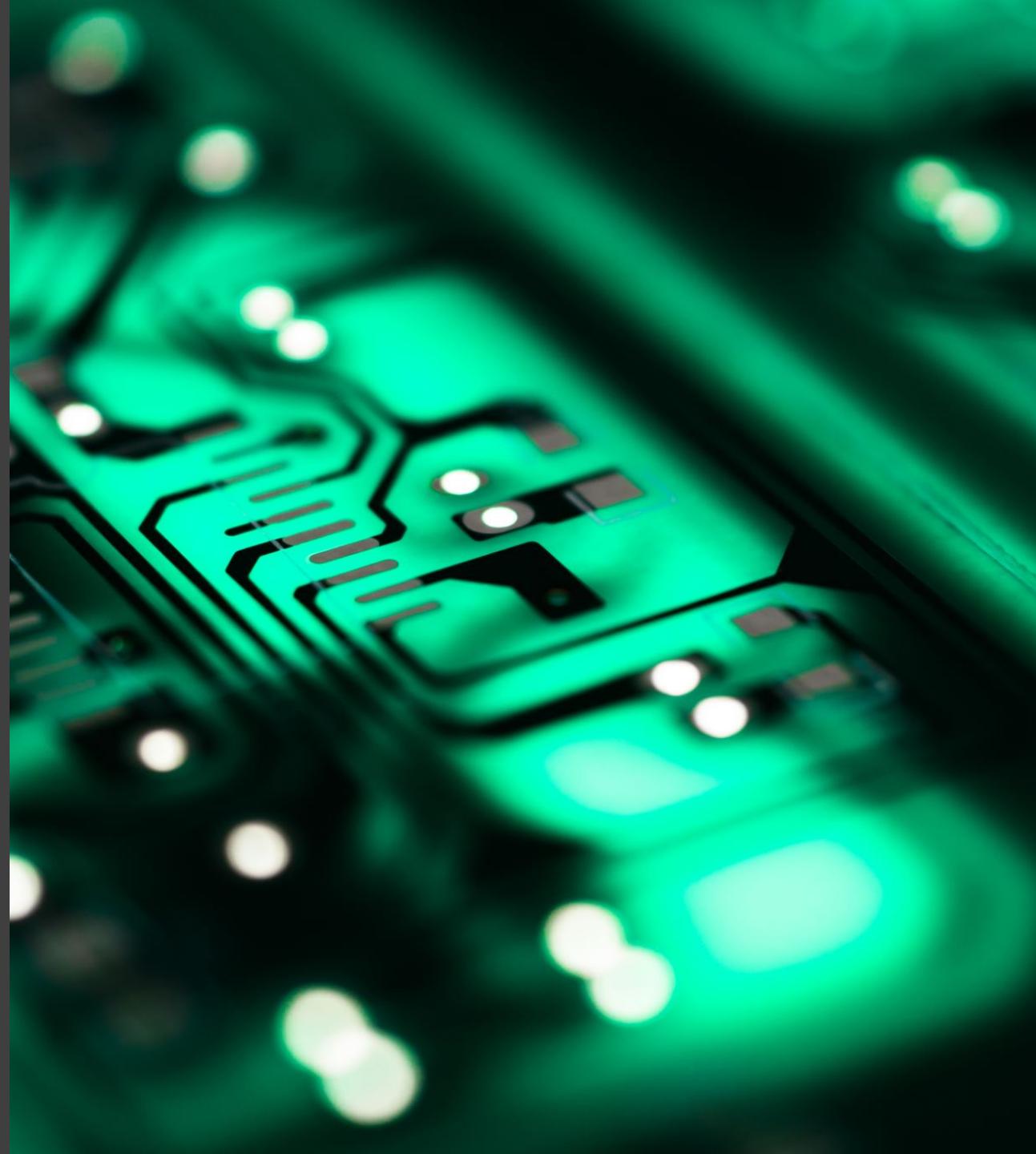


# Effective data visualization using python

Serghei Mangul, Ph.D

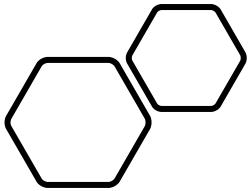
Assistant Professor of Clinical Pharmacy and Biological Sciences,  
University of Southern California



# Learning objectives

- By the end of this module you should be able to:
  - Visually displayed data in real time
  - Understand the strengths and weaknesses of different visualizations techniques
  - Create various type of plots
  - Use seaborn and matplotlib to make elements of plots easier to read and understand

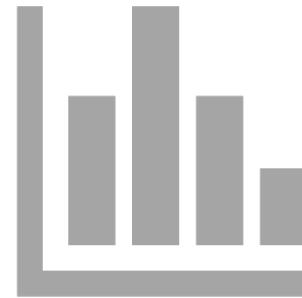




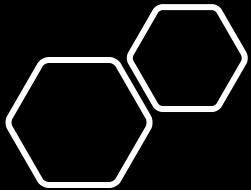
# Why is data visualization a critical skill?



We are drowning in information, but  
starving for knowledge (John Naisbitt)



Creating data visualizations requires  
more than computational skills



# Why are figures important?

Visualizations stand out

- First things you notice

A picture tells a thousand words

- Cliché but true

They are more spreadable

- An image can go viral, a passage likely won't

They are multi-purpose

- Can be used in papers, posters, talks, etc.

# What is the goal of data visualization?



Make your result clear

The main point of the visualization is, ideally, gathered at first glance



Retain focus

If a figure is too busy, the impact can be lost



Don't fool your audience

Visualization should show what is in the data



Be transparent

Make sure labels and titles are descriptive and not ambiguous

# More on the use of figures

- Object of this module is to learn how to make visuals
- If you would like to explore more about how to effectively use visuals
  - <https://cs.stanford.edu/~marinka/slides/marinka-figures19.pdf>
  - These slides by Marinka Zitnik have some very useful information

# GUI-based visualization\*

## Pros

- Fast and easy to use
- Many powerful adjustments and calculations can be made without underlying knowledge

## Cons

- Many powerful adjustments and calculations can be made without underlying knowledge
- Not reproducible, all adjustments are made by hand

# Code-based visualization

## Pros

- Easy to share( more on this in later modules)
- Can be combined with data manipulation and cleaning
- Very transparent and reproducible
- Scalable

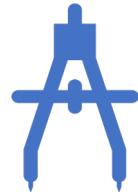
## Cons

- Higher barrier of entry
- Less intuitive



# Python Libraries for visualization

---



## Matplotlib

Includes the basic functions and objects for creating plots

Many of the functions relate to the axis and figure objects

Has built-in visualization functions. These can be used, but seaborn tends to look nicer (a subjective measure)



## Seaborn

Builds on top of the matplotlib objects

Includes high level interfaces for creating visualizations

# Seaborn example datasets

- Seaborn has example datasets available online
- These can be accessed with

```
seaborn.load_dataset(<dataset_name>)
```

# Import seaborn

```
import seaborn as sns
```

# Let's practise

```
categorical_data = sns.load_dataset('tips')
```

We will use the “tips” dataset for  
the first examples in these slides

```
categorical_data = sns.load_dataset('tips')  
categorical_data.head()
```

	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

# Seaborn Visualizations

## Categorical Plots

- Bar plots
- Box plots
- Strip plots
- Violin plots
- Swarm plots
- Boxen plots
- Point plots
- Count plots

## Relational plots

- Line plots
- Scatter plots

## Matrix Plots

- Heatmaps
- Clustermaps

# Categorical plots

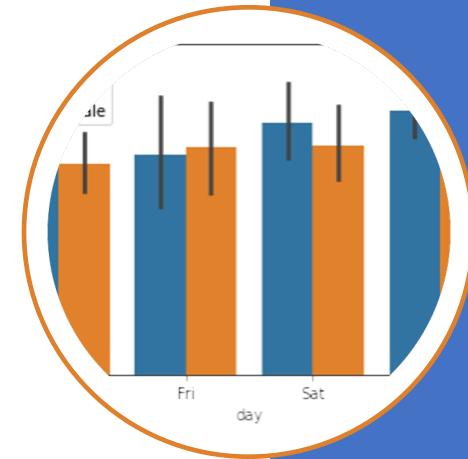
---



- Plots where we have a categorical variable (e.g. color, time, tissue site) and a numerical variable or measurement
  - Typically aim to compare one measurement across many categories
- Many different ways to plot

# Bar plots

- Useful for displaying percentages
  - Preferred over pie charts
- Can also be used to compare means or medians
- Does not convey distribution nor sample size

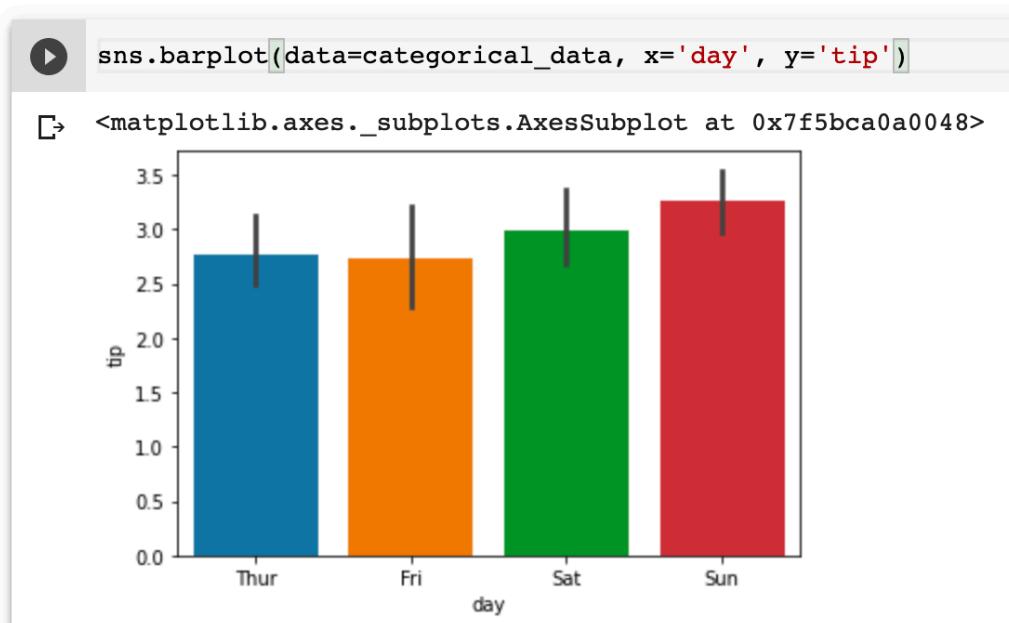


# Draw bar plots grouped by a categorical variable

```
sns.barplot(data=categorical_data, x='day', y='tip')
```

Name of the data frame

Column names

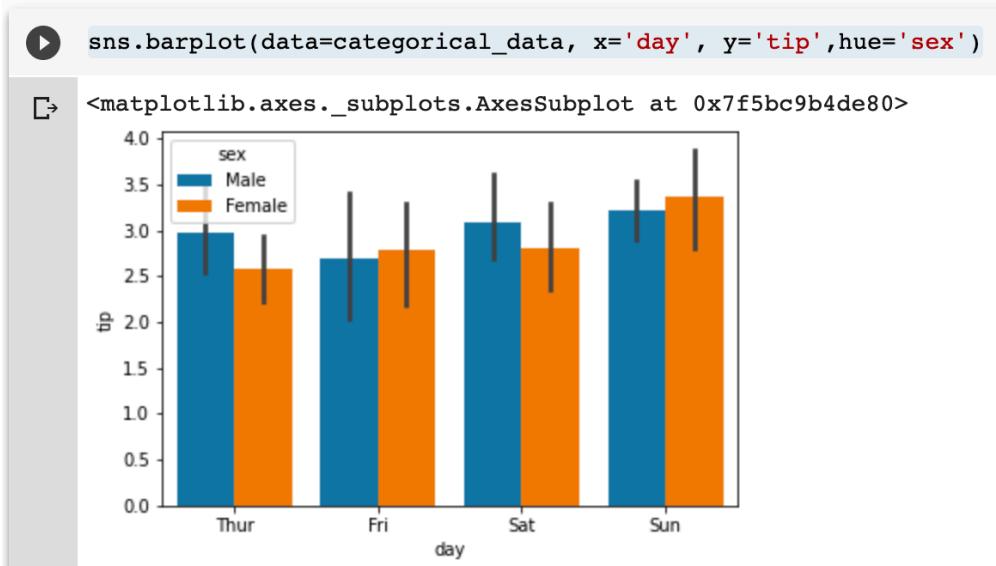


# Bar plot with nested grouping by a two variable

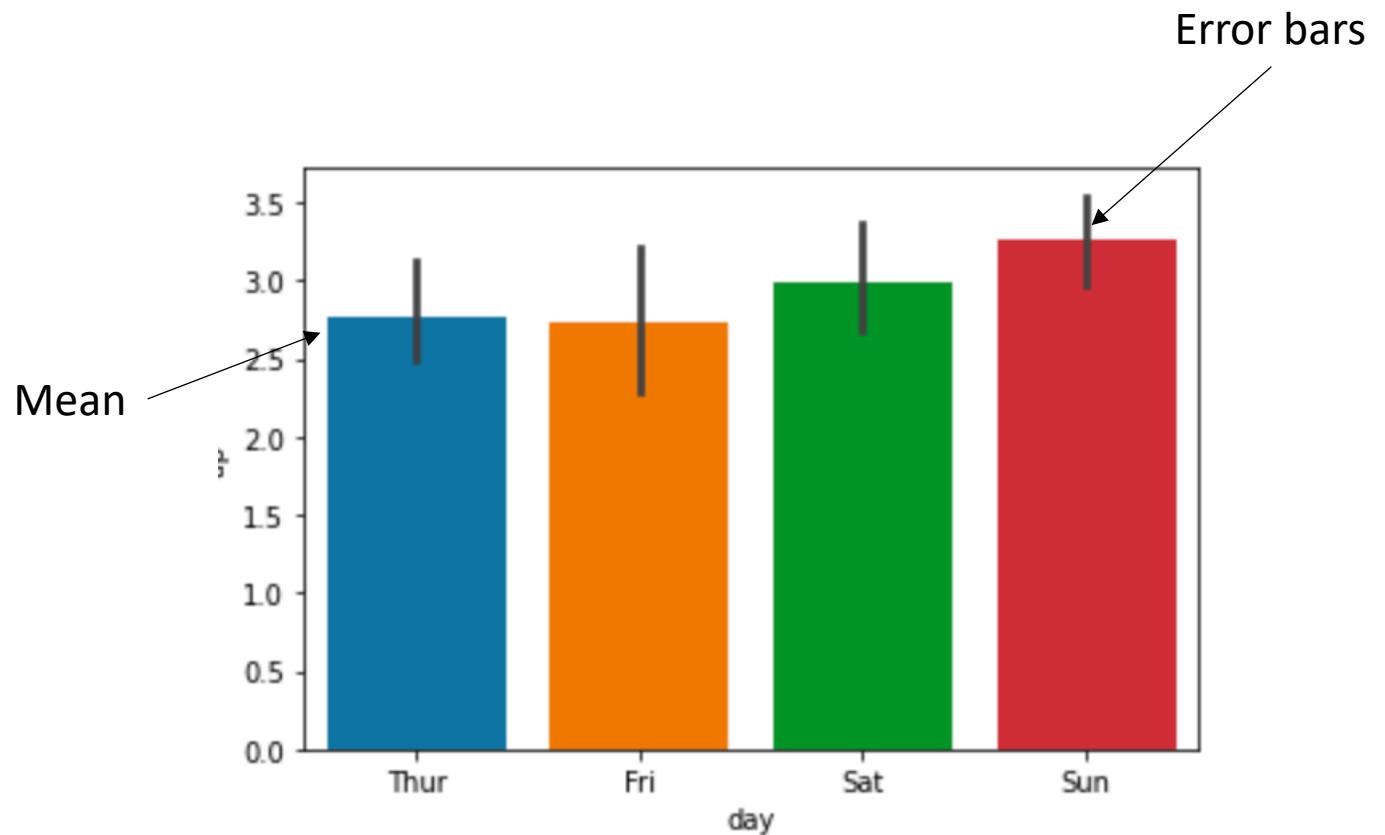
```
sns.barplot(data=categorical_data, x='day', y='tip', hue='sex')
```



Split each bar into two  
based on this variable



# Parameters of bar plot



# Median instead of mean

```
sns.barplot(x='day', y='tip', data=categorical_data, estimator=np.median)
```

NumPy library

Median

# NumPy

```
import numpy as np
```

library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

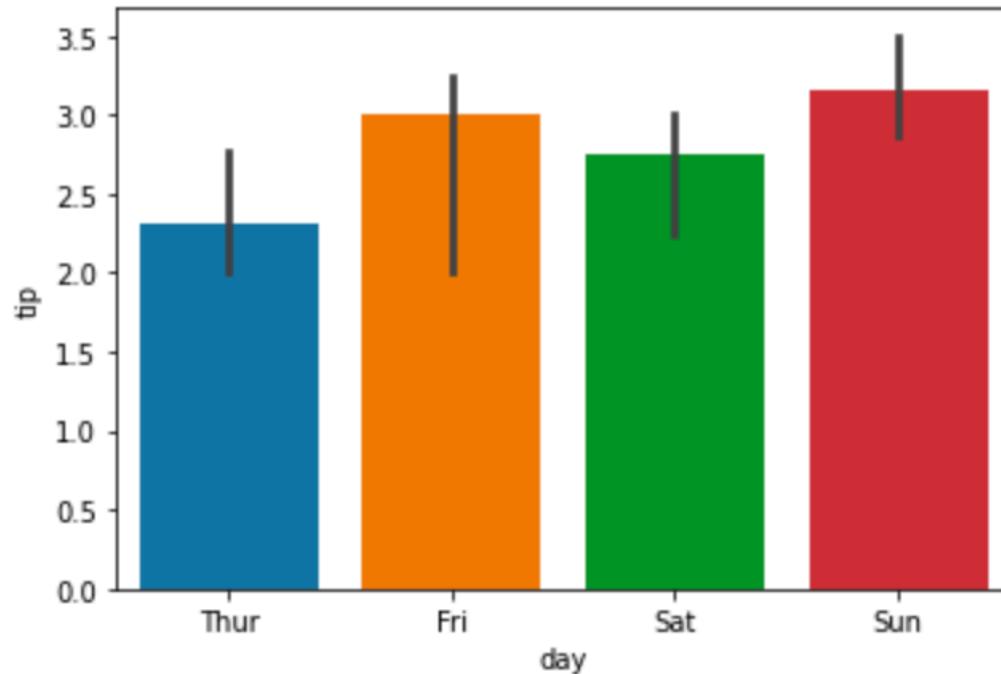
# Let's practice



```
import numpy as np  
sns.barplot(x='day', y='tip', data=categorical_data, estimator=np.median)
```



```
<matplotlib.axes._subplots.AxesSubplot at 0x7f06ca11dd68>
```

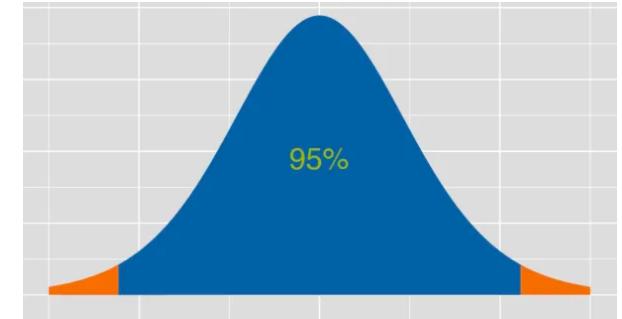


# Error bars

- Default confidence interval for error bars is 95%
  - Can be changed with ci parameter
  - This is calculated using bootstrapping
  - Bootstrapping resamples from the sample with replacement 1000 times to generate the variance of the estimator

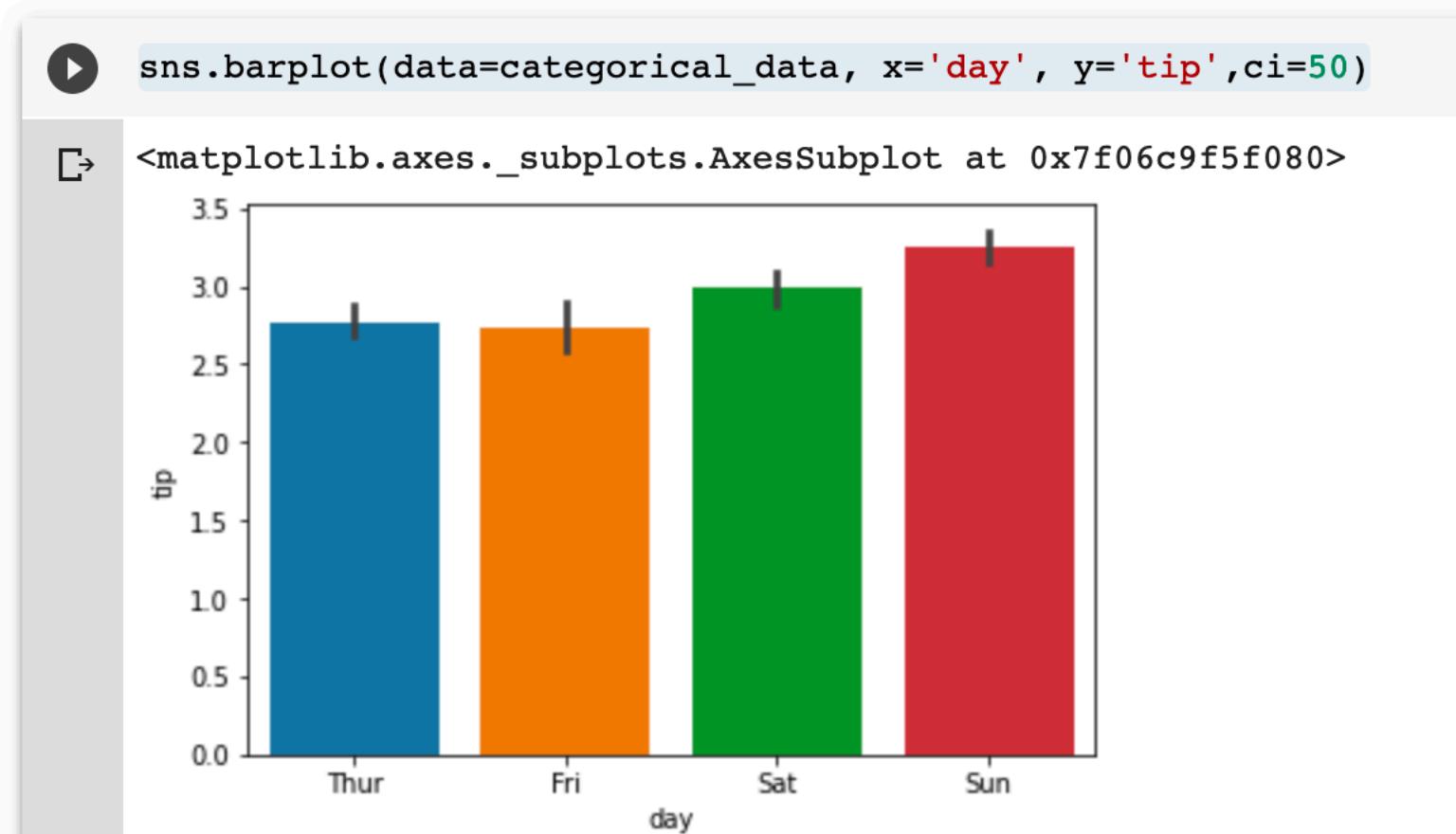
# Change confidence interval

```
sns.barplot(data=categorical_data, x='day', y='tip', ci=50)
```



Default confidence interval for error bars is 95%

# Let's practice

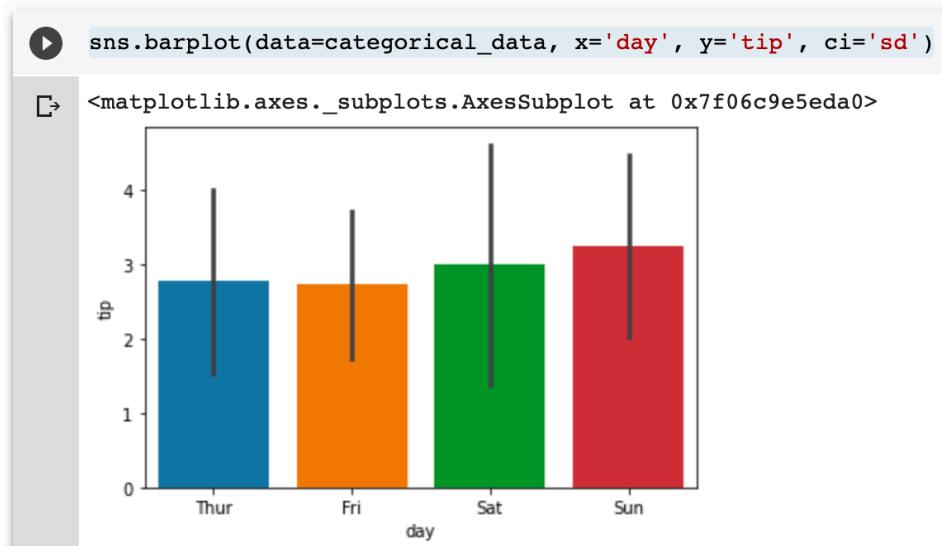


# Change confidence interval

```
sns.barplot(data=categorical_data, x='day', y='tip', ci='sd')
```



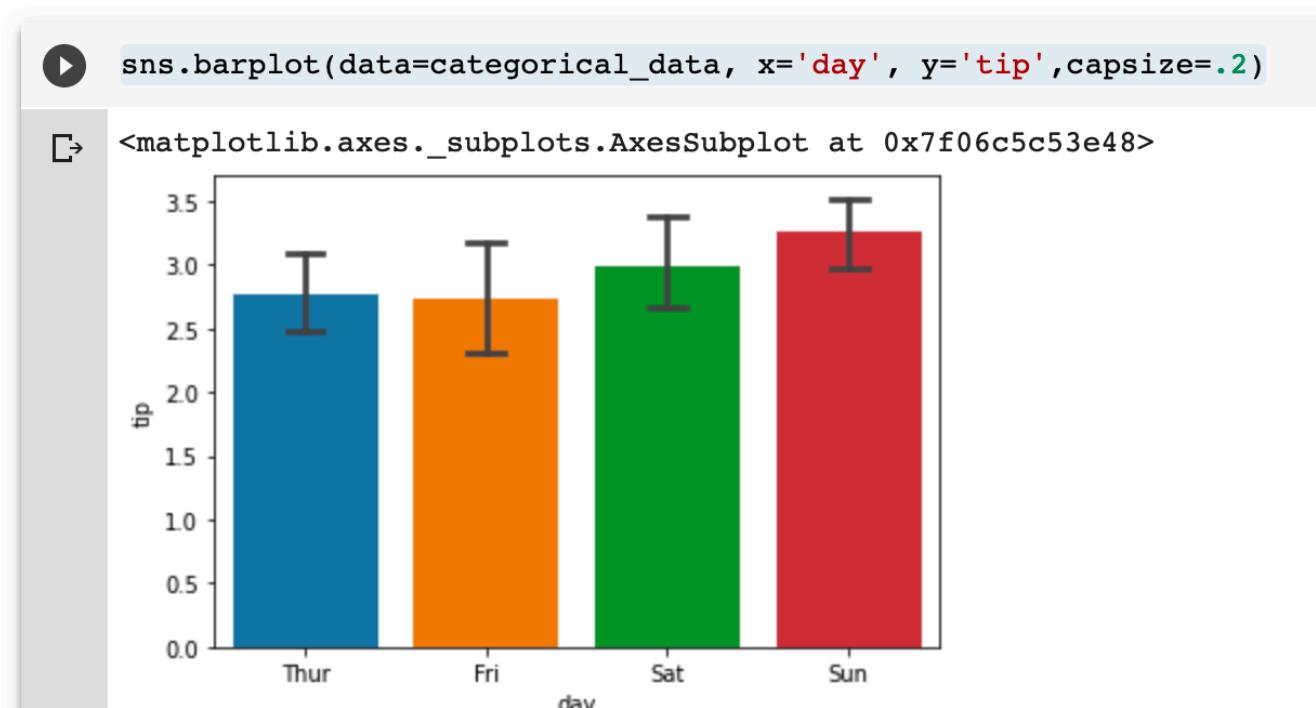
changes this to just use standard deviation



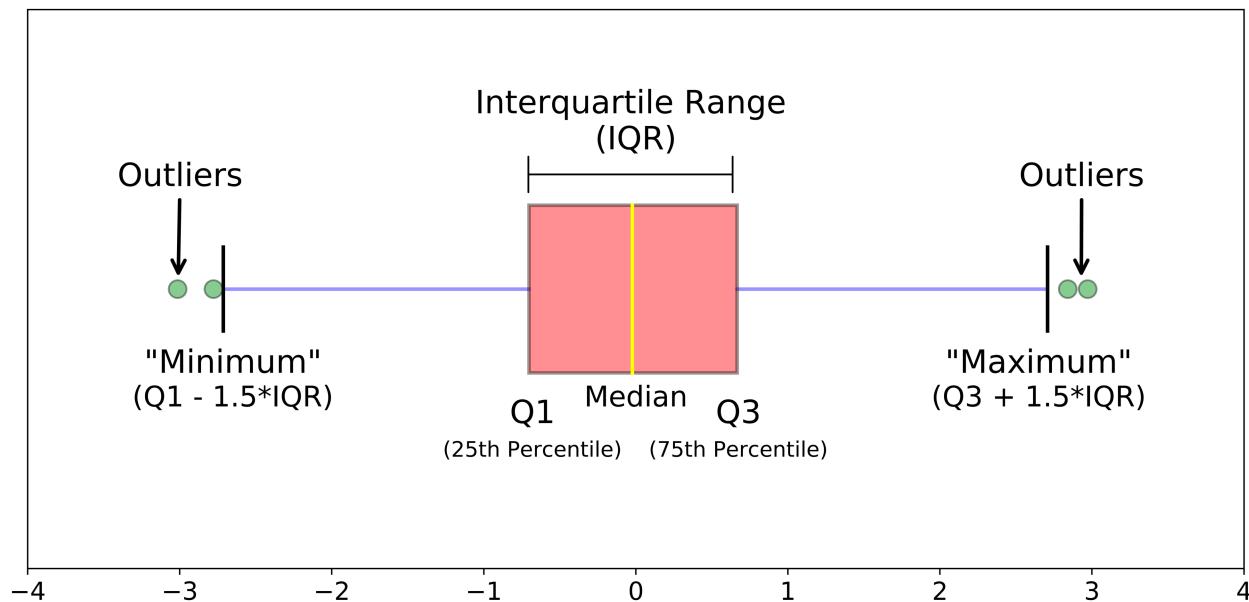
# More options for bar plot

- We can add “caps” on error bars of certain size

```
sns.barplot(data=categorical_data, x='day', y='tip', capsize=.2)
```



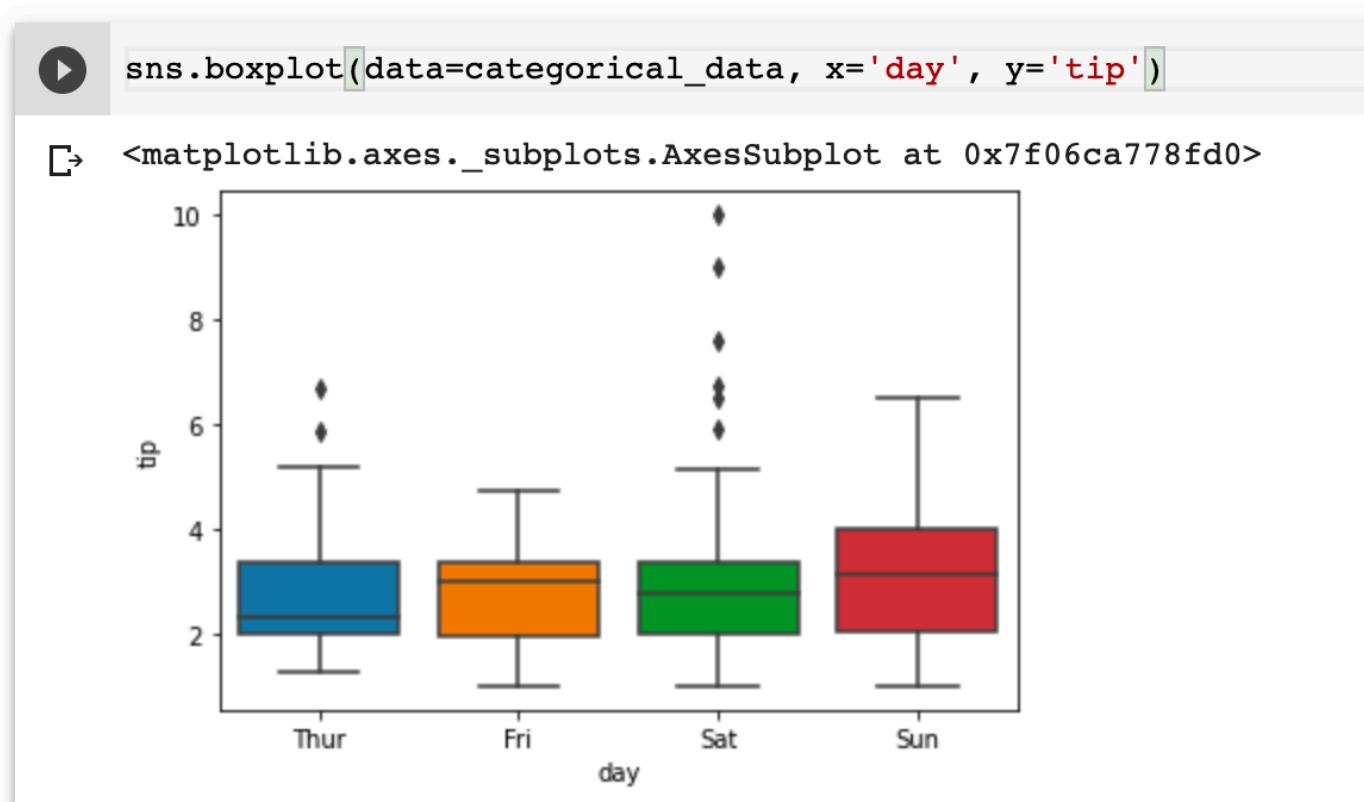
# Box plots



- Sometimes referred as whisker plot
- Useful for displaying quantiles
- Conveys distribution information
- Able to displays outliers
- Does not convey sample size

# Let's practice

```
sns.boxplot(data=categorical_data, x='day', y='tip')
```



# Outliers



Any point outside of the whisker range will be labelled as an outlier



The size of these points can be changed with the fliersize parameter



Setting it to 0 will effectively remove these points

# Let's practice

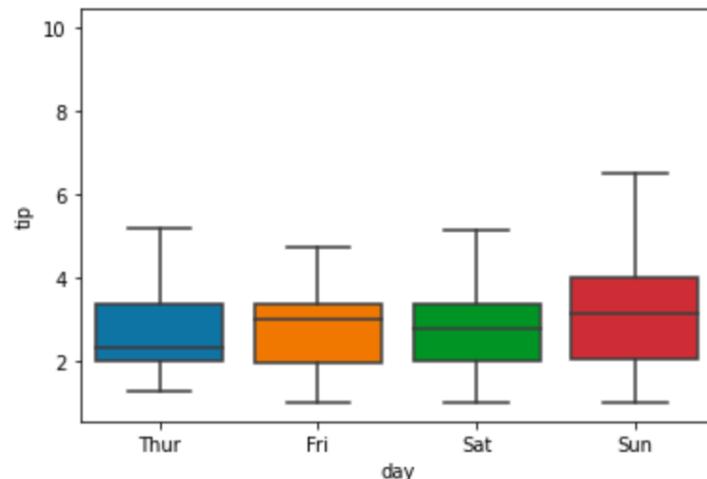
```
sns.boxplot(data=categorical_data, x='day', y='tip', fliersize=0)
```



Setting it to 0 will effectively remove these points

```
sns.boxplot(data=categorical_data, x='day', y='tip', fliersize=0)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f06c9dcdb00>
```

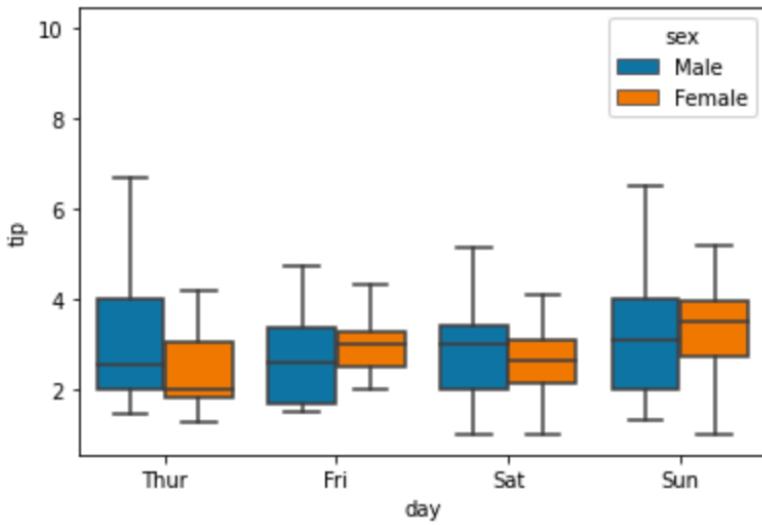


# Box plot with nested grouping by a two variable

```
sns.boxplot(data=categorical_data, x='day', y='tip', fliersize=0, hue='sex')
```

```
▶ sns.boxplot(data=categorical_data, x='day', y='tip', fliersize=0, hue='sex')
```

```
↪ <matplotlib.axes._subplots.AxesSubplot at 0x7f06c9aa6630>
```



Split each box plot into two based on this variable



# Clearly express distributions

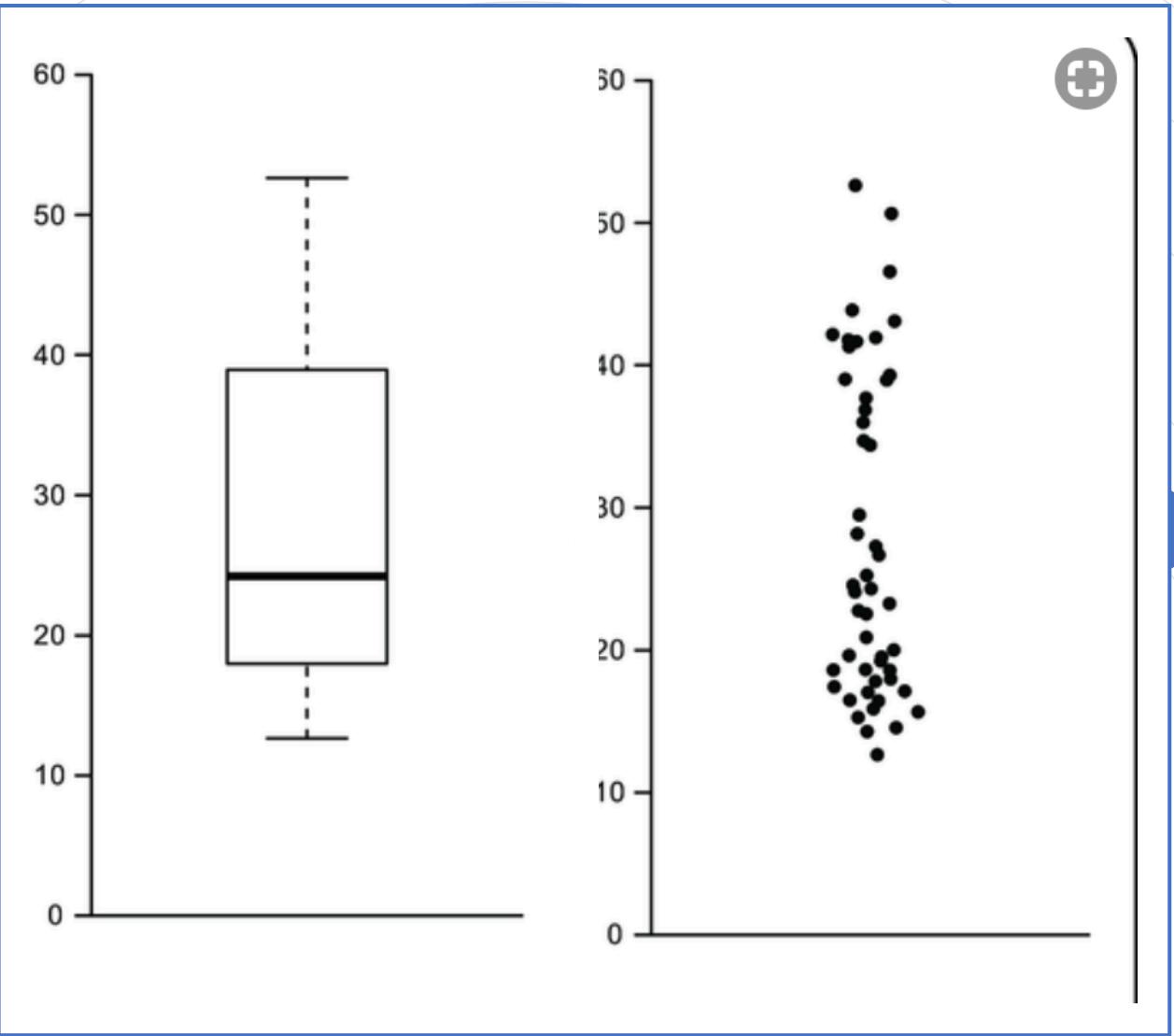


Visualizations can be misleading

Mean values can be inflated by outliers  
Differences may seem more significant because  
of small samples



Showing the data point may solve those problems



## Multimodal distribution

Box plots cannot clearly describe multimodal distributions

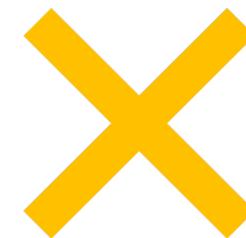
# Strip plots



Often overlaid onto  
another plot



Clearly displays sample size  
and distribution



Does not convey estimators  
like mean or median

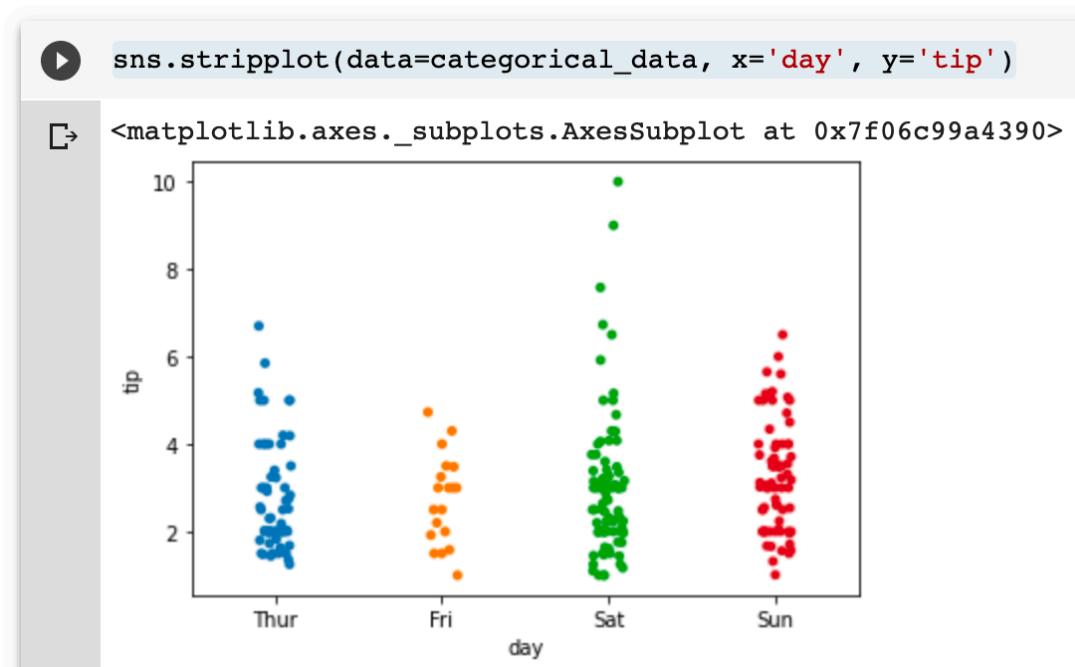
# Strip plots

---

- Running the same function multiple times will produce slightly different plots
- The points have random horizontal spread

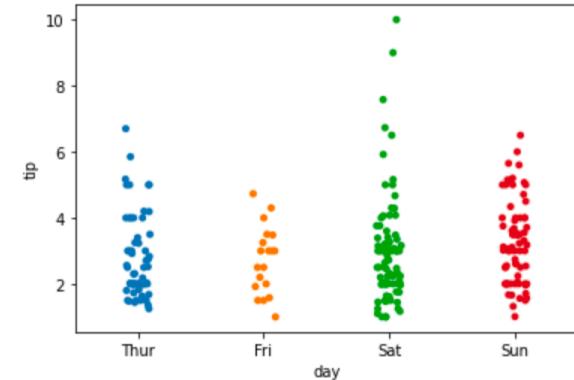
# Let's practice

```
sns.stripplot(data=categorical_data, x='day', y='tip')
```

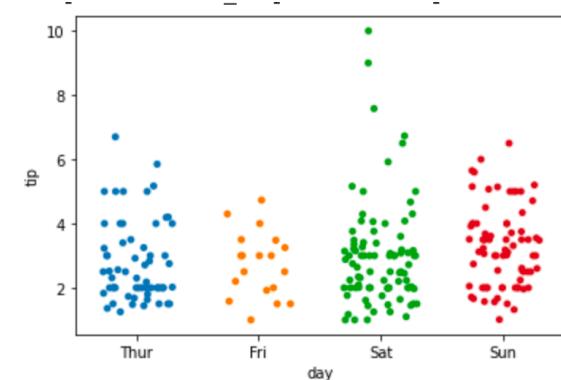


# Jitter parameter

- If you have many similar points, there is a high chance that they will overlap
- The jitter parameter can change the horizontal spread of the points



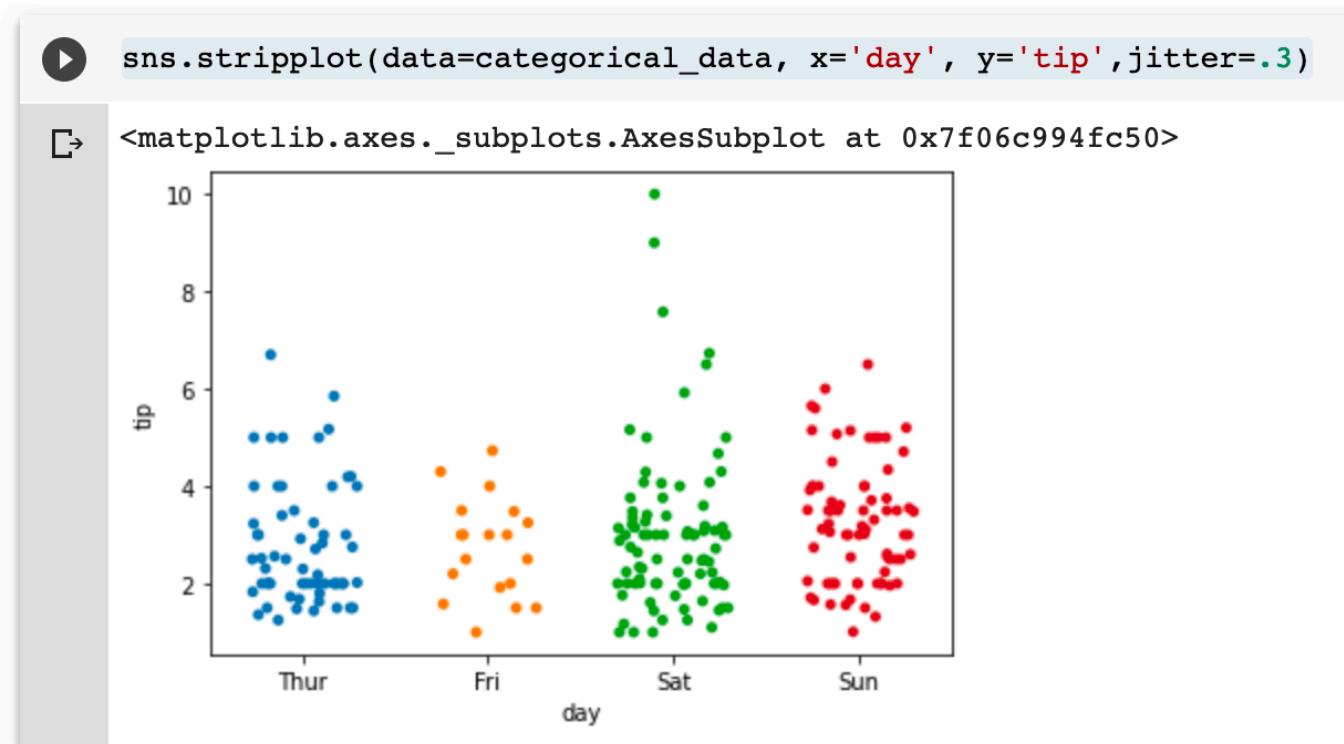
default



Jitter = 0.3

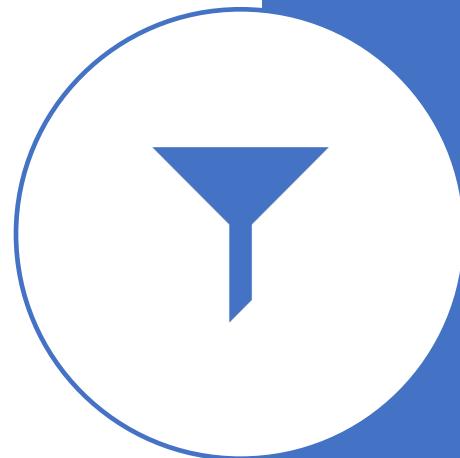
# Let's practice

```
sns.stripplot(data=categorical_data, x='day', y='tip', jitter=.3)
```



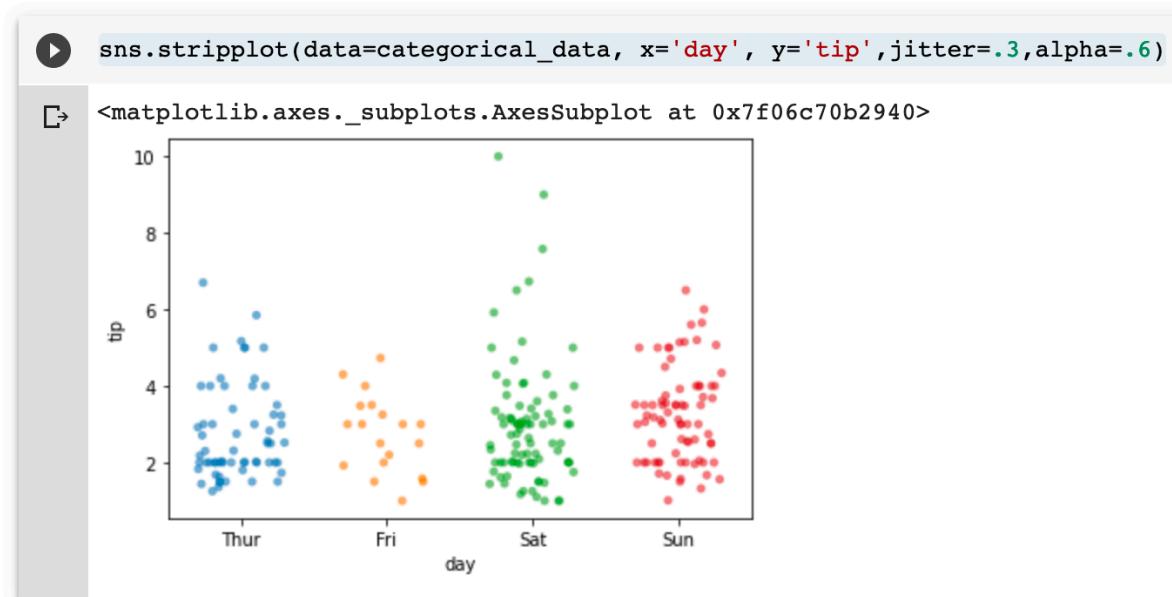
# Overlapping points in strip plots

- The alpha parameter can also help to see the overlapped points better
- Alpha controls transparency, and can be used on other plots as well



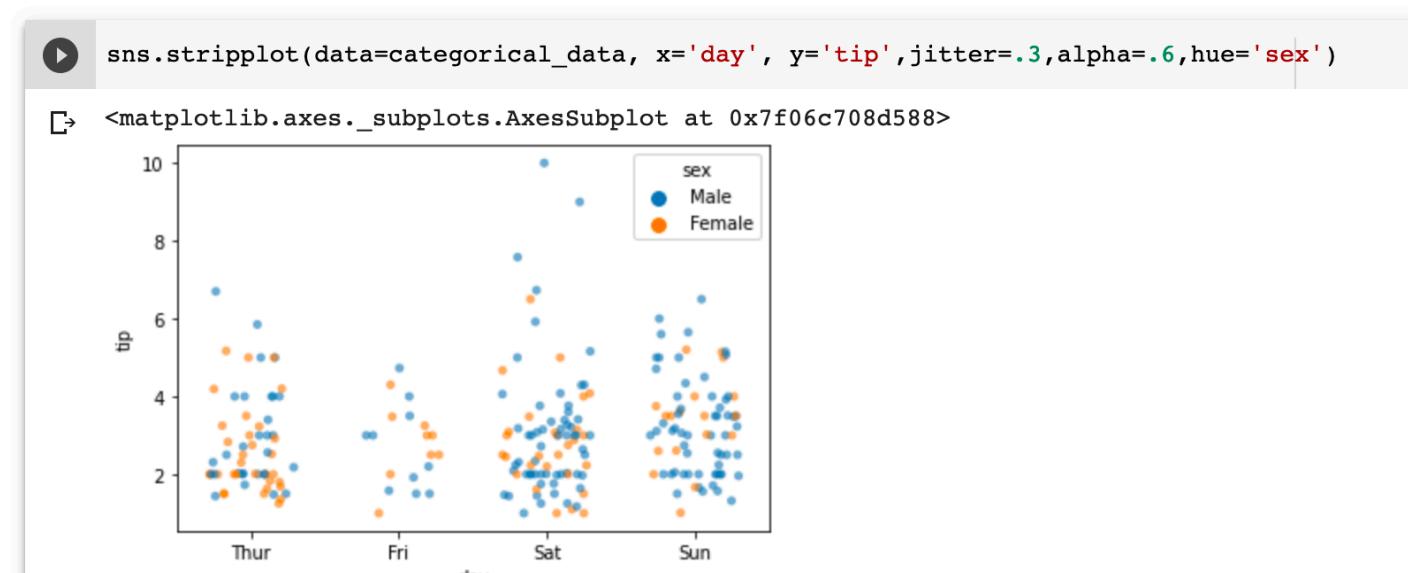
# Let's practice

```
sns.stripplot(data=categorical_data, x='day', y='tip', jitter=.3, alpha=.6)
```



# Nested grouping by a two variable

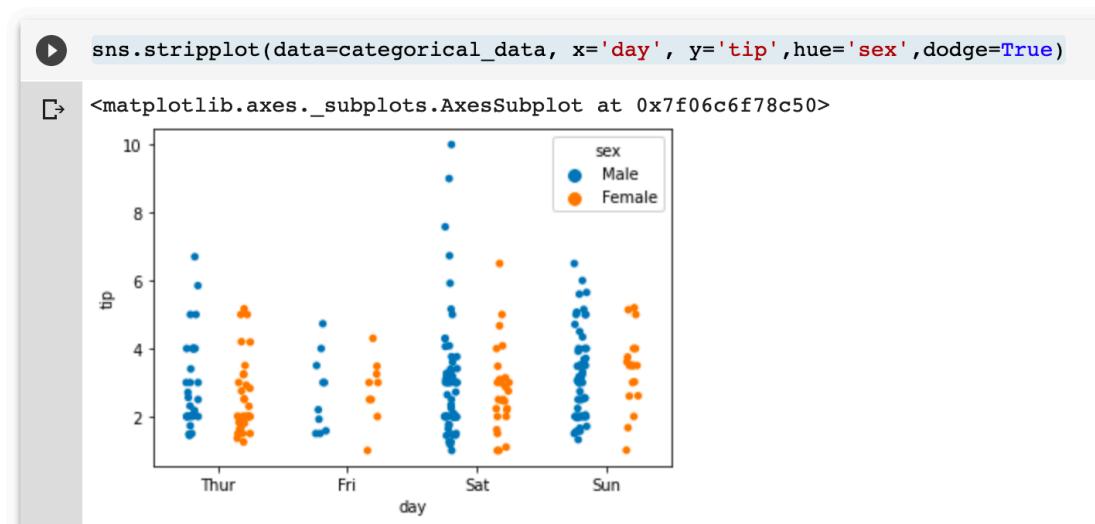
```
sns.stripplot(data=categorical_data, x='day', y='tip', jitter=.3, alpha=.6, hue='sex')
```



# Dodge option

- By default, points separated by hue will overlap
- Using dodge parameter data point will be separated

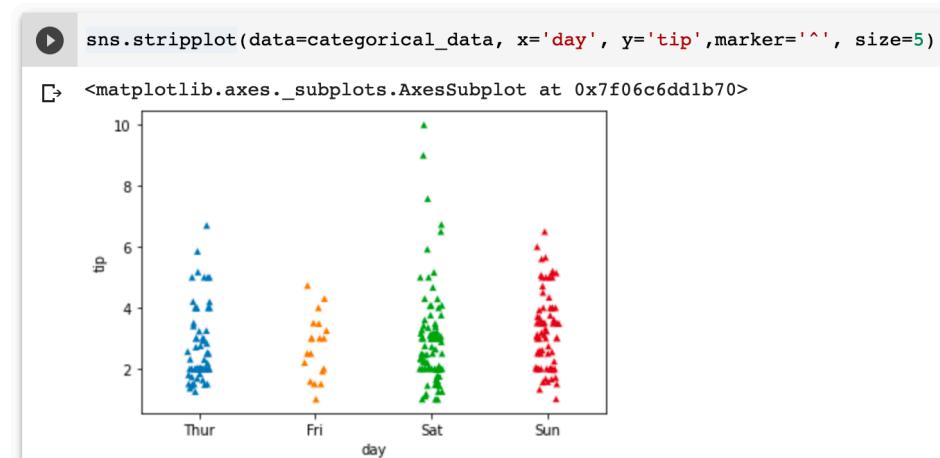
```
sns.stripplot(data=categorical_data, x='day', y='tip', hue='sex', dodge=True)
```



# More options for strip plots

- The size and shape of every point is also editable
- There are plenty of built-in markers to use

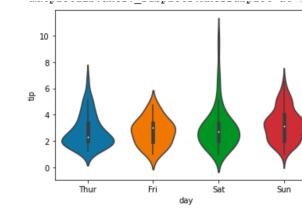
```
sns.stripplot(data=categorical_data, x='day', y='tip', marker='^', size=5)
```



Read more at <https://seaborn.pydata.org/generated/seaborn.stripplot.html>

# Violin plots

- Uses kernel density estimation to display the underlying distribution
- Displays distribution and estimator
- Estimation is influenced by sample size



## Kernel density estimation

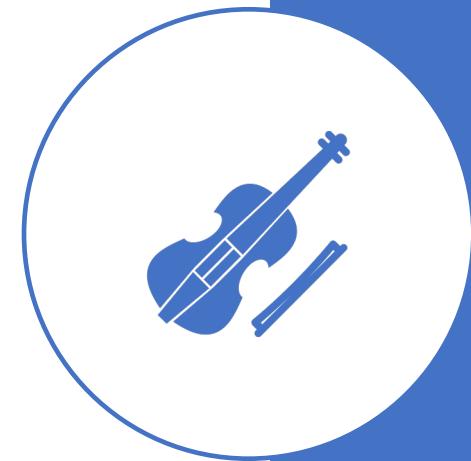
- Useful if you want to visualize just the “shape” of some data, as a kind of continuous replacement for the discrete histogram

Read more here

<https://mathisonian.github.io/kde/>

# Advantages of violin plots

- A violin plot is more informative than a plain box plot
- Violin plot shows the full distribution of the data.

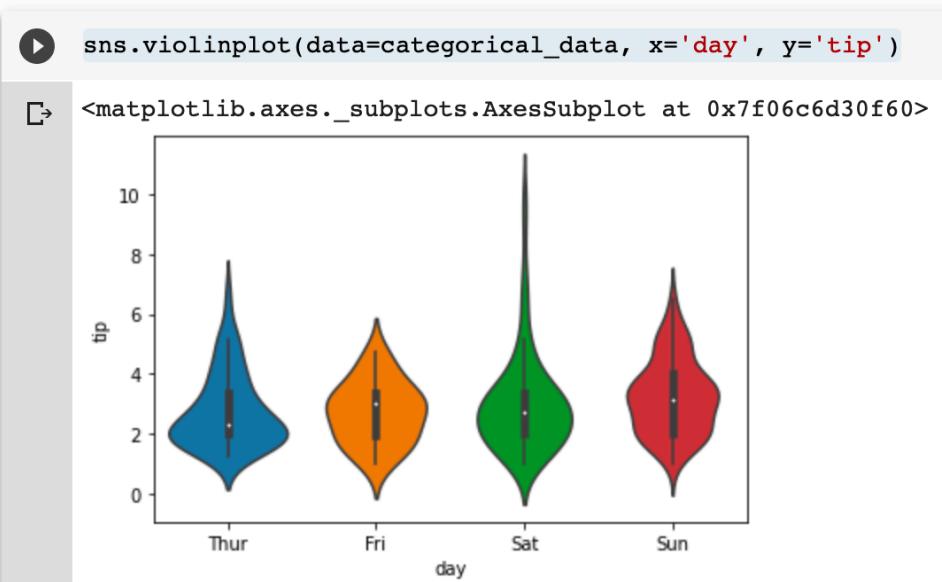


# Let's practice

```
sns.violinplot(data=categorical_data, x='day', y='tip')
```



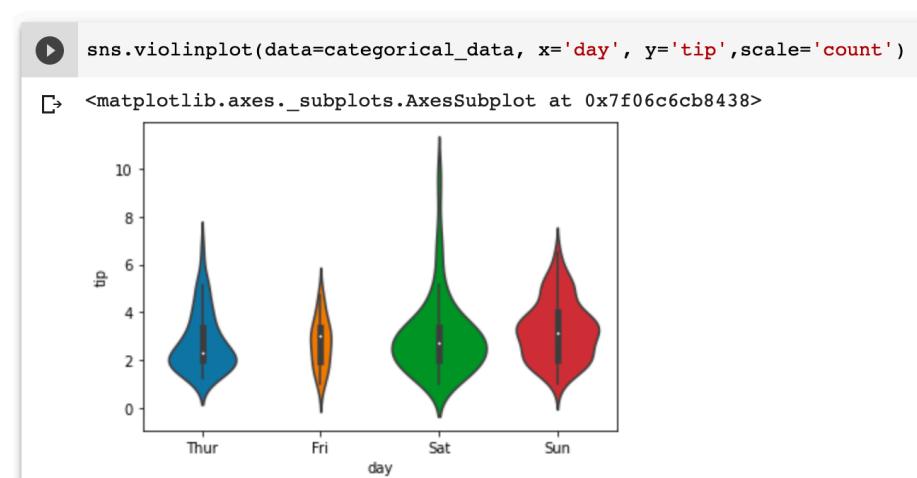
Type of plot



# Relative sample size

- The scale parameter changes effect used for width of each violin
- By default the area of each violin will be constant
- Width is scaled by the number of observations in each violin

```
sns.violinplot(data=categorical_data, x='day', y='tip', scale='count')
```



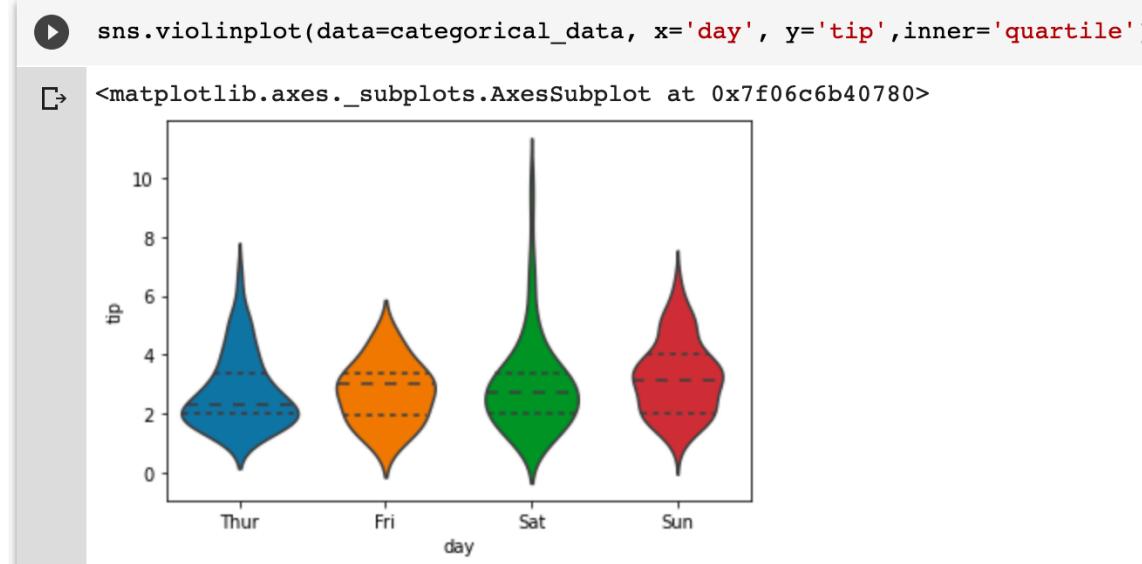
# The inner parameter

- The inner parameter affects what the representation inside the violin
  - **box (default)**: A mini boxplot is drawn
  - **quartile**: Quartile lines are drawn
  - **stick or point**: Individual points drawn as sticks or points
  - **None**: Nothing inside

# Let's practice

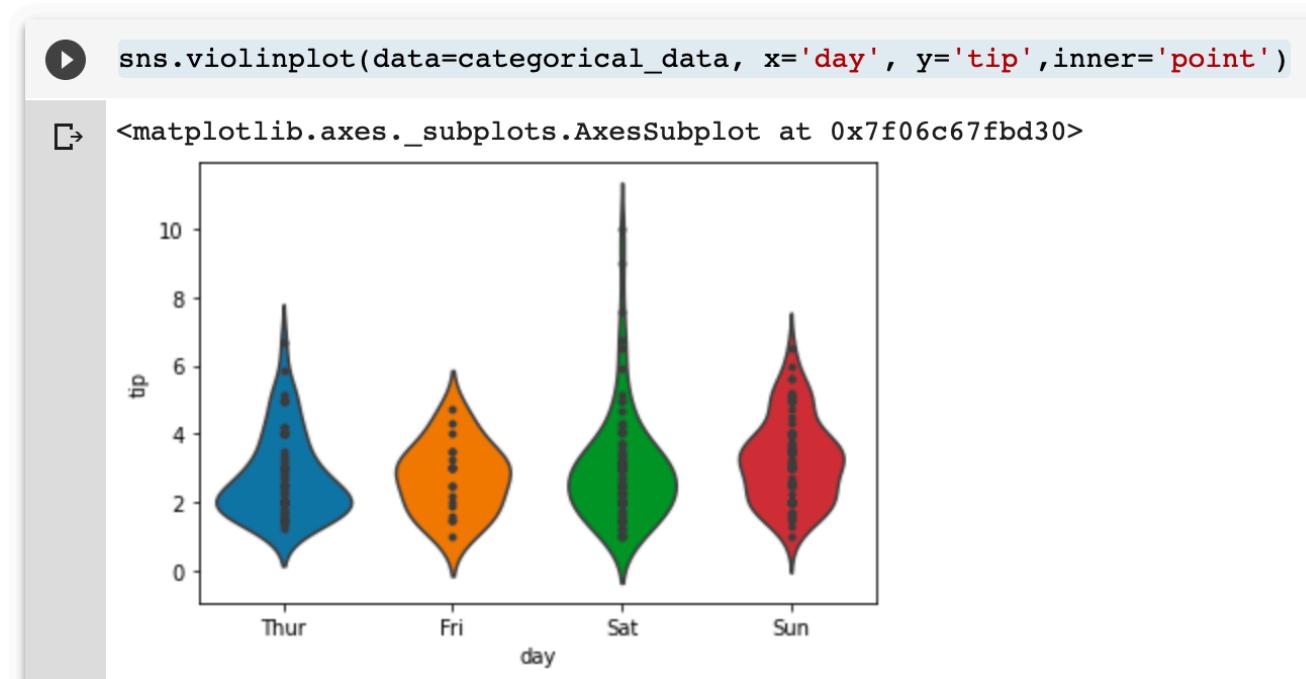
```
sns.violinplot(data=categorical_data, x='day', y='tip', inner='quartile')
```

affects what the representation inside the violin



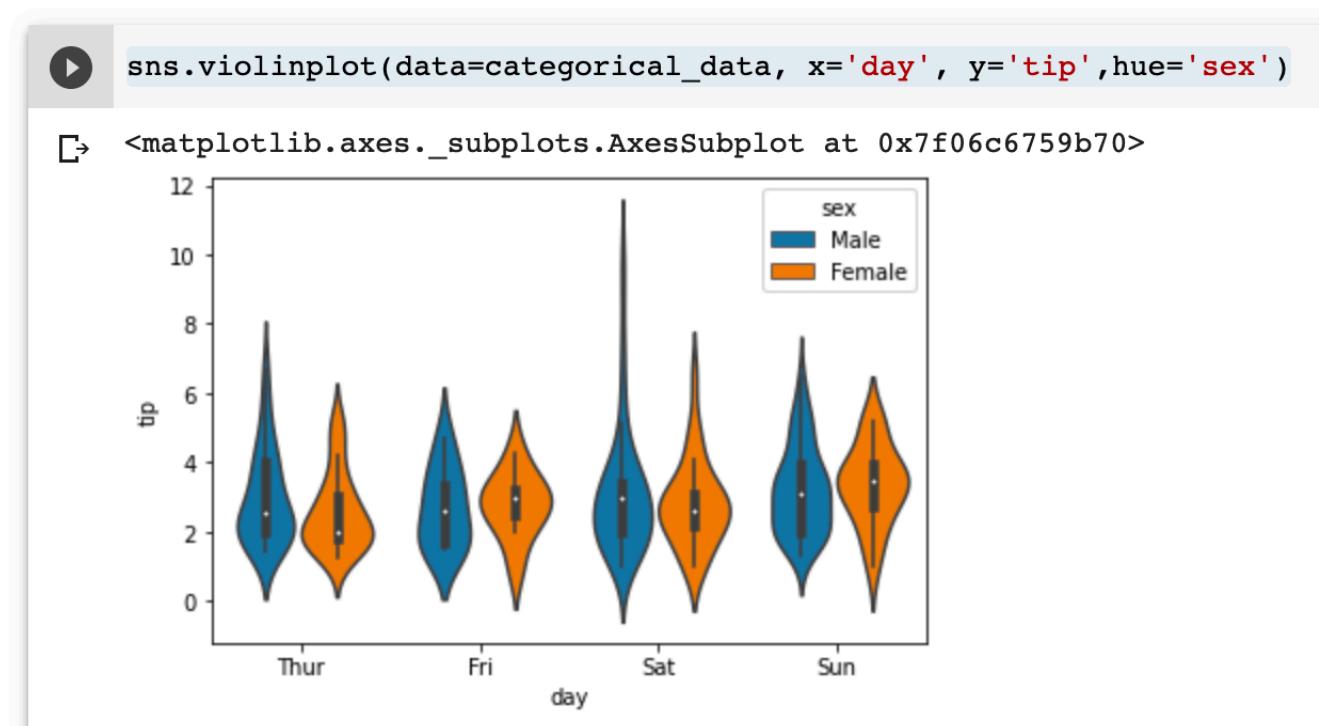
# Let's practice

```
sns.violinplot(data=categorical_data, x='day', y='tip', inner='point')
```



# Violin plot with nested grouping by a two variable

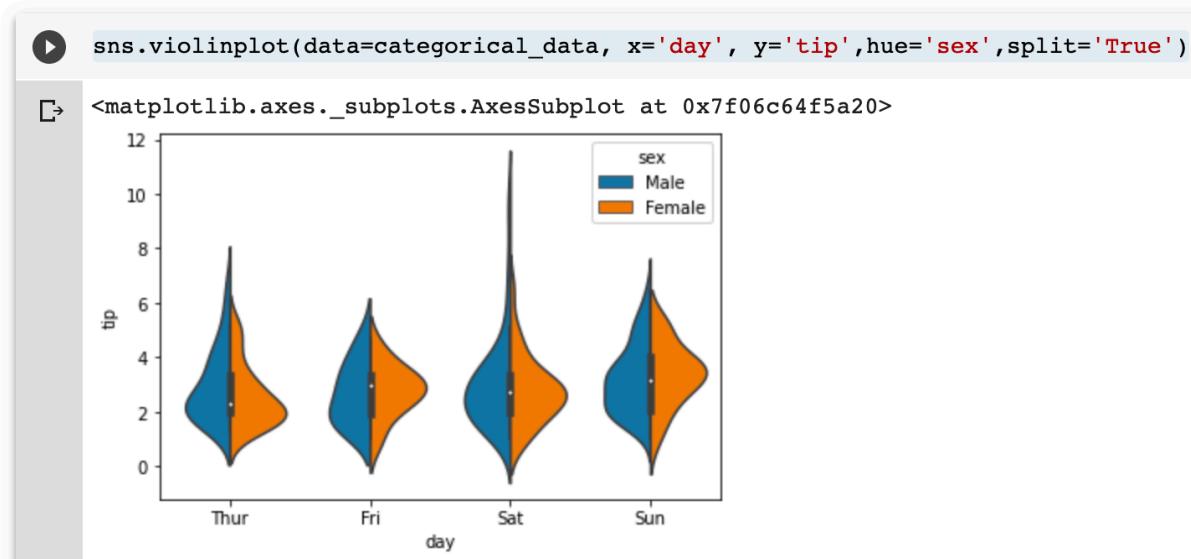
```
sns.violinplot(data=categorical_data, x='day', y='tip',hue='sex')
```



# Split parameter

- Each violin will be split into two halves

```
sns.violinplot(data=categorical_data, x='day', y='tip', hue='sex', split='True')
```



# Ordering plot elements



By default, levels inferred from the data



Explicitly setting orders can help keep consistency



To shorten plot functions, one list can be saved to a variable and called for each plot

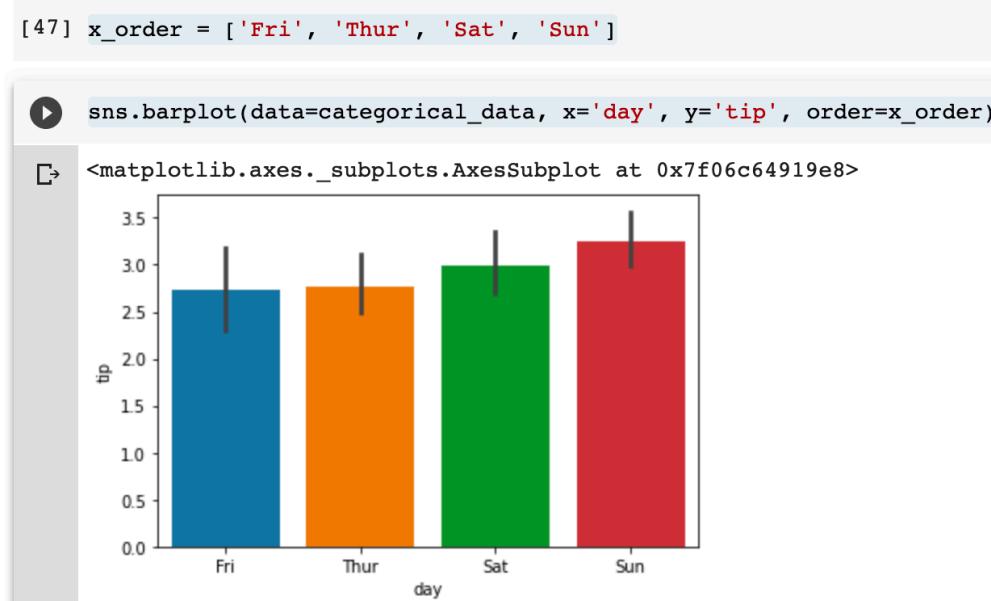
# Let's practice

```
sns.barplot(data=categorical_data, x='day', y='tip', order=['Fri', 'Thur', 'Sat', 'Sun'])
```



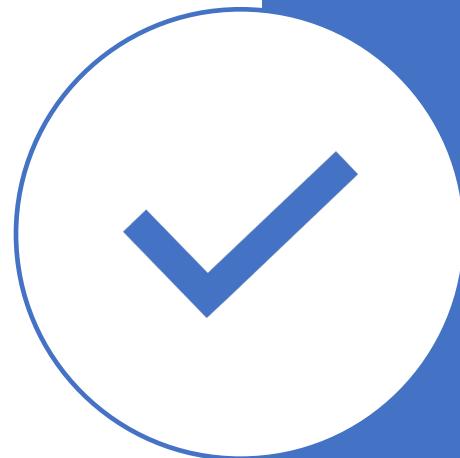
# Ordering plot elements

```
x_order = ['Fri', 'Thur', 'Sat', 'Sun']  
sns.barplot(data=categorical_data, x='day', y='tip', order=x_order)
```



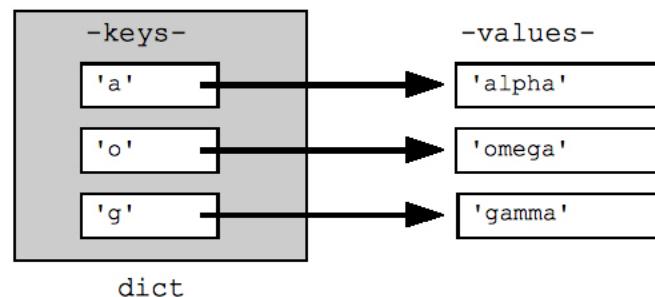
# Define a pallet

- Define the pallet using dictionary data structure
- Only needs to be used once per notebook



# Dictionary data structure

- A **dictionary** is a general-purpose **data structure** for storing a group of objects. A **dictionary** has a set of keys and each key has a single associated value



keys  
values

```
color_dict=dict({'Thur':'purple','Fri':'orange','Sat':'brown','Sun':'gray'})
```

# Let's practice

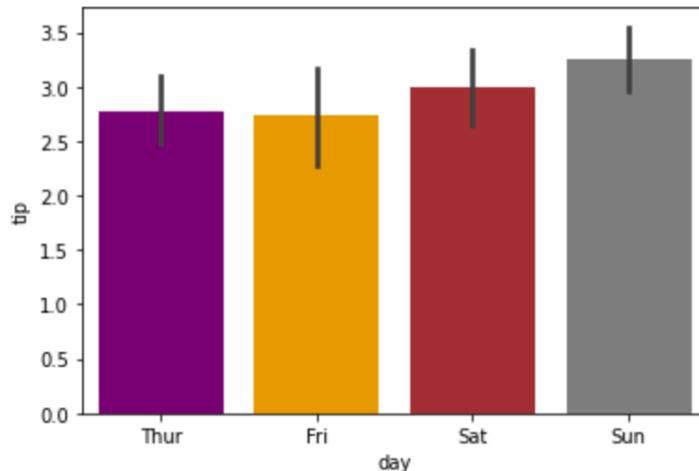
Provide custom palette

```
sns.barplot(data=categorical_data, x='day', y='tip', palette=color_dict)
```

```
[ ] color_dict=dict({'Thur':'purple','Fri':'orange','Sat':'brown','Sun':'gray'})
```

```
▶ sns.barplot(data=categorical_data, x='day', y='tip', palette=color_dict)
```

```
↪ <matplotlib.axes._subplots.AxesSubplot at 0x7f7455b75d30>
```



# Ordering plot elements

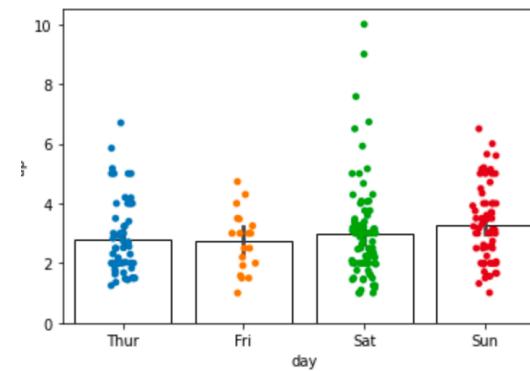
- The `hue_order` parameter affects the hue categories

```
sns.barplot(data=categorical_data, x='sex', y='tip', hue_order=x_order,hue='day')
```



# Combining categorical plots

Oftentimes it is useful to combine multiple plots on one set of axes to reduce the hidden information



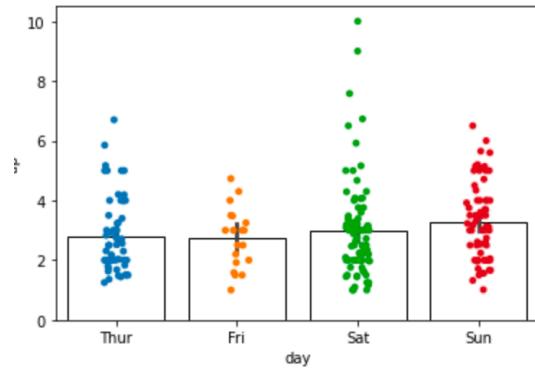
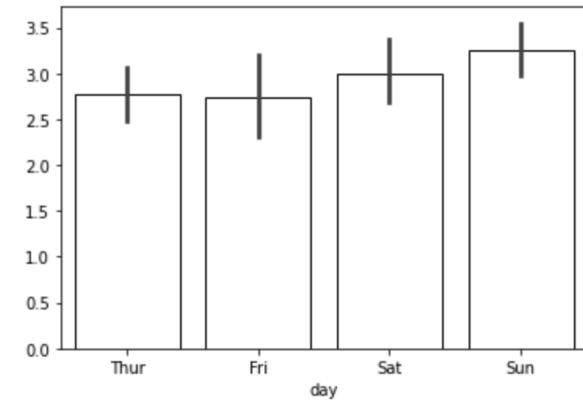
A bar plot and strip plot combine the point distribution with an estimator

# Let's practice

```
sns.barplot(data=categorical_data, x='day',  
y='tip', edgecolor='black', facecolor='white'  
)
```

Color inside the bar

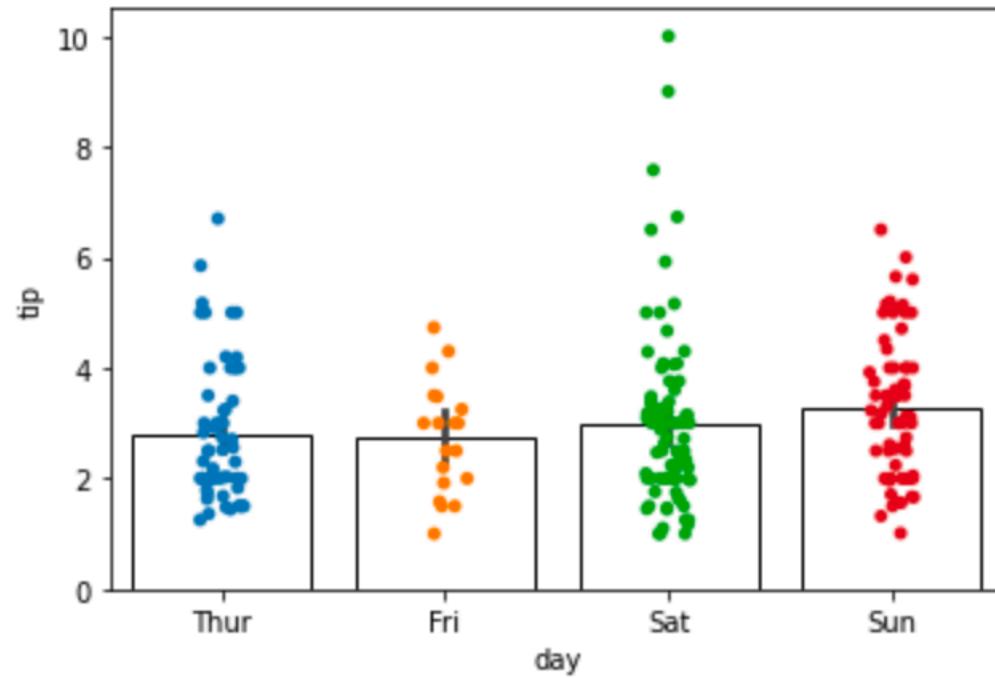
```
sns.stripplot(data=categorical_data,  
x='day', y='tip')
```



# Let's practice

```
sns.barplot(data=categorical_data, x='day', y='tip', edgecolor='black', facecolor='white')  
sns.stripplot(data=categorical_data, x='day', y='tip')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f06c591d5c0>
```



# Improve the aesthetics and clarity

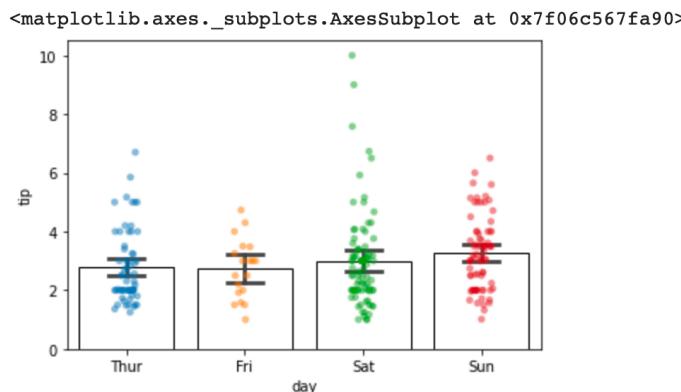
```
sns.barplot(data=categorical_data, x='day',  
y='tip', edgecolor='black', facecolor='white', capsize=.3)
```

Make error bars more visible

```
sns.stripplot(data=categorical_data, x='day', y='tip', alpha=.5)
```

Adjust transparency

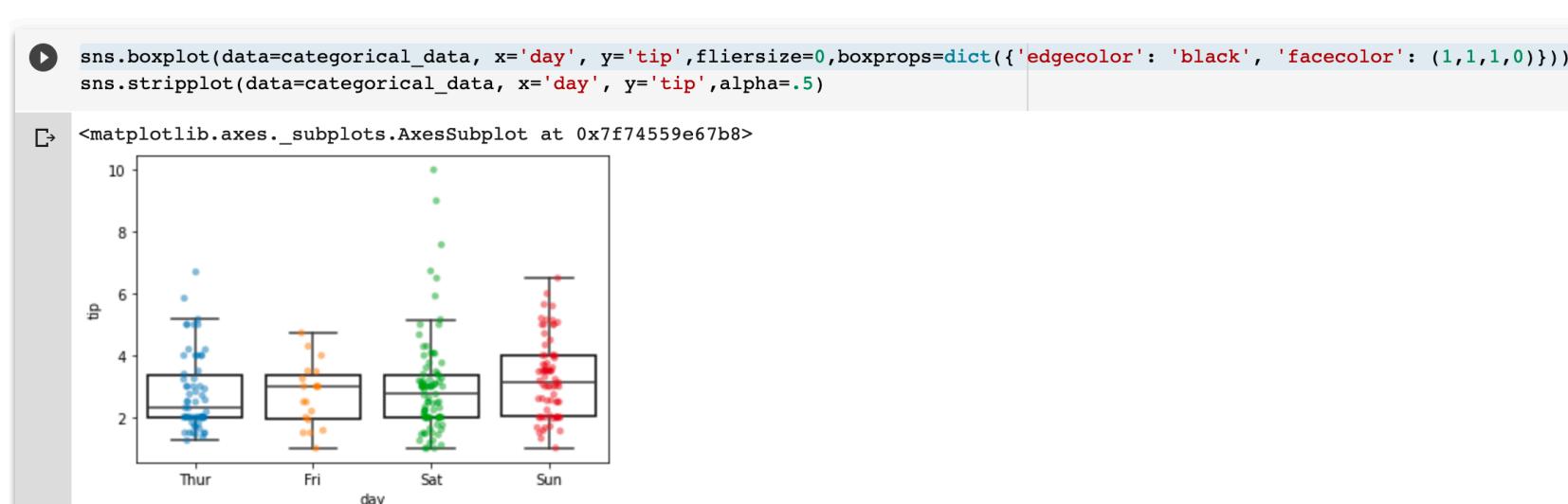
```
▶ sns.barplot(data=categorical_data, x='day', y='tip', edgecolor='black', facecolor='white', capsize=.3)  
sns.stripplot(data=categorical_data, x='day', y='tip', alpha=.5)
```



# Combining categorical plots

```
sns.boxplot(data=categorical_data, x='day', y='tip', fliersize=0, boxprops=dict({'edgecolor': 'black', 'facecolor': (1,1,1,0)}))
```

```
sns.stripplot(data=categorical_data, x='day', y='tip', alpha=.5)
```



Read about boxprops here [https://matplotlib.org/api/\\_as\\_gen/matplotlib.axes.Axes.boxplot.html#matplotlib.axes.Axes.boxplot](https://matplotlib.org/api/_as_gen/matplotlib.axes.Axes.boxplot.html#matplotlib.axes.Axes.boxplot)

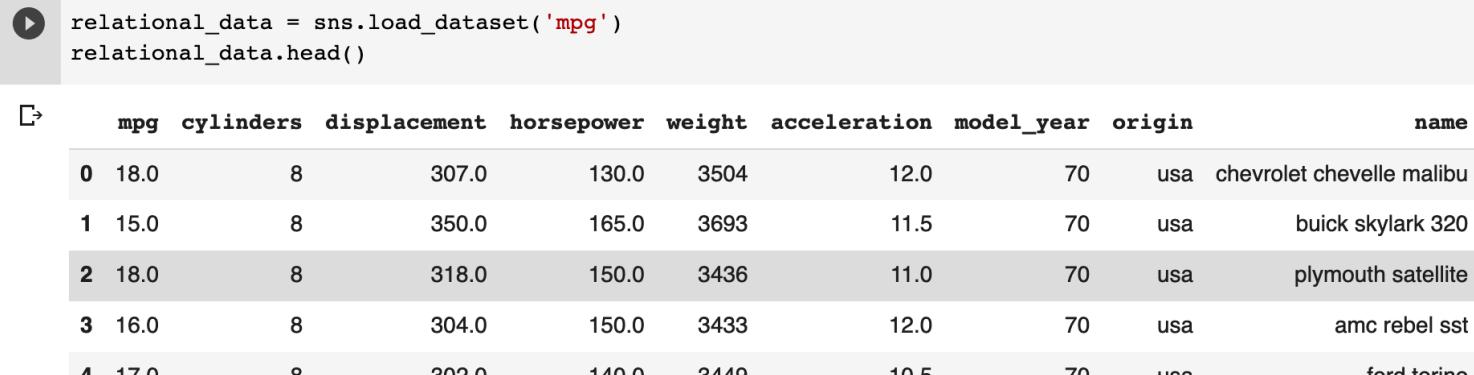
# Relational plots

- Plots where we want to compare the relationship between two variables
- Typically axes are made of two numerical variables

# Test data

- For these examples we will use the “mpg” dataset

```
relational_data = sns.load_dataset('mpg')
```



A screenshot of a Jupyter Notebook cell. The cell contains two parts: a code block and its output. The code block is:

```
relational_data = sns.load_dataset('mpg')
relational_data.head()
```

The output shows the first five rows of the 'mpg' dataset as a pandas DataFrame:

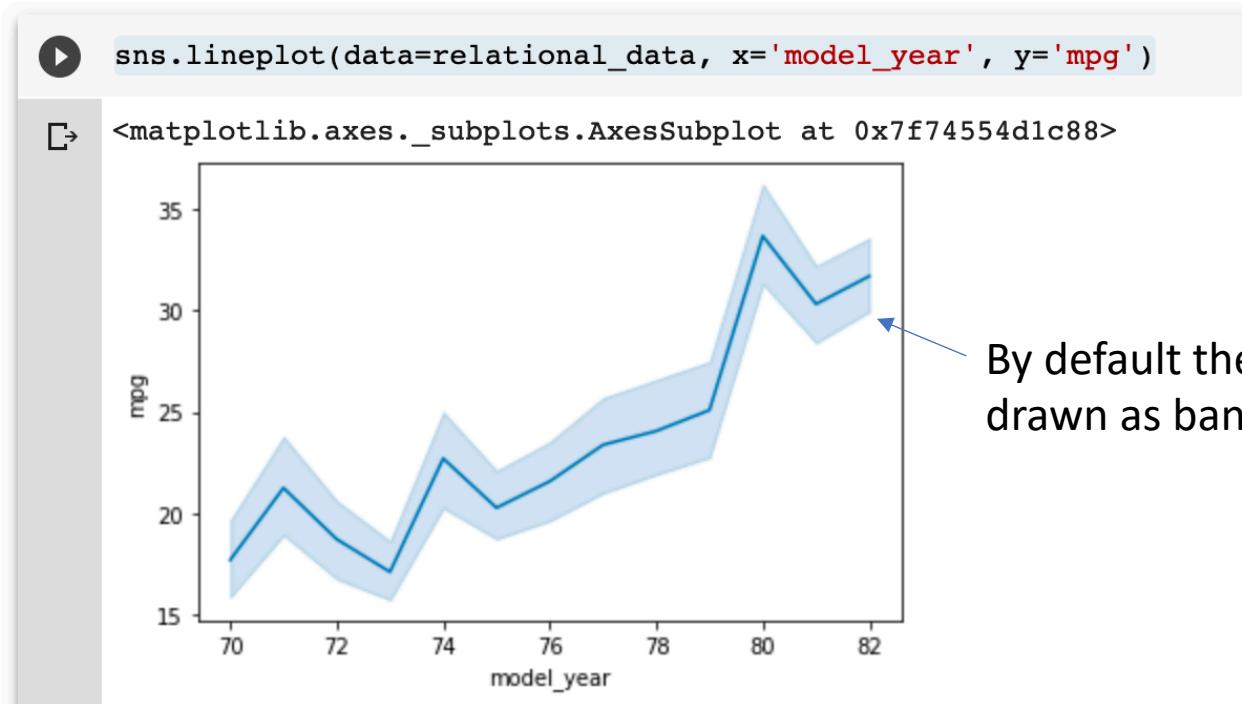
	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	name
0	18.0	8	307.0	130.0	3504	12.0	70	usa	chevrolet chevelle malibu
1	15.0	8	350.0	165.0	3693	11.5	70	usa	buick skylark 320
2	18.0	8	318.0	150.0	3436	11.0	70	usa	plymouth satellite
3	16.0	8	304.0	150.0	3433	12.0	70	usa	amc rebel sst
4	17.0	8	302.0	140.0	3449	10.5	70	usa	ford torino

## Line plots

- Can be used to compare means or medians over the span of the x-axis
- Does not convey distribution nor sample size

```
sns.lineplot(data=relational_data, x='model_year', y='mpg')
```

# Let's practice

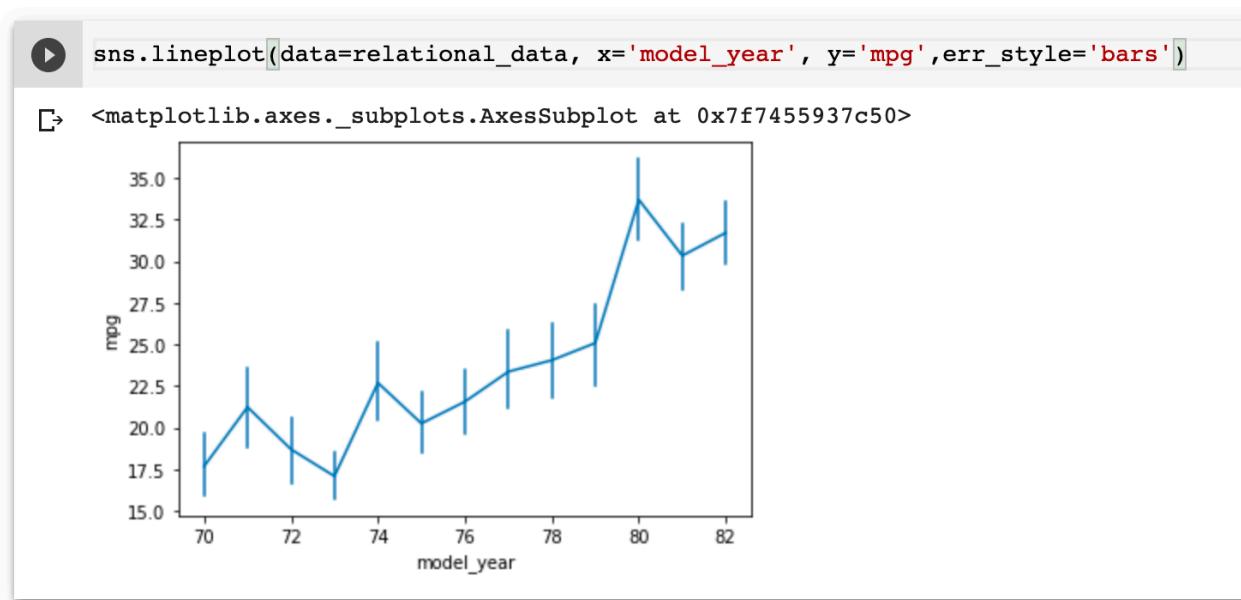


By default the distribution is drawn as bands around the line.

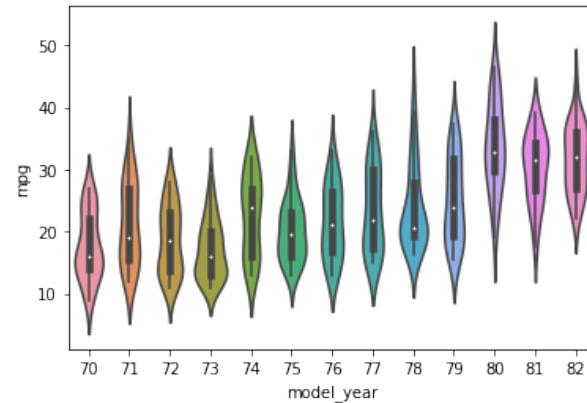
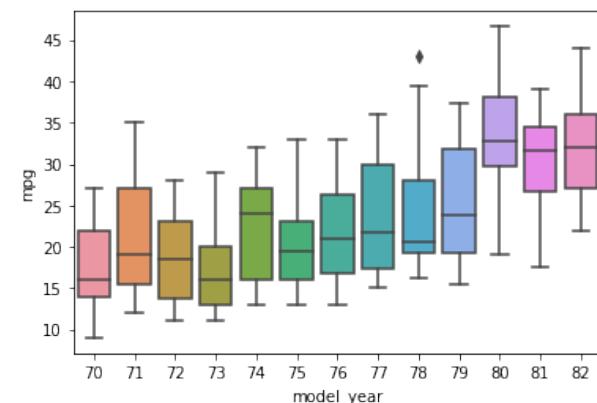
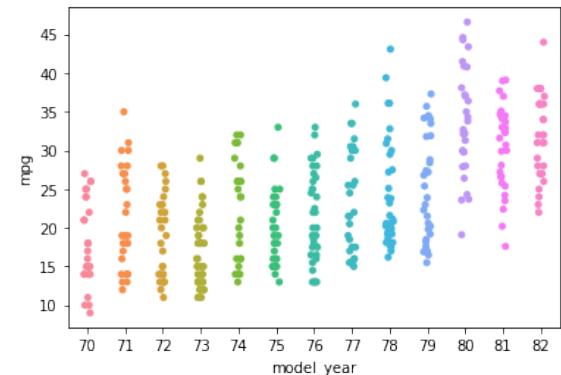
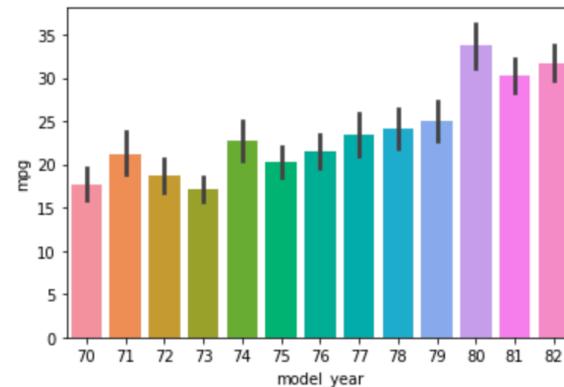
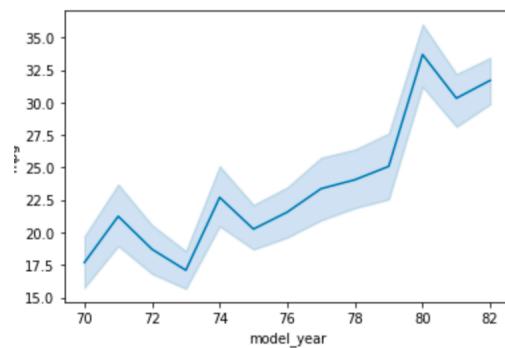
# Error bars

- Error bars can be added with the `err_style` parameter

```
sns.lineplot(data=relational_data, x='model_year', y='mpg', err_style='bars')
```



# Line plot can be represented as a categorical plot



bar  
strip  
box  
violin

```
sns._plot(data=relational_data, x='model_year', y='mpg')
```

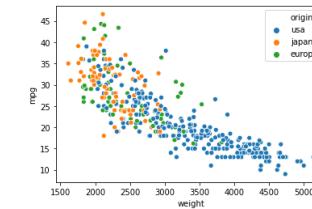
# More option for line plot

- ci
- hue

Read more about line plot at <https://seaborn.pydata.org/generated/seaborn.lineplot.html>

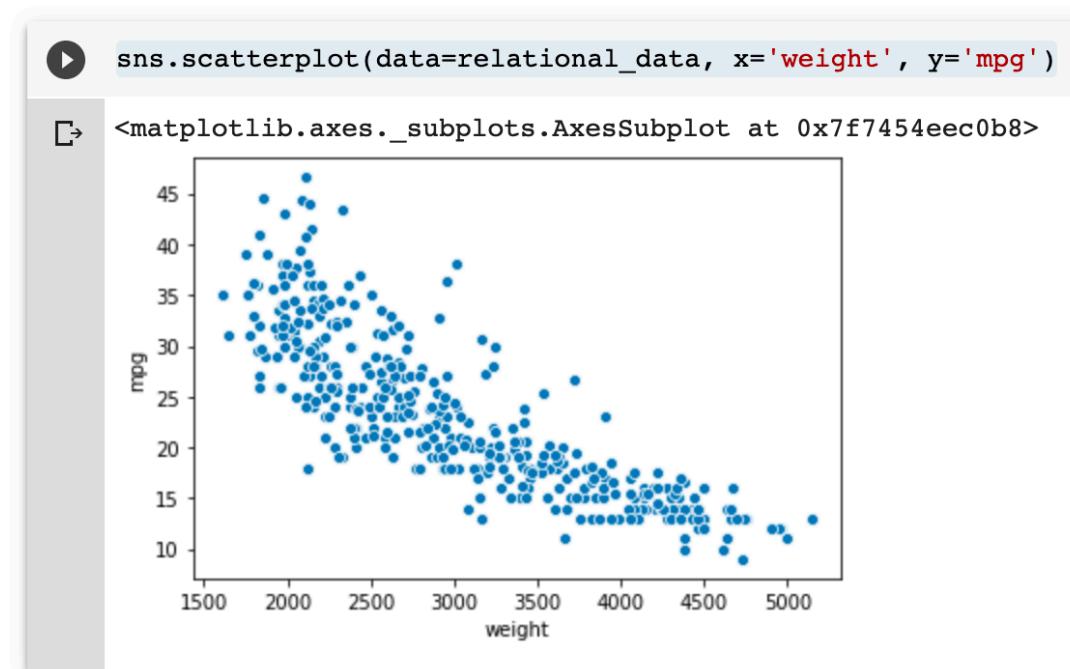
# Scatter plots

- Visualizes the relationship between two numerical variables
- Clearly displays sample size
- Does not convey estimators like mean or median



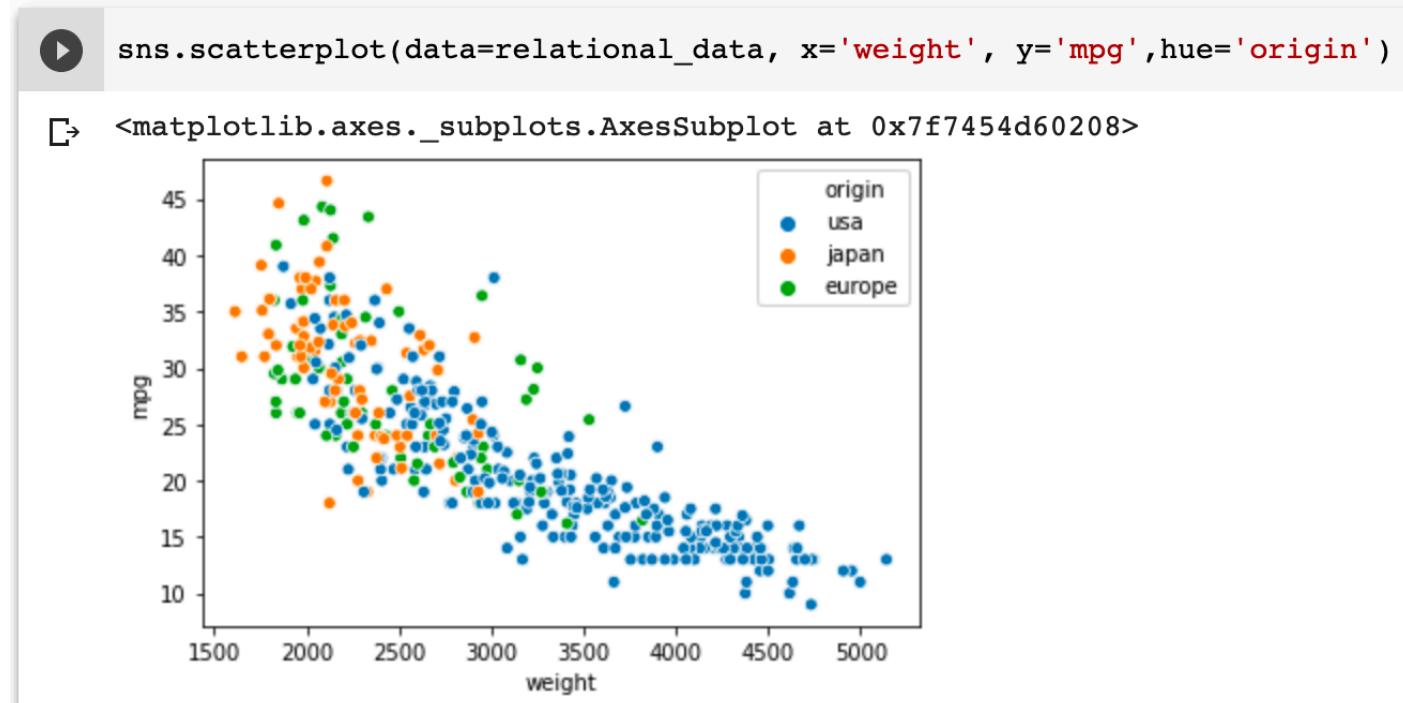
# Scatter plots using seaborn

```
sns.scatterplot(data=relational_data, x='weight', y='mpg')
```



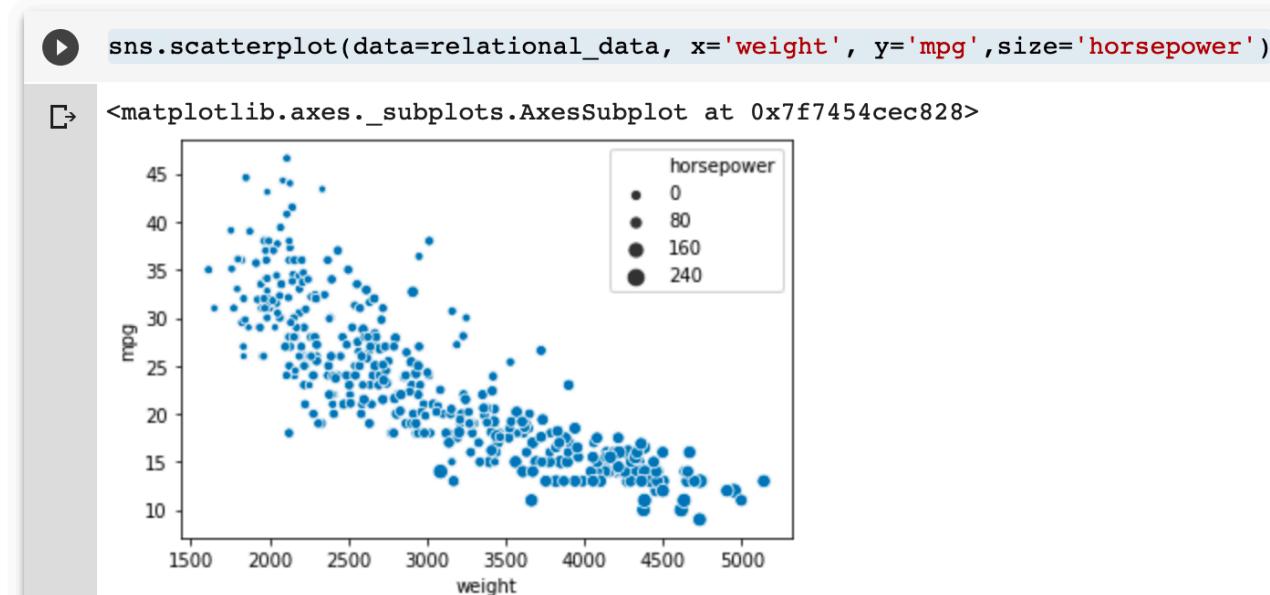
# Hue option for scatter plots

```
sns.scatterplot(data=relational_data, x='weight', y='mpg', hue='origin')
```



# Size of data points

```
sns.scatterplot(data=relational_data, x='weight', y='mpg', size='horsepower')
```

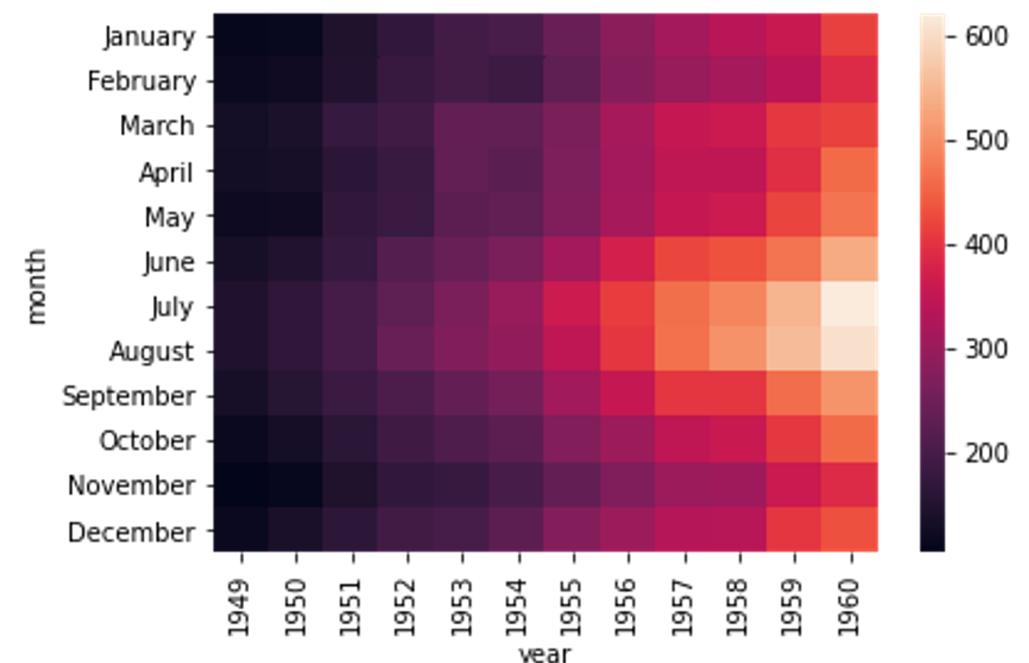


Read more about scatter plots at

# Heatmaps

- Plots specifically designed to display rectangular data
- What the plot displays depends on the value calculated in each intersection
- Particularly useful since many journals do not allow tables as part of multi-panel figures
- Only able to show one of sample size, median, or mean

Elements within the data are intersections of the categories



# Test dataset

- We will use the “flights” dataset
- However, this will need to be transformed to be rectangular

```
matrix_data = sns.load_dataset('flights')
```

```
▶ matrix_data = sns.load_dataset('flights')
matrix_data.head()
```

	year	month	passengers
0	1949	January	112
1	1949	February	118
2	1949	March	132
3	1949	April	129
4	1949	May	121

# Input data from heatmap

- Requires some work before plotting to get data into the correct shape

	year	month	passengers
0	1949	January	112
1	1949	February	118
2	1949	March	132
3	1949	April	129
4	1949	May	121



	year	1949	1950	1951	1952	1953	1954	1955	1956	1957	1958	1959	1960
	month												
	January	112	115	145	171	196	204	242	284	315	340	360	417
	February	118	126	150	180	196	188	233	277	301	318	342	391
	March	132	141	178	193	236	235	267	317	356	362	406	419
	April	129	135	163	181	235	227	269	313	348	348	396	461
	May	121	125	172	183	229	234	270	318	355	363	420	472

# The pivot function

- In Pandas, the pivot table function takes simple data frame as input, and performs grouped operations that provides a multidimensional summary of the data.

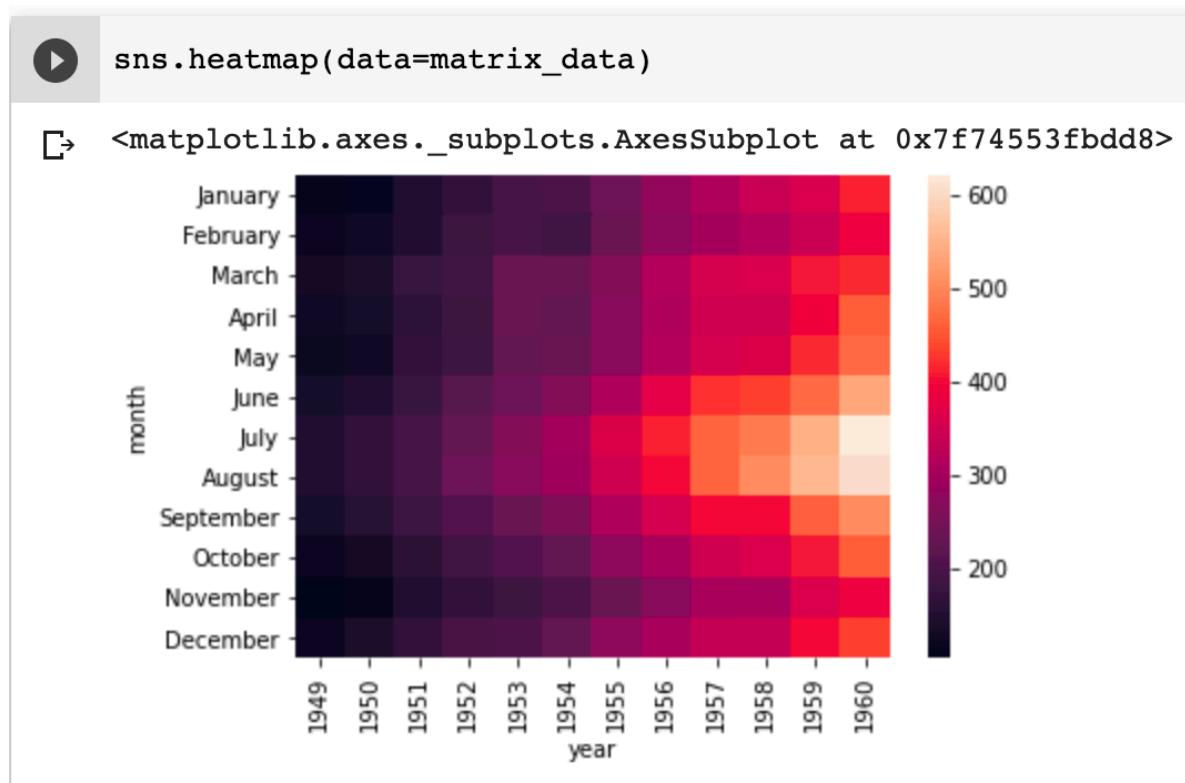
```
matrix_data = matrix_data.pivot('month', 'year', 'passengers')
```

```
matrix_data = sns.load_dataset('flights')
matrix_data = matrix_data.pivot('month', 'year', 'passengers')
matrix_data.head()
```

	year	1949	1950	1951	1952	1953	1954	1955	1956	1957	1958	1959	1960
	month												
January		112	115	145	171	196	204	242	284	315	340	360	417
February		118	126	150	180	196	188	233	277	301	318	342	391
March		132	141	178	193	236	235	267	317	356	362	406	419
April		129	135	163	181	235	227	269	313	348	348	396	461
May		121	125	172	183	229	234	270	318	355	363	420	472

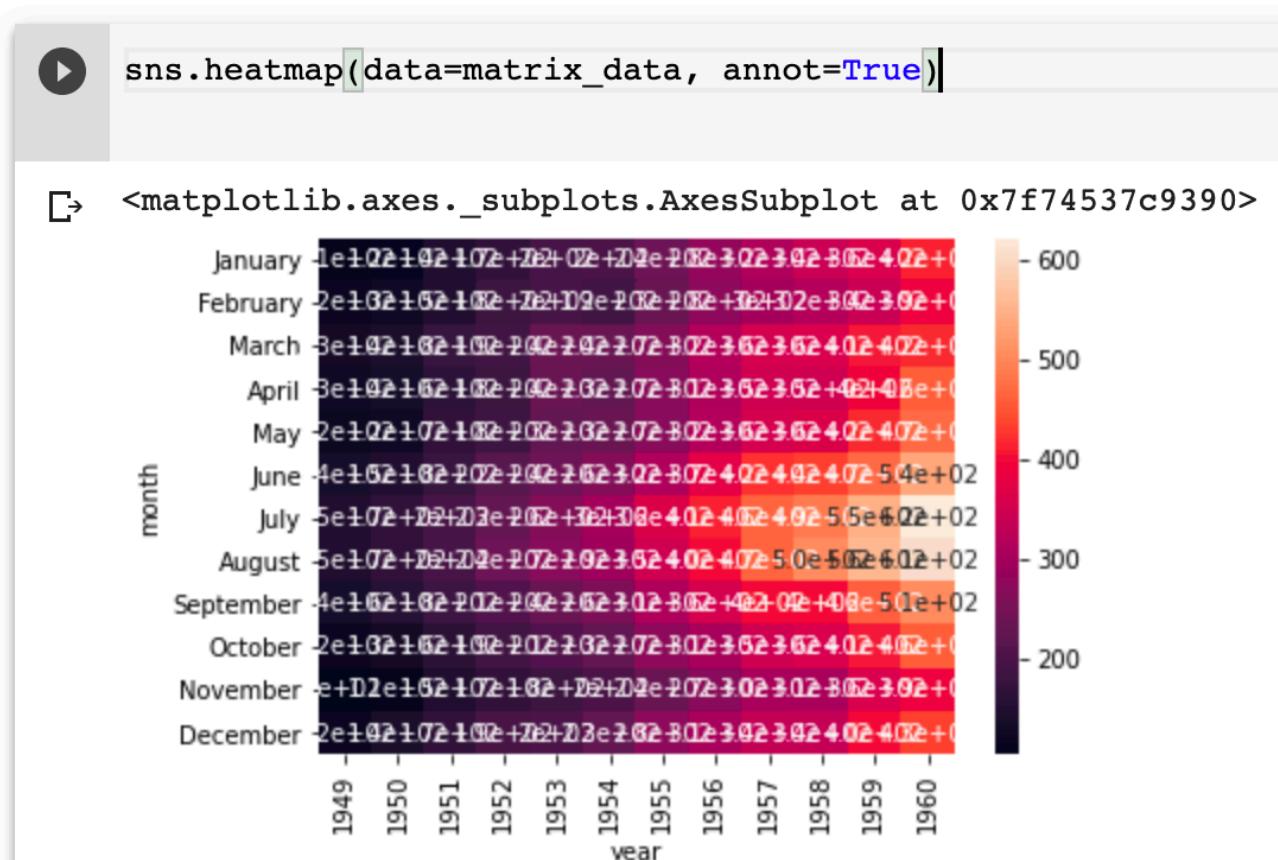
# Heatmap

```
sns.heatmap(data=matrix_data)
```



# Display the raw values in heatmap

```
sns.heatmap(data=matrix_data, annot=True)
```



# fmt parameter

fmt='d'

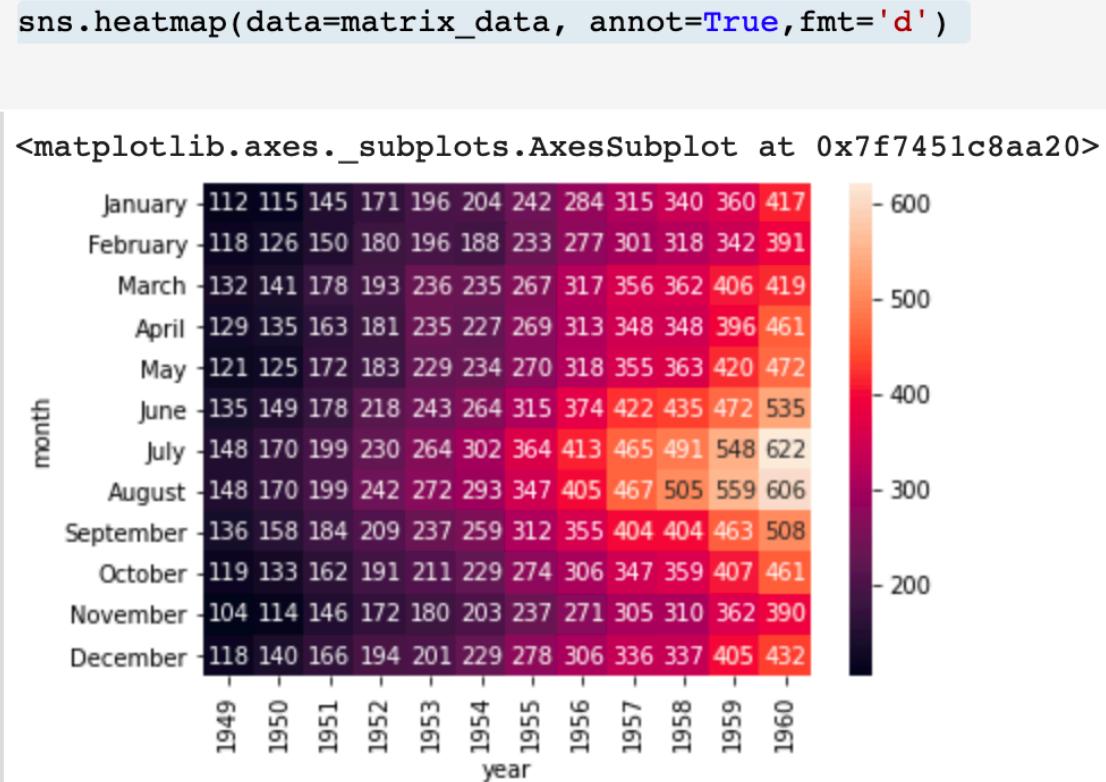
- 'd' is for a decimal integer

fmt='.nf'

- $n$  is the number of sig figs to include on either side of the decimal point
- '.1f' would create 602.0

# Let's practice

```
sns.heatmap(data=matrix_data, annot=True, fmt='d')
```



# Customizing color palette

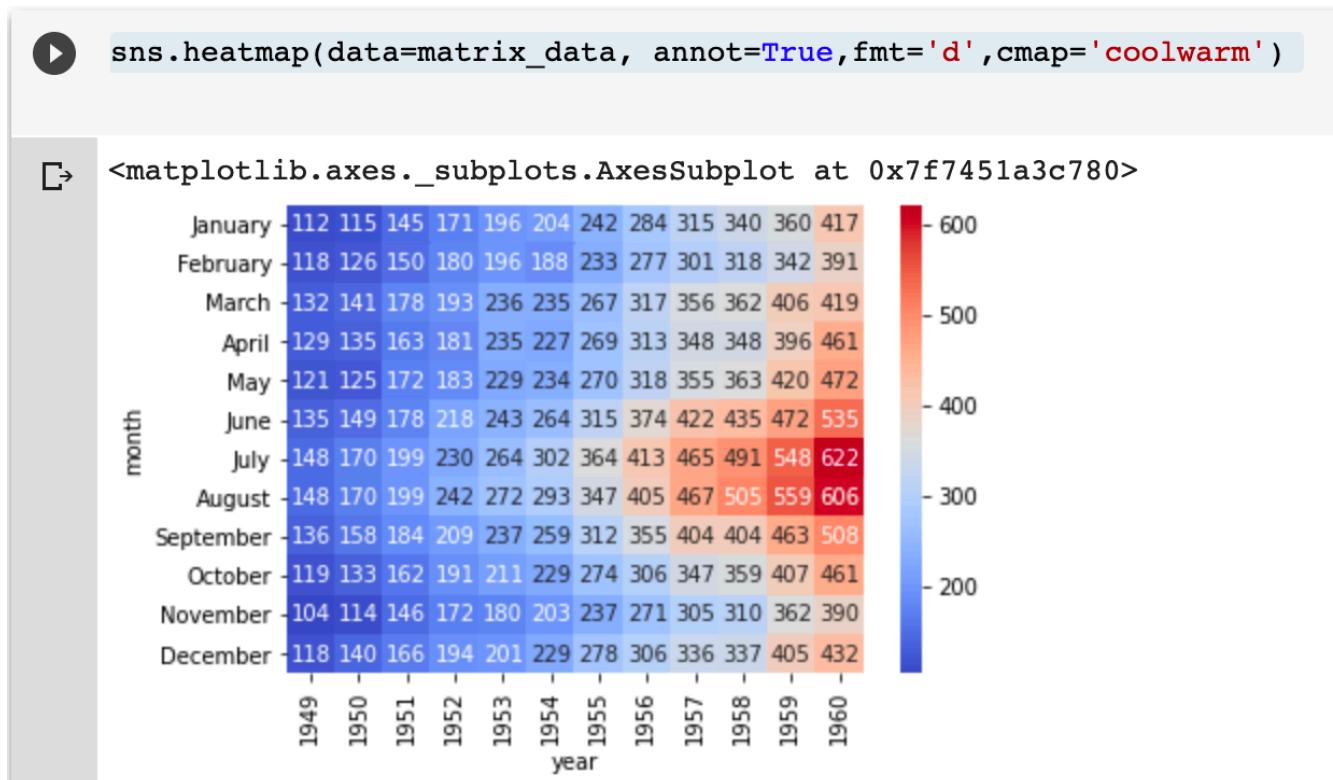
- Every plot has the option of changing the color palette
  - Seaborn/Matplotlib have a variety of premade palettes to choose from
    - [https://seaborn.pydata.org/tutorial/color\\_palettes.html](https://seaborn.pydata.org/tutorial/color_palettes.html)
- It is also possible to create your own palette
  - Simplest way is to use the hex codes of colors you'd like to use (e.g. '000000', '#f7941f', '#00b9f2', '#00a875', '#ecde38', '#f15a22', '#da6fab')
  - It is highly recommended you use a color-blind friendly palette like this one

Color	Color name	RGB (1–255)	CMYK (%)	P	D
	Black	0, 0, 0	0, 0, 0, 100		
	Orange	230, 159, 0	0, 50, 100, 0		
	Sky blue	86, 180, 233	80, 0, 0, 0		
	Bluish green	0, 158, 115	97, 0, 75, 0		
	Yellow	240, 228, 66	10, 5, 90, 0		
	Blue	0, 114, 178	100, 50, 0, 0		
	Vermillion	213, 94, 0	0, 80, 100, 0		
	Reddish purple	204, 121, 167	10, 70, 0, 0		

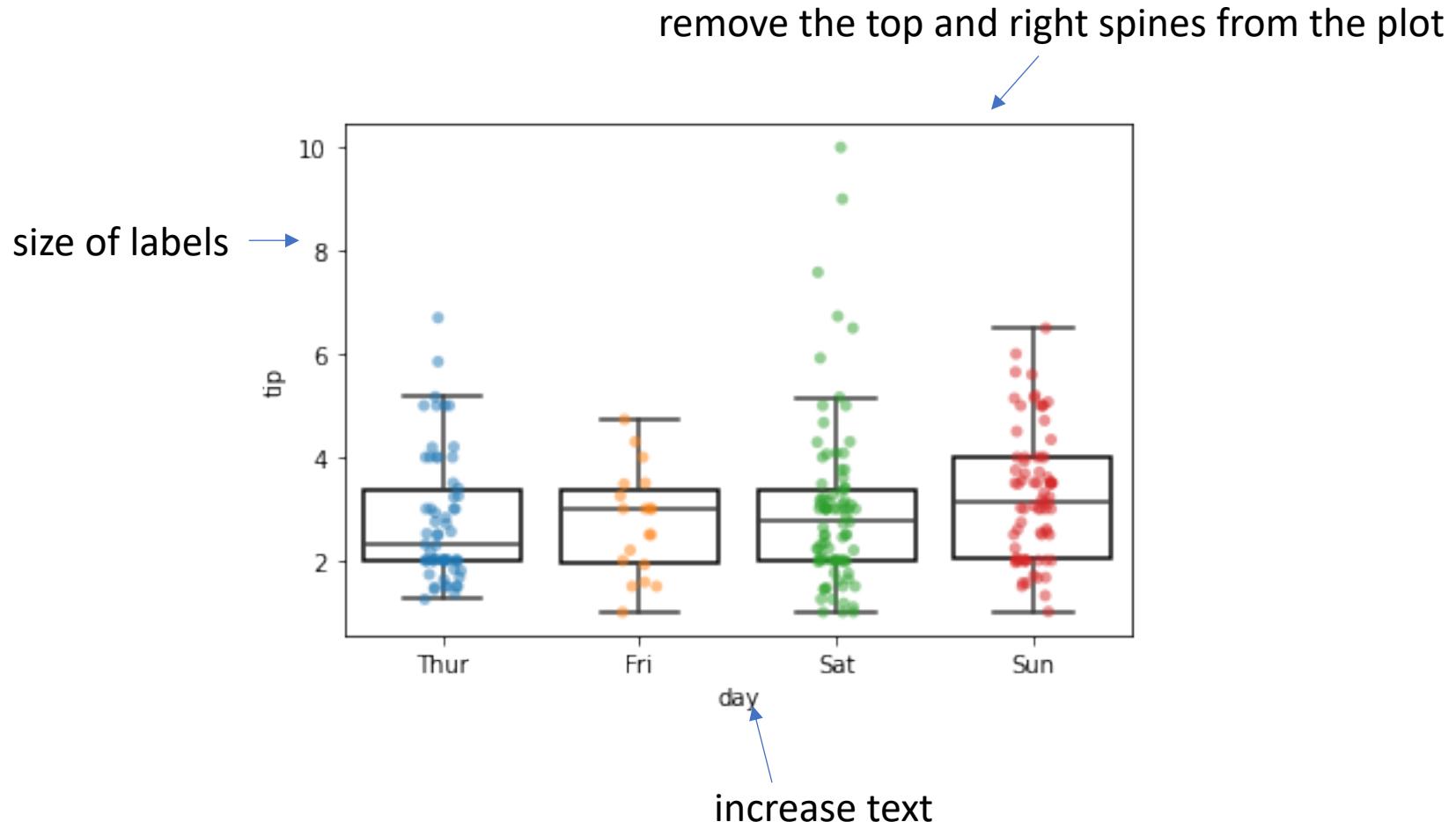
Wong, B. Points of view: Color blindness. *Nat Methods* 8, 441 (2011).  
<https://doi.org/10.1038/nmeth.1618>

# Customizing color palette

```
sns.heatmap(data=matrix_data, annot=True, fmt='d', cmap='coolwarm')
```



# Aesthetics of figure



# Parameters to improve aesthetics

- `sns.set_style('style_name')`
  - These styles impact the aesthetics of the plot (primarily the axes and background)
  - Options include 'darkgrid', 'whitegrid', 'dark', 'white', and 'ticks'
- `sns.set_context('context_name')`
  - These contexts impact the size of labels, lines and other elements
  - Options include 'paper', 'notebook', 'talk', and 'poster'
- `sns.despine()`
  - This is not global, but is a nice utility to use on individual plots
  - It removes the top and right spines from the plot



# Let's practice

```
g=sns.set_style("white")
g=sns.set_context("poster")
g=sns.boxplot(data=categorical_data, x='day',
y='tip',fliersize=0,boxprops=dict({'edgecolor': 'black', 'facecolor': (1,1,1,0)}))
g=sns.stripplot(data=categorical_data, x='day', y='tip',alpha=.5)
g.set(xlabel='Tip', ylabel='Day')
g=sns.despine()
```

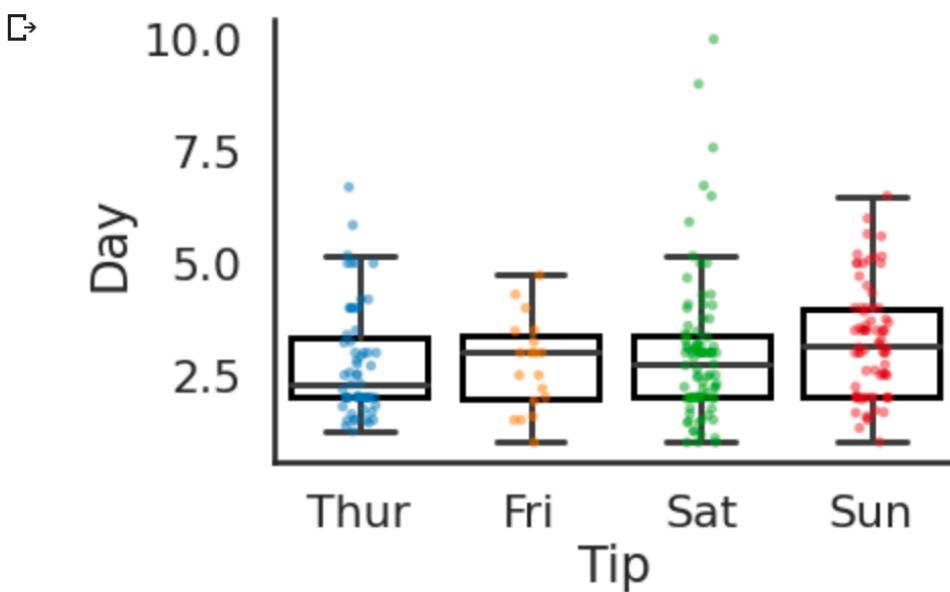
Editing axis titles to change the label text (make it more descriptive)



# Let's practice



```
g=sns.set_style("white")
g=sns.set_context("poster")
g=sns.boxplot(data=categorical_data, x='day', y='tip',fliersize=0,boxprops=dict({'edgecolor': 'black', 'facecolor': (1,1,1,0)}))
g=sns.stripplot(data=categorical_data, x='day', y='tip',alpha=.5)
g.set(xlabel='Tip', ylabel='Day')
g=sns.despine()
```



# Saving plots to a file

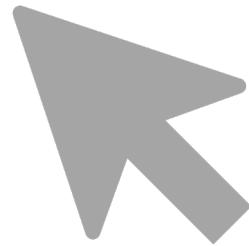
- Once you are satisfied with your figure, you can save it to a file
- This is done with the matplotlib savefig function, which requires the filename as an argument



# Format of file



The file extension you use in your filename  
determines the filetype it is saved as



In Google Colab, you must specify the path  
in your filename

# PNG vs PDF

- While PNG and PDF both are formats you can save a figure as, they have different properties
  - PNG files are rasterized, meaning that it's already been converted to pixels
  - PDF files are vector images, meaning they are still defined as vectors on a cartesian plane
- Why does this distinction matter?
  - If you save a file as a raster image, it is no longer editable
    - Increasing the size of the image increases the size of the pixels
  - A vector image is still editable
    - increasing the size of the image will properly scale the entire plane accordingly
- It is recommended to always save plots as vector images, if possible, so that it can be edited/scaled as necessary later

# Import matplotlib

```
import matplotlib.pyplot as plt
```

# Let's practice

Import matplotlib

```
import matplotlib.pyplot as plt  
g=sns.set_style("white")  
g=sns.set_context("poster")  
g=sns.boxplot(data=categorical_data, x='day',  
y='tip',fliersize=0,boxprops=dict({'edgecolor': 'black', 'facecolor': (1,1,1,0)}))  
g=sns.stripplot(data=categorical_data, x='day', y='tip',alpha=.5)  
g.set(xlabel='Tip', ylabel='Day')  
g=sns.despine()  
plt.savefig("/content/drive/My Drive/data_science_workshop/boxplot.pdf")
```

Path to Google Drive  
Make sure rive already  
mounted

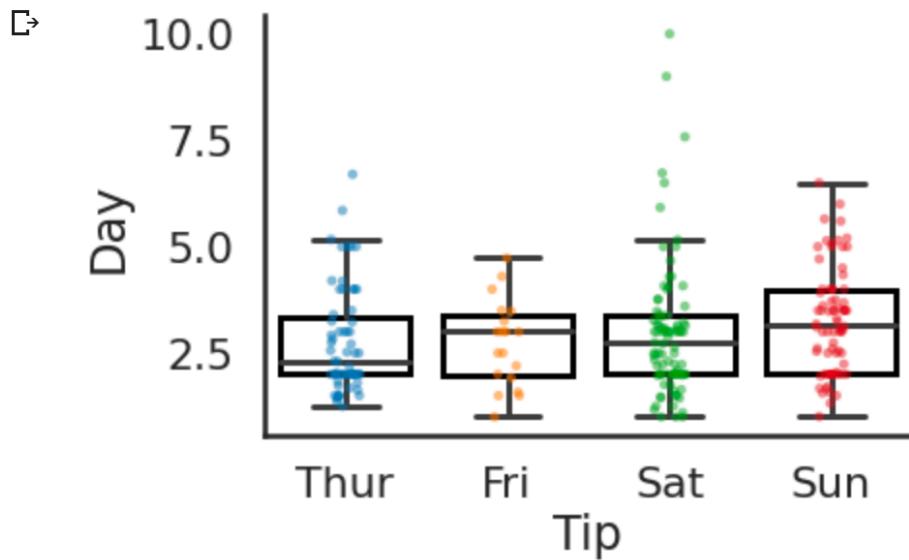
Directory you created  
in your Google Drive

Pdf file to be created  
with the figure

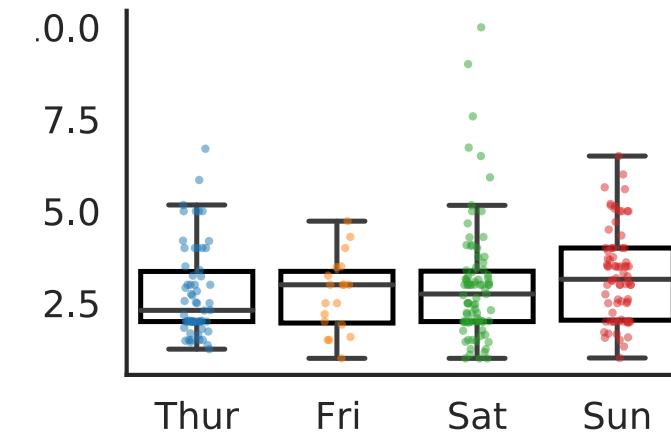
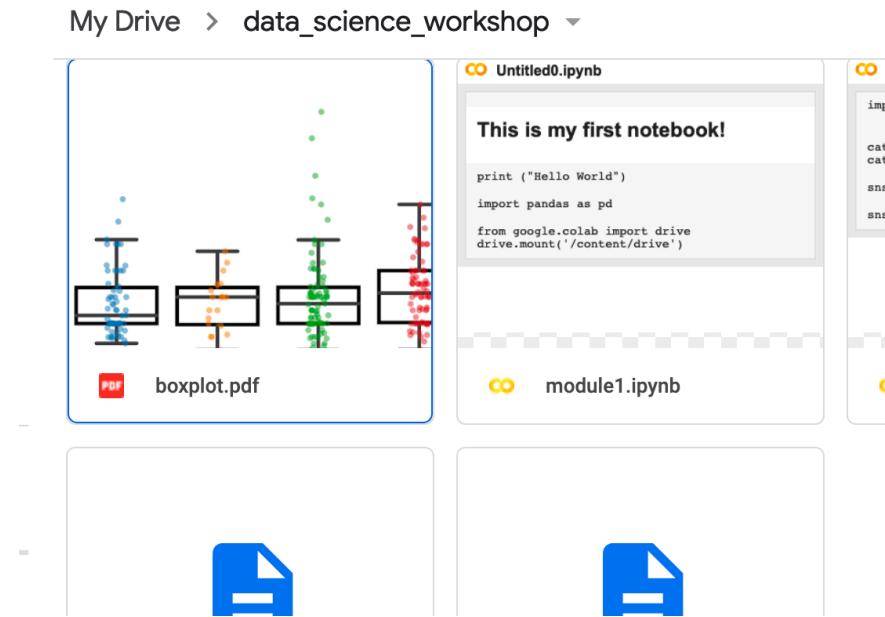
```
from google.colab import drive  
drive.mount('/content/drive')
```

# Let's practice

```
import matplotlib.pyplot as plt
g=sns.set_style("white")
g=sns.set_context("poster")
g=sns.boxplot(data=categorical_data, x='day', y='tip',fliersize=0,boxprops=dict({'edgecolor': 'black', 'facecolor': (1,1,1,0)}))
g=sns.stripplot(data=categorical_data, x='day', y='tip',alpha=.5)
g.set(xlabel='Tip', ylabel='Day')
g.despine()
plt.savefig("/content/drive/My Drive/data_science_workshop/boxplot.pdf")
```



# Let's open the file



# Fit the issues of truncated figure

```
g.set(xlabel='Tip', ylabel='Day')
```



- If you customize your plot legend to be outside of that range, or if you have plot elements that extend beyond that range, they can be cut out of the saved figure.
- By using `bbox_inches='tight'`, you will make sure to encapsulate all plot elements in the tightest bbox possible. This makes sure those elements aren't cut out.

```
import matplotlib.pyplot as plt
g=sns.set_style("white")
g=sns.set_context("poster")
g=sns.boxplot(data=categorical_data, x='day', y='tip', fliersize=0, boxprops=dict({'edgecolor': 'black', 'facecolor': (1,1,1,0)}))
g=sns.stripplot(data=categorical_data, x='day', y='tip', alpha=.5)
g.set(xlabel='Tip', ylabel='Day')
g.despine()
plt.savefig("/content/drive/My Drive/data_science_workshop/boxplot.pdf",bbox_inches='tight')
```

Added

# Nature Requirements: Reporting Summary

- Required with submission to any Nature family journal
- Sets out guidelines for submitting reproducible work
- This impacts how we must produce and share data and visualizations

Editorial | [Open Access](#) | Published: 12 September 2018

## **Reproducibility: let's get it right from the start**

*Nature Communications* 9, Article number: 3716 (2018) | [Cite this article](#)

23k Accesses | 2 Citations | 118 Altmetric | [Metrics](#)

From September 12th 2018, *Nature Communications* will be setting a higher standard of data reporting for papers under peer review. We believe that sharing raw data at an early stage with editors and reviewers is the best way to build confidence in the reproducibility of your findings. Learn here how to ensure that your paper makes the grade.

# Steps to comply with requirements and beyond

1

Share data filtering  
and preprocessing of  
the data

2

Use effective  
visualization to show  
distribution and  
underlying data  
points

3

Share the code and  
data to make it  
reproducible