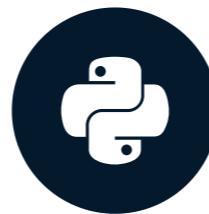


# Point estimate intervals

IMPROVING YOUR DATA VISUALIZATIONS IN PYTHON

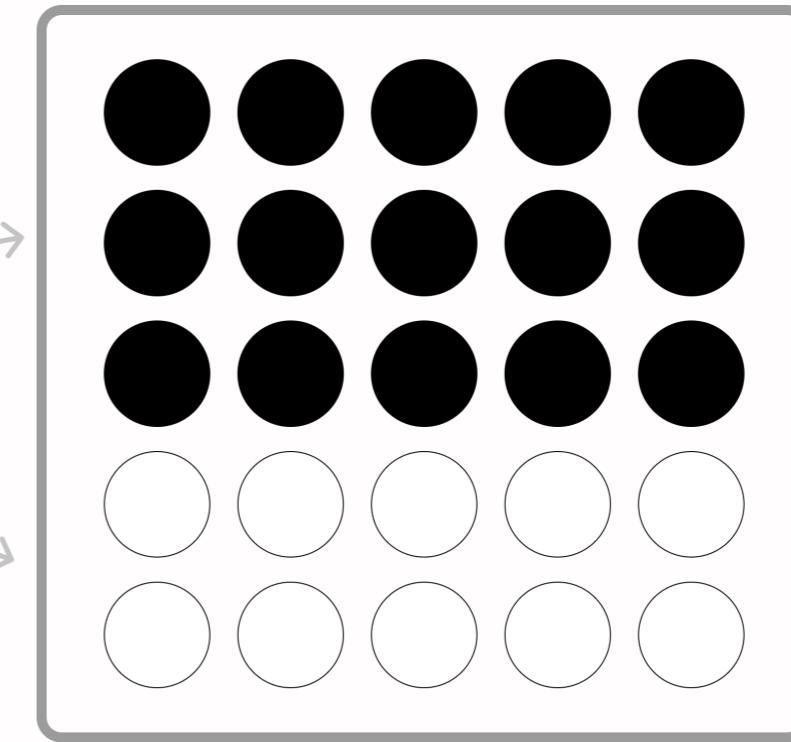


**Nick Strayer**  
Instructor

# What is uncertainty?

Population of interest:  
E.g. All sheep on farm

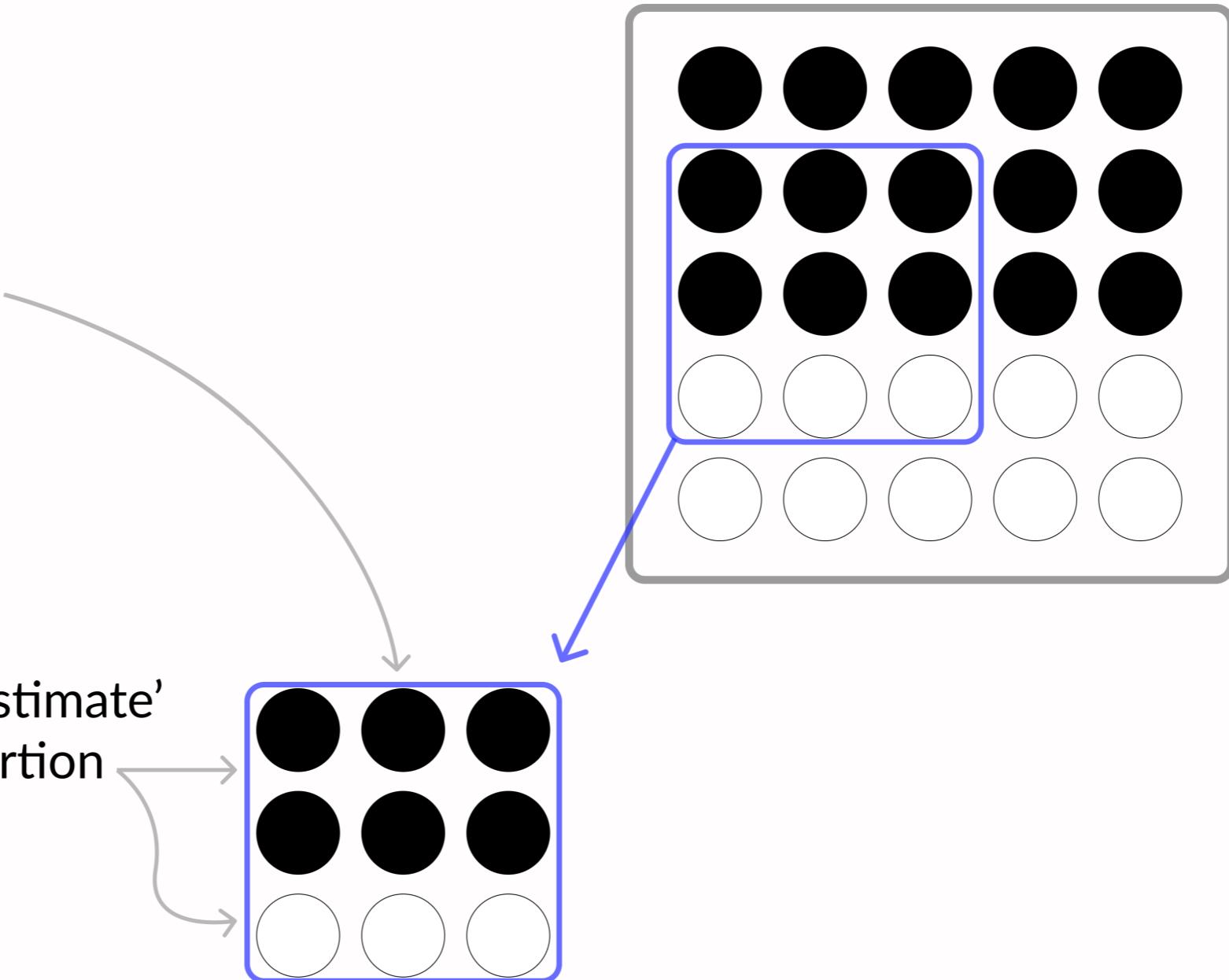
True proportion black to white sheep



# What is uncertainty?

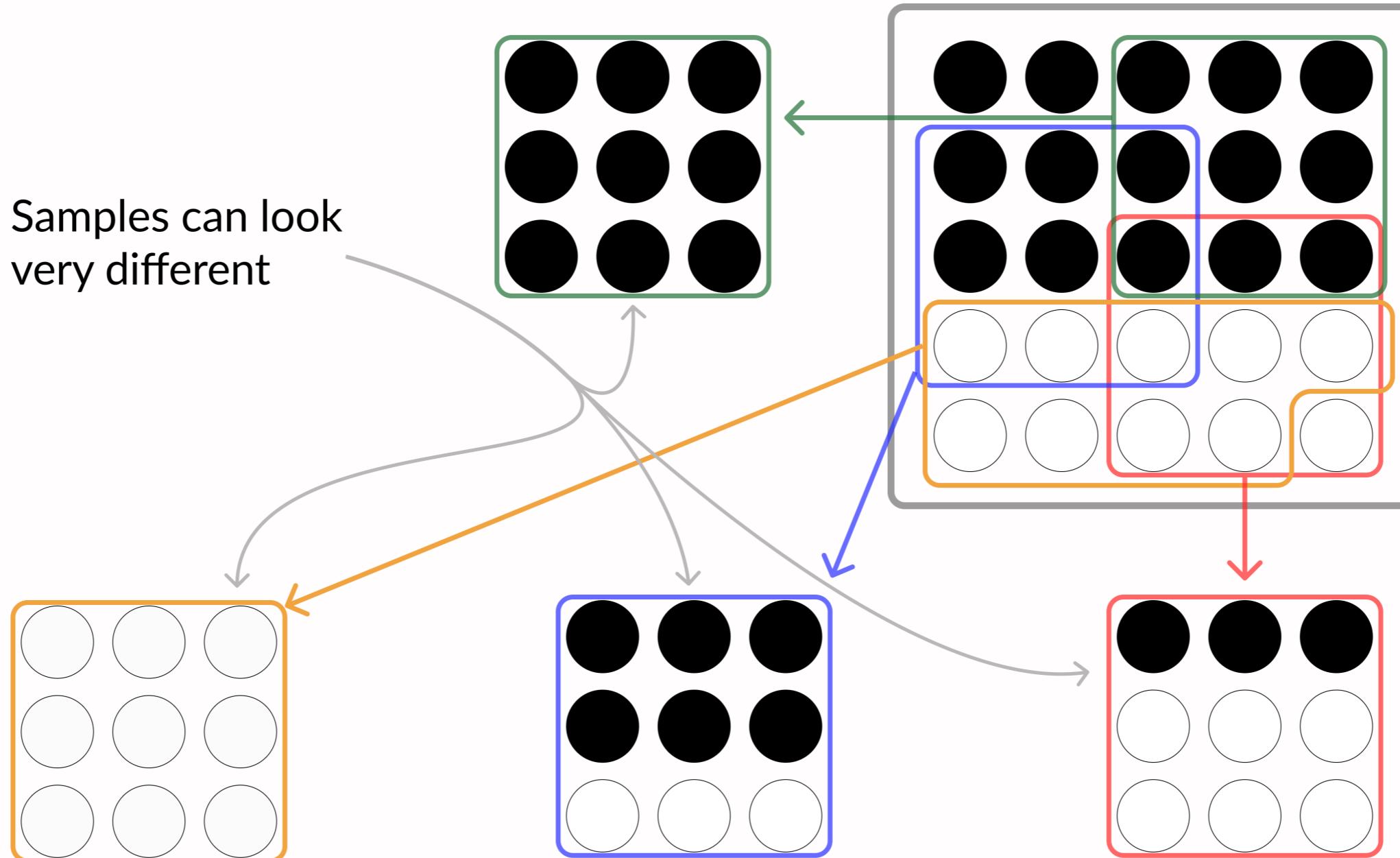
Sample taken  
from population

Used to get an 'estimate'  
of the true proportion



# What is uncertainty?

Samples can look  
very different



# When is uncertainty important?

## Needs uncertainty

- *Estimates from sample*
  - Average of a subset
  - Linear model coefficients

## Doesn't need uncertainty

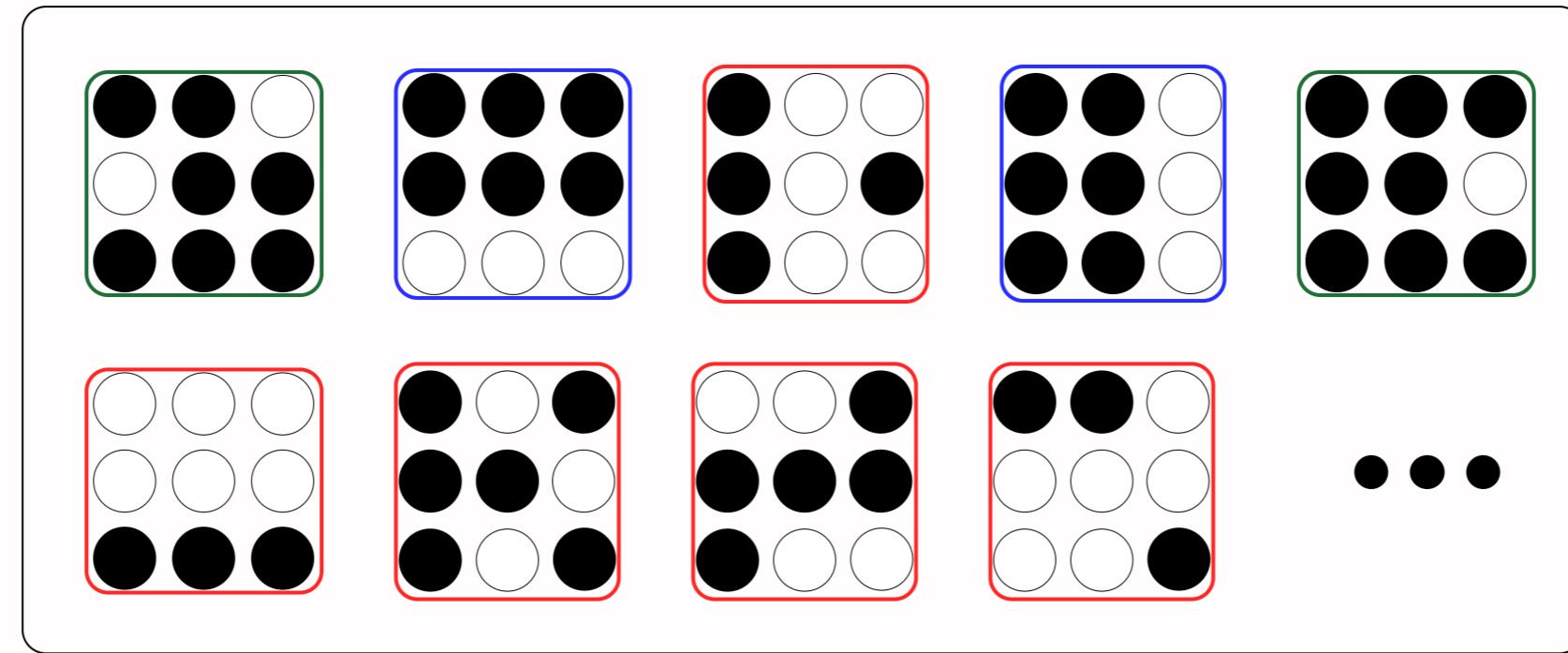
- *Facts*
  - Counts
  - Summaries of an entire population

