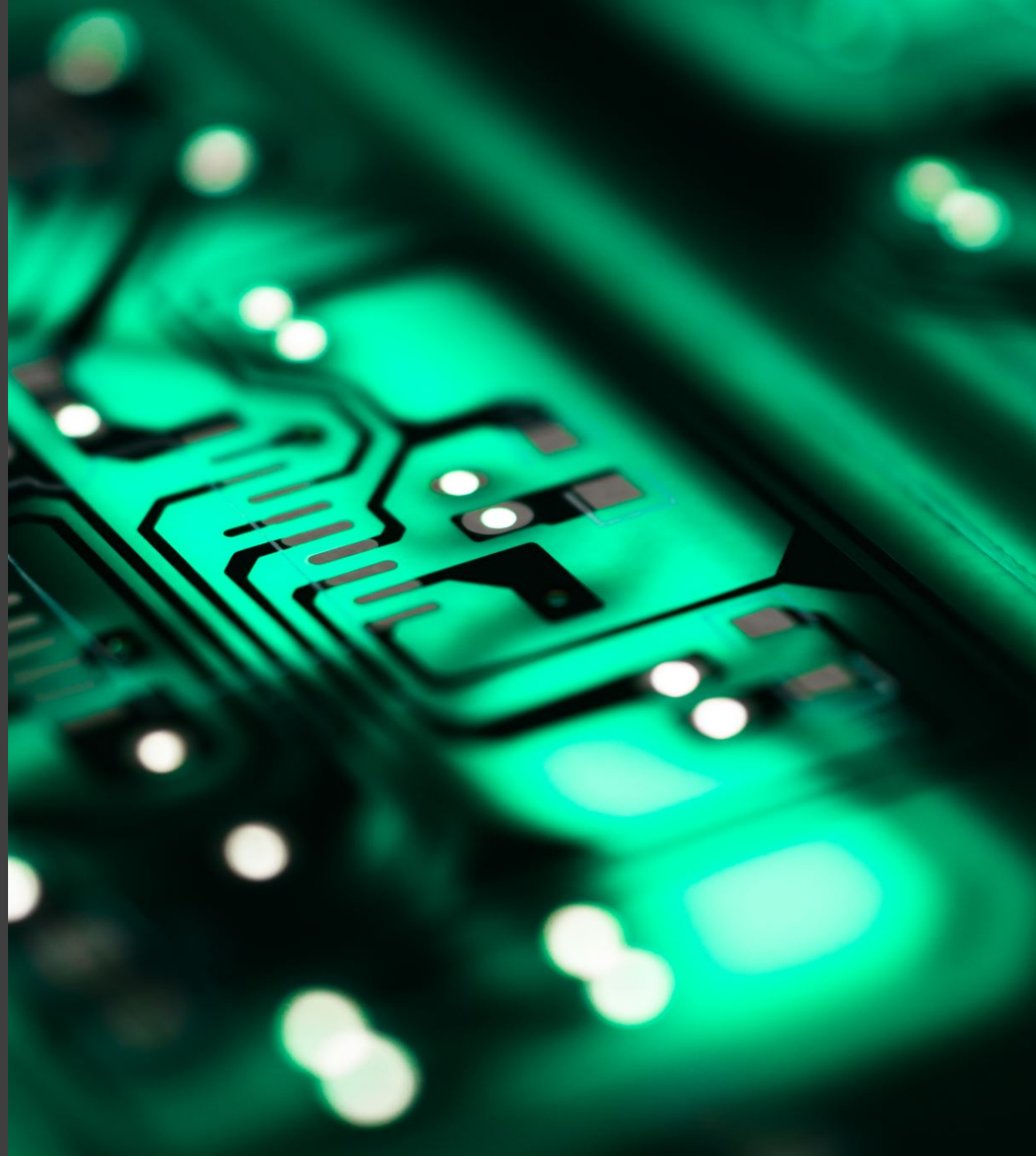
Effective data visualization using python

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

*Excel, Prism, etc.

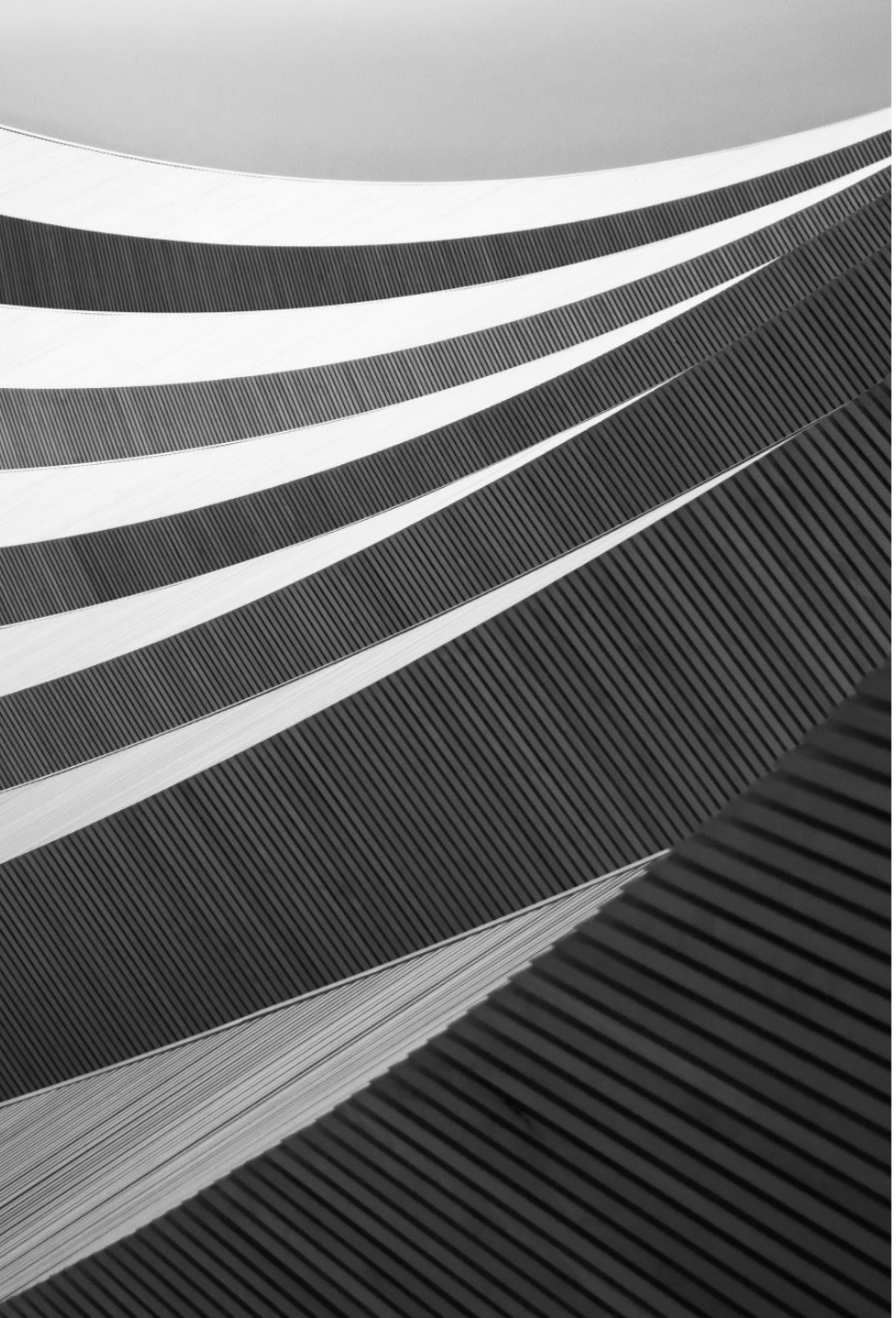
Code-based visualization

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 looks 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

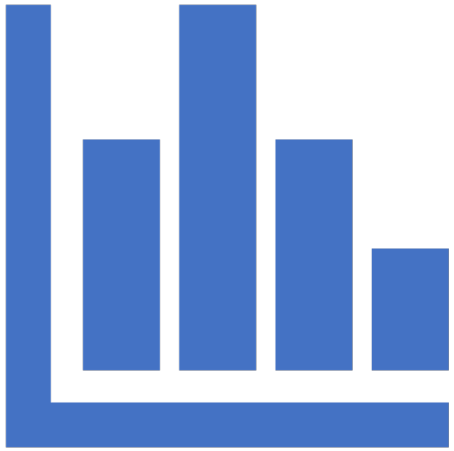
Relational plots

- Line plots
- Scatter plots

Matrix Plots

- Heatmaps
- Clustermaps

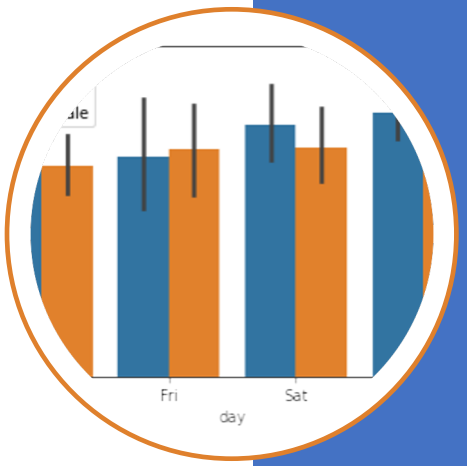
Categorical plots



- 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

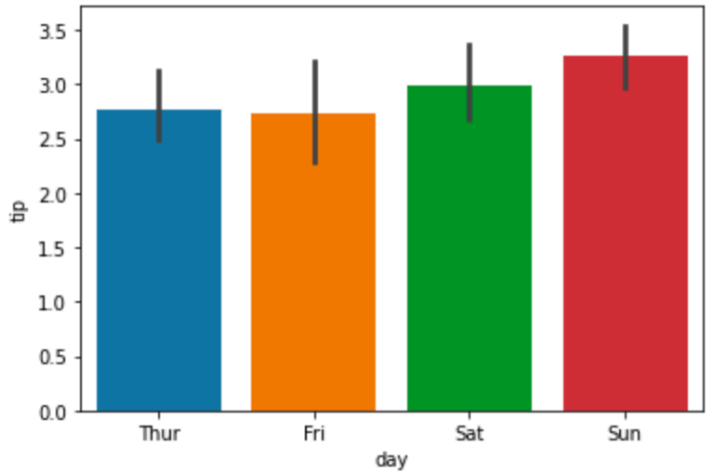
```
sns.barplot(data=categorical_data, x='day', y='tip')
```

Name of the data frame

Column names

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

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5bca0a0048>
```



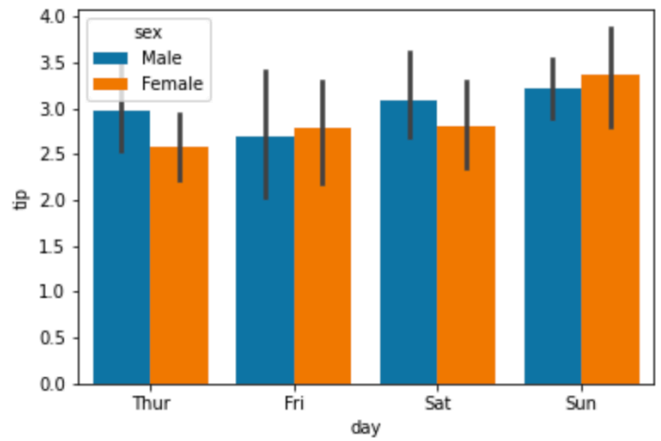
Bar plot with nested grouping by a two variable

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

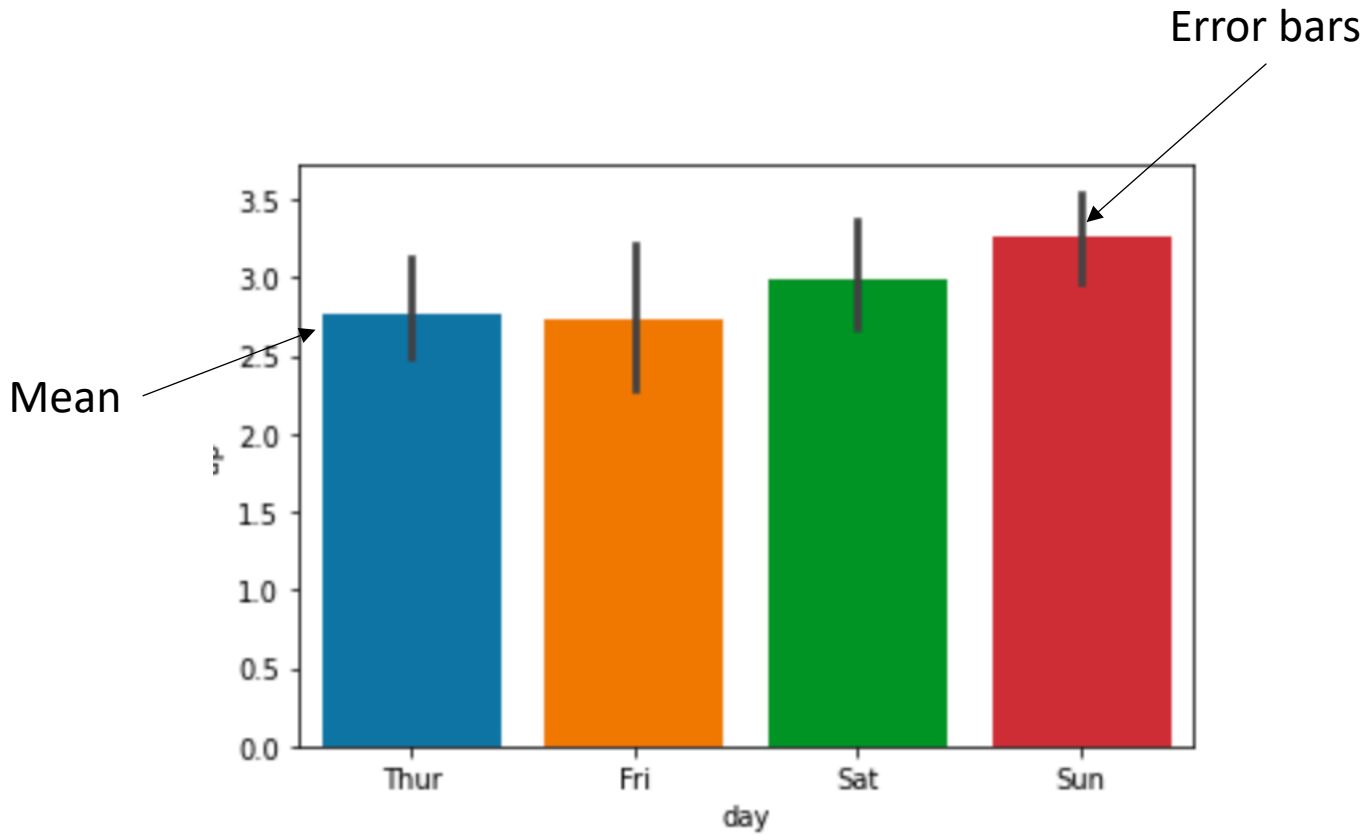
Split each bar into two based on this variable

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

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5bc9b4de80>
```



Parameters of bar plot



Median instead of mean

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

NumPy library

Median

NumPy

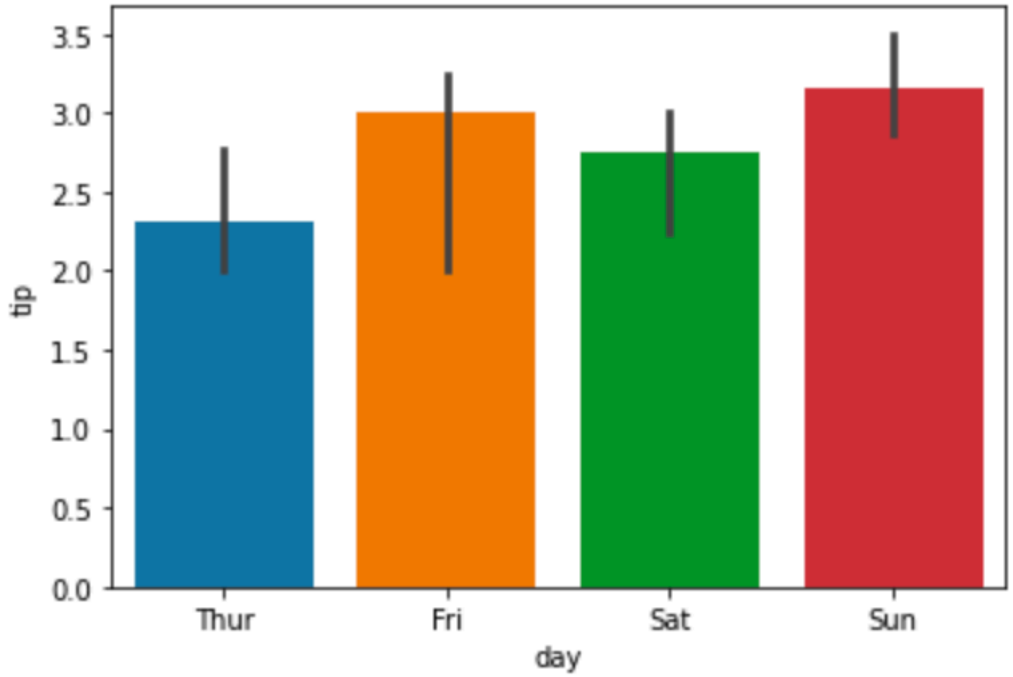
```
import numpy as np
```

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)
```

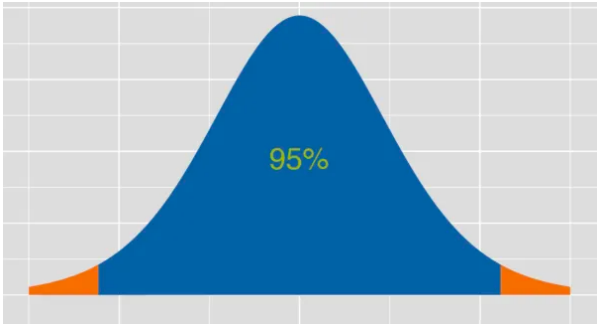
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f06ca11dd68>
```



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



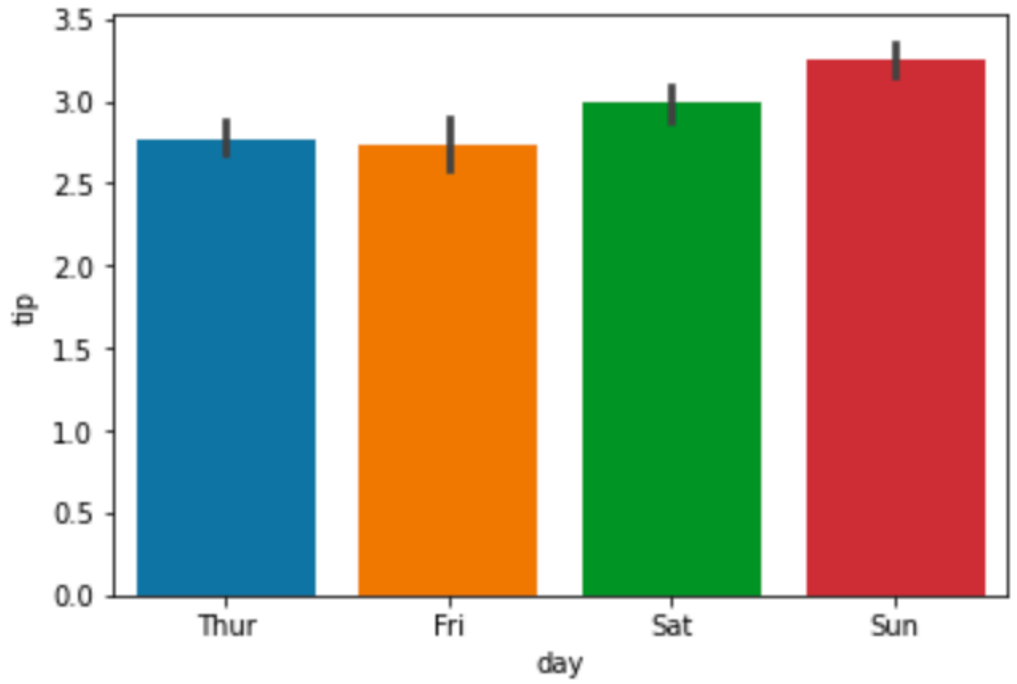

```
sns.barplot(data=categorical_data, x='day', y='tip',ci=50)
```

Default confidence interval for error bars is 95%

Let’s practice

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

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f06c9f5f080>
```



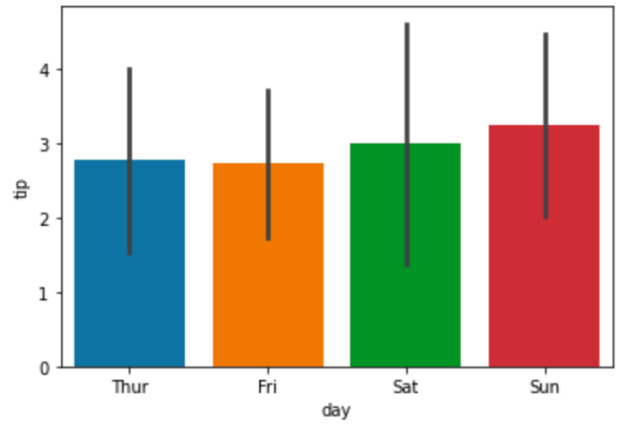
Change confidence interval

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

changes this to just use standard deviation

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

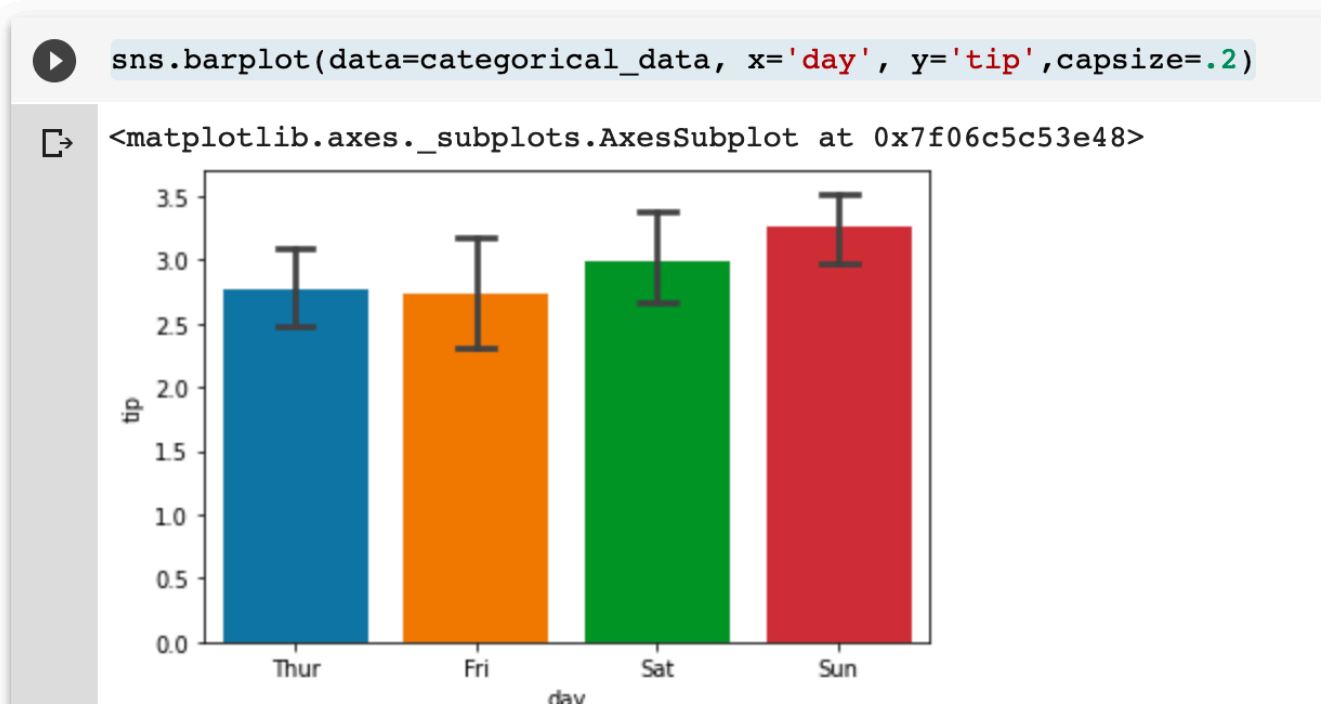
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f06c9e5eda0>
```



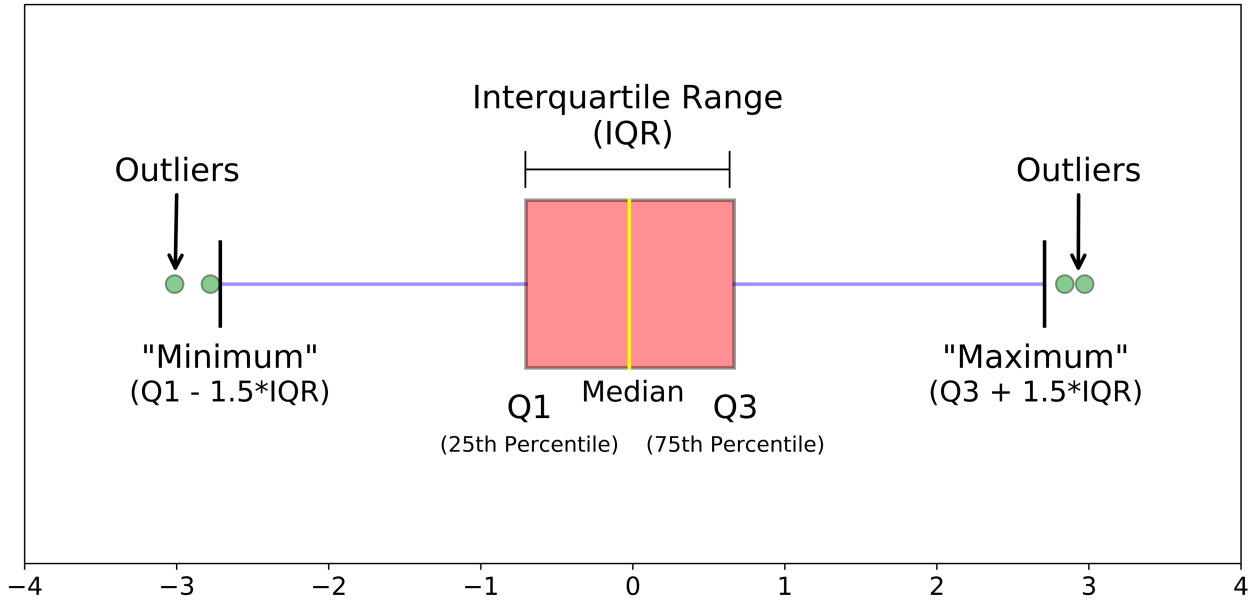
More options for bar plot

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

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



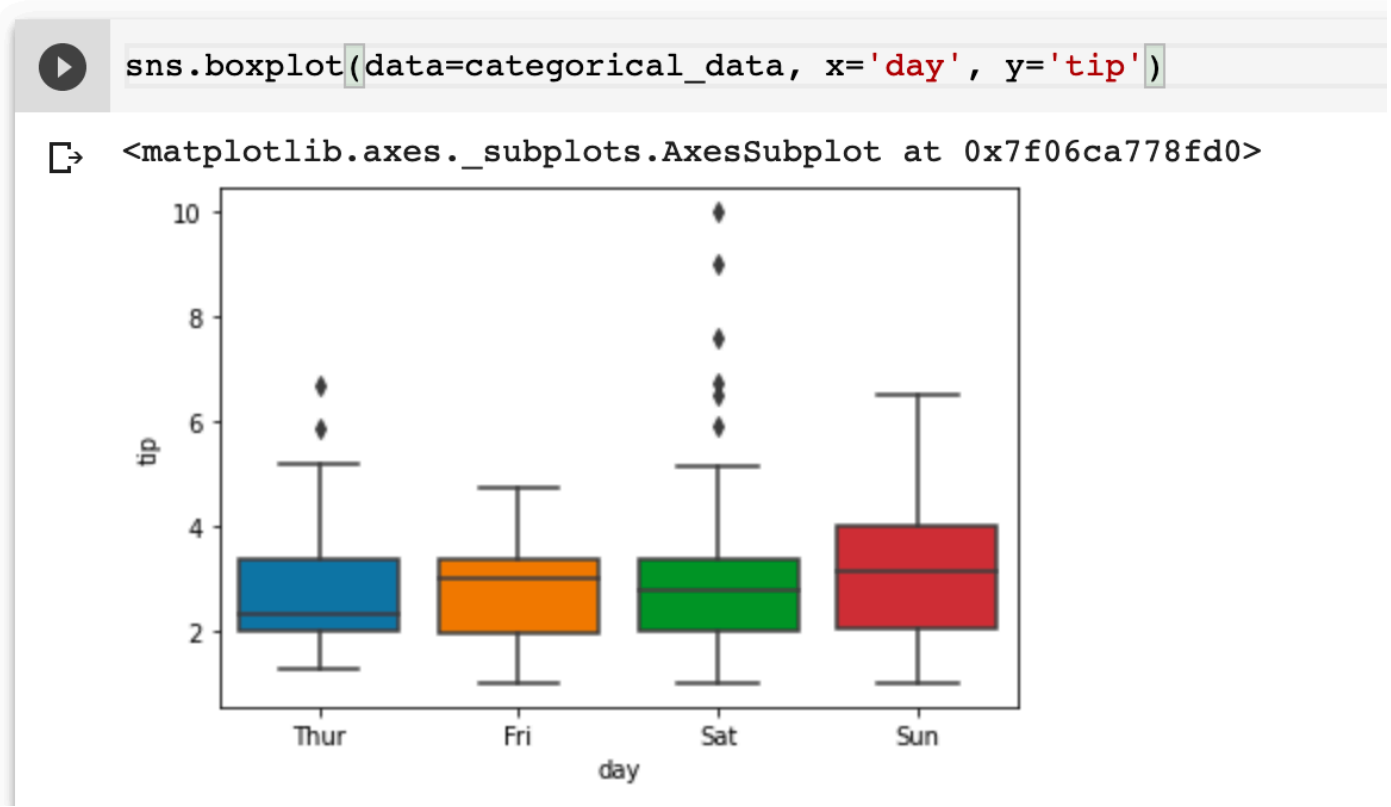
Box plots



- 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

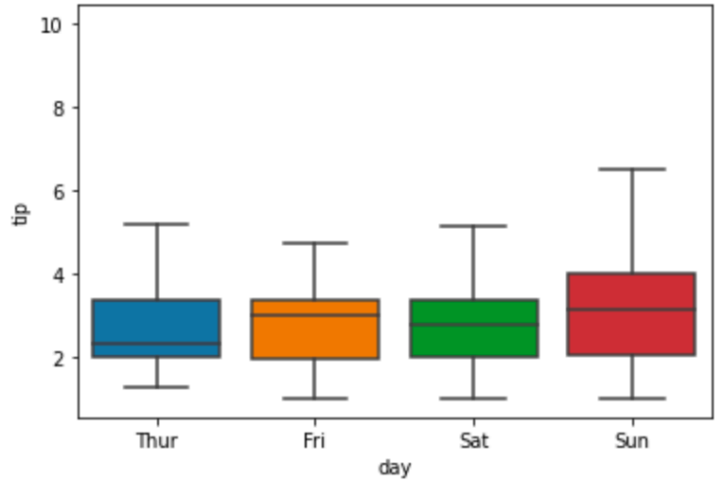
Let’s practice

```
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>
```



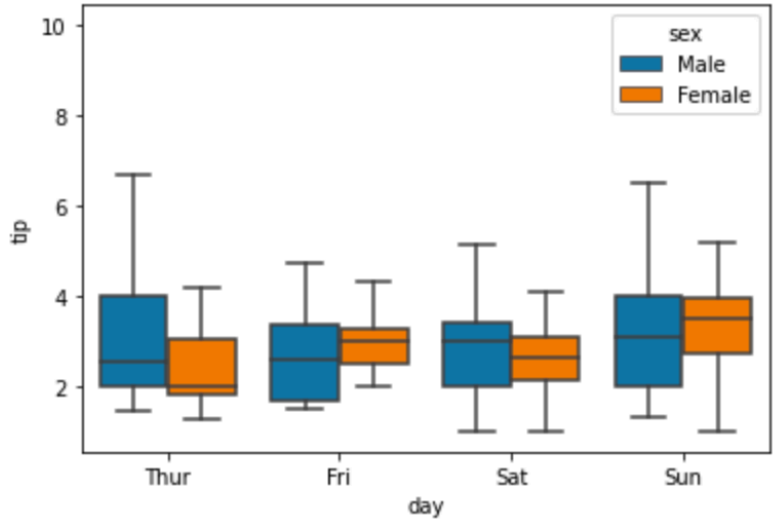
Box plot with nested grouping by a two variable

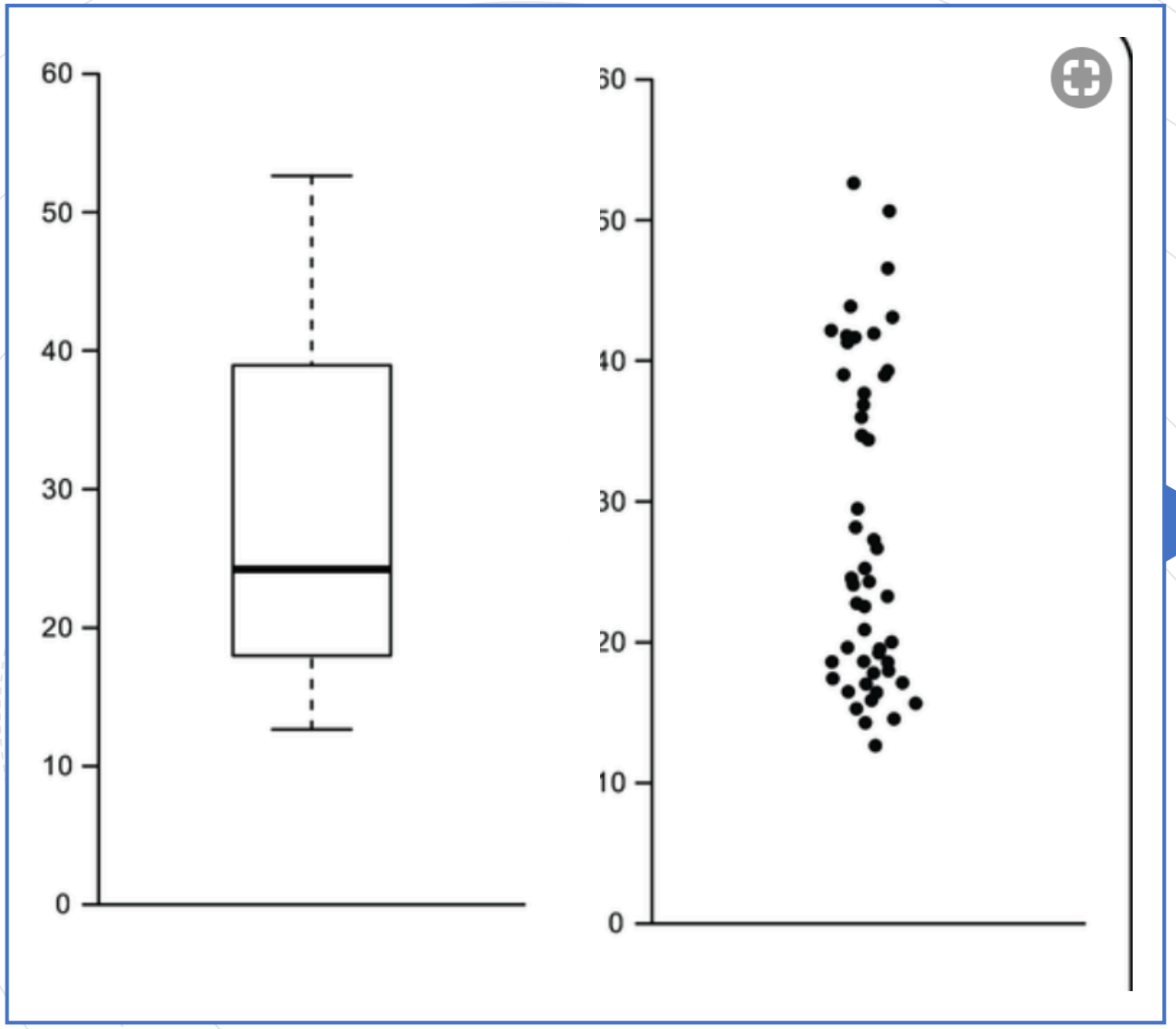
```
sns.boxplot(data=categorical_data, x='day', y='tip',fliersize=0,hue='sex')
```

Split each box plot into two based on this variable

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

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





Multimodal distribution

Box plots cannot clearly describe multimodal distributions

Martinez, Edson Zangiacomi. "Description of continuous data using bar graphs: a misleading approach." *Revista da Sociedade Brasileira de Medicina Tropical* 48.4 (2015): 494-497.

Strip plots

Often overlaid onto another plot

Clearly displays sample size and distribution

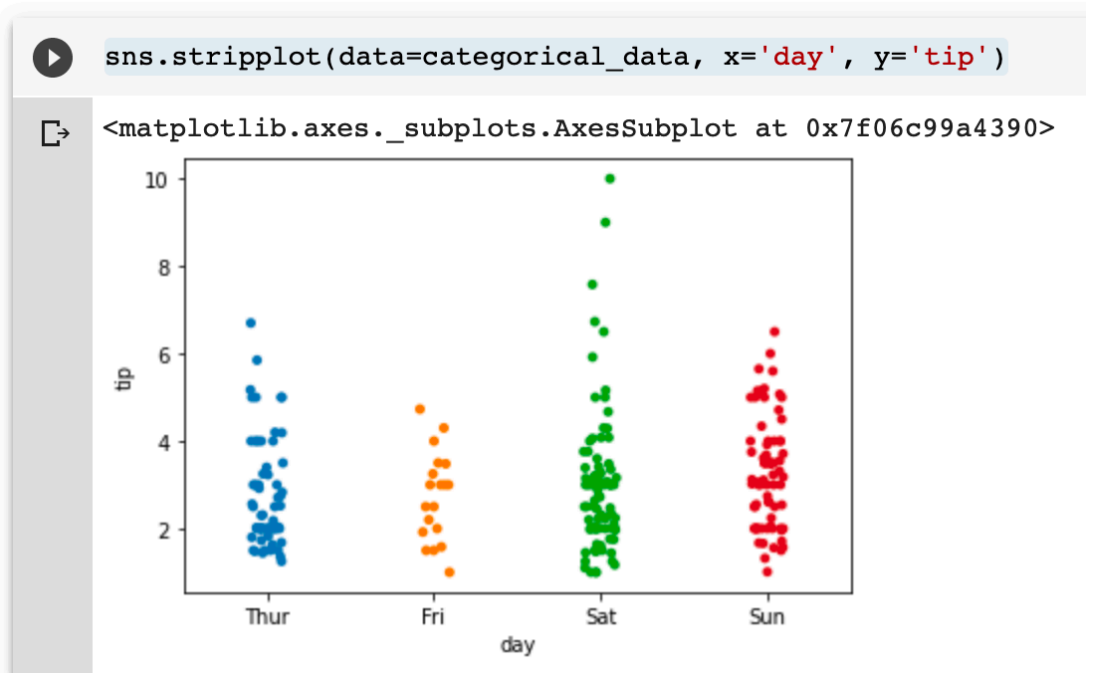
Does not convey estimators like mean or median

Strip plots

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

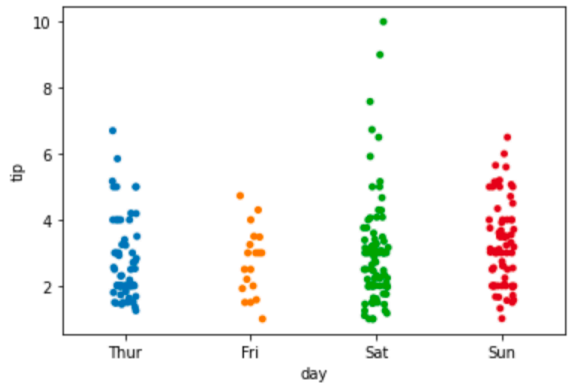
Let’s practice

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

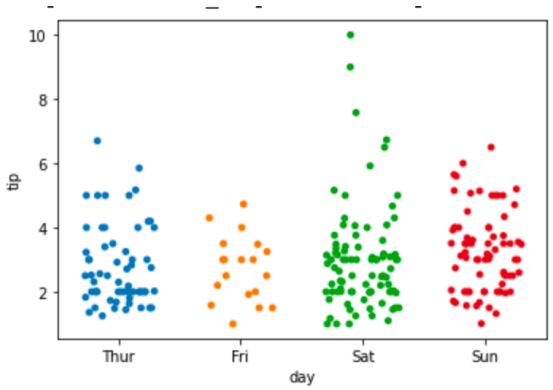


Jitter parameter

- 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



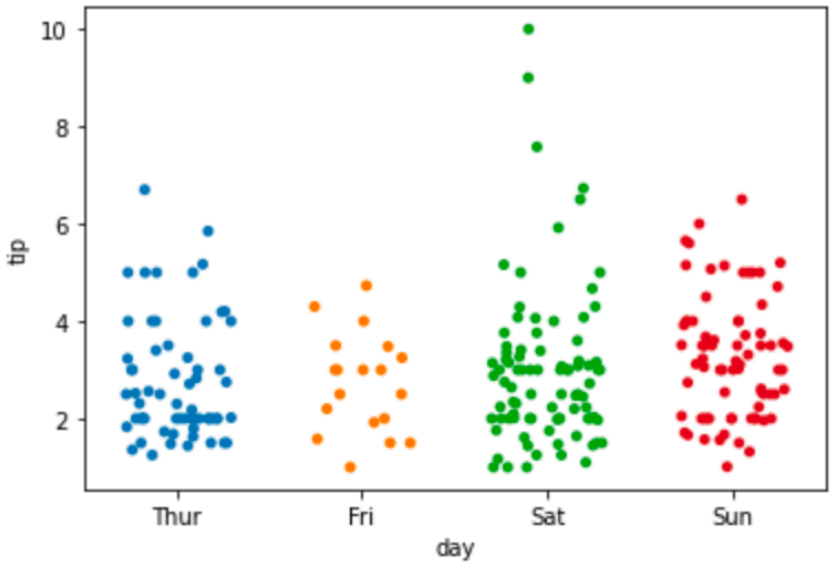
Jitter =0.3

Let’s practice

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

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

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f06c994fc50>
```

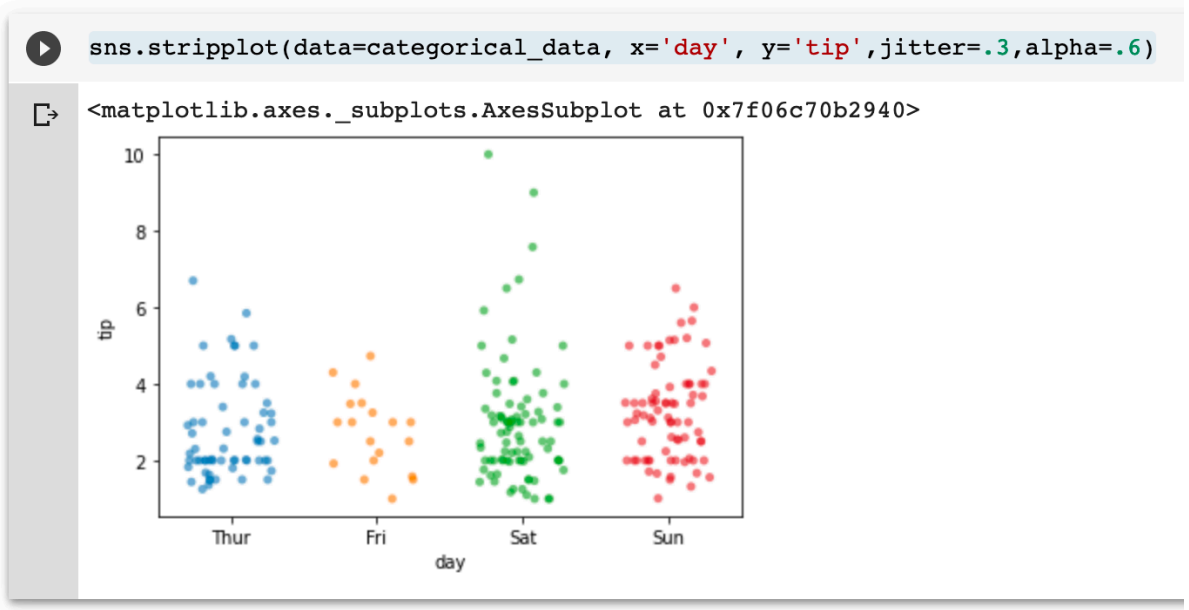


Overlapping points in strip plots

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

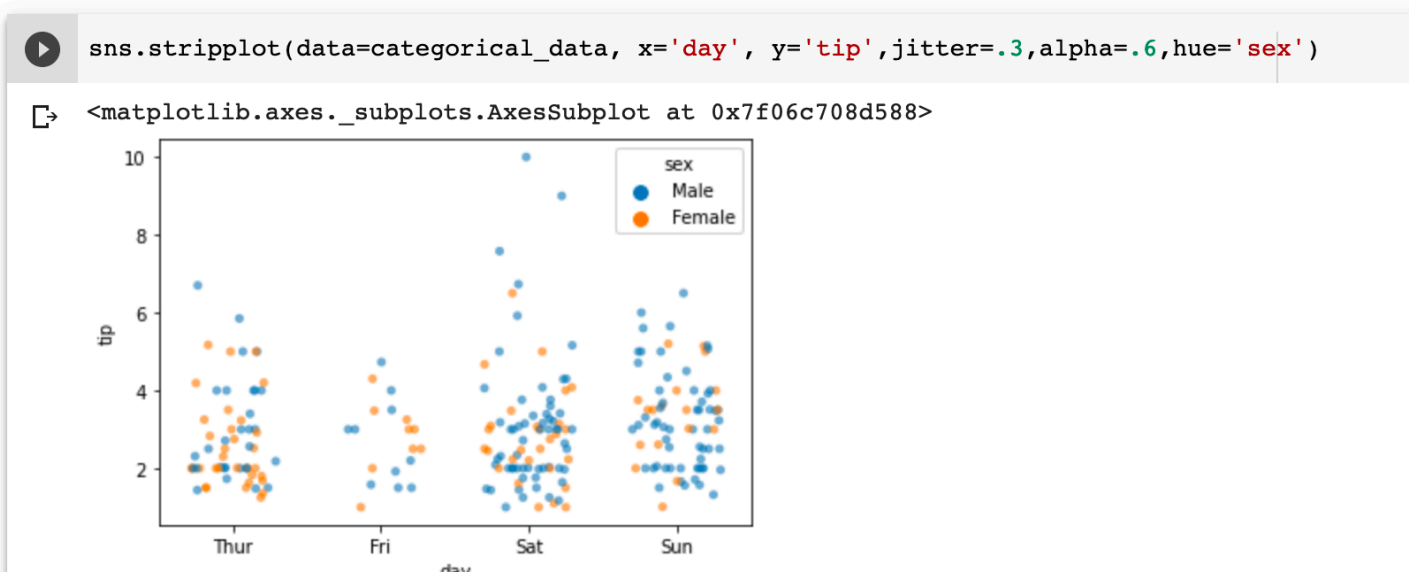
Let’s practice

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



Nested grouping by a two variable

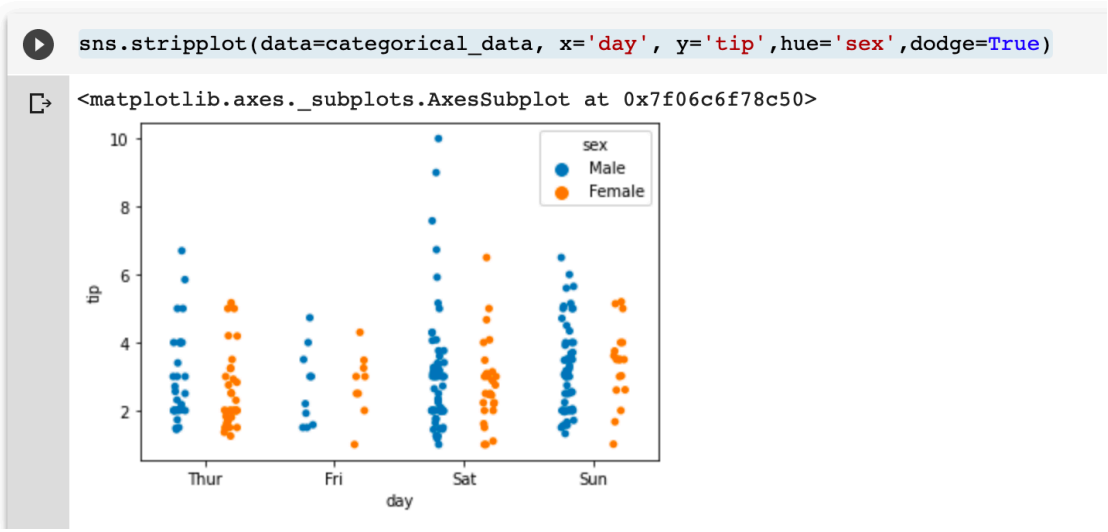
```
sns.stripplot(data=categorical_data, x='day', y='tip',jitter=.3,alpha=.6,hue='sex')
```



Dodge option

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

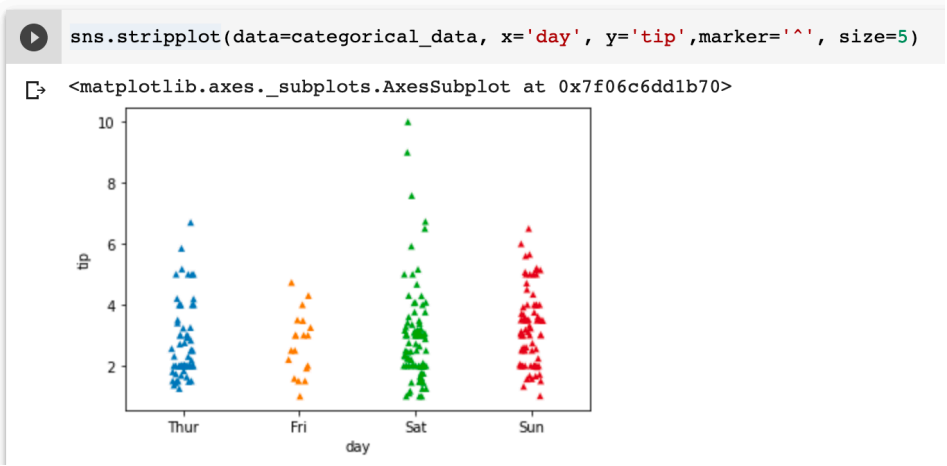
```
sns.stripplot(data=categorical_data, x='day', y='tip',hue='sex',dodge=True)
```



More options for strip plots

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

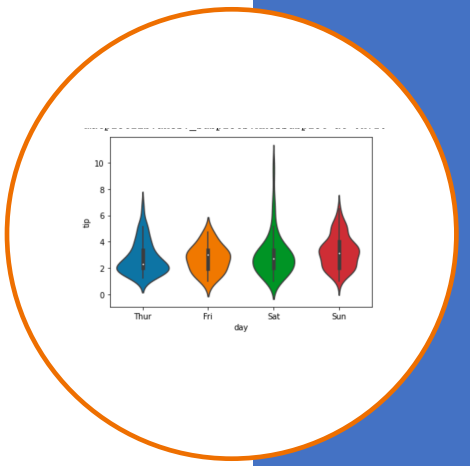
```
sns.stripplot(data=categorical_data, x='day', y='tip',marker='^', size=5)
```



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.
- The difference is particularly useful when the data distribution is multimodal

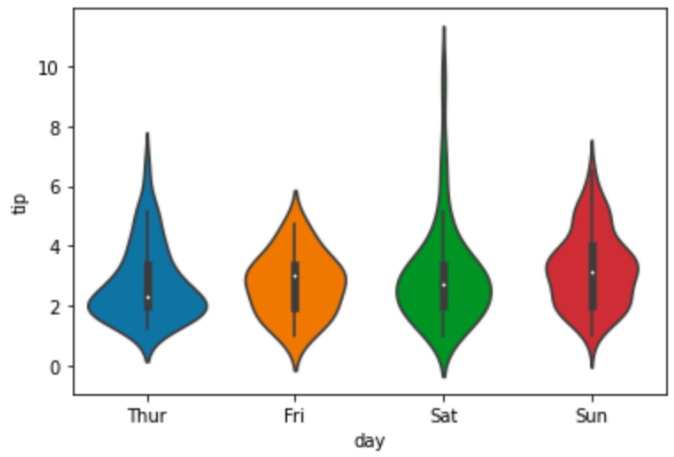
Let’s practice

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

Type of plot

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

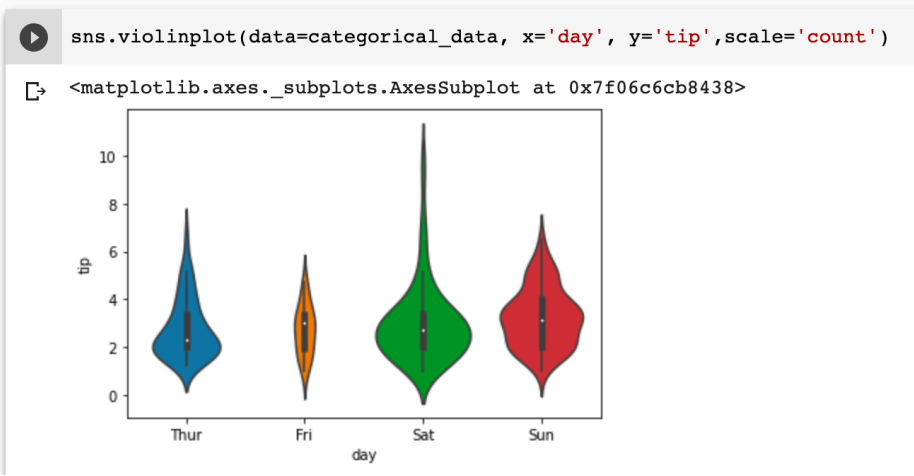
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f06c6d30f60>
```



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

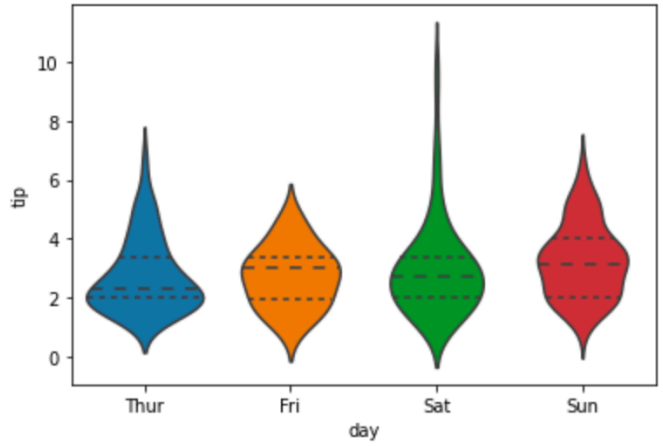
Let’s practice

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

affects what the representation inside the violin

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

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f06c6b40780>
```

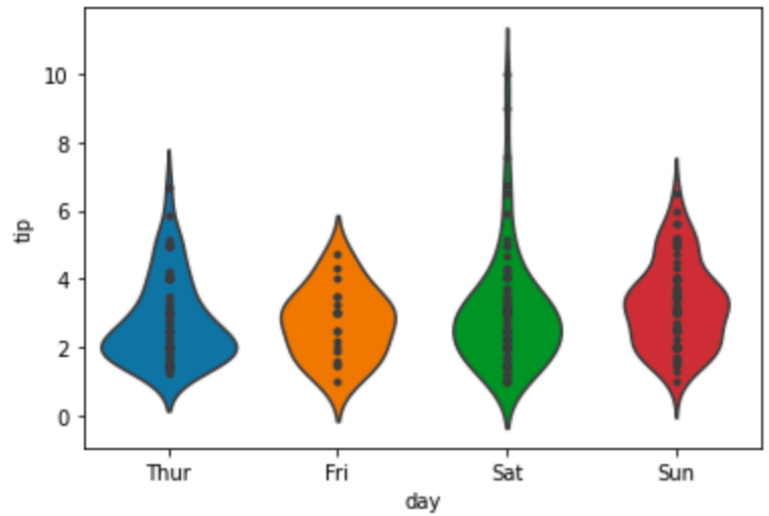


Let’s practice

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

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

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f06c67fbd30>
```

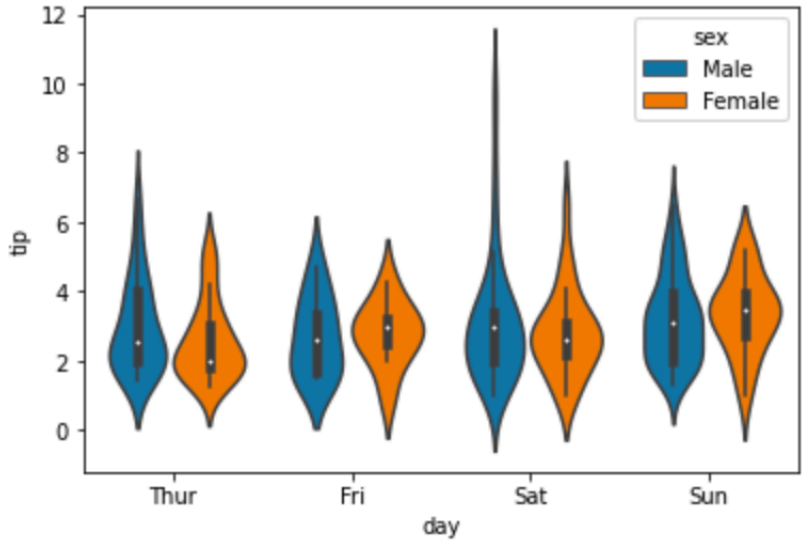


Violin plot with nested grouping by a two variable

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

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

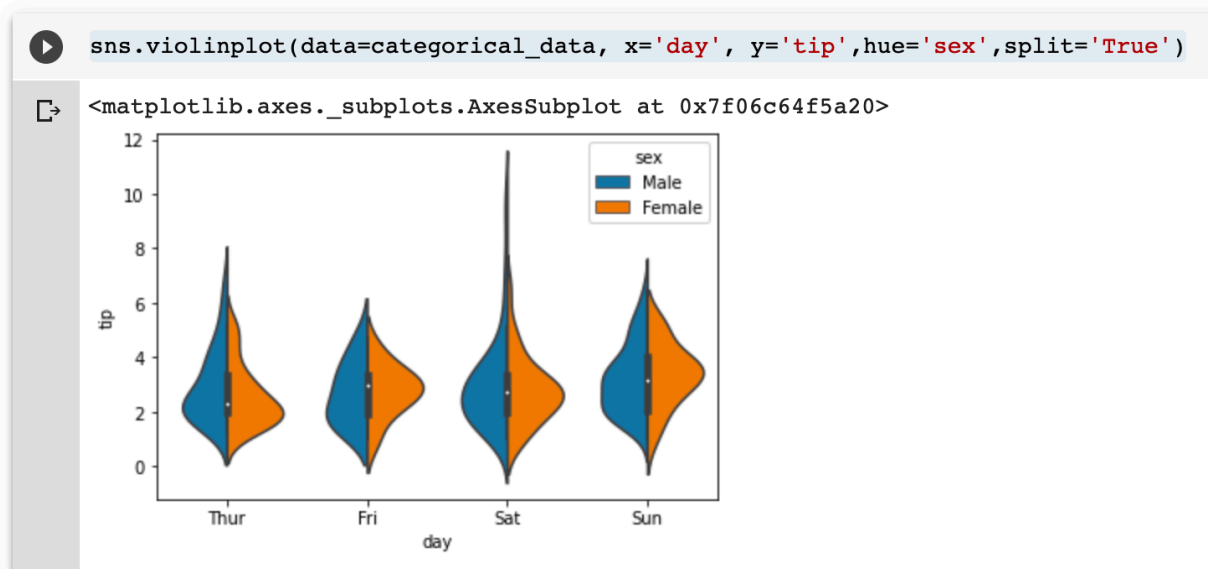
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f06c6759b70>
```



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

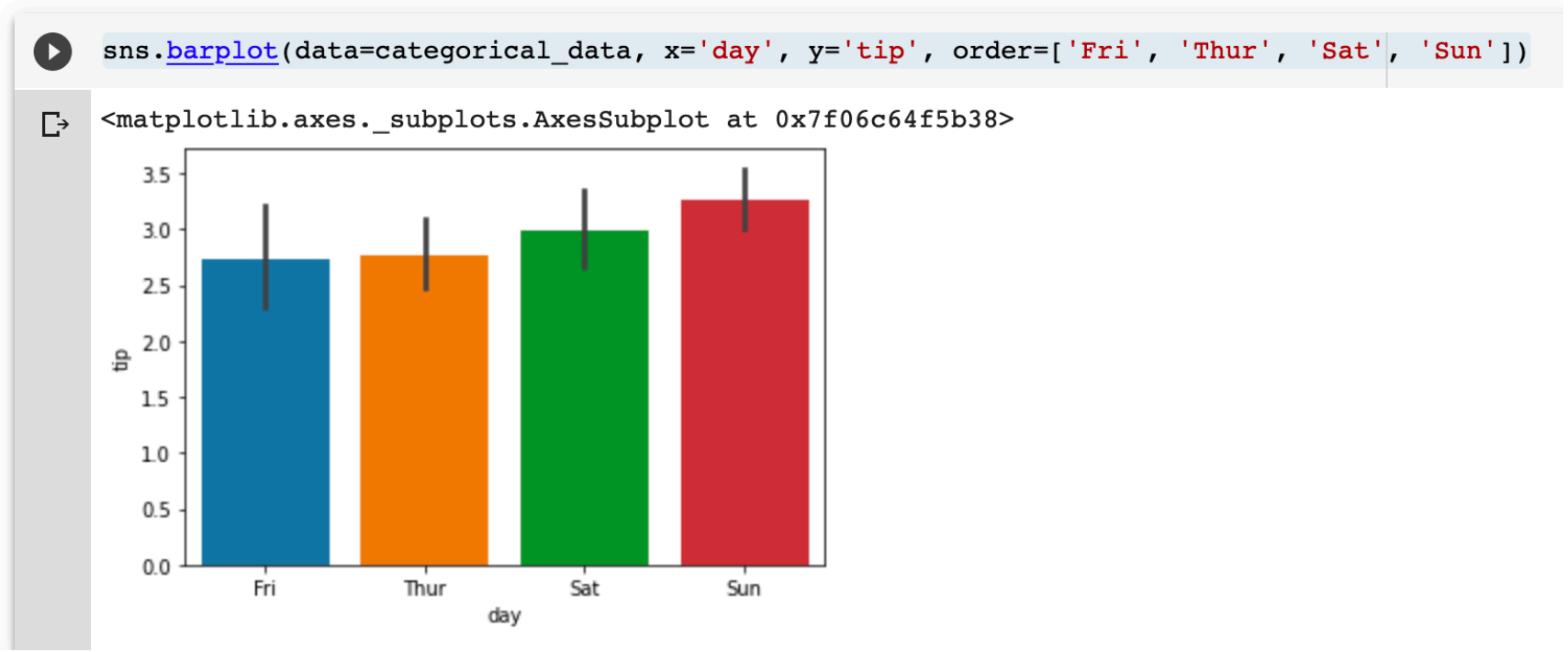
By default, levels inferred from the data

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'])
```



Define a list and use it across all plots

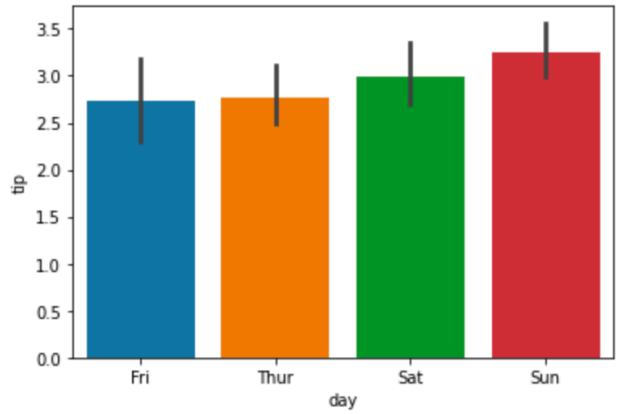
```
x_order = ['Fri', 'Thur', 'Sat', 'Sun']
```

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

```
[47] x_order = ['Fri', 'Thur', 'Sat', 'Sun']

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

<matplotlib.axes._subplots.AxesSubplot at 0x7f06c64919e8>
```



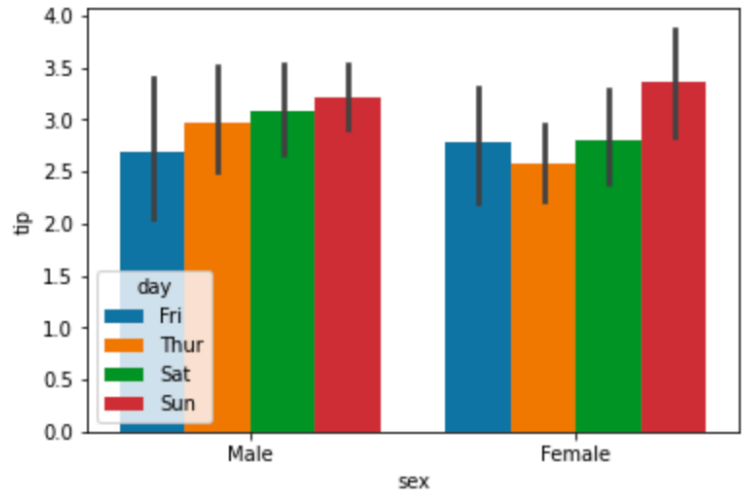
Ordering plot elements

- The hue_order parameter affects the hue categories

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

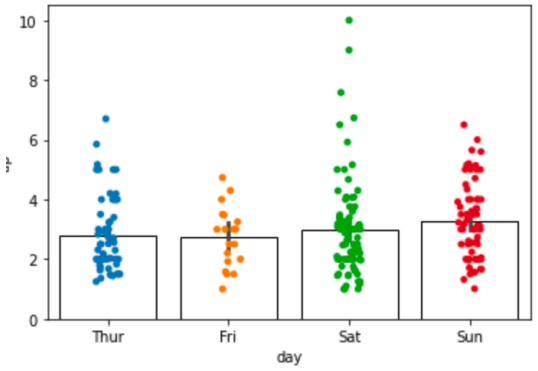
```
sns.barplot(data=categorical_data, x='sex', y='tip', hue_order=x_order,hue='day')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f06c61650b8>
```



Combining categorical plots

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

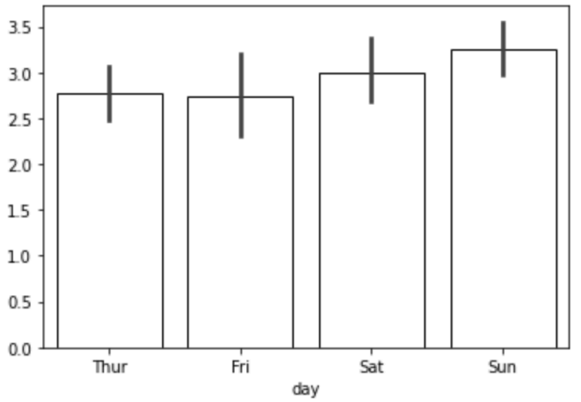


A bar plot and strip plot combine the point distribution with an estimator

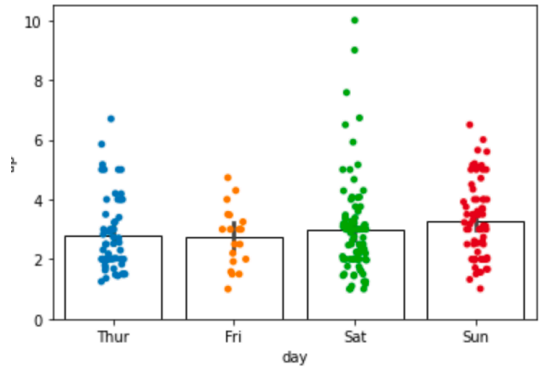
Let’s practice

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

Color inside the bar



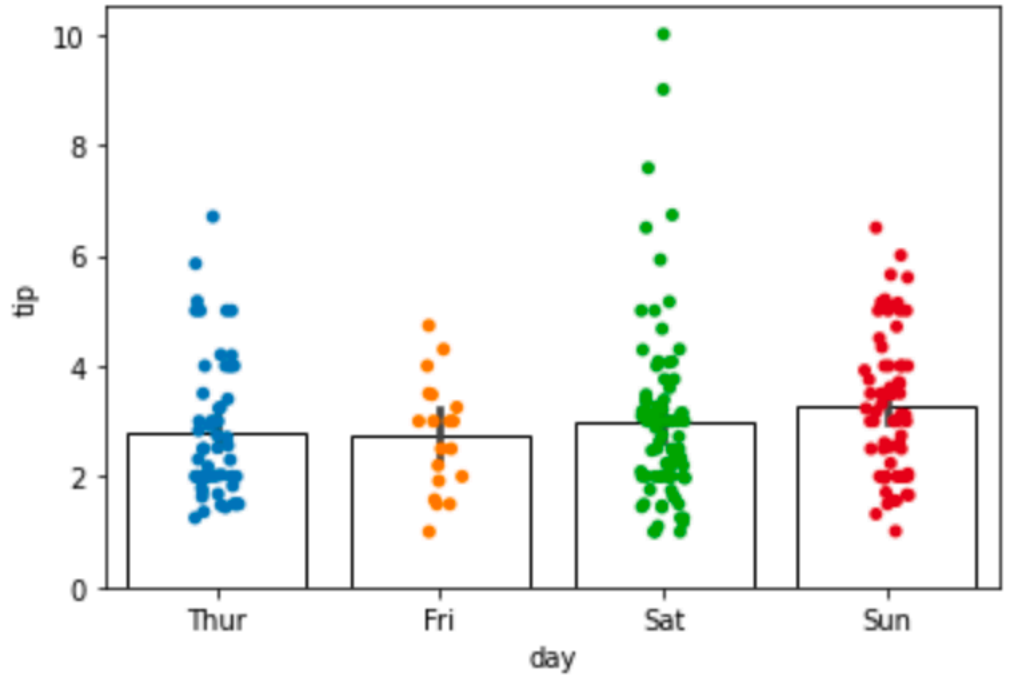

```
sns.stripplot(data=categorical_data,
x='day', y='tip')
```



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)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f06c567fa90>
```