Plotly and the Plotly Figure

INTRODUCTION TO DATA VISUALIZATION WITH PLOTLY IN PYTHON

Alex Scriven
Data Scientist

What is Plotly?

- A JavaScript graphing library
  - Don't worry - no need to know JavaScript!
- Plotly has a Python wrapper

Why Plotly?

Plotly has a number of unique advantages:

- Fast and easy to implement simple plots
- Low code/low effort options using `plotly.express`
- (If desired) Extremely customizable
- Interactive plots by default

Creating Plotly Figures

Plotly graphs can be created:

1. With `plotly.express` for simple, quick plots (`px`)
2. With `plotly.graph_objects` (`go`) for more customization
3. With `plotly.figure_factory` for specific, advanced figures

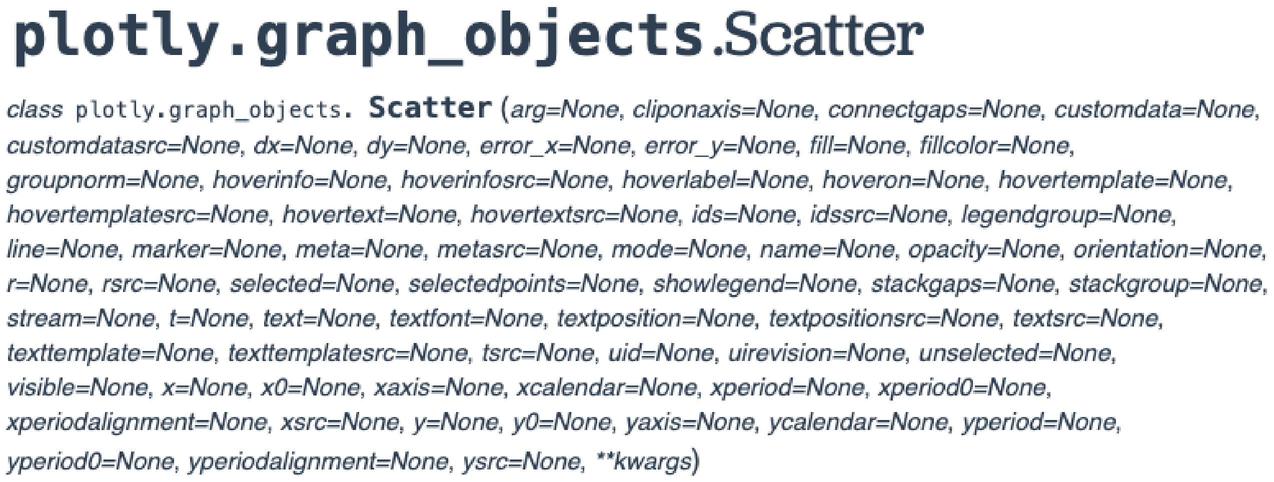
We will spend most of our time on **1** and **2**!

The importance of documentation

Save the links to key documentation!

1. Interactive, introductory docs (with many examples!)
    - https://plotly.com/python
2. Graph_objects pages for specific plots
    - Index here
    - For example, `go.scatter` here
3. The base `go.Figure` documentation linked here
    - Important when we cover `update_layout()` later!

The `go.scatter` documentation page:



plotly.graph_objects.Scatter

```
class plotly.graph_objects. Scatter (arg=None, cliponaxis=None, connectgaps=None, customdata=None,
customdatasrc=None, dx=None, dy=None, error_x=None, error_y=None, fill=None, fillcolor=None,
groupnorm=None, hoverinfo=None, hoverinfosrc=None, hoverlabel=None, hoveron=None, hovertemplate=None,
hovertemplatesrc=None, hovertext=None, hovertextsrc=None, ids=None, idssrc=None, legendgroup=None,
line=None, marker=None, meta=None, metasrc=None, mode=None, name=None, opacity=None, orientation=None,
r=None, rsrc=None, selected=None, selectedpoints=None, showlegend=None, stackgaps=None, stackgroup=None,
stream=None, t=None, text=None, textfont=None, textposition=None, textpositionsrc=None, textsrc=None,
texttemplate=None, texttemplatesrc=None, tsrc=None, uid=None, uirevision=None, unselected=None,
visible=None, x=None, x0=None, xaxis=None, xcalendar=None, xperiod=None, xperiod0=None,
xperiodalignment=None, xsrc=None, y=None, y0=None, yaxis=None, ycalendar=None, yperiod=None,
yperiod0=None, yperiodalignment=None, ysrc=None, **kwargs)
```

The Plotly Figure

A Plotly Figure has 3 main components:

- `layout` : Dictionary controlling style of the figure
  - One `layout` per figure
- `data` : List of dictionaries setting graph type and data itself
  - Data + type = a `trace` . There are over 40 types!
  - Can have multiple traces per graph
- `frames` : For animated plots (beyond this course)

Inside a Plotly Figure

Let's see inside an example Plotly `figure` object:

```
print(fig)
```

```
Figure({'data': [{'type': 'bar',
        'x': [Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday],
         'y': [28, 27, 25, 31, 32, 35, 36]}],
    'layout': {'template': '...',
              'title': {'font': {'color': 'red', 'size': 15},
               'text': 'Temperatures of the week', 'x': 0.5}}})
```

- What do you think this graph will look like?

Inside our Figure

```
Figure({ 'data': [{'type': 'bar',
     'x': [Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday],
     'y': [28, 27, 25, 31, 32, 35, 36]}],
     'layout': {'template': '...','title': {'font': {'color': 'red', 'size': 15},
     'text': 'Temperatures of the week', 'x': 0.5}}})
```

- Type 'bar'
- An X and Y axis with data noted
- A title with some text around temperatures of the week

Guess: A bar chart of temperatures of the days of the week

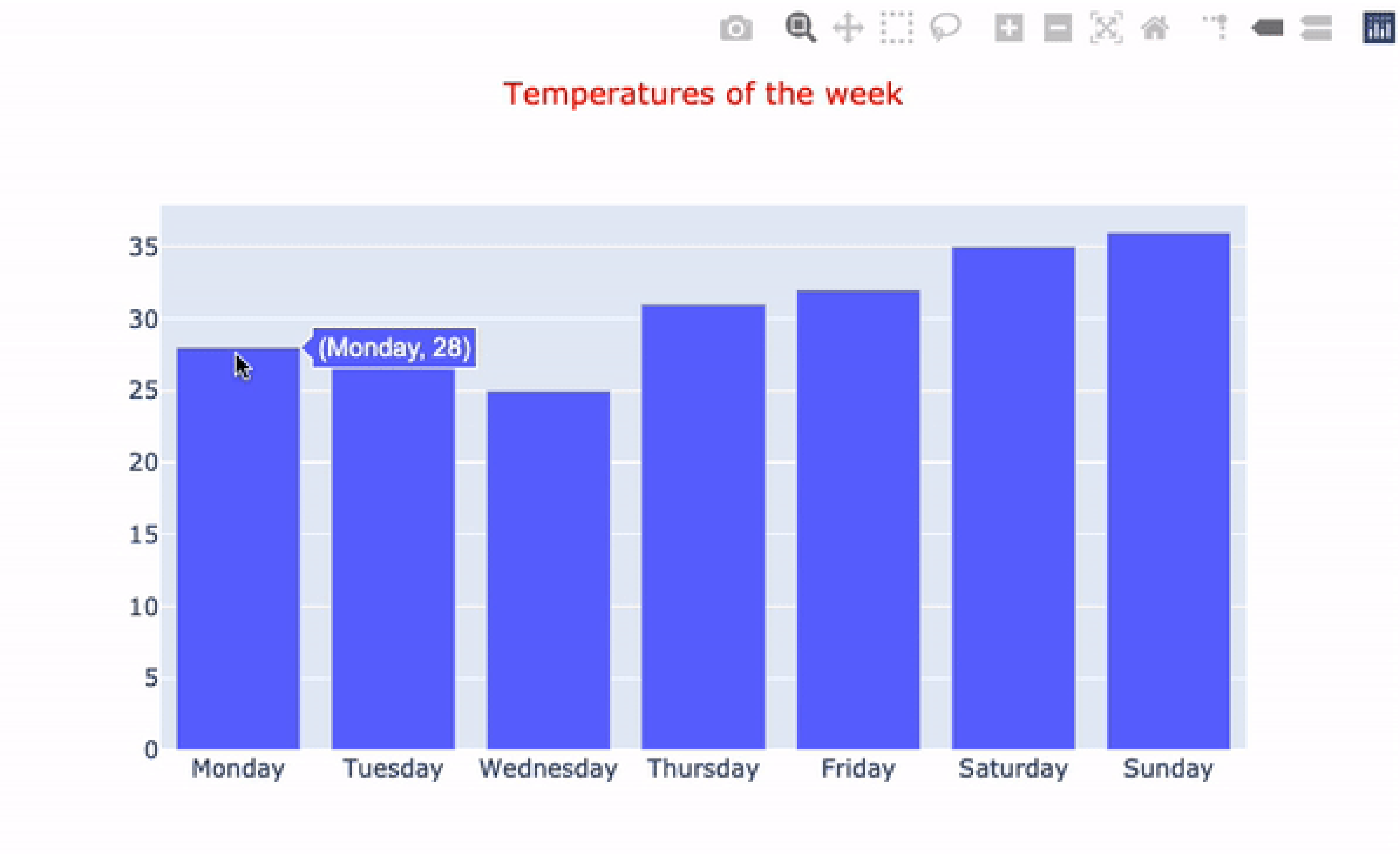
Creating our Figure

Constructing this figure from scratch (just this once!):

```python
import plotly.graph_objects as go
figure_config = dict({ "data": [{"type": "bar",
              "x": ["Monday", "Tuesday", "Wednesday",
              "Thursday", "Friday", "Saturday", "Sunday"],
              "y": [28, 27, 25, 31, 32, 35, 36]}],
            "layout": {"title": {"text": "Temperatures of the week",
            "x": 0.5, "font": {'color': 'red', 'size': 15}}}})
fig = go.Figure(figure_config)
fig.show()
```

Our Figure revealed

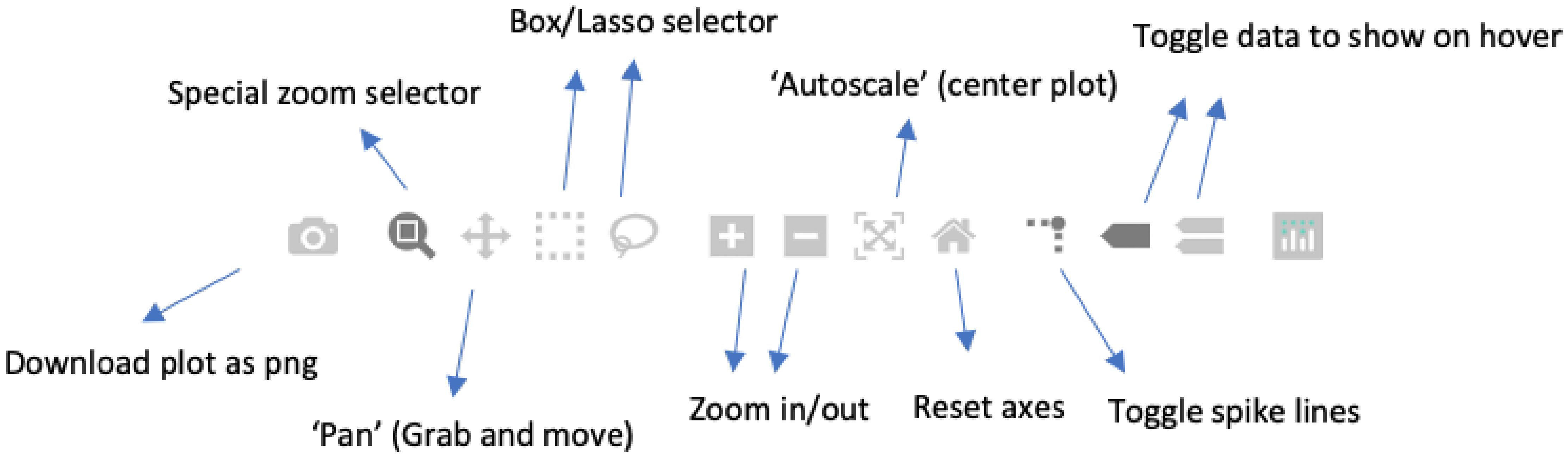
Let's see what is produced!



Plotly's instant interactivity

Plotly provides instant interactivity:

- Hover over data points
- Extra interactive buttons



Let's practice!

INTRODUCTION TO DATA VISUALIZATION WITH PLOTLY IN PYTHON

Univariate visualizations

INTRODUCTION TO DATA VISUALIZATION WITH PLOTLY IN PYTHON

Alex Scriven
Data Scientist

Our approach

Plotly shortcut methods:

1. `plotly.express`
   - Specify a DataFrame and its columns as arguments
   - Quick, nice but less customization
2. `graph_objects` `go.X` methods (`go.Bar()` , `Go.Scatter()`) etc.
   - Many more customization options, but more code needed

What are univariate plots?

Univariate plots display only one variable

For analyzing the *distribution* of that variable

Common univariate plots:

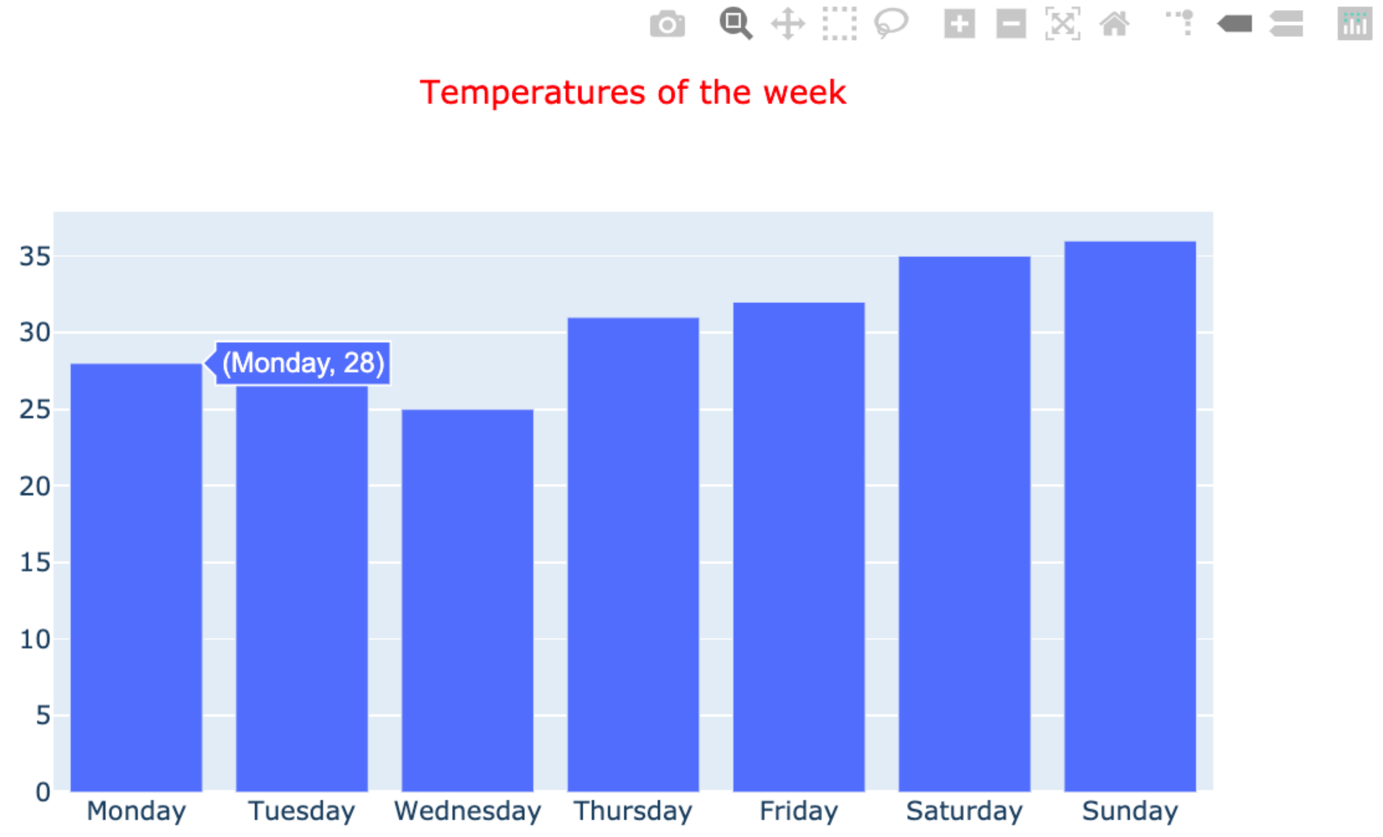
- Bar chart
- Histogram
- Box plot
- Density plots

Bar charts

Bar charts have:

- X-axis with a bar per group
  - One group = one bar! (Hence UNI-variate)
- The y-axis height represents the value of some variable

We built one in the last lesson!



Bar charts with plotly.express

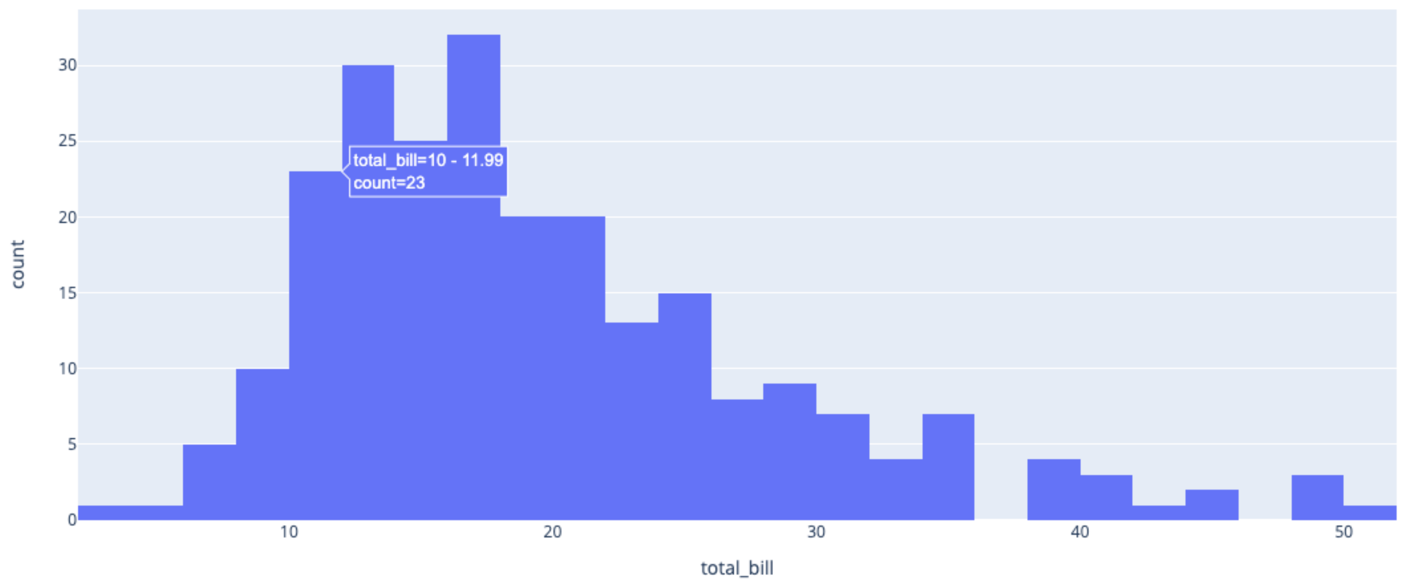
Let's rebuild with `plotly.express`

```
import plotly.express as px
weekly_temps = pd.DataFrame({
    'day': ['Monday', 'Tuesday',
            'Wednesday', 'Thursday', 'Friday',
            'Saturday', 'Sunday'],
    'temp': [28, 27, 25, 31, 32, 35, 36]})
fig = px.bar(data_frame=weekly_temps, x='day', y='temp')
fig.show()
```

Histograms

Histograms have:

- Multiple columns (called 'bins') representing a range of values
  - The height of each bar = count of samples within that bin range
- The number of bins can be manual or automatic



Our dataset

Dataset collected by scientific researchers on Penguins!

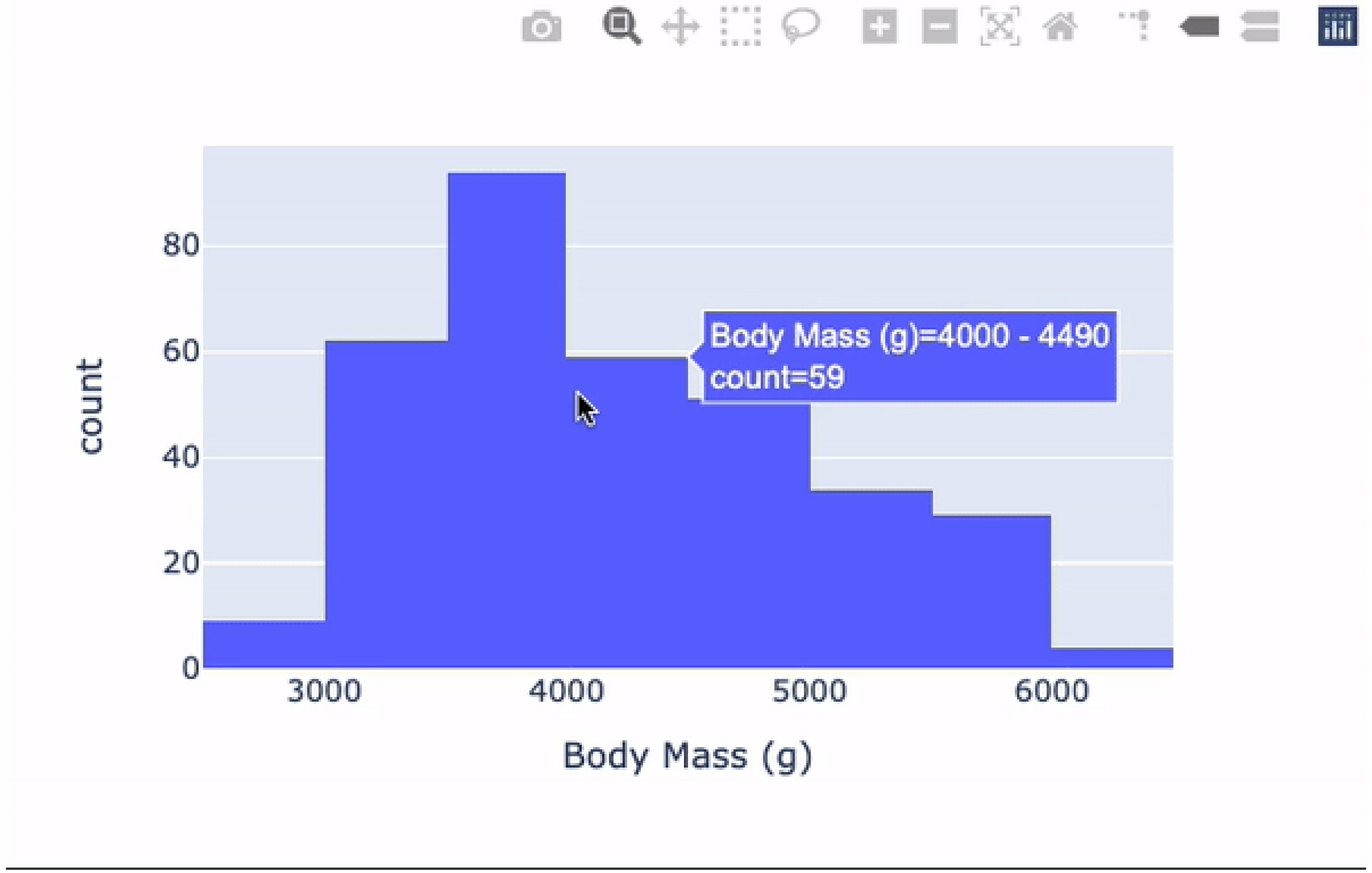
- Contains various body measurements like, beak size, weight, etc.
- Contains different species, genders, and ages of penguins

Histograms with plotly.express

We can create a simple histogram:

```
fig = px.histogram(
            data_frame=penguins,
            x='Body Mass (g)',
            nbins=10)
fig.show()
```

This is what is produced:



Useful histogram arguments

Other `px.histogram` arguments :

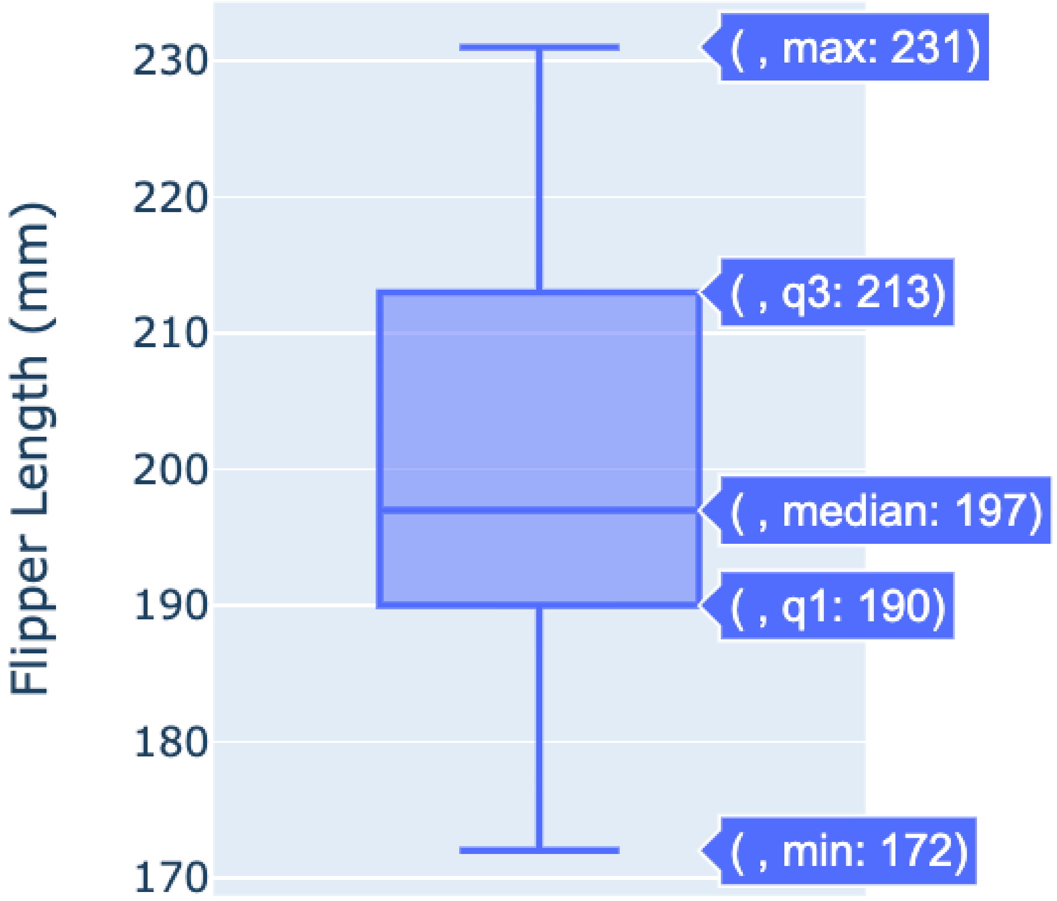
- `orientation` : To orient the plot vertically (`v`) or horizontally (`v`)
- `histfunc` : Set the bin aggregation (eg: average, min, max).

Check the **docs** for more!

Box (and whisker) plots

Summarizes a variable visually using quartile calculations;

- Middle area represents *interquartile range*
  - Top line = 3rd quartile (75th percentile)
  - Middle line = median (50th percentile)
  - Bottom line = first quartile (25th percentile)
- Top/bottom bars = min/max, excluding outliers
- Outlying dots are outliers

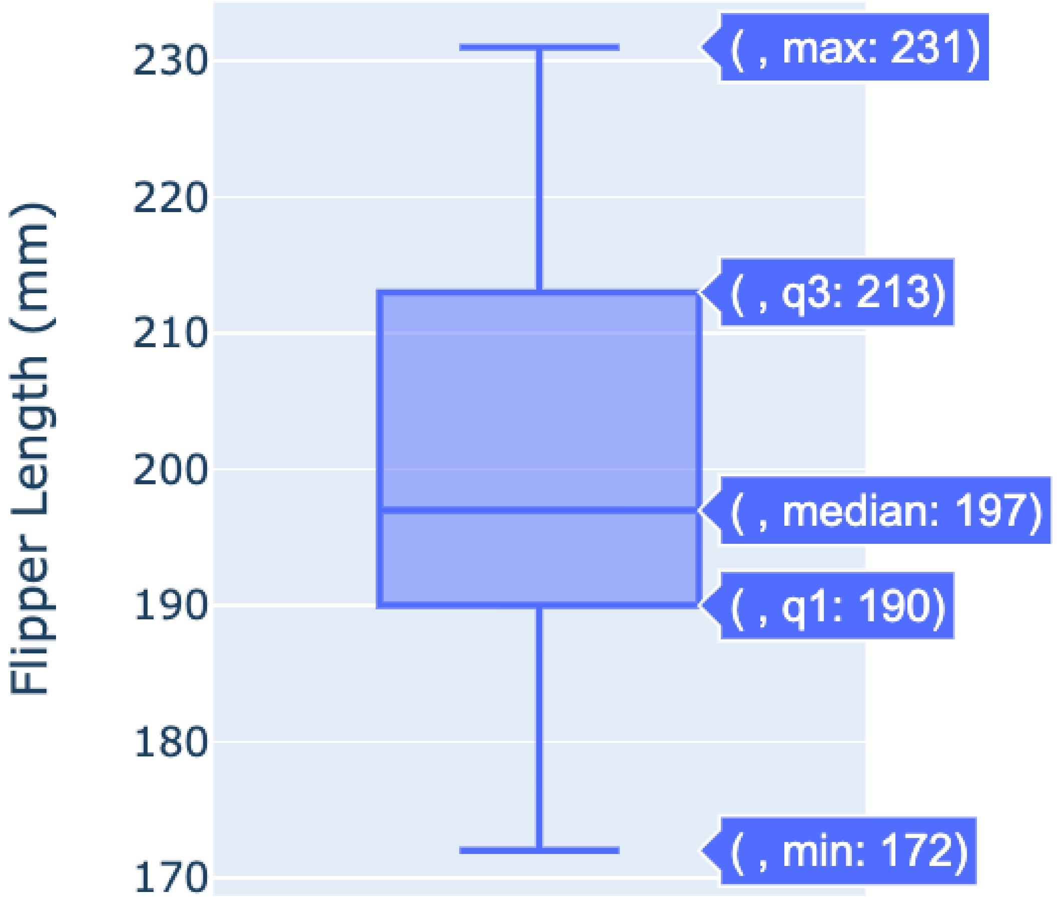


Box plots with plotly.express

This is what is produced:

Let's create a simple box plot:

```
fig = px.box(data_frame=penguins,
             y="Flipper Length (mm)")
fig.show()
```



Useful box plot arguments

Useful box plot arguments:

- `hover_data` : A list of column name(s) to display on hover
  - Useful to understand outliers
- `points` : Further specify how to show outliers

Check the **docs** for more!

Let's practice!

