Python Tutorial Series 2019
Data Analysis & Visualization 1
NumPy, MatPlotLib and 2D Plots

Ph.D. Bruno C. Quint - bquint@gemini.edu

SUSD - GEMINI South, La Serena, Chile

Zen of Python

```
In [1]: import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

Previously...

Basics I

What is Python?

What will you need?

Python as a terminal

Python as a script

Types of variables

Basics II

Code Flow

Functions

Classes

Methods

Attributes

Sub-Classes

Basics III

(Re)using code

Modules

Packages

Python2 to Python3

External packages

Virtual Environments

Conda and Anaconda

Data Analysis and Visualization

DataViz 1

iPython / Jupyter

NumPy

MatPlotLib

2D Plots

DataViz 2

SciPy

Image Display

Image Analysis

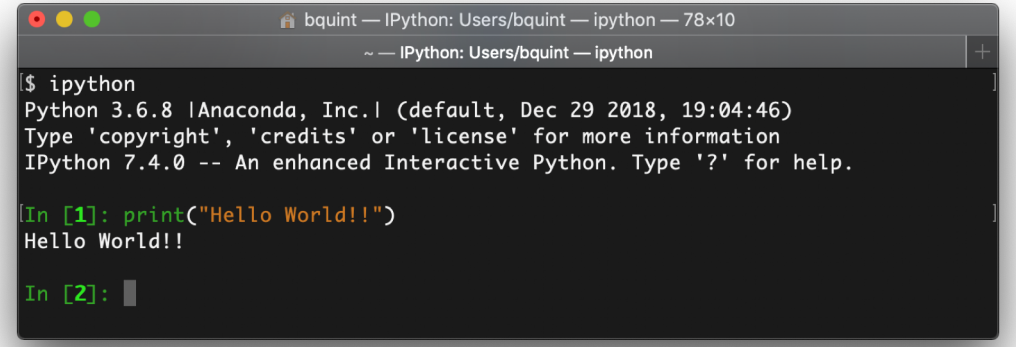
Image Manipulation

Why should I use iPython?

- Colors
- Auto complete
- The “?” and “??” commands
- Magic Functions

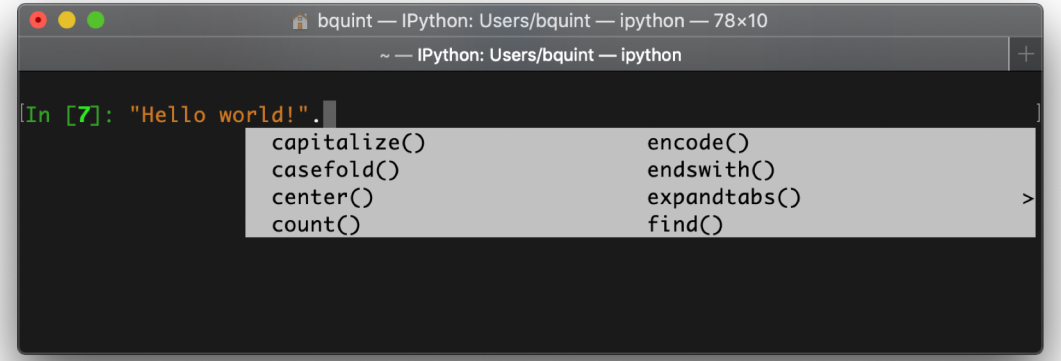
Why should I use iPython?

- Colors
- Auto complete
- The “?” and “??” commands
- Magic Functions

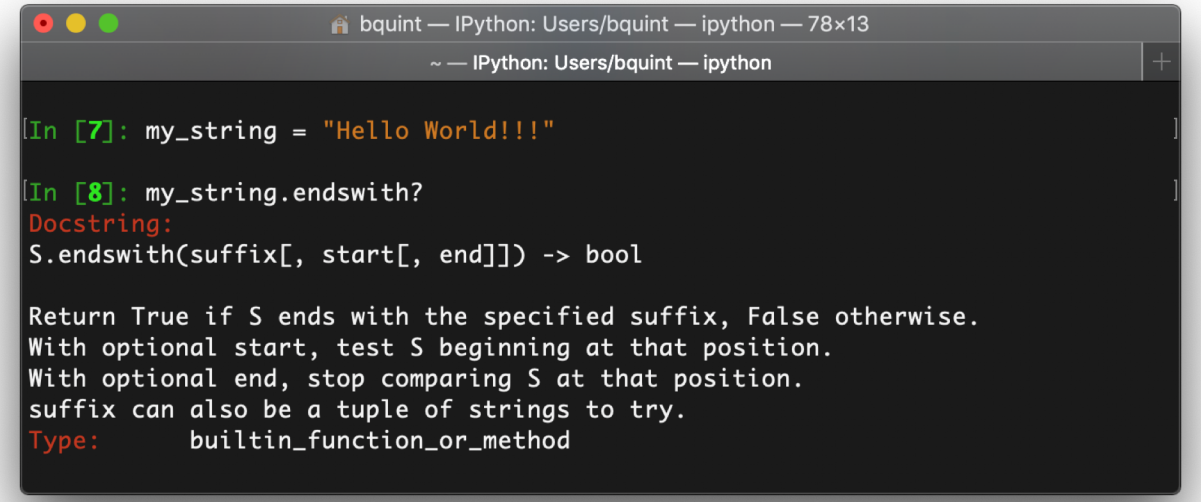


Why should I use iPython?

- Colors
- Auto complete
- The “?” and “??” commands
- Magic Functions

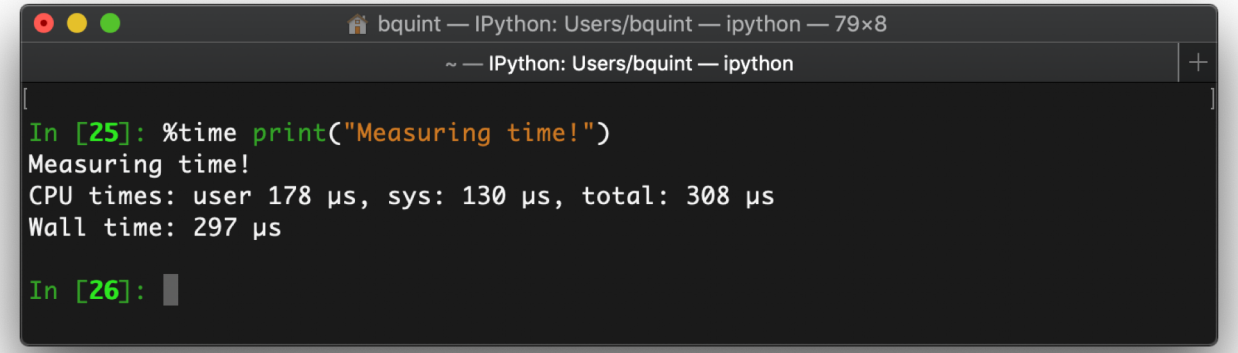


Why should I use iPython?



- Colors
- Auto complete
- The “?” and “??” commands
- Magic Functions

Why should I use iPython?



- Colors
- Auto complete
- The “?” and “??” commands
- Magic Functions

What about Jupyter?

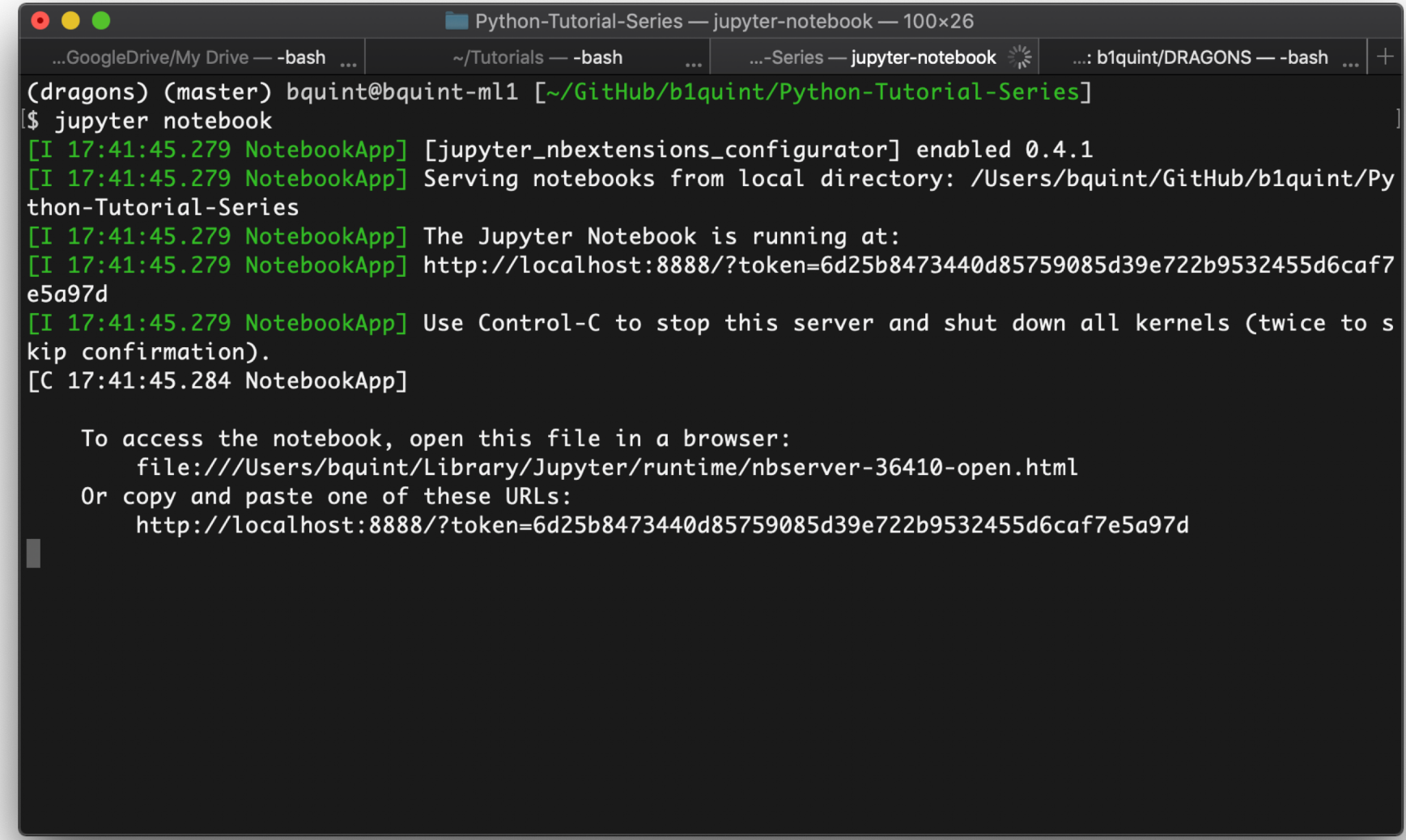
- AnaConda installations have Jupyter included
- MiniConda installations don't
- Check Virtual Environment
- Install `nbextensions`

```
$ pip install jupyter_contrib_nbextensions
```

```
$ conda install -c conda-forge jupyter_contrib_nbextensions
```

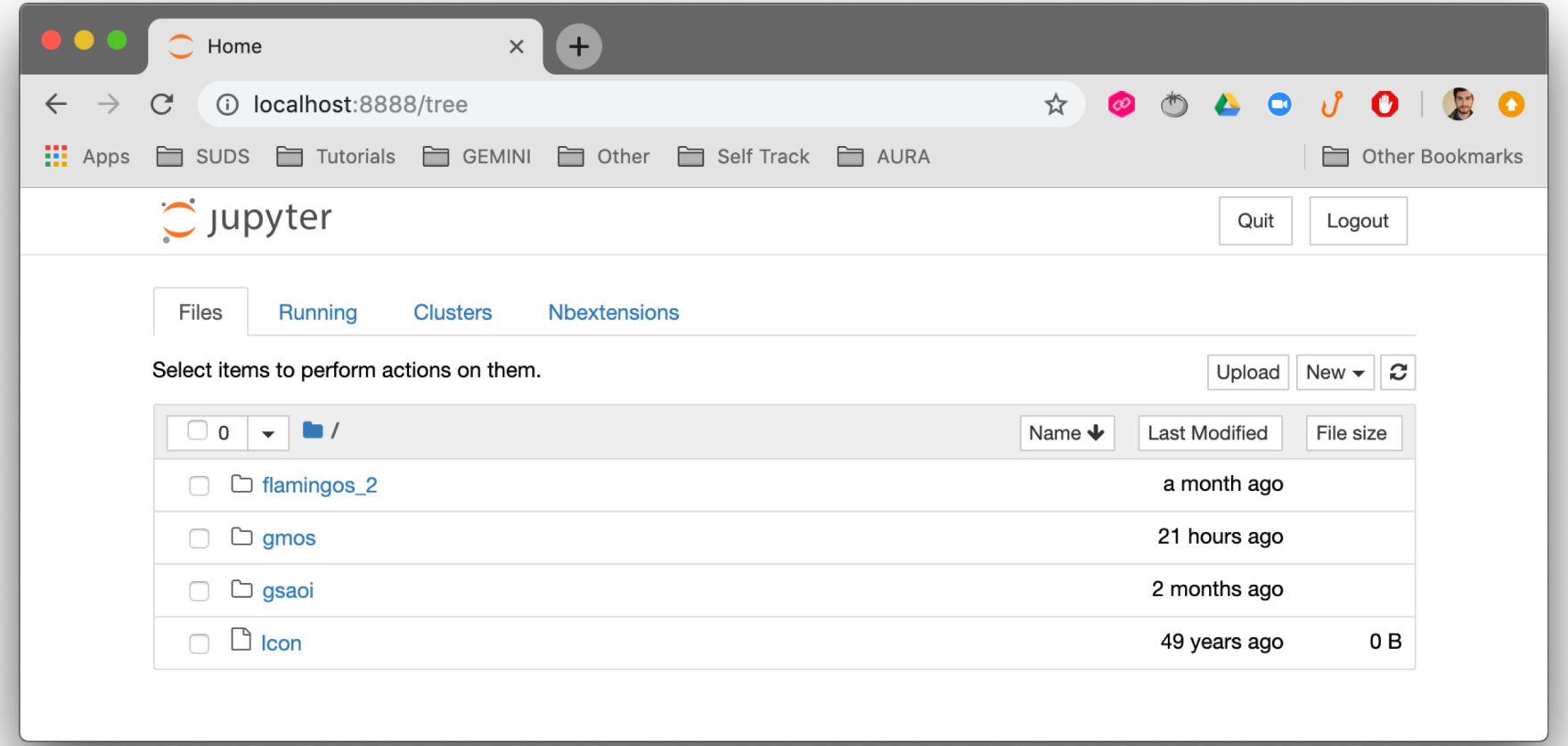
What about Jupyter?

Starting the server



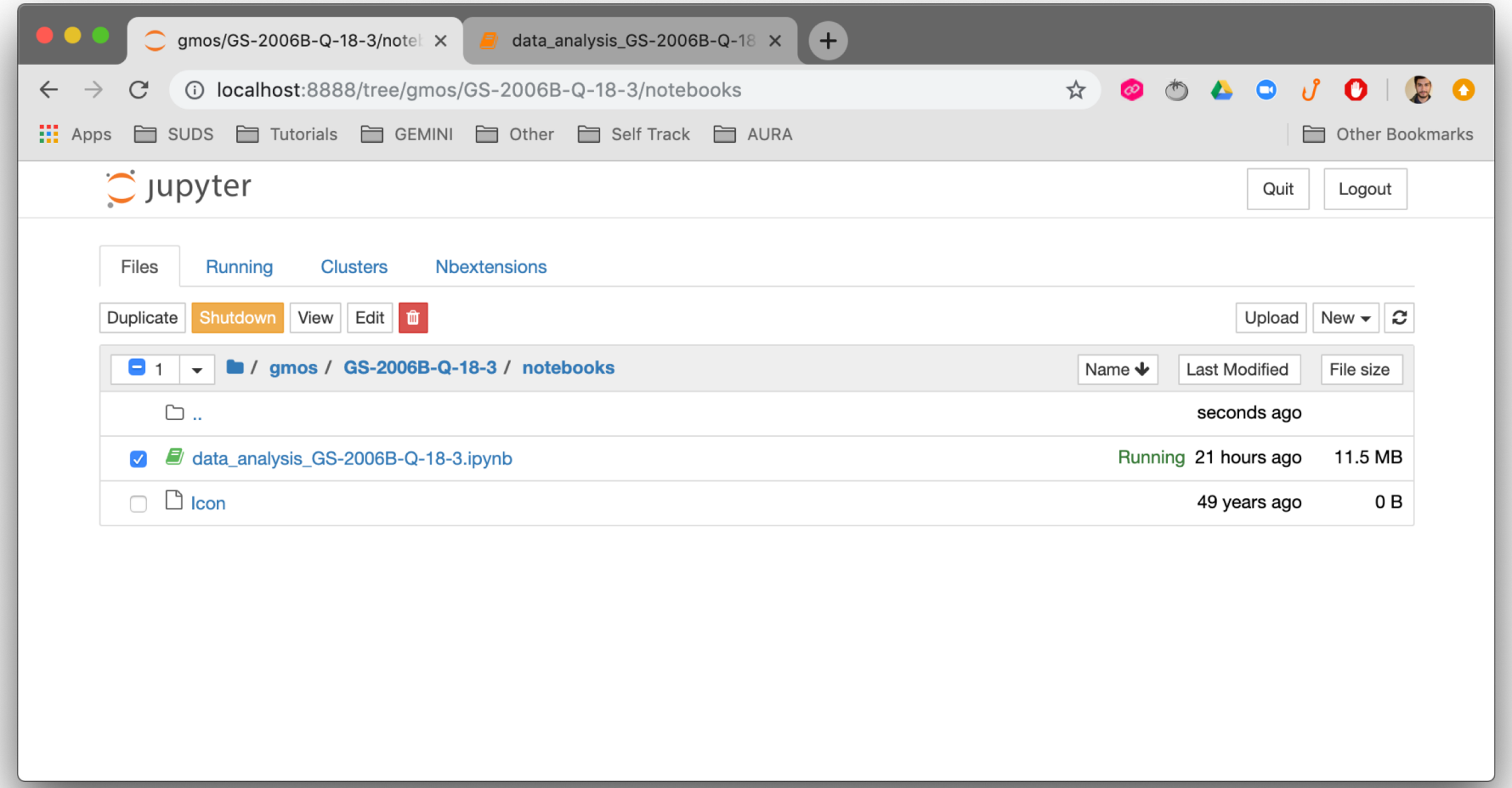
What about Jupyter?

Starting the server



What about Jupyter?

Browsing

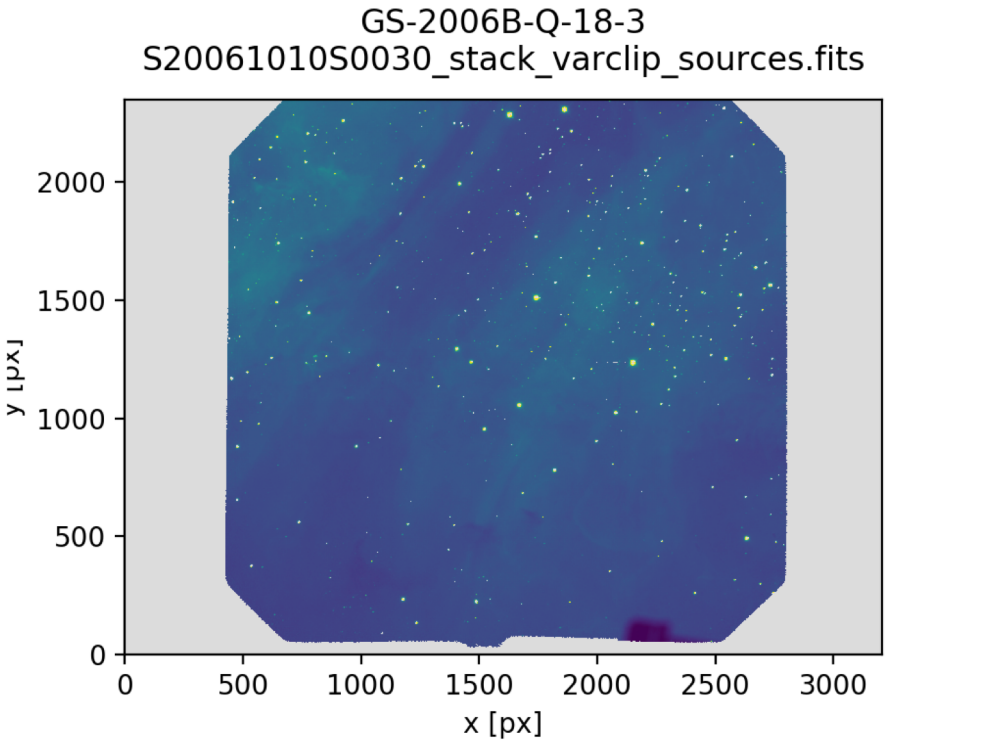


What about Jupyter?

Browsing

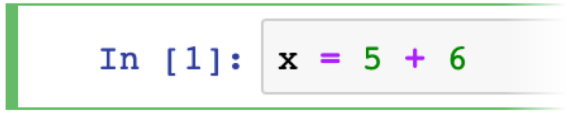
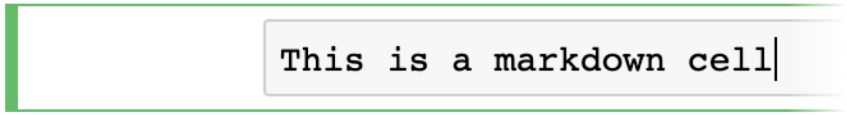
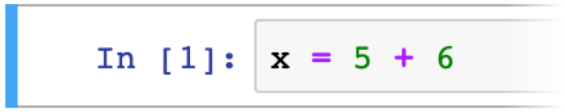
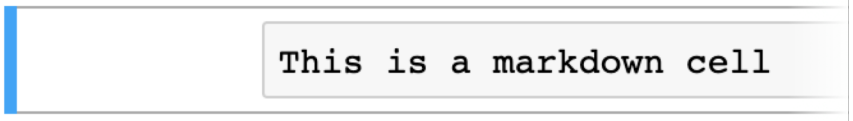
```
In [28]: display_sef('../red/S20061010S0030_stack_varclip_sources.fits')
```

../red/S20061010S0030_stack_varclip_sources.fits



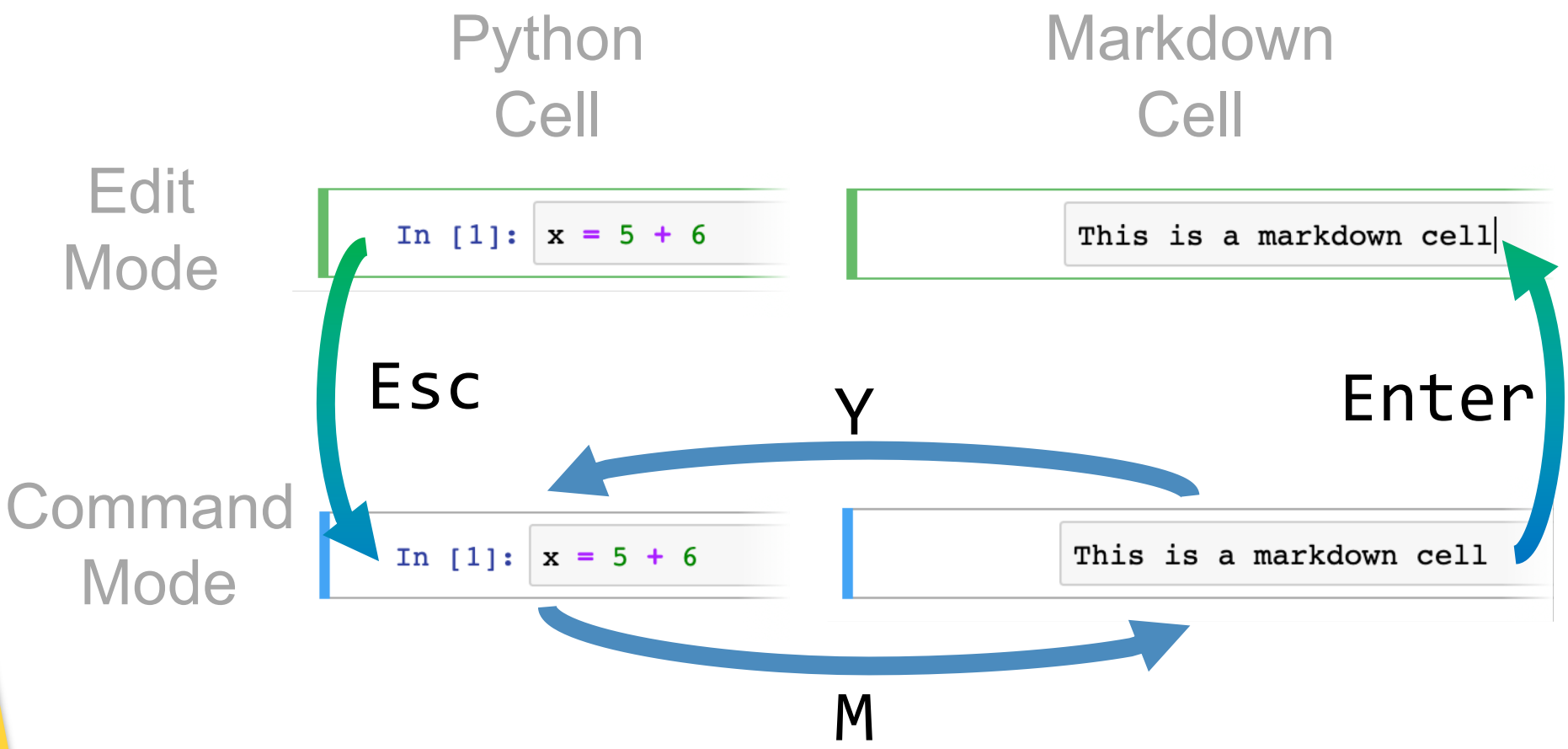
Jupyter

Different Modes and Cells

	Python Cell	Markdown Cell
Edit Mode		
Command Mode		

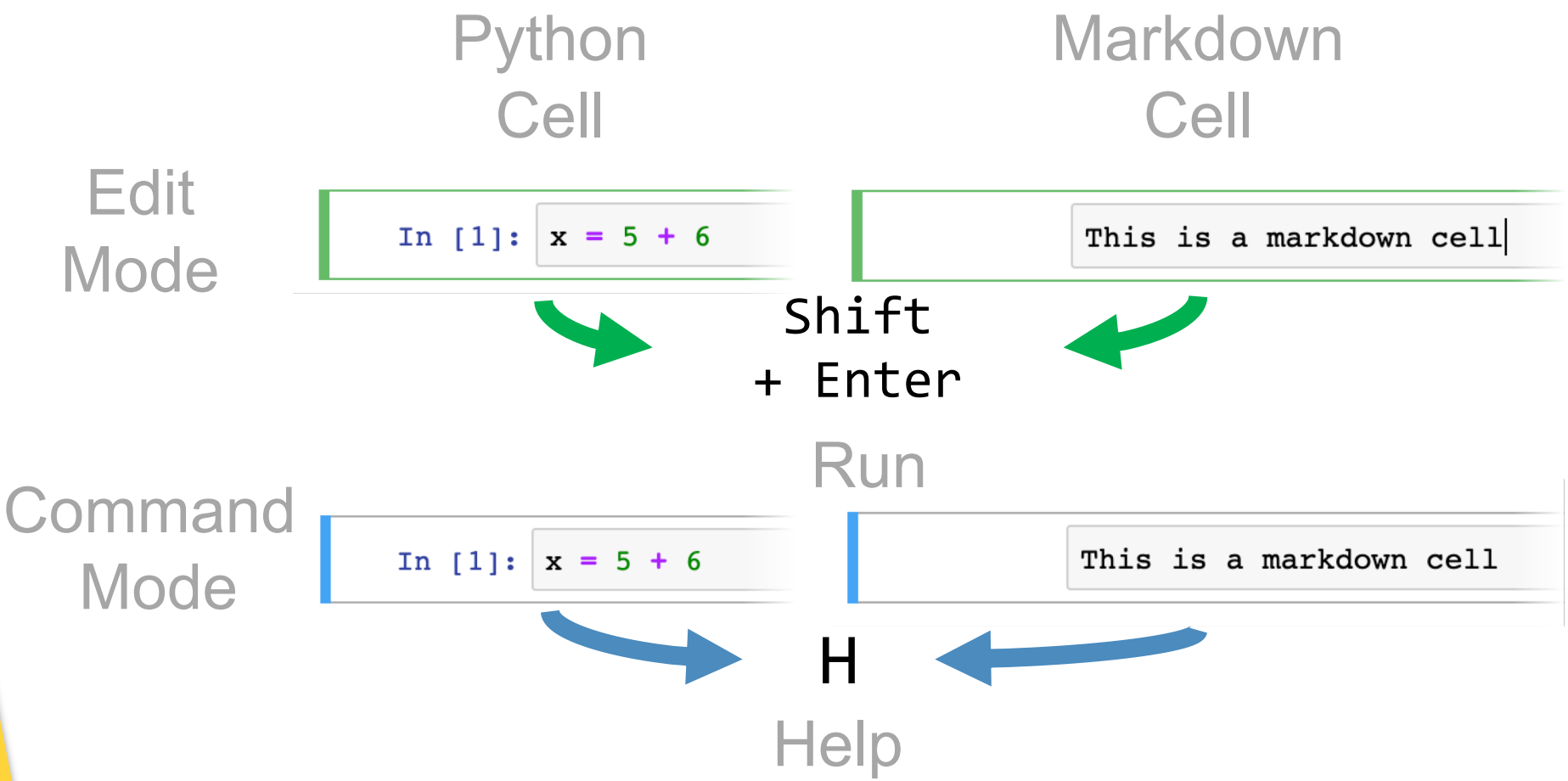
Jupyter

Different Modes and Cells



Jupyter

Different Modes and Cells

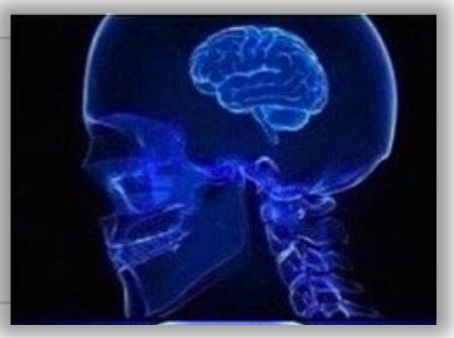


NumPy Why you should use it now!!

```
x_list = [-1, 0, 1, 2, 3, 4, 5]
print(type(x_list))
```

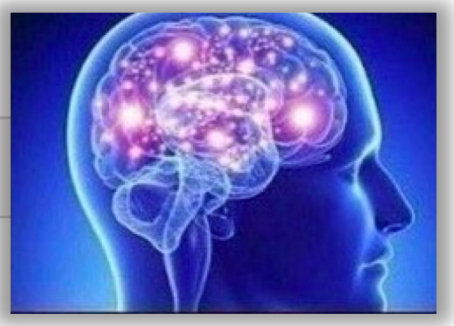
```
<class 'list'>
```

```
y_list = []
for x in x_list:
    y = x ** 2
    y_list.append(y)
```



List comprehension

```
y_list = [x ** 2 for x in x_list]
```

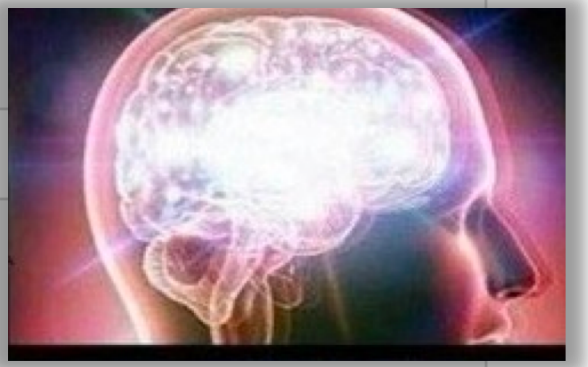


NumPy Why you should use it now!!

```
x_list = [-1, 0, 1, 2, 3, 4, 5]
print(type(x_list))
```

```
<class 'list'>
```

```
import numpy as np
x_array = np.array(x_list)
y_array = x_array ** 2
```



```
import numpy as np
x_array = np.arange(-1, 6)
y_array = x_array ** 2
```

NumPy Why you should use it now!!

Using a **for** loop

```
%%timeit
x_list = range(-1, 6)
y_list = []
for x in x_list:
    y_list.append(x ** 2)
```

```
2.03 µs ± 15.8 ns per loop (mean ± std. dev.
of 7 runs, 100000 loops each)
```

Using **list comprehension**

```
%%timeit
x_list = range(-1, 6)
y_list = [x ** 2 for x in x_list]
```

```
1.9 µs ± 36.8 ns per loop (mean ± std. dev.
of 7 runs, 1000000 loops each)
```

NumPy Why you should use it now!!

Using a **for** loop

```
%%timeit
x_list = range(-1, 6)
y_list = []
for x in x_list:
    y_list.append(x ** 2)
```

```
2.03 µs ± 15.8 ns per loop (mean ± std. dev.
of 7 runs, 100000 loops each)
```

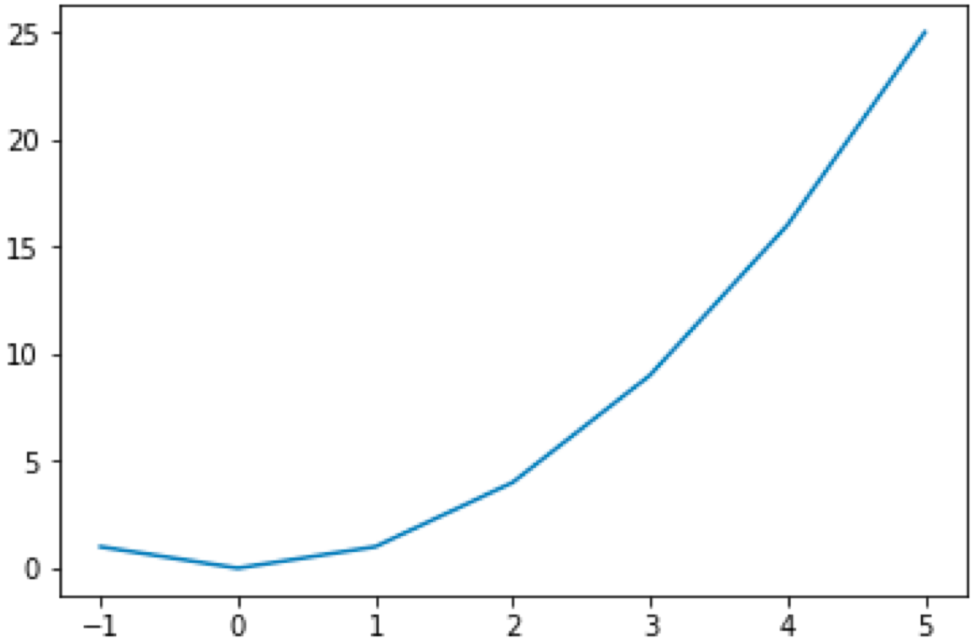
Using **NumPy**

```
%%timeit
x_array = np.arange(-1, 6)
y_array = x_array ** 2
```

```
1.23 µs ± 15.3 ns per loop (mean ± std. dev.
of 7 runs, 1000000 loops each)
```

MatPlotLib A simple plot

```
from matplotlib.pyplot import *

plot(x_array, y_array)
show()
```



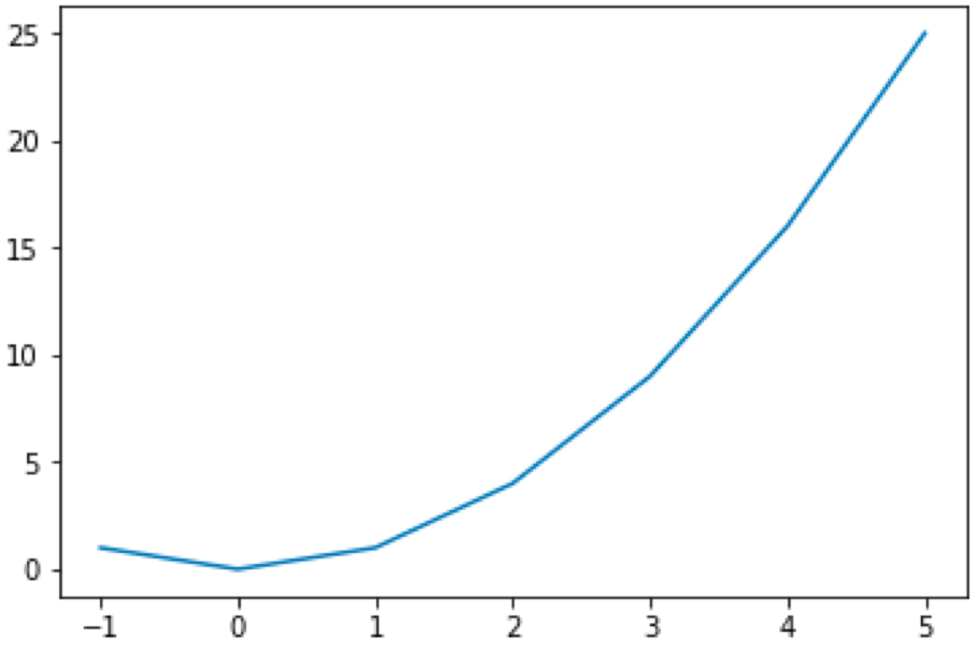
MatPlotLib A simple plot

```
from matplotlib.pyplot import *

plot(x_array, y_array)
show()
```

Avoid using

```
from ... import *
```

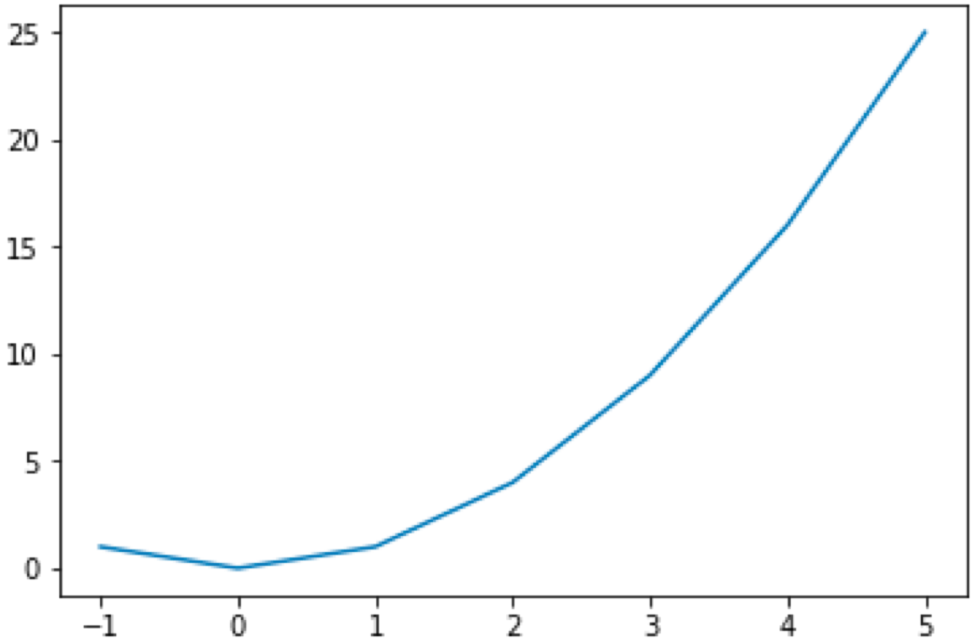
To prevent variables redefinition and to keep references



MatPlotLib A simple plot

Let's do it the "right" way

```
from matplotlib import pyplot as plt

plt.plot(x_array, y_array)
plt.show()
```

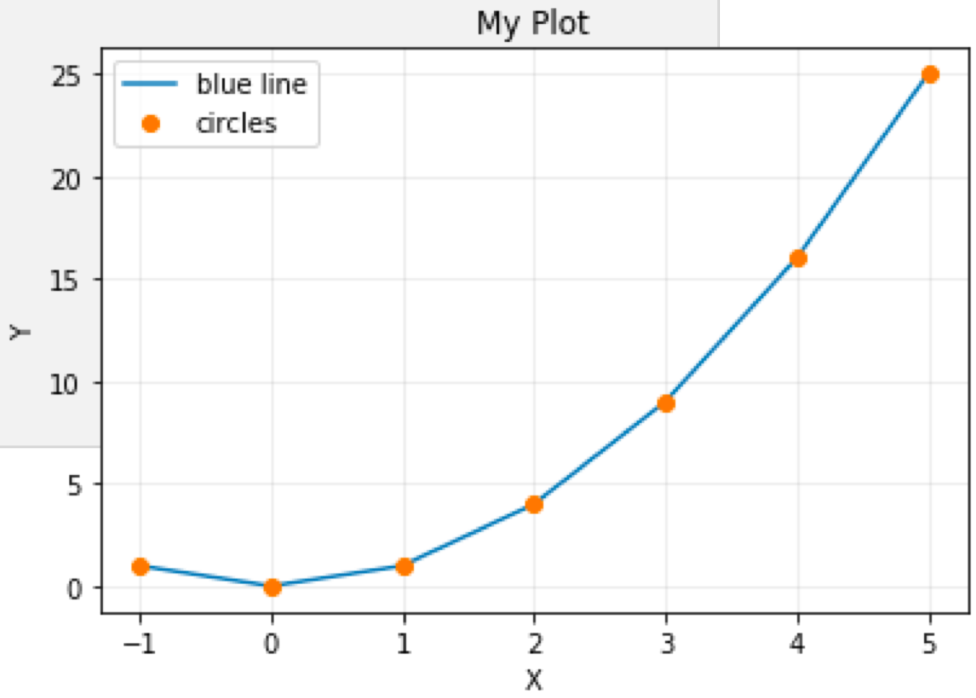


MatPlotLib Basic Customization

```
from matplotlib import pyplot as plt

plt.plot(x_array, y_array, label='blue line')
plt.plot(x_array, y_array, 'o', label='circles')

plt.grid(alpha=0.25)
plt.legend(loc='best')

plt.xlabel('X')
plt.ylabel('Y')
plt.title('My Plot’)

plt.show()
```

MatPlotLib Basic Customization

```
from matplotlib import pyplot as plt

plt.plot(x_array, y_array, label='blue line')
plt.plot(x_array, y_array, 'o', label='circles')

plt.grid(alpha=0.25)
plt.legend(loc='best')

plt.xlabel('X')
plt.ylabel('Y')
plt.title('My Plot’)

plt.show()
```

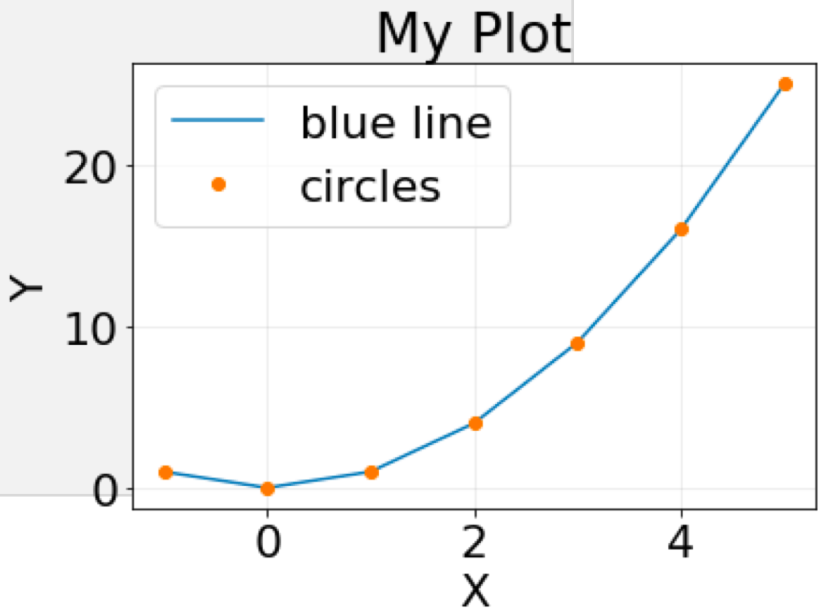


MatPlotLib Basic Customization

```
from matplotlib import pyplot as plt

plt.rcParams.update({'font.size': 22})

plt.plot(x_array, y_array, label='blue line')
plt.plot(x_array, y_array, 'o', label='circles')

plt.grid(alpha=0.25)
plt.legend(loc='best')

plt.xlabel('X')
plt.ylabel('Y')
plt.title('My Plot’)

plt.show()
```

MatPlotLib Basic Customization

```
from matplotlib import pyplot as plt

plt.rcParams.update({'font.size': 22})

plt.plot(x_array, y_array, label='blue line')
plt.plot(x_array, y_array, 'o', label='circles')

plt.grid(alpha=0.25)
plt.legend(loc='best')

plt.xlabel('X')
plt.ylabel('Y')
plt.title('My Plot')

plt.show()
```

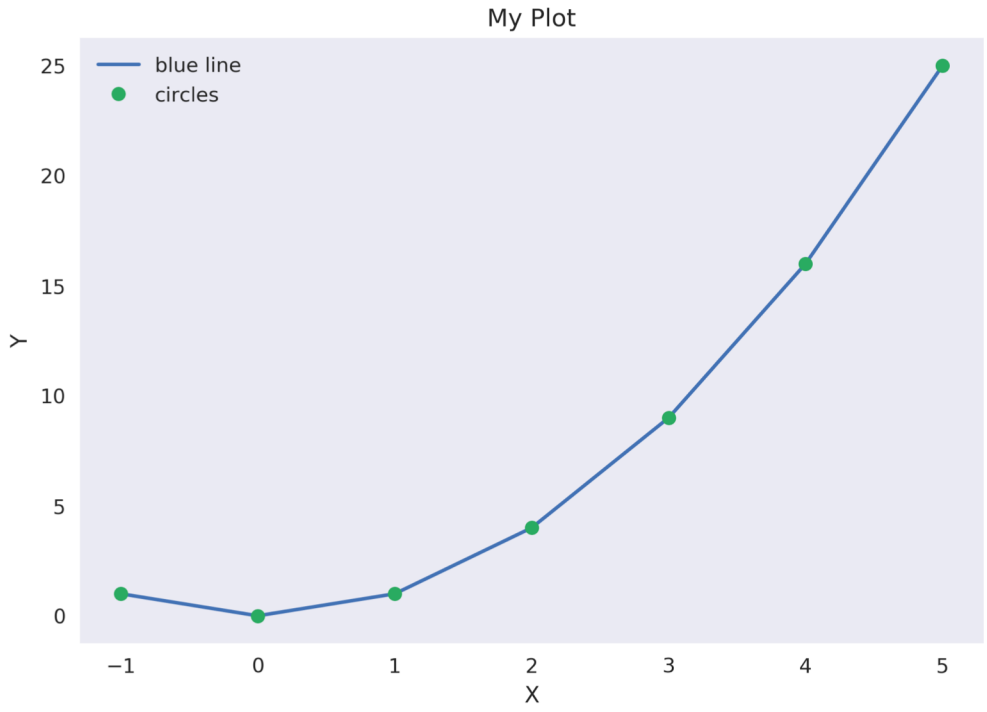


MatPlotLib Using Styles with **context manager**

```
with plt.style.context('seaborn'):

    plt.plot(x_array, y_array, label='blue line')
    plt.plot(x_array, y_array, 'o', label='circles')

    plt.grid(alpha=0.25)
    plt.legend(loc='best')

    plt.xlabel('X')
    plt.ylabel('Y')
    plt.title('My Plot’)

    plt.show()
```

MatPlotLib Using Styles with **context manager**

```
with plt.style.context('seaborn'):

    plt.plot(x_array, y_array, label='blue line')
    plt.plot(x_array, y_array, 'o', label='circles')

    plt.grid(alpha=0.25)
    plt.legend(loc='best')

    plt.xlabel('X')
    plt.ylabel('Y')
    plt.title('My Plot’)

    plt.show()
```

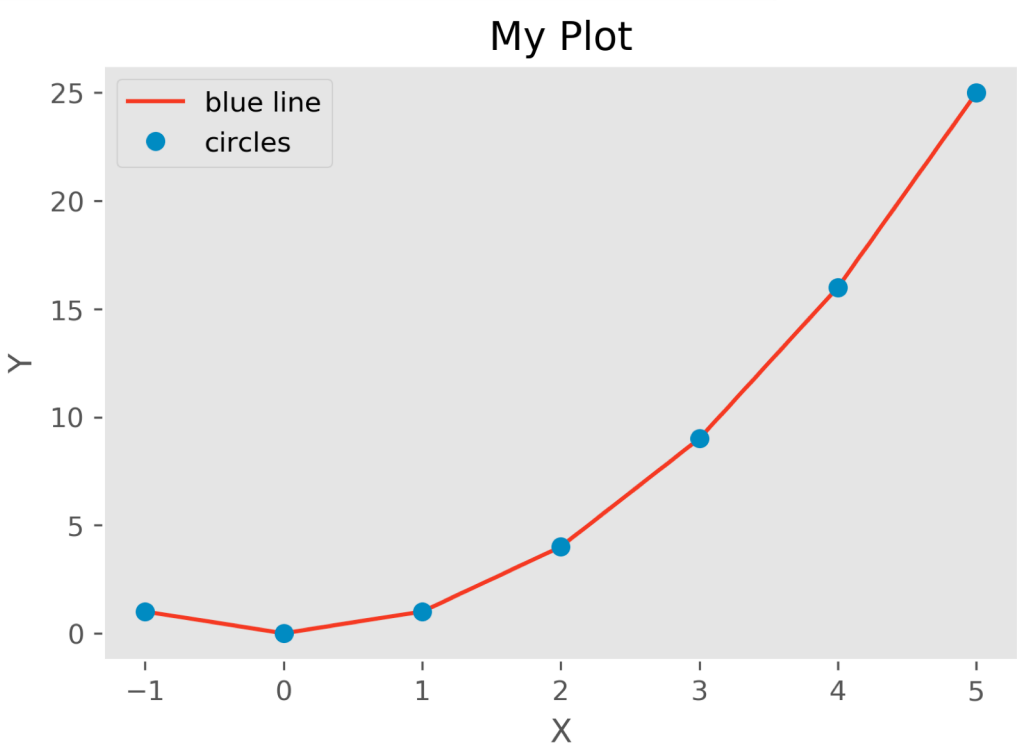


MatPlotLib Using Styles with `plt.style.use`

```
plt.style.use('ggplot'):

plt.plot(x_array, y_array, label='blue line')
plt.plot(x_array, y_array, 'o', label='circles')

plt.grid(alpha=0.25)
plt.legend(loc='best')

plt.xlabel('X')
plt.ylabel('Y')
plt.title('My Plot’)

plt.show()
```

MatPlotLib Using Styles with `plt.style.use`

```
plt.style.use('ggplot'):

plt.plot(x_array, y_array, label='blue line')
plt.plot(x_array, y_array, 'o', label='circles')

plt.grid(alpha=0.25)
plt.legend(loc='best')

plt.xlabel('X')
plt.ylabel('Y')
plt.title('My Plot’)

plt.show()
```

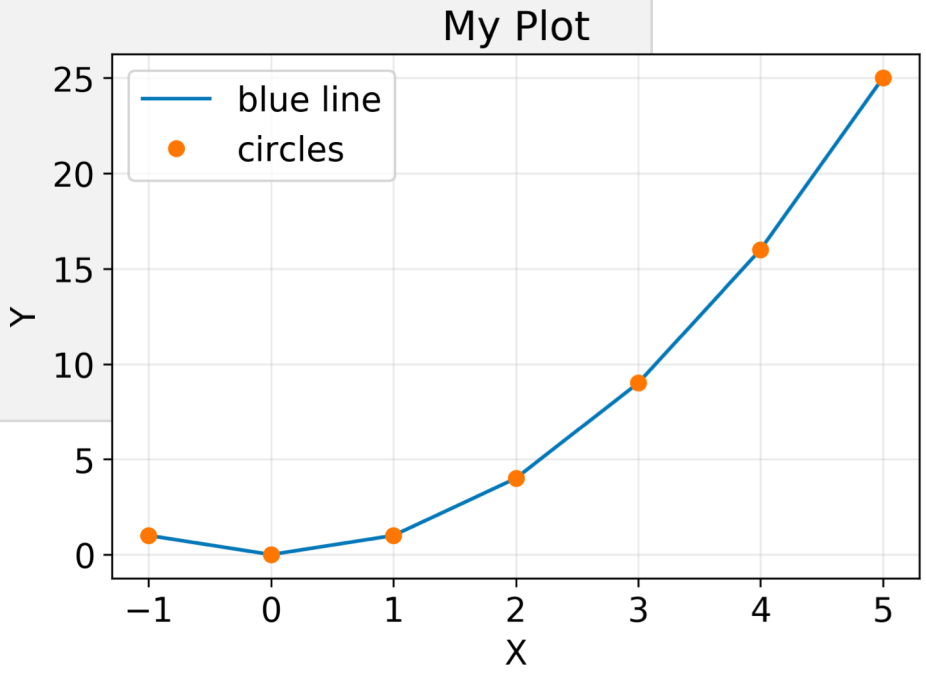


MatPlotLib Working with Figures and Axes

```
my_figure, my_axes = plt.subplots()

my_axes.plot(x_array, y_array, label='blue line')
my_axes.plot(x_array, y_array, 'o', label='circles')

my_axes.grid(alpha=0.25)
my_axes.legend(loc='best')

my_axes.set_xlabel('X')
my_axes.set_ylabel('Y')
my_axes.set_title('My Plot')

plt.show()
```

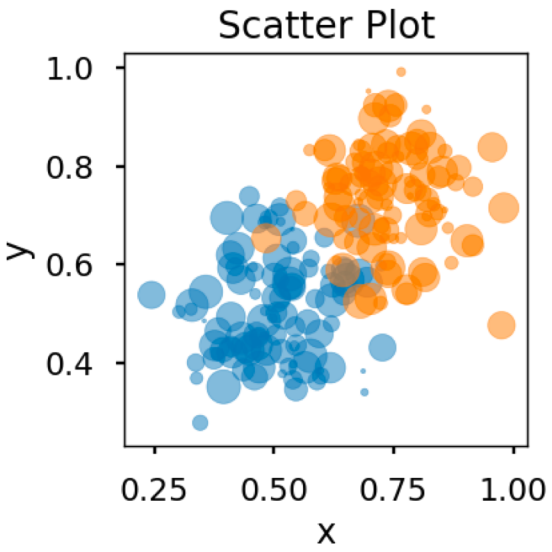
MatPlotLib Working with Figures and Axes

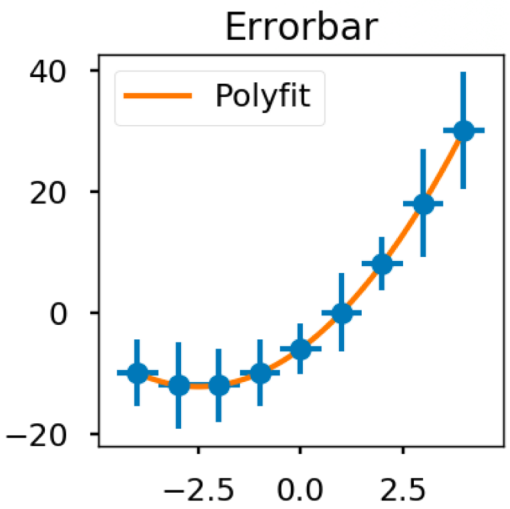
```
my_figure, my_axes = plt.subplots()

my_axes.plot(x_array, y_array, label='blue line')
my_axes.plot(x_array, y_array, 'o', label='circles')

my_axes.grid(alpha=0.25)
my_axes.legend(loc='best')

my_axes.set_xlabel('X')
my_axes.set_ylabel('Y')
my_axes.set_title('My Plot')

plt.show()
```

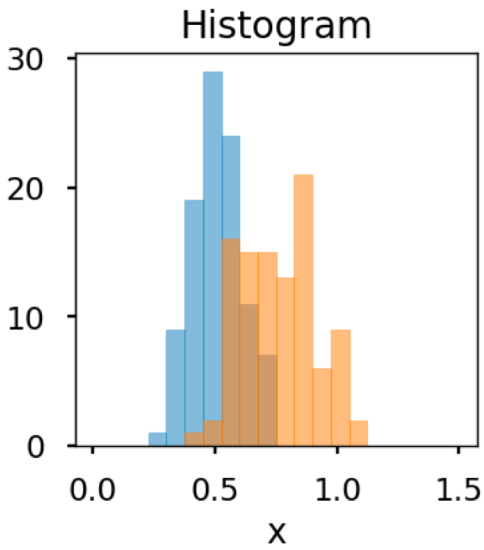


MatPlotLib Plot types

https://matplotlib.org/gallery

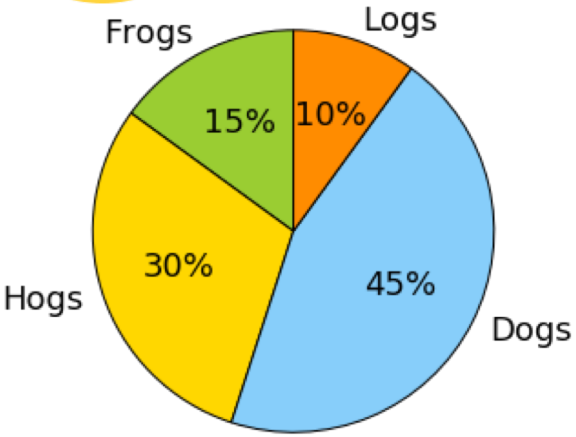




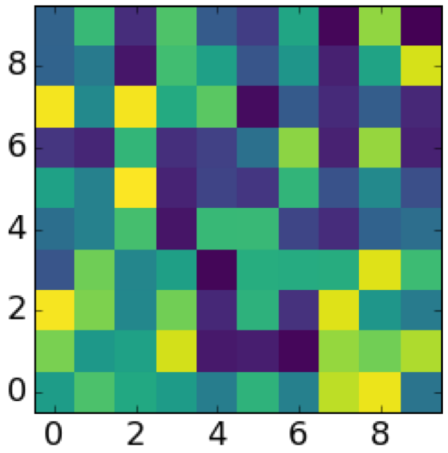


MatPlotLib Plot types

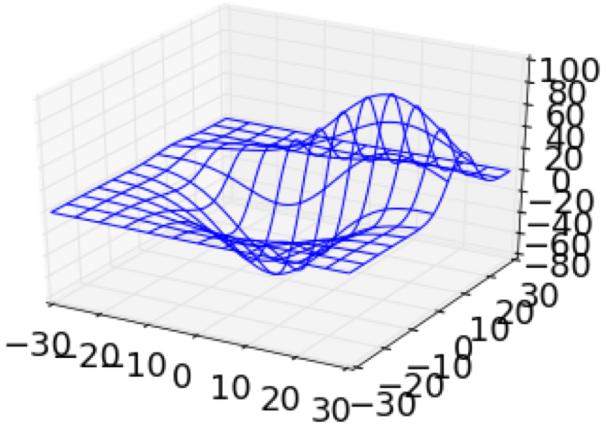
https://matplotlib.org/gallery



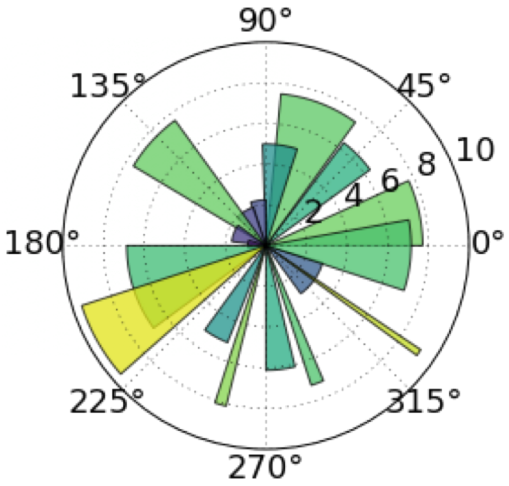
Pie Plots



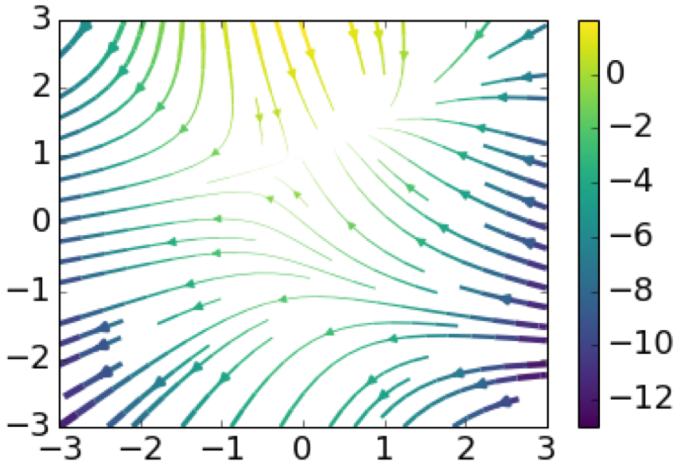
Display Images



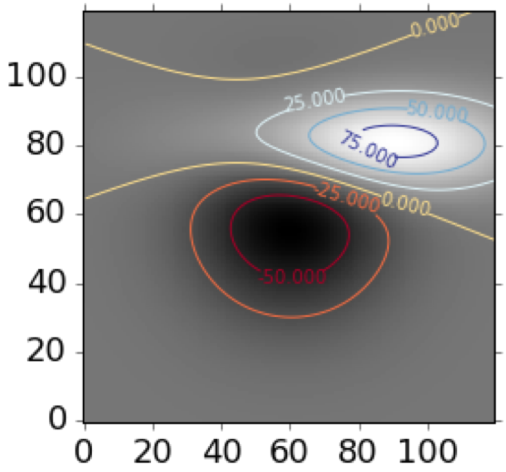
Plots 3D



Polar Plots



Stream Plots



Contour Plots

Wrapping Up a Real Case

- Read text file using NumPy
- Using `plt.plot(...)`
- Customize
- Fit results using `np.polyfit`
- Multiple plots
- Using `plt.scatter(...)`

Wrapping Up a Real Case

data/my_table.txt

0	X_IMAGE	6	FLUX_1
1	Y_IMAGE	7	FLUX_2
2	X_WORLD	8	FLUX_3
3	Y_WORLD	9	FLUX_4
4	**FLUX_STACK**	**10**	**MEDIAN**
5	FLUX_0	**11**	**SNR**

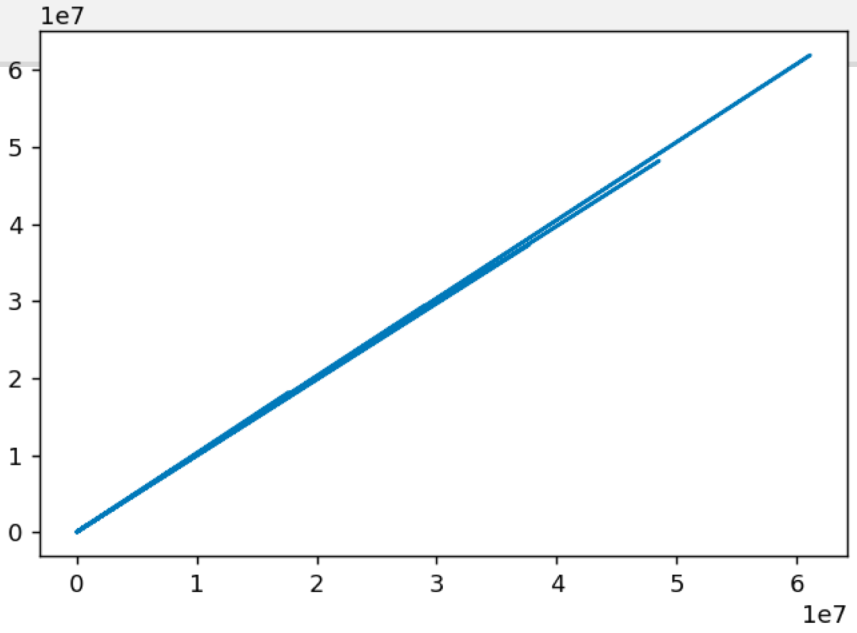
Read text file using NumPy

```
filename = 'data/my_table.txt’
flux_stack, flux_median, snr = np.loadtxt(
    filename, usecols=[4, 10, 11], unpack=True)
```

Wrapping Up a Real Case

A simple plot

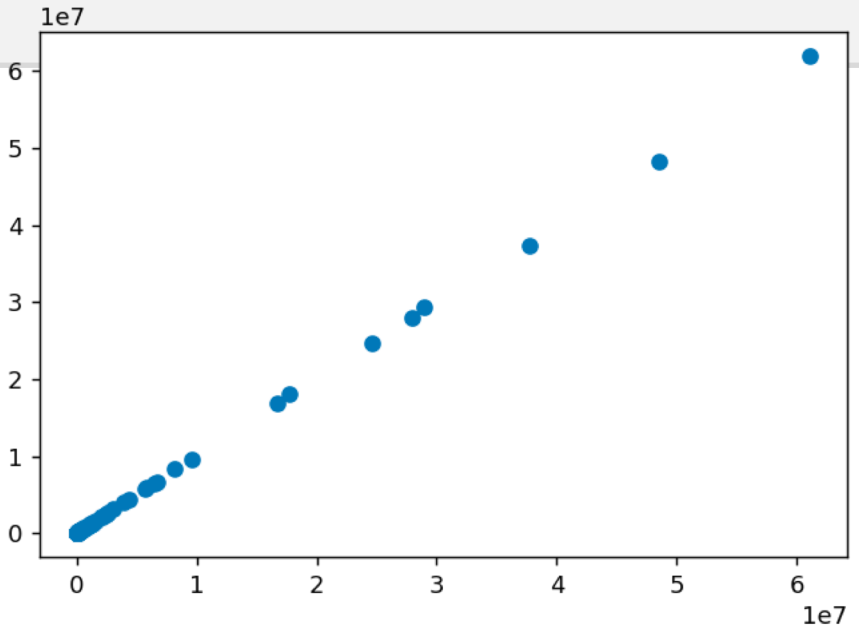
```
fig, ax = plt.subplots()

ax.plot(flux_median, flux_stack)

plt.show()
```



Wrapping Up a Real Case

Changing Marker and LineStyle

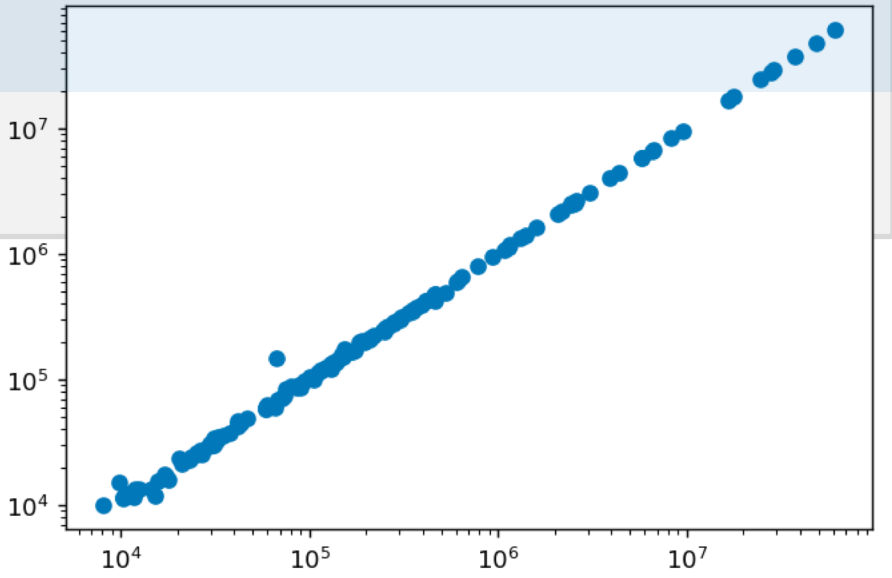
```
fig, ax = plt.subplots()

ax.plot(flux_median, flux_stack, 'o')

plt.show()
```



Wrapping Up a Real Case

Changing Marker and LineStyle

```
fig, ax = plt.subplots()

ax.plot(flux_median, flux_stack, 'o’)

ax.set_xscale('log')
ax.set_yscale('log')

plt.show()
```



Wrapping Up a Real Case

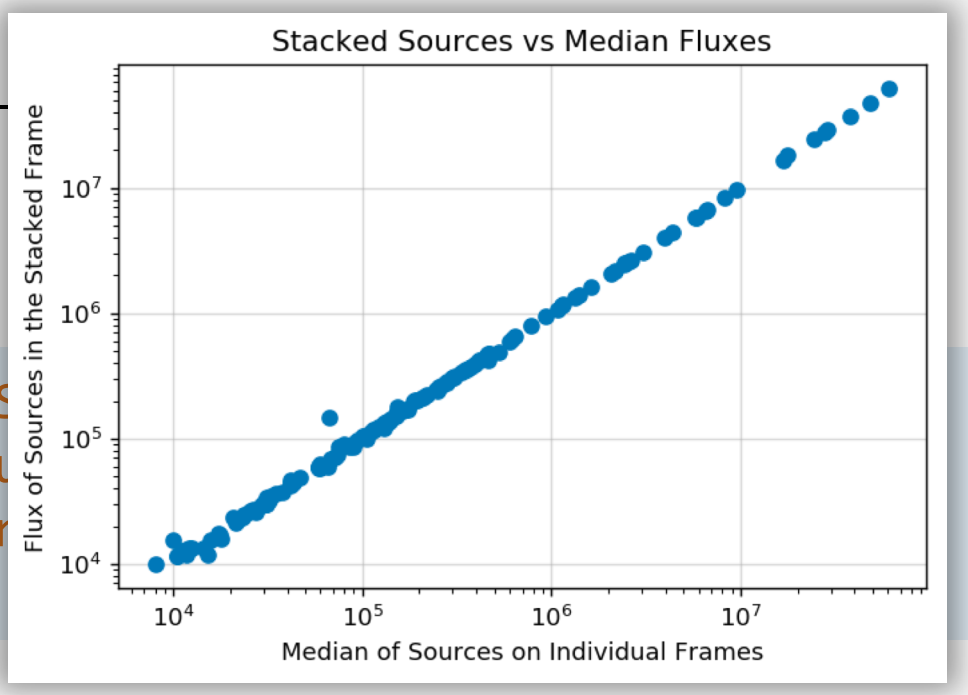
Adding Labels and Title

```
fig, ax = plt.subplots()

ax.plot(flux_median, flux_stack, 'o’)

ax.set_xscale('log')
ax.set_yscale('log’)

ax.set_xlabel('Median of Sources on Individual Frames')
ax.set_ylabel('Flux of Sources in the Stacked Frame')
ax.set_title('Stacked Sources vs Median Fluxes')
ax.grid(alpha=0.5)

plt.show()
```

Wrapping Up a Real Case

Adding Labels and Title

```
fig, ax = plt.subplots()

ax.plot(flux_median, flux_

ax.set_xscale('log')
ax.set_yscale('log')

ax.set_xlabel('Median of S
ax.set_ylabel('Flux of Sou
ax.set_title('Stacked Sour
ax.grid(alpha=0.5)

plt.show()
```



Wrapping Up a Real Case

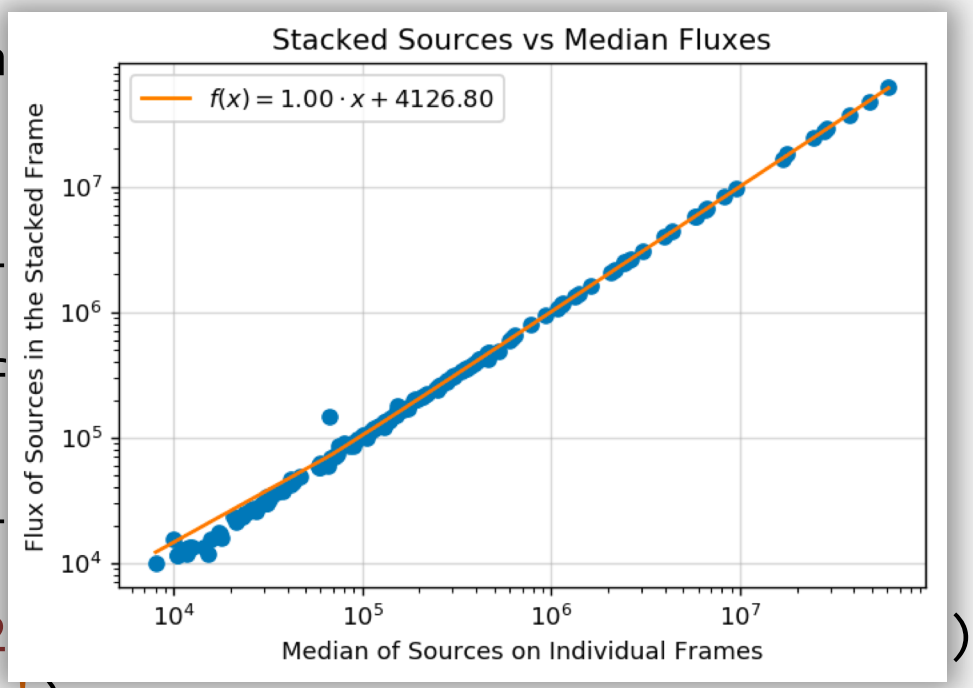
Linear Fitting

```
...
p = np.polyfit(flux_median, flux_stack, 1)

x_fit = np.linspace(
    flux_median.min(), flux_median.max(), 1000)

y_fit = np.polyval(p, x_fit)

ax.loglog(flux_median, flux_stack, 'o')
ax.plot(x_fit, y_fit, '-’,
    label=u'$f(x) = {0:.2f}\cdot x + {1:.2f}$'.format(*p))
ax.legend(loc='upper left')
...
```

Wrapping Up a Real Case

Linear Fitting

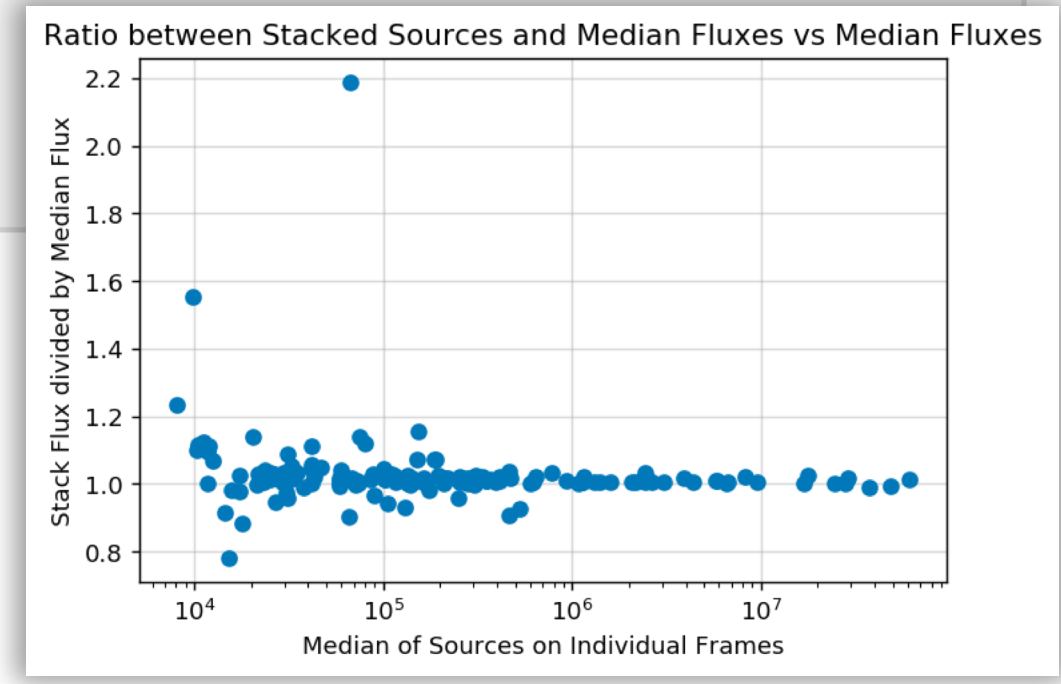
```
...
p = np.polyfit(flux_media

x_fit = np.linspace(
    flux_median.min(), fl

y_fit = np.polyval(p, x_f

ax.loglog(flux_median, fl
ax.plot(x_fit, y_fit, '-'
    label=u'$f(x) = {0:.2
ax.legend(loc='upper left')
...
```



Wrapping Up a Real Case

Plotting ratio

```
...
x = flux_median
y = flux_stack / flux_median

ax.plot(x, y, 'o')
ax.set_xscale('log')
...
```



Wrapping Up a Real Case

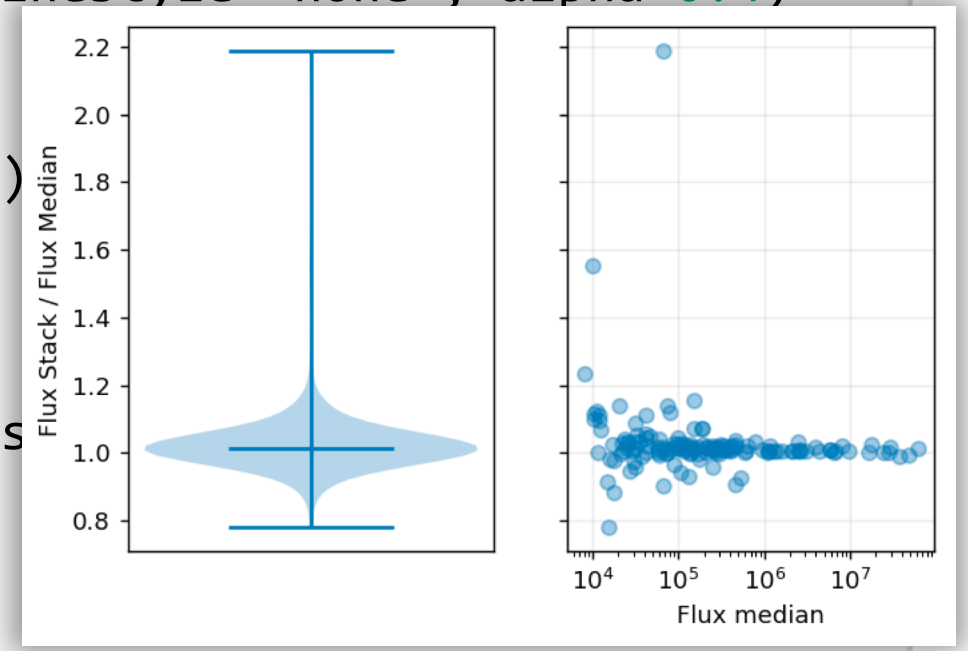
Adding a Violin Plot

```
fig, axes = plt.subplots(ncols=2, sharey=True)

ax1 = axes[1]
ax1.plot(x, y, marker='o', linestyle='none', alpha=0.4)

ax1.set_xscale('log')
ax1.set_xlabel('Flux median')
ax1.grid(alpha=0.25)

ax2 = axes[0]
ax2.violinplot(y, showmedians=True)
ax2.set_ylabel('Flux Stack / Flux Median')
ax2.set_xticks([])
...
```

Wrapping Up a Real Case

Adding a Violin Plot

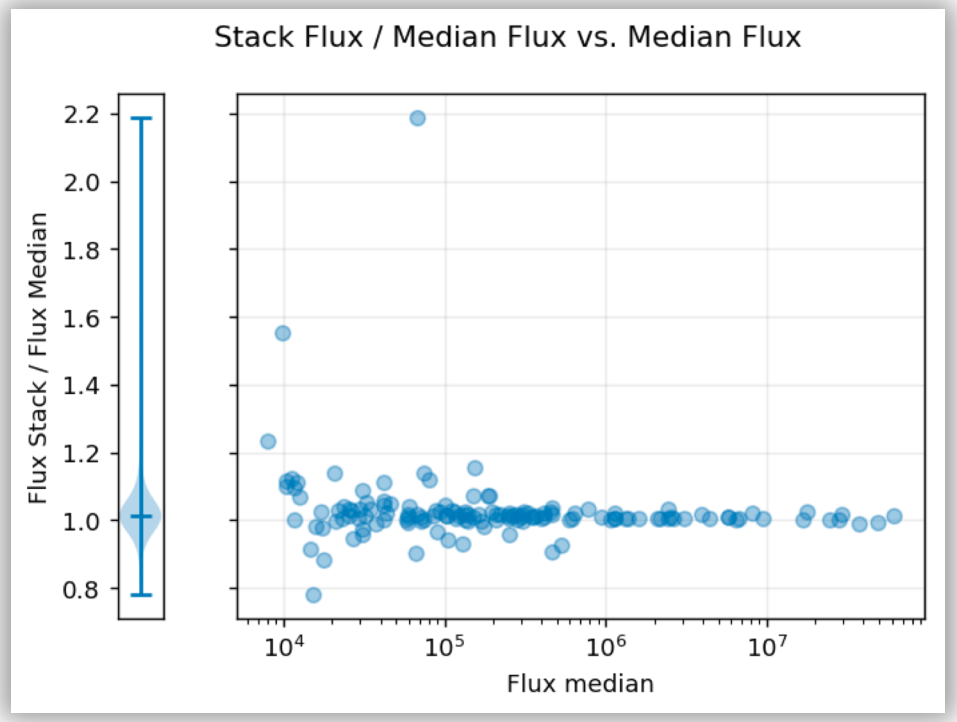
```
fig, axes = plt.subplots(ncols=2, sharey=True)

ax1 = axes[1]
ax1.plot(x, y, marker='o', linestyle='none', alpha=0.4)

ax1.set_xscale('log')
ax1.set_xlabel('Flux median')
ax1.grid(alpha=0.25)

ax2 = axes[0]
ax2.violinplot(y, showmedians
ax2.set_ylabel('Flux Stack /
ax2.set_xticks([])
...
```



Wrapping Up a Real Case

Adding a Violin Plot

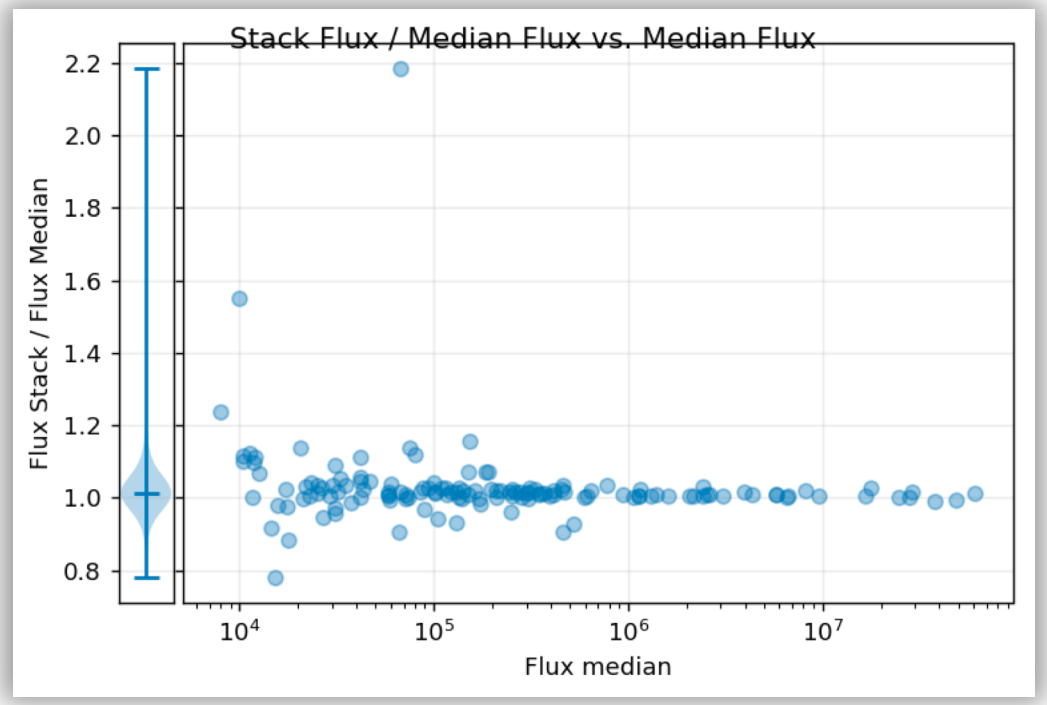
```
fig, axes = plt.subplots(ncols=2, sharey=True,
    gridspec_kw={'width_ratios': [1, 15]})
```



Wrapping Up a Real Case

Adding a Violin Plot

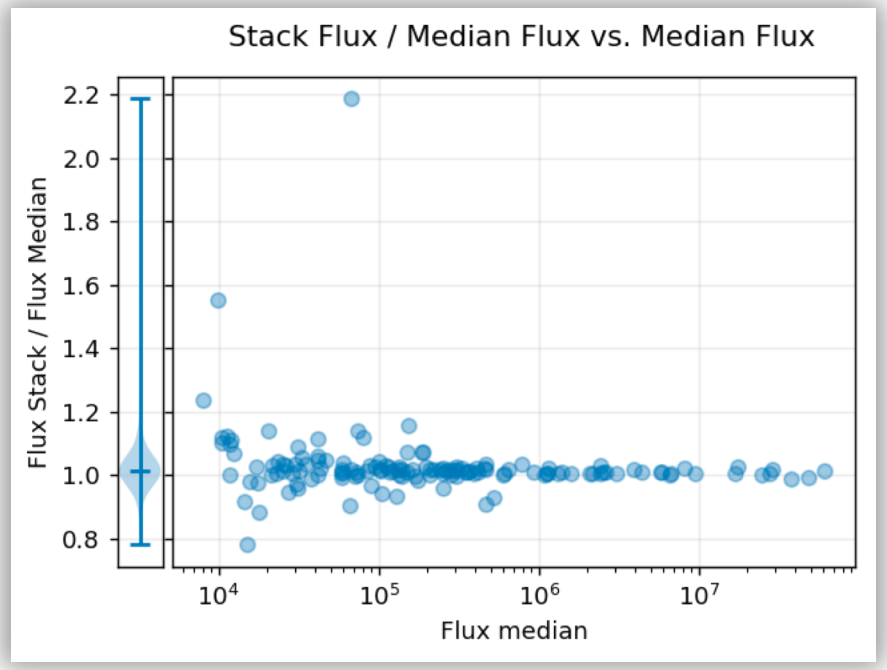
```
fig.tight_layout(w_pad=0)
plt.show()
```



Wrapping Up a Real Case

Adding a Violin Plot

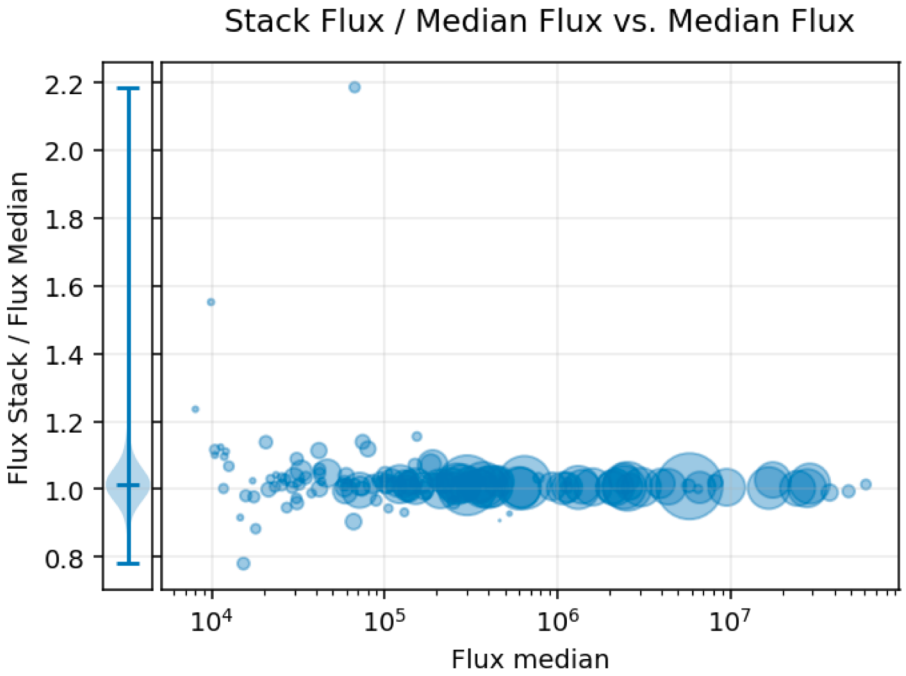
```
fig.tight_layout(w_pad=0, rect=[0, 0.05, 0.85, 0.95])
plt.show()
```



Wrapping Up a Real Case

Using `plt.scatter` with different marker sizes

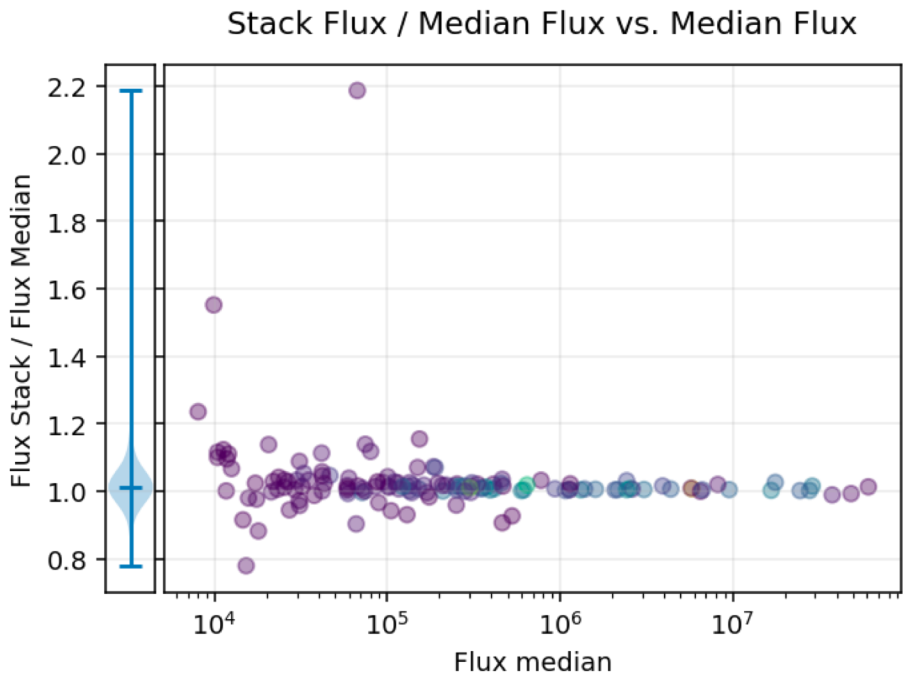
```
ax1 = axes[1]
ax1.scatter(x, y, s=z, marker='o', alpha=0.4)
```



Wrapping Up a Real Case

Using `plt.scatter` with different marker colors

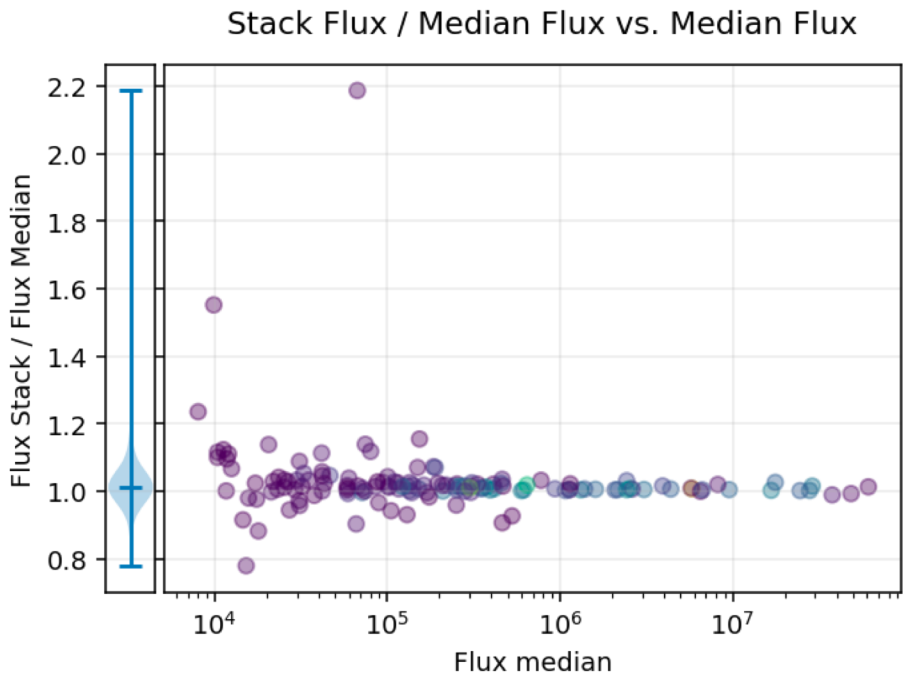
```
ax1 = axes[1]
ax1.scatter(x, y, c=z, marker='o', alpha=0.4)
```



Wrapping Up a Real Case

Using `plt.scatter` with different marker colors

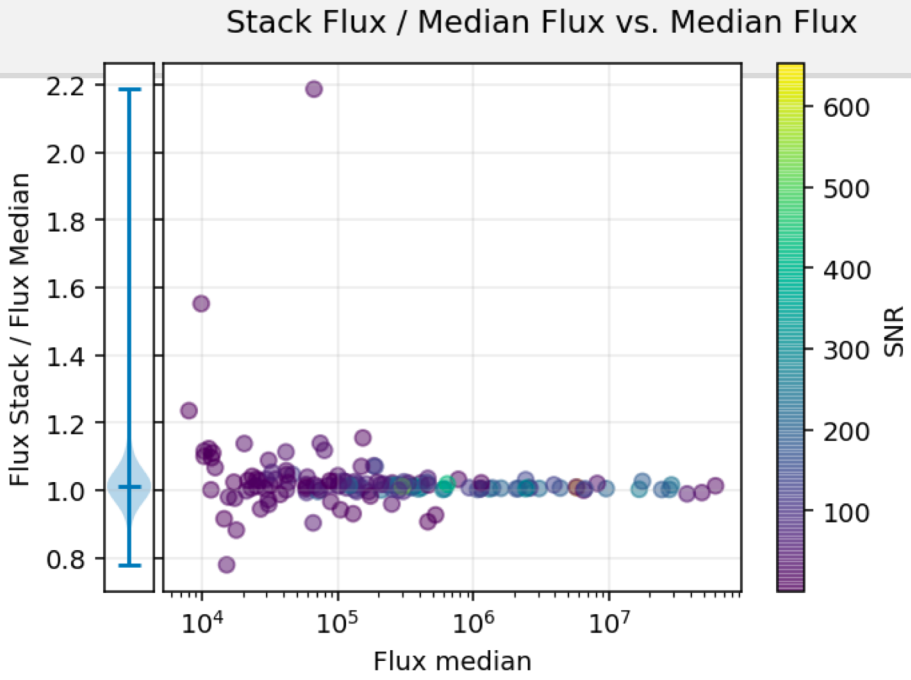
```
ax1 = axes[1]
ax1.scatter(x, y, c=z, marker='o', alpha=0.4)
```



Wrapping Up a Real Case

Using `plt.scatter` with different marker colors

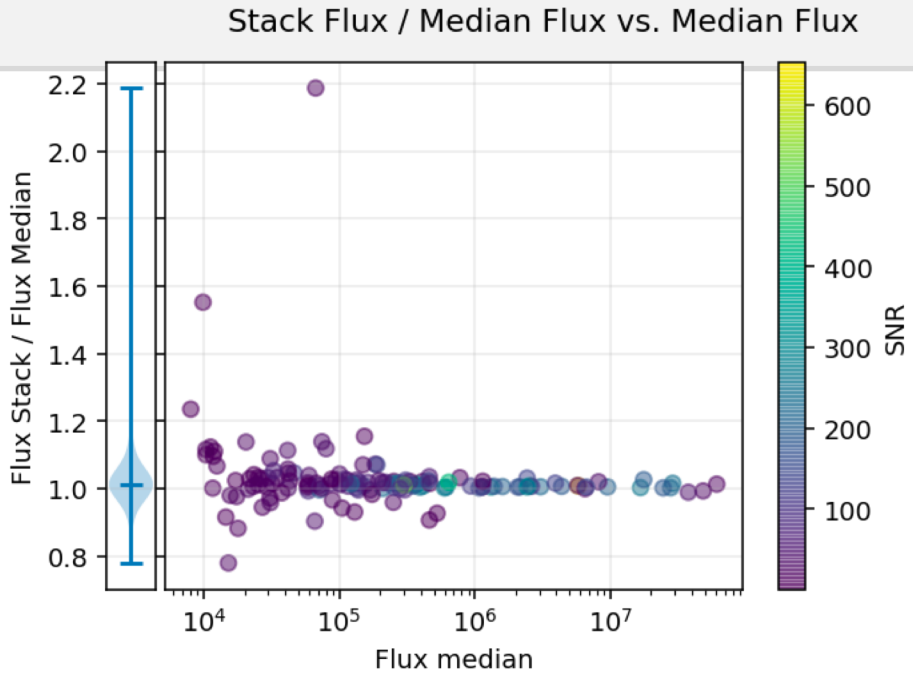
```
ax1 = axes[1]
im1 = ax1.scatter(x, y, c=z, marker='o', alpha=0.4)
...
fig.colorbar(im1, label='SNR’)
plt.show()
```



Wrapping Up a Real Case

Using `plt.scatter` with different marker colors

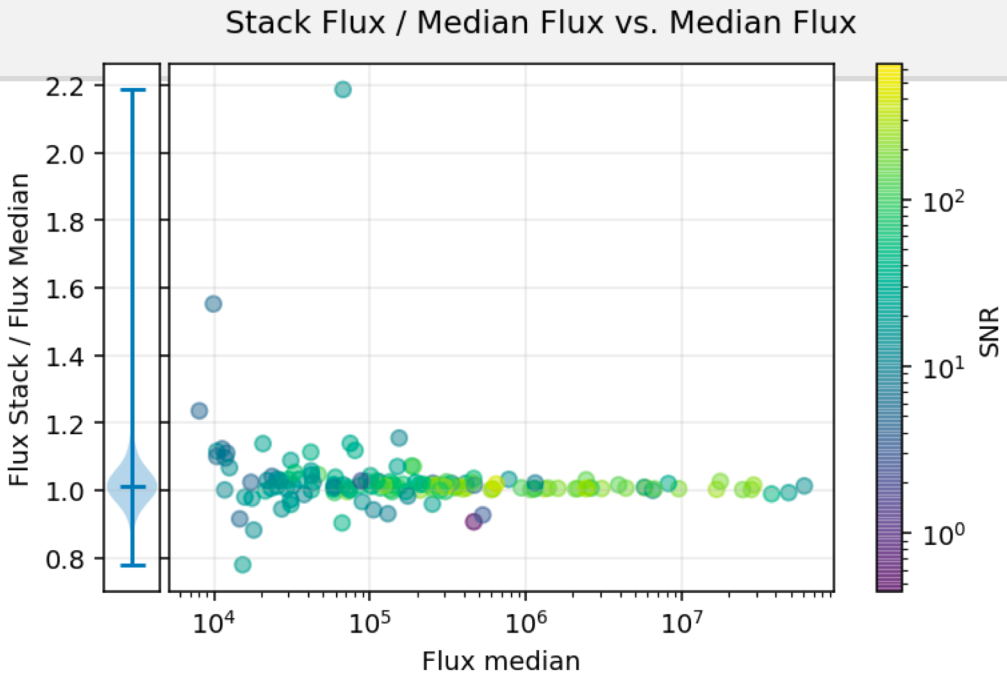
```
from matplotlib import colors as mcolors
...
im1 = ax1.scatter(x, y, c=z, marker='o', alpha=0.5,
                  norm=mcolors.LogNorm())
...
```



Wrapping Up a Real Case

Using `plt.scatter` with different marker colors

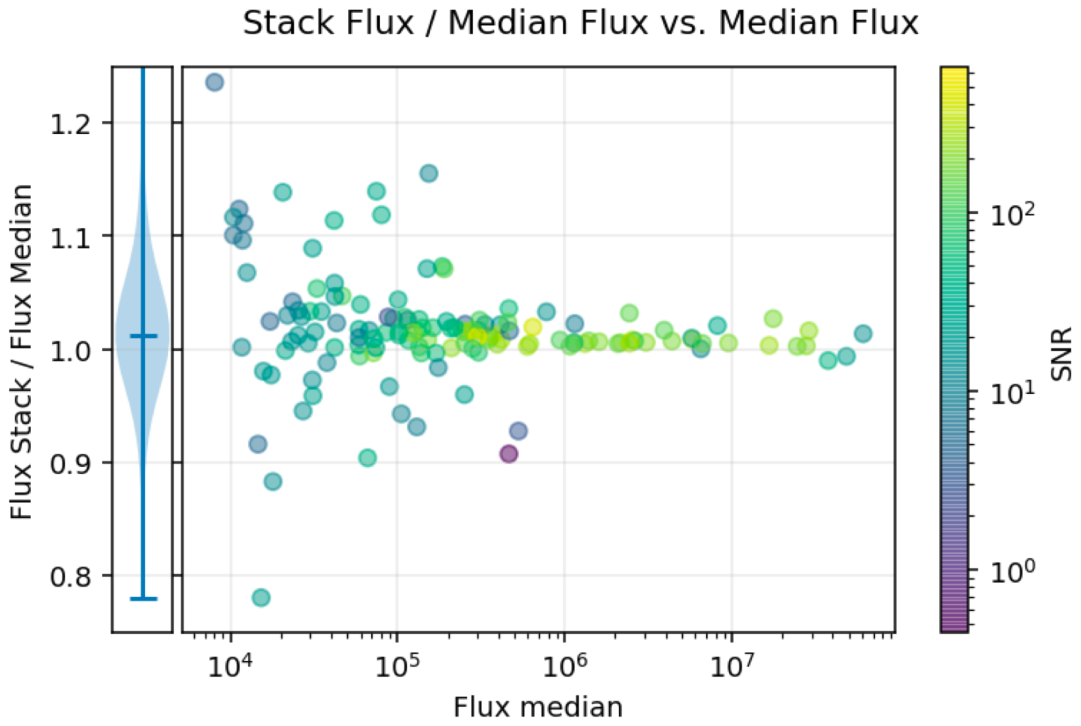
```
from matplotlib import colors as mcolors
...
im1 = ax1.scatter(x, y, c=z, marker='o', alpha=0.5,
                  norm=mcolors.LogNorm())
...
```



Wrapping Up a Real Case

Set plot limits

```
ax1.set_ylim(0.75, 1.25)
```



Questions?