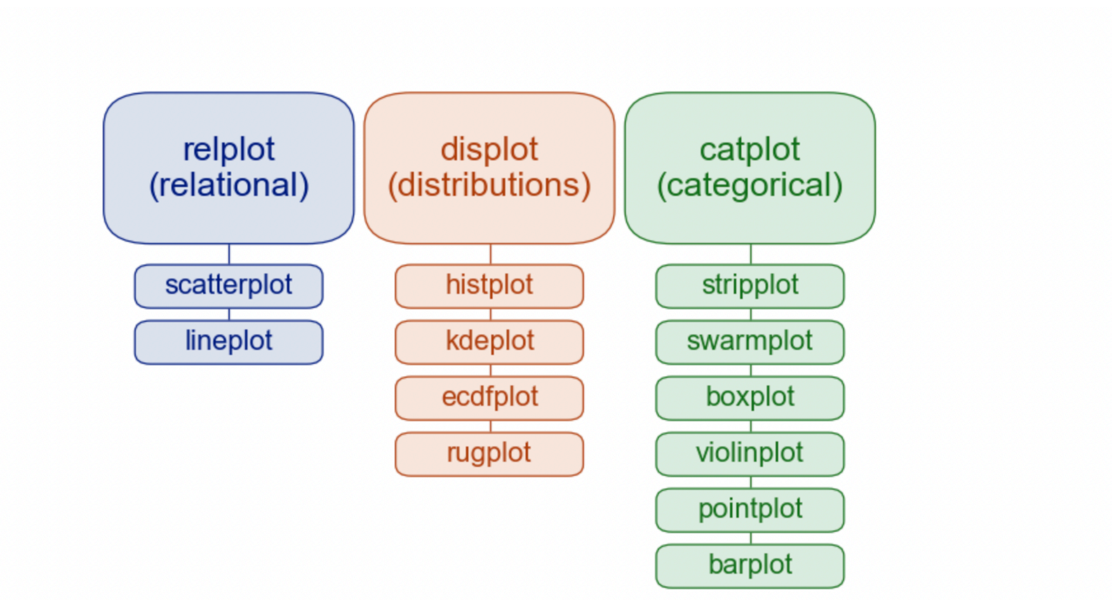
tu12_SeabornReview

February 23, 2023

1 Seaborn Review

Seaborn is a really great package for quickly producing nice plots. It's basic structure looks like this



The top row (the larger boxes) are figure-level plots. They handle much of the busywork of making a nice figure for you, and also allow you to select which underlying plot type to use (e.g. lineplot vs scatterplot). The underlying plots are axes-level plots. If you call them directly, they will return a matplotlib `axes` object, which you can then use to customize the plot. The topmost underlying plots are the default plot types for the figure-level plots.

There are also a couple functions, `pairplot()` and `jointplot()` that produce some common figures using a mix of plot types. We've made examples of both of these "by hand" already in this course.

Seaborn is built on top of matplotlib so, ultimately, everything in a seaborn figure is an `axes` or other matplotlib `artist`. This means that you can always use matplotlib methods if you need to do some low level customization of your figures.

1.1 Preliminaries

First, let's import what we'll need:

[1]:
```python
# This is all we should need in theory
import seaborn as sns

# But I had to do this to get plots to show for some random reason
import matplotlib.pyplot as plt
%matplotlib inline
```

1.2 Figure level plots

We'll start with some figure level plots.

1.2.1 Relational plots

Several example data sets come with seaborn. Here's one about tipping:

[2]:
```python
tips = sns.load_dataset("tips")
```

Let's peek at the data set after loading (always!):

[3]:
```python
tips
```

[3]:
```
     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 x 7 columns]
```

Use the cell below to `describe` the numerical data in `tips`:

[4]:
```python
tips.describe()
```

[4]:
```
       total_bill         tip        size
count  244.000000  244.000000  244.000000
mean    19.785943    2.998279    2.569672
std      8.902412    1.383638    0.951100
min      3.070000    1.000000    1.000000
25%     13.347500    2.000000    2.000000
50%     17.795000    2.900000    2.000000
```

```
75%      24.127500    3.562500    3.000000
max      50.810000   10.000000    6.000000
```

Based on this summary of the numerical data in `tips`, do you think this is a recent US data set? * `No, I don't think this is the recent US data set.`

Count the number of smokers in `tips`.

[18]:
```
# Make Layout for 'Yes'
smoker = tips[['sex','smoker']]
smoker_yes = smoker[smoker['smoker'] == 'Yes']
smoker_yes
```

[18]:
```
        sex smoker
56     Male    Yes
58     Male    Yes
60     Male    Yes
61     Male    Yes
62     Male    Yes
..      ...    ...
234    Male    Yes
236    Male    Yes
237    Male    Yes
240  Female    Yes
241    Male    Yes

[93 rows x 2 columns]
```

[20]:
```
smoker_yes['smoker'].count()
```

[20]: 93

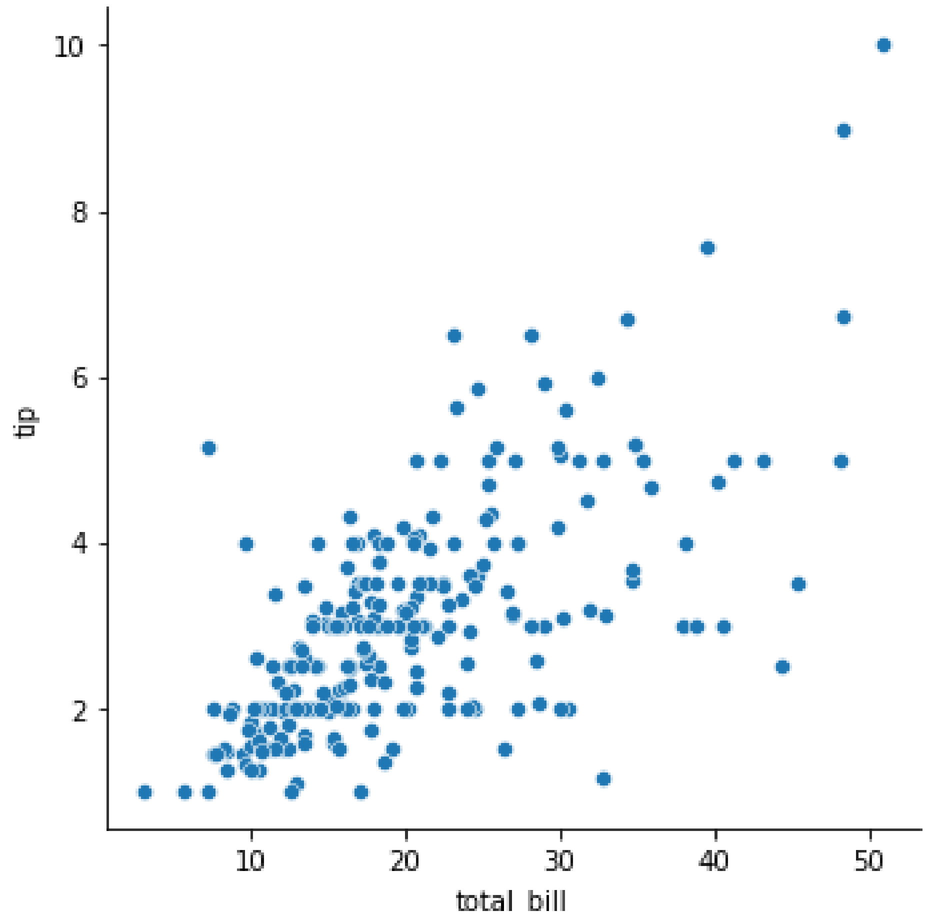
Does this confirm or refute your guess about the origin of the data made on the numerical summary? * `I am still confident that this data is ont recent.`

---

scatter plots Let's make a call to `relplot`:

[21]:
```
sns.relplot(data=tips, x="total_bill", y="tip")
```

[21]: `<seaborn.axisgrid.FacetGrid at 0x7f96412fcbe0>`



We can see that the vast majority of tips fall below a 20% line, so it probably is an old data set (assuming its from the US).

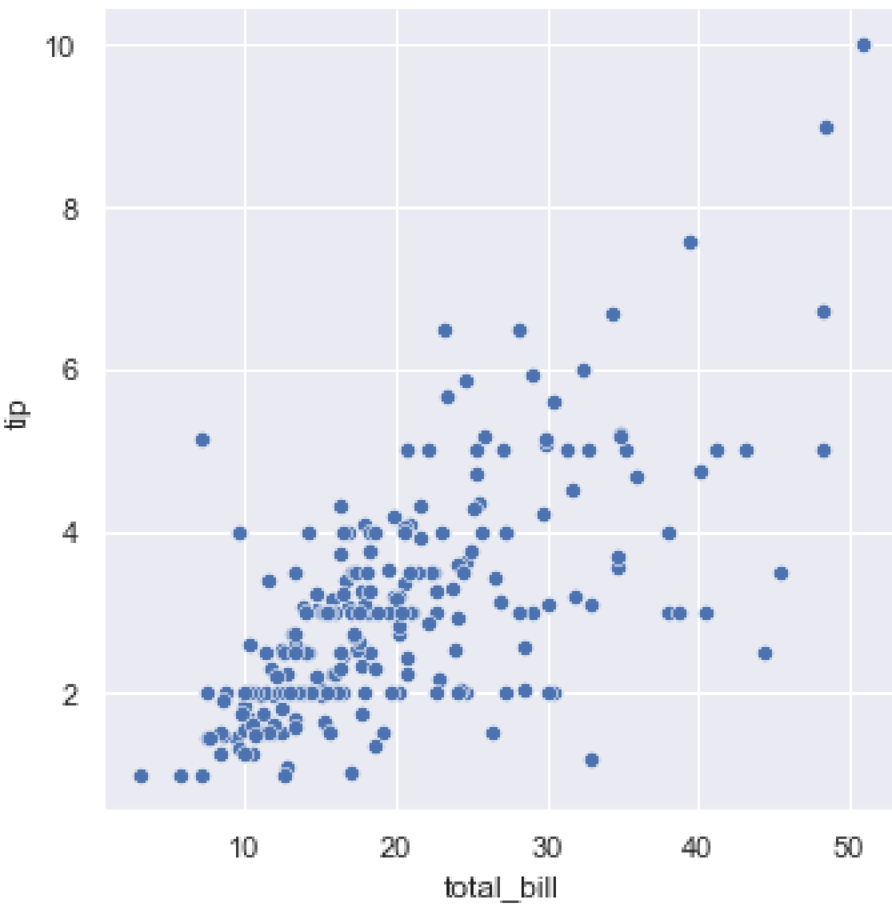
In terms of the plot *per se* though… so far, not a huge jump up from `matplotlib`…

Seaborn has five built in themes: `darkgrid` (the default), `whitegrid`, `dark`, `white`, and `ticks`. Let's set the default theme and replot.

[22]:
```
sns.set_theme()
```

[23]:
```
sns.relplot(data=tips, x="total_bill", y="tip")
```

[23]: `<seaborn.axisgrid.FacetGrid at 0x7f96413a58b0>`

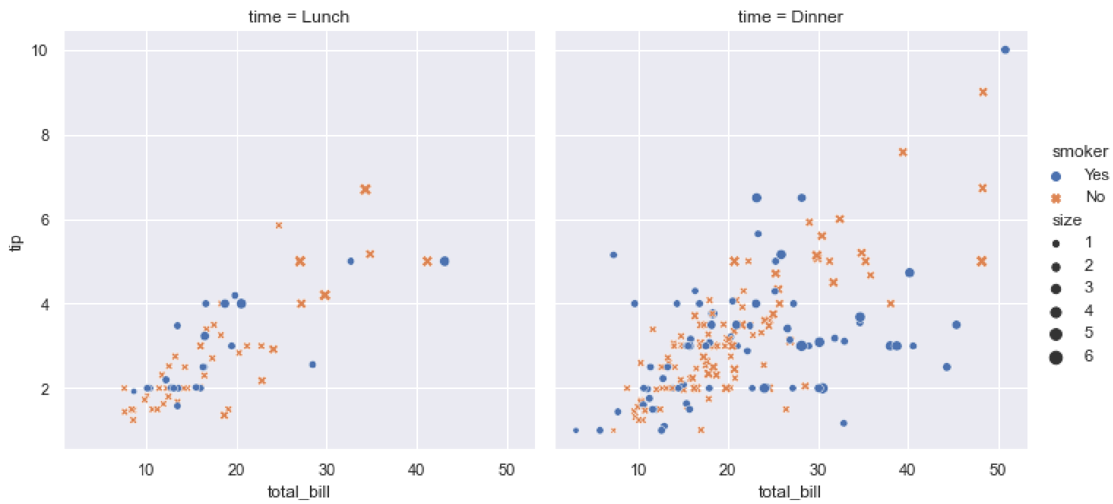


That’s better, and not unlike ggplot2’s default theme in R!

Now, let’s look at how easy it is to make a fancy plot by assigning other variables to plot aesthetic properties.

[24]:
```python
sns.relplot(
    data=tips,
    x="total_bill", y="tip", col="time",
    hue="smoker", style="smoker", size="size",
)
```

[24]: <seaborn.axisgrid.FacetGrid at 0x7f963744dd90>

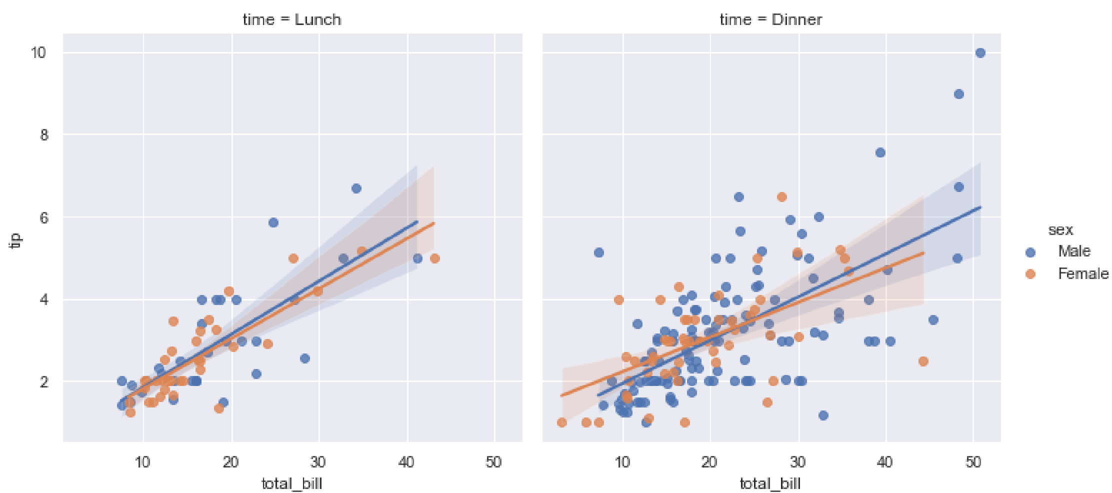


Now that's a lot of plot for very little effort!

scatter plot with regression Let's look at the tipping rate for females vs. males.

[25]:
```
sns.lmplot(data=tips, x="total_bill", y="tip", col="time", hue="sex")
```

[25]: <seaborn.axisgrid.FacetGrid at 0x7f96418778b0>



Those look identical, so there is (was) gender equality in tipping at least.

But let's do take a moment at what seaborn has done under the hood for us here. It has:

- run a linear regression on tip vs. bill separately for each group that we defined with our column and color specification
- plotted the regression lines

- computed 95% CIs on the fits, and plotted those as shaded areas.

Impressive. For those of you coming from R/tidyverse land, this is probably reminding you of ggplot() in more ways than the appearance of the background.

You might be wondering where we got `lmplot()`, as it's not one of the plots listed in the seaborn diagram above. Seaborn does have some "hidden gems" like `lmplot()` and `jointplot()` that you'll only discover by looking at the seaborn documentation.

Pro tip: There are two *really* useful places in the seaborn documentation to look for stuff:

- the function interface reference
- the gallery

In the gallery, you can browse plot types by appearance; the mouse-over text will tell you the function used to make the plot, and clicking on the plot will take you to the example code!

Now let's look at the tipping rate for smokers vs. non-smokers.

[26]:
```
sns.lmplot(data=tips, x="total_bill", y="tip", col="time", hue="smoker")
```

[26]: <seaborn.axisgrid.FacetGrid at 0x7f96413a5340>

Hm. It looks like, at dinner at least, smokers may tip less (although the uncertainty on the fits is too high to be completely sure).

line plots Scatter plots are usefully for pairwise data in which the pairs themselves are independent of one another.

In other cases, the data are ordered by the x values. Often this is due to the y values unfolding over time.

Let's load another data set to look at this.

[27]:
```
dots = sns.load_dataset("dots")
```

[28]:
```python
# take a peek as always
dots
```

[28]:
```
    align choice  time  coherence  firing_rate
0    dots     T1   -80        0.0    33.189967
1    dots     T1   -80        3.2    31.691726
2    dots     T1   -80        6.4    34.279840
3    dots     T1   -80       12.8    32.631874
4    dots     T1   -80       25.6    35.060487
..    …      …    …         …          …
843  sacc     T2   300        3.2    33.281734
844  sacc     T2   300        6.4    27.583979
845  sacc     T2   300       12.8    28.511530
846  sacc     T2   300       25.6    27.009804
847  sacc     T2   300       51.2    30.959302

[848 rows x 5 columns]
```

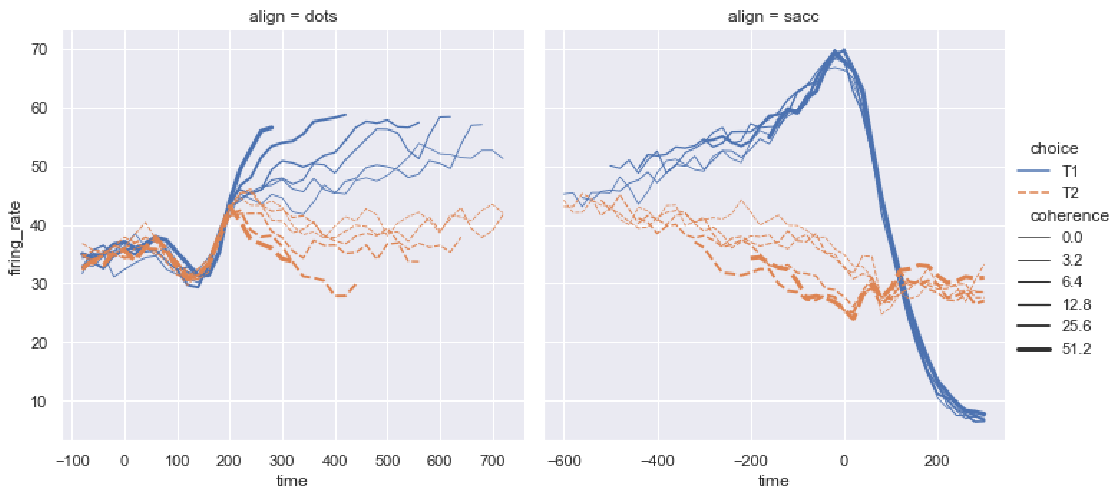
The main data here are the firing rate of a neuron in the superior colliculus (a brain area crucial for moving the eyes) as a function of time. In the experiment, moving stimuli of various strengths ('coherence') appear on the screen, and the subject has to decide which way they moved (left or right, say). Later, the stimuli disappear and are replaced by two targets. The subject has to indicate their choice by moving their eyes to the appropriate target (left or right). Sometimes, the target corresponding to the subject's choice was within the neurons "area of responsibility" in the visual field (`choice` = `T1`), and sometimes without (`choice` = `T2`)

The neural recordings can either be aligned in time to the moment the stimuli appeared (`align` = `dots`) or to the precise moment the eye movement (a saccade) began (`align` = `sacc`)

Let's plot!

[29]:
```python
sns.relplot(
    data=dots, kind="line",
    x="time", y="firing_rate", col="align",
    hue="choice", size="coherence", style="choice",
    facet_kws=dict(sharex=False),
)
```

[29]: <seaborn.axisgrid.FacetGrid at 0x7f9641dfb160>



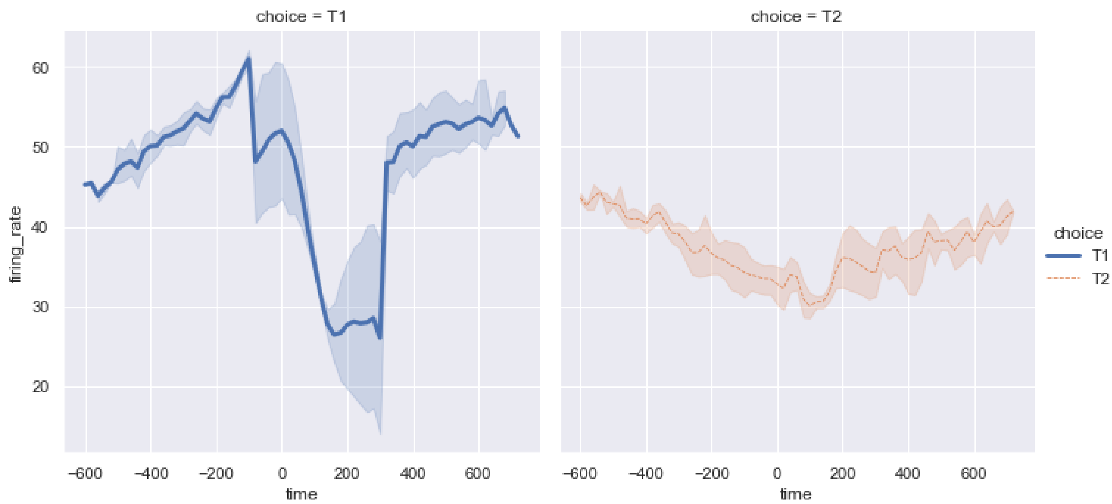
Even if we knew nothing about the experiment, we could see that there is cool stuff going on. The plot makes it very clear that there is a huge effect of choice (eye movement direction), there is differential build-up of activity starting about 200 msec after the stimulus comes on, and that this neuron is almost certain involved in driving the eye movement in the T1 direction.

That's a lot of plot for very little effort. Notice that we didn't even have to remember to use different arguments for `size` and `style` when we switched from scatter to line plots – we used the same arguments and seaborn interpreted them as "line size" or "marker size", etc., as appropriate.

Re-make the above plot so that color codes stimulus strength and size codes choice.

[35]:
```
sns.relplot(
    data=dots, kind="line",
    x="time", y="firing_rate", col="choice",
    hue="choice", size="choice", style="choice",
    facet_kws=dict(sharex=False),
)
```

[35]: <seaborn.axisgrid.FacetGrid at 0x7f9623a600a0>



Whether that made the plot better or worse is debatable, but the point is that we could so easily change it and see!

Automagic uncertainties (graphical stats) Let's load another data set. This data set is from a functional MRI experiment.

[36]:
```
fmri = sns.load_dataset("fmri")
```

[37]:
```
# peek at the data
fmri
```

[37]:
```
     subject  timepoint event    region    signal
0        s13         18  stim  parietal -0.017552
1         s5         14  stim  parietal -0.080883
2        s12         18  stim  parietal -0.081033
3        s11         18  stim  parietal -0.046134
4        s10         18  stim  parietal -0.037970
…        …          …   …       …       …
1059      s0          8   cue   frontal  0.018165
1060     s13          7   cue   frontal -0.029130
1061     s12          7   cue   frontal -0.004939
1062     s11          7   cue   frontal -0.025367
1063      s0          0   cue  parietal -0.006899

[1064 rows x 5 columns]
```

So we have a fairly large data set consisting of 5 variables:

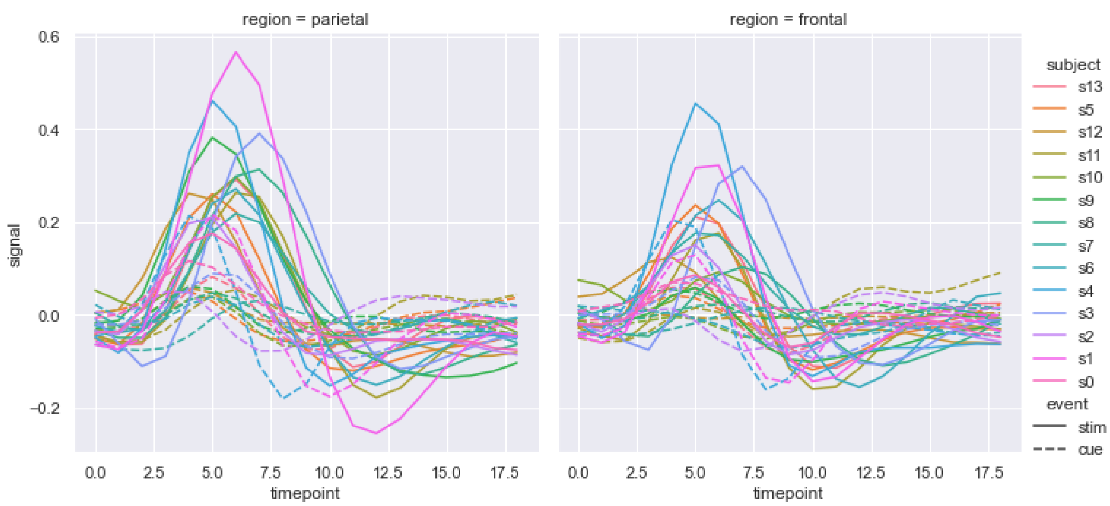
- an fMRI signal

- time
- brain area
- type of event
- person

We can assign all 5 variables to aesthetic elements of a figure:

[38]:
```
sns.relplot(
    data=fmri, kind="line",
    x="timepoint", y="signal", col="region",
    hue="subject", style="event",
)
```

[38]: <seaborn.axisgrid.FacetGrid at 0x7f9623dbe940>

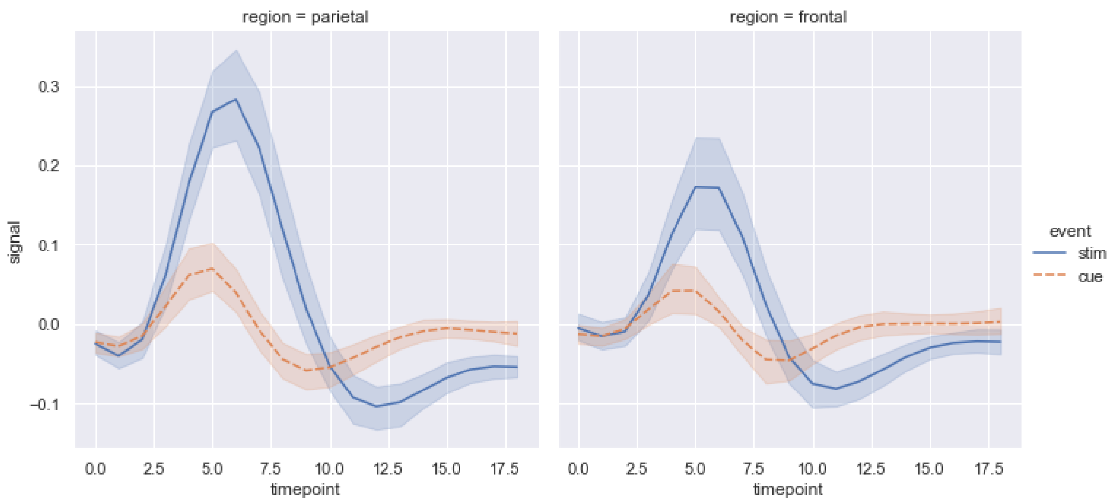


That worked, and an aficionado might be able to see what's going on. But the plot might be easier to interpret if we took the mean across subjects and just plot that (which was probably the whole point of running multiple subjects in this experiment).

Let's see what happens if we leave subject out of our aesthetic specification…

[39]:
```
sns.relplot(
    data=fmri, kind="line",
    x="timepoint", y="signal", col="region",
    hue="event", style="event",
)
```

[39]: <seaborn.axisgrid.FacetGrid at 0x7f9623e0fcd0>



Nice! The `relplot()` function figured out that, if we didn't a given variable explicitly coded in our plot, then we probably wanted to average across it.

And because seaborn was written by an actual scientist, `lineplot()` (which we called above via the `kind` argument) also included 95% CIs on the mean computed by bootstrapping.

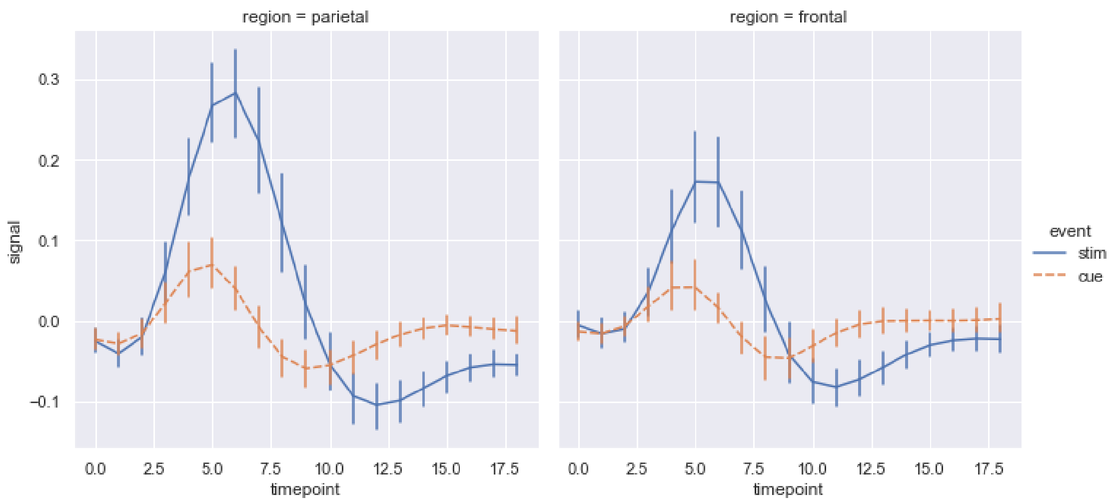
In the above plot, it looks like the standard deviation of activation might be proportional to mean activation.

Check this quickly by just re-making the above plot, but having seaborn compute the standard deviation instead of the mean for us.

The doc page for `lineplot()` is here.

[42]:
```
sns.relplot(
    data=fmri, kind="line",
    x="timepoint", y="signal", col="region",
    hue="event", style="event", err_style="bars"
)
```

[42]: `<seaborn.axisgrid.FacetGrid at 0x7f9628ae9f70>`



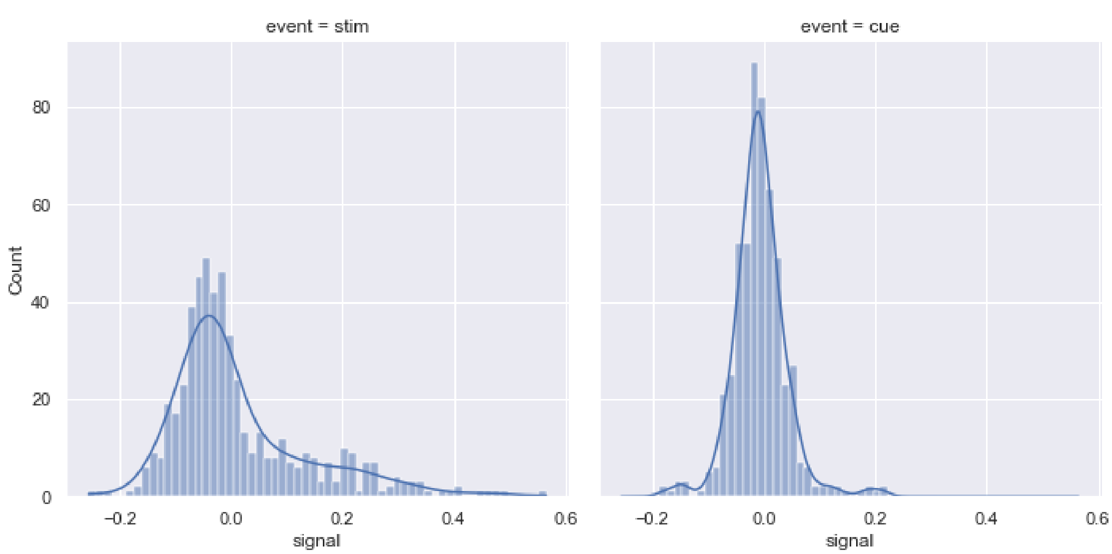
Sure enough!

1.2.2 Distribution plots

Seaborn makes it very easy to plot data distributions. Here's one for our fMRI data. The `kde` argument is going to add a 'kernel density estimate' (a best guess as to what the smooth version of the distribution looks like) for us.

[43]:
```
sns.displot(data=fmri, x="signal", col="event", kde=True)
```

[43]:
```
<seaborn.axisgrid.FacetGrid at 0x7f9624bdfe50>
```

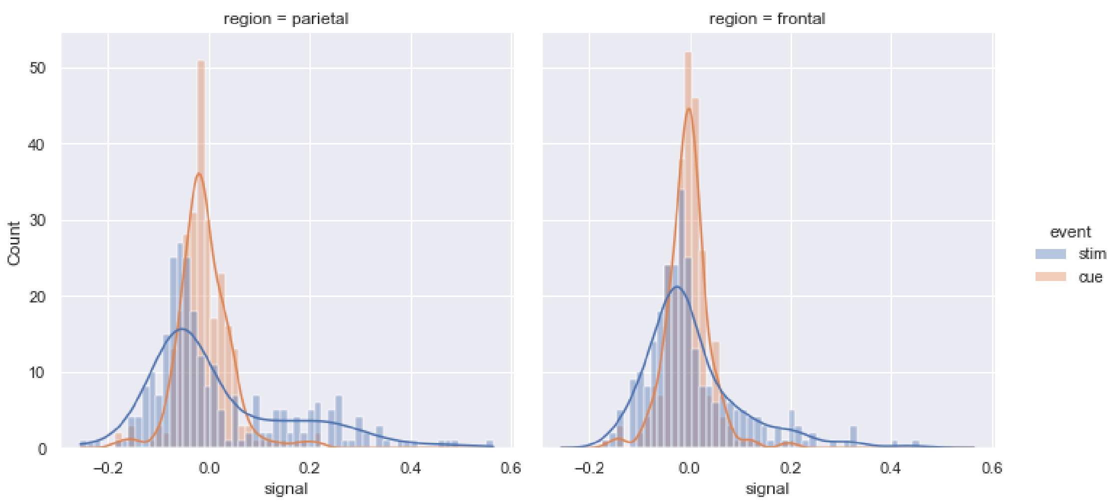


That was easy but it's a little hard to compare these two distributions as is. Is overall activation higher for stimuli vs. cue? Or is overall activation the thing that matters?

Re-make the figure above so that visually comparing this distributions is easier. Hint: change the aesthetic property that 'event' maps to.

[51]:
```
sns.displot(data=fmri, x="signal", col="region", hue = 'event', kde=True, alpha␣
    ↪= 0.4)
```

[51]:
```
<seaborn.axisgrid.FacetGrid at 0x7f962beff670>
```

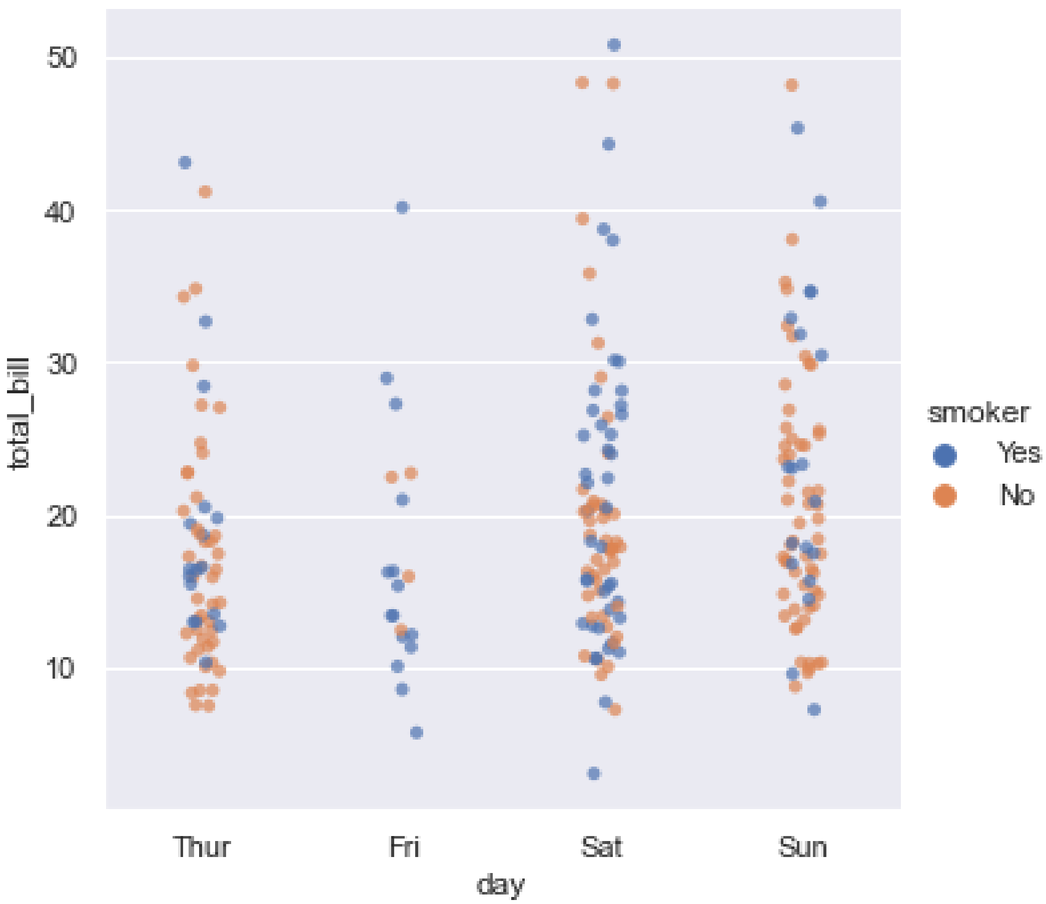


1.2.3 Categorical plots

Let's try playing with `catplot()`

[47]:
```
sns.catplot(data=tips, x="day", y="total_bill", hue="smoker", alpha = 0.7)
```

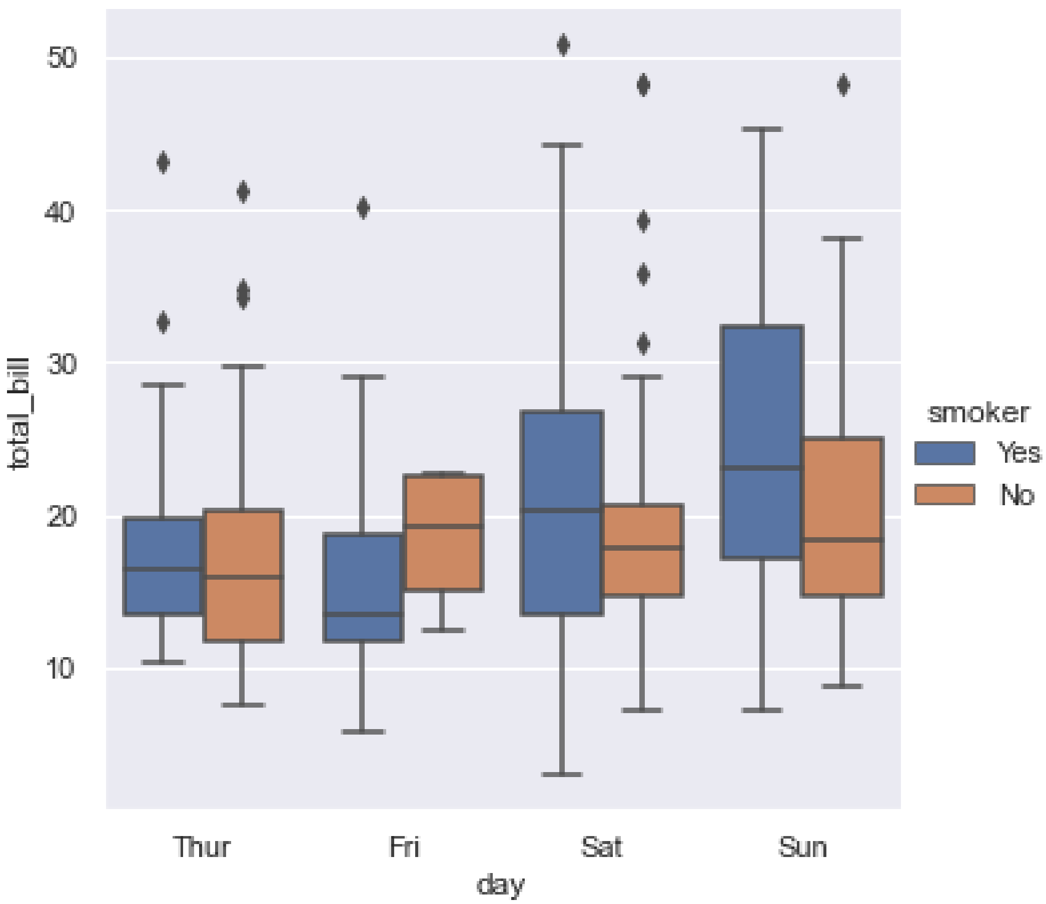
[47]:
```
<seaborn.axisgrid.FacetGrid at 0x7f962afaa4f0>
```



A `stripplot` is the default axes-level plot for `catplot()` (and notice that the default axes-level plots are the first ones listed under their corresponding figure-level counterparts. But we can have it call `boxplot()` for us by telling it that we want `kind="box"`.

[52]:
```
sns.catplot(data=tips, kind="box", x="day", y="total_bill", hue="smoker")
```

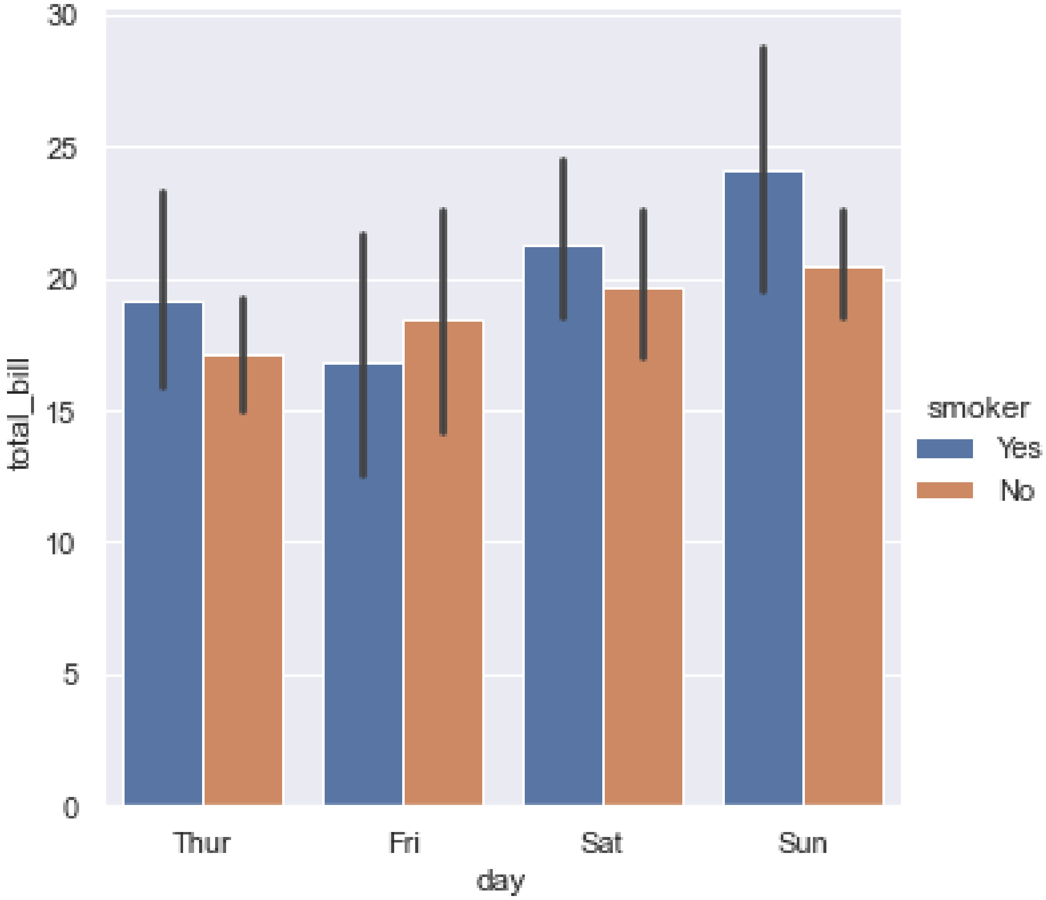
[52]: `<seaborn.axisgrid.FacetGrid at 0x7f962b7eea90>`



If we request a bar plot, notice that seaborn computes and plots the means and also displays confidence intervals – another indication that seaborn was written by a scientist, not a programmer.

[53]:
```
sns.catplot(data=tips, kind="bar", x="day", y="total_bill", hue="smoker")
```

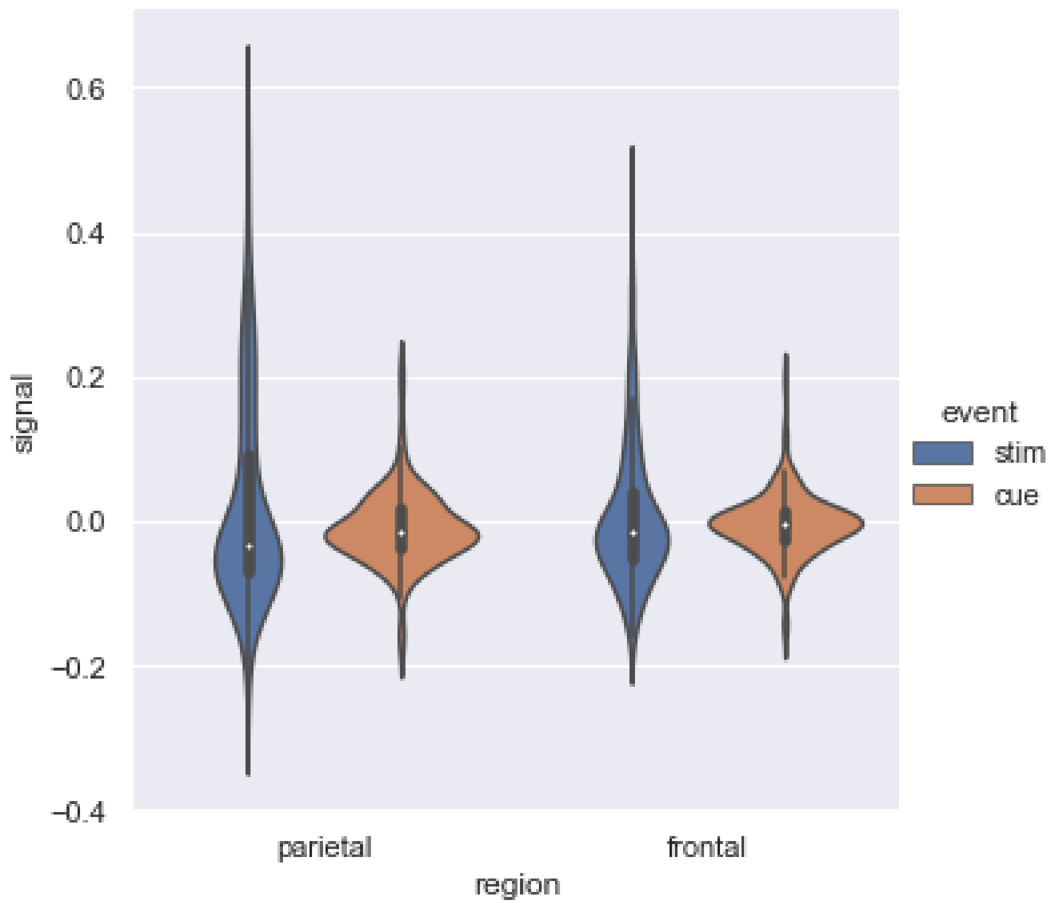
[53]:
```
<seaborn.axisgrid.FacetGrid at 0x7f9624bdfe80>
```



Make a violin plot showing signal (y axis) broken out by brain region and event type.

[61]:
```
sns.catplot(data=fmri, kind="violin", x="region", y="signal", hue="event")
```

[61]: <seaborn.axisgrid.FacetGrid at 0x7f962d06b100>



1.2.4 Useful built in plots

Let's remake a plot similar to one we've made before.

First, we'll import numpy and pandas so we can make and store some data.

[62]:
```
import numpy as np
import pandas as pd
```

Then make some data like we did when reviewing matplotlib, but we'll convert to a pandas `DataFrame` at the end.

[63]:
```
### make some data to play with
my_means = [0, 0]
my_cov = [[2, -1.9], [-1.9, 3]]
my_n = 5000

my_rng = np.random.default_rng(42)
```

```
x, y = my_rng.multivariate_normal(my_means, my_cov, my_n).T

df = pd.DataFrame(dict(x=x, y=y))
```

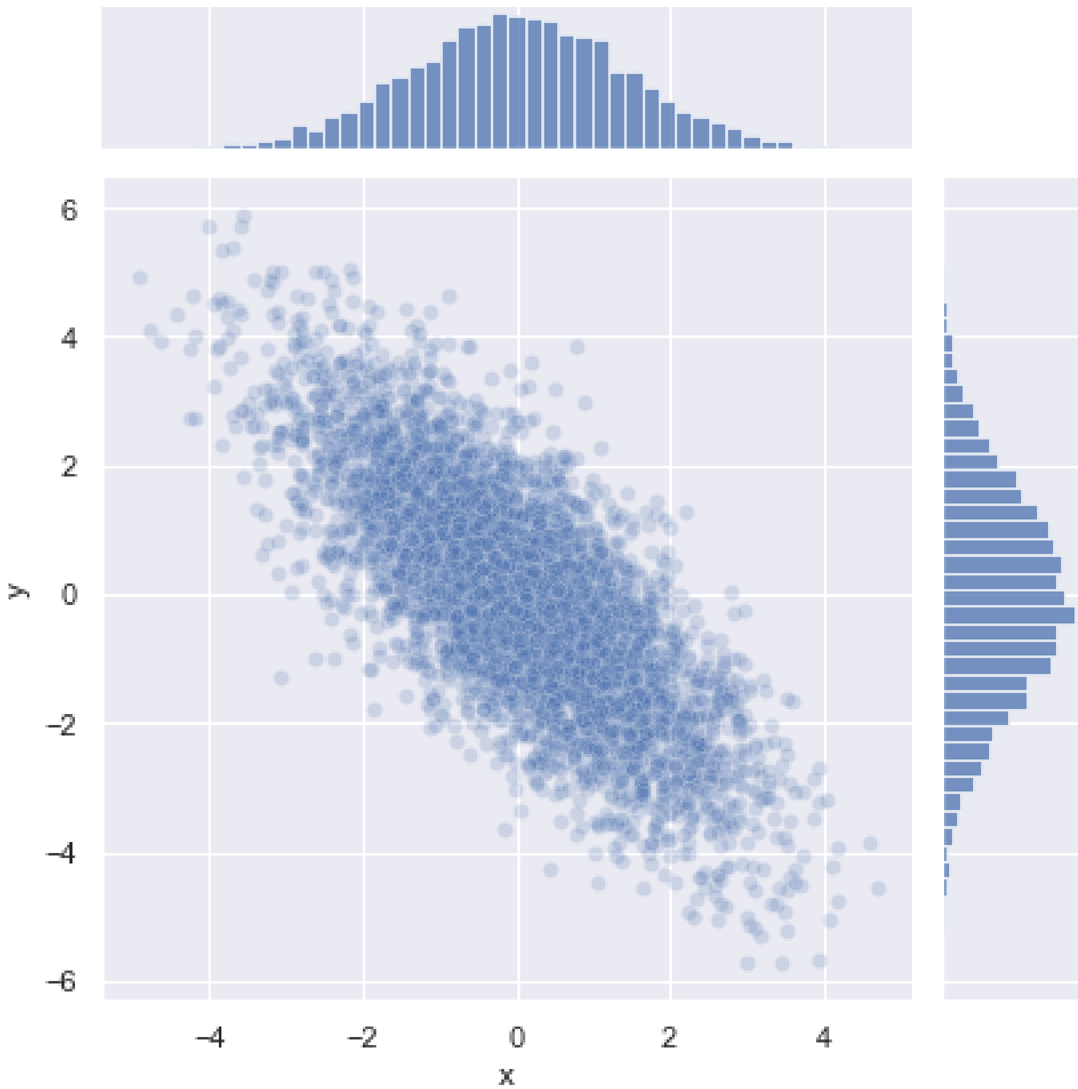
[65]:
```
df
```

[65]:
```
              x         y
0     -0.995728  0.045372
1     -0.423101  1.675997
2      1.762372 -3.846656
3     -0.348170  0.072671
4     -0.472629 -0.409169
…           …         …
4995 -1.612883 -0.109418
4996  1.085352 -0.573411
4997  1.747769 -0.325720
4998  0.130903 -0.067648
4999 -1.556930  1.800533

[5000 rows x 2 columns]
```

And then we'll call the seaborn function `jointplot()` to make a "joint distribution plot".

[64]:
```
sns.jointplot(data=df, x='x', y='y', alpha = 0.2)
```

[64]:
```
<seaborn.axisgrid.JointGrid at 0x7f962d2729a0>
```



That was easy!

In order to really flex seaborn's muscles, though, let's load the built-in "penguins" data set.

[66]:
```
penguins = sns.load_dataset("penguins")
```

[67]:
```
penguins
```

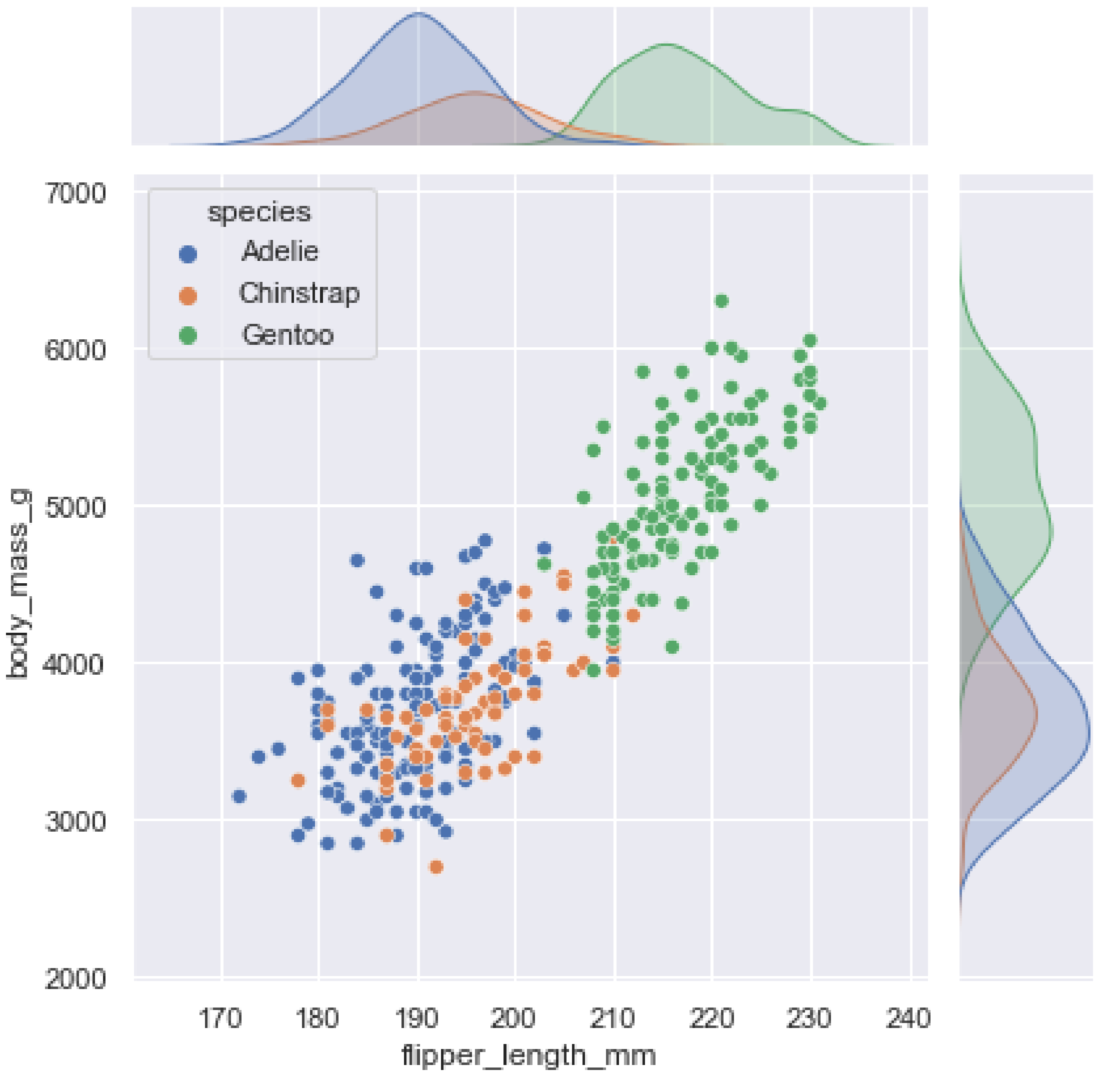
[67]:
```
    species     island  bill_length_mm  bill_depth_mm  flipper_length_mm  \
0    Adelie  Torgersen            39.1           18.7              181.0
1    Adelie  Torgersen            39.5           17.4              186.0
2    Adelie  Torgersen            40.3           18.0              195.0
3    Adelie  Torgersen             NaN            NaN                NaN
4    Adelie  Torgersen            36.7           19.3              193.0
```

```
..      ...        ...          ...          ...            ...
339  Gentoo    Biscoe          NaN          NaN              NaN
340  Gentoo    Biscoe         46.8         14.3            215.0
341  Gentoo    Biscoe         50.4         15.7            222.0
342  Gentoo    Biscoe         45.2         14.8            212.0
343  Gentoo    Biscoe         49.9         16.1            213.0

     body_mass_g     sex
0         3750.0    Male
1         3800.0  Female
2         3250.0  Female
3            NaN     NaN
4         3450.0  Female
..           ...     ...
339          NaN     NaN
340       4850.0  Female
341       5750.0    Male
342       5200.0  Female
343       5400.0    Male

[344 rows x 7 columns]
```

So it looks like we have four measurements taken on penguins of different species, etc. Let's make a joint plot of some of these data.

[68]:
```
sns.jointplot(data=penguins, x="flipper_length_mm", y="body_mass_g",␣
↪hue="species")
```

[68]:
```
<seaborn.axisgrid.JointGrid at 0x7f962d644850>
```

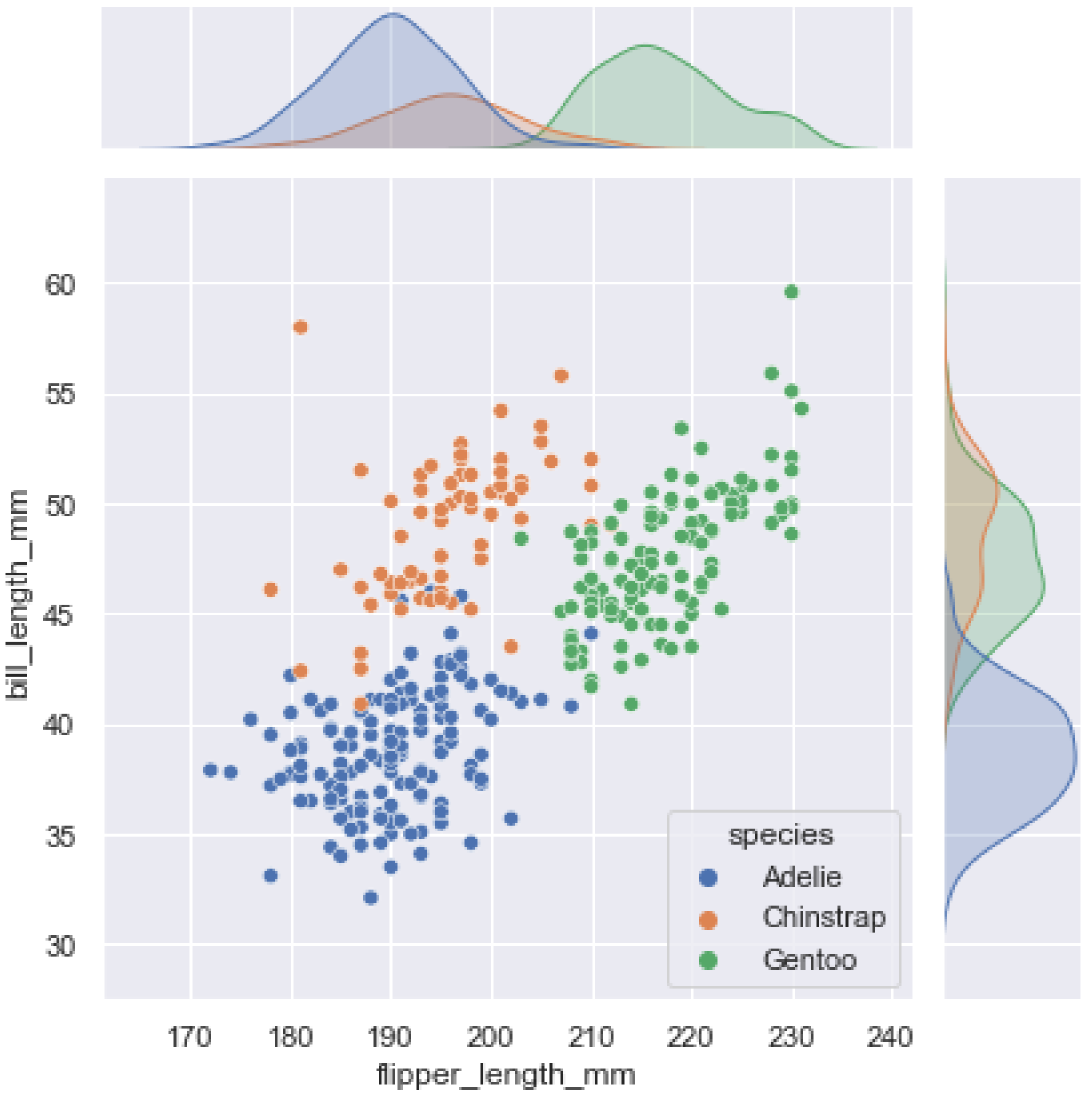


Ah, that's nice! We can see that body weight and flipper length are about as correlated as any biological measurement can get. Moreover, Gentoos are clearly biggest of the three types of dinosaur.

Re-make the above plot using bill length instead of body mass on the y axis.

[69]:
```
sns.jointplot(data=penguins, x="flipper_length_mm", y="bill_length_mm",␣
↪hue="species")
```

[69]:
```
<seaborn.axisgrid.JointGrid at 0x7f962d878af0>
```



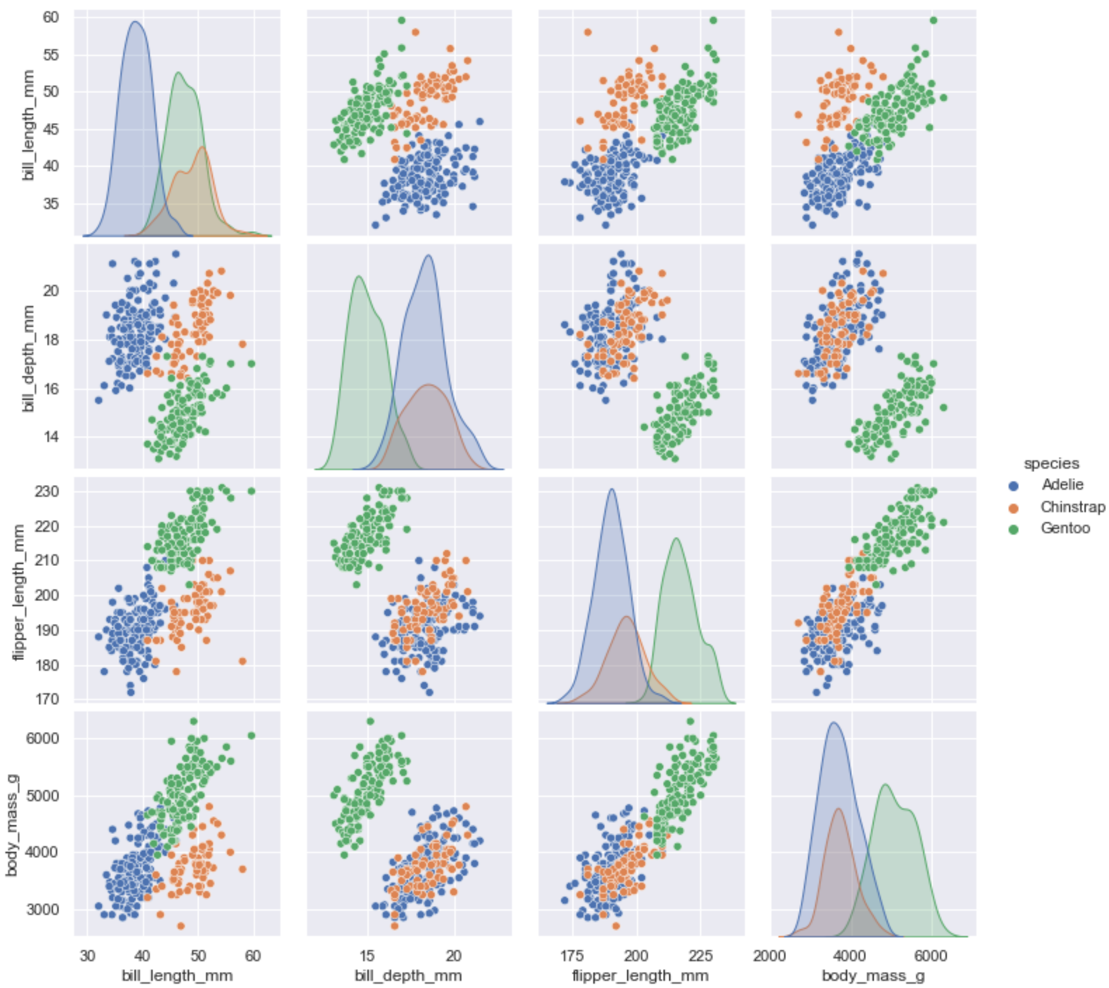
What does this plot tell you? Could you discriminate amongst the species using either one of the variables alone? How about using both variables?

- `This plot tells me that each species have somewhat distinguish flipper length and bill length because we can see that each species have their own range and ratio. We can also distinguish each species using only either flipper length or bill length.`

Make a `pairplot()` of the penguin data, using species to set the color.

[70]:
```
sns.pairplot(data=penguins, hue="species")
```

[70]: `<seaborn.axisgrid.PairGrid at 0x7f962dc85b20>`



What combinations of variables would allow you to do a decent job of categorizing the species? Could you do it just based on bill measurements?

- `I think that Bill Length x Bill Depth would be an amazing category to use in this sample because each species have their own range of valuable.`

1.2.5 matplotlib style customization

The axes-level functions return a matplotlib axes object.

[71]:
```
ax = sns.scatterplot(data=df, x='x', y='y', alpha = 0.2)
```

We can verify this by checking its type.

[72]:
```
type(ax)
```

[72]: `matplotlib.axes._subplots.AxesSubplot`

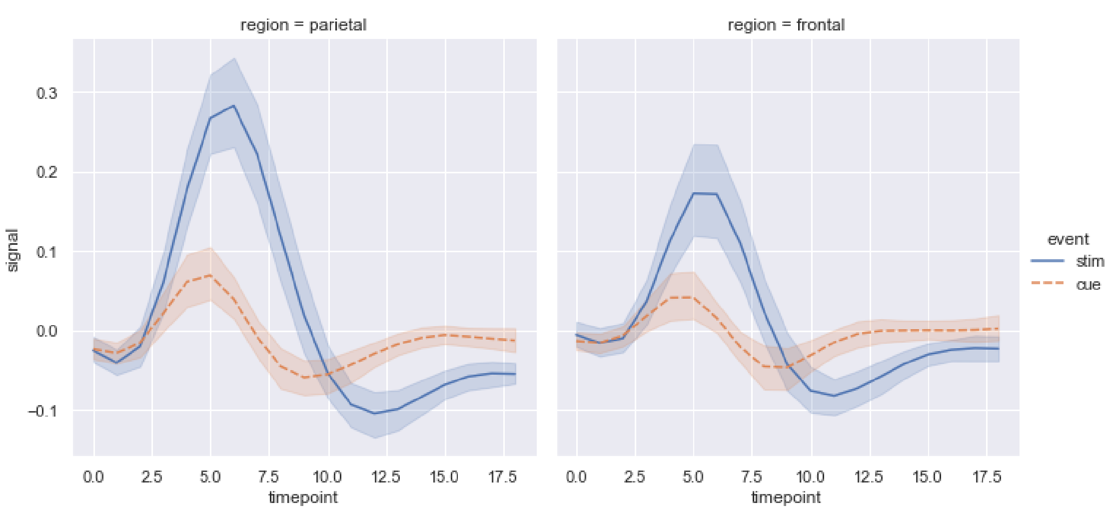
You can use this to customize your plot using the matplotlib axes methods, like `set_label()`, etc., just as though you had created the plot directly in matplotlib.

[73]:
```
ax = sns.scatterplot(data=df, x='x', y='y', alpha = 0.2)
ax.set_xlabel("Ex")
ax.set_ylabel("Why?")
ax.set_title('Why vs. Ex')
```

[73]: `Text(0.5, 1.0, 'Why vs. Ex')`

If we use one of the higher level plotting functions, we get back something called a FacetGrid that seaborn has created.

[74]:
```
fg = sns.relplot(
    data=fmri, kind="line",
    x="timepoint", y="signal", col="region",
    hue="event", style="event",
)
```



[75]:
```
type(fg)
```

[75]:
```
seaborn.axisgrid.FacetGrid
```

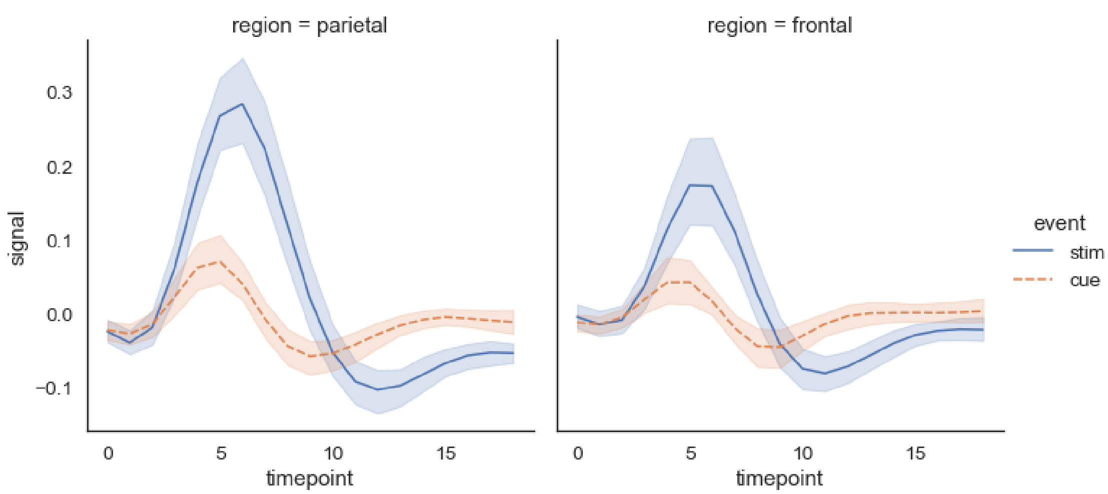
If we type "fg." and hit <TAB>, we can see that it has a lot of methods we can put to work for us.

[ ]:
```
fg.
```

It also gives us access to the matplotlib figure via `fg.fig`, so we can set figure level properties.

In the cell below, add a line at the bottom to set the figure's background to a different color.

[99]:
```
fg = sns.relplot(
    data=fmri, kind="line",
    x="timepoint", y="signal", col="region",
    hue="event", style="event",
)

# set background to a different color
sns.set_theme(style="white", font_scale=1.25)
```

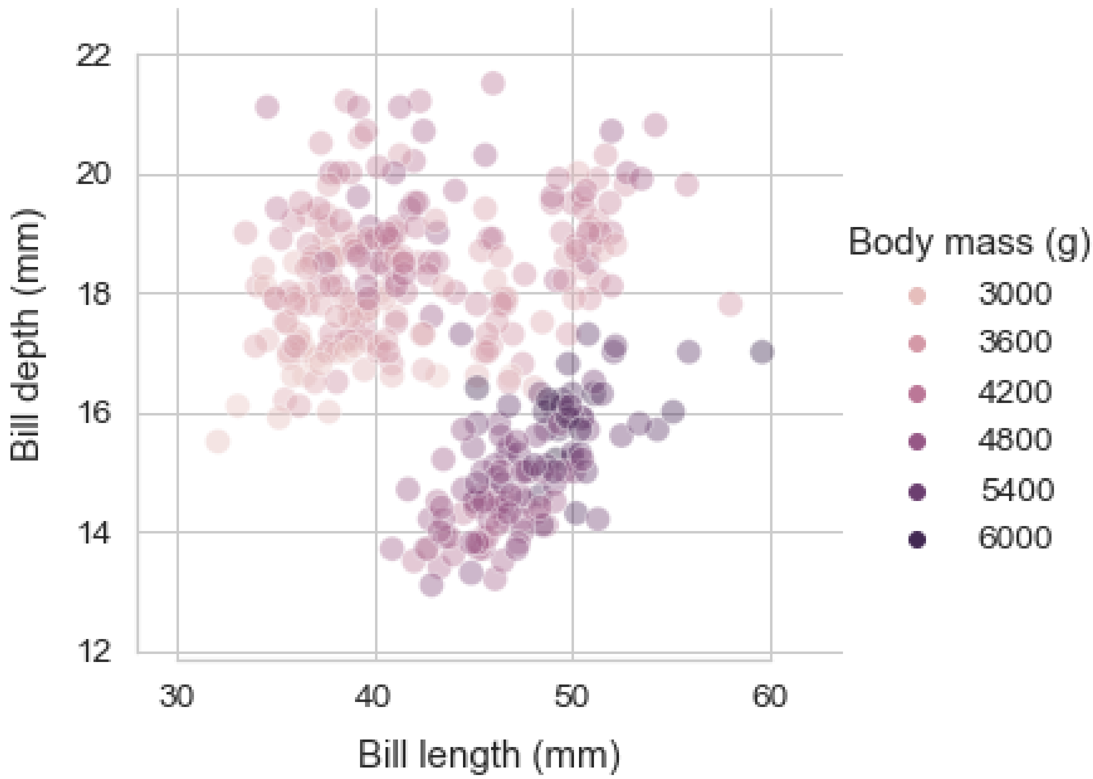


Here's a fancy example.

[93]:
```
sns.set_theme(style="whitegrid", font_scale=1.25)
fg = sns.relplot(
    data=penguins,
    x="bill_length_mm", y="bill_depth_mm", hue="body_mass_g",
```

```
    marker="o", s=100, alpha = 0.4
)
fg.set_axis_labels("Bill length (mm)", "Bill depth (mm)", labelpad=10)
fg.legend.set_title("Body mass (g)")
fg.figure.set_size_inches(6.5, 4.5)
fg.ax.margins(.15)
fg.despine(trim=True)
```

[93]: <seaborn.axisgrid.FacetGrid at 0x7f9616e45ac0>



1.3 Conclusion

Seaborn is a very powerful plotting package. It gives us both high-level easy plotting combined with the ability to do low-level customization if we wish. For custom figure layouts, matplotlib might be required. But for many plotting tasks, seaborn makes nice looking plots and can be a huge time saver!