08-plotting

December 5, 2019

Contents

1 Plotting with matplotlib

Documentation of matplotlib for version 3.x is located here

Run the cell below to check the version of the matplotlib used in this notebook.

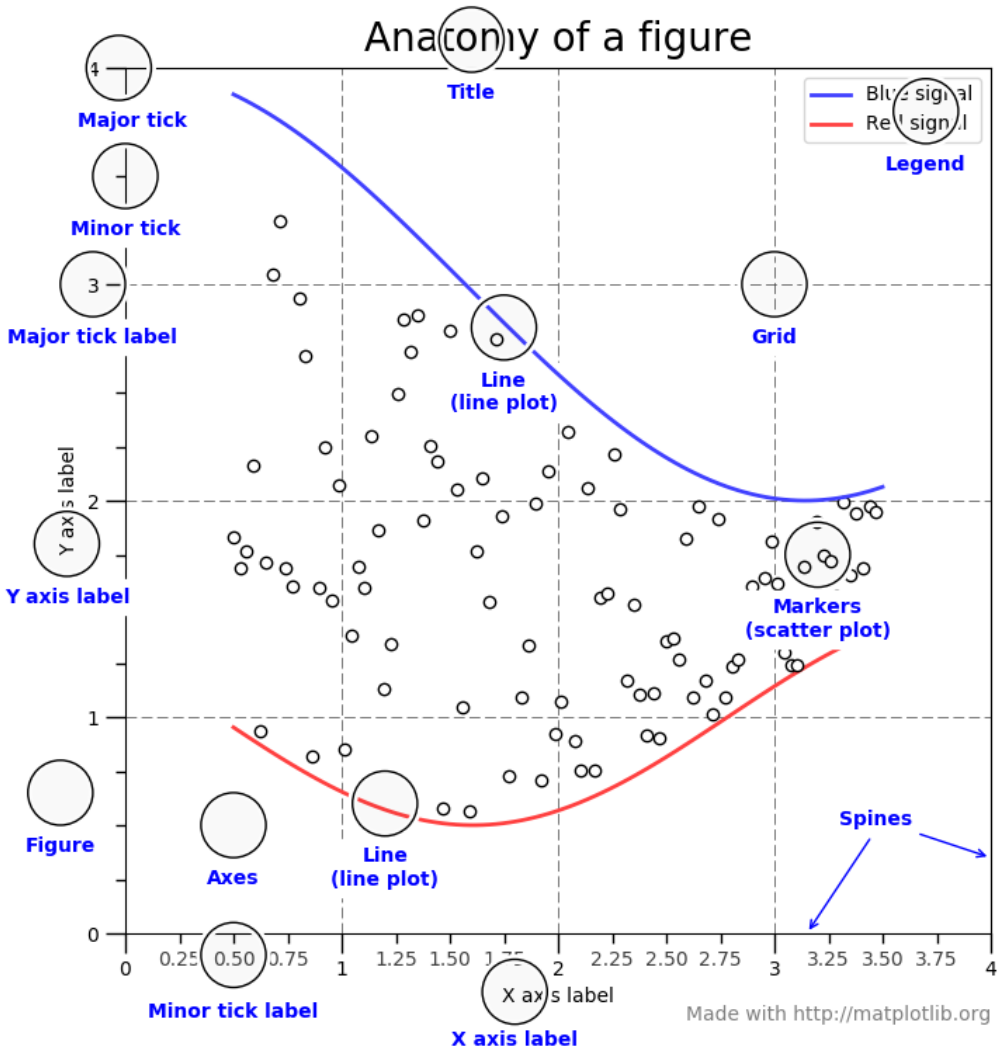
[1]:
```
%load_ext watermark
%watermark  -p matplotlib
```

```
matplotlib 3.1.1
```

We alse need to run the following cell once so that the output of plotting function is displayed within notebook environment.

[2]:
```
%matplotlib inline
```

Here's the anatomy of a figure (taken from matplotlib documentation). As you can see, a figure is composed of many components, thus we need access to almost all of components to generate a correct, aesthetic and reproducible figure.



Let's start drawing a plot. We need to import the matplotlib and its pyplot module. This needs to be done once per session but you might notice it has been called many times. Also, for practical reasons, the pyplot object will be called `plt` and this is just a norm, you can name it anything you like.

`Matplotlib` is the whole package; `matplotlib.pyplot` is a module in matplotlib

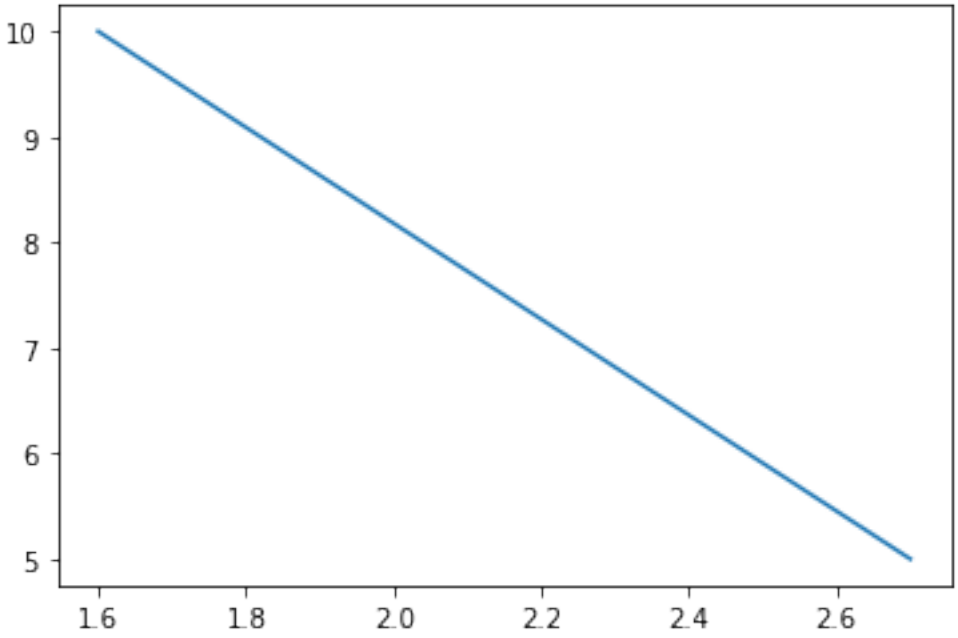
import matplotlib.pyplot as plt plt.plot([1.6, 2.7])

Looks like `plot` function accepts a list (or array) and plots it. The list is accepted as y values and x values are automatically assigned 0 and 1.

But, we can provide two lists as `x` and `y` values.

[3]:
```
import matplotlib.pyplot as plt
plt.plot([1.6, 2.7],[10,5])
```

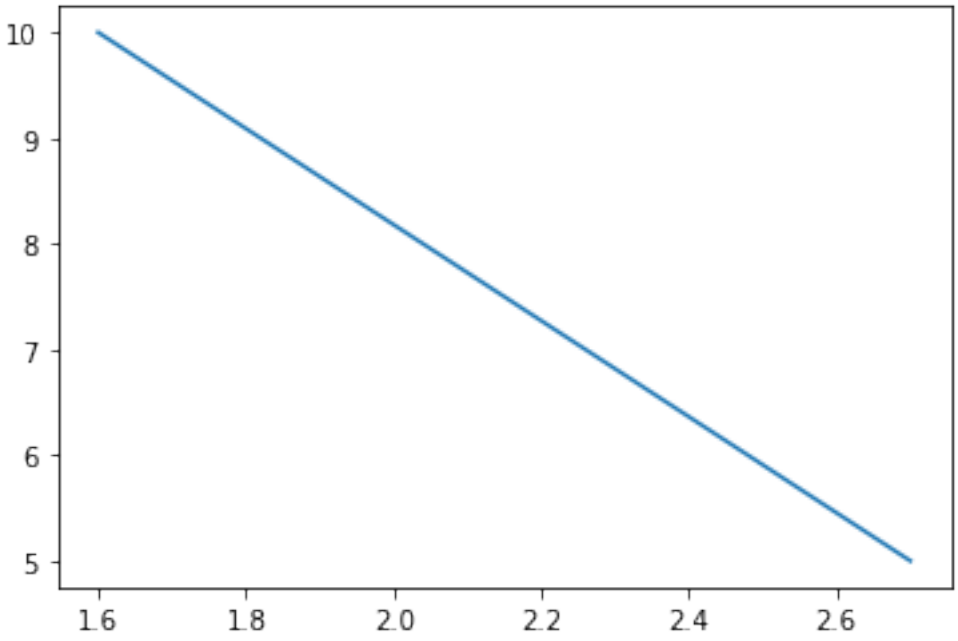
[3]: `[<matplotlib.lines.Line2D at 0x7f24b82a2780>]`



