Packages Galore

In [ ]:
```
import os
import sys
import re
import warnings
os.getcwd()
```

Out[ ]:
```
'c:\\Users\\ebaca\\Desktop\\Astro 480 Labs and PSets\\Lab 4 - Data Reduction & Pho
tometry'
```

In [ ]:
```
sys.path.append('..\\..\\interacting_galaxies') # for laptop
sys.path.append('..\\..\\Interacting Galaxies Project') # for desktop
from funcrefs import fnrefs as rfs
from funcrefs import convenience_functions as cf

import numpy as np
import astropy.units as u
from copy import deepcopy

import matplotlib.pyplot as plt
from matplotlib.colors import LinearSegmentedColormap
from matplotlib.patches import Rectangle
import astropy.visualization as vis
import seaborn as sns

from astropy.io import fits
from astropy.wcs import WCS
import ccdproc as ccdp
from astropy.nddata import CCDData

import photutils as pu
from astropy.stats import sigma_clipped_stats, SigmaClip
# from specutils.analysis import fwhm    # pip install specutils
from photutils.detection import DAOStarFinder
from photutils.aperture import CircularAperture, CircularAnnulus, aperture_photomet
import photutils.background as pb
from photutils.psf import PSFPhotometry, IntegratedGaussianPRF
```

Notes for Observation Shifts

- for ngc 2998 if the picture given looks weird it's probably the spectrum pic
- darks are the same length as images you want to reduce (but you can also add some together)
- create a table that lists file names, filter, exposure time, and kind of file
- manually check for correct settings by downloading and opening fits file (don't trust preview screen)
- keep original data as raw
- ccd data reduction guide
- no overscans on ARCSAT

Data Reduction

CCD Data Reduction Guide

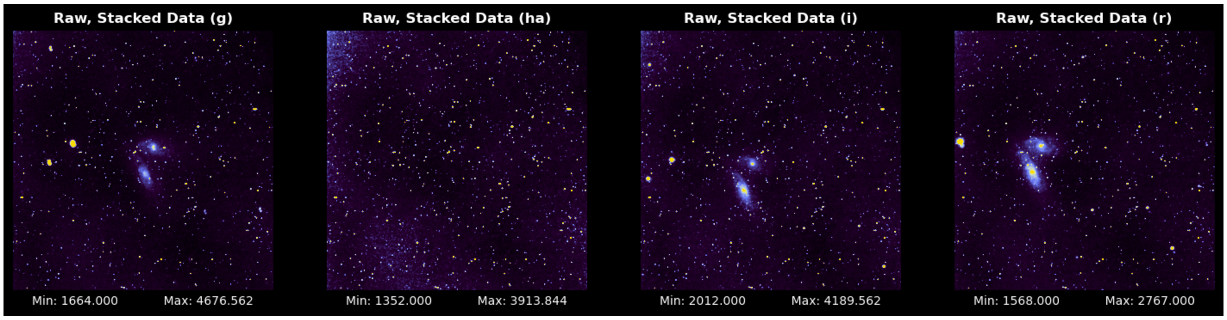
In [ ]:
```
# creating a custom colormap
N = 256
colors = ["black", "xkcd:very dark purple",
          "xkcd:cornflower", "xkcd:light blue grey", "xkcd:light khaki",
          "xkcd:dandelion"]
custom = LinearSegmentedColormap.from_list("custom", colors, N=N)
```

Plotting Uncalibrated Stacks

In [ ]:
```
raw_ngc4567 = {
    'g': rfs.create_stack('../../Interacting Galaxies Project/NGC 4567/raw data/',
    'ha': rfs.create_stack('../../Interacting Galaxies Project/NGC 4567/raw data/',
    'r': rfs.create_stack('../../Interacting Galaxies Project/NGC 4567/raw data/',
    'i': rfs.create_stack('../../Interacting Galaxies Project/NGC 4567/raw data/',
}
```

In [ ]:
```
# for a 4x1 grid, swap the rows and columns and figsize
fig, axs = plt.subplots(1, 4, figsize=(18, 9), facecolor='black')

for i, filter in enumerate(['g', 'ha', 'i', 'r']):
    data = raw_ngc4567[filter]
    # Raw data
    vis_vmin, vis_vmax = np.percentile(data, [1, 99.75])
    norm = vis.ImageNormalize(data, vmin=vis_vmin, vmax=vis_vmax, interval=vis.ZSca

    axs[i].imshow(data, cmap=custom, norm=norm, interpolation='gaussian')
    axs[i].set_title(f'Raw, Stacked Data ({filter})', weight='bold', color='white')
    axs[i].set_xlabel(f'Min: {vis_vmin:.3f}           Max: {vis_vmax:.3f}', color=

plt.setp(plt.gcf().get_axes(), xticks=[], yticks=[])
```

Out[ ]: []



Bias & Overscan

1. **Define the bias. Include both a description of what it looks like physically, why it occurs, and how we take it into account (how do we remove its effect on our data images?)**

   A bias is a 0-second blank photo that records electronic noise coming from the camera's sensor. Physically, it appears as a dark, uniform gray image but can reveal imperfections of the camera's sensor if you look closely at the patterns. Every pixel on the sensor has a small amount of noise associated with it, which is present regardless of whether the sensor is exposed to light. This noise can vary from pixel to pixel. Biases are taken into account by taking multiple and stacking together as a master bias, where it is then subtracted from the raw image data to be calibrated.

2. **Why would we use more than one bias frame?**

   The more bias frames you have the better you can remove noisy data from the pure photometric data. This is helpful for being more precise with your values so that the image data is best accurately portrayed.

3. **List the average value of each of your bias frames, as well as your final master bias.**

   See below.

4. **Define "overscan".**

   A region of the detector that is not exposed to light but still has the same readout process as the illuminated part of the detector. This region is used to measure and correct for variations in the electronic readout noise and bias level.

5. ~~**Skip 5 (ARCSAT has no overscan)**~~

In [ ]:
```
bias_files = []
print("Average Bias Values (for each file)")
for file in os.listdir('../../Interacting Galaxies Project/biases'):
    if re.search("Bias_", file):
        with fits.open('../../Interacting Galaxies Project/biases/' + file) as hdul
            data = hdul[0].data
            print(f"{np.mean(data):.3f}")
            bias_files.append(np.mean(data))
```

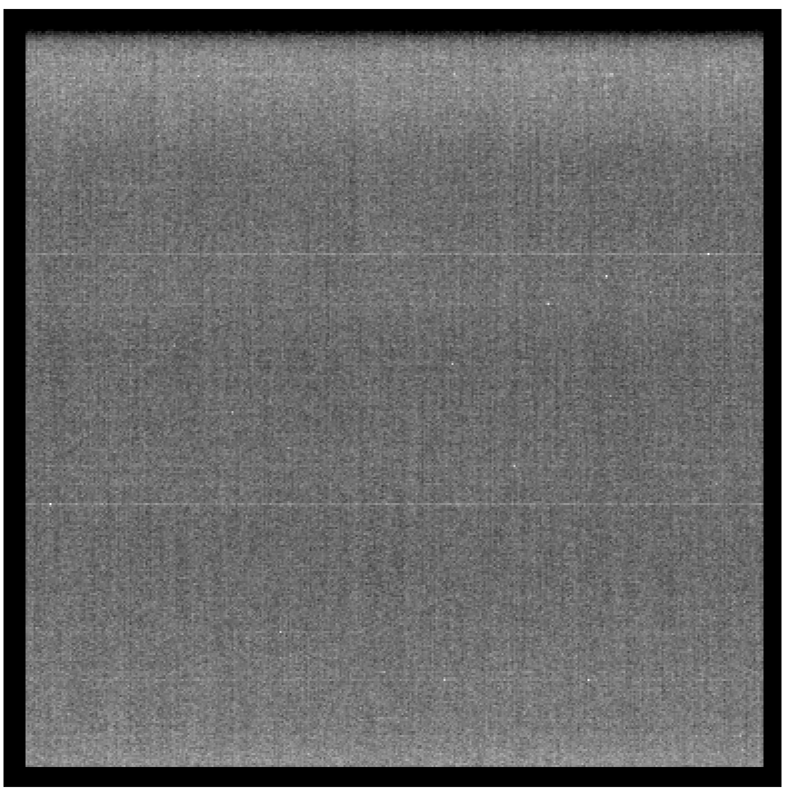
```
print(f"Standard Deviation: {np.std(bias_files):.3f}")

bias = fits.open('../../Interacting Galaxies Project/biases/master_bias.fits')[0].d
print(f"\nMaster Bias avg: {np.mean(bias):.3f}")
```

```
Average Bias Values (for each file)
1292.332
1292.311
1292.364
1292.303
1292.327
1292.282
1292.422
1292.429
1292.397
1292.422
1292.411
1292.365
Standard Deviation: 0.050

Master Bias avg: 1292.391
```

In [ ]:

```
plt.figure(facecolor='black')
b_vmin, b_vmax = np.percentile(bias, [1, 99.75])
dark_norm = vis.ImageNormalize(bias, vmin=b_vmin, vmax=b_vmax, interval=vis.ZScaleI
plt.imshow(bias, cmap='grey', norm=dark_norm, interpolation='hermite')
plt.xticks([])
plt.yticks([])
```

Out[ ]: ([], [])



Flat Fields

6. **Which filters do you have flat fields for? List the flat field file names for each filter that you need. Do you have more flats than you need? Confirm that your flat fields have the same binning as your data.**

   Filters: g, H-α, i, r.

   I'm not sure if there's a minimum amount of flats to have but in my opinion it doesn't hurt to have more! There are 5 flats for each filter.

   See below for file names and confirming binning.

7. **Why do we need flat fields for each filter we have used for observations?**

   Similar to biases, these help to remove noise in the raw photometric data but are helpful for specifically pinpointing noise more prevalent in the correlating filter. Biases are more general and pertain to the noise coming from the telescope itself, where flats are for noise coming from the atmosphere or other parts of the sky.

In [ ]:
```
# normalizing the flats before stacking
norm_flat = { # only flats need to be normalized
    'g': rfs.create_stack('../../Interacting Galaxies Project/flats/g flats/', keyw
    'ha': rfs.create_stack('../../Interacting Galaxies Project/flats/ha flats/', ke
    'i': rfs.create_stack('../../Interacting Galaxies Project/flats/i flats/', keyw
    'r': rfs.create_stack('../../Interacting Galaxies Project/flats/r flats/', keyw
}
'''my function uses median combine instead of sigma clipping'''
```

Out[ ]:
```
'my function uses median combine instead of sigma clipping'
```

In [ ]:
```
# printing out raw file names and normalized master values
for filter in ["g flats", "ha flats", "i flats", "r flats"]:
    relpath = "../../Interacting Galaxies Project/flats/"
    nm = [file for file in os.listdir(os.path.relpath(relpath + filter)) if file.st

    # printing out normalized average value
    print(f"\n[{filter}] ({len(nm)} files)".ljust(35) + (f"[Master Average] {np.mea

    filter_frames = []
    for file in nm:
        with fits.open(relpath + "/" + filter + "/" + file) as hdul:
            data = hdul[0].data
            filter_frames.append(np.mean(data))
            print(f"{file:<35} Average Value: {np.mean(data):>11.3f}")

    # the raw standard deviation is really high --> i.e. why flats are normalized b
    print(f"{'Standard Deviation:':>50} {np.std(filter_frames):>11.3f}")
```

```
[g flats] (5 files)              [Master Average]    7746.297
domeflat_g_001-2_050224.fits       Average Value:   18175.704
domeflat_g_001_050224.fits         Average Value:   18162.824
domeflat_g_001_050524.fits         Average Value:   18175.602
domeflat_g_002_050524.fits         Average Value:   19789.304
domeflat_g_003_050524.fits         Average Value:   19939.818
                               Standard Deviation:     830.865

[ha flats] (5 files)             [Master Average]    7676.117
domeflat_Ha_001-2_050224.fits      Average Value:   18246.795
domeflat_Ha_001_050224.fits        Average Value:   18232.767
domeflat_Ha_001_050524.fits        Average Value:   18264.449
domeflat_Ha_002_050524.fits        Average Value:   19799.890
domeflat_Ha_003_050524.fits        Average Value:   19992.406
                               Standard Deviation:     809.777

[i flats] (5 files)              [Master Average]    8138.241
domeflat_i_001-2_050224.fits       Average Value:   19155.744
domeflat_i_001_050224.fits         Average Value:   19175.730
domeflat_i_001_050524.fits         Average Value:   18668.241
domeflat_i_002_050524.fits         Average Value:   19740.962
domeflat_i_003_050524.fits         Average Value:   19919.097
                               Standard Deviation:     448.998

[r flats] (5 files)              [Master Average]    7972.609
domeflat_r_001-2_050224.fits       Average Value:   18941.049
domeflat_r_001_050224.fits         Average Value:   19066.175
domeflat_r_001_050524.fits         Average Value:   19074.010
domeflat_r_002_050524.fits         Average Value:   19832.463
domeflat_r_003_050524.fits         Average Value:   19891.546
                               Standard Deviation:     412.165
```

