Extra: Kernel Density Estimation

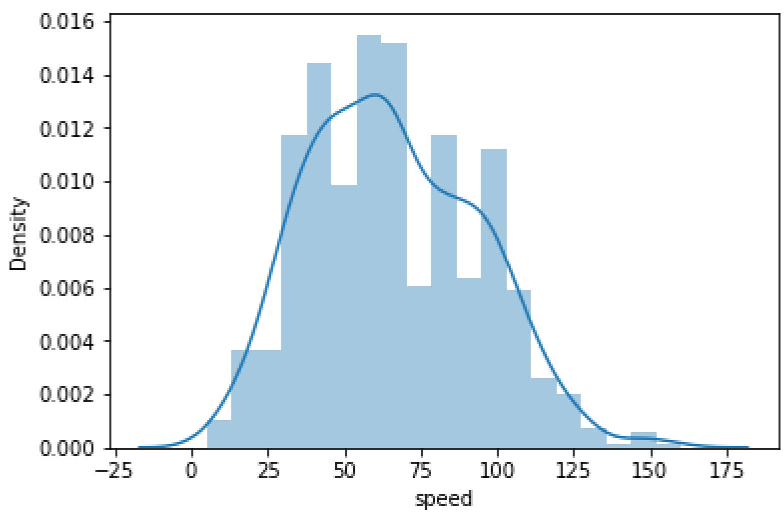
At the end of this lesson and the next, you'll find some extra concepts that didn't really fit into the main lesson flow. These concepts will cover a few additional univariate plots that you might be interested in using or might observe in your own research. While bar charts and histograms should cover your most common needs, the plots in this section might prove useful for both the exploratory and explanatory sides of data visualization.

Kernel Density Estimation

Earlier in this lesson, you saw an example of [kernel density estimation]() (KDE) through the use of seaborn's [distplot]() function, which plots a KDE on top of a histogram.

Example 1. Plot the Kernel Density Estimation (KDE)

```
# The pokemon dataset is available to download at the bottom of this page.
sb.distplot(pokemon['speed']);
```



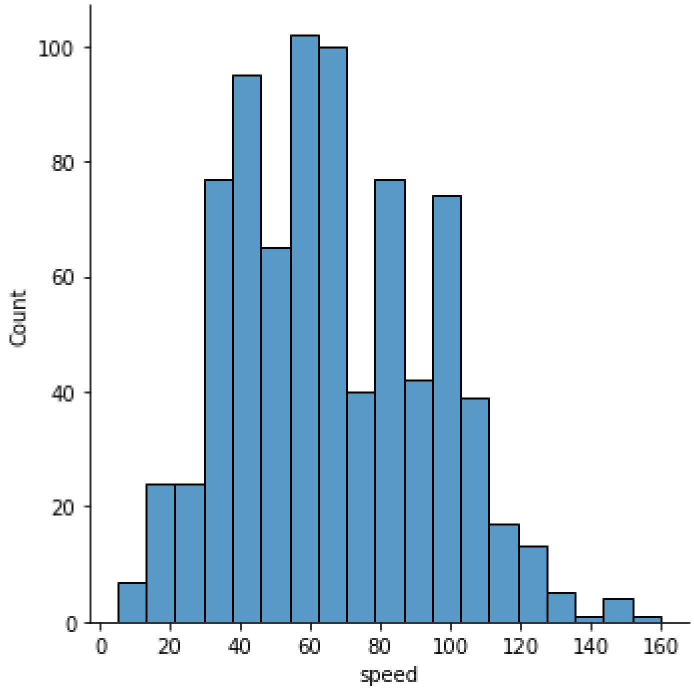
Note - The distplot() function is deprecated in Seaborn v 0.11.0, and will be removed in a future version. The alternative is either of the following:

1. [displot()]() - A figure-level function with similar flexibility.
2. [histplot()]() - An axes-level function for histograms.

