Illustration of **HDBSCAN** (Hierarchical Density-Based Spatial Clustering of Applications with Noise

HDBSCAN uses a density-based approach which makes few implicit assumptions about the clusters. It is a non-parametric method that looks for a cluster hierarchy shaped by the multivariate modes of the underlying distribution. Rather than looking for clusters with a particular shape, it looks for regions of the data that are denser than the surrounding space.

```
from sklearn.datasets import make_blobs
import pandas as pd
blobs, labels = make_blobs(n_samples=2000, n_features=10)
pd.DataFrame(blobs).head()
```

	0	1	2	3	4	5	6	7	8	9
0	5.479981	7.527964	-6.301394	-9.160561	-9.434875	2.258409	-0.272756	-6.869747	-4.257054	1.519784
1	4.295540	9.303668	-6.757800	-9.586401	-8.143580	2.886716	-0.651843	-6.840010	-3.734954	2.712942
2	4.053331	9.569083	-6.301322	-10.174186	-8.406922	2.429171	-0.924639	-7.570285	-3.699698	2.098373
3	7.363827	-5.816362	3.782239	4.917067	8.632500	2.538719	0.590096	0.486736	4.700632	-6.873296
4	3.440991	8.040023	-5.891293	-9.673307	-9.564448	1.920921	-0.578209	-7.777130	-2.847379	0.946074

```
# Load hdbscan module
!pip install hdbscan
import hdbscan
```

```
Collecting hdbscan
  Downloading https://files.pythonhosted.org/packages/22/2f/2423d844072f007a74214c1adc46260e45f034bb1679ccadfbb8a601f647/hdbscan-
     |████████████████████████████████| 4.7MB 2.9MB/s
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
    Preparing wheel metadata ... done
Requirement already satisfied: cython>=0.27 in /usr/local/lib/python3.6/dist-packages (from hdbscan) (0.29.20)
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages (from hdbscan) (1.12.0)
Requirement already satisfied: scipy>=0.9 in /usr/local/lib/python3.6/dist-packages (from hdbscan) (1.4.1)
Requirement already satisfied: numpy>=1.16.0 in /usr/local/lib/python3.6/dist-packages (from hdbscan) (1.18.5)
Requirement already satisfied: scikit-learn>=0.17 in /usr/local/lib/python3.6/dist-packages (from hdbscan) (0.22.2.post1)
Requirement already satisfied: joblib in /usr/local/lib/python3.6/dist-packages (from hdbscan) (0.15.1)
Building wheels for collected packages: hdbscan
  Building wheel for hdbscan (PEP 517) ... done
  Created wheel for hdbscan: filename=hdbscan-0.8.26-cp36-cp36m-linux_x86_64.whl size=2307170 sha256=f6b902d7534e9945ba6e54547f0
  Stored in directory: /root/.cache/pip/wheels/82/38/41/372f034d8abd271ef7787a681e0a47fc05d472683a7eb088ed
Successfully built hdbscan
Installing collected packages: hdbscan
Successfully installed hdbscan-0.8.26
```

```
clusterer = hdbscan.HDBSCAN()
clusterer.fit(blobs)
```

```
HDBSCAN(algorithm='best', allow_single_cluster=False, alpha=1.0,
        approx_min_span_tree=True, cluster_selection_epsilon=0.0,
        cluster_selection_method='eom', core_dist_n_jobs=4,
        gen_min_span_tree=False, leaf_size=40,
        match_reference_implementation=False, memory=Memory(location=None),
        metric='euclidean', min_cluster_size=5, min_samples=None, p=None,
        prediction_data=False)
```

```
# Here is how we get the clusters
clusterer.labels_
```

```
array([0, 0, 0, ..., 1, 2, 0])
```

```
# We can determine the number of clusters found by finding the largest cluster label
clusterer.labels_.max()
```

```
2
```

```
"""Each data point is assigned a cluster membership score ranging from 0.0 to 1.0. A score of 0.0 represents
a sample that is not in the cluster at all (all noise points will get this score) while a score of 1.0 represents
a sample that is at the heart of the cluster (note that this is not the spatial centroid notion of core)."""
```

```
'Each data point is assigned a cluster membership score ranging from 0.0 to 1.0. A score of 0.0 represents \na sample that is not
```

```
# Provide the cluster probabilities
clusterer.probabilities_
```

```
array([0.84389614, 1.        , 1.        , ..., 0.75285225, 0.72261508,
       1.        ])
```

```
# What metrics support HDBSCAN?
hdbscan.dist_metrics.METRIC_MAPPING
```

```
{'arccos': hdbscan.dist_metrics.ArccosDistance,
 'braycurtis': hdbscan.dist_metrics.BrayCurtisDistance,
 'canberra': hdbscan.dist_metrics.CanberraDistance,
 'chebyshev': hdbscan.dist_metrics.ChebyshevDistance,
 'cityblock': hdbscan.dist_metrics.ManhattanDistance,
 'cosine': hdbscan.dist_metrics.ArccosDistance,
 'dice': hdbscan.dist_metrics.DiceDistance,
 'euclidean': hdbscan.dist_metrics.EuclideanDistance,
 'hamming': hdbscan.dist_metrics.HammingDistance,
 'haversine': hdbscan.dist_metrics.HaversineDistance,
 'infinity': hdbscan.dist_metrics.ChebyshevDistance,
 'jaccard': hdbscan.dist_metrics.JaccardDistance,
 'kulsinski': hdbscan.dist_metrics.KulsinskiDistance,
 'l1': hdbscan.dist_metrics.ManhattanDistance,
 'l2': hdbscan.dist_metrics.EuclideanDistance,
 'mahalanobis': hdbscan.dist_metrics.MahalanobisDistance,
 'manhattan': hdbscan.dist_metrics.ManhattanDistance,
 'matching': hdbscan.dist_metrics.MatchingDistance,
 'minkowski': hdbscan.dist_metrics.MinkowskiDistance,
 'p': hdbscan.dist_metrics.MinkowskiDistance,
 'pyfunc': hdbscan.dist_metrics.PyFuncDistance,
 'rogerstanimoto': hdbscan.dist_metrics.RogersTanimotoDistance,
 'russellrao': hdbscan.dist_metrics.RussellRaoDistance,
 'seuclidean': hdbscan.dist_metrics.SEuclideanDistance,
 'sokalmichener': hdbscan.dist_metrics.SokalMichenerDistance,
 'sokalsneath': hdbscan.dist_metrics.SokalSneathDistance,
 'wminkowski': hdbscan.dist_metrics.WMinkowskiDistance}
```

```
# Say we are looking at Manhattan distance
clusterer = hdbscan.HDBSCAN(metric='manhattan')
clusterer.fit(blobs)
clusterer.labels_
```

```
array([0, 0, 0, ..., 1, 2, 0])
```

```
"""What if you don’t have a nice set of points in a vector space, but only have a pairwise distance matrix providing
the distance between each pair of points? This is a common situation."""
```

```
'What if you don’t have a nice set of points in a vector space, but only have a pairwise distance matrix providing \nthe distance
```

```
from sklearn.metrics.pairwise import pairwise_distances
distance_matrix = pairwise_distances(blobs)
clusterer = hdbscan.HDBSCAN(metric='precomputed')
clusterer.fit(distance_matrix)
clusterer.labels_
```

```
array([0, 0, 0, ..., 1, 2, 0])
```