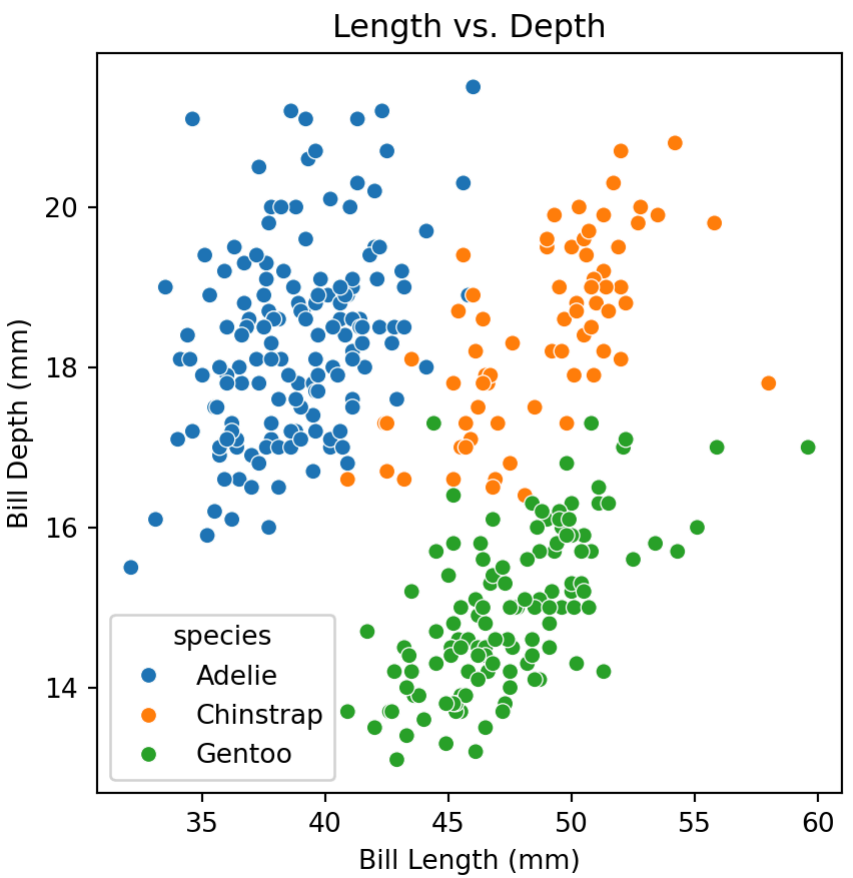

island = Torgersen
island = Biscoe
island = Dream
bill_depth_mm
bill_length_mm
species
Adelie
Chinstrap
Gentoo

