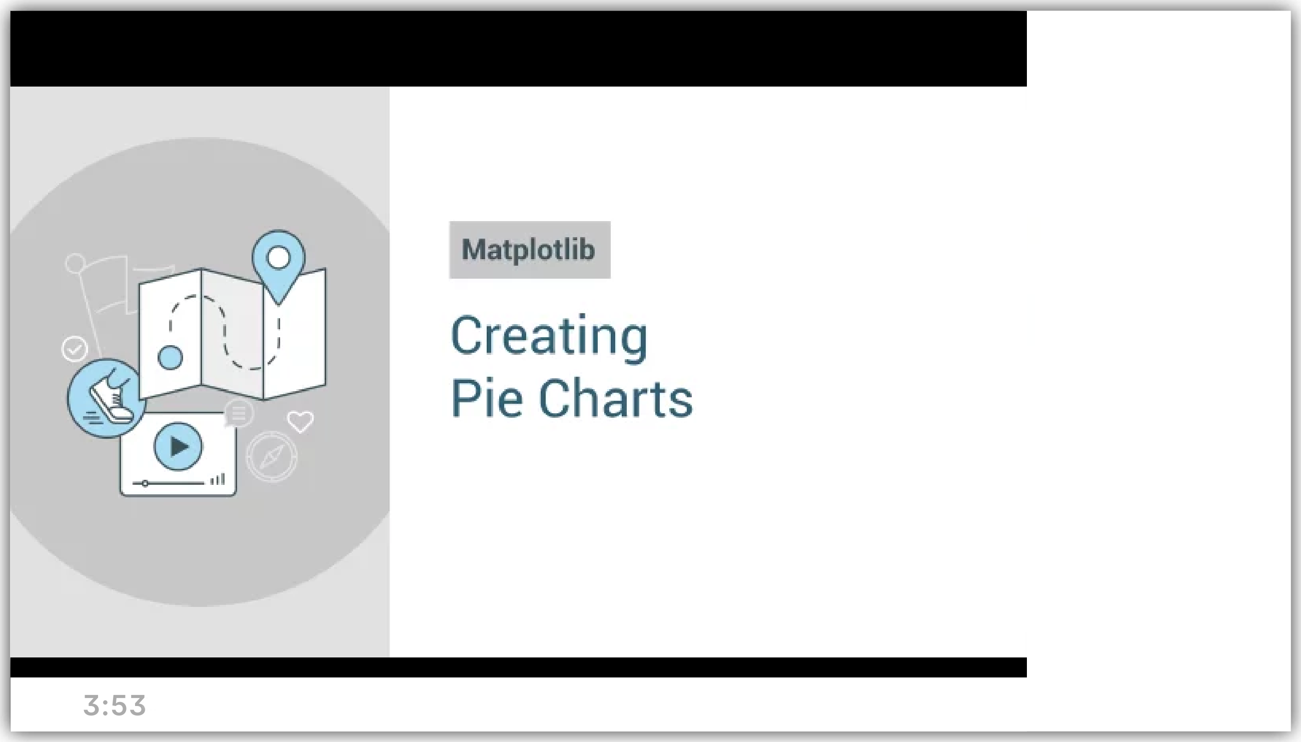
5.1.8

Create Pie Charts

After working all afternoon with Sasha, you've mastered the scatter plot. This presentation is starting to actually sound like it will be fun!

There's one last chart type to explore before you work on adding details to make your graphs stand out—and that's pie charts!

Pie charts aren't as common as line charts, bar charts, or scatter plots. Nonetheless, they provide a really great way to visualize percentages, which you know this dataset may call for. Additionally, they are named after a very delicious dessert, so working on them is always a bit sweeter than working on other charts!



Pie charts allow us to depict percentages of categories as wedges of a pie. One limitation of pie charts is that they can only represent one dataset or category. However, pie charts do have a high visual appeal. We can make them even more visually appealing by choosing vibrant colors and giving each pie wedge a label.

As we've done before, let's learn how to create a pie chart using both the MATLAB method and the object-oriented interface approach.

Create a Pie Chart Using the MATLAB Method

Let's begin with the MATLAB approach. As you've probably noticed, most of these charts follow a similar code structure. Just like we used the `plt.barh()` function to create a horizontal bar chart and the `plt.scatter()` function to create a scatter plot, we'll use the `plt.pie()` function to create a pie chart.

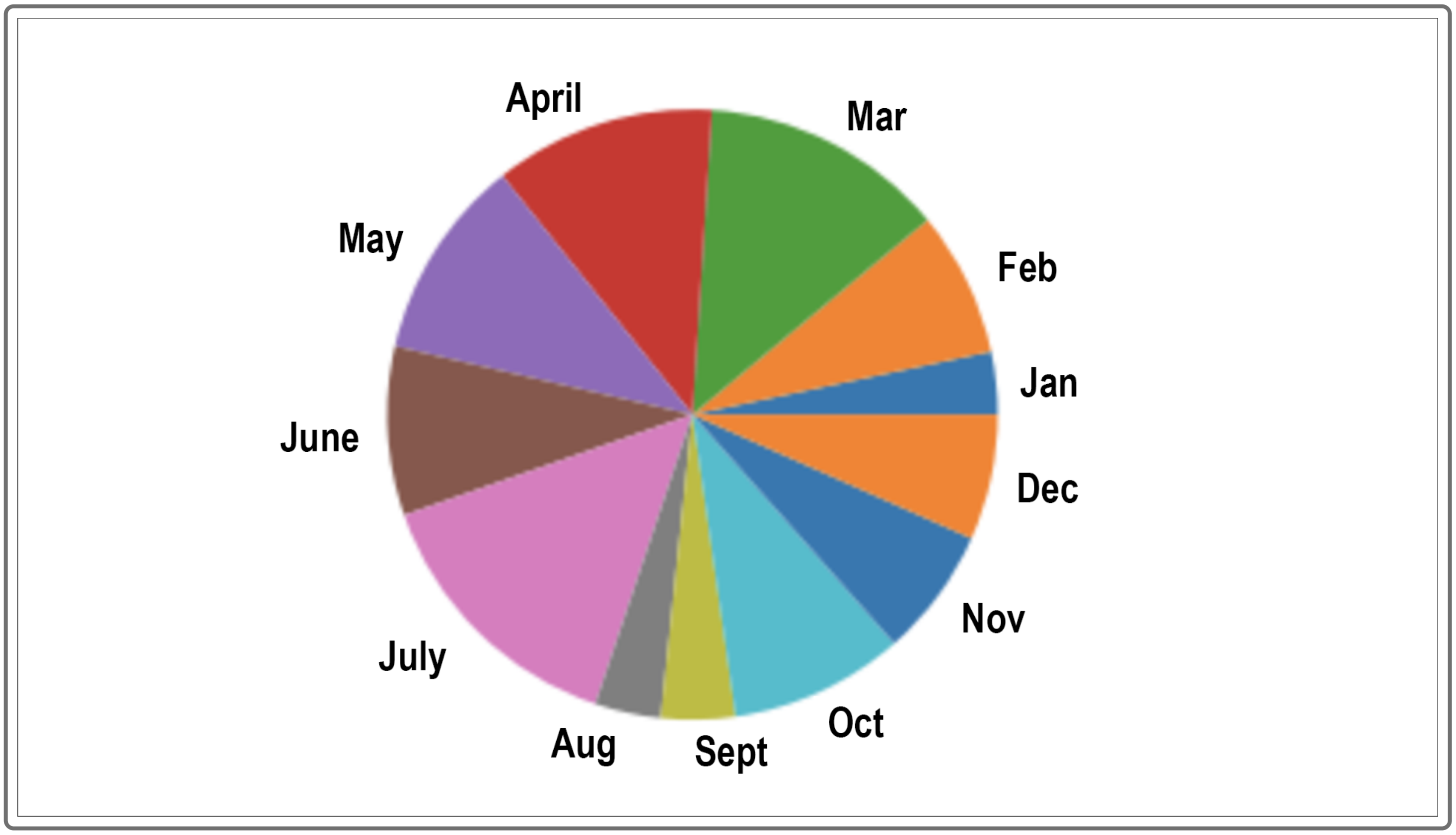
The `plt.pie()` function requires values that are in an array. We can also add labels that are in an array for each pie wedge, but this is optional.

For our first pie chart, we'll use the x_axis data (months) as labels, and the y-axis (total fare for each month) as values.

Add the following code to a new cell:

```
plt.pie(y_axis, labels=x_axis)
plt.show()
```

When you run the cell, the pie chart should look similar to this:



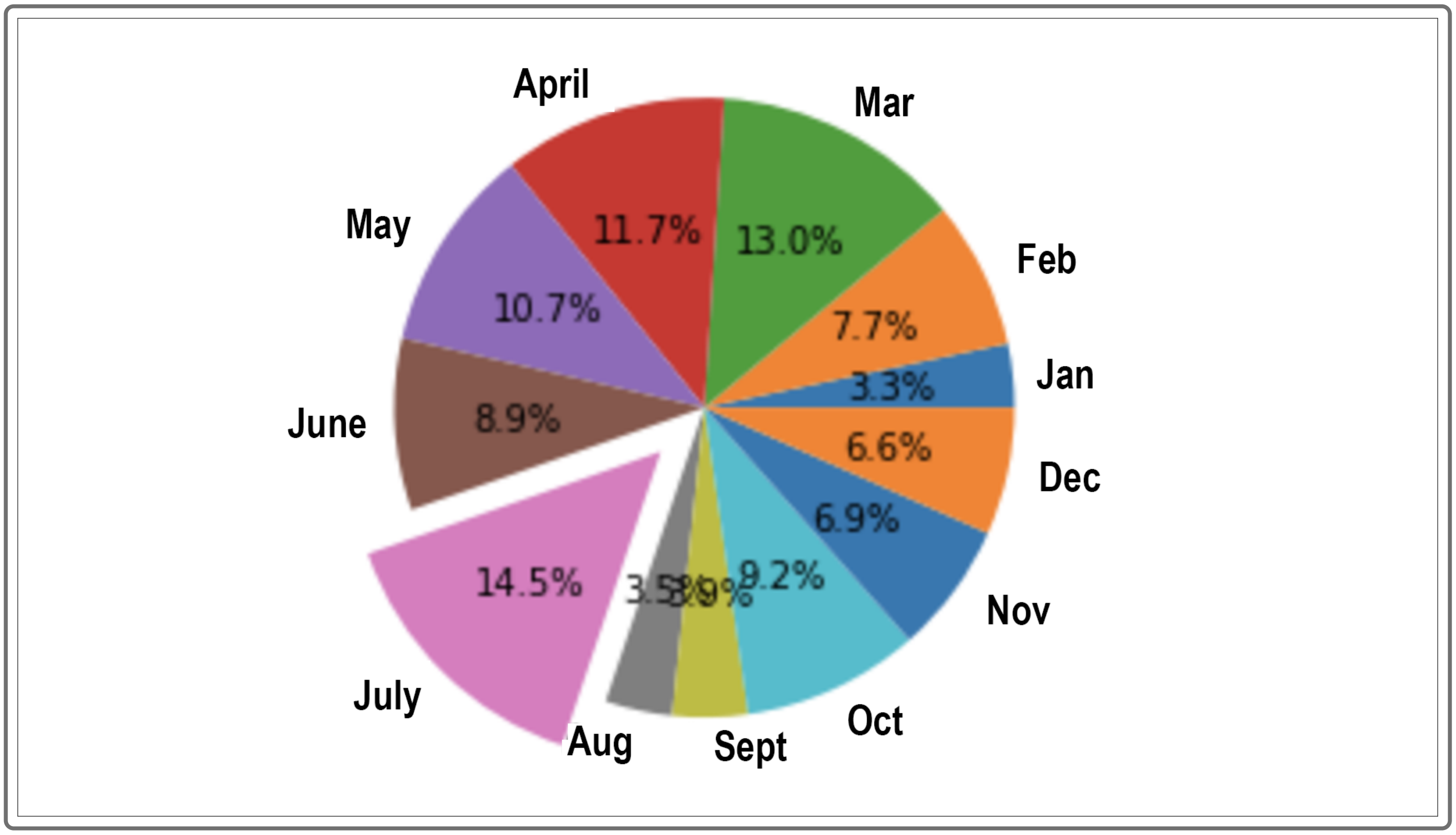
Let's make this chart more impactful by adding some bling. We have a few options with pie charts: we can add percentages to the wedges, add specific colors, and explode or "pop out" one or more wedges.

Let's add percentages for each month and "explode" the largest percentage, which is July, the seventh value in the `x_axis`. The "explode" parameter will offset the indicated wedge by a fraction of the radius, where "0" is zero distance from the center of the pie, and "1" is completely outside the diameter of the pie.

Add the following code to a new cell:

```
explode_values = (0, 0, 0, 0, 0, 0, 0.2, 0, 0, 0, 0, 0)
plt.pie(y_axis, explode=explode_values, labels=x_axis, autopct='%.1f%%')
```

When you run the code, the pie chart will look like this:



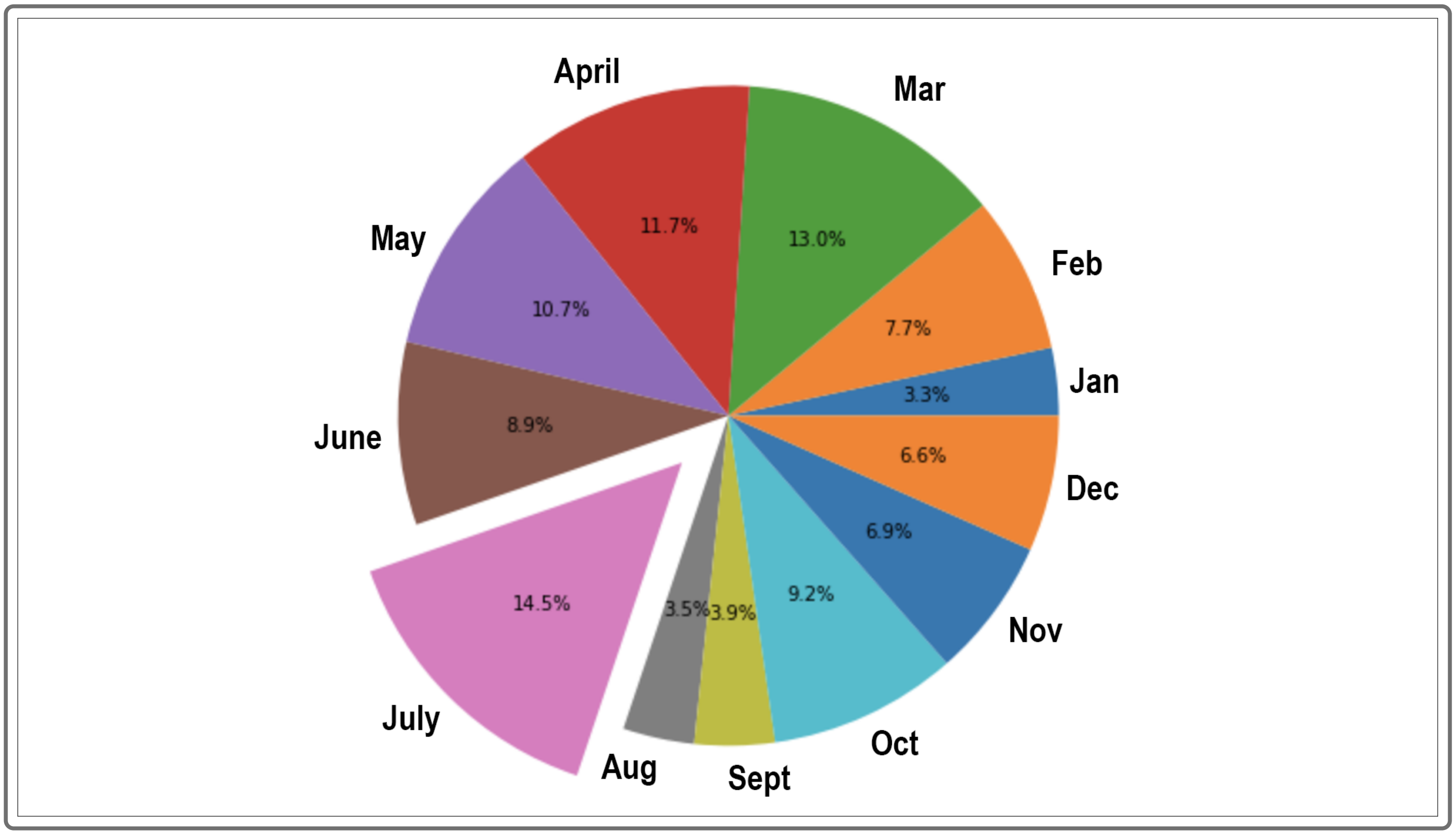
Let's break down what the graphing code is doing:

- To explode a pie wedge, we use array-like data points (like the tuple `explode_values = (0, 0, 0, 0, 0, 0, 0.2, 0, 0, 0, 0, 0)`) for each value in the pie chart.
  - The length of the tuple should be equal to the number of wedges in the pie chart.
  - The value specified to "explode" a wedge is the fraction of the radius from the center of the pie for each wedge.

- To pop out the seventh month, July, we use a decimal-point value of `0.2`.
- To add the percentage of each wedge on the pie chart, we use the `autopct` parameter and provide the format of one decimal place using `.1f%`.
- The `%` before and after the `.1f%` formats the number as a percentage.

The percentages on the pie chart seem a bit crowded. Let's increase the size of the pie chart by using `plt.subplots(figsize=(8, 8))`.

Add this code on the first line in the previous code cell and run the cell again. You should see the following chart:



Now the percentages are not so crowded. But the colors for each wedge may not be the best choice. When we don't specify the colors as parameters (i.e., `"colors="`), then Matplotlib provides default "colors in the currently active cycle," according to the **documentation (https://matplotlib.org/stable/gallery/color/named_colors.html#sphx-glr-gallery-color-named-colors-py)** for the `"colors="` parameter.

Our last pie chart modification will be to change the color of each pie wedge. The choice of colors for your visualizations is a big deal, and frankly, not everyone has an eye for color. Color is also a matter of preference, and what pleases one person may turn off someone else. In general, it is a best practice to choose lighter, brighter, and analogous colors over darker tones.

We're going to change the colors of each pie wedge by using **the CSS colors in the gallery in the Matplotlib documentation (https://matplotlib.org/stable/gallery/color/named_colors.html#sphx-glr-gallery-color-named-colors-py)**.

We'll add the `"colors"` parameter in the `pie()` function with our other parameters, but first we need to assign a `"colors"` variable to an array with the names of our 12 colors.

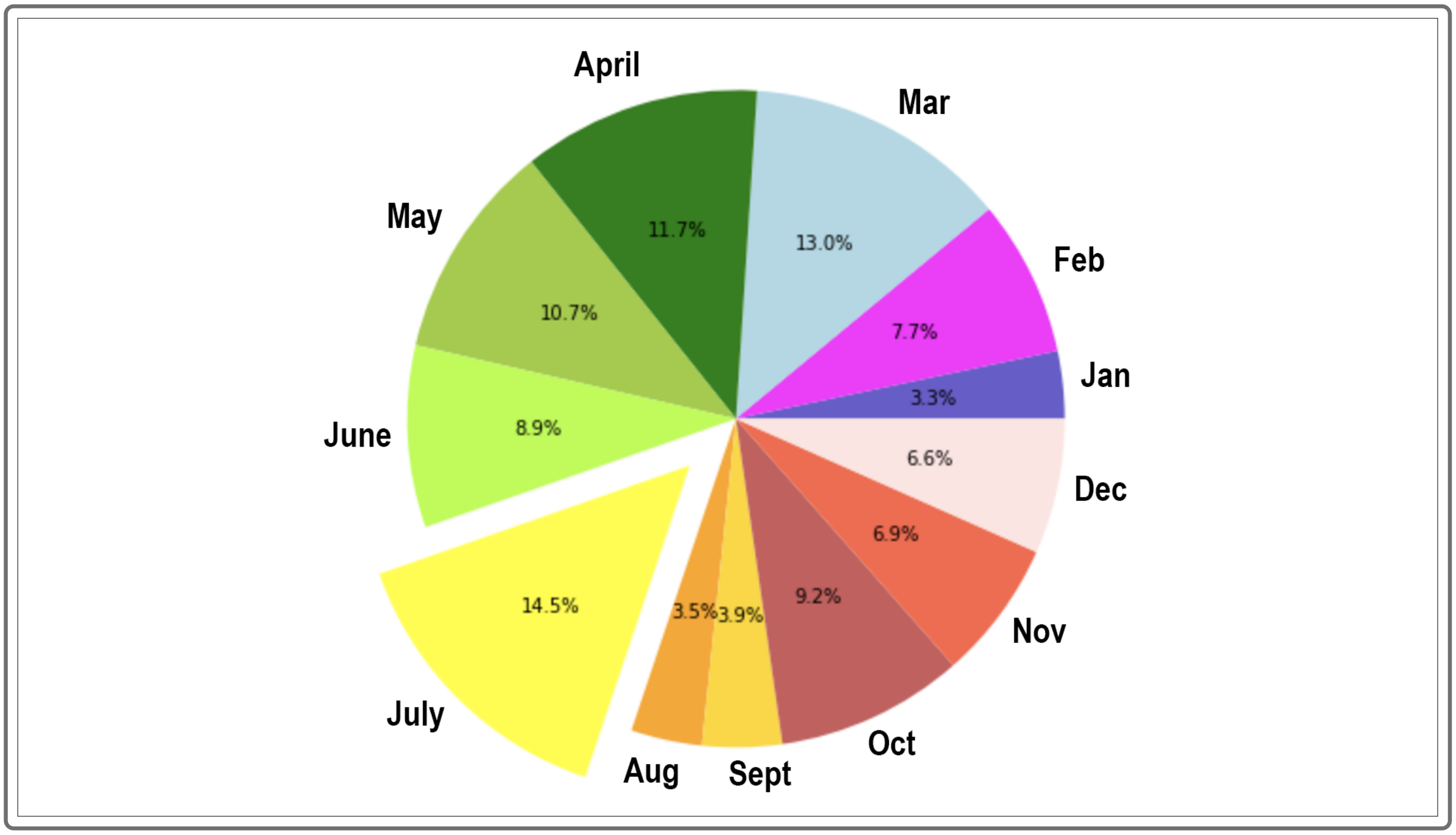
Add the following code to a new cell:

```
# Assign 12 colors, one for each month.
colors = ["slateblue", "magenta", "lightblue", "green", "yellowgreen", "g
explode_values = (0, 0, 0, 0, 0, 0, 0.2, 0, 0, 0, 0, 0)
```

```
plt.subplots(figsize=(8, 8))
plt.pie(y_axis,
    explode=explode_values,
    colors=colors,
    labels=x_axis,
    autopct='%.1f%%')

plt.show()
```

When you run the cell, the pie chart should look like this:



NOTE

For more information, see the [Matplotlib documentation on creating a pie chart using the MATLAB method (https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.pie.html#matplotlib.pyplot.scatter)](https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.pie.html#matplotlib.pyplot.scatter).

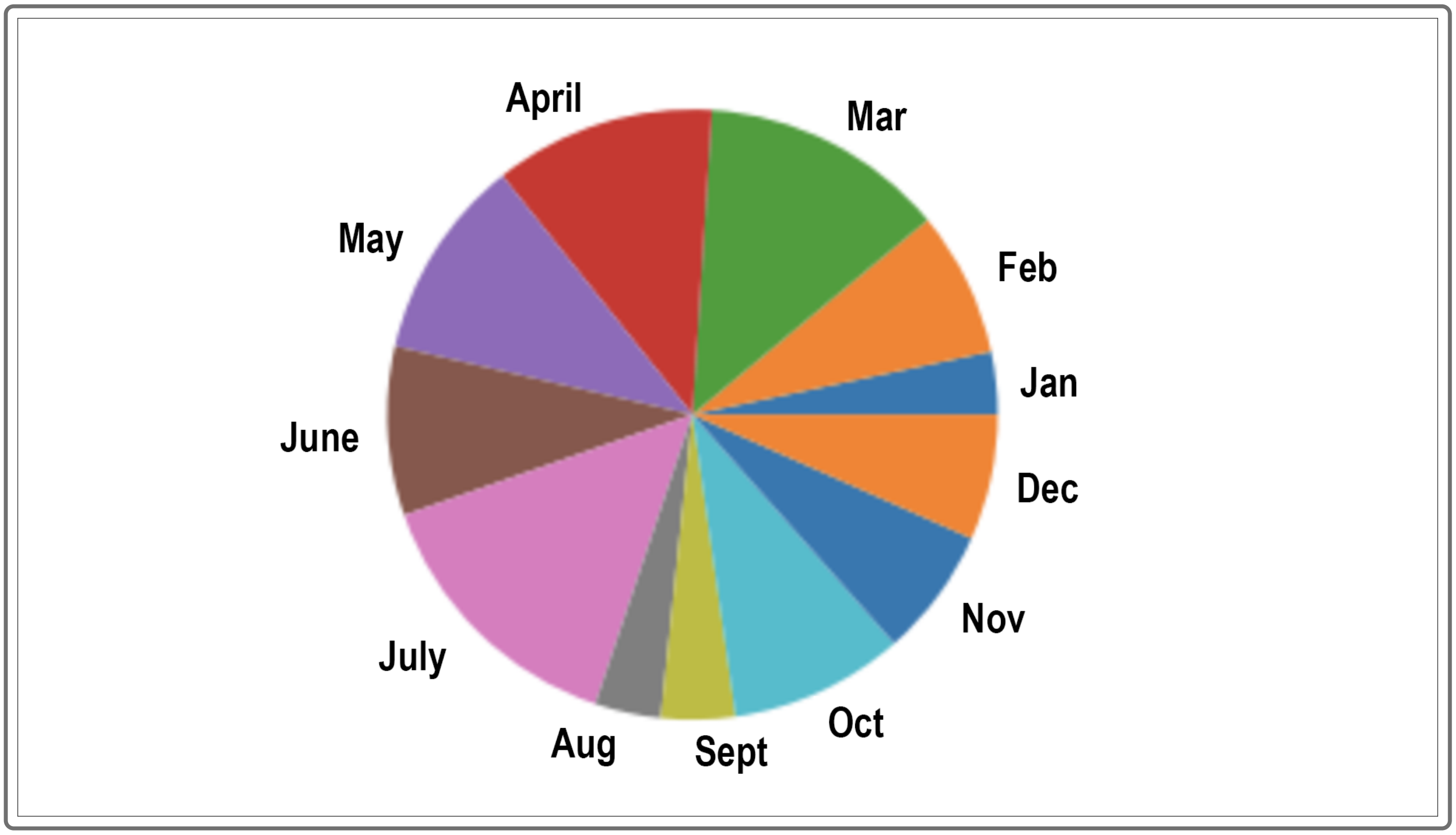
Create a Pie Chart Using the Object-Oriented Interface

Can you guess what we're going to do next? You got it—create the same pie chart using the object-oriented approach.

Add the following to your existing code:

```
fig, ax = plt.subplots()
ax.pie(y_axis,labels=x_axis)
plt.show()
```

When you run this code, you'll see the same graph that we created using the MATLAB approach:

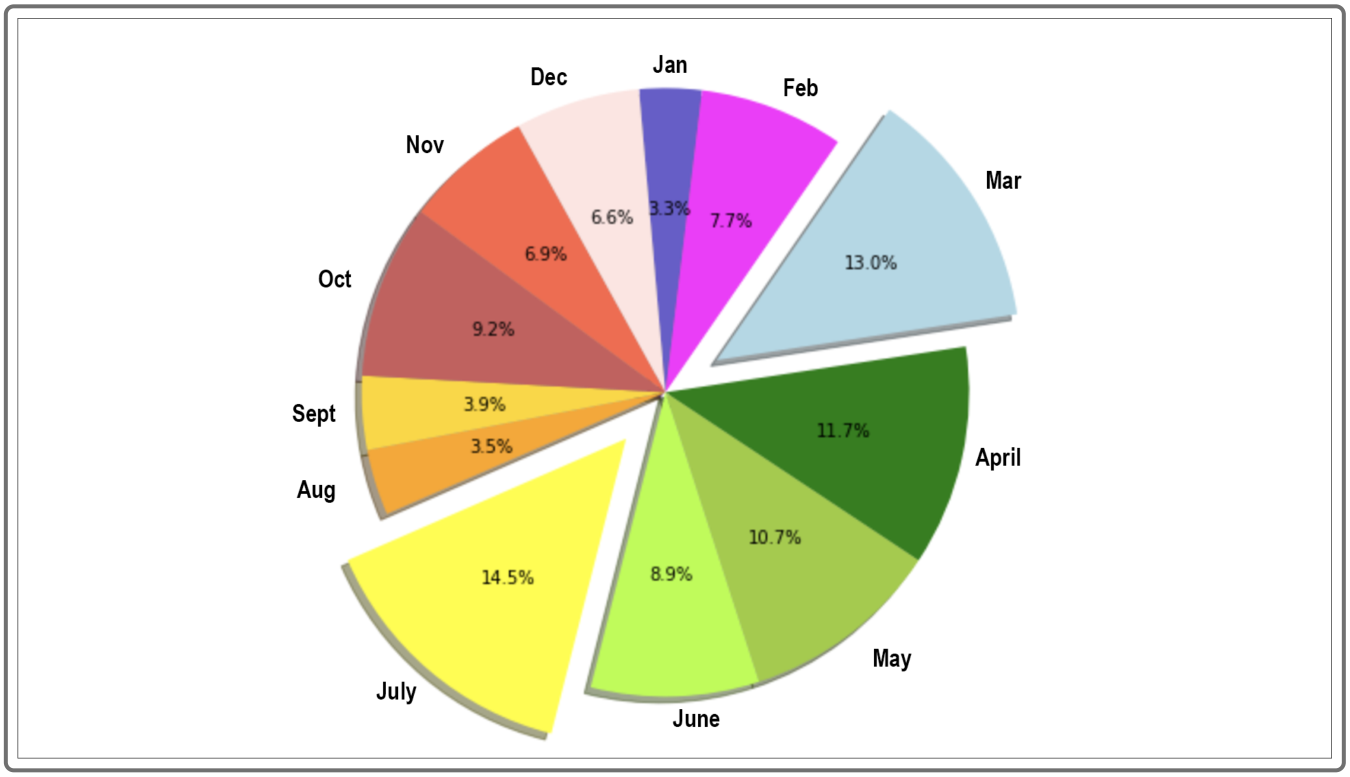


SKILL DRILL

Using the object-oriented approach, make the following changes to the pie chart:

1. Add a percentage to one decimal place to each wedge of the pie.
2. Increase the figure size to 8x8.

3. Explode the two highest percentage months.
4. Add a shadow.
5. Add a start angle so that January is at the top.
6. Reverse the order so that the month order is in a clockwise direction.
7. Add new colors of your own choosing or use the colors from the previous pie chart.

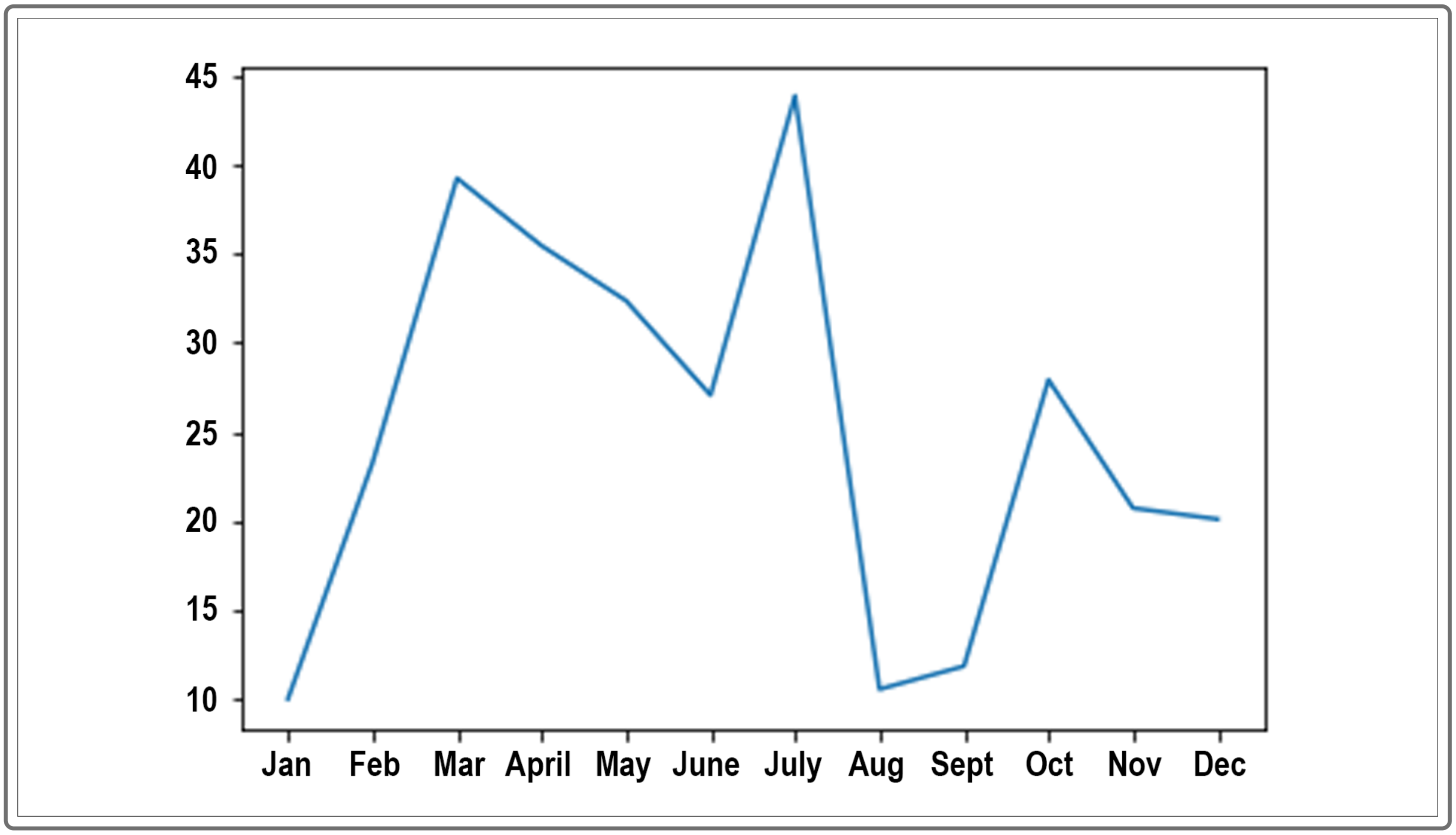


Which Chart Is Best?

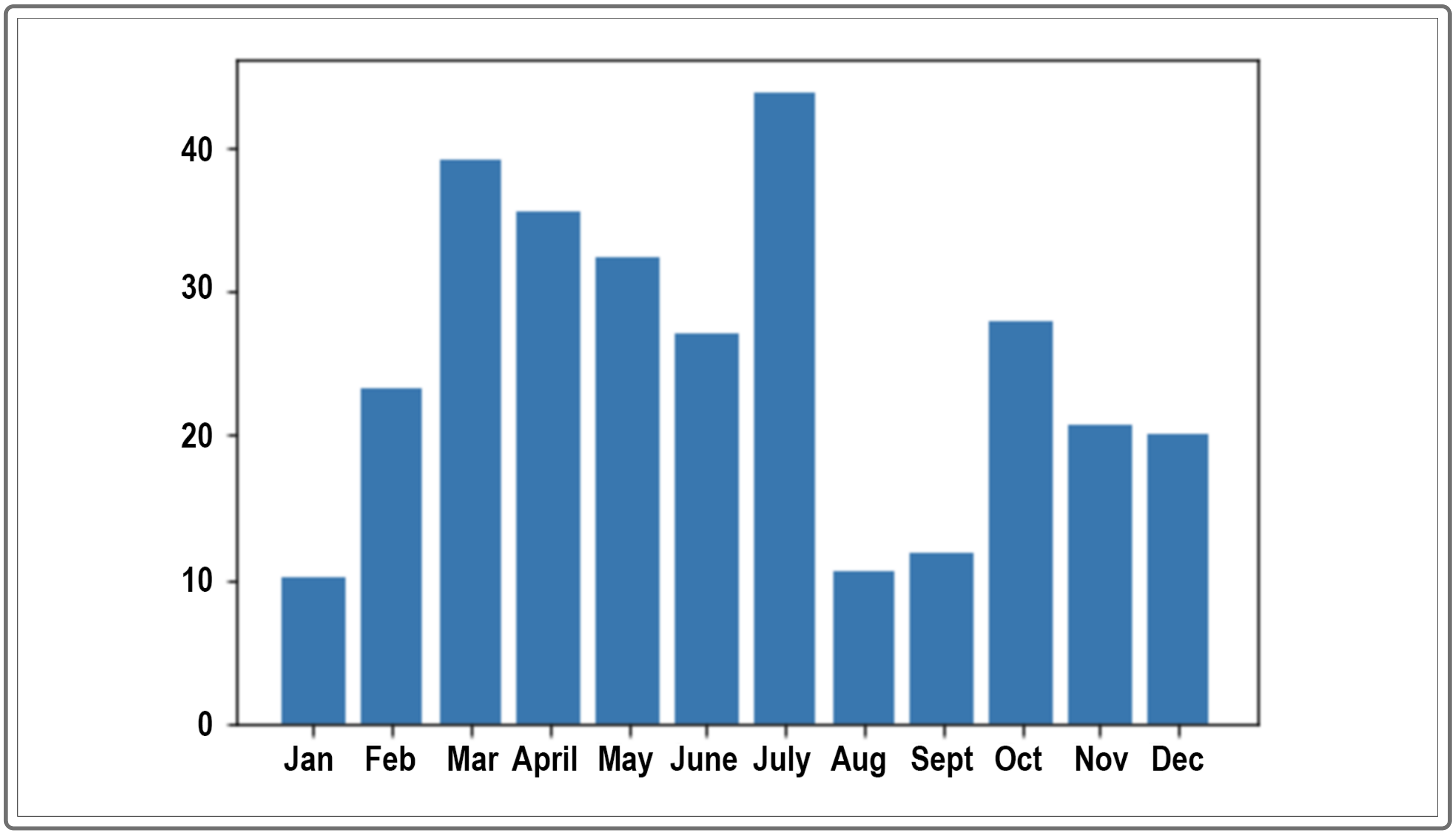
Great job! You now know how to create line charts, bar charts, scatter plots, bubble charts, and pie charts. Before moving on to adding extra details to these charts, let's reflect. Which chart do you think is best for graphing the ride-sharing data? Your CEO will most likely ask why you chose the charts you did, so you'd better be prepared to answer.

Let's review the pie chart first since it's fresh in our minds. In this chart, we can see that some months contributed a very small amount of the overall fare collected for the year. We can also see that March through July are the most lucrative for ride-sharing. How does this compare to our other charts?

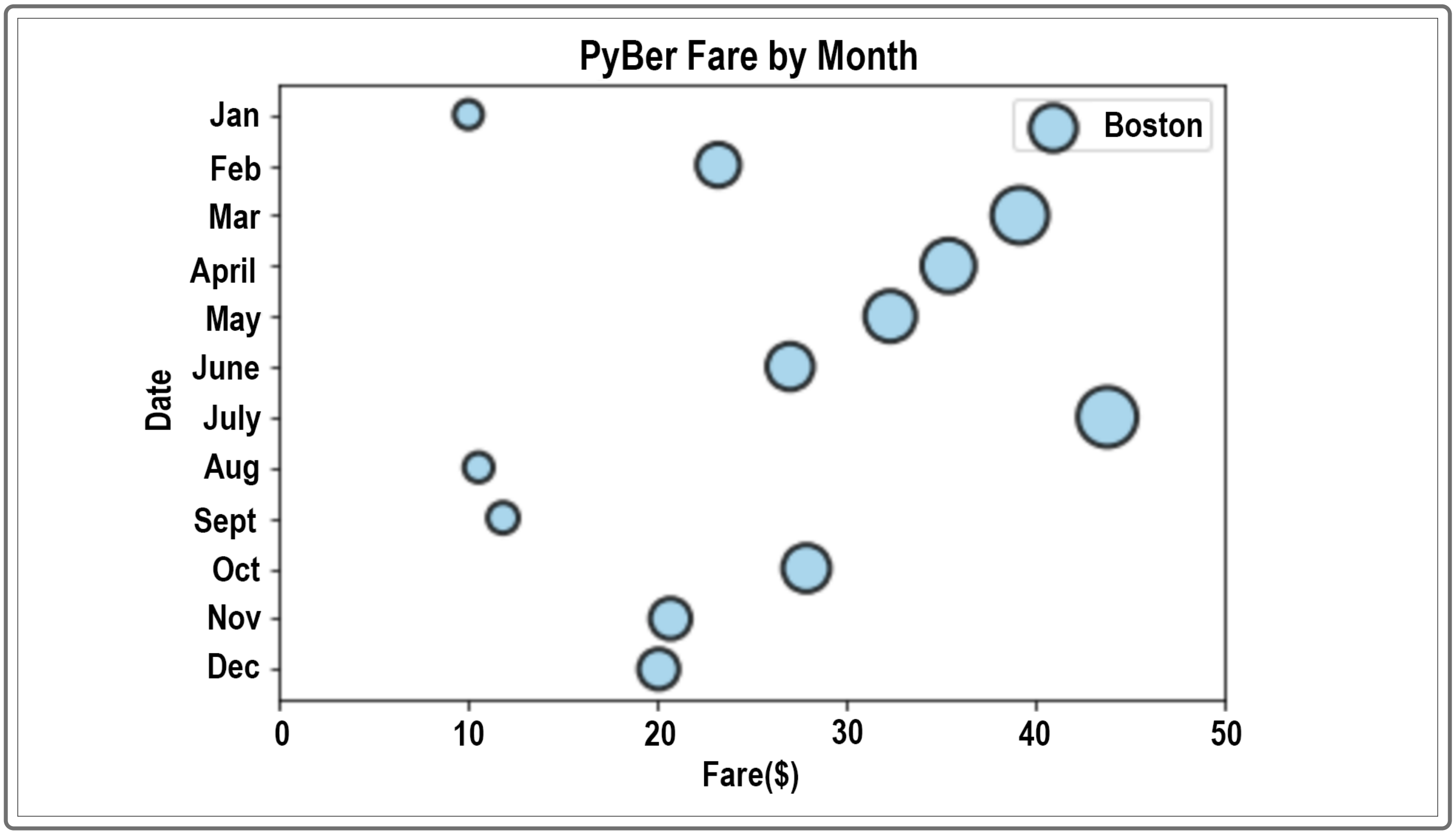
The line chart conveyed the same information: we can see that March through July are the most lucrative for ride-sharing, with a small peak in October.



The vertical and horizontal bar charts also conveyed the same information:



The scatter plot didn't do a great job of getting that information across, but the bubble chart we created in the Skill Drill did a decent job of conveying the same information as the line, bar, and pie charts, because the bubbles increased in diameter as the average fare increased. Still, the bubble chart may not be a good chart to use for this particular dataset, because there are only two data points for this scatter plot: the months on the y-axis and the fare amount on the x-axis. If we had a third parameter that could be the size of the marker, like the number of riders for each month, then using the scatter plot might be a good choice.



As a data analyst, creating visualizations is part of the exploratory data analysis process. Deciding which chart represents the data the best is trial and error. As you gain more experience at creating charts, you will know which charts will best represent the data.

Now that we have learned how to create a variety of charts, let's learn how to annotate charts by adding error bars and changing the major and minor ticks on the axes.

NOTE

For more information, see the **Matplotlib documentation on creating a pie chart using the object-oriented interface method (https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.pie.html)**.