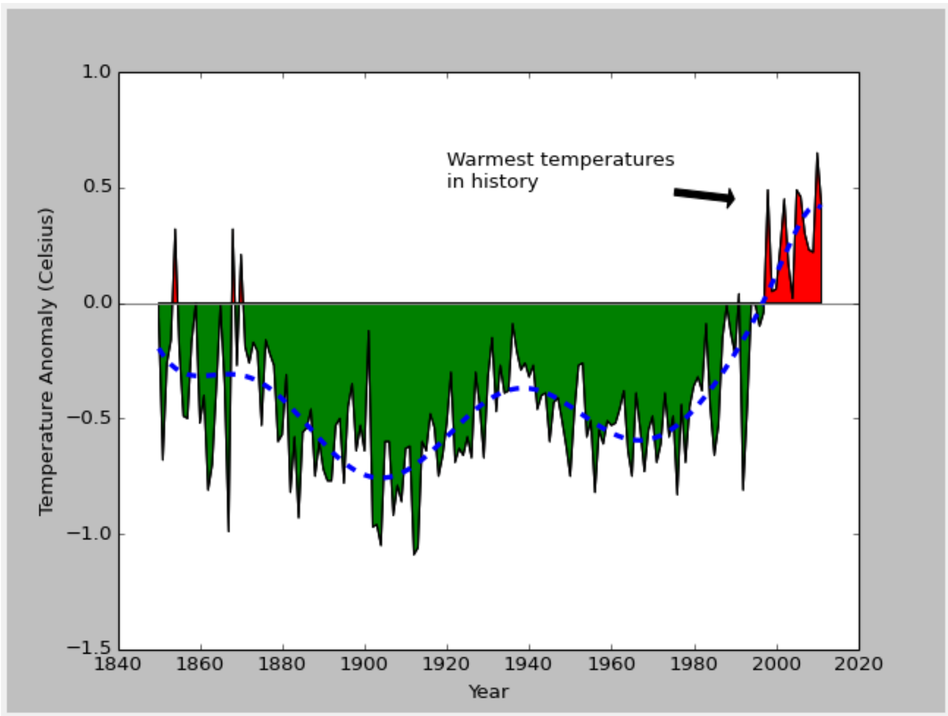
33.1 Introduction to plotting and visualizing data

Many programs interact with sets of data, such as a list of ocean temperatures or daily calorie expenditure. A program can graphically plot, or *visualize*, such data.

Figure 33.1.1: Plot of ocean temperature from 1850 to 2011.



Source: Data source, ocean_temp.csv.

The ***matplotlib*** package can be used for plotting in Python. matplotlib replicates the plotting capability of MATLAB, an engineering-oriented programming language. matplotlib is short for "MATLAB plotting library."

matplotlib is not included with Python, but can be downloaded and installed from https://matplotlib.org/stable/users/installing/. matplotlib also requires the NumPy package.

PARTICIPATION ACTIVITY 33.1.1: Introduction to plotting using matplotlib.

1) matplotlib is a package that ________ .

- helps the programmer debug their program's syntax.
- allows the programmer to display complex math equations.
- enables creating visualizations of data using graphs and charts.

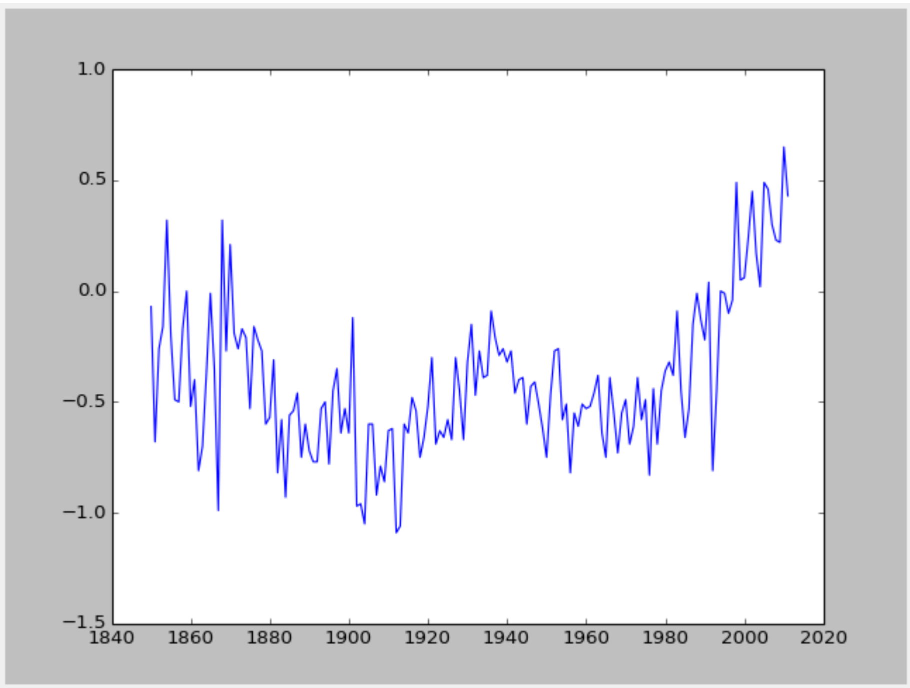
2) matplotlib is installed by default with Python.

- True
- False

A program to plot ocean temperature is below. File ocean_temp.csv contains the data, with one temperature on each line, for year 1850, then 1851, etc.

Figure 33.1.2: A program to plot ocean temperatures read from a file.

```
import matplotlib.pyplot as plt

with open('ocean_temp.csv') as temp_file:
    temps = []
    for t in temp_file:
        temps.append(float(t))

years = range(1850, 2012)

plt.plot(years, temps)
plt.show()
```

The program imports the pyplot module from the matplotlib package, renaming matplotlib.pyplot to *plt* using the *as* keyword. The ***as*** keyword renames an imported module or package. The program then reads the temperatures from a file and stores the temperatures in a list. The plt.***show()*** function displays the graph.

The plt.***plot()*** function plots data onto the graph. plot() accepts various arguments. Above, two lists are passed to the function: The years list is the x-coordinate of each point to plot, and the temps list is the y-coordinate. plot() combines the lists into (x, y) coordinates. Above, years[0] is 1850 and temps[0] is -0.1, so plot() draws a point at (1850, -0.1). The next coordinate is (years[1], temps[1]), or (1851, -0.7). plot() also draws a line between successive points.

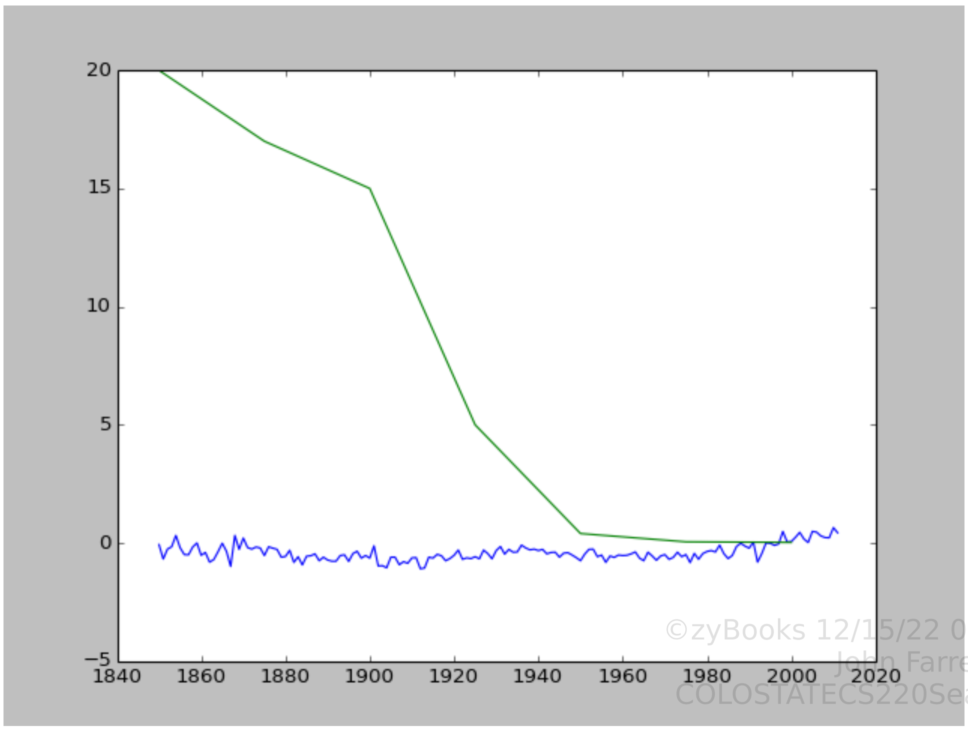
If provided just one list, as in plt.plot(temps), plot() uses 0, 1, ... for x values, as in (0, temps[0]), (1, temps[1]), etc.

Calling plot multiple times draws multiple lines.

Figure 33.1.3: Plotting multiple lines in the same graph.

The below image shows the result when plot() is called twice. The first call plots ocean temperatures per year, and the second call plots the number of pirates (suggesting a correlation between rising ocean temperature and a decrease in piracy).


```
import matplotlib.pyplot as plt

with open('ocean_temp.csv') as temp_file:
    temps = []
    for t in temp_file:
        temps.append(float(t))

temp_years = range(1850, 2012)
plt.plot(temp_years, temps)

pirate_years = range(1850, 2025, 25)
num_pirates_thousands = [20, 17, 15, 5, 0.4, 0.05, 0.025]
plt.plot(pirate_years, num_pirates_thousands)
plt.show()
```



PARTICIPATION ACTIVITY 33.1.2: Plotting data using matplotlib.

1) The plot() function of matplotlib.pyplot can accept as an argument

- a string of text to draw on the graph.
- A dictionary of x, y values.
- two lists of x, y coordinates, e.g., plot([1, 2, 3], [4.0, 3.5, 4.2]).

2) The function call plt.plot([5, 10, 15], [0.25, 0.34, 0.44]) plots an x,y coordinate at

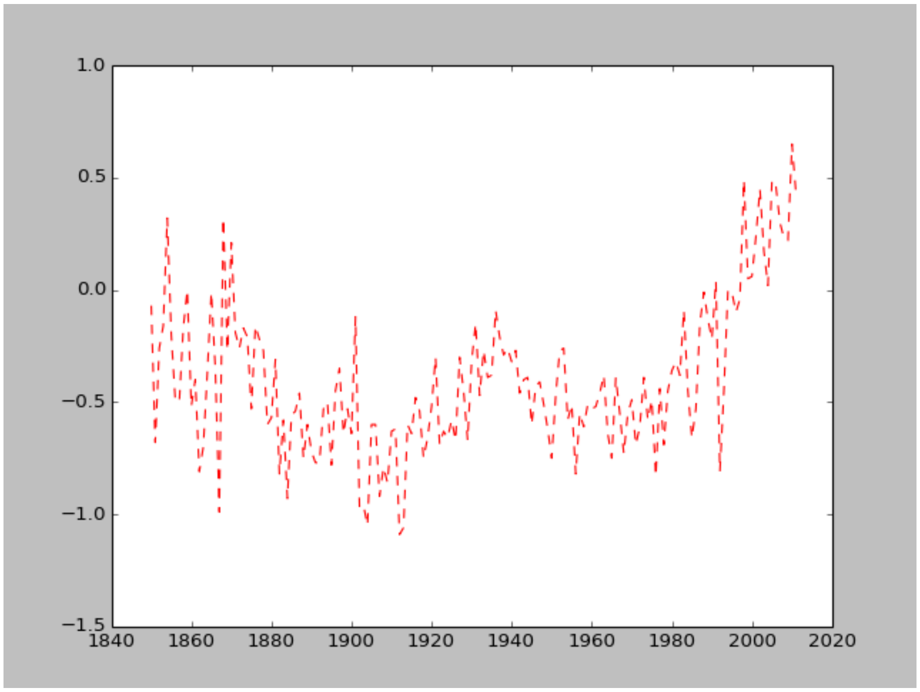
- (5, 0.34)
- (15, 0.44)
- Error

33.2 Styling plots

The plot() function takes an optional ***format string*** argument that specifies the color and style of the plotted line. For example, plot(x_values, y_values, 'r--') uses 'r' to specify a red color, and '--' to specify a dashed line.

Figure 33.2.1: Format string 'r--' sets line color to red and line style to dashed.

```
import matplotlib.pyplot as plt

with open('ocean_temp.csv') as temp_file:
    temps = []
    for t in temp_file:
        temps.append(float(t))

years = range(1850, 2012)

plt.plot(years, temps, 'r--')
plt.show()
```

The below tables describe format string colors and styles. The default format string is 'b-' (solid blue line).

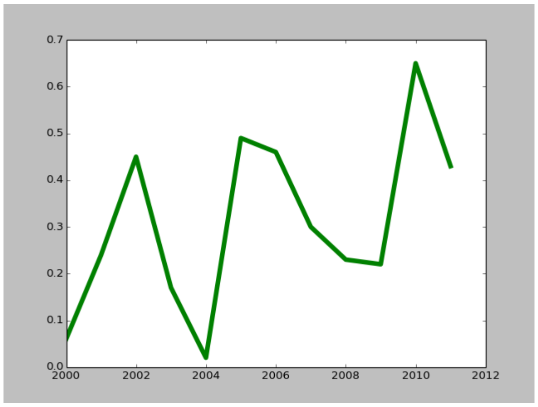
Table 33.2.1: Characters to specify the line color, line style, or marker style.

Character(s)	Line color/style	Character(s)	Marker style	Character(s)	Marker style
b	Blue	.	Point marker	1	Tri-down marker
g	Green	,	Pixel marker	2	Tri-up marker
r	Red	o	Circle marker	3	Tri-left marker
w	White	+	Plus marker	4	Tri-right marker
k	Black	X	X marker	h	Hexagon1 marker
y	Yellow	v	Triangle-down marker	H	Hexagon2 marker
m	Magenta	^	Triangle-up marker	D	Diamond marker
c	Cyan	<	Triangle-left marker	d	Thin diamond marker
-	Solid line	>	Triangle-right marker	|	Vertical line marker
--	Dashed line	*	Star marker	_	Horizontal line marker
-.	Dashed-dot line	p	Pentagon marker	s	Square marker
:	Dotted line				

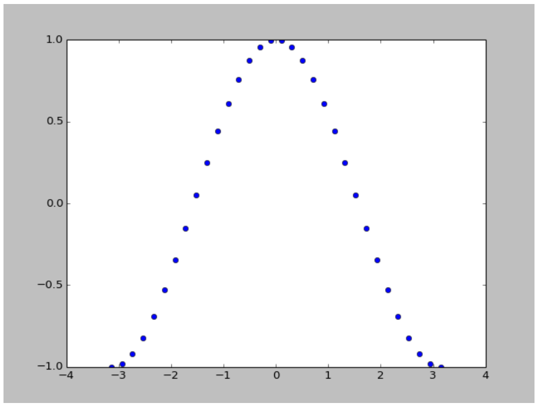
©zyBooks 12/15/22 00:58 1361995 John Farrell COLOSTATECS220SeaboltFall2022

PARTICIPATION ACTIVITY 33.2.1: Line style format strings.

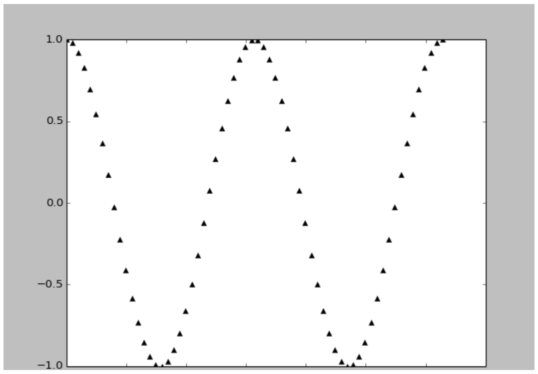
Select the format string used to style the line.

1) 

- g--
- c-
- g-
- g---
- c-.

2) 

- bo
- ob
- b-
- bx
- b*

3) 

- b+
- k+
- kv
- k^
- k>

Format strings are a shortcut to setting line properties. A ***line property*** is an attribute of the line object created by matplotlib when plot() is called. Line properties determine how that line is displayed when show() is called.

There are more line properties than just color and style. The below table describes the most relevant properties.

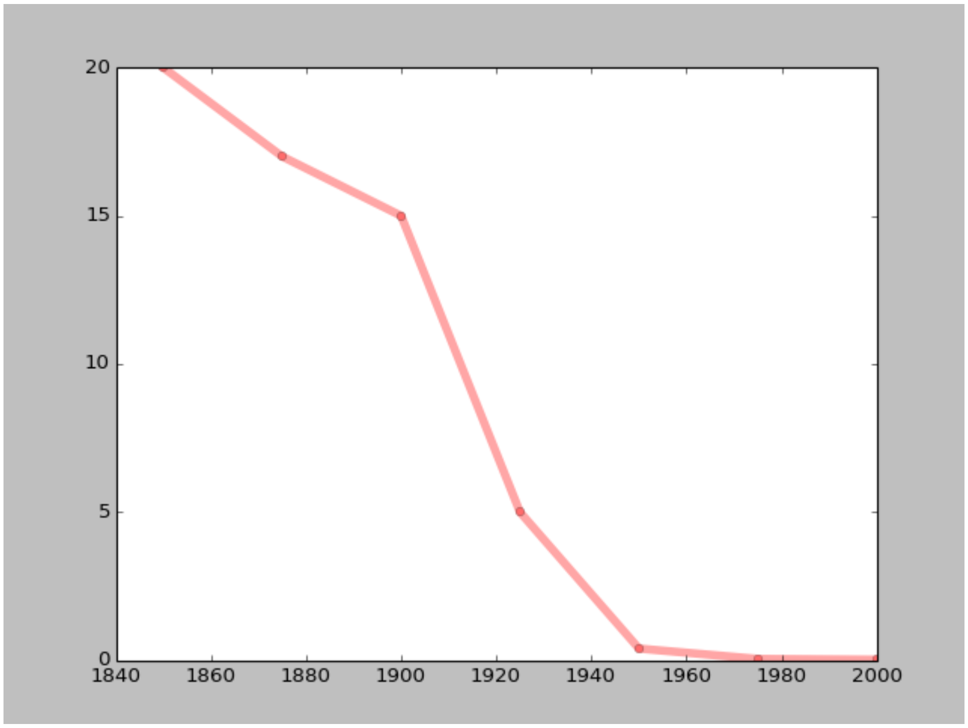
Table 33.2.2: Line properties.

Property	Possible property values	Description
alpha	float	alpha compositing enables transparency
antialiased	Boolean	Enabled anti-aliasing of the line
color	A matplotlib color	Color of the markers, line
solid_capstyle	'butt', 'round', or 'projecting'	How the cap of a line appears
solid_joinstyle	'miter', 'round', or 'bevel'	How the join of a line appears
data	[x_data, y_data]	The arrays of x and y coordinates
label	string	The label to use for the line
linestyle	'-', '--', '-.', ':', ... (see above)	The style of the line
linewidth	float	The width of the line when drawn.
marker	'+', ',', '.', '1', '2', ... (see above)	The style of the marker to use
markersize	float	The size of the marker
visible	Boolean	Show/hide the line

Format strings provide useful shortcuts to the color, linestyle, and marker properties. Use keyword arguments to change other properties' values.

Figure 33.2.2: Use keyword args to change line properties.

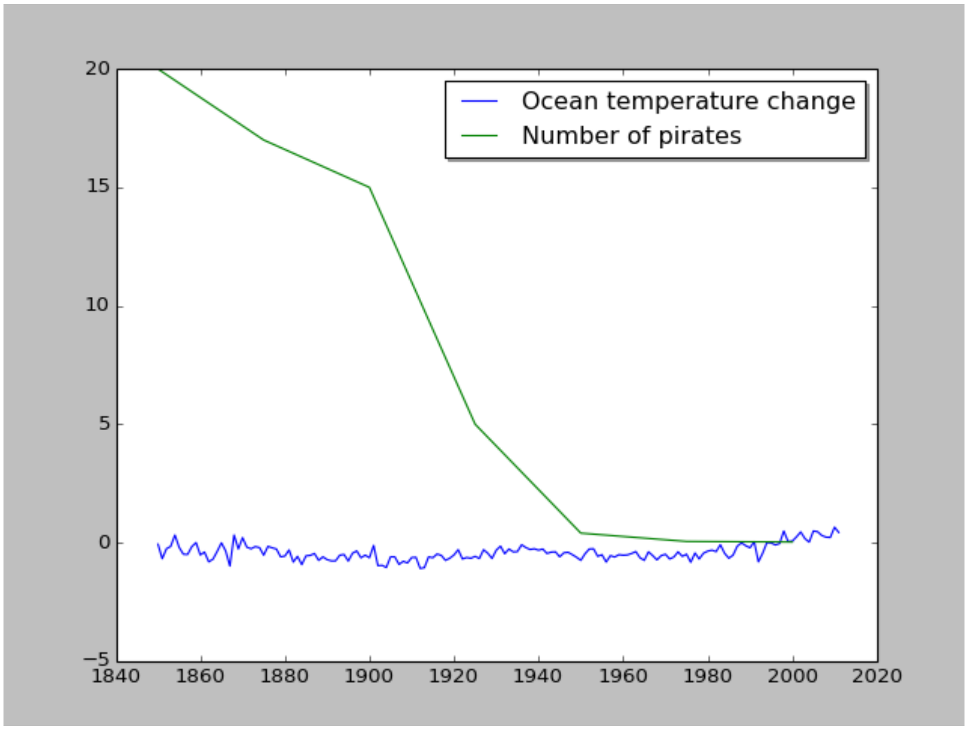
```
import matplotlib.pyplot as plt

pirate_years = range(1850, 2025, 25)
number_of_pirates_thousands = [20, 17, 15, 5, 0.4, 0.05,
0.025]
plt.plot(pirate_years, number_of_pirates_thousands, 'ro-',
         linewidth=5, markersize=5, alpha=0.35)
plt.show()
```

The plt.legend() function displays a legend of the lines, using the label arguments passed to plot() as the text. Various keyword arguments can be given to customize the legend's appearance.

Figure 33.2.3: Adding a legend to a plot.

```
import matplotlib.pyplot as plt

with open('ocean_temp.csv') as temp_file:
    temps = []
    for t in temp_file:
        temps.append(float(t))

temp_years = range(1850, 2012)
plt.plot(temp_years, temps, label="Ocean temperature
change")

p_years = range(1850, 2025, 25)
pirates_thousands = [20, 17, 15, 5, 0.4, 0.05, 0.025]
plt.plot(p_years, pirates_thousands, label="Number of
pirates")
plt.legend(shadow=True, loc="upper right")
plt.show()
```

PARTICIPATION ACTIVITY 33.2.2: Line properties and legends.

1) Set the plotted line's marker size to 10.

```
plt.plot(times,
temperatures,
                    )
```

Check Show answer

2) [illegible]h of the plotted line to 3, and the color of the line to green.


```
plt.plot(times,
temperatures,
                    )
```

Check Show answer

3) Enable a legend located in the bottom right of a graph.

```
plt.
```

Check Show answer

Exploring further:

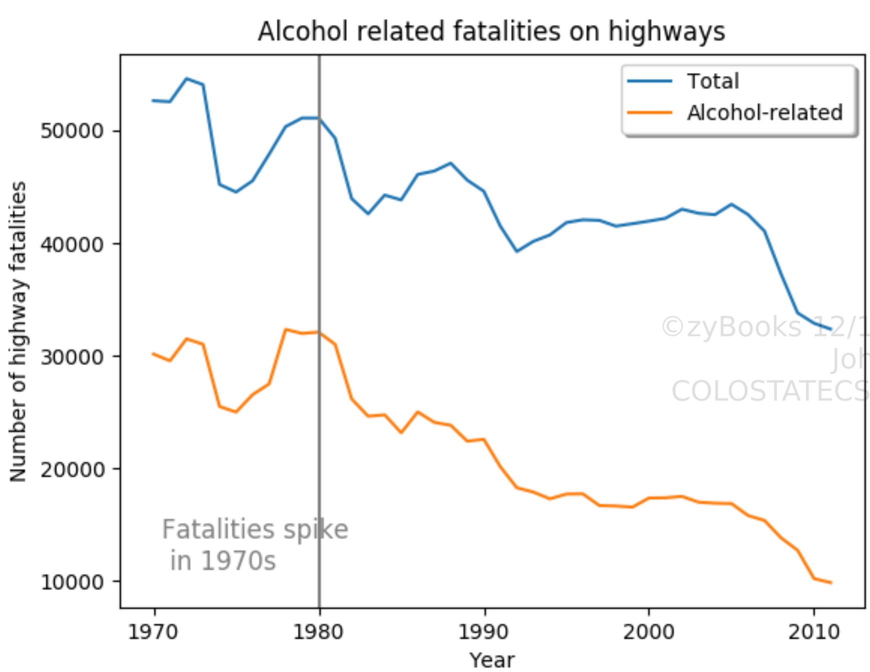
- The plot() function
- More on customizing legends

33.3 Text and annotations

Text labels can help draw attention to interesting parts of a plot. Consider the plot below where a text label marks an important point on the x-axis.

Figure 33.3.1: Adding text to a plot.

```
import matplotlib.pyplot as plt

with open('dd_stats.csv') as f:
    total_fatalities = []
    alcohol_fatalities = []
    for line in f:
        total, alcohol = line.split(',')
        total_fatalities.append(int(total))
        alcohol_fatalities.append(int(alcohol))

years = range(1970, 2012)
plt.plot(years, total_fatalities, label="Total")
plt.plot(years, alcohol_fatalities, label="Alcohol-related")

plt.xlabel('Year')
plt.ylabel('Number of highway fatalities')
plt.legend(shadow=True, loc="upper right")

# Add plot title
plt.title("Alcohol related fatalities on highways")

# Add text giving x,y coordinates of the plot
plt.text(1970.5, 11000, 'Fatalities spike\n in 1970s', color='grey',
fontsize=12)

# Add a vertical line at x-coordinate 1980
plt.axvline(1980, color='grey')

plt.show()
```

©zyBooks 12/15/22 00:58 1361995
John Farrell
COLOSTATECS220SeaboltFall2022

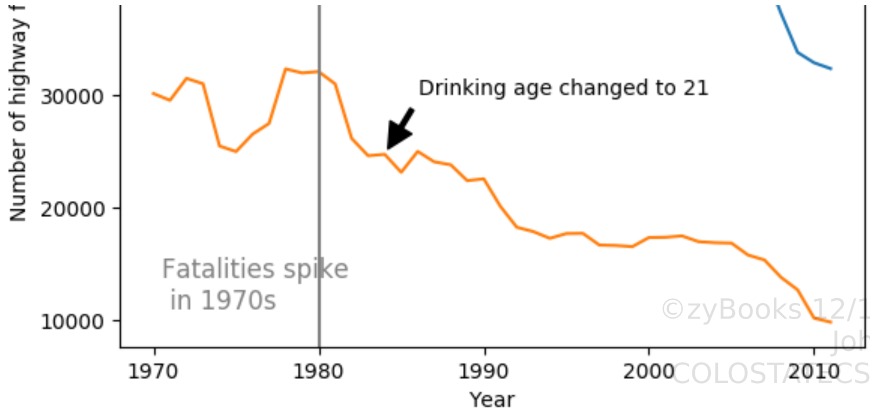
Source: Data source, dd_stats.csv

The text() function draws a string label on the plot. The first two arguments specify an x,y coordinate of the label. Optional keyword arguments customize the appearance of the label.

The annotate() function creates an ***annotation*** that links a text label with a specific data point. The programmer specifies the coordinates of the text label and the data point, and an arrow is automatically drawn from text to data point.

Figure 33.3.2: Annotating a specific data point.

```
import matplotlib.pyplot as plt

with open('dd_stats.csv') as f:
    total_fatalities = []
    alcohol_fatalities = []
    for line in f:
        total, alcohol = line.split(',')
        total_fatalities.append(int(total))
        alcohol_fatalities.append(int(alcohol))

years = range(1970, 2012)
plt.plot(years, total_fatalities, label="Total")
plt.plot(years, alcohol_fatalities, label="Alcohol-related")

plt.xlabel('Year')
plt.ylabel('Number of highway fatalities')
plt.legend(shadow=True, loc="upper right")

# Add plot title
plt.title("Alcohol related fatalities on highways")

# Add text giving x,y coordinates of the plot
plt.text(1970.5, 11000, 'Fatalities spike\n in 1970s', color='grey',
fontsize=12)

# Add a vertical line at x-coordinate 1980
plt.axvline(1980, color='grey')

# Add annotation
arrow_properties = {
    'facecolor': 'black',
    'shrink': 0.1,
    'headlength': 10,
    'width': 2
}

plt.annotate('Drinking age changed to 21',
             xy=(1984, 24762),
             xytext=(1986, 30000),
             arrowprops=arrow_properties)

plt.show()
```

The first argument to annotate() is the label to display, which is placed at the coordinate described by xytext. Argument xy is the datapoint that the arrowhead points to. The arrow's appearance can be customized by passing a dictionary of arrow properties.

PARTICIPATION ACTIVITY 33.3.1: Text and annotations.

1) Draw the string "Peak current" at coordinate (5, 10).

```
plt.text(          )
```

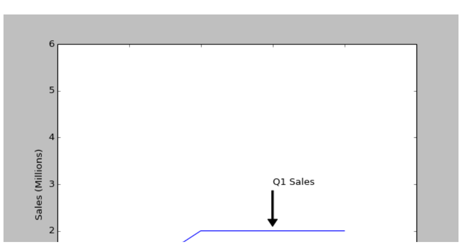
Check Show answer

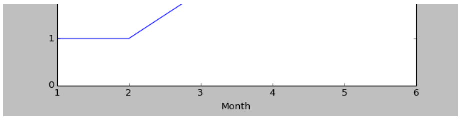
2) Annotate the data point at (100, 5), placing the text 'Peak current' at (115, 10).

```
plt.annotate('Peak
current',          )
```

Check Show answer

3) Complete the call that results in the following annotation:






```
plt.annotate(
```
`)`

Check

Show answer

Exploring further:

- Customizing the appearance of text labels
- Customizing the appearance of arrows

33.4 Numpy

The ***numpy*** package provides tools for scientific and mathematical computations in Python. For example, numpy includes functions that can be used to perform common linear algebra operations, fast fourier transforms, and statistics. Numpy must be downloaded and installed from https://numpy.org/install/

Numpy uses an ***array*** data type that is conceptually similar to a list, consisting of an ordered set of elements of the same type. An array can be created using the array() constructor from the numpy package. Multidimensional arrays are created by specifying a list with a tuple for each dimension's elements.

Figure 33.4.1: Creating arrays.

```
import numpy as np

# 1-dimension array
my_array1 = np.array([15.5, 25.11,
19.0])
print('my_array_1:')
print(my_array1)
print()

# 2-dimension array
my_array2 = np.array([(34, 25), (16,
12)])
print('my_array_2:')
print(my_array2)
```

```
my_array_1:
[ 15.5   25.11  19.
]

my_array_2:
[[34 25]
 [16 12]]
```


Sometimes an array must be created before the element values are known. Changing the size of an array is an expensive computation, so numpy provides functions that create empty pre-sized arrays. The zeros() function creates an array with a 0 for every element, and ones() uses 1 for every element. The argument to zeros() and ones() is an integer (length) for a 1-dimensional array, or a tuple (row length, column length) for a 2-dimensional array.

Figure 33.4.2: Pre-initialized arrays.

```
import numpy as np

zero_array = np.zeros(5)    # 1-dimension array with 5
elements
print('zero_array:')
print(zero_array)
print()

one_array = np.ones((5, 2))  # 2-dimension array with
5 rows and 2 elements per row (2 columns)
print('one_array:')
print(one_array)
```

```
zero_array:
[ 0.  0.  0.
0.  0.]

one_array:
[[ 1.  1.]
 [ 1.  1.]
 [ 1.  1.]
 [ 1.  1.]
 [ 1.  1.]]
```


PARTICIPATION ACTIVITY 33.4.1: Creating arrays.

Choose the answer that creates the shown array:

1) `[ 5 10 15 ]`

- np.array(5, 10, 15)
- np.array([5, 10, 15])

2)
```
[[ 0.  0.]
 [0.  0.]
 [0.  0.]]
```

- np.zeros((3, 2))
- np.zeros((2,3))

A common operation is to create a sequence of numbers, like 0, 1, 2, ... using range(). However, range() creates sequences of integers only and can not generate fractional values. The ***linspace*** numpy function creates a sequence by segmenting a given range with a specified number of points. For example, linspace(0, 1, 11) creates a sequence with 11 elements between 0 and 1 inclusive: 0, 0.1, 0.2, ..., 0.9, 1.0.

Figure 33.4.3: Creating sequences using linspace().

```python
import numpy as np

print(np.linspace(0, 1, 11))
print()
print(np.linspace(0,
np.sin(np.pi/4), 20))
```

```
[ 0.   0.1  0.2  0.3  0.4  0.5  0.6  0.7
0.8  0.9  1. ]

[ 0.          0.03721615  0.07443229
0.11164844  0.14886459  0.18608073
  0.22329688  0.26051302  0.29772917
0.33494532  0.37216146  0.40937761
  0.44659376  0.4838099   0.52102605
0.5582422   0.59545834  0.63267449
  0.66989063  0.70710678]
```

PARTICIPATION ACTIVITY 33.4.2: Creating sequences.

1) Use np.linspace() to create the sequence:

`[ 0.25 0.5 0.75 1.0 ]`

Check Show answer

Mathematical operations between arrays are performed between the matching elements of each array. For example, [5 5 5] + [1 2 3] would compute [5+1 5+2 5+3], or [6 7 8]. The program below shows some common array operations.

Figure 33.4.4: Array operations program.


```
import numpy as np

array1 = np.array([10, 20, 30, 40])
array2 = np.array([1, 2, 3, 4])

# Some common array operations

print('Adding arrays (array1 + array2)')
print(array1 + array2)

print('\nSubtracting arrays (array1 -
array2)')
print(array1 - array2)

print('\nMultiplying arrays (array1 *
array2)')
print(array1 * array2)

print('\nMatrix multiply (dot(array1 *
array2))')
print(np.dot(array1, array2))

print('\nFinding square root of each
element in array1')
print(np.sqrt(array1))

print('\nFinding minimum element in
array1')
print(array1.min())

print('\nFinding maximum element in
array1')
print(array1.max())
```

```
Adding arrays (array1 + array2)
[11 22 33 44]

Subtracting arrays (array1 -
array2)
[ 9 18 27 36]

Multiplying arrays (array1 *
array2)
[ 10  40  90 160]

Matrix multiply (dot(array1 *
array2))
300

Finding square root of each
element in array1
[ 3.16227766  4.47213595
5.47722558  6.32455532]

Finding minimum element in
array1
10

Finding maximum element in
array1
40
```


Exploring further:

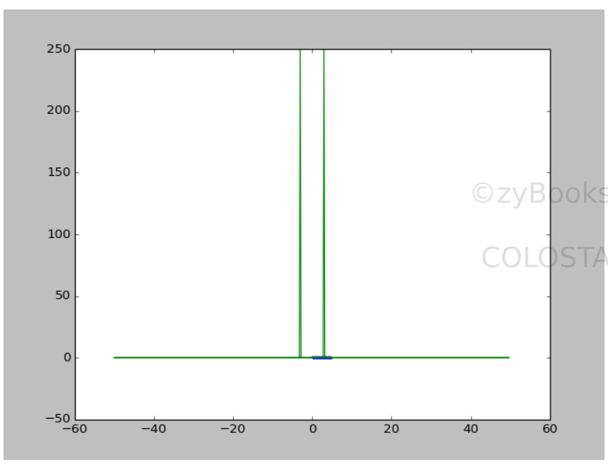
- numpy documentation
- numpy tutorial

33.5 Multiple plots

A plot with too much data can be difficult to read. Furthermore, if different data series in the plot have different ranges of values, then interpreting the data becomes impossible. Consider the program below that plots two data series:

Figure 33.5.1: Two types of data on the same plot.

```
import numpy as np
import matplotlib.pyplot as plt

# Wave parameters
FREQUENCY = 3
SAMPLING_RATE = 100
TIME_STEP = 1.0 / SAMPLING_RATE

# Like range() for floating point
t1 = np.arange(0.0, 5.0, TIME_STEP)

# Compute a sine wave
wave = np.sin(FREQUENCY*2*np.pi*t1)

# Compute Fast Fourier Transform (FFT)
fft_result = np.fft.fft(wave)

# Compute x-axis values for frequency
domain
t2 = np.fft.fftfreq(len(t1), TIME_STEP)

plt.plot(t1, wave)
plt.plot(t2, np.abs(fft_result))
plt.show()
```

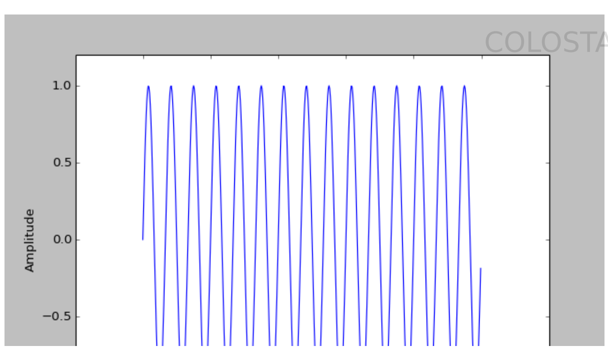


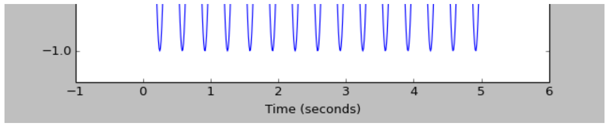
The above program attempts to plot a 3 Hertz sine wave and the amplitude spectrum of the Fast Fourier Transform (FFT) of the wave. However, the plot does not convey much useful information because the axes have been automatically scaled to fit the larger FFT result values, making the sine wave (in blue) difficult to see.

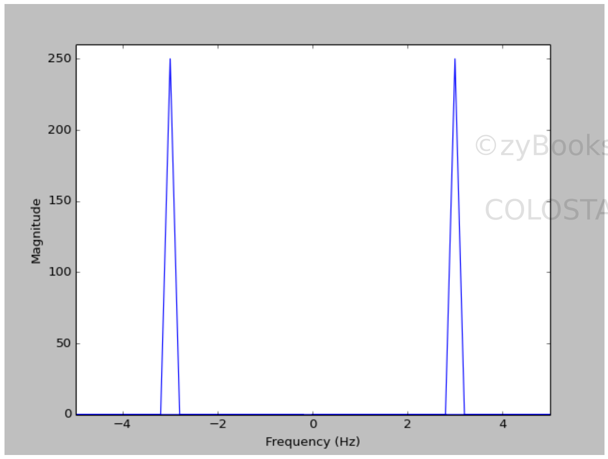
The two plotted series require different axes; the x-axis of the sine wave is time (seconds), and the x-axis of the FFT result is frequency (Hertz). The ***figure()*** function can be used to create multiple figures. Each figure corresponds to a window frame to be opened by matplotlib, and each figure can contain a plot that uses different axes.

Figure 33.5.2: Using multiple figures.

```
import numpy as np
import matplotlib.pyplot as plt

# Unique identifiers for each figure
FIGURE1 = 1
FIGURE2 = 2

# Wave parameters
FREQUENCY = 3
SAMPLING_RATE = 100
TIME_STEP = 1.0 / SAMPLING_RATE

# Like range() for floating point
t1 = np.arange(0.0, 5.0, TIME_STEP)

# Compute a sine wave
wave = np.sin(FREQUENCY*2*np.pi*t1)

# Compute Fast Fourier Transform (FFT)
fft_result = np.fft.fft(wave)

# Compute x-axis values for frequency domain
t2 = np.fft.fftfreq(len(t1), TIME_STEP)

plt.figure(FIGURE1)
plt.plot(t1, wave)
plt.xlabel("Time (seconds)")
plt.ylabel("Amplitude")
plt.axis([-1, 6, -1.2, 1.2])  # Empty space
buffer

plt.figure(FIGURE2)
plt.plot(t2, np.abs(fft_result))
plt.xlabel("Frequency (Hz)")
plt.ylabel("Magnitude")

# Set plot axis ranges [min_x, max_x, min_y,
max_y]
plt.axis([-5, 5, 0, 260])

plt.show()
```





The figure() function sets the current figure, using the argument to identify the figure to activate. Subsequent calls like plt.plot() and plt.xlabel() affect the current figure. The first figure is created by matplotlib automatically; calling figure(FIGURE1) in previous examples was unnecessary. The call to figure(FIGURE2) is needed to create a new figure for the FFT plot.

The ***plt.axis()*** function is used to set the range of the x and y axes. A single list argument specifies the minimum and maximum values of each axis: [min_x, max_x, min_y, max_y]. Above, the axes of the signal frequency plot are set to show only the interesting region of the plot.

PARTICIPATION ACTIVITY | 33.5.1: Multiple figures.

1) FIGURE1 has an x-axis label of "Seconds".

```
plt.figure(FIGURE1)
plt.plot(t1, y1)
plt.xlabel("Time")
plt.ylabel("Volts")

plt.figure(FIGURE2)
plt.plot(t2, y2)
plt.xlabel("Seconds")
```


- True
- False

2) FIGURE1 and FIGURE2 both have a legend.

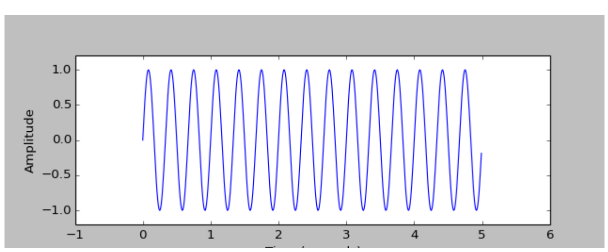
```
plt.figure(FIGURE1)
plt.plot(t1, y1, label="One")
plt.legend()

plt.figure(FIGURE2)
plt.plot(t2, y2, label="Two")
plt.xlabel("Seconds")
```

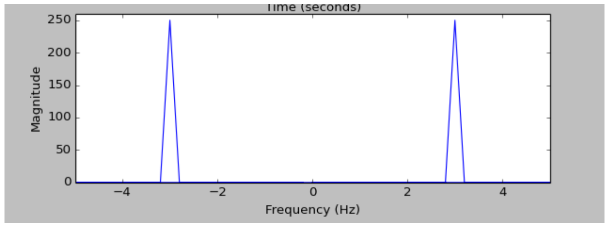
- True
- False

The ***subplot()*** function allows multiple plots to be created in a single figure, with each subplot having its own set of axes. Subplots are often preferable to multiple figures when related data is plotted.

Figure 33.5.3: Using subplots.

```
import numpy as np
import matplotlib.pyplot as plt

# Unique identifiers for each figure
FIGURE1 = 1
FIGURE2 = 2

# Wave parameters
FREQUENCY = 3
SAMPLING_RATE = 100
TIME_STEP = 1.0 / SAMPLING_RATE

# Like range() for floating point
t1 = np.arange(0.0, 5.0, TIME_STEP)

# Compute a sine wave
wave = np.sin(FREQUENCY*2*np.pi*t1)

# Compute Fast Fourier Transform (FFT)
fft_result = np.fft.fft(wave)

# Compute x-axis values for frequency domain
t2 = np.fft.fftfreq(len(t1), TIME_STEP)

plt.subplot(2, 1, 1)  # 2 rows, 1 column. Use loc
1
plt.plot(t1, wave)
plt.xlabel("Time (seconds)")
plt.ylabel("Amplitude")
plt.axis([-1, 6, -1.2, 1.2])  # Empty space buffer

plt.subplot(2, 1, 2)  # 2 rows, 1 column. Use loc
2
plt.plot(t2, np.abs(fft_result))
plt.xlabel("Frequency (Hz)")
plt.ylabel("Magnitude")

# Set plot axis ranges [min_x, max_x, min_y,
max_y]
plt.axis([-5, 5, 0, 260])

plt.tight_layout(pad=1.0) # Add padding between
plots
plt.show()
```

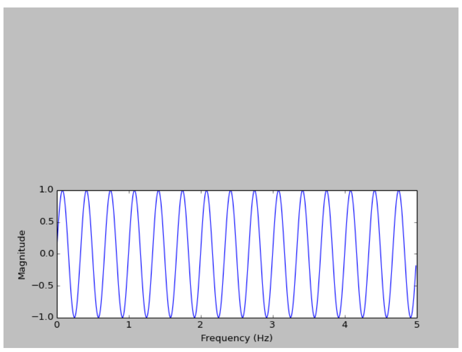





Subplot() sets the active subplot; subsequent calls affect only the current figure and subplot. The first and second arguments specify the number of rows and columns to use. The third argument specifies the current active subplot, and must contain a value between 1 and (number of rows * number of columns).

PARTICIPATION ACTIVITY 33.5.2: Subplots.

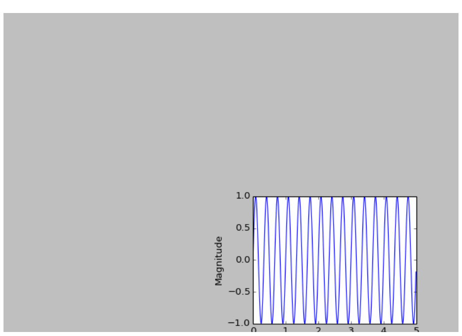
1) Complete the subplot() call to set the active subplot to the shown subplot below.




```
plt.subplot(
)
```

Check Show answer

2) Complete the subplot() call to set the active subplot to the shown subplot below.




```
plt.subplot(          )
```

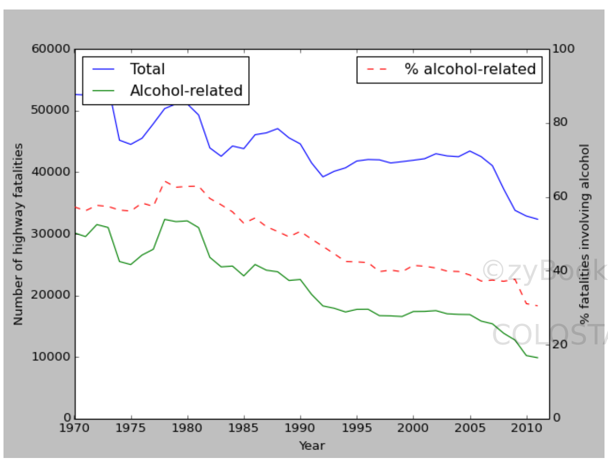
Check

Show answer

In some cases a second y-axis allows combining different types of data into the same plot, as long as the x-axis units are the same. The ***twinx()*** function creates a second axis on a plot.

Figure 33.5.4: Adding a second y-axis on the right side of a plot.

```
import matplotlib.pyplot as plt

with open('dd_stats.csv') as f:
    total_fatalities = []
    alcohol_fatalities = []
    percentages = []
    for line in f:
        total, alcohol = line.split(',')
        total_fatalities.append(int(total))
        alcohol_fatalities.append(int(alcohol))
        percentages.append(float(alcohol) / float(total) * 100)

years = range(1970, 2012)

figure = plt.figure()
left_axis = figure.add_subplot(1, 1, 1)
right_axis = left_axis.twinx()

left_axis.plot(years, total_fatalities, label="Total")
left_axis.plot(years, alcohol_fatalities, label="Alcohol-related")
right_axis.plot(years, percentages, 'r--', label="% alcohol-related")
right_axis.axis([1970, 2012, 0, 100])

left_axis.set_xlabel('Year')
left_axis.set_ylabel('Number of highway fatalities')
left_axis.legend(loc="upper left")
right_axis.set_ylabel('% fatalities involving alcohol')
right_axis.legend(loc="upper right")

plt.show()
```

The program above adds a new data series showing the percentage of fatalities related to alcohol for a given year. y-axis values of these percentages range from 0-100, while the y-axis values of fatalities range from 0-60000. A separate y-axis is required (otherwise the percentage data series would be indistinguishable from 0 once the plot is scaled).

figure.addsubplot() is called, which returns the subplot axis (the actual creation of the default subplot is not important or necessary). twinx() is called to create the right-side axis. left_axis and right_axis can then be used to set axis labels, plot data series, and enable legends.

PARTICIPATION ACTIVITY 33.5.3: Using multiple axes, subplots, and figures.

1) A 2nd y-axis is only useful if the x-axis values of all the data series in the plot are similar.
   - True
   - False

2) set_right_ylabel() creates the right-side axis.
   - True
   - False

33.6 LAB: Descending selection sort with output during execution

Write a program that takes an integer list as input and sorts the list into descending order using selection sort. The program should use nested loops and output the list after each iteration of the outer loop, thus outputting the list N-1 times (where N is the size of the list).

Ex: If the input is:

```
20 10 30 40
```


the output is:

```
[40, 10, 30, 20]
[40, 30, 10, 20]
[40, 30, 20, 10]
```

Ex: If the input is:

```
7 8 3
```

the output is:

```
[8, 7, 3]
[8, 7, 3]
```


Note: Use `print(numbers)` to output the list numbers and achieve the format shown in the example.

422102.2723990.qx3zqy7

LAB ACTIVITY | 33.6.1: LAB: Descending selection sort with output during execution | 0 / 10

main.py

Load default template...

```
# Type your code here.

```

Develop mode | Submit mode

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program

Input (from above) → **main.py** (Your program) → 0

Program output displayed here

Coding trail of your work What is this?

History of your effort will appear here once you begin working on this zyLab.

33.7 LAB: Sorting user IDs

Given code that reads user IDs (until -1), complete the quicksort() and partition() functions to sort the IDs in ascending order using the Quicksort algorithm. Increment the global variable num_calls in quicksort() to keep track of how many times quicksort() is called. The given code outputs num_calls followed by the sorted IDs.

Ex: If the input is:

```
kaylasimms
julia
myron1994
kaylajones
-1
```

the output is:

```
7
julia
kaylajones
kaylasimms
myron1994
```

422102.2723990.qx3zqy7

LAB ACTIVITY | 33.7.1: LAB: Sorting user IDs | 0 / 10

main.py Load default template...

```
# Global variable
num_calls = 0

```

Develop mode **Submit mode**

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program

Input (from above) → **main.py** (Your program) → 0

Program output displayed here

Coding trail of your work What is this?

```
History of your effort will appear here once you begin
working on this zyLab.
```


33.8 LAB: Insertion sort

This section's content is not available for print.

33.9 LAB: Merge sort

This section's content is not available for print.

33.10 LAB: Binary Search

This section's content is not available for print.