# axes-level plots

# axes-level functions

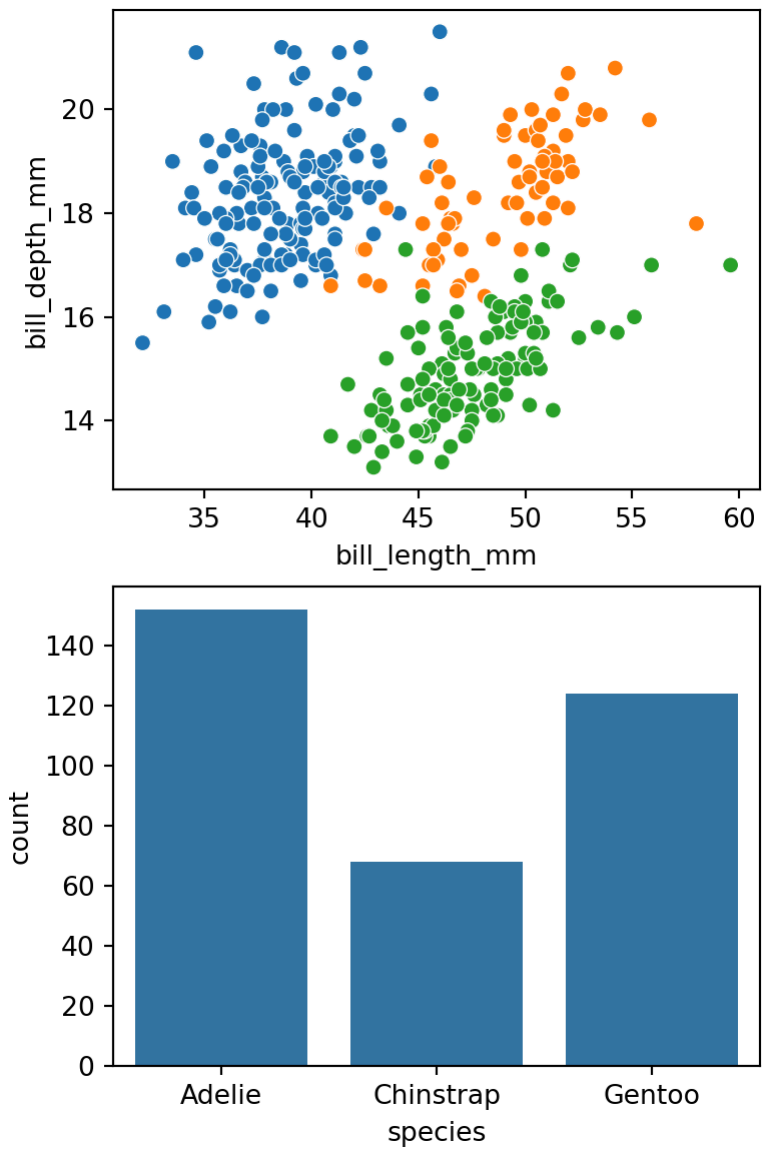
These functions return a `matplotlib.pyplot.Axes` object instead of a `FacetGrid`, giving more direct control over the plot using basic matplotlib tools.

```
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()
```



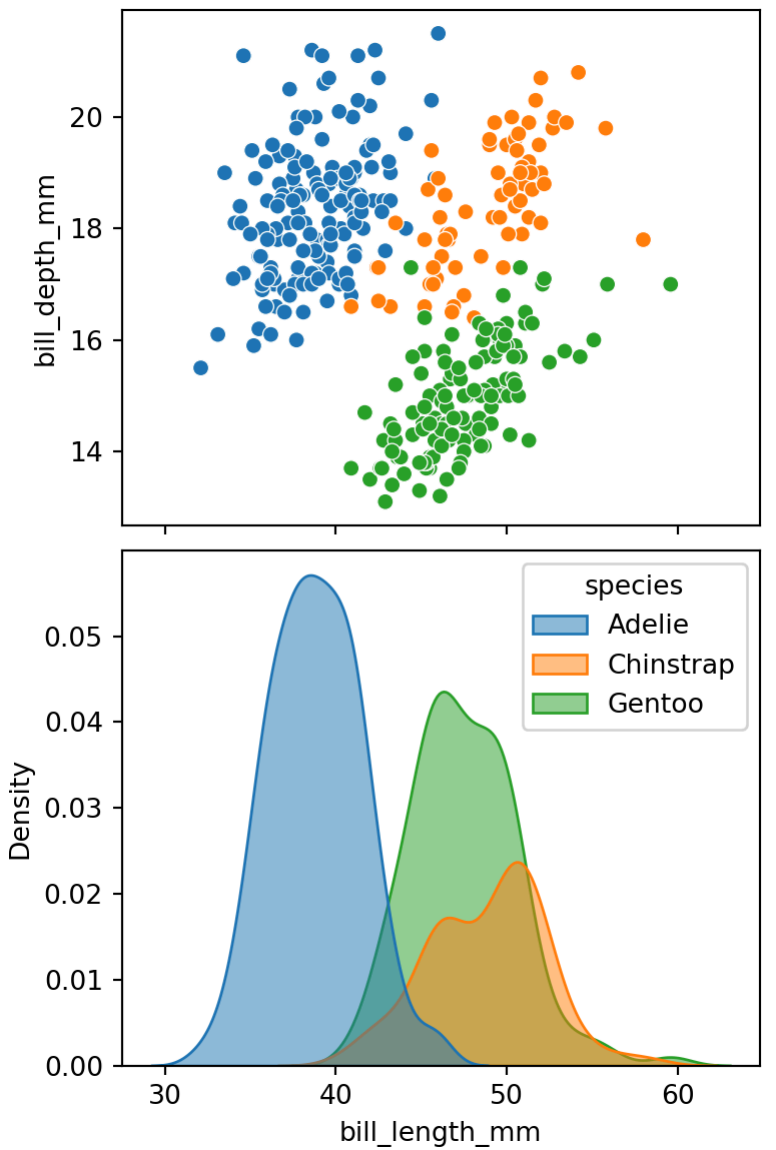
# subplots - pyplot style

```
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()
```



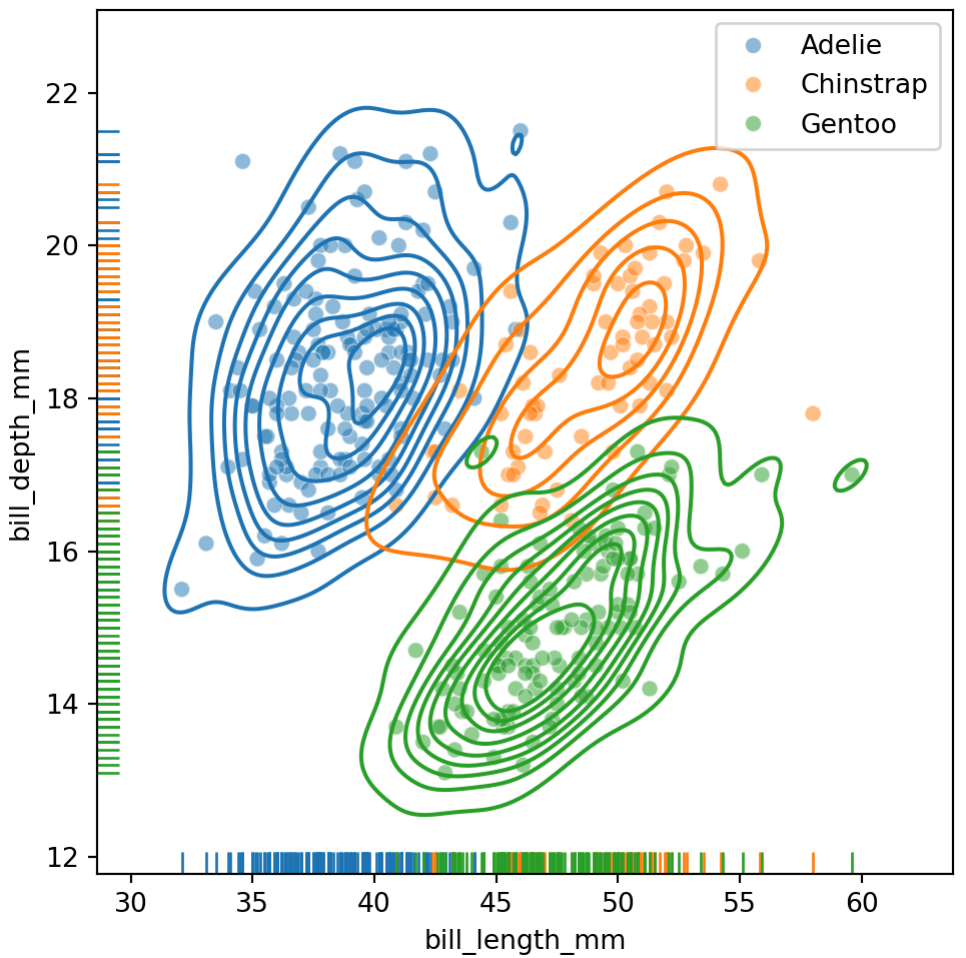
# subplots - OO style

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

sns.scatterplot(
  data = penguins,
  x = "bill_length_mm", y = "bill_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