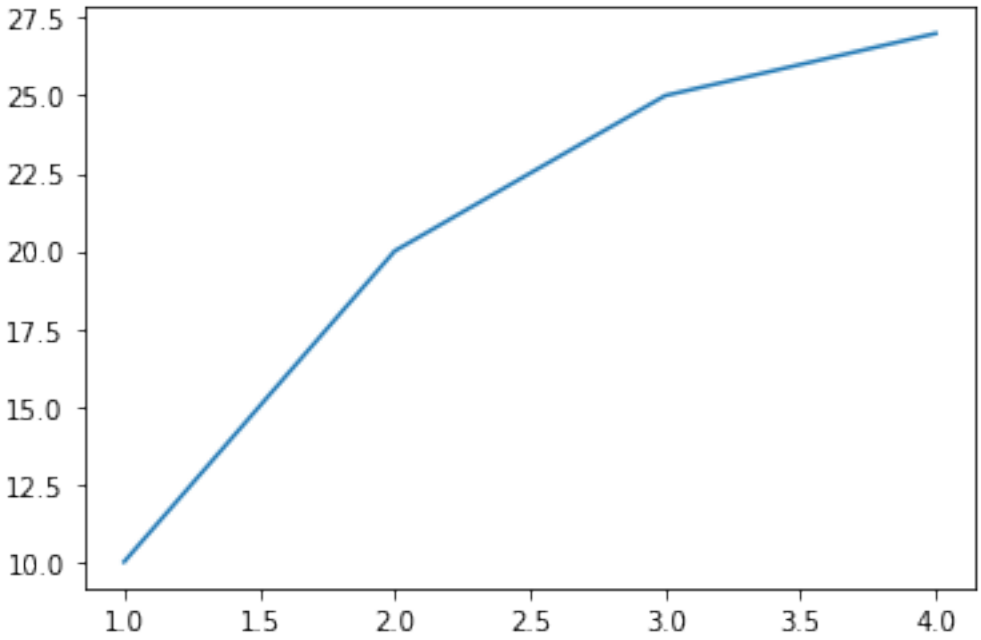
Notice that `plot` function is printing the object itself and notebook is showing the image. For clearer output let's use `show()` function.

In other environments, you **need* to use `show` function to see the resulting plot.

[4]:
```
plt.plot([1.6, 2.7],[10,5])
plt.show()
```

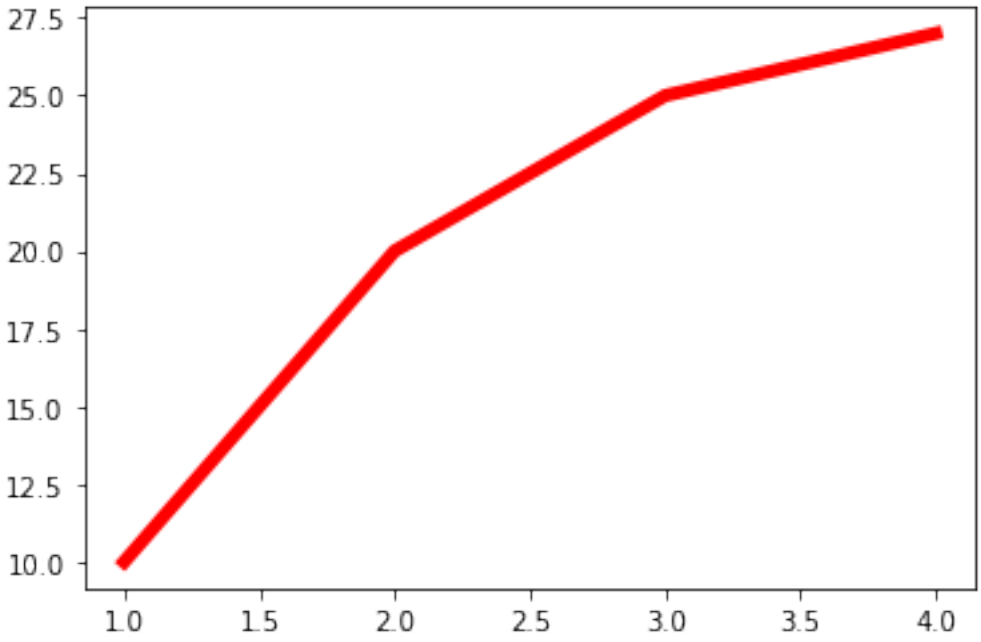


[5]:
```
x = [1,2,3,4]
y = [10,20,25,27]
plt.plot(x,y)
plt.show()
```



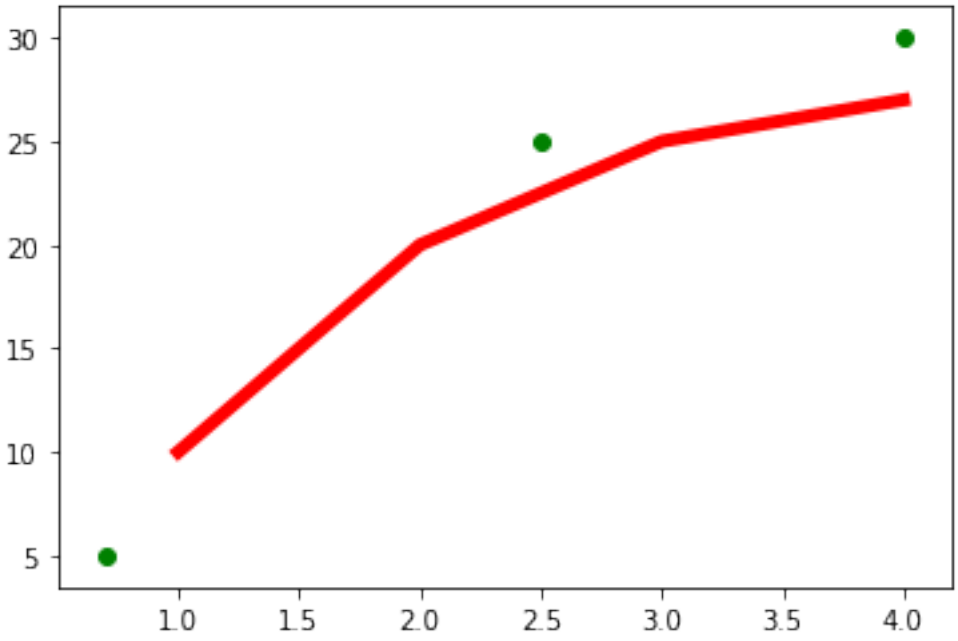
As we mentioned earlier, a figure has many components and we have access to them. Let's change color and width of the line.

[6]:
```
plt.plot(x, y, color='red', linewidth=5)
plt.show()
```



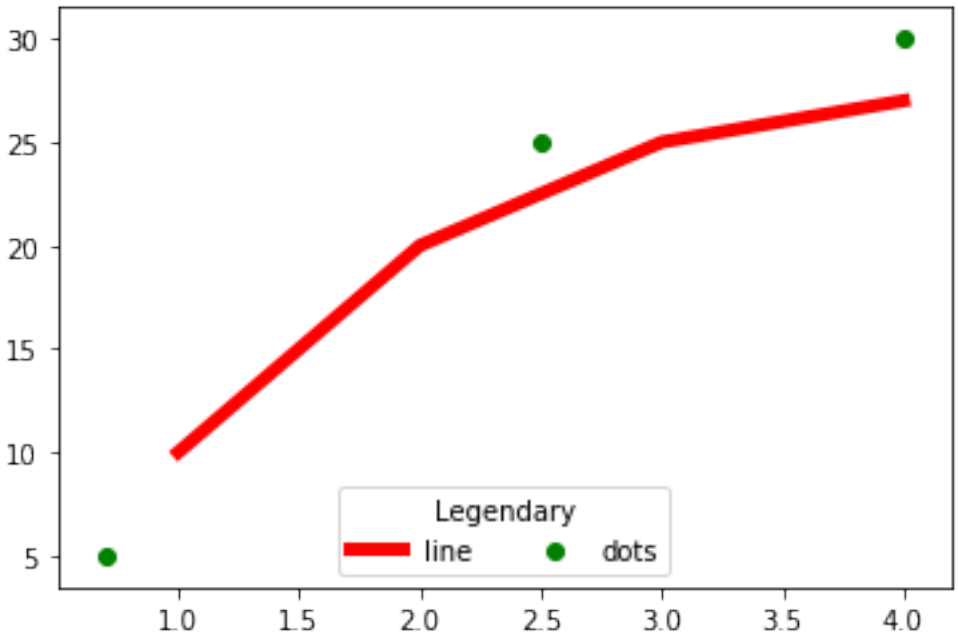
Now, let's overlay separate layers of drawings. As you can see the default type of drawing is line. Scatter plot takes (x,y) values and draws point at given coordinates.

[7]:
```
plt.plot(x, y, color='red', linewidth=5)
plt.scatter([0.7,2.5,4], [5,25,30], c='green')
plt.show()
```

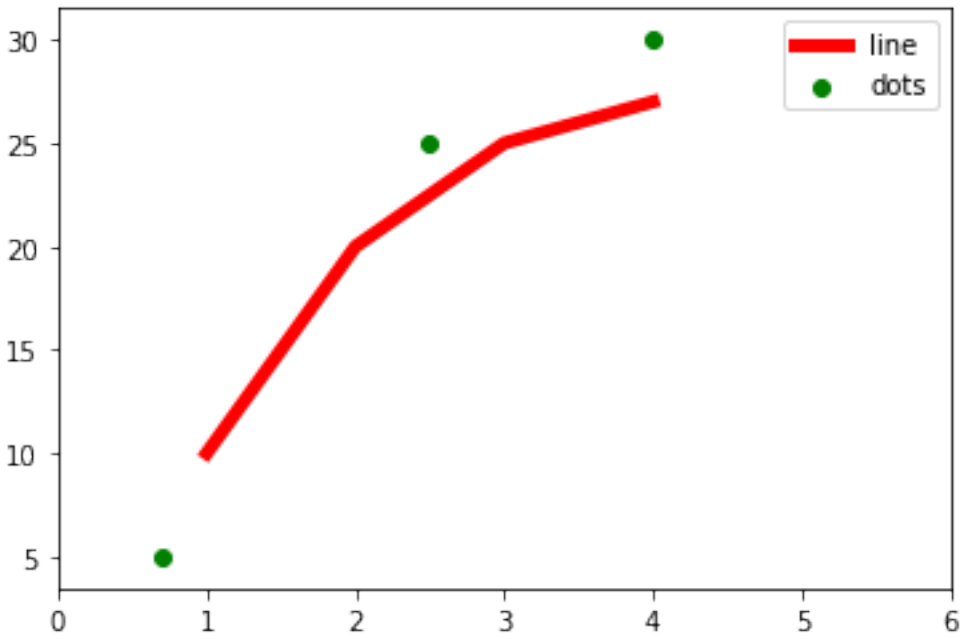


We can adjust color or type of the plot. Additionally, we can modify figure components such as `xlim`, `legend`, `title`, `xlabel` and `ylabel`.

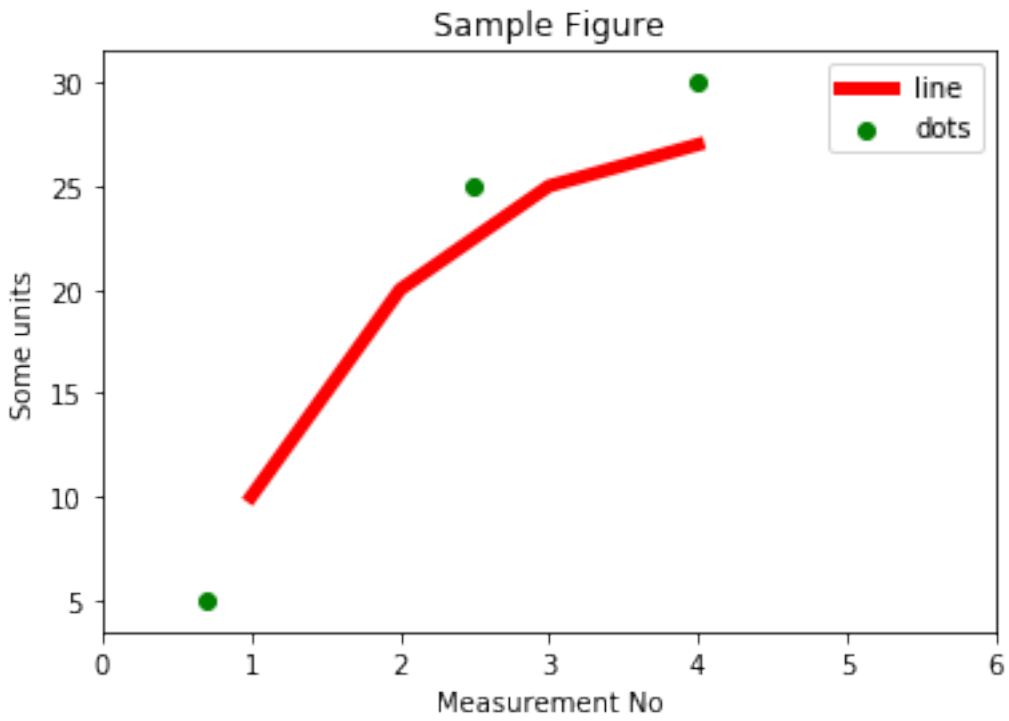
[8]:
```
plt.plot(x, y, color='red', linewidth=5, label="line")
plt.scatter([0.7,2.5,4], [5,25,30], c='green', label="dots")
plt.legend(loc="lower center", title="Legendary", ncol=2)
plt.show()
```



[9]:
```
plt.plot(x, y, color='red', linewidth=5, label="line")
plt.scatter([0.7,2.5,4], [5,25,30], c='green', label="dots")
plt.legend()
plt.xlim(0,6)
plt.show()
```

