Visualization

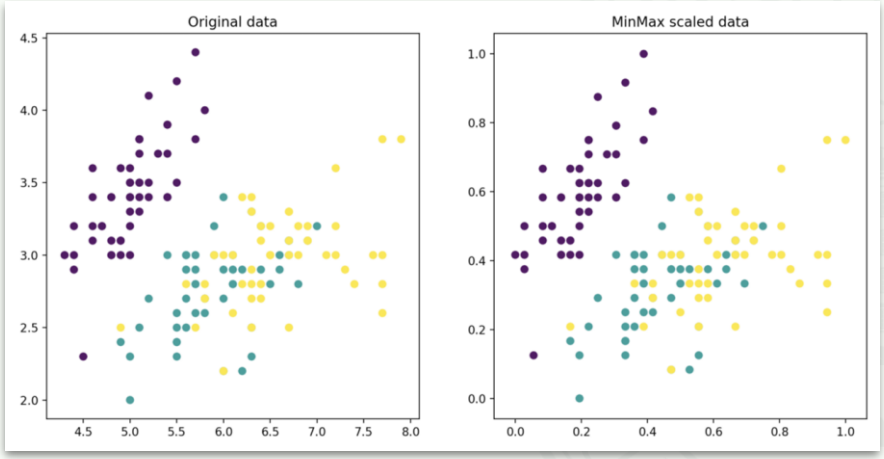
Prof. Murillo
Computational Mathematics, Science and Engineering
Michigan State University

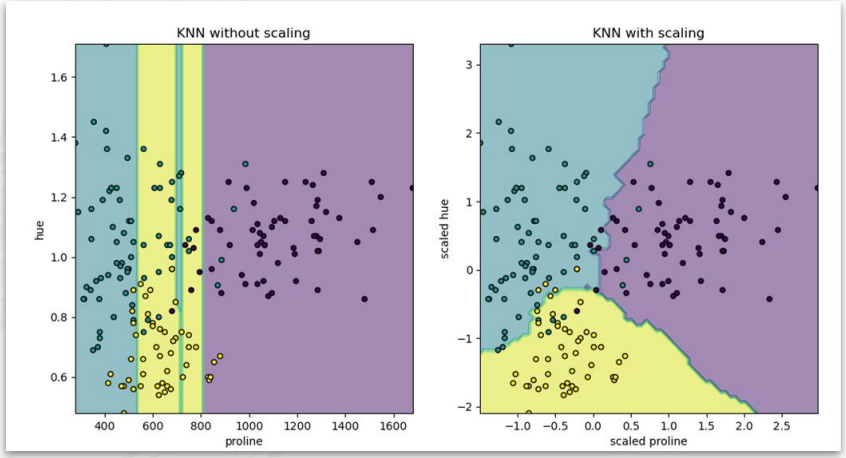
Today's Plan

- A few ideas I didn't get to from last week
- basics of visualization
  - telling the story
  - nuts and bolts of creating visuals
- next week: IDA and EDA
  - focus on Seaborn

There are Many Scaling Methods

In addition to standard (z score) scaling, specialized variants exist.





Scaling can have a dramatic effect on modeling.

CA Housing Dataset

7.2.7. California Housing dataset

Data Set Characteristics:

Number of Instances: 20640

Number of Attributes: 8 numeric, predictive attributes and the target

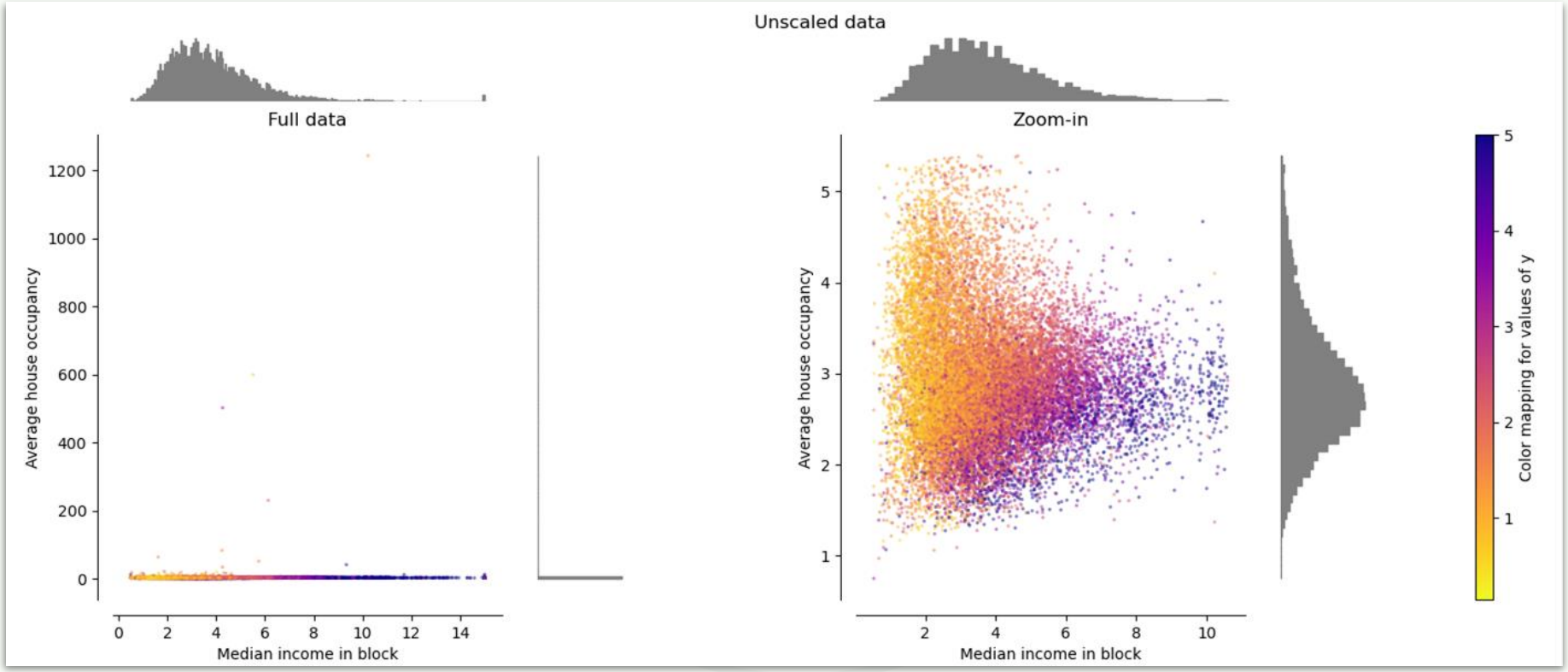
Attribute Information:

- MedInc median income in block group
- HouseAge median house age in block group
- AveRooms average number of rooms per household
- AveBedrms average number of bedrooms per household
- Population block group population
- AveOccup average number of household members
- Latitude block group latitude
- Longitude block group longitude

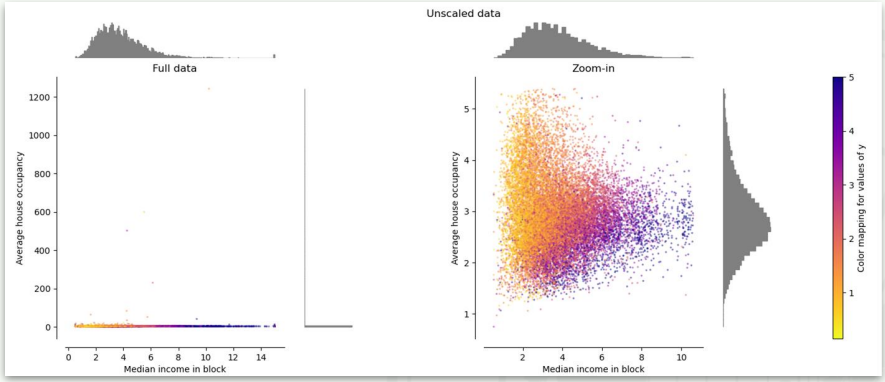
Missing Attribute Values: None

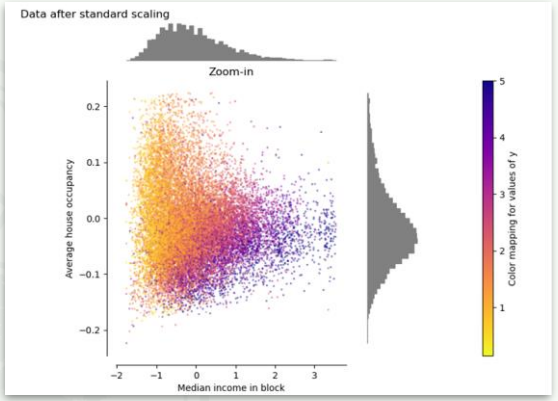
We also have scaled home values.

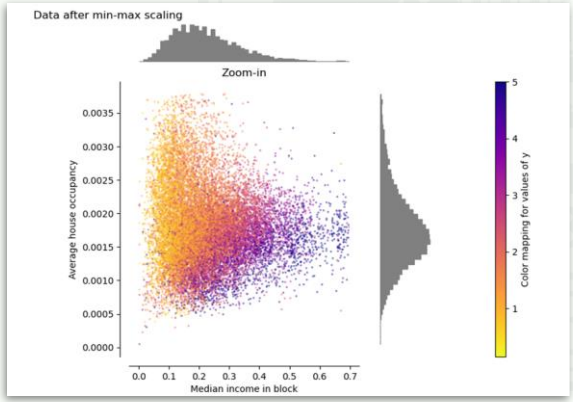
Impact on Outliers

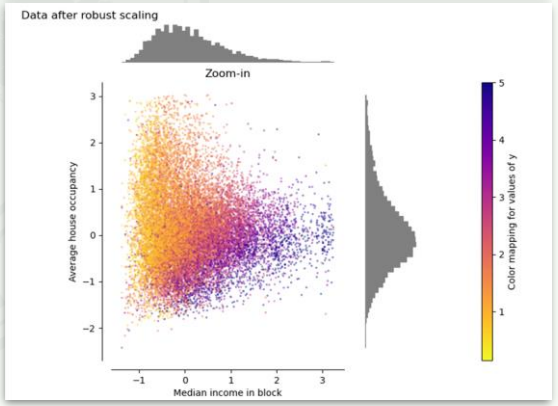


Impact on Outliers: Comparison

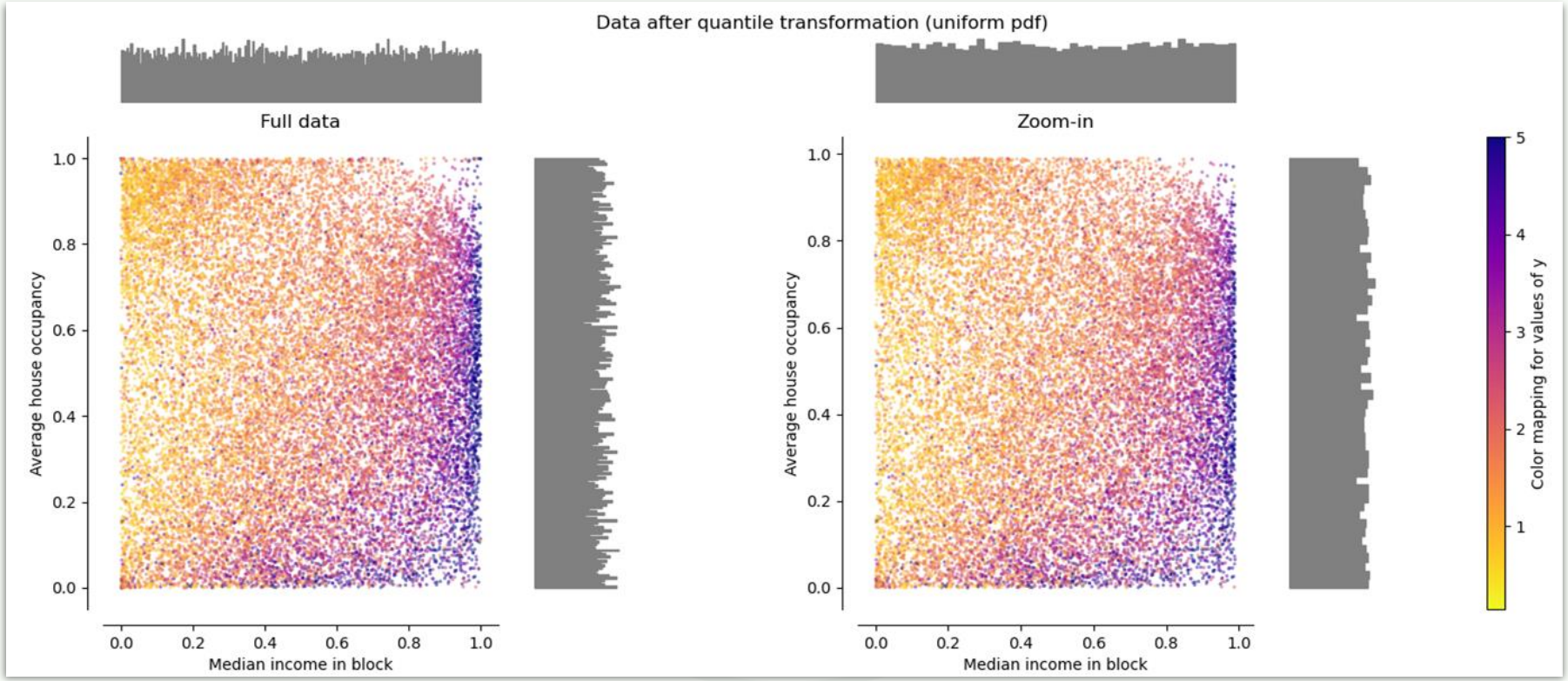




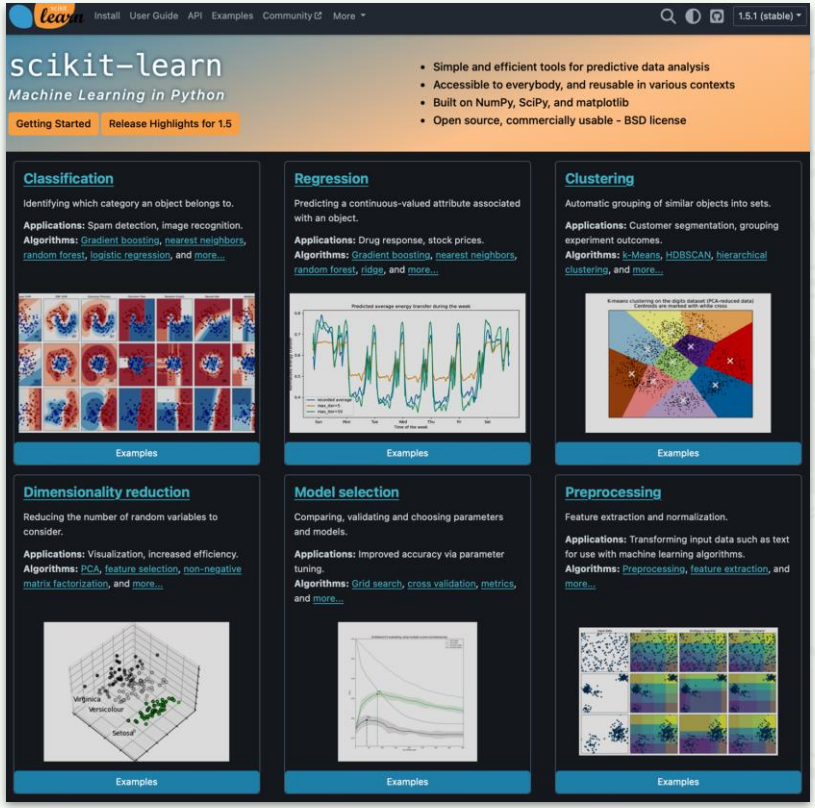


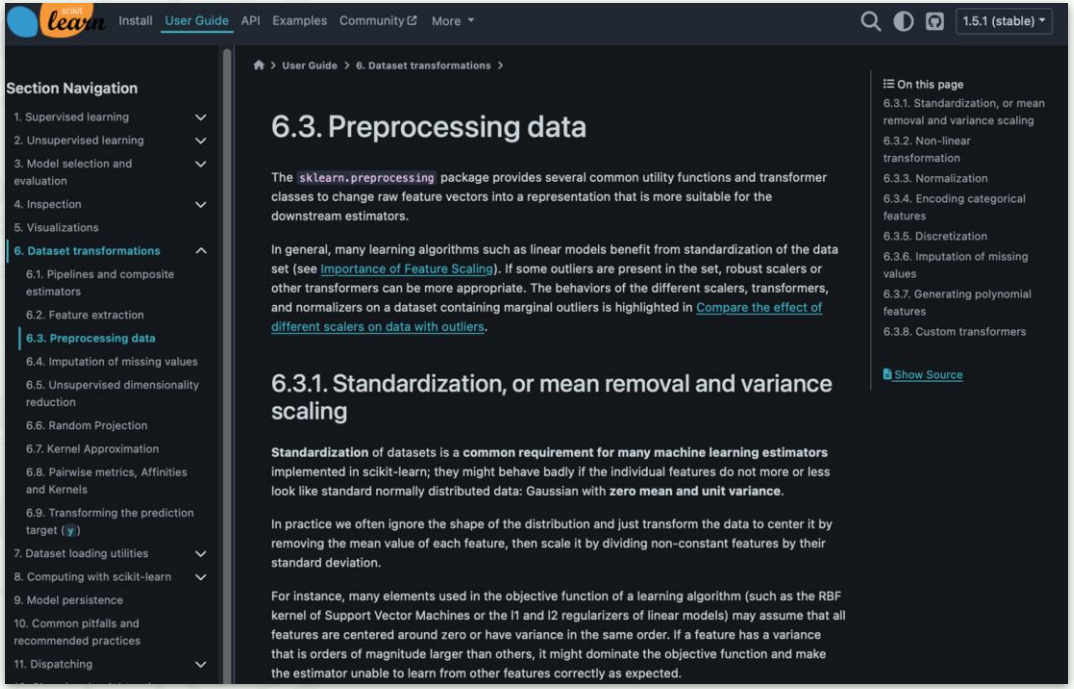


Nonlinear Scaling



Interlude on `scikit-learn` and scaling





API of `StandardScaler`

import libraries

```
[6]  ✓ 0.0s
1  from sklearn.preprocessing import StandardScaler
2  import numpy as np
```

fake data

```
[7]  ✓ 0.0s
1  X_train = np.array([[ 1., -1.,  2.],
2                      [ 2.,  0.,  0.],
3                      [ 0.,  1., -1.]])
```

check

```
[8]  ✓ 0.0s
1  X_train
```

```
array([[ 1., -1.,  2.],
       [ 2.,  0.,  0.],
       [ 0.,  1., -1.]])
```

create object

fit object to data

```
[18] ✓ 0.0s
1  scaler = StandardScaler()
2  scaler.fit(X_train)
```

μ

```
[12] ✓ 0.0s
1  scaler.mean_
```

```
array([0.81649658, 0.81649658, 1.24721913])
```

σ

```
[13] ✓ 0.0s
1  scaler.scale_
```

```
array([0.81649658, 0.81649658, 1.24721913])
```

apply to data and create new array

```
[16] ✓ 0.0s
1  X_scaled = scaler.transform(X_train)
```

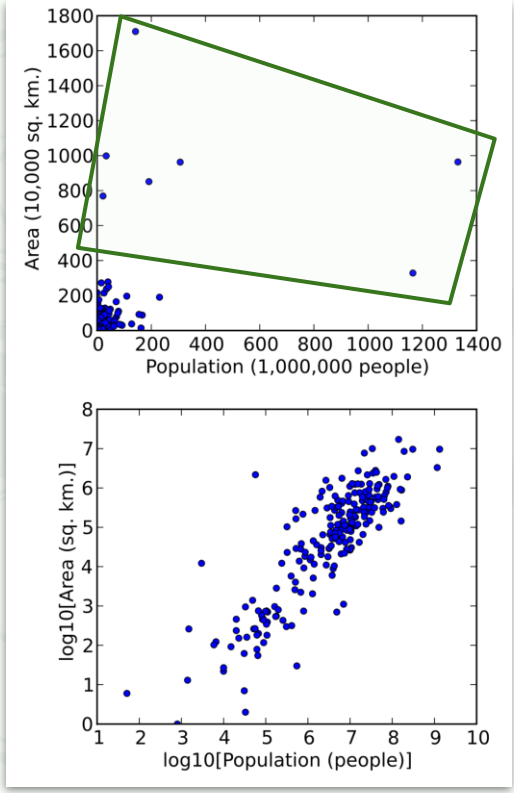
check

```
[17] ✓ 0.0s
1  X_scaled
```

```
array([[ 0.        , -1.22474487,  1.33630621],
       [ 1.22474487,  0.        , -0.26726124],
       [-1.22474487,  1.22474487, -1.06904497]])
```

Transforming

Transforming is a related operation, and might be used with scaling.

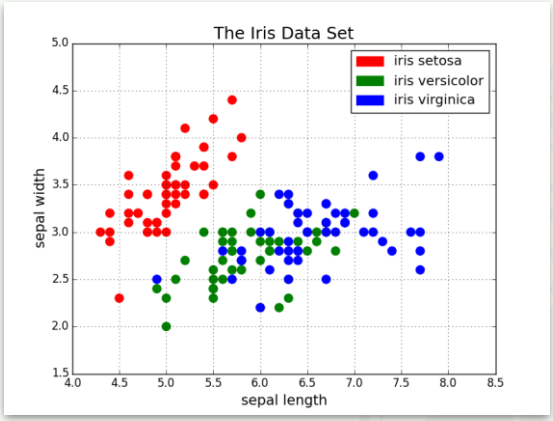


models will be dominated by these values

z-scaling doesn't help here

log-log transformed: much more balanced

Encoding



How do we handle categorical data?

Encoding maps the category onto a numerical value.

There are many ways to do this - very problem dependent.

- **label encoder**: encode target labels with *[0,N-1]* *(lossy)*
- **ordinal encoder**: features converted to ordinal numbers *[0,N-1]*
- **one-hot encoder**: creates binary feature for each category

Compare Label and One-Hot Encoding

Label Encoding

Food Name	Categorical #	Calories
Apple	1	95
Chicken	2	231
Broccoli	3	50

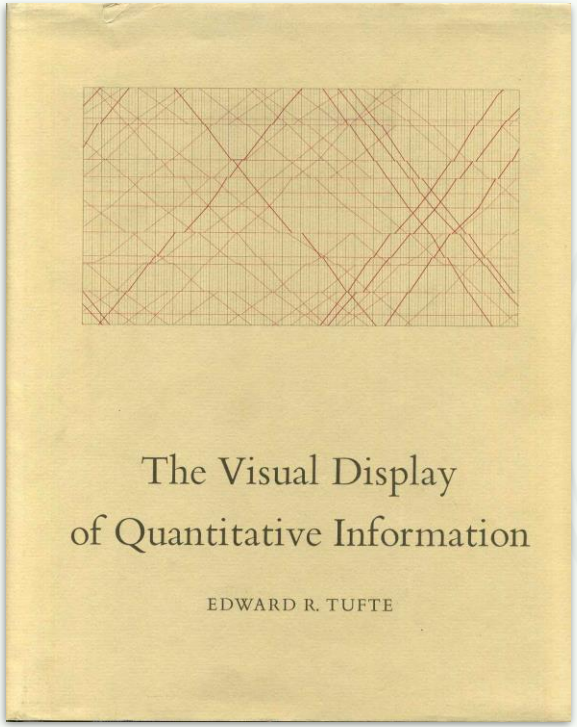
One Hot Encoding

Apple	Chicken	Broccoli	Calories
1	0	0	95
0	1	0	231
0	0	1	50

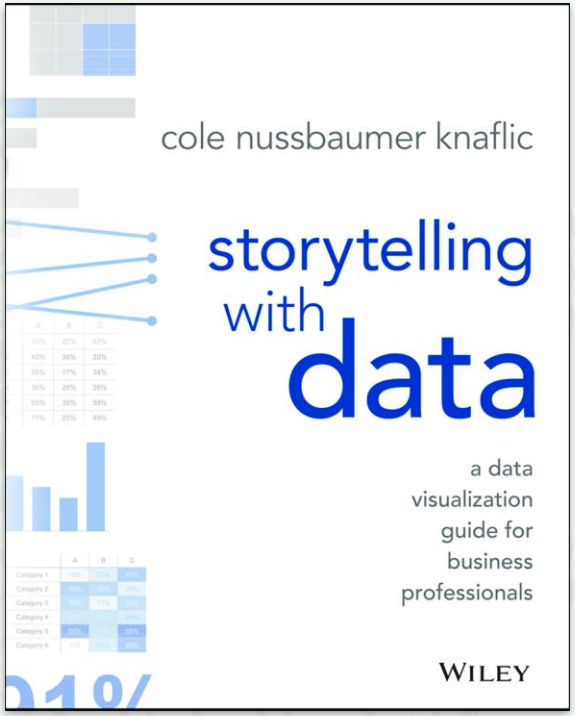
Today's Plan

- A few ideas I didn't get to from last week
- **basics of visualization**
  - telling the story
  - nuts and bolts of creating visuals
- next week: IDA and EDA
  - focus on Seaborn

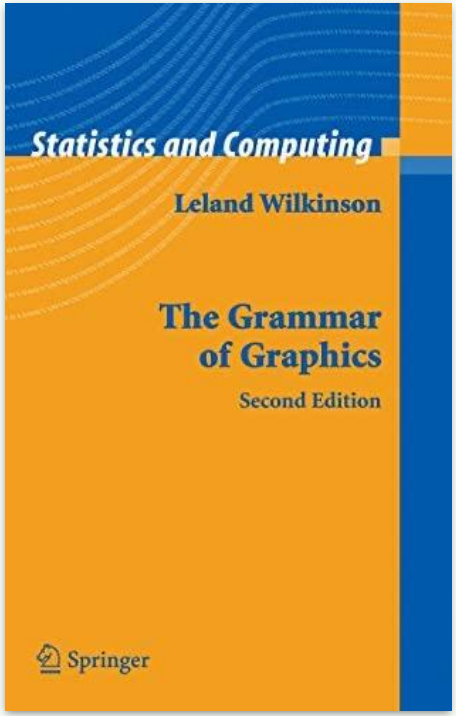
Three Classics You May Wish To Own



classic on good practice

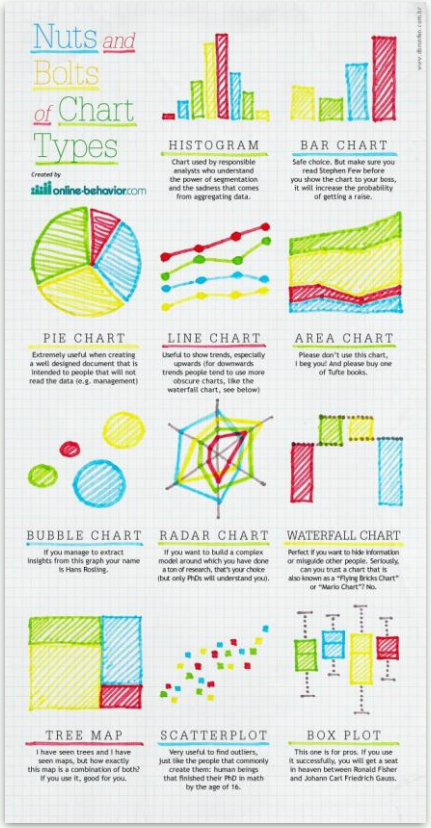


focuses on the message



technical approach to graphics

Plotting By Type

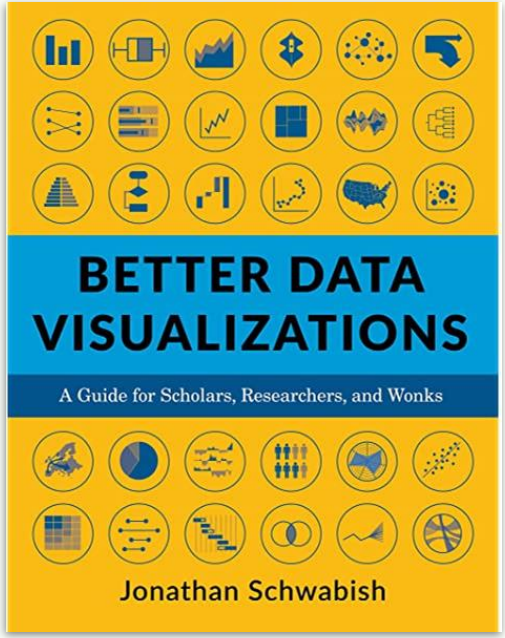


There are many standard types of plots in wide use.

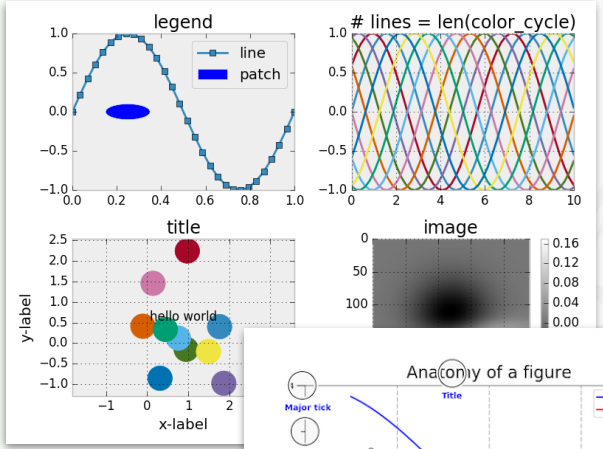
Most of these are built into matplotlib.

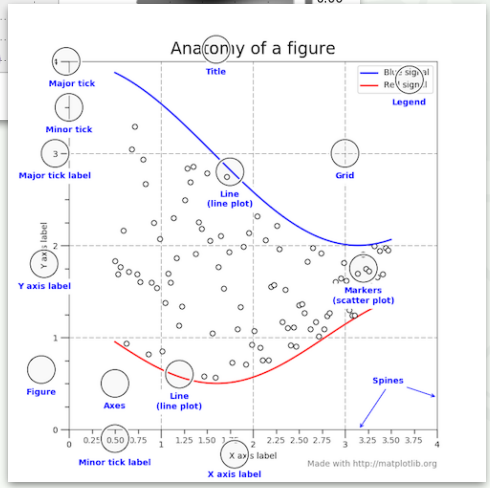
You can quickly make these plots with very few lines of code.

For most plotting tasks, these standard plots are all you need!