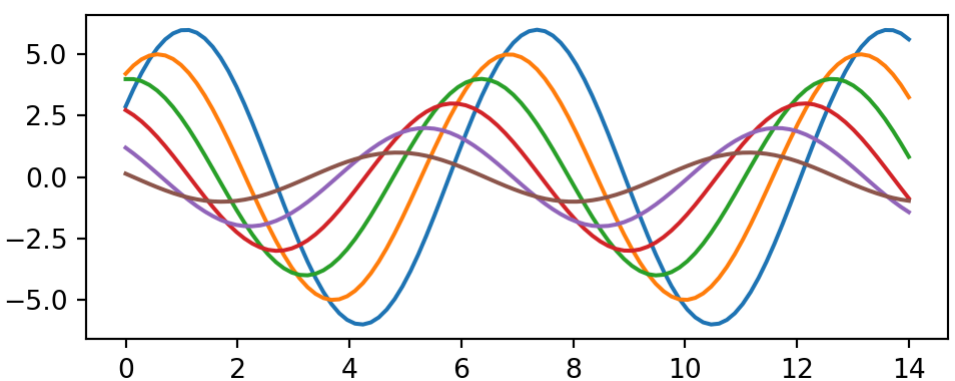
```
plt.figure(figsize=(5,5),
           layout = "constrained")

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()
```



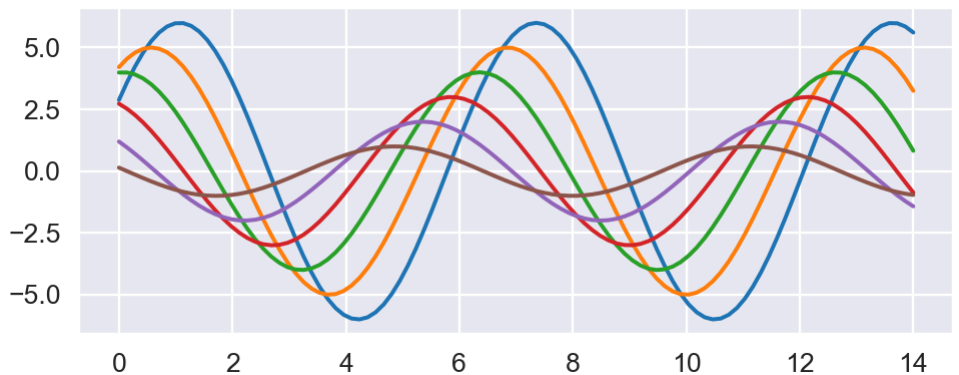
# Themes

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()`.

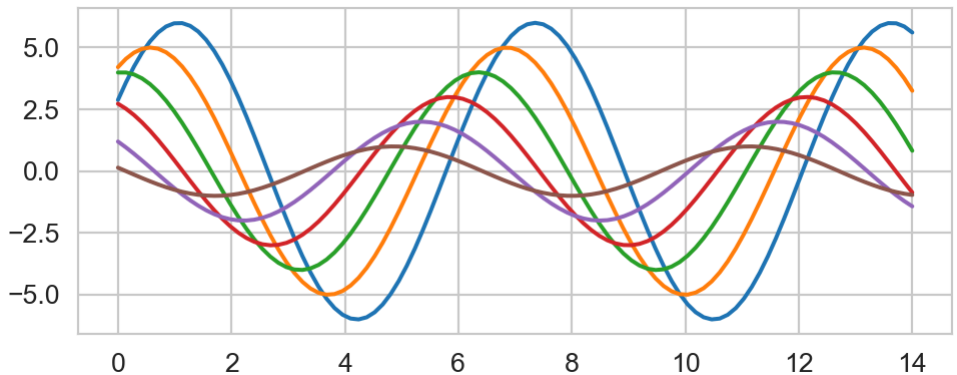
```
def sinplot():
    plt.figure(figsize=(5,2), layout = "cons
    x = np.linspace(0, 14, 100)
    for i in range(1, 7):
        plt.plot(x, np.sin(x + i * .5) * (7
    plt.show()

sinplot()
```



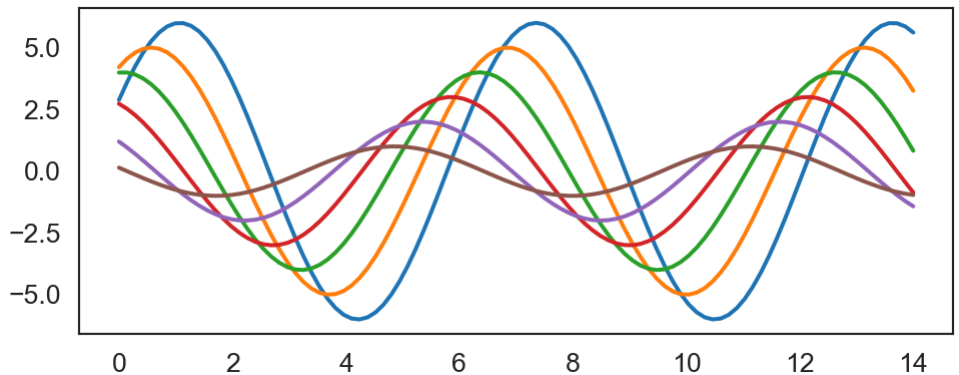

```
with sns.axes_style("darkgrid"):
  sinplot()
```



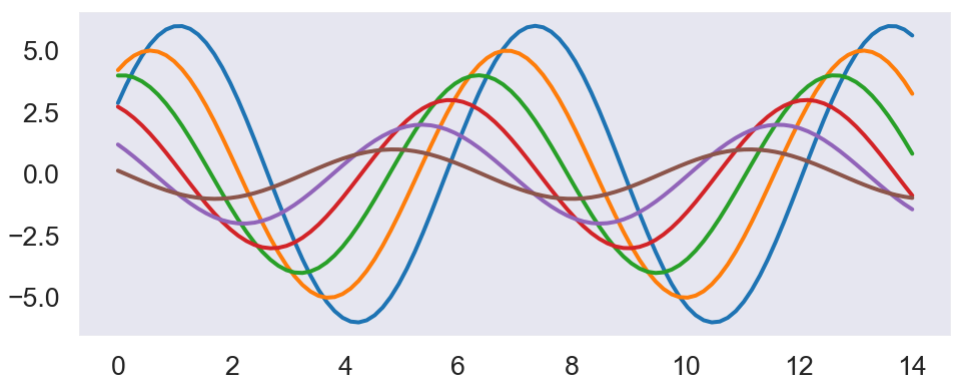
```
with sns.axes_style("whitegrid"):
  sinplot()
```



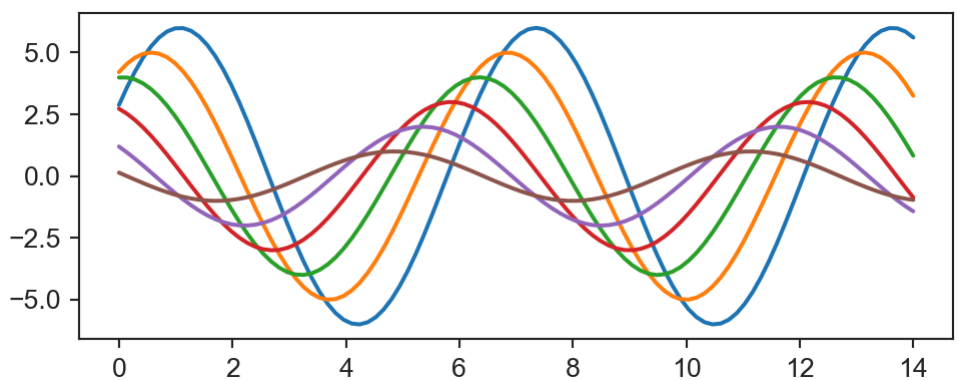

```
with sns.axes_style("white"):
  sinplot()
```




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

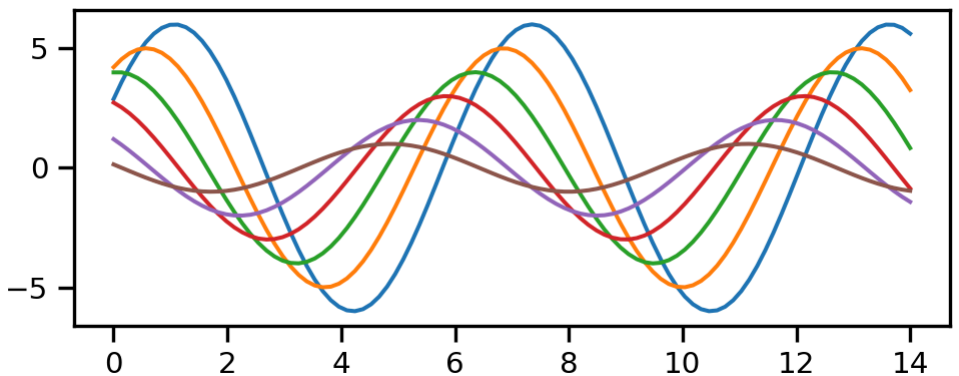



```
with sns.axes_style("ticks"):
  sinplot()
```

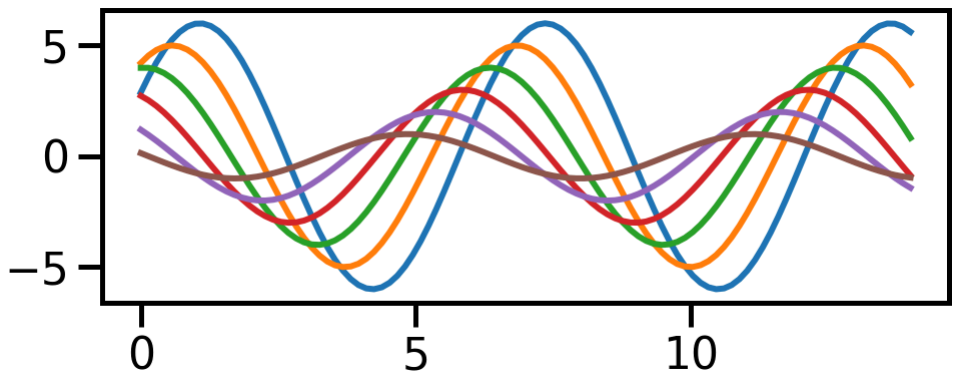


# Context

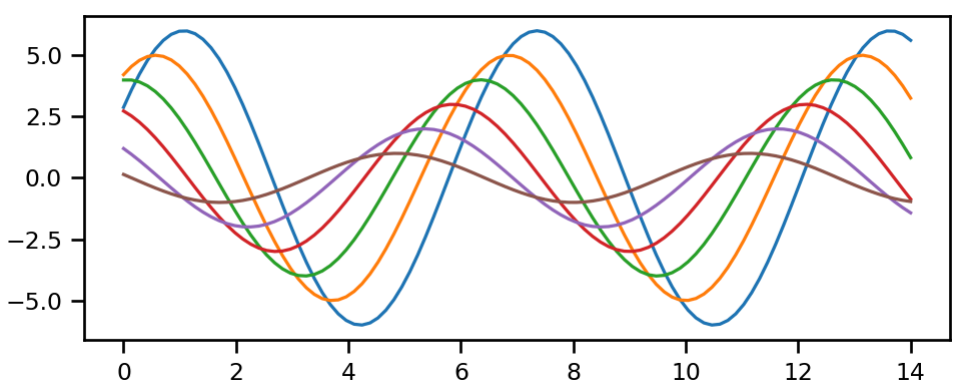
```
1 sns.set_context("notebook")
2 sinplot()
```



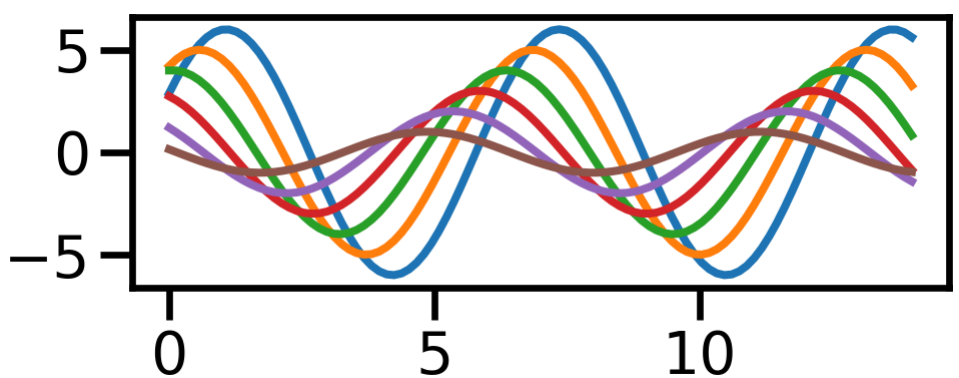

```
1 sns.set_context("talk")
2 sinplot()
```




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




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



# Color palettes

All of the examples below 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")
```

# 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")
```

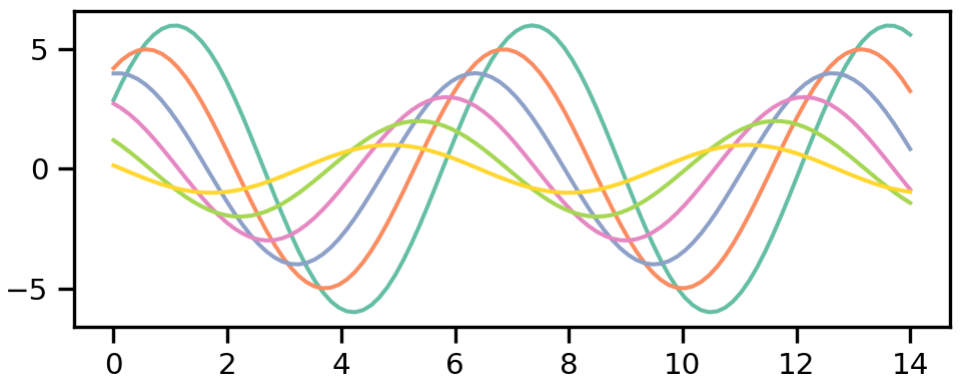
```
show_cont_palette("dark:salmon_r")
```

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

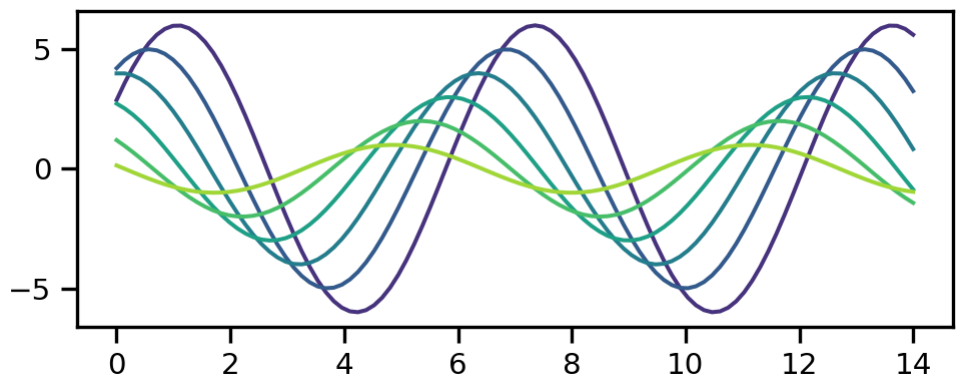
# Applying palettes

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

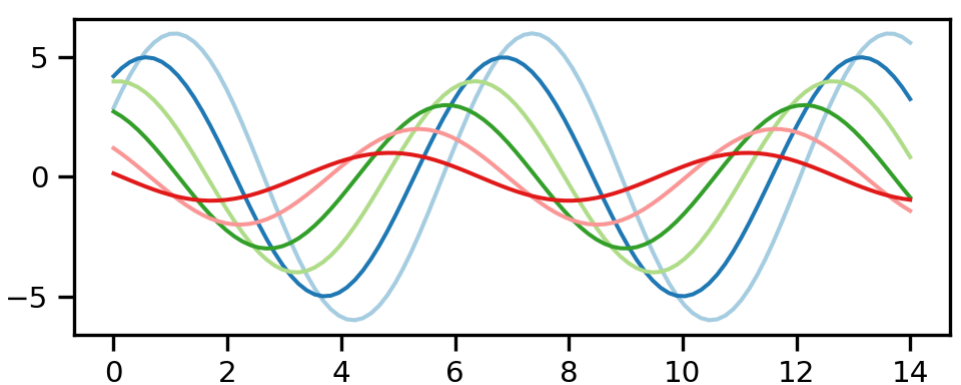
```
1 sns.set_palette("Set2")
2 sinplot()
```




```
1 sns.set_palette("viridis")
2 sinplot()
```




```
1 sns.set_palette("Paired")
2 sinplot()
```




```
1 sns.set_palette("rocket")
2 sinplot()
```