Lectures 10

CSS 507 Data Collection, Wrangling, Analysis and Visualization

By: PhD, Andrey Bogdanchikov
Edited by: MSc, Elnura Nabigazinova

Content

Introduction

There are many visualization techniques that can be used to visualize your data and there is no single solution make visualization perfect.

So in this lecture we will focus on the modern visualization library seaborn which is based on generally known matplotlib library and plays well with pandas dataframes.

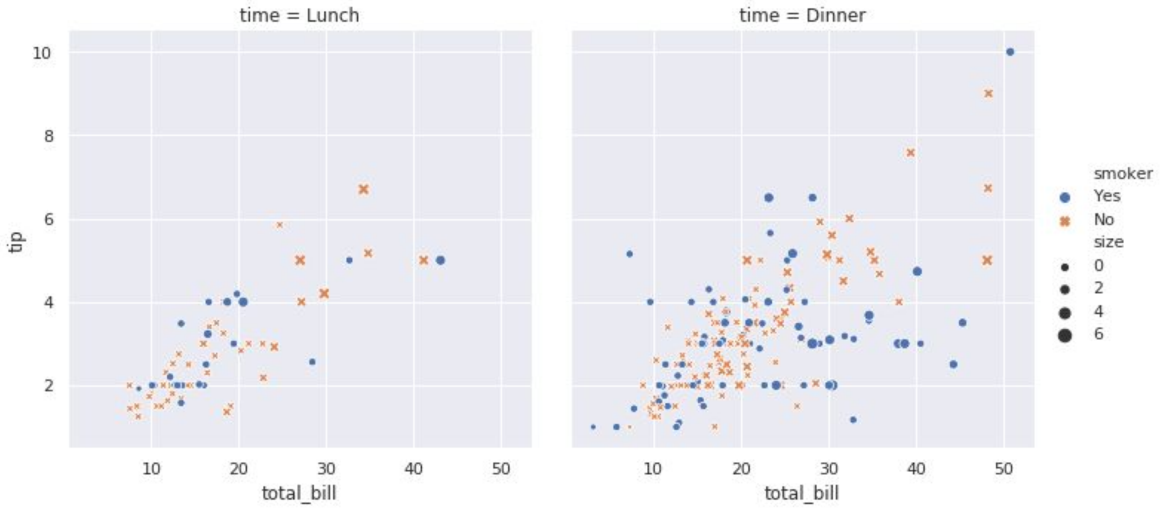
import seaborn as sns

Seaborn provides an API on top of Matplotlib that offers sane choices for plot style and color defaults, defines simple high-level functions for common statistical plot types, and integrates with the functionality provided by Pandas DataFrames.

Seaborn functionality

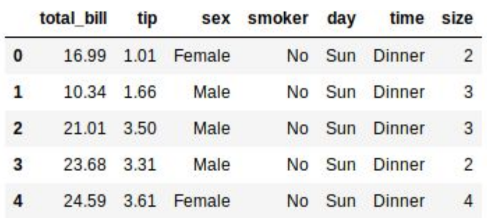
- A dataset-oriented API for examining relationships between multiple variables
- Specialized support for using categorical variables to show observations or aggregate statistics
- Options for visualizing univariate or bivariate distributions and for comparing them between subsets of data
- Automatic estimation and plotting of linear regression models for different kinds dependent variables
- Convenient views onto the overall structure of complex datasets
- High-level abstractions for structuring multi-plot grids that let you easily build complex visualizations
- Concise control over matplotlib figure styling with several built-in themes
- Tools for choosing color palettes that faithfully reveal patterns in your data

Visualization



- Seaborn aims to make visualization a central part of exploring and understanding data.
- Its dataset-oriented plotting functions operate on dataframes and arrays containing whole datasets and internally perform the necessary semantic mapping and statistical aggregation to produce informative plots.

Explanation



	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

```
sns.relplot(x="total_bill", y="tip", col="time",
        hue="smoker", style="smoker", size="size",
        data=tips)
```

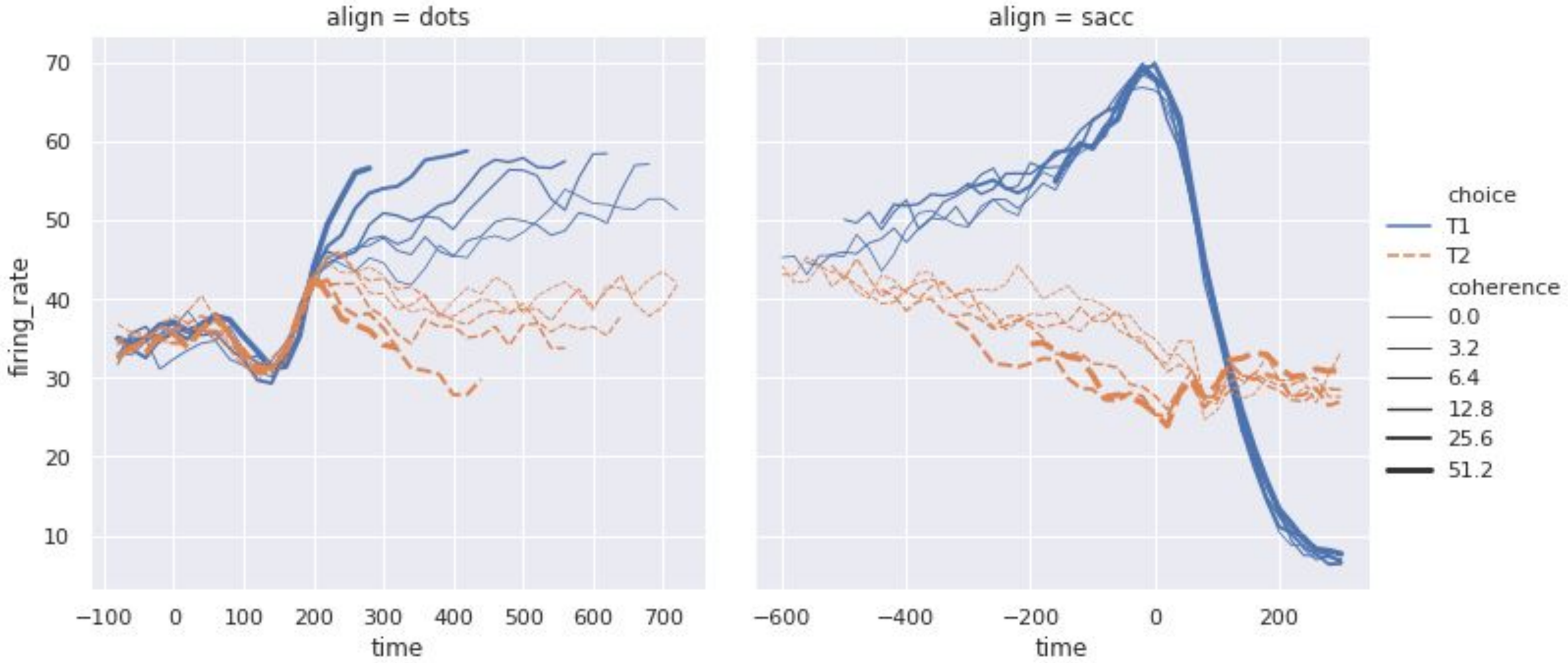
This particular plot shows the relationship between five variables in the tips dataset.
Three are numeric, and two are categorical. Two numeric variables (total_bill and tip) determined the position of each point on the axes, and the third (size) determined the size of each point. One categorical variable split the dataset onto two different axes (facets), and the other determined the color and shape of each point.

API abstraction across visualizations

There is no universal best way to visualize data. Different questions are best answered by different kinds of visualizations. Seaborn tries to make it easy to switch between different visual representations that can be parameterized with the same dataset-oriented API.

```
dots = sns.load_dataset("dots")
sns.relplot(x="time", y="firing_rate", col="align",
        hue="choice", size="coherence", style="choice",
        facet_kws=dict(sharex=False),
        kind="line", legend="full", data=dots);
```

Dots plot


align = dots
align = sacc
firing_rate
70
60
50
40
30
20
10
−100
0
100
200
300
400
500
600
700
time
−600
−400
−200
0
200
time
choice
T1
T2
coherence
0.0
3.2
6.4
12.8
25.6
51.2

relplot()

The function relplot() is named that way because it is designed to visualize many different statistical relationships.

While scatter plots are a highly effective way of doing this, relationships where one variable represents a measure of time are better represented by a line.

The relplot() function has a convenient kind parameter to let you easily switch to this alternate representation.

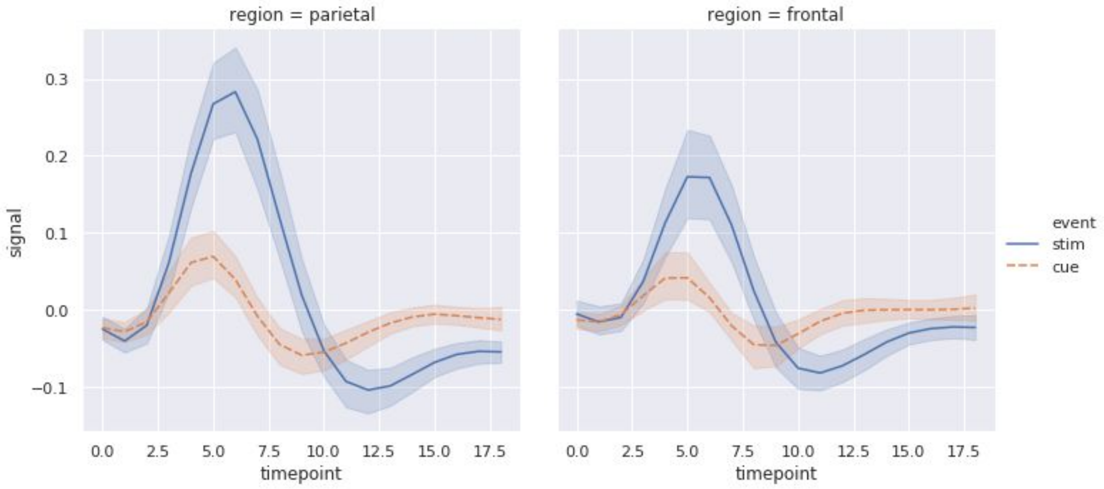
Statistical estimation and error bars

Often we are interested in the average value of one variable as a function of other variables. Many seaborn functions can automatically perform the statistical estimation that is necessary to answer these questions:

```
fmri = sns.load_dataset("fmri")
sns.relplot(x="timepoint", y="signal", col="region",
        hue="event", style="event",
        kind="line", data=fmri);
```

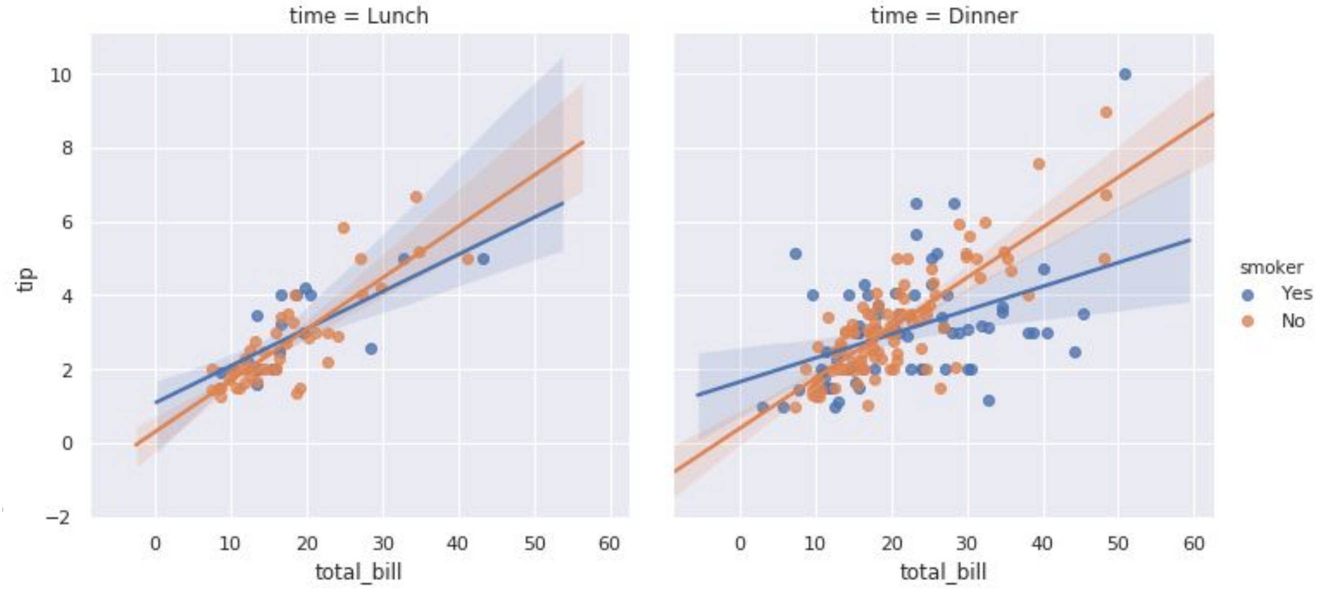
Estimates

When statistical values are estimated, seaborn will use bootstrapping to compute confidence intervals and draw error bars representing the uncertainty of the estimate.

