

# Python Tutorial Series 2019

## Data Analysis & Visualization 1

### NumPy, Matplotlib and 2D Plots

Ph.D. Bruno C. Quint - [bquint@gemini.edu](mailto: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

```
$ ipython
Python 3.6.8 |Anaconda, Inc.| (default, Dec 29 2018, 19:04:46)
Type 'copyright', 'credits' or 'license' for more information
IPython 7.4.0 -- An enhanced Interactive Python. Type '?' for help.

[In [1]: print("Hello World!!")
Hello World!!

In [2]: ]
```



# Why should I use iPython?

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

A screenshot of an iPython notebook interface. The title bar says "bquint — IPython: Users/bquint — ipython — 78x10" and the subtitle bar says "~ — IPython: Users/bquint — ipython". In the main area, there is an input cell labeled "[In 7]: "Hello world!". The user has typed ".**" and is using tab completion. A dropdown menu shows several methods: "capitalize()", "casefold()", "center()", "count()", "encode()", "endswith()", "expandtabs()", and "find()". The "center()" method is highlighted.**



# Why should I use iPython?

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

```
bquint — IPython: Users/bquint — ipython — 78x13
~ — IPython: Users/bquint — ipython

[In [7]: my_string = "Hello World!!!"

[In [8]: my_string.endswith?
Docstring:
S.endswith(suffix[, start[, end]]) -> bool

Return True if S ends with the specified suffix, False otherwise.
With optional start, test S beginning at that position.
With optional end, stop comparing S at that position.
suffix can also be a tuple of strings to try.
Type:    builtin_function_or_method
```



# Why should I use iPython?

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

A screenshot of an iPython terminal window. The title bar shows "bquint — IPython: Users/bquint — ipython — 79x8". The window contains the following text:

```
In [25]: %time print("Measuring time!")
Measuring time!
CPU times: user 178 µs, sys: 130 µs, total: 308 µs
Wall time: 297 µs

In [26]:
```



# What about Jupyter?

Before we start

- 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

```
Python-Tutorial-Series — jupyter-notebook — 100x26
...GoogleDrive/My Drive — -bash ...
... ~/Tutorials — -bash ...
... --Series — jupyter-notebook ...
... b1quint/DRAGONS — -bash ...
(drangons) (master) bquint@bquint-ml1 [~/GitHub/b1quint/Python-Tutorial-Series]
[$ jupyter notebook
[I 17:41:45.279 NotebookApp] [jupyter_nbextensions_configurator] enabled 0.4.1
[I 17:41:45.279 NotebookApp] Serving notebooks from local directory: /Users/bquint/GitHub/b1quint/Python-Tutorial-Series
[I 17:41:45.279 NotebookApp] The Jupyter Notebook is running at:
[I 17:41:45.279 NotebookApp] http://localhost:8888/?token=6d25b8473440d85759085d39e722b9532455d6caf7e5a97d
[I 17:41:45.279 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 17:41:45.284 NotebookApp]

To access the notebook, open this file in a browser:
file:///Users/bquint/Library/Jupyter/runtime/nbsolver-36410-open.html
Or copy and paste one of these URLs:
http://localhost:8888/?token=6d25b8473440d85759085d39e722b9532455d6caf7e5a97d
```



# What about Jupyter?

Starting the server

localhost:8888/tree

jupyter

Files    Running    Clusters    Nbextensions

Select items to perform actions on them.

	Name	Last Modified	File size
<input type="checkbox"/> 0	/		
<input type="checkbox"/> flaminos_2		a month ago	
<input type="checkbox"/> gmos		21 hours ago	
<input type="checkbox"/> gsaoi		2 months ago	
<input type="checkbox"/> Icon		49 years ago	0 B



# What about Jupyter?

Browsing

The screenshot shows a web browser window with two tabs: "gmos/GS-2006B-Q-18-3/notebooks" and "data\_analysis\_GS-2006B-Q-18". The main content area is titled "jupyter" and displays a file list under the "Files" tab. The list includes:

Name	Last Modified	File size
..	seconds ago	
<input checked="" type="checkbox"/> data_analysis_GS-2006B-Q-18-3.ipynb	21 hours ago	11.5 MB
<input type="checkbox"/> Icon	49 years ago	0 B

Buttons for "Duplicate", "Shutdown", "View", "Edit", "Upload", "New", and "Logout" are visible at the top of the list.



# What about Jupyter?

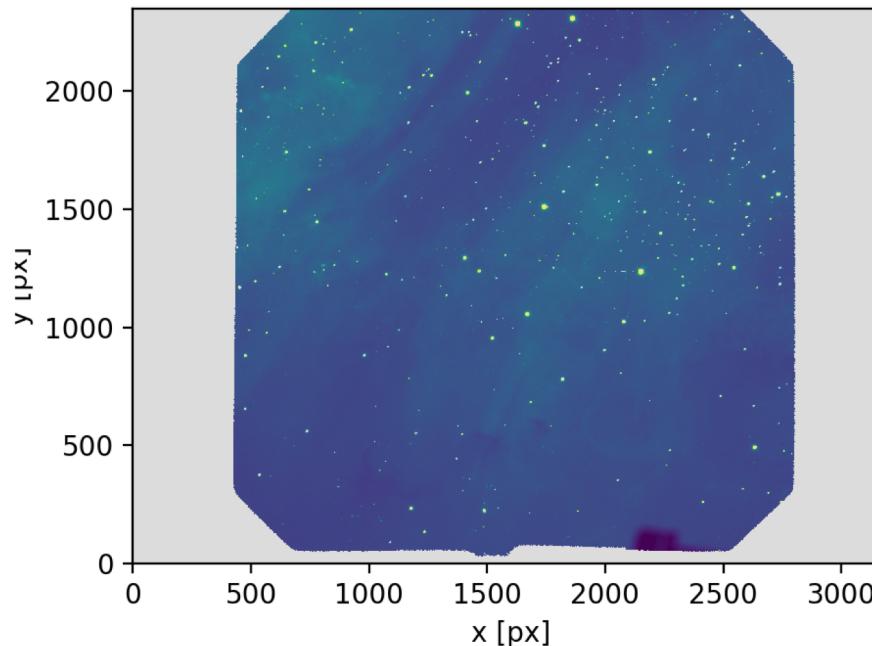
Browsing

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

..../red/S20061010S0030\_stack\_varclip\_sources.fits



GS-2006B-Q-18-3  
S20061010S0030\_stack\_varclip\_sources.fits





# Jupyter

## Different Modes and Cells

Edit  
Mode

Python  
Cell

Markdown  
Cell

In [1]: `x = 5 + 6`

This is a markdown cell|

Command  
Mode

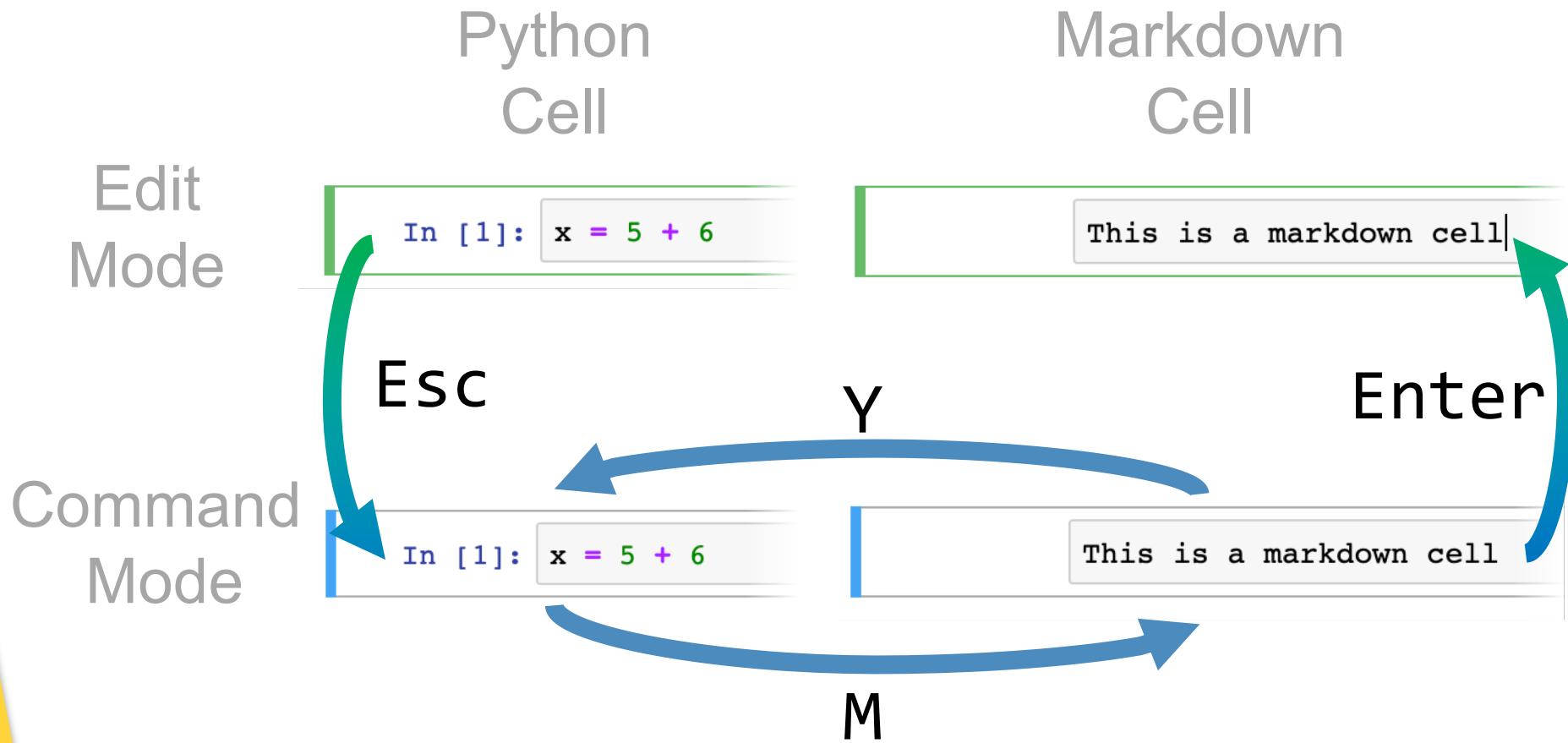
In [1]: `x = 5 + 6`

This is a markdown cell



# Jupyter

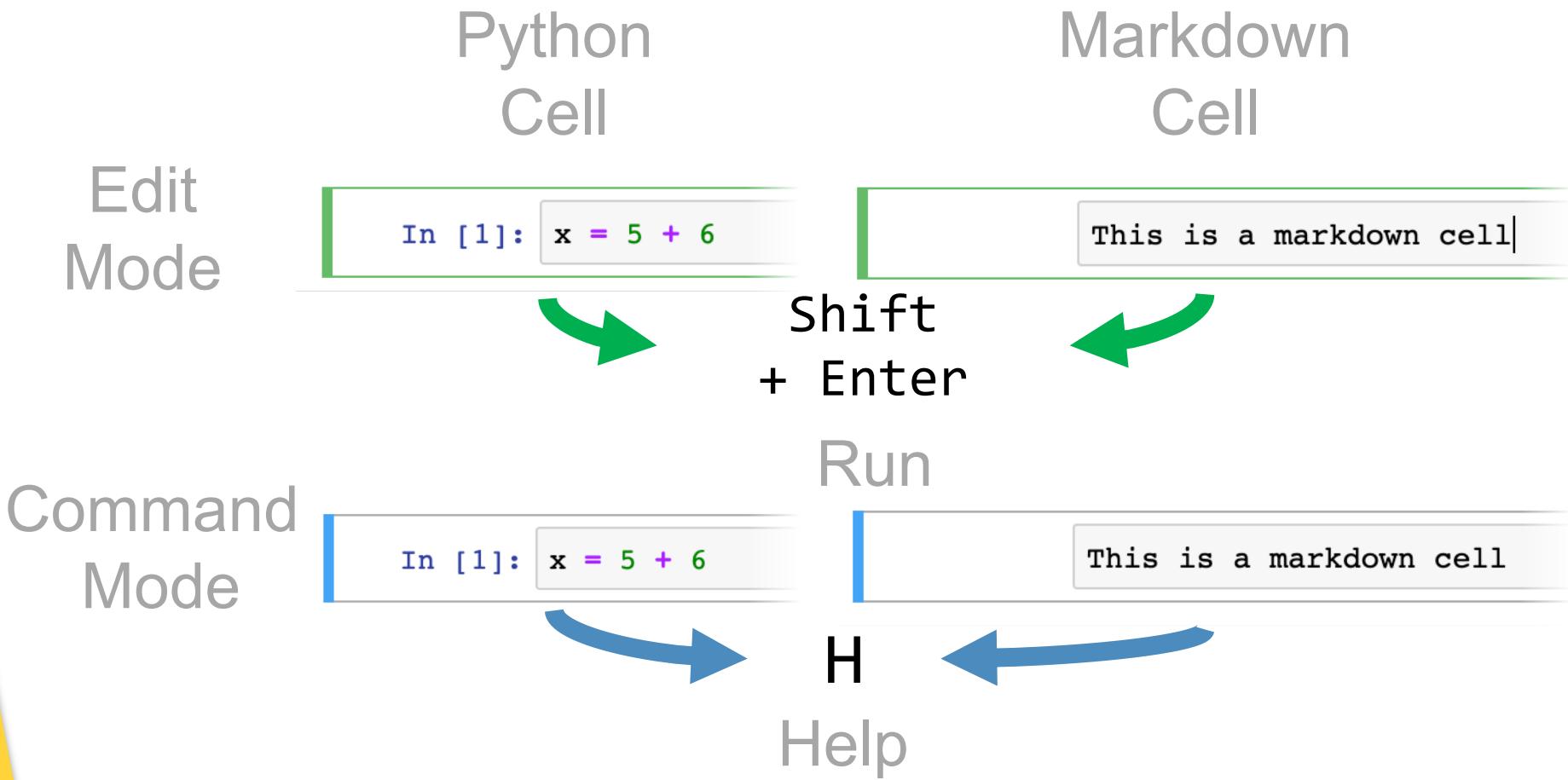
## Different Modes and Cells





# Jupyter

## Different Modes and Cells



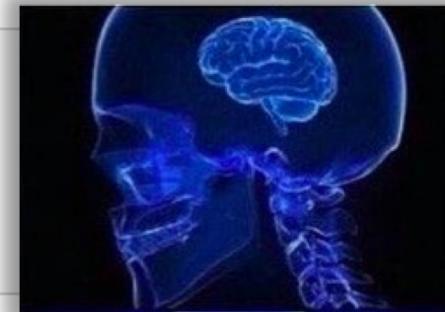


# NumPy Why you should use it now!!

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

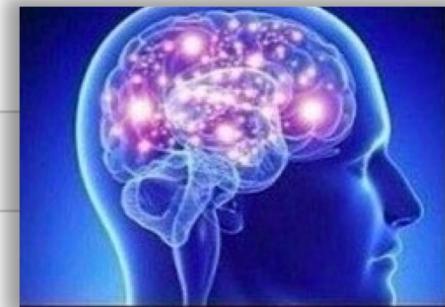
```
<class 'list'>
```

```
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

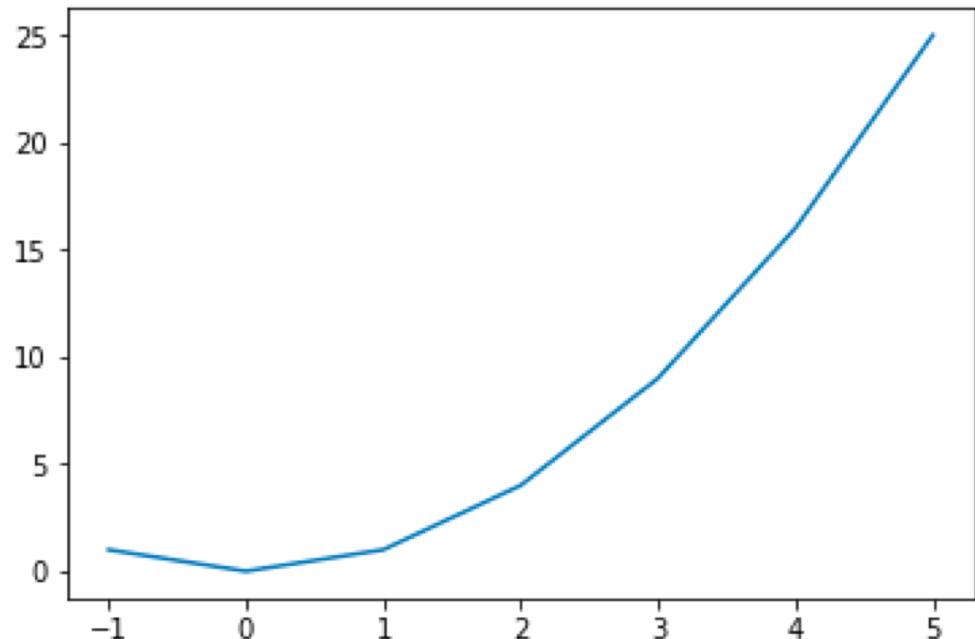
```
%%timeit
x_array = np.arange(-1, 6)
y_array = x_array ** 2
```

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



