

## Distance Measures and Clustering Group T4-2

1.0

Generated by Doxygen 1.9.1



<b>1 Description</b>	<b>1</b>
1.1 Frontend	1
1.2 Documentation	1
1.3 Dependencies and Sources	1
1.4 Authors	2
<b>2 Namespace Index</b>	<b>3</b>
2.1 Namespace List	3
<b>3 Hierarchical Index</b>	<b>5</b>
3.1 Class Hierarchy	5
<b>4 Class Index</b>	<b>7</b>
4.1 Class List	7
<b>5 File Index</b>	<b>9</b>
5.1 File List	9
<b>6 Namespace Documentation</b>	<b>11</b>
6.1 clustering Namespace Reference	11
6.2 comparison_plots Namespace Reference	11
6.2.1 Variable Documentation	12
6.2.1.1 all_dist	12
6.2.1.2 all_k	12
6.2.1.3 all_kalgos	12
6.2.1.4 all_kalgos_df	13
6.2.1.5 all_kalgos_int	13
6.2.1.6 ax	13
6.2.1.7 bbox_to_anchor	13
6.2.1.8 borderaxespad	13
6.2.1.9 c	13
6.2.1.10 cluster	13
6.2.1.11 clustered_data	14
6.2.1.12 clusters	14
6.2.1.13 d	14
6.2.1.14 data	14
6.2.1.15 datasets	14
6.2.1.16 distances	14
6.2.1.17 fig	14
6.2.1.18 figsize	14
6.2.1.19 hue	15
6.2.1.20 l1	15
6.2.1.21 index	15
6.2.1.22 index_ext_eval	15

6.2.1.23 index_int_eval	15
6.2.1.24 index_score	15
6.2.1.25 index_scores	15
6.2.1.26 k	16
6.2.1.27 kalgoclass	16
6.2.1.28 kalgos	16
6.2.1.29 kind	16
6.2.1.30 labels	16
6.2.1.31 legend	16
6.2.1.32 loc	16
6.2.1.33 num_of_classes	17
6.2.1.34 predicted	17
6.2.1.35 results	17
6.2.1.36 s	17
6.2.1.37 seed	17
6.2.1.38 stuff	17
6.2.1.39 style	17
6.2.1.40 title	17
6.2.1.41 x	18
6.2.1.42 y	18
6.3 dbscan Namespace Reference	18
6.4 dbscan_comparision_plots Namespace Reference	18
6.4.1 Function Documentation	18
6.4.1.1 plot_kdist()	18
6.4.2 Variable Documentation	19
6.4.2.1 datasets	19
6.4.2.2 distances	19
6.4.2.3 heu	19
6.4.2.4 kdists	19
6.5 dbscan_heuristic Namespace Reference	19
6.6 dbscan_solutions Namespace Reference	19
6.6.1 Function Documentation	20
6.6.1.1 save_score()	20
6.6.2 Variable Documentation	20
6.6.2.1 alg	20
6.6.2.2 best_results	20
6.6.2.3 centers	20
6.6.2.4 clustered_data	21
6.6.2.5 clusters	21
6.6.2.6 datasets	21
6.6.2.7 distances	21
6.6.2.8 external	21

6.6.2.9 indices	21
6.6.2.10 internal	21
6.6.2.11 score	21
6.6.2.12 seed	22
6.7 heuristic_web Namespace Reference	22
6.7.1 Variable Documentation	22
6.7.1.1 cluster_dist	22
6.7.1.2 cluster_dist_desc	23
6.7.1.3 col1	23
6.7.1.4 col2	23
6.7.1.5 dataset	23
6.7.1.6 df	23
6.7.1.7 heu	23
6.7.1.8 k	24
6.7.1.9 kdist	24
6.7.1.10 key	24
6.7.1.11 line	24
6.7.1.12 nearest	24
6.7.1.13 page_icon	24
6.7.1.14 page_title	24
6.7.1.15 points	25
6.7.1.16 reverse	25
6.7.1.17 rules	25
6.7.1.18 selectors	25
6.7.1.19 submit_button	25
6.7.1.20 text	25
6.7.1.21 textp	25
6.7.1.22 use_container_width	26
6.7.1.23 yaxis	26
6.8 indices Namespace Reference	26
6.9 kmeans Namespace Reference	26
6.10 kmedians Namespace Reference	26
6.11 kmedoids Namespace Reference	26
6.12 make_timing_table Namespace Reference	27
6.12.1 Variable Documentation	27
6.12.1.1 axis	27
6.12.1.2 dataset_cluster	27
6.12.1.3 datasets	27
6.12.1.4 dbscan_comb	27
6.12.1.5 distances	28
6.12.1.6 kalgoclass	28
6.12.1.7 kalgos	28

6.12.1.8 table_array . . . . .	28
6.12.1.9 timing_results . . . . .	28
6.12.1.10 timing_table . . . . .	28
6.13 result_calculation Namespace Reference . . . . .	28
6.13.1 Variable Documentation . . . . .	29
6.13.1.1 alg . . . . .	29
6.13.1.2 c . . . . .	29
6.13.1.3 centers . . . . .	29
6.13.1.4 clusters . . . . .	29
6.13.1.5 d . . . . .	29
6.13.1.6 datasets . . . . .	29
6.13.1.7 distances . . . . .	29
6.13.1.8 eps . . . . .	30
6.13.1.9 k . . . . .	30
6.13.1.10 kalgoclass . . . . .	30
6.13.1.11 kalgos . . . . .	30
6.13.1.12 m . . . . .	30
6.13.1.13 minpts . . . . .	30
6.13.1.14 results . . . . .	30
6.13.1.15 s . . . . .	31
6.13.1.16 seed . . . . .	31
6.14 results Namespace Reference . . . . .	31
6.15 SessionState Namespace Reference . . . . .	31
6.15.1 Function Documentation . . . . .	31
6.15.1.1 get() . . . . .	32
6.16 timing Namespace Reference . . . . .	32
6.16.1 Variable Documentation . . . . .	32
6.16.1.1 after . . . . .	33
6.16.1.2 alg . . . . .	33
6.16.1.3 before . . . . .	33
6.16.1.4 centers . . . . .	33
6.16.1.5 clusters . . . . .	33
6.16.1.6 dataset_cluster . . . . .	33
6.16.1.7 dbscan_comb . . . . .	33
6.16.1.8 distances . . . . .	34
6.16.1.9 kalgoclass . . . . .	34
6.16.1.10 kalgos . . . . .	34
6.16.1.11 n . . . . .	34
6.16.1.12 time . . . . .	34
6.16.1.13 timing . . . . .	34
6.17 web_frontend Namespace Reference . . . . .	34
6.17.1 Function Documentation . . . . .	36

6.17.1.1 clustering()	36
6.17.1.2 create_cluster()	36
6.17.1.3 plotting()	37
6.17.2 Variable Documentation	37
6.17.2.1 add_result	37
6.17.2.2 base	37
6.17.2.3 cluster	37
6.17.2.4 cluster_algo	38
6.17.2.5 cluster_algo_class	38
6.17.2.6 cluster_dist	38
6.17.2.7 cluster_dist_desc	38
6.17.2.8 clusterset	38
6.17.2.9 col1	38
6.17.2.10 col2	38
6.17.2.11 data_select	39
6.17.2.12 dataexpander	39
6.17.2.13 dataset	39
6.17.2.14 datasetinformation	39
6.17.2.15 datasets	39
6.17.2.16 desc	39
6.17.2.17 desc_list	39
6.17.2.18 df	40
6.17.2.19 df_for	40
6.17.2.20 dfclusterdata	40
6.17.2.21 epsilon	40
6.17.2.22 fig1	40
6.17.2.23 fig2	40
6.17.2.24 l1	40
6.17.2.25 index_eval	41
6.17.2.26 indices_data	41
6.17.2.27 k_value	41
6.17.2.28 labels	41
6.17.2.29 minpts	41
6.17.2.30 page_icon	41
6.17.2.31 page_title	41
6.17.2.32 params	42
6.17.2.33 perp	42
6.17.2.34 precalc	42
6.17.2.35 predicted	42
6.17.2.36 reset_tmp	42
6.17.2.37 resulthandler	42
6.17.2.38 results	42

6.17.2.39 score	42
6.17.2.40 seaplots	43
6.17.2.41 seed	43
6.17.2.42 seeded	43
6.17.2.43 session_state	43
6.17.2.44 stats	43
6.17.2.45 title	43
6.17.2.46 use_container_width	43
6.17.2.47 val	43
<b>7 Class Documentation</b>	<b>45</b>
7.1 clustering.Clustering Class Reference	45
7.1.1 Detailed Description	46
7.1.2 Constructor & Destructor Documentation	46
7.1.2.1 __init__()	46
7.1.3 Member Function Documentation	46
7.1.3.1 cluster()	47
7.1.3.2 house_load()	47
7.1.3.3 load_data()	47
7.1.3.4 pyc_metric()	47
7.1.4 Member Data Documentation	48
7.1.4.1 data	48
7.1.4.2 datadf	48
7.1.4.3 dataset	48
7.1.4.4 labels	48
7.1.4.5 metric	49
7.1.4.6 seed	49
7.2 dbscan.DBSCANClustering Class Reference	49
7.2.1 Detailed Description	50
7.2.2 Constructor & Destructor Documentation	50
7.2.2.1 __init__()	50
7.2.3 Member Function Documentation	50
7.2.3.1 cluster()	50
7.2.3.2 package()	51
7.2.4 Member Data Documentation	51
7.2.4.1 data	51
7.2.4.2 dataset	51
7.2.4.3 labels	51
7.2.4.4 metric	52
7.3 dbscan_heuristic.DBSCANHeuristic Class Reference	52
7.3.1 Detailed Description	52
7.3.2 Constructor & Destructor Documentation	52



7.3.2.1 <code>__init__()</code>	53
7.3.3 Member Function Documentation	53
7.3.3.1 <code>kdist()</code>	53
7.3.3.2 <code>plot_kdist()</code>	53
7.3.3.3 <code>set_dataset()</code>	53
7.3.3.4 <code>set_metric()</code>	54
7.3.4 Member Data Documentation	54
7.3.4.1 <code>clustering</code>	54
7.3.4.2 <code>k</code>	54
7.3.4.3 <code>metric</code>	54
7.4 <code>indices</code> . <code>Indices</code> Class Reference	55
7.4.1 Detailed Description	55
7.4.2 Constructor & Destructor Documentation	55
7.4.2.1 <code>__init__()</code>	55
7.4.3 Member Function Documentation	56
7.4.3.1 <code>index_external()</code>	56
7.4.3.2 <code>index_internal()</code>	56
7.4.4 Member Data Documentation	56
7.4.4.1 <code>cluster_calc</code>	56
7.4.4.2 <code>cluster_label</code>	57
7.5 <code>kmeans</code> . <code>kmeansClustering</code> Class Reference	57
7.5.1 Detailed Description	57
7.5.2 Constructor & Destructor Documentation	58
7.5.2.1 <code>__init__()</code>	58
7.5.3 Member Function Documentation	58
7.5.3.1 <code>cluster()</code>	58
7.5.4 Member Data Documentation	58
7.5.4.1 <code>data</code>	59
7.5.4.2 <code>dataset</code>	59
7.5.4.3 <code>labels</code>	59
7.5.4.4 <code>metric</code>	59
7.5.4.5 <code>seed</code>	59
7.6 <code>kmedians</code> . <code>kmediansClustering</code> Class Reference	60
7.6.1 Detailed Description	60
7.6.2 Constructor & Destructor Documentation	60
7.6.2.1 <code>__init__()</code>	60
7.6.3 Member Function Documentation	61
7.6.3.1 <code>cluster()</code>	61
7.6.4 Member Data Documentation	61
7.6.4.1 <code>data</code>	61
7.6.4.2 <code>dataset</code>	61
7.6.4.3 <code>labels</code>	62

7.6.4.4 metric	62
7.6.4.5 seed	62
7.7 kmedoids.kmedoidsClustering Class Reference	62
7.7.1 Detailed Description	63
7.7.2 Constructor & Destructor Documentation	63
7.7.2.1 __init__()	63
7.7.3 Member Function Documentation	63
7.7.3.1 cluster()	64
7.7.3.2 package()	64
7.7.4 Member Data Documentation	64
7.7.4.1 data	64
7.7.4.2 dataset	65
7.7.4.3 labels	65
7.7.4.4 metric	65
7.7.4.5 seed	65
7.8 results.Results Class Reference	65
7.8.1 Detailed Description	66
7.8.2 Constructor & Destructor Documentation	66
7.8.2.1 __init__()	66
7.8.3 Member Function Documentation	66
7.8.3.1 get_path()	67
7.8.3.2 load_set()	67
7.8.3.3 save_set()	68
7.8.3.4 set_exists()	68
7.8.4 Member Data Documentation	68
7.8.4.1 parent	69
7.9 SessionState.SessionState Class Reference	69
7.9.1 Constructor & Destructor Documentation	69
7.9.1.1 __init__()	69
<b>8 File Documentation</b>	<b>71</b>
8.1 clustering.py File Reference	71
8.1.1 Detailed Description	71
8.2 comparison_plots.py File Reference	71
8.2.1 Detailed Description	72
8.3 dbscan.py File Reference	73
8.3.1 Detailed Description	73
8.4 dbscan_comparison_plots.py File Reference	73
8.5 dbscan_heuristic.py File Reference	73
8.5.1 Detailed Description	74
8.6 dbscan_solutions.py File Reference	74
8.7 heuristic_web.py File Reference	74

---

8.7.1 Detailed Description . . . . .	75
8.8 indices.py File Reference . . . . .	75
8.8.1 Detailed Description . . . . .	76
8.9 kmeans.py File Reference . . . . .	76
8.9.1 Detailed Description . . . . .	76
8.10 kmedians.py File Reference . . . . .	76
8.10.1 Detailed Description . . . . .	77
8.11 kmedoids.py File Reference . . . . .	77
8.11.1 Detailed Description . . . . .	77
8.12 make_timing_table.py File Reference . . . . .	77
8.12.1 Detailed Description . . . . .	78
8.13 result_calculation.py File Reference . . . . .	78
8.14 results.py File Reference . . . . .	78
8.14.1 Detailed Description . . . . .	79
8.15 SessionState.py File Reference . . . . .	79
8.15.1 Detailed Description . . . . .	79
8.15.1.1 Usage . . . . .	79
8.16 timing.py File Reference . . . . .	80
8.17 web_frontend.py File Reference . . . . .	80
8.17.1 Detailed Description . . . . .	82
8.18 /home/nordegraf/Uni/8.Semester/DataScienceI/datascience1_group42/README.md File Reference	82
<b>Index</b>	<b>83</b>



# Chapter 1

## Description

This is a group project done for the Lecture "Data Science 1" at the Goethe University Frankfurt exploring the effects of different distance measures on distance-based clustering algorithms.

### 1.1 Frontend

The web frontend is accessible [here](#).

A user manual can be looked up [here](#)

### 1.2 Documentation

The doxygen Documentation of the codebase can be accessed [here](#).

A PDF Documentation is also available in the docs directory (file: [documentation.pdf](#))

### 1.3 Dependencies and Sources

The [SessionState.py](#) is directly taken from a [gist](#) by Thiago Teixeira, user [tvst](#) on github. We take absolutly no credit for it!

The code for the altair chart used for displaying the DBSCAN heuristic is based heavily upon the multiline tooltip example from the altair example gallery ( [Link](#)).

