Assignment 2A

Wilhelm Öberg
Philip Rettig

January 2024

Principal Component Analysis

Question 2A.1: *Explain why this data-centering step is required while performing PCA. What could be an undesirable effect if we perform PCA on non-centered data?*

Solution:
In PCA we compare and rank each datapoints variance. Striving to keep the features with high variance to perserve as much information as possible in the reduced version of the datapoint. Variance is the expected value of a variabels squared deviation from the mean. A high variance along a particular direction indicates that this direction captures a significant amount of the datas variability. If datapoints are not normalized to mean zero, it is useless to compare datapoints' variances to eachother. An undesireble effect if we perform PCA on non-centered data could be the loss of important information and an incorrect datapoint after reconstruction of the original dimensions.

Question 2A.2: *Does the previous argument imply that a single SVD operation is sufficient to perform PCA both on the rows and the columns of a data matrix?*
Justify your answer.

Solution:
Yes. Performing PCA on the rows gives us $\mathbf{A}^T\mathbf{A}$ and on columns gives us $\mathbf{A}\mathbf{A}^T$. Performing SVD on $\mathbf{A}$ decomposes $\mathbf{A}$ into $\mathbf{U}\Sigma\mathbf{U}^T$, i.e $\mathbf{A}^T\mathbf{A} = \mathbf{V}\Sigma^2\mathbf{V}^T$ and $\mathbf{A}\mathbf{A}^T = \mathbf{U}\Sigma^2\mathbf{U}^T$. Since the diagonal matrix is symmetrical, and $\mathbf{U}$ and $\mathbf{V}$ provides the eigenvectors (principle direction) of the columns and rows seperately, it's sufficient to perform SVD once.

Question 2A.3: *Show that the variance of the dataset Y, as defined in Equation (1), can be expressed as a function of the singular values of Y, and in particular*

$$Var(\mathcal{Y}) = \sum_{i=1}^{d} \sigma_i^2$$

Consider a dataset $\mathcal{Y} = \{\mathbf{y}_1, \ldots, \mathbf{y}_n\}$ with n points of dimension d. The *variance* of the dataset $\mathcal{Y}$ is

$$Var(\mathcal{Y}) = \sum_{\mathbf{y} \in \mathcal{Y}} ||\mathbf{y} - \bar{\mathbf{y}}||_2^2$$

where $\bar{\mathbf{y}} = \frac{1}{n} \sum_{\mathbf{y} \in \mathcal{Y}} \mathbf{y}$ is the mean point. Further, assume that the data are zero-centered, so $\bar{\mathbf{y}} = 0$. The dataset $\mathcal{Y}$ can be represented as a $d \times n$ matrix $\mathbf{Y}$, and consider the SVD of $\mathbf{Y} = \mathbf{U}\Sigma\mathbf{V}^T$

Show that:

$$Var(\mathcal{Y}) = \sum_{i=1}^{d} \sigma_i^2$$

Initially the variance of the dataset is defined as:

$$Var(\mathcal{Y}) = \sum_{\mathbf{y} \in \mathcal{Y}} ||\mathbf{y} - \bar{\mathbf{y}}||_2^2 = \sum_{\mathbf{y} \in \mathcal{Y}} ||\mathbf{y}||_2^2 = \mathbf{Y}^T\mathbf{Y} = (\mathbf{U}\Sigma\mathbf{V}^T)^T(\mathbf{U}\Sigma\mathbf{V}^T)$$

Since $\Sigma = \Sigma^T$, $\mathbf{V}^T = \mathbf{U}$ and $\mathbf{V}^T\mathbf{V} = \mathbf{I}$

$$(\mathbf{U}\Sigma\mathbf{V}^T)^T(\mathbf{U}\Sigma\mathbf{V}^T) = \mathbf{V}\Sigma^2\mathbf{V}^T$$

Since $\mathbf{V}$ and $\mathbf{V}^T$ are orthogonal, the expression can be simplified to the following

$$\mathbf{V}\Sigma^2\mathbf{V}^T = \Sigma^2 \cdot \mathbf{V}\mathbf{V}^T = \Sigma^2$$

Finally we can rewrite Σ^2 from matrix form to the dot-product form,

$$\Sigma^2 = \sum_{i=1}^{d} \sigma_i^2$$

giving us the final answer that the sum of the singular values squared represent the total variance of the dataset $\mathcal{Y}$ given the all dimensions d.

Question 2A.4: *Show that the variance of the projected data X is given by*

$$Var(\mathcal{X}) = \sum_{i=1}^{k} \sigma_i^2$$

Solution:

$$Var(\mathcal{X}) = \sum_{x \in \mathcal{X}} ||x||_2^2 = \mathbf{X}^T\mathbf{X}$$

Since $\mathbf{X} = \mathbf{W}^T\mathbf{Y}$,

$$\mathbf{X}^T\mathbf{X} = (\mathbf{W}^T\mathbf{Y})^T(\mathbf{W}^T\mathbf{Y})$$

Let's focus on $\mathbf{W}^T\mathbf{Y}$ in combination with the SVD of $\mathbf{Y} = \mathbf{U}\Sigma\mathbf{V}^T$ which gives,

$$\mathbf{X} = \mathbf{W}^T\mathbf{U}\Sigma\mathbf{V}^T$$

Since $\mathbf{W}$ is composed of the first k columns of $\mathbf{V}$, the product $\mathbf{W}^T\mathbf{U}$ will be an identity matrix for the corresponding first k rows and columns, and zero elsewhere. Therefore, the multiplication simplifies to:

$$\mathbf{X} = \Sigma_k\mathbf{V}^T$$

Going back to $Var(\mathcal{X})$,

$$\mathbf{X}^T\mathbf{X} = (\Sigma_k\mathbf{V}^T)^T(\Sigma_k\mathbf{V}^T)$$

Since $\Sigma_k = \Sigma_k^T$ due to orthogonality and $\mathbf{V}\mathbf{V}^T = \mathbf{I}$ the expression simplifies to:

$$(\Sigma_k\mathbf{V}^T)^T(\Sigma_k\mathbf{V}^T) = \Sigma_k^2 = \sum_{i=1}^{k} \sigma_i^2$$

The variance of the projected data $\mathbf{X}$ is the sum of the squares of the singular values. This is because the variance captured by each principal component is given by the corresponding singular value squared, and $\mathbf{X}$ contains only the first k components.

Question 2A.5: *Show that the variance of the residual data Z is given by*

$$Var(\mathcal{Z}) = \sum_{i=k+1}^{d} \sigma_i^2$$

Solution:
The variance of the residual data, $\mathcal{Z} = \{z_1, \ldots, z_n\}$ where $z_i = y_i - \mathbf{W}\mathbf{W}^T y_i$, is given by the reconstruction error, commonly denoted as

$$\begin{aligned}
&\text{Var(Z)} = \text{E}_W = \mathbb{E}_y \left[||y - \mathbf{W}\mathbf{W}^T y||_2^2 \right] \\
&= \mathbb{E}_y \left[(y - \mathbf{W}\mathbf{W}^T y)^T (y - \mathbf{W}\mathbf{W}^T y) \right] \\
&= \mathbb{E}_y \left[y^T y - 2y^T \mathbf{W}\mathbf{W}^T y + y^T \mathbf{W}\mathbf{W}^T \mathbf{W}\mathbf{W}^T y \right] \\
&= \mathbb{E}_y \left[y^T y - y^T \mathbf{W}\mathbf{W}^T y \right] \\
&= \mathbb{E}_y \left[y^T y \right] - \mathbb{E}_y \left[y^T \mathbf{W}\mathbf{W}^T y \right]
\end{aligned}$$

We can subsitute the terms in the expression above with answers from questions 2A.3 and 2A.4, where $\mathbb{E}_y \left[y^T y \right] = \sum_{i=1}^{d} \sigma_i^2$ and $\mathbb{E}_y \left[y^T \mathbf{W}\mathbf{W}^T y \right] = \sum_{i=1}^{k} \sigma_i^2$, giving the following expression:

$$E_W = \mathbb{E}_y \left[y^T y \right] - \mathbb{E}_y \left[y^T \mathbf{W}\mathbf{W}^T y \right] = \sum_{i=1}^{d} \sigma_i^2 - \sum_{i=1}^{k} \sigma_i^2 = \sum_{i=k+1}^{d} \sigma_i^2$$

The results indicate the total dataset variance minus the variance explained by the PCA describes the reconstruction error.

2A/2 PCA vs. Johnson-Lindenstrauss random projections

Question 2A.6: *Provide a qualitative comparison (short discussion) between the two methods, PCA vs. Johnson-Lindenstrauss random projections, in terms of (i) projection error; (ii) computational efficiency; and (iii) target usecases.*

Solution:
While both JL and PCA aim to reduce dimensionality of the data, they differ in how the achieve that goal. The differences in projection error, computitional efficiency and target usecases follow:

Projection error:
Both Principal Component Analysis (PCA) and the Johnson-Lindenstrauss (JL) lemma are based on linear mappings. PCA aims to preserve distances on expectation, but there can be instances where the distance preservation property does not hold, especially for outliers. The two methods differ primarily in the objective function they optimize, i.e., the type of reconstruction error they aim to minimize.

PCA minimizes the reconstruction error of the projection matrix $\mathbf{W}$ by:

$$E_{\mathbf{W}} = \mathbb{E}[\|\mathbf{y} - dec(cod(\mathbf{y}))\|^2] = \mathbb{E}[\|\mathbf{y} - \mathbf{W}\mathbf{W}^T\mathbf{y}\|^2] \tag{1}$$

Which is equivalent to maximizing:

$$\mathbb{E}[\mathbf{y}^T\mathbf{W}\mathbf{W}^T\mathbf{y}] = \frac{1}{n}tr(\mathbf{Y}^T\mathbf{W}\mathbf{W}^T\mathbf{Y}) \tag{2}$$

Minimizing this reconstruction error guarantees that distances are preserved on expectation. However, for outliers, the error can grow quickly. The JL lemma allows us to preserve distances for outliers as well. In the JL approach, we construct a mapping that approximately preserves the pairwise distances between all points, with high probability. I.e JL approximates preservation of the distance which decreases interpretability and accuracy.

Computational efficiency:
The computational cost of PCA can be high for large datasets since it involves the computation of the covariance matrix and its eigendecomposition. This process requires $O(d^2n + d^3)$ operations where d is the dimensionality of the data and n is the number of data points.

In contrast, JL random projections do not require eigendecomposition or the computation of covariances. As a result, they are more computationally efficient, especially for large d. The JL projections also have the advantage that the dimensionality k of the target space grows only logarithmically with

the number of points n, i.e., $k = O(\log(n))$. Furthermore, random projections scale the lengths of vectors by $1/\sqrt{d}$, which is beneficial for datasets with high dimensionality.

Target usecases:
PCA is somewhat more accurate than JL, but lacks in computational efficiecy. JL is faster than PCA but do not render a result as accurate as PCA. JL random projections are useful when dealing with very high-dimensional data, where PCA becomes computationally infesasible. PCA is ideal for e.g noise reduction and feature extraction. So we use PCA when weäre dealing with low-dimensional data and value accuracy and JL on high-dimensional data.

Programming task — MDS

Question 2A.7: (Data acquisition and processing)
In the website HTTPS://CADMUS.EUI.EU/HANDLE/1814/74918 *you can find a dataset providing voting information for members of the European Parliament (MEPs) on different issues*

We will conceptualize the data as a set of points, where each data point contains information about the votes of one MEP. In addition, for each MEP we have information about (i) their country and (ii) the European Parliament political group (EPG) they belong. This additional information, Country and EPG, can be seen as labels (colors) associated with the MEPs

We want to estimate the degree to which two MEPs vote in a similar manner. Thus, we need to define a function that assigns similarity values to pairs of MEPs based on their votes. Your first task is to define two different similarity functions for this problem. You should aim for definitions of similarity that are meaningful for this domain and this dataset. Present the two functions that you came up with, and explain why you chose them. Please also discuss the transformation steps required for your similarity computation, e.g., mapping categorical to numerical values, handling missing values, etc.

Your second task is to preprocess the data and compute the pairwise similarity matrix between MEPs for the two functions you defined. Depending on the efficiency of your implementation, it is possible that computing such a matrix is computationally challenging. To reduce the computational cost of the exercise, it is OK to consider only a subset of MEPs, however, make sure that you take a large enough subset. Explain your reasoning for selecting that particular subset.

Solution:

Data-preprocessing: We have 4 datasets consisting of voting data. We refer to them as EP6, EP7, EP8 and EP9. Documentation of the voting data provides explanation of the labels in the column of every vote.

Code	Meaning
0	not an MEP [at the time of the vote]
1	for
2	against
3	abstention
4	absent
5	did not vote
6	motivated [only for EP6: MEP was absent, explained by a certificate]

Table 1: Legend of the coding of columns 11 to N in RCV files.

Intuitively there are some values in the voting data that are irrelevant when it comes to explaining voting behaviour. It is hard to justify keeping 0, 4, 5 and 6 in the dataset since they don't provide any relevant information that we can utilize in measuring distances between each MEP. We replace all the 0s, 4s, 5a and 6s with NaN values provided by the numpy function np.nan(). Furthermore we adjust the dataset to only contain information of the actual votes. Since each row is indexed to a MEP, we can extract relevant information needed (Country and EPG) with this index.

Similarity functions

Function 1. Construct a similarity matrix by comparing each datapoint with all other MEPs'. If two MEPs voted the same in one vote, increase their similarity score by one. To account for MEPs voting something other than 1,2 or 3, the vote in which it occured did not count towards similarity between MEPs. Finally the similarity score is normalized by the amount of valid votes the pair of MEPs were compared on.

Function 2. For every pair of MEPs, we utilize scipys function that measures jaccard similarity. Jaccard similarity measures sets of positive occurences and disregards sets of negative occurences. Mathematically this is defined by the $J(A,B) = \frac{|A \cap B|}{|A \cup B|}$, and the distance is defined by $J_{distance}(A,B) = 1 - J(A,B)$. Focusing on the similarities rather than the differences is benifical in our case since we have to ignore a lot of data due to relevance.

Results

The scatter plots display how the members of the european parliament vote compared to each other. A data point located at origo indicates that that member votes exactly like the average voter. Longer distance between voters

means less similarity between how they vote. For every dataset we display one scatter plot where the data points are labeled by which country the member of the parliament is from, and one where the data points are labeled by which party they belong to. A trend we found was that the UK in general does not vote like the rest of the parliament. Other than that we did not identify any obvious patterns. Bigger and more clear versions of the images are attatched in the jupyter notebook containing the code.

In [1]:
```
import pandas as pd
from scipy.spatial.distance import pdist, squareform
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
EP6_xlsx = "./VoteWatch-EP-voting-data_2004-2022/EP6_RCVs_2022_06_13.xlsx"
```

In [3]:
```
EP7_xlsx = "./VoteWatch-EP-voting-data_2004-2022/EP7_RCVs_2014_06_19.xlsx"
```

In [4]:
```
EP8_xlsx = "./VoteWatch-EP-voting-data_2004-2022/EP8_RCVs_2019_06_25.xlsx"
```

In [5]:
```
EP9_xlsx = "./VoteWatch-EP-voting-data_2004-2022/EP9_RCVs_2022_06_22.xlsx"
```

In [6]:
```
#EP6
frameEP6 = pd.read_excel(EP6_xlsx, header=1)
```

In [7]:
```
#EP7
frameEP7 = pd.read_excel(EP7_xlsx, header=0)
```

In [9]:
```
#EP8
frameEP8 = pd.read_excel(EP8_xlsx, header=0)
```

In [8]:
```
#EP9
frameEP9 = pd.read_excel(EP9_xlsx, header=0)
```

In [10]:
```
frameEP6 = frameEP6.dropna()
frameEP7 = frameEP7.dropna()
frameEP8 = frameEP8.dropna()
frameEP9 = frameEP9.dropna()
```

In [11]:
```
def similarity_function_1(frame):

    frame = frame.iloc[:, 10:]

    data = frame.to_numpy()

    mask = (data >= 1) & (data <= 3)

    filtered_data = np.where(mask, data, np.nan)

    cols = data.shape[1]
    rows = data.shape[0]

    similarity_matrix = np.zeros((rows,rows))

    for i in range(rows):
        temp = filtered_data - filtered_data[i]
        nan_count = np.count_nonzero(np.isnan(temp), axis=1)
        similarity_values = abs((np.count_nonzero(temp, axis=1) - cols) / (cols-nan_count + 0.00000001))
        similarity_matrix[:][i] = similarity_values

    return similarity_matrix
```

In [12]:
```
distance_matrix_EP6 = abs(similarity_function_1(frameEP6) - 1).round(6)
distance_matrix_EP7 = abs(similarity_function_1(frameEP7) - 1).round(6)
distance_matrix_EP8 = abs(similarity_function_1(frameEP8) - 1).round(6)
distance_matrix_EP9 = abs(similarity_function_1(frameEP9) - 1).round(6)
```

In [32]:
```
distance_matrix_EP6
```

Out[32]:
```
array([[0.      , 0.51797 , 0.114202, ..., 0.434586, 0.535   , 0.607843],
       [0.51797 , 0.      , 0.548235, ..., 1.      , 1.      , 1.      ],
       [0.114202, 0.548235, 0.      , ..., 0.367347, 0.578199, 0.604743],
       ...,
       [0.434586, 1.      , 0.367347, ..., 0.      , 0.241379, 0.255034],
       [0.535   , 1.      , 0.578199, ..., 0.241379, 0.      , 0.006452],
       [0.607843, 1.      , 0.604743, ..., 0.255034, 0.006452, 0.      ]])
```

In [29]:
```
def jaccard_similarity(frame):

    obj = frame.iloc[:, 10:]
    jaccard_distances = pdist(obj, metric='jaccard')
    jaccard_similarity_scores = 1 - squareform(jaccard_distances)

    similarity_matrix = pd.DataFrame(jaccard_similarity_scores, index=frame['FullName'], columns=frame['FullNam

    return similarity_matrix
```

In [30]:
```
distance_matrix_EP6_jaccard = abs(jaccard_similarity(frameEP6) - 1).round(6)
distance_matrix_EP7_jaccard = abs(jaccard_similarity(frameEP7) - 1).round(6)
distance_matrix_EP8_jaccard = abs(jaccard_similarity(frameEP8) - 1).round(6)
distance_matrix_EP9_jaccard = abs(jaccard_similarity(frameEP9) - 1).round(6)
```

In [31]:
```
distance_matrix_EP6_jaccard
```

Out[31]:

FullName	ADAMOU, Adamos	ADWENT, Filip	AGNOLETTO, Vittorio	ALBERTINI, Gabriele	ALLISTER, James Hugh	ALVARO, Alexander Nuno	ANDERSSON, Jan	ANDREJEVS, Georgs	ANDRIA, Alfonso	ANDRIKIENE, Laima Liucija
FullName										
ADAMOU, Adamos	0.000000	0.955477	0.429585	0.727698	0.794644	0.668495	0.569124	0.661558	0.780287	0.727537
ADWENT, Filip	0.955477	0.000000	0.963543	0.962575	0.953057	0.959026	0.964510	0.956606	0.944258	0.957574
AGNOLETTO, Vittorio	0.429585	0.963543	0.000000	0.643814	0.788030	0.615099	0.507985	0.608808	0.727698	0.696241
ALBERTINI, Gabriele	0.727698	0.962575	0.643814	0.000000	0.654138	0.387321	0.471044	0.378287	0.660590	0.329408
ALLISTER, James Hugh	0.794644	0.953057	0.788030	0.654138	0.000000	0.675915	0.759316	0.700436	0.790611	0.666882
...	...	...	...	...	...	...	...	...	...	...
SOULAGE, Bernard	0.961768	1.000000	0.948540	0.935151	0.975964	0.940474	0.932570	0.936441	1.000000	0.939990
ANTOCHI, Alin Lucian Emanuel	0.933376	1.000000	0.921923	0.916438	0.960961	0.913212	0.887724	0.907727	1.000000	0.916922
D?NCIL?, Vasilica Viorica	0.937248	1.000000	0.938377	0.917245	0.943217	0.914664	0.896274	0.907404	1.000000	0.934183
BAUTISTA, Daniel	0.984191	1.000000	0.973221	0.957574	0.955154	0.966446	0.963059	0.967091	1.000000	0.950799
LIEPI?A, Liene	0.984675	1.000000	0.963381	0.958542	0.972738	0.967091	0.962736	0.966124	1.000000	0.949508

937 rows × 937 columns

In [13]:
```
def get_positions(distance_matrix):
    a = [distance_matrix.mean(axis=1)]
    b = [distance_matrix.mean(axis=0)]
    double_centered_mtrx = distance_matrix - np.transpose(a) - np.transpose(b) + distance_matrix.mean()
    eigenvalues, eigenvectors = np.linalg.eigh(double_centered_mtrx)

    sorted_indices = np.argsort(eigenvalues)[::-1]
    top_two_eigenvectors = eigenvectors[:, sorted_indices[:2]]

    coordinates = np.sqrt(np.abs(eigenvalues[sorted_indices[:2]])) * top_two_eigenvectors
    mep_coordinates = coordinates[:, :2]
    mep_positions = pd.DataFrame(mep_coordinates, columns=['X', 'Y'])
    positions = np.transpose(mep_positions.to_numpy())

    return mep_positions
```

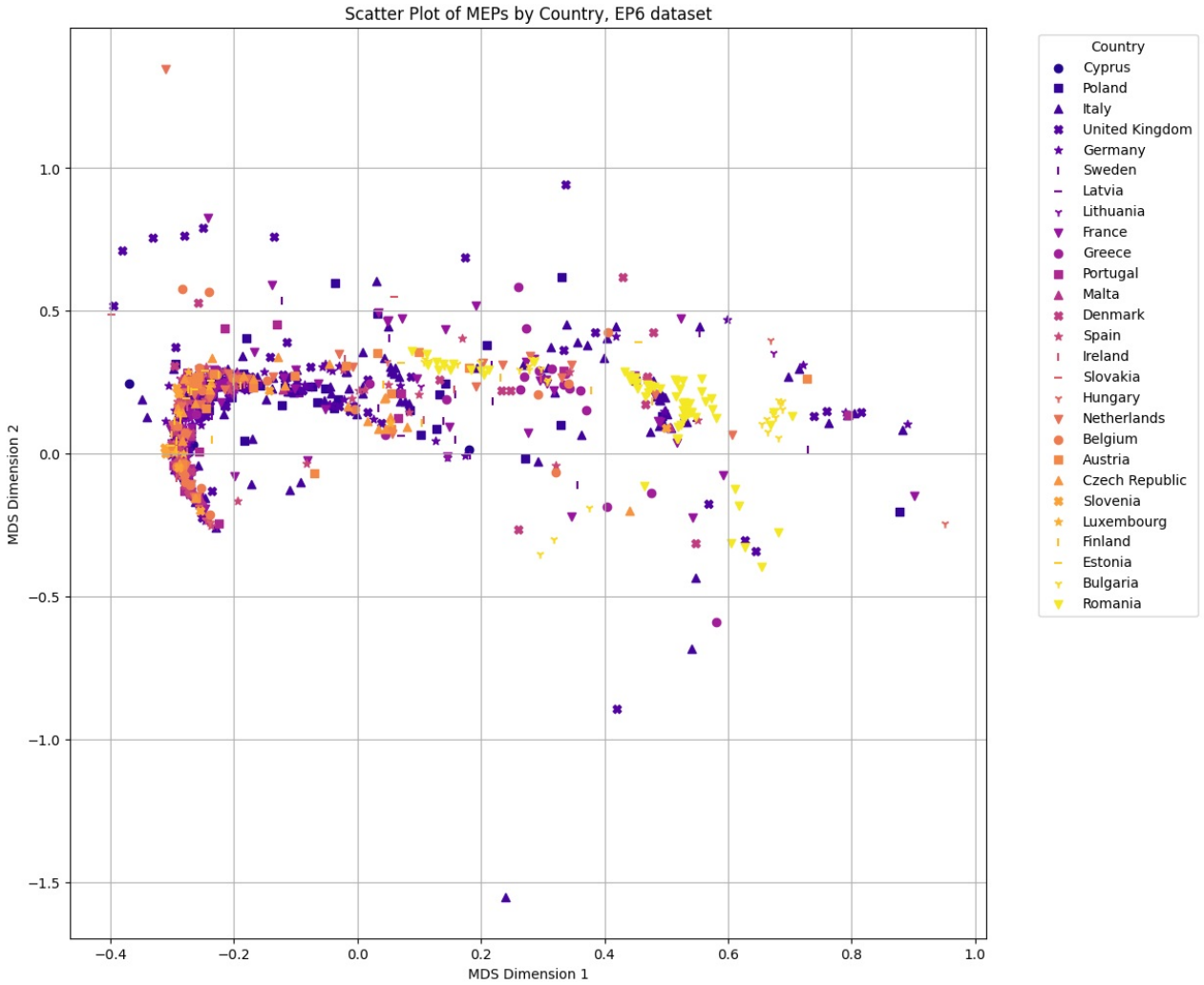
In [14]:
```
mep_positions_EP6 = get_positions(distance_matrix_EP6)
mep_positions_EP7 = get_positions(distance_matrix_EP7)
mep_positions_EP8 = get_positions(distance_matrix_EP8)
mep_positions_EP9 = get_positions(distance_matrix_EP9)
```

In [15]:
```
mepdata_EP6 = frameEP6.loc[:,['Country', 'EPG']].reset_index()
mepdata_EP6 = mepdata_EP6.drop(['index'],axis=1)

mepdata_EP7 = frameEP7.loc[:,['Country', 'EPG']].reset_index()
mepdata_EP7 = mepdata_EP7.drop(['index'],axis=1)

mepdata_EP8 = frameEP8.loc[:,['Country', 'EPG']].reset_index()
mepdata_EP8 = mepdata_EP8.drop(['index'],axis=1)

mepdata_EP9 = frameEP9.loc[:,['Country', 'EPG']].reset_index()
mepdata_EP9 = mepdata_EP9.drop(['index'],axis=1)
```

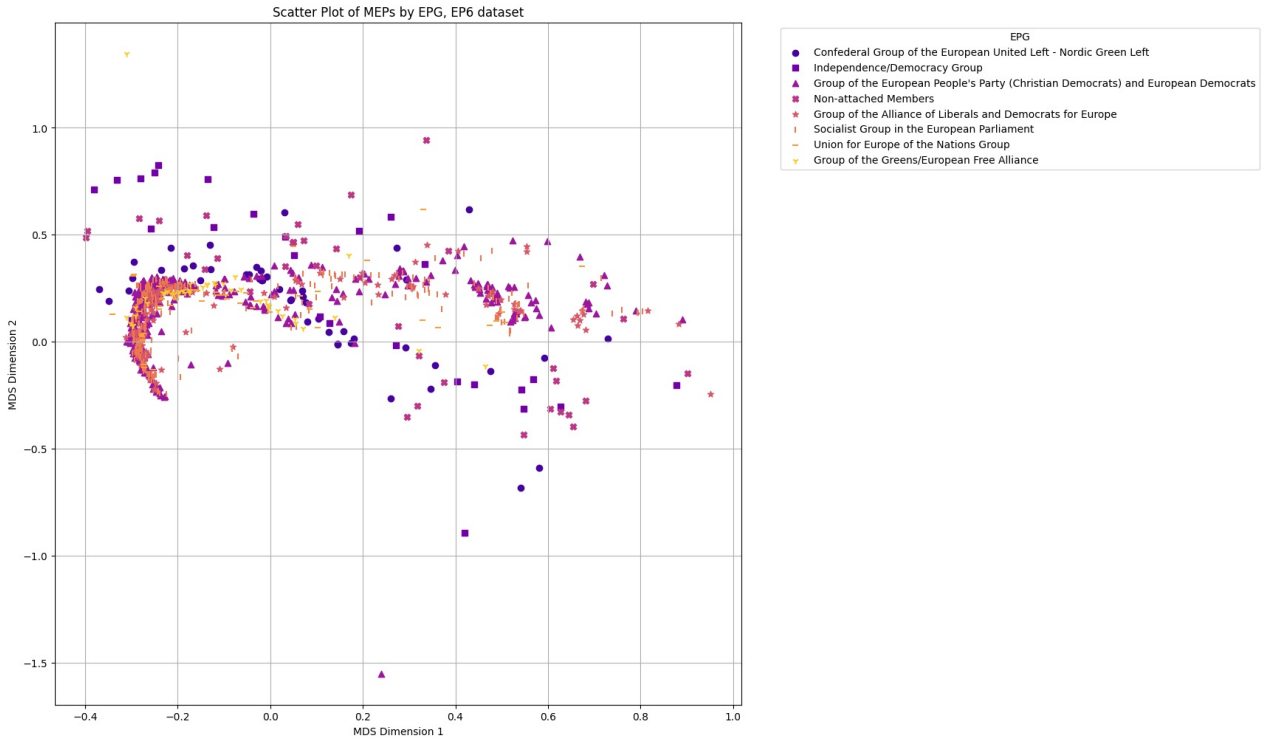
In [16]:
```
merged_pos_EP6 = pd.concat([mep_positions_EP6,mepdata_EP6],axis=1)
merged_pos_EP7 = pd.concat([mep_positions_EP7,mepdata_EP7],axis=1)
merged_pos_EP8 = pd.concat([mep_positions_EP8,mepdata_EP8],axis=1)
merged_pos_EP9 = pd.concat([mep_positions_EP9,mepdata_EP9],axis=1)
```

In [17]:
```
def plot_dataset(data,country,dataset_name):

    if country:
        label = 'Country'
    else:
        label = 'EPG'

    plt.figure(figsize=(12, 12))

    num_labels = data[label].nunique()
    palette = sns.color_palette("plasma", num_labels)

    markers = ['o', 's', '^', 'X', '*', '|', '_', '1', 'v']

    for i, x in enumerate(data[label].unique()):
        label_data = data[data[label] == x]
        plt.scatter(label_data['X'], label_data['Y'], color=palette[i], marker=markers[i % len(markers)], label=

    plt.title(f'Scatter Plot of MEPs by {label}, {dataset_name} dataset')
    plt.xlabel('MDS Dimension 1')
    plt.ylabel('MDS Dimension 2')
    plt.grid(True)

    plt.legend(title=label, bbox_to_anchor=(1.05, 1), loc='upper left')

    plt.show()
```

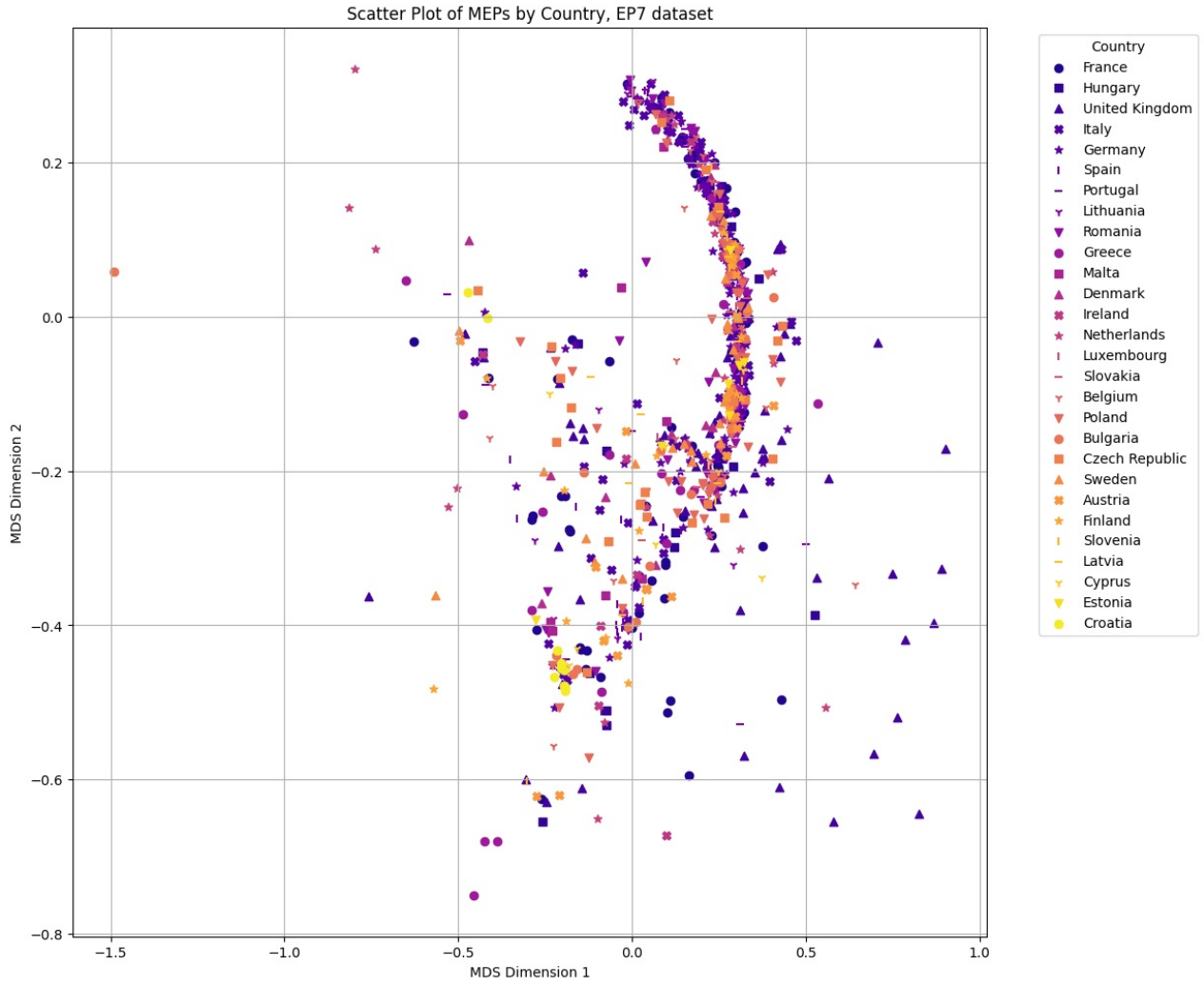
In [18]:
```
plot_dataset(merged_pos_EP6,dataset_name='EP6',country=True)
```



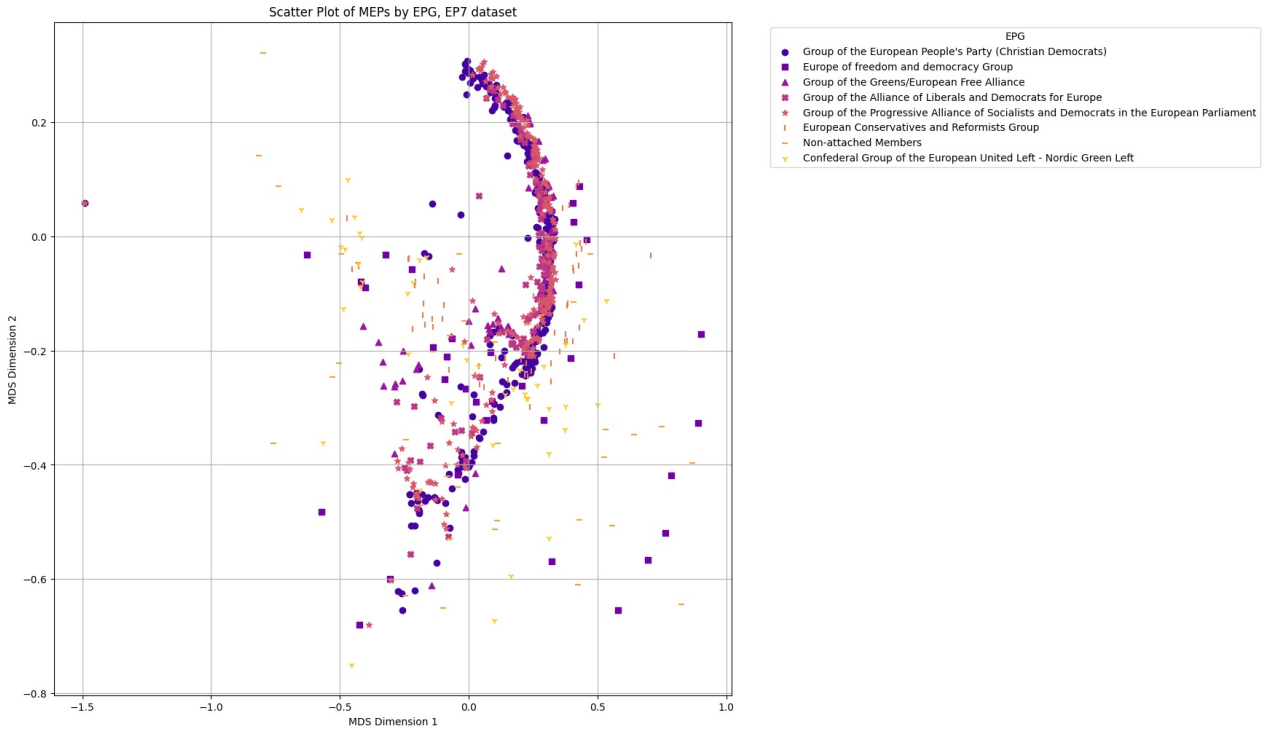
In [19]:
```
plot_dataset(merged_pos_EP6,dataset_name='EP6',country=False)
```



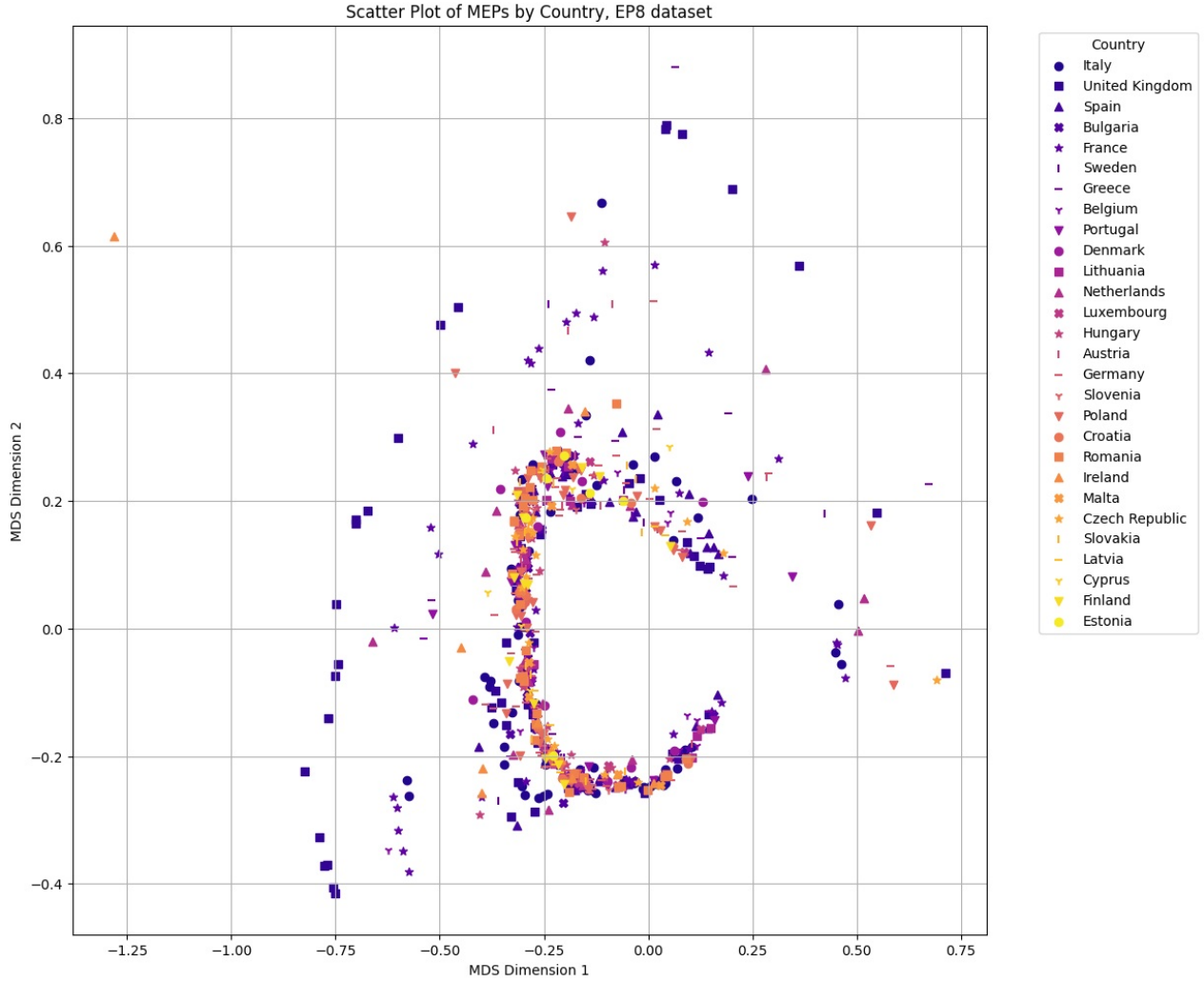
In [20]:
```
plot_dataset(merged_pos_EP7,dataset_name='EP7',country=True)
```



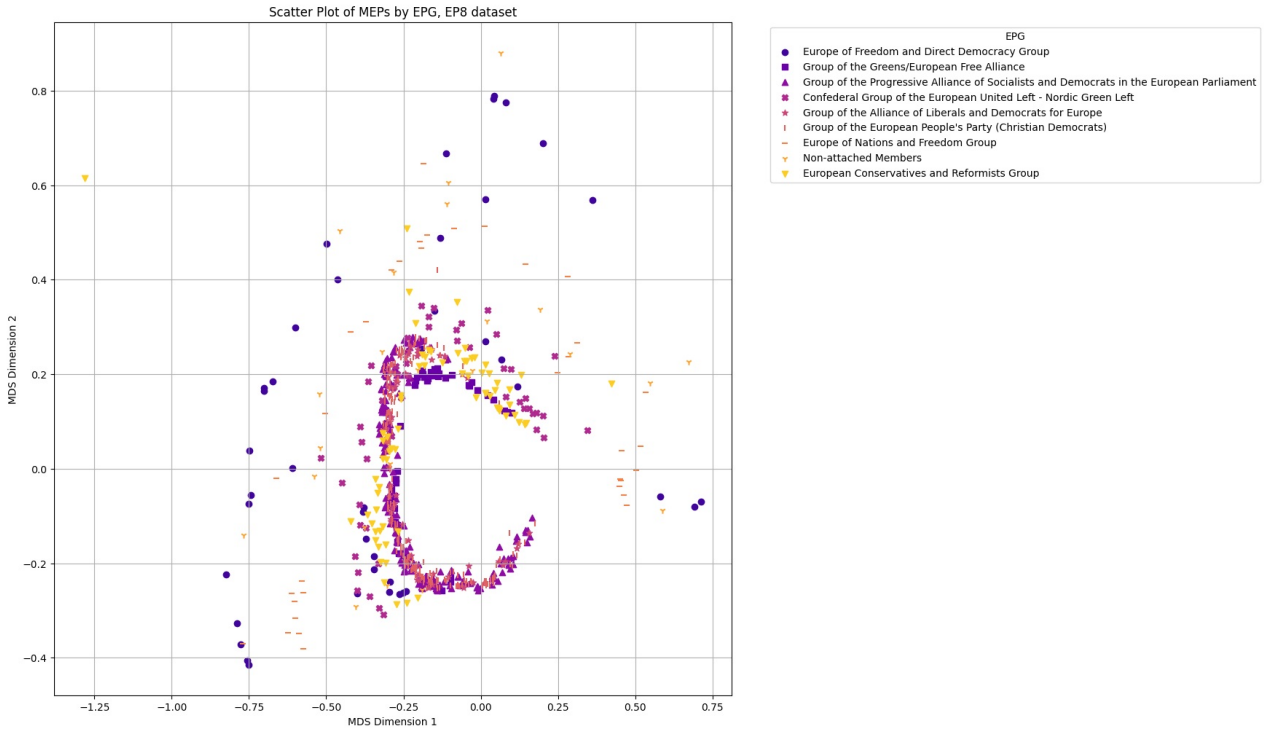
In [21]:
```
plot_dataset(merged_pos_EP7,dataset_name='EP7',country=False)
```



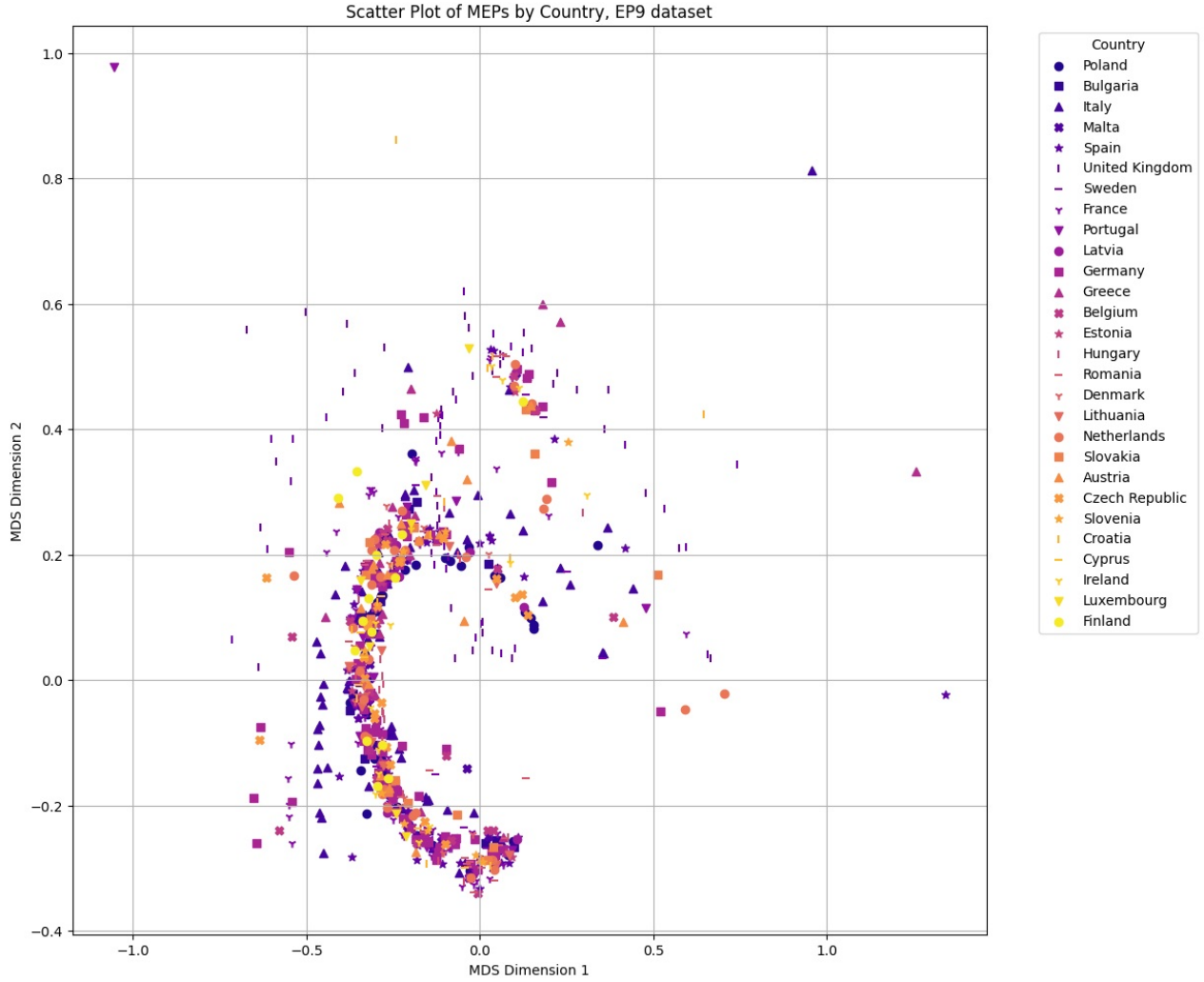
In [23]:
```
plot_dataset(merged_pos_EP8,dataset_name='EP8',country=True)
```



In [24]:
```
plot_dataset(merged_pos_EP8,dataset_name='EP8',country=False)
```



In [26]:
```
plot_dataset(merged_pos_EP9,dataset_name='EP9',country=True)
```



In [27]:
```
plot_dataset(merged_pos_EP9,dataset_name='EP9',country=False)
```