import pandas as pd

import re

from sklearn.preprocessing import LabelEncoder

from sklearn.manifold import TSNE

import umap

import matplotlib.pyplot as plt

data = []

with open('C:/Users/TO THANH/OneDrive/Máy tính/hôm nay/university.data', 'r') as file:

    university = {}

    for line in file:

        if '(def-instance' in line:

            university = {'name': line.split()[1]}

        elif '))' in line:

            data.append(university)

            university = {}

        else:

            matches = re.findall('\((.\*?)\)', line)

            if matches:

                split\_data = matches[0].split(None, 1)

                if len(split\_data) == 2:

                    attribute, value = split\_data

                    university[attribute] = value

df = pd.DataFrame(data)

le = LabelEncoder()

for column in df.columns:

    df[column] = le.fit\_transform(df[column].astype(str))

tsne = TSNE(n\_components=2, random\_state=0)

data\_2d\_tsne = tsne.fit\_transform(df)

reducer = umap.UMAP()

data\_2d\_umap = reducer.fit\_transform(df)

fig, ax = plt.subplots(1, 2, figsize=(15, 7))

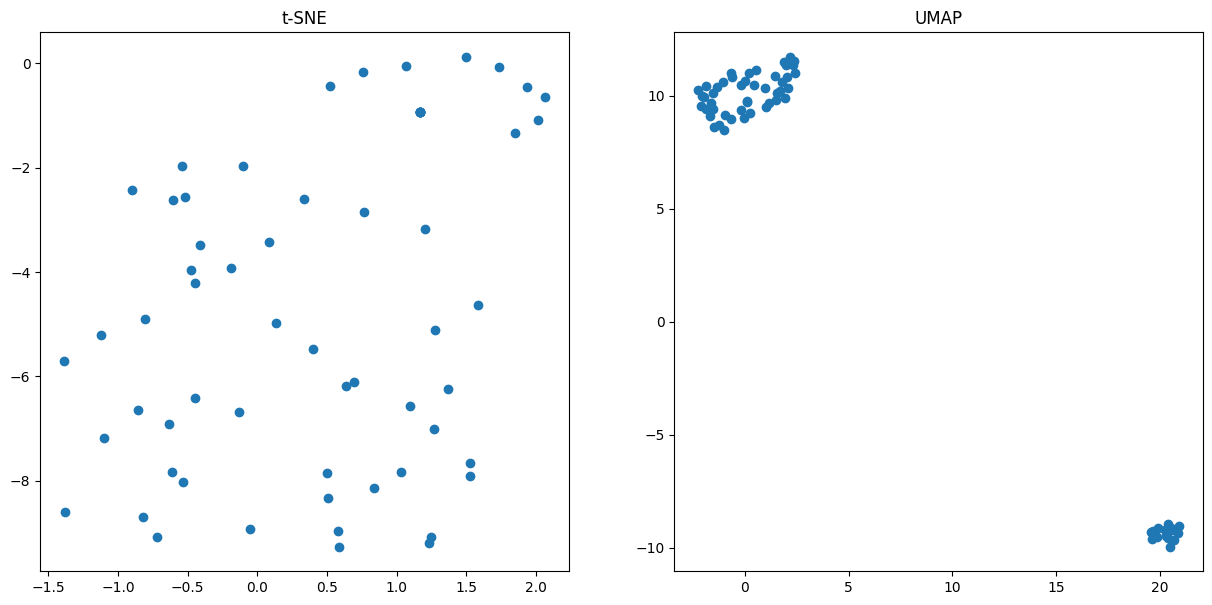
ax[0].scatter(data\_2d\_tsne[:, 0], data\_2d\_tsne[:, 1])

ax[0].set\_title('t-SNE')

ax[1].scatter(data\_2d\_umap[:, 0], data\_2d\_umap[:, 1])

ax[1].set\_title('UMAP')

plt.show()



from sklearn.preprocessing import MinMaxScaler, StandardScaler, RobustScaler

min\_max\_scaler = MinMaxScaler()

df\_min\_max = min\_max\_scaler.fit\_transform(df)

standard\_scaler = StandardScaler()

df\_standard = standard\_scaler.fit\_transform(df)

robust\_scaler = RobustScaler()

df\_robust = robust\_scaler.fit\_transform(df)

scalers = [('MinMax', df\_min\_max), ('Standard', df\_standard), ('Robust', df\_robust)]

for name, scaled\_df in scalers:

    tsne = TSNE(n\_components=2, random\_state=0)

    data\_2d\_tsne = tsne.fit\_transform(scaled\_df)

    reducer = umap.UMAP()

    data\_2d\_umap = reducer.fit\_transform(scaled\_df)

    fig, ax = plt.subplots(1, 2, figsize=(15, 7))

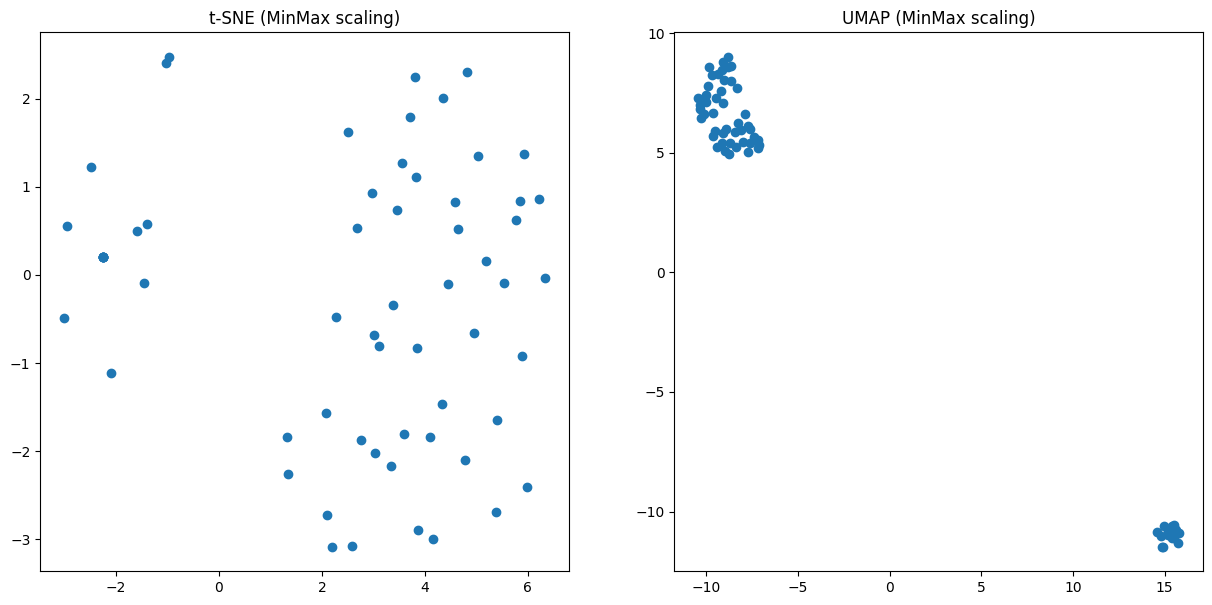
    ax[0].scatter(data\_2d\_tsne[:, 0], data\_2d\_tsne[:, 1])

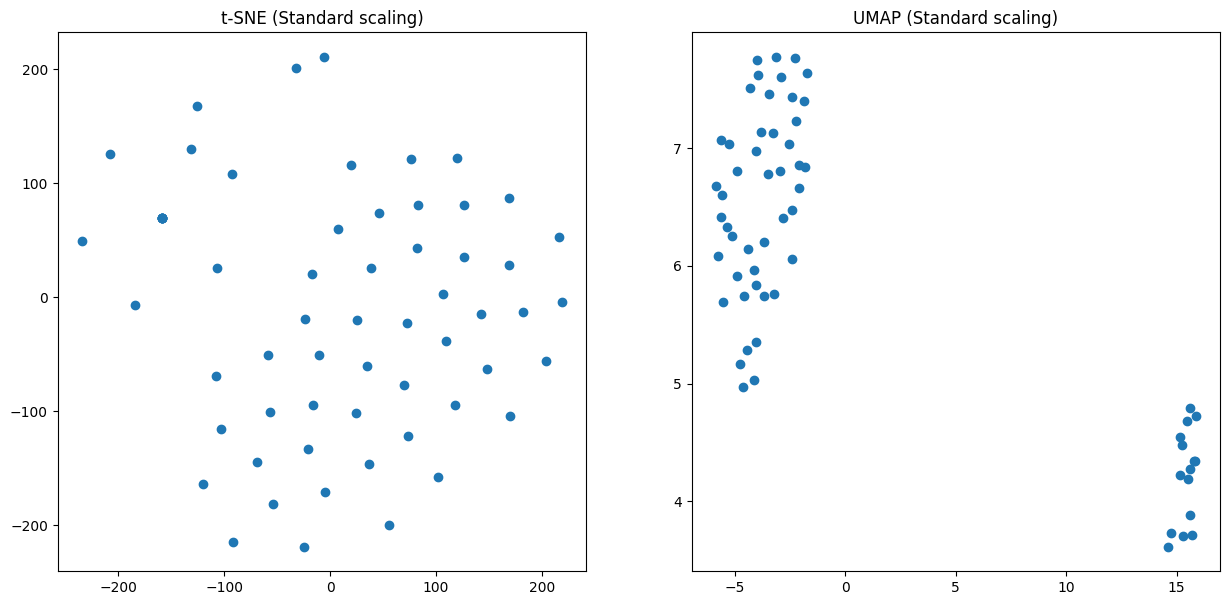
    ax[0].set\_title(f't-SNE ({name} scaling)')

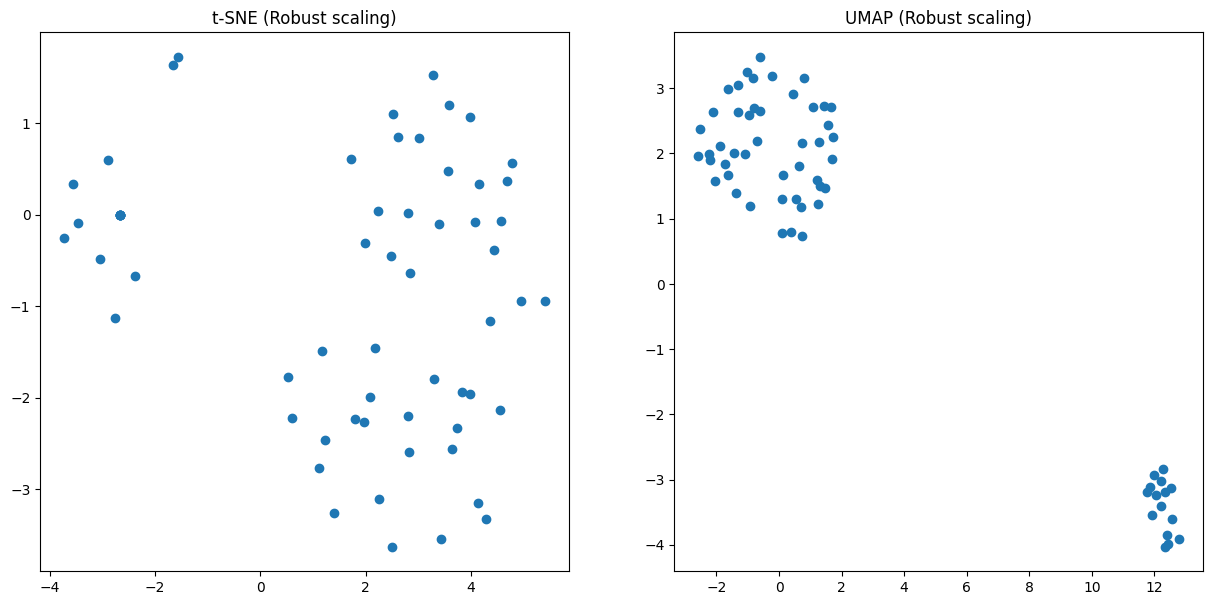
    ax[1].scatter(data\_2d\_umap[:, 0], data\_2d\_umap[:, 1])

    ax[1].set\_title(f'UMAP ({name} scaling)')

    plt.show()







На рисунке мы видим результаты применения трех методов масштабирования: MinMax Scaling, Standard Scaling и Robust Scaling, а затем использования двух методов уменьшения размерности данных, t-SNE и UMAP, для визуализации данных.

1. Масштабирование MinMax. На этом рисунке данные после масштабирования MinMax показывают четкое разделение между группами данных в обоих методах уменьшения размерности. Это вместо MinMax Scaling может помочь сохранить угловую структуру данных.

2. Стандартное масштабирование. Данные в масштабе Stardard также показывают разделение между группами данных, но не так четко, как при масштабировании MinMax. Это может быть связано с тем, что стандартное масштабирование преобразует данные к нормальному распределению, возможно, охлаждая некоторую информацию об исходной структуре данных.

3. Робастное масштабирование. Данные после Робастного масштабирования показывают, что разделение между группами данных не такое четкое, как в двух вышеупомянутых методах. Это может быть связано с тем, что надежное масштабирование в основном направлено на минимизацию влияния выбросов, которые могут не отражать исходную структуру данных.

Судя по визуализации, MinMax Scaling лучше всего сохраняет исходную структуру данных при одновременном уменьшении размерности данных. Однако обратите внимание, что выбор подходящего метода масштабирования зависит от конкретного набора данных и цели задачи.

import seaborn as sns

min\_max\_scaler = MinMaxScaler()

df\_min\_max = min\_max\_scaler.fit\_transform(df)

df\_min\_max = pd.DataFrame(df\_min\_max, columns = df.columns)

standard\_scaler = StandardScaler()

df\_standard = standard\_scaler.fit\_transform(df)

df\_standard = pd.DataFrame(df\_standard, columns = df.columns)

robust\_scaler = RobustScaler()

df\_robust = robust\_scaler.fit\_transform(df)

df\_robust = pd.DataFrame(df\_robust, columns = df.columns)

fig, (ax1, ax2, ax3, ax4) = plt.subplots(ncols = 4, figsize =(20, 5))

ax1.set\_title('Before Scaling')

for column in df.columns:

    sns.kdeplot(df[column], ax = ax1)

ax2.set\_title('After Robust Scaling')

for column in df\_robust.columns:

    sns.kdeplot(df\_robust[column], ax = ax2)

ax3.set\_title('After Standard Scaling')

for column in df\_standard.columns:

    sns.kdeplot(df\_standard[column], ax = ax3)

ax4.set\_title('After Min-Max Scaling')

for column in df\_min\_max.columns:

    sns.kdeplot(df\_min\_max[column], ax = ax4)

plt.show()

На итоговой диаграмме KDE (Kernel Density Estimate) мы видим, что после применения методов масштабирования распределение данных изменилось. MinMax Scaling и Standard Scaling делают данные более согласованными, а Robust Scaling сохраняет исходную форму распределения, но уменьшает влияние выбросов. 