See the same example with newer displot() function:

```
# Use this new function only with Seaborn 0.11.0 and above.
# The kind argument can take any one value from {“hist”, “kde”, “ecdf”}.
sb.displot(pokemon['speed'], kind='hist');
```

```
# Use the 'kde' kind for kernel density estimation
# sb.displot(pokemon['speed'], kind='kde');
```



Kernel density estimation is one way of estimating the probability density function of a variable. In a KDE plot, you can think of each observation as replaced by a small ‘lump’ of area. Stacking these lumps all together produces the final density curve. The default settings use a normal-distribution kernel, but most software that can produce KDE plots also include other kernel function options.

Seaborn's distplot function calls another function, kdeplot, to generate the KDE. The demonstration code below also uses a third function called by distplot for illustration, rugplot(). In a rugplot, data points are depicted as dashes on a number line.

Example 2. Demonstrating distplot() and rugplot() to plot the KDE

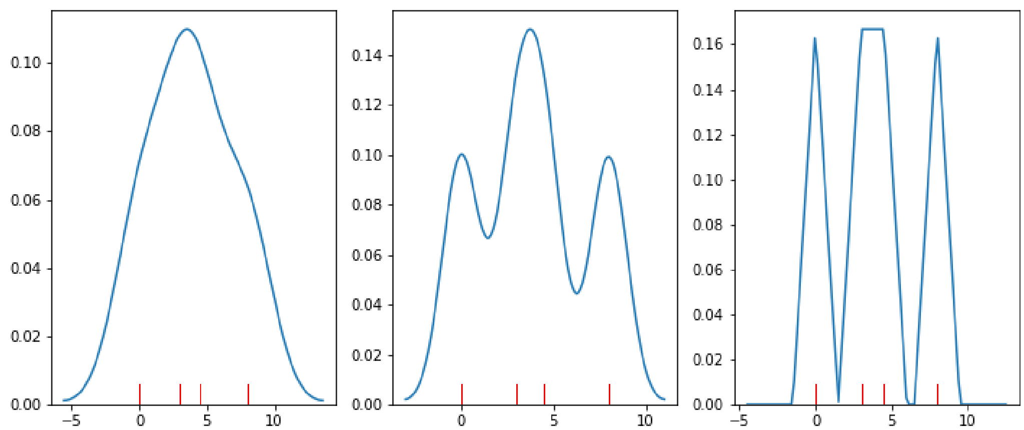
```
data = [0.0, 3.0, 4.5, 8.0]
plt.figure(figsize = [12, 5])


# left plot: showing kde lumps with the default settings
plt.subplot(1, 3, 1)
sb.distplot(data, hist = False, rug = True, rug_kws = {'color' : 'r'})


# central plot: kde with narrow bandwidth to show individual probability lumps
```

```
plt.subplot(1, 3, 2)
sb.distplot(data, hist = False, rug = True, rug_kws = {'color' : 'r'},
        kde_kws = {'bw' : 1})

# right plot: choosing a different, triangular kernel function (lump shape)
plt.subplot(1, 3, 3)
sb.distplot(data, hist = False, rug = True, rug_kws = {'color' : 'r'},
        kde_kws = {'bw' : 1.5, 'kernel' : 'tri'})
```



Interpreting proportions from this plot type is slightly trickier than a standard histogram: the vertical axis indicates a density of data rather than straightforward proportions. Under a KDE plot, the total area between the 0-line and the curve will be 1. The probability of an outcome falling between two values is found by computing the area under the curve that falls between those values. Making area judgments like this without computer assistance is difficult and likely to be inaccurate.

Despite the fact that making specific probability judgments are not as intuitive with KDE plots as histograms, there are still reasons to use kernel density estimation. If there are relatively few data points available, KDE provides a smooth estimate of the overall distribution of data. These ideas may not be so easily conveyed through histograms, in which the large discreteness of jumps may end up misleading.

It should also be noted that there is a bandwidth parameter in KDE that specifies how wide the density lumps are. Similar to bin width for histograms, we need to choose a bandwidth size that best shows the signal in the data. A too-small bandwidth can make the data look noisier than it really is, and a too-large bandwidth can smooth out useful features that we could use to make inferences about the data. It's good to keep this in mind in case the default bandwidths chosen by your visualization software don't look quite right or if you need to perform further investigations.