HW 1

Name: Xihao Cao

ID: U54244272

Date: 09/25/2022

Task1

(1) Show the images selected

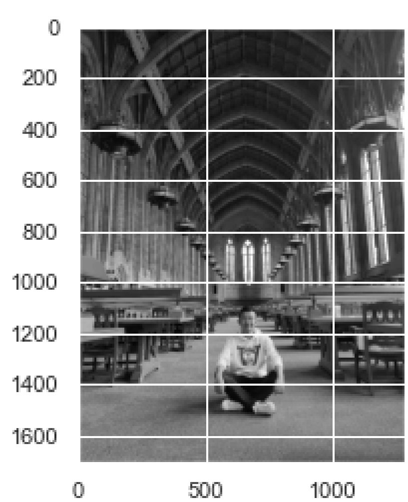
In [270…

```
# Task 1
import numpy as np
import matplotlib.pyplot as plt
import skimage.io
import skimage.color
%matplotlib widget
#reference: https://datacarpentry.org/image-processing/05-creating-histograms/

# read the images
withme = skimage.io.imread(fname = "withme.png", as_gray = True)
withoutme = skimage.io.imread(fname = "withoutme.png", as_gray = True)

# disply the images
fig, ax = plt.subplots()
plt.imshow(withme, cmap = "gray")
fig, ax = plt.subplots()
plt.imshow(withoutme, cmap = "gray")
```

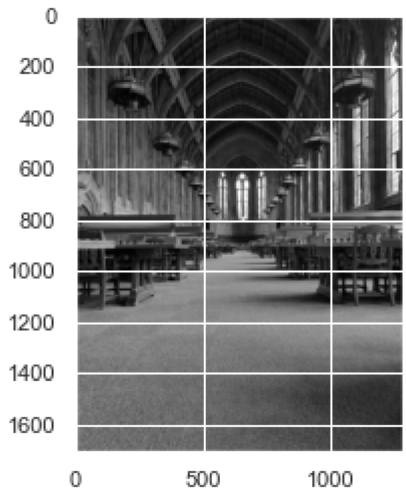
Out[270]:

```
<matplotlib.image.AxesImage at 0x2b13274f0>
```

Figure



Figure



The two imgaes selected are ploted above, the first one is me sitting on the library floor, while the other is the same place without me.

(2) Plot the gray scale histograms of two images

In [271…

```
#hiscount stores the count of each gray scale from 0-255, binedges stores the b
hiscount1, bin_edges1 = np.histogram(withme, bins = 256, range = (0, 1))
hiscount2, bin_edges2 = np.histogram(withoutme, bins = 256, range = (0, 1))
```

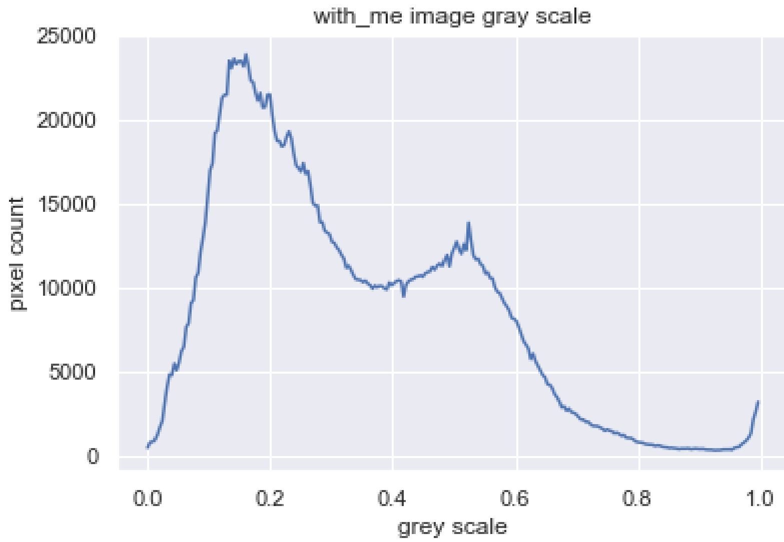
In [272…

```
#plot the gray scale histograms
plt.figure()
plt.title("with_me image gray scale")
plt.xlabel("grey scale")
plt.ylabel("pixel count")
plt.plot(bin_edges1[0:-1], hiscount1)

plt.figure()
plt.title("without_me image gray scale")
plt.xlabel("grey scale")
plt.ylabel("pixel count")
plt.plot(bin_edges2[0:-1], hiscount2)
```

