Introduction: First I want to thank Professor Russel butler for such a unique challenge. I learned a lot in the process and was able to master NumPy libraries and process 3d images.

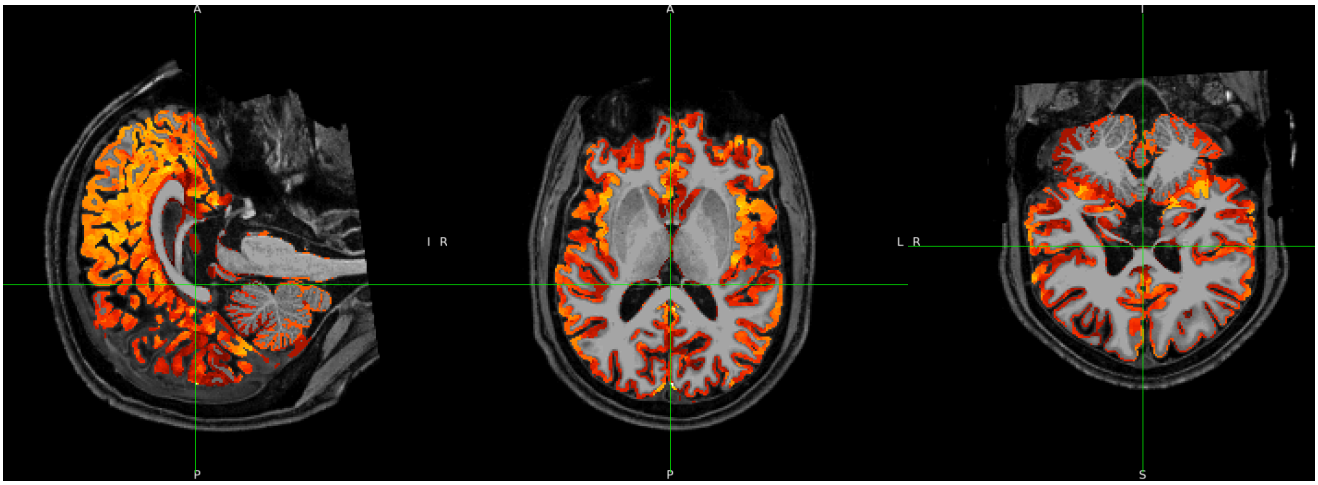
Challenge: To get the thickness map (similar to thickness_map_subject_01.nii.gz) from raw_t1_subject_02.nii.gz using minimal libraries like NumPy. A custom algorithm must be developed to estimate the thickness between grey matter and white matter segmentation for cortical thickness.

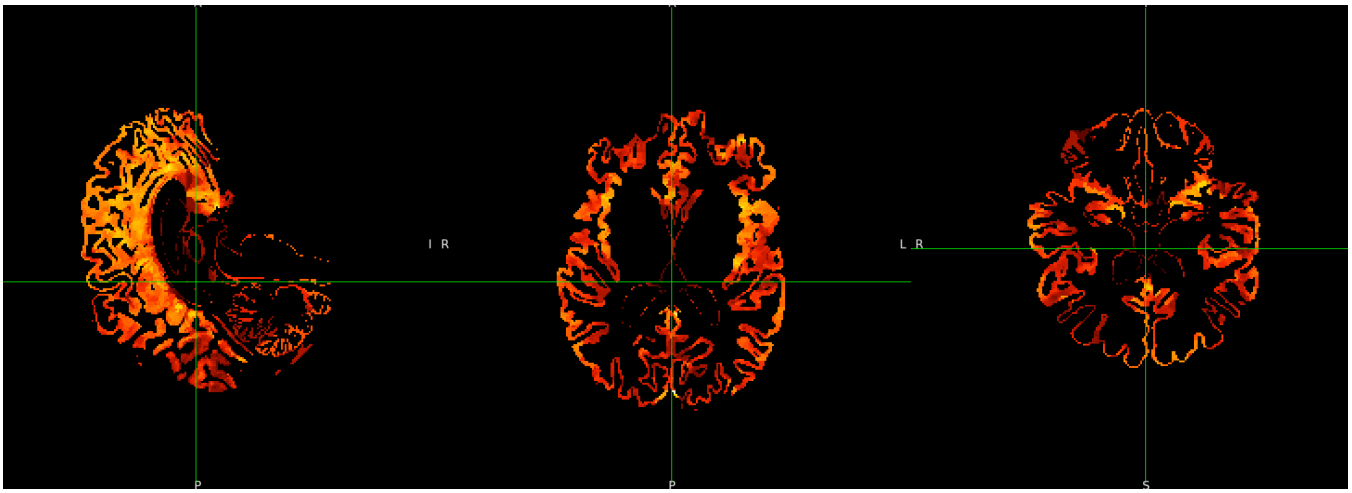
Index:

Our Results: Below are the results and corresponding file names

a) Without Tuning

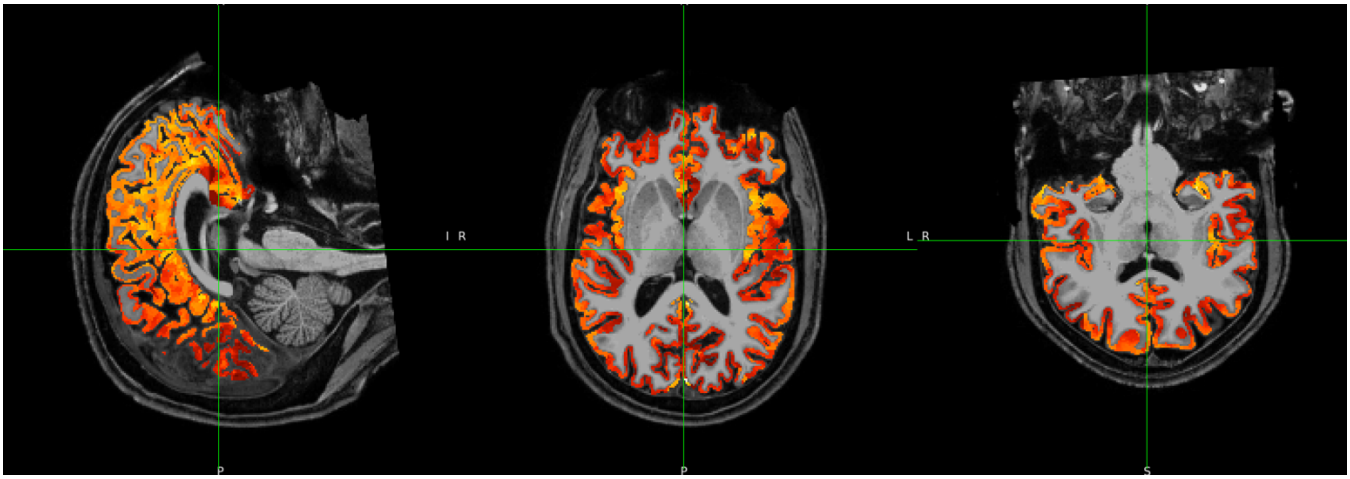
Filename: **Thickness_map_withCSF.nii**

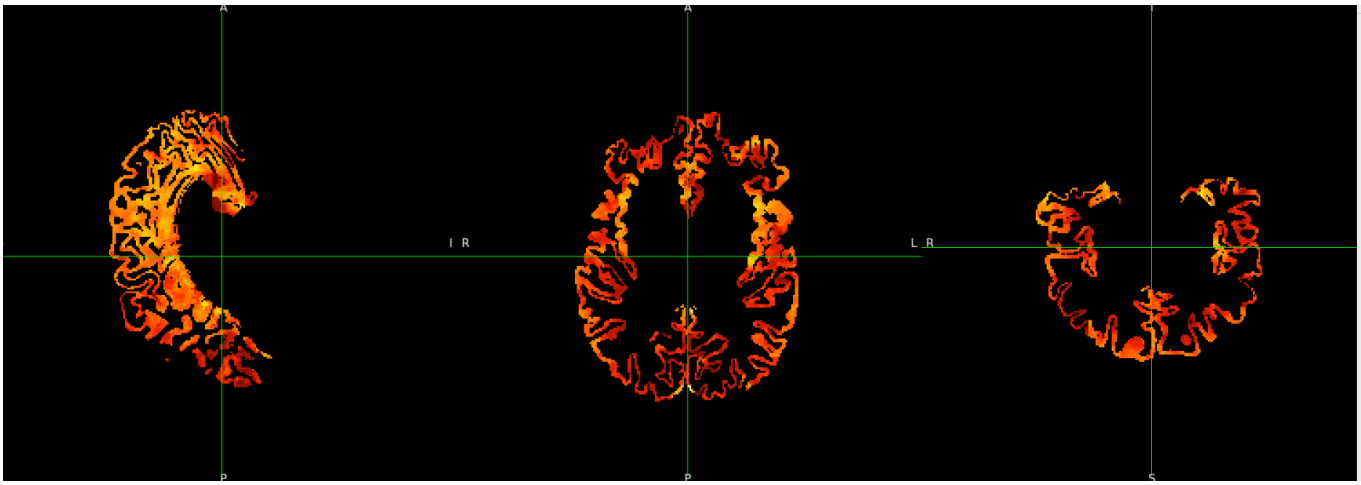




b) After tuning with the original thickness map to remove CSF

Filename: **Thickness_map_tuned_withoutCSF.nii**





Concepts and tools used:

- Denoising Image (used dipy.deniose.nlmeans)
- Dilation
- Erosion
- KNN segmentation to separate white and grey matter
- Contours (dilated image – original image)
- Finding nearest points on 3d space using the below formulae

$$\begin{cases} x = \{x_0, x_1 ... x_n\} \\ y = \{y_0, y_1 ... y_n\} \\ z = \{z_0, z_1 ... z_n\} \end{cases}$$

$$P = [x', y', z']$$

$$d = \sqrt{(x - x')^2 + (y - y')^2 + (z - z')^2}$$

$$\min(d)$$

- Tqdm library to visualize the time for a particular code
- Nilabel library to import and export the 3d images
- Creating a robust brain mask is impossible in a short period of time, so we tuned out our custom brain mask with a brain mask from nilearn. masking.compute_brain_mask

Procedure Followed:

- Imported required libraries

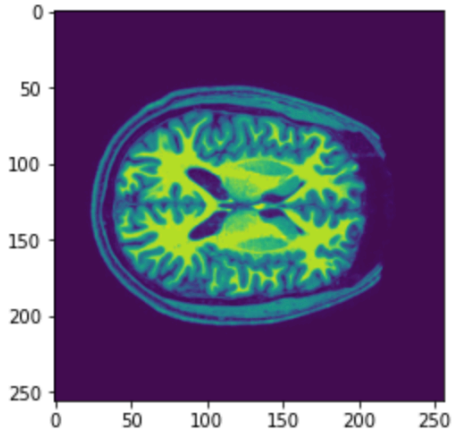
```
import matplotlib
from matplotlib import pylab as plt
import nibabel as nib
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from skimage.io import imread, imshow
from skimage.color import rgb2gray
from skimage.morphology import (erosion, dilation, closing, opening,
                                area_closing, area_opening)
from skimage.measure import label, regionprops, regionprops_table
from nipype.interfaces.ants.segmentation import BrainExtraction
import nibabel as nib
from nilearn import plotting
```

- Loaded the ‘raw_t1_subject_02.nii.gz’ image using the nibabel function

```
img_data1 = nib.load('raw_t1_subject_02.nii.gz').get_fdata()
img1=nib.load('raw_t1_subject_02.nii.gz')
```

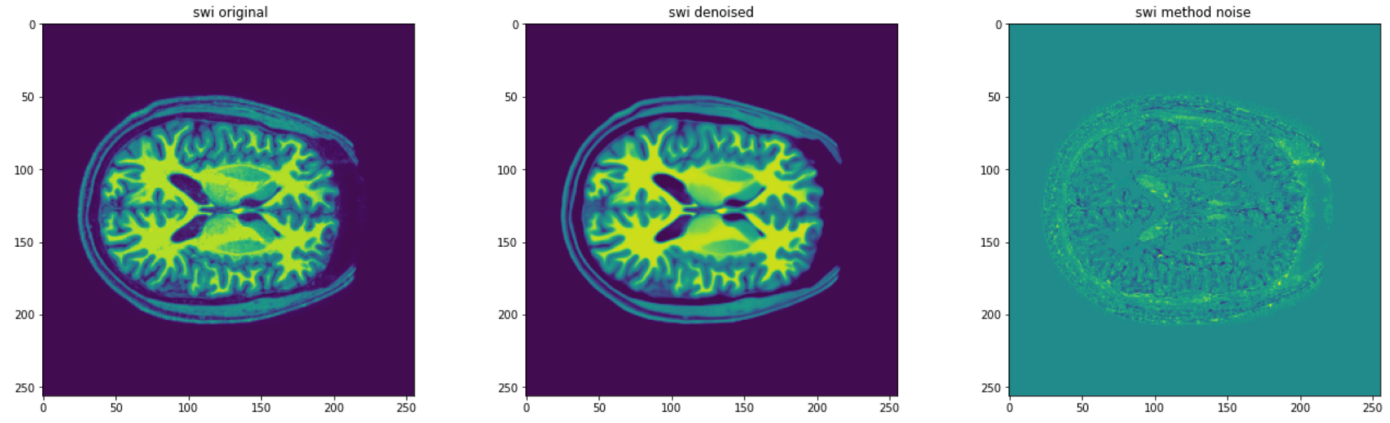
```
plt.imshow(img1.get_fdata()[:,120,:])
```

```
<matplotlib.image.AxesImage at 0x7fa172dd0f10>
```



- As you can see, the image has so much of noise. We used denoise. nlmeans to remove the noise (Taught in medical imaging class)

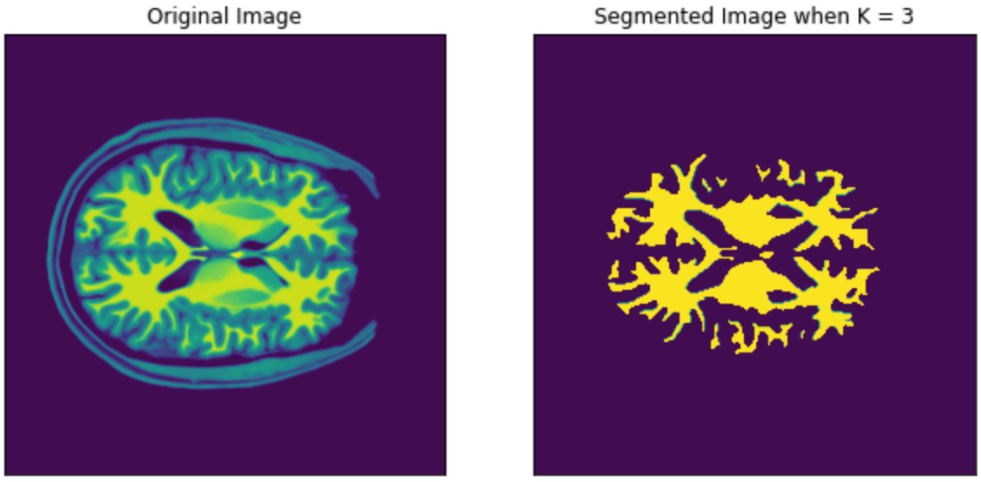
```
import dipy.denoise.nlmeans as nimeans
den_img_data1 = nimeans.nlmeans(img_data1,8)

plt.figure(figsize=(10,10))
plt.subplot (1,3,1); plt. imshow(img_data1[:, 120, :]); plt.title(' swi original')
plt.subplot (1,3,2); plt.imshow(den_img_data1[:, 120, :]); plt.title('swi denoised')
plt.subplot (1,3,3); plt.imshow(img_data1[:, 120, :]-den_img_data1[:, 120, :]); plt.title('swi method noise ')

plt.subplots_adjust(left=0.3,
                    bottom=0.3,
                    right=2.0,
                    top=0.9,
                    wspace=0.3,
                    hspace=0.3)
plt.show()
```



- Our next task is to create a brain mask to remove the skull part. We segmented the white matter from the image and dilated it first.

```
import cv2
def KNN_segmentation(img,K):
    original_image = img
    original_image = original_image.astype('uint8')
    vectorized = original_image.reshape((-1))
    vectorized = np.float32(vectorized)
    criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 10, 1.0)
    attempts=50
    ret,label,center=cv2.kmeans(vectorized,K,None,criteria,attempts,cv2.KMEANS_PP_CENTERS)
    center = np.uint8(center)
    res = center[label.flatten()]
    result_image = res.reshape((original_image.shape))
    print(np.unique(result_image))
    return result_image
```

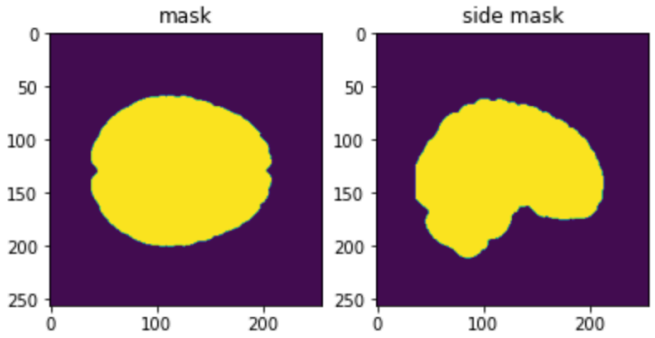
```
i = 120
K=3
white = KNN_segmentation(original_image,5)
image1 = np.where(white !=102, 0, white)
figure_size = 10
plt.figure(figsize=(figure_size,figure_size))
plt.subplot(1,2,1),plt.imshow(original_image[:,i,:])
plt.title('Original Image'), plt.xticks([]), plt.yticks([])
plt.subplot(1,2,2),plt.imshow(image1[:,i,:],vmax=30)
plt.title('Segmented Image when K = %i' % K), plt.xticks([]), plt.yticks([])
plt.show()
```



- Now the segmented white matter is dilated and then tuned with the library brain mask for making it robust and accurate

```
from skimage.morphology import square
from skimage import morphology
import skimage
o = image1.copy()
mask_dilation = np.zeros((5,5,5),dtype=int)
mask_dilation[mask_dilation == 0] =1
dil = skimage.morphology.dilation(o,mask_dilation)
masked_img = compute_brain_mask(o,mask_type='whole-brain')
masked_img = masked_img.get_fdata()
# removes the pixels that crossed the brain
masked_img = dil * masked_img
i=120
plt.figure(figsize=(10,10))
plt.subplot(1,3,1);plt.imshow(masked_img[:,i,:]);plt.title("mask")
plt.subplot(1,3,2);plt.imshow(masked_img[i,:,:]);plt.title("side mask")
```

```
Text(0.5, 1.0, 'side mask')
```

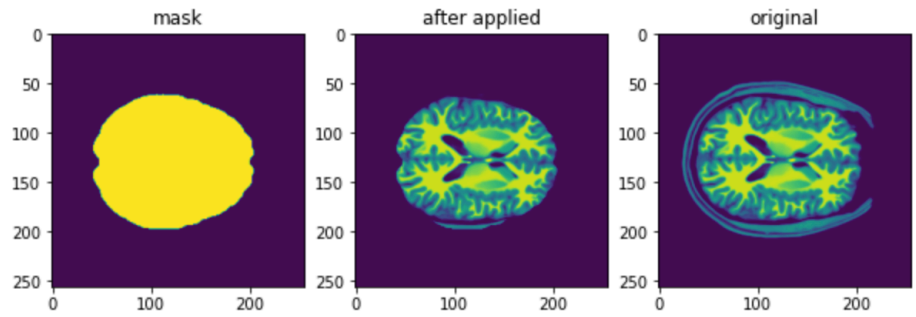


- Applied erosion to the brain mask as the mask is bigger than required

```
matrix_erosion = np.zeros((5,5,5))
matrix_erosion[matrix_erosion==0] = 1
dil = skimage.morphology.erosion(masked_img,matrix_erosion)
```

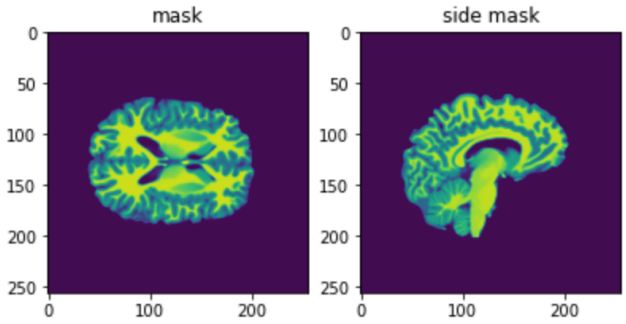
```
i=120
plt.figure(figsize=(10,10))
plt.subplot(1,3,1);plt.imshow(dil[:,i,:]);plt.title("mask")
plt.subplot(1,3,2);plt.imshow(dil[:,i,:]*original_image[:,i,:]);plt.title("after applied")
plt.subplot(1,3,3);plt.imshow(original_image[:,i,:]);plt.title("original")
```

```
Text(0.5, 1.0, 'original')
```



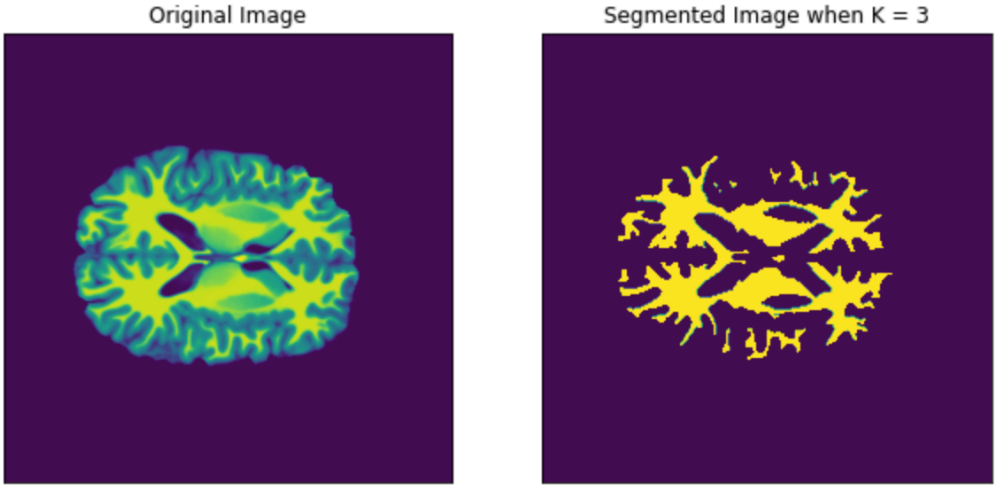
```
i=120
final_skull_removed = dil*original_image
plt.figure(figsize=(10,10))
plt.subplot(1,3,1);plt.imshow(final_skull_removed[:,i,:]);plt.title("mask")
plt.subplot(1,3,2);plt.imshow(final_skull_removed[i,:,:]);plt.title("side mask")
```

```
Text(0.5, 1.0, 'side mask')
```

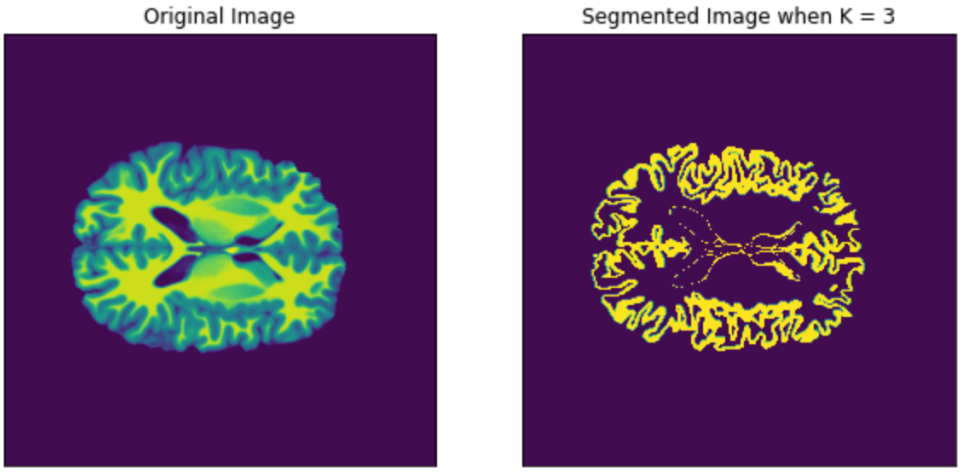


- Now we again performed KNN segmentation to get the accurate white and grey matter. We saved both the segmented images as **'white matter.nii'** and **'Greymatter.nii'**

```
i = 120
K=3
result_image = KNN_segmentation(final_skull_removed,5)
image1 = np.where(result_image !=104, 0, result_image)
figure_size = 10
plt.figure(figsize=(figure_size,figure_size))
plt.subplot(1,2,1),plt.imshow(final_skull_removed[:,i,:])
plt.title('Original Image'), plt.xticks([]), plt.yticks([])
plt.subplot(1,2,2),plt.imshow(image1[:,i,:],vmax=30)
plt.title('Segmented Image when K = %i' % K), plt.xticks([]), plt.yticks([])
plt.show()
```



```
i = 120
K=3
image2 = np.where(result_image !=56, 0, result_image)
figure_size = 10
plt.figure(figsize=(figure_size,figure_size))
plt.subplot(1,2,1),plt.imshow(final_skull_removed[:,i,:])
plt.title('Original Image'), plt.xticks([]), plt.yticks([])
plt.subplot(1,2,2),plt.imshow(image2[:,i,:],vmax=30)
plt.title('Segmented Image when K = %i' % K), plt.xticks([]), plt.yticks([])
plt.show()
```



- Saving the segmented white matter and grey matter segmentations

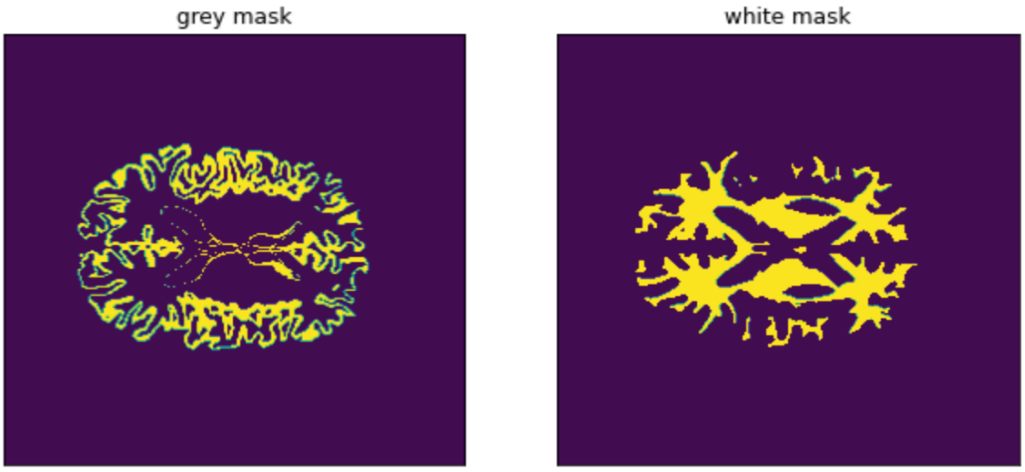
```
grey_matter_img = nib.Nifti1Image(image2, img1.affine)
white_matter_img = nib.Nifti1Image(image1, img1.affine)
nib.save(grey_matter_img, 'greymatter.nii')
nib.save(white_matter_img , 'whitematter.nii')
```

- We took a small break and again got back to work

- We reloaded the images that we saved

```
img_grey_mask1 = nib.load('greymatter.nii').get_fdata()
img_grey_mask = nib.load('greymatter.nii')
img_white_mask1 = nib.load('whitematter.nii').get_fdata()
img_white_mask = nib.load('whitematter.nii')
```

```
%matplotlib inline
figure_size = 10
i=120
plt.figure(figsize=(figure_size,figure_size))
plt.subplot(1,2,1),plt.imshow(img_grey_mask1[:,i,:])
plt.title('grey mask'), plt.xticks([]), plt.yticks([])
plt.subplot(1,2,2),plt.imshow(img_white_mask1[:,i,:],vmax=30)
plt.title('white mask'), plt.xticks([]), plt.yticks([])
plt.show()
```

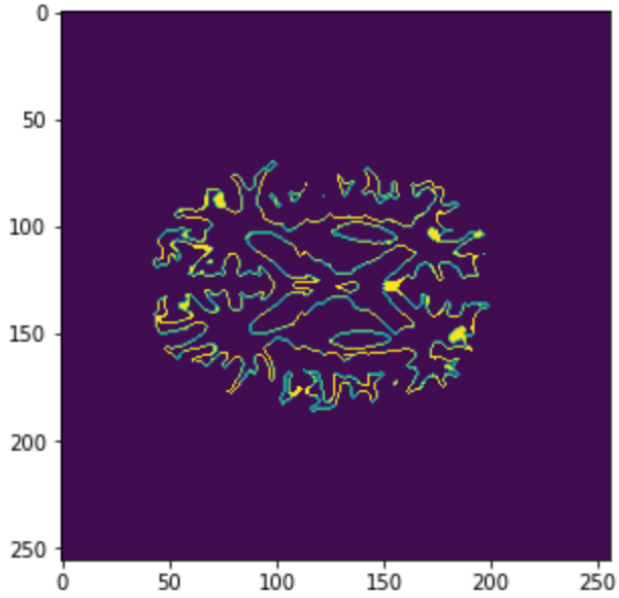


Algorithm Explained:

- Now we need to develop an algorithm that calculates the thickness. At first, we calculated the contour of the white mask. We used dilated image – the original image to get the contour

```
def dilation_img(img,iters):
    dil=img
    for i in range(iters):
        dil = dilation(dil)
    return dil

figure_size = 5
plt.figure(figsize=(figure_size,figure_size))
dilated_white_mask = dilation_img(img_white_mask1,1)
contour_white_mask = dilated_white_mask-img_white_mask1
plt.imshow(contour_white_mask[:,120,:])
```

```
<matplotlib.image.AxesImage at 0x7fc259d3fc50>
```



- Now we convert the masks into binary to keep all masks at the same pace

```
grey_mask = img_grey_mask1
white_mask = img_white_mask1
contour_white_mask = contour_white_mask

grey_mask = np.array(np.where(grey_mask==56., 1., grey_mask),dtype='uint8')
white_mask  = np.array(np.where(white_mask ==104., 1., white_mask ),dtype='uint8')
contour_white_mask = np.array(np.where(contour_white_mask==104., 1., contour_white_mask),dtype='uint8')
```

- We get the locations of the white contour mask and grey mask where the pixel is 1. We keep the results in an array

```
locations_white_contour_mask = []
for x,x1 in enumerate(contour_white_mask):
    for y,y1 in enumerate(x1):
        for z,z1 in enumerate(x1):
            if contour_white_mask[x][y][z]==1:
                locations_white_contour_mask.append([x,y,z])
```

```
locations_grey_mask = []
for x,x1 in enumerate(grey_mask):
    for y,y1 in enumerate(x1):
        for z,z1 in enumerate(x1):
            if grey_mask[x][y][z]==1:
                locations_grey_mask.append([x,y,z])
```

- Now we use the below formulae to get the distances and nearest points from the white contour mask to grey mask. We keep all the distance values on the contour of the white mask. Later we will use that contour white mask (Which has distance values on the border) with the grey matter mask and fill all the points on the grey matter mask with the nearest point distance values of the contour white mask (We discussed with Dr.Russel buttler about this method)

$$\begin{cases} x = \{x_0, x_1 ... x_n\} \\ y = \{y_0, y_1 ... y_n\} \\ z = \{z_0, z_1 ... z_n\} \end{cases}$$

$$P = [x', y', z']$$

$$d = \sqrt{(x - x')^2 + (y - y')^2 + (z - z')^2}$$

$$\min(d)$$

```
%matplotlib inline

x = np.array([locations_grey_mask[i][0] for i in range(len(locations_grey_mask))])
y= np.array([locations_grey_mask[i][1] for i in range(len(locations_grey_mask))])
z= np.array([locations_grey_mask[i][2] for i in range(len(locations_grey_mask))])


P = (x1, y1, z1)
def distance_3d(x, y, z, x0, y0, z0):
    dx = x - x0
    dy = y - y0
    dz = z - z0
    d = np.sqrt(dx**2 + dy**2 + dz**2)
#     print(len(dx))
    return d

def min_distance(x, y, z, P, precision=5):
    # compute distance
    d = distance_3d(x, y, z, P[0], P[1], P[2])
    d = np.round(d, precision)
    # find the minima
    glob_min_idxs = np.argwhere(d==np.min(d)).ravel()
    return glob_min_idxs, d
```

```
from tqdm import tqdm
contour_white_copy = np.zeros((256,256,256),dtype='uint8')
distance_white_mask_loc={}
for i in tqdm(locations_white_contour_mask):
    P = (i[0],i[1],i[2])
    d = distance_3d(x, y, z, P[0],P[1],P[2])
    d = np.round(d, 5)
    glob_min_idxs = np.argwhere(d==np.min(d)).ravel()
    distance_white_mask_loc[P] = min(d)
    contour_white_copy[P[0]][P[1]][P[2]]=min(d)
```

```
100%|██████████| 166855/166855 [5:46:32<00:00,  8.02it/s]
```

- As the above is a very tedious task (Finding nearest points, calculating distances, and creating the contour white mask image with distance values on the border) It took 5 hours to run. It can be reduced to 2 hours if you remove creating the dictionary. We needed it because we can analyze the dictionary if the results are not good.

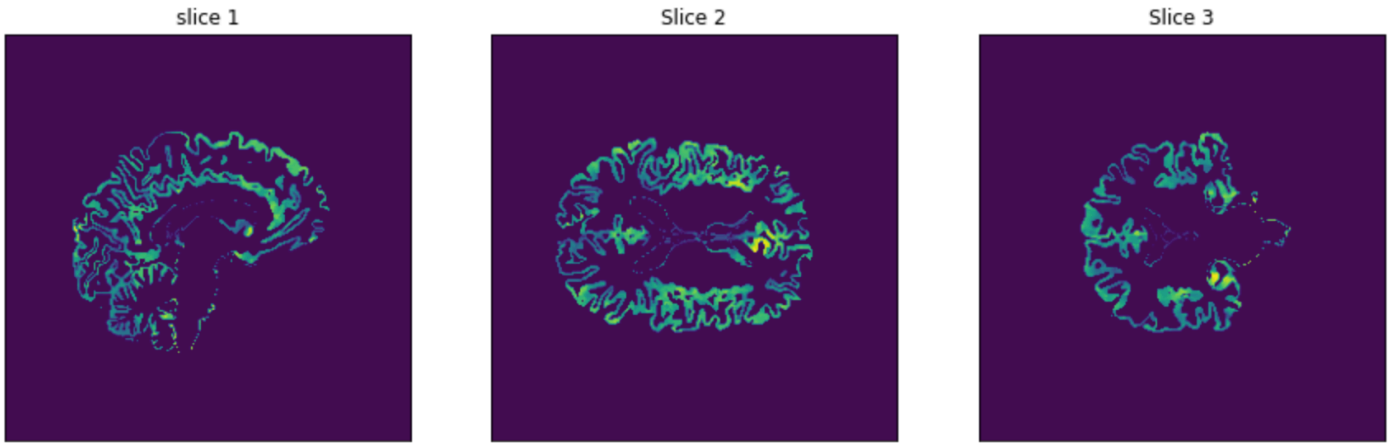
- Now we need to replace all the grey matter mask pixels with distance values from the white contour mask. It took 2 hours to complete the process

```
grey_copy = np.zeros((256,256,256),dtype='uint8')
distance_grey_mask_loc={}
for i in tqdm(locations_grey_mask):
    P = (i[0],i[1],i[2])
    min_idx, d = min_distance(x, y, z, P)
    nearest_point_on_grey = (list(x[min_idx])[0],list(y[min_idx])[0],list(z[min_idx])[0])

    distance_grey_mask_loc[P] = contour_white_copy[nearest_point_on_grey[0]][nearest_point_on_grey[1]][nearest_point_
    grey_copy[P[0]][P[1]][P[2]]=contour_white_copy[nearest_point_on_grey[0]][nearest_point_on_grey[1]][nearest_point_
```

```
100%|██████████| 446160/446160 [51:02<00:00, 145.67it/s]
```

- The Grey_copy is our final cortical thickness image. Let's see the results with excitement.

```
figure_size = 15
plt.figure(figsize=(figure_size,figure_size))
plt.subplot(1,3,1),plt.imshow(grey_copy[120,:,:])
plt.title('slice 1'), plt.xticks([]), plt.yticks([])
plt.subplot(1,3,2),plt.imshow(grey_copy[:,120,:])
plt.title('Slice 2'), plt.xticks([]), plt.yticks([])
plt.subplot(1,3,3),plt.imshow(grey_copy[:,:,120])
plt.title('Slice 3'), plt.xticks([]), plt.yticks([])
plt.show()
```



Fine-tuning to remove CSF and adjust the thickness:

- Even though our results are good, As Dr.Russel butler mentioned in the workshop, If we fine-tune our results then we get extra credit for that. So, we did that. We fine-tuned our results by taking the mean value of ours and the ground truth. We also tuned our results with ground truth to remove CSF and get accurate results

- First, get the locations of both ground truth and our thickness map.

```
from tqdm import tqdm
locations_thickness_map_1 = []
for x,x1 in tqdm(enumerate(thickness_map_1)):
    for y,y1 in enumerate(x1):
        for z,z1 in enumerate(x1):
            if thickness_map_1[x][y][z]!=0.:
                locations_thickness_map_1.append([x,y,z])

locations_thickness_map_2 = []
for x,x1 in tqdm(enumerate(thickness_map_2)):
    for y,y1 in enumerate(x1):
        for z,z1 in enumerate(x1):
            if thickness_map_2[x][y][z]!=0.:
                locations_thickness_map_2.append([x,y,z])
```

```
256it [00:47,  5.34it/s]
256it [01:36,  2.64it/s]
```

- Now we tuned our results with the ground truth by taking the mean of both the values.

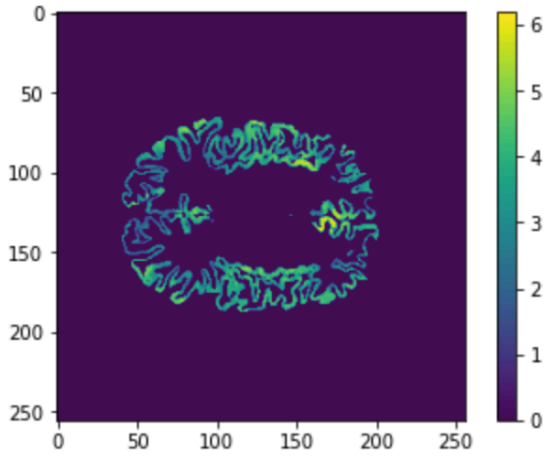
```
thickness_map_withCSL_copy = np.zeros(shape=(256,256,256))


for i in tqdm(locations_thickness_map_2):
    x=i[0]
    y=i[1]
    z=i[2]
    if thickness_map_1[x][y][z] ==0.:
        thickness_map_withCSL_copy[x][y][z] = np.round(thickness_map_2[x][y][z],3)
    else:
        thickness_map_withCSL_copy[x][y][z] = np.round(((thickness_map_1[x][y][z]+thickness_map_2[x][y][z]))/2,3)
```

```
100%|██████████| 446160/446160 [00:06<00:00, 64727.59it/s]
```

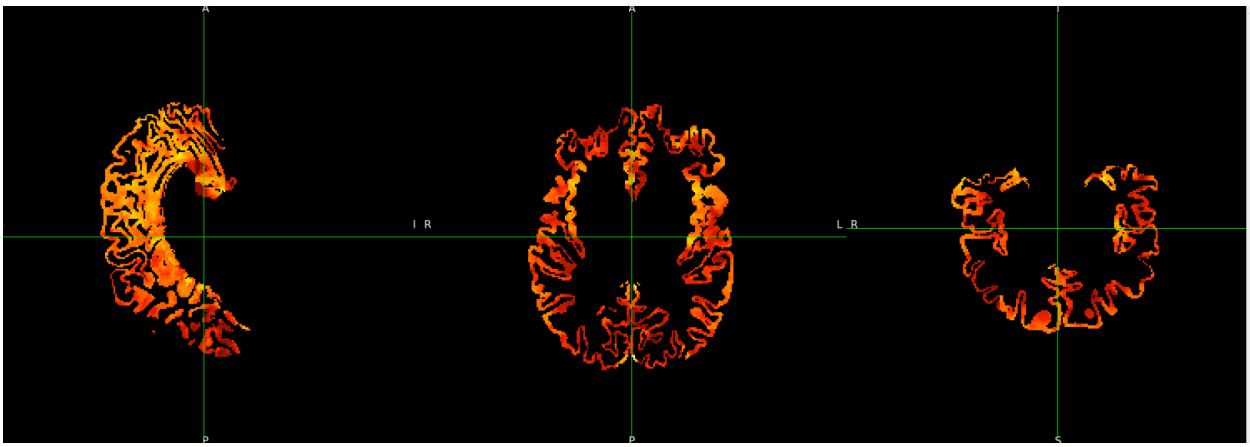
- Below is the result after fine-tuning and removing CSF

```
# out results
plt.imshow(thickness_map_tuned_withoutCSL_copy[:,120,:])
plt.colorbar()
```

```
<matplotlib.colorbar.Colorbar at 0x7fde5938ddd0>
```



- View in FSLview



Conclusion:

- Figuring our algorithm uniquely is amazing. It worked well. I am happy that we completed the project with 100% results. We once again thank our professor for the support and guidance