5.1.9 Chart Extras

You've made a line chart, vertical and horizontal bar charts, scatter plots, bubble charts, and pie charts. But you're not going to stop there—oh, no—you're going to make the most impressive presentation your CEO has ever seen, and that means having a chart ready to show on the spot no matter the question she asks or edit she requests.

To do that, you need to think critically about the visualizations you've made so far. What if she wants to see error bars? Maybe you should add minor x- or y-ticks, or change the scale for the major ticks. Whatever the additional details she wants, you're going to be ready. So you roll up your sleeves and get back to coding!

With Matplotlib, we can add a variety of extras to our charts. Let's add some more enhancements to our bag of tricks, like error bars, minor x- and y-ticks, and custom increments for the major x- and y-ticks.

Add Error Bars to a Line Chart

Adding error bars can show either the standard deviation, standard error, confidence intervals, or minimum and maximum values of a dataset. When added to a chart, they can visually show the variability of the plotted data. By looking at the error bars, one can infer the significance of the data.

To create a line chart with error bars using the MATLAB approach, we use the `errorbar()` function. We also add the x and y data parameters and the `yerr=` parameter inside parentheses, as shown here:

```
plt.errorbar(x, y, yerr=<value>)
```

To the ride-sharing line chart, let's add the standard deviation of the fare, or y-axis, as error bars.

Create a new Jupyter Notebook file. Add the following code blocks in order to a new cell and run each cell:

```
%matplotlib inline
```

```
# Import dependencies.
import matplotlib.pyplot as plt
import statistics
```

We are importing a new dependency, the statistics module, which comes with the Anaconda installation. We're going to use the statistics module so that we can add the y-error bars to line and bar charts. With the statistics module, we can calculate the standard deviation of the fare amount.

REWIND

The standard deviation is a measure of the amount of variation, or spread, of a set of values from the mean.

```
# 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.
```

Next, we'll calculate the standard deviation of the `y_axis` values using the statistics module.

Add the following code to a new cell:

```
# Get the standard deviation of the values in the y-axis.
stdev = statistics.stdev(y_axis)
stdev
```

When you run the cell, it will compute the standard deviation and output that number, as shown in the following image:

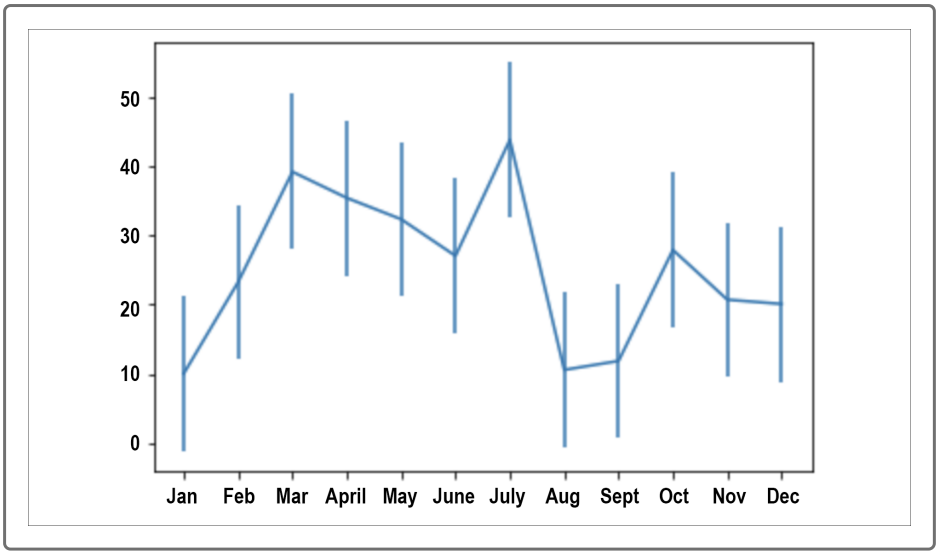
```
# Get the standard deviation of the values in the y_axis.
stdev = statistics.stdev(y_axis)
stdev
```

```
11.208367917035753
```

Next, add the `x_axis`, `y_axis`, and `yerr=stdev` to the `errorbar` function:

```
plt.errorbar(x_axis, y_axis, yerr=stdev)
```

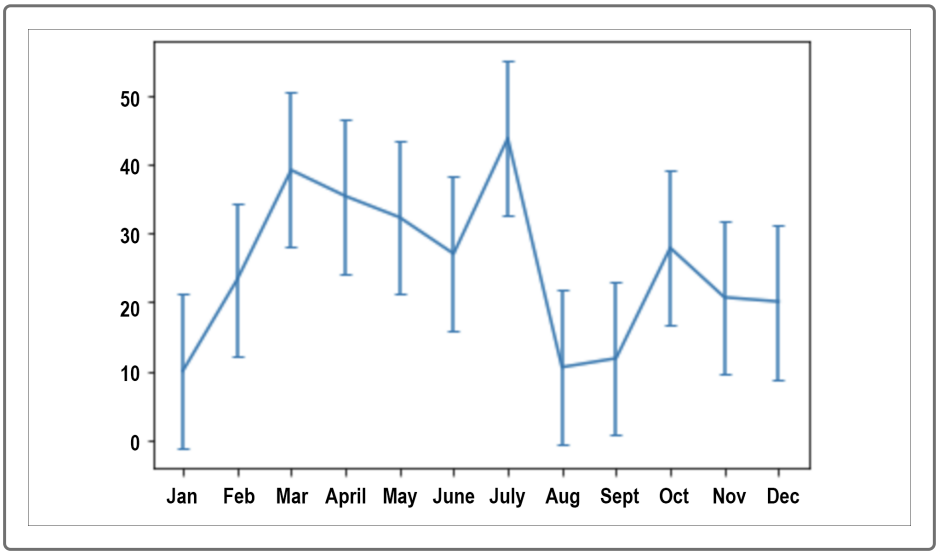
When you run the cell, you'll see the following line chart with error bars for the fare amounts:



We can make the error bars look more professional by adding a "cap" to the error bars that's equal to a scalar value. To do this, we use the `capsize=` parameter. Let's add `capsize=3` to the error bars. Add the following code:

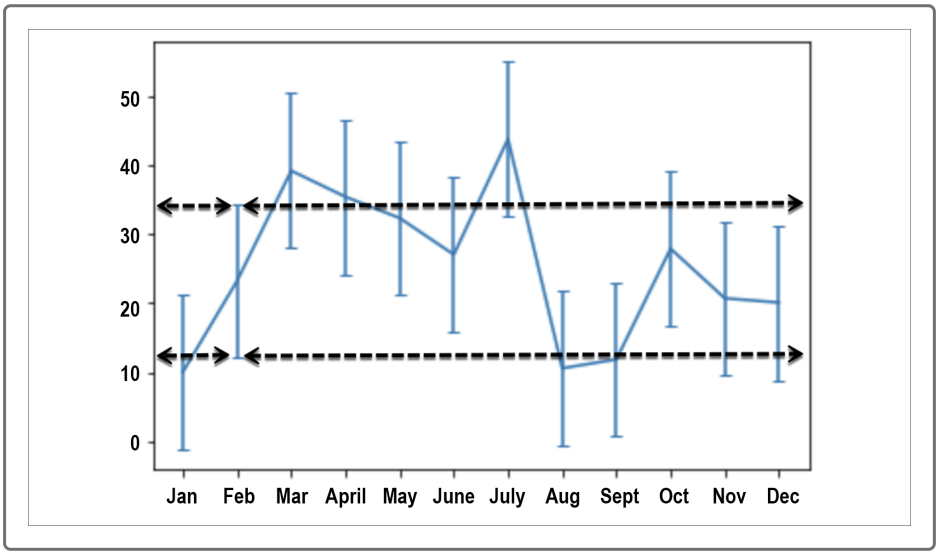
```
plt.errorbar(x_axis, y_axis, yerr=stdev, capsize=3)
```

When you run the code, you'll see the same chart but with caps on the error bars:



Now that we've added error bars to the line chart, we can begin to make assumptions about the plotted data.

For instance, in the following figure, we have added dotted lines for the one standard deviation, 11.2, above the mean and one standard deviation below the mean for February.

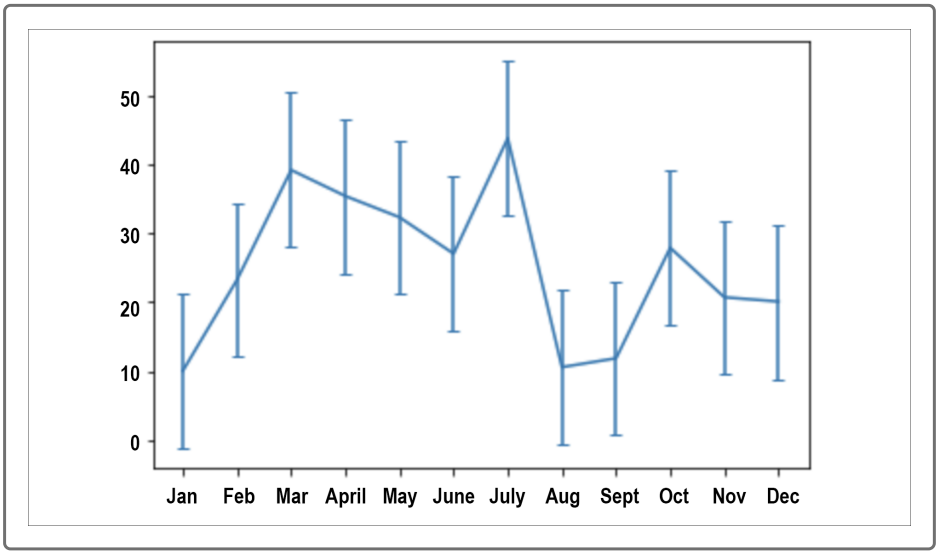


Here, the standard deviations above and below the mean overlap with the error bars from some of the other months. When comparing all the months like this, we can assume that there is very little variability in the fares from month to month. And overall, there is a wide range in the average fare price for every month.

Now let's add error bars and a capsize using the object-oriented approach. Add the following to a new cell.

```
fig, ax = plt.subplots()
ax.errorbar(x_axis, y_axis, yerr=stdev, capsize=3)
plt.show()
```

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



NOTE

> For more information about adding error bars to line charts using the MATLAB and the object-oriented methods, please refer to the following documentation:
>
> **Matplotlib documentation on matplotlib.pyplot.errorbar (https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.errorbar.html)**
>
> **Matplotlib documentation on axes.Axes.errorbar (https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.axes.Axes.errorbar.html)**

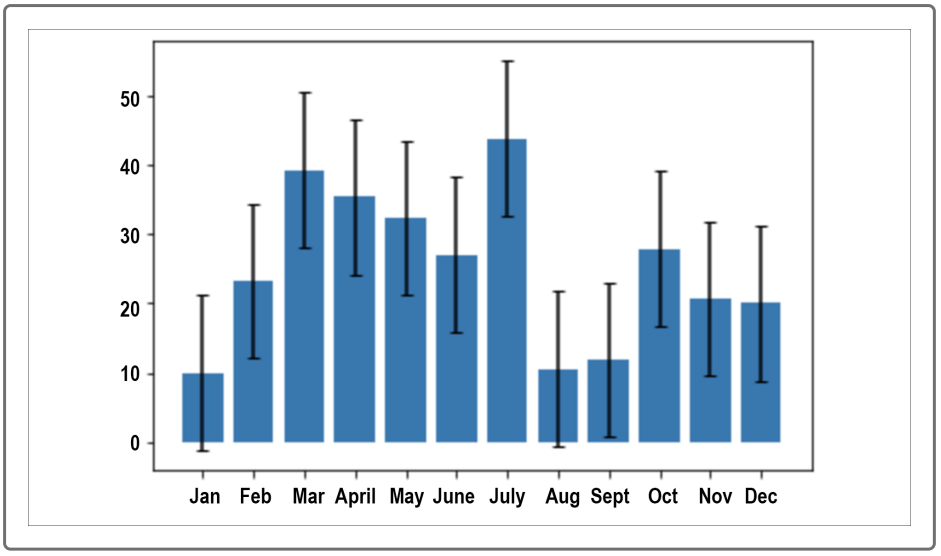
Add Error Bars to a Bar Chart

To add error bars and caps to a bar chart using the MATLAB approach, we can use the `yerr=` and the `capsize=` parameters with the `plt.bar()` function.

Add the following code to a new cell:

```
plt.bar(x_axis, y_axis, yerr=stdev, capsize=3)
```

When you run the cell, the bar chart with error bars will look like this:

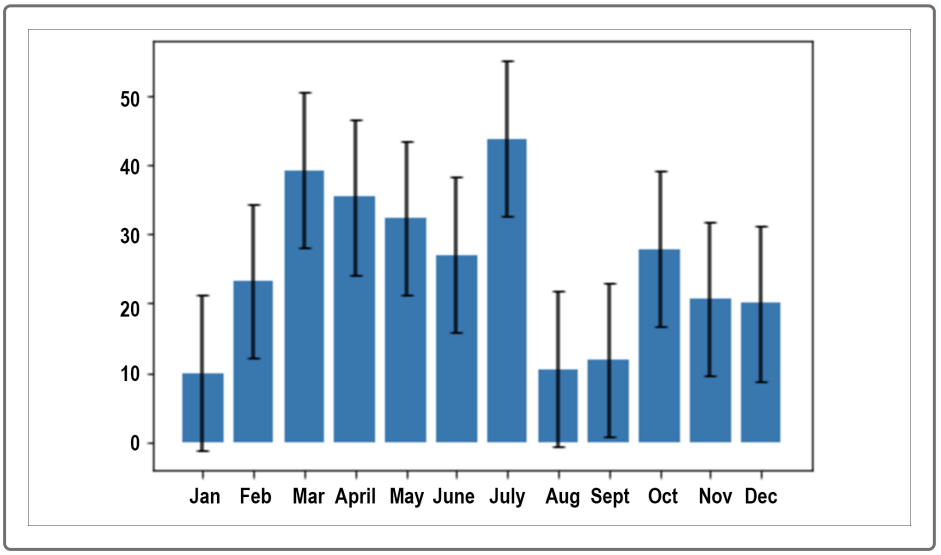


Now let's create the same graph using the object-oriented approach.

Add the following code to a new cell:

```
fig, ax = plt.subplots()
ax.bar(x_axis, y_axis, yerr=stdev, capsize=3)
plt.show()
```

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



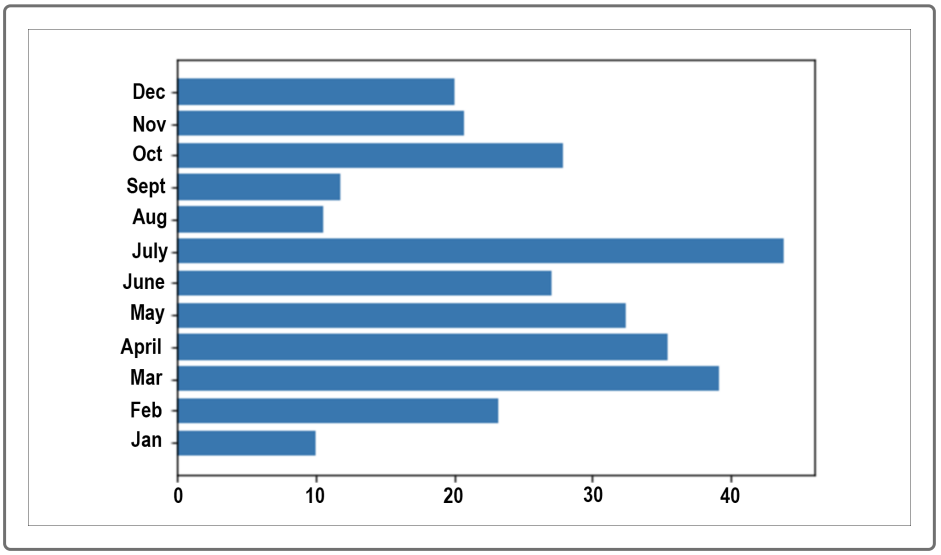
NOTE

> For more information about adding error bars to bar charts using the MATLAB and the object-oriented methods, please see the following documentation:
>
> **Matplotlib documentation on matplotlib.pyplot.bar (https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.bar.html)**
>
> **Matplotlib documentation on axes.Axes.bar (https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.axes.Axes.bar.html)**

Change the Major Ticks

When we graphed our horizontal bar chart with the fare on the x-axis, Matplotlib automatically added the ticks. We can customize our charts so that the major ticks on an axis are more spread out or closer together. Which one you choose depends on what the data looks like on a graph.

Take a look at the horizontal bar chart. When it was plotted, the major ticks on the x-axis were spaced every $10. This spacing makes it difficult to estimate the average fares for some months. Let's see if we can improve that.

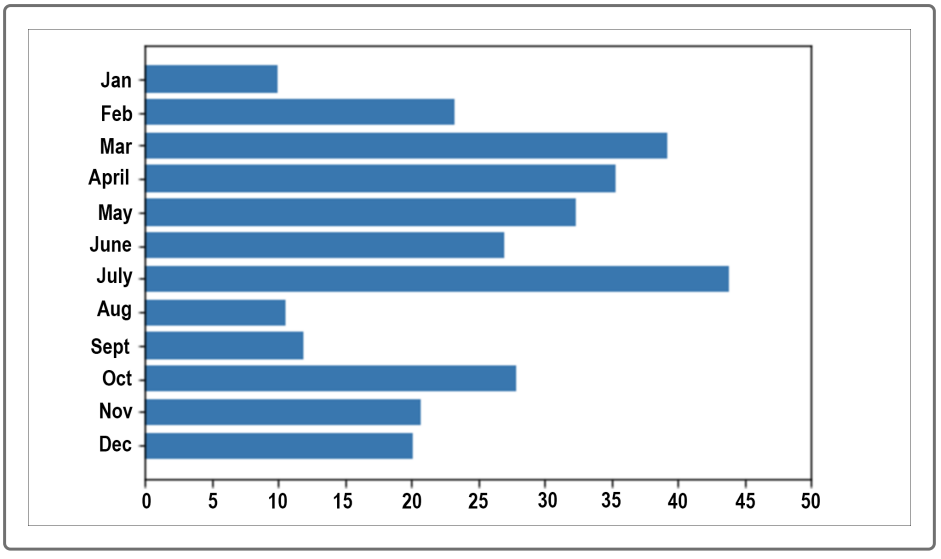


To adjust the major x-axis ticks on a horizontal bar chart, we use the `xticks()` function.

Add the following code to a new cell:

```
import numpy as np
plt.barh(x_axis, y_axis)
plt.xticks(np.arange(0, 51, step=5.0))
plt.gca().invert_yaxis()
```

When you run the cell, the output will be a horizontal bar chart with the major ticks at every $5 increment:



As you can see, setting the major ticks to $5 increments makes it easier to estimate the average fare for some of the months.

Did you notice that we added some code that you hadn't seen yet? Let's break it down:

- We imported the `numpy` module. NumPy is a numerical mathematics library that can be used to create arrays or matrices of numbers.
- We created a horizontal bar graph using the `barh()` function and passed the x- and y-axis data inside the parentheses.
- To create the $5 increments for the x-ticks, we use the `numpy` module to create an array of numbers using the `arange()` function. We defined the lower and upper limit of the x-axis and the increment frequency with the code: `np.arange(0, 51, step=5.0)`.
- Finally, we use `gca()` method to get the current axes and invert them.

Let's drill down a little more into `step=`. If we were to import the `numpy` module and add the code snippet—`np.arange(0, 51, step=5.0)`—in a new cell like this:

```
import numpy as np
np.arange(0, 51, step=5.0)
```

The output of the `arange()` function would be in increments of 5, from zero to 50. If we didn't include `step=`, we would get all the numbers from zero to 50.

Here's an example of creating a range between zero and 50 with an increment of 5:

```
np.arange(0, 51, step=5.0)
```

```
array([ 0.,  5., 10., 15., 20., 25., 30., 35., 40., 45., 50.])
```

You can create the same graph using the object-oriented approach with the following code:

```
fig, ax = plt.subplots()
ax.barh(x_axis, y_axis)
ax.set_xticks(np.arange(0, 51, step=5.0))
plt.show()
```

Alternatively, you can use `ax.xaxis.set_ticks()` instead of `ax.set_xticks()`.

If our y-axis had numerical values, we could change how often every major y-tick occurs by using the `yticks()` function for the MATLAB approach and the `set_yticks()` function for the object-oriented approach.

NOTE

> For further information on NumPy and the `arange` function, please see the following documentation:
>
> **NumPy website (https://numpy.org/)**
>
> **SciPy page on numpy.arange (https://docs.scipy.org/doc/numpy/reference/generated/numpy.arange.html)**

Add Minor Ticks

We can also add minor ticks without numerical values between the major ticks on the x- or y-axis. Minor ticks make it even easier to estimate values for the data in the chart.

To add minor ticks to the x-axis on a horizontal bar chart, we use the `set_minor_locator` function with the `xaxis` class in the object-oriented method, like this:

```
x.xaxis.set_minor_locator()
```

Inside the `set_minor_locator` function, we add the `MultipleLocator` method and pass how often we want our minor ticks.

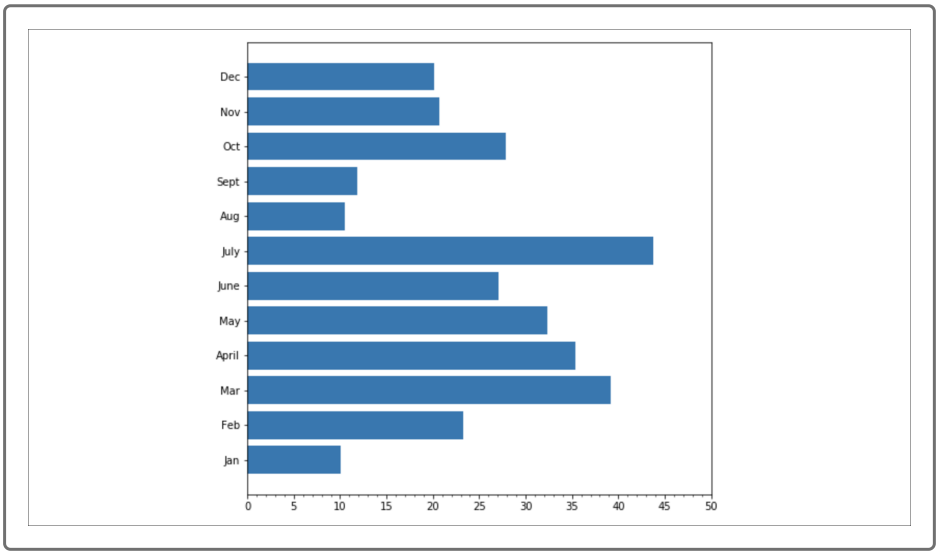
To use these functions and methods, we have to import the `MultipleLocator` method from the `matplotlib.ticker` function.

Add the following code to a new cell:

```
from matplotlib.ticker import MultipleLocator
# Increase the size of the plot figure.
fig, ax = plt.subplots(figsize=(8, 8))
ax.barh(x_axis, y_axis)
ax.set_xticks(np.arange(0, 51, step=5.0))

# Create minor ticks at an increment of 1.
ax.xaxis.set_minor_locator(MultipleLocator(1))
plt.show()
```

Now, the figure will have minor ticks every $1 increment and major ticks at every $5 increment, as seen here:



NOTE

> For more information, see the **Matplotlib documentation on major and minor ticks (https://matplotlib.org/stable/gallery/ticks_and_spines/major_minor_demo.html)** .