Setup

```
# Install TensorFlow Dececision Forests and the dependencies used in this colab
!pip install tensorflow_decision_forests plotly scikit-learn wurlitzer -U -qq
```

```
                                        15.3/15.3 MB 25.8 MB/s eta 0:
                                        15.6/15.6 MB 34.3 MB/s eta 0:
                                        12.1/12.1 MB 34.0 MB/s eta 0:
    ERROR: pip's dependency resolver does not currently take into account all t
    lida 0.0.10 requires fastapi, which is not installed.
    lida 0.0.10 requires kaleido, which is not installed.
    lida 0.0.10 requires python-multipart, which is not installed.
    lida 0.0.10 requires uvicorn, which is not installed.
```

```
import tensorflow_decision_forests as tfdf

import matplotlib.colors as mcolors
import math
import os
import numpy as np
import pandas as pd
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
from plotly.offline import iplot
import plotly.graph_objs as go
```

create random dataset

```
import numpy as np
import pandas as pd

import seaborn as sns
import matplotlib.pyplot as plt
```

```
# Set random seed for reproducibility
np.random.seed(42)

# Define parameters for the four different settings
settings = [
    {"mean": [1, 2, 3, 4, 5, 6, 7, 8], "covariance": np.eye(8)},
```

```
    {"mean": [10, 9, 8, 7, 6, 5, 4, 3], "covariance": np.diag([1, 2, 3, 4, 5, 6
    {"mean": [5, 5, 5, 5, 5, 5, 5, 5], "covariance": np.ones((8, 8)) * 0.5},
    {"mean": [0, 1, 0, 1, 0, 1, 0, 1], "covariance": np.array([[1, 0, 0, 0, 0,
                                                                [0, 2, 0, 0, 0,
                                                                [0, 0, 3, 0, 0,
                                                                [0, 0, 0, 4, 0,
                                                                [0, 0, 0, 0, 5,
                                                                [0, 0, 0, 0, 0,
                                                                [0, 0, 0, 0, 0,
                                                                [0, 0, 0, 0, 0,
]

# Initialize an empty list to store generated data
data_points = []

# Generate 500 data points for each parameter setting
for setting in settings:
    mean_vector = setting["mean"]
    covariance_matrix = setting["covariance"]

    # Generate 500 data points with multivariate normal distribution
    data = np.random.multivariate_normal(mean_vector, covariance_matrix, size=5

    # Append the generated data to the list
    data_points.append(data)

# Concatenate the data points along the first axis to create the final dataset
final_dataset = np.concatenate(data_points, axis=0)

# Print the shape of the final dataset
print("Shape of the final dataset:", final_dataset.shape)

final_dataset= pd.DataFrame(final_dataset)
deneme = final_dataset
```

```
Shape of the final dataset: (2000, 8)
```

```
final_dataset['class'] = 1
final_dataset
```

	0	1	2	3	4	5	6	
0	1.496714	1.861736	3.647689	5.523030	4.765847	5.765863	8.579213	8.767
1	0.530526	2.542560	2.536582	3.534270	5.241962	4.086720	5.275082	7.437
2	-0.012831	2.314247	2.091976	2.587696	6.465649	5.774224	7.067528	6.575
3	0.455617	2.110923	1.849006	4.375698	4.399361	5.708306	6.398293	9.852
4	0.986503	0.942289	3.822545	2.779156	5.208864	4.040330	5.671814	8.196
...	...	...	...	...	...	...	...	
1995	1.107273	0.444435	2.110790	0.105510	0.339918	-2.853897	1.538169	-3.007
1996	0.061788	1.094171	4.761031	1.584071	-2.388226	6.046785	2.645728	-0.454
1997	-0.356542	2.707558	-0.721904	-3.005897	2.531872	-3.181833	0.108690	3.230
1998	0.636953	-0.758819	-0.316832	1.549028	-2.732980	-1.084797	2.326314	-0.962
1999	0.565200	1.925869	-0.489812	3.380723	1.051830	0.286385	-0.951822	2.313

2000 rows × 9 columns

```
column_names = deneme.columns.tolist()
new_column_names = {
    column_names[0]: 'NewFeature1',
    column_names[1]: 'NewFeature2',
    column_names[2]: 'NewFeature3',
    column_names[3]: 'NewFeature4',
    column_names[4]: 'NewFeature5',
    column_names[5]: 'NewFeature6',
    column_names[6]: 'NewFeature7',
    column_names[7]: 'NewFeature8',


}

deneme = deneme.rename(columns=new_column_names)
deneme.dtypes
deneme.columns.astype(str)
```

```
Index(['NewFeature1', 'NewFeature2', 'NewFeature3', 'NewFeature4',
       'NewFeature5', 'NewFeature6', 'NewFeature7', 'NewFeature8',
'class'],
      dtype='object')
```

```
deneme
```

	NewFeature1	NewFeature2	NewFeature3	NewFeature4	NewFeature5	NewFea
0	1.496714	1.861736	3.647689	5.523030	4.765847	5.
1	0.530526	2.542560	2.536582	3.534270	5.241962	4.
2	-0.012831	2.314247	2.091976	2.587696	6.465649	5.
3	0.455617	2.110923	1.849006	4.375698	4.399361	5.
4	0.986503	0.942289	3.822545	2.779156	5.208864	4.
...	...	...	...	...	...	
1995	1.107273	0.444435	2.110790	0.105510	0.339918	-2.
1996	0.061788	1.094171	4.761031	1.584071	-2.388226	6.
1997	-0.356542	2.707558	-0.721904	-3.005897	2.531872	-3.
1998	0.636953	-0.758819	-0.316832	1.549028	-2.732980	-1.
1999	0.565200	1.925869	-0.489812	3.380723	1.051830	0.

2000 rows × 9 columns

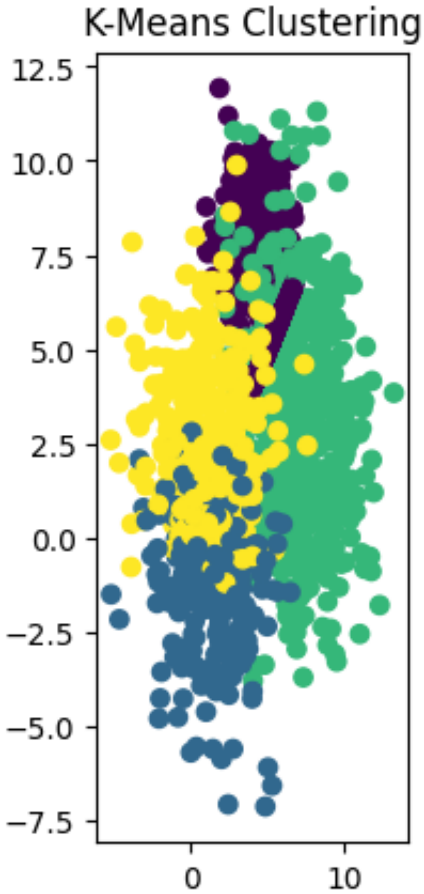
```
from sklearn.cluster import KMeans, AgglomerativeClustering

kmeans = KMeans(n_clusters=4, random_state=0)
clusters_kmeans = kmeans.fit_predict(deneme)
clusters_kmeans
```

```
array([0, 0, 0, ..., 3, 1, 3], dtype=int32)
```

```
plt.subplot(132)
plt.scatter(deneme['NewFeature4'], deneme['NewFeature8'], c=clusters_kmeans, cm
plt.title("K-Means Clustering")
```

```
Text(0.5, 1.0, 'K-Means Clustering')
```




```
def mean(deneme, clusters_list):
    cluster_means = []
    for cluster_index in clusters_list:
        cluster_data = deneme.iloc[cluster_index]
        mean = np.mean(cluster_data, axis=0)
        cluster_means.append(mean)
    return cluster_means
```

```
kmeans_means = mean(deneme, clusters_kmeans)
```

```
kmeans_means
```

```
    3.7626378802570923,
    3.7626378802570923,
    3.7626378802570923,
    3.7626378802570923,
    3.7626378802570923,
    3.7626378802570923,
    3.7626378802570923,
    3.7626378802570923,
    3.7626378802570923,
    3.7626378802570923,
    3.7626378802570923,
    3.7626378802570923,
    3.7626378802570923,
    3.7626378802570923,
    3.7626378802570923,
    3.7626378802570923,
    3.7626378802570923,
    3.7626378802570923,
    3.7626378802570923,
    3.7626378802570923,
    3.7626378802570923,
    3.7626378802570923,
    3.7626378802570923,
    3.7626378802570923,
    3.7626378802570923,
    3.7626378802570923,
    3.7626378802570923,
    3.7626378802570923,
    3.7626378802570923,
    3.7626378802570923,
    3.7626378802570923,
    3.7626378802570923,
    3.7626378802570923,
    3.7626378802570923,
    3.7626378802570923,
    3.7626378802570923,
    3.7626378802570923,
    3.7626378802570923,
    3.7626378802570923,
    3.7626378802570923,
    3.7626378802570923,
    3.7626378802570923,
    3.7626378802570923,
    3.7626378802570923,
    3.7626378802570923,
    3.7626378802570923,
    3.7626378802570923,
    3.7626378802570923,
```

```
3.7626378802570923,
3.7626378802570923,
3.7626378802570923,
3.7626378802570923,
3.7626378802570923,
3.7626378802570923,
3.7626378802570923,
3.7626378802570923,
3.7626378802570923,
3.7626378802570923,
3.7626378802570923,
```

```
kmeans_means[-1]
```

```
4.016609270963945
```

```
kmeans_means[-501]
```

```
4.600836162148505
```

```
kmeans_means[-1001]
```

```
3.7626378802570923
```

```
kmeans_means[-1501]
```

```
4.600836162148505
```

```
kmeans_covariances
```

```
array(7.10791743),
array(7.10791743),
array(7.10791743),
array(7.10791743),
array(7.10791743),
array(7.10791743),
array(7.10791743),
array(7.10791743),
array(7.10791743),
array(7.10791743),
array(7.10791743),
array(7.10791743),
array(7.10791743),
array(7.10791743),
array(7.10791743),
array(7.10791743),
array(7.10791743),
array(7.10791743),
array(7.10791743),
```

```
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743)
```

```
import numpy as np

# Assuming 'original_data' is your original dataset with shape (n_samples, n_fe
# Replace this with your actual dataset
np.random.seed(42)
original_data = final_dataset

# Function to permute each variable independently
```

```
def permute_variable(variable):
    return np.random.permutation(variable)

# Permute each variable in the original data independently
synthetic_data = np.apply_along_axis(permute_variable, axis=0, arr=original_dat

# Print the first few rows of the synthetic data
print("Synthetic Data:")
print(synthetic_data[:5, :])
synthetic_data= pd.DataFrame(synthetic_data)
synthetic_data
```

```
Synthetic Data:
[[-1.11542     0.05939614  5.33956288  2.67352787  5.0003385   5.491968
   4.77933876  4.01161911  1.        ]
 [ 0.94759555  2.72113536  2.01849135  4.36150353  5.2088636   4.67754204
   4.46971248  4.97803484  1.        ]
 [ 4.52083821  1.69542487  5.05741225  2.4969197   5.4148655   4.81899682
   7.36659825  8.8177663   1.        ]
 [10.01502775  6.01623815  7.04292182  5.13652964  5.40150077  5.91763253
   6.72686954  1.83333317  1.        ]
 [ 5.40079755 -0.05837364  3.84843088 -0.808801    6.10387734  3.63123537
   5.26985394 -3.39383747  1.        ]]
```

	0	1	2	3	4	5	6	
0	-1.115420	0.059396	5.339563	2.673528	5.000339	5.491968	4.779339	4.0116
1	0.947596	2.721135	2.018491	4.361504	5.208864	4.677542	4.469712	4.9780
2	4.520838	1.695425	5.057412	2.496920	5.414866	4.818997	7.366598	8.8177
3	10.015028	6.016238	7.042922	5.136530	5.401501	5.917633	6.726870	1.8333
4	5.400798	-0.058374	3.848431	-0.808801	6.103877	3.631235	5.269854	-3.3938
...	...	...	...	...	...	...	...	
1995	4.846778	2.820158	8.505003	3.644273	5.363558	6.861091	7.634721	0.3325
1996	5.094596	5.379973	-2.362289	0.568864	5.822160	7.909417	6.474120	4.1096
1997	11.805574	6.011951	4.223961	2.810333	6.465221	8.165002	6.953079	4.9948
1998	5.937028	2.601207	5.702064	-1.297167	5.151158	4.184350	4.051180	7.6762
1999	4.784226	5.759215	4.194763	4.872505	0.359952	5.251684	3.918547	-0.0431

2000 rows × 9 columns

```
synthetic_data['class'] = 2
synthetic_data
```

	0	1	2	3	4	5	6	
0	-1.115420	0.059396	5.339563	2.673528	5.000339	5.491968	4.779339	4.0116
1	0.947596	2.721135	2.018491	4.361504	5.208864	4.677542	4.469712	4.9780
2	4.520838	1.695425	5.057412	2.496920	5.414866	4.818997	7.366598	8.8177
3	10.015028	6.016238	7.042922	5.136530	5.401501	5.917633	6.726870	1.8333
4	5.400798	-0.058374	3.848431	-0.808801	6.103877	3.631235	5.269854	-3.3938
...	...	...	...	...	...	...	...	
1995	4.846778	2.820158	8.505003	3.644273	5.363558	6.861091	7.634721	0.3325
1996	5.094596	5.379973	-2.362289	0.568864	5.822160	7.909417	6.474120	4.1096
1997	11.805574	6.011951	4.223961	2.810333	6.465221	8.165002	6.953079	4.9948
1998	5.937028	2.601207	5.702064	-1.297167	5.151158	4.184350	4.051180	7.6762
1999	4.784226	5.759215	4.194763	4.872505	0.359952	5.251684	3.918547	-0.0431

2000 rows × 10 columns

```
df_combined = pd.concat([final_dataset, synthetic_data], ignore_index=True)
df_co= df_combined

df_combined
```

	0	1	2	3	4	5	6	
0	1.496714	1.861736	3.647689	5.523030	4.765847	5.765863	8.579213	8.76743
1	0.530526	2.542560	2.536582	3.534270	5.241962	4.086720	5.275082	7.43771
2	-0.012831	2.314247	2.091976	2.587696	6.465649	5.774224	7.067528	6.57525
3	0.455617	2.110923	1.849006	4.375698	4.399361	5.708306	6.398293	9.85227
4	0.986503	0.942289	3.822545	2.779156	5.208864	4.040330	5.671814	8.19686
...	...	...	...	...	...	...	...	.
3995	4.846778	2.820158	8.505003	3.644273	5.363558	6.861091	7.634721	0.33254
3996	5.094596	5.379973	-2.362289	0.568864	5.822160	7.909417	6.474120	4.10961
3997	11.805574	6.011951	4.223961	2.810333	6.465221	8.165002	6.953079	4.99480
3998	5.937028	2.601207	5.702064	-1.297167	5.151158	4.184350	4.051180	7.67626
3999	4.784226	5.759215	4.194763	4.872505	0.359952	5.251684	3.918547	-0.04313

4000 rows × 10 columns

```
column_names = df_combined.columns.tolist()
new_column_names = {
    column_names[0]: 'NewFeature1',
    column_names[1]: 'NewFeature2',
    column_names[2]: 'NewFeature3',
    column_names[3]: 'NewFeature4',
    column_names[4]: 'NewFeature5',
    column_names[5]: 'NewFeature6',
    column_names[6]: 'NewFeature7',
    column_names[7]: 'NewFeature8',
    column_names[9]: 'NewFeature9',

}

df_combined = df_combined.rename(columns=new_column_names)
df_combined.dtypes
```

```
NewFeature1    float64
NewFeature2    float64
NewFeature3    float64
NewFeature4    float64
NewFeature5    float64
NewFeature6    float64
NewFeature7    float64
NewFeature8    float64
class            int64
NewFeature9    float64
dtype: object
```

```
df_combined
```

	NewFeature1	NewFeature2	NewFeature3	NewFeature4	NewFeature5	NewFea
0	1.496714	1.861736	3.647689	5.523030	4.765847	5.
1	0.530526	2.542560	2.536582	3.534270	5.241962	4.
2	-0.012831	2.314247	2.091976	2.587696	6.465649	5.
3	0.455617	2.110923	1.849006	4.375698	4.399361	5.
4	0.986503	0.942289	3.822545	2.779156	5.208864	4.
...	...	...	...	...	...	
3995	4.846778	2.820158	8.505003	3.644273	5.363558	6.
3996	5.094596	5.379973	-2.362289	0.568864	5.822160	7.
3997	11.805574	6.011951	4.223961	2.810333	6.465221	8.
3998	5.937028	2.601207	5.702064	-1.297167	5.151158	4.
3999	4.784226	5.759215	4.194763	4.872505	0.359952	5.

4000 rows × 10 columns

```
df_combined = df_combined.drop('NewFeature9', axis= 1)
df_co= df_combined
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-252-2e802197c4bc> in <cell line: 1>()
----> 1 df_combined = df_combined.drop('NewFeature9', axis= 1)
      2 df_co= df_combined

5 frames

/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in drop(self, labels, errors)
   6932         if mask.any():
   6933             if errors != "ignore":
-> 6934                 raise KeyError(f"{list(labels[mask])} not found in axis")
   6935             indexer = indexer[~mask]
   6936         return self.delete(indexer)

KeyError: "['NewFeature9'] not found in axis"
```

```
df_co= df_combined
```

```
from sklearn.model_selection import train_test_split
```

```
train, test = train_test_split(df_combined, test_size=0.2, random_state=0)
```

```
test
```

	NewFeature1	NewFeature2	NewFeature3	NewFeature4	NewFeature5	NewFea
2230	1.685870	0.444104	2.664354	4.564048	-0.775611	6.
668	10.375456	9.868003	8.058941	3.002222	1.077870	4.
3616	0.289670	8.716253	5.606850	0.188825	5.247081	5.
2363	0.122017	2.579291	2.058499	4.676469	4.790272	1.
142	2.189470	0.772392	3.597400	4.701173	4.702436	7.
...	...	...	...	...	...	
1118	5.698623	5.698623	5.698623	5.698623	5.698623	5.
3572	12.343654	1.243236	5.251684	0.132305	10.989783	-4.
2482	0.324822	8.365665	2.605373	4.727725	4.059877	2.
643	9.147775	12.218503	8.563917	8.493223	5.527330	4.
299	-0.496529	1.349976	2.916562	2.550355	4.078140	4.

800 rows × 9 columns

```
train
```

	NewFeature1	NewFeature2	NewFeature3	NewFeature4	NewFeature5	NewFea
1161	4.985772	4.985772	4.985772	4.985772	4.985772	4.
2355	10.739180	2.418899	9.255877	2.368724	4.834632	6.
1831	2.112089	-3.596666	-2.439475	-3.796326	-2.582678	4.
156	1.233786	0.444104	3.330880	4.833529	3.006264	6.
195	0.978633	1.252788	0.575760	4.884045	5.736844	5.
...	...	...	...	...	...	
835	9.074608	8.270400	8.248214	7.032250	5.885695	6.
3264	-0.709029	0.569225	5.376283	3.002222	9.355987	4.
1653	0.169403	-1.269536	1.083356	-0.172570	-2.933132	1.
2607	3.898921	0.209714	1.142117	4.282009	5.908440	5.
2732	1.897066	2.010233	0.340587	9.499421	5.256578	5.

3200 rows × 9 columns

```
# , and convert it into a TensorFlow dataset.
train_ds = tfdf.keras.pd_dataframe_to_tf_dataset(train, label="class")
test_ds = tfdf.keras.pd_dataframe_to_tf_dataset(test, label="class")
```

```
train
```

	NewFeature1	NewFeature2	NewFeature3	NewFeature4	NewFeature5	NewFea
1161	4.985772	4.985772	4.985772	4.985772	4.985772	4.
2355	10.739180	2.418899	9.255877	2.368724	4.834632	6.
1831	2.112089	-3.596666	-2.439475	-3.796326	-2.582678	4.
156	1.233786	0.444104	3.330880	4.833529	3.006264	6.
195	0.978633	1.252788	0.575760	4.884045	5.736844	5.
...	...	...	...	...	...	
835	9.074608	8.270400	8.248214	7.032250	5.885695	6.
3264	-0.709029	0.569225	5.376283	3.002222	9.355987	4.
1653	0.169403	-1.269536	1.083356	-0.172570	-2.933132	1.
2607	3.898921	0.209714	1.142117	4.282009	5.908440	5.
2732	1.897066	2.010233	0.340587	9.499421	5.256578	5.

3200 rows × 9 columns

Following are the first five examples of the training dataset. Notice that different columns represent different quantities. For example, how would you compare the distance between *relationship* and *age*?

```
# Print the first 5 examples.
train.head()
```

	NewFeature1	NewFeature2	NewFeature3	NewFeature4	NewFeature5	NewFea
1161	4.985772	4.985772	4.985772	4.985772	4.985772	4.
2355	10.739180	2.418899	9.255877	2.368724	4.834632	6.
1831	2.112089	-3.596666	-2.439475	-3.796326	-2.582678	4.
156	1.233786	0.444104	3.330880	4.833529	3.006264	6.
195	0.978633	1.252788	0.575760	4.884045	5.736844	5.

A Random Forest is trained as follows:

```
# Train a Random Forest
model = tfdf.keras.RandomForestModel(num_trees=1000)
model.fit(train_ds)
```

```
Use /tmp/tmpoi5n3lxx as temporary training directory
Reading training dataset...
Training dataset read in 0:00:05.284839. Found 3200 examples.
Training model...
Model trained in 0:00:10.380981
Compiling model...
Model compiled.
<keras.src.callbacks.History at 0x7dd61b2608e0>
```

The performance of the Random Forest model is:

```
model_inspector = model.make_inspector()
out_of_bag_accuracy = model_inspector.evaluation().accuracy
print(f"Out-of-bag accuracy: {out_of_bag_accuracy:.4f}")
```

```
Out-of-bag accuracy: 0.9656
```

This is an expected accuracy value for Random Forest models on this dataset. It indicates that the model is correctly trained.

We can also measure the accuracy of the model on the test datasets:

```
# The test accuracy is measured on the test datasets.
model.compile(["accuracy"])
test_accuracy = model.evaluate(test_ds, return_dict=True, verbose=0)["accuracy'
print(f"Test accuracy: {test_accuracy:.4f}")
```

```
Test accuracy: 0.9600
```

Proximities

First, we inspect the number of trees in the model and the number of examples in the test datasets.

```
print("The model contains", model_inspector.num_trees(), "trees.")
print("The test dataset contains", test.shape[0], "examples.")
```

```
The model contains 1000 trees.
The test dataset contains 800 examples.
```

```
test
```

	NewFeature1	NewFeature2	NewFeature3	NewFeature4	NewFeature5	NewFea
2230	1.685870	0.444104	2.664354	4.564048	-0.775611	6.
668	10.375456	9.868003	8.058941	3.002222	1.077870	4.
3616	0.289670	8.716253	5.606850	0.188825	5.247081	5.
2363	0.122017	2.579291	2.058499	4.676469	4.790272	1.
142	2.189470	0.772392	3.597400	4.701173	4.702436	7.
...	...	...	...	...	...	
1118	5.698623	5.698623	5.698623	5.698623	5.698623	5.
3572	12.343654	1.243236	5.251684	0.132305	10.989783	-4.
2482	0.324822	8.365665	2.605373	4.727725	4.059877	2.
643	9.147775	12.218503	8.563917	8.493223	5.527330	4.
299	-0.496529	1.349976	2.916562	2.550355	4.078140	4.

800 rows × 9 columns

The method predict_get_leaves() returns the index of the active leaf for each example and each tree.

```
leaves = model.predict_get_leaves(test_ds)
print("The leaf indices:\n", leaves)
```

```
WARNING:tensorflow:AutoGraph could not transform <function simple_ml_infere
Please report this to the TensorFlow team. When filing the bug, set the ver
Cause: could not get source code
To silence this warning, decorate the function with @tf.autograph.experimen
WARNING: AutoGraph could not transform <function simple_ml_inference_leaf_i
Please report this to the TensorFlow team. When filing the bug, set the ver
Cause: could not get source code
To silence this warning, decorate the function with @tf.autograph.experimen
The leaf indices:
 [[ 18  35  26 ...   9  10  38]
 [ 65  73 105 ...  40  70  45]
 [ 64  22  48 ...  38  54  16]
 ...
 [ 42  56 101 ...  27  49  63]
 [104  81  81 ...  44  83 113]
 [  6  50  21 ...   9  20  23]]
```

```
print("The predicted leaves have shape", leaves.shape,
      "(we expect [num_examples, num_trees]")
```

```
The predicted leaves have shape (800, 1000) (we expect [num_examples, num_t
```

Double-click (or enter) to edit

```
def compute_proximity(leaves, step_size=100):
  """Computes the proximity between each pair of examples.



  Returns:
    The example pair-wise proximity matrix of shape [n,n] with "n" the number c
    examples.
  """

  example_idx = 0
  num_examples = leaves.shape[0]
  t_leaves = np.transpose(leaves)
  proximities = []

  # Instead of computing the proximity in between all the examples at the same
  # time, we compute the similarity in blocks of "step_size" examples. This
  # makes the code more efficient with the the numpy broadcast.
  while example_idx < num_examples:
    end_idx = min(example_idx + step_size, num_examples)
    proximities.append(
        np.mean(
            leaves[..., np.newaxis] == t_leaves[:,
                                                example_idx:end_idx][np.newaxis
                                                                     ...],
            axis=1))
    example_idx = end_idx
  return np.concatenate(proximities, axis=1)


proximity = compute_proximity(leaves)
print("The shape of proximity is", proximity.shape)
```

```
The shape of proximity is (800, 800)
```

Here, `proximity[i,j]` is the proximity in between the example `i` and `j`.

The proximity matrix:

```
proximity
```

```
array([[1.   , 0.025, 0.   , ..., 0.005, 0.   , 0.032],
       [0.025, 1.   , 0.001, ..., 0.003, 0.004, 0.   ],
       [0.   , 0.001, 1.   , ..., 0.247, 0.   , 0.001],
       ...,
       [0.005, 0.003, 0.247, ..., 1.   , 0.   , 0.   ],
       [0.   , 0.004, 0.   , ..., 0.   , 1.   , 0.   ],
       [0.032, 0.   , 0.001, ..., 0.   , 0.   , 1.   ]])
```

The proximity matrix has several interesting properties, notably, it is symmetrical, positive, and the diagonal elements are all 1.

Projection

Our first use of the proximity is to project the examples on the two dimensional plane.

If $\text{prox} \in [0, 1]$ is a proximity, $1 - \text{prox}$ is a distance between examples. Breiman proposes to compute the inner products of those distances, and to plot the eigenvalues. See details here.

Instead, we will use the t-SNE which is a more modern way to visualize high-dimensional data.

Note: We use the t-SNE's Scikit-learn implementation.

```
distance = 1 - proximity

t_sne = TSNE(
    # Number of dimensions to display. 3d is also possible.
    n_components=2,
    # Control the shape of the projection. Higher values create more
    # distinct but also more collapsed clusters. Can be in 5-50.
    perplexity=20,
    metric="precomputed",
    init="random",
    verbose=1,
    learning_rate="auto").fit_transform(distance)
```

```
[t-SNE] Computing 61 nearest neighbors...
[t-SNE] Indexed 800 samples in 0.003s...
[t-SNE] Computed neighbors for 800 samples in 0.023s...
[t-SNE] Computed conditional probabilities for sample 800 / 800
[t-SNE] Mean sigma: 0.190733
[t-SNE] KL divergence after 250 iterations with early exaggeration: 62.3745
[t-SNE] KL divergence after 1000 iterations: 0.754572
```

```
distance
```

```
array([[0.   , 0.975, 1.   , ..., 0.995, 1.   , 0.968],
       [0.975, 0.   , 0.999, ..., 0.997, 0.996, 1.   ],
       [1.   , 0.999, 0.   , ..., 0.753, 1.   , 0.999],
       ...,
       [0.995, 0.997, 0.753, ..., 0.   , 1.   , 1.   ],
       [1.   , 0.996, 1.   , ..., 1.   , 0.   , 1.   ],
       [0.968, 1.   , 0.999, ..., 1.   , 1.   , 0.   ]])
```

```
from scipy.cluster.hierarchy import linkage,dendrogram,fcluster
dict_linkage = {}
list_method= ['centroid','ward','median']

for k in list_method:
    dict_linkage[k] = linkage(
      y = distance,
      method = k,
      optimal_ordering = True)
```

```
<ipython-input-91-34d71b2801a1>:6: ClusterWarning:

scipy.cluster: The symmetric non-negative hollow observation matrix looks s
```

```
dict_linkage['ward']
```

```
array([[ 423.        ,   23.        ,    0.        ,    2.        ],
       [  81.        ,  178.        ,    0.        ,    2.        ],
       [ 117.        ,  175.        ,    0.        ,    2.        ],
       ...,
       [1586.        , 1595.        ,   41.52721597,  578.        ],
       [1596.        , 1588.        ,   42.36614711,  698.        ],
       [1594.        , 1597.        ,   77.96526644,  800.        ]])
```

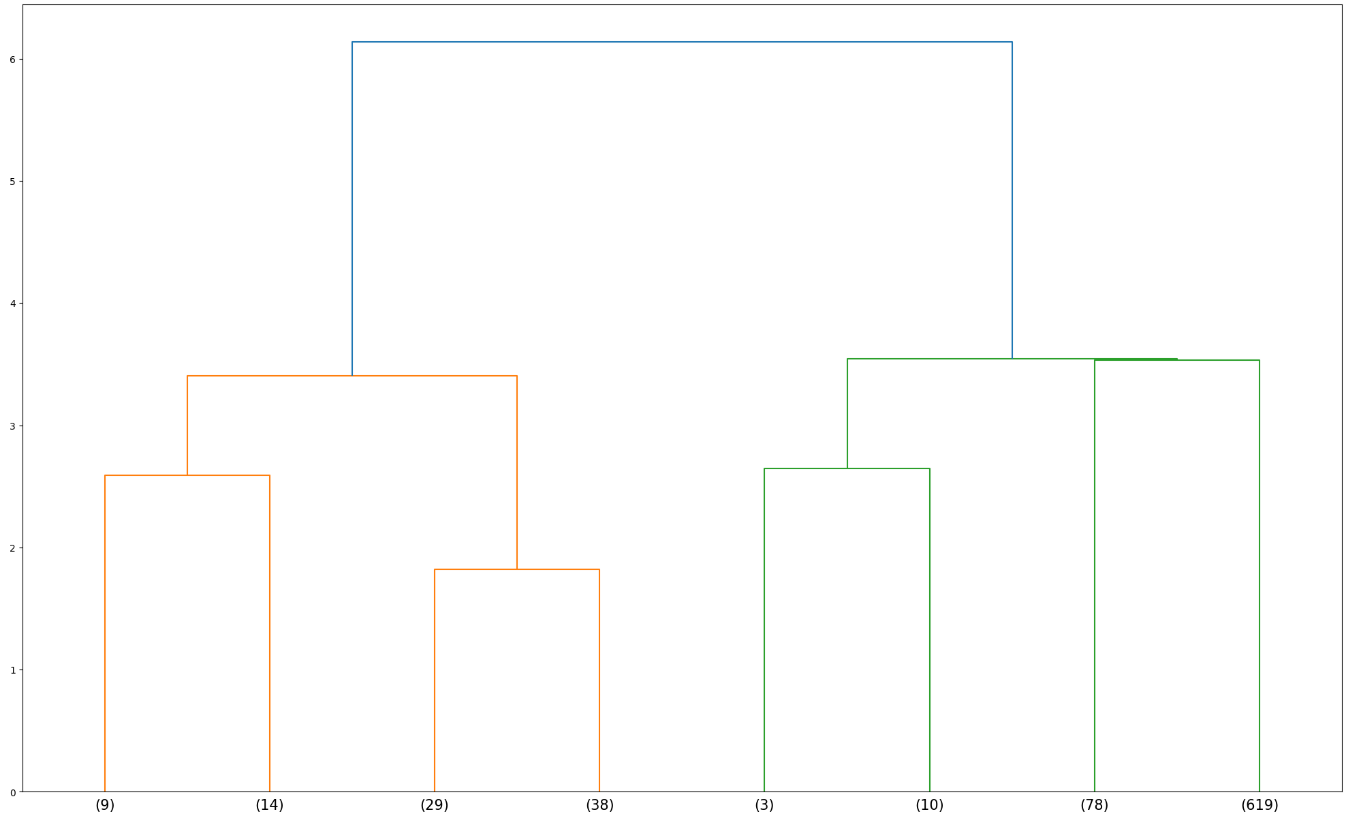
```
dict_linkage['median']
```

```
array([[ 646.        ,  316.        ,    0.        ,    2.        ],
       [ 178.        ,   81.        ,    0.        ,    2.        ],
       [  23.        ,  423.        ,    0.        ,    2.        ],
       ...,
       [1594.        , 1595.        ,    3.29344253,  684.        ],
       [1596.        , 1515.        ,    3.72884517,  729.        ],
       [1588.        , 1597.        ,    5.91211814,  800.        ]])
```

```
dict_linkage['centroid']
```

```
array([[ 316.        ,  646.        ,    0.        ,    2.        ],
       [  81.        ,  178.        ,    0.        ,    2.        ],
       [ 423.        ,   23.        ,    0.        ,    2.        ],
       ...,
       [1594.        , 1585.        ,    3.53510235,  697.        ],
       [1596.        , 1593.        ,    3.5433901 ,  710.        ],
       [1595.        , 1597.        ,    6.13654295,  800.        ]])
```

```
from matplotlib import pyplot as plt
fig = plt.figure(figsize=(25, 15))
dendrogram(
  Z = dict_linkage['centroid'],
  truncate_mode='level', p=2,
  count_sort = True,
  distance_sort = True,
  orientation = 'top',
  leaf_font_size = 15)
plt.show()
```


0
1
2
3
4
5
6
(9)
(14)
(29)
(38)
(3)
(10)
(78)
(619)

```
cl = dict_linkage['centroid']
cl1 = dict_linkage['ward']
cl2 = dict_linkage['median']


numclust =4
```

```
fl_centroid = fcluster(cl,numclust,criterion='maxclust')
fl_centroid
```

```
array([2, 2, 2, 2, 2, 2, 3, 2, 4, 1, 2, 2, 2, 2, 2, 3, 2, 2, 2, 2, 2, 1,
       2, 1, 2, 2, 2, 2, 2, 2, 3, 2, 1, 2, 1, 3, 2, 2, 2, 2, 2, 3, 2, 2,
       2, 2, 2, 2, 2, 3, 1, 2, 2, 3, 1, 1, 2, 2, 2, 2, 2, 2, 3, 2, 2, 2,
       2, 2, 3, 1, 2, 3, 2, 2, 1, 2, 2, 2, 2, 2, 1, 1, 2, 2, 2, 1, 2, 1,
       2, 2, 2, 2, 2, 2, 2, 3, 2, 1, 2, 2, 2, 2, 2, 2, 2, 3, 1, 3, 1, 2,
       2, 2, 2, 2, 2, 4, 1, 1, 3, 3, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2,
       2, 1, 2, 2, 3, 3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 2, 3, 2,
       2, 2, 2, 2, 2, 3, 2, 2, 2, 2, 2, 2, 2, 4, 1, 2, 2, 2, 2, 2, 2, 1,
       2, 2, 1, 2, 4, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 1, 3, 1, 2, 2, 4, 3,
       2, 1, 2, 2, 2, 2, 2, 1, 1, 2, 2, 2, 2, 2, 2, 3, 2, 2, 2, 3, 2, 2,
       3, 1, 2, 2, 3, 2, 2, 2, 2, 2, 2, 2, 3, 1, 2, 2, 2, 3, 2, 2, 1, 2,
       1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 2, 3, 2, 2, 2, 2, 2, 1, 3,
       2, 2, 2, 3, 1, 3, 1, 2, 2, 2, 3, 2, 2, 4, 1, 2, 3, 2, 2, 2, 3, 3,
       2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2, 3, 1, 2, 1, 2, 3, 2,
       2, 2, 2, 2, 2, 1, 1, 2, 3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 3, 2, 1, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       3, 2, 2, 2, 2, 2, 2, 3, 2, 3, 2, 4, 2, 2, 2, 2, 2, 3, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 1, 3, 2, 2, 2, 2, 2, 2, 2, 3, 4, 2, 2, 2, 2,
       1, 2, 3, 2, 1, 1, 2, 2, 1, 2, 2, 4, 1, 2, 2, 2, 2, 2, 1, 3, 2, 2,
       2, 2, 2, 2, 3, 3, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 1, 2, 2,
       2, 1, 2, 2, 2, 1, 2, 2, 2, 1, 2, 3, 3, 2, 3, 2, 2, 2, 3, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 1, 2, 3, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2,
       2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 4, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1,
       2, 2, 2, 2, 4, 2, 2, 1, 3, 3, 3, 3, 2, 2, 2, 2, 1, 2, 2, 2, 3, 2,
       2, 2, 2, 2, 2, 2, 3, 3, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2,
       1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 3, 1, 2, 2, 2, 2, 4, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 3,
       2, 2, 2, 2, 2, 2, 2, 3, 1, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3,
       2, 2, 2, 3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 3, 2, 2, 1, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 3,
       3, 2, 3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 4, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 2, 2, 2, 2, 2, 2, 2, 1, 3, 2, 2,
       2, 2, 1, 2, 3, 2, 2, 2, 3, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 2, 2,
       2, 2, 2, 1, 2, 2, 2, 2], dtype=int32)
```

```
fl_median = fcluster(cl2,numclust,criterion='maxclust')
fl_median
```

```
array([2, 2, 2, 2, 2, 2, 4, 2, 3, 1, 2, 2, 2, 2, 2, 4, 2, 2, 2, 2, 2, 1,
       2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 3, 4, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 1, 2, 2, 4, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 4, 1, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 1, 2, 2, 2, 1, 2, 3,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 2, 2, 2, 2, 2, 2, 2, 2, 1, 4, 1, 2,
       2, 2, 2, 2, 2, 3, 1, 1, 2, 4, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2,
       2, 1, 2, 2, 4, 4, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 4, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 2, 2, 2, 2, 2, 2, 1,
       2, 2, 1, 2, 3, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 3, 4, 1, 2, 2, 3, 2,
       2, 1, 2, 2, 2, 2, 2, 3, 3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       4, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 4, 1, 2, 2, 2, 4, 2, 2, 1, 2,
       1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 4, 2, 2, 2, 2, 2, 2, 2, 2, 1, 4,
       2, 2, 2, 4, 1, 2, 3, 2, 2, 2, 4, 2, 2, 3, 1, 2, 2, 2, 2, 2, 4, 4,
       2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2, 1, 2, 2, 2,
       2, 2, 2, 2, 2, 3, 3, 2, 4, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 1, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 4, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 2, 2, 2, 2,
       3, 2, 2, 2, 1, 1, 2, 2, 1, 2, 2, 3, 3, 2, 2, 2, 2, 2, 1, 4, 2, 2,
       2, 2, 2, 2, 4, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 3, 2, 2, 3, 2, 2,
       2, 1, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 4, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 1, 2, 4, 2, 2, 2, 2, 2, 1, 2, 2, 3, 2, 2, 2,
       2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1,
       2, 2, 2, 2, 3, 2, 2, 3, 4, 4, 4, 4, 2, 2, 2, 2, 1, 2, 2, 2, 4, 2,
       2, 2, 2, 2, 2, 2, 4, 4, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2,
       1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 4, 1, 2, 2, 2, 2, 3, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 4, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 4,
       2, 2, 2, 2, 2, 2, 2, 4, 1, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 4,
       2, 2, 2, 4, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 4, 2, 2, 1, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 4,
       2, 2, 4, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 3, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 4, 2, 2, 2, 2, 2, 2, 2, 1, 4, 2, 2,
       2, 2, 1, 2, 2, 2, 2, 2, 4, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 2, 2,
       2, 2, 2, 1, 2, 2, 2, 2], dtype=int32)
```

```
fl_ward = fcluster(cl1,numclust,criterion='maxclust')
fl_ward
```

```
array([3, 3, 3, 3, 4, 3, 2, 4, 1, 1, 3, 4, 3, 3, 3, 2, 3, 3, 3, 3, 3, 1,
       3, 1, 3, 3, 3, 3, 3, 3, 2, 4, 1, 3, 1, 2, 3, 3, 3, 3, 3, 2, 3, 3,
       3, 3, 3, 3, 3, 2, 1, 4, 3, 2, 1, 1, 3, 3, 3, 3, 4, 3, 2, 3, 3, 4,
       3, 4, 2, 1, 3, 2, 4, 3, 1, 3, 3, 3, 3, 3, 1, 1, 3, 3, 3, 1, 3, 1,
       4, 4, 3, 3, 3, 3, 4, 2, 3, 1, 4, 3, 3, 3, 3, 3, 3, 3, 1, 2, 1, 3,
       4, 3, 3, 3, 3, 1, 1, 1, 3, 2, 3, 3, 3, 3, 3, 1, 3, 3, 4, 3, 3, 3,
       3, 1, 3, 4, 2, 2, 3, 4, 3, 3, 3, 4, 4, 3, 3, 3, 3, 3, 2, 3, 3, 3,
       4, 4, 3, 4, 3, 2, 3, 3, 3, 4, 3, 3, 3, 1, 1, 3, 3, 3, 3, 3, 3, 1,
       4, 3, 1, 3, 1, 3, 3, 3, 3, 3, 3, 1, 4, 3, 3, 4, 3, 3, 3, 3, 3, 4,
       4, 1, 3, 3, 3, 4, 3, 3, 4, 3, 3, 3, 4, 1, 3, 1, 2, 1, 3, 3, 1, 2,
       3, 1, 3, 3, 3, 3, 4, 1, 1, 3, 3, 3, 3, 4, 3, 2, 3, 4, 4, 3, 4, 3,
       2, 1, 3, 3, 2, 3, 3, 3, 3, 3, 3, 3, 2, 1, 3, 3, 4, 2, 3, 3, 1, 3,
       1, 3, 3, 4, 3, 3, 3, 3, 3, 4, 3, 2, 2, 4, 2, 3, 3, 3, 3, 3, 1, 2,
       4, 3, 3, 2, 1, 2, 1, 4, 3, 3, 2, 3, 3, 1, 1, 3, 2, 3, 3, 3, 2, 2,
       3, 3, 3, 4, 3, 3, 3, 3, 1, 3, 4, 3, 3, 1, 4, 2, 1, 3, 1, 4, 3, 3,
       3, 4, 3, 3, 3, 1, 1, 3, 2, 3, 3, 3, 3, 3, 3, 4, 3, 3, 3, 4, 4, 4,
       3, 3, 3, 3, 4, 2, 3, 1, 3, 1, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       2, 3, 3, 3, 4, 3, 4, 3, 3, 2, 3, 3, 3, 3, 3, 3, 4, 2, 3, 3, 3, 4,
       3, 3, 3, 3, 4, 3, 3, 1, 2, 3, 3, 3, 4, 3, 3, 3, 2, 1, 3, 3, 3, 4,
       1, 4, 2, 3, 1, 1, 4, 3, 1, 3, 3, 1, 1, 3, 3, 3, 3, 3, 1, 2, 3, 4,
       3, 3, 3, 3, 2, 2, 3, 3, 3, 3, 3, 3, 1, 3, 3, 3, 1, 3, 3, 1, 4, 3,
       3, 1, 4, 3, 4, 1, 3, 3, 3, 1, 3, 2, 2, 3, 2, 4, 3, 3, 2, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 1, 3, 2, 3, 3, 3, 3, 3, 1, 3, 3, 1, 3, 3, 3,
       4, 3, 3, 1, 3, 3, 3, 3, 3, 4, 4, 3, 4, 3, 3, 3, 3, 1, 1, 3, 4, 3,
       4, 3, 3, 3, 3, 3, 3, 3, 3, 3, 1, 1, 4, 3, 3, 4, 3, 3, 3, 4, 3, 1,
       4, 3, 3, 3, 1, 3, 3, 1, 2, 2, 2, 2, 4, 3, 3, 3, 1, 4, 3, 3, 2, 3,
       3, 3, 4, 3, 3, 3, 2, 2, 4, 3, 3, 3, 3, 3, 3, 1, 3, 3, 3, 3, 1, 3,
       1, 3, 3, 4, 4, 3, 3, 1, 3, 3, 3, 2, 1, 3, 3, 3, 3, 1, 4, 3, 3, 3,
       3, 4, 3, 4, 3, 3, 2, 3, 3, 3, 4, 4, 4, 4, 3, 4, 3, 3, 1, 3, 3, 2,
       3, 3, 3, 4, 3, 3, 3, 2, 1, 3, 1, 3, 3, 3, 3, 3, 3, 3, 4, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 3, 3, 4, 3, 4, 3, 3, 3, 4, 4, 2,
       4, 3, 3, 2, 3, 3, 3, 4, 3, 4, 3, 3, 4, 3, 1, 2, 3, 4, 1, 3, 3, 3,
       3, 3, 4, 3, 4, 3, 3, 1, 3, 3, 3, 4, 3, 4, 3, 3, 3, 3, 3, 3, 1, 2,
       2, 3, 2, 3, 3, 3, 3, 3, 3, 4, 3, 3, 3, 3, 1, 3, 3, 4, 4, 3, 1, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 4, 4, 2, 3, 3, 4, 3, 3, 3, 4, 1, 2, 3, 3,
       3, 3, 1, 3, 2, 3, 4, 3, 2, 1, 3, 3, 3, 3, 3, 3, 3, 3, 1, 1, 3, 4,
       3, 3, 3, 1, 3, 3, 3, 4], dtype=int32)
```

```
def mean(deneme, clusters_list):
    cluster_means = []
    for cluster_index in clusters_list:
        cluster_data = deneme.iloc[cluster_index]
        mean = np.mean(cluster_data, axis=0)
        cluster_means.append(mean)
    return cluster_means
```

```
means_fl_centroid = mean(df_combined, fl_centroid)
means_fl_median = mean(df_combined, fl_median)
means_fl_ward = mean(df_combined, fl_ward)
```

```
means_fl_ward
```

```
[4.016609270963945,
 4.016609270963945,
 4.016609270963945,
 4.016609270963945,
 3.627595751837112,
 4.016609270963945,
 3.7626378802570923,
 3.627595751837112,
 3.576157208491392,
 3.576157208491392,
 4.016609270963945,
 3.627595751837112,
 4.016609270963945,
 4.016609270963945,
 4.016609270963945,
 3.7626378802570923,
 4.016609270963945,
 4.016609270963945,
 4.016609270963945,
 4.016609270963945,
 4.016609270963945,
 3.576157208491392,
 4.016609270963945,
 3.576157208491392,
 4.016609270963945,
 4.016609270963945,
 4.016609270963945,
 4.016609270963945,
 4.016609270963945,
 4.016609270963945,
 3.7626378802570923,
 3.627595751837112,
 3.576157208491392,
 4.016609270963945,
 3.576157208491392,
 3.7626378802570923,
 4.016609270963945,
 4.016609270963945,
 4.016609270963945,
 4.016609270963945,
 4.016609270963945,
 3.7626378802570923,
 4.016609270963945,
 4.016609270963945,
 4.016609270963945,
```

```
     4.016609270963945,
     4.016609270963945,
     4.016609270963945,
     4.016609270963945,
     3.7626378802570923,
     3.576157208491392,
     3.627595751837112,
     4.016609270963945,
     3.7626378802570923,
     3.576157208491392,
     3.576157208491392,
     4.016609270963945,
     4.016609270963945,
     4.016609270963945
```

```
means_fl_median
```

```
     3.7626378802570923,
     3.627595751837112,
     3.7626378802570923,
     3.7626378802570923,
     3.7626378802570923,
     3.7626378802570923,
     3.7626378802570923,
     3.7626378802570923,
     3.7626378802570923,
     3.7626378802570923,
     3.7626378802570923,
     3.7626378802570923,
     3.7626378802570923,
     3.576157208491392,
     3.7626378802570923,
     3.7626378802570923,
     3.7626378802570923,
     3.7626378802570923,
     3.7626378802570923,
     4.016609270963945,
     3.7626378802570923,
     3.7626378802570923,
     3.7626378802570923,
     3.7626378802570923,
     3.7626378802570923,
     3.7626378802570923,
     3.7626378802570923,
     3.7626378802570923,
     3.7626378802570923,
     3.7626378802570923,
     3.7626378802570923,
     3.627595751837112,
     3.7626378802570923,
     3.7626378802570923,
     3.7626378802570923,
     3.7626378802570923,
```

```
     3.7626378802570923,
     3.7626378802570923,
     3.7626378802570923,
     3.576157208491392,
     3.627595751837112,
     3.7626378802570923,
     3.7626378802570923,
     3.7626378802570923,
     3.7626378802570923,
     3.576157208491392,
     3.7626378802570923,
     3.7626378802570923,
     3.7626378802570923,
     3.7626378802570923,
     3.7626378802570923,
     3.627595751837112,
     3.576157208491392,
     3.7626378802570923,
     3.7626378802570923,
     3.7626378802570923,
     3.7626378802570923,
     3.7626378802570923,
     3.7626378802570923,
     3.7626378802570923
```

```
means_fl_centroid
```

```
     3.7626378802570923,
     3.7626378802570923,
     3.7626378802570923,
     3.7626378802570923,
     3.7626378802570923,
     3.7626378802570923,
     3.7626378802570923,
     3.7626378802570923,
     3.7626378802570923,
     4.016609270963945,
     3.7626378802570923,
     3.7626378802570923,
     3.7626378802570923,
     3.7626378802570923,
     3.7626378802570923,
     3.7626378802570923,
     3.7626378802570923,
     3.7626378802570923,
     3.7626378802570923,
     3.7626378802570923,
     3.7626378802570923,
     3.576157208491392,
     3.7626378802570923,
     3.7626378802570923,
     4.016609270963945,
     3.7626378802570923,
     3.7626378802570923,
```

```
 3.7626378802570923,
 3.7626378802570923,
 3.7626378802570923,
 3.7626378802570923,
 3.7626378802570923,
 4.016609270963945,
 3.576157208491392,
 3.7626378802570923,
 3.576157208491392,
 3.7626378802570923,
 3.7626378802570923,
 3.7626378802570923,
 3.7626378802570923,
 3.7626378802570923,
 3.7626378802570923,
 3.7626378802570923,
 3.7626378802570923,
 3.7626378802570923,
 3.7626378802570923,
 3.7626378802570923,
 3.7626378802570923,
 3.7626378802570923,
 3.7626378802570923,
 3.7626378802570923,
 3.7626378802570923,
 3.7626378802570923,
 3.7626378802570923,
 3.7626378802570923,
 3.7626378802570923,
 3.7626378802570923,
 3.7626378802570923,
 3.7626378802570923,
 3.7626378802570923
```

```python
def covariance(deneme, clusters):
    covariances = []
    for cluster_index in clusters:
        cluster_data = deneme.iloc[cluster_index]
        covariance = np.cov(cluster_data, rowvar=False)
        covariances.append(covariance)
    return covariances


covariance_centroid = covariance(df_combined, fl_centroid)
covariance_centroid
```

```
[array(7.29270138),
 array(7.29270138),
 array(7.29270138),
 array(7.29270138),
 array(7.29270138),
 array(7.29270138),
 array(9.12193409),
```

```
        array(7.29270138),
        array(6.19701633),
        array(4.86209838),
        array(7.29270138),
        array(7.29270138),
        array(7.29270138),
        array(7.29270138),
        array(7.29270138),
        array(9.12193409),
        array(7.29270138),
        array(7.29270138),
        array(7.29270138),
        array(7.29270138),
        array(7.29270138),
        array(4.86209838),
        array(7.29270138),
        array(4.86209838),
        array(7.29270138),
        array(7.29270138),
        array(7.29270138),
        array(7.29270138),
        array(7.29270138),
        array(7.29270138),
        array(9.12193409),
        array(7.29270138),
        array(4.86209838),
        array(7.29270138),
        array(4.86209838),
        array(9.12193409),
        array(7.29270138),
        array(7.29270138),
        array(7.29270138),
        array(7.29270138),
        array(7.29270138),
        array(9.12193409),
        array(7.29270138),
        array(7.29270138),
        array(7.29270138),
        array(7.29270138),
        array(7.29270138),
        array(7.29270138),
        array(7.29270138),
        array(9.12193409),
        array(4.86209838),
        array(7.29270138),
        array(7.29270138),
        array(9.12193409),
        array(4.86209838),
        array(4.86209838),
        array(7.29270138),
        array(7.29270138),
        array(7.29270138),
```

```
covariance_median = covariance(df_combined, fi_median)
covariance_median
```

```
     array(7.29270138),
     array(7.29270138),
     array(7.29270138),
     array(7.29270138),
     array(7.29270138),
     array(9.12193409),
     array(9.12193409),
     array(7.29270138),
     array(7.29270138),
     array(7.29270138),
     array(7.29270138),
     array(7.29270138),
     array(7.29270138),
     array(4.86209838),
     array(7.29270138),
     array(7.29270138),
     array(4.86209838),
     array(7.29270138),
     array(9.12193409),
     array(7.29270138),
     array(7.29270138),
     array(7.29270138),
     array(7.29270138),
     array(7.29270138),
     array(7.29270138),
     array(4.86209838),
     array(7.29270138),
     array(7.29270138),
     array(7.29270138),
     array(7.29270138),
     array(7.29270138),
     array(7.29270138),
     array(7.29270138),
     array(7.29270138),
     array(7.29270138),
     array(7.29270138),
     array(7.29270138),
     array(4.86209838),
     array(7.29270138),
     array(7.29270138),
     array(7.29270138),
     array(7.29270138),
     array(7.29270138),
     array(7.29270138),
     array(7.29270138),
     array(7.29270138),
     array(7.29270138),
     array(7.29270138),
     array(7.29270138),
     array(4.86209838),
     array(7.29270138),
```

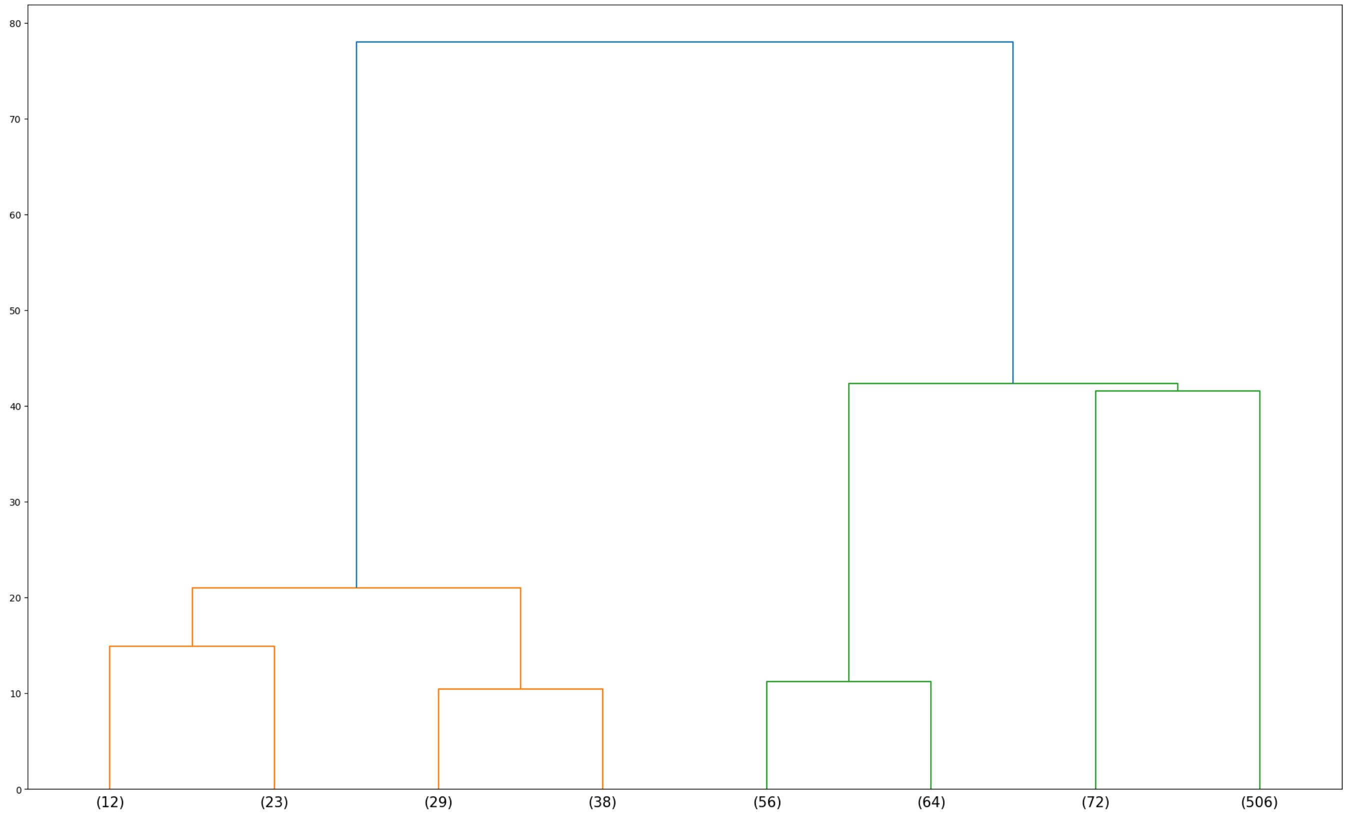
```
      array(9.12193409),
      array(6.19701633),
      array(4.86209838),
      array(7.29270138),
      array(7.29270138),
      array(9.12193409),
      array(7.29270138),
      array(7.29270138),
      array(4.86209838)
```

```
covariance_ward = covariance(df_combined, fl_ward)
covariance_ward
```

```
      array(6.19701633),
      array(9.12193409),
      array(9.12193409),
      array(9.12193409),
      array(9.12193409),
      array(6.19701633),
      array(7.29270138),
      array(9.12193409),
      array(4.86209838),
      array(6.19701633),
      array(9.12193409),
      array(9.12193409),
      array(9.12193409),
      array(9.12193409),
      array(9.12193409),
      array(9.12193409),
      array(9.12193409),
      array(4.86209838),
      array(7.29270138),
      array(4.86209838),
      array(9.12193409),
      array(6.19701633),
      array(9.12193409),
      array(9.12193409),
      array(9.12193409),
      array(9.12193409),
      array(4.86209838),
      array(4.86209838),
      array(4.86209838),
      array(9.12193409),
      array(7.29270138),
      array(9.12193409),
      array(9.12193409),
      array(9.12193409),
      array(9.12193409),
      array(9.12193409),
      array(4.86209838),
      array(9.12193409),
      array(9.12193409),
      array(6.19701633),
      array(9.12193409),
```

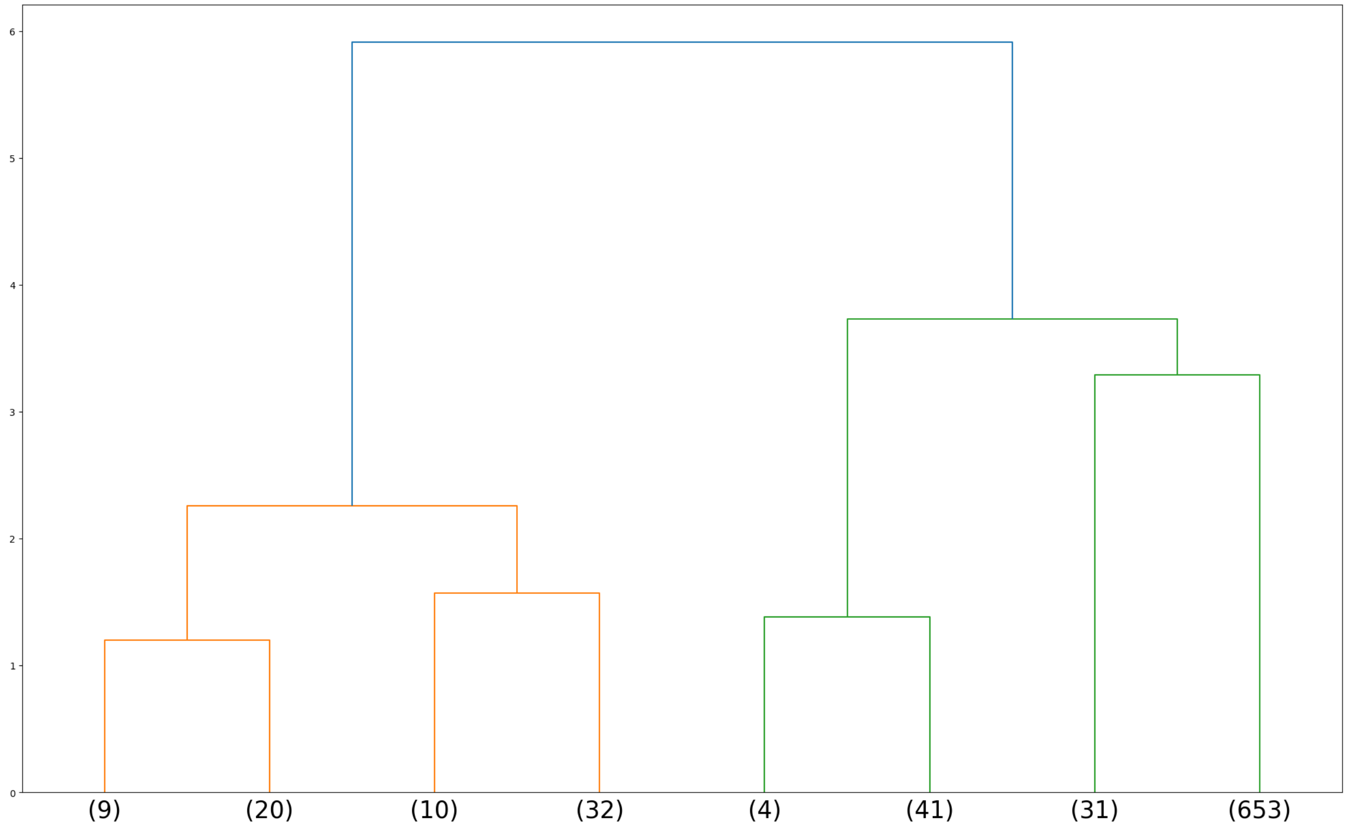
```
      array(9.12193409),
      array(9.12193409),
      array(9.12193409),
      array(4.86209838),
      array(9.12193409),
      array(6.19701633),
      array(7.29270138),
      array(7.29270138),
      array(9.12193409),
      array(6.19701633),
      array(9.12193409),
      array(9.12193409),
      array(9.12193409),
      array(6.19701633),
      array(6.19701633),
      array(9.12193409),
      array(9.12193409),
      array(9.12193409),
      array(9.12193409)
```

```
from matplotlib import pyplot as plt
fig = plt.figure(figsize=(25, 15))
dendrogram(
  Z = dict_linkage['ward'],
  count_sort = True,
  truncate_mode='level', p=2,
  distance_sort = True,
  orientation = 'top',
  leaf_font_size = 15)
plt.show()
```




```
from matplotlib import pyplot as plt
fig = plt.figure(figsize=(25, 15))
dendrogram(
  Z = dict_linkage['median'],
```

```
    truncate_mode='level', p=2,
    count_sort = True,
    distance_sort = True,
    orientation = 'top',
    leaf_font_size = 25)
plt.show()
```


6
5
4
3
2
1
0
(9)
(20)
(10)
(32)
(4)
(41)
(31)
(653)

```
import pandas as pd
import numpy as np

# Number of data points
num_data_points = 2000

# Number of variables
num_variables = 8

# Probability of success for each variable (adjust as needed)
prob_success = 0.5

# Generate a DataFrame with random Bernoulli variables
data = np.random.choice([0, 1], size=(num_data_points, num_variables), p=[1 - p
columns = [f'NewFeature{i+8}' for i in range(1, num_variables + 1)]
df_bernoulli = pd.DataFrame(data, columns=columns)

# Display the DataFrame
print(df_bernoulli.head())
```

```
   NewFeature9  NewFeature10  NewFeature11  NewFeature12  NewFeature13  \
0            0             1             1             0             0
1            1             1             0             0             0
2            1             0             0             1             0
3            1             0             1             1             0
4            1             0             0             0             0

   NewFeature14  NewFeature15  NewFeature16
0             0             1             1
1             0             1             1
2             0             0             0
3             1             0             0
4             0             0             0
```

```
df_comb = pd.concat([df_ca, df_bernoulli], axis=1)
df_comb
```

	NewFeature1	NewFeature2	NewFeature3	NewFeature4	NewFeature5	NewFea
0	1.496714	1.861736	3.647689	5.523030	4.765847	5.
1	0.530526	2.542560	2.536582	3.534270	5.241962	4.
2	-0.012831	2.314247	2.091976	2.587696	6.465649	5.
3	0.455617	2.110923	1.849006	4.375698	4.399361	5.
4	0.986503	0.942289	3.822545	2.779156	5.208864	4.
...	...	...	...	...	...	
1995	1.107273	0.444435	2.110790	0.105510	0.339918	-2.
1996	0.061788	1.094171	4.761031	1.584071	-2.388226	6.
1997	-0.356542	2.707558	-0.721904	-3.005897	2.531872	-3.
1998	0.636953	-0.758819	-0.316832	1.549028	-2.732980	-1.
1999	0.565200	1.925869	-0.489812	3.380723	1.051830	0.

2000 rows × 17 columns

```
df_comb
```

	NewFeature1	NewFeature2	NewFeature3	NewFeature4	NewFeature5	NewFea
0	1.496714	1.861736	3.647689	5.523030	4.765847	5.
1	0.530526	2.542560	2.536582	3.534270	5.241962	4.
2	-0.012831	2.314247	2.091976	2.587696	6.465649	5.
3	0.455617	2.110923	1.849006	4.375698	4.399361	5.
4	0.986503	0.942289	3.822545	2.779156	5.208864	4.
...	...	...	...	...	...	
1995	1.107273	0.444435	2.110790	0.105510	0.339918	-2.
1996	0.061788	1.094171	4.761031	1.584071	-2.388226	6.
1997	-0.356542	2.707558	-0.721904	-3.005897	2.531872	-3.
1998	0.636953	-0.758819	-0.316832	1.549028	-2.732980	-1.
1999	0.565200	1.925869	-0.489812	3.380723	1.051830	0.

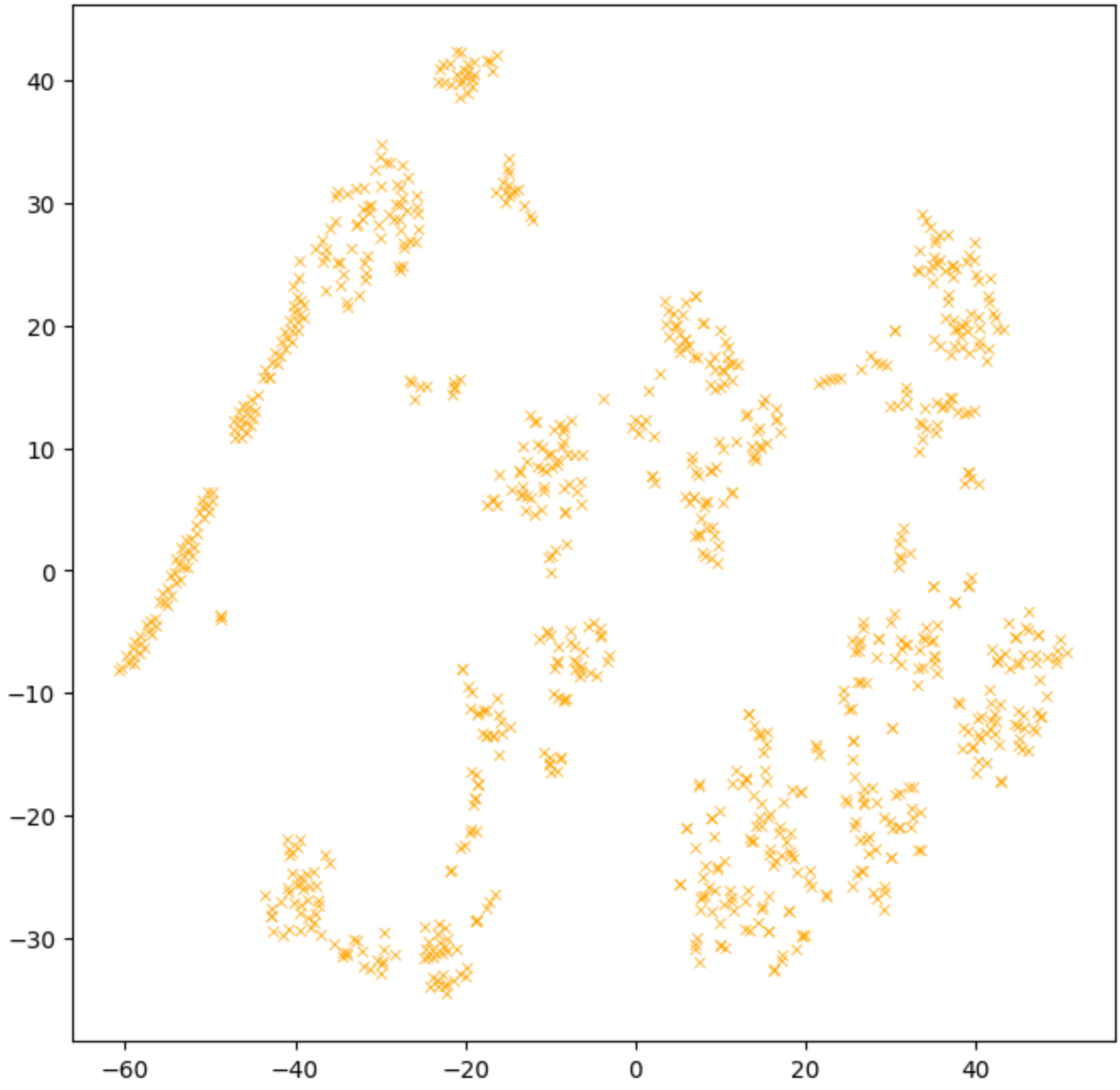
2000 rows × 17 columns

The next plot shows a two-dimensional projection of the test example features. The color of the points represent the label values. Note that the label values were not available to the model.

```
fig, ax = plt.subplots(1, 1, figsize=(8, 8))
ax.grid(False)

# Color the points according to the label value.
colors = (test["class"] == "").map(lambda x: ["orange",'green'][x])
ax.scatter(
    t_sne[:, 0], t_sne[:, 1], c=colors, linewidths=0.5, marker="x", s=20)
```

```
<matplotlib.collections.PathCollection at 0x7dd61d303c40>
```



With bernouli

```
df_comb
```

	NewFeature1	NewFeature2	NewFeature3	NewFeature4	NewFeature5	NewFea
0	1.496714	1.861736	3.647689	5.523030	4.765847	5.
1	0.530526	2.542560	2.536582	3.534270	5.241962	4.
2	-0.012831	2.314247	2.091976	2.587696	6.465649	5.
3	0.455617	2.110923	1.849006	4.375698	4.399361	5.
4	0.986503	0.942289	3.822545	2.779156	5.208864	4.
...	...	...	...	...	...	
1995	1.107273	0.444435	2.110790	0.105510	0.339918	-2.
1996	0.061788	1.094171	4.761031	1.584071	-2.388226	6.
1997	-0.356542	2.707558	-0.721904	-3.005897	2.531872	-3.
1998	0.636953	-0.758819	-0.316832	1.549028	-2.732980	-1.
1999	0.565200	1.925869	-0.489812	3.380723	1.051830	0.

2000 rows × 9 columns

```
df_co
```

	NewFeature1	NewFeature2	NewFeature3	NewFeature4	NewFeature5	NewFea
0	1.496714	1.861736	3.647689	5.523030	4.765847	5.
1	0.530526	2.542560	2.536582	3.534270	5.241962	4.
2	-0.012831	2.314247	2.091976	2.587696	6.465649	5.
3	0.455617	2.110923	1.849006	4.375698	4.399361	5.
4	0.986503	0.942289	3.822545	2.779156	5.208864	4.
...	...	...	...	...	...	
3995	4.846778	2.820158	8.505003	3.644273	5.363558	6.
3996	5.094596	5.379973	-2.362289	0.568864	5.822160	7.
3997	11.805574	6.011951	4.223961	2.810333	6.465221	8.
3998	5.937028	2.601207	5.702064	-1.297167	5.151158	4.
3999	4.784226	5.759215	4.194763	4.872505	0.359952	5.

4000 rows × 9 columns

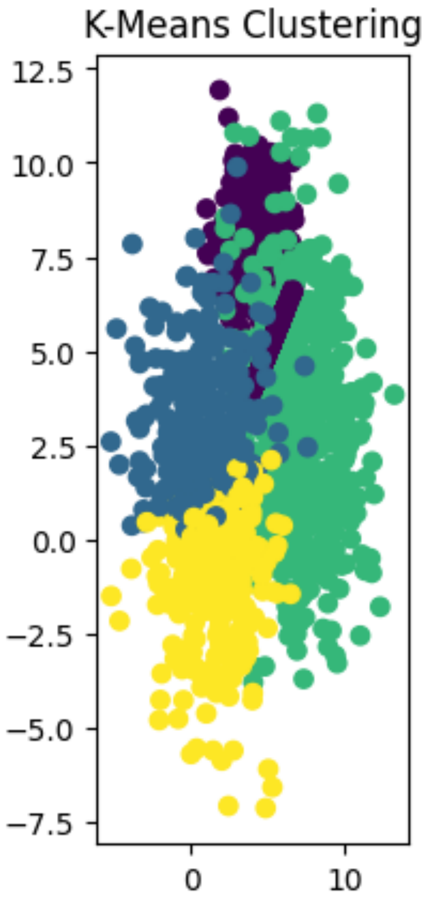
```
from sklearn.cluster import KMeans, AgglomerativeClustering

kmeans = KMeans(n_clusters=4, random_state=0)
clusters_kmeans = kmeans.fit_predict(df_comb)
clusters_kmeans
```

```
array([0, 0, 0, ..., 1, 3, 1], dtype=int32)
```

```
plt.subplot(132)
plt.scatter(df_comb['NewFeature4'], df_comb['NewFeature8'], c=clusters_kmeans,
plt.title("K-Means Clustering")
```

```
Text(0.5, 1.0, 'K-Means Clustering')
```




```
def mean(deneme, clusters_list):
    cluster_means = []
    for cluster_index in clusters_list:
        cluster_data = deneme.iloc[cluster_index]
        mean = np.mean(cluster_data, axis=0)
        cluster_means.append(mean)
    return cluster_means
```

```
kmeans_means = mean(df_comb, clusters_kmeans)
```

```
kmeans_means
```

```
 2.1096318189596377,
 2.1096318189596377,
 2.1096318189596377,
 2.1096318189596377,
 2.1096318189596377,
```

```
         2.1096318189596377,
```

```
kmeans_means[-1]
```

```
2.1285538162601485
```

```
kmeans_means[-501]
```

```
2.6710309093727385
```

```
kmeans_means[-1001]
```

```
2.1096318189596377
```

```
kmeans_means[-1501]
```

```
2.6710309093727385
```

```
kmeans_covariances
```

```
 array(7.10791743),
 array(7.10791743),
 array(7.10791743),
 array(7.10791743),
 array(7.10791743),
 array(7.10791743),
 array(7.10791743),
 array(7.10791743),
 array(7.10791743),
 array(7.10791743),
 array(7.10791743),
 array(7.10791743),
 array(7.10791743),
 array(7.10791743),
 array(7.10791743),
 array(7.10791743),
 array(7.10791743),
 array(7.10791743),
 array(7.10791743),
 array(7.10791743),
 array(7.10791743),
 array(7.10791743),
 array(7.10791743),
 array(7.10791743),
 array(7.10791743),
 array(7.10791743),
 array(7.10791743),
 array(7.10791743),
 array(7.10791743),
```

```
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
       array(7.10791743),
```

```
import pandas as pd
import numpy as np

# Number of data points
num_data_points = 4000

# Number of variables
num_variables = 8

# Probability of success for each variable (adjust as needed)
prob_success = 0.5

# Generate a DataFrame with random Bernoulli variables
data = np.random.choice([0, 1], size=(num_data_points, num_variables), p=[1 - p
columns = [f'NewFeature{i+8}' for i in range(1, num_variables + 1)]
df_bernoulli = pd.DataFrame(data, columns=columns)

# Display the DataFrame
print(df_bernoulli.head())
```

```
   NewFeature9  NewFeature10  NewFeature11  NewFeature12  NewFeature13  \
0            1             0             0             1             1
1            0             1             1             0             0
2            0             1             1             1             1
3            0             0             0             0             0
4            1             0             0             1             1

   NewFeature14  NewFeature15  NewFeature16
0             1             1             1
1             0             0             0
2             0             1             1
3             0             0             0
4             0             1             0
```

```
df_co = pd.concat([df_co, df_bernoulli], axis=1)
```

```
df_co
```

	NewFeature1	NewFeature2	NewFeature3	NewFeature4	NewFeature5	NewFea
0	1.496714	1.861736	3.647689	5.523030	4.765847	5.
1	0.530526	2.542560	2.536582	3.534270	5.241962	4.
2	-0.012831	2.314247	2.091976	2.587696	6.465649	5.
3	0.455617	2.110923	1.849006	4.375698	4.399361	5.
4	0.986503	0.942289	3.822545	2.779156	5.208864	4.
...	...	...	...	...	...	
3995	4.846778	2.820158	8.505003	3.644273	5.363558	6.
3996	5.094596	5.379973	-2.362289	0.568864	5.822160	7.
3997	11.805574	6.011951	4.223961	2.810333	6.465221	8.
3998	5.937028	2.601207	5.702064	-1.297167	5.151158	4.
3999	4.784226	5.759215	4.194763	4.872505	0.359952	5.

4000 rows × 17 columns

```
df_combined = df_co
```

```
from sklearn.model_selection import train_test_split


train, test = train_test_split(df_combined, test_size=0.2, random_state=0)
```

```
test
```

	NewFeature1	NewFeature2	NewFeature3	NewFeature4	NewFeature5	NewFea
2230	1.685870	0.444104	2.664354	4.564048	-0.775611	6.
668	10.375456	9.868003	8.058941	3.002222	1.077870	4.
3616	0.289670	8.716253	5.606850	0.188825	5.247081	5.
2363	0.122017	2.579291	2.058499	4.676469	4.790272	1.
142	2.189470	0.772392	3.597400	4.701173	4.702436	7.
...	...	...	...	...	...	
1118	5.698623	5.698623	5.698623	5.698623	5.698623	5.
3572	12.343654	1.243236	5.251684	0.132305	10.989783	-4.
2482	0.324822	8.365665	2.605373	4.727725	4.059877	2.
643	9.147775	12.218503	8.563917	8.493223	5.527330	4.
299	-0.496529	1.349976	2.916562	2.550355	4.078140	4.

800 rows × 17 columns

```
train
```

	NewFeature1	NewFeature2	NewFeature3	NewFeature4	NewFeature5	NewFea
1161	4.985772	4.985772	4.985772	4.985772	4.985772	4.
2355	10.739180	2.418899	9.255877	2.368724	4.834632	6.
1831	2.112089	-3.596666	-2.439475	-3.796326	-2.582678	4.
156	1.233786	0.444104	3.330880	4.833529	3.006264	6.
195	0.978633	1.252788	0.575760	4.884045	5.736844	5.
...	...	...	...	...	...	
835	9.074608	8.270400	8.248214	7.032250	5.885695	6.
3264	-0.709029	0.569225	5.376283	3.002222	9.355987	4.
1653	0.169403	-1.269536	1.083356	-0.172570	-2.933132	1.
2607	3.898921	0.209714	1.142117	4.282009	5.908440	5.
2732	1.897066	2.010233	0.340587	9.499421	5.256578	5.

3200 rows × 17 columns

```
# , and convert it into a TensorFlow dataset.
train_ds = tfdf.keras.pd_dataframe_to_tf_dataset(train, label="class")
test_ds = tfdf.keras.pd_dataframe_to_tf_dataset(test, label="class")
```

```
train
```

	NewFeature1	NewFeature2	NewFeature3	NewFeature4	NewFeature5	NewFea
1161	4.985772	4.985772	4.985772	4.985772	4.985772	4.
2355	10.739180	2.418899	9.255877	2.368724	4.834632	6.
1831	2.112089	-3.596666	-2.439475	-3.796326	-2.582678	4.
156	1.233786	0.444104	3.330880	4.833529	3.006264	6.
195	0.978633	1.252788	0.575760	4.884045	5.736844	5.
...	...	...	...	...	...	
835	9.074608	8.270400	8.248214	7.032250	5.885695	6.
3264	-0.709029	0.569225	5.376283	3.002222	9.355987	4.
1653	0.169403	-1.269536	1.083356	-0.172570	-2.933132	1.
2607	3.898921	0.209714	1.142117	4.282009	5.908440	5.
2732	1.897066	2.010233	0.340587	9.499421	5.256578	5.

3200 rows × 17 columns

Following are the first five examples of the training dataset. Notice that different columns represent different quantities. For example, how would you compare the distance between *relationship* and *age*?

```
# Print the first 5 examples.
train.head()
```

	NewFeature1	NewFeature2	NewFeature3	NewFeature4	NewFeature5	NewFea
1161	4.985772	4.985772	4.985772	4.985772	4.985772	4.
2355	10.739180	2.418899	9.255877	2.368724	4.834632	6.
1831	2.112089	-3.596666	-2.439475	-3.796326	-2.582678	4.
156	1.233786	0.444104	3.330880	4.833529	3.006264	6.
195	0.978633	1.252788	0.575760	4.884045	5.736844	5.

A Random Forest is trained as follows:

```
# Train a Random Forest
model = tfdf.keras.RandomForestModel(num_trees=1000)
model.fit(train_ds)
```

```
Use /tmp/tmpm2bybnyy as temporary training directory
Reading training dataset...
Training dataset read in 0:00:00.418791. Found 3200 examples.
Training model...
Model trained in 0:00:11.378140
Compiling model...
Model compiled.
<keras.src.callbacks.History at 0x7dd606c36ad0>
```

The performance of the Random Forest model is:

```
model_inspector = model.make_inspector()
out_of_bag_accuracy = model_inspector.evaluation().accuracy
print(f"Out-of-bag accuracy: {out_of_bag_accuracy:.4f}")
```

```
Out-of-bag accuracy: 0.9666
```

This is an expected accuracy value for Random Forest models on this dataset. It indicates that the model is correctly trained.

We can also measure the accuracy of the model on the test datasets:

```
# The test accuracy is measured on the test datasets.
model.compile(["accuracy"])
test_accuracy = model.evaluate(test_ds, return_dict=True, verbose=0)["accuracy'
print(f"Test accuracy: {test_accuracy:.4f}")
```

```
Test accuracy: 0.9550
```

Proximities

First, we inspect the number of trees in the model and the number of examples in the test datasets.

```
print("The model contains", model_inspector.num_trees(), "trees.")
print("The test dataset contains", test.shape[0], "examples.")
```

```
The model contains 1000 trees.
The test dataset contains 800 examples.
```

```
test
```

	NewFeature1	NewFeature2	NewFeature3	NewFeature4	NewFeature5	NewFea
2230	1.685870	0.444104	2.664354	4.564048	-0.775611	6.
668	10.375456	9.868003	8.058941	3.002222	1.077870	4.
3616	0.289670	8.716253	5.606850	0.188825	5.247081	5.
2363	0.122017	2.579291	2.058499	4.676469	4.790272	1.
142	2.189470	0.772392	3.597400	4.701173	4.702436	7.
...	...	...	...	...	...	
1118	5.698623	5.698623	5.698623	5.698623	5.698623	5.
3572	12.343654	1.243236	5.251684	0.132305	10.989783	-4.
2482	0.324822	8.365665	2.605373	4.727725	4.059877	2.
643	9.147775	12.218503	8.563917	8.493223	5.527330	4.
299	-0.496529	1.349976	2.916562	2.550355	4.078140	4.

800 rows × 17 columns

The method predict_get_leaves() returns the index of the active leaf for each example and each tree.

```
leaves = model.predict_get_leaves(test_ds)
print("The leaf indices:\n", leaves)
```

```
The leaf indices:
 [[ 25  59   8 ...  15  90  60]
 [ 31 162 121 ...  76 111  90]
 [  8 151  45 ...  54  56 100]
 ...
 [102 151  44 ...  55 125  64]
 [124 164  61 ...  86 140 119]
 [ 82  68   7 ...  42 106  39]]
```

```
print("The predicted leaves have shape", leaves.shape,
      "(we expect [num_examples, num_trees]")
```

```
The predicted leaves have shape (800, 1000) (we expect [num_examples, num_t
```

Double-click (or enter) to edit

```
def compute_proximity(leaves, step_size=100):
  """Computes the proximity between each pair of examples.



  Returns:
    The example pair-wise proximity matrix of shape [n,n] with "n" the number c
    examples.
  """

  example_idx = 0
  num_examples = leaves.shape[0]
  t_leaves = np.transpose(leaves)
  proximities = []

  # Instead of computing the proximity in between all the examples at the same
  # time, we compute the similarity in blocks of "step_size" examples. This
  # makes the code more efficient with the the numpy broadcast.
  while example_idx < num_examples:
    end_idx = min(example_idx + step_size, num_examples)
    proximities.append(
        np.mean(
            leaves[..., np.newaxis] == t_leaves[:,
                                                 example_idx:end_idx][np.newaxis
                                                                     ...],
            axis=1))
    example_idx = end_idx
  return np.concatenate(proximities, axis=1)


proximity = compute_proximity(leaves)
print("The shape of proximity is", proximity.shape)
```

```
The shape of proximity is (800, 800)
```

Double-click (or enter) to edit

```
proximity
```

```
array([[1.   , 0.014, 0.002, ..., 0.012, 0.   , 0.02 ],
       [0.014, 1.   , 0.009, ..., 0.004, 0.002, 0.   ],
       [0.002, 0.009, 1.   , ..., 0.187, 0.   , 0.   ],
       ...,
       [0.012, 0.004, 0.187, ..., 1.   , 0.   , 0.   ],
       [0.   , 0.002, 0.   , ..., 0.   , 1.   , 0.   ],
       [0.02 , 0.   , 0.   , ..., 0.   , 0.   , 1.   ]])
```

The proximity matrix has several interesting properties, notably, it is symmetrical, positive, and the diagonal elements are all 1.

```
distance = 1 - proximity

t_sne = TSNE(
    # Number of dimensions to display. 3d is also possible.
    n_components=2,
    # Control the shape of the projection. Higher values create more
    # distinct but also more collapsed clusters. Can be in 5-50.
    perplexity=20,
    metric="precomputed",
    init="random",
    verbose=1,
    learning_rate="auto").fit_transform(distance)
```

```
[t-SNE] Computing 61 nearest neighbors...
[t-SNE] Indexed 800 samples in 0.003s...
[t-SNE] Computed neighbors for 800 samples in 0.019s...
[t-SNE] Computed conditional probabilities for sample 800 / 800
[t-SNE] Mean sigma: 0.252337
[t-SNE] KL divergence after 250 iterations with early exaggeration: 63.6637
[t-SNE] KL divergence after 1000 iterations: 0.860965
```

```
distance
```

```
array([[0.   , 0.986, 0.998, ..., 0.988, 1.   , 0.98 ],
       [0.986, 0.   , 0.991, ..., 0.996, 0.998, 1.   ],
       [0.998, 0.991, 0.   , ..., 0.813, 1.   , 1.   ],
       ...,
       [0.988, 0.996, 0.813, ..., 0.   , 1.   , 1.   ],
       [1.   , 0.998, 1.   , ..., 1.   , 0.   , 1.   ],
       [0.98 , 1.   , 1.   , ..., 1.   , 1.   , 0.   ]])
```

```
from scipy.cluster.hierarchy import linkage,dendrogram,fcluster
dict_linkage = {}
list_method= ['centroid','ward','median']

for k in list_method:
    dict_linkage[k] = linkage(
      y = distance,
      method = k,
      optimal_ordering = True)
```

```
<ipython-input-315-f10fa8e46781>:6: ClusterWarning:

scipy.cluster: The symmetric non-negative hollow observation matrix looks s
```

```
dict_linkage['ward']
```

```
array([[2.28000000e+02, 5.23000000e+02, 2.00157438e-01, 2.00000000e+00],
       [4.30000000e+02, 1.68000000e+02, 2.31721816e-01, 2.00000000e+00],
       [8.50000000e+01, 4.71000000e+02, 2.46748860e-01, 2.00000000e+00],
       ...,
       [1.59500000e+03, 1.57200000e+03, 2.78503103e+01, 5.82000000e+02],
       [1.59200000e+03, 1.59600000e+03, 2.82066088e+01, 7.10000000e+02],
       [1.59400000e+03, 1.59700000e+03, 5.76742682e+01, 8.00000000e+02]])
```

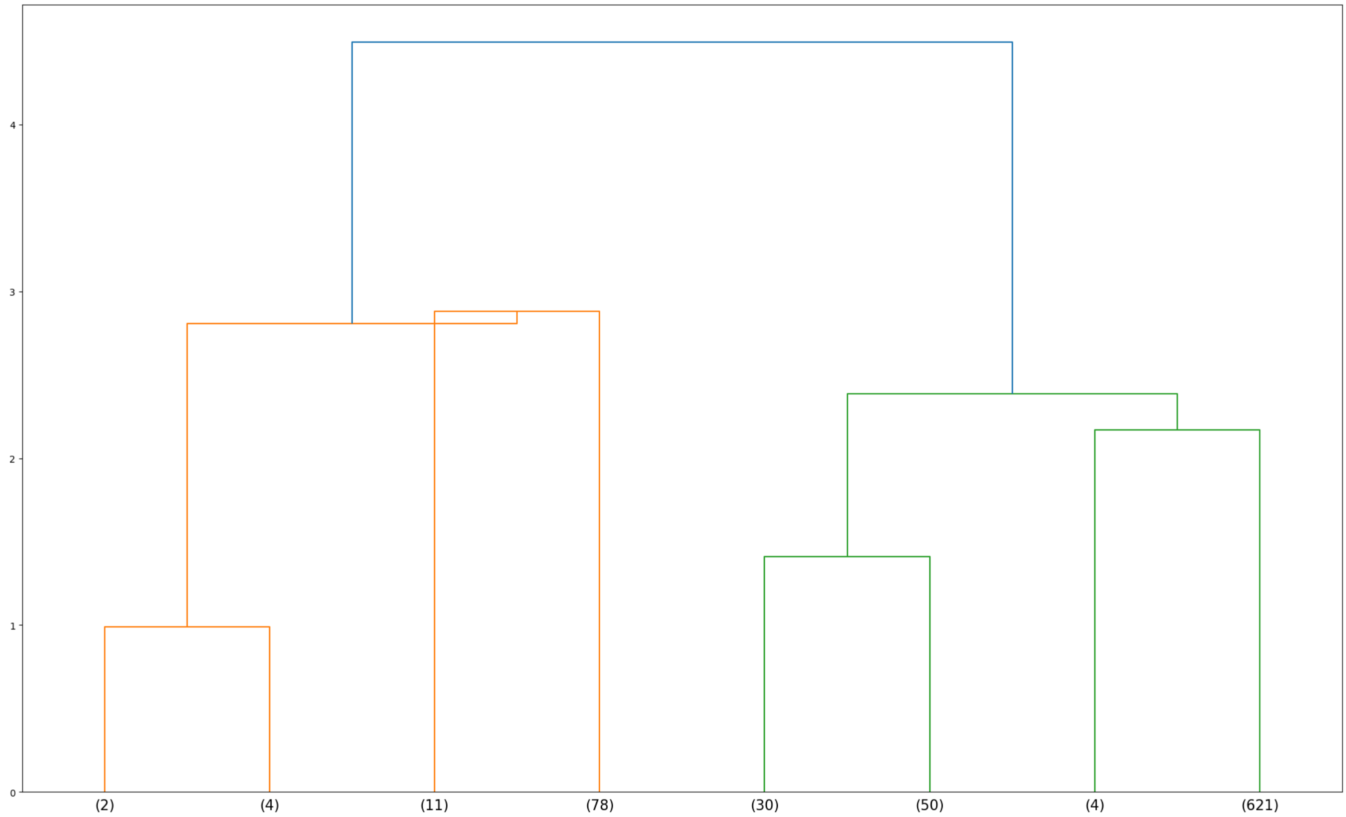
```
dict_linkage['median']
```

```
array([[2.28000000e+02, 5.23000000e+02, 2.00157438e-01, 2.00000000e+00],
       [4.30000000e+02, 1.68000000e+02, 2.31721816e-01, 2.00000000e+00],
       [8.50000000e+01, 4.71000000e+02, 2.46748860e-01, 2.00000000e+00],
       ...,
       [1.59200000e+03, 1.59500000e+03, 2.46004126e+00, 9.80000000e+01],
       [1.59400000e+03, 3.11000000e+02, 2.48159331e+00, 7.02000000e+02],
       [1.59600000e+03, 1.59700000e+03, 3.39140179e+00, 8.00000000e+02]])
```

```
dict_linkage['centroid']
```

```
array([[2.28000000e+02, 5.23000000e+02, 2.00157438e-01, 2.00000000e+00],
       [4.30000000e+02, 1.68000000e+02, 2.31721816e-01, 2.00000000e+00],
       [8.50000000e+01, 4.71000000e+02, 2.46748860e-01, 2.00000000e+00],
       ...,
       [1.40000000e+03, 1.59100000e+03, 2.88174508e+00, 8.90000000e+01],
       [1.59600000e+03, 1.03800000e+03, 2.80561016e+00, 9.50000000e+01],
       [1.59500000e+03, 1.59700000e+03, 4.49353136e+00, 8.00000000e+02]])
```

```
from matplotlib import pyplot as plt
fig = plt.figure(figsize=(25, 15))
dendrogram(
  Z = dict_linkage['centroid'],
  truncate_mode='level', p=2,
  count_sort = True,
  distance_sort = True,
  orientation = 'top',
  leaf_font_size = 15)
plt.show()
```


4
3
2
1
0
(2)
(4)
(11)
(78)
(30)
(50)
(4)
(621)

```
cl = dict_linkage['centroid']
cl1 = dict_linkage['ward']
cl2 = dict_linkage['median']


numclust =4
```

```
fl_centroid = fcluster(cl,numclust,criterion='maxclust')
fl_centroid
```

```
array([1, 1, 1, 1, 1, 1, 1, 1, 4, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3,
       1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 3, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 3, 1, 1, 1, 3, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 3, 1, 1, 1, 1, 3, 1, 1, 1, 1, 1, 3, 3, 1, 1, 1, 3, 1, 3,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 1, 3, 1,
       1, 1, 1, 1, 1, 4, 3, 3, 1, 1, 1, 1, 1, 1, 1, 3, 1, 1, 1, 1, 1, 1,
       1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 3,
       1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 1, 2, 1, 3, 1, 1, 4, 1,
       1, 3, 1, 1, 1, 1, 1, 3, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 1, 1, 1, 1, 1, 1, 3, 1,
       3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 1,
       1, 1, 1, 1, 3, 1, 3, 1, 1, 1, 1, 1, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 3, 1, 1, 1, 1, 3, 1, 1, 3, 1, 3, 1, 1, 1,
       1, 1, 1, 1, 1, 2, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 3, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 4, 1, 1, 1, 1,
       2, 1, 1, 1, 3, 3, 1, 1, 3, 1, 1, 4, 2, 1, 1, 1, 1, 1, 3, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 1, 1, 1, 3, 1, 1, 3, 1, 1,
       1, 3, 1, 1, 1, 3, 1, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 3, 1, 1, 3, 1, 1, 1,
       1, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3,
       1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 3, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 1, 1, 1, 1, 3, 1,
       3, 1, 1, 1, 1, 1, 1, 3, 1, 1, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 3, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 1, 1, 1, 3, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 1, 1, 1,
       1, 1, 3, 1, 1, 1, 1, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 3, 3, 1, 1,
       1, 1, 1, 3, 1, 1, 1, 1], dtype=int32)
```

```
fl_median = fcluster(cl2,numclust,criterion='maxclust')
fl_median
```

```
array([3, 3, 3, 3, 3, 3, 3, 3, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 2,
       3, 2, 3, 3, 3, 3, 3, 3, 3, 3, 1, 3, 1, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 2, 3, 3, 3, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 2, 3, 3, 3, 3, 2, 3, 3, 3, 3, 3, 1, 2, 3, 3, 3, 2, 3, 1,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 1, 3, 3, 3, 3, 3, 3, 3, 3, 2, 3, 2, 3,
       3, 3, 3, 3, 3, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 1, 3, 3, 3, 3, 3, 3,
       3, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 2, 1, 3, 3, 3, 3, 3, 3, 2,
       3, 3, 2, 3, 3, 3, 3, 3, 3, 3, 3, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 1, 3, 1, 3, 2, 3, 3, 2, 3,
       3, 2, 3, 3, 3, 3, 3, 1, 1, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 2, 3, 3, 3, 3, 3, 3, 2, 3,
       2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 2, 3,
       3, 3, 3, 3, 2, 3, 1, 3, 3, 3, 3, 3, 3, 2, 2, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 4, 3, 3, 3, 3, 2, 3, 3, 3, 3, 2, 3, 3, 2, 3, 2, 3, 3, 3,
       3, 3, 3, 3, 3, 1, 1, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 1, 3, 1, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 1, 3, 3, 3, 3, 3, 3, 3, 3, 3, 2, 3, 3, 3, 3,
       1, 3, 3, 3, 2, 2, 3, 3, 2, 3, 3, 2, 1, 3, 3, 3, 3, 3, 2, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 2, 3, 3, 3, 1, 3, 3, 1, 3, 3,
       3, 1, 3, 3, 3, 1, 3, 3, 3, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 2, 3, 3, 3, 3, 3, 3, 3, 2, 3, 3, 1, 3, 3, 3,
       3, 3, 3, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 1, 1, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 1,
       3, 3, 3, 3, 3, 3, 3, 1, 3, 3, 3, 3, 3, 3, 3, 3, 1, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 2, 3, 3, 3, 3, 2, 3,
       2, 3, 3, 3, 3, 3, 3, 2, 3, 3, 3, 3, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 1, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 2, 3, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 2, 3, 3, 3, 2, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 2, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 2, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 1, 3, 3, 3,
       3, 3, 2, 3, 3, 3, 3, 3, 3, 2, 3, 3, 3, 3, 3, 3, 3, 3, 2, 1, 3, 3,
       3, 3, 3, 1, 3, 3, 3, 3], dtype=int32)
```

```
fl_ward = fcluster(cl1,numclust,criterion='maxclust')
fl_ward
```

```
array([3, 3, 3, 3, 2, 3, 4, 2, 2, 1, 3, 2, 3, 3, 3, 4, 3, 3, 3, 3, 3, 1,
       3, 1, 3, 3, 3, 3, 3, 3, 4, 2, 1, 3, 1, 4, 3, 3, 3, 3, 3, 4, 3, 3,
       3, 3, 3, 3, 3, 4, 1, 2, 3, 4, 1, 1, 3, 3, 3, 3, 2, 3, 3, 3, 3, 3,
       3, 2, 4, 1, 3, 4, 2, 3, 1, 3, 3, 3, 3, 3, 1, 1, 3, 3, 3, 1, 3, 1,
       2, 2, 3, 3, 3, 3, 2, 3, 3, 1, 2, 3, 3, 3, 3, 3, 3, 4, 1, 4, 1, 3,
       2, 3, 3, 3, 3, 2, 1, 1, 3, 4, 3, 3, 3, 3, 3, 1, 3, 3, 2, 3, 3, 3,
       3, 1, 3, 2, 4, 4, 3, 2, 3, 3, 3, 2, 2, 3, 3, 3, 3, 3, 4, 3, 3, 3,
       2, 2, 3, 2, 3, 3, 3, 3, 3, 2, 3, 3, 3, 3, 1, 3, 3, 3, 3, 3, 3, 1,
       2, 3, 1, 3, 2, 3, 3, 3, 3, 3, 3, 1, 2, 3, 3, 2, 3, 3, 3, 3, 3, 2,
       2, 1, 3, 3, 3, 2, 3, 3, 2, 3, 3, 3, 2, 1, 3, 1, 4, 1, 3, 3, 2, 3,
       3, 1, 3, 3, 3, 3, 2, 1, 1, 3, 3, 3, 3, 2, 3, 4, 3, 2, 2, 3, 2, 3,
       4, 1, 3, 3, 4, 3, 3, 2, 3, 3, 3, 3, 4, 1, 3, 3, 2, 4, 3, 3, 1, 3,
       1, 3, 3, 2, 3, 3, 3, 3, 3, 2, 3, 4, 4, 2, 4, 3, 3, 3, 3, 3, 1, 4,
       2, 3, 3, 4, 1, 3, 1, 2, 3, 3, 4, 3, 3, 3, 1, 3, 3, 3, 3, 3, 4, 4,
       3, 3, 3, 2, 3, 3, 3, 3, 1, 3, 2, 3, 3, 1, 2, 3, 1, 3, 1, 2, 3, 3,
       3, 2, 3, 3, 3, 1, 1, 3, 4, 3, 3, 3, 3, 3, 3, 2, 2, 3, 3, 2, 2, 3,
       3, 3, 3, 3, 2, 3, 3, 1, 3, 1, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 2, 3, 2, 3, 3, 4, 3, 3, 3, 3, 3, 3, 2, 4, 3, 3, 3, 2,
       3, 3, 3, 3, 2, 3, 3, 1, 4, 3, 3, 3, 2, 3, 3, 3, 4, 2, 3, 3, 3, 2,
       1, 2, 4, 3, 1, 1, 2, 3, 1, 3, 3, 2, 1, 3, 3, 3, 3, 3, 1, 4, 3, 2,
       3, 3, 3, 3, 4, 4, 3, 3, 3, 3, 3, 3, 1, 3, 3, 3, 1, 3, 3, 1, 2, 3,
       3, 1, 2, 3, 2, 1, 3, 3, 3, 1, 3, 4, 3, 3, 3, 2, 3, 3, 4, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 1, 3, 4, 3, 3, 3, 3, 3, 1, 3, 3, 1, 3, 3, 3,
       2, 3, 3, 1, 3, 3, 3, 3, 3, 2, 2, 3, 2, 3, 3, 3, 3, 1, 1, 3, 2, 3,
       2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 1, 2, 2, 3, 3, 2, 3, 3, 3, 2, 3, 1,
       2, 3, 3, 3, 2, 3, 3, 1, 4, 4, 4, 4, 2, 3, 3, 3, 1, 2, 3, 3, 4, 3,
       3, 3, 2, 3, 3, 3, 4, 4, 2, 3, 3, 3, 3, 3, 3, 1, 3, 3, 3, 3, 1, 3,
       1, 3, 3, 2, 2, 3, 3, 1, 3, 3, 3, 4, 1, 3, 3, 3, 3, 2, 2, 3, 3, 3,
       3, 2, 3, 2, 3, 3, 4, 3, 3, 3, 2, 2, 2, 2, 3, 2, 3, 3, 1, 3, 3, 4,
       3, 3, 3, 2, 3, 3, 3, 4, 1, 3, 1, 3, 3, 3, 3, 3, 3, 3, 2, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 2, 2, 2, 2, 2, 3, 3, 2, 3, 3, 3, 3, 3, 2, 2, 4,
       3, 3, 3, 4, 3, 3, 3, 2, 3, 2, 3, 3, 2, 3, 1, 4, 3, 2, 1, 3, 3, 3,
       3, 3, 2, 3, 2, 3, 3, 1, 3, 3, 3, 2, 3, 2, 3, 3, 3, 3, 3, 3, 1, 4,
       4, 3, 4, 3, 3, 3, 3, 3, 3, 2, 3, 3, 3, 3, 1, 3, 3, 2, 2, 3, 2, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 2, 2, 4, 3, 3, 2, 3, 3, 3, 2, 1, 4, 3, 3,
       3, 3, 1, 3, 3, 3, 2, 3, 4, 1, 3, 3, 3, 3, 3, 3, 3, 3, 1, 1, 3, 2,
       3, 3, 3, 1, 3, 3, 3, 2], dtype=int32)
```

```
def mean(deneme, clusters_list):
    cluster_means = []
    for cluster_index in clusters_list:
        cluster_data = deneme.iloc[cluster_index]
        mean = np.mean(cluster_data, axis=0)
        cluster_means.append(mean)
    return cluster_means
```

```
means_fl_centroid = mean(df_combined, fl_centroid)
means_fl_median = mean(df_combined, fl_median)
means_fl_ward = mean(df_combined, fl_ward)
```

```
means_fl_ward
```

```
     2.12644020227503,
     2.1557859862667064,
     2.12644020227503,
     2.12644020227503,
     2.12644020227503,
     2.12644020227503,
     2.12644020227503,
     2.12644020227503,
     2.344925936606696,
     2.12644020227503,
     2.12644020227503,
     2.12644020227503,
     2.12644020227503,
     2.010906757436619,
     2.12644020227503,
     2.12644020227503,
     2.344925936606696,
     2.344925936606696,
     2.12644020227503,
     2.344925936606696,
     2.12644020227503,
     2.12644020227503,
     2.12644020227503,
     2.12644020227503,
     2.12644020227503,
     2.12644020227503,
     2.12644020227503,
     2.12644020227503,
     2.12644020227503,
     2.344925936606696,
     2.344925936606696,
     2.1557859862667064,
     2.12644020227503,
     2.12644020227503,
     2.344925936606696,
     2.12644020227503,
     2.12644020227503,
     2.12644020227503,
     2.344925936606696,
     2.010906757436619,
     2.1557859862667064,
     2.12644020227503,
     2.12644020227503,
     2.12644020227503,
     2.12644020227503,
     2.010906757436619,
```

```
2.12644020227503,
2.12644020227503,
2.12644020227503,
2.344925936606696,
2.12644020227503,
2.1557859862667064,
2.010906757436619,
2.12644020227503,
2.12644020227503,
2.12644020227503,
2.12644020227503,
2.12644020227503,
2.12644020227503,
2.12644020227503
```

```
means_fl_median
```

```
2.12644020227503,
2.12644020227503,
2.12644020227503,
2.12644020227503,
2.12644020227503,
2.12644020227503,
2.12644020227503,
2.12644020227503,
2.12644020227503,
2.12644020227503,
2.12644020227503,
2.12644020227503,
2.12644020227503,
2.344925936606696,
2.12644020227503,
2.12644020227503,
2.12644020227503,
2.12644020227503,
2.12644020227503,
2.12644020227503,
2.12644020227503,
2.12644020227503,
2.12644020227503,
2.12644020227503,
2.12644020227503,
2.12644020227503,
2.12644020227503,
2.12644020227503,
2.12644020227503,
2.12644020227503,
2.12644020227503,
2.12644020227503,
2.12644020227503,
2.12644020227503,
2.12644020227503,
2.12644020227503,
```

```
      2.12644020227503,
      2.12644020227503,
      2.12644020227503,
      2.010906757436619,
      2.12644020227503,
      2.12644020227503,
      2.12644020227503,
      2.12644020227503,
      2.12644020227503,
      2.344925936606696,
      2.12644020227503,
      2.12644020227503,
      2.12644020227503,
      2.12644020227503,
      2.12644020227503,
      2.12644020227503,
      2.344925936606696,
      2.12644020227503,
      2.12644020227503,
      2.12644020227503,
      2.12644020227503,
      2.12644020227503,
      2.12644020227503,
      2.12644020227503
```

```
means_fl_centroid
```

```
      2.010906757436619,
      2.010906757436619,
      2.010906757436619,
      2.010906757436619,
      2.010906757436619,
      2.010906757436619,
      2.010906757436619,
      2.010906757436619,
      2.010906757436619,
      2.010906757436619,
      2.010906757436619,
      2.010906757436619,
      2.010906757436619,
      2.12644020227503,
      2.010906757436619,
      2.010906757436619,
      2.010906757436619,
      2.010906757436619,
      2.010906757436619,
      2.010906757436619,
      2.010906757436619,
      2.010906757436619,
      2.010906757436619,
      2.010906757436619,
      2.010906757436619,
      2.010906757436619,
      2.010906757436619,
```

```
      2.010906757436619,
      2.010906757436619,
      2.010906757436619,
      2.010906757436619,
      2.010906757436619,
      2.010906757436619,
      2.010906757436619,
      2.010906757436619,
      2.010906757436619,
      2.010906757436619,
      2.010906757436619,
      2.010906757436619,
      2.12644020227503,
      2.010906757436619,
      2.010906757436619,
      2.010906757436619,
      2.010906757436619,
      2.010906757436619,
      2.12644020227503,
      2.010906757436619,
      2.010906757436619,
      2.010906757436619,
      2.010906757436619,
      2.010906757436619,
      2.010906757436619,
      2.12644020227503,
      2.010906757436619,
      2.010906757436619,
      2.010906757436619,
      2.010906757436619,
      2.010906757436619,
      2.010906757436619,
      2.010906757436619
```

```
def covariance(deneme, clusters):
    covariances = []
    for cluster_index in clusters:
        cluster_data = deneme.iloc[cluster_index]
        covariance = np.cov(cluster_data, rowvar=False)
        covariances.append(covariance)
    return covariances


covariance_centroid = covariance(df_combined, fl_centroid)
covariance_centroid
```

```
      array(5.45332554),
      array(5.45332554),
      array(5.45332554),
      array(5.45332554),
      array(5.45332554),
      array(5.45332554),
      array(5.45332554),
```

```
        array(5.45332554),
        array(5.45332554),
        array(5.45332554),
        array(5.45332554),
        array(8.83150676),
        array(5.45332554),
        array(5.45332554),
        array(5.45332554),
        array(5.45332554),
        array(5.45332554),
        array(5.45332554),
        array(5.45332554),
        array(5.45332554),
        array(5.45332554),
        array(5.45332554),
        array(5.45332554),
        array(5.45332554),
        array(5.45332554),
        array(5.45332554),
        array(5.45332554),
        array(5.45332554),
        array(5.45332554),
        array(5.45332554),
        array(5.45332554),
        array(5.45332554),
        array(5.45332554),
        array(5.45332554),
        array(5.45332554),
        array(5.45332554),
        array(5.45332554),
        array(8.83150676),
        array(5.45332554),
        array(5.45332554),
        array(5.45332554),
        array(5.45332554),
        array(5.45332554),
        array(8.83150676),
        array(5.45332554),
        array(5.45332554),
        array(5.45332554),
        array(5.45332554),
        array(5.45332554),
        array(5.45332554),
        array(8.83150676),
        array(5.45332554),
        array(5.45332554),
        array(5.45332554),
        array(5.45332554),
        array(5.45332554),
        array(5.45332554),
        array(5.45332554),
        array(5.45332554),
        array(8.83150676),

covariance_median = covariance(df_combined, f1_median)
```

```
covariance_median = covariance(df_combined, ft_median)
covariance_median
```

```
     array(8.83150676),
     array(8.83150676),
     array(8.83150676),
     array(8.83150676),
     array(8.83150676),
     array(8.83150676),
     array(8.83150676),
     array(8.83150676),
     array(8.83150676),
     array(8.83150676),
     array(8.83150676),
     array(8.83150676),
     array(6.14256784),
     array(8.83150676),
     array(8.83150676),
     array(8.83150676),
     array(8.83150676),
     array(8.83150676),
     array(8.83150676),
     array(8.83150676),
     array(8.83150676),
     array(8.83150676),
     array(8.83150676),
     array(8.83150676),
     array(8.83150676),
     array(8.83150676),
     array(8.83150676),
     array(8.83150676),
     array(8.83150676),
     array(8.83150676),
     array(8.83150676),
     array(8.83150676),
     array(8.83150676),
     array(8.83150676),
     array(8.83150676),
     array(8.83150676),
     array(8.83150676),
     array(8.83150676),
     array(5.45332554),
     array(8.83150676),
     array(8.83150676),
     array(8.83150676),
     array(8.83150676),
     array(8.83150676),
     array(6.14256784),
     array(8.83150676),
     array(8.83150676),
     array(8.83150676),
     array(8.83150676),
     array(8.83150676),
     array(8.83150676),
```

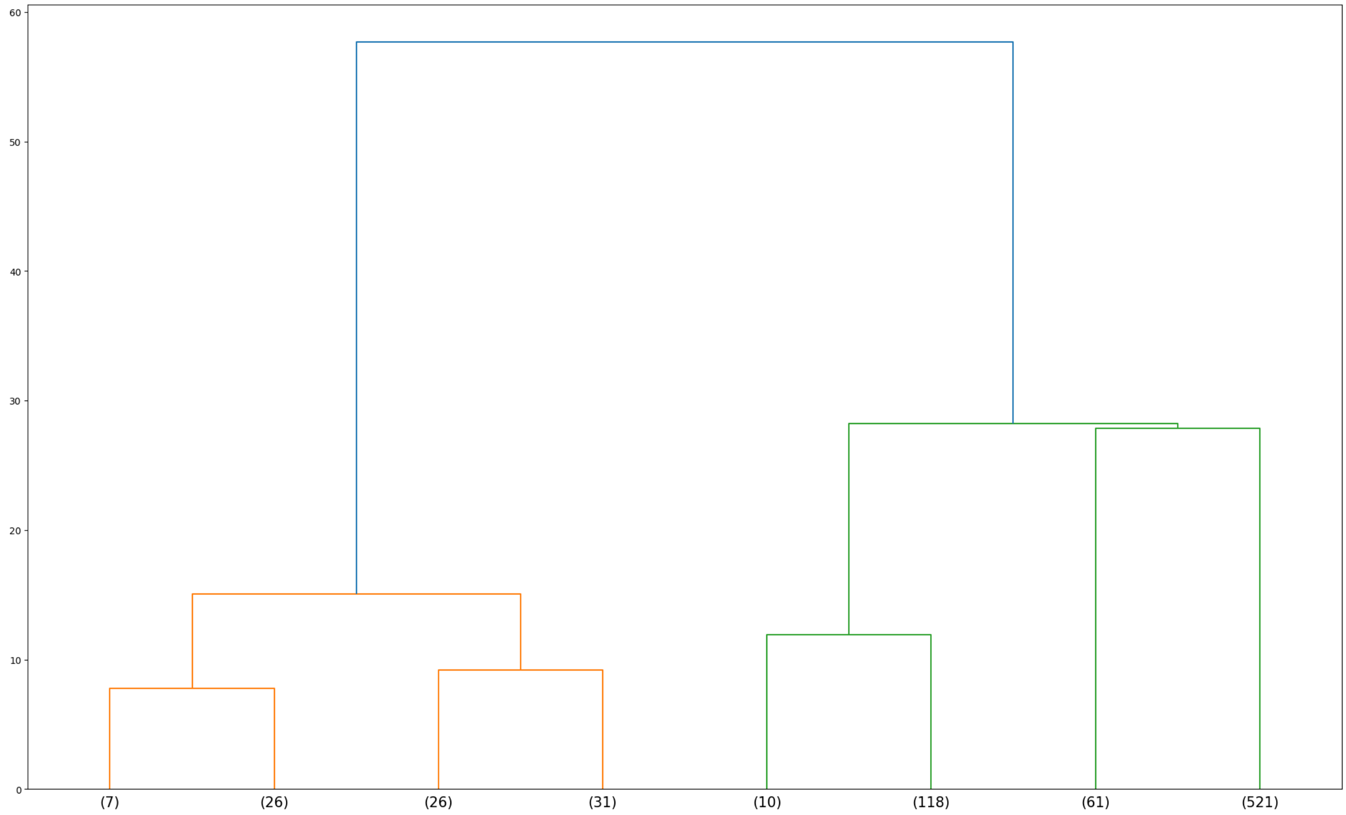
```
       array(6.14256784),
       array(8.83150676),
       array(8.83150676),
       array(8.83150676),
       array(8.83150676),
       array(8.83150676),
       array(8.83150676),
       array(8.83150676),
       array(8.83150676),
```

```
covariance_ward = covariance(df_combined, fl_ward)
covariance_ward
```

```
       array(5.81282278),
       array(8.83150676),
       array(8.83150676),
       array(8.83150676),
       array(8.83150676),
       array(8.83150676),
       array(8.83150676),
       array(6.14256784),
       array(8.83150676),
       array(8.83150676),
       array(8.83150676),
       array(8.83150676),
       array(5.45332554),
       array(8.83150676),
       array(8.83150676),
       array(6.14256784),
       array(6.14256784),
       array(8.83150676),
       array(6.14256784),
       array(8.83150676),
       array(8.83150676),
       array(8.83150676),
       array(8.83150676),
       array(8.83150676),
       array(8.83150676),
       array(8.83150676),
       array(8.83150676),
       array(8.83150676),
       array(6.14256784),
       array(6.14256784),
       array(5.81282278),
       array(8.83150676),
       array(8.83150676),
       array(6.14256784),
       array(8.83150676),
       array(8.83150676),
       array(8.83150676),
       array(6.14256784),
       array(5.45332554),
       array(5.81282278),
       array(8.83150676),
```

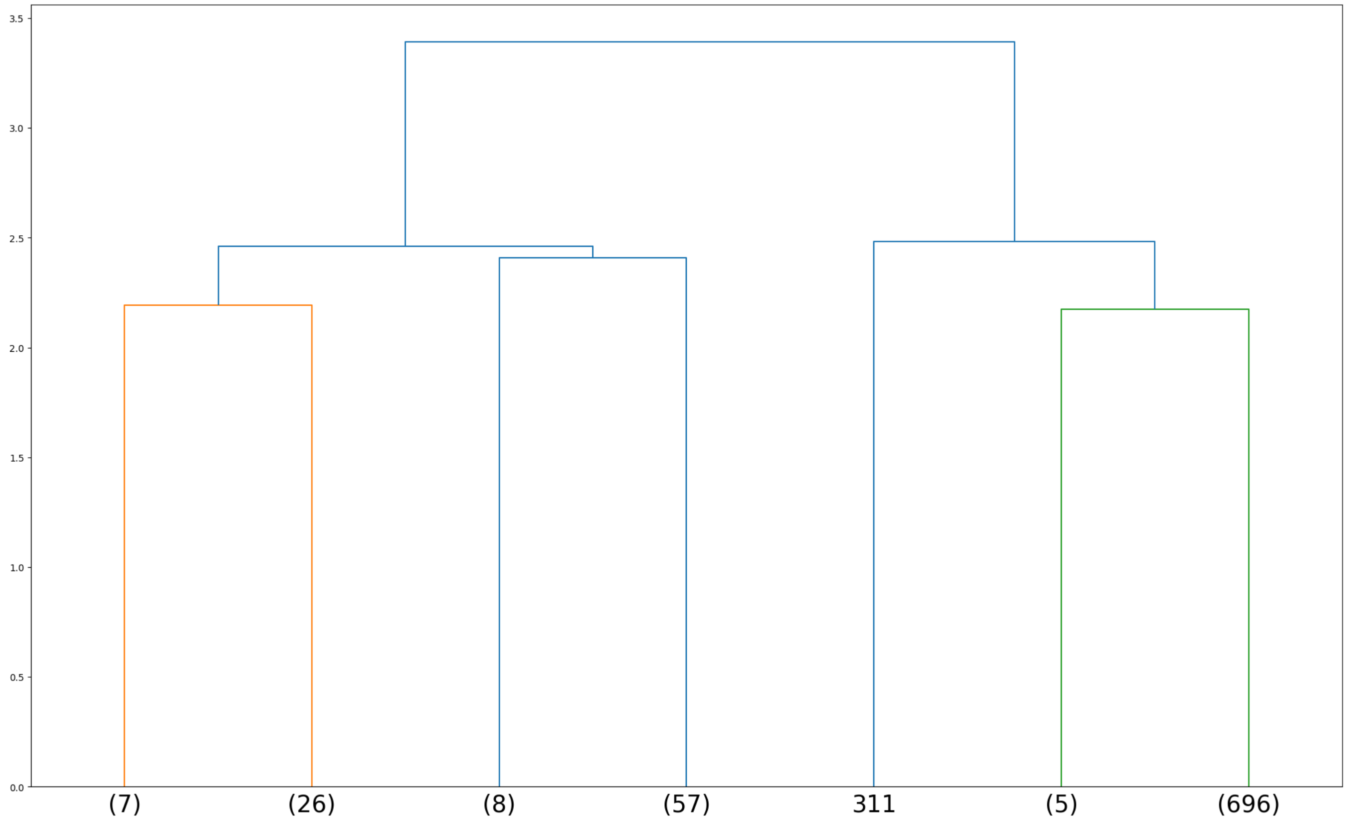
```
       array(8.83150676),
       array(8.83150676),
       array(8.83150676),
       array(5.45332554),
       array(8.83150676),
       array(8.83150676),
       array(8.83150676),
       array(6.14256784),
       array(8.83150676),
       array(5.81282278),
       array(5.45332554),
       array(8.83150676),
       array(8.83150676),
       array(8.83150676),
       array(8.83150676),
       array(8.83150676),
       array(8.83150676),
       array(8.83150676),
       array(8.83150676)
```

```
from matplotlib import pyplot as plt
fig = plt.figure(figsize=(25, 15))
dendrogram(
  Z = dict_linkage['ward'],
  count_sort = True,
  truncate_mode='level', p=2,
  distance_sort = True,
  orientation = 'top',
  leaf_font_size = 15)
plt.show()
```




```
from matplotlib import pyplot as plt
fig = plt.figure(figsize=(25, 15))
dendrogram(
  Z = dict_linkage['median'],
```

```
    truncate_mode='level', p=2,
    count_sort = True,
    distance_sort = True,
    orientation = 'top',
    leaf_font_size = 25)
plt.show()
```



Double-click (or enter) to edit

In this homework i have seen the difference between random forest clustering and k means clustering. It was quite interesting for me that how things changes when we add noise data. Morever it can be seen in my project that choosing different algoritms could influence the clustering results. The properties of the data and the objectives of the analysis should guide the choice of a suitable distance metric and clustering algorithm. Random forest works for this type of problem when we increase the features it tended to give more compact solution.