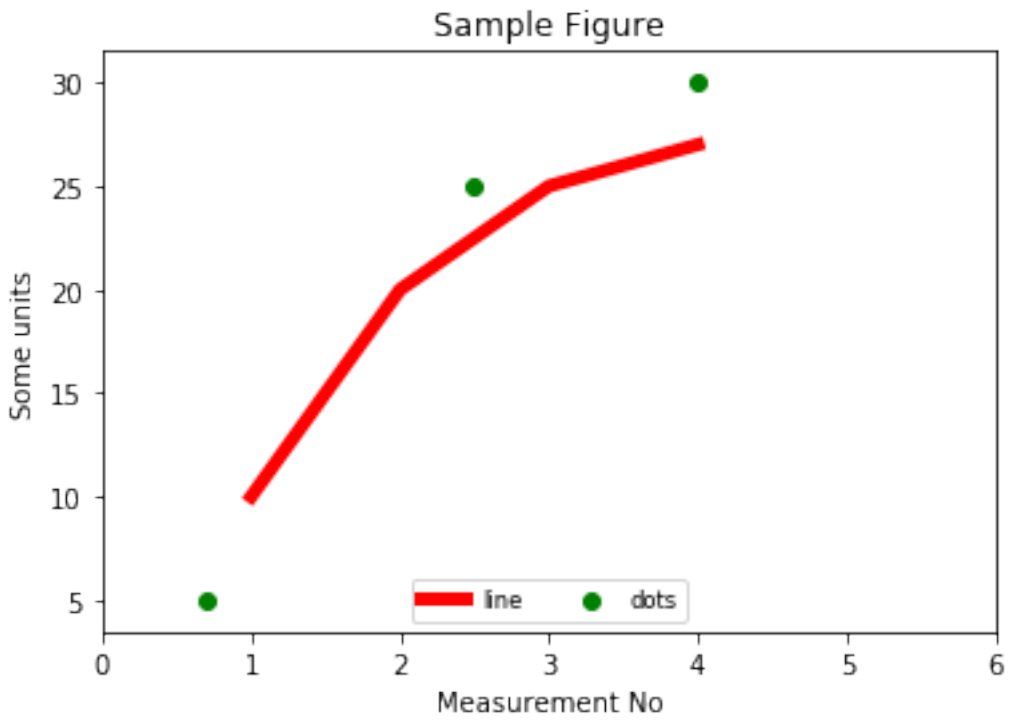


[10]:
```
plt.plot(x, y, color='red', linewidth=5, label="line")
plt.scatter([0.7,2.5,4], [5,25,30], c='green', label="dots")
plt.legend()
plt.xlim(0,6)
plt.title("Sample Figure")
plt.xlabel("Measurement No")
plt.ylabel("Some units")
plt.show()
```

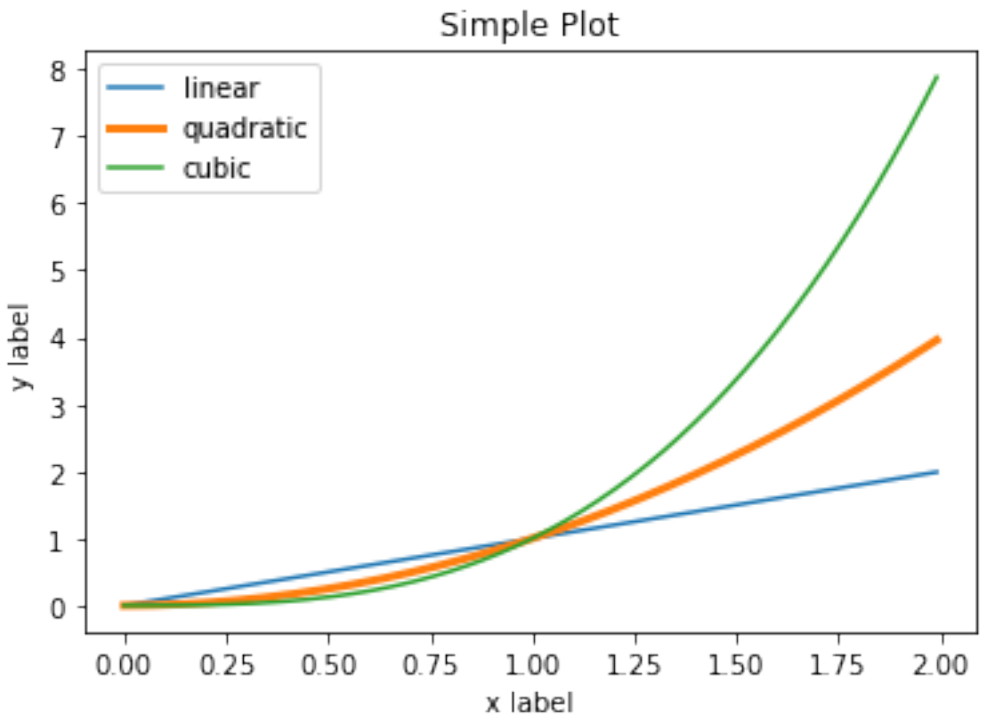


Many of the functions have numerous arguments to adjust. If you don't define any argument then defaults will be used. For example, we didn't provide any details to `legend()` function but it still worked. Let's see what we can change in `legend()` function from the manual and adjust some parameters.

[11]:
```
plt.plot(x, y, color='red', linewidth=5, label="line")
plt.scatter([0.7,2.5,4], [5,25,30], c='green', label="dots")
plt.legend(loc="lower center", ncol=2, fontsize='small')
plt.xlim(0,6)
plt.title("Sample Figure")
plt.xlabel("Measurement No")
plt.ylabel("Some units")
plt.show()
```

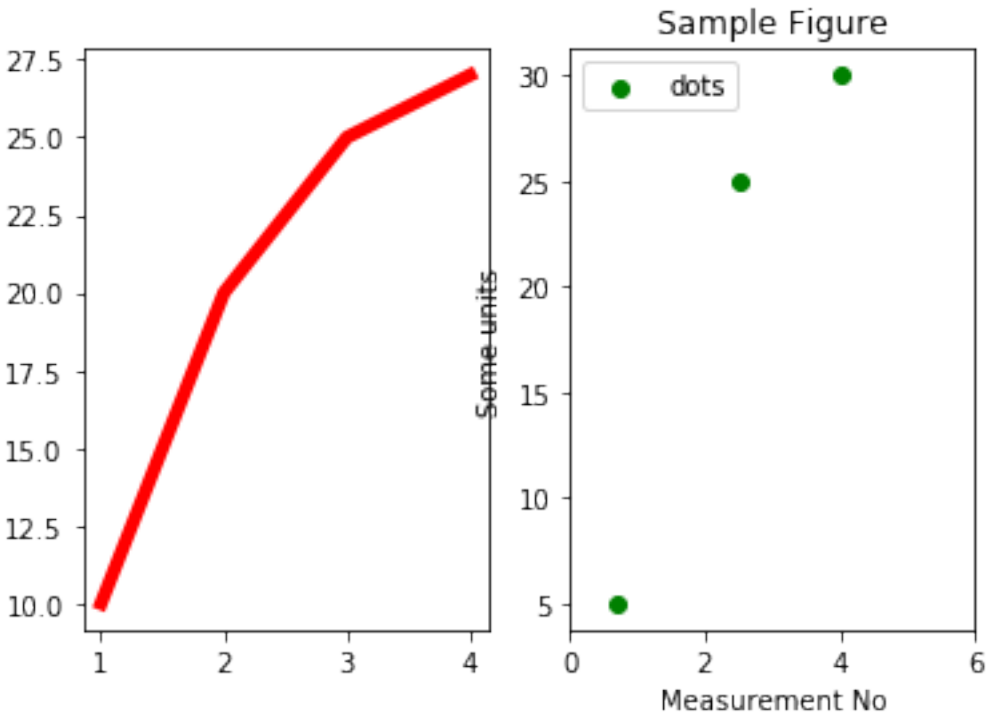


[12]:
```
# modified from https://matplotlib.org/tutorials/introductory/usage.
 ↪html#sphx-glr-tutorials-introductory-usage-py

#x_array = np.linspace(0, 2, 100)
#plt.plot(x_array, x_array, label='linear')
#plt.plot(x_array, x_array**2, label='quadratic')
#plt.plot(x_array, x_array**3, label='cubic')

x2=[x * 0.01 for x in range(0, 200)]
plt.plot(x2,x2, label='linear')
plt.plot(x2,[x**2 for x in x2], label='quadratic', lw=3)
plt.plot(x2,[x**3 for x in x2], label='cubic')

plt.xlabel('x label')
plt.ylabel('y label')
plt.title("Simple Plot")
plt.legend()
plt.show()
```

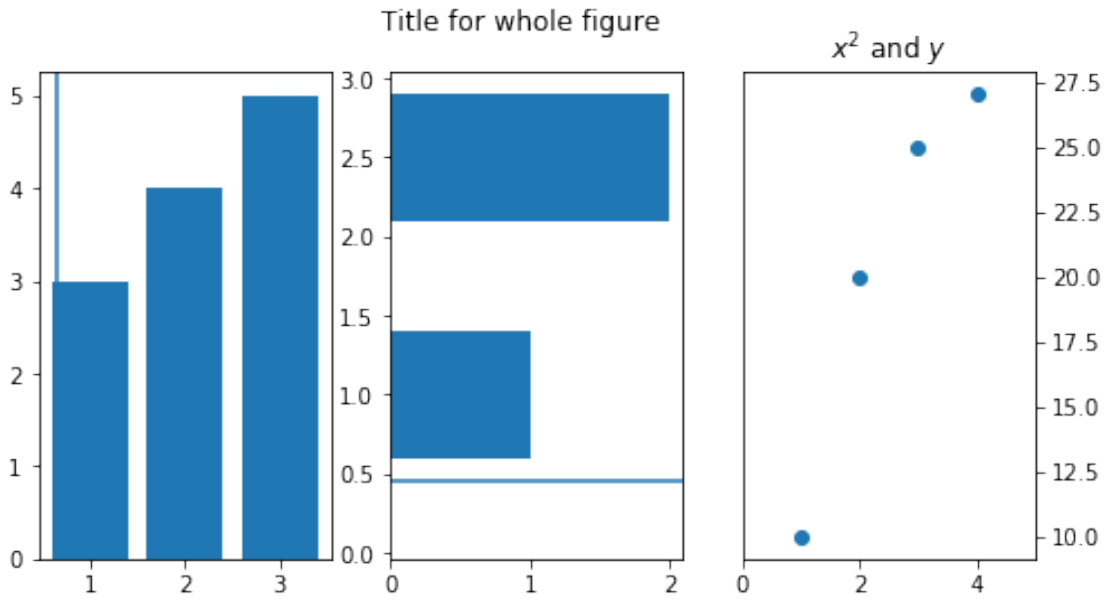


1.1 Subplots

[13]:
```
plt.subplot(121)  # 1 column 2 rows and this is first plot
plt.plot(x, y, color='red', linewidth=5, label="line")
plt.subplot(122)  # following commands are applied to second plot
plt.scatter([0.7,2.5,4], [5,25,30], c='green', label="dots")
plt.legend()
plt.xlim(0,6)
plt.title("Sample Figure")
plt.xlabel("Measurement No")
plt.ylabel("Some units")
plt.show()
```



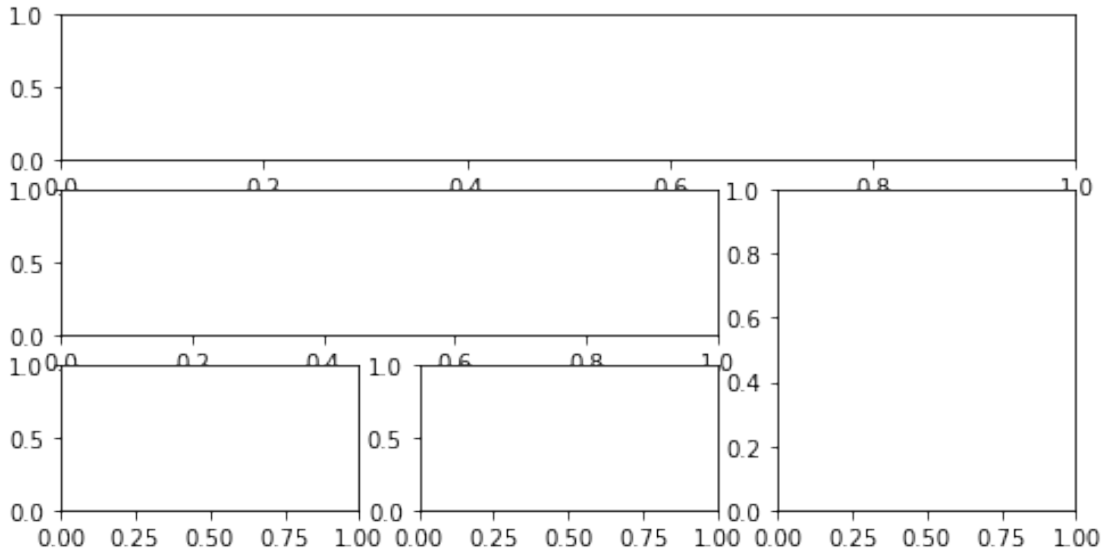
[14]:
```
import matplotlib.pyplot as plt

# Initialize the plot
fig = plt.figure(figsize=(8,4))
ax1 = fig.add_subplot(131)
ax2 = fig.add_subplot(132)
ax3 = fig.add_subplot(133)

# Plot the data
ax1.bar([1,2,3],[3,4,5])
ax2.barh([0.5,1,2.5],[0,1,2])
ax2.axhline(0.45)
ax1.axvline(0.65)
ax3.scatter(x,y)
ax3.set_xlim(0,5)
ax3.yaxis.tick_right()
ax3.set_title('$x^2$ and $y$')

fig.suptitle("Title for whole figure")
plt.show()  # or plt.savefig('images/foo.png')
```



1.2 subplot2grid

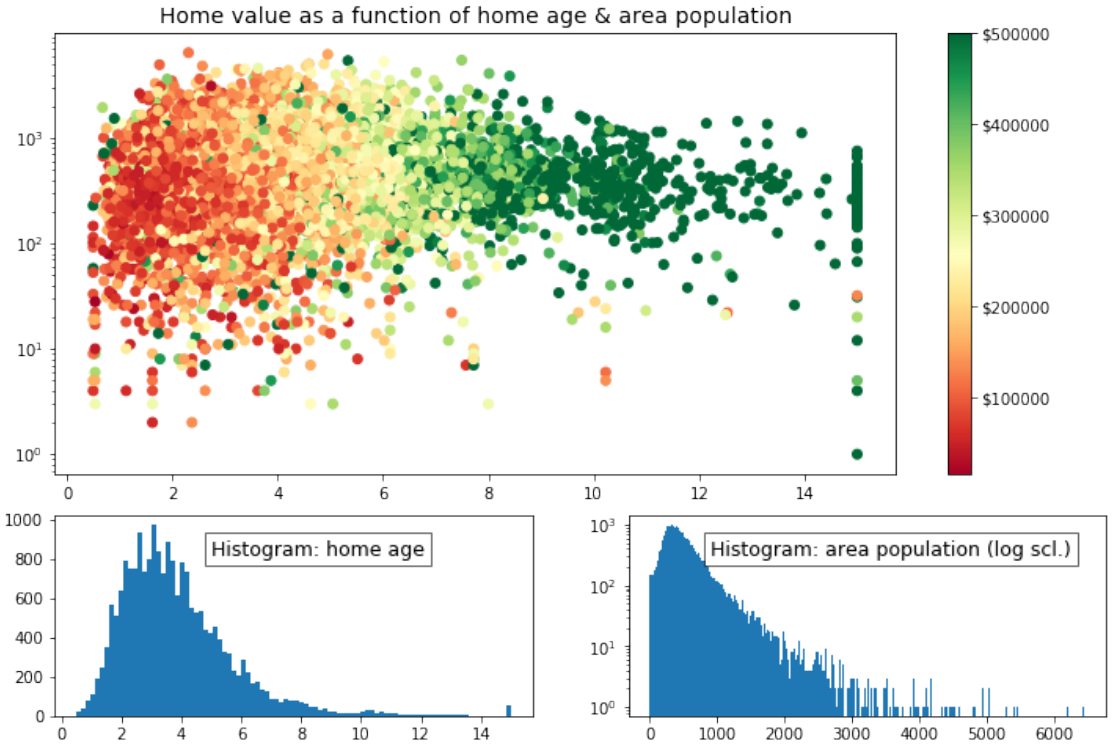
`subplot2grid()` is a helper function that is similar to `subplot()` but uses 0-based indexing and let subplot to **occupy multiple cells**.

[15]:
```
plt.figure(figsize=(8,4))
ax1 = plt.subplot2grid((3, 3), (0, 0), colspan=3)
ax2 = plt.subplot2grid((3, 3), (1, 0), colspan=2)
ax3 = plt.subplot2grid((3, 3), (1, 2), rowspan=2)
ax4 = plt.subplot2grid((3, 3), (2, 0))
ax5 = plt.subplot2grid((3, 3), (2, 1))
plt.show()
```



[16]:
```
# modified from https://realpython.com/python-matplotlib-guide/
# from io import BytesIO
# import tarfile
# from urllib.request import urlopen
# url = 'http://www.dcc.fc.up.pt/~ltorgo/Regression/cal_housing.tgz'
# b = BytesIO(urlopen(url).read())
# fpath = 'CaliforniaHousing/cal_housing.data'
# with tarfile.open(mode='r', fileobj=b) as archive:
#     housing = np.loadtxt(archive.extractfile(fpath), delimiter=',')
import numpy as np
housing = np.loadtxt("data/cal_housing.data", delimiter=',')
y = housing[:, -1]
pop, age = housing[:, [4, 7]].T

def add_titlebox(ax, text):
    ax.text(.55, .8, text,
        horizontalalignment='center',
        transform=ax.transAxes,
        bbox=dict(facecolor='white', alpha=0.6),
        fontsize=12.5)
    return ax

gridsize = (3, 2)
fig = plt.figure(figsize=(12, 8))
ax1 = plt.subplot2grid(gridsize, (0, 0), colspan=2, rowspan=2)
ax2 = plt.subplot2grid(gridsize, (2, 0))
ax3 = plt.subplot2grid(gridsize, (2, 1))
```

```
ax1.set_title('Home value as a function of home age & area population',
              fontsize=14)
sctr = ax1.scatter(x=age, y=pop, c=y, cmap='RdYlGn')
plt.colorbar(sctr, ax=ax1, format='$%d')
ax1.set_yscale('log')
ax2.hist(age, bins='auto')
ax3.hist(pop, bins='auto', log=True)

add_titlebox(ax2, 'Histogram: home age')
add_titlebox(ax3, 'Histogram: area population (log scl.)')
plt.show()
```



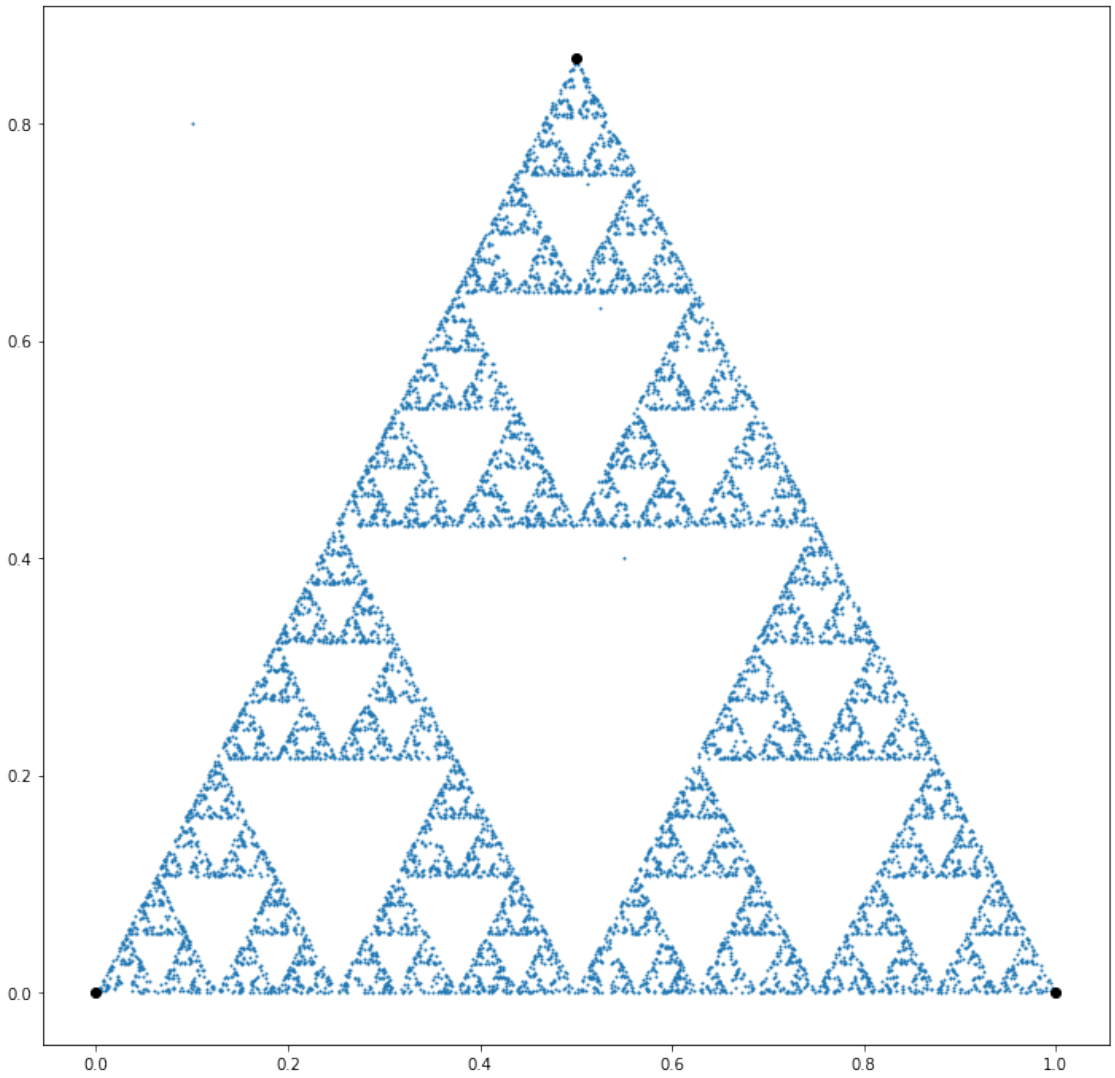
1.3 Chaos game

Definition of the game: Assume we have 3 vertices A,B and C forming a triangular area. Pick a random point within triangle and then iterate over these steps: * pick a random vertex * move the point halfway between current position and selected vertex's location

If we do this many times what type of plot do you expect? Let's find out by the help of plot function.

[17]:
```
import random
from IPython.display import clear_output
import matplotlib.pyplot as plt
```

```
def move_point(point,vertex):
    x=(point[0]+vertex[0])/2
    y=(point[1]+vertex[1])/2
    return (x,y)

# vertex={'A':(0.5,1), 'B':(0,0), 'C':(1,0)}
vertex=[(0.5,0.86),(0,0),(1,0)]
x=[]
y=[]

point=[0.1,0.8]
x.append(point[0])
y.append(point[1])

for i in range(10000):
    random_index = random.randrange(0, 3)
    point=move_point(point,vertex[random_index])
    x.append(point[0])
    y.append(point[1])

plt.figure(figsize=(12,12))
plt.scatter(x,y,s=1)
plt.scatter(*zip(*vertex), c='black')
plt.show()
```

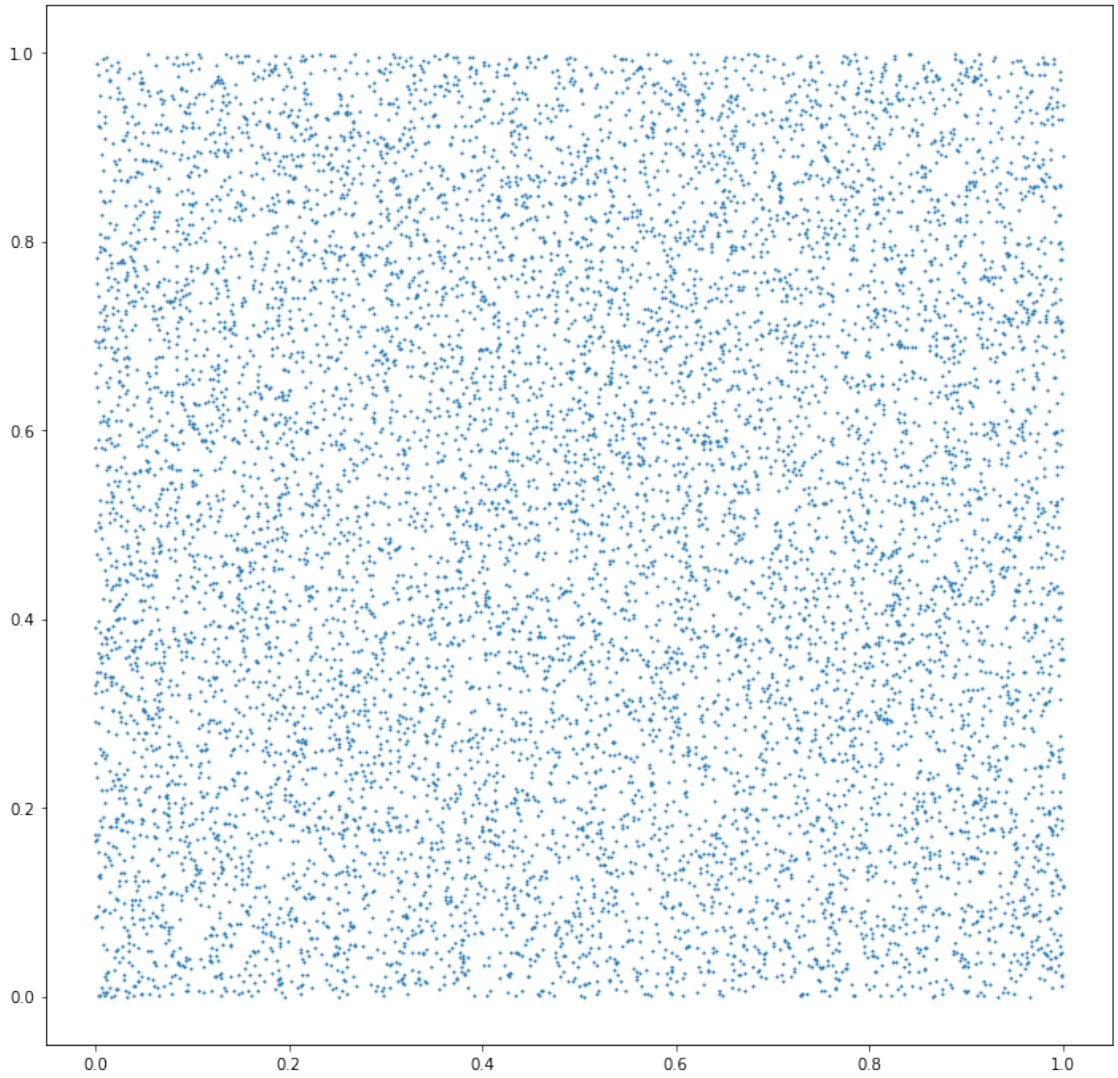


1.3.1 Chaos game - square

Let's see what type of pattern we get for 4 vertices.

[18]:
```
import random
import matplotlib.pyplot as plt

def move_point(point,vertex):
    x=(point[0]+vertex[0])/2
    y=(point[1]+vertex[1])/2
    return (x,y)

# vertex={'A':(0.5,1), 'B':(0,0), 'C':(1,0)}
vertex=[(0,0),(1,0),(0,1),(1,1)]
```

```
x=[]
y=[]

point=[0.5,0.5]
for i in range(10000):
    random_index = random.randrange(0, 4)
    point=move_point(point,vertex[random_index])
    x.append(point[0])
    y.append(point[1])

plt.figure(figsize=(12,12))
plt.scatter(x,y,s=1)
plt.show()
```

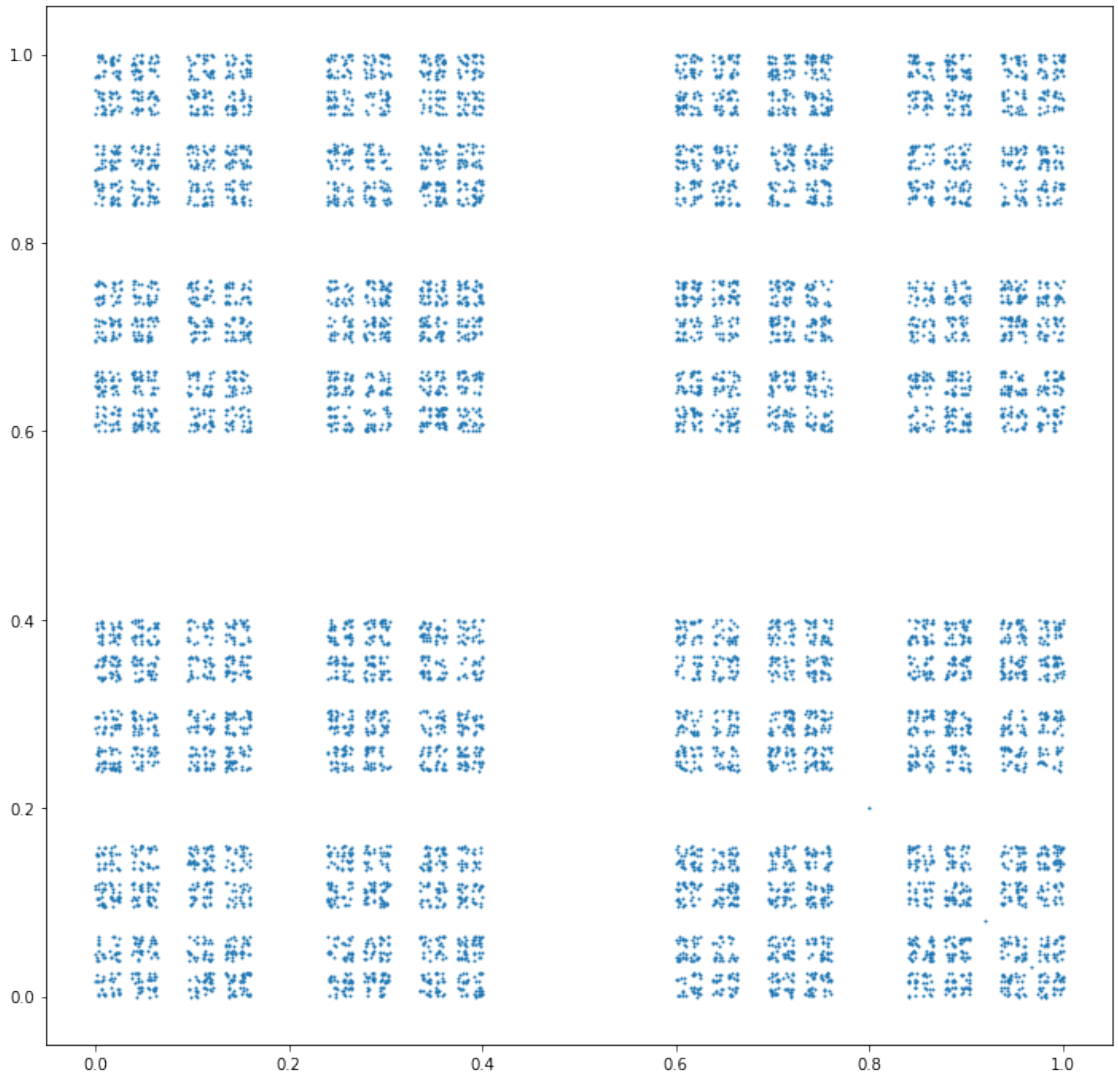


1.3.2 Generic approach

When we introduced the concept, the point was said to move halfway. But the amount of movement can be different than half. So let's rewrite the code so that the point moves a certain ratio `r`.

Then, let's see what type of pattern we get for a ratio other than half (the example below uses ratio of **0.4**)

[19]:
```
import random
import matplotlib.pyplot as plt

def move_point_gen(point,vertex,r):
    x=point[0]+(vertex[0]-point[0])*(1-r)
    y=point[1]+(vertex[1]-point[1])*(1-r)
    return (x,y)

# vertex={'A':(0.5,1), 'B':(0,0), 'C':(1,0)}
vertex=[(0,0),(1,0),(0,1),(1,1)]
x=[]
y=[]

point=[0.5,0.5]
for i in range(10000):
    random_index = random.randrange(0, 4)
    point=move_point_gen(point,vertex[random_index],0.4)
    x.append(point[0])
    y.append(point[1])

plt.figure(figsize=(12,12))
plt.scatter(x,y,s=1)
plt.show()
```



Please check if results match with Wolfram website.

1.4 plot word/char count

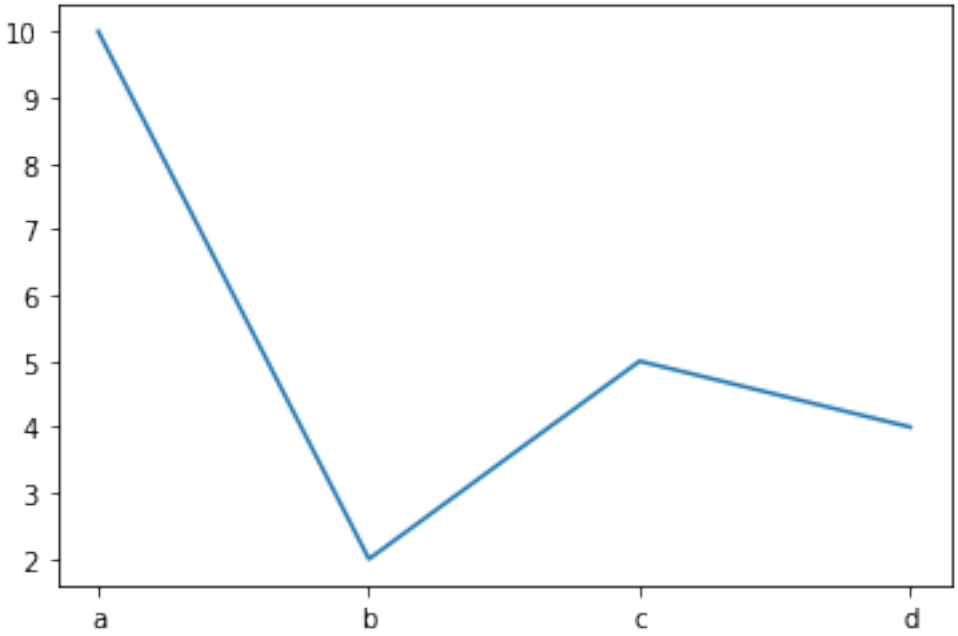
[20]:
```
dict1 = {'a': 10, 'b': 2, 'c': 5, 'd': 4}
chars= list(dict1.keys())
counts = [dict1[x] for x in chars]
```

[21]:
```
print(chars, counts)
```

```
['a', 'b', 'c', 'd'] [10, 2, 5, 4]
```

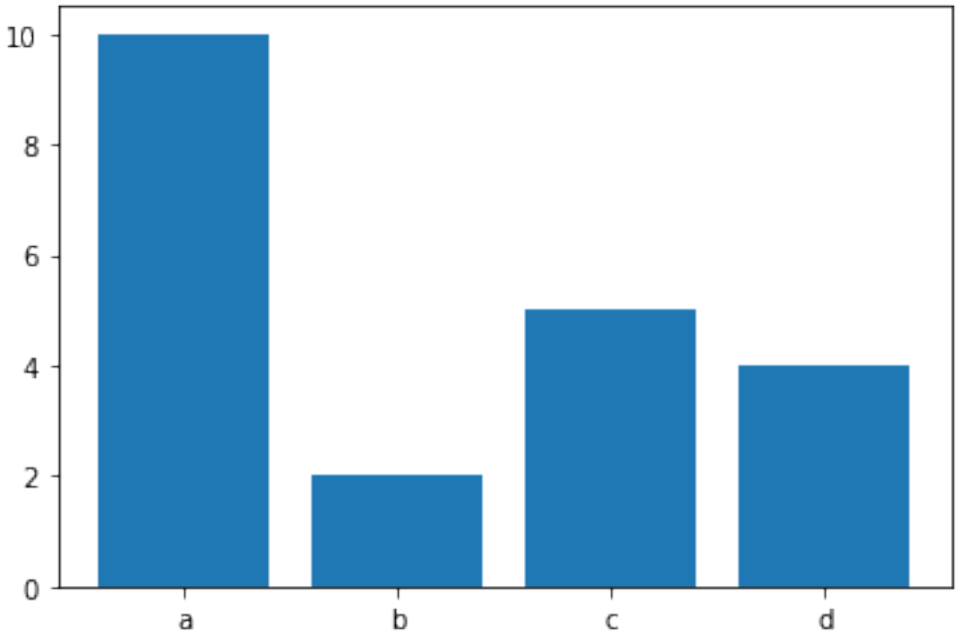
[22]:
```
import matplotlib.pyplot as plt
plt.plot(chars,counts)
```

[22]: [<matplotlib.lines.Line2D at 0x7f24b560d550>]



[23]:
```
plt.bar(chars,counts)
```

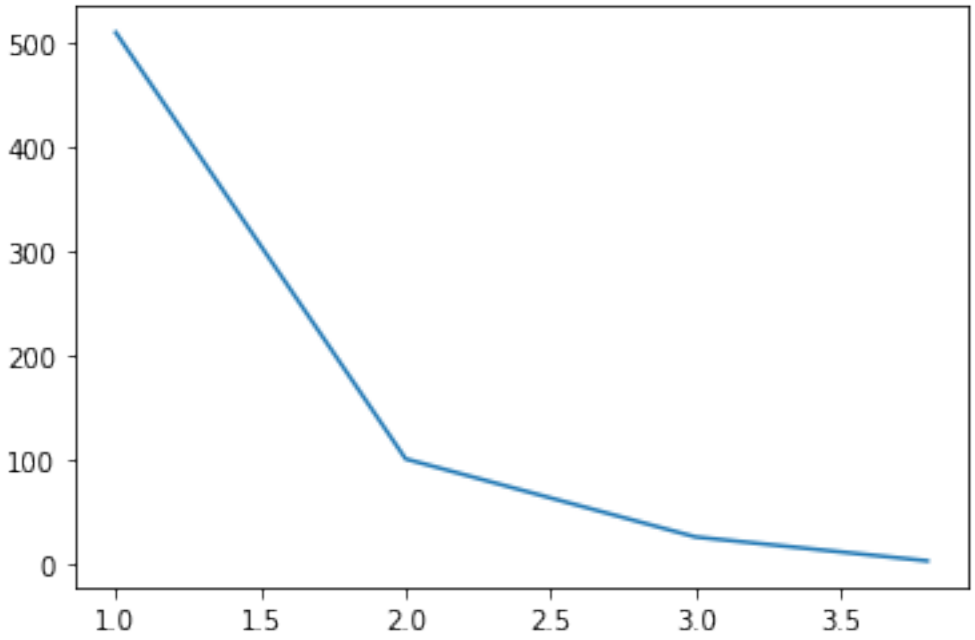
[23]: <BarContainer object of 4 artists>



1.5 Misc

There's a practical way to plot data if you have tuples of data points. The * operator is used for unpacking. Please search online for explanation and more examples. And `zip` is used to combine two (or more) list element-wise.

[24]:
```
import matplotlib.pyplot as plt
data = [(1,510), (2, 100), (3, 25), (3.8, 2)]
m, n = zip(*data)
plt.plot(m,n)
plt.show()
```



[25]:
```
data
```

[25]:
```
[(1, 510), (2, 100), (3, 25), (3.8, 2)]
```

[26]:
```
list(zip(*data))
```

[26]:
```
[(1, 2, 3, 3.8), (510, 100, 25, 2)]
```

[27]:
```
p,q = zip(*data)
```

[28]:
```
print(q)
```

```
(510, 100, 25, 2)
```

Without unpack or zip..

[29]:
```
data
```

[29]:
```
[(1, 510), (2, 100), (3, 25), (3.8, 2)]
```

[30]:
```
x_values = [n[0] for n in data]
x_values
```

[30]:
```
[1, 2, 3, 3.8]
```

[31]:
```
y_values = [ n[1] for n in data]
y_values
```

[31]:
```
[510, 100, 25, 2]
```