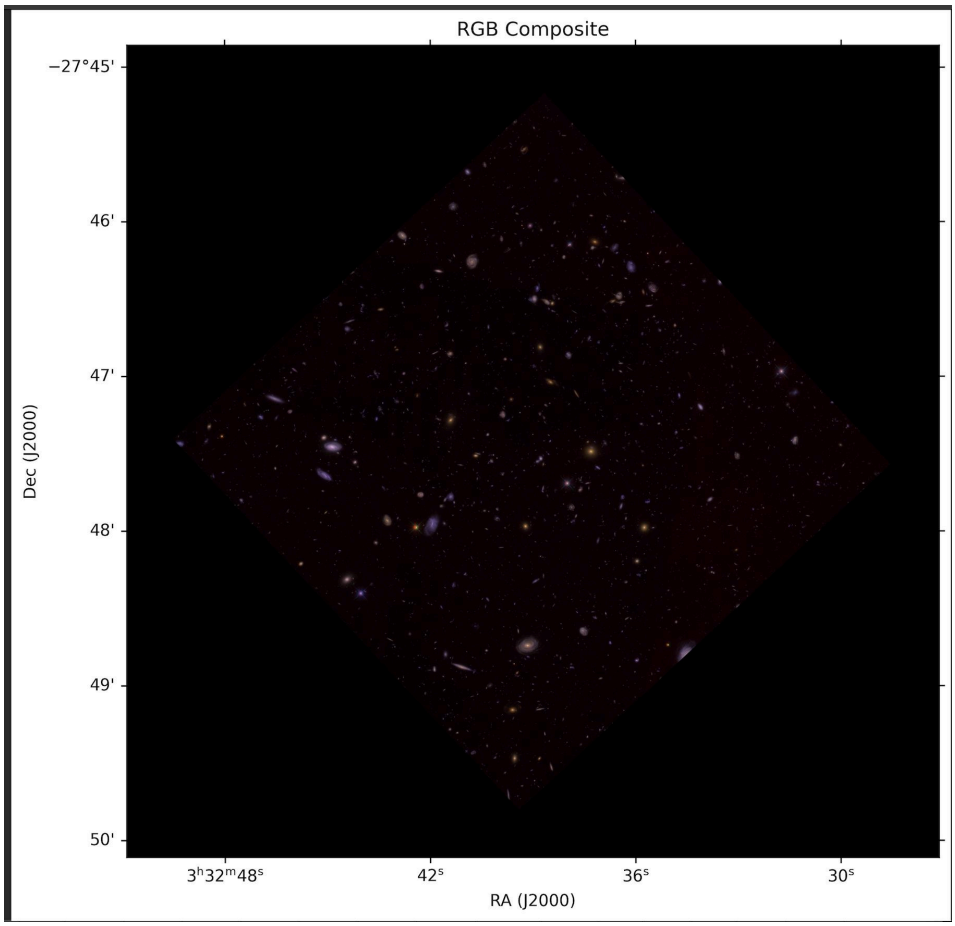
Write-Up: Includes all plots and certain coding choices that weren't highlighted in the README.txt , sort of a redundancy but kept both.

Some quick references that are also mentioned in the README

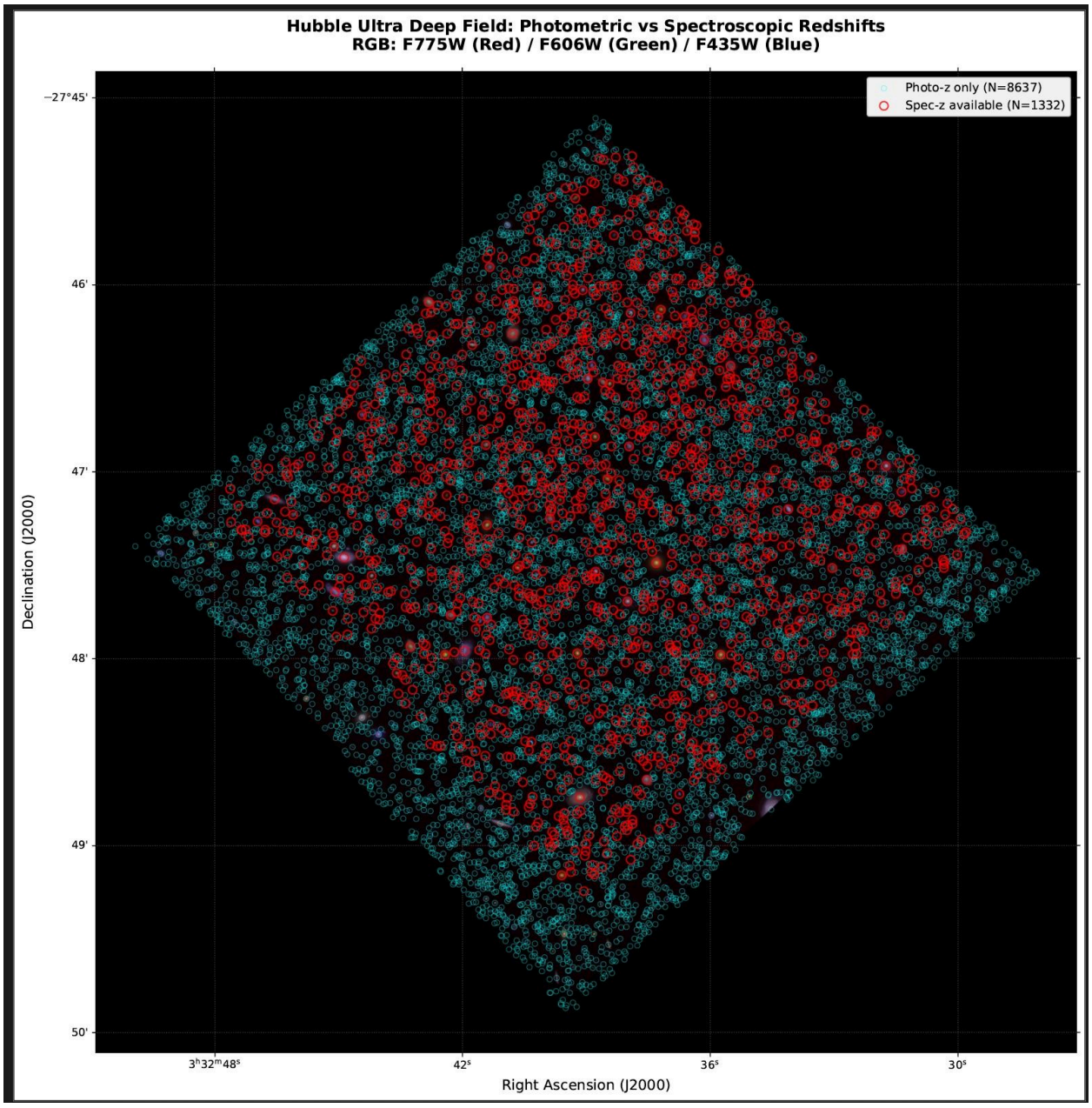
Fits files: https://mast.stsci.edu/portal/Mashup/Clients/Mast/Portal.html
Spectroscopy catalog:
https://www.aanda.org/articles/aa/full_html/2017/12/aa31195-17/aa31195-17.html (Inami+2017)
Photometry catalog : https://ui.adsabs.harvard.edu/abs/2016yCat..51500031R/abstract (Rafelski+2015)
np.ma.filled : https://numpy.org/doc/2.3/reference/generated/numpy.ma.filled.html
Lupton RGB: https://docs.astropy.org/en/stable/api/astropy.visualization.make_lupton_rgb.html

RGB COMPOSITE:



I used the fits files from the MAST website to generate this 3-color composite plot. I did not use nstack as the saturation of the images was a little difficult to discern [i.e., i did not know what stretches to use] I ended up using make_lupton_rgb (2004) from astropy, which had an inbuilt Asinh stretch, and I messed around a little with the Q and stretch values to get a good saturation and contrast.

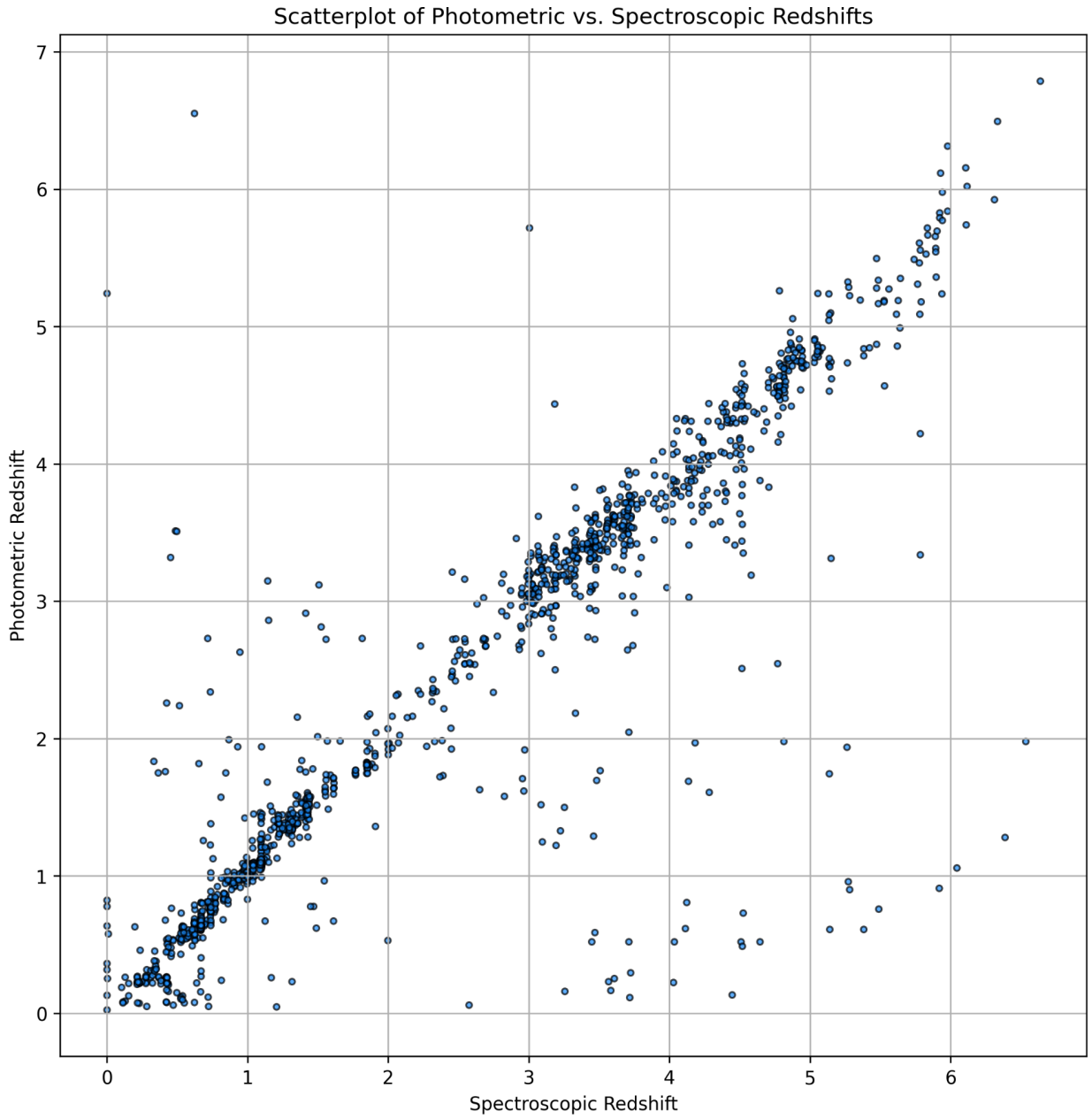
Overlay with cross-matching catalogs:



As mentioned in the readme, for the cross-verification of catalogs, I considered various modules like the np.core.records.fromarrays, and the xmatch module from Astropy. However, I thought of an intuitive way that seemed to work best for me. I explained this in detail in the documentation but I used
.match_to_catalog_sky from skycoords to get the nearest coordinate match from the photometry catalog to the spectroscopy catalog. Before doing this, however, I realized that the number of significant figures in the ra and dec of both catalogs were different, so I rounded the catalogs to

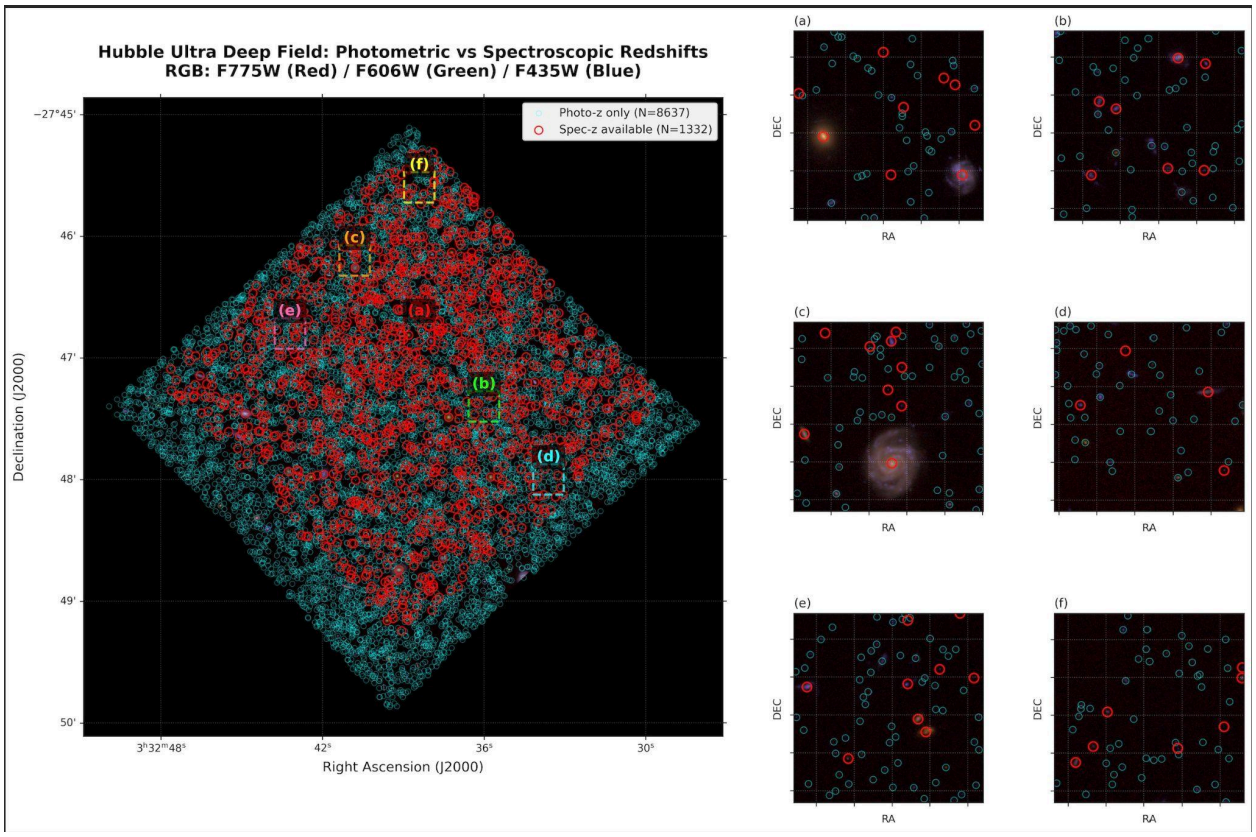
6 decimal places for consistency. Now, if there was a match, the value of the .match_to_catalog_sky would honestly just be 0, since they seemed to perfectly match for a few points that I tested for eye. Still, I gave a 0.0036 arcsecond buffer in the catalog matching, and did the matching using a rather intricate method using np.where(mask) and a basic mask which returned true if the separation < 0.0036. After this was done, I saved the indices where the Mask gave me a true value in the spectrscopy catalog and cross matched those indices with the photometry catalog: I.e., I took advantage of the fact that all the photometric redshift galaxies were present in the spectroscopy catalog, but not vice versa.

Scatterplot of photometric vs spectroscopic redshifts



The main issue i faced was that a few values in the spectroscopy catalog had their respective z values but the corresponding z values in the photometric catalogs were sometimes masked as “---” or vice versa. In such cases, I had to convert those points, using np.ma.filled with nan values and then mask the nan and negative values to get a final list of z values that had real positive decimal values in both catalogs. Once this was done, I was able to readily plot the points.

Insets:



For the insets! The most challenging part of this project– I wasn't able to plot the coordinates next to the inset as the wcs scaling changes once we extract the subregion because the pixels now make up different amounts of the RA and DEC. I initially tried creating a new WcS, copying the old one and trimming it somehow, but this just was NOT working. I decided to just plot the pixels as is and remove the tic labels to avoid any irrelevant and redundant pixel axes. To plot these, I created a gridspec using matplotlib to create the insets. I had two functions on this overall, one that created the insets, and one that created the overlayer in the main grid. These were interconnected and called on each other to run. I had to recovery the pixel_scale using the determinant of the wcs pixel scale matrix in arcseconds to accurately extract a subregion from the main plot. I then also had to overlay the plots and did that in a similar fashion. Now that i had added the extra “has spec” column in both csv files, this was easily done. The choice of the squares and their locations was pretty arbitrary and i got it from trial and error once I could

grasp how much small decimal points shifted the square. For this reason, i also removed the extra test line from the code, and commented them out.