# Data Visualization

**AcadView**

May 23, 2018

---

## 1 Overview

When dealing with a set of data, often the first thing youll want to do is get a sense for how the variables are distributed. This chapter of the tutorial will give a brief introduction to some of the tools in 'seaborn for examining univariate and bivariate distributions.

## 2 Visualizing the distribution of a dataset

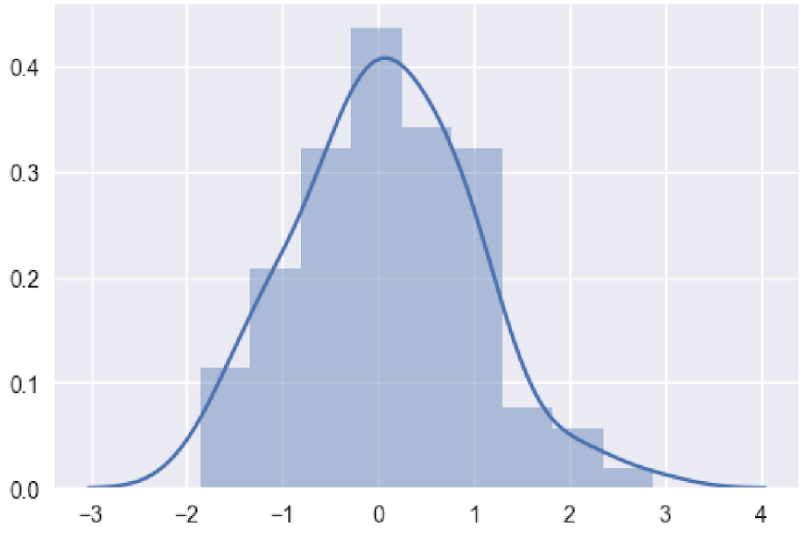
### 2.1 Plotting univariate distributions

The most convenient way to take a quick look at a univariate distribution in seaborn is the distplot() function. By default, this will draw a histogram and fit a kernel density estimate (KDE).

#### 2.1.1 Histograms

Histograms are likely familiar, and a hist function already exists in matplotlib. A histogram represents the distribution of data by forming bins along the range of the data and then drawing bars to show the number of observations that fall in each bin.

```
x = np.random.normal(size=100)
sns.distplot(x);
```



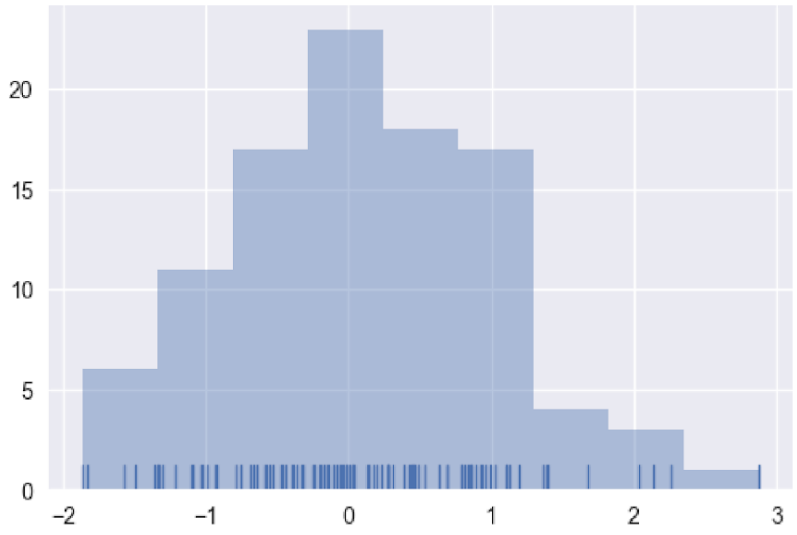

```
import matplotlib.pyplot as plt
```

To illustrate this, lets remove the density curve and add a rug plot, which draws a small vertical tick at each observation. You can make the rug plot itself with the rugplot() function, but it is also available in distplot():

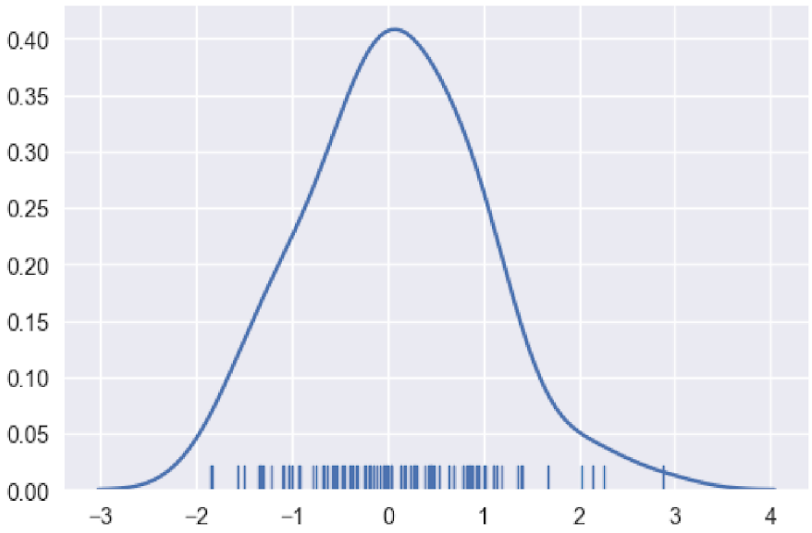
### 2.1.2 Kernel density estimation

The kernel density estimate may be less familiar, but it can be a useful tool for plotting the shape of a distribution. Like the histogram, the KDE plots encodes the density of observations on one axis with height along the other axis:

```
sns.distplot(x, kde=False, rug=True);
```

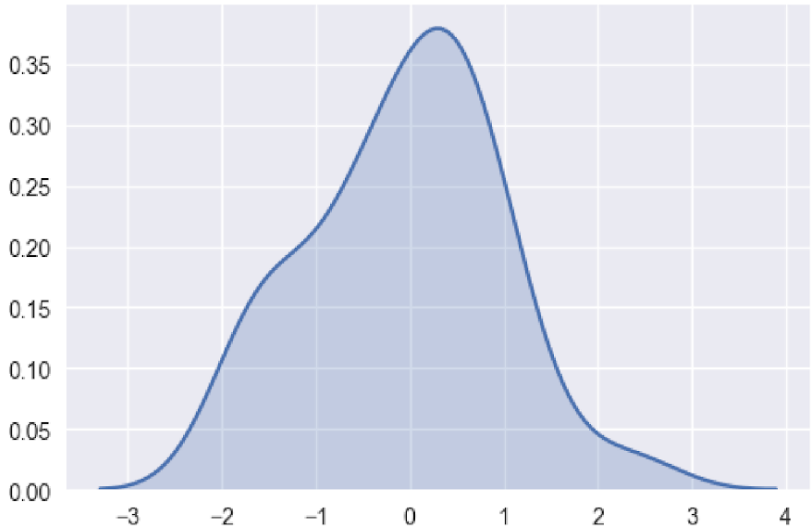



```
sns.distplot(x, hist=False, rug=True);
```



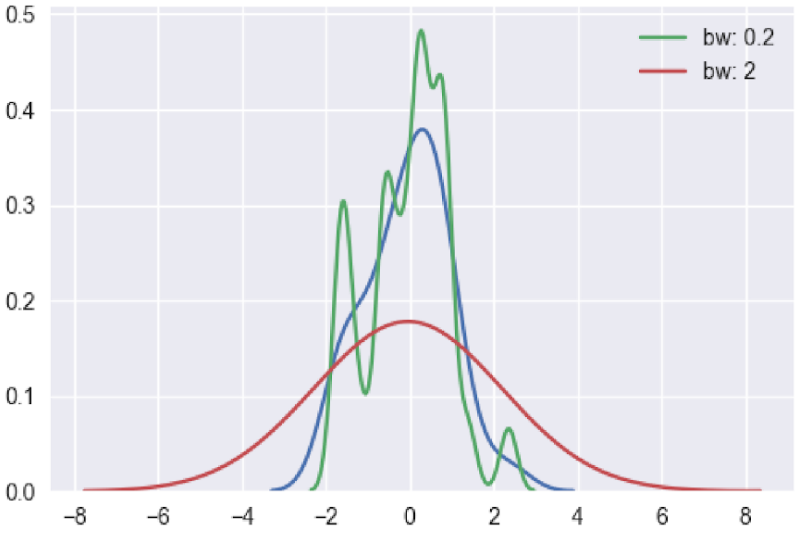
We can see that if we use the kdeplot() function in seaborn, we get the same curve. This function is used by distplot(), but it provides a more direct interface with easier access to other options when you just want the density estimate:

```
sns.kdeplot(x, shade=True);
```




The bandwidth (bw) parameter of the KDE controls how tightly the estimation is fit to the data, much like the bin size in a histogram. It corresponds to the width of the kernels we plotted above. The default behavior tries to guess a good value using a common reference rule, but it may be helpful to try larger or smaller values:

```
sns.kdeplot(x)
sns.kdeplot(x, bw=.2, label="bw: 0.2")
sns.kdeplot(x, bw=2, label="bw: 2")
plt.legend();
```



## 2.2 Plotting bivariate distributions

It can also be useful to visualize a bivariate distribution of two variables. The easiest way to do this in seaborn is to just use the jointplot() function, which creates a multi-panel figure that shows both the bivariate (or joint) relationship between two variables along with the univariate (or marginal) distribution of each on separate axes.

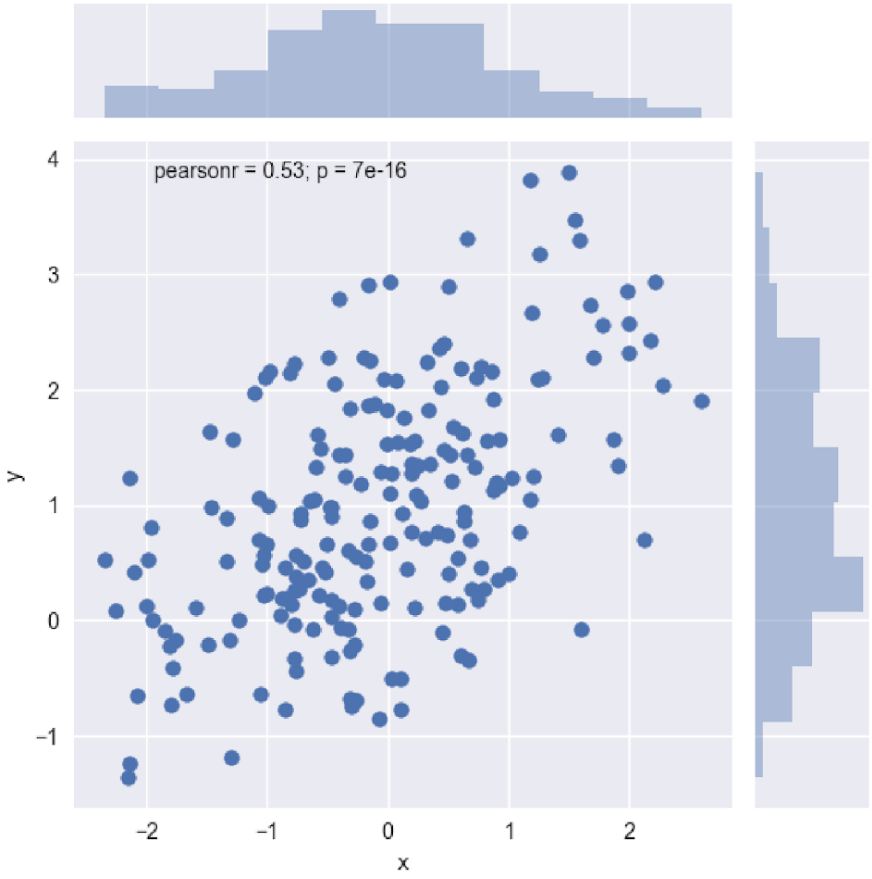
### 2.2.1 Scatterplots

The most familiar way to visualize a bivariate distribution is a scatterplot, where each

```
mean, cov = [0, 1], [(1, .5), (.5, 1)]
data = np.random.multivariate_normal(mean, cov, 200)
df = pd.DataFrame(data, columns=["x", "y"])
```

observation is shown with point at the x and y values. This is analgous to a rug plot on two dimensions. You can draw a scatterplot with the matplotlib plt.scatter function, and it is also the default kind of plot shown by the jointplot() function:

```
sns.jointplot(x="x", y="y", data=df);
```



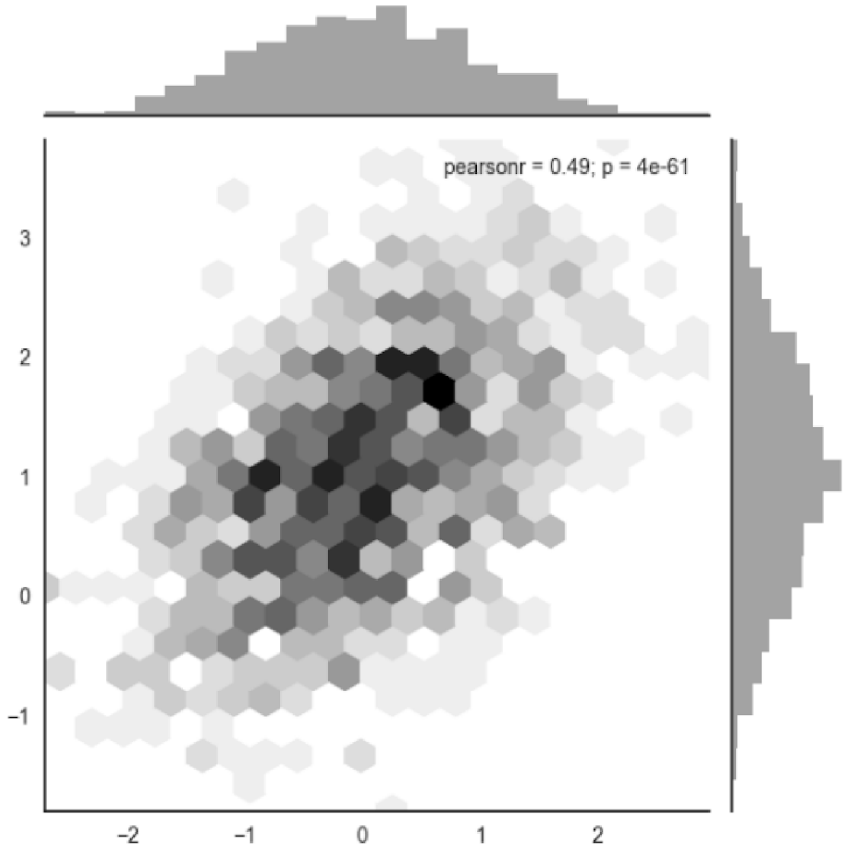
### 2.2.2 Hexbin plots

The bivariate analogue of a histogram is known as a hexbin plot, because it shows the counts of observations that fall within hexagonal bins. This plot works best with relatively large datasets. Its available through the matplotlib plt.hexbin function and as a style in jointplot(). It looks best with a white background:

### 2.2.3 Kernel density estimation

It is also possible to use the kernel density estimation procedure described above to visualize a bivariate distribution. In seaborn, this kind of plot is shown with a contour plot and is available as a style in jointplot():

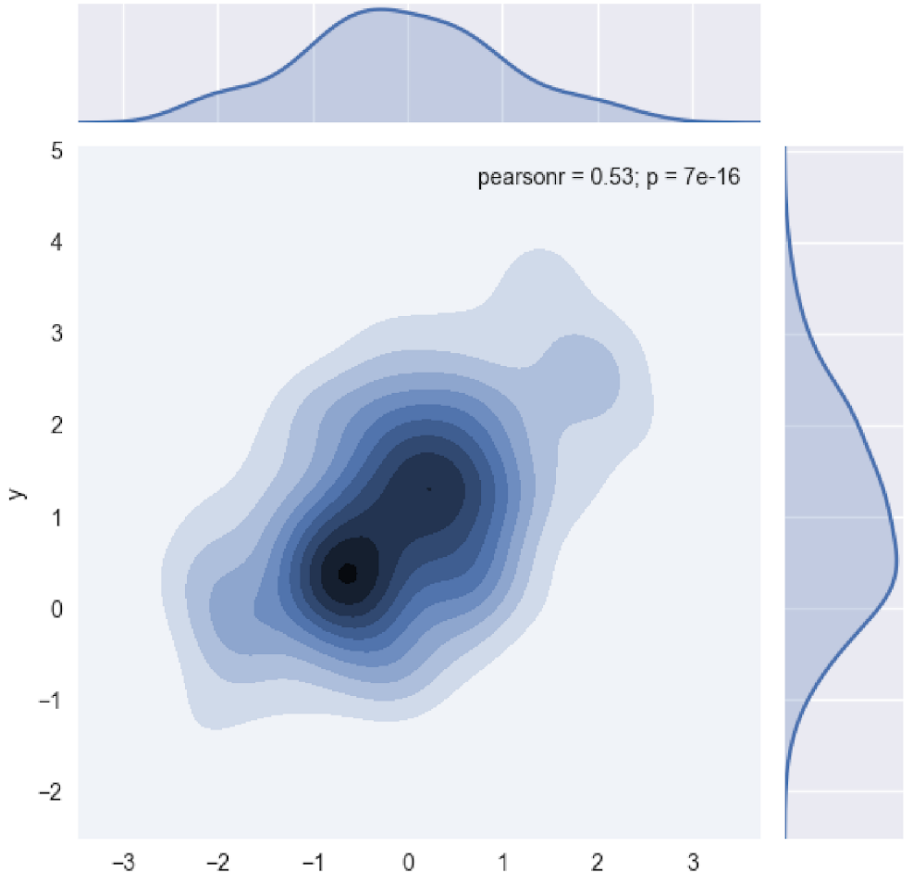
```
x, y = np.random.multivariate_normal(mean, cov, 1000).T
with sns.axes_style("white"):
    sns.jointplot(x=x, y=y, kind="hex", color="k");
```



## 2.3 Visualizing pairwise relationships in a dataset

To plot multiple pairwise bivariate distributions in a dataset, you can use the pairplot() function. This creates a matrix of axes and shows the relationship for each pair of columns

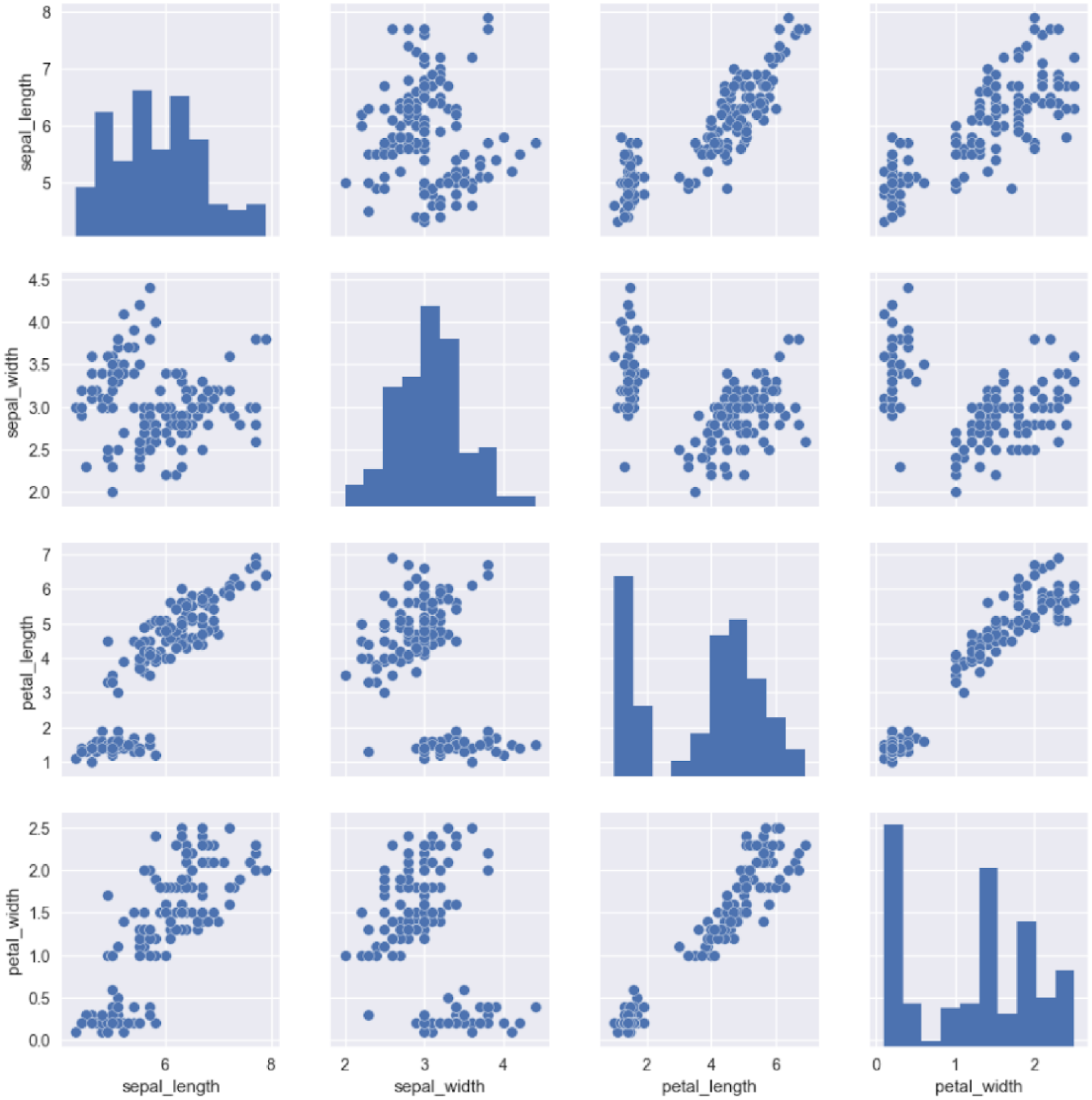
```
sns.jointplot(x="x", y="y", data=df, kind="kde");
```




in a DataFrame. by default, it also draws the univariate distribution of each variable on the diagonal Axes:

# 3 Plotting with categorical data

```
iris = sns.load_dataset("iris")
sns.pairplot(iris);
```



Its useful to divide seaborns 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() and stripplot(), the second includes boxplot(), and the third includes barplot(). 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.

```
%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")))
```

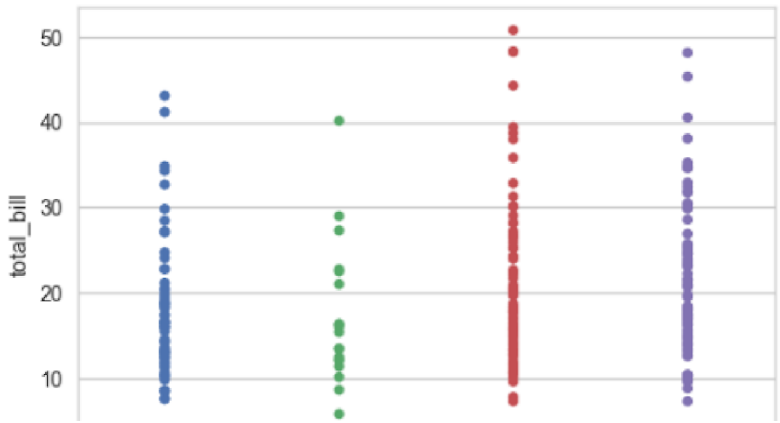
```
titanic = sns.load_dataset("titanic")
tips = sns.load_dataset("tips")
iris = sns.load_dataset("iris")
```

## 3.1 Categorical scatterplots

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

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);
```

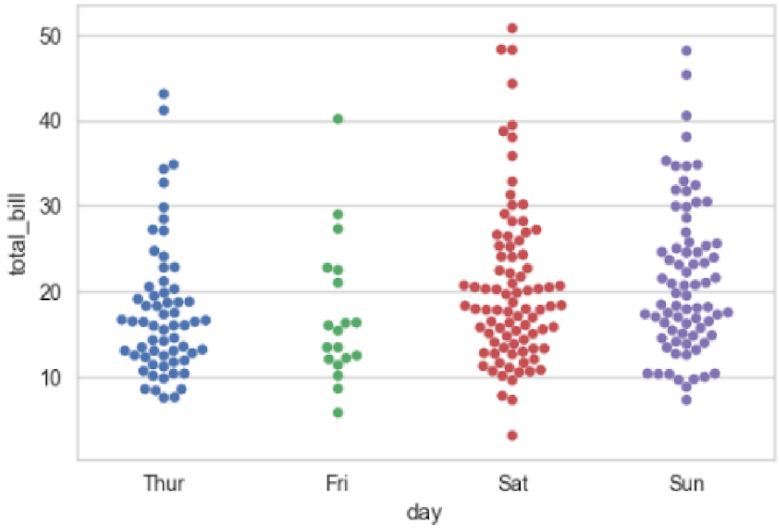


A different approach would be to use the function swarmplot(), which positions each scatterplot point on the categorical axis with an algorithm that avoids overlapping points:

## 3.2 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

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



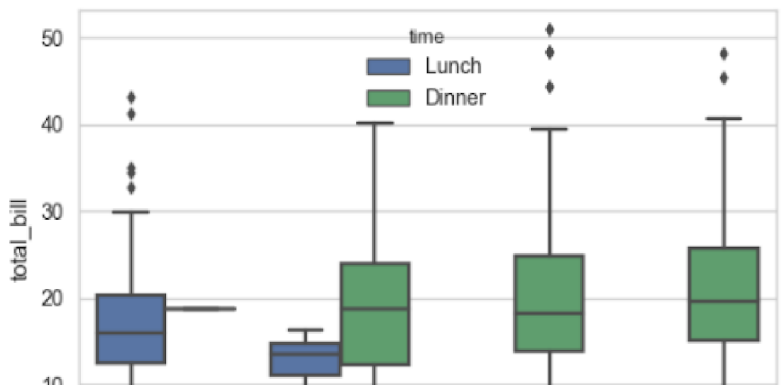

category levels. These generalize some of the approaches we discussed in the chapter to the case where we want to quickly compare across several distributions.

### 3.2.1 Boxplots

The first is the familiar 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:

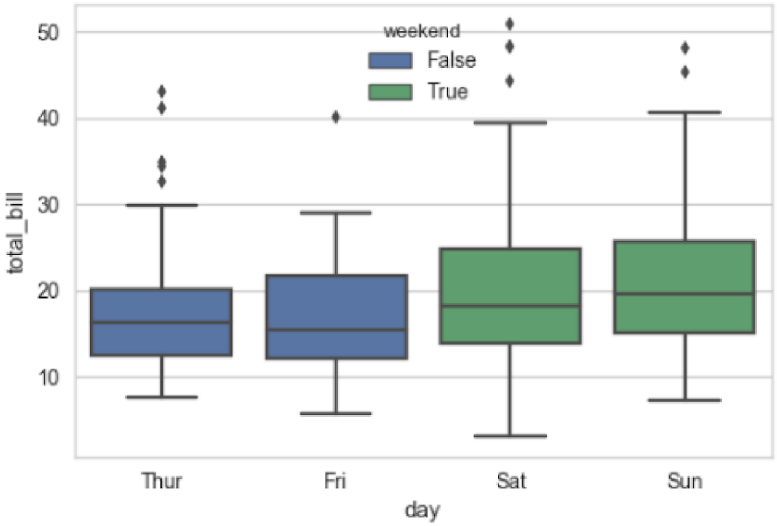
For boxplots, the assumption when using a hue variable is that it is nested within the x or y variable. This means that by default, the boxes for different levels of hue will be offset, as

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



you can see above. If your hue variable is not nested, you can set the dodge parameter to disable offsetting:

```
tips["weekend"] = tips["day"].isin(["Sat", "Sun"])
sns.boxplot(x="day", y="total_bill", hue="weekend", data=tips, dodge=False);
```



## 3.3 Statistical estimation within categories

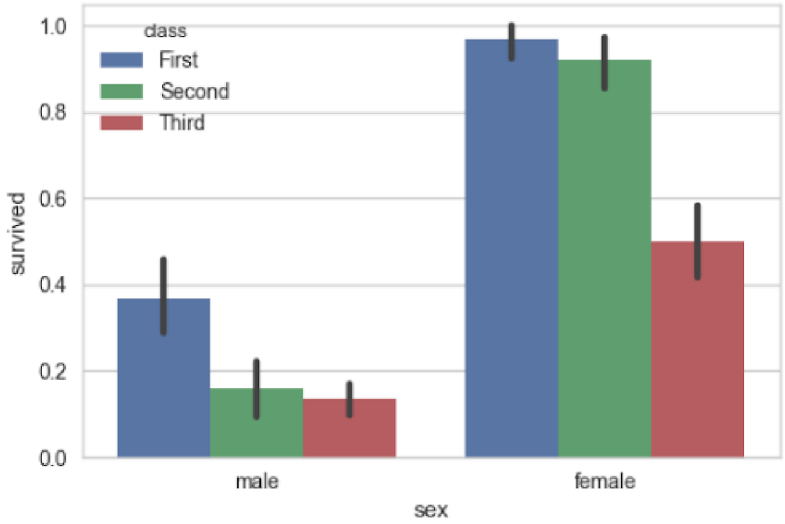
Often, rather than showing the distribution within each category, you might want to show the central tendency of the values. Seaborn has two main ways to show this information, but importantly, the basic API for these functions is identical to that for the ones discussed above.

### 3.3.1 Bar plots

A familiar style of plot that accomplishes this goal is a bar plot. In seaborn, the 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:

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, its easy to do so with the countplot() function:

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



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