Darks

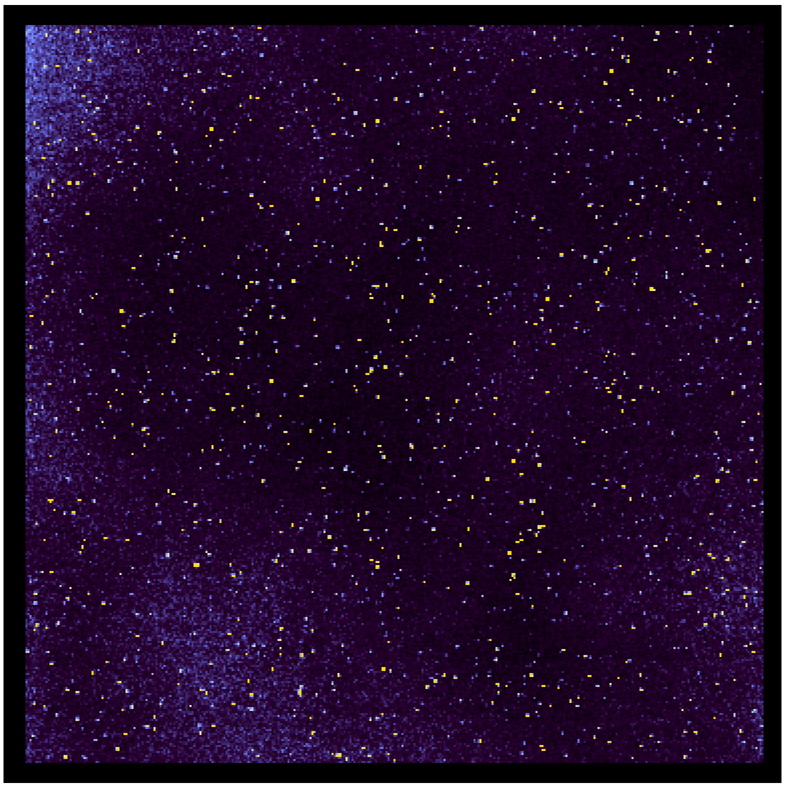
If your science images here are short, the dark current is extremely low and so we ignore it here. If you are taking longer images, or on instruments with lots of dark current, you can't neglect this term of the noise in reduction. Make sure to take into account darks

In [ ]:
```
dark_files = []
print("Average Dark Values (for each file)")
for file in os.listdir('../../Interacting Galaxies Project/darks'):
    if re.search("Dark_", file):
        with fits.open('../../Interacting Galaxies Project/darks/' + file) as hdul:
            data = hdul[0].data
            print(f"{np.mean(data):.3f}")
            dark_files.append(np.mean(data))
print(f"Standard Deviation: {np.std(dark_files):.3f}")

dark = fits.open('../../Interacting Galaxies Project/darks/master_dark.fits')[0].da
print(f"\nMaster Dark avg: {np.mean(bias):.3f}")
```

```
Average Dark Values (for each file)
1344.315
1344.136
1344.089
Standard Deviation: 0.097

Master Dark avg: 1292.391
```

In [ ]:
```
plt.figure(facecolor='black')
d_vmin, d_vmax = np.percentile(dark, [1, 99.75])
dark_norm = vis.ImageNormalize(dark, vmin=d_vmin, vmax=d_vmax, interval=vis.ZScaleI
plt.imshow(dark, cmap=custom, norm=dark_norm, interpolation='hermite')
plt.xticks([])
plt.yticks([])
```

Out[ ]:
```
([], [])
```



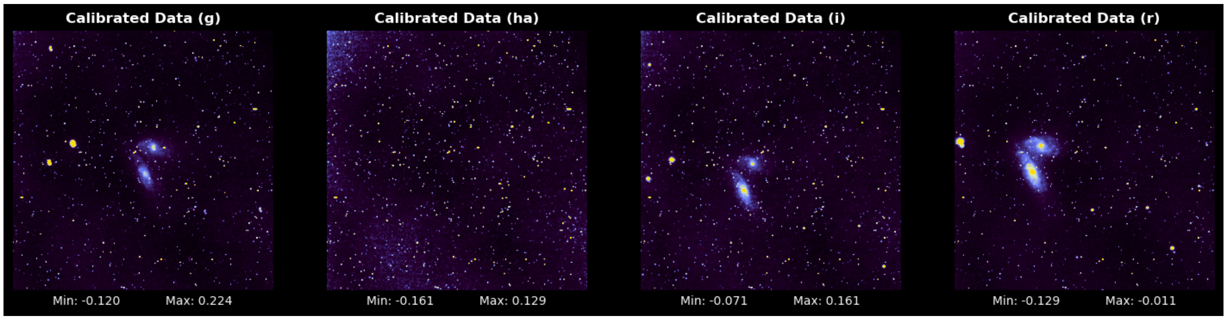
Saving/Visualizing Subtractions

In [ ]:
```
ngc_calbr = {
    'g': fits.open('../../Interacting Galaxies Project/NGC 4567/calibrated_g.fits')
    'ha': fits.open('../../Interacting Galaxies Project/NGC 4567/calibrated_ha.fits
    'i': fits.open('../../Interacting Galaxies Project/NGC 4567/calibrated_i.fits')
    'r': fits.open('../../Interacting Galaxies Project/NGC 4567/calibrated_r.fits')
}
```

In [ ]:
```
# for a 4x1 grid, swap the rows and columns and figsize
fig, axs = plt.subplots(1, 4, figsize=(18, 9), facecolor='black')

for i, filter in enumerate(['g', 'ha', 'i', 'r']):
    data = ngc_calbr[filter]
    vis_vmin, vis_vmax = np.percentile(data, [1, 99.75])
```

```
    norm = vis.ImageNormalize(data, vmin=vis_vmin, vmax=vis_vmax, interval=vis.ZSca

    axs[i].imshow(data, cmap=custom, norm=norm, interpolation='gaussian')
    axs[i].set_title(f'Calibrated Data ({filter})', weight='bold', color='white')
    axs[i].set_xlabel(f'Min: {vis_vmin:.3f}           Max: {vis_vmax:.3f}', color=

plt.setp(plt.gcf().get_axes(), xticks=[], yticks=[])
```

Out[ ]: []



Data Analysis (Photometry)

You now have images that are as close as we can get to what was emitted from our source. You can either use one of the broadband images from your project data or download the image from Canvas to use for this portion of the lab. It will provide an example for the different approaches to photometry you can then use for your project. (The next portion looks for point sources - if you use an image that has a diffuse image, like the nebulas, it might do some weird things for some of the options like DAOStarFinder. If you're having trouble navigating that - just grab the image on canvas.)

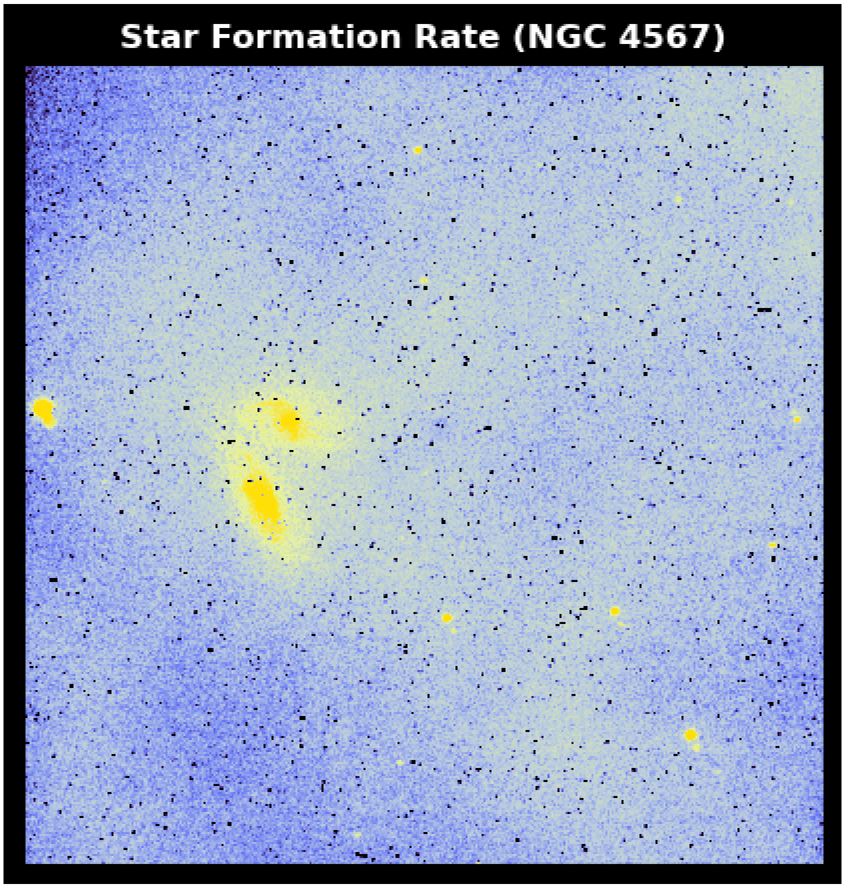
We are going to use the "photutils" python package to measure the signal.

For this section, I will be using the photometric data for star formation rates in objects NGC 4567 (R-band minus H-α band).

Here's an initial look of NGC 4567's star formation rate before background subtraction:

In [ ]:

```
sfr_ngc = ngc_calbr['r'] - ngc_calbr['ha']

plt.figure(facecolor='black', figsize=(4.5, 4.5))

vmin, vmax = np.percentile(sfr_ngc, [1, 99.75])
vnorm = vis.ImageNormalize(sfr_ngc, vmin=vmin, vmax=vmax, interval=vis.ZScaleInterv
plt.imshow(sfr_ngc, cmap=custom, norm=vnorm, interpolation='hermite')
plt.title('Star Formation Rate (NGC 4567)', weight='bold', color='white')

plt.setp(plt.gcf().get_axes(), xticks=[], yticks=[])
plt.tight_layout()
```



Subtracting The Sky

We are going to be subtracting the sky on the fly as we use aperture photometry to calculate the flux of the stars in our field. If your primary science object is diffuse or extended, just choose stars in your field that aren't as overexposed/saturated as your object.

8. What drawbacks could exist from this method of sky subtraction?

   There could be challenges with this method because it can be hard to distinguish pure data from noisy data. Also, images with galaxies may be harder to distinguish/identify compared to images with just stars - this was one issue I felt I came across during this portion

9. Why does sky subtraction matter?

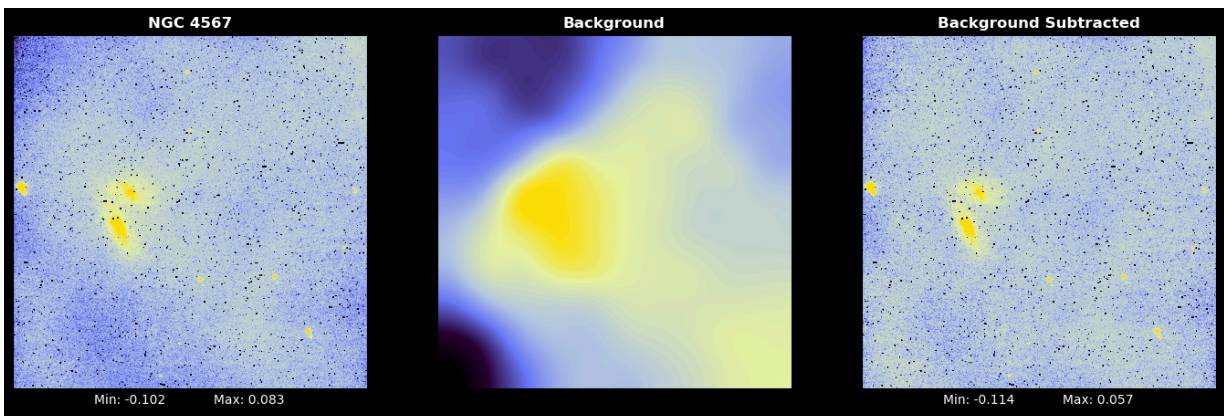
   This can help create a clearer picture and normalize values to account for light from the sky so that the numbers in a histogram are potentially less skewed, also it helps to reduce noisy effects on aperture photometry later

Photutils background estimation

In [ ]:
```
bkg = pb.Background2D(sfr_ngc, box_size=(100, 100), filter_size=(3, 3), sigma_clip=
                      mask=(sfr_ngc == 0), bkg_estimator=pb.MedianBackground())
# sigma_clip=SigmaClip(sigma=3),
# mask=(ngc_r_calbr == 0),
```

```
# A filter size of 1 (or (1, 1)) means no filtering.

bsub_sfr = sfr_ngc - bkg.background

# --------------------------------------------------------------------------------
plt.figure(figsize=(17, 9), facecolor='black')

plt.subplot(1, 3, 1)
vmin, vmax = np.percentile(sfr_ngc, [1, 99.75])
vnorm = vis.ImageNormalize(sfr_ngc, vmin=vmin, vmax=vmax, interval=vis.ZScaleInterv
plt.imshow(sfr_ngc, cmap=custom, norm=vnorm, interpolation='hermite')
plt.title('NGC 4567', weight='bold', color='white')
plt.xlabel(f'Min: {vmin:.3f}            Max: {vmax:.3f}', color='white')

plt.subplot(1, 3, 2)
plt.imshow(bkg.background, origin='lower', cmap=custom, interpolation='hermite')
plt.title('Background', weight='bold', color='white')

plt.subplot(1, 3, 3)
vmin, vmax = np.percentile(bsub_sfr, [1, 99.75])
vnorm = vis.ImageNormalize(bsub_sfr, vmin=vmin, vmax=vmax, interval=vis.ZScaleInter
plt.imshow(bsub_sfr, cmap=custom, norm=vnorm, interpolation='hermite')
plt.title('Background Subtracted', weight='bold', color='white')
plt.xlabel(f'Min: {vmin:.3f}            Max: {vmax:.3f}', color='white')

plt.setp(plt.gcf().get_axes(), xticks=[], yticks=[])
```

```
Out[ ]: []
```



Source Identification

Identify the sources to perform photometry on in your image. DAOStarFinder is one method provided within photutils. Plot the stars found on top of your image. Adjust your parameters to select a smaller set of stars (Between 10-100ish. Depending on your image). Replot. Remove any “bad” stars from the sources array by hand (hot spots, stars on the edge, stars on the overscan)

- I will be following this documentation from photutils

10. How many objects did you identify with DAOStarFinder? Look at your image. Do you think that’s a reasonable number? What did you change to adjust the number of stars

detected? How many stars did you end up with?

(see below) I started off with a bigger sized snippet of the whole image (around 500x500 pixels). then i saw that the number of stars was pretty good, though I was preferring less so I decreased the dimensions of the snippet to eventually what is below (335x335 px). This resulted in 11 stars total, to which I reduced it to 9 after reviewing bad sources. I also adjusted FWHM from 3 (in example I was following) to 4, where I was successfully able to get sources marked at the center of the galaxies instead of somewhere to the side.

In [ ]:

```
# selecting a portion of the region to get a set of stars
x_start, x_end = 160, 495
y_start, y_end = 360, 695
snip = bsub_sfr[y_start:y_end, x_start:x_end]
print(f"Shape: {snip.shape} pixels")

snip_mean, snip_median, snip_std = sigma_clipped_stats(snip, sigma=3)
print(f"Mean: {snip_mean:.3f}", f"\nMedian: {snip_median:.3f}", f"\nStd: {snip_std:
```

```
Shape: (335, 335) pixels
Mean: 0.001
Median: 0.004
Std: 0.021
```

In [ ]:

```
daofind = DAOStarFinder(fwhm=4, threshold=5.*snip_std) # fwhm=3 in example
                    #   fwhm: The full-width half-maximum (FWHM) of the major axis
sources = daofind(snip - snip_median)

for col in sources.colnames:
    if col not in ('id', 'xcentroid', 'ycentroid'):
        sources[col].format = '%.2f'  # for consistent table output
sources.pprint(max_width=100, max_lines=7)

positions = np.transpose((sources['xcentroid'], sources['ycentroid']))
apertures = CircularAperture(positions, r=10)
annuli = CircularAnnulus(positions, r_in=10, r_out=20)
```

```
 id      xcentroid          ycentroid      sharpness  roundness1 roundness2  npix sky
peak flux  mag
--- ------------------ ----------------- --------- ---------- ---------- ----- ----
---- ---- -----
  1 142.22972184952087 59.70851154914619      0.37       0.00       0.01 25.00 0.00
0.02 4.57 -1.65
  2 137.7950554098001 64.27152196199076      0.38      -0.02       0.07 25.00 0.00
0.02 4.57 -1.65
...                ...               ...       ...        ...        ...   ...  ...
...  ...   ...
 10  92.31134856880864 308.0187079225787      0.35       0.41      -0.56 25.00 0.00
0.01 1.00 -0.00
 11  85.87219500422762 328.5349151124505      0.28      -0.85      -0.41 25.00 0.00
0.00 1.24 -0.23
Length = 11 rows
```

Aperture Photometry

photutils aperture photometry

Photutils provides many different kinds of apertures. We will use two here, but other cases (such as galaxies) might require apertures that aren't round. First, use the "circular aperture" Choose your method and execute your aperture photometry. Review the output. Then plot your apertures on top of your image.

Do the same thing, but instead of using just a circular aperture, we're going to use annuli. This means you can specify a region for the star, and a nearby region for the sky. This allows us to use the sky that is close to the individual objects to subtract.

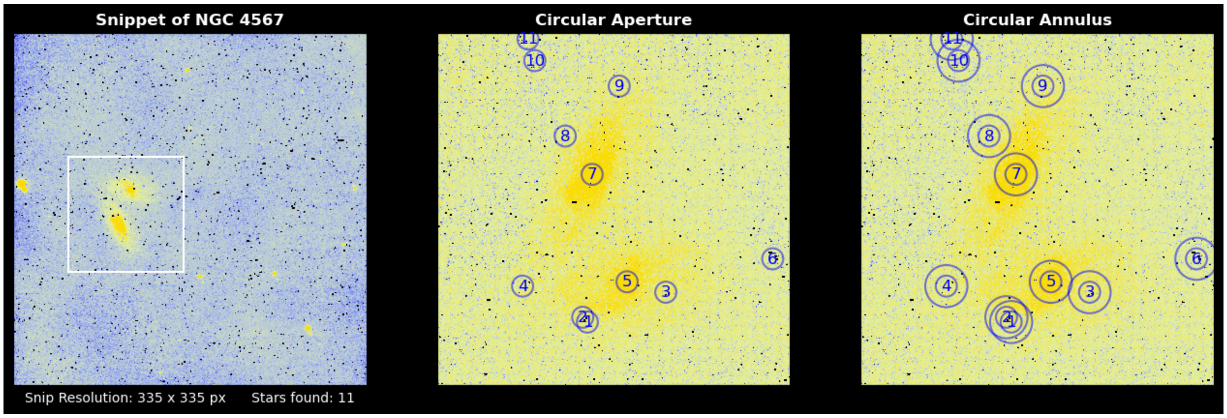
Pick your object aperture and note it in your Q12 answers. Pick your sky annulus minimum and maximum note it in your Q12 answers.

Now, run as above, except using the "CircularAnnulus" command. Use these two measurements of flux to remove the background from the originally measured flux, and print those results.

visualizing sources from daostarfinder

In [ ]:
```
# viewing selection & apertures
with warnings.catch_warnings(): # booo warnings
    warnings.simplefilter('ignore')

    fig = plt.figure(figsize=(16, 9), facecolor='black')

    # plotting sfr image and marking the snippet ---------------------------------
    plt.subplot(1, 3, 1)
    plt.title(f'Snippet of NGC 4567', weight='bold', color='white')
    plt.xlabel(f'Snip Resolution: {snip.shape[0]} x {snip.shape[1]} px      Stars f

    vmin, vmax = np.percentile(bsub_sfr, [1, 99.75])
    vnorm = vis.ImageNormalize(bsub_sfr, vmin=vmin, vmax=vmax, interval=vis.ZScaleI
    plt.imshow(bsub_sfr, cmap=custom, norm=vnorm, interpolation='hermite')

    snip_width, snip_height = x_end - x_start, y_end - y_start
    extent = [x_start, x_start + snip_width, y_start, y_start + snip_height]
    rect = Rectangle((x_start, y_start), snip_width, snip_height, edgecolor='white'
    plt.gca().add_patch(rect)


    # plotting apertures/annului --------------------------------------------------
    norm = vis.ImageNormalize(snip, vmin=vmin, vmax=vmax, interval=vis.ZScaleInterv

    plt.subplot(1, 3, 2)
    plt.imshow(snip, cmap=custom, origin='lower', norm=norm, interpolation='hermite
    apertures.plot(color='blue', lw=1.5, alpha=0.5)
    for i, (x, y) in enumerate(zip(sources['xcentroid'], sources['ycentroid'])):
        plt.text(x, y, f"{sources['id'][i]}", color='blue', fontsize=12, ha='center
    plt.title(f'Circular Aperture', weight='bold', color='white')

    plt.subplot(1, 3, 3)
```

```
    plt.imshow(snip, cmap=custom, origin='lower', norm=norm, interpolation='hermite
    annuli.plot(color='blue', lw=1.5, alpha=0.5)
    for i, (x, y) in enumerate(zip(sources['xcentroid'], sources['ycentroid'])):
        plt.text(x, y, f"{sources['id'][i]}", color='blue', fontsize=12, ha='center
    plt.title(f'Circular Annulus', weight='bold', color='white')

    plt.setp(plt.gcf().get_axes(), xticks=[], yticks=[])
```



manually removing bad stars **by id number**

In [ ]:
```
sources = sources[~np.isin(sources['id'], [11, 6, 1])]
sources.pprint(max_width=100)

# redeclaring everything again to make sure everything's on the same page
positions = np.transpose((sources['xcentroid'], sources['ycentroid']))
apertures = CircularAperture(positions, r=10)
annuli = CircularAnnulus(positions, r_in=10, r_out=20)
```

```
 id      xcentroid          ycentroid      sharpness roundness1 roundness2  npix sky
peak flux  mag
--- ------------------ ------------------ --------- ---------- ---------- ----- ----
---- ---- -----
  2  137.7950554098001  64.27152196199076      0.38      -0.02       0.07 25.00 0.00
0.02 4.57 -1.65
  3 216.64888139253873  88.05205732393533      0.31       0.08       0.52 25.00 0.00
0.02 2.20 -0.85
  4  80.75190381392842  93.83833500578672      0.28       0.46      -0.72 25.00 0.00
0.01 1.52 -0.45
  5 179.87490323617945  97.95699856829381      0.42      -0.47      -0.09 25.00 0.00
0.28 1.32 -0.30
  7  146.7637516564575  199.9580568595561      0.42       0.90       0.10 25.00 0.00
0.27 1.19 -0.19
  8 121.23838094315283  236.3707648232261      0.31      -0.50       0.89 25.00 0.00
0.02 1.48 -0.42
  9 172.39538683735128 283.82514580882787      0.25       0.83      -0.65 25.00 0.00
0.01 1.05 -0.06
 10  92.31134856880864  308.0187079225787      0.35       0.41      -0.56 25.00 0.00
0.01 1.00 -0.00
```

continuing to photometry and getting magnitudes of the galaxies

In [ ]:
```
phot_table = aperture_photometry(bsub_sfr, apertures)
phot_table['annulus_sum'] = aperture_photometry(bsub_sfr, annuli)['aperture_sum']
```

```
aperture_areas = pu.aperture.PixelAperture.area_overlap(apertures, bsub_sfr)
annuli_areas = pu.aperture.PixelAperture.area_overlap(annuli, bsub_sfr)

aperstats = ApertureStats(bsub_sfr, apertures)
annustats = ApertureStats(bsub_sfr, annuli, sigma_clip=SigmaClip(sigma=3.0, maxiter

# total background within the circular aperture and annulus
phot_table['aper_total_bkg'] = aperstats.median * aperture_areas
phot_table['annu_total_bkg'] = annustats.median * annuli_areas

# background-subtracted photometry
phot_table['aperture_sum_bkgsub'] = phot_table['aperture_sum'] - phot_table['aper_t
phot_table['annuli_sum_bkgsub'] = phot_table['annulus_sum'] - phot_table['annu_tota

print(f" Mean Apertures: {aperstats.median}\n \n", f"Mean Annuli: {annustats.median

for col in phot_table.colnames:
    if col not in ('id'):
        phot_table[col].format = '%.2f'  # for consistent table output

phot_table['id'] = sources['id']
phot_table.pprint(max_lines=13, max_width=200)
```

```
 Mean Apertures: [-4.26777050e-03 -7.07681773e-05 -6.36765528e-03 -5.58873283e-04
  1.99182868e-03 -5.06306099e-04  4.88107854e-03 -1.25256782e-03]

 Mean Annuli: [-0.0051085  -0.00073529 -0.00666895 -0.00035985  0.00266695 -0.000689
62
  0.00587416 -0.00096243]


 id      xcenter            ycenter       aperture_sum annulus_sum aper_total_bkg an
nu_total_bkg aperture_sum_bkgsub annuli_sum_bkgsub
           pix                pix
--- ------------------ ------------------ ------------ ----------- -------------- --
------------ ------------------- -----------------
  2  137.7950554098001  64.27152196199076        -4.35      -12.95          -1.34
-4.81               -3.01              -8.14
  3 216.64888139253873  88.05205732393533        -2.65      -15.26          -0.02
-0.69               -2.62             -14.57
  4  80.75190381392842  93.83833500578672        -6.37      -19.30          -2.00
-6.29               -4.37             -13.01
  5 179.87490323617945  97.95699856829381        -2.84      -14.54          -0.18
-0.34               -2.66             -14.20
  7  146.7637516564575  199.9580568595561        -3.73       -9.90           0.63
2.51              -4.35             -12.41
  8 121.23838094315283  236.3707648232261        -2.33      -10.46          -0.16
-0.65               -2.17              -9.81
  9 172.39538683735128 283.82514580882787        -1.47       -4.60           1.53
5.54              -3.01             -10.13
 10  92.31134856880864  308.0187079225787        -2.81       -9.13          -0.39
-0.91               -2.42              -8.22
```

11. Are you happy with your apertures? Why or why not? Save the list of sources for comparison.

    I like the annulus magnitudes better than the apertures, because they record greater magnitudes and exclude the background inside the outer annulus which makes the measurement more accurate. With the apertures, they are easily skewed by noise around the sources, so it makes it easy to decide that using annuli is the better way to go.

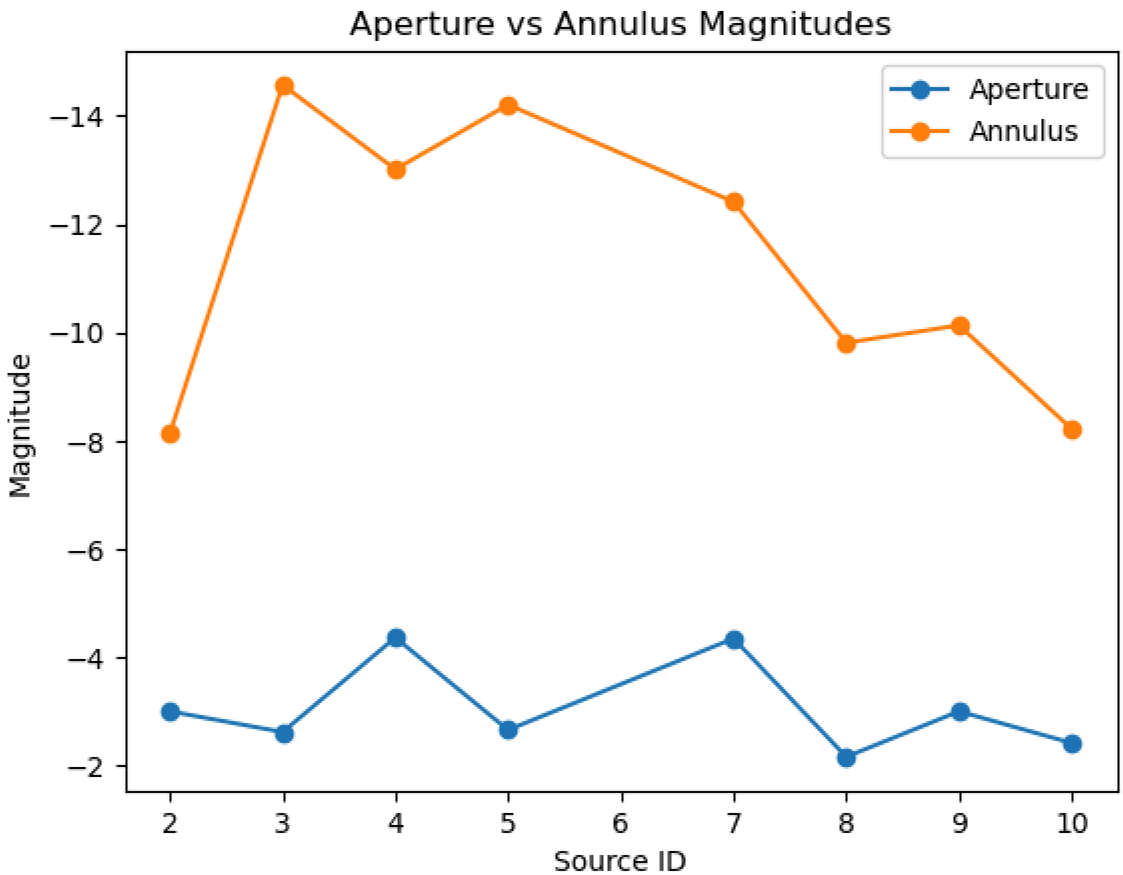
In [ ]:
```
for source_id in [5, 7]:
    source_row = sources[sources['id'] == source_id]
    x, y = np.round(source_row['xcentroid'][0], 2), np.round(source_row['ycentroid'
    phot_row = phot_table[(np.round(phot_table['xcenter'].value, 2) == x) & (np.rou

    print(f"[ID {source_id}]")
    print(f"  Aperture Magnitude: {phot_row['aperture_sum_bkgsub'][0]:.2f}")
    print(f"  Annulus Magnitude: {phot_row['annuli_sum_bkgsub'][0]:.2f}")
    print('\n')
```

```
[ID 5]
  Aperture Magnitude: -2.66
  Annulus Magnitude: -14.20


[ID 7]
  Aperture Magnitude: -4.35
  Annulus Magnitude: -12.41
```

12. Make a plot comparing the measurements of the two different methods

In [ ]:
```
plt.plot(phot_table['id'], phot_table['aperture_sum_bkgsub'], marker='o', linestyle
plt.plot(phot_table['id'], phot_table['annuli_sum_bkgsub'], marker='o', linestyle='
plt.xlabel('Source ID')
plt.ylabel('Magnitude')
plt.ylim(plt.ylim()[::-1]) # invert y-axis to show brighter objects at the top
plt.title('Aperture vs Annulus Magnitudes')
plt.legend()
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x18ea4037d10>
```



Calculating Flux

Use PSF fitting to calculate the flux of the same set of sources you found with DAOstarfinder. Make sure to sky subtract

13. Make a plot comparing the PSF fitting output to the aperture output. Are they the same? Why or why not?

    The PSF fittings are a little stranger - for the most part, they're resulting in really dim magnitudes (lower than circular perture), except for source #2, which sky rockets to ~120 in magnitude! This makes the PSF fittings feel really inconsistent, as the circular apertures and annuli were at least consistent with each other. See below for plot

14. What do you need to do differently for your science image? (For example - are you studying one point source, many point sources, or a diffuse object? What does that change?)

    I could definitely remove source #3, though I would be hesitatnt to since it's relatively close to the galaxy's center, so I'd want to keep it there to monitor magnitudes of other parts of the galaxy. I am also using the multiple point source method, which could also be a reason why the numbers look so weird. Between that and the fact that I'm looking at a galaxy could be the two main factors as to why the PSF fitting method looks strange in this situation, and one way to change that would be to look at non-galaxy objects like stars.

15. Did you prefer PSF fitting or aperture photometry? Think of one situation that you think would be best for each method.

    I prefer the aperture photometry (specifically annulus) over PSF fitting. I liked annulus the most because it works well for my situation, given that I'm using photometry on galaxies, so the annulus allows me to extend the radius outward to account for other parts of the galaxy. Circular aperture, although an option for this situation, seems better for singular objects like stars rather than galaxies, since they're usually expected to be a circular "dot" rather than an ellipse or other. PSF fitting would be suited best for images that are distorted or blurry, allowing to identify positions and brightness of stars.

photutils psf photometry

In [ ]:
```
psf_model = IntegratedGaussianPRF(flux=1, sigma=2.7 / 2.35)
fit_shape = (5, 5)
psfphot = PSFPhotometry(psf_model, fit_shape, finder=daofind,
                        aperture_radius=4)
phot = psfphot(snip)
phot = phot[~np.isin(phot['id'], [11, 6, 1])]
phot.pprint(max_lines=13, max_width=200)
```

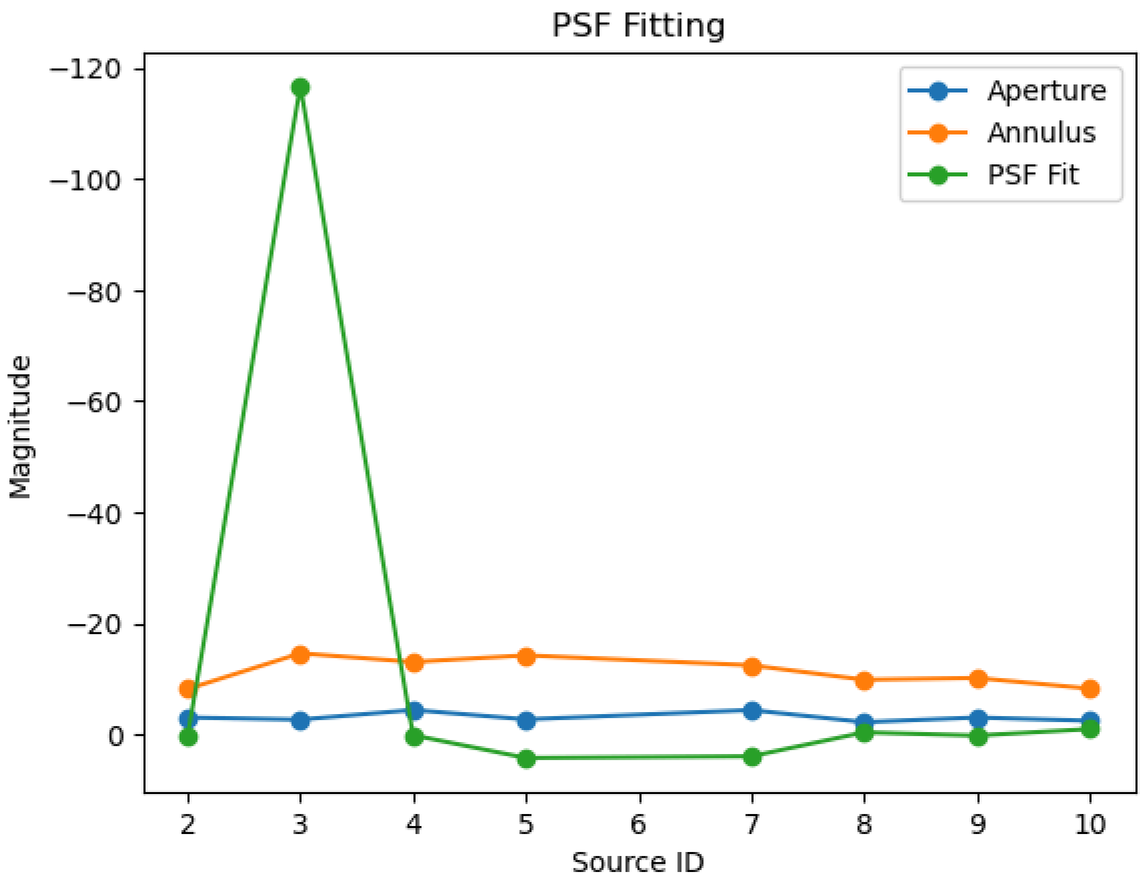
```
 id group_id local_bkg       x_init             y_init             flux_init
x_fit         ...       flux_err      npixfit group_size          qfit
cfit          flags
--- -------- --------- ------------------ ------------------ -------------------- --
---------------- ... ------------------- ------- ---------- --------------------- --
-------------------- -----
  2        2       0.0  137.7950554098001  64.27152196199076  -3.3827293595554395 13
9.33326836376173 ... 0.10404463050116786      25          1     2.0599698978206478
0.03375971032823363      0
  3        3       0.0 216.64888139253873  88.05205732393533  -1.1397955928753818 22
1.05489762253572 ...                 nan      25          1 -0.007518105406363414 -
7.895045480036578e-05     12
  4        4       0.0  80.75190381392842  93.83833500578672  -0.5590965044686451  7
9.89501236231463 ... 0.09042469875197508      25          1     2.052926084534316
0.04329851824301431      0
  5        5       0.0 179.87490323617945  97.95699856829381    9.965032789766978 18
0.02994899011793 ...  0.5358248866547475      25          1    0.6196283950302639
-0.04060533234392836      0
  7        7       0.0  146.7637516564575 199.95805685955608    10.04929778699413  1
46.9534510789961 ...  0.5167630289599099      25          1    0.6316011391907531
-0.04320112223897045      0
  8        8       0.0 121.23838094315283  236.3707648232261  -0.6306684510036639 12
2.53680902865253 ... 0.17626951579290223      25          1   -1.9019013965686393
-0.0711843978930803      4
  9        9       0.0 172.39538683735128 283.82514580882787 -0.11977906097851233 17
4.71761619417575 ... 0.46842331334059084      25          1    1.7506865119143311
-0.11914988883125104      0
 10       10       0.0  92.31134856880864  308.0187079225787   -0.944930665649742
91.2808157110611 ...    2.061435541086818      25          1   -0.7182636025692325  -
0.020216219693651505      4
```

```
WARNING: One or more fit(s) may not have converged. Please check the "flags" column
in the output table. [photutils.psf.photometry]
```

In [ ]:

```python
plt.plot(phot_table['id'], phot_table['aperture_sum_bkgsub'], marker='o', linestyle
plt.plot(phot_table['id'], phot_table['annuli_sum_bkgsub'], marker='o', linestyle='
plt.plot(phot['id'], phot['flux_fit'], marker='o', linestyle='-', lw=1.5, label='PS
plt.xlabel('Source ID')
plt.ylabel('Magnitude')
plt.ylim(plt.ylim()[::-1]) # invert y-axis to show brighter objects at the top
plt.title('PSF Fitting')
plt.legend()
```

Out[ ]: <matplotlib.legend.Legend at 0x18ea9772490>



In [ ]: