ISYE6740-HW3

pkubsad

February 2021

1. Order of faces using ISOMAP [50 points]

(a) (10 points) Visualize the similarity graph (you can either show the adjacency matrix, or similar to the lecture slides, visualize the graph using graph visualization packages such as Gephi (https://gephi.org) and illustrate a few images corresponds to nodes at different parts of the graph, e.g., mark them by hand or use software packages).

Answer:
I calculated the threshold epsilon such that every node has atleast 50 nieghbours. But this resulted in some nodes having close to 700 to 800 nodes. As discussed in piazza, https://piazza.com/class/khkx47q2d3p5ln?cid=356$_f$60.*Iswitchedbyimplementationtousekneighbor*

(a) Adjacency/Similarity matrix

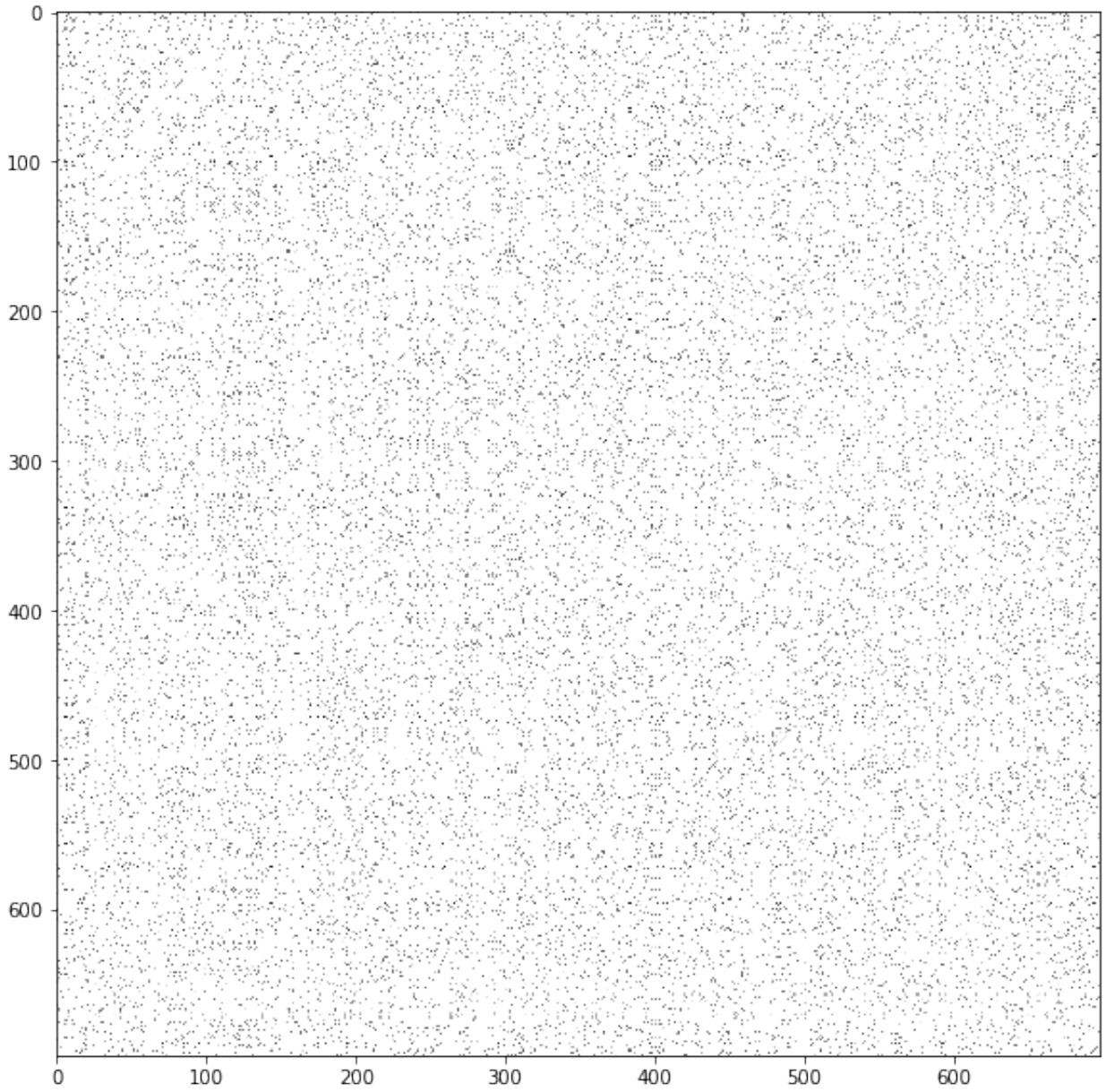


Figure 1: Adjacency/Similarity matrix

(b) Adjacency as network of graph

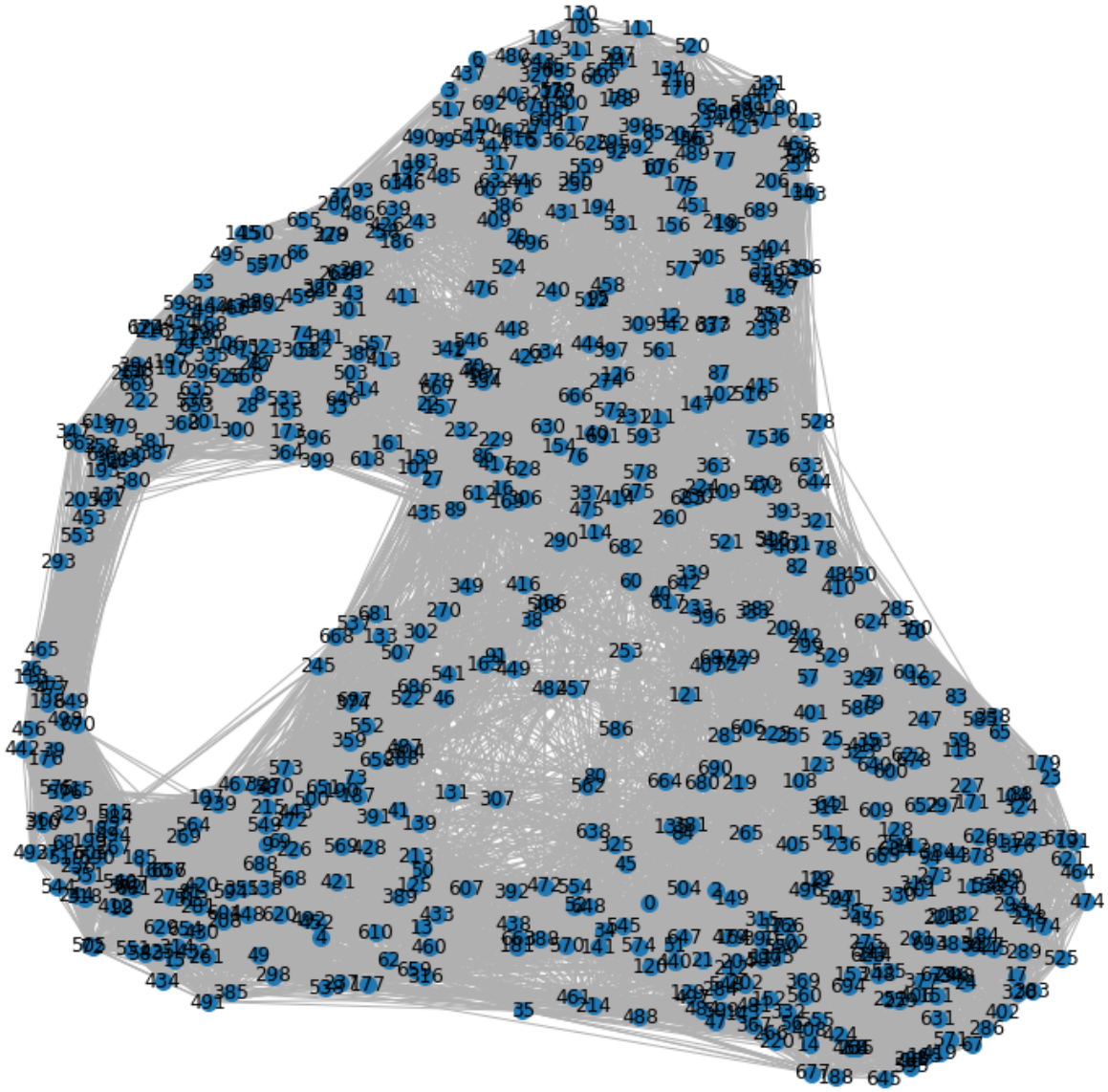


Figure 2: Adjacency matrix as network of graph

(c) Sample Images in adjacency graph,(I could not add the image in the graph, so am printing them separately

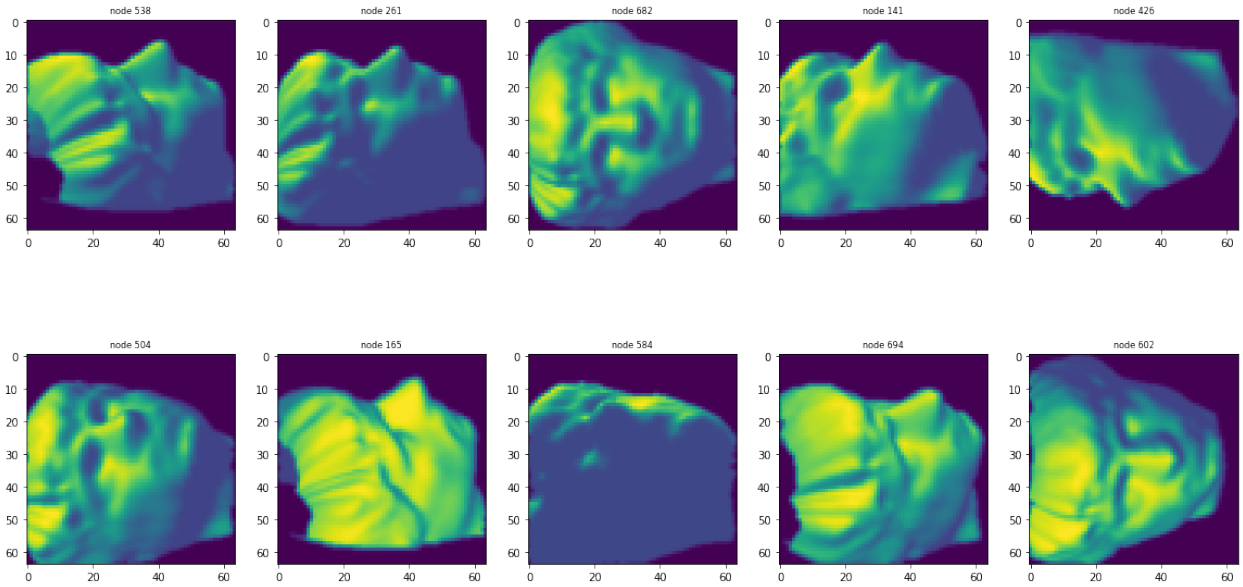


Figure 3: Images in adjacency graph

(b) (20 points) Implement the ISOMAP algorithm yourself to obtain a two-dimensional low-dimensional embedding. Plot the embeddings using a scatter plot, similar to the plots in lecture slides. Find a few images in the embedding space and show what these images look like. Comment on do you see any visual similarity among them and their arrangement, similar to what you seen in the paper?

Answer:
Observations looking at the output graph and running the program again and again:

(a) The faces pointing in a particular direction are grouped together.

(b) Faces looking in left direction are at the bottom of the graph.

(c) Faces looking in the right direction are at the top of the graph.

(d) Faces looking up are to the left, looking down are to the right of the graph.

(e) Everytime we run the program, the positions of the faces migth change, but they are grouped together.

The network graph I obtained for isomap with euclidean distance:

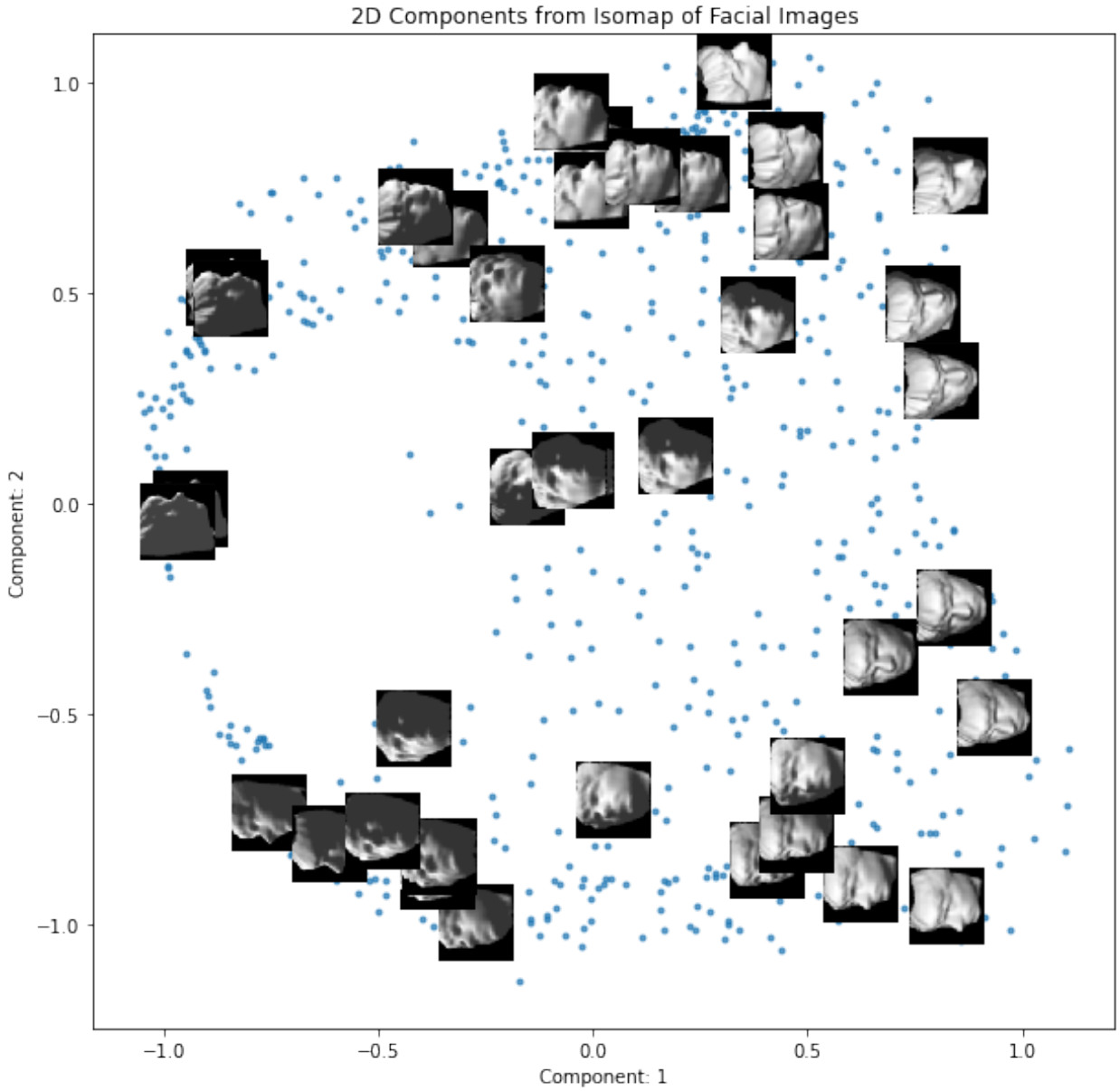


Figure 4: Isomap - euclidean distance

(c) (10 points) Now choose ℓ_1 distance (or Manhattan distance) between images (recall the definition from "Clustering" lecture)). Repeat the steps above. Use ϵ-ISOMAP to obtain a $k = 2$ dimensional embedding. Present a plot of this embedding. Do you see any difference by choosing a different similarity measure by comparing results in Part (b) and Part (c)?

