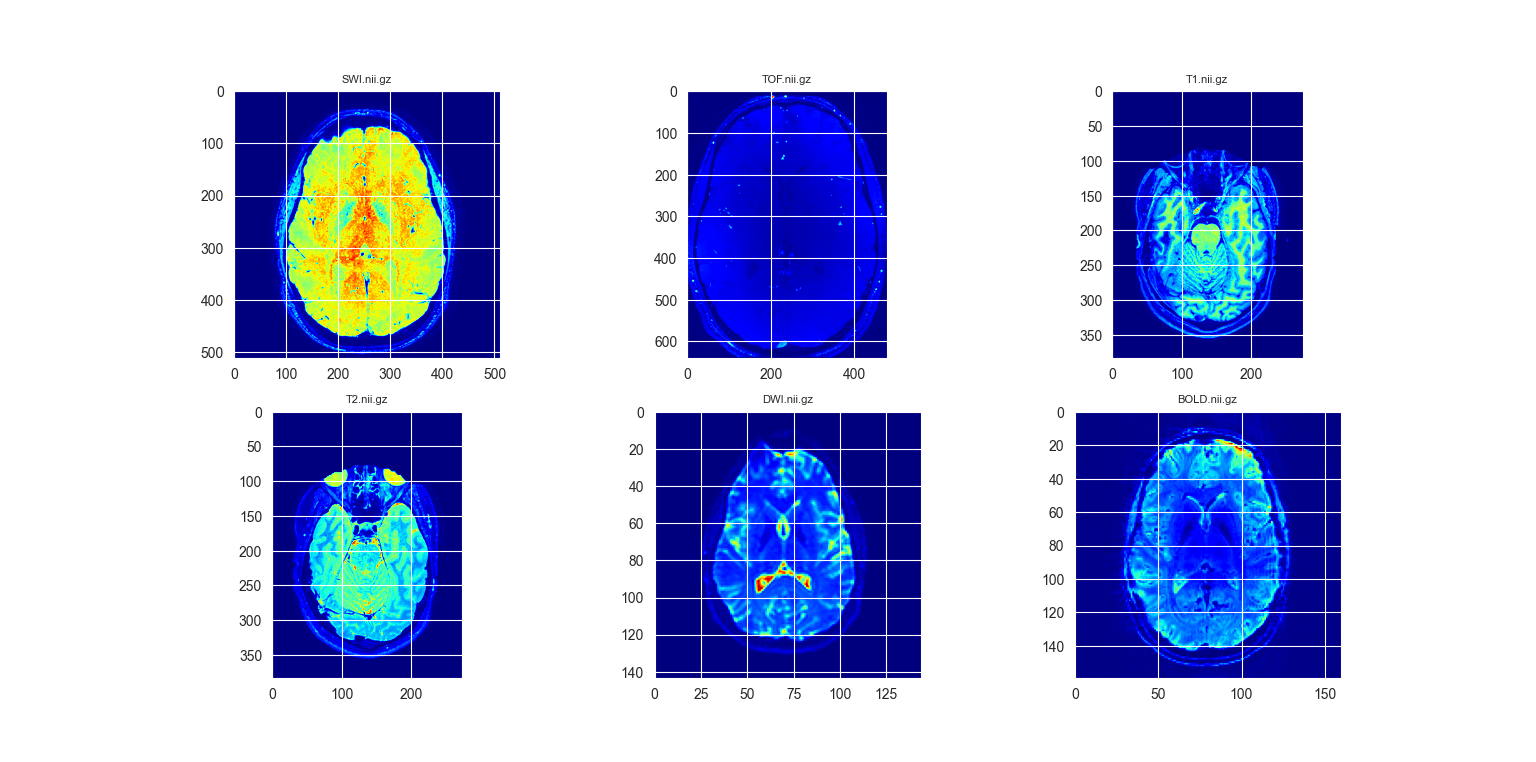
**Assigment1**

part1-a)



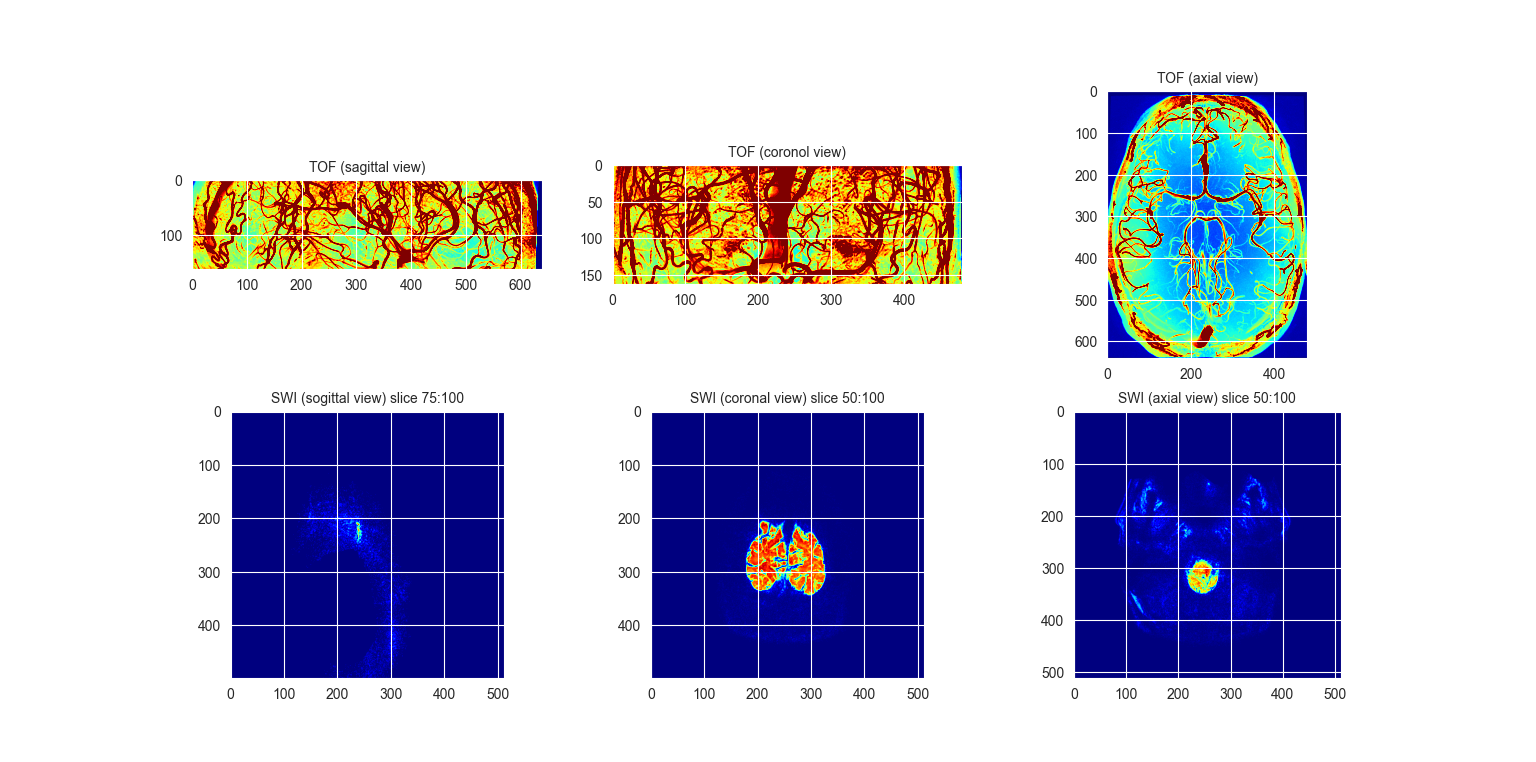
import os  
import numpy as np  
import nibabel as nib  
import matplotlib.pyplot as plt  
import seaborn as sns  
  
#Dataset of image path  
image\_path\_list = ['d:\ImgDrButtler\SWI.nii.gz', 'd:\ImgDrButtler\\TOF.nii.gz',  
'd:\ImgDrButtler\T1.nii.gz', 'd:\ImgDrButtler\T2.nii.gz',  
 'd:\ImgDrButtler\DWI.nii.gz', 'd:\ImgDrButtler\BOLD.nii.gz']

#Read image as nifti image

swi=nib.load(image\_path\_list[0]).get\_fdata()  
tof=nib.load(image\_path\_list[1]).get\_fdata()  
t1=nib.load(image\_path\_list[2]).get\_fdata()  
t2=nib.load(image\_path\_list[3]).get\_fdata()  
dwi=nib.load(image\_path\_list[4]).get\_fdata()  
bold=nib.load(image\_path\_list[5]).get\_fdata()

#set style of backgroung  
sns.set\_style('darkgrid')  
  
  
#function for show dimesion of images /show 2axes in z slice   
#image(depth/2) / show 3D and 4D image by try except/plot setting  
def show\_3D\_image(image\_obj,titleP):  
 image\_data = image\_obj.get\_fdata()  
 try:  
 height, width, depth = image\_data.shape  
 half\_depth = int(depth / 2)  
 plt.imshow(np.rot90(image\_data[:, :, int(half\_depth)]), cmap='jet')  
 except:  
 print('except')  
 height, width, depth, channel = image\_data.shape  
 half\_depth = int(depth / 2)  
 plt.imshow(np.rot90(image\_data[:, :, int(half\_depth), 0]), cmap='jet')  
 plt.title(titleP, fontsize=8)  
 plt.axis('off')  
  
num = 1  
#create canvas for images  
plt.figure()  
print(len(image\_path\_list))  
  
#for\_loop for switch between images  
for i in (image\_path\_list):  
 #Read .nii file  
 image\_obj = nib.load(i)  
 print(i)  
 #devided plot for show 15 images-6 image for part1  
 plt.subplot(2, 3, num)  
 #call manually function by nibabel image and name of image  
 show\_3D\_image(image\_obj,os.path.basename(i))  
 print(f'num={num}')  
 num += 1  
  
#finall show  
plt.show()

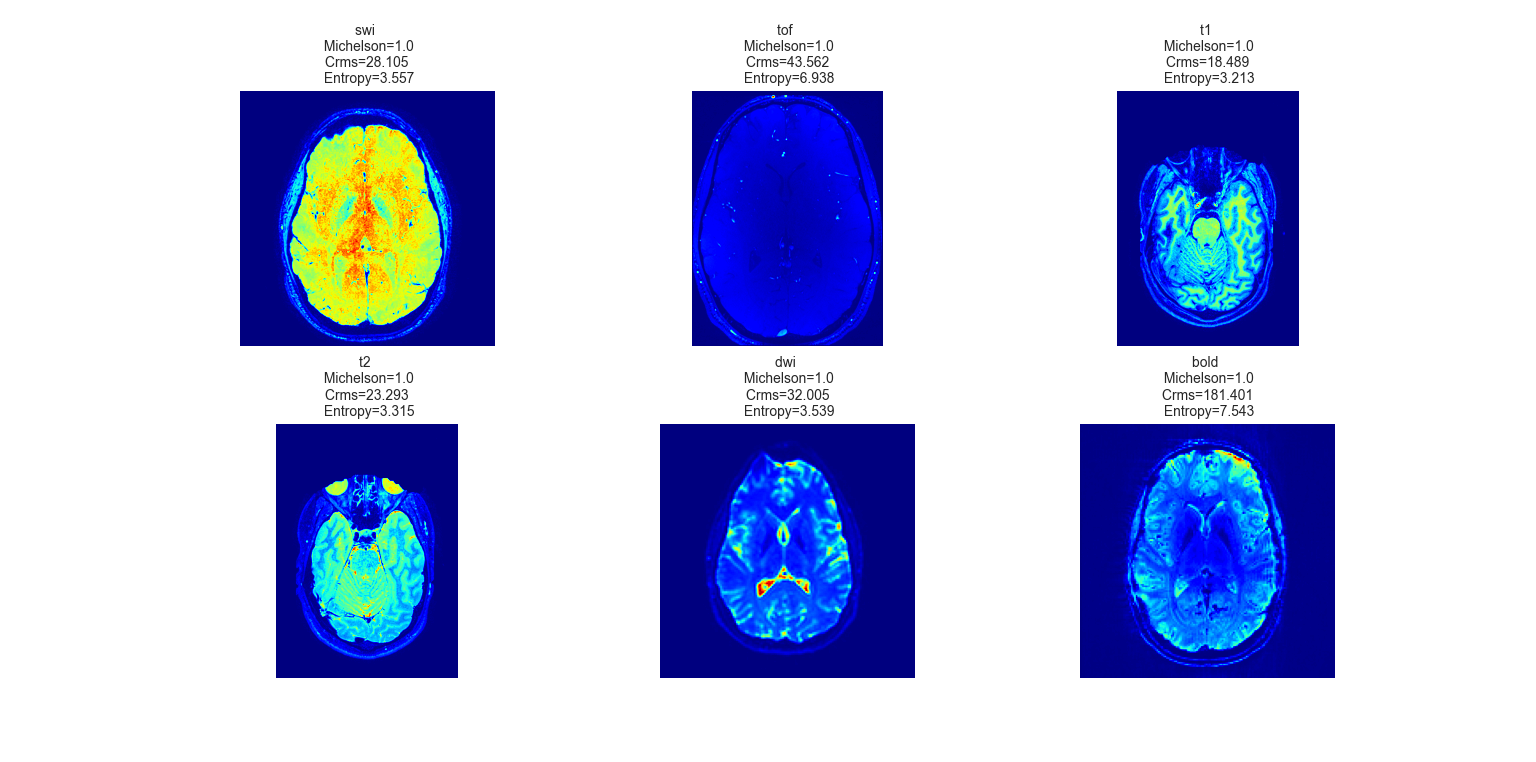
Part 1-b)



#show colorful(jet) Min and Max Intensity(np.max|np.min) of TOF and SWI for each dimension(axis=0|1|2)  
#calculate np.max for tof image  
plt.subplot(2,3,1)  
plt.imshow(np.rot90(np.max(tof, axis=0)), cmap='jet', vmax=300)  
plt.title('TOF (sagittal view)', fontsize=10)  
  
  
plt.subplot(2,3,2)  
plt.imshow(np.rot90(np.max(tof, axis=1)), cmap='jet', vmax=300)  
plt.title('TOF (coronol view)', fontsize=10)  
  
plt.subplot(2,3,3)  
plt.imshow(np.rot90(np.max(tof, axis=2)), cmap='jet', vmax=300)  
plt.title('TOF (axial view)', fontsize=10)  
  
#calulate np.min for swi image  
plt.subplot(2,3,4)  
plt.imshow(np.rot90(np.min(swi[75:100,:,:], axis=0)), cmap='jet')  
plt.title('SWI (sogittal view) slice 75:100', fontsize=10)  
  
  
plt.subplot(2,3,5)  
plt.imshow(np.rot90(np.min(swi[:, 50:100, :], axis=1)), cmap='jet')  
plt.title('SWI (coronal view) slice 50:100', fontsize=10)  
  
  
plt.subplot(2,3,6)  
plt.imshow(np.rot90(np.min(nib.load(image\_path\_list[0]).get\_fdata()[:,:,50:100], axis=2)), cmap='jet')  
plt.title('SWI (axial view) slice 50:100', fontsize=10)

