ISOMAP: A Non-Linear Dimensionality Reduction Technique

Python Implementation & Simulations

Authored By: Bushra Haque, Luis Sierra Muntané, Yichen Ji

This notebook details the code and corresponding simulation results in relation to the STA 2201 S25 Final Project. It is recommended that the notebook only be exported to an HTML file since all visualizations were completed with the Plotly library which tends to run into display errors if exported to a PDF.

In [34]:
```
# IMPORTS
import pandas as pd
import numpy as np
import scipy.io
from sklearn.metrics import pairwise_distances
from sklearn.datasets import make_swiss_roll
from torchvision import datasets, transforms
import kagglehub

import plotly.express as px
from plotly.subplots import make_subplots
import plotly.io as pio
from PIL import Image

import time
from random import seed as set_seed, sample

pio.renderers.default = 'notebook_connected'
```

1. ISOMAP Implementation (& Some Plotting Helper Functions)

See the project report for the specific algorithm designs for the two ISOMAP variants. The implementation divides the algorithm into three distinct chunks:

1. Create neighourhood graph based on $\boldsymbol{k}$-NN or $\boldsymbol{\varepsilon}$-NN.
2. Compute shortest distance graph using the Ffloyd-Warshall algorithm.
3. Perform Multidimensional Scaling (MDS) to retrieve low-dimensional representation.

Please see `implement.py` and `plot.py` for detailed function documentation. They have been omitted in this notebook.

In [35]:
```
# STEP 1: NEIGHBOURHOOD GRAPH

def construct_neighbourhood_graph(X, metric='euclidean', variant='k', hp=3):

    n = X.shape[0]

    pwD = pairwise_distances(X, metric=metric)

    G = np.full((n, n), np.inf)

    # Determine indices of neighbours:
    if variant=='k':
        knn_indices = np.argsort(pwD, axis=1)[:, 1:hp+1]

        row_indices = np.repeat(np.arange(n), hp)
        col_indices = knn_indices.ravel()
    elif variant=='eps':
        mask = (pwD < hp) & (pwD > 0)
        row_indices, col_indices = np.where(mask)

    distances = pwD[row_indices, col_indices]

    # Set weights of neighbours as distance:
    G[row_indices, col_indices] = distances
    G[col_indices, row_indices] = distances

    return G
```

In [36]:
```
# STEP 2: FLOYD-WARSHALL

def construct_shortest_distance_graph(G):

    n = G.shape[0]
    D = G.copy()

    # Iterate over all vertices to update shortest paths:
    for v in range(n):
        D = np.minimum(D, D[:, v][:, np.newaxis] + D[v, :])

    return D
```

In [37]:
```
# STEP 3: MDS

def construct_low_dim_embedding(D, d=2, seed=2201):

    n = D.shape[0]
    D_squared = D ** 2

    # Compute Gram matrix after double-centering:
    C = np.eye(n) - np.ones((n, n)) / n
    tau = -0.5 * (C @ D_squared @ C)

    # Perform eigendecomposition:
    set_seed(seed)
    eigvals, eigvecs = np.linalg.eig(tau)

    idx = np.argsort(eigvals)[::-1]
```

```
    eigvals = eigvals[idx]
    eigvecs = eigvecs[:, idx]

    # Select top d eigenvectors to construct embedding:
    L = np.diag(np.sqrt(eigvals[:d]))
    Y = eigvecs[:, :d] @ L

    return Y
```

In [38]:
```
# MAIN FUNCTION

def isomap(X, variant='k', hp=3, d=2, seed=2201):

    G = construct_neighbourhood_graph(X, metric='euclidean', variant=variant, hp=hp)
    D = construct_shortest_distance_graph(G)
    Y = construct_low_dim_embedding(D, d=d, seed=seed)

    return G, D, Y
```

Additional plotting helper functions:

In [39]:
```
# CONSTRUCT SCATTERPLOT OF ANY 2D/3D DATA

def scatterplot(X, title=None, labels=None, cpal='Turbo'):

    n, d = X.shape

    assert d==2 or d==3, "This function only accepts embeddings in 2D or 3D - try again."

    if title is None:
        title = f"{d}D Embedding from ISOMAP"

    if d == 2:
        if labels is not None:
            fig = px.scatter(x=X[:, 0], y=X[:, 1], title=title, color=labels, color_continuous_scale=cpal)
        else:
            fig = px.scatter(x=X[:, 0], y=X[:, 1], title=title)
    elif d == 3:
        if labels is not None:
            fig = px.scatter_3d(x=X[:, 0], y=X[:, 1], z=X[:, 2], title=title, color=labels, color_continuous_scale=cpal)
        else:
            fig = px.scatter_3d(x=X[:, 0], y=X[:, 1], z=X[:, 2], title=title)

    fig.update_traces(marker_size=5)

    return fig
```

In [40]:
```
# ADD IMAGE ANNOTATIONS

def image_annotations(fig, X, image_paths, image_idx, sizex=5, sizey=5, subset=False, subset_idx=None):

    if subset:
        image_idx = np.where(np.isin(subset_idx, image_idx))[0]

    for i, image_path in zip(image_idx, image_paths):
        image = Image.open(image_path)
        fig.add_layout_image(
            dict(
                source=image,
                x=X[i, 0],
                y=X[i, 1],
                xref='x',
                yref='y',
                sizex=sizex,
                sizey=sizey,
                xanchor='center',
                yanchor='middle'
            )
        )

    return None
```

2. ISOMAP Simulations

The following datasets were selected to understand the effectiveness of ISOMAP (using the above implementation):

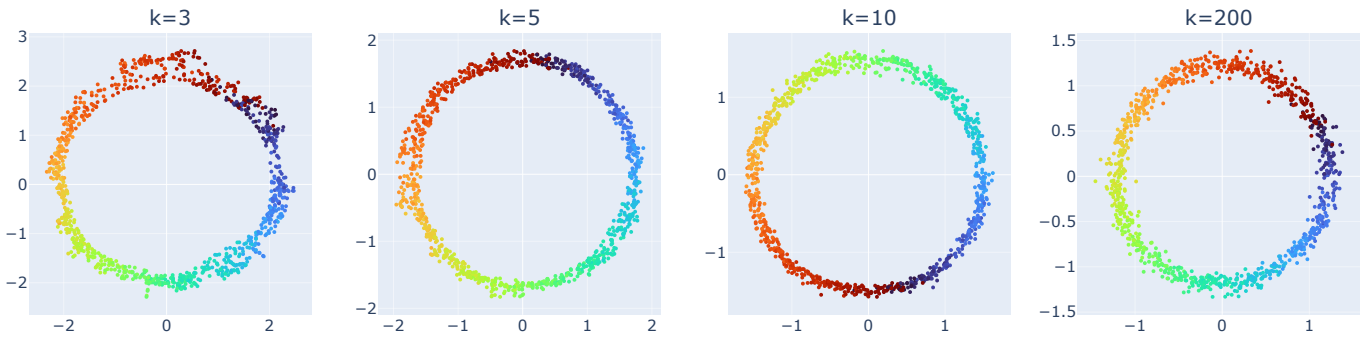
1. Cylinder
2. Swiss Roll
3. Stanford Faces
4. MNIST

2.1 Cylinder

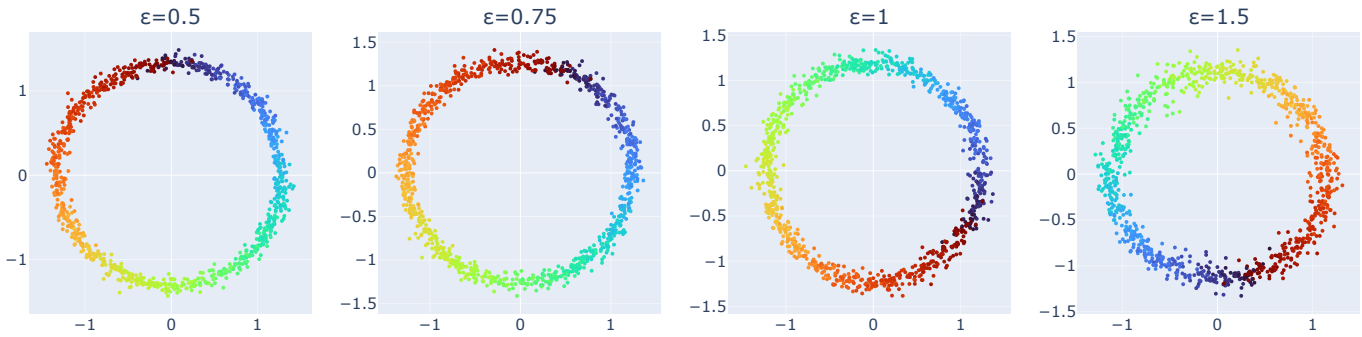
In [41]:
```
# GENERATE DATA

set_seed(2201)

n_points = 1000
theta = np.random.uniform(0, 2 * np.pi, n_points)

z = np.random.uniform(-1, 1, n_points)
x = np.cos(theta) + 0.1 * np.random.normal(size=n_points)
y = np.sin(theta) + 0.1 * np.random.normal(size=n_points)

cylinder = np.column_stack((x, y, z))

fig_cyl = scatterplot(cylinder, title='')
fig_cyl.update_traces(marker=dict(color=theta, colorscale='Turbo'), marker_size=3)
```

```
fig_cyl.update_layout(width=600, margin=dict(l=20, r=20, t=20, b=20))
fig_cyl.show()
```

In [42]:
```
# k-ISOMAP

variant='k'
hp_values = [3,5,10,200]

fig_multi = make_subplots(rows=1, cols=4, subplot_titles=[f'{variant}={hp}' for hp in hp_values])

for i, hp in enumerate(hp_values):
    # print(f'Now running {variant}-ISOMAP for {variant}={hp}')
    embed = isomap(cylinder, variant=variant, hp=hp, d=2)[2]
    fig_embed = scatterplot(embed, title=f'')
    fig_embed.update_xaxes(range=(-3, 3))
    fig_embed.update_yaxes(range=(-3, 3))
    fig_embed.update_traces(marker=dict(color=theta, colorscale='Turbo'), marker_size=3)

    fig_multi.add_traces(fig_embed.data, rows=[1]*len(fig_embed.data), cols=[i+1]*len(fig_embed.data))

fig_multi.update_layout(height=400, width=1200)
fig_multi.show()
```



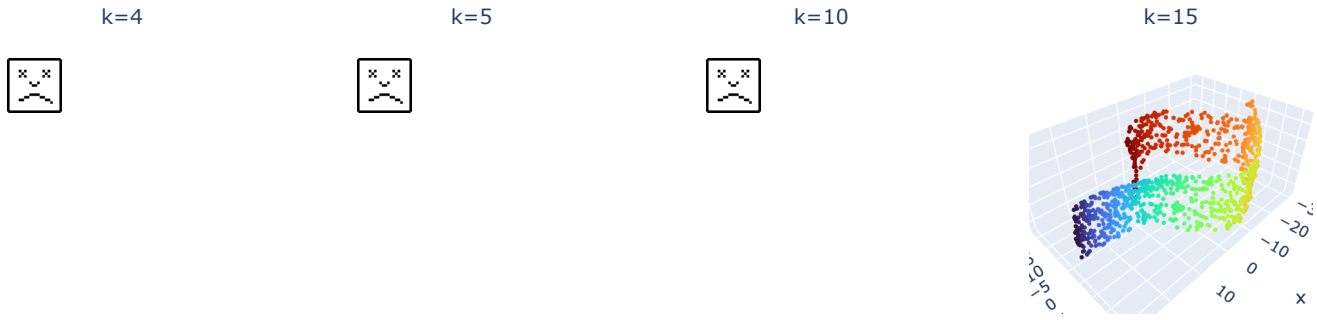
In [43]:
```
# eps-ISOMAP

variant='eps'
hp_values = [0.5,0.75,1,1.5]

fig_multi = make_subplots(rows=1, cols=4, subplot_titles=[f'ε={hp}' for hp in hp_values])

for i, hp in enumerate(hp_values):
    # print(f'Now running {variant}-ISOMAP for {variant}={hp}')
    cyl_embed = isomap(cylinder, variant=variant, hp=hp, d=2)[2]
    fig_embed = scatterplot(cyl_embed, title=f'')
    fig_embed.update_xaxes(range=(-2.5, 2.5))
    fig_embed.update_yaxes(range=(-2.5, 2.5))
    fig_embed.update_traces(marker=dict(color=theta, colorscale='Turbo'), marker_size=3)

    fig_multi.add_traces(fig_embed.data, rows=[1]*len(fig_embed.data), cols=[i+1]*len(fig_embed.data))

fig_multi.update_layout(height=400, width=1200)
fig_multi.show()
```



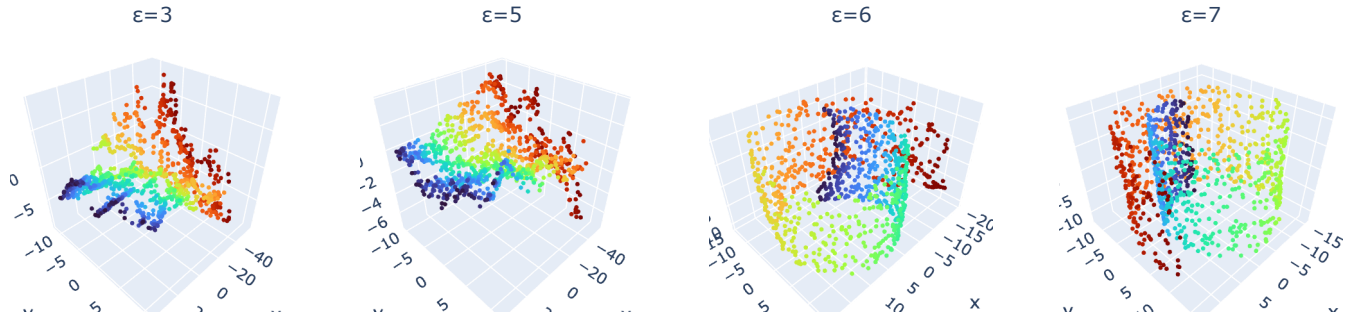
2.2 Swiss Roll

In [44]:
```
# GENERATE DATA
swiss_roll, t = make_swiss_roll(n_samples=1000, noise=0.1, random_state=2201)

fig_swiss_roll = scatterplot(swiss_roll, title='')
fig_swiss_roll.update_traces(marker=dict(color=t, colorscale='Turbo'), marker_size=3)
fig_swiss_roll.update_layout(width=600, margin=dict(l=20, r=20, t=20, b=20))
fig_swiss_roll.show()
```

In [45]:
```
# k-ISOMAP

variant='k'
hp_values = [4,5,10,15]

fig_multi = make_subplots(
    rows=1, cols=4, subplot_titles=[f'{variant}={hp}' for hp in hp_values],
    specs=[[{'type': 'scatter3d'}, {'type': 'scatter3d'}, {'type': 'scatter3d'}, {'type': 'scatter3d'}]]
)

for i, hp in enumerate(hp_values):
    # print(f'Now running {variant}-ISOMAP for {variant}={hp}')
    embed = isomap(swiss_roll, variant=variant, hp=hp, d=3)[2]
    fig_embed = scatterplot(embed, title='')
    fig_embed.update_traces(marker=dict(color=t, colorscale='Turbo'), marker_size=2)

    fig_multi.add_traces(fig_embed.data, rows=[1]*len(fig_embed.data), cols=[i+1]*len(fig_embed.data))

fig_multi.update_layout(height=400, width=1200)
fig_multi.show()
```

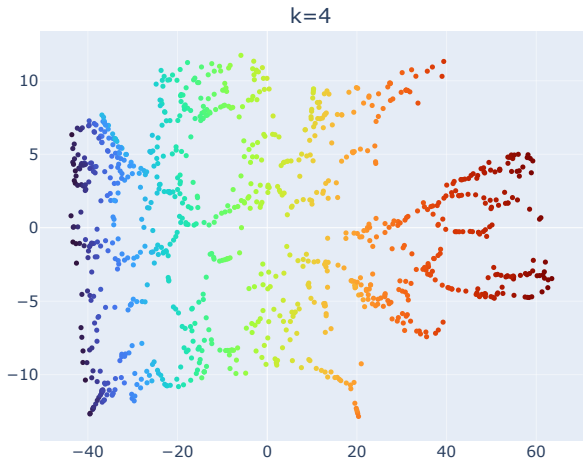


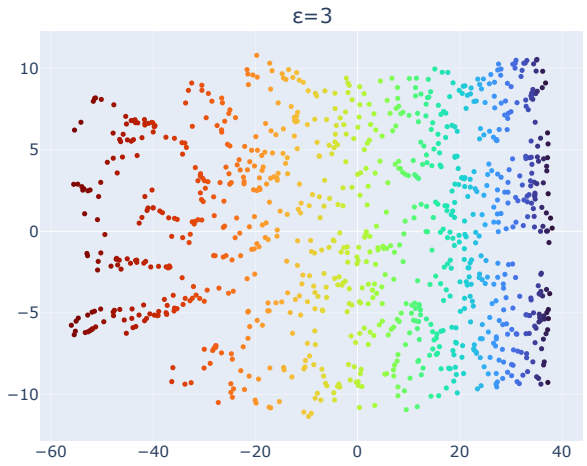
In [46]:
```
# eps-ISOMAP

variant='eps'
hp_values = [3,5,6,7]

fig_multi = make_subplots(
    rows=1, cols=4, subplot_titles=[f'ε={hp}' for hp in hp_values],
    specs=[[{'type': 'scatter3d'}, {'type': 'scatter3d'}, {'type': 'scatter3d'}, {'type': 'scatter3d'}]]
)

for i, hp in enumerate(hp_values):
    # print(f'Now running {variant}-ISOMAP for {variant}={hp}')
    embed = isomap(swiss_roll, variant=variant, hp=hp, d=3)[2]
    fig_embed = scatterplot(embed, title='')
    fig_embed.update_traces(marker=dict(color=t, colorscale='Turbo'), marker_size=2)

    fig_multi.add_traces(fig_embed.data, rows=[1]*len(fig_embed.data), cols=[i+1]*len(fig_embed.data))

fig_multi.update_layout(height=400, width=1200)
fig_multi.show()
```



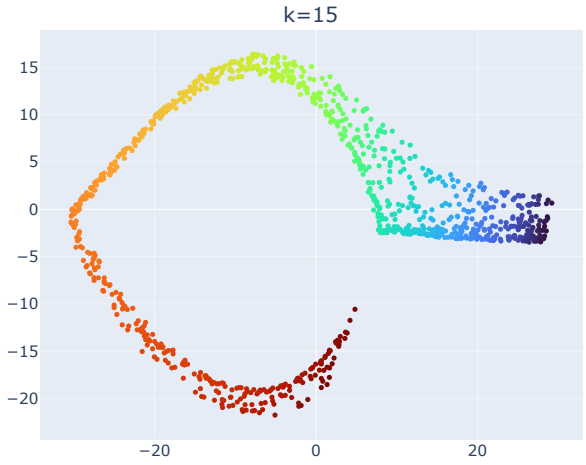
Unrolled Swiss Roll

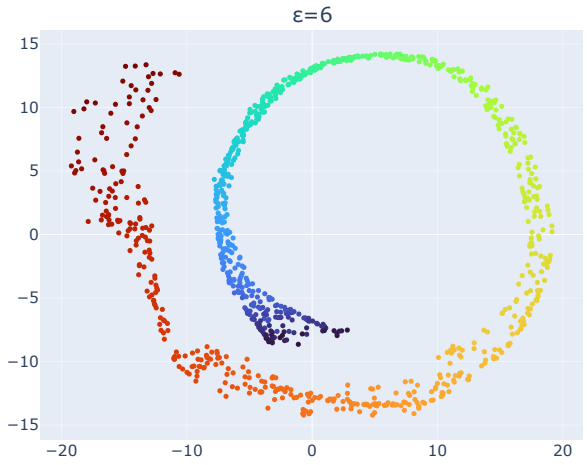
In [47]:
```
# UNROLLED GOOD

fig_multi = make_subplots(
    rows=1, cols=2, subplot_titles=['k=4', 'ε=3']
)

# k-ISOMAP
k_embed = isomap(swiss_roll, variant='k', hp=4, d=2)[2]
fig_k_embed = scatterplot(k_embed, title='')
fig_k_embed.update_traces(marker=dict(color=t, colorscale='Turbo'), marker_size=4)

fig_multi.add_traces(fig_k_embed.data, rows=[1]*len(fig_k_embed.data), cols=[1]*len(fig_k_embed.data))

# eps-ISOMAP
e_embed = isomap(swiss_roll, variant='eps', hp=3, d=2)[2]
fig_e_embed = scatterplot(e_embed, title='')
fig_e_embed.update_traces(marker=dict(color=t, colorscale='Turbo'), marker_size=4)

fig_multi.add_traces(fig_e_embed.data, rows=[1]*len(fig_e_embed.data), cols=[2]*len(fig_e_embed.data))
fig_multi.update_layout(height=400, width=1000)
fig_multi.update_layout( margin=dict(l=20, r=20, t=50, b=30))
fig_multi.show()
```





In [48]:
```
# UNROLLED BAD

fig_multi = make_subplots(
    rows=1, cols=2, subplot_titles=['k=15', 'ε=6']
)

# k-ISOMAP
k_embed = isomap(swiss_roll, variant='k', hp=16, d=2)[2]
fig_k_embed = scatterplot(k_embed, title='')
fig_k_embed.update_traces(marker=dict(color=t, colorscale='Turbo'), marker_size=4)

fig_multi.add_traces(fig_k_embed.data, rows=[1]*len(fig_k_embed.data), cols=[1]*len(fig_k_embed.data))

# eps-ISOMAP
e_embed = isomap(swiss_roll, variant='eps', hp=6, d=2)[2]
fig_e_embed = scatterplot(e_embed, title='')
fig_e_embed.update_traces(marker=dict(color=t, colorscale='Turbo'), marker_size=4)

fig_multi.add_traces(fig_e_embed.data, rows=[1]*len(fig_e_embed.data), cols=[2]*len(fig_e_embed.data))
fig_multi.update_layout(height=400, width=1000)
fig_multi.update_layout( margin=dict(l=20, r=20, t=50, b=30))
fig_multi.show()
```





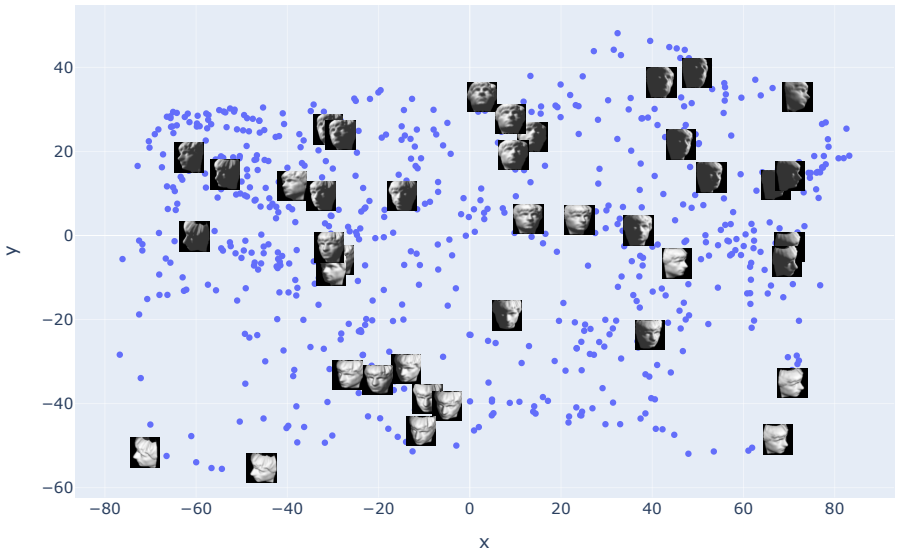
2.3 Stanford Faces

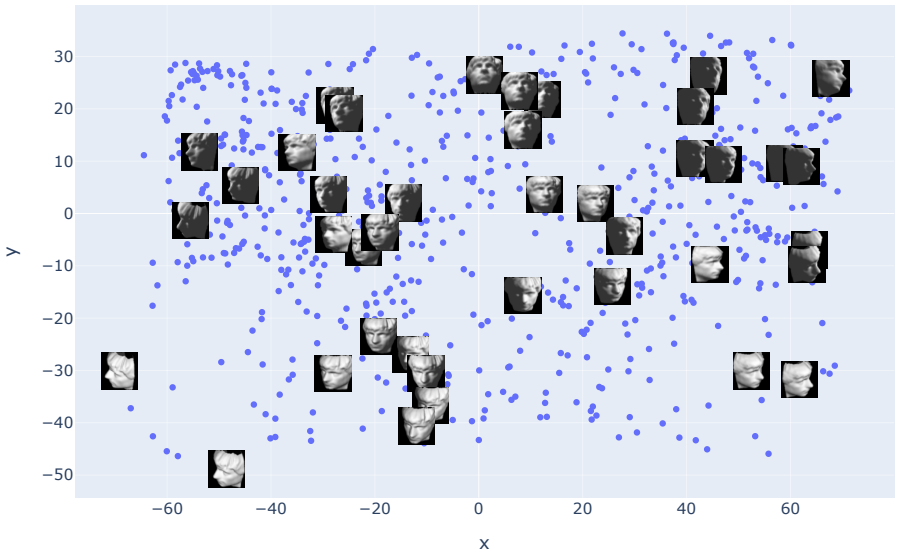
Multiples sources have cited that this infamous dataset originates from Stanford but there doesn't appear to be any active/discoverable directories hosting the dataset. For the purposes of this project, it was sourced from here: https://github.com/lwileczek/isomap/tree/master/data

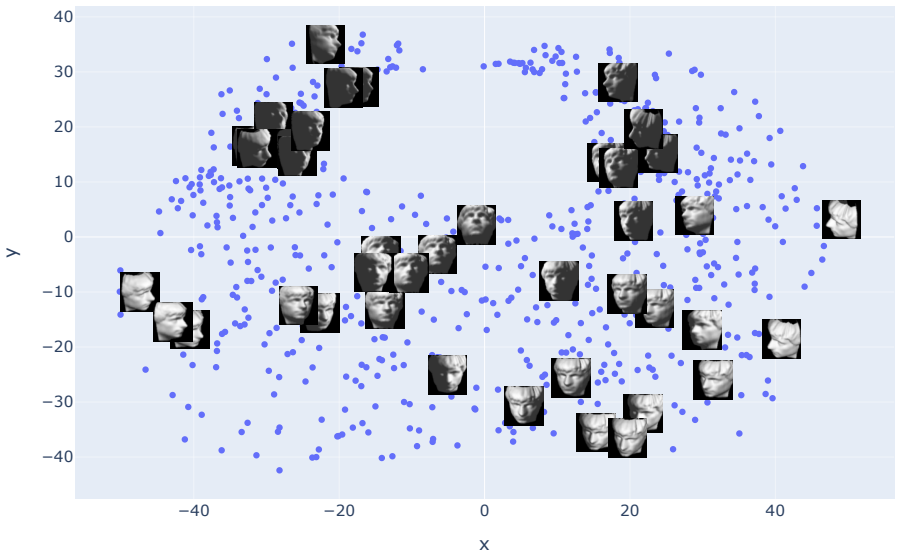
In [49]:
```
# LOAD STANFORD FACES DATASET
faces = scipy.io.loadmat('/Users/bushra/Documents/STA2201/isomap/data/faces/face_data.mat')['images'].T

# Select images to annotate
set_seed(2201)
image_idx = sample(range(faces.shape[0]), 40)
image_paths = []

for i in image_idx:
    image_array = faces[i].reshape(64, 64).T
    image = Image.fromarray((image_array * 255).astype(np.uint8))
    image_path = f"/Users/bushra/Documents/STA2201/isomap/data/faces/face_{i}.png"
    image.save(image_path)
    image_paths.append(image_path)
```

In [50]:
```
# k-ISOMAP

variant='k'
hp_values = [4,6,10] # k
img_size = [7,7,7]

for hp, sz in zip(hp_values, img_size):
    face_embed = isomap(faces, variant=variant, hp=hp, d=2)[2]
```

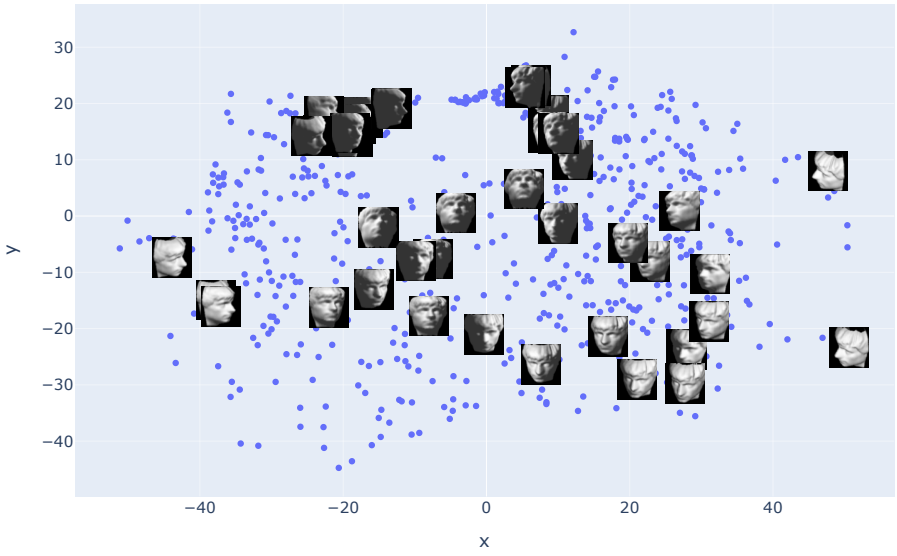
```
    title = f'{variant}-ISOMAP Embedding of Faces, {variant}={hp}'
    title = ''
    fig_faces_embed = scatterplot(face_embed, title=title)
    image_annotations(fig=fig_faces_embed, X=face_embed, image_paths=image_paths, image_idx=image_idx, sizex=sz, sizey=sz)
    fig_faces_embed.update_layout(width=800)
    fig_faces_embed.show()
```

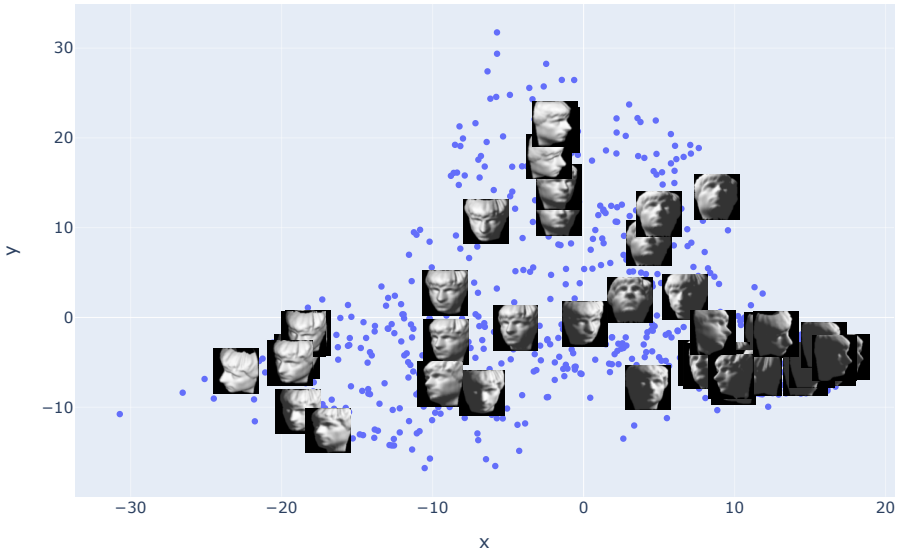


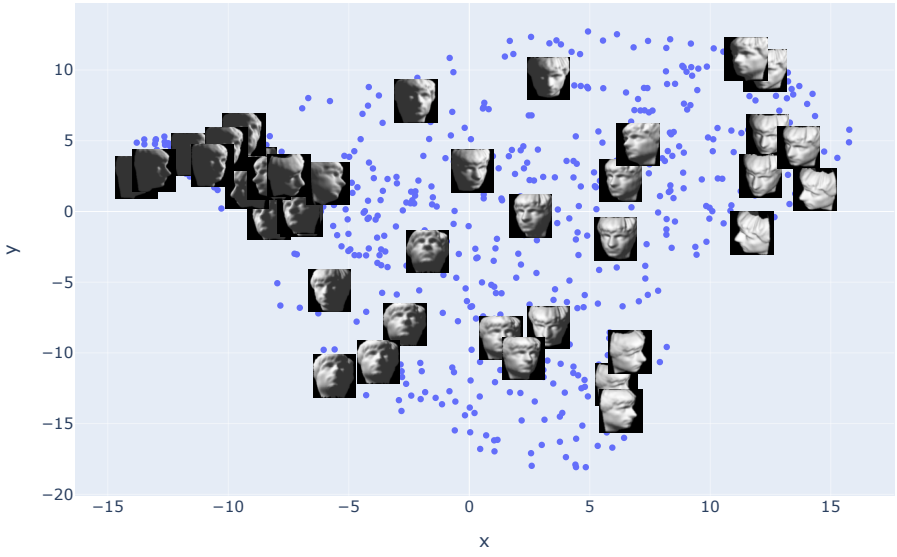




In [51]:
```
# eps-ISOMAP

variant='eps'
hp_values = [11,20,45] # eps
img_size = [7,5,3]

for hp, sz in zip(hp_values, img_size):
    face_embed = isomap(faces, variant=variant, hp=hp, d=2)[2]

    title = f'ε-ISOMAP Embedding of Faces, ε={hp}'
    title = ''
    fig_faces_embed = scatterplot(face_embed, title=title)
    image_annotations(fig=fig_faces_embed, X=face_embed, image_paths=image_paths, image_idx=image_idx, sizex=sz, sizey=sz)
    fig_faces_embed.update_layout(width=800)
    fig_faces_embed.show()
```





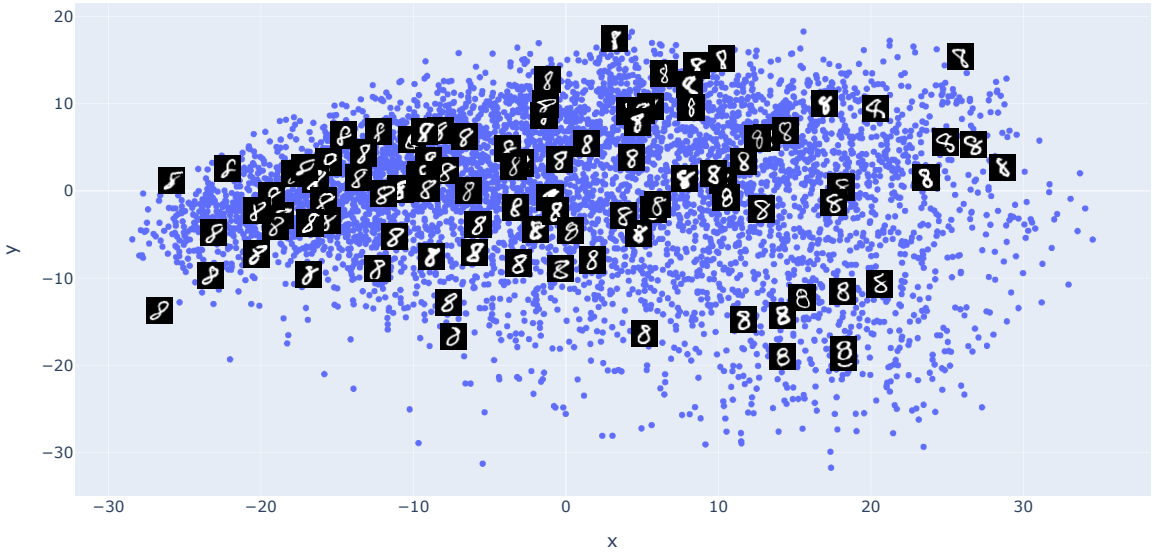


2.4 MNIST

2.4.1 Only Digit 8

In [52]:
```
# LOAD & PREP DATA
DIGITS = [8]

# Load MNIST via PyTorch
transform = transforms.ToTensor()
mnist_train = datasets.MNIST(root='./data', train=True, download=True, transform=transform)

# Format & filter data
mnist = np.array([image.flatten() for image, _ in mnist_train])
labels = np.array([label for _, label in mnist_train])

mask = np.isin(labels, DIGITS)
subset_idx = np.where(mask)[0]

mnist_sub = mnist[mask]
labels_sub = labels[mask]

# Select images to annotate
set_seed(2201)
image_idx = np.sort(np.random.choice(subset_idx, size=100, replace=False))
image_paths = []

for i in image_idx:
    image_array = mnist[i].reshape(28, 28)
    image = Image.fromarray((image_array * 255).astype(np.uint8))
    image_path = f'/Users/bushra/Documents/STA2201/isomap/data/mnist/image_{i}.png'
    image.save(image_path)
    image_paths.append(image_path)
```

In [53]:
```
# k-ISOMAP (pre-computed & saved)

# # Perform ISOMAP
# start_time = time.time()
# mnist_embed = isomap(mnist_sub, variant='k', hp=6, d=2)[2]
# end_time = time.time()
# print(f'ISOMAP Execution time: {end_time-start_time:.4f} seconds')

# # Save for good measure
# np.save('isomap/data/mnist/embedding_digit_8_.npy', mnist_embed)
mnist_embed = np.load('/Users/bushra/Documents/STA2201/isomap/data/mnist/embedding_digit_8_.npy')

# Plot
title = 'k-ISOMAP Embedding of MNIST Digit 8, k=6'
title = ''
fig_mnist_embed = scatterplot(mnist_embed, title=title)
image_annotations(
    fig=fig_mnist_embed,
    X=mnist_embed,
    image_paths=image_paths,
    image_idx=image_idx,
    subset=True,
    subset_idx=subset_idx,
    sizex=3, sizey=3)
fig_mnist_embed.update_layout(width=1000)
fig_mnist_embed.show()
```



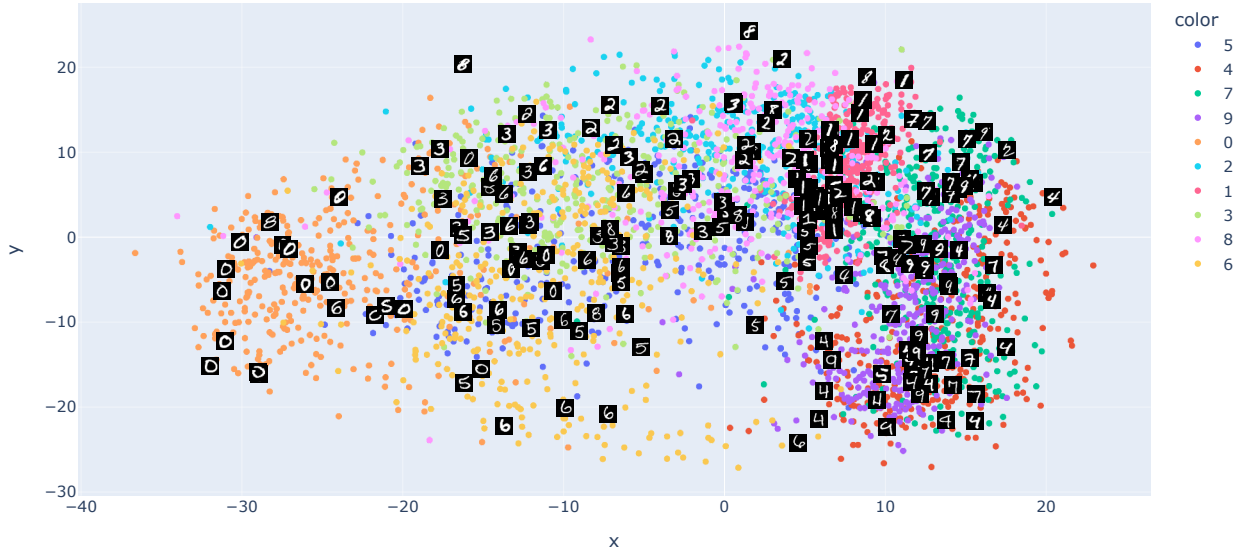
2.4.2 Uniformly Sample per Class

In [54]:
```
# # FIRST SAMPLE FROM CLASS FOR ISOMAP
# set_seed(2201)
# digit_idx = np.array([], dtype=int)
# image_idx = np.array([], dtype=int)

# for digit in range(10):
#     curr_digit_idx = np.where(labels == digit)[0]
#     sampled_digit_idx = np.random.choice(curr_digit_idx, size=400, replace=False)
#     digit_idx = np.append(digit_idx, sampled_digit_idx)

#     sampled_image_idx = np.random.choice(sampled_digit_idx, size=20, replace=False)
#     image_idx = np.append(image_idx, sampled_image_idx)

# digit_idx = np.sort(digit_idx)
# image_idx = np.sort(image_idx)

digit_idx = np.load('/Users/bushra/Documents/STA2201/isomap/data/mnist/unif_digits.npy')
image_idx = np.load('/Users/bushra/Documents/STA2201/isomap/data/mnist/unif_images.npy')
image_paths = []

for i in image_idx:
    image_array = mnist[i].reshape(28, 28)
    image = Image.fromarray((image_array * 255).astype(np.uint8))
    image_path = f'/Users/bushra/Documents/STA2201/isomap/data/mnist/image_{i}.png'
    image.save(image_path)
    image_paths.append(image_path)
```

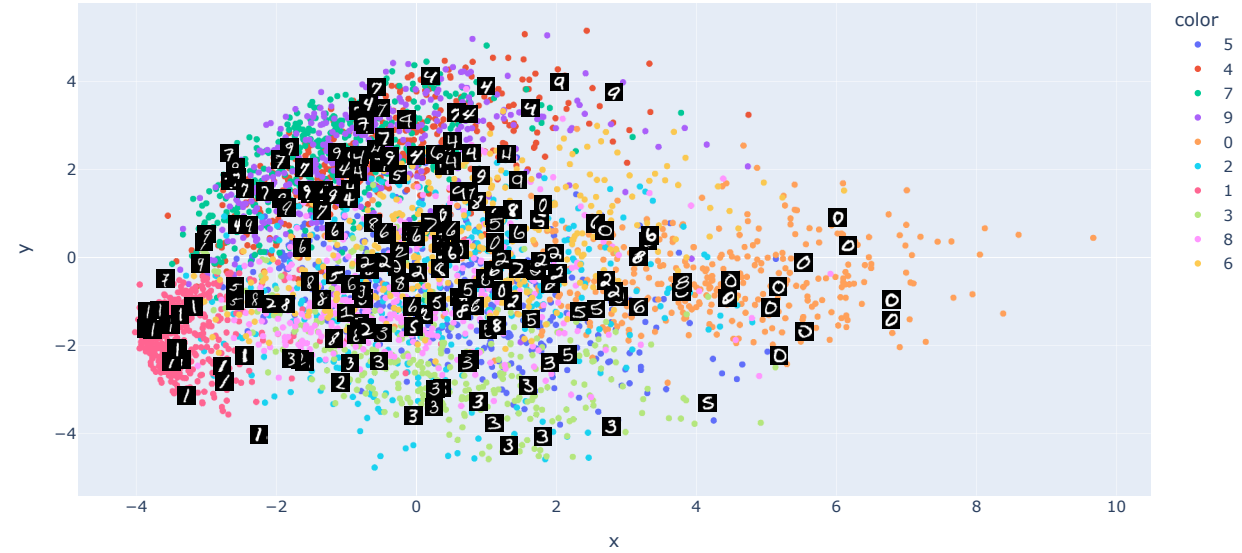
In [55]:
```
mnist_sub = mnist[digit_idx]
labels_sub = labels[digit_idx]

# np.save('/Users/bushra/Documents/STA2201/isomap/data/mnist/unif_digits.npy', digit_idx)
# np.save('/Users/bushra/Documents/STA2201/isomap/data/mnist/unif_images.npy', image_idx)

np.unique(labels_sub, return_counts=True)
```

Out[55]:
```
(array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),
 array([400, 400, 400, 400, 400, 400, 400, 400, 400, 400]))
```

In [56]:
```
# k-ISOMAP (pre-computed & saved)

# # Perform ISOMAP
# start_time = time.time()
# mnist_embed = isomap(mnist_sub, variant='k', hp=6, d=2)[2]
# end_time = time.time()
# print(f'ISOMAP Execution time: {end_time-start_time:.4f} seconds')

# # Save for good measure
# np.save('/Users/bushra/Documents/STA2201/isomap/data/mnist/embedding_unif_k_6.npy', mnist_embed)
mnist_embed = np.load('/Users/bushra/Documents/STA2201/isomap/data/mnist/embedding_unif_k_6.npy')

# Plot
title = 'k-ISOMAP Embedding of MNIST (400 Per Digit), k=6'
title = ''
fig_mnist_embed = scatterplot(mnist_embed, title=title, labels=[str(d) for d in labels_sub])
image_annotations(
    fig=fig_mnist_embed,
    X=mnist_embed,
    image_paths=image_paths,
    image_idx=image_idx,
    subset=True,
    subset_idx=digit_idx,
    sizex=2, sizey=2)
fig_mnist_embed.update_layout(width=1000)
fig_mnist_embed.show()
```



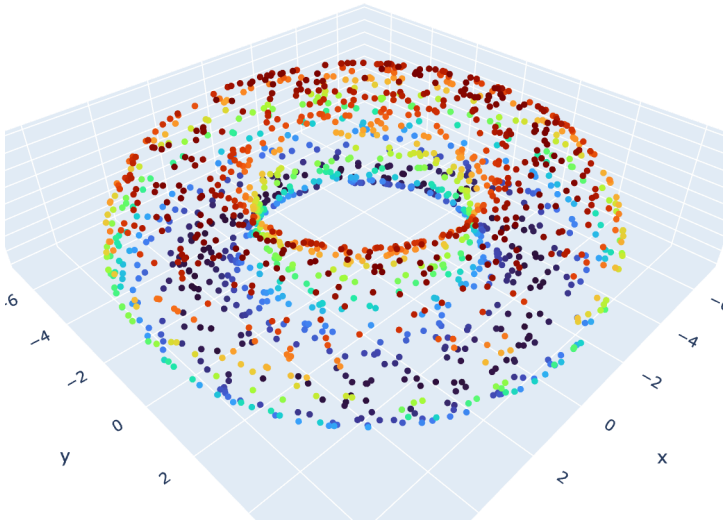
In [57]:
```
# ε-ISOMAP (pre-computed & saved)

# # Perform ISOMAP
# start_time = time.time()
# mnist_embed_e = isomap(mnist_sub, variant='eps', hp=15, d=2)[2]
# end_time = time.time()
# print(f'ISOMAP Execution time: {end_time-start_time:.4f} seconds')

# # Save for good measure
# np.save('/Users/bushra/Documents/STA2201/isomap/data/mnist/embedding_unif_e.npy', mnist_embed_e)
mnist_embed_e = np.load('/Users/bushra/Documents/STA2201/isomap/data/mnist/embedding_unif_e.npy')

# Plot
title = 'ε-ISOMAP Embedding of MNIST (400 Per Digit), ε=15'
title = ''
fig_mnist_embed = scatterplot(mnist_embed_e, title=title, labels=[str(d) for d in labels_sub])
image_annotations(
    fig=fig_mnist_embed,
    X=mnist_embed_e,
    image_paths=image_paths,
    image_idx=image_idx,
    subset=True,
    subset_idx=digit_idx,
    sizex=0.4, sizey=0.4)
fig_mnist_embed.update_layout(width=1000)
fig_mnist_embed.show()
```



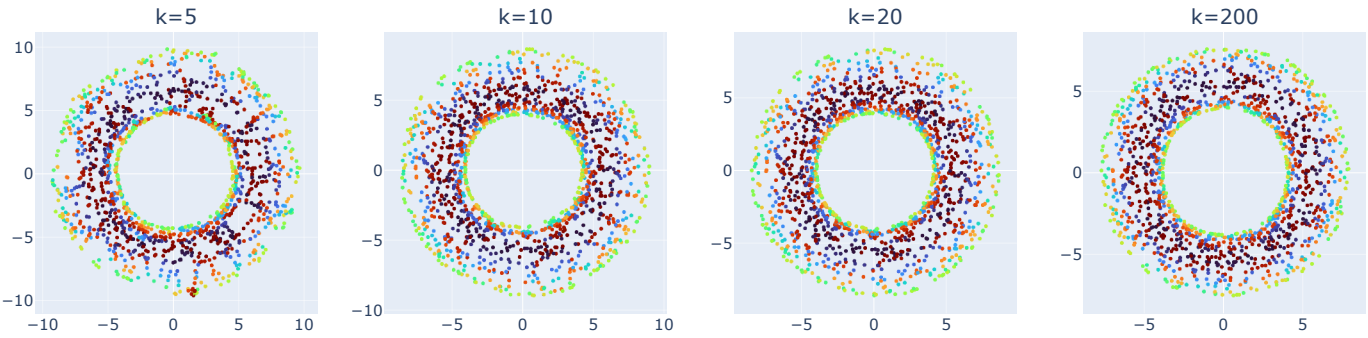
2.5 Torus/Doughnut

In [58]:
```
# GENERATE DATA

n = 1500
R = 5  # center to doughnut
r = 2  # doughnut cross-section radius

theta = np.random.uniform(0, 2 * np.pi, n)
phi = np.random.uniform(0, 2 * np.pi, n)
```

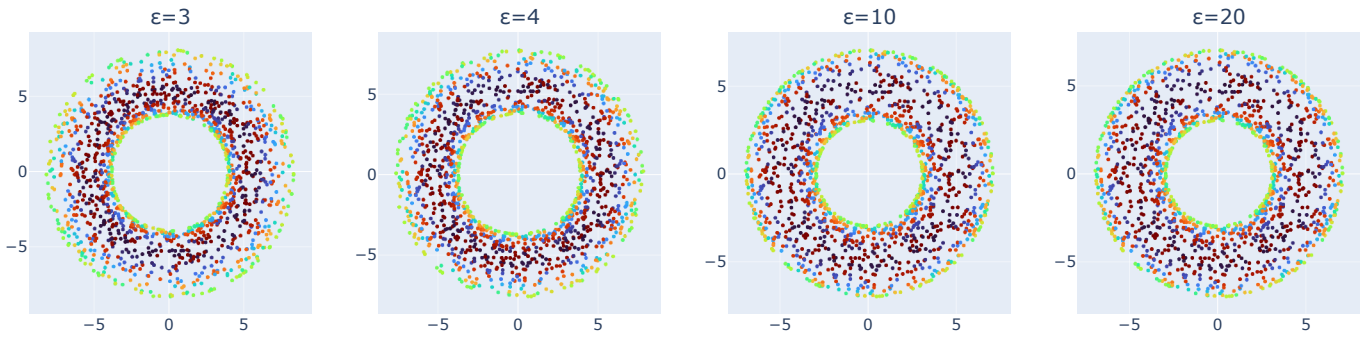
```
# Parametric equations
x = (R + r * np.cos(phi)) * np.cos(theta)
y = (R + r * np.cos(phi)) * np.sin(theta)
z = r * np.sin(phi)

torus = np.vstack((x, y, z)).T
fig_torus = scatterplot(torus, title='')
fig_torus.update_traces(marker=dict(color=z, colorscale='Turbo'), marker_size=3)
fig_torus.update_layout(width=600, margin=dict(l=20, r=20, t=20, b=20))
fig_torus.show()
```



In [59]:
```
# k-ISOMAP

variant='k'
hp_values = [5,10,20,200]

fig_multi = make_subplots(rows=1, cols=4, subplot_titles=[f'{variant}={hp}' for hp in hp_values])

for i, hp in enumerate(hp_values):
    # print(f'Now running {variant}-ISOMAP for {variant}={hp}')
    embed = isomap(torus, variant=variant, hp=hp, d=2)[2]
    fig_embed = scatterplot(embed, title=f'')
    fig_embed.update_traces(marker=dict(color=z, colorscale='Turbo'), marker_size=3)

    fig_multi.add_traces(fig_embed.data, rows=[1]*len(fig_embed.data), cols=[i+1]*len(fig_embed.data))

fig_multi.update_layout(height=400, width=1200)
fig_multi.show()
```



In [60]:
```
# eps-ISOMAP

variant='eps'
hp_values = [3,4,10,20]

fig_multi = make_subplots(rows=1, cols=4, subplot_titles=[f'ε={hp}' for hp in hp_values])

for i, hp in enumerate(hp_values):
    # print(f'Now running {variant}-ISOMAP for {variant}={hp}')
    embed = isomap(torus, variant=variant, hp=hp, d=2)[2]
    fig_embed = scatterplot(embed, title=f'')
    fig_embed.update_traces(marker=dict(color=z, colorscale='Turbo'), marker_size=3)

    fig_multi.add_traces(fig_embed.data, rows=[1]*len(fig_embed.data), cols=[i+1]*len(fig_embed.data))

fig_multi.update_layout(height=400, width=1200)
fig_multi.show()
```



In [ ]: