Python: Data Visualization Notes

Klein
carlj.klein@gmail.com

1 Learning Objectives

What good is a data analysis if the answers or findings can't be conveyed properly? Or perhaps even more detrimental, what good is a dataset if the proper questions or hypotheses can't be formed in the first place? How can we properly tell the story the data is providing? Enter in data visualization! Both explaining the data and exploring the data can be significantly helped through the process of data visualization.

Data visualization is two-pronged:

- Exploratory Analysis: Helping to understand the data prior to analysis. Searching for relationships and insights.
- Explanatory Analysis: Helping to present analysis findings, helping to tell a story with the data. After insights were found.

And it's all a part of the entire data analysis process. We can also simplify the data analysis process into 5 steps:

1. Extract
2. Clean: Exploratory
3. Explore: Exploratory
4. Analyze: Exploratory OR Explanatory
5. Share: Explanatory

In this course, we'll be using the matplotlib, seaborn, and Pandas libraries to assist in the data analysis process.

Concepts

- Design of Visualization
- Exploration of Data

  - Univariate Exploration of Data
  - Bivariate Exploration of Data
  - Multivariate Exploration of Data
- Explanatory Visualizations
- Visualization Case Study

2 Design of Visualization

To begin our discussion of visualization, we need to cover some basic vocabulary and distinctions.
Data can be broken into two main categories, each of which can be broken down further:

- Qualitative / Categorical:
  - Nominal: No order
  - Ordinal: Intrinsic Order
- Quantitative / Numerical:
  - Interval: Absolute differences are meaningful (addition and subtraction follows logic)
  - Ratio: relative differences are meaningful (multiplication and division follows logic)

It should be noted that the quantitative data type can be also be broken down into discrete and continuous variables.

What about those 3-dimensions charts or fun backgrounds that we used to add to our science experiment plots as kids? That used to add some fun to our projects, right? While fun for the youth, it turns out there is an empirical rule when figuring out how much the additional "junk" either adds or detracts from conveying the data.

- Data-Ink Ratio = data-ink / total ink used to print the graphic. The higher the Data-Ink Ratio, the better conveyed data is.

Can visualizations be purposefully misleading, even when using the data appropriately? Absolutely!

A great example of this is trying to over-inflate the difference or change between data points during different time periods. Say a presenter is trying to make a claim there was a very large change from one year to the next. We'll say in year

1 the y-value was 100, and in year 2 the y-value was 105. The presenter changes the window of the graph to display from a y-value of 99 to a y-value of 106, and the x-values are only the two years. Obviously, this is going to look like a massive change! In reality, had the reporter shown the data at a true scale, the visual shows in actuality that the change isn't so tremendous.

This concept also has an empirical rule:

- Lie Factor = size of effect shown in graphic / size of effect shown in data

  = (change in visual / visual start) / (change in data / data start)

  As was said in the example, this can be used to purposefully distort data. In fact, a Lie Factor 1 suggests a misleading visual, and even greater than that suggest an even greater disparity from the truth.

Away from the empirical side of visuals, and more into the logical, we come across the common mistake of using too many colors! Colors can be useful when separating categories, however, they it's very easy to cause redundancy with them. Here are some tips when using color:

- Get it right in black and white (and shades of grey)
- Use less intense colors such as natural or pastel, and higher grey colors. The eye can actually concentrate longer under these conditions.
- Color facilitates communication. Use color to separate the data into groups of interest, not just to color a visual.
- Design for Color Blindness. Stay away from red / green pallets, and use blue / orange pallets.

Don't want to overdo it on the color schemes? Don't forget about other visual queues such as shape and size.

Some tips on shape, size & other tools:

- Use different types of encodings, rather than using color (square / dot vs. colors to separate groups of interest).
- Color and shape are good for categorical variables.
- Size of marker can assist in adding additional quantitative data.

3 Exploration of Data

We have a dataset on a topic or concept that has been deemed worthy for inspection! Surely, there are some insights to be gained from it. We load up the data, and then we hit a wall... Which columns are important? What questions can be answered? We can find the general statistics of the set, so what?

This section will help with the initial process of data exploration, helping to find what is actually useful and should be examined further in the data. We'll start with single variables from the data and move into visually pairing multiple data points at once.

We'll be using a few different Python libraries in this section:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
```

3.1 Univariate Exploration of Data

The first tools added into our data exploration toolkit will be for univariate data, or examining the data a single variable at a time. Even though we're going to be using several different libraries, we'll still try to keep track of new and essential commands and terminology:

Commands:

- df.head: Retrieves the first few rows of data from a Pandas DataFrame. Able to specify how many rows
- sb.countplot: Seaborn's command for generating a bar chart.
- df[column].value_counts().index: Will return an immutable sequence sorted number of categorical entries (highest to lowest).
- df.melt: Will unpivot a DataFrame from wide to long format (i.e. we can "melt" two entries together).

Terminology:

- bar chart: Useful in univariate data to show counts across categories.
- relative frequency: Shows proportion of each category in population.

We'll be using a Pokemon dataset for our examples. Let's get an idea of what our data looks like:

```
pokemon = pd.read_csv("pokemon.csv")
pokemon.shape
# pokemon.shape = (807, 14)
pokemon.head(10)
```

Pokemon.Head(10)

	id	species	generation_id	height	weight	base_experience	type_1
0	1	bulbasaur	1	0.7	6.9	64	grass
1	2	ivysaur	1	1.0	13.0	142	grass
2	3	venusaur	1	2.0	100.0	236	grass
3	4	charmander	1	0.6	8.5	62	fire
4	5	charmeleon	1	1.1	19.0	142	fire
5	6	charizard	1	1.7	90.5	240	fire
6	7	squirtle	1	0.5	9.0	63	water
7	8	wartortle	1	1.0	22.5	142	water
8	9	blastoise	1	1.6	85.5	239	water
9	10	caterpie	1	0.3	2.9	39	bug

	type_2	hp	attack	defense	speed	special-attack	special-defense
0	poison	45	49	49	45	65	65
1	poison	60	62	63	60	80	80
2	poison	80	82	83	80	100	100
3	NaN	39	52	43	65	60	50
4	NaN	58	64	58	80	80	65
5	flying	78	84	78	100	109	85
6	NaN	44	48	65	43	50	64
7	NaN	59	63	80	58	65	80
8	NaN	79	83	100	78	85	105
9	NaN	45	30	35	45	20	20

Now that we've had a peak at the data, let's get into some visualization.

```
# create a bar chart for one of the columns (generation_id)
# this will show counts of each pokemon for each generation
sb.countplot(data = pokemon, x = 'generation_id')
# See Figure 1

# let's use a single neutral color to reduce redundancy and make
                                it easier on the eyes
base_color = sb.color_palette()[0]
sb.countplot(data = pokemon, x = 'generation_id', color =
                                base_color)
# See Figure 2
```

A simple, yet effective visual trick is adding some order to your plots via sorting.

```
# let's sort the data highest -> lowest
# we can use pandas series function: value_counts()
gen_order = pokemon['generation_id'].value_counts().index
sb.countplot(data = pokemon, x = 'generation_id', color =
                                base_color, order = gen_order)
# See Figure 3
```

Let's combine the previous coloring and sorting to look at a new variable. Additionally, since we know the x-axis labels are somewhat long. Two favorable ways

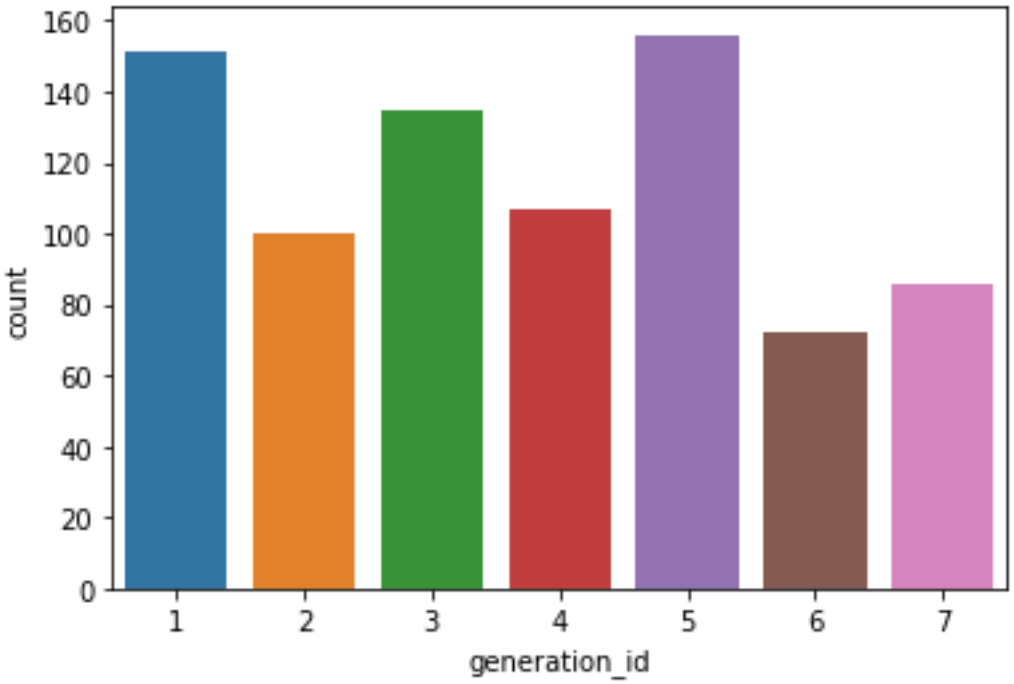

Figure 1: Generic bar chart

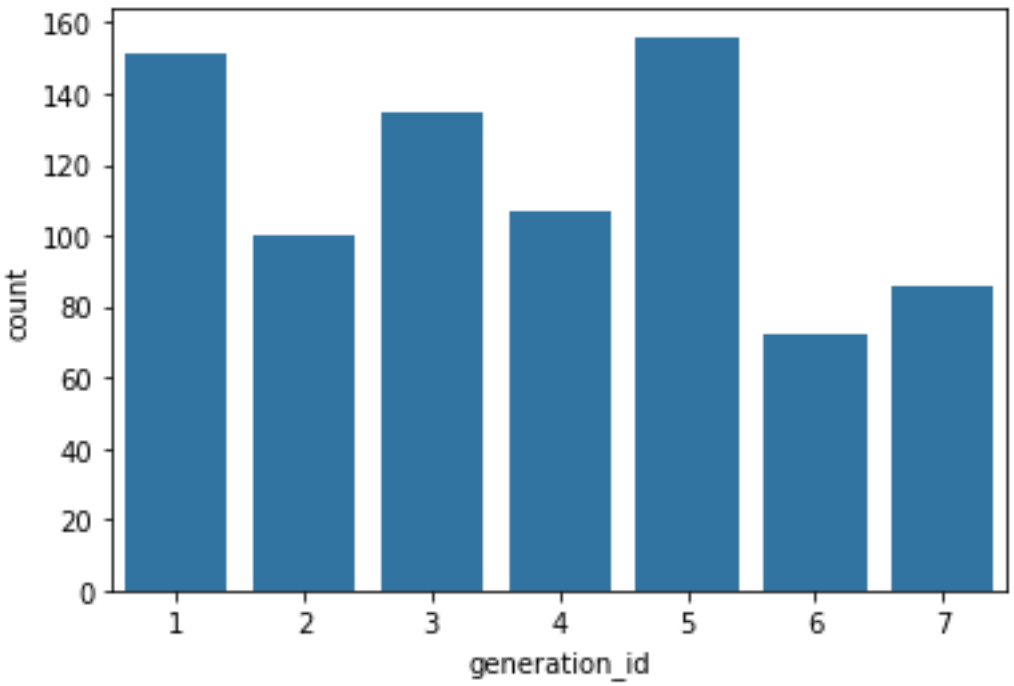

Figure 2: Bar chart with neutral coloring

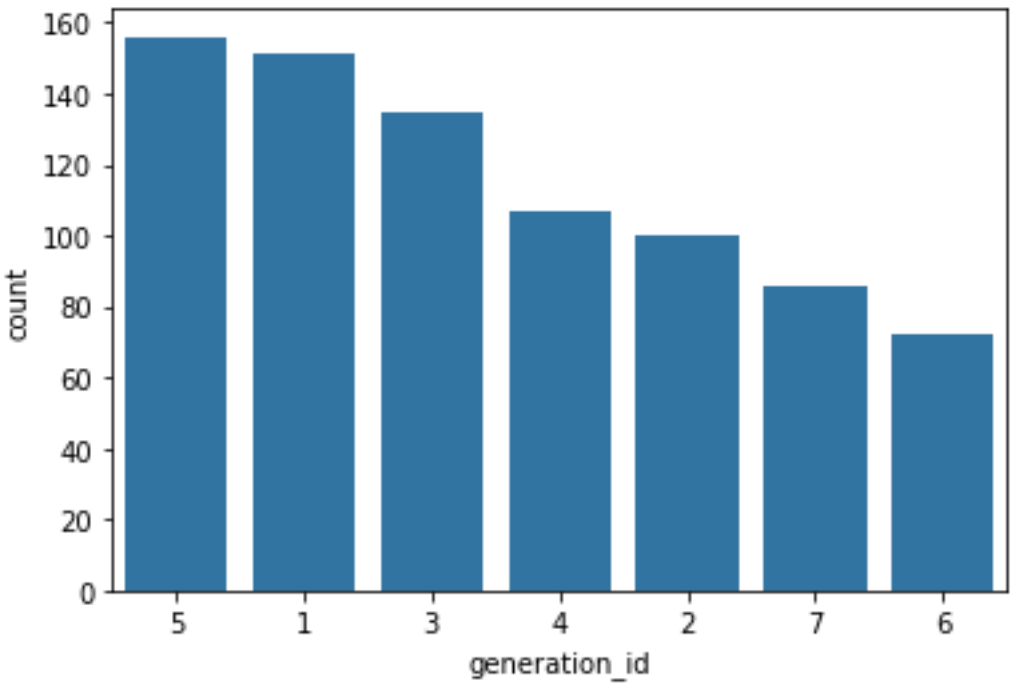

Figure 3: Bar chart with neutral coloring and ordering applied

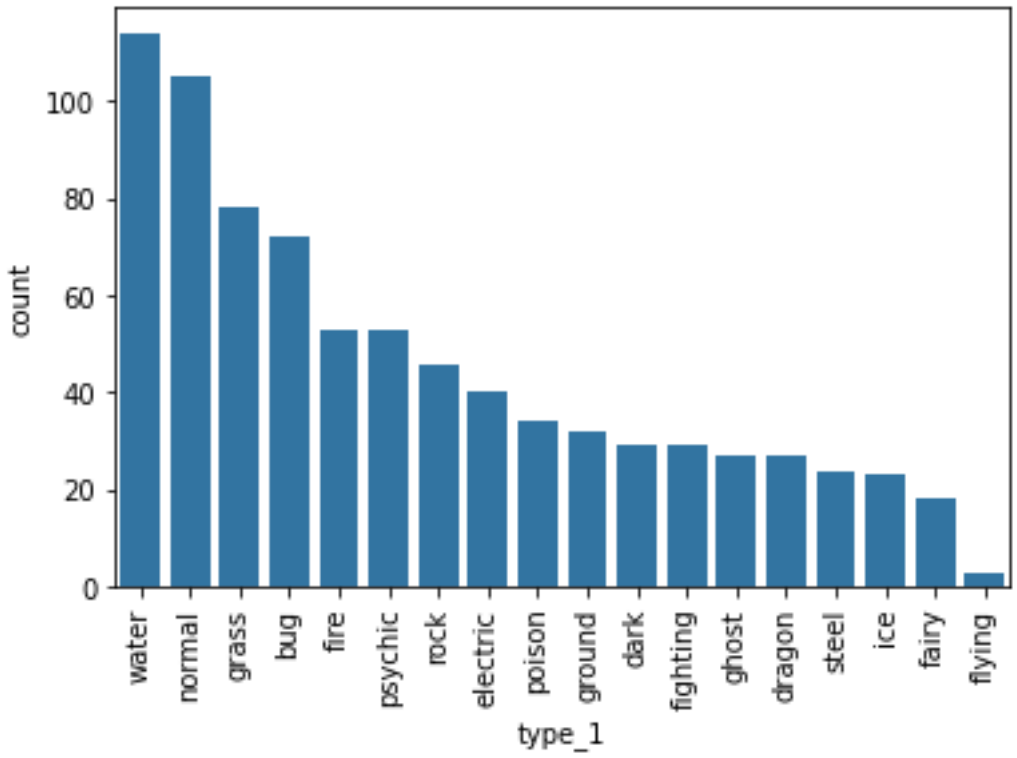

Figure 4: Standard bar chart format with x-axis labels rotated

of dealing with this is to either rotate the x-axis labels, or create a horizontal bar chart.

```
type_order = pokemon['type_1'].value_counts().index
sb.countplot(data = pokemon, x = 'type_1', color = base_color,
                                order = type_order)
plt.xticks(rotation = 90)
# See Figure 4

# Change x to y to change from vertical to horizontal bar chart
sb.countplot(data = pokemon, y = 'type_1', color = base_color,
                                order = type_order)
# See Figure 5
```

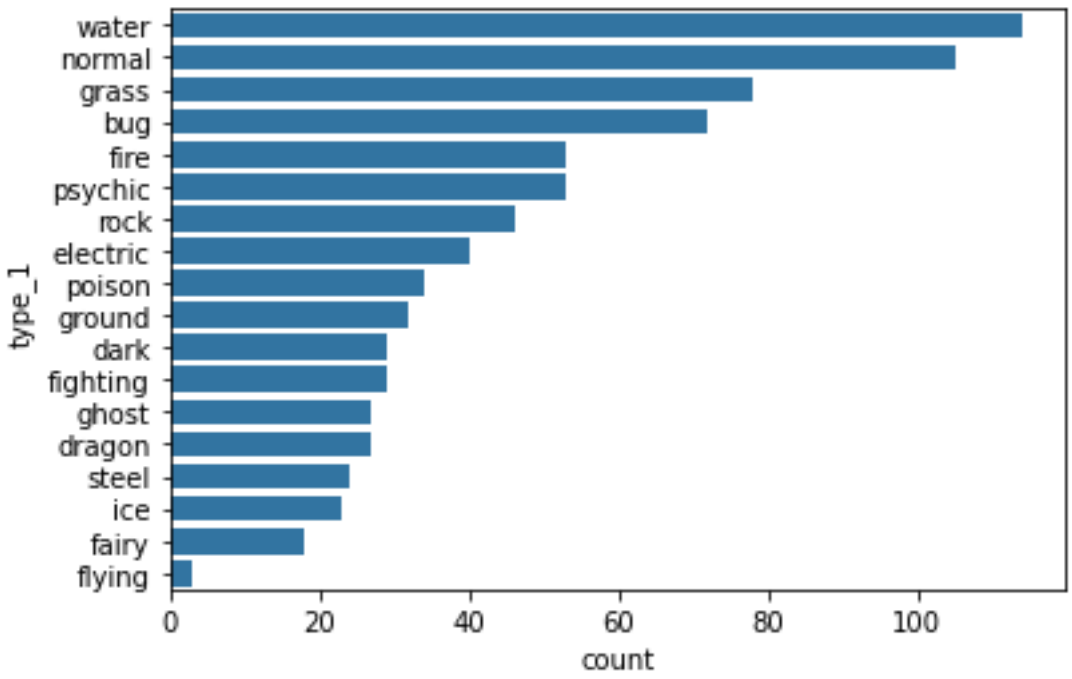

Figure 5: Standard bar chart format in vertical form