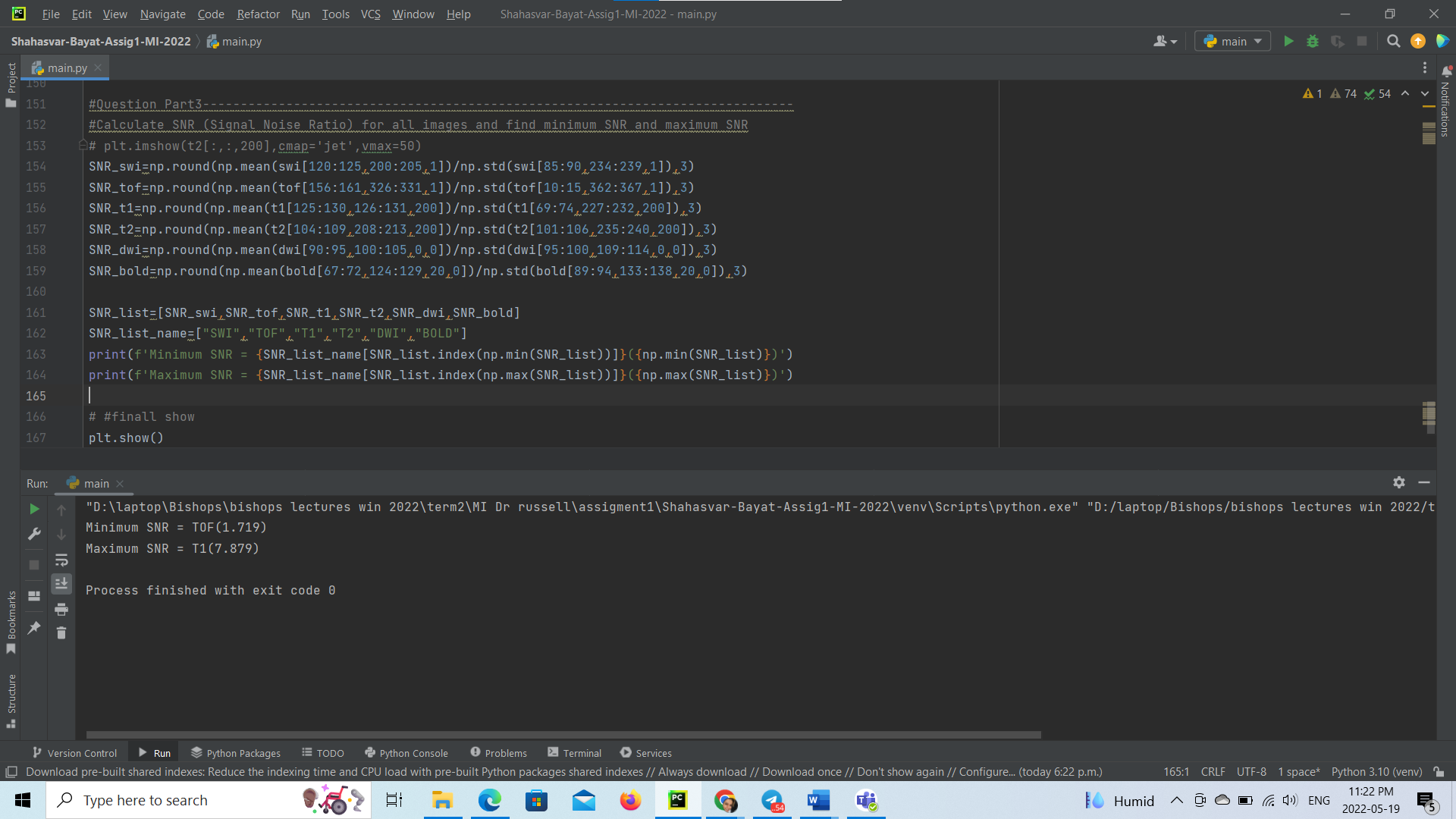
plt.show()

Part 2)



# Calculate Contrast of all images by Michelson - RMS -Entropy  
import skimage.measure as msr  
#calculate michelson contrast for each image  
swi\_Mc=((swi.max()-swi.min())/(swi.max()+swi.min()))  
#calculate Rms contrast for each image  
swi\_Rms=np.round(np.sqrt(1/(swi.size-1)\*np.sum((swi-np.mean(swi))\*\*2)), 3)  
#calculate Entropy for each image  
swi\_entr = np.round(msr.shannon\_entropy(swi),3)  
  
tof\_Mc=((tof.max()-tof.min())/(tof.max()+tof.min()))  
tof\_Rms=np.round(np.sqrt(1/(tof.size-1)\*np.sum((tof-np.mean(tof))\*\*2)),3)  
tof\_entr = np.round(msr.shannon\_entropy(tof),3)  
  
  
t1\_Mc=((t1.max()-t1.min())/(t1.max()+t1.min()))  
t1\_Rms=np.round(np.sqrt(1/(t1.size-1)\*np.sum((t1-np.mean(t1))\*\*2)),3)  
t1\_entr = np.round(msr.shannon\_entropy(t1),3)  
  
t2\_Mc=((t2.max()-t2.min())/(t2.max()+t2.min()))  
t2\_Rms=np.round(np.sqrt(1/(t2.size-1)\*np.sum((t2-np.mean(t2))\*\*2)),3)  
t2\_entr = np.round(msr.shannon\_entropy(t2),3)  
  
dwi\_Mc=((dwi.max()-dwi.min())/(dwi.max()+dwi.min()))  
dwi\_Rms=np.round(np.sqrt(1/(dwi.size-1)\*np.sum((dwi-np.mean(dwi))\*\*2)),3)  
dwi\_entr = np.round(msr.shannon\_entropy(dwi),3)  
  
bold\_Mc=((bold.max()-bold.min())/(bold.max()+bold.min()))  
bold\_Rms=np.round(np.sqrt(1/(bold.size-1)\*np.sum((bold-np.mean(bold))\*\*2)),3)  
bold\_entr = np.round(msr.shannon\_entropy(bold),3)  
  
#plot all images with their contrast  
plt.subplot(2,3,1)  
plt.imshow(np.rot90(swi[:,:,int(swi.shape[2]/2)]),cmap='jet')  
plt.title(f'swi \n Michelson={swi\_Mc}\n Crms={swi\_Rms} \n Entropy={swi\_entr}',fontsize=10)  
plt.axis('off')  
  
plt.subplot(2,3,2)  
plt.imshow(np.rot90(tof[:,:,int(tof.shape[2]/2)]),cmap='jet')  
plt.title(f'tof \n Michelson={tof\_Mc}\n Crms={tof\_Rms} \n Entropy={tof\_entr}',fontsize=10)  
plt.axis('off')  
  
plt.subplot(2,3,3)  
plt.imshow(np.rot90(t1[:,:,int(t1.shape[2]/2)]),cmap='jet')  
plt.title(f't1 \n Michelson={t1\_Mc}\n Crms={t1\_Rms} \n Entropy={t1\_entr}',fontsize=10)  
plt.axis('off')  
  
plt.subplot(2,3,4)  
plt.imshow(np.rot90(t2[:,:,int(t2.shape[2]/2)]),cmap='jet')  
plt.title(f't2 \n Michelson={t2\_Mc}\n Crms={t2\_Rms} \n Entropy={t2\_entr}',fontsize=10)  
plt.axis('off')  
  
plt.subplot(2,3,5)  
plt.imshow(np.rot90(dwi[:,:,int(dwi.shape[2]/2),0]),cmap='jet')  
plt.title(f'dwi \n Michelson={dwi\_Mc}\n Crms={dwi\_Rms} \n Entropy={dwi\_entr}',fontsize=10)  
plt.axis('off')  
# print(dwi.shape)  
  
plt.subplot(2,3,6)  
plt.imshow(np.rot90(bold[:,:,int(bold.shape[2]/2),0]),cmap='jet')  
plt.title(f'bold \n Michelson={bold\_Mc}\n Crms={bold\_Rms} \n Entropy={bold\_entr}',fontsize=10)  
plt.axis('off')

Part 3-a)

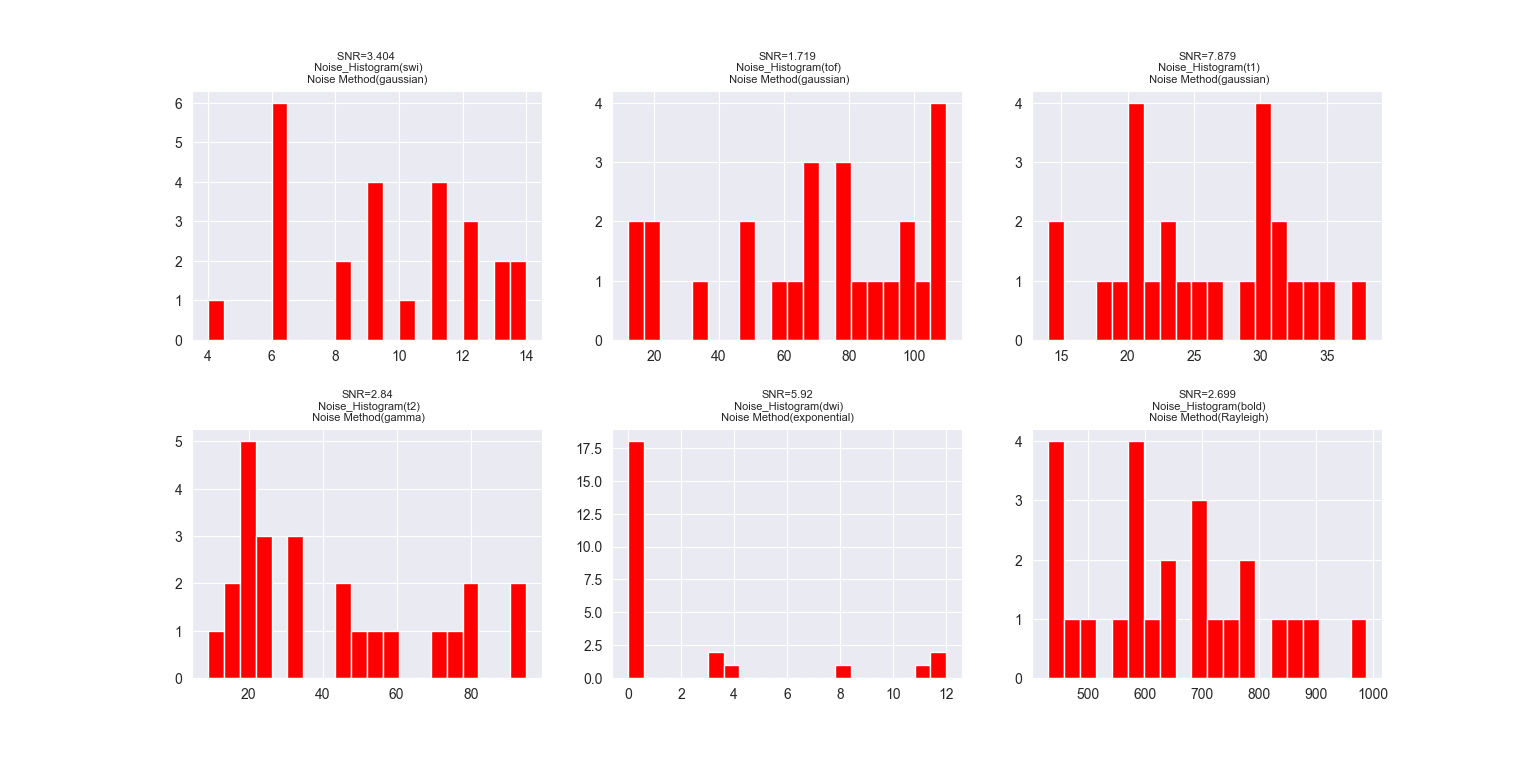


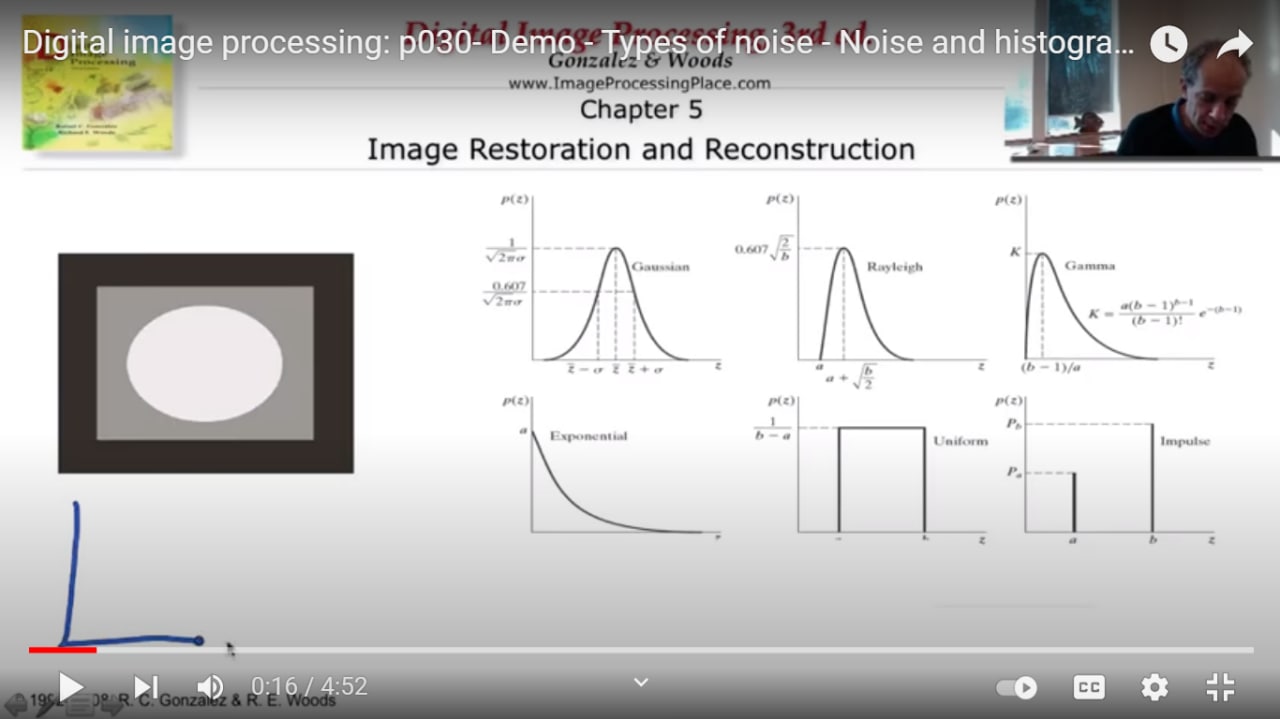
#Calculate SNR (Signal Noise Ratio(np.mean(main image)/np.std(noise))) for all images   
# plt.imshow(t2[:,:,200],cmap='jet',vmax=50)  
SNR\_swi=np.round(np.mean(swi[120:125,200:205,1])/np.std(swi[85:90,234:239,1]),3)  
SNR\_tof=np.round(np.mean(tof[156:161,326:331,1])/np.std(tof[10:15,362:367,1]),3)  
SNR\_t1=np.round(np.mean(t1[125:130,126:131,200])/np.std(t1[69:74,227:232,200]),3)  
SNR\_t2=np.round(np.mean(t2[104:109,208:213,200])/np.std(t2[101:106,235:240,200]),3)  
SNR\_dwi=np.round(np.mean(dwi[90:95,100:105,0,0])/np.std(dwi[95:100,109:114,0,0]),3)  
SNR\_bold=np.round(np.mean(bold[67:72,124:129,20,0])/np.std(bold[89:94,133:138,20,0]),3)

#find and show the minimum SNR and maximum SNR  
SNR\_list=[SNR\_swi,SNR\_tof,SNR\_t1,SNR\_t2,SNR\_dwi,SNR\_bold]  
SNR\_list\_name=["SWI","TOF","T1","T2","DWI","BOLD"]  
print(f'Minimum SNR = {SNR\_list\_name[SNR\_list.index(np.min(SNR\_list))]}({np.min(SNR\_list)})')  
print(f'Maximum SNR = {SNR\_list\_name[SNR\_list.index(np.max(SNR\_list))]}({np.max(SNR\_list)})')

plt.show()

Part 3-b)

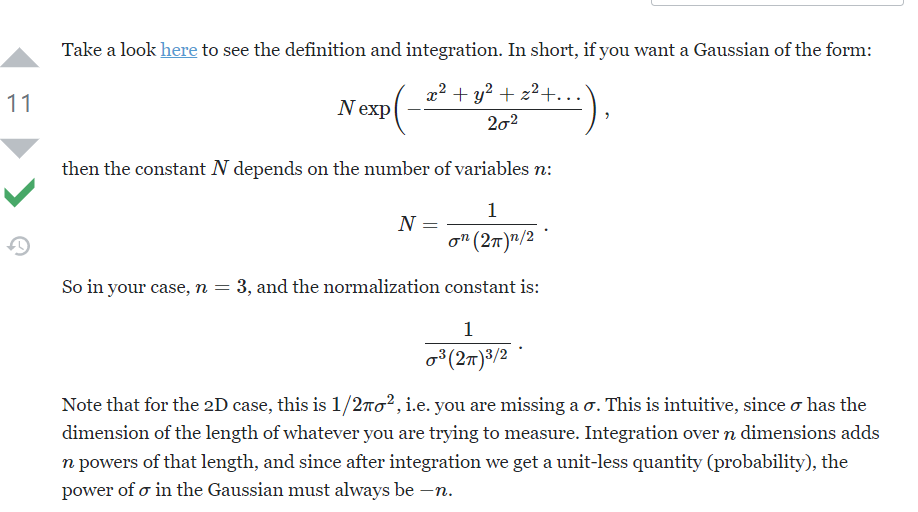


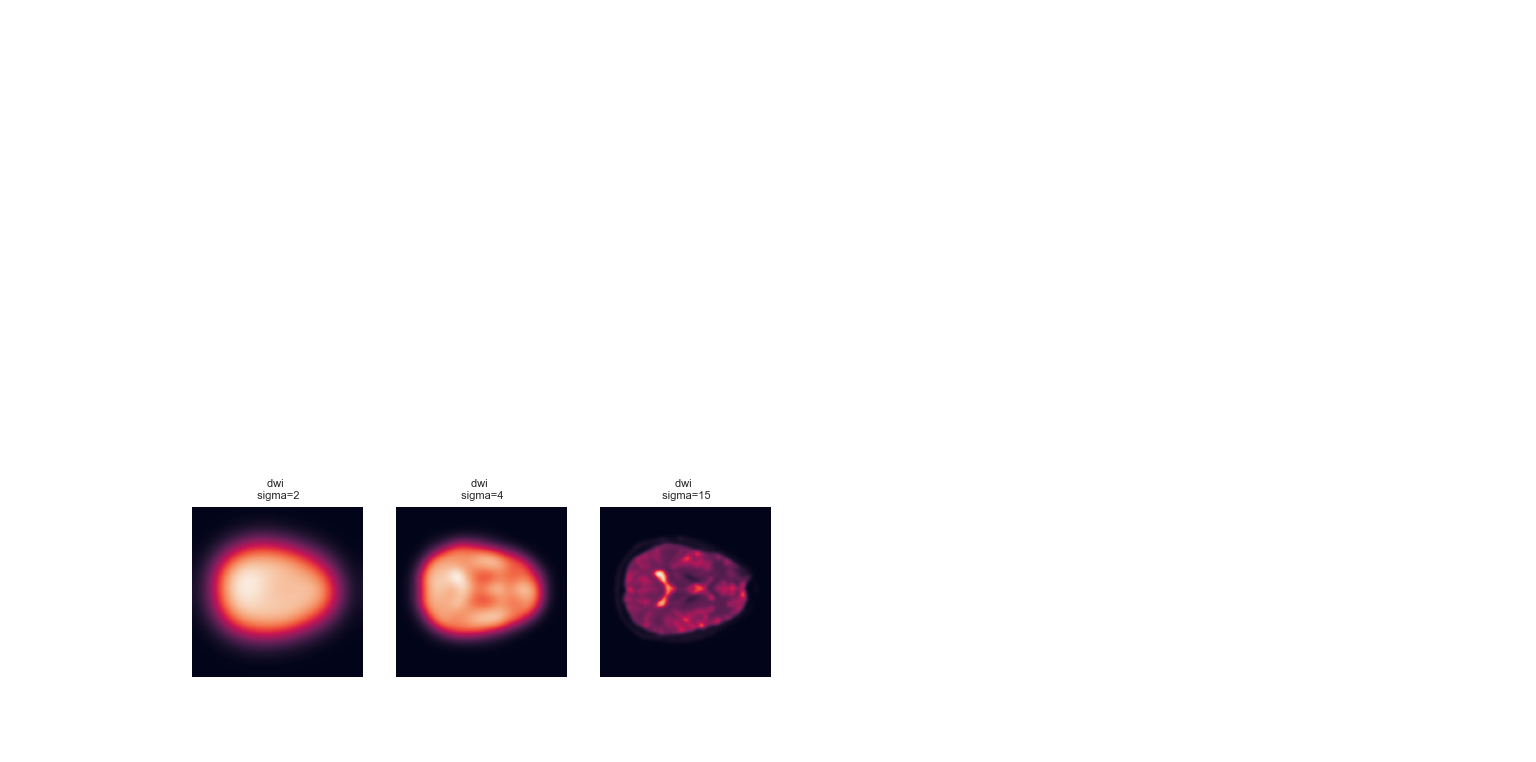
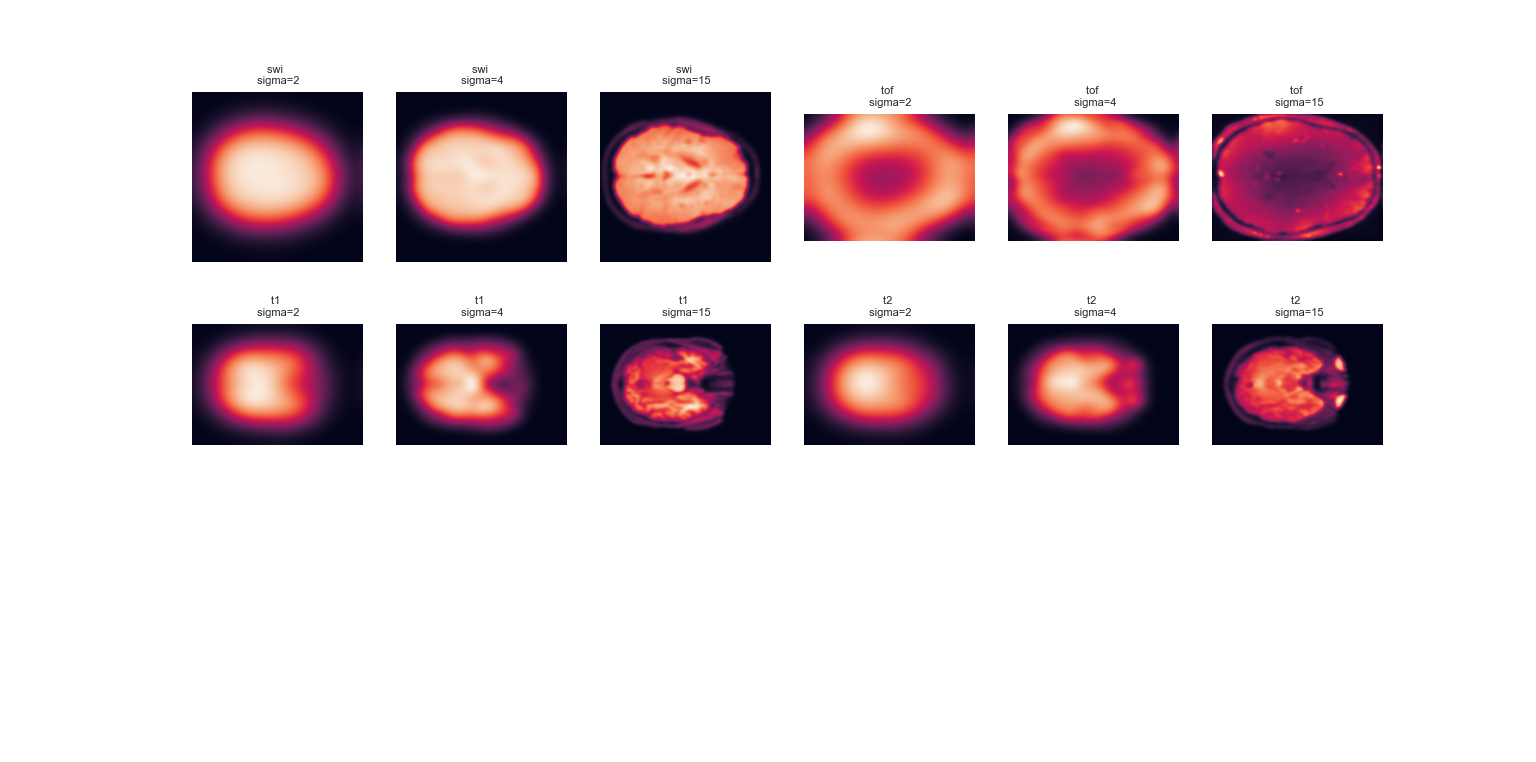
According to the following image, we detect the noise method of each image by their histogram:

# Question Part 3-b----------------------------------------------------------------------------  
#plot Histogram of noise of all image with their SNRs and names  
SNR\_swi=np.round(np.mean(swi[120:125,200:205,1])/np.std(swi[85:90,234:239,1]),3)  
SNR\_tof=np.round(np.mean(tof[156:161,326:331,1])/np.std(tof[10:15,362:367,1]),3)  
SNR\_t1=np.round(np.mean(t1[125:130,126:131,200])/np.std(t1[69:74,227:232,200]),3)  
SNR\_t2=np.round(np.mean(t2[104:109,208:213,200])/np.std(t2[101:106,235:240,200]),3)  
SNR\_dwi=np.round(np.mean(dwi[90:95,100:105,0,0])/np.std(dwi[95:100,109:114,0,0]),3)  
SNR\_bold=np.round(np.mean(bold[67:72,124:129,20,0])/np.std(bold[89:94,133:138,20,0]),3)  
  
plt.subplot(2,3,1)  
plt.hist(swi[85:90,234:239,1].ravel(),20,color='red')  
plt.title(f'SNR={SNR\_swi} \n Noise\_Histogram(swi)\n Noise Method(gaussian) ',fontsize=8)  
  
plt.subplot(2,3,2)  
plt.hist(tof[10:15,362:367,1].ravel(),20,color='red')  
plt.title(f'SNR={SNR\_tof}\n Noise\_Histogram(tof)\n Noise Method(gaussian)',fontsize=8)  
  
plt.subplot(2,3,3)  
plt.hist(t1[69:74,227:232,200].ravel(),20,color='red')  
plt.title(f'SNR={SNR\_t1}\n Noise\_Histogram(t1)\n Noise Method(gaussian)',fontsize=8)  
  
plt.subplot(2,3,4)  
plt.hist(t2[101:106,235:240,200].ravel(),20,color='red')  
plt.title(f'SNR={SNR\_t2}\n Noise\_Histogram(t2)\n Noise Method(gamma)',fontsize=8)  
  
plt.subplot(2,3,5)  
plt.hist(dwi[95:100,109:114,0,0].ravel(),20,color='red')  
plt.title(f'SNR={SNR\_dwi}\n Noise\_Histogram(dwi)\n Noise Method(exponential) ',fontsize=8)  
  
plt.subplot(2,3,6)  
plt.hist(bold[89:94,133:138,20,0].ravel(),20,color='red')  
plt.title(f'SNR={SNR\_bold}\n Noise\_Histogram(bold)\n Noise Method(Rayleigh)',fontsize=8)

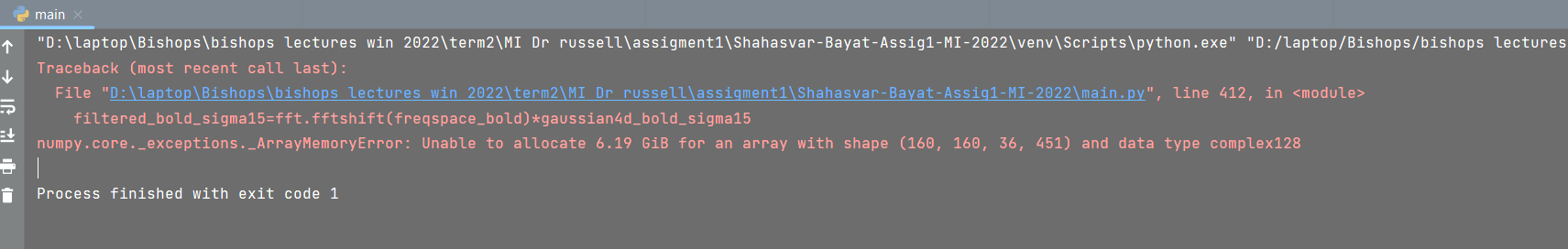
Part 4-a)

Calculate gaussian filter for noise reduction by bellow formula and then multiplying by frequency space representation of image(calculate furrier transform)



Due to the power of our computer, we had to run and capture part 4-a separately:

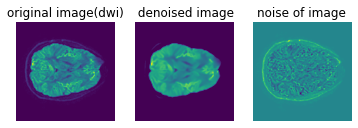
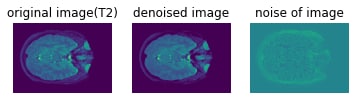
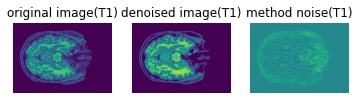
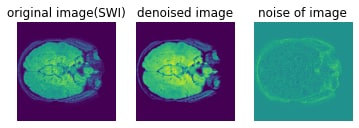
Notic: for ‘bold’ image,our system and also google colab couldn’t show the output(because of low capacity of RAM). we show the error below(but we are sure that our code like for dwi image is correct) :

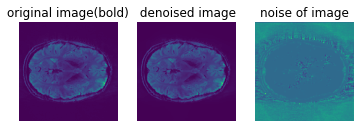


Code for 4-a)

#Question 4-a--------------------------------------------------  
#we use linear filtering(gaussian) for noise reduction,  
#that in this assigment we should multiplying in the  
#frequency domain that we gain by furrior transform  
import numpy.fft as fft  
  
#3 number of sigma  
sigma\_list=[2,4,15]  
  
#swi.............................................................  
#create frequency space by 3d furrior transform for each image that has equal shape with image  
freqspace\_swi=fft.fftn(swi)  
#create 3d grid to define qaussian filter over it  
xv,yv,zv=np.mgrid[-swi.shape[0]//2:swi.shape[0]//2,  
 -swi.shape[1]//2:swi.shape[1]//2,  
 -swi.shape[2]//2:swi.shape[2]//2]  
#define gaussian 3d function to aatenuate the high frequency anf accenuate the low frequency  
gaussian3d\_swi\_sigma2=(1/(sigma\_list[0]\*\*3\*(2\*np.pi)\*\*(3/2)))\*np.exp(-(xv\*\*2+yv\*\*2+zv\*\*2)/(2\*sigma\_list[0]\*\*2))  
gaussian3d\_swi\_sigma4=(1/(sigma\_list[1]\*\*3\*(2\*np.pi)\*\*(3/2)))\*np.exp(-(xv\*\*2+yv\*\*2+zv\*\*2)/(2\*sigma\_list[1]\*\*2))  
gaussian3d\_swi\_sigma15=(1/(sigma\_list[2]\*\*3\*(2\*np.pi)\*\*(3/2)))\*np.exp(-(xv\*\*2+yv\*\*2+zv\*\*2)/(2\*sigma\_list[2]\*\*2))  
#apply final step of filter on the image by multiplying qaussian function to frequency space  
filtered\_swi\_sigma2=fft.fftshift(freqspace\_swi)\*gaussian3d\_swi\_sigma2  
filtered\_swi\_sigma4=fft.fftshift(freqspace\_swi)\*gaussian3d\_swi\_sigma4  
filtered\_swi\_sigma15=fft.fftshift(freqspace\_swi)\*gaussian3d\_swi\_sigma15  
#invert the image to show the image by ifftn  
inv\_swi\_sigma2=np.abs(fft.ifftn(fft.fftshift(filtered\_swi\_sigma2)))  
inv\_swi\_sigma4=np.abs(fft.ifftn(fft.fftshift(filtered\_swi\_sigma4)))  
inv\_swi\_sigma15=np.abs(fft.ifftn(fft.fftshift(filtered\_swi\_sigma15)))  
#show the noise reduction of image  
plt.subplot(3,6,1)  
plt.imshow(inv\_swi\_sigma2[:,:,swi.shape[2]//2])  
plt.title(f'swi \n sigma=2', fontsize=8)  
plt.axis('off')  
plt.subplot(3,6,2)  
plt.imshow(inv\_swi\_sigma4[:,:,swi.shape[2]//2])  
plt.title(f'swi \n sigma=4', fontsize=8)  
plt.axis('off')  
plt.subplot(3,6,3)  
plt.imshow(inv\_swi\_sigma15[:,:,swi.shape[2]//2])  
plt.title(f'swi \n sigma=15', fontsize=8)  
plt.axis('off')  
  
#Tof..........................................  
#create frequency space by 3d furrior transform for each image that has equal shape with image  
freqspace\_tof=fft.fftn(tof)  
#create 3d grid to define qaussian filter over it  
xv,yv,zv=np.mgrid[-tof.shape[0]//2:tof.shape[0]//2,  
 -tof.shape[1]//2:tof.shape[1]//2,  
 -tof.shape[2]//2:tof.shape[2]//2]  
#define gaussian 3d function to aatenuate the high frequency anf accenuate the low frequency  
gaussian3d\_tof\_sigma2=(1/(sigma\_list[0]\*\*3\*(2\*np.pi)\*\*(3/2)))\*np.exp(-(xv\*\*2+yv\*\*2+zv\*\*2)/(2\*sigma\_list[0]\*\*2))  
gaussian3d\_tof\_sigma4=(1/(sigma\_list[1]\*\*3\*(2\*np.pi)\*\*(3/2)))\*np.exp(-(xv\*\*2+yv\*\*2+zv\*\*2)/(2\*sigma\_list[1]\*\*2))  
gaussian3d\_tof\_sigma15=(1/(sigma\_list[2]\*\*3\*(2\*np.pi)\*\*(3/2)))\*np.exp(-(xv\*\*2+yv\*\*2+zv\*\*2)/(2\*sigma\_list[2]\*\*2))  
#apply final step of filter on the image by multiplying qaussian function to frequency space  
filtered\_tof\_sigma2=fft.fftshift(freqspace\_tof)\*gaussian3d\_tof\_sigma2  
filtered\_tof\_sigma4=fft.fftshift(freqspace\_tof)\*gaussian3d\_tof\_sigma4  
filtered\_tof\_sigma15=fft.fftshift(freqspace\_tof)\*gaussian3d\_tof\_sigma15  
#invert the image to show the image by ifftn  
inv\_tof\_sigma2=np.abs(fft.ifftn(fft.fftshift(filtered\_tof\_sigma2)))  
inv\_tof\_sigma4=np.abs(fft.ifftn(fft.fftshift(filtered\_tof\_sigma4)))  
inv\_tof\_sigma15=np.abs(fft.ifftn(fft.fftshift(filtered\_tof\_sigma15)))  
#show the noise reduction of image  
plt.subplot(3,6,4)  
plt.imshow(inv\_tof\_sigma2[:,:,tof.shape[2]//2])  
plt.title(f'tof \n sigma=2', fontsize=8)  
plt.axis('off')  
plt.subplot(3,6,5)  
plt.imshow(inv\_tof\_sigma4[:,:,tof.shape[2]//2])  
plt.title(f'tof \n sigma=4', fontsize=8)  
plt.axis('off')  
plt.subplot(3,6,6)  
plt.imshow(inv\_tof\_sigma15[:,:,tof.shape[2]//2])  
plt.title(f'tof \n sigma=15', fontsize=8)  
plt.axis('off')  
  
