Similarity Report for Parth Chopra and Jon Gill

50% Similarity

Matches

Parth Chopra PS4.py

354 - 378 (1) 326 - 350

```
figure, ax = plt.subplots(3, 3, sharex='row', sharey='row')
delta, STEP_SIZE = 0.01, 0.01
final_stress_values = []
iter_no = 1

for i in range(3):
    for j in range(3):
        positions = np.random.rand(len(similarity), 2)
        for iteration in range(100):
            positions -= (compute_gradient(positions, delta) * STEP_SIZE)
        final_stress_values.append(StressCalc(positions, distance))

        pos = 0
        for sport in positions:
```

Jon Gill PS4_Jon_Gill.py

```
delta = 0.01
alpha = 0.01
final_stress_vals = []
iteration = 1

for i in range(3):
    for j in range(3):
        positions = np.random.rand(len(similarity), 2)
        for iteration in range(100):
            positions -= (compute_gradient(positions, delta) * alpha)
        final_stress_vals.append(StressCalc(positions, distance))

        idx = 0
        for sport in positions:
            ax[i][j].plot(sport[0], sport[1], '.')
```

```
            ax[i][j].plot(sport[0], sport[1], '.')
            ax[i][j].text(sport[0], sport[1] + 0.01, sport_names[pos])
            pos += 1

        ax[i][j].set_title(f'Plot of Sports at Final Positions, Iteration #{iter_no}')
        ax[i][j].set_xlabel('Final x-position')
        ax[i][j].set_ylabel('Final y-position')

        iter_no += 1

figure.set_size_inches(15, 15)
figure.savefig('PS4_Q7_1.png')
```

```
            ax[i][j].text(sport[0], sport[1] + 0.01, sport_names[idx])
            idx += 1

        ax[i][j].set_title(f'Plot of Sports at Final Positions, Iteration #{iteration}')
        ax[i][j].set_xlabel('Final x-position')
        ax[i][j].set_ylabel('Final y-position')

        iter_no += 1

figure.set_size_inches(15, 15)
figure.savefig('PS4_Q7_1.png')
```

321 - 336 2 296 - 310

```
#YOUR CODE HERE
figure, ax = plt.subplots()

MDS_distances = np.zeros(distance.shape)

for i in range(len(positions)):
    for j in range(len(positions)):
        MDS_distances[i][j] = EuclideanDistance(positions[i], positions[j])

```

```
figure, ax = plt.subplots()

mds_distances = np.zeros(distance.shape)

for i in range(len(positions)):
    for j in range(len(positions)):
        mds_distances[i][j] = EuclideanDistance(positions[i], positions[j])

```

```
plt.scatter(MDS_distances, distance, s=10,
c='green')
plt.xlabel('Distances obtained by running MDS')
plt.ylabel('Reported distances (d = 1-s)')
plt.title('MDS Distances Plotted Against
Reported Distances')

figure.set_size_inches(10, 6)
figure.savefig('PS4_Q6.png')
```

```
ax.scatter(mds_distances, distance, s=10,
c='blue')
ax.set_xlabel('Distances from running MDS')
ax.set_ylabel('Reported distances $(d = 1-s)$')
ax.set_title('MDS Distances vs. Reported
Distances')

figure.set_size_inches(10, 10)
figure.savefig('PS4_Q6.png')
```

272 - 284 3 249 - 261

```
#YOUR CODE HERE
pos = 0
figure, ax = plt.subplots()

for sport in positions:
    plt.plot(sport[0], sport[1], '.')
    plt.text(sport[0], sport[1] + 0.01,
sport_names[pos])
    pos += 1

plt.title('Plot of Sports at Locations of
Minimized Stress')
plt.xlabel('Final x-position')
plt.ylabel('Final y-position')
```

```
idx = 0
figure, ax = plt.subplots()

for sport in positions:
    ax.plot(sport[0], sport[1], '.')
    ax.text(sport[0], sport[1] + 0.01,
sport_names[idx])
    idx += 1

ax.set_title('Sports at Locations of Minimized
Stress')
ax.set_xlabel('Final x-position')
ax.set_ylabel('Final y-position')
```

```
figure.set_size_inches(10, 6)
figure.savefig('PS4_Q5_2.png')

```

```
figure.set_size_inches(10, 10)
figure.savefig('PS4_Q5_2.png')

```

119 - 129 4 118 - 127

```
#YOUR CODE HERE
def convert_similarity_to_distance(similarity):
    distance = np.zeros(similarity.shape)
    for i in range(len(similarity)):
        for j in range(len(similarity[0])):
            distance[i][j] = 1 - similarity[i]
[j]

    return distance

distance =
convert_similarity_to_distance(similarity)

distance_1d = np.reshape(distance,
len(distance) * len(distance[0]))
similarity_1d = np.reshape(similarity,
len(similarity) * len(similarity[0]))

```

```

def similarity_to_distance(similarity):
    distance = np.zeros(similarity.shape)
    for i in range(len(similarity)):
        for j in range(len(similarity[0])):
            distance[i][j] = 1 - similarity[i]
[j]
    return distance

distance = similarity_to_distance(similarity)
distance_flattened = distance.flatten()
similarity_flattened = similarity.flatten()

figure, ax = plt.subplots()
```

456 - 464 5 416 - 424

```

ax.plot(steps, stress_values, color='red')
```

```

ax.plot(stress_vals, color='red')
```

```python

plt.title('Stress Plotted Over Time')
plt.xlabel('Step Number')
plt.ylabel('Stress Value')
plt.legend(['$a = 0.02$', '$a = 0.05$'])

figure.set_size_inches(10, 6)
figure.savefig('PS4_Q8.png')


```

```python

ax.set_title('Stress Plotted Over Time')
ax.set_xlabel('Step Number')
ax.set_ylabel('Stress Value')
plt.legend(['$a = 0.02$', '$a = 0.05$'])

figure.set_size_inches(10, 10)
figure.savefig('PS4_Q8.png')


```

227 - 233 6 208 - 214

```python

6
    return (StressCalc(plus_delta, distance) - StressCalc(minus_delta, distance)) / (2 * delta)

def compute_gradient(positions, delta):
    gradient_matrix = np.zeros(positions.shape)
    for i in range(len(gradient_matrix)):
        for j in range(len(gradient_matrix[0])):
            gradient_matrix[i][j] = gradient_at_point(positions, i, j, delta)

    return gradient_matrix
```

```python
delta = .01
6
positions = np.random.rand(len(similarity),2)

def compute_gradient(positions, delta):
    gradient_matrix = np.zeros(positions.shape)
    for i in range(len(gradient_matrix)):
        for j in range(len(gradient_matrix[0])):
            plus_delta = positions.copy()
            minus_delta = positions.copy()
            plus_delta[i][j] += delta
```

All Code

Parth Chopra PS4.py

```
#!/usr/bin/env python
# coding: utf-8

# <div style="background-color: #c1f2a5">
#
#
# # PS4
#
# In this problem set, you will implement multidimensional scaling (MDS) from scratch. You may use standard matrix/vector libraries (e.g. numpy) but you must implement two dimensional MDS itself on your own and not use an existing software package. MDS attempts to find an arrangement of points such that the distances between points match human-judged similarities.
#
# # Instructions
#
#
#
# Remember to do your problem set in Python 3. Fill in `#YOUR CODE HERE`.
#
# Make sure:
# - that all plots are scaled in such a way that you can see what is going on (while still respecting specific plotting instructions)
# - that the general patterns are fairly represented.
# - to label all x- and y-axes, and to include a title.
#
# </div>

# In[137]:
```

```
import numpy as np
import matplotlib.pyplot as plt
# to import the data set
from scipy.io import loadmat


# ## Import and examine data
#
# We will be using a data set from Romney, A. K., Brewer, D. D., & Batchelder, W. H. (1993). Predicting Clustering from Semantic Structure. Psychological Science, 4(1), 28-34, via https://faculty.sites.uci.edu/mdlee/similarity-data/. The data set is saved in PS4_dataset.mat, and includes pairwise similarity measures between 21 sports. Make sure that the PS notebook and the data set are in the same directory!
#
# As our first step, we will download and examine the data:
#

# In[138]:


data_set = loadmat('PS4_dataset.mat')
similarity = data_set['similarity']
sport_names = data_set['sport_names']


# As we can see, our data contains information for 21 different sports as listed below:

# In[139]:


print(sport_names)
```

```
# We also have a similarity matrix for each sport, which gives us the psychological similarity of that sport with all the other sports in the data:

# In[140]:


#Look at the first similarity matrix, which corresponds to football's similarity with itself and all other sports
print(similarity[0])


# ## Q1. Visualize similarity [5pts, HELP]

# Plot the "similarity" measures from the data as a heatmap. Don't forget to:
#
#        1)Label the heatmap's rows and columns with the corresponding sport (rotate the x-axis labels by 45 degrees so that the labels are readable)
#
#        2)Add a title to your figure (e.g. similarity)
#
#        3)Add a colorbar. Limit the colobar values between 0 and 1.
#
#        4)Use default colormap
#
#        5)Upload figure PS4_Q1.png to gradescope.
#
# Hint - look up matplotlib's imshow.

# In[155]:


#YOUR CODE HERE
figure, ax = plt.subplots()
```

```
heatmap = ax.imshow(similarity, interpolation='nearest')
ax.set_xticks(range(len(sport_names)))
ax.set_yticks(range(len(sport_names)))
ax.set_xticklabels(sport_names)
ax.set_yticklabels(sport_names)

plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
         rotation_mode="anchor")
plt.title("Psychological similarity between sports")

plt.colorbar(heatmap)
figure.set_size_inches(10, 10)

figure.savefig('PS4_Q1.png')


# ## Q2. Distance [2 pts, SOLO]
#
# To implement MDS, we need a measure of psychological **distance**. The dataset includes measures of similarity, not distance.
#
# Here we will use *d = 1-s* as a  method to transform similarity to distance.
#
# Write a function that converts all similarity measures in the dataset into distances, using the above provided transformation method. Function should return the output called distance (Hint: this variable will be used as an input in some of the functions you'll write in the following questions).
#
# Plot a scatterplot of the dataset's distances (x axis) against their similarity (y axis). Label your figure.
#
# Upload figure PS4_Q2.png to gradescope.
#

```

```
# In[159]:


#YOUR CODE HERE
def convert_similarity_to_distance(similarity):
    distance = np.zeros(similarity.shape)
    for i in range(len(similarity)):
        for j in range(len(similarity[0])):
            distance[i][j] = 1 - similarity[i][j]

    return distance

distance = convert_similarity_to_distance(similarity)

distance_1d = np.reshape(distance, len(distance) * len(distance[0]))
similarity_1d = np.reshape(similarity, len(similarity) * len(similarity[0]))

figure, ax = plt.subplots()

plt.scatter(distance_1d, similarity_1d)
plt.xlabel('Distance')
plt.ylabel('Similarity')
plt.title('Dataset distances vs. similarities')

figure.set_size_inches(10, 6)

figure.savefig('PS4_Q2.png')


# ## Q3. Stress [5 pts, SOLO]
#
```

```
# To perform MDS, we will try to find, for each sport i, a position $p_i=(x_i,y_i)$ in the 2d space that captures the participants’ similarities. To do so, we will build an algorithm that minimizes the stress. We’ll define stress slightly differently than in class- the squared difference between psychological distance $\psi_{ij}= (1-s_{ij})$ and the MDS distance in 2D space:
#
# $$ \mathrm{Stress \ S} = \sum_{i>j} (\psi_{ij} - dist(p_i,p_j))^2$$
#
# Where $\psi$ is the psychological distance between sport i and sport j that was reported by subjects, and *dist(pi,pj)* corresponds to the **Euclidean distance**:$\sqrt{(x_i-x_j)^2 + (y_i-y_j)^2}$
#
# Write a function that computes the Euclidean Distance between two points $p_1$ and $p2$. Then, write a function that takes a $(n,2)$ (n=number of sports) matrix of $(x,y)$ positions for each sport, and computes the stress based on the equation above, using your Euclidean Distance function.
#
# Copy the StressCalc function into gradescope.


# In[143]:



def EuclideanDistance(p1,p2):
    ''' Takes positions defined by p1 and p2, and returns a euclidean distance value (single number).
    Implement EQ equation provided in the question. Hint: if p1=p2, the function should return the value of 0'''
    #YOUR CODE HERE
    return ((p1[0] - p2[0]) ** 2 + (p1[1] - p2[1]) ** 2) ** 0.5


# In[144]:


```

```python
def StressCalc(positions, distance):
    ''' Takes positions (n,2) and (n,n) matrix of distance measures
    (You will use the distance matrix from Q2).
    Uses these distances and the EuclideanDistance function above which computes ED based on positions
    to calculate the Stress between psychological and ED distances, according to the provided formula.'''

    #YOUR CODE HERE
    stress = 0

    for i in range(len(distance)):
        for j in range(i, len(distance)):
            stress += ((distance[i][j] - EuclideanDistance(positions[i], positions[j]))) ** 2

    return stress



# In[145]:


# Test case!
'''
Test case for StressCalc: create an array of positions, where each entry is 1.
Use this positions matrix and distance matrix from Q2 to call StressCalc function
'''

positions = np.ones((len(similarity),2))
print(['Stress value should be 111.57. Output stress value is: ' + str(StressCalc(positions,distance))])


```

```
# ## Q4. Gradient [10 pts, HELP]
# To minimize the stress, we will numerically compute the gradient using a multidimensional version of the simple rule for derivatives:
#
# $$ \frac{df}{dp}(p) = \frac{f(p+\delta)-f(p-\delta)}{2\delta}$$
#
# where $\delta$ takes on a small value, and $f$ is the stress function you wrote in the previous question. To compute the gradient, we will compute this approximate derivative with respect to each coordinate of each point.
#
# Write a function that takes an n-by-2 matrix (n=number of sports) of (x,y) positions for each sport and computes the gradient (i.e. applies the numerical rule above to each coordinate location). This should return an n-by-2 gradient matrix.
#
#
# Use $\delta$ = 0.01
#
# Copy your code into gradescope.
#

# In[180]:


delta = .01
positions = np.random.rand(len(similarity),2)

#YOUR CODE HERE
def gradient_at_point(positions, i, j, delta):
    plus_delta, minus_delta = positions.copy(), positions.copy()

    plus_delta[i][j] += delta
    minus_delta[i][j] -= delta

```

```
    return (StressCalc(plus_delta, distance) - StressCalc(minus_delta, distance)) / (2 * delta)

def compute_gradient(positions, delta):
    gradient_matrix = np.zeros(positions.shape)
    for i in range(len(gradient_matrix)):
        for j in range(len(gradient_matrix[0])):
            gradient_matrix[i][j] = gradient_at_point(positions, i, j, delta)

    return gradient_matrix


# ## Q5.1 MDS [10 pts, HELP]
#
# Write the MDS code: the code that follows a gradient in order to find positions that minimize the stress. Start from a random position, and be sure to take small steps in the direction of the gradient (e.g.  α*gradient, with step size  α=0.01), to find a set of positions that minimizes the stress. Use 100 steps of gradient descent.
#
# Copy your code in gradescope.

# In[181]:


#YOUR CODE HERE
STEP_SIZE = 0.01
print(f'Initial Positions: \n {positions}')
steps, stress_values = [], []

for iteration in range(100):
    positions -= (compute_gradient(positions, delta) * STEP_SIZE)
    step_no, stress_value = iteration + 1, StressCalc(positions, distance)
    print(f'Iteration #{step_no}, Total Stress: {stress_value}')
    steps.append(step_no)
```

```
    stress_values.append(stress_value)

print(f'Final Positions: \n {positions}')


# ## Q5.2 [5 pts, SOLO]
#
# Plot the names of sports at the resulting coordinates.Hint: look up axis.text for plotting the sports names.
#
# Upload PS4_Q5_2.png in gradescope.

# In[182]:


#YOUR CODE HERE
pos = 0
figure, ax = plt.subplots()

for sport in positions:
    plt.plot(sport[0], sport[1], '.')
    plt.text(sport[0], sport[1] + 0.01, sport_names[pos])
    pos += 1

plt.title('Plot of Sports at Locations of Minimized Stress')
plt.xlabel('Final x-position')
plt.ylabel('Final y-position')

figure.set_size_inches(10, 6)
figure.savefig('PS4_Q5_2.png')


# ## Q5.3 [5 pts, SOLO]
# Plot the stress as a function of step number (x axis = step number, y axis= stress).
```

```
#
# Upload PS4_Q5_3 in gradescope.

# In[183]:


#YOUR CODE HERE
figure, ax = plt.subplots()

plt.plot(steps, stress_values, '.')
plt.xlabel('Step Number')
plt.ylabel('Stress Value')
plt.title('Stress Plotted as a Function of Step Number')

figure.set_size_inches(10, 6)
figure.savefig('PS4_Q5_3.png')


# ## Q6. Validation [5pts, SOLO]
#
# Make a scatter plot of the distances obtained by running your MDS function vs. people's reported distances *d=(1-s)*.
#
# Upload PS4_Q6.png to gradescope.
#
# Briefly describe what a good and bad MDS-psychological distance relationship would look like, and whether yours is good or bad. Enter your response in gradescope.
#

# In[184]:


#YOUR CODE HERE
figure, ax = plt.subplots()
```

```
MDS_distances = np.zeros(distance.shape)

for i in range(len(positions)):
    for j in range(len(positions)):
        MDS_distances[i][j] = EuclideanDistance(positions[i], positions[j])


plt.scatter(MDS_distances, distance, s=10, c='green')
plt.xlabel('Distances obtained by running MDS')
plt.ylabel('Reported distances (d = 1-s)')
plt.title('MDS Distances Plotted Against Reported Distances')

figure.set_size_inches(10, 6)
figure.savefig('PS4_Q6.png')


# Intuitively, it makes sense for a good MDS-psychological distance relationship to show the points lying along some sort of trend line, because this indicates that distances obtained by running our MDS function are closely related to people's reported distances. In other words, the points on the scatter plot should have similar x- and y-values and be pointing to the top right, indicating a positive correlation. From the scatter plot, we can see that our relationship doesn't seem too bad, though there doesn't seem to be any strong positive correlation. That being said, the points aren't randomly distributed and seem to follow a very weak positive trend, which is a good sign.

# ## Q7.1 Iterating MDS [3pts, SOLO]
#
# Run your MDS code 9 times, and plot the corresponding final positions in a figure with subplots in a 3x3 grid. Indicate the code iteration number in each subplot title. Scale the figure size using figsize=(15,15).
#
# Are they all the same or not? Why? Enter your response in gradescope.
#
```

```
# Upload PS4_Q7_1.png in gradescope.

# In[189]:


#YOUR CODE HERE
figure, ax = plt.subplots(3, 3, sharex='row', sharey='row')
delta, STEP_SIZE = 0.01, 0.01
final_stress_values = []
iter_no = 1

for i in range(3):
    for j in range(3):
        positions = np.random.rand(len(similarity), 2)
        for iteration in range(100):
            positions -= (compute_gradient(positions, delta) * STEP_SIZE)
        final_stress_values.append(StressCalc(positions, distance))

        pos = 0
        for sport in positions:
            ax[i][j].plot(sport[0], sport[1], '.')
            ax[i][j].text(sport[0], sport[1] + 0.01, sport_names[pos])
            pos += 1

        ax[i][j].set_title(f'Plot of Sports at Final Positions, Iteration #{iter_no}')
        ax[i][j].set_xlabel('Final x-position')
        ax[i][j].set_ylabel('Final y-position')

        iter_no += 1

figure.set_size_inches(15, 15)
figure.savefig('PS4_Q7_1.png')


```

```
# These plots are not all the same. This is expected however, since we are starting at random initial positions to begin with. This means that when we perform gradient descent, we will almost surely end with different final values that minimize the final stress value.

# ## Q7.2 Best representation [3pts, SOLO]
# In another figure, plot the final stress value as a function of the MDS iteration (9) in the previous question. If you wanted to pick the best final representation based on this plot, how would you do it? What criteria would you use? Which iteration is your best?
#
# Enter your answer in gradescope.
#
# Upload PS4_Q7_2.png.

# In[199]:


#YOUR CODE HERE
figure, ax = plt.subplots()

plt.plot([i+1 for i in range(9)], final_stress_values, 'bo')
plt.title('Final Stress Value Plotted as a Function of Iteration #')
plt.xlabel('Iteration Number')
plt.ylabel('Final Stress Value')

figure.set_size_inches(10, 6)
figure.savefig('PS4_Q7_2.png')


# If I wanted to pick the best final representation, I would pick the iteration which resulted in the minimum final stress value. The stress function is our equivalent of a loss function, which indicates how far our representation is from the optimal stress value for our initial sport positions. A low stress value indicates that our measure of "similarity" between two sports is close to the given similarity values. Iteration 1 is my best.

```

```
# ## Q7.3 [4pts, SOLO]
# Do your best results agree with your intuitions about how this domain might be organized? Why or why not? Answer in 2-3 sentences.
#
# Enter your response in gradescope.
#

# Yes, my best results with my intuitions of how these sports should be organized. We can notice some type of clustering happening here, where ball sports like volleyball, softball, and and basketball have very similar final positions, and other sports which are not very similar (like track and boxing) are located far from one another. This is interesting because clustering is the natural way to organize sports like these, i.e. we have categories like 'ball sports' and 'water sports' for a reason.

# ## Q8 [5pts, SOLO]
#
# Run MDS 2 times, with 2 different step sizes (α=.02 and  α=.05). Plot Stress over time for each run in the same plot. Don't forget to add a legend,labeling which MDS step size the line refers to, in addition to the usual axis labels and title. What happens if you use a bigger step in your MDS? Why?
#
#
# Enter your answer in gradescope.
#
# Upload PS4_Q8.png.

# In[208]:


#YOUR CODE HERE
figure, ax = plt.subplots()
delta = 0.01
init_positions = np.random.rand(len(similarity), 2)

```

```
# Step size = 0.02
positions = init_positions.copy()
steps, stress_values = [], []
STEP_SIZE = 0.02

for iteration in range(100):
    positions -= (compute_gradient(positions, delta) * STEP_SIZE)
    step_no, stress_value = iteration + 1, StressCalc(positions, distance)
    steps.append(step_no)
    stress_values.append(stress_value)

ax.plot(steps, stress_values, color='blue')

# Step size = 0.05
positions = init_positions.copy()
steps, stress_values = [], []
STEP_SIZE = 0.05

for iteration in range(100):
    positions -= (compute_gradient(positions, delta) * STEP_SIZE)
    step_no, stress_value = iteration + 1, StressCalc(positions, distance)
    steps.append(step_no)
    stress_values.append(stress_value)

ax.plot(steps, stress_values, color='red')

plt.title('Stress Plotted Over Time')
plt.xlabel('Step Number')
plt.ylabel('Stress Value')
plt.legend(['$a = 0.02$', '$a = 0.05$'])

figure.set_size_inches(10, 6)
figure.savefig('PS4_Q8.png')

```

5

```
# If we use a bigger step size, we see that the plot of stress over time follows a much less well defined path, and seems to actually work against finding the optimal stress value. This is because a larger step size may not allow gradient descent to converge properly and find a global minimum of the overall stress function, causing our values of stress over time to vastly oscillate and differ as we perform more iterations. This is why determining the 'ideal' step size when performing gradient descent is so integral, as it may be the difference between convergence and non-convergence.

# <div style="background-color: #c1f2a5">
#
# # Submission
#
# When you're done with your problem set, do the following:
# - Upload your answers in Gradescope's PS4.
# - Upload your code as .py file in PS4-code in Gradescope (To convert the notebook into .py file click on File > Download as > Python (.py)).
#
#
#
#
# </div>

# In[ ]:
```

Jon Gill PS4_Jon_Gill.py

```
#!/usr/bin/env python
# coding: utf-8
```

```

# <div style="background-color: #c1f2a5">
#
#
# # PS4
#
# In this problem set, you will implement multidimensional scaling (MDS) from scratch. You may use standard matrix/vector libraries (e.g. numpy) but you must implement two dimensional MDS itself on your own and not use an existing software package. MDS attempts to find an arrangement of points such that the distances between points match human-judged similarities.
#
# # Instructions
#
#
#
# Remember to do your problem set in Python 3. Fill in `#YOUR CODE HERE`.
#
# Make sure:
# - that all plots are scaled in such a way that you can see what is going on (while still respecting specific plotting instructions)
# - that the general patterns are fairly represented.
# - to label all x- and y-axes, and to include a title.
#
# </div>

# In[1]:


import numpy as np
import matplotlib.pyplot as plt
# to import the data set
from scipy.io import loadmat

```

```

# ## Import and examine data
#
# We will be using a data set from Romney, A. K., Brewer, D. D., & Batchelder, W. H. (1993). Predicting Clustering from Semantic Structure. Psychological Science, 4(1), 28-34, via https://faculty.sites.uci.edu/mdlee/similarity-data/. The data set is saved in PS4_dataset.mat, and includes pairwise similarity measures between 21 sports. Make sure that the PS notebook and the data set are in the same directory!
#
# As our first step, we will download and examine the data:
#

# In[2]:


data_set = loadmat('PS4_dataset.mat')
similarity = data_set['similarity']
sport_names = data_set['sport_names']


# As we can see, our data contains information for 21 different sports as listed below:

# In[3]:


print(sport_names)


# We also have a similarity matrix for each sport, which gives us the psychological similarity of that sport with all the other sports in the data:

# In[4]:


```

```
#Look at the first similarity matrix, which corresponds to football's similarity with itself and all other sports
print(similarity[0])


# ## Q1. Visualize similarity [5pts, HELP]

# Plot the "similarity" measures from the data as a heatmap. Don't forget to:
#
#        1)Label the heatmap's rows and columns with the corresponding sport (rotate the x-axis labels by 45 degrees so that the labels are readable)
#
#        2)Add a title to your figure (e.g. similarity)
#
#        3)Add a colorbar. Limit the colobar values between 0 and 1.
#
#        4)Use default colormap
#
#        5)Upload figure PS4_Q1.png to gradescope.
#
# Hint - look up matplotlib's imshow.

# In[32]:


figure, ax = plt.subplots()
heatmap = ax.imshow(similarity, interpolation='nearest')
num_sports = len(sport_names)
ax.set_xticks(range(num_sports))
ax.set_xticklabels(sport_names)
ax.set_yticks(range(num_sports))
ax.set_yticklabels(sport_names)
ax.set_title("Psychological Similarity Between Sports")
labels = ax.get_xticklabels()
```

```
heat_map = ax.imshow(similarity, interpolation='nearest')

plt.setp(labels, rotation=45, ha="right", rotation_mode="anchor")
plt.colorbar(heat_map)

figure.set_size_inches(15, 15)
figure.savefig('PS4_Q1.png')


# ## Q2. Distance [2 pts, SOLO]
#
# To implement MDS, we need a measure of psychological **distance**. The dataset includes measures of similarity, not distance.
#
# Here we will use *d = 1-s* as a  method to transform similarity to distance.
#
# Write a function that converts all similarity measures in the dataset into distances, using the above provided transformation method. Function should return the output called distance (Hint: this variable will be used as an input in some of the functions you'll write in the following questions).
#
# Plot a scatterplot of the dataset's distances (x axis) against their similarity (y axis). Label your figure.
#
# Upload figure PS4_Q2.png to gradescope.
#

# In[30]:


def similarity_to_distance(similarity):
    distance = np.zeros(similarity.shape)
    for i in range(len(similarity)):
        for j in range(len(similarity[0])):
```

```
            distance[i][j] = 1 - similarity[i][j]
    return distance

distance = similarity_to_distance(similarity)
distance_flattened = distance.flatten()
similarity_flattened = similarity.flatten()

figure, ax = plt.subplots()
plt.scatter(distance_flattened, similarity_flattened)
ax.set_xlabel('Distance')
ax.set_ylabel('Similarity')
ax.set_title('Distances vs. Similarities')

figure.set_size_inches(10, 10)
figure.savefig('PS4_Q2.png')


# ## Q3. Stress [5 pts, SOLO]
#
# To perform MDS, we will try to find, for each sport i, a position $p_i=(x_i,y_i)$ in the 2d space that captures the participants’ similarities. To do so, we will build an algorithm that minimizes the stress. We’ll define stress slightly differently than in class- the squared difference between psychological distance $\psi_{ij}= (1-s_{ij})$ and the MDS distance in 2D space:
#
# $$ \mathrm{Stress \ S} = \sum_{i\neq j} (\psi_{ij} - dist(p_i,p_j))^2$$
#
# Where $\psi$ is the psychological distance between sport i and sport j that was reported by subjects, and *dist(pi,pj)* corresponds to the **Euclidean distance**:$\sqrt{(x_i-x_j)^2 + (y_i-y_j)^2}$
#
```

```
# Write a function that computes the Euclidean Distance between two points $p_1$ and $p2$. Then, write a function that takes a $(n,2)$ (n=number of sports) matrix of $(x,y)$ positions for each sport, and computes the stress based on the equation above, using your Euclidean Distance function.
#
# Copy the StressCalc function into gradescope.

# In[33]:


def EuclideanDistance(p1,p2):
    ''' Takes positions defined by p1 and p2, and returns a euclidean distance value (single number).
    Implement EQ equation provided in the question. Hint: if p1=p2, the function should return the value of 0'''
    return pow(pow((p1[0] - p2[0]), 2) + pow((p1[1] - p2[1]), 2), 0.5)


# In[37]:


def StressCalc(positions, distance):
    ''' Takes positions (n,2) and (n,n) matrix of distance measures
    (You will use the distance matrix from Q2).
    Uses these distances and the EuclideanDistance function above which computes ED based on positions
    to calculate the Stress between psychological and ED distances, according to the provided formula.'''
    stress = 0
    for i in range(len(distance)):
        for j in range(i, len(distance)):
            d = EuclideanDistance(positions[i], positions[j])
            stress += pow((distance[i][j] - d), 2)
    return stress
```

```
# In[38]:


# Test case!
'''
Test case for StressCalc: create an array of positions, where each entry is 1.
Use this positions matrix and distance matrix from Q2 to call StressCalc function
'''

positions = np.ones((len(similarity),2))
print(['Stress value should be 111.57. Output stress value is: ' +
str(StressCalc(positions,distance))])


# ## Q4. Gradient [10 pts, HELP]
# To minimize the stress, we will numerically compute the gradient using a multidimensional version of the simple rule for derivatives:
#
# $$ \frac{df}{dp}(p) = \frac{f(p+\delta)-f(p-\delta)}{2\delta}$$
#
# where $\delta$ takes on a small value, and $f$ is the stress function you wrote in the previous question. To compute the gradient, we will compute this approximate derivative with respect to each coordinate of each point.
#
# Write a function that takes an n-by-2 matrix (n=number of sports) of (x,y) positions for each sport and computes the gradient (i.e. applies the numerical rule above to each coordinate location). This should return an n-by-2 gradient matrix.
#
#
# Use $\delta$ = 0.01
#
# Copy your code into gradescope.
```

```
#

# In[50]:


delta = .01
positions = np.random.rand(len(similarity),2)

def compute_gradient(positions, delta):
    gradient_matrix = np.zeros(positions.shape)
    for i in range(len(gradient_matrix)):
        for j in range(len(gradient_matrix[0])):
            plus_delta = positions.copy()
            minus_delta = positions.copy()
            plus_delta[i][j] += delta
            minus_delta[i][j] -= delta
            diff = (StressCalc(plus_delta, distance) - StressCalc(minus_delta, distance))
            gradient_matrix[i][j] = diff / (2 * delta)
    return gradient_matrix


# ## Q5.1 MDS [10 pts, HELP]
#
# Write the MDS code: the code that follows a gradient in order to find positions that minimize the stress. Start from a random position, and be sure to take small steps in the direction of the gradient (e.g.  α*gradient, with step size  α=0.01), to find a set of positions that minimizes the stress. Use 100 steps of gradient descent.
#
# Copy your code in gradescope.

# In[51]:


alpha = 0.01
```

```
stress_vals = []
for iteration in range(1, 101):
    positions -= (compute_gradient(positions, delta) * alpha)
    stress_val = StressCalc(positions, distance)
    stress_vals.append(stress_val)


# ## Q5.2 [5 pts, SOLO]
#
# Plot the names of sports at the resulting coordinates.Hint: look up axis.text for plotting the sports names.
#
# Upload PS4_Q5_2.png in gradescope.

# In[56]:


idx = 0
figure, ax = plt.subplots()

for sport in positions:
    ax.plot(sport[0], sport[1], '.')
    ax.text(sport[0], sport[1] + 0.01, sport_names[idx])
    idx += 1

ax.set_title('Sports at Locations of Minimized Stress')
ax.set_xlabel('Final x-position')
ax.set_ylabel('Final y-position')

figure.set_size_inches(10, 10)
figure.savefig('PS4_Q5_2.png')


# ## Q5.3 [5 pts, SOLO]
```

3

```
# Plot the stress as a function of step number (x axis = step number, y axis= stress).
#
# Upload PS4_Q5_3 in gradescope.

# In[60]:


figure, ax = plt.subplots()

ax.plot(stress_values, '.')
ax.set_xlabel('Step Number')
ax.set_ylabel('Stress')
ax.set_title('Stress vs. Step Number')

figure.set_size_inches(10, 10)
figure.savefig('PS4_Q5_3.png')


# ## Q6. Validation [5pts, SOLO]
#
# Make a scatter plot of the distances obtained by running your MDS function vs. people's reported distances *d=(1-s)*.
#
# Upload PS4_Q6.png to gradescope.
#
# Briefly describe what a good and bad MDS-psychological distance relationship would look like, and whether yours is good or bad. Enter your response in gradescope.
#

# In[67]:


figure, ax = plt.subplots()

```

2

```
mds_distances = np.zeros(distance.shape)

for i in range(len(positions)):
    for j in range(len(positions)):
        mds_distances[i][j] = EuclideanDistance(positions[i], positions[j])

ax.scatter(mds_distances, distance, s=10, c='blue')
ax.set_xlabel('Distances from running MDS')
ax.set_ylabel('Reported distances $(d = 1-s)$')
ax.set_title('MDS Distances vs. Reported Distances')

figure.set_size_inches(10, 10)
figure.savefig('PS4_Q6.png')


# ## Q7.1 Iterating MDS [3pts, SOLO]
#
# Run your MDS code 9 times, and plot the corresponding final positions in a figure with subplots in a 3x3 grid. Indicate the code iteration number in each subplot title. Scale the figure size using figsize=(15,15).
#
# Are they all the same or not? Why? Enter your response in gradescope.
#
# Upload PS4_Q7_1.png in gradescope.

# In[81]:


figure, ax = plt.subplots(3, 3, sharex='row', sharey='row')
delta = 0.01
alpha = 0.01
final_stress_vals = []
iteration = 1

```

```
for i in range(3):
    for j in range(3):
        positions = np.random.rand(len(similarity), 2)
        for iteration in range(100):
            positions -= (compute_gradient(positions, delta) * alpha)
        final_stress_vals.append(StressCalc(positions, distance))

        idx = 0
        for sport in positions:
            ax[i][j].plot(sport[0], sport[1], '.')
            ax[i][j].text(sport[0], sport[1] + 0.01, sport_names[idx])
            idx += 1

        ax[i][j].set_title(f'Plot of Sports at Final Positions, Iteration #{iteration}')
        ax[i][j].set_xlabel('Final x-position')
        ax[i][j].set_ylabel('Final y-position')

        iter_no += 1

figure.set_size_inches(15, 15)
figure.savefig('PS4_Q7_1.png')


# ## Q7.2 Best representation [3pts, SOLO]
# In another figure, plot the final stress value as a function of the MDS iteration (9) in the previous question. If you wanted to pick the best final representation based on this plot, how would you do it? What criteria would you use? Which iteration is your best?
#
# Enter your answer in gradescope.
#
# Upload PS4_Q7_2.png.

# In[76]:

```

```

figure, ax = plt.subplots()

ax.plot([i for i in range(1,10)], final_stress_values, 'go')
ax.set_title('Final Stress Value vs. Iteration #')
ax.set_xlabel('Iteration #')
ax.set_ylabel('Final Stress Value')

figure.set_size_inches(10, 10)
figure.savefig('PS4_Q7_2.png')


# ## Q7.3 [4pts, SOLO]
# Do your best results agree with your intuitions about how this domain might be organized? Why or why not? Answer in 2-3 sentences.
#
# Enter your response in gradescope.
#

# ## Q8 [5pts, SOLO]
#
# Run MDS 2 times, with 2 different step sizes (α=.02 and  α=.05). Plot Stress over time for each run in the same plot. Don't forget to add a legend,labeling which MDS step size the line refers to, in addition to the usual axis labels and title. What happens if you use a bigger step in your MDS? Why?
#
#
# Enter your answer in gradescope.
#
# Upload PS4_Q8.png.

# In[80]:


```

```
figure, ax = plt.subplots()
delta = 0.01
init_positions = np.random.rand(len(similarity), 2)

positions = init_positions.copy()
stress_vals = []
alpha = 0.02

for iteration in range(1, 101):
    positions -= (compute_gradient(positions, delta) * alpha)
    stress_val = StressCalc(positions, distance)
    stress_vals.append(stress_val)

ax.plot(stress_vals, color='green')

positions = init_positions.copy()
stress_vals = []
alpha = 0.05

for iteration in range(1, 101):
    positions -= (compute_gradient(positions, delta) * alpha)
    stress_val = StressCalc(positions, distance)
    stress_vals.append(stress_val)

ax.plot(stress_vals, color='red')

ax.set_title('Stress Plotted Over Time')
ax.set_xlabel('Step Number')
ax.set_ylabel('Stress Value')
plt.legend(['$a = 0.02$', '$a = 0.05$'])

figure.set_size_inches(10, 10)
figure.savefig('PS4_Q8.png')
```

5

```
# <div style="background-color: #c1f2a5">
#
# # Submission
#
# When you're done with your problem set, do the following:
# - Upload your answers in Gradescope's PS4.
# - Upload your code as .py file in PS4-code in Gradescope (To convert the notebook into .py file click on File > Download as > Python (.py)).
#
#
#
#
# </div>

# In[ ]:
```