1.3 Visualizing Variables

May 8, 2019

1 1.3 Visualizing Variables

The classic expression, "A picture is worth a thousand words," applies to data science. A graphic usually conveys more information (and more efficiently) than the summary statistics we examined in the previous section. In this section, you will learn how to produce simple graphics for quantitative and categorical variables. For a more detailed discussion of graphics, see Chapter 10.

1.1 Graphics in the Jupyter Notebook

By default, Python outputs graphics to a new window. But what if we want a record of the graphic? We could save the graphic to an image file, but then the graphic and the code that generated it would live in different files. If these files were to ever get separated, then it may be difficult to regenerate the graphic. In other words, the standard Python workflow is not **reproducible**.

Jupyter notebooks support a reproducible workflow, by allowing graphics to be embedded directly in a notebook. Now, the graphic and the code that generated it live in the same file, adjacent to one another. To make graphics show up in the Jupyter notebook, we have to specify that `matplotlib` (the main graphics library in Python) should output the graphic to the "inline" backend, as opposed to, for example, a backend that makes the graphic appear in a new window. To specify a backend for `matplotlib`, we run a so-called **magic command** (or just **magic**, for short). Magic commands modify the behavior of a notebook or an individual cell. For example, the `%timeit` magic, which we will use later in this book, times how long it takes to run a line of code. You can recognize magics because they are preceded by `%` or `%%`. For a full list of magics, consult the documentation.

The `%matplotlib` magic below allows you to specify a backend. In general, if you plan to create graphics in the Jupyter notebook, then the following magic should be the first line in your notebook.

```
In [1]: %matplotlib inline
```

1.2 Visualizing Quantitative Variables

Graphics can help us understand how the values of a quantitative variable are distributed. We will study two types of visualizations for quantitative variables: histograms and densities.

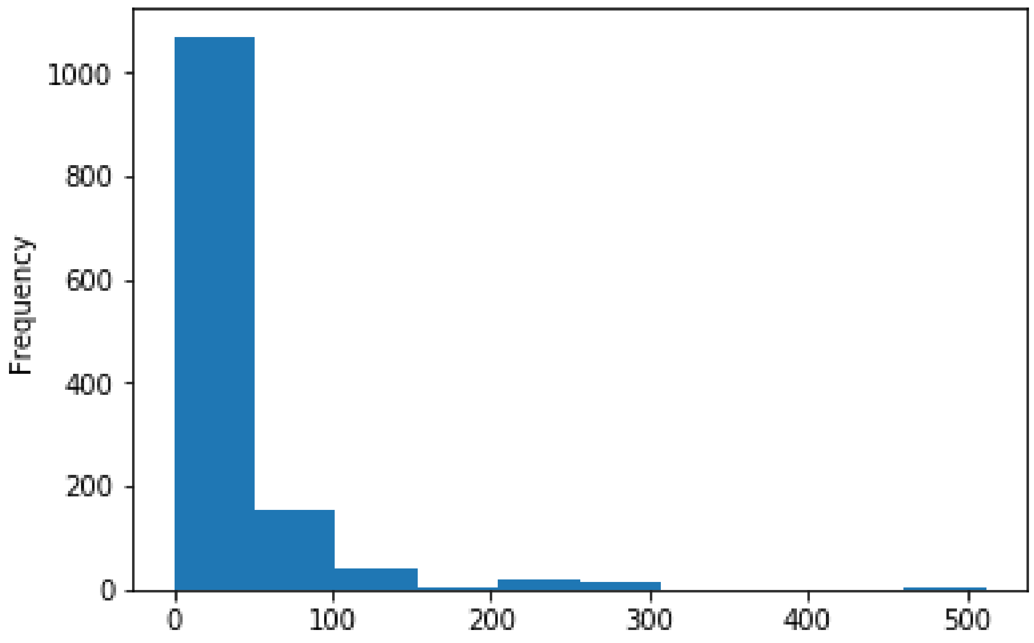
1.2.1 Histograms

The standard visualization for a single quantitative variable is the **histogram**. A histogram sorts the values into bins and uses bars to represent the number of values in each bin.

To make a histogram, we call the `.plot.hist()` method of the selected variable. All of the plotting functions in `pandas` are preceded by `.plot`.

```
In [2]: import pandas as pd
        df = pd.read_csv(
            "https://raw.githubusercontent.com/dlsun/data-science-book/master/data/titanic.csv
        )

        df.fare.plot.hist()
```

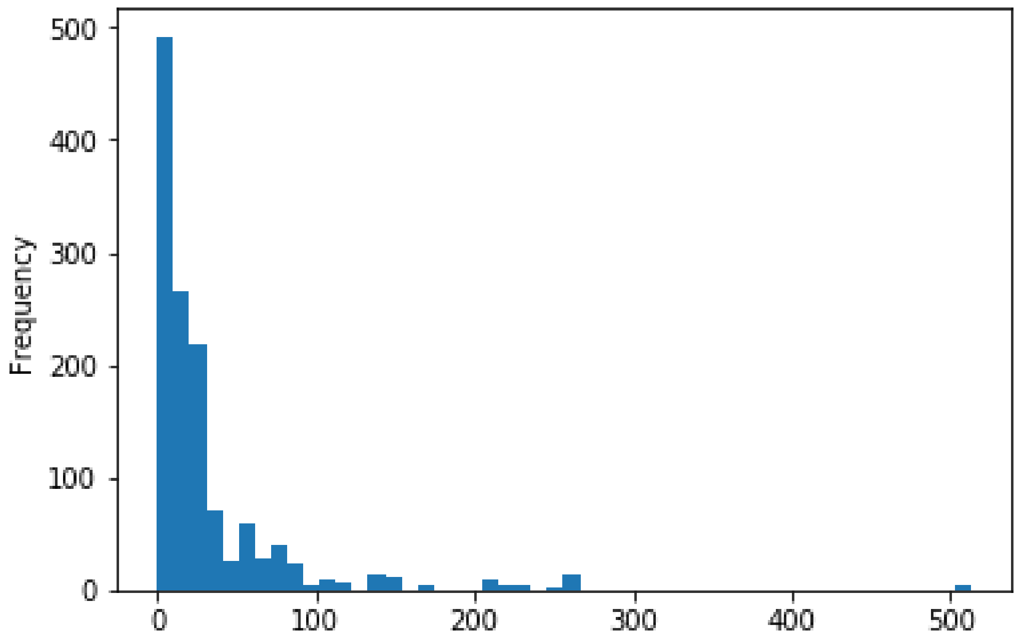
```
Out[2]: <matplotlib.axes._subplots.AxesSubplot at 0x7f85e8b456d8>
```



It seems that we did not get enough resolution to really determine what is going on at the lower end of the scale. Let's request more bins.

```
In [3]: df.fare.plot.hist(bins=50)
```

```
Out[3]: <matplotlib.axes._subplots.AxesSubplot at 0x7f85e90d49e8>
```

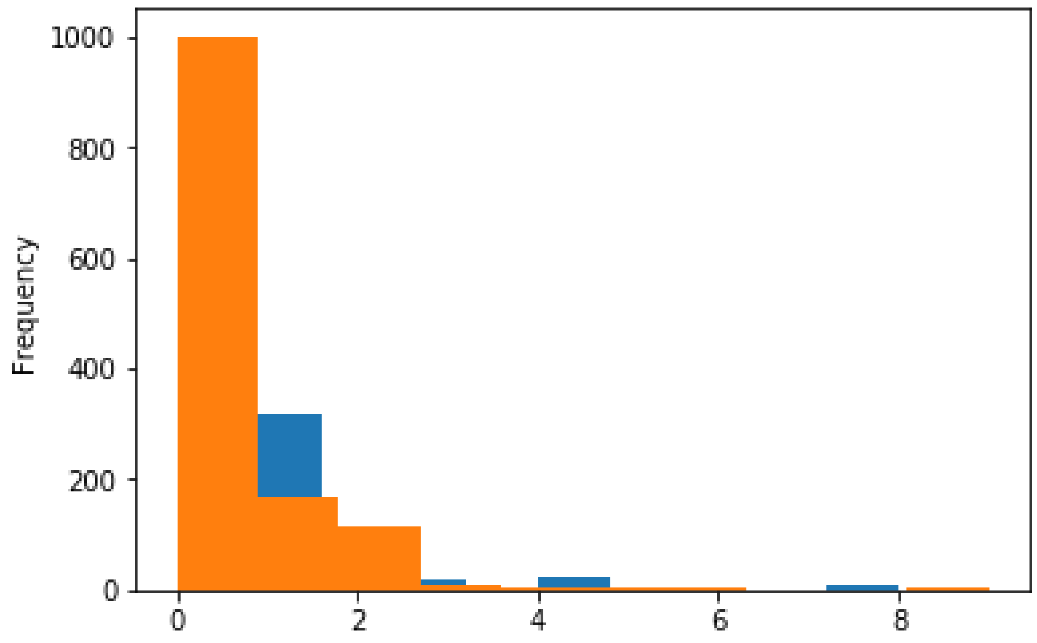


From this graph, we see a concentration of values around 10-30 (which we previously identified as the "center") and a spread of about 30-50 (which we previously identified as the "spread"). We also see the outlier who paid more than č500. We also see features that were not obvious before: the skewed shape of the distribution, the gap between č300 and č500, and so on. This single picture has managed to convey more information than a dozen summary statistics.

We might want to plot more than one histogram on the same graphic to make for easy comparison. To do this, we simply make multiple calls to plotting functions within the same cell. For example, if we wanted to compare the distributions of the number of siblings/spouses and the number of parents/children that accompanied passengers, we could call `.plot.hist()` twice.

```
In [4]: df.sibsp.plot.hist()
        df.parch.plot.hist()
```

```
Out[4]: <matplotlib.axes._subplots.AxesSubplot at 0x7f85e8fb3be0>
```

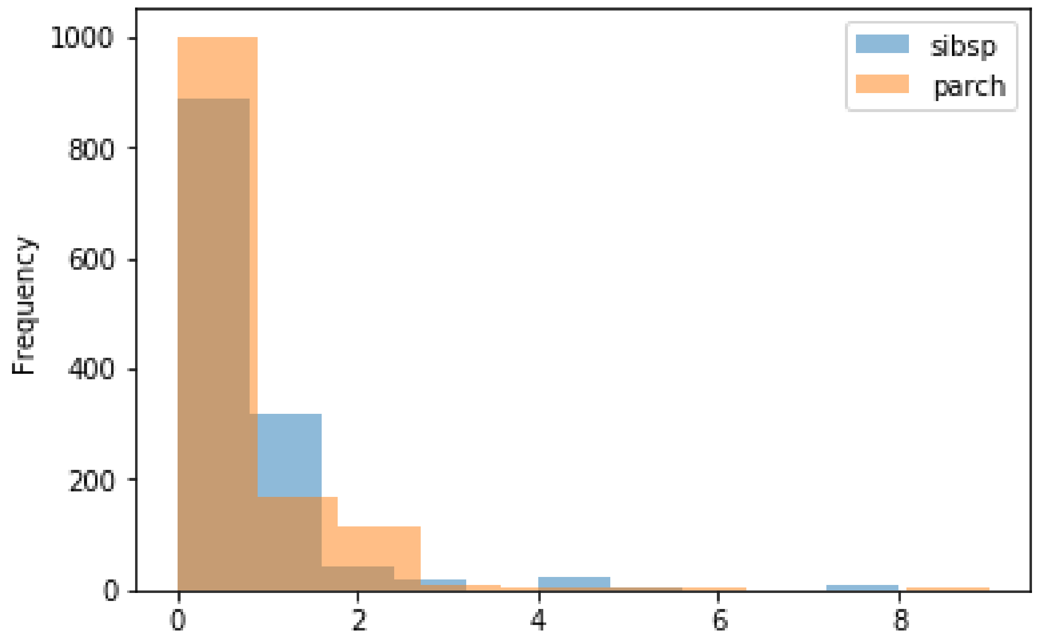


Notice that pandas automatically plotted the two histograms using different colors. There are two problems with this plot. First, we don't know which color corresponds to which variable. Second, we cannot see the blue histogram underneath the orange histogram because the colors are opaque.

To solve the first problem, we add a legend for each variable by specifying `legend=True`. To solve the second problem, we set the transparency `alpha`, which is a number between 0 and 1, with 0 being perfectly transparent and 1 being completely opaque. Try varying `alpha` to get a feel for what it does.

```
In [5]: df.sibsp.plot.hist(legend=True, alpha=.5)
        df.parch.plot.hist(legend=True, alpha=.5)
```

```
Out[5]: <matplotlib.axes._subplots.AxesSubplot at 0x7f85e8f37978>
```



The parents/children histogram is higher at 0 and 2, but the sibling/spouse histogram is higher at 1. This makes sense because

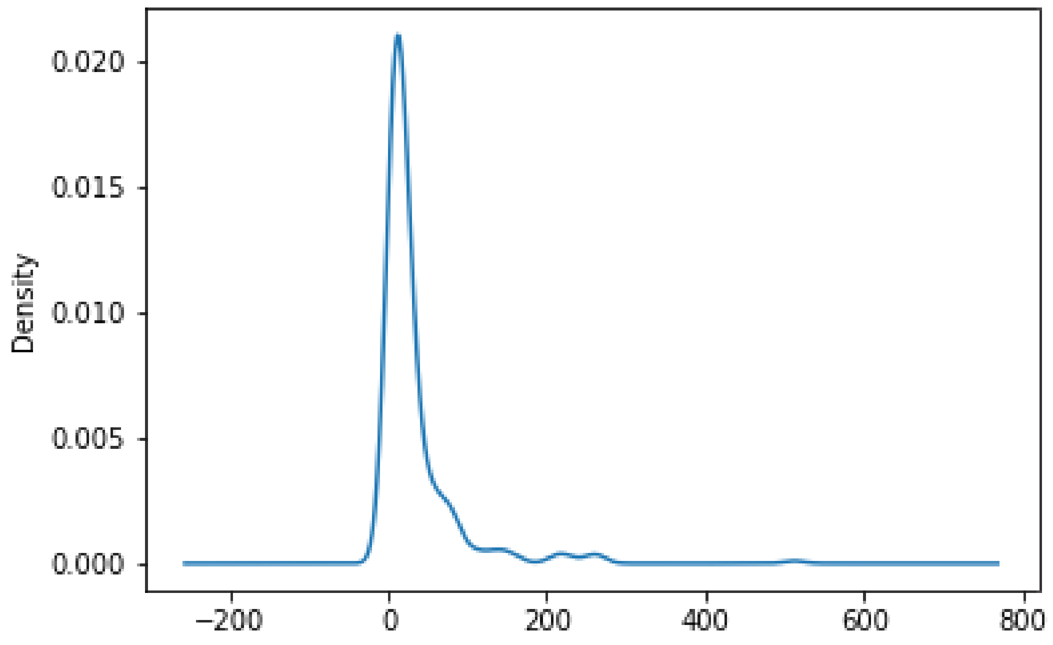
1. There were many childless couples on the Titanic (i.e., 0 children).
2. You can only have 1 spouse, but many children travel with 2 parents.

1.2.2 Densities

Another way to visualize the distribution of a quantitative variable is by plotting its **density**. A density plot turns the jagged histogram into a smooth curve, allowing the user to focus on the general shape of the distribution.

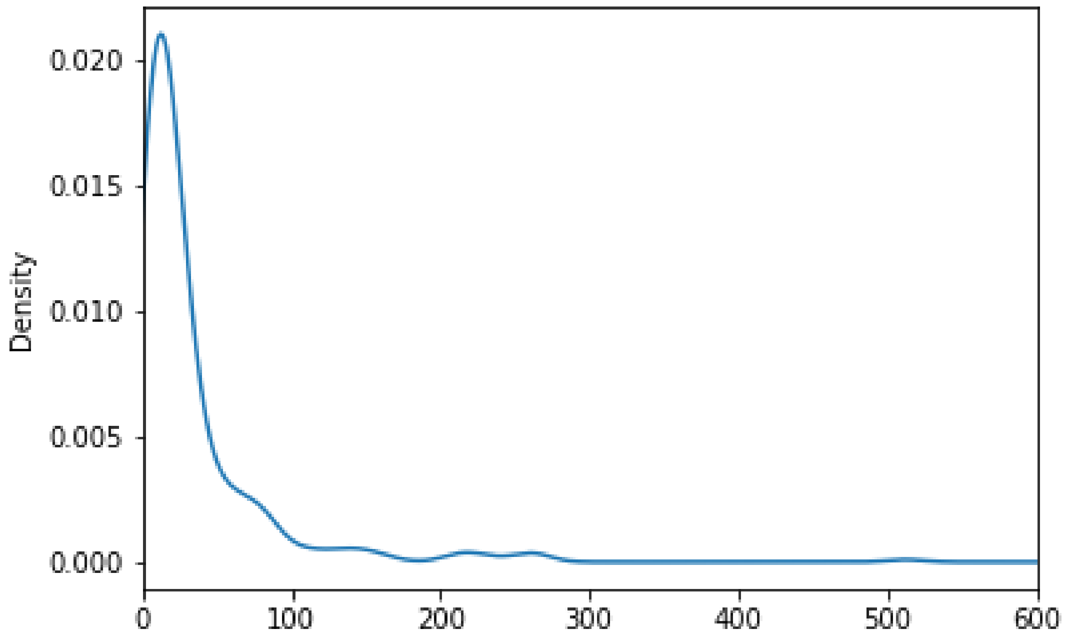
```
In [6]: df.fare.plot.density()
```

```
Out[6]: <matplotlib.axes._subplots.AxesSubplot at 0x7f85e8f0e0b8>
```



The x-axis is too wide. (You cannot have negative fares.) We can set the limits of the x-axis manually using the `xlim` argument.

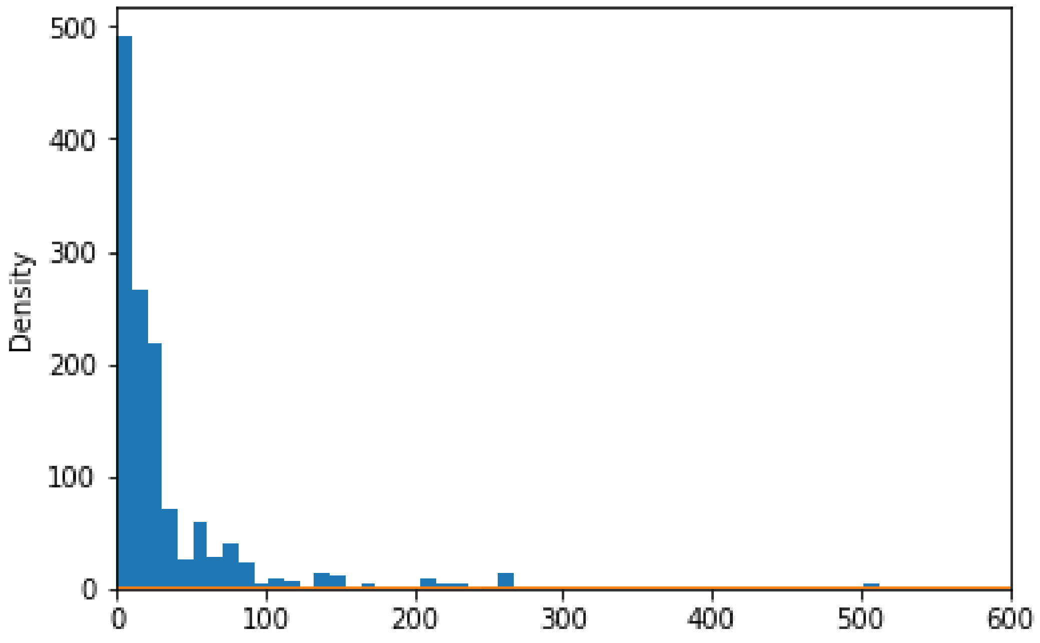
```
In [7]: df.fare.plot.density(xlim=(0, 600))

Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x7f85dea84828>
```



Let's superimpose this density curve on top of the histogram, by making two calls to plotting functions:

```
In [8]: df.fare.plot.hist(bins=50)
        df.fare.plot.density(xlim=(0, 600))
```

```
Out[8]: <matplotlib.axes._subplots.AxesSubplot at 0x7f85dea655c0>
```



If you squint at this plot, you will see an orange line at the bottom of the plot. This is supposed to be the density. But why does it appear as a flat line? The y-axis offers a hint. When we made the density plot earlier, the y-axis extended from 0 to about 0.02. Now the y-axis extends all the way to 500. On such a scale, a curve that fluctuates between 0 and 0.02 will appear to be a flat line!

The problem is that the histogram and the density are currently on different scales. By default, histograms display counts, while densities are defined so that the total area under the curve is 1. To be able to display a histogram and density on the same graph, we have to normalize the histogram so that the total area of the bars is 1. We can do this by setting the option `density=True`.

```
In [9]: df.fare.plot.hist(bins=50, density=True)
        df.fare.plot.density(xlim=(0, 600))
```

```
Out[9]: <matplotlib.axes._subplots.AxesSubplot at 0x7f85de9a21d0>
```