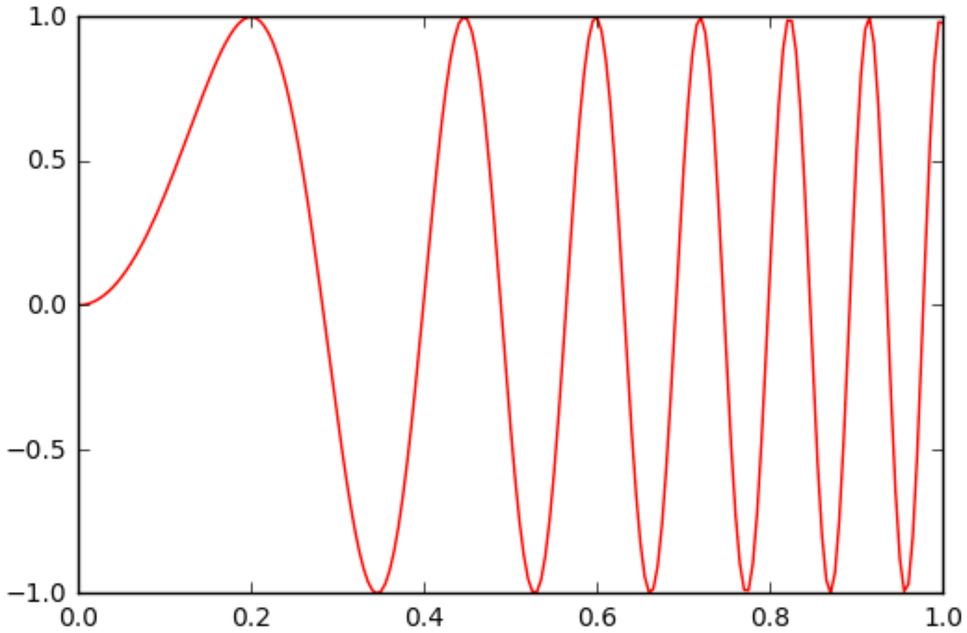
INTRODUCTION TO DATA VISUALIZATION WITH PYTHON

Introduction to Data Visualization with Python

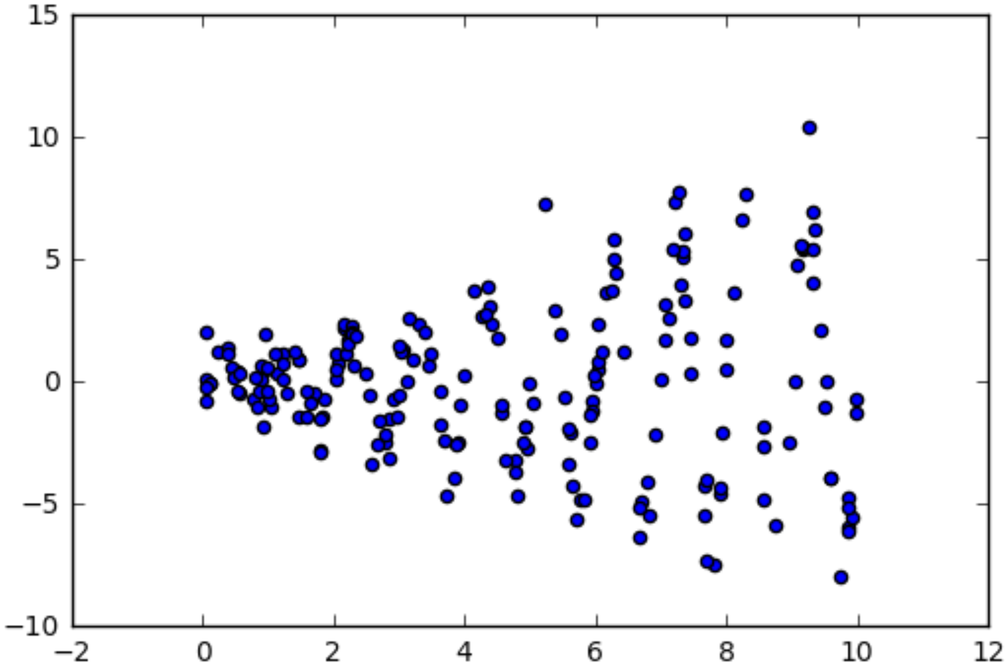
Reminder: Line plots

```
In [1]: import numpy as np

In [2]: import matplotlib.pyplot as plt

In [3]: x = np.linspace(0, 1, 201)

In [4]: y = np.sin((2*np.pi*x)**2)

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

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



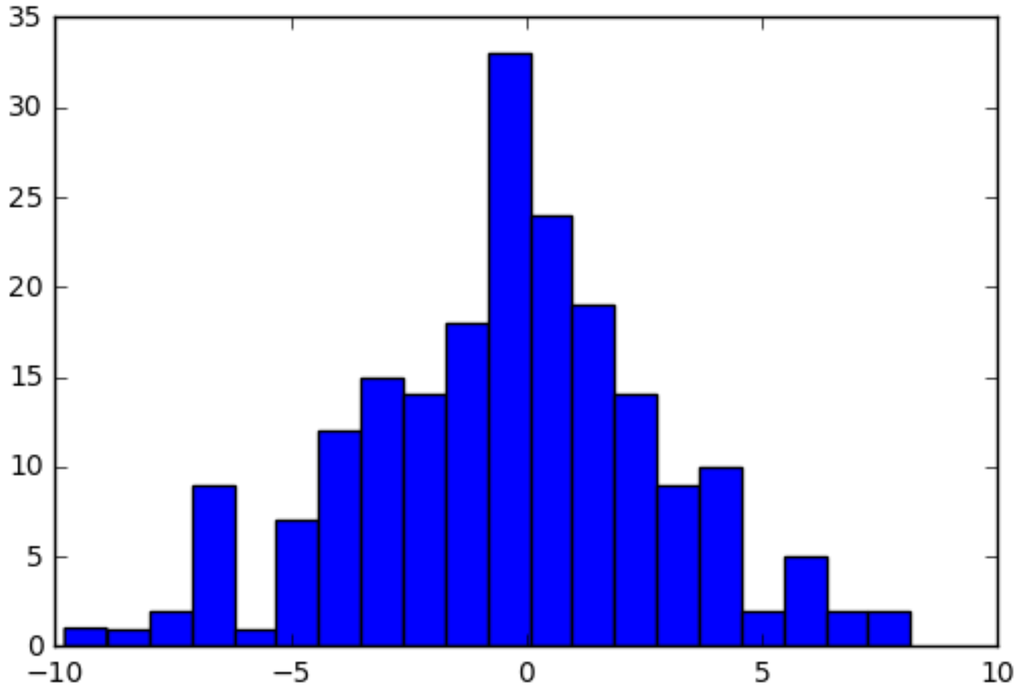
Reminder: Scatter plots

```
In [1]: import numpy as np

In [2]: import matplotlib.pyplot as plt

In [3]: x = 10*np.random.rand(200,1)

In [4]: y = (0.2 + 0.8*x) * np.sin(2*np.pi*x) + \
   ...:     np.random.randn(200,1)

In [5]: plt.scatter(x,y)

