Parameter evaluation

The goal of this notebook is to decide on the best parameters for our TDA. This follows the two heuristics in Chang et al.

Parameters are used for the clustering algorithm and the Mapper algorithm. We try out different clustering algorithms, parameters for the clustering algorithm, and parameters for the clustering algorithm.

COND is removed in this analysis.

Load libraries

In [1]:

```
import kmapper as km
# import sklearn

from sklearn.cluster import DBSCAN # clustering algorithm
from sklearn.decomposition import PCA # projection (lens) creation
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import RobustScaler

import hdbscan

# from sklearn import ensemble
# from sklearn.manifold import MDS

import plotly.graph_objs as go
# from ipywidgets import interactive, HBox, VBox, widgets, interact # ?
# import dash_html_components as html # ?
# import dash_core_components as dcc # ?

from kmapper.plotlyviz import * # static and interactive plots
import psutil # for plotlyviz
import kaleido # for plotlyviz
# import networkx # ?

# import dash # ?
import warnings #?
warnings.filterwarnings("ignore")

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [25]:

```
import json as js
import pickle as pk
```

Read data

- Take subset of the data (only the 10 relevant continuous variables)
- Standardize the variables

In [2]:

```
water20 = pd.read_csv("../../LTRM data/RF interpolation/water_full.csv")
```

In [3]:

```
water20.head()
```

Out[3]:

	SHEETBAR	DATE	LATITUDE	LONGITUDE	FLDNUM	STRATUM	LOCATCD	TN	TP	TEMP	DO	TURB	COND	VEL	SS	WDP	CHLcal	SECCHI	YEAR	SEASON
0	41000065	07/26/1993	44.571864	-92.510970	Lake City, MN	Main channel	9312103	3.955	0.228	23.0	6.6	28	550.0	0.50	42.3	2.2	9.44875	40	1993	2
1	41000066	07/26/1993	44.575497	-92.518497	Lake City, MN	Main channel	9312002	4.876	0.229	23.0	6.6	28	554.0	0.72	37.6	8.2	8.24230	42	1993	2
2	41000067	07/26/1993	44.573718	-92.523549	Lake City, MN	Main channel	9312102	3.955	0.220	22.9	6.3	24	564.0	0.66	34.1	4.3	8.72488	43	1993	2
3	41000068	07/26/1993	44.566588	-92.541238	Lake City, MN	Main channel	9312003	4.257	0.212	22.9	6.4	28	563.0	0.69	33.4	9.1	8.48359	38	1993	2
4	41000069	07/26/1993	44.568419	-92.548780	Lake City, MN	Main channel	9312104	4.030	0.237	23.0	6.6	33	556.0	0.68	48.0	6.7	9.52918	45	1993	2

In [4]:

```
X = water20[["WDP", "SECCHI", "TEMP", "DO", "TURB",
             "VEL", "TP", "TN", "SS", "CHLcal"]]

X = StandardScaler().fit_transform(X)

# n_data = watershort.shape[0]
n_data = water20.shape[0]
```

In [5]:

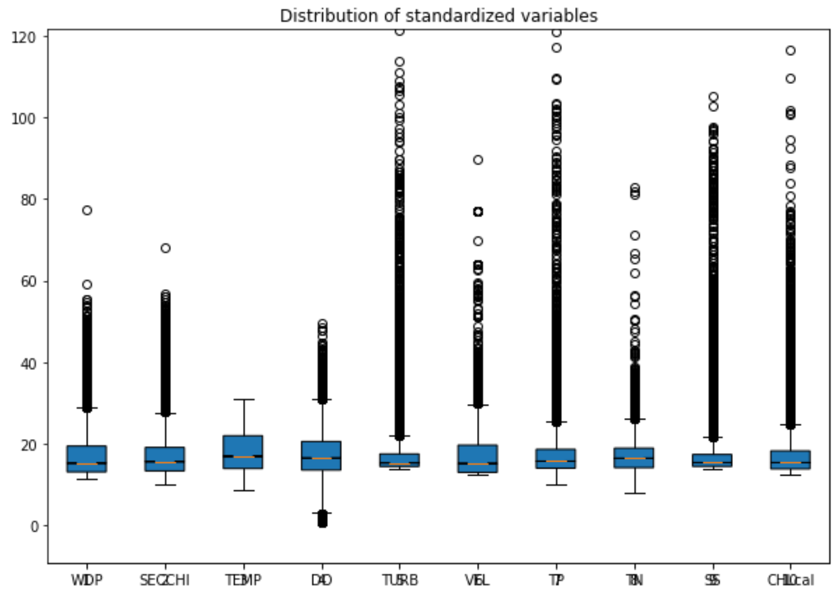
```
X = pd.DataFrame(X, columns = ["WDP", "SECCHI", "TEMP", "DO", "TURB",
                               "VEL", "TP", "TN", "SS", "CHLcal"])

X.head()
```

Out[5]:

	WDP	SECCHI	TEMP	DO	TURB	VEL	TP	TN	SS	CHLcal
0	-0.362348	-0.319580	0.861452	-1.070321	-0.201687	0.381970	0.178906	0.729310	-0.110019	-0.561444
1	1.517428	-0.268383	0.861452	-1.070321	-0.201687	0.923263	0.185471	1.305956	-0.165548	-0.601495
2	0.295574	-0.242785	0.850848	-1.163927	-0.263691	0.775638	0.126391	0.729310	-0.206899	-0.585475
3	1.799394	-0.370777	0.850848	-1.132725	-0.201687	0.849450	0.073876	0.918395	-0.215169	-0.593485
4	1.047484	-0.191589	0.861452	-1.070321	-0.124182	0.824846	0.237986	0.776269	-0.042676	-0.558773

In [50]:

```
fig = plt.figure(figsize =(10, 7))

# Creating plot
plt.boxplot(X)
ax = fig.add_subplot(111)

bp = ax.boxplot(X, patch_artist = True,
                notch ='True', vert = 1)


ax.set_xticklabels(["WDP", "SECCHI", "TEMP", "DO", "TURB",
                    "VEL", "TP", "TN", "SS", "CHLcal"])

ax.set_ylim(-5, 20)
ax.get_yaxis().set_visible(False)

#ax.set_yticklables([-5, 0, 5, 10, 15, 20])

plt.title("Distribution of standardized variables")


# show plot
plt.show(bp)
```



In [6]:

```
Z = water20[["WDP", "SECCHI", "TEMP", "DO", "TURB",
             "VEL", "TP", "TN", "SS", "CHLcal"]]

Z = RobustScaler().fit_transform(Z)

Z = pd.DataFrame(Z, columns = ["WDP", "SECCHI", "TEMP", "DO", "TURB",
                              "VEL", "TP", "TN", "SS", "CHLcal"])

Z.head()
```

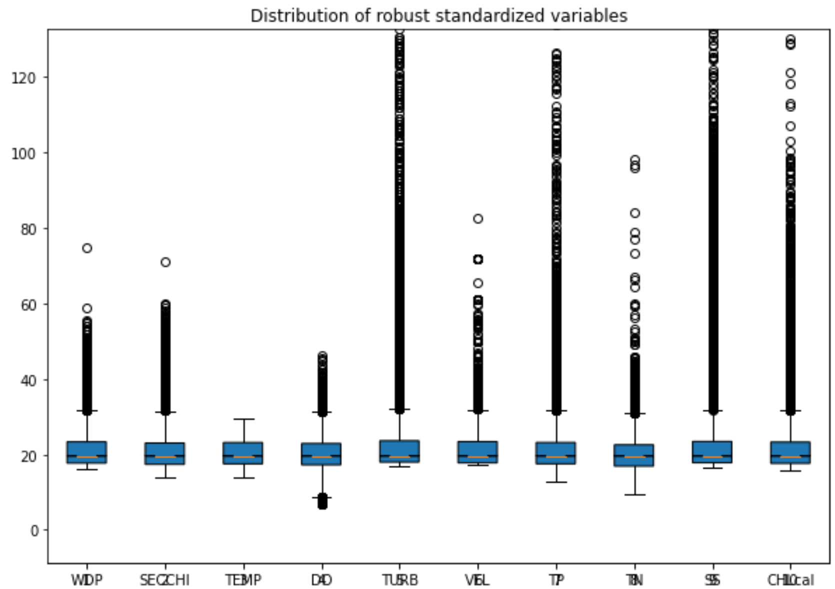
Out[6]:

	WDP	SECCHI	TEMP	DO	TURB	VEL	TP	TN	SS	CHLcal
0	-0.018088	-0.046512	0.572414	-0.720930	0.210526	0.576923	0.481481	0.918033	0.361752	-0.294693
1	1.532300	0.000000	0.572414	-0.720930	0.210526	1.000000	0.488889	1.547131	0.262323	-0.342123
2	0.524548	0.023256	0.565517	-0.790698	0.105263	0.884615	0.422222	0.918033	0.188280	-0.323151
3	1.764858	-0.093023	0.565517	-0.767442	0.210526	0.942308	0.362963	1.124317	0.173472	-0.332637
4	1.144703	0.069767	0.572414	-0.720930	0.342105	0.923077	0.548148	0.969262	0.482336	-0.291531

In [7]:

```
fig = plt.figure(figsize =(10, 7))

# Creating plot
plt.boxplot(Z)
ax = fig.add_subplot(111)

bp = ax.boxplot(Z, patch_artist = True,
                notch ='True', vert = 1)


ax.set_xticklabels(["WDP", "SECCHI", "TEMP", "DO", "TURB",
                    "VEL", "TP", "TN", "SS", "CHLcal"])

ax.set_ylim(-5, 20)
ax.get_yaxis().set_visible(False)
```

```
#ax.set_yticklables([-5, 0, 5, 10, 15, 20])

plt.title("Distribution of robust standardized variables")


# show plot
plt.show(bp)
```



Define functions

In [35]:
```
def mapper_pca2_db(df, DBSCAN_EPSILON = 10, DBSCAN_MIN_SAMPLES = 20,
                N_CUBES = [10,10], PERC_OVERLAP = [.25,.25], return_with_df = False):
    """
    """

    X = df[["WDP", "SECCHI", "TEMP", "DO", "TURB",
            "VEL", "TP", "TN", "SS", "CHLcal"]]

    # for discerning primary variables in PCA
    continuous_variables = ["WDP", "SECCHI", "TEMP", "DO", "TURB",
                            "VEL", "TP", "TN", "SS", "CHLcal"]

    var_to_index = {continuous_variables[i] : i for i in range(len(continuous_variables))}
    projected_vars = continuous_variables
    projected_var_indices = [var_to_index[var] for var in projected_vars]

#     if X.shape[0]<10:
#         #print(X)
#         print("Not enough data in ", title, "_size = ", X.shape[0])
#         return(X.shape[0])

    # to match up indices in scomplex with the original dataframe X
    X.reset_index(drop = True, inplace = True)

    # create instance of clustering alg
```

```
    cluster_alg = DBSCAN(eps = DBSCAN_EPSILON, min_samples = DBSCAN_MIN_SAMPLES,
                         metric='euclidean')

    # instantiate kepler mapper object
    mapper = km.KeplerMapper(verbose = 0)

    # defining filter function as projection on to the first 2 component axis
    pca = PCA(n_components = 2)
    lens = pca.fit_transform(X)

#     for j in range(2):
#         pc_j = pca.components_[j]
#         largest_magnitude = max(abs(pc_j))
#         idx_magnitude = np.where(abs(pc_j) == largest_magnitude)[0][0]

#         print("*** PCA", j+1, " ***")
#         print("Primary variable: ", continuous_variables[idx_magnitude])
#         print("Corresponding component: ", pc_j[idx_magnitude])
#         print("Explained variance: ", pca.explained_variance_ratio_[j])

    summary_variable = mapper.project(np.array(X), projection=projected_var_indices, scaler=None)
    # similar to fit transform

    # Generate the simplicial complex
    scomplex = mapper.map(lens, X,
                          cover=km.Cover(n_cubes = N_CUBES, perc_overlap = PERC_OVERLAP),
                          clusterer = cluster_alg)

    if return_with_df:
        return(scomplex, X)

    return(scomplex)
```

In [36]:

```
def mapper_pca2_hdb(df, HDB_MIN_CLUSTER = 45, HDB_MIN_SAMPLES = 10, HDB_EPSILON = 1,
                    N_CUBES = [10,10], PERC_OVERLAP = [.25,.25], return_with_df = False):
    """
    """

    X = df[["WDP", "SECCHI", "TEMP", "DO", "TURB",
            "VEL", "TP", "TN", "SS", "CHLcal"]]

    # for discerning primary variables in PCA
    continuous_variables = ["WDP", "SECCHI", "TEMP", "DO", "TURB",
                            "VEL", "TP", "TN", "SS", "CHLcal"]

    var_to_index = {continuous_variables[i] : i for i in range(len(continuous_variables))}
    projected_vars = continuous_variables
    projected_var_indices = [var_to_index[var] for var in projected_vars]

#     if X.shape[0]<10:
#         #print(X)
#         print("Not enough data in ", title, "_size = ", X.shape[0])
#         return(X.shape[0])

    # to match up indices in scomplex with the original dataframe X
    X.reset_index(drop = True, inplace = True)

    # create instance of clustering alg
    cluster_alg = hdbscan.HDBSCAN(min_cluster_size = HDB_MIN_CLUSTER, min_samples = HDB_MIN_SAMPLES,
                                  cluster_selection_epsilon= HDB_EPSILON, cluster_selection_method = 'eom')

    # instantiate kepler mapper object
    mapper = km.KeplerMapper(verbose = 0)
```

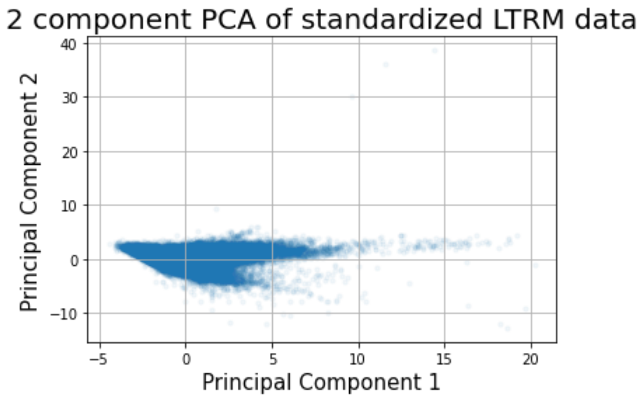
```
    # defining filter function as projection on to the first 2 component axis
    pca = PCA(n_components = 2)
    lens = pca.fit_transform(X)

#     for j in range(2):
#         pc_j = pca.components_[j]
#         largest_magnitude = max(abs(pc_j))
#         idx_magnitude = np.where(abs(pc_j) == largest_magnitude)[0][0]

#         print("*** PCA", j+1, " ***")
#         print("Primary variable: ", continuous_variables[idx_magnitude])
#         print("Corresponding component: ", pc_j[idx_magnitude])
#         print("Explained variance: ", pca.explained_variance_ratio_[j])

    summary_variable = mapper.project(np.array(X), projection=projected_var_indices, scaler=None)
    # similar to fit transform

    # Generate the simplicial complex
    scomplex = mapper.map(lens, X,
                          cover=km.Cover(n_cubes = N_CUBES, perc_overlap = PERC_OVERLAP),
                          clusterer = cluster_alg)

    if return_with_df:
        return(scomplex, X)

    return(scomplex)
```

PCA2 projection

In [10]:
```
pca = PCA(n_components = 2)
lens = pca.fit_transform(X)

lens[:,0]
```

Out[10]:
```
array([ 0.57250455,  1.23518396,  0.76338673, ..., -0.19200403,
       -0.13443336, -0.20681377])
```

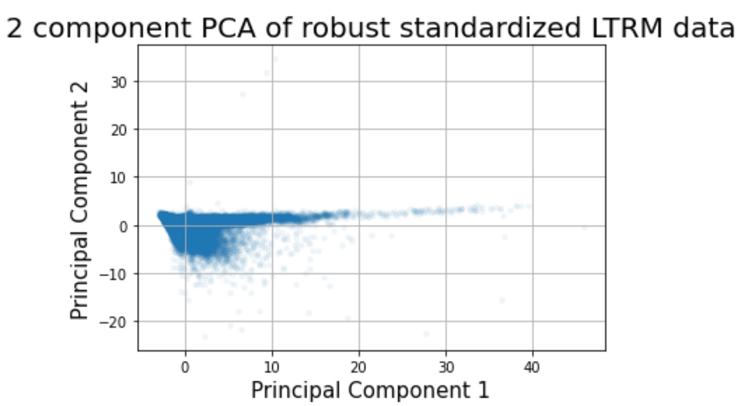
In [11]:
```
fig, ax = plt.subplots()
ax.scatter(lens[:,0], lens[:,1], s = 10, alpha = 0.047)
ax.set_xlabel('Principal Component 1', fontsize = 15)
ax.set_ylabel('Principal Component 2', fontsize = 15)
ax.set_title('2 component PCA of standardized LTRM data', fontsize = 20)
ax.grid(True)

plt.show()
```



In [12]:
```
pca = PCA(n_components = 2)
lens = pca.fit_transform(Z)

lens[:,0]
```

Out[12]:
```
array([-0.11138297,  0.10533746, -0.20813589, ..., -0.07760448,
        0.05332483, -0.08866132])
```

In [13]:
```
fig, ax = plt.subplots()
ax.scatter(lens[:,0], lens[:,1], s = 10, alpha = 0.047)
ax.set_xlabel('Principal Component 1', fontsize = 15)
ax.set_ylabel('Principal Component 2', fontsize = 15)
ax.set_title('2 component PCA of robust standardized LTRM data', fontsize = 20)
ax.grid(True)

plt.show()
```



Incorporating the clustering algorithms with km

Setting parameter options for clustering

TDA parameters

In [14]:

```python
n_cubes_lst = []

for n in [75, 100]:
    n_cubes_lst.append([n, n])

perc_overlap_lst = []

for perc in [0.5]:
    perc = round(perc, 2)
    perc_overlap_lst.append([perc, perc])

# tda params
print(n_cubes_lst)
print(perc_overlap_lst)
```

```
[[75, 75], [100, 100]]
[[0.5, 0.5]]
```

Running DBScan and TDA

In [15]:

```python
# dbscan parameters

eps_lst = np.linspace(0.75, 1, 2) # variables are standardized
min_samples_lst = [10]
print(eps_lst)
print(min_samples_lst)
```

```
[0.75 1.  ]
[10]
```

In [16]:

```python
db_params = []
db_scomplex = []

for epsilon in eps_lst:

    for min_samples in min_samples_lst: # min_samples = 10

        for n_cubes in n_cubes_lst:

            for perc in perc_overlap_lst: # perc_overlap = [0.5, 0.5]

                db_params.append('epsilon = ' + str(epsilon) + ', min_samples = ' + str(min_samples) +
                                 ', n_cubes = ' + str(n_cubes) + ", and perc_overlap = " + str(perc))

                db_scomplex.append(mapper_pca2_db(X, DBSCAN_EPSILON = epsilon, DBSCAN_MIN_SAMPLES = min_samples,
                                                  N_CUBES = n_cubes, PERC_OVERLAP = perc, return_with_df = False))

                idx = len(db_params)-1

                print("*** ", idx, " ***")
                print(db_params[idx])

                all_nodes = db_scomplex[idx].get('nodes')

                obsv_per_node = []

                for node in all_nodes:
                    obsv_per_node.append(len(all_nodes.get(node)))

                print("The maximum data points in a node is ", max(obsv_per_node))
                print("The minimum data points in a node is ", min(obsv_per_node))
                print("The number of unique samples is ", get_mapper_graph(db_scomplex[idx])[1]["n_unique"])
```

```
***  0  ***
epsilon = 0.75, min_samples = 10, n_cubes = [75, 75], and perc_overlap = [0.5, 0.5]
The maximum data points in a node is  7090
The minimum data points in a node is  3
The number of unique samples is  68938
***  1  ***
epsilon = 0.75, min_samples = 10, n_cubes = [100, 100], and perc_overlap = [0.5, 0.5]
The maximum data points in a node is  4190
The minimum data points in a node is  4
The number of unique samples is  68036
***  2  ***
epsilon = 1.0, min_samples = 10, n_cubes = [75, 75], and perc_overlap = [0.5, 0.5]
The maximum data points in a node is  7374
The minimum data points in a node is  4
The number of unique samples is  73521
***  3  ***
epsilon = 1.0, min_samples = 10, n_cubes = [100, 100], and perc_overlap = [0.5, 0.5]
The maximum data points in a node is  4387
The minimum data points in a node is  4
The number of unique samples is  72891
```

In [17]:

```python
# dbscan with robust scaling

db_params_rob = []
db_scomplex_rob = []

for epsilon in eps_lst:

    for min_samples in min_samples_lst: # min_samp = 10

        for n_cubes in n_cubes_lst:

            for perc in perc_overlap_lst: # perc_overlap = [0.5, 0.5]

                db_params_rob.append('epsilon = ' + str(epsilon) + ', min_samples = ' + str(min_samples) +
                                 ', n_cubes = ' + str(n_cubes) + ", and perc_overlap = " + str(perc))

                db_scomplex_rob.append(mapper_pca2_db(Z, DBSCAN_EPSILON = epsilon, DBSCAN_MIN_SAMPLES = min_samples,
                                                  N_CUBES = n_cubes, PERC_OVERLAP = perc, return_with_df = False))

                idx = len(db_params_rob)-1

                print("*** ", idx, " ***")
                print(db_params_rob[idx])

                all_nodes = db_scomplex_rob[idx].get('nodes')

                obsv_per_node = []

                for node in all_nodes:
                    obsv_per_node.append(len(all_nodes.get(node)))

                print("The maximum data points in a node is ", max(obsv_per_node))
                print("The minimum data points in a node is ", min(obsv_per_node))
                print("The number of unique samples is ", get_mapper_graph(db_scomplex[idx])[1]["n_unique"])
```

```
***  0  ***
epsilon = 0.75, min_samples = 10, n_cubes = [75, 75], and perc_overlap = [0.5, 0.5]
The maximum data points in a node is  22399
The minimum data points in a node is  6
The number of unique samples is  68938
***  1  ***
epsilon = 0.75, min_samples = 10, n_cubes = [100, 100], and perc_overlap = [0.5, 0.5]
The maximum data points in a node is  14619
The minimum data points in a node is  4
The number of unique samples is  68036
```

```
***  2  ***
epsilon = 1.0, min_samples = 10, n_cubes = [75, 75], and perc_overlap = [0.5, 0.5]
The maximum data points in a node is  22627
The minimum data points in a node is  5
The number of unique samples is  73521
***  3  ***
epsilon = 1.0, min_samples = 10, n_cubes = [100, 100], and perc_overlap = [0.5, 0.5]
The maximum data points in a node is  14788
The minimum data points in a node is  3
The number of unique samples is  72891
```

Selected parameters for DBscan, TDA mapper

Scaling: standard scaling because robust scaling causes 1. lost of data through noise and 2. nodes with many data points

Epsilon = 1

min_samples = 10

n_cubes = [75, 75]

perc_overlap = [0.5, 0.5]

In [21]:

```
plotlyviz(db_scomplex[2], title = db_params[2],
          graph_layout='fr', dashboard = True)
# CHOSEN ONE
```

HDBScan and TDA

Use the cluster selection epsilon method so that it is a hybrid of DBscan and HDBscan.

In [18]:

```
# hdbscan

min_cluster_lst = np.linspace(10, 15, 2)
print(min_cluster_lst)
print(min_samples_lst)
```

```
[10. 15.]
[10]
```

In [19]:

```
# hdbscan

hdb_params = []
hdb_scomplex = []

for min_clust in min_cluster_lst: # min_clust = 10

    min_clust = int(min_clust)

    for min_samples in min_samples_lst: # min_samples = 10

        min_samples = int(min_samples)

        for n_cubes in n_cubes_lst:

            for perc in perc_overlap_lst: # perc_overlap = [0.5, 0.5]

                hdb_params.append('min_clust = ' + str(min_clust) + ', min_samples = ' + str(min_samples) +
                                  ', n_cubes = ' + str(n_cubes) + ", and perc_overlap = " + str(perc))
                hdb_scomplex.append(mapper_pca2_hdb(X, HDB_MIN_CLUSTER = min_clust, HDB_MIN_SAMPLES = min_samples,
```

```
                                                    HDB_EPSILON = 1,N_CUBES = n_cubes, PERC_OVERLAP = perc,
                                                    return_with_df = False))

                idx = len(hdb_params)-1

                print("*** ", idx, " ***")
                print(hdb_params[idx])

                all_nodes = hdb_scomplex[idx].get('nodes')

                obsv_per_node = []

                for node in all_nodes:
                    obsv_per_node.append(len(all_nodes.get(node)))

                print("The maximum data points in a node is ", max(obsv_per_node))
                print("The minimum data points in a node is ", min(obsv_per_node))
                print("The number of unique samples is ", get_mapper_graph(hdb_scomplex[idx])[1]["n_unique"])
```

```
***  0  ***
min_clust = 10, min_samples = 10, n_cubes = [75, 75], and perc_overlap = [0.5, 0.5]
The maximum data points in a node is  6980
The minimum data points in a node is  10
The number of unique samples is  69244
***  1  ***
min_clust = 10, min_samples = 10, n_cubes = [100, 100], and perc_overlap = [0.5, 0.5]
The maximum data points in a node is  4148
The minimum data points in a node is  10
The number of unique samples is  66795
***  2  ***
min_clust = 15, min_samples = 10, n_cubes = [75, 75], and perc_overlap = [0.5, 0.5]
The maximum data points in a node is  6980
The minimum data points in a node is  15
The number of unique samples is  66696
***  3  ***
min_clust = 15, min_samples = 10, n_cubes = [100, 100], and perc_overlap = [0.5, 0.5]
The maximum data points in a node is  4148
The minimum data points in a node is  15
The number of unique samples is  63468
```

In [20]:

```
# hdbscan with robust scaling

hdb_params_rob = []
hdb_scomplex_rob = []

for min_clust in min_cluster_lst: # min_clust = 10

    min_clust = int(min_clust)

    for min_samples in min_samples_lst: # min_samp = 10

        min_samples = int(min_samples)

        for n_cubes in n_cubes_lst:

            for perc in perc_overlap_lst: # perc_overlap = [0.5, 0.5]

                hdb_params_rob.append('min_clust = ' + str(min_clust) + ', min_samples = ' + str(min_samples) +
                                 ', n_cubes = ' + str(n_cubes) + ", and perc_overlap = " + str(perc))
                hdb_scomplex_rob.append(mapper_pca2_hdb(Z, HDB_MIN_CLUSTER = min_clust, HDB_MIN_SAMPLES = min_samples,
                                                        HDB_EPSILON = 1,N_CUBES = n_cubes, PERC_OVERLAP = perc,
                                                        return_with_df = False))

                idx = len(hdb_params_rob)-1

                print("*** ", idx, " ***")
```

```
                print(hdb_params_rob[idx])

                all_nodes = hdb_scomplex_rob[idx].get('nodes')

                obsv_per_node = []

                for node in all_nodes:
                    obsv_per_node.append(len(all_nodes.get(node)))

                print("The maximum data points in a node is ", max(obsv_per_node))
                print("The minimum data points in a node is ", min(obsv_per_node))
                print("The number of unique samples is ", get_mapper_graph(hdb_scomplex[idx])[1]["n_unique"])
```

```
***  0  ***
min_clust = 10, min_samples = 10, n_cubes = [75, 75], and perc_overlap = [0.5, 0.5]
The maximum data points in a node is  17667
The minimum data points in a node is  10
The number of unique samples is  69244
***  1  ***
min_clust = 10, min_samples = 10, n_cubes = [100, 100], and perc_overlap = [0.5, 0.5]
The maximum data points in a node is  12639
The minimum data points in a node is  10
The number of unique samples is  66795
***  2  ***
min_clust = 15, min_samples = 10, n_cubes = [75, 75], and perc_overlap = [0.5, 0.5]
The maximum data points in a node is  16588
The minimum data points in a node is  15
The number of unique samples is  66696
***  3  ***
min_clust = 15, min_samples = 10, n_cubes = [100, 100], and perc_overlap = [0.5, 0.5]
The maximum data points in a node is  11144
The minimum data points in a node is  15
The number of unique samples is  63468
```

Selected parameters for HDBscan, TDA mapper

Scaling: standardized scaling, for the same reason as above

Epsilon = 1

min_cluster_size = 10

min_samples = 10

n_cubes = [75, 75]

perc_overlap = [0.5, 0.5]

In [22]:

```
plotlyviz(hdb_scomplex[0], title = hdb_params[0],
          graph_layout='fr', dashboard = True)
# chosen one
```

In [23]:

```
plotlyviz(hdb_scomplex[1], title = hdb_params[1],
          graph_layout='fr', dashboard = True) #alternative
```

Save selected parameters into .html and .json

In [24]:

```
mapper = km.KeplerMapper(verbose=0)
```