5.1.5 Create Bar Charts Using the MATLAB Approach

Great job! You've completed the first graph in any data analyst's tool belt: the line chart. And you created it using two different methods!

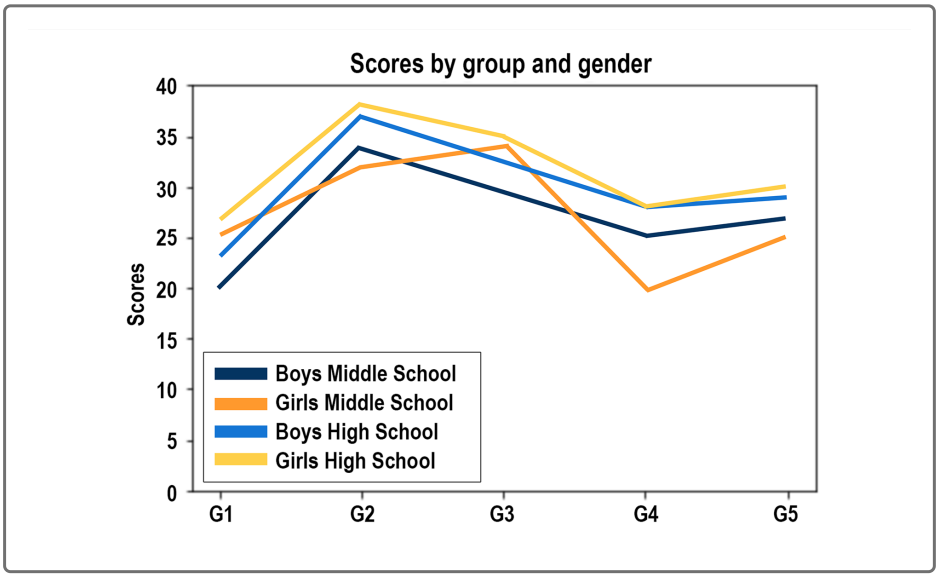
Of course, this was just step one—and you're not going to stop at step one! Both Omar and V. Isualize will want to see visualizations that really illuminate the data.

Next, you'll create and annotate vertical and horizontal bar charts. Bar charts are particularly helpful when comparing datasets over time or when your line chart starts to look cluttered because it has too many lines. So stretch, refill your coffee, and get back to coding!

A bar chart tells a different visual story than a line chart. There are many benefits to using a bar chart. They're good at displaying discrete data (i.e., data based on counts) in distinct columns. They also tend to be visually strong. For example, adding color to the bars can make a chart really shine for your audience.

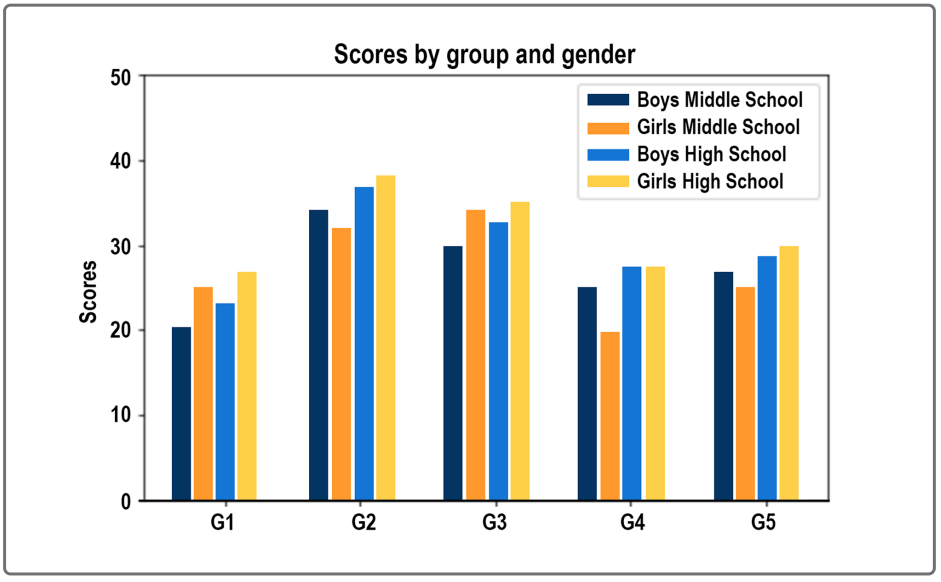
You can use a bar chart to clearly compare two or more datasets, either by displaying them side by side or by using a stacked bar chart. Bar charts can provide more visual appeal and information compared to having multiple lines on the same graph, which can mask the underlying information.

For instance, consider the following line chart, which has lines for four different groups:



This chart might be good for a quick glance at the data, but it's not one that you'd want to show stakeholders. The lines are too close together, two lines are almost on top of each other, and it's hard to tell which group is which from the line graph color. You could make the line thickness darker, but some lines might overlap in that case.

A better way to show this data would be a bar chart, where we can clearly see the differences among the groups and the bar color of each group:



Let's dive into learning how to create bar charts with Matplotlib using the MATLAB method. We'll use the same ride-sharing data as before to create a bar chart. Here's that data:

```
# Set the x-axis to a list of strings for each month.
x_axis = ["Jan", "Feb", "Mar", "April", "May", "June", "July", "Aug", "Sept"

# Set the y-axis to a list of floats as the total fare in US dollars accumul
y_axis = [10.02, 23.24, 39.20, 35.42, 32.34, 27.04, 43.82, 10.56, 11.85, 27.
```

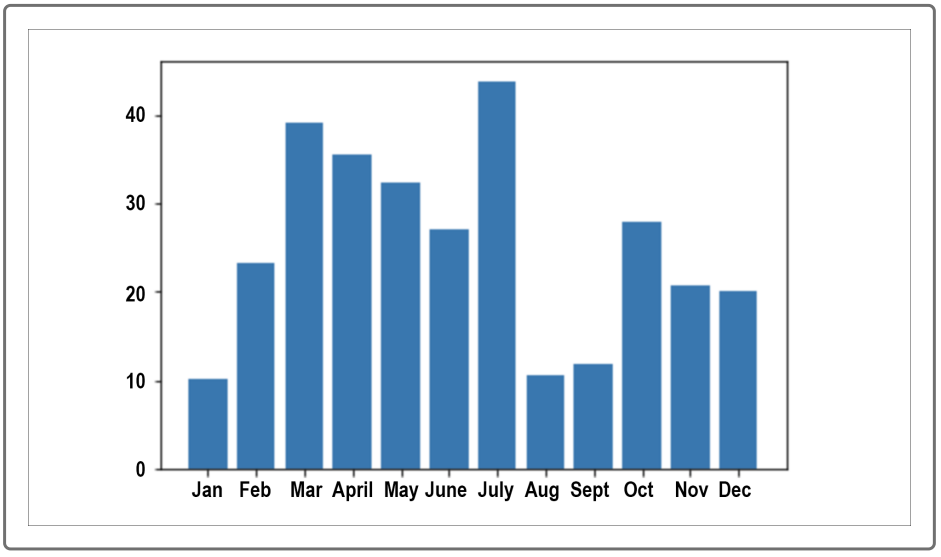
Create a Vertical Bar Chart

If you're graphing time-series data or ordinal values (e.g., age ranges, salary ranges, or groups), a vertical bar chart is a good choice. Let's learn how to do that.

To create a vertical bar chart, use the `plt.bar()` function. In the `matplotlib_practice.ipynb` file, add the following x and y data arguments inside parentheses:

```
# Create the plot
plt.bar(x_axis, y_axis)
```

When you run this cell, you'll see the ride-sharing data as a bar graph, as shown here:

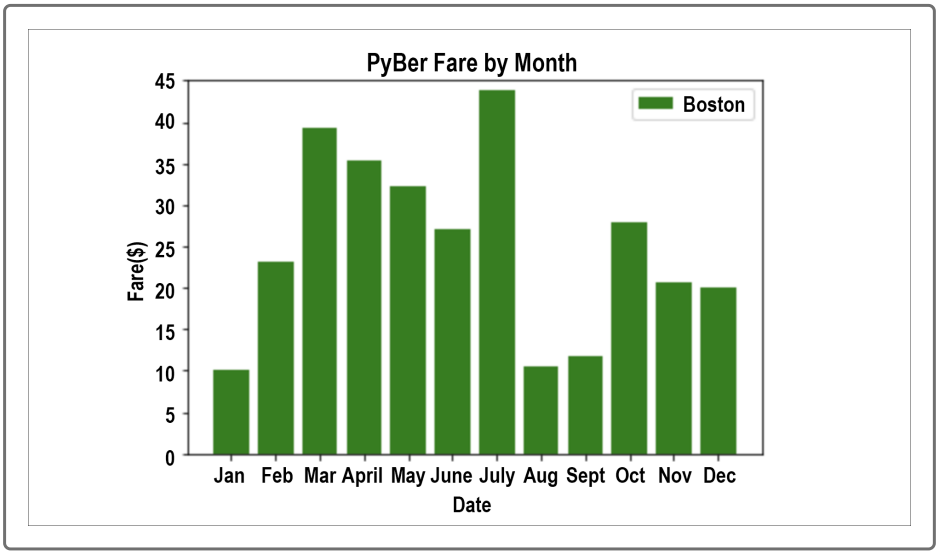


We can annotate the bar chart like we annotated the line chart. However, we won't need the `marker` or `linewidth` attributes.

To annotate the bar chart, add the following code to a new cell:

```
# Create the plot.
plt.bar(x_axis, y_axis, color="green", label='Boston')
# Create labels for the x and y axes.
plt.xlabel("Date")
plt.ylabel("Fare($)")
# Create a title.
plt.title("PyBer Fare by Month")
# Add the legend.
plt.legend()
```

After running the cell, our graph should look like this:



NOTE

> For more information, see the **Matplotlib documentation on creating bar charts using the MATLAB method (https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.bar.html)** .

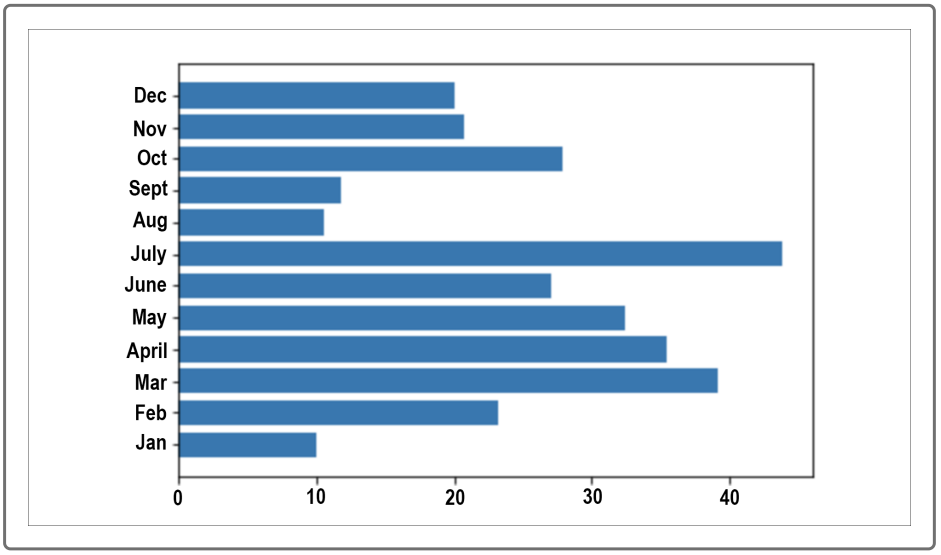
Create a Horizontal Bar Chart

If you're graphing nominal variables (e.g., favorite music groups among teens, number of votes by county or state) where you sort the data from greatest to least or least to greatest, it's best to use a horizontal bar chart.

To create a horizontal chart, we use the `plt.barh()` function:

```
# Create the plot
plt.barh(x_axis, y_axis)
```

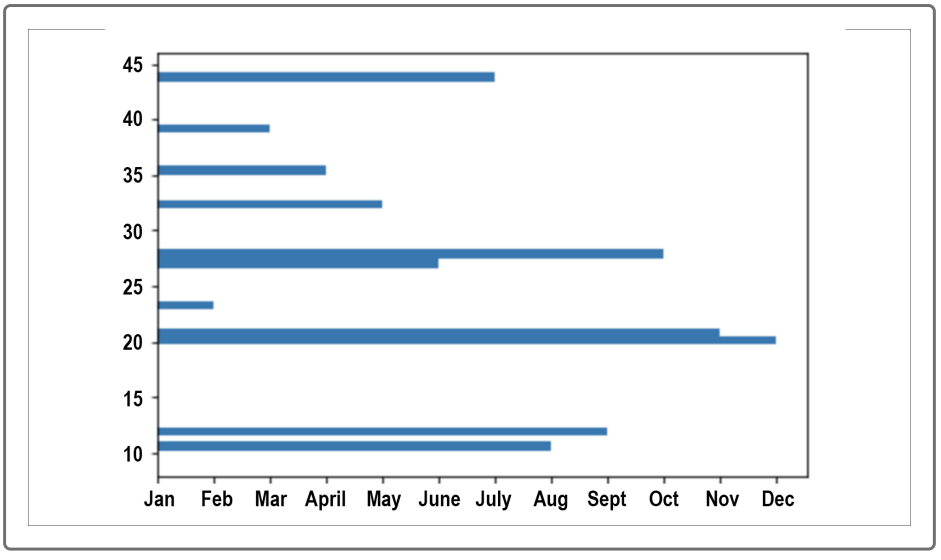
When we run this cell, our bar chart looks like this:



If you want the data on opposite axes, switch the arguments in the `barh()` function and run the cell again:

```
# Create the plot
plt.barh(y_axis, x_axis)
```

When we run this cell, our bar chart looks like this:



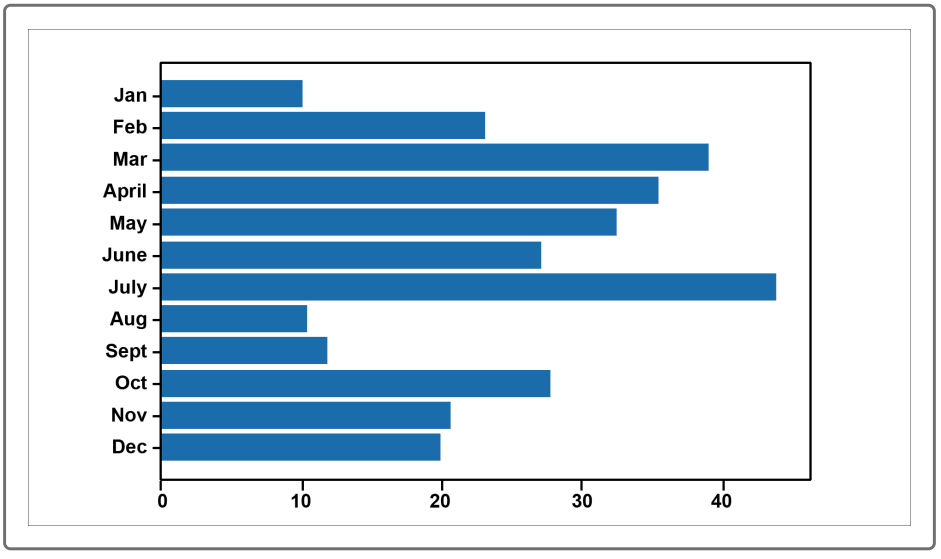
The horizontal bar chart is not a good way to represent the data because the bar should convey an *amount* on the x-axis—not a date. Trying to determine each month's average fare is more challenging in this chart compared to the previous horizontal chart, but it's good to know that we can switch data to graph on the opposite axes if we need to.

We should invert the y-axis of the previous chart to have "January" at the top and "December" at the bottom. Can you think of the reason for this? Right—because we don't want to show the CEO ride-sharing data with the months in the wrong order!

To invert the y-axis to have the months in ascending order, use the `gca()` method. The `gca()` method means "get current axes." We can chain the `gca()` method to the `invert_yaxis()` method by using `gca().invert_yaxis()`, as shown here:

```
# Create the plot.
plt.barh(x_axis, y_axis)
plt.gca().invert_yaxis()
```

When we run this cell, our bar chart should look like this:



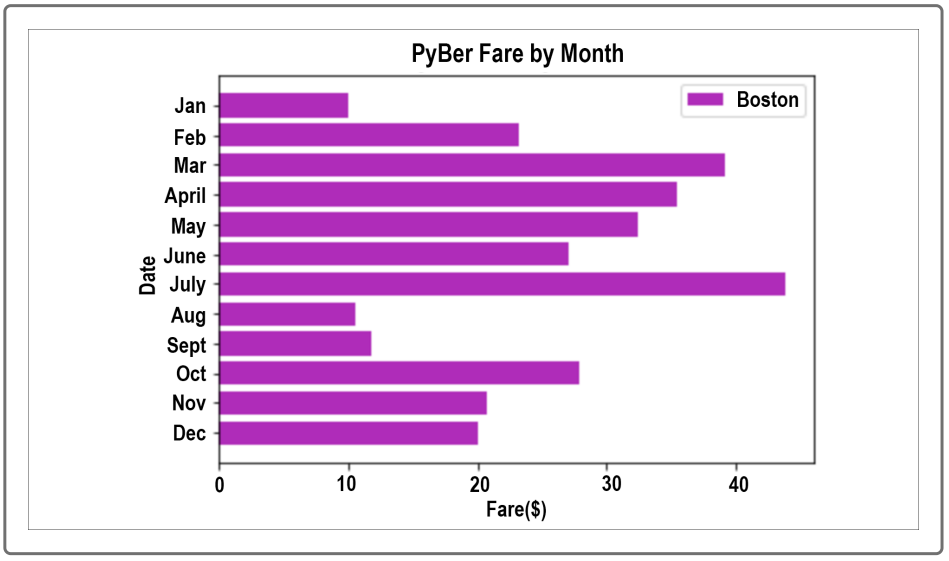
We can also annotate a horizontal bar chart as we did with a vertical bar chart.

SKILL DRILL

Using the Matplotlib MATLAB plotting approach, make the following changes:

1. Change the bar colors to magenta.
2. Add a legend for the city of Boston.
3. Add a title and axes labels.
4. Invert the y-axis so that January is at the top.

When you're done, your chart should look similar to this:



NOTE

> For more information, see the **Matplotlib documentation on creating horizontal bar charts using the MATLAB method (https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.barh.html#matplotlib.pyplot.barh)** .