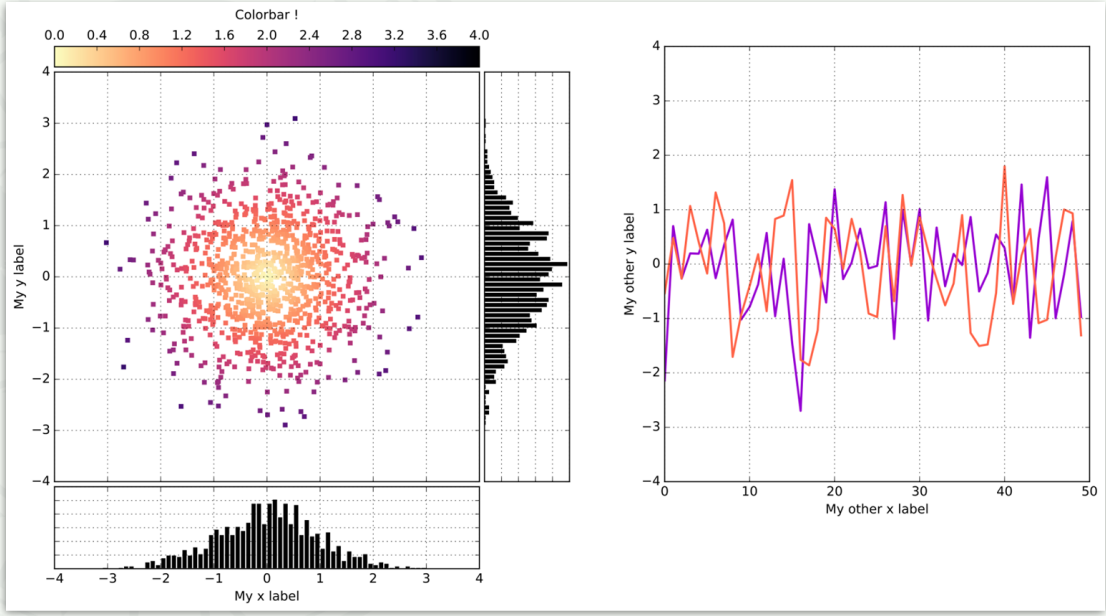


Custom Plotting





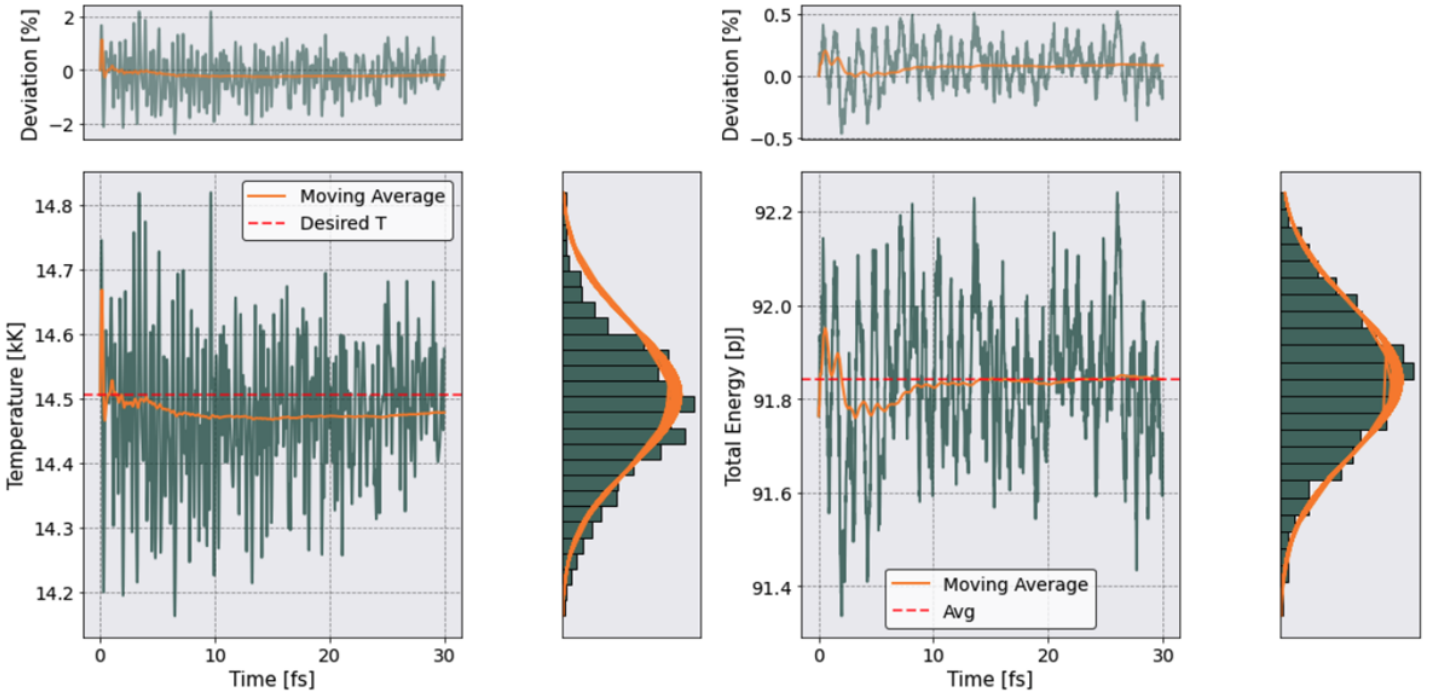
You want complete control over what your plot looks like.



Data From A Simulation

Job ID: yocp
Phase: Production
No. of species = 1
Species 1 : H
  No. of particles = 10000
  Temperature = 14.51 [kK]

Total N = 10000
Thermostat: berendsen
  Berendsen rate = 1.00
  Equilibration cycles = 167
Potential: yukawa
  Coupling Const = 1.01e+02
  Tot Force Error = 6.44e-06
Integrator: verlet
  Δt = 2.00 [as]
Completed steps = 15000
Total steps = 15000
100.00 % Completed
Production time = 30.00 [fs]
Production cycles = 502



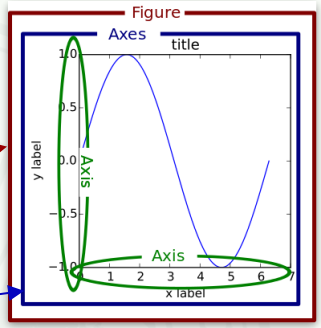
Plot Structure

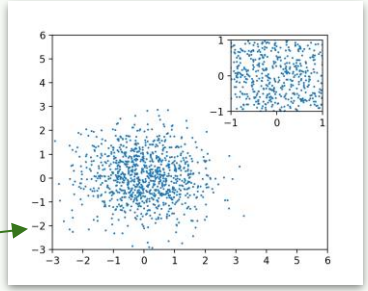
Plots are typically organized around three concepts:

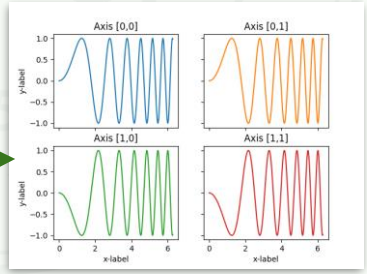
Figure: this is the frame. You can have many of these, and they form separate entities.

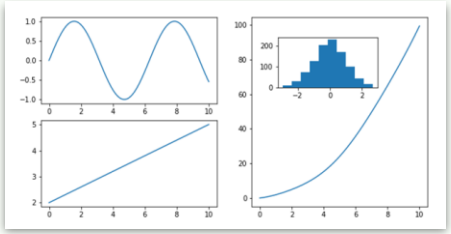
Axis: this is the plot itself, with its axes. It has no position - it is free floating in the frame somewhere. You can have many of these within a single figure (frame).

Subplot: these are axes in some sort of predetermined arrangement.









example of using both

Trends in Visualization For Data Science: GoG

```python
import altair as alt
import pandas as pd

data = pd.DataFrame({'x': ['A', 'B', 'C', 'D', 'E'],
                     'y': [5, 3, 6, 7, 2]})
alt.Chart(data).mark_bar().encode(
    x='x',
    y='y',
)
```

bqplot

latest

Search docs

Introduction

Usage

API Reference Documentation

Docs » bqplot: Plotting for Jupyter

bqplot: Plotting for Jupyter

- Introduction
  - Goals
  - Installation

plotnine 0.10.1 API Gallery Tutorials Site Page Search

A Grammar of Graphics for Python

plotnine is an implementation of a *grammar of graphics* in Python, it is based on ggplot2. The grammar allows users to compose plots by explicitly mapping data to the visual objects that make up the plot.

Plotting with a grammar is powerful, it makes custom (and otherwise complex) plots easy to think about and then create, while the simple plots remain simple.

Example

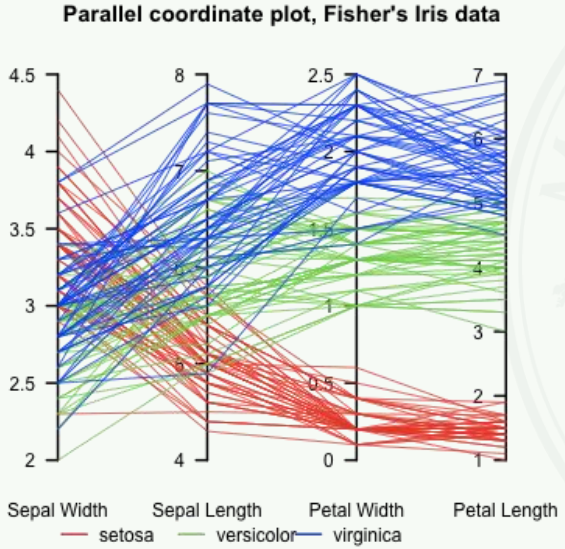
```python
from plotnine import ggplot, geom_point, aes, stat_smooth, facet_wrap
from plotnine.data import mtcars

(ggplot(mtcars, aes('wt', 'mpg', color='factor(gear)'))
 + geom_point()
 + stat_smooth(method='lm')
 + facet_wrap('~gear'))
```

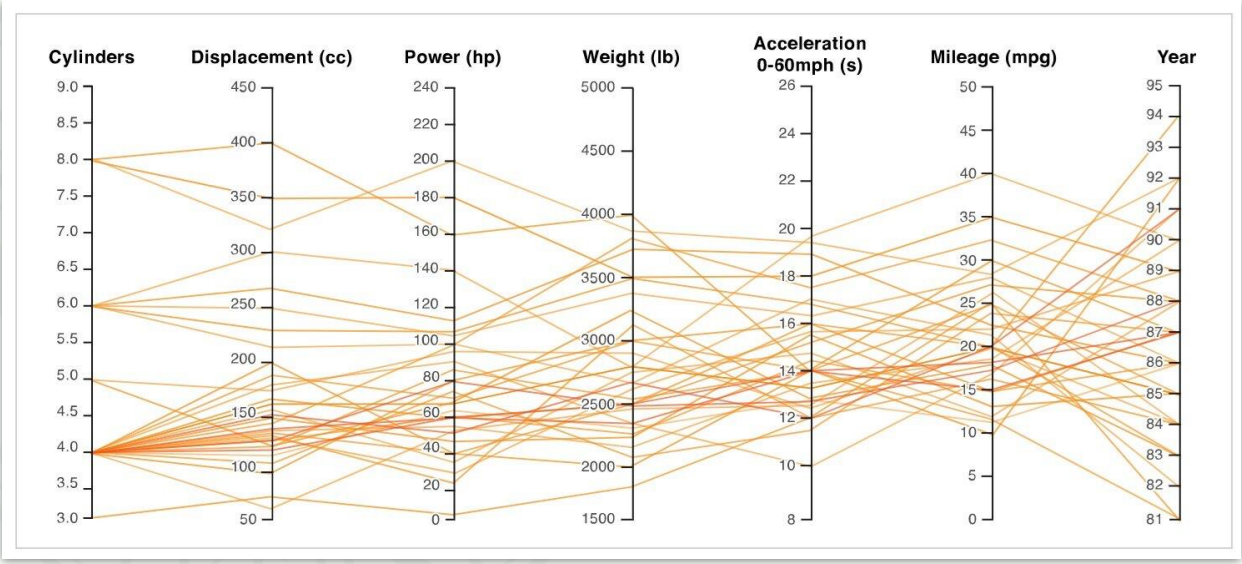
The seaborn.objects interface

The `seaborn.objects` namespace was introduced in version 0.12 as a completely new interface for making seaborn plots. It offers a more consistent and flexible API, comprising a collection of composable classes for transforming and plotting data. In contrast to the existing `seaborn` functions, the new interface aims to support end-to-end plot specification and customization without dropping down to matplotlib (although it will remain possible to do so if necessary).

Trends in Visualization For Data Science: Types

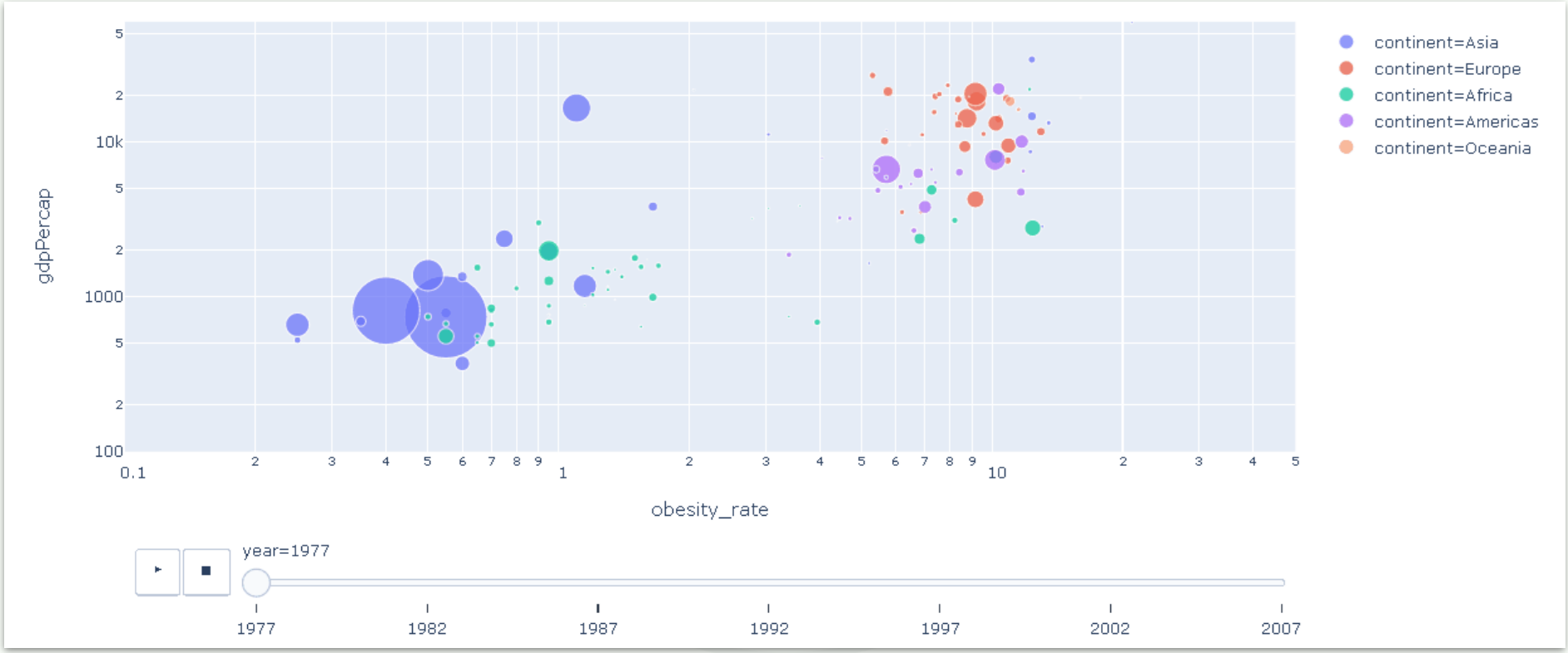
Important one: **parallel plot**.





