CSE 4510/5310
BIG DATA

Instructor: Fitzroy Nembhard, Ph.D.

Week 5b

Visualization:
Matplotlib, SYMPY, Seaborn, and Plotly

ADDENDUM

FLORIDA TECH
FLORIDA'S STEM UNIVERSITY®

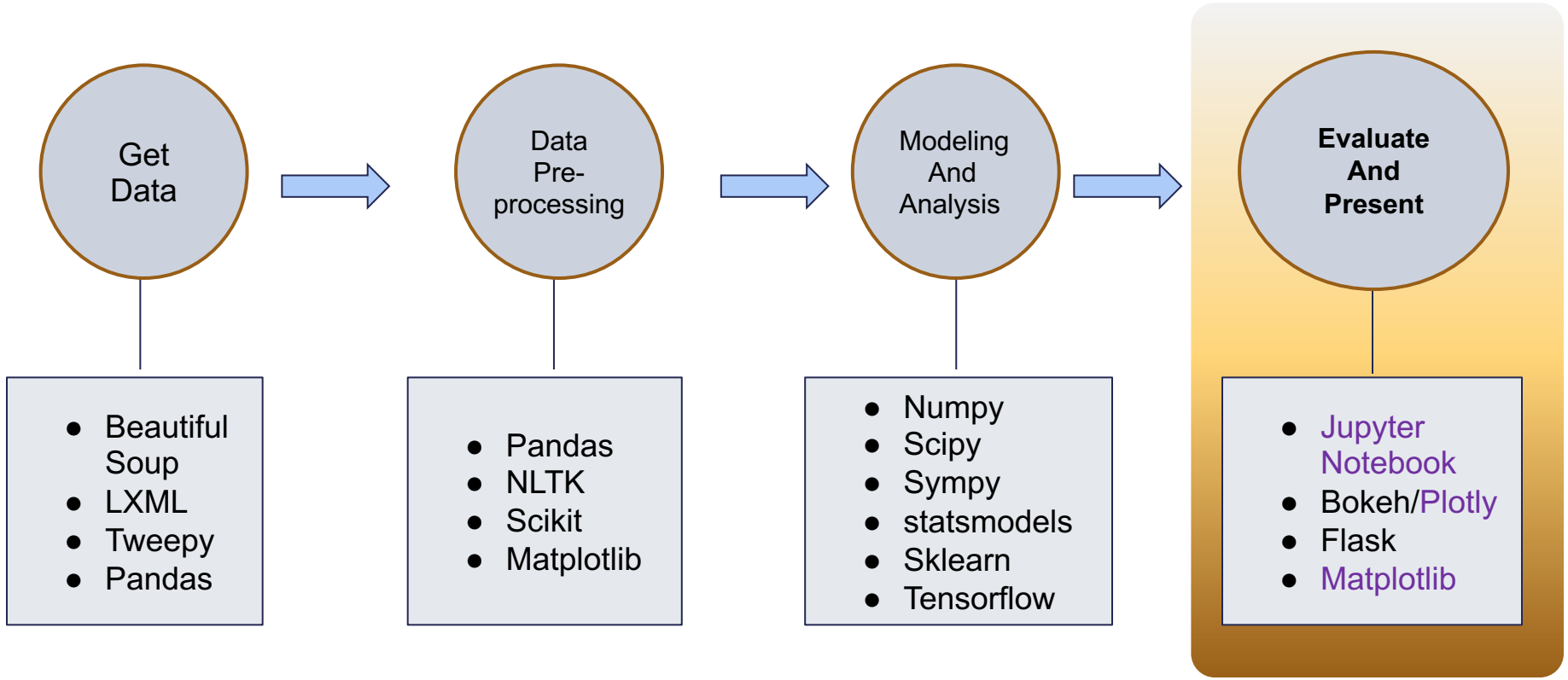
Distribution

- All slides included in this class are for the exclusive use of students and instructors associated with Database Systems (CSE 4020/5260) at the Florida Institute of Technology
- Redistribution of the slides is not permitted without the written consent of the author.

Goals

- To discuss best practices for creating visualizations
- To create plots using Matplotlib
- To create plots using Seaborn
- To plot mathematical equations using Sympy
- To briefly explore Plotly by creating a Choropleth Map

Where are we in the Data Science Process?



Best practices for creating visualizations

5B.1 VISUALIZATION

Take Home Lesson

- "You have the power and responsibility to produce meaningful and interpretable presentations of your work. Effective visualization involves an iterative process of looking at the data, deciding what story it is trying to tell, and then improving the display to tell the story better."

  -- The Data Science Manual, Steven Skiena

- **Before going deeper into Matplotlib and Seaborn, I will discuss a few tips on best practices for visualizations**

The Importance of Data Visualization in BI

Data visualization helps a business to achieve numerous goals

- Converting business data into interactive graphs for dynamic interpretation to serve the business goals
- Transforming data into visually appealing, interactive dashboards of various data sources to provide insights
- Creating more attractive and informative dashboards of various graphical data representations
- Making appropriate decisions by drilling into the data and finding insights
- Figuring out the patterns, trends, and correlations in the data being analyzed to determine where they must improve operational processes and thereby grow their business
- Giving a fuller picture of the data under analysis
- Organizing and presenting massive data intuitively to present important findings from the data
- Making better, faster, and informed decisions with data visualization

Deciding on a Plot

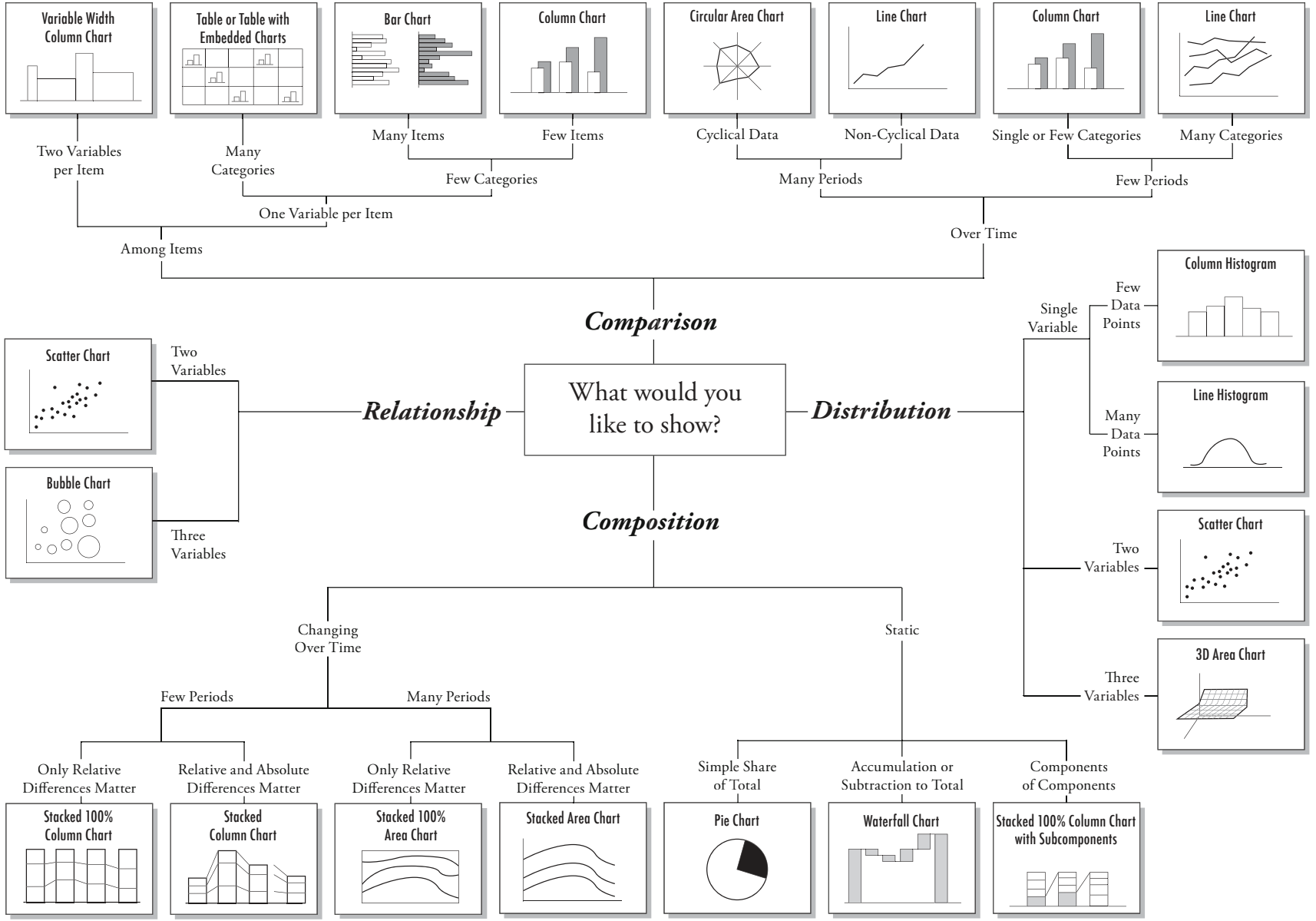


Chart Types - Summary

Single variable

- Dot plot
- Jitter plot
- Error bar plot
- Box-and-whisker plot
- Histogram
  - Proper selection of bin width is important
  - Outliers should be discarded
- Kernel density estimate
- Cumulative distribution function

Two variables

- Bar chart
  - One variable is discrete
- Scatter plot
- Line plot
- Log-log plot
  - Very useful for power law data

More than two variables

- Stacked plots
  - Stack variable is discrete
- Parallel coordinate plot
  - One discrete variable, an arbitrary number of other variables

Best Practices for Presenting Tabular Data

- Right justify numbers

left	center	right
1	1	1
10	10	10
100	100	100
1000	1000	1000

- Order rows to invite comparison
- Order columns to highlight importance
- Choose a standard precision like 2DP
- Avoid excessive-length column descriptors
- Use emphasis, font, or color to highlight important entries

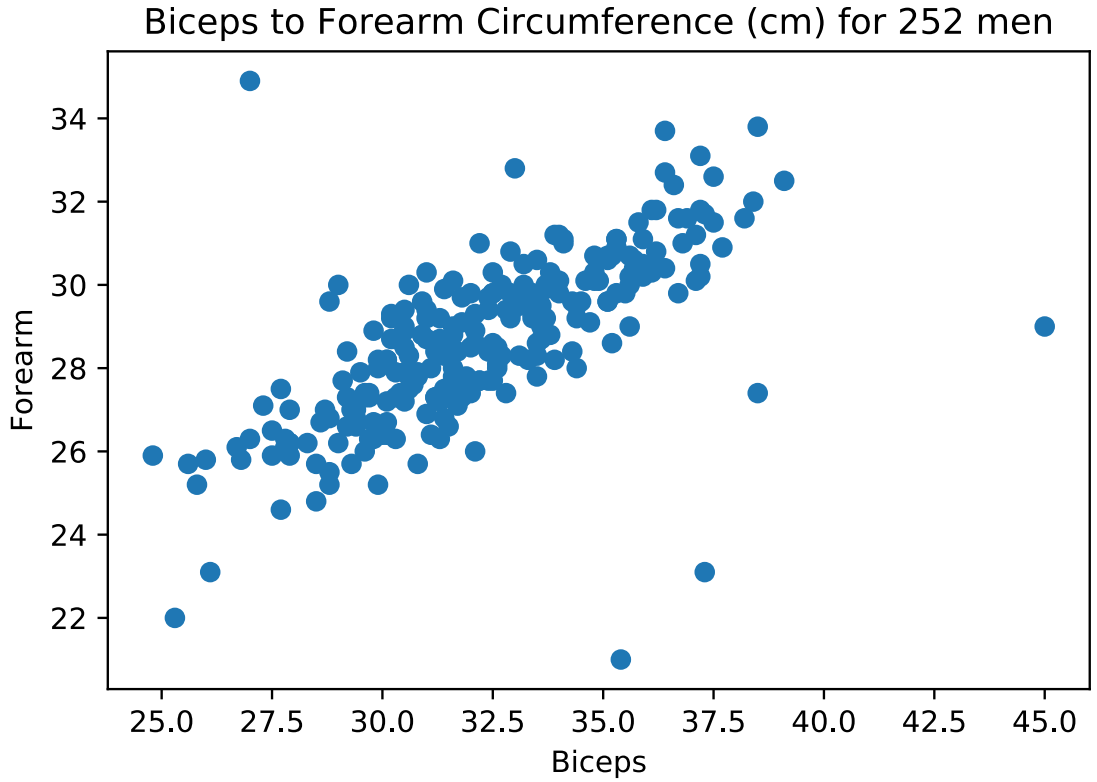
Country	Area	Density	Birthrate	Population	Mortality	GDP
Russia	17075200	8.37	99.6	142893540	15.39	8900.0
Mexico	1972550	54.47	92.2	107449525	20.91	9000.0
Japan	377835	337.35	99.0	127463611	3.26	28200.0
United Kingdom	244820	247.57	99.0	60609153	5.16	27700.0
New Zealand	268680	15.17	99.0	4076140	5.85	21600.0
Afghanistan	647500	47.96	36.0	31056997	163.07	700.0
Israel	20770	305.83	95.4	6352117	7.03	19800.0
United States	9631420	30.99	97.0	298444215	6.5	37800.0
China	9596960	136.92	90.9	1313973713	24.18	5000.0
Tajikistan	143100	51.16	99.4	7320815	110.76	1000.0
Burma	678500	69.83	85.3	47382633	67.24	1800.0
Tanzania	945087	39.62	78.2	37445392	98.54	600.0
Tonga	748	153.33	98.5	114689	12.62	2200.0
Germany	357021	230.86	99.0	82422299	4.16	27600.0
Australia	7686850	2.64	100.0	20264082	4.69	29000.0

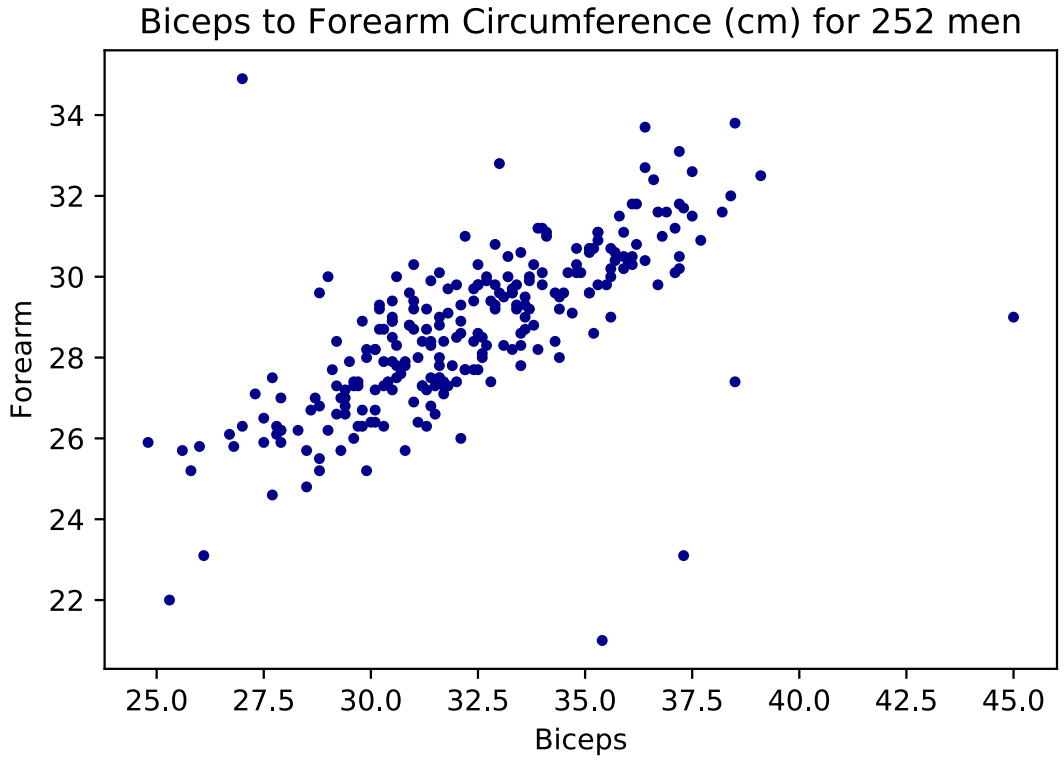
Country	Population	Area	Density	Mortality	GDP	Birth Rate
Afghanistan	31,056,997	647,500	47.96	**163.07**	700	36.0
Australia	20,264,082	7,686,850	2.64	4.69	29,000	**100.0**
Burma	47,382,633	678,500	69.83	67.24	1,800	85.3
China	**1,313,973,713**	9,596,960	136.92	24.18	5,000	90.9
Germany	82,422,299	357,021	230.86	4.16	27,600	99.0
Israel	6,352,117	20,770	305.83	7.03	19,800	95.4
Japan	127,463,611	377,835	**337.35**	3.26	28,200	99.0
Mexico	107,449,525	1,972,550	54.47	20.91	9,000	92.2
New Zealand	4,076,140	268,680	15.17	5.85	21,600	99.0
Russia	142,893,540	**17,075,200**	8.37	15.39	8,900	99.6
Tajikistan	7,320,815	143,100	51.16	110.76	1,000	99.4
Tanzania	37,445,392	945,087	39.62	98.54	600	78.2
Tonga	114,689	748	153.33	12.62	2,200	98.5
United Kingdom	60,609,153	244,820	247.57	5.16	27,700	99.0
United States	298,444,215	9,631,420	30.99	6.50	**37,800**	97.0

The Data Science Manual, Steven Skiena

Best Practices for Scatter Plots

- Smaller dots on scatter plots (right) reveal more detail than the default dot size (left).

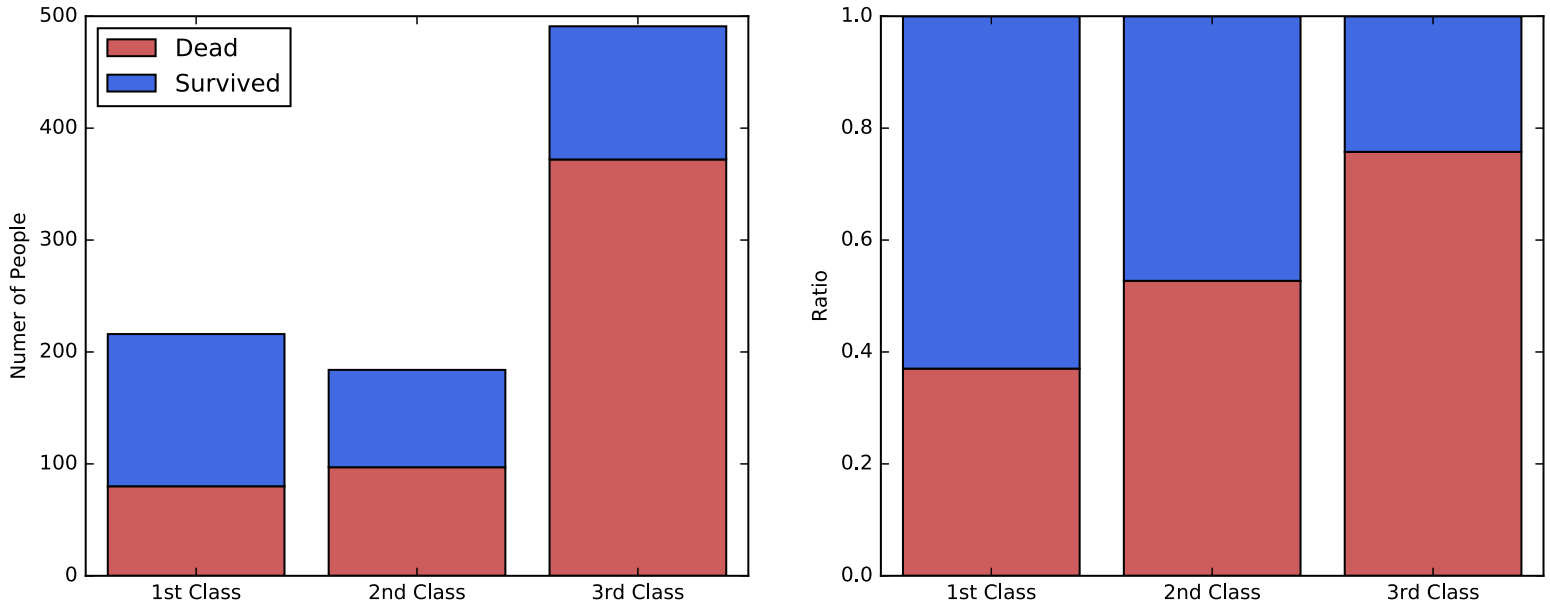




Dataset source: http://lib.stat.cmu.edu/datasets/bodyfat

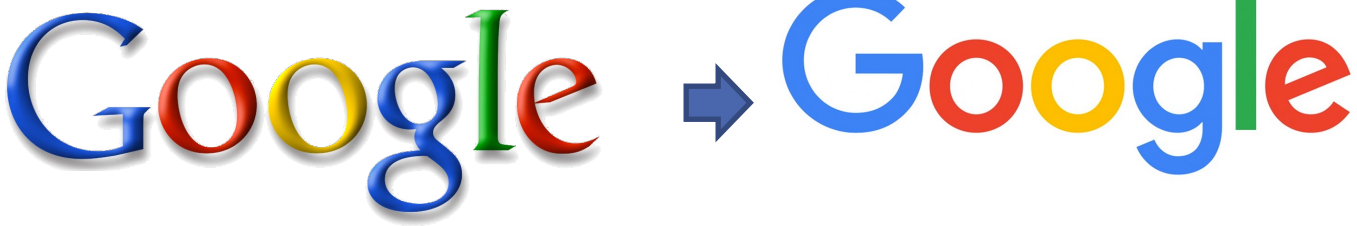
Best Practices Stacked Plots

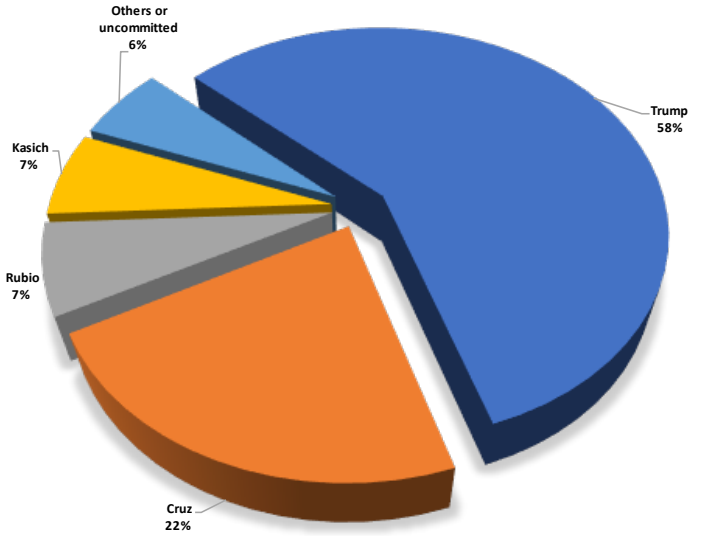
- The stacked plot (left) informs us of the size of each class, but scaled bars (right) better capture proportions

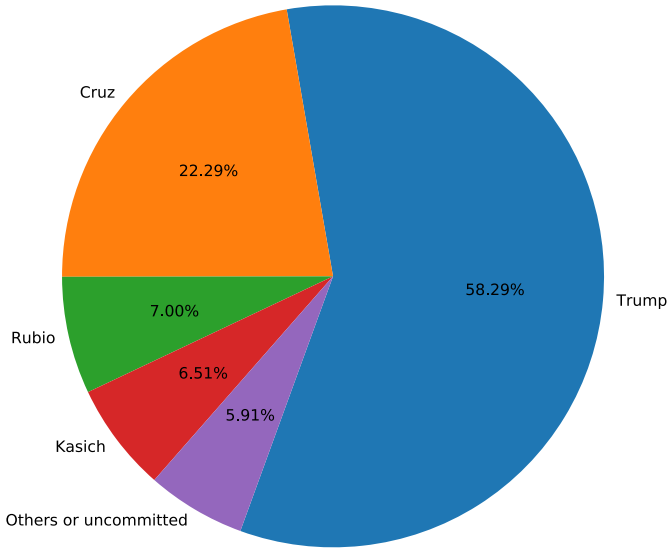


Best Practices for Pie Charts

- In the chart to the right, the 3D effects and separation/explosion are pure ***chartjunk*** that only obscures the relationship between the size of the slices.
- The lower pie is two-dimensional, but is better at conveying the distribution of votes
- But why do we need a pie chart at all? A little table of labels/colors with an extra column with percentages would be more concise and informative
- Simplicity is better.

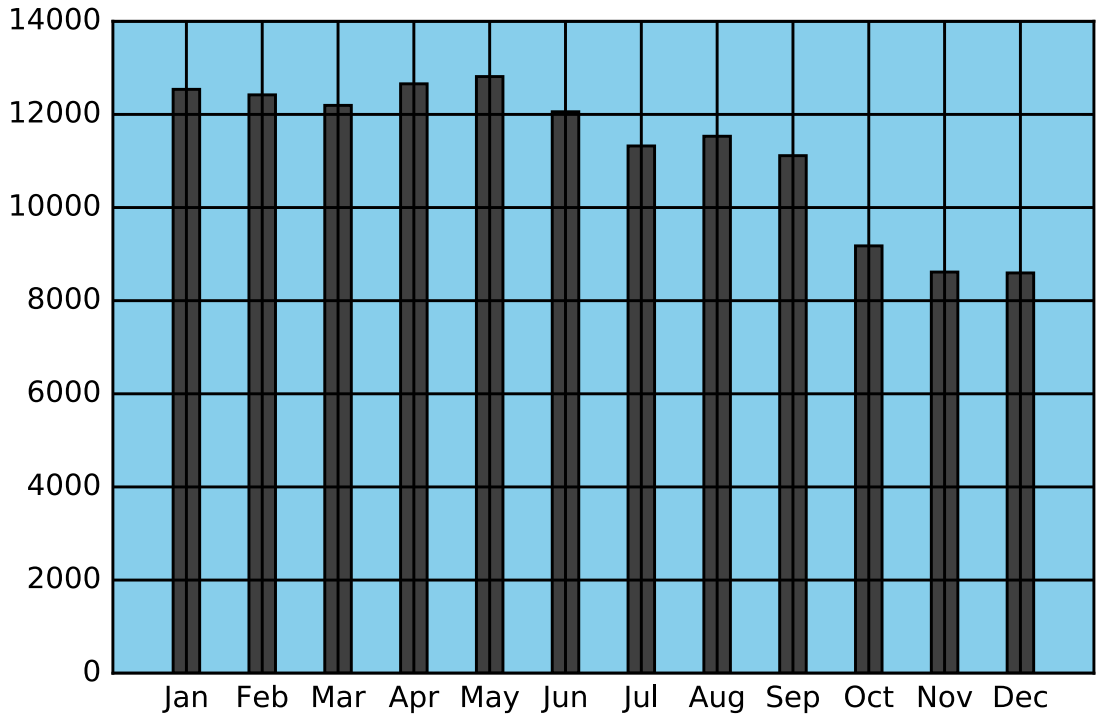






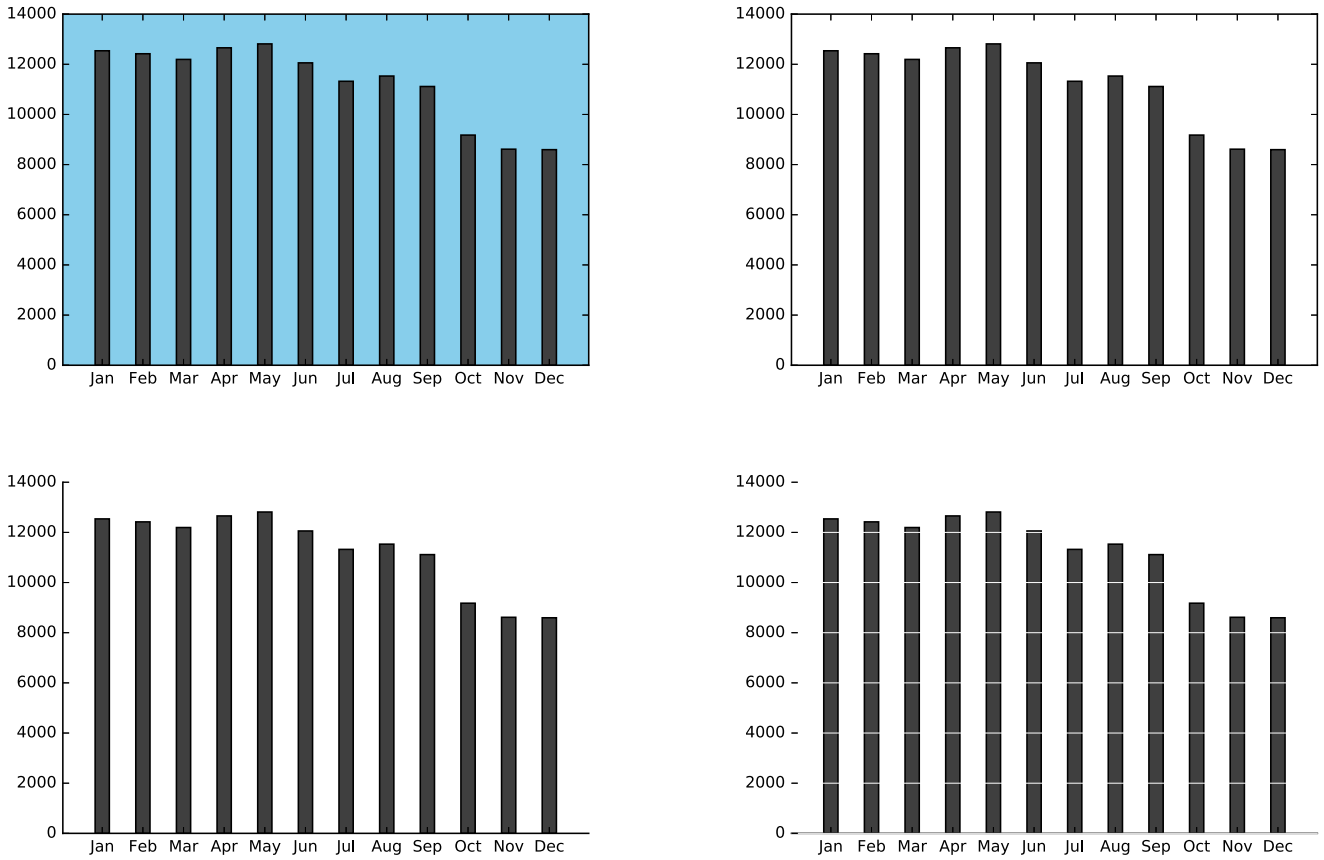
Improving Plots

- A plot showing monthly time series of sales.
- How can we improve/simplify this bar chart showing time series data?



Improving Plots (Cont'd)

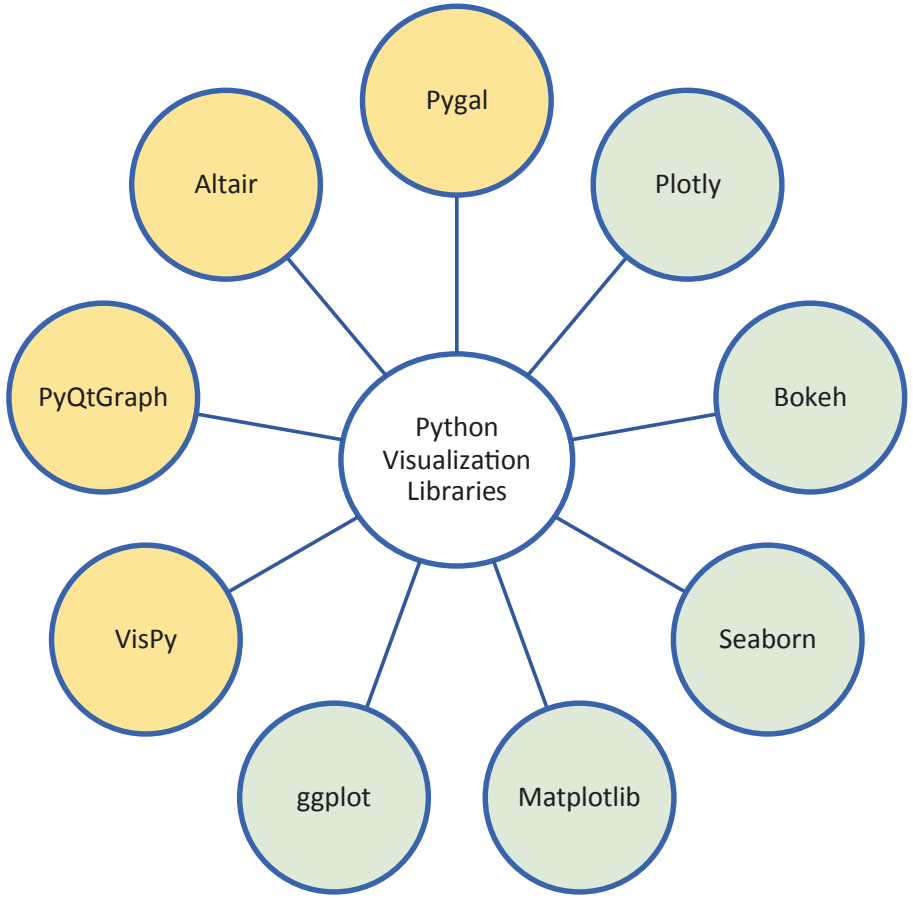
- The resulting chart after performing four successive simplifications by removing extraneous non-data elements



Improving Plots (Cont'd)

- Follow the data-ink ratio rule

Popular Python Visualization Libraries



5B.2 MATPLOTLIB

Simple Plots

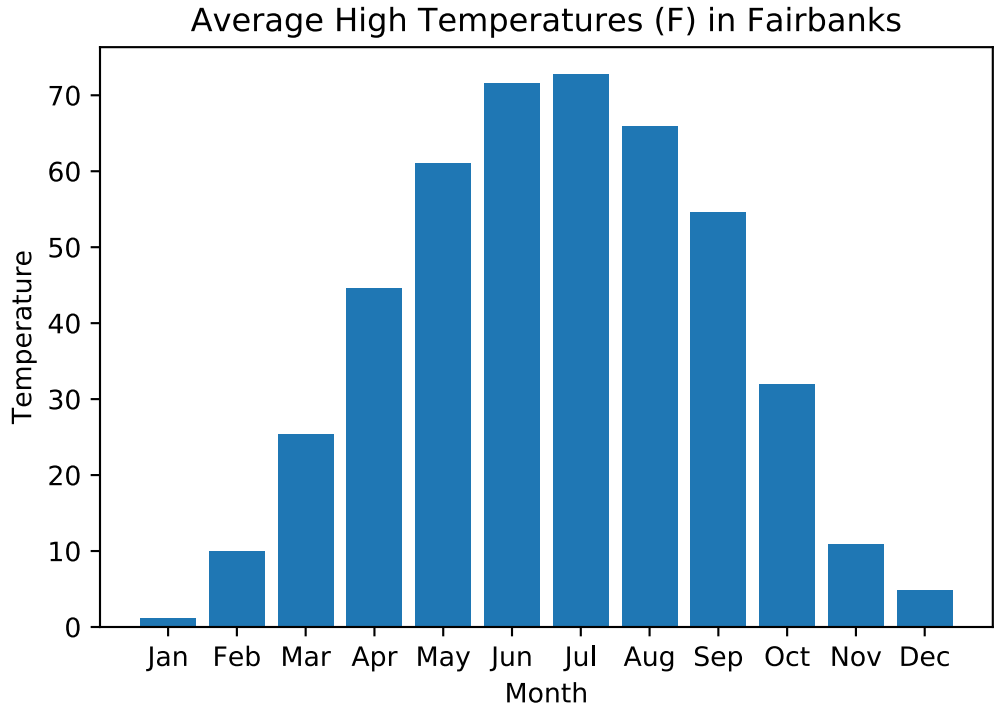
- **Decision: Should I accept a data scientist job in Fairbanks, Alaska?**
- You obtained the following temperatures
- You can scan through the numbers and try to make sense of them, or you can create a graph to help you see patterns that may not be so obvious from the raw numbers.

Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1.1	10	25.4	44.5	61	71.6	72.7	65.9	54.6	31.9	10.9	4.8

Simple Plots (Cont'd)

- **Decision: Should I accept a data scientist job in Fairbanks, Alaska?**
- Let's create a simple bar-chart using matplotlib

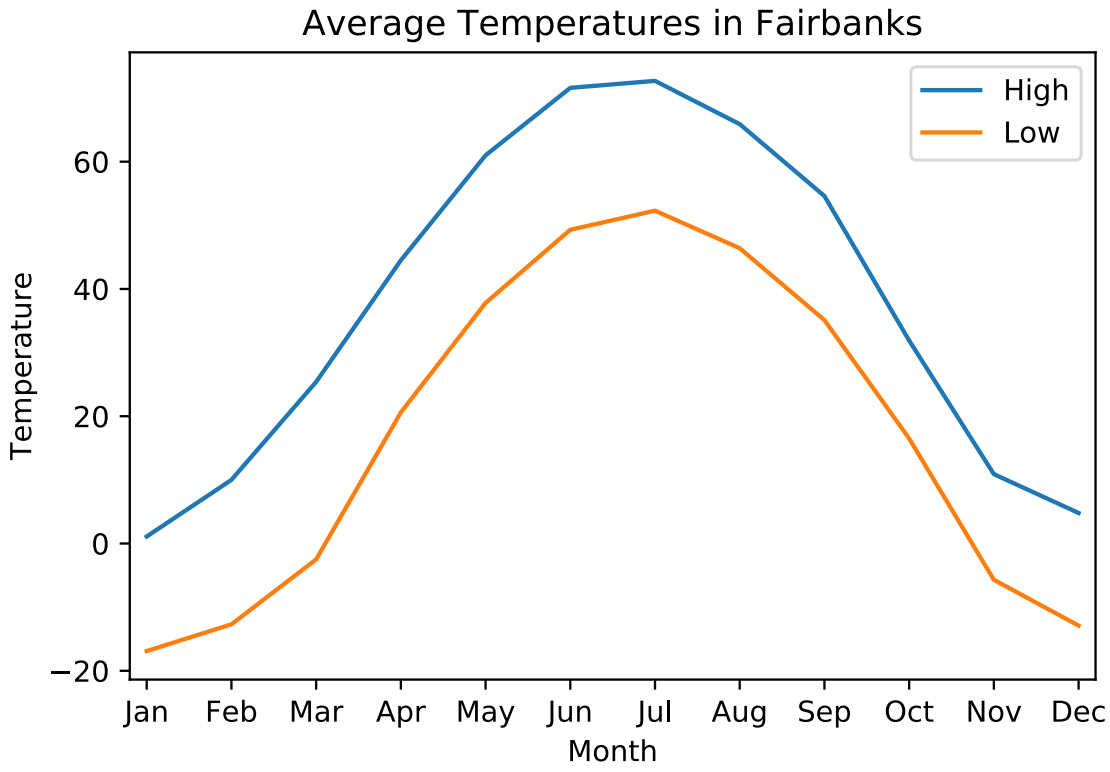
```
from matplotlib import pyplot
# we can also import matplotlib as follows:
# import matplotlib.pyplot as plt

%matplotlib inline

high_temps = [1.1,10.0,25.4,44.5,61.0,71.6,
              72.7,65.9,54.6,31.9,10.9,4.8]

months =["Jan","Feb","Mar","Apr","May","Jun",
         "Jul","Aug","Sep","Oct","Nov","Dec"]

pyplot.bar(months, high_temps)
pyplot.xlabel("Month")
pyplot.ylabel("Temperature")
pyplot.title("Average High Temperatures (F)
              in Fairbanks")
```



Simple Plots (Cont'd)

- **Decision: Should I accept a data scientist job in Fairbanks, Alaska?**
- Let's create a simple line-chart using matplotlib

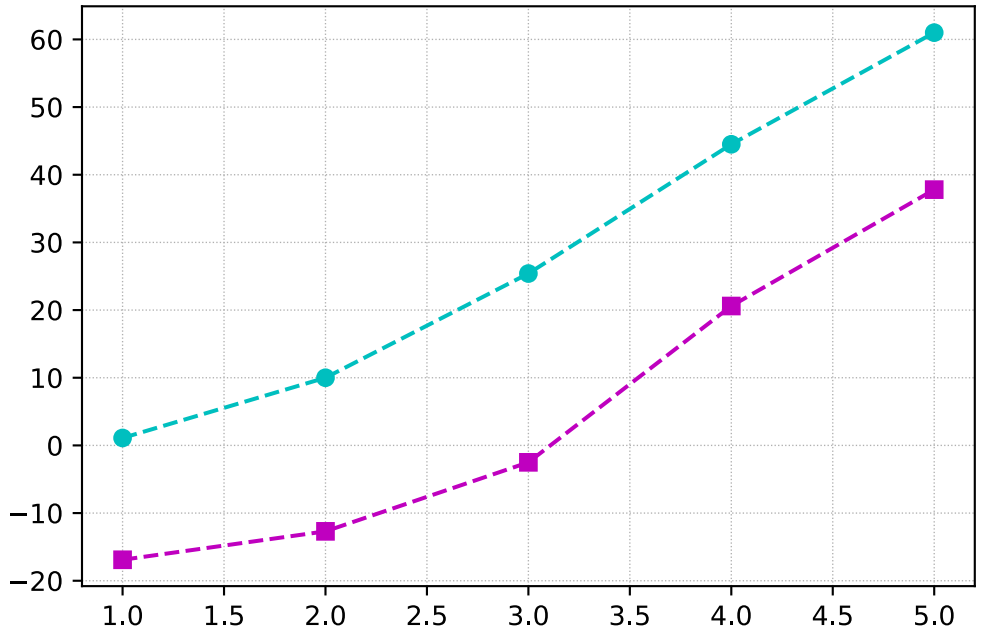
```
low_temps = [-16.9, -12.7, -2.5, 20.6, 37.8, 49.3,
             52.3, 46.4, 35.1, 16.5, -5.7, -12.9]

points = range(1, len(low_temps) + 1) # 1-12

pyplot.plot(points, high_temps)
pyplot.plot(points, low_temps)

# Add descriptive information.
pyplot.title("Average Temperatures in Fairbanks")
pyplot.xlabel("Month")
pyplot.ylabel("Temperature")
pyplot.legend(["High", "Low"])
pyplot.xticks(points, months)

# Change the x limits to give some padding
pyplot.xlim(0.8, 12.2)
```



Simple Plots (Cont'd)

- **Decision: Should I accept a data scientist job in Fairbanks, Alaska?**
- Styling a plot with markers and gridlines

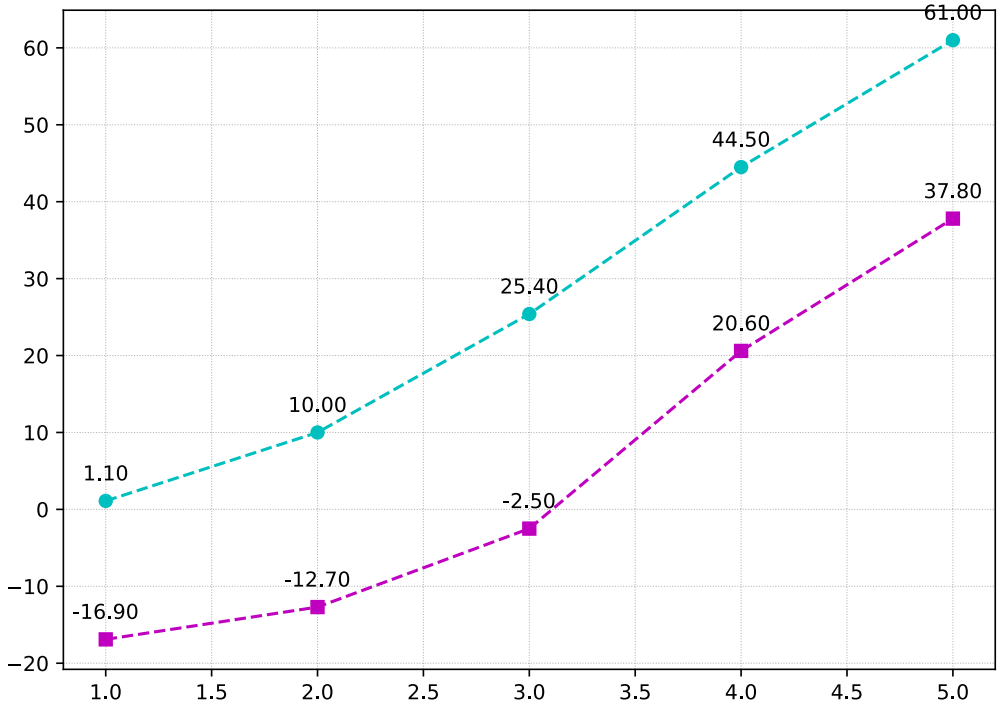
```
# Highs
pyplot.plot([1, 2, 3, 4, 5], [1.1, 10.0, 25.4, 44.5, 61.0], "c--o")

# Lows
pyplot.plot([1, 2, 3, 4, 5], [-16.9, -12.7, -2.5, 20.6, 37.8], "m--s")

pyplot.grid("on", linestyle=':', linewidth=0.5)
```



Plots with Annotation

- **Decision: Should I accept a data scientist job in Fairbanks, Alaska?**
- Annotating a chart

```
# First, store the data points in variables
values_1 = [1.1, 10.0, 25.4, 44.5, 61.0]
values_2 = [-16.9, -12.7, -2.5, 20.6, 37.8]
points = range(1, len(values_1) + 1)

...
# Add annotation for first line
for x,y in zip(points, values_1):
    label = "{:.2f}".format(y) # formatting values as float 2DP
    pyplot.annotate(label, # the text
        (x,y), # the point to label
        textcoords="offset points", # text position
        xytext=(0,10), # distance from text to points (x,y)
        ha='center') # horizontal alignment -- left, right or center

# repeat this code for the next line
```

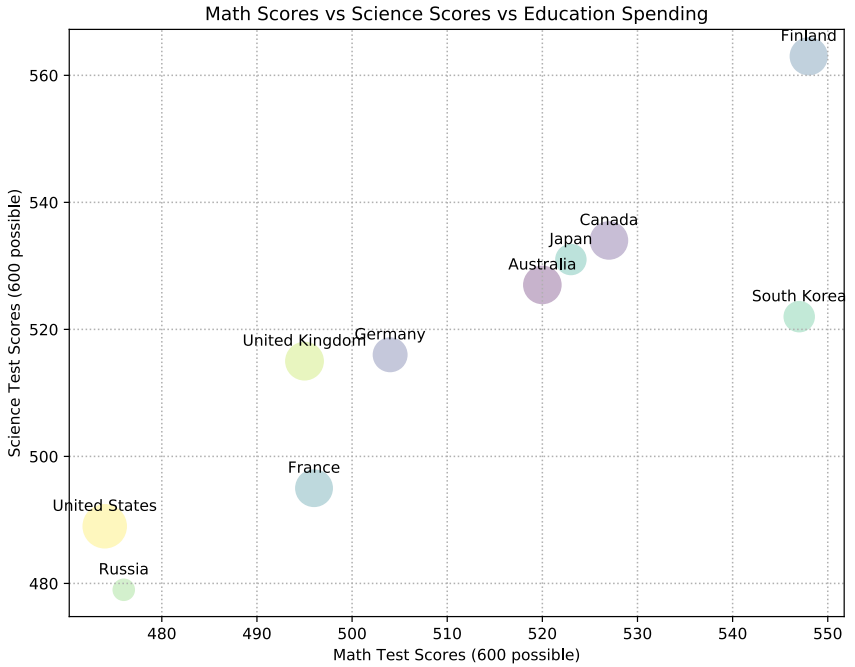


Creating a Bubble Chart

- A ***bubble chart*** illustrates the relationship between three-dimensional data, that is, each data point is made of three values (x, y, z), where z specifies the size of the bubble

```
# First, store the data points in variables
countries = []
math_scores = []
sci_scores = []
spending = []
# Extract these scores from the multiline string -->
# spending=(dollars / 10) to scale the bubbles so that
# they don't overlap
...
pyplot.scatter(math_scores, sci_scores, spending,
range(0, len(countries)), alpha=0.3)

# Label each bubble.
for i in range(len(countries)) :
    pyplot.annotate(countries[i], # this is the text
                # this is the point to label
                (math_scores[i], sci_scores[i]),
                textcoords="offset points", # how to position the text
                xytext=(0,10), # distance from text to points (x,y)
                ha='center') # horizontal alignment can be left, right or center
```

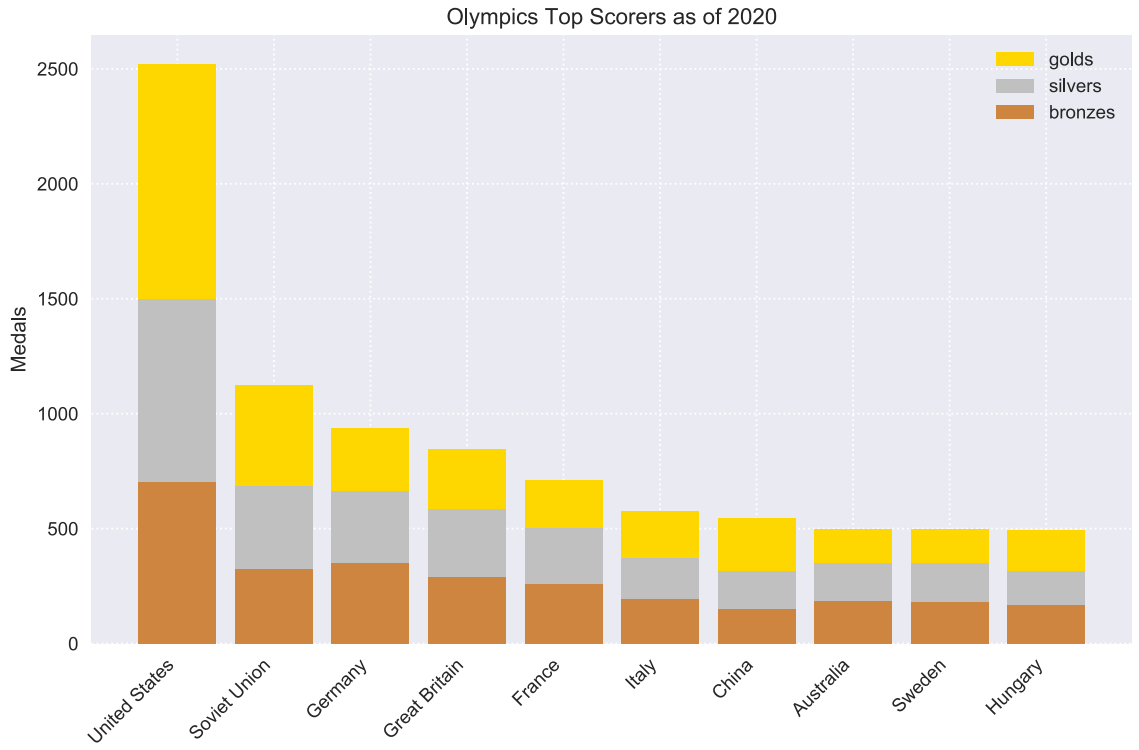
```
# country, dollars, math, science,
data = """
Australia, 5766, 520, 527
Canada, 5749, 527, 534
Germany, 4682, 504, 516
Finland, 5653, 548, 563
France, 5541, 496, 495
Japan, 3756, 523, 531
South Korea, 3759, 547, 522
Russia, 1850, 476, 479
United Kingdom, 5834, 495, 515
United States, 7743, 474, 489"""
```



Plotting a Stacked Plot/Column Chart

- **Example: All-time Gold Medals as of 2020**

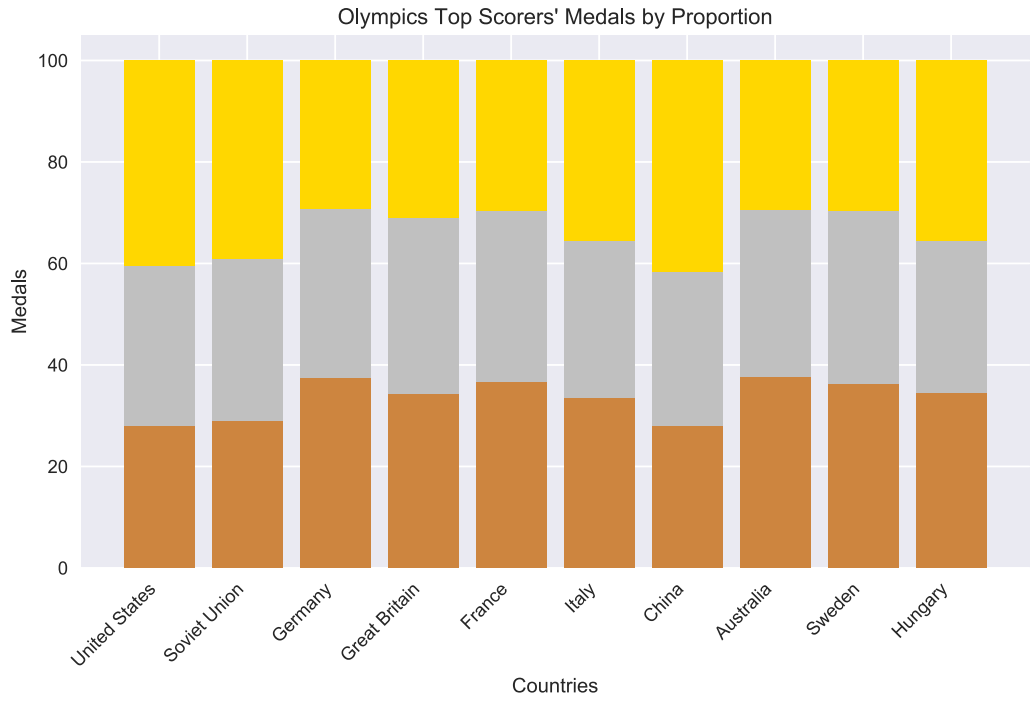
```
# Get the data here:
# https://www.topendsports.com/events/summer/medal-tally/all-time.htm

medals = pd.read_csv('olympic_medals.csv')
countries = medals["Team"]
bronzes = medals["bronze"]
silvers = medals["silver"]
golds = medals["gold"]
ind = medals["rank"]

plt.bar(ind, golds, width=0.8, label='golds', color='gold',
bottom=silvers+bronzes)
plt.bar(ind, silvers, width=0.8, label='silvers', color='silver',
bottom=bronzes)
plt.bar(ind, bronzes, width=0.8, label='bronzes', color='#CD853F')
plt.xticks(ind, countries)

plt.setp(plt.gca().get_xticklabels(), rotation=45,
horizontalalignment='right') # rotate the tick labels

# You could also rotate the ticks using the following:
# plt.xticks(index, label, fontsize=5, rotation=45)
```



Plotting a 100% Stacked Plot/Column Chart

- Example: All-time Gold Medals as of 2020

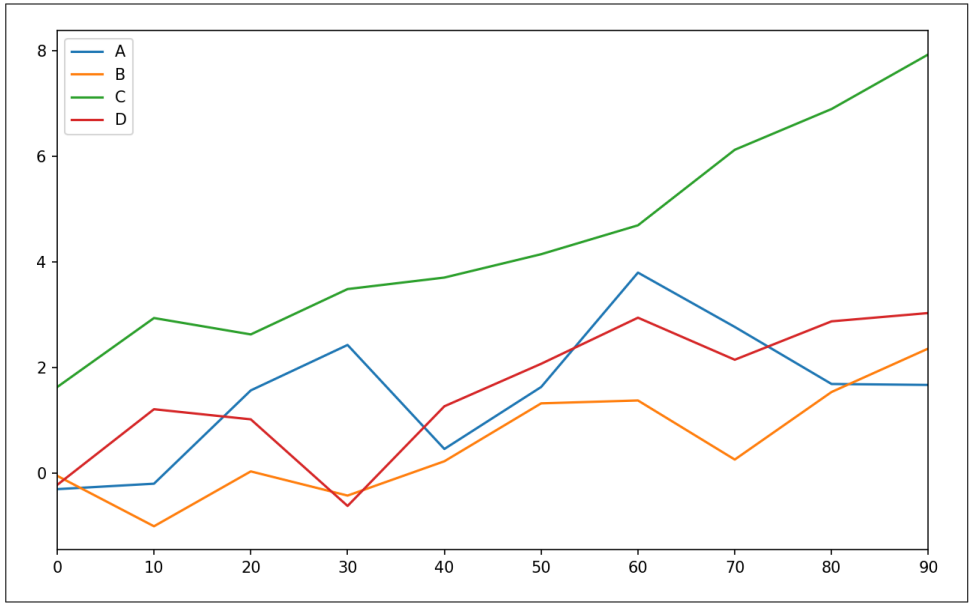
```
# Get the data here:
# https://www.topendsports.com/events/summer/medal-tally/all-time.htm

total = bronzes + silvers + golds
proportion_bronzes = np.true_divide(bronzes, total) * 100
proportion_silvers = np.true_divide(silvers, total) * 100
proportion_golds = np.true_divide(golds, total) * 100

plt.bar(ind, proportion_golds, width=0.8, label='gold', color='gold',
bottom=proportion_bronzes+proportion_silvers)
plt.bar(ind, proportion_silvers, width=0.8, label='silver',
color='silver', bottom=proportion_bronzes)
plt.bar(ind, proportion_bronzes, width=0.8, label='bronze’,
color='#CD853F')
```



Plotting Data from a Dataframe

- DataFrame's **plot** method plots each of its columns as a different line on the same subplot, creating a legend automatically

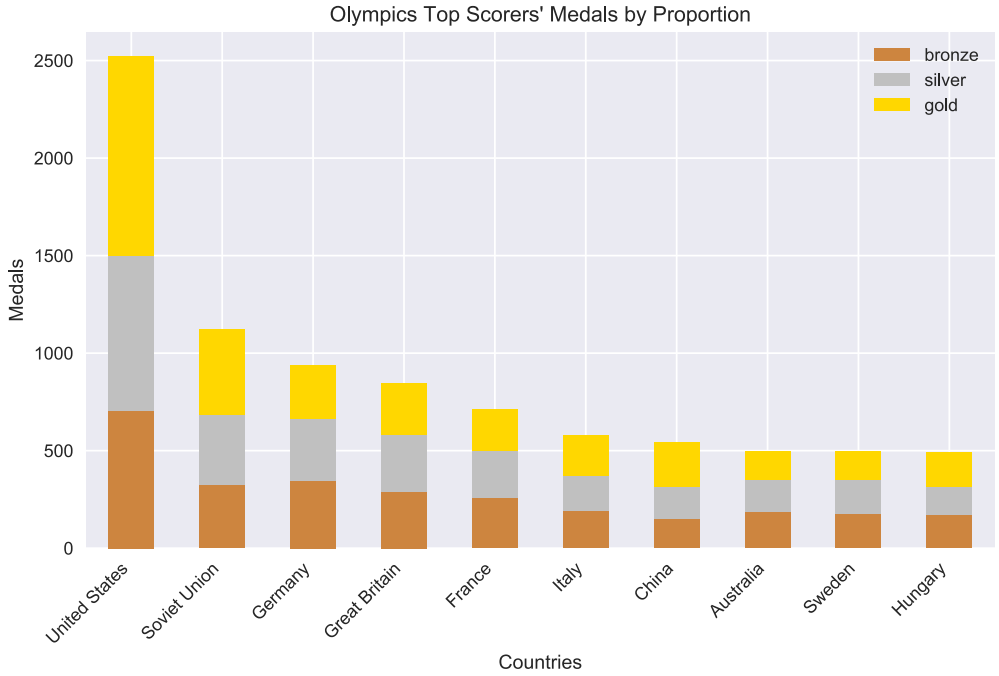
```
df = pd.DataFrame(np.random.randn(10, 4).cumsum(0),
columns=['A', 'B', 'C', 'D'],
index=np.arange(0, 100, 10))

df.plot()
```



Plotting a Stacked Plot from a Pandas DataFrame

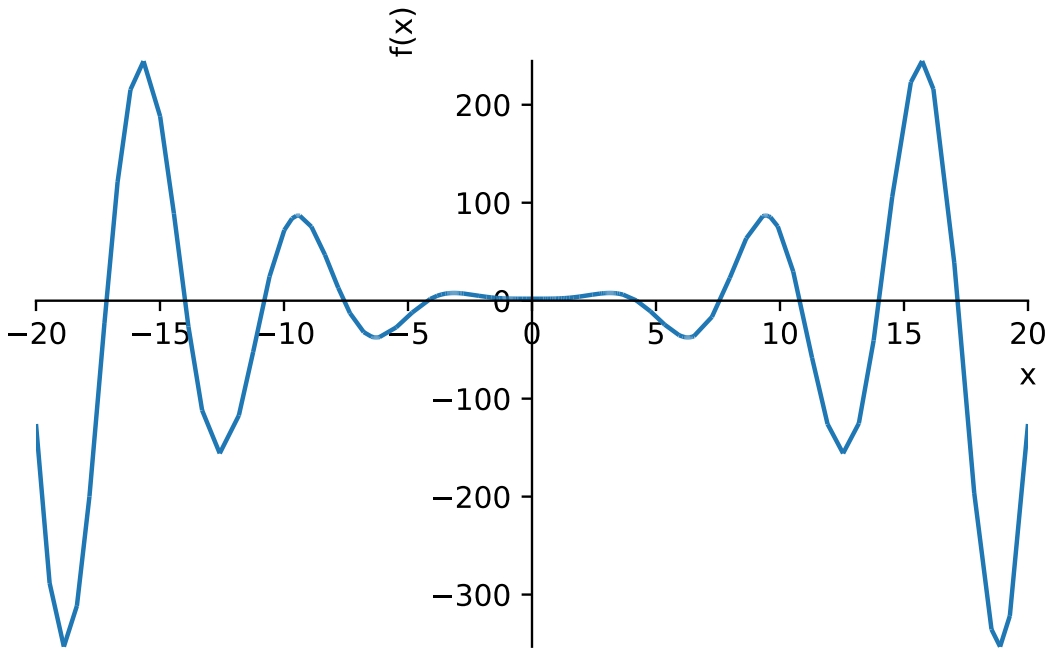
- Example: Plotting All-time Gold Medals as of 2020 from the DataFrame

```
# Get the data here:
# https://www.topendsports.com/events/summer/medal-tally/all-time.htm

medals = pd.read_csv('olympic_medals.csv', index_col='Team')
stack_info = medals[["bronze", "silver", "gold"]]

plot = stack_info.plot(kind='bar', stacked=True, color=["#CD853F", "silver", "gold"], title="Olympics Top Scorers' Medals by Proportion")

plot.set(xlabel='Countries', ylabel='Medals')
plt.setp(plt.gca().get_xticklabels(), rotation=45, horizontalalignment='right') # rotate the tick labels
```



Plotting Mathematical Functions Using Sympy

```
from sympy import *
%matplotlib inline
init_printing()
x = sympify("x") # make x a mathematical expression/variable
plot(-x**2 * cos(x) + 2 * x * sin(x) + 2 * cos(x), (x, -20, 20))
```



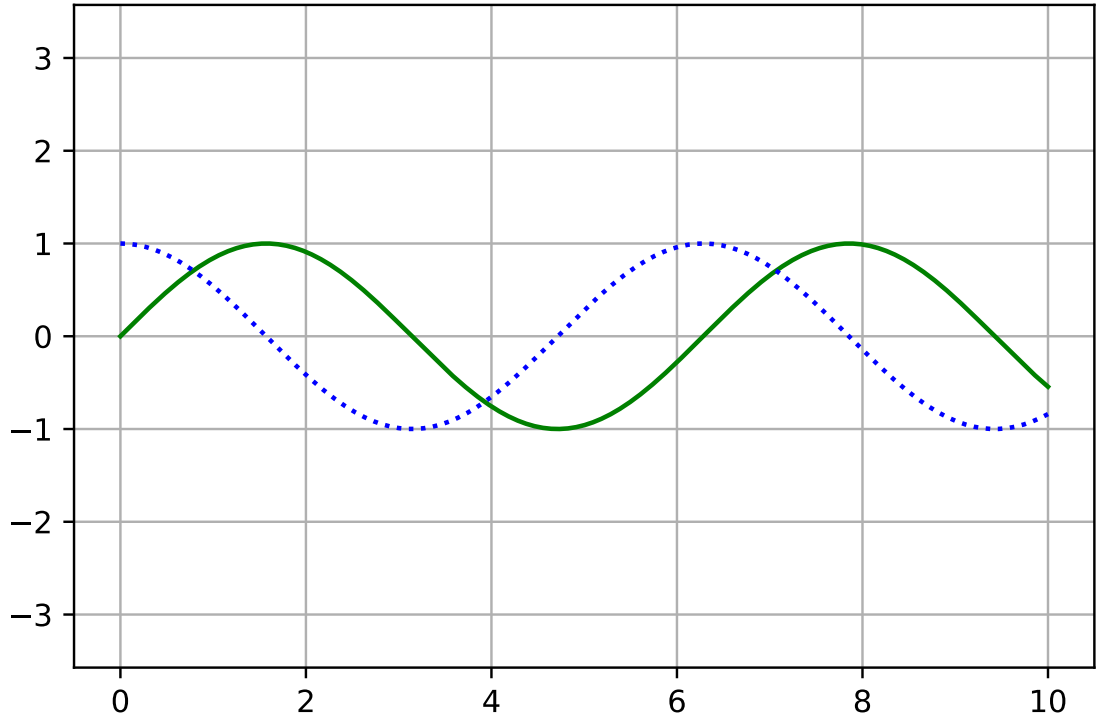
Plotting Mathematical Functions with Numpy Equations

```
x = np.linspace(0, 10, 1000)
plt.plot(x, np.sin(x), '-g', label='sin(x)')
plt.plot(x, np.cos(x), ':b', label='cos(x)')
plt.axis('equal')
plt.grid("on")
```

How does linspace work?

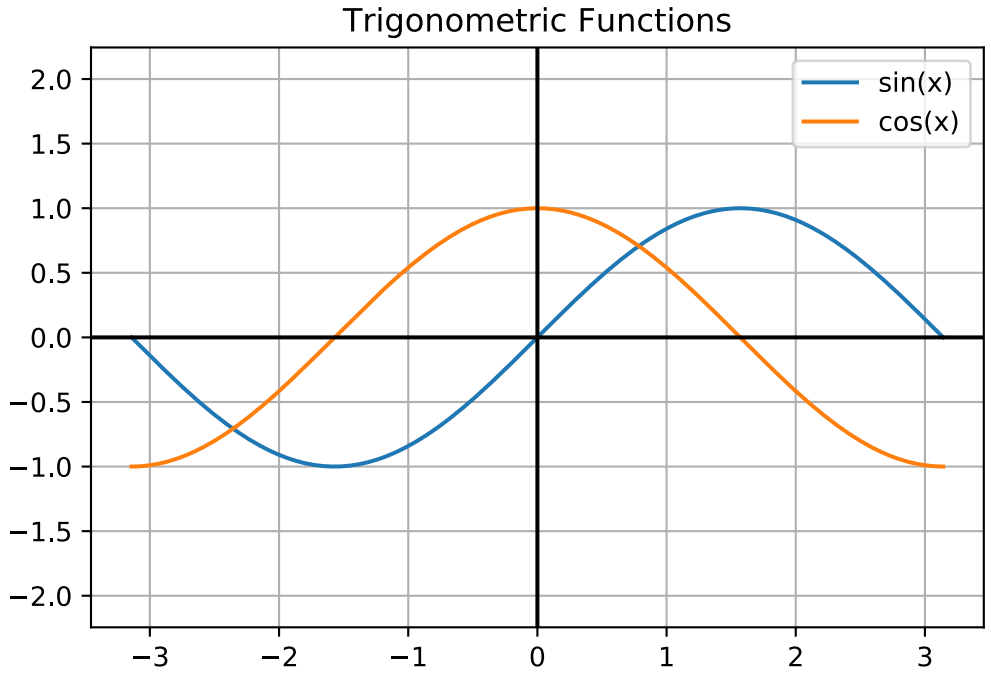
It creates sequences of evenly spaced values within a defined interval.

np.linspace(start = 0, stop = 100, num = 5)



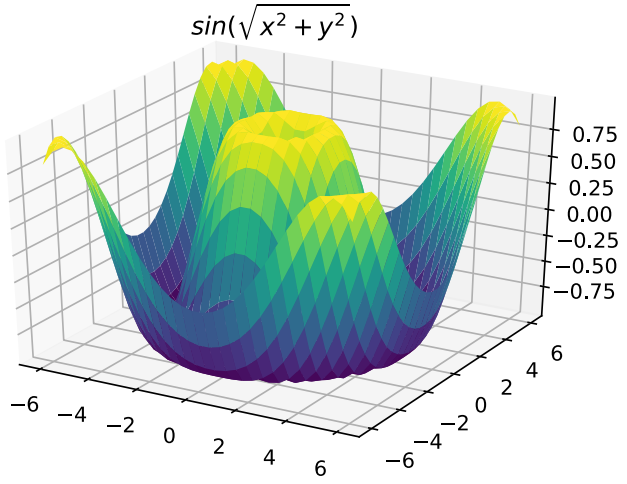
Plotting Mathematical Functions Using Matplotlib

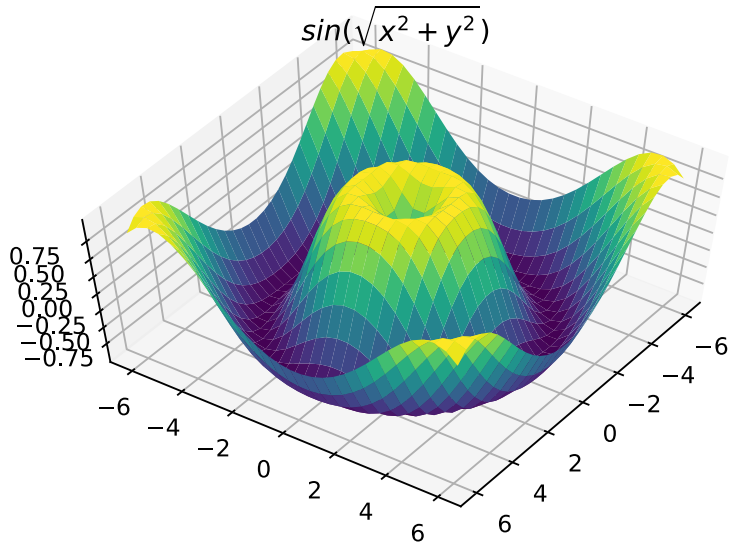
```
from math import pi, sin, cos

# Create empty lists to store the y-values for the sine and
cosine curves.
sin_y = []
cos_y = []

# The x-values will be the same for both curves.
trig_x = []

# Compute the y-values for the sine and cosine curves.
angle = -180
while angle <= 180 :
    x = pi / 180 * angle
    trig_x.append(x)

    y = sin(x)
    sin_y.append(y)

    y = cos(x)
    cos_y.append(y)
    angle = angle + 1

# Plot the two curves.
plt.plot(trig_x, sin_y)
plt.plot(trig_x, cos_y)
```



Plotting Mathematical Functions – 3D surfaces

```
def f(x, y):
    return np.sin(np.sqrt(x ** 2 + y ** 2))

x = np.linspace(-6, 6, 30)
y = np.linspace(-6, 6, 30)

X, Y = np.meshgrid(x, y)
Z = f(X,Y)

ax = plt.axes(projection='3d')
ax.plot_surface(X, Y, Z, rstride=1, cstride=1,
cmap='viridis', edgecolor='none')
ax.set_title('$sin(\sqrt{x^2+y^2})$')
```

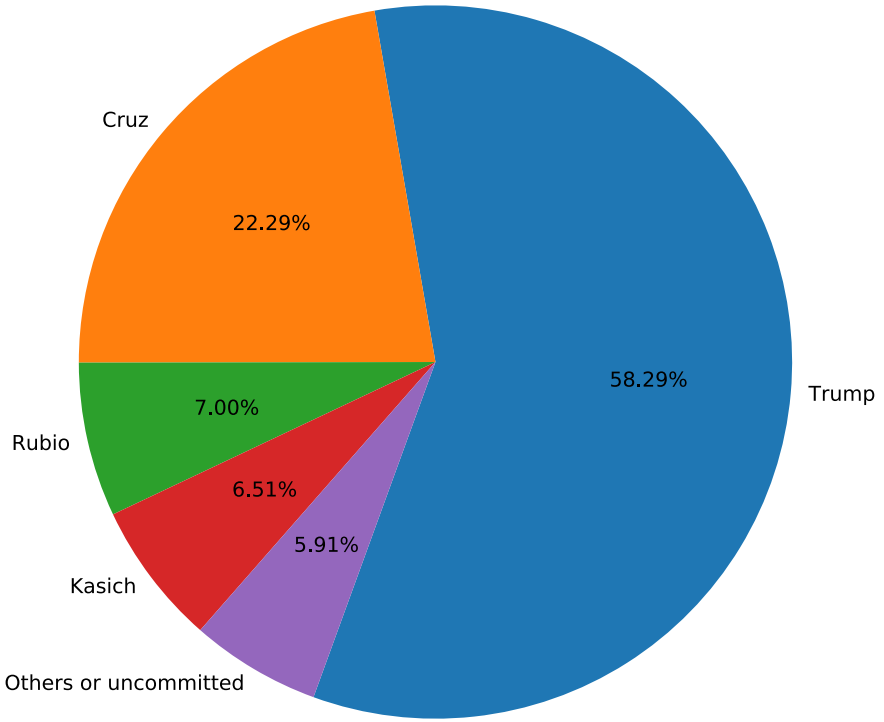





```
# Change the view with the following
ax.view_init(60, 35)

# You could also create your own color palettes.
This is discussed later
```

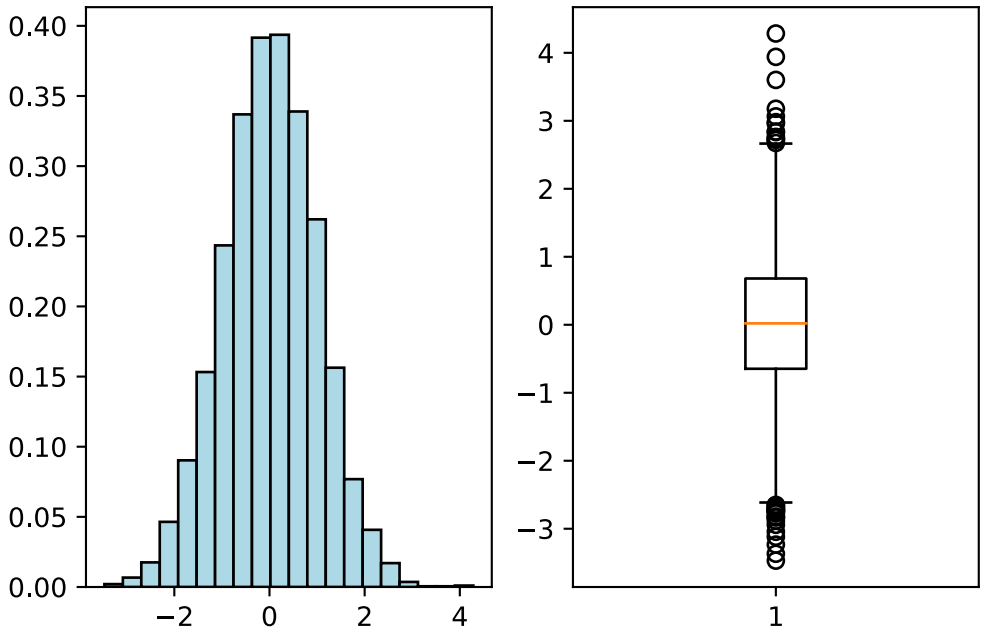
Creating a Pie-Chart

```
import matplotlib.pyplot as plt
slices = [1441,551,173,161,146] # counts

pres_names = ["Trump", "Cruz", "Rubio", "Kasich",
"Others or uncommitted"]

plt.figure(figsize=(8,8))

plt.pie(slices,
        labels=pres_names,
        startangle=-110,
        shadow=False,
        autopct="%1.2f%%", labeldistance=1.05)
plt.show();
```



Plotting Multiple Plots Side-by-side

```
import matplotlib.pyplot as plt
import numpy as np
sample_figure = plt.figure()

# 1 assign an arbitrary number to the axis for the first figure
ax1 = sample_figure.add_subplot(121)
# 2 assign an arbitrary number to the axis for the second figure
ax2 = sample_figure.add_subplot(122)

# Create plots based on random data
data = np.random.randn(5000)
ax1.hist(data, bins=20, normed=1, color="lightblue", edgecolor="black")
ax2.boxplot(data)
plt.show();
```



Other Plotting Tips

- **Change the font-size of a label**
  - plt.ylabel('Income', fontsize = 15) #for y label
- **Add transpacency to a plot**
  - plt.plot(x, y, alpha = 0.1)
- **Save plot at a certain resolution**
  - plot.savefig('my_plot.png', dpi=300)
- **From MATLAB-style code to OOP**
  - plt.xlabel() → ax.set_xlabel()
  - plt.ylabel() → ax.set_ylabel()
  - plt.xlim() → ax.set_xlim()
  - plt.ylim() → ax.set_ylim()
  - plt.title() → ax.set_title()
  - **Or set at once**
  - ax = plt.axes()
  - ax.plot(x, np.sin(x))
  - ax.set(xlim=(0, 10), ylim=(-2, 2), xlabel='x', ylabel='sin(x)', title='A Simple Plot');

Plotting Functions in Matplotlib Pyplot

Function	Description
`pyplot.bar(`*x-value*`, `*y-value*`)` `pyplot.bar([`*x-values*`], [`*y-values*`])`	Plots a single bar on the graph or multiple bars when the *x*- and *y*-values are provided as lists.
`pyplot.plot([`*x-coords*`], [`*y-coords*`])` `pyplot.plot([`*x-coords*`], [`*y-coords*`], `*format*`)`	Plots a line graph. The color and style of the line can be specified with a format string.
`pyplot.grid("on")`	Adds a grid to the graph.
`pyplot.xlim(`*min*`, `*max*`)` `pyplot.ylim(`*min*`, `*max*`)`	Sets the range of *x*- or *y*-values shown on the graph.
`pyplot.title(`*text*`)`	Adds a title to the graph.
`pyplot.xlabel(`*text*`)` `pyplot.ylabel(`*text*`)`	Adds a label below the *x*-axis or to the left of the *y*-axis.
`pyplot.legend([`$label_1$`, `$label_2$`, ...])`	Adds a legend for multiple lines.
`pyplot.xticks([`$x\text{-}coord_1$`, `$x\text{-}coord_2$`, ...],` `[`$label_1$`, `$label_2$`, ...])`	Adds labels below the tick marks along the *x*-axis.
`pyplot.yticks([`$y\text{-}coord_1$`, `$y\text{-}coord_2$`, ...],` `[`$label_1$`, `$label_2$`, ...])`	Adds labels to the left of the tick marks along the *y*-axis.
`pyplot.show()`	Displays the plot.

Changing the Appearance of a Pyplot Chart

Color Codes

Character	Color
b	Blue
g	Green
r	Red
c	Cyan
m	Magenta
y	Yellow
k	Black
w	White

Line Styles

Character	Style
-	Solid
--	Dashed
:	Dotted
-.	Alternating dashes and dots

```
# Plots with a red dashed
line with circles for
points

pyplot.plot(x, y, "r--o")
```

Marker Styles

Character	Description
.	Point marker
o	Circle marker
v	Triangle down marker
^	Triangle up marker
s	Square marker
*	Star marker
D	Diamond marker

Styling Plots

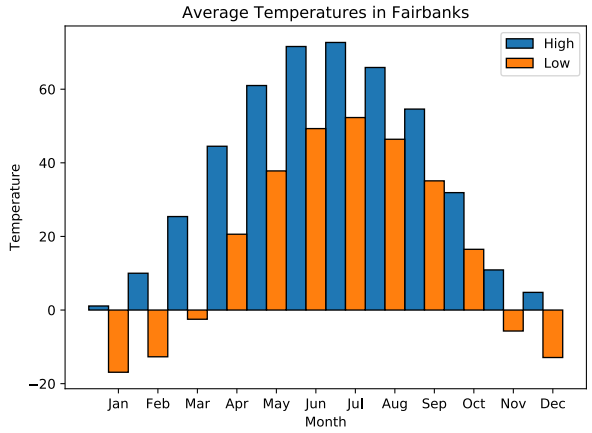
- You can specify the style to use for your plots from a variety of plots using the following line of code:
  - `plt.style.use('ggplot')  # mimics plot style in r`

- **To print out all available styles use**:
  - `print(plt.style.available)`

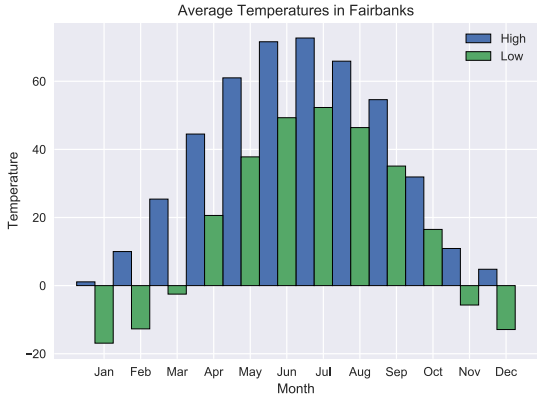
```
'Solarize_Light2', '_classic_test_patch', 'bmh', 'classic', 'dark_background',
'fast', 'fivethirtyeight', 'ggplot', 'grayscale', 'seaborn', 'seaborn-bright',
'seaborn-colorblind', 'seaborn-dark', 'seaborn-dark-palette', 'seaborn-darkgrid',
'seaborn-deep', 'seaborn-muted', 'seaborn-notebook', 'seaborn-paper', 'seaborn-
pastel', 'seaborn-poster', 'seaborn-talk', 'seaborn-ticks', 'seaborn-white',
'seaborn-whitegrid', 'tableau-colorblind10'
```

Styling Plots

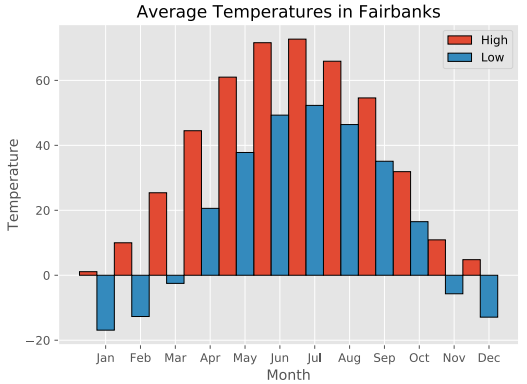
IPython Default



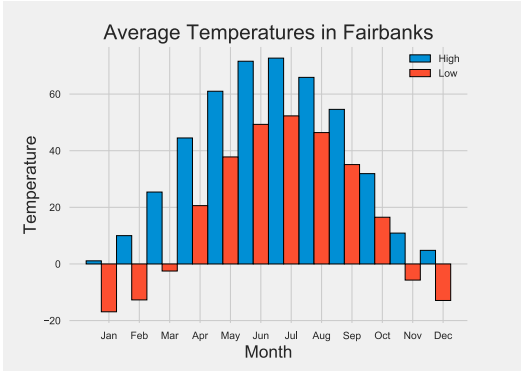
seaborn



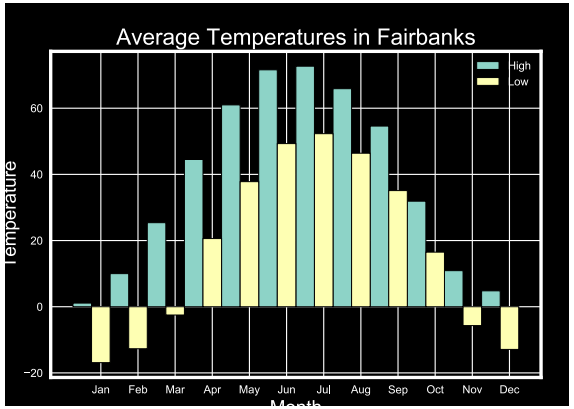
ggplot (to mimic plots in r)



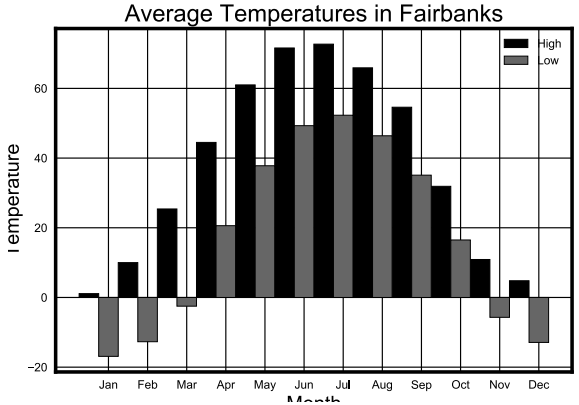
fivethirtyeight



dark_background



grayscale

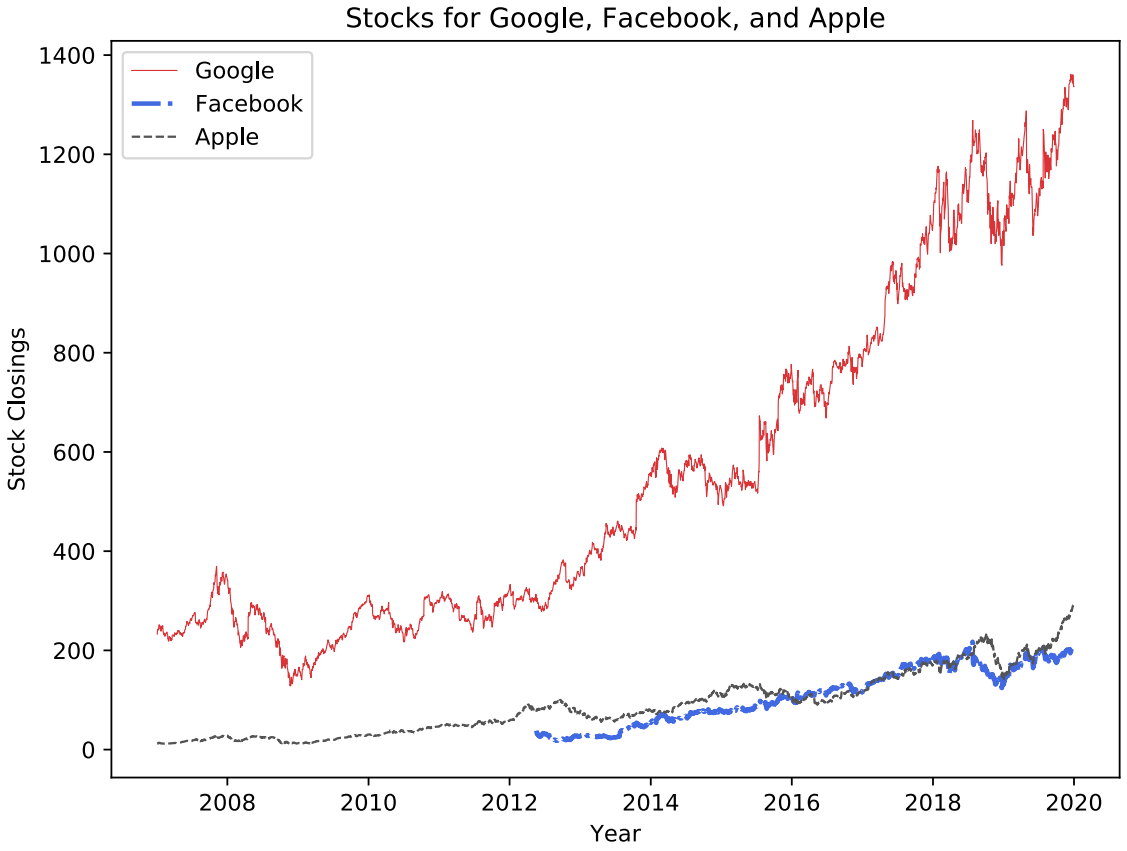


Named Colors for Plots

Adding a Legend

- The easiest way to add a legend to a Matplotlib plot is to pass the label argument when adding each piece of the plot:

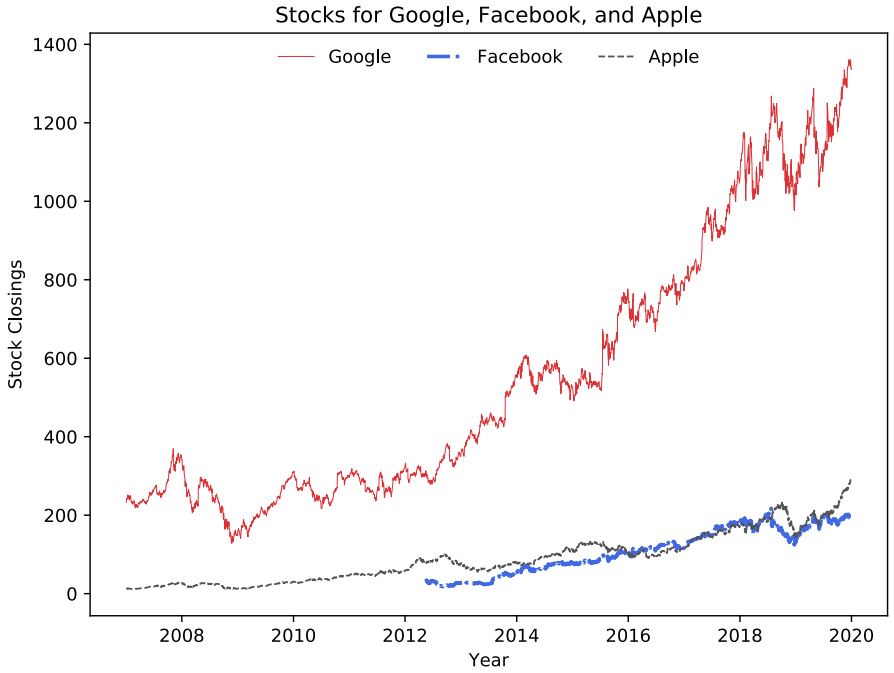
```
from pandas_datareader import data
# Get stocks for each company
apple = data.DataReader('AAPL',
start='2007', end='2020',
                data_source='yahoo')
...
ax.plot(goog["Close"], 'k-', label='Google',
color="#db3236", linewidth=0.5)
ax.plot(fb["Close"], 'k-.',
label='Facebook', color="royalblue",
linewidth=2)
ax.plot(apple["Close"], 'k--',
label='Apple', color="#555555", linewidth=1)
ax.legend(loc='best')
```



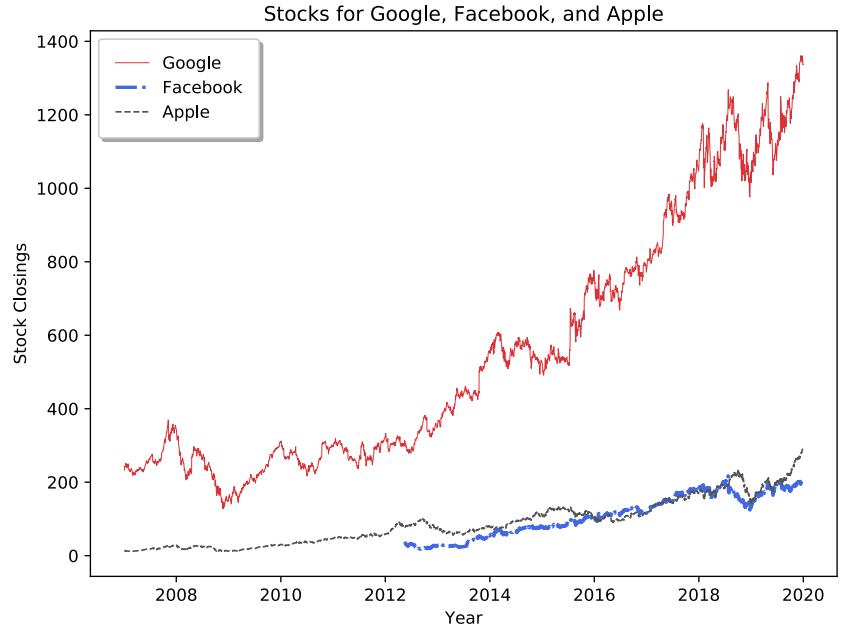
Styling Legends

- We can put the legend in one line and remove the frame
- We can also use a rounded box (fancybox) or add a shadow, change the transparency (alpha value) of the frame, or change the padding around the text

```
ax.legend(frameon=False, loc='upper center',
ncol=3)
```



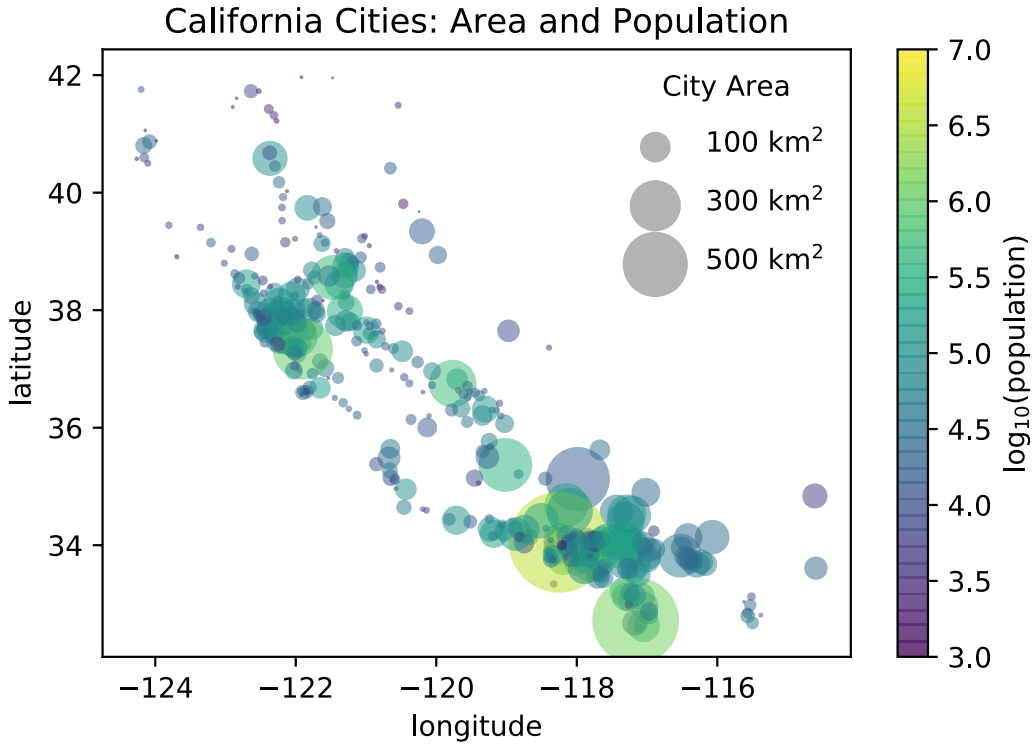

```
ax.legend(fancybox=True, framealpha=1,
shadow=True, borderpad=1)
```



Legend Based on Size of Plot Points

- We can use use the size of points to indicate populations of California cities

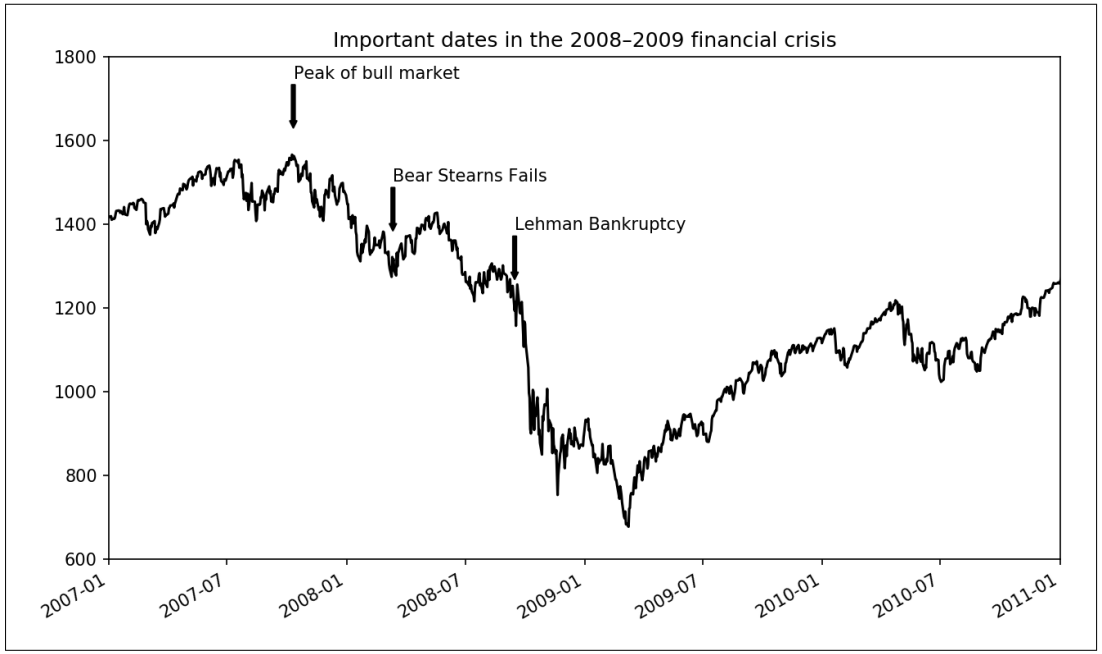
```
cities = pd.read_csv('california_cities.csv')
# Extract the data of interest
lat, lon = cities['latd'], cities['longd']
population, area = cities['population_total'],
cities['area_total_km2']
# Scatter the points, using size and color but no label
plt.scatter(lon, lat, label=None,
c=np.log10(population), cmap='viridis',
s=area, linewidth=0, alpha=0.5)
plt.axis(aspect='equal')
plt.xlabel('longitude')
plt.ylabel('latitude')
plt.colorbar(label='log$_{10}$(population)')
plt.clim(3, 7)
# Here we create a legend:
# we'll plot empty lists with the desired size and label
for area in [100, 300, 500]:
    plt.scatter([], [], c='k', alpha=0.3, s=area, label=str(area) + ' km$^2$')
plt.legend(scatterpoints=1, frameon=False, labelspacing=1, title='City Area')
plt.title('California Cities: Area and Population')
```



Adding Annotations with Arrows

- Let's plot the closing S&P 500 index price since 2007 (obtained from Yahoo! Finance) and annotate it with some of the important dates from the 2008–2009 financial crisis
- Locate the ***spx*** dataset on Canvas

```
from datetime import datetime
data = pd.read_csv('spx.csv', index_col=0, parse_dates=True)
spx = data['SPX']
spx.plot(ax=ax, style='k-')
crisis_data = [
(datetime(2007, 10, 11), 'Peak of bull market'),
(datetime(2008, 3, 12), 'Bear Stearns Fails'),
(datetime(2008, 9, 15), 'Lehman Bankruptcy')
]
for date, label in crisis_data:
      ax.annotate(label, xy=(date, spx.asof(date) + 75),
      xytext=(date, spx.asof(date) + 225),
      arrowprops=dict(facecolor='black', headwidth=4, width=2,
                      headlength=4),
      horizontalalignment='left', verticalalignment='top')
# Zoom in on 2007-2010
ax.set_xlim(['1/1/2007', '1/1/2011'])
ax.set_ylim([600, 1800])

# asof can be misleading because it returns only one data point
matching that date and not a range of values
```



Series.plot method arguments

- The plot attribute contains a “family” of methods for different plot types. For example, df.plot() is equivalent to df.plot.line()

Argument	Description
`label`	Label for plot legend
`ax`	matplotlib subplot object to plot on; if nothing passed, uses active matplotlib subplot
`style`	Style string, like `'ko--'`, to be passed to matplotlib
`alpha`	The plot fill opacity (from 0 to 1)
`kind`	Can be `'area'`, `'bar'`, `'barh'`, `'density'`, `'hist'`, `'kde'`, `'line'`, `'pie'`
`logy`	Use logarithmic scaling on the y-axis
`use_index`	Use the object index for tick labels
`rot`	Rotation of tick labels (0 through 360)
`xticks`	Values to use for x-axis ticks
`yticks`	Values to use for y-axis ticks
`xlim`	x-axis limits (e.g., `[0, 10]`)
`ylim`	y-axis limits
`grid`	Display axis grid (on by default)

DataFrame-specific plot arguments

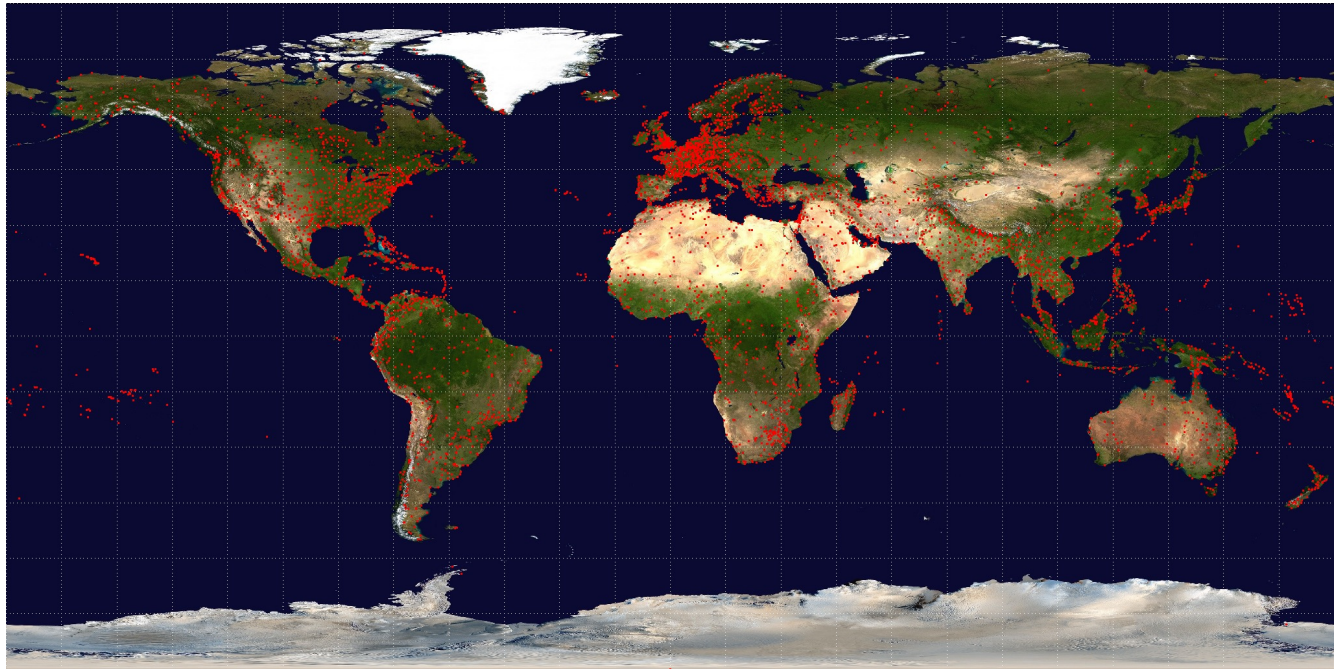
- DataFrame has a number of options allowing some flexibility with how the columns are handled such as using the same subplot or creating separate subplots

Argument	Description
`subplots`	Plot each DataFrame column in a separate subplot
`sharex`	If `subplots=True`, share the same x-axis, linking ticks and limits
`sharey`	If `subplots=True`, share the same y-axis
`figsize`	Size of figure to create as tuple
`title`	Plot title as string
`legend`	Add a subplot legend (`True` by default)
`sort_columns`	Plot columns in alphabetical order; by default uses existing column order

Geographic Data with Cartopy

- Create a plot showing all the airports in the following file: https://raw.githubusercontent.com/jpatokal/openflights/master/data/airports.dat

- Sample entries
- 507,"London Heathrow Airport","London","United Kingdom","LHR","EGLL",51.4706,-0.461941,83,0,"E","Europe/London","airport","OurAirports"
- 26,"Kugaaruk Airport","Pelly Bay","Canada","YBB","CYBB",68.534401,-89.808098,56,-7,"A","America/Edmonton","airport","OurAirports"
- 3127,"Pokhara Airport","Pokhara","Nepal","PKR","VNPK",28.200899124145508,83.98210144042969,2712,5.75,"N","Asia/Katmandu","airport","OurAirports"
  8810,"Hamburg Hbf","Hamburg","Germany","ZMB",\N,53.552776,10.006683,30,1,"E","Europe/Berlin","station","User"



Source: https://openflights.org/data.html

5B.3 SEABORN

Seaborn

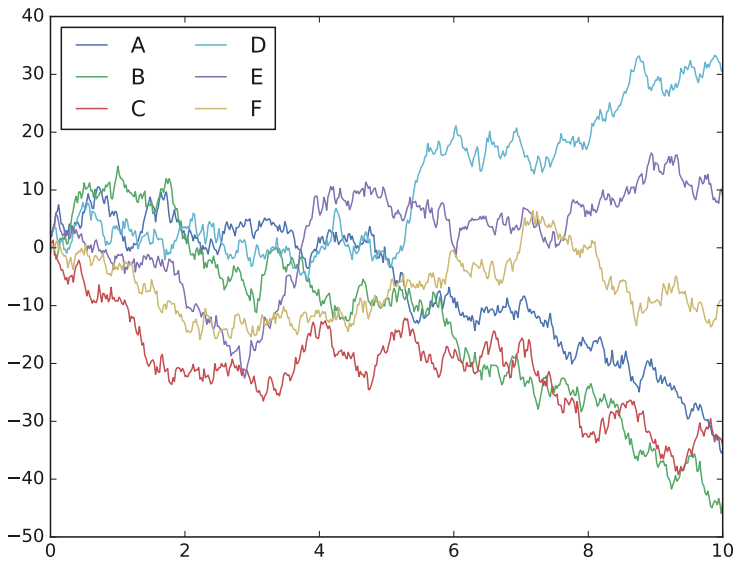
- **Seaborn** is a statistical graphics library created by Michael Waskom.
- It provides an API on top of Matplotlib that offers sane choices for plot style and color defaults, defines simple high-level functions for common statistical plot types, and integrates with the functionality provided by Pandas DataFrames.

- The common practice is to import seaborn as follows:
  - `import seaborn as sns`
  - `sns.set(style="whitegrid") # white, dark, whitegrid, darkgrid, ticks`
- Recall that you can use seaborn styles without importing the package
  - `from matplotlib import pyplot`
  - `pyplot.style.use('seaborn')`

Seaborn

- A classic plot vs a seaborn plot

```
rng = np.random.RandomState(0)
x = np.linspace(0, 10, 500)
y = np.cumsum(rng.randn(500, 6), 0)
plt.plot(x, y)
```



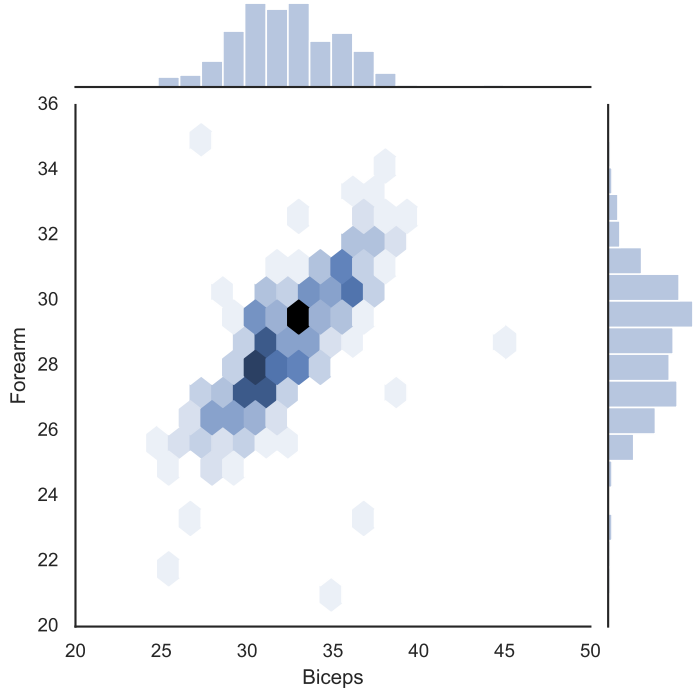

```
import seaborn as sns
rng = np.random.RandomState(0)
x = np.linspace(0, 10, 500)
y = np.cumsum(rng.randn(500, 6), 0)
sns.set()
plt.plot(x, y)
```

Plotting a Jointplot in Seaborn

- We can plot joint distribution and the marginal distributions together using sns.jointplot
- It has kinds such as **scatter, reg, resid, kde, and hex**

- **Kernel Density Estimate** (KDE) is used for visualizing the Probability Density of a continuous variable
- **"hex"** represents a joint histogram using hexagonal bins
- **"resid"** plots the residual of the data to the regression line

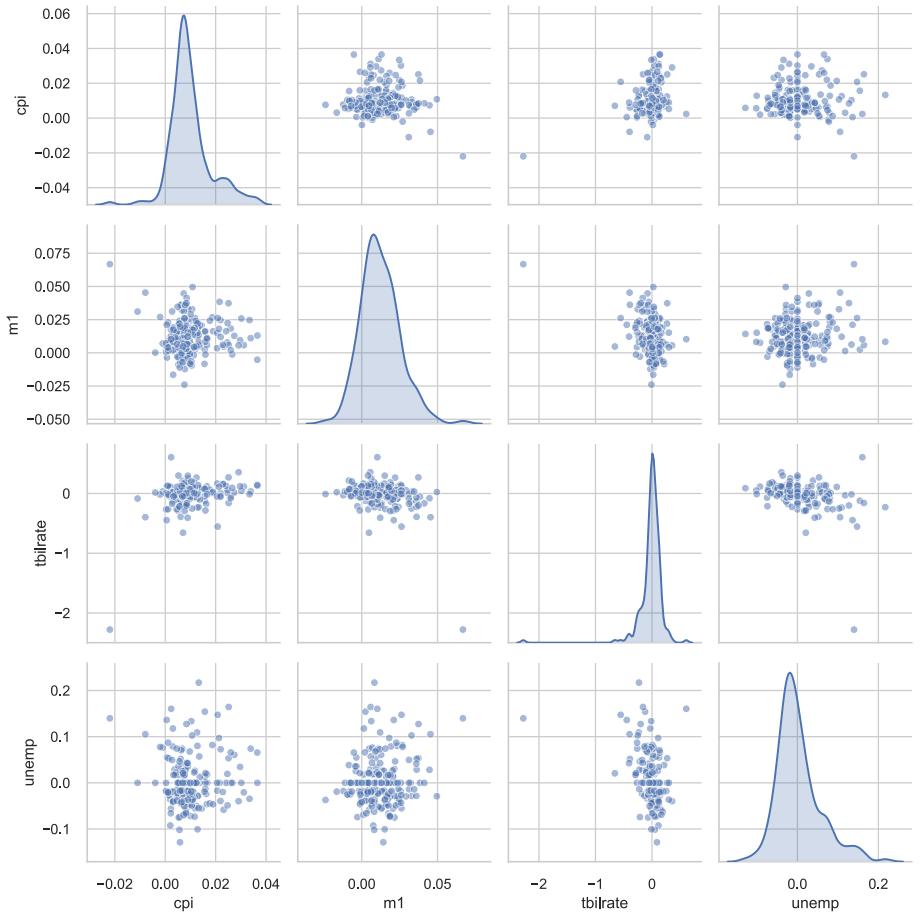
```
# Locate the BodyFat dataset here : http://lib.stat.cmu.edu/datasets/bodyfat

data = pd.read_csv("BodyFat.csv")
data = pd.DataFrame(data, columns=["Biceps", "Forearm"])

with sns.axes_style('white'):
    sns.jointplot("Biceps", "Forearm", data, kind='hex')
```



Plotting a Pairplot in Seaborn

- In exploratory data analysis, it is helpful to be able to look at all the scatter plots among a group of variables; this is known as a ***pairs*** plot or ***scatter plot matrix***

```
macro = pd.read_csv('macrodata.csv')
data = macro[['cpi', 'm1', 'tbilrate', 'unemp']]
trans_data = np.log(data).diff().dropna()


# styles are white, dark, whitegrid, darkgrid, ticks
sns.set(style="whitegrid")
sns.pairplot(trans_data, diag_kind='kde', plot_kws={'alpha': 0.5})
```

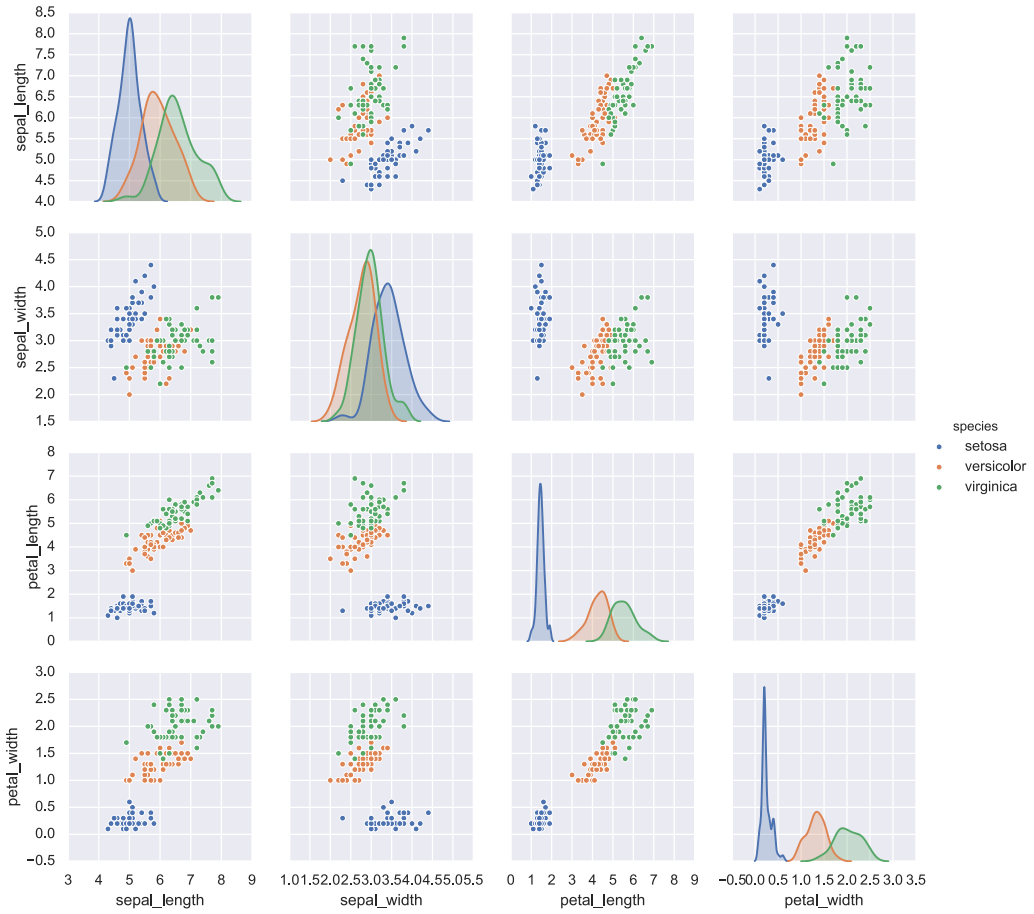


Plotting a Pairplot in Seaborn (Cont'd)

- In exploratory data analysis it's helpful to be able to look at all the scatter plots among a group of variables; this is known as a ***pairs*** plot or ***scatter plot matrix***

```
# The iris dataset is provided with the seaborn package.
# Other datasets include the tips dataset, titanic, etc.
iris = sns.load_dataset("iris")
sns.pairplot(iris, hue='species', height=2.5);
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica



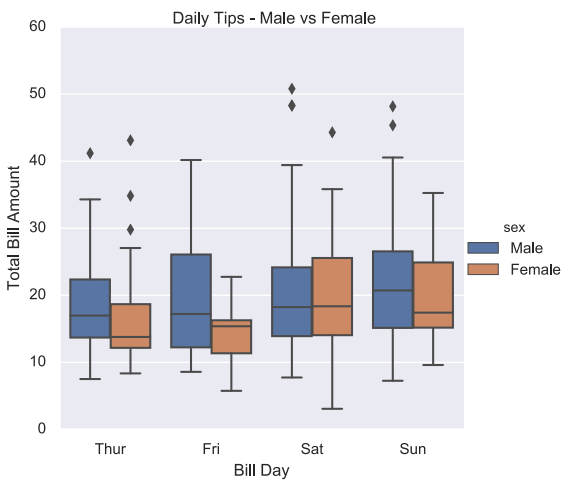
Plotting a Category Plot in Seaborn

- In seaborn, there are several different ways to visualize a relationship involving categorical data. The ***catplot*** and ***FacetGrid*** are two examples.

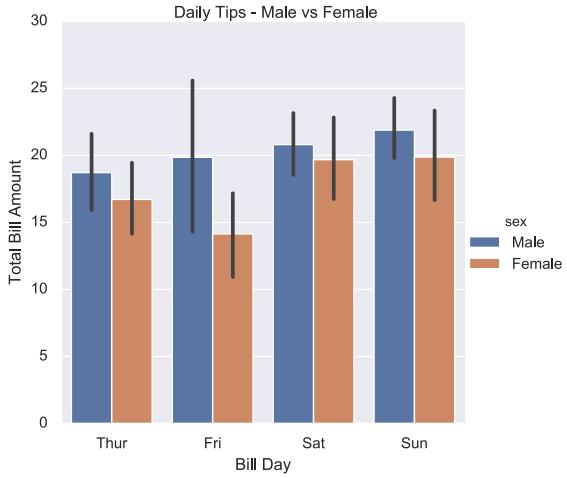
total_bill	tip	sex	smoker	day	time	size	
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4
...	...	...	...	...	...	...	...
239	29.03	5.92	Male	No	Sat	Dinner	3
240	27.18	2.00	Female	Yes	Sat	Dinner	2
241	22.67	2.00	Male	Yes	Sat	Dinner	2
242	17.82	1.75	Male	No	Sat	Dinner	2
243	18.78	3.00	Female	No	Thur	Dinner	2

```
# For the tips dataset, a food server recorded the data on all customers they served during an interval of two and a half months in early 1990.

tips = sns.load_dataset('tips')
with sns.axes_style(style='darkgrid'):
    g = sns.catplot("day", "total_bill","sex", data=tips, kind="bar")
    g.set_axis_labels("Bill Day", "Total Bill Amount")
    g.fig.suptitle('Daily Tips - Male vs Female')
```

kind="box"



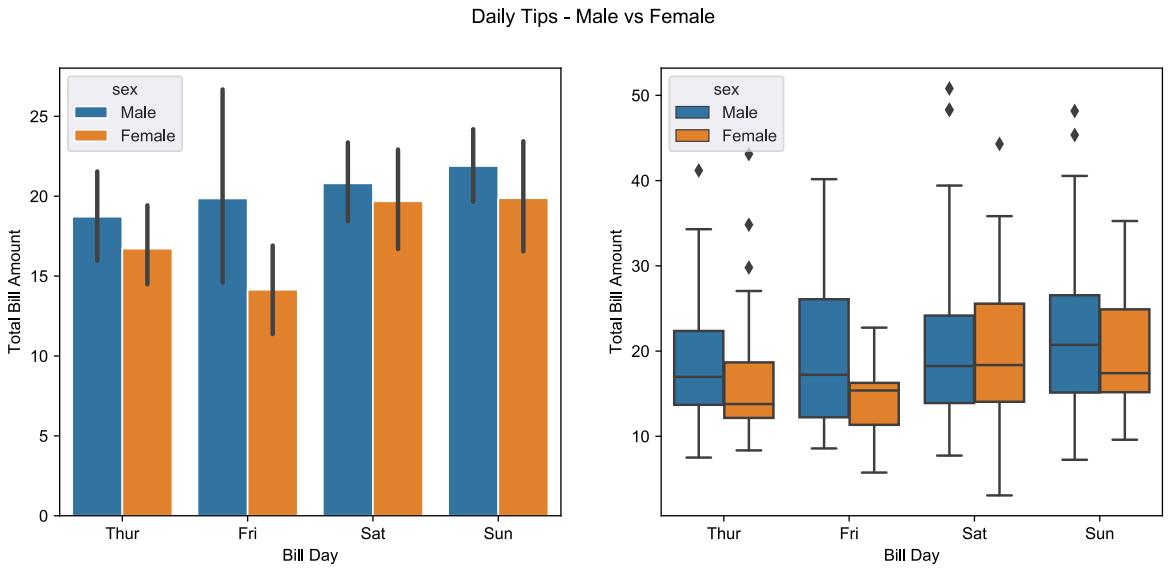
kind="bar"



Plotting Subplots in Seaborn

- It may be of interest to plot two or more plots on one plot

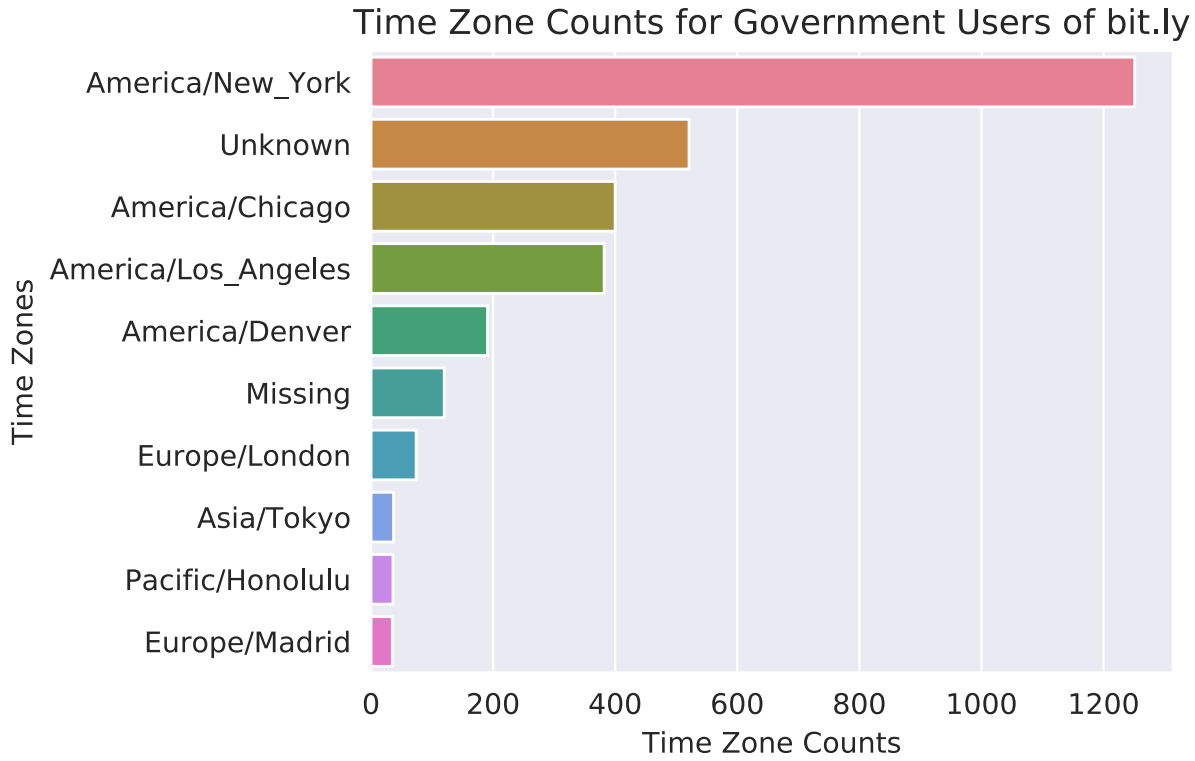
```
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
tips = sns.load_dataset('tips')

fig, axes = plt.subplots(1, 2, figsize=(12,5)) #numrows, numcols

fig.suptitle('Daily Tips - Male vs Female')

with sns.axes_style(style='darkgrid'):
    # first plot
    g= sns.catplot("day", "total_bill","sex", data=tips, kind="bar", ax=axes[0])
    axes[0].set_xlabel("Bill Day")
    axes[0].set_ylabel("Total Bill Amount")
    # second plot
    g = sns.catplot("day", "total_bill","sex", data=tips, kind="box", ax=axes[1])
    axes[1].set_xlabel("Bill Day")
    axes[1].set_ylabel("Total Bill Amount")
    plt.close(2) # close the empty default plot
    plt.close(3) # close the empty default plot
    plt.show()
```



Plotting a Horizontal *Barplot* in Seaborn

- Let's summarize and plot data from a json file
- Locate *bitly_usagov_example.json* on Canvas

```
# The data represents a feed of anonymous data gathered from users who
used bit.ly as of 2011 to shorten links ending with .gov or .mil

# lines=True when json file consists of a list of json lines
df = pd.read_json ("bitly_usagov_example.json", lines=True)

clean_tz = df['tz'].fillna('Missing')
clean_tz[clean_tz == ''] = 'Unknown'
tz_counts = clean_tz.value_counts()
with sns.axes_style(style='darkgrid'):
    ax = sns.barplot(y=subset.index, x=subset.values,
     palette = 'husl')
    ax.set(xlabel='Time Zone Counts', ylabel='Time Zones',
     title="Time Zone Counts for Government Users of bit.ly")
plt.tight_layout()

# husl is the default palette for seaborn barplots, etc.
# There are six variations of the default palette
(deep, muted, pastel, bright, dark, and colorblind), but there are many
other palettes. A few examples are shown on the next slide. You could
also create your own palette.
```



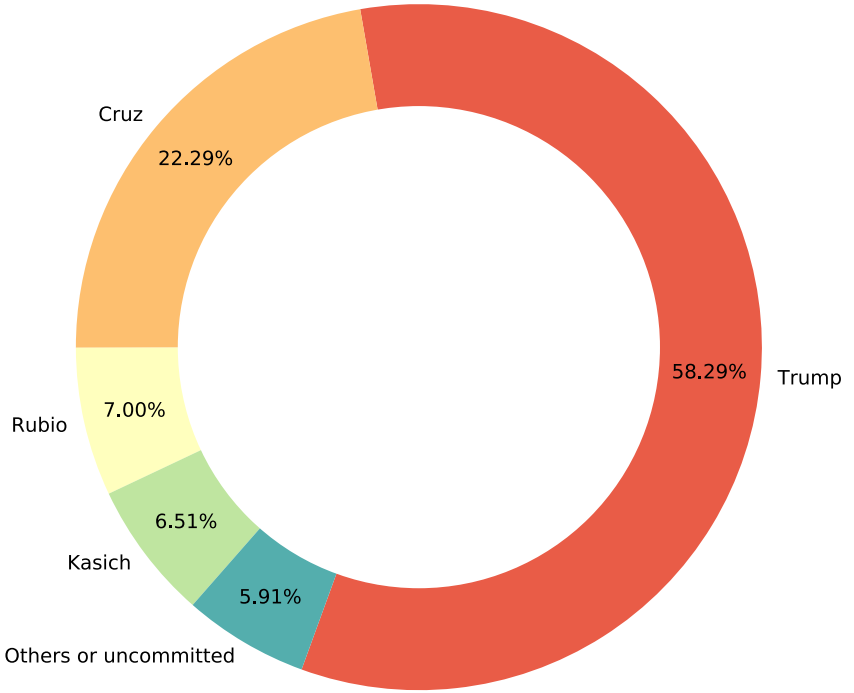
Example Seaborn Color Palettes

- A color palette can also be used as a ***cmap*** for plots such as 3D surfaces, kde-plots, etc.
- You may select a palette as follows:
  - `my_palette = sns.color_palette("husl")`
- You may also create a palette:

```
my_palette = sns.dark_palette("purple", as_cmap=True)
Example use: sns.kdeplot(x, y, cmap=my_palette);
```

Creating a Donut Chart (With colors from a Color Palette)

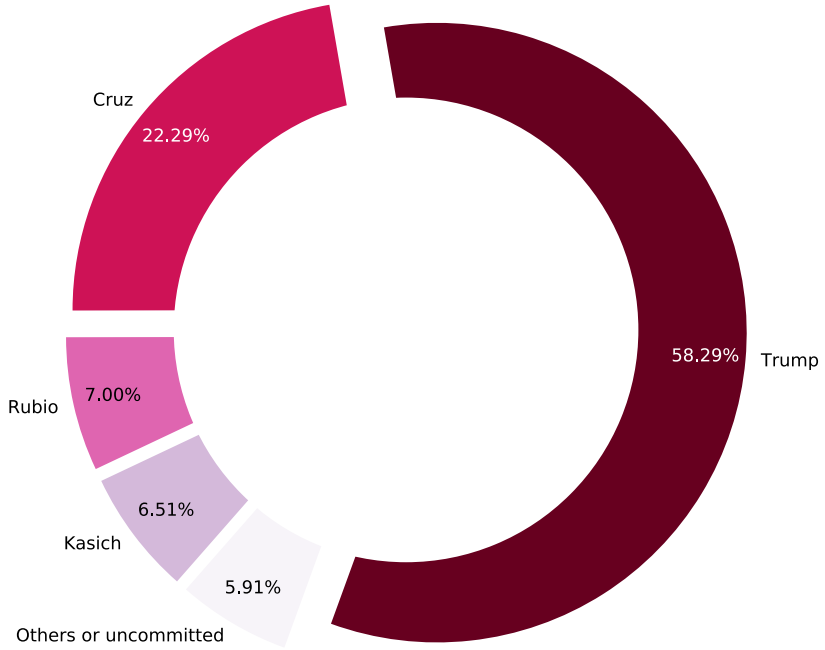
```
import matplotlib.pyplot as plt
slices = [1441,551,173,161,146]

pres_names = ["Trump", "Cruz", "Rubio", "Kasich", "Others or uncommitted"]

# create a list of colors based on a built-in palette
# specify the number of colors based on the slices
my_palette = sns.color_palette("Spectral", len(slices))
pal_hls = my_palette.as_hex();

plt.figure(figsize=(8,8))

# Create a circle for the center of the plot
my_circle=plt.Circle( (0,0), 0.7, color="white")
p = plt.gcf()
p.gca().add_artist(my_circle)

plt.pie(slices,
        labels=pres_names,
        startangle=-110,
        shadow=False,
        colors = pal_hls, # You may pass in a list of colors of your choice
        autopct="%1.2f%%", labeldistance=1.05, pctdistance=0.85)
plt.show();
```



Creating a Donut Chart – Another Trick (2 Pies)

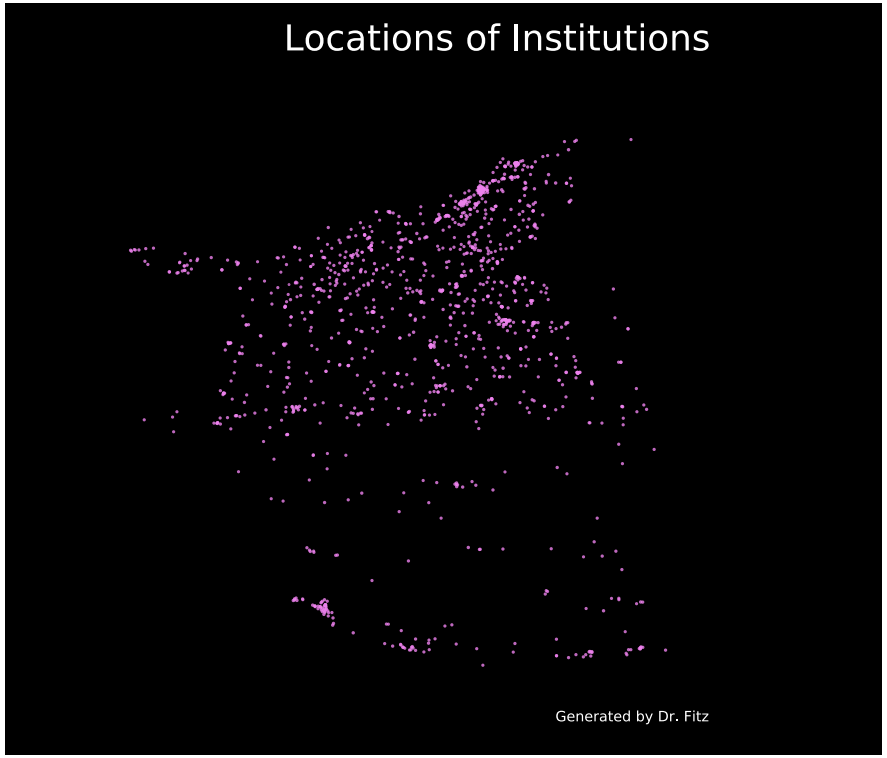
```
import matplotlib.pyplot as plt
import matplotlib.colors as colors
slices = [1441,551,173,161,146]
explode = (0.1, 0.1, 0.1, 0.1, 0.1) # may be a tuple or list
candidates= ["Trump", "Cruz", "Rubio", "Kasich", "Others or uncommitted"]

# create a list of colors based on a built-in colormap (PuRd)
# specify the number of colors based on the slices
cmap = plt.cm.get_cmap('PuRd', len(slices))
slice_colors = []
for i in range(cmap.N):
    rgba = cmap(i)
    slice_colors.append(colors.rgb2hex(rgba))
fig, ax = plt.subplots()
fig.set_figheight(8)
fig.set_figwidth(8)
patches, texts, autotexts = ax.pie(slices,
        colors = reversed(slice_colors), # Puts darkred first
        labels=candidates,
        startangle=-110,
        explode=explode,
        shadow= False,
        autopct='%1.2f%%', labeldistance=1.05, pctdistance=0.87)
autotexts[0].set_color('white') # You may change the color of
                                # a certain percent label for contrast and
                                # readability
# You may also create a doughnut using 2 pie charts
ax.pie([1], radius=0.75, colors=["white"])
plt.show();
```

If plt.figure(figsize(w, h)) does not work when using subplots, you may also use fig.set_size_inches(w, h) to change the size of the figure.



Geographic Data with Seaborn Regplot

- Plotting locations of institutions using dots based on the ***college-scorecard*** dataset.
- The points are placed using latitude and longitude. Notice the faint shape of the U.S.
- Here, I attempt to size the points by the institutional SAT score average: score/1600.




```
# Locate the data here: https://collegescorecard.ed.gov/data/
college = pd.read_csv('Most-Recent-Cohorts-All-Data-Elements.csv')

#first drop records where latitude, longitude, and sat_avg is null
clean_data = ...
df = pd.DataFrame()
df["x"] = clean_data["LATITUDE"]
df["y"] = clean_data["LONGITUDE"]
df["size"] = clean_data["SAT_AVG"]

import seaborn as sns
plt.style.use('dark_background')
plt.figure(figsize=(12,10))
locations = sns.regplot('x', 'y',
                        data=df,
                        fit_reg=False, scatter_kws={'s': np.true_divide(df["size"], 1600), "color":
"violet"})
locations.set(title="Locations of Institutions", xticks=[], yticks=[], xlabel="", ylabel="")
sns.despine(left=True, bottom=True) # Remove the spines from the plot
# spines are lines connecting the axis tick marks and noting the boundaries of the data area
```

Practice Exercise 1

- Create a stacked plot of the following data

2008 births by race and Hispanic Origin			
	White	Black	Hispanic
Single	2,184,914	599,536	1,017,139
Twin	82,903	22,924	23,266
Multiple	4,493	569	834

Practice Exercise 2

- Create a stacked plot of the following data

2008 Deliveries by Race and Hispanic Origin (in thousands)			
	White	Black	Hispanic
Vaginal	1,527	406	717
Caesarean	733	214	322
Not stated	8	2	3

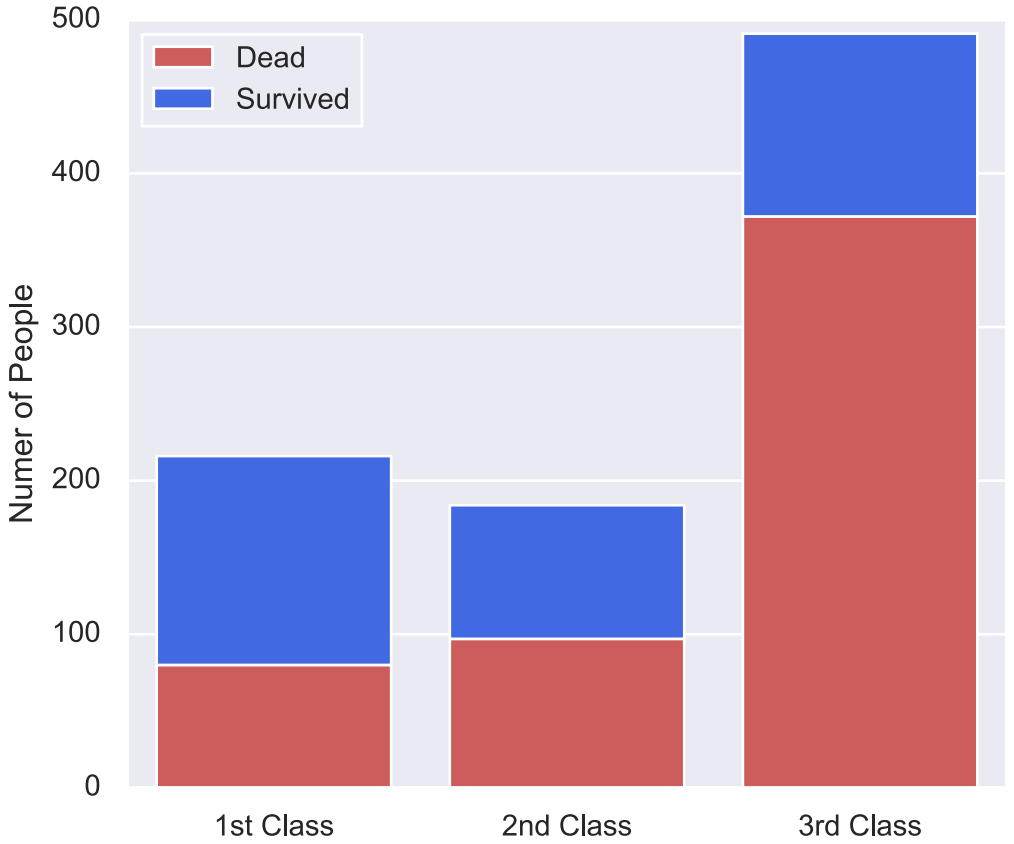
Practice Exercise 3

- Given the following titanic dataset from seaborn, create the following ***bar plot*** (plt.bar) using matplotlib

```
titanic = sns.load_dataset('titanic')
```

survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone	
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
886	0	2	male	27.0	0	0	13.0000	S	Second	man	True	NaN	Southampton	no	True
887	1	1	female	19.0	0	0	30.0000	S	First	woman	False	B	Southampton	yes	True
888	0	3	female	NaN	1	2	23.4500	S	Third	woman	False	NaN	Southampton	no	False
889	1	1	male	26.0	0	0	30.0000	C	First	man	True	C	Cherbourg	yes	True
890	0	3	male	32.0	0	0	7.7500	Q	Third	man	True	NaN	Queenstown	no	True

891 rows × 15 columns



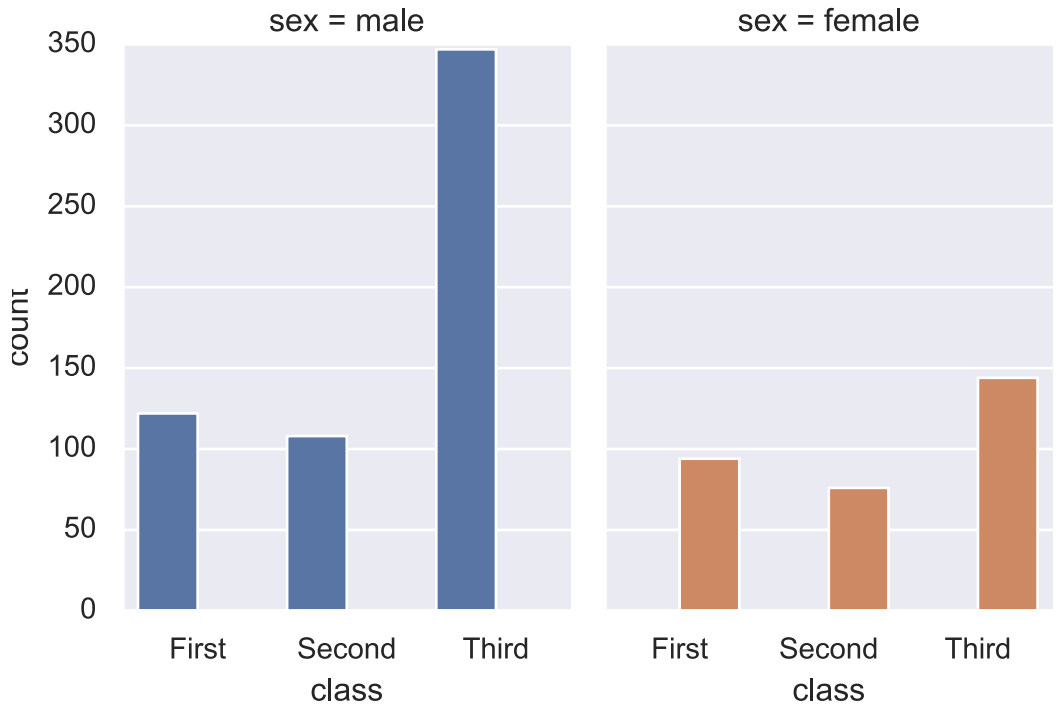
Practice Exercise 4

- Given the following titanic dataset from seaborn, create the following **catplot** using seaborn

```
titanic = sns.load_dataset('titanic')
```

survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone	
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
886	0	2	male	27.0	0	0	13.0000	S	Second	man	True	NaN	Southampton	no	True
887	1	1	female	19.0	0	0	30.0000	S	First	woman	False	B	Southampton	yes	True
888	0	3	female	NaN	1	2	23.4500	S	Third	woman	False	NaN	Southampton	no	False
889	1	1	male	26.0	0	0	30.0000	C	First	man	True	C	Cherbourg	yes	True
890	0	3	male	32.0	0	0	7.7500	Q	Third	man	True	NaN	Queenstown	no	True

891 rows × 15 columns



Other Visualization Packages

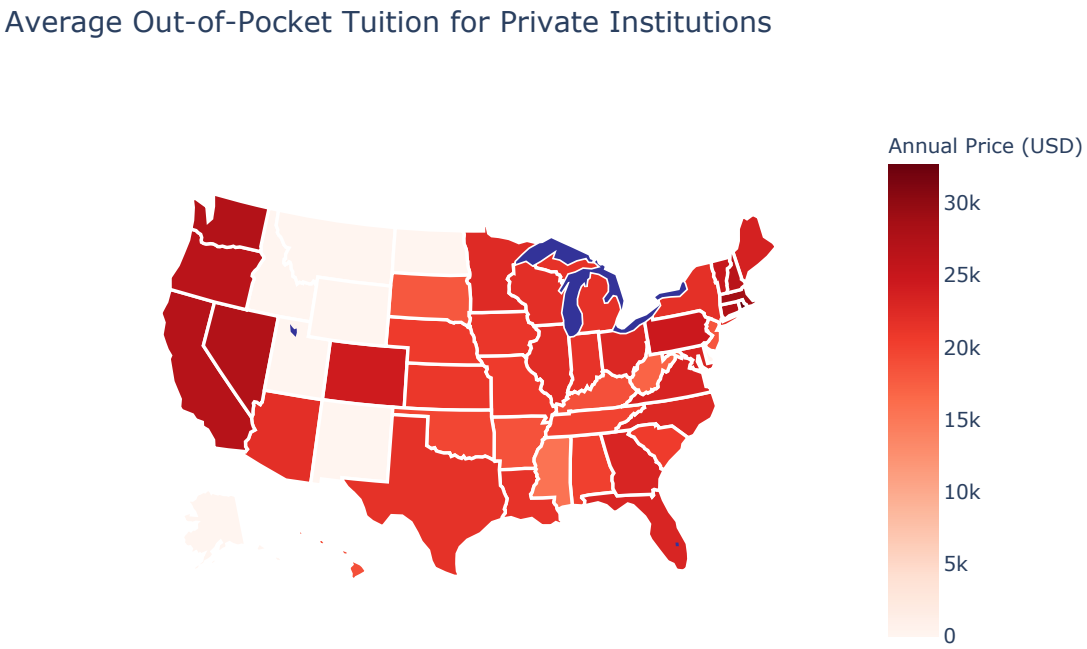
- **Bokeh** is a JavaScript visualization library with a Python frontend that creates highly interactive visualizations capable of handling very large and/or streaming datasets. The Python frontend outputs a JSON data structure that can be interpreted by the Bokeh JS engine. (https://www.datacamp.com/cheat-sheet/python-data-visualization-bokeh-cheat-sheet)
- **Plotly** is the eponymous open source product of the Plotly company, and is similar in spirit to Bokeh. Because Plotly is the main product of a startup, it is receiving a high level of development effort.
- **Vispy** is an actively developed project focused on dynamic visualizations of very large datasets. Because it is built to target OpenGL and make use of efficient graphics processors in your computer, it is able to render some quite large and stunning visualizations.
- **Vega** and **Vega-Lite** are declarative graphics representations, and are the product of years of research into the fundamental language of data visualization. The reference rendering implementation is JavaScript, but the API is language agnostic. There is a Python API under development in the Altair package

pip install plotly==4.6.0

5B.4 INTRO TO PLOTLY

Geographic Data with Plotly - Choropleth

- A **Choropleth Map** is a map composed of colored polygons. It is used to represent spatial variations of a quantity
- Let's plot the average out-of-pocket tuition for private institutions based on the college-scorecard dataset that we used on the previous slide
- Learn more about Choropleth Maps here: https://plotly.com/python/choropleth-maps/
- The code for generating the avg tuition Choropleth follows on the next slides.



Geographic Data with Plotly – Choropleth (Cont'd)

- First let us extract data summaries

```
college = pd.read_csv('Most-Recent-Cohorts-All-Data-Elements.csv')

cols=['STABBR', 'NPT4_PRIV', 'PREDDEG', 'CONTROL']
df_map = college[cols]

# Extract state names
states = df_map['STABBR'].value_counts().index.tolist()

# Select (1) private, (2) predominantly bachelors awarding inst.
df_map = df_map[(df_map['CONTROL'] != 1) & (df_map['PREDDEG'] == 3)].dropna()

# identify those states with at least 5 private schools
gt_five = sum(df_map['STABBR'].value_counts() > 5)
gt_five_states = df_map['STABBR'].value_counts().index.tolist()[:gt_five]

# Find the average out of pocket tuition for each state
df_map_summary = df_map.groupby(['STABBR'], as_index=False)['NPT4_PRIV'].mean()

# Disqualify states with less than 5 private schools
df_map_summary['NPT4_PRIV'][~df_map_summary['STABBR'].isin(gt_five_states)] = 0

# Add another value for Wyoming (originally a missing value)
df_map_summary.loc[-1] = ['WY', 0]

# Learn more about the variables in the dataset by finding the data dictionary here:
https://collegescorecard.ed.gov/data/documentation/

# Some code obtained from https://aidaylanan.github.io/college-affordability-2.html
# with modifications to support the updated plotly express (particularly the next slide)
```

Locate the data here: https://collegescorecard.ed.gov/data/

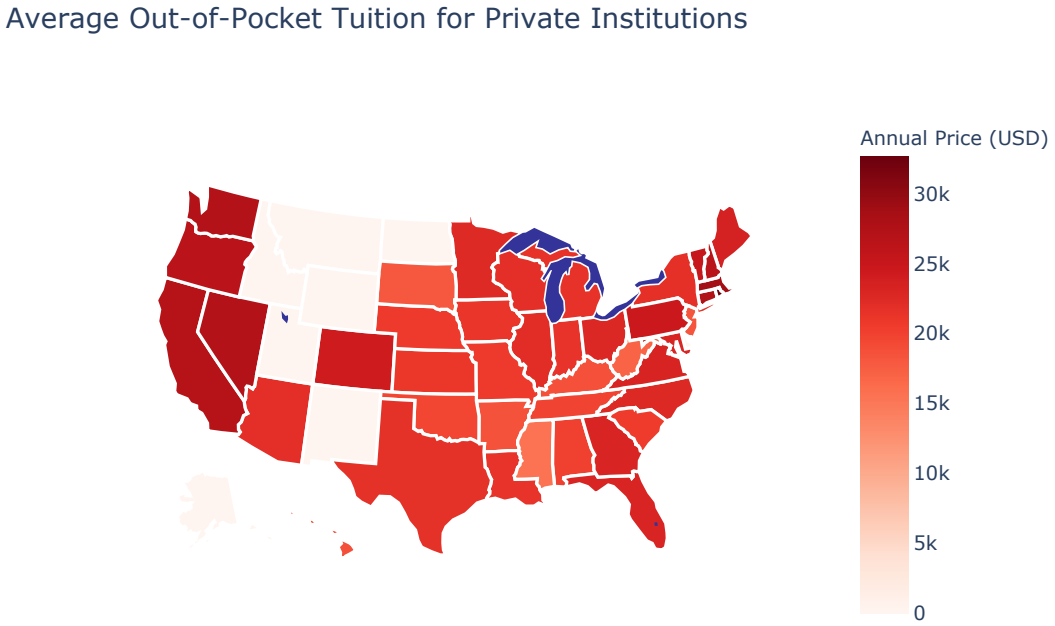
Tips about key fields

STABBR – State abbreviation
NPT4_PRIV -- Average net price for Title IV institutions (private for-profit and nonprofit institutions)
CONTROL --- is ownership of institution. A value of 1 means it is public
PREDDEG -- Predominant undergraduate degree awarded
0 Not classified
1 Predominantly certificate-degree granting
2 Predominantly associate's-degree granting
3 Predominantly bachelor's-degree granting
4 Entirely graduate-degree granting

Geographic Data with Plotly – Choropleth (Cont'd)

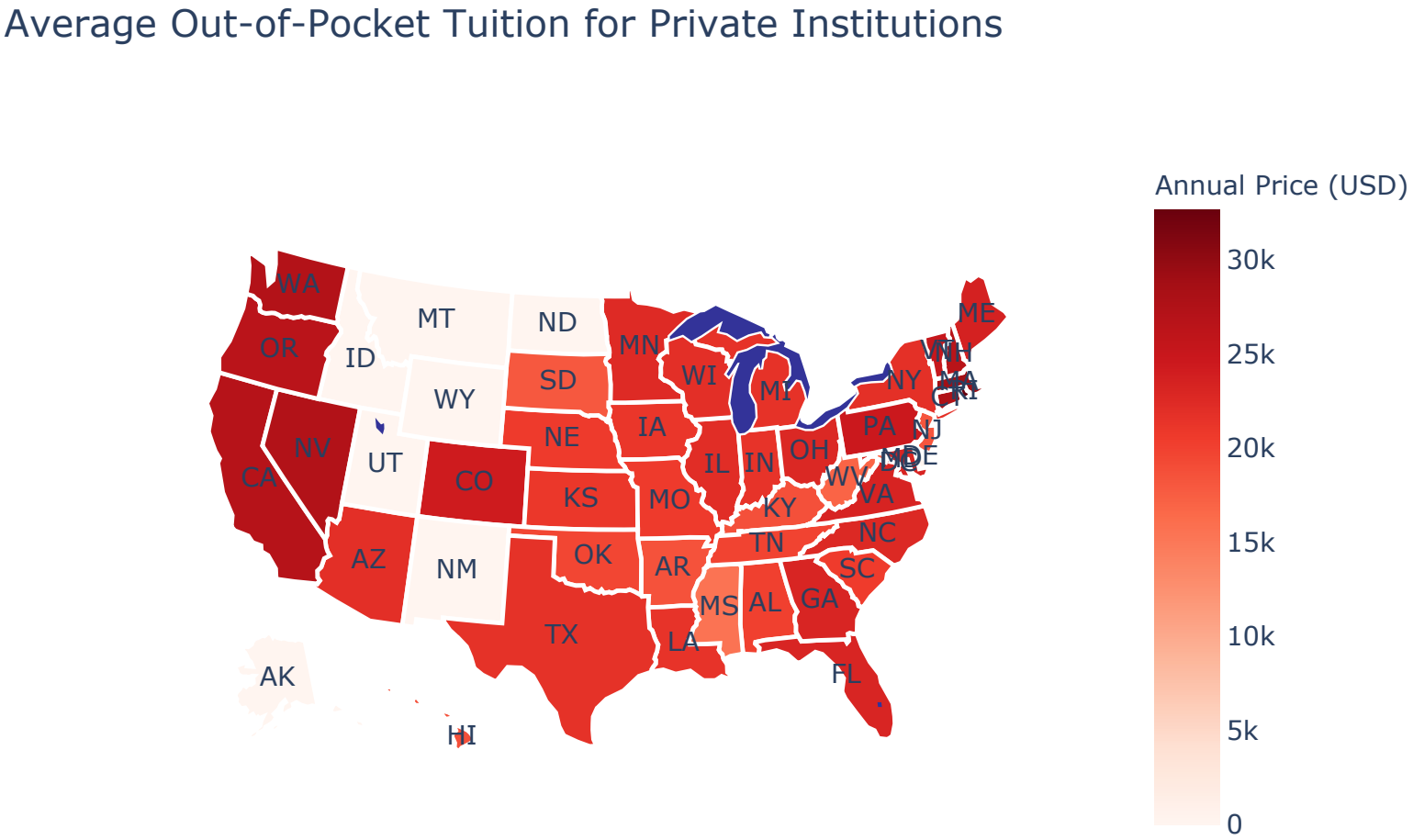
- Now let's use ***Plotly*** express to display the Choropleth
- Install plotly: `pip install plotly==4.6.0`

```
import plotly.express as px
import plotly.graph_objects as go

fig = go.Figure(data=go.Choropleth(
    locations = df_map_summary['STABBR'], # Spatial coordinates
    z = df_map_summary['NPT4_PRIV'].astype(float), # the data to show (color-coded)
    # set of locations: ISO-3, USA-states, country names. E.g. ISO-3: USA, CHN, CAN
    locationmode = 'USA-states',
    colorscale = 'Reds',
    colorbar_title = "Annual Price (USD)",
    marker = dict(
            line = dict (
                color = 'rgb(255,255,255)',
                width = 2
            ) )
))

fig.update_layout(
    title_text = 'Average Out-of-Pocket Tuition for Private Institutions',
    geo_scope='usa', # limit map scope to USA,
    geo = dict(
            scope='usa',
            projection=dict( type='albers usa' ),
            showlakes = True,
            lakecolor = 'rgb(51, 51, 153)'),

)

fig.show()
```



Plotly Choropleth Exercise

- Research how to add annotation to a Plotly Choropleth and generate the following Chart



The End