Answer:
Observations:

(a) The resulting network graph is somewhat similar to euclidean graph.

(b) I see few faces grouped in the incorrect groups compared to euclidean distance.

The network graph I obtained for isomap with manhattan distance:

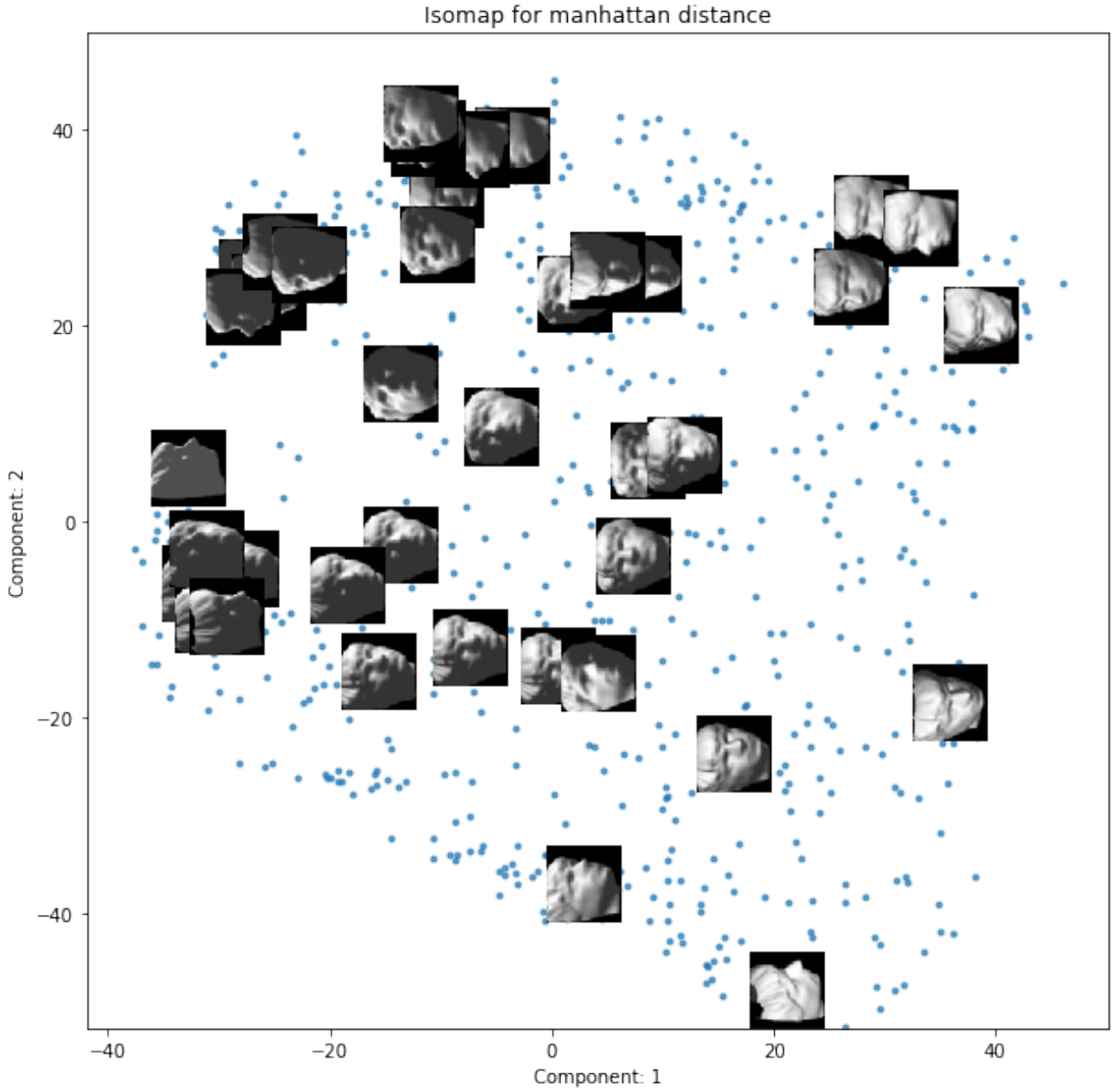


Figure 5: Isomap - manhattan distance

(d) (10 points) Perform PCA (you can now use your implementation written in Question 1) on the images and project them into the top 2 principal components. Again show them on a scatter plot. Explain whether or you see a more meaningful projection using

ISOMAP than PCA.

Answer:
I have run the PCA algorithm on the distances matrix calculated by useing cdist function for euclidean distance.

(a) PCA model, confuses quite a few faces into wrong categories compared to ISOMAP.

(b) The groupings can be logically identified as faces poinitng in certain direction, but there accuracy seems low.

(c) From the newtork graph, clearly ISOMAP has more meaningful projections compared to PCA.

Resulting network graph:

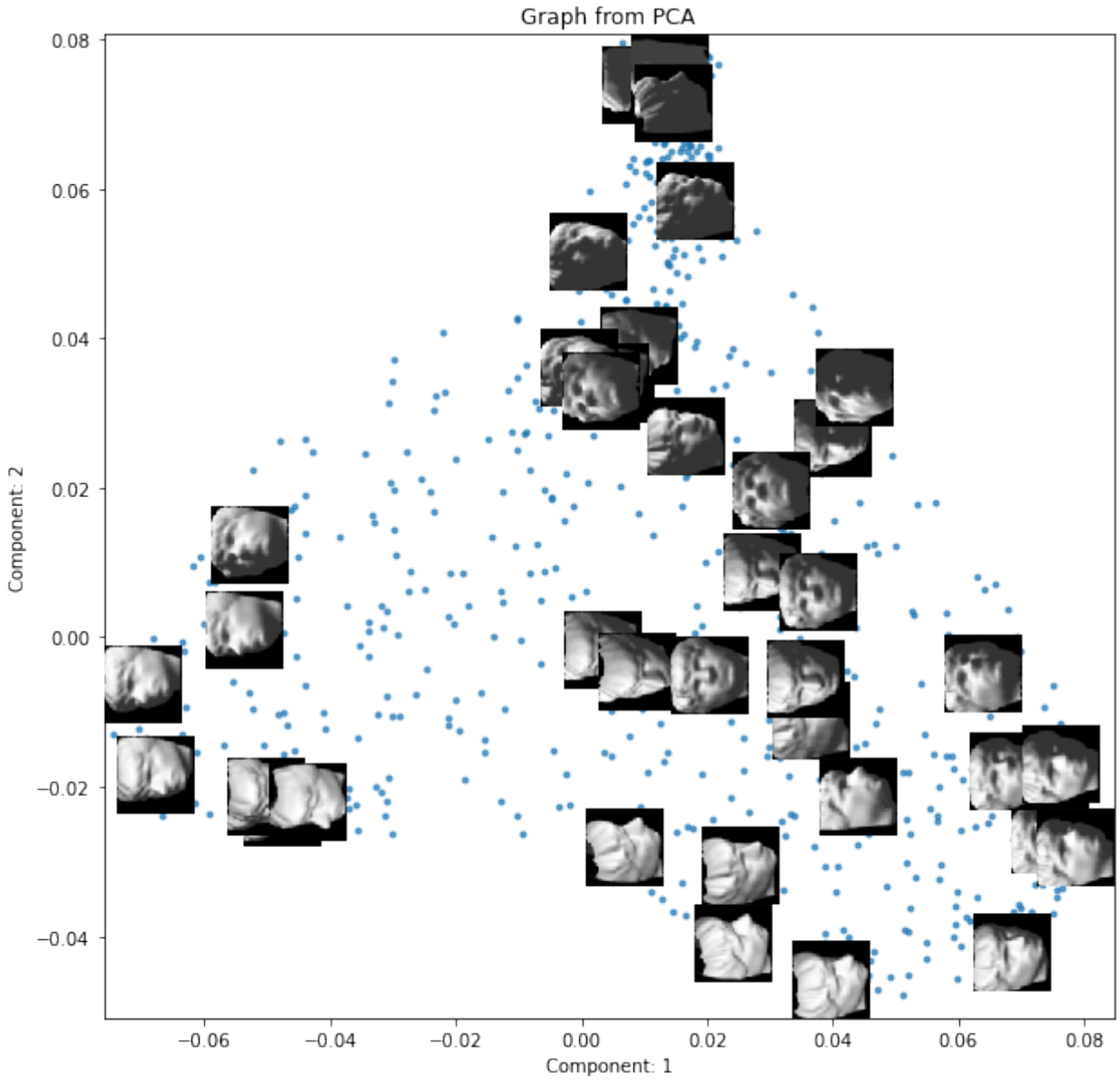


Figure 6: PCA - euclidean distance

2. Density estimation: Psychological experiments. (50 points)

(a) (10 points) Form the 1-dimensional histogram and KDE to estimate the distributions of amygdala and acc, respectively. For this question, you can ignore the variable orientation. Decide on a suitable number of bins so you can see the shape of the distribution clearly.

Answer:

For Histogram: I ran the plot for various bin size. I have used seaborn package's hist-plot method to plot the histogram. This method takes in bin width instead of size. Calculation I used for bin width:

$$bin\_width = (maxValueInData - minValueInData)/number\_of\_bins$$

Looking at the plots, we can conclude the number of bins = 10 produces a better distribution plot.

For KDE plot, am using seaborn packages's kdeplot method. I calculated the factor 'h' value using the formula provided by prof:

$$h \approx 1.06\hat{\sigma}n^{-1/5},$$

But this was resulting in plots not consistent with histograms. When I dug deeper into implementation of the seaborn package, the bandwidth parameter accounts the std-deviation part of the above equation by itself. We have to provide the rest of the value. I verified this by calculating the default co-varaince factor that will be used if we dont provide any value.

band width factor calculated = (n ** (-1/5))
band width factor calualted = 90 ** (-1/5)= 0.4065851364889782

band width factor caluclated by gaussian kde library method = 0.4065851364889782

band width used by the method = bandwidth factor / std deviation of the data
band width used by the method = 0.4065851364889782 / 0.020321534068902875
band width used by the method = 0.008262433703070296

calculated by formula prof provided
h $\approx 1.06\hat{\sigma}n^{-1/5}$
$h \approx 1.06\hat{0}.001984299693119354 * 90^{-1/5}$,
$h \approx 0.00085519396$

Below are the histogram plots and kde plots for 1d:

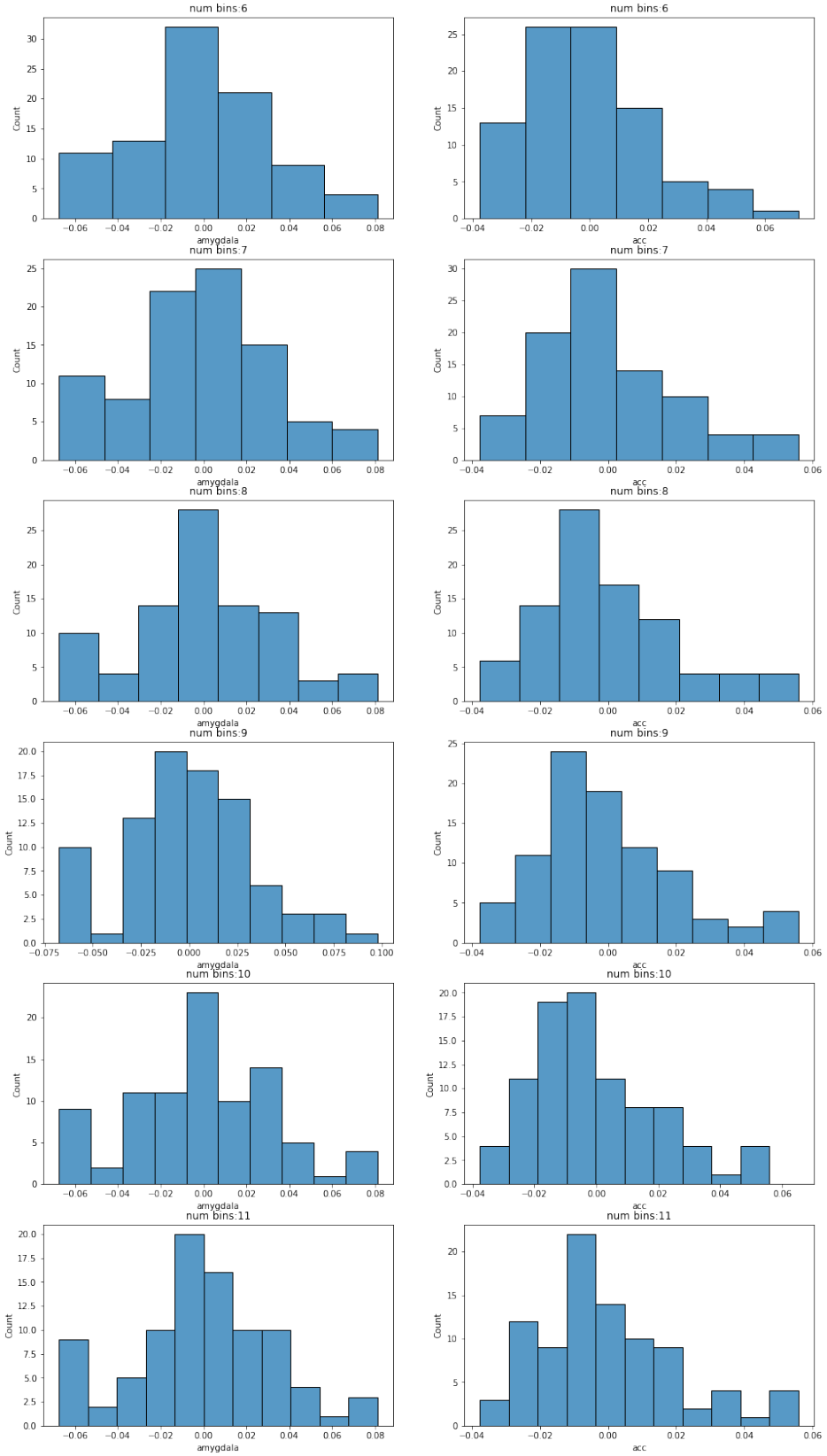


Figure 7: 1d Histogram

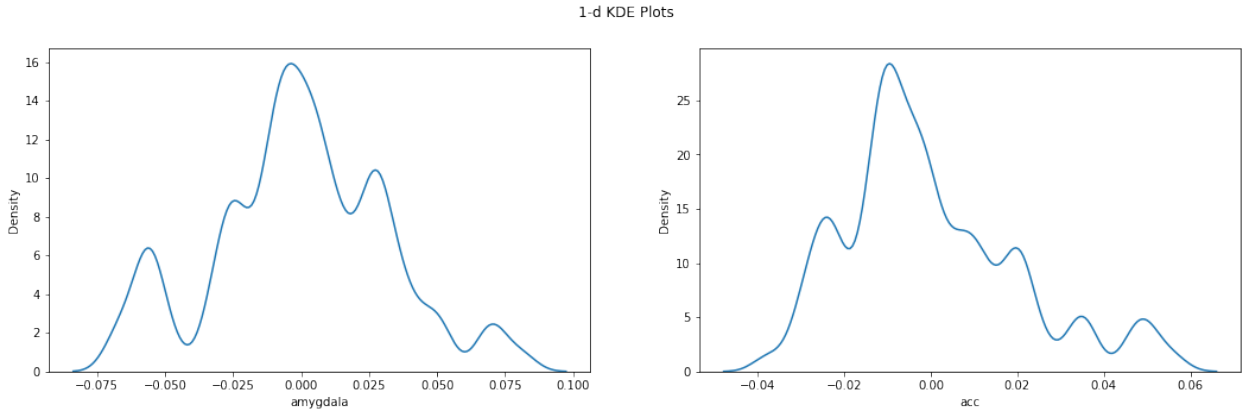


Figure 8: 1d KDE

(b) (10 points) Form 2-dimensional histogram for the pairs of variables (amygdala, acc). Decide on a suitable number of bins so you can see the shape of the distribution clearly.

