scalar = **StandardScaler**().fit(new\_cust)

**Standardize features by removing the mean and scaling to unit variance**

The idea behind StandardScaler is that it will transform your data such that its distribution will have a mean value 0 and standard deviation of 1

The preprocessing module further provides a utility class **[StandardScaler](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html" \l "sklearn.preprocessing.StandardScaler" \o "sklearn.preprocessing.StandardScaler)** that implements the Transformer API to compute the mean and standard deviation on a training set so as to be able to later reapply the same transformation on the testing set.

X = scalar.transform(new\_cust)

**Transform**(0 method can be used to sparse the input

**Perform standardization by centering and scaling**

**scipy.spatial.distance.squareform**

Convert a vector-form distance vector to a square-form distance matrix, and vice-versa.

**squareform(***X***,***force='no'***,***checks=True***)**

**X*ndarray***

Either a condensed or redundant distance matrix.

**force*str, optional***

As with MATLAB(TM), if force is equal to 'tovector' or 'tomatrix', the input will be treated as a distance matrix or distance vector respectively.

**checks*bool*** : If set to False, no checks will be made for matrix symmetry nor zero diagonals.

**Returns**

**Y*ndarray***

If a condensed distance matrix is passed, a redundant one is returned, or if a redundant one is passed, a condensed distance matrix is returned.

**PDIST**

**pdist(***X***,***metric='euclidean'***,***\*args***,***\*\*kwargs***)**

Pairwise distances between observations in n-dimensional space.

X=n dimensional array

Metric: ***str or function, optional.*** The distance metric to use (cityblock, Euclidean, Chebyshev, minkowski,)

**args*tuple. Deprecated.***

Additional arguments should be passed as keyword arguments

**\*kwargs*dict, optional***

Extra arguments to *metric*: refer to each metric documentation for a list of all possible arguments.

**Returns**

**Y*ndarray***

Returns a condensed distance matrix Y

**from scipy.cluster.hierarchy import linkage**

Perform hierarchical/agglomerative clustering.

**linkage(***y***,***method='single'***,***metric='euclidean'***,***optimal\_ordering=False)*

*y= array*…distance matrix

method= linkage algorithm to use

**metric=** The distance metric to use in the case that y is a collection of observation vectors

**optimal\_ordering*bool, optional***

If True, the linkage matrix will be reordered so that the distance between successive leaves is minimal. This results in a more intuitive tree structure when the data are visualized. defaults to False, because this algorithm can be slow, particularly on large datasets

**Returns**

**Z*ndarray***

The hierarchical clustering encoded as a linkage matrix.

# scipy.cluster.hierarchy.dendrogram

Plot the hierarchical clustering as a dendrogram.

The dendrogram illustrates how each cluster is composed by drawing a U-shaped link between a non-singleton cluster and its children. The top of the U-link indicates a cluster merge. The two legs of the U-link indicate which clusters were merged. The length of the two legs of the U-link represents the distance between the child clusters. It is also the cophenetic distance between original observations in the two children clusters.

**scipy.cluster.hierarchy.dendrogram(***Z***,***p=30***,***truncate\_mode=None***,***color\_threshold=None***,***get\_leaves=True***,***orientation='top'***,***labels=None***,***count\_sort=False***,***distance\_sort=False***,***show\_leaf\_counts=True***,***no\_plot=False***,***no\_labels=False***,***leaf\_font\_size=None***,***leaf\_rotation=None***,***leaf\_label\_func=None***,***show\_contracted=False***,***link\_color\_func=None***,***ax=None***,***above\_threshold\_color='C0'***)**

Z= array

P=parameter for truncate mode