Week 8 - Plotting with the Plotly API

Plotting Options in Python

1. `matplotlib` - The classic library, with options to change just about everything. Creates static images
2. `seaborn` - A stats-focused wrapper for `matplotlib`
3. `pygal` - For SVG-based graphics, interactive plots
4. `plotly` - Based on D3.js, allows for dynamic plotting and data dashboards hosted online
5. `bokeh` - Also based on D3.js functions
6. `ggplot` - uses the Grammar of Graphics structure to mimic R's ggplot2

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)

Getting Started

```
from plotly.offline import plot
import plotly.graph_objs as go
```

First, we want to import `plot`, which is needed to generate plots offline. We don't want to use the Plotly servers at the moment.

Next, we import the graphing objects, which include things like Scatter plots and Histograms, and allow us to construct our visualization.

We will create a new script for each of our visuals for now.

Creating Plot Objects

```
trace = go.Scatter( # initialize scatter object
  x = list(range(2000,2010)), # pass x, y values
  y = [29.8,30.1,30.5,30.6,31.3,31.7,32.6,33.1,32.7,32.8])
  )

data=go.Data([trace]) # Process the plots

plot(data) # Render the plots
```

In this (very) simple example, we plot some time series data. Our plot is provided as an html page opened in the default browser.

Creating Plot Objects

Let's add some formatting. First, we can change the marker size and the color of our plot:

```
trace = go.Scatter( # initialize scatter object
  x = list(range(2000,2010)), # pass x, y values
  y = [29.8,30.1,30.5,30.6,31.3,31.7,32.6,33.1,32.7,32.8],
  marker =  {'color': 'green', # choose the marker color
    'symbol': 0, # choose a shape
    'size': "20"}) # choose a size

data=go.Data([trace]) # Process the plots

plot(data) # Render the plots
```

Your plot should refresh in the browser.

Creating Plot Objects

Next, we can smooth our line between markers:

```
trace = go.Scatter( # initialize scatter object
  x = list(range(2000,2010)), # pass x, y values
  y = [29.8,30.1,30.5,30.6,31.3,31.7,32.6,33.1,32.7,32.8],
  marker =  {'color': 'green', # choose the marker color
    'symbol': 0, # choose a shape
    'size': "20"}, # choose a size
    line=dict(
        shape='spline' # spline smoothing
    ))

data=go.Data([trace]) # Process the plot(s)

plot(data) # Render the plot(s)
```

Creating Plot Objects

We can add text to our markers:

```
trace = go.Scatter( # initialize scatter object
  x = list(range(2000,2010)), # pass x, y values
  y = [29.8,30.1,30.5,30.6,31.3,31.7,32.6,33.1,32.7,32.8],
  marker =  {'color': 'green', # choose the marker color
    'symbol': 0, # choose a shape
    'size': "20"}, # choose a size
    line=dict(
        shape='spline' # spline smoothing
    ),
    text=['Year: ' + str(i) for i in
        list(range(2000,2010))], # hover text
    name='PCC') # name for legends)

data=go.Data([trace]) # Process the plots

plot(data) # Render the plots
```

Creating Plot Objects

We can add information to our plot by adding `Layout` and `Figure` objects:

```
data=go.Data([trace]) # Process the plots

layout=go.Layout(title="Per Capita Cheese Consumption",
                 # configure the plot
  xaxis={'title':'Year'},  # layout and name
  yaxis={'title':'Pounds of Cheese'})  # the axes.

figure=go.Figure(data=data,layout=layout)
# combine data and layout code

plot(figure) # Render the plots
```

Creating Plot Objects

Let's add a second series:

```
trace2 = go.Scatter( # initialize scatter object
  x = list(range(2000,2010)), # pass x, y values
  y = [327,456,509,497,596,573,661,741,809,717],
  marker =  {'color': 'grey', # choose the marker color
    'symbol': 0, # choose a shape
    'size': "20"}, # choose a size
    line=dict(
        shape='spline' # spline smoothing
    ),
    text=['Year: ' + str(i) for i in
        list(range(2000,2010))], # hover text
    name='DIB',
    yaxis='y2') # name for legends
```

Creating Plot Objects

We also need to update our data and layout objects:

```
data=go.Data([trace, trace2]) # Process the plots

layout=go.Layout(title="Per Capita Cheese Consumption",
                 # configure the plot
  xaxis={'title':'Year',
         'showgrid':False}, # hide the gridlines
  yaxis={'title':'Pounds of Cheese',
         'showgrid':False},

figure=go.Figure(data=data,layout=layout)
# combine data and layout code

plot(figure) # Render the plots
```

Creating Plot Objects

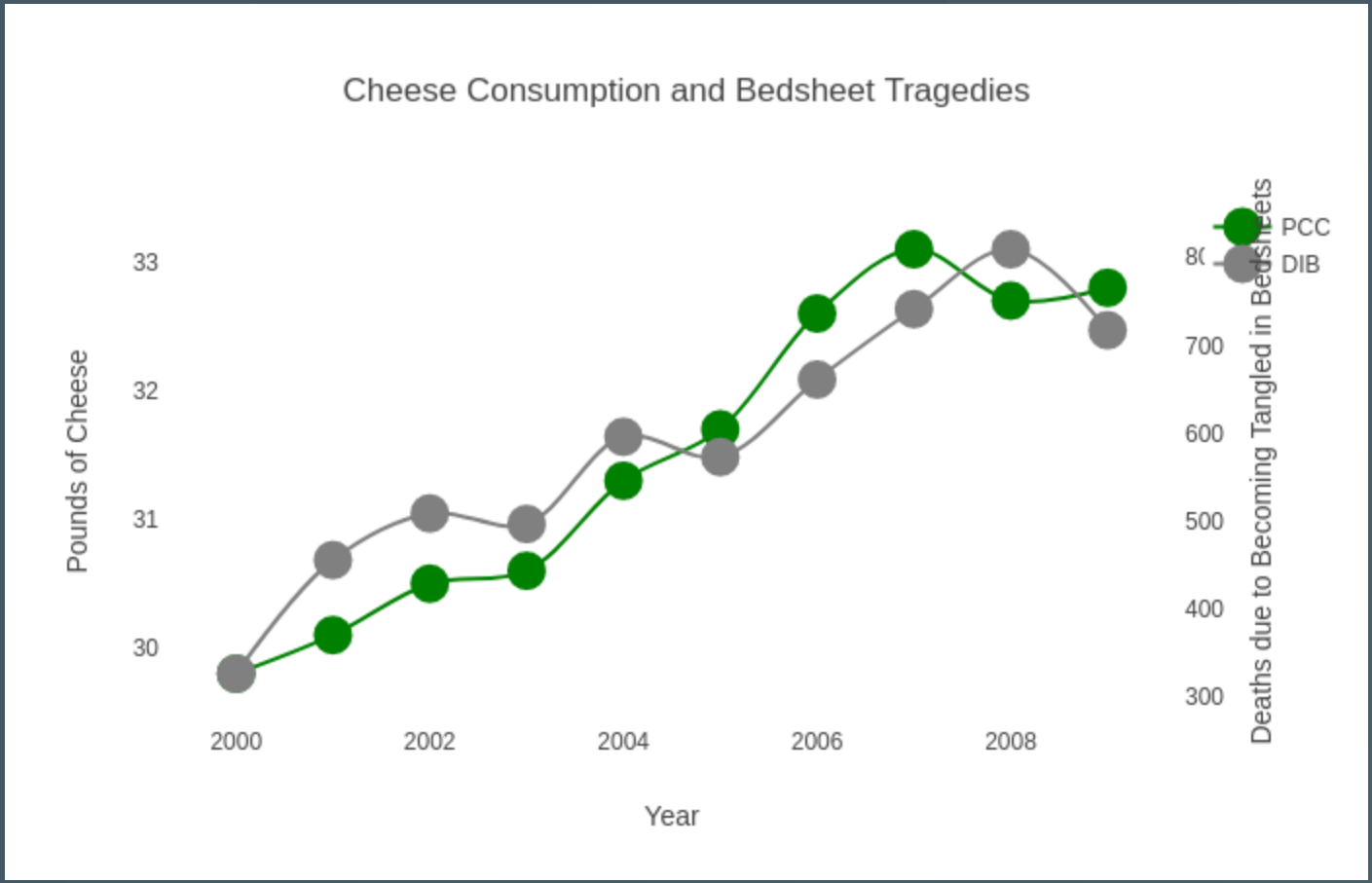
AND we should add a secondary y axis:

```
data=go.Data([trace, trace2]) # Process the plots

layout=go.Layout(title=
        "Cheese Consumption and Bedsheet Tragedies",
                  # configure the plot
  xaxis={'title':'Year',
         'showgrid':False}, # hide the gridlines
  yaxis={'title':'Pounds of Cheese',
         'showgrid':False},
  yaxis2={'title': # add secondary axis
          "Deaths due to Becoming Tangled in Bedsheets",
          'overlaying': 'y', # it is a y, not x, axis
          'side':'right', # show values on right side
          'showgrid':False})  # hide gridlines
```

Creating Plot Objects

Our plot now looks something like this:



Using Existing Data

Let's import average household income data for Nebraska using the ACS data at dadata.cba.edu:

```
from sqlalchemy import create_engine
import pandas as pd

SELECT = """SELECT AVG(hhincome) AS hhincome, year
  FROM ACS
  WHERE statefip=31
  GROUP BY year
  ORDER BY year"""
conn = create_engine(
  'mysql+mysqlconnector://viewer:@dadata.cba.edu:3306/ACS'
  )

data = pd.read_sql(SELECT, conn)
```

Using Existing Data

```
trace = go.Scatter( # initialize scatter object
  x = data['year'],
  y = data['hhincome'],
  marker =  {'color': 'green',
    'symbol': 0,
    'size': "12"},
  mode="markers+lines",
  name='Household Income Over Time')

data=go.Data([trace])
layout=go.Layout(title="Household Income",
  xaxis={'title':'Year'},
  yaxis={'title':'Income ($)'})

figure=go.Figure(data=data,layout=layout)
plot(figure)
```

Let's Use More States!

First, update our dataset:

```
SELECT = """SELECT AVG(hhincome) AS hhincome, year,
    statefip
  FROM ACS
  WHERE statefip=31 or statefip=19
  GROUP BY year, statefip
  ORDER BY year, statefip"""
conn = create_engine(
  'mysql+mysqlconnector://viewer:@dadata.cba.edu:3306/ACS'
  )

data = pd.read_sql(SELECT, conn)
```

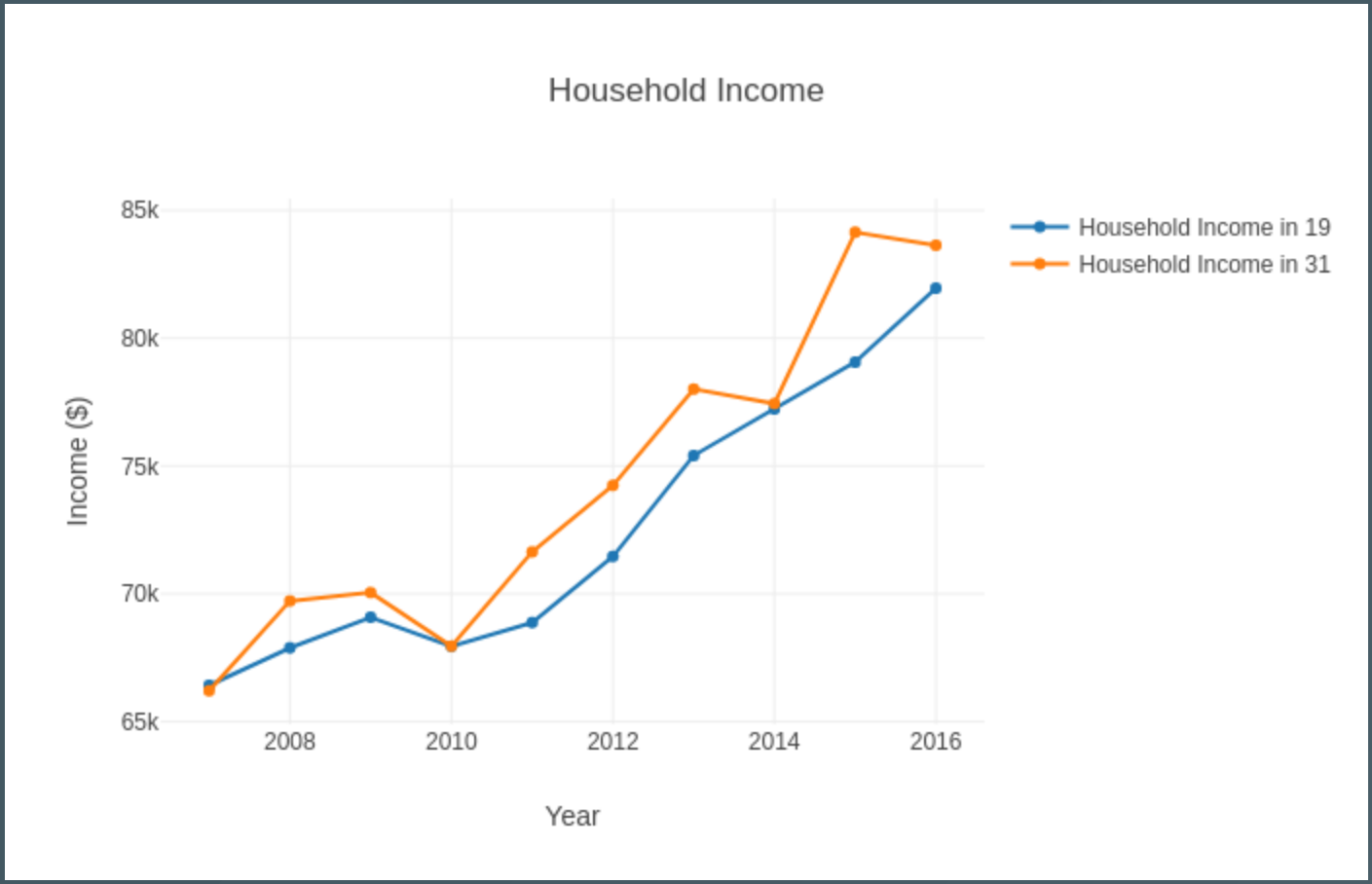
Let's Use More States!

```
traces = []
for i in data['statefip'].unique():
  small_data = data.loc[data['statefip']==i, :]
  traces.append(go.Scatter( # initialize scatter object
    x = small_data['year'],
    y = small_data['hhincome'],
    mode="markers+lines",
    name='Household Income in {}'.format(i)))

data=go.Data(traces)
layout=go.Layout(title="Household Income",
  xaxis={'title':'Year'},
  yaxis={'title':'Income ($)'})

figure=go.Figure(data=data,layout=layout)
plot(figure)
```

Let's Use More States!



Other Plot Types

We can do a LOT more than scatter plots!

- Box Plots
- Histograms, with distribution stats
- Heatmaps
- Choropleth, Line, and Bubble Maps

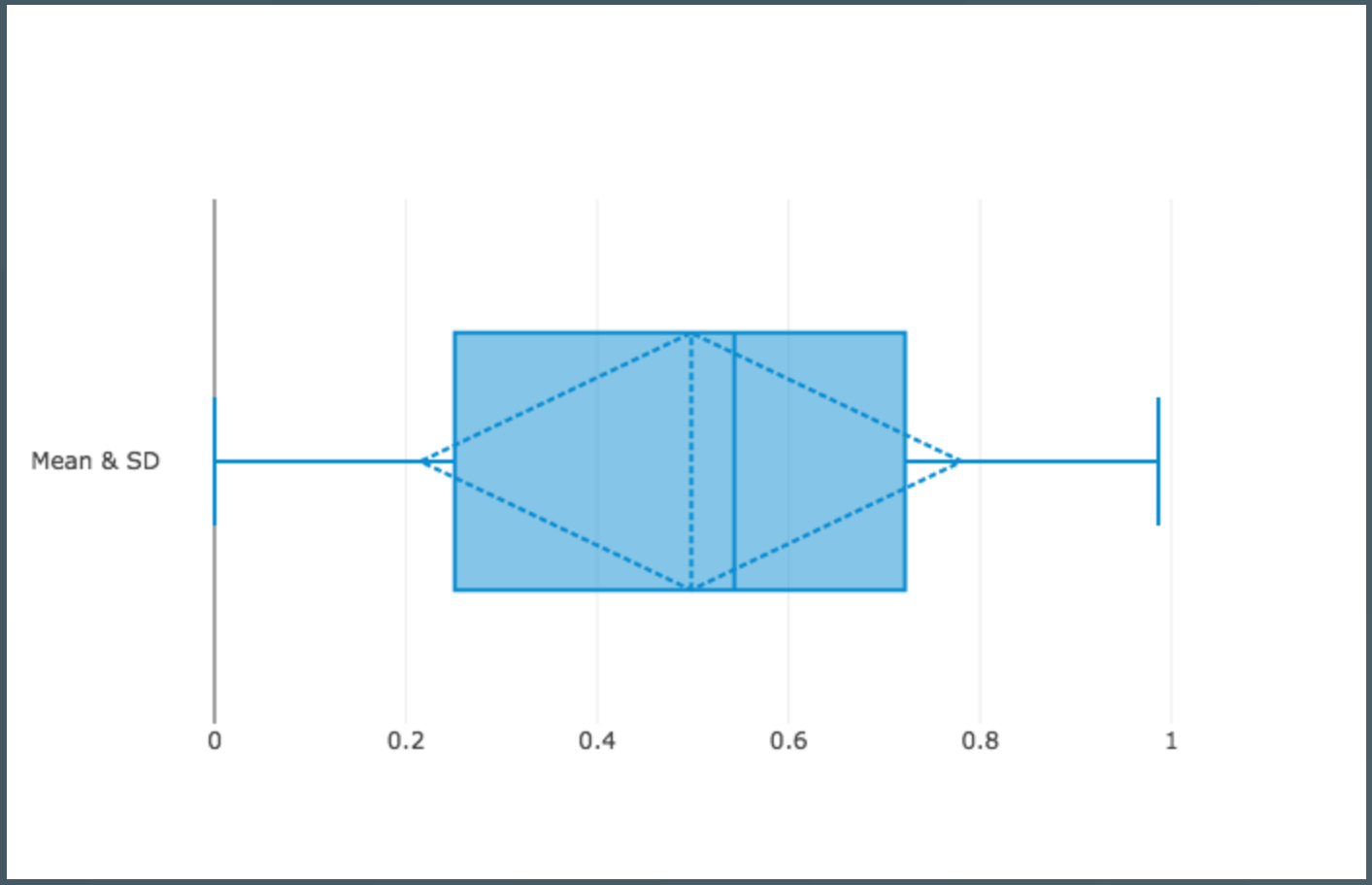
among many others.

Box Plots

```
trace1 = go.Box(
  y=np.random.rand(100),
  name='Mean & SD',
  marker=dict(
    color='rgb(10, 140, 208)',
  ),
  boxmean='sd' # Shows quartiles AND Std Dev on plot
)

data = go.Data([trace1])
figure = go.Figure(data=data)
plot(figure)
```

Box Plots

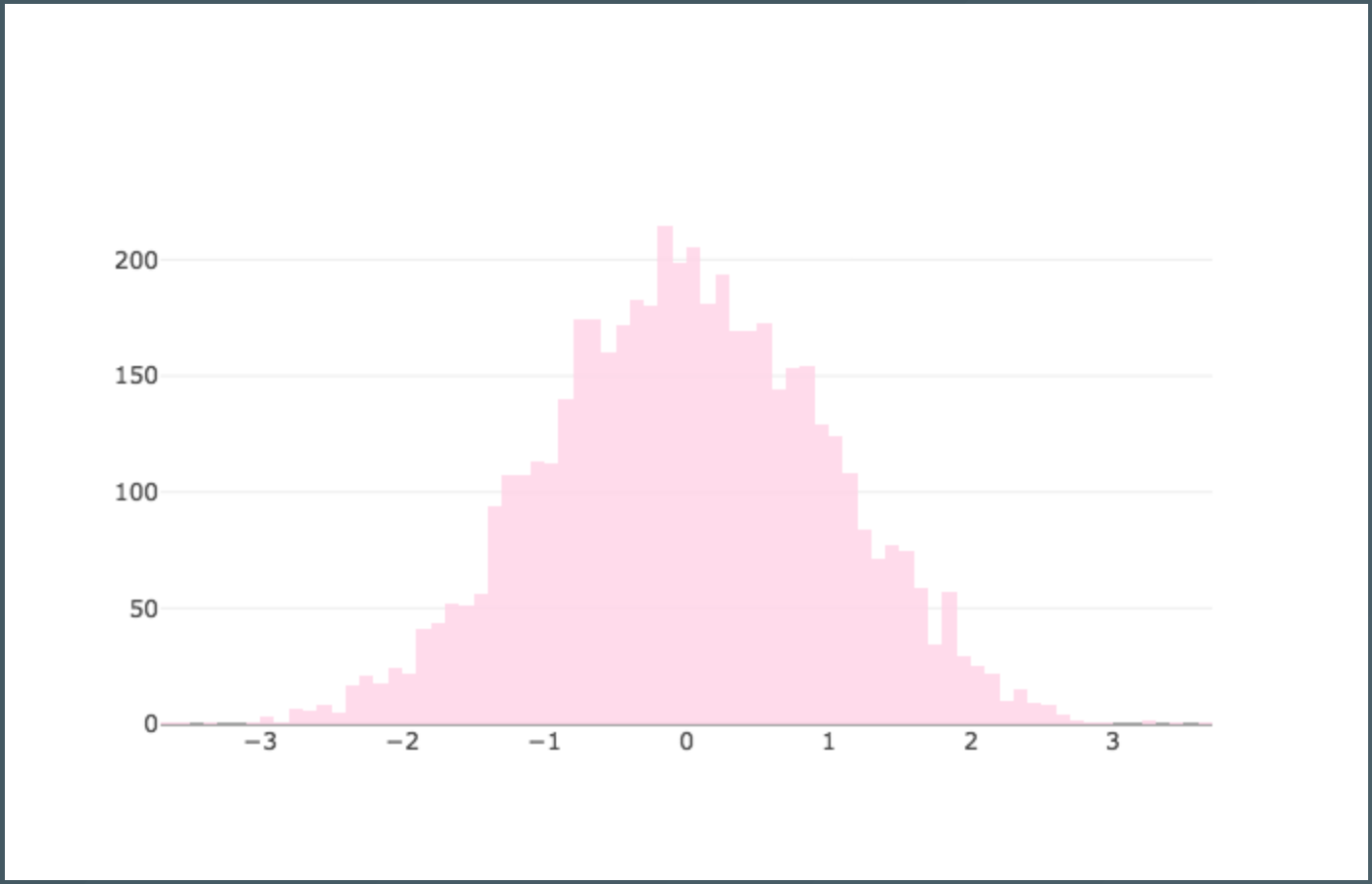


Histograms

```
trace1 = go.Histogram(
  x=np.random.randn(5000),
  histnorm='count',
  xbins=dict( # Declare bin size
    start=-4.0,
    end=4.0,
    size=0.1
  ),
  marker=dict( # Customize markers
    color='#FFD7E9',
  ),
  opacity=0.9
)

data = go.Data([trace1])
figure = go.Figure(data=data)
plot(figure)
```

Histograms



Histograms and Distributions

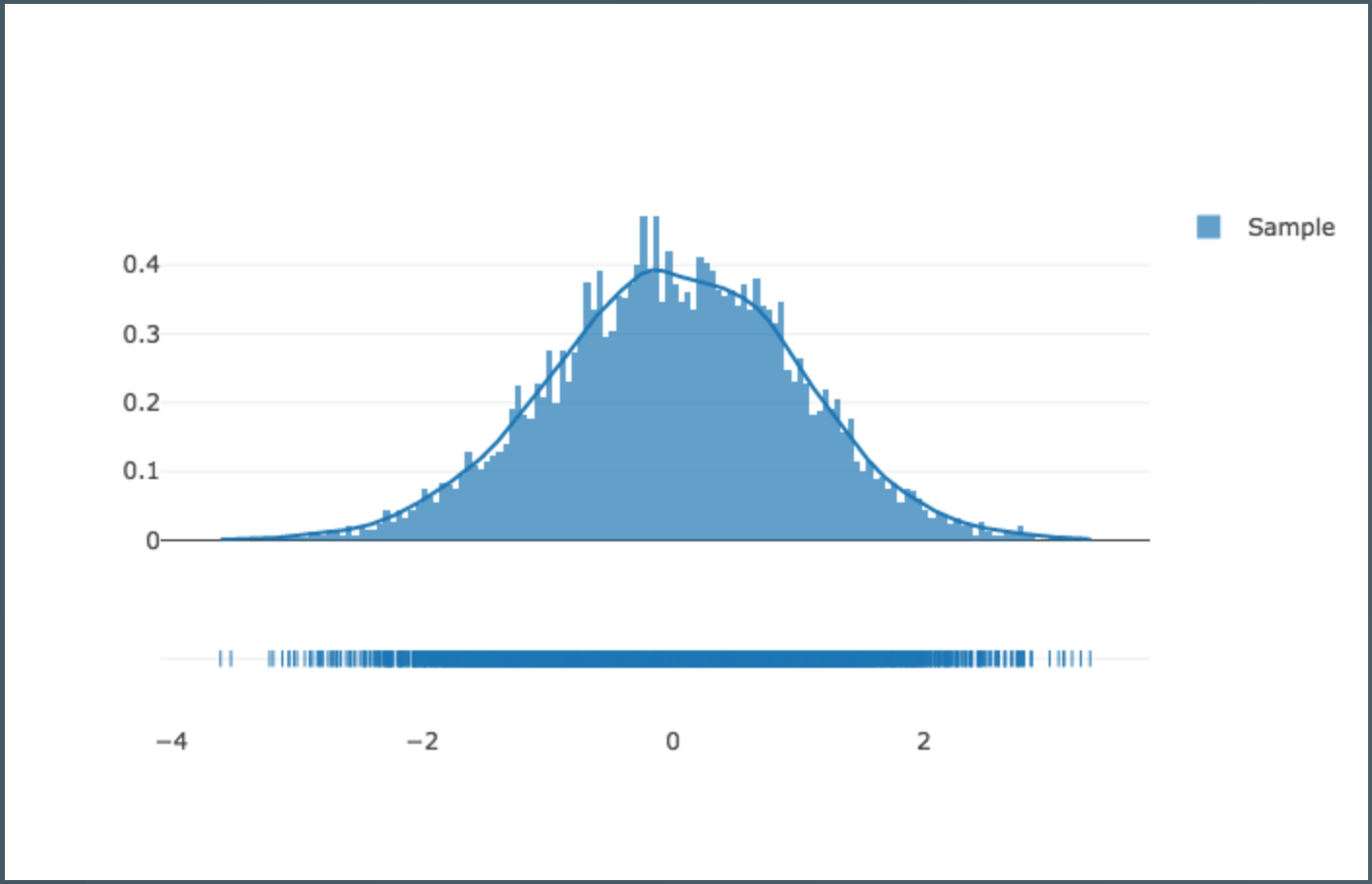
```
import plotly.figure_factory as ff

x = np.random.randn(5000)
hist_data = [x]
group_labels = ['Sample'] # Labels our 'rug' plot

fig = ff.create_distplot(hist_data,
  group_labels,
  bin_size=0.05,
  show_hist=True,  # Toggle histogram
  show_curve=True, # Toggle smoothed distribution
  show_rug=True    # Toggle rug plot
  )

plot(fig)
```

Histograms and Distributions

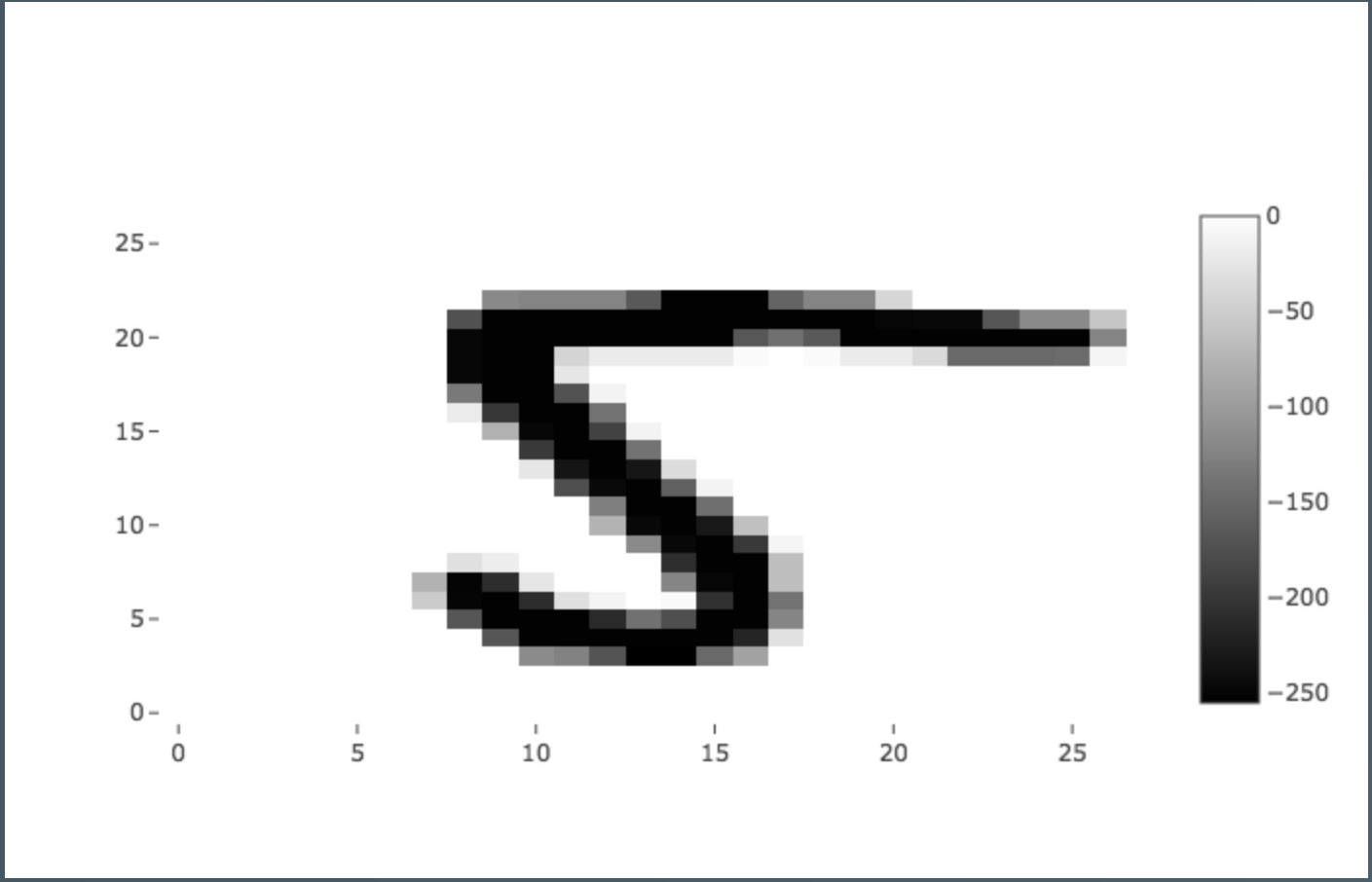


Heatmaps

```
# Create an array of data (its from MNIST)
x = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
x = - np.array(x).reshape(28,28)


trace = go.Heatmap(z = x, colorscale = "Greys")
data=[trace]
plot(data)
```

Heatmaps



Choropleth Maps

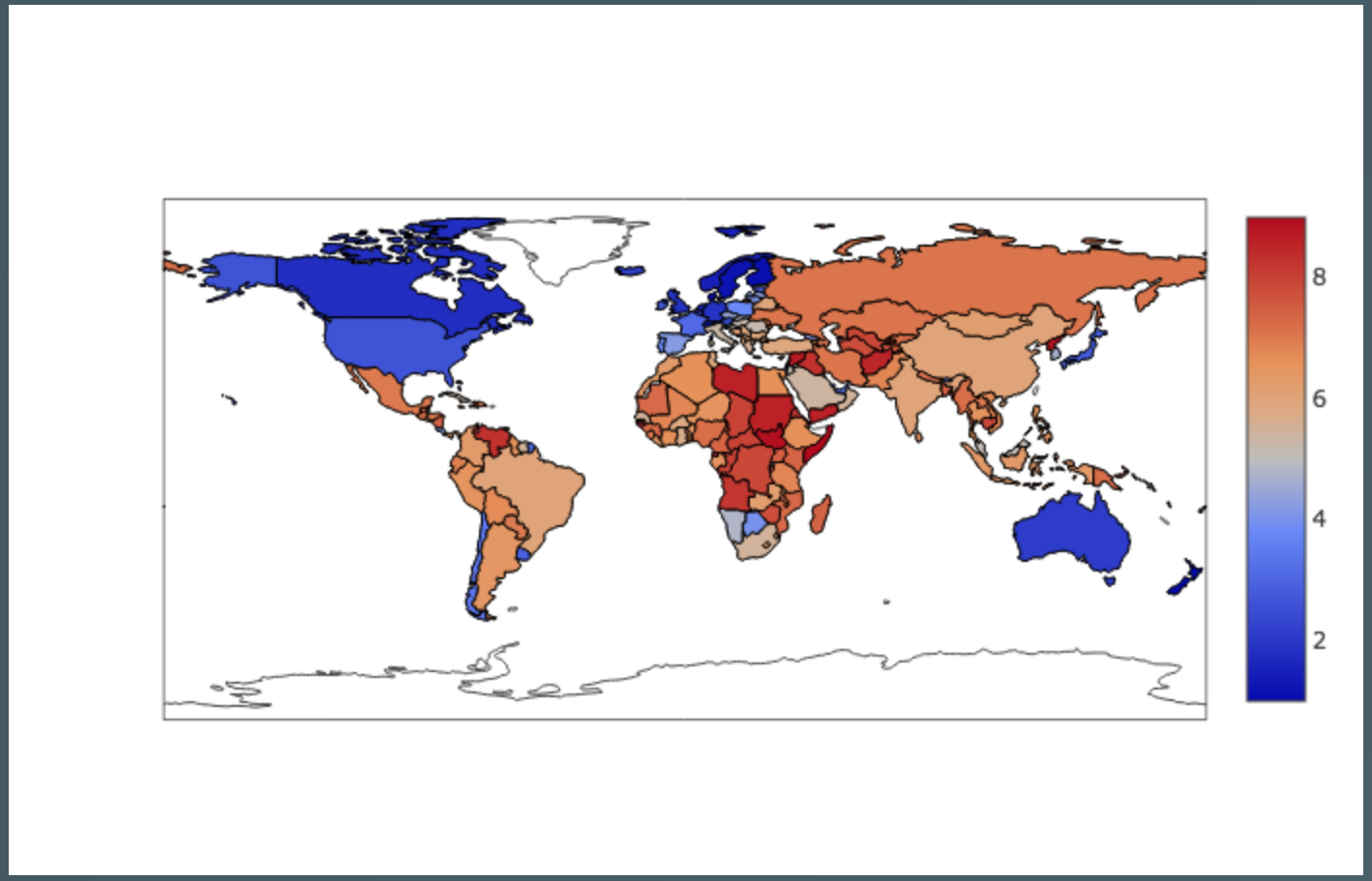
```
data = pd.read_csv("corruption2018.csv")

pdata = go.Choropleth(
        locations = data['Abbr'],
        z = data['Index'],
        text = (data['Name'], data['Index']),
        autocolorscale = False,
        colorscale = 'Virginica',
        showscale = True,
    )


figure = go.Figure(data=[pdata])
plot(figure)
```

Map data from the INFORM Index

Choropleth Maps



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

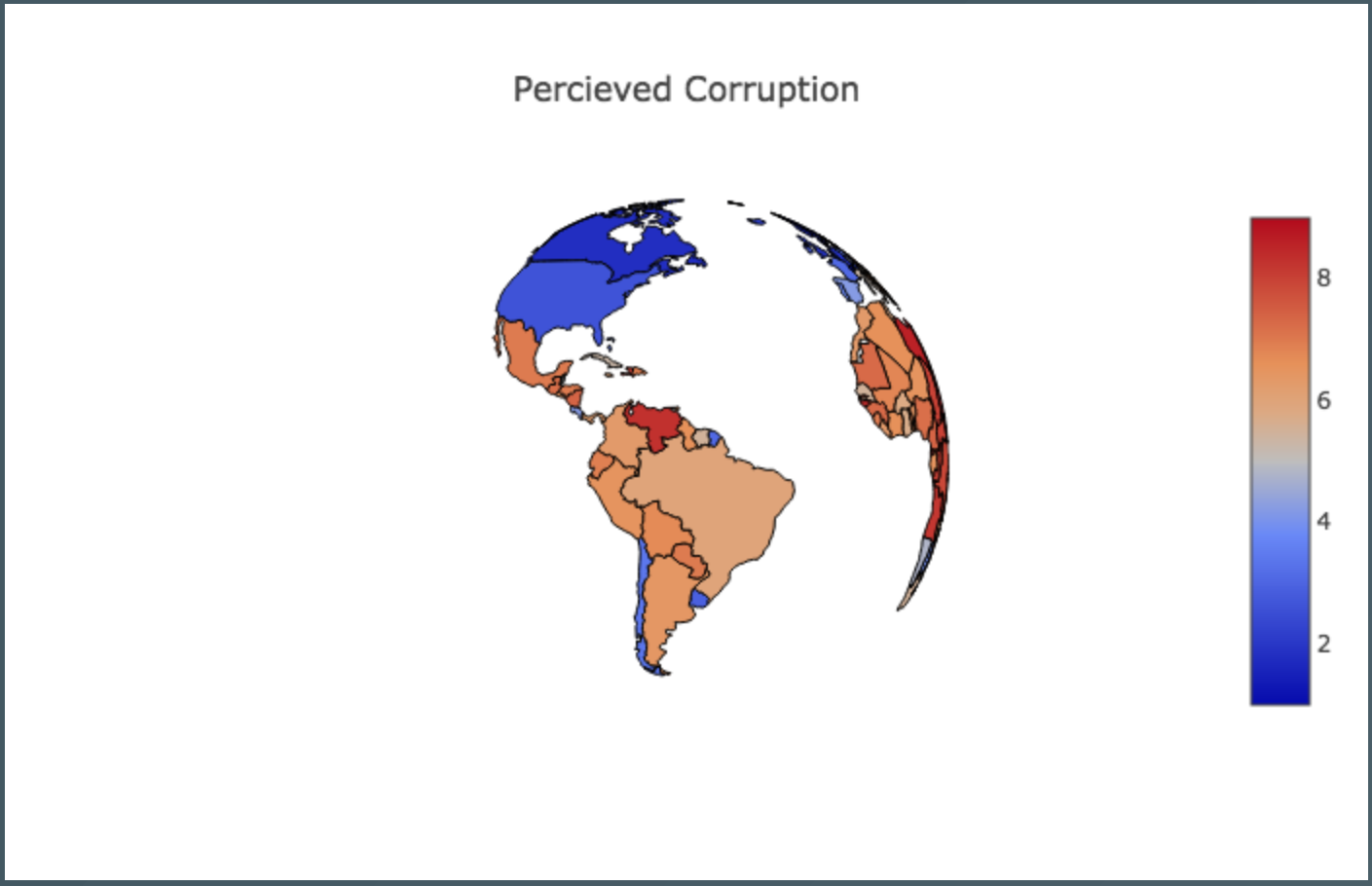
Here is a link to the full documentation

Choropleth Maps - Layout

```
layout = go.Layout(
    title = "Percieved Corruption",
    geo = dict(projection = dict(type='orthographic'),
               showcoastlines=False,
               showcountries = False,
               showframe = False,
               showrivers=False,
               scope = 'all'
        )
    )

figure = go.Figure(data=[pdata], layout=layout)
plot(figure)
```

Choropleth Maps



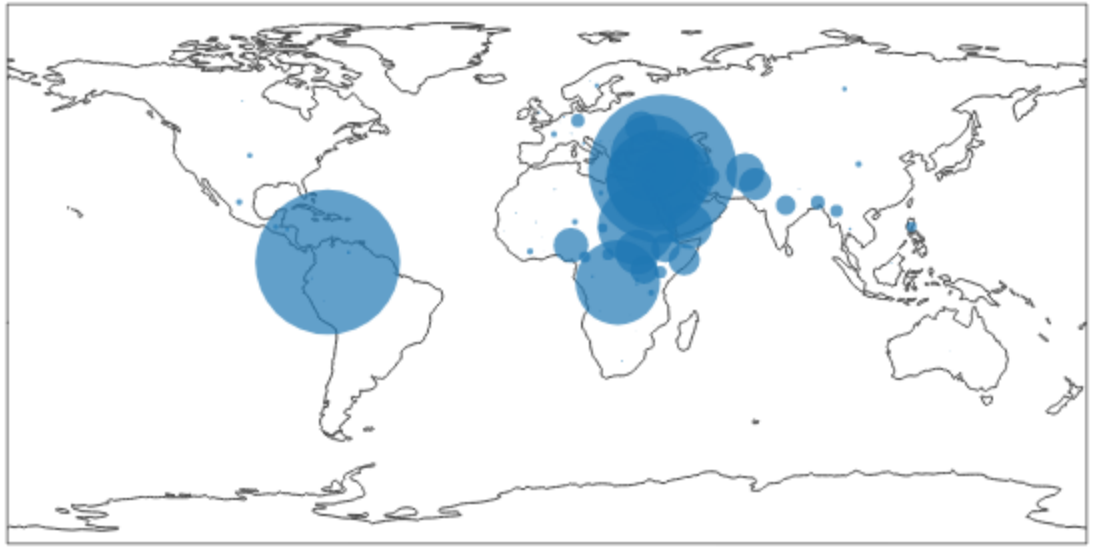
Bubble Maps

```
data = pd.read_csv("displaced2018.csv")

pdata = go.Scattergeo(
            locationmode = 'country names',
            locations = data['Name'],
            marker = dict(
                size = data['Displaced']/100000,
                line = dict(width = 0)
            ),
            text = data['Displaced']
            )

figure = go.Figure(data=[pdata])
plot(figure)
```

Bubble Maps

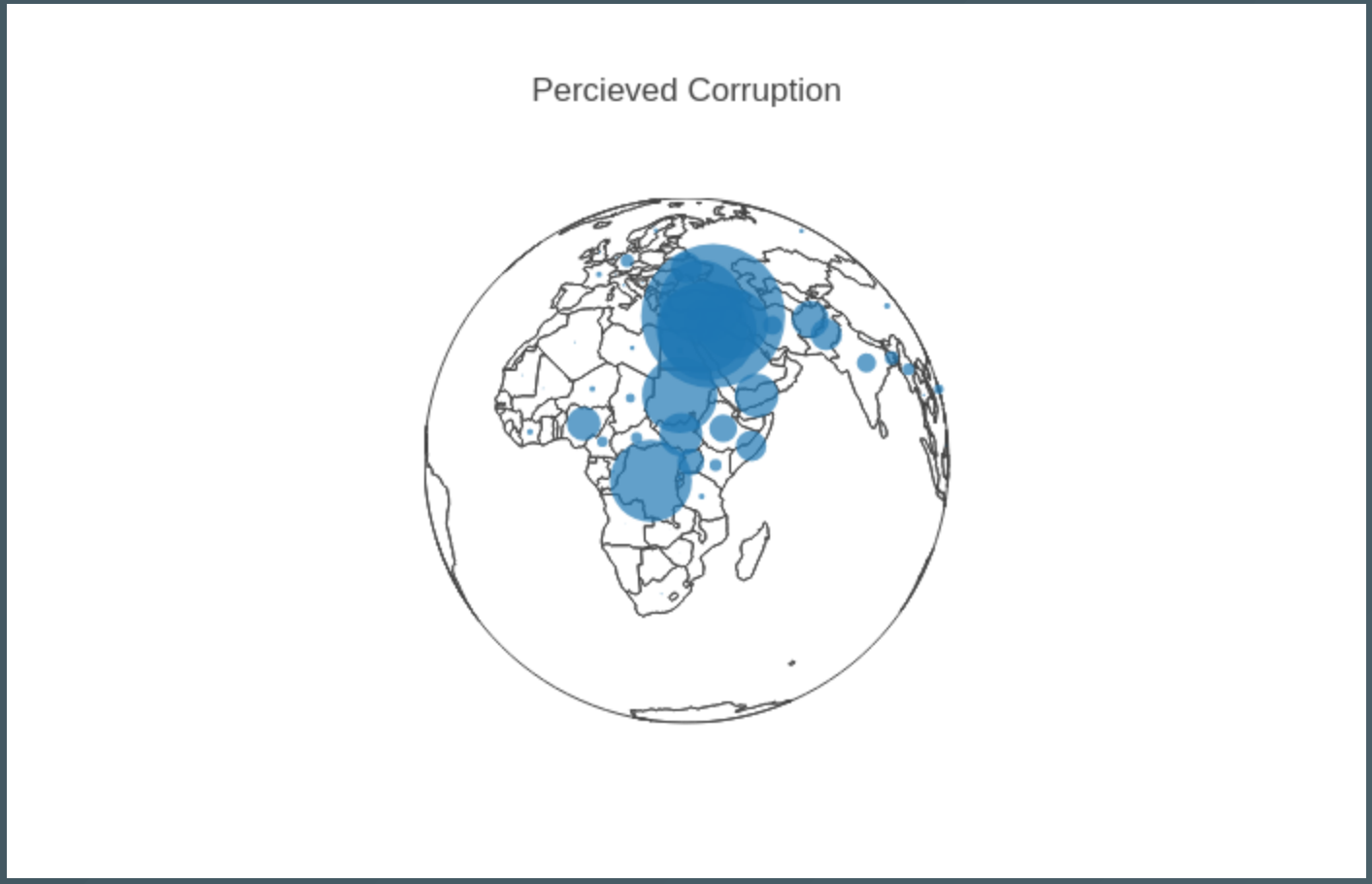


Bubble Maps

```
...
layout = go.Layout(
    title = "Percieved Corruption",
    geo = dict(projection = dict(type='orthographic'),
               showcountries = True,
        )
    )

figure = go.Figure(data=[pdata], layout=layout)
plot(figure)
```

Bubble Maps



For Lab Tonight

Let's make use of this week's and last week's data together!

- Draw data from either the ACS database or the NFL database on dadata.cba.edu (you choose the data)
- Generate five plots that you find interesting.
- Use at least three different kinds of plots