INTRODUCTION TO DATA VISUALIZATION WITH PLOTLY IN PYTHON

Customizing color

INTRODUCTION TO DATA VISUALIZATION WITH PLOTLY IN PYTHON

Alex Scriven
Data Scientist

Customization in general

How to customize plots:

1. At figure creation if an argument exists (like `color` !)
2. Using an **important** function `update_layout()`
   - Takes a dictionary argument
   - E.g.: `fig.update_layout({'title':{text:'A New Title'}})`

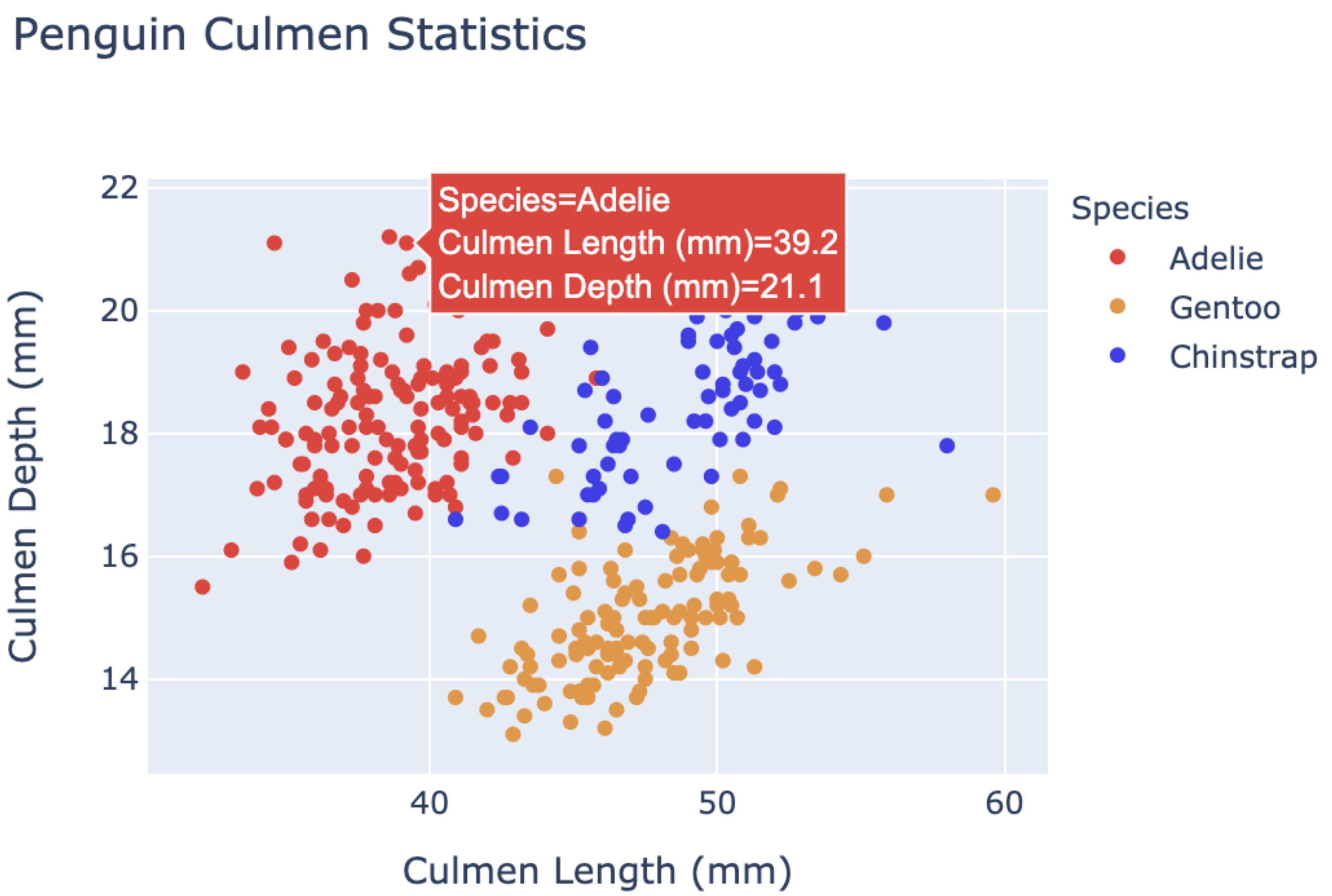
The method chosen depends on plot type how it was created.

MANY properties possible — See the **documentation**

Why customize color?

Customizing color can help you

1. Make plots look awesome!
2. Convey analytical insights
   - Color in this scatterplot adds a 3rd dimension.



Some color theory

Computers use RGB encoding to specify colors:

- RGB = A 3-digit code (each 0-255) mixing **R**ed, **G**reen, **B**lue together to make colors.
  - Imagine mixing Red, Green and Blue paint together!
  - (0,0,255) is totally blue and (255,255,0) is yellow

See more in **this article**

Some other examples of RGB colors:

Color	RGB Code
🟪	(245, 66, 230)
🟩	(105, 245, 66)
🟥	(245, 66, 87)
🟦	(50, 47, 247)

Specifying colors in plotly.express

In `plotly.express` :

- Often a `color` argument (DataFrame column)
    - A different (*automatic*) color given to each category in this column
    - A color scale/range is used if numerical column specified

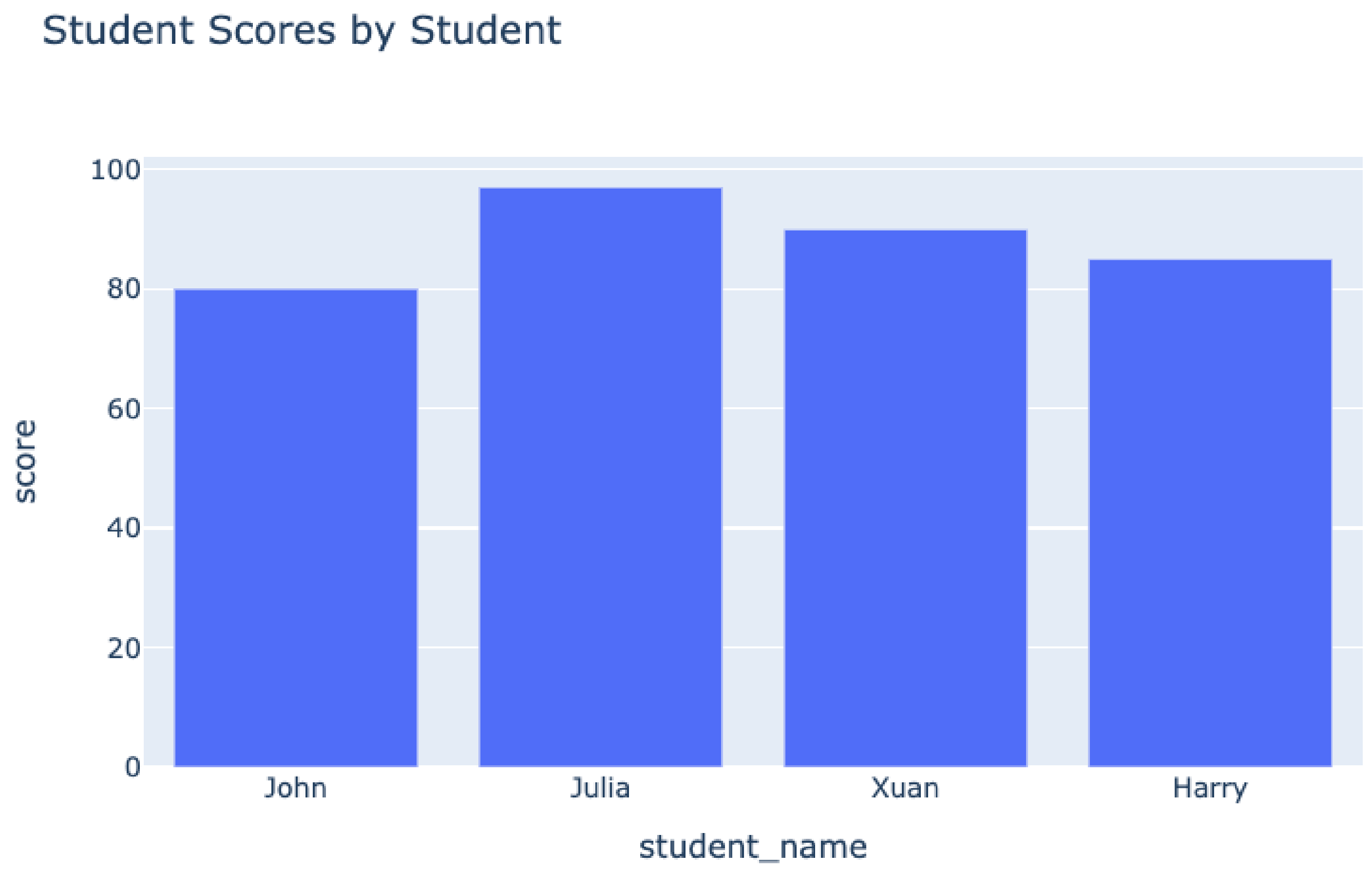
Our simple bar chart from a previous lesson (adding a `City` column)

```
fig = px.bar(data_frame=student_scores,
             x='student_name',
             y='score',
             title='Student Scores by Student'
             color='city')
fig.show()
```

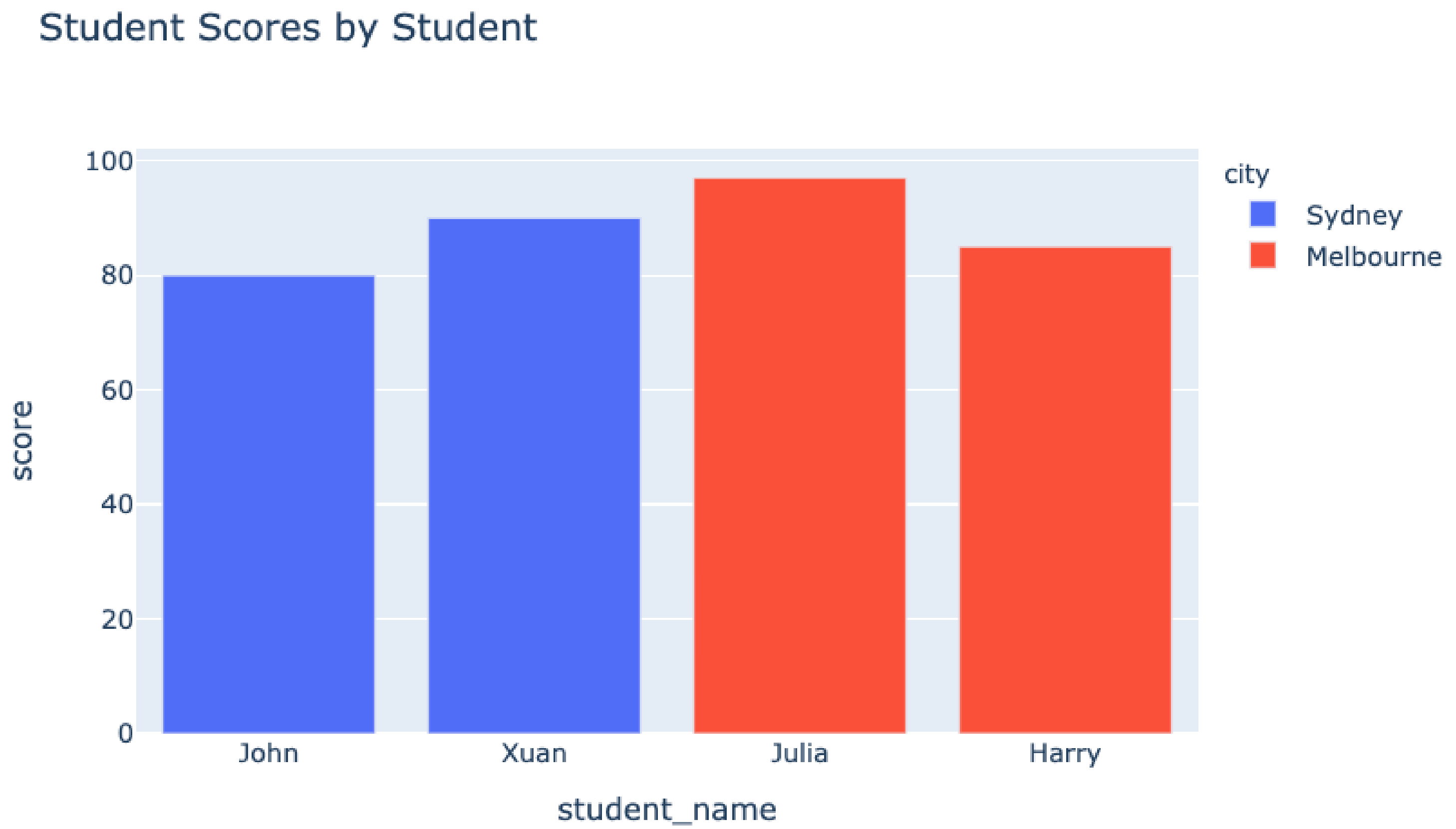
[1] Make sure to check the documentation for each figure.

Our colors revealed

The plot before:



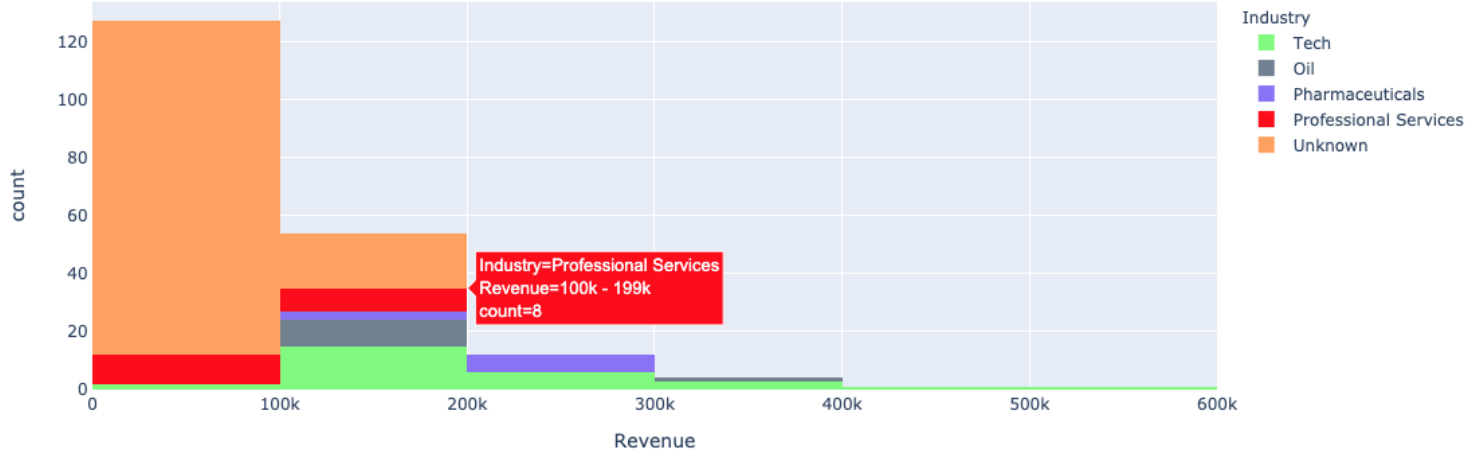
Our plot after:

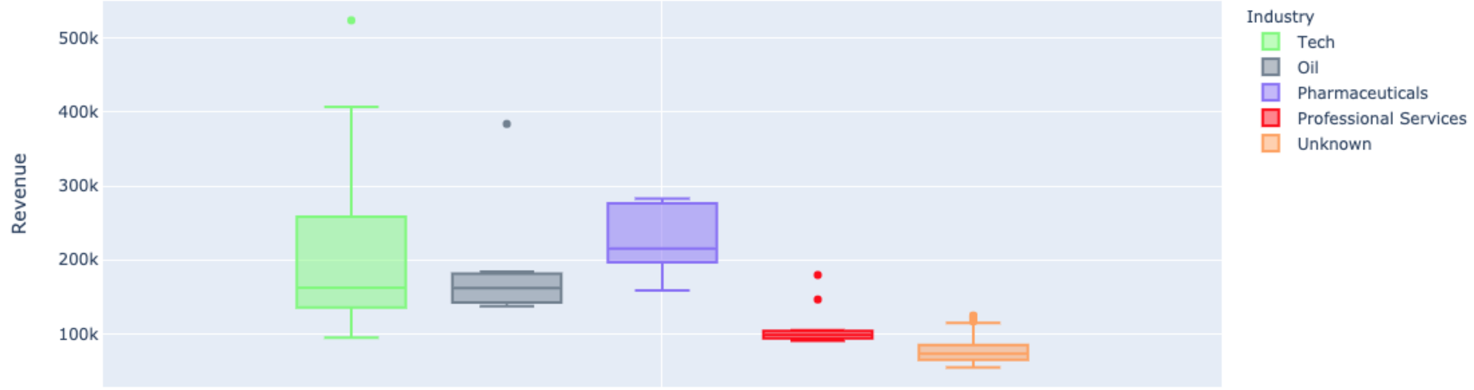


Color with univariate plots

Using `plotly.express` `color` argument with univariate (bar, histogram) plots:

- Histograms - stacked bars
- Box plots - produces multiple (one per category)





Specific colors in plotly.express

What if we don't like the automatic colors?

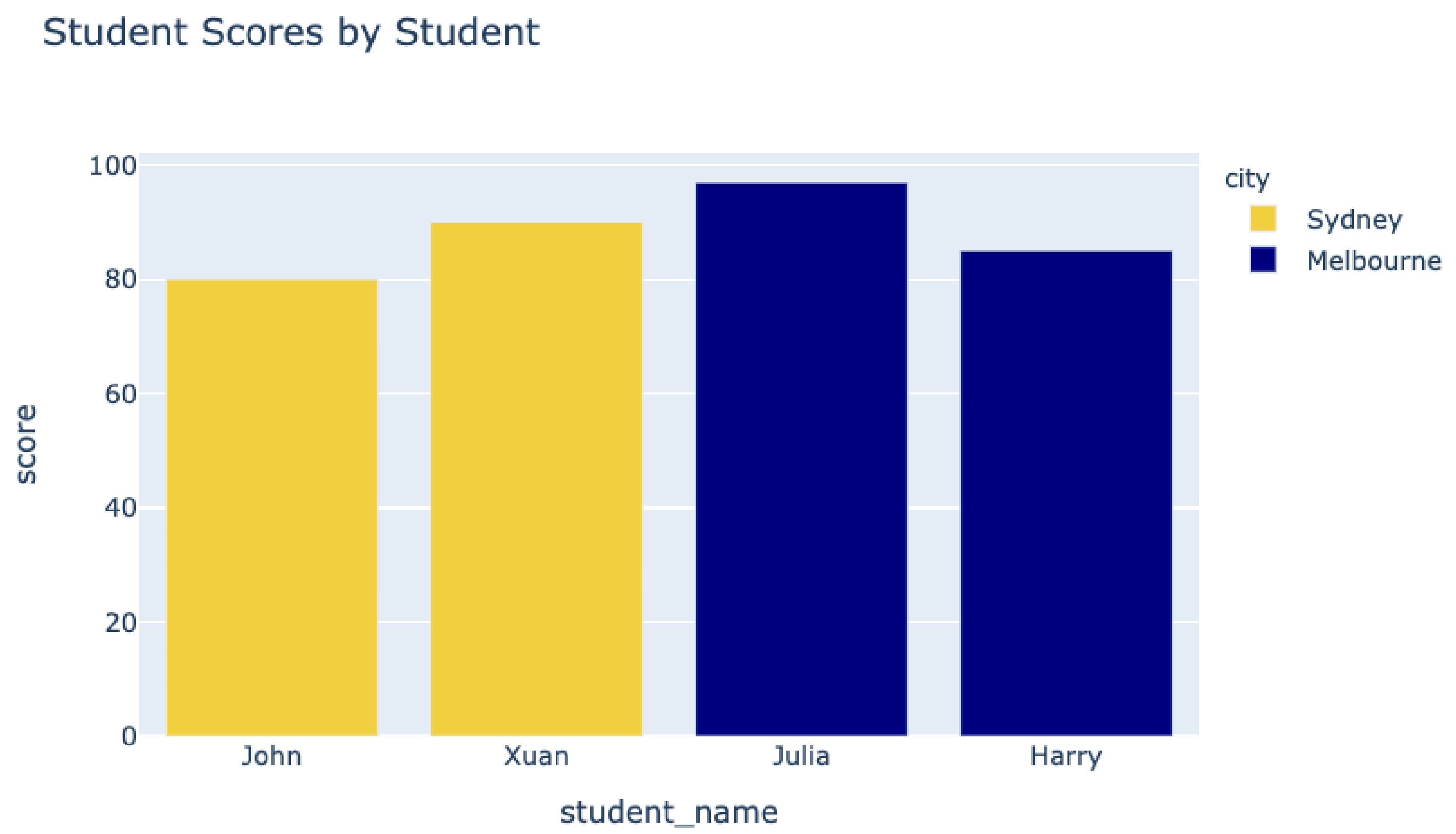
- `color_discrete_map` : A dictionary mapping specific categorical values to colors using a string RGB code specification — `'rgb(X,X,X)'`
- Can also express (basic) colors as strings such as `'red'` , `'green'` etc.

Our specific colors

Let's update our colors. Sandy yellow for 'Sydney' and navy blue for 'Melbourne'

```
fig = px.bar(
    data_frame=student_scores,
    x='student_name', y='score',
    title="Student Scores by Student",
    color_discrete_map={
    'Melbourne': 'rgb(0,0,128)',
    'Sydney': 'rgb(235, 207, 52)'},
    color='city')
```

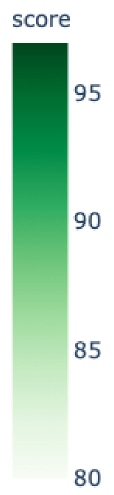
Produces:



Color scales in plotly.express

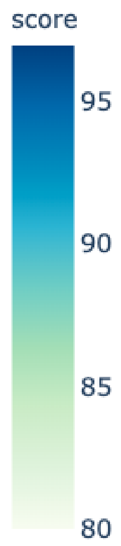
You can create color scales too.

- Single color scales. For example, light to dark green.



- Multiple colors to merge into each other. For example, green into blue.

`color_continuous_scale` allows us to do this with built-in or constructed color scales.



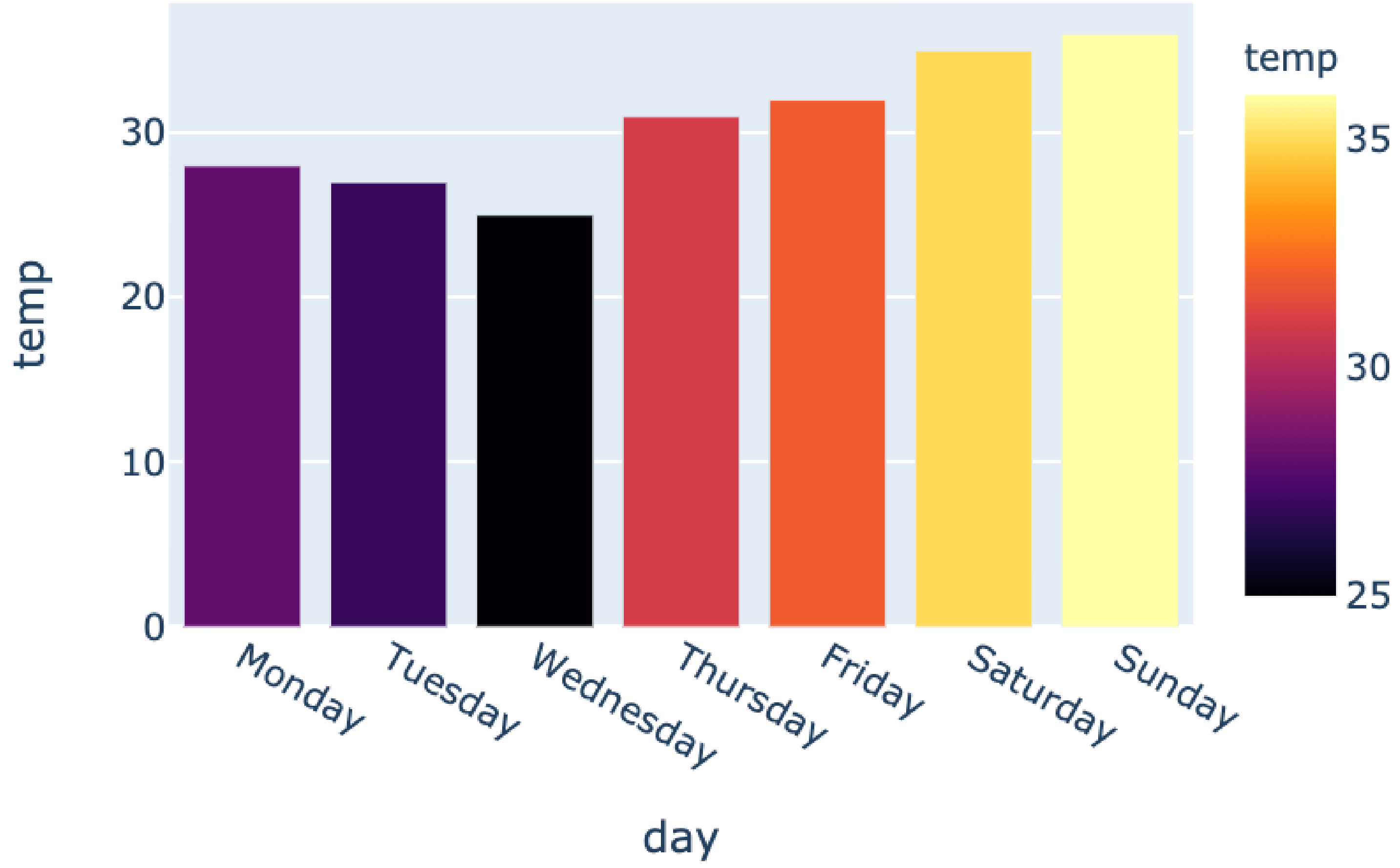
Using built-in color scales

Let's use a built-in color scale:

```
fig = px.bar(data_frame=weekly_temps,
    x='day', y='temp',
    color='temp',
    color_continuous_scale='inferno')
fig.show()
```

- Many **built-in scales** available

Our plot:

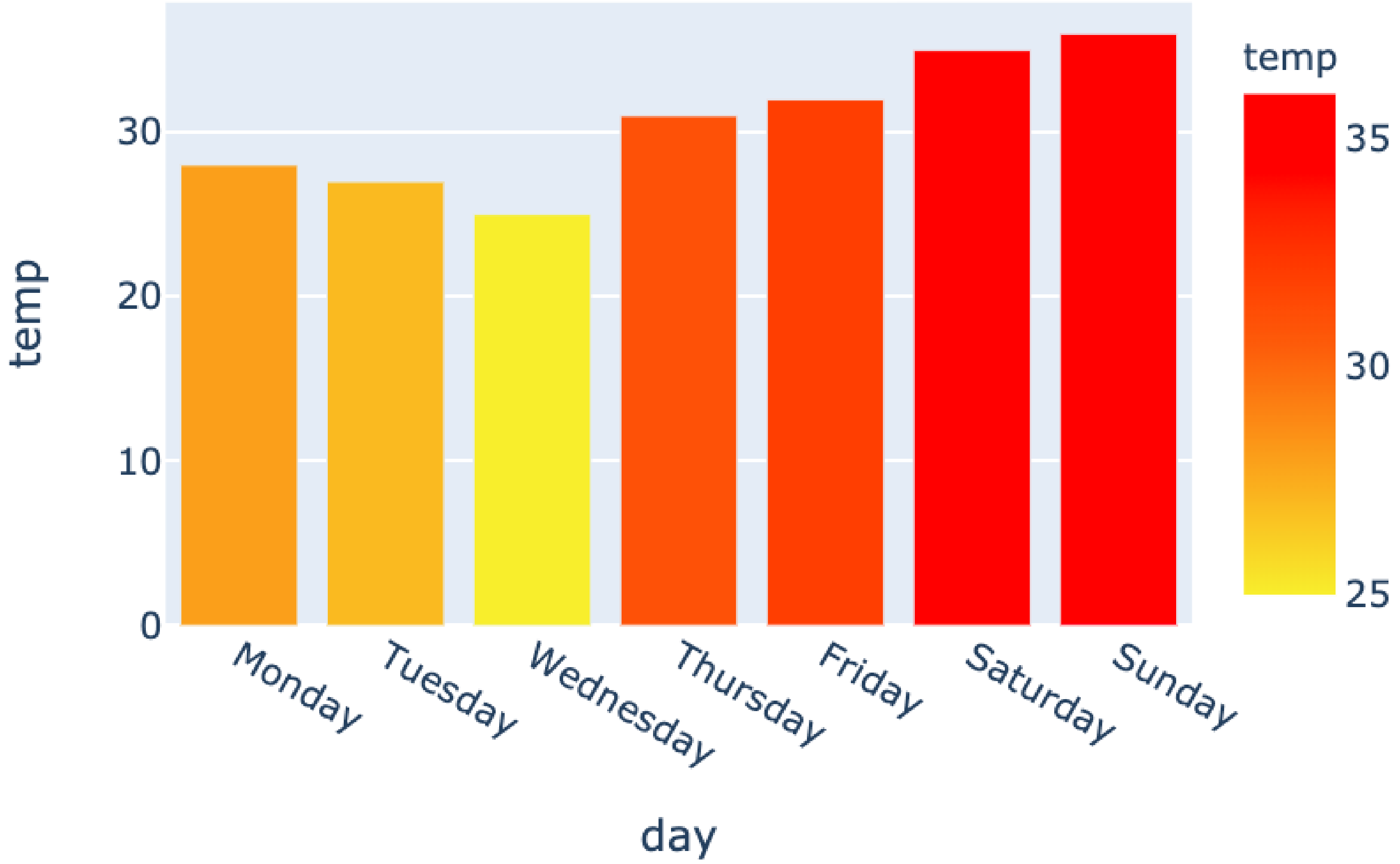


Constructing our own color range

Let's construct our own color scale - yellow through orange to red

```
my_scale=[('rgb(242, 238, 10)'),
          ('rgb(242, 95, 10)'),
          ('rgb(255,0,0)')]
fig = px.bar(data_frame=weekly_temps,
    x='day', y='temp',
    color_continuous_scale=my_scale,
    color='temp')
```

Our plot:



Let's practice!

INTRODUCTION TO DATA VISUALIZATION WITH PLOTLY IN PYTHON

Bivariate visualizations

INTRODUCTION TO DATA VISUALIZATION WITH PLOTLY IN PYTHON

Alex Scriven
Data Scientist

What are bivariate visualizations?

Bivariate plots are those which display (and can therefore compare) two variables.

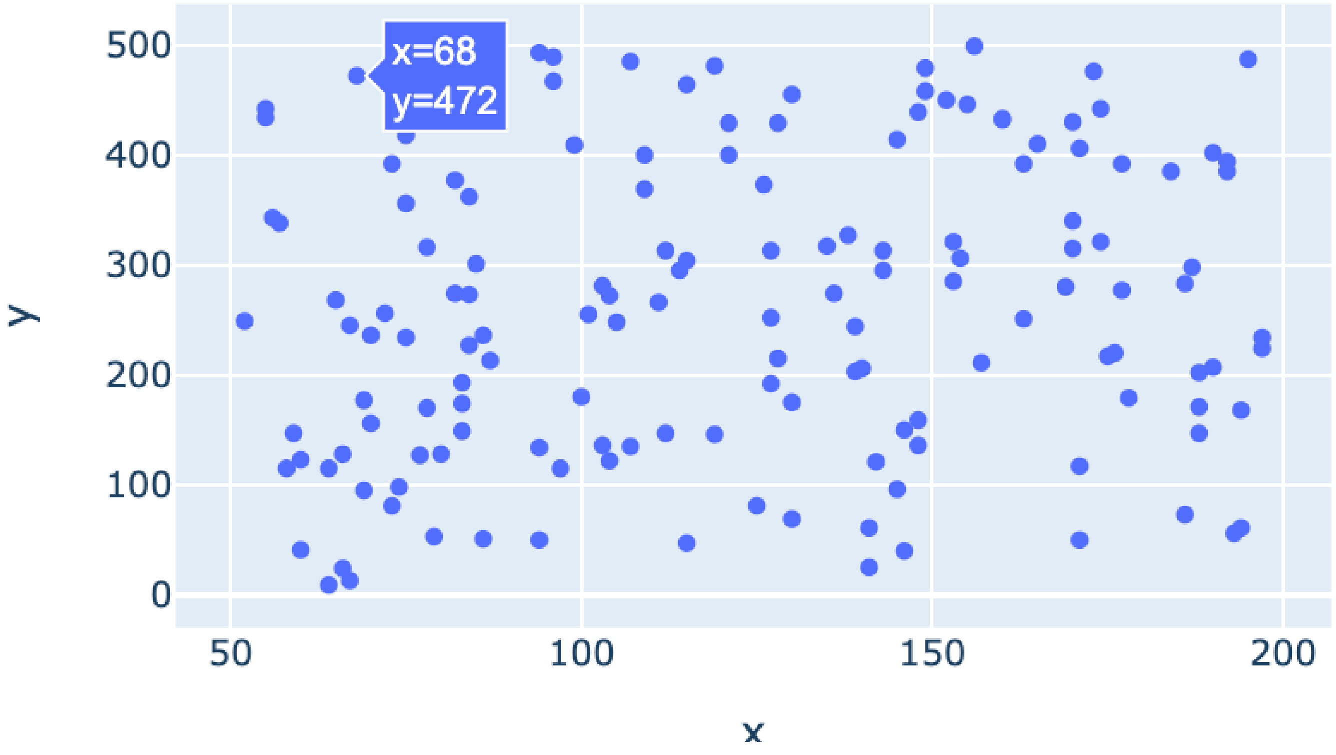
Common bivariate plots include:

- scatterplots
- Correlation plots
- Line charts

scatterplot

A scatterplot is a plot consisting of:

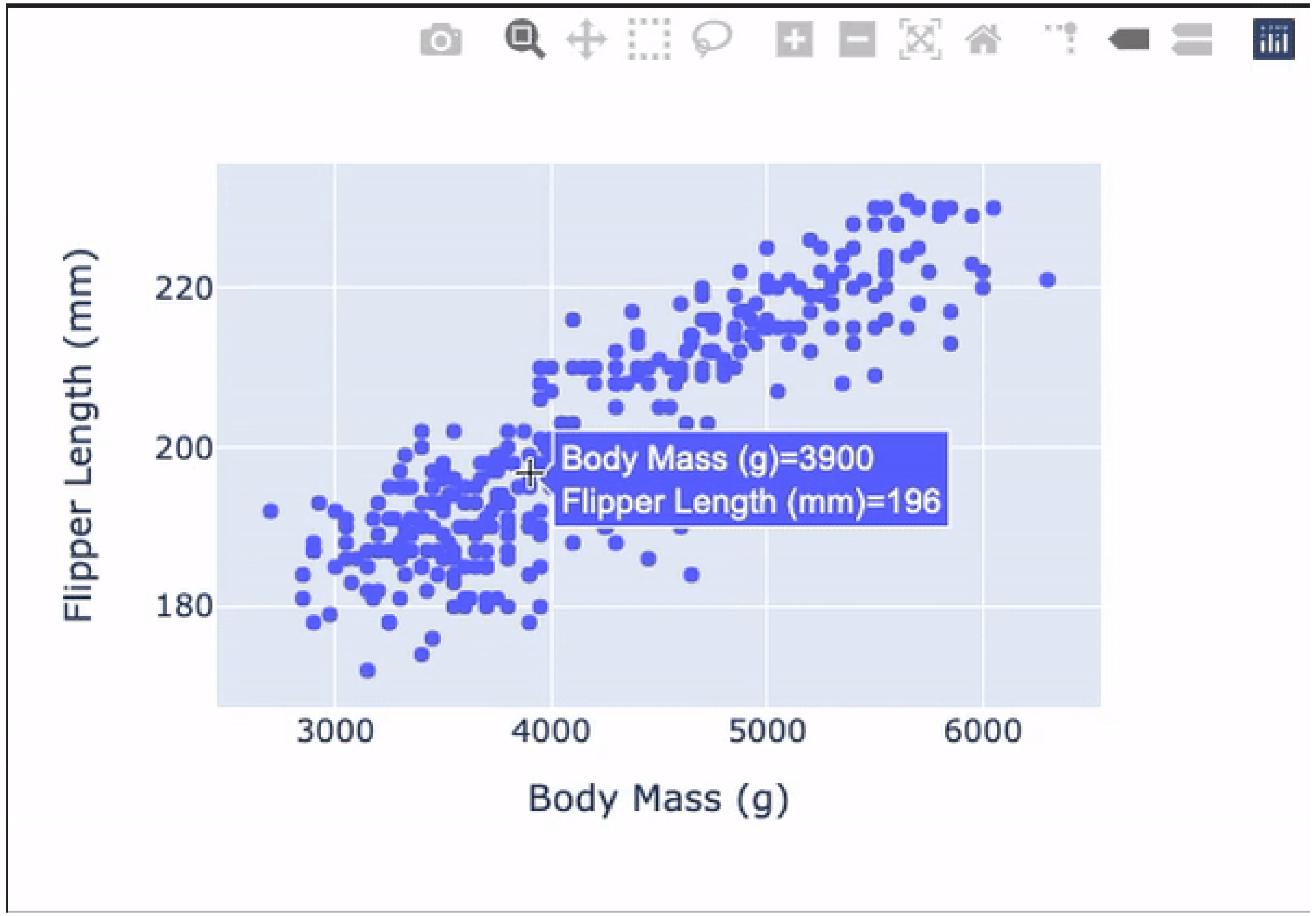
- A y-axis representing one variable
- An x-axis representing a different variable
- Each point is a dot on the graph, e.g., (68, 472)



scatterplot with plotly.express

Visualizing Flipper Length and Body Mass with `plotly.express`:

```
import plotly.express as px
fig = px.scatter(
        data_frame=penguins,
        x="Body Mass (g)",
        y="Flipper Length (mm)")
fig.show()
```



More plotly.express arguments

Useful `plotly.express` scatterplot arguments:

- `trendline` : Add different types of trend lines
- `symbol` : Set different symbols for different categories

Check the **documentation** for more!

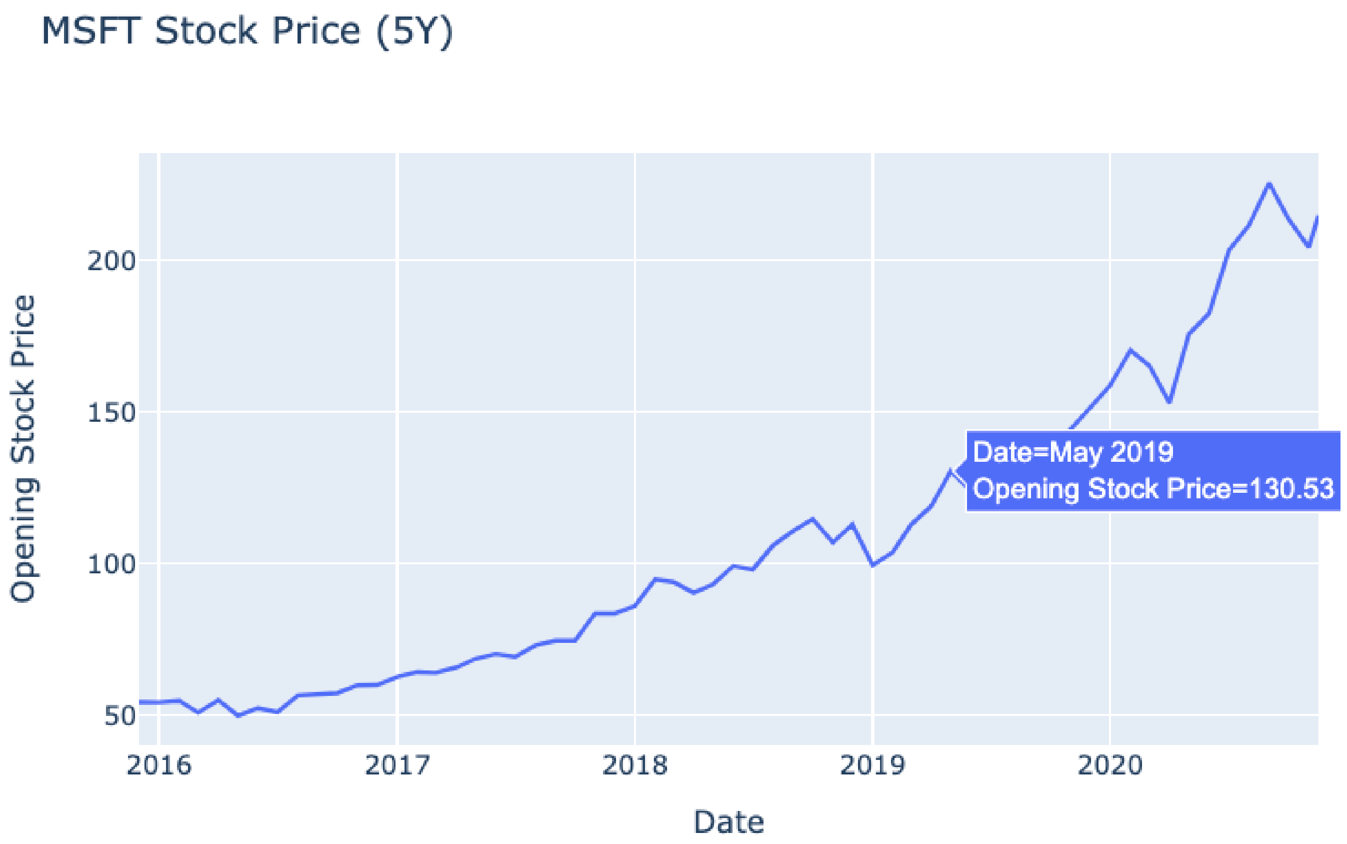
Line charts in plotly.express

A line chart is used to plot some variable (y-axis) over time (x-axis).

Let's visualize Microsoft's stock price.

```
fig = px.line(
  data_frame=msft_stock,
  x='Date',
  y='Open',
  title='MSFT Stock Price (5Y)')
fig.show()
```

Here is our simple line chart:



scatterplots and line plots with graph_objects

For more customization, `graph_objects` uses `go.Scatter()` for both scatter and line plots.

Here is the code for our penguins scatterplot using `graph_objects`

Here is the code for our line chart with `graph_objects`

- Remember to set 'mode'
  - And use DataFrame subsets, not column names

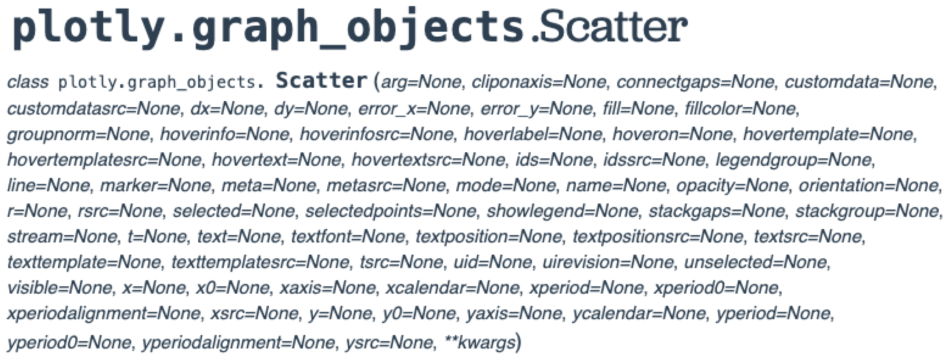
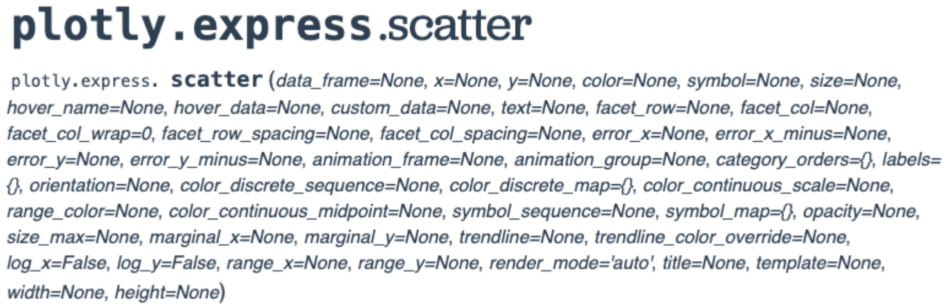
```python
import plotly.graph_objects as go
```

```python
fig = go.Figure(go.Scatter(
  x=penguins['Body Mass (g)'],
  y=penguins['Flipper Length (mm)'],
  mode='markers'))
```

```python
fig = go.Figure(go.Scatter(
  x=msft_stock['Date'],
  y=msft_stock['Opening Stock Price'],
  mode='lines'))
```

graph_objects vs. plotly.express?

When should we use `plotly.express` or `graph_objects`? Largely, it is about customization - `graph_objects` has many more options!

graph_objects	express
 plotly.graph_objects.Scatter class plotly.graph_objects. Scatter (arg=None, cliponaxis=None, connectgaps=None, customdata=None, customdatasrc=None, dx=None, dy=None, error_x=None, error_y=None, fill=None, fillcolor=None, groupnorm=None, hoverinfo=None, hoverinfosrc=None, hoverlabel=None, hoveron=None, hovertemplate=None, hovertemplatesrc=None, hovertext=None, hovertextsrc=None, ids=None, idssrc=None, legendgroup=None, line=None, marker=None, meta=None, metasrc=None, mode=None, name=None, opacity=None, orientation=None, r=None, rsrc=None, selected=None, selectedpoints=None, showlegend=None, stackgaps=None, stackgroup=None, stream=None, t=None, text=None, textfont=None, textposition=None, textpositionsrc=None, textsrc=None, texttemplate=None, texttemplatesrc=None, tsrc=None, uid=None, uirevision=None, unselected=None, visible=None, x=None, x0=None, xaxis=None, xcalendar=None, xperiod=None, xperiod0=None, xperiodalignment=None, xsrc=None, y=None, y0=None, yaxis=None, ycalendar=None, yperiod=None, yperiod0=None, yperiodalignment=None, ysrc=None, **kwargs)	 plotly.express.scatter plotly.express. scatter (data_frame=None, x=None, y=None, color=None, symbol=None, size=None, hover_name=None, hover_data=None, custom_data=None, text=None, facet_row=None, facet_col=None, facet_col_wrap=0, facet_row_spacing=None, facet_col_spacing=None, error_x=None, error_x_minus=None, error_y=None, error_y_minus=None, animation_frame=None, animation_group=None, category_orders={}, labels={}, orientation=None, color_discrete_sequence=None, color_discrete_map={}, color_continuous_scale=None, range_color=None, color_continuous_midpoint=None, symbol_sequence=None, symbol_map={}, opacity=None, size_max=None, marginal_x=None, marginal_y=None, trendline=None, trendline_color_override=None, log_x=False, log_y=False, range_x=None, range_y=None, render_mode='auto', title=None, template=None, width=None, height=None)

Correlation plot

A correlation plot is a way to visualize correlations between variables.

The Pearson Correlation Coefficient summarizes this relationship

- Has a value -1 to 1
- 1 is totally positively correlated
  - As x increases, y increases
- 0 is not at all correlated
  - No relationship between x and y
- -1 is totally negatively correlated
  - As x increases, y decreases

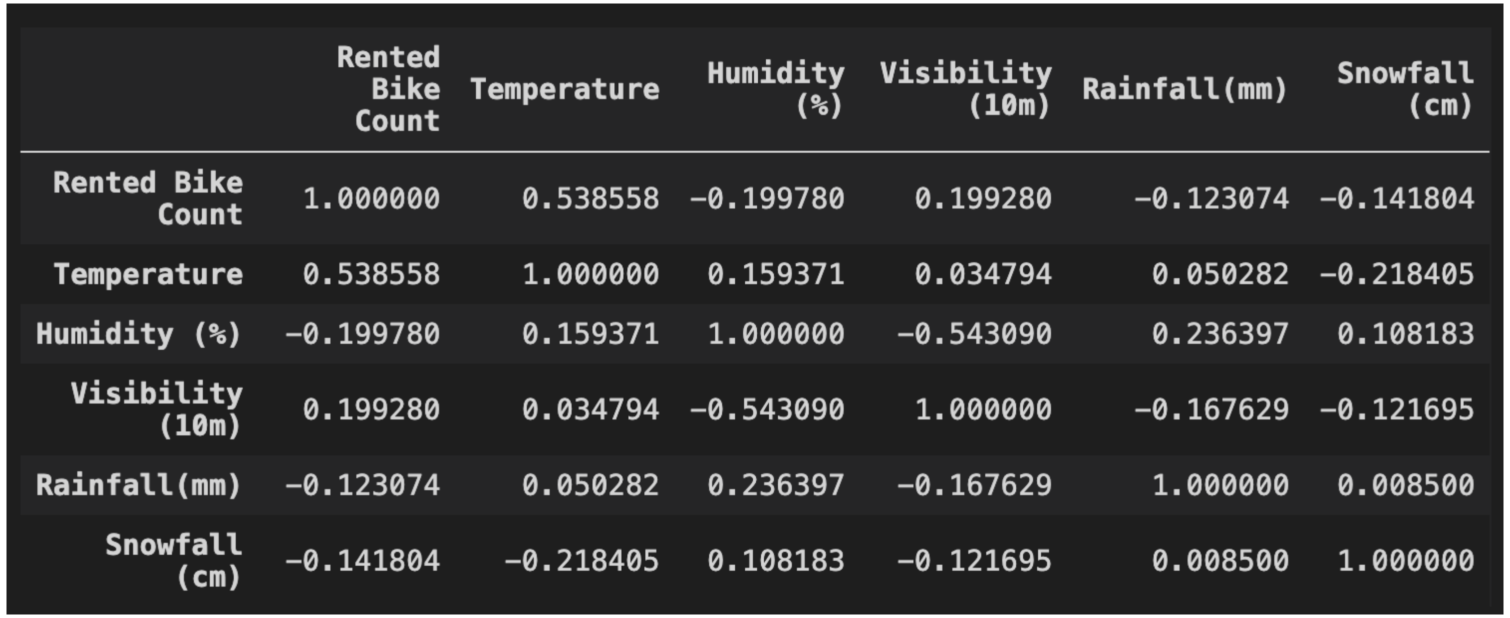
Correlation plot setup

`df` contains data on bike sharing rental numbers in Korea with various weather variables

`pandas` provides a method to create the data needed:

```
cr = df.corr(method='pearson')
print(cr)
```

Our Pearson correlation table:



	Rented Bike Count	Temperature	Humidity (%)	Visibility (10m)	Rainfall(mm)	Snowfall (cm)
Rented Bike Count	1.000000	0.538558	-0.199780	0.199280	-0.123074	-0.141804
Temperature	0.538558	1.000000	0.159371	0.034794	0.050282	-0.218405
Humidity (%)	-0.199780	0.159371	1.000000	-0.543090	0.236397	0.108183
Visibility (10m)	0.199280	0.034794	-0.543090	1.000000	-0.167629	-0.121695
Rainfall(mm)	-0.123074	0.050282	0.236397	-0.167629	1.000000	0.008500
Snowfall (cm)	-0.141804	-0.218405	0.108183	-0.121695	0.008500	1.000000

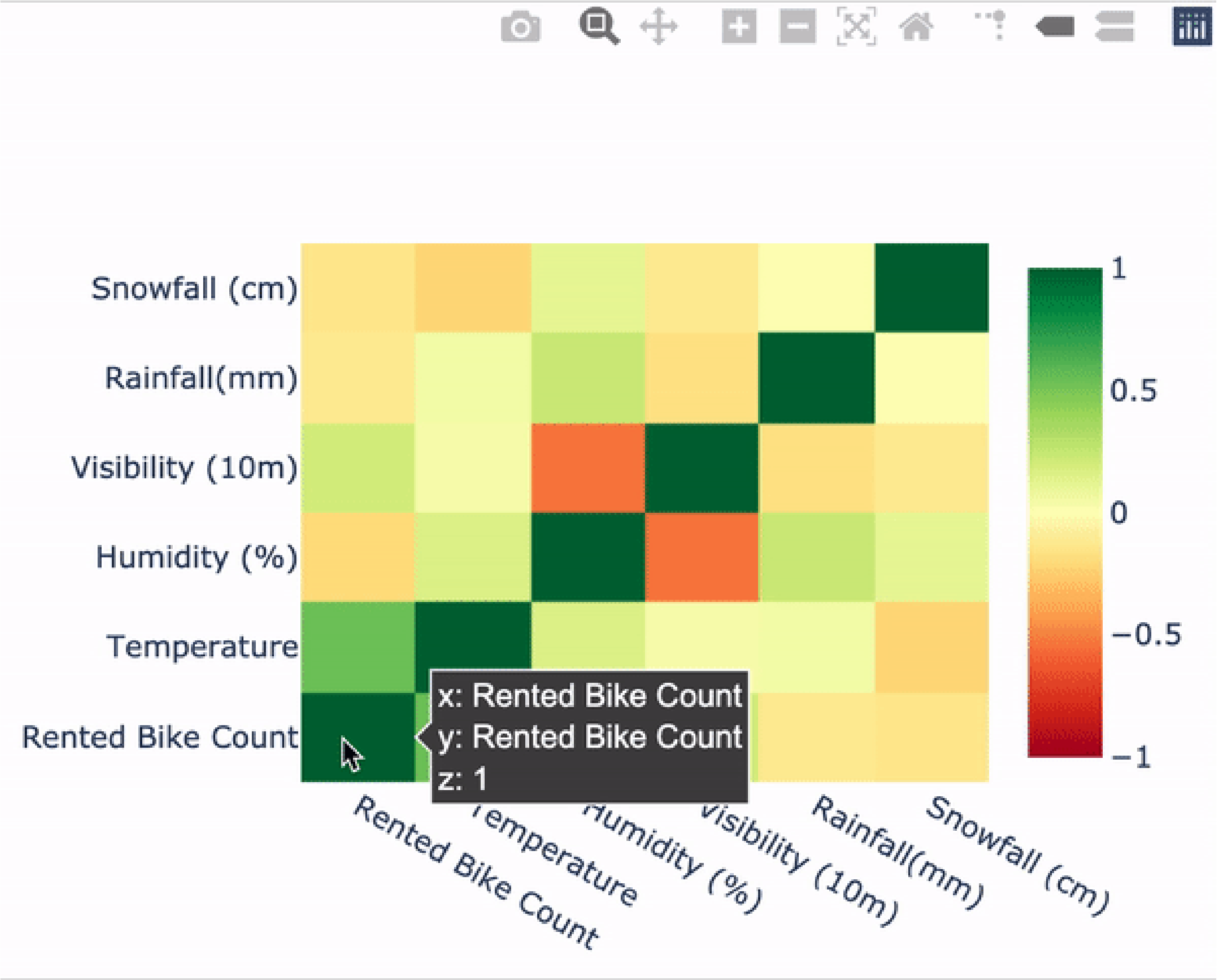
Correlation plot with Plotly

Let's build a correlation plot:

```python
import plotly.graph_objects as go
fig = go.Figure(go.Heatmap(
        x=cr.columns,
        y=cr.columns,
        z=cr.values.tolist(),
        colorscale='rdylgn', zmin=-1, zmax=1))
fig.show()
```

Our correlation plot

Voila!



Let's practice!

INTRODUCTION TO DATA VISUALIZATION WITH PLOTLY IN PYTHON

Customizing hover information and legends

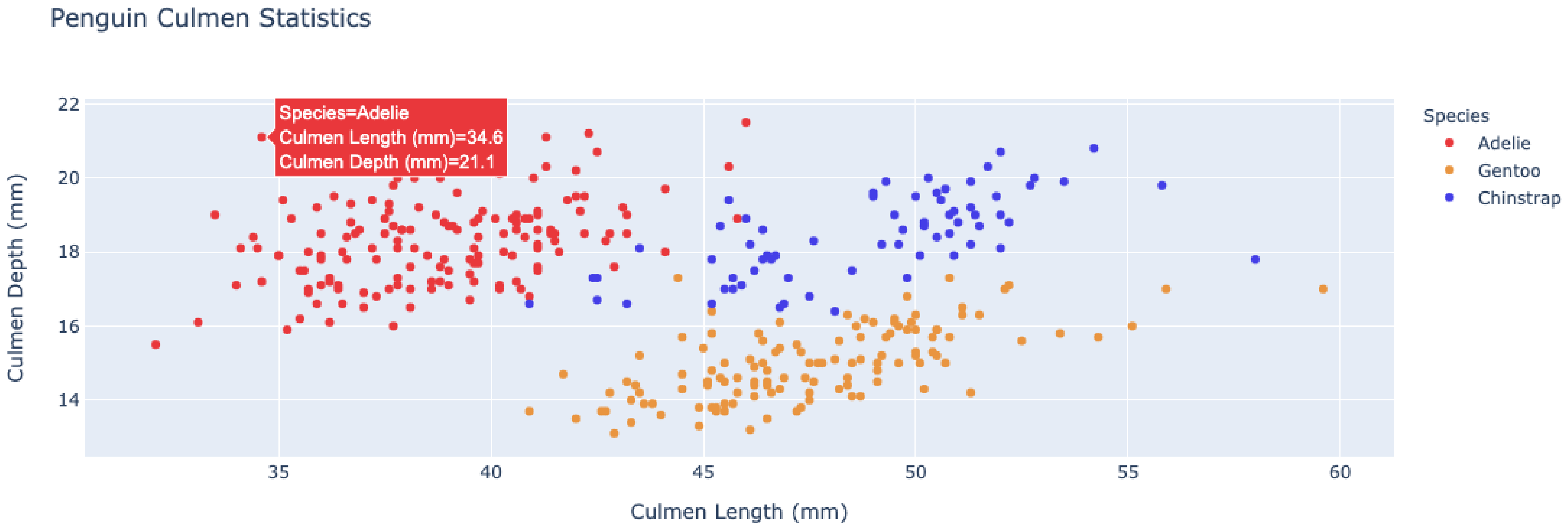
INTRODUCTION TO DATA VISUALIZATION WITH PLOTLY IN PYTHON

Alex Scriven
Data Scientist

What do we mean by hover?

Hover information: The text and data that appears when your mouse hovers over a data point in a Plotly visualization.

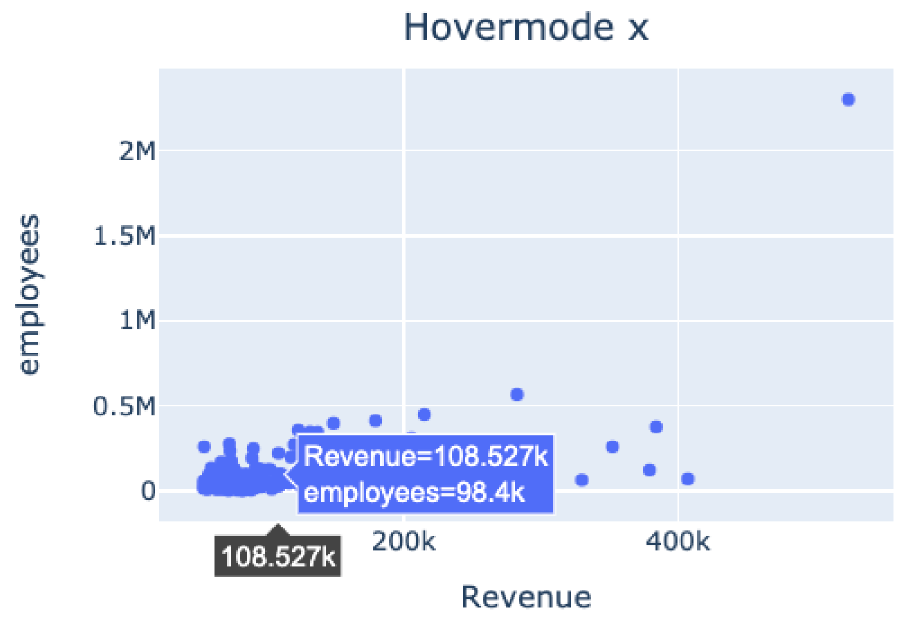
By default, you get some hover information already:

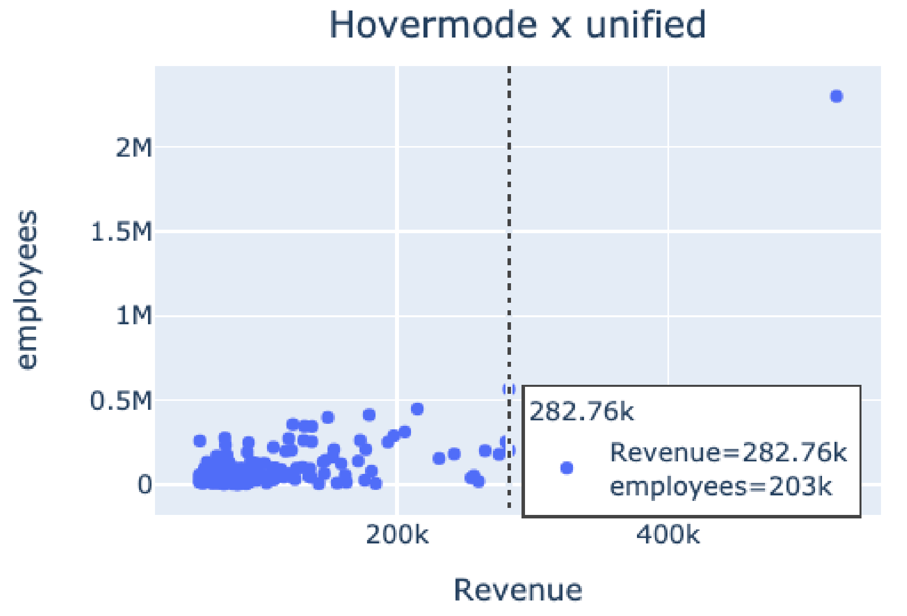


Other default hover information

The relevant layout argument is `hovermode` , which can be set to different values:

- `x` or `y` : adds a highlight on the x or y axis
- `x unified` / `y unified` : A dotted line appears on the relevant axis (`x` here) and the hover-box is formatted





Hover information using plotly.express

Customizing hover data in `plotly.express` :

- `hover_name` = A specified column that will appear in bold at the top of the hover box
- `hover_data` = A **list** of columns to include or a **dictionary** to include/exclude columns
    - `{column_name: False}` (this will exclude `column_name`)

No extensive formatting options

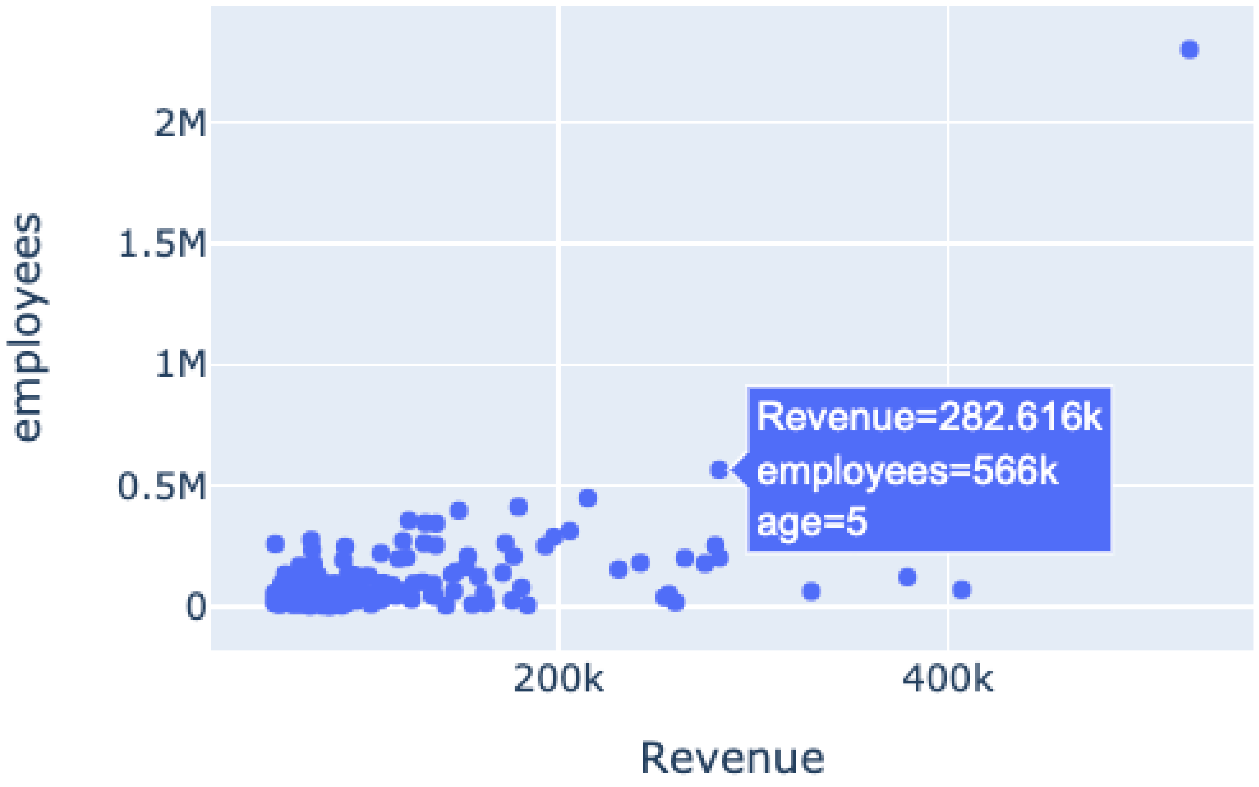
Variables in hover information

Hover columns don't need to be in the plot!

- E.g.: Revenue vs. company size with age of company displayed on hover

```
fig = px.scatter(revenues,
        x="Revenue",
        y="employees",
        hover_data=['age'])
fig.show()
```

We can see `age` in the hover!



Styling hover information

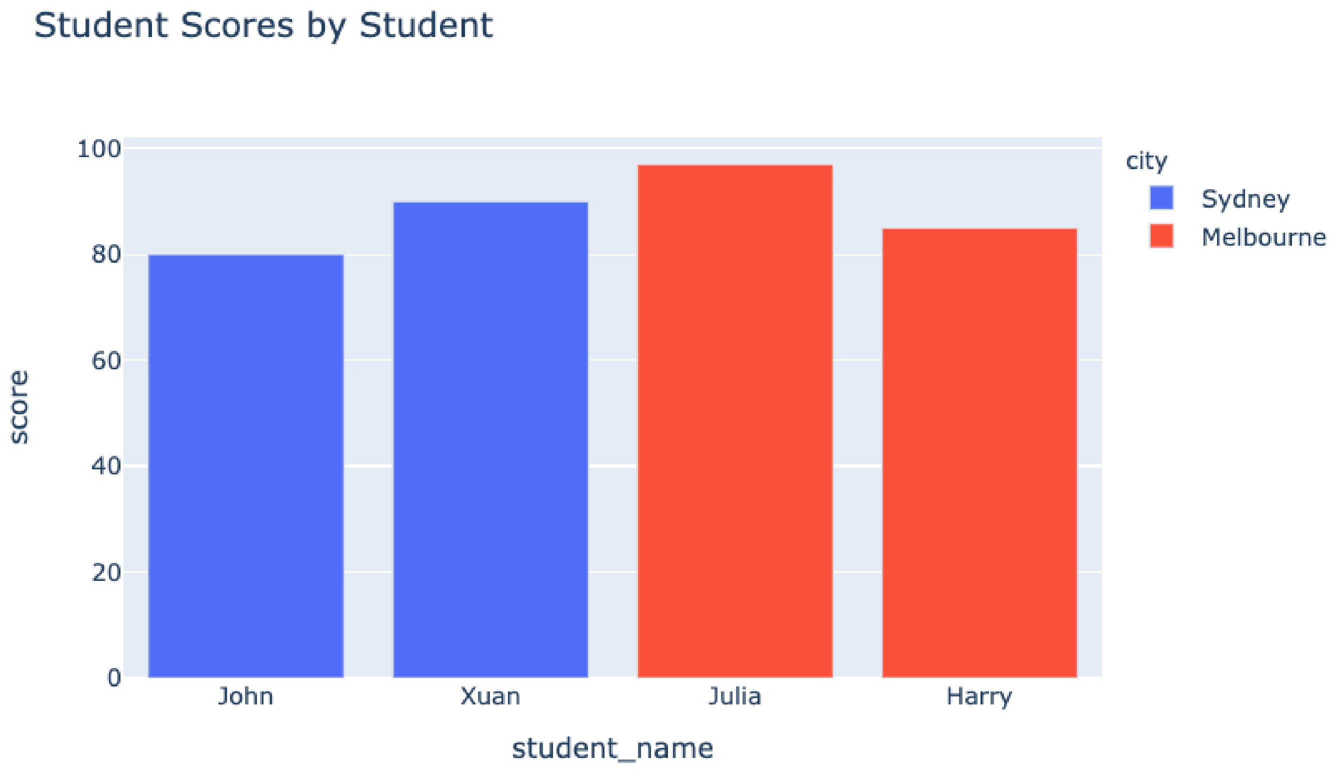
There are two main ways to style hover information:

1. Using the `hoverlabel` layout element
    - A dictionary of stylistic properties (background colors, borders, font, sizings, etc.)
2. Using the `hovertemplate` layout element
    - An HTML-like string to style the text (beyond this course)

What is a legend?

A legend is an information box that provides a key to the elements inside the plot, particularly the color or style.

- Legends often automatically appear with plotly.
  - For example, when adding colors to our bar chart



Creating and styling the legend

You can turn on and style the legend using `update_layout()`

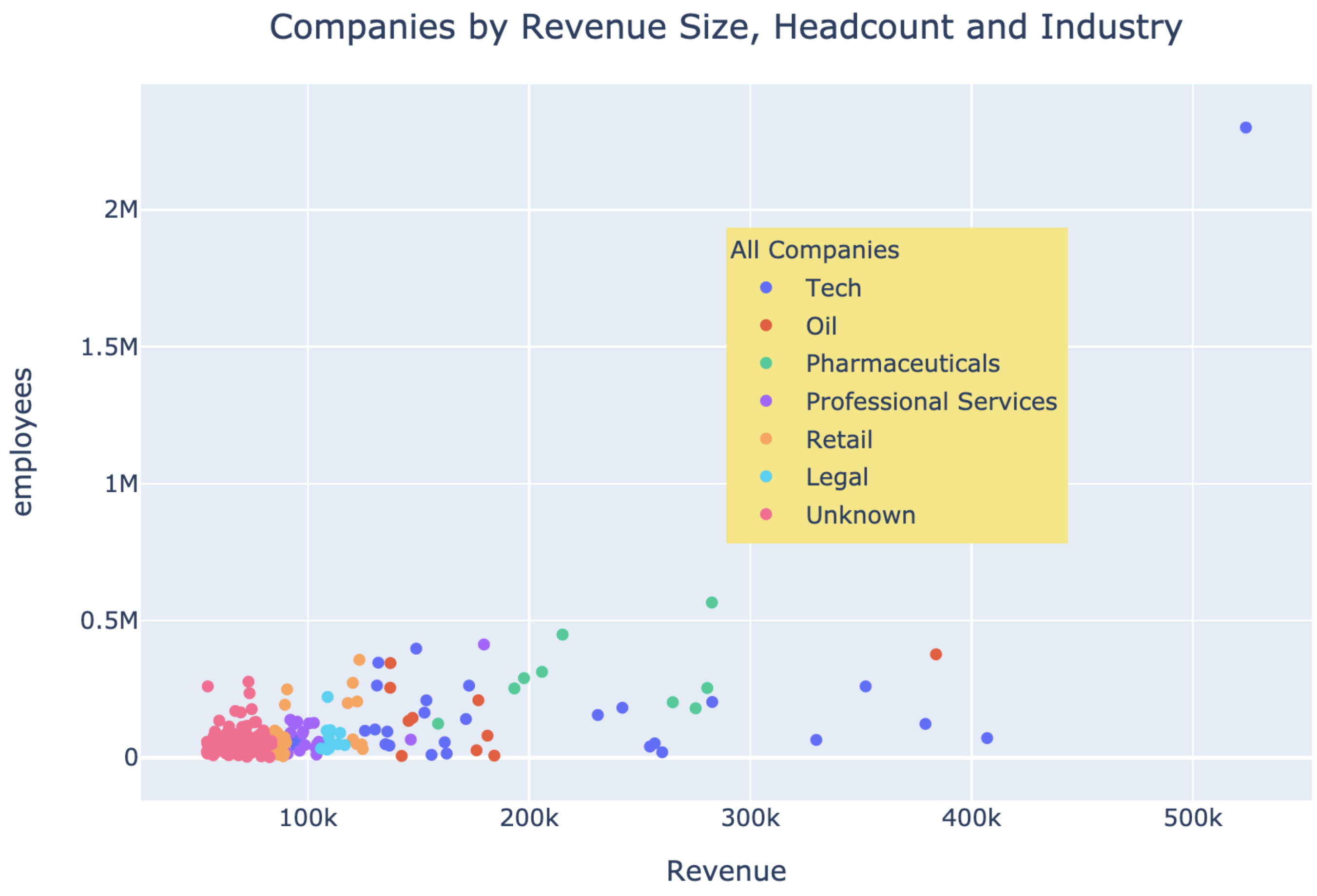
- `showlegend` = `True` shows the default legend
- `legend` = a dictionary specifying styles and positioning of the legend
  - `x` , `y` : (0-1) the percentage across x or y axis to position
  - Other stylistic elements such as `bgcolor` (background color), `borderwidth` , `title` , and `font`

As always - check the documentation (link) for more!

A styled legend

We can create a styled legend and position it:

```
fig.update_layout({
    'showlegend': True,
    'legend': {
      'title': 'All Companies',
      'x': 0.5, 'y': 0.8
      'bgcolor': 'rgb(246,228,129)'}
})
```



Let's practice!

INTRODUCTION TO DATA VISUALIZATION WITH PLOTLY IN PYTHON

Adding annotations

INTRODUCTION TO DATA VISUALIZATION WITH PLOTLY IN PYTHON

Alex Scriven
Data Scientist

What are annotations?

Annotations are extra boxes of text and data added to a plot.

Unlike hover information, annotations are always present.

They serve two primary purposes:

1. Data-linked annotations (draw attention, add notes) on a particular point
2. Add extra notes to a plot,
   - Much like adding a text-box in Microsoft Word

Creating annotations

In Plotly you can add annotations in several ways:

1. Using `add_annotation()`
   - Adds a **single annotation**
2. Using `update_layout()` and the `annotations` argument
   - A list of `annotation` objects
   - Useful if adding many annotations

For consistency, we'll stick with `update_layout()`

Important annotation arguments

There are several key elements of an `annotation` (dictionary) worth highlighting:

- `showarrow` = `True` / `False`
  - Determines whether an arrow will be drawn from the box to the given `x` / `y` coordinates
  - You can style the arrow as well!
- `text` = The actual text to be displayed
  - You can insert variables into this text too
- `x` and `y` : coordinates at which to place the annotation

Be careful placing annotations absolutely - if your data changes, things may overlap!

Positioning annotations

By default, the `x` and `y` arguments will be in the units of the plot to link to a data point.

However, you can position absolutely by:

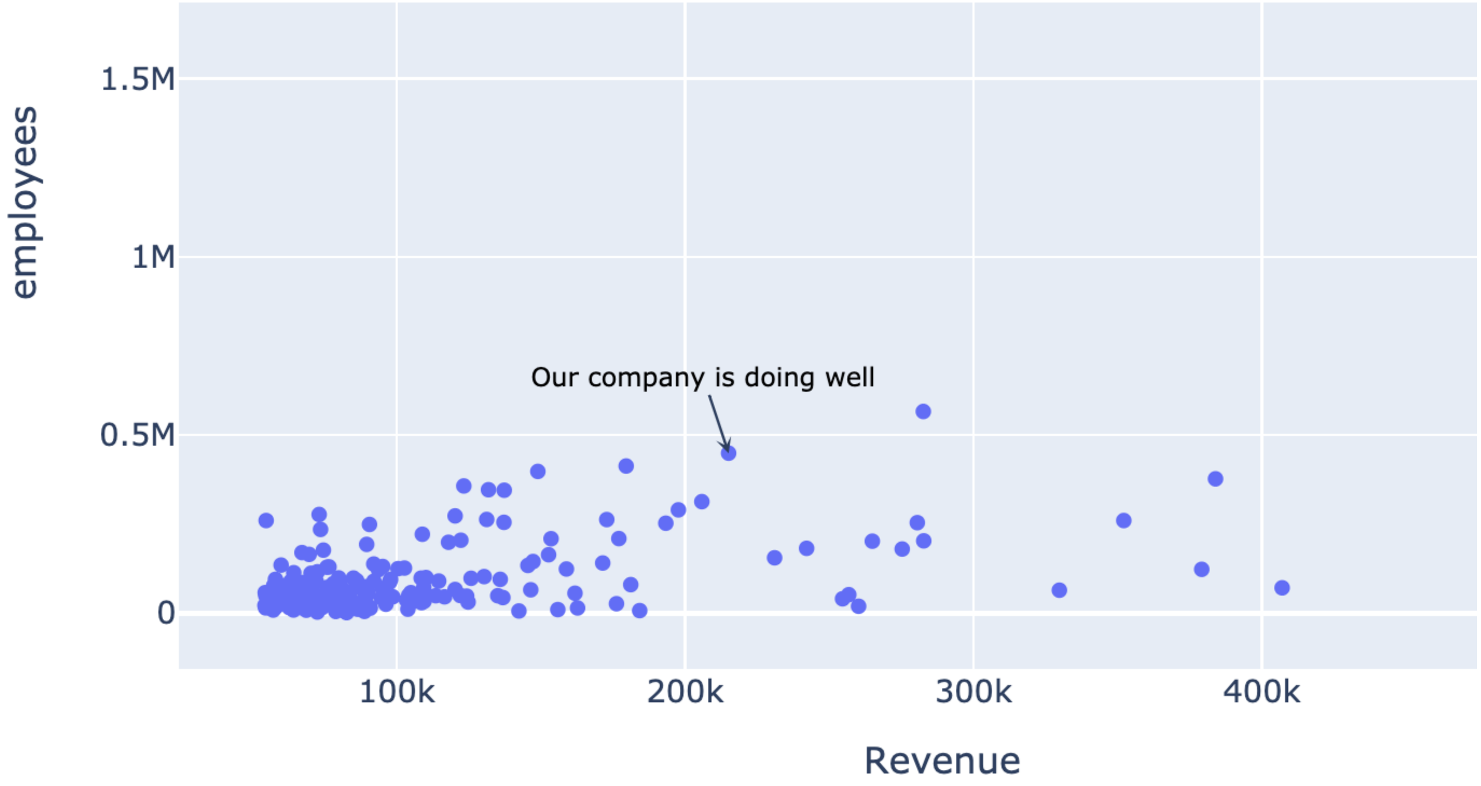
- Setting the arguments `xref` and `yref` to `paper`
  - Now the `x` and `y` parameters are 0-1 positions
  - A position of (`x=0.5` , `y=0.5`) would be right in the middle of the plot

Data-linked annotations

Let's annotate **our** company (we know the revenue and employee count) on our previous scatterplot.

```
my_annotation = {
    'x': 215111, 'y': 449000,
    'showarrow': True,'arrowhead': 3,
    'text': "Our company is doing well",
    'font' : {'size': 10, 'color': 'black'}}
fig.update_layout({'annotations': [my_annotation]})
fig.show()
```

Nice! We can see our company clearly:

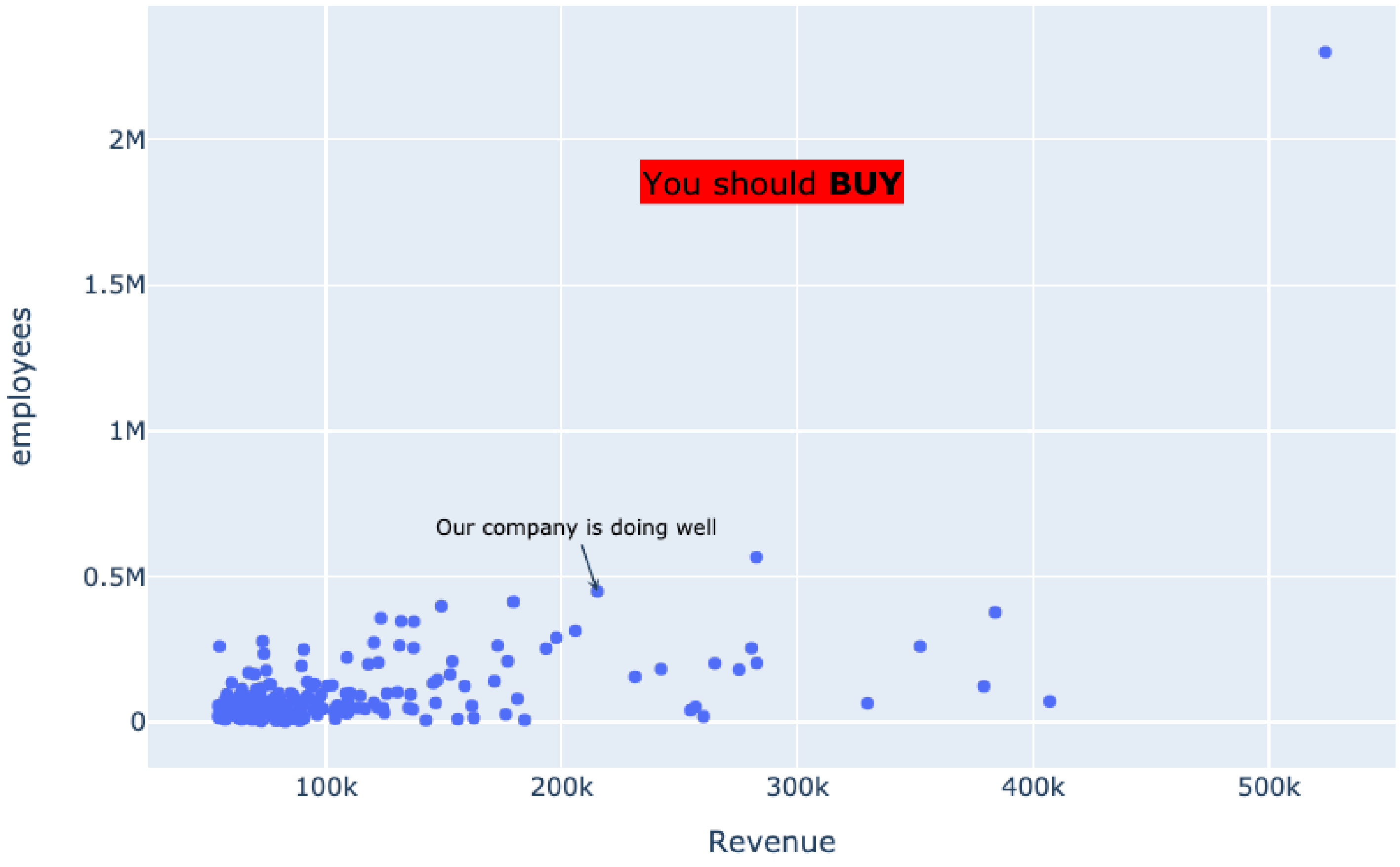


Floating annotation

We can also have a floating annotation, positioned absolutely.

```
float_annotation = {
    'xref': 'paper', 'yref': 'paper',
    'x': 0.5, 'y': 0.8,
    'showarrow': False,
    'text': "You should <b>BUY<b>",
    'font' : {'size': 15,'color': 'black'},
    'bgcolor': 'rgb(255,0,0)'}
```

We get a strong message!



Let's practice!

INTRODUCTION TO DATA VISUALIZATION WITH PLOTLY IN PYTHON

Editing plot axes

INTRODUCTION TO DATA VISUALIZATION WITH PLOTLY IN PYTHON

Alex Scriven
Data Scientist

Our dataset

Using the penguins dataset, let's aggregate flipper size by species:

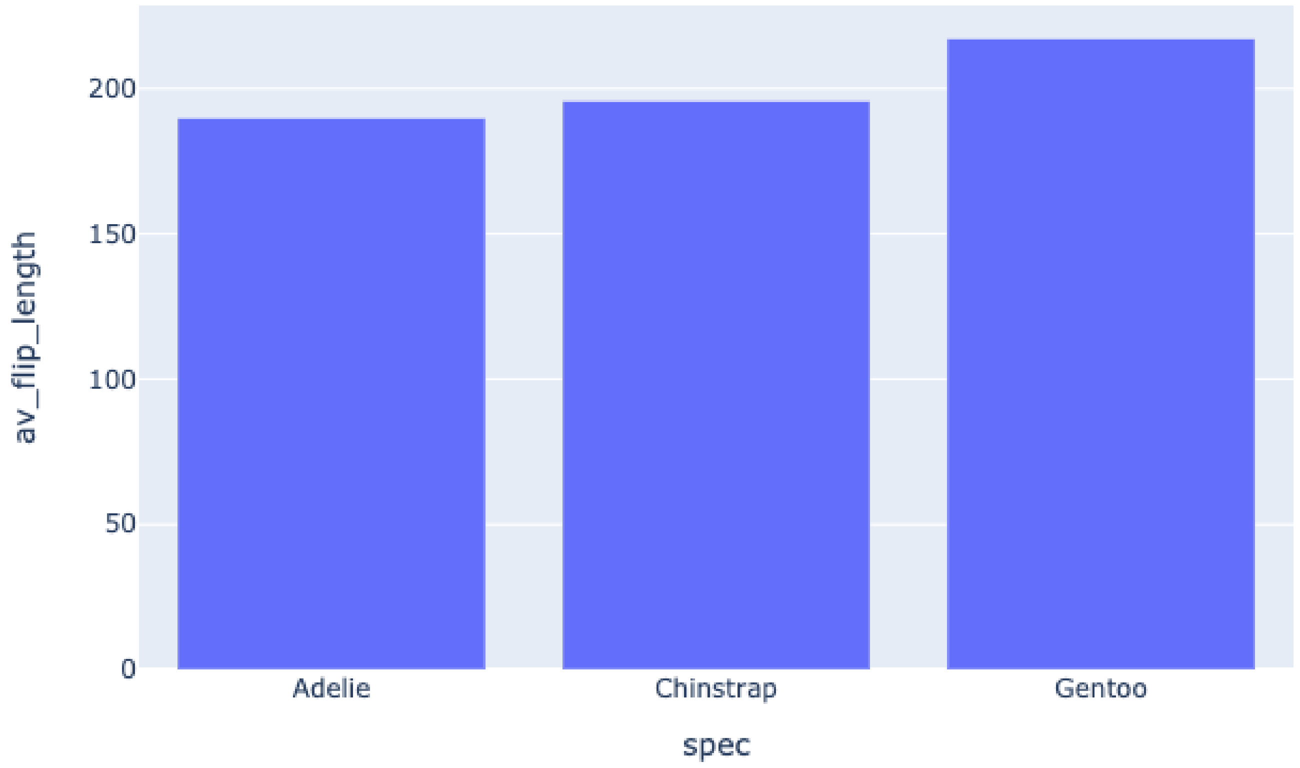
spec	av_flip_length
Adelie	189.953642
Chinstrap	195.823529
Gentoo	217.186992

Those columns aren't labeled well for presentation!

The default axis titles

Let's create a simple bar chart:

```
fig = px.bar(penguin_flippers,
            x='spec',
            y='av_flip_length')
fig.show()
```



This works, but those axes titles aren't great.

Editing axis titles

`plotly` often has 'shortcut' functions:

```
fig.update_xaxes(title_text='Species')
fig.update_yaxes(title_text='Average Flipper Length')
```

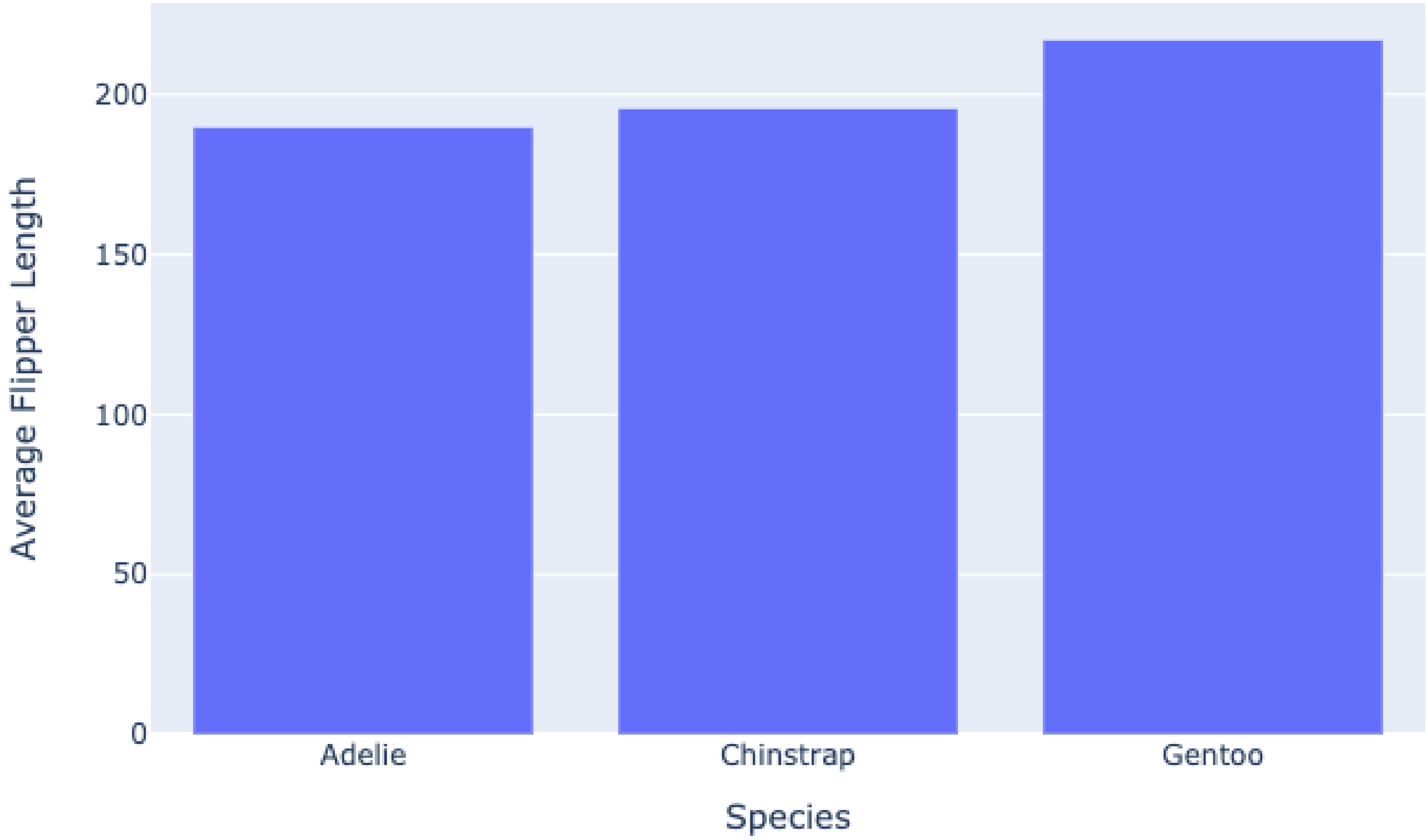
Or with the more general `update_layout()`

```
fig.update_layout('xaxis': {'title': {'text': 'Species'}},
                  'yaxis': {'title':{'text': 'Average Flipper Length'}})
```

We will stick with `update_layout()` for consistency

Cleaning up our plot

Both methods will produce a more presentation-worthy chart.



Which method to use?

The shortcut method is helpful to quickly change just that one attribute.

To further style axes, the `update_layout()` method allows you to edit:

- Font family, font size
- Text angle
- Text color
- Much more!

See more on the **Plotly documentation**

Editing axes ranges

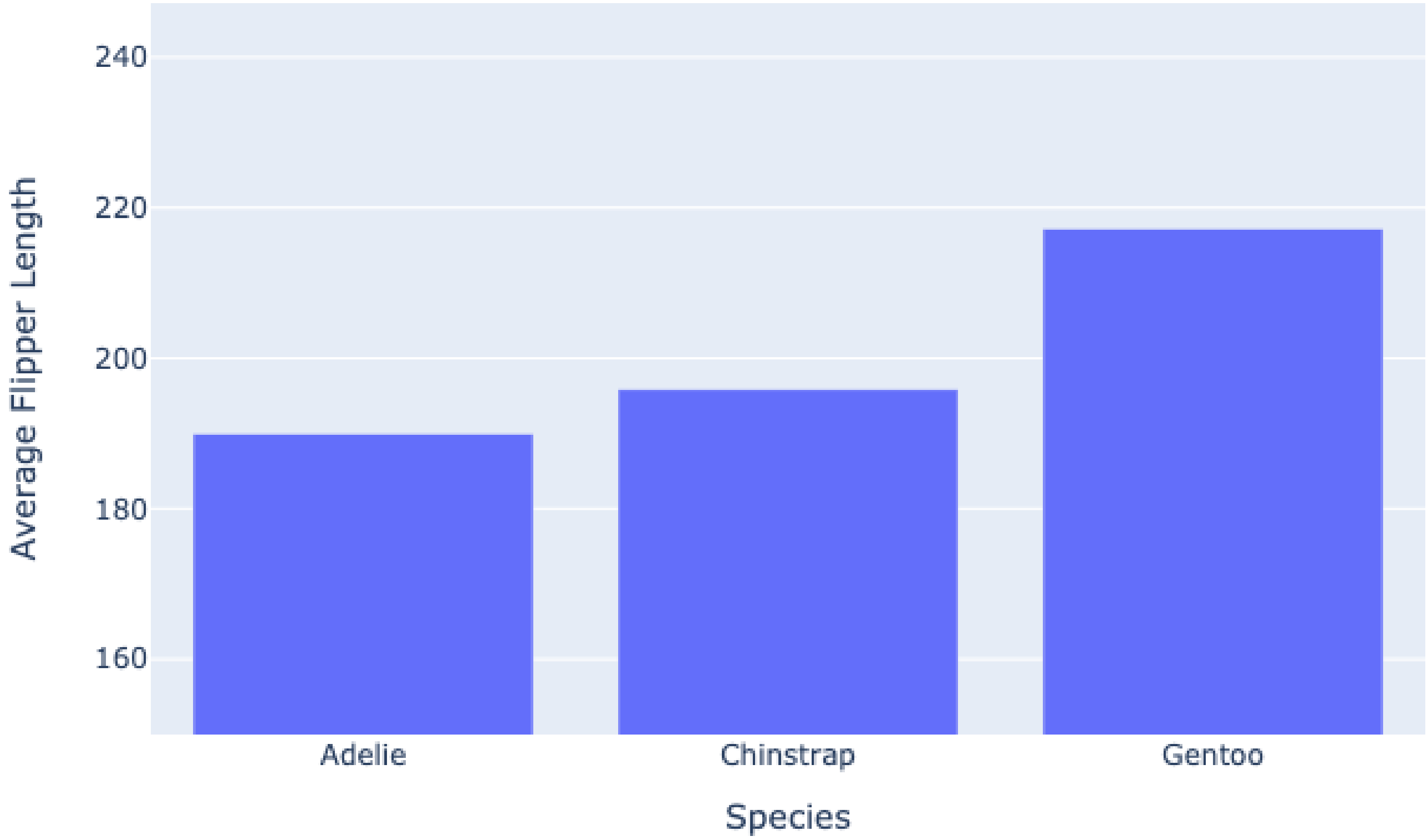
Plotly automatically calculates axes ranges from your data - this may not be desired!

Let's set the y-axis to start at 150 and go up to a small buffer (30) past the maximum flipper length

```
fig.update_layout({'yaxis':
           {'range' : [150,
                       penguin_flippers['av_flip_length'].max() + 30]}
})
```

Our new axes ranges

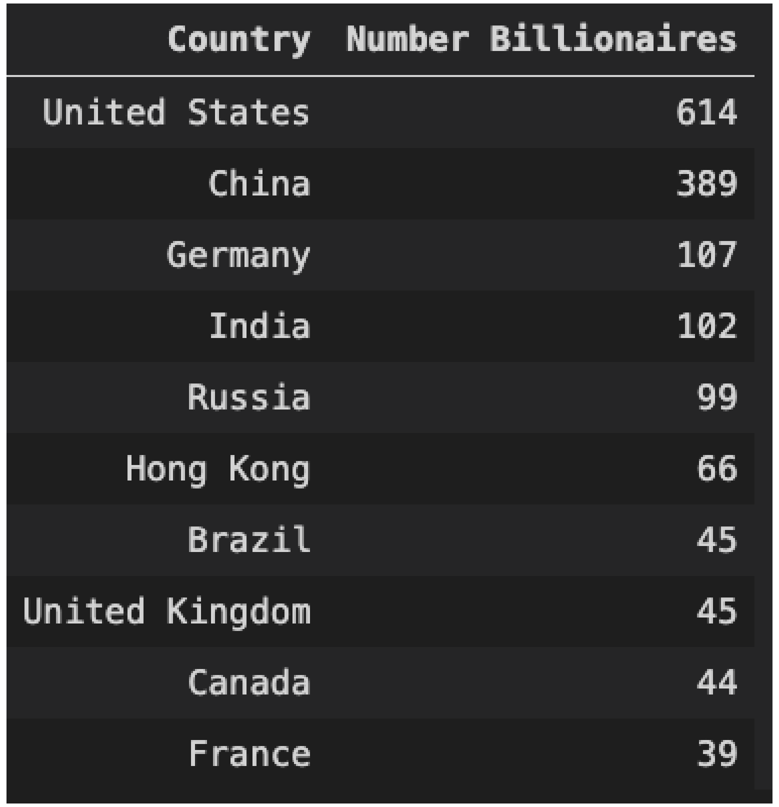
We get specific axes:



Data scale issues

What happens when some data points are **much** larger than others?

- Top 10 countries by number of billionaires

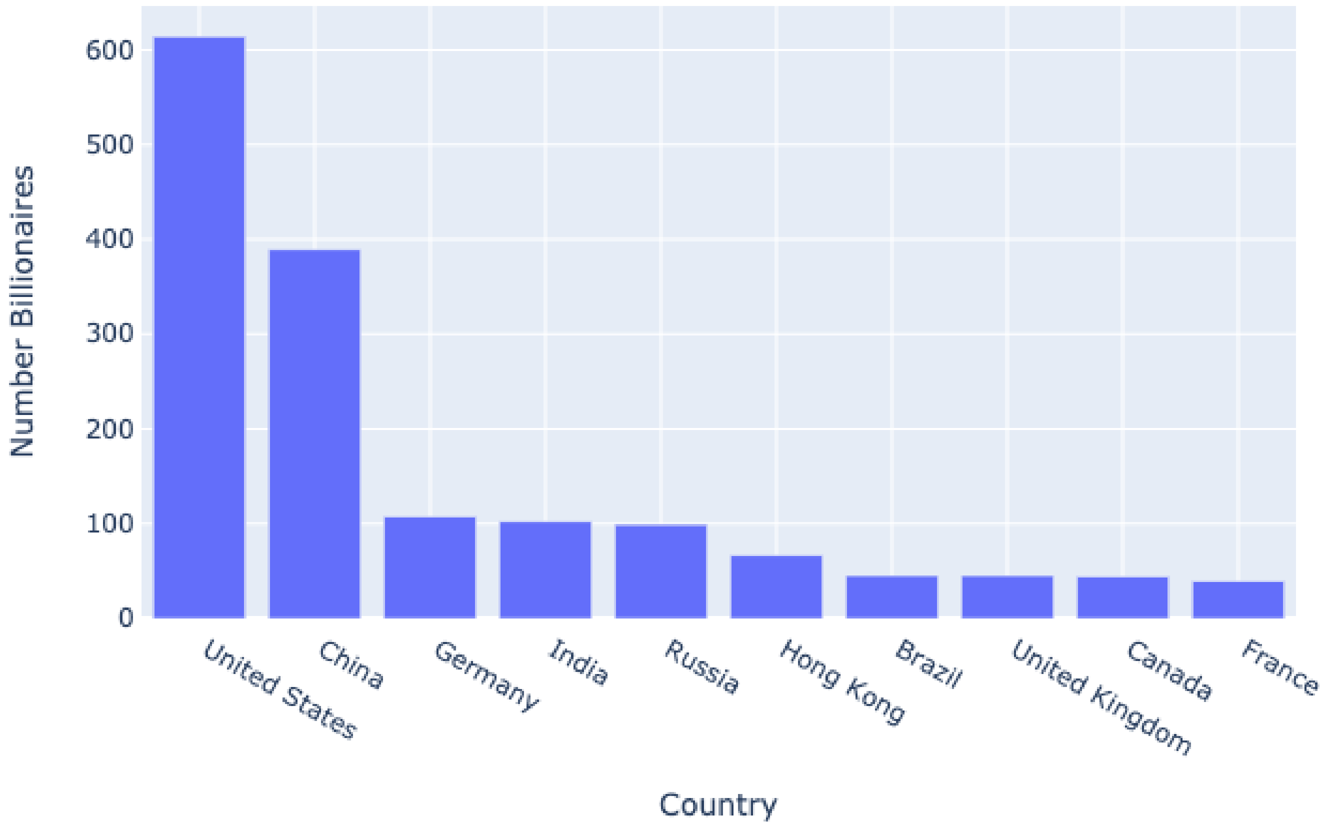


Country	Number Billionaires
United States	614
China	389
Germany	107
India	102
Russia	99
Hong Kong	66
Brazil	45
United Kingdom	45
Canada	44
France	39

Our scale problem

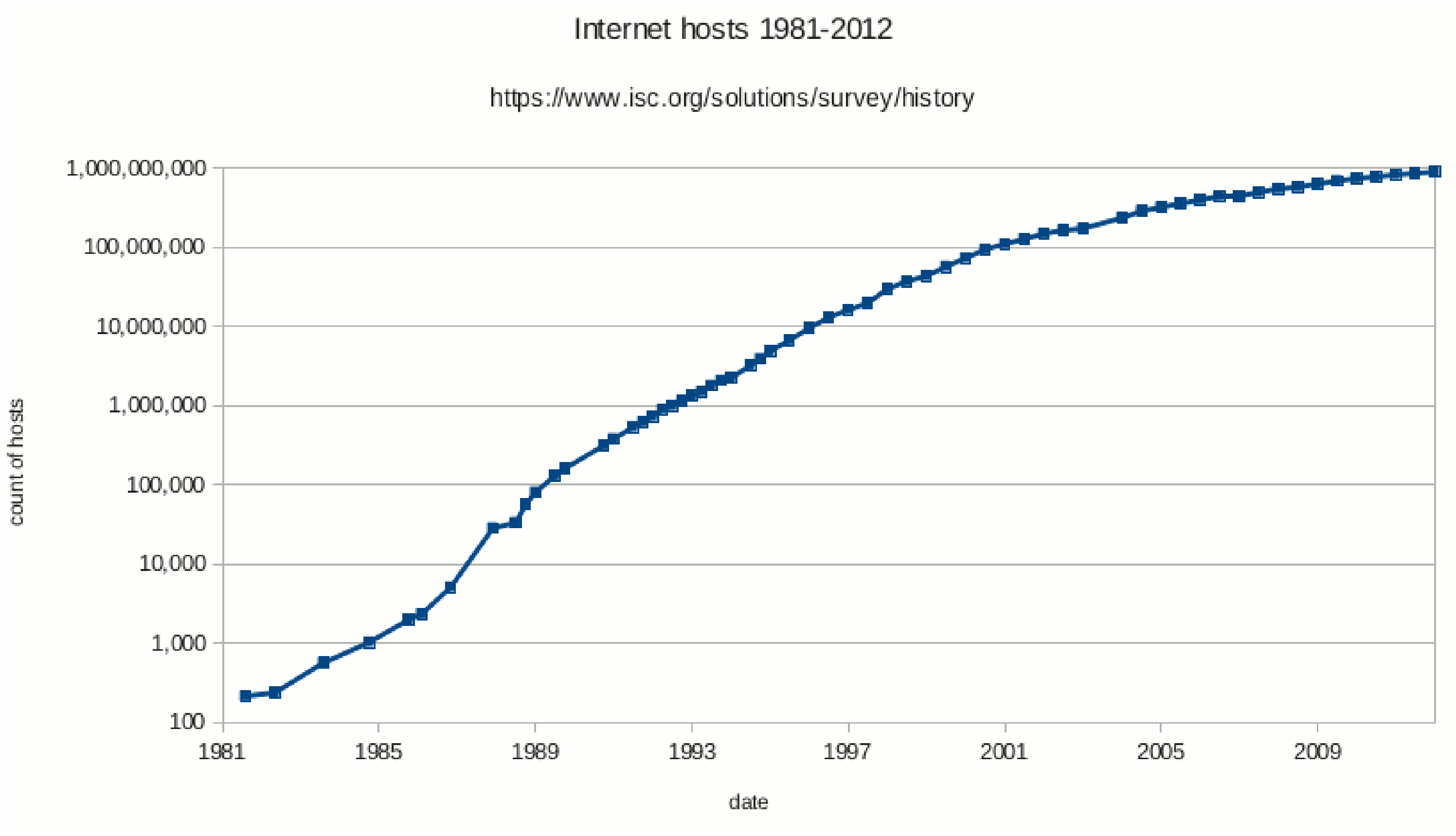
Let's plot without any adjustment:

```
fig = px.bar(billionaire_data,
             x='Country',
             y='Number Billionaires')
fig.show()
```



The log scale

- Common scale used to plot data with large value differences.
- It looks like this:



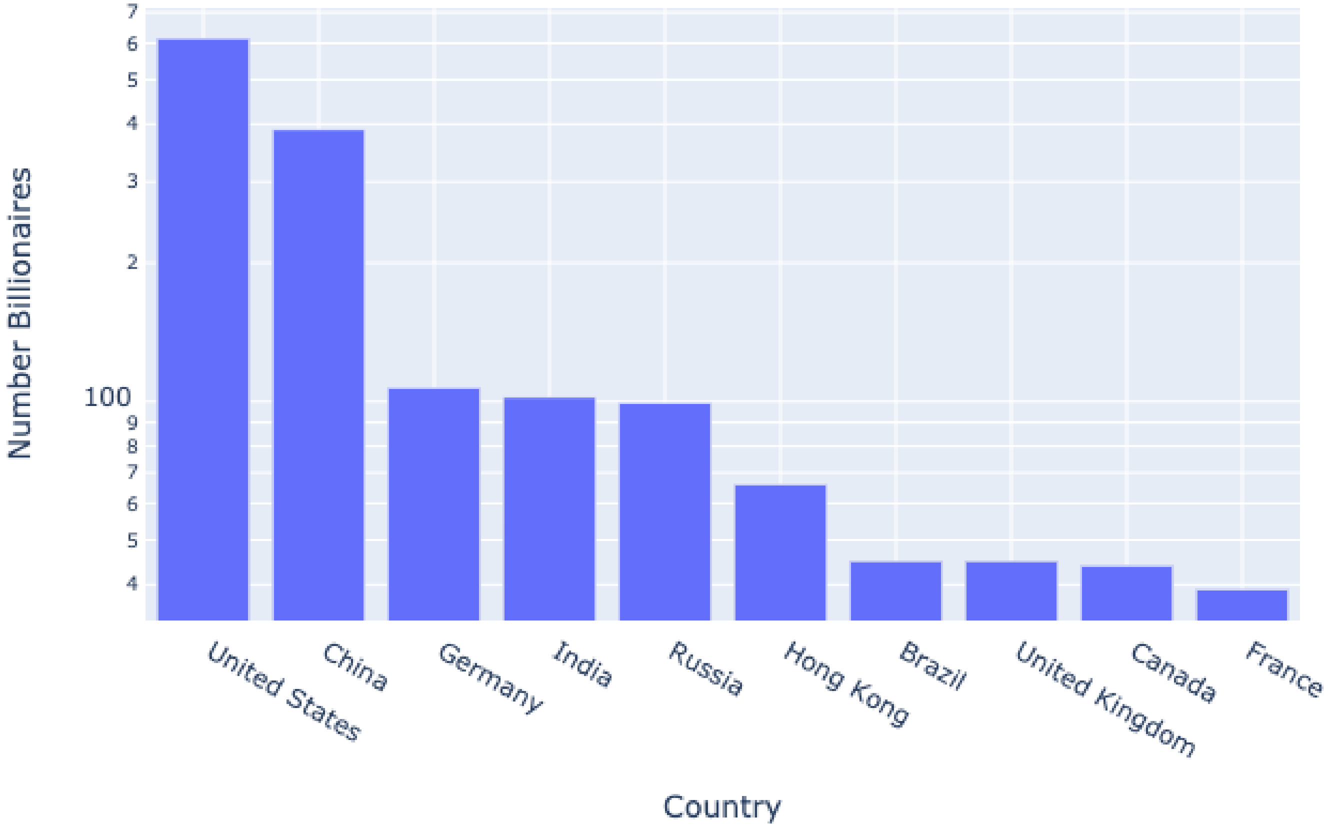
Ticks on our y-axis aren't uniform (10,20, 30, etc.)

Each tick is an *order of magnitude* bigger (10, 100, 1000, etc.)

Using log with our data

Plotly has `log_y` and `log_x` arguments

```
fig = px.bar(billionaire_data,
             x='Country',
             y='Number Billionaires',
             log_y=True)
fig.show()
```



That's better!

Log scale: a word of warning

When visualizing data, you are telling a *story*.

If your audience doesn't know what a `log` scale is, there may be miscommunication.

- So remember to keep your audience in mind!

Let's practice!

INTRODUCTION TO DATA VISUALIZATION WITH PLOTLY IN PYTHON

Subplots

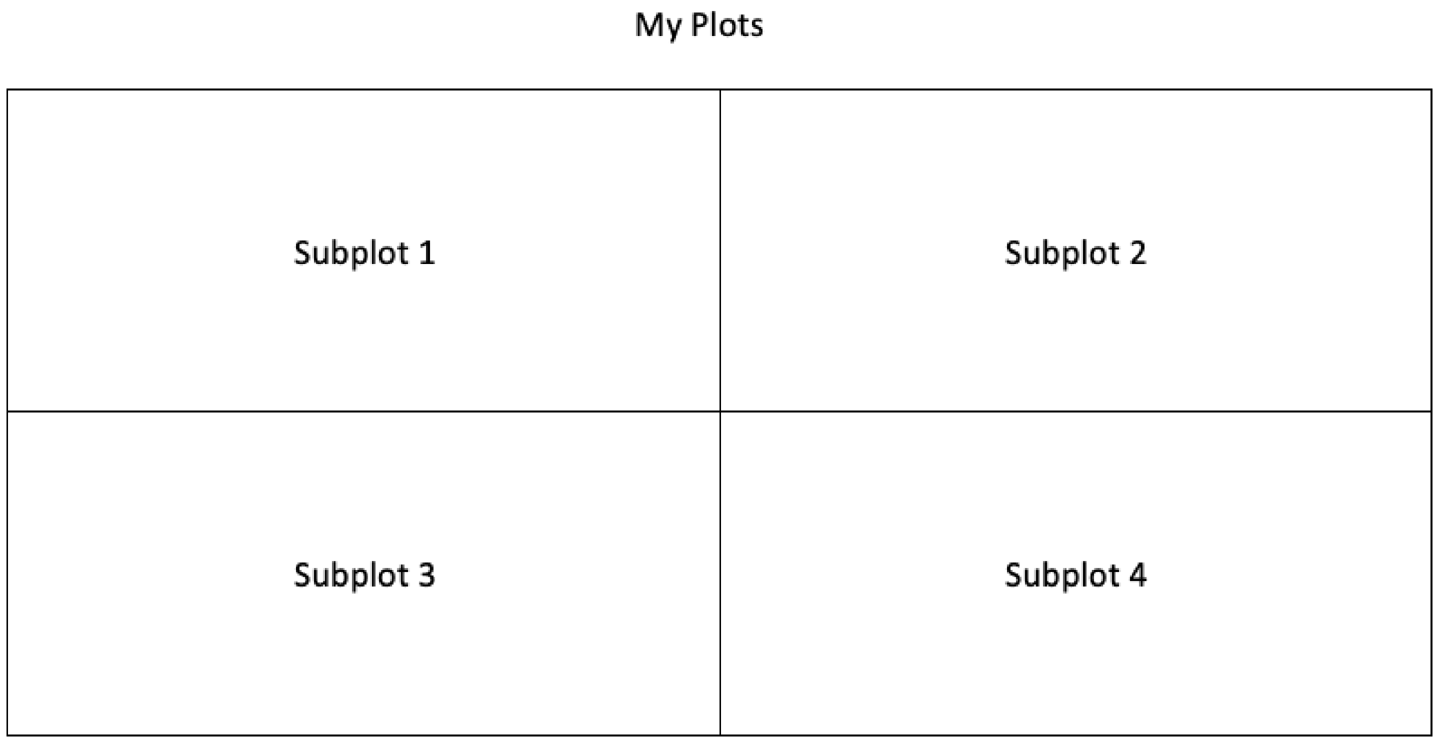
INTRODUCTION TO DATA VISUALIZATION WITH PLOTLY IN PYTHON

Alex Scriven
Data Scientist

What are subplots?

- Subplots: 'mini-plots' positioned in a grid arrangement
- Display different plot types (same data) or different data subsets
- Many are possible - but more will make each plot smaller!

For example:



My Plots

Subplot 1

Subplot 2

Subplot 3

Subplot 4

A reminder of traces

Remember discussing 'traces' earlier?

- Each set of `data` + graph `type` is a trace.
- You can build a plot by using `fig.add_trace(X)` where X is a `graph_objects` object (such as `go.Scatter()` or `go.Bar()`)
  - So far we haven't needed to do this.

To add data to each subplot, we will use `.add_trace()` .

graph_objects (go) vs plotly.express (px)

`graph_objects` and `plotly.express` often similar but have slight differences:

`add_trace()` takes `px` plots but the code is complex and not best-practice so we will use `go`

Check equivalent documentation for more help (`px` histogram vs `go` histogram)

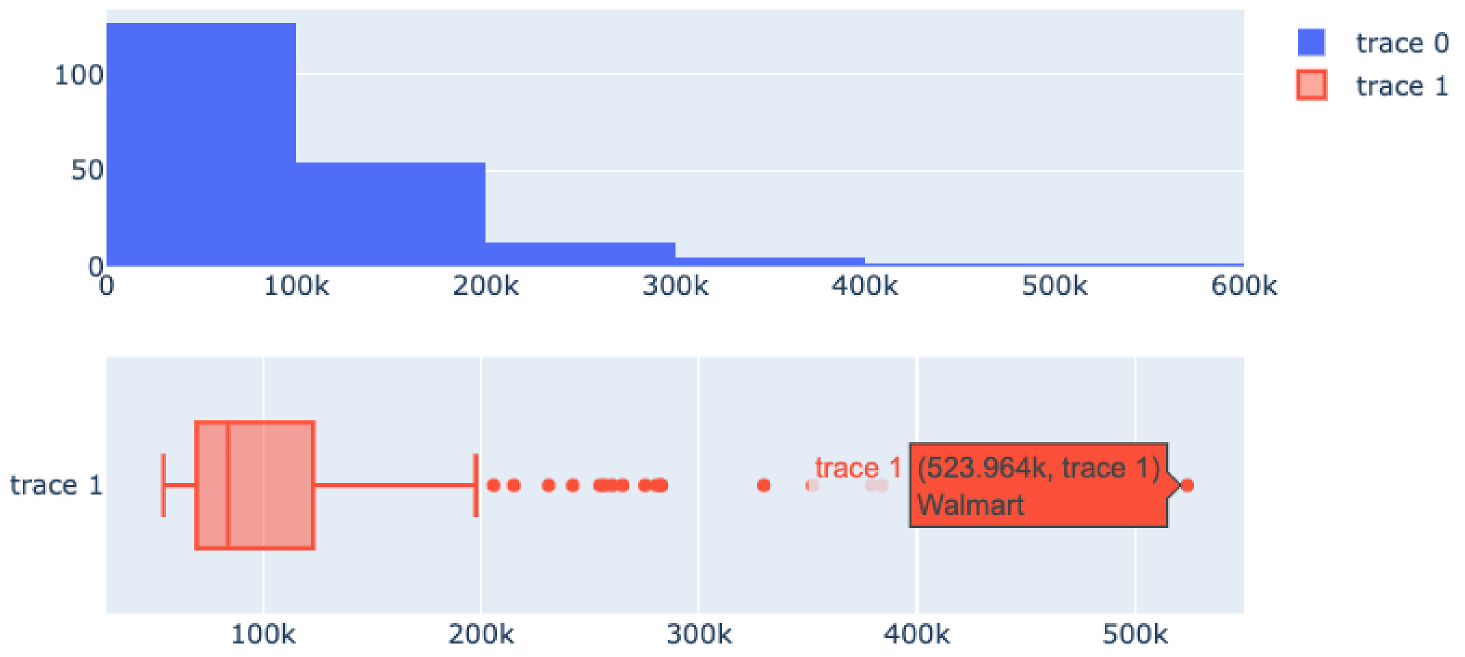
```
# With graph_objects
go.Histogram(x=revenues['Revenue'],
    nbinsx=5, name='Histogram')

# With plotly.express
px.histogram(data_frame=revenues,
    x='Revenue', nbins=5,
    title='Histogram')
```

Creating a 1x2 subplot

Let's build a 1x2 subplot (histogram + box plot) from the `revenues` DataFrame:

```
from plotly.subplots import make_subplots
fig = make_subplots(rows=2, cols=1)
fig.add_trace(
 go.Histogram(x=revenues['Revenue'], nbinsx
 row=1, col=1)
fig.add_trace(
 go.Box(x=revenues['Revenue'],
 hovertext=revenues['Company']),
 row=2, col=1)
fig.show()
```

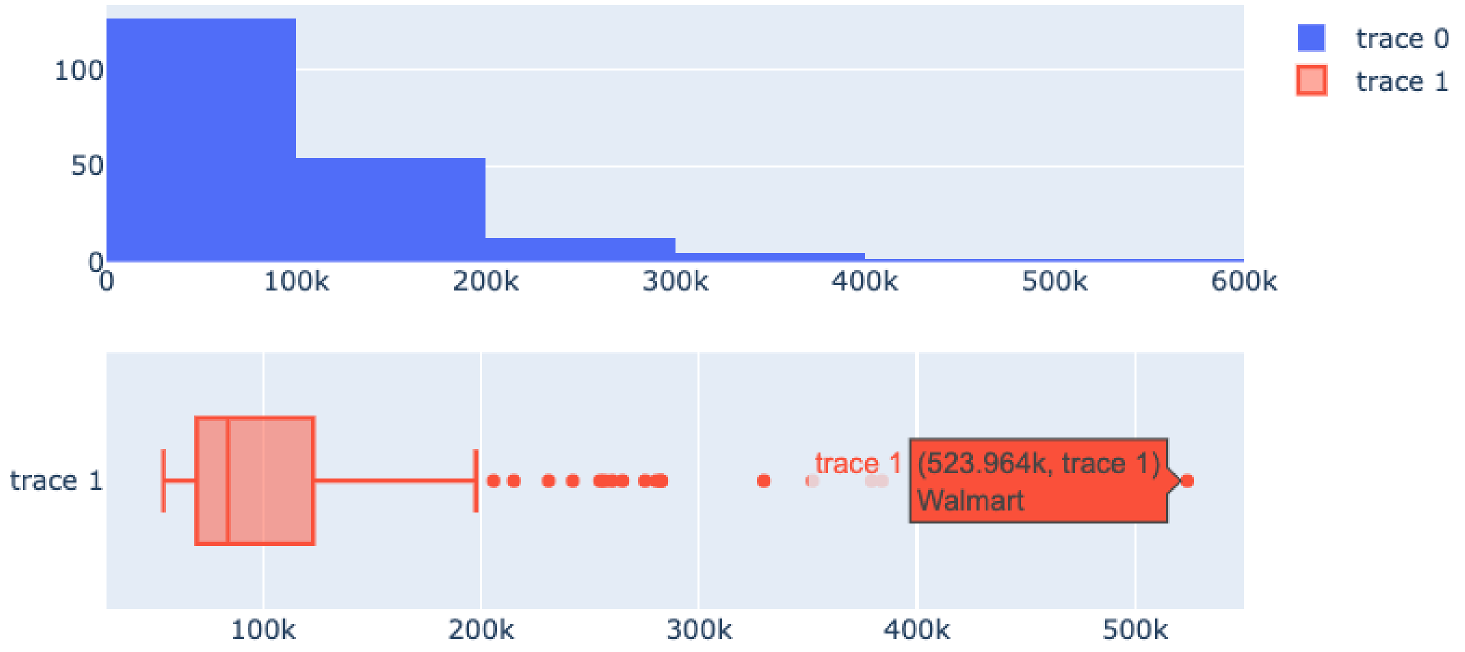
Our plots:



Customizing subplots

Some stylistic elements that need attention:

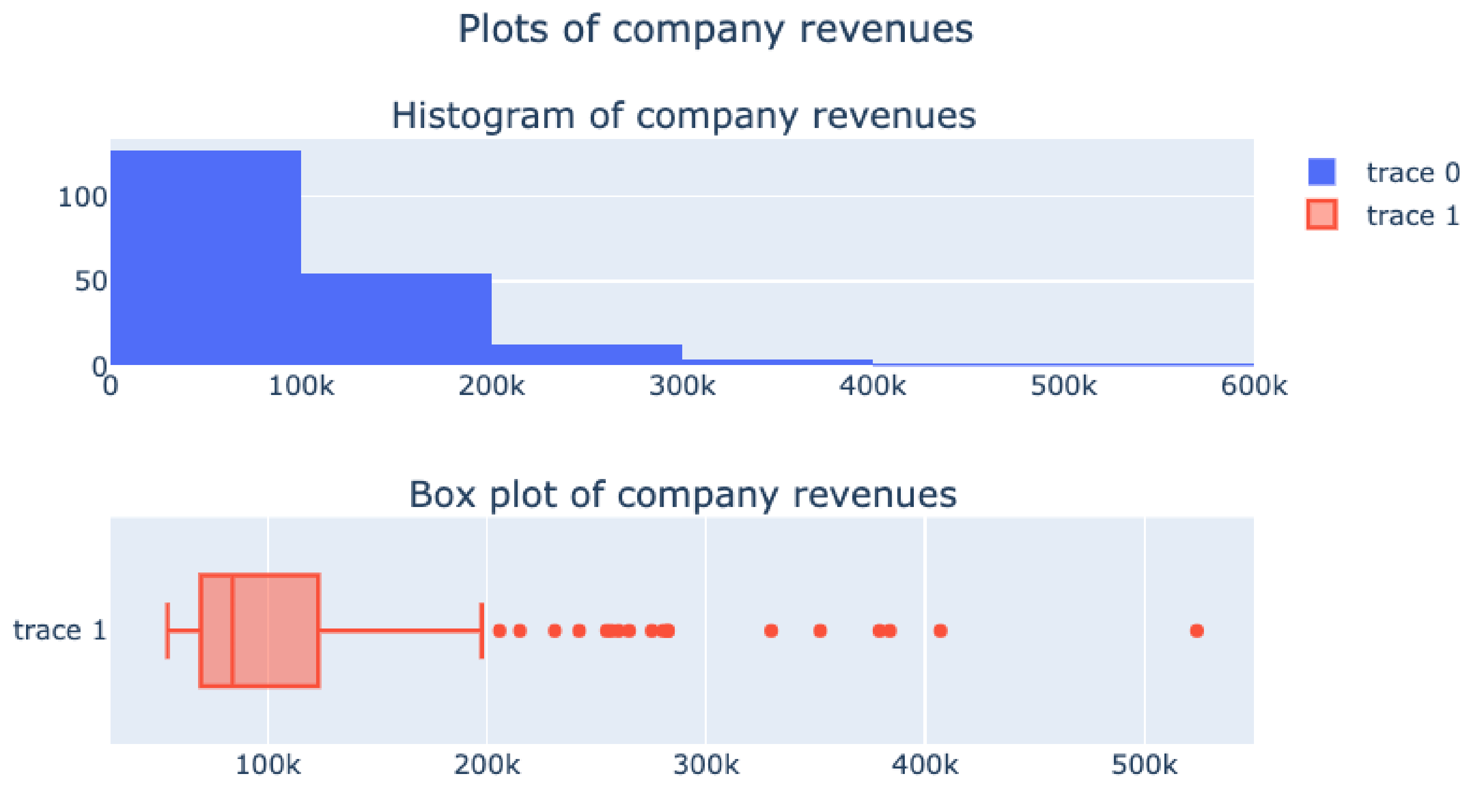
1. No overall plot title
2. No subplot titles
3. The legend says 'trace 1'/ 'trace 2'
4. Other customization skills!



Subplot titles

Let's fix the titles:

```
from plotly.subplots import make_subplots
fig = make_subplots(rows=2, cols=1,
    subplot_titles=[
    'Histogram of company revenues',
    'Box plot of company revenues'])
## Add in traces (fig.add_trace())
fig.update_layout({'title': {'text':
    'Plots of company revenues',
    'x': 0.5, 'y': 0.9}})
fig.show()
```

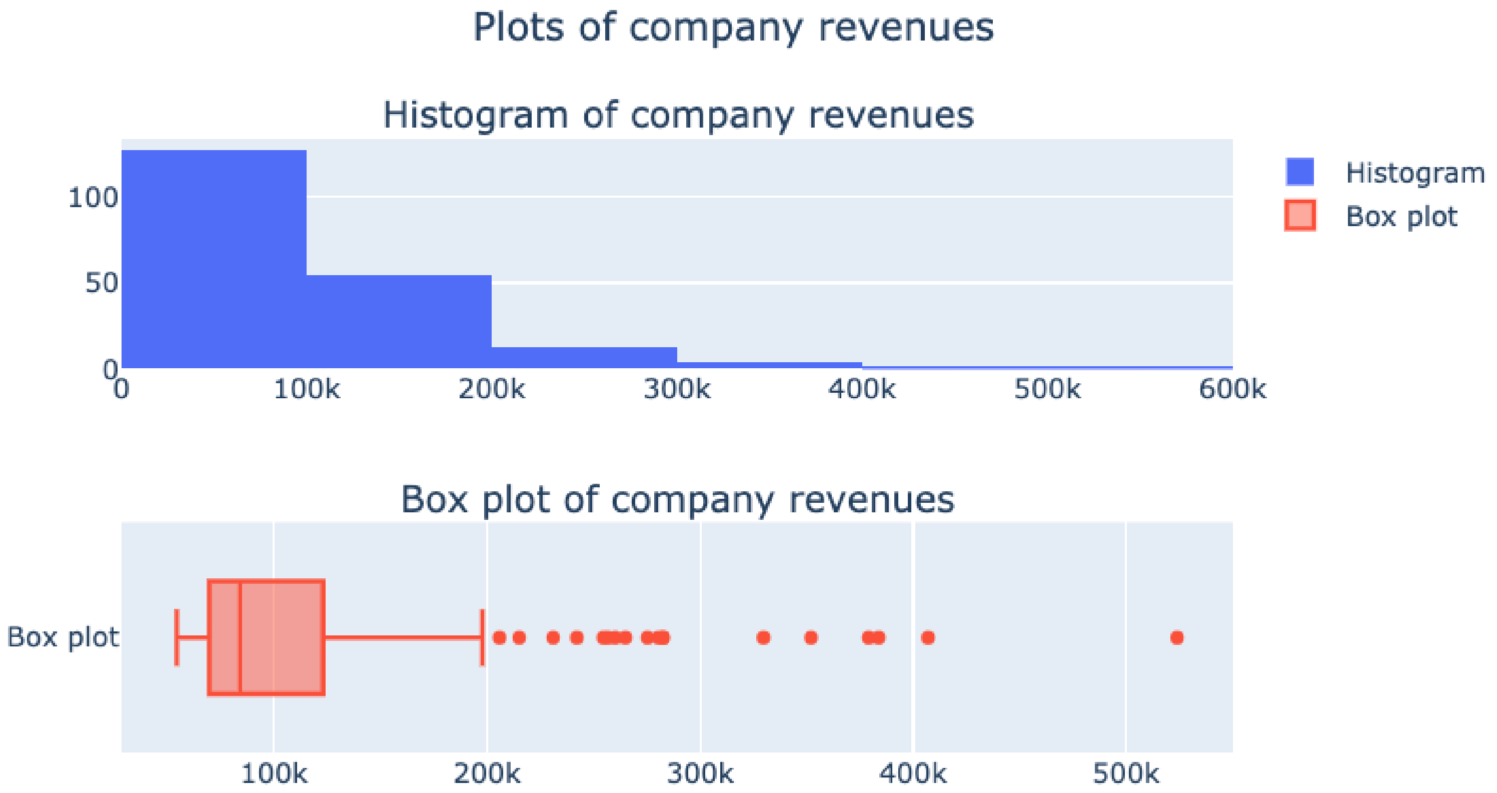


Note: More options available in the (**documentation**)

Subplot legends

Let's fix the legend names:

```
fig.add_trace(
  go.Histogram(x=revenues.Revenue,
  nbinsx=5, name='Histogram'),
  row=1, col=1)
fig.add_trace(
    go.Box(x=revenues.Revenue,
    hovertext=revenues['Company'],
    name='Box plot'),
    row=2, col=1)
```

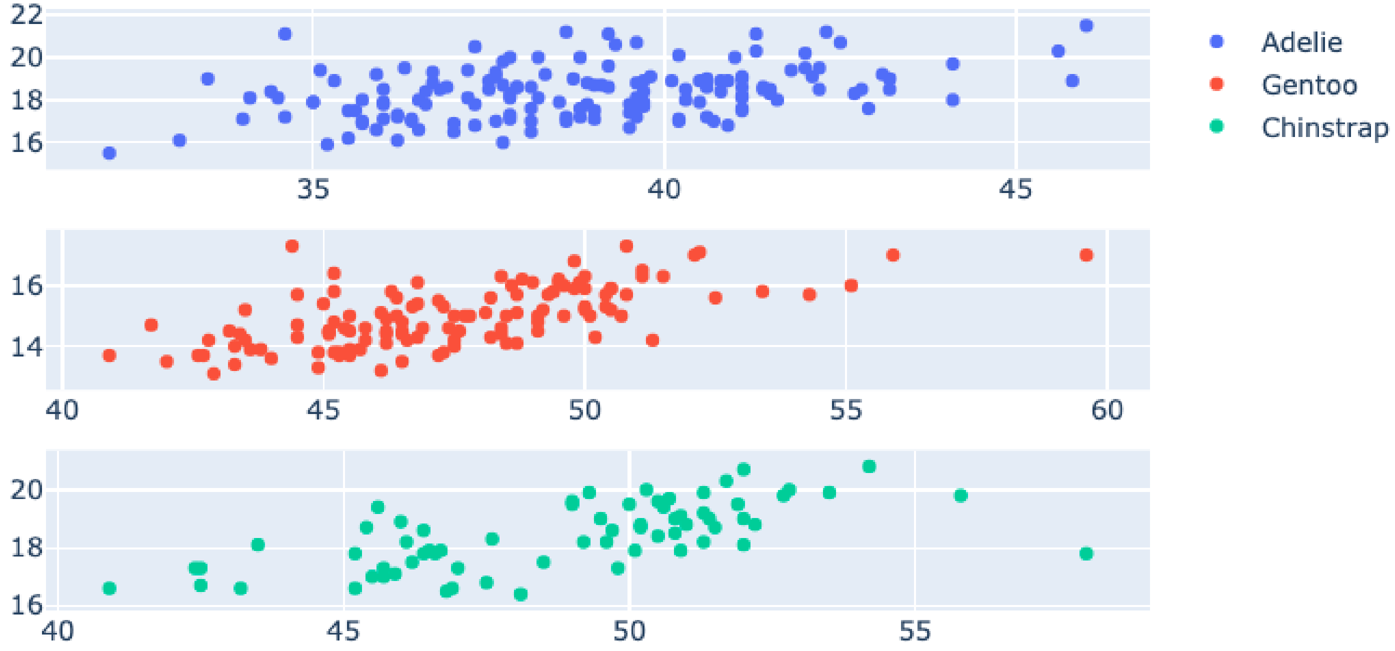


Stacked subplots

Let's redo our penguins scatterplot with subplots, splitting out the species:

```
fig = make_subplots(rows=3, cols=1)
row_num = 1
for species in ['Adelie', 'Gentoo', 'Chinstrap']:
  df = penguins[penguins['Species'] == species]
  fig.add_trace(
    go.Scatter(x=df['Culmen Length (mm)'],
               y=df['Culmen Depth (mm)'],
               name=species, mode='markers'),
    row=row_num, col=1)
  row_num +=1
fig.show()
```

Different x-axes?

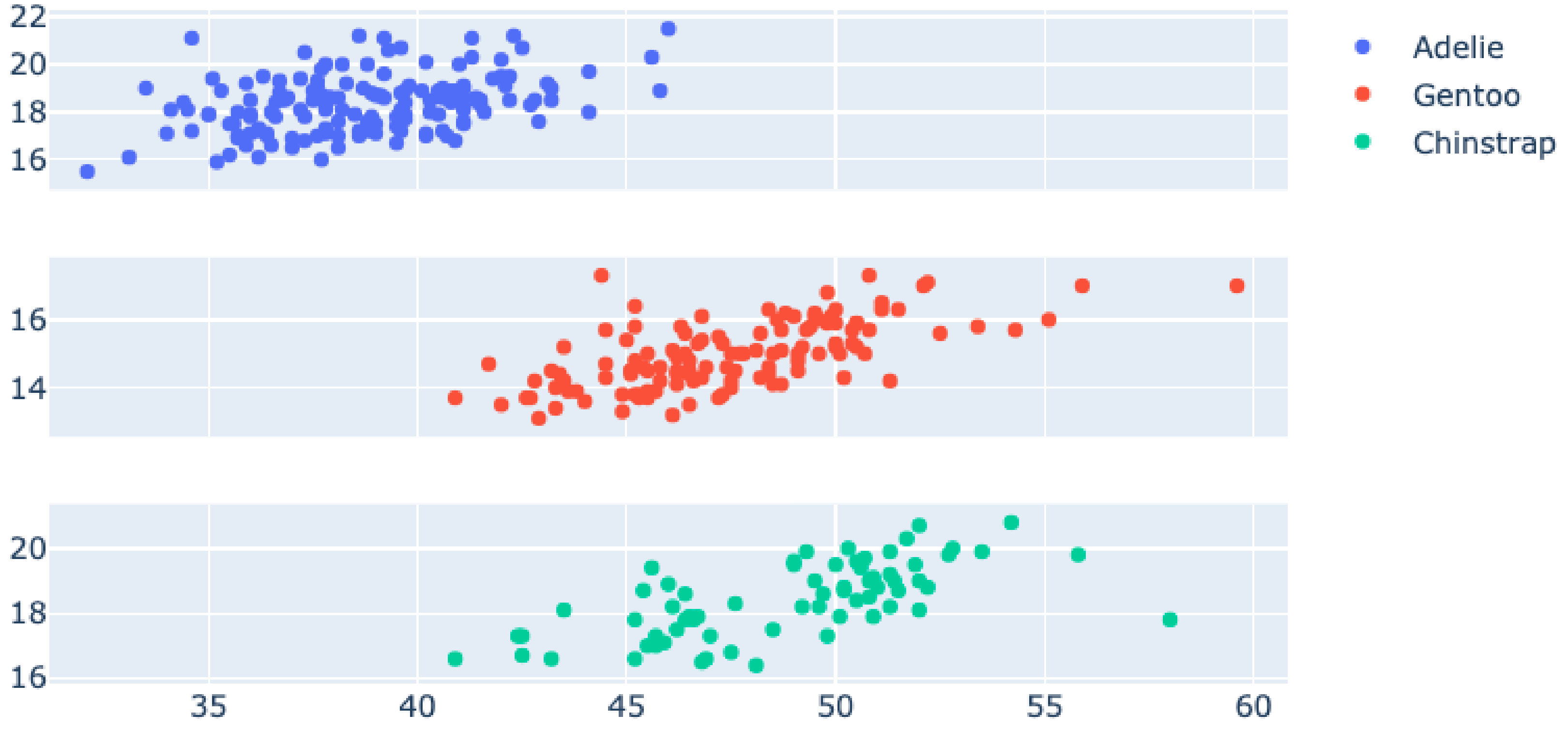


Subplots with shared axes

Let's fix this by making the x-axis 'shared':

```
fig = make_subplots(rows=3, cols=1, shared_xaxes=True)
```

That's better!



Let's practice!

INTRODUCTION TO DATA VISUALIZATION WITH PLOTLY IN PYTHON

Layering multiple plots

INTRODUCTION TO DATA VISUALIZATION WITH PLOTLY IN PYTHON

Alex Scriven
Data Scientist

What is plot layering?

Layering plots = multiple plots on top of each other

- No separate grid location (or separate plot)
- We use `add_trace()`
- Some 'shortcut' functions exist:
  - `add_bar()` , `add_area()` , `add_box()` , etc.
  - Search for 'add_' on the figure **documentation** for more

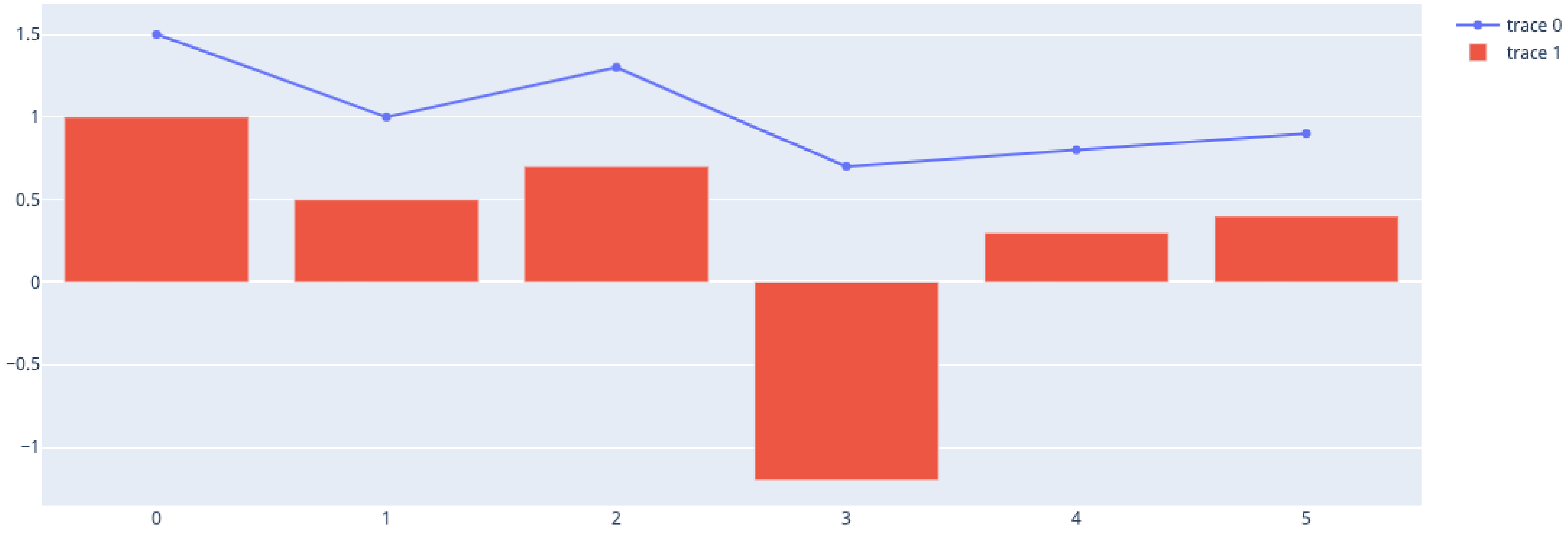
Why layer plots?

Layering plots is useful for:

- Accessing more customization (same type)
  - For example, layering multiple line charts
- Displaying complementary plot types
- Using different plot types to draw focus
- Keeping visualizations tight for close comparisons
  - Compared to split out subplots or separate plots

Bar + line layered plot

- A bar chart with a line-chart layered over the top is common
- Allows analyzing trends in multiple variables over time

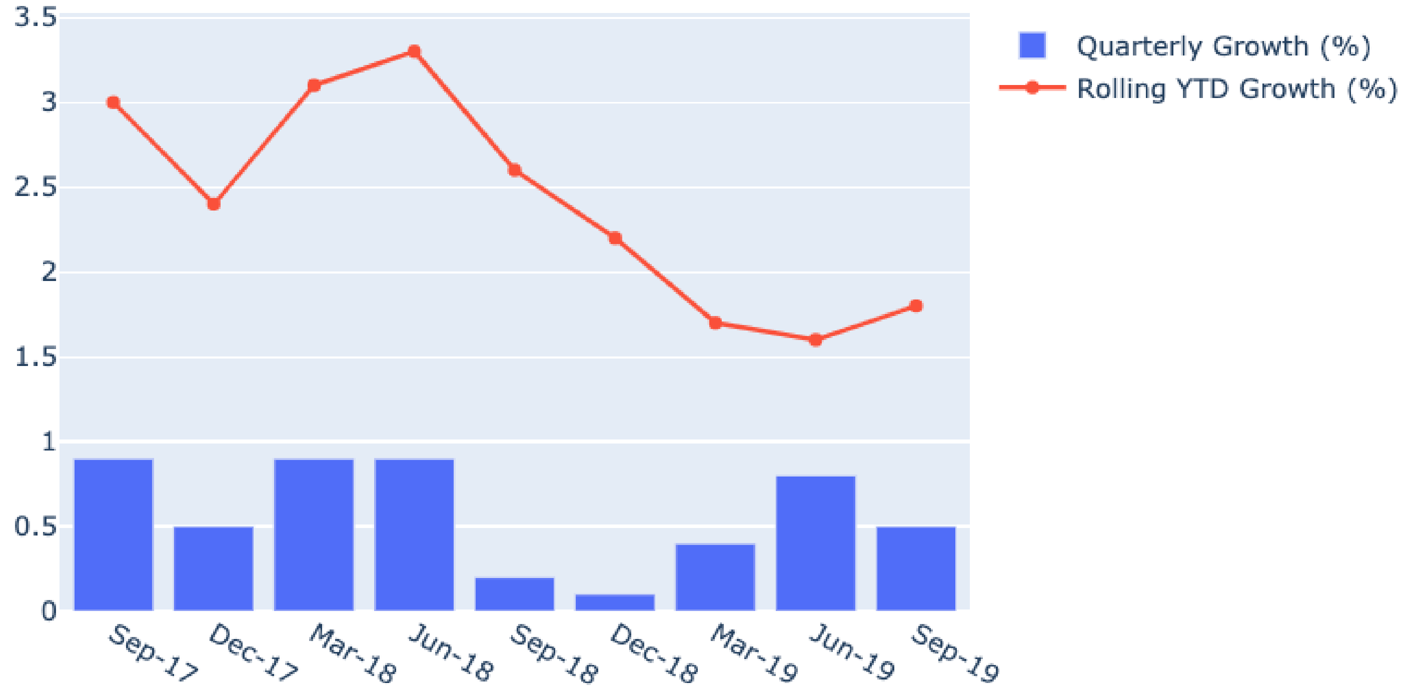


GDP growth layered plot

Consider the Australian GDP growth per quarter (and yearly rolling growth)

```
fig = go.Figure()
fig.add_trace(go.Bar(x=gdp['Date'],
    y=gdp['Quarterly growth (%)'],
    name='Quarterly Growth (%)'))
fig.add_trace(go.Scatter(x=gdp['Date'],
    y=gdp['Rolling YTD growth (%)'],
    name='Rolling YTD Growth (%)',
    mode='lines+markers'))
fig.show()
```

Here is our plot:



Nonsensical combinations

Layering many types of traces is possible, but stick to those that make sense:

- Line + another plot to show trend, such as
    - Line + bar plots
    - Line + scatterplots
- The same type (line + line, bar + bar)
- Make sure the x and y axes have the same units!

Let's practice!

INTRODUCTION TO DATA VISUALIZATION WITH PLOTLY IN PYTHON

Time buttons

INTRODUCTION TO DATA VISUALIZATION WITH PLOTLY IN PYTHON

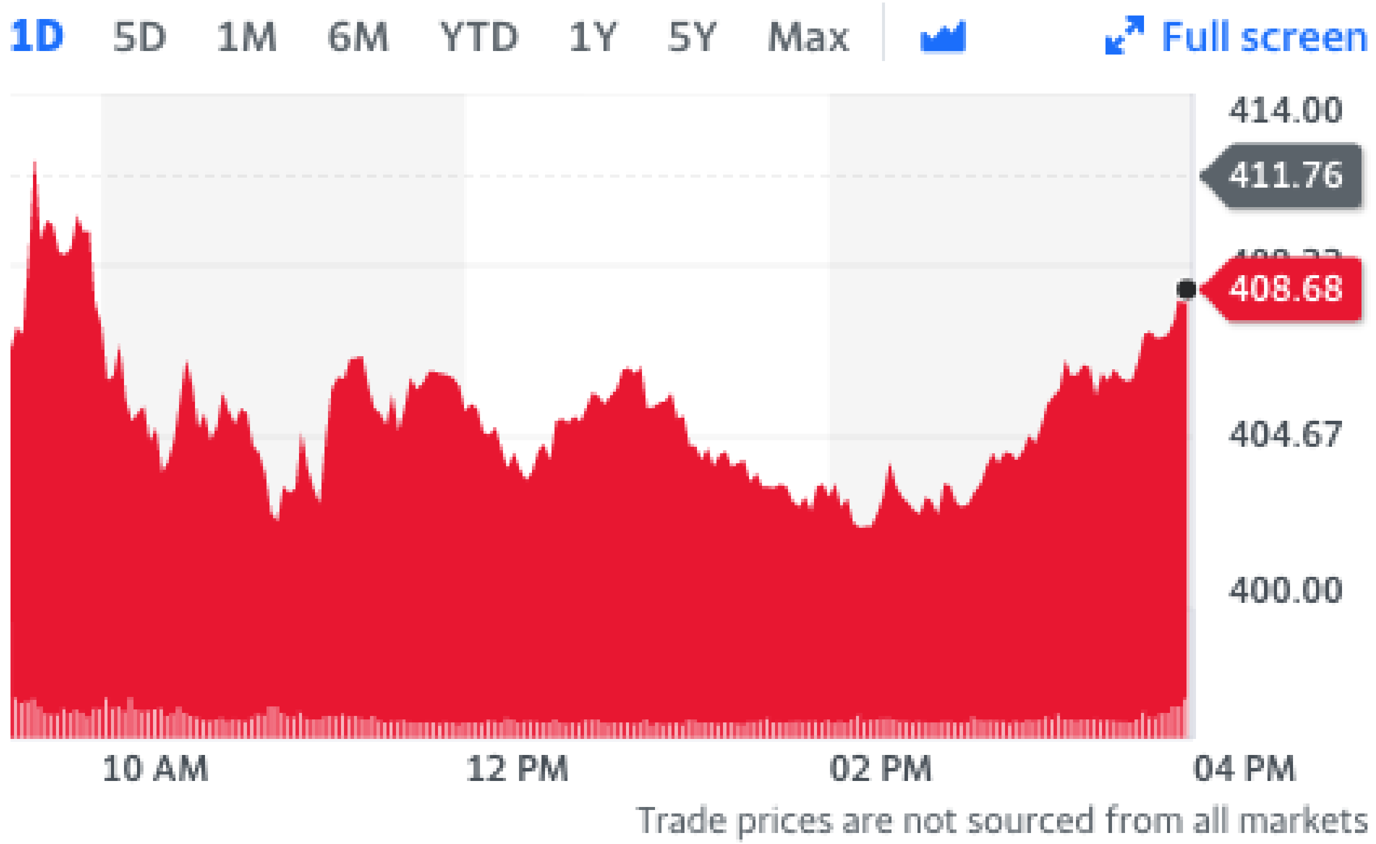
Alex Scriven
Data Scientist

What are time buttons?

Time buttons allow filter/zoom in line charts.

Often seen on most stock websites such as Yahoo Finance (TESLA stock);

- 1D = Show data for the last day, 1M = for the last month, 1Y = for the last year, etc.
- YTD = Show data for the 'year to date'



Time buttons in Plotly

Time buttons in Plotly are a dictionary specifying:

- `label` = Text to appear on the button
- `count` = How many `step` s to take when clicking the button
- `step` = What date period to move (`'month'` , `'year'` , `'day'` , etc.)
- `stepmode` = Either `'todate'` or `'backwards'`
  - `'todate'` = From the beginning of the nearest whole time period denoted in `step` (after going backwards by `count`)
  - `'backwards'` = Just go backwards by `count`

'todate' vs. 'backward'

To illustrate `todate` vs. `backward`, consider a dataset finishing on October 20th and a 6-month button (`count=6` , `step='month'`) with each option.

- `stepmode='backward'` would zoom the plot to start on **April 20th** (6 months backward)
- `stepmode='todate'` would zoom the plot to start on **May 1st** (start of the nearest month to April 20th)

Sydney rainfall example

Let's chart the rainfall from a weather station in Sydney in 2020.

Create the buttons

- Buttons are specified as a list of dictionaries

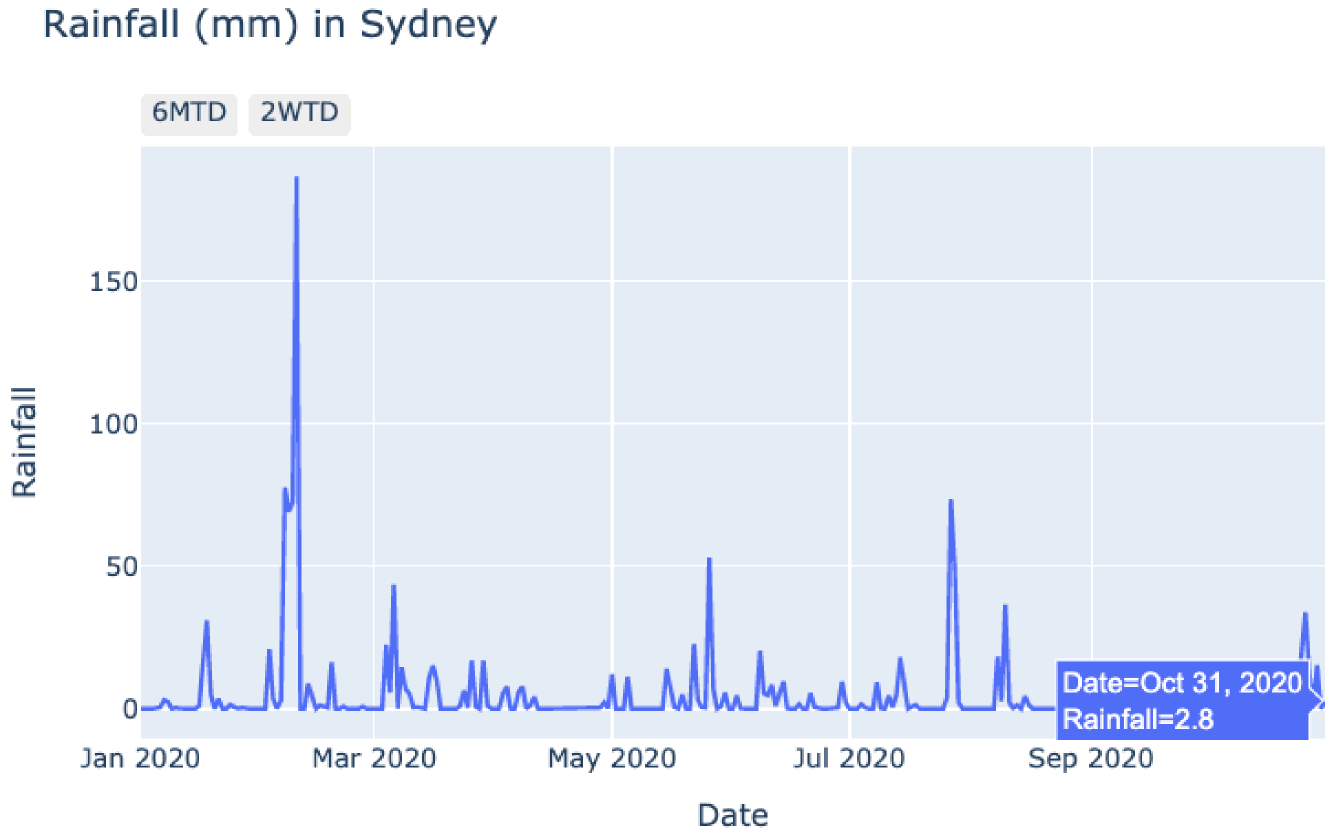
```
date_buttons = [
{'count': 6, 'step': "month", 'stepmode': "todate", 'label': "6MTD"},
{'count': 14, 'step': "day", 'stepmode': "todate", 'label': "2WTD"}
]
```

Adding the time buttons

Now let's create the chart and add them;

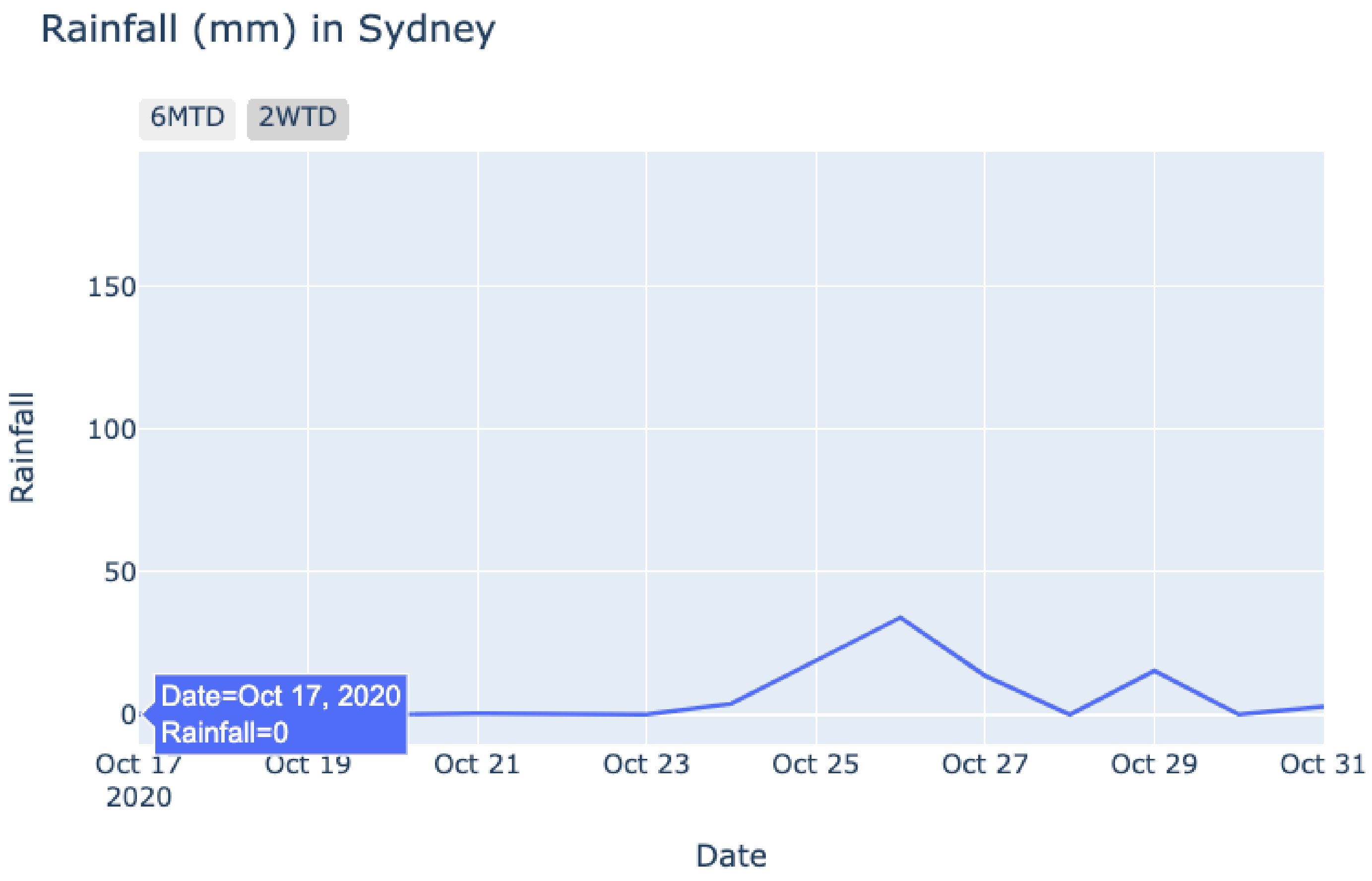
```
fig = px.line(data_frame=rain, x='Date',
        y='Rainfall',
        title="Rainfall (mm) in Sydney")
fig.update_layout(
    {'xaxis':
      {'rangeselector':
        {'buttons': date_buttons}
    }})
fig.show()
```

Our line chart has the buttons:

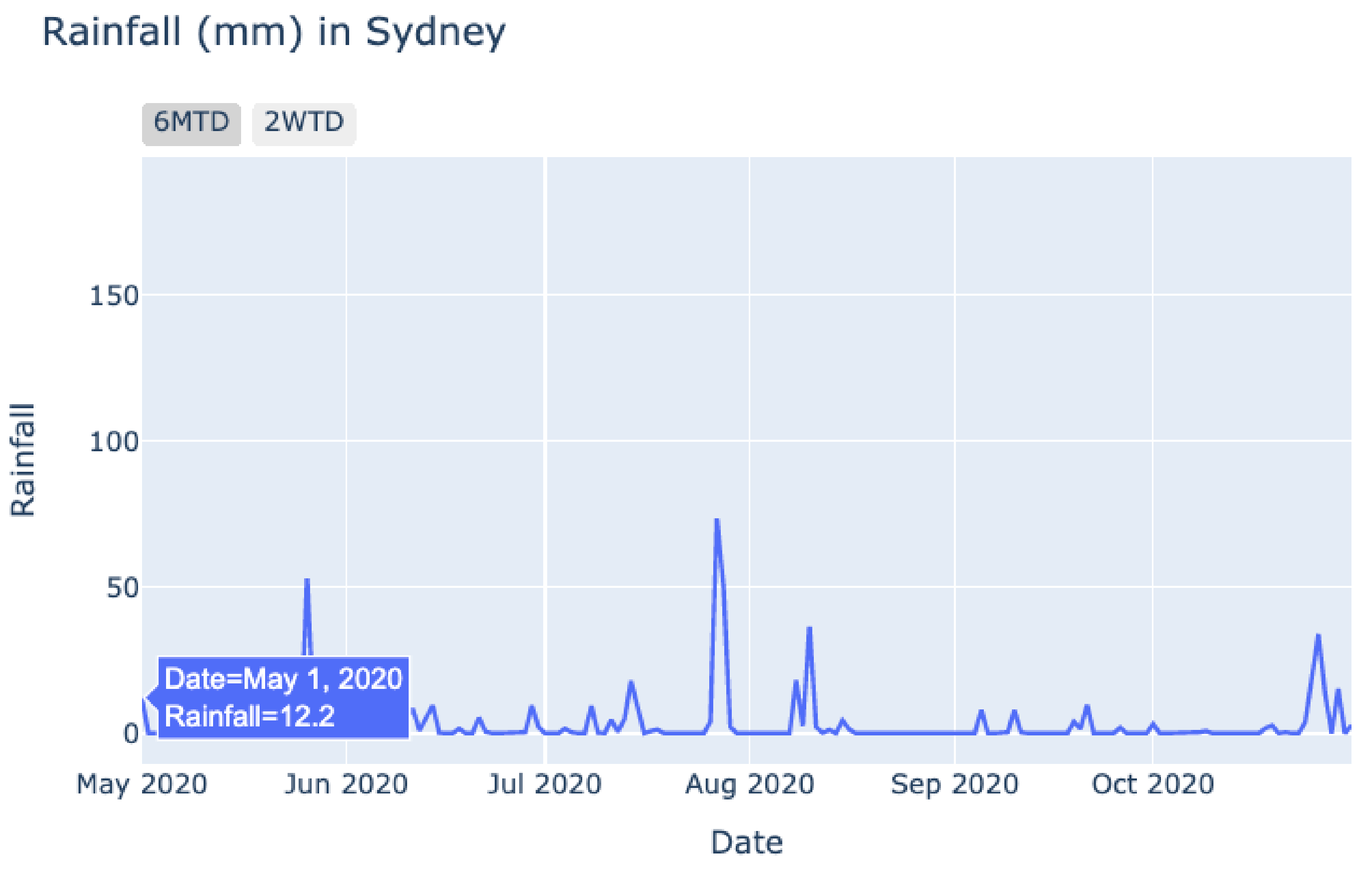


Clicking our time buttons

Clicking the **2WTD** button:



Clicking the **6MTD** button:



Let's practice!

INTRODUCTION TO DATA VISUALIZATION WITH PLOTLY IN PYTHON

Custom buttons

INTRODUCTION TO DATA VISUALIZATION WITH PLOTLY IN PYTHON

Alex Scriven
Data Scientist

What can custom buttons do?

Custom buttons can:

- Update the data or layout elements of a plot
    - All of our `update_layout()` customizations could be in a button!
- Assist with animations (beyond the scope of this course)

Custom buttons in Plotly

Buttons are added via an `updatemenus` argument (a list of dictionaries) with important arguments:

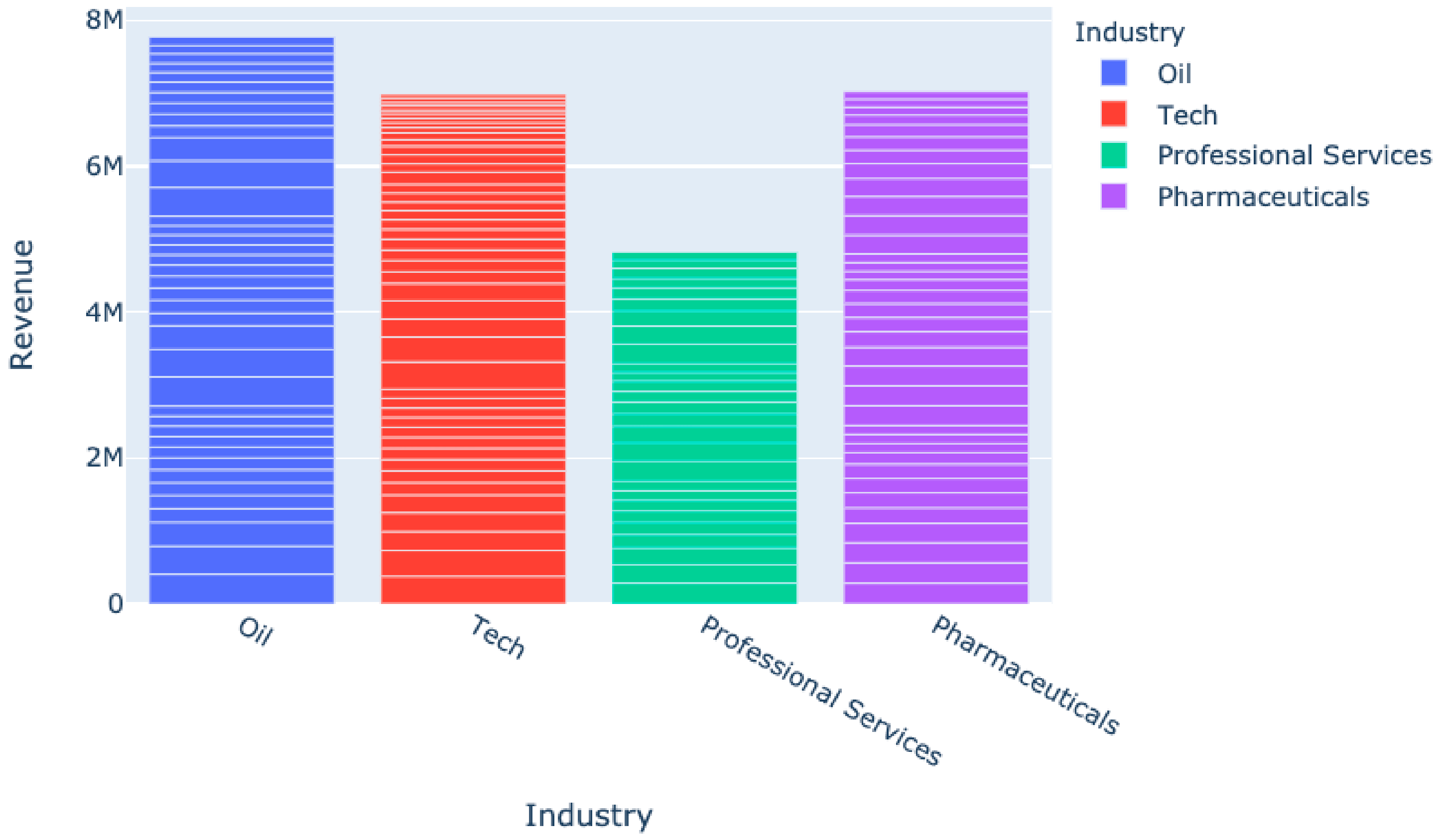
- **`type`** : `buttons` or `dropdown`
  - We will cover dropdowns later!
- **`direction`** : Button orientation
  - Buttons can be beside (`left`) or on top of (`down`) each other
- **`x`** / **`y`** : Floats to set the button positions as you have done before
- **`showactive`** : `True` / `False` to show the `active` (index of button) as pressed or not.
  - The active button is the currently selected one.
- **`buttons`** : A list of `button` objects

Plot type with buttons

Let's first set up a bar chart:

```
fig = px.bar(
      data_frame=revenues,
    x='Industry', y='Revenue',
    color='Industry')
fig.show()
```

Our simple bar chart:



Button set up

Create the buttons to switch plot type:

```
my_buttons = [
{'label': "Bar plot",
 'method': "update",
 'args': [{"type": "bar"}]},
{'label': "scatterplot",
 'method': "update",
 'args': [{"type": "scatter", 'mode': 'markers'}]},
]
```

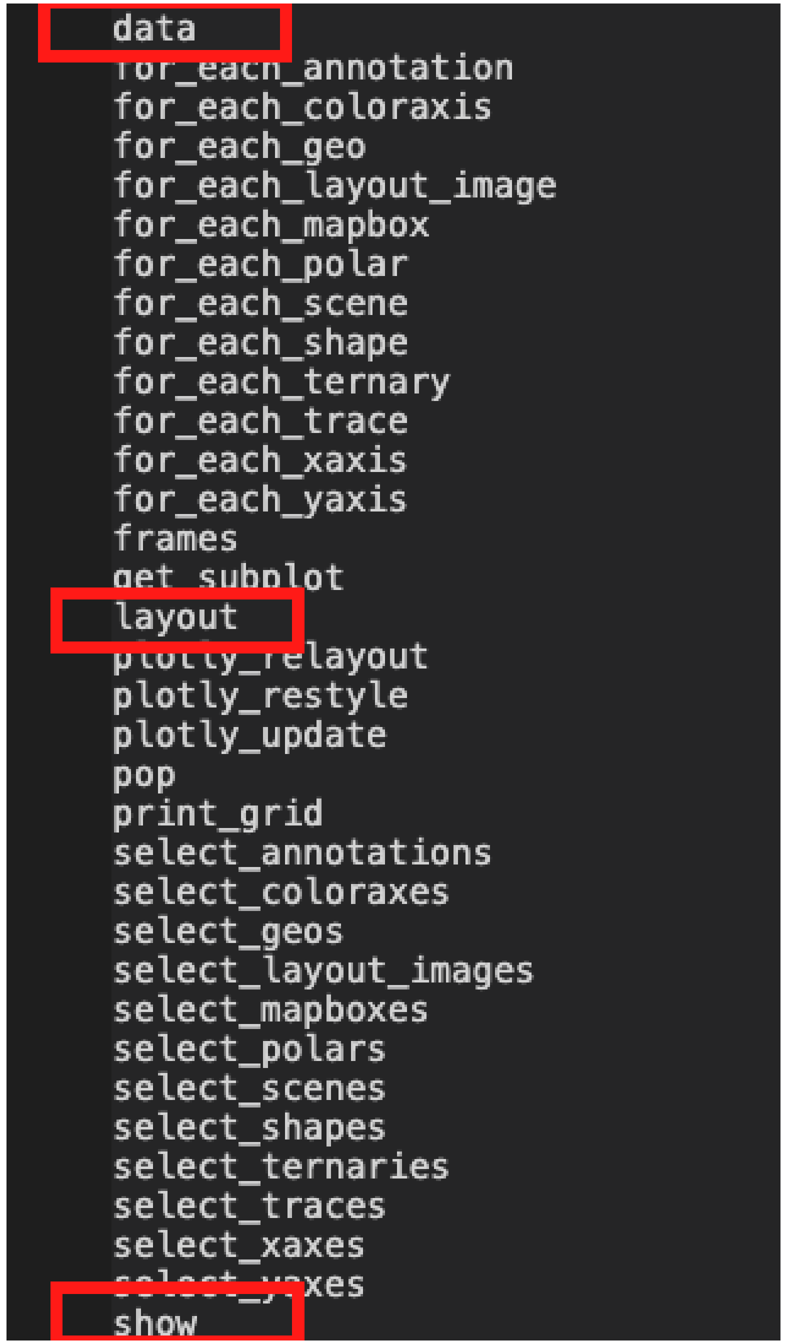
The args argument

One of the most confusing arguments in Plotly!

- Its structure is:

```
[{dictionary to send to data}, {dictionary to send to layout}]
```

- See what happens when we use Python's `dir` on our figure object to see the internal structure
  - There are some familiar faces! (much more is printed)

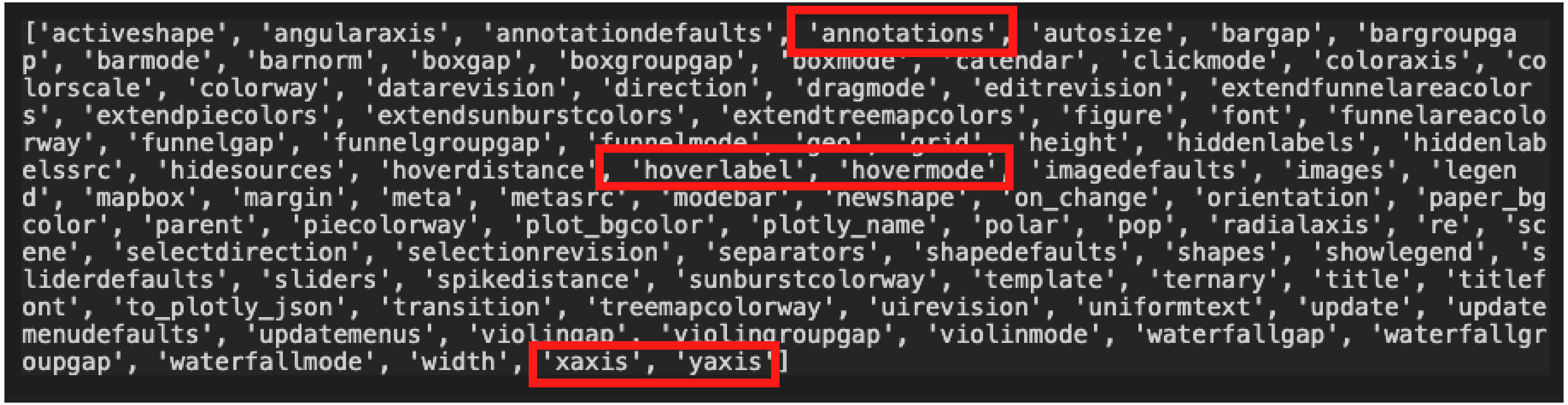


```
data
for_each_annotation
for_each_coloraxis
for_each_geo
for_each_layout_image
for_each_mapbox
for_each_polar
for_each_scene
for_each_shape
for_each_ternary
for_each_trace
for_each_xaxis
for_each_yaxis
frames
get_subplot
layout
plotly_relayout
plotly_restyle
plotly_update
pop
print_grid
select_annotations
select_coloraxes
select_geos
select_layout_images
select_mapboxes
select_polars
select_scenes
select_shapes
select_ternaries
select_traces
select_xaxes
select_yaxes
show
```

Using args for layout updates

Let's see what is inside the figure's `layout` element:

```
dir(fig.layout)
```



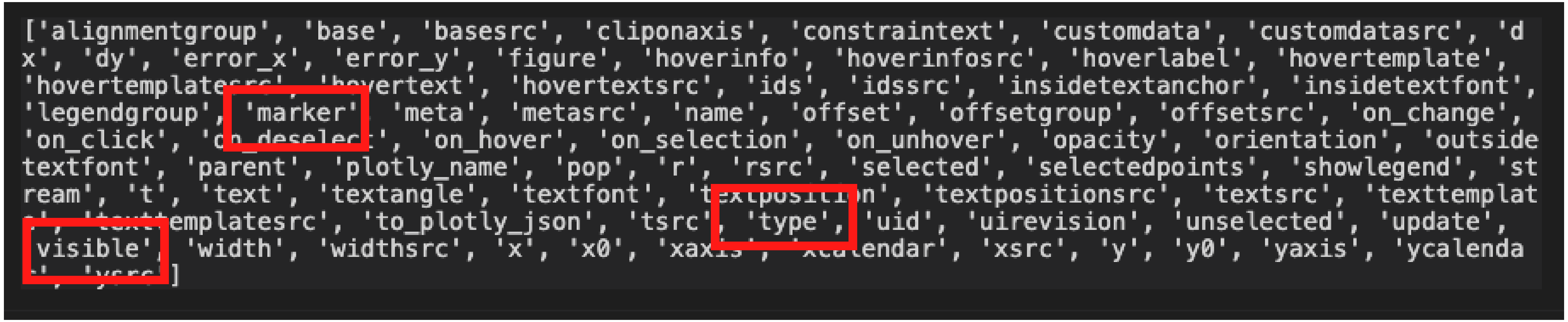
```
['activeshape', 'angularaxis', 'annotationdefaults', 'annotations', 'autosize', 'bargap', 'bargroupga
p', 'barmode', 'barnorm', 'boxgap', 'boxgroupgap', 'boxmode', 'calendar', 'clickmode', 'coloraxis', 'co
lorscale', 'colorway', 'datarevision', 'direction', 'dragmode', 'editrevision', 'extendfunnelareacolor
s', 'extendpiecolors', 'extendsunburstcolors', 'extendtreemapcolors', 'figure', 'font', 'funnelareacolo
rway', 'funnelgap', 'funnelgroupgap', 'funnelmode', 'geo', 'grid', 'height', 'hiddenlabels', 'hiddenlab
elssrc', 'hidesources', 'hoverdistance', 'hoverlabel', 'hovermode', 'imagedefaults', 'images', 'legen
d', 'mapbox', 'margin', 'meta', 'metasrc', 'modebar', 'newshape', 'on_change', 'orientation', 'paper_bg
color', 'parent', 'piecolorway', 'plot_bgcolor', 'plotly_name', 'polar', 'pop', 'radialaxis', 're', 'sc
ene', 'selectdirection', 'selectionrevision', 'separators', 'shapedefaults', 'shapes', 'showlegend', 's
liderdefaults', 'sliders', 'spikedistance', 'sunburstcolorway', 'template', 'ternary', 'title', 'titlef
ont', 'to_plotly_json', 'transition', 'treemapcolorway', 'uirevision', 'uniformtext', 'update', 'update
menudefaults', 'updatemenus', 'violingap', 'violingroupgap', 'violinmode', 'waterfallgap', 'waterfallgr
oupgap', 'waterfallmode', 'width', 'xaxis', 'yaxis']
```

Phew! There are many, but some should be familiar.

Using args for data updates

Let's also what is inside the figure's `data` element (of the first trace):

```python
dir(fig.data[0])
```



```
['alignmentgroup', 'base', 'basesrc', 'cliponaxis', 'constraintext', 'customdata', 'customdatasrc', 'd
x', 'dy', 'error_x', 'error_y', 'figure', 'hoverinfo', 'hoverinfosrc', 'hoverlabel', 'hovertemplate',
'hovertemplatesrc', 'hovertext', 'hovertextsrc', 'ids', 'idssrc', 'insidetextanchor', 'insidetextfont',
'legendgroup', 'marker', 'meta', 'metasrc', 'name', 'offset', 'offsetgroup', 'offsetsrc', 'on_change',
'on_click', 'on_deselect', 'on_hover', 'on_selection', 'on_unhover', 'opacity', 'orientation', 'outside
textfont', 'parent', 'plotly_name', 'pop', 'r', 'rsrc', 'selected', 'selectedpoints', 'showlegend', 'st
ream', 't', 'text', 'textangle', 'textfont', 'textposition', 'textpositionsrc', 'textsrc', 'texttemplat
e', 'texttemplatesrc', 'to_plotly_json', 'tsrc', 'type', 'uid', 'uirevision', 'unselected', 'update',
'visible', 'width', 'widthsrc', 'x', 'x0', 'xaxis', 'xcalendar', 'xsrc', 'y', 'y0', 'yaxis', 'ycalenda
r', 'ysrc']
```

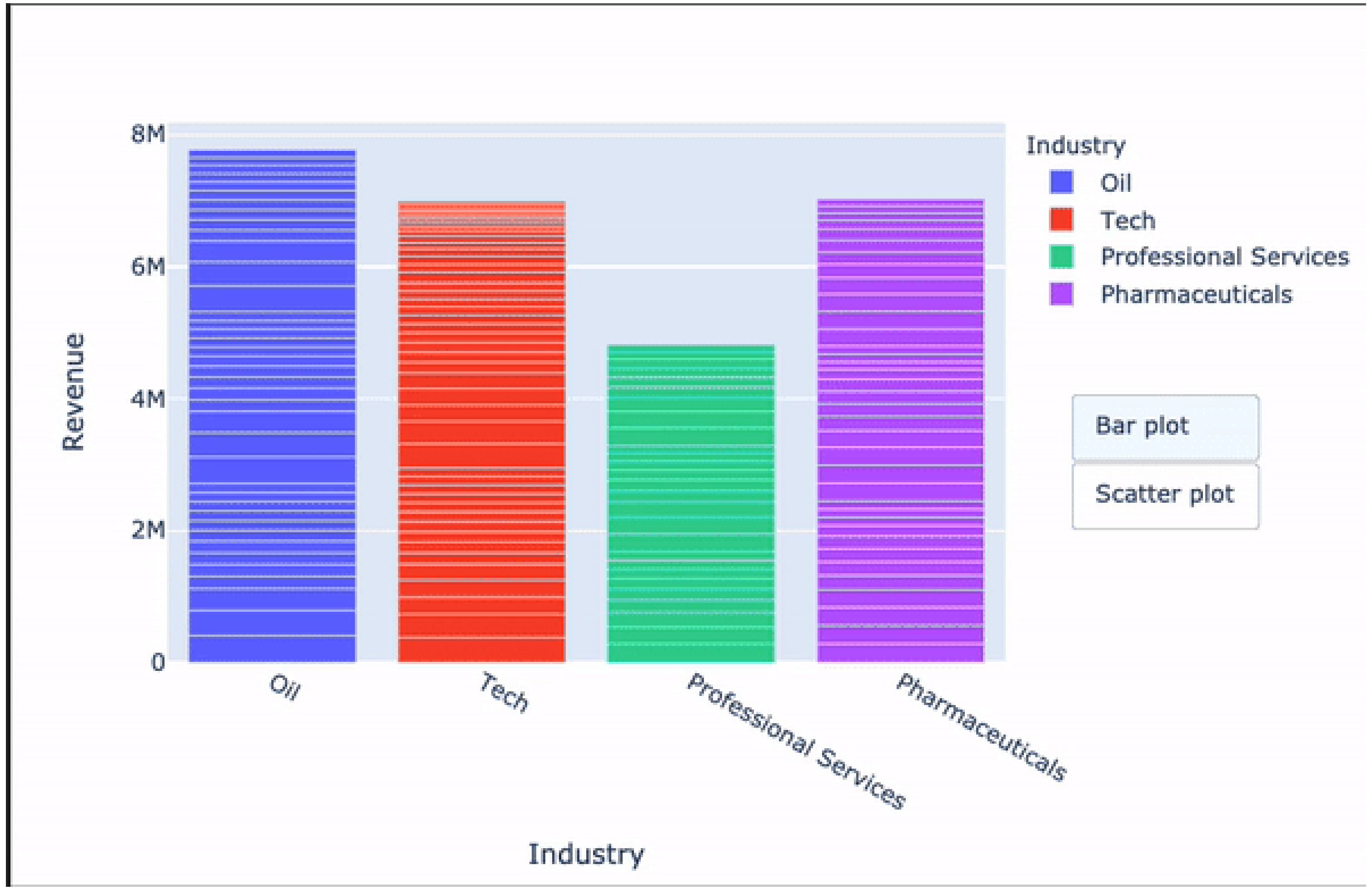
Some are familiar and some are worth noting for later!

Button interactivity

Set the button placement, stacking, and focus:

```
fig.update_layout({
  'updatemenus': [{'type': "buttons",
                   'direction': 'down',
                   'x': 1.3, 'y': 0.5,
                   'showactive': True,
                   'active': 0,
                   'buttons': my_buttons}]
  })
fig.show()
```

Our buttons at work!



Let's practice!

INTRODUCTION TO DATA VISUALIZATION WITH PLOTLY IN PYTHON

Dropdowns

INTRODUCTION TO DATA VISUALIZATION WITH PLOTLY IN PYTHON

Alex Scriven
Data Scientist

What is a dropdown?

- Allows user to select from a set of options
- These options will alter the plot in various ways

Dropdowns in Plotly

Dropdowns are created very similarly to buttons.

Create a figure and loop through DataFrames to add traces:

```
fig = go.Figure()
for suburb in ['Ashfield', 'Lidcombe', 'Bondi Junction']:
    df = syd_houses[syd_houses.Suburb == suburb]
    fig.add_trace(go.Bar(x=df['Year'], y=df['Median House Price'], name=suburb))
```

Why so many traces? Our dropdown is going to show/hide different ones!

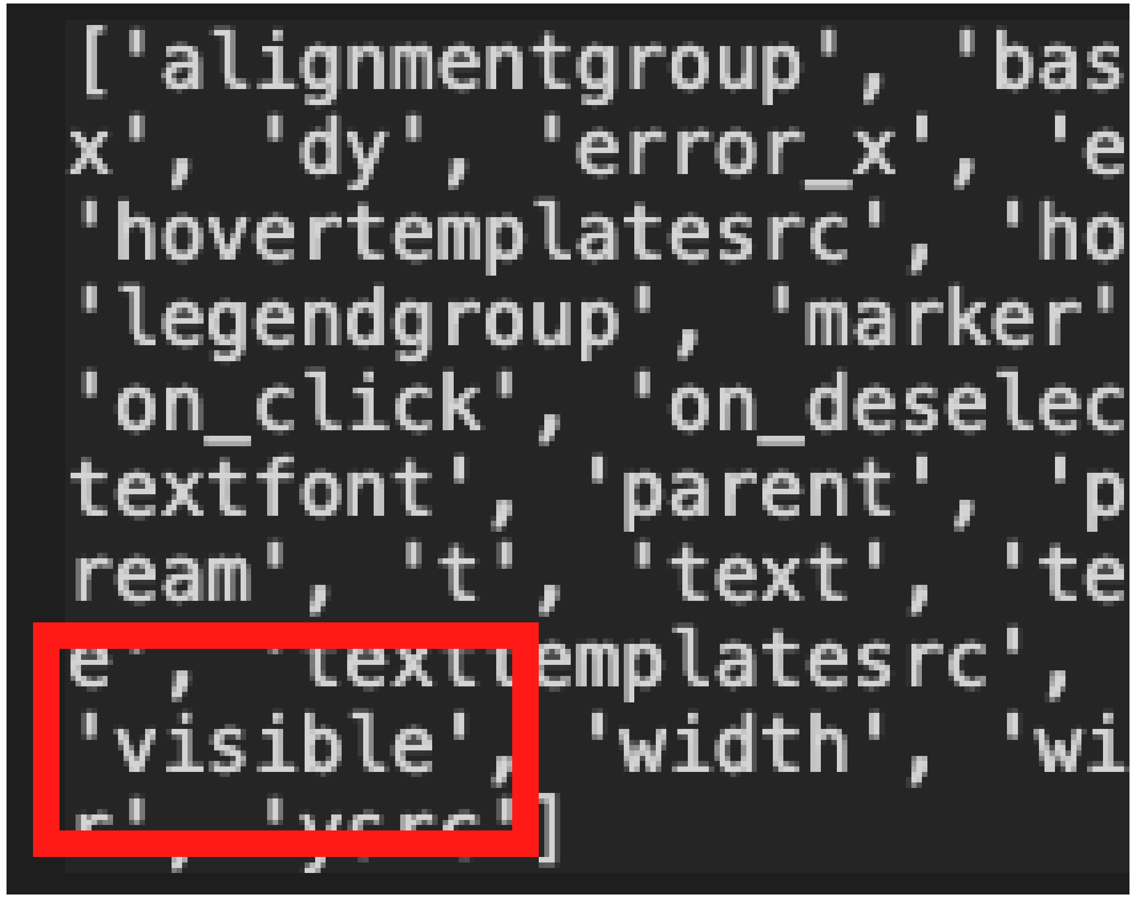
Hiding a trace

Recall what we can update in a figure's `data` element?

- The `visible` argument determines whether traces are visible (`True`) or not (`False`)
- We could use `args` to update the `visible` argument of different traces

```
args:[{'visible': [True, False, False]}]
```

- We can use a list for the `args` value to update all three traces



```
['alignmentgroup', 'bas
x', 'dy', 'error_x', 'e
'hovertemplatesrc', 'ho
'legendgroup', 'marker'
'on_click', 'on_deselec
textfont', 'parent', 'p
ream', 't', 'text', 'te
e', 'texttemplatesrc',
'visible', 'width', 'wi
r', 'ysrc']
```

The dropdown object

The dropdown object, like the button object, is also a list with the same arguments.

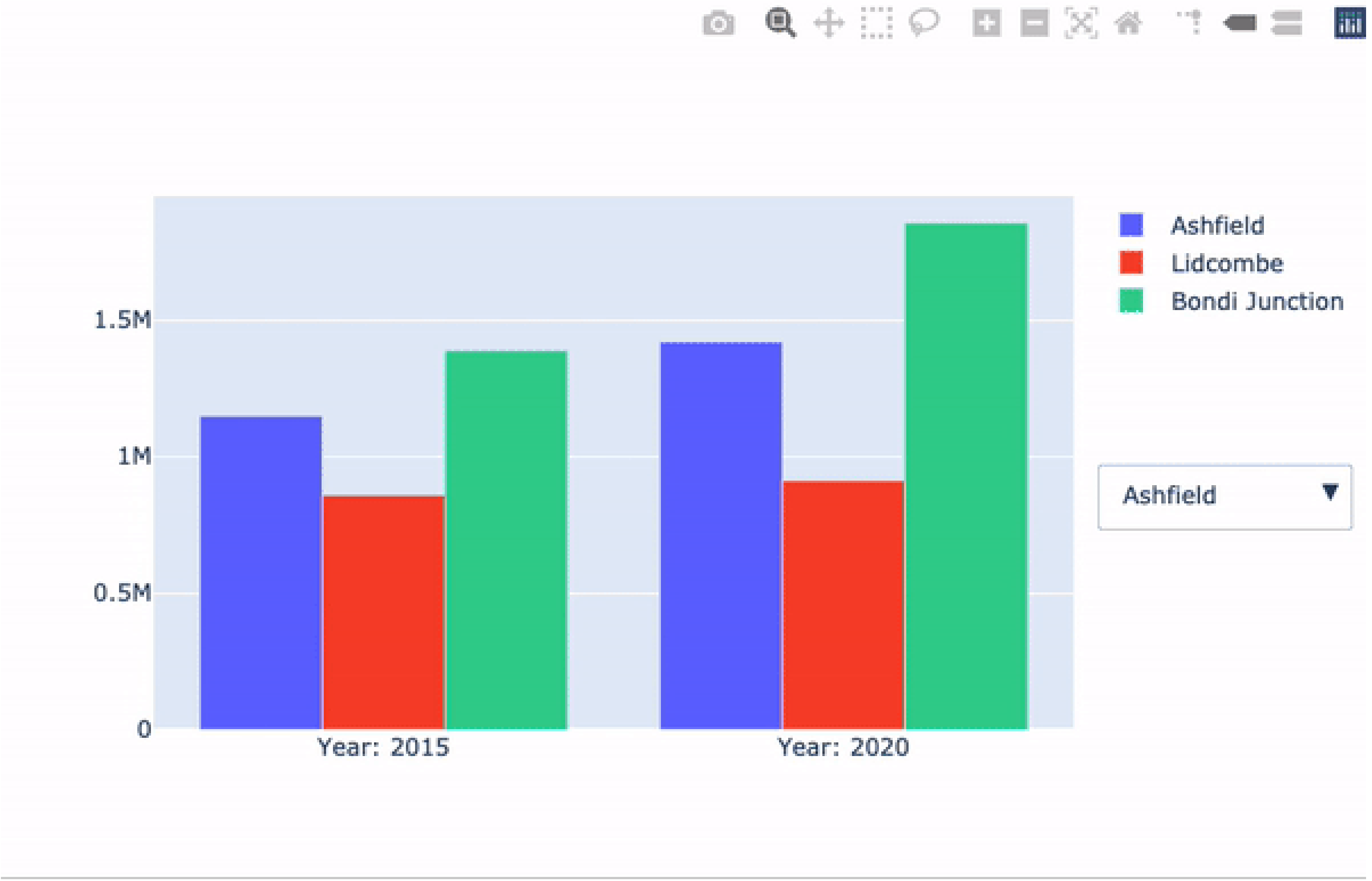
```
# Create the dropdown
dropdown_buttons = [
  {'label': 'Ashfield', 'method': 'update',
   'args': [{'visible': [True, False, False]},
           {'title': 'Ashfield'}]},
  {'label': 'Lidcombe', 'method': 'update',
  'args': [{'visible': [False, True, False]},
          {'title': 'Lidcombe'}]},
  {'label': "Bondi Junction", 'method': "update",
  'args': [{"visible": [False, False, True]},
          {'title': 'Bondi Junction'}]}
]
```

Adding the dropdown

Adding the dropdown is also very similar:

```
fig.update_layout({
  'updatemenus':[{
    'type': "dropdown",
    'x': 1.3,
    'y': 0.5,
    'showactive': True,
    'active': 0,
    'buttons': dropdown_buttons}]
})
fig.show()
```

Our dropdown:



Let's practice!

INTRODUCTION TO DATA VISUALIZATION WITH PLOTLY IN PYTHON

Sliders

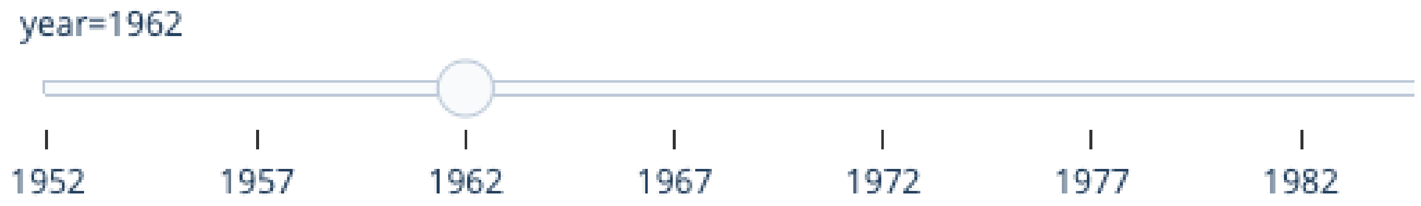
INTRODUCTION TO DATA VISUALIZATION WITH PLOTLY IN PYTHON

Alex Scriven
Data Scientist

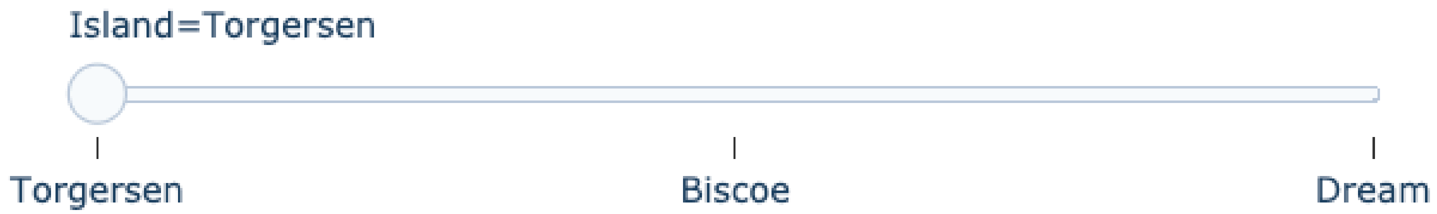
What are sliders?

- An interactive element to toggle between values and update your plot
- Often used for viewing data over time, such as data from different years
- Can be used for any group, such as penguin islands
- Ensure it makes sense in your plot

A year slider:



A penguin island slider:



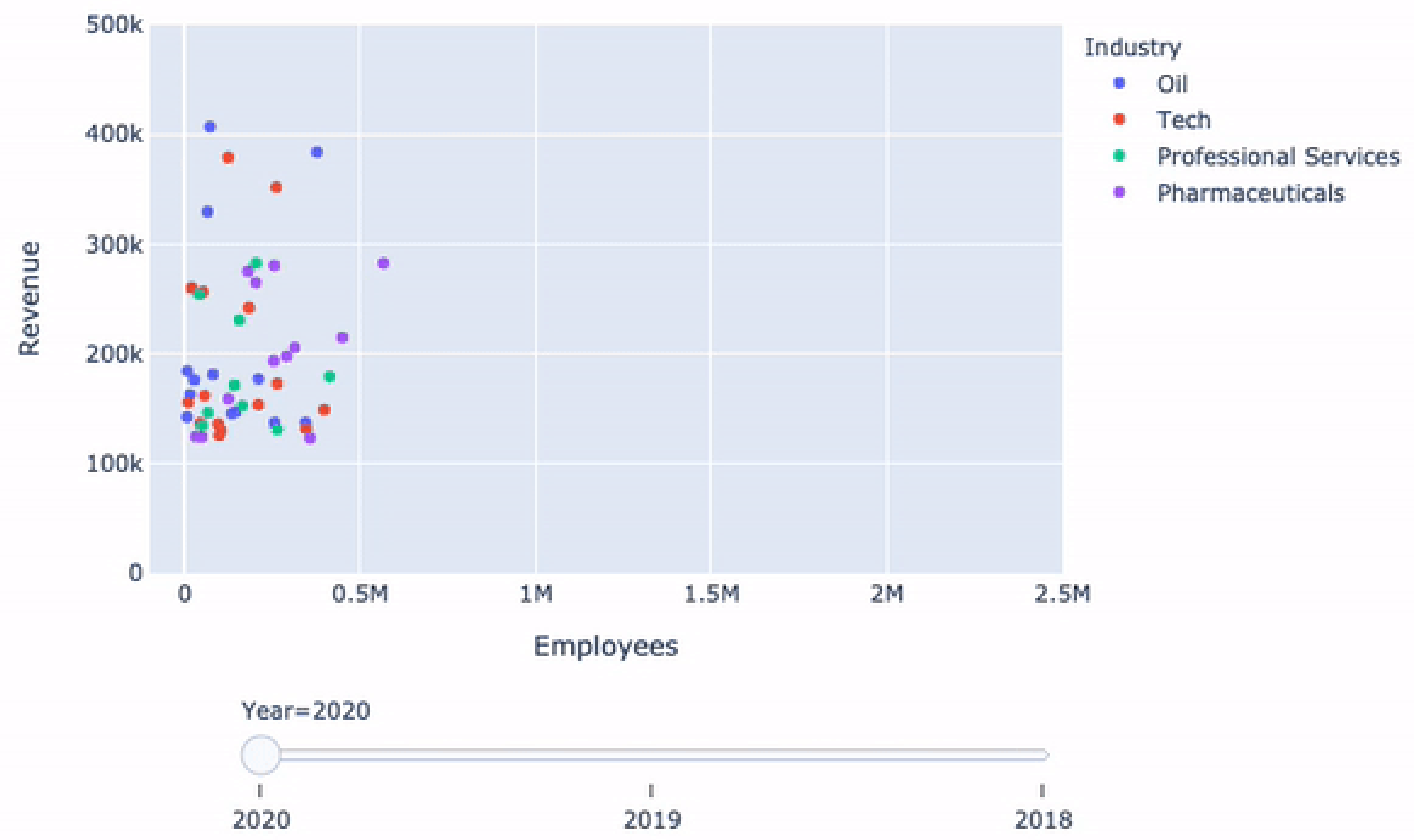
Sliders in plotly.express

`plotly.express` allows sliders via the `animation_frame` and `animation_group` arguments

- `animation_frame` : What will be on the slider (`Year` or `Island` on previous slide)
- `animation_group` : How to tell Plotly it is the same object over time

Revenue vs. Employees with slider

```
fig = px.scatter(
  data_frame=revenues,
  y='Revenue',
  x='Employees',
  color='Industry',
  animation_frame='Year',
  animation_group='Company'))

fig.update_layout({
  'yaxis': {'range': [0, 500000]},
  'xaxis': {'range': [-100000, 2500000]}
})

fig['layout'].pop('updatemenus')
fig.show()
```



plotly.express limitation: animate method

`plotly.express` sliders have a key limitation - the `animation` slider method

In the `Figure` object

```
fig['layout']['sliders'][0].steps[0]['method']
```

```
animate
```

- With `plotly.express` , you can't update data or layout — only animate the **same data point** over different 'frames'.
- To solve this, we need to use `graph_objects` to create the slider

Sliders with graph_objects

To use `graph_objects` , we need to:

1. Create a figure object with necessary traces
2. Create a sliders object to show/hide traces
3. Update the layout to add the slider to the figure

Creating the figure

Let's create the figure and add traces

```
fig = go.Figure()
for island in ['Torgersen', 'Biscoe', 'Dream']:
    df = penguins[penguins.Island == island]
    fig.add_trace(go.Scatter(
            x=df["Culmen Length (mm)"],
            y=df["Culmen Depth (mm)"], mode='markers', name=island))
```

Creating the slider

Let's create the slider object:

```
sliders = [
  {'steps':[
    {'method': 'update', 'label': 'Torgersen',
    'args': [{'visible': [True, False, False]}]},
    {'method': 'update', 'label': 'Bisco',
    'args': [{'visible': [False, True , False]}]},
    {'method': 'update', 'label': 'Dream',
    'args': [{'visible': [False, False, True]}]}
  ]}
]
```

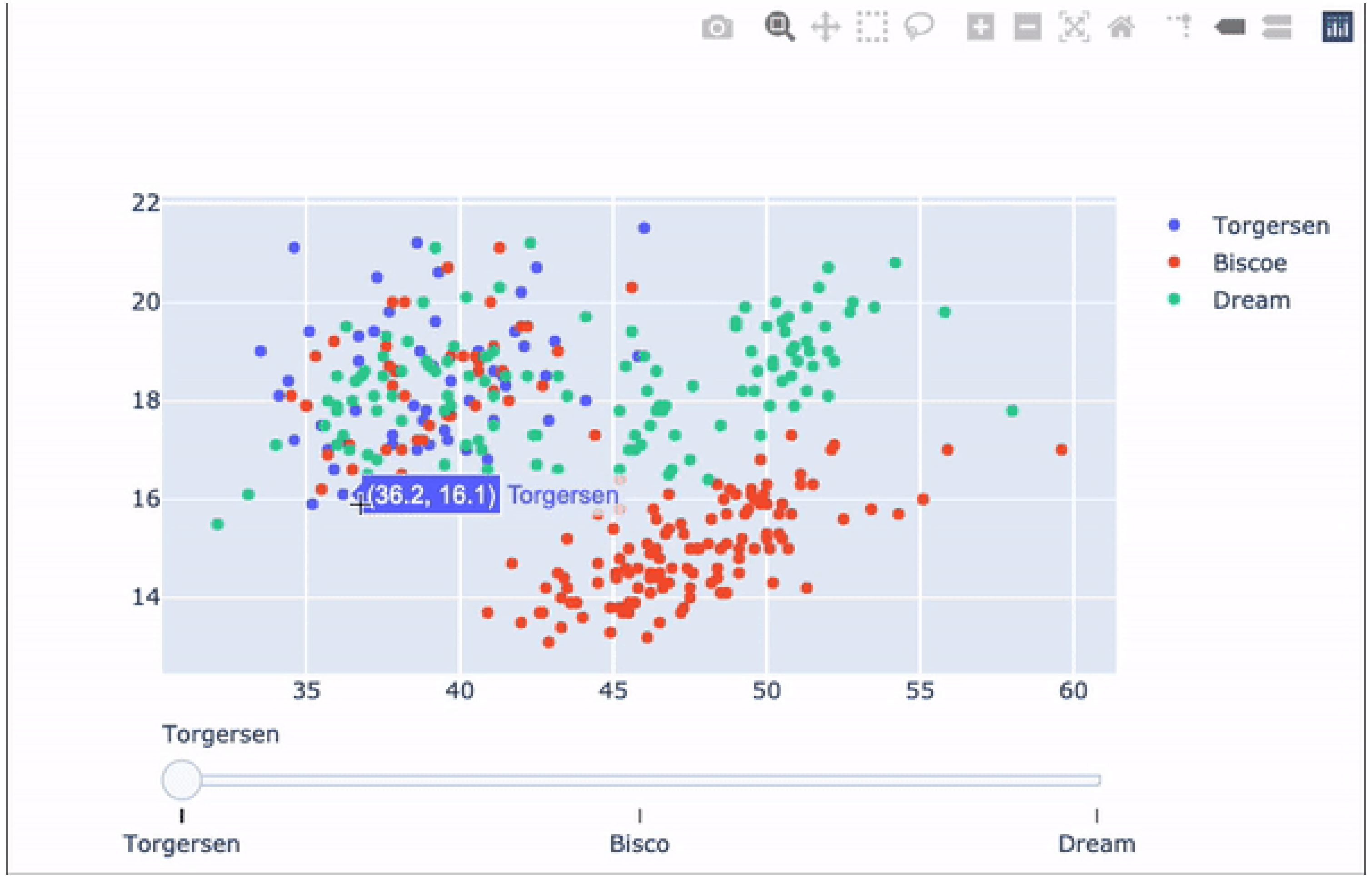
More formatting options available in the **docs**!

Adding the slider

Now we can add the slider to our figure:

```
fig.update_layout({'sliders': sliders})
fig.show()
```

The first screen was a bit funny huh? Let's fix that!

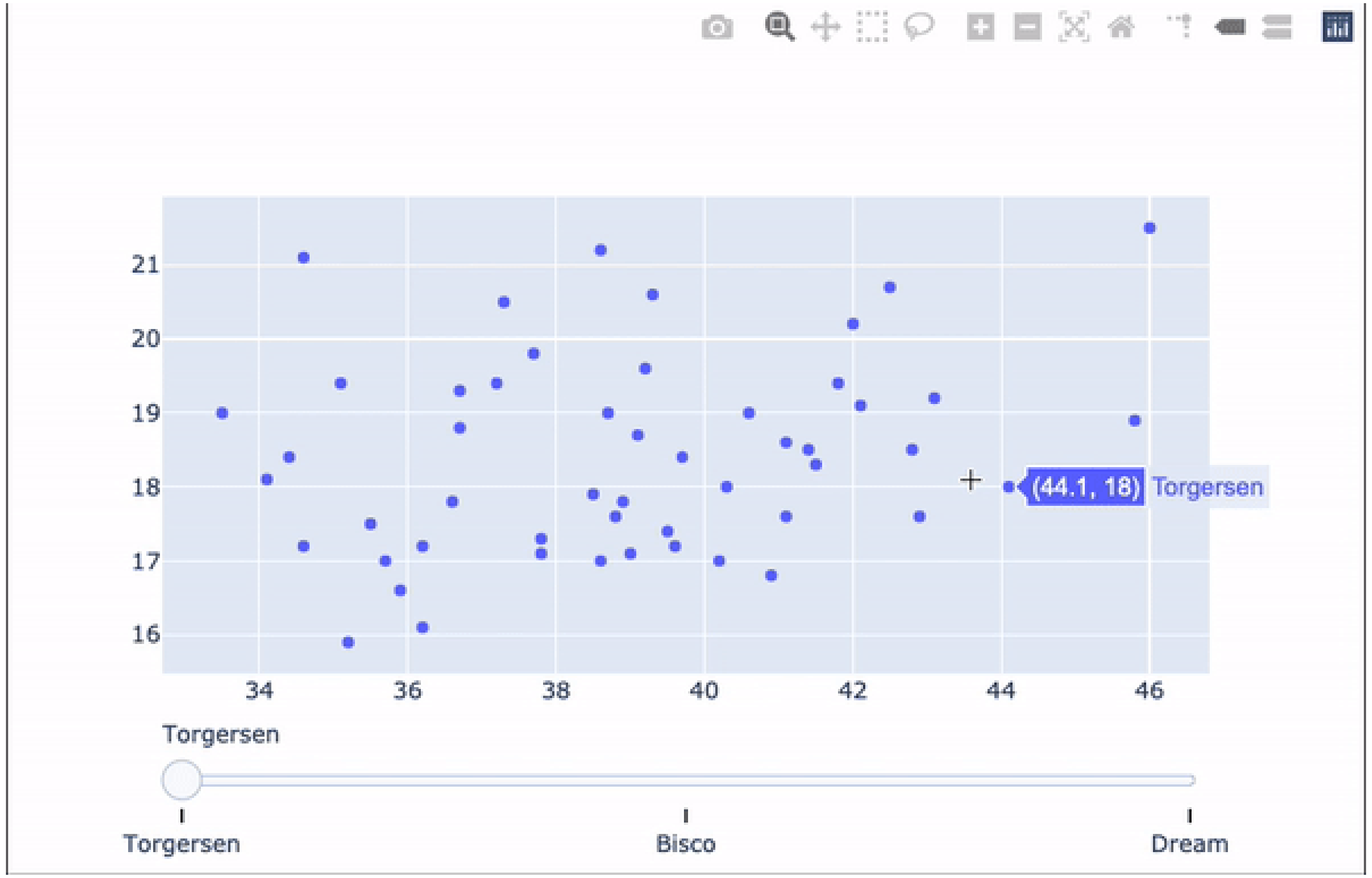


Fixing the initial display

We can fix the initial display by setting only the relevant traces to show.

```
# Make traces invisible
fig.data[1].visible=False
fig.data[2].visible=False

fig.update_layout({'sliders': sliders})
fig.show()
```

Much better!



Let's practice!

INTRODUCTION TO DATA VISUALIZATION WITH PLOTLY IN PYTHON

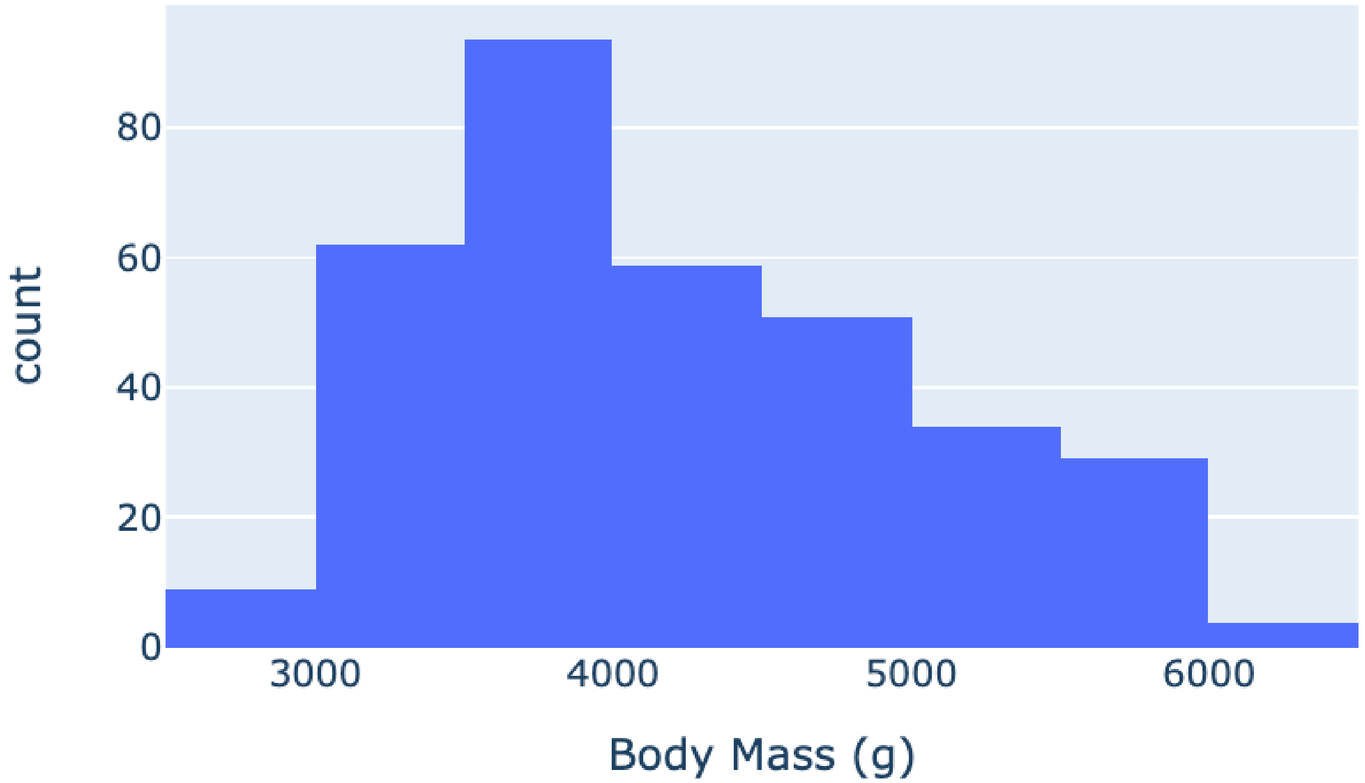
What you learned

INTRODUCTION TO DATA VISUALIZATION WITH PLOTLY IN PYTHON

Alex Scriven
Data Scientist

Chapter 1

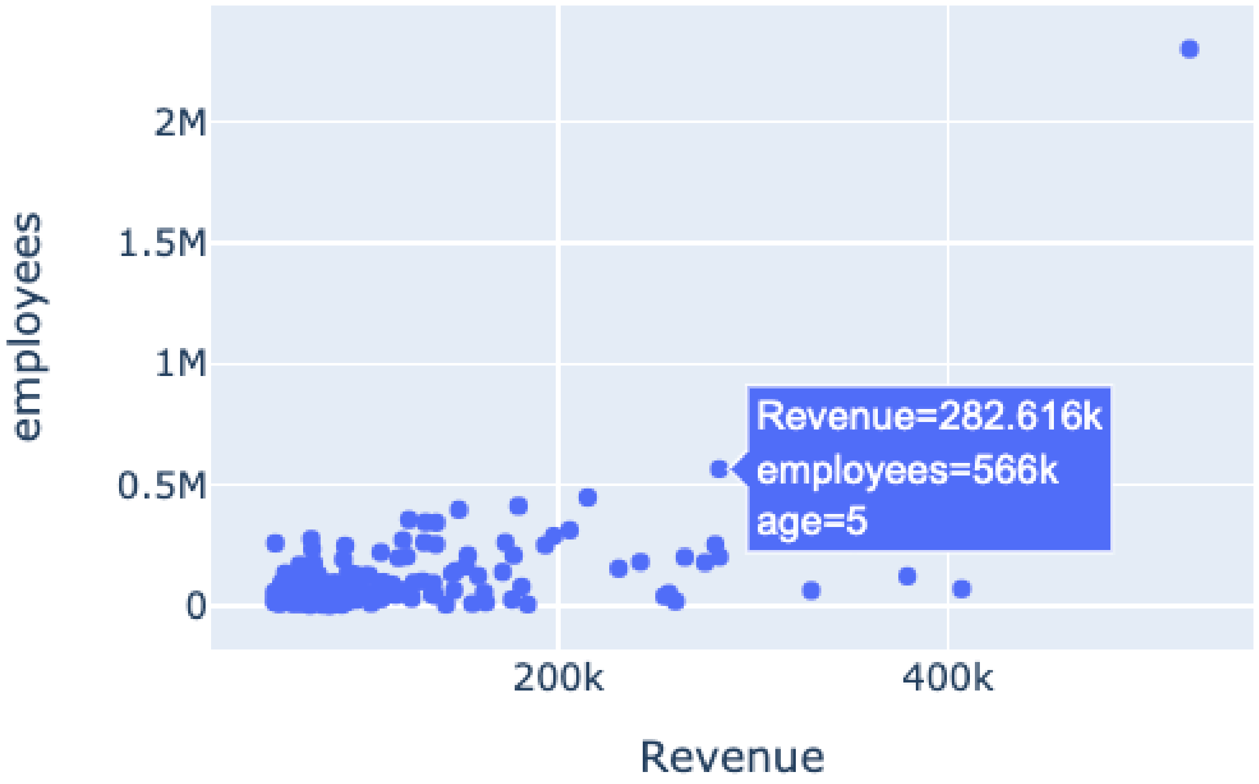
- The Plotly figure
- Univariate plots such as box plots and histograms
- Styled plots using color



Chapter 2

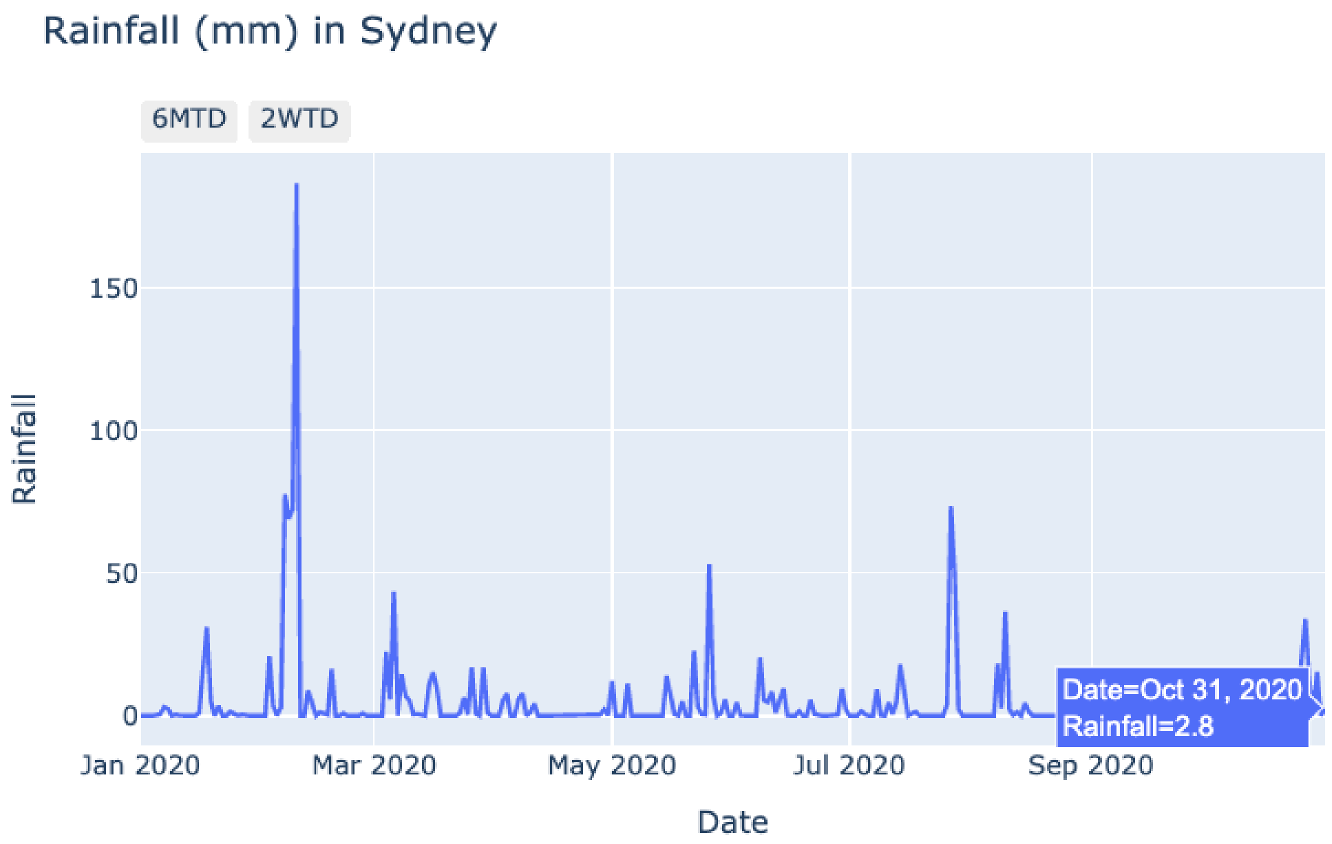
- Bivariate visualizations such as scatterplots and bar plots
- Customized your plots further with:
    - Hover information and legends
    - Annotations
    - Custom plot axes

Recall seeing company `age` (another variable) in the hover!



Chapter 3

- Advanced customization
  - Subplots of same or different types
  - Layering multiple plots on the same chart
  - An introduction to time buttons

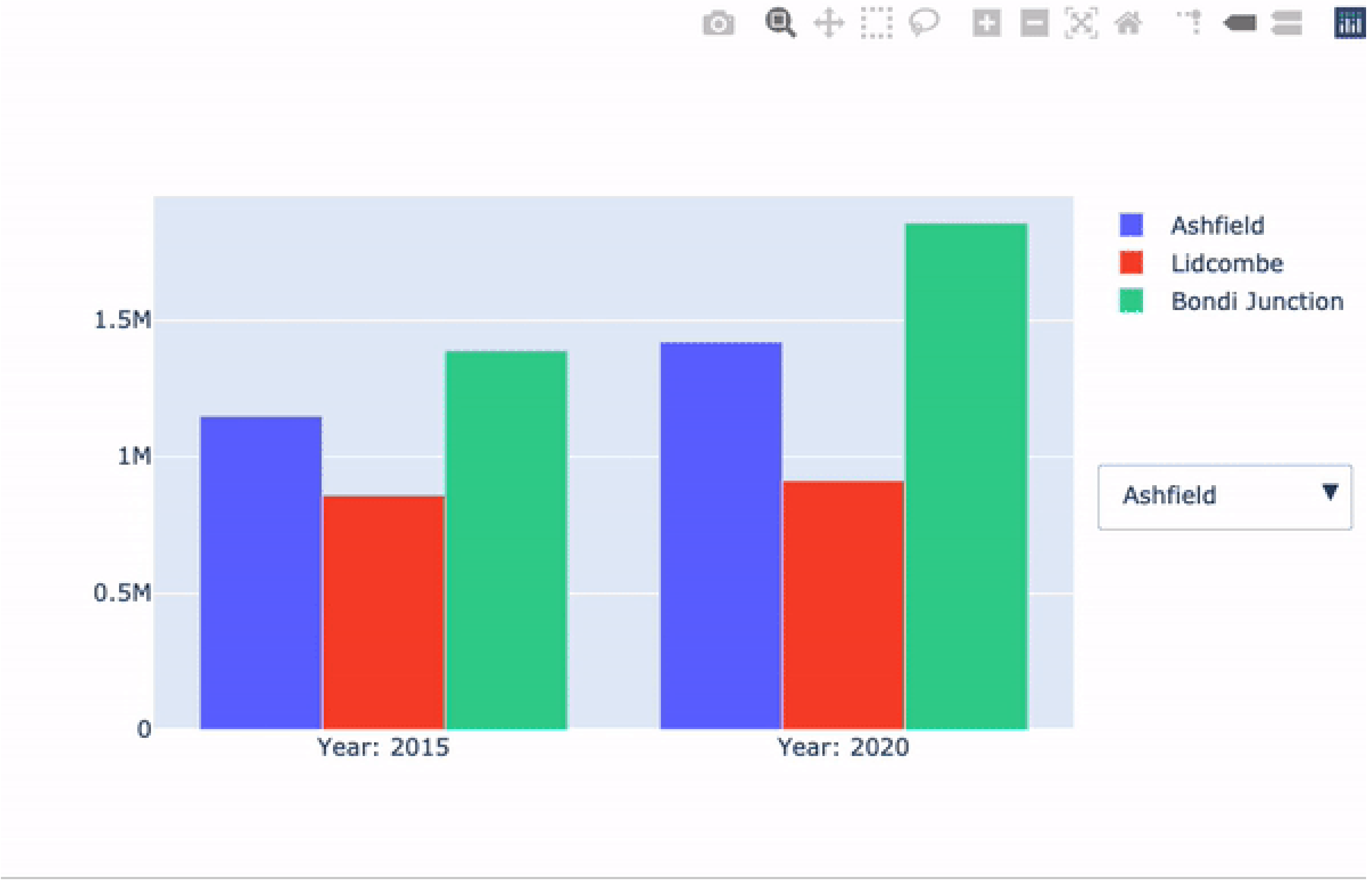


Chapter 4

Using interactive elements:

- Buttons
- Dropdowns
- Sliders

Your houses dropdown:



Thank you!

INTRODUCTION TO DATA VISUALIZATION WITH PLOTLY IN PYTHON