# Why is uncertainty important?

- Helps inform confidence in estimate
- Necessary for decision making
- Acknowledges limitations of data

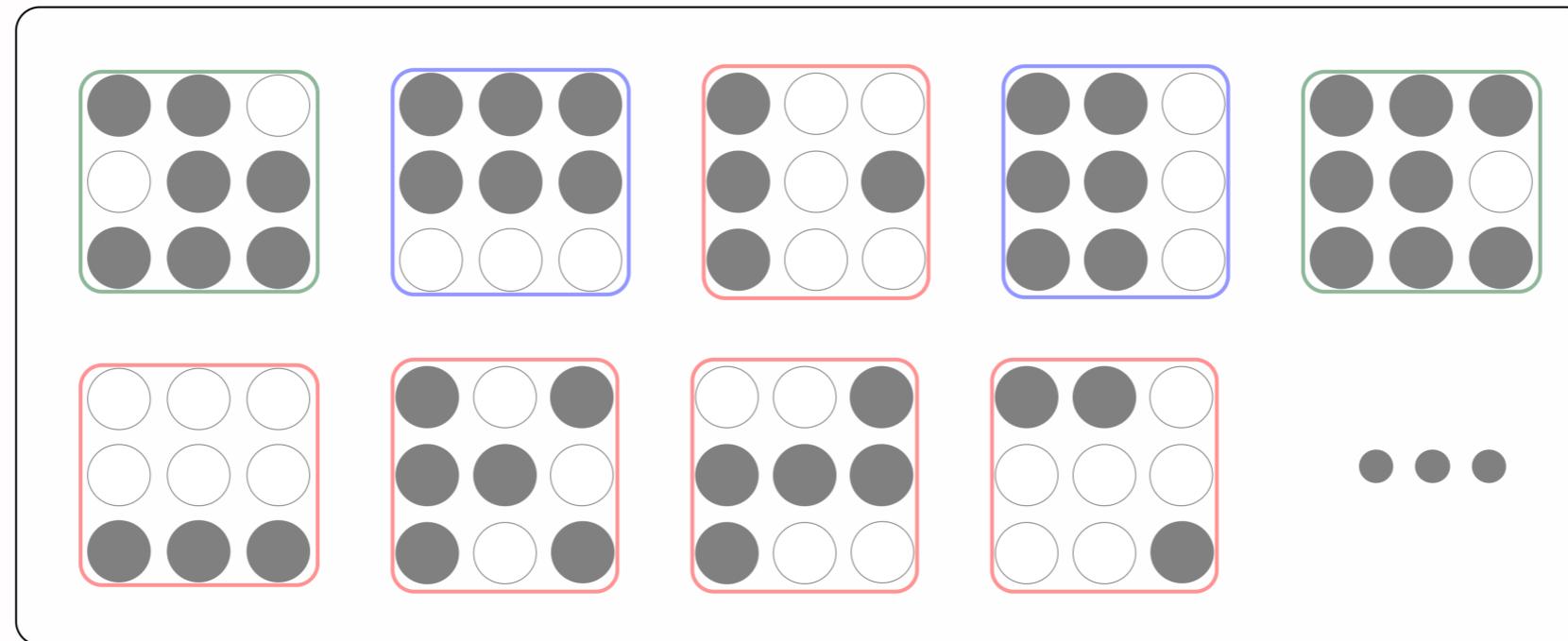


# The Confidence Interval

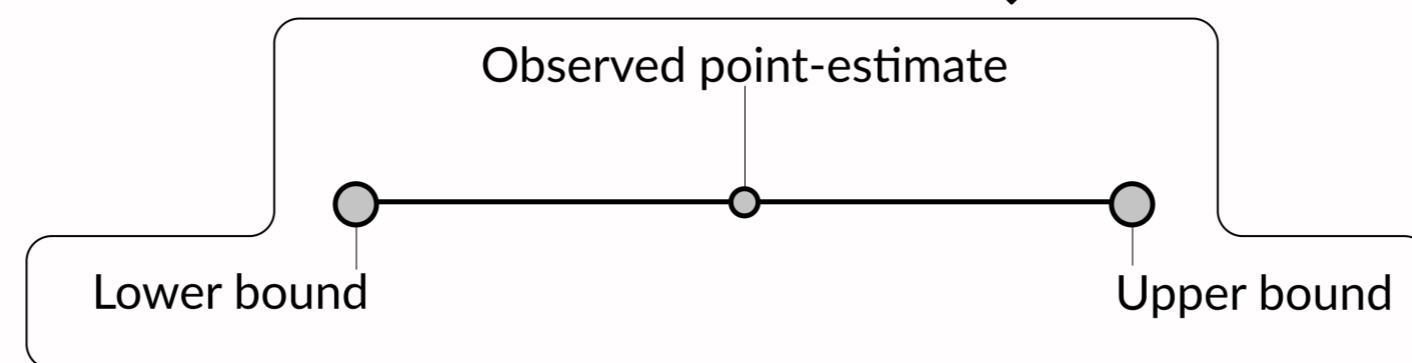


Pretend you could take infinite resamples of your data

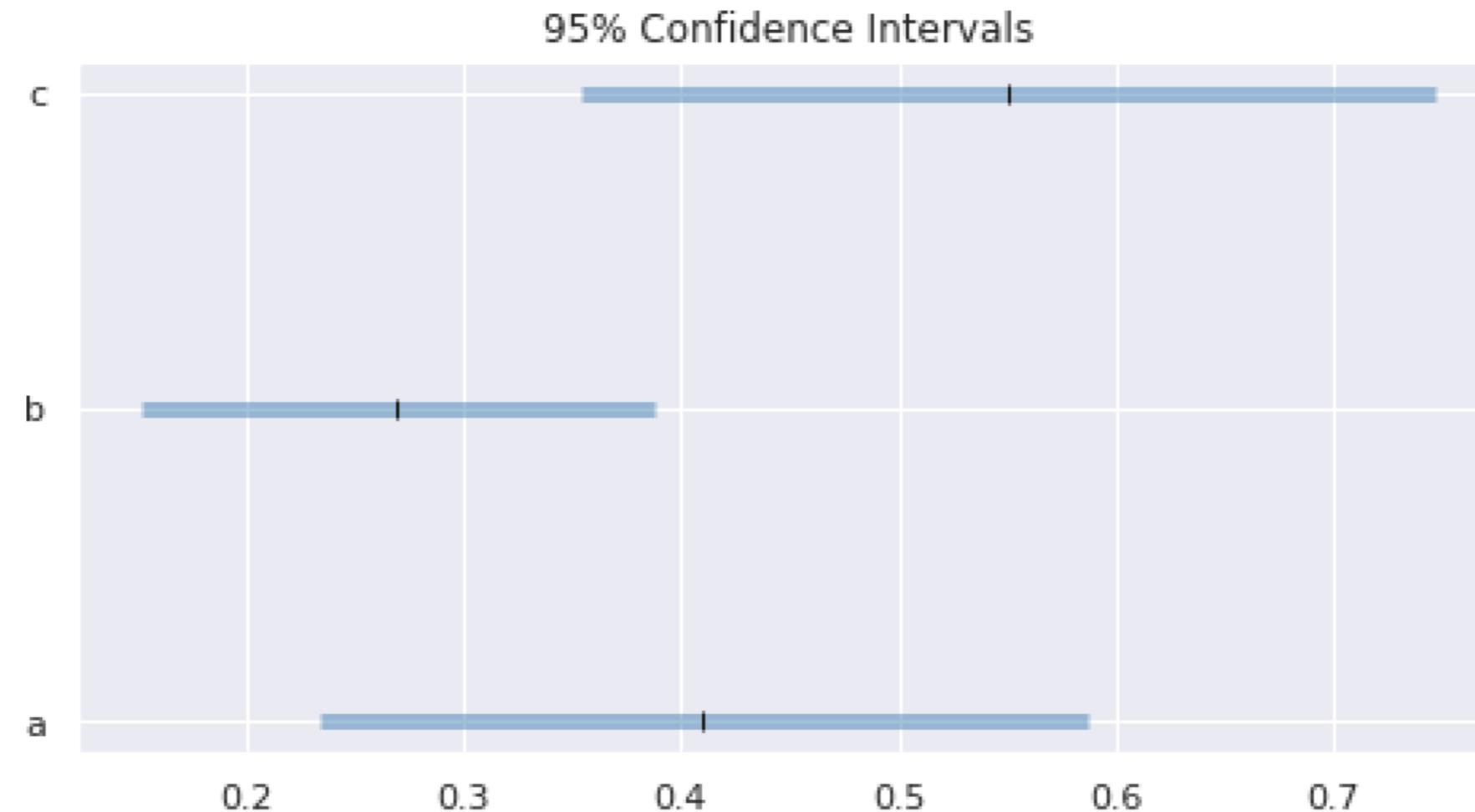
# The Confidence Interval



**95%** of estimates from the resamples will fall inside the **95% confidence interval**



```
plt.hlines(xmin='lower', xmax='upper', y='y', data=data,  
          linewidth=5, color='steelblue', alpha=0.5)  
  
# Point-estimate for reference  
plt.plot('est', 'y', 'k|', data = data)
```

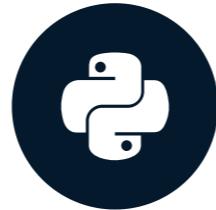


# **Let's show some uncertainty!**

**IMPROVING YOUR DATA VISUALIZATIONS IN PYTHON**

# Confidence bands

IMPROVING YOUR DATA VISUALIZATIONS IN PYTHON

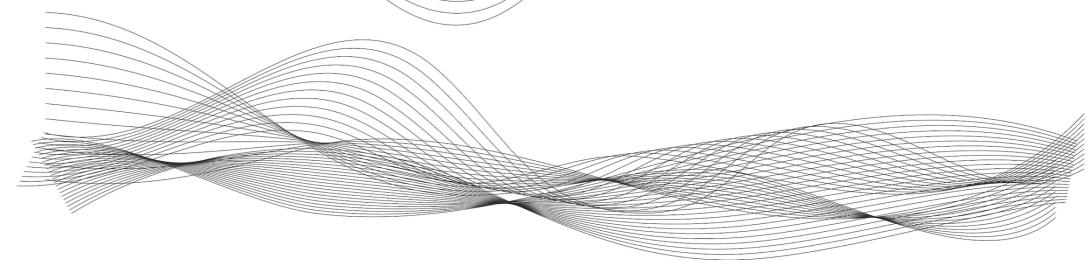
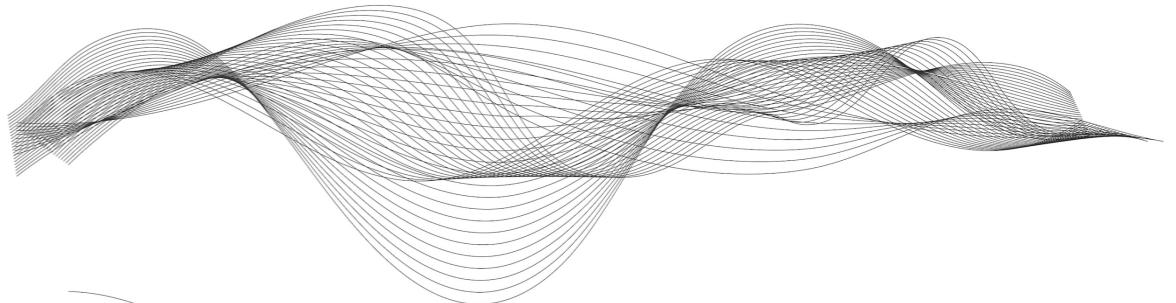
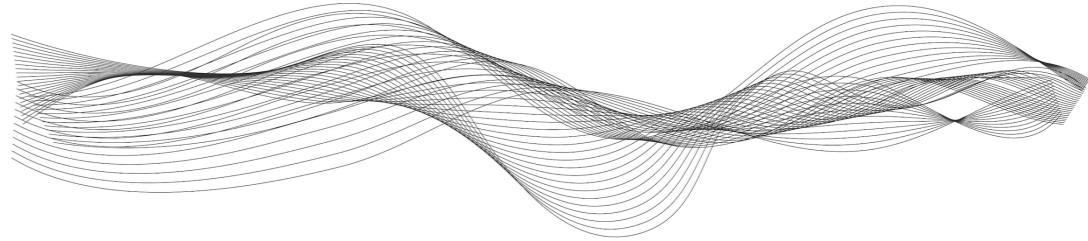
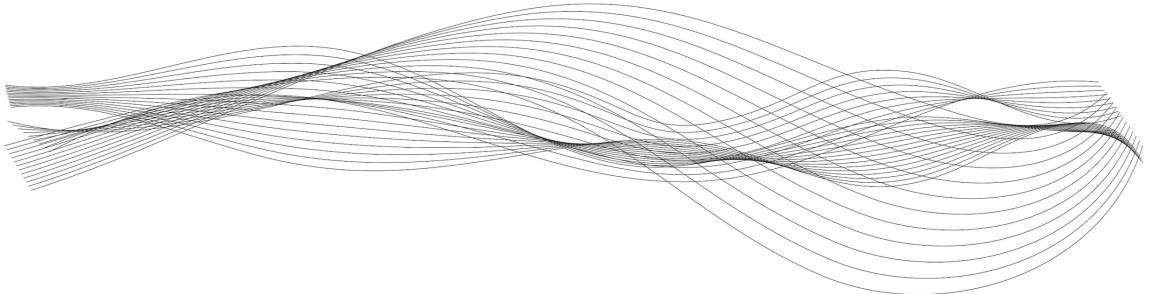


Nick Strayer

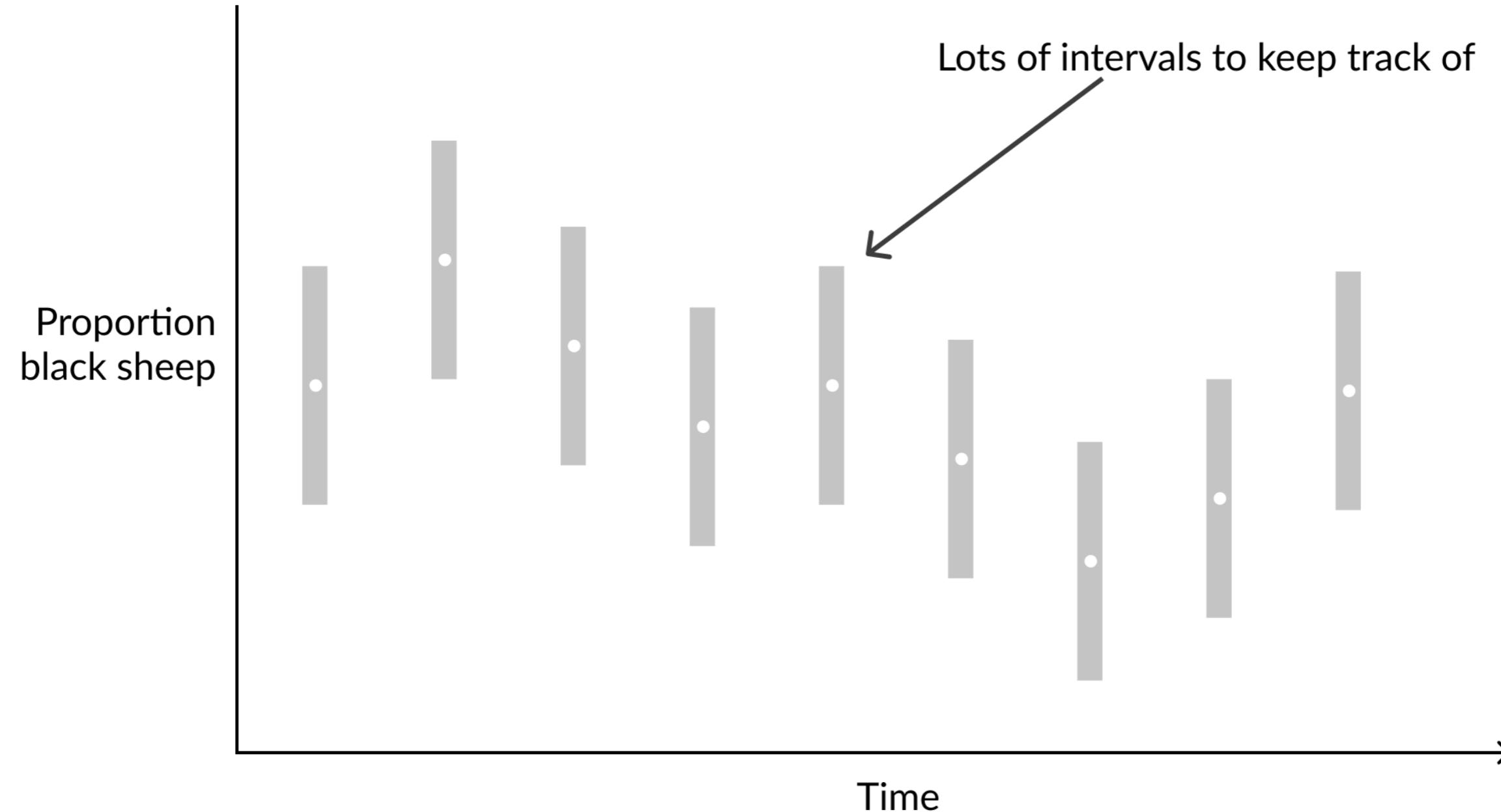
Instructor

# Continuous estimation functions

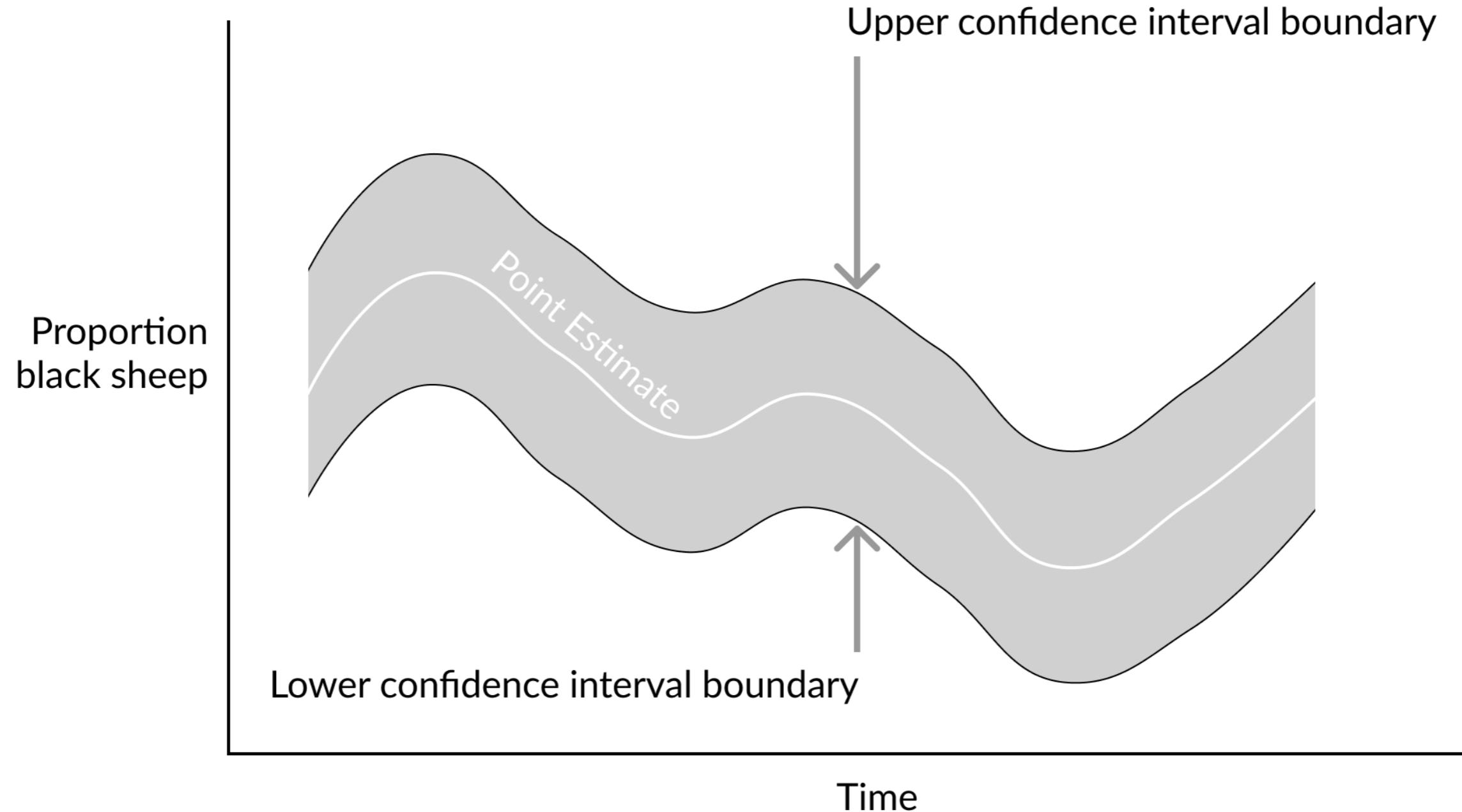
- Estimates over a continuous axis
- Often over time
- Have uncertainty and should be plotted as such.



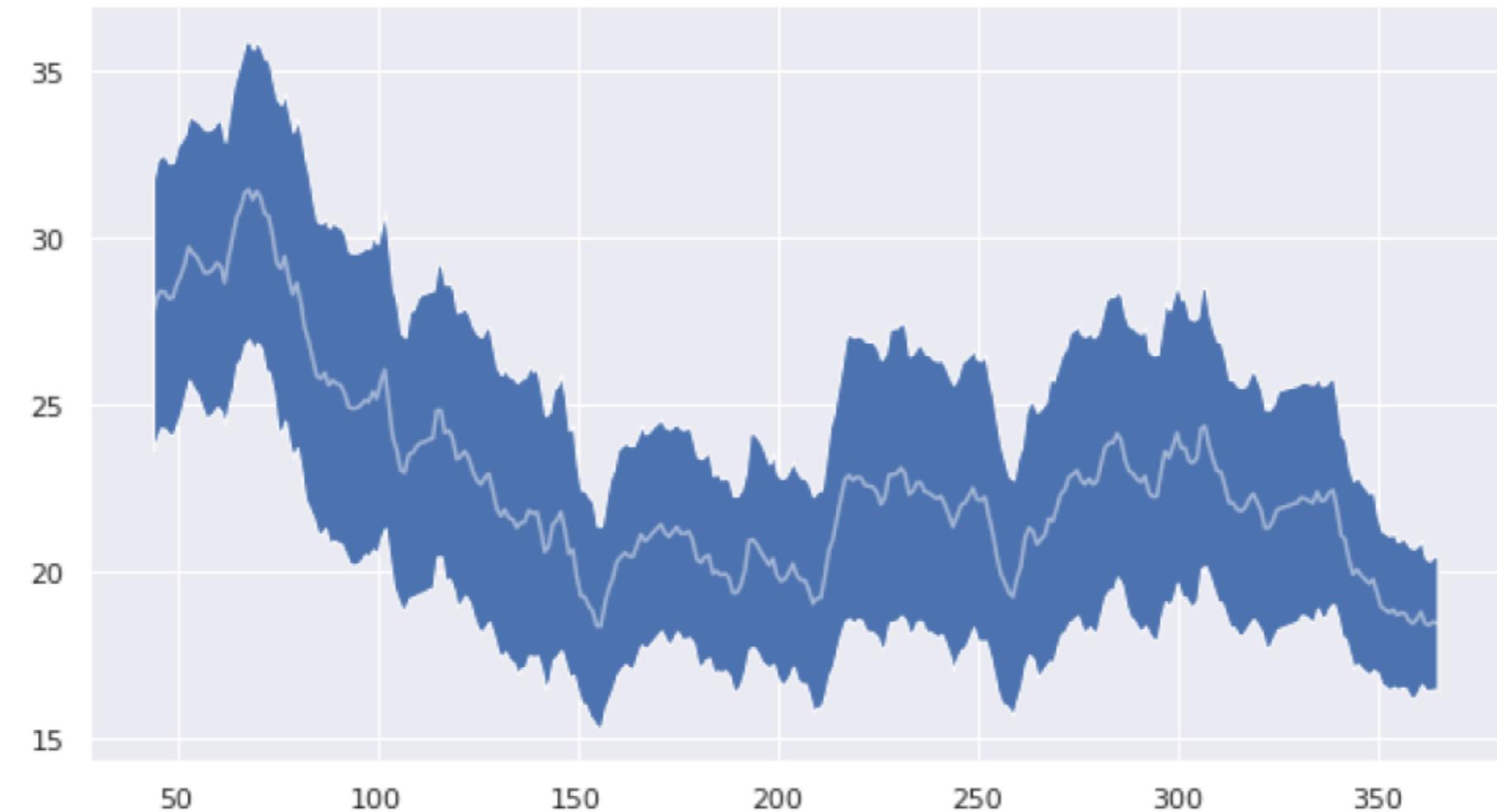
## Lots of confidence intervals



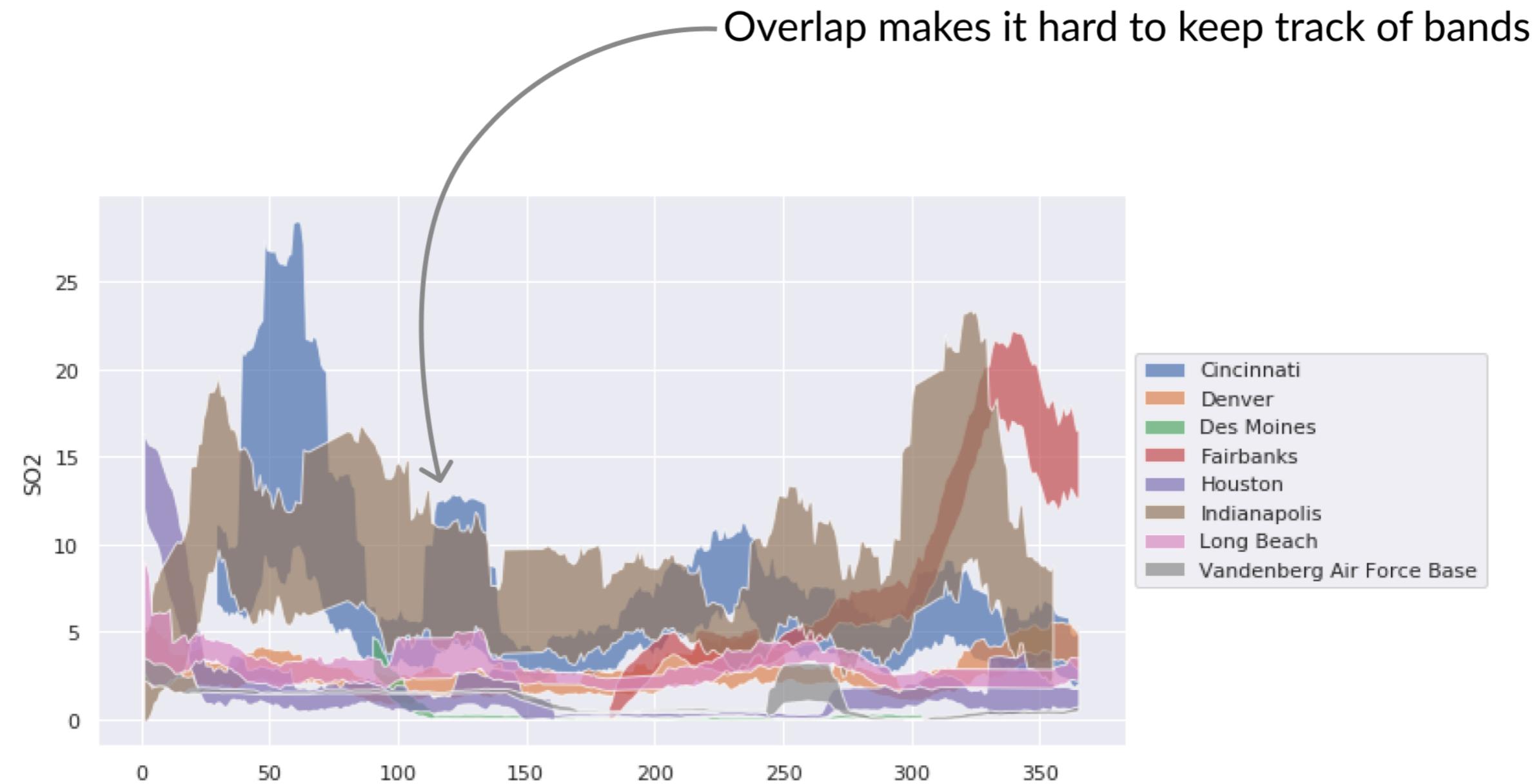
# The confidence band



```
# Set lower and upper bounds of ribbon
plt.fill_between(x='day', y1='lower', y2='upper', data=cinci_so2)
# Add point-estimate reference line
plt.plot('day', 'mean', 'w-', alpha=0.5, data=data)
```



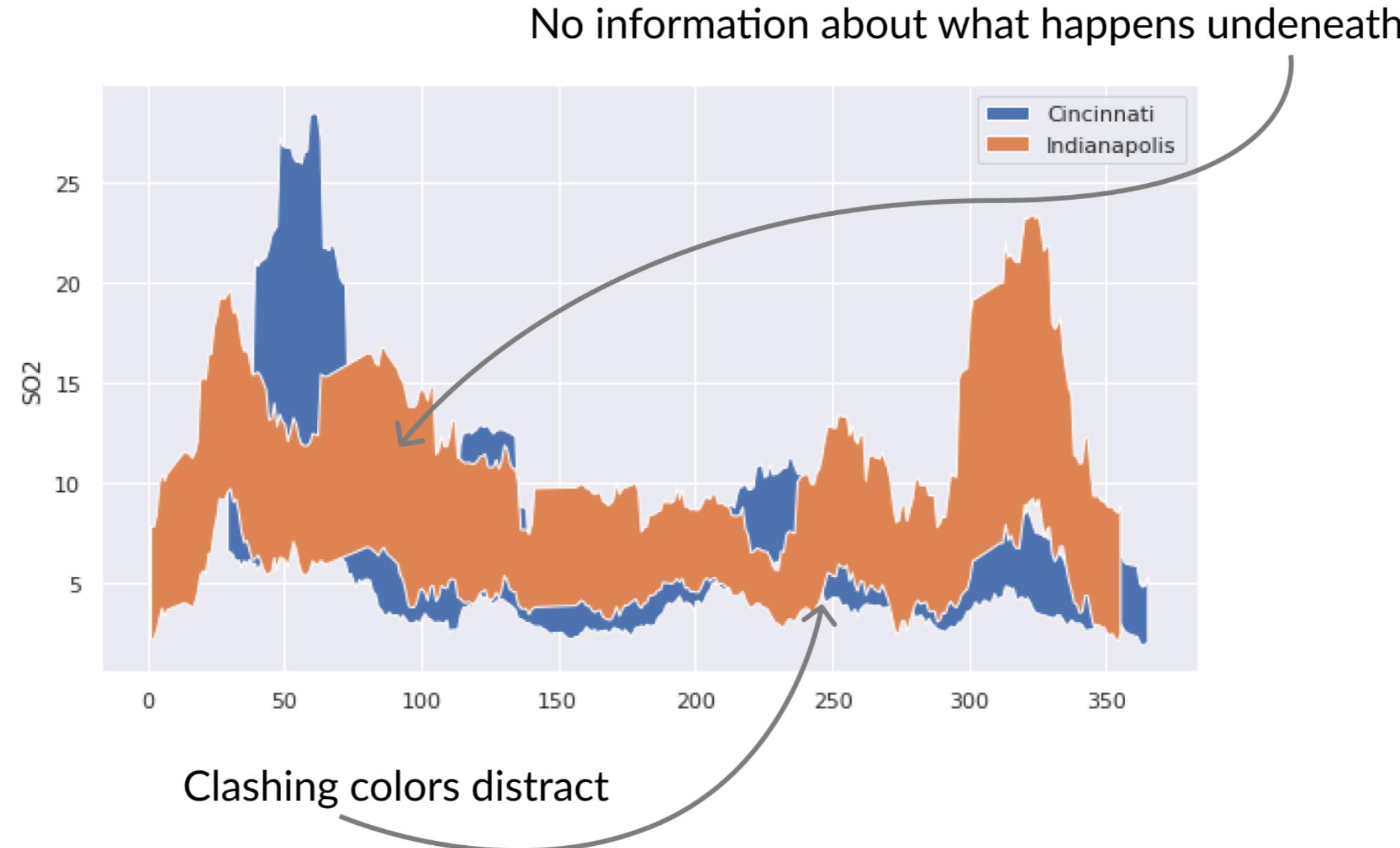
# Separate your bands



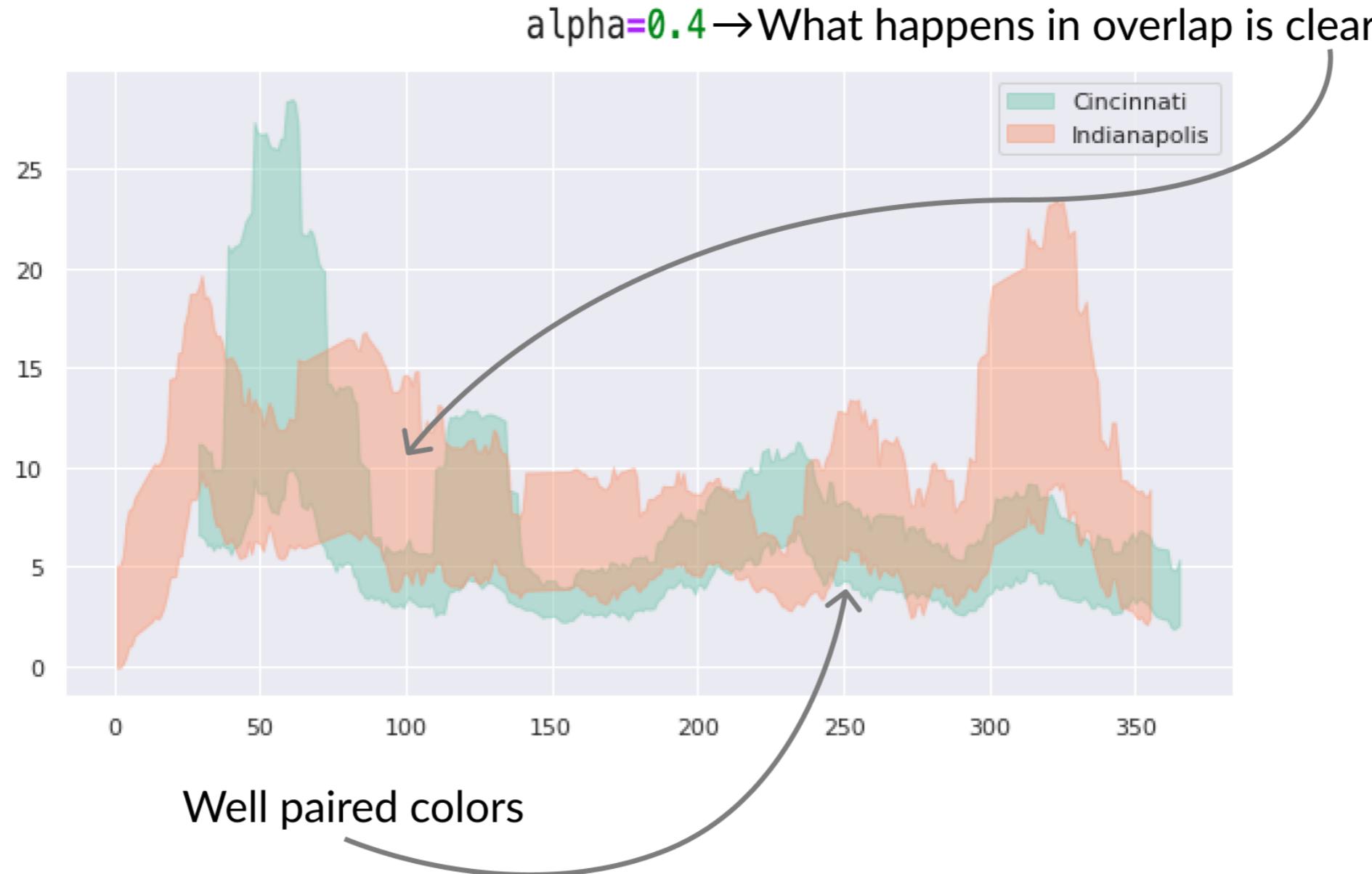
## Separate your bands



# Cleaning up confidence band comparisons



# Cleaning up confidence band comparisons

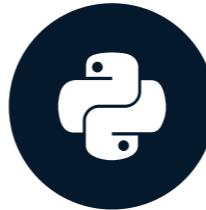


# **Let's draw some bands!**

**IMPROVING YOUR DATA VISUALIZATIONS IN PYTHON**

# Beyond 95%

IMPROVING YOUR DATA VISUALIZATIONS IN PYTHON

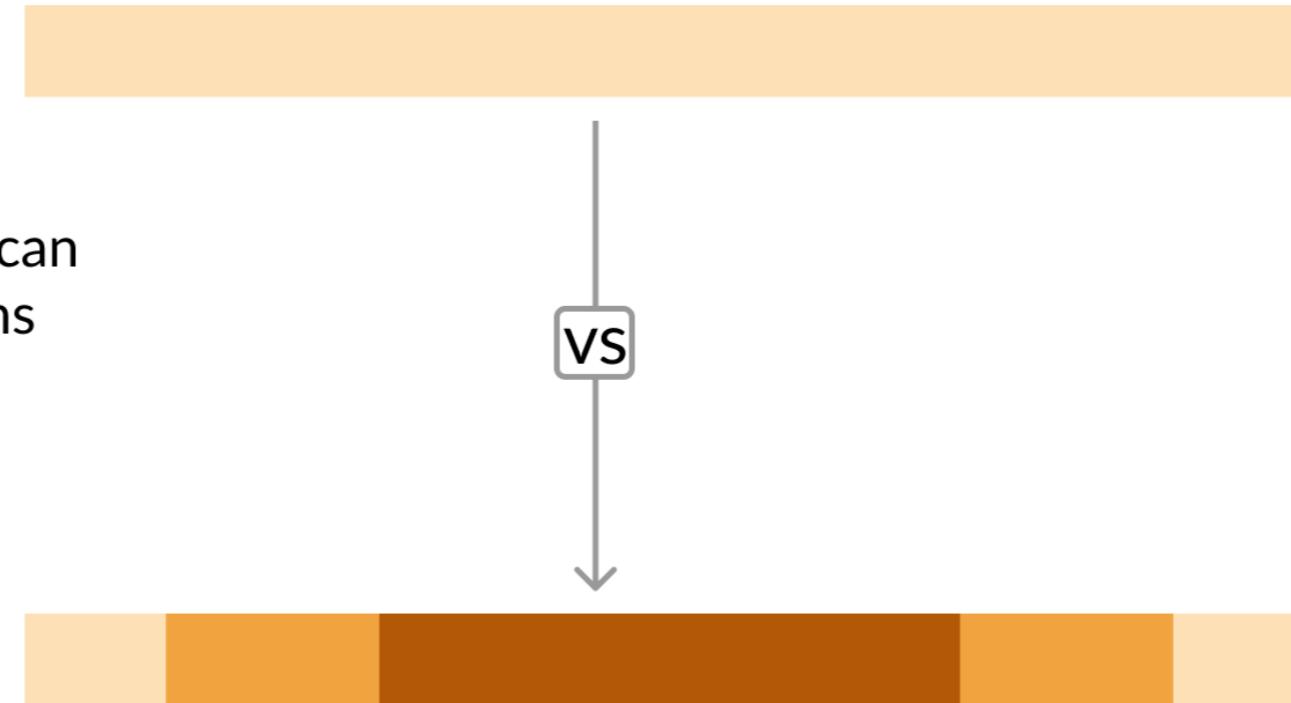


**Nick Strayer**

Instructor

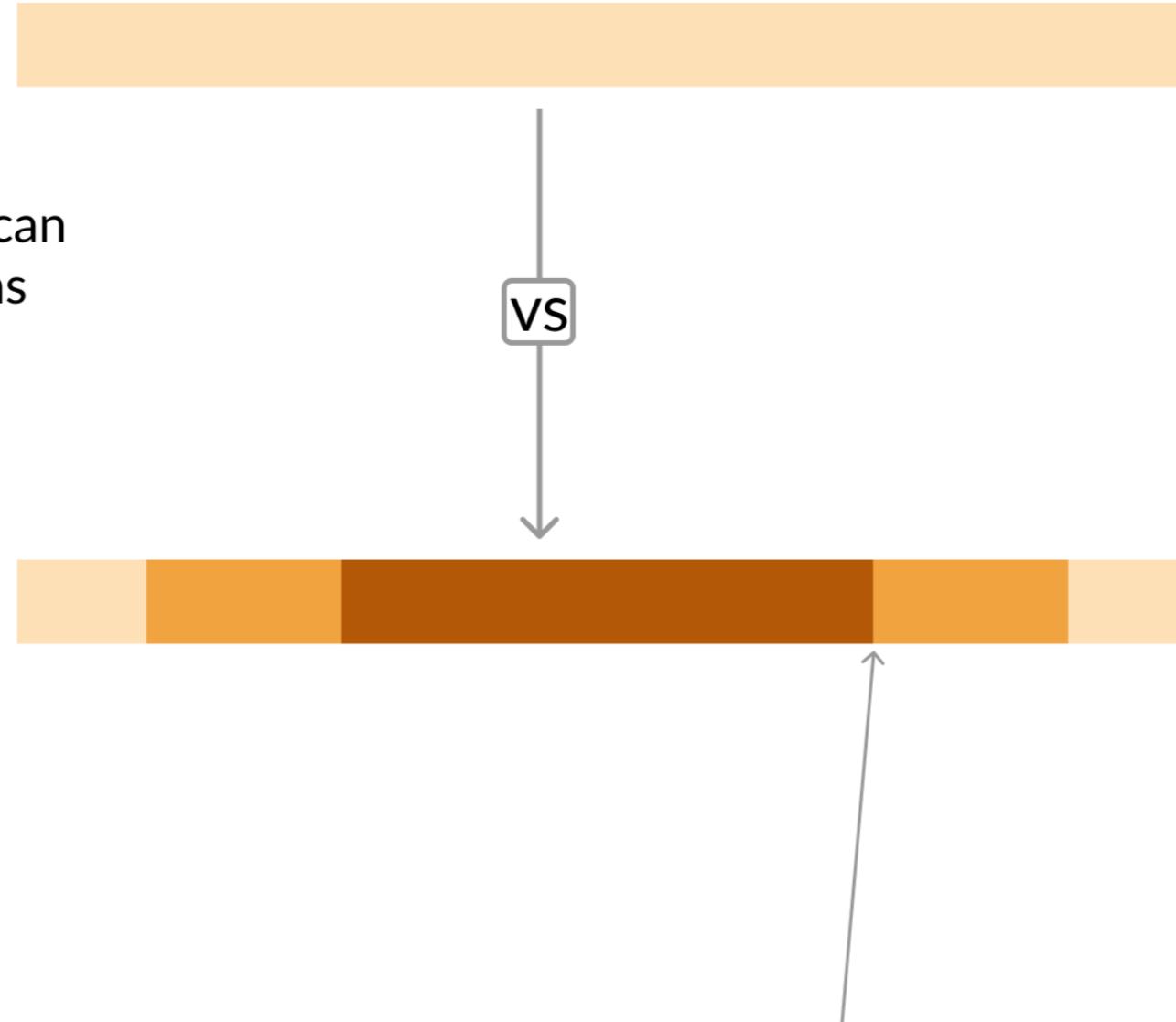
## Why show more than one interval?

Different interval widths can answer different questions



## Why show more than one interval?

Different interval widths can answer different questions



## Why show more than one interval?

Different interval widths can answer different questions

VS



Need a standard 95%?

Is a rough 90% good enough?

# Why show more than one interval?

Different interval widths can answer different questions

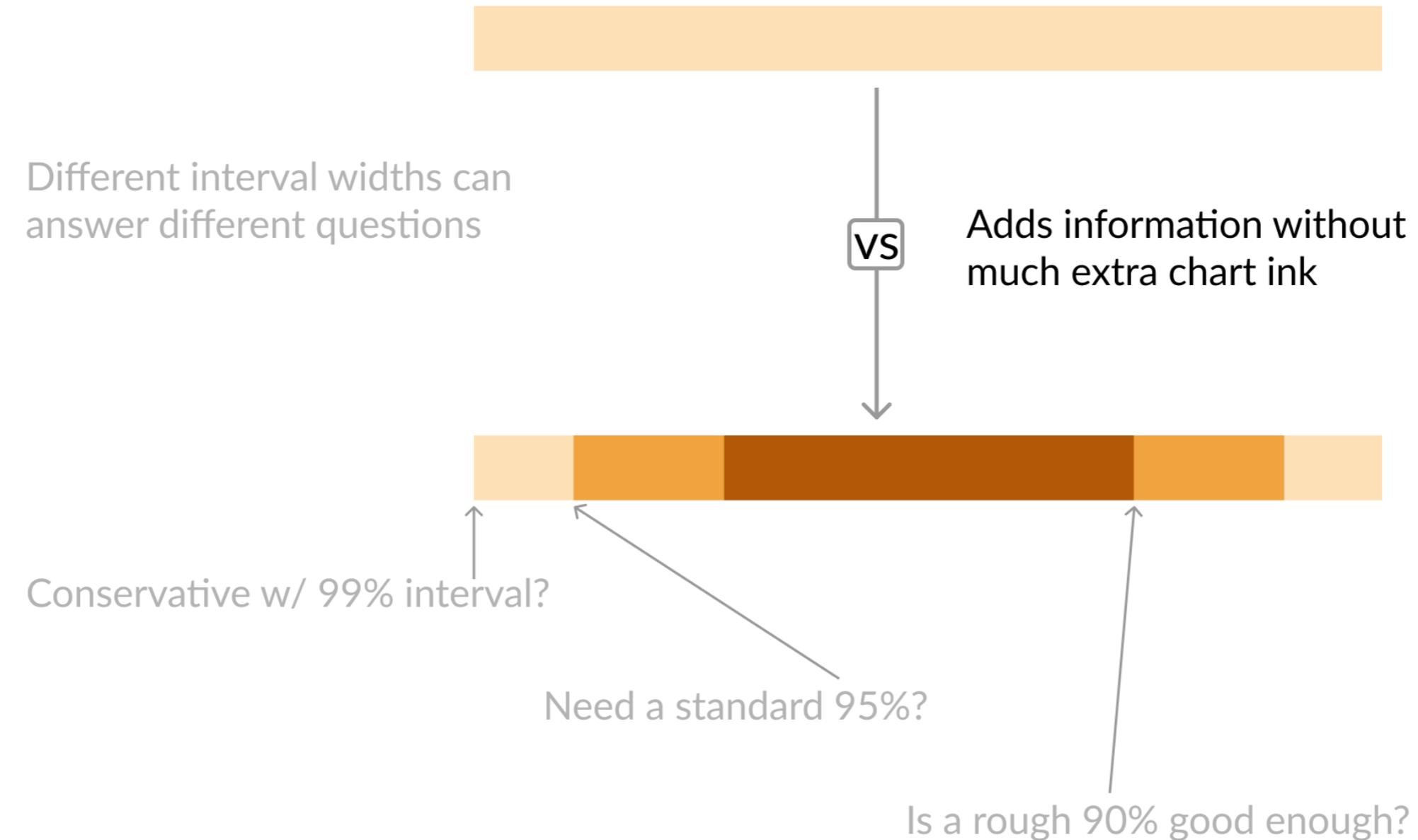
VS

Conservative w/ 99% interval?

Need a standard 95%?

Is a rough 90% good enough?

## Why show more than one interval?

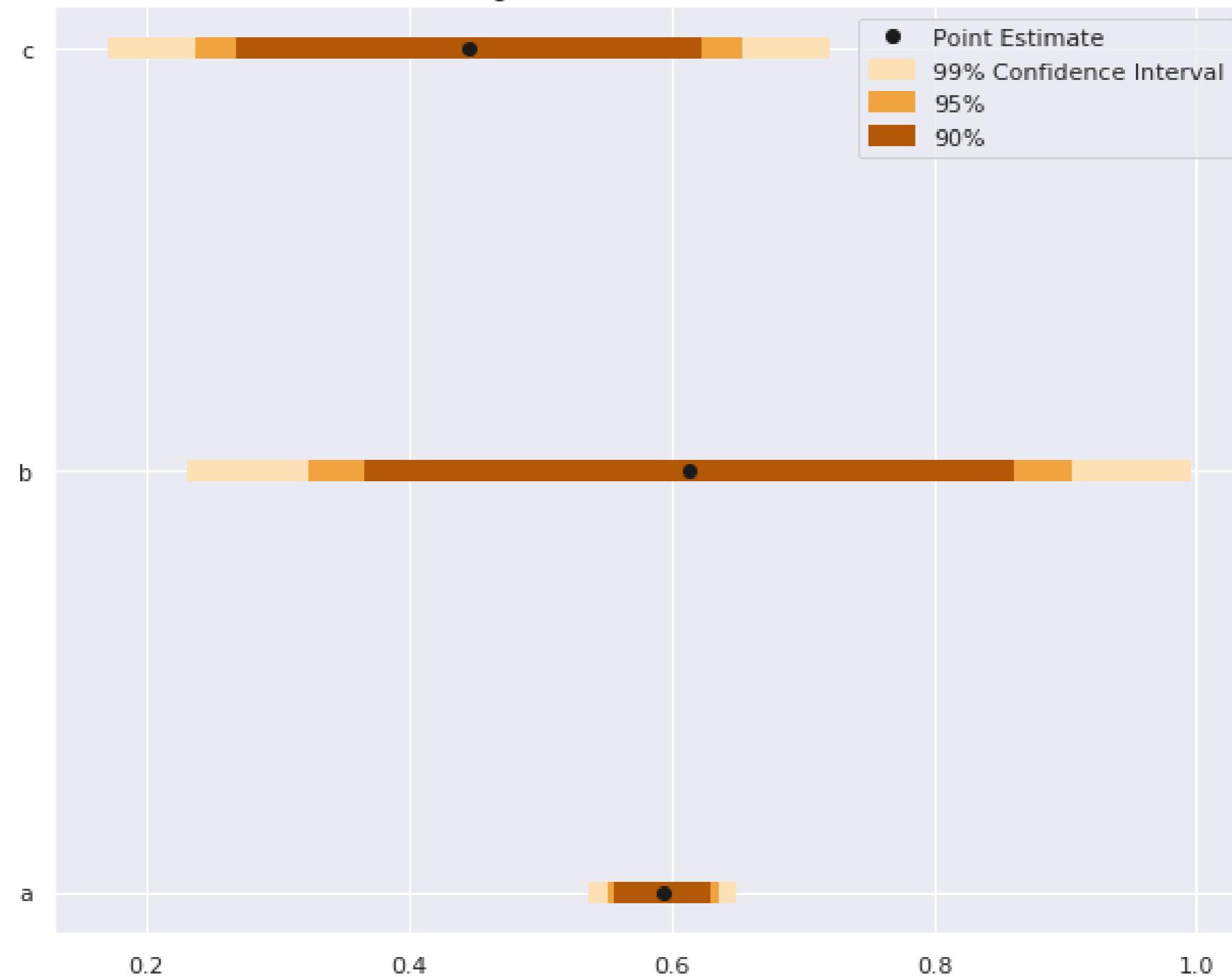


```
# Interval size setup
sizes      = ['99%', '95%', '90%']
Z_scores   = [2.58,    1.96,   1.67]
colors     = ['#fee0b6', '#f1a340', '#b35806']

for size, z, color in zip(sizes, Z_scores, colors):
    plt.hlines(y = data.y,
                # Calculate lower and upper boundaries
                xmin = data['est'] - z*data['std_err'],
                xmax = data['est'] + z*data['std_err'],
                # Color by interval size
                color = color,
                # Make line thicker for visibility
                linewidth = 7,
                # Label line so legend text is clear
                label = size)

plt.plot('est', 'y', 'ko', data = data, label = 'Point Estimate')
plt.legend()
```

### Point Estimate along with 99,95, and 90% Confidence Intervals



# Coloring your intervals

Lighter colors on inside create illusion of two intervals



Colors too similar make reading difficult

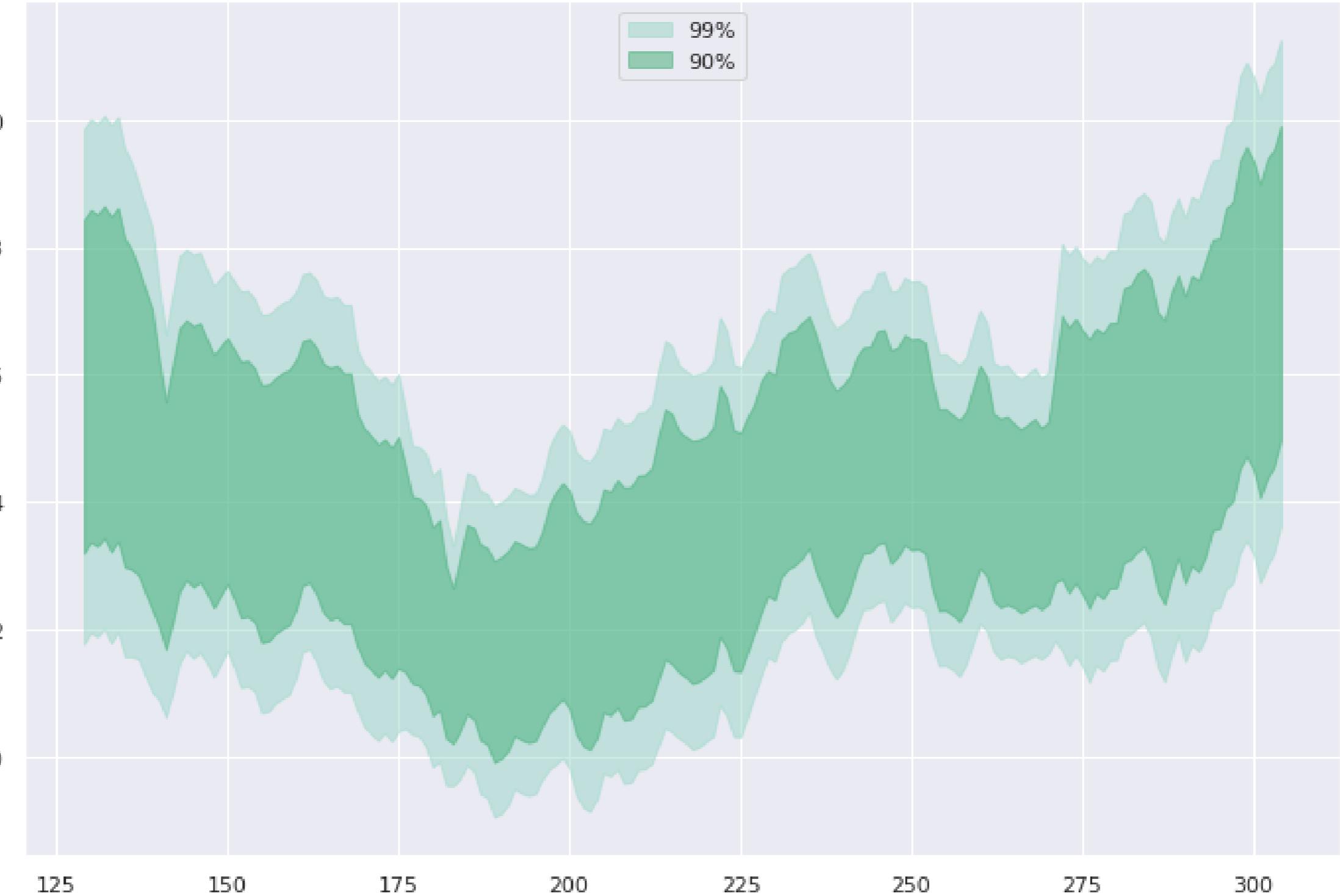


Well separated ordinal palette with darker colors inside



```
widths      = [      '99%',      '90%']
z_scores   = [      2.58,      1.67]
colors     = ['#99d8c9', '#41ae76']

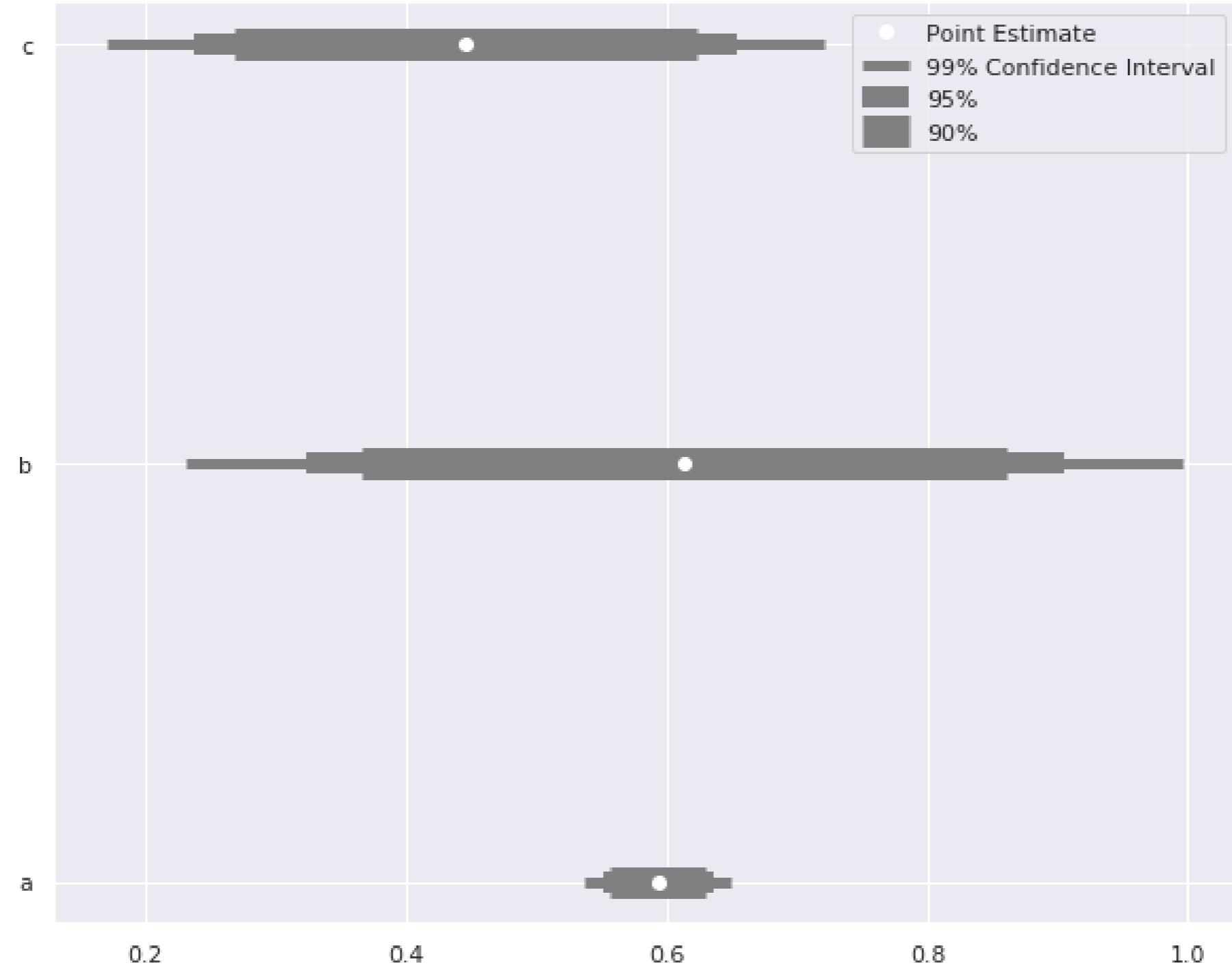
for percent, Z, color in zip(widths, z_scores, colors):
    # Set color to distinguish bands
    plt.fill_between(
        x=data.day,
        y1=data['mean'] - Z*data['std_err'],
        y2=data['mean'] + Z*data['std_err'],
        color=color,
        # Lower opacity so grid can show through
        alpha=0.5,
        # Give each band id for the legend
        label=percent)
```



```
sizes = ['99% Confidence Interval', '95%', '90%']

# Set up different line widths for intervals
widths    = [    5,      10,      15]
Z_scores = [2.58,    1.96,    1.67]

for size, z, width in zip(sizes, Z_scores, widths):
    plt.hlines(
        y = data.y, label = size,
        xmin = data['est'] - z*data['std_err'],
        xmax = data['est'] + z*data['std_err'],
        color = 'grey'
        # Adjust line thickness by interval
        linewidth = width)
plt.plot('est', 'y', 'wo', data = data, label = 'Point Estimate')
plt.legend()
```



# **Let's expand our boundaries!**

**IMPROVING YOUR DATA VISUALIZATIONS IN PYTHON**

# Visualizing the bootstrap

IMPROVING YOUR DATA VISUALIZATIONS IN PYTHON



Nick Strayer

Instructor

# Confidence interval issues

This point is no more ‘supported’ than this one

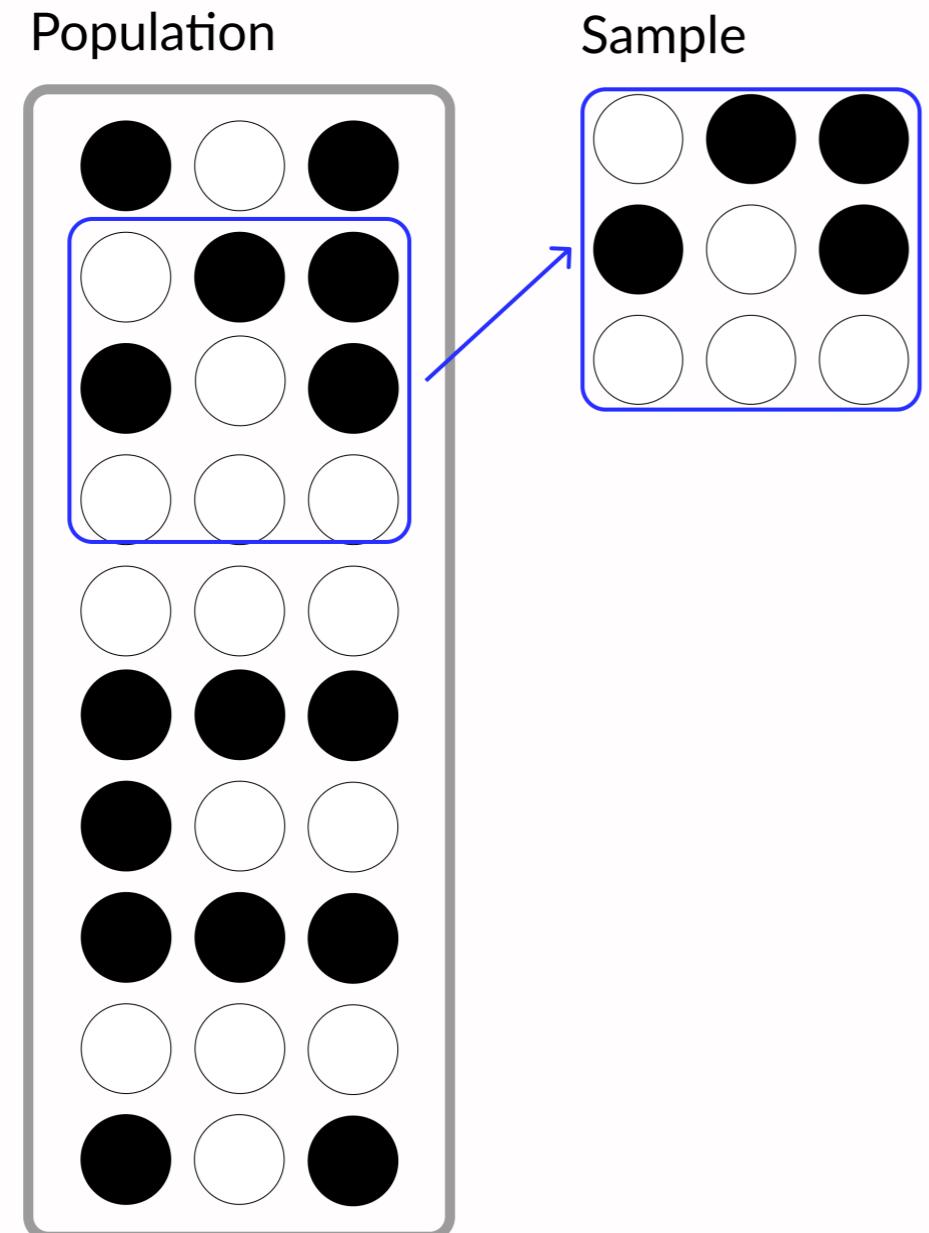


# Confidence interval issues

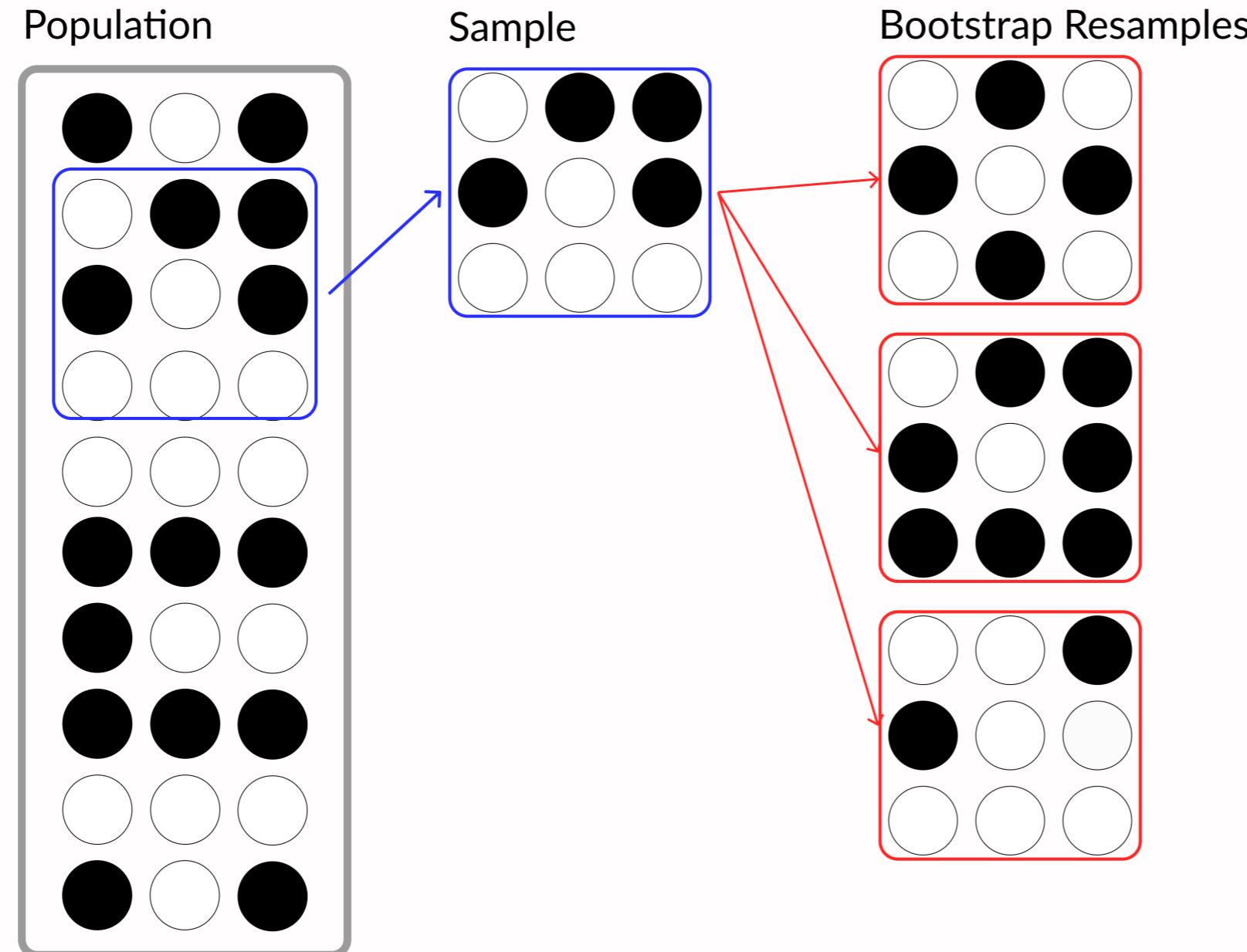
It would be nice to have statements about support within our interval



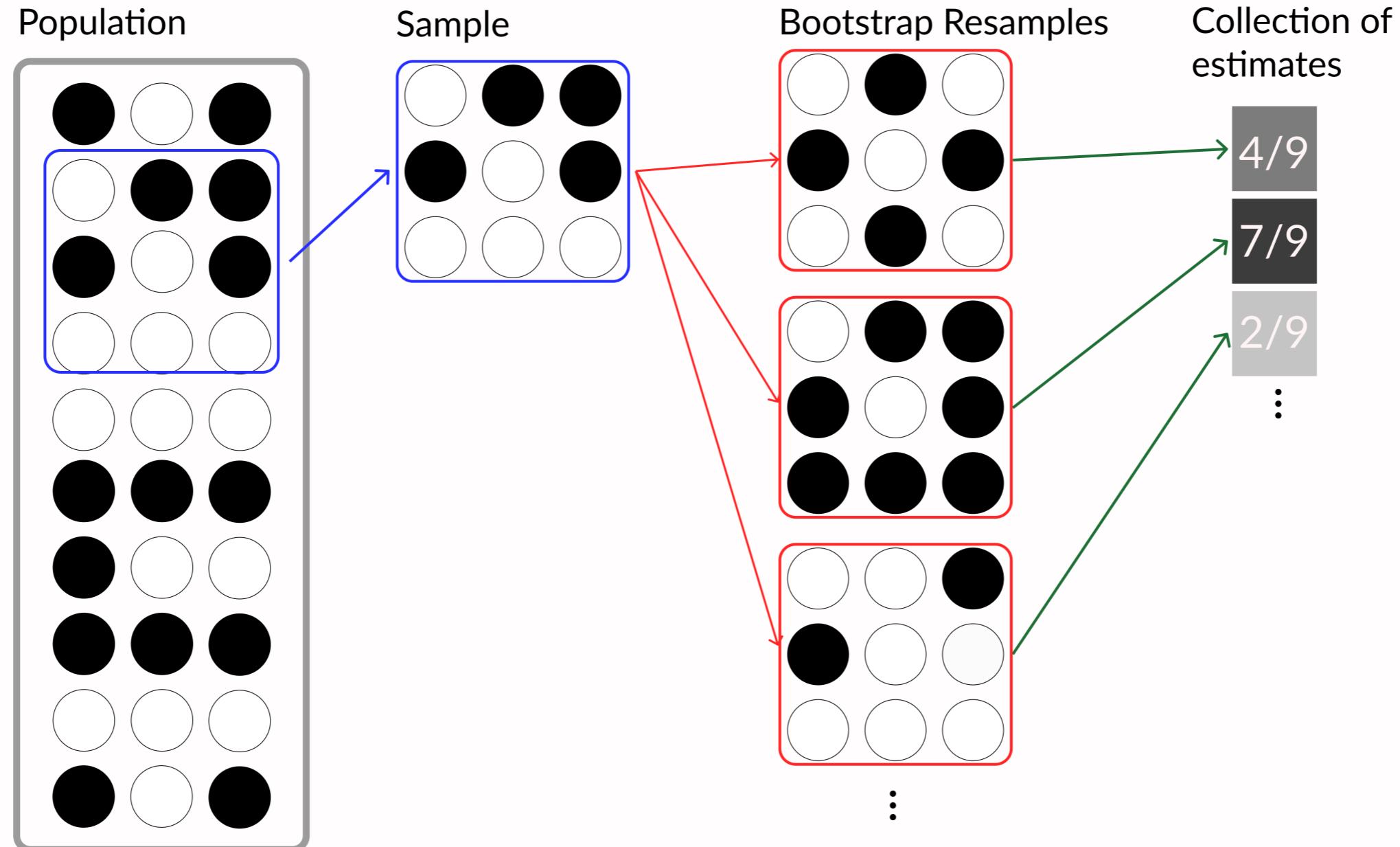
# The “Bootstrap”



# The “Bootstrap”



# The “Bootstrap”



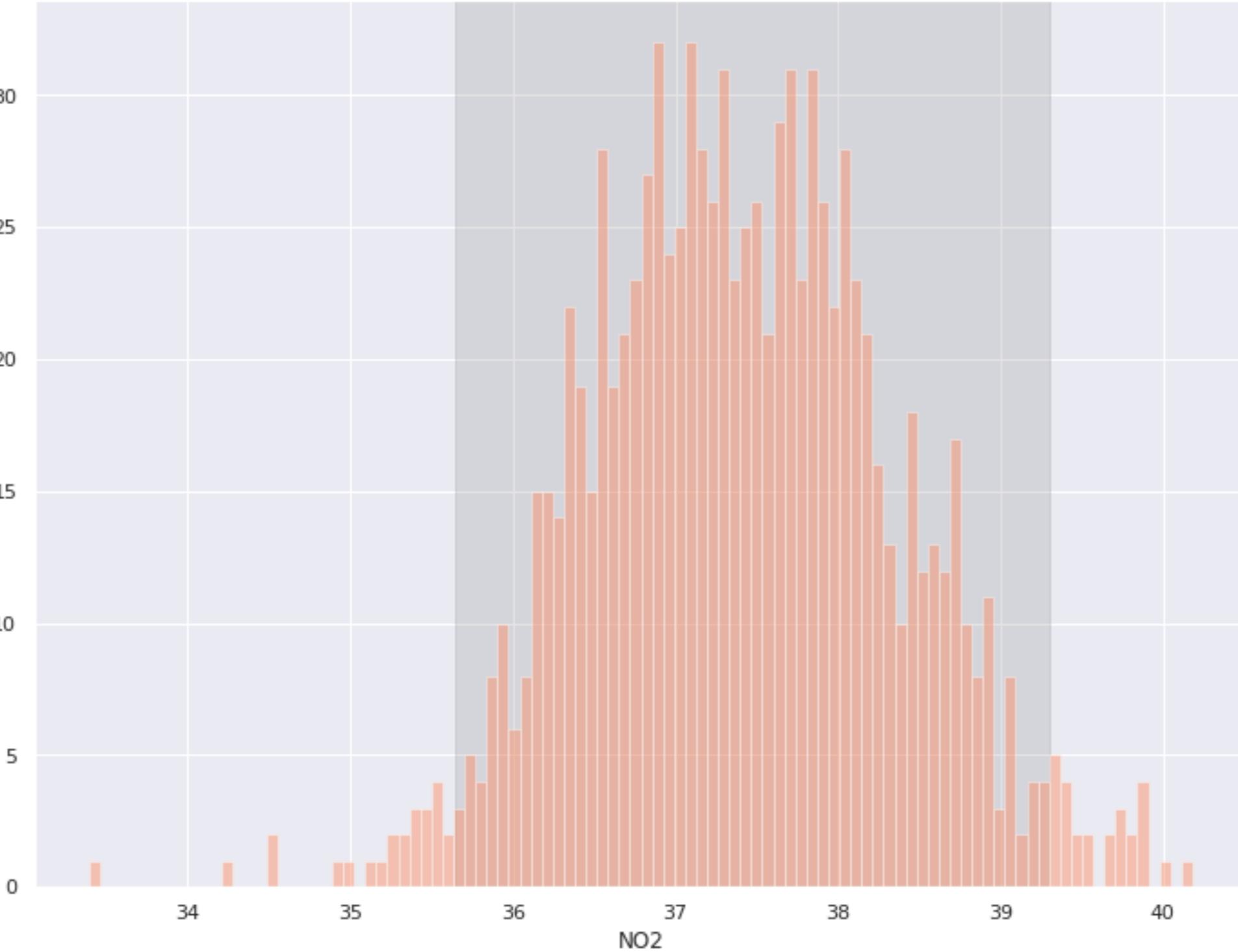
```
denver_may = pollution.query("city == 'Denver' & month == 8")

# Perform bootstrapped mean on a vector
def bootstrap(data, n_boots):
    return [np.mean(np.random.choice(data, len(data)))
            for _ in range(n_boots)]

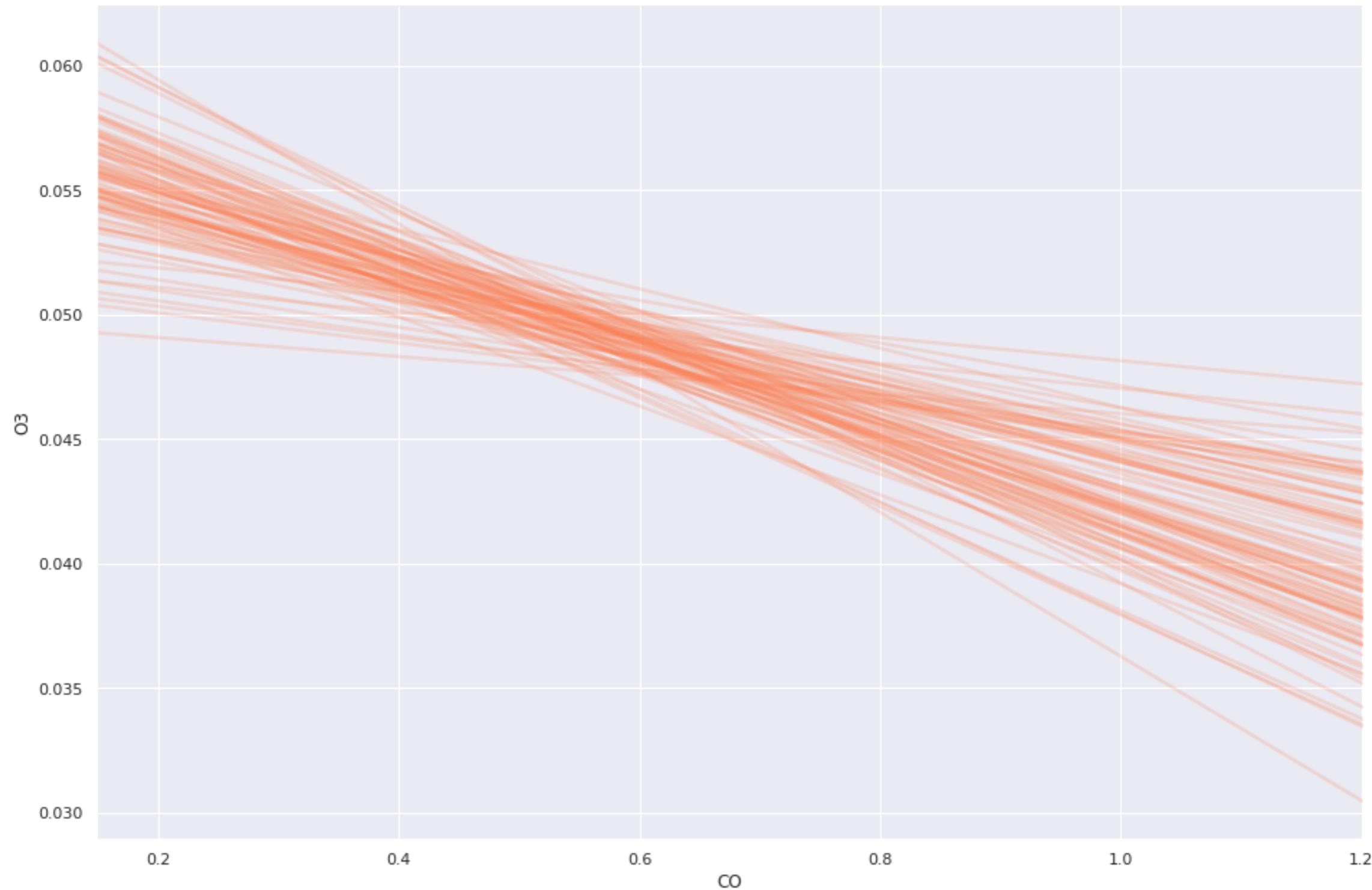
# Generate 1,000 bootstrap samples
boot_means = bootstrap(denver_may.N02, 1000)

# Get lower and upper 95% interval bounds
lower, upper = np.percentile(boot_means, [2.5, 97.5])

# Shaded background of interval
plt.axvspan(lower, upper, color='grey', alpha=0.2)
# Plot histogram of samples
sns.distplot(boot_means, bins = 100, kde=False)
```



```
# Make dataframe of bootstrapped data
denver_may_boot = pd.concat([
    denver_may.sample(n=len(denver_may), replace=True).assign(sample=i)
    for i in range(100)])  
  
# Plot regressions for each sample
sns.lmplot('CO', 'O3', data=denver_may_boot, scatter=False,
            # Tell seaborn to draw a regression
            # line for each resample's data
            hue='sample',
            # Make lines orange and transparent
            line_kws = {'color': 'coral', 'alpha': 0.2},
            # No confidence intervals
            ci=None, legend = False)
```

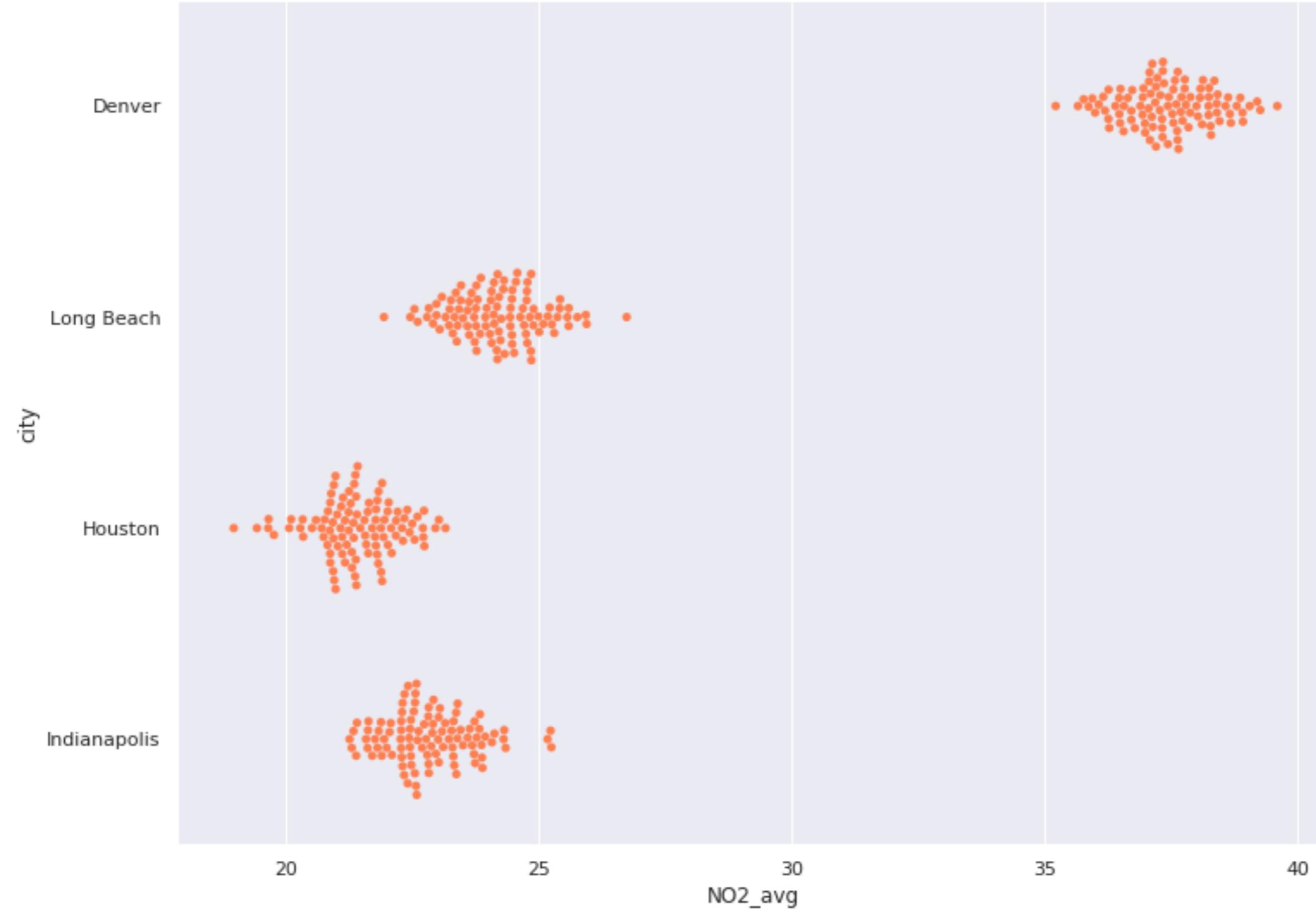


```
aug_pol = pollution.query("month == 8")

# Holder DataFrame for bootstrap samples
city_boots = pd.DataFrame()

for city in ['Denver', 'Long Beach', 'Houston', 'Indianapolis']:
    # Filter to city's N02
    city_N02 = aug_pol[aug_pol.city == city].N02
    # Perform 100 bootstrap samples of city's N02 & put in DataFrame
    cur_boot = pd.DataFrame({ 'N02_avg': bootstrap(city_N02, 100),
                               'city': city })
    # Append to other city's bootstraps
    city_boots = pd.concat([city_boots, cur_boot])

# Use beeswarm plot to visualize bootstrap samples
sns.swarmplot(y="city", x="N02_avg", data=city_boots,
               # Set all the colors to be the same
               color='coral')
```



# Let's get (re)sampling

IMPROVING YOUR DATA VISUALIZATIONS IN PYTHON