Bivariate visualizations

INTRODUCTION TO DATA VISUALIZATION WITH PLOTLY IN PYTHON

Alex Scriven
Data Scientist

datacamp

What are bivariate visualizations?

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

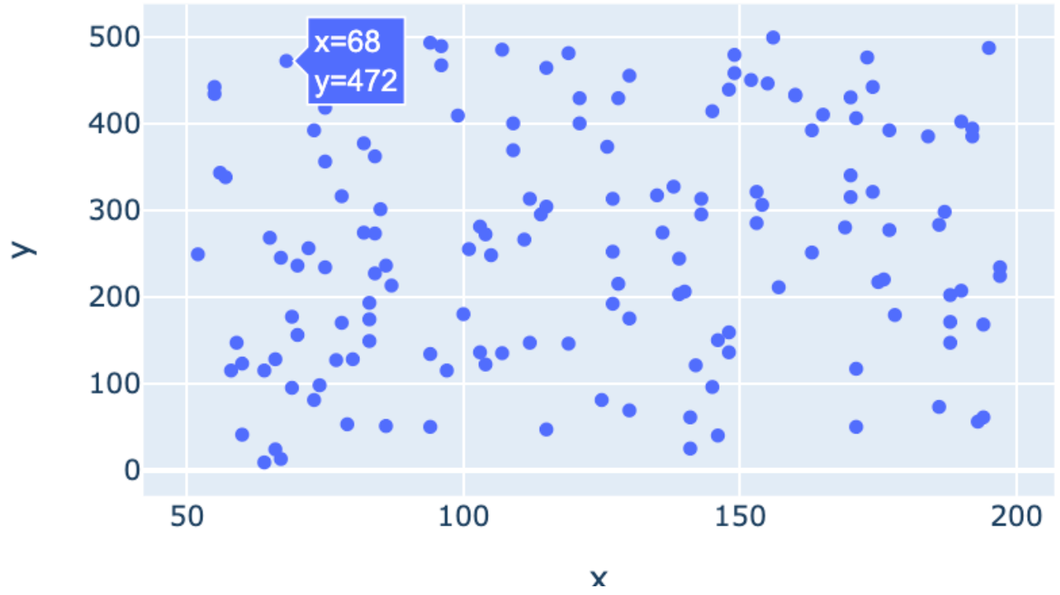
Common bivariate plots include:

- scatterplots
- Correlation plots
- Line charts

scatterplot

A scatterplot is a plot consisting of:

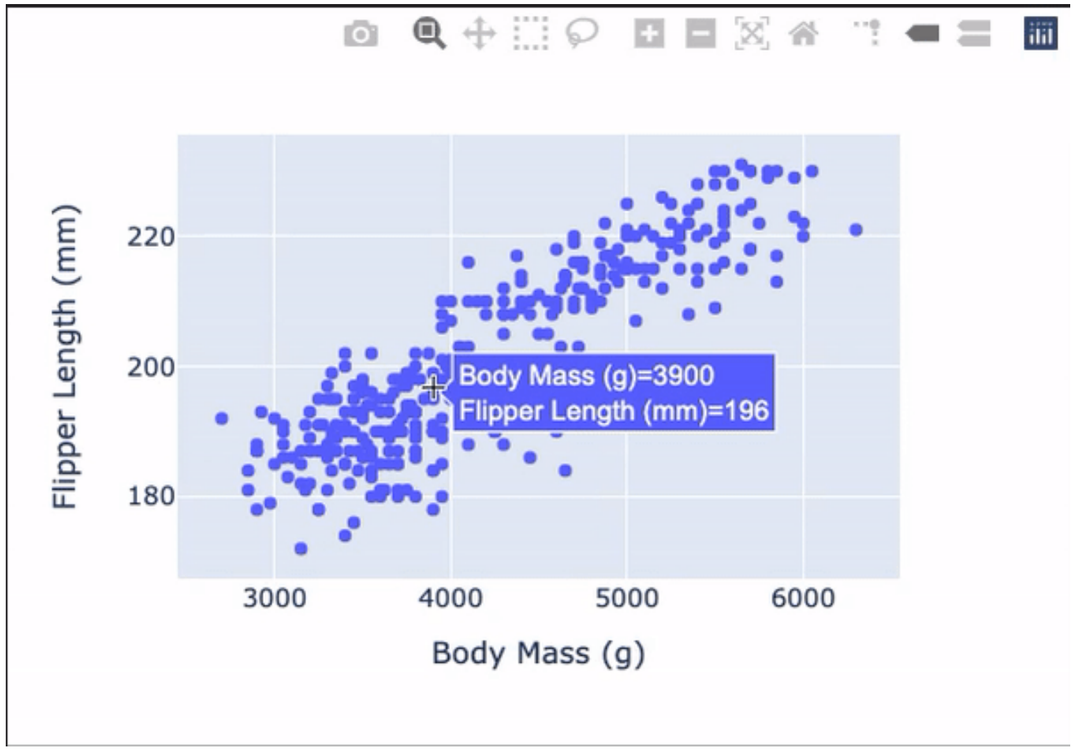
- A y-axis representing one variable
- An x-axis representing a different variable
- Each point is a dot on the graph, e.g., (68, 472)



