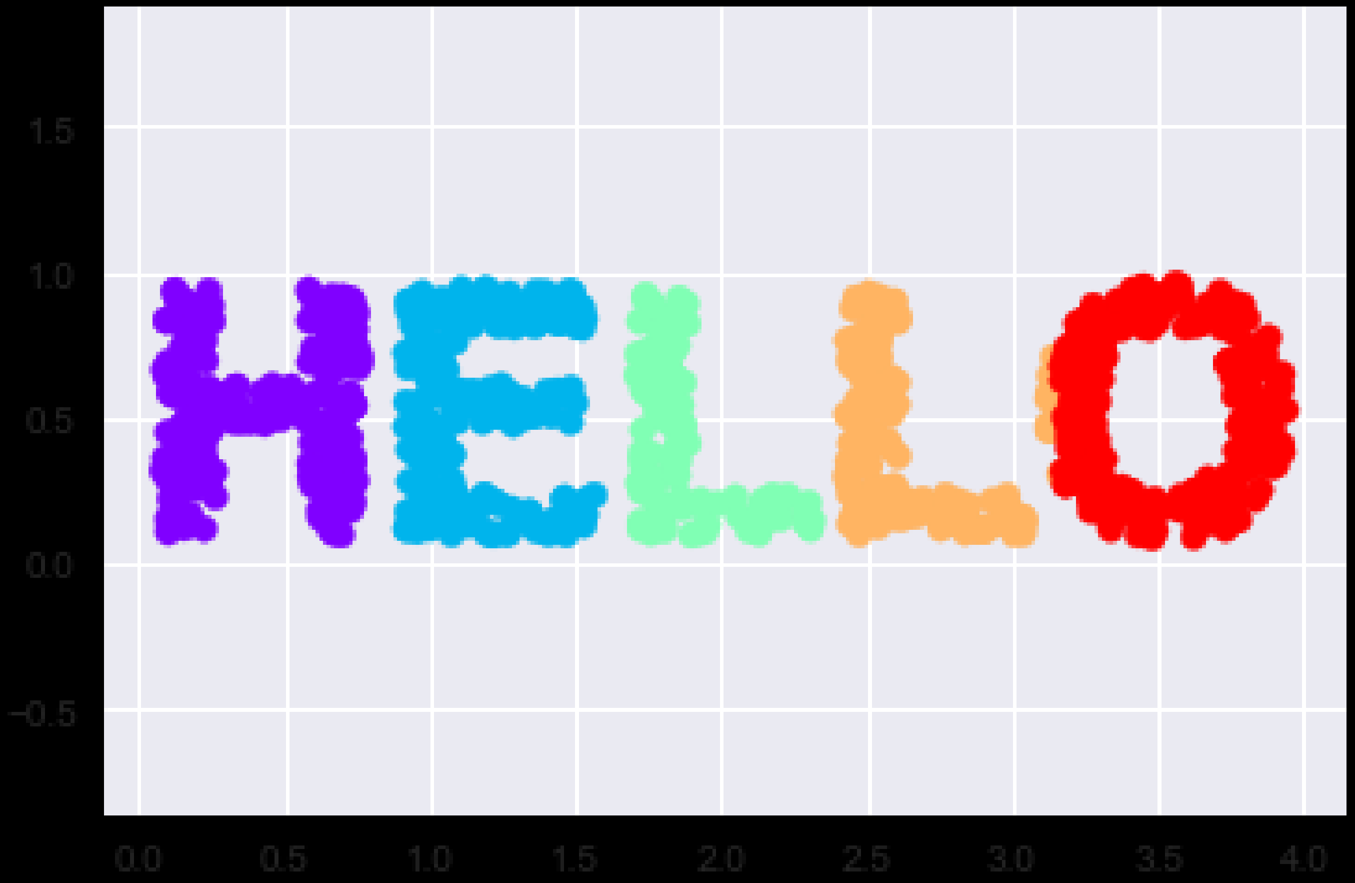
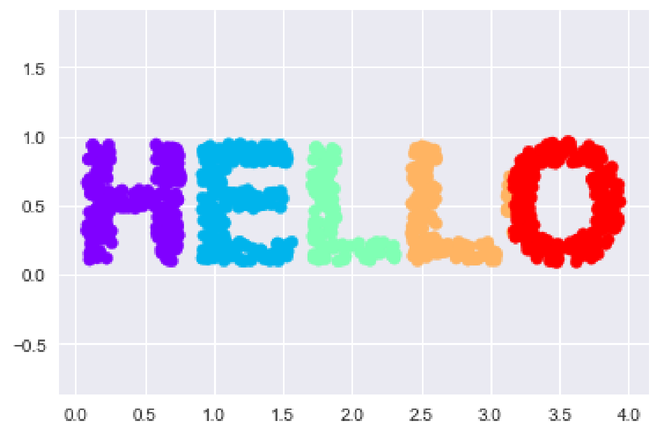
Manifold Learning

Sources: Elements of Stat. Learning & J. VanderPlas's DSHB

Manifold Learning: "HELLO"



Manifold Learning: "HELLO"

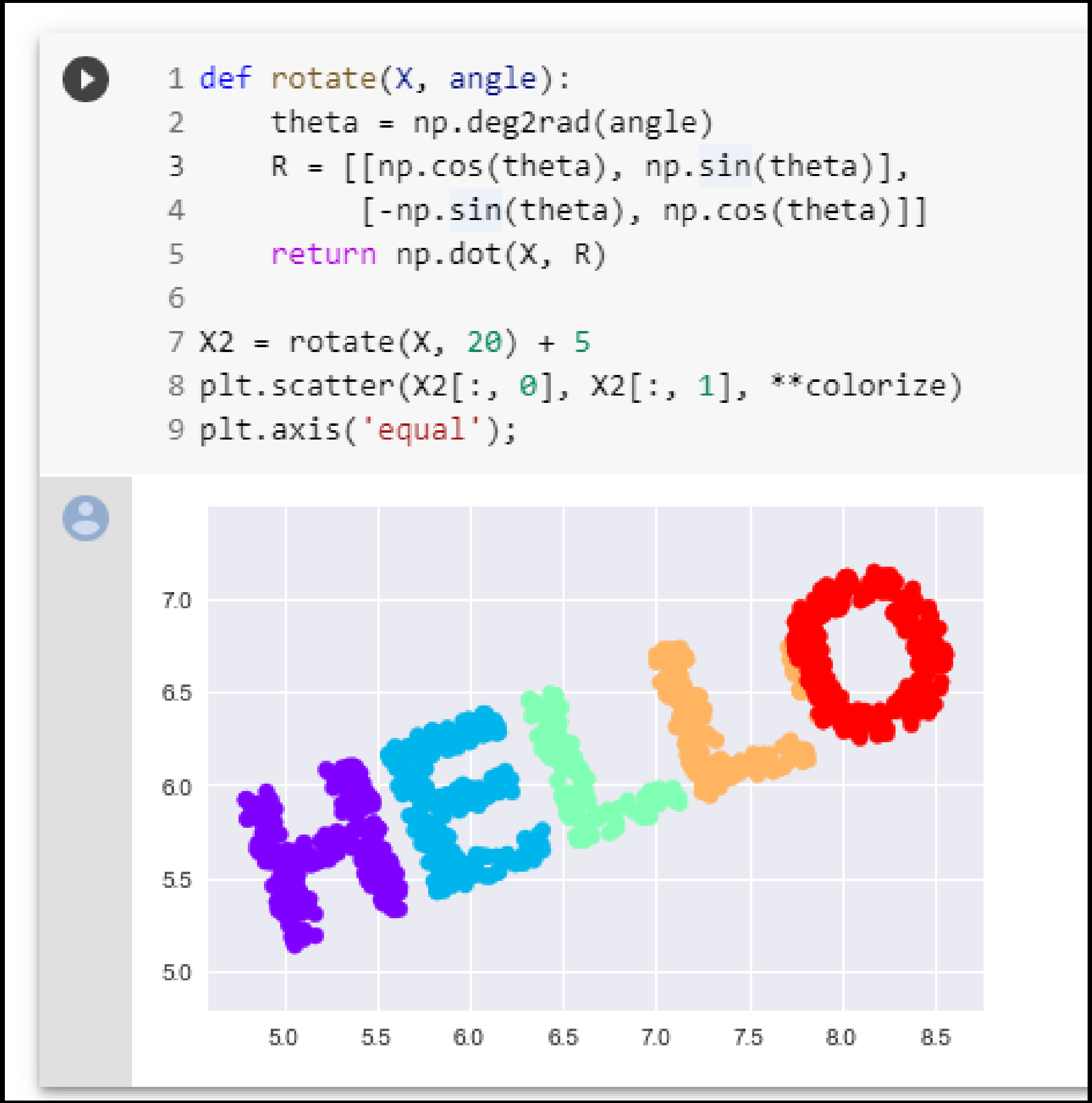


```
X
```

```
array([[0.08373647, 0.31577315],
       [0.08729359, 0.66378969],
       [0.08849421, 0.32361022],
       ...,
       [3.92447163, 0.64950893],
       [3.92503031, 0.3816165 ],
       [3.93760872, 0.52569018]])
```

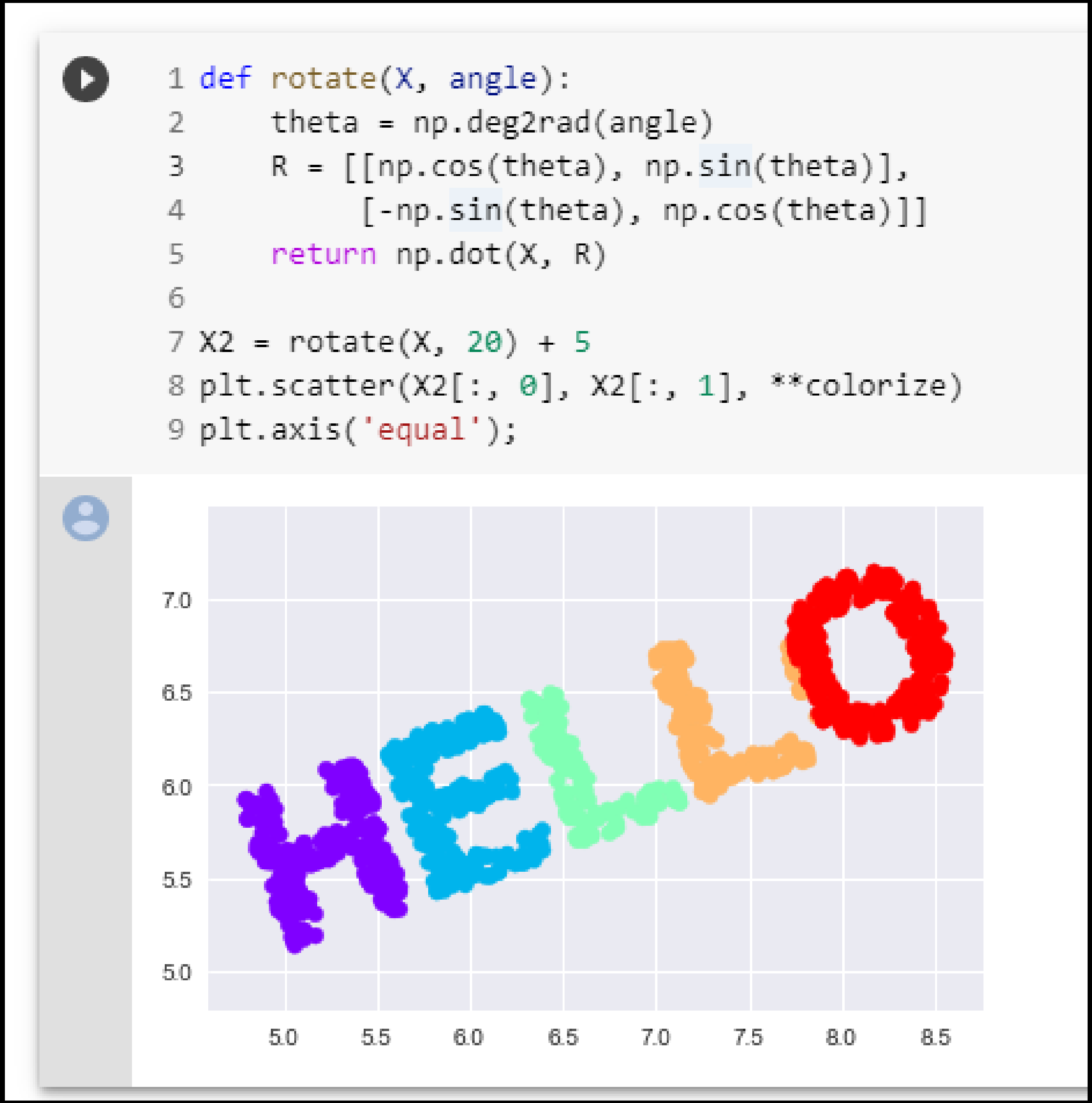
The output is two dimensional, and consists of points drawn in the shape of the word, "HELLO". This data form will help us to see visually what these algorithms are doing.

Manifold Learning: "HELLO"



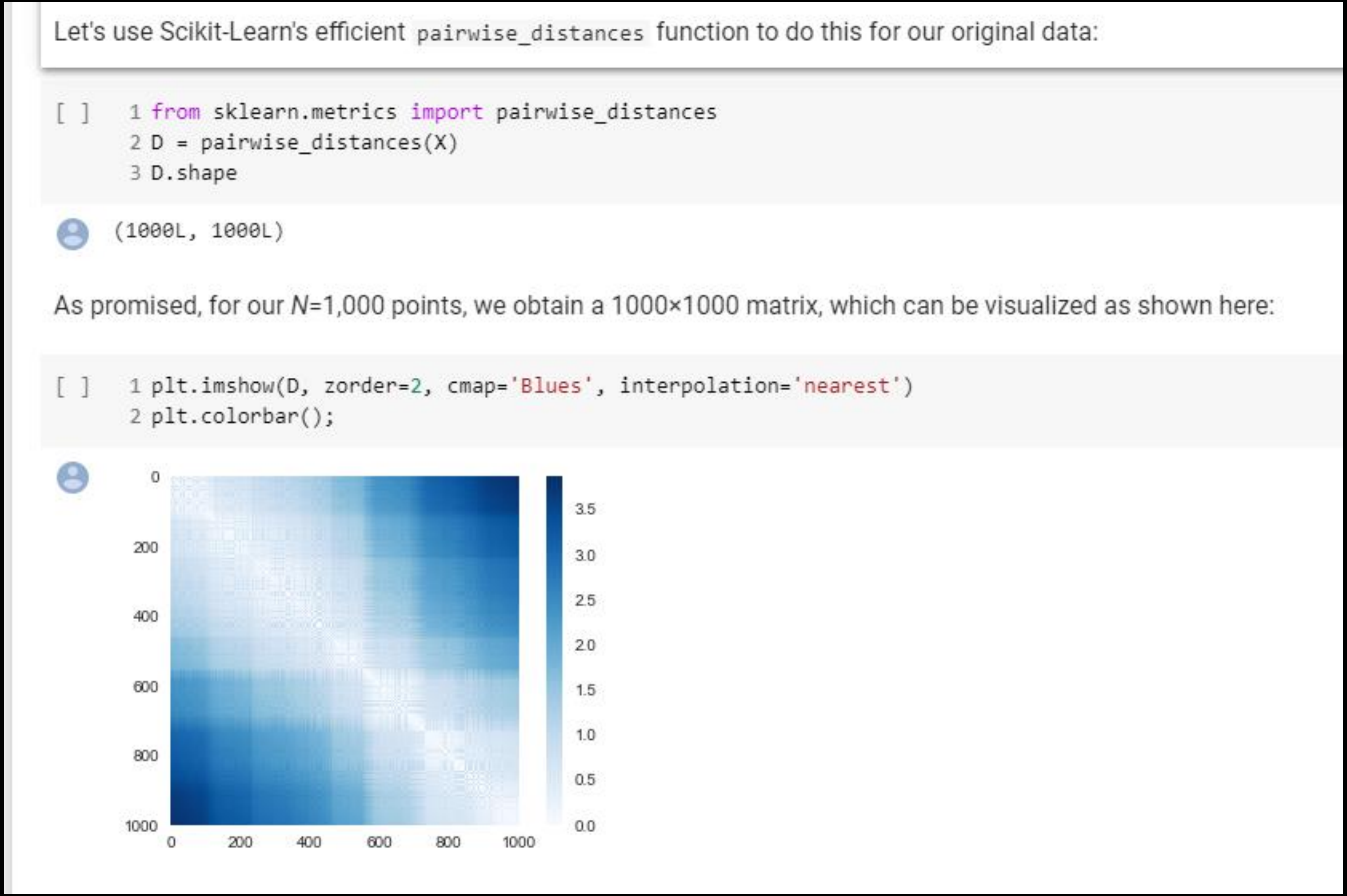
```python
def rotate(X, angle):
    theta = np.deg2rad(angle)
    R = [[np.cos(theta), np.sin(theta)],
         [-np.sin(theta), np.cos(theta)]]
    return np.dot(X, R)

X2 = rotate(X, 20) + 5
plt.scatter(X2[:, 0], X2[:, 1], **colorize)
plt.axis('equal');
```

Manifold Learning: "HELLO"



```
def rotate(X, angle):
    theta = np.deg2rad(angle)
    R = [[np.cos(theta), np.sin(theta)],
         [-np.sin(theta), np.cos(theta)]]
    return np.dot(X, R)

X2 = rotate(X, 20) + 5
plt.scatter(X2[:, 0], X2[:, 1], **colorize)
plt.axis('equal');
```

Multidimensional Scaling (MDS)



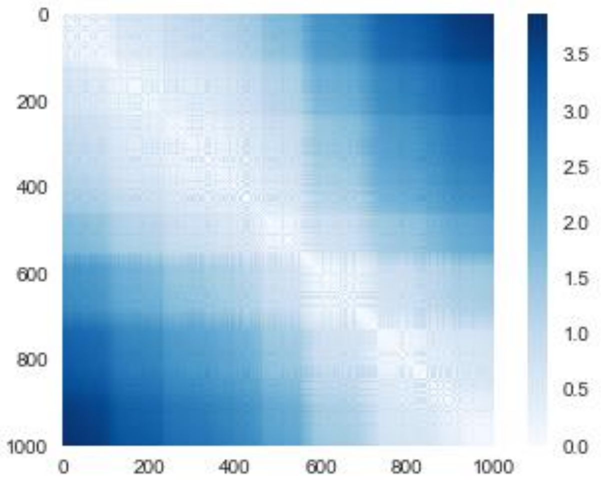
Let's use Scikit-Learn's efficient `pairwise_distances` function to do this for our original data:

```
[ ]  from sklearn.metrics import pairwise_distances
     D = pairwise_distances(X)
     D.shape
```

```
(1000L, 1000L)
```

As promised, for our *N*=1,000 points, we obtain a 1000×1000 matrix, which can be visualized as shown here:

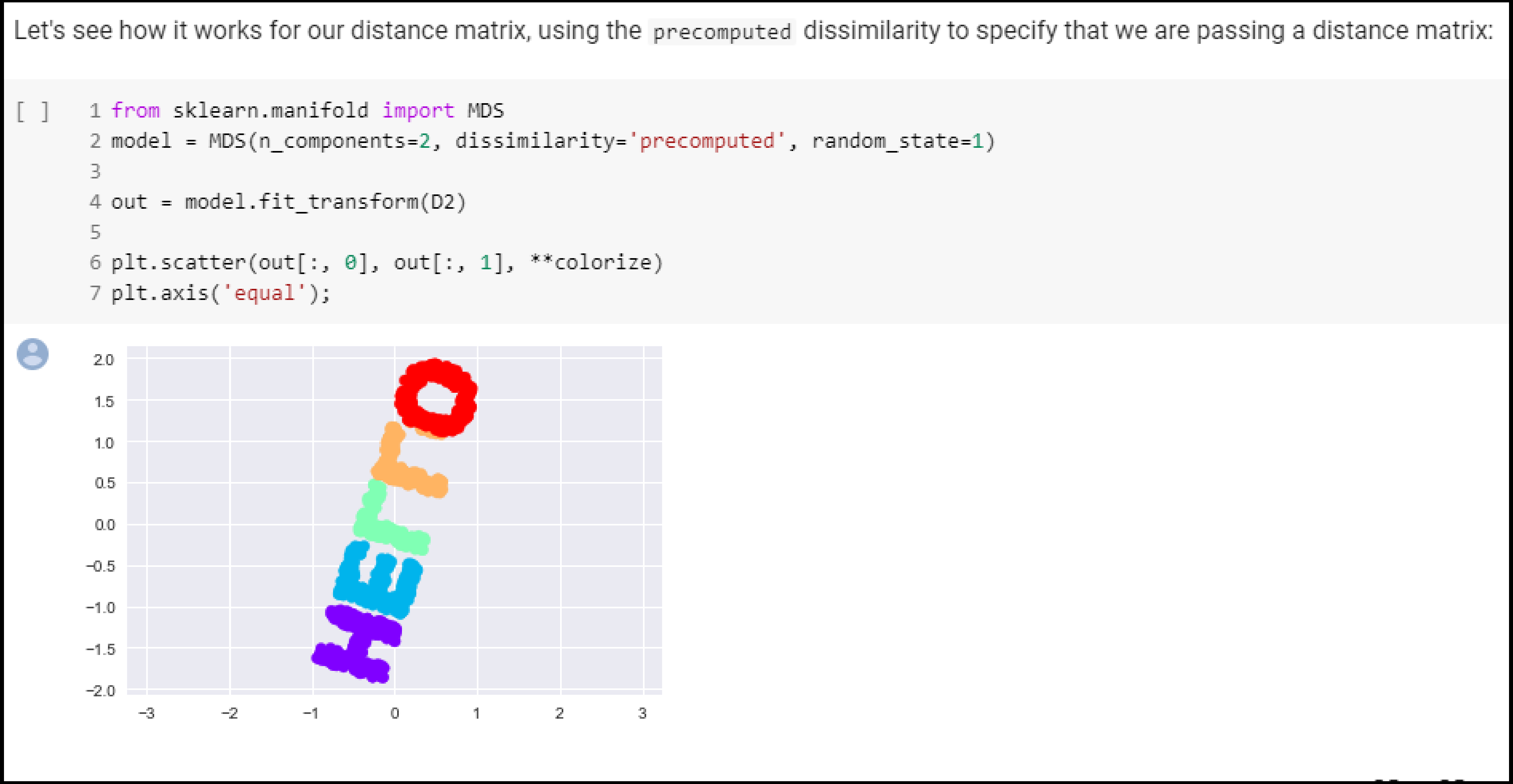
```
[ ]  plt.imshow(D, zorder=2, cmap='Blues', interpolation='nearest')
     plt.colorbar();
```



Multidimensional Scaling (MDS)

- **This is exactly what the multidimensional scaling algorithm aims to do:**
  - **given a distance matrix between points, it recovers a** D**-dimensional coordinate representation of the data.**
- Like PCA, to create useful visualizations we use it to create two dimensional visualizations
  - (i.e.- fit manifold “paper” to data and flatten it out into a visualization)

Multidimensional Scaling (MDS)

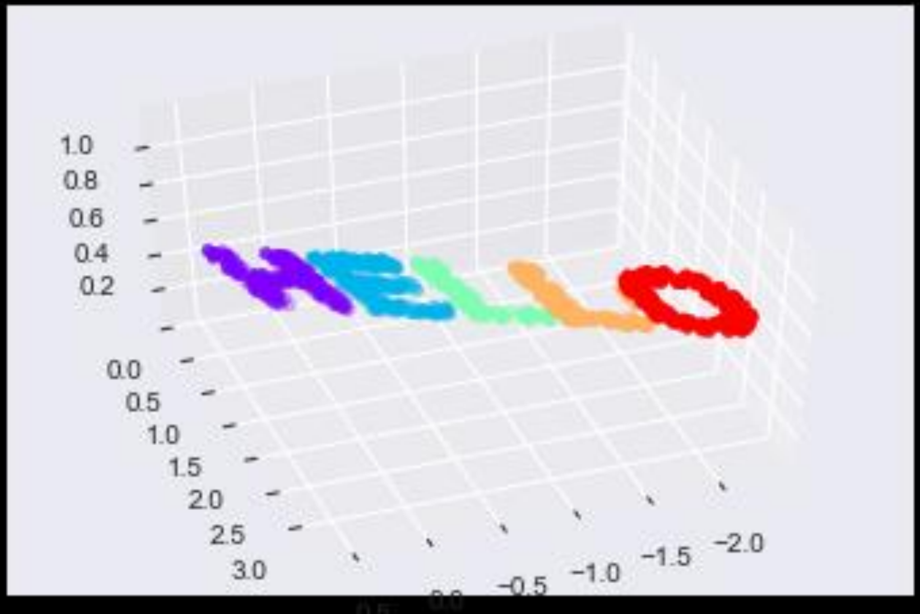


Let's see how it works for our distance matrix, using the `precomputed` dissimilarity to specify that we are passing a distance matrix:

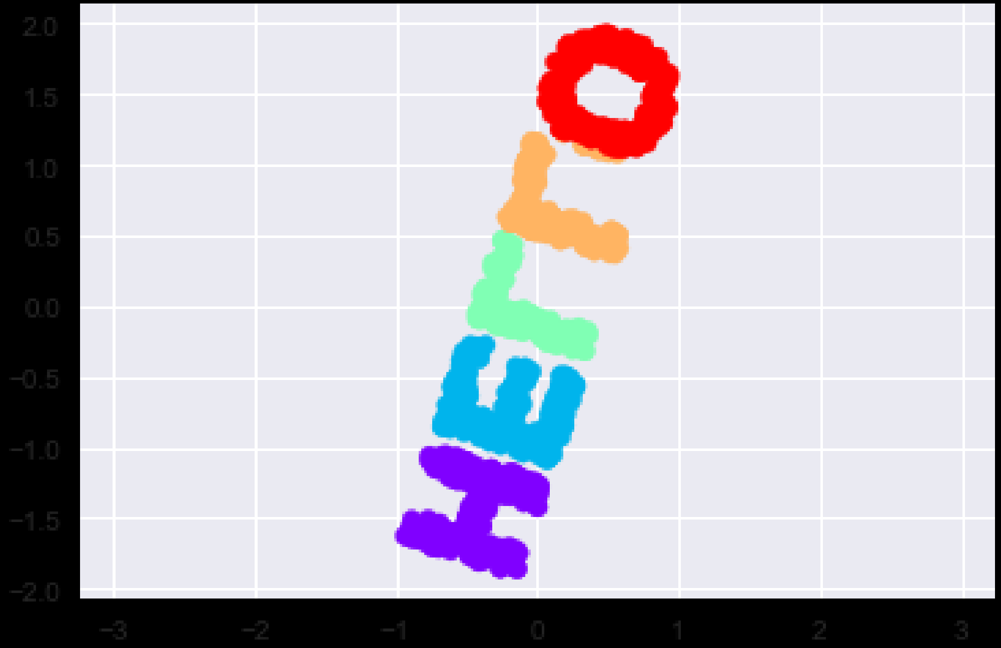
```
[ ]
from sklearn.manifold import MDS
model = MDS(n_components=2, dissimilarity='precomputed', random_state=1)

out = model.fit_transform(D2)

plt.scatter(out[:, 0], out[:, 1], **colorize)
plt.axis('equal');
```

MDS as Manifold Learning

New three dimensional data:

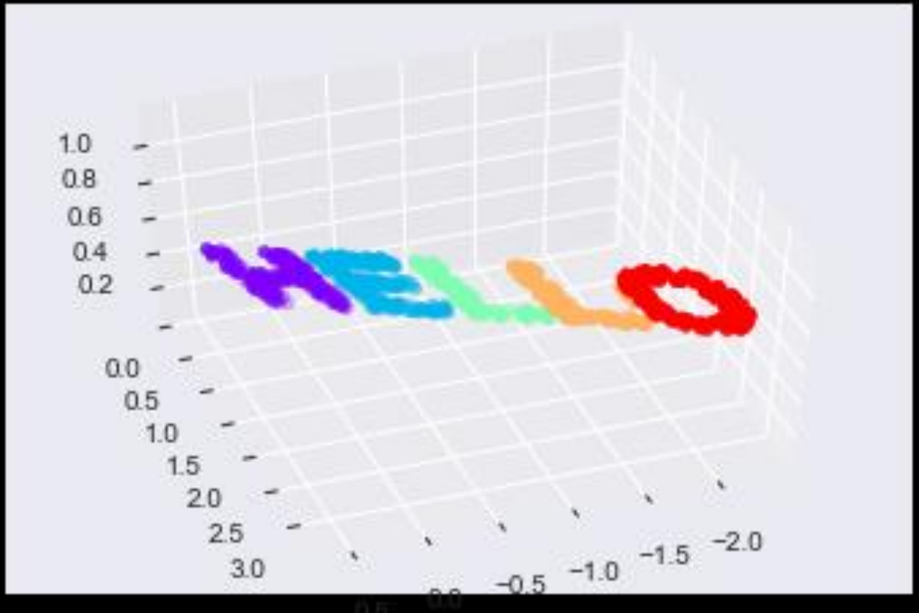


MDS captures following in two dimensions:

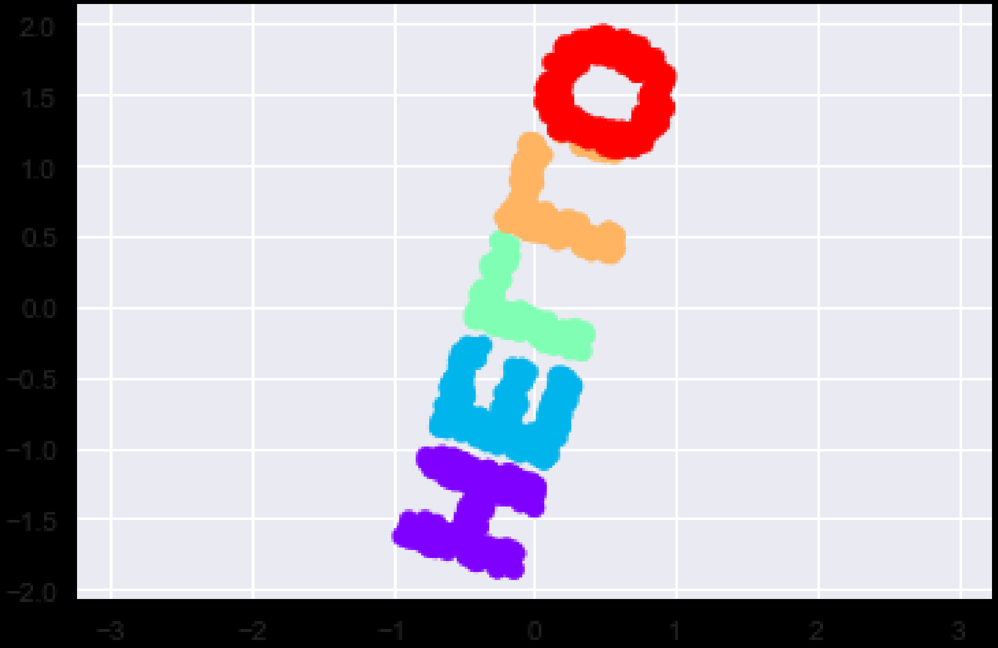


MDS as Manifold Learning

New three dimensional data:



MDS captures following in two dimensions:

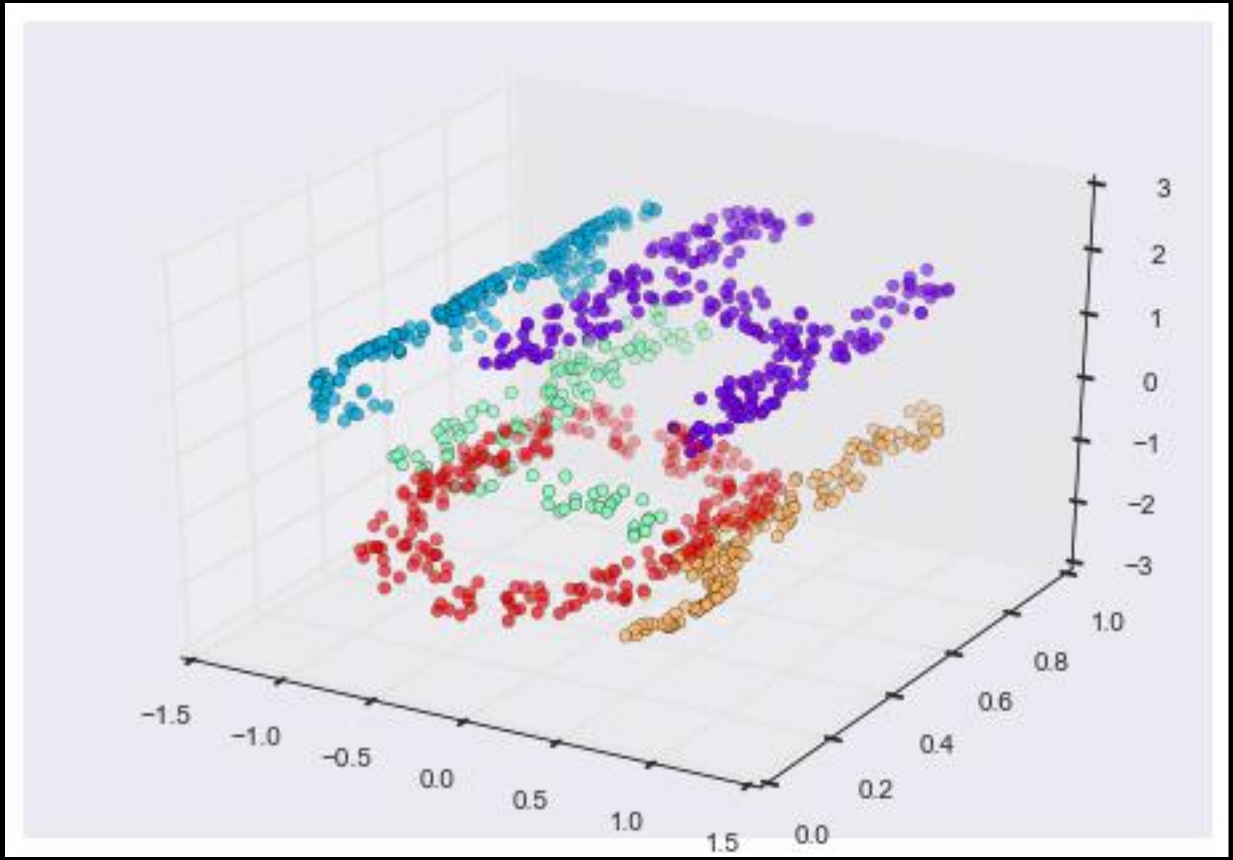


<u>Demonstrates goal of MDS:</u>

Given high-dimensional embedded data, it seeks a low-dimensional representation of the data that preserves certain relationships within the data.

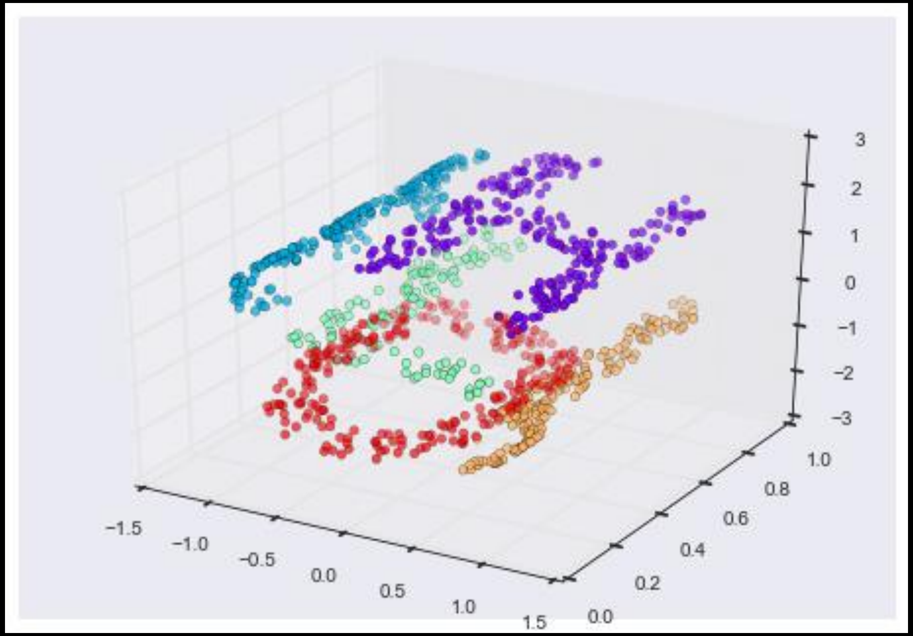
Nonlinear Embeddings: Where MDS Fails

Non linear example of HELLO data
in 3D:

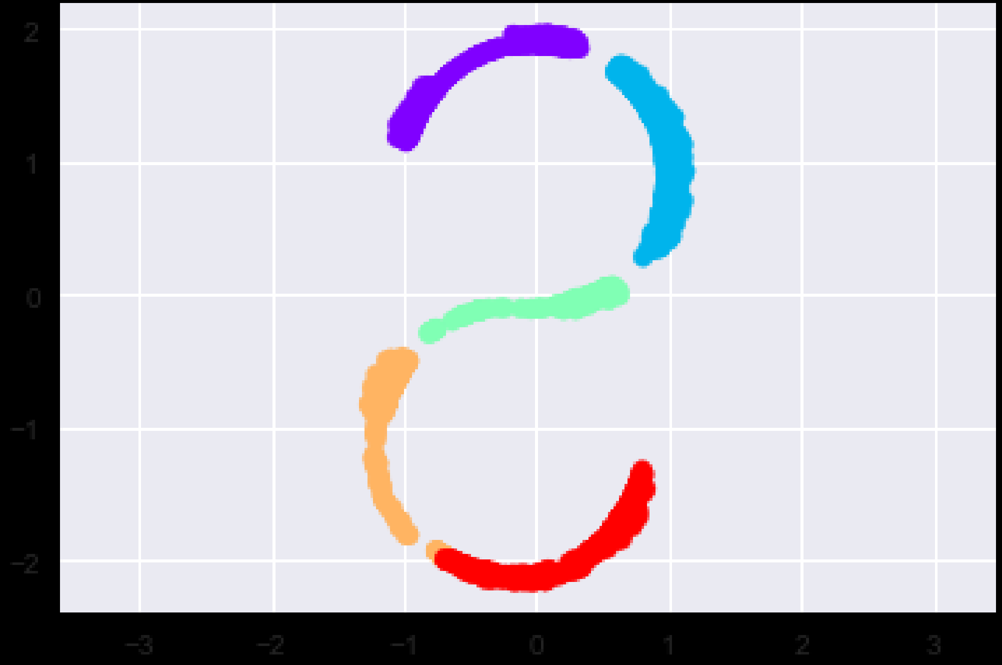


Nonlinear Embeddings: Where MDS Fails

Non linear example of HELLO data in 3D:

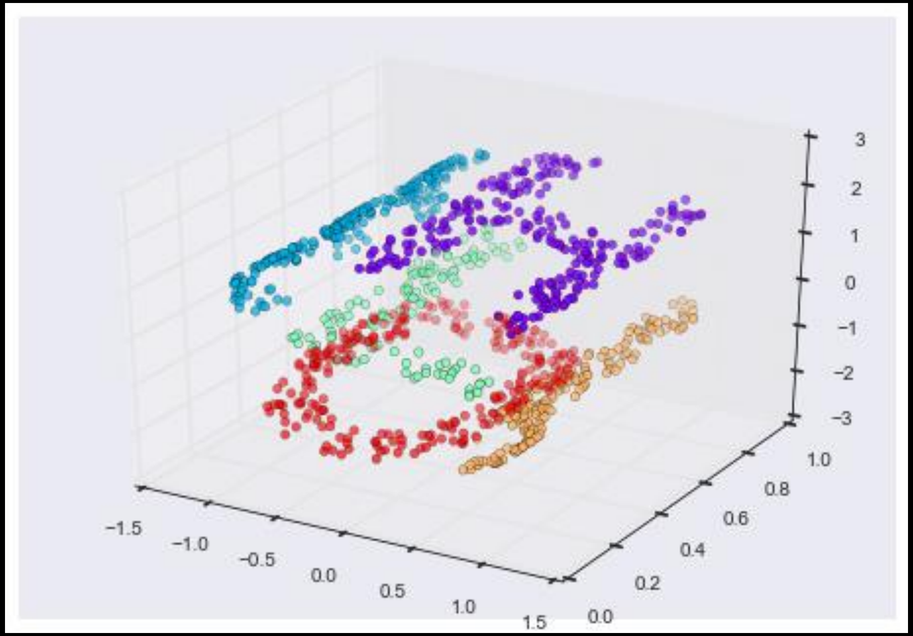


MDS 2D representation of this S shaped HELLO data:

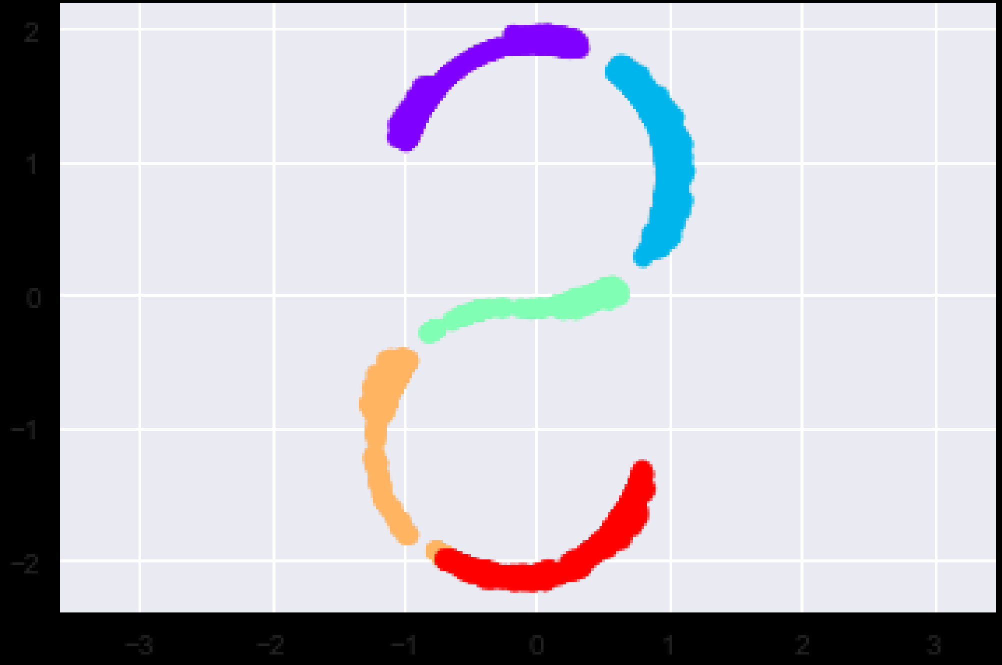


Nonlinear Embeddings: Where MDS Fails

Non linear example of HELLO data in 3D:



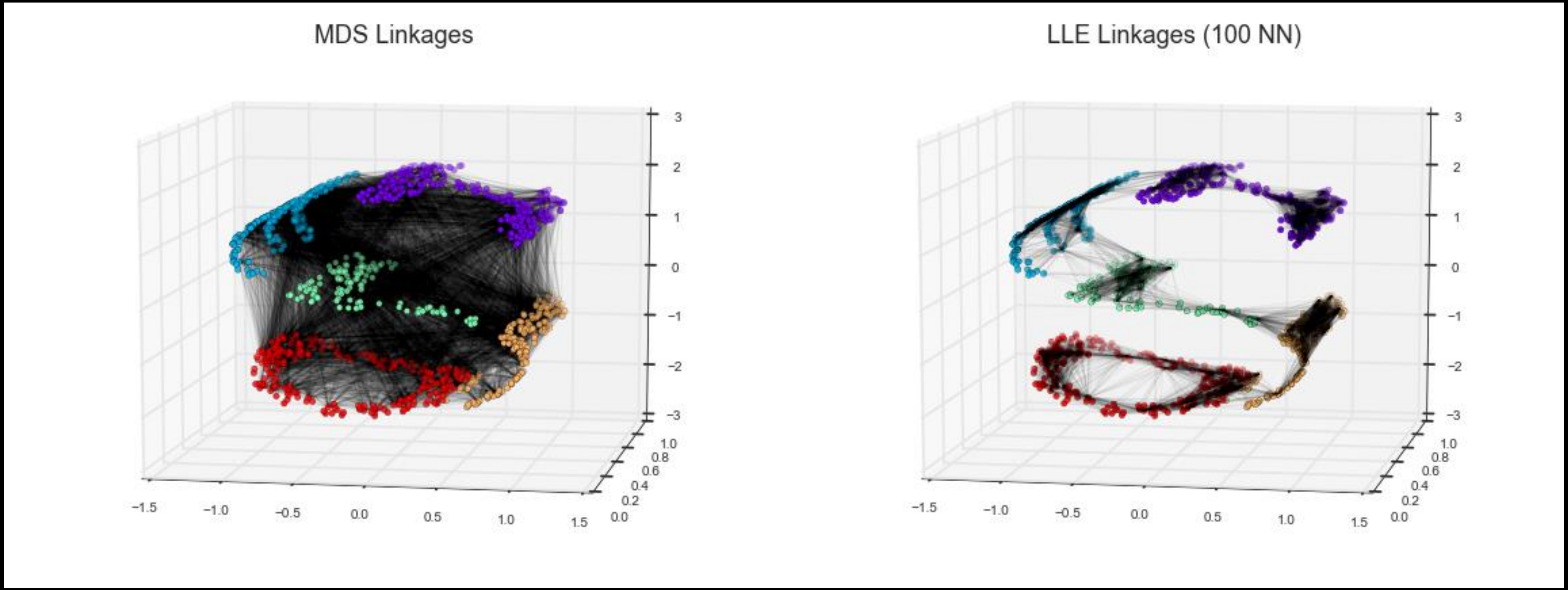
MDS 2D representation of this S shaped HELLO data:



Nonlinear Manifolds: Locally Linear Embedding

MDS Linkages (all pairwise links):

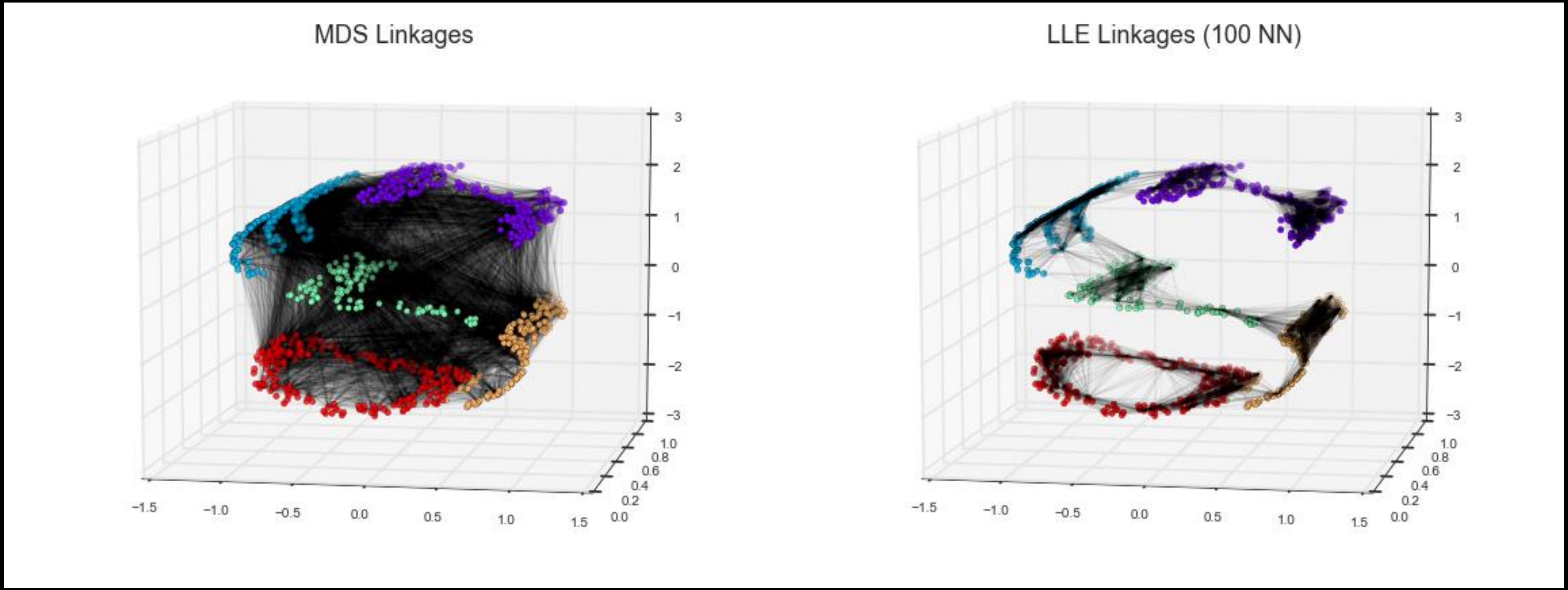
LLE Linkages (Nearest Neighbor links):



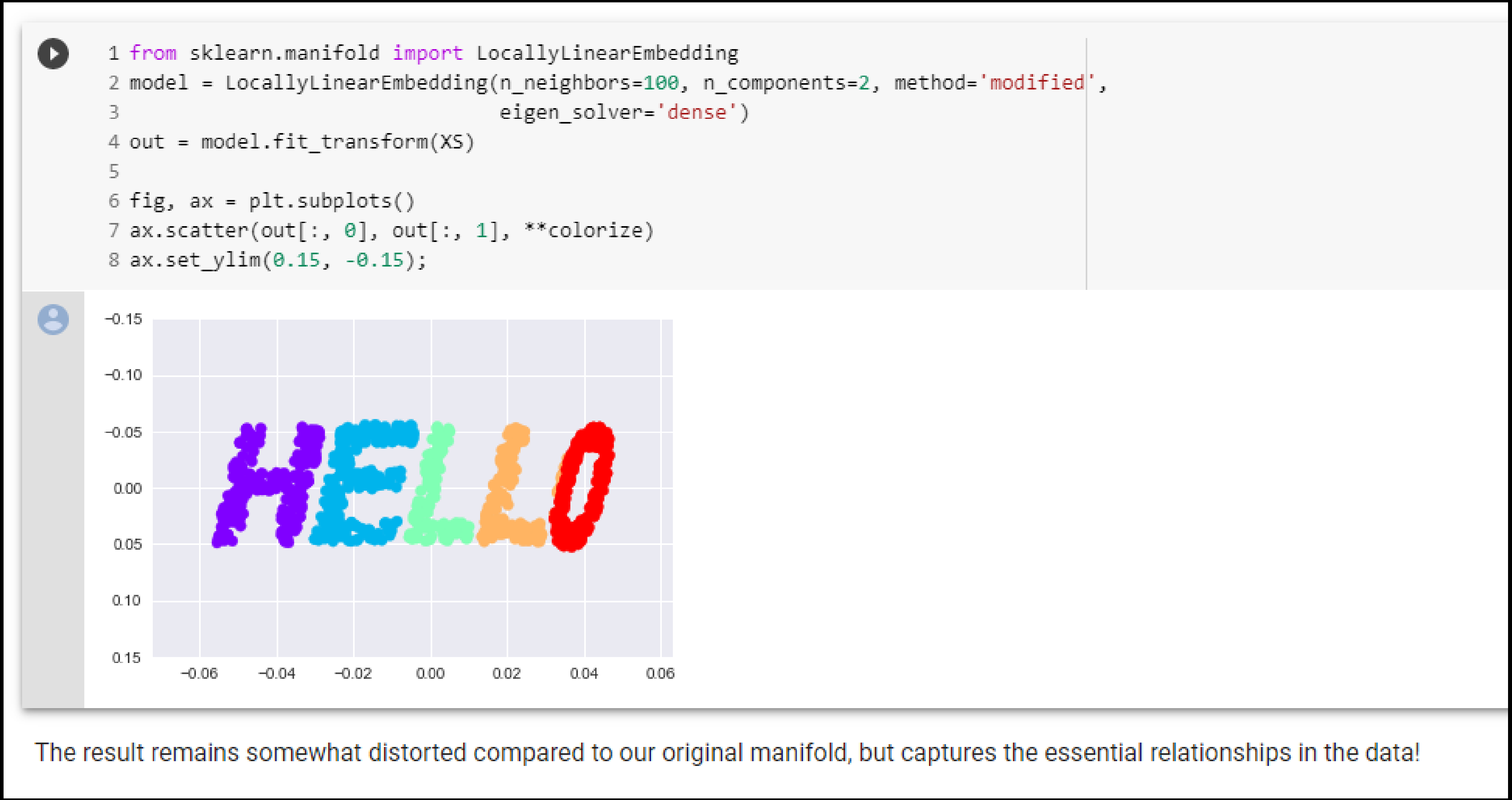
Nonlinear Manifolds: Locally Linear Embedding

MDS Linkages (all pairwise links):

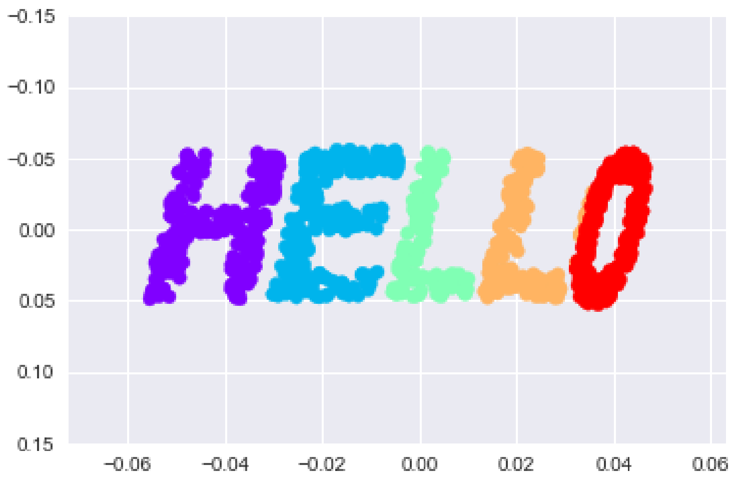
LLE Linkages (Nearest Neighbor links):



Nonlinear Manifolds: Locally Linear Embedding



```
from sklearn.manifold import LocallyLinearEmbedding
model = LocallyLinearEmbedding(n_neighbors=100, n_components=2, method='modified',
                               eigen_solver='dense')
out = model.fit_transform(XS)

fig, ax = plt.subplots()
ax.scatter(out[:, 0], out[:, 1], **colorize)
ax.set_ylim(0.15, -0.15);
```



The result remains somewhat distorted compared to our original manifold, but captures the essential relationships in the data!

Important Notes: Manifold Learning versus PCA

- In practice manifold learning techniques are rarely used for anything more than simple qualitative visualization of high-dimensional data.
- The following are some of the particular challenges of manifold learning, which all contrast poorly with PCA:
    - no good framework for handling missing data. In contrast, there are straightforward iterative approaches for missing data in PCA.
- The presence of noise in the data can "short-circuit" the manifold and drastically change the embedding. In contrast, PCA naturally filters noise from the most important components.
- The manifold embedding result is generally highly dependent on the number of neighbors chosen, and there is generally no solid quantitative way to choose an optimal number of neighbors. In contrast, PCA does not involve such a choice.
- In manifold learning, the globally optimal number of output dimensions is difficult to determine. In contrast, PCA lets you find the output dimension based on the explained variance.
- With all that on the table, the only clear advantage of manifold learning methods over PCA is their ability to preserve nonlinear relationships in the data.

Important Notes: Manifold Learning versus PCA

- Further VanderPlas finds that:
  - For toy problems such as the S-curve we saw before, locally linear embedding (LLE) and its variants (especially modified LLE), perform very well. This is implemented in sklearn.manifold.LocallyLinearEmbedding.
  - For high-dimensional data from real-world sources, LLE often produces poor results, and isometric mapping (IsoMap) seems to generally lead to more meaningful embeddings. This is implemented in sklearn.manifold.Isomap
    - See Manifold Learning .ipynb for IsoMap example