Assignment 5: Matrix as a Linear Transformation

DTSC 680: Applied Machine Learning

Name: Juliana Meirelles

The purpose of this assignment is for you to gain experience working with matrices and vectors, as well as to understand the idea that a matrix can be viewed as a ***linear transformation***. This requires us to think of a matrix $\mathbf{A}$ as an object that "acts" (or "operates") on a vector $\mathbf{x}$ by multiplication to produce a new vector called $\mathbf{Ax}$. The original vector is $\mathbf{x}$, and the transformed vector is $\mathbf{Ax}$. In this assignment you will transform an ensemble of vectors using several different matrices (i.e. several different transformations) and observe the results.

Before you continue, take a look at this video from 3Blue1Brown: Linear transformations and matrices

In Assignment 4, you computed matrix-vector multiplications by hand for just a single vector - that is, you computed $\mathbf{b}$ as $\mathbf{Ax} = \mathbf{b}$ manually. For instance, you might have computed the following by hand:

$$\begin{bmatrix} 4 & -3 & 1 & 3 \\ 2 & 0 & 5 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 5 \\ 8 \end{bmatrix}$$

$$\begin{bmatrix} 4 & -3 & 1 & 3 \\ 2 & 0 & 5 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 4 \\ -1 \\ 3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

These equations say that multiplication by the matrix $\mathbf{A}$ transforms the vector $\begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}^T$ into $\begin{bmatrix} 5 & 8 \end{bmatrix}^T$ and transforms the vector $\begin{bmatrix} 1 & 4 & -1 & 3 \end{bmatrix}^T$ into $\begin{bmatrix} 0 & 0 \end{bmatrix}^T$. (Note: The superscript T denotes the transpose, which essentially means "the rows become columns, and the columns become rows".)

Suppose that we wanted to perform this transformation not for just one vector (i.e. one single data point), but for many vectors (i.e. many data points) at the same time. We could accomplish this by thinking of our vector now as a matrix of vectors, and we will apply the linear transformation to each of these vectors in the matrix. (So, now we will be performing matrix multiplication.) For example, to apply the linear transformation to the above two vectors at the same time, we would write:

$$\begin{bmatrix} 4 & -3 & 1 & 3 \\ 2 & 0 & 5 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 1 \\ 1 & 4 \\ 1 & -1 \\ 1 & 3 \end{bmatrix} = \begin{bmatrix} 5 & 0 \\ 8 & 0 \end{bmatrix}$$

So, what will we be doing in this assignment? You will plot a simple dataset of points, where each point can be thought of as an individual vector in 2D Cartesian space. You will then perform several different geometric linear transformations in real 2D space on this data (i.e. "do some matrix multiplication with a supplied matrix"). You will then plot the original data and the transformed data, and observe the transformation. At the end, you will be asked to correctly label each of the different linear transformation matrices with its name based on the transformation results.

The data you will read in has the typical format for 2D spatial data (typically for our class, that is) - there are three columns for the x, y, and z coordinates. For example, the data you import will have the following form (Actually, this is not true! Your data will be in 2 dimensions, but I will show you 3 here.):

$$\begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \\ \vdots & \vdots & \vdots \\ x_n & y_n & z_n \end{bmatrix}$$

We can view each data point as a vector. So, the first vector would be

$$\mathbf{v} = \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix}$$

Do you see how we need a column vector for the matrix multiplication to work properly, but each vector appears as a row vector in the matrix? Well, in order for the matrix multiplication to work properly, the dimensions of our matrices must agree. Accordingly, we must employ the transpose when multiplying our matrix of vectors by our transformation matrix. What I am referring to is that, for matrix multiplication to work, the number of columns in the first matrix must be equal to the number of rows in the second matrix - this is spelled out in one of the lecture videos you watched for Assignment 4.

A couple last notes: Use NumPy arrays to represent your matrices, so that matrix multiplication can be performed using the dot() method. Also, you can get the transpose of a Numpy array by using its T attribute.

Preliminaries

Let's import some common packages:

In [241…

```
import matplotlib.pyplot as plt
import matplotlib as mpl
from matplotlib import cm
import numpy as np
import pandas as pd
%matplotlib inline
mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)
import os

# Where to save the figures
PROJECT_ROOT_DIR = "."
FOLDER = "figures"
IMAGES_PATH = os.path.join(PROJECT_ROOT_DIR, FOLDER)
os.makedirs(IMAGES_PATH, exist_ok=True)

def save_fig(fig_id, tight_layout=True, fig_extension="png", resolution=300):
    path = os.path.join(IMAGES_PATH, fig_id + "." + fig_extension)
    print("Saving figure", fig_id)
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format=fig_extension, dpi=resolution)
```

Write Plotting Function

You must define a function called `plotTransformedData(original_df, trans_df, colors, size, limits)` that accepts two Pandas DataFrames (composed of 2 spatial coordinates each) called `original_df` and `trans_df`. The former is the original data that you will read in, and the latter is that data after you applied a linear transformation to it. You will plot both, side-by-side, to compare the before and after effects of the transformation. The function must use scatter() to plot the data.

Additionally, the `plotTransformedData` function accepts three other parameters: `colors` , `size` , and `limits` .

- The `colors` parameter is passed to the `scatter` function through its `c` parameter. The `colors` parameter must be a $(n, 3)$ NumPy array, where n is the number of data points and each of the three columns corresponds to an RGB value in the range [0,1]. *Hint: The first point is black and the last point is red. Look up the RGB colors for these two colors and think about how you could get the colors to gradually go from black to red.*
- The `size` parameter is passed to the `scatter` function through its `s` parameter. The `size` parameter controls the size of the marker used to represent the data point, and must be a Numpy array of size $(n,)$. (You can use np.linspace() to create the array).
- Finally, the `limits` parameter is a list that looks like this: `limits = [xmin, xmax, ymin, ymax]` , where the values are the limits for the x and y axes. You must specify these `limits` values so that they are equal to mine (I used -2.25 and +2.25 for both x and y as my limits).

You must place the definition of the `plotTransformedData(original_df, trans_df, colors, size, limits)` function in the following cell.

Note: An example figure is shown below that you should emulate as closely as possible.

In [242…

```
def plotTransformedData(original_df, trans_df, colors, size, limits):
    original = original_df

    x_trans = trans_df

    sizes = np.linspace(12, 100, 25)

    colors = np.array([[0,0,0], #black
                       [.01,0,0],
                       [.015,0,0],
                       [.02,0,0],
                       [.025,0,0],
                       [.30,0,0], #red-black
                       [.30,0,0],
                       [.30,0,0],
                       [.30,0,0],
                       [.30,0,0],
                       [.60,0,0], #redder
                       [.60,0,0],
                       [.60,0,0],
                       [.60,0,0],
                       [.60,0,0],
```

```
                        [.60,0,0], #more red
                        [.80,0,0],
                        [.80,0,0],
                        [.80,0,0],
                        [.80,0,0],
                        [1,0,0],   #red
                        [1,0,0],
                        [1,0,0],
                        [1,0,0],
                        [1,0,0]])

    limits = [-2.25, 2.25, -2.25, 2.25]

    plt.figure(figsize=(15,5))
    plt.subplot(1, 2, 1)
    plt.scatter(original.iloc[:,0], original.iloc[:,1], s = size, c = colors)
    plt.xlabel("X")
    plt.ylabel("y")
    plt.title('Original Data')
    plt.axis([-2, 2, -2, 2])
    plt.xlim(limits[0], limits[1])
    plt.ylim(limits[2], limits[3])

    plt.subplot(1, 2, 2)
    plt.scatter(x_trans.iloc[:,0], x_trans.iloc[:,1], s = size, c = colors)
    plt.xlabel("X")
    plt.ylabel("y")
    plt.title('Transformed Data')
    plt.axis([-2, 2, -2, 2])
    plt.xlim(limits[0], limits[1])
    plt.ylim(limits[2], limits[3])


    save_fig("2D Data Plot Initial Data")
    plt.show()
```

Import Data

Import data from the file called `2D_data.csv` . Name the returned DataFrame `data` .

In [244…
```
data = pd.read_csv("2D_data.csv")
```

Plot Initial Data

Begin by specifying the `colors`, `sizes`, and `limits` parameters for the `plotTransformedData` function. You will not call the `plotTransformedData(original_df, trans_df, colors, size, limits)` function in this section, but rather you'll invoke it below in the section called *Perform Linear Transformations and Plot Results*. However, you will plot the `data` DataFrame here, merely to inspect it, using the marker colors, marker sizes, and axis limits specified by the `colors`, `sizes`, and `limits` parameters.

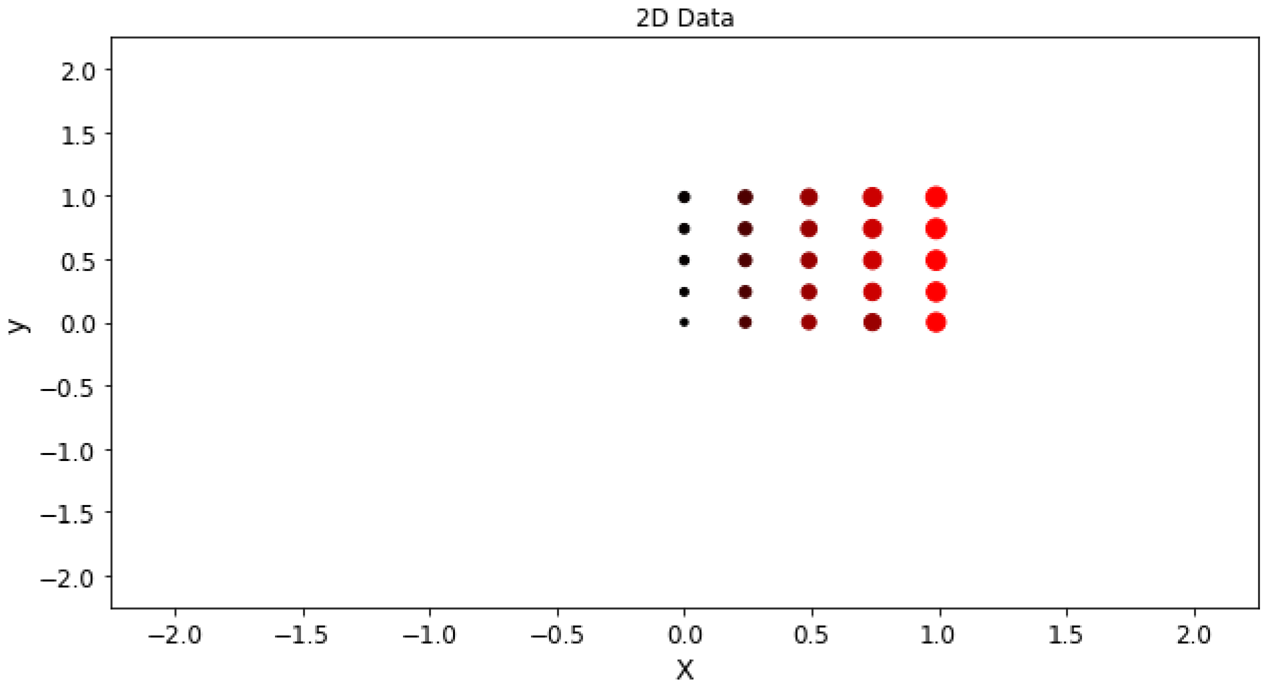
So, plot `data` here. You should emulate the marker colors and sizes shown below as best as you can. The marker sizes range from small to large, and the marker colors vary from black to red.

In [245…

```
sizes = np.linspace(12, 100, 25)

colors = np.array([[0,0,0], #black
                  [.01,0,0],
                  [.015,0,0],
                  [.02,0,0],
                  [.025,0,0],
                  [.30,0,0], #red-black
                  [.30,0,0],
                  [.30,0,0],
                  [.30,0,0],
                  [.30,0,0],
                  [.60,0,0], #redder
                  [.60,0,0],
                  [.60,0,0],
                  [.60,0,0],
                  [.60,0,0],
                  [.60,0,0], #more red
                  [.80,0,0],
                  [.80,0,0],
                  [.80,0,0],
                  [.80,0,0],
                  [1,0,0], #red
                  [1,0,0],
                  [1,0,0],
                  [1,0,0],
                  [1,0,0]])

limits = [-2.25, 2.25, -2.25, 2.25]
```

```python
plt.figure(figsize=(9,5))
plt.scatter(data['x'],data['y'], s = sizes, c = colors)
plt.xlabel("X")
plt.ylabel("y")
plt.title('2D Data')
plt.axis([-2, 2, -2, 2])
plt.xlim(limits[0], limits[1])
plt.ylim(limits[2], limits[3])
save_fig("2D Data Plot Initial Data")
plt.show()
```

```
Saving figure 2D Data Plot Initial Data
```



Define Data Matrix and Transformation Matrices

Create a NumPy array called `X` from the `data` DataFrame above. Further, you must define the following transformation matrices. You must declare these matrices to be NumPy arrays, and name them A1, A2, ..., AN, as shown.

In [246…

```python
# -------------------- Define X NumPy Array -------------------- #
```

```python
# Data Matrix

X = data.to_numpy()

# -------------------- Define 13 NumPy Arrays -------------------- #

k = 0.5
greek = np.pi / 4
c = 2.0
r = 0.5
m = 0.5

A1 = np.array(([-1, 0], [0, -1]))
A2 = np.array(([k, 0], [0, 1]))
A3 = np.array(([np.cos((greek)), -np.sin((greek))], [np.sin((greek)), np.cos((greek))]))
A4 = np.array(([-1, 0], [0, 1]))
A5 = np.array(([1, 0], [0, c]))
A6 = np.array(([1, 0], [r, 1]))
A7 = np.array(([1, m], [0, 1]))
A8 = np.array(([0, -1], [-1, 0]))
A9 = np.array(([1, 0], [0, 0]))
A10 = np.array(([0, 0], [0, 1]))
A11 = np.array(([1, 0], [0, -1]))
A12 = np.array(([0, 1], [1, 0]))
A13 = np.array(([1, 0], [0, 1]))
```

Perform Linear Transformations and Plot Results

In this section, you will perform the following:

$$\mathbf{X}_{trans} = \mathbf{A}\mathbf{X}^T$$

where $\mathbf{X}$ is the data you read in from the file, $\mathbf{A}$ is any one of the thirteen transformation matrices, and $\mathbf{X}_{trans}$ is the transformed data. You should express the matrix multiplication in one line of code!

Important: When performing the matrix multiplication, please note that the order in scalar or dot product matters. A dot B is not the same as B dot A.

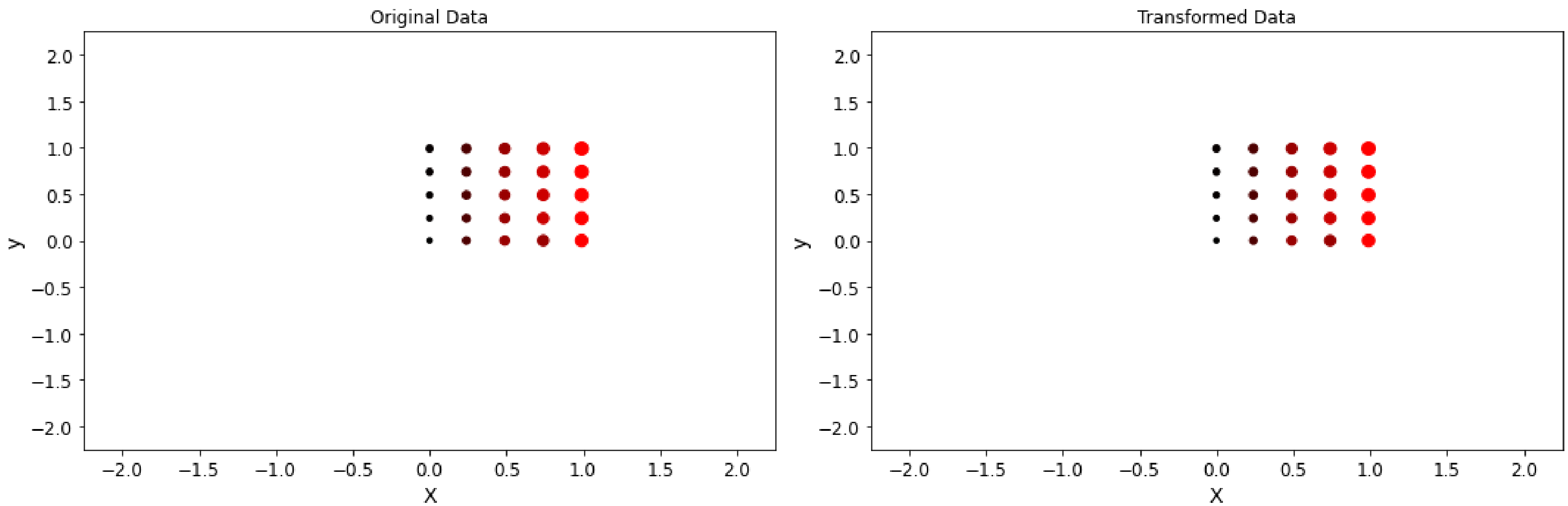
Identity Matrix

Any matrix $\mathbf{A}$ multiplied by the ***Identity Matrix*** $\mathbf{I}$ is equal to itself! That is, $\mathbf{AI} = \mathbf{A}$. The 2D identity matrix is given below.

In [247…

```
# Perform Matrix Multiplication

#transformed = A * XT

identity = np.dot(A13, X.T)

identity = pd.DataFrame(identity.T)

# # Plot Transformation

plotTransformedData(data, identity, colors, size, limits)
```

```
Saving figure 2D Data Plot Initial Data
```



For the rest of the following linear transformations, you can Google the matrix transformation definitions to learn more.

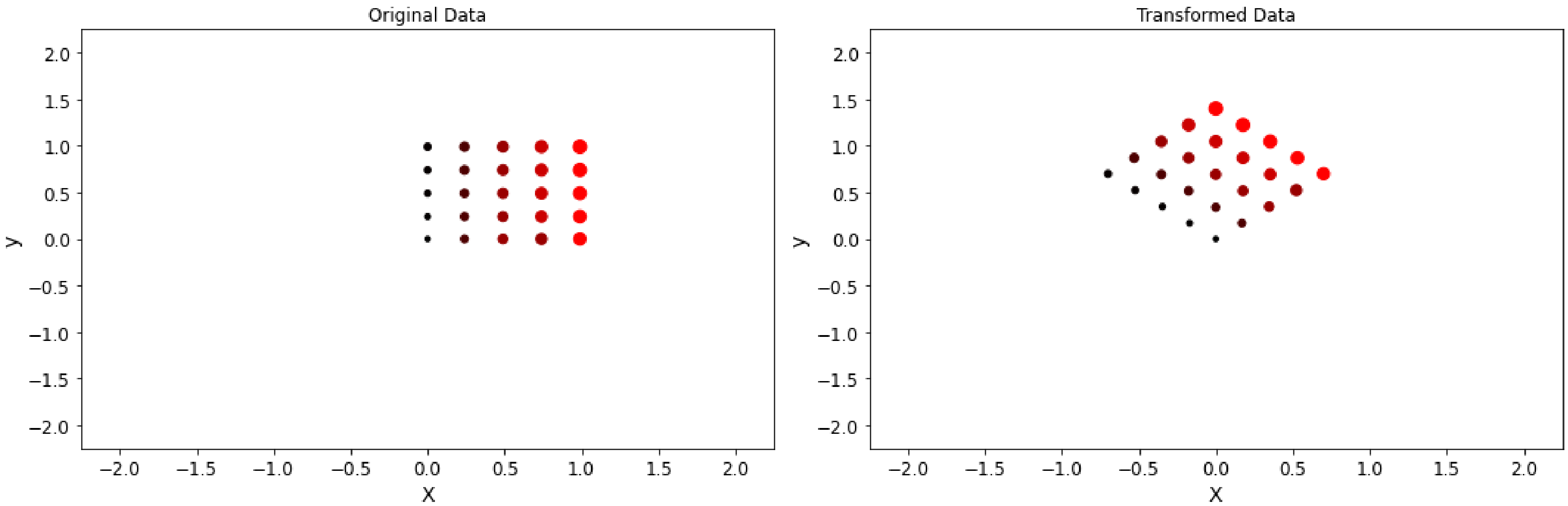
Rotation Matrix

In [248…

```
# Perform Matrix Multiplication

rotation = np.dot(A3, X.T)

rotation = pd.DataFrame(rotation.T)

# Plot Transformation
```

```
plotTransformedData(data, rotation, colors, size, limits)
```

```
Saving figure 2D Data Plot Initial Data
```

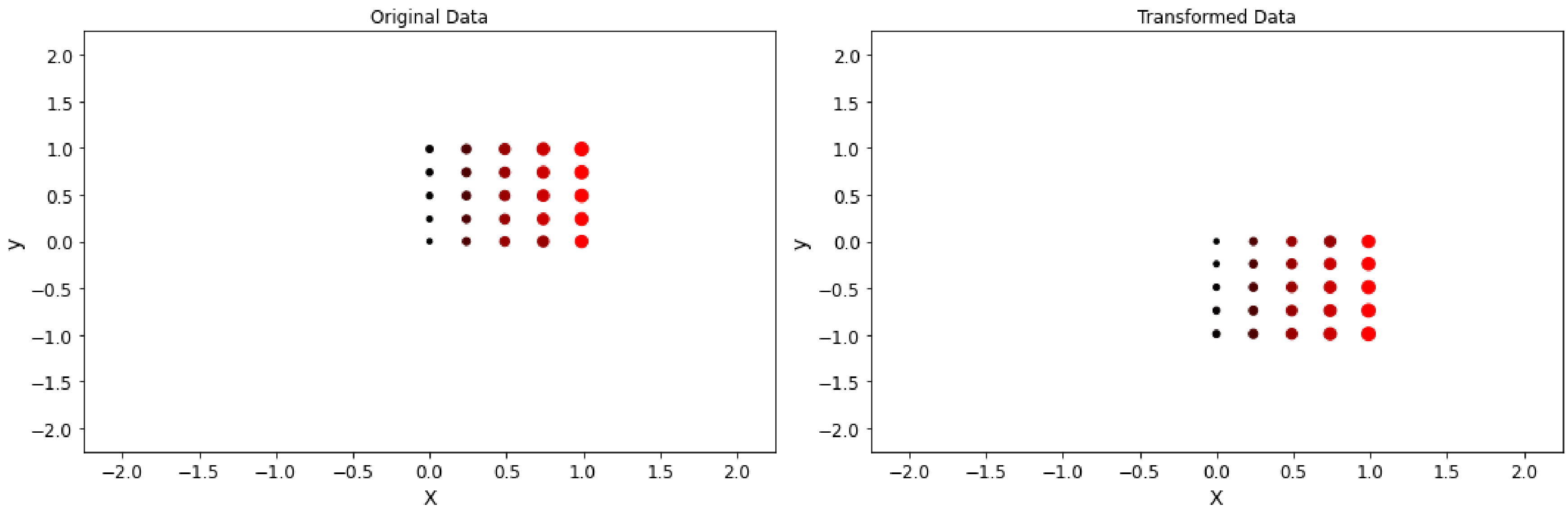


Reflection Matrices

In [249…

```
# Perform Matrix Multiplication

# Reflection through the x Axis Matrix

x_reflect = np.dot(A11, X.T)
x_reflect = pd.DataFrame(x_reflect.T)

# Plot Transformation

plotTransformedData(data, x_reflect, colors, size, limits)
```

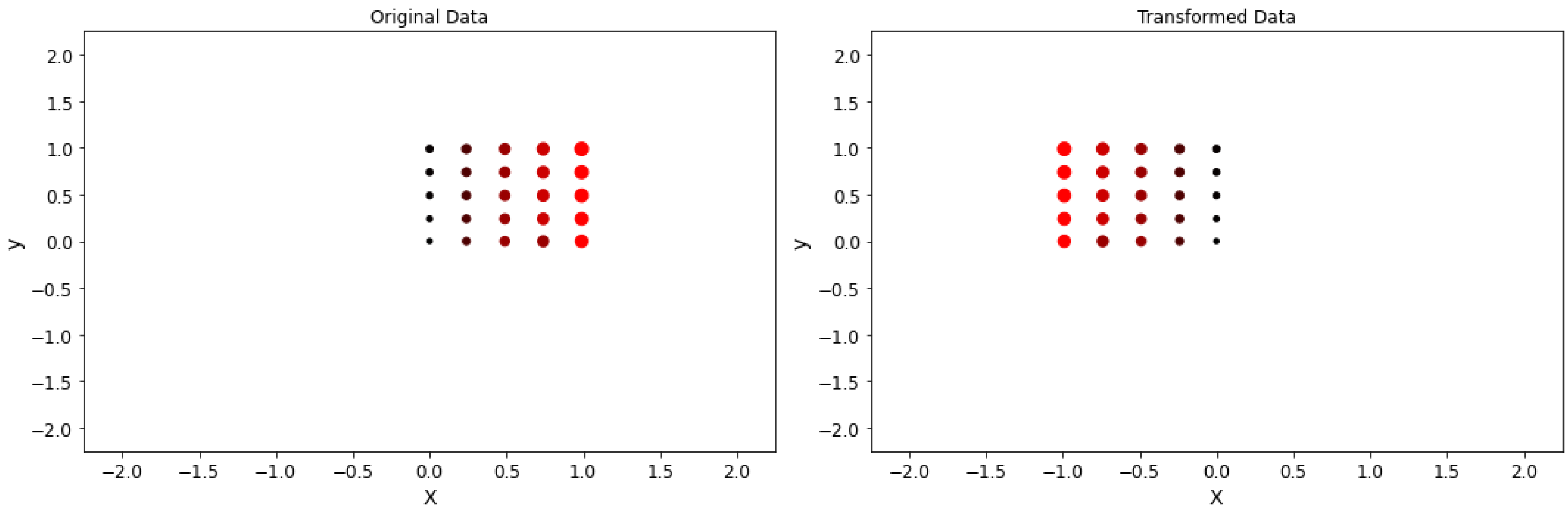
```
Saving figure 2D Data Plot Initial Data
```



In [250…

```
# Perform Matrix Multiplication

# Reflection through the y Axis Matrix

y_reflect = np.dot(A4, X.T)
y_reflect = pd.DataFrame(y_reflect.T)

# Plot Transformation

plotTransformedData(data, y_reflect, colors, size, limits)
```

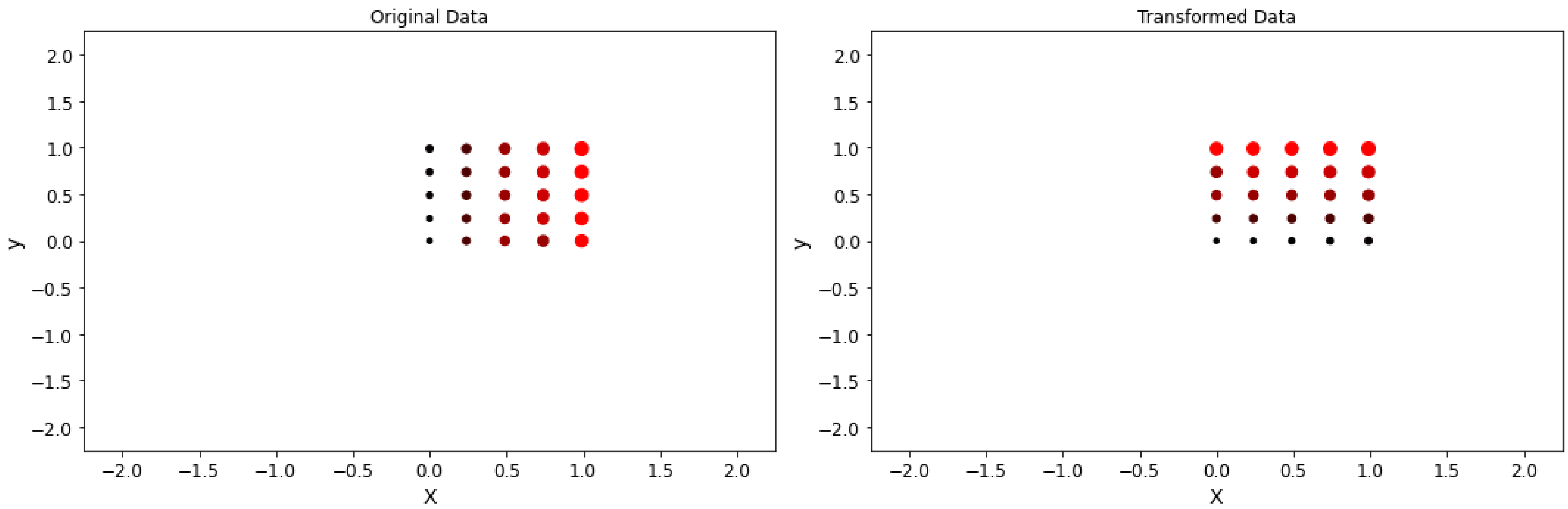
```
Saving figure 2D Data Plot Initial Data
```



In [251…

```
# Perform Matrix Multiplication

#Reflection through the Line y = x Matrix

yx_reflect = np.dot(A12, X.T)
yx_reflect = pd.DataFrame(yx_reflect.T)

# Plot Transformation

plotTransformedData(data, yx_reflect, colors, size, limits)
```

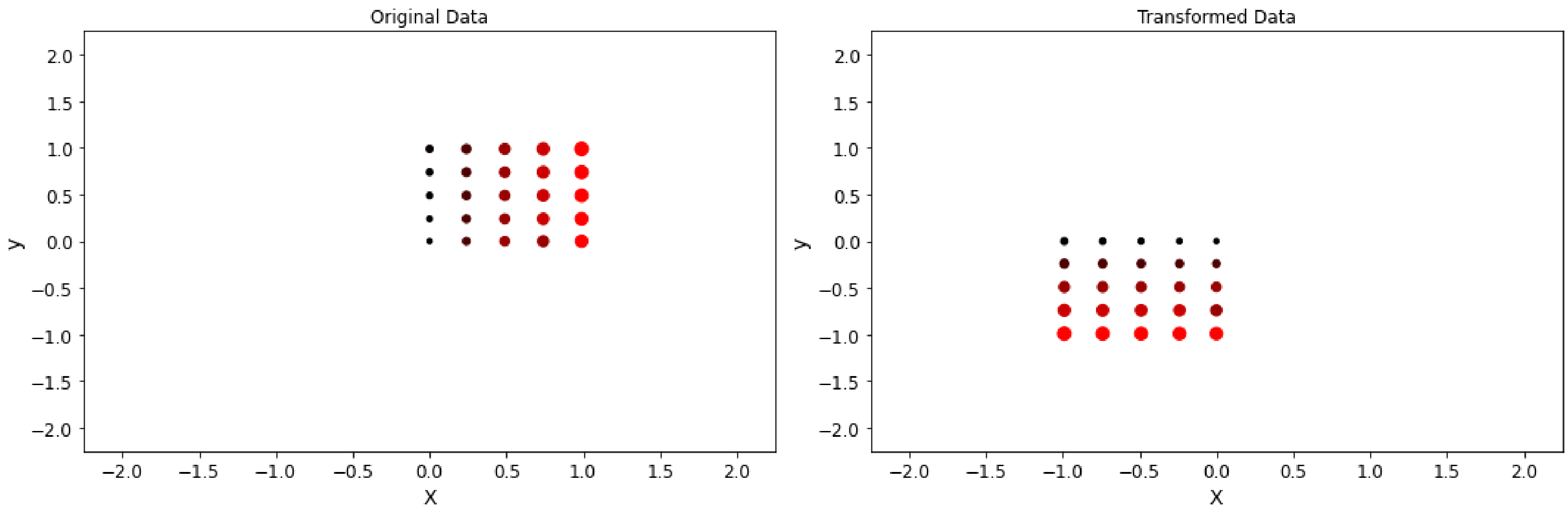
```
Saving figure 2D Data Plot Initial Data
```



In [252…

```
# Perform Matrix Multiplication

# Reflection through the Line y = -x Matrix

neg_yx_reflect = np.dot(A8, X.T)
neg_yx_reflect = pd.DataFrame(neg_yx_reflect.T)

# Plot Transformation

plotTransformedData(data, neg_yx_reflect, colors, size, limits)
```

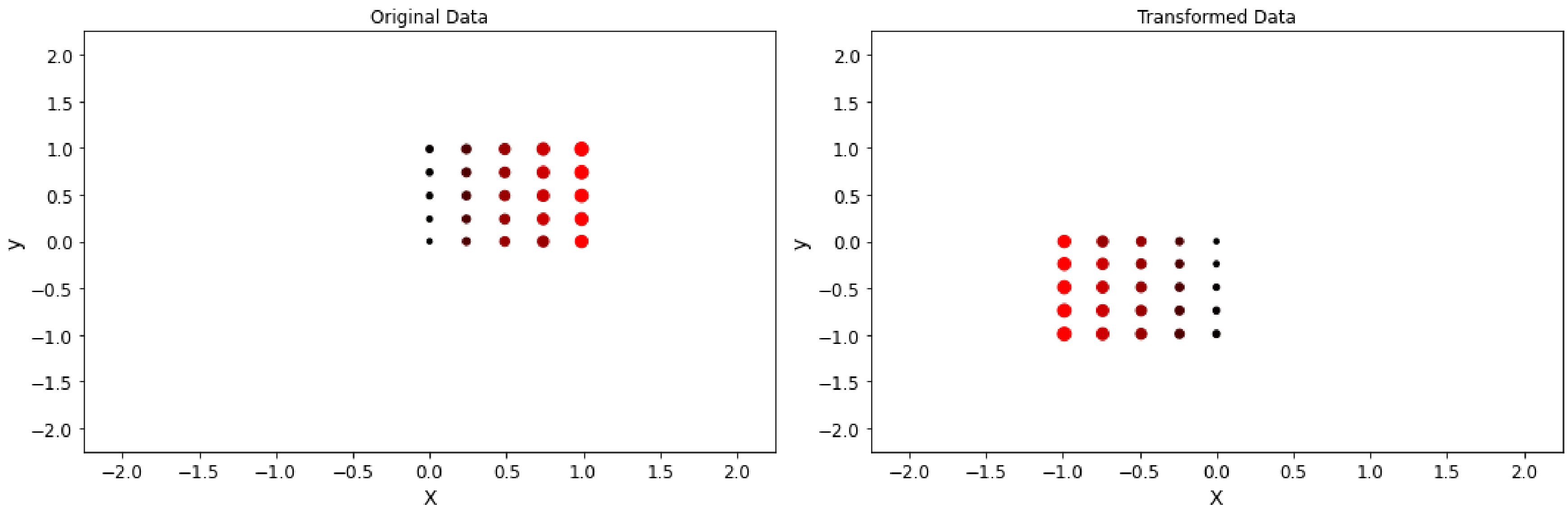
```
Saving figure 2D Data Plot Initial Data
```



In [253…

```
# Perform Matrix Multiplication

#Reflection through the Origin Matrix

origin_reflect = np.dot(A1, X.T)
origin_reflect = pd.DataFrame(origin_reflect.T)

# Plot Transformation

plotTransformedData(data, origin_reflect, colors, size, limits)
```

```
Saving figure 2D Data Plot Initial Data
```

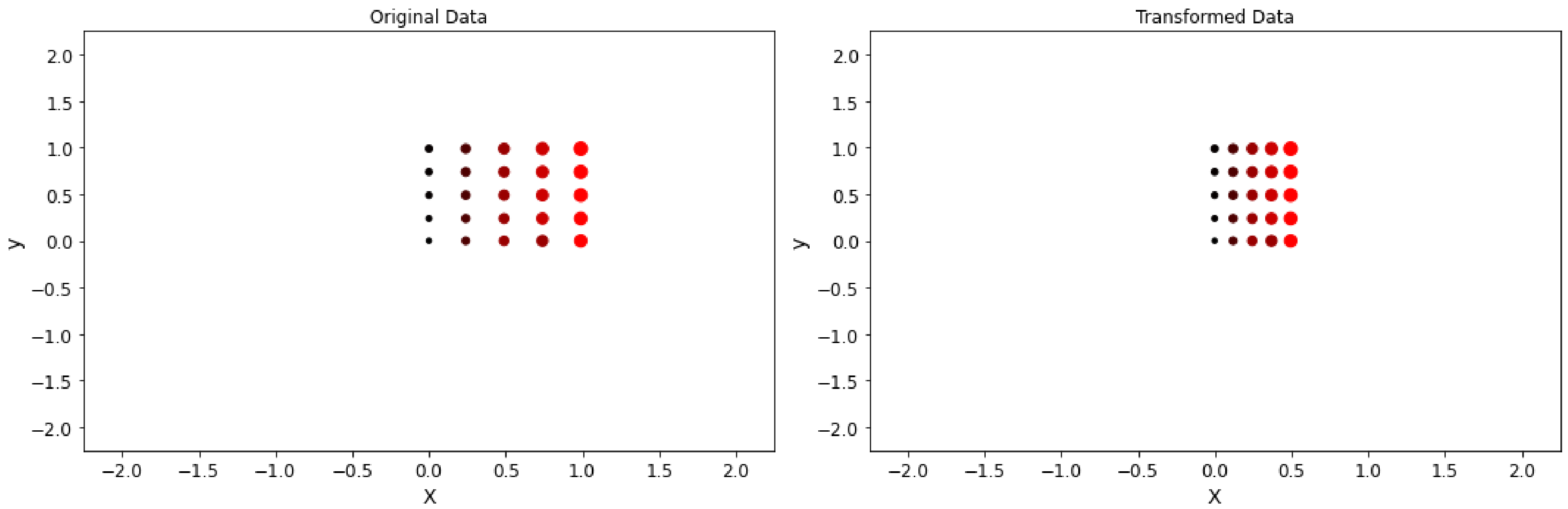


Contraction and Expansion Matrices

In [254…

```
# Perform Matrix Multiplication

#Horizontal Contraction/Expansion Matrix

hcontract = np.dot(A2, X.T)

hcontract = pd.DataFrame(hcontract.T)

# Plot Transformation

plotTransformedData(data, hcontract, colors, size, limits)
```

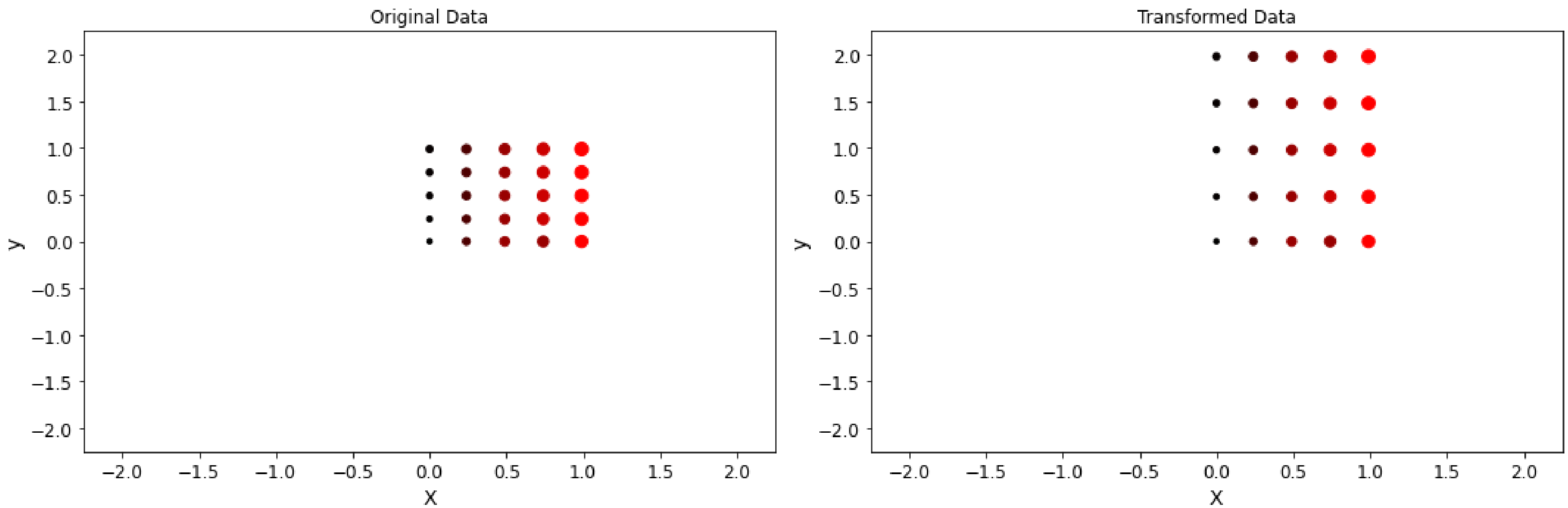
```
Saving figure 2D Data Plot Initial Data
```



In [255…

```
# Perform Matrix Multiplication

#A5: Vertical Contraction/Expansion Matrix

vcontract = np.dot(A5, X.T)

vcontract = pd.DataFrame(vcontract.T)

# Plot Transformation

plotTransformedData(data, vcontract, colors, size, limits)
```

```
Saving figure 2D Data Plot Initial Data
```

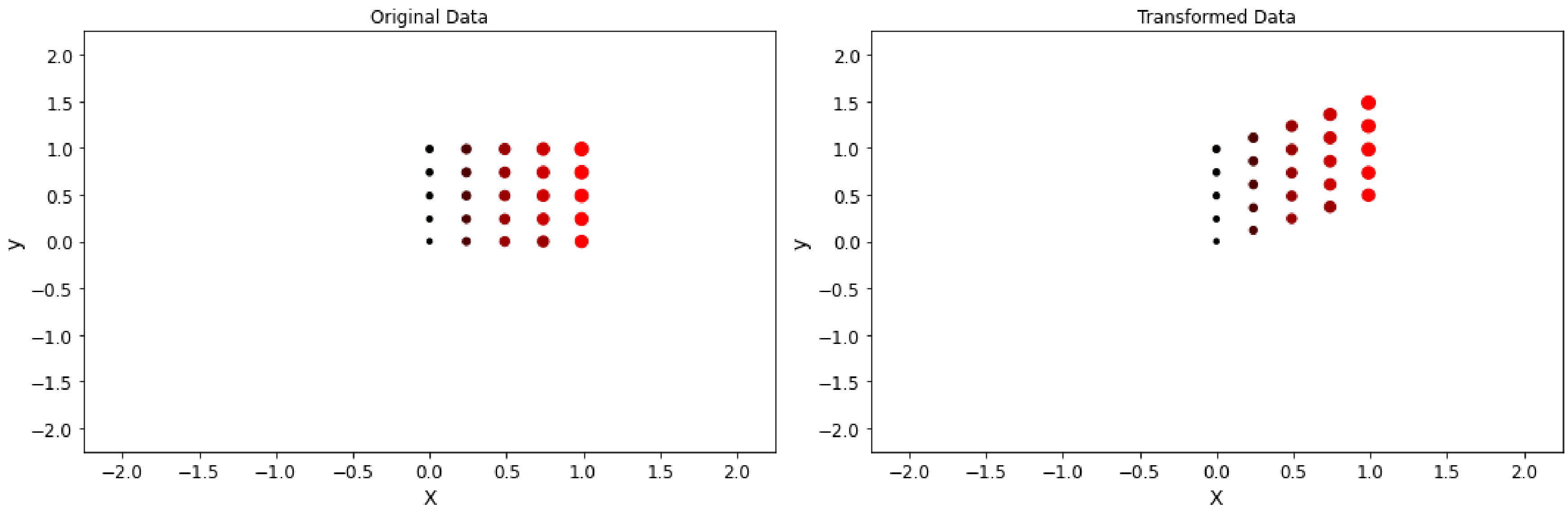


Shearing Matrices

In [256…

```
# Perform Matrix Multiplication

#A6: Vertical Shear Matrix

vshear = np.dot(A6, X.T)

vshear = pd.DataFrame(vshear.T)

# Plot Transformation

plotTransformedData(data, vshear, colors, size, limits)
```

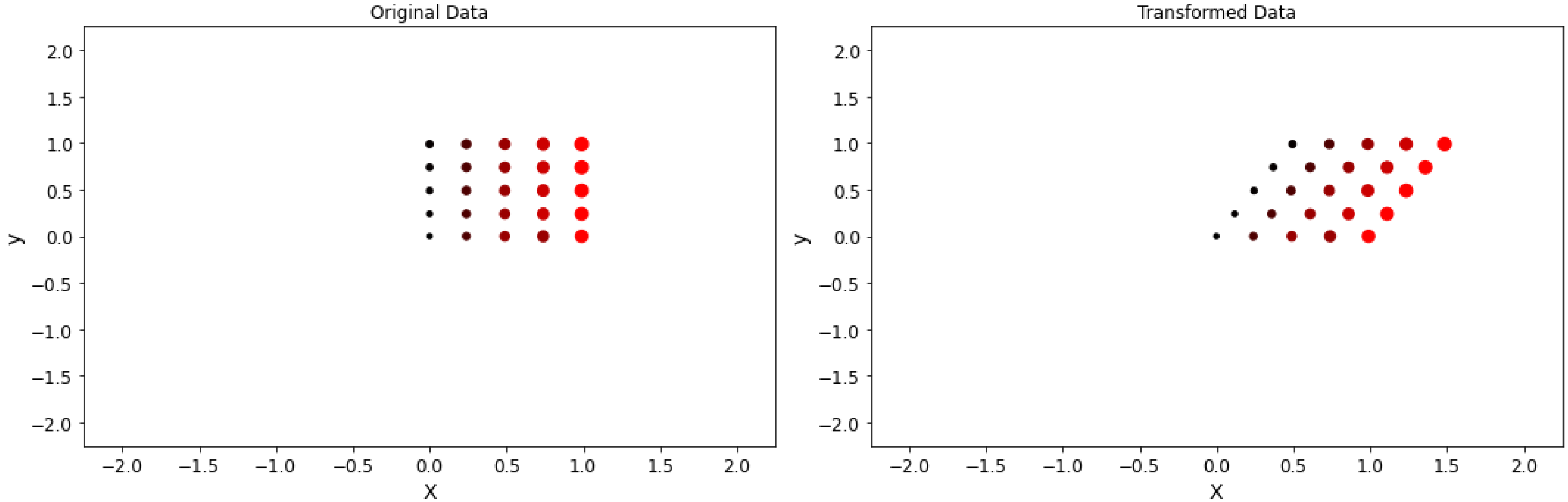
```
Saving figure 2D Data Plot Initial Data
```



In [257…

```
# Perform Matrix Multiplication

#A7: Hortizontal Shear Matrix

hshear = np.dot(A7, X.T)

hshear = pd.DataFrame(hshear.T)

# Plot Transformation

plotTransformedData(data, hshear, colors, size, limits)
```

```
Saving figure 2D Data Plot Initial Data
```

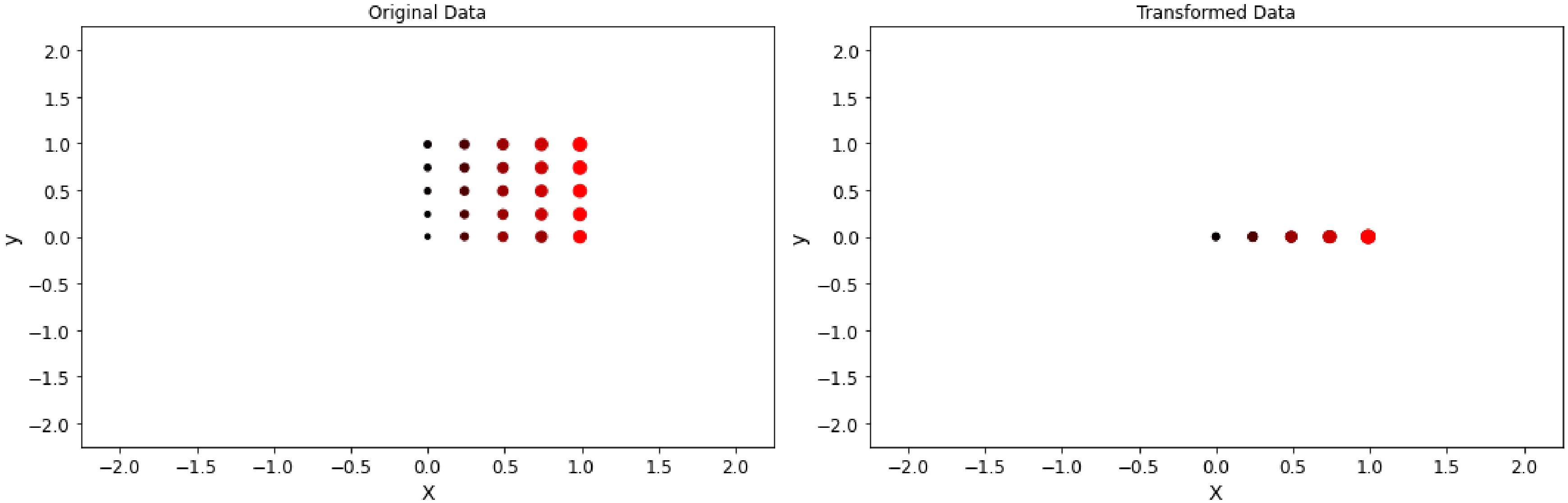


Projection Matrices

In [258…

```
# Perform Matrix Multiplication

#A9: Projection onto X Axis matrix

project_x = np.dot(A9, X.T)

project_x = pd.DataFrame(project_x.T)

# Plot Transformation

plotTransformedData(data, project_x, colors, size, limits)
```

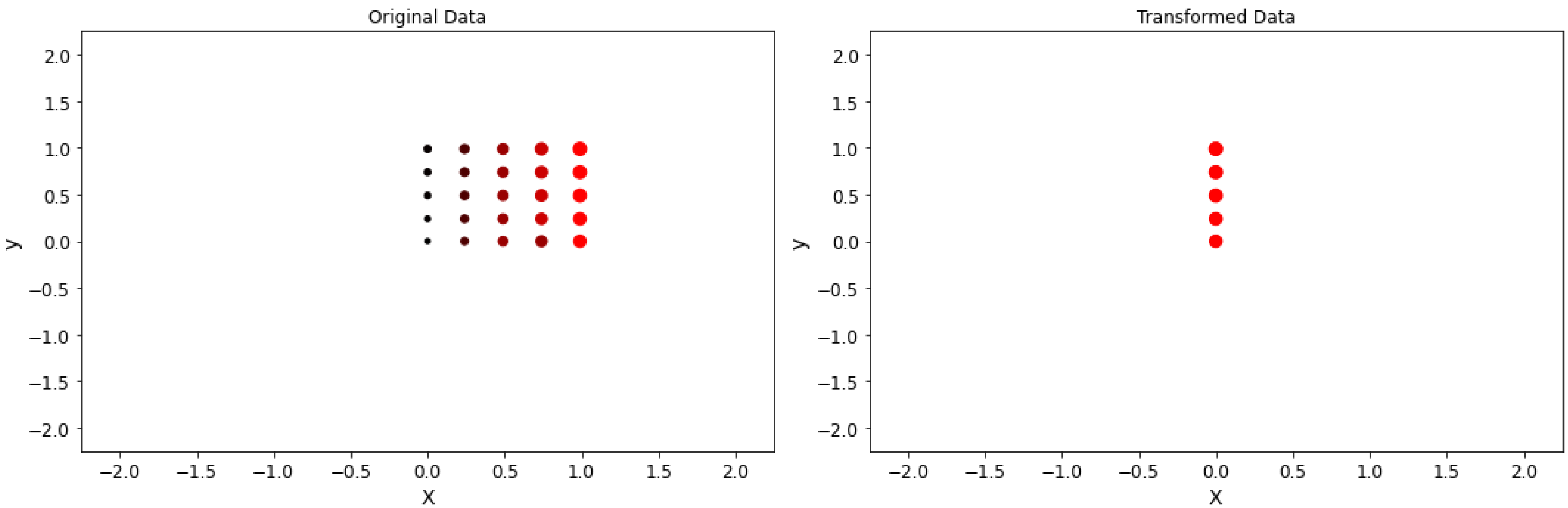
```
Saving figure 2D Data Plot Initial Data
```



In [259…

```
# Perform Matrix Multiplication

#A10: Projection onto y Axis matrix

project_y = np.dot(A10, X.T)

project_y = pd.DataFrame(project_y.T)

# Plot Transformation

plotTransformedData(data, project_y, colors, size, limits)
```

```
Saving figure 2D Data Plot Initial Data
```



Connect the Dots

In this section, you must correctly label linear transformations their names. The pool of transformations is given below:

- Identity Matrix
- Rotation Matrix
- Reflection through the x Axis Matrix
- Reflection through the y Axis Matrix
- Reflection through the Line y = x Matrix
- Reflection through the Line y = -x Matrix
- Reflection through the Origin Matrix
- Horizontal Contraction/Expansion Matrix
- Vertical Contraction/Expansion Matrix
- Hortizontal Shear Matrix
- Vertical Shear Matrix
- Projection onto x Axis Matrix
- Projection onto y Axis Matrix

You must correctly label the following linear transformations with the above names.

1. A1: Reflection through the Origin Matrix
2. A2: Horizontal Contraction/Expansion Matrix
3. A3: Rotation Matrix
4. A4: Reflection through the y Axis Matrix
5. A5: Vertical Contraction/Expansion Matrix
6. A6: Vertical Shear Matrix
7. A7: Hortizontal Shear Matrix
8. A8: Reflection through the line y = -x matrix
9. A9: Projection onto x Axis Matrix
10. A10: Projection onto y Axis Matrix
11. A11: Reflection through the x Axis Matrix
12. A12: Reflection through the Line y = x Matrix
13. A13: Identity Matrix