# Non-Positional Encodings for Third Variables

There are four major cases to consider when we want to plot three variables together:

* three numeric variables
* two numeric variables and one categorical variable
* one numeric variable and two categorical variables
* three categorical variables

If we have at least two numeric variables, as in the first two cases, one common method for depicting the data is by using a scatterplot to encode two of the numeric variables, then using a non-positional encoding on the points to convey the value on the third variable, whether numeric or categorical. (You will see additional techniques later in the lesson that can also be applied to the other two cases, i.e. where we have at least two categorical variables.)

Three main non-positional encodings stand out: shape, size, and color. For Matplotlib and Seaborn, color is the easiest of these three encodings to apply for a third variable. Color can be used to encode both qualitative and quantitative data, with different types of color palettes used for different use cases. Because of how broadly color can be used, a dedicated discussion of color and its use in Matplotlib and Seaborn will be deferred to the next page in the lesson.

**Encoding via Shape**

Shape is a good encoding for categorical variables, using one shape for each level of the categorical variable. Unfortunately, there is no built-in way to automatically assign different shapes in a single call of the scatter or regplot function. Instead, we need to write a loop to call our plotting function multiple times, isolating data points by categorical level and setting a different "marker" argument value for each one.

cat\_markers = [['A', 'o'],

['B', 's']]

**for** cat, marker **in** cat\_markers:

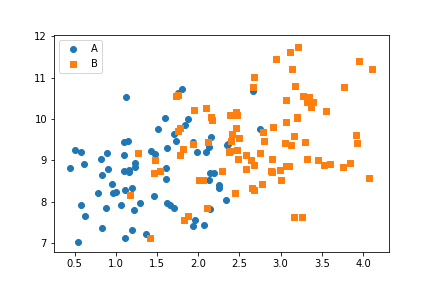
df\_cat = df[df['cat\_var1'] == cat]

plt.scatter(data = df\_cat, x = 'num\_var1', y = 'num\_var2', marker = marker)

plt.legend(['A','B'])

The 'o' string specifies circular markers for members of category 'A', while the 's' string specifies square markers for members of category 'B'. The [**legend**](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.legend.html) function adds a legend to the plot, with one marker for every scatter call made. The function argument sets the labels for those points.

(Documentation: [**matplotlib built-in markers**](https://matplotlib.org/api/markers_api.html), [**marker reference example**](https://matplotlib.org/examples/lines_bars_and_markers/marker_reference.html))

**[[](https://classroom.udacity.com/nanodegrees/nd002-ent/parts/9c2eb5bf-2bda-4e0c-85bf-d997082a8571/modules/22d4ed2d-b711-4e1d-8fed-604f524f0595/lessons/444a6d18-33cd-46a9-83cb-6fdbf53261fd/concepts/0f3b9898-d917-4cc8-a688-bef35d8566fb)](https://classroom.udacity.com/nanodegrees/nd002-ent/parts/9c2eb5bf-2bda-4e0c-85bf-d997082a8571/modules/22d4ed2d-b711-4e1d-8fed-604f524f0595/lessons/444a6d18-33cd-46a9-83cb-6fdbf53261fd/concepts/0f3b9898-d917-4cc8-a688-bef35d8566fb)**

From the positional encodings in the plot, you can see that there is a modest positive relationship between the two numeric variables. Adding the categorical variable via shape encoding, we can see that points of category 'A' tend to be smaller than those of category 'B' in terms of the numeric x-variable ("num\_var1"). Neither category seems to have an advantage in terms of variability or value for the numeric y-variable ("num\_var2").

Note that the two categories have automatically been double-encoded with different colors, in addition to the explicitly specified markers. If we wanted the points to have the same color as well, we could do that through the "c" parameter in scatter or "color" in regplot. A small example of where this is useful will be seen in the next section.

**Encoding via Size**

Point size is a good encoding for numeric variables. Usually, we want the numeric values to be proportional to the area of the point markers; this is the default functionality of the "s" parameter in scatter. (You need to refer to "s" through a dictionary assigned to the "scatter\_kws" parameter when working with regplot.)

plt.scatter(data = df, x = 'num\_var1', y = 'num\_var2', s = 'num\_var3')

*# dummy series for adding legend*

sizes = [20, 35, 50]

base\_color = sb.color\_palette()[0]

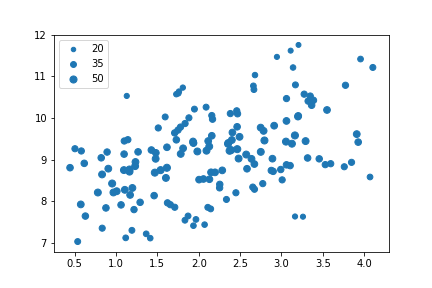
legend\_obj = []

**for** s **in** sizes:

legend\_obj.append(plt.scatter([], [], s = s, color = base\_color))

plt.legend(legend\_obj, sizes)

While the creation of the plot is easier with size, we need to put in extra leg work to create a legend. Since, as noted above, legend will only create one legend entry per plot function call, we need to create additional dummy scatter objects with reference sizes to create the plot. The color is fixed so that all of the legend entries have the same color, and two arguments are provided to the legend function: the list of plotting objects to depict in the legend, and their labels.

**[[](https://classroom.udacity.com/nanodegrees/nd002-ent/parts/9c2eb5bf-2bda-4e0c-85bf-d997082a8571/modules/22d4ed2d-b711-4e1d-8fed-604f524f0595/lessons/444a6d18-33cd-46a9-83cb-6fdbf53261fd/concepts/0f3b9898-d917-4cc8-a688-bef35d8566fb)](https://classroom.udacity.com/nanodegrees/nd002-ent/parts/9c2eb5bf-2bda-4e0c-85bf-d997082a8571/modules/22d4ed2d-b711-4e1d-8fed-604f524f0595/lessons/444a6d18-33cd-46a9-83cb-6fdbf53261fd/concepts/0f3b9898-d917-4cc8-a688-bef35d8566fb)**

The size encoding for the third numeric variable ("num\_var3") shows that its values are largest in the 'middle' of the distribution of values, and smaller on the upper and bottom edges. It is also clear that size is much less precise an encoding than position, so it is better used to make general, qualitative judgments than precise judgments.

In the case of the example, the data was also scaled in a way that the marker sizes made sense as given. You might need to apply a scaling factor (e.g., multiplying or dividing all values by 2) or shift in order to make the size encoding interpretable. In particular, if the values in your third numeric variable include negative values, then you might want to choose a color encoding instead.

**A Warning on Combining Encodings**

It might seem plausible to combine both size and shape encodings into the same plot, to depict the trend in four variables at once. Technically, this may be true, but there are some cautions to be taken with this approach. One surface issue is that the code to depict the plot and a reasonable legend gets complicated. A more important issue is that point areas won't all be the same even with the same value, depending on the shape of the marker:

plt.figure(figsize = [5,5])

common\_size = 2500

plt.scatter([0],[1], marker = 's', s = common\_size)

plt.scatter([1],[1], marker = '^', s = common\_size)

plt.scatter([0],[0], marker = 'o', s = common\_size)

plt.scatter([1],[0], marker = 'X', s = common\_size)

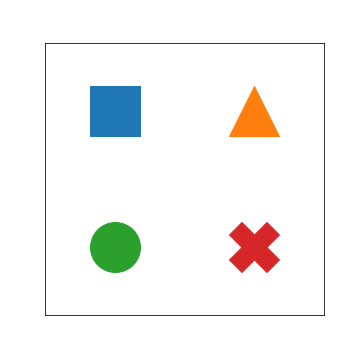
*# a little bit of aesthetic cleaning*

plt.xlim(-0.5,1.5)

plt.xticks([])

plt.ylim(-0.5,1.5)

plt.yticks([])

**[[](https://classroom.udacity.com/nanodegrees/nd002-ent/parts/9c2eb5bf-2bda-4e0c-85bf-d997082a8571/modules/22d4ed2d-b711-4e1d-8fed-604f524f0595/lessons/444a6d18-33cd-46a9-83cb-6fdbf53261fd/concepts/0f3b9898-d917-4cc8-a688-bef35d8566fb)](https://classroom.udacity.com/nanodegrees/nd002-ent/parts/9c2eb5bf-2bda-4e0c-85bf-d997082a8571/modules/22d4ed2d-b711-4e1d-8fed-604f524f0595/lessons/444a6d18-33cd-46a9-83cb-6fdbf53261fd/concepts/0f3b9898-d917-4cc8-a688-bef35d8566fb)**

Despite having the same "s" values, the triangle, circle, and X markers all look smaller (have a smaller area) than the square. Perhaps this isn't a major concern, considering that size is better used as a qualitative measure for exploration. But it's still something to consider. A little more discussion of the number of variables that can be reasonably packed into a plot can be found later in the lesson ("How much is too much?").

**Further Reading**

* Bloomberg: [**Tracking Super Bowl Ticket Prices**](https://www.bloomberg.com/graphics/infographics/tracking-super-bowl-ticket-prices.html) - This visualization shows an interesting encoding for a third variable: connecting points by a line to show the changes in the two axis values across time. A different marker at each end of the path shows the starting point and the end point.

# Encoding via Color

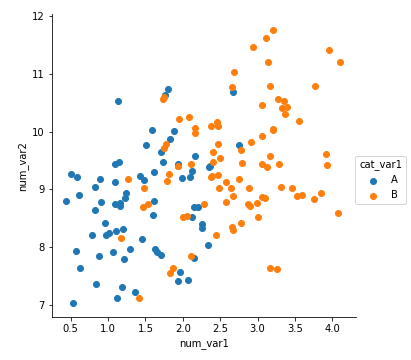
Color is a very common encoding for variables, for both qualitative and quantitative variables. You've already seen this employed in previous lessons where position could not be used to encode a value: color for category in a clustered bar chart, and color for count in a heat map (both as a 2-d histogram and as a 2-d bar chart). Here, we'll look at how to employ color in scatterplots, as well as discuss more about color palette choices depending on the type of data you have.

If you have a qualitative variable, you can set different colors for different levels of a categorical variable through the "hue" parameter on seaborn's **[FacetGrid](https://seaborn.pydata.org/generated/seaborn.FacetGrid.html" \t "_blank)** class.

g = sb.FacetGrid(data = df, hue = 'cat\_var1', size = 5)

g.map(plt.scatter, 'num\_var1', 'num\_var2')

g.add\_legend()

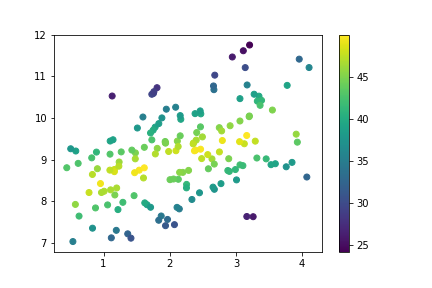
**[[](https://classroom.udacity.com/nanodegrees/nd002-ent/parts/9c2eb5bf-2bda-4e0c-85bf-d997082a8571/modules/22d4ed2d-b711-4e1d-8fed-604f524f0595/lessons/444a6d18-33cd-46a9-83cb-6fdbf53261fd/concepts/ba31fc66-d628-4125-a937-2b3e7974a696)](https://classroom.udacity.com/nanodegrees/nd002-ent/parts/9c2eb5bf-2bda-4e0c-85bf-d997082a8571/modules/22d4ed2d-b711-4e1d-8fed-604f524f0595/lessons/444a6d18-33cd-46a9-83cb-6fdbf53261fd/concepts/ba31fc66-d628-4125-a937-2b3e7974a696)**

We get a plot that is similar to what was observed in the previous page: points of category A tend to be smaller than category B on "num\_var1" for the x-axis, and there doesn't appear to be any interaction with "num\_var2" as depicted on the y-axis.

For quantitative variables, we should not take the same approach, since **FacetGrid** expects any variable input for subsetting to be categorical. Instead, we can set color based on numeric value in the scatter function through the "c" parameter, much like how we set up marker sizes through "s". (Unlike with size, we don't have an easy way of setting color by numeric value through regplot due to how its "color" argument is set up.)

plt.scatter(data = df, x = 'num\_var1', y = 'num\_var2', c = 'num\_var3')

plt.colorbar()

**[[](https://classroom.udacity.com/nanodegrees/nd002-ent/parts/9c2eb5bf-2bda-4e0c-85bf-d997082a8571/modules/22d4ed2d-b711-4e1d-8fed-604f524f0595/lessons/444a6d18-33cd-46a9-83cb-6fdbf53261fd/concepts/ba31fc66-d628-4125-a937-2b3e7974a696)](https://classroom.udacity.com/nanodegrees/nd002-ent/parts/9c2eb5bf-2bda-4e0c-85bf-d997082a8571/modules/22d4ed2d-b711-4e1d-8fed-604f524f0595/lessons/444a6d18-33cd-46a9-83cb-6fdbf53261fd/concepts/ba31fc66-d628-4125-a937-2b3e7974a696)**

As we saw before, when points lie further away from the central line through the data, the value on the third variable decreases. Compared to the size encoding, we have a more precise idea of value on the third variable, though certainly not as precise as positional encodings.

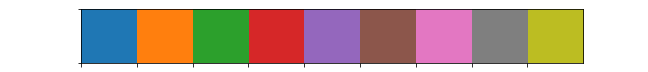
## Color Palettes

Depending on the type of data you have, you may want to change the type of color palette that you use to depict your data. There are three major classes of color palette to consider: qualitative, sequential, and diverging.

**Qualitative** palettes are built for nominal-type data. This is the palette class taken by the default palette.

sb.palplot(sb.color\_palette(n\_colors=9))

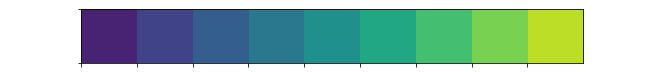
(Documentation: seaborn **[palplot](https://seaborn.pydata.org/generated/seaborn.palplot.html" \t "_blank)**, **[color\_palette](https://seaborn.pydata.org/generated/seaborn.color_palette.html" \t "_blank)**)

**[[](https://classroom.udacity.com/nanodegrees/nd002-ent/parts/9c2eb5bf-2bda-4e0c-85bf-d997082a8571/modules/22d4ed2d-b711-4e1d-8fed-604f524f0595/lessons/444a6d18-33cd-46a9-83cb-6fdbf53261fd/concepts/ba31fc66-d628-4125-a937-2b3e7974a696)](https://classroom.udacity.com/nanodegrees/nd002-ent/parts/9c2eb5bf-2bda-4e0c-85bf-d997082a8571/modules/22d4ed2d-b711-4e1d-8fed-604f524f0595/lessons/444a6d18-33cd-46a9-83cb-6fdbf53261fd/concepts/ba31fc66-d628-4125-a937-2b3e7974a696)**

In a qualitative palette, consecutive color values are distinct so that there is no inherent ordering of levels implied. Colors in a good qualitative palette should also try and avoid drastic changes in brightness and saturation that would cause a reader to interpret one category as being more important than the others - unless that emphasis is deliberate and purposeful.

For other types of data (ordinal and numeric), a choice may need to be made between a sequential scale and a diverging scale. In a **sequential** palette, consecutive color values should follow each other systematically. Typically, this follows a light-to-dark trend across a single or small range of hues, where light colors indicate low values and dark colors indicate high values. The default sequential color map, "viridis", takes the opposite approach, with dark colors indicating low values, and light values indicating high.

sb.palplot(sb.color\_palette('viridis', 9))

**[[](https://classroom.udacity.com/nanodegrees/nd002-ent/parts/9c2eb5bf-2bda-4e0c-85bf-d997082a8571/modules/22d4ed2d-b711-4e1d-8fed-604f524f0595/lessons/444a6d18-33cd-46a9-83cb-6fdbf53261fd/concepts/ba31fc66-d628-4125-a937-2b3e7974a696)](https://classroom.udacity.com/nanodegrees/nd002-ent/parts/9c2eb5bf-2bda-4e0c-85bf-d997082a8571/modules/22d4ed2d-b711-4e1d-8fed-604f524f0595/lessons/444a6d18-33cd-46a9-83cb-6fdbf53261fd/concepts/ba31fc66-d628-4125-a937-2b3e7974a696)**

Most of the time, a sequential palette will depict ordinal or numeric data just fine. However, if there is a meaningful zero or center value for the variable, you may want to consider using a **diverging** palette. In a diverging palette, two sequential palettes with different hues are put back to back, with a common color (usually white or gray) connecting them. One hue indicates values greater than the center point, while the other indicates values smaller than the center.

sb.palplot(sb.color\_palette('vlag', 9))

**[[](https://classroom.udacity.com/nanodegrees/nd002-ent/parts/9c2eb5bf-2bda-4e0c-85bf-d997082a8571/modules/22d4ed2d-b711-4e1d-8fed-604f524f0595/lessons/444a6d18-33cd-46a9-83cb-6fdbf53261fd/concepts/ba31fc66-d628-4125-a937-2b3e7974a696)](https://classroom.udacity.com/nanodegrees/nd002-ent/parts/9c2eb5bf-2bda-4e0c-85bf-d997082a8571/modules/22d4ed2d-b711-4e1d-8fed-604f524f0595/lessons/444a6d18-33cd-46a9-83cb-6fdbf53261fd/concepts/ba31fc66-d628-4125-a937-2b3e7974a696)**

## Selecting Color Palettes

If you want to change the color map for your plot, the easiest way of doing so is by using one of the built-ins from Matplotlib or Seaborn. [**This part**](https://matplotlib.org/api/pyplot_summary.html#colors-in-matplotlib) of the Matplotlib documentation has a list of strings that can be understood for color mappings. For most of your purposes, stick with the palettes noted in the top few tables as built-in for Matplotlib ('viridis', etc.) or from ColorBrewer; the remaining palettes may not be as perceptually consistent. Seaborn also adds in a number of its own palettes:

* **Qualitative** (all up to 6 colors): 'deep', 'pastel', 'dark', 'muted', 'bright', 'colorblind'
* **Sequential**: 'rocket' (white-orange-red-purple-black), 'mako' (mint-green-blue-purple-black)
* **Diverging**: 'vlag' (blue-white-red), 'icefire' (blue-black-orange)

For all of these strings, appending '\_r' reverses the palette, which is useful if a sequential or diverging palette is rendered counter to your expectations.

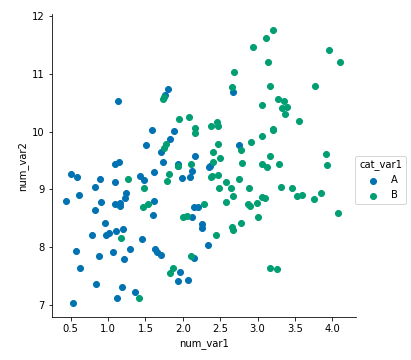
A color palette can be set in **FacetGrid** through the "palette" parameter, and in scatter through the "cmap" parameter.

g = sb.FacetGrid(data = df, hue = 'cat\_var1', size = 5,

palette = 'colorblind')

g.map(plt.scatter, 'num\_var1', 'num\_var2')

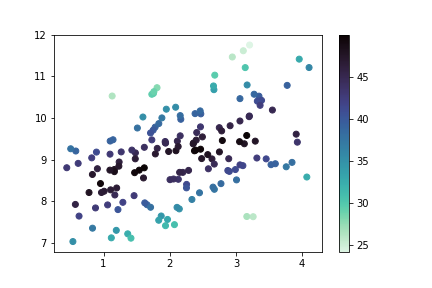
g.add\_legend()

**[[](https://classroom.udacity.com/nanodegrees/nd002-ent/parts/9c2eb5bf-2bda-4e0c-85bf-d997082a8571/modules/22d4ed2d-b711-4e1d-8fed-604f524f0595/lessons/444a6d18-33cd-46a9-83cb-6fdbf53261fd/concepts/ba31fc66-d628-4125-a937-2b3e7974a696)](https://classroom.udacity.com/nanodegrees/nd002-ent/parts/9c2eb5bf-2bda-4e0c-85bf-d997082a8571/modules/22d4ed2d-b711-4e1d-8fed-604f524f0595/lessons/444a6d18-33cd-46a9-83cb-6fdbf53261fd/concepts/ba31fc66-d628-4125-a937-2b3e7974a696)**

plt.scatter(data = df, x = 'num\_var1', y = 'num\_var2', c = 'num\_var3',

cmap = 'mako\_r')

plt.colorbar()

**[[](https://classroom.udacity.com/nanodegrees/nd002-ent/parts/9c2eb5bf-2bda-4e0c-85bf-d997082a8571/modules/22d4ed2d-b711-4e1d-8fed-604f524f0595/lessons/444a6d18-33cd-46a9-83cb-6fdbf53261fd/concepts/ba31fc66-d628-4125-a937-2b3e7974a696)](https://classroom.udacity.com/nanodegrees/nd002-ent/parts/9c2eb5bf-2bda-4e0c-85bf-d997082a8571/modules/22d4ed2d-b711-4e1d-8fed-604f524f0595/lessons/444a6d18-33cd-46a9-83cb-6fdbf53261fd/concepts/ba31fc66-d628-4125-a937-2b3e7974a696)**

When using a diverging color palette, you will likely need to specify the "vmin" and "vmax" parameters in order to have the neutral point in the palette meet the center point in the scale. Alternatively, solutions that create a different normalization function like the one posted in [**this Stack Overflow thread**](https://stackoverflow.com/questions/20144529/shifted-colorbar-matplotlib) can be used for finer control over the color map. Diverging color scales are common enough for the heatmap type that there is a "center" parameter for setting the central value. You'll see a demonstration of this later in the lesson ("Plot Matrices").

## Warnings on Color

There are a couple of things to be aware of in general, when working with color. First of all, try and consider color blindness when selecting color for your plots. You don't want your visualization to shut out the 8% of the population that have some kind of color vision deficiency. Fortunately, the built-in color palettes highlighted in the previous section should minimize these concerns. If you use a different, or custom palette, it might be worth checking your visualization's interpretability through a color blindness sim like [**this one**](http://www.color-blindness.com/coblis-color-blindness-simulator/).

Secondly, you want to be aware of the effect of transparency and overlap on interpretability. If points of different color on a qualitative scale overlap, the result may be a third color that cannot be matched to something in the palette. If multiple points on a quantitative scale overlap, then the result may be a value that does not actually exist in the data. To be safe here, avoid or minimize transparency in plots with color. You may need to plot only a sample of your points in order to make sure that the effect of the third variable is clearly visible.

plt.figure(figsize = [5,5])

*# left: qualitative points*

plt.scatter(0,0.5,s = 1e4, c = sb.color\_palette()[0], alpha = 0.5)

plt.scatter(0,-0.5,s = 1e4, c = sb.color\_palette()[1], alpha = 0.5)

*# right: quantitative points*

plt.scatter(1,0.5,s = 1e4, c = sb.color\_palette('Blues')[2], alpha = 0.5)

plt.scatter(1,-0.5,s = 1e4, c = sb.color\_palette('Blues')[4], alpha = 0.5)

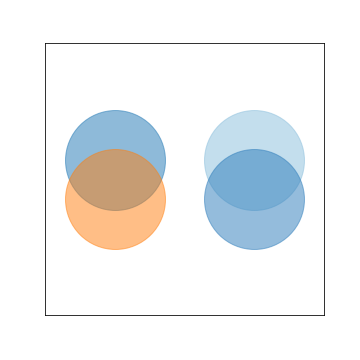
*# set axes for point overlap*

plt.xlim(-0.5,1.5)

plt.ylim(-3.5,3.5)

plt.xticks([])

plt.yticks([])

**[[](https://classroom.udacity.com/nanodegrees/nd002-ent/parts/9c2eb5bf-2bda-4e0c-85bf-d997082a8571/modules/22d4ed2d-b711-4e1d-8fed-604f524f0595/lessons/444a6d18-33cd-46a9-83cb-6fdbf53261fd/concepts/ba31fc66-d628-4125-a937-2b3e7974a696)](https://classroom.udacity.com/nanodegrees/nd002-ent/parts/9c2eb5bf-2bda-4e0c-85bf-d997082a8571/modules/22d4ed2d-b711-4e1d-8fed-604f524f0595/lessons/444a6d18-33cd-46a9-83cb-6fdbf53261fd/concepts/ba31fc66-d628-4125-a937-2b3e7974a696)**

## Further Reading

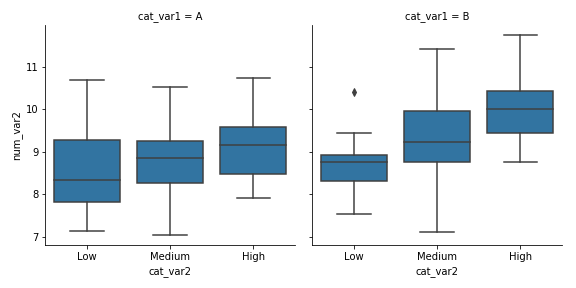
* Matplotlib tutorial: [**Colormaps in Matplotlib**](https://matplotlib.org/tutorials/colors/colormaps.html)
* Seaborn tutorial: [**Choosing color palettes**](https://seaborn.pydata.org/tutorial/color_palettes.html)
* Eager Eyes: [**How The Rainbow Color Map Misleads**](https://eagereyes.org/basics/rainbow-color-map) - seaborn will refuse to accept the 'jet' palette string, which corresponds with a rainbow-colored palette that covers the entire range of hues. This, and the next link, will tell you why it's been forbidden.
* Agile Scientific: [**No more rainbows!**](https://agilescientific.com/blog/2017/12/14/no-more-rainbows)
* Datawrapper: [**How to Choose a Color Palette for Choropleth Maps**](https://blog.datawrapper.de/how-to-choose-a-color-palette-for-choropleth-maps/) - Though this article discusses color in the context of maps and for a specific software tool, it's a useful reference if you want to create a non-linear normalization function.

# Faceting for Multivariate Data

In the previous lesson, you saw how **FacetGrid** could be used to subset your dataset across levels of a categorical variable, and then create one plot for each subset. Where the faceted plots demonstrated were univariate before, you can actually use any plot type, allowing you to facet bivariate plots to create a multivariate visualization.

g = sb.FacetGrid(data = df, col = 'cat\_var1', size = 4)

g.map(sb.boxplot, 'cat\_var2', 'num\_var2')

**[[](https://classroom.udacity.com/nanodegrees/nd002-ent/parts/9c2eb5bf-2bda-4e0c-85bf-d997082a8571/modules/22d4ed2d-b711-4e1d-8fed-604f524f0595/lessons/444a6d18-33cd-46a9-83cb-6fdbf53261fd/concepts/d64700c6-2de9-4183-a9f9-2ba7779602b9)](https://classroom.udacity.com/nanodegrees/nd002-ent/parts/9c2eb5bf-2bda-4e0c-85bf-d997082a8571/modules/22d4ed2d-b711-4e1d-8fed-604f524f0595/lessons/444a6d18-33cd-46a9-83cb-6fdbf53261fd/concepts/d64700c6-2de9-4183-a9f9-2ba7779602b9)**

The faceted box plot suggests a slight interaction between the two categorical variables, where, in level B of "cat\_var1", the level of "cat\_var2" seems to be have a larger effect on the value of "num\_var2", compared to the trend within "cat\_var1" level A.

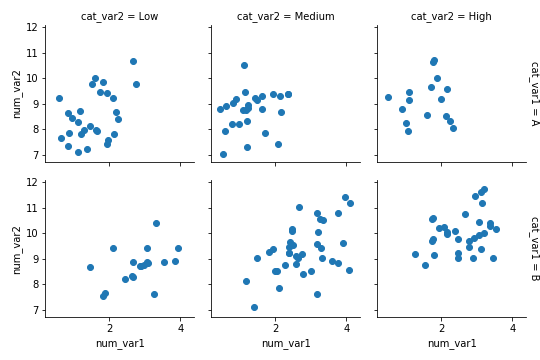
**FacetGrid** also allows for faceting a variable not just by columns, but also by rows. We can set one categorical variable on each of the two facet axes for one additional method of depicting multivariate trends.

g = sb.FacetGrid(data = df, col = 'cat\_var2', row = 'cat\_var1', size = 2.5,

margin\_titles = **True**)

g.map(plt.scatter, 'num\_var1', 'num\_var2')

Setting margin\_titles = True means that instead of each facet being labeled with the combination of row and column variable, labels are placed separately on the top and right margins of the facet grid. This is a boon, since the default plot titles are usually too long.

**[[](https://classroom.udacity.com/nanodegrees/nd002-ent/parts/9c2eb5bf-2bda-4e0c-85bf-d997082a8571/modules/22d4ed2d-b711-4e1d-8fed-604f524f0595/lessons/444a6d18-33cd-46a9-83cb-6fdbf53261fd/concepts/d64700c6-2de9-4183-a9f9-2ba7779602b9)](https://classroom.udacity.com/nanodegrees/nd002-ent/parts/9c2eb5bf-2bda-4e0c-85bf-d997082a8571/modules/22d4ed2d-b711-4e1d-8fed-604f524f0595/lessons/444a6d18-33cd-46a9-83cb-6fdbf53261fd/concepts/d64700c6-2de9-4183-a9f9-2ba7779602b9)**

# Other Adaptations of Bivariate Plots

You also saw one other way of expanding univariate plots into bivariate plots in the previous lesson: substituting count on a bar chart or histogram for the mean, median, or some other statistic of a second variable. This adaptation can also be done for bivariate plots like the heat map, clustered bar chart, and line plot, to allow them to depict multivariate relationships.

If we want to depict the mean of a third variable in a **2-d histogram**, we need to change the weights of points in the hist2d function similar to how we changed the weights in the 1-d histogram.

xbin\_edges = np.arange(0.25, df['num\_var1'].max()+0.5, 0.5)

ybin\_edges = np.arange(7, df['num\_var2'].max()+0.5, 0.5)

*# count number of points in each bin*

xbin\_idxs = pd.cut(df['num\_var1'], xbin\_edges, right = **False**,

include\_lowest = **True**, labels = **False**).astype(int)

ybin\_idxs = pd.cut(df['num\_var2'], ybin\_edges, right = **False**,

include\_lowest = **True**, labels = **False**).astype(int)

pts\_per\_bin = df.groupby([xbin\_idxs, ybin\_idxs]).size()

pts\_per\_bin = pts\_per\_bin.reset\_index()

pts\_per\_bin = pts\_per\_bin.pivot(index = 'num\_var1', columns = 'num\_var2').values

z\_wts = df['num\_var3'] / pts\_per\_bin[xbin\_idxs, ybin\_idxs]

*# plot the data using the calculated weights*

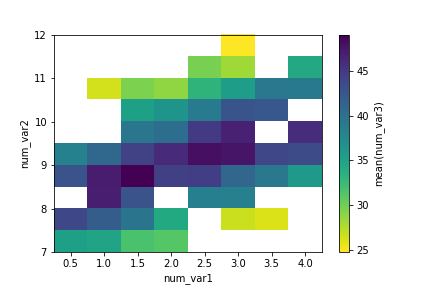
plt.hist2d(data = df, x = 'num\_var1', y = 'num\_var2', weights = z\_wts,

bins = [xbin\_edges, ybin\_edges], cmap = 'viridis\_r', cmin = 0.5);

plt.xlabel('num\_var1')

plt.ylabel('num\_var2');

plt.colorbar(label = 'mean(num\_var3)');

**[[](https://classroom.udacity.com/nanodegrees/nd002-ent/parts/9c2eb5bf-2bda-4e0c-85bf-d997082a8571/modules/22d4ed2d-b711-4e1d-8fed-604f524f0595/lessons/444a6d18-33cd-46a9-83cb-6fdbf53261fd/concepts/c8f7618a-637d-427a-b8df-83a0bc45a4cf)](https://classroom.udacity.com/nanodegrees/nd002-ent/parts/9c2eb5bf-2bda-4e0c-85bf-d997082a8571/modules/22d4ed2d-b711-4e1d-8fed-604f524f0595/lessons/444a6d18-33cd-46a9-83cb-6fdbf53261fd/concepts/c8f7618a-637d-427a-b8df-83a0bc45a4cf)**

Considering how few data points there are in the example, and how cleanly the third variable is distributed, the adapted heat map is a bit excessive in terms of work. The low level of point overlap observed earlier means that the scatterplot with color or size encoding was sufficient for depicting the data. You'll be more likely to use the heat map if there is a lot of data to be aggregated.

The code for the **2-d bar chart** doesn't actually change much. The actual heatmap call is still the same, only the aggregation of values changes. Instead of taking size after the groupby operation, we compute the mean across dataframe columns and isolate the column of interest.

cat\_means = df.groupby(['cat\_var1', 'cat\_var2']).mean()['num\_var2']

cat\_means = cat\_means.reset\_index(name = 'num\_var2\_avg')

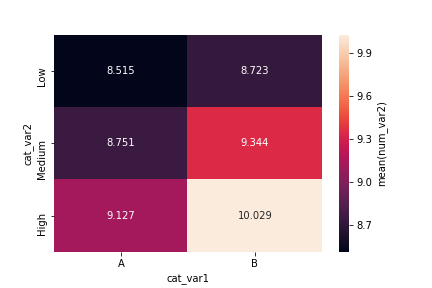
cat\_means = cat\_means.pivot(index = 'cat\_var2', columns = 'cat\_var1',

values = 'num\_var2\_avg')

sb.heatmap(cat\_means, annot = **True**, fmt = '.3f',

cbar\_kws = {'label' : 'mean(num\_var2)'})

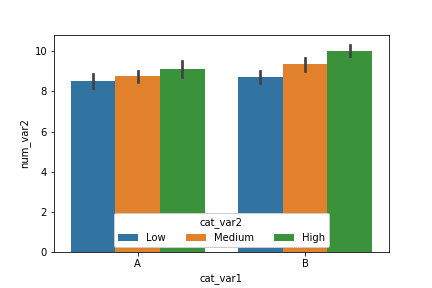
Note how the "cbar\_kws" provides an additional argument to the colorbar component of the heat map call.

**[[](https://classroom.udacity.com/nanodegrees/nd002-ent/parts/9c2eb5bf-2bda-4e0c-85bf-d997082a8571/modules/22d4ed2d-b711-4e1d-8fed-604f524f0595/lessons/444a6d18-33cd-46a9-83cb-6fdbf53261fd/concepts/c8f7618a-637d-427a-b8df-83a0bc45a4cf)](https://classroom.udacity.com/nanodegrees/nd002-ent/parts/9c2eb5bf-2bda-4e0c-85bf-d997082a8571/modules/22d4ed2d-b711-4e1d-8fed-604f524f0595/lessons/444a6d18-33cd-46a9-83cb-6fdbf53261fd/concepts/c8f7618a-637d-427a-b8df-83a0bc45a4cf)**

An alternative approach for two categorical variables and one numeric variable is to adapt a **clustered bar chart** using the barplot function instead of the countplot function:

ax = sb.barplot(data = df, x = 'cat\_var1', y = 'num\_var2', hue = 'cat\_var2')

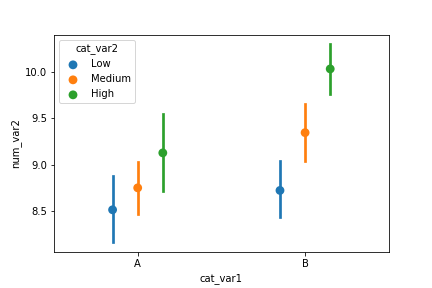
ax.legend(loc = 8, ncol = 3, framealpha = 1, title = 'cat\_var2')

**[[](https://classroom.udacity.com/nanodegrees/nd002-ent/parts/9c2eb5bf-2bda-4e0c-85bf-d997082a8571/modules/22d4ed2d-b711-4e1d-8fed-604f524f0595/lessons/444a6d18-33cd-46a9-83cb-6fdbf53261fd/concepts/c8f7618a-637d-427a-b8df-83a0bc45a4cf)](https://classroom.udacity.com/nanodegrees/nd002-ent/parts/9c2eb5bf-2bda-4e0c-85bf-d997082a8571/modules/22d4ed2d-b711-4e1d-8fed-604f524f0595/lessons/444a6d18-33cd-46a9-83cb-6fdbf53261fd/concepts/c8f7618a-637d-427a-b8df-83a0bc45a4cf)**

The "hue" parameter can also be used in a similar fashion in the boxplot, violinplot, and pointplot functions to add a categorical third variable to those plots in a clustered fashion. As a special note for pointplot, the default rendering aligns all levels of the "hue" categorical variable vertically. Use the "dodge" parameter to shift the levels in a clustered fashion:

ax = sb.pointplot(data = df, x = 'cat\_var1', y = 'num\_var2', hue = 'cat\_var2',

dodge = 0.3, linestyles = "")

**[[](https://classroom.udacity.com/nanodegrees/nd002-ent/parts/9c2eb5bf-2bda-4e0c-85bf-d997082a8571/modules/22d4ed2d-b711-4e1d-8fed-604f524f0595/lessons/444a6d18-33cd-46a9-83cb-6fdbf53261fd/concepts/c8f7618a-637d-427a-b8df-83a0bc45a4cf)](https://classroom.udacity.com/nanodegrees/nd002-ent/parts/9c2eb5bf-2bda-4e0c-85bf-d997082a8571/modules/22d4ed2d-b711-4e1d-8fed-604f524f0595/lessons/444a6d18-33cd-46a9-83cb-6fdbf53261fd/concepts/c8f7618a-637d-427a-b8df-83a0bc45a4cf)**

As a final example, a **line plot** can be adapted from previous code showing how to create frequency polygons for levels of a categorical variable. In this case as well, we create a custom function to send to a **FacetGrid** object's map function that computes the means in each bin, then plots them as lines via errorbar.

**def** **mean\_poly**(x, y, bins = 10, \*\*kwargs):

""" Custom adapted line plot code. """

*# set bin edges if none or int specified*

**if** type(bins) == int:

bins = np.linspace(x.min(), x.max(), bins+1)

bin\_centers = (bin\_edges[1:] + bin\_edges[:-1]) / 2

*# compute counts*

data\_bins = pd.cut(x, bins, right = **False**,

include\_lowest = **True**)

means = y.groupby(data\_bins).mean()

*# create plot*

plt.errorbar(x = bin\_centers, y = means, \*\*kwargs)

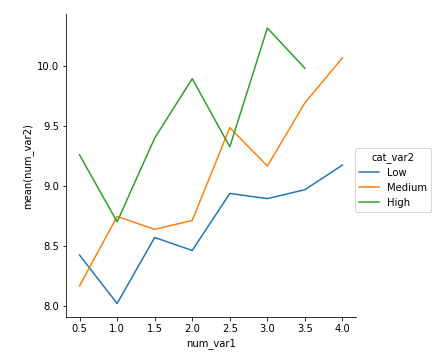
bin\_edges = np.arange(0.25, df['num\_var1'].max()+0.5, 0.5)

g = sb.FacetGrid(data = df, hue = 'cat\_var2', size = 5)

g.map(mean\_poly, "num\_var1", "num\_var2", bins = bin\_edges)

g.set\_ylabels('mean(num\_var2)')

g.add\_legend()

**[[](https://classroom.udacity.com/nanodegrees/nd002-ent/parts/9c2eb5bf-2bda-4e0c-85bf-d997082a8571/modules/22d4ed2d-b711-4e1d-8fed-604f524f0595/lessons/444a6d18-33cd-46a9-83cb-6fdbf53261fd/concepts/c8f7618a-637d-427a-b8df-83a0bc45a4cf)](https://classroom.udacity.com/nanodegrees/nd002-ent/parts/9c2eb5bf-2bda-4e0c-85bf-d997082a8571/modules/22d4ed2d-b711-4e1d-8fed-604f524f0595/lessons/444a6d18-33cd-46a9-83cb-6fdbf53261fd/concepts/c8f7618a-637d-427a-b8df-83a0bc45a4cf)**

# Plot Matrices

To move back to bivariate exploration for a bit, you might come out of your initial univariate investigation of the data wanting to look at the relationship between many pairs of variables. Rather than generate these bivariate plots one by one, a preliminary option you might consider for exploration is the creation of a plot matrix. In a plot matrix, a matrix of plots is generated. Each row and column represents a different variable, and a subplot against those variables is generated in each plot matrix cell. This contrasts with faceting, where rows and columns will subset the data, and the same variables are depicted in each subplot.

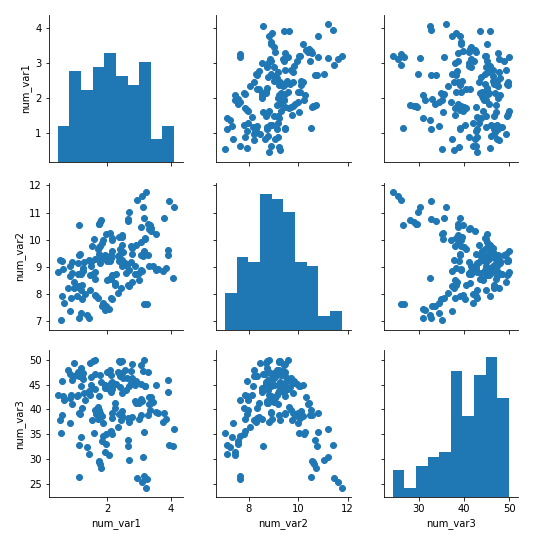
Seaborn's **[PairGrid](https://seaborn.pydata.org/generated/seaborn.PairGrid.html" \t "_blank)** class facilitates the creation of this kind of plot matrix.

g = sb.PairGrid(data = df, vars = ['num\_var1', 'num\_var2', 'num\_var3'])

g.map\_diag(plt.hist)

g.map\_offdiag(plt.scatter)

By default, **PairGrid** only expects to depict numeric variables; a typical invocation of **PairGrid** plots the same variables on the horizontal and vertical axes. On the diagonals, where the row and column variables match, a histogram is plotted. Off the diagonals, a scatterplot between the two variables is created.

**[[](https://classroom.udacity.com/nanodegrees/nd002-ent/parts/9c2eb5bf-2bda-4e0c-85bf-d997082a8571/modules/22d4ed2d-b711-4e1d-8fed-604f524f0595/lessons/444a6d18-33cd-46a9-83cb-6fdbf53261fd/concepts/d0c00cb4-0d15-455e-8906-c838b653c84c)](https://classroom.udacity.com/nanodegrees/nd002-ent/parts/9c2eb5bf-2bda-4e0c-85bf-d997082a8571/modules/22d4ed2d-b711-4e1d-8fed-604f524f0595/lessons/444a6d18-33cd-46a9-83cb-6fdbf53261fd/concepts/d0c00cb4-0d15-455e-8906-c838b653c84c)**

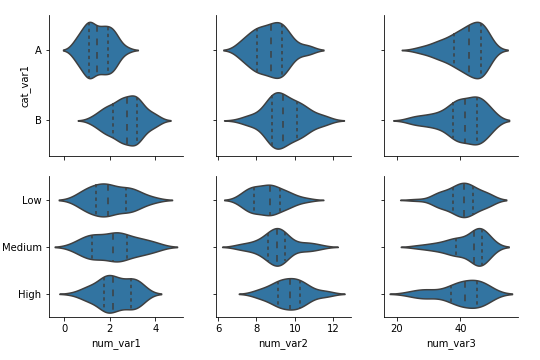
The **[pairplot](https://seaborn.pydata.org/generated/seaborn.pairplot.html" \t "_blank)** function can also be used to render this common use case in a single call.

For other relationships, the flexibility of **PairGrid** shines. For example, if we want to look at the relationship between the numeric and categorical variables in the data, we need to set the different variable types on the rows and columns, then use an appropriate plot type for all matrix cells.

g = sb.PairGrid(data = df, x\_vars = ['num\_var1', 'num\_var2', 'num\_var3'],

y\_vars = ['cat\_var1','cat\_var2'])

g.map(sb.violinplot, inner = 'quartile')

**[[](https://classroom.udacity.com/nanodegrees/nd002-ent/parts/9c2eb5bf-2bda-4e0c-85bf-d997082a8571/modules/22d4ed2d-b711-4e1d-8fed-604f524f0595/lessons/444a6d18-33cd-46a9-83cb-6fdbf53261fd/concepts/d0c00cb4-0d15-455e-8906-c838b653c84c)](https://classroom.udacity.com/nanodegrees/nd002-ent/parts/9c2eb5bf-2bda-4e0c-85bf-d997082a8571/modules/22d4ed2d-b711-4e1d-8fed-604f524f0595/lessons/444a6d18-33cd-46a9-83cb-6fdbf53261fd/concepts/d0c00cb4-0d15-455e-8906-c838b653c84c)**

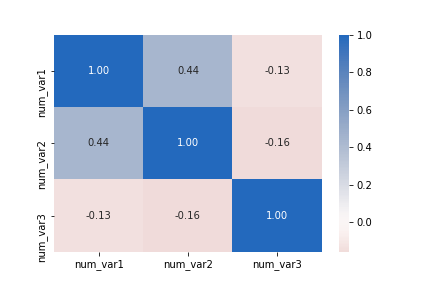
When you choose to create a plot matrix, be aware that the time it takes to render the plot depends on the number of data points you have and the number of variables you want to plot. Increasing the number of variables increases the number of plots that need to be rendered in a quadratic fashion. In addition, increasing the number of variables means that the individual subplot size needs to be reduced in order to fit the matrix width on your screen. That means that, if you have a lot of data, it might be difficult to see the relationships between variables due to overplotting, and it will take a long time to complete. One recommended approach is to take a random subset of the data to plot in the plot matrix instead. Use the plot matrix to identify interesting variable pairs, and then follow it up with individual plots on the full data.

**Correlation Matrices**

For numeric variables, it can be useful to create a correlation matrix as part of your exploration. While it's true that the [**.corr**](https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.corr.html) function is perfectly fine for computing and returning a matrix of correlation coefficients, it's not too much trouble to plot the matrix as a heat map to make it easier to see the strength of the relationships.

sb.heatmap(df.corr(), annot = **True**, fmt = '.2f', cmap = 'vlag\_r', center = 0)

Instead of using the default sequential color map, a diverging color map is specified and its center is set to 0. That way, we can use hue to tell if a correlation is positive or negative, and see its strength from the hue's intensity.

**[[](https://classroom.udacity.com/nanodegrees/nd002-ent/parts/9c2eb5bf-2bda-4e0c-85bf-d997082a8571/modules/22d4ed2d-b711-4e1d-8fed-604f524f0595/lessons/444a6d18-33cd-46a9-83cb-6fdbf53261fd/concepts/d0c00cb4-0d15-455e-8906-c838b653c84c)](https://classroom.udacity.com/nanodegrees/nd002-ent/parts/9c2eb5bf-2bda-4e0c-85bf-d997082a8571/modules/22d4ed2d-b711-4e1d-8fed-604f524f0595/lessons/444a6d18-33cd-46a9-83cb-6fdbf53261fd/concepts/d0c00cb4-0d15-455e-8906-c838b653c84c)**

Don't forget that correlations are computed based on strength of linear relationship. Compare the correlation between "num\_var2" and "num\_var3" to the corresponding cell in the first plot matrix as an example of how a correlation statistic might be misleading.

# Feature Engineering

This is not so much an additional technique for adding variables to your plot, but a reminder that feature engineering is a tool that you can leverage as you explore and learn about your data. As you explore a dataset, you might find that two variables are related in some way. Feature engineering is all about creating a new variable with a sum, difference, product, or ratio between those original variables that may lend a better insight into the research questions you seek to answer.

For example, if you have one variable that gives a count of crime incidents, and a second one that gives population totals, then you may want to engineer a new variable by dividing the former by the latter, obtaining an incident rate. This would account for a possible relationship between the original features where if there are more people, there might naturally be more chances for crimes to occur. If we looked at the raw counts rather than the incident rate, we risk just seeing information about population sizes rather than what we might really want.

Another way that you can perform feature engineering is to use the cut function to divide a numeric variable into ordered bins. When we split a numeric variable into ordinal bins, it opens it up to more visual encodings. For example, we might facet plots by bins of a numeric variable, or use discrete color bins rather than a continuous color scale. This kind of discretization step might help in storytelling by clearing up noise, allowing the reader to concentrate on major trends in the data. Of course, the bins might also mislead if they're spaced improperly – check out [**this PolicyViz post**](https://policyviz.com/2017/11/02/choosing-map-bins/) if you'd like to see a deeper discussion in the context of map-based visualizations.

## How Much is Too Much?

By now, you've seen a lot of ways of expanding the number of variables that can be depicted in a single visualization. The basic positional axes can handle two variables: one on the horizontal and one on the vertical. You can facet by both columns and rows to add up to two variables. Encodings in shape, size, and color could add as many as three more. However, you should try and resist the temptation to overuse your newfound power, and to instead practice some restraint in the number of variables that you include in any one plot.

When you have only two variables plotted, A and B, you have only one relationship to understand. Add in a third variable C, and you have three pairwise relationships: A vs. B, A vs. C, and B vs. C. You also need to consider one interaction effect between all three variables: Does the value of C affect the relationship between A and B? With a fourth variable, you have six possible pairwise relationships and four different three-variable interaction effects. This exponential explosion of possible relationships with the number of variables means that there is a potential for cognitive overload if the data isn't conveyed clearly.

This is why it is so important to approach data exploration systematically, rather than just throw as many variables together as possible immediately. When you move from univariate visualizations to bivariate visualizations, you augment your previous understanding of individual distributions by seeing how they relate to one another. If you look at pairwise visualizations before putting together a trivariate plot, then you will have a clear view to how the interaction, if present, changes your previous understanding of the marginal pairwise relationship.

When you move on to explanatory data visualizations, try to limit the number of variables that are introduced at the same time and make sure that the encoding choices convey the main findings to your reader in the clearest way possible. While it's good to keep a soft limit of about three or four variables in a single visualization, you can exceed this if the trends are clear or you introduce features to your reader in a systematic way.