Assignment_1-code

January 19, 2024

1 Assignment 1 - Lab Exercises

1.1 1. Short Python Practice

1.1.1 Task: Create a class named Dog

- Each instance of the class should have a variable, name (should be set on creating one)
- The class has two main methods, action and age
- Action is an instance method taking in an *optional* parameter, quietly. Calling action prints "WOOF WOOF WOOF" if nothing is passed in, "woof" if quietly is passed in as True. (Use only one hardcoded string in the method)
- Age is a class method that takes in a list of ages in human years and returns it in dog years (Use list comprehension)

[1]:
```python
class Dog:

    def __init__(self,name):
        # complete the class
        self.name =name
        return None
    def action(self,quietly=True):
      if quietly == True:
        print("woof")
      else:
        print("WOOF WOOF WOOF")
    def age(self,ages_list):
      output = [a/7 for a in ages_list]
      return output
```

[2]:
```python
human_ages = [3, 4, 7, 4, 10, 6]
```

1. Create an instance of this class, and print its name
2. Call action with both possible options for the parameter
3. Call age on the provided array above

[3]:
```python
# complete the task above
dog_1 = Dog("Jack")
print(dog_1.name)
dog_1.action(True)
```

```
dog_1.action(False)
dog_1.age(human_ages)
```

```
Jack
woof
WOOF WOOF WOOF
```

```
[3]: [0.42857142857142855,
      0.5714285714285714,
      1.0,
      0.5714285714285714,
      1.4285714285714286,
      0.8571428571428571]
```

1.2 2. Numpy Practice

Numpy is a commonly used library in Python for handling data, especially helpful for handling arrays/multi-dimensional arrays. It's particularly important for helping bridge the ineffiencies of Python as a high level language with the improved performance of handling data structures in low level languages like C.

1.2.1 Part 1: General Numpy Array Exercises

1. Create a matrix, with shape 10 x 6, of ones
2. Create 10 matrices, with shape 50 x 20, of random integers from -5 to 5
3. Print the number of elements equal between a pair of matrices, for each possible pair in the 10 created above except for with itself (Don't compare the 1st with the 1st)

```
[ ]: import numpy as np

     # example
     die_roll = np.random.randint(1, 7) #note that the high value is exclusive
     print("Rolled a " + str(die_roll))
```

```
Rolled a 2
```

```
[ ]: #complete part 1
     matrix= np.ones([10,6])
     print(matrix, np.shape(matrix))
     mat = np.zeros([10,50,20])
     for i in range(10):
       mat[i,:,:] = np.random.randint(-5,6,[50,20])

     print("An example matrix: ",mat[0,:,:])

     num_equal = 0
     for i in range(10):
       for j in range(i+1,10):
```

```python
    cond = mat[i,:,:] == mat[j,:,:]
    num_equal += np.sum(cond)
print("Number of equal random numbers:",num_equal)
```

```
[[1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1.]] (10, 6)
An example matrix:  [[-4.  4. -3.  0.  5.  2.  5.  4. -3.  4.  4.  3.  2. -5.
1. -3.  5.  3.
  -5.  4.]
 [ 1. -2.  2.  3.  1.  5.  3.  4.  4. -5.  2.  2.  3.  1.  0.  0.  1. -4.
   1.  4.]
 [ 5.  2. -3.  4. -3. -3. -4.  1.  4.  4.  1. -2. -2. -5. -3.  1.  0. -5.
  -2. -3.]
 [-5. -5.  4.  5. -2. -3. -4.  1.  3.  3.  4.  2. -4.  2. -3.  0. -3.  4.
  -5.  3.]
 [ 4. -1. -2. -5.  2.  4. -4. -3.  5.  2. -4. -2. -3. -2. -3.  5.  2.  0.
  -3.  0.]
 [ 2. -4. -1.  0. -1.  4. -5.  4.  1. -2. -5.  3.  0. -1.  3. -2. -2. -3.
  -3.  5.]
 [ 4.  1. -5.  4.  1.  1. -2. -1. -1. -2.  0. -2.  4.  3. -4. -5.  5.  2.
  -5.  1.]
 [ 0. -3.  4.  2. -1. -2. -4.  0. -2.  1. -3.  4.  3.  2.  2. -5. -3. -4.
  -5.  5.]
 [-3. -3.  4.  4.  3.  5. -5.  1.  5. -2.  0.  0. -4. -5.  5. -5. -5.  5.
  -1. -2.]
 [-1.  5. -2.  0.  0.  2.  3.  3.  1. -1.  4.  3. -2.  5. -5.  1.  3.  5.
  -4.  1.]
 [ 0. -1. -2.  1.  3. -2.  1. -2.  2.  5.  5. -2. -5.  4.  1.  4. -4.  2.
   1. -1.]
 [ 3.  2. -4.  1.  5. -2.  4. -4.  5. -2.  3.  5. -2. -4. -3.  1.  5. -4.
   3. -4.]
 [ 5.  2. -3.  1. -4.  5.  5.  0.  3. -5.  3.  3. -4.  1.  4.  3. -5.  1.
   2. -3.]
 [ 3. -2.  0. -4. -4. -4. -1.  1. -2. -2.  2. -1.  3.  5.  1.  1.  0. -4.
   2. -1.]
 [ 0. -5.  2. -4. -3.  2.  4.  2. -5.  2.  0. -4. -5.  0.  1. -3. -4. -2.
  -4. -4.]
 [ 5. -1.  5. -2.  0.  4.  1. -3. -1.  2.  4.  0.  4. -1. -1.  1.  5. -5.
  -4. -1.]
```

```
[-2.  1.  0.  5.  5. -4.  1.  2. -3.  2. -2.  1. -4.  1. -3.  4. -5. -2.
  3.  5.]
[-4. -1.  3. -5. -2. -4. -3.  0. -3.  3.  1. -2. -5. -1.  1.  0.  4.  0.
 -1. -2.]
[ 5. -5. -1.  2.  2. -3.  0. -2.  2. -1. -5. -3. -5.  4.  1. -3.  3. -3.
 -3.  0.]
[-2.  2.  4.  0.  1.  2.  1.  3.  5.  4. -3.  3. -2.  3.  5. -1. -2. -5.
  2.  1.]
[ 3.  0.  3. -2.  3.  0.  3.  1. -5. -4.  2.  4.  2.  2.  1.  2. -5.  2.
 -1.  4.]
[ 3. -3.  3.  5. -4.  2.  1. -5. -3. -1.  1. -2. -5.  5. -5.  2. -3. -5.
 -2. -3.]
[ 5. -5. -5.  3.  5.  5.  3. -1.  2.  2. -5.  4.  0. -3.  0.  5. -1. -4.
 -1.  3.]
[-1. -3.  0.  1.  3.  2. -2.  5.  0. -4.  1. -3. -4. -1.  5. -2.  4. -1.
  5. -3.]
[ 1. -3.  4.  4.  3. -3. -5.  1.  1.  0. -5.  0. -2.  3.  4. -4. -1.  0.
 -4.  4.]
[ 2.  1.  0.  0.  5. -3.  4.  2.  5.  5.  0.  1.  5.  1.  0.  2. -3.  3.
 -3. -3.]
[-3.  0.  2.  3.  2.  0. -1.  2.  1.  4. -5.  3. -3.  4. -5.  5. -4.  4.
  3.  0.]
[ 2.  4.  2.  0. -2.  0. -2. -1.  1. -2. -3. -5.  0.  0.  4.  2. -2.  5.
  5. -3.]
[ 3. -3.  2.  1. -4.  1.  5.  2.  5.  5.  0. -2.  4.  3.  2.  2. -5. -2.
 -2. -3.]
[ 3. -2. -4. -1. -3. -4.  0. -1. -3.  2.  1.  1.  1.  0. -5. -5.  2. -3.
 -3. -5.]
[ 5.  5.  1.  2.  4.  0. -5.  2. -4. -2. -4.  4. -1.  2.  0.  2. -2.  4.
  2.  1.]
[ 2. -2.  3.  2.  3.  2. -3.  0.  2.  2. -4.  5. -3.  4.  3.  4. -5.  2.
  1.  5.]
[ 4. -2.  3.  2.  5.  0.  0.  5. -5.  0.  3.  5.  0.  4. -4.  1.  3.  2.
 -4. -4.]
[ 5. -2. -3. -2.  2.  3. -2.  3.  1.  5.  3.  5.  3.  5. -5.  0. -4.  5.
 -4.  5.]
[ 0. -2. -4.  3.  1.  0.  5. -1. -2.  2. -4.  5.  0.  0. -2. -3.  5.  3.
 -2. -4.]
[ 3. -2. -5. -5.  1.  2. -2. -1.  3. -5.  5.  5. -4. -1.  3. -1. -3. -5.
 -1.  4.]
[ 4.  5. -4.  0. -1. -2.  2. -4.  4. -4.  5.  1. -1. -4. -1.  2.  5.  4.
  5. -4.]
[ 5.  2.  0.  5.  4.  0. -5. -3. -4. -5.  2.  5. -3. -1. -2.  1.  2. -1.
  5.  5.]
[-5. -5.  2. -4. -3.  0.  2. -1. -4. -3. -4. -4. -4.  1.  0.  1.  3.  5.
  5. -1.]
[ 0.  0. -2.  5. -1. -1. -4.  1. -2. -4.  3.  3. -3.  3.  3. -1. -2.  3.
  2. -3.]
```

```
[ 4. -1.  0.  2.  2. -2.  2. -1.  2.  5. -3.  1. -4. -1. -2. -2.  4.  3.
 -3.  3.]
[-2. -5. -1.  4.  3.  4. -5.  5.  3.  3.  2.  0. -5.  3.  4. -2.  2.  4.
  2. -5.]
[-4. -4.  4.  3. -5.  2. -4. -1. -4.  2.  4. -2.  4.  2. -2. -4.  0. -5.
  2.  0.]
[-5.  1. -4.  0. -3.  3.  1.  5. -3.  0. -1.  3.  3.  5.  0.  3.  5.  0.
  4. -3.]
[ 1.  1. -2.  5.  0. -1.  3.  3.  3. -3.  0. -3. -5.  5. -4.  3.  3.  3.
  2. -2.]
[ 1.  4. -5.  2.  4.  2. -5.  2. -4.  1.  2.  0. -5. -2. -1.  0. -2. -4.
 -4. -4.]
[-5. -1. -3. -2. -2. -5. -4.  4. -1.  0.  3. -1. -4.  1. -1. -5.  2. -5.
  3. -2.]
[ 2. -2.  4.  2.  0.  5.  5.  1.  1.  4.  5. -3. -1.  5.  0.  5. -2. -5.
 -3. -4.]
[-2. -3.  2. -1. -2.  1. -3.  3.  3. -5. -1.  1.  3.  3. -3. -5.  1. -1.
 -1. -2.]
[ 4.  1. -1.  3. -3.  3. -2. -2.  1.  1.  5. -3. -2. -5.  2. -3. -2. -2.
  3.  3.]]
Number of equal random numbers: 4142
```

1.2.2 Part 2: Splicing and some Numpy Methods

1. Find the pseudoinverse of one of the matrices from Part 1.2
2. Turns out the last 40 rows and last 10 columns of data were useless. Set a new variable to the matrix used above, without the last 40 rows or 10 columns.
3. Find the inverse for this square matrix.

Optional: You can use the same command for 1 and 3 (briefly explain why if you do)

[ ]:
```python
# complete part 2
print("The shape of the matrix is: ",np.shape(mat[0,:,:]))
pseudo_inverse = np.linalg.pinv(mat[0,:,:])
print("The shape of the mp inverse is: ",np.shape(pseudo_inverse))
sub_mat = mat[0,:10,:10]
print("The shape of the sub matrix is: ",np.shape(sub_mat))
sub_mat_inv = np.linalg.inv(sub_mat)
```

```
The shape of the matrix is:  (50, 20)
The shape of the mp inverse is:  (20, 50)
The shape of the sub matrix is:  (10, 10)
```

1.2.3 Part 3: Matrix Methods

1. Generate 2 more matrices, *a* and *b* with shape 2 x 3 and 4 x 3.
2. Create 2 matrices, *a_on_b* and *b_on_a* - for the first, stack a on top of b, and the opposite for the second. The join them to create *abba*, with *b_on_a* to the right of *a_on_b*
3. Print the right eigenvalues and eigenvectors for *abba*

```
[24]: # complete part 3
a = np.random.randint(0,4,[2,3])
b = np.random.randint(0,4,[4,3])

a_on_b = np.vstack((a,b))
b_on_a = np.vstack((b,a))

print(np.shape(a),np.shape(b),np.shape(a_on_b))
print(a_on_b)
print(b_on_a)
abba = np.concatenate((a_on_b,b_on_a),axis=-1)
print(np.shape(abba))
print(abba)
eig_values, eig_vectors = np.linalg.eig(abba)
print("Eigenvalues are", eig_values)
print("Eigenvectors are",eig_vectors)
```

```
(2, 3) (4, 3) (6, 3)
[[0 0 3]
 [2 1 1]
 [3 2 1]
 [3 0 0]
 [0 0 2]
 [0 2 2]]
[[3 2 1]
 [3 0 0]
 [0 0 2]
 [0 2 2]
 [0 0 3]
 [2 1 1]]
(6, 6)
[[0 0 3 3 2 1]
 [2 1 1 3 0 0]
 [3 2 1 0 0 2]
 [3 0 0 0 2 2]
 [0 0 2 0 0 3]
 [0 2 2 2 1 1]]
Eigenvalues are [ 7.52148902+0.j         -4.57141911+0.j
0.50814079+2.79657403j
  0.50814079-2.79657403j  0.56268542+0.j         -1.52903691+0.j        ]
Eigenvectors are [[-0.47113201+0.j         -0.5977217 +0.j
-0.10207052+0.28808591j
  -0.10207052-0.28808591j  0.30357288+0.j          0.05640412+0.j        ]
 [-0.39007464+0.j         -0.15981348+0.j          0.56290445+0.j
   0.56290445-0.j         -0.12735334+0.j         -0.44781289+0.j        ]
 [-0.46626591+0.j          0.48945736+0.j          0.07388609-0.30532518j
   0.07388609+0.30532518j  0.693425  +0.j         -0.0142596 +0.j        ]
 [-0.37844585+0.j          0.53212463+0.j          -0.04887159+0.43445244j
  -0.04887159-0.43445244j -0.4149591 +0.j           0.34466223+0.j        ]
 [-0.29293879+0.j         -0.01260979+0.j          -0.46576303-0.13635747j
  -0.46576303+0.13635747j -0.09247663+0.j          -0.72934505+0.j        ]
 [-0.42360136+0.j         -0.30709003+0.j          -0.00103721-0.25372641j
  -0.00103721+0.25372641j -0.47962842+0.j           0.38123824+0.j        ]]
```

1.3 3. Matplotlib Practice

Matplotlib is a commonly used visualization library.

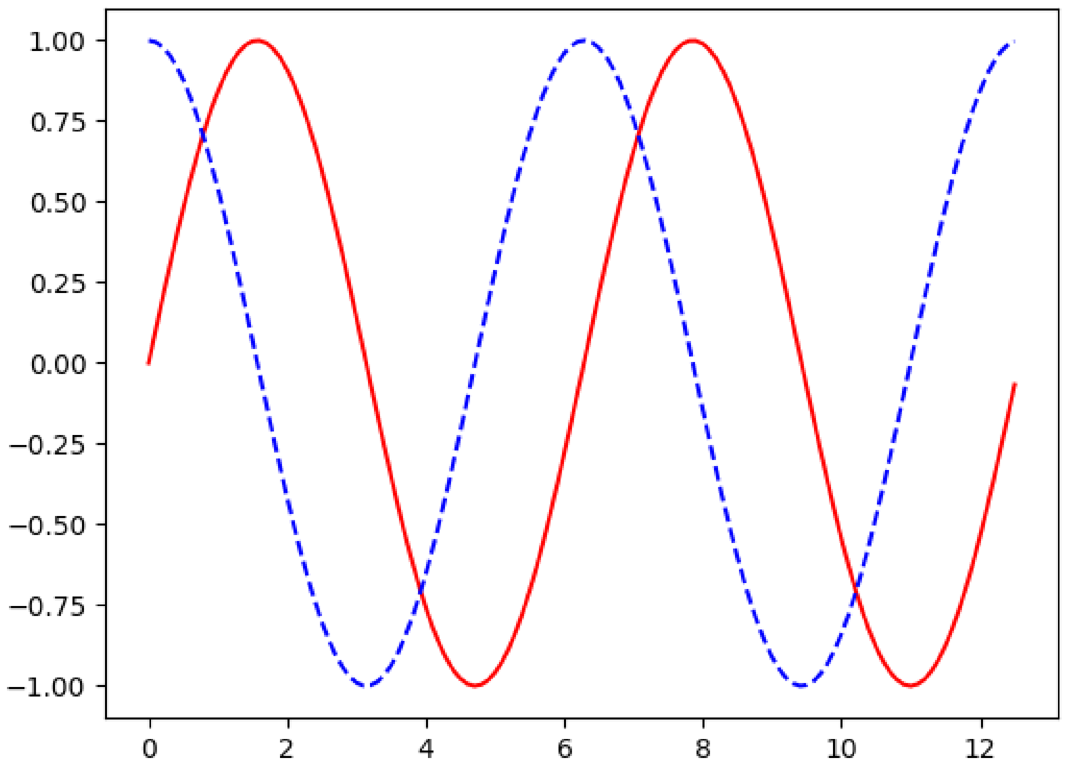
1.3.1 Part 1: Generate plots

1. Generate sine (y_sin) and cosine (y_cos) data for x from 0 to 4 * pi (Use np.arange with step size 0.1)
2. Create a plot with both on the same chart. The sine wave should be solid, cosine dashed
3. Create a second plot with 2 subplots, with the first plotting the sine wave and second plotting the cosine wave

[25]:
```
import matplotlib.pyplot as plt
```

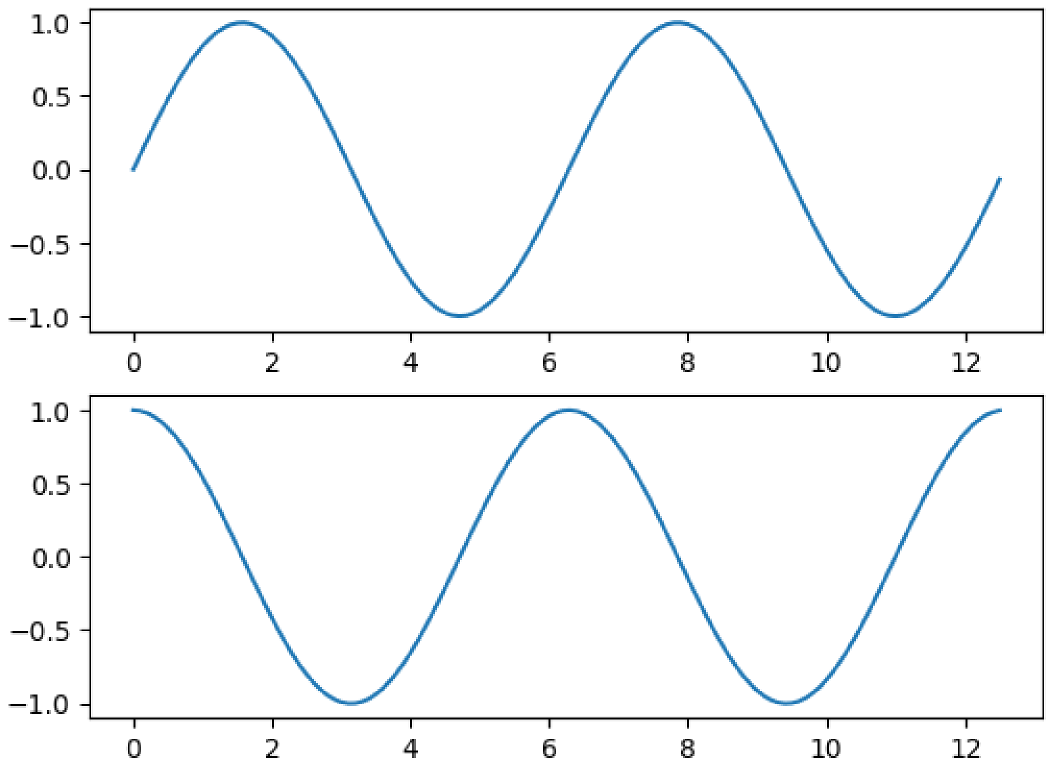
[26]:
```
# complete 1
x = np.arange(0,4*np.pi,0.1)
y_sin = np.sin(x)
y_cos = np.cos(x)
```

[31]:
```
# complete 2
fig_1 = plt.figure()
plt.plot(x,y_sin,'r')
plt.plot(x,y_cos,'b--')
plt.show()
```



```
[32]: # complete 3
      fig_2 = plt.figure()
      plt.subplot(2,1,1)
      plt.plot(x,y_sin)
      plt.subplot(2,1,2)
      plt.plot(x,y_cos)
```

```
[32]: [<matplotlib.lines.Line2D at 0x7f09c18f6080>]
```



1.3.2 Part 2: Save Output

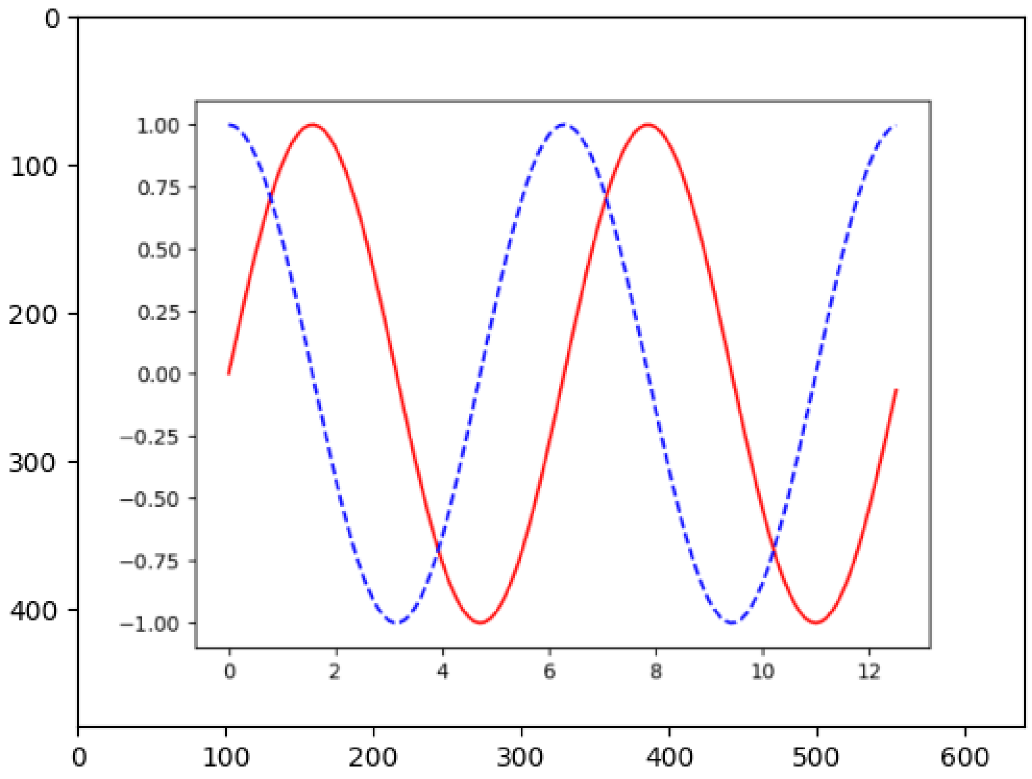
1. For the above plots, save each as a image file. (Hint: Go back and create variables for each at the start with *var = plt.figure()*)
2. Open both files in this notebook

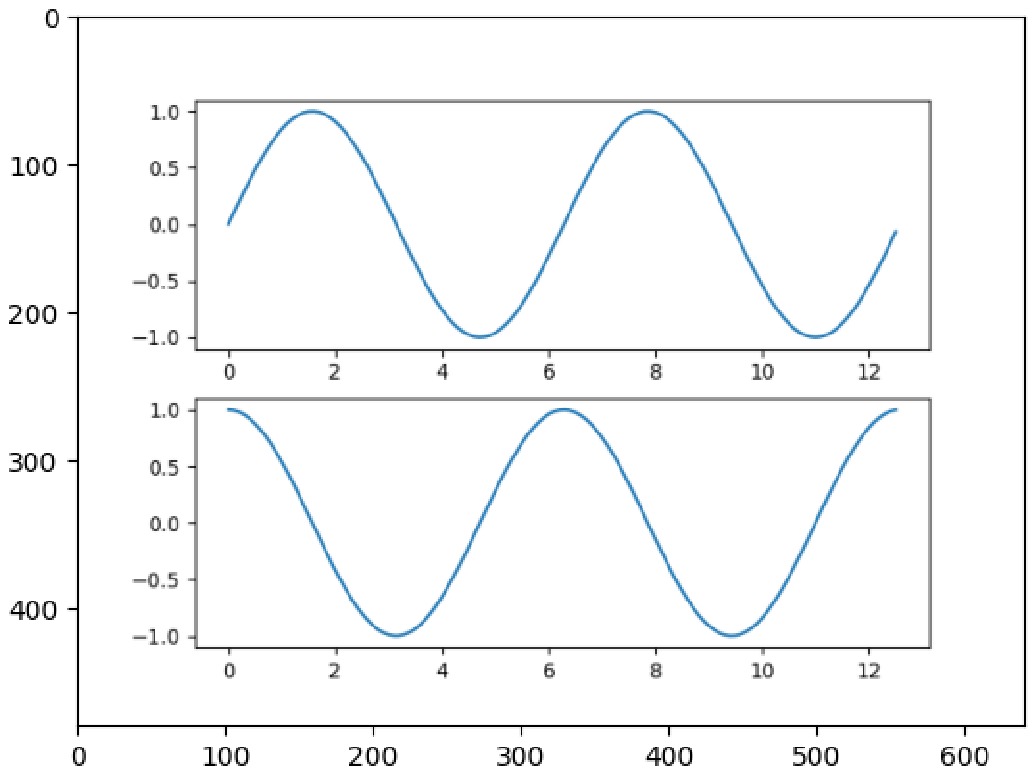
```
[35]: # complete 1
      fig_1.savefig("fig1.png")
      fig_2.savefig("fig2.png")
```

```
[37]: from IPython.display import Image
```

```
[40]: # complete 2
      image_1 = plt.imread("fig1.png")
      plt.imshow(image_1)
      plt.figure()
      image_2 = plt.imread("fig2.png")
      plt.imshow(image_2)
```

```
[40]: <matplotlib.image.AxesImage at 0x7f09c1a63ee0>
```


0
100
200
300
400
500
600
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
2
4
6
8
10
12


0
100
200
300
400
500
600
1.0
0.5
0.0
−0.5
−1.0
0
2
4
6
8
10
12