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

```python
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:

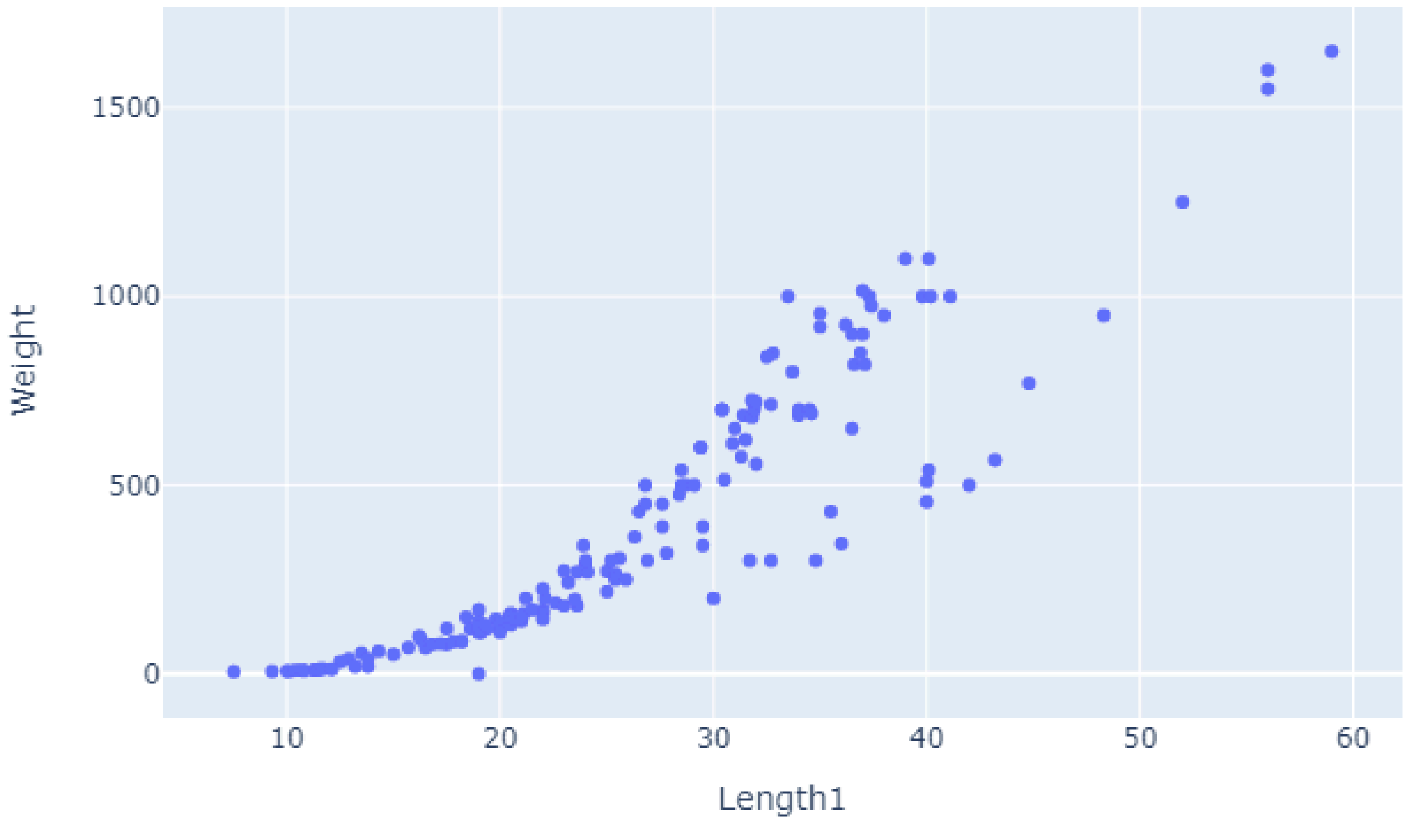
```python
import pandas as pd

data = pd.read_csv( # put link back on one line!
    "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 data about length and weight. Our figure is rendered in the notebook.

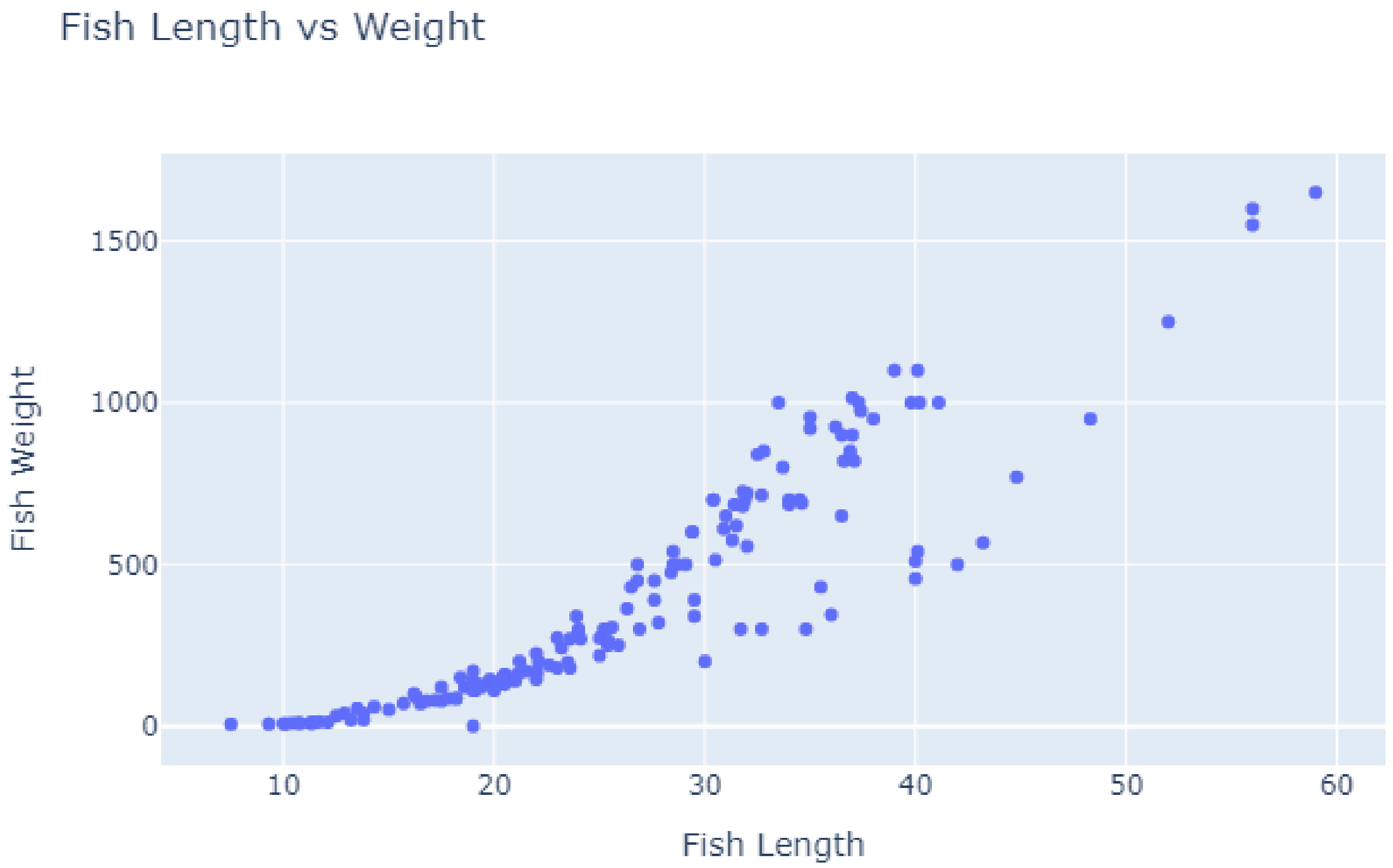


Weight
Length1
0
500
1000
1500
10
20
30
40
50
60

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
    })
```

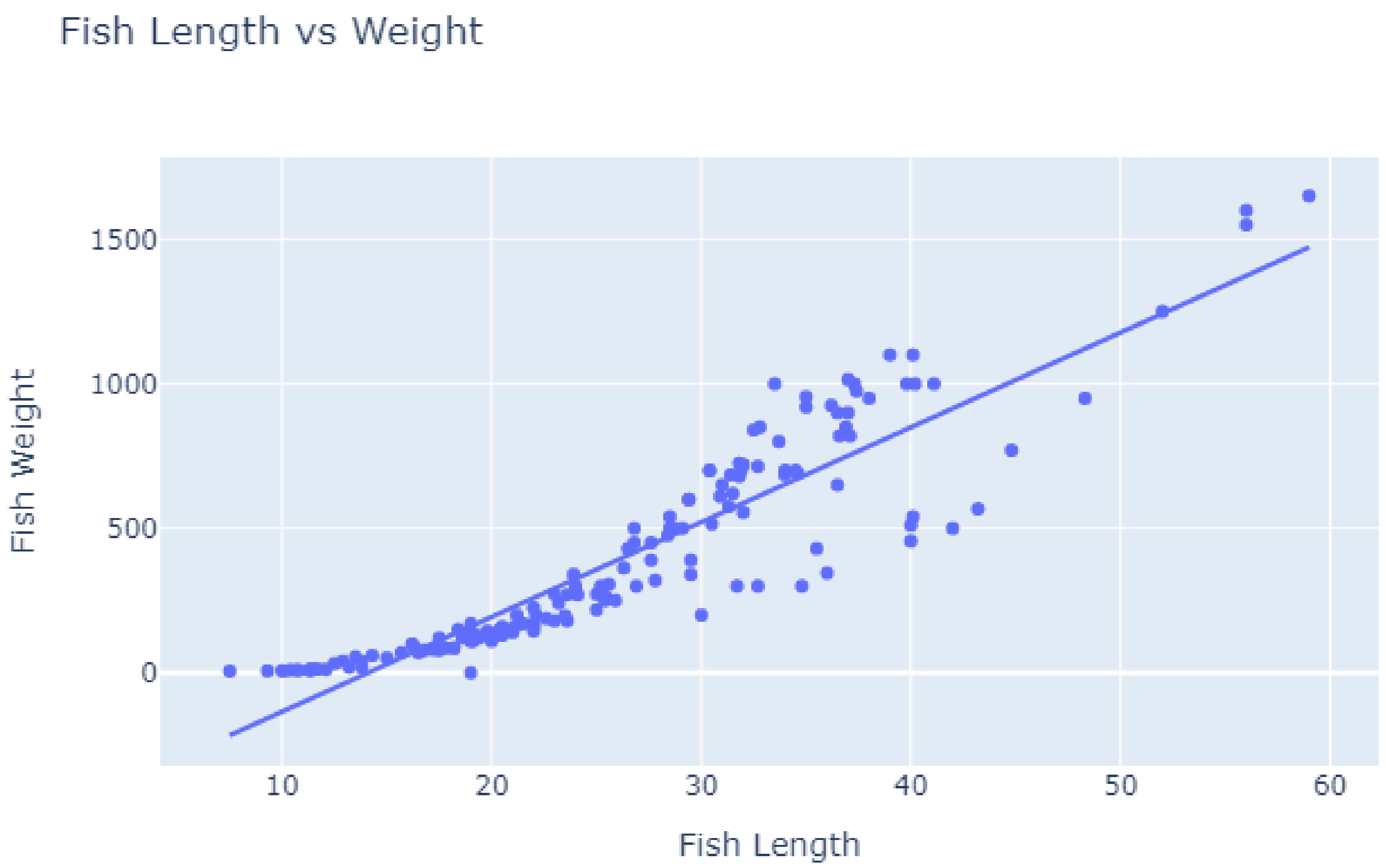

Fish Length vs Weight
Fish Weight
Fish Length
0
500
1000
1500
10
20
30
40
50
60

Trendlines

Next, we can add a regression trendline:

```python
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
)
```

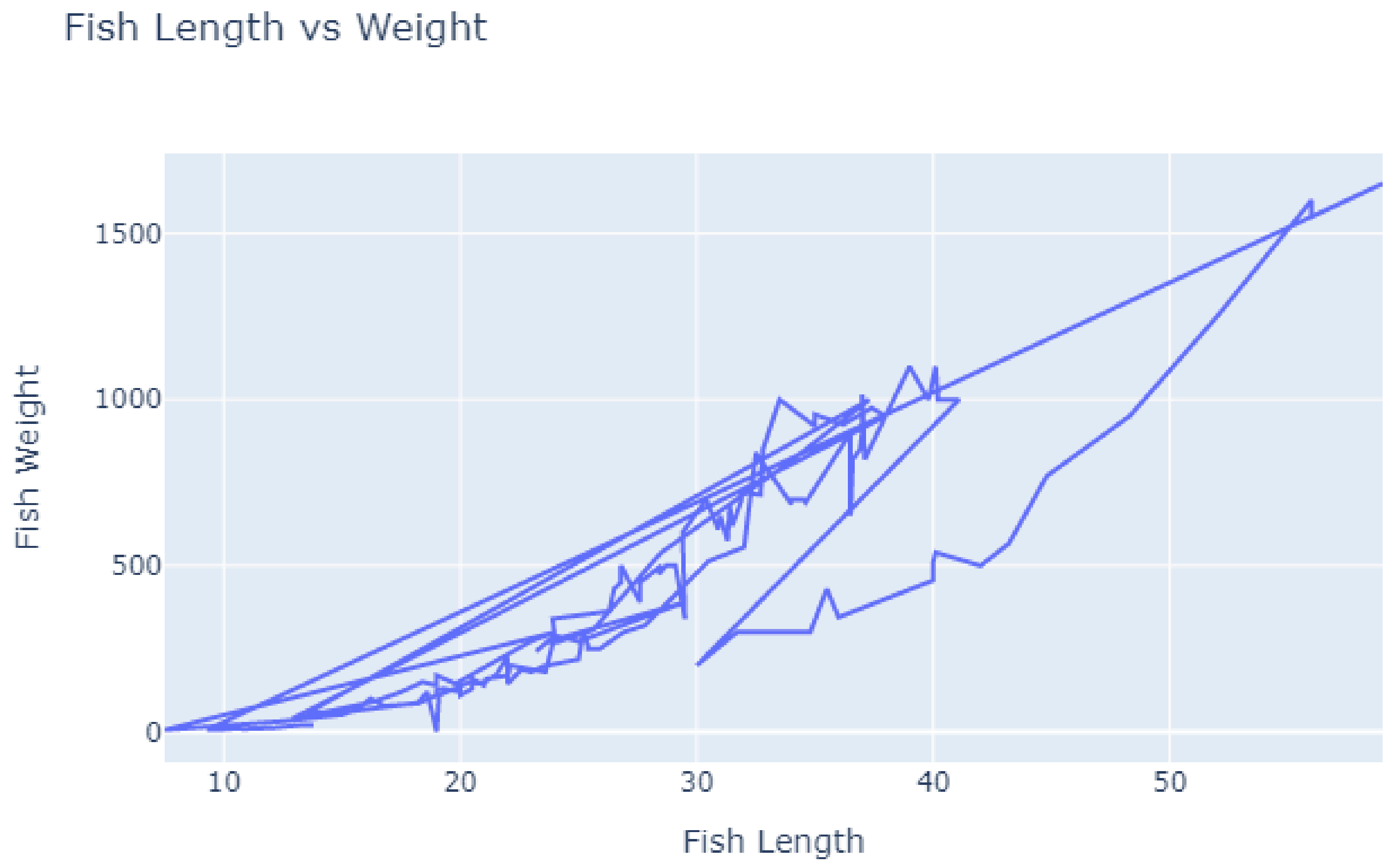
We can also use `lowess` trendlines!


Fish Length vs Weight
Fish Weight
1500
1000
500
0
10
20
30
40
50
60
Fish Length

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
    })
```



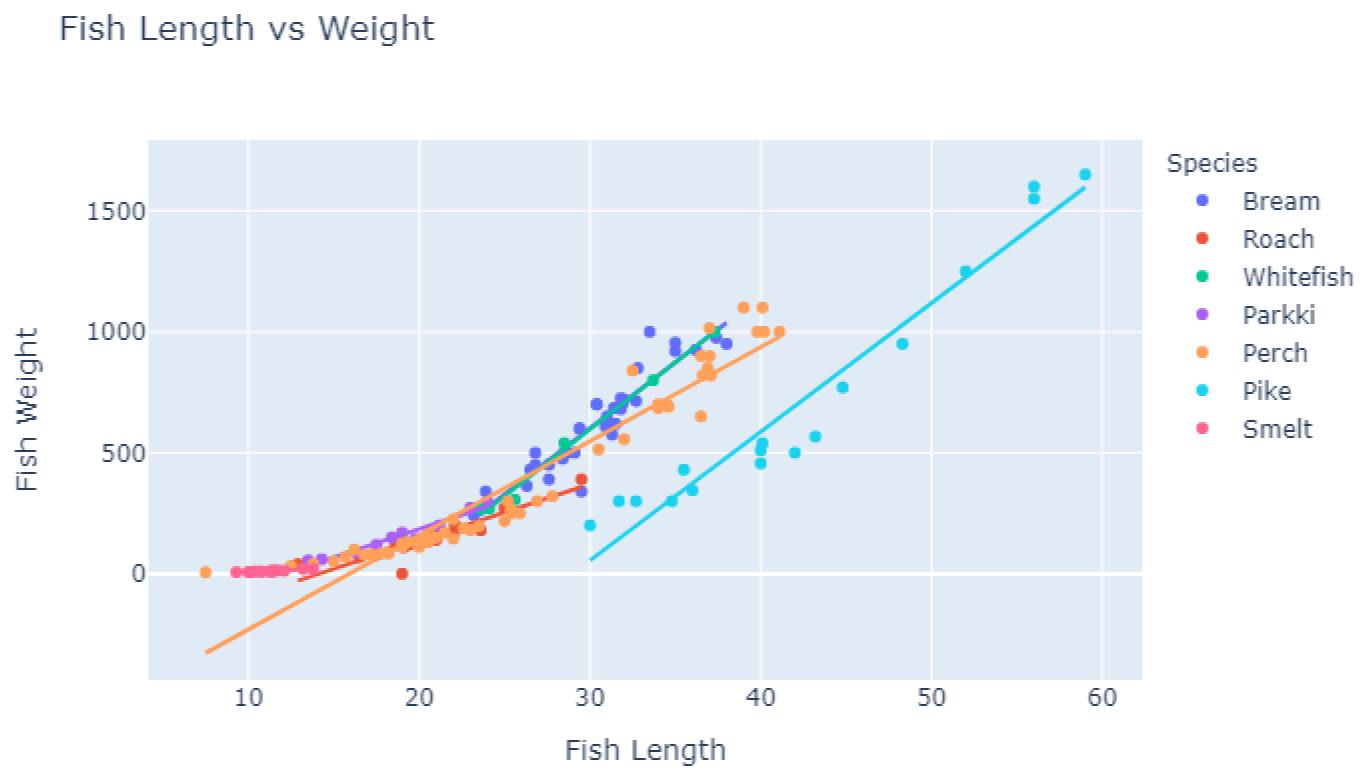
Clearly, not helpful here... (our data is not ordinal)

Creating Plot Objects

Let's show multiple series by separating our observations according to species:

```python
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!

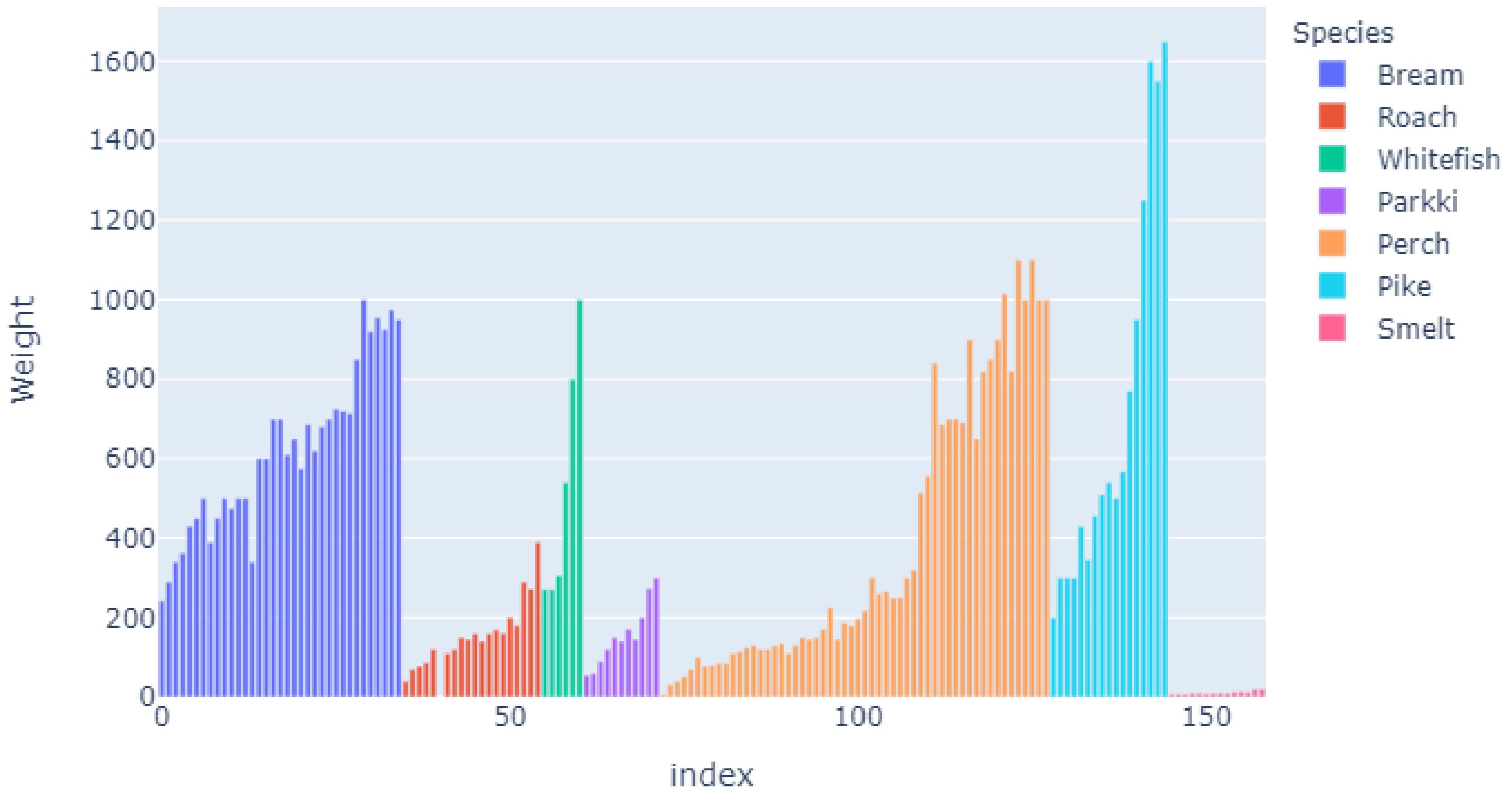
- Bar Charts
- Box Plots
- Histograms, with distribution stats, too!
- Heatmaps
- Choropleth, Line, and Bubble Maps

among many others.

Using Bar Charts

First, we can make a bar chart:

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

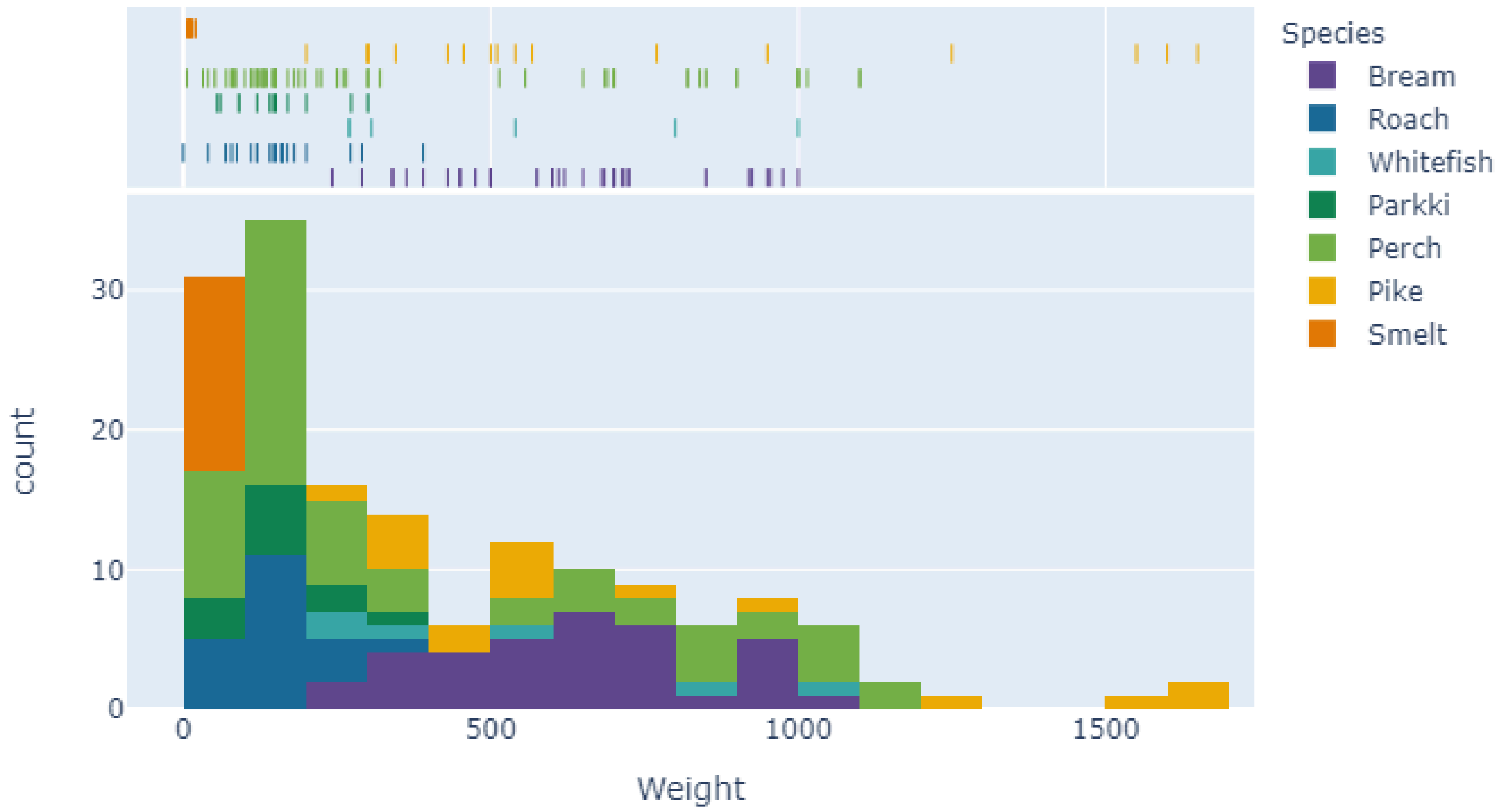


Weight
1600
1400
1200
1000
800
600
400
200
0
0
50
100
150
index
Species
Bream
Roach
Whitefish
Parkki
Perch
Pike
Smelt

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:

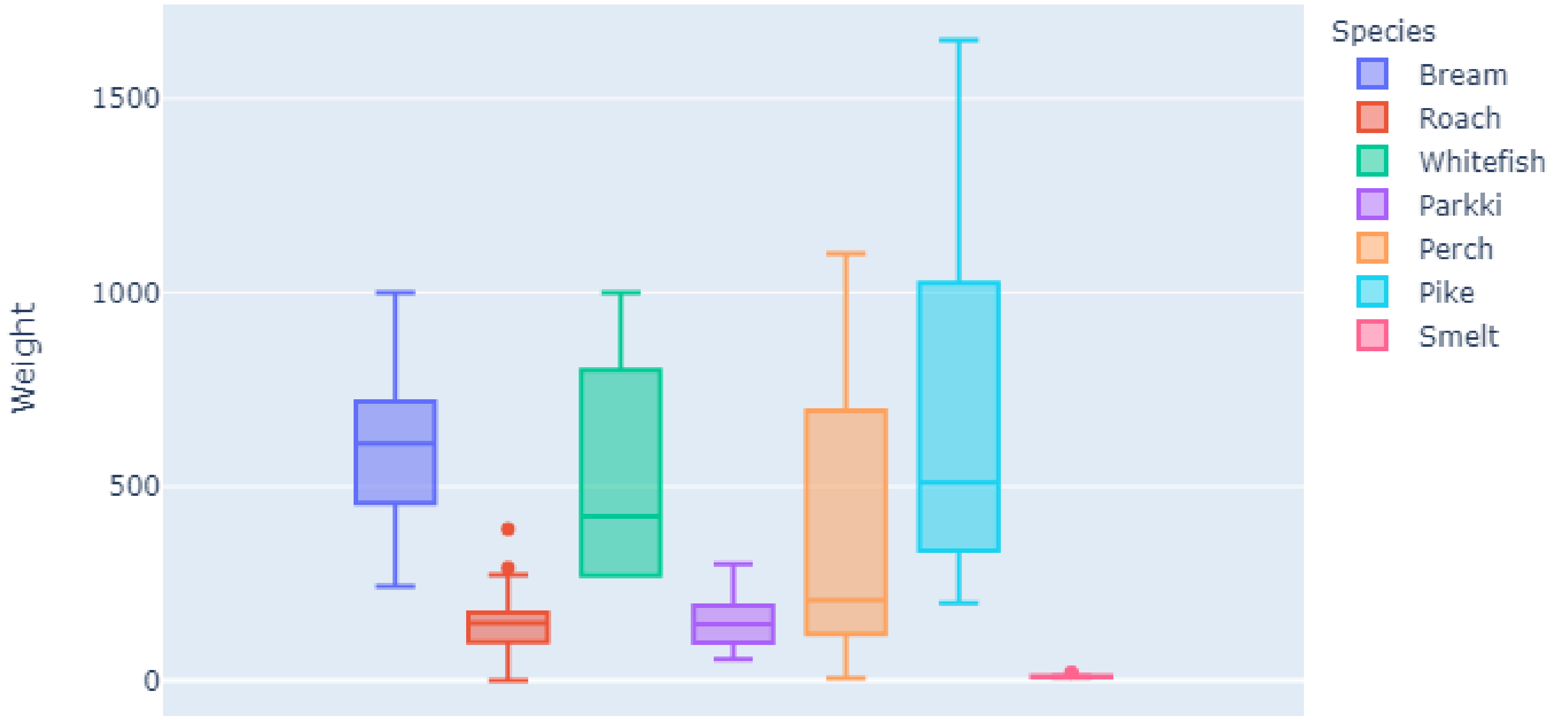
```
px.histogram(data,
             x="Weight",
             marginal="rug",
             color="Species",
             color_discrete_sequence=px.colors.qualitative.Prism)
```



Species
Bream
Roach
Whitefish
Parkki
Perch
Pike
Smelt
count
0
10
20
30
0
500
1000
1500
Weight

Box Plots

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

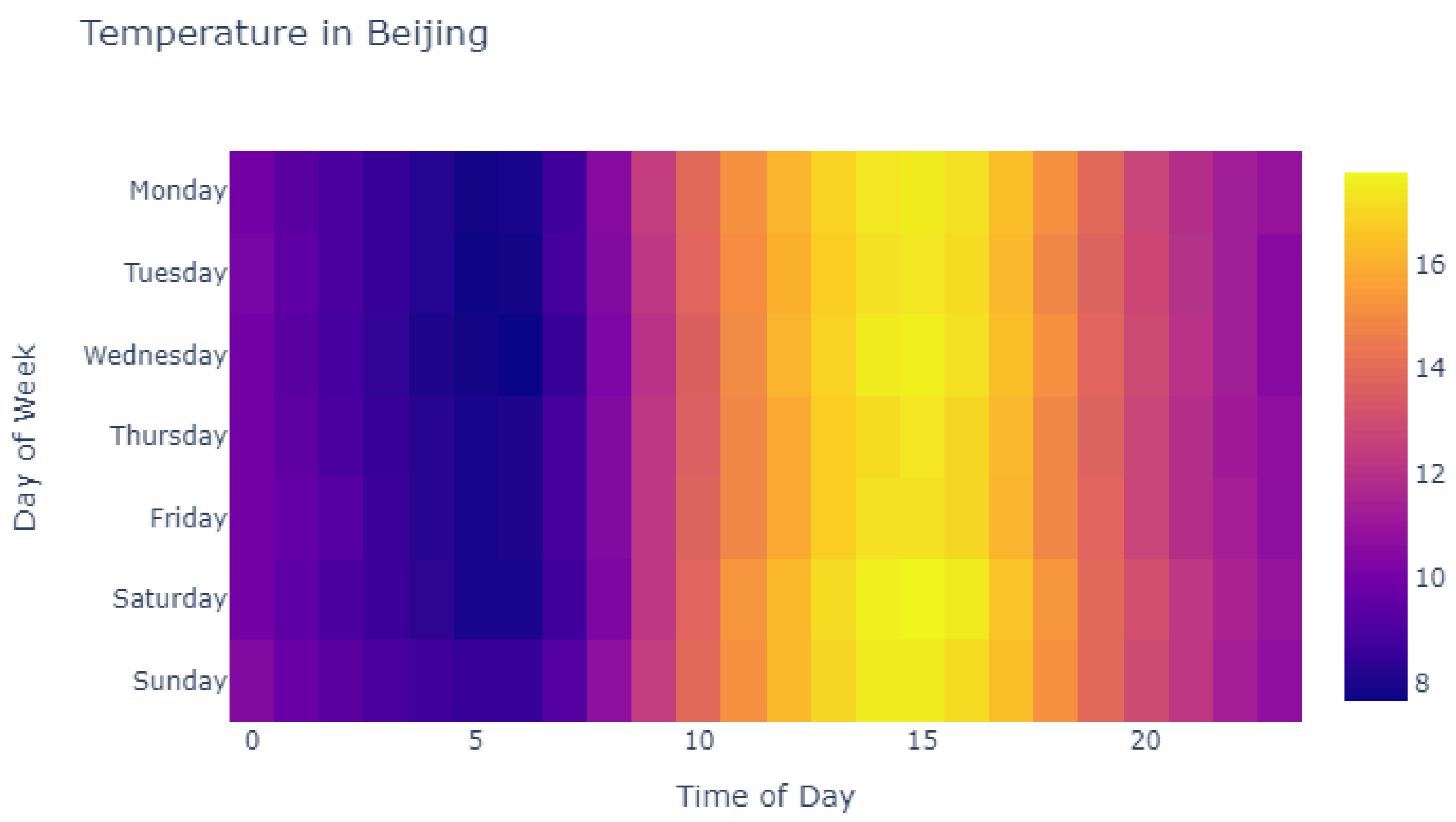

Weight
1500
1000
500
0
Species
Bream
Roach
Whitefish
Parkki
Perch
Pike
Smelt

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'])
```

