HW 5

-- Serat Saad

Feedback:

Your notebook has a very nice combination of markdown narrative and code commenting. These make it really easy to follow your narrative and understand your process. You had a big error at the very beginning of your notebook and it propagated through into everything: your V-band filters were mislabeled and you actually grabbed some H-alpha, some U, and some B. You could have realized it looking at how different the flats were and how some were saturated, but regardless it probably made it difficult to figure out how best to combine the flats since you didn't actually get any stars in those frames. I took off points for this issue once at the beginning, but be aware that all your saved files probably need to be redone before HW6. According to git, your submission was 1 day late. Otherwise, this was a very nice submission!

Here's a revision for HW 5.
I revised HW 4R again where I made a mistake while indexing different files for different filters and saving the filter name in the header. I am submitting that revision with this revision. So, I think the filter names should be okay now and the following few cells will properly filter out the V filters.

I also went through the fifth step where I made a mistake while calculating the gain and readnoise. This is the reason why there a lot of variations in gain and read noise. In this revision, I used only flat_fielded flats and calculates gain and read noise again.

This jupyter file size became very large, so I didn't load the images and the cells so that it doesn't get large to upload into github. I have attached a pdf with this submission so that you can go through the outputs.

This is also the reason why I couldn't submit before deadline last time as it wasn't uploading. I tried to use git lfs, but I think the file size is too large this time due to all the images that git lfs is not working.

The following two cells will import all the necessary ibraries and directories necessary for the homework

In [1]:
```
# import block
import ccdproc as ccdp
import numpy as np
from astropy.io import fits
from matplotlib import pyplot as plt
from matplotlib import rc
%matplotlib inline
from astropy.visualization import hist
from ccdproc import ImageFileCollection
import ccdproc as ccdp
from astropy.modeling import fitting
from astropy.modeling.models import Polynomial1D,Chebyshev1D,Legendre1D,Hermit
from astropy.nddata import CCDData
from datetime import datetime
from astropy.stats import sigma_clip
from astropy.nddata import block_reduce, Cutout2D
from astropy.modeling import models
import random
import math


import seaborn as sns
from astropy.io import fits
import matplotlib.pyplot as plt
import numpy as np
import ccdproc
from astropy.nddata import CCDData
from ccdproc import subtract_overscan, trim_image, combine
from astropy.visualization import hist
from astropy.modeling import models

from convenience_function import show_image
```

In [2]:
```
data_dir = 'Imaging/'            # raw data directory
reduced_dir = 'Imaging_reduced/' # reduced working directory
```

We can start working with the skyflat files to see how different combining methods work for them

In the following cell we get all the v filter skyflat images and their files

Comment: Something has gone wrong, because a019 is not a V-band flat (the log says it is H-alpha). You're going to be combining H-alpha, U, and B band flats using these files.

It should be correct this time.

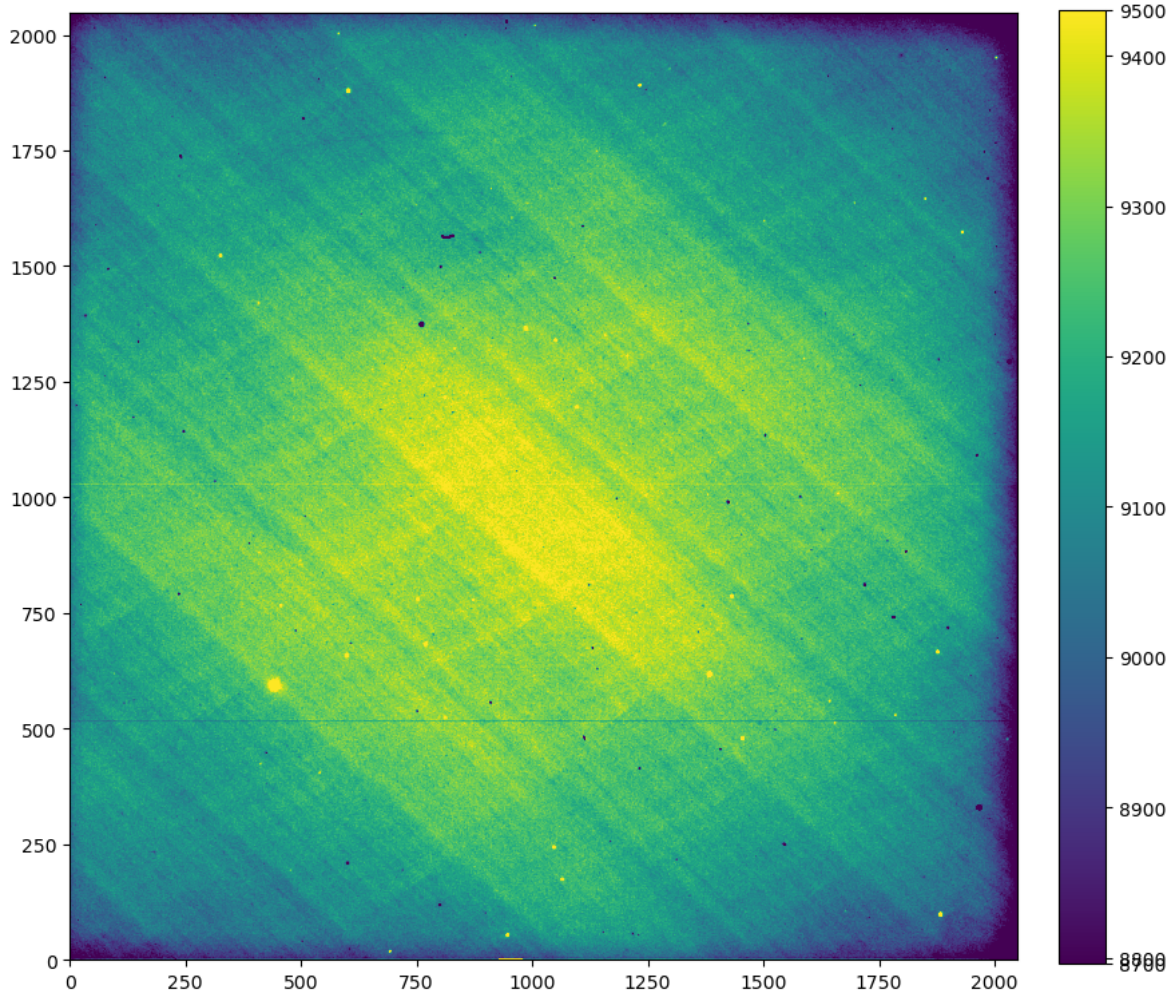
In [3]:
```
imgs = ccdp.ImageFileCollection(reduced_dir,glob_include='*otz.fits')
v_flat_files = imgs.files_filtered(imagetyp='skyflat',filter='V',include_path=
v_flat_imgs  = imgs.filter(imagetyp='skyflat',filter='V')
v_flat_imgs.summary
v_flat_imgs.summary
```