#t1.......................................................  
#create frequency space by 3d furrior transform for each image that has equal shape with image  
freqspace\_t1=fft.fftn(t1)  
#create 3d grid to define qaussian filter over it  
xv,yv,zv=np.mgrid[-t1.shape[0]//2:t1.shape[0]//2,  
 -t1.shape[1]//2:t1.shape[1]//2,  
 -t1.shape[2]//2:t1.shape[2]//2]  
#define gaussian 3d function to aatenuate the high frequency anf accenuate the low frequency  
gaussian3d\_t1\_sigma2=(1/(sigma\_list[0]\*\*3\*(2\*np.pi)\*\*(3/2)))\*np.exp(-(xv\*\*2+yv\*\*2+zv\*\*2)/(2\*sigma\_list[0]\*\*2))  
gaussian3d\_t1\_sigma4=(1/(sigma\_list[1]\*\*3\*(2\*np.pi)\*\*(3/2)))\*np.exp(-(xv\*\*2+yv\*\*2+zv\*\*2)/(2\*sigma\_list[1]\*\*2))  
gaussian3d\_t1\_sigma15=(1/(sigma\_list[2]\*\*3\*(2\*np.pi)\*\*(3/2)))\*np.exp(-(xv\*\*2+yv\*\*2+zv\*\*2)/(2\*sigma\_list[2]\*\*2))  
#apply final step of filter on the image by multiplying qaussian function to frequency space  
filtered\_t1\_sigma2=fft.fftshift(freqspace\_t1)\*gaussian3d\_t1\_sigma2  
filtered\_t1\_sigma4=fft.fftshift(freqspace\_t1)\*gaussian3d\_t1\_sigma4  
filtered\_t1\_sigma15=fft.fftshift(freqspace\_t1)\*gaussian3d\_t1\_sigma15  
#invert the image to show the image by ifftn  
inv\_t1\_sigma2=np.abs(fft.ifftn(fft.fftshift(filtered\_t1\_sigma2)))  
inv\_t1\_sigma4=np.abs(fft.ifftn(fft.fftshift(filtered\_t1\_sigma4)))  
inv\_t1\_sigma15=np.abs(fft.ifftn(fft.fftshift(filtered\_t1\_sigma15)))  
#show the noise reduction of image  
plt.subplot(3,6,7)  
plt.imshow(inv\_t1\_sigma2[:,:,t1.shape[2]//2])  
plt.title(f't1 \n sigma=2', fontsize=8)  
plt.axis('off')  
plt.subplot(3,6,8)  
plt.imshow(inv\_t1\_sigma4[:,:,t1.shape[2]//2])  
plt.title(f't1 \n sigma=4', fontsize=8)  
plt.axis('off')  
plt.subplot(3,6,9)  
plt.imshow(inv\_t1\_sigma15[:,:,t1.shape[2]//2])  
plt.title(f't1 \n sigma=15', fontsize=8)  
plt.axis('off')  
  
#t2...........................................................  
#create frequency space by 3d furrior transform for each image that has equal shape with image  
freqspace\_t2=fft.fftn(t2)  
#create 3d grid to define qaussian filter over it  
xv,yv,zv=np.mgrid[-t2.shape[0]//2:t2.shape[0]//2,  
 -t2.shape[1]//2:t2.shape[1]//2,  
 -t2.shape[2]//2:t2.shape[2]//2]  
#define gaussian 3d function to aatenuate the high frequency anf accenuate the low frequency  
gaussian3d\_t2\_sigma2=(1/(sigma\_list[0]\*\*3\*(2\*np.pi)\*\*(3/2)))\*np.exp(-(xv\*\*2+yv\*\*2+zv\*\*2)/(2\*sigma\_list[0]\*\*2))  
gaussian3d\_t2\_sigma4=(1/(sigma\_list[1]\*\*3\*(2\*np.pi)\*\*(3/2)))\*np.exp(-(xv\*\*2+yv\*\*2+zv\*\*2)/(2\*sigma\_list[1]\*\*2))  
gaussian3d\_t2\_sigma15=(1/(sigma\_list[2]\*\*3\*(2\*np.pi)\*\*(3/2)))\*np.exp(-(xv\*\*2+yv\*\*2+zv\*\*2)/(2\*sigma\_list[2]\*\*2))  
#apply final step of filter on the image by multiplying qaussian function to frequency space  
filtered\_t2\_sigma2=fft.fftshift(freqspace\_t2)\*gaussian3d\_t2\_sigma2  
filtered\_t2\_sigma4=fft.fftshift(freqspace\_t2)\*gaussian3d\_t2\_sigma4  
filtered\_t2\_sigma15=fft.fftshift(freqspace\_t2)\*gaussian3d\_t2\_sigma15  
#invert the image to show the image by ifftn  
inv\_t2\_sigma2=np.abs(fft.ifftn(fft.fftshift(filtered\_t2\_sigma2)))  
inv\_t2\_sigma4=np.abs(fft.ifftn(fft.fftshift(filtered\_t2\_sigma4)))  
inv\_t2\_sigma15=np.abs(fft.ifftn(fft.fftshift(filtered\_t2\_sigma15)))  
#show the noise reduction of image  
plt.subplot(3,6,10)  
plt.imshow(inv\_t2\_sigma2[:,:,t2.shape[2]//2])  
plt.title(f't2 \n sigma=2', fontsize=8)  
plt.axis('off')  
plt.subplot(3,6,11)  
plt.imshow(inv\_t2\_sigma4[:,:,t2.shape[2]//2])  
plt.title(f't2 \n sigma=4', fontsize=8)  
plt.axis('off')  
plt.subplot(3,6,12)  
plt.imshow(inv\_t2\_sigma15[:,:,t2.shape[2]//2])  
plt.title(f't2 \n sigma=15', fontsize=8)  
plt.axis('off')  
  
#dwi.....................................................  
#create frequency space by 4d furrior transform for each image that has equal shape with image  
freqspace\_dwi=fft.fftn(dwi)  
#create 3d grid to define qaussian filter over it  
xv,yv,zv,vv=np.mgrid[-dwi.shape[0]//2:dwi.shape[0]//2,  
 -dwi.shape[1]//2:dwi.shape[1]//2,  
 -dwi.shape[2]//2:dwi.shape[2]//2,  
 -dwi.shape[3]//2:dwi.shape[3]//2]  
#define gaussian 4d function to aatenuate the high frequency anf accenuate the low frequency  
gaussian4d\_dwi\_sigma2=(1/(sigma\_list[0]\*\*4\*(2\*np.pi)\*\*(4/2)))\*np.exp(-(xv\*\*2+yv\*\*2+zv\*\*2+vv\*\*2)/(2\*sigma\_list[0]\*\*2))  
gaussian4d\_dwi\_sigma4=(1/(sigma\_list[1]\*\*4\*(2\*np.pi)\*\*(4/2)))\*np.exp(-(xv\*\*2+yv\*\*2+zv\*\*2+vv\*\*2)/(2\*sigma\_list[1]\*\*2))  
gaussian4d\_dwi\_sigma15=(1/(sigma\_list[2]\*\*4\*(2\*np.pi)\*\*(4/2)))\*np.exp(-(xv\*\*2+yv\*\*2+zv\*\*2+vv\*\*2)/(2\*sigma\_list[2]\*\*2))  
#apply final step of filter on the image by multiplying qaussian function to frequency space  
filtered\_dwi\_sigma2=fft.fftshift(freqspace\_dwi)\*gaussian4d\_dwi\_sigma2  
filtered\_dwi\_sigma4=fft.fftshift(freqspace\_dwi)\*gaussian4d\_dwi\_sigma4  
filtered\_dwi\_sigma15=fft.fftshift(freqspace\_dwi)\*gaussian4d\_dwi\_sigma15  
#invert the image to show the image by ifftn  
inv\_dwi\_sigma2=np.abs(fft.ifftn(fft.fftshift(filtered\_dwi\_sigma2)))  
inv\_dwi\_sigma4=np.abs(fft.ifftn(fft.fftshift(filtered\_dwi\_sigma4)))  
inv\_dwi\_sigma15=np.abs(fft.ifftn(fft.fftshift(filtered\_dwi\_sigma15)))  
#show the noise reduction of image  
plt.subplot(3,6,13)  
plt.imshow(inv\_dwi\_sigma2[:,:,dwi.shape[2]//2,0])  
plt.title(f'dwi \n sigma=2', fontsize=8)  
plt.axis('off')  
plt.subplot(3,6,14)  
plt.imshow(inv\_dwi\_sigma4[:,:,dwi.shape[2]//2,0])  
plt.title(f'dwi \n sigma=4', fontsize=8)  
plt.axis('off')  
plt.subplot(3,6,15)  
plt.imshow(inv\_dwi\_sigma15[:,:,dwi.shape[2]//2,0])  
plt.title(f'dwi \n sigma=15', fontsize=8)  
plt.axis('off')  
  
#bold.....................................................  
#create frequency space by 3d furrior transform for each image that has equal shape with image  
freqspace\_bold=fft.fftn(bold)  
#create 4d grid to define qaussian filter over it  
xv,yv,zv,vv=np.mgrid[-bold.shape[0]//2:bold.shape[0]//2,  
 -bold.shape[1]//2:bold.shape[1]//2,  
 -bold.shape[2]//2:bold.shape[2]//2,  
 -bold.shape[3]//2:bold.shape[3]//2]  
#define gaussian 4d function to aatenuate the high frequency anf accenuate the low frequency  
gaussian4d\_bold\_sigma2=(1/(sigma\_list[0]\*\*4\*(2\*np.pi)\*\*(4/2)))\*np.exp(-(xv\*\*2+yv\*\*2+zv\*\*2+vv\*\*2)/(2\*sigma\_list[0]\*\*2))  
gaussian4d\_bold\_sigma4=(1/(sigma\_list[1]\*\*4\*(2\*np.pi)\*\*(4/2)))\*np.exp(-(xv\*\*2+yv\*\*2+zv\*\*2+vv\*\*2)/(2\*sigma\_list[1]\*\*2))  
gaussian4d\_bold\_sigma15=(1/(sigma\_list[2]\*\*4\*(2\*np.pi)\*\*(4/2)))\*np.exp(-(xv\*\*2+yv\*\*2+zv\*\*2+vv\*\*2)/(2\*sigma\_list[2]\*\*2))  
#apply final step of filter on the image by multiplying qaussian function to frequency space  
filtered\_bold\_sigma2=fft.fftshift(freqspace\_bold)\*gaussian4d\_bold\_sigma2  
filtered\_bold\_sigma4=fft.fftshift(freqspace\_bold)\*gaussian4d\_bold\_sigma4  
filtered\_bold\_sigma15=fft.fftshift(freqspace\_bold)\*gaussian4d\_bold\_sigma15  
#invert the image to show the image by ifftn  
inv\_bold\_sigma2=np.abs(fft.ifftn(fft.fftshift(filtered\_bold\_sigma2)))  
inv\_bold\_sigma4=np.abs(fft.ifftn(fft.fftshift(filtered\_bold\_sigma4)))  
inv\_bold\_sigma15=np.abs(fft.ifftn(fft.fftshift(filtered\_bold\_sigma15)))  
#show the noise reduction of image  
plt.subplot(3,6,16)  
plt.imshow(inv\_bold\_sigma2[:,:,bold.shape[2]//2,0])  
plt.title(f'bold \n sigma=2', fontsize=8)  
plt.axis('off')  
plt.subplot(3,6,17)  
plt.imshow(inv\_bold\_sigma4[:,:,bold.shape[2]//2,0])  
plt.title(f'bold \n sigma=4', fontsize=8)  
plt.axis('off')  
plt.subplot(3,6,18)  
plt.imshow(inv\_bold\_sigma15[:,:,bold.shape[2]//2,0])  
plt.title(f'bold \n sigma=15', fontsize=8)  
plt.axis('off')

Part 4-b)





# Question 4\_b---------------------------------------------------  
import dipy.denoise.nlmeans as nlmeans  
  
# swi................................  
#calculate the denoised image of original image  
den\_swi = nlmeans.nlmeans(swi,10)  
  
#show the original image with noise  
plt.subplot(1,3,1)  
plt.imshow(swi[:,:,200])  
plt.title("original image(swi)")  
plt.axis("off")  
  
#show the denoised image  
plt.subplot(1,3,2)  
plt.imshow(den\_swi[:,:,200])  
plt.title("denoised image(swi)")  
plt.axis("off")  
  
#calculate and show the noise of image by subtracing the denoise image of original image  
plt.subplot(1,3,3)  
plt.imshow(swi[:,:200]-den\_swi[:,:,200])  
plt.title("noise of image(swi)")  
plt.axis("off")

# tof..............................  
den\_tof = nlmeans.nlmeans(tof,10)  
plt.subplot(1,3,1)  
plt.imshow(tof[:,:,200])  
plt.title("original image(tof)")  
plt.axis("off")  
  
plt.subplot(1,3,2)  
plt.imshow(den\_tof[:,:,200])  
plt.title("denoised image(tof)")  
plt.axis("off")  
  
plt.subplot(1,3,3)  
plt.imshow(tof[:,:200]-den\_tof[:,:,200])  
plt.title("noise of image(tof)")  
plt.axis("off")  
  
# t1..............................  
den\_t1 = nlmeans.nlmeans(t1,10)  
plt.subplot(1,3,1)  
plt.imshow(t1[:,:,200])  
plt.title("original image(t1)")  
plt.axis("off")  
  
plt.subplot(1,3,2)  
plt.imshow(den\_t1[:,:,200])  
plt.title("denoised image(t1)")  
plt.axis("off")  
  
plt.subplot(1,3,3)  
plt.imshow(t1[:,:200]-den\_t1[:,:,200])  
plt.title("noise of image(t1)")  
plt.axis("off")  
  
# t2..............................  
den\_t2 = nlmeans.nlmeans(t2,10)  
plt.subplot(1,3,1)  
plt.imshow(t2[:,:,200])  
plt.title("original image(t2)")  
plt.axis("off")  
  
plt.subplot(1,3,2)  
plt.imshow(den\_t2[:,:,200])  
plt.title("denoised image(t2)")  
plt.axis("off")  
  
plt.subplot(1,3,3)  
plt.imshow(t2[:,:200]-den\_t2[:,:,200])  
plt.title("noise of image(t2)")  
plt.axis("off")  
  
# dwi..............................  
den\_dwi = nlmeans.nlmeans(dwi,10)  
plt.subplot(1,3,1)

plt.imshow(dwi[:,:,35,10])

plt.title("original image(dwi)")

plt.axis("off")

plt.subplot(1,3,2)

plt.imshow(den\_dwi[:,:,35,10])

plt.title(" denoised image")

plt.axis("off")

plt.subplot(1,3,3)

plt.imshow(dwi[:,:,35,10]-den\_dwi[:,:,35,10])

plt.title("noise of image")

plt.axis("off")

# bold..............................  
den\_bold = nlmeans.nlmeans(bold,10)  
plt.subplot(1,3,1)  
plt.imshow(bold[:,:,16,10])  
plt.title("original image(bold)")  
plt.axis("off")  
  
plt.subplot(1,3,2)  
plt.imshow(den\_bold[:,:,16,10])  
plt.title("denoised image(bold)")  
plt.axis("off")  
  
plt.subplot(1,3,3)  
plt.imshow(bold[:,:,16,10]-den\_bold[:,:,16,10])  
plt.title("noise of image(bold)")  
plt.axis("off")

plt.show()