# Matplotlib A simple plot

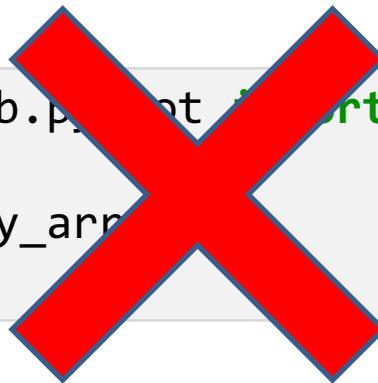
```
from matplotlib.pyplot import *
plot(x_array, y_array)
show()
```





# Matplotlib A simple plot

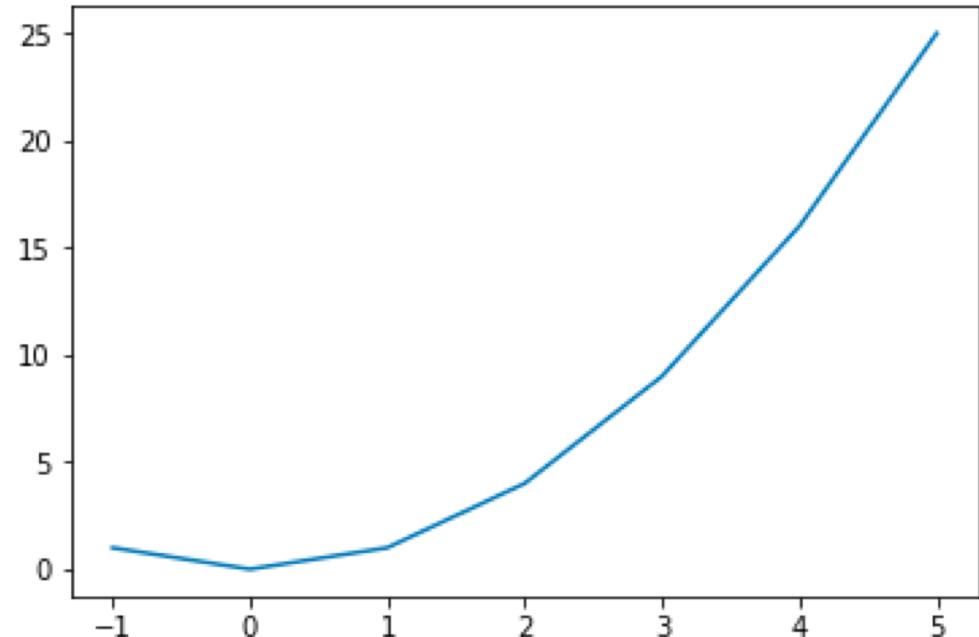
```
from matplotlib.pyplot import *
plot(x_array, y_array)
show()
```



Avoid using

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

To prevent variables redefinition  
and to keep references

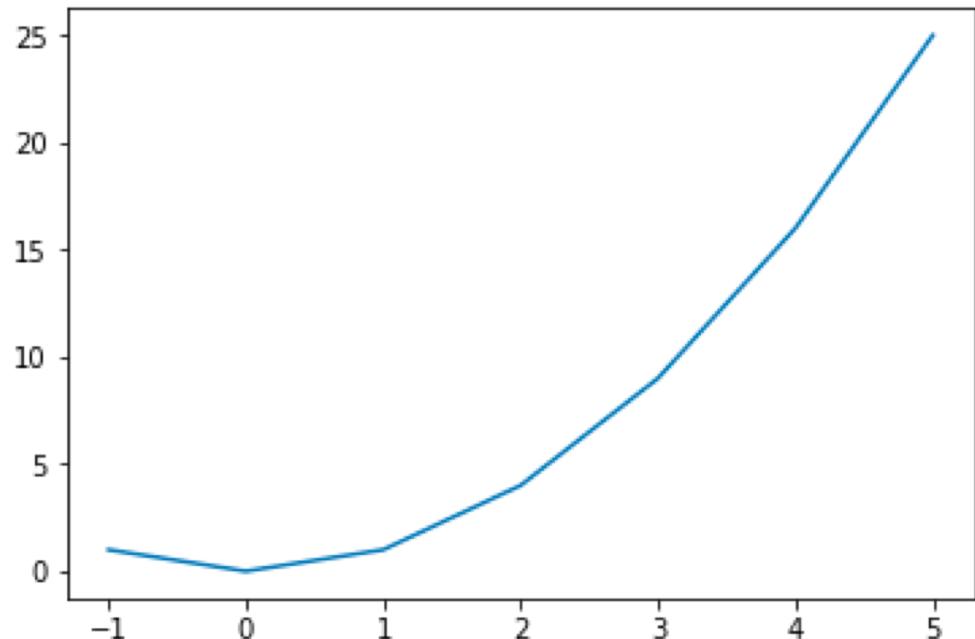




# Matplotlib A simple plot

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

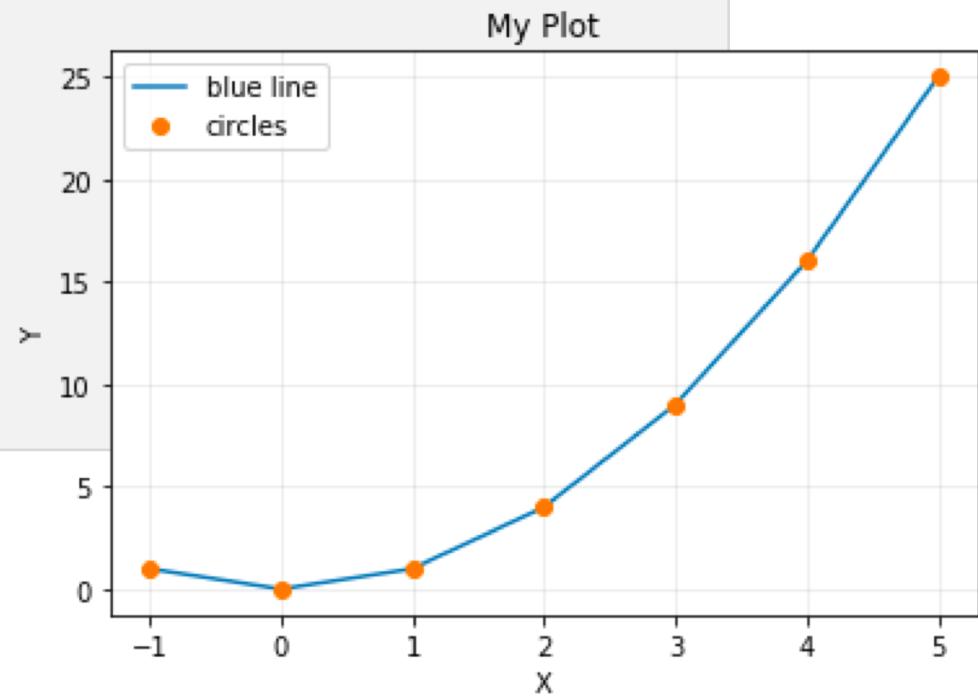
```
from matplotlib import pyplot as plt

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

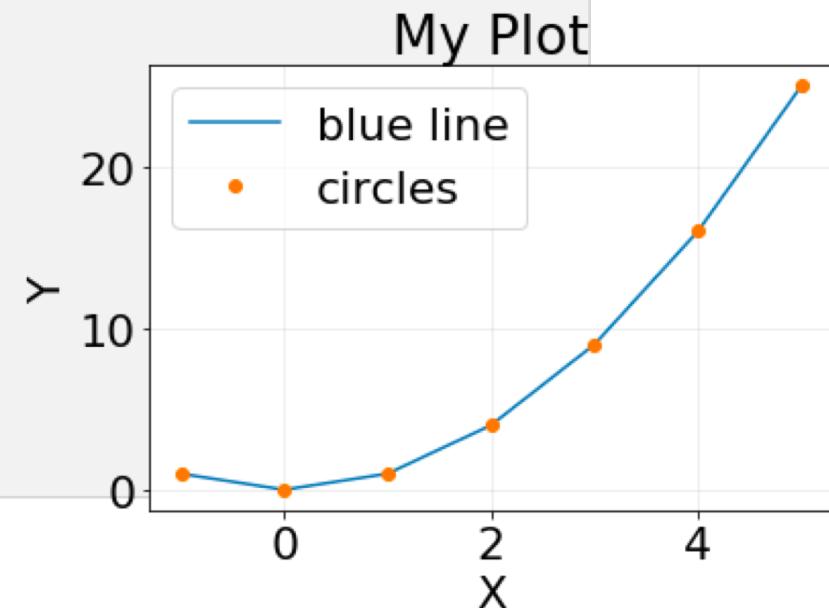
plt.rcParams.update({'font.size': 22})

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

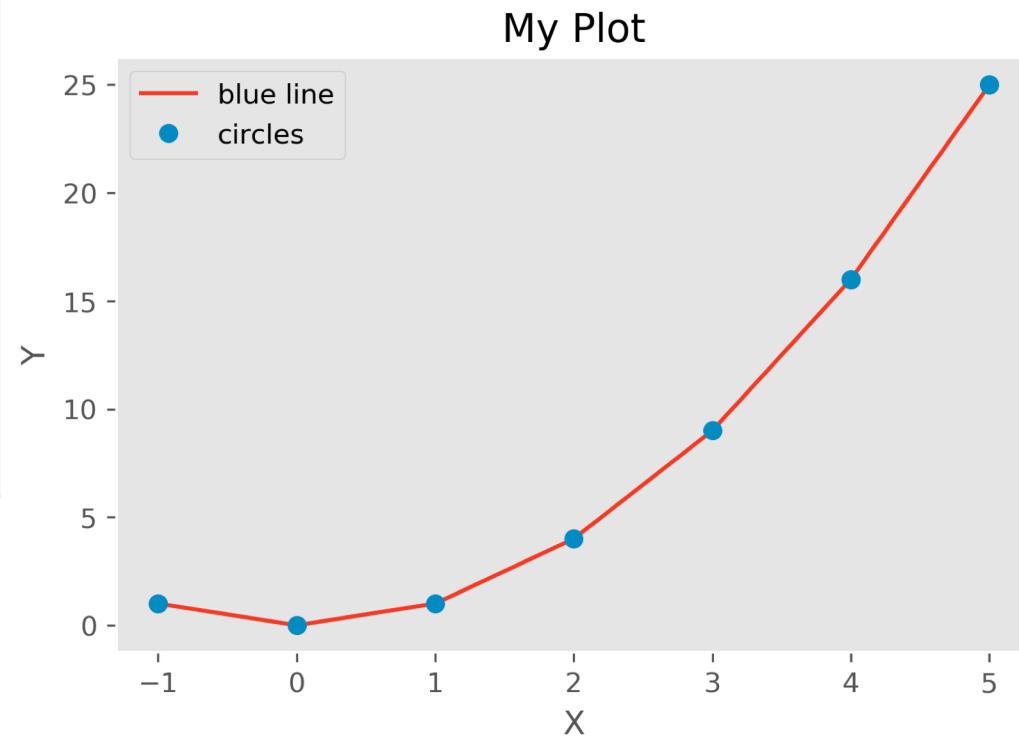
```
plt.style.use('ggplot'):
```

```
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

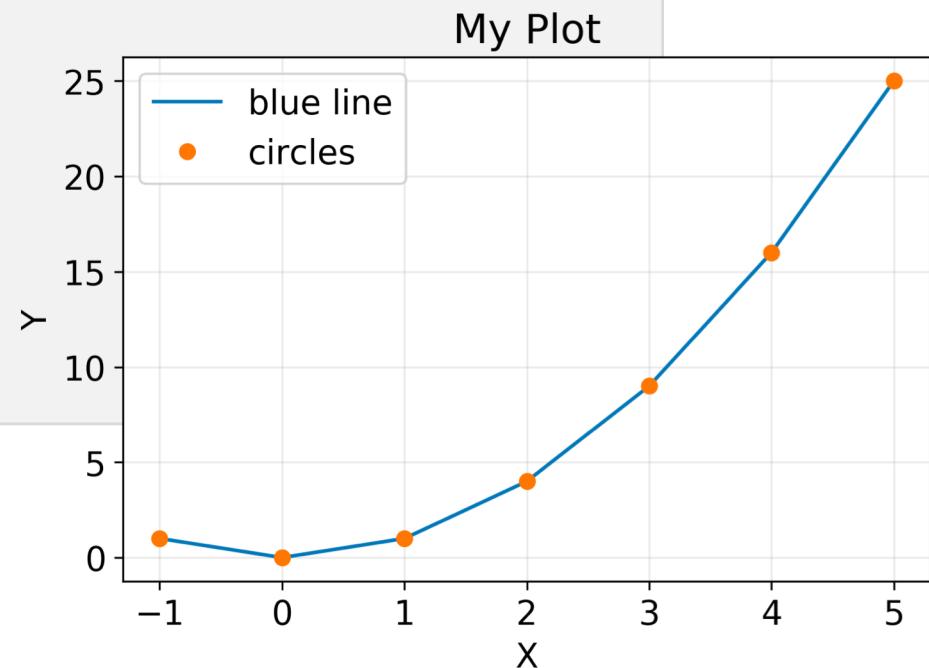
```
my_figure, my_axes = plt.subplots()