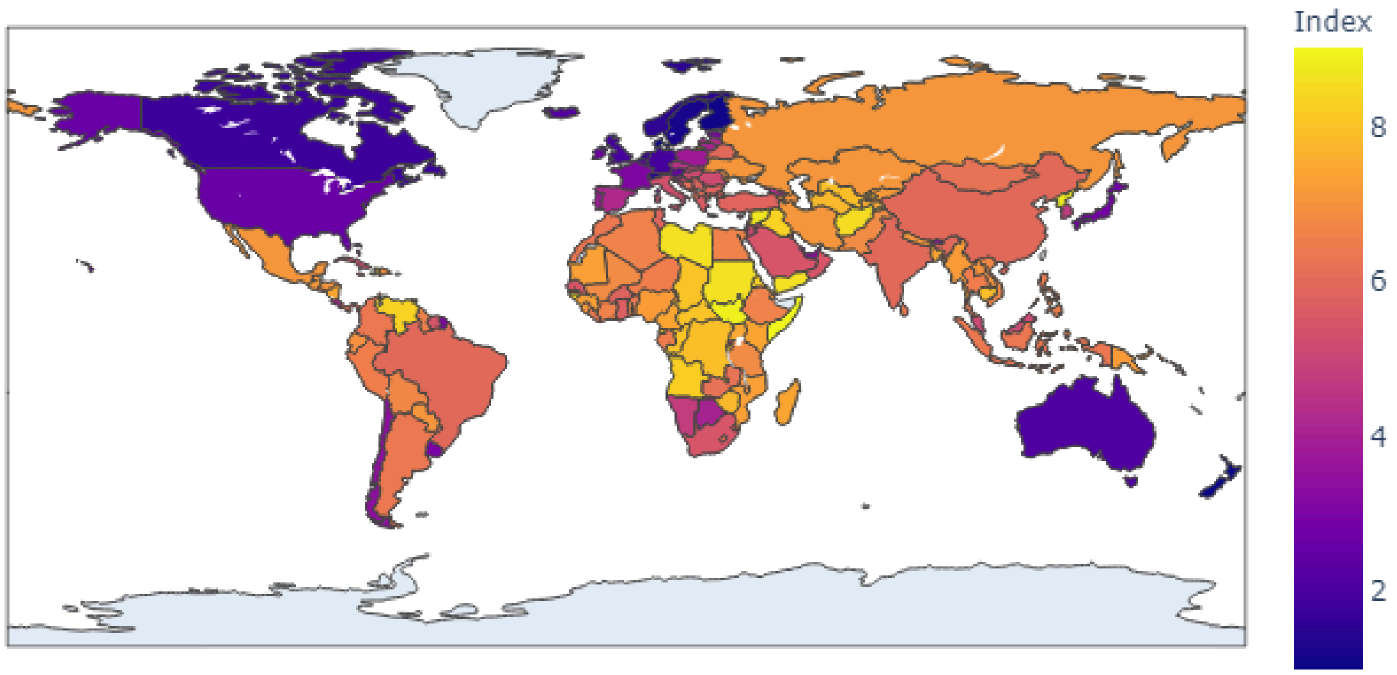

Temperature in Beijing
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday
Day of Week
0
5
10
15
20
Time of Day
16
14
12
10
8

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


Index
8
6
4
2

Mapping Options: Layout->Geo

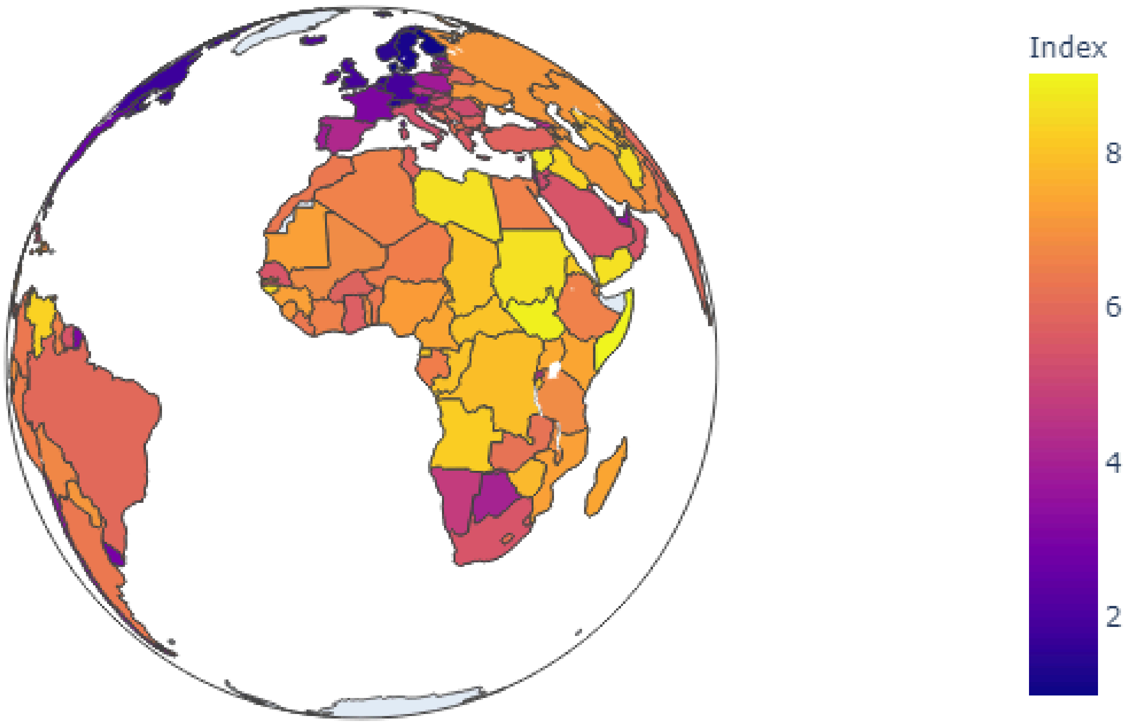
We have many additional options that we can pass to the layout of our plot when dealing with geographic data.

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

Here is a link to the full documentation

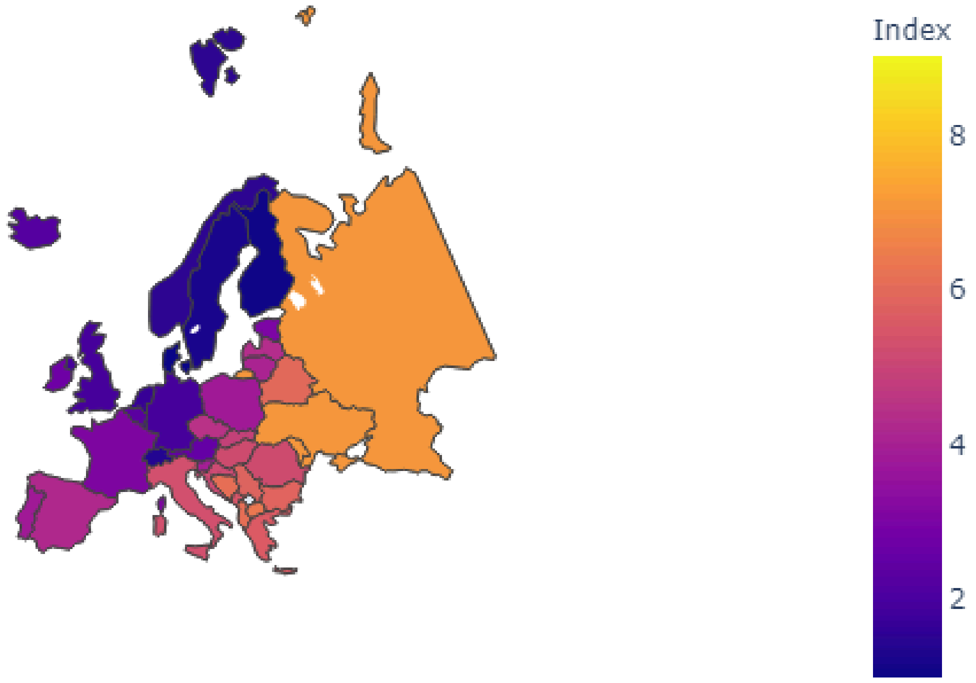
Choropleth Maps - Projection

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


Index
8
6
4
2

Choropleth Maps - Scope

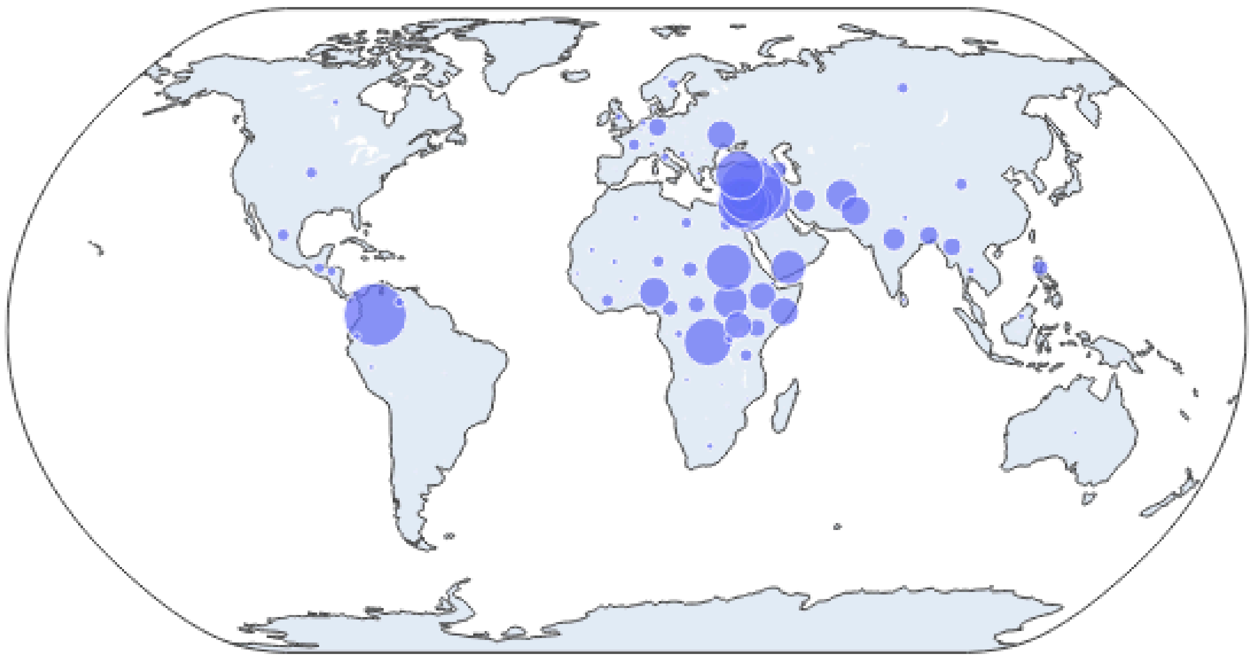
```
px.choropleth(data, locations = 'Abbr',
     color = 'Index',
     hover_name= "Name",
     scope = "europe"
     )
```


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")
```



Alternatives - `lets_plot`

lets-plot offers grammar of graphics plotting in Python

- If you're coming from `ggplot2`, this might be a great alternative for you!
- It still allows for lots of straightforward plotting, and exporting to html for embedding into websites!

Alternatives

- `altair` - vega plotting library
- `bokeh` - like plotly but different
- ⭐ `seaborn` - particularly strong in statistical plots
- `plotnine` - more grammar of graphics
- `matplotlib` - does ALL THE THINGS, but is hard to use
- ⭐ `folium` - for hardcore mapping

Lab Time!