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



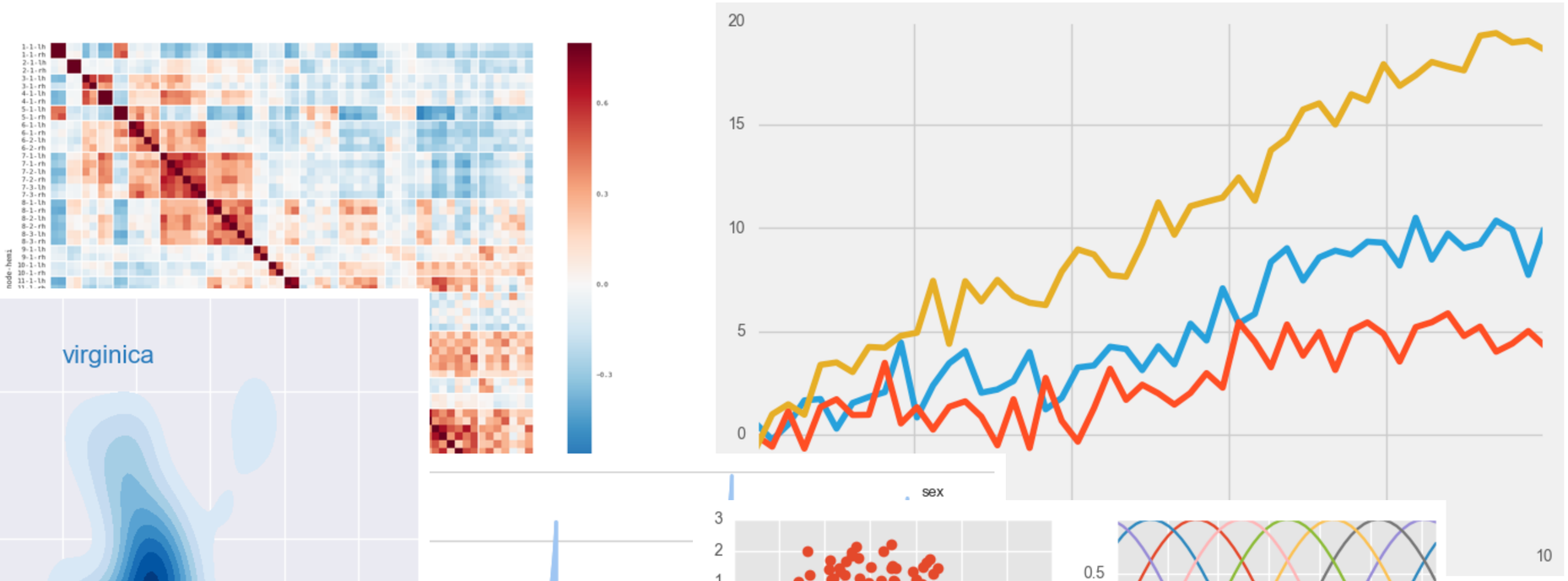
Reminder: Histograms

```
In [1]: import numpy as np

In [2]: import matplotlib.pyplot as plt

In [3]: x = 10*np.random.rand(200,1)

In [4]: y = (0.2 + 0.8*x) * \
   ...:     np.sin(2*np.pi*x) + \
   ...:     np.random.randn(200,1)

In [5]: plt.hist(y, bins=20)

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



What you will learn

- Customizing of plots: axes, annotations, legends
- Overlaying multiple plots and subplots
- Visualizing 2D arrays, 2D data sets
- Working with color maps
- Producing statistical graphics
- Plotting time series
- Working with images


virginica
setosa
sepal_length
sepal_width
sex
Fri
day

INTRODUCTION TO DATA VISUALIZATION WITH PYTHON

See you in the course!

INTRODUCTION TO DATA VISUALIZATION WITH PYTHON

Plotting multiple graphs

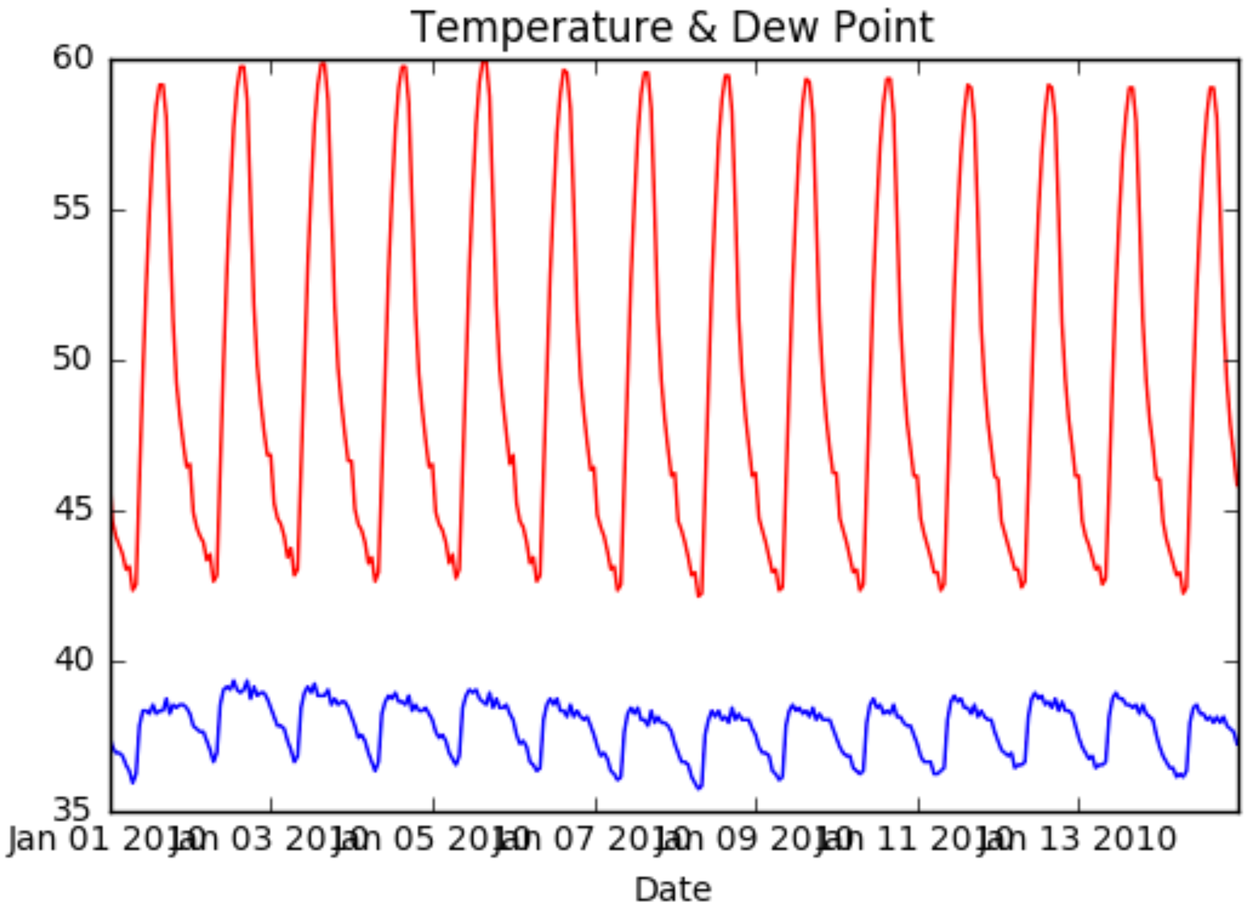
Strategies

- Plotting many graphs on common axes
- Creating axes within a figure
- Creating subplots within a figure

Graphs on common axes

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

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

In [3]: plt.plot(t, dewpoint, 'blue') # Appears on same axes

In [4]: plt.xlabel('Date')

In [5]: plt.title('Temperature & Dew Point')

In [6]: plt.show() # Renders plot objects to screen
```

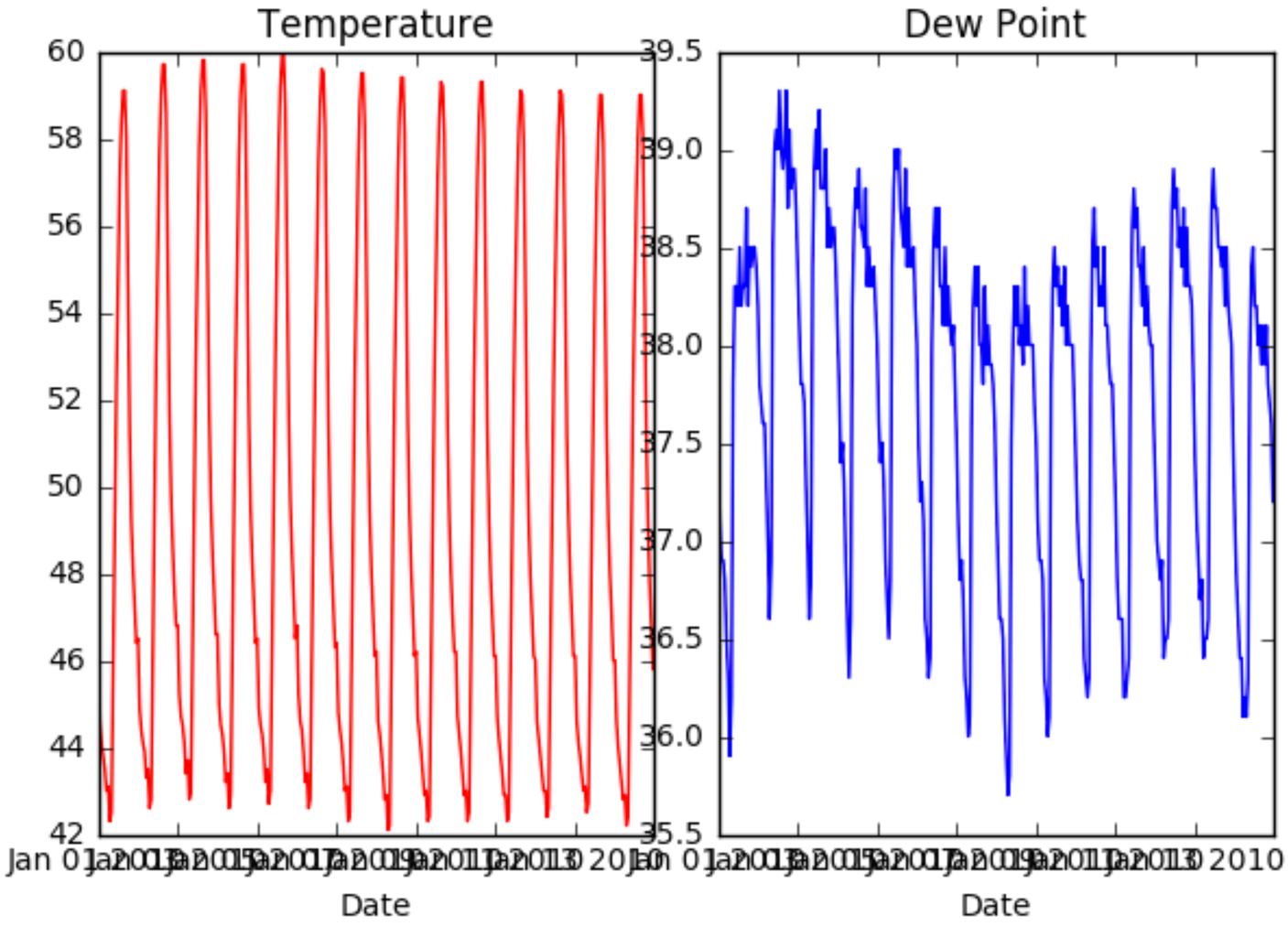
Graphs on common axes



Using axes()

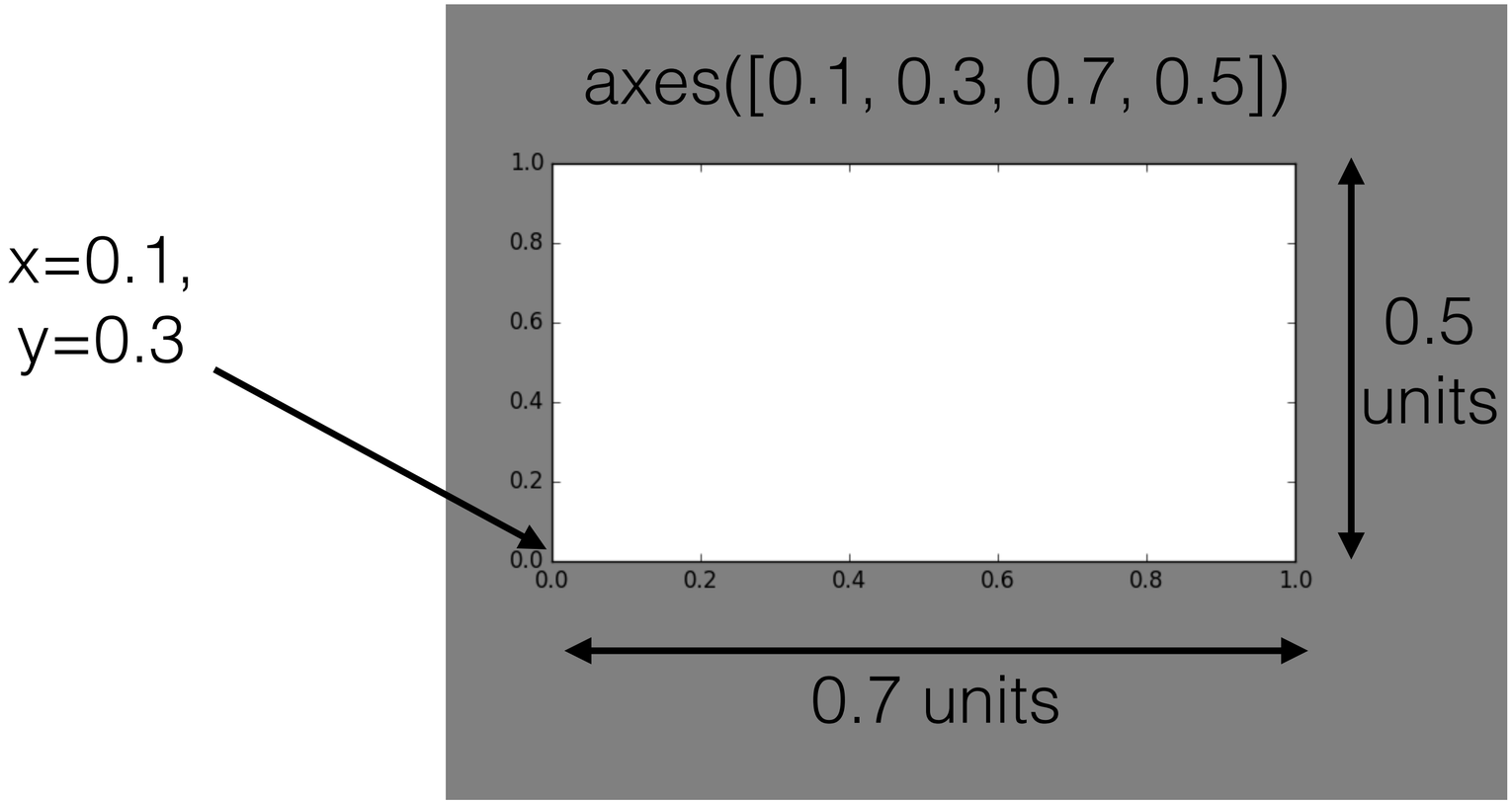
```
In [1]: plt.axes([0.05,0.05,0.425,0.9])

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

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

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

In [5]: plt.axes([0.525,0.05,0.425,0.9])

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

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

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

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

Using axes()

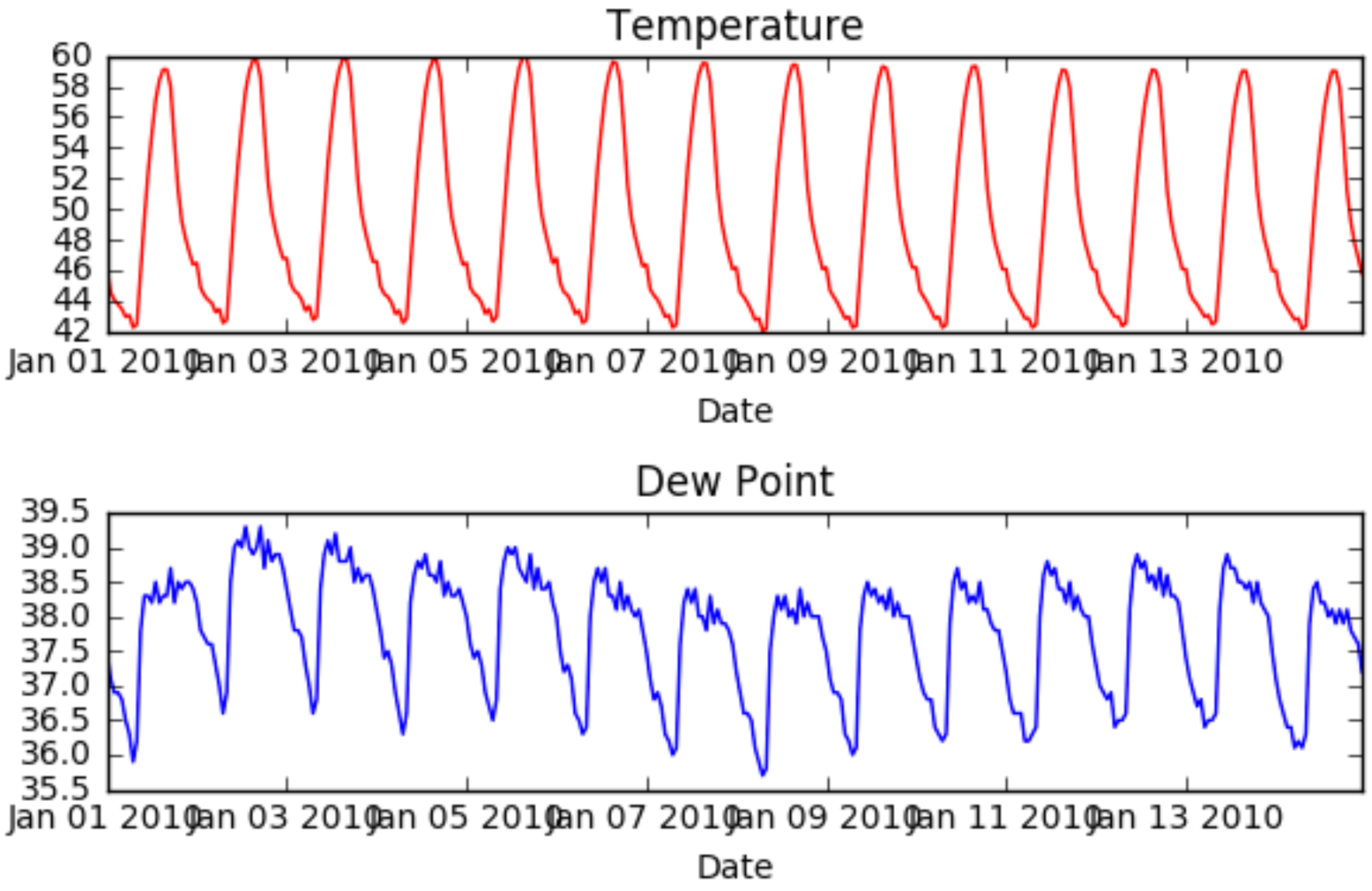

Temperature
60
58
56
54
52
50
48
46
44
42
Jan 01 2010
Date
Dew Point
39.5
39.0
38.5
38.0
37.5
37.0
36.5
36.0
35.5
Jan 01 2010
Date

The axes() command

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



Using subplot()



Using subplot()

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

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

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

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

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

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

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

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

In [9]: plt.tight_layout()

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

The subplot() command

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

INTRODUCTION TO DATA VISUALIZATION WITH PYTHON

Let’s practice!

INTRODUCTION TO DATA VISUALIZATION WITH PYTHON

Customizing axes

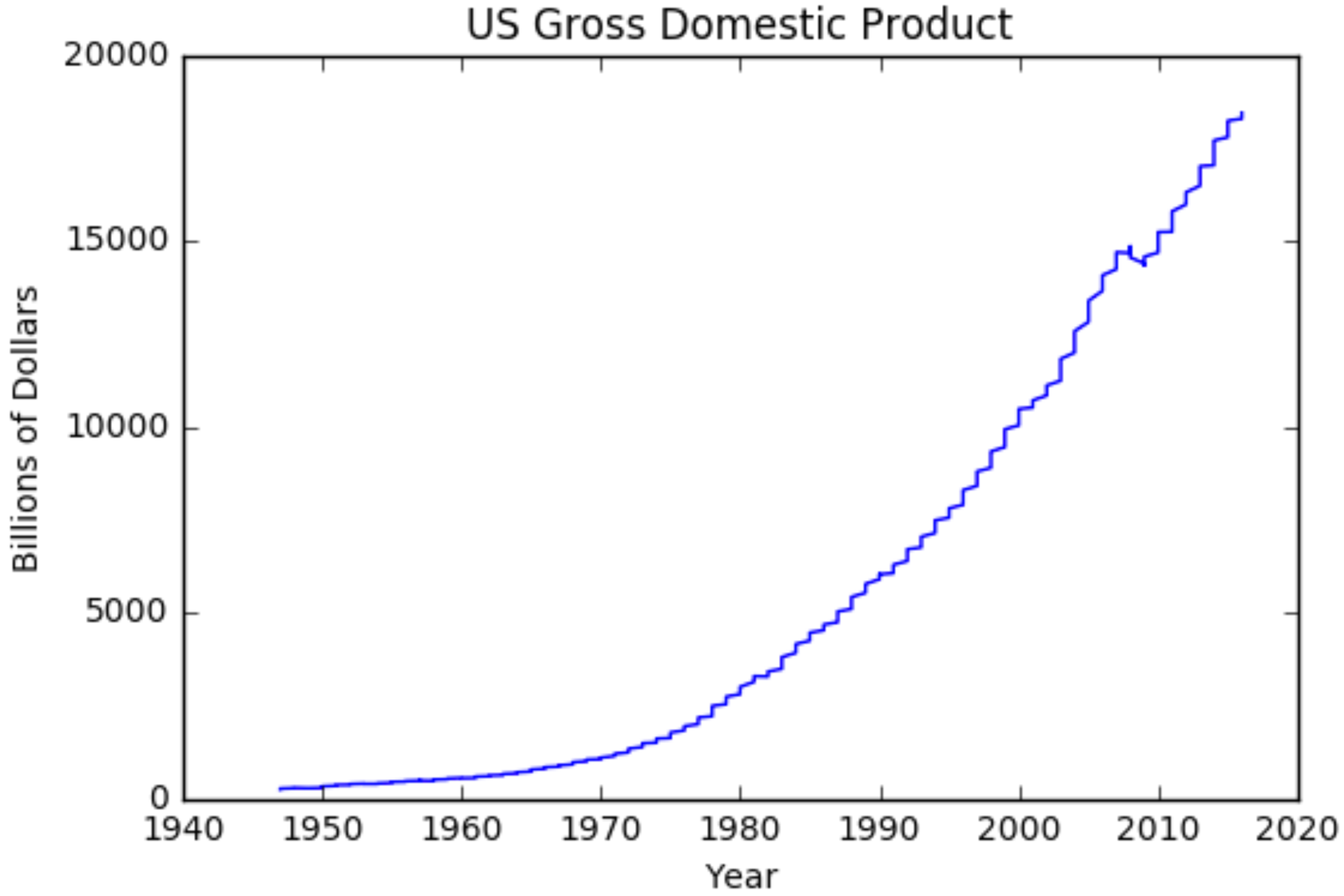
Controlling axis extents

- 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

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()
```

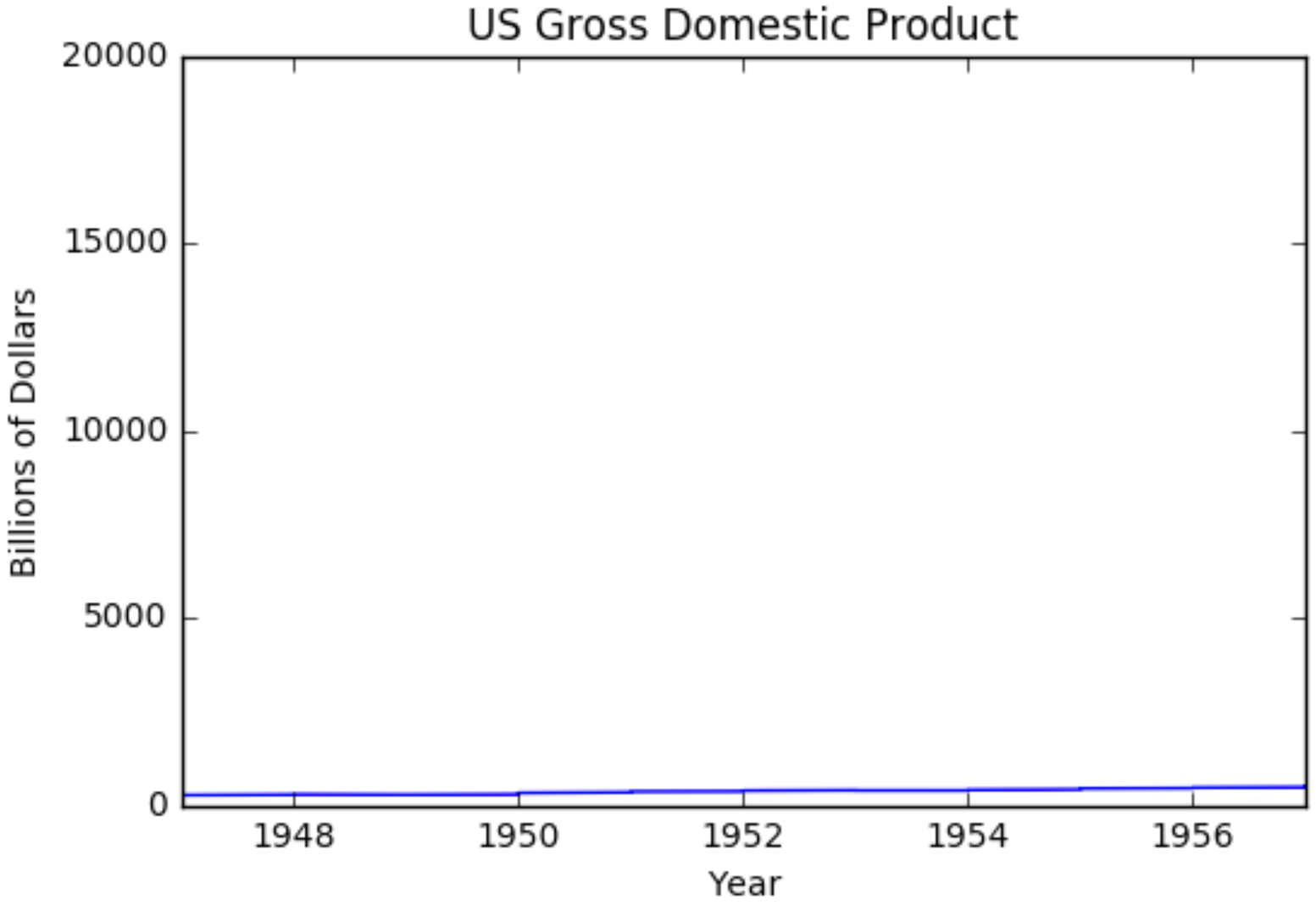
GDP over time



Using xlim()

```
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.show()
```

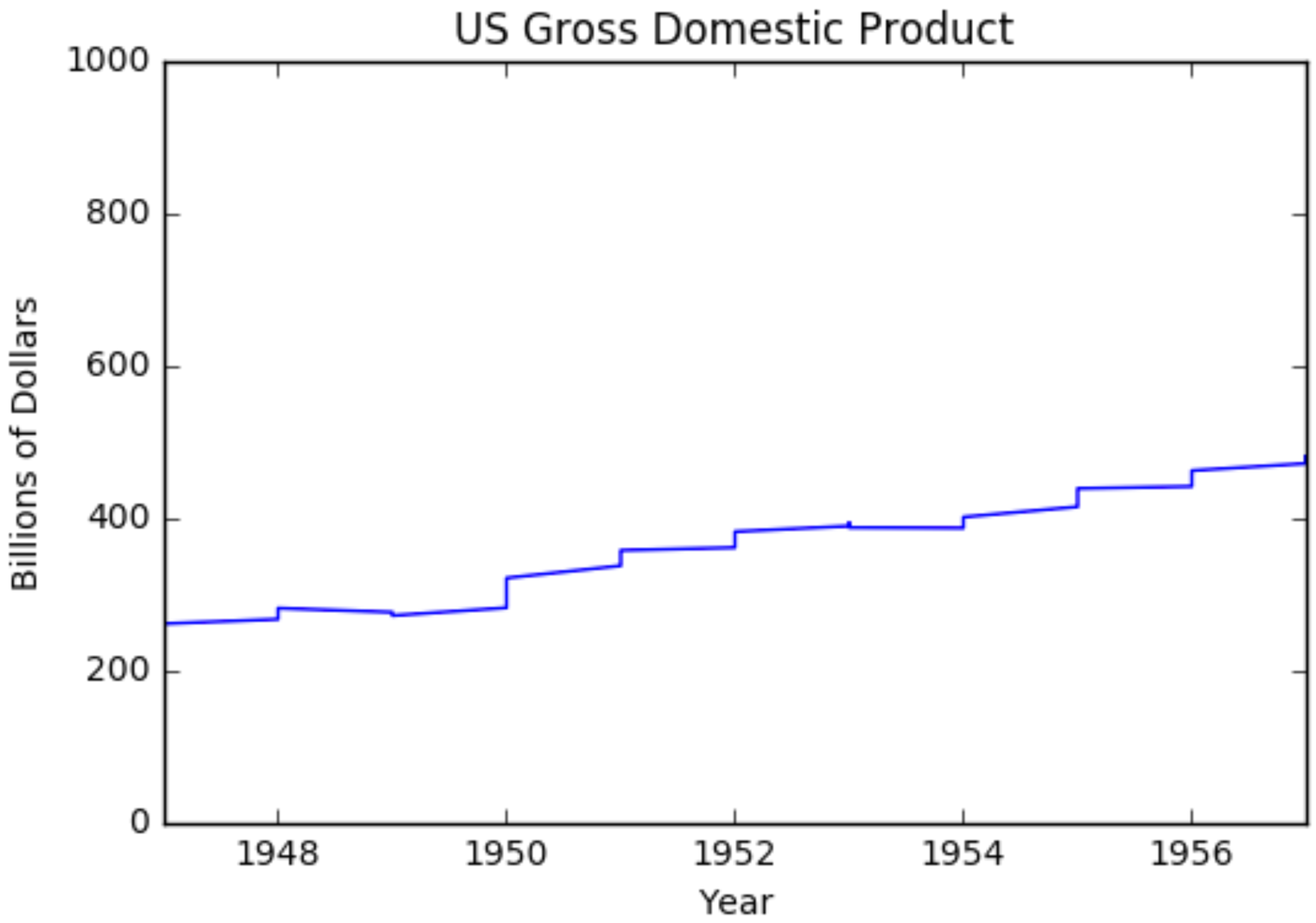
Using xlim()


US Gross Domestic Product
20000
15000
10000
5000
0
Billions of Dollars
1948
1950
1952
1954
1956
Year

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()
```

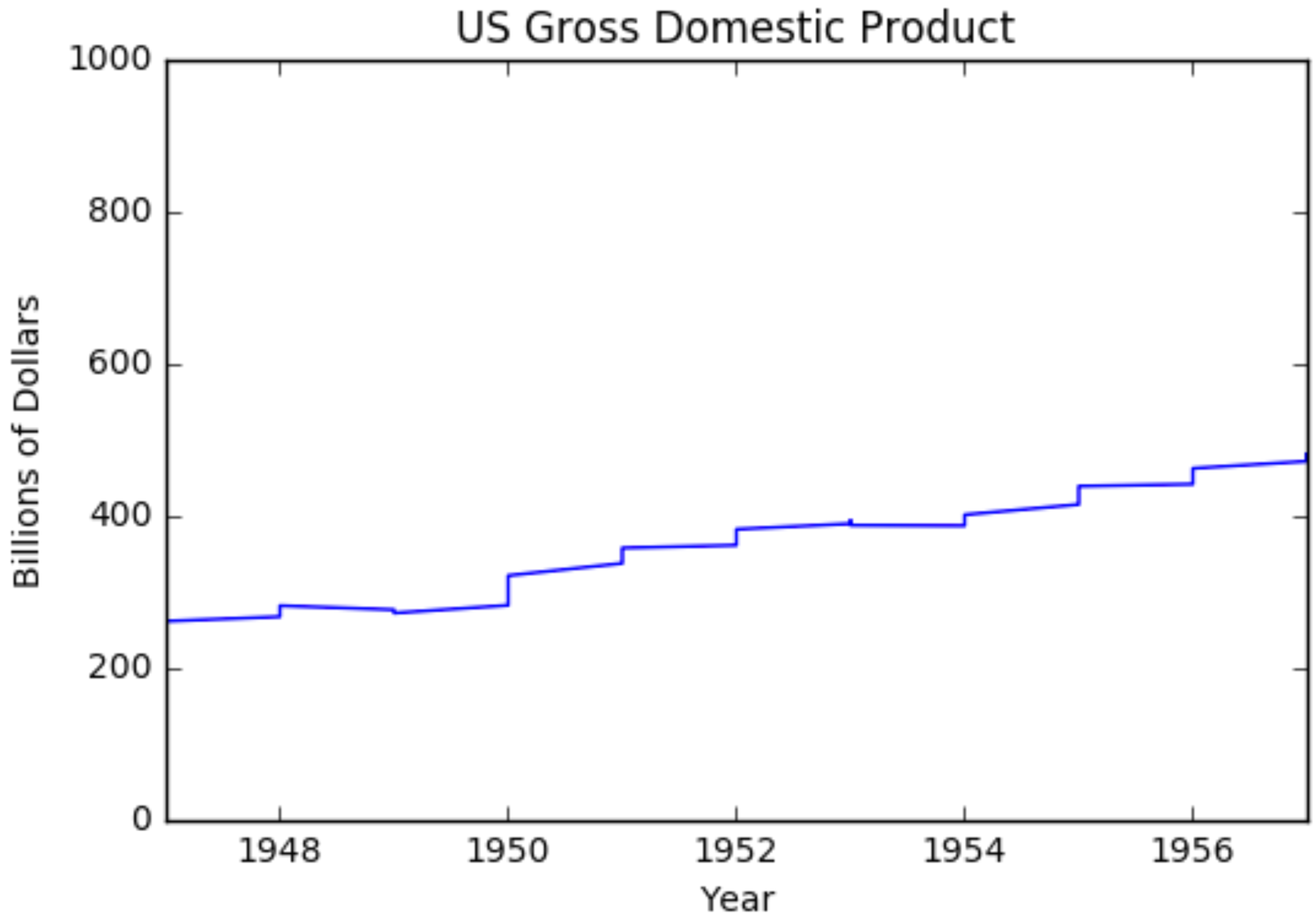
Using xlim() & ylim()



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()
```

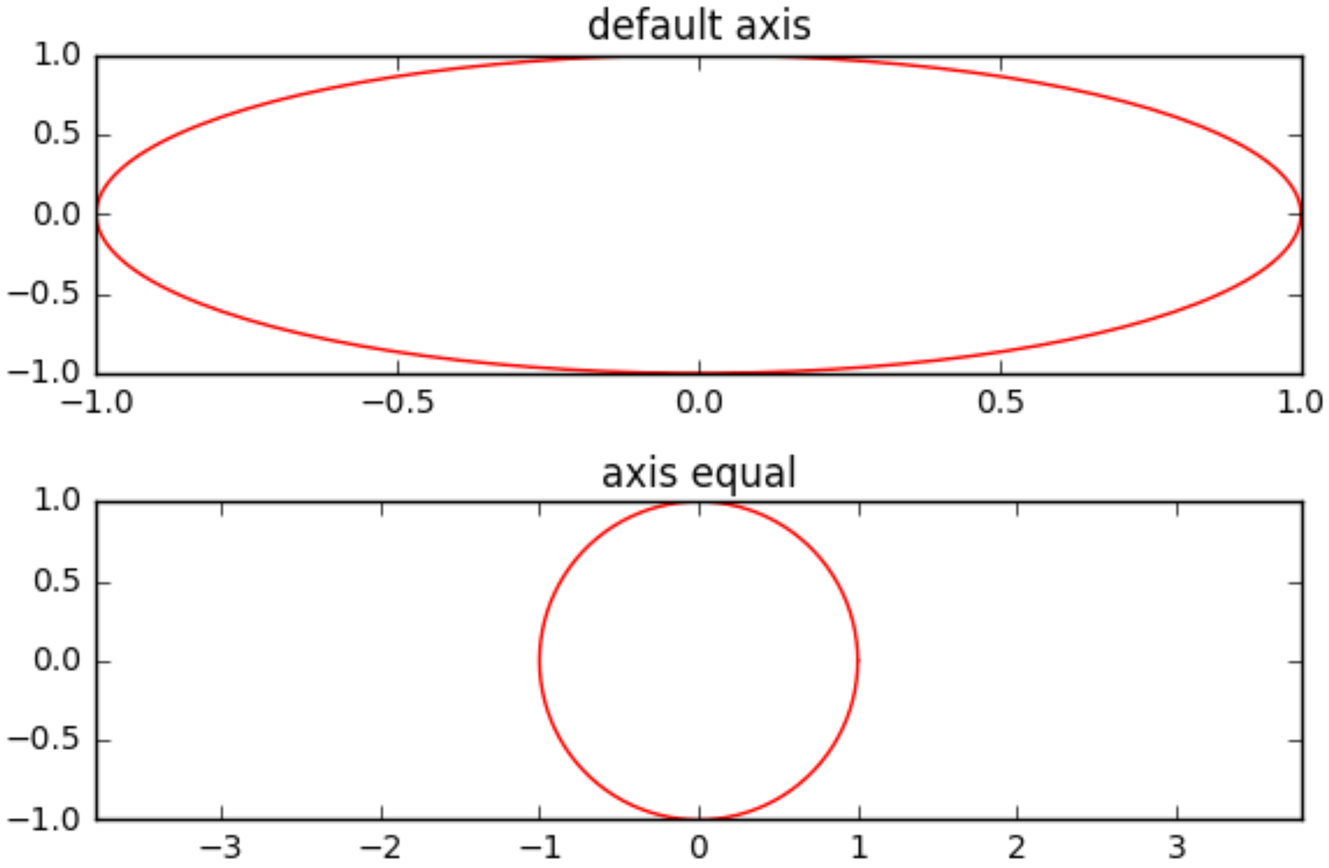
Using axis()


US Gross Domestic Product
1000
800
600
400
200
0
Billions of Dollars
1948
1950
1952
1954
1956
Year

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')



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()
```

INTRODUCTION TO DATA VISUALIZATION WITH PYTHON

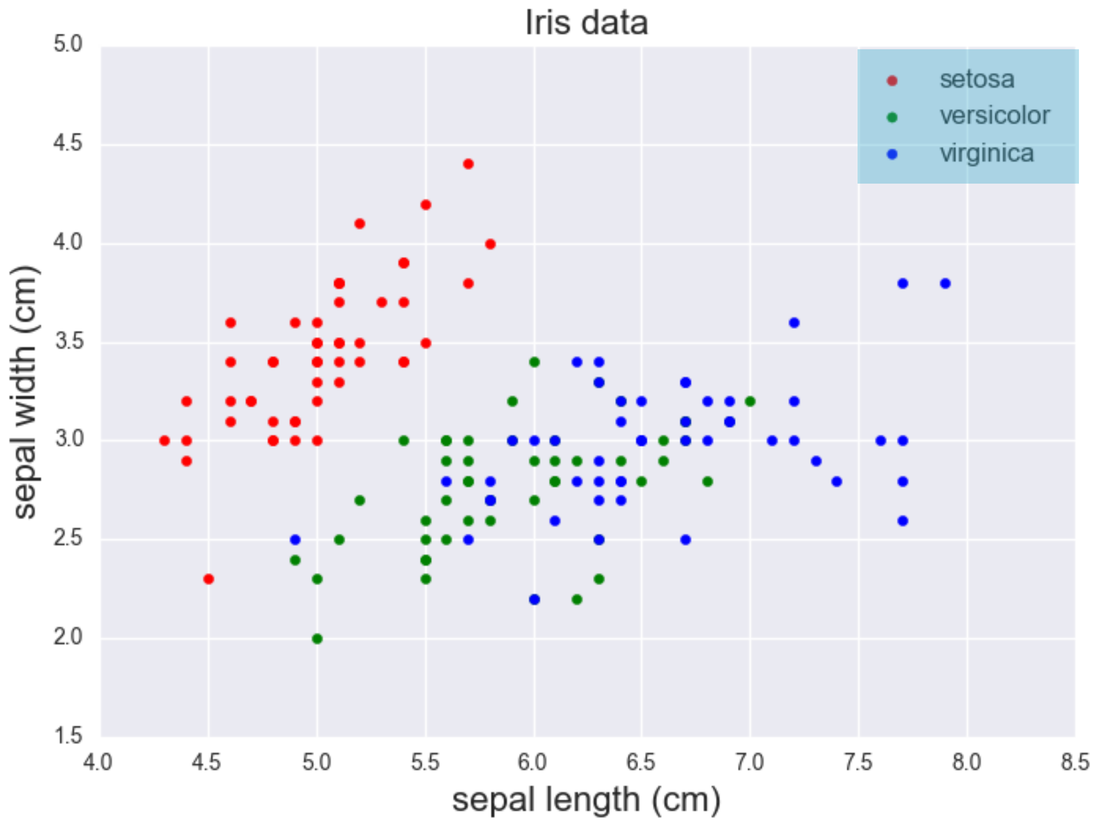
Let’s practice!

INTRODUCTION TO DATA VISUALIZATION WITH PYTHON

Legends, annotations, and styles

Legends

- provide labels for overlaid points and curves



Legend

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')
```

Using legend()

```
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 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

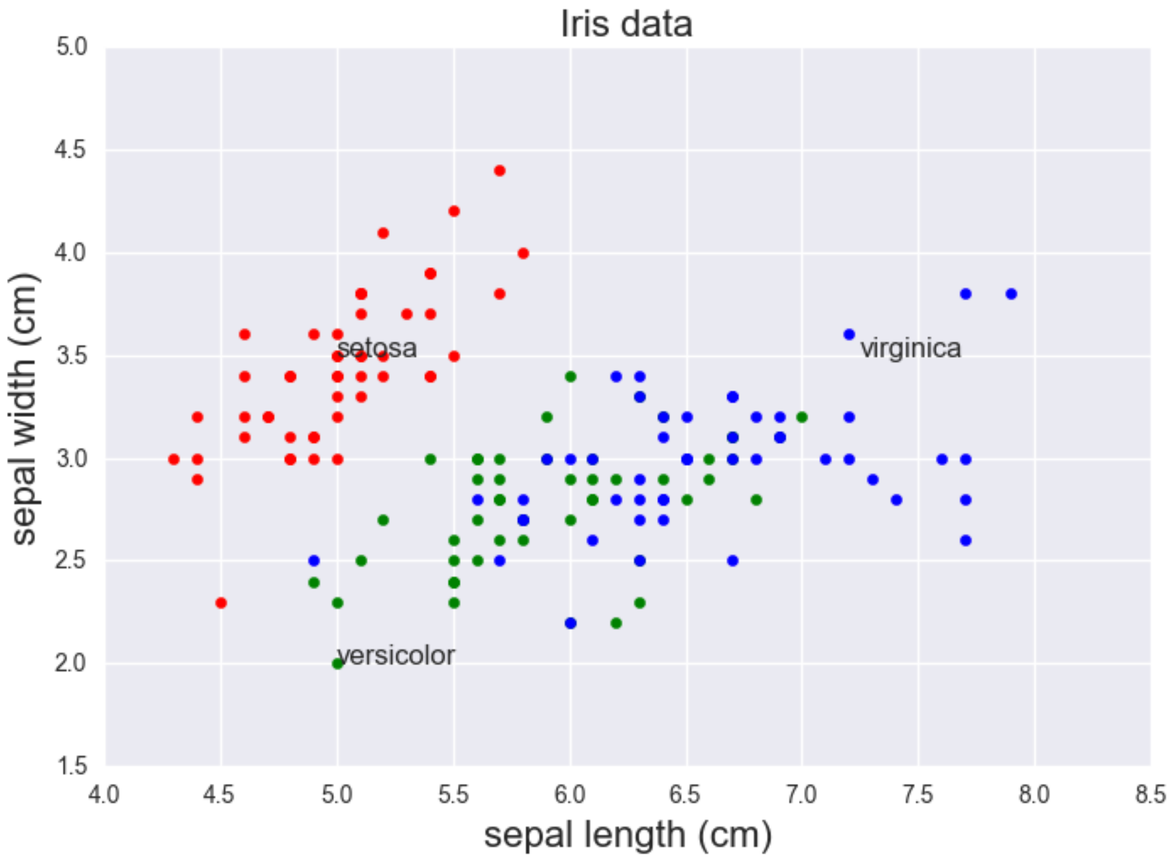
Plot annotations

- 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('virginica', xy=(7.25, 3.5))

In [3]: plt.annotate('versicolor', xy=(5.0, 2.0))

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

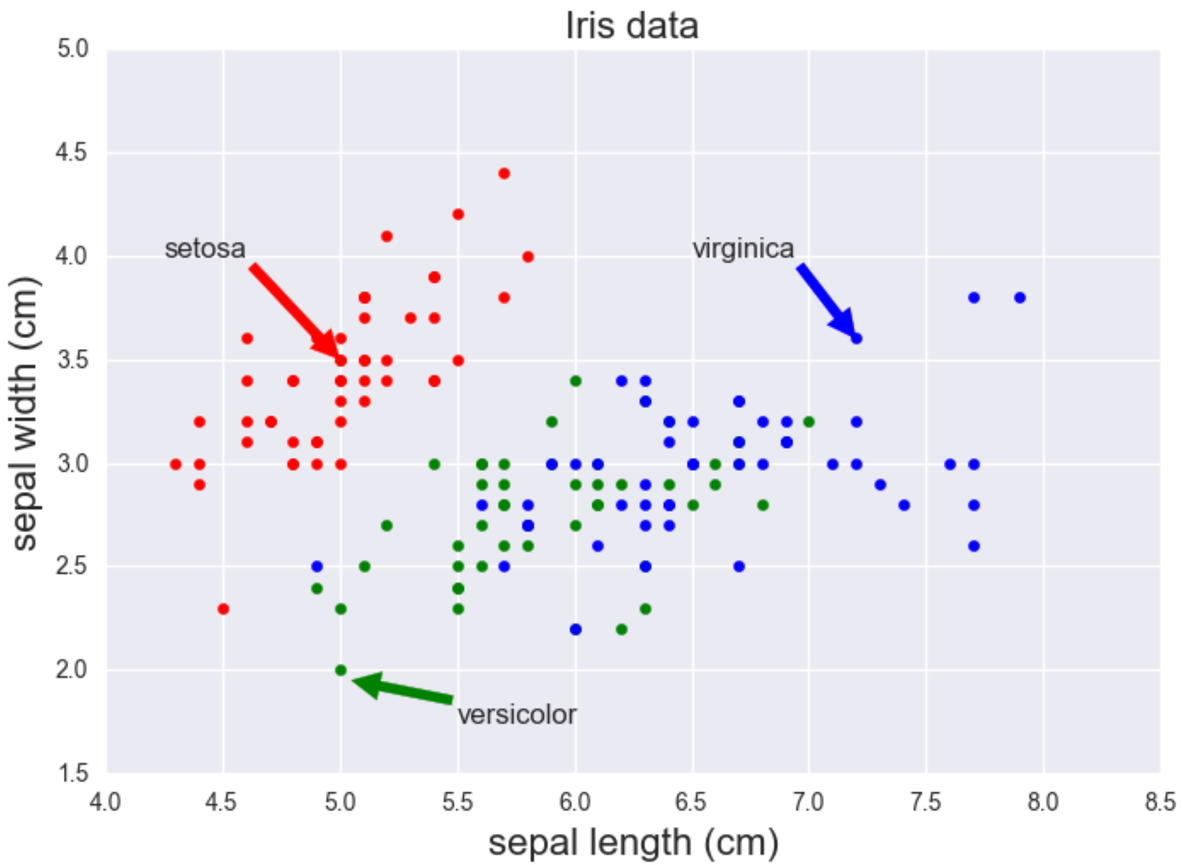
Using annotate() for text



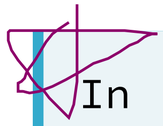
Options for annotate()

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

Using annotate() for arrows



Using annotate() for arrows



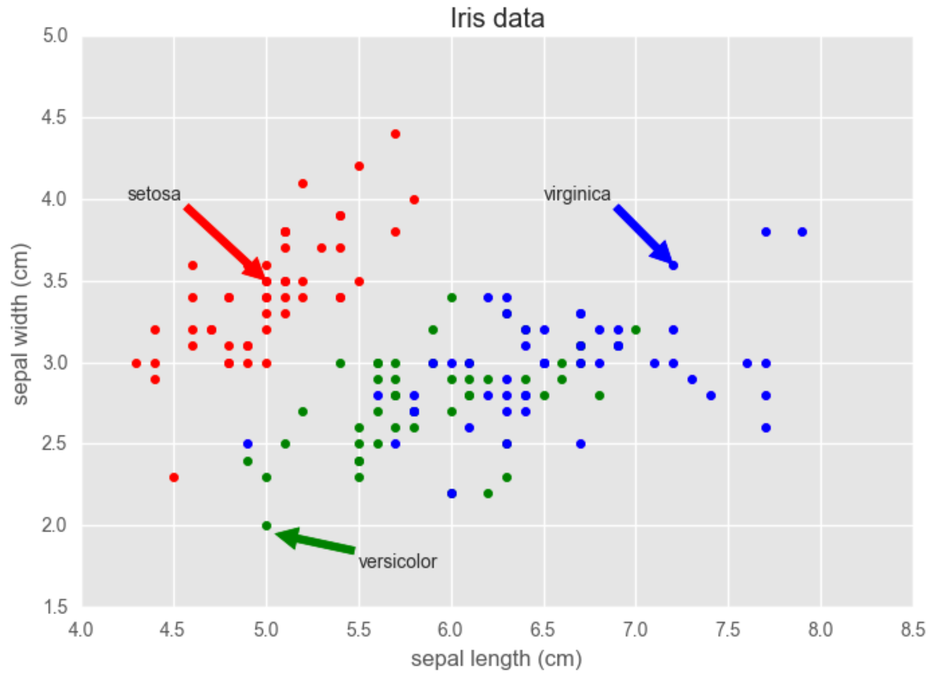
```
In [1]: plt.annotate('setosa', xy=(5.0, 3.5),
   ...:              xytext=(4.25, 4.0), arrowprops={'color':'red'})

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

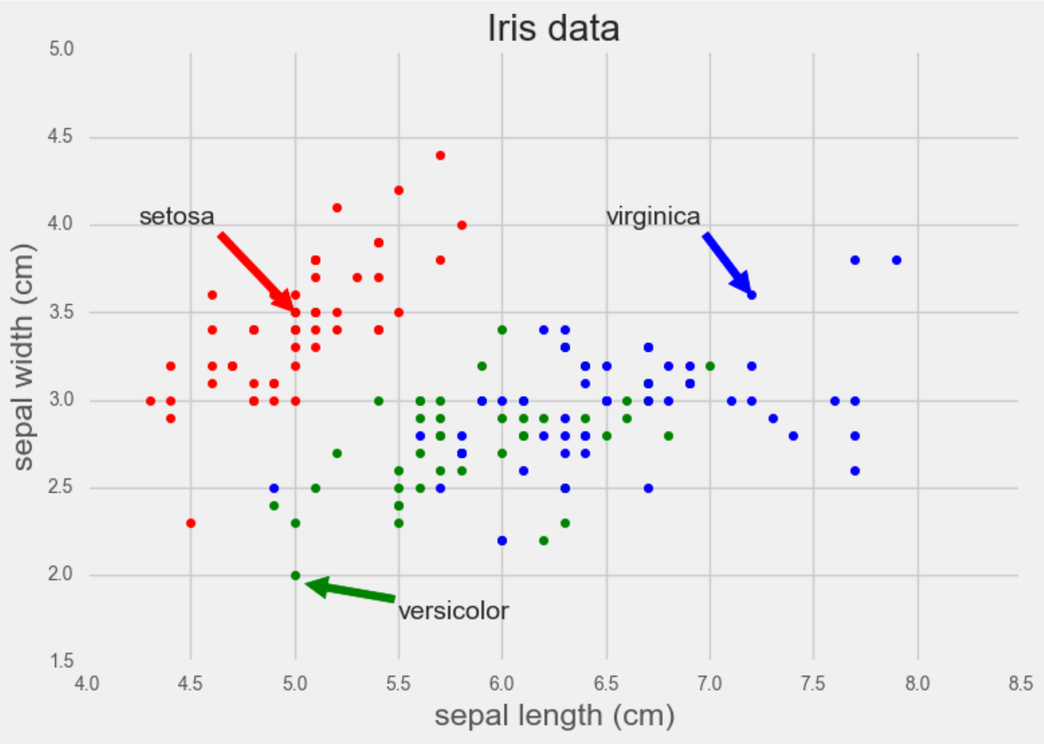
ggplot style

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

In [2]: plt.style.use('ggplot')
```



fivethirtyeight style

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

In [2]: plt.style.use('fivethirtyeight')
```



INTRODUCTION TO DATA VISUALIZATION WITH PYTHON

Let’s practice!