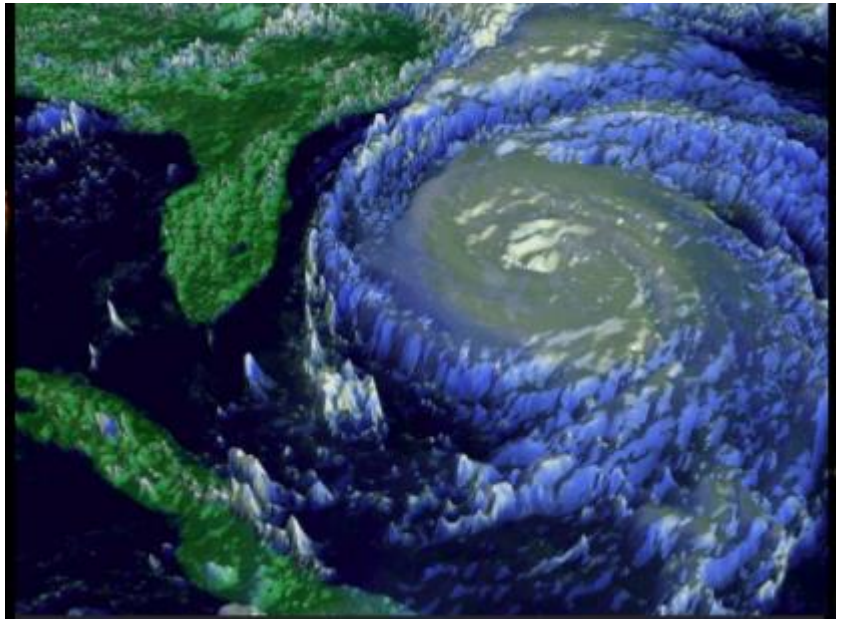
Data Visualization

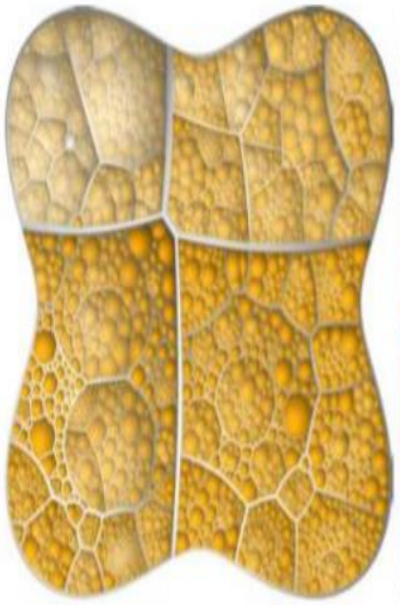
Goals

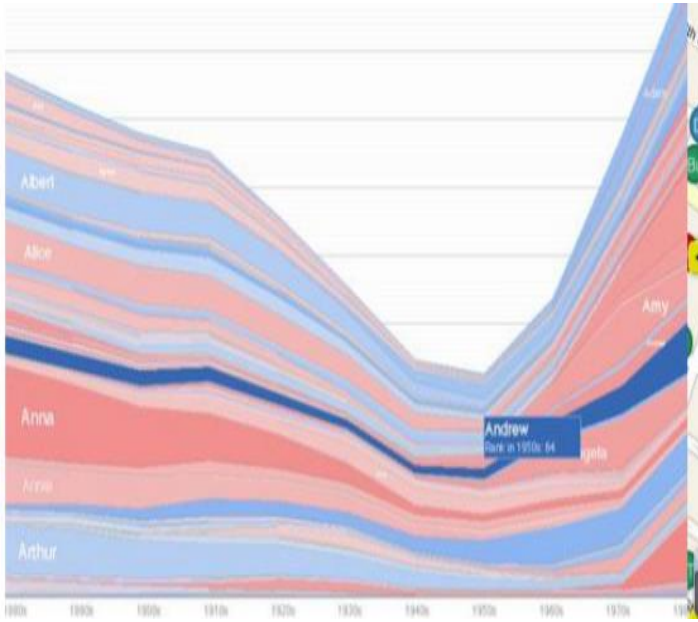
- Understand what makes a visualization effective through the study of core principles
- Critically evaluate a visual representation of data by looking at various examples in media (newspapers, television and so on)
- Gain hands-on experience with visualization tools
- Incorporate visualization principles to build an interactive visualization of your own data

What is Data Visualization?

- Visual Representation of Data
- For exploration, discovery, insight, ..
- Interactive component provides more insight as compared to a static image







Types of Data Visualization

- **Scientific visualization, information visualization, and visual analytics are often seen as the three main branches of visualization.**
- Scientific Visualization
  - – Structural Data – Seismic, Medical, ..
- Information Visualization
  - – No inherent structure – News, stock market, top grossing movies, Facebook connections
- Visual Analytics
  - – Use visualization to understand and synthesize large amounts of multimodal data – audio, video, text, images, networks of people ..

Five Principles for Good Data Visualization

1. **Good data visualization is informative**

- Well presented data forms the backbone of a compelling story
- It has the power to strengthen and illuminate a narrative
- Improving understanding and focusing on what’s important

2. Good data visualization is well balanced

- Communicating quantitative data effectively requires the right balance of components
- Color is used with purpose and is not distracting
- All parts are labeled and include a legend when necessary
- The scale of the visualization must be immediately identifiable
- The standard lexicon of graphs are often all that is required (do not use pie charts)

3. Good data visualization is equally concerned with what is not displayed

People are easily overwhelmed with extraneous details

Simplify and reduce what is being presented to what is essential

4. Good data visualization is created with pure data

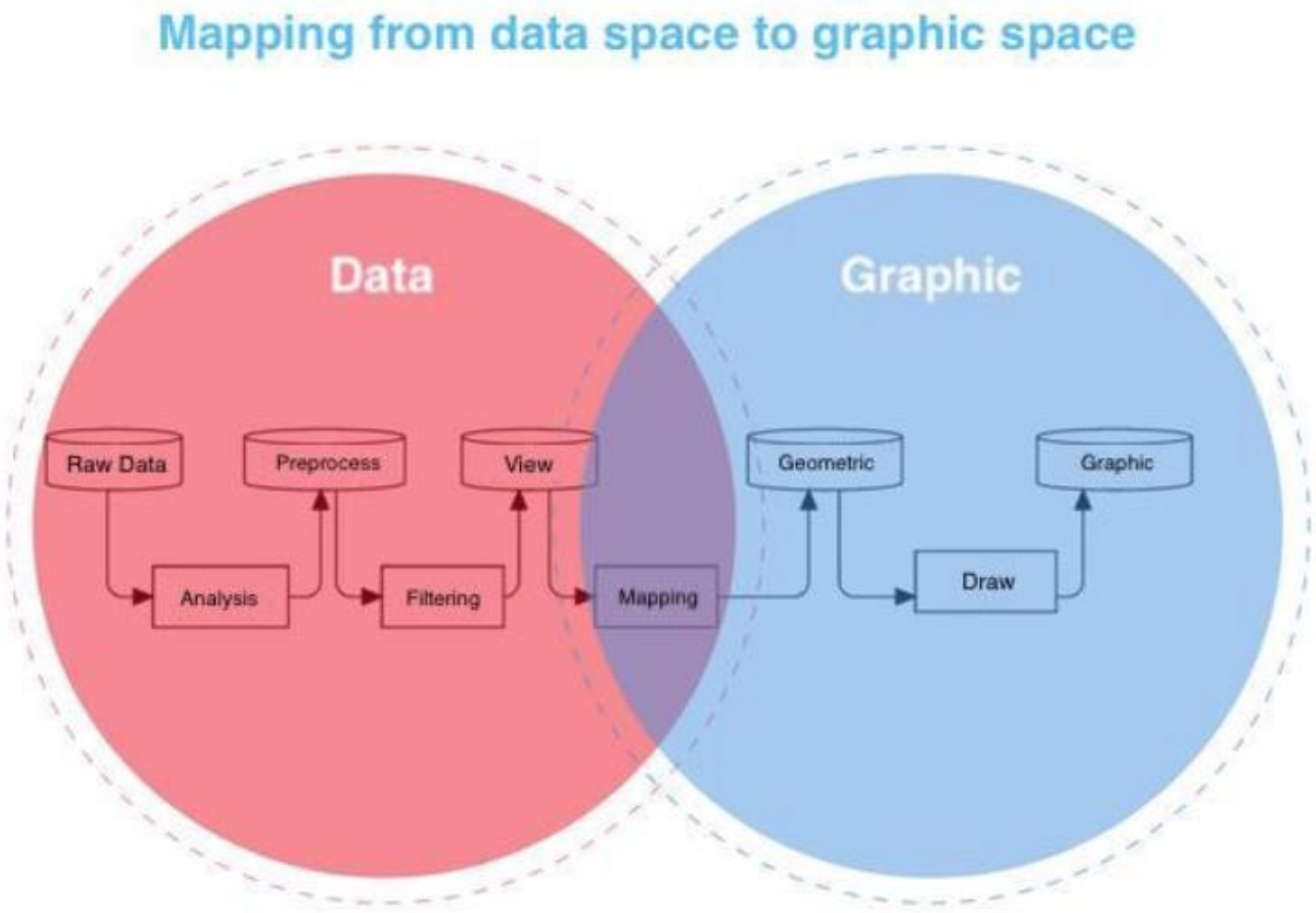
- Avoid utilizing muddy or incomplete sources of data
- Misleading the audience with false information or lack of clarity is in poor taste
- Ultimately, good data visualization enables better decisions and actions

5. Good data visualization is human

- Parsing large data quantities of data is beyond human perception
- The goal with any kind of data visualization is to augment and improve human perception
- Just like a microscope it allows us to explore data within the realm of our understanding

How to achieve data visualization?

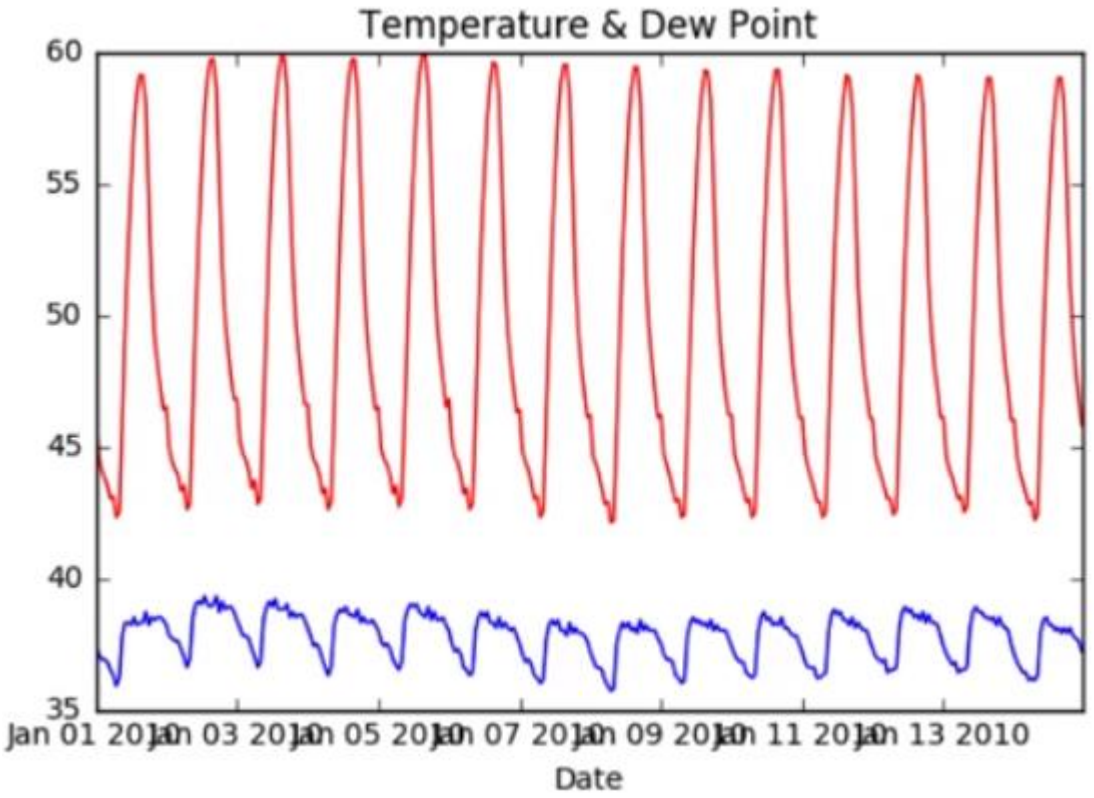
- Technically, the simplest understanding of data visualization is the mapping from data space to graphic space



Using Matplotlib

- Graphs on common axes

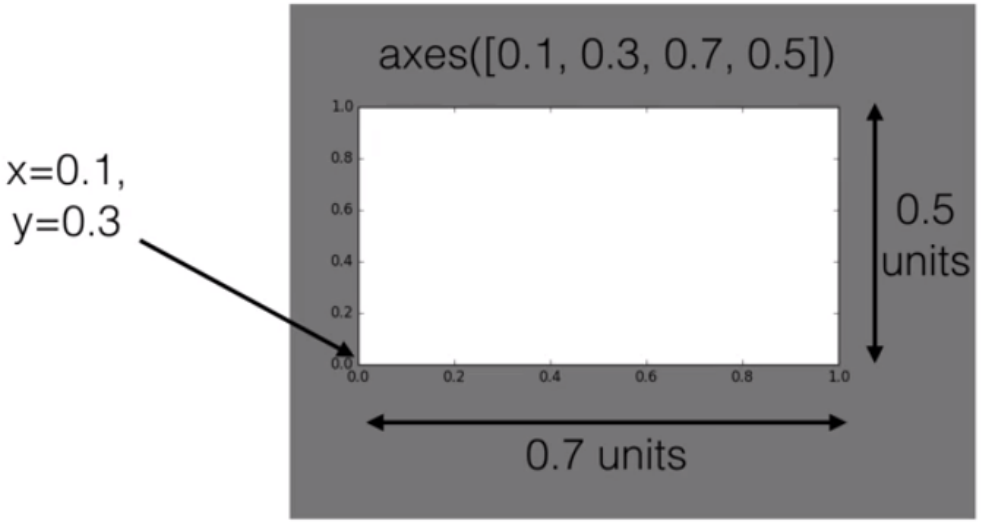
```
In [1] : import matplotlib.pyplot as plt
In [2] : plt.plot(t, temperature, ’r’)
In [3] : plt.plot(t, dewpoint, ’b’)
In [4] : plt.xlabel(‘date’)
In [5] : plt.show()
```



Using axes()

```
In [1]: plt.axes([0.05,0.05,0.425,0.9])

In [2]: plt.plot(t, temperature, 'r')

In [3]: plt.xlabel('Date')

In [4]: plt.title('Temperature')

In [5]: plt.axes([0.05,0.05,0.425,0.9])

In [6]: plt.plot(t, dewpoint, 'b')

In [7]: plt.xlabel('Date')

In [8]: plt.title('Dew Point')

In [9]: plt.show()
```

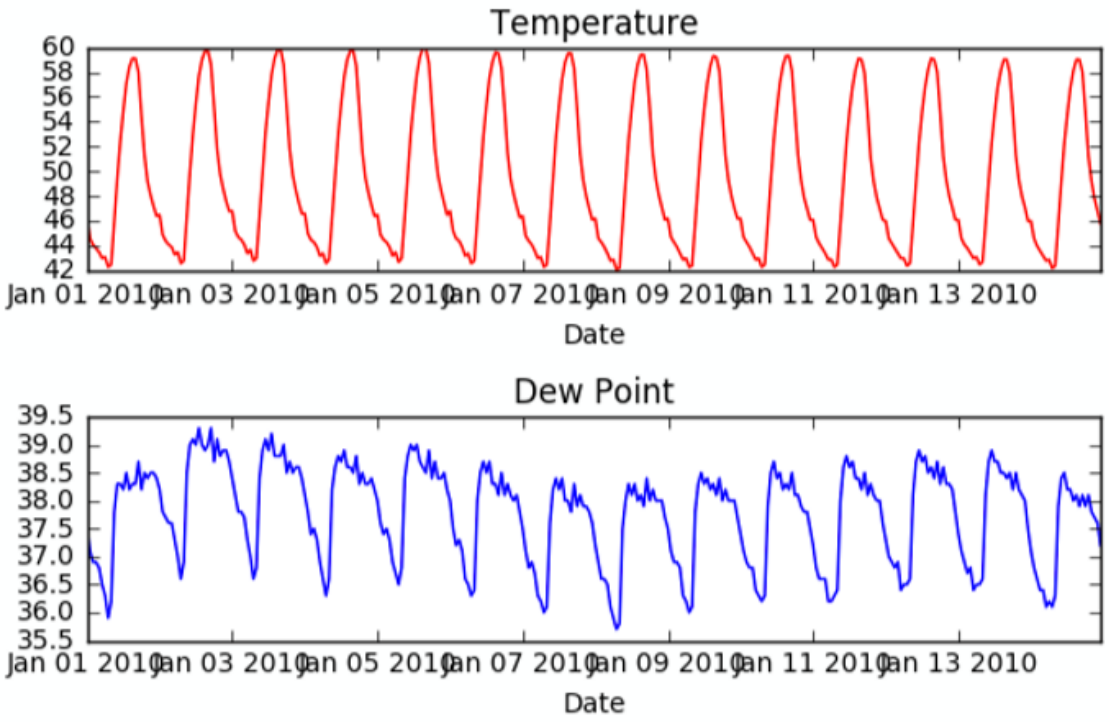
- Syntax: axes([x_lo, y_lo, width, height])
- Units between 0 and 1 (figure dimensions)



Using subplot()

- Syntax: subplot(nrows, ncols, nsubplot)
- Subplot ordering:
  - Row-wise from top left
  - Indexed from 1

```
In [1]: plt.subplot(2, 1, 1)

In [2]: plt.plot(t, temperature, 'r')

In [3]: plt.xlabel('Date')

In [4]: plt.title('Temperature')

In [5]: plt.subplot(2, 1, 2)

In [6]: plt.plot(t, dewpoint, 'b')

In [7]: plt.xlabel('Date')

In [8]: plt.title('Dew Point')

In [9]: plt.tight_layout()

In [10]: plt.show()
```



Controlling axis extents

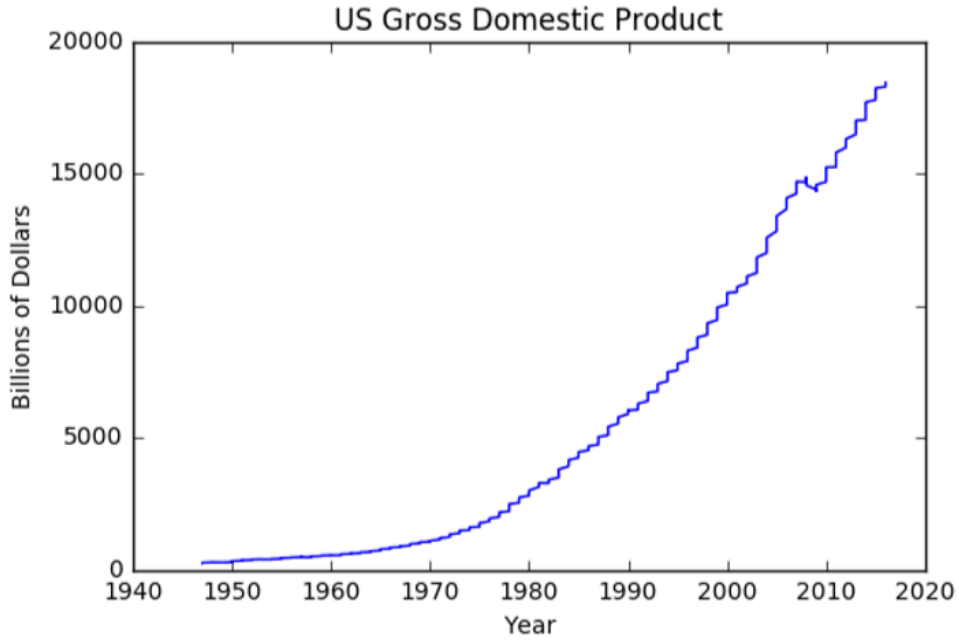
- axis([xmin, xmax, ymin, ymax]) sets axis extents
- Control over individual axis extents
  - xlim([xmin, xmax])
  - ylim([ymin, ymax])
- Can use tuples, lists for extents
  - e.g., xlim((-2, 3)) works
  - e.g., xlim([-2, 3]) works also

Example

GDP over time

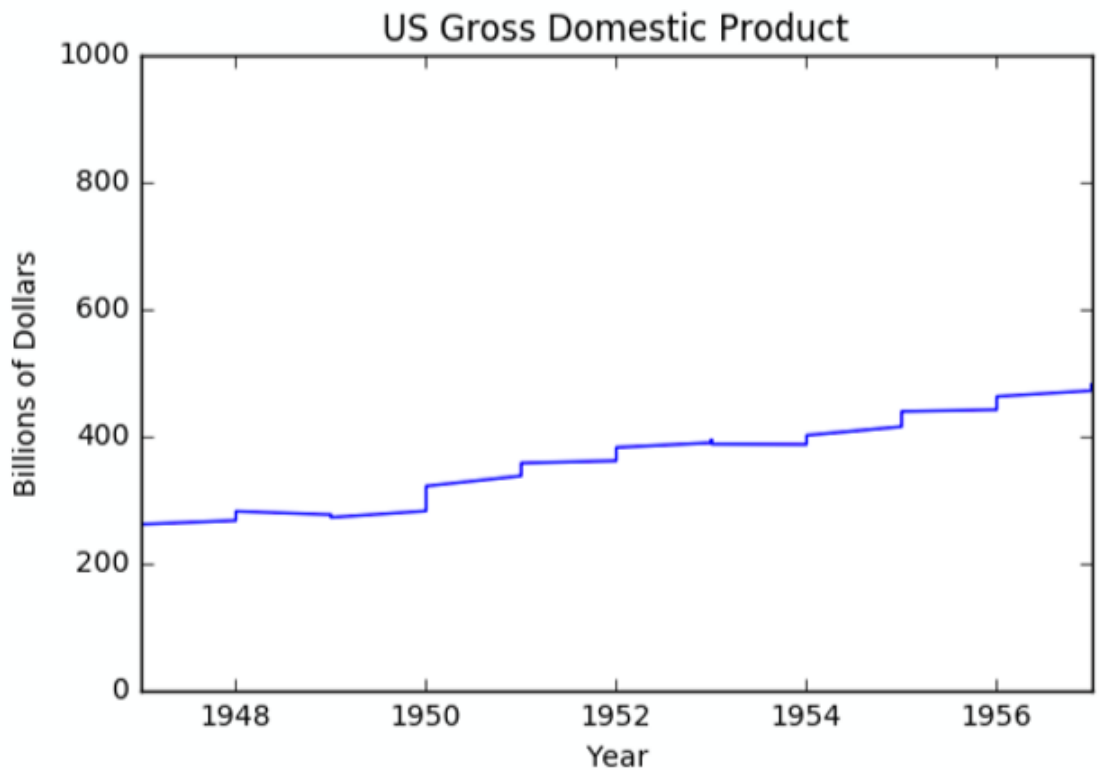
```
In [1]: import matplotlib.pyplot as plt

In [2]: plt.plot(yr, gdp)

In [3]: plt.xlabel('Year')

In [4]: plt.ylabel('Billions of Dollars')

In [5]: plt.title('US Gross Domestic Product')

In [6]: plt.show()
```

GDP over time



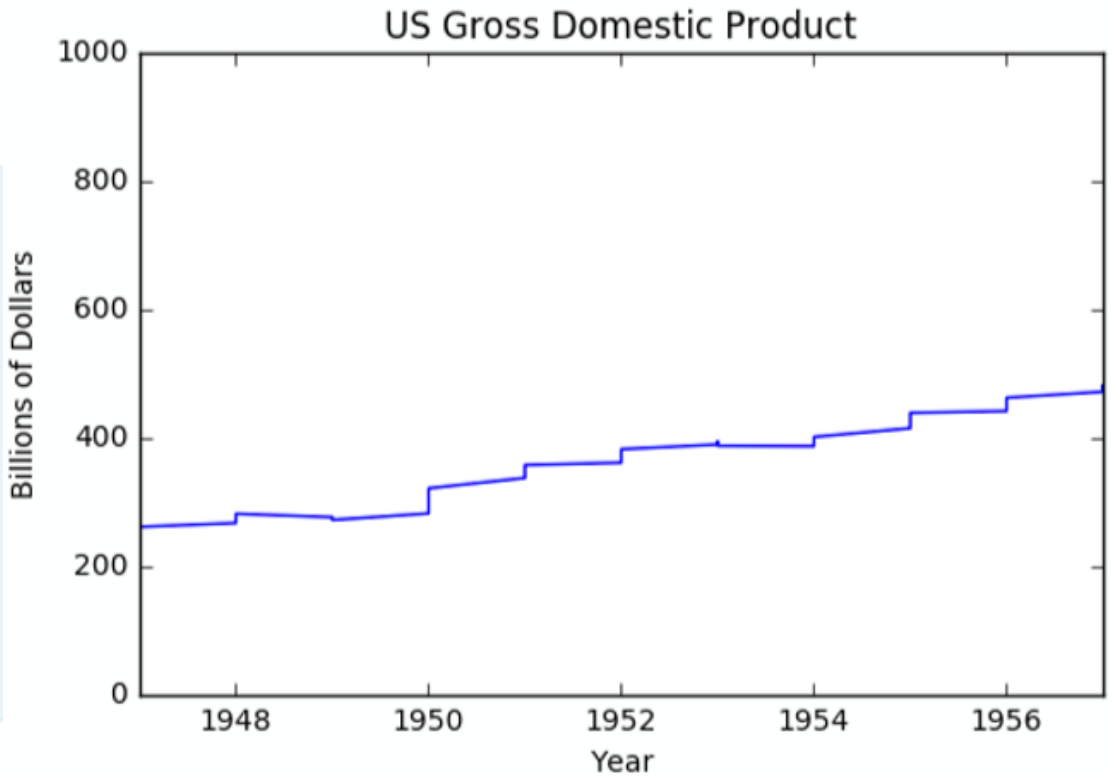
Using xlim() & ylim()

```
In [1]: plt.plot(yr, gdp)

In [2]: plt.xlabel('Year')

In [3]: plt.ylabel('Billions of Dollars')

In [4]: plt.title('US Gross Domestic Product')

In [5]: plt.xlim((1947, 1957))

In [6]: plt.ylim((0, 1000))

In [7]: plt.show()
```



Using axis()

```
In [1]: plt.plot(yr, gdp)

In [2]: plt.xlabel('Year')

In [3]: plt.ylabel('Billions of Dollars')

In [4]: plt.title('US Gross Domestic Product')

In [5]: plt.axis((1947, 1957, 0, 600))

In [6]: plt.show()
```

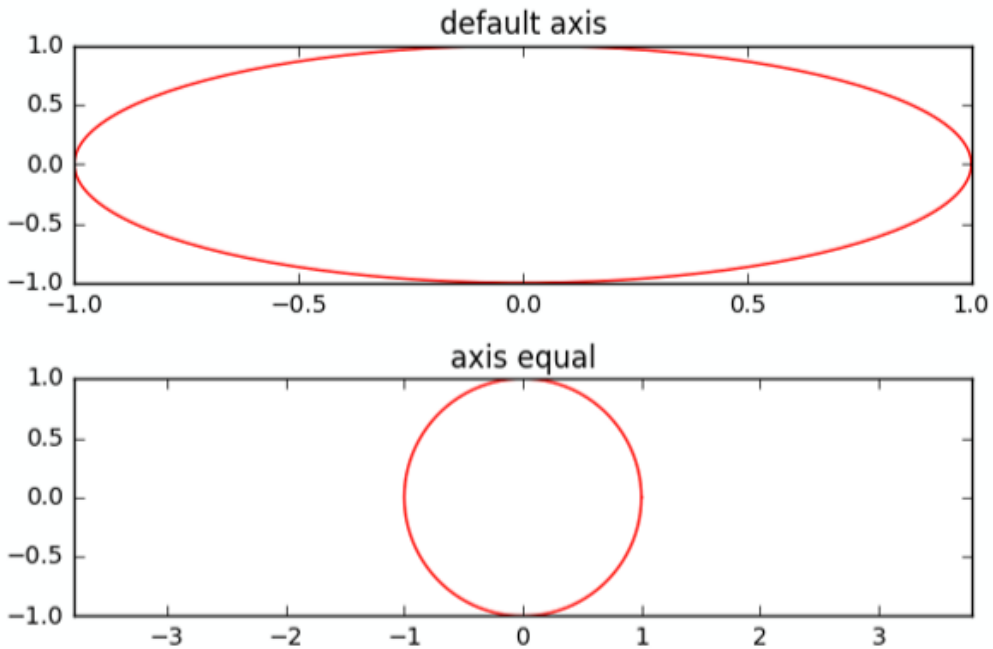


Other axis() options

Invocation	Result
axis('off')	turns off axis lines, labels
axis('equal')	equal scaling on x, y axes
axis('square')	forces square plot
axis('tight')	sets xlim(), ylim() to show all data

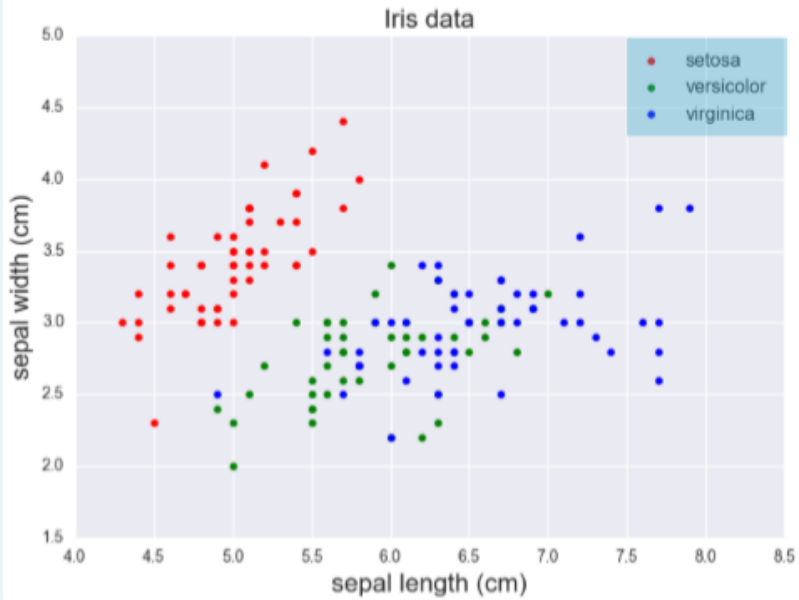
Using axis('equal')

```
In [1]: plt.subplot(2, 1, 1)

In [2]: plt.plot(x, y, 'red')

In [3]: plt.title('default axis')

In [4]: plt.subplot(2, 1, 2)

In [5]: plt.plot(x, y, 'red')

In [6]: plt.axis('equal')

In [7]: plt.title('axis equal')

In [8]: plt.tight_layout()

In [9]: plt.show()
```



Using legend()

```
In [1]: import matplotlib.pyplot as plt

In [2]: plt.scatter(setosa_len, setosa_wid,
   ...:             marker='o', color='red', label='setosa')

In [3]: plt.scatter(versicolor_len, versicolor_wid,
   ...:             marker='o', color='green', label='versicolor')

In [4]: plt.scatter(virginica_len, virginica_wid,
   ...:             marker='o', color='blue', label='virginica')

In [5]: plt.legend(loc='upper right')

In [6]: plt.title('Iris data')

In [7]: plt.xlabel('sepal length (cm)')

In [8]: plt.ylabel('sepal width (cm)')

In [9]: plt.show()
```



Legend

Legend locations

string	code	string	code	string	code
'upper left'	2	'upper center'	9	'upper right'	1
'center left'	6	'center'	10	'center right'	7
'lower left'	3	'lower center'	8	'lower right'	4
'best'	0			'right'	5

Plot annotations

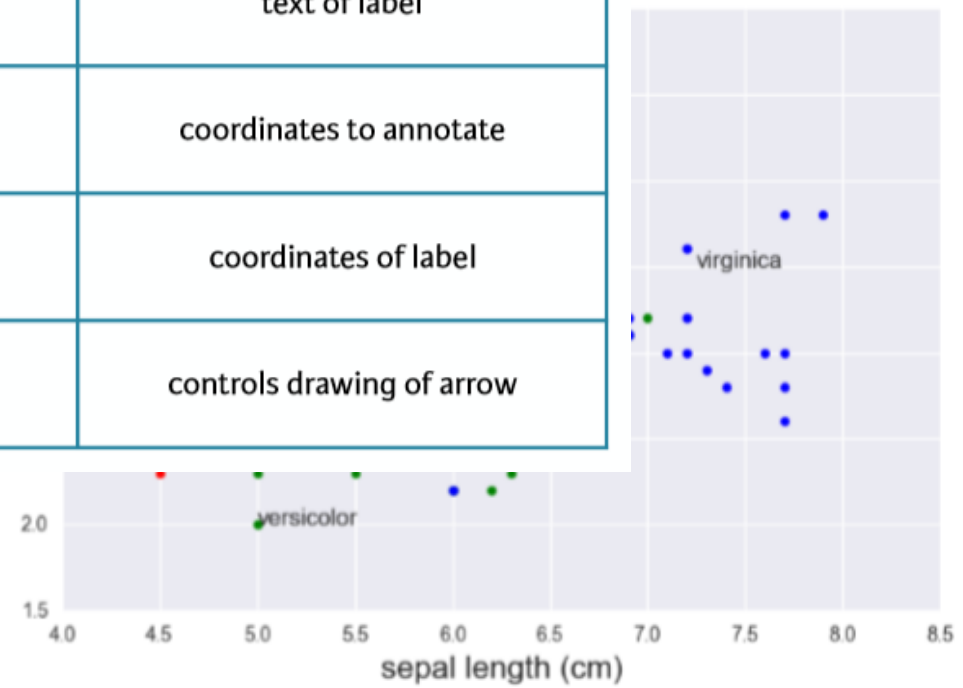
- Text labels and arrows using annotate() method
- Flexible specification of coordinates
- Keyword arrowprops: dict of arrow properties
  - width
  - color
  - etc.

Using annotate() for text

```
In [1]: plt.annotate('setosa', xy=(5.0, 3.5))

In [2]: plt.annotate('virgini

In [3]: plt.annotate('versico

In [4]: plt.show()
```

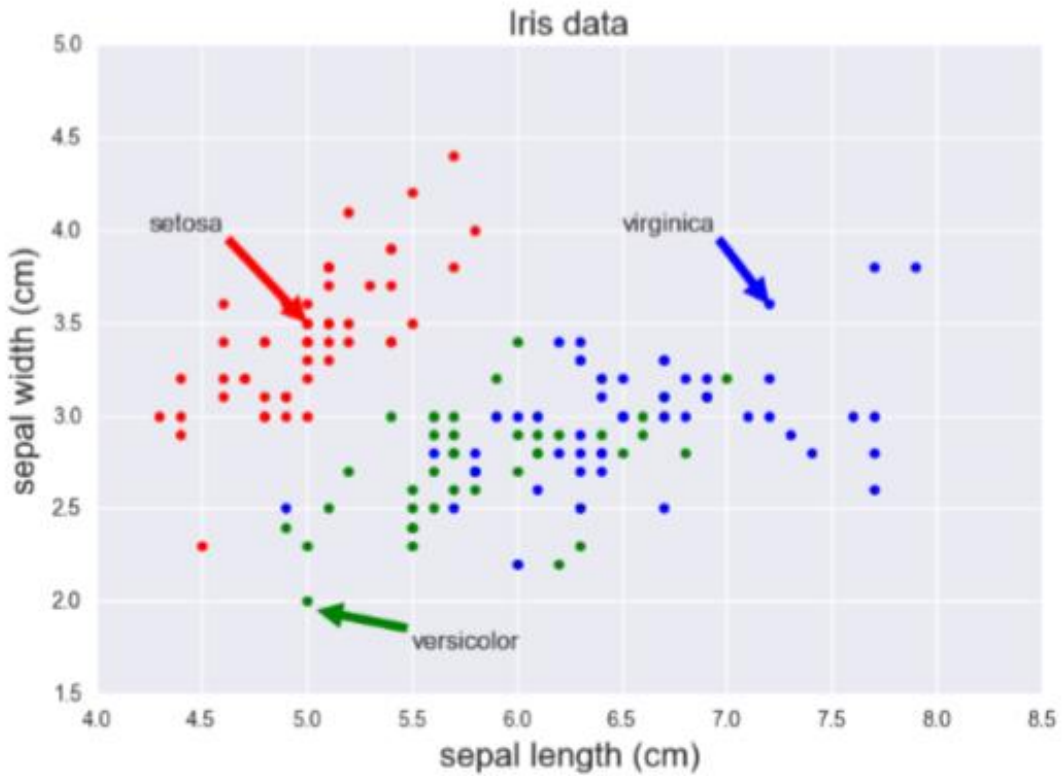
Options for annotate()

option	description
s	text of label
xy	coordinates to annotate
xytext	coordinates of label
arrowprops	controls drawing of arrow



Using annotate() for arrows

```
In [1]: plt.annotate('setosa', xy=(5.0, 3.5),
   ...:              xytext=(4.25, 4.0), arrowprops={'color':'red'})

In [2]: plt.annotate('virginica', xy=(7.2, 3.6),
   ...:              xytext=(6.5, 4.0), arrowprops={'color':'blue'})

In [3]: plt.annotate('versicolor', xy=(5.05, 1.95),
   ...:              xytext=(5.5, 1.75),
   ...:              arrowprops={'color':'green'})

In [4]: plt.show()
```



Working with plot styles

- Style sheets in Matplotlib
- Defaults for lines, points, backgrounds, etc.
- Switch styles globally with plt.style.use()
- plt.style.available: list of styles

Example:

plt.style.use('ggplot')
plt.style.use('fivethirtyeight')