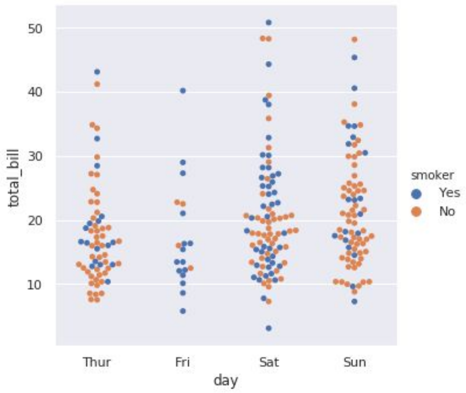


Statistical Estimates

Statistical estimation in seaborn goes beyond descriptive statistics. For example, it is also possible to enhance a scatterplot to include a linear regression model (and its uncertainty) using lmplot():



Specialized categorical plots

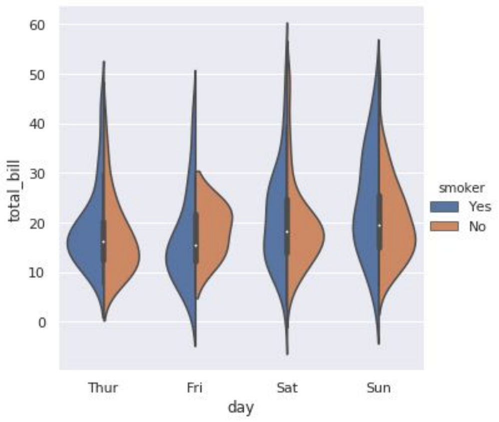


Standard scatter and line plots visualize relationships between numerical variables, but many data analyses involve categorical variables.

There are several specialized plot types in seaborn that are optimized for visualizing this kind of data. They can be accessed through catplot(). Similar to relplot(), the idea of catplot() is that it exposes a common dataset-oriented API that generalizes over different representations of the relationship between one numeric variable and one (or more) categorical variables.

```
sns.catplot(x="day", y="total_bill", hue="smoker",
        kind="swarm", data=tips);
```

Violin



Alternately, you could use kernel density estimation to represent the underlying distribution that the points are sampled from:

```
sns.catplot(x="day", y="total_bill", hue="smoker",
        kind="violin", split=True, data=tips);
```

Figure-level and axes-level functions

How do these tools work? It’s important to know about a major distinction between seaborn plotting functions. All of the plots shown so far have been made with “figure-level” functions.

These are optimized for exploratory analysis because they set up the matplotlib figure containing the plot(s) and make it easy to spread out the visualization across multiple axes. They also handle some tricky business like putting the legend outside the axes.

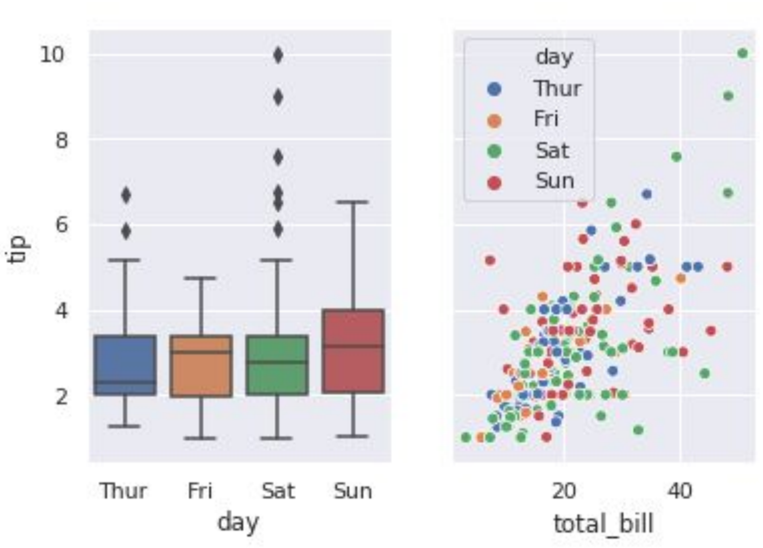
To do these things, they use a seaborn FacetGrid.

Combining different plots

Each different figure-level plot kind combines a particular “axes-level” function with the FacetGrid object. For example, the scatter plots are drawn using the scatterplot() function, and the bar plots are drawn using the barplot() function. These functions are called “axes-level” because they draw onto a single matplotlib axes and don’t otherwise affect the rest of the figure.

The upshot is that the figure-level function needs to control the figure it lives in, while axes-level functions can be combined into a more complex matplotlib figure with other axes that may or may not have seaborn plots on them:

Two plots

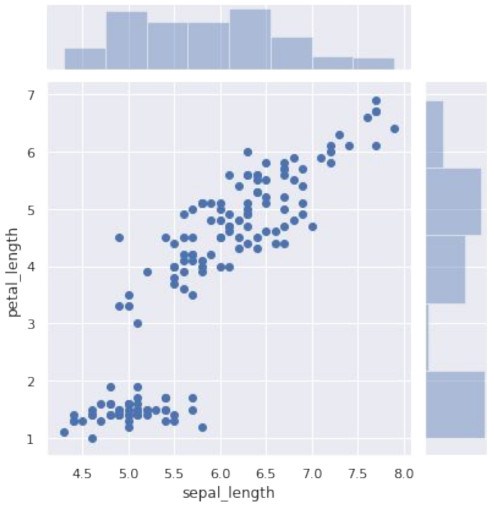


Controlling the size of the figure-level functions works a little bit differently than it does for other matplotlib figures.

Instead of setting the overall figure size, the figure-level functions are parameterized by the size of each facet.

```
import matplotlib.pyplot as plt
f, axes = plt.subplots(1, 2, sharey=True, figsize=(6, 4))
sns.boxplot(x="day", y="tip", data=tips, ax=axes[0])
sns.scatterplot(x="total_bill", y="tip", hue="day", data=tips,
ax=axes[1]);
```

Visualizing dataset structure

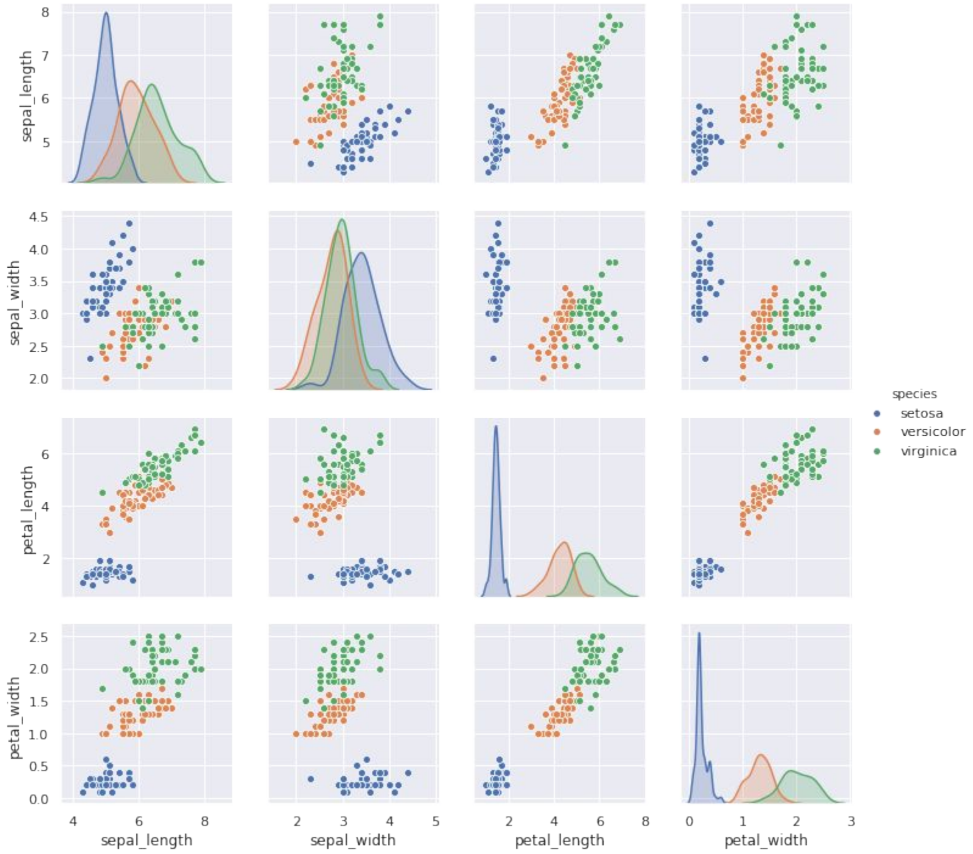


There are two other kinds of figure-level functions in seaborn that can be used to make visualizations with multiple plots. They are each oriented towards illuminating the structure of a dataset. One, jointplot(), focuses on a single relationship:

- iris = sns.load_dataset("iris")
  sns.jointplot(x="sepal_length", y="petal_length", data=iris);

The other, pairplot(), takes a broader view, showing all pairwise relationships and the marginal distributions, optionally conditioned on a categorical variable :

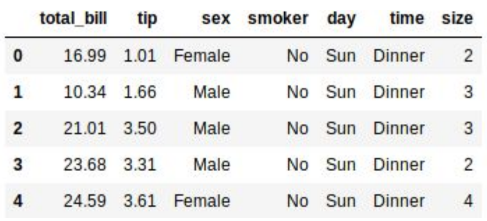
- sns.pairplot(data=iris, hue="species");



sns.pairplot()

Both jointplot() and pairplot() have a few different options for visual representation, and they are built on top of classes that allow more thoroughly customized multi-plot figures (JointGrid and PairGrid, respectively).

Organizing datasets



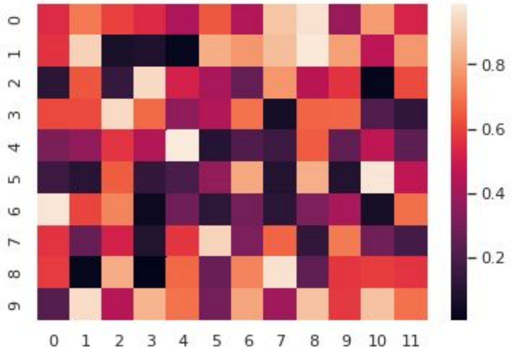
	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

As mentioned above, seaborn will be most powerful when your datasets have a particular organization. This format is alternately called “long-form” or “tidy” data and is described in detail by Hadley Wickham in this academic paper. The rules can be simply stated:

1. Each variable is a column
2. Each observation is a row

A helpful mindset for determining whether your data are tidy is to think backwards from the plot you want to draw. From this perspective, a “variable” is something that will be assigned a role in the plot. It may be useful to look at the example datasets and see how they are structured.

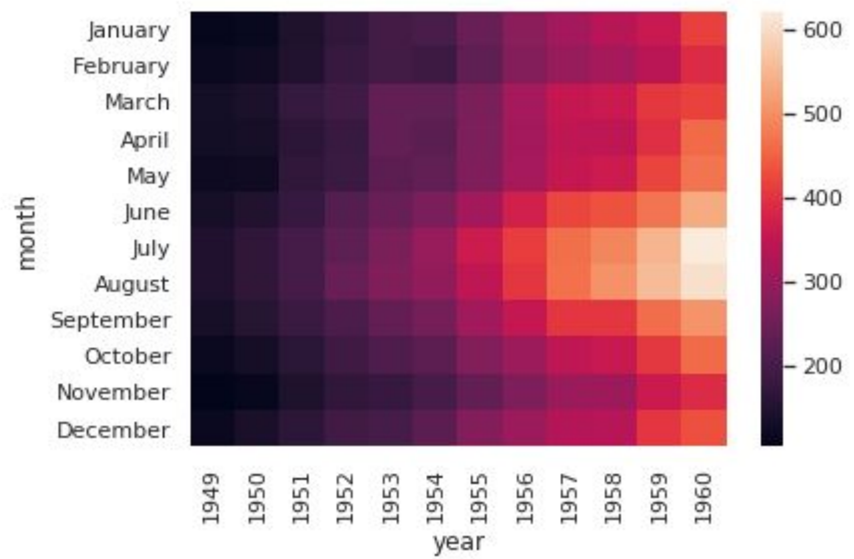
Heatmap



Plot rectangular data as a color-encoded matrix.

```
>>> import numpy as np; np.random.seed(0)
>>> import seaborn as sns; sns.set()
>>> uniform_data = np.random.rand(10, 12)
>>> ax = sns.heatmap(uniform_data)
```

heatmap



Plot a dataframe with meaningful row and column labels:

```
>>> flights = sns.load_dataset("flights")
>>> flights = flights.pivot("month", "year",
"passengers")
>>> ax = sns.heatmap(flights)
```

Visualization methods

How to Pick the Right Chart Type?

Making sense of facts, numbers, and measurements is a form of art – the art of data visualization. There is a load of data in the sea of noise. To turn your numbers into knowledge, your job is not only to separate noise from the data, but also to present it the right way.

Data Visualization Best Practices

There are four basic presentation types that you can use to present your data:

- Comparison
- Composition
- Distribution
- Relationship

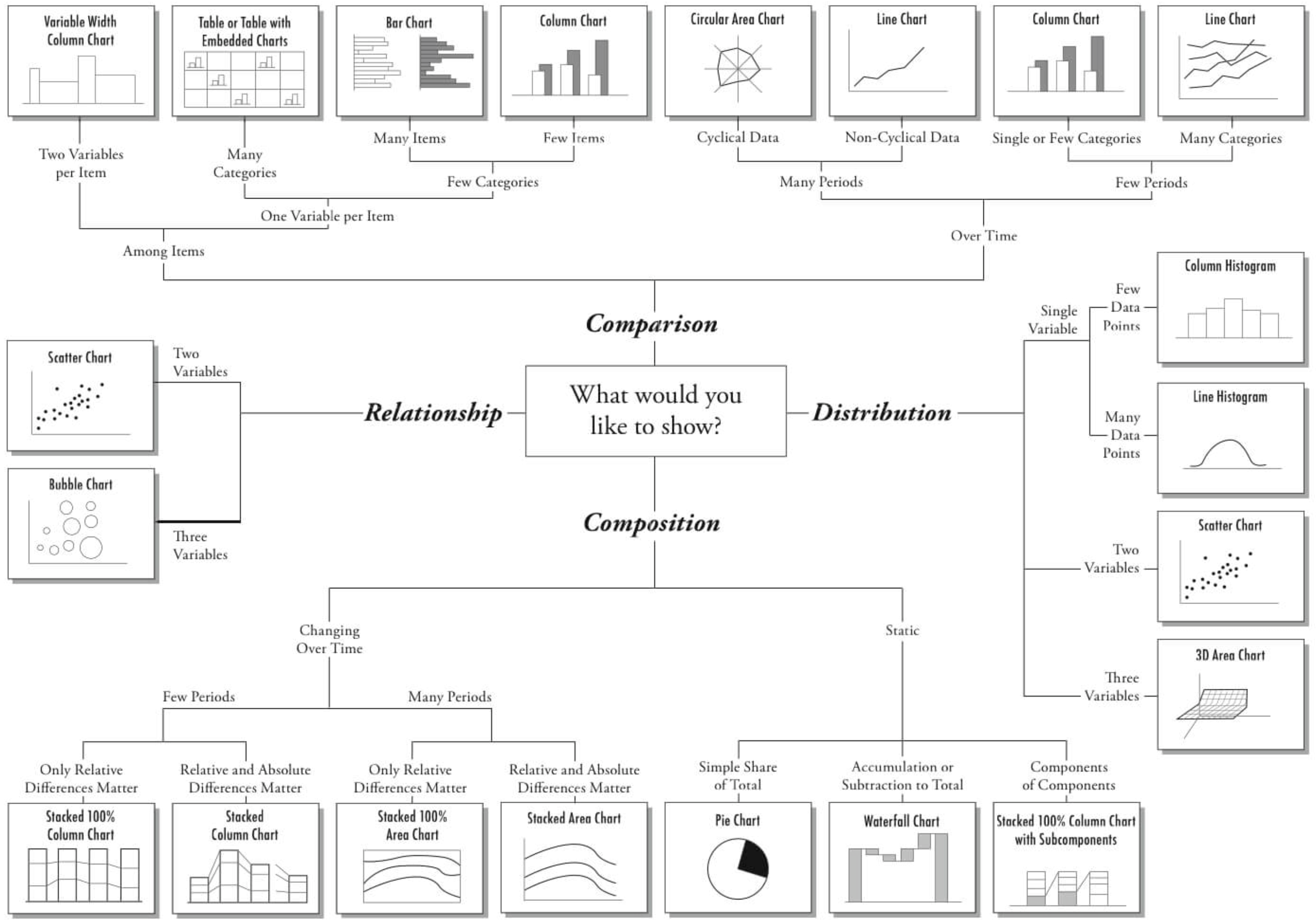
Unless you are a statistician or a data-analyst, you are most likely using only the two, most commonly used types of data analysis: Comparison or Composition.

Selecting the Right Chart

To determine which chart is best suited for each of those presentation types, first you must answer a few questions:

- How many variables do you want to show in a single chart? One, two, three, many?
- How many items (data points) will you display for each variable? Only a few or many?
- Will you display values over a period of time, or among items or groups?

Chart Suggestions—A Thought-Starter



Tables

Tables are essentially the source for all the charts. They are best used for comparison, composition, or relationship analysis when there are only few variables and data points. It would not make much sense to create a chart if the data can be easily interpreted from the table.

Use tables when:

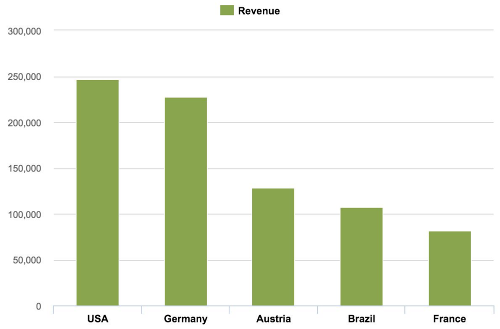
- You need to compare or look up individual values.
- You require precise values.
- Values involve multiple units of measure.
- The data has to communicate quantitative information, but not trends.

Use charts when the data presentation:

- Is used to convey a message that is contained in the shape of the data.
- Is used to show a relationship between many values.

For example, if you want to show the rate of change, like sudden drop of temperature, it is best to use a chart that shows the slope of a line because rate of change is not easily grasped from a table.

Column Charts



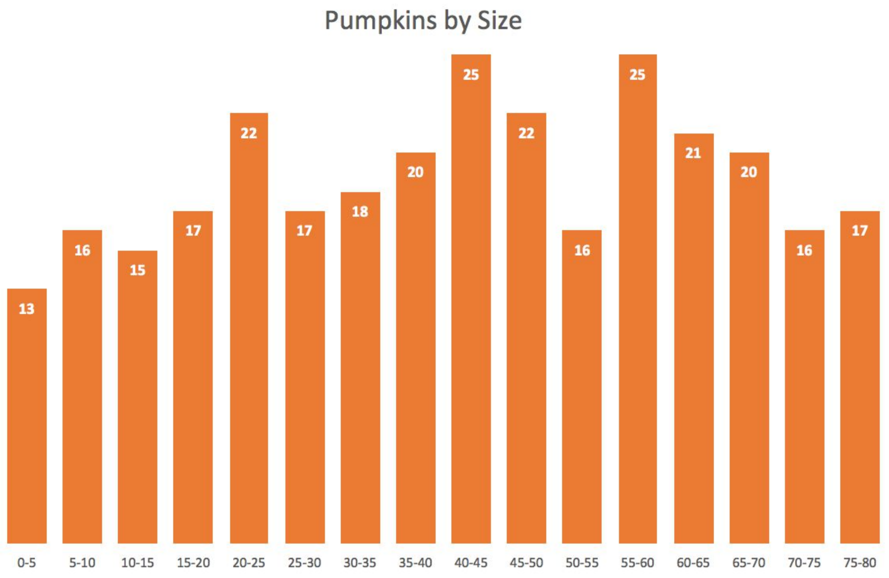
The column chart is probably the most used chart type. This chart is best used to compare different values when specific values are important, and it is expected that users will look up and compare individual values between each column.

With column charts you could compare values for different categories or compare value changes over a period of time for a single category.

Best practices for column charts

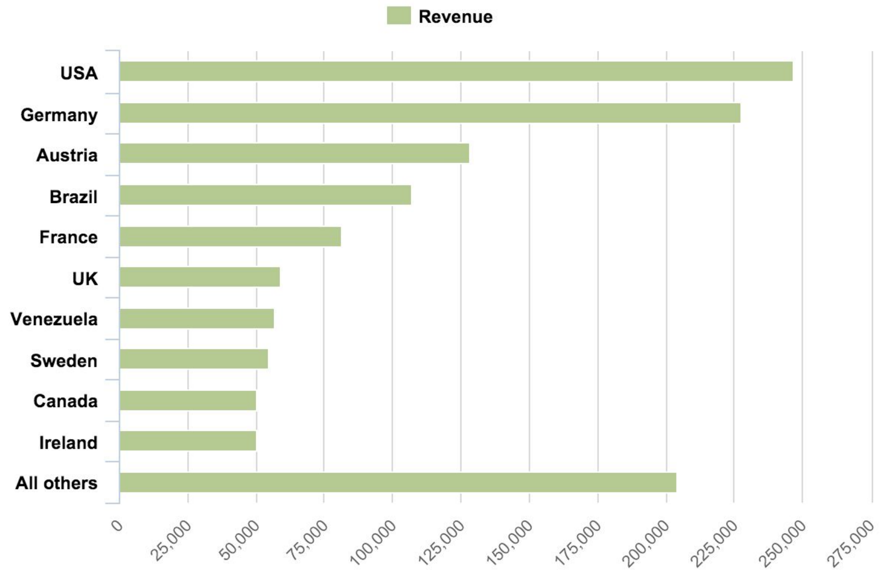
- Use column charts for comparison if the number of categories is quite small — up to five, but not more than seven categories.
- If one of your data dimensions is time — including years, quarters, months, weeks, days, or hours — you should always set time dimension on the horizontal axis.
- In charts, time should always run from left to right, never from top to bottom.
- For column charts, the numerical axis must start at zero. Our eyes are very sensitive to the height of columns, and we can draw inaccurate conclusions when those bars are truncated.
- Avoid using pattern lines or fills. Use border only for highlights.
- Only use column charts to show trends if there are a reasonably-low number of data points (less than 20) and if every data point has a clearly-visible value.

Column Histograms



Histogram is a common variation of column charts used to present distribution and relationships of a single variable over a set of categories. A good example of a histogram would be a distribution of grades on a school exam or the sizes of pumpkins, divided by size group, in a pumpkin festival.

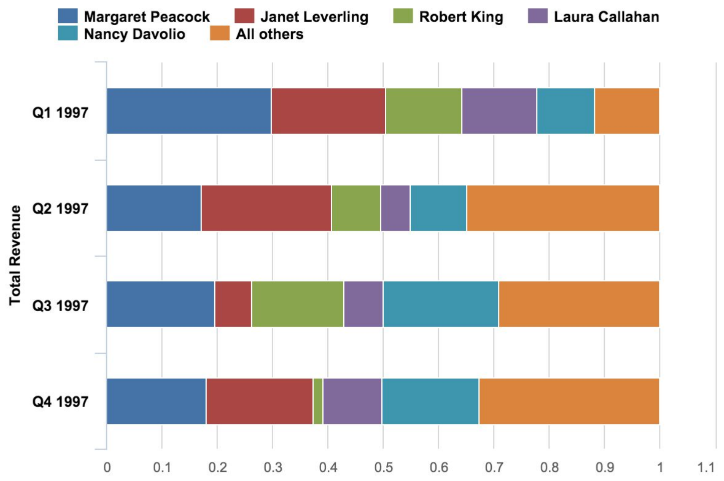
Bar Charts



Bar charts are essentially horizontal column charts.

If you have long category names, it is best to use bar charts because they give more space for long text. You should also use bar charts, instead of column charts, when the number of categories is greater than seven (but not more than fifteen) or for displaying a set with negative numbers.

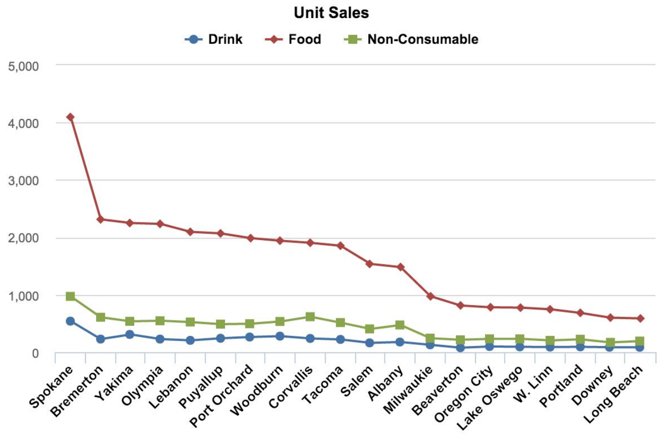
Stacked Bar Charts



I’m not quite sure about a good application of stacked bar charts — except when there are only a few variables, composition parts, and the emphasis is on composition, not comparison.

Stacked bars are not good for comparison or relationship analysis. The only common baseline is along the left axis of the chart, so you can only reliably compare values in the first series and for the sum of all series.

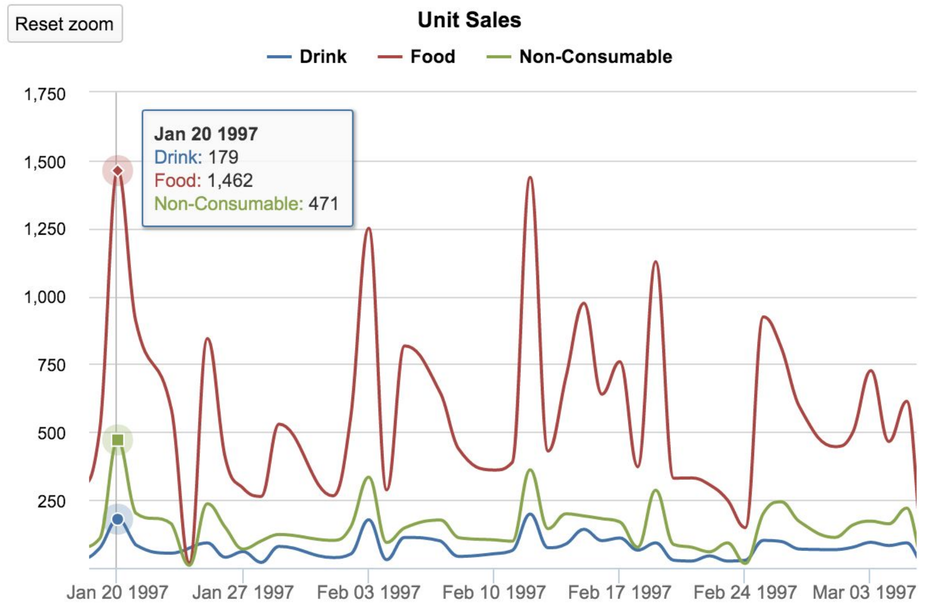
Line Charts



Line charts are among the most frequently used chart types. Use lines when you have a continuous data set. These are best suited for trend-based visualizations of data over a period of time, when the number of data points is very high (more than 20).

With line charts, the emphasis is on the continuation or the flow of the values (a trend), but there is still some support for single value comparisons, using data markers (only with less than 20 data points.)

Timeline Charts

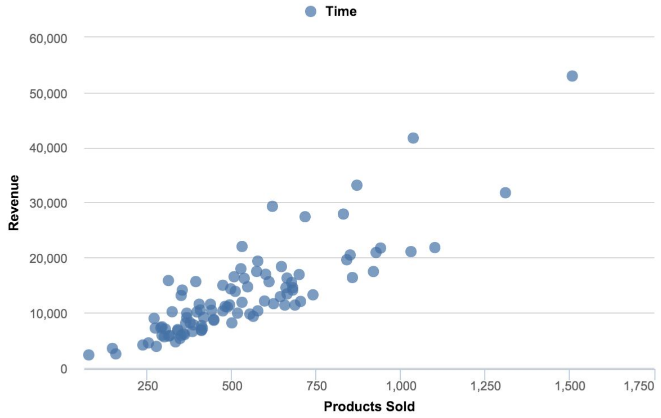


The timeline chart is a variation of line charts. Obviously, any line chart that shows values over a period of time is a timeline chart. The only difference is in functionality — most timeline charts will let you zoom in and out and compress or stretch the time axis to see more details or overall trends.

The Dos and Don’ts for Line Charts

- Use lines to present continuous data in an interval scale, where intervals are equal in size.
- For line charts, the axis may not start from zero if the intended message of the chart is the rate of change or overall trend, not exact values or comparison. It’s best to start the axis with zero for wide audiences because some people may otherwise interpret the chart incorrectly.
- In line charts, time should always run from left to right.
- Do not skip values for consistent data intervals presenting trend information, for example, certain days with zero values.
- Remove guidelines to emphasize the trend, rate of change, and to reduce distraction.
- Use a proper aspect ratio to show important information and avoid dramatic slope effects. For the best perception, aim for a 45-degree slope. (https://eagereyes.org/basics/banking-45-degrees)

Scatter Charts

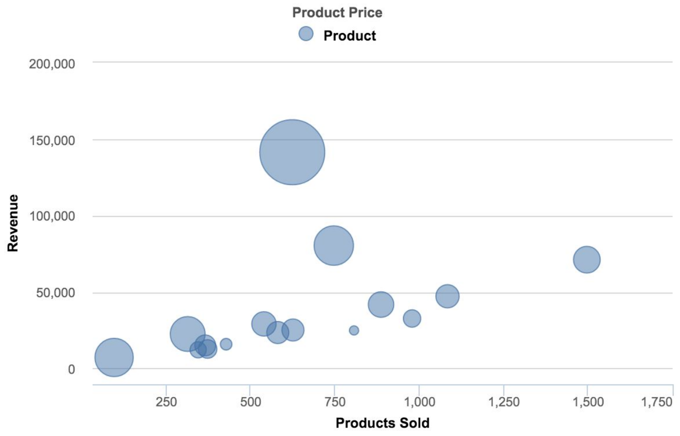


Scatter charts are primarily used for correlation and distribution analysis. Good for showing the relationship between two different variables where one correlates to another (or doesn't).

Scatter charts can also show the data distribution or clustering trends and help you spot anomalies or outliers.

A good example of scatter charts would be a chart showing marketing spending vs. revenue.

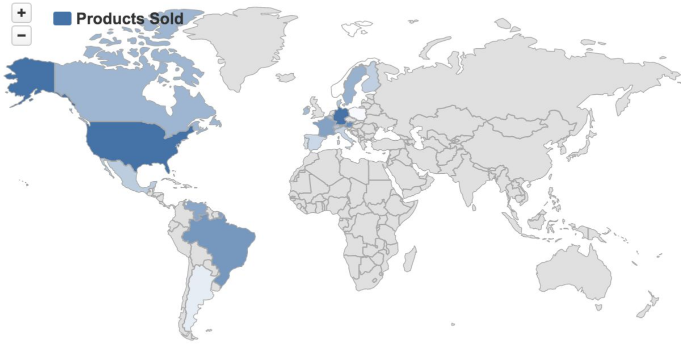
Bubble Charts



A bubble chart is a great option if you need to add another dimension to a scatter plot chart. Scatter plots compare two values, but you can add bubble size as the third variable and thus enable comparison. If the bubbles are very similar in size, use labels.

We could in fact add the fourth variable by color-grading those bubbles or displaying them as pie charts, but that’s probably too much.

Map Charts



Map charts are good for giving your numbers a geographical context to quickly spot best and worst performing areas, trends, and outliers. If you have any kind of location data like coordinates, country names, state names or abbreviations, or addresses, you can plot related data on a map.

Maps won’t be very good for comparing exact values, because map charts are usually color scaled and humans are quite bad at distinguishing shades of colors. Sometimes it’s better to use overlay bubbles or numbers if you need to convey exact numbers or enable comparison.

Data Visualization Do's and Don'ts – A General Conclusion

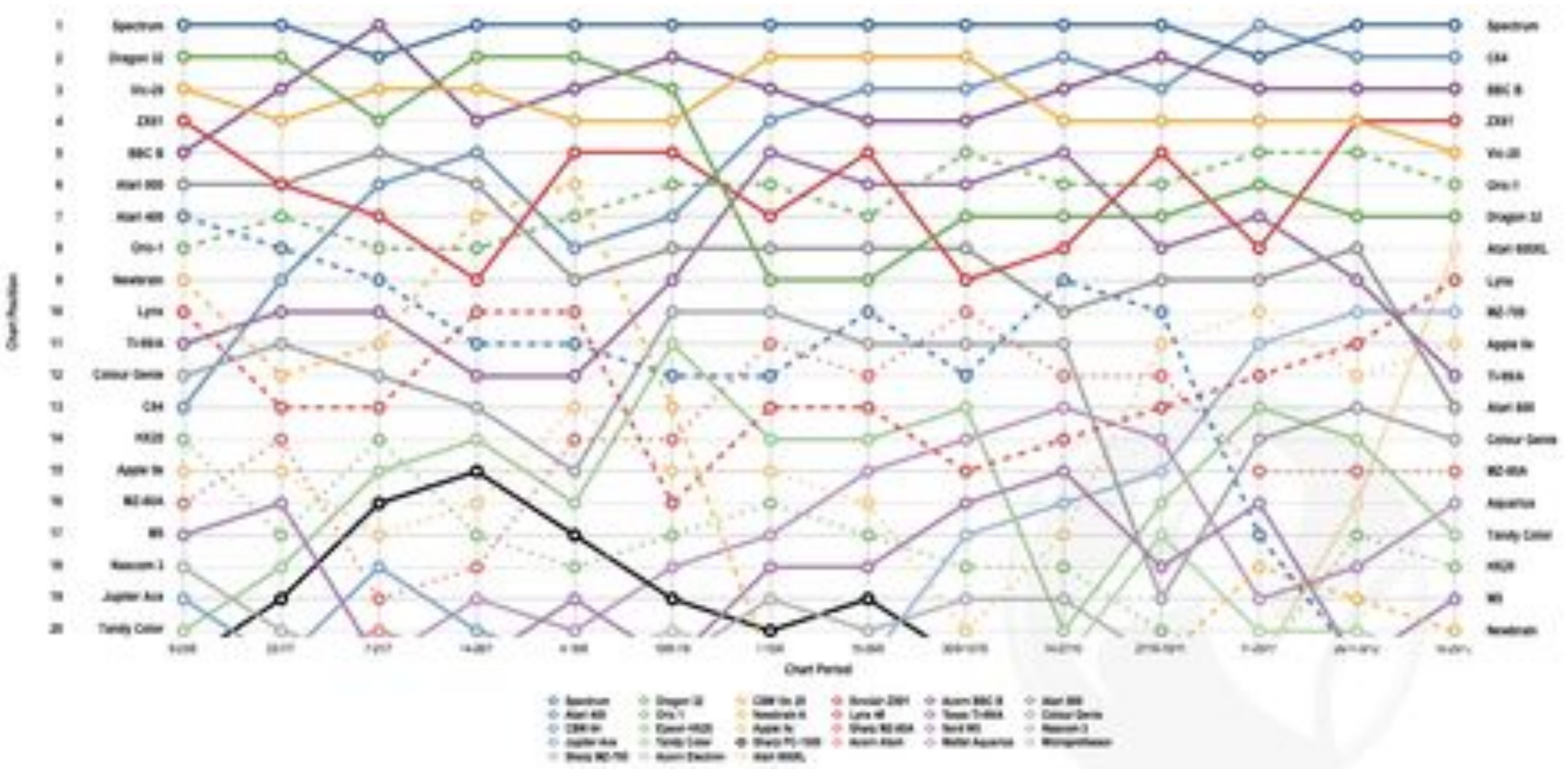
- Time axis. When using time in charts, set it on the horizontal axis. Time should run from left to right. Do not skip values (time periods), even if there are no values.
- Proportional values. The numbers in a chart (displayed as bar, area, bubble, or other physically measured element in the chart) should be directly proportional to the numerical quantities presented.
- Data-Ink Ratio. Remove any excess information, lines, colors, and text from a chart that does not add value. More about data-Ink ratio
- Sorting. For column and bar charts, to enable easier comparison, sort your data in ascending or descending order by the value, not alphabetically. This applies also to pie charts.

Data Visualization Do’s and Don’ts – A General Conclusion

- Legend. You don’t need a legend if you have only one data category.
- Labels. Use labels directly on the line, column, bar, pie, etc., whenever possible, to avoid indirect look-up.
- Inflation adjustment. When using monetary values in a long-term series, make sure to adjust for inflation. (EU Inflation rates, US InflationM rates)
- Colors. In any chart, don’t use more than six colors.
- Colors. For comparing the same value at different time periods, use the same color in a different intensity (from light to dark).
- Colors. For different categories, use different colors. The most widely used colors are black, white, red, green, blue, and yellow.

Data Visualization Do's and Don'ts – A General Conclusion

- Colors. Keep the same color palette or style for all charts in the series, and same axes and labels for similar charts to make your charts consistent and easy to compare.
- Colors. Check how your charts would look when printed out in grayscale. If you cannot distinguish color differences, you should change hue and saturation of colors.
- Colors. Seven to 10 percent of men have color deficiency. Keep that in mind when creating charts, ensuring they are readable for color-blind people. Use Vischeck to test your images. Or, try to use color palettes that are friendly to color-blind people.
- Data Complexity. Don't add too much information to a single chart. If necessary, split data in two charts, use highlighting, simplify colors, or change chart type. Credit: Junkcharts



Spectrum
Dragon 32
Vic-20
ZX81
BBC B
Atari 800
Atari 400
Oric-1
Newbrain
Lynx
TI-99/4
Colour Genie
C64
HX20
Apple IIe
MZ-80A
M5
Nascom 2
Jupiter Ace
Tandy Color
Chart Position
Chart Period
CBM 64
Atari 800XL
MZ-700
Apple IIe
Aquarius
Spectrum
Dragon 32
CBM Vic 20
Sinclair ZX81
Acorn BBC B
Atari 800
Atari 400
Oric 1
Newbrain A
Lynx 48
Texas TI-99/4
Colour Genie
CBM 64
Epson HX20
Apple IIe
Sharp MZ-80A
Sord M5
Nascom 2
Jupiter Ace
Tandy Color
Sharp PC-1500
Acorn Atom
Mattel Aquarius
Microprofessor
Sharp MZ-700
Acorn Electron
Atari 800XL

Conclusion

Check following links for info:

- Seaborn docs - https://seaborn.pydata.org/introduction.html
- Tidy data- http://vita.had.co.nz/papers/tidy-data.html
- Visualization types - https://eazybi.com/blog/data_visualization_and_chart_types/

Extra

- chart-selection-diagram.png

Thank You