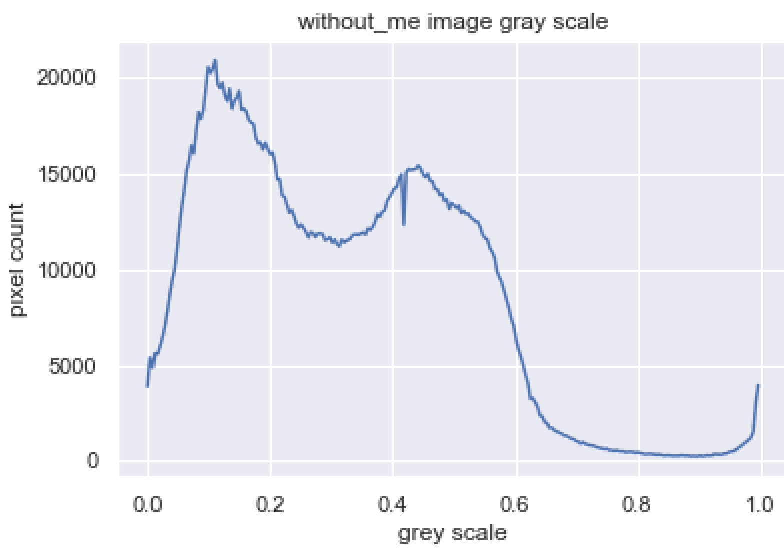
Out[272]:

```
[<matplotlib.lines.Line2D at 0x15565c2b0>]
```

Figure



Figure



In the two plots above I plot the gray scale frequency distributions of two images, as we can see, there is not much difference between them.

In [273…

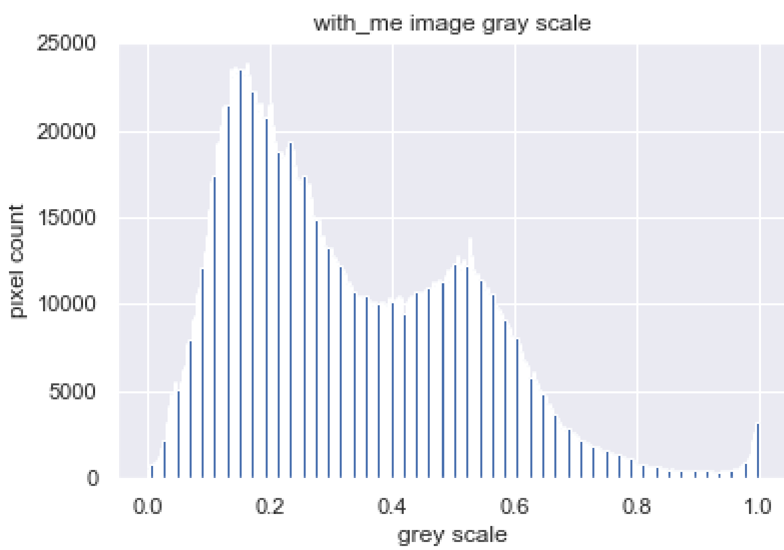
```
# try another plot function
with_gray = withme.flatten()
plt.figure()
plt.title("with_me image gray scale")
plt.xlabel("grey scale")
plt.ylabel("pixel count")
plt.hist(with_gray, bins=256, range=(0, 1))

without_gray = withoutme.flatten()
plt.figure()
plt.title("without_me image gray scale")
plt.xlabel("grey scale")
plt.ylabel("pixel count")
plt.hist(without_gray, bins=256, range=(0, 1))
```

Out[273]:

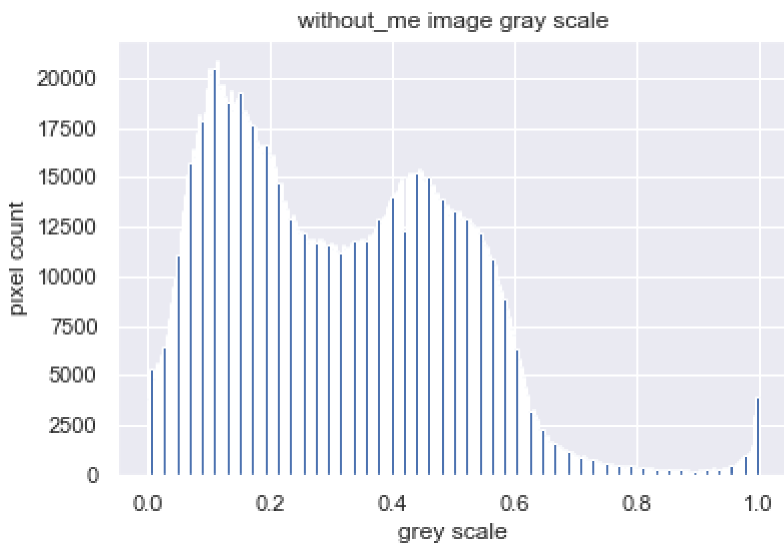
```
(array([ 3881.,  5388.,  4864.,  5601.,  5610.,  6026.,  6504.,  7082.,
         7860.,  8727.,  9448., 10027., 11055., 12264., 13294., 14141.,
        15152., 15695., 16467., 16052., 17239., 18170., 17856., 18296.,
        19433., 20534., 20212., 20479., 20898., 19656., 19428., 19704.,
        19075., 18766., 19403., 18357., 18757., 18953., 19265., 18287.,
        18359., 18208., 17813., 17661., 17606., 16870., 16570., 16640.,
        16275., 16614., 16296., 16013., 16113., 15568., 14701., 14703.,
        13863., 13790., 13377., 12965., 13103., 12784., 12377., 12180.,
        12347., 12173., 11938., 11668., 11938., 11903., 11691., 11861.,
        11888., 11838., 11529., 11592., 11686., 11397., 11568., 11313.,
        11202., 11566., 11410., 11525., 11528., 11685., 11819., 11834.,
        11806., 11872., 11922., 11826., 12142., 12067., 12221., 12489.,
        12898., 12745., 13014., 13094., 13568., 13775., 13985., 14229.,
        14285., 14743., 14972., 12312., 15058., 15247., 15194., 15219.,
        15251., 15424., 15291., 14966., 14825., 14992., 14634., 14595.,
        14212., 14199., 13884., 13961., 13557., 13658., 13160., 13472.,
        13351., 13208., 13341., 12949., 13066., 12896., 12916., 12709.,
        12648., 12509., 12482., 12201., 11808., 11644., 11548., 11121.,
        10902., 10617.,  9905.,  9587.,  9320.,  8856.,  8375.,  7972.,
         7409.,  7060.,  6344.,  5817.,  5430.,  4995.,  4472.,  4047.,
         3224.,  3281.,  3042.,  2841.,  2370.,  2324.,  2041.,  1977.,
         1713.,  1736.,  1598.,  1536.,  1469.,  1430.,  1330.,  1307.,
         1254.,  1174.,  1138.,  1044.,  1004.,   904.,   974.,   883.,
          846.,   819.,   813.,   757.,   710.,   679.,   642.,   614.,
          648.,   552.,   560.,   523.,   553.,   501.,   502.,   499.,
          444.,   465.,   474.,   455.,   417.,   447.,   407.,   387.,
          346.,   360.,   370.,   355.,   334.,   319.,   336.,   298.,
          272.,   292.,   292.,   280.,   258.,   272.,   260.,   290.,
          286.,   264.,   285.,   238.,   252.,   250.,   238.,   283.,
          237.,   260.,   297.,   270.,   285.,   355.,   344.,   336.,
          326.,   386.,   381.,   434.,   496.,   524.,   605.,   707.,
          793.,   904.,   995.,  1097.,  1227.,  1563.,  3036.,  3961.]),
 array([0.        , 0.00390625, 0.0078125 , 0.01171875, 0.015625  ,
        0.01953125, 0.0234375 , 0.02734375, 0.03125   , 0.03515625,
        0.0390625 , 0.04296875, 0.046875  , 0.05078125, 0.0546875 ,
        0.05859375, 0.0625    , 0.06640625, 0.0703125 , 0.07421875,
        0.078125  , 0.08203125, 0.0859375 , 0.08984375, 0.09375   ,
        0.09765625, 0.1015625 , 0.10546875, 0.109375  , 0.11328125,
        0.1171875 , 0.12109375, 0.125     , 0.12890625, 0.1328125 ,
        0.13671875, 0.140625  , 0.14453125, 0.1484375 , 0.15234375,
        0.15625   , 0.16015625, 0.1640625 , 0.16796875, 0.171875  ,
        0.17578125, 0.1796875 , 0.18359375, 0.1875    , 0.19140625,
        0.1953125 , 0.19921875, 0.203125  , 0.20703125, 0.2109375 ,
        0.21484375, 0.21875   , 0.22265625, 0.2265625 , 0.23046875,
        0.234375  , 0.23828125, 0.2421875 , 0.24609375, 0.25      ,
        0.25390625, 0.2578125 , 0.26171875, 0.265625  , 0.26953125,
        0.2734375 , 0.27734375, 0.28125   , 0.28515625, 0.2890625 ,
        0.29296875, 0.296875  , 0.30078125, 0.3046875 , 0.30859375,
        0.3125    , 0.31640625, 0.3203125 , 0.32421875, 0.328125  ,
        0.33203125, 0.3359375 , 0.33984375, 0.34375   , 0.34765625,
        0.3515625 , 0.35546875, 0.359375  , 0.36328125, 0.3671875 ,
        0.37109375, 0.375     , 0.37890625, 0.3828125 , 0.38671875,
        0.390625  , 0.39453125, 0.3984375 , 0.40234375, 0.40625   ,
        0.41015625, 0.4140625 , 0.41796875, 0.421875  , 0.42578125,
        0.4296875 , 0.43359375, 0.4375    , 0.44140625, 0.4453125 ,
        0.44921875, 0.453125  , 0.45703125, 0.4609375 , 0.46484375,
        0.46875   , 0.47265625, 0.4765625 , 0.48046875, 0.484375  ,
        0.48828125, 0.4921875 , 0.49609375, 0.5       , 0.50390625,
        0.5078125 , 0.51171875, 0.515625  , 0.51953125, 0.5234375 ,
        0.52734375, 0.53125   , 0.53515625, 0.5390625 , 0.54296875,
```

```
       0.546875  , 0.55078125, 0.5546875 , 0.55859375, 0.5625    ,
       0.56640625, 0.5703125 , 0.57421875, 0.578125  , 0.58203125,
       0.5859375 , 0.58984375, 0.59375   , 0.59765625, 0.6015625 ,
       0.60546875, 0.609375  , 0.61328125, 0.6171875 , 0.62109375,
       0.625     , 0.62890625, 0.6328125 , 0.63671875, 0.640625  ,
       0.64453125, 0.6484375 , 0.65234375, 0.65625   , 0.66015625,
       0.6640625 , 0.66796875, 0.671875  , 0.67578125, 0.6796875 ,
       0.68359375, 0.6875    , 0.69140625, 0.6953125 , 0.69921875,
       0.703125  , 0.70703125, 0.7109375 , 0.71484375, 0.71875   ,
       0.72265625, 0.7265625 , 0.73046875, 0.734375  , 0.73828125,
       0.7421875 , 0.74609375, 0.75      , 0.75390625, 0.7578125 ,
       0.76171875, 0.765625  , 0.76953125, 0.7734375 , 0.77734375,
       0.78125   , 0.78515625, 0.7890625 , 0.79296875, 0.796875  ,
       0.80078125, 0.8046875 , 0.80859375, 0.8125    , 0.81640625,
       0.8203125 , 0.82421875, 0.828125  , 0.83203125, 0.8359375 ,
       0.83984375, 0.84375   , 0.84765625, 0.8515625 , 0.85546875,
       0.859375  , 0.86328125, 0.8671875 , 0.87109375, 0.875     ,
       0.87890625, 0.8828125 , 0.88671875, 0.890625  , 0.89453125,
       0.8984375 , 0.90234375, 0.90625   , 0.91015625, 0.9140625 ,
       0.91796875, 0.921875  , 0.92578125, 0.9296875 , 0.93359375,
       0.9375    , 0.94140625, 0.9453125 , 0.94921875, 0.953125  ,
       0.95703125, 0.9609375 , 0.96484375, 0.96875   , 0.97265625,
       0.9765625 , 0.98046875, 0.984375  , 0.98828125, 0.9921875 ,
       0.99609375, 1.        ]),
<BarContainer object of 256 artists>)
```

Figure



Figure



The two plots above are also the gray scale distributions of the two images, but in a histogram format.

(3) calculated the kl div, js div, and conduct ks test on the images

In [274…

```
# calculate KL-div
def kl_divergence(p, q):
    # normalise
    p = 1.0*p / np.sum(p, axis=0)
    q = 1.0*q / np.sum(q, axis=0)
    #additional check to not divide by zero
    return np.sum(np.where((p != 0) & (q != 0), p * np.log(p / q), 0))


# calculate JS-div
def js_divergence(p,q):
    p = 1.0*p / np.sum(p, axis=0)
    q = 1.0*q / np.sum(q, axis=0)
    m = 0.5 * (p + q)
    js = (kl_divergence(p, m) + kl_divergence(q, m))/2
    return js

print(kl_divergence(with_gray, without_gray))
print(kl_divergence(without_gray, with_gray))

print(js_divergence(with_gray, without_gray))
print(js_divergence(without_gray, with_gray))


# perform Kolmogorov-Smirnov Test
from scipy.stats import kstest
print(kstest(with_gray, without_gray))
```

```
/var/folders/dm/4c8gs1vn1t7brq1lhj0fj1cc0000gn/T/ipykernel_14523/2280212357.p
y:7: RuntimeWarning: divide by zero encountered in true_divide
  return np.sum(np.where((p != 0) & (q != 0), p * np.log(p / q), 0))
/var/folders/dm/4c8gs1vn1t7brq1lhj0fj1cc0000gn/T/ipykernel_14523/2280212357.p
y:7: RuntimeWarning: divide by zero encountered in log
  return np.sum(np.where((p != 0) & (q != 0), p * np.log(p / q), 0))
/var/folders/dm/4c8gs1vn1t7brq1lhj0fj1cc0000gn/T/ipykernel_14523/2280212357.p
y:7: RuntimeWarning: invalid value encountered in multiply
  return np.sum(np.where((p != 0) & (q != 0), p * np.log(p / q), 0))
0.33075205822183734
0.32174107266354685
0.07207435666194287
0.07207435666194287
KstestResult(statistic=0.06991592174677608, pvalue=0.0)
```

The kl-div(with me || without me) is 0.33075205822183734, while the kl-div(without me || with me) is 0.32174107266354685, which are relative small since the two images seleceted are very similar.

The js-div(with me || without me) = js-div(without me || with me) = 0.07207435666194287, which is also not big, also states that the two images are fairly similar in gray scale.

The KS test gives us a statistic = 0.06991592174677608 with pvalue = 0.0, also supports the viewpoint that two images are similar.

Task 2

(1) Simulate 3D data

In [291…

```
# reference: https://towardsdatascience.com/gaussian-mixture-models-d13a5e915c8
import numpy as np
import sklearn.mixture as skm
from sklearn.mixture import GaussianMixture
from matplotlib import pyplot as plt
import seaborn as sns
sns.set()

## Generate synthetic data
N,D = 200, 3 # number of points and dimenstinality


means = np.array([[0.4, 0.3, 0.4],
                  [0, 0.5, 0.5],
                  [0.7, 0.7, 0.4]])
covs = np.array([np.diag([0.01, 0.01, 0.03]),
                 np.diag([0.01, 0.05, 0.01]),
                 np.diag([0.03, 0.07, 0.01])])
n_gaussians = means.shape[0]

# Next, we generate points using a multivariate normal distribution

points = []
for i in range(n_gaussians):
    x = np.random.multivariate_normal(means[i], covs[i], N )
```

```
    points.append(x)
points = np.concatenate(points)
```

(2) Cluster the data using Guassian Mixture Modeling with EM

In [292…

```
#reference: https://www.itzikbs.com/gaussian-mixture-model-gmm-3d-point-cloud-c
#fit the gaussian models with 1, 5, 30 iterations

gmm1 = skm.GaussianMixture(n_components=n_gaussians, covariance_type='diag', ma
gmm1.fit(points)

gmm2 = skm.GaussianMixture(n_components=n_gaussians, covariance_type='diag', ma
gmm2.fit(points)

gmm3 = skm.GaussianMixture(n_components=n_gaussians, covariance_type='diag', ma
gmm3.fit(points)
```

```
/Users/xihaoc/opt/anaconda3/lib/python3.9/site-packages/sklearn/mixture/_base.
py:277: ConvergenceWarning: Initialization 1 did not converge. Try different i
nit parameters, or increase max_iter, tol or check for degenerate data.
  warnings.warn(
```

Out[292]:

```
GaussianMixture(covariance_type='diag', max_iter=30, n_components=3)
```

The itertation times of gmm1,2,3 are 5, 12, 30 respectively.

(3)

In [277…

```
# This is a visualization function used to plot the GMM
import matplotlib.patches as patches
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.cm as cmx
import os
# reference: https://github.com/sitzikbs/gmm_tutorial/blob/master/visualizatior

def visualize_3d_gmm(points, w, mu, stdev, export=True):
    '''
    plots points and their corresponding gmm model in 3D
    Input:
        points: N X 3, sampled points
        w: n_gaussians, gmm weights
        mu: 3 X n_gaussians, gmm means
        stdev: 3 X n_gaussians, gmm standard deviation (assuming diagonal covar
    Output:
        None
    '''

    n_gaussians = mu.shape[1]
    N = int(np.round(points.shape[0] / n_gaussians))
    # Visualize data
    fig = plt.figure(figsize=(8, 8))
    axes = fig.add_subplot(111, projection='3d')
    axes.set_xlim([-1, 1])
    axes.set_ylim([-1, 1])
    axes.set_zlim([-1, 1])
    plt.set_cmap('Set1')
    colors = cmx.Set1(np.linspace(0, 1, n_gaussians))
```

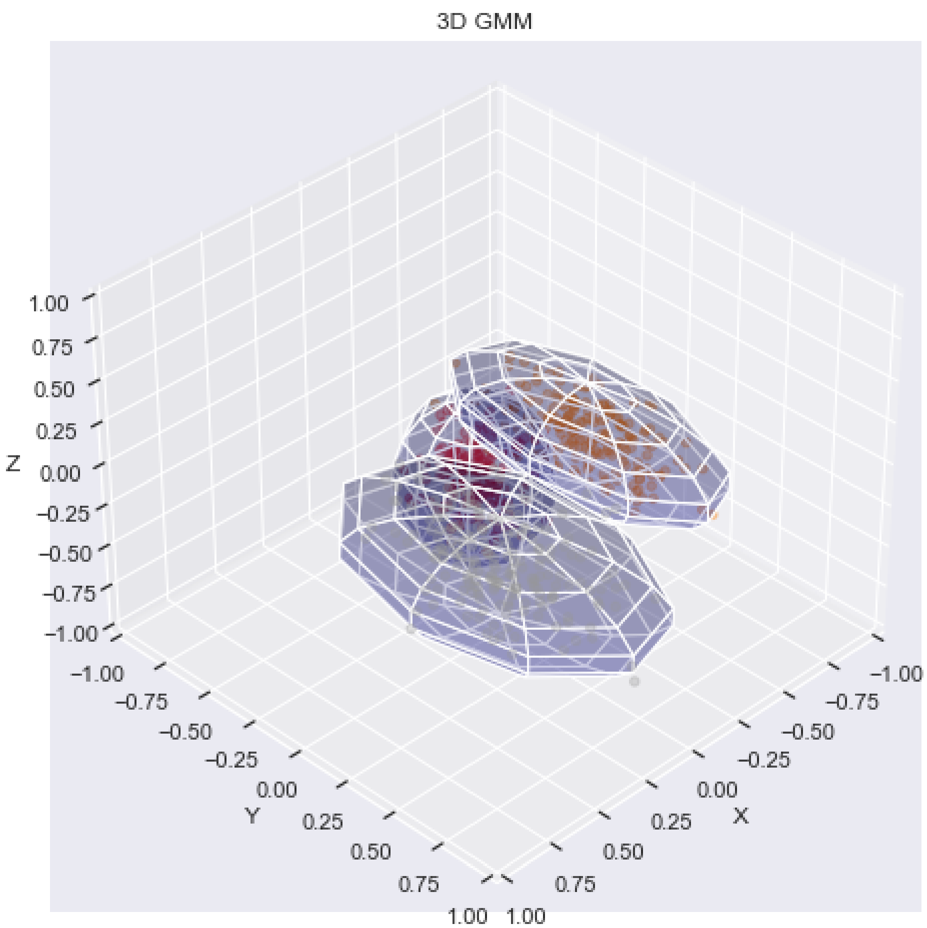
```
    for i in range(n_gaussians):
        idx = range(i * N, (i + 1) * N)
        axes.scatter(points[idx, 0], points[idx, 1], points[idx, 2], alpha=0.3,
        plot_sphere(w=w[i], c=mu[:, i], r=stdev[:, i], ax=axes)

    plt.title('3D GMM')
    axes.set_xlabel('X')
    axes.set_ylabel('Y')
    axes.set_zlabel('Z')
    axes.view_init(35.246, 45)
    if export:
        if not os.path.exists('images/'): os.mkdir('images/')
        plt.savefig('images/3D_GMM_demonstration.png', dpi=100, format='png')
    plt.show()



def plot_sphere(w=0, c=[0,0,0], r=[1, 1, 1], subdev=10, ax=None, sigma_multipli
    '''
        plot a sphere surface
        Input:
            c: 3 elements list, sphere center
            r: 3 element list, sphere original scale in each axis ( allowing to
            subdiv: scalar, number of subdivisions (subdivision^2 points sample
            ax: optional pyplot axis object to plot the sphere in.
            sigma_multiplier: sphere additional scale (choosing an std value wh
        Output:
            ax: pyplot axis object
    '''

    if ax is None:
        fig = plt.figure()
        ax = fig.add_subplot(111, projection='3d')
    pi = np.pi
    cos = np.cos
    sin = np.sin
    phi, theta = np.mgrid[0.0:pi:complex(0,subdev), 0.0:2.0 * pi:complex(0,subd
    x = sigma_multiplier*r[0] * sin(phi) * cos(theta) + c[0]
    y = sigma_multiplier*r[1] * sin(phi) * sin(theta) + c[1]
    z = sigma_multiplier*r[2] * cos(phi) + c[2]
    cmap = cmx.ScalarMappable()
    cmap.set_cmap('jet')
    c = cmap.to_rgba(w)

    ax.plot_surface(x, y, z, color=c, alpha=0.2, linewidth=1)

    return ax
```

In [293…

```
visualize_3d_gmm(points, gmm1.weights_, gmm1.means_.T, np.sqrt(gmm1.covariances

visualize_3d_gmm(points, gmm2.weights_, gmm2.means_.T, np.sqrt(gmm2.covariances

visualize_3d_gmm(points, gmm3.weights_, gmm3.means_.T, np.sqrt(gmm3.covariances
```

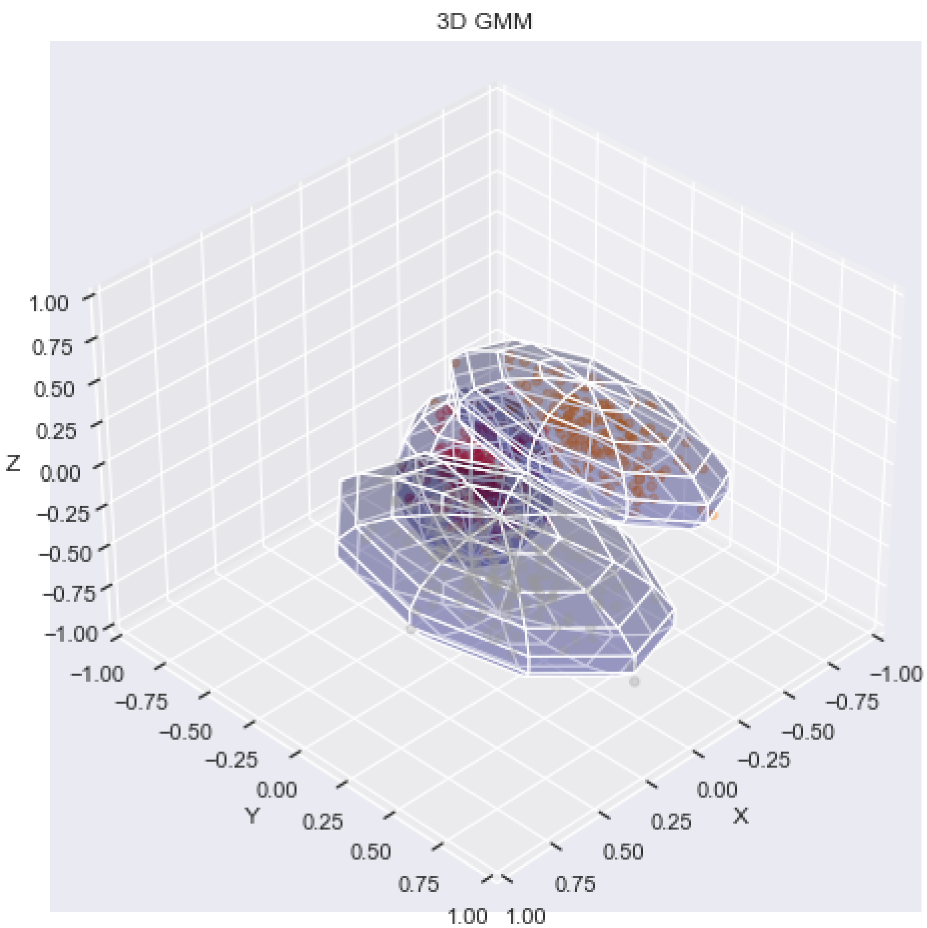
```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length matches with
*x* & *y*.  Please use the *color* keyword-argument or provide a 2D array with
a single row if you intend to specify the same RGB or RGBA value for all point
s.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length matches with
*x* & *y*.  Please use the *color* keyword-argument or provide a 2D array with
a single row if you intend to specify the same RGB or RGBA value for all point
s.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length matches with
*x* & *y*.  Please use the *color* keyword-argument or provide a 2D array with
a single row if you intend to specify the same RGB or RGBA value for all point
s.
```

Figure



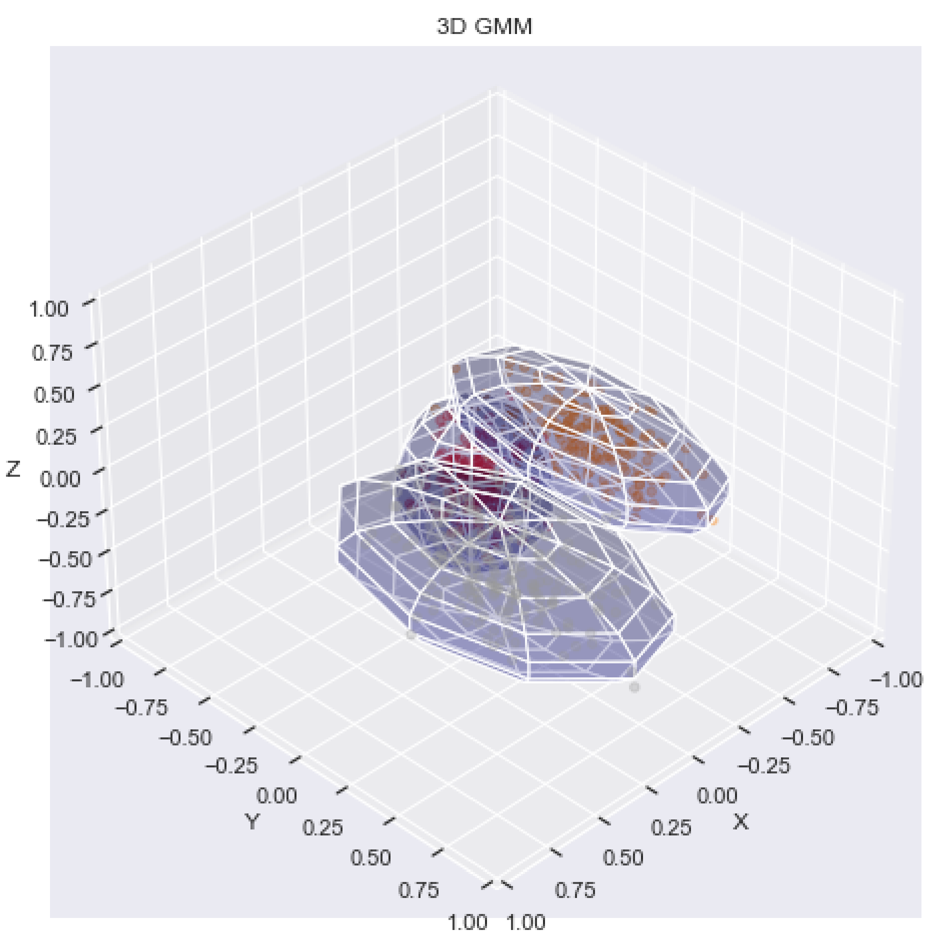
```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length matches with
*x* & *y*.  Please use the *color* keyword-argument or provide a 2D array with
a single row if you intend to specify the same RGB or RGBA value for all point
s.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length matches with
*x* & *y*.  Please use the *color* keyword-argument or provide a 2D array with
a single row if you intend to specify the same RGB or RGBA value for all point
s.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length matches with
*x* & *y*.  Please use the *color* keyword-argument or provide a 2D array with
a single row if you intend to specify the same RGB or RGBA value for all point
s.
```

Figure



```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length matches with
*x* & *y*.  Please use the *color* keyword-argument or provide a 2D array with
a single row if you intend to specify the same RGB or RGBA value for all point
s.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length matches with
*x* & *y*.  Please use the *color* keyword-argument or provide a 2D array with
a single row if you intend to specify the same RGB or RGBA value for all point
s.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length matches with
*x* & *y*.  Please use the *color* keyword-argument or provide a 2D array with
a single row if you intend to specify the same RGB or RGBA value for all point
s.
```

Figure



As we can see, as the EM iteration times increase, the model gets a better ability to cluster the data. However, since the simulated data are designed to have clear boundaries, so it does not need that much of iterations to get the optimal model, so the difference between each iterations are not huge.