The project depends on following python libraries:

- [matplotlib](#)
- [numpy](#)
- [pandas](#)
- [pyclustering](#)
- [scikit-learn](#)
- [scikit-learn-extra](#)
- [seaborn](#)
- [streamlit](#)
- [altair](#)

## 1.4 Authors

- Niklas Conen
- Jonas Elpelt
- Franziska Hicking
- Julian Rummel

So long, and thanks for all the fish

## Chapter 2

# Namespace Index

### 2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

<a href="#">clustering</a>	11
<a href="#">comparison_plots</a>	11
<a href="#">dbscan</a>	18
<a href="#">dbscan_comparison_plots</a>	18
<a href="#">dbscan_heuristic</a>	19
<a href="#">dbscan_solutions</a>	19
<a href="#">heuristic_web</a>	22
<a href="#">indices</a>	26
<a href="#">kmeans</a>	26
<a href="#">kmedians</a>	26
<a href="#">kmedoids</a>	26
<a href="#">make_timing_table</a>	27
<a href="#">result_calculation</a>	28
<a href="#">results</a>	31
<a href="#">SessionState</a>	31
<a href="#">timing</a>	32
<a href="#">web_frontend</a>	34





## Chapter 3

# Hierarchical Index

### 3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

clustering.Clustering . . . . .	45
dbscan.DBSCANClustering . . . . .	49
kmeans.kmeansClustering . . . . .	57
kmedians.kmediansClustering . . . . .	60
kmedoids.kmedoidsClustering . . . . .	62
dbscan_heuristic.DBSCANHeuristic . . . . .	52
indices.Indices . . . . .	55
object	
SessionState.SessionState . . . . .	69
results.Results . . . . .	65



## Chapter 4

# Class Index

### 4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

<a href="#">clustering.Clustering</a>	Base Class for all subsequent clustering algorithms implements all functions needed for running the different cluster algorithms . . . . .	45
<a href="#">dbscan.DBSCANClustering</a>	Implements DBSCAN Clustering uses the scikit-learn DBSCAN implementation . . . . .	49
<a href="#">dbscan_heuristic.DBSCANHeuristic</a>	Implements the DBSCAN heuristic proposed in the original DBSCAN paper: . . . . .	52
<a href="#">indices.Indices</a>	Calculates <a href="#">Indices</a> for computed cluster labels uses the scikit library . . . . .	55
<a href="#">kmeans.kmeansClustering</a>	Class implementing k-Means Clustering uses the pyclustering k-means implementation centers can be initialised using the k++ or the random initialiser . . . . .	57
<a href="#">kmedians.kmediansClustering</a>	Implements k-Medians Clustering uses the pyclustering k-medians implementation centers are initialised using the random initialiser . . . . .	60
<a href="#">kmedoids.kmedoidsClustering</a>	Implements k-Medians Clustering uses the scikit-learn-extra k-medoids implementation centers are set using the k++ initialiser if not set differently . . . . .	62
<a href="#">results.Results</a>	Class for easily saving and loading already calculated clustering results  every dataset has a folder containing subfolders for every clustering algorithm containing more subfolders for every distance measure . . . . .	65
<a href="#">SessionState.SessionState</a>		69



## Chapter 5

# File Index

### 5.1 File List

Here is a list of all files with brief descriptions:

<a href="#">clustering.py</a>	
Clustering base class . . . . .	71
<a href="#">comparison_plots.py</a>	
Script for generating plots for comparing different parameters . . . . .	71
<a href="#">dbscan.py</a>	
Implementation of the DBSCAN algorithm . . . . .	73
<a href="#">dbscan_comparison_plots.py</a>	
Implementation of DBSCAN parameter estimation heuristic . . . . .	73
<a href="#">dbscan_heuristic.py</a>	
Implementation of DBSCAN parameter estimation heuristic . . . . .	73
<a href="#">dbscan_solutions.py</a>	
Webfrontend for the DBSCAN heuristic implemented using streamlit . . . . .	74
<a href="#">heuristic_web.py</a>	
Evaluation Modul to compare clustering results . . . . .	75
<a href="#">indices.py</a>	
Implementation of the k-means algorithm . . . . .	76
<a href="#">kmeans.py</a>	
Implementation of the k-medians algorithm . . . . .	76
<a href="#">kmedians.py</a>	
Implementation of the k-medoids algorithm . . . . .	77
<a href="#">kmedoids.py</a>	
Script for generating latex tables for timing results . . . . .	77
<a href="#">make_timing_table.py</a>	
Handler for saving and loading results . . . . .	78
<a href="#">result_calculation.py</a>	
<a href="#">results.py</a>	
Taken from <a href="https://gist.github.com/tvst/036da038ab3e999a64497f42de966a92">https://gist.github.com/tvst/036da038ab3e999a64497f42de966a92</a>	
79	
<a href="#">SessionState.py</a>	
Timing.py . . . . .	80
<a href="#">timing.py</a>	
Webfrontend for project . . . . .	80
<a href="#">web_frontend.py</a>	



## Chapter 6

# Namespace Documentation

### 6.1 clustering Namespace Reference

#### Classes

- class [Clustering](#)

*Base Class for all subsequent clustering algorithms  
implements all functions needed for running the different  
cluster algorithms.*

### 6.2 comparison\_plots Namespace Reference

#### Variables

- list [kalgos](#) = ['kmeans', 'kmedians', 'kmedoids']
- dictionary [kalgoclass](#) = {'kmeans': [kmeansClustering](#), 'kmedians': [kmediansClustering](#), 'kmedoids': [kmedoidsClustering](#)}
- list [distances](#) = ["euclidean", "manhattan", "chebyshev", "cosine"]
- list [datasets](#) = ["iris", "wine", "diabetes", "housevotes"]
- list [index\\_ext\\_eval](#) = ["ARI", "AMI", "Completeness Score", "Homogeneity Score"]
- list [index\\_int\\_eval](#) = ["Silhouette Score"]
- list [num\\_of\\_classes](#) = [3,3,2,2]
- int [seed](#) = 42
- [results](#) = [Results](#)("./code/results")
- [all\\_kalgos](#) = np.zeros((3,4, 9,len([index\\_ext\\_eval](#))))
- [all\\_kalgos\\_int](#) = pd.DataFrame(columns=['k', 'Distance (Silhouette Score)', 'Distance (Clustering)', 'sil\_score', 'kalgo'])
- [all\\_dist](#) = np.zeros((4, 9,len([index\\_ext\\_eval](#))))
- [all\\_k](#) = np.zeros((9,len([index\\_ext\\_eval](#))))
- [clusters](#)
- [stuff](#)
- [s](#)
- [c](#)
- [d](#)
- [k](#)
- dictionary [cluster](#) = [kalgoclass](#)[c]([d](#), [s](#), [seed](#))

- `clustered_data` = `np.zeros(len(cluster.data))`
- dictionary `labels` = `cluster.labels.tolist()`
- `predicted` = `clustered_data.tolist()`
- `l1` = `Indices(predicted, labels)`
- `index_scores` = `np.zeros_like(index_ext_eval, dtype=float)`
- `index_score` = `l1.index_internal(index=index_int_eval[0], points=cluster.data.tolist(), metric=di)`
- `index`
- `fig` = `plt.figure(figsize=(15, 10))`
- `bbox_to_anchor`
- `loc`
- `borderaxespad`
- `ax`
- `figsize`
- `data`
- `x`
- `y`
- `hue`
- `style`
- `legend`
- `all_kalgos_df` = `pd.DataFrame(all_kalgos[:, :, num_of_classes[isx]-2, i], columns=distances, index=kalgos)`
- `kind`
- `title`

## 6.2.1 Variable Documentation

### 6.2.1.1 `all_dist`

```
comparison_plots.all_dist = np.zeros((4, 9, len(index_ext_eval)))
```

### 6.2.1.2 `all_k`

```
comparison_plots.all_k = np.zeros((9, len(index_ext_eval)))
```

### 6.2.1.3 `all_kalgos`

```
comparison_plots.all_kalgos = np.zeros((3, 4, 9, len(index_ext_eval)))
```



#### 6.2.1.4 all\_kalgos\_df

```
comparison_plots.all_kalgos_df = pd.DataFrame(all_kalgos[:, :, num_of_classes[isx]-2, i], columns=distances,
index=kalgos)
```

#### 6.2.1.5 all\_kalgos\_int

```
comparison_plots.all_kalgos_int = pd.DataFrame(columns=['k', 'Distance (Silhouette Score)',
'Distance (Clustering)', 'sil_score', 'kalgo'])
```

#### 6.2.1.6 ax

```
comparison_plots.ax
```

#### 6.2.1.7 bbox\_to\_anchor

```
comparison_plots.bbox_to_anchor
```

#### 6.2.1.8 borderaxespad

```
comparison_plots.borderaxespad
```

#### 6.2.1.9 c

```
comparison_plots.c
```

#### 6.2.1.10 cluster

```
dictionary comparison_plots.cluster = kalgoclass[c](d, s, seed)
```

#### 6.2.1.11 clustered\_data

```
comparison_plots.clustered_data = np.zeros(len(cluster.data))
```

#### 6.2.1.12 clusters

```
comparison_plots.clusters
```

#### 6.2.1.13 d

```
comparison_plots.d
```

#### 6.2.1.14 data

```
comparison_plots.data
```

#### 6.2.1.15 datasets

```
list comparison_plots.datasets = ["iris", "wine", "diabetes", "housevotes"]
```

#### 6.2.1.16 distances

```
comparison_plots.distances = ["euclidean", "manhattan", "chebyshev", "cosine"]
```

#### 6.2.1.17 fig

```
comparison_plots.fig = plt.figure(figsize=(15, 10))
```

#### 6.2.1.18 figsize

```
comparison_plots.figsize
```

#### 6.2.1.19 hue

```
comparison_plots.hue
```

#### 6.2.1.20 I1

```
comparison_plots.I1 = Indices(predicted, labels)
```

#### 6.2.1.21 index

```
comparison_plots.index
```

#### 6.2.1.22 index\_ext\_eval

```
list comparison_plots.index_ext_eval = ["ARI", "AMI", "Completeness Score", "Homogeneity Score"]
```

#### 6.2.1.23 index\_int\_eval

```
list comparison_plots.index_int_eval = ["Silhouette Score"]
```

#### 6.2.1.24 index\_score

```
comparison_plots.index_score = I1.index_internal(index=index_int_eval[0], points=cluster.↵  
data.tolist(), metric=di)
```

#### 6.2.1.25 index\_scores

```
comparison_plots.index_scores = np.zeros_like(index_ext_eval, dtype=float)
```

#### 6.2.1.26 k

`comparison_plots.k`

#### 6.2.1.27 kalgoclass

```
dictionary comparison_plots.kalgoclass = {'kmeans': kmeansClustering, 'kmedians': kmediansClustering,  
'kmedoids': kmedoidsClustering}
```

#### 6.2.1.28 kalgos

```
list comparison_plots.kalgos = ['kmeans', 'kmedians', 'kmedoids']
```

#### 6.2.1.29 kind

`comparison_plots.kind`

#### 6.2.1.30 labels

```
dictionary comparison_plots.labels = cluster.labels.tolist()
```

#### 6.2.1.31 legend

`comparison_plots.legend`

#### 6.2.1.32 loc

`comparison_plots.loc`

#### 6.2.1.33 num\_of\_classes

```
list comparison_plots.num_of_classes = [3,3,2,2]
```

#### 6.2.1.34 predicted

```
comparison_plots.predicted = clustered_data.tolist()
```

#### 6.2.1.35 results

```
comparison_plots.results = Results("./code/results")
```

#### 6.2.1.36 s

```
comparison_plots.s
```

#### 6.2.1.37 seed

```
int comparison_plots.seed = 42
```

#### 6.2.1.38 stuff

```
comparison_plots.stuff
```

#### 6.2.1.39 style

```
comparison_plots.style
```

#### 6.2.1.40 title

```
comparison_plots.title
```

#### 6.2.1.41 x

`comparison_plots.x`

#### 6.2.1.42 y

`comparison_plots.y`

## 6.3 dbscan Namespace Reference

### Classes

- class [DBSCANClustering](#)  
*implements DBSCAN Clustering*  
*uses the scikit-learn DBSCAN implementation*

## 6.4 dbscan\_comparision\_plots Namespace Reference

### Functions

- def [plot\\_kdist](#) ([kdists](#), dataset)  
*plots the sorted kdist graph using matplotlib*

### Variables

- list [distances](#) = ["euclidean", "manhattan", "chebyshev", "cosine"]
- list [datasets](#) = ["iris", "wine", "diabetes", "housevotes"]
- [heu](#) = [DBSCANHeuristic](#)()
- list [kdists](#) = []

### 6.4.1 Function Documentation

#### 6.4.1.1 plot\_kdist()

```
def dbscan_comparision_plots.plot_kdist (
    kdists,
    dataset )
```

plots the sorted kdist graph using matplotlib

## Parameters

<i>k-dist</i>	list containing the k-distances for every point of the dataset
---------------	--

## 6.4.2 Variable Documentation

### 6.4.2.1 datasets

```
list dbscan_comparision_plots.datasets = ["iris", "wine", "diabetes", "housevotes"]
```

### 6.4.2.2 distances

```
list dbscan_comparision_plots.distances = ["euclidean", "manhattan", "chebyshev", "cosine"]
```

### 6.4.2.3 heu

```
dbscan_comparision_plots.heu = DBSCANHeuristic()
```

### 6.4.2.4 kdists

```
list dbscan_comparision_plots.kdists = []
```

## 6.5 dbscan\_heuristic Namespace Reference

### Classes

- class [DBSCANHeuristic](#)  
*implements the DBSCAN heuristic proposed in the original DBSCAN paper:*

## 6.6 dbscan\_solutions Namespace Reference

### Functions

- def [save\\_score](#) (key, [score](#), m, e)

## Variables

- list `distances` = ["euclidean", "manhattan", "chebyshev", "cosine"]
- list `datasets` = ["iris", "wine", "diabetes", "housevotes"]
- int `seed` = 42
- list `external` = ["ARI", "AMI", "Completeness Score", "Homogeneity Score"]
- list `internal` = ["Silhouette Score"]
- dictionary `best_results` = {}
- `alg` = `DBSCANClustering`(d, s, `seed`)
- `clusters`
- `centers`
- `clustered_data` = `np.zeros`(len(`alg.data`))
- `indices` = `Indices`(`clustered_data.tolist()`, `alg.labels.tolist()`)
- `score` = `indices.index_external`(ind)

## 6.6.1 Function Documentation

### 6.6.1.1 `save_score()`

```
def dbscan_solutions.save_score (
    key,
    score,
    m,
    e )
```

## 6.6.2 Variable Documentation

### 6.6.2.1 `alg`

```
dbscan_solutions.alg = DBSCANClustering(d, s, seed)
```

### 6.6.2.2 `best_results`

```
dictionary dbscan_solutions.best_results = {}
```

### 6.6.2.3 `centers`

```
dbscan_solutions.centers
```



#### 6.6.2.4 clustered\_data

```
dbscan_solutions.clustered_data = np.zeros(len(alg.data))
```

#### 6.6.2.5 clusters

```
dbscan_solutions.clusters
```

#### 6.6.2.6 datasets

```
list dbscan_solutions.datasets = ["iris", "wine", "diabetes", "housevotes"]
```

#### 6.6.2.7 distances

```
list dbscan_solutions.distances = ["euclidean", "manhattan", "chebyshev", "cosine"]
```

#### 6.6.2.8 external

```
list dbscan_solutions.external = ["ARI", "AMI", "Completeness Score", "Homogeneity Score"]
```

#### 6.6.2.9 indices

```
dbscan_solutions.indices = Indices(clustered_data.tolist(), alg.labels.tolist())
```

#### 6.6.2.10 internal

```
list dbscan_solutions.internal = ["Silhouette Score"]
```

#### 6.6.2.11 score

```
dbscan_solutions.score = indices.index_external(ind)
```

### 6.6.2.12 seed

```
int dbscan_solutions.seed = 42
```

## 6.7 heuristic\_web Namespace Reference

### Variables

- [page\\_title](#)
- [page\\_icon](#)
- [key](#)
- [col1](#)
- [col2](#)
- [dataset](#) = col1.selectbox('Choose a beautiful dataset',['iris', 'wine', 'diabetes', 'housevotes'])
- dictionary [cluster\\_dist\\_desc](#)
- [cluster\\_dist](#) = col1.selectbox('Choose an awesome distance measure',list(cluster\_dist\_desc.keys()))
- [k](#) = col2.slider("Choose a nice value for k", min\_value=1, max\_value=20, step=1, value=4)
- [submit\\_button](#) = st.form\_submit\_button(label='Calculate [kdist](#) Graph')
- [heu](#) = DBSCANHeuristic()
- [kdist](#) = heu.kdist([k](#))
- [reverse](#)
- [df](#)
- [nearest](#) = alt.selection(type='single', nearest=True, on='mouseover', fields=[[points](#)], empty='none')
- [yaxis](#) = alt.Y("dist", axis=alt.Axis(title=f"{[k](#)}-dist"))
- [line](#)
- [selectors](#) = alt.Chart([df](#)).mark\_point().encode(x=[points](#), opacity=alt.value(0)).add\_selection([nearest](#))
- [points](#) = line.mark\_point(color="red").encode(opacity=alt.condition([nearest](#), alt.value(1), alt.value(0)))
- [text](#) = line.mark\_text(aligned='left', dx=5, dy=-5, color="red").encode(text=alt.condition([nearest](#), "label:N", alt.value(' '))).transform\_calculate(label=f"distance: " + format(datum.dist, ".2f"))
- [textp](#) = line.mark\_text(aligned='left', dx=5, dy=-15, color="red").encode([text](#)=alt.condition([nearest](#), "label:N", alt.value(' '))).transform\_calculate(label=f"format( (1 - (datum.points-1) / {len([kdist](#))} ) \* 100, ".2f") + "% core [points](#)")
- [rules](#) = alt.Chart([df](#)).mark\_rule(color='gray').encode(x="points").transform\_filter([nearest](#))
- [use\\_container\\_width](#)

### 6.7.1 Variable Documentation

#### 6.7.1.1 cluster\_dist

```
heuristic_web.cluster_dist = col1.selectbox('Choose an awesome distance measure',list(cluster↵
_dist_desc.keys()))
```

### 6.7.1.2 cluster\_dist\_desc

dictionary heuristic\_web.cluster\_dist\_desc

#### Initial value:

```
1 = {'euclidean': 'd(x,y) = \sqrt{\sum_{i=1}^n (|x_i-y_i|)^2}',
2   'manhattan': 'd(x,y) = \sum\limits_{i=1}^n |x_i - y_i|',
3   'chebyshev': 'd(x,y) = \max(|x_i - y_i|)',
4   'cosine': 'd(x,y) = \frac{\arccos(\frac{\sum_{i=1}^n x_i y_i}{\sqrt{\sum_{i=1}^n x_i^2} \sqrt{\sum_{i=1}^n y_i^2}})}{\pi}'}
```

### 6.7.1.3 col1

heuristic\_web.col1

### 6.7.1.4 col2

heuristic\_web.col2

### 6.7.1.5 dataset

```
heuristic_web.dataset = col1.selectbox('Choose a beautiful dataset', ['iris', 'wine', 'diabetes',
'housevotes'])
```

### 6.7.1.6 df

heuristic\_web.df

#### Initial value:

```
1 = pd.DataFrame(
2     [[i+1, kdist[i]] for i in range(len(kdist))],
3     columns=["points", "dist"])
```

### 6.7.1.7 heu

heuristic\_web.heu = DBSCANHeuristic()

#### 6.7.1.8 k

```
heuristic_web.k = col2.slider("Choose a nice value for k", min_value=1, max_value=20, step=1, value=4)
```

#### 6.7.1.9 kdist

```
heuristic_web.kdist = heu.kdist(k)
```

#### 6.7.1.10 key

```
heuristic_web.key
```

#### 6.7.1.11 line

```
heuristic_web.line
```

##### Initial value:

```
1 = alt.Chart(df).mark_line().encode(x="points", y=yaxis).properties({
2     title=f"DBSCAN Heuristic k={k}, {cluster_dist} distance")
```

#### 6.7.1.12 nearest

```
heuristic_web.nearest = alt.selection(type='single', nearest=True, on='mouseover', fields=['points'], empty='none')
```

#### 6.7.1.13 page\_icon

```
heuristic_web.page_icon
```

#### 6.7.1.14 page\_title

```
heuristic_web.page_title
```

#### 6.7.1.15 points

```
heuristic_web.points = line.mark_point(color="red").encode(opacity=alt.condition(nearest,
alt.value(1), alt.value(0)))
```

#### 6.7.1.16 reverse

```
heuristic_web.reverse
```

#### 6.7.1.17 rules

```
heuristic_web.rules = alt.Chart(df).mark_rule(color='gray').encode(x="points").transform_↵
filter(nearest)
```

#### 6.7.1.18 selectors

```
heuristic_web.selectors = alt.Chart(df).mark_point().encode(x='points', opacity=alt.value(0)).add↵
_selection(nearest)
```

#### 6.7.1.19 submit\_button

```
heuristic_web.submit_button = st.form_submit_button(label='Calculate kdist Graph')
```

#### 6.7.1.20 text

```
heuristic_web.text = line.mark_text(align='left', dx=5, dy=-5, color="red").encode(text=alt.↵
condition(nearest, "label:N", alt.value(' '))).transform_calculate(label=f"distance: " +
format(datum.dist, ".2f"))
```

#### 6.7.1.21 textp

```
heuristic_web.textp = line.mark_text(align='left', dx=5, dy=-15, color="red").encode(text=alt.↵
condition(nearest, "label:N", alt.value(' '))).transform_calculate(label=f'format( (1 - (datum.↵
points-1) / {len(kdist)}) * 100, ".2f") + "% core points"')
```

#### 6.7.1.22 use\_container\_width

```
heuristic_web.use_container_width
```

#### 6.7.1.23 yaxis

```
heuristic_web.yaxis = alt.Y("dist", axis=alt.Axis(title=f"{k}-dist"))
```

## 6.8 indices Namespace Reference

### Classes

- class [Indices](#)  
*calculates [Indices](#) for computed cluster labels uses the scikit library*

## 6.9 kmeans Namespace Reference

### Classes

- class [kmeansClustering](#)  
*Class implementing k-Means Clustering  
uses the pyclustering k-means implementation  
centers can be initialised using the k++ or the random initialiser.*

## 6.10 kmedians Namespace Reference

### Classes

- class [kmediansClustering](#)  
*implements k-Medians Clustering uses the pyclustering k-medians implementation centers are initialised using the random initialiser*

## 6.11 kmedoids Namespace Reference

### Classes

- class [kmedoidsClustering](#)  
*implements k-Medians Clustering  
uses the scikit-learn-extra k-medoids implementation  
centers are set using the k++ initialiser if not set differently*

## 6.12 make\_timing\_table Namespace Reference

### Variables

- list `kalgos` = ['kmeans', 'kmedians', 'kmedoids']
- dictionary `kalgoclass` = {'kmeans': `kmeansClustering`, 'kmedians': `kmediansClustering`, 'kmedoids': `kmedoidsClustering`}
- list `distances` = ["euclidean", "manhattan", "chebyshev", "cosine"]
- list `datasets` = ["iris", "wine", "diabetes", "housevotes"]
- dictionary `dataset_cluster` = {"iris": 3, "wine" : 3, "diabetes" : 2, "housevotes" : 2}
- dictionary `dbscan_comb`
- `timing_results` = json.load(f)
- `table_array` = np.zeros((4,4))
- `timing_table` = pd.DataFrame(`table_array`, columns=["Euclidean", "Manhattan", "Chebyshev", "Cosine"], index=["K-Means", 'K-Medians', 'K-Medoids']+['DBSCAN'])
- `axis`

### 6.12.1 Variable Documentation

#### 6.12.1.1 `axis`

```
make_timing_table.axis
```

#### 6.12.1.2 `dataset_cluster`

```
dictionary make_timing_table.dataset_cluster = {"iris": 3, "wine" : 3, "diabetes" : 2,
"housevotes" : 2}
```

#### 6.12.1.3 `datasets`

```
list make_timing_table.datasets = ["iris", "wine", "diabetes", "housevotes"]
```

#### 6.12.1.4 `dbscan_comb`

```
dictionary make_timing_table.dbscan_comb
```

##### Initial value:

```
1 = {"euclideaniris" : {"minpts" : 3, "eps" : 0.4}, "euclideanwine" : {"minpts" : 18, "eps" : 2.4},
2 "euclidean diabetes" : {"minpts" : 3, "eps" : 0.1}, "euclideanhousevotes" : {"minpts" : 18, "eps" : 2.5},
3
4 "manhattaniris" : {"minpts" : 1, "eps" : 1.2}, "manhattanwine" : {"minpts" : 18, "eps" : 6.9},
5 "manhattandiabetes" : {"minpts" : 2, "eps" : 0.3}, "manhattanhousevotes" : {"minpts" : 18, "eps" : 6.0},
6
7 "chebyshevis" : {"minpts" : 1, "eps" : 0.7}, "chebyshev wine" : {"minpts" : 17, "eps" : 1.3},
8 "chebyshev diabetes" : {"minpts" : 1, "eps" : 0.1}, "chebyshevhousevotes" : {"minpts" : 4, "eps" : 0.1},
9
10 "cosineiris" : {"minpts" : 1, "eps" : 0.1}, "cosinewine" : {"minpts" : 16, "eps" : 0.3},
11 "cosinediabetes" : {"minpts" : 10, "eps" : 0.2}, "cosinehousevotes" : {"minpts" : 18, "eps" : 0.2},
12 }
```

### 6.12.1.5 distances

```
list make_timing_table.distances = ["euclidean", "manhattan", "chebyshev", "cosine"]
```

### 6.12.1.6 kalgoclass

```
dictionary make_timing_table.kalgoclass = {'kmeans': kmeansClustering, 'kmedians': kmediansClustering,
'kmedoids': kmedoidsClustering}
```

### 6.12.1.7 kalgos

```
list make_timing_table.kalgos = ['kmeans', 'kmedians', 'kmedoids']
```

### 6.12.1.8 table\_array

```
make_timing_table.table_array = np.zeros((4,4))
```

### 6.12.1.9 timing\_results

```
make_timing_table.timing_results = json.load(f)
```

### 6.12.1.10 timing\_table

```
make_timing_table.timing_table = pd.DataFrame(table_array, columns=["Euclidean", "Manhattan",
"Chebyshev", "Cosine"], index=['K-Means', 'K-Medians', 'K-Medoids']+['DBSCAN'])
```

## 6.13 result\_calculation Namespace Reference

### Variables

- list `kalgos` = ['kmeans', 'kmedians', 'kmedoids']
- dictionary `kalgoclass` = {'kmeans': `kmeansClustering`, 'kmedians': `kmediansClustering`, 'kmedoids': `kmedoidsClustering`}
- list `distances` = ["euclidean", "manhattan", "chebyshev", "cosine"]
- list `datasets` = ["iris", "wine", "diabetes", "housevotes"]
- int `seed` = 42
- `results` = `Results`("./code/results")
- `s`
- `c`
- `d`
- `k`
- dictionary `alg` = `kalgoclass`[`c`](`d`, `s`, `seed`)
- `clusters`
- `centers`
- `minpts`
- `m`
- `eps`



## 6.13.1 Variable Documentation

### 6.13.1.1 alg

```
result_calculation.alg = kalgoclass[c](d, s, seed)
```

### 6.13.1.2 c

```
result_calculation.c
```

### 6.13.1.3 centers

```
result_calculation.centers
```

### 6.13.1.4 clusters

```
result_calculation.clusters
```

### 6.13.1.5 d

```
result_calculation.d
```

### 6.13.1.6 datasets

```
list result_calculation.datasets = ["iris", "wine", "diabetes", "housevotes"]
```

### 6.13.1.7 distances

```
list result_calculation.distances = ["euclidean", "manhattan", "chebyshev", "cosine"]
```

#### 6.13.1.8 eps

```
result_calculation.eps
```

#### 6.13.1.9 k

```
result_calculation.k
```

#### 6.13.1.10 kalgoclass

```
dictionary result_calculation.kalgoclass = {'kmeans': kmeansClustering, 'kmedians': kmediansClustering,  
'kmedoids': kmedoidsClustering}
```

#### 6.13.1.11 kalgos

```
list result_calculation.kalgos = ['kmeans', 'kmedians', 'kmedoids']
```

#### 6.13.1.12 m

```
result_calculation.m
```

#### 6.13.1.13 minpts

```
result_calculation.minpts
```

#### 6.13.1.14 results

```
result_calculation.results = Results("./code/results")
```

#### 6.13.1.15 s

```
result_calculation.s
```

#### 6.13.1.16 seed

```
int result_calculation.seed = 42
```

## 6.14 results Namespace Reference

### Classes

- class [Results](#)

*class for easily saving and loading already calculated clustering results*

*every dataset has a folder containing subfolders for every clustering algorithm containing more subfolders for every distance measure.*

## 6.15 SessionState Namespace Reference

### Classes

- class [SessionState](#)

### Functions

- def [get](#) (\*\*kwargs)

*Gets a [SessionState](#) object for the current session.*

#### 6.15.1 Function Documentation

### 6.15.1.1 `get()`

```
def SessionState.get (
    ** kwargs )
```

Gets a [SessionState](#) object for the current session.

Creates a new object if necessary.

Parameters

-----

```
**kwargs : any
    Default values you want to add to the session state, if we're creating a
    new one.
```

Example

-----

```
>>> session_state = get(user_name='', favorite_color='black')
>>> session_state.user_name
''
>>> session_state.user_name = 'Mary'
>>> session_state.favorite_color
'black'
```

Since you set `user_name` above, next time your script runs this will be the result:

```
>>> session_state = get(user_name='', favorite_color='black')
>>> session_state.user_name
'Mary'
```

## 6.16 timing Namespace Reference

### Variables

- list `kalgos` = ['kmeans', 'kmedians', 'kmedoids']
- dictionary `kalgoclass` = {'kmeans': [kmeansClustering](#), 'kmedians': [kmediansClustering](#), 'kmedoids': [kmedoidsClustering](#)}
- list `distances` = ["euclidean", "manhattan", "chebyshev", "cosine"]
- dictionary `dataset_cluster` = {"iris": 3, "wine": 3, "diabetes": 2, "housevotes": 2}
- dictionary `timing` = {}
- int `n` = 10
- int `time` = 0
- dictionary `alg` = `kalgoclass[c](d, s)`
- `before` = `datetime.now()`
- `clusters`
- `centers`
- `after` = `datetime.now()`
- dictionary `dbscan_comb`

### 6.16.1 Variable Documentation

### 6.16.1.1 after

```
timing.after = datetime.now()
```

### 6.16.1.2 alg

```
timing.alg = kalgoclass[c](d, s)
```

### 6.16.1.3 before

```
timing.before = datetime.now()
```

### 6.16.1.4 centers

```
timing.centers
```

### 6.16.1.5 clusters

```
timing.clusters
```

### 6.16.1.6 dataset\_cluster

```
dictionary timing.dataset_cluster = {"iris": 3, "wine": 3, "diabetes": 2, "housevotes": 2}
```

### 6.16.1.7 dbscan\_comb

```
dictionary timing.dbscan_comb
```

#### Initial value:

```
1 = {"euclideaniris": {"minpts": 3, "eps": 0.4}, "euclideanwine": {"minpts": 18, "eps": 2.4},
2 "euclideanidiabetes": {"minpts": 3, "eps": 0.1}, "euclideanhousevotes": {"minpts": 18, "eps": 2.5},
3
4 "manhattaniris": {"minpts": 1, "eps": 1.2}, "manhattanwine": {"minpts": 18, "eps": 6.9},
5 "manhattandiabetes": {"minpts": 2, "eps": 0.3}, "manhattanhousevotes": {"minpts": 18, "eps": 6.0},
6
7 "chebysheviris": {"minpts": 1, "eps": 0.7}, "chebyshevwine": {"minpts": 17, "eps": 1.3},
8 "chebyshevidiabetes": {"minpts": 1, "eps": 0.1}, "chebyshevhousevotes": {"minpts": 4, "eps": 0.1},
9
10 "cosineiris": {"minpts": 1, "eps": 0.1}, "cosinewine": {"minpts": 16, "eps": 0.3},
11 "cosinediabetes": {"minpts": 10, "eps": 0.2}, "cosinehousevotes": {"minpts": 18, "eps": 0.2},
12 }
```

#### 6.16.1.8 distances

```
list timing.distances = ["euclidean", "manhattan", "chebyshev", "cosine"]
```

#### 6.16.1.9 kalgoclass

```
dictionary timing.kalgoclass = {'kmeans': kmeansClustering, 'kmedians': kmediansClustering,  
'kmedoids': kmedoidsClustering}
```

#### 6.16.1.10 kalgos

```
list timing.kalgos = ['kmeans', 'kmedians', 'kmedoids']
```

#### 6.16.1.11 n

```
int timing.n = 10
```

#### 6.16.1.12 time

```
int timing.time = 0
```

#### 6.16.1.13 timing

```
dictionary timing.timing = {}
```

## 6.17 web\_frontend Namespace Reference

### Functions

- def [create\\_cluster](#) ([cluster\\_algo](#), [cluster\\_dist](#), [dataset](#), [seed](#))  
*creates a cluster algorithm instance.*
- def [clustering](#) ([cluster](#), [params](#), [cluster\\_algo](#))  
*calculates clustering results.*
- def [plotting](#) ()  
*generates the plots for the frontend.*

## Variables

- `page_title`
- `page_icon`
- `session_state = SessionState.get(indices_data=pd.DataFrame())`
- `seeded = st.checkbox('Use precalculated results (with random seed for reproduction).', value=True)`
- `seed = None`
- `seaplots = st.checkbox('Use interactive charts', value=True)`
- `resulthandler = Results("./code/results")`
- `col1`
- `col2`
- `dataset = col1.selectbox('Choose a beautiful dataset', ['iris', 'wine', 'diabetes', 'housevotes'])`
- dictionary `cluster_dist_desc`
- `cluster_dist = col1.selectbox('Choose an awesome distance measure', list(cluster_dist_desc.keys()))`
- dictionary `cluster_algo_class = {'kmeans': kmeansClustering, 'kmedians': kmediansClustering, 'kmedoids': kmedoidsClustering, 'DBSCAN': DBSCANClustering}`
- `cluster_algo = col2.selectbox('Choose a lovely clustering algorithm', list(cluster_algo_class.keys()))`
- dictionary `params = {}`
- `epsilon = col2.slider("Choose a nice value for epsilon", min_value=0.1, max_value=20.0, step=0.1)`
- `minpts = col2.slider("Choose a minimal number of nearest points", min_value=1, max_value=20, step=1, value=5)`
- `k_value = col2.slider("Choose a nice value for k (number of clusters)", min_value=2, max_value=10, step=1, value=3)`
- `def cluster = create_cluster(cluster_algo, cluster_dist, dataset, seed)`
- `datasetinformation = st.beta_expander("dataset information")`
- `perp = col1.slider("Perplexity for t-SNE", 5, 50, 25)`
- `dfclusterdata = pd.DataFrame()`
- `fig1`
- `fig2`
- `use_container_width`
- `clusterset = set(clustered_data)`
- `dataexpander = st.beta_expander("data")`
- `add_result = col1.button('Add')`
- `reset_tmp = col2.button('Reset')`
- `indices_data`
- `string val = "epsilon="+str(epsilon)+" , np="+str(minpts)`
- `df = session_state.indices_data`
- `def labels = cluster.labels.tolist()`
- `predicted = clustered_data.tolist()`
- `list precalc = []`
- `list index_eval = ["ARI", "AMI", "Completeness Score", "Homogeneity Score", "Silhouette Score"]`
- `l1 = Indices(predicted, labels)`
- `score = l1.index_external(index_eval[i])`
- `list datasets = []`
- `list results = [[1, "maximum reference value"]]`
- `list desc_list = []`
- `desc = np.array(desc_list)`
- `stats = np.zeros(len(results))`
- `df_for = pd.DataFrame(results, columns=["Score", "Data"])`
- `data_select = alt.selection_multi(fields=["Data"], name="Datapoint")`
- `string title = "Index:" + " " + index_eval + " , " + " " + "Dataset:" + " " + datasets[0]`
- `base`

## 6.17.1 Function Documentation

### 6.17.1.1 clustering()

```
def web_frontend.clustering (
    cluster,
    params,
    cluster_algo )
```

calculates clustering results.

uses the streamlit caching decorator

#### Parameters

<i>cluster</i>	cluster algorithm object
<i>params</i>	dictionary containing parameters needed for the cluster algorithm, either k or minpts and eps

#### Returns

cluster results, cluster centers, clustered data

### 6.17.1.2 create\_cluster()

```
def web_frontend.create_cluster (
    cluster_algo,
    cluster_dist,
    dataset,
    seed )
```

creates a cluster algorithm instance.

takes all parameters needed for creating such instance. uses the streamlit caching decorator

#### Parameters

<i>cluster_algo</i>	string containing name of cluster algorithm used ("kmeans", "kmedians", "kmedoids", "DBSCAN")
<i>cluster_dist</i>	string containing the distance measure used ("euclidean", "manhattan", "chebyshev", "cosine")
<i>dataset</i>	string containing the datasets name ("iris", "wine", "diabetes", "housevotes")
<i>seed</i>	seed for cluster algorithm, None if a random seed should be used

#### Returns

cluster results, cluster centers, clustered data



### 6.17.1.3 plotting()

```
def web_frontend.plotting ( )
```

generates the plots for the frontend.

uses the streamlit chacheing decorator

#### Returns

TSNE and PCA projections of clustering results either as seaborn or altair plots

## 6.17.2 Variable Documentation

### 6.17.2.1 add\_result

```
web_frontend.add_result = coll.button('Add')
```

### 6.17.2.2 base

```
web_frontend.base
```

#### Initial value:

```
1 = alt.Chart(  
2     df_for, width=(len(results)*120), height=500).mark_bar().configure(  
3     lineBreak = ", "  
4 ).properties(  
5     title = title  
6 ).encode(  
7     x = alt.X("Data", axis=alt.Axis(labelAngle=0)),  
8     y = alt.Y("Score:Q"),  
9     tooltip = ("Data", "Score"),  
10    opacity=alt.condition(data_select, alt.value(1), alt.value(0.0)),  
11    color=alt.Color("Data", legend=None)  
12 ).add_selection(  
13     data_select  
14 ).configure_view(  
15     strokeOpacity=0  
16 ).interactive()
```

### 6.17.2.3 cluster

```
def web_frontend.cluster = create_cluster(cluster_algo, cluster_dist, dataset, seed)
```

#### 6.17.2.4 cluster\_algo

```
web_frontend.cluster_algo = col2.selectbox('Choose a lovely clustering algorithm', list(cluster←
_algo_class.keys()))
```

#### 6.17.2.5 cluster\_algo\_class

```
dictionary web_frontend.cluster_algo_class = {'kmeans': kmeansClustering, 'kmedians': kmediansClustering,
'kmedoids': kmedoidsClustering, 'DBSCAN': DBSCANClustering}
```

#### 6.17.2.6 cluster\_dist

```
web_frontend.cluster_dist = col1.selectbox('Choose an awesome distance measure', list(cluster←
_dist_desc.keys()))
```

#### 6.17.2.7 cluster\_dist\_desc

```
dictionary web_frontend.cluster_dist_desc
```

##### Initial value:

```
1 = {'euclidean': 'd(x,y) = \sqrt{\sum_{i=1}^n (|x_i-y_i|)^2}',
2      'manhattan': 'd(x,y) = \sum\limits_{i=1}^n |x_i - y_i|',
3      'chebyshev': 'd(x,y) = \max(|x_i - y_i|)',
4      'cosine': 'd(x,y) = \frac{\arccos(\frac{\sum_{i=1}^n x_i
y_i}{\sqrt{\sum_{i=1}^n x_i^2 \sum_{i=1}^n y_i^2}})}{\pi}'}
```

#### 6.17.2.8 clusterset

```
web_frontend.clusterset = set(clustered_data)
```

#### 6.17.2.9 col1

```
web_frontend.col1
```

#### 6.17.2.10 col2

```
web_frontend.col2
```

#### 6.17.2.11 data\_select

```
web_frontend.data_select = alt.selection_multi(fields=["Data"], name="Datapoint")
```

#### 6.17.2.12 dataexpander

```
web_frontend.dataexpander = st.beta_expander("data")
```

#### 6.17.2.13 dataset

```
web_frontend.dataset = coll.selectbox('Choose a beautiful dataset',['iris', 'wine', 'diabetes',  
'housevotes'])
```

#### 6.17.2.14 datasetinformation

```
web_frontend.datasetinformation = st.beta_expander("dataset information")
```

#### 6.17.2.15 datasets

```
list web_frontend.datasets = []
```

#### 6.17.2.16 desc

```
web_frontend.desc = np.array(desc_list)
```

#### 6.17.2.17 desc\_list

```
list web_frontend.desc_list = []
```

**6.17.2.18 df**

```
web_frontend.df = session_state.indices_data
```

**6.17.2.19 df\_for**

```
web_frontend.df_for = pd.DataFrame(results, columns=["Score", "Data"])
```

**6.17.2.20 dfclusterdata**

```
web_frontend.dfclusterdata = pd.DataFrame()
```

**6.17.2.21 epsilon**

```
web_frontend.epsilon = col2.slider("Choose a nice value for epsilon", min_value=0.1, max_value=20.0, step=0.1)
```

**6.17.2.22 fig1**

```
web_frontend.fig1
```

**6.17.2.23 fig2**

```
web_frontend.fig2
```

**6.17.2.24 l1**

```
web_frontend.l1 = Indices(predicted, labels)
```

#### 6.17.2.25 index\_eval

```
web_frontend.index_eval = ["ARI", "AMI", "Completeness Score", "Homogeneity Score", "Silhouette Score"]
```

#### 6.17.2.26 indices\_data

```
web_frontend.indices_data
```

#### 6.17.2.27 k\_value

```
web_frontend.k_value = col2.slider("Choose a nice value for k (number of clusters)", min_value=2, max_value=10, step=1, value=3)
```

#### 6.17.2.28 labels

```
def web_frontend.labels = cluster.labels.tolist()
```

#### 6.17.2.29 minpts

```
web_frontend.minpts = col2.slider("Choose a minimal number of nearest points", min_value=1, max_value=20, step=1, value=5)
```

#### 6.17.2.30 page\_icon

```
web_frontend.page_icon
```

#### 6.17.2.31 page\_title

```
web_frontend.page_title
```

#### 6.17.2.32 params

```
dictionary web_frontend.params = {}
```

#### 6.17.2.33 perp

```
web_frontend.perp = coll.slider("Perplexity for t-SNE", 5, 50, 25)
```

#### 6.17.2.34 precalc

```
list web_frontend.precalc = []
```

#### 6.17.2.35 predicted

```
web_frontend.predicted = clustered_data.tolist()
```

#### 6.17.2.36 reset\_tmp

```
web_frontend.reset_tmp = col2.button('Reset')
```

#### 6.17.2.37 resulthandler

```
web_frontend.resulthandler = Results("./code/results")
```

#### 6.17.2.38 results

```
list web_frontend.results = [[1, "maximum reference value"]]
```

#### 6.17.2.39 score

```
web_frontend.score = I1.index_external(index_eval[i])
```

#### 6.17.2.40 seaplots

```
web_frontend.seaplots = st.checkbox('Use interactive charts', value=True)
```

#### 6.17.2.41 seed

```
int web_frontend.seed = None
```

#### 6.17.2.42 seeded

```
web_frontend.seeded = st.checkbox('Use precalculated results (with random seed for reproduction).',  
value=True)
```

#### 6.17.2.43 session\_state

```
web_frontend.session_state = SessionState.get(indices_data=pd.DataFrame())
```

#### 6.17.2.44 stats

```
web_frontend.stats = np.zeros(len(results))
```

#### 6.17.2.45 title

```
string web_frontend.title = "Index:" + " " + index_eval + "," + " " + "Dataset:" + " " + datasets[0]
```

#### 6.17.2.46 use\_container\_width

```
web_frontend.use_container_width
```

#### 6.17.2.47 val

```
string web_frontend.val = "epsilon="+str(epsilon)+", np="+str(minpts)
```





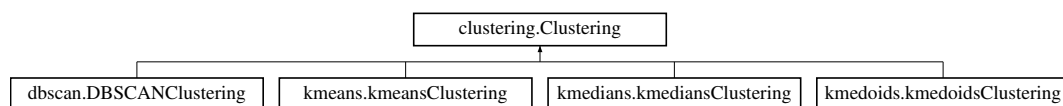
## Chapter 7

# Class Documentation

### 7.1 clustering.Clustering Class Reference

Base Class for all subsequent clustering algorithms  
implements all functions needed for running the different  
cluster algorithms.

Inheritance diagram for clustering.Clustering:



#### Public Member Functions

- `def __init__ (self, metric, dataset, seed=None)`  
*constructor*
- `def pyc_metric (self, metric)`  
*returns a distance metric which is usable by the pyclustering algorithms*
- `def load_data (self)`  
*loads in a dataset, standardises it and sets it as self.data attribute*
- `def house_load (self, path, skip=1)`  
*loads the housevotes dataset and encodes it using One-Hot-Encoding  
democrats are labeled as 1, republicans as 0*
- `def cluster (self)`  
*does nothing in the meta class.*

## Public Attributes

- [metric](#)  
*metric name as string or pyclustering distance\_metric object*
- [dataset](#)  
*dataset name as string*
- [data](#)  
*data that gets clustered*
- [labels](#)  
*expected cluster values*
- [seed](#)  
*seed for initializer, None if no seed is used*
- [datadf](#)  
*dataset as pandas frame.*

### 7.1.1 Detailed Description

Base Class for all subsequent clustering algorithms  
implements all functions needed for running the different  
cluster algorithms.

### 7.1.2 Constructor & Destructor Documentation

#### 7.1.2.1 `__init__()`

```
def clustering.Clustering.__init__ (
    self,
    metric,
    dataset,
    seed = None )
```

constructor

#### Parameters

<i>metric</i>	metric description as string. allowed: "euclidean", "manhattan", "chebyshev", "cosine"
<i>dataset</i>	dataset given as string. allowed: "diabetes", "iris", "wine", "housevotes"

Reimplemented in [kmedoids.kmedoidsClustering](#), [kmedians.kmediansClustering](#), [kmeans.kmeansClustering](#), and [dbscan.DBSCANClustering](#).

### 7.1.3 Member Function Documentation

### 7.1.3.1 cluster()

```
def clustering.Clustering.cluster (
    self )
```

does nothing in the meta class.

needs to be implemented in the inheriting cluster algorithm classes

### 7.1.3.2 house\_load()

```
def clustering.Clustering.house_load (
    self,
    path,
    skip = 1 )
```

loads the housevotes dataset and encodes it using One-Hot-Encoding  
democrats are labeled as 1, republicans as 0

#### Parameters

<i>path</i>	filepath to the dataset
<i>skip</i>	number of lines that get skipped when reading in a file

#### Returns

One-Hot-Encoded housevotes dataset and labels as array of 1s and 0s

### 7.1.3.3 load\_data()

```
def clustering.Clustering.load_data (
    self )
```

loads in a dataset, standardises it and sets it as self.data attribute

### 7.1.3.4 pyc\_metric()

```
def clustering.Clustering.pyc_metric (
    self,
    metric )
```

returns a distance metric which is usable by the pyclustering algorithms

#### Parameters

<i>distance</i>	metric string. allowed: "euclidean", "manhattan", "chebyshev", "cosine"
-----------------	---

#### Returns

pyclustering distance\_metric object, None when distance is not supported

### 7.1.4 Member Data Documentation

#### 7.1.4.1 data

`clustering.Clustering.data`

data that gets clustered

#### 7.1.4.2 datadf

`clustering.Clustering.datadf`

dataset as pandas frame.

needed for web frontend later

#### 7.1.4.3 dataset

`clustering.Clustering.dataset`

dataset name as string

#### 7.1.4.4 labels

`clustering.Clustering.labels`

expected cluster values

#### 7.1.4.5 metric

`clustering.Clustering.metric`

metric name as string or pyclustering distance\_metric object

#### 7.1.4.6 seed

`clustering.Clustering.seed`

seed for initializer, None if no seed is used

The documentation for this class was generated from the following file:

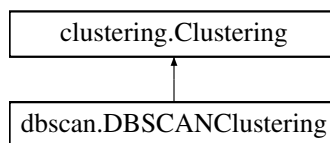
- [clustering.py](#)

## 7.2 dbscan.DBSCANClustering Class Reference

implements DBSCAN Clustering

uses the scikit-learn DBSCAN implementation

Inheritance diagram for `dbscan.DBSCANClustering`:



### Public Member Functions

- `def __init__(self, metric, dataset, seed=None)`  
*constructor, seed can be given but is not used.*
- `def cluster(self, eps, minpts)`  
*clustering method.*
- `def package(self, labels)`  
*rearranges the result to a format similar to the one of the pyclustering algorithms  
allows for easier access in the streamlit interface*

### Public Attributes

- `metric`  
*metric name as string*
- `dataset`  
*dataset name as string*
- `data`  
*data that gets clustered*
- `labels`  
*expected cluster values*

## 7.2.1 Detailed Description

implements DBSCAN Clustering  
uses the scikit-learn DBSCAN implementation

## 7.2.2 Constructor & Destructor Documentation

### 7.2.2.1 `__init__()`

```
def dbscan.DBSCANClustering.__init__ (
    self,
    metric,
    dataset,
    seed = None )
```

constructor, seed can be given but is not used.

its passing is allowed to simplify the code in the web frontend

#### Parameters

<i>metric</i>	metric description as string. allowed: "euclidean", "manhattan", "chebyshev", "cosine"
<i>dataset</i>	dataset given as string. allowed: "diabetes", "iris", "wine", "housevotes"

Reimplemented from [clustering.Clustering](#).

## 7.2.3 Member Function Documentation

### 7.2.3.1 `cluster()`

```
def dbscan.DBSCANClustering.cluster (
    self,
    eps,
    minpts )
```

clustering method.

Will execute clustering on the data saved in self.data with the metric given in self.metric  
params are the same as in the DBSCAN paper

#### Parameters

<i>eps</i>	Distance for the Eps-Neighbourhood
<i>minPts</i>	Minmal number of points in a cluster

**Returns**

formatted clustered data

**7.2.3.2 package()**

```
def dbscan.DBSCANClustering.package (
    self,
    labels )
```

rearranges the result to a format similar to the one of the pyclustering algorithms  
allows for easier access in the streamlit interface

**Parameters**

<i>labels</i>	cluster labels DBSCAN assigns to a point
---------------	--

**Returns**

clusters as list of lists of indices of points and noise as list of indices of points

**7.2.4 Member Data Documentation****7.2.4.1 data**

```
dbscan.DBSCANClustering.data
```

data that gets clustered

**7.2.4.2 dataset**

```
dbscan.DBSCANClustering.dataset
```

dataset name as string

**7.2.4.3 labels**

```
dbscan.DBSCANClustering.labels
```

expected cluster values

#### 7.2.4.4 metric

`dbscan.DBSCANClustering.metric`

metric name as string

The documentation for this class was generated from the following file:

- [dbscan.py](#)

### 7.3 dbscan\_heuristic.DBSCANHeuristic Class Reference

implements the DBSCAN heuristic proposed in the original DBSCAN paper:

#### Public Member Functions

- `def \_\_init\_\_ (self)`  
*constructor.*
- `def set\_metric (self, metric)`  
*setter for the metric*
- `def set\_dataset (self, dataset)`  
*sets and loads the dataset using the DBSCANClustering objects `load_data()` function*
- `def kdist (self, k)`  
*calculates all  $k$ -distances for the dataset*
- `def plot\_kdist (self, kdist)`  
*plots the sorted  $kdist$  graph using `matplotlib`*

#### Public Attributes

- [clustering](#)  
*DBSCANClustering object for loading datasets.*
- [metric](#)  
*metric as string ("euclidean", "cosine", "manhattan", "chebyshev")*
- [k](#)  
*variable  $k$  used in the  $k$ -dist calculation*

#### 7.3.1 Detailed Description

implements the DBSCAN heuristic proposed in the original DBSCAN paper:

Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In Proceedings of the Second International Conference on Knowledge Discovery and Data Mining, KDD'96, page 226–231. AAAI Press, 1996.

#### 7.3.2 Constructor & Destructor Documentation



### 7.3.2.1 `__init__()`

```
def dbscan_heuristic.DBSCANHeuristic.__init__ (
    self )
```

constructor.

uses a DBSCANClustering object for loading the datasets

## 7.3.3 Member Function Documentation

### 7.3.3.1 `kdist()`

```
def dbscan_heuristic.DBSCANHeuristic.kdist (
    self,
    k )
```

calculates all k-distances for the dataset

Parameters

<i>k</i>	variable for the k-dist. Natural Number.
----------	--

### 7.3.3.2 `plot_kdist()`

```
def dbscan_heuristic.DBSCANHeuristic.plot_kdist (
    self,
    kdist )
```

plots the sorted kdist graph using matplotlib

Parameters

<i>k-dist</i>	list containing the k-distances for every point of the dataset
---------------	--

### 7.3.3.3 `set_dataset()`

```
def dbscan_heuristic.DBSCANHeuristic.set_dataset (
    self,
    dataset )
```

sets and loads the dataset using the DBSCANClustering objects `load_data()` function

**Parameters**

<i>dataset</i>	string with name of the dataset used ("iris", "wine", "diabetes", "housevotes")
----------------	---

**7.3.3.4 set\_metric()**

```
def dbscan_heuristic.DBSCANHeuristic.set_metric (
    self,
    metric )
```

setter for the metric

**Parameters**

<i>metric</i>	string containing name of the metric ("euclidean", "cosine", "manhattan", "chebyshev")
---------------	--

**7.3.4 Member Data Documentation****7.3.4.1 clustering**

```
dbscan_heuristic.DBSCANHeuristic.clustering
```

DBSCANClustering object for loading datasets.

**7.3.4.2 k**

```
dbscan_heuristic.DBSCANHeuristic.k
```

variable k used in the k-dist calculation

**7.3.4.3 metric**

```
dbscan_heuristic.DBSCANHeuristic.metric
```

metric as string ("euclidean", "cosine", "manhattan", "chebyshev")

The documentation for this class was generated from the following file:

- [dbscan\\_heuristic.py](#)

## 7.4 indices.Indices Class Reference

calculates [Indices](#) for computed cluster labels uses the scikit library

### Public Member Functions

- `def __init__ (self, cluster\_calc, cluster\_label)`  
*constructor*
- `def index\_external (self, index)`  
*Function to calculate external index scores*  
*ARI, AMI, Homogeneity Score and Completeness Score @params index string with name of index used ("ARI", "AMI", "Homogeneity Score", "Completeness Score")*
- `def index\_internal (self, index, points, metric)`  
*Function to calculate internal index scores*

### Public Attributes

- [cluster\\_calc](#)  
*calculated clustering results*
- [cluster\\_label](#)  
*expected cluster results*

### 7.4.1 Detailed Description

calculates [Indices](#) for computed cluster labels uses the scikit library

### 7.4.2 Constructor & Destructor Documentation

#### 7.4.2.1 \_\_init\_\_()

```
def indices.Indices.__init__ (
    self,
    cluster_calc,
    cluster_label )
```

constructor

Parameters

<i>cluster_calc</i>	calculated clustering results
<i>cluster_label</i>	expected cluster results

## 7.4.3 Member Function Documentation

### 7.4.3.1 `index_external()`

```
def indices.Indices.index_external (
    self,
    index )
```

Function to calculate external index scores

ARI, AMI, Homogeneity Score and Completeness Score @params index string with name of index used ("ARI", "AMI", "Homogeneity Score", "Completeness Score")

### 7.4.3.2 `index_internal()`

```
def indices.Indices.index_internal (
    self,
    index,
    points,
    metric )
```

Function to calculate internal index scores

#### Parameters

<i>index</i>	string with name of index used ("Silhouette Score")
<i>points</i>	data points of the selected dataset
<i>metric</i>	metric used for calculation of silhouette score

## 7.4.4 Member Data Documentation

### 7.4.4.1 `cluster_calc`

```
indices.Indices.cluster_calc
```

calculated clustering results

#### 7.4.4.2 cluster\_label

`indices.Indices.cluster_label`

expected cluster results

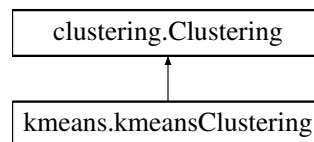
The documentation for this class was generated from the following file:

- [indices.py](#)

## 7.5 kmeans.kmeansClustering Class Reference

Class implementing k-Means Clustering  
uses the pyclustering k-means implementation  
centers can be initialised using the k++ or the random initialiser.

Inheritance diagram for kmeans.kmeansClustering:



### Public Member Functions

- `def __init__(self, metric, dataset, seed=None)`  
*constructor*
- `def cluster(self, k, plusplus=True)`  
*clustering method.*

### Public Attributes

- `metric`  
*metric name as pyclustering distance\_metric object*
- `dataset`  
*dataset name as string*
- `data`  
*data that gets clustered*
- `labels`  
*expected cluster values*
- `seed`  
*seed for initializer, None if no seed is used*

### 7.5.1 Detailed Description

Class implementing k-Means Clustering  
uses the pyclustering k-means implementation  
centers can be initialised using the k++ or the random initialiser.

## 7.5.2 Constructor & Destructor Documentation

### 7.5.2.1 `__init__()`

```
def kmeans.kmeansClustering.__init__ (
    self,
    metric,
    dataset,
    seed = None )
```

constructor

#### Parameters

<i>metric</i>	metric description as string. allowed: "euclidean", "manhattan", "chebyshev", "cosine"
<i>dataset</i>	dataset given as string. allowed: "diabetes", "iris", "wine", "housevotes"

Reimplemented from [clustering.Clustering](#).

## 7.5.3 Member Function Documentation

### 7.5.3.1 `cluster()`

```
def kmeans.kmeansClustering.cluster (
    self,
    k,
    plusplus = True )
```

clustering method.

Will execute clustering on the data saved in self.data with the metric given in self.metric

#### Parameters

<i>k</i>	number of clusters that are generated
<i>plusplus</i>	will use k++ initialiser if true

#### Returns

clusters as list of lists of indices of points and final cluster centers

## 7.5.4 Member Data Documentation

#### 7.5.4.1 data

`kmeans.kmeansClustering.data`

data that gets clustered

#### 7.5.4.2 dataset

`kmeans.kmeansClustering.dataset`

dataset name as string

#### 7.5.4.3 labels

`kmeans.kmeansClustering.labels`

expected cluster values

#### 7.5.4.4 metric

`kmeans.kmeansClustering.metric`

metric name as pyclustering distance\_metric object

#### 7.5.4.5 seed

`kmeans.kmeansClustering.seed`

seed for initializer, None if no seed is used

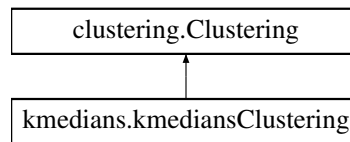
The documentation for this class was generated from the following file:

- [kmeans.py](#)

## 7.6 kmedians.kmediansClustering Class Reference

implements k-Medians Clustering uses the pyclustering k-medians implementation centers are initialised using the random initialiser

Inheritance diagram for kmedians.kmediansClustering:



### Public Member Functions

- `def __init__ (self, metric, dataset, seed=None)`  
*constructor*
- `def cluster (self, k, plusplus=True)`  
*clustering method.*

### Public Attributes

- [metric](#)  
*metric name as pyclustering distance\_metric object*
- [dataset](#)  
*dataset name as string*
- [data](#)  
*data that gets clustered*
- [labels](#)  
*expected cluster values*
- [seed](#)  
*seed for initializer, None if no seed is used*

### 7.6.1 Detailed Description

implements k-Medians Clustering uses the pyclustering k-medians implementation centers are initialised using the random initialiser

### 7.6.2 Constructor & Destructor Documentation

#### 7.6.2.1 \_\_init\_\_()

```

def kmedians.kmediansClustering.__init__ (
    self,
    metric,
    dataset,
    seed = None )
  
```

constructor



## Parameters

<i>metric</i>	metric description as string. allowed: "euclidean", "manhattan", "chebyshev", "cosine"
<i>dataset</i>	dataset given as string. allowed: "diabetes", "iris", "wine", "housevotes"

Reimplemented from [clustering.Clustering](#).

## 7.6.3 Member Function Documentation

### 7.6.3.1 cluster()

```
def kmedians.kmediansClustering.cluster (
    self,
    k,
    plusplus = True )
```

clustering method.

Will execute clustering on the data saved in self.data with the metric given in self.metric

## Parameters

<i>k</i>	number of clusters that are generated
----------	---------------------------------------

## Returns

clusters as list of lists of indices of points and final cluster medians

## 7.6.4 Member Data Documentation

### 7.6.4.1 data

```
kmedians.kmediansClustering.data
```

data that gets clustered

### 7.6.4.2 dataset

```
kmedians.kmediansClustering.dataset
```

dataset name as string

#### 7.6.4.3 labels

`kmedians.kmediansClustering.labels`

expected cluster values

#### 7.6.4.4 metric

`kmedians.kmediansClustering.metric`

metric name as pyclustering distance\_metric object

#### 7.6.4.5 seed

`kmedians.kmediansClustering.seed`

seed for initializer, None if no seed is used

The documentation for this class was generated from the following file:

- [kmedians.py](#)

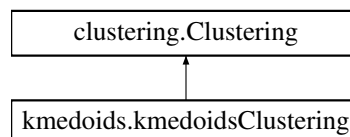
## 7.7 kmedoids.kmedoidsClustering Class Reference

implements k-Medians Clustering

uses the scikit-learn-extra k-medoids implementation

centers are set using the k++ initialiser if not set differently

Inheritance diagram for `kmedoids.kmedoidsClustering`:



### Public Member Functions

- `def __init__ (self, metric, dataset, seed=None)`  
*constructor*
- `def cluster (self, k, init="k-medoids++")`  
*clustering method.*
- `def package (self, labels)`  
*rearranges the result to a format similar to the one of the pyclustering algorithms allows for easier access in the streamlit interface*

## Public Attributes

- [metric](#)  
*metric name as string*
- [dataset](#)  
*dataset name as string*
- [data](#)  
*data that gets clustered*
- [labels](#)  
*expected cluster values*
- [seed](#)  
*seed for initializer, None if no seed is used*

### 7.7.1 Detailed Description

implements k-Medians Clustering  
uses the scikit-learn-extra k-medoids implementation  
centers are set using the k++ initialiser if not set differently

### 7.7.2 Constructor & Destructor Documentation

#### 7.7.2.1 `__init__()`

```
def kmedoids.kmedoidsClustering.__init__ (
    self,
    metric,
    dataset,
    seed = None )
```

constructor

Parameters

<i>metric</i>	metric description as string. allowed: "euclidean", "manhattan", "chebyshev", "cosine"
<i>dataset</i>	dataset given as string. allowed: "diabetes", "iris", "wine", "housevotes"

Reimplemented from [clustering.Clustering](#).

### 7.7.3 Member Function Documentation

### 7.7.3.1 cluster()

```
def kmedoids.kmedoidsClustering.cluster (
    self,
    k,
    init = "k-medoids++" )
```

clustering method.

Will execute clustering on the data saved in self.data with the metric given in self.metric

#### Parameters

<i>k</i>	number of clusters that are generated
<i>init</i>	initialisation parameter. Default: "k-medoids++"

#### Returns

clusters as list of lists of indices of points, final cluster centers

### 7.7.3.2 package()

```
def kmedoids.kmedoidsClustering.package (
    self,
    labels )
```

rearranges the result to a format similar to the one of the pyclustering algorithms allows for easier access in the streamlit interface

#### Parameters

<i>labels</i>	labels returned from the KMedoids algorithm
---------------	---

#### Returns

clusters formatted similarly to the pyclustering algorithms

## 7.7.4 Member Data Documentation

### 7.7.4.1 data

```
kmedoids.kmedoidsClustering.data
```

data that gets clustered

#### 7.7.4.2 dataset

`kmedoids.kmedoidsClustering.dataset`

dataset name as string

#### 7.7.4.3 labels

`kmedoids.kmedoidsClustering.labels`

expected cluster values

#### 7.7.4.4 metric

`kmedoids.kmedoidsClustering.metric`

metric name as string

#### 7.7.4.5 seed

`kmedoids.kmedoidsClustering.seed`

seed for initializer, None if no seed is used

The documentation for this class was generated from the following file:

- [kmedoids.py](#)

## 7.8 results.Results Class Reference

class for easily saving and loading already calculated clustering results

every dataset has a folder containing subfolders for every clustering algorithm containing more subfolders for every distance measure.

## Public Member Functions

- `def __init__ (self, parentpath)`  
*constructor.*
- `def get_path (self, dataset, algorithm, metric, **kwargs)`  
*builds and returns the filepath to the json file fitting the given parameters*
- `def set_exists (self, dataset, algorithm, metric, **kwargs)`  
*checks if a file for a result defined by the parameters exists*
- `def load_set (self, dataset, algorithm, metric, **kwargs)`  
*loads results fitting the given parameters from a json file*
- `def save_set (self, dataset, algorithm, metric, clusters, centers, **kwargs)`  
*saves cluster results in a json file*

## Public Attributes

- `parent`

### 7.8.1 Detailed Description

class for easily saving and loading already calculated clustering results

every dataset has a folder containing subfolders for every clustering algorithm containing more subfolders for every distance measure.

Clustering results are saved as json files in their respective folders.

### 7.8.2 Constructor & Destructor Documentation

#### 7.8.2.1 \_\_init\_\_()

```
def results.Results.__init__ (
    self,
    parentpath )
```

constructor.

needs the filepath to the parent directory where the json files are supposed to be saved

Parameters

<code>parentpath</code>	filepath to the parent directory
-------------------------	----------------------------------

### 7.8.3 Member Function Documentation

### 7.8.3.1 get\_path()

```
def results.Results.get_path (
    self,
    dataset,
    algorithm,
    metric,
    ** kwargs )
```

builds and returns the filepath to the json file fitting the given parameters

#### Parameters

<i>dataset</i>	string with the name of the dataset ("iris", "wine", "diabetes", "DBSCAN")
<i>algorithm</i>	string with the name of the algorithm ("kmeans", "kmedians", "kmedoids", "DBSCAN")
<i>metric</i>	string with the name of the distance measure ("euclidean", "cosine", "chebyshev", "manhattan")
<i>**kwargs</i>	algorithm specific parameters. Needs to be either "k" or "minpts" and "eps"

#### Returns

filepath to the correct json file

### 7.8.3.2 load\_set()

```
def results.Results.load_set (
    self,
    dataset,
    algorithm,
    metric,
    ** kwargs )
```

loads results fitting the given parameters from a json file

#### Parameters

<i>dataset</i>	string with the name of the dataset ("iris", "wine", "diabetes", "DBSCAN")
<i>algorithm</i>	string with the name of the algorithm ("kmeans", "kmedians", "kmedoids", "DBSCAN")
<i>metric</i>	string with the name of the distance measure ("euclidean", "cosine", "chebyshev", "manhattan")
<i>**kwargs</i>	algorithm specific parameters. Needs to be either "k" or "minpts" and "eps"

#### Returns

loaded clustering results (clusters and centers)

### 7.8.3.3 save\_set()

```
def results.Results.save_set (
    self,
    dataset,
    algorithm,
    metric,
    clusters,
    centers,
    ** kwargs )
```

saves cluster results in a json file

#### Parameters

<i>dataset</i>	string with the name of the dataset ("iris", "wine", "diabetes", "DBSCAN")
<i>algorithm</i>	string with the name of the algorithm ("kmeans", "kmedians", "kmedoids", "DBSCAN")
<i>metric</i>	string with the name of the distance measure ("euclidean", "cosine", "chebyshev", "manhattan")
<i>**kwargs</i>	algorithm specific parameters. Needs to be either "k" or "minpts" and "eps"

### 7.8.3.4 set\_exists()

```
def results.Results.set_exists (
    self,
    dataset,
    algorithm,
    metric,
    ** kwargs )
```

checks if a file for a result defined by the parameters exists

#### Parameters

<i>dataset</i>	string with the name of the dataset ("iris", "wine", "diabetes", "DBSCAN")
<i>algorithm</i>	string with the name of the algorithm ("kmeans", "kmedians", "kmedoids", "DBSCAN")
<i>metric</i>	string with the name of the distance measure ("euclidean", "cosine", "chebyshev", "manhattan")
<i>**kwargs</i>	algorithm specific parameters. Needs to be either "k" or "minpts" and "eps"

#### Returns

True if file exists, False if not

## 7.8.4 Member Data Documentation



#### 7.8.4.1 parent

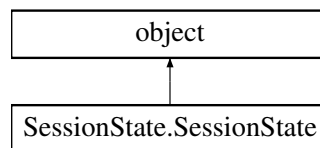
`results.Results.parent`

The documentation for this class was generated from the following file:

- [results.py](#)

## 7.9 SessionState.SessionState Class Reference

Inheritance diagram for SessionState.SessionState:



### Public Member Functions

- `def __init__(self, **kwargs)`  
A new [SessionState](#) object.

### 7.9.1 Constructor & Destructor Documentation

#### 7.9.1.1 \_\_init\_\_()

```
def SessionState.SessionState.__init__(
    self,
    **kwargs )
```

A new [SessionState](#) object.

```
Parameters
-----
**kwargs : any
    Default values for the session state.

Example
-----
>>> session_state = SessionState(user_name='', favorite_color='black')
>>> session_state.user_name = 'Mary'
''
>>> session_state.favorite_color
'black'
```

The documentation for this class was generated from the following file:

- [SessionState.py](#)



## Chapter 8

# File Documentation

### 8.1 clustering.py File Reference

contains the clustering base class

#### Classes

- class [clustering.Clustering](#)  
*Base Class for all subsequent clustering algorithms  
implements all functions needed for running the different  
cluster algorithms.*

#### Namespaces

- [clustering](#)

#### 8.1.1 Detailed Description

contains the clustering base class

### 8.2 comparison\_plots.py File Reference

script for generating plots for comparing different parameters

#### Namespaces

- [comparison\\_plots](#)

## Variables

- list `comparison_plots.kalgos` = ['kmeans', 'kmedians', 'kmedoids']
- dictionary `comparison_plots.kalgoclass` = {'kmeans': kmeansClustering, 'kmedians': kmediansClustering, 'kmedoids': kmedoidsClustering}
- list `comparison_plots.distances` = ["euclidean", "manhattan", "chebyshev", "cosine"]
- list `comparison_plots.datasets` = ["iris", "wine", "diabetes", "housevotes"]
- list `comparison_plots.index_ext_eval` = ["ARI", "AMI", "Completeness Score", "Homogeneity Score"]
- list `comparison_plots.index_int_eval` = ["Silhouette Score"]
- list `comparison_plots.num_of_classes` = [3,3,2,2]
- int `comparison_plots.seed` = 42
- `comparison_plots.results` = Results("./code/results")
- `comparison_plots.all_kalgos` = np.zeros((3,4, 9,len(index\_ext\_eval)))
- `comparison_plots.all_kalgos_int` = pd.DataFrame(columns=['k', 'Distance (Silhouette Score)', 'Distance (Clustering)', 'sil\_score', 'kalgo'])
- `comparison_plots.all_dist` = np.zeros((4, 9,len(index\_ext\_eval)))
- `comparison_plots.all_k` = np.zeros((9,len(index\_ext\_eval)))
- `comparison_plots.clusters`
- `comparison_plots.stuff`
- `comparison_plots.s`
- `comparison_plots.c`
- `comparison_plots.d`
- `comparison_plots.k`
- dictionary `comparison_plots.cluster` = kalgoclass[c](d, s, seed)
- `comparison_plots.clustered_data` = np.zeros(len(cluster.data))
- dictionary `comparison_plots.labels` = cluster.labels.tolist()
- `comparison_plots.predicted` = clustered\_data.tolist()
- `comparison_plots.l1` = Indices(predicted, labels)
- `comparison_plots.index_scores` = np.zeros\_like(index\_ext\_eval, dtype=float)
- `comparison_plots.index_score` = l1.index\_internal(index=index\_int\_eval[0], points=cluster.data.tolist(), metric=di)
- `comparison_plots.index`
- `comparison_plots.fig` = plt.figure(figsize=(15, 10))
- `comparison_plots.bbox_to_anchor`
- `comparison_plots.loc`
- `comparison_plots.borderaxespad`
- `comparison_plots.ax`
- `comparison_plots.figsize`
- `comparison_plots.data`
- `comparison_plots.x`
- `comparison_plots.y`
- `comparison_plots.hue`
- `comparison_plots.style`
- `comparison_plots.legend`
- `comparison_plots.all_kalgos_df` = pd.DataFrame(all\_kalgos[:, :, num\_of\_classes[isx]-2, i], columns=distances, index=kalgos)
- `comparison_plots.kind`
- `comparison_plots.title`

### 8.2.1 Detailed Description

script for generating plots for comparing different parameters

## 8.3 dbscan.py File Reference

implementation of the DBSCAN algorithm.

### Classes

- class [dbscan.DBSCANClustering](#)  
*implements DBSCAN Clustering*  
*uses the scikit-learn DBSCAN implementation*

### Namespaces

- [dbscan](#)

#### 8.3.1 Detailed Description

implementation of the DBSCAN algorithm.

## 8.4 dbscan\_comparision\_plots.py File Reference

### Namespaces

- [dbscan\\_comparision\\_plots](#)

### Functions

- def [dbscan\\_comparision\\_plots.plot\\_kdist](#) (kdists, dataset)  
*plots the sorted kdist graph using matplotlib*

### Variables

- list [dbscan\\_comparision\\_plots.distances](#) = ["euclidean", "manhattan", "chebyshev", "cosine"]
- list [dbscan\\_comparision\\_plots.datasets](#) = ["iris", "wine", "diabetes", "housevotes"]
- [dbscan\\_comparision\\_plots.heu](#) = DBSCANHeuristic()
- list [dbscan\\_comparision\\_plots.kdists](#) = []

## 8.5 dbscan\_heuristic.py File Reference

implementation of DBSCAN parameter estimation heuristic

### Classes

- class [dbscan\\_heuristic.DBSCANHeuristic](#)  
*implements the DBSCAN heuristic proposed in the original DBSCAN paper:*

## Namespaces

- [dbscan\\_heuristic](#)

### 8.5.1 Detailed Description

implementation of DBSCAN parameter estimation heuristic

## 8.6 dbscan\_solutions.py File Reference

### Namespaces

- [dbscan\\_solutions](#)

### Functions

- def [dbscan\\_solutions.save\\_score](#) (key, score, m, e)

### Variables

- list [dbscan\\_solutions.distances](#) = ["euclidean", "manhattan", "chebyshev", "cosine"]
- list [dbscan\\_solutions.datasets](#) = ["iris", "wine", "diabetes", "housevotes"]
- int [dbscan\\_solutions.seed](#) = 42
- list [dbscan\\_solutions.external](#) = ["ARI", "AMI", "Completeness Score", "Homogeneity Score"]
- list [dbscan\\_solutions.internal](#) = ["Silhouette Score"]
- dictionary [dbscan\\_solutions.best\\_results](#) = {}
- [dbscan\\_solutions.alg](#) = DBSCANClustering(d, s, seed)
- [dbscan\\_solutions.clusters](#)
- [dbscan\\_solutions.centers](#)
- [dbscan\\_solutions.clustered\\_data](#) = np.zeros(len(alg.data))
- [dbscan\\_solutions.indices](#) = Indices(clustered\_data.tolist(), alg.labels.tolist())
- [dbscan\\_solutions.score](#) = indices.index\_external(ind)

## 8.7 heuristic\_web.py File Reference

webfrontend for the DBSCAN heuristic implemented using streamlit.

### Namespaces

- [heuristic\\_web](#)

## Variables

- [heuristic\\_web.page\\_title](#)
- [heuristic\\_web.page\\_icon](#)
- [heuristic\\_web.key](#)
- [heuristic\\_web.col1](#)
- [heuristic\\_web.col2](#)
- [heuristic\\_web.dataset](#) = col1.selectbox('Choose a beautiful dataset',['iris', 'wine', 'diabetes', 'housevotes'])
- dictionary [heuristic\\_web.cluster\\_dist\\_desc](#)
- [heuristic\\_web.cluster\\_dist](#) = col1.selectbox('Choose an awesome distance measure',list(cluster\_dist\_desc.keys()))
- [heuristic\\_web.k](#) = col2.slider("Choose a nice value for k", min\_value=1, max\_value=20, step=1, value=4)
- [heuristic\\_web.submit\\_button](#) = st.form\_submit\_button(label='Calculate kdist Graph')
- [heuristic\\_web.heu](#) = DBSCANHeuristic()
- [heuristic\\_web.kdist](#) = heu.kdist(k)
- [heuristic\\_web.reverse](#)
- [heuristic\\_web.df](#)
- [heuristic\\_web.nearest](#) = alt.selection(type='single', nearest=True, on='mouseover', fields=['points'], empty='none')
- [heuristic\\_web.yaxis](#) = alt.Y("dist", axis=alt.Axis(title=f"{k}-dist"))
- [heuristic\\_web.line](#)
- [heuristic\\_web.selectors](#) = alt.Chart(df).mark\_point().encode(x='points', opacity=alt.value(0)).add\_selection(nearest)
- [heuristic\\_web.points](#) = line.mark\_point(color="red").encode(opacity=alt.condition(nearest, alt.value(1), alt.value(0)))
- [heuristic\\_web.text](#) = line.mark\_text(aligned='left', dx=5, dy=-5, color="red").encode(text=alt.condition(nearest, "label:N", alt.value(' '))).transform\_calculate(label=f"distance: " + format(datum.dist, ".2f"))
- [heuristic\\_web.textp](#) = line.mark\_text(aligned='left', dx=5, dy=-15, color="red").encode(text=alt.condition(nearest, "label:N", alt.value(' '))).transform\_calculate(label=f"format( ( 1 - (datum.points-1) / {len(kdist)}) \* 100, ".2f") + "% core points")
- [heuristic\\_web.rules](#) = alt.Chart(df).mark\_rule(color='gray').encode(x="points").transform\_filter(nearest)
- [heuristic\\_web.use\\_container\\_width](#)

### 8.7.1 Detailed Description

webfrontend for the DBSCAN heuristic implemented using streamlit.

uses altair charts for displaying the chart

## 8.8 indices.py File Reference

Evaluation Modul to compare clustering results.

## Classes

- class [indices.Indices](#)  
calculates [Indices](#) for computed cluster labels uses the scikit library

## Namespaces

- [indices](#)

### 8.8.1 Detailed Description

Evaluation Modul to compare clustering results.

## 8.9 kmeans.py File Reference

implementation of the k-means algorithm.

### Classes

- class [kmeans.kmeansClustering](#)  
*Class implementing k-Means Clustering  
uses the pyclustering k-means implementation  
centers can be initialised using the k++ or the random initialiser.*

## Namespaces

- [kmeans](#)

### 8.9.1 Detailed Description

implementation of the k-means algorithm.

## 8.10 kmedians.py File Reference

implementation of the k-medians algorithm.

### Classes

- class [kmedians.kmediansClustering](#)  
*implements k-Medians Clustering uses the pyclustering k-medians implementation centers are initialised using the random initialiser*

## Namespaces

- [kmedians](#)



### 8.10.1 Detailed Description

implementation of the k-medians algorithm.

## 8.11 kmedoids.py File Reference

implementation of the k-medoids algorithm.

### Classes

- class [kmedoids.kmedoidsClustering](#)  
*implements k-Medians Clustering*  
*uses the scikit-learn-extra k-medoids implementation*  
*centers are set using the k++ initialiser if not set differently*

### Namespaces

- [kmedoids](#)

### 8.11.1 Detailed Description

implementation of the k-medoids algorithm.

## 8.12 make\_timing\_table.py File Reference

script for generating latex tables for timing results

### Namespaces

- [make\\_timing\\_table](#)

### Variables

- list [make\\_timing\\_table.kalgos](#) = ['kmeans', 'kmedians', 'kmedoids']
- dictionary [make\\_timing\\_table.kalgoclass](#) = {'kmeans': kmeansClustering, 'kmedians': kmediansClustering, 'kmedoids': kmedoidsClustering}
- list [make\\_timing\\_table.distances](#) = ["euclidean", "manhattan", "chebyshev", "cosine"]
- list [make\\_timing\\_table.datasets](#) = ["iris", "wine", "diabetes", "housevotes"]
- dictionary [make\\_timing\\_table.dataset\\_cluster](#) = {"iris": 3, "wine" : 3, "diabetes" : 2, "housevotes" : 2}
- dictionary [make\\_timing\\_table.dbscan\\_comb](#)
- [make\\_timing\\_table.timing\\_results](#) = json.load(f)
- [make\\_timing\\_table.table\\_array](#) = np.zeros((4,4))
- [make\\_timing\\_table.timing\\_table](#) = pd.DataFrame(table\_array, columns=["Euclidean", "Manhattan", "Chebyshev", "Cosine"], index=['K-Means', 'K-Medians', 'K-Medoids']+['DBSCAN'])
- [make\\_timing\\_table.axis](#)

### 8.12.1 Detailed Description

script for generating latex tables for timing results

## 8.13 result\_calculation.py File Reference

### Namespaces

- [result\\_calculation](#)

### Variables

- list [result\\_calculation.kalgos](#) = ['kmeans', 'kmedians', 'kmedoids']
- dictionary [result\\_calculation.kalgoclass](#) = {'kmeans': kmeansClustering, 'kmedians': kmediansClustering, 'kmedoids': kmedoidsClustering}
- list [result\\_calculation.distances](#) = ["euclidean", "manhattan", "chebyshev", "cosine"]
- list [result\\_calculation.datasets](#) = ["iris", "wine", "diabetes", "housevotes"]
- int [result\\_calculation.seed](#) = 42
- [result\\_calculation.results](#) = Results("./code/results")
- [result\\_calculation.s](#)
- [result\\_calculation.c](#)
- [result\\_calculation.d](#)
- [result\\_calculation.k](#)
- dictionary [result\\_calculation.alg](#) = kalgoclass[c](d, s, seed)
- [result\\_calculation.clusters](#)
- [result\\_calculation.centers](#)
- [result\\_calculation.minpts](#)
- [result\\_calculation.m](#)
- [result\\_calculation.eps](#)

## 8.14 results.py File Reference

handler for saving and loading results.

### Classes

- class [results.Results](#)
  - class for easily saving and loading already calculated clustering results*
  - every dataset has a folder containing subfolders for every clustering algorithm containing more subfolders for every distance measure.*

### Namespaces

- [results](#)

### 8.14.1 Detailed Description

handler for saving and loading results.

## 8.15 SessionState.py File Reference

taken from <https://gist.github.com/tvst/036da038ab3e999a64497f42de966a92>.

### Classes

- class [SessionState.SessionState](#)

### Namespaces

- [SessionState](#)

### Functions

- def [SessionState.get](#) (\*\*kwargs)  
*Gets a [SessionState](#) object for the current session.*

### 8.15.1 Detailed Description

taken from <https://gist.github.com/tvst/036da038ab3e999a64497f42de966a92>.

Please refer to this gist and its original author

Hack to add per-session state to Streamlit.

#### 8.15.1.1 Usage

```
import SessionState
session_state = SessionState.get(user_name="", favorite_color='black') session_↵
_state.user_name
```

"

```
session_state.user_name = 'Mary' session_state.favorite_color
```

'black'

Since you set user\_name above, next time your script runs this will be the result:

```
session_state = get(user_name="", favorite_color='black') session_state.user_↵
name
```

'Mary'

## 8.16 timing.py File Reference

### Namespaces

- [timing](#)

### Variables

- list [timing.kalgos](#) = ['kmeans', 'kmedians', 'kmedoids']
- dictionary [timing.kalgoclass](#) = {'kmeans': kmeansClustering, 'kmedians': kmediansClustering, 'kmedoids': kmedoidsClustering}
- list [timing.distances](#) = ["euclidean", "manhattan", "chebyshev", "cosine"]
- dictionary [timing.dataset\\_cluster](#) = {"iris": 3, "wine": 3, "diabetes": 2, "housevotes": 2}
- dictionary [timing.timing](#) = {}
- int [timing.n](#) = 10
- int [timing.time](#) = 0
- dictionary [timing.alg](#) = kalgoclass[c](d, s)
- [timing.before](#) = datetime.now()
- [timing.clusters](#)
- [timing.centers](#)
- [timing.after](#) = datetime.now()
- dictionary [timing.dbscan\\_comb](#)

## 8.17 web\_frontend.py File Reference

webfrontend for project.

### Namespaces

- [web\\_frontend](#)

### Functions

- def [web\\_frontend.create\\_cluster](#) (cluster\_algo, cluster\_dist, dataset, seed)  
*creates a cluster algorithm instance.*
- def [web\\_frontend.clustering](#) (cluster, params, cluster\_algo)  
*calculates clustering results.*
- def [web\\_frontend.plotting](#) ()  
*generates the plots for the frontend.*

## Variables

- `web_frontend.page_title`
- `web_frontend.page_icon`
- `web_frontend.session_state = SessionState.get(indices_data=pd.DataFrame())`
- `web_frontend.seeded = st.checkbox('Use precalculated results (with random seed for reproduction).', value=True)`
- `web_frontend.seed = None`
- `web_frontend.seaplots = st.checkbox('Use interactive charts', value=True)`
- `web_frontend.resulthandler = Results("./code/results")`
- `web_frontend.col1`
- `web_frontend.col2`
- `web_frontend.dataset = col1.selectbox('Choose a beautiful dataset',['iris', 'wine', 'diabetes', 'housevotes'])`
- dictionary `web_frontend.cluster_dist_desc`
- `web_frontend.cluster_dist = col1.selectbox('Choose an awesome distance measure', list(cluster_dist_desc.keys()))`
- dictionary `web_frontend.cluster_algo_class = {'kmeans': kmeansClustering, 'kmedians': kmediansClustering, 'kmedoids': kmedoidsClustering, 'DBSCAN': DBSCANClustering}`
- `web_frontend.cluster_algo = col2.selectbox('Choose a lovely clustering algorithm', list(cluster_algo_class.keys()))`
- dictionary `web_frontend.params = {}`
- `web_frontend.epsilon = col2.slider("Choose a nice value for epsilon", min_value=0.1, max_value=20, step=0.1)`
- `web_frontend.minpts = col2.slider("Choose a minimal number of nearest points", min_value=1, max_value=20, step=1, value=5)`
- `web_frontend.k_value = col2.slider("Choose a nice value for k (number of clusters)", min_value=2, max_value=10, step=1, value=3)`
- `def web_frontend.cluster = create_cluster(cluster_algo, cluster_dist, dataset, seed)`
- `web_frontend.datasetinformation = st.beta_expander("dataset information")`
- `web_frontend.perp = col1.slider("Perplexity for t-SNE", 5, 50, 25)`
- `web_frontend.dfclusterdata = pd.DataFrame()`
- `web_frontend.fig1`
- `web_frontend.fig2`
- `web_frontend.use_container_width`
- `web_frontend.clusterset = set(clustered_data)`
- `web_frontend.dataexpander = st.beta_expander("data")`
- `web_frontend.add_result = col1.button('Add')`
- `web_frontend.reset_tmp = col2.button('Reset')`
- `web_frontend.indices_data`
- `string web_frontend.val = "epsilon="+str(epsilon)+" , np="+str(minpts)`
- `web_frontend.df = session_state.indices_data`
- `def web_frontend.labels = cluster.labels.tolist()`
- `web_frontend.predicted = clustered_data.tolist()`
- list `web_frontend.precalc = []`
- list `web_frontend.index_eval = ["ARI", "AMI", "Completeness Score", "Homogeneity Score", "Silhouette Score"]`
- `web_frontend.l1 = Indices(predicted, labels)`
- `web_frontend.score = l1.index_external(index_eval[i])`
- list `web_frontend.datasets = []`
- list `web_frontend.results = [[1, "maximum reference value"]]`
- list `web_frontend.desc_list = []`
- `web_frontend.desc = np.array(desc_list)`
- `web_frontend.stats = np.zeros(len(results))`
- `web_frontend.df_for = pd.DataFrame(results, columns=["Score","Data"])`
- `web_frontend.data_select = alt.selection_multi(fields=["Data"], name="Datapoint")`
- `string web_frontend.title = "Index:" + " " + index_eval + "," + " " + "Dataset:" + " " + datasets[0]`
- `web_frontend.base`

### 8.17.1 Detailed Description

webfrontend for project.

implemented using streamlit. displays parameter selection, clustering results, data and the evaluation module.  
Charts can be displayed using seaborn or altair

## 8.18 [/home/nordegraf/Uni/8.Semester/DataScienceI/datascience1\\_group42/README.md](#) File Reference

# Index

/home/nordegraf/Uni/8.Semester/DataScienceI/datasciencecenter/group42/README.md,

- [82](#)
- `__init__`
  - [clustering.Clustering, 46](#)
  - [dbscan.DBSCANClustering, 50](#)
  - [dbscan\\_heuristic.DBSCANHeuristic, 52](#)
  - [indices.Indices, 55](#)
  - [kmeans.kmeansClustering, 58](#)
  - [kmedians.kmediansClustering, 60](#)
  - [kmedoids.kmedoidsClustering, 63](#)
  - [results.Results, 66](#)
  - [SessionState.SessionState, 69](#)
- `add_result`
  - [web\\_frontend, 37](#)
- `after`
  - [timing, 32](#)
- `alg`
  - [dbscan\\_solutions, 20](#)
  - [result\\_calculation, 29](#)
  - [timing, 33](#)
- `all_dist`
  - [comparison\\_plots, 12](#)
- `all_k`
  - [comparison\\_plots, 12](#)
- `all_kalgos`
  - [comparison\\_plots, 12](#)
- `all_kalgos_df`
  - [comparison\\_plots, 12](#)
- `all_kalgos_int`
  - [comparison\\_plots, 13](#)
- `ax`
  - [comparison\\_plots, 13](#)
- `axis`
  - [make\\_timing\\_table, 27](#)
- `base`
  - [web\\_frontend, 37](#)
- `bbox_to_anchor`
  - [comparison\\_plots, 13](#)
- `before`
  - [timing, 33](#)
- `best_results`
  - [dbscan\\_solutions, 20](#)
- `borderaxespad`
  - [comparison\\_plots, 13](#)
- `c`
  - [comparison\\_plots, 13](#)
  - [result\\_calculation, 29](#)
- `center`
  - [dbscan\\_solutions, 20](#)
  - [result\\_calculation, 29](#)
  - [timing, 33](#)
- `cluster`
  - [clustering.Clustering, 46](#)
  - [comparison\\_plots, 13](#)
  - [dbscan.DBSCANClustering, 50](#)
  - [kmeans.kmeansClustering, 58](#)
  - [kmedians.kmediansClustering, 61](#)
  - [kmedoids.kmedoidsClustering, 63](#)
  - [web\\_frontend, 37](#)
- `cluster_algo`
  - [web\\_frontend, 37](#)
- `cluster_algo_class`
  - [web\\_frontend, 38](#)
- `cluster_calc`
  - [indices.Indices, 56](#)
- `cluster_dist`
  - [heuristic\\_web, 22](#)
  - [web\\_frontend, 38](#)
- `cluster_dist_desc`
  - [heuristic\\_web, 22](#)
  - [web\\_frontend, 38](#)
- `cluster_label`
  - [indices.Indices, 56](#)
- `clustered_data`
  - [comparison\\_plots, 13](#)
  - [dbscan\\_solutions, 20](#)
- `clustering, 11`
  - [dbscan\\_heuristic.DBSCANHeuristic, 54](#)
  - [web\\_frontend, 36](#)
- `clustering.Clustering, 45`
  - [\\_\\_init\\_\\_, 46](#)
  - [cluster, 46](#)
  - [data, 48](#)
  - [datadf, 48](#)
  - [dataset, 48](#)
  - [house\\_load, 47](#)
  - [labels, 48](#)
  - [load\\_data, 47](#)
  - [metric, 48](#)
  - [pyc\\_metric, 47](#)
  - [seed, 49](#)
- `clustering.py, 71`
- `clusters`
  - [comparison\\_plots, 14](#)
  - [dbscan\\_solutions, 21](#)
  - [result\\_calculation, 29](#)

- timing, 33
- clusterset
  - web\_frontend, 38
- col1
  - heuristic\_web, 23
  - web\_frontend, 38
- col2
  - heuristic\_web, 23
  - web\_frontend, 38
- comparison\_plots, 11
  - all\_dist, 12
  - all\_k, 12
  - all\_kalgos, 12
  - all\_kalgos\_df, 12
  - all\_kalgos\_int, 13
  - ax, 13
  - bbox\_to\_anchor, 13
  - borderaxespad, 13
  - c, 13
  - cluster, 13
  - clustered\_data, 13
  - clusters, 14
  - d, 14
  - data, 14
  - datasets, 14
  - distances, 14
  - fig, 14
  - figsize, 14
  - hue, 14
  - l1, 15
  - index, 15
  - index\_ext\_eval, 15
  - index\_int\_eval, 15
  - index\_score, 15
  - index\_scores, 15
  - k, 15
  - kalgoclass, 16
  - kalgos, 16
  - kind, 16
  - labels, 16
  - legend, 16
  - loc, 16
  - num\_of\_classes, 16
  - predicted, 17
  - results, 17
  - s, 17
  - seed, 17
  - stuff, 17
  - style, 17
  - title, 17
  - x, 17
  - y, 18
- comparison\_plots.py, 71
- create\_cluster
  - web\_frontend, 36
- d
  - comparison\_plots, 14
  - result\_calculation, 29
- data
  - clustering.Clustering, 48
  - comparison\_plots, 14
  - dbscan.DBSCANClustering, 51
  - kmeans.kmeansClustering, 58
  - kmedians.kmediansClustering, 61
  - kmedoids.kmedoidsClustering, 64
- data\_select
  - web\_frontend, 38
- datadf
  - clustering.Clustering, 48
- dataexpander
  - web\_frontend, 39
- dataset
  - clustering.Clustering, 48
  - dbscan.DBSCANClustering, 51
  - heuristic\_web, 23
  - kmeans.kmeansClustering, 59
  - kmedians.kmediansClustering, 61
  - kmedoids.kmedoidsClustering, 64
  - web\_frontend, 39
- dataset\_cluster
  - make\_timing\_table, 27
  - timing, 33
- datasetinformation
  - web\_frontend, 39
- datasets
  - comparison\_plots, 14
  - dbscan\_comparison\_plots, 19
  - dbscan\_solutions, 21
  - make\_timing\_table, 27
  - result\_calculation, 29
  - web\_frontend, 39
- dbscan, 18
- dbscan.DBSCANClustering, 49
  - \_\_init\_\_, 50
  - cluster, 50
  - data, 51
  - dataset, 51
  - labels, 51
  - metric, 51
  - package, 51
- dbscan.py, 73
- dbscan\_comb
  - make\_timing\_table, 27
  - timing, 33
- dbscan\_comparison\_plots, 18
  - datasets, 19
  - distances, 19
  - heu, 19
  - kdist, 19
  - plot\_kdist, 18
- dbscan\_comparison\_plots.py, 73
- dbscan\_heuristic, 19
- dbscan\_heuristic.DBSCANHeuristic, 52
  - \_\_init\_\_, 52
  - clustering, 54
  - k, 54



- kdist, [53](#)
  - metric, [54](#)
  - plot\_kdist, [53](#)
  - set\_dataset, [53](#)
  - set\_metric, [54](#)
- dbscan\_heuristic.py, [73](#)
- dbscan\_solutions, [19](#)
  - alg, [20](#)
  - best\_results, [20](#)
  - centers, [20](#)
  - clustered\_data, [20](#)
  - clusters, [21](#)
  - datasets, [21](#)
  - distances, [21](#)
  - external, [21](#)
  - indices, [21](#)
  - internal, [21](#)
  - save\_score, [20](#)
  - score, [21](#)
  - seed, [21](#)
- dbscan\_solutions.py, [74](#)
- desc
  - web\_frontend, [39](#)
- desc\_list
  - web\_frontend, [39](#)
- df
  - heuristic\_web, [23](#)
  - web\_frontend, [39](#)
- df\_for
  - web\_frontend, [40](#)
- dfclusterdata
  - web\_frontend, [40](#)
- distances
  - comparison\_plots, [14](#)
  - dbscan\_comparision\_plots, [19](#)
  - dbscan\_solutions, [21](#)
  - make\_timing\_table, [27](#)
  - result\_calculation, [29](#)
  - timing, [33](#)
- eps
  - result\_calculation, [29](#)
- epsilon
  - web\_frontend, [40](#)
- external
  - dbscan\_solutions, [21](#)
- fig
  - comparison\_plots, [14](#)
- fig1
  - web\_frontend, [40](#)
- fig2
  - web\_frontend, [40](#)
- figsize
  - comparison\_plots, [14](#)
- get
  - SessionState, [31](#)
- get\_path
  - results.Results, [66](#)
- heu
  - dbscan\_comparision\_plots, [19](#)
  - heuristic\_web, [23](#)
- heuristic\_web, [22](#)
  - cluster\_dist, [22](#)
  - cluster\_dist\_desc, [22](#)
  - col1, [23](#)
  - col2, [23](#)
  - dataset, [23](#)
  - df, [23](#)
  - heu, [23](#)
  - k, [23](#)
  - kdist, [24](#)
  - key, [24](#)
  - line, [24](#)
  - nearest, [24](#)
  - page\_icon, [24](#)
  - page\_title, [24](#)
  - points, [24](#)
  - reverse, [25](#)
  - rules, [25](#)
  - selectors, [25](#)
  - submit\_button, [25](#)
  - text, [25](#)
  - textp, [25](#)
  - use\_container\_width, [25](#)
  - yaxis, [26](#)
- heuristic\_web.py, [74](#)
- house\_load
  - clustering.Clustering, [47](#)
- hue
  - comparison\_plots, [14](#)
- l1
  - comparison\_plots, [15](#)
  - web\_frontend, [40](#)
- index
  - comparison\_plots, [15](#)
- index\_eval
  - web\_frontend, [40](#)
- index\_ext\_eval
  - comparison\_plots, [15](#)
- index\_external
  - indices.Indices, [56](#)
- index\_int\_eval
  - comparison\_plots, [15](#)
- index\_internal
  - indices.Indices, [56](#)
- index\_score
  - comparison\_plots, [15](#)
- index\_scores
  - comparison\_plots, [15](#)
- indices, [26](#)
  - dbscan\_solutions, [21](#)
- indices.Indices, [55](#)
  - \_\_init\_\_, [55](#)
  - cluster\_calc, [56](#)

- cluster\_label, 56
  - index\_external, 56
  - index\_internal, 56
- indices.py, 75
- indices\_data
  - web\_frontend, 41
- internal
  - dbscan\_solutions, 21
- k
  - comparison\_plots, 15
  - dbscan\_heuristic.DBSCANHeuristic, 54
  - heuristic\_web, 23
  - result\_calculation, 30
- k\_value
  - web\_frontend, 41
- kalgoclass
  - comparison\_plots, 16
  - make\_timing\_table, 28
  - result\_calculation, 30
  - timing, 34
- kalgos
  - comparison\_plots, 16
  - make\_timing\_table, 28
  - result\_calculation, 30
  - timing, 34
- kdist
  - dbscan\_heuristic.DBSCANHeuristic, 53
  - heuristic\_web, 24
- kdist
  - dbscan\_comparison\_plots, 19
- key
  - heuristic\_web, 24
- kind
  - comparison\_plots, 16
- kmeans, 26
- kmeans.kmeansClustering, 57
  - \_\_init\_\_, 58
  - cluster, 58
  - data, 58
  - dataset, 59
  - labels, 59
  - metric, 59
  - seed, 59
- kmeans.py, 76
- kmedians, 26
- kmedians.kmediansClustering, 60
  - \_\_init\_\_, 60
  - cluster, 61
  - data, 61
  - dataset, 61
  - labels, 61
  - metric, 62
  - seed, 62
- kmedians.py, 76
- kmedoids, 26
- kmedoids.kmedoidsClustering, 62
  - \_\_init\_\_, 63
  - cluster, 63
- data, 64
  - dataset, 64
  - labels, 65
  - metric, 65
  - package, 64
  - seed, 65
- kmedoids.py, 77
- labels
  - clustering.Clustering, 48
  - comparison\_plots, 16
  - dbscan.DBSCANClustering, 51
  - kmeans.kmeansClustering, 59
  - kmedians.kmediansClustering, 61
  - kmedoids.kmedoidsClustering, 65
  - web\_frontend, 41
- legend
  - comparison\_plots, 16
- line
  - heuristic\_web, 24
- load\_data
  - clustering.Clustering, 47
- load\_set
  - results.Results, 67
- loc
  - comparison\_plots, 16
- m
  - result\_calculation, 30
- make\_timing\_table, 27
  - axis, 27
  - dataset\_cluster, 27
  - datasets, 27
  - dbscan\_comb, 27
  - distances, 27
  - kalgoclass, 28
  - kalgos, 28
  - table\_array, 28
  - timing\_results, 28
  - timing\_table, 28
- make\_timing\_table.py, 77
- metric
  - clustering.Clustering, 48
  - dbscan.DBSCANClustering, 51
  - dbscan\_heuristic.DBSCANHeuristic, 54
  - kmeans.kmeansClustering, 59
  - kmedians.kmediansClustering, 62
  - kmedoids.kmedoidsClustering, 65
- minpts
  - result\_calculation, 30
  - web\_frontend, 41
- n
  - timing, 34
- nearest
  - heuristic\_web, 24
- num\_of\_classes
  - comparison\_plots, 16

- package
  - dbscan.DBSCANClustering, 51
  - kmedoids.kmedoidsClustering, 64
- page\_icon
  - heuristic\_web, 24
  - web\_frontend, 41
- page\_title
  - heuristic\_web, 24
  - web\_frontend, 41
- params
  - web\_frontend, 41
- parent
  - results.Results, 68
- perp
  - web\_frontend, 42
- plot\_kdist
  - dbscan\_comparison\_plots, 18
  - dbscan\_heuristic.DBSCANHeuristic, 53
- plotting
  - web\_frontend, 36
- points
  - heuristic\_web, 24
- precalc
  - web\_frontend, 42
- predicted
  - comparison\_plots, 17
  - web\_frontend, 42
- pyc\_metric
  - clustering.Clustering, 47
- reset\_tmp
  - web\_frontend, 42
- result\_calculation, 28
  - alg, 29
  - c, 29
  - centers, 29
  - clusters, 29
  - d, 29
  - datasets, 29
  - distances, 29
  - eps, 29
  - k, 30
  - kalgoclass, 30
  - kalgos, 30
  - m, 30
  - minpts, 30
  - results, 30
  - s, 30
  - seed, 31
- result\_calculation.py, 78
- resulthandler
  - web\_frontend, 42
- results, 31
  - comparison\_plots, 17
  - result\_calculation, 30
  - web\_frontend, 42
- results.py, 78
- results.Results, 65
  - \_\_init\_\_, 66
  - get\_path, 66
  - load\_set, 67
  - parent, 68
  - save\_set, 67
  - set\_exists, 68
- reverse
  - heuristic\_web, 25
- rules
  - heuristic\_web, 25
- s
  - comparison\_plots, 17
  - result\_calculation, 30
- save\_score
  - dbscan\_solutions, 20
- save\_set
  - results.Results, 67
- score
  - dbscan\_solutions, 21
  - web\_frontend, 42
- seaplots
  - web\_frontend, 42
- seed
  - clustering.Clustering, 49
  - comparison\_plots, 17
  - dbscan\_solutions, 21
  - kmeans.kmeansClustering, 59
  - kmedians.kmediansClustering, 62
  - kmedoids.kmedoidsClustering, 65
  - result\_calculation, 31
  - web\_frontend, 43
- seeded
  - web\_frontend, 43
- selectors
  - heuristic\_web, 25
- session\_state
  - web\_frontend, 43
- SessionState, 31
  - get, 31
- SessionState.py, 79
- SessionState.SessionState, 69
  - \_\_init\_\_, 69
- set\_dataset
  - dbscan\_heuristic.DBSCANHeuristic, 53
- set\_exists
  - results.Results, 68
- set\_metric
  - dbscan\_heuristic.DBSCANHeuristic, 54
- stats
  - web\_frontend, 43
- stuff
  - comparison\_plots, 17
- style
  - comparison\_plots, 17
- submit\_button
  - heuristic\_web, 25
- table\_array
  - make\_timing\_table, 28

- text
  - heuristic\_web, 25
- textp
  - heuristic\_web, 25
- time
  - timing, 34
- timing, 32
  - after, 32
  - alg, 33
  - before, 33
  - centers, 33
  - clusters, 33
  - dataset\_cluster, 33
  - dbscan\_comb, 33
  - distances, 33
  - kalgoclass, 34
  - kalgos, 34
  - n, 34
  - time, 34
  - timing, 34
- timing.py, 80
- timing\_results
  - make\_timing\_table, 28
- timing\_table
  - make\_timing\_table, 28
- title
  - comparison\_plots, 17
  - web\_frontend, 43
- use\_container\_width
  - heuristic\_web, 25
  - web\_frontend, 43
- val
  - web\_frontend, 43
- web\_frontend, 34
  - add\_result, 37
  - base, 37
  - cluster, 37
  - cluster\_algo, 37
  - cluster\_algo\_class, 38
  - cluster\_dist, 38
  - cluster\_dist\_desc, 38
  - clustering, 36
  - clusterset, 38
  - col1, 38
  - col2, 38
  - create\_cluster, 36
  - data\_select, 38
  - dataexpander, 39
  - dataset, 39
  - datasetinformation, 39
  - datasets, 39
  - desc, 39
  - desc\_list, 39
  - df, 39
  - df\_for, 40
  - dfclusterdata, 40
  - epsilon, 40
  - fig1, 40
  - fig2, 40
  - l1, 40
  - index\_eval, 40
  - indices\_data, 41
  - k\_value, 41
  - labels, 41
  - minpts, 41
  - page\_icon, 41
  - page\_title, 41
  - params, 41
  - perp, 42
  - plotting, 36
  - precalc, 42
  - predicted, 42
  - reset\_tmp, 42
  - resulthandler, 42
  - results, 42
  - score, 42
  - seaplots, 42
  - seed, 43
  - seeded, 43
  - session\_state, 43
  - stats, 43
  - title, 43
  - use\_container\_width, 43
  - val, 43
- web\_frontend.py, 80
- x
  - comparison\_plots, 17
- y
  - comparison\_plots, 18
- yaxis
  - heuristic\_web, 26