# How can I perform multidimensional scaling in Python?

Authored by

**stats writer**

June 26, 2024

Multidimensional scaling is a statistical technique used to analyze and visualize high-dimensional data by reducing its dimensionality to two or three dimensions. In Python, this technique can be performed using the scikit-learn library, specifically the "MDS" function in the "manifold" module. This function takes in a distance or dissimilarity matrix as input and outputs the coordinates of the data points in a lower-dimensional space. The resulting coordinates can then be plotted to visualize the similarities and differences between the data points. Overall, performing multidimensional scaling in Python allows for a better understanding and interpretation of complex data sets.

## Perform Multidimensional Scaling in Python

**In statistics, multidimensional scaling is a way to visualize the similarity of observations in a dataset in an abstract cartesian space (usually a 2-D space).**

**The easiest way to perform multidimensional scaling in Python is by using the MDS() function from the sklearn.manifold sub-module.**

**The following example shows how to use this function in practice.**

### Example: Multidimensional Scaling in Python

**Suppose we have the following pandas DataFrame that contains information about various basketball players:**

**import pandas as pd**

```
#create DataFrane
df = pd.DataFrame({'player': ,
'points': ,
'assists': ,
'blocks': ,
'rebounds': })

#set player column as index column
df = df.set_index('player')

#view Dataframe
print(df)
```

```
points assists blocks rebounds
player
A 4 3 7 4
B 4 2 3 5
C 6 2 6 5
D 7 5 7 6
E 8 4 5 5
F 14 8 8 8
G 16 7 8 10
H 19 6 4 4
I 25 8 2 3
J 25 10 2 2
```

K 28 11 1 2

We can use the following code to perform multidimensional scaling with the MDS() function from the sklearn.manifold module:

from sklearn.manifoldimport MDS

#perform multi-dimensional scaling
mds = MDS(random_state=0)
scaled_df = mds.fit_transform(df)

#view results of multi-dimensional scaling
print(scaled_df)

]

Each row from the original DataFrame has been reduced to an (x, y) coordinate.

We can use the following code to visualize these coordinates in a 2-D space:

import matplotlib.pyplot as plt

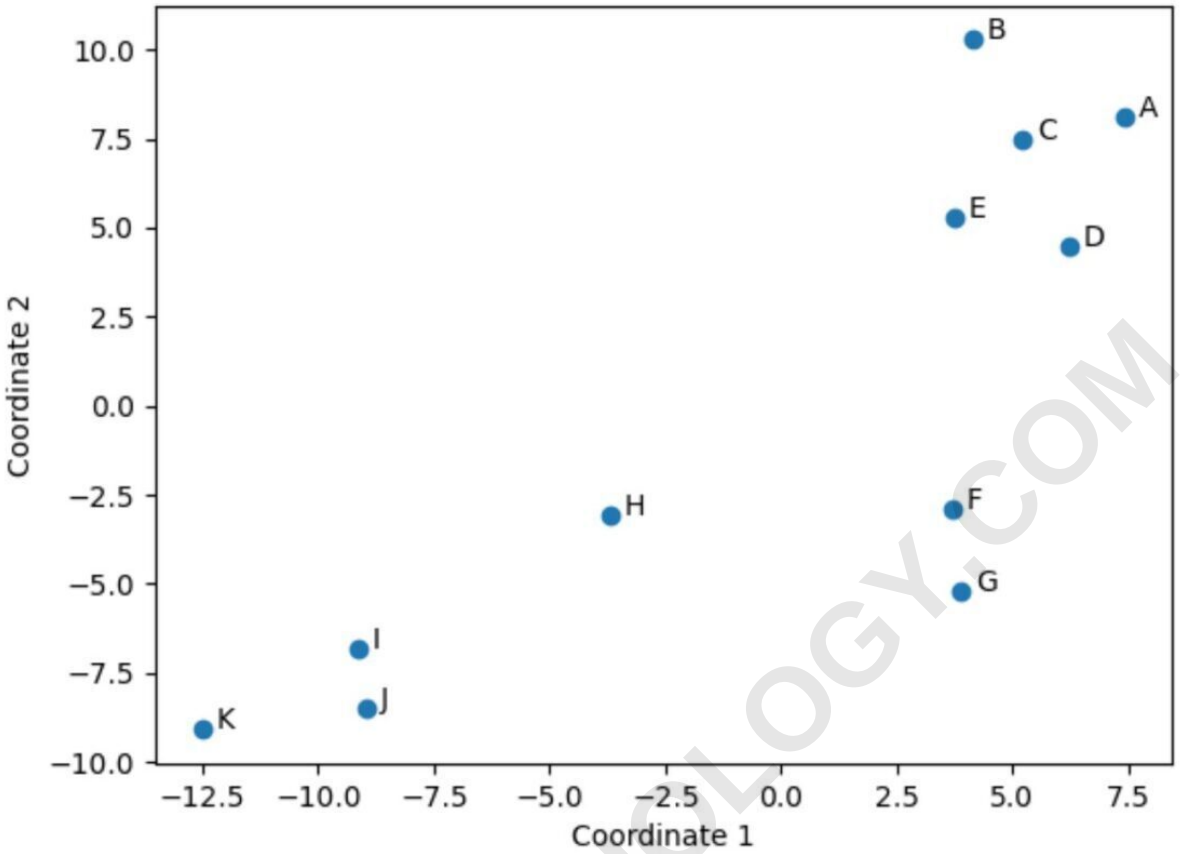
#create scatterplot

```
plt.scatter(scaled_df, scaled_df)

#add axis labels
plt.xlabel('Coordinate 1')
plt.ylabel('Coordinate 2')

#add lables to each point
for i, txt in enumerate(df.index):
plt.annotate(txt, (scaled_df+.3, scaled_df))

#display scatterplot
plt.show()
```






**Players from the original DataFrame who have similar values across the original four columns (points, assists, blocks, and rebounds) are located close to each other in the plot.**

**For example, players F and G are located close to each other. Here are their values from the original DataFrame:**

**#select rows with index labels 'F' and 'G'**

**df.loc]**

points assists blocks rebounds
player
F 14 8 8 8
G 16 7 8 10

Their values for points, assists, blocks, and rebounds are all fairly similar, which explains why they're located so close together in the 2-D plot.

By contrast, consider players B and K who are located far apart in the plot.

#select rows with index labels 'B' and 'K'
df.loc]

points assists blocks rebounds
player
B 4 2 3 5
K 28 11 1 2

Thus, the 2-D plot is a nice way to visualize how similar each players are across all of the variables in the DataFframe.

Players who have similar stats are grouped close

**together while players who have very different stats are located far apart from each other in the plot.**

**The following tutorials explain how to perform other common tasks in Python:**