Chapter 5 Improving perception; Best practices for visualization

Steve Elston

April 14, 2020

Introduction to Perception and Visualization

In this chapter we will investigate techniques to maximize the information a viewer perceives from data visualizations. Creating visualization which optimize human perception can highlight important insights. When faced with trying to understand complex data sets the limits of human perception become a significant factor. Applying the results of the considerable research on human perceptions for data visualization can help the process considerably.

This chapter explores using plot aesthetics to improve perception of complex relationships. We take a very broad view of the term 'aesthetic' here. By plot aesthetics we mean properties of a visualization which highlight aspects of the data relationships. Additionally, aesthetics are used to project additional dimensions of complex data onto the 2-dimensional plot surface. In summary, applying aesthetics with a bit of cleverness can lead to interesting insights about a data set.

In each section of this chapter we will explore one aspect of using Python visualization tools to effectively understand the relationships in complex data sets. These sections are ordered by how good human perception is for those aesthetics:

1. **Easy to perceive plot aesthetics:** We will start with generally easy to perceive aesthetics. These aesthetics are the most powerful at enhancing a viewer's perception. If possible, these aesthetics should be preferred when creating visualizations.
2. **Aesthetics with moderate perceptive power:** There are certain aesthetics, which are effective, but only when used within their limits.
3. **Aesthetics with limited perceptive power:** We will review some commonly used aesthetics which have limited perception power. Such aesthetics must be used sparingly. None the less, these aesthetics can still be useful in highlighting certain relationships in complex data sets.

There are many possible plot properties or aesthetics one can use for visualization. This chapter is by no means comprehensive. Specific examples of aesthetics discussed here, the applicable data types, and the relative perceptive power are summarized in the table below:

Property or Aesthetic	Perception	Data Types
Aspect ratio	Good	Numeric
Regression lines	Good	Numeric plus categorical
Marker position	Good	Numeric
Bar length	Good	Counts, numeric
Sequential color palette	Moderate	Numeric, ordered categorical
Marker size	Moderate	Numeric, ordered categorical
Line types	Limited	Categorical

Property or Aesthetic	Perception	Data Types
Qualitative color palette	Limited	Categorical
Marker shape	Limited	Categorical
Area	Limited	Numeric or categorical
Angle	Limited	Numeric

Limits of perception Don't over do it! All aesthetics have limitations. Using too many colors, shapes, line styles, etc, can be confusing. Poor practice will obfuscate rather than enlighten.

Easy to Percieve Aesthetics

In this section we will explore some plot aesthetics and attributes which maximize human perception. By employing these aesthetics or plot attributes you can create visualizations that are more likely to highlight key relationships of your data set.

For the running example in this chapter we will work with a data set containing the prices and characteristics of a number of automobiles. The ultimate goal is to understand the relationship between the price and fuel economy of a car and its characteristics.

```python
import pandas as pd
import numpy as np
import statsmodels.api as sm
from math import log, sqrt
import seaborn as sns
from math import log, sqrt, sin
import matplotlib.pyplot as plt
#%matplotlib inline

auto_price = pd.read_csv('../data//AutoPricesClean.csv')
```

Aspect Ratio and Banking

Changing the **aspect ratio** has a significant influence on how a viewer perceives a chart. The correct aspect ratio can help highlight important relationships in complex data sets. Conversely, poorly chosen or exaggerated aspect ratios can greatly distort human perception of the displayed relationship.

We express aspect ratio as follows:

$$aspect\ ratio = \frac{width}{height}\ \ : 1$$

The aspect ratio can be expressed either way. A tall narrow plot might have an aspect ration of 1:4. Whereas, a wide low plot might have an aspect ration of 4:1.

The key to understanding how the aspect ratio affects perception is the **banking angle** (Cleveland 1993). The banking angle is the average absolute angle of the curve displayed in a graph. Cleveland has shown that finding an aspect ratio which gives a banking angle of approximately 45° optimizes perception of changes in slope.

Let's look at an example of how aspect ratio and banking angle can change perception of a plot. The annual count of sunspots time series ("Yearly Sunspot Data," n.d.) is one of the longest scientific time series in existence. The code in the cell below contains the sunspot count from 1700 to 1980.

```
sunspots_data = sm.datasets.sunspots.load_pandas().data
sunspots_data.head()
```

```
##      YEAR  SUNACTIVITY
## 0  1700.0          5.0
## 1  1701.0         11.0
## 2  1702.0         16.0
## 3  1703.0         23.0
## 4  1704.0         36.0
```

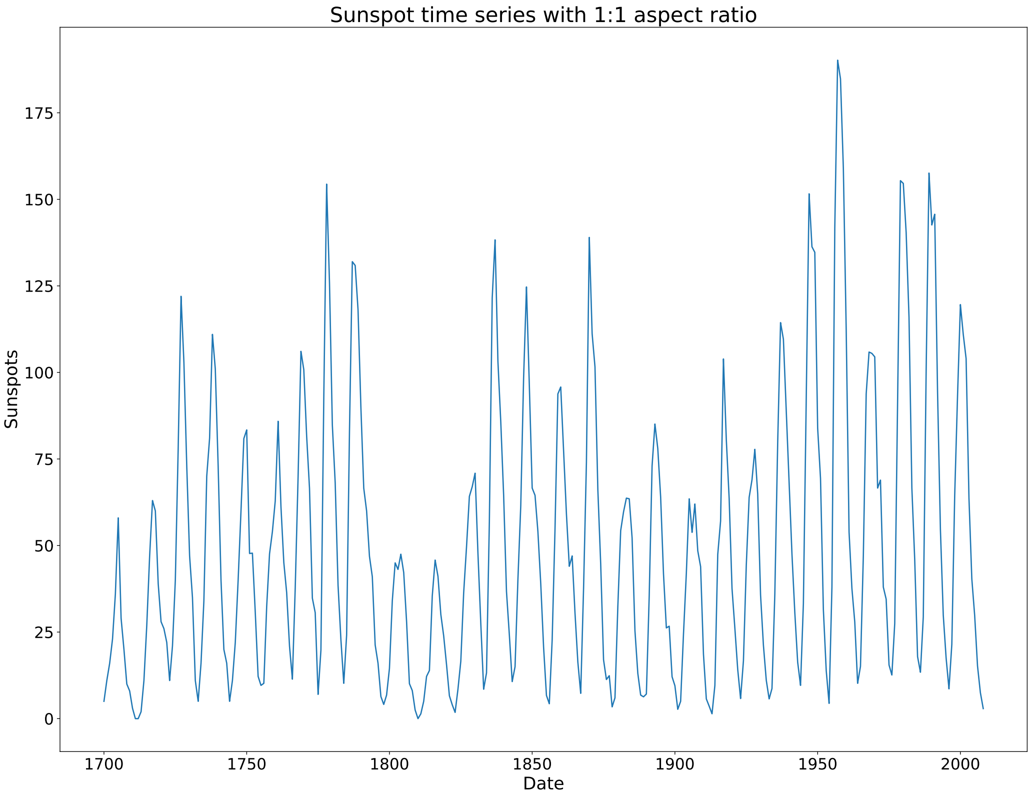
The two columns are the year and the number of sunspots.

We will make a first plot of this time series with an aspect ratio of about 1:1. The code below makes a **time series plot** with the date on the horizontal axis and the number of sunspots in each year on the vertical axis. Since these observations are ordered in time, a line plot is used.

```
fig, ax = plt.subplots(figsize=(20, 20))
fig.subplots_adjust(bottom=0.3)
ax = sns.lineplot('YEAR', 'SUNACTIVITY', data=sunspots_data, ci=None, ax=ax)
```

```
## C:\Users\StevePC2\Anaconda3\lib\site-packages\seaborn\_decorators.py:43: FutureWarning: Pass the foll
##   FutureWarning
```

```
_=ax.set_title('Sunspot time series with 1:1 aspect ratio', fontsize=24)
_=ax.set_xlabel('Date', fontsize=20)
_=ax.set_ylabel('Sunspots', fontsize=20)
_=ax.tick_params(labelsize=18)
plt.show()
```



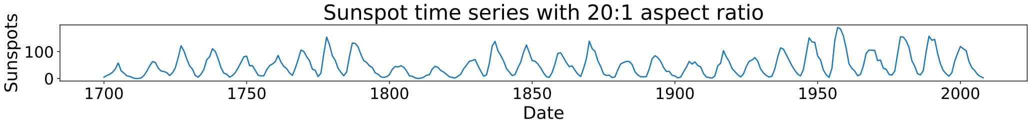
There are two points to notice about the time series displayed here: 1. The number of sunspots is periodic in time. This is the well-known **sunspot cycle**, with a period of approximately 11 years.
2. Given this aspect ratio, each cycle appears to be symmetric. But, notice that the average slope or banking angle in this plot is high, much greater than 45°.

What happens if we change the aspect ratio to about 15:1 to improve the banking angle. The code in the cell below creates just such a plot.

```
fig, ax = plt.subplots(figsize=(20, 3))
fig.subplots_adjust(bottom=0.4,top=0.7)
ax = sns.lineplot('YEAR', 'SUNACTIVITY', data=sunspots_data, ci=None, ax=ax)
```

```
## C:\Users\StevePC2\Anaconda3\lib\site-packages\seaborn\_decorators.py:43: FutureWarning: Pass the fol
##   FutureWarning
```

```
_=ax.set_title('Sunspot time series with 20:1 aspect ratio', fontsize=24)
_=ax.set_xlabel('Date', fontsize=20)
_=ax.set_ylabel('Sunspots', fontsize=20)
_=ax.tick_params(labelsize=18)
plt.show()
```



You can see that the banking angle in the plot is now considerably lower than before. The banking angle for most of each cycle is much closer to 45°. The lower banking angle brings a new, and important, property of the time series to light. The rise of the cycle is generally much faster than the drop in activity. You can see this in the asymmetry of most of the cycles.

> **Exercise 5-1:** To better understand the effects of changing aspect ratio and therefore banking angle, try the following. Create 3 scatter plots using the Seaborn `lmplot` function with the following properties: 1. Place price on the vertical axis and city MPG on the horizontal axis.
> 2. Use fuel type for the `hue` argument. Include a second order polynomial curve.
> 3. The aspect ratio of these three plots should be a) 1:1, b) 1:4, c) 4:1.
> 4. For `lmplot` you will need to set an appropriate `height` and `aspect` argument to make each of these plots.
> 5. Set the `truncate` argument to true.
> Reference to the Seaborn documentation will be helpful. Notice that the relationship between the price and city MPG does not follow a straight line for either fuel type for the 1:1 aspect ratio. How does increasing or decreasing the aspect ratio change your perception of these relationships?

> **Exercise 5-2:** Poor human perception of angle was mentioned in the introduction to this chapter. To demonstrate this create three line plots with the following end points using the same set of axes: a) (0,0) and (10,8), b) (0,5) and (10,14), and c) (0,10) and (10,20). Make sure the aspect ratio of the plots are approximately 1:1. Notice these plots all have the same slopes. But, how well can you perceive these different slopes?

Marker Position and Scatter Plots

Human perception of even small differences in position is quite acute. This acute perception makes scatter plots and other plots using marker position quite effective. As a result, scatter plots are one of the most widely used, useful, and powerful plot types, despite the apparent simplicity.

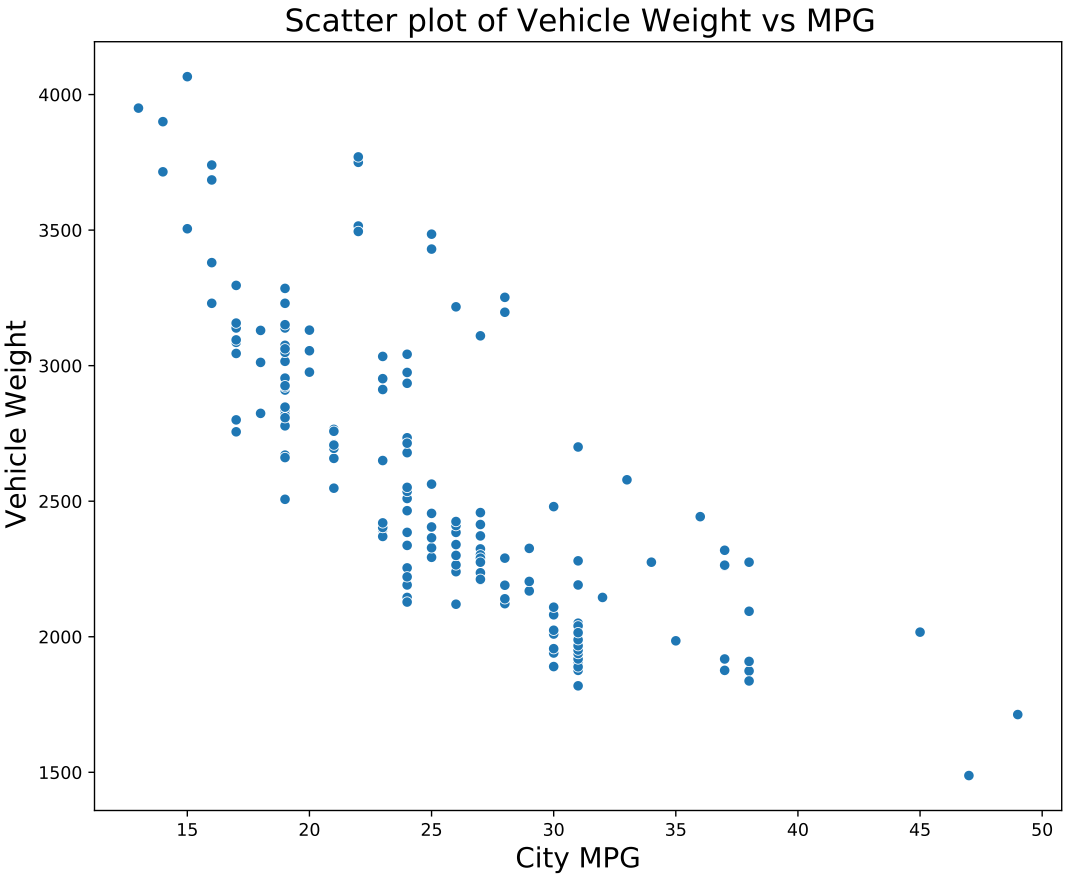
Scatter plots show the relationship between two variables in the form of marker positions on the plot. In simple terms, the values along a horizontal axis are plotted against a vertical axis.

As a reference point for a running example in this chapter we will start with a basic scatter plot. The code below creates a plot of city MPG on the horizontal axis and vehicle curb weight on the vertical axis.

```
fig = plt.figure(figsize=(10, 8))
ax = sns.scatterplot('city_mpg', 'curb_weight', data=auto_price)
```

```
## C:\Users\StevePC2\Anaconda3\lib\site-packages\seaborn\_decorators.py:43: FutureWarning: Pass the fol
##   FutureWarning
```

```
_=ax.set_title('Scatter plot of Vehicle Weight vs MPG', fontsize=18) # Give the plot a main title
_=ax.set_xlabel('City MPG', fontsize=16) # Set text for the x axis
_=ax.set_ylabel('Vehicle Weight', fontsize=16)# Set text for y axis
plt.show()
```



In this plot you can see very small differences in the vehicle weight for each value of city MPG. These values are quite small in some cases, yet you can easily see the differences. Notice that city MPG is quantized in 1 MPG units.

> **Exercise: 5-3** You can demonstrate the perceptual power of position by creating the following plot. Create a data frame with a series of x,y pairs of points, running from (0.0,0.0) to (4.0,4.0) by increments of 1.0. In the same data frame create a set of x,y pairs with the same values plus a small offset in the x and y. The offsets start at 0.00001 and end at 0.1. As an example, the first set of two x,y pairs will have values, (0.0,0.0) and (0.00001,0.00001), and the last set of x,y pairs have values, (4.0,4.0) and (4.1,4.1). Make a scatter plot of these x,y values in the data frame. Make sure the size of the plot is large enough to clearly see as many differences as possible. Use the `figsize` argument to set the plot area. Which of these differences can you see? What does this tell you about perception of small differences in position on a plot?

Bar Length and Ordered Bar plots

Length is another plot aesthetic or property for which human perception is quite sensitive. Most people can discern small differences in length. Thus, perception of relationships in bar plots can be quite good.

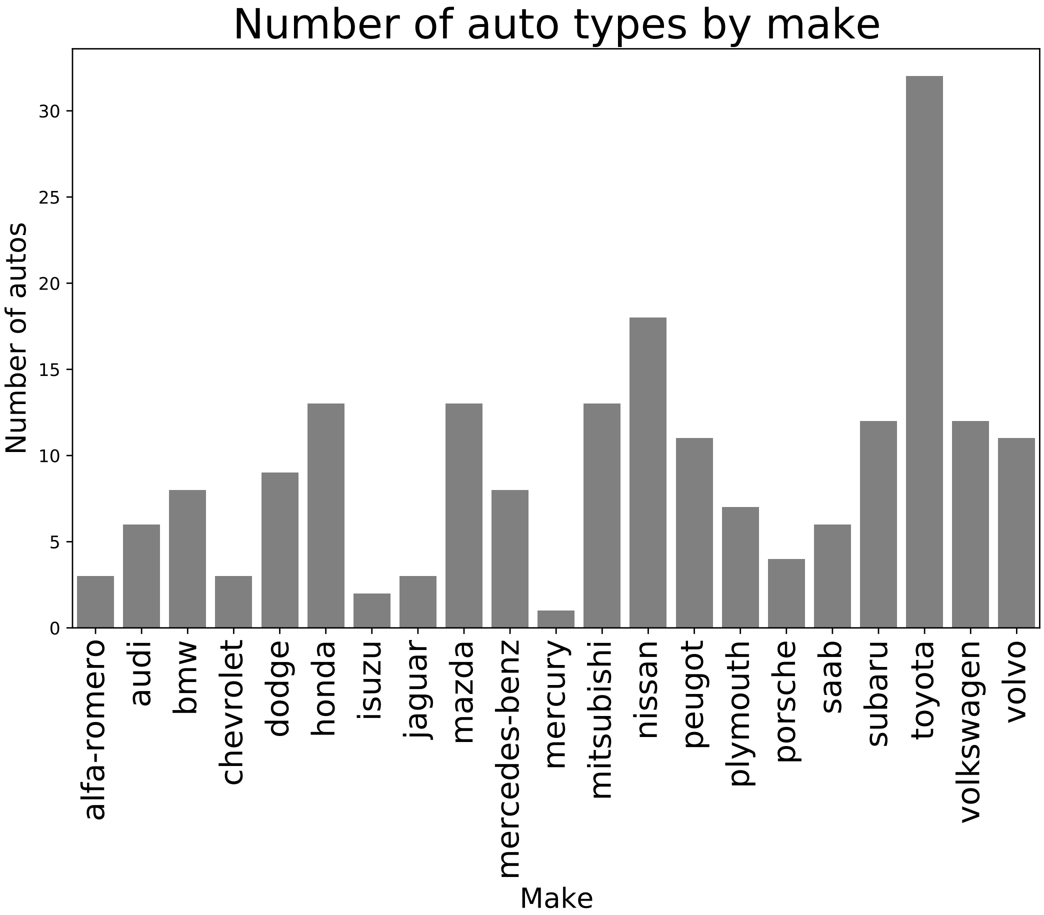
Bar plots are used to display the counts of unique values of a categorical variable. The length of the bar represents the count for each unique category of the variable. Small differences in the counts, resulting in small differences in the bar lengths are quite perceptible.

The code in the cell below uses the Seaborn `countplot` function to create a bar chart of the number of cars by make.

```
fig, ax = plt.subplots(figsize=(10, 8))
fig.subplots_adjust(bottom=0.3)
ax = sns.countplot('make', data=auto_price, color='gray', ax=ax)
```

```
## C:\Users\StevePC2\Anaconda3\lib\site-packages\seaborn\_decorators.py:43: FutureWarning: Pass the foll
##   FutureWarning
```

```
_=ax.set_title('Number of auto types by make', fontsize=24) # Give the plot a main title
_=ax.set_xlabel('Make', fontsize=16) # Set text for the x axis
_=ax.set_ylabel('Number of autos', fontsize=16)# Set text for y axis
_=ax.set_xticklabels(ax.get_xticklabels(), rotation=90, fontsize=18)
plt.show()
```



This chart is hard to interpret since the bars are not ordered by height. You cannot tell which makes have identical counts or which makes might have slightly higher or lower count. Even though human perception of differences in length is generally quite good, if the lengths being judged are separated by other bars, determining small differences in length is difficult.

What is the solution? The bars in the plot must be ordered by length. Placing the bars in increasing or decreasing order greatly enhances perception.

The code below orders the counts using **chained** Pandas methods:
1. The `value_counts` method computes the course. 2. The `reset_index` method fixes the index of the Pandas series in count order.

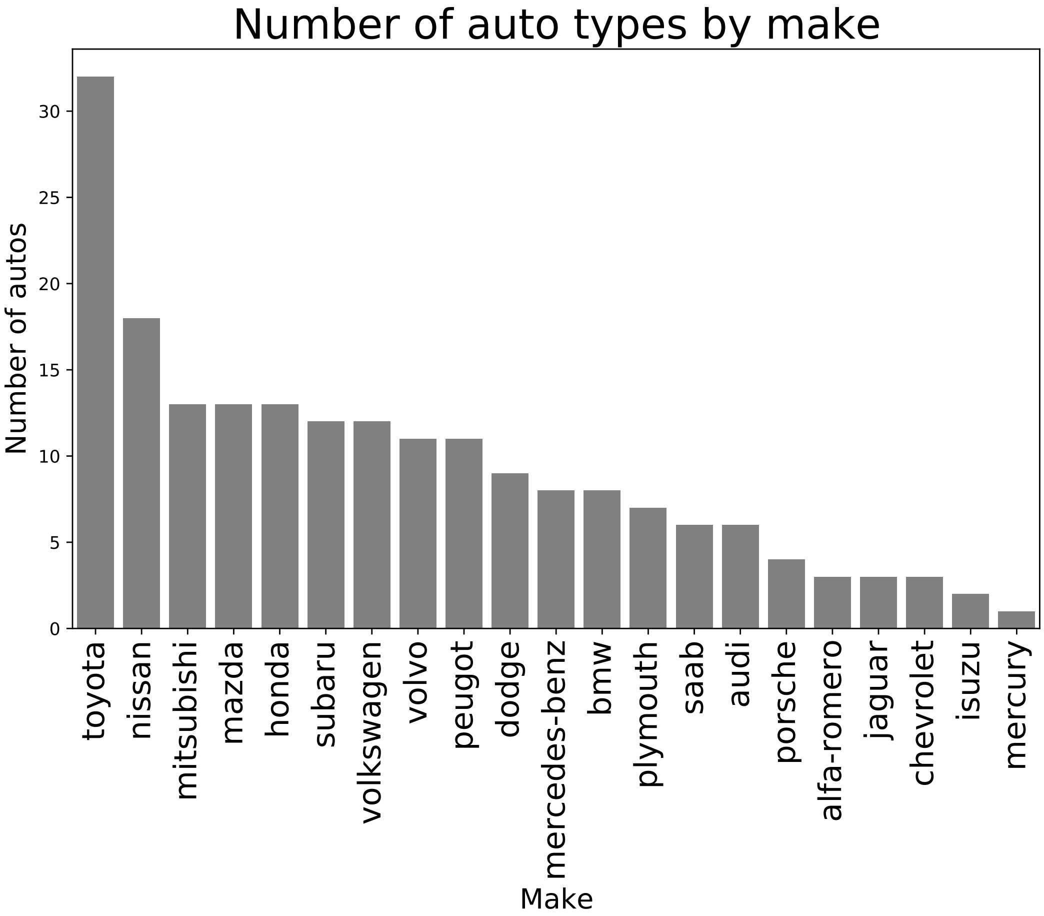
The Seaborn `barplot` function is similar to `countplot`, but works on numerical counts. Since the series object is ordered `barplot` creates an ordered bar plot.

```
counts = auto_price['make'].value_counts().reset_index()
print(counts.columns)
```

```
## Index(['index', 'make'], dtype='object')
```

```
fig, ax = plt.subplots(figsize=(10, 8))
fig.subplots_adjust(bottom=0.3)
ax = sns.barplot(x='index', y='make', data=counts, color='gray', ax=ax)
```

```
_=ax.set_title('Number of auto types by make', fontsize=24) # Give the plot a main title
_=ax.set_xlabel('Make', fontsize=16) # Set text for the x axis
_=ax.set_ylabel('Number of autos', fontsize=16)# Set text for y axis
_=ax.set_xticklabels(ax.get_xticklabels(), rotation=90, fontsize=18)
plt.show()
```



Now the bar plot is ordered. You can easily see the relationship of the number of auto models by make.

It is worth considering commonly used alternatives to a simple bar plot a stacked bar charts or pie chart: - **Stacked bar charts:** When multiple categories are stacked into single bar, the viewer often has difficulty determining the relative lengths of the sub-segments in the bars. Except in special cases, the alignment of the sub-segments will be somewhat random and hard to compare. The perceptual problem is similar to the un-ordered bar chart.
- **Pie charts:** Pie charts use area or angle, rather than length to show relative values. Human perception of both area and angle is quite poor in general. Interpreting pie charts depend on perception of either area, angle, or both. Further, when there are more than a few categories the small differences in the area of the slices become completely unnoticeable.

Is there a good alternative to stacked bar charts and pie charts? Yes. In the previous chapter we created a bar plot of one variable grouped by the `hue` argument variable. Such a plot is much easier to interpret that a stacked bar chart in general.

Exercise 5-4: To demonstrate the interpretation problems with pie charts, create a pie chart.

> As of this writing, there is no pie chart in Seaborn. Therefore you can use the pie chart method in Pandas, `dataframe.plot.pie`. Is it possible to interpret the differences in the numbers of models for each manufacturer? Why is this?

Aesthetics with Moderate Perceptive Power

Many plot aesthetics are effective if used correctly, and within their limitations. In this section we address a few widely used examples.

Sequential and Divergent Color Palettes

Use of **color** as an aesthetic in visualization is a complicated subject. While color is often used, it is also often abused and leads to misleading or uninterpretable results.

To structure our discussion, we will define two particular cases for the use of color in visualization:
- A **qualitative palette** is a palette of individual colors used to display categorical values. We will address this case latter.
- **Sequential palettes** and **divergent palettes** are a sequence of colors used to display a quantitative variable or ordered categorical variable. A sequential palette contains a color sequence of changing hue, which corresponds to a sequence of changing values of a quantitative variable, or ordered categorical variable. A divergent palette has two sequences of hues with a gap in the center. Divergent palettes are useful for quantitative variables with positive and negative values.

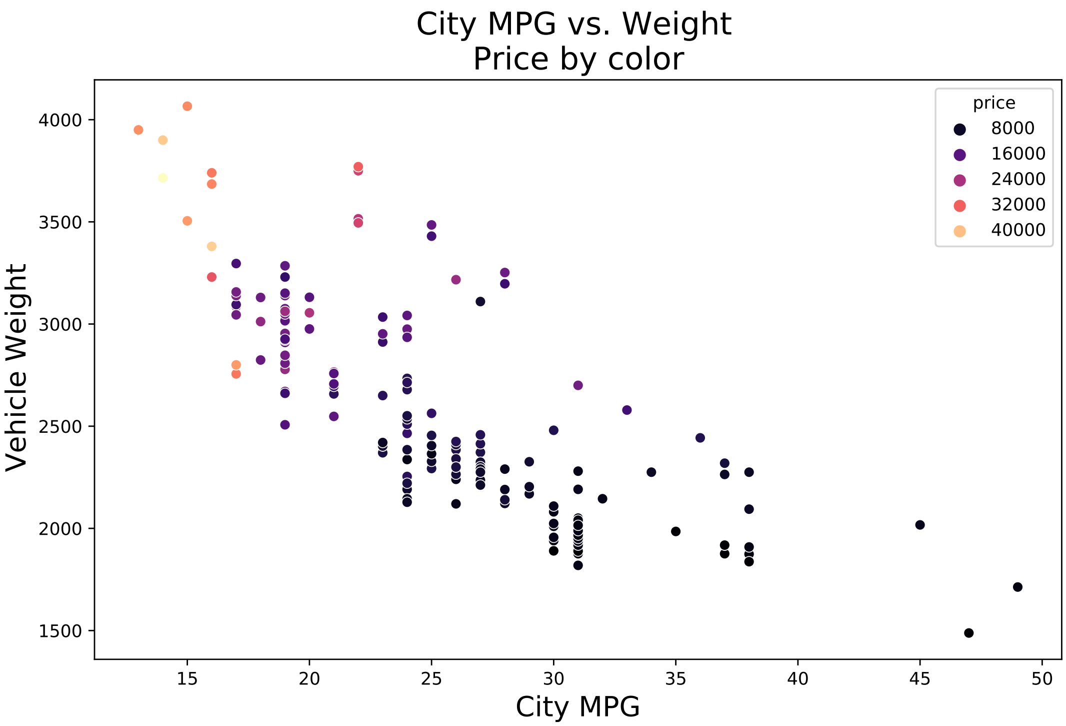
> **Limits of color:** Regardless of the approach there are some significant limitations: 1. A significant number of people are color blind. Red-green color blindness is most common, particularly in men. Keep this fact in mind when selecting a color palette. 2. Even the best sequential or divergent palettes show only relative value of numeric variables. Perception of exact numeric values is difficult, except in special cases.

Example with sequential color palette

When doing data visualization in Python you have the choice of several sets of well designed sequential and divergent color palettes. There is an extensive set of palette choices in Seaborn. Matplotlib contains another set of high quality palettes. These Matplotlib options, including **perceptually uniform palettes**, expertly designed to optimize human perception.

The code in the cell below creates a scatter plot with city MPG on the horizontal axis and vehicle curb weight on the vertical axis. The vehicle price shown uses the sequential `magma` palette from Matplotlib. Magma uses 'hotter' colors to show larger values of a quantitative variable.

```
ax = sns.scatterplot(x='city_mpg', y='curb_weight', data=auto_price,
                     hue = 'price', palette = 'magma')
_=ax.set_title('City MPG vs. Weight \nPrice by color', fontsize=18)
_=ax.set_xlabel('City MPG', fontsize=16)
_=ax.set_ylabel('Vehicle Weight', fontsize=16)
plt.show()
```



Examine the plot, noticing the general trend from the lower right to the upper left. The lowest weight, highest mileage, autos generally have the lowest prices. The highest weight, lowest mileage, autos generally have the highest prices. Within these general trends some deviation is noticeable, but the general trend is clear from the color progression of the palette.

While it is hard to draw quantitative conclusions from color progressions, they can help show general trends. This can be done effectively with the correct choice of palette. Best choice of palette to highlight a relationship can take some experimentation.

Sequential Color palettes and Heat Maps

A **heat map** or **raster plot** allows you to visualize data which can be arranged on a regular grid or lattice. The heat map uses a **sequential color map** to show the values, or heat, on the grid. Data arranged on a lattice is fairly common in many areas: spatial analysis, image analysis, genetic marker mapping, and correlation analysis.

Python's packages have several useful plotting methods to display lattice data, or data arranged in a N row X M column numeric array. In this case we will use the `heatmap` method from Seaborn.

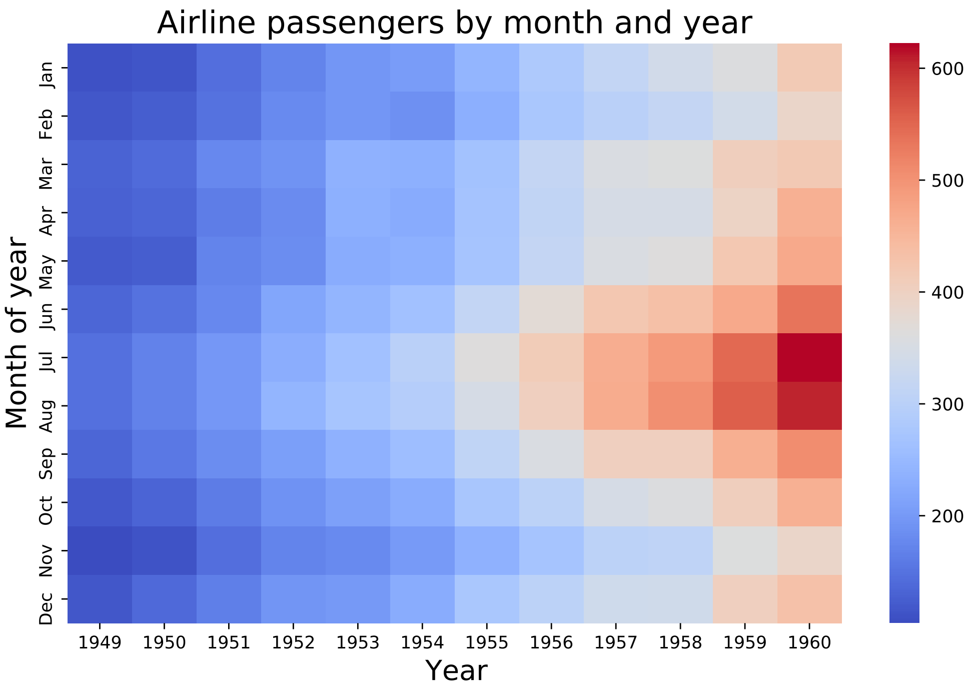
In this example we will create a heatmap of the number of airline passengers arranged by year horizontally and months vertically. These data are in the Seaborn example data sets.

```
airlines = sns.load_dataset("flights")
airlines.head()
```

```
##    year month  passengers
## 0  1949   Jan         112
## 1  1949   Feb         118
## 2  1949   Mar         132
## 3  1949   Apr         129
## 4  1949   May         121
```

The data frame is organized in tabular form, not a lattice. The Pandas `pivot` method is used to organize these data into the required lattice form. With the data correctly organized, the Seaborn `heatmap` function creates the display.

```
airlines = airlines.pivot("month", "year", "passengers")
ax = sns.heatmap(airlines, cmap='coolwarm')
_=ax.set_title('Airline passengers by month and year', fontsize=18)
_=ax.set_xlabel('Year', fontsize=16)
_=ax.set_ylabel('Month of year', fontsize=16)
plt.show()
```



The display shows several important relationships in these data.
- The number of airline passengers increases significantly with year.
- Air travel is more popular in the summer months.

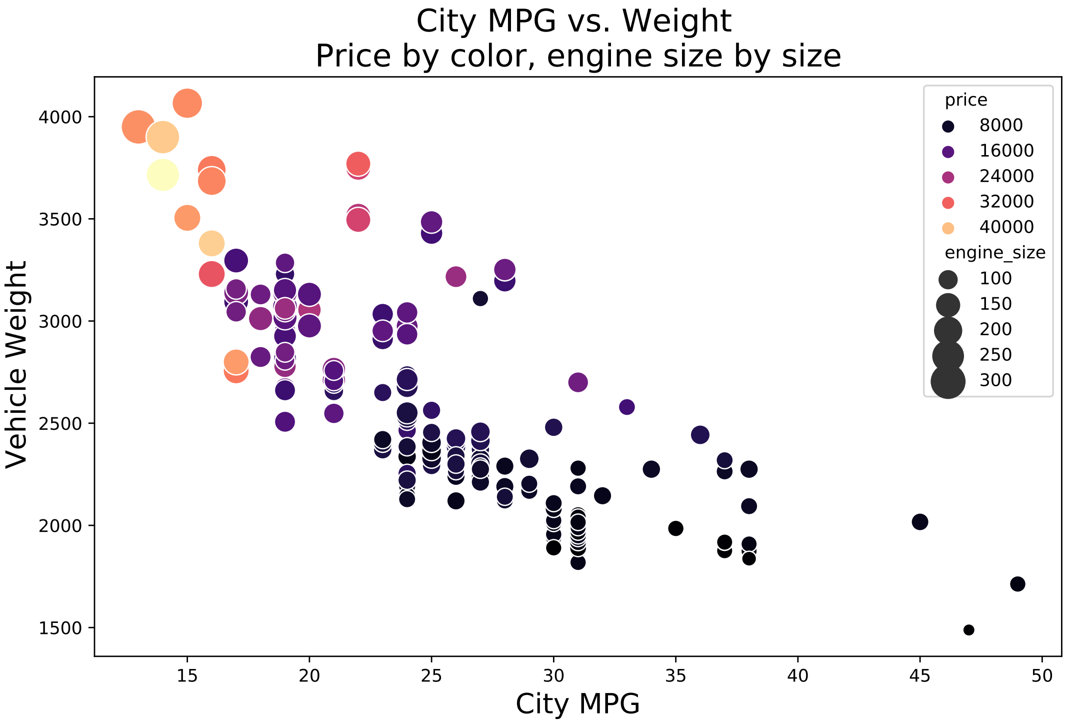
Marker Size

Another moderately effective aesthetic useful for quantitative variables is **marker size**. As with a well chosen color palette, used properly, marker size can highlight important trends in complex data sets. But there are also limitations. The viewer can generally perceive relative differences, but not actual values.

Several Seaborn functions, including `scatterplot`, have a `size` argument, which sets marker size by values of the variable assigned. The range of marker sizes must also be specified with the `sizes` argument. Markers which are too small cannot be seen easily. Whereas, if the markers are too large, they will overlap and obscure one another. Some experimentation is usually required for a specific application.

An example, setting marker size to the size of the engine, is shown in the code below.

```python
ax = sns.scatterplot(x='city_mpg', y='curb_weight', data=auto_price,
                     hue = 'price', palette = 'magma',
                     size = 'engine_size', sizes = (50.0, 400.0))
_=ax.set_title('City MPG vs. Weight \nPrice by color, engine size by size', fontsize=18)
_=ax.set_xlabel('City MPG', fontsize=16)
_=ax.set_ylabel('Vehicle Weight', fontsize=16)
plt.show()
```



Notice that the engine size generally increases with vehicle weight. As with price, the trend is clear from the changing marker sizes. However, perception of specific numeric values of engine size is not possible.

> **Exercise 5-5:** Consider how many dimensions of the auto data are projected onto the 2-dimensional surface of this plot. How can you best describe the relationship between all of these variables?

Aesthetics with Limited Perceptive Power

We have explored some aesthetics which are easy to perceive, and therefore quite powerful, and some others with moderate perceptive power. You may be surprised to learn that some commonly used plot aesthetics are not that easy to perceive. Without care, these aesthetics do not promote perception in the way a creator might think they will.

Line Plots and Line Type

We have already encountered line plots. Line plots connect discrete, ordered, data points by a line. It is common practice to use different line pattern types to differentiate data grouped by a categorical variable. While this approach can be effective, it is only useful for a limited number of lines on one graph. The use of too many similar line pattern on one plot leads to viewer confusion and poor perception of the data relationships. In summary, line pattern can be effective, but only when used sparingly.

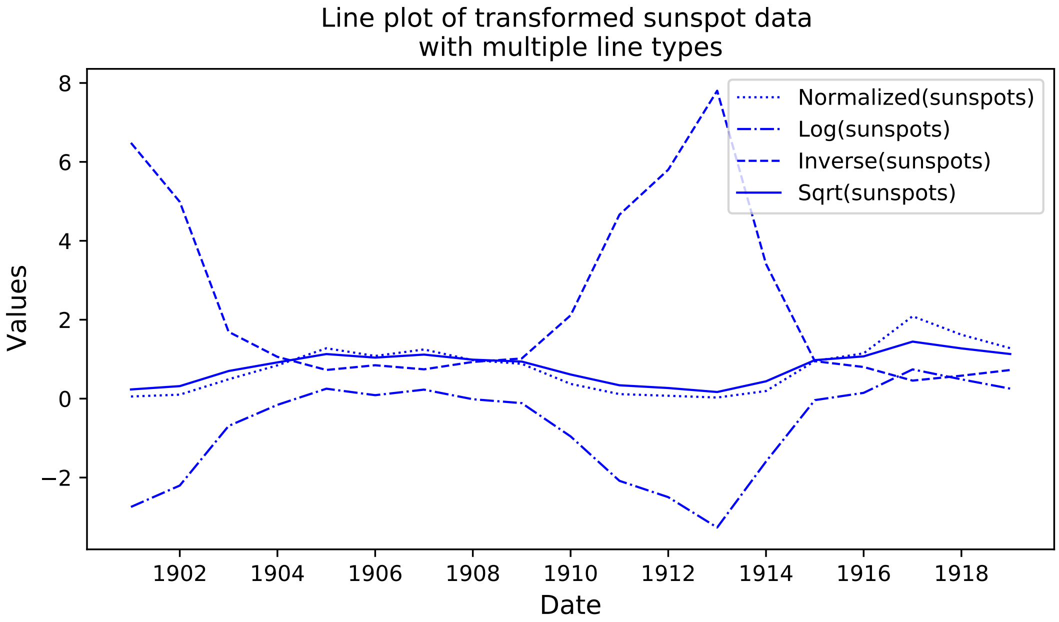
The code below creates a data frame using various transformations of the sunspot data.

```
## Compute some transformed variables
sunspot_mean = sunspots_data.loc[:,'SUNACTIVITY'].mean()
y1 = [z/sunspot_mean for z in sunspots_data.loc[:,'SUNACTIVITY']]
y2 = [log(z + 0.01) for z in y1]
y3 = [1.0/(z + 0.1) for z in y1]
y4 = [sqrt(z) for z in y1]
## Construct the data frame
df = pd.DataFrame({'Date':sunspots_data.loc[:,'YEAR'],'Normalized':y1, 'Log':y2, 'Inverse':y3, 'Sqrt':y
## Convert the floating point year to a Pandas datetime type
df.loc[:,'Date'] = pd.to_datetime(df.loc[:,'Date'].astype(int),format='%Y')
df.head()
```

```
##         Date  Normalized       Log   Inverse      Sqrt
## 0 1700-01-01    0.100498 -2.202755  4.987574  0.317015
## 1 1701-01-01    0.221096 -1.464921  3.114332  0.470209
## 2 1702-01-01    0.321594 -1.103843  2.371948  0.567093
## 3 1703-01-01    0.462292 -0.750158  1.778435  0.679921
## 4 1704-01-01    0.723587 -0.309808  1.214200  0.850639
```

Now, we will make a time series plot of these transformed variables using a different line type for each variable. You can find matplotlib line styles options in the documentation. The code below uses 4 line types to display the transformed variables on a time series plot.

```
styles = [':','-.','--','-'] # Some line styles
fig = plt.figure(figsize=(8, 4)) # define plot area
ax = fig.gca() # define axis
temp = df[(df['Date'] > '1900-01-01') & (df['Date'] <= '1919-12-01')]
## Iterate over the time series and line styles to plot
for col, style in zip(temp[['Normalized','Log','Inverse','Sqrt']], styles):
    _=ax.plot(temp.Date, temp[col], linestyle = style, color = 'b', label=(col+'(sunspots)'), linewidth
## Annotate the plot, including legend
_=ax.set_title('Line plot of transformed sunspot data \nwith multiple line types')
_=ax.set_xlabel('Date', fontsize=12) # Set text for the x axis
_=ax.set_ylabel('Values', fontsize=12)# Set text for y axis
_=ax.legend()
plt.show()
```



Examine the plot above and notice that with just 4 types, the lines on the plot are distinct. This is even the case for the normalized sunspots and the square root of the normalized sunspots, which run quite close together. But, it is not hard to imagine that adding too many line types to a plot will cause perception problems for the viewer.

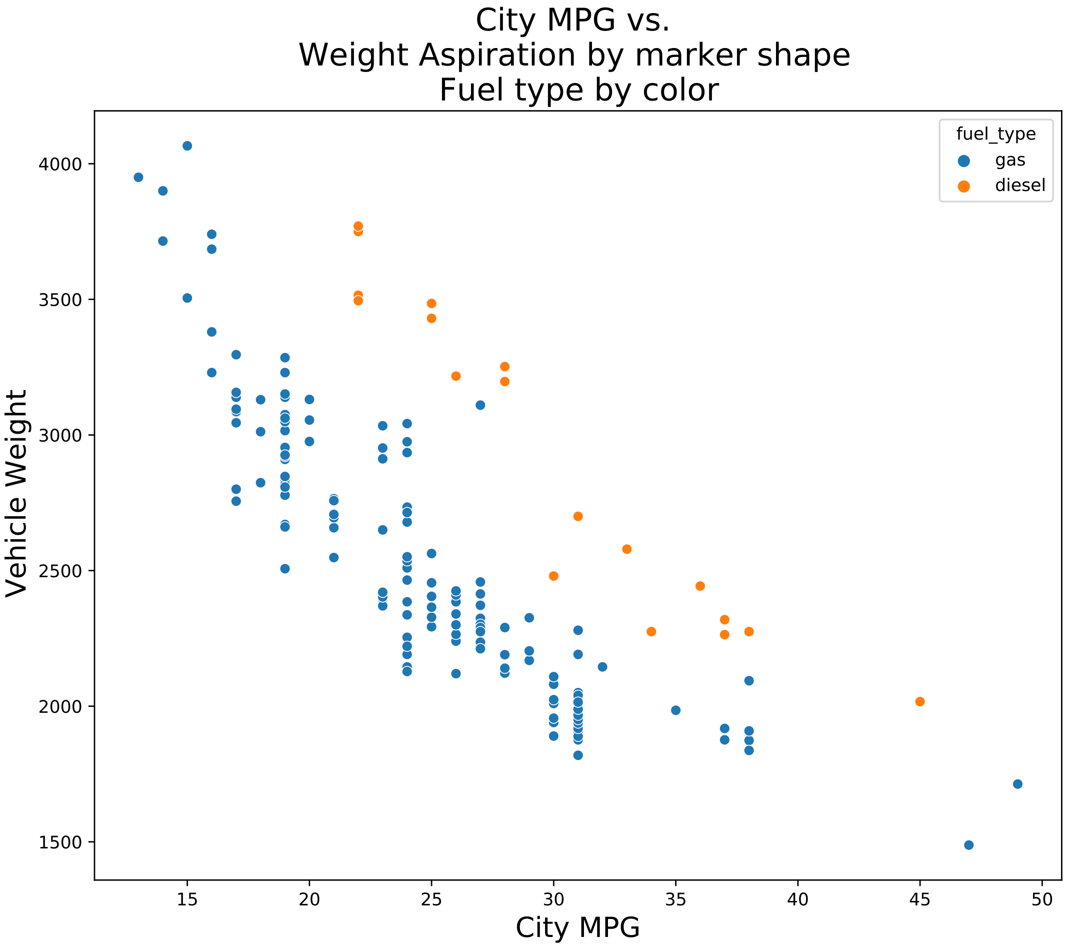
In principle, a combination of color and line type can be used to make lines distinct. However, this method must be used cautiously. The result can easily be *line spaghetti* which is difficult to understand, at best. The limits of perception are hit quickly.

Qualitative Color Palettes

Qualitative (or categorical) color palettes are often used to display data grouped by a categorical variable. If the number of categories are few and the colors chosen are distinctive, this method can be effective. However, for more than a few categories, or if the choice of color is poor, this method is not effective. The result is often a confusing array or symbols with hard to distinguish colors, which will confound the understanding of the viewer.

As an example where a qualitative palette works well consider the scatter plot shown below. There are only two categories and the colors are quite distinctive.

```
fig, ax = plt.subplots(figsize=(10, 8))
_=sns.scatterplot(x = 'city_mpg', y = 'curb_weight',
                  hue = 'fuel_type',
                  data=auto_price,
                  ax=ax)
_=ax.set_title('City MPG vs. \nWeight Aspiration by marker shape \nFuel type by color', fontsize=18)
_=ax.set_xlabel('City MPG', fontsize=16)
_=ax.set_ylabel('Vehicle Weight', fontsize=16)
plt.show()
```



We can easily see the distinctive relationships for both gas and diesel fueled cars. Using a few categorical colors from a well chosen palette can be quite effective.

> **Exercise 5-6:** What happens if a qualitative color palette is used with a large number of categories. To find out, create a scatter plot with city MPG on the horizontal axis and price on the vertical axis. Set the `hue` parameter to the make variable. Is it possible to interpret the colors by make? What does this tell you about the limitations of using a larger number of colors from a qualitative palette?

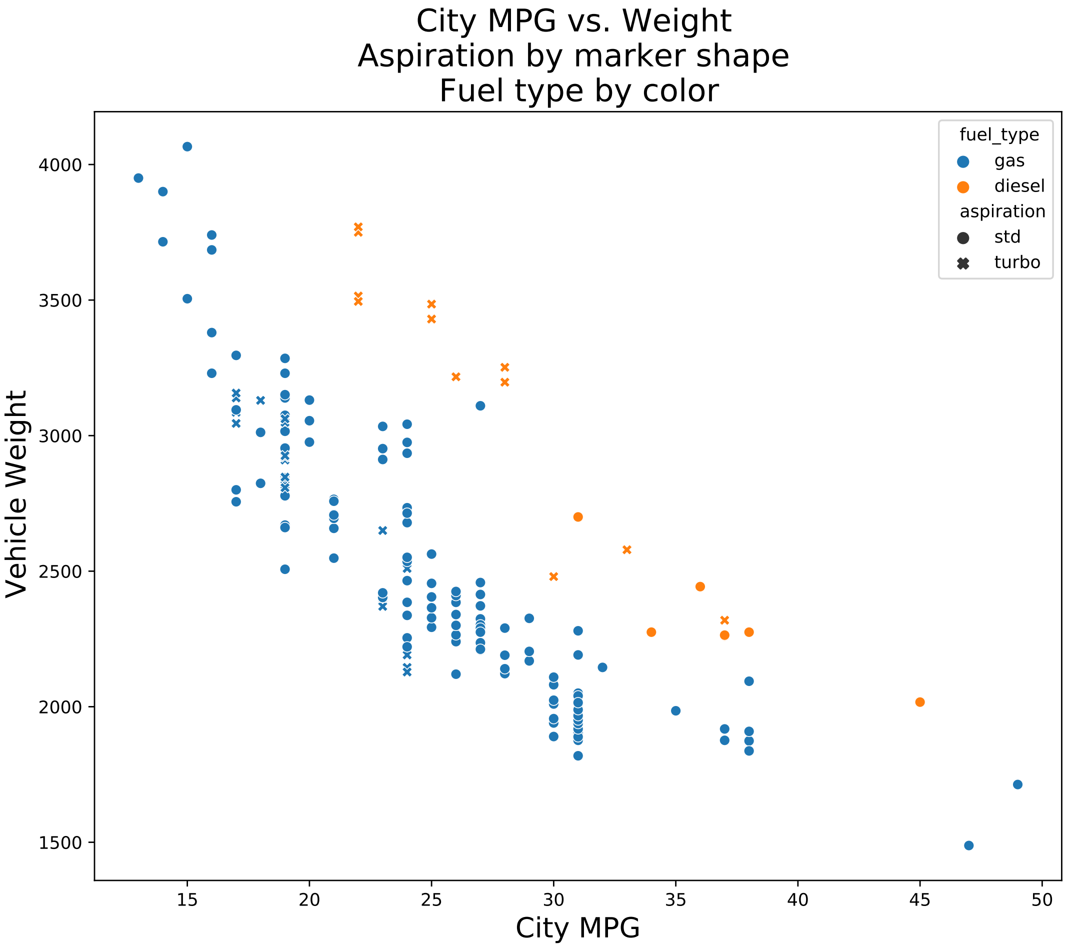
> **Exercise 5-7:** What are a number of alternatives to the hard to understand plot you created in the previous exercise>? The idea of ordered box plots can be applied to many data sets, including the automotive data. Create an ordered set of box plots of city MPG on the vertical axis and make on the horrizontal axis. Order the makes by the median MPG for each make. Additionally, create an ordered set of box plots of price on the vertical asis and make on the horrizontal axis. Order the makes by the median price for each make. What do these two box plots show you that was difficult to perceive in the scatter plot? What information is lost when using ordered box plots?

Marker Shape

Marker shape is another commonly used plot aesthetic for displaying categorical relationships. As with qualitative color palettes, this aesthetic is only useful when two conditions are met: 1. The number of categories is small. 2. Distinctive shape are chosen for the markers.

The code in the cell below uses the scatterplot function from the Seaborn package to plot aspiration as a marker type and color (hue) for fuel type.

```
ax=sns.scatterplot(x = 'city_mpg', y = 'curb_weight',
                        style = 'aspiration',
                        hue = 'fuel_type',
                        data=auto_price)
_=ax.set_title('City MPG vs. Weight \nAspiration by marker shape \nFuel type by color', fontsize=18)
_=ax.set_xlabel('City MPG', fontsize=16)
_=ax.set_ylabel('Vehicle Weight', fontsize=16)
plt.show()
```



The resulting plot above takes a bit of study to see any useful relationship. Several relationships can be noticed: 1. For the gas cars, it is clear that for the most part, only cars in the middle of the weight and fuel economy range have turbo aspiration. 2. Most diesel cars have turbo aspiration.

It is clear that without the color coding of the fuel type these interpretations could not be found. This illustrates a key aspect of visualization for complex data. Very often **multiple plot attributes are needed to understand relationships**.

> **Exercise 5-8:** Is it possible to project 5-dimensions of the auto data set onto the 2-dimensional plot surface? Create a scatter plot with price on the vertical axis and vehicle weight on the

horizontal axis. Use these aesthetics, a) vehicle curb weight as marker size, b) aspiration as marker color (hue), and c) fuel type as marker shape. You will likely need to try different marker size ranges. What can you tell about the relationships between these variables from this visualization.

Exercise 5-9: What happens when there are a large number of symbol shapes? To find out, create a scatter plot with city MPG on the horizontal axis, price on the vertical axis, the `hue` argument set to fuel type. Set the `style` argument to the engine type. Can you find any useful pattern in the resulting plot? What does this tell you about using a large number of marker shapes?

Summary

In this chapter we explored the following key points:
- Proper use of plot aesthetics enable projection of multiple dimensions of complex data onto the 2-dimensional plot surface.
- All plot aesthetics have limitations which must be understood to use them effectively.
- The effectiveness of a plot aesthetic varies with the type and the application.

Copyright 2020, 2021, Stephen F Elston. All rights reserved.

Bibliography

.

Cleveland, William S. 1993. *Visualizing Data*. Hobart Press.

"Yearly Sunspot Data." n.d. Royal Observatory of Belgium. http://www.sidc.be/silso/datafiles.