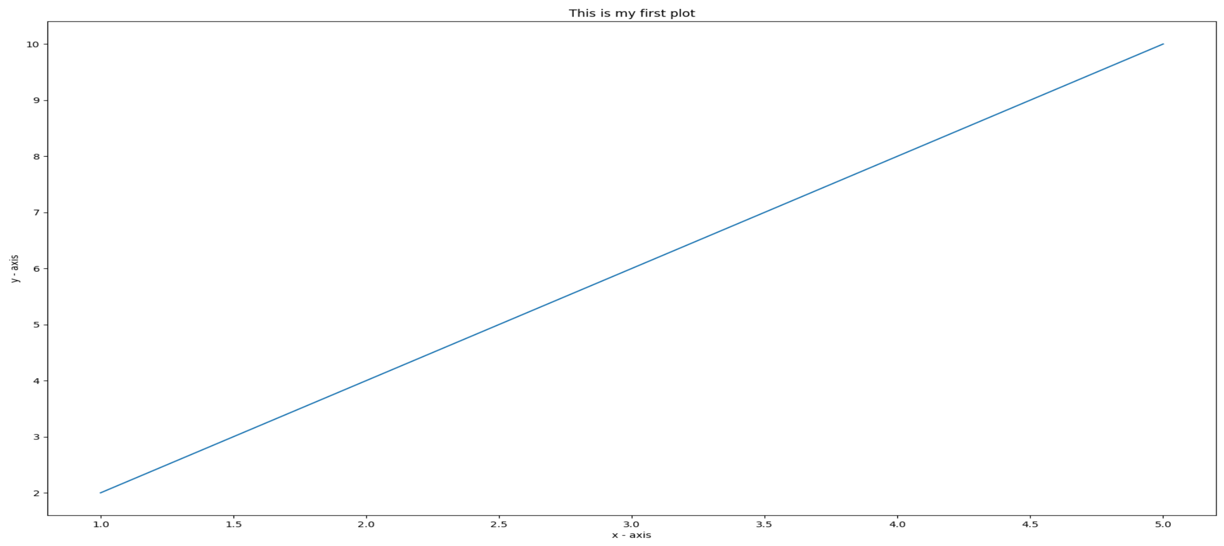
ENGR 13300
Python Project Resource – Image IO

Plotting:

Matplotlib is a Python library that helps in visualizing and analyzing the with the help of graphical, pictorial visualizations that can be simulated. The following code can help in understanding the plotting using matplotlib:

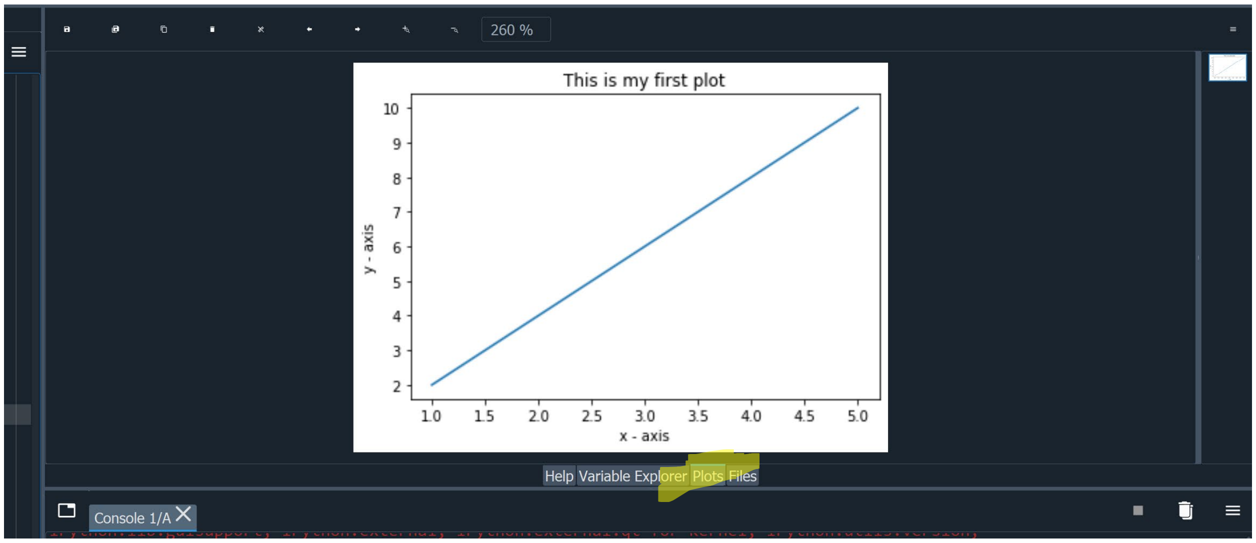
```
import matplotlib.pyplot as plt

# Data for plotting
x = [1,2,3,4,5]
y = [2,4,6,8,10]

#plotting the points
plt.plot(x, y)

#naming the x axis
plt.xlabel('x - axis')

#naming the y axis
plt.ylabel('y - axis')

#Title of the plot
plt.title('This is my first plot')

#function to show the plot
plt.show()
```

Output:



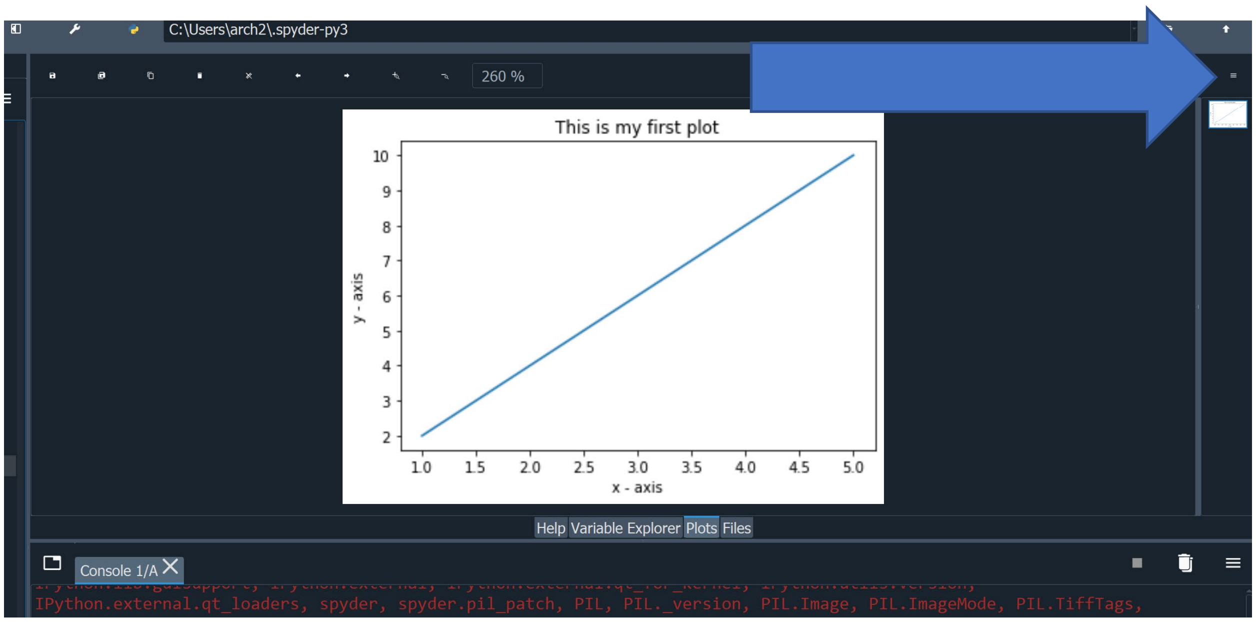
Plot settings:

1. Plots are always displayed (if you have any) in the plots tab. To look at the plots displayed in the plots tab (above the console), select the “Plots” tab:

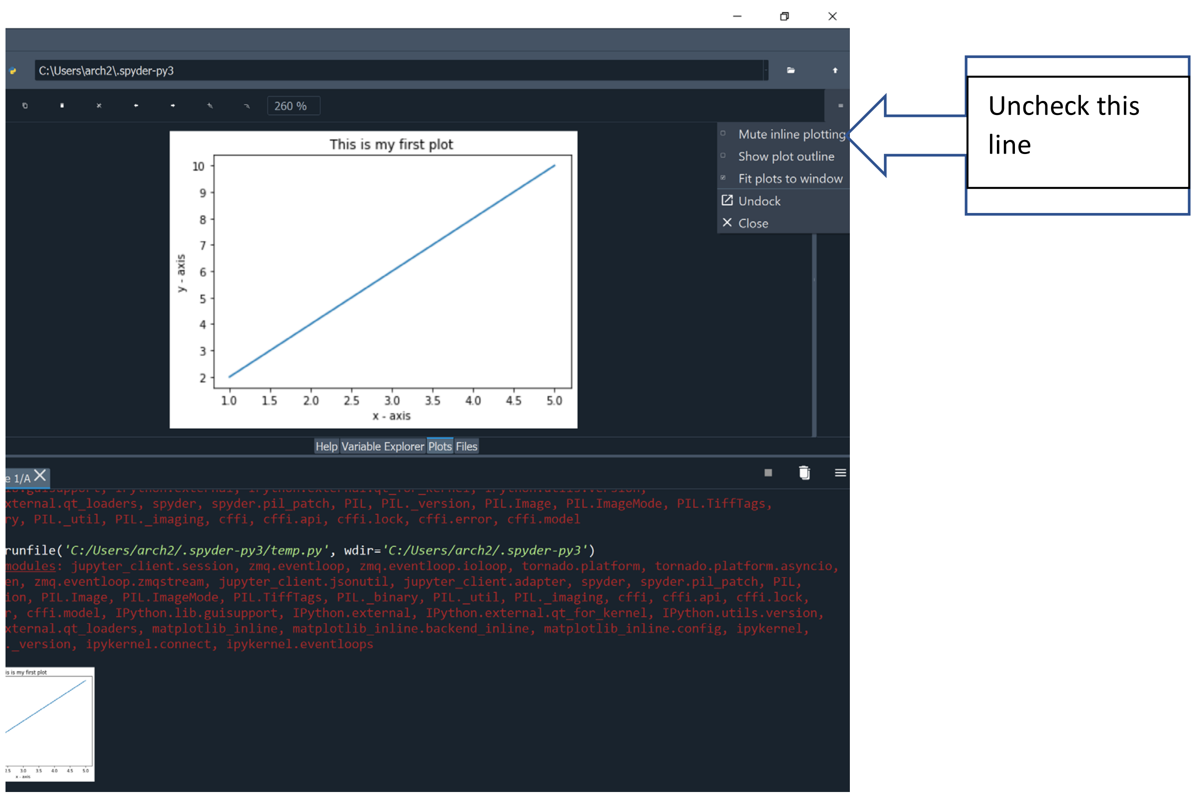


2. If you want your plots to be displayed inline (in the console):

   Tools -> Preferences -> Ipython Console -> Graphics and under Graphics Backend select “inline”. Also, make sure to uncheck “mute inline plotting” under the plots tab.


C:\Users\arch2\.spyder-py3
260 %
This is my first plot
y - axis
x - axis
Help Variable Explorer Plots Files
Console 1/A

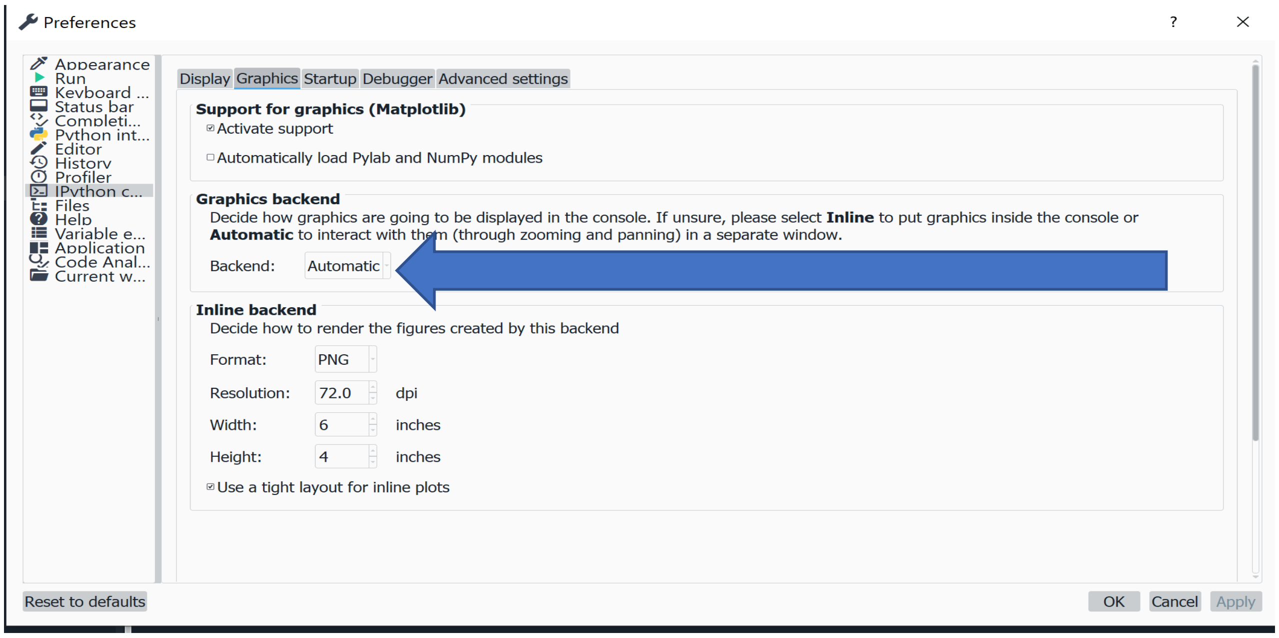
```
IPython.external.qt_loaders, spyder, spyder.pil_patch, PIL, PIL._version, PIL.Image, PIL.ImageMode, PIL.TiffTags,
```



```
xternal.qt_loaders, spyder, spyder.pil_patch, PIL, PIL._version, PIL.Image, PIL.ImageMode, PIL.TiffTags,
ry, PIL._util, PIL._imaging, cffi, cffi.api, cffi.lock, cffi.error, cffi.model

runfile('C:/Users/arch2/.spyder-py3/temp.py', wdir='C:/Users/arch2/.spyder-py3')
modules: jupyter_client.session, zmq.eventloop, zmq.eventloop.ioloop, tornado.platform, tornado.platform.asyncio,
en, zmq.eventloop.zmqstream, jupyter_client.jsonutil, jupyter_client.adapter, spyder, spyder.pil_patch, PIL,
ion, PIL.Image, PIL.ImageMode, PIL.TiffTags, PIL._binary, PIL._util, PIL._imaging, cffi, cffi.api, cffi.lock,
r, cffi.model, IPython.lib.guisupport, IPython.external, IPython.external.qt_for_kernel, IPython.utils.version,
xternal.qt_loaders, matplotlib_inline, matplotlib_inline.backend_inline, matplotlib_inline.config, ipykernel,
._version, ipykernel.connect, ipykernel.eventloops
```

3. If you want your plots to pop up in a different window:

   `Tools -> Preferences -> Ipython Console -> Graphics and under Graphics Backend select "Automatic".`



Support for graphics (Matplotlib)

☒ Activate support

☐ Automatically load Pylab and NumPy modules

Graphics backend

Decide how graphics are going to be displayed in the console. If unsure, please select **Inline** to put graphics inside the console or **Automatic** to interact with them (through zooming and panning) in a separate window.

Backend: Automatic

Inline backend

Decide how to render the figures created by this backend

Format: PNG

Resolution: 72.0 dpi

Width: 6 inches

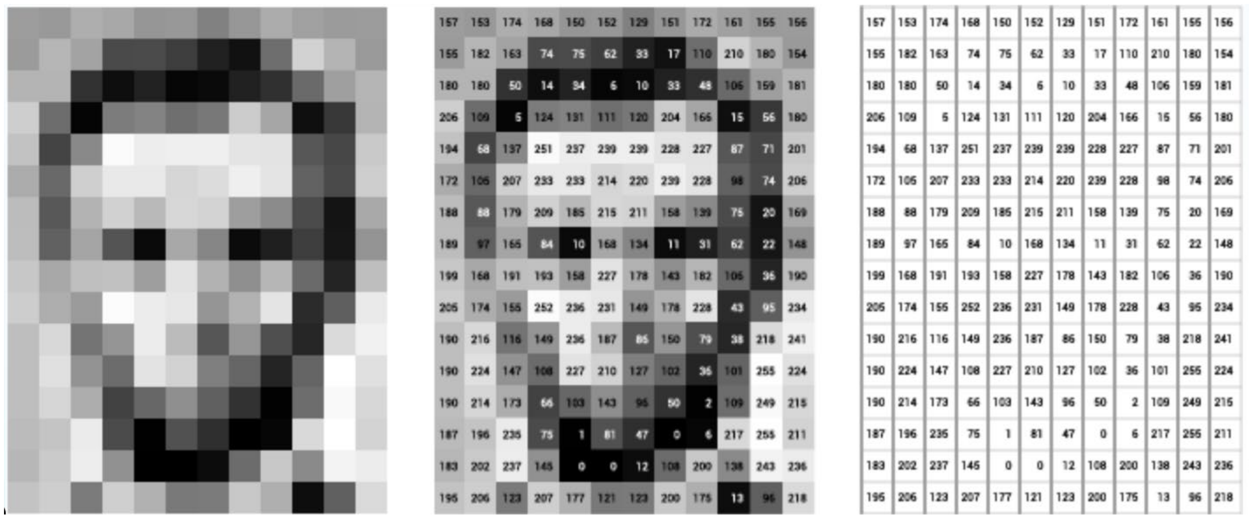
Height: 4 inches

☒ Use a tight layout for inline plots

Digital Images:

Before any processing can be performed on an image, a digital representation of it must be obtained. Digital images can be represented by a matrix of numbers. Each element of the matrix is called a picture element or pixel.

In the below black and white image, the image is represented by a matrix of 12 pixels wide by 16 pixels high. In each pixel is a value representing the amount of energy of the pixel. Pixel values will range from 0 to 255, with 0 being black and 255 being white.



157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

Figure 1. Digital Image (Source: Image Processing and Analysis Tutorial – Part II, Edward J. Delp, 8/29/2020, slide 2)

A digital image (such as a `.png` file) is a matrix of numbers with a few additional numbers added to the beginning.

Color Images:

The Human Visual System has rods and cones which are sensitive to light and color. There are three types of cones which are sensitive to a particular color range of light: red, green, and blue. Any color is a weighted sum of red, green, and blue light. However, there is more than one way to represent the colors, which are known as color spaces. Examples of color spaces are the Red, Green, Blue (RGB) and YUV (luminance/chrominance) color spaces. In contrast to black and white (B&W) images, a color image is represented by multiple matrices, one for each of the components of the color space. For RGB images, there are three matrices: Red, Green, and Blue. Like B&W images, each location in the matrix represents one pixel. When all three matrices are combined, they generate a color in one pixel.

Reading Image files:

In order to work with images, the image has to be first read by your program as a matrix of Blue, Green, and Red pixel values. To read an image located in the same directory as your program is located, you may use the following syntax

1. If you are importing a jpg file using matplotlib:

```
import matplotlib.pyplot as plt
image = plt.imread('Sample.jpg')
```

2. If you are importing a png file using matplotlib:

```
import matplotlib.pyplot as plt
image = plt.imread('Sample.png')
image = (image*255).astype(numpy.uint8)
```

3. If you are importing a tiff file using matplotlib:

```
import matplotlib.pyplot as plt
image = plt.imread('Sample.tiff')[:,:,:3]
```

The `image` will be a three-dimensional array.

1. To access the intensity value of each pixel in different color channels:

```
Red_pixel_value = image[row_index,column_index,0]
Green_pixel_value = image[row_index,column_index,1]
Blue_pixel_value = image[row_index,column_index,2]
```

Showing Image files:

In order to display image files to the user,

1. With matplotlib: The following snippet of code will display the image in Spyder console:

```
import matplotlib.pyplot as plt
#Read and Process image file
plt.imshow(image)

#Read and Process image file using "gray" color map
plt.imshow(image, cmap='gray')
```

Writing Image files:

To write an image back in the same directory as your program is located, you may use the following syntax,

1. With matplotlib:

```
import matplotlib.pyplot as plt
#Read and Process image file
plt.imsave('Sample_copy.tiff',image)

#Read and Process image file using “grey” color map
plt.imsave('Sample_copy.tiff ', image, cmap='gray')
```

Important Notes:

- For reading or writing images located in a different folder, you will have to use the complete path to the image (ex. `'C:\...\Sample.jpg'`)
- The image arrays are stored as an array of pixel data, where each (x,y) coordinate has three values associated to the brightness of Red, Blue, and Green (RBG) colors. The brightness ranges from 0 being dark to 255 being bright, and values have to strictly be integers. **However, `matplotlib` rescales the 8 bit data from each channel to floating point data between 0.0 and 1.0. if you import an PNG (.png) image. You will need to convert it back to unsigned integer ranges from 0 to 255.**
- Although different libraries share similar syntax and data types, and thus gives the freedom to use functions of different libraries interchangeably, use caution as image standards used may differ. For instance,
    - `matplotlib` stores images in `numpy` Array, whereas other tools do not. As noted above, it also rescales the 8 bit values to floating point data.
    - `matplotlib` stores each pixel value as RGB, while OpenCV, a popular image processing library, stores pixels as BGR.

Sample programs:

1. Read an image, display its dimension and show the image

```
import matplotlib.pyplot as plt

image = plt.imread('sample.jpg')
print(f'Dimensions of the image is {image.shape}')
plt.imshow(image)
```

Working with Matrices in Python: https://www.programiz.com/python-programming/matrix

Useful Libraries:

- `matplotlib`
- `numpy`

Python does not have a built-in type for matrices, but a matrix can be expressed as a list of a list.

Example: 3 x 4 Matrix

$$\begin{array}{cc} \textbf{4 columns} & \\ \begin{bmatrix} 127 & 126 & 80 & 60 \\ 120 & 110 & 65 & 50 \\ 100 & 80 & 50 & 30 \end{bmatrix} & \textbf{3 rows} \end{array}$$

In Python:

A = [[127, 126, 80, 60],

[120, 110, 65, 50],

[100, 80, 50, 30]]

To transpose this matrix, we would need to use nested loops:

```
A_t = [[0,0,0,0],
       [0,0,0,0],
       [0,0,0,0]]
# iterate through rows
for i in range(len(A)):
   # iterate through columns
   for j in range(len(A[0])):
       A_t[j][i] = A[i][j]
```

However, it is much easier to perform matrix operations using NumPy N-dimensional array objects:

```
import numpy as np
A = np.array([[127,126,80,60],[ 120,110,65,50],[ 100,80,50,30]])

#To access rows
print(f’A[0] = {A[0]}’) # First row
print(f’A[2] = {A[2]}’) # Third row

# To access columns
print(f’A[:,0] = {A[:,0]}’) # First column
print(f’A[:,3] = {A[:,3]}’) # Fourth column


# Transpose using numpy
```

```
A_t = A.transpose()
print(A_t)
```

Will output:

```
A[0] =  [127 126 80 60]
A[2] =  [100 80 50 30]
A[:,0] =  [127 120 100]
A[:,3] =  [60 50 30]
[[127 120 100]
 [126 110  80]
 [ 80  65  50]
 [ 60  50  30]]
```