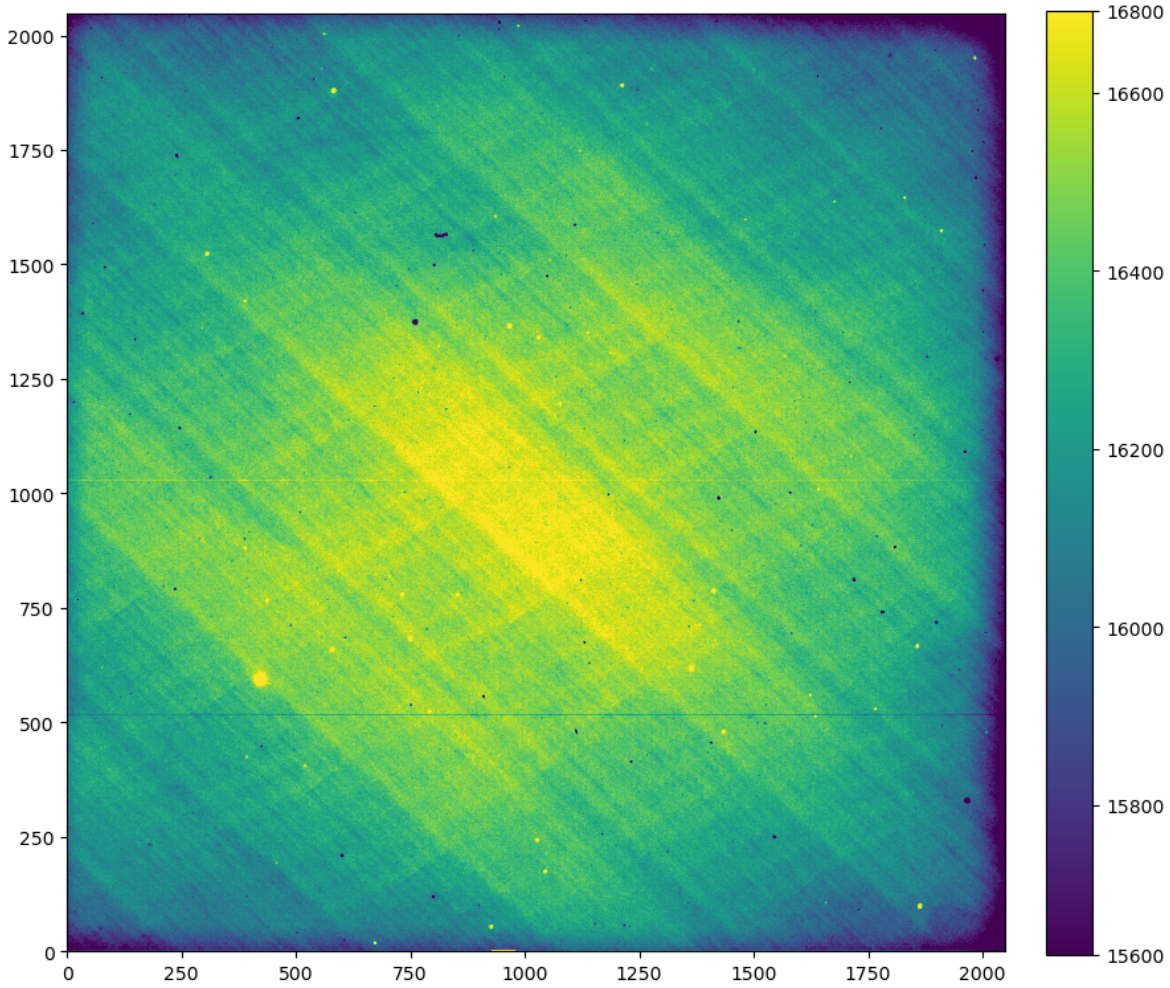
Out[3]: *Table masked=True length=7*

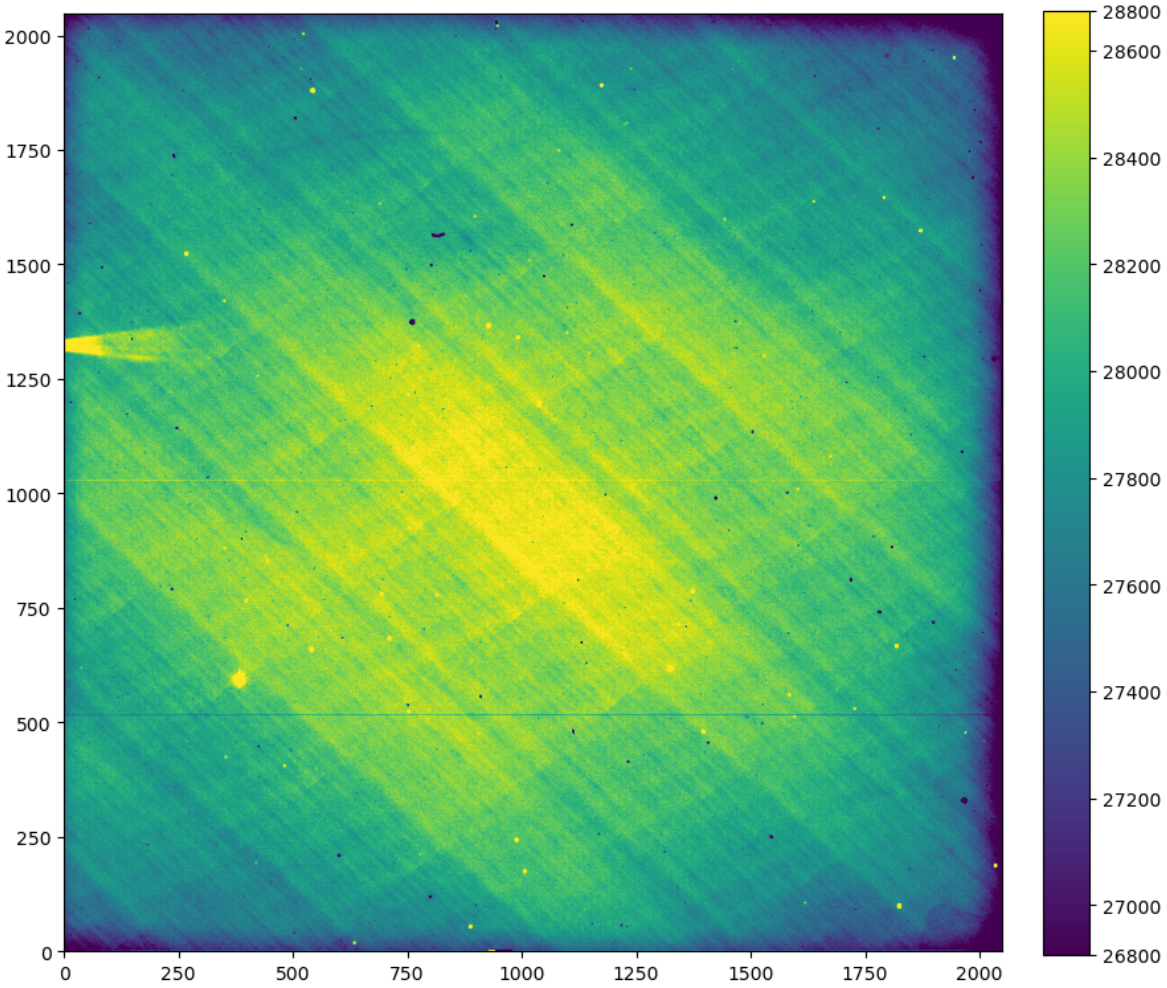
file	simple	bitpix	naxis	naxis1	naxis2	extend	origin	date
str28	bool	int32	int32	int32	int32	bool	str37	str19
Imaging_reduced/a043otz.fits	True	-64	2	2048	2048	True	NOAO-IRAF FITS Image Kernel July 2003	2009-09-15T15:55:55

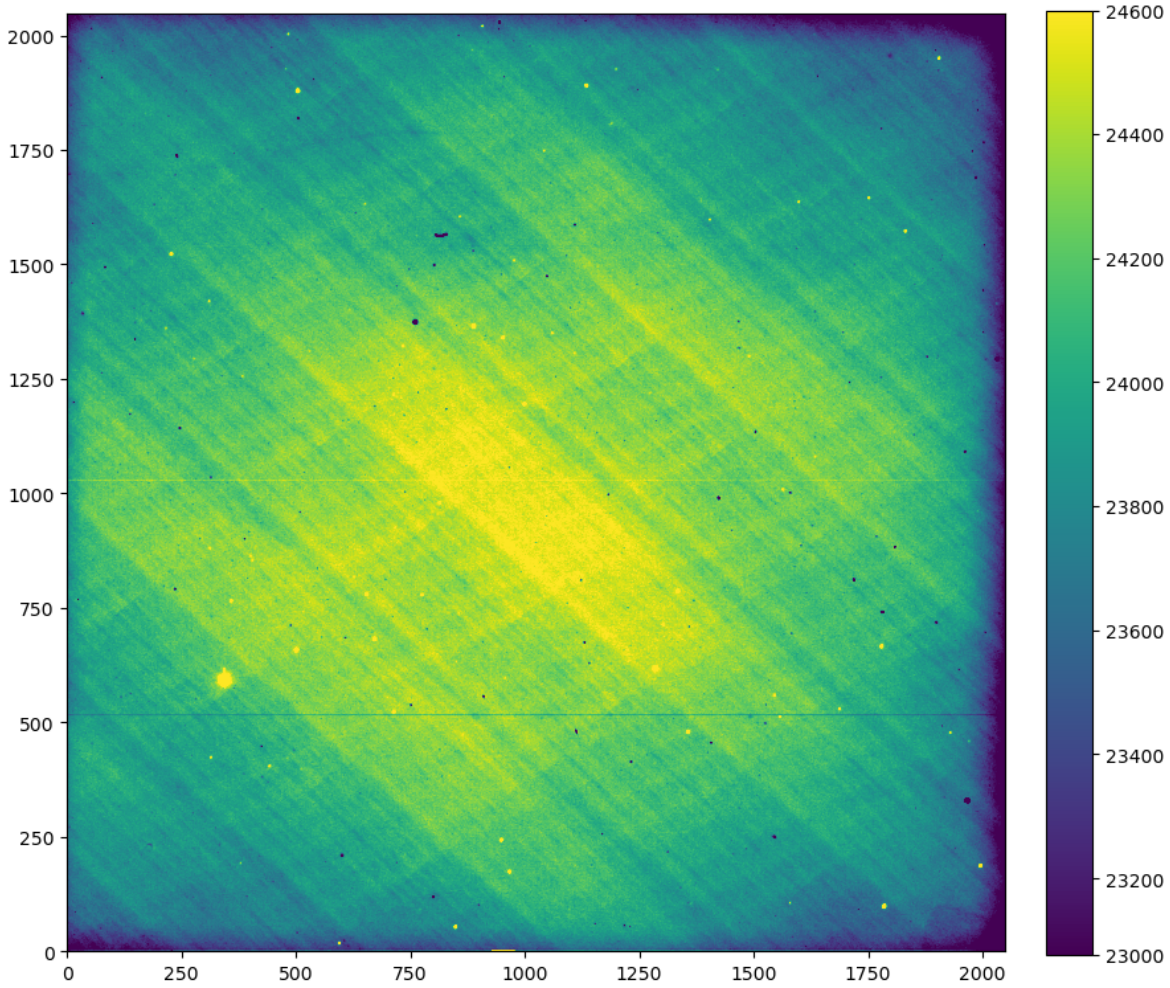
The following cell will plot the v filter flat images

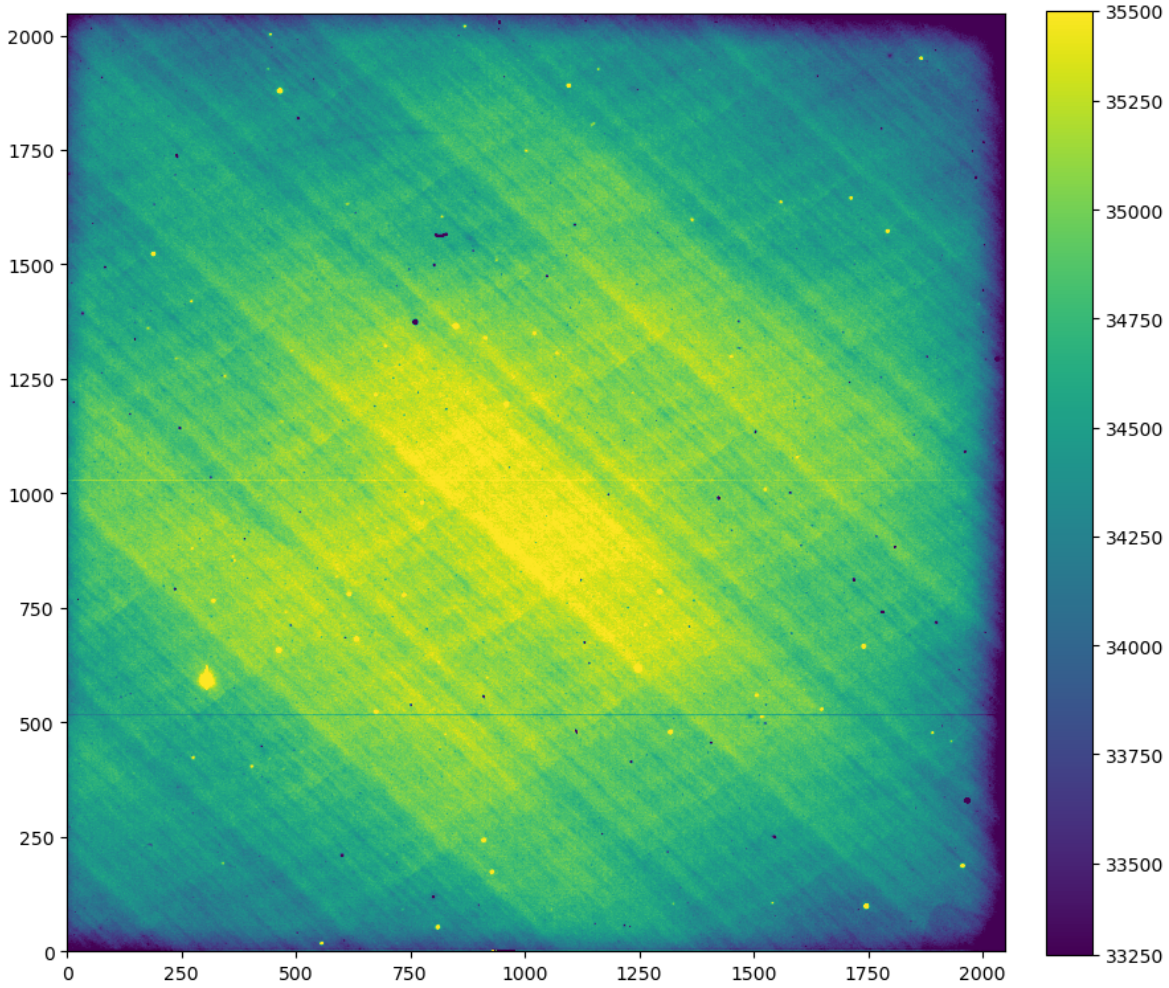
In [4]:
```
all_data = []

for i in range(len(v_flat_files)):
    with fits.open(v_flat_files[i]) as hdul:
        data = hdul[0].data
    show_image(data)
```

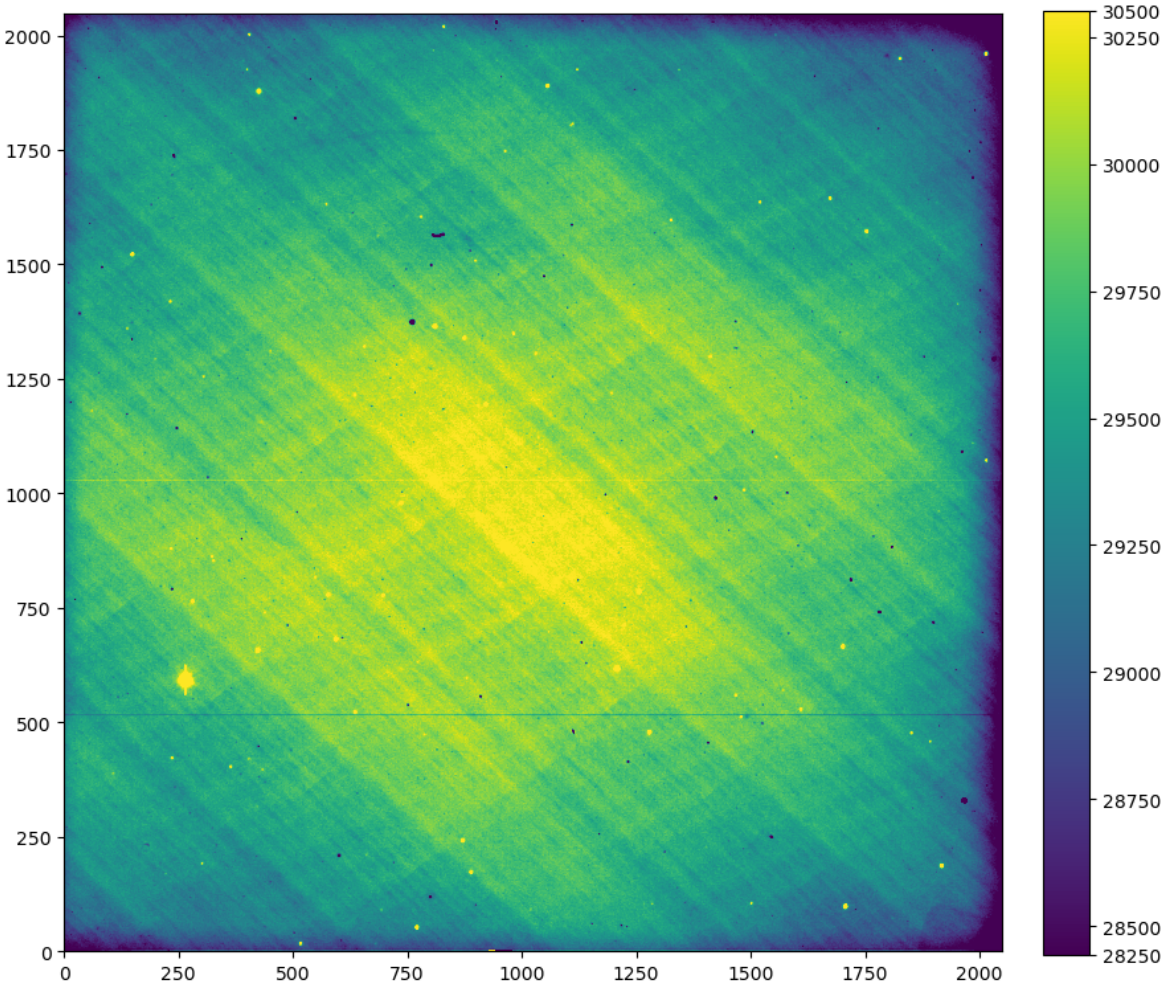


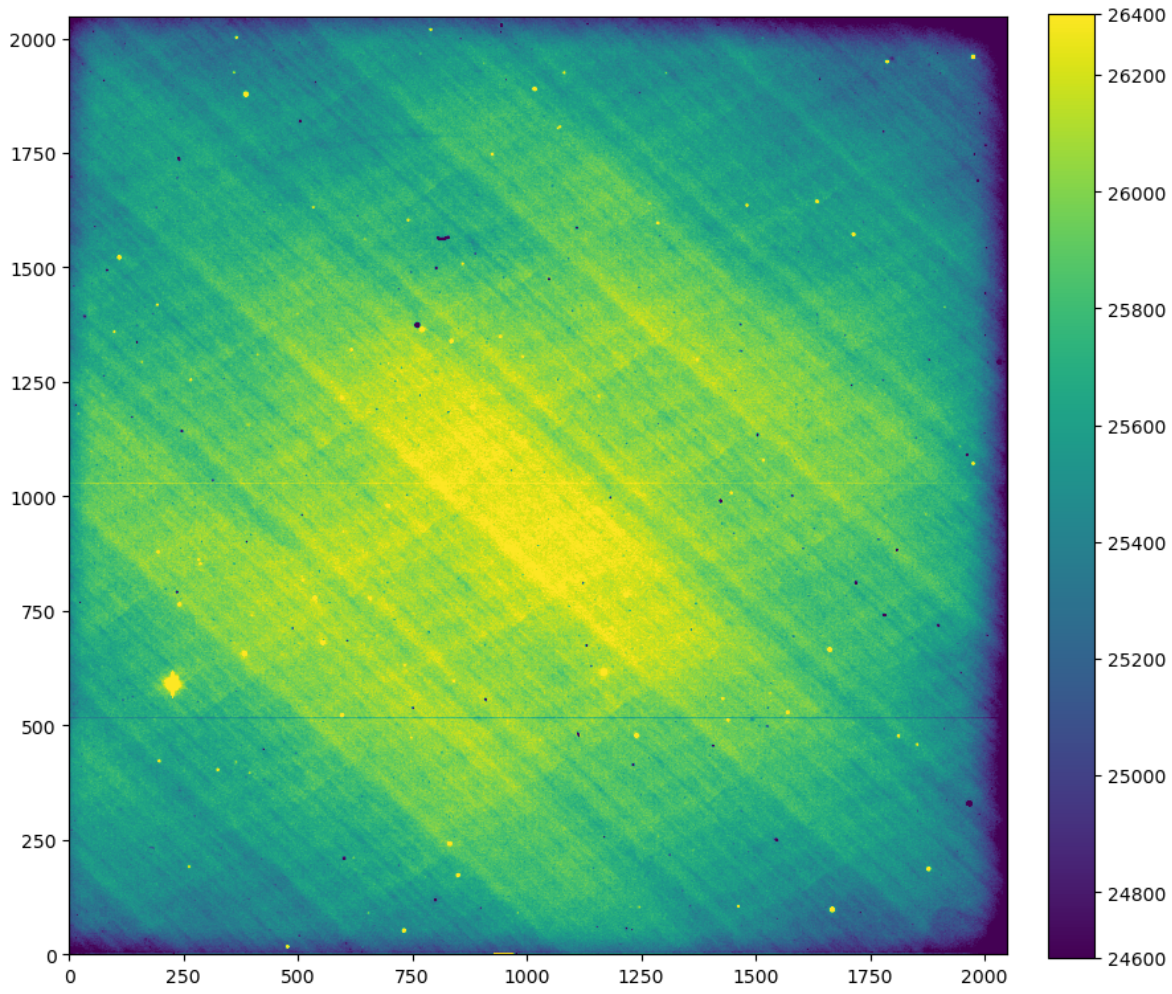

2000
1750
1500
1250
1000
750
500
250
0
0
250
500
750
1000
1250
1500
1750
2000
16800
16600
16400
16200
16000
15800
15600


2000
1750
1500
1250
1000
750
500
250
0
0
250
500
750
1000
1250
1500
1750
2000
28800
28600
28400
28200
28000
27800
27600
27400
27200
27000
26800


2000
1750
1500
1250
1000
750
500
250
0
0
250
500
750
1000
1250
1500
1750
2000
24600
24400
24200
24000
23800
23600
23400
23200
23000


0
250
500
750
1000
1250
1500
1750
2000
35500
35250
35000
34750
34500
34250
34000
33750
33500
33250


2000
1750
1500
1250
1000
750
500
250
0
0
250
500
750
1000
1250
1500
1750
2000
30500
30250
30000
29750
29500
29250
29000
28750
28500
28250



In the following cell we can find rms values for different methods so that we can compare each of the methods with one another.

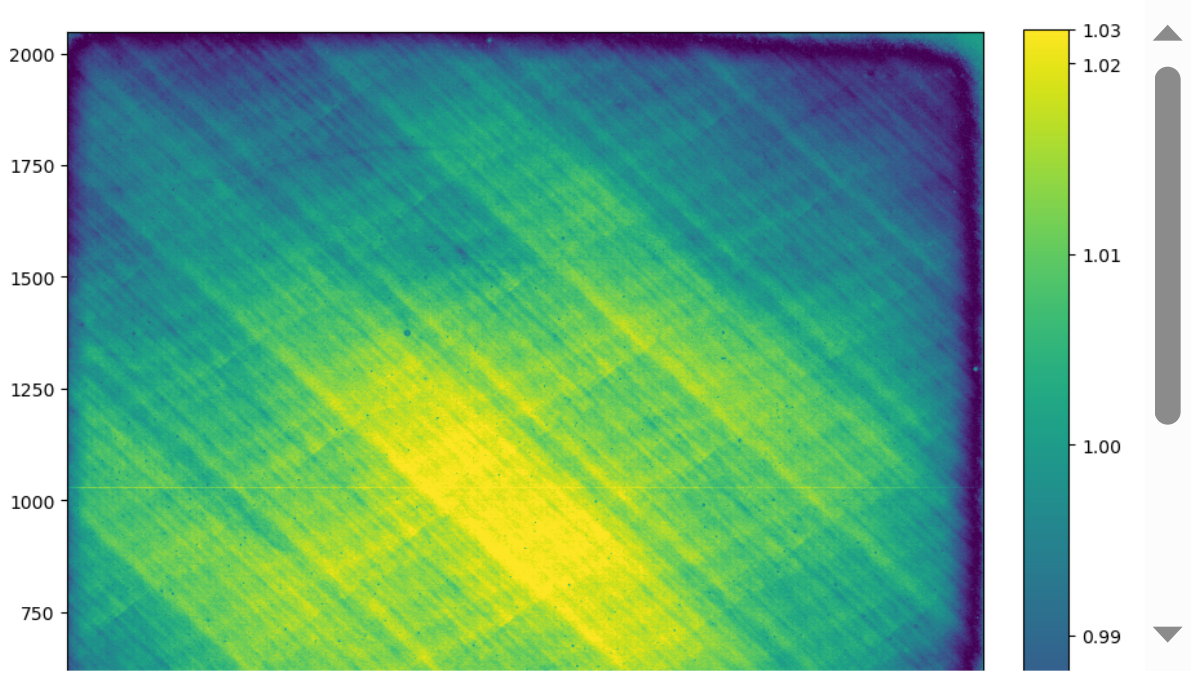
In [5]:
```
def extract_data(file, sigma_clip):
    # This function extracts data from a file. It takes two parameters:
    # 1. File of the source - file
    # 2. Whether we are following the sigma clip method or not - sigma_clip
    with fits.open(file) as hdul:
        data = hdul[0].data
        data /= np.mean(data)
        if sigma_clip==True:
            sigma, mean = np.std(data), np.mean(data)
            data = np.where((data > 2*sigma + mean) | (data < mean - 2*sigma),
        data = np.where(np.isnan(data), np.nanmean(data), data)
        return data, np.mean(data)

# Processing all files and compute the master flats for sigma clipping
all_data_normalized, _ = zip(*(extract_data(file, True) for file in v_flat_fil
master_flat_sigma_clip = np.nanmean(all_data_normalized, axis=0)


# Processing all files and compute the master flats for other cases
all_data, weights = zip(*(extract_data(file, False) for file in v_flat_files))
master_flat_mean = np.mean(all_data, axis=0)
master_flat_median = np.median(all_data, axis=0)
master_flat_weighted = np.average(all_data, axis=0, weights=weights)

# Calculating and printing the rms calues

rms_mean = np.sqrt(np.mean(master_flat_mean**2))
rms_median = np.sqrt(np.mean(master_flat_median**2))
rms_sigma_clip = np.sqrt(np.mean(master_flat_sigma_clip**2))
rms_weighted = np.sqrt(np.mean(master_flat_weighted**2))

print(f"RMS (mean): {rms_mean}")
print(f"RMS (median): {rms_median}")
print(f"RMS (sigma-clipping): {rms_sigma_clip}")
print(f"RMS (weighted average): {rms_weighted}")
```

```
RMS (mean): 1.0001202010421548
RMS (median): 0.9999694360973599
RMS (sigma-clipping): 1.0006297750340538
RMS (weighted average): 1.000120201042155
```

We can ignore the mean and median method as they are trivial and is not giving a significant advantage comparing to ther methods. Between the sigma-clipping method and the weighted average method, the sigma clip method has less rms value, means the noise is less for this method. We can try this method for other filters. The master flat for this method is plotted in the following cell.

I also tried to combine with different sigma clip intervals. And I found when the range is between +5σ and -3σ , it works the best. But now in this revision, after correcting all the files it looks like when the range is between 2σ or 3σ it works the best.

In [6]:
```
show_image(master_flat_sigma_clip)
```

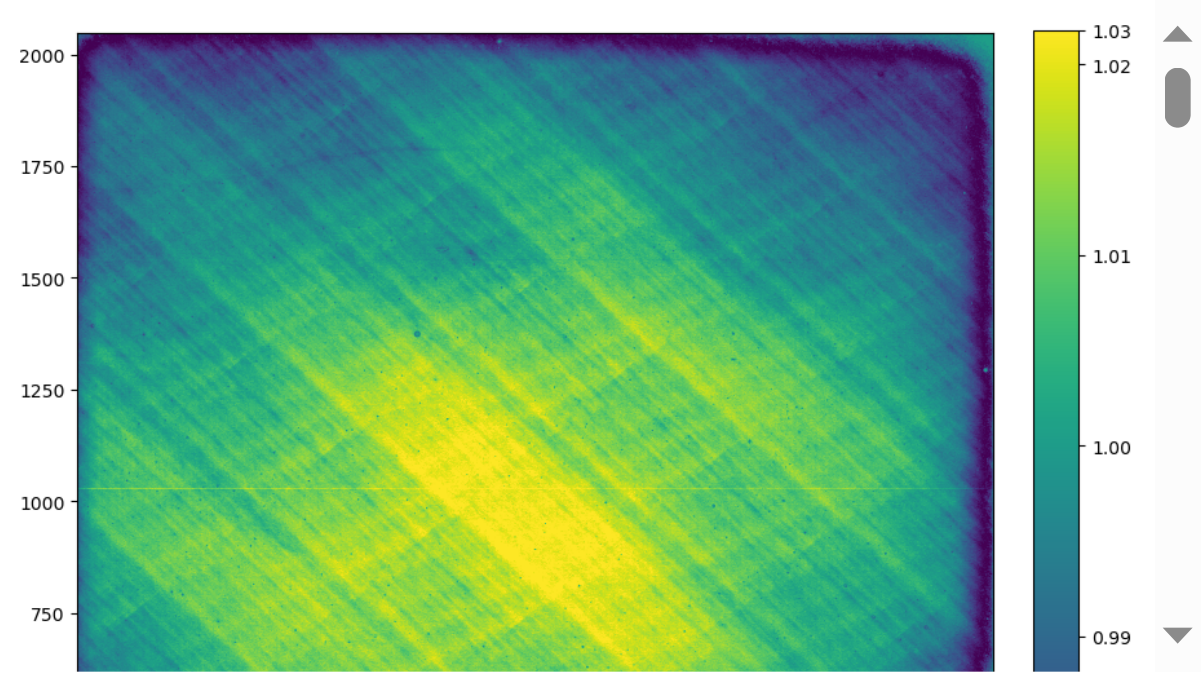


Now we can get the master flat for each of the filter

In [7]:
```
imgs = ccdp.ImageFileCollection(reduced_dir,glob_include='*otz.fits')

# For V filter
v_flat_files = imgs.files_filtered(imagetyp='skyflat',filter='V',include_path=
all_data, _ = zip(*(extract_data(file, True) for file in v_flat_files))
v_master_flat = np.nanmean(all_data, axis=0)

# For U filter
u_flat_files = imgs.files_filtered(imagetyp='skyflat',filter='U',include_path=
all_data, _ = zip(*(extract_data(file, True) for file in u_flat_files))
u_master_flat = np.nanmean(all_data, axis=0)

# For B filter
b_flat_files = imgs.files_filtered(imagetyp='skyflat',filter='B',include_path=
all_data, _ = zip(*(extract_data(file, True) for file in b_flat_files))
b_master_flat = np.nanmean(all_data, axis=0)

# For R filter
r_flat_files = imgs.files_filtered(imagetyp='skyflat',filter='R',include_path=
all_data, _ = zip(*(extract_data(file, True) for file in r_flat_files))
r_master_flat = np.nanmean(all_data, axis=0)

# For I filter
i_flat_files = imgs.files_filtered(imagetyp='skyflat',filter='I',include_path=
all_data, _ = zip(*(extract_data(file, True) for file in i_flat_files))
i_master_flat = np.nanmean(all_data, axis=0)

# For Ha filter
ha_flat_files = imgs.files_filtered(imagetyp='skyflat',filter='Ha',include_pat
all_data, _ = zip(*(extract_data(file, True) for file in ha_flat_files))
ha_master_flat = np.nanmean(all_data, axis=0)
```

Now we can get plot the master falt for each filter

In [8]:
```
# For V filter
show_image(v_master_flat)

# For U filter
show_image(u_master_flat)

# For B filter
show_image(b_master_flat)

# For R filter
show_image(r_master_flat)

# For I filter
show_image(i_master_flat)

# For Ha filter
show_image(ha_master_flat)
```



In [9]:
```
v_master_flat = (v_master_flat/np.mean(v_master_flat))
u_master_flat = (u_master_flat/np.mean(u_master_flat))
b_master_flat = (b_master_flat/np.mean(b_master_flat))
r_master_flat = (r_master_flat/np.mean(r_master_flat))
i_master_flat = (i_master_flat/np.mean(i_master_flat))
ha_master_flat = (ha_master_flat/np.mean(ha_master_flat))
```

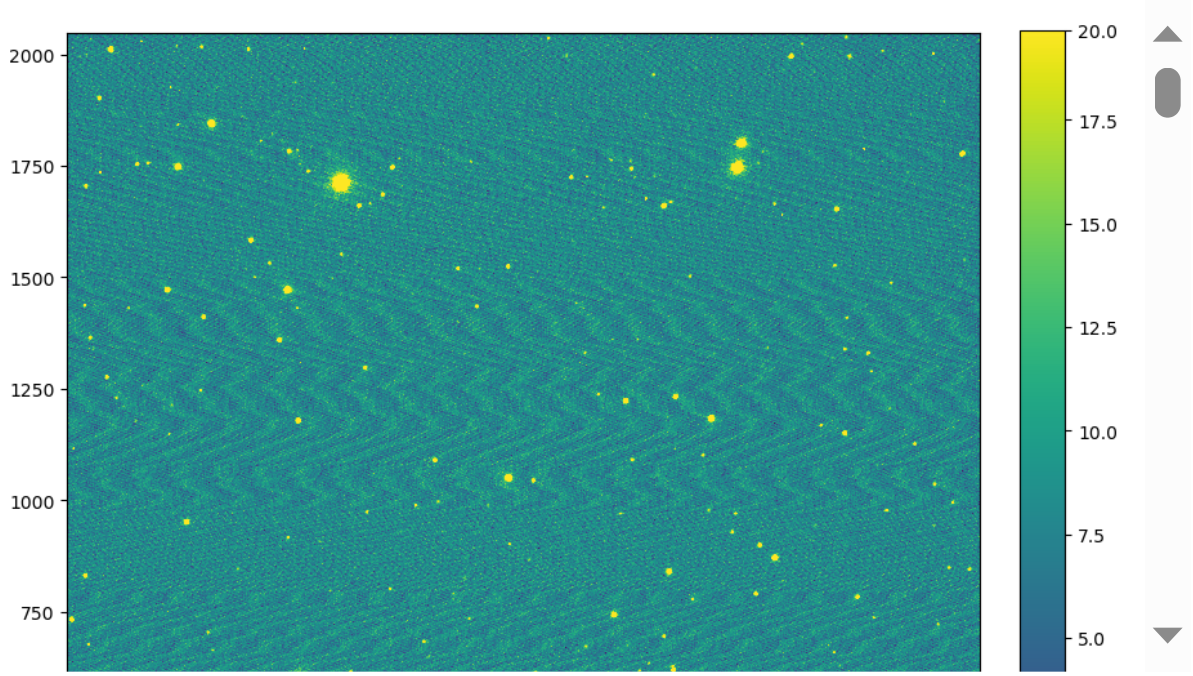
Now we can divide all the science filed by the normalized master flat.

In the following few cells we divide the science image by the master flat, save it, and plot it.

