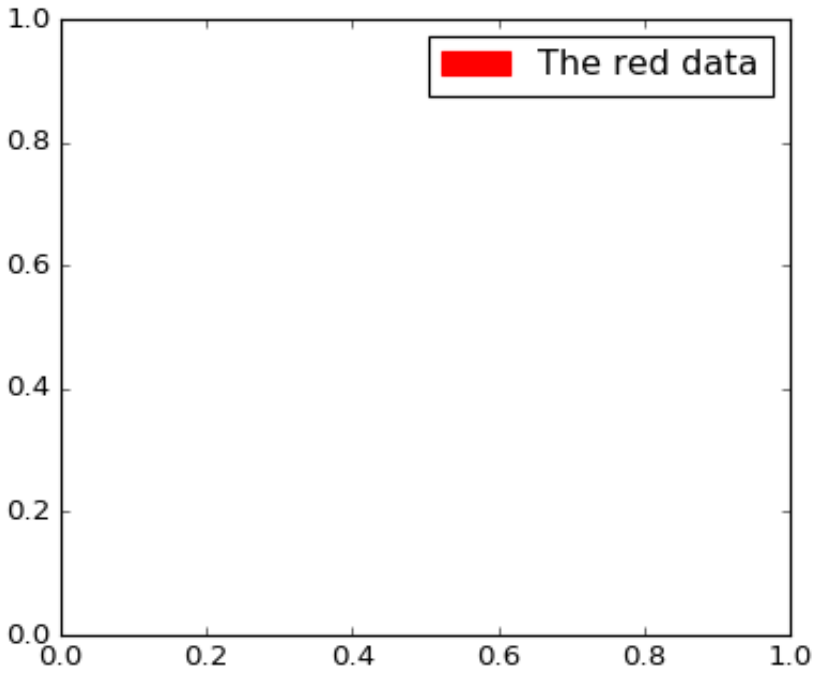
Legend guide — Matplotlib 1.5.1 documentation



There are many supported legend handles, instead of creating a patch of color we could have created a line with a marker:

```
import matplotlib.lines as mlines
import matplotlib.pyplot as plt

blue_line = mlines.Line2D([], [], color='blue', marker='*',
                          markersize=15, label='Blue stars')
plt.legend(handles=[blue_line])

plt.show()
```

Legend location¶

The location of the legend can be specified by the keyword argument *loc*. Please see the documentation at `legend()` for more details.

The `bbox_to_anchor` keyword gives a great degree of control for manual legend placement. For example, if you want your axes legend located at the figure's top right-hand corner instead of the axes' corner, simply specify the corner's location, and the coordinate system of that location:

```
plt.legend(bbox_to_anchor=(1, 1),
```

```
           bbox_transform=plt.gcf().transFigure)
```

More examples of custom legend placement:

```
import matplotlib.pyplot as plt


plt.subplot(211)
plt.plot([1,2,3], label="test1")
plt.plot([3,2,1], label="test2")
# Place a legend above this legend, expanding itself to
# fully use the given bounding box.
plt.legend(bbox_to_anchor=(0., 1.02, 1., .102), loc=3,
           ncol=2, mode="expand", borderaxespad=0.)

plt.subplot(223)
plt.plot([1,2,3], label="test1")
plt.plot([3,2,1], label="test2")
# Place a legend to the right of this smaller figure.
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)

plt.show()
```

Multiple legends on the same Axes

Sometimes it is more clear to split legend entries across multiple legends. Whilst the instinctive approach to doing this might be to call the `legend()` function multiple times, you will find that only one legend ever exists on the Axes. This has been done so that it is possible to call `legend()` repeatedly to update the legend to the latest handles on the Axes, so to persist old legend instances, we must add them manually to the Axes:

```
import matplotlib.pyplot as plt

line1, = plt.plot([1,2,3], label="Line 1", linestyle='--')
line2, = plt.plot([3,2,1], label="Line 2", linewidth=4)

# Create a legend for the first line.
first_legend = plt.legend(handles=[line1], loc=1)

# Add the legend manually to the current Axes.
ax = plt.gca().add_artist(first_legend)

# Create another legend for the second line.
```

```
plt.legend(handles=[line2], loc=4)

plt.show()
```

Legend Handlers

In order to create legend entries, handles are given as an argument to an appropriate `HandlerBase` subclass. The choice of handler subclass is determined by the following rules:

> 1. Update `get_legend_handler_map()` with the value in the `handler_map` keyword.
> 2. Check if the `handle` is in the newly created `handler_map`.
> 3. Check if the type of `handle` is in the newly created `handler_map`.
> 4. Check if any of the types in the `handle` 's mro is in the newly created `handler_map`.

For completeness, this logic is mostly implemented in `get_legend_handler()`.

All of this flexibility means that we have the necessary hooks to implement custom handlers for our own type of legend key.

The simplest example of using custom handlers is to instantiate one of the existing `HandlerBase` subclasses. For the sake of simplicity, let's choose `matplotlib.legend_handler.HandlerLine2D` which accepts a `numpoints` argument (note numpoints is a keyword on the `legend()` function for convenience). We can then pass the mapping of instance to Handler as a keyword to legend.

```
import matplotlib.pyplot as plt
from matplotlib.legend_handler import HandlerLine2D

line1, = plt.plot([3,2,1], marker='o', label='Line 1')
line2, = plt.plot([1,2,3], marker='o', label='Line 2')

plt.legend(handler_map={line1: HandlerLine2D(numpoints=4)})
```

As you can see, "Line 1" now has 4 marker points, where "Line 2" has 2 (the default). Try the above code, only change the map's key from `line1` to `type(line1)`. Notice how now both `Line2D` instances get 4 markers.

Along with handlers for complex plot types such as errorbars, stem plots and histograms, the default `handler_map` has a special `tuple` handler (`HandlerTuple`) which simply plots the handles on top of one another for each item in the given tuple. The following example demonstrates combining two legend keys on top of one another:

```
import matplotlib.pyplot as plt
from numpy.random import randn
```

```
z = randn(10)

red_dot, = plt.plot(z, "ro", markersize=15)
# Put a white cross over some of the data.
white_cross, = plt.plot(z[:5], "w+", markeredgewidth=3, markersize=15)

plt.legend([red_dot, (red_dot, white_cross)], ["Attr A", "Attr A+B"])
```