Useful for high-dimensional data.

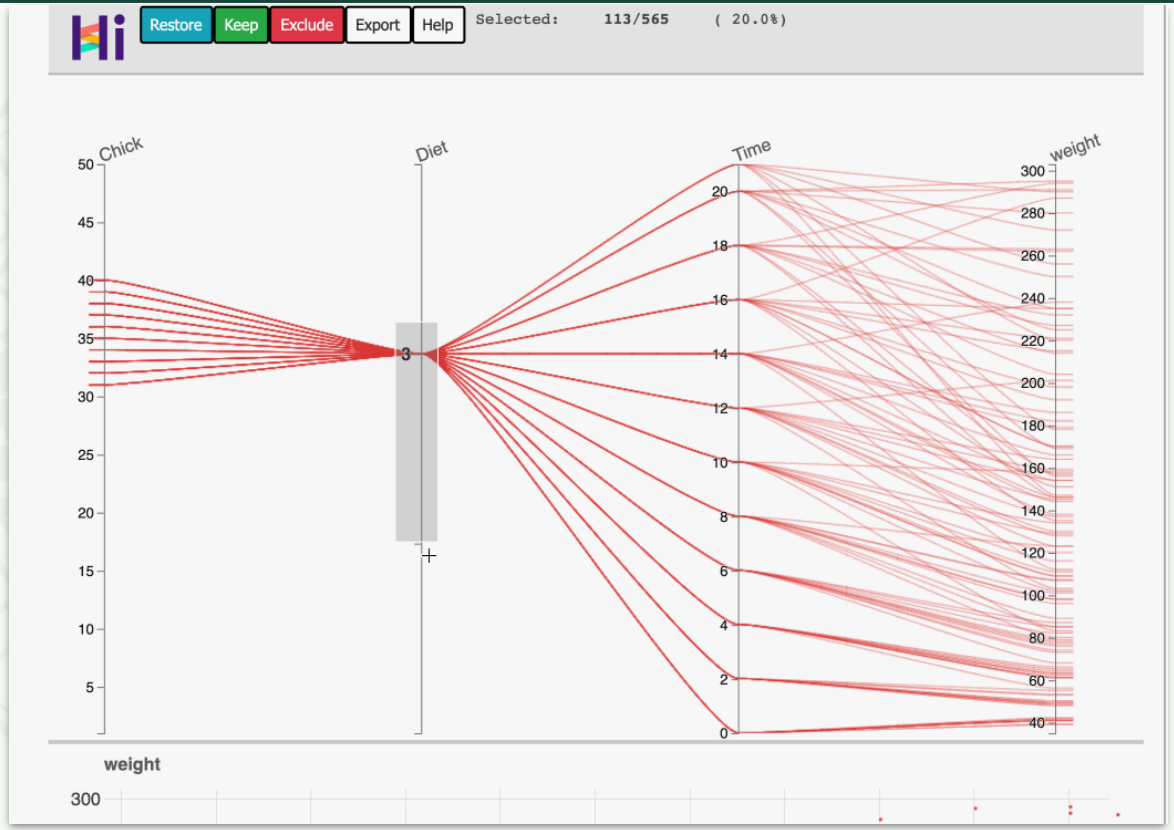
Trends in Visualization For Data Science: Interactivity



Trends in Visualization For Data Science: Web Interactivity

Hi-Plot: Interactive Parallel Plot

HiPlot



Communicating Your Message

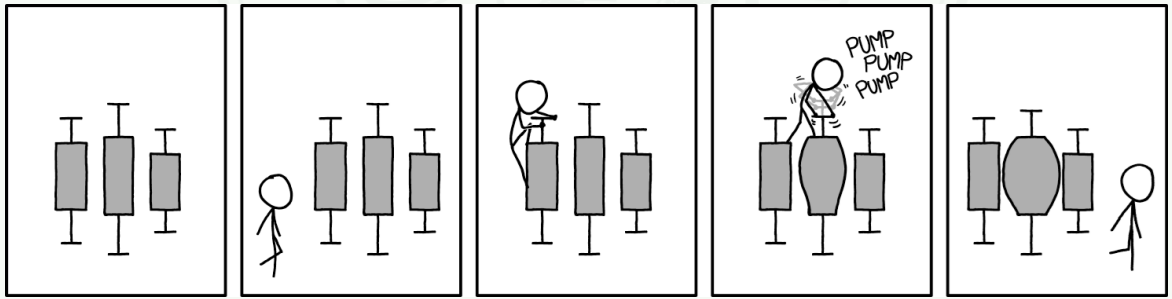
You now know the mechanics of making visualizations, including very rich data science charts.

But, are you telling the right story?

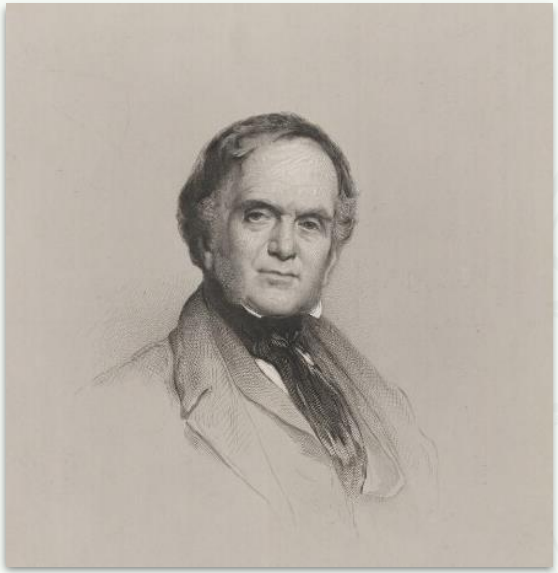
Are you communicating and connecting with your audience?

In this next segment, we will examine how to efficiently tell the most accurate and compelling story.

Parts of what is to come are somewhat subjective - use the ideas as guides for good practice, but break the rules when it works for you.

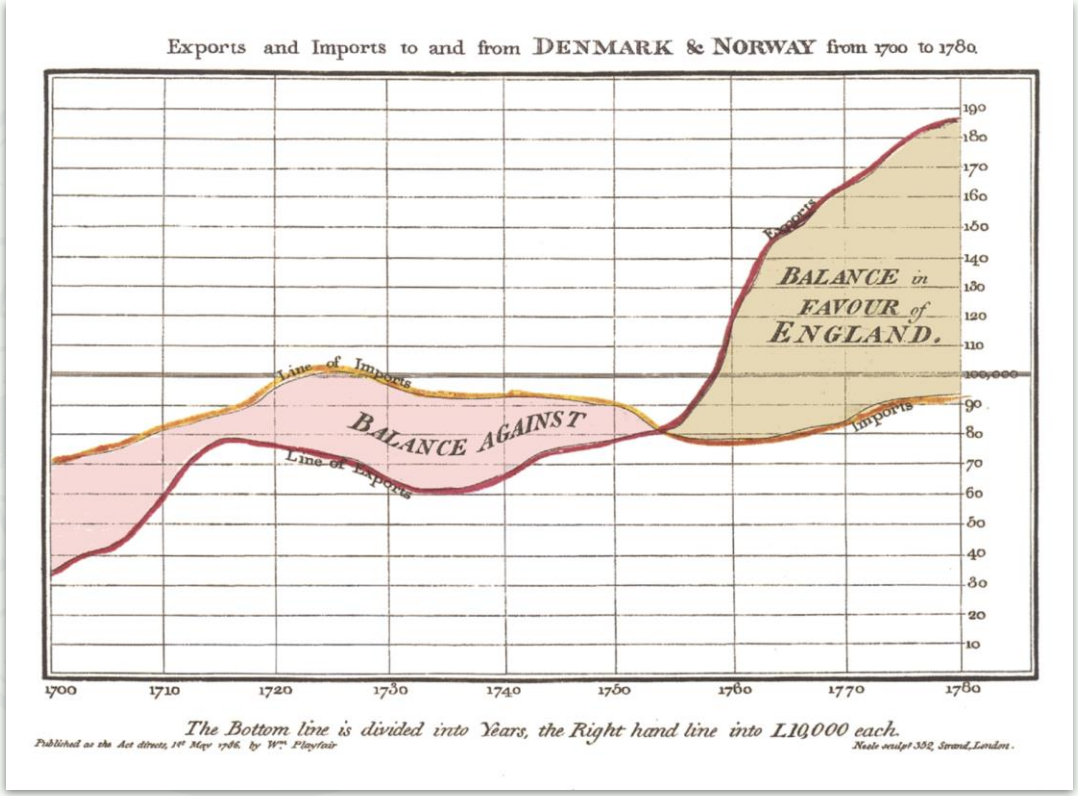


Visualization is An Old Science



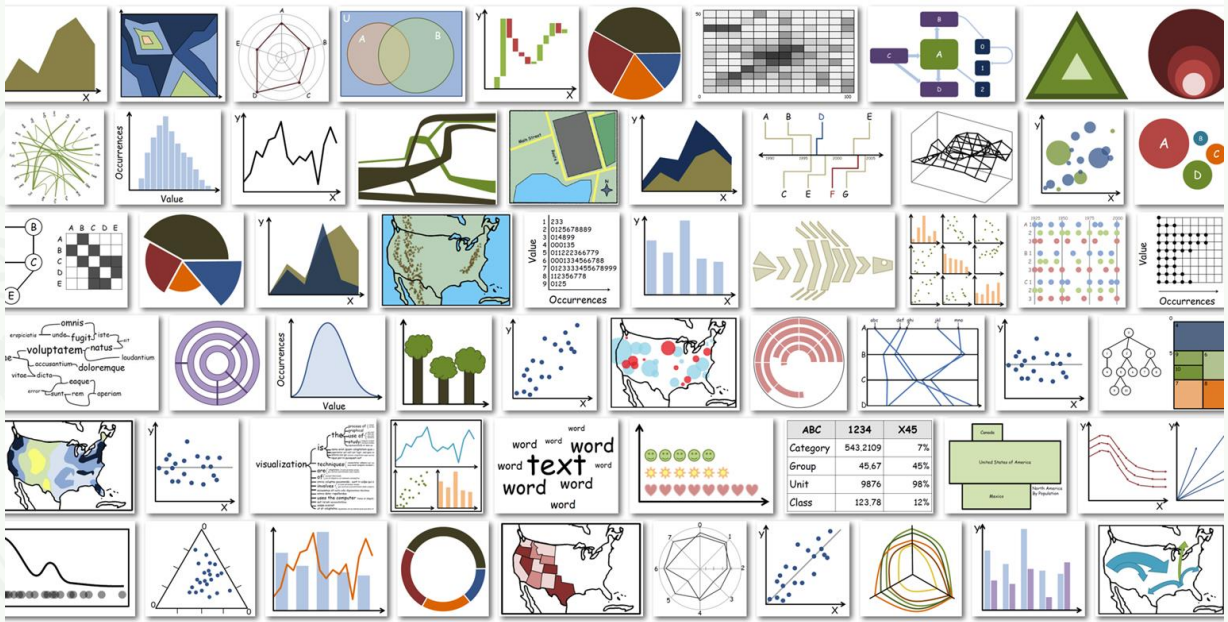
William Playfair
1759 - 1823

Considered the “father of visualization”.



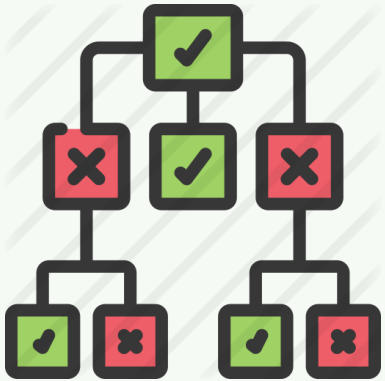
Visualization is Communication

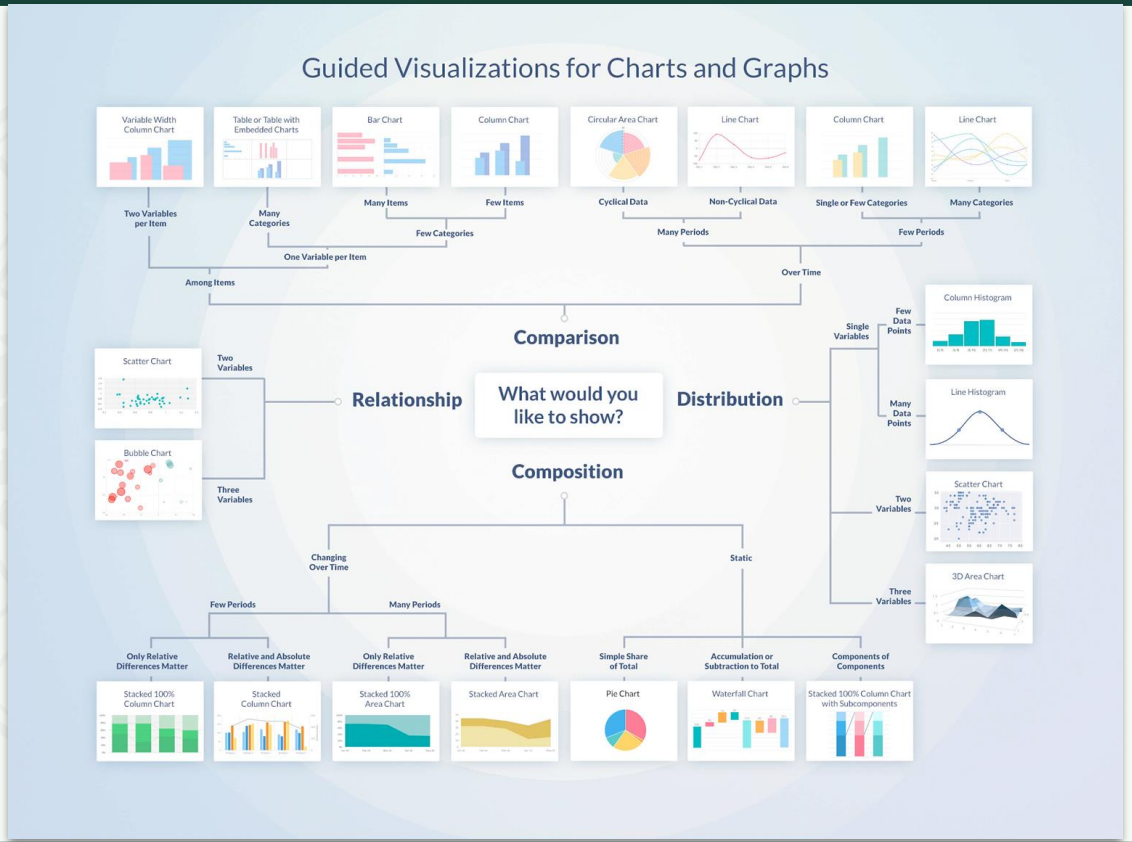
Choose the best visualization format to communicate your message.



Visualizations come in various types.