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

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

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

```
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

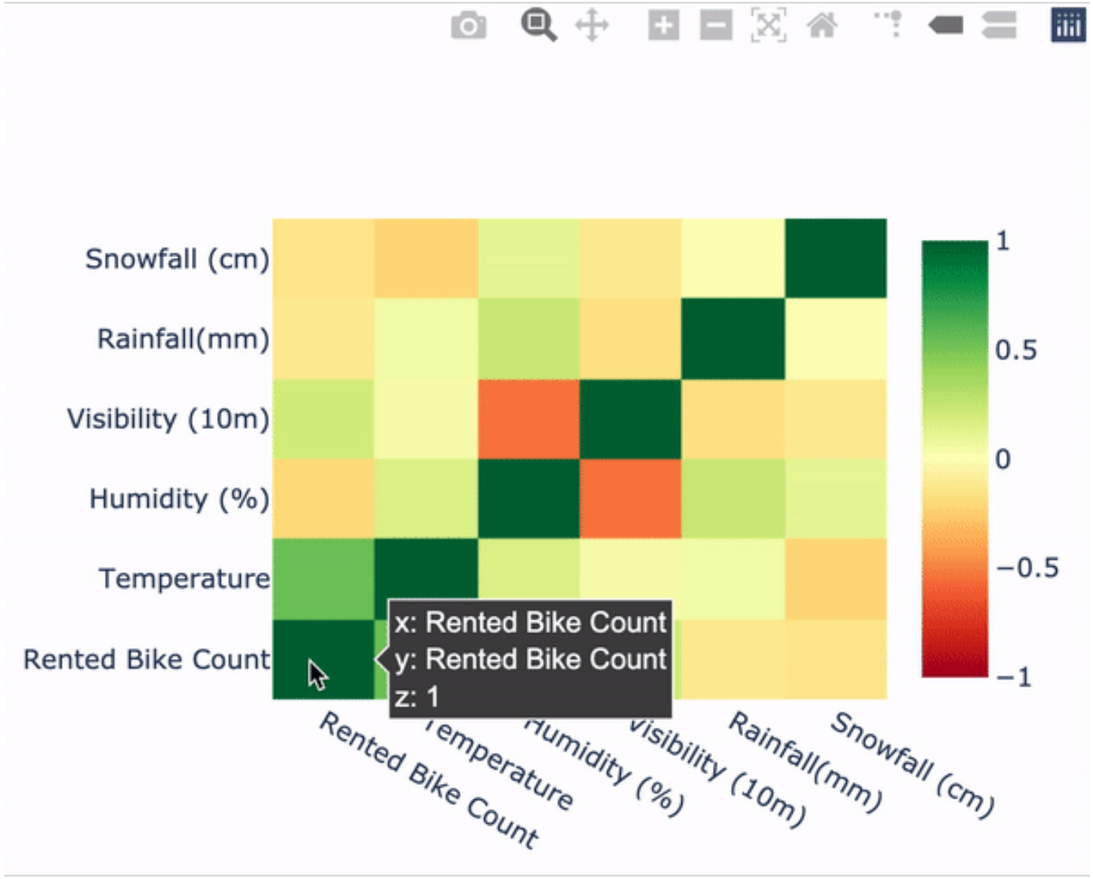
Correlation plot with Plotly

Let's build a correlation plot:

```
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

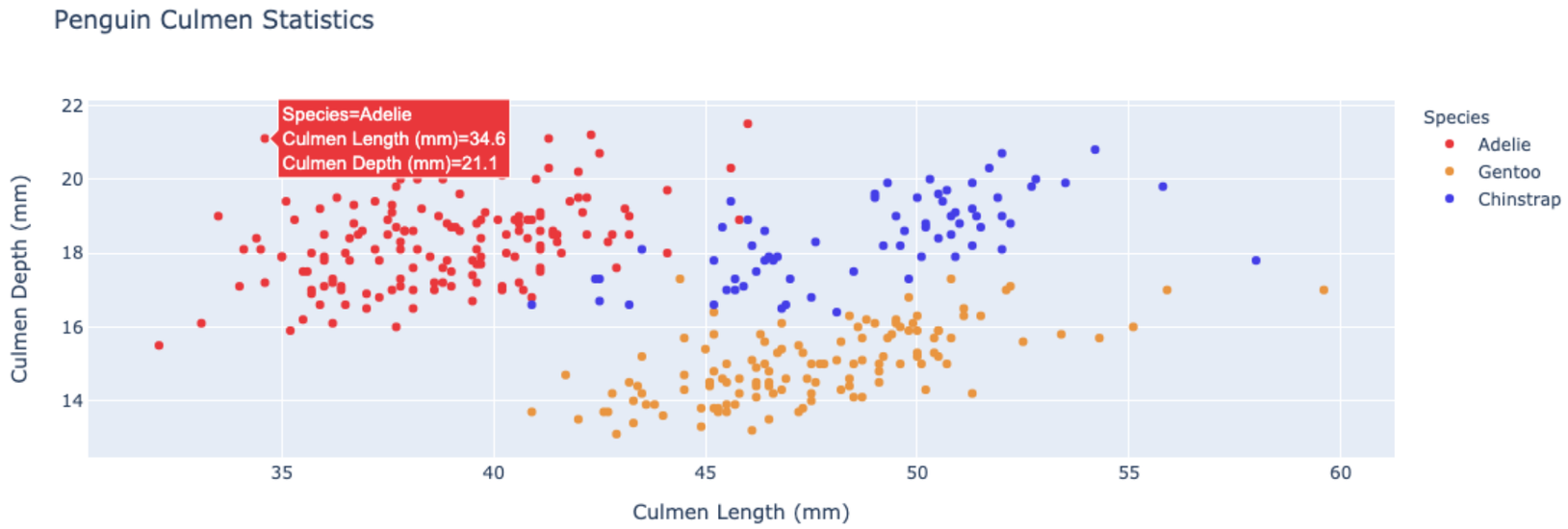
Alex Scriven
Data Scientist

datacamp

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.

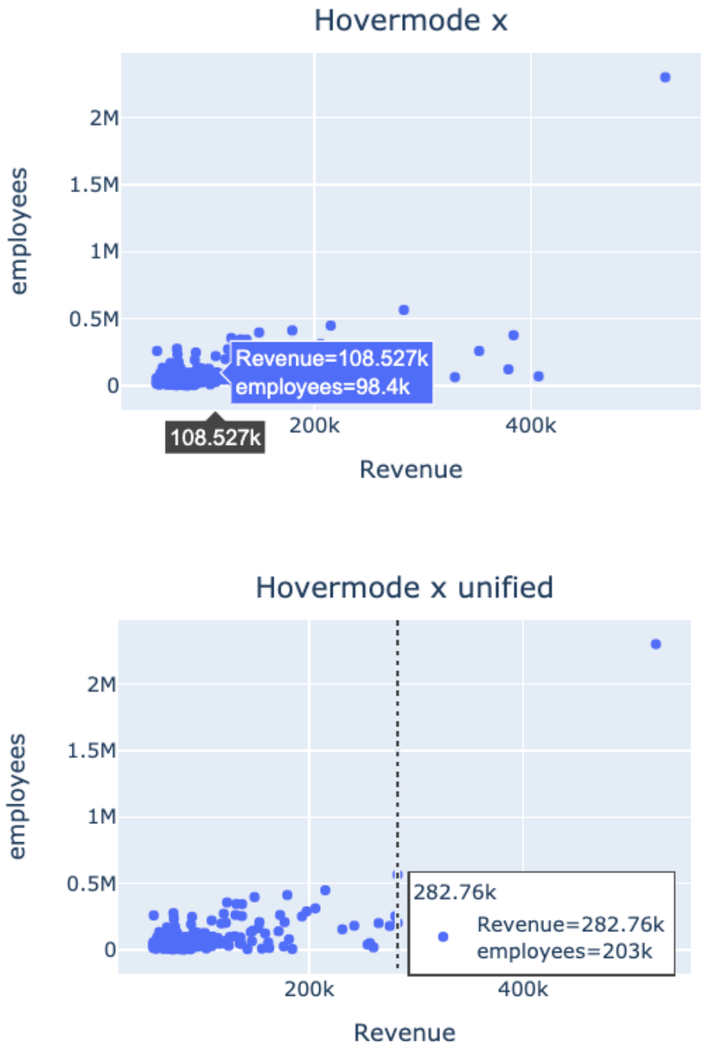
By default, you get some hover information already:



Other default hover information

The relevant layout argument is `hovermode` (defaults to `closest`), 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

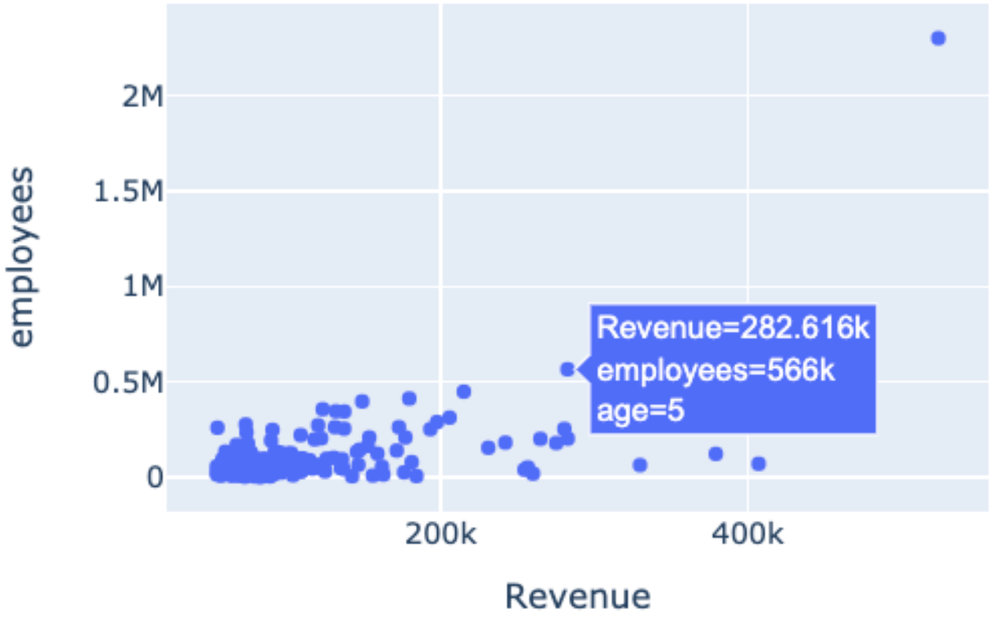
Variables in hover information

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

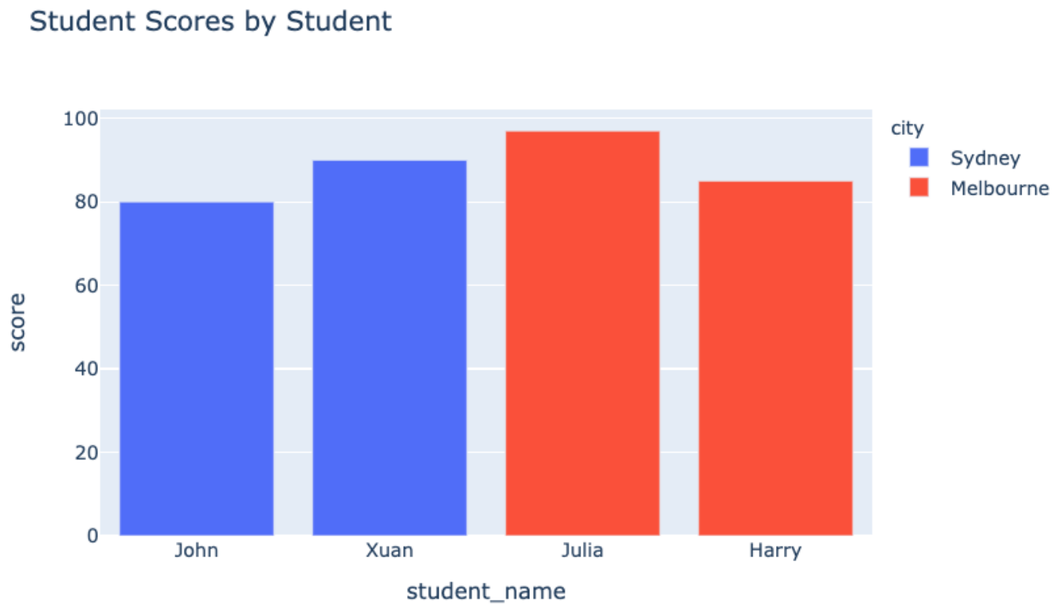
There are two main ways to style hover information:

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()`

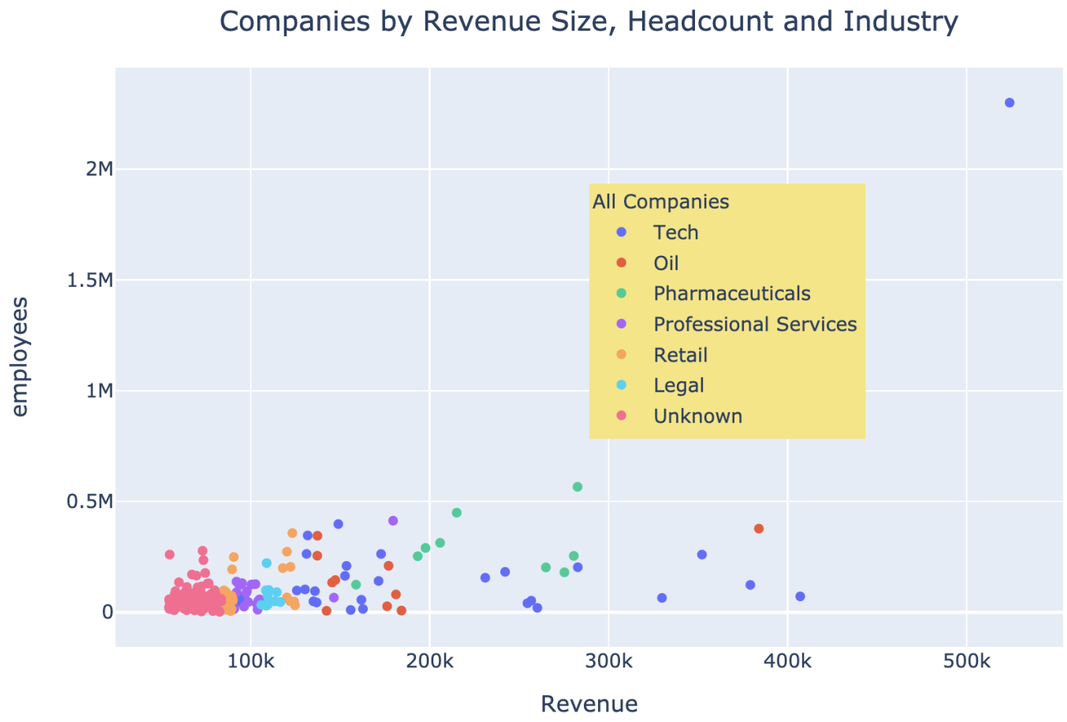
- `showlegend` = `True` shows the default legend
- `legend` = a dictionary specifying styles and positioning of the legend
  - `x` , `y` : (0-1) the percentage across x or y axis to position
  - Other stylistic elements such as `bgcolor` (background color), `borderwidth` , `title` , and `font`

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

datacamp

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

datacamp

Our dataset

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

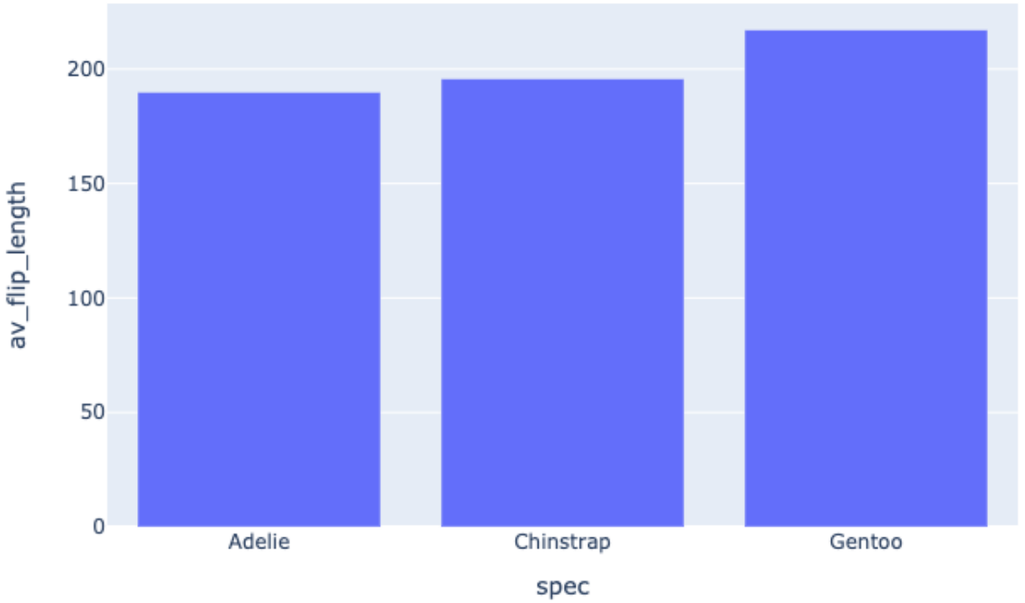
spec	av_flip_length
Adelie	189.953642
Chinstrap	195.823529
Gentoo	217.186992

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')
```

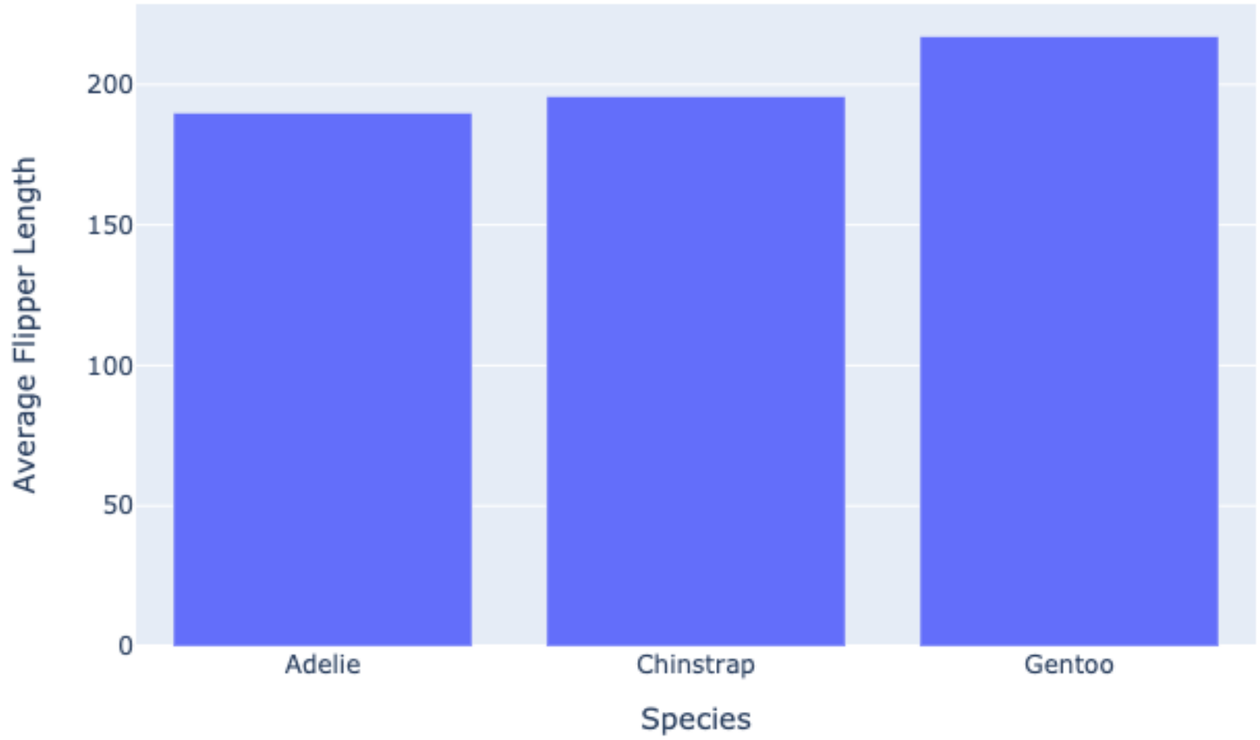
Or with the more general `update_layout()`

```
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

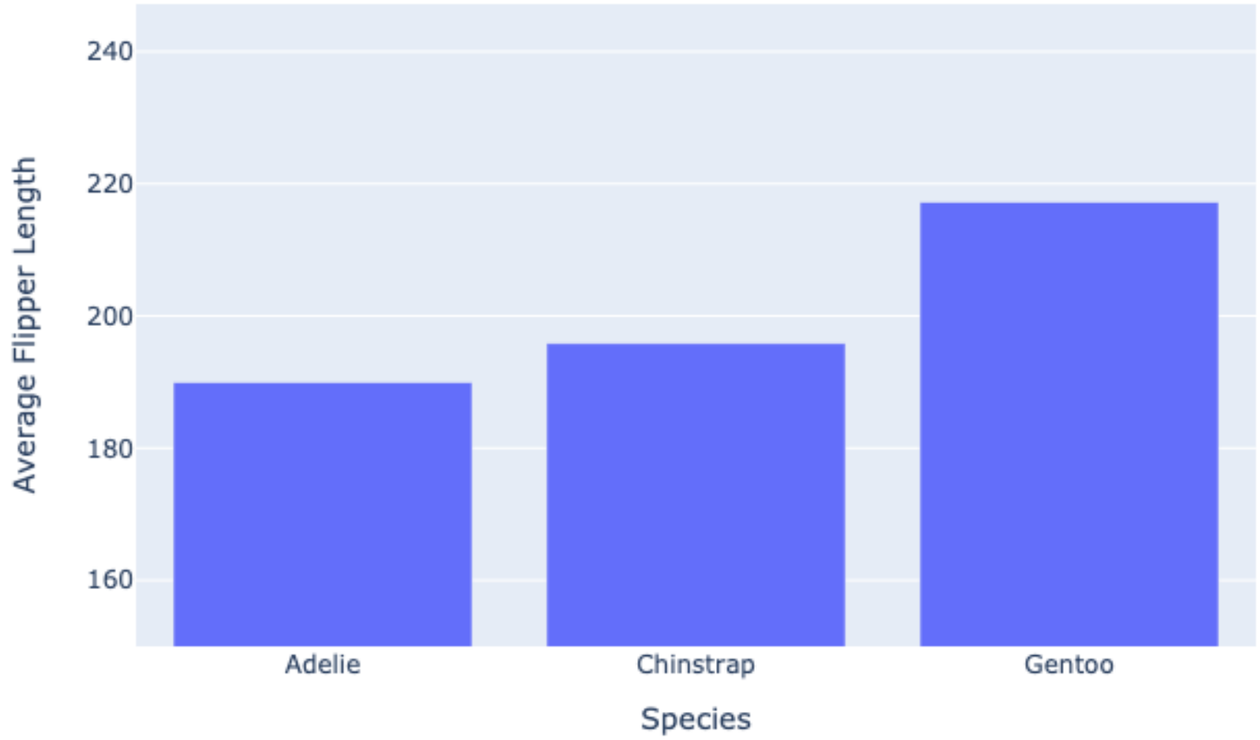
Plotly automatically calculates axes ranges from your data - this may not be desired!

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?

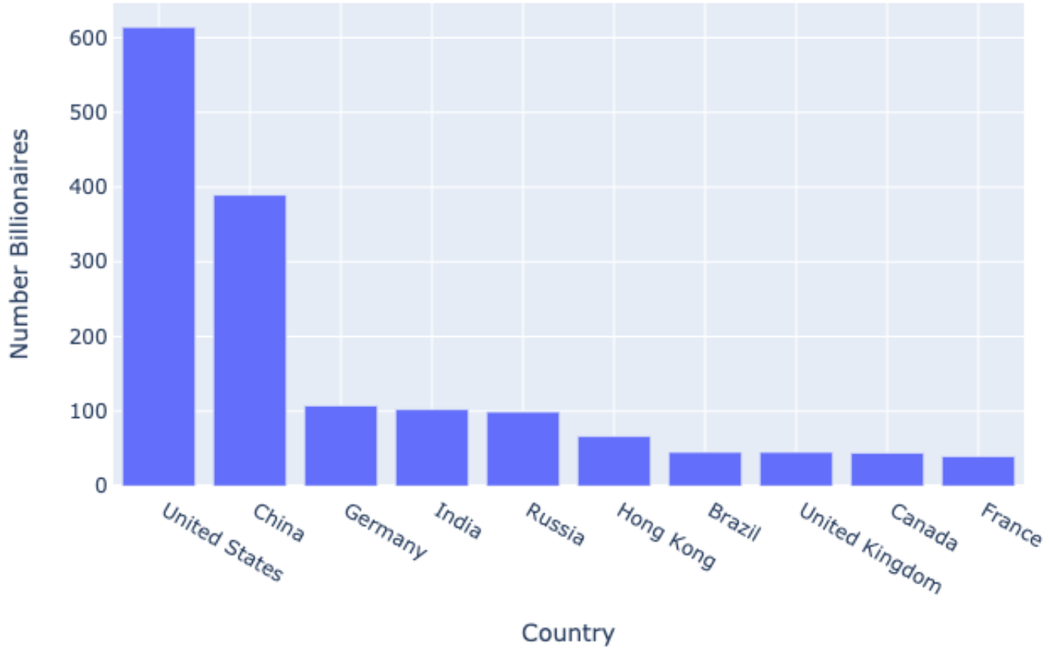
- Top 10 countries by number of billionaires

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

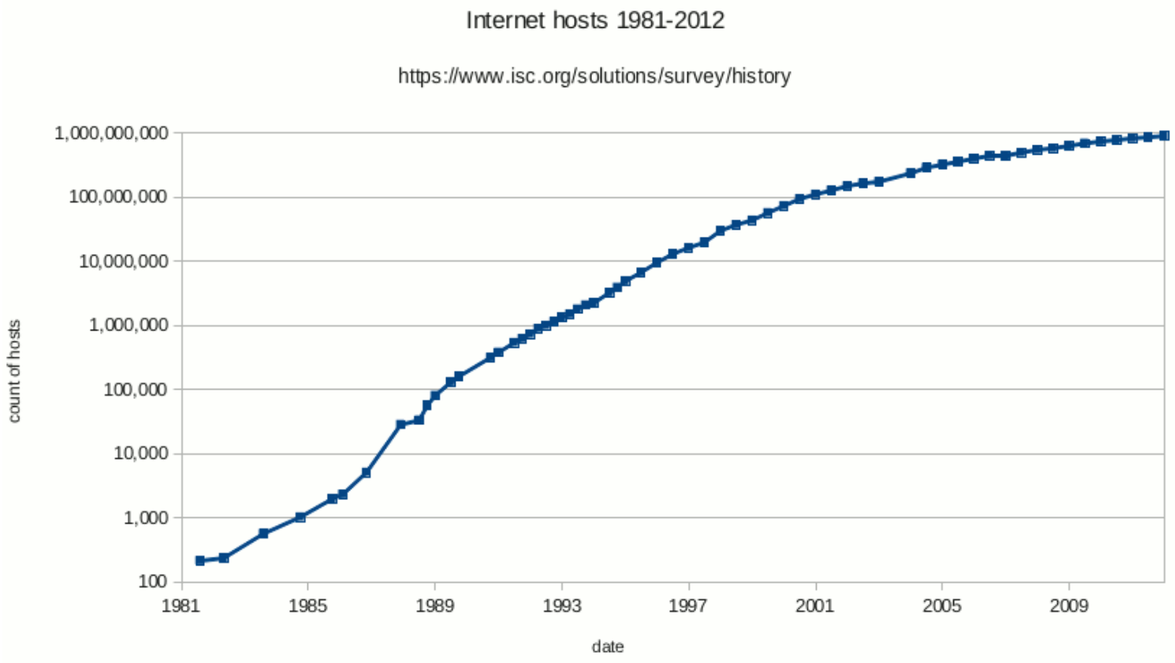
Let's plot without any adjustment:

```
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:



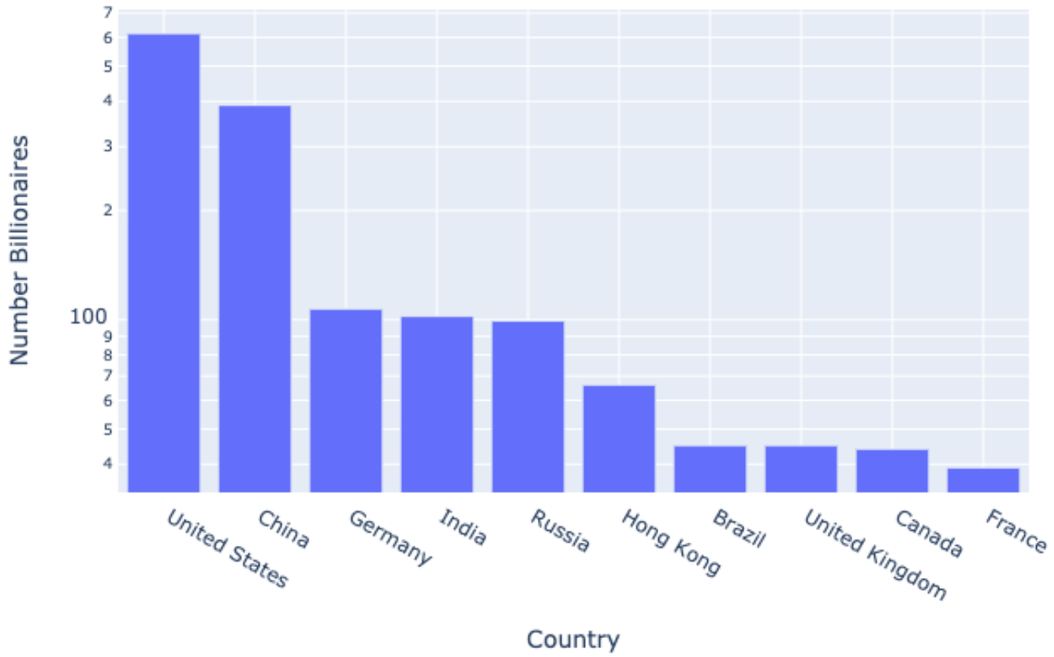
Ticks on our y-axis aren't uniform (10,20, 30, etc.)

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