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')

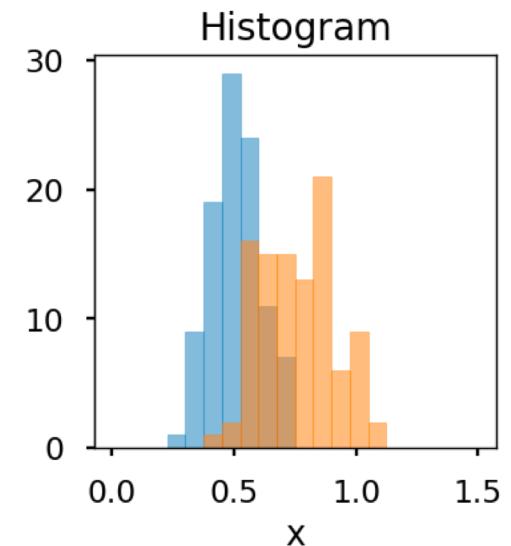
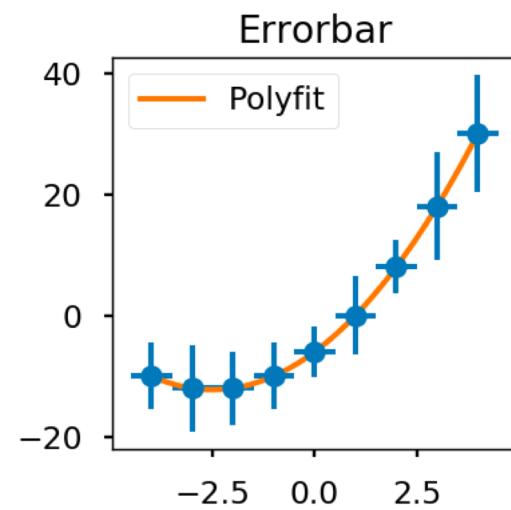
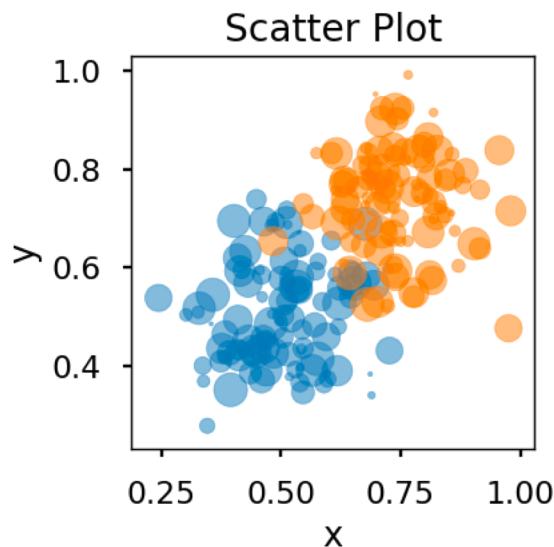
plt.show()
```

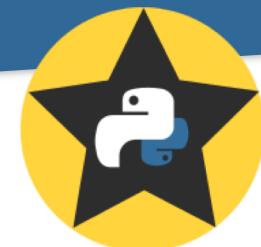




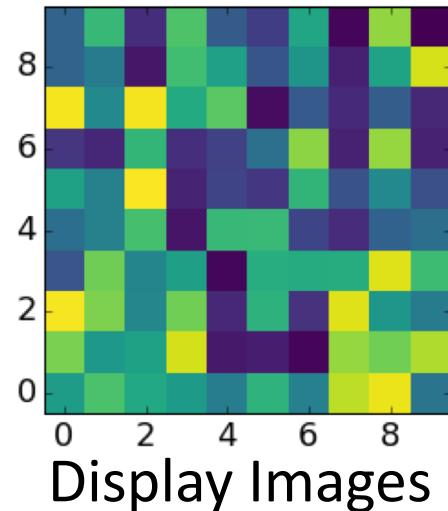
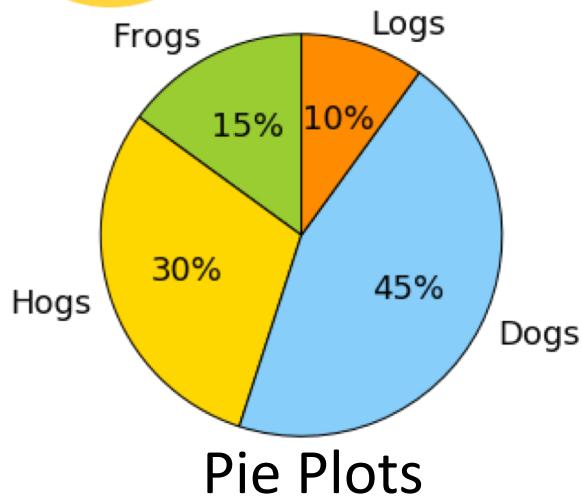
# MatPlotLib Plot types

<https://matplotlib.org/gallery>

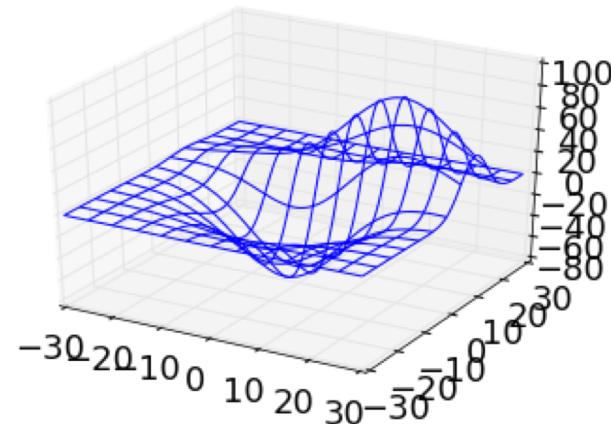




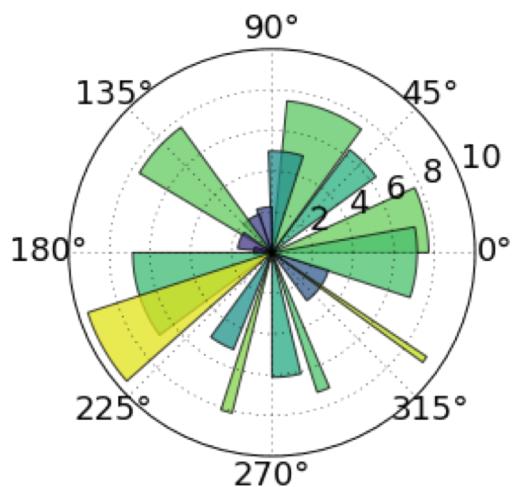
# MatPlotLib Plot types



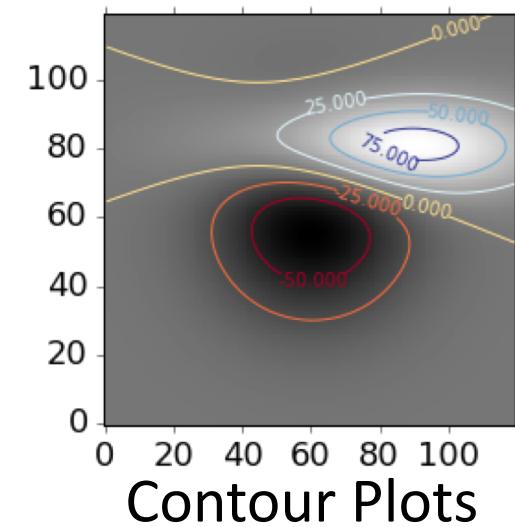
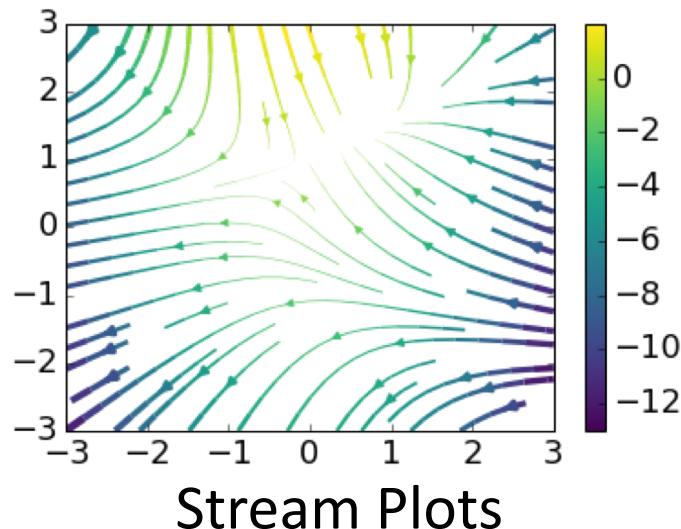
<https://matplotlib.org/gallery>



**Plots 3D**



**Polar 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
<b>4</b>	<b>FLUX_STACK</b>	<b>10</b>	<b>MEDIAN</b>
5	FLUX_0	11	SNR

Read text file using NumPy

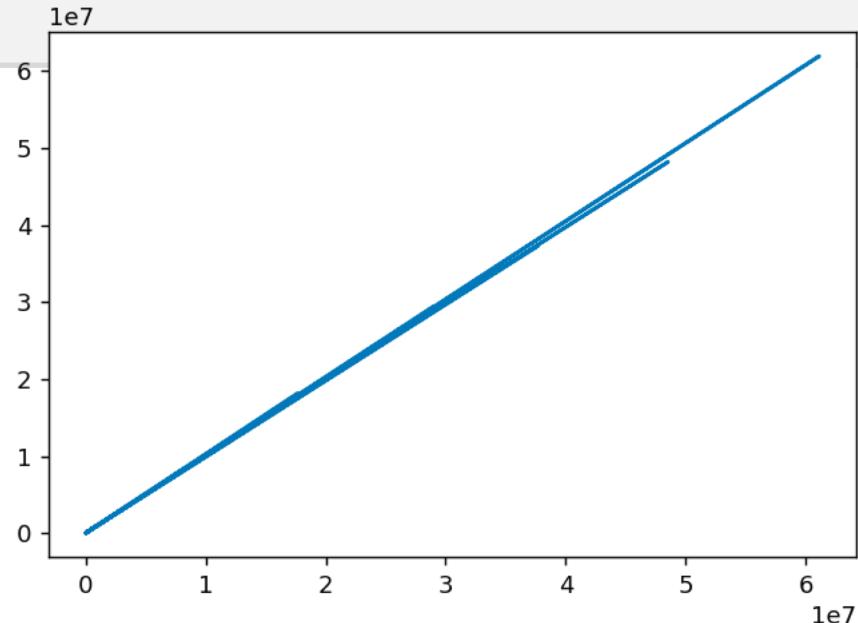
```
filename = 'data/my_table.txt'  
flux_stack, flux_median, snr = np.loadtxt(  
    filename, usecols=[4, 10, 11], unpack=True)
```



# Wrapping Up a Real Case

A simple plot

```
fig, ax = plt.subplots()  
  
ax.plot(flux_median, flux_stack)  
  
plt.show()
```

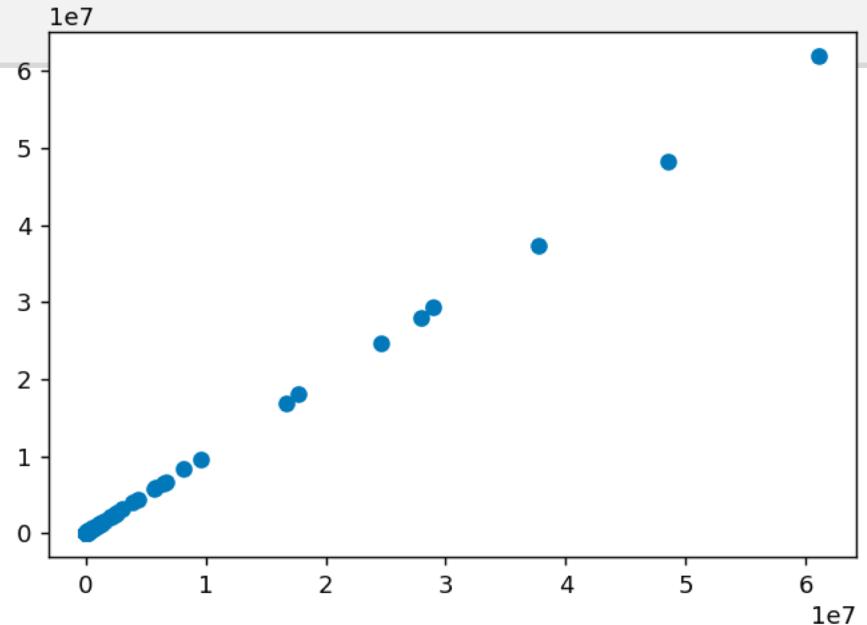




# Wrapping Up a Real Case

## Changing Marker and LineStyle

```
fig, ax = plt.subplots()  
  
ax.plot(flux_median, flux_stack, 'o')  
  
plt.show()
```

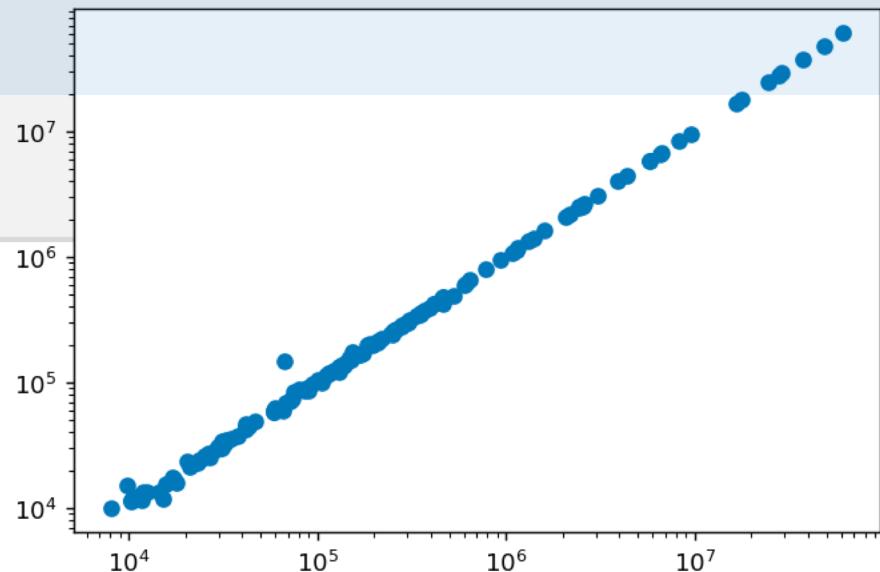




# Wrapping Up a Real Case

## 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

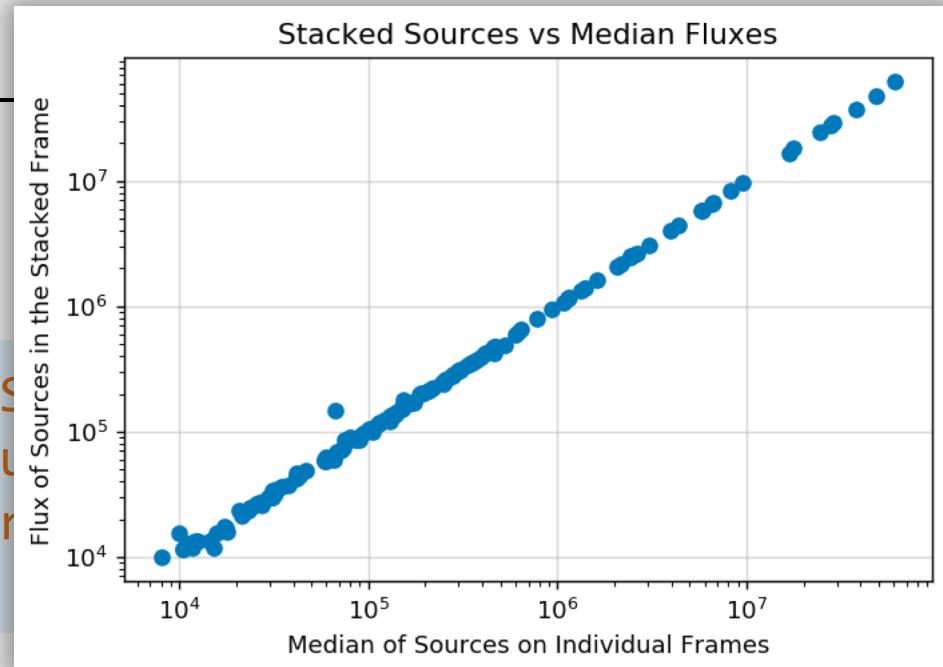
```
fig, ax = plt.subplots()

ax.plot(flux_median, flux_stacked)

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

## 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'$f(x) = {:.2f} \cdot x + {:.2f}$'.format(*p))
ax.legend(loc='upper left')

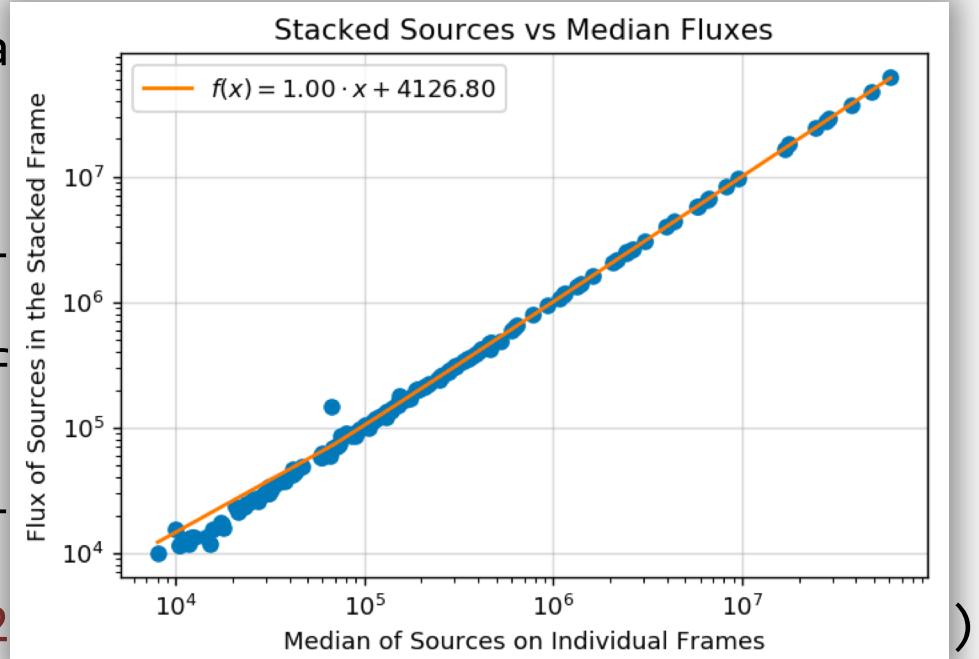
...
```



# 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')  
...
```

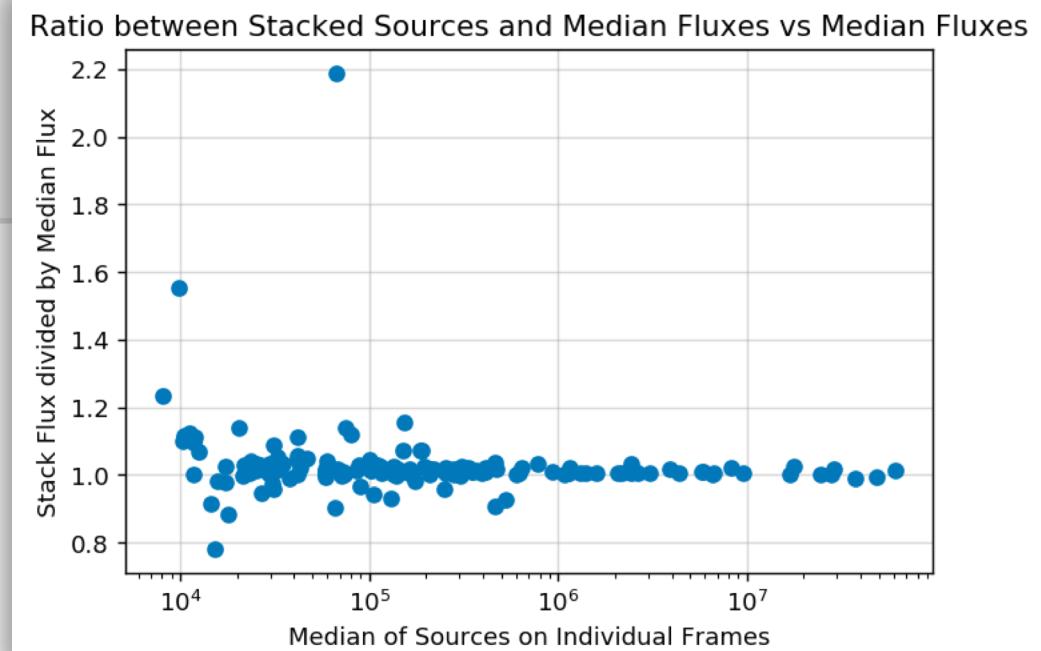




# Wrapping Up a Real Case

## 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

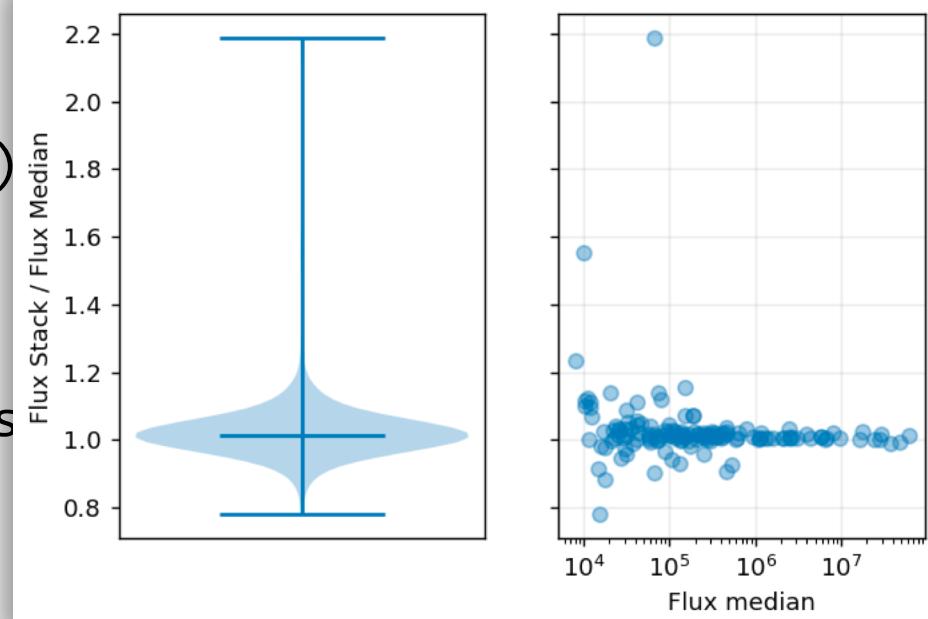
```
fig, axes = plt.subplots(ncols=2, sharey=True)

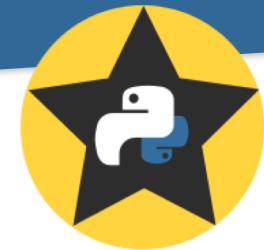
ax1 = axes[1]
ax1.plot(x, y, marker='o', linestyle='none', alpha=0.4)

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

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

...
```

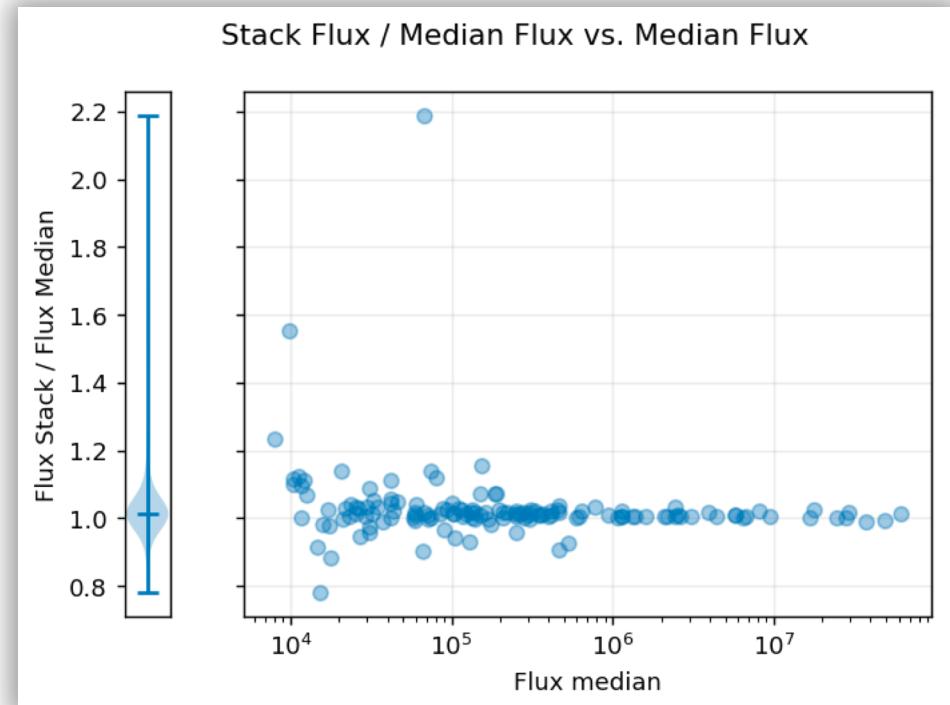




# Wrapping Up a Real Case

## Adding a Violin Plot

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

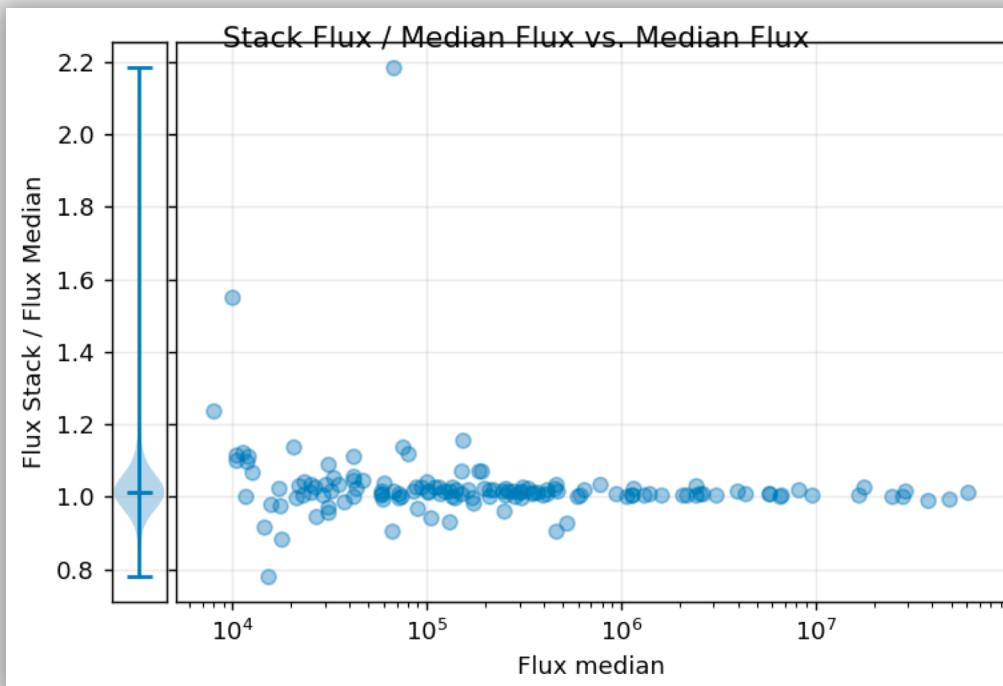




# Wrapping Up a Real Case

## Adding a Violin Plot

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

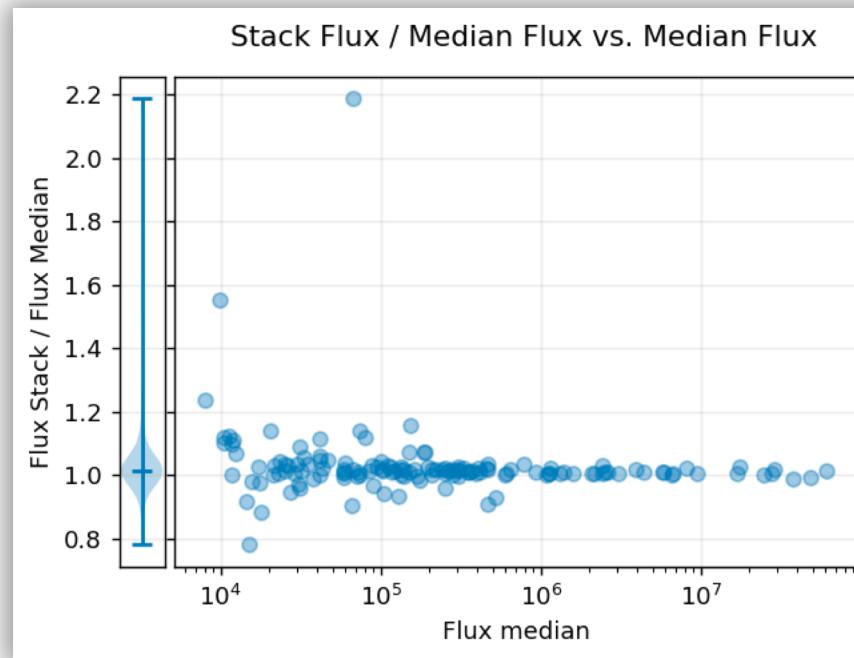




# Wrapping Up a Real Case

## Adding a Violin Plot

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

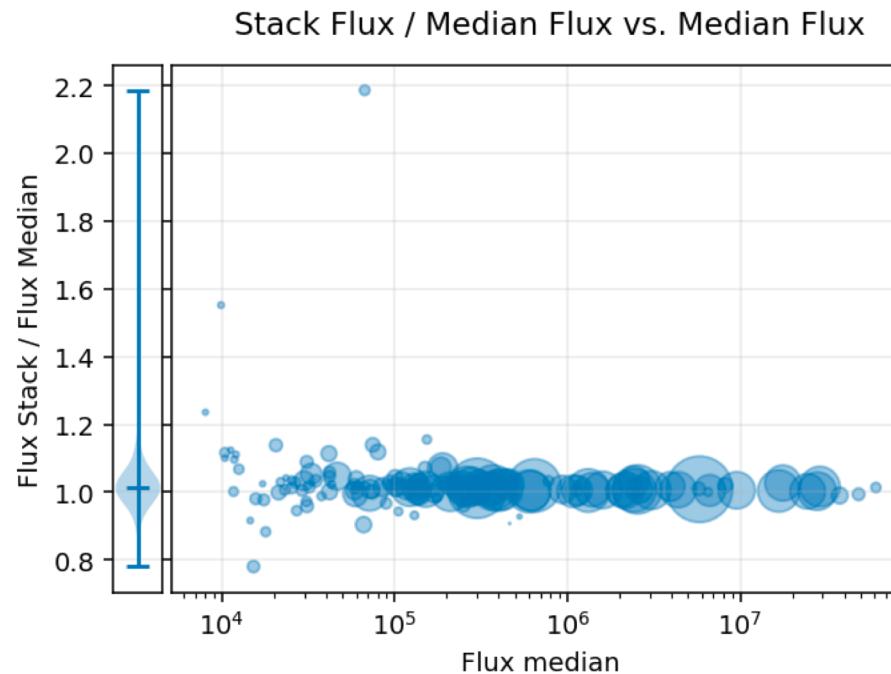




# Wrapping Up a Real Case

Using plt.scatter with different marker sizes

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

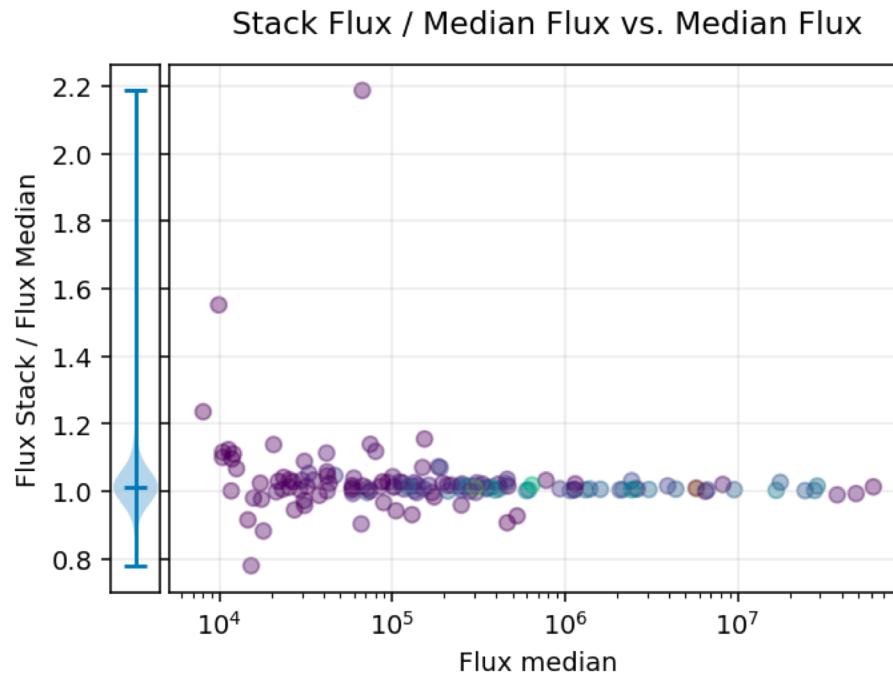




# Wrapping Up a Real Case

Using plt.scatter with different marker colors

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

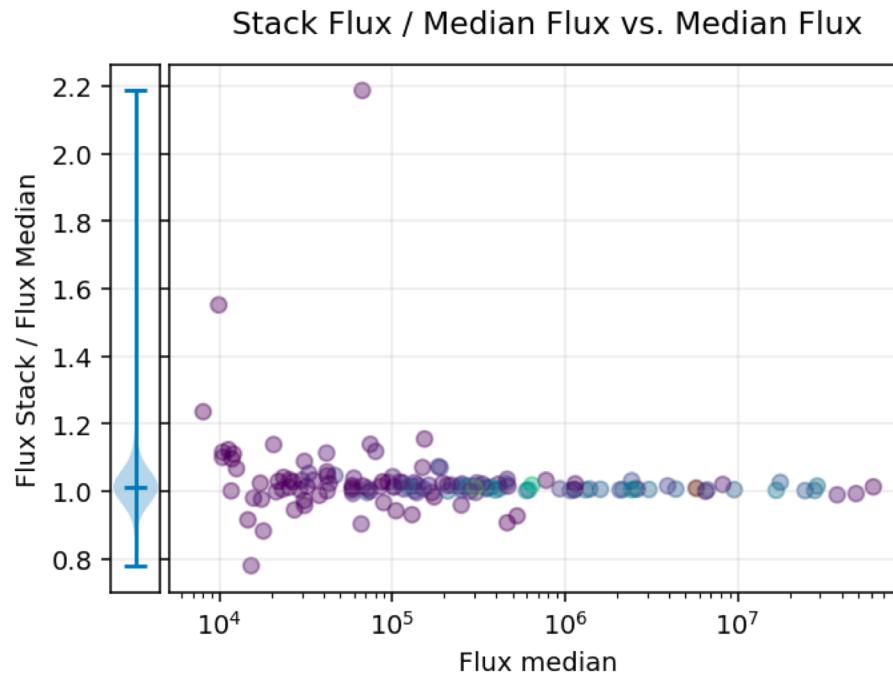




# Wrapping Up a Real Case

Using plt.scatter with different marker colors

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

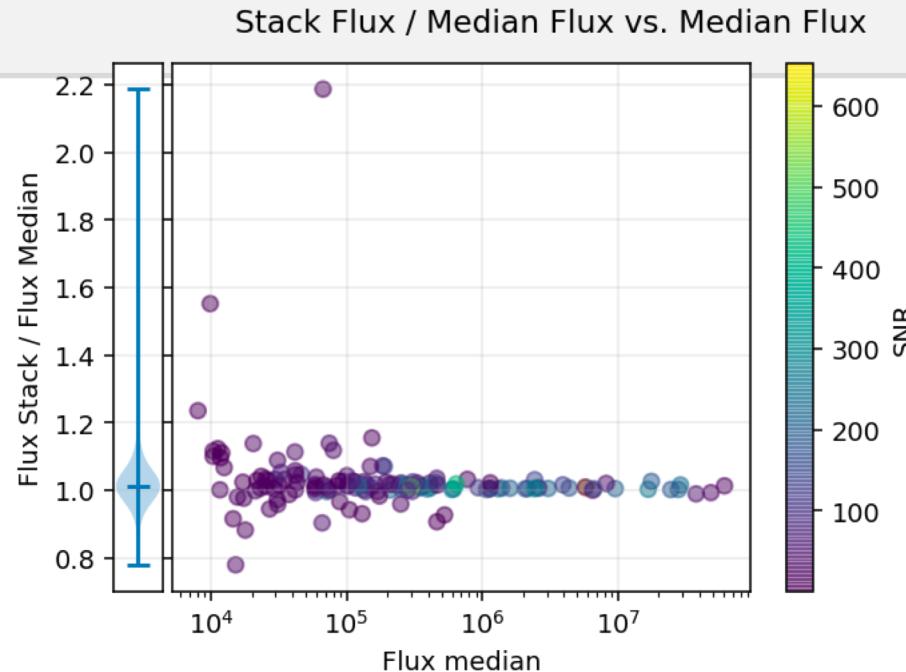




# Wrapping Up a Real Case

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()
```

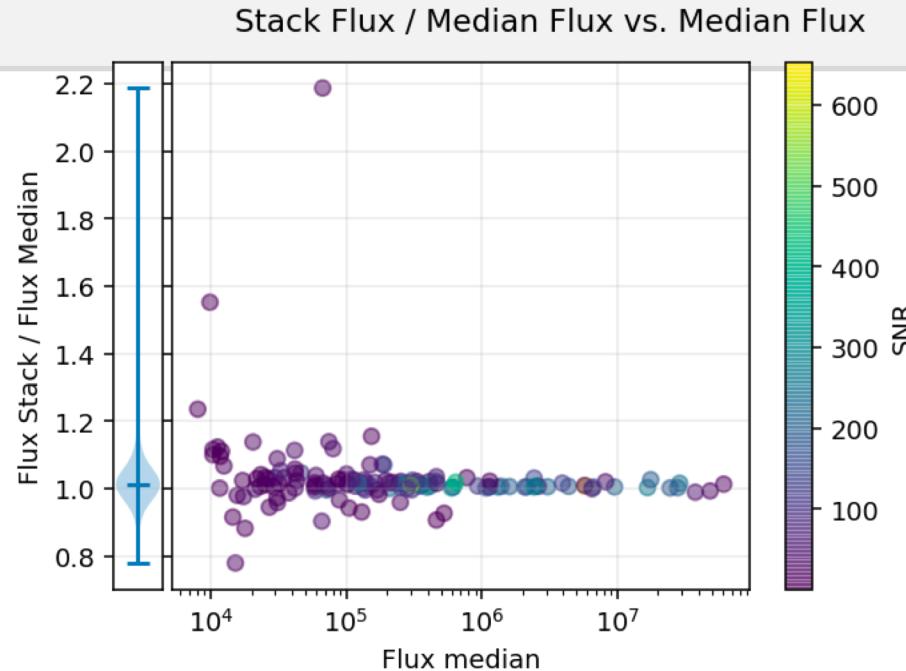




# Wrapping Up a Real Case

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())  
...
```

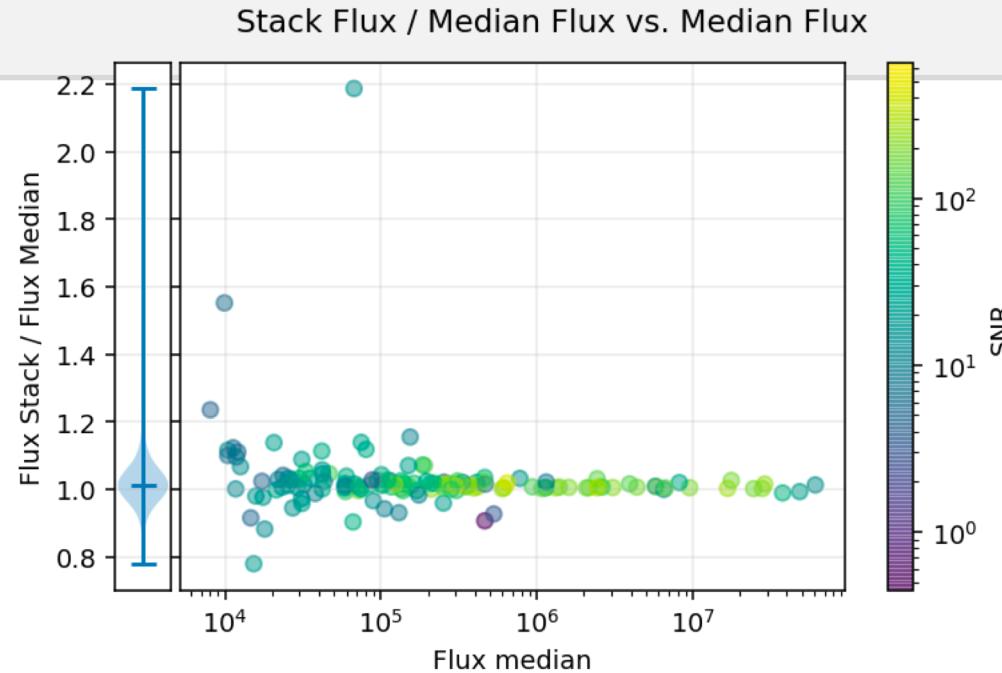




# Wrapping Up a Real Case

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())  
...
```

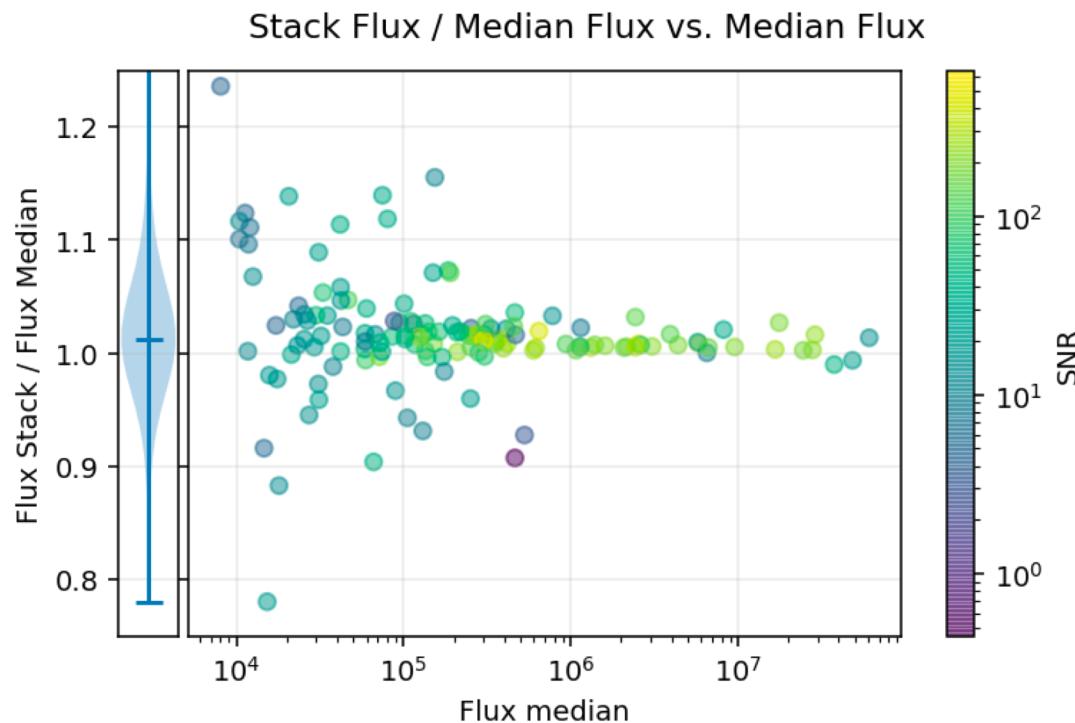




# Wrapping Up a Real Case

Set plot limits

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





# Questions?