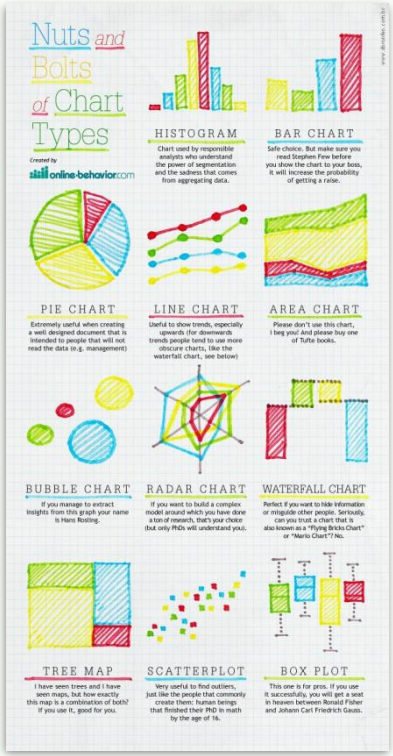
Think about your goal in terms of a decision tree of choices from the most basic to the detailed.





Use Standard Types, Or Invent Your Own

plotting by standard type



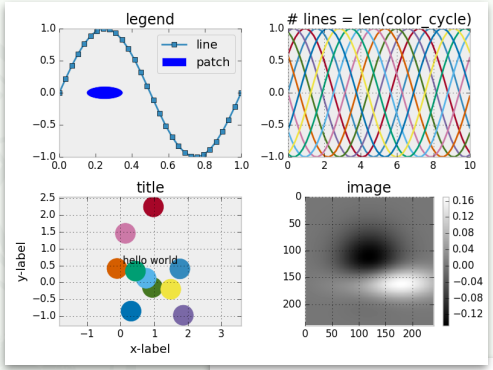
There are many standard types of plots in wide use.

Most of these are built into matplotlib.

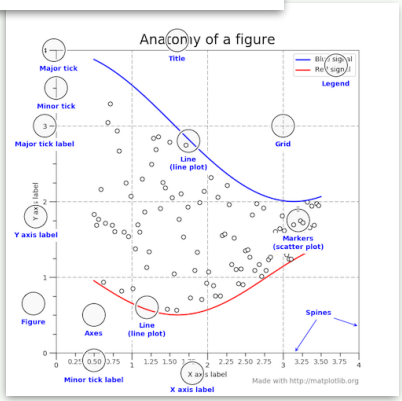
You can quickly make these plots with very few lines of code.

For most plotting tasks, these standard plots are all you need!

custom plotting

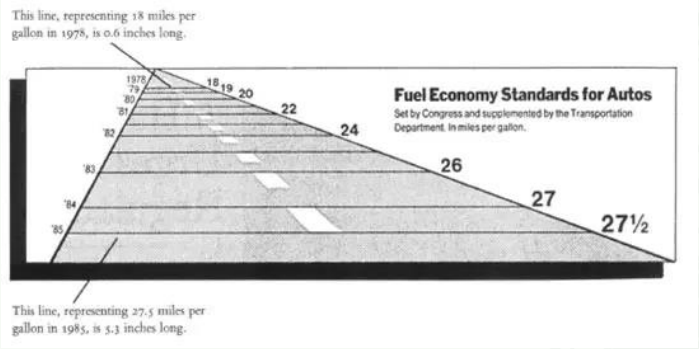


In some cases you want to take complete control over what your plot looks like.



Three* Rules From Edward Tufte

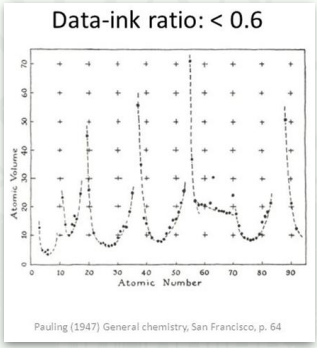
graphical integrity

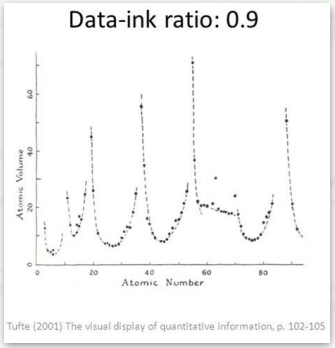


This chart has what Tufte calls a large "Lie Factor": the scales in the plot mislead the viewer because they are not scaled to the data.

1. Make your plots honest.
2. Remove ink that adds no value.
3. Don't use gimmicks that make your chart difficult to understand.

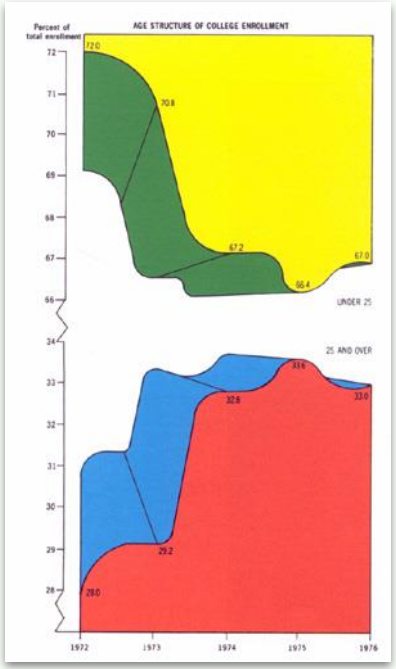
data ink





Is your extra ink making the plot less easy to read?

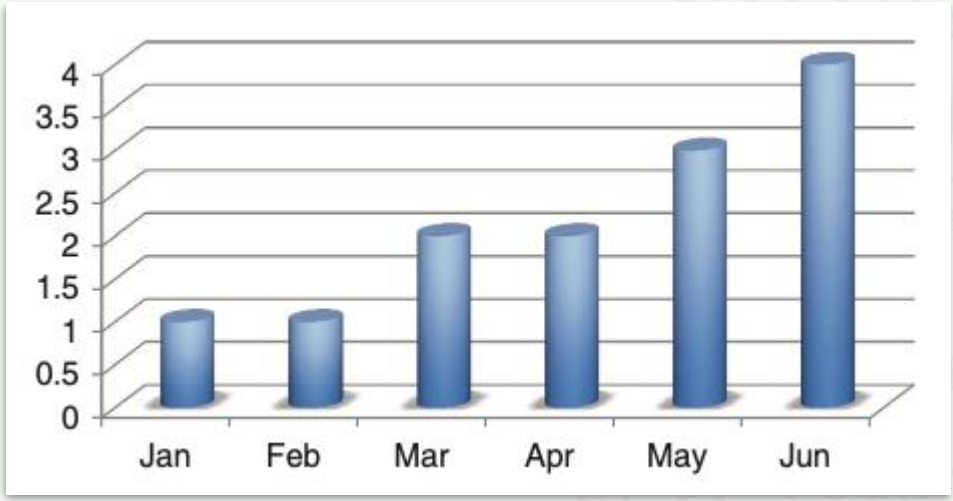
chartjunk



Can you decipher what this is supposed to be showing?

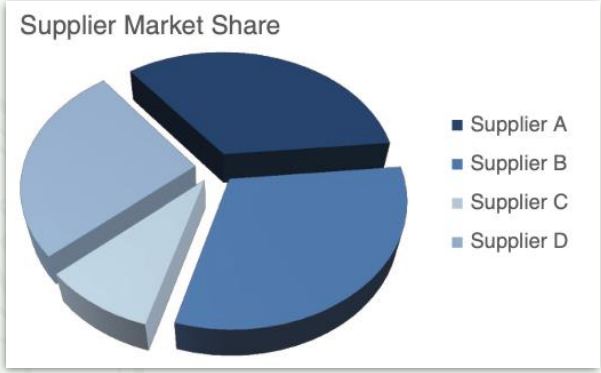
* There are many more rules in his books.

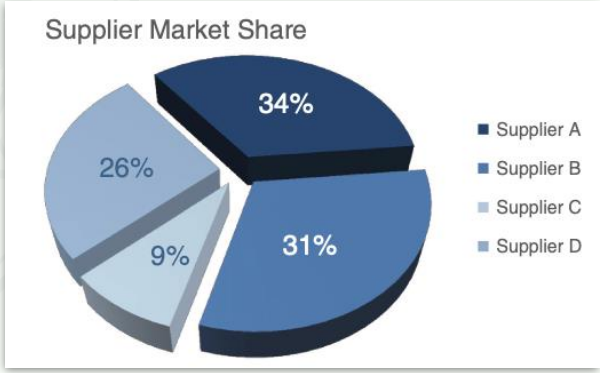
Fancy Plots Can Mislead



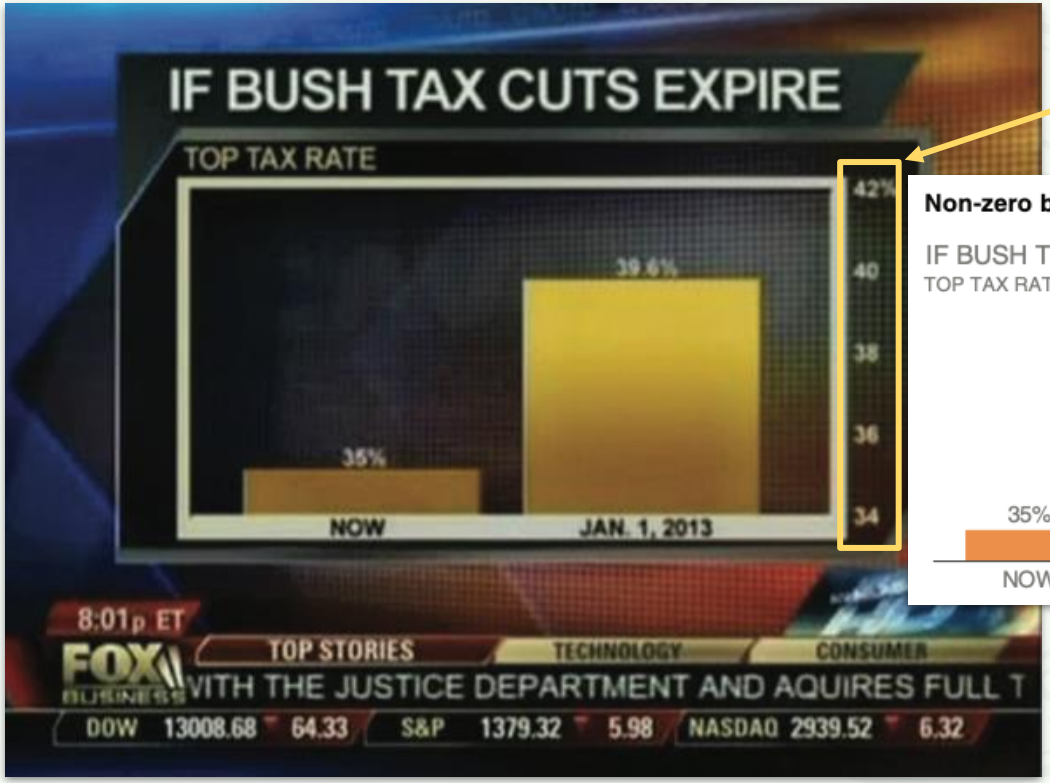
How precisely can you write down the actual values?

For example: is the value for Jan less than or more than 1?

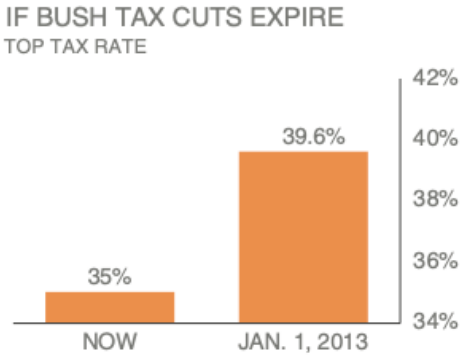




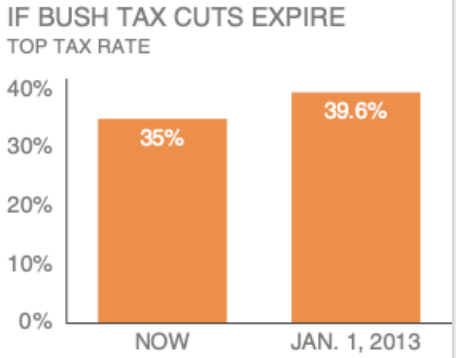
In The News: Example of Misleading Plot



Non-zero baseline: as originally graphed



Zero baseline: as it should be graphed



Overuse Of Color Can Mislead

This creates a “puzzle” for the viewer, which is to keep in their mind the ordering of colors in the rainbow as they look over the chart.

Worse, the viewer has to “turn off” social biases, such as “red = bad”, or bright colors are more important (e.g., yellow).

Country Level Sales Rank Top 5 Drugs

Rainbow distribution in color indicates sales rank in given country from #1 (red) to #10 or higher (dark purple)

Country	A	B	C	D	E
AUS	1	2	3	6	7
BRA	1	3	4	5	6
CAN	2	3	6	12	8
CHI	1	2	8	4	7
FRA	3	2	4	8	10
GER	3	1	6	5	4
IND	4	1	8	10	5
ITA	2	4	10	9	8
MEX	1	5	4	6	3
RUS	4	3	7	9	12
SPA	2	3	4	5	11
TUR	7	2	3	4	8
UK	1	2	3	6	7
US	1	2	4	3	5

Top 5 drugs: country-level sales rank

RANK	1	2	3	4	5+
COUNTRY I DRUG	A	B	C	D	E
Australia	1	2	3	6	7
Brazil	1	3	4	5	6
Canada	2	3	6	12	8
China	1	2	8	4	7
France	3	2	4	8	10
Germany	3	1	6	5	4
India	4	1	8	10	5
Italy	2	4	10	9	8
Mexico	1	5	4	6	3
Russia	4	3	7	9	12
Spain	2	3	4	5	11
Turkey	7	2	3	4	8
United Kingdom	1	2	3	6	7
United States	1	2	4	3	5

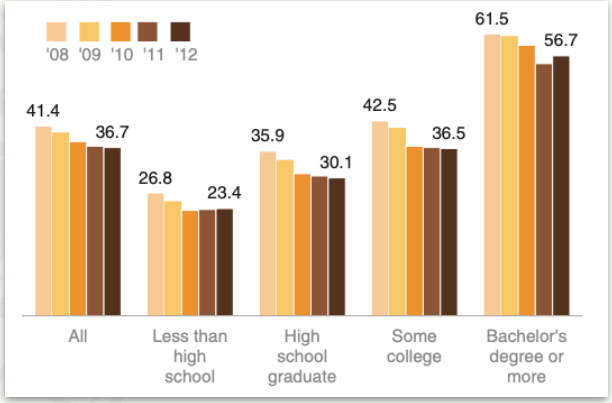
remake

Walkthrough of Plot Design

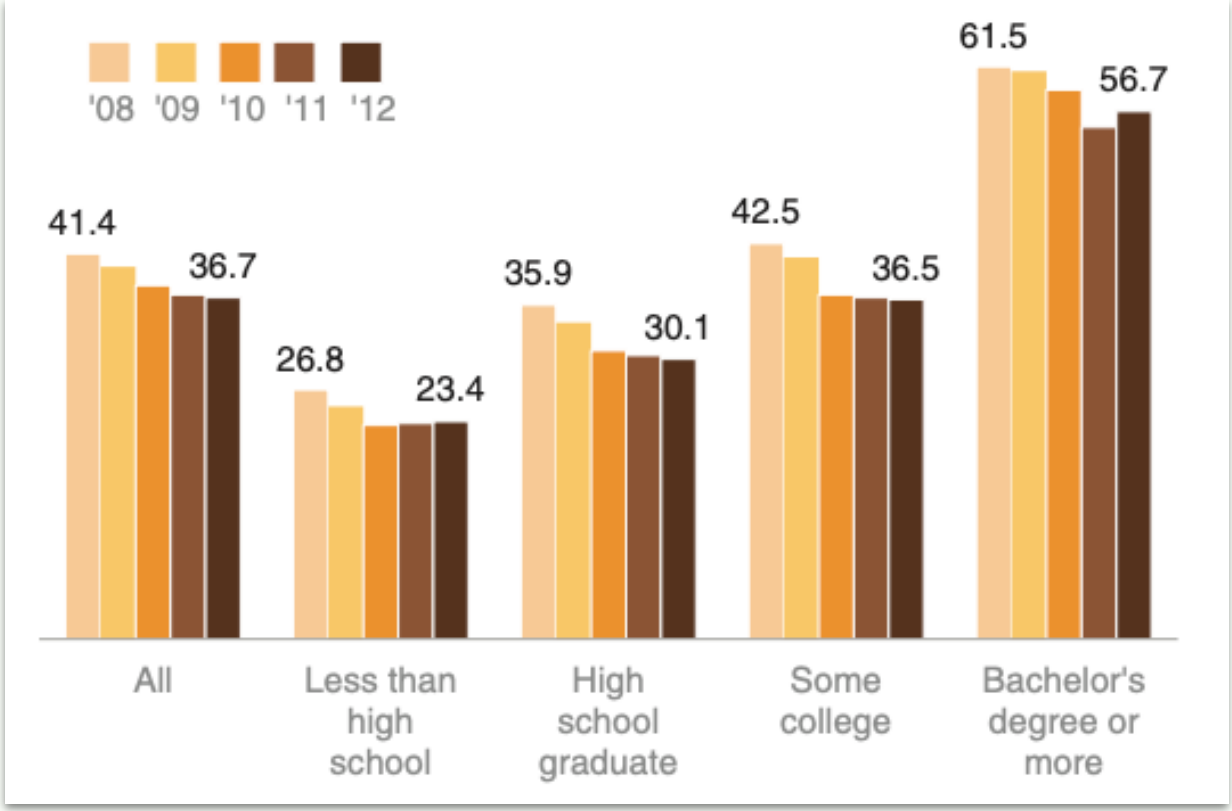
Let's take some reasonable plots and see if we can apply some of these ideas to improve them.

