Plotting with categorical data

We previously (regression.html#regression-tutorial) learned how to use scatterplots and regression model fits to visualize the relationship between two variables and how it changes across levels of additional categorical variables. However, what if one of the main variables you are interested in is categorical? In this case, the scatterplot and regression model approach won't work. There are several options, however, for visualizing such a relationship, which we will discuss in this tutorial.

It's useful to divide seaborn's categorical plots into three groups: those that show each observation at each level of the categorical variable, those that show an abstract representation of each *distribution* of observations, and those that apply a statistical estimation to show a measure of central tendency and confidence interval. The first includes the functions **`swarmplot()`** (../generated/seaborn.swarmplot.html#seaborn.swarmplot) and **`stripplot()`** (../generated/seaborn.stripplot.html#seaborn.stripplot), the second includes **`boxplot()`** (../generated/seaborn.boxplot.html#seaborn.boxplot) and **`violinplot()`** (../generated/seaborn.violinplot.html#seaborn.violinplot), and the third includes **`barplot()`** (../generated/seaborn.barplot.html#seaborn.barplot) and **`pointplot()`** (../generated/seaborn.pointplot.html#seaborn.pointplot). These functions all share a basic API for how they accept data, although each has specific parameters that control the particulars of the visualization that is applied to that data.

Much like the relationship between **`regplot()`** (../generated/seaborn.regplot.html#seaborn.regplot) and **`lmplot()`** (../generated/seaborn.lmplot.html#seaborn.lmplot), in seaborn there are both relatively low-level and relatively high-level approaches for making categorical plots. The functions named above are all low-level in that they plot onto a specific matplotlib axes. There is also the higher-level **`factorplot()`** (../generated/seaborn.factorplot.html#seaborn.factorplot), which combines these functions with a **`FacetGrid`** (../generated/seaborn.FacetGrid.html#seaborn.FacetGrid) to apply a categorical plot across a grid of figure panels.

It is easiest and best to invoke these functions with a DataFrame that is in "tidy" (http://vita.had.co.nz/papers/tidy-data.pdf) format, although the lower-level functions also accept wide-form DataFrames or simple vectors of observations. See below for examples.

```
%matplotlib inline
```

```
import numpy as np
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
```

```
import seaborn as sns
sns.set(style="whitegrid", color_codes=True)
```

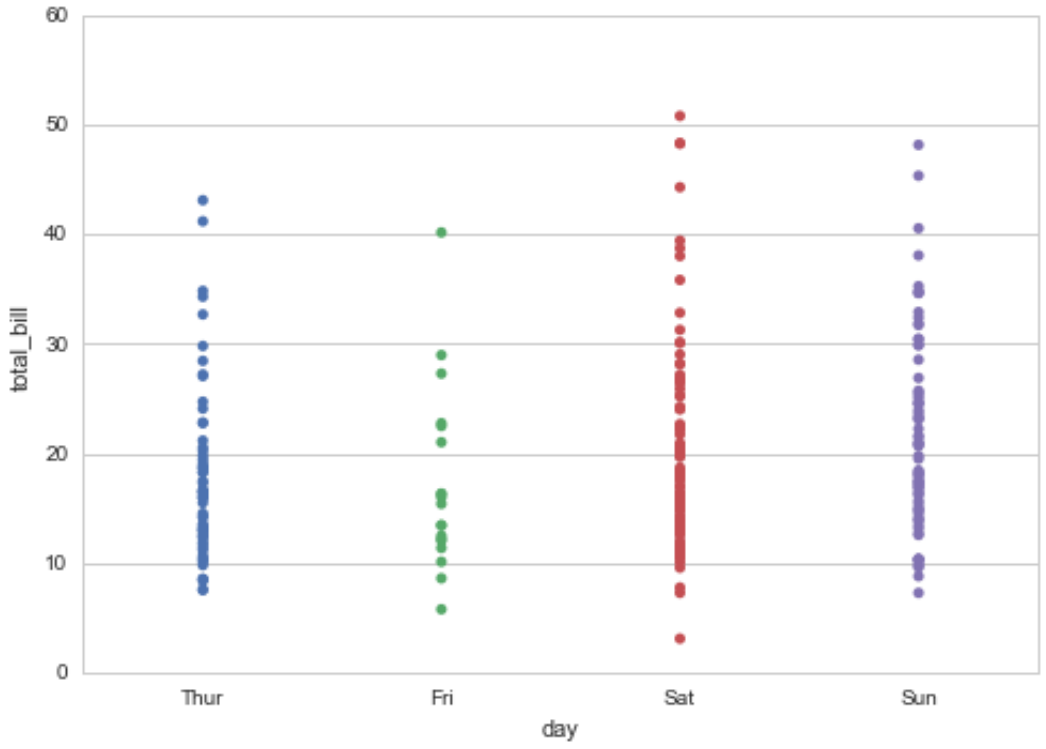
```
np.random.seed(sum(map(ord, "categorical")))
```

```
titanic = sns.load_dataset("titanic")
tips = sns.load_dataset("tips")
iris = sns.load_dataset("iris")
```

Categorical scatterplots

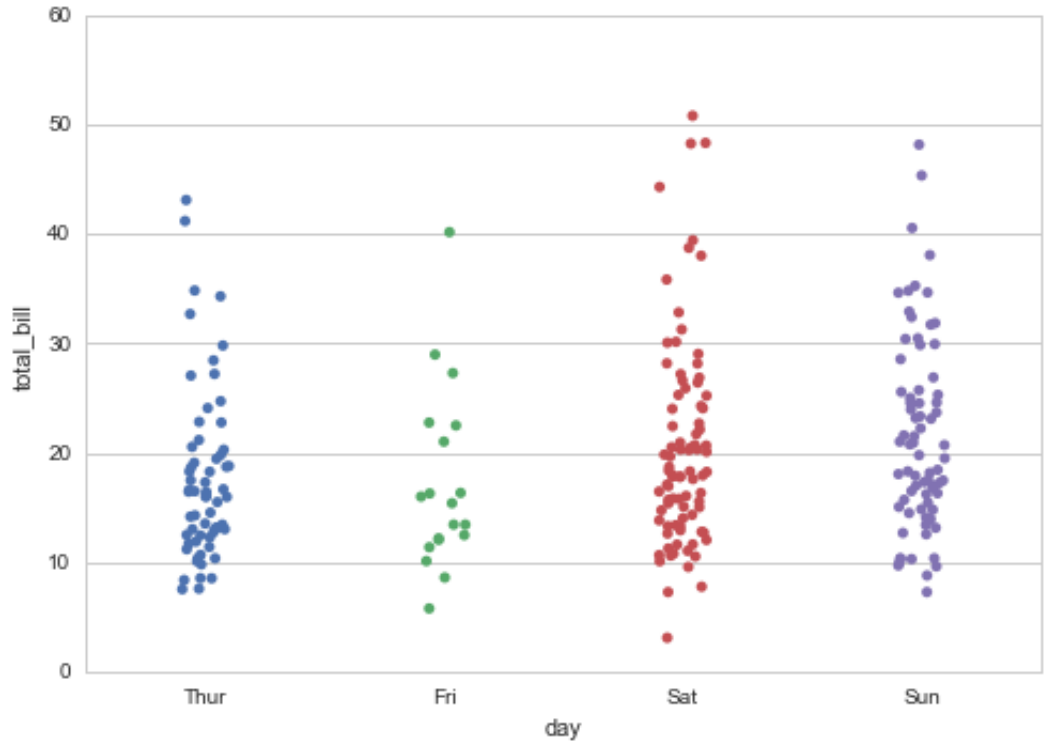
A simple way to show the the values of some quantitative variable across the levels of a categorical variable uses `stripplot()` (../generated/seaborn.stripplot.html#seaborn.stripplot), which generalizes a scatterplot to the case where one of the variables is categorical:

```
sns.stripplot(x="day", y="total_bill", data=tips);
```



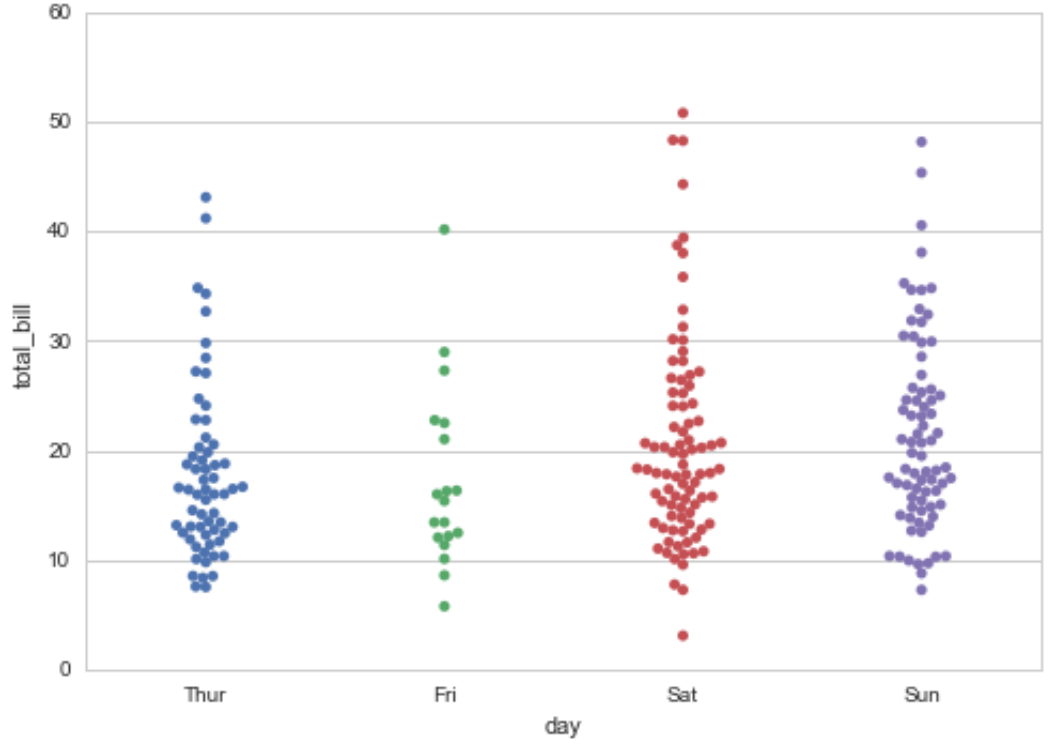
In a strip plot, the scatterplot points will usually overlap. This makes it difficult to see the full distribution of data. One easy solution is to adjust the positions (only along the categorical axis) using some random “jitter”:

```
sns.stripplot(x="day", y="total_bill", data=tips, jitter=True);
```



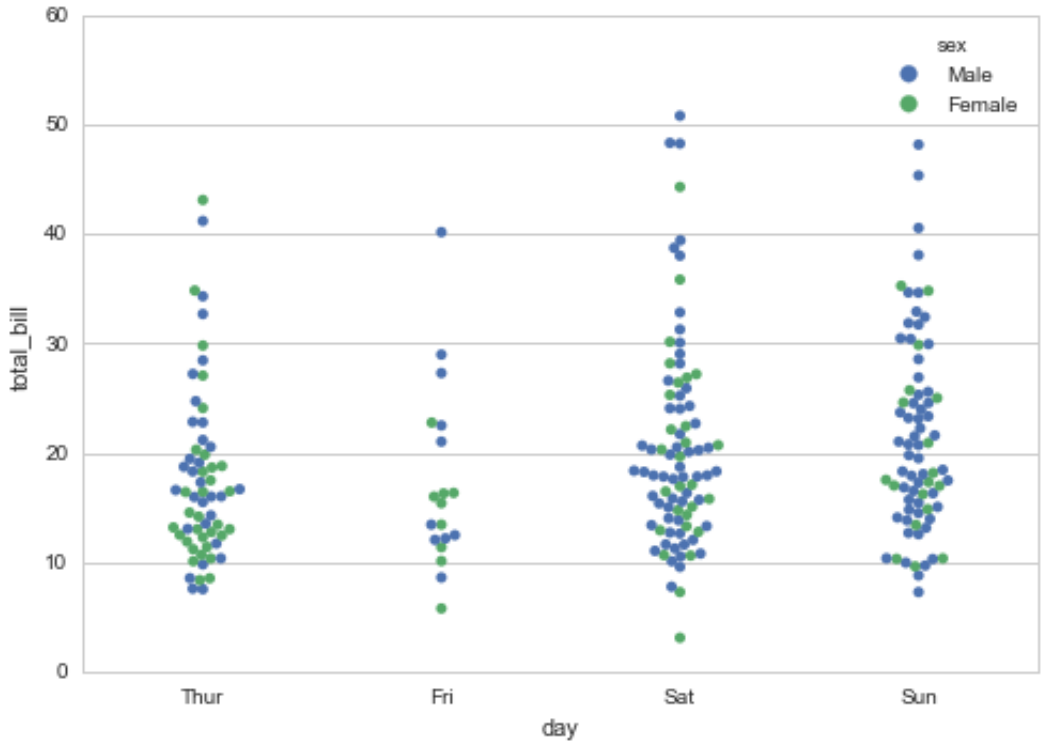
A different approach would be to use the function **`swarmplot()`** (../generated/seaborn.swarmplot.html#seaborn.swarmplot), which positions each scatterplot point on the categorical axis with an algorithm that avoids overlapping points:

```
sns.swarmplot(x="day", y="total_bill", data=tips);
```



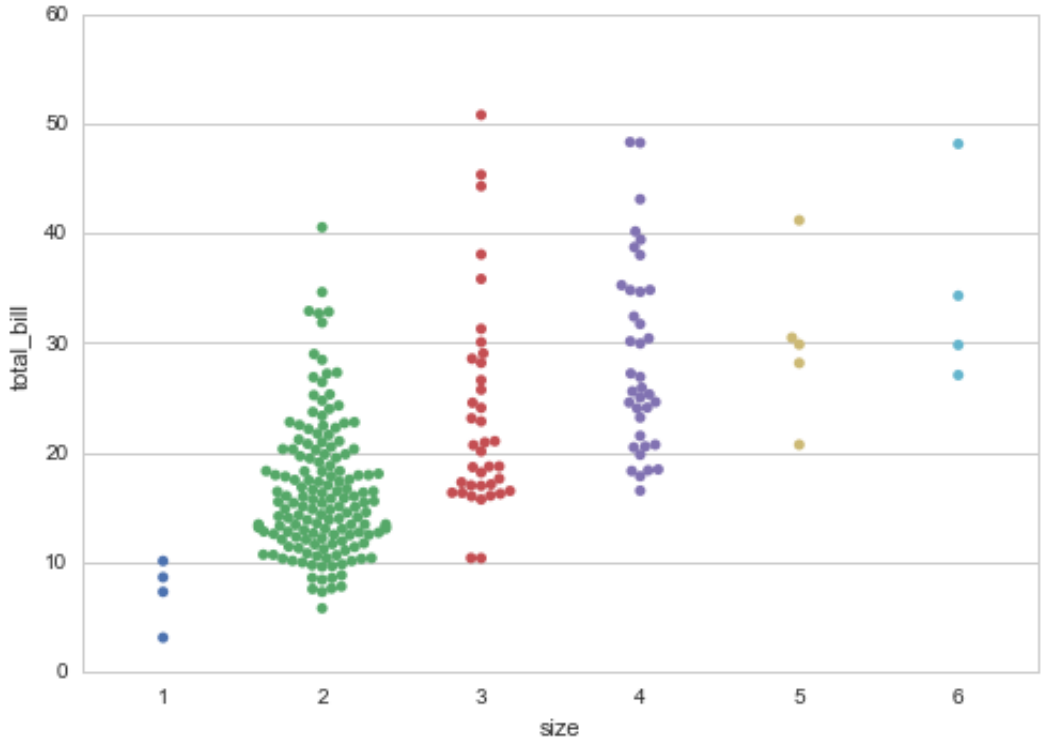
It's also possible to add a nested categorical variable with the `hue` parameter. Above the color and position on the categorical axis are redundant, but now each provides information about one of the two variables:

```
sns.swarmplot(x="day", y="total_bill", hue="sex", data=tips);
```



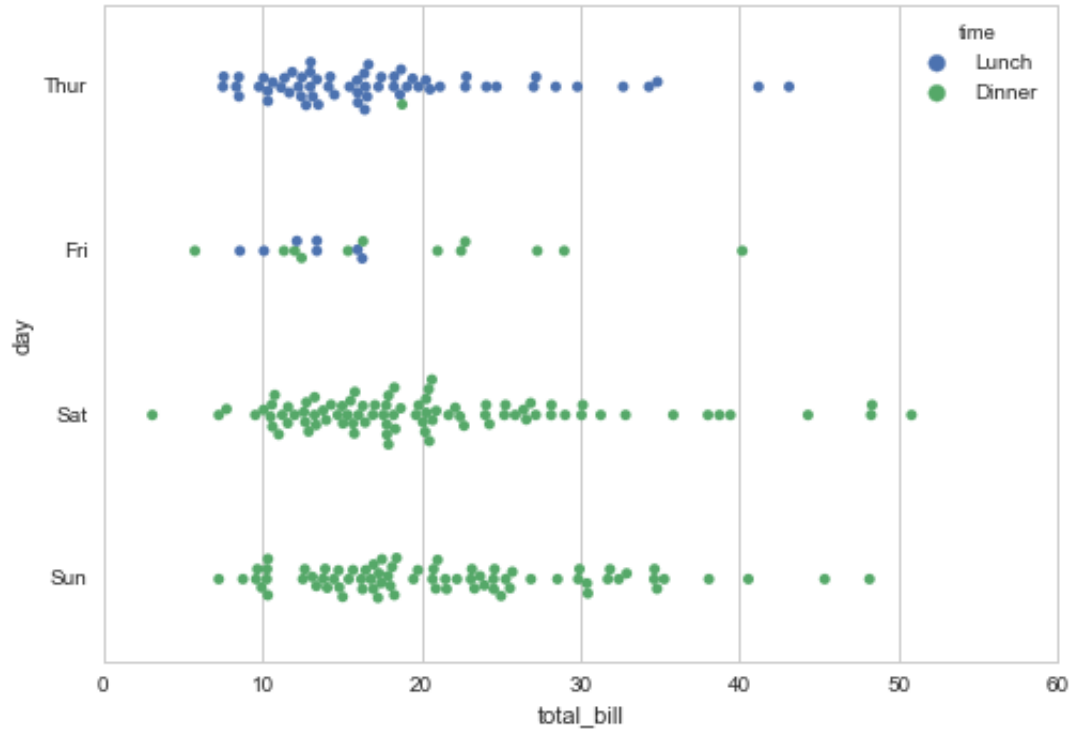
In general, the seaborn categorical plotting functions try to infer the order of categories from the data. If your data have a pandas `Categorical` datatype, then the default order of the categories can be set there. For other datatypes, string-typed categories will be plotted in the order they appear in the DataFrame, but categories that look numerical will be sorted:

```
sns.swarmplot(x="size", y="total_bill", data=tips);
```



With these plots, it's often helpful to put the categorical variable on the vertical axis (this is particularly useful when the category names are relatively long or there are many categories). You can force an orientation using the `orient` keyword, but usually plot orientation can be inferred from the datatypes of the variables passed to `x` and/or `y`:

```
sns.swarmplot(x="total_bill", y="day", hue="time", data=tips);
```



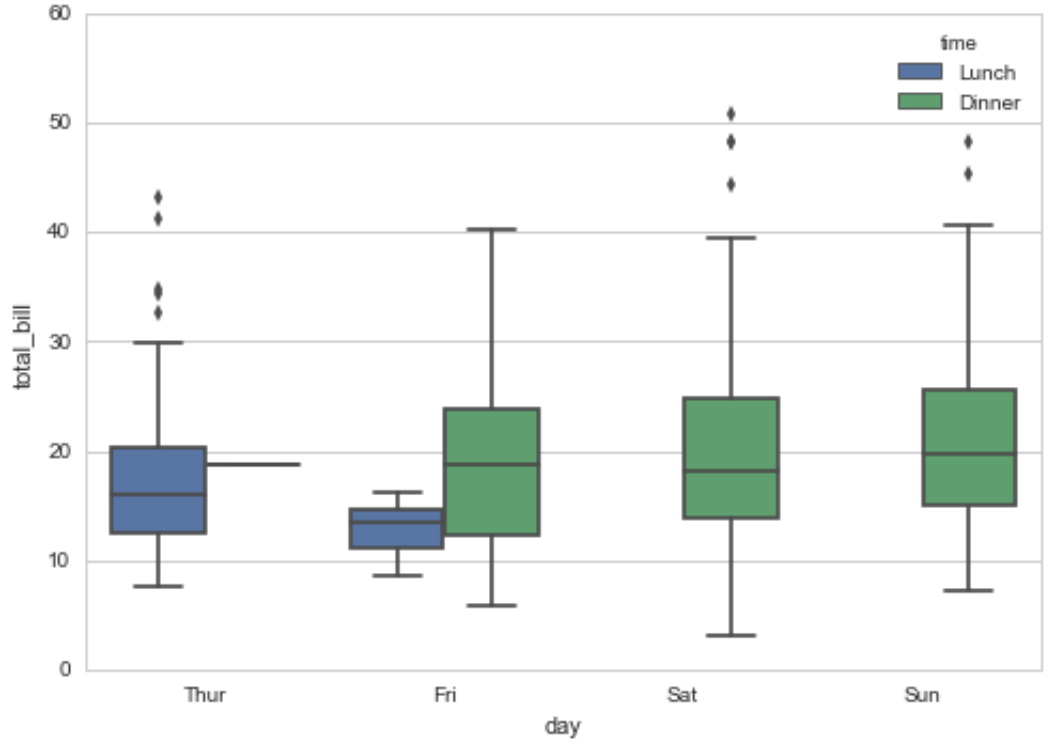
Distributions of observations within categories

At a certain point, the categorical scatterplot approach becomes limited in the information it can provide about the distribution of values within each category. There are several ways to summarize this information in ways that facilitate easy comparisons across the category levels. These generalize some of the approaches we discussed in the chapter (distributions.html#distribution-tutorial) to the case where we want to quickly compare across several distributions.

Boxplots

The first is the familiar **`boxplot()`** (../generated/seaborn.boxplot.html#seaborn.boxplot). This kind of plot shows the three quartile values of the distribution along with extreme values. The "whiskers" extend to points that lie within 1.5 IQRs of the lower and upper quartile, and then observations that fall outside this range are displayed independently. Importantly, this means that each value in the boxplot corresponds to an actual observation in the data:

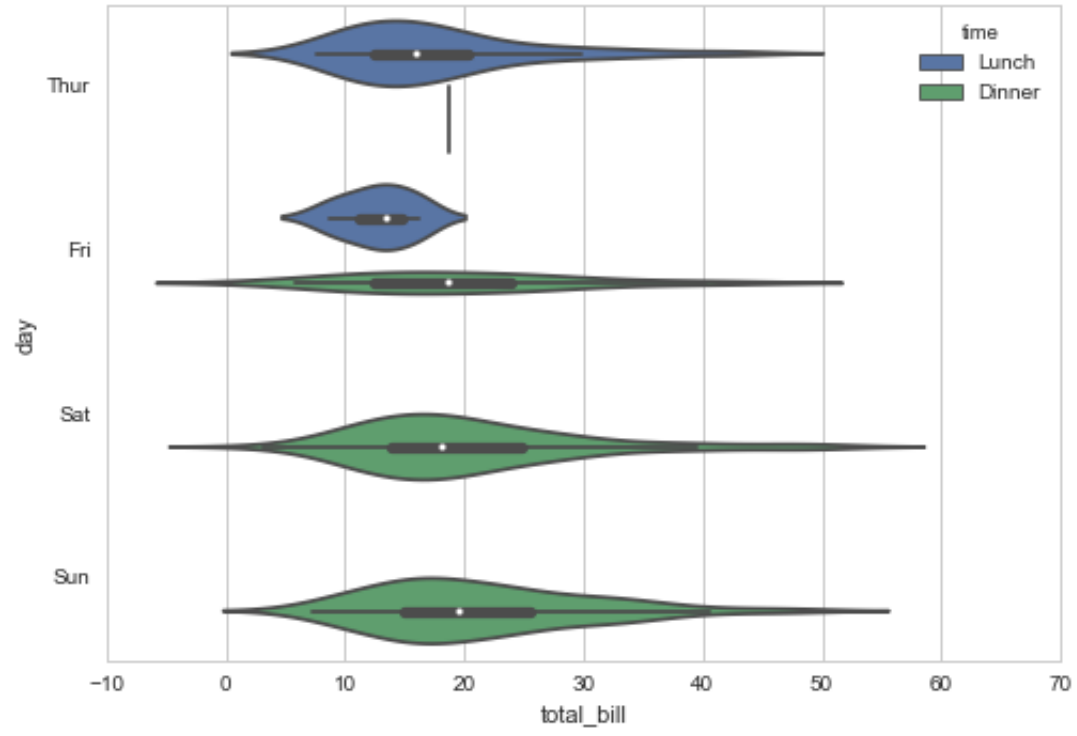
```
sns.boxplot(x="day", y="total_bill", hue="time", data=tips);
```



Violinplots

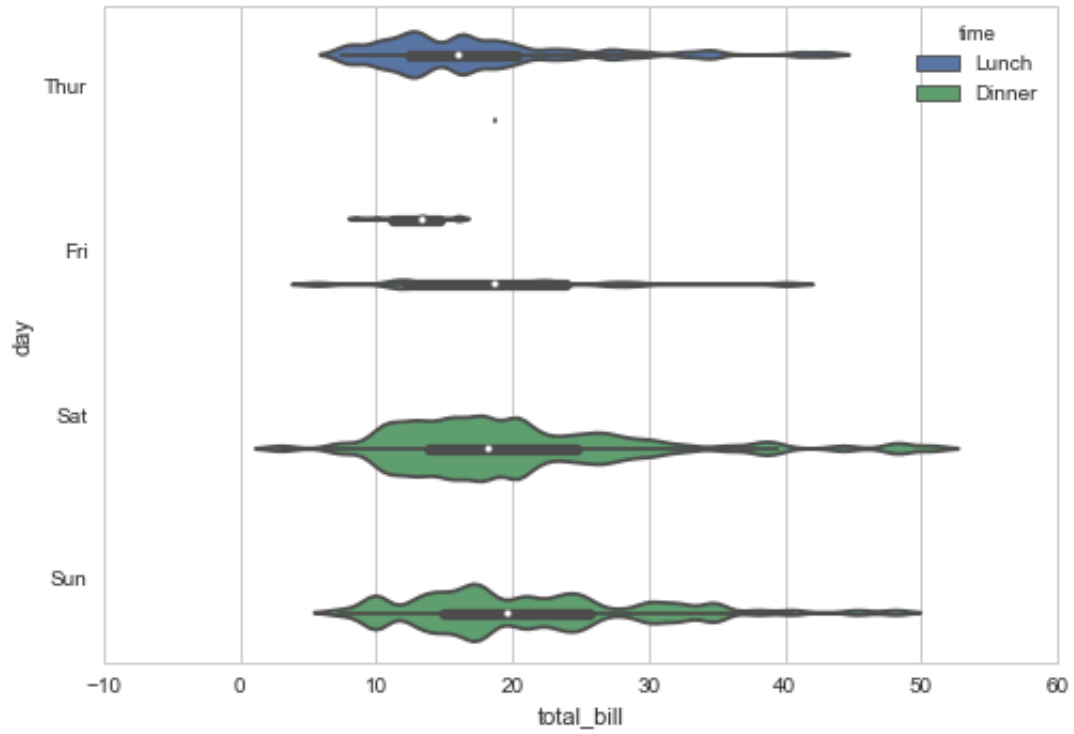
A different approach is a **`violinplot()`** (../generated/seaborn.violinplot.html#seaborn.violinplot), which combines a boxplot with the kernel density estimation procedure described in the distributions (distributions.html#distribution-tutorial) tutorial:

```
sns.violinplot(x="total_bill", y="day", hue="time", data=tips);
```



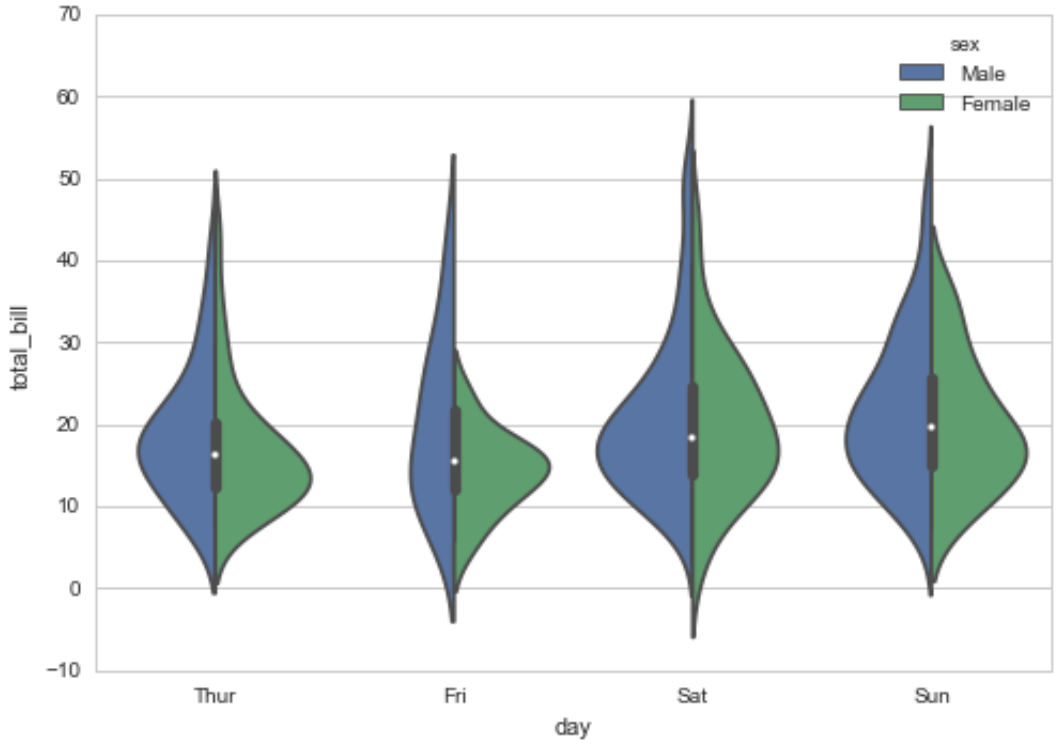
This approach uses the kernel density estimate to provide a better description of the distribution of values. Additionally, the quartile and whikser values from the boxplot are shown inside the violin. Because the violinplot uses a KDE, there are some other parameters that may need tweaking, adding some complexity relative to the straightforward boxplot:

```
sns.violinplot(x="total_bill", y="day", hue="time", data=tips,
               bw=.1, scale="count", scale_hue=False);
```



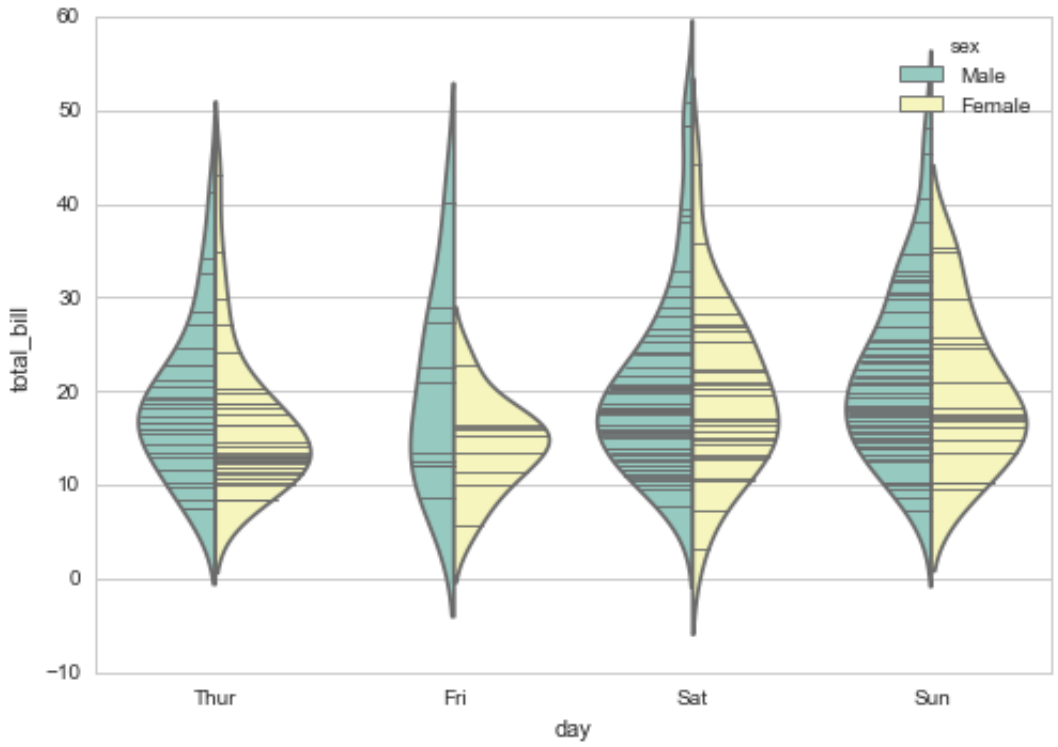
It's also possible to "split" the violins when the hue parameter has only two levels, which can allow for a more efficient use of space:

```
sns.violinplot(x="day", y="total_bill", hue="sex", data=tips, split=True);
```



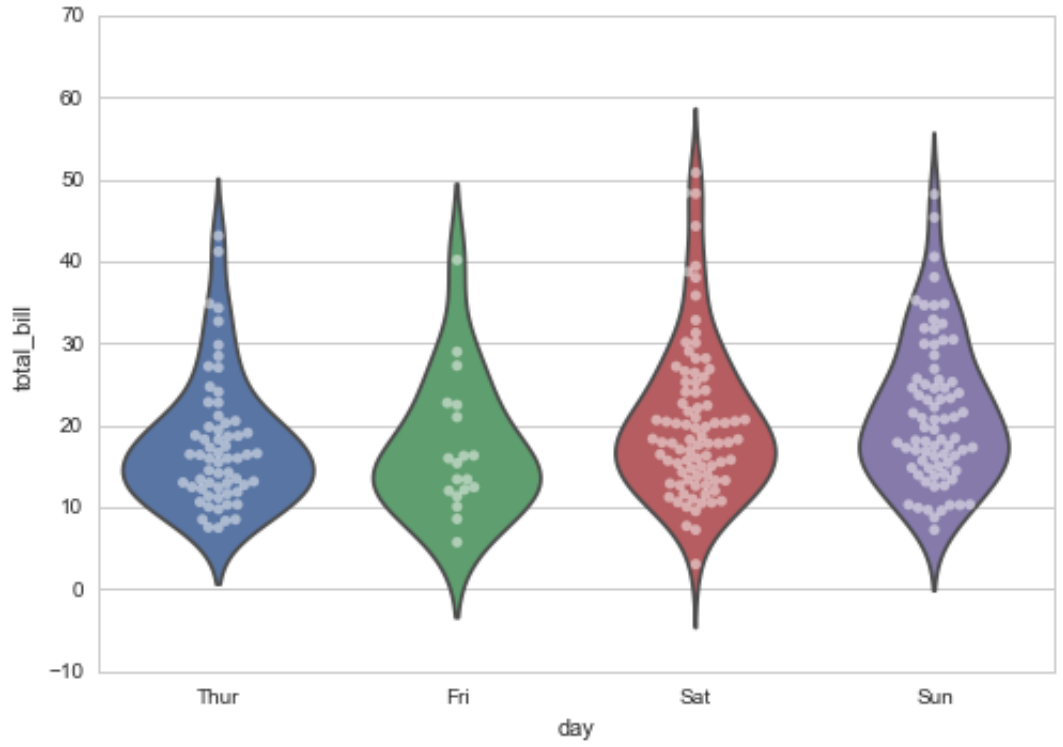
Finally, there are several options for the plot that is drawn on the interior of the violins, including ways to show each individual observation instead of the summary boxplot values:

```
sns.violinplot(x="day", y="total_bill", hue="sex", data=tips,
               split=True, inner="stick", palette="Set3");
```



It can also be useful to combine **`swarmplot()`** (../generated/seaborn.swarmplot.html#seaborn.swarmplot) or **`swarmplot()`** (../generated/seaborn.swarmplot.html#seaborn.swarmplot) with **`violinplot()`** (../generated/seaborn.violinplot.html#seaborn.violinplot) or **`boxplot()`** (../generated/seaborn.boxplot.html#seaborn.boxplot) to show each observation along with a summary of the distribution:

```
sns.violinplot(x="day", y="total_bill", data=tips, inner=None)
sns.swarmplot(x="day", y="total_bill", data=tips, color="w", alpha=.5);
```



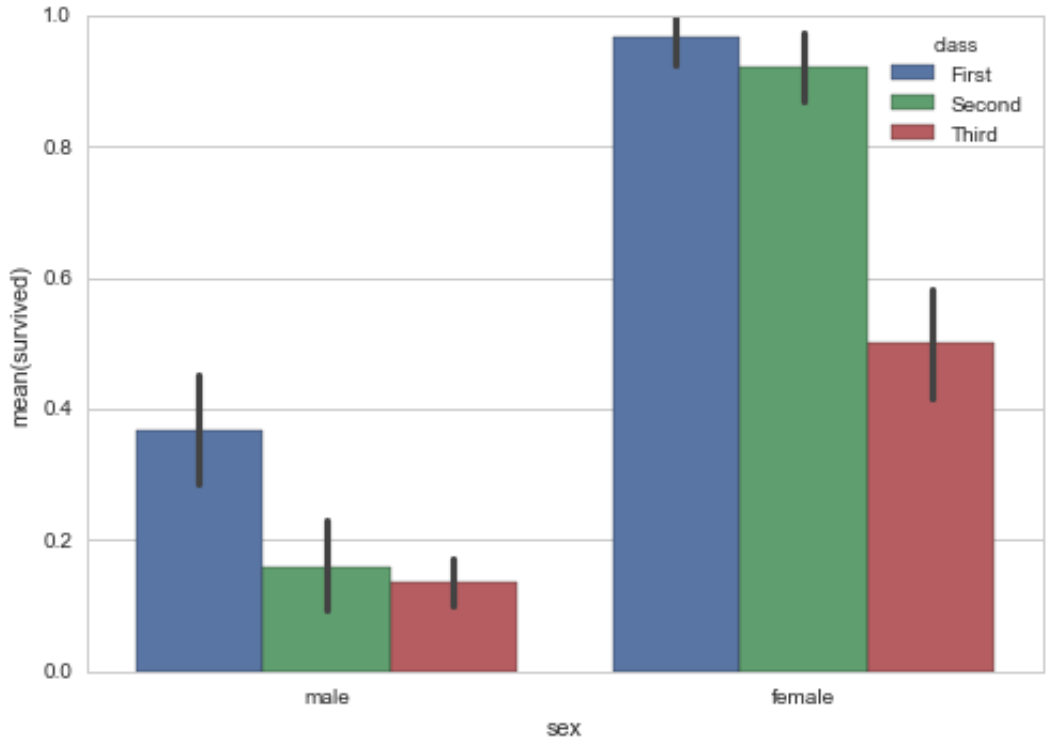
Statistical estimation within categories

Often, rather than showing the distribution within each category, you might want to show the central tendency of the values. Seaborn has two main ways to show this information, but importantly, the basic API for these functions is identical to that for the ones discussed above.

Bar plots

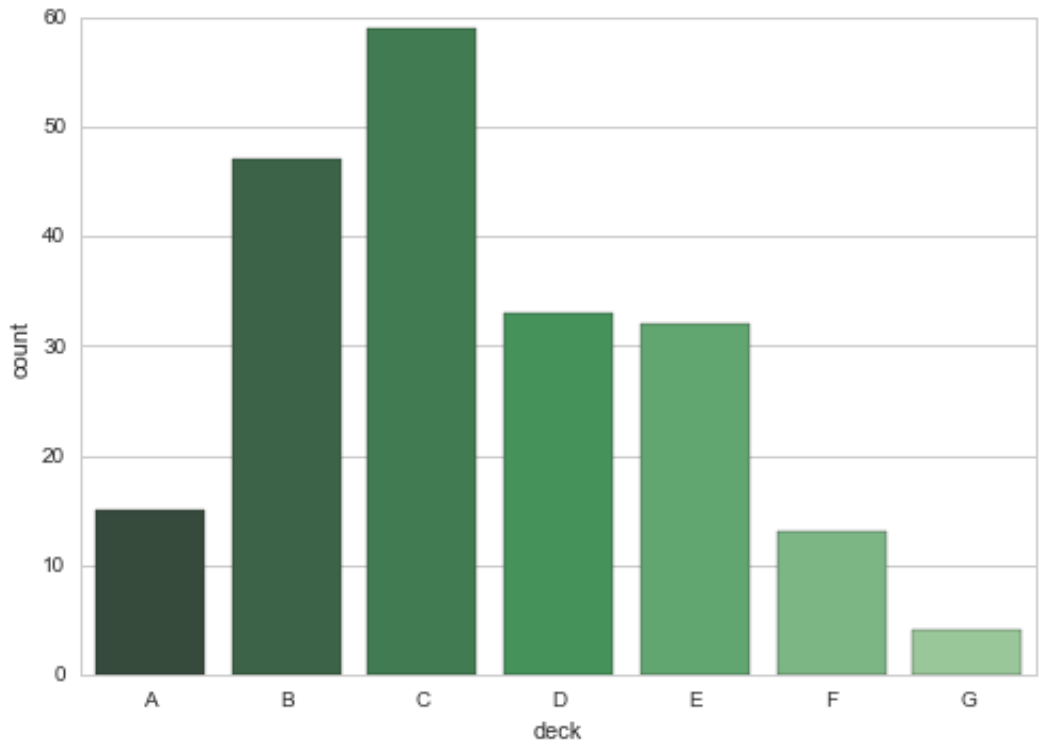
A familiar style of plot that accomplishes this goal is a bar plot. In seaborn, the **`barplot()`** (../generated/seaborn.barplot.html#seaborn.barplot) function operates on a full dataset and shows an arbitrary estimate, using the mean by default. When there are multiple observations in each category, it also uses bootstrapping to compute a confidence interval around the estimate and plots that using error bars:

```
sns.barplot(x="sex", y="survived", hue="class", data=titanic);
```



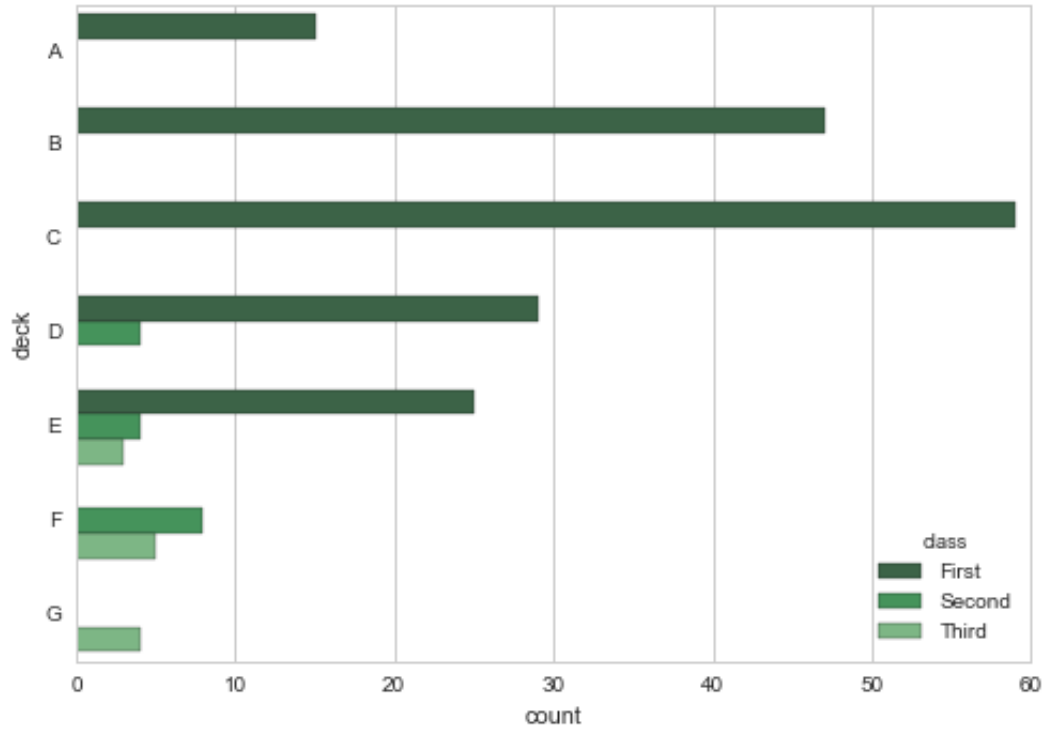
A special case for the bar plot is when you want to show the number of observations in each category rather than computing a statistic for a second variable. This is similar to a histogram over a categorical, rather than quantitative, variable. In seaborn, it's easy to do so with the **`countplot()`** (../generated/seaborn.countplot.html#seaborn.countplot) function:

```
sns.countplot(x="deck", data=titanic, palette="Greens_d");
```



Both **`barplot()`** (../generated/seaborn.barplot.html#seaborn.barplot) and **`countplot()`** (../generated/seaborn.countplot.html#seaborn.countplot) can be invoked with all of the options discussed above, along with others that are demonstrated in the detailed documentation for each function:

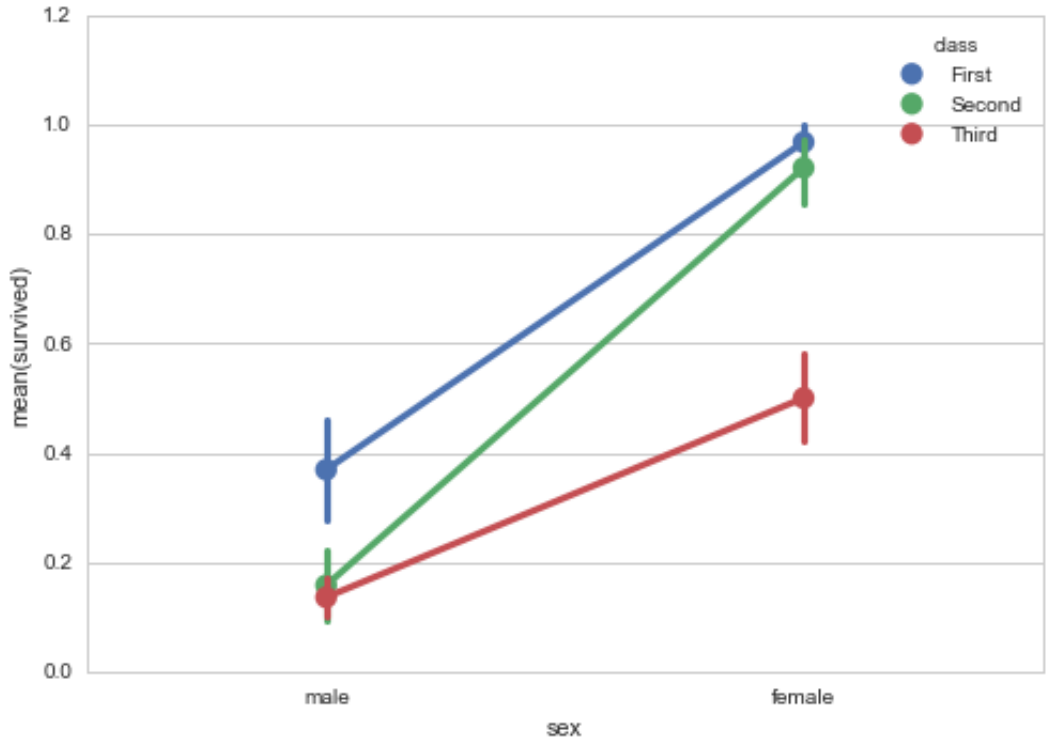
```
sns.countplot(y="deck", hue="class", data=titanic, palette="Greens_d");
```



Point plots

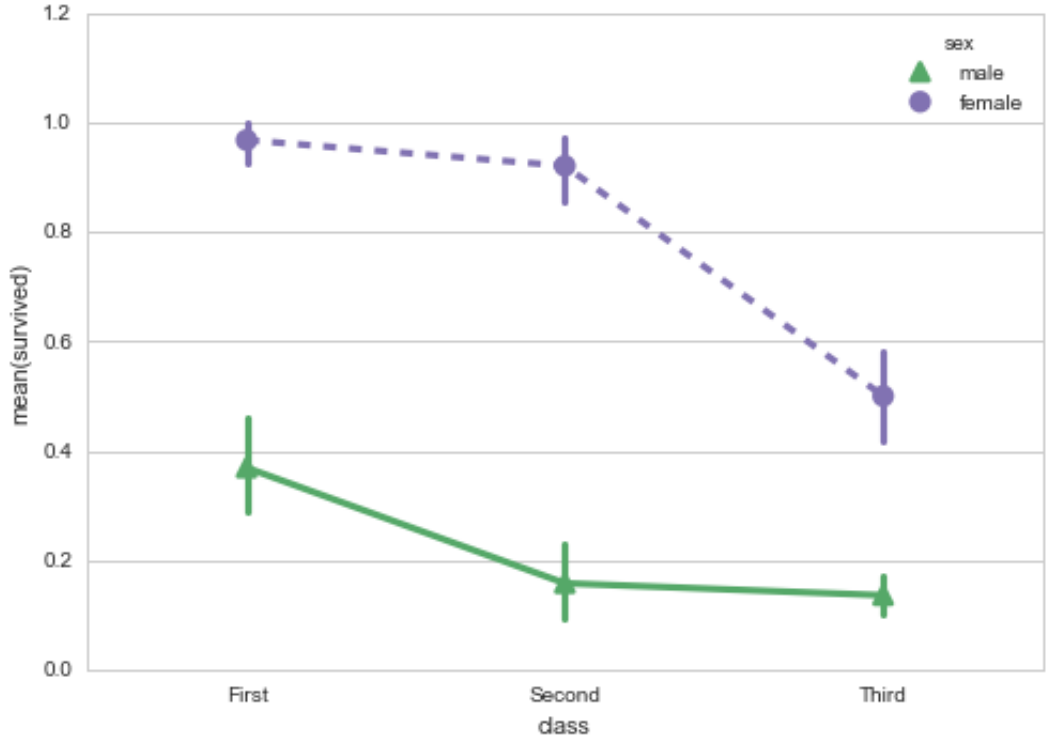
An alternative style for visualizing the same information is offered by the **`pointplot()`** (../generated/seaborn.pointplot.html#seaborn.pointplot) function. This function also encodes the value of the estimate with height on the other axis, but rather than show a full bar it just plots the point estimate and confidence interval. Additionally, pointplot connects points from the same `hue` category. This makes it easy to see how the main relationship is changing as a function of a second variable, because your eyes are quite good at picking up on differences of slopes:

```
sns.pointplot(x="sex", y="survived", hue="class", data=titanic);
```



To make figures that reproduce well in black and white, it can be good to use different markers and line styles for the levels of the `hue` category:

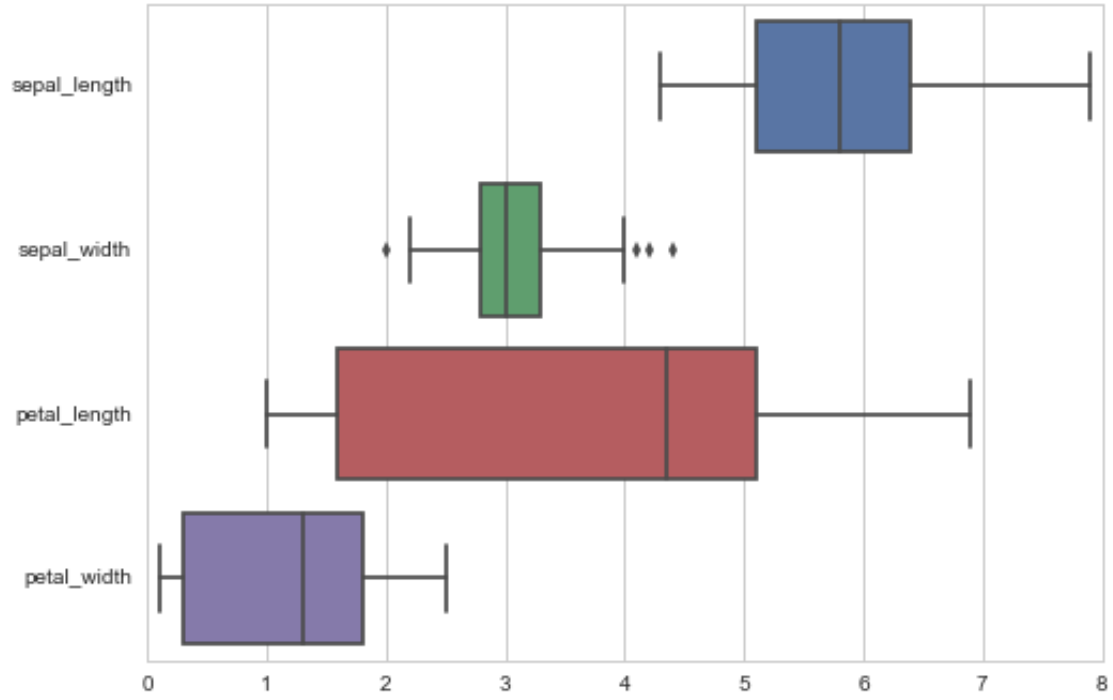
```
sns.pointplot(x="class", y="survived", hue="sex", data=titanic,
              palette={"male": "g", "female": "m"},
              markers=["^", "o"], linestyles=["-", "--"]);
```



Plotting “wide-form” data

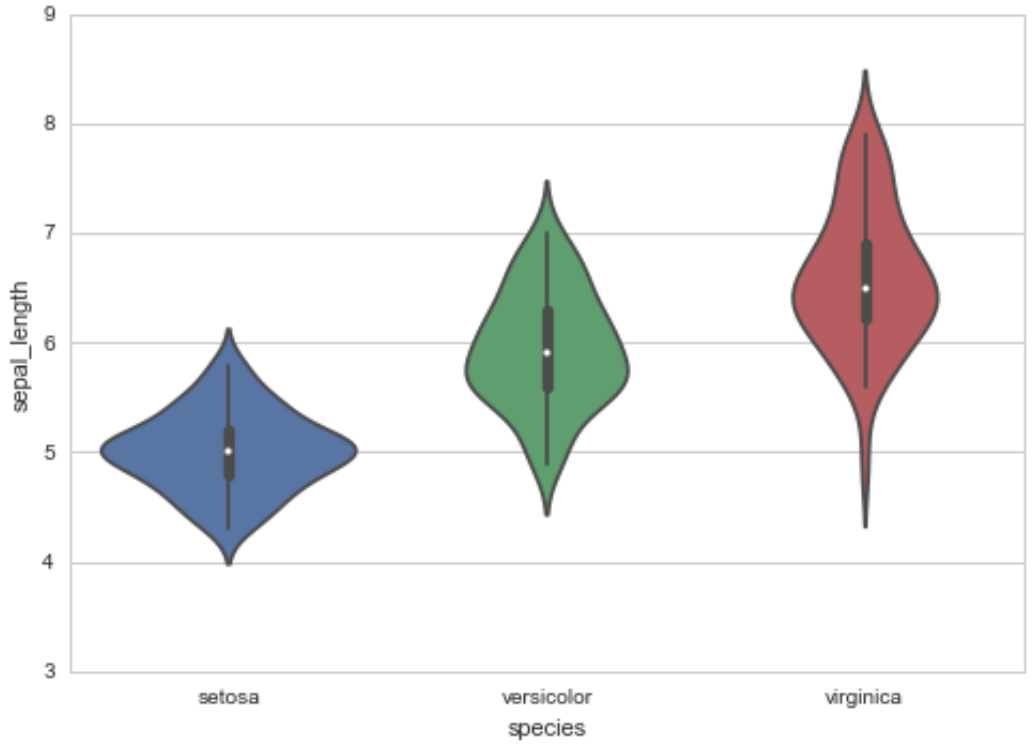
While using “long-form” or “tidy” data is preferred, these functions can also by applied to “wide-form” data in a variety of formats, including pandas DataFrames or two-dimensional numpy arrays. These objects should be passed directly to the `data` parameter:

```
sns.boxplot(data=iris, orient="h");
```



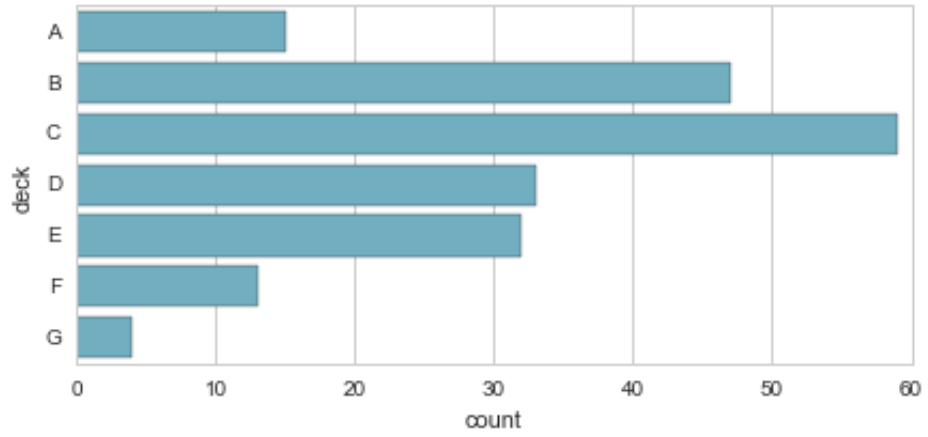
Additionally, these functions accept vectors of Pandas or numpy objects rather than variables in a `DataFrame`:

```
sns.violinplot(x=iris.species, y=iris.sepal_length);
```



To control the size and shape of plots made by the functions discussed above, you must set up the figure yourself using matplotlib commands. Of course, this also means that the plots can happily coexist in a multi-panel figure with other kinds of plots:

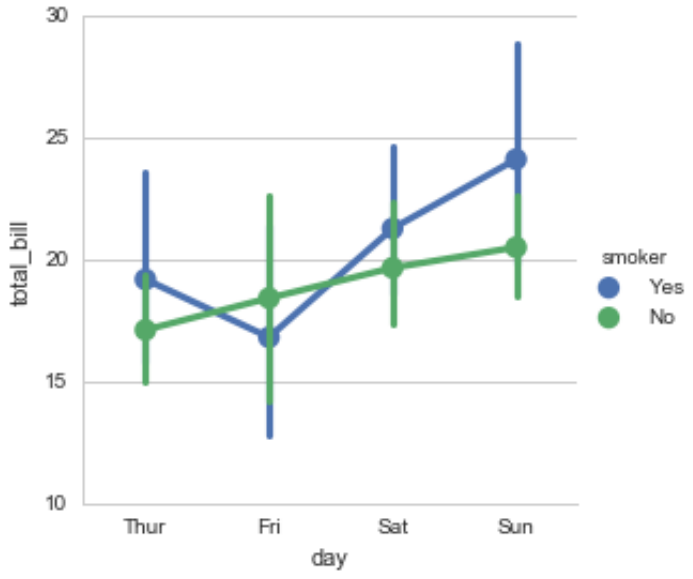
```
f, ax = plt.subplots(figsize=(7, 3))
sns.countplot(y="deck", data=titanic, color="c");
```



Drawing multi-panel categorical plots

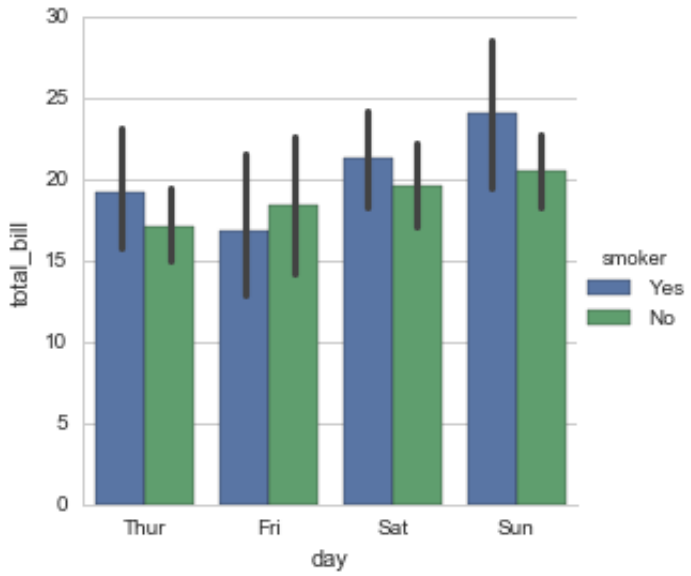
As we mentioned above, there are two ways to draw categorical plots in seaborn. Similar to the duality in the regression plots, you can either use the functions introduced above, or the higher-level function **`factorplot()`** (../generated/seaborn.factorplot.html#seaborn.factorplot), which combines these functions with a **`FacetGrid()`** (../generated/seaborn.FacetGrid.html#seaborn.FacetGrid) to add the ability to examine additional categories through the larger structure of the figure. By default, **`factorplot()`** (../generated/seaborn.factorplot.html#seaborn.factorplot) produces a **`pairplot()`** (../generated/seaborn.pairplot.html#seaborn.pairplot):

```
sns.factorplot(x="day", y="total_bill", hue="smoker", data=tips);
```



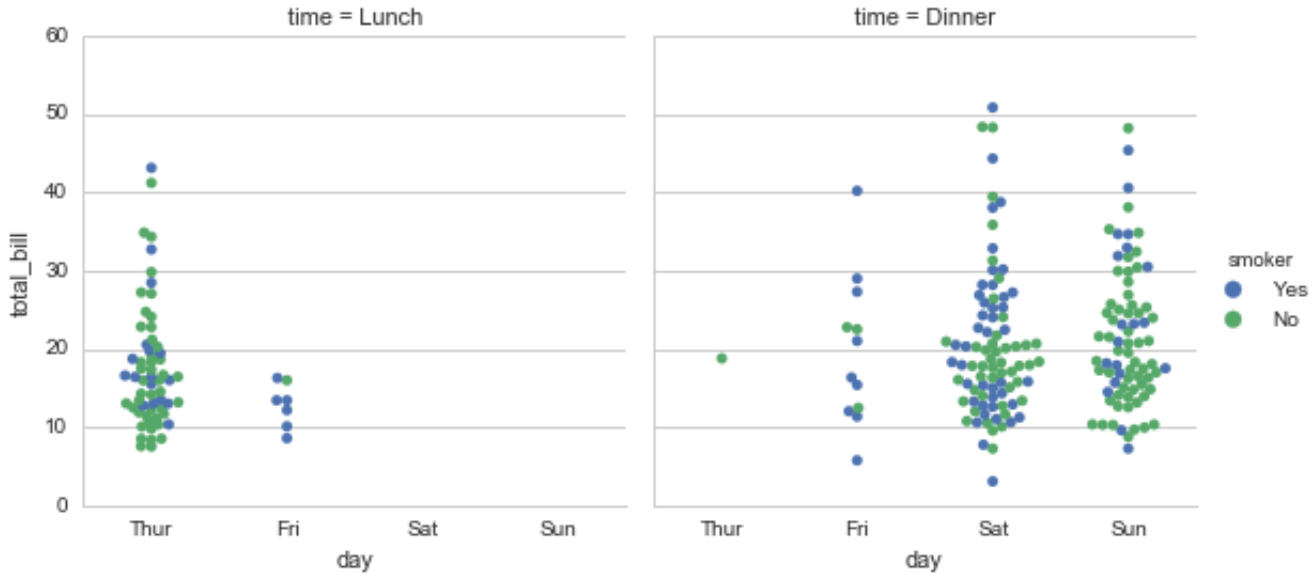
However, the `kind` parameter lets you chose any of the kinds of plots discussed above:

```
sns.factorplot(x="day", y="total_bill", hue="smoker", data=tips, kind="bar");
```



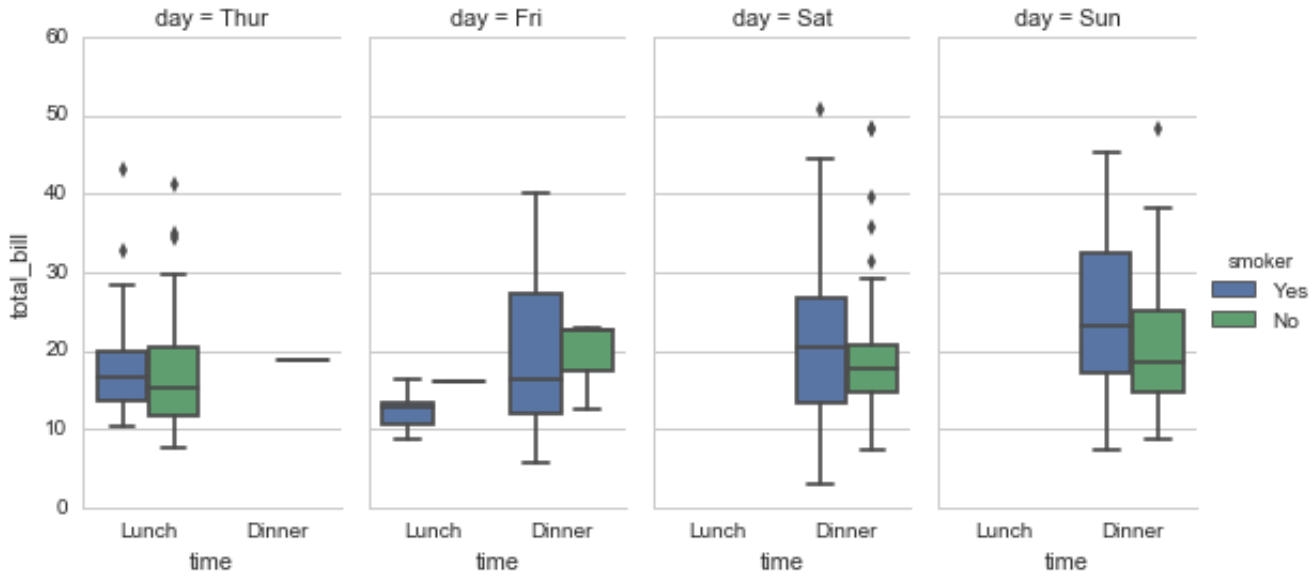
The main advantage of using a **`factorplot()`** (../generated/seaborn.factorplot.html#seaborn.factorplot) is that it is very easy to "facet" the plot and investigate the role of other categorical variables:

```
sns.factorplot(x="day", y="total_bill", hue="smoker",
               col="time", data=tips, kind="swarm");
```



Any kind of plot can be drawn. Because of the way **`FacetGrid`** (../generated/seaborn.FacetGrid.html#seaborn.FacetGrid) works, to change the size and shape of the figure you need to specify the `size` and `aspect` arguments, which apply to each facet:

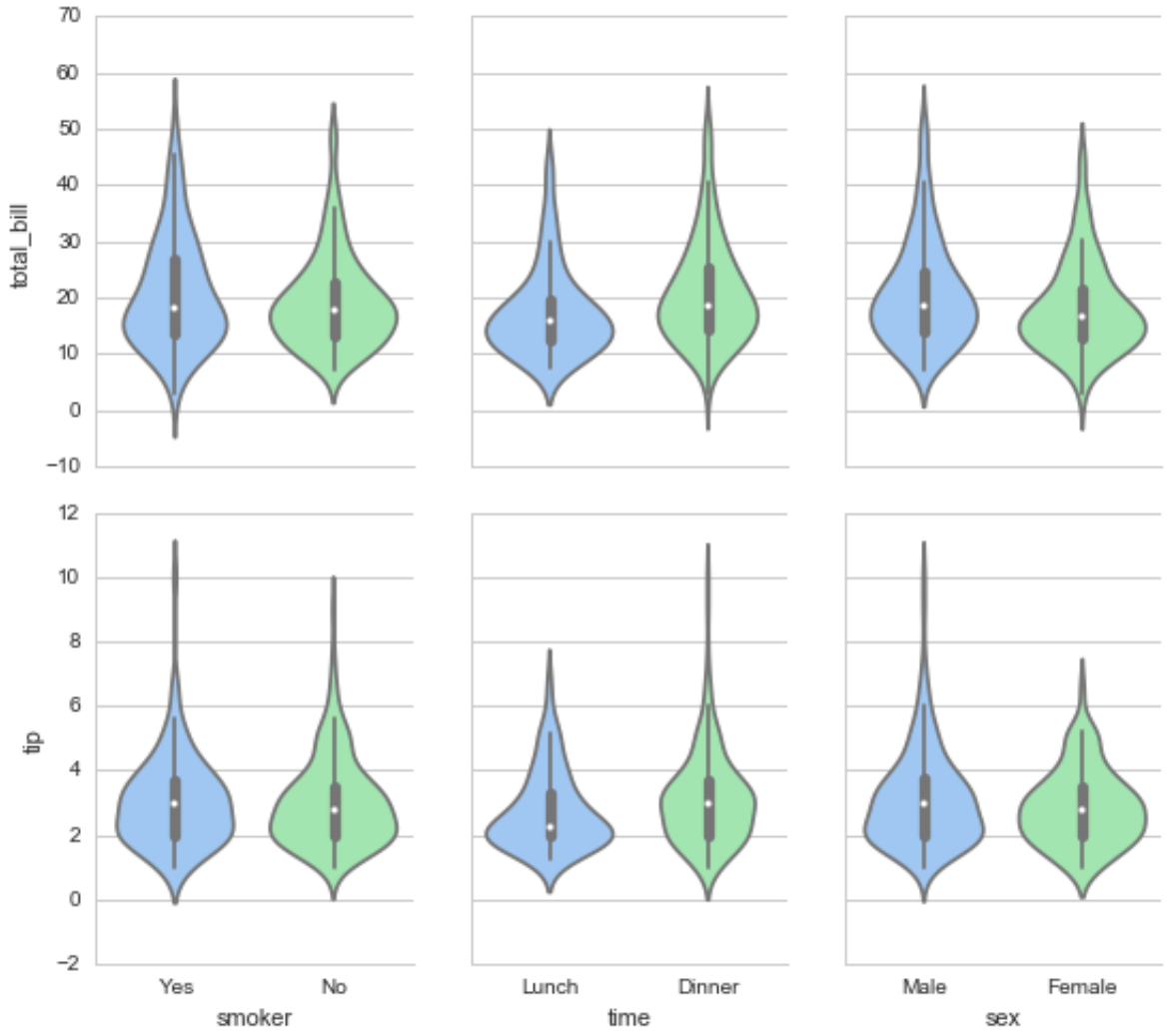
```
sns.factorplot(x="time", y="total_bill", hue="smoker",
               col="day", data=tips, kind="box", size=4, aspect=.5);
```



It is important to note that you could also make this plot by using **`boxplot()`** (../generated/seaborn.boxplot.html#seaborn.boxplot) and **`FacetGrid`** (../generated/seaborn.FacetGrid.html#seaborn.FacetGrid) directly. However, special care must be taken to ensure that the order of the categorical variables is enforced in each facet, either by using data with a `Categorical` datatype or by passing `order` and `hue_order`.

Because of the generalized API of the categorical plots, they should be easy to apply to other more complex contexts. For example, they are easily combined with a **`PairGrid`** (../generated/seaborn.PairGrid.html#seaborn.PairGrid) to show categorical relationships across several different variables:

```
g = sns.PairGrid(tips,
                 x_vars=["smoker", "time", "sex"],
                 y_vars=["total_bill", "tip"],
                 aspect=.75, size=3.5)
g.map(sns.violinplot, palette="pastel");
```



Source (../_sources/tutorial/categorical.txt)

Back to top