

Using Plotly to Make Figures and Charts

Why Use Plotly?

Plotly is a good choice for several reasons:

- It allows for easy interactive plotting
- Interactive plots can be embedded in notebooks
- Can be run on a server
- Plotly has developed a dashboard API to complement their plotting library (similar to Shiny for R)
- It also has a shorthand library `plotly_express` for rapid exploration

Getting Started

```
import plotly.express as px
```

First, we want to import `plotly.express`, which will serve as the engine for creating our figures in `plotly`.

Using Existing Data

Let's import a `pandas` Data Frame to play with some 🐟 data:

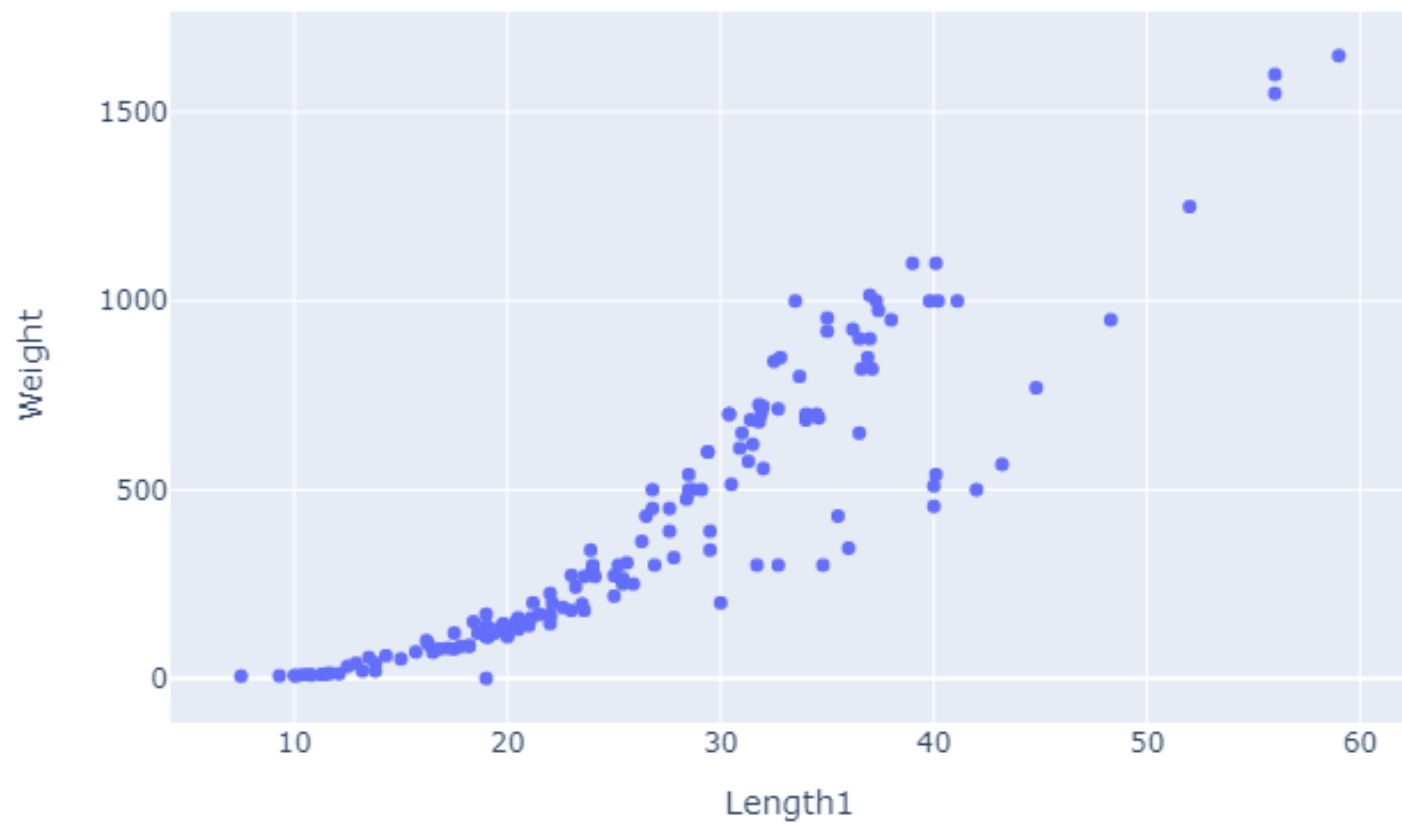
```
import pandas as pd

data = pd.read_csv(
    "https://github.com/dustywhite7/pythonMikkeli/raw/master/exampleData/fishWeight.csv")
```

Creating Plot Objects

```
px.scatter(data, x='Length1', y='Weight')
```

In this (very) simple example, we plot some time series data. Our figure is rendered in the notebook.

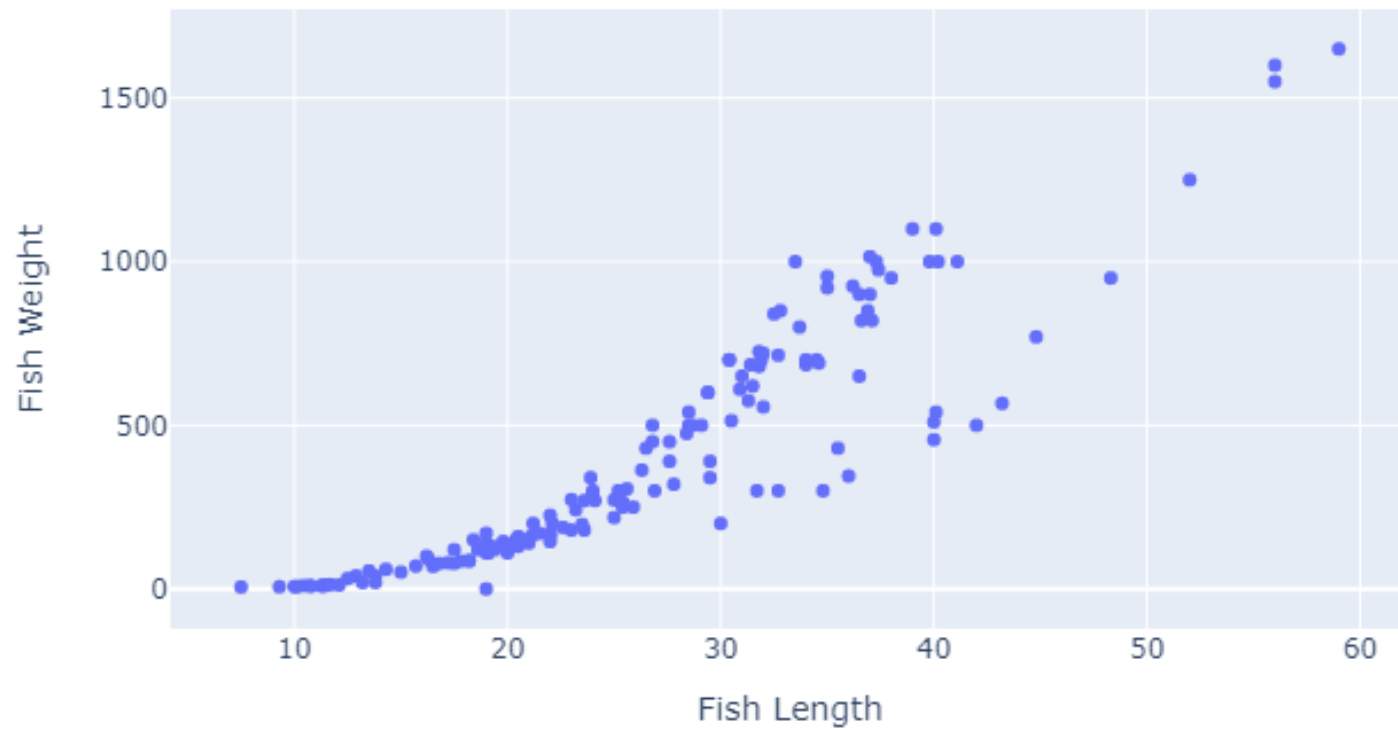


Formatting

Let's add some formatting. First, we can change the axis labels and title to match :

```
px.scatter(data, x='Length1', y='Weight',  
           title = "Fish Length vs Weight", # update the title of the figure  
           labels = { # dictionary for axis labels  
               'Length1' : 'Fish Length', # key should match original label  
               'Weight' : "Fish Weight" # value should be new label value  
           })
```

Fish Length vs Weight



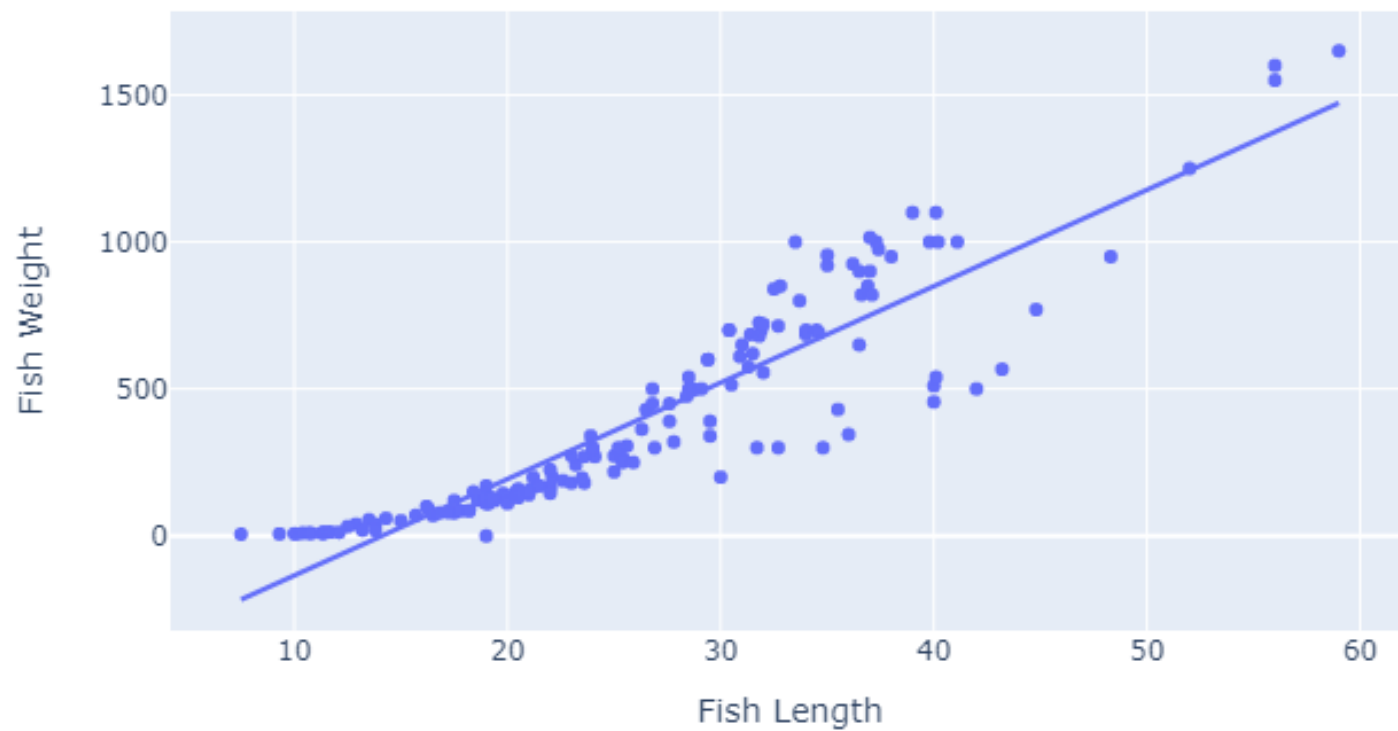
Trendlines

Next, we can add a regression trendline:

```
px.scatter(data, x='Length1', y='Weight',  
           title = "Fish Length vs Weight", # update the title of the figure  
           labels = { # dictionary for axis labels  
               'Length1' : 'Fish Length', # key should match original label  
               'Weight' : "Fish Weight" # value should be new label value  
           },  
           trendline = 'ols' # add a linear trendline  
           )
```

We can also use `lowess` trendlines!

Fish Length vs Weight

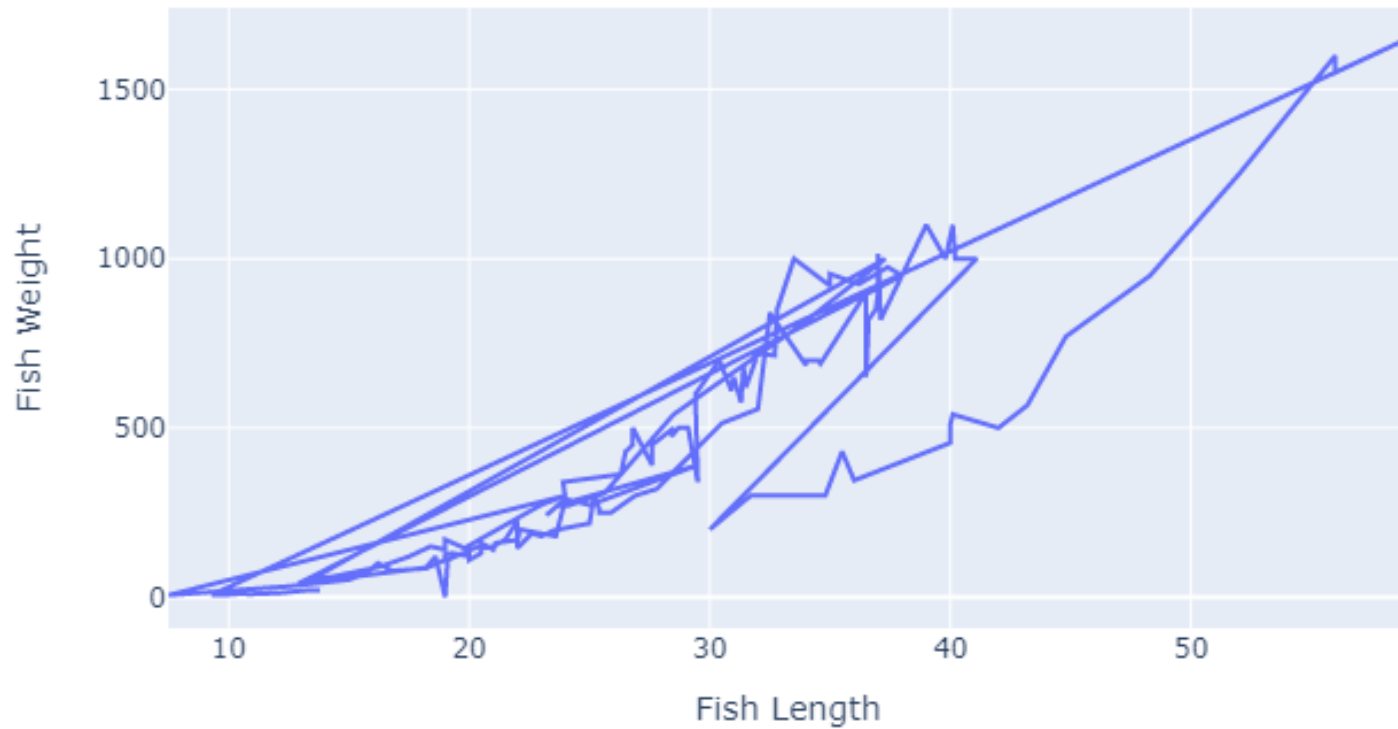


Line Charts

We could instead use line charts

```
px.line(data, x='Length1', y='Weight',  
        title = "Fish Length vs Weight", # update the title of the figure  
        labels = { # dictionary for axis labels  
            'Length1' : 'Fish Length', # key should match original label  
            'Weight' : "Fish Weight" # value should be new label value  
        })
```

Fish Length vs Weight



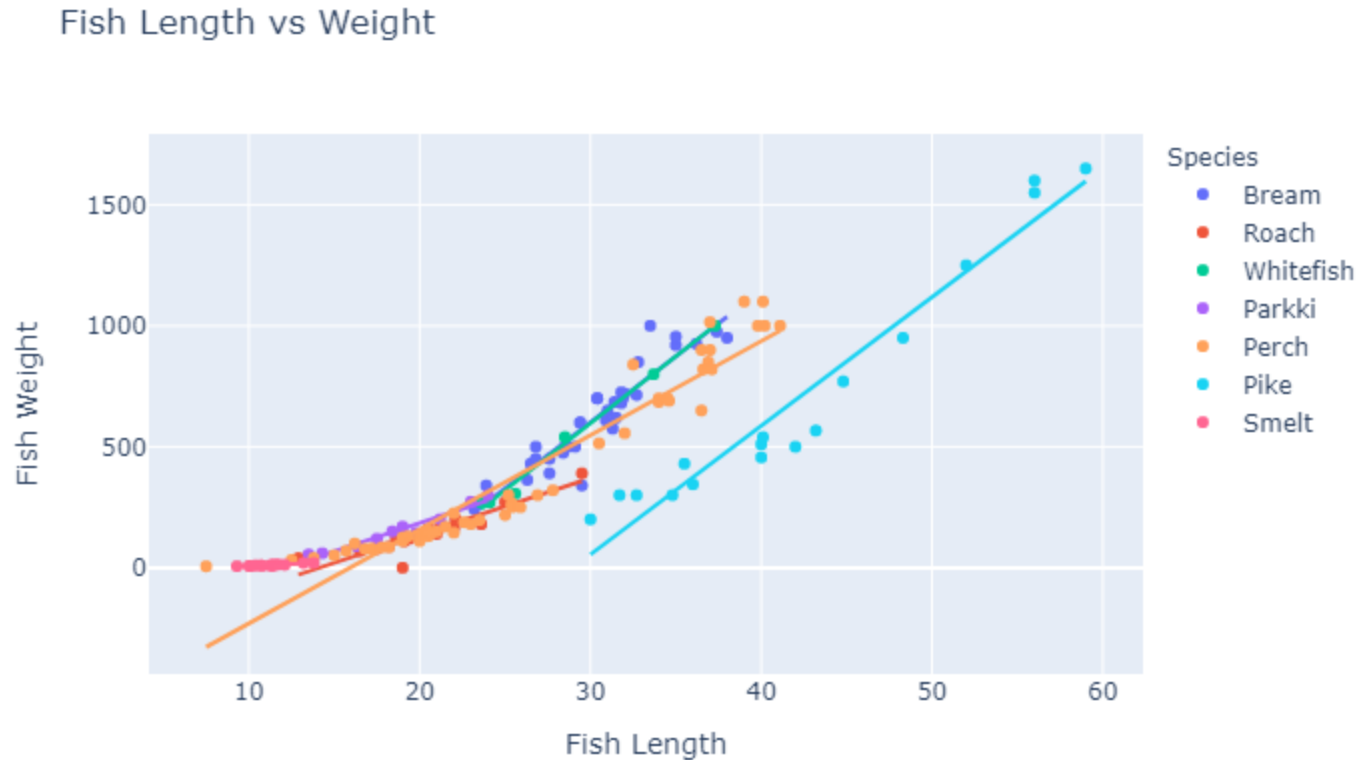
Clearly, not helpful here...

Creating Plot Objects

Let's mark multiple series by separating our observations by fish species:

```
px.scatter(data, x='Length1', y='Weight',  
           title = "Fish Length vs Weight", # update the title of the figure  
           labels = { # dictionary for axis labels  
               'Length1' : 'Fish Length', # key should match original label  
               'Weight' : "Fish Weight" # value should be new label value  
           },  
           trendline = 'ols', # add a linear trendline,  
           color = 'Species'  
)
```

Creating Plot Objects



Note that we even get a separate trend line for each color group! 😊

Other Plot Types

We can do a LOT more than scatter plots!

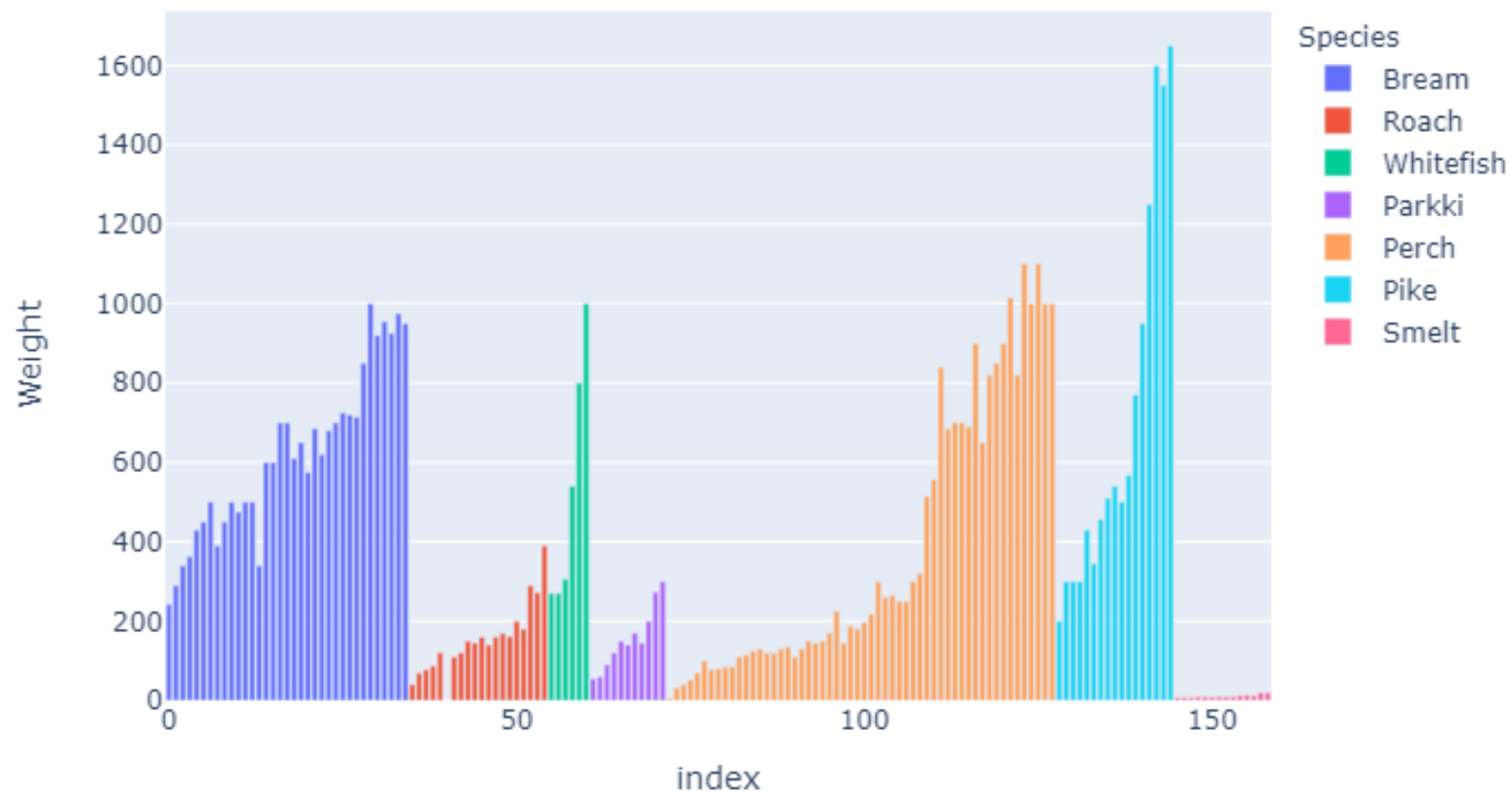
- Bar Charts
- Box Plots
- Histograms, with distribution stats, too!
- Heatmaps
- Choropleth, Line, and Bubble Maps

among many others.

Using Bar Charts

First, we can make a bar chart:

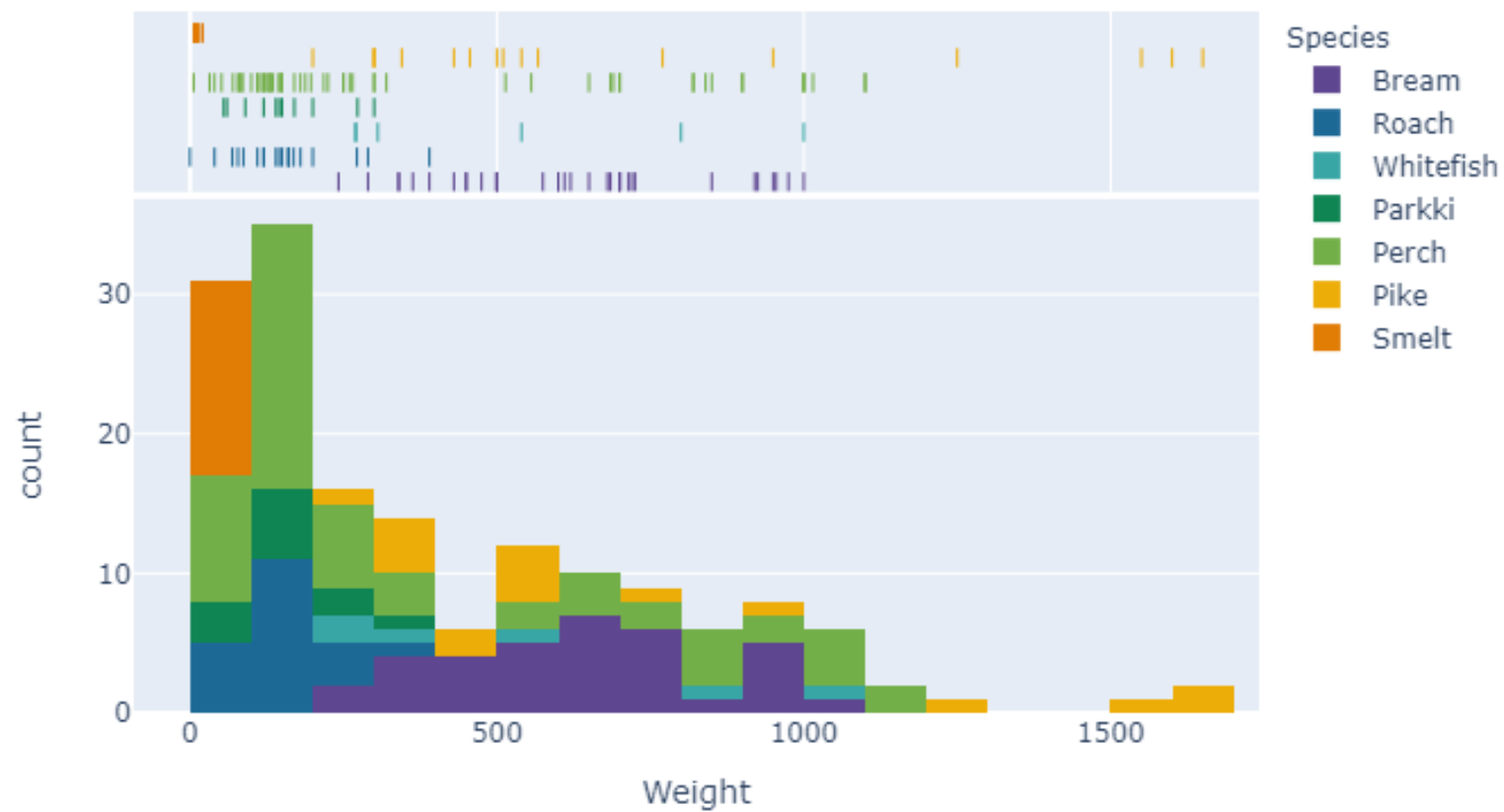
```
px.bar(data, y="Weight", color="Species")
```

Histogram

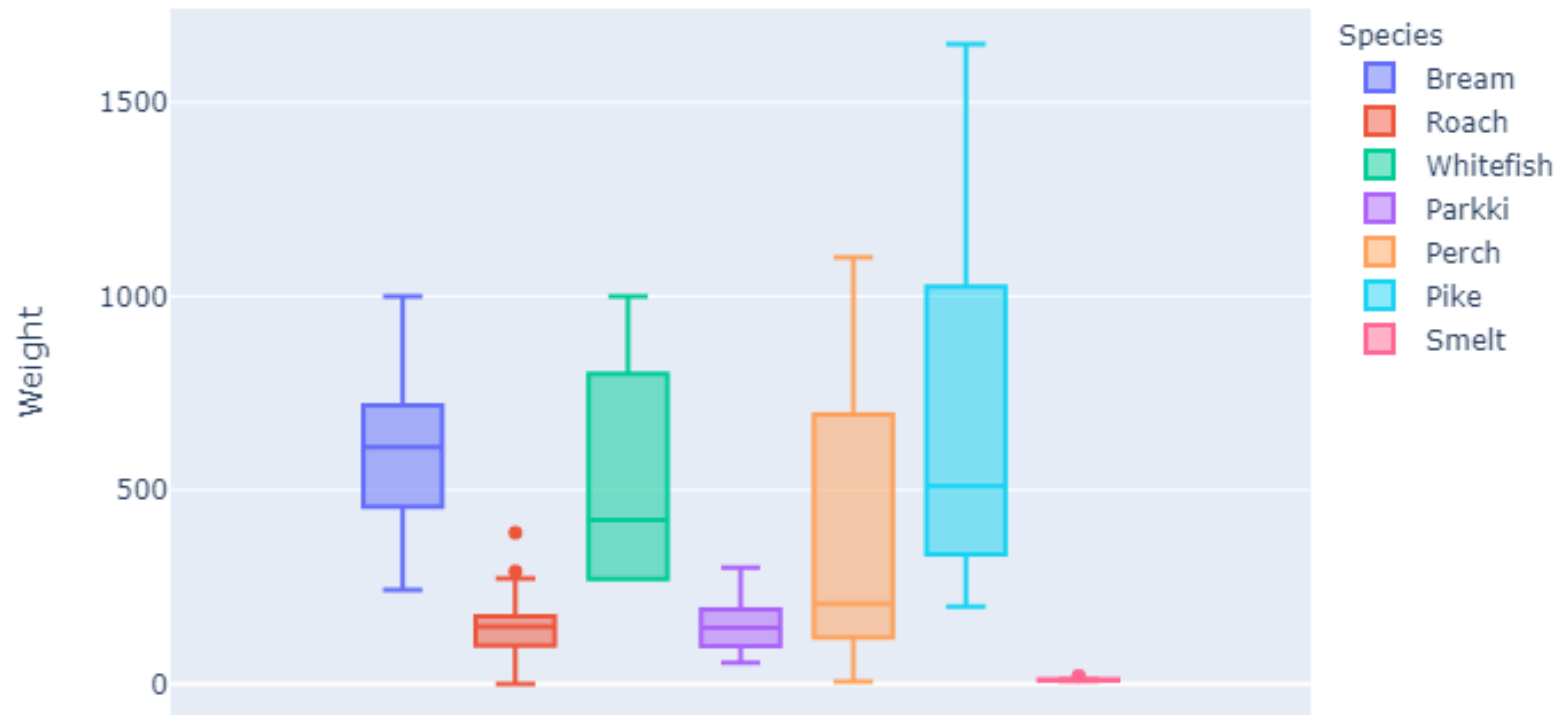
Maybe that data would do better if we could aggregate it in bins to better understand how many fish were observed in each weight bin:

```
px.histogram(data,  
             x="Weight",  
             marginal="rug",  
             color="Species",  
             color_discrete_sequence=px.colors.qualitative.Prism)
```



Box Plots

```
px.box(data, y="Weight", color="Species")
```



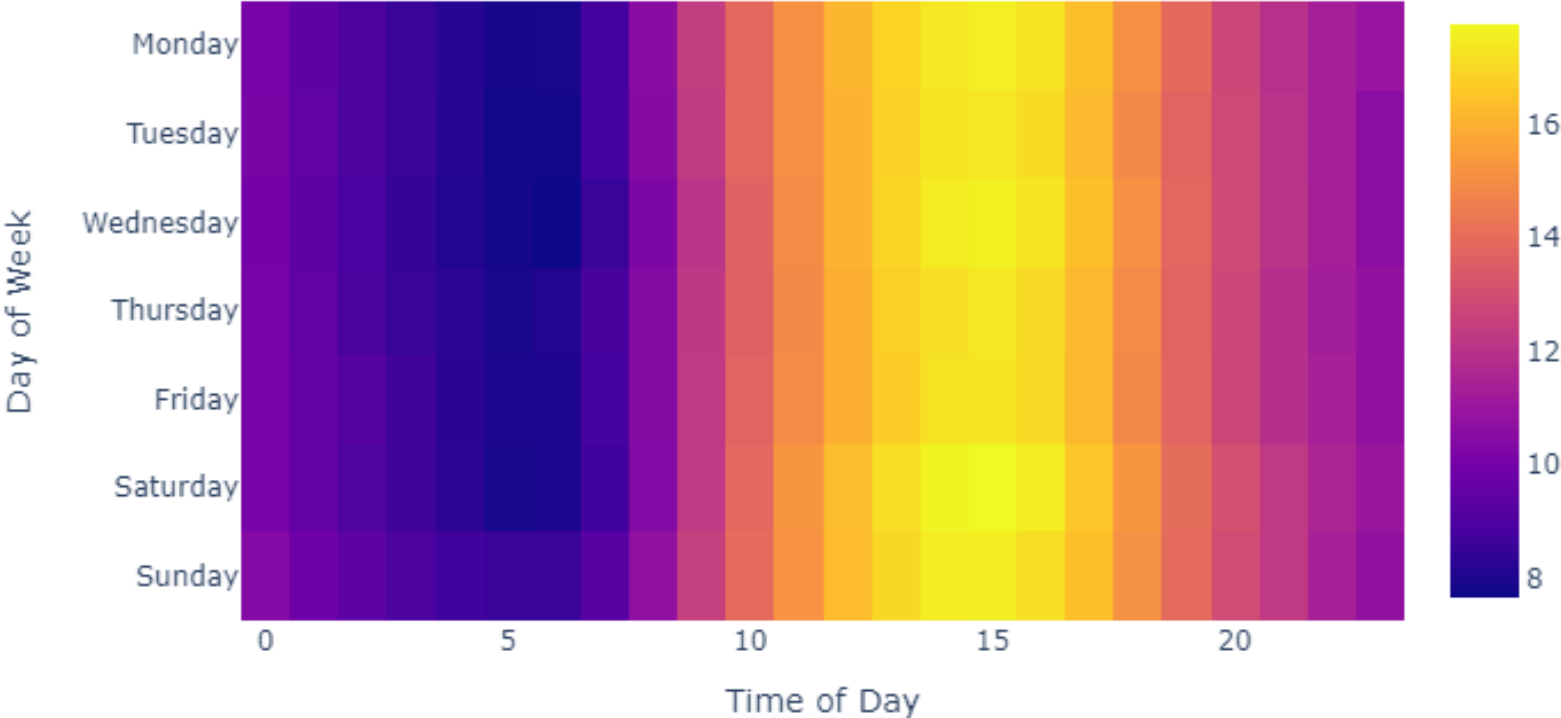
Heatmaps

```
data = pd.read_csv(
    "https://raw.githubusercontent.com/dustywhite7/pythonMikkeli/master/exampleData/pollutionBeijing.csv")

data['datetime'] = pd.to_datetime(data['datetime'])
data['weekday'] = data['datetime'].dt.dayofweek
data['hour'] = data['datetime'].dt.hour
data = data.groupby(['weekday', 'hour'])['TEMP'].mean()
data = data.values.reshape((7,24))

px.imshow(data, title="Temperature in Beijing" ,
           labels=dict(y="Day of Week", x="Time of Day"),
           y=['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'])
```

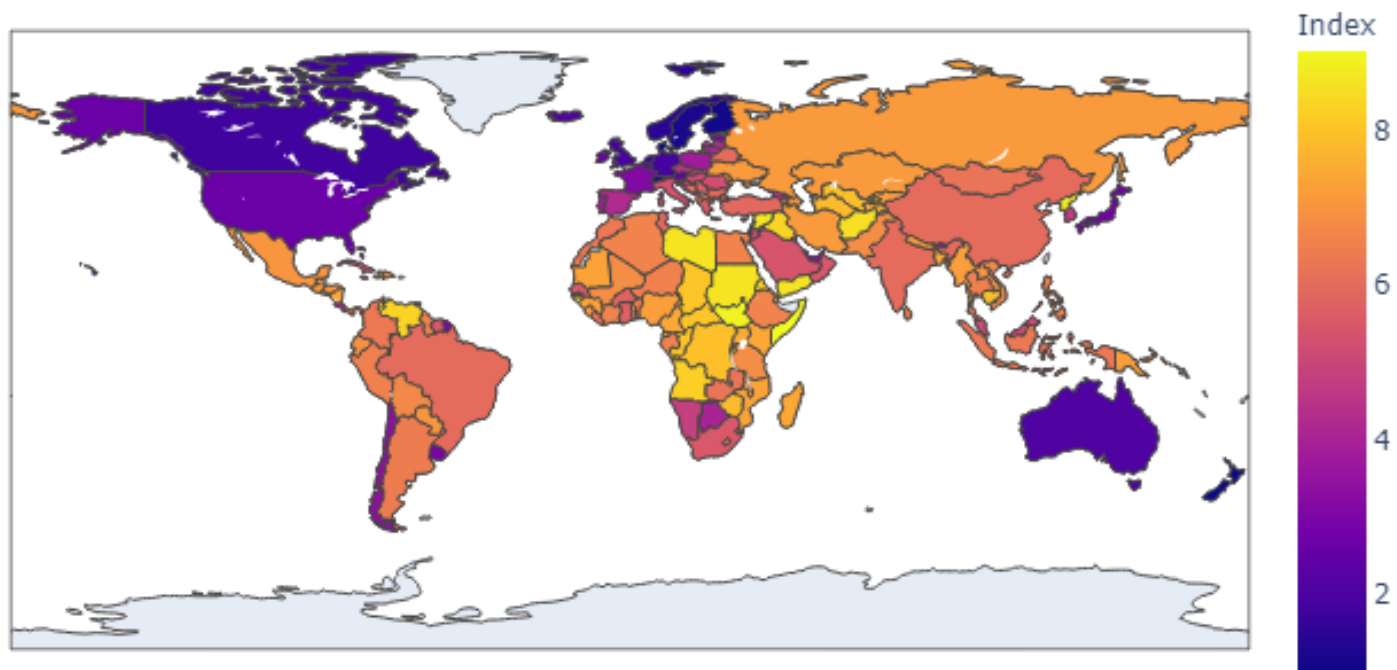
Temperature in Beijing



Choropleth Maps

```
data = pd.read_csv(  
    "https://raw.githubusercontent.com/dustywhite7/Econ8320/master/LabCode/corruption2018.csv")  
  
px.choropleth(data, locations = 'Abbr',  
    color = 'Index',  
    hover_name= "Name"  
    )
```

Map data from the [INFORM Index](#)



Mapping Options: Layout->Geo

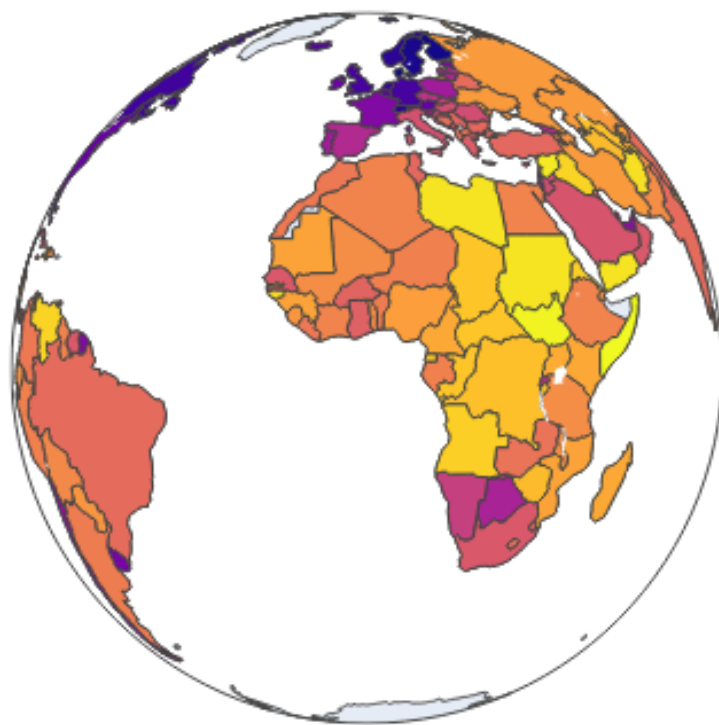
We have many additional options that we can pass to the layout of our plot when dealing with geographic data.

- Map projection
- Map scope
- Country lines
- Lots more

Here is a link to the [full documentation](#)

Choropleth Maps - Projection

```
px.choropleth(data, locations = 'Abbr',  
              color = 'Index',  
              hover_name= "Name",  
              projection = "orthographic"  
              )
```



Index

8

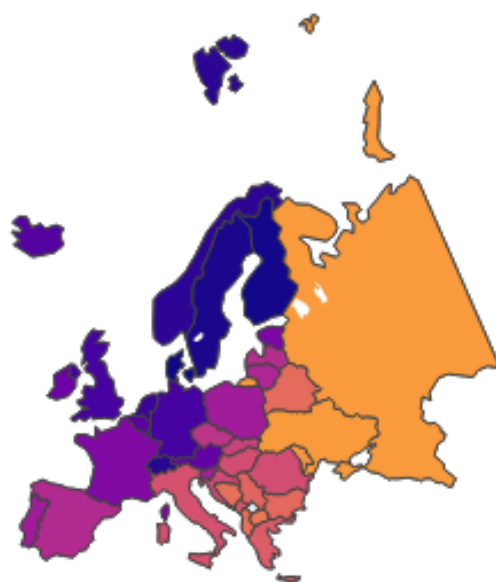
6

4

2

Choropleth Maps - Scope

```
px.choropleth(data, locations = 'Abbr',  
  color = 'Index',  
  hover_name= "Name",  
  scope = "europe"  
)
```



Index



8

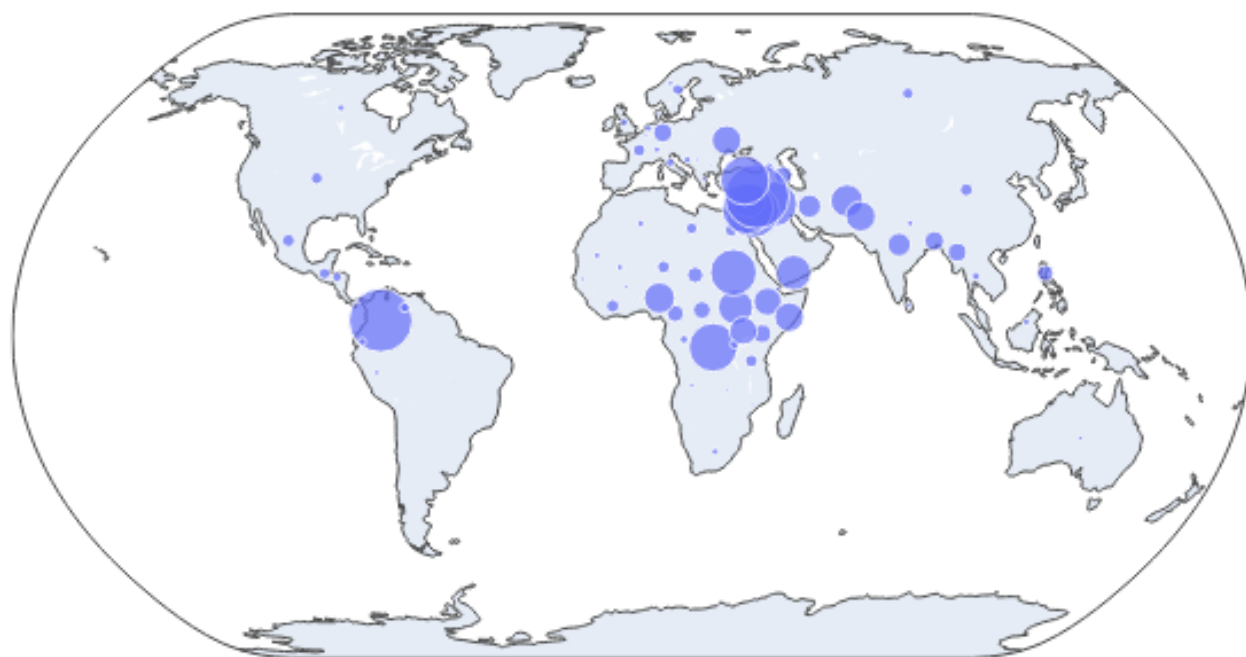
6

4

2

Bubble Maps

```
data = pd.read_csv(  
    "https://raw.githubusercontent.com/dustywhite7/Econ8320/master/LabCode/displaced2018.csv")  
  
px.scatter_geo(data, locations="Abbr",  
               hover_name="Name", size="Displaced",  
               projection="natural earth")
```



Lab Time!