The most important goal is to be able to communicate what our point is, not just throw data at the viewer to try to figure out.

Tell your story.

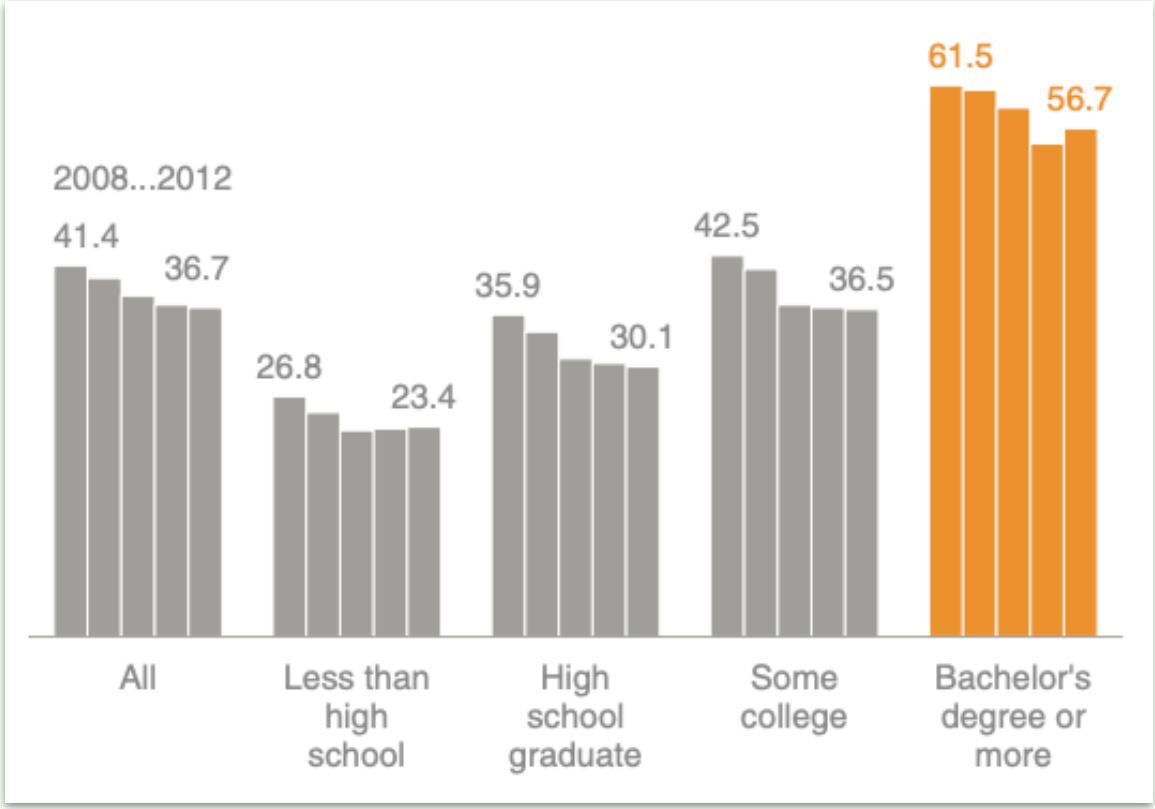


First Version - nice!

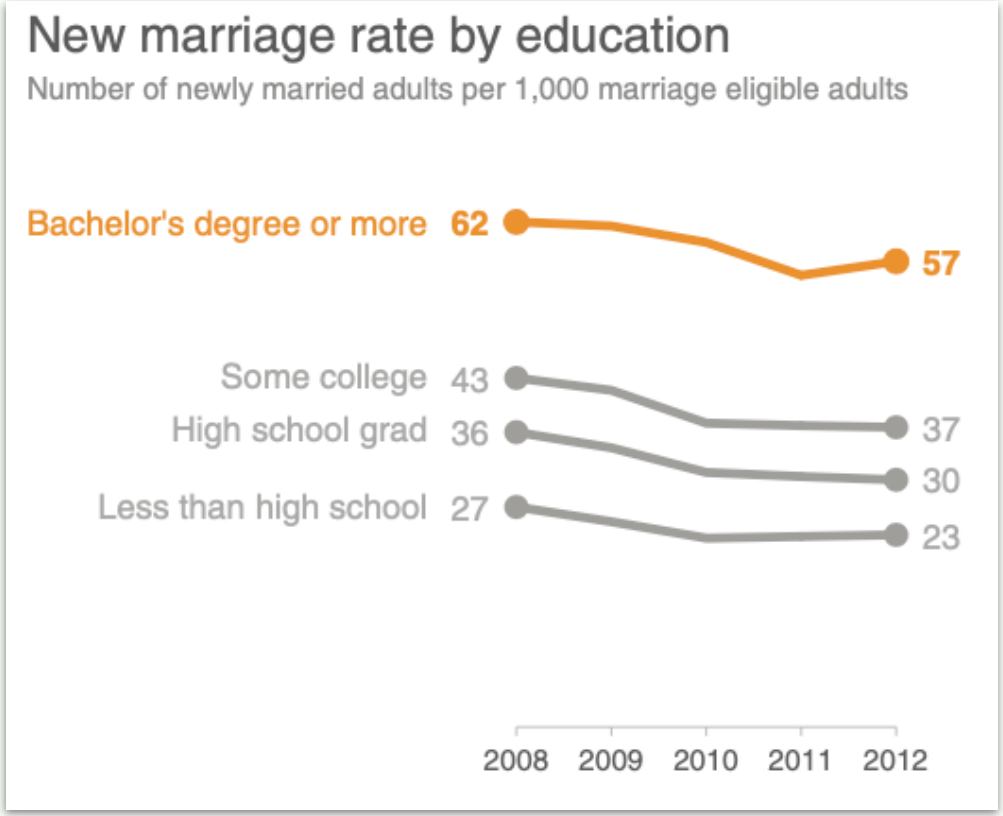


Redirect Attention To Your Story

What you might do at this step greatly depends on the story you wish to tell.



Think Outside The Box

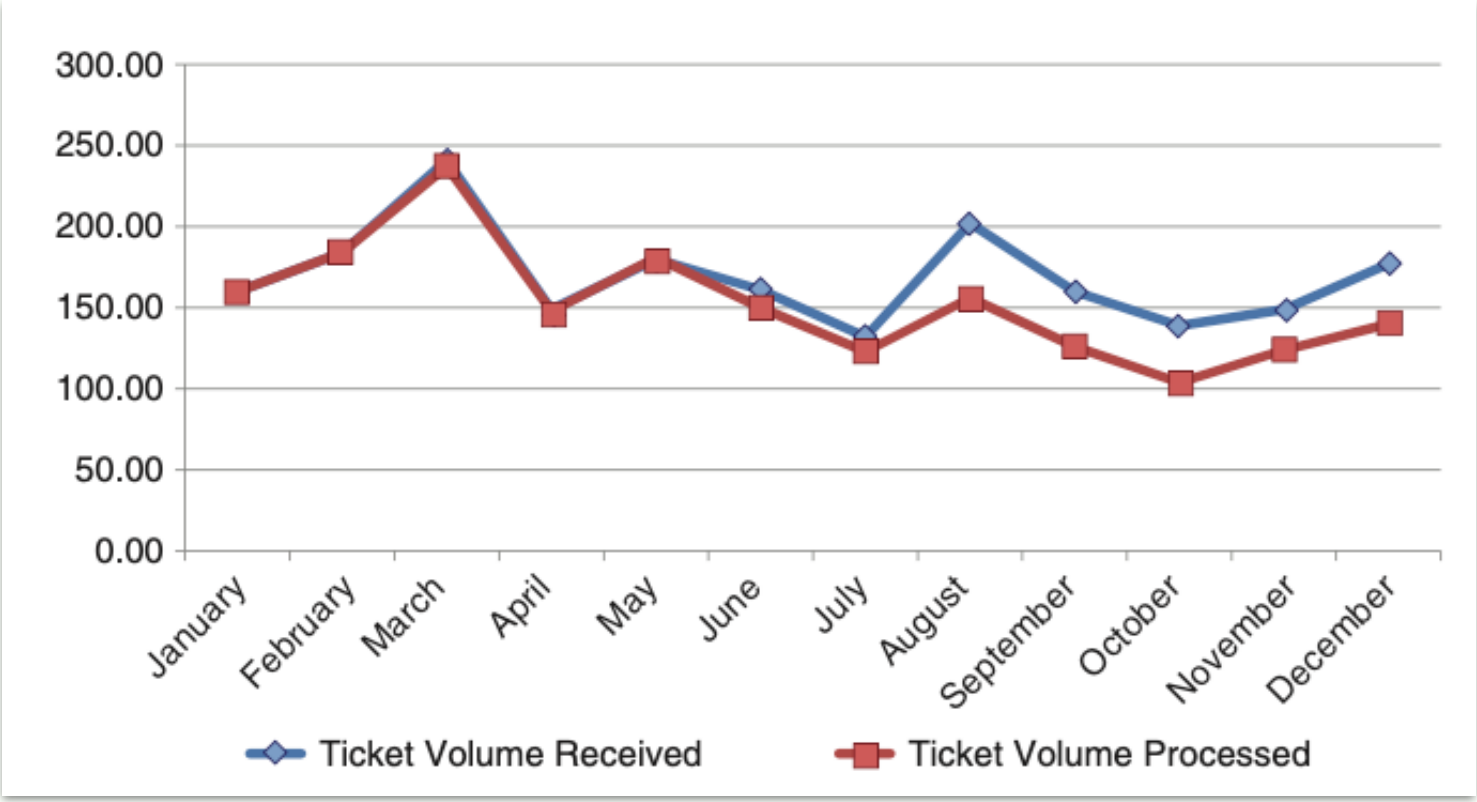


Is this an easier way to make your point?

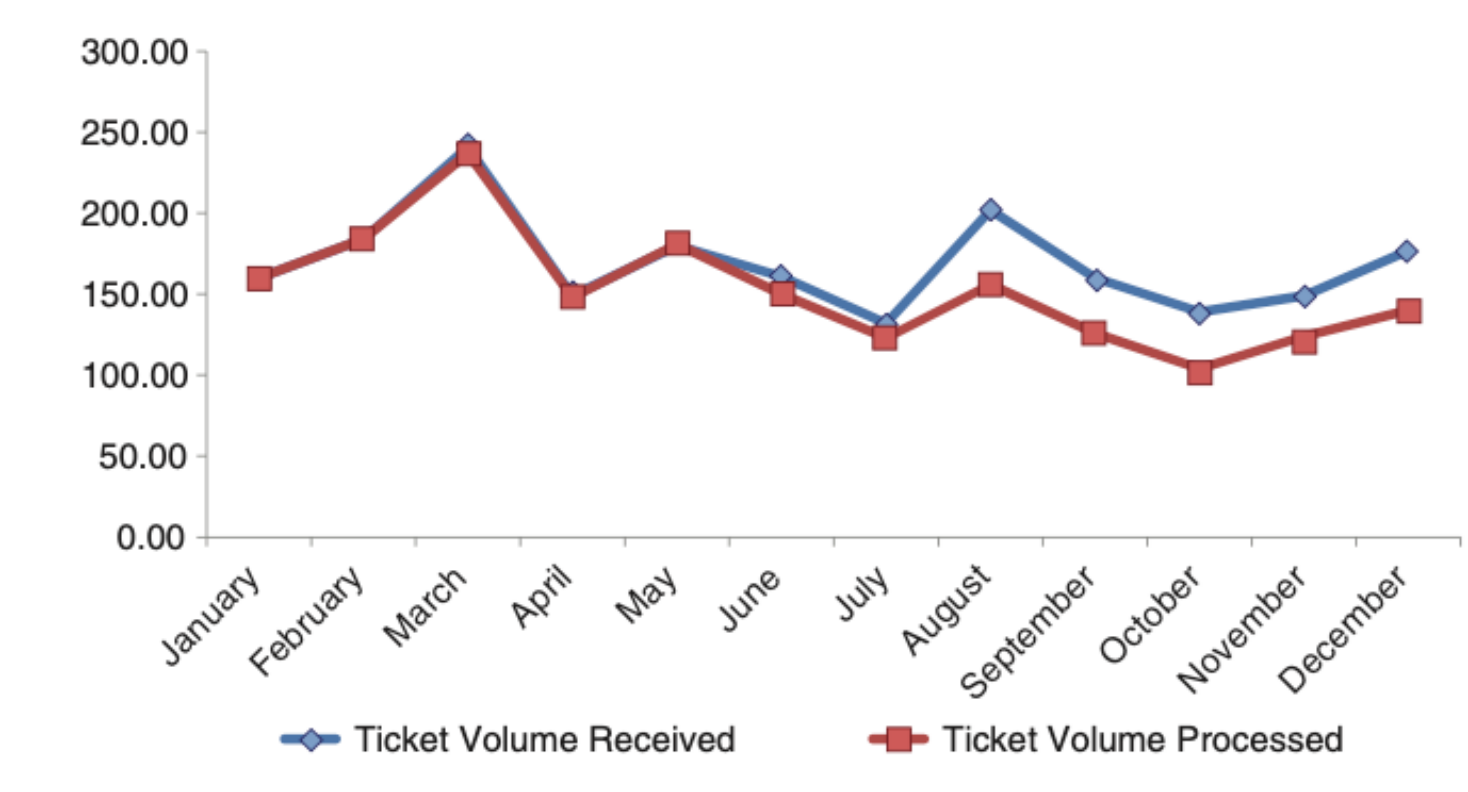
Walkthrough of Plot Design

Let's do another one in **nine** steps.

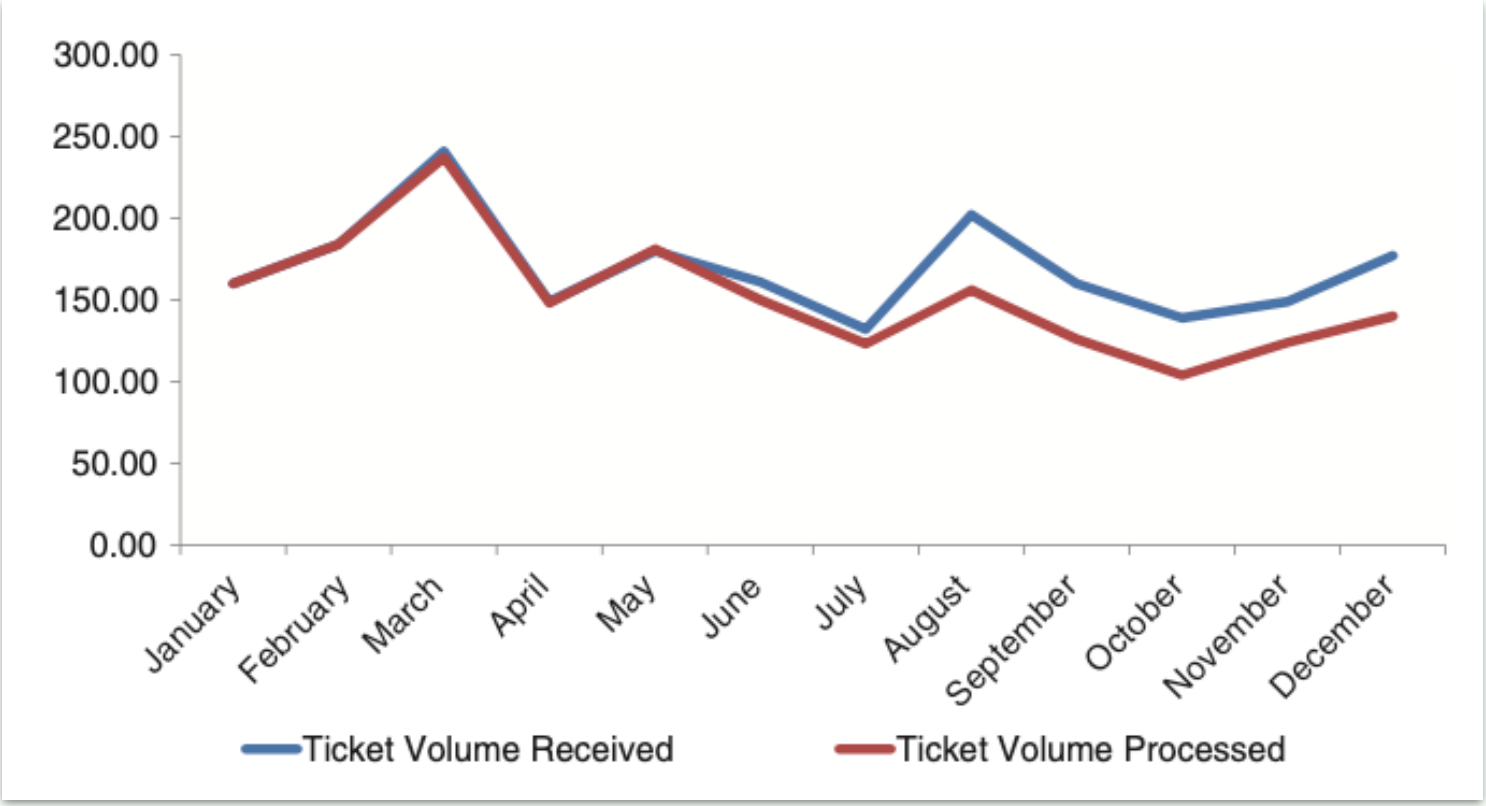
Starting Point (Not Bad!)



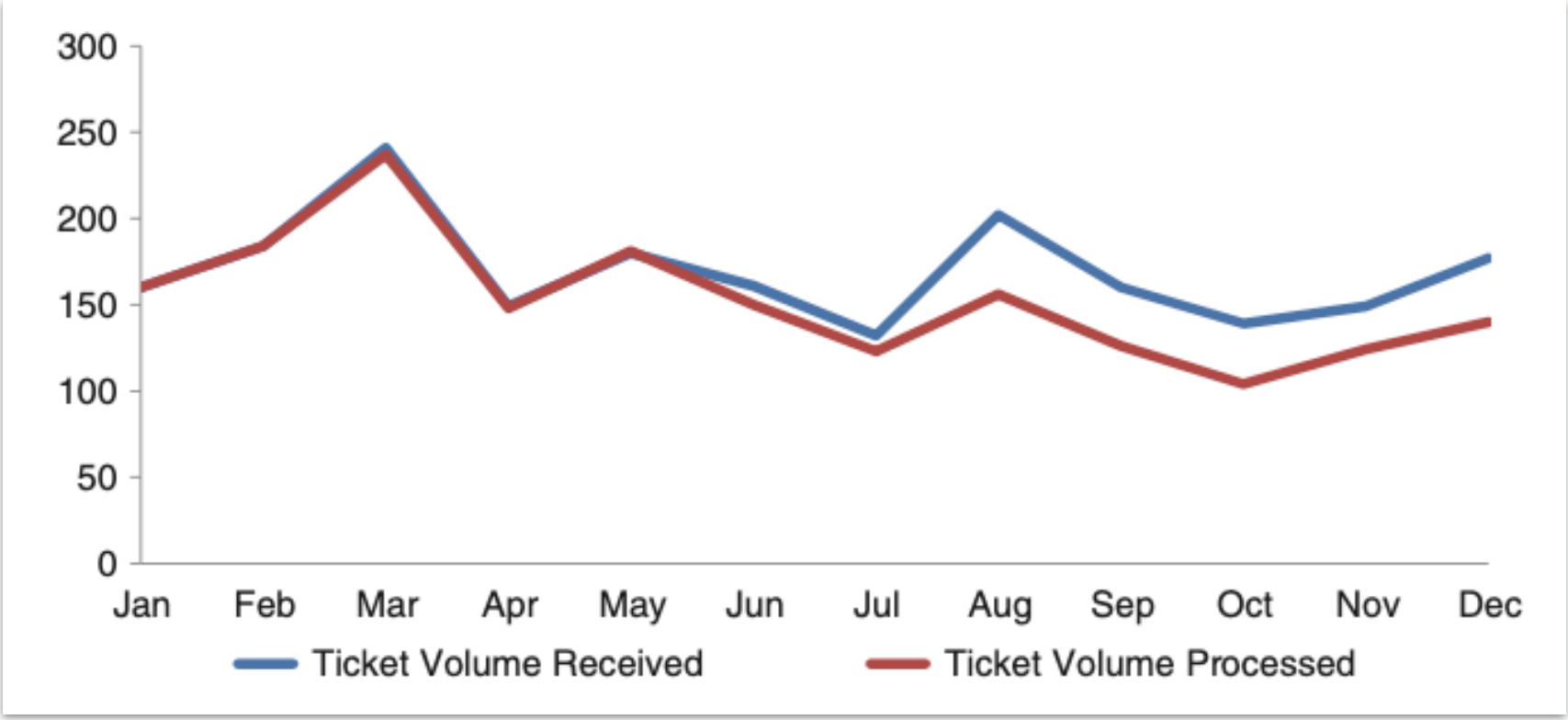
Step 2: Do We Need Grid Lines?



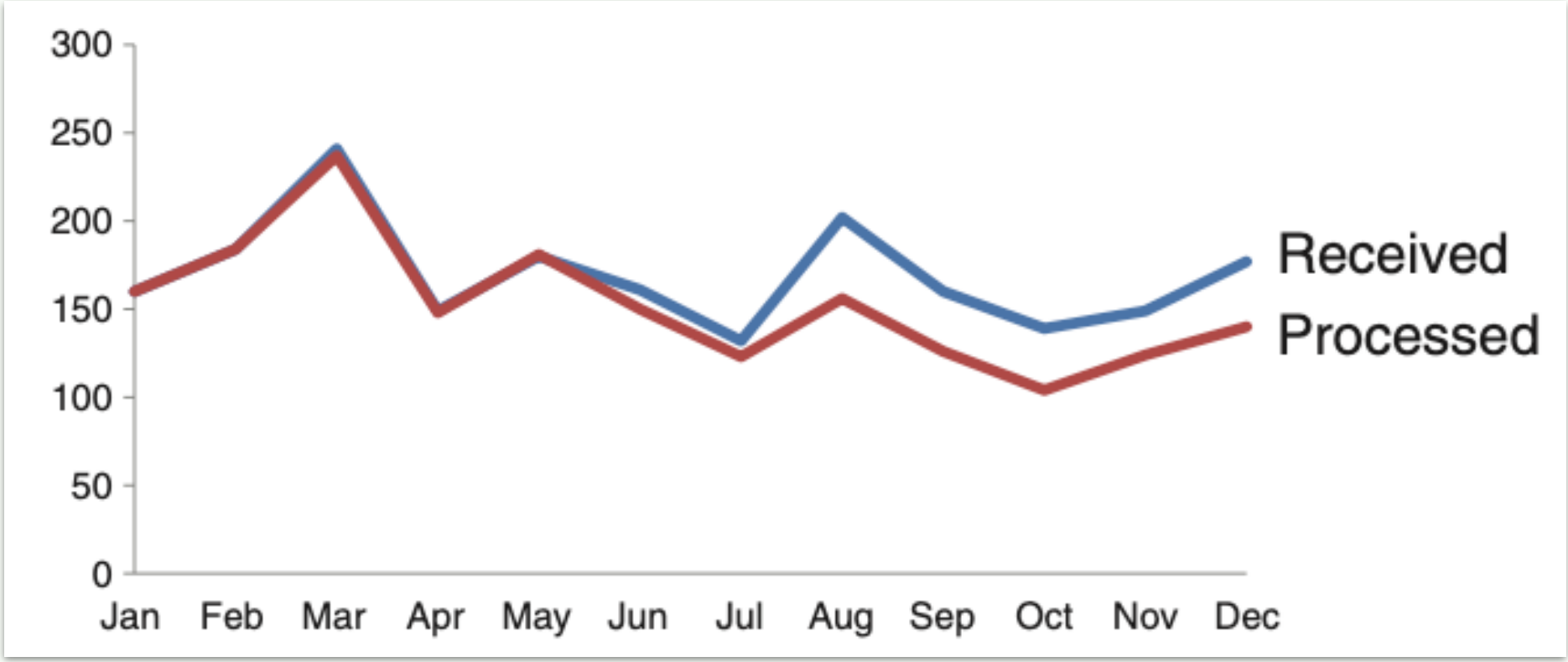
Step 3: Markers Are Redundant?



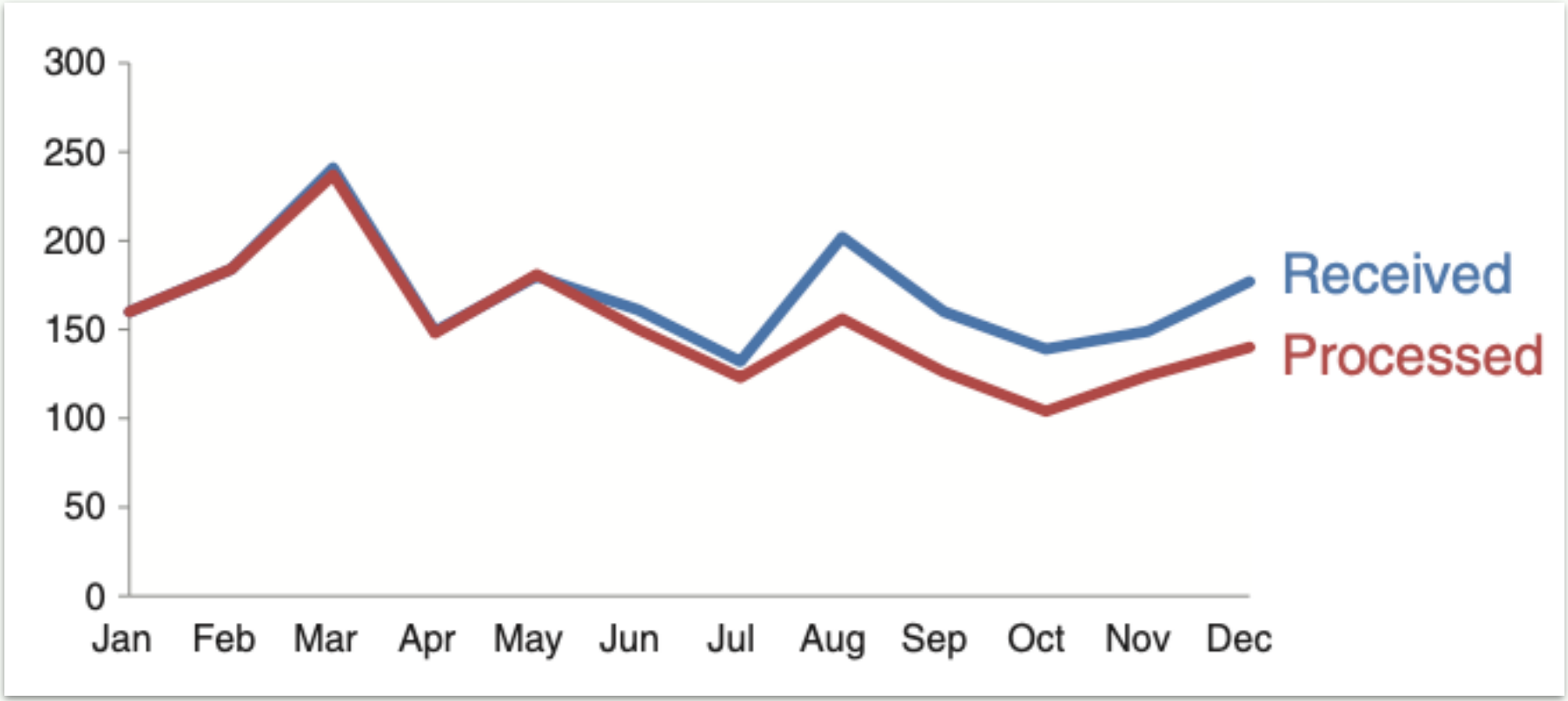
Step 4: Too Much Axis-Label Clutter



Step 5: Move Legend To Data

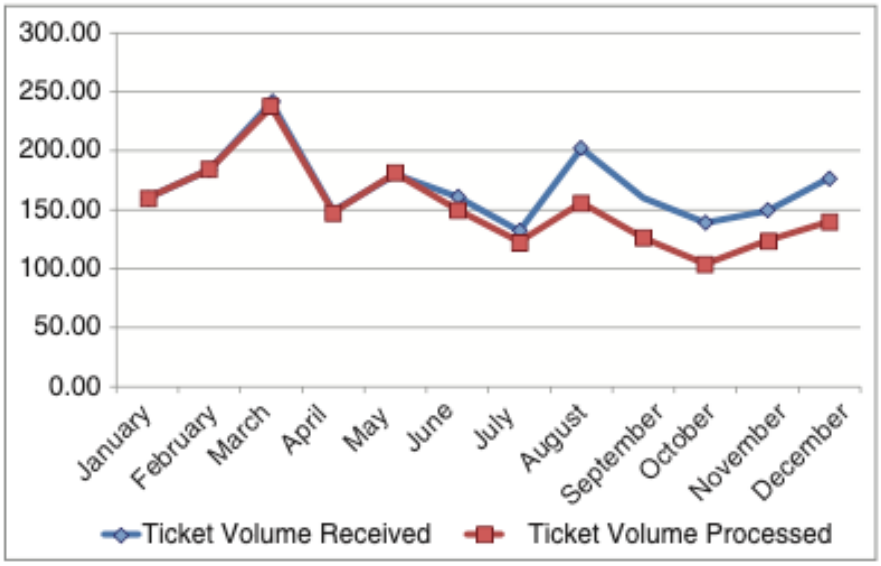


Step 6: Color Code Labels To Data

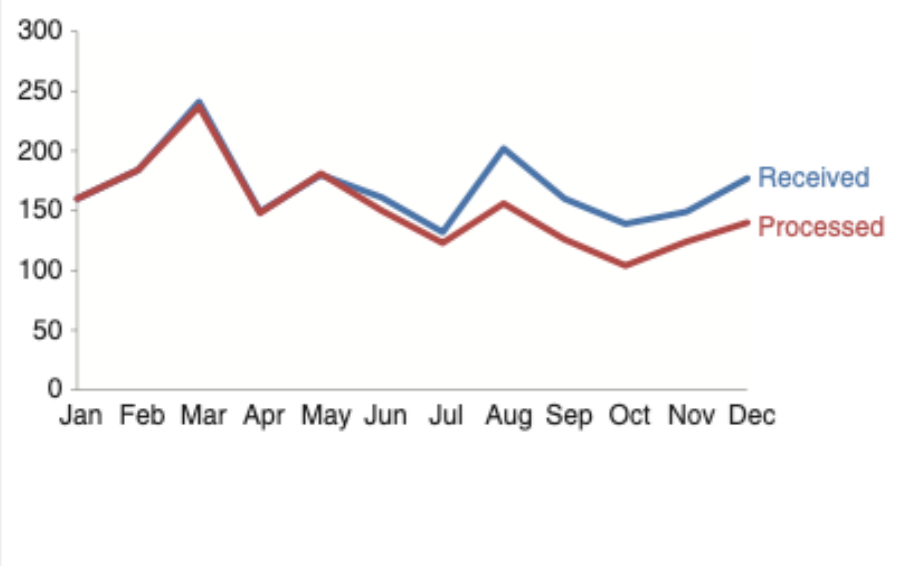


Summary So Far

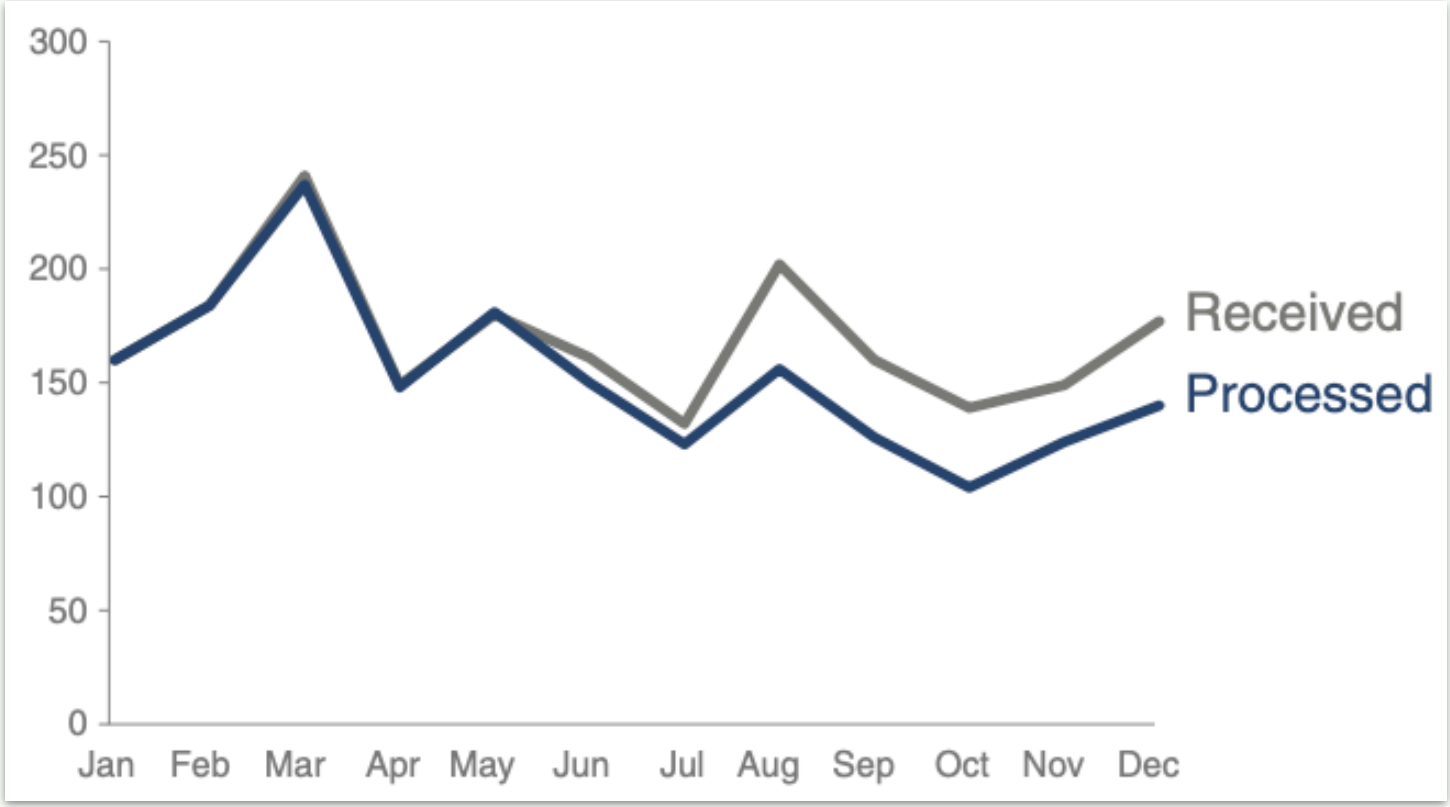
starting point



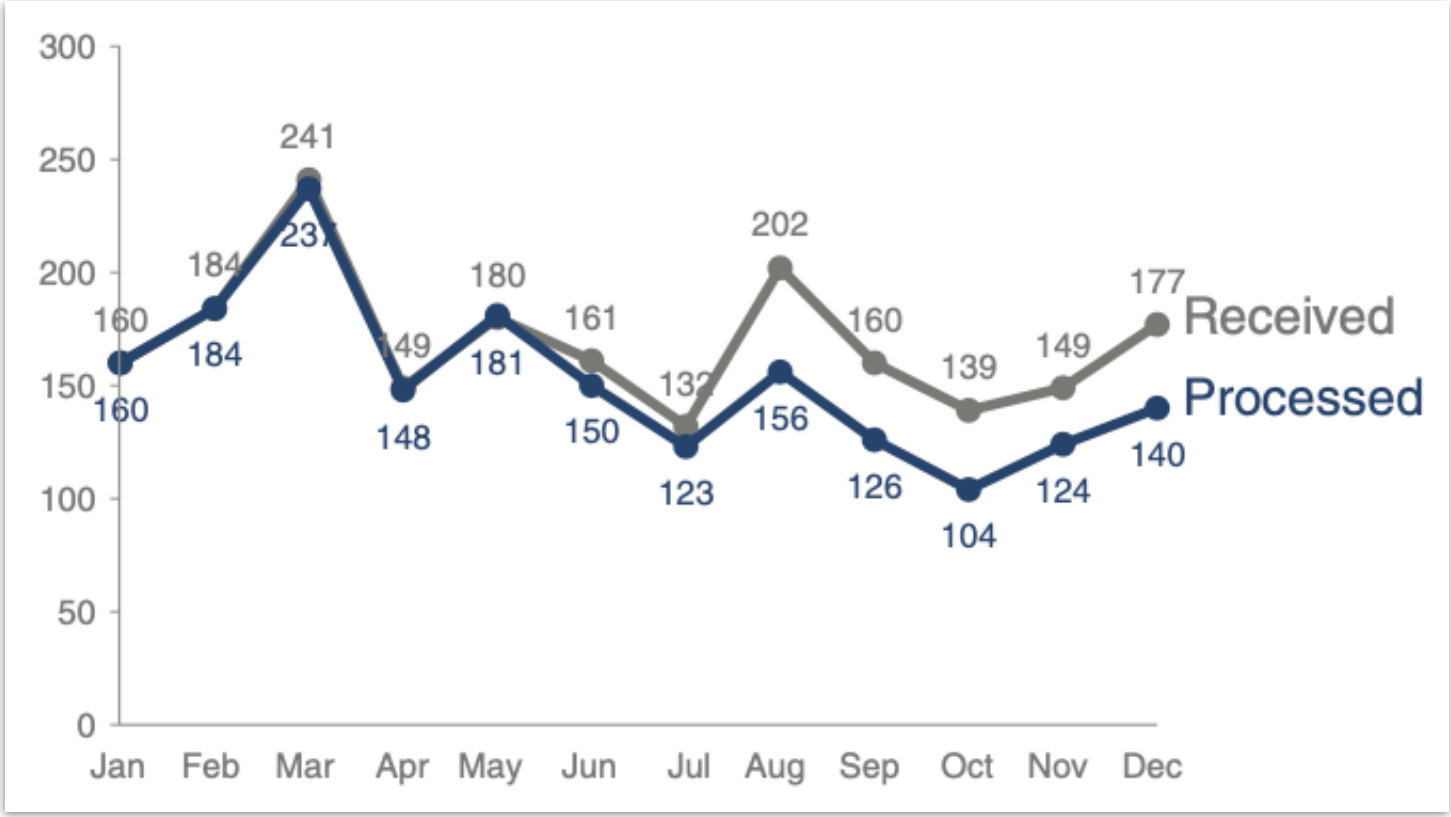
clean, quick to comprehend!



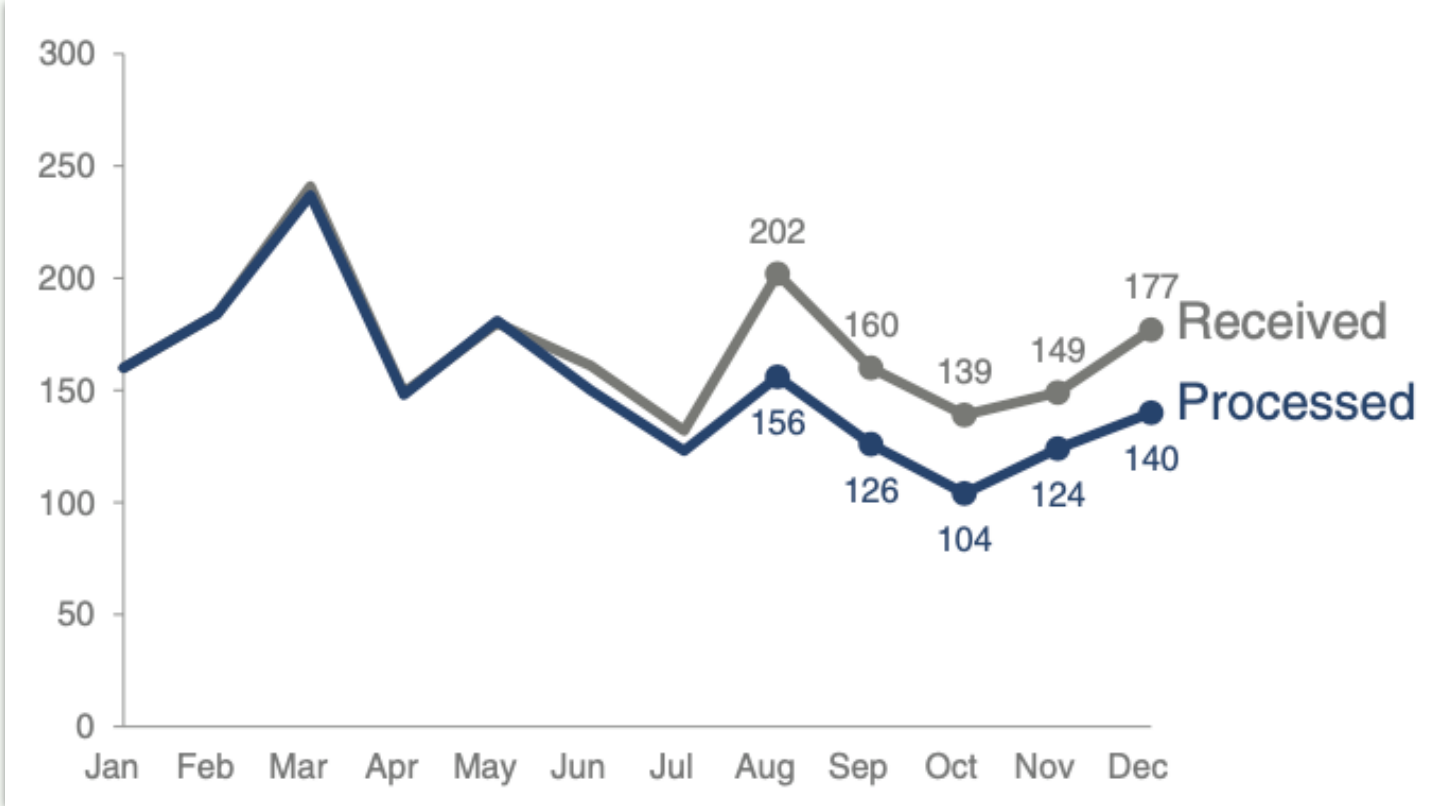
Step 7: Gray-Out Less Important Story



Step 8: Highlight Specific Values

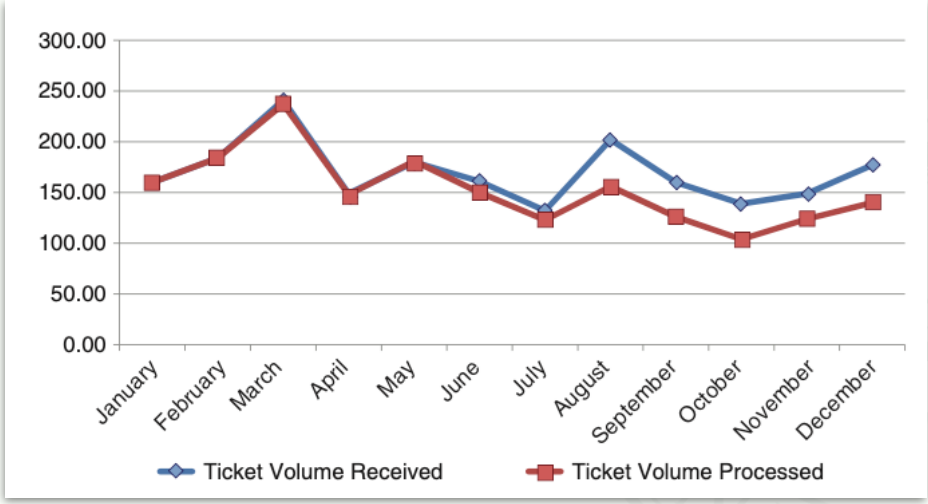


Step 9: But, Only Where The Story Is!



We're Done!

starting point



A story has emerged...