Answer:
I have printed the heat map and 3d projection of 2 dimensional histogram. Based on the 3d projection of the bins, we can say bin size of 10 is appropriate value for this data.

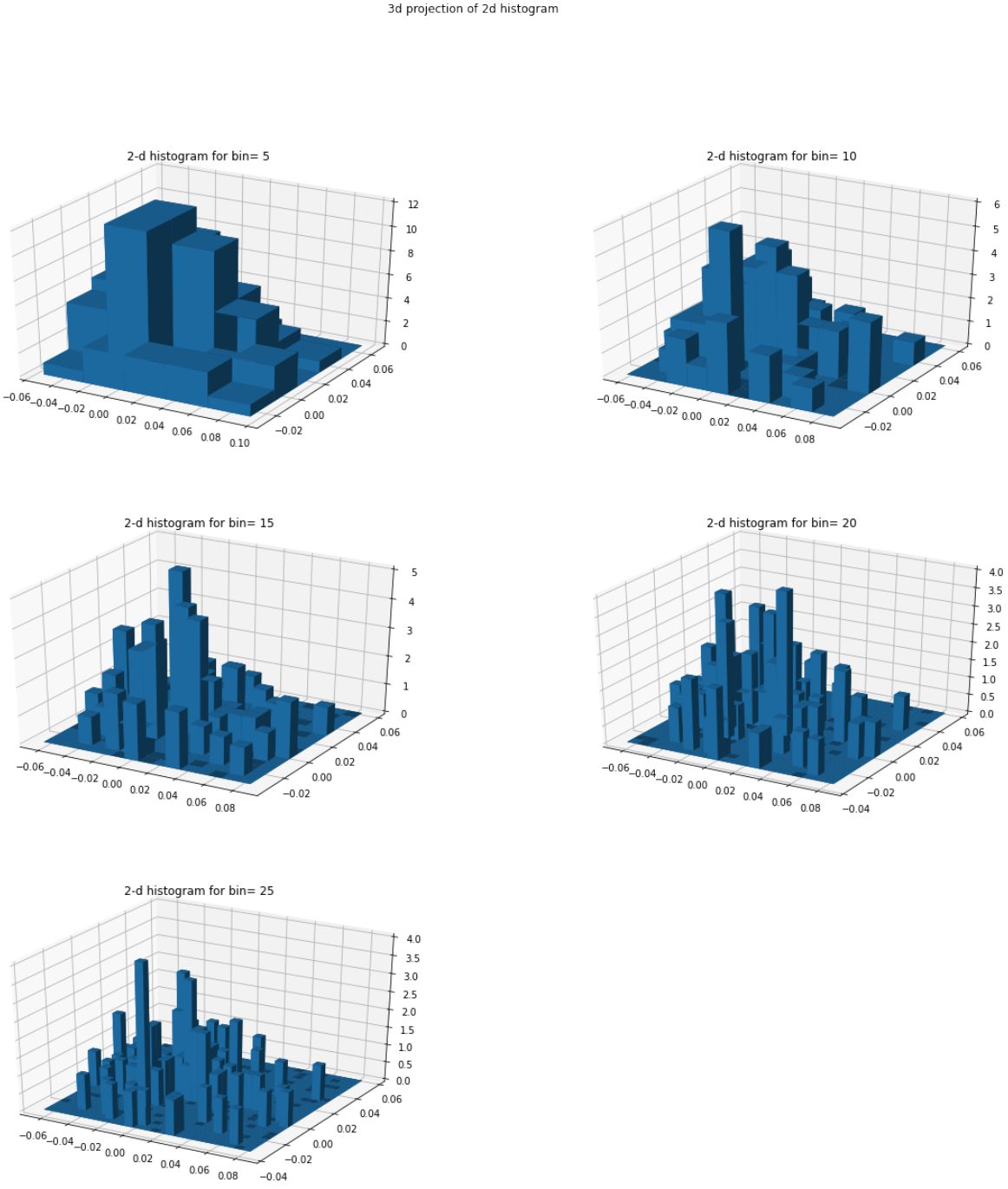


Figure 9: 3d projection of 2d histogram

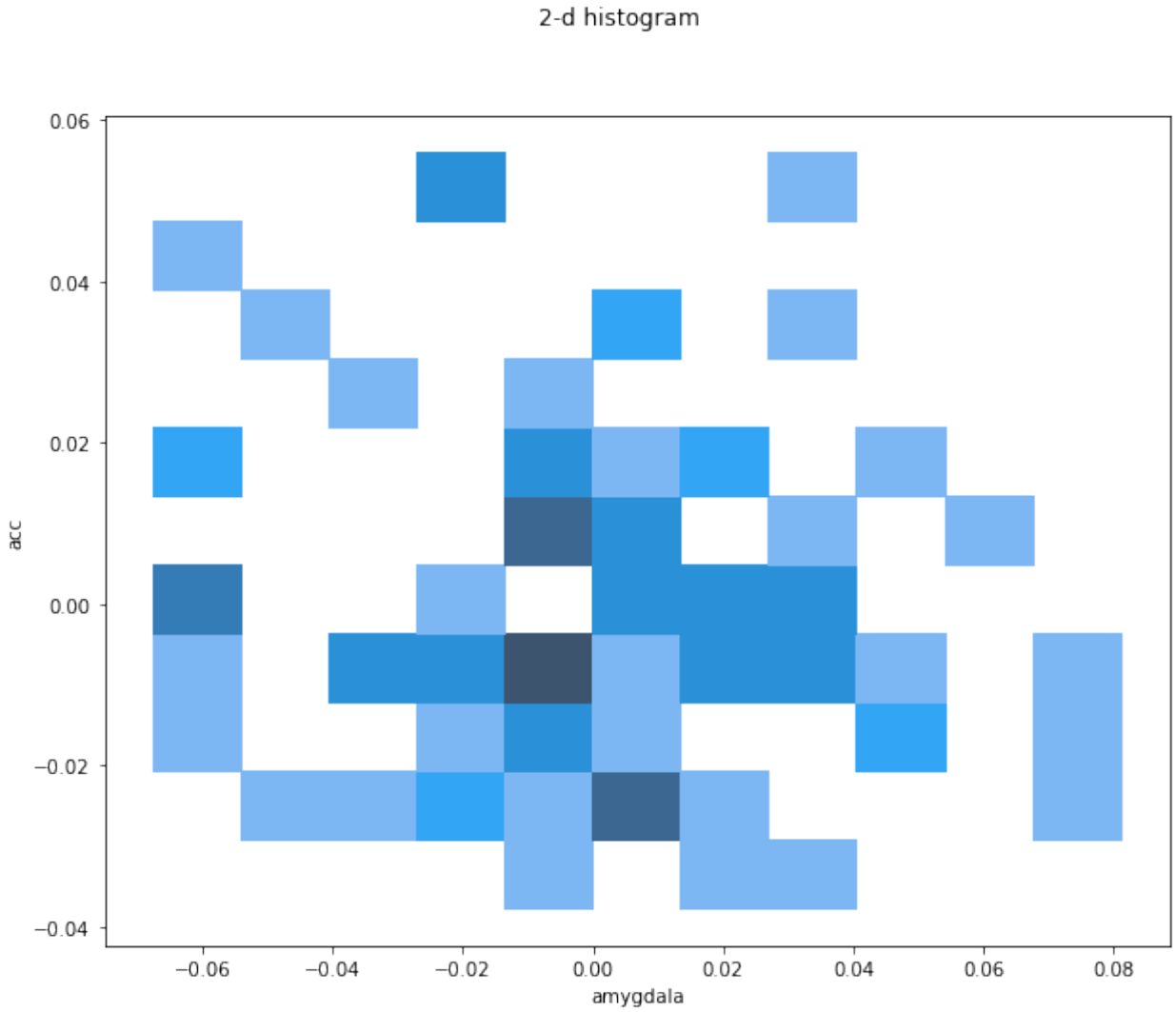


Figure 10: Heat Map of 2d histogram

(c) (10 points) Use kernel-density-estimation (KDE) to estimate the 2-dimensional density function of (amygdala, acc) (this means for this question, you can ignore the variable orientation). Set an appropriate kernel bandwidth $h > 0$.

Answer:
Ran the seaborn package's kdeplot on 2 dimensions with different bandwidth factors. The bandwidth with good contours were obtained at bw factor =0.5. When I checked the default implementation of method in the package, it uses scott method which is n**(-1/d+4) = n**(1/6) = 0.4723815372

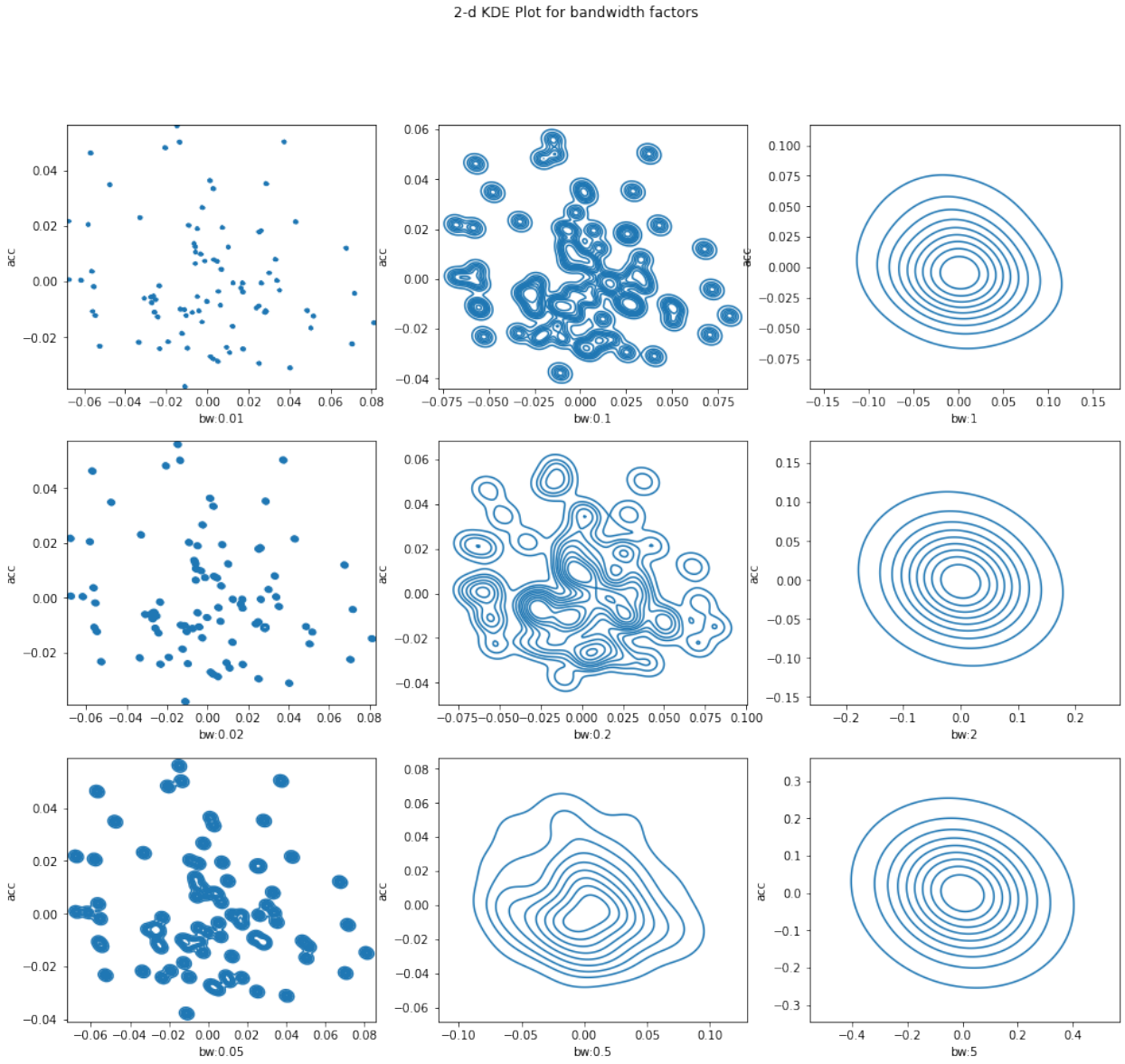


Figure 11: 2d kde

I have calculated p(amygdala,acc) - p(amygdala)*p(acc). If the variables are independent this factor will be remaining closer to zero. But we observe that this factor is fluctuating well above zero. Hence we can conclude that these 2 variables are not independent.

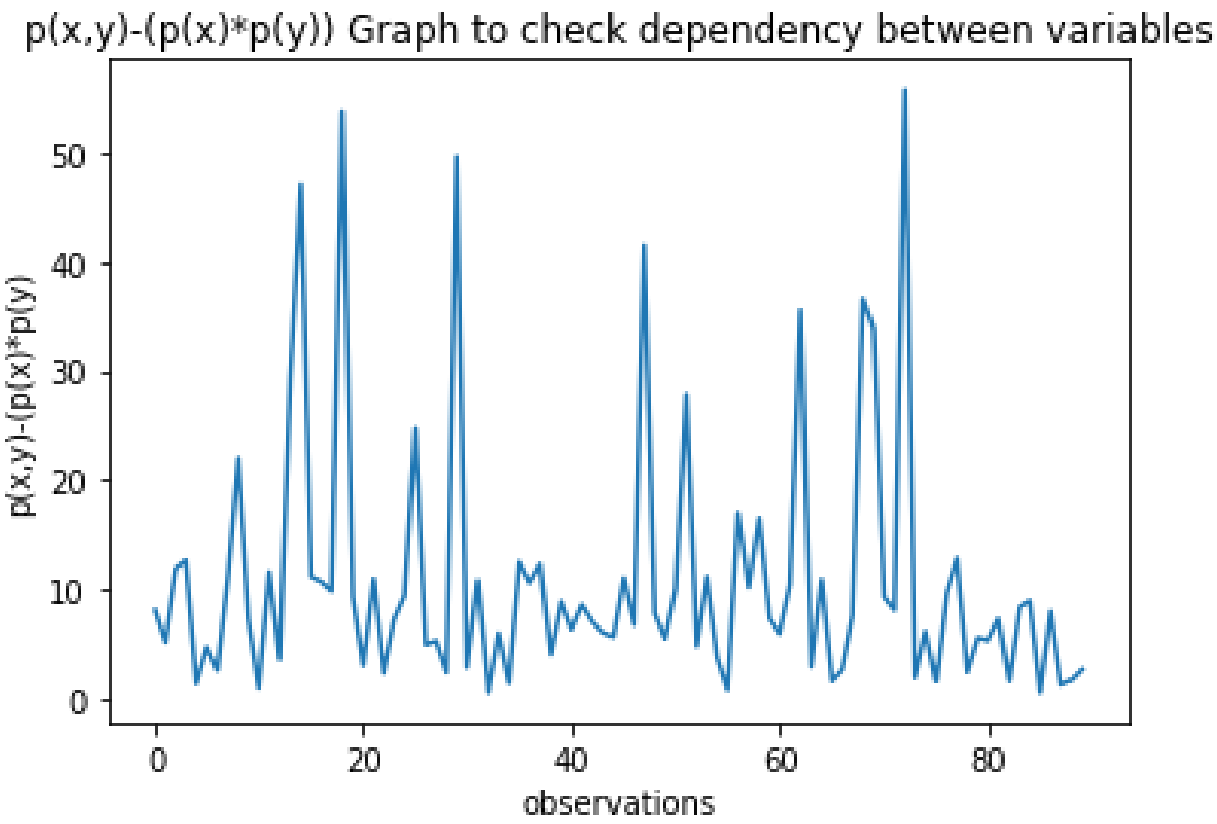


Figure 12: p(x,y)-p(x)*p(y)

(10 points) We will consider the variable orientation and consider conditional distributions. Please plot the estimated conditional distribution of amygdala conditioning on political orientation: $p(\text{amygdala}|\text{orientation} = c)$, $c = 2, \ldots, 5$, using KDE. Set an appropriate kernel bandwidth $h > 0$. Do the same for the volume of the acc: plot $p(\text{acc}|\text{orientation} = c)$, $c = 2, \ldots, 5$ using KDE. (Note that the conditional distribution can be understood as fitting a distribution for the data with the same orientation. Thus you should plot 8 one-dimensional distribution functions in total for this question.)

Answer:

The sample mean calculated for all the Cs:

C	2	3	4	5
Amygdala	0.01906153846153846	0.0005875	-0.004719512195121951	-0.0056916666666
ACC	-0.014769230769230769	0.0016708333333	0.0013097560975609756	0.0081416666666

Even though the means for these orientations are close to each other, it seems like the the data set is similar to for all orientations. But if we look at the conditional distributions below, we can see that distributions for these conditional orientations are different. This

tells that apart from the standard statistical measures like mean, distribution of the data gives full picture of the nature of the data.

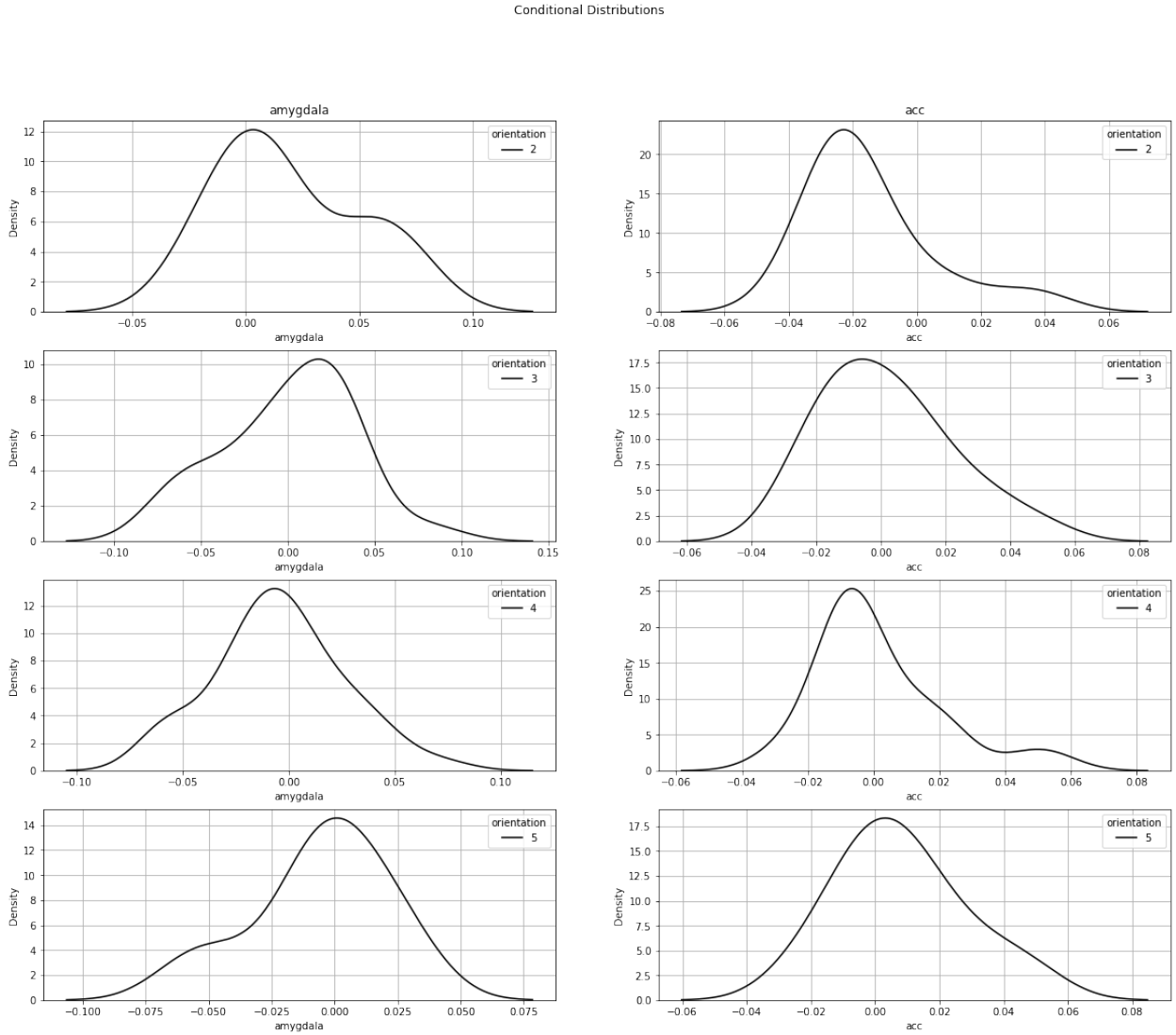


Figure 13: conditional distributions

(10 points) Again we will consider the variable orientation. We will estimate the conditional *joint* distribution of the volume of the amygdala and acc, conditioning on a function of political orientation: $p(\text{amygdala}, \text{acc}|\text{orientation} = c)$, $c = 2, \ldots, 5$. You will use two-dimensional KDE to achieve the goal; et an appropriate kernel bandwidth $h > 0$. Please show the two-dimensional KDE (e.g., two-dimensional heat-map, two-dimensional contour plot, etc.).

Answer: Based on the distribution plots we can conclude that:

1. Folks with orientation 2 have almost opposite amygdala and acc values compared to

orientation 5. This can be observed when we look at denser distribution point for o=2 is around (-0.02, -.025) and for o=5 is (0.01, 0.025).

2. The distribution of orientation 3 and 4 are somewhat similar at the denser parts. Both have concentrations around (0,0). The outliers can drag the distributions in the farther quartiles.

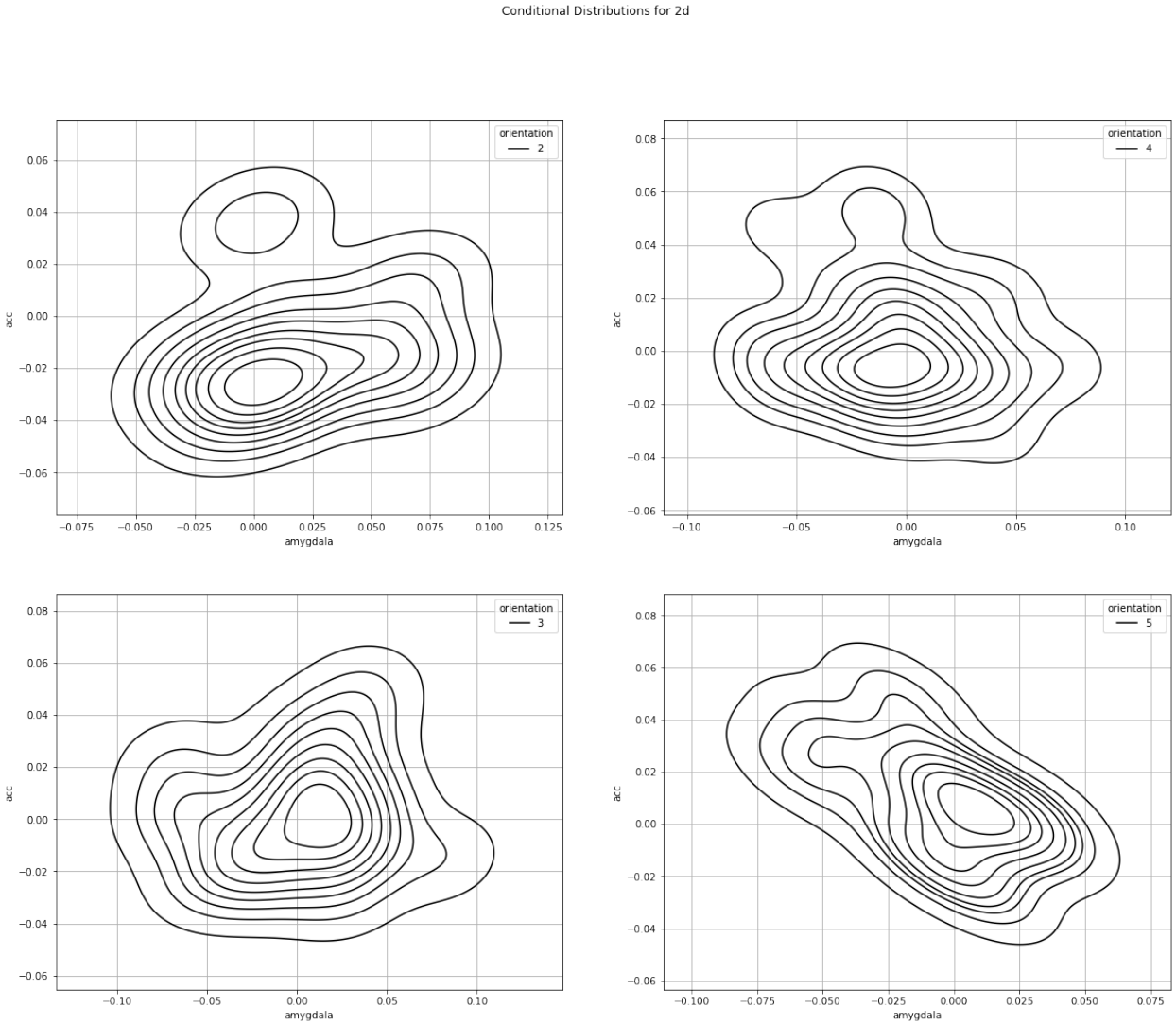


Figure 14: conditional distributions for 2d

Q1

February 23, 2021

[1]:
```
import math
from scipy.io import loadmat
from scipy.spatial import distance
import numpy as np
from matplotlib import pyplot as plt
import sys
sys.path.insert(0, 'data/ShortestPath/')

import Matrix_D as sp
from sklearn.utils.graph import graph_shortest_path
import networkx as nx
from scipy.sparse.linalg import eigs
import pandas as pd
from sklearn.neighbors import kneighbors_graph
```

[2]:
```
# source: https://keestalkstech.com/2020/05/
 ↪plotting-a-grid-of-pil-images-in-jupyter/
# This method displays face images as a grid.
# I was unable to add the images on the graph, so am printing them on the side

def display_images(
    num=10,
    columns=5, width=20, height=8, max_images=10,
    label_wrap_length=50, label_font_size=8):
    imageCount, pixelsCount = images.T.shape
    df=pd.DataFrame(images.T)


    pixelsindim = int(math.sqrt(pixelsCount))

    height = max(height, int(num/columns) * height)
    plt.figure(figsize=(width, height))
    i=0
    for j in range(num):
        img_num = np.random.randint(0, imageCount)
        img = df.iloc[img_num,:].values.reshape(pixelsindim, pixelsindim)
        plt.subplot(num / columns + 1, columns, i + 1)
```

```
        plt.imshow(img)
        i+=1
        title="node %s"%img_num
        plt.title(title, fontsize=label_font_size);
```

[3]:
```
# Source: https://stackoverflow.com/questions/29572623/
 ↪plot-networkx-graph-from-adjacency-matrix-in-csv-file
# This method plots adjacency matrix as a network graph
def show_graph_with_labels(adjacency_matrix):
    rows, cols = np.where(adjacency_matrix != 0.)
    edges = zip(rows.tolist(), cols.tolist())
    gr = nx.Graph()
    gr.add_edges_from(edges)
    fig=plt.figure(figsize=(10,10))
    nx.draw(gr, node_size=100, with_labels=True, edge_color='#B0B0B0')
    plt.show()
    display_images()
```

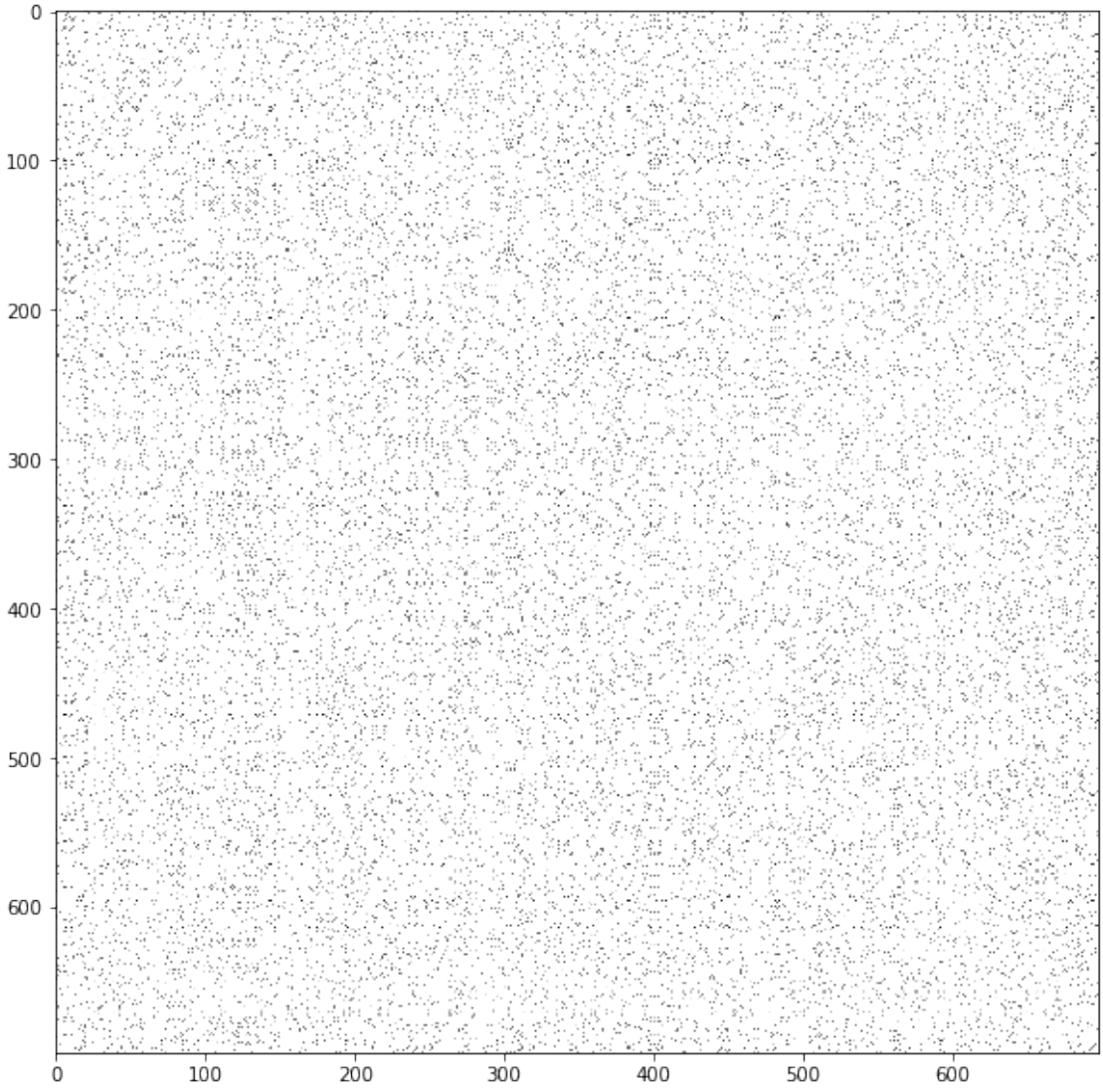
[4]:
```
# source: https://benalexkeen.com/isomap-for-dimensionality-reduction-in-python/
# This method plots isomap as scatter plot with faces embedded in them
def drawScatterPlot(z,index,title="Isomap for euclidean distance"):
    z=z.real
    isoMapData = pd.DataFrame(z, columns=['c 2', 'c 1'])
    df=pd.DataFrame(images.T)
    imageCount, pixelsCount = images.T.shape
    pixelsIndim = int(math.sqrt(pixelsCount))

    fig = plt.figure()
    fig.set_size_inches(10, 10)
    ax = fig.add_subplot(111)
    ax.set_title(title)
    ax.set_xlabel('Component: 1')
    ax.set_ylabel('Component: 2')

    # Show 40 of the images ont the plot
    x_size = (max(isoMapData['c 1']) - min(isoMapData['c 1'])) * 0.08
    y_size = (max(isoMapData['c 2']) - min(isoMapData['c 2'])) * 0.08
    for i in range(40):
        img_num = np.random.randint(0, imageCount)
        x0 = isoMapData.loc[img_num, 'c 1'] - (x_size / 2.)
        y0 = isoMapData.loc[img_num, 'c 2'] - (y_size / 2.)
        x1 = isoMapData.loc[img_num, 'c 1'] + (x_size / 2.)
        y1 = isoMapData.loc[img_num, 'c 2'] + (y_size / 2.)
        img = df.iloc[img_num,:].values.reshape(pixelsIndim, pixelsIndim)
        ax.imshow(img, aspect='auto', cmap=plt.cm.gray,
```

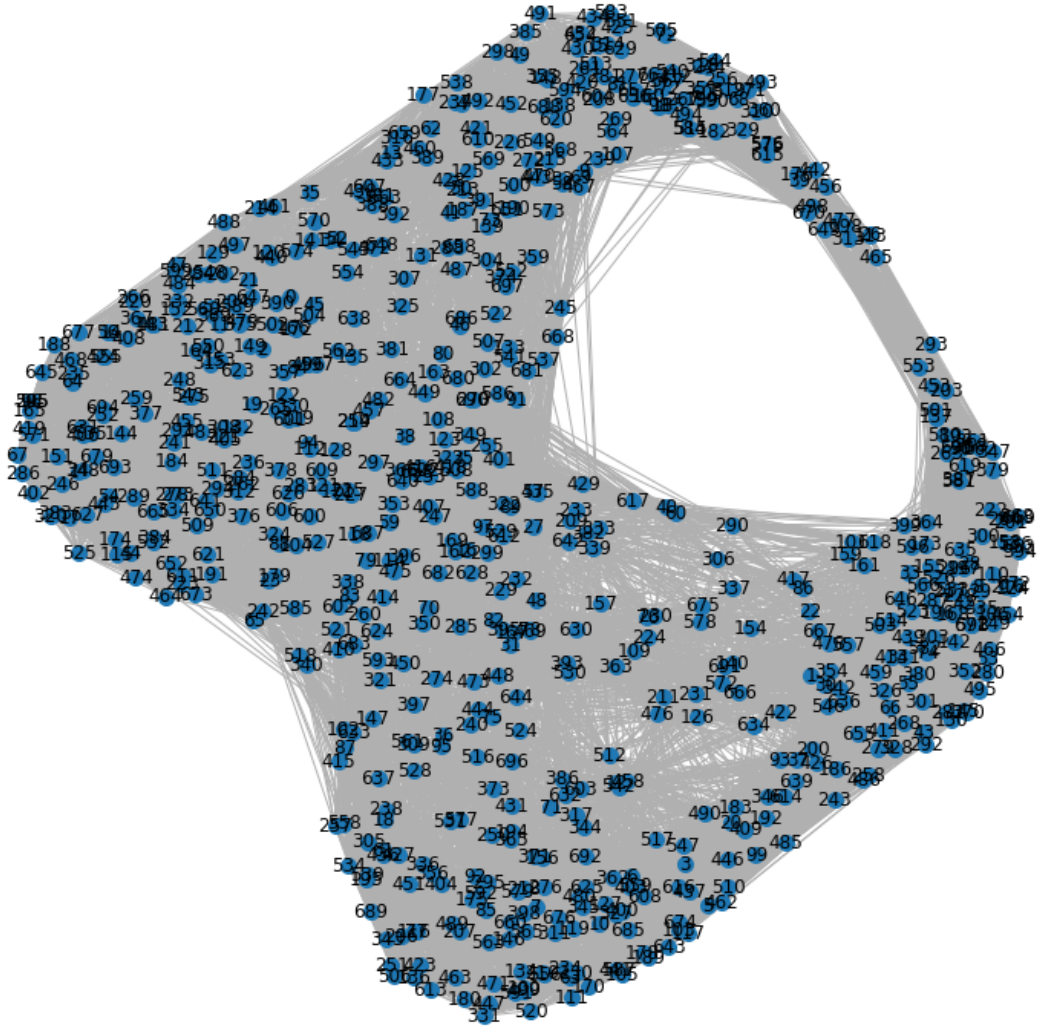
```
                    interpolation='nearest', zorder=100000, extent=(x0, x1, y0,␣
↪y1))

    # Show 2D components plot
    ax.scatter(isoMapData['c 1'], isoMapData['c 2'], marker='.',alpha=0.7)
```

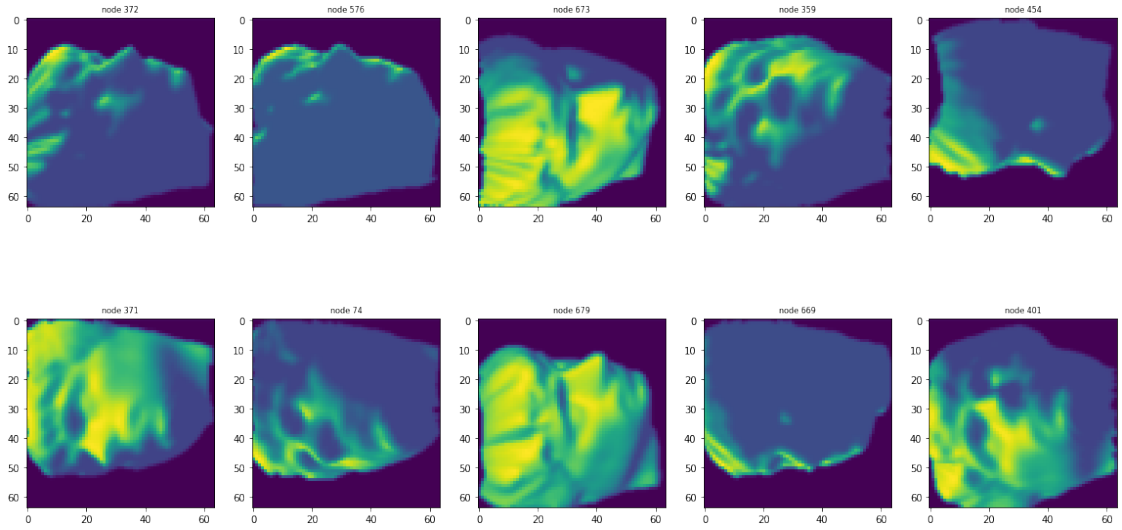
[5]:
```
def draw_adjacency_matrix(adjacency_matrix):
    #Plot adjacency matrix in toned-down black and white
    fig = plt.figure(figsize=(10, 10)) # in inches
    plt.imshow(adjacency_matrix,
                    cmap="Greys",
                    interpolation="none")
    plt.show()
```

[6]:
```
#q1.a: code to print adjacency matrix and plot a network graph

images=loadmat('data/isomap.mat')['images']
print(images.shape)


m,n=images.T.shape

A=kneighbors_graph(images.T,50,mode='distance',metric='euclidean')
A=A.toarray().astype(float)
print("*** details A ****")
print("Adjacency matrix of shape:",A.shape)
draw_adjacency_matrix(A)
show_graph_with_labels(A)
```

```
(4096, 698)
*** details A ****
Adjacency matrix of shape: (698, 698)
```


0
100
200
300
400
500
600
0
100
200
300
400
500
600

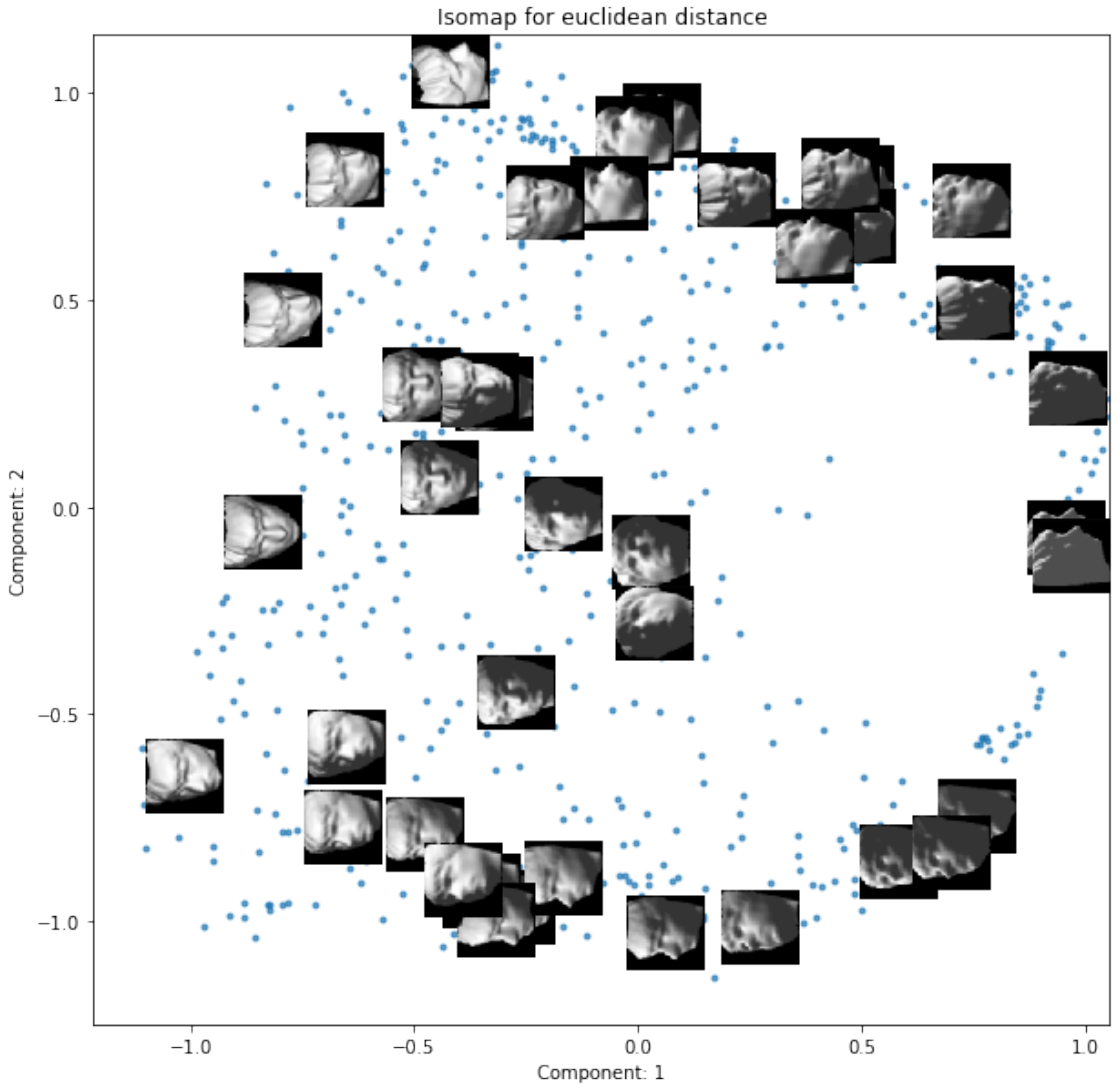




[7]:
```
# q1.b: Code to run isomap for euclidean distance
D=graph_shortest_path(A)
print("*** details of D ****")
display(D)
print(D.shape)

I = np.eye(N=D.shape[1])
H = I - 1/m * np.ones(I.shape)

# Compute the C matrix:
C = -1/(2*m) * H @ D**2 @ H
print("**** C details *****")
print(C)

# Compute the eigenvalues and eigenvectors of C and sort them in descending␣
 ↪order:
eigenValues, eigenVectors = eigs(C, k=2)
eigenValues=eigenValues.real
eigenVectors=eigenVectors.real
print("eigen values :",eigenValues[:2])

# Normalize the leading eigenvectors:
Z = eigenVectors * np.sqrt(eigenValues)
print("*** isomap details***")
print(Z)

drawScatterPlot(Z,2)
plt.show()
```

```
*** details of D ****
```

```
array([[ 0.        , 36.81445264,  6.74323967, ..., 38.37208728,
        42.92944356, 22.12792861],
       [35.79581505,  0.        , 35.65572383, ..., 19.38947669,
        31.94099475, 43.63678962],
       [ 6.74323967, 36.86151895,  0.        , ..., 41.56405181,
        37.99022264, 21.9105186 ],
       ...,
       [38.21289582, 25.50792808, 43.38415169, ...,  0.        ,
        42.97281714, 34.58582119],
       [37.79770022, 31.94099475, 37.46881102, ..., 40.40947504,
         0.        , 28.56874432],
       [22.12792861, 44.00676204, 21.9105186 , ..., 32.86808328,
        33.61094962,  0.        ]])


(698, 698)
**** C details *****
[[ 0.50752435 -0.29311327  0.48111232 … -0.36384742 -0.32799925
   0.35290803]
 [-0.35194931  0.73612675 -0.33861849 …  0.47997809  0.31972648
  -0.6019695 ]
 [ 0.43729314 -0.33325592  0.47602622 … -0.58428066 -0.07935411
   0.32210783]
 …
 [-0.31877799  0.43135441 -0.61488051 …  0.91059701 -0.11096383
   0.0664994 ]
 [-0.33012935  0.1326592  -0.30623649 … -0.2930788   1.1779024
   0.30475163]
 [ 0.26344154 -0.60284504  0.27646052 …  0.02369686  0.28958443
   0.81032269]]
eigen values : [303.10086475 221.12984176]
*** isomap details***
[[ 0.73164404 -0.03533823]
 [-0.72057881 -0.03554422]
 [ 0.68634448 -0.24258927]
 …
 [-0.55562141  0.7708762 ]
 [-0.52981403 -0.1869635 ]
 [ 0.39033427  0.31566979]]
```



[8]:
```
# q1.c code to run isomap for manhattan distance
m,n=images.T.shape

A=kneighbors_graph(images.T,50,mode='distance',p=1,metric='minkowski')
A=A.toarray().astype(float)

D=graph_shortest_path(A)
print("*** details of D ****")
display(D)
print(D.shape)

I = np.eye(N=D.shape[1])
H = I - 1/m * np.ones(I.shape)
```

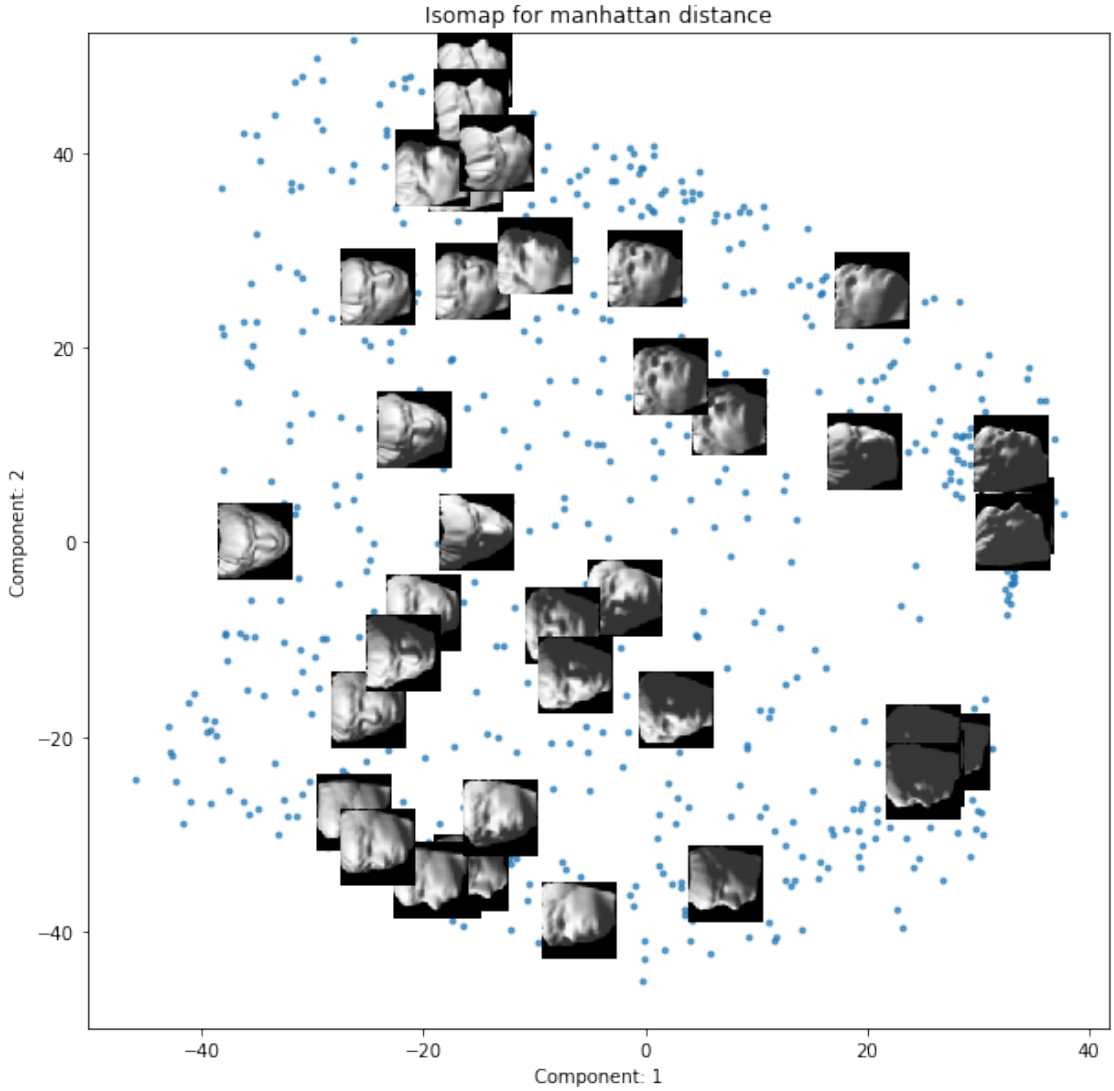
```
# Compute the C matrix:
C = -1/(2*m) * H @ D**2 @ H
print("**** C details *****")
print(C)

# Compute the eigenvalues and eigenvectors of C and sort them in descending␣
 ↪order:
eigenValues, eigenVectors = eigs(C, k=2)
eigenValues=eigenValues.real
eigenVectors=eigenVectors.real
print("eigen values :",eigenValues[:2])

# Normalize the leading eigenvectors:
Z = eigenVectors * np.sqrt(eigenValues)
print("*** isomap details***")
print(Z)

drawScatterPlot(Z,2,title="Isomap for manhattan distance")
plt.show()
```

```
*** details of D ****
```

```
array([[   0.        , 1590.47686887,  257.36994485, ..., 1475.7786152 ,
        1761.47800245,  890.07999387],
       [1576.54659926,    0.        , 1571.76559436, ...,  671.17362132,
        1466.35934436, 1645.21473652],
       [ 257.36994485, 1576.26792279,    0.        , ..., 1603.74270833,
        1619.39721201,  897.41136642],
       ...,
       [1548.45085784,  718.41424632, 1710.38213848, ...,    0.        ,
        1803.39227941, 1037.29825368],
       [1523.97898284, 1466.35934436, 1619.39721201, ..., 1681.54249387,
           0.        , 1093.54650735],
       [ 947.30033701, 1648.8004902 ,  908.37166054, ...,  995.19365809,
        1203.25324755,    0.        ]])
```

```
(698, 698)
**** C details *****
[[  979.9677822   -725.89920509   952.05336484 …  -545.76384555
   -521.17781346   501.41920502]
 [ -769.34382615  1117.27905051  -739.02659268 …   722.79537043
    192.32877731  -838.85941917]
 [  835.72557148  -790.45990112   902.70984827 …  -924.84048675
   -273.87510199   395.23897383]
 …
 [ -651.30332786   802.71342429 -1009.7820088  …  1100.63164914
```

```
   -541.93237708   384.4423554 ]
 [ -287.46801983   -57.86302735  -482.78652143 …  -614.8835283
   2097.71639401   608.56103329]
 [  396.0278817   -802.35232734   467.30984209 …   363.77048924
    723.22347793  1127.8090495 ]]
eigen values : [445045.89541166 347736.65621263]
*** isomap details***
[[ 30.9325751   -1.15195352]
 [-28.84565696   2.77627652]
 [ 28.24253364 -10.23749415]
 …
 [-21.76925144  27.73882627]
 [-23.18829618  -2.13559227]
 [  6.63697035  18.33386291]]
```

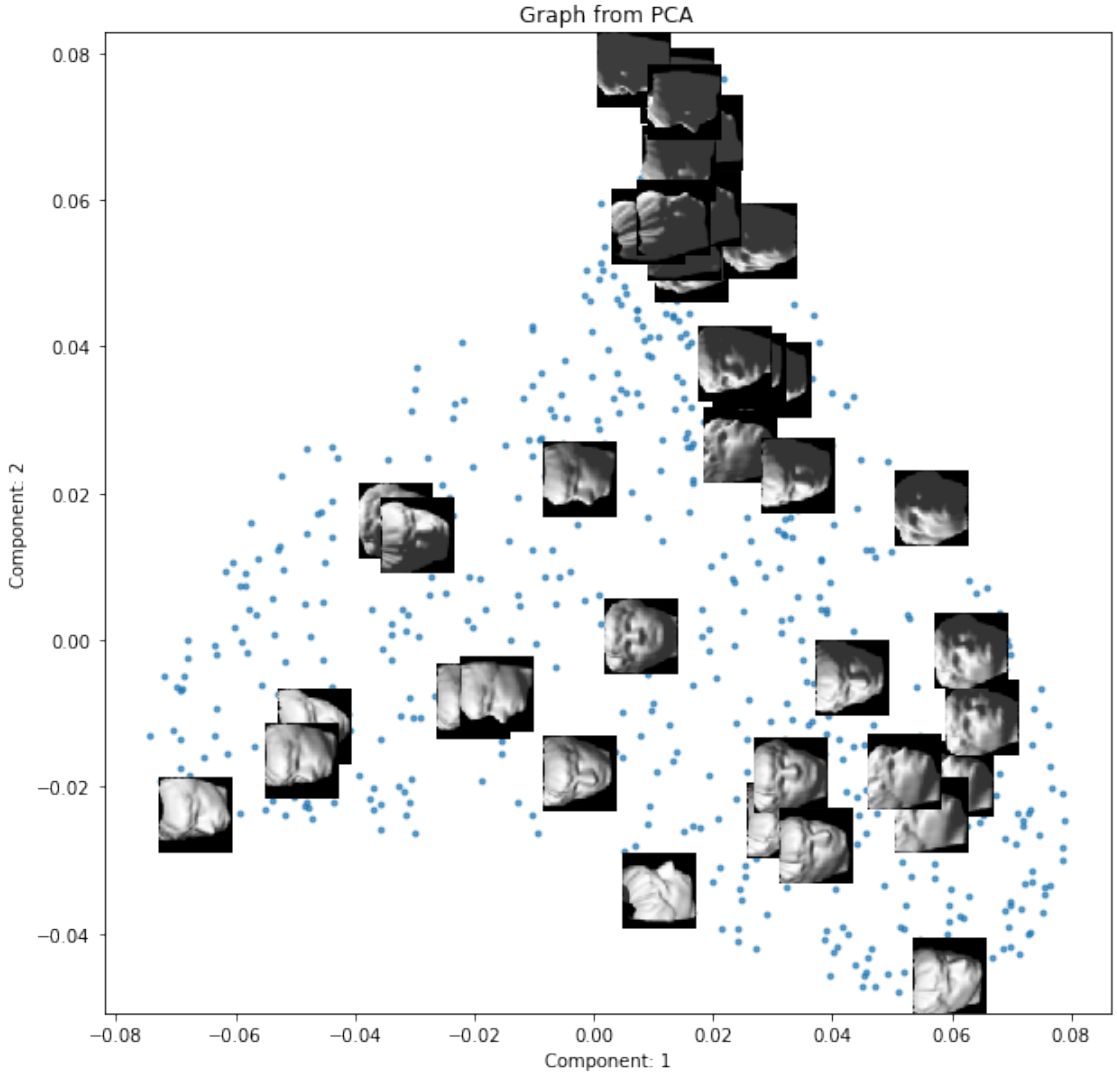


[9]:
```
# Reference: Demo code provided by prof X, along with the hw.

def apply_pca(pca_matrix,K=2):
    m,n= pca_matrix.shape
    mu = np.mean(pca_matrix,axis = 1)
    xc = pca_matrix - mu[:,None]
    C = np.dot(xc,xc.T)/m
    print("---C1----")
    print(C)

    S,W = eigs(C,2)
    S = S.real
    W = W.real
    print ("\n======== Top  Eigen Vectors ==========\n")
    print (W)
    print ("\n======== Top  Eigenvalues ==========\n")
    print (S)
    return S,W
```

[10]:
```
from sklearn.decomposition import PCA
d = distance.cdist(images.T, images.T, 'euclidean')
print("*** details of distances****")
display(d)

S,W=apply_pca(d.T)
drawScatterPlot(W,2,title="Graph from PCA")
plt.show()
```

```
*** details of distances****
array([[ 0.        , 18.83094952,  6.74323967, ..., 21.51126745,
        22.79289298, 18.03618033],
       [18.83094952,  0.        , 19.55307161, ..., 15.07435566,
        21.63387369, 20.97399746],
       [ 6.74323967, 19.55307161,  0.        , ..., 22.82140093,
        23.24040786, 18.68417978],
       ...,
       [21.51126745, 15.07435566, 22.82140093, ...,  0.        ,
        23.3365749 , 17.19515048],
       [22.79289298, 21.63387369, 23.24040786, ..., 23.3365749 ,
         0.        , 20.31353772],
       [18.03618033, 20.97399746, 18.68417978, ..., 17.19515048,
        20.31353772,  0.        ]])


---C1----
[[13.99582915  0.74685118 14.56571246 … -3.00558668 -3.18077021
   3.70773587]
```

```
 [ 0.74685118 15.26490047 -0.82776168 … 18.59427461  0.136154
   4.37421422]
 [14.56571246 -0.82776168 16.19120044 … -7.84411074 -4.06809814
   0.8084214 ]
 …
 [-3.00558668 18.59427461 -7.84411074 … 43.09176571  1.23221152
  19.89812247]
 [-3.18077021  0.136154   -4.06809814 …  1.23221152 20.63188708
   7.84308486]
 [ 3.70773587  4.37421422  0.8084214  … 19.89812247  7.84308486
  23.08388038]]

======== Top  Eigen Vectors ==========

[[-0.00594183  0.03684158]
 [ 0.02986931  0.01825719]
 [-0.01543769  0.03861289]
 …
 [ 0.07635516  0.01113168]
 [ 0.0074551  -0.05913976]
 [ 0.04278859 -0.01019788]]

======== Top  Eigenvalues ==========

[7015.75375415 3728.94324829]
```


Graph from PCA
0.08
0.06
0.04
0.02
0.00
−0.02
−0.04
Component: 2
−0.08
−0.06
−0.04
−0.02
0.00
0.02
0.04
0.06
0.08
Component: 1

Q2

February 23, 2021

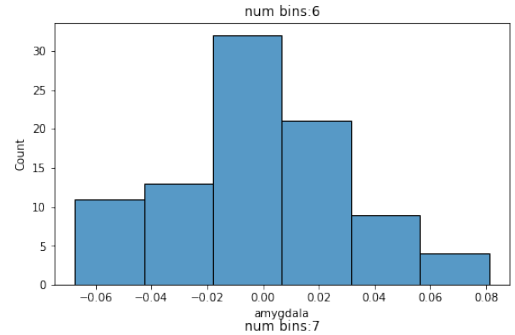
[295]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from scipy.stats import gaussian_kde
from sklearn import preprocessing
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import LeaveOneOut
from sklearn.neighbors import KernelDensity
from sklearn.neighbors import kneighbors_graph
```

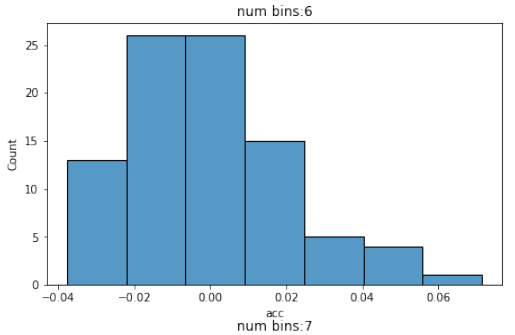
[296]:
```
# 2.a 1-d Histogram
data = pd.read_csv('data/n90pol.csv')
display(data)
a=data.iloc[:, 1].to_numpy()
display(a)

fig, axes = plt.subplots(6, 2,figsize=(16,30))
i=0


for num_of_bins in range(6,12):
    amygdala_minRange=min(data.iloc[:,0])
    amygdala_maxRange=max(data.iloc[:,0])
    amyg_binSize=(amygdala_maxRange - amygdala_minRange)/num_of_bins

    acc_minRange=min(data.iloc[:,1])
    acc_maxRange=max(data.iloc[:,1])
    acc_binSize=(acc_maxRange - acc_minRange)/num_of_bins

    sns.histplot(ax=axes[i][0],data=data, x="amygdala", binwidth=amyg_binSize)
    axes[i][0].set_title("num bins:%s"%num_of_bins)
    axes[i][0].set_xlabel("amygdala")

    sns.histplot(ax=axes[i][1],data=data, x="acc", binwidth=acc_binSize)
    axes[i][1].set_title("num bins:%s"%num_of_bins)
    axes[i][1].set_xlabel("acc")
```

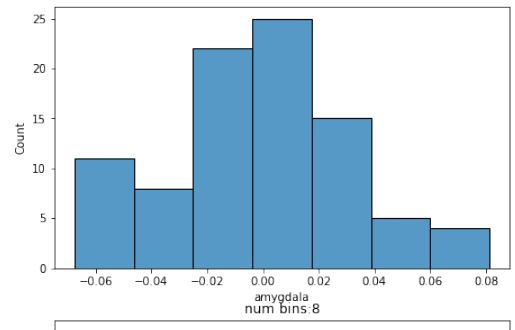
```
    i+=1
```

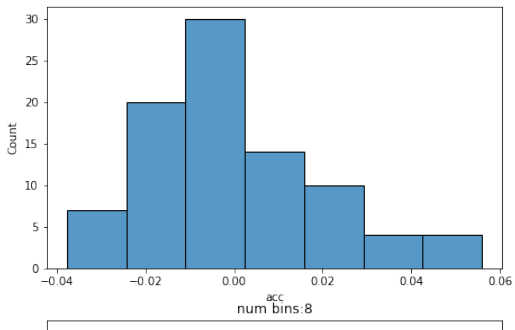
```
    amygdala     acc  orientation
0     0.0051 -0.0286            2
1    -0.0674  0.0007            3
2    -0.0257 -0.0110            3
3     0.0504 -0.0167            2
4     0.0125 -0.0005            5
..       ...     ...          ...
85    0.0174 -0.0242            2
86    0.0251 -0.0087            3
87    0.0676  0.0120            2
88   -0.0097 -0.0239            3
89    0.0374  0.0502            3

[90 rows x 3 columns]
```

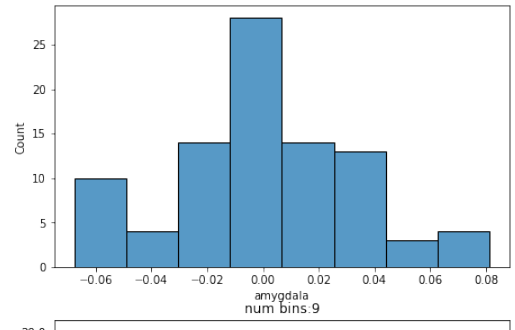
```
array([-0.0286,  0.0007, -0.011 , -0.0167, -0.0005,  0.0266, -0.0052,
        0.0099,  0.0124,  0.0217, -0.0018, -0.0224, -0.027 , -0.0122,
       -0.0085,  0.0031, -0.0218,  0.019 , -0.0054, -0.0107,  0.0334,
        0.0194,  0.0205, -0.0105,  0.0559, -0.0161,  0.0202, -0.0037,
        0.0037, -0.0105,  0.0215,  0.0348, -0.0042, -0.0216, -0.0023,
       -0.006 ,  0.0125, -0.0015, -0.0294, -0.0122, -0.0377,  0.0481,
       -0.0005, -0.0056, -0.0278, -0.0095,  0.0501, -0.0111,  0.0004,
        0.0065,  0.0044, -0.0099,  0.008 , -0.0127, -0.0255,  0.0178,
        0.0073, -0.0104,  0.0079,  0.0074,  0.023 ,  0.0137, -0.0035,
       -0.0065, -0.0005,  0.0363,  0.0005, -0.0107, -0.0145, -0.0101,
       -0.0076,  0.0106, -0.0071,  0.0183, -0.0241, -0.0236, -0.0232,
       -0.0124,  0.0352, -0.0031, -0.0186,  0.0462, -0.031 , -0.0111,
       -0.0148, -0.0242, -0.0087,  0.012 , -0.0239,  0.0502])
```

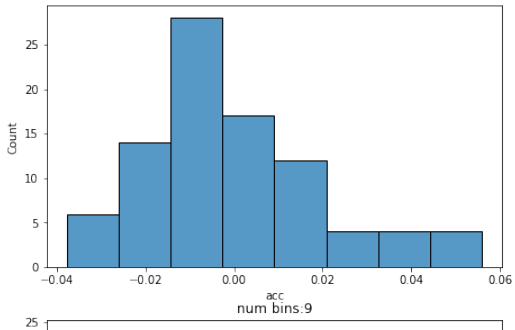

num bins:6
Count
amygdala

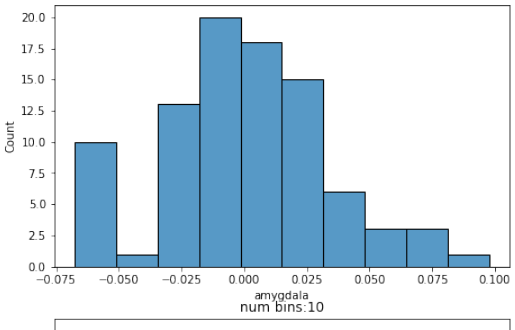

num bins:6
Count
acc

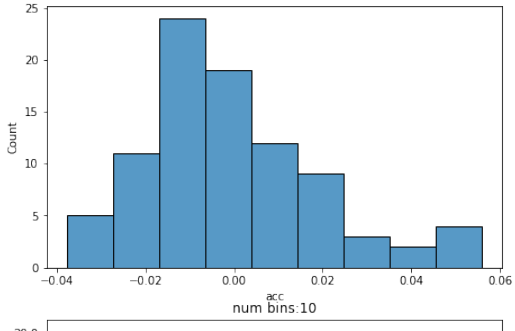

num bins:7
Count
amygdala

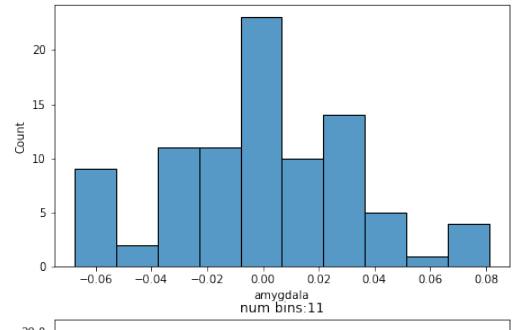

num bins:7
Count
acc

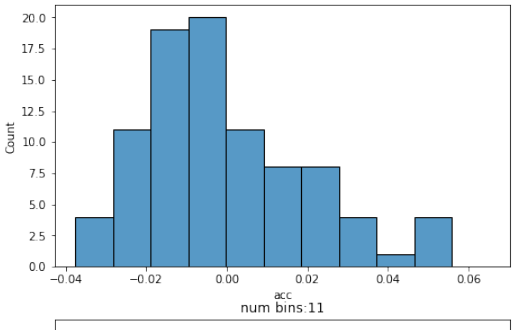

num bins:8
Count
amygdala

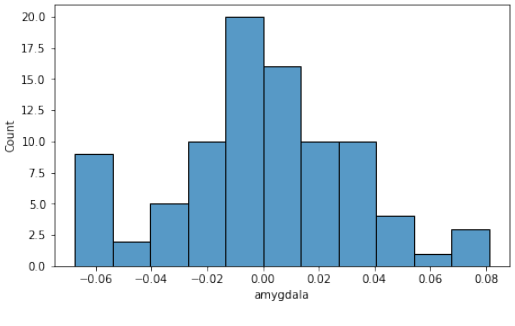

num bins:8
Count
acc

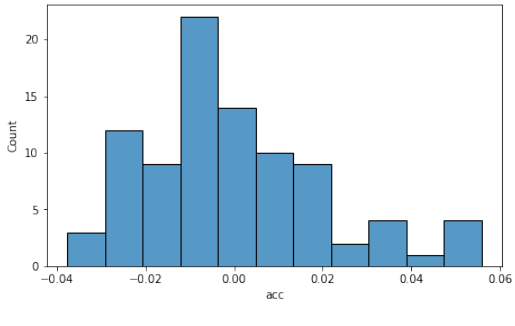

num bins:9
Count
amygdala


num bins:9
Count
acc


num bins:10
Count
amygdala


num bins:10
Count
acc

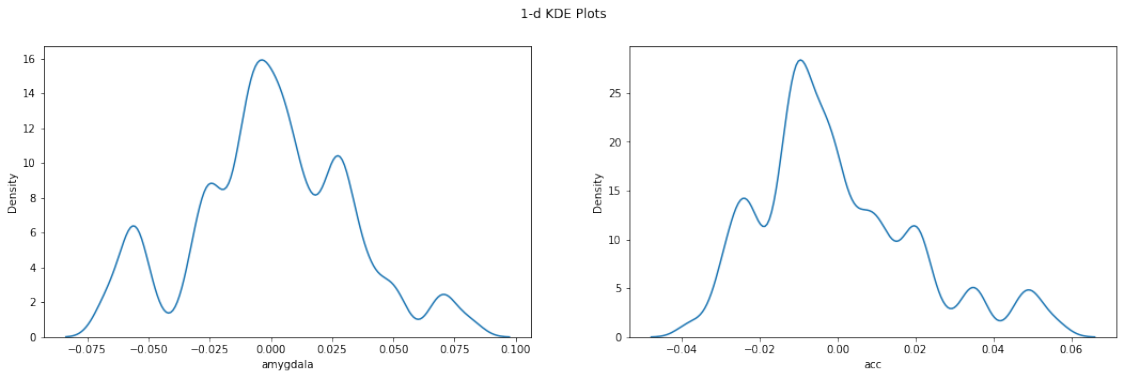

num bins:11
Count
amygdala


num bins:11
Count
acc

[284]:
```
#### 2.a 1-d KDE Plot

## KDE Plots
sigma=np.std(data, ddof=1) / np.sqrt(np.size(data))
n = data.shape[0]
print("std dev and n for amygdala:",sigma["amygdala"], n)

bw=(n ** (-1/5))
print("bandwidth calualted: ",bw)

kde = gaussian_kde(a)
f = kde.covariance_factor()
bw_scipy = f * a.std()
print(a.std())
print("bandwidth caluclated by gaussian kde library method:",bw_scipy)


fig2, axes2 = plt.subplots(1, 2,figsize=(18,5))
fig2.suptitle('1-d KDE Plots')
sns.kdeplot(ax=axes2[0],data=data, x="amygdala",bw_adjust=bw)
sns.kdeplot(ax=axes2[1],data=data, x="acc",bw_adjust=bw)
```

```
std dev and n for amygdala: 0.001984299693119354 90
bandwidth calualted:  0.4065851364889782
0.020321534068902875
bandwidth caluclated by gaussian kde library method: 0.008262433703070296
```

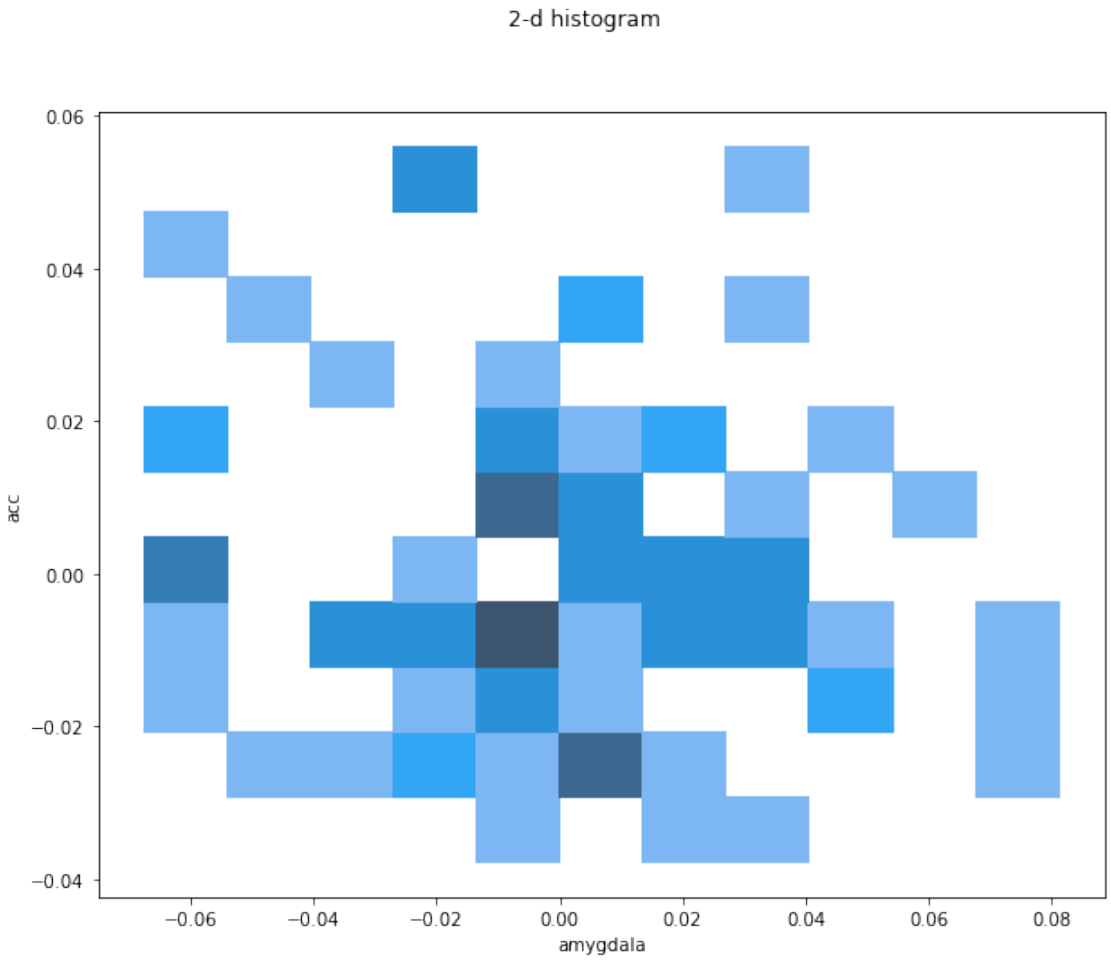
[284]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x127952430>
```



[297]:
```
# 2dimensional histogram
# Q2.b histogram
```

```
fig, axes = plt.subplots(figsize=(10,8))
fig.suptitle('2-d histogram')

sns.histplot(data=data, x="amygdala", y="acc",␣
 ↪binwidth=(amyg_binSize,acc_binSize))
```

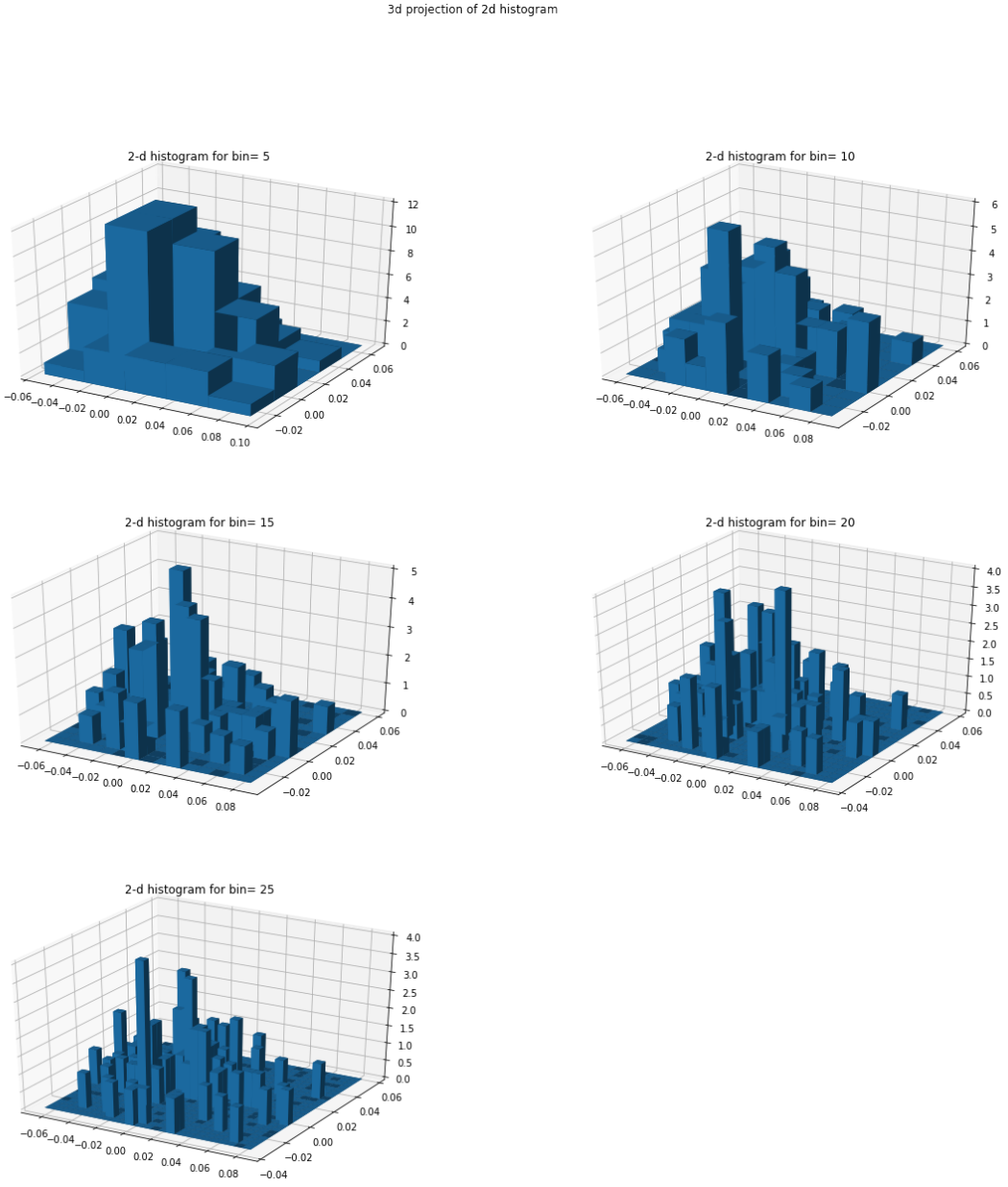
[297]: <matplotlib.axes._subplots.AxesSubplot at 0x1225f3280>



[292]:
```
#Source: demo code provided by prof X.
# Q2c 3d projection of 2d histogram
data = pd.read_csv('data/n90pol.csv',header=0).to_numpy()
print(data.min(0))
print(data.max(0))
# for 2 dimensional data
min_data = data.min(0)
max_data = data.max(0)
fig = plt.figure(figsize=(20,20))
```

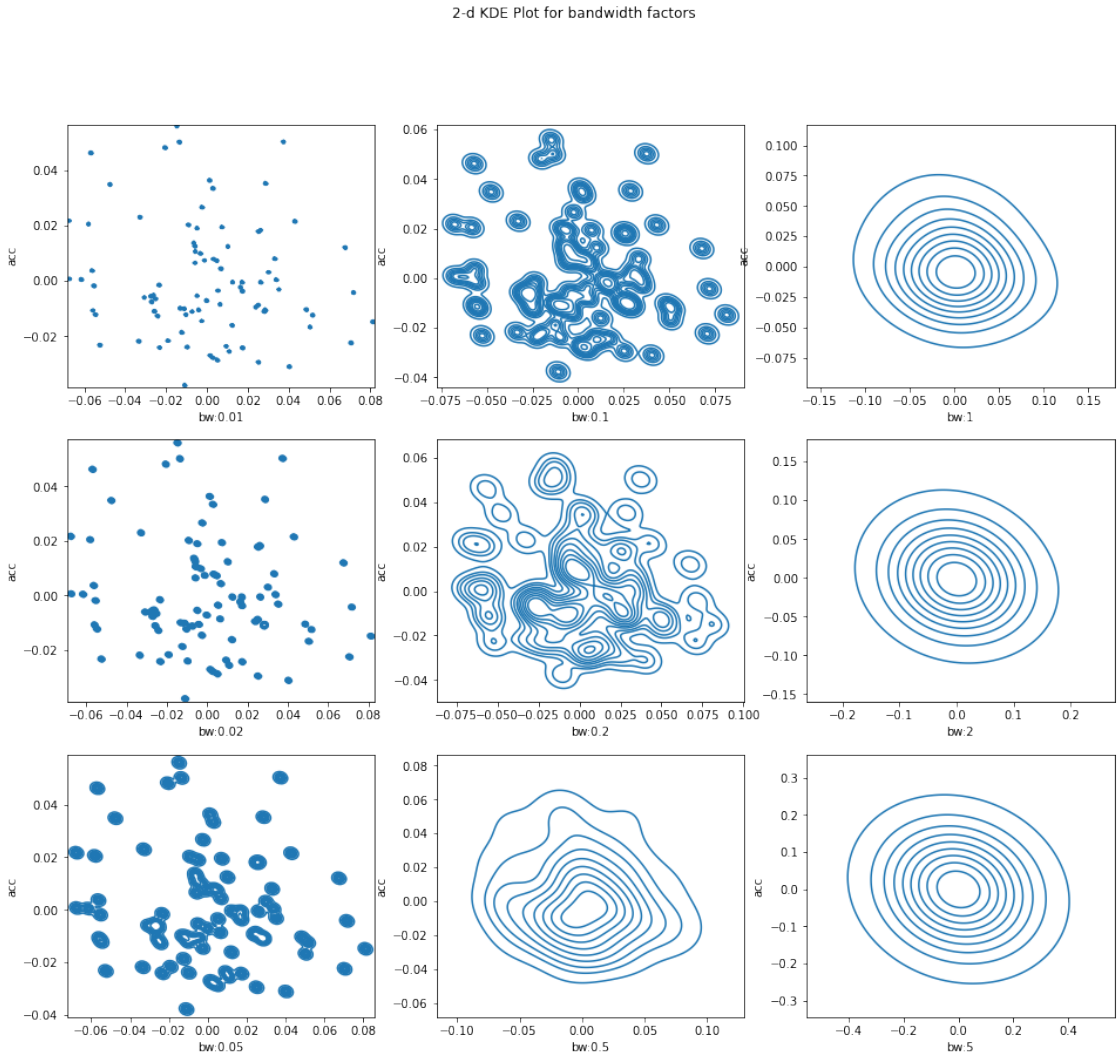
```
fig.suptitle("3d projection of 2d histogram")
i=0
for nbin in (5,10,15,20,25):

    ax = fig.add_subplot(3,2,i+1, projection='3d')
    ax.set_title("2-d histogram for bin= %s"%nbin)
    hist, xedges, yedges = np.histogram2d(data[:,0], data[:,1], bins=nbin)
    xpos, ypos = np.meshgrid(xedges[:-1]+xedges[1:], yedges[:-1]+yedges[1:])
    xpos = xpos.flatten()/2.
    ypos = ypos.flatten()/2.
    zpos = np.zeros_like (xpos)
    dx = xedges [1] - xedges [0]
    dy = yedges [1] - yedges [0]
    dz = hist.flatten()
    ax.bar3d(xpos, ypos, zpos, dx, dy, dz )
    i+=1
```

```
[-0.0676 -0.0377  2.    ]
[0.0812 0.0559 5.    ]
```



[302]:
```
#source: https://jakevdp.github.io/PythonDataScienceHandbook/05.
 ↪13-kernel-density-estimation.html
# Q2c - 2d kde
bws=(0.01,0.02,0.05,0.1,0.2,0.5,1,2,5)
fig, axes = plt.subplots(3,3,figsize=(16,14))
i=0
j=0
for bw in bws:
    fig.suptitle('2-d KDE Plot for bandwidth factors')
    sns.kdeplot(ax=axes[i][j],data=data, x="amygdala", y="acc",bw_method=bw)
```

```
    axes[i][j].set_xlabel("bw:%s"%bw)
    i+=1
    if i==3:
        j+=1
        i=0
```

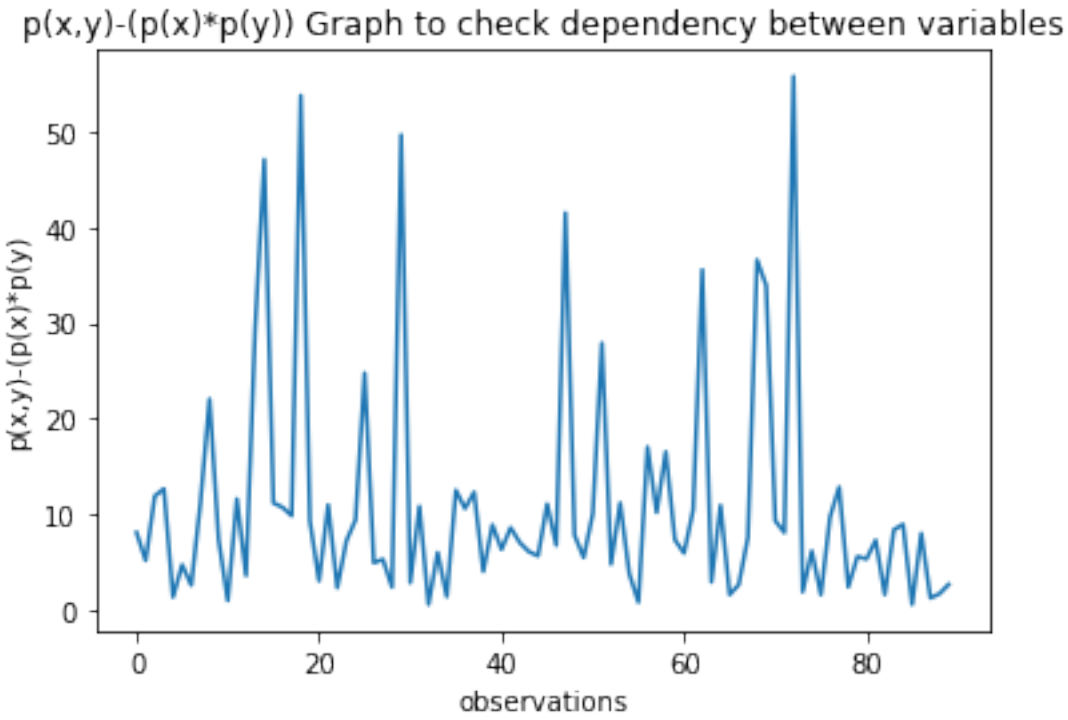


[306]:
```
# q2.c verification if the 2 variables are independent

# df.loc[0:1, 'Name':'Address']
data_2d_array=data.iloc[:,0:2].values
# display(data_2d_array.T)
print(data_2d_array.T.shape)
b=np.atleast_2d(a)
print(type(b))
```

```
print(b.shape)

amygdala_data=np.atleast_2d(data.iloc[:, 0].to_numpy())
acc_data=np.atleast_2d(data.iloc[:, 1].to_numpy())

kde = gaussian_kde(data_2d_array.T)
p_x_y=kde.evaluate(data_2d_array.T)

kde_x = gaussian_kde(np.atleast_2d(amygdala_data))
p_x=kde_x.evaluate(np.atleast_2d(amygdala_data))

kde_y = gaussian_kde(np.atleast_2d(acc_data))
p_y=kde_y.evaluate(np.atleast_2d(acc_data))

print("*** y ***")
print(np.atleast_2d(data.iloc[0:5, 1]))

print("*** x ***")
print(np.atleast_2d(data.iloc[0:5, 0]))


print("*** p_x_y ****")
print(p_x_y.shape)
print(p_x_y[:5])

print("*** p_x ****")
print(p_x[:5])

print("*** p_y ****")
print(p_y[:5])
dif=p_x_y - (p_x * p_y)
print("*** diff ***")
print(dif[:5])


plt.plot(range(0,90),abs(dif))
plt.title('p(x,y)-(p(x)*p(y)) Graph to check dependency between variables')
plt.xlabel('observations')
plt.ylabel('p(x,y)-(p(x)*p(y)')
plt.show()
```

```
(2, 90)
<class 'numpy.ndarray'>
(1, 90)
*** y ***
[[-0.0286  0.0007 -0.011  -0.0167 -0.0005]]
*** x ***
```

```
[[ 0.0051 -0.0674 -0.0257  0.0504  0.0125]]
*** p_x_y ****
(90,)
[115.26710645  54.5068419  169.32980185  71.88165141 208.22954794]
*** p_x ****
[12.19781554  2.70213481  7.68356675  3.44448479 11.03490713]
*** p_y ****
[ 8.79082075 18.28101722 20.49433324 17.20496917 18.98572666]
*** diff ***
[ 8.0382965   5.10906886 11.86022436 12.61939671 -1.27618253]
```

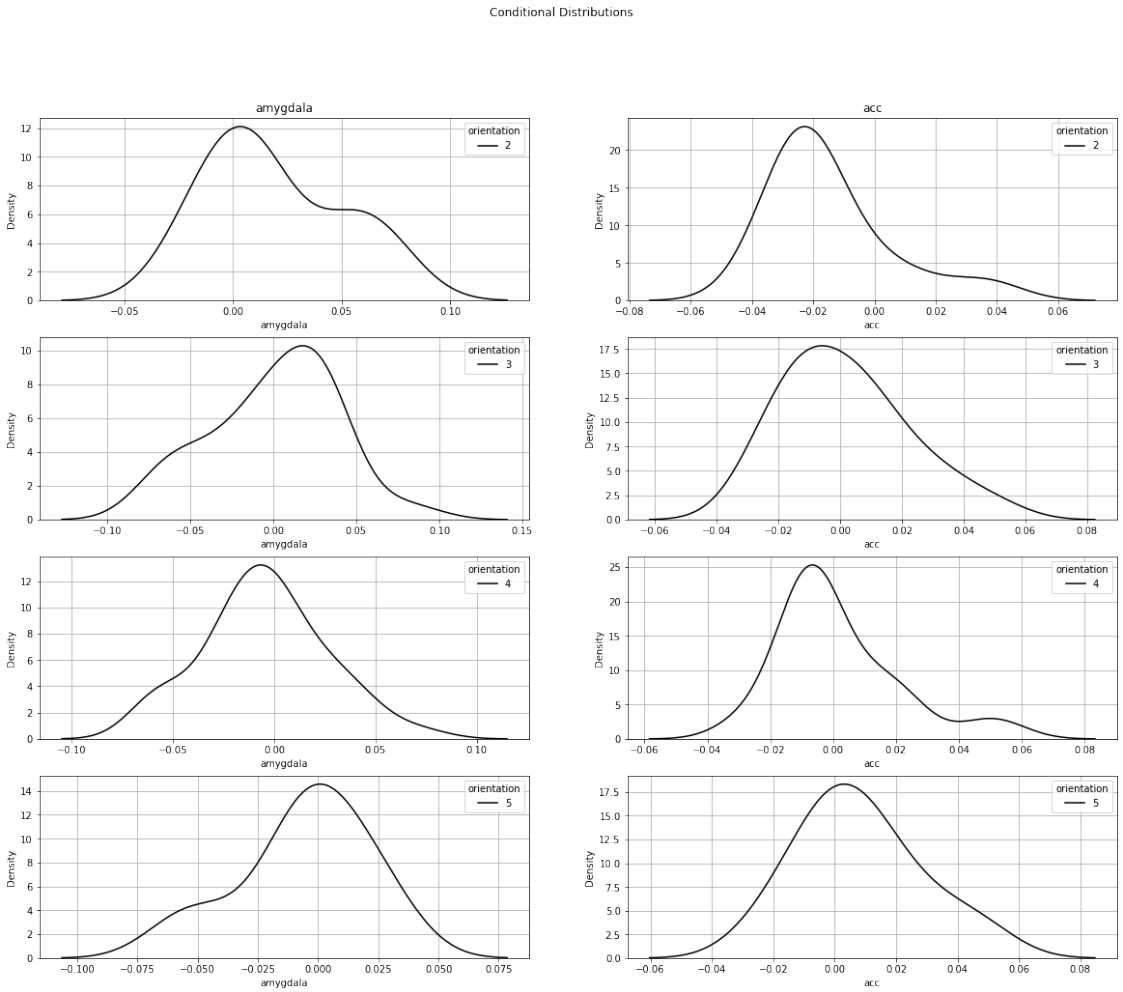


[333]:
```
fig, axes = plt.subplots(4,2,figsize=(20,16))
fig.suptitle("Conditional Distributions")
means=[]
i=0
j=0
# amygdala
for orientation in [2,3,4,5]:
    means.append(np.mean(data[data.orientation==orientation].iloc[:,0]))
    sns.kdeplot(ax=axes[i][j],data=data[data.orientation==orientation],␣
 ↪x='amygdala', hue='orientation',palette='gist_earth')
    axes[i][j].grid(b=True,which='both')
    i+=1
# acc
```

```
i=0
j=1
for orientation in [2,3,4,5]:
    means.append(np.mean(data[data.orientation==orientation].iloc[:,1]))
    sns.kdeplot(ax=axes[i][j],data=data[data.orientation==orientation],␣
 ↪x='acc', hue='orientation',palette='gist_earth')
    axes[i][j].grid(b=True,which='both')
    i+=1
print(means)
axes[0][0].set_title("amygdala")
axes[0][1].set_title("acc")
```

```
[0.01906153846153846, 0.0005875, -0.004719512195121951, -0.005691666666666665,
-0.014769230769230769, 0.0016708333333333338, 0.0013097560975609756,
0.008141666666666667]
```

[333]: Text(0.5, 1.0, 'acc')



[338]:
```
fig, axes = plt.subplots(2,2,figsize=(20,16))
fig.suptitle("Conditional Distributions for 2d")
means=[]
i=0
j=0
# amygdala
for orientation in [2,3,4,5]:
    sns.kdeplot(ax=axes[i][j],data=data[data.orientation==orientation],␣
 ↪x='amygdala',y='acc',hue='orientation',palette='gist_earth')
    axes[i][j].grid(b=True,which='both')
    i+=1
    if i>1:
        j+=1
        i=0
```