import xml.etree.ElementTree as et  
import tkinter.filedialog  
import numpy as np  
import pandas as pd  
  
import scipy.cluster  
from scipy.cluster.hierarchy import dendrogram  
from scipy.cluster.hierarchy import ClusterWarning  
from scipy.spatial import distance\_matrix  
from scipy.cluster import hierarchy  
  
  
  
from sklearn import preprocessing  
from sklearn.metrics import silhouette\_score  
from sklearn.metrics import silhouette\_samples  
from sklearn.cluster import AgglomerativeClustering  
from yellowbrick.cluster import SilhouetteVisualizer  
  
  
  
import gower  
import qgrid  
import scipy.cluster.hierarchy as sch  
  
  
  
  
import matplotlib.pyplot as plt  
from warnings import simplefilter  
  
  
  
simplefilter("ignore", ClusterWarning)  
  
  
label\_encoder = preprocessing.LabelEncoder()  
scalar = preprocessing.MinMaxScaler()

"""Dostać się do korzeny drzewa"""  
def get\_xml\_root():  
 etree = et.parse(tkinter.filedialog.askopenfilename())  
 return etree.getroot()

"""Dostać się do atributów tabeli decizinej"""  
def get\_attributes(xml\_root):  
 attribute\_attr\_list = [a.attrib for a in xml\_root.find('attributes').iter('name')]  
 attribute\_attr\_dict = attribute\_attr\_list.copy()  
 attributes\_attr\_data\_frame = pd.DataFrame(list(attribute\_attr\_dict))  
 attributes\_attr\_data\_frame.drop('valueID', axis='columns', inplace=True)  
 attributes\_attr\_data\_frame.dropna(subset=['attributeID'], inplace=True)  
 return attributes\_attr\_data\_frame

def init\_rule\_matrix(rule\_data\_frame, attribute\_data\_frame):  
 for x in attribute\_data\_frame.values:  
 rule\_data\_frame['attribute\_' + x] = np.nan  
 return rule\_data\_frame

def get\_rule\_by\_id(xml\_root, id\_rule):  
 for rule in xml\_root.iter('rule'):  
 if rule.attrib['ruleID'] == id\_rule:  
 return rule

def populate\_matrix\_with\_conditions(xml\_root, data\_frame):  
 for i in range(len(list(data\_frame.ruleID))):  
 for k in list(get\_rule\_by\_id(xml\_root, data\_frame.ruleID[i]).find('conditions')):  
 data\_frame.loc[  
 data\_frame['ruleID'] == data\_frame.ruleID[i], str(  
 'attribute\_' + list(k)[0].attrib['attributeID'])] = list(k)[2].text  
 return data\_frame

def populate\_matrix\_with\_conclusions(xml\_root, data\_frame):  
 for i in range(len(list(data\_frame.ruleID))):  
 for k in list(get\_rule\_by\_id(xml\_root, data\_frame.ruleID[i]).find('conclusion')):  
 data\_frame.loc[  
 data\_frame['ruleID'] == data\_frame.ruleID[i], str(  
 'attribute\_' + list(k)[0].attrib['attributeID'])] = list(k)[2].text  
 return data\_frame

def normilize\_types\_of\_column\_values(data\_frame, column):  
 try:  
 data\_frame[column] = pd.to\_numeric(data\_frame[column], downcast="float")  
 except:  
 data\_frame[column] = data\_frame[column].astype(str)

def get\_simbolic\_values(xml\_root):  
 symbolic\_value = []  
 t = [a for a in xml\_root.find('attributes').iter('symbolic\_value')]  
 for x in range(len(list(t))):  
 symbolic\_value.append(list(t)[x].find('name').text)  
 return set(symbolic\_value)

def plot\_dendrogram(model, \*\*kwargs):  
 counts = np.zeros(model.children\_.shape[0])  
 n\_samples = len(model.labels\_)  
 for i, merge in enumerate(model.children\_):  
 current\_count = 0  
 for child\_idx in merge:  
 if child\_idx < n\_samples:  
 current\_count += 1  
 else:  
 current\_count += counts[child\_idx - n\_samples]  
 counts[i] = current\_count  
  
 linkage\_matrix = np.column\_stack([model.children\_, model.distances\_,  
 counts]).astype(float)  
 dendrogram(linkage\_matrix, \*\*kwargs)

if \_\_name\_\_ == '\_\_main\_\_':  
 root = get\_xml\_root()  
 symbolic\_value = list(get\_simbolic\_values(root))  
 rule\_attrs = get\_rules(root)  
 print(rule\_attrs)  
 attribute\_attrs = get\_attributes(root)  
 print(attribute\_attrs)  
 init\_data = init\_rule\_matrix(rule\_attrs, attribute\_attrs)  
 print(init\_data)  
 populate\_matrix\_with\_conditions(root, init\_data)

number ruleID  
0 1 1878  
1 2 1879  
2 3 1880  
3 4 1881  
4 5 1882  
.. ... ...  
411 412 2289  
412 413 2290  
413 414 2291  
414 415 2292  
415 416 2293  
  
[416 rows x 2 columns]  
 attributeID  
0 426  
1 427  
2 428  
3 429  
4 430  
5 431  
6 432  
7 433  
8 434  
9 435  
10 436  
11 437  
12 438  
21 439  
28 440  
36 441  
44 442  
53 443  
54 444  
63 445  
72 446  
81 447  
90 448  
99 449  
100 450  
101 451  
102 452  
103 453  
104 454  
105 455  
114 456  
123 457  
132 458  
141 459  
 number ruleID attribute\_426 attribute\_427 attribute\_428 attribute\_429 \  
0 1 1878 NaN NaN NaN NaN   
1 2 1879 NaN NaN NaN NaN   
2 3 1880 NaN NaN NaN NaN   
3 4 1881 NaN NaN NaN NaN   
4 5 1882 NaN NaN NaN NaN   
.. ... ... ... ... ... ...   
411 412 2289 NaN NaN NaN NaN   
412 413 2290 NaN NaN NaN NaN   
413 414 2291 NaN NaN NaN NaN   
414 415 2292 NaN NaN NaN NaN   
415 416 2293 NaN NaN NaN NaN   
  
 attribute\_430 attribute\_431 attribute\_432 attribute\_433 ... \  
0 NaN NaN NaN NaN ...   
1 NaN NaN NaN NaN ...   
2 NaN NaN NaN NaN ...   
3 NaN NaN NaN NaN ...   
4 NaN NaN NaN NaN ...   
.. ... ... ... ... ...   
411 NaN NaN NaN NaN ...   
412 NaN NaN NaN NaN ...   
413 NaN NaN NaN NaN ...   
414 NaN NaN NaN NaN ...   
415 NaN NaN NaN NaN ...   
  
 attribute\_450 attribute\_451 attribute\_452 attribute\_453 \  
0 NaN NaN NaN NaN   
1 NaN NaN NaN NaN   
2 NaN NaN NaN NaN   
3 NaN NaN NaN NaN   
4 NaN NaN NaN NaN   
.. ... ... ... ...   
411 NaN NaN NaN NaN   
412 NaN NaN NaN NaN   
413 NaN NaN NaN NaN   
414 NaN NaN NaN NaN   
415 NaN NaN NaN NaN   
  
 attribute\_454 attribute\_455 attribute\_456 attribute\_457 \  
0 NaN NaN NaN NaN   
1 NaN NaN NaN NaN   
2 NaN NaN NaN NaN   
3 NaN NaN NaN NaN   
4 NaN NaN NaN NaN   
.. ... ... ... ...   
411 NaN NaN NaN NaN   
412 NaN NaN NaN NaN   
413 NaN NaN NaN NaN   
414 NaN NaN NaN NaN   
415 NaN NaN NaN NaN   
  
 attribute\_458 attribute\_459   
0 NaN NaN   
1 NaN NaN   
2 NaN NaN   
3 NaN NaN   
4 NaN NaN   
.. ... ...   
411 NaN NaN   
412 NaN NaN   
413 NaN NaN   
414 NaN NaN   
415 NaN NaN   
  
[416 rows x 36 columns]

data = populate\_matrix\_with\_conclusions(root, init\_data)

data.to\_excel(r'export\_dataframe.xlsx', index = False, header=True)

data.drop(data.columns[[0]], axis = 1, inplace = True)

data

ruleID attribute\_722 attribute\_723 attribute\_724 attribute\_725 \  
0 5650 a1 b1 NaN NaN   
1 5651 NaN b1 c1 NaN   
2 5652 NaN NaN c1 d1   
3 5653 NaN NaN NaN d1   
4 5654 NaN NaN NaN NaN   
5 5655 NaN NaN NaN NaN   
6 5656 NaN NaN NaN NaN   
7 5657 NaN NaN NaN NaN   
8 5658 NaN NaN NaN NaN   
9 5659 NaN NaN NaN NaN   
10 11654 NaN NaN NaN NaN   
  
 attribute\_726 attribute\_727 attribute\_728 attribute\_729 attribute\_730 \  
0 NaN NaN NaN NaN NaN   
1 NaN NaN NaN NaN NaN   
2 NaN NaN NaN NaN NaN   
3 e1 NaN NaN NaN NaN   
4 e1 f1 NaN NaN NaN   
5 NaN f1 g1 NaN NaN   
6 NaN NaN g1 h1 NaN   
7 NaN NaN g1 h1 i1   
8 NaN NaN NaN NaN i1   
9 NaN NaN NaN NaN NaN   
10 NaN NaN NaN NaN NaN   
  
 attribute\_731 attribute\_732 attribute\_733   
0 NaN NaN NaN   
1 NaN NaN NaN   
2 NaN NaN NaN   
3 NaN NaN NaN   
4 NaN NaN NaN   
5 NaN NaN NaN   
6 NaN NaN NaN   
7 NaN NaN NaN   
8 j1 NaN NaN   
9 j1 k1 NaN   
10 j1 NaN kk1

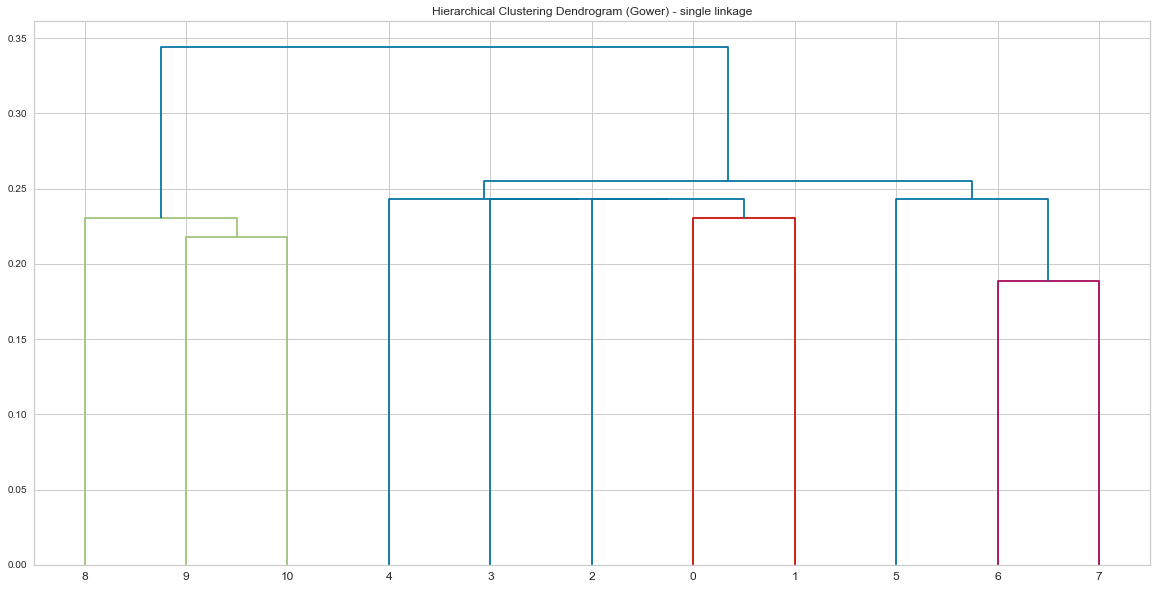
gower\_data = data

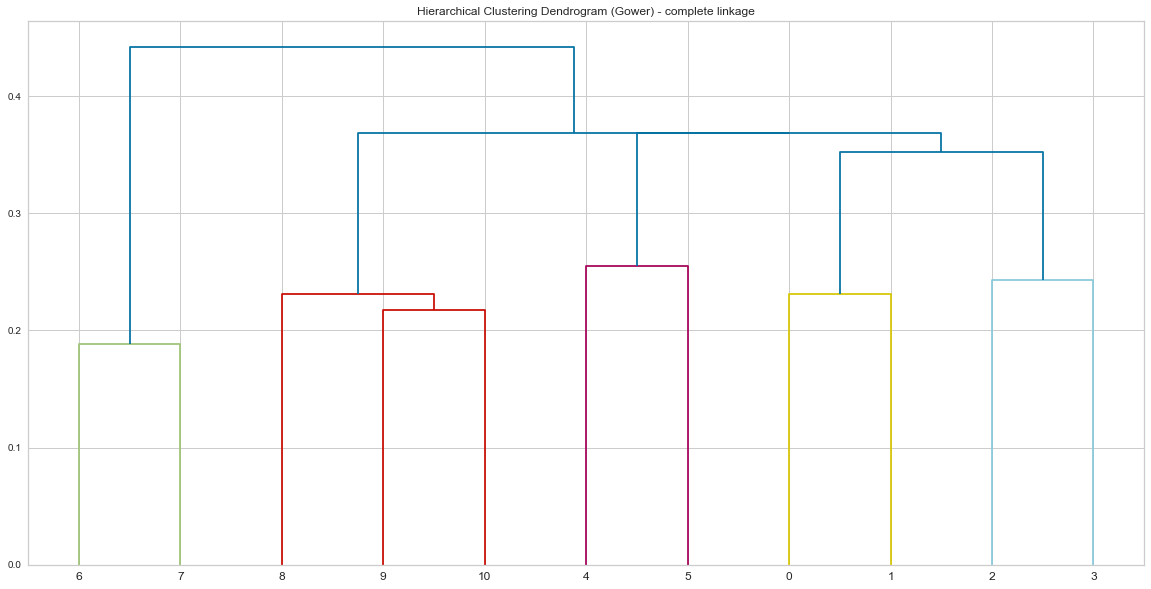
gower\_data.set\_index('ruleID')

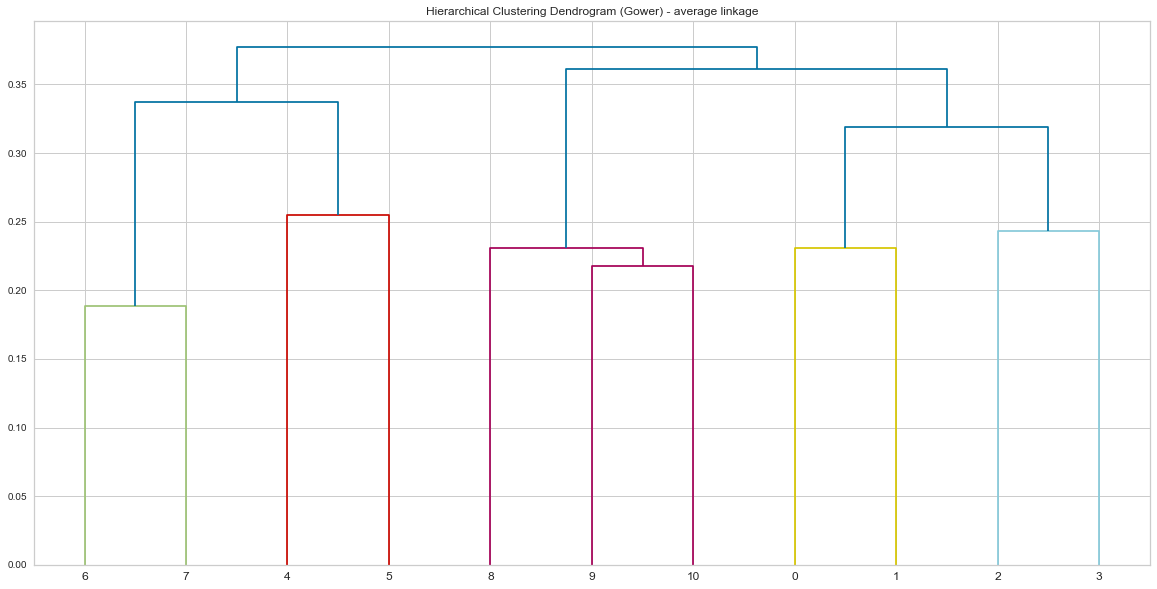
attribute\_722 attribute\_723 attribute\_724 attribute\_725 attribute\_726 \  
ruleID   
5650 a1 b1 NaN NaN NaN   
5651 NaN b1 c1 NaN NaN   
5652 NaN NaN c1 d1 NaN   
5653 NaN NaN NaN d1 e1   
5654 NaN NaN NaN NaN e1   
5655 NaN NaN NaN NaN NaN   
5656 NaN NaN NaN NaN NaN   
5657 NaN NaN NaN NaN NaN   
5658 NaN NaN NaN NaN NaN   
5659 NaN NaN NaN NaN NaN   
11654 NaN NaN NaN NaN NaN   
  
 attribute\_727 attribute\_728 attribute\_729 attribute\_730 attribute\_731 \  
ruleID   
5650 NaN NaN NaN NaN NaN   
5651 NaN NaN NaN NaN NaN   
5652 NaN NaN NaN NaN NaN   
5653 NaN NaN NaN NaN NaN   
5654 f1 NaN NaN NaN NaN   
5655 f1 g1 NaN NaN NaN   
5656 NaN g1 h1 NaN NaN   
5657 NaN g1 h1 i1 NaN   
5658 NaN NaN NaN i1 j1   
5659 NaN NaN NaN NaN j1   
11654 NaN NaN NaN NaN j1   
  
 attribute\_732 attribute\_733   
ruleID   
5650 NaN NaN   
5651 NaN NaN   
5652 NaN NaN   
5653 NaN NaN   
5654 NaN NaN   
5655 NaN NaN   
5656 NaN NaN   
5657 NaN NaN   
5658 NaN NaN   
5659 k1 NaN   
11654 NaN kk1

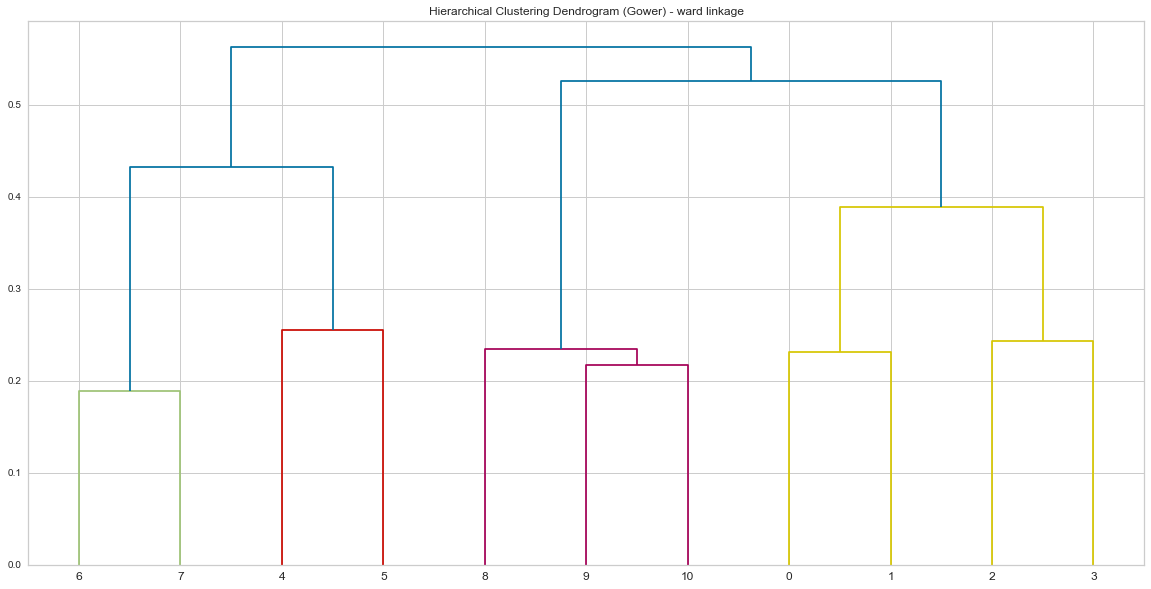
gdistance = gower.gower\_matrix(gower\_data)

for g\_linkage in ('single', 'complete', 'average', 'ward'):  
 clusters = scipy.cluster.hierarchy.linkage(gdistance, method=g\_linkage)  
 plt.figure(figsize=(20,10))  
 plt.title(f'Hierarchical Clustering Dendrogram (Gower) - {g\_linkage} linkage')  
 dn = hierarchy.dendrogram(clusters)









data.info()

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 17 entries, 0 to 16  
Data columns (total 13 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 ruleID 17 non-null object  
 1 attribute\_1220 6 non-null object  
 2 attribute\_1221 7 non-null object  
 3 attribute\_1222 6 non-null object  
 4 attribute\_1223 5 non-null object  
 5 attribute\_1224 2 non-null object  
 6 attribute\_1225 4 non-null object  
 7 attribute\_1226 2 non-null object  
 8 attribute\_1227 3 non-null object  
 9 attribute\_1228 3 non-null object  
 10 attribute\_1229 3 non-null object  
 11 attribute\_1230 2 non-null object  
 12 attribute\_1231 1 non-null object  
dtypes: object(13)  
memory usage: 1.9+ KB

normalized\_data = data

normalized\_data.set\_index('ruleID')

attribute\_1220 attribute\_1221 attribute\_1222 attribute\_1223 \  
ruleID   
11655 a1 b1 NaN NaN   
11656 a1 NaN NaN c1   
11657 NaN NaN NaN NaN   
11658 NaN b2 NaN NaN   
11659 NaN b1 NaN NaN   
11660 NaN NaN NaN c1   
11661 NaN NaN NaN NaN   
11662 NaN NaN NaN NaN   
11663 NaN NaN NaN NaN   
11664 NaN NaN NaN NaN   
11665 NaN NaN NaN NaN   
11666 NaN NaN j1 NaN   
11667 NaN NaN j1 NaN   
11668 a1 b1 j1 NaN   
11669 a1 b1 j1 c2   
11670 a1 b1 j1 c2   
11671 a1 b1 j1 c2   
  
 attribute\_1224 attribute\_1225 attribute\_1226 attribute\_1227 \  
ruleID   
11655 NaN NaN NaN NaN   
11656 NaN NaN NaN NaN   
11657 d1 e1 NaN NaN   
11658 d1 NaN f1 NaN   
11659 NaN NaN NaN g1   
11660 NaN NaN NaN g1   
11661 NaN e1 NaN NaN   
11662 NaN NaN f1 NaN   
11663 NaN NaN NaN g1   
11664 NaN NaN NaN NaN   
11665 NaN NaN NaN NaN   
11666 NaN NaN NaN NaN   
11667 NaN NaN NaN NaN   
11668 NaN NaN NaN NaN   
11669 NaN NaN NaN NaN   
11670 NaN e2 NaN NaN   
11671 NaN e2 NaN NaN   
  
 attribute\_1228 attribute\_1229 attribute\_1230 attribute\_1231   
ruleID   
11655 NaN NaN NaN NaN   
11656 NaN NaN NaN NaN   
11657 NaN NaN NaN NaN   
11658 NaN NaN NaN NaN   
11659 NaN NaN NaN NaN   
11660 NaN NaN NaN NaN   
11661 h1 NaN NaN NaN   
11662 h1 NaN NaN NaN   
11663 NaN i1 NaN NaN   
11664 NaN i1 k1 NaN   
11665 NaN i1 k1 NaN   
11666 h1 NaN NaN NaN   
11667 NaN NaN NaN l1   
11668 NaN NaN NaN NaN   
11669 NaN NaN NaN NaN   
11670 NaN NaN NaN NaN   
11671 NaN NaN NaN NaN

for column in data.columns:  
 normilize\_types\_of\_column\_values(normalized\_data, column)

normalized\_data

ruleID attribute\_1220 attribute\_1221 attribute\_1222 attribute\_1223 \  
0 11655.0 a1 b1 nan nan   
1 11656.0 a1 nan nan c1   
2 11657.0 nan nan nan nan   
3 11658.0 nan b2 nan nan   
4 11659.0 nan b1 nan nan   
5 11660.0 nan nan nan c1   
6 11661.0 nan nan nan nan   
7 11662.0 nan nan nan nan   
8 11663.0 nan nan nan nan   
9 11664.0 nan nan nan nan   
10 11665.0 nan nan nan nan   
11 11666.0 nan nan j1 nan   
12 11667.0 nan nan j1 nan   
13 11668.0 a1 b1 j1 nan   
14 11669.0 a1 b1 j1 c2   
15 11670.0 a1 b1 j1 c2   
16 11671.0 a1 b1 j1 c2   
  
 attribute\_1224 attribute\_1225 attribute\_1226 attribute\_1227 attribute\_1228 \  
0 nan nan nan nan nan   
1 nan nan nan nan nan   
2 d1 e1 nan nan nan   
3 d1 nan f1 nan nan   
4 nan nan nan g1 nan   
5 nan nan nan g1 nan   
6 nan e1 nan nan h1   
7 nan nan f1 nan h1   
8 nan nan nan g1 nan   
9 nan nan nan nan nan   
10 nan nan nan nan nan   
11 nan nan nan nan h1   
12 nan nan nan nan nan   
13 nan nan nan nan nan   
14 nan nan nan nan nan   
15 nan e2 nan nan nan   
16 nan e2 nan nan nan   
  
 attribute\_1229 attribute\_1230 attribute\_1231   
0 nan nan nan   
1 nan nan nan   
2 nan nan nan   
3 nan nan nan   
4 nan nan nan   
5 nan nan nan   
6 nan nan nan   
7 nan nan nan   
8 i1 nan nan   
9 i1 k1 nan   
10 i1 k1 nan   
11 nan nan nan   
12 nan nan l1   
13 nan nan nan   
14 nan nan nan   
15 nan nan nan   
16 nan nan nan

# encode\_categorical\_values(data)  
columns = list(data.columns)

columns

['ruleID',  
 'attribute\_1220',  
 'attribute\_1221',  
 'attribute\_1222',  
 'attribute\_1223',  
 'attribute\_1224',  
 'attribute\_1225',  
 'attribute\_1226',  
 'attribute\_1227',  
 'attribute\_1228',  
 'attribute\_1229',  
 'attribute\_1230',  
 'attribute\_1231']

numerical\_columns = []  
non\_numerocal\_columns = []  
for col in columns:  
 if np.issubdtype(data[col].dtype, np.number):  
 numerical\_columns.append(col)  
 else:  
 non\_numerocal\_columns.append(col)

# data[numerical\_columns] = data[numerical\_columns].fillna(float(0))

numerical\_data = data[numerical\_columns]

numerical\_data = numerical\_data.fillna(float(0))

numerical\_data.set\_index('ruleID')

Empty DataFrame  
Columns: []  
Index: [11655.0, 11656.0, 11657.0, 11658.0, 11659.0, 11660.0, 11661.0, 11662.0, 11663.0, 11664.0, 11665.0, 11666.0, 11667.0, 11668.0, 11669.0, 11670.0, 11671.0]

scalar.fit(numerical\_data)

MinMaxScaler()

x = scalar.transform(numerical\_data)

normalized\_data[numerical\_columns] = pd.DataFrame(x)

normalized\_data

ruleID attribute\_1220 attribute\_1221 attribute\_1222 attribute\_1223 \  
0 0.0000 a1 b1 nan nan   
1 0.0625 a1 nan nan c1   
2 0.1250 nan nan nan nan   
3 0.1875 nan b2 nan nan   
4 0.2500 nan b1 nan nan   
5 0.3125 nan nan nan c1   
6 0.3750 nan nan nan nan   
7 0.4375 nan nan nan nan   
8 0.5000 nan nan nan nan   
9 0.5625 nan nan nan nan   
10 0.6250 nan nan nan nan   
11 0.6875 nan nan j1 nan   
12 0.7500 nan nan j1 nan   
13 0.8125 a1 b1 j1 nan   
14 0.8750 a1 b1 j1 c2   
15 0.9375 a1 b1 j1 c2   
16 1.0000 a1 b1 j1 c2   
  
 attribute\_1224 attribute\_1225 attribute\_1226 attribute\_1227 attribute\_1228 \  
0 nan nan nan nan nan   
1 nan nan nan nan nan   
2 d1 e1 nan nan nan   
3 d1 nan f1 nan nan   
4 nan nan nan g1 nan   
5 nan nan nan g1 nan   
6 nan e1 nan nan h1   
7 nan nan f1 nan h1   
8 nan nan nan g1 nan   
9 nan nan nan nan nan   
10 nan nan nan nan nan   
11 nan nan nan nan h1   
12 nan nan nan nan nan   
13 nan nan nan nan nan   
14 nan nan nan nan nan   
15 nan e2 nan nan nan   
16 nan e2 nan nan nan   
  
 attribute\_1229 attribute\_1230 attribute\_1231   
0 nan nan nan   
1 nan nan nan   
2 nan nan nan   
3 nan nan nan   
4 nan nan nan   
5 nan nan nan   
6 nan nan nan   
7 nan nan nan   
8 i1 nan nan   
9 i1 k1 nan   
10 i1 k1 nan   
11 nan nan nan   
12 nan nan l1   
13 nan nan nan   
14 nan nan nan   
15 nan nan nan   
16 nan nan nan

for s in symbolic\_value:  
 for categorical in non\_numerocal\_columns:  
 data.loc[data[categorical] == s, categorical] = symbolic\_value.index(s)

for categorical in non\_numerocal\_columns:  
 data.loc[data[categorical] == 'nan', categorical] = 8

normalized\_data

ruleID attribute\_1220 attribute\_1221 attribute\_1222 attribute\_1223 \  
0 0.0000 13 6 8 8   
1 0.0625 13 8 8 12   
2 0.1250 8 8 8 8   
3 0.1875 8 1 8 8   
4 0.2500 8 6 8 8   
5 0.3125 8 8 8 12   
6 0.3750 8 8 8 8   
7 0.4375 8 8 8 8   
8 0.5000 8 8 8 8   
9 0.5625 8 8 8 8   
10 0.6250 8 8 8 8   
11 0.6875 8 8 5 8   
12 0.7500 8 8 5 8   
13 0.8125 13 6 5 8   
14 0.8750 13 6 5 3   
15 0.9375 13 6 5 3   
16 1.0000 13 6 5 3   
  
 attribute\_1224 attribute\_1225 attribute\_1226 attribute\_1227 attribute\_1228 \  
0 8 8 8 8 8   
1 8 8 8 8 8   
2 7 4 8 8 8   
3 7 8 14 8 8   
4 8 8 8 2 8   
5 8 8 8 2 8   
6 8 4 8 8 11   
7 8 8 14 8 11   
8 8 8 8 2 8   
9 8 8 8 8 8   
10 8 8 8 8 8   
11 8 8 8 8 11   
12 8 8 8 8 8   
13 8 8 8 8 8   
14 8 8 8 8 8   
15 8 15 8 8 8   
16 8 15 8 8 8   
  
 attribute\_1229 attribute\_1230 attribute\_1231   
0 8 8 8   
1 8 8 8   
2 8 8 8   
3 8 8 8   
4 8 8 8   
5 8 8 8   
6 8 8 8   
7 8 8 8   
8 10 8 8   
9 10 8 8   
10 10 8 8   
11 8 8 8   
12 8 8 9   
13 8 8 8   
14 8 8 8   
15 8 8 8   
16 8 8 8

indexed\_data = data.set\_index(data.ruleID)

distance\_mat = pd.DataFrame(distance\_matrix(indexed\_data.to\_numpy(), indexed\_data.to\_numpy()), index=indexed\_data.index, columns=indexed\_data.index)

distance\_mat

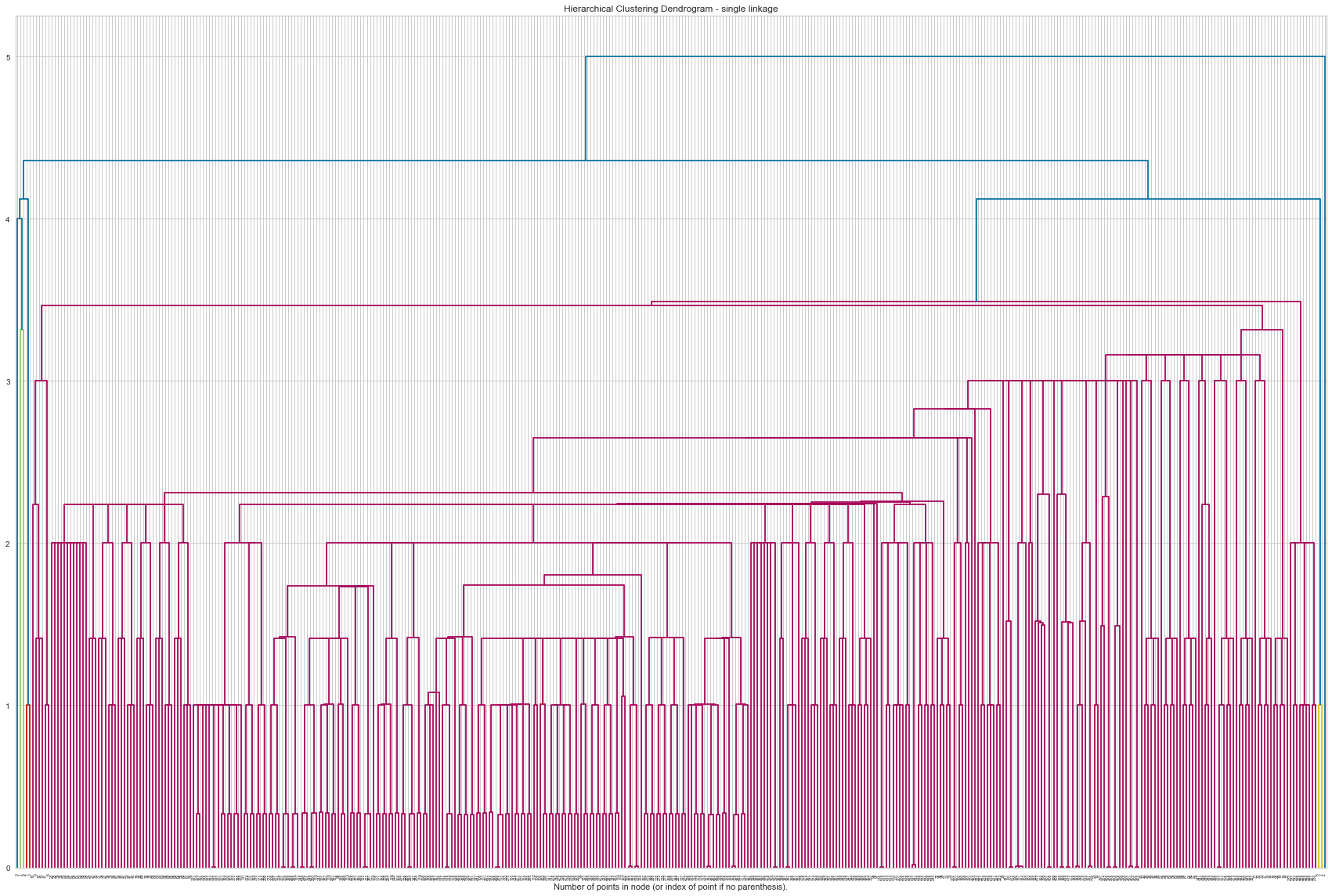
ruleID 0.0000 0.0625 0.1250 0.1875 0.2500 0.3125 0.3750 \  
ruleID   
0.0000 0 4.47257 6.78348 9.32926 7.81425 9.00542 7.35803   
0.0625 4.47257 0 7.61603 11.2701 9.00195 7.81425 8.13005   
0.1250 6.78348 7.61603 0 10.0501 7.55087 8.30874 3.17214   
0.1875 9.32926 11.2701 10.0501 0 9.89969 11.748 10.5373   
0.2500 7.81425 9.00195 7.55087 9.89969 0 4.47257 8.06323   
0.3125 9.00542 7.81425 8.30874 11.748 4.47257 0 8.77519   
0.3750 7.35803 8.13005 3.17214 10.5373 8.06323 8.77519 0   
0.4375 8.61344 9.2812 7.88021 7.68521 9.22145 9.84965 7.21137   
0.5000 8.32166 9.01063 7.55914 11.2293 2.83945 4.47606 8.06323   
0.5625 5.77204 6.72681 4.60341 9.49424 6.64061 7.48749 5.38843   
0.6250 5.77846 6.73175 4.60977 9.49692 6.64384 7.48984 5.39096   
0.6875 6.89004 7.70653 5.94276 10.2103 7.62833 8.375 5.00976   
0.7500 6.28987 7.17444 5.23361 9.81409 7.08872 7.88615 5.92795   
0.8125 3.10808 5.43714 7.448 9.81787 8.38549 9.5 7.9493   
0.8750 5.89624 9.72935 8.97566 11.0215 9.76681 12.4626 9.39415   
0.9375 9.15854 11.9902 13.6257 13.06 12.0197 14.2965 13.9038   
1.0000 9.16515 11.995 13.6296 13.0637 12.0234 14.2994 13.9065   
  
ruleID 0.4375 0.5000 0.5625 0.6250 0.6875 0.7500 0.8125 \  
ruleID   
0.0000 8.61344 8.32166 5.77204 5.77846 6.89004 6.28987 3.10808   
0.0625 9.2812 9.01063 6.72681 6.73175 7.70653 7.17444 5.43714   
0.1250 7.88021 7.55914 4.60341 4.60977 5.94276 5.23361 7.448   
0.1875 7.68521 11.2293 9.49424 9.49692 10.2103 9.81409 9.81787   
0.2500 9.22145 2.83945 6.64061 6.64384 7.62833 7.08872 8.38549   
0.3125 9.84965 4.47606 7.48749 7.48984 8.375 7.88615 9.5   
0.3750 7.21137 8.06323 5.38843 5.39096 5.00976 5.92795 7.9493   
0.4375 0 9.21976 7.00112 7.00251 6.71286 7.42278 9.11815   
0.5000 9.21976 0 6.00033 6.0013 7.61808 7.07549 8.83729   
0.5625 7.00112 6.00033 0 0.0625 4.69208 3.74635 6.48556   
0.6250 7.00251 6.0013 0.0625 0 4.69083 3.74374 6.48345   
0.6875 6.71286 7.61808 4.69208 4.69083 0 3.1629 6.16568   
0.7500 7.42278 7.07549 3.74635 3.74374 3.1629 0 5.47758   
0.8125 9.11815 8.83729 6.48556 6.48345 6.16568 5.47758 0   
0.8750 10.4015 10.1558 8.19132 8.18917 7.93947 7.41725 5.00039   
0.9375 12.5399 12.3366 10.7769 10.7749 10.586 10.1998 8.60323   
1.0000 12.5426 12.339 10.7792 10.7769 10.5876 10.2011 8.60437   
  
ruleID 0.8750 0.9375 1.0000   
ruleID   
0.0000 5.89624 9.15854 9.16515   
0.0625 9.72935 11.9902 11.995   
0.1250 8.97566 13.6257 13.6296   
0.1875 11.0215 13.06 13.0637   
0.2500 9.76681 12.0197 12.0234   
0.3125 12.4626 14.2965 14.2994   
0.3750 9.39415 13.9038 13.9065   
0.4375 10.4015 12.5399 12.5426   
0.5000 10.1558 12.3366 12.339   
0.5625 8.19132 10.7769 10.7792   
0.6250 8.18917 10.7749 10.7769   
0.6875 7.93947 10.586 10.5876   
0.7500 7.41725 10.1998 10.2011   
0.8125 5.00039 8.60323 8.60437   
0.8750 0 7.00028 7.00112   
0.9375 7.00028 0 0.0625   
1.0000 7.00112 0.0625 0

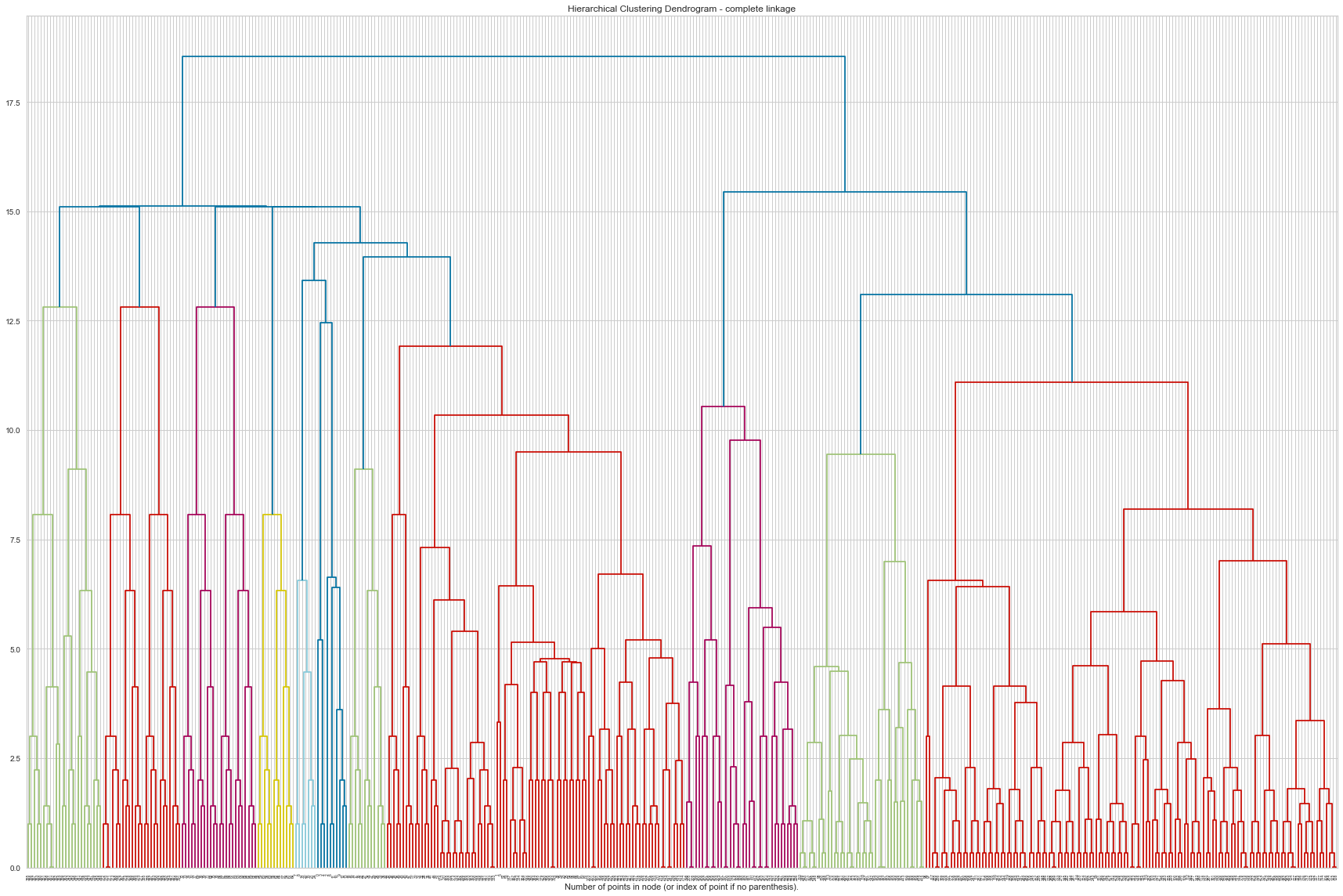
distance\_matrix(indexed\_data.to\_numpy(), indexed\_data.to\_numpy())

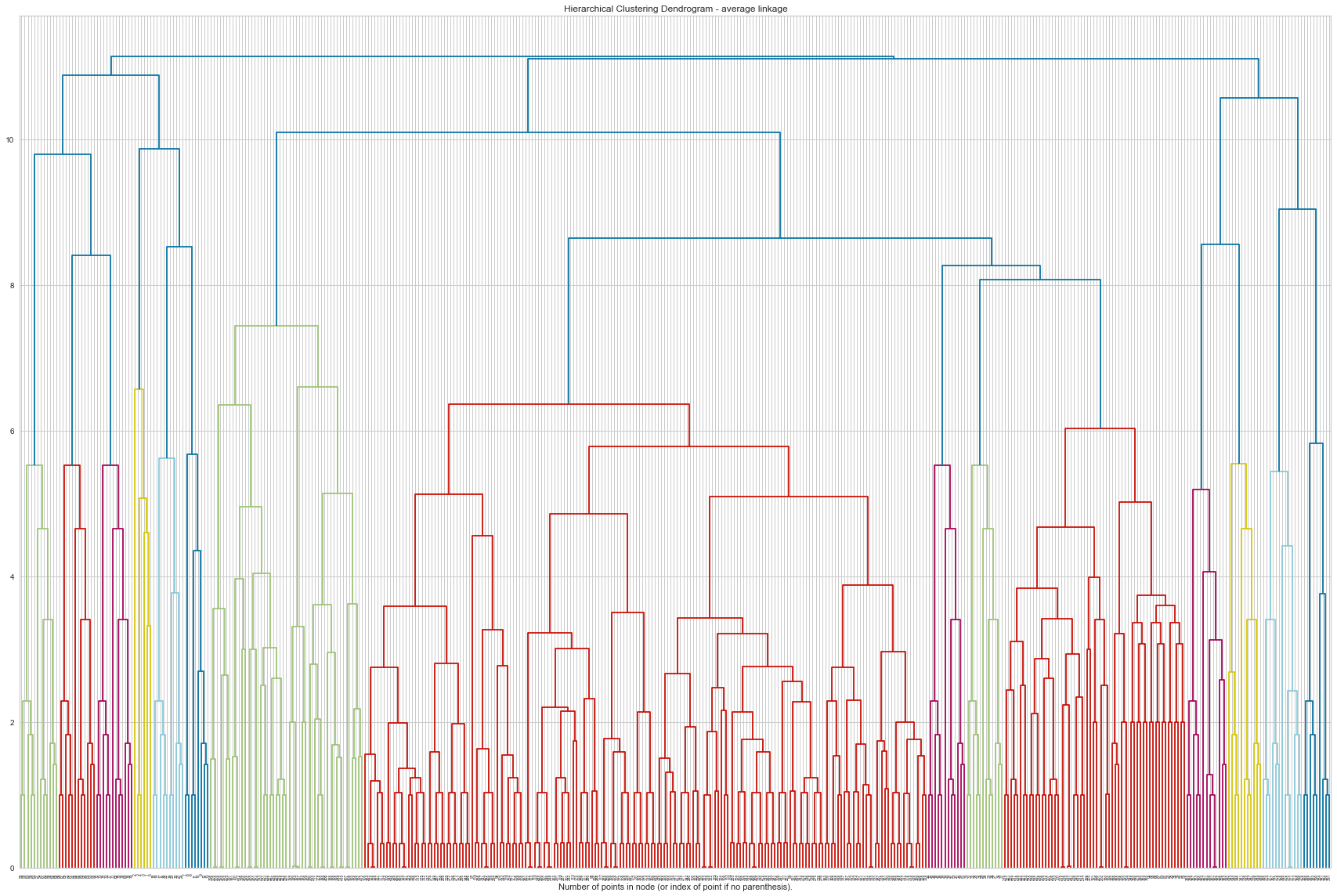
array([[0.0, 4.472572665703711, 6.783481775607568, 9.329263435555886,  
 7.814249804043892, 9.005423712963205, 7.358031326380718,  
 8.613443344563196, 8.32165848854662, 5.772036577326932,  
 5.77846216566311, 6.890040366354903, 6.289872812704562,  
 3.108079189789089, 5.896238207535378, 9.158542801668833,  
 9.16515138991168],  
 [4.472572665703711, 0.0, 7.616029559422679, 11.270120895536126,  
 9.001952913118354, 7.814249804043892, 8.130046509707062,  
 9.281197390423285, 9.010627406013413, 6.726812023536855,  
 6.731746151631091, 7.706531320899176, 7.174444664920066,  
 5.437140792732887, 9.729345108998858, 11.990230398119962,  
 11.994953365895176],  
 [6.783481775607568, 7.616029559422679, 0.0, 10.050069962443047,  
 7.550869155269478, 8.308739751009174, 3.17214438511238,  
 7.88020661213905, 7.559141816370427, 4.603412457080074,  
 4.6097722286464435, 5.942760827258657, 5.23360535386458,  
 7.447996794440771, 8.97566153550812, 13.625716724268122,  
 13.629586384039685],  
 [9.329263435555886, 11.270120895536126, 10.050069962443047, 0.0,  
 9.89969223006453, 11.748005149811606, 10.537322062554603,  
 7.685213074469699, 11.229321272899801, 9.494241675879122,  
 9.496915617715048, 10.21028892833107, 9.814092227506322,  
 9.817872732929471, 11.021463435043461, 13.059957886608977,  
 13.06369611748528],  
 [7.814249804043892, 9.001952913118354, 7.550869155269478,  
 9.89969223006453, 0.0, 4.472572665703711, 8.06322671143507,  
 9.221450875540139, 2.839454172900137, 6.640606617621616,  
 6.643841132959156, 7.628329191244961, 7.088723439378913,  
 8.385487836136905, 9.766812427808778, 12.019677876299347,  
 12.023414656411049],  
 [9.005423712963205, 7.814249804043892, 8.308739751009174,  
 11.748005149811606, 4.472572665703711, 0.0, 8.775186963820202,  
 9.849651009045955, 4.47606481744847, 7.487489565935968,  
 7.489836864044503, 8.375, 7.886152816804909, 9.5,  
 12.462600300499089, 14.29652492740806, 14.299393562315851],  
 [7.358031326380718, 8.130046509707062, 3.17214438511238,  
 10.537322062554603, 8.06322671143507, 8.775186963820202, 0.0,  
 7.211373395546787, 8.06322671143507, 5.388427994322648,  
 5.390964663211956, 5.009756106837937, 5.927952850689688,  
 7.949302249254333, 9.394147114027968, 13.903827036107721,  
 13.906495784344811],  
 [8.613443344563196, 9.281197390423285, 7.88020661213905,  
 7.685213074469699, 9.221450875540139, 9.849651009045955,  
 7.211373395546787, 0.0, 9.219756301009262, 7.001115982470223,  
 7.002510710452359, 6.712860791048776, 7.422779550141578,  
 9.118148112418442, 10.401509806273317, 12.539936203984453,  
 12.542583715088371],  
 [8.32165848854662, 9.010627406013413, 7.559141816370427,  
 11.229321272899801, 2.839454172900137, 4.47606481744847,  
 8.06322671143507, 9.219756301009262, 0.0, 6.000325512003495,  
 6.001301942078902, 7.618080877097591, 7.075485849042453,  
 8.837287833379651, 10.155817298474801, 12.336588112196987,  
 12.338962679253067],  
 [5.772036577326932, 6.726812023536855, 4.603412457080074,  
 9.494241675879122, 6.640606617621616, 7.487489565935968,  
 5.388427994322648, 7.001115982470223, 6.000325512003495, 0.0,  
 0.0625, 4.692081094780865, 3.7463523926614273, 6.485560885536423,  
 8.191315904663915, 10.776855988645297, 10.779211763853608],  
 [5.77846216566311, 6.731746151631091, 4.6097722286464435,  
 9.496915617715048, 6.643841132959156, 7.489836864044503,  
 5.390964663211956, 7.002510710452359, 6.001301942078902, 0.0625,  
 0.0, 4.690832148990198, 3.743744782967984, 6.4834524946204395,  
 8.1891696770796, 10.774862238098454, 10.776855988645297],  
 [6.890040366354903, 7.706531320899176, 5.942760827258657,  
 10.21028892833107, 7.628329191244961, 8.375, 5.009756106837937,  
 6.712860791048776, 7.618080877097591, 4.692081094780865,  
 4.690832148990198, 0.0, 3.1628952322199986, 6.165681227569262,  
 7.939468259902548, 10.585957679870065, 10.587618063096157],  
 [6.289872812704562, 7.174444664920066, 5.23360535386458,  
 9.814092227506322, 7.088723439378913, 7.886152816804909,  
 5.927952850689688, 7.422779550141578, 7.075485849042453,  
 3.7463523926614273, 3.743744782967984, 3.1628952322199986, 0.0,  
 5.477582153651372, 7.417251849573399, 10.19976255851086,  
 10.201102881551583],  
 [3.108079189789089, 5.437140792732887, 7.447996794440771,  
 9.817872732929471, 8.385487836136905, 9.5, 7.949302249254333,  
 9.118148112418442, 8.837287833379651, 6.485560885536423,  
 6.4834524946204395, 6.165681227569262, 5.477582153651372, 0.0,  
 5.000390609742403, 8.603233403784882, 8.604368439926315],  
 [5.896238207535378, 9.729345108998858, 8.97566153550812,  
 11.021463435043461, 9.766812427808778, 12.462600300499089,  
 9.394147114027968, 10.401509806273317, 10.155817298474801,  
 8.191315904663915, 8.1891696770796, 7.939468259902548,  
 7.417251849573399, 5.000390609742403, 0.0, 7.0002790122965814,  
 7.001115982470223],  
 [9.158542801668833, 11.990230398119962, 13.625716724268122,  
 13.059957886608977, 12.019677876299347, 14.29652492740806,  
 13.903827036107721, 12.539936203984453, 12.336588112196987,  
 10.776855988645297, 10.774862238098454, 10.585957679870065,  
 10.19976255851086, 8.603233403784882, 7.0002790122965814, 0.0,  
 0.0625],  
 [9.16515138991168, 11.994953365895176, 13.629586384039685,  
 13.06369611748528, 12.023414656411049, 14.299393562315851,  
 13.906495784344811, 12.542583715088371, 12.338962679253067,  
 10.779211763853608, 10.776855988645297, 10.587618063096157,  
 10.201102881551583, 8.604368439926315, 7.001115982470223, 0.0625,  
 0.0]], dtype=object)

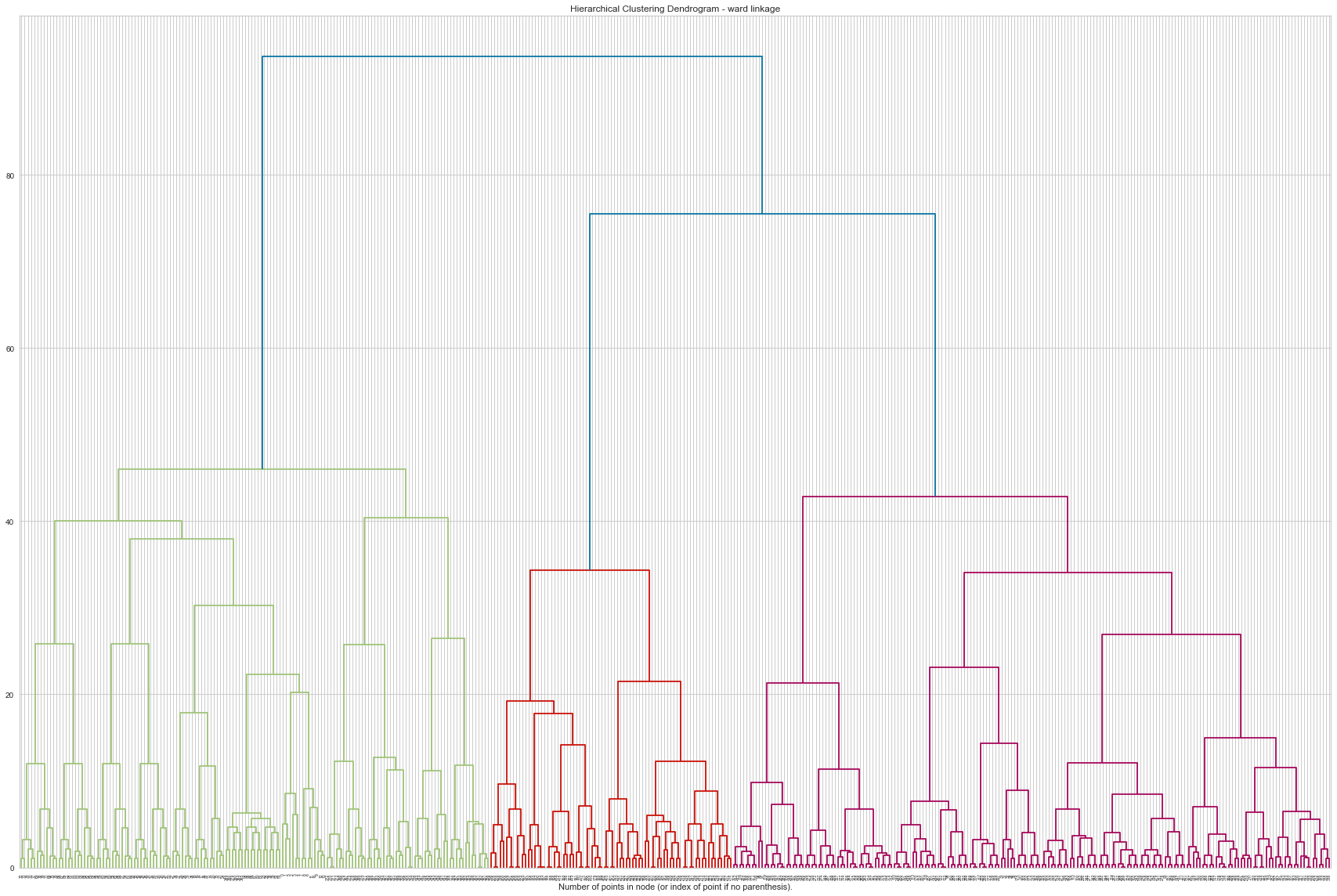
# distance\_mat

for linkage in ('single', 'complete', 'average', 'ward'):  
 model = AgglomerativeClustering(distance\_threshold=0, n\_clusters=None, linkage=linkage)  
 model = model.fit(indexed\_data)  
 plt.figure(figsize=(30,20))  
 plt.title(f'Hierarchical Clustering Dendrogram - {linkage} linkage')  
 plot\_dendrogram(model)  
 plt.xlabel("Number of points in node (or index of point if no parenthesis).")  
 plt.show()



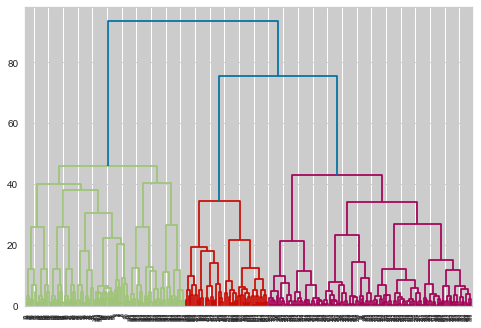






# print(model.distances\_)  
  
# means\_lst = []  
# for label in range(model.labels\_):  
# means\_lst.append(sample\_silhouette\_values[model.labels\_ == label].mean())

dep = sch.dendrogram(sch.linkage(indexed\_data,method='ward'))



agg\_clustering = AgglomerativeClustering(affinity = 'euclidean', linkage = 'ward')  
  
   
#predicting the labels  
   
agg\_clustering.fit(indexed\_data)  
  
  
score = silhouette\_score(indexed\_data[indexed\_data.columns[1:]].values, agg\_clustering.labels\_, metric='euclidean')  
  
score  
  
  
  
visualizer = SilhouetteVisualizer(agg\_clustering, colors='yellowbrick')  
# visualizer.fit(X)  
   
# #Plotting the results

score

0.28179392216067667