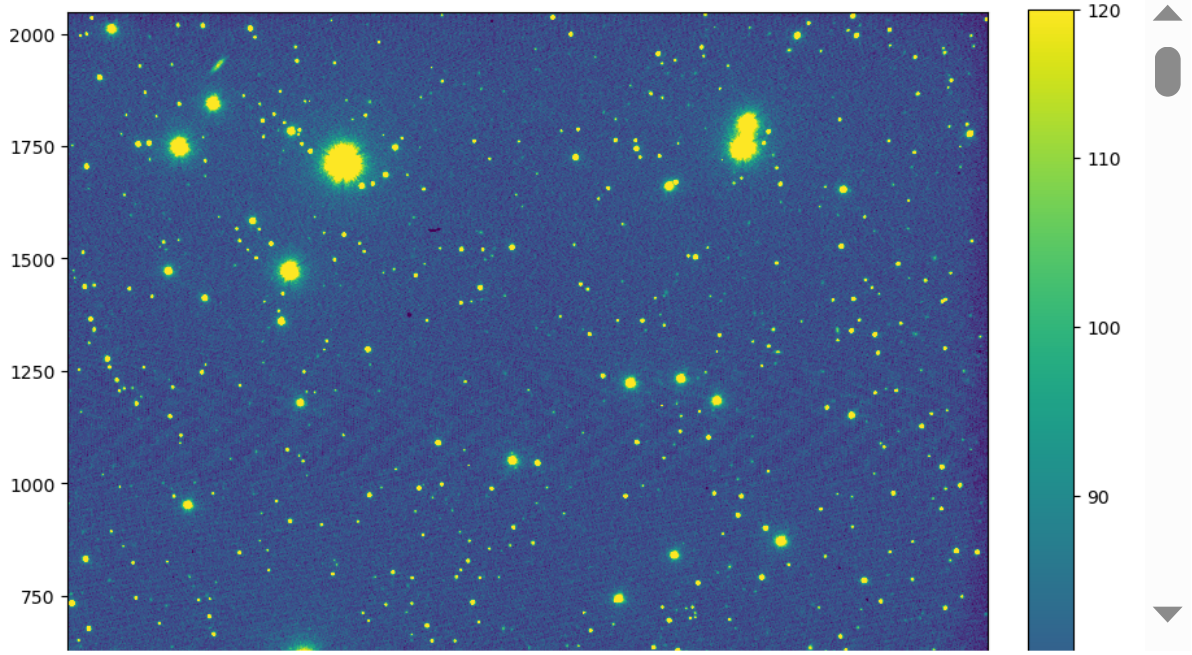
In [10]:
```
v_science_files = imgs.files_filtered(imagetyp='science',filter='V',include_pa
print(v_science_files)

# Plotting and saving for V filter
for file in v_science_files:
    with fits.open(file) as hdul:
        data = hdul[0].data
        header = hdul[0].header
print(v_science_files)
    corrected_data = data / v_master_flat

    new_filename = file.replace('.fits', 'f.fits')

    fits.writeto(new_filename, corrected_data, header, overwrite=True)

    show_image(corrected_data)
```

```
['Imaging_reduced/a064otz.fits', 'Imaging_reduced/a065otz.fits', 'Imaging
_reduced/a066otz.fits', 'Imaging_reduced/a067otz.fits', 'Imaging_reduced/
a068otz.fits', 'Imaging_reduced/a069otz.fits', 'Imaging_reduced/a070otz.f
its', 'Imaging_reduced/a071otz.fits', 'Imaging_reduced/a072otz.fits', 'Im
aging_reduced/a073otz.fits', 'Imaging_reduced/a074otz.fits', 'Imaging_red
uced/a075otz.fits', 'Imaging_reduced/a076otz.fits', 'Imaging_reduced/a077
otz.fits', 'Imaging_reduced/a078otz.fits', 'Imaging_reduced/a079otz.fit
s', 'Imaging_reduced/a080otz.fits', 'Imaging_reduced/a085otz.fits', 'Imag
ing_reduced/a086otz.fits', 'Imaging_reduced/a134otz.fits', 'Imaging_reduc
ed/a135otz.fits', 'Imaging_reduced/a157otz.fits', 'Imaging_reduced/a158ot
z.fits', 'Imaging_reduced/a159otz.fits', 'Imaging_reduced/a170otz.fits',
'Imaging_reduced/a171otz.fits', 'Imaging_reduced/a172otz.fits', 'Imaging_
reduced/a183otz.fits', 'Imaging_reduced/a184otz.fits', 'Imaging_reduced/a
206otz.fits', 'Imaging_reduced/a207otz.fits', 'Imaging_reduced/a215otz.fi
ts', 'Imaging_reduced/a216otz.fits', 'Imaging_reduced/a217otz.fits', 'Ima
ging_reduced/a227otz.fits', 'Imaging_reduced/a228otz.fits', 'Imaging_redu
ced/a229otz.fits', 'Imaging_reduced/a230otz.fits']

C:\Users\serat\Downloads\convenience_function.py:46: RuntimeWarning: More
than 20 figures have been opened. Figures created through the pyplot inte
```

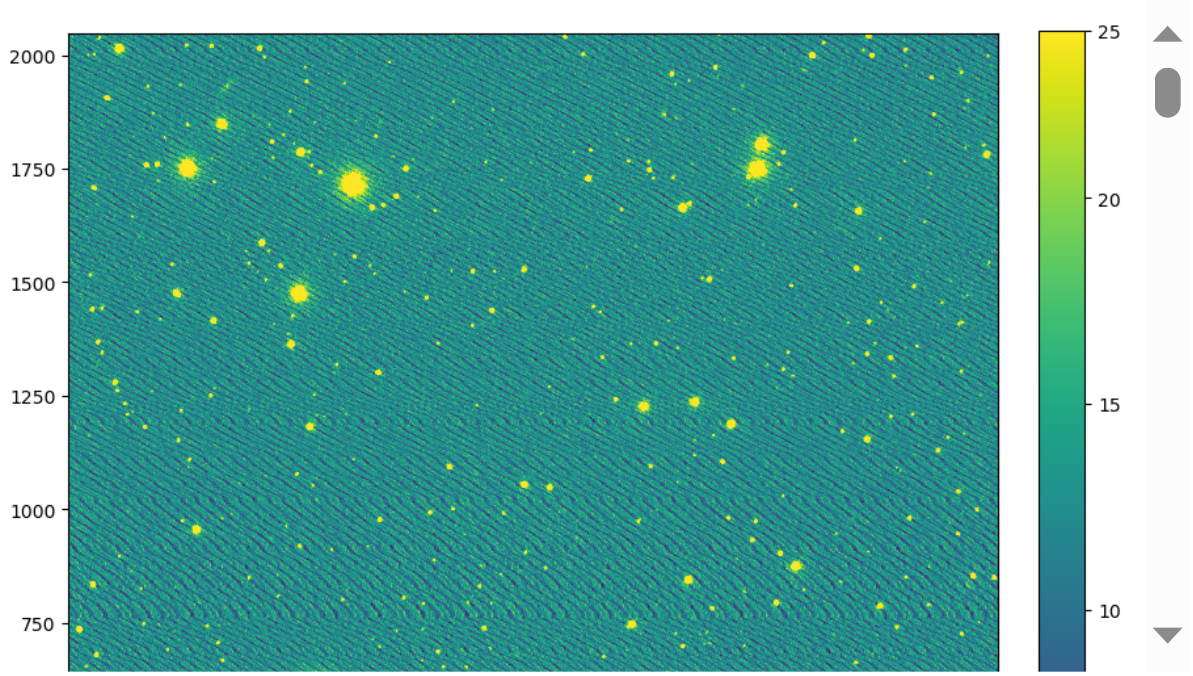
In [11]:
```
u_science_files = imgs.files_filtered(imagetyp='science',filter='U',include_pa

# Plotting and saving for U filter
for file in u_science_files:
    with fits.open(file) as hdul:
        data = hdul[0].data
        header = hdul[0].header

    corrected_data = data / u_master_flat

    new_filename = file.replace('.fits', 'f.fits')

    fits.writeto(new_filename, corrected_data, header, overwrite=True)

    show_image(corrected_data)
```



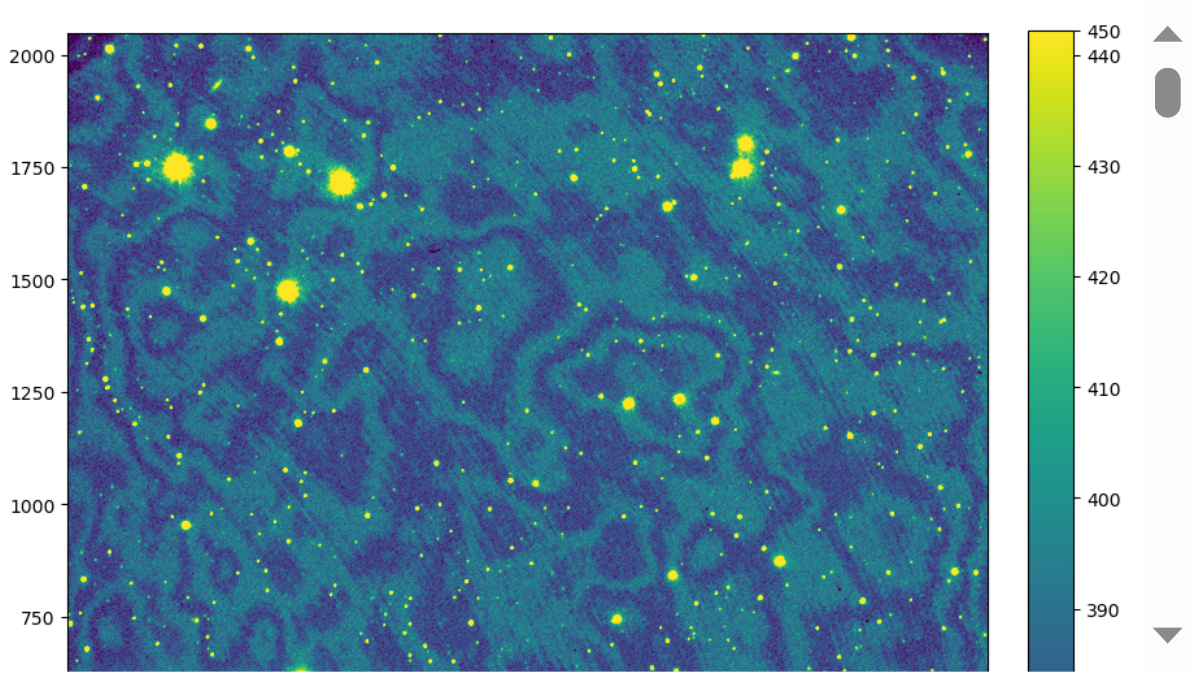
In [12]:
```
b_science_files = imgs.files_filtered(imagetyp='science',filter='B',include_pa

# Plotting and saving for B filter
for file in b_science_files:
    with fits.open(file) as hdul:
        data = hdul[0].data
        header = hdul[0].header

    corrected_data = data / b_master_flat

    new_filename = file.replace('.fits', 'f.fits')

    fits.writeto(new_filename, corrected_data, header, overwrite=True)

    show_image(corrected_data)
```



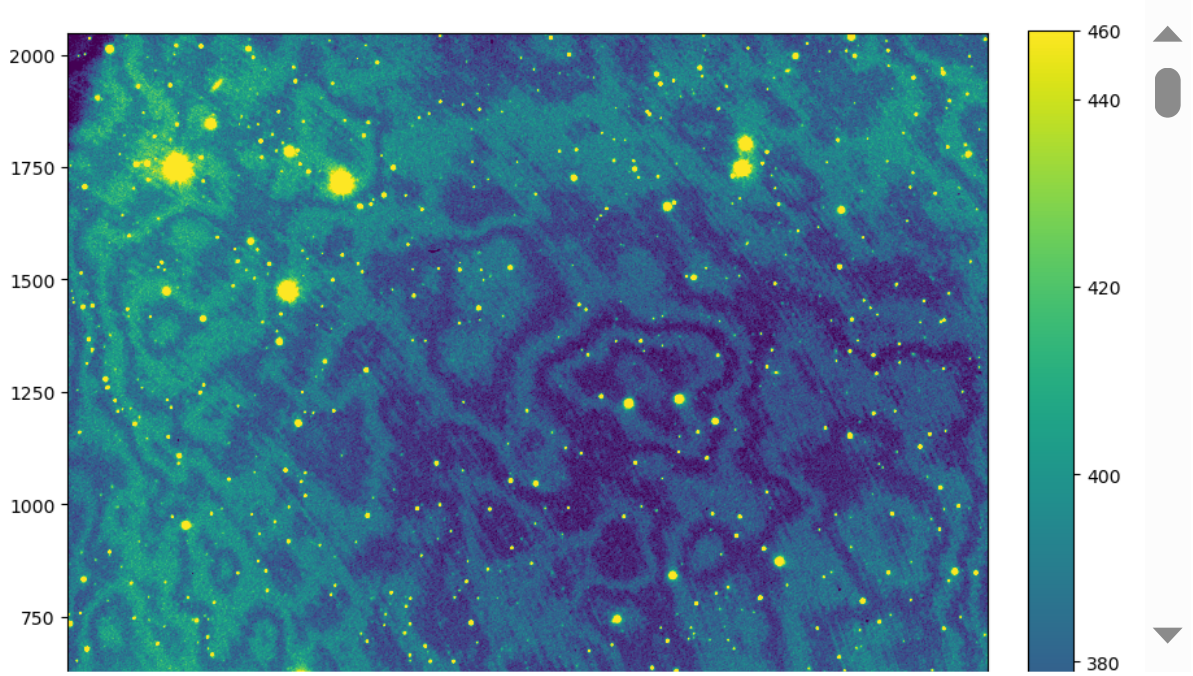
In [13]:
```
r_science_files = imgs.files_filtered(imagetyp='science',filter='R',include_pa

# Plotting and saving for R filter
for file in v_science_files:
    with fits.open(file) as hdul:
        data = hdul[0].data
        header = hdul[0].header

    corrected_data = data / r_master_flat

    new_filename = file.replace('.fits', 'f.fits')

    fits.writeto(new_filename, corrected_data, header, overwrite=True)

    show_image(corrected_data)
```



In [14]:
```
i_science_files = imgs.files_filtered(imagetyp='science',filter='I',include_pa

# Plotting and saving for I filter
for file in i_science_files:
    with fits.open(file) as hdul:
        data = hdul[0].data
        header = hdul[0].header

    corrected_data = data / i_master_flat

    new_filename = file.replace('.fits', 'f.fits')

    fits.writeto(new_filename, corrected_data, header, overwrite=True)

    show_image(corrected_data)
```



In [15]:
```
ha_science_files = imgs.files_filtered(imagetyp='science',filter='Ha',include_

# Plotting and saving for Ha filter
for file in i_science_files:
    with fits.open(file) as hdul:
        data = hdul[0].data
        header = hdul[0].header

    corrected_data = data / ha_master_flat

    new_filename = file.replace('.fits', 'f.fits')

    fits.writeto(new_filename, corrected_data, header, overwrite=True)

    show_image(corrected_data)
```



Let's call a few bias and flat flies. Then we can find its mean values, differences between pairs of bias files, and the we can find the gain and read noise of them using the following equations:

$$Gain = \frac{(\bar{F1} + \bar{F2}) - (\bar{B1} + \bar{B2})}{\sigma^2_{F1-F2} - \sigma^2_{B1-B2}}$$

and,

$$Read\ Noise = \frac{Gain \cdot \sigma_{B1-B2}}{\sqrt{2}}$$

Where $F1$ and $F2$ are the flat images and $B1$ and $B2$ are the bias images.

Comment: The flat files that you use for this calculation need to be the same (i.e. the counts can't be super different, identical filter). Random pairs of flats need a correction, so that's going to contribute to why your values are so diverse.

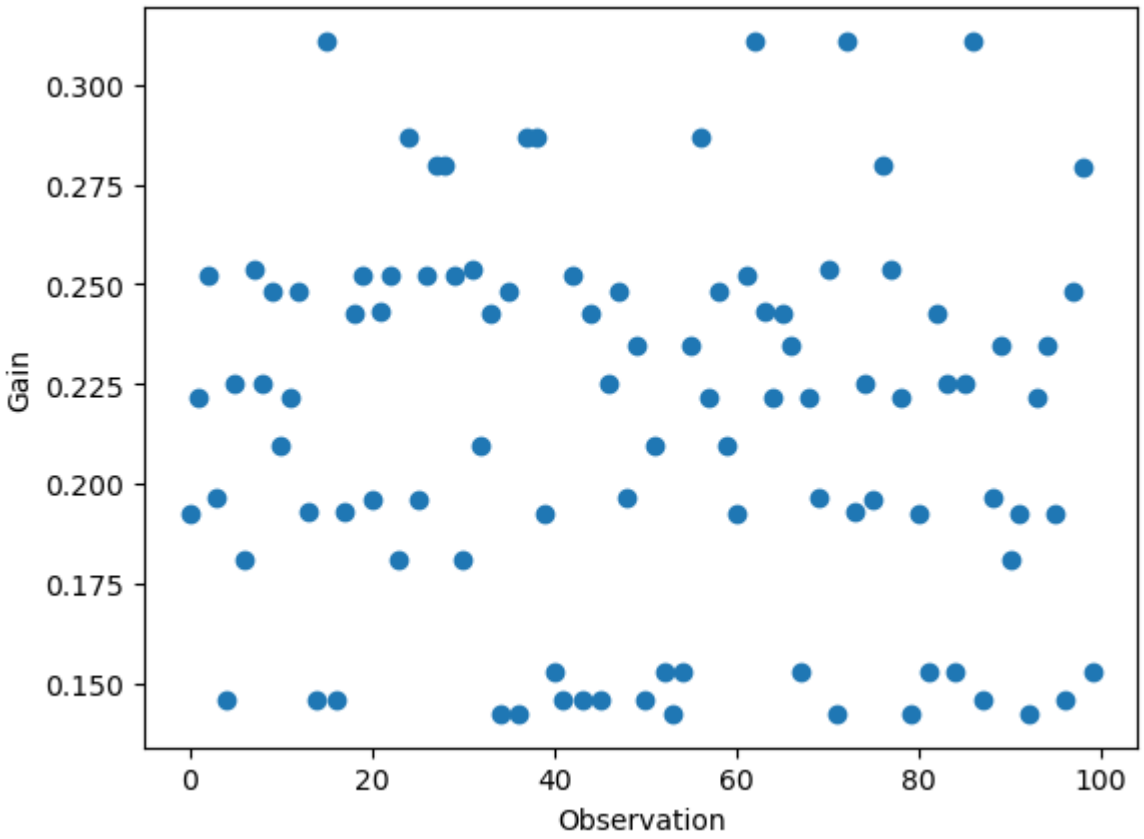
You need to use flat-fielded flats for this, too. This was given in the prompt

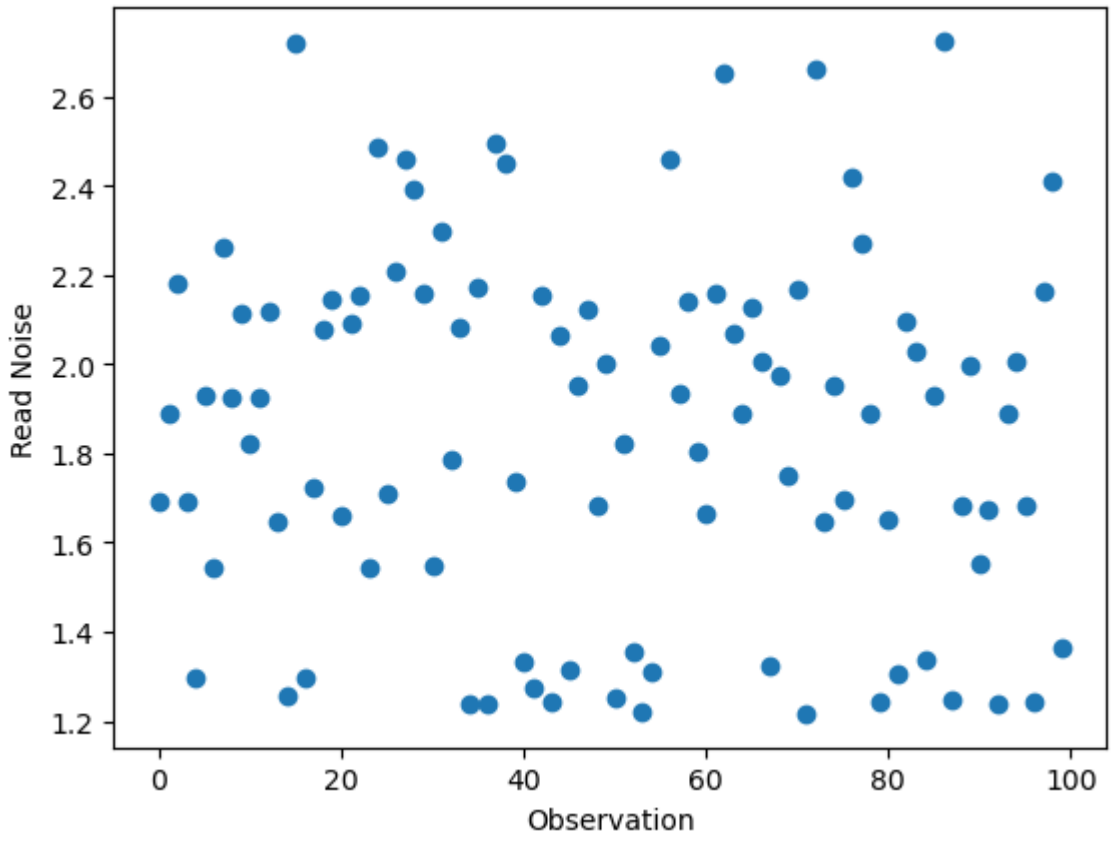
I went through this part again and clacuated the gain and read noise only for flat_fielded flats.

In [16]:
```
# Calling the bias and flat files for V filter
flat_files = imgs.files_filtered(imagetyp='skyflat',filter='V',include_path=Tr
bias_files = imgs.files_filtered(imagetyp='bias',include_path=True)


gain = []
read_noise = []
# Selecting a pair in random from two files in random for 100 times,
# then going through the pair to get their difference and mean, and std of dif
for i in range(100):
    flat_pair = random.sample(flat_files, 2)
    flat1 = fits.open(flat_pair[0])[0].data
    flat2 = fits.open(flat_pair[1])[0].data


    mean_flat = [np.mean(flat1), np.mean(flat2)] # Getting the mean value of t
    diff_flat = flat1 - flat2
    sigma_diff_flat = np.std(diff_flat)

    bias_pair = random.sample(bias_files, 2)
    bias1 = fits.open(bias_pair[0])[0].data
    bias2 = fits.open(bias_pair[1])[0].data
    mean_bias = [np.mean(bias1), np.mean(bias2)] # Getting the mean value of t
    diff_bias = bias1 - bias2
    sigma_diff_bias = np.std(diff_bias)

    # Getting the gain
    g = ((mean_flat[0] + mean_flat[1]) - (mean_bias[0] + mean_bias[1]))/(sigma
    gain.append(g)

    # Getting the read_noise
    rn = (g * sigma_diff_bias) / math.sqrt(2)
    read_noise.append(rn)
```

In the following cell we can plot all these 100 gain and read noise values to see how they are distributed.

In [17]:
```
line_len = np.arange(len(gain))

plt.scatter(line_len, gain)
plt.xlabel("Observation")
plt.ylabel("Gain")
plt.show()

plt.scatter(line_len, read_noise)
plt.xlabel("Observation")
plt.ylabel("Read Noise")
plt.show()
```





We can also find the mean values of these 100 different gain and read noise.

In [18]:
```
v_mean_gain = np.mean(gain)
v_mean_rn = np.mean(read_noise)

print("Gain:", round(v_mean_gain, 3))
print("Read Noise", round(v_mean_rn, 3))
```

```
Gain: 0.215
Read Noise 1.856
```

We can claculate the gain and read noise for the flats of other filters and see how they can be compared between each other:

In [19]:
```
# The following function can be helpful so that we don't need to repeat the sa
def calculate_gain_and_read_noise(filter_type, imgs, num_iterations=100):
    flat_files = imgs.files_filtered(imagetyp='skyflat', filter=filter_type, i
    bias_files = imgs.files_filtered(imagetyp='bias', include_path=True)

    gain = []
    read_noise = []

    for _ in range(num_iterations):

        flat_pair = random.sample(flat_files, 2)
        flat1 = fits.open(flat_pair[0])[0].data
        flat2 = fits.open(flat_pair[1])[0].data

        bias_pair = random.sample(bias_files, 2)
        bias1 = fits.open(bias_pair[0])[0].data
        bias2 = fits.open(bias_pair[1])[0].data

        mean_flat = [np.mean(flat1), np.mean(flat2)]
        diff_flat = flat1 - flat2
        sigma_diff_flat = np.std(diff_flat)

        mean_bias = [np.mean(bias1), np.mean(bias2)]
        diff_bias = bias1 - bias2
        sigma_diff_bias = np.std(diff_bias)

        g = ((mean_flat[0] + mean_flat[1]) - (mean_bias[0] + mean_bias[1])) /
        gain.append(g)

        rn = (g * sigma_diff_bias) / math.sqrt(2)
        read_noise.append(rn)

    mean_gain = np.mean(gain)
    mean_read_noise = np.mean(read_noise)

    return mean_gain, mean_read_noise


v_gain, v_read_noise = calculate_gain_and_read_noise('V', imgs)
u_gain, u_read_noise = calculate_gain_and_read_noise('U', imgs)
b_gain, b_read_noise = calculate_gain_and_read_noise('B', imgs)
i_gain, i_read_noise = calculate_gain_and_read_noise('I', imgs)
r_gain, r_read_noise = calculate_gain_and_read_noise('R', imgs)
```

Now we can can compare the gain and read noise in the following cell by printing them:

In [20]:
```
print("Filter Type | Gain | Read Noise")
print("-----------------------------------")
print(f"U           | {u_gain:.2f} | {u_read_noise:.2f}")
print(f"V           | {v_gain:.2f} | {v_read_noise:.2f}")
print(f"B           | {b_gain:.2f} | {b_read_noise:.2f}")
print(f"I           | {i_gain:.2f} | {i_read_noise:.2f}")
print(f"R           | {r_gain:.2f} | {r_read_noise:.2f}")
```

```
Filter Type | Gain | Read Noise
-----------------------------------
U           | 0.47 | 4.08
V           | 0.21 | 1.85
B           | 0.43 | 3.72
I           | 0.03 | 0.30
R           | 0.06 | 0.50
```

In [ ]: