mds_tsne_lab

April 4, 2023

1 Multi-Dimension Scaling

is a distance-preserving manifold learning method. All manifold learning algorithms assume the dataset lies on a smooth, non linear manifold of low dimension and that a mapping f: R^D -> R^d (D»d) can be found by preserving one or more properties of the higher dimension space. Distance preserving methods assume that a manifold can be defined by the pairwise distances of its points. In distance preserving methods, a low dimensional embedding is obtained from the higher dimension in such a way that pairwise distances between the points remain same. Some distance preserving methods preserve spatial distances (MDS) while some preserve graph distances.

MDS is not a single method but a family of methods. MDS takes a dissimilarity matrix D where D_{ij} represents the dissimilarity between points i and j and produces a mapping on a lower dimension, preserving the dissimilarities as closely as possible. The dissimilarity matrix could be observed or calculated from the given dataset.

MDS can be divided into two categories:

- **Metric MDS** - Metric MDS is used for quantitative data and tries to preserve the original dissimilarity metrics.
- **Non-Metric MDS** - Non-metric MDS is used for ordinal data. It tries to keep the order of dissimialrity metrics intact.

2 t-Distributed Stochastic Neighbor Embedding (t-SNE)

t-Distributed Stochastic Neighbor Embedding (t-SNE) is a non-linear technique for dimensionality reduction that is particularly well suited for the visualization of high-dimensional datasets. It is extensively applied in image processing, NLP, genomic data and speech processing. To keep things simple, here's a brief overview of working of t-SNE:

- The algorithms starts by calculating the probability of similarity of points in high-dimensional space and calculating the probability of similarity of points in the corresponding low-dimensional space. The similarity of points is calculated as the conditional probability that a point A would choose point B as its neighbor if neighbors were picked in proportion to their probability density under a Gaussian (normal distribution) centered at A.
- It then tries to minimize the difference between these conditional probabilities (or similarities) in higher-dimensional and lower-dimensional space for a perfect representation of data points in lower-dimensional space.

- To measure the minimization of the sum of difference of conditional probability t-SNE minimizes the sum of Kullback-Leibler divergence of overall data points using a gradient descent method.

In simpler terms, t-Distributed stochastic neighbor embedding (t-SNE) minimizes the divergence between two distributions: a distribution that measures pairwise similarities of the input objects and a distribution that measures pairwise similarities of the corresponding low-dimensional points in the embedding.

In this way, t-SNE maps the multi-dimensional data to a lower dimensional space and attempts to find patterns in the data by identifying observed clusters based on similarity of data points with multiple features. However, after this process, the input features are no longer identifiable, and you cannot make any inference based only on the output of t-SNE. Hence it is mainly a data exploration and visualization technique.

I recommend to familiarize with this amazing blogpost https://distill.pub/2016/misread-tsne/ which graphically shows traps associated with use of t-SNE and how hyperparameters selection can affect final embeding.

2.0.1 To better illustrate properties of individual methods of reducing dimensionality (from this and previous lab), create a set of points in the shape of a roulade. To do it:

1. Prepare a point generator for the circles in this generator, ensure that the radius decreases with the next steps (in this way you will get a spiral)
2. Treat the spiral as a cross-sectional solid, creating a croissant / crescent roll
3. Move the points in the roll with a slight noise to avoid their perfect placement
4. For better visualization, you can give individual points a color depending, e.g. on the distance from the center of the roulade
5. Generated roulade should resemble the picture below

[ ]:
```
from time import time
import math
import random
#import mkl
import glob

import numpy as np
import pandas as pd
import seaborn as sn
import matplotlib.pyplot as plt
from scipy import stats
from mpl_toolkits.mplot3d import Axes3D
from PIL import Image
from matplotlib.cbook import get_sample_data
from matplotlib.offsetbox import TextArea, DrawingArea, OffsetImage,␣
↪AnnotationBbox
```

```
from IPython.display import Image
import tensorflow as tf
from tensorflow import keras

sn.set(color_codes=True)
np.random.seed(1234)
#mkl.set_num_threads(4)
%matplotlib inline
%config InlineBackend.figure_format = 'retina'
plt.rcParams["figure.figsize"] = [20, 20]
```

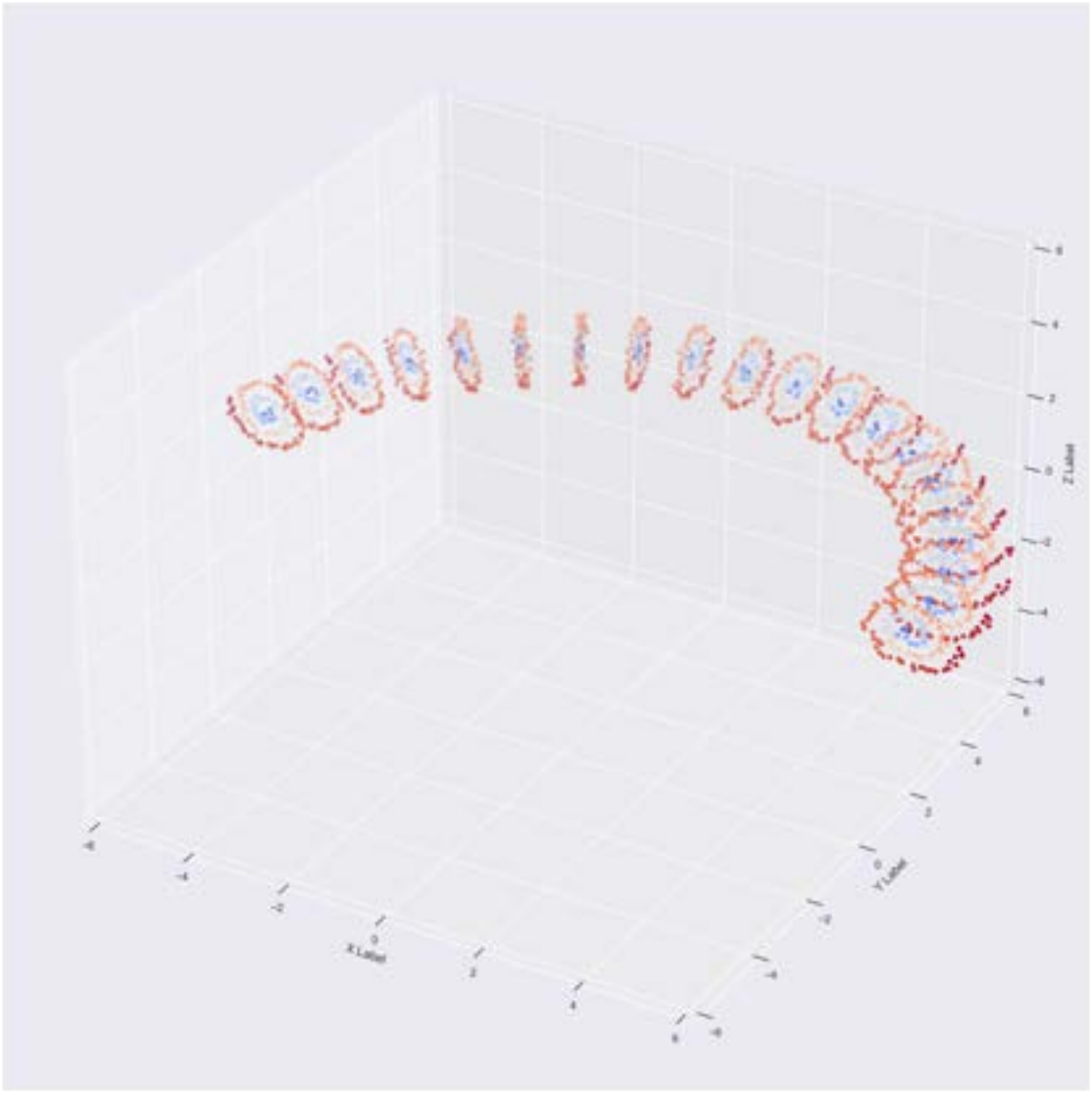
2.1 Roulade points generator

[ ]:
```
def random_spiral_point(density=2):
    p = random.random()**0.5
    return p*math.cos(p*density*np.pi*2), p*math.sin(p*density*np.pi*2)

def uniform_spiral(density=3, steps = 100):
    x, y = [] ,[]
    for i in range(steps):
        x.append((i/steps)**0.5*math.cos((i/steps)**0.5*density*np.pi*2))
        y.append((i/steps)**0.5*math.sin((i/steps)**0.5*density*np.pi*2))

    return x, y

def spiral2roll(xli, yli, width=7, angle=0.1):
    nx, ny, nz, d = [], [], [], []
    for x,y in zip(xli, yli):
        p = random.random()
        nx.append((width+x)*math.cos(angle*p*np.pi*2))
        ny.append((width+x)*math.sin(angle*p*np.pi*2))
        nz.append(y)
        d.append((x**2+y**2)**0.5)
    return nx, ny, nz, d

def uniform_roll(xli, yli, width=7, angle=0.25, density=5, noise_factor=0.05):
    nx, ny, nz, d = [], [], [], []
    d1 = [((x**2+y**2)**0.5) for x,y in zip(xli, yli)]
    for i in range(density):
        nx.extend([(width+x)*math.cos(angle*i/density*np.pi*2) + (random.
 ↪random()-0.5)*noise_factor for x in xli])
        ny.extend([(width+x)*math.sin(angle*i/density*np.pi*2) + (random.
 ↪random()-0.5)*noise_factor  for x in xli])
        nz.extend(yli)
        d.extend(d1)
    return nx, ny, nz, d
```

```
def roll_generator(to_array=True, spiral_density=3, spiral_steps = 20, width=7,␣
 ↪angle=0.3, density=5):
    x, y = uniform_spiral(density=spiral_density, steps = spiral_steps)
    x,y,z,d = uniform_roll(x, y, width=width, angle=angle, density=density)

    if to_array:
        return np.array([x,y,z]).T, d
    else:
        return x,y,z,d
```

[ ]:
```
def plot_roulade(x,y,z,d=None):
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    ax.scatter(x, y, z, c=d, cmap=plt.cm.coolwarm)
    ax.set_xlabel('X Label')
    ax.set_ylabel('Y Label')
    ax.set_zlabel('Z Label')
    ax.set_xlim([-6, 6])
    ax.set_ylim([-6, 6])
    ax.set_zlim([-6, 6])
```

[ ]:
```
xl, yl = uniform_spiral()
x,y,z,d = uniform_roll(xl,yl, width=7, angle=0.5, density=20, noise_factor=0.15)
plot_roulade(x,y,z,d=d)
```



Visualize the prepared set of points by projecting onto a 2D plane, changing the following aspects in subsequent approaches:

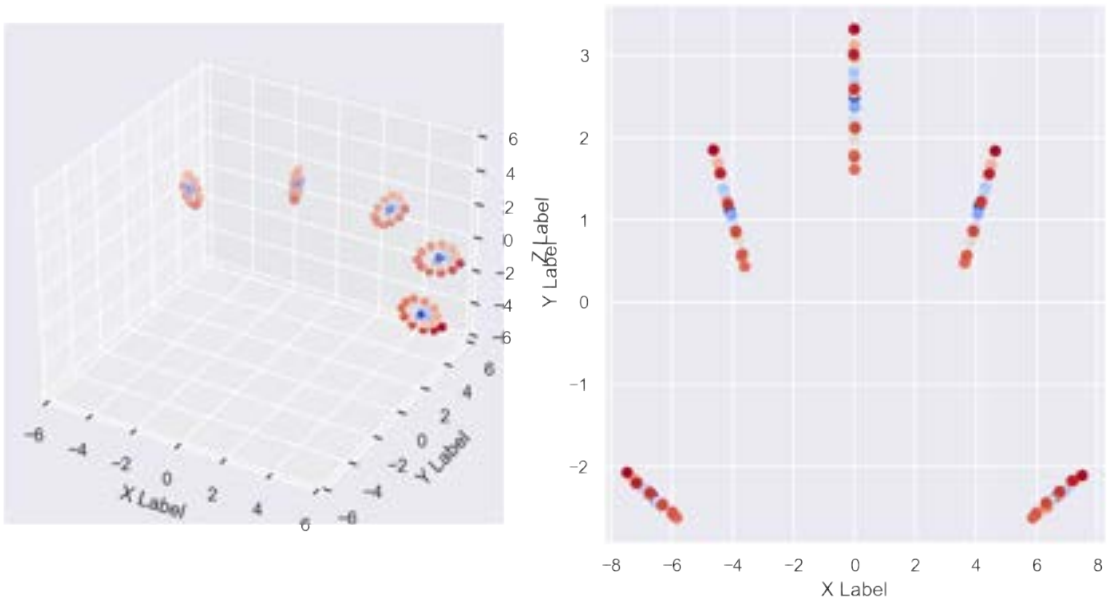
1. Density of points generated, in terms of points per spiral
2. Distance between roll layers, layer density
3. Visualization method: PCA, KernelPCA, MDS, t-SNE
4. generate at least a few plots per method

Save each generated chart, you will later need it to embeding images

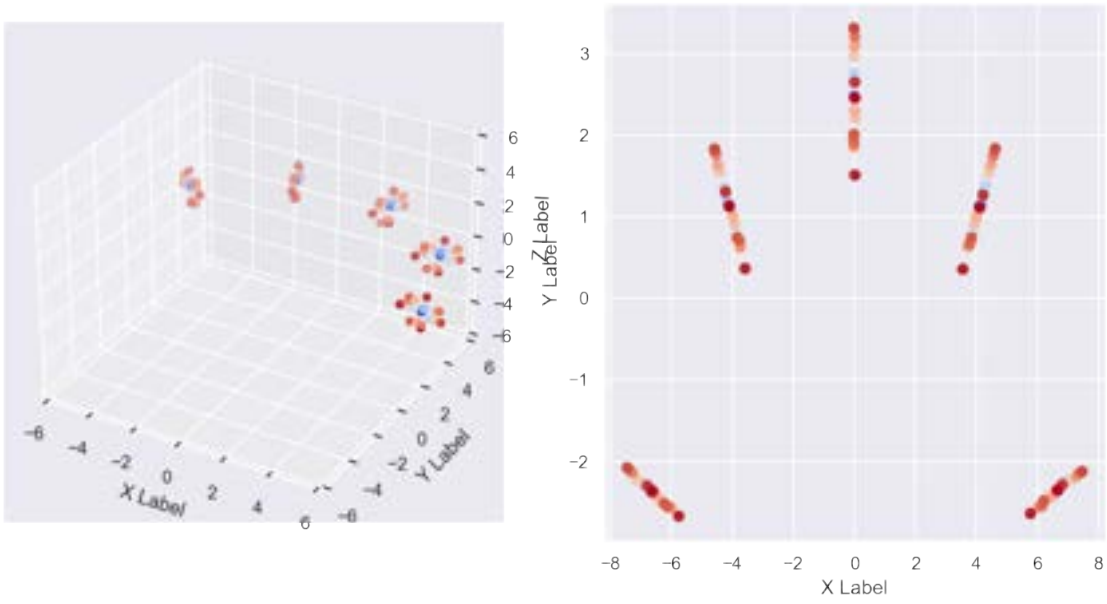
PCA implementation

[ ]:
```
def show_PCA(spiral_density=3, spiral_steps = 20, width=7, angle=0.5,␣
 ↪roll_density=5):
    fig = plt.figure()
    x,y,z,d=roll_generator(to_array=False, spiral_density=spiral_density,
                           spiral_steps = spiral_steps, width=width,
                           angle=angle, density=roll_density)
    from mpl_toolkits.mplot3d import Axes3D

    # fig = plt.figure()
    ax = fig.add_subplot(121, projection='3d')
    ax.scatter(x, y, z, c=d, cmap=plt.cm.coolwarm)
    ax.set_xlabel('X Label')
    ax.set_ylabel('Y Label')
    ax.set_zlabel('Z Label')
    ax.set_xlim([-6, 6])
    ax.set_ylim([-6, 6])
    ax.set_zlim([-6, 6])

    ax2 = fig.add_subplot(122)
    points, d=roll_generator(to_array=True, spiral_density=spiral_density,
                           spiral_steps = spiral_steps, width=width,
                           angle=angle, density=roll_density)

    from sklearn.decomposition import PCA
    embedding = PCA(n_components=2)
    points_transformed = embedding.fit_transform(points)
    points_transformed.shape

    points_transformed_t = points_transformed.T
    points_transformed_t.shape

    ax2.scatter(points_transformed_t[0], points_transformed_t[1], c=d, cmap=plt.
 ↪cm.coolwarm)
    ax2.set_xlabel('X Label')
    ax2.set_ylabel('Y Label')

    plt.rcParams["figure.figsize"] = [12, 6]

    extent = ax2.get_window_extent().transformed(fig.dpi_scale_trans.inverted())
    fig.
 ↪savefig(f'pca_spst{spiral_steps}_roldens{roll_density}_w{width}_a{angle}_sd{spiral_density}
 ↪png', bbox_inches=extent)
```

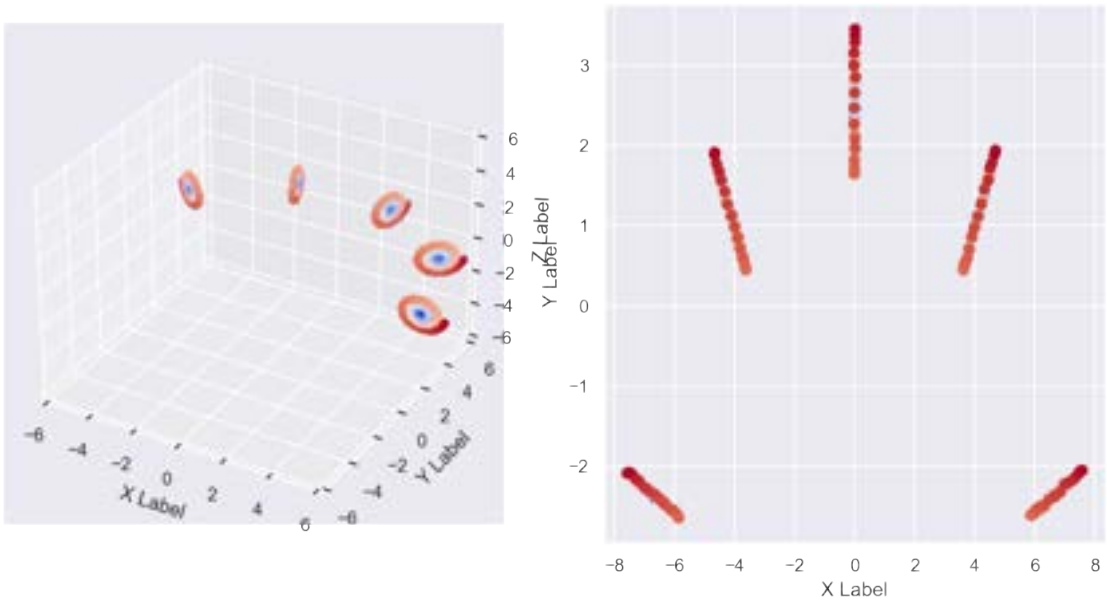
[ ]:
```
# Default params
show_PCA(spiral_density=3, spiral_steps = 20, width=7, angle=0.5,␣
 ↪roll_density=5)
```



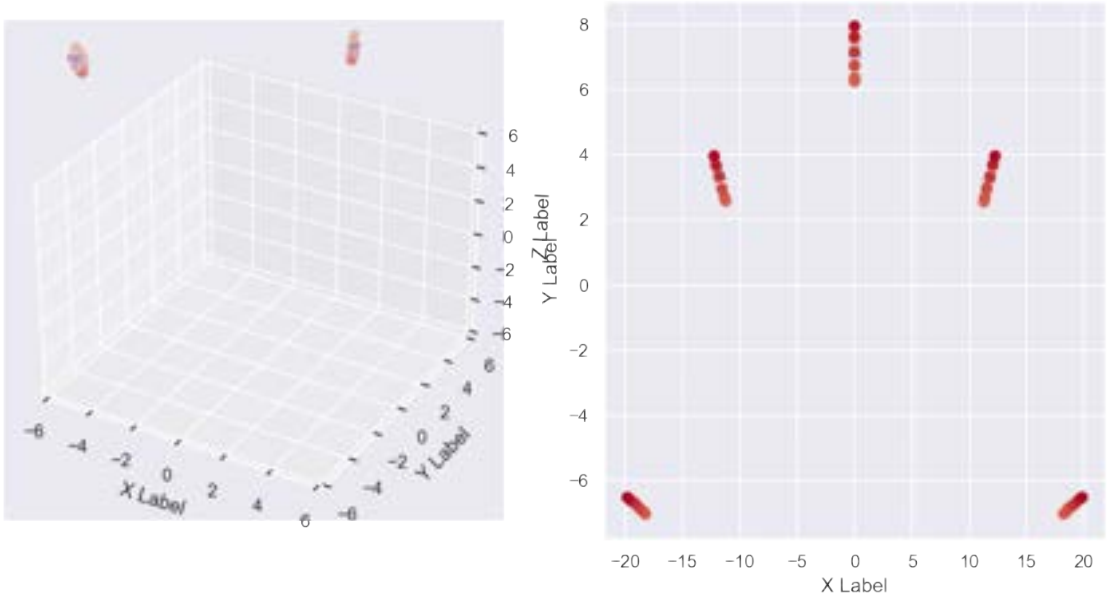
[ ]:
```
show_PCA(spiral_density=10, spiral_steps = 20, width=7, angle=0.5,␣
↪roll_density=5)
```



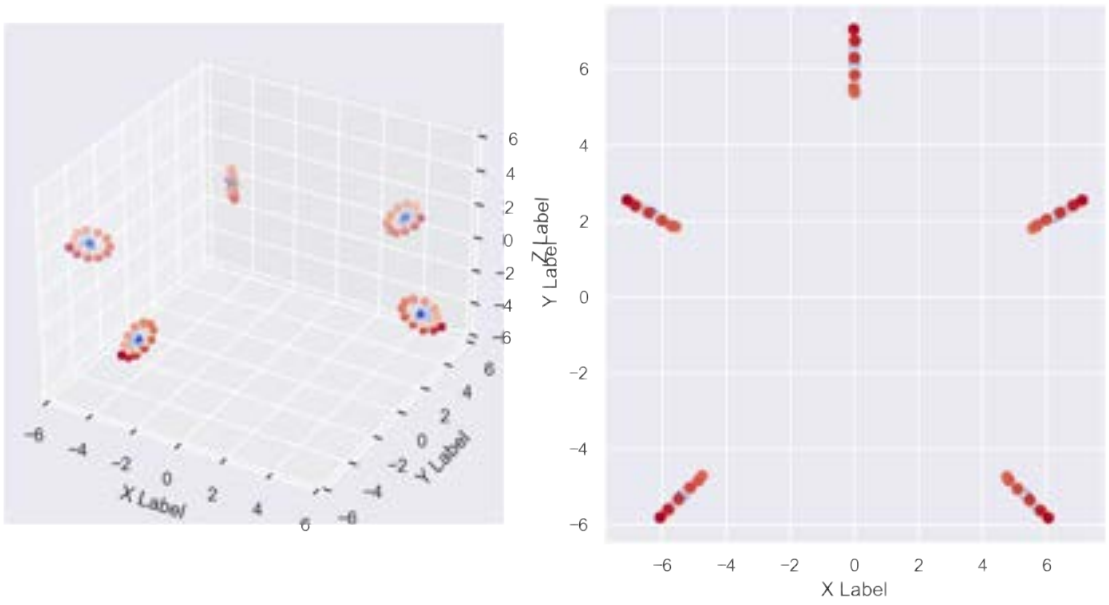
[ ]:
```
show_PCA(spiral_density=3, spiral_steps = 50, width=7, angle=0.5,␣
↪roll_density=5)
```



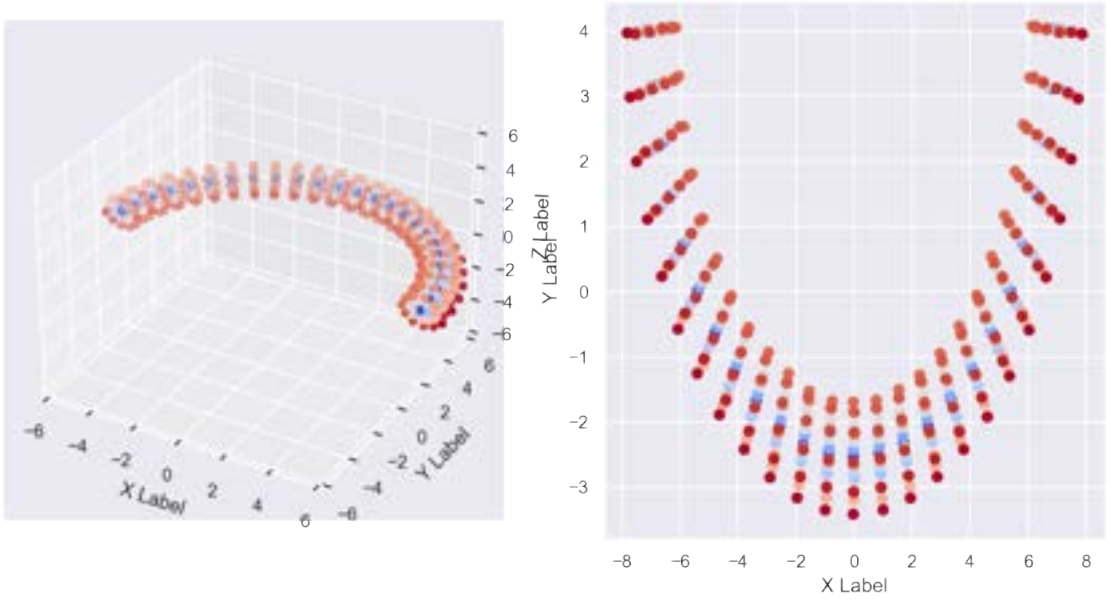
[ ]:
```
show_PCA(spiral_density=3, spiral_steps = 20, width=20, angle=0.5,␣
↪roll_density=5)
```



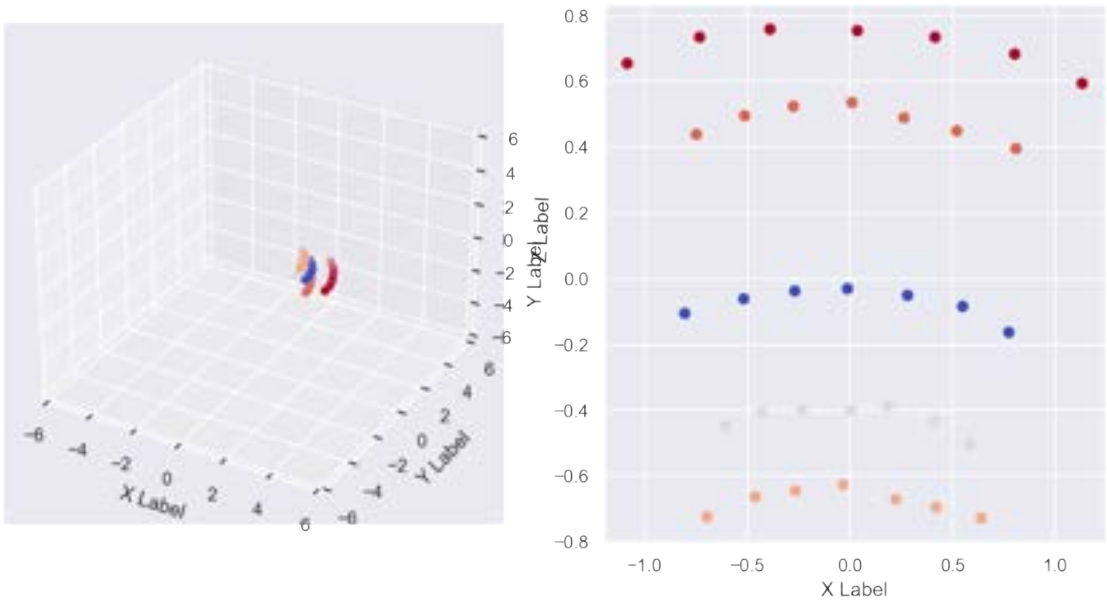
[ ]:
```
show_PCA(spiral_density=3, spiral_steps = 20, width=7, angle=0.9,␣
↪roll_density=5)
```



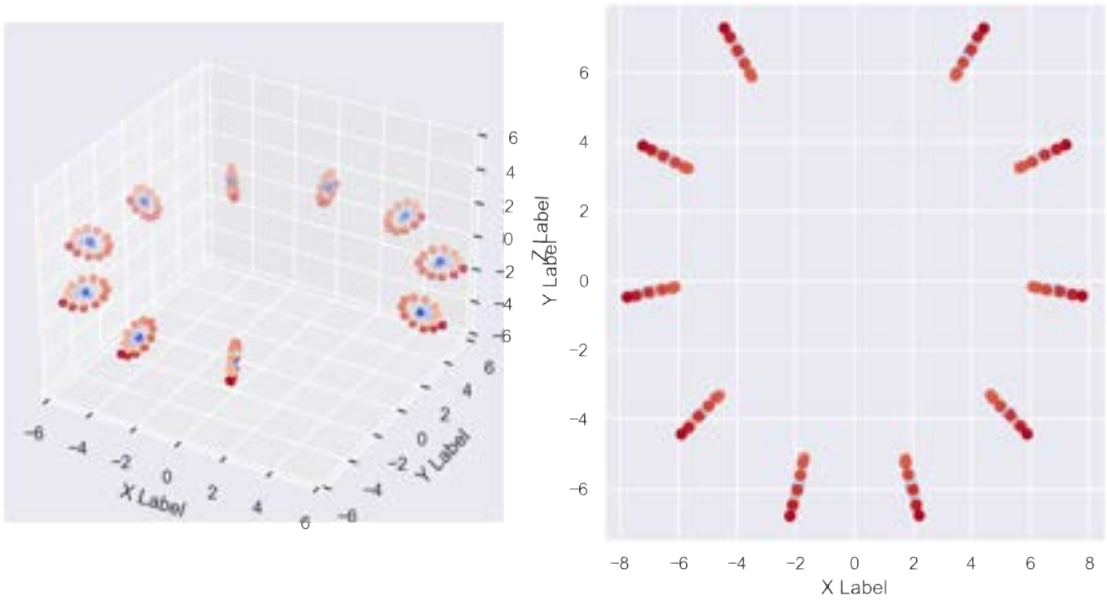
[ ]:
```
show_PCA(spiral_density=3, spiral_steps = 20, width=7, angle=0.5,␣
↪roll_density=25)
```



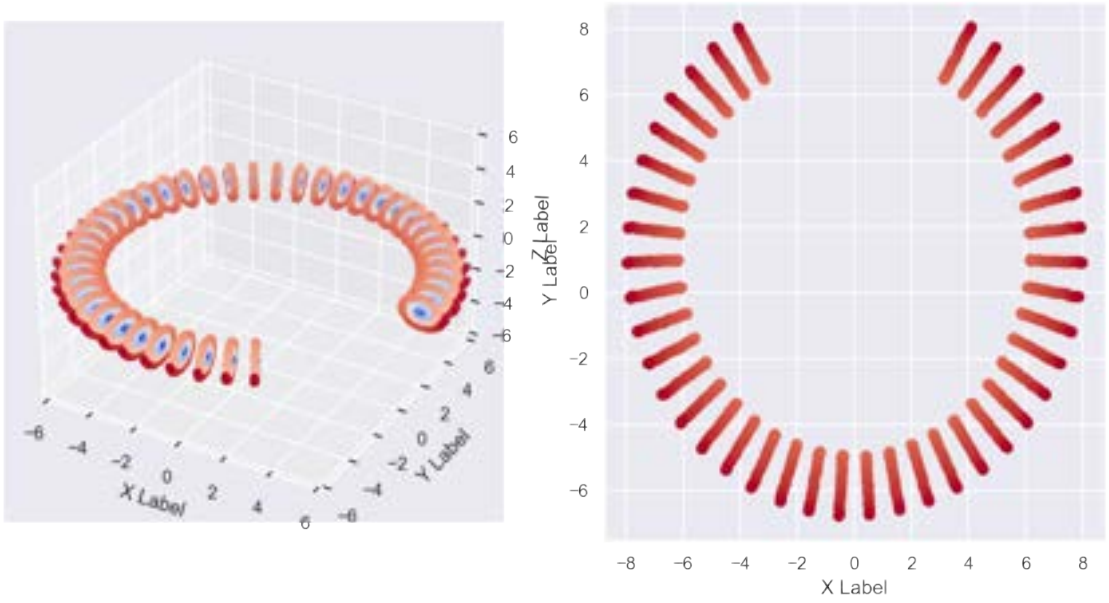
[ ]:
```
show_PCA(spiral_density=10, spiral_steps = 5, width=2, angle=0.15,␣
↪roll_density=7)
```



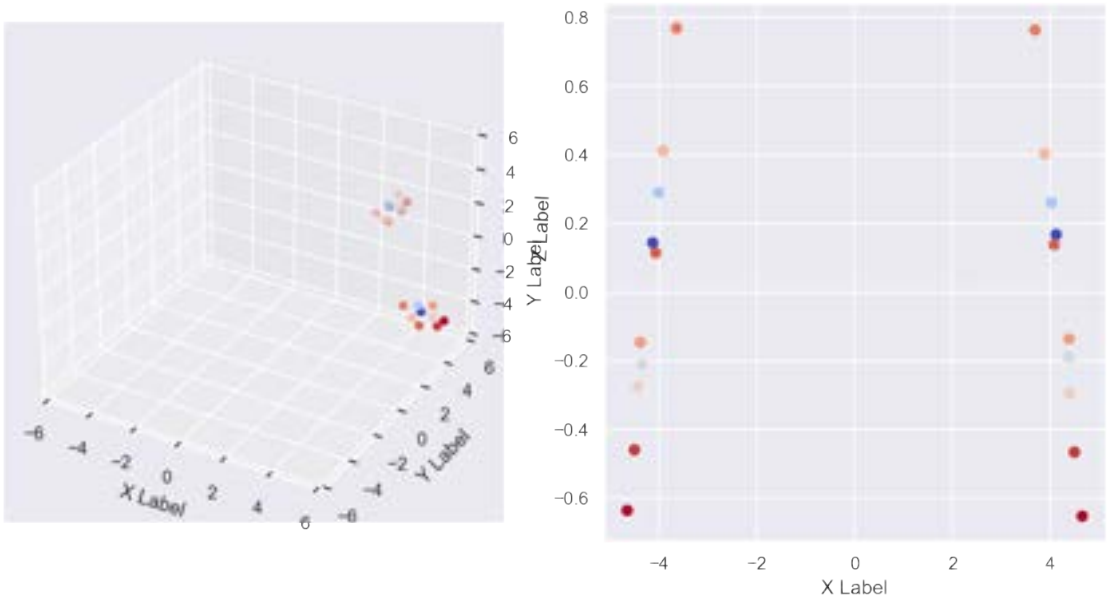
[ ]:
```
show_PCA(spiral_density=3, spiral_steps = 20, width=7, angle=0.9,␣
↪roll_density=10)
```



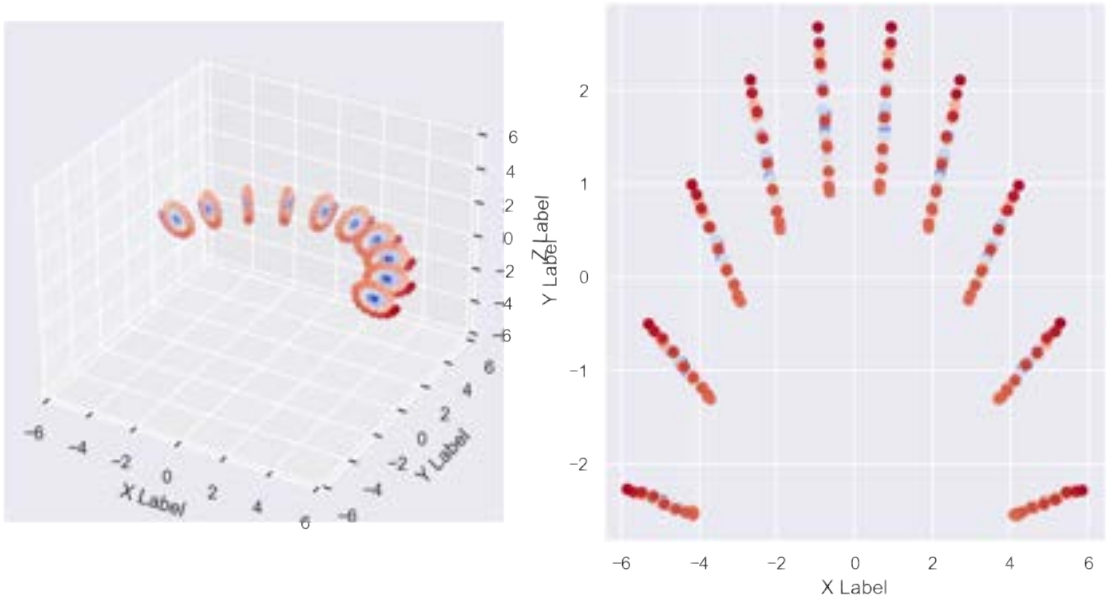
[ ]:
```
show_PCA(spiral_density=3, spiral_steps = 60, width=7, angle=0.85,␣
↪roll_density=40)
```



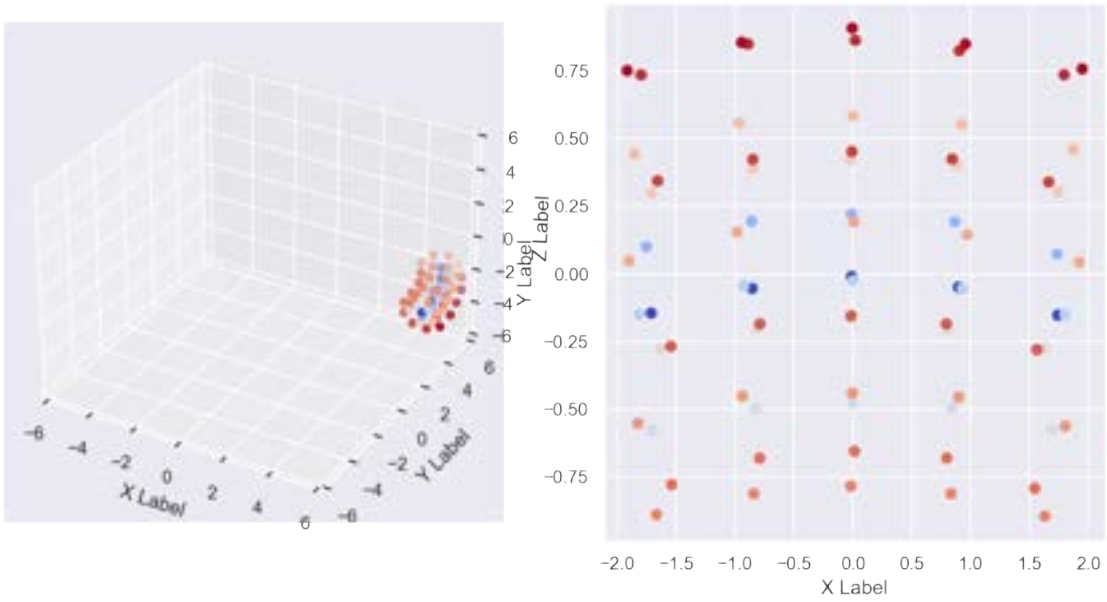
[ ]:
```
show_PCA(spiral_density=20, spiral_steps = 10, width=7, angle=0.4,␣
↪roll_density=2)
```



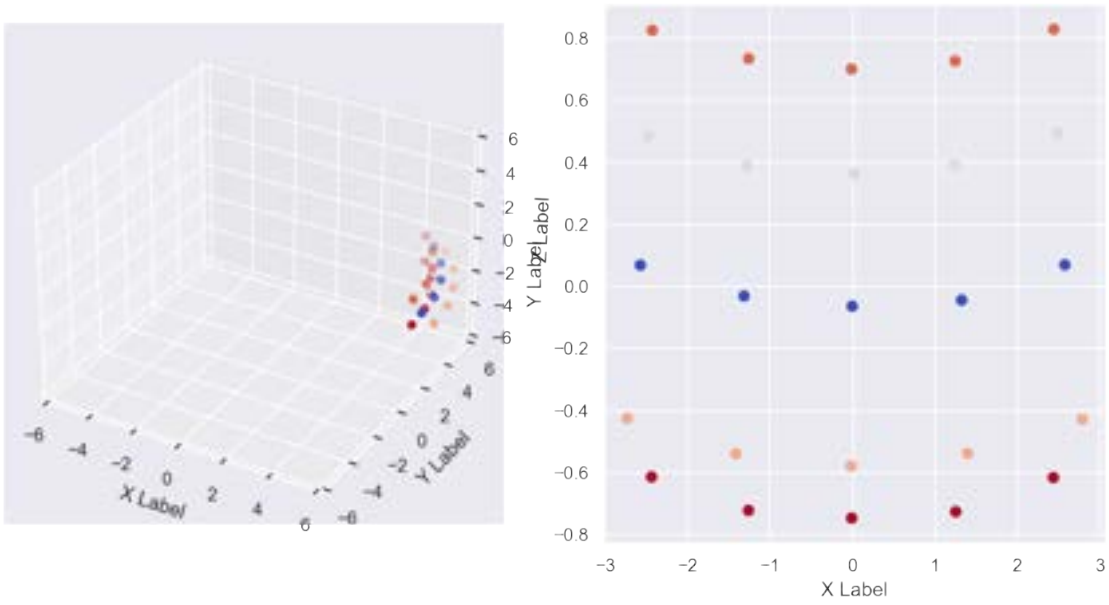
[ ]:
```
show_PCA(spiral_density=3, spiral_steps = 30, width=5, angle=0.5,␣
↪roll_density=10)
```



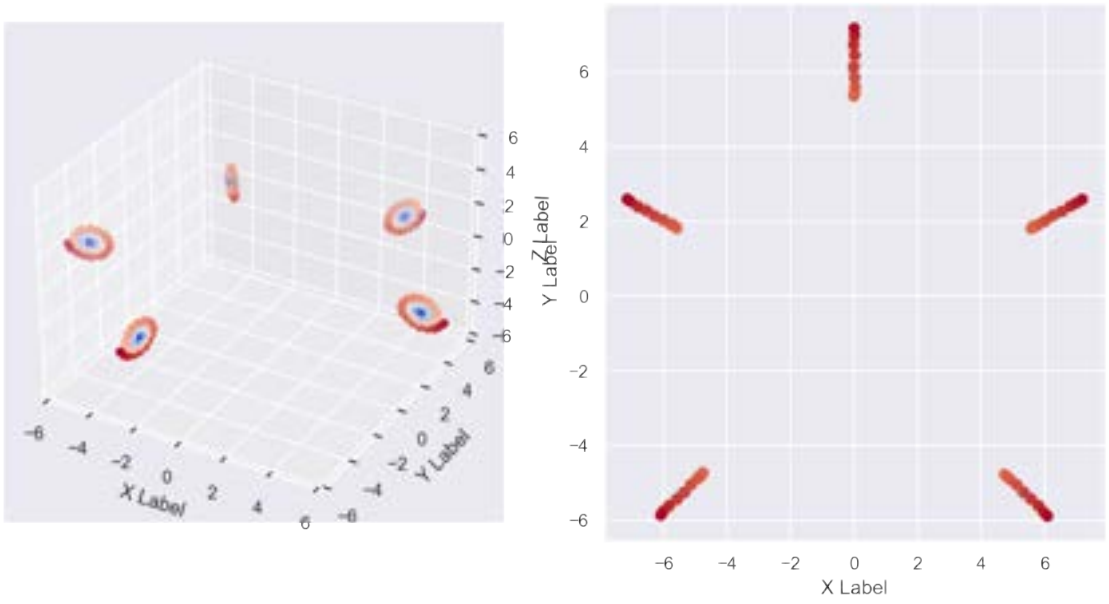
[ ]:
```
show_PCA(spiral_density=3, spiral_steps = 15, width=7, angle=0.1,␣
↪roll_density=5)
```



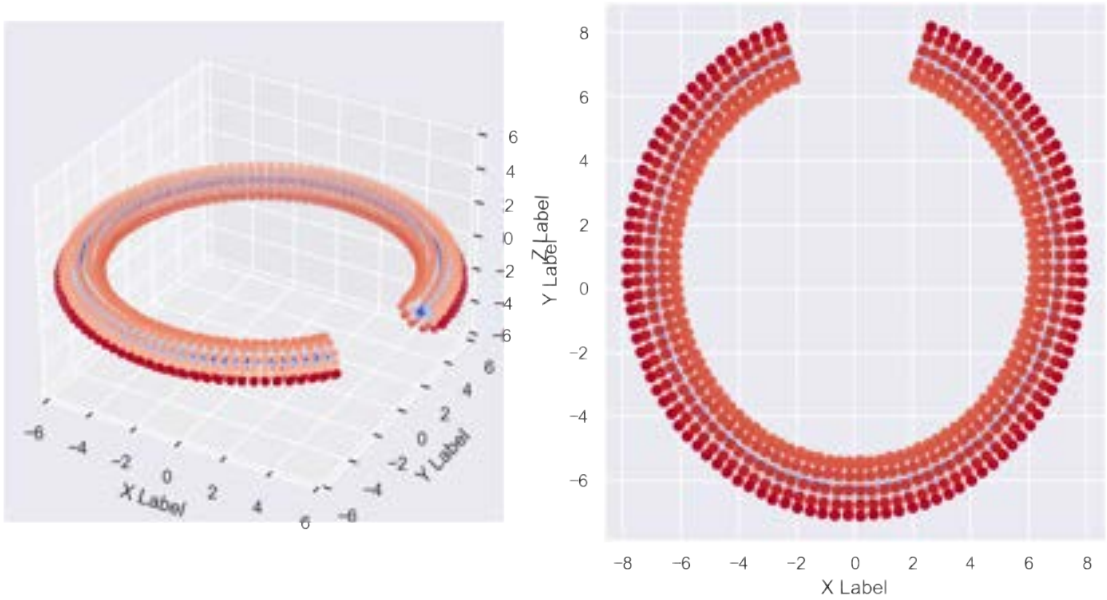
[ ]:
```
show_PCA(spiral_density=3, spiral_steps = 5, width=7, angle=0.15,␣
↪roll_density=5)
```



[ ]:
```
show_PCA(spiral_density=3, spiral_steps = 30, width=7, angle=0.9,␣
↪roll_density=5)
```



[ ]:
```
show_PCA(spiral_density=3, spiral_steps = 20, width=7, angle=0.9,␣
↪roll_density=100)
```

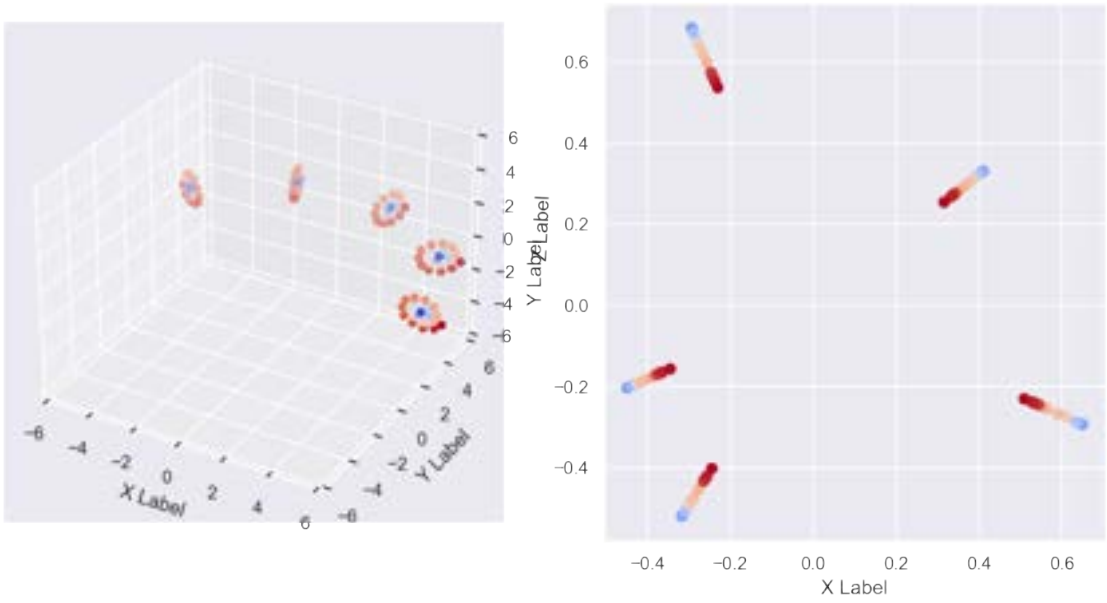


KernelPCA implementation

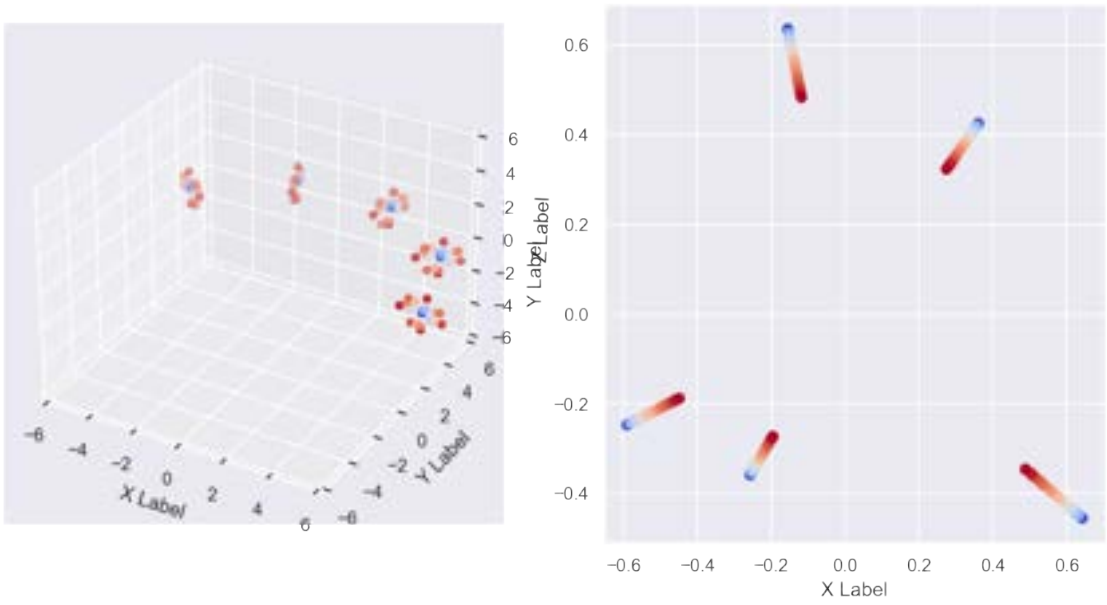
[ ]:
```
def show_KernelPCA(spiral_density=3, spiral_steps = 20, width=7, angle=0.5,␣
↪roll_density=5):
    fig = plt.figure()
    x,y,z,d=roll_generator(to_array=False, spiral_density=spiral_density,
                           spiral_steps = spiral_steps, width=width,
                           angle=angle, density=roll_density)
    from mpl_toolkits.mplot3d import Axes3D

    # fig = plt.figure()
    ax = fig.add_subplot(121, projection='3d')
    ax.scatter(x, y, z, c=d, cmap=plt.cm.coolwarm)
    ax.set_xlabel('X Label')
    ax.set_ylabel('Y Label')
    ax.set_zlabel('Z Label')
    ax.set_xlim([-6, 6])
    ax.set_ylim([-6, 6])
    ax.set_zlim([-6, 6])

    ax2 = fig.add_subplot(122)
    points, d=roll_generator(to_array=True, spiral_density=spiral_density,
                           spiral_steps = spiral_steps, width=width,
                           angle=angle, density=roll_density)

    from sklearn.decomposition import KernelPCA
    embedding = KernelPCA(n_components=2, kernel="rbf")
```

```
    points_transformed = embedding.fit_transform(points)
    points_transformed.shape

    points_transformed_t = points_transformed.T
    points_transformed_t.shape

    ax2.scatter(points_transformed_t[0], points_transformed_t[1], c=d, cmap=plt.
 ↪cm.coolwarm)
    ax2.set_xlabel('X Label')
    ax2.set_ylabel('Y Label')

    plt.rcParams["figure.figsize"] = [12, 6]

    extent = ax2.get_window_extent().transformed(fig.dpi_scale_trans.inverted())
    fig.
 ↪savefig(f'kpca_spst{spiral_steps}_roldens{roll_density}_w{width}_a{angle}_sd{spiral_density
 ↪png', bbox_inches=extent)
```

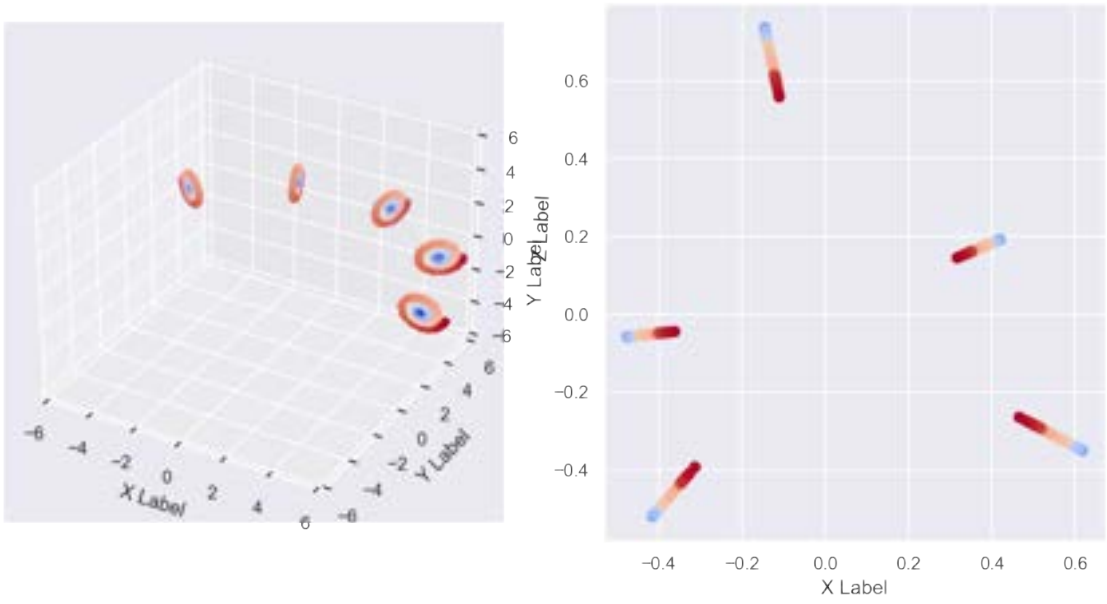
[ ]:
```
# Default params
show_KernelPCA(spiral_density=3, spiral_steps = 20, width=7, angle=0.5,␣
 ↪roll_density=5)
```



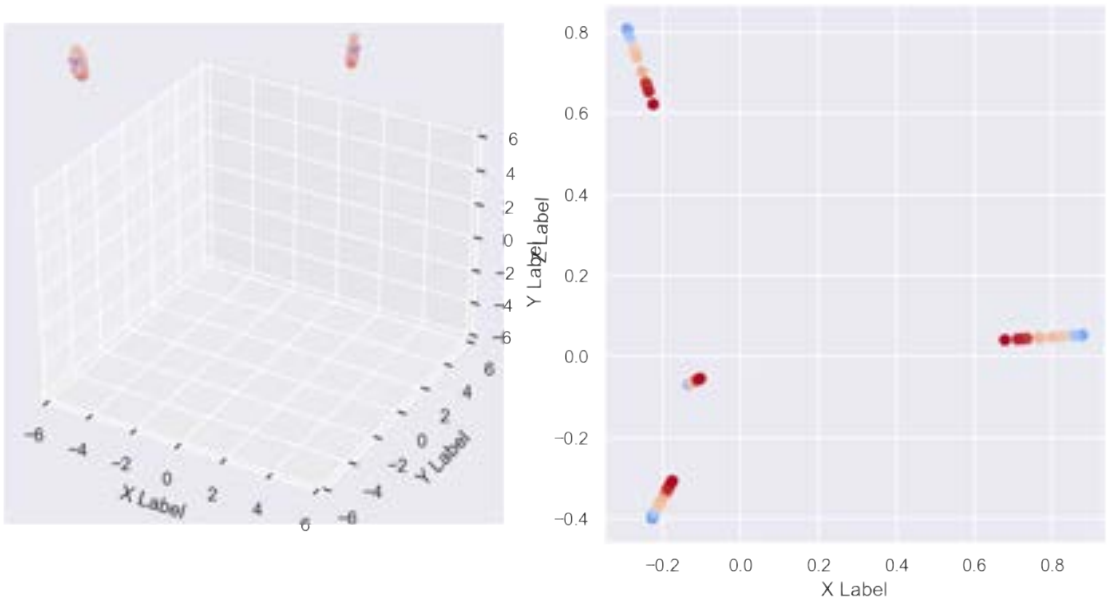
[ ]:
```
show_KernelPCA(spiral_density=10, spiral_steps = 20, width=7, angle=0.5,␣
 ↪roll_density=5)
```



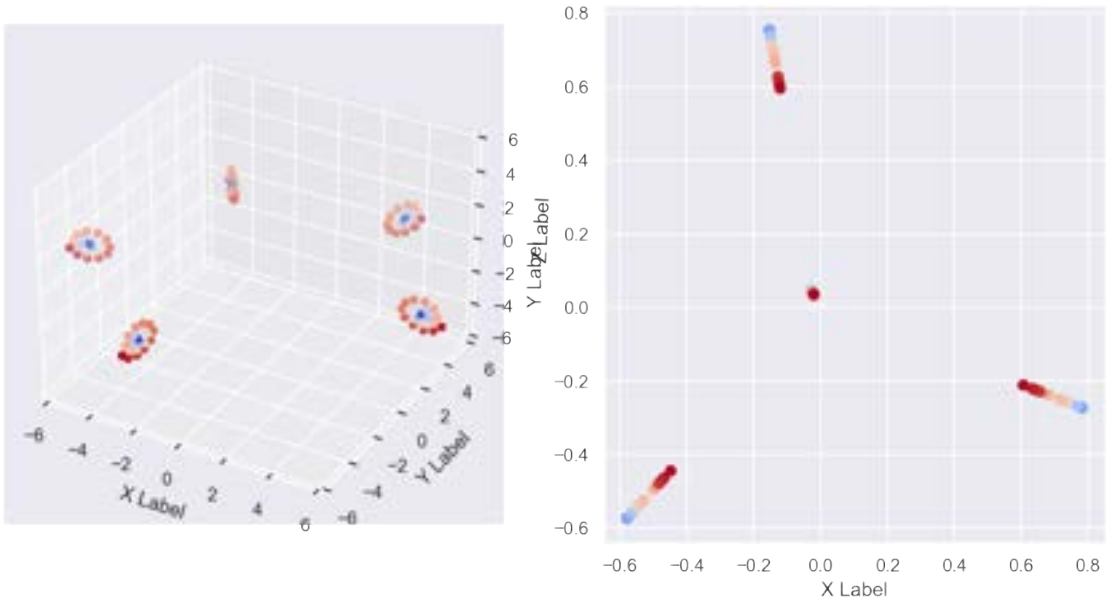
[ ]:
```
show_KernelPCA(spiral_density=3, spiral_steps = 50, width=7, angle=0.5,␣
↪roll_density=5)
```



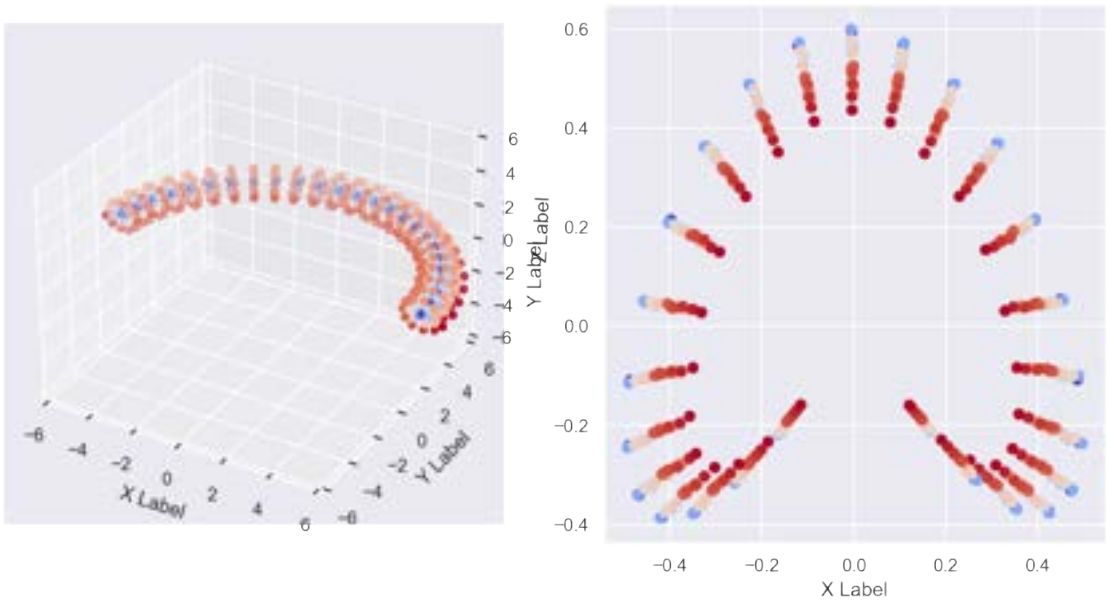
[ ]:
```
show_KernelPCA(spiral_density=3, spiral_steps = 20, width=20, angle=0.5,␣
↪roll_density=5)
```



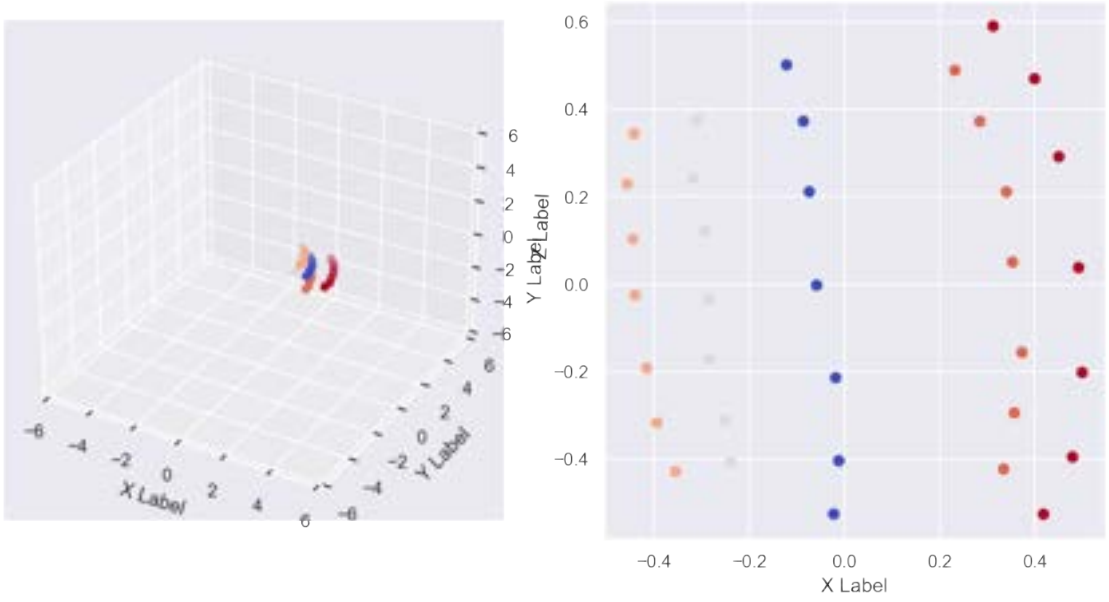
[ ]:
```
show_KernelPCA(spiral_density=3, spiral_steps = 20, width=7, angle=0.9,␣
↪roll_density=5)
```



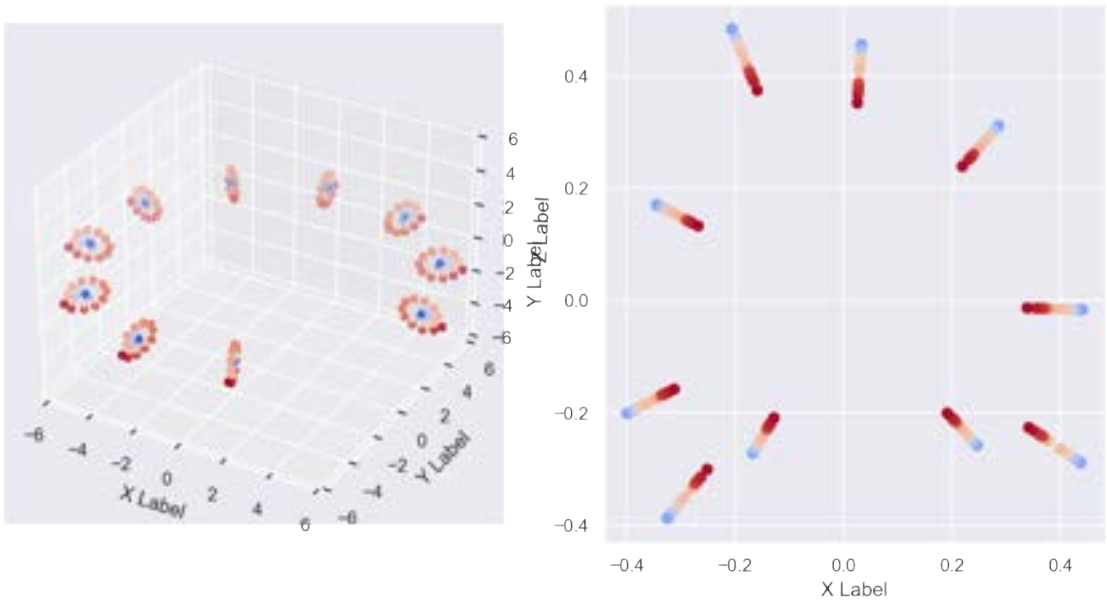
[ ]:
```
show_KernelPCA(spiral_density=3, spiral_steps = 20, width=7, angle=0.5,␣
↪roll_density=25)
```



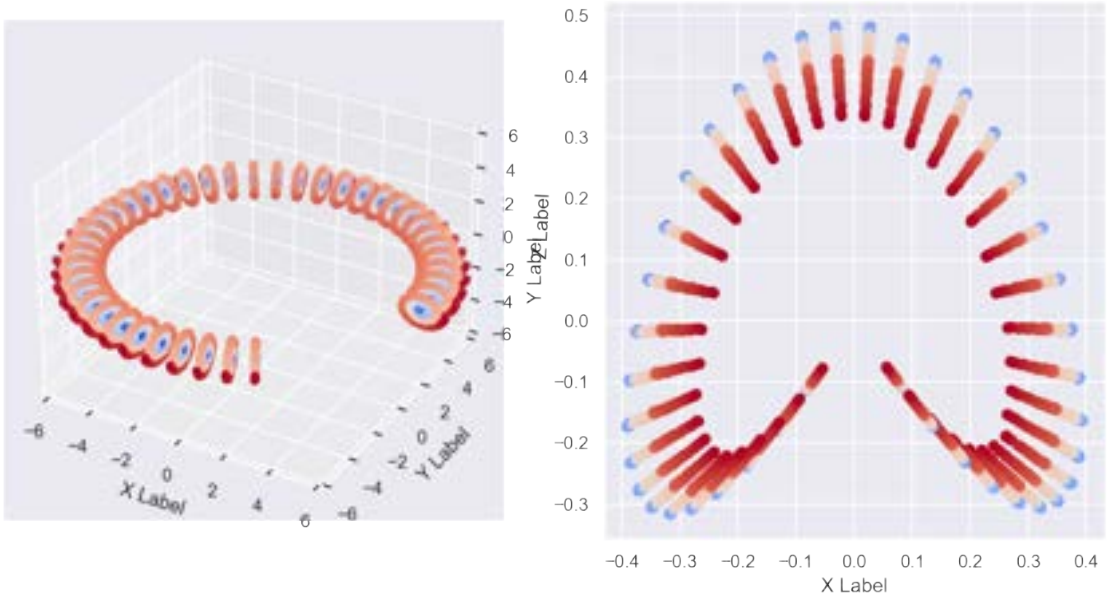
[ ]:
```
show_KernelPCA(spiral_density=10, spiral_steps = 5, width=2, angle=0.15,␣
↪roll_density=7)
```



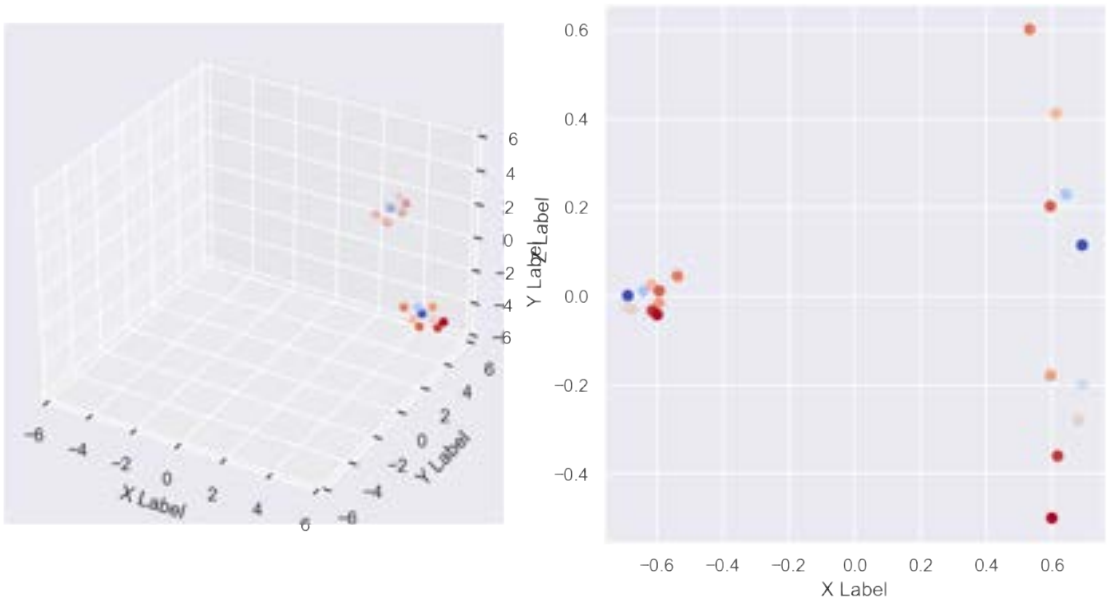
[ ]:
```
show_KernelPCA(spiral_density=3, spiral_steps = 20, width=7, angle=0.9,␣
↪roll_density=10)
```



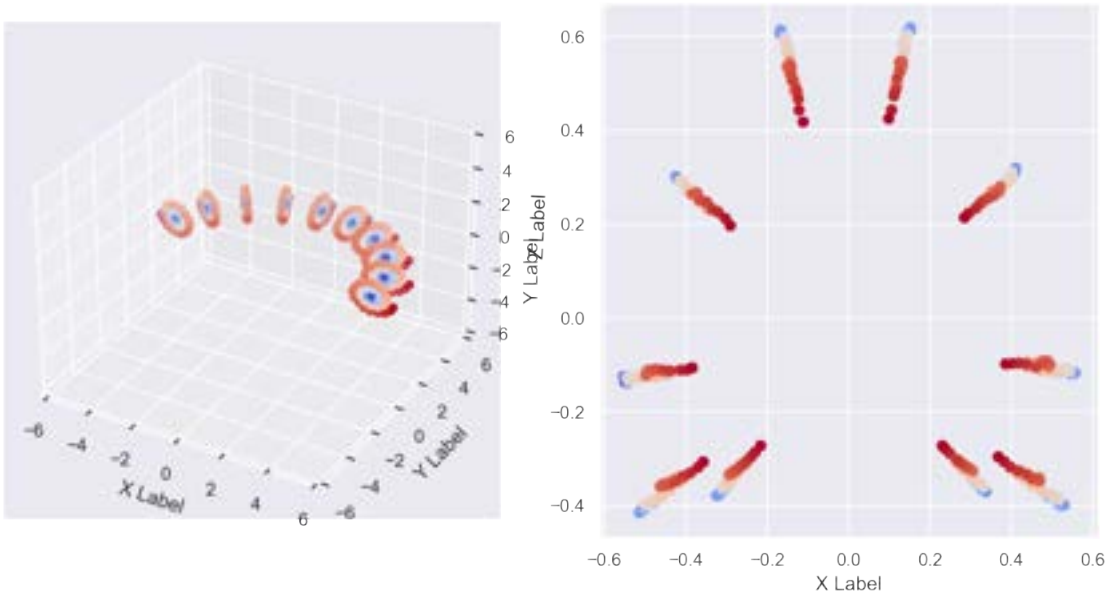
[ ]:
```
show_KernelPCA(spiral_density=3, spiral_steps = 60, width=7, angle=0.85,␣
↪roll_density=40)
```



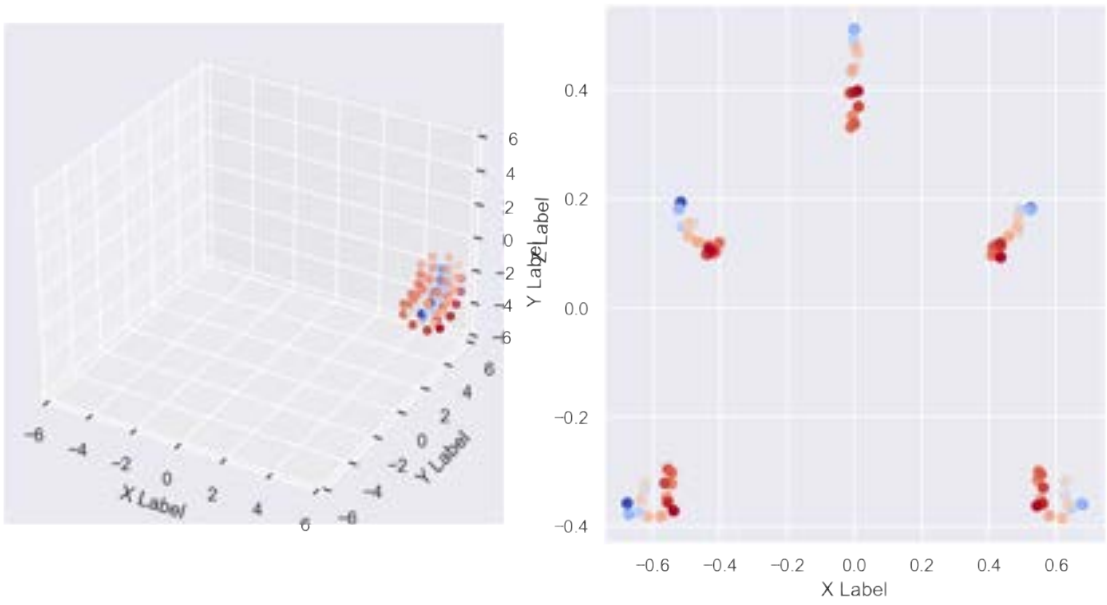
[ ]:
```
show_KernelPCA(spiral_density=20, spiral_steps = 10, width=7, angle=0.4,␣
↪roll_density=2)
```



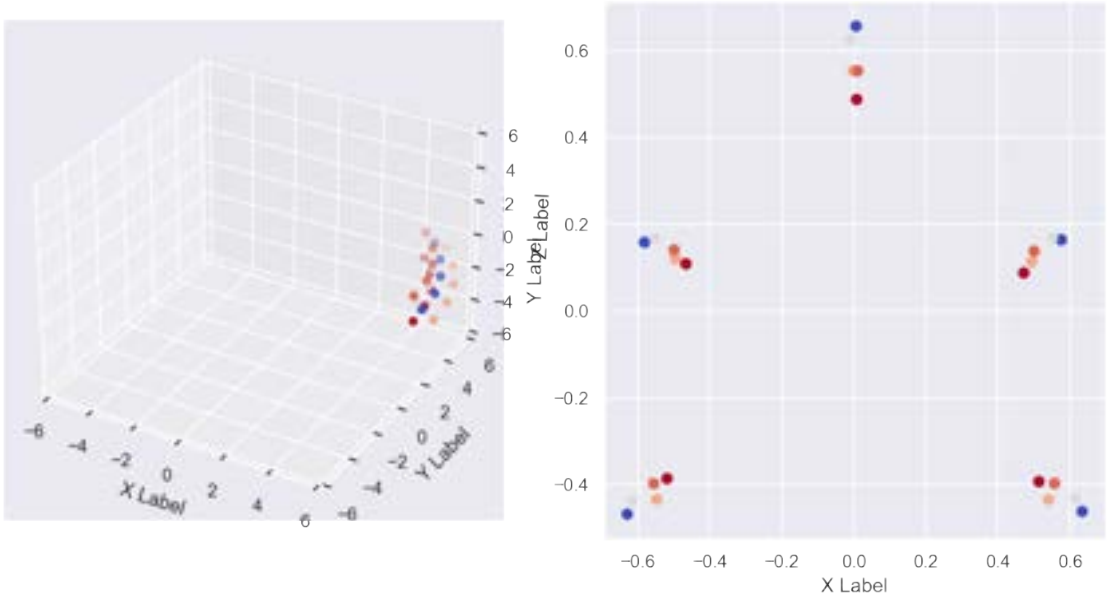
[ ]:
```
show_KernelPCA(spiral_density=3, spiral_steps = 30, width=5, angle=0.5,␣
↪roll_density=10)
```



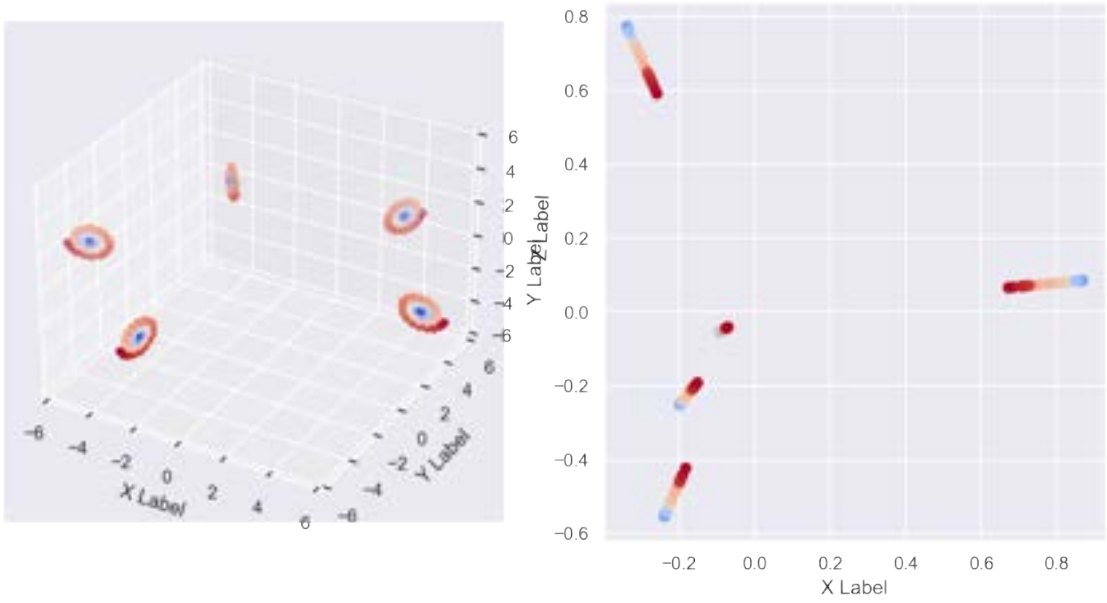
[ ]:
```
show_KernelPCA(spiral_density=3, spiral_steps = 15, width=7, angle=0.1,␣
↪roll_density=5)
```



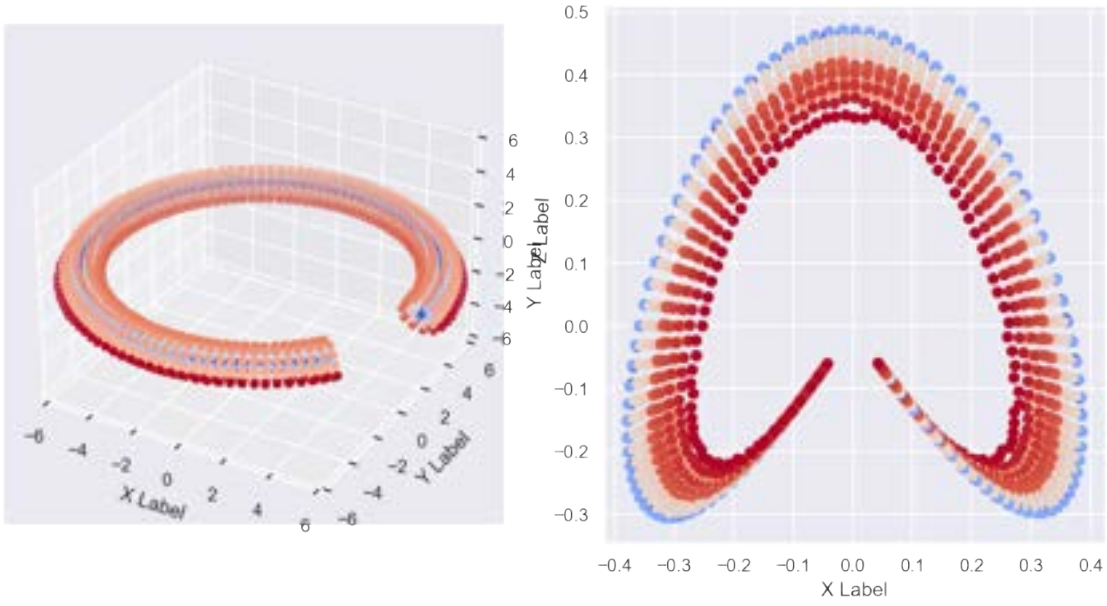
[ ]:
```
show_KernelPCA(spiral_density=3, spiral_steps = 5, width=7, angle=0.15,␣
↪roll_density=5)
```



[ ]:
```
show_KernelPCA(spiral_density=3, spiral_steps = 30, width=7, angle=0.9,␣
↪roll_density=5)
```



[ ]:
```
show_KernelPCA(spiral_density=3, spiral_steps = 20, width=7, angle=0.9,␣
↪roll_density=100)
```

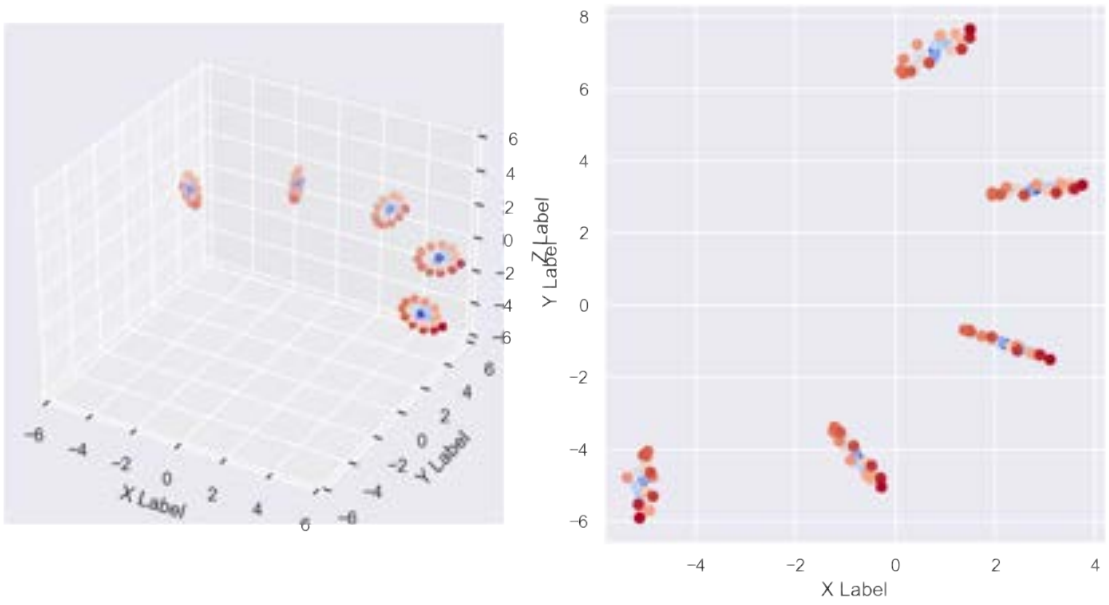


MDE implementation

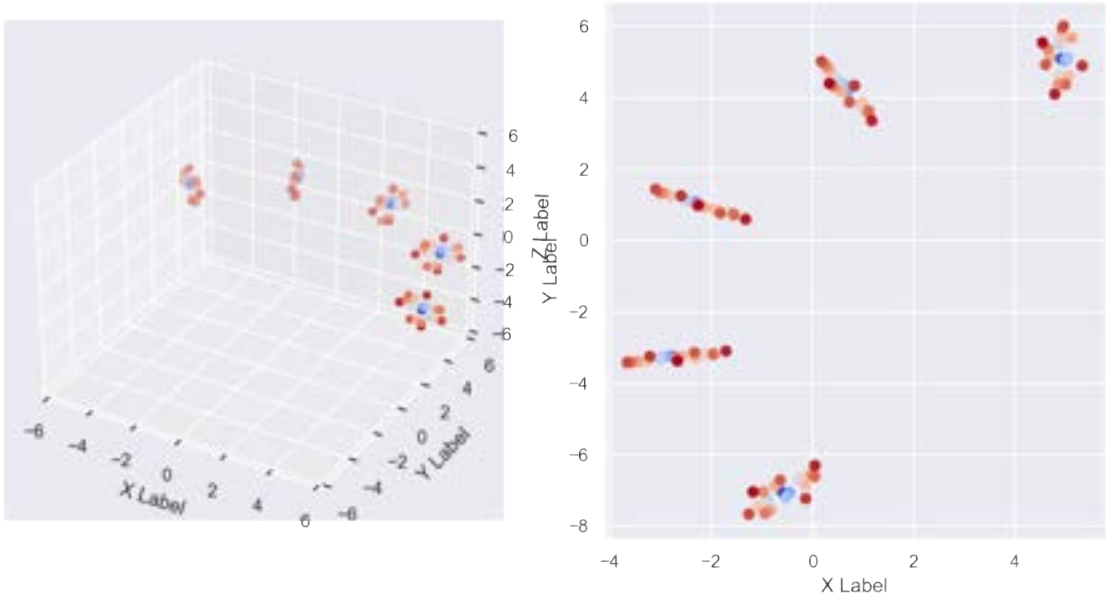
[ ]:
```
def show_MDE(spiral_density=3, spiral_steps = 20, width=7, angle=0.5,␣
↪roll_density=5):
```

```
    fig = plt.figure()
    x,y,z,d=roll_generator(to_array=False, spiral_density=spiral_density,
                           spiral_steps = spiral_steps, width=width,
                           angle=angle, density=roll_density)
    from mpl_toolkits.mplot3d import Axes3D

    # fig = plt.figure()
    ax = fig.add_subplot(121, projection='3d')
    ax.scatter(x, y, z, c=d, cmap=plt.cm.coolwarm)
    ax.set_xlabel('X Label')
    ax.set_ylabel('Y Label')
    ax.set_zlabel('Z Label')
    ax.set_xlim([-6, 6])
    ax.set_ylim([-6, 6])
    ax.set_zlim([-6, 6])

    ax2 = fig.add_subplot(122)
    points, d=roll_generator(to_array=True, spiral_density=spiral_density,
                           spiral_steps = spiral_steps, width=width,
                           angle=angle, density=roll_density)

    from sklearn.manifold import MDS
    embedding = MDS(n_components=2)
    points_transformed = embedding.fit_transform(points)
    points_transformed.shape

    points_transformed_t = points_transformed.T
    points_transformed_t.shape

    ax2.scatter(points_transformed_t[0], points_transformed_t[1], c=d, cmap=plt.
 ↪cm.coolwarm)
    ax2.set_xlabel('X Label')
    ax2.set_ylabel('Y Label')

    plt.rcParams["figure.figsize"] = [12, 6]

    extent = ax2.get_window_extent().transformed(fig.dpi_scale_trans.inverted())
    fig.
 ↪savefig(f'mde_spst{spiral_steps}_roldens{roll_density}_w{width}_a{angle}_sd{spiral_density}
 ↪png', bbox_inches=extent)
```

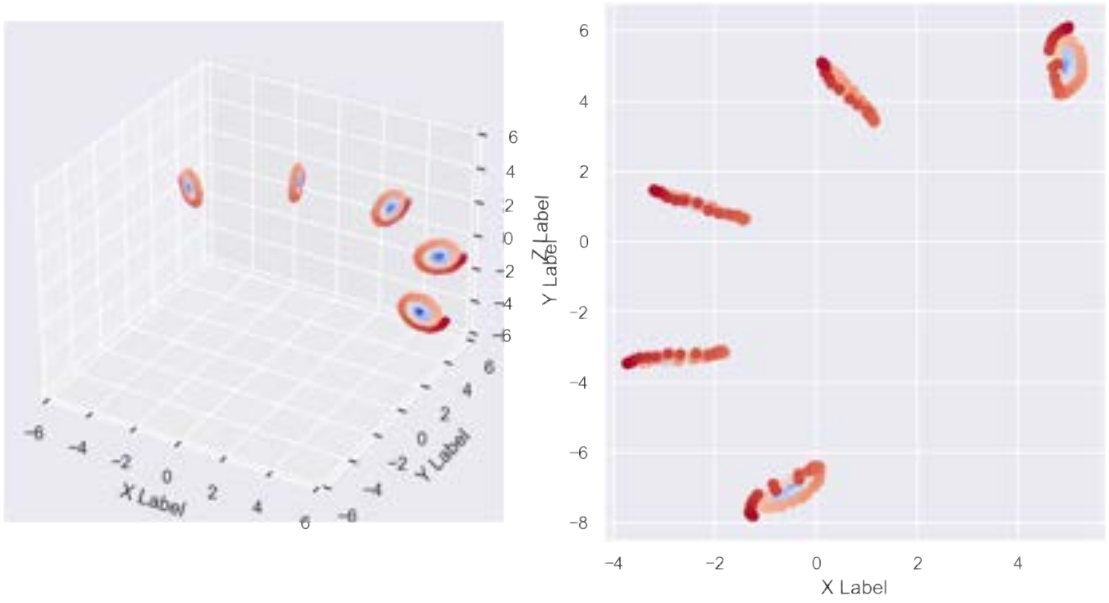
[ ]:
```
# Default params
show_MDE(spiral_density=3, spiral_steps = 20, width=7, angle=0.5,␣
 ↪roll_density=5)
```



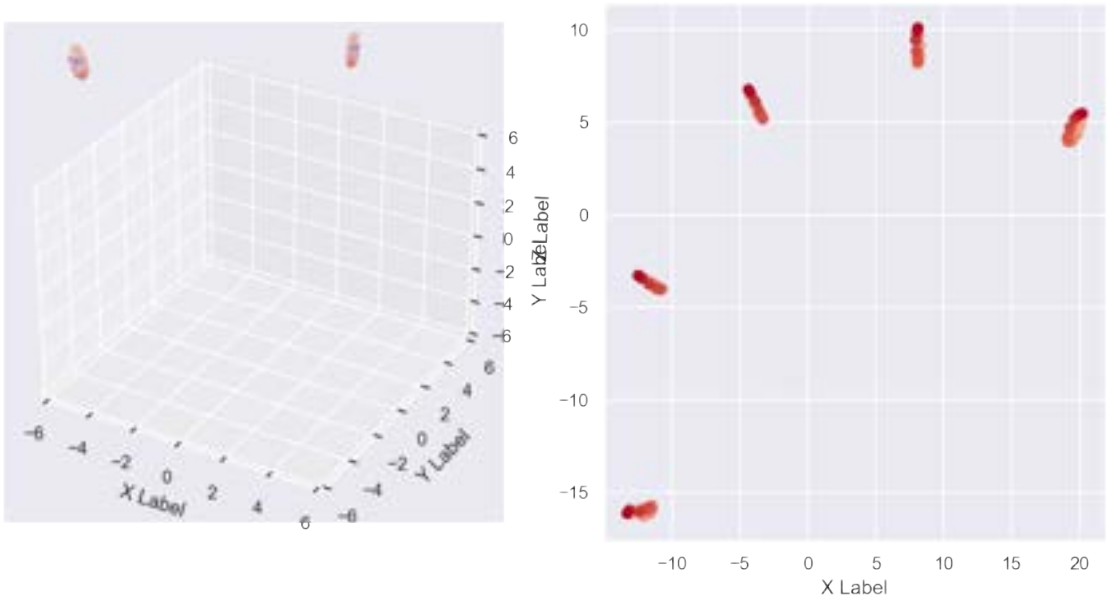
[ ]:
```
show_MDE(spiral_density=10, spiral_steps = 20, width=7, angle=0.5,␣
↪roll_density=5)
```



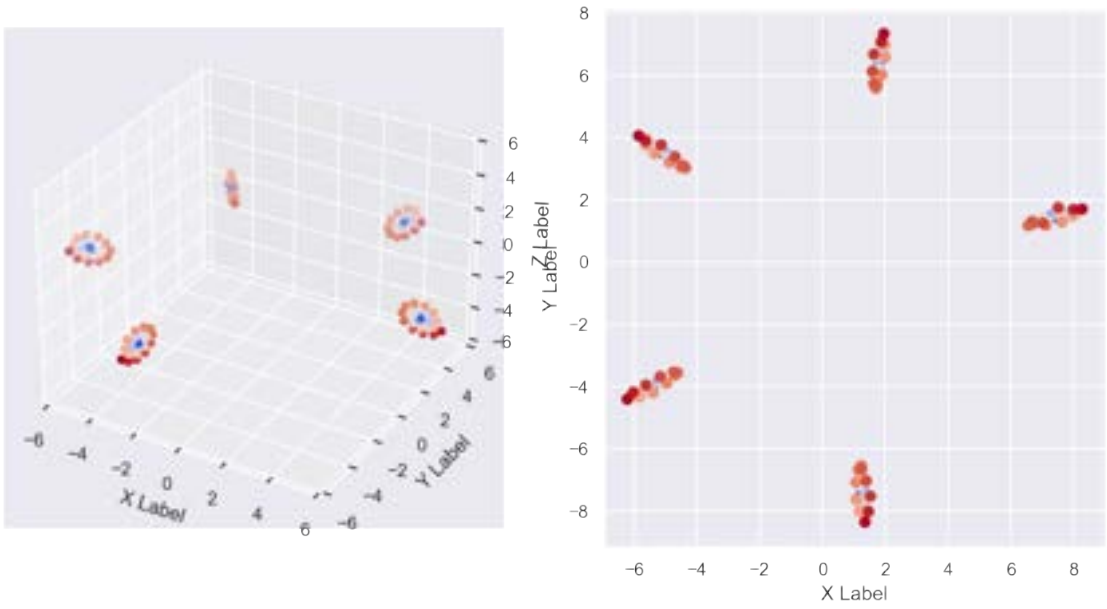
[ ]:
```
show_MDE(spiral_density=3, spiral_steps = 50, width=7, angle=0.5,␣
↪roll_density=5)
```



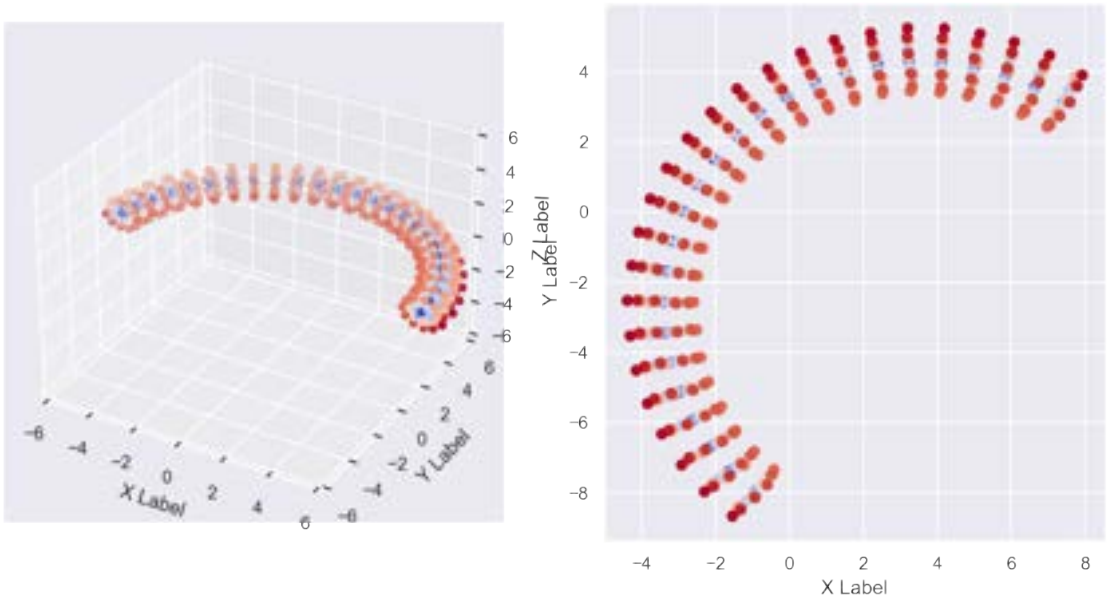
[ ]:
```
show_MDE(spiral_density=3, spiral_steps = 20, width=20, angle=0.5,␣
↪roll_density=5)
```



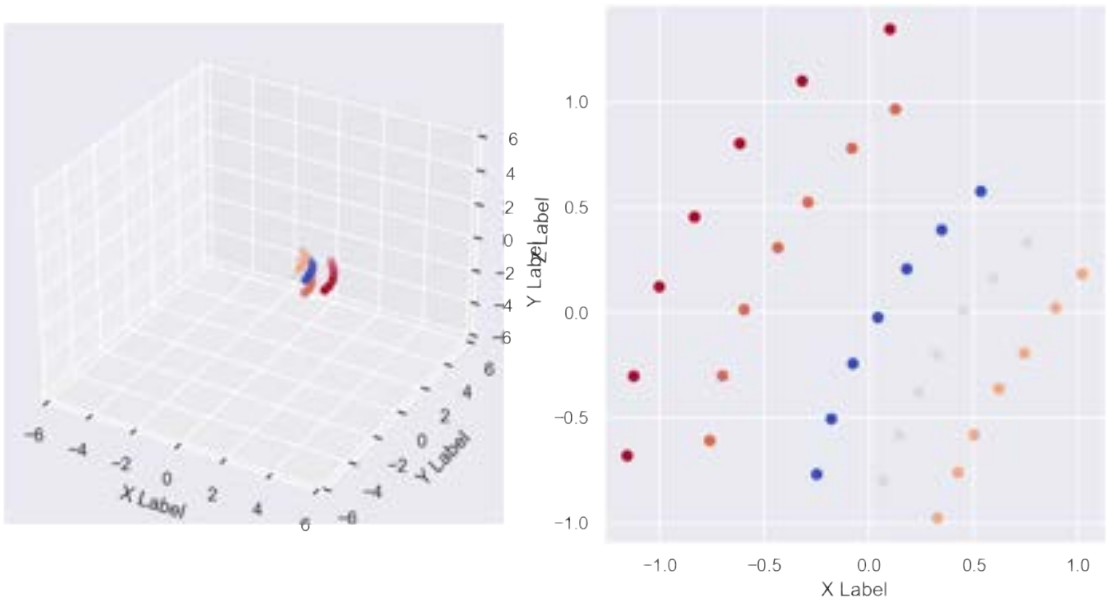
[ ]:
```
show_MDE(spiral_density=3, spiral_steps = 20, width=7, angle=0.9,␣
↪roll_density=5)
```



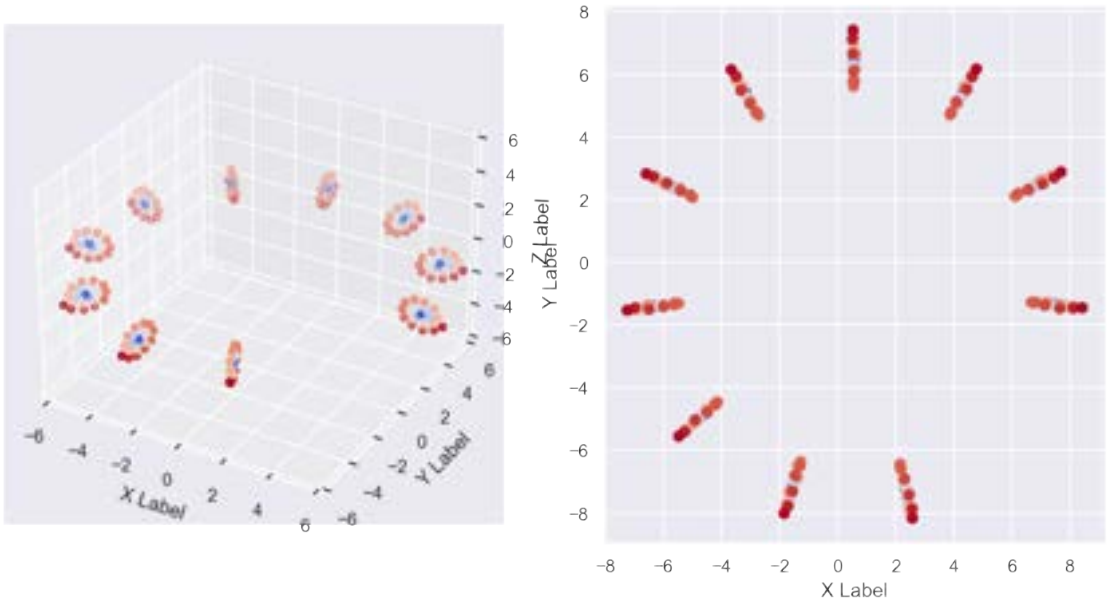
[ ]:
```
show_MDE(spiral_density=3, spiral_steps = 20, width=7, angle=0.5,␣
↪roll_density=25)
```



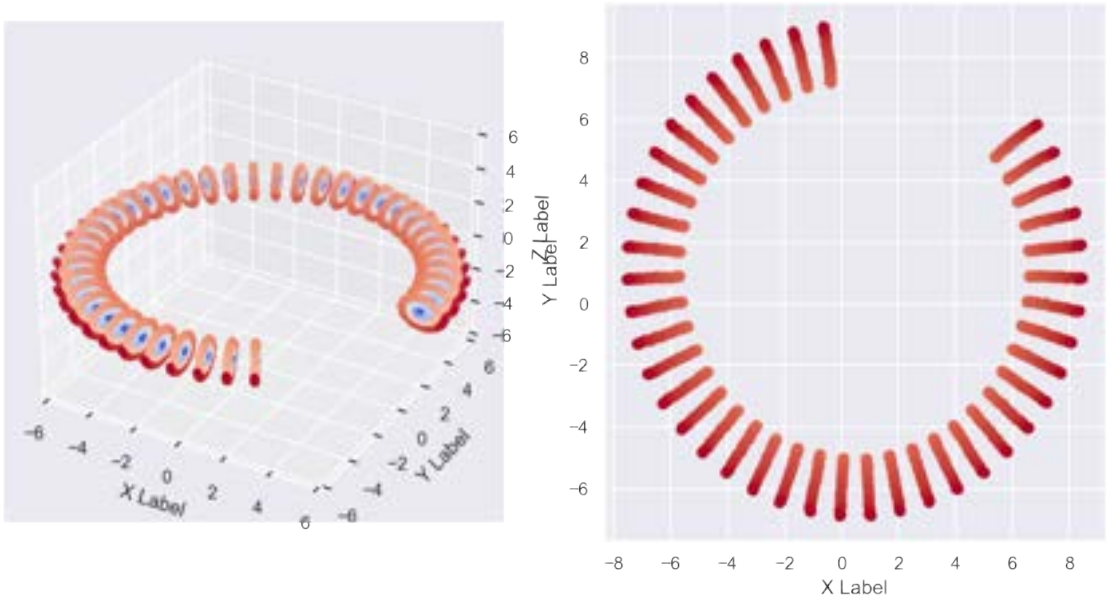
[ ]:
```
show_MDE(spiral_density=10, spiral_steps = 5, width=2, angle=0.15,␣
↪roll_density=7)
```



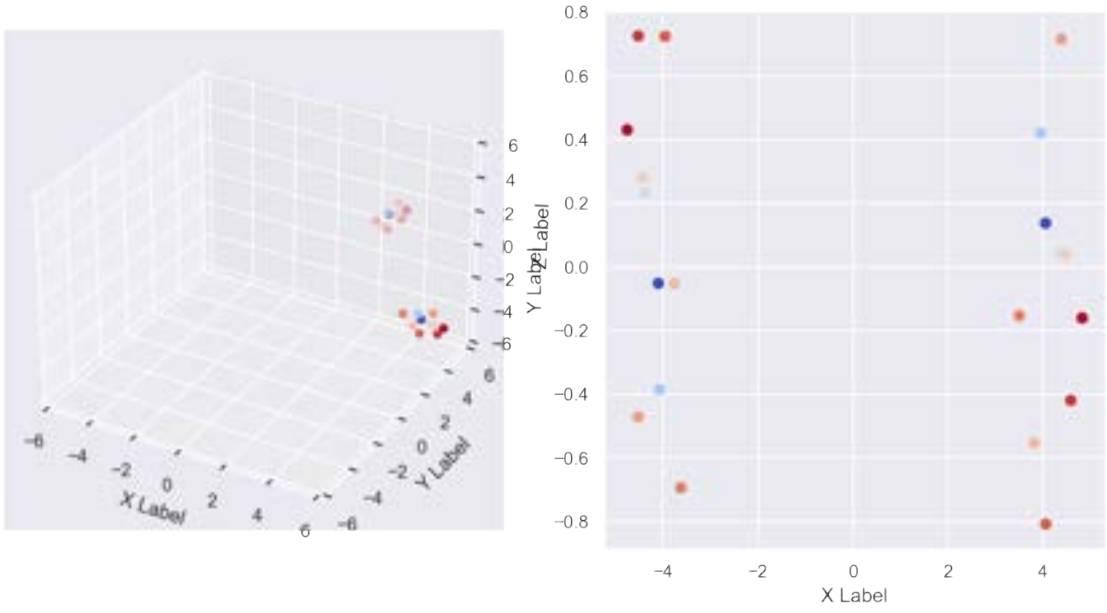
[ ]:
```
show_MDE(spiral_density=3, spiral_steps = 20, width=7, angle=0.9,␣
↪roll_density=10)
```



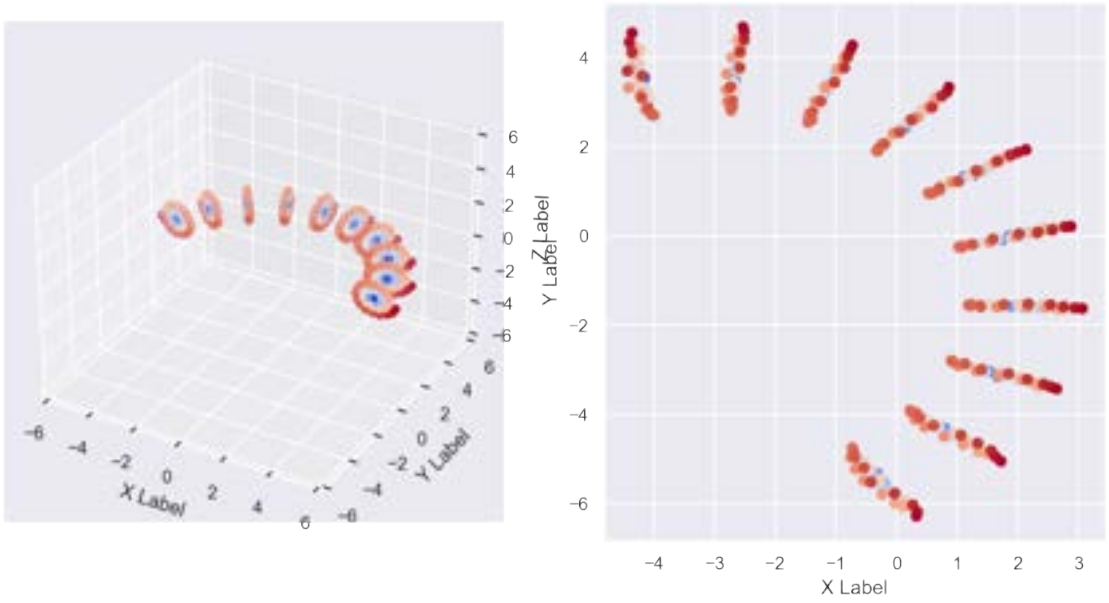
[ ]:
```
show_MDE(spiral_density=3, spiral_steps = 60, width=7, angle=0.85,␣
↪roll_density=40)
```



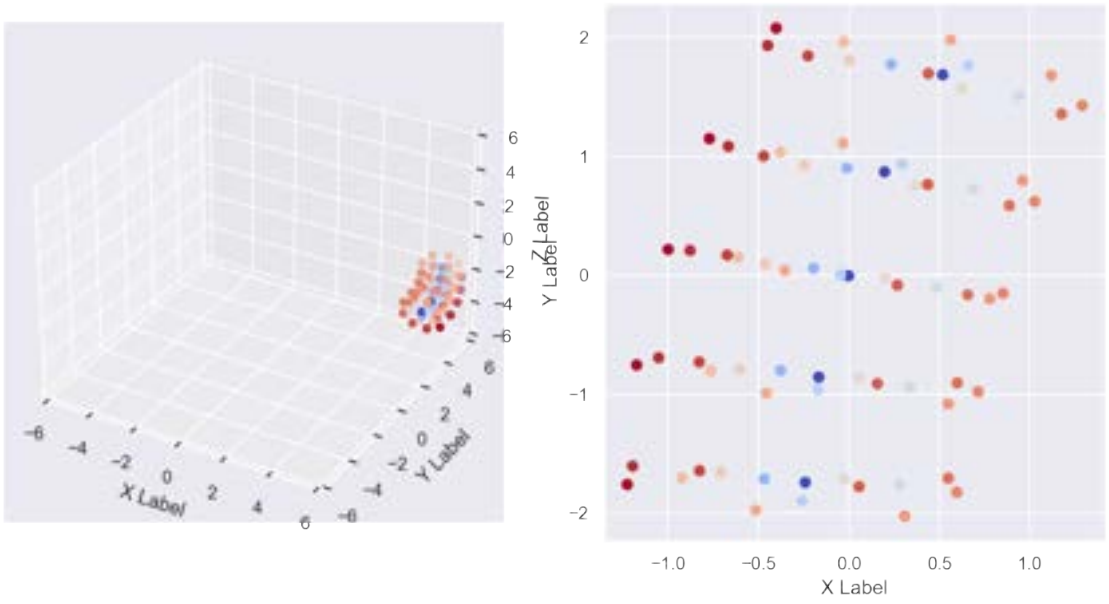
[ ]:
```
show_MDE(spiral_density=20, spiral_steps = 10, width=7, angle=0.4,␣
↪roll_density=2)
```



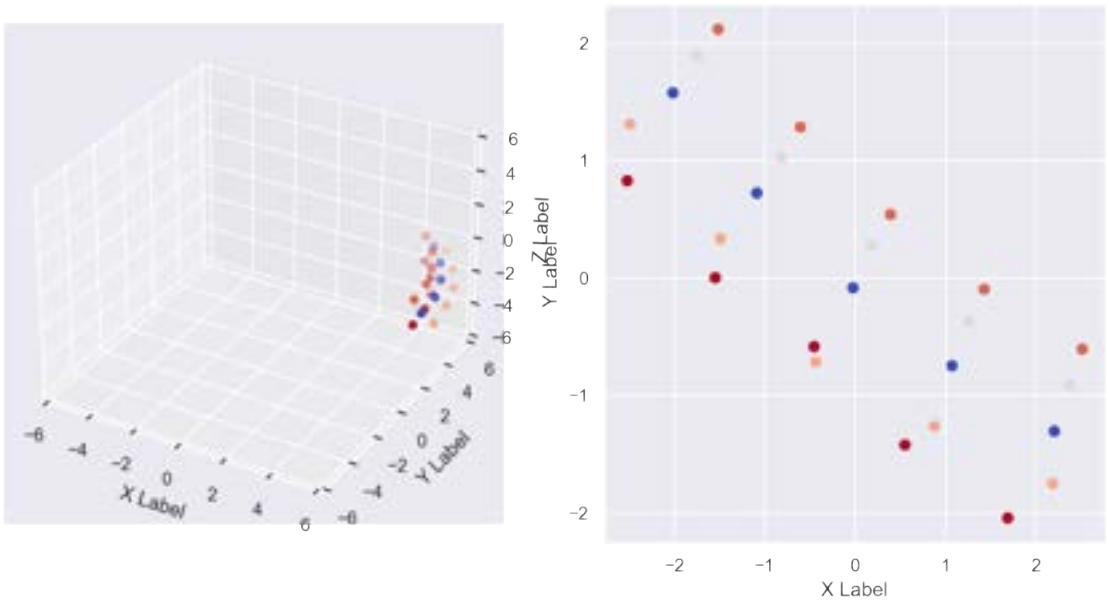
[ ]:
```
show_MDE(spiral_density=3, spiral_steps = 30, width=5, angle=0.5,␣
↪roll_density=10)
```



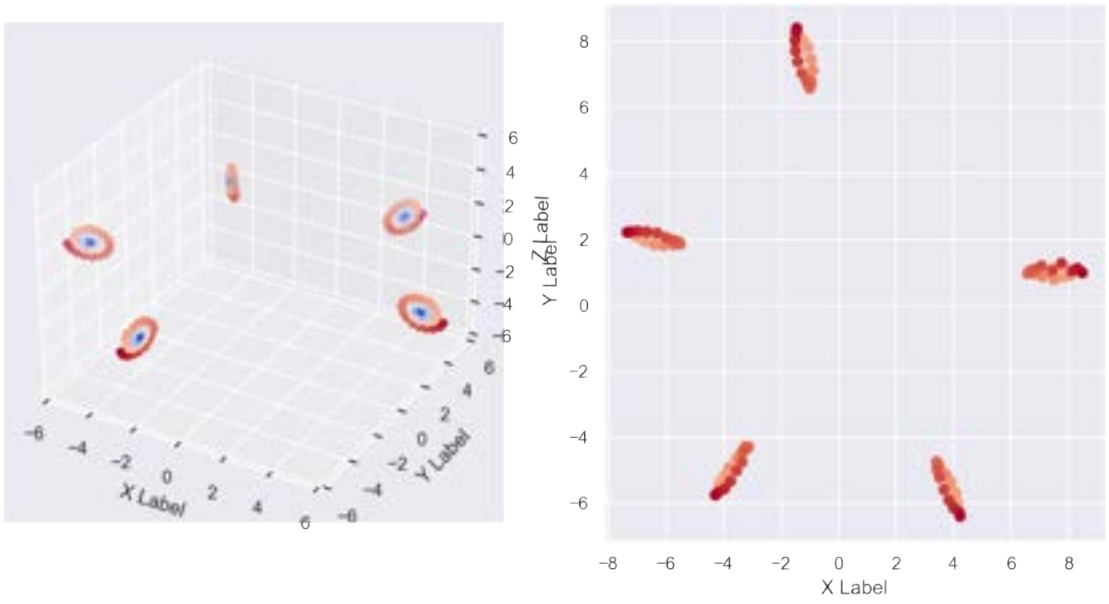
[ ]:
```
show_MDE(spiral_density=3, spiral_steps = 15, width=7, angle=0.1,␣
↪roll_density=5)
```



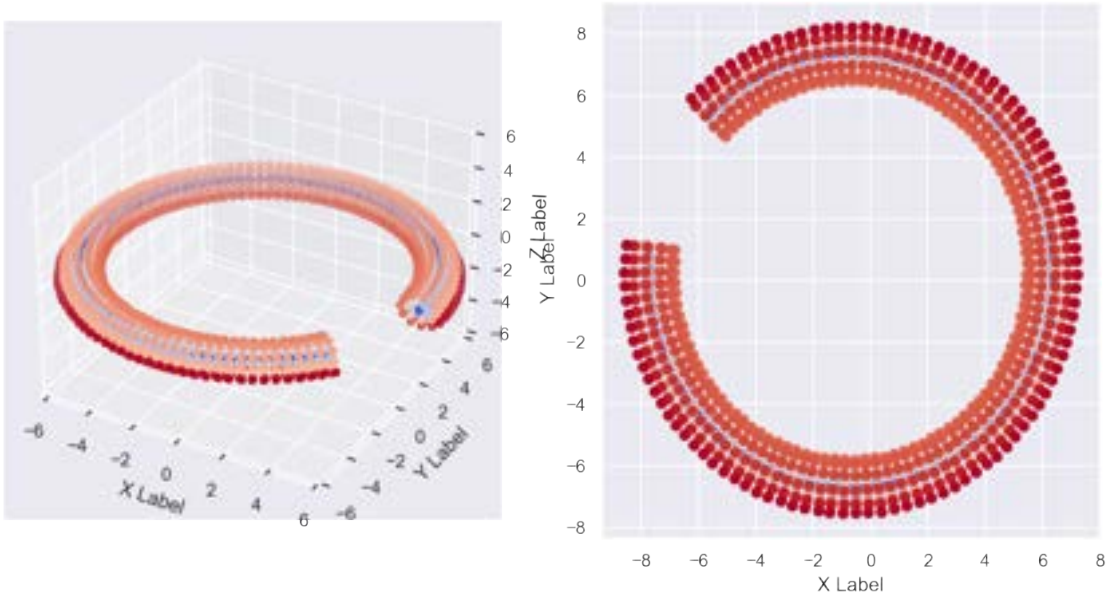
[ ]:
```
show_MDE(spiral_density=3, spiral_steps = 5, width=7, angle=0.15,␣
↪roll_density=5)
```



[ ]:
```
show_MDE(spiral_density=3, spiral_steps = 30, width=7, angle=0.9,␣
↪roll_density=5)
```



[ ]:
```
show_MDE(spiral_density=3, spiral_steps = 20, width=7, angle=0.9,␣
↪roll_density=100)
```

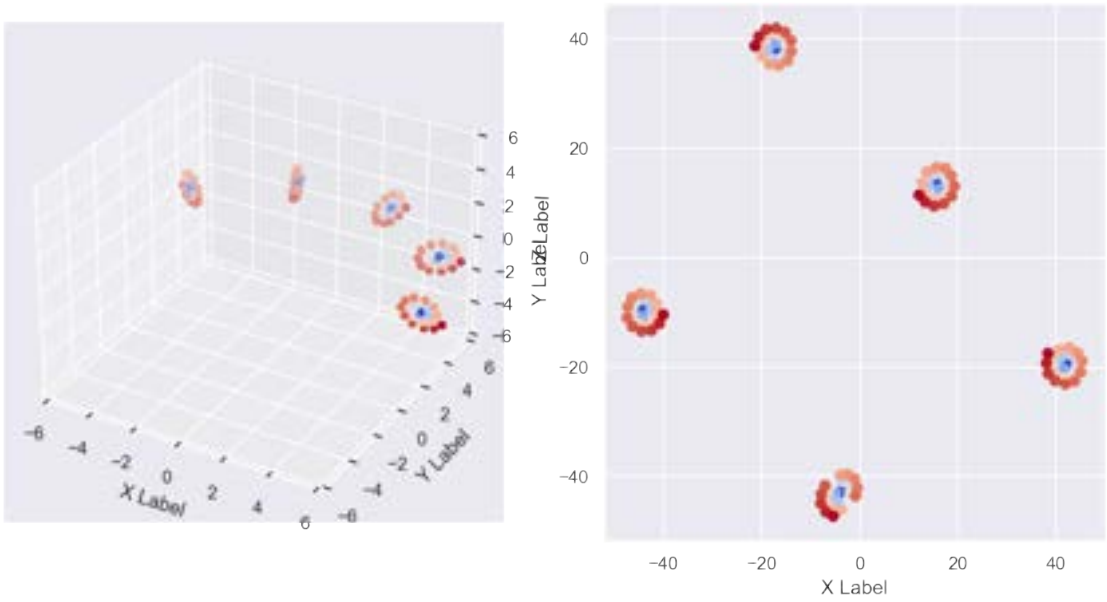


t-SNE implementation

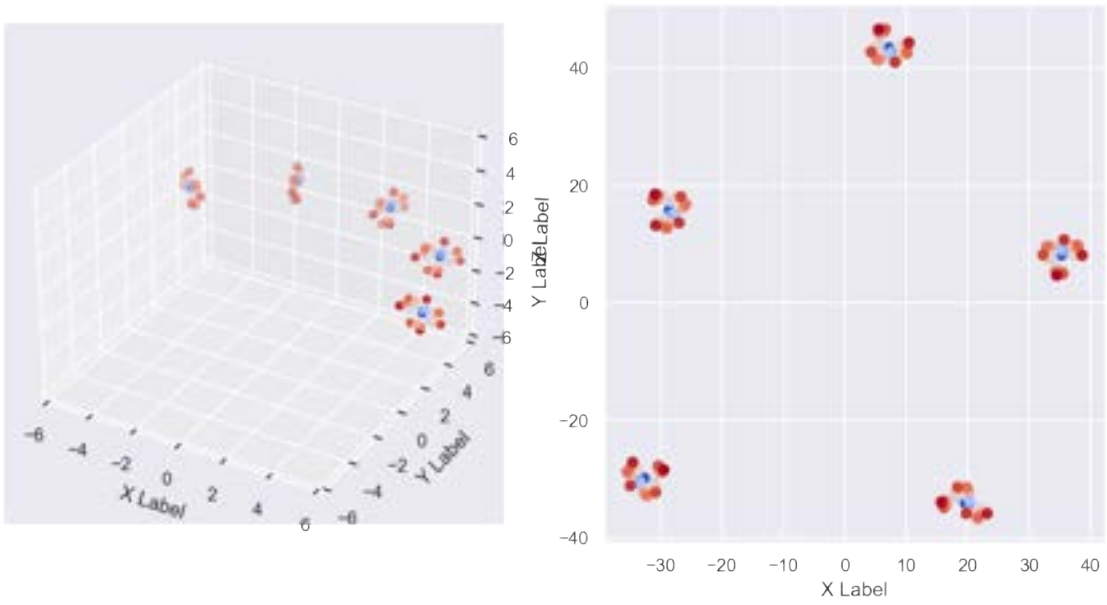
[ ]:
```
def show_tSNE(spiral_density=3, spiral_steps = 20, width=7, angle=0.5,␣
↪roll_density=5, n_neighbors=5,
              metric='euclidean', perplexity=30):
    fig = plt.figure()
    x,y,z,d=roll_generator(to_array=False, spiral_density=spiral_density,
                           spiral_steps = spiral_steps, width=width,
                           angle=angle, density=roll_density)
    from mpl_toolkits.mplot3d import Axes3D

    # fig = plt.figure()
    ax = fig.add_subplot(121, projection='3d')
    ax.scatter(x, y, z, c=d, cmap=plt.cm.coolwarm)
    ax.set_xlabel('X Label')
    ax.set_ylabel('Y Label')
    ax.set_zlabel('Z Label')
    ax.set_xlim([-6, 6])
    ax.set_ylim([-6, 6])
    ax.set_zlim([-6, 6])

    ax2 = fig.add_subplot(122)
    points, d=roll_generator(to_array=True, spiral_density=spiral_density,
                           spiral_steps = spiral_steps, width=width,
                           angle=angle, density=roll_density)

    from sklearn.manifold import TSNE
```

```
    points_transformed = TSNE(n_components=2, metric=metric,␣
↪perplexity=perplexity).fit_transform(points)

    points_transformed_t = points_transformed.T


    ax2.scatter(points_transformed_t[0], points_transformed_t[1], c=d, cmap=plt.
↪cm.coolwarm)
    ax2.set_xlabel('X Label')
    ax2.set_ylabel('Y Label')

    plt.rcParams["figure.figsize"] = [12, 6]

    extent = ax2.get_window_extent().transformed(fig.dpi_scale_trans.inverted())
    fig.
↪savefig(f'tsne_n{n_neighbors}_spst{spiral_steps}_roldens{roll_density}_w{width}_a{angle}_sd
↪png', bbox_inches=extent)
```

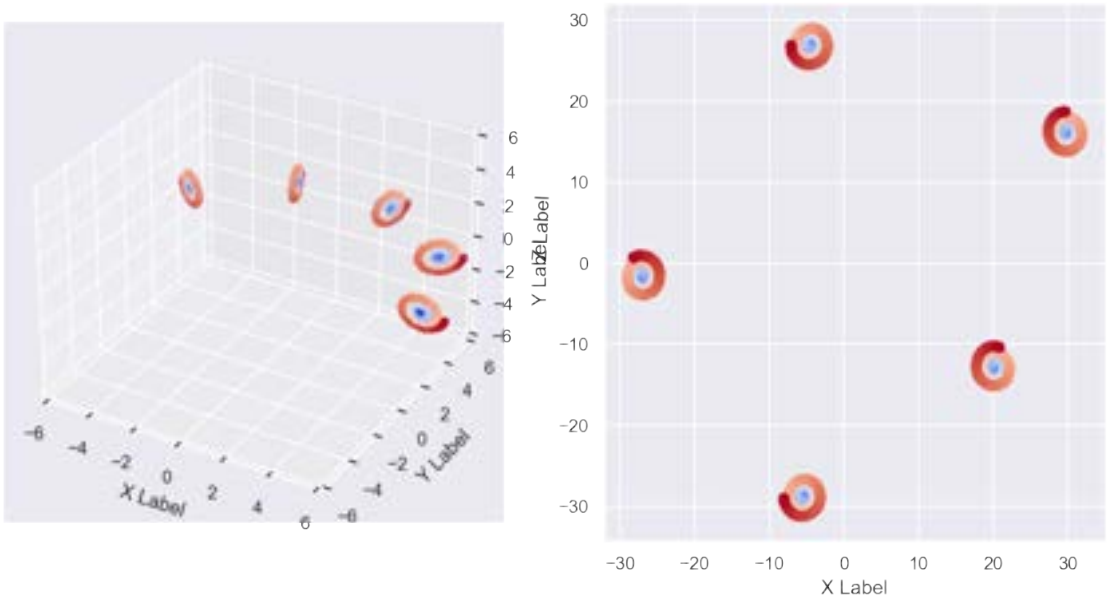
[ ]:
```
# Default params + perplexity = 10
show_tSNE(spiral_density=3, spiral_steps = 20, width=7, angle=0.5,␣
↪roll_density=5, n_neighbors=5, perplexity=10)
```



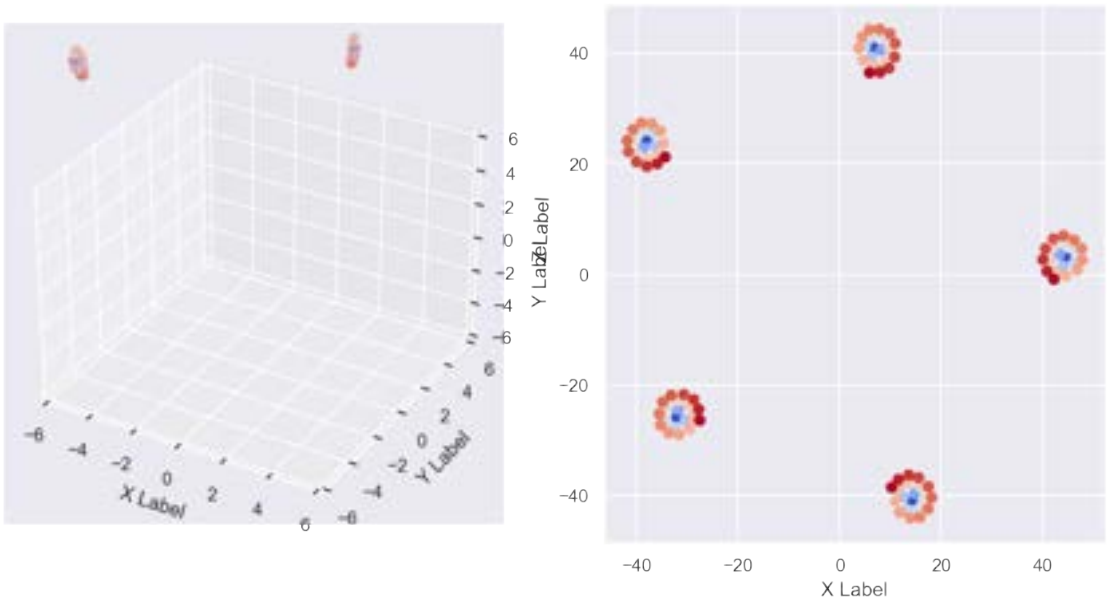
[ ]:
```
show_tSNE(spiral_density=10, spiral_steps = 20, width=7, angle=0.5,␣
↪roll_density=5, n_neighbors=5, perplexity=10)
```



[ ]:
```
show_tSNE(spiral_density=3, spiral_steps = 50, width=7, angle=0.5,␣
↪roll_density=5, n_neighbors=5, perplexity=30)
```

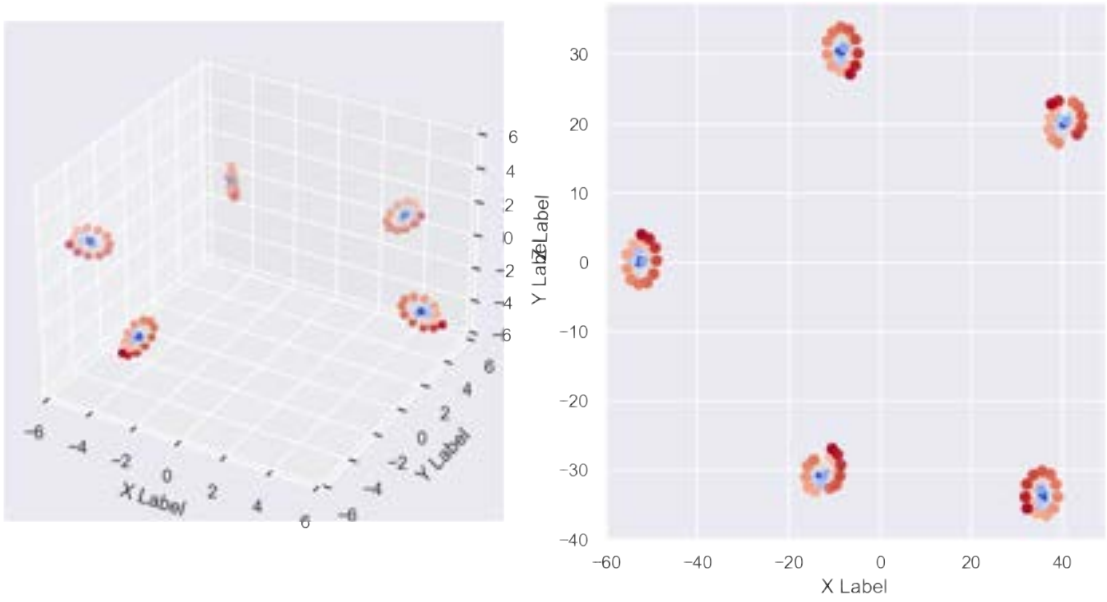


[ ]:
```
show_tSNE(spiral_density=3, spiral_steps = 20, width=20, angle=0.5,␣
↪roll_density=5, n_neighbors=5, perplexity=10)
```

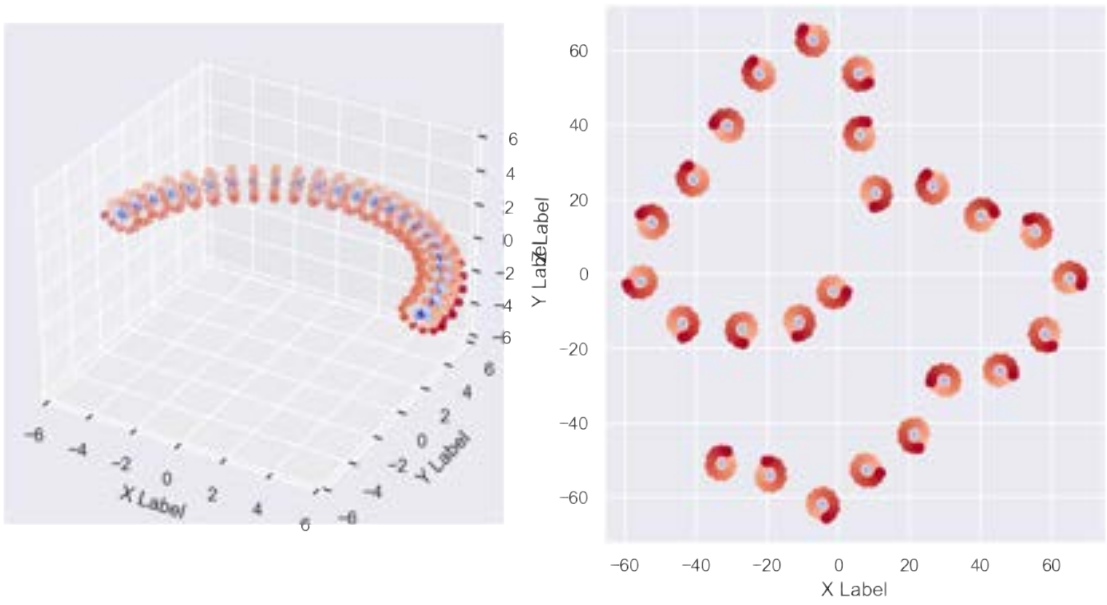


NOTE: IM BARDZIEJ SPARSE TYM LEPSZE MNIEJSZE PERPLEXITY

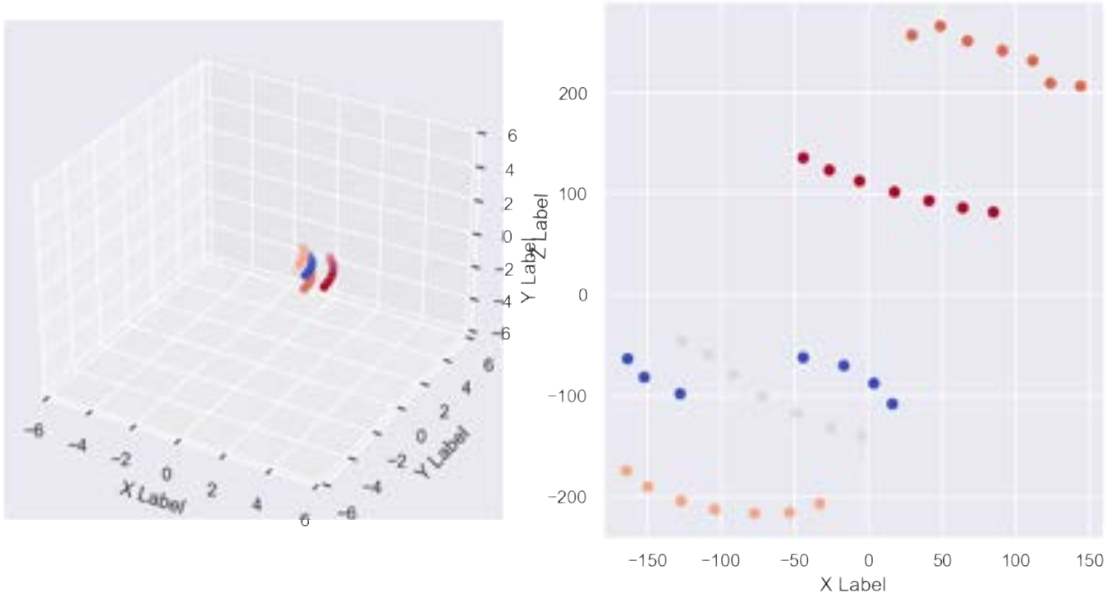
[ ]:
```
show_tSNE(spiral_density=3, spiral_steps = 20, width=7, angle=0.9,␣
↪roll_density=5, n_neighbors=5, perplexity=10)
```



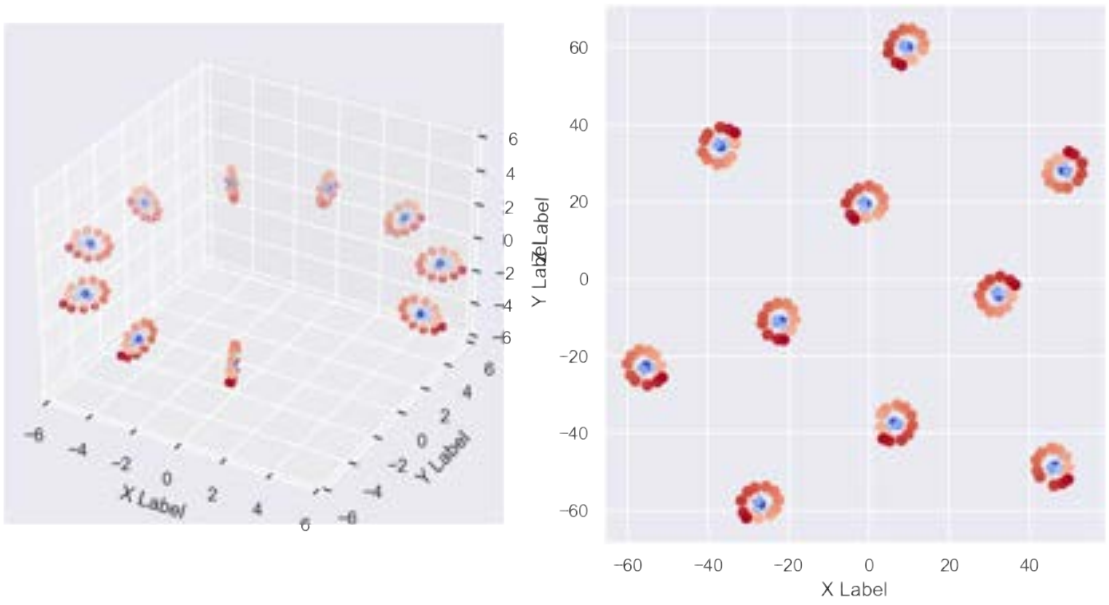
[ ]:
```
show_tSNE(spiral_density=3, spiral_steps = 20, width=7, angle=0.5,␣
↪roll_density=25, n_neighbors=5, perplexity=10)
```



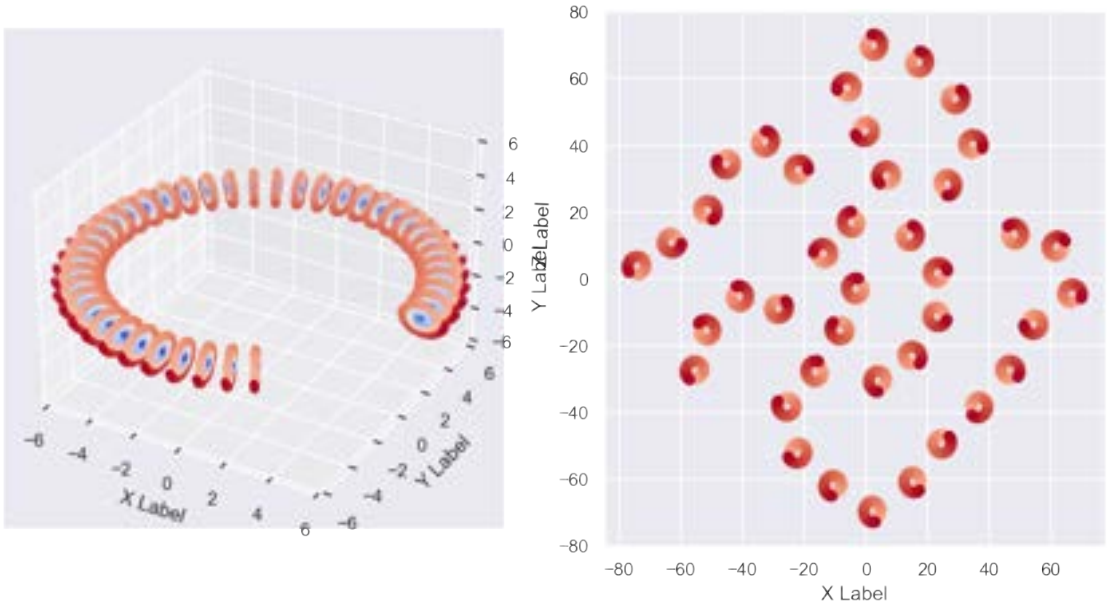
[ ]:
```
show_tSNE(spiral_density=10, spiral_steps = 5, width=2, angle=0.15,␣
↪roll_density=7, n_neighbors=2, perplexity=5)
```



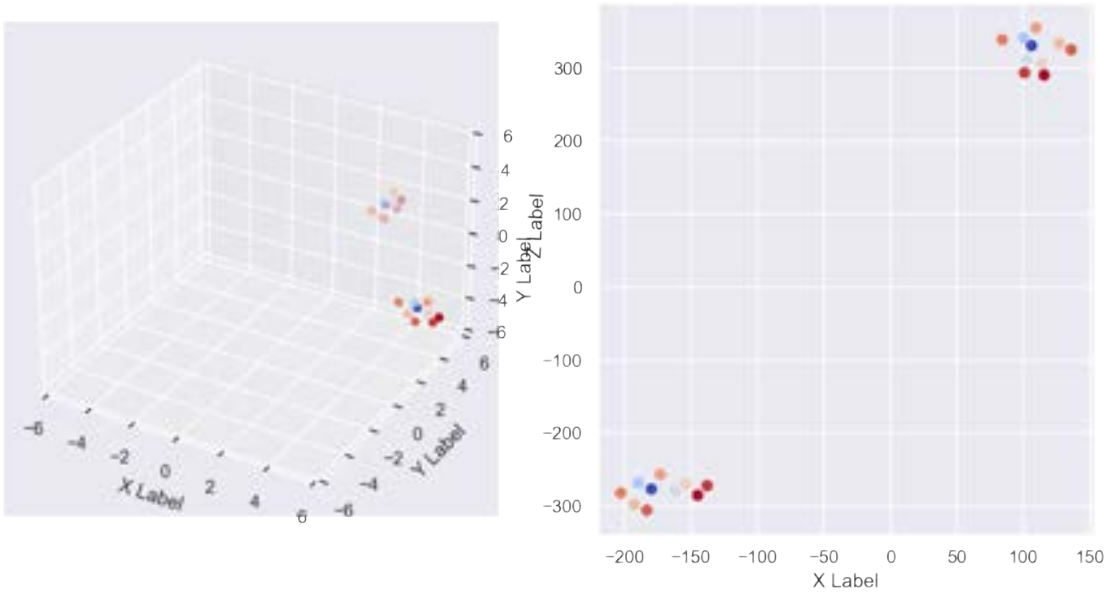
[ ]:
```
show_tSNE(spiral_density=3, spiral_steps = 20, width=7, angle=0.9,␣
↪roll_density=10, n_neighbors=5, perplexity=5)
```



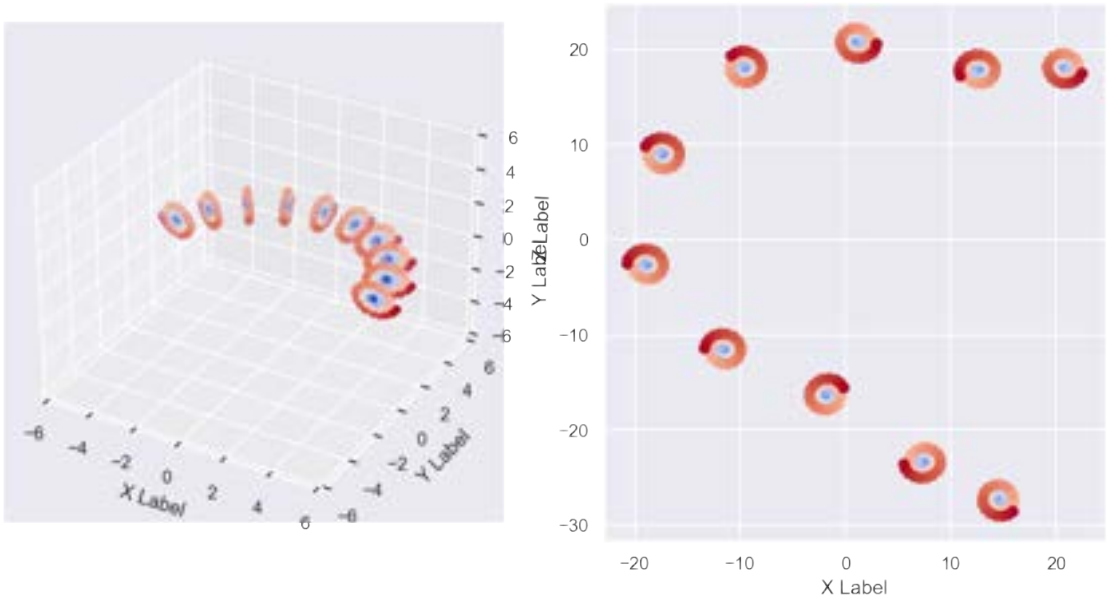
[ ]:
```
show_tSNE(spiral_density=3, spiral_steps = 60, width=7, angle=0.85,␣
↪roll_density=40, n_neighbors=5, perplexity=30)
```



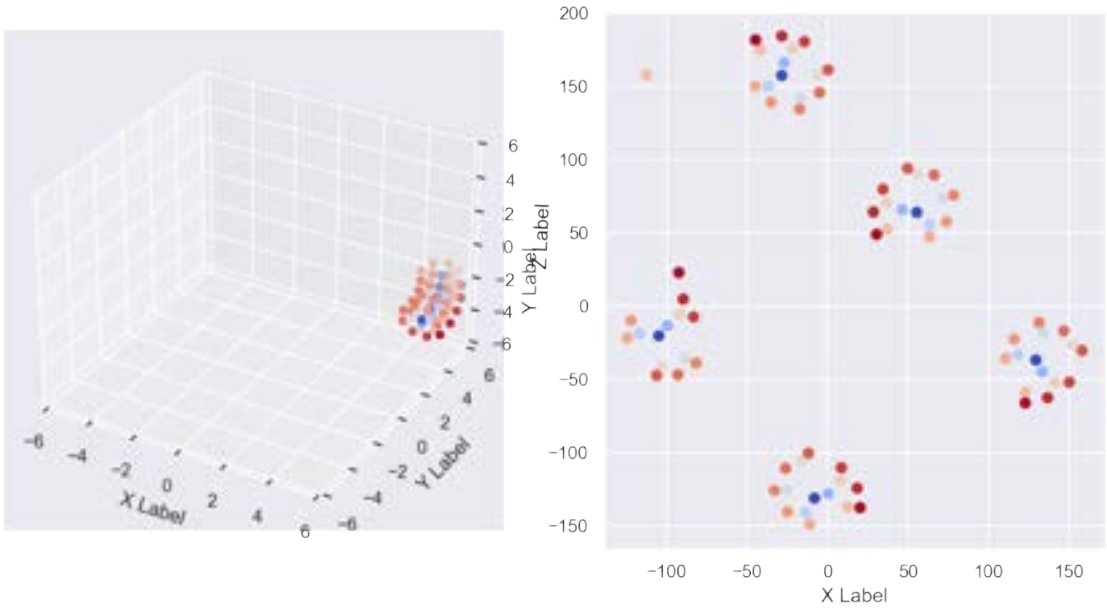
[ ]:
```
show_tSNE(spiral_density=20, spiral_steps = 10, width=7, angle=0.4,␣
↪roll_density=2, n_neighbors=2, perplexity=5)
```



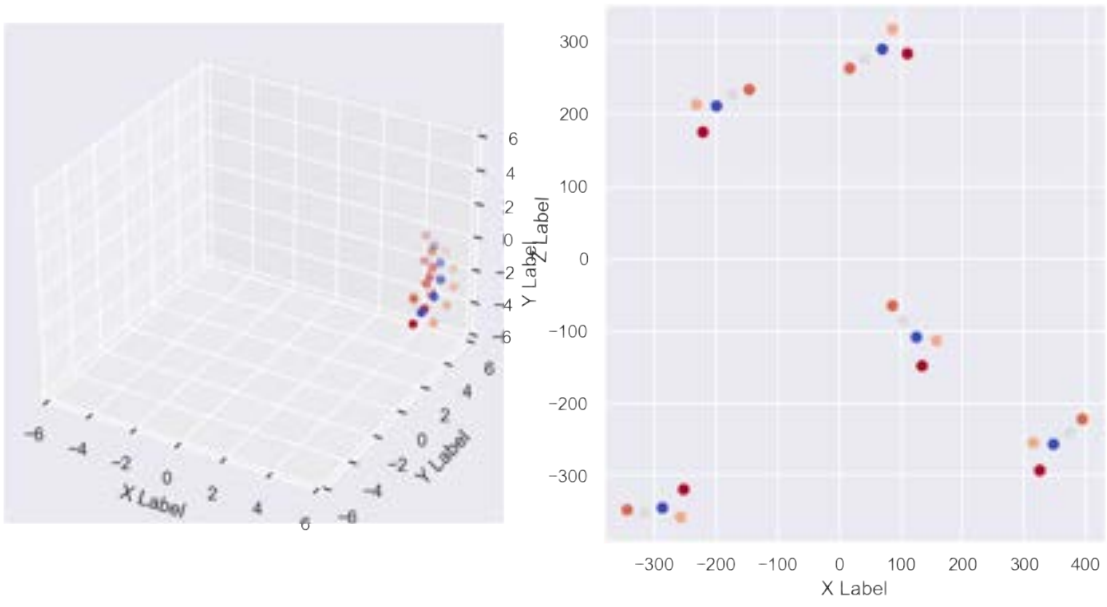
[ ]:
```
show_tSNE(spiral_density=3, spiral_steps = 30, width=5, angle=0.5,␣
↪roll_density=10, n_neighbors=5, perplexity=30)
```



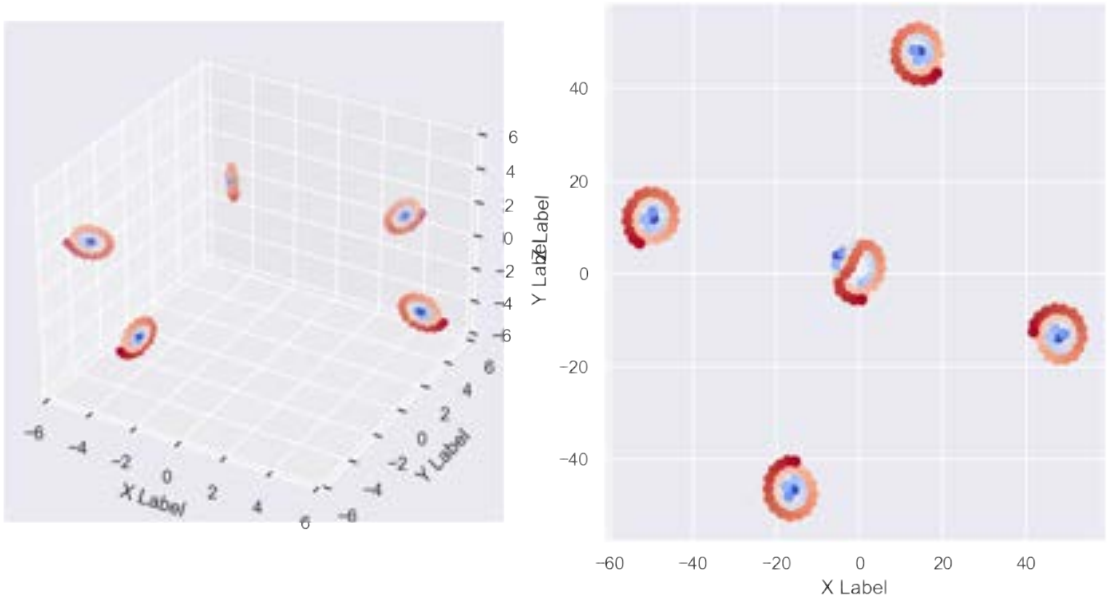
[ ]:
```
show_tSNE(spiral_density=3, spiral_steps = 15, width=7, angle=0.1,␣
↪roll_density=5, n_neighbors=5, perplexity=5)
```



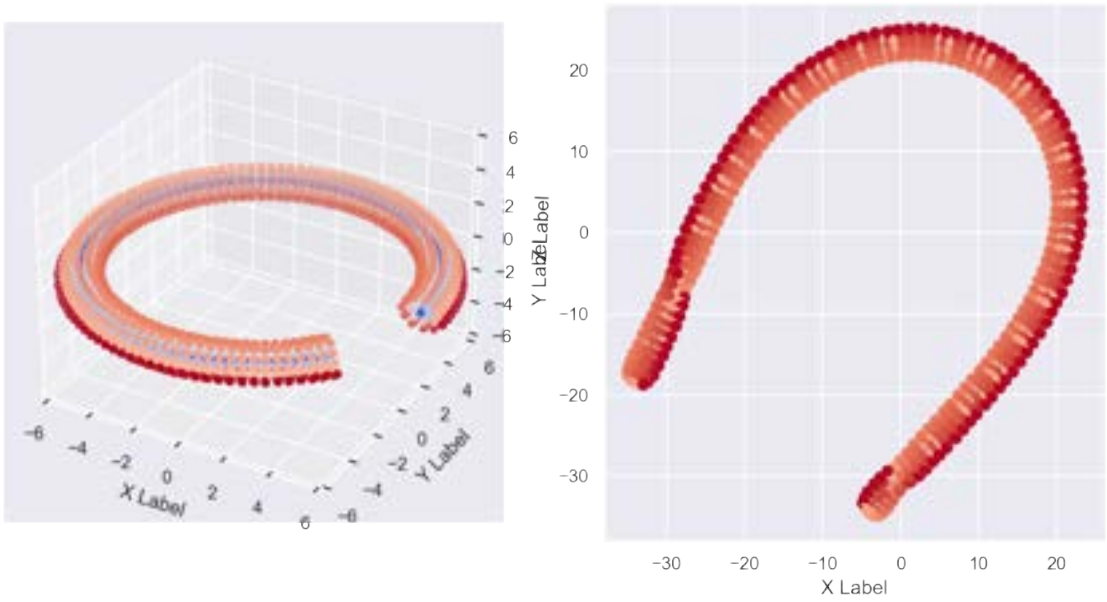
[ ]:
```
show_tSNE(spiral_density=3, spiral_steps = 5, width=7, angle=0.15,␣
↪roll_density=5, n_neighbors=2, perplexity=3)
```



[ ]:
```
show_tSNE(spiral_density=3, spiral_steps = 30, width=7, angle=0.9,␣
↪roll_density=5, n_neighbors=5, perplexity=10)
```



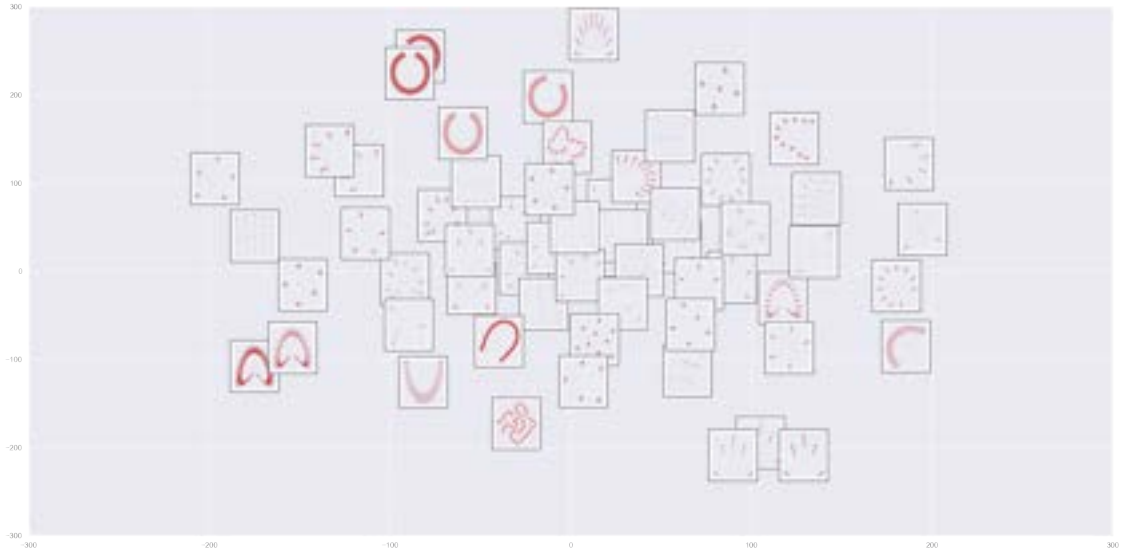
[ ]:
```
show_tSNE(spiral_density=3, spiral_steps = 20, width=7, angle=0.9,␣
↪roll_density=100, n_neighbors=10, perplexity=80)
```



2.1.1 In this way, you should get a set of images showing the various available projections of the original roulade to the 2D plane. Treat these images as elements of a new data set, we treat image pixels as a high-dimensional vector. Use t-SNE to project the newly created set of 2d projection thumbnails into a 2-dimensional space.

[ ]:
```
def visualize_images_space(perplexity: int=5):
    import os
    import PIL
    from sklearn.preprocessing import StandardScaler
    from sklearn.manifold import TSNE
    from matplotlib.offsetbox import OffsetImage, AnnotationBbox

    # load images to list
    imgs = list()
    paths = list()
    for root, dirs, files in os.walk(os.getcwd()):
        for file in files:
            if file.endswith(".png"):
                img = np.asarray(PIL.Image.open(file).convert("L"))
                imgs.append(img.flatten())
                paths.append(os.path.join(root,file))
    imgs = np.array(imgs)

    imgs_std = StandardScaler().fit_transform(imgs)
    tsne = TSNE(n_components=2, perplexity=perplexity)
    imgs_tsne = tsne.fit_transform(imgs_std)

    fig, ax = plt.subplots(figsize=(28,14))
    for file, position in zip(paths, imgs_tsne):
        im = plt.imread(file, format="png")

        imagebox = OffsetImage(im, zoom=0.2)
        imagebox.image.axes = ax

        ab = AnnotationBbox(imagebox, (position[0], position[1]))
        ax.add_artist(ab)

    ax.set_xlim(-300, 300)
    ax.set_ylim(-300, 300)
```

[ ]:
```
visualize_images_space(perplexity=10)
```



3 homework - work with more complex data (higher dimensionality).

- use a basic machine learning set representing black and white scans of 32x32 pixel handwritten digits (MNIST)
- apply dimensionality reduction on these data using PCA, MDS and t-SNE methods
- Plot results
- Pick first 1000-2000 points as TSNE takes a lot of time for all samples

[ ]:
```
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.
 ↪load_data(path='mnist.npz')
print(x_train.shape)
print(y_train.shape)
```

```
(60000, 28, 28)
(60000,)
```

[ ]:
```
from sklearn.decomposition import PCA, KernelPCA
from sklearn.manifold import TSNE, MDS
from sklearn.preprocessing import StandardScaler

# Methods
pca = PCA(n_components=2)
kpca = KernelPCA(n_components=2, kernel="rbf")
tsne = TSNE(n_components=2, init="pca", perplexity=30.)
mds = MDS(n_components=2)
```

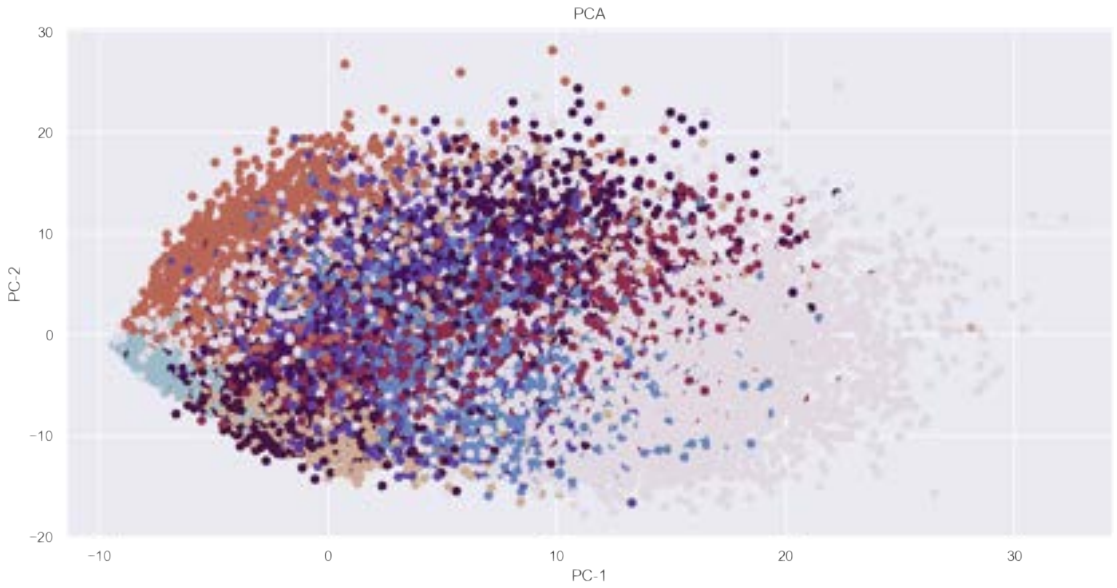
[ ]:
```
x_train_std = StandardScaler().fit_transform(x_train.reshape(x_train.shape[0],␣
↪-1))
```

3.0.1 PCA on MNIST

[ ]:
```
x_train_pca = pca.fit_transform(x_train_std)
```

[ ]:
```
plt.figure(figsize=(14,7))
plt.scatter(x_train_pca[:, 0], x_train_pca[:, 1], c=y_train, cmap="twilight")
plt.xlabel("PC-1")
plt.ylabel("PC-2")
plt.title("PCA")
```

[ ]:
```
Text(0.5, 1.0, 'PCA')
```

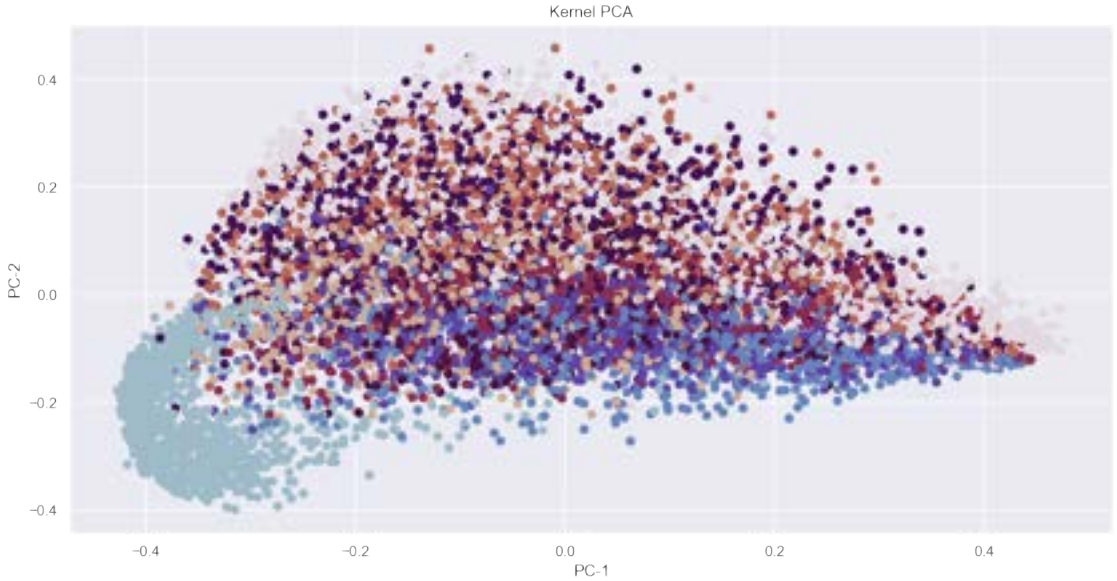


3.0.2 Gaussian Kernel PCA on MNIST

[ ]:
```
x_train_kpca = kpca.fit_transform(x_train_std[:10000])
```

[ ]:
```
plt.figure(figsize=(14,7))
plt.scatter(x_train_kpca[:, 0], x_train_kpca[:, 1], c=y_train[:10000],␣
↪cmap="twilight")
plt.xlabel("PC-1")
plt.ylabel("PC-2")
plt.title("Kernel PCA")
```

[ ]:
```
Text(0.5, 1.0, 'Kernel PCA')
```

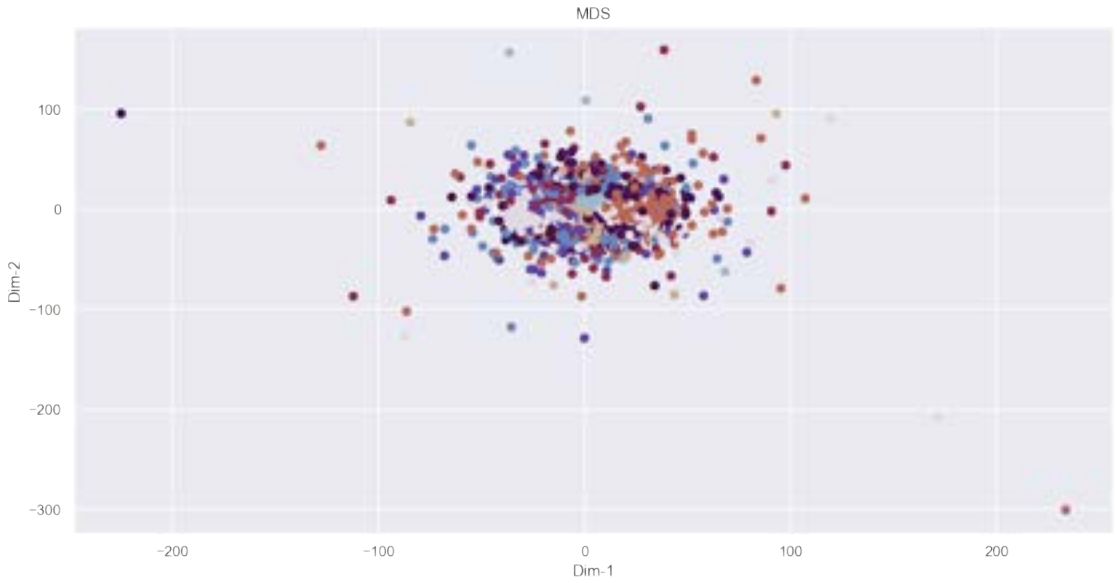


3.0.3 MDS on MNIST

[ ]:
```
x_train_mds = mds.fit_transform(x_train_std[:5000])
```

[ ]:
```
plt.figure(figsize=(14,7))
plt.scatter(x_train_mds[:, 0], x_train_mds[:, 1], c=y_train[:5000],␣
↪cmap="twilight")
plt.xlabel("Dim-1")
plt.ylabel("Dim-2")
plt.title("MDS")
```

[ ]:
```
Text(0.5, 1.0, 'MDS')
```

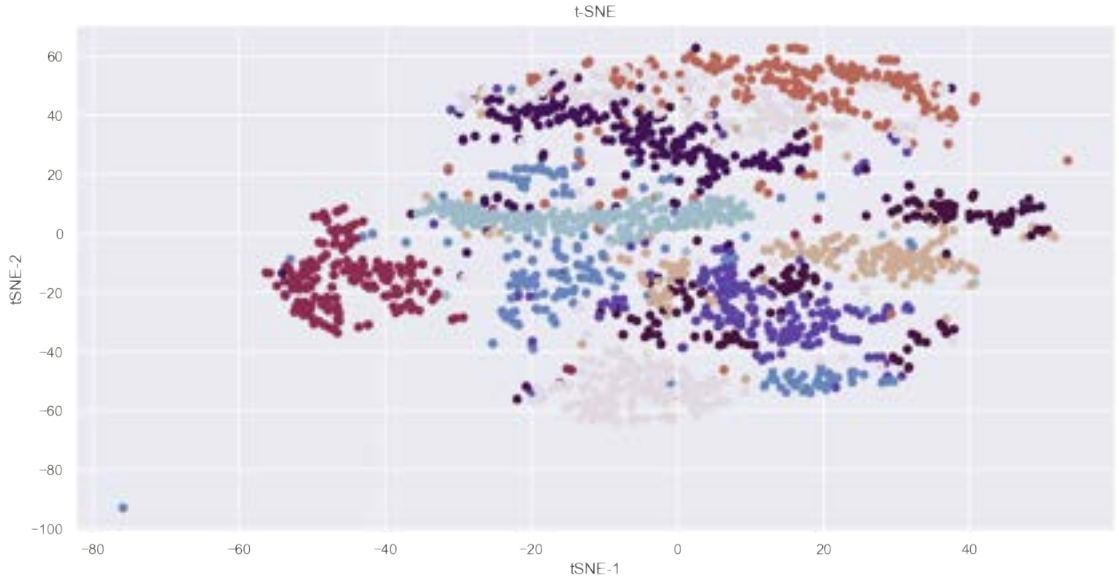


3.0.4 t-SNE on MNIST

[ ]:
```
x_train_tsne = tsne.fit_transform(x_train_std[:2000])
```

[ ]:
```
plt.figure(figsize=(14,7))
plt.scatter(x_train_tsne[:, 0], x_train_tsne[:, 1], c=y_train[:2000],␣
↪cmap="twilight")
plt.xlabel("tSNE-1")
plt.ylabel("tSNE-2")
plt.title("t-SNE")
```

[ ]:
```
Text(0.5, 1.0, 't-SNE')
```


t-SNE
60
40
20
0
−20
−40
−60
−80
−100
tSNE-2
−80
−60
−40
−20
0
20
40
tSNE-1

[ ]: