The kmeans addon

Artur Tarassow and Allin Cottrell

October 29, 2026

1 Introduction

In conjunction with gretl's built-in `kmeans` function this package provides functionality to identify clusters in multidimensional data using the K-means algorithm. This algorithm divides a set of m observations, each of dimension n, into disjoint clusters, $C_i, i = 1, \ldots, K$. Each cluster is defined by its centroid, an n-vector comprising the means of the variables for the observations it includes. In this context the variables that constitute the n dimensions of the data are often called features. Useful references include Hartigan and Wong (1979) and James *et al.* (2023, ch. 12).

The aim of K-means is to minimize a loss function, namely the total distance between the data-points and their associated cluster centroids (sometimes referred to as "inertia"). Most commonly the metric is squared Euclidean distance (sum of squares of the distances in each of the n dimensions) but other metrics—such as Manhattan (sum of the absolute distances in each dimension) or Chebyshev (greatest absolute distance in any dimension)—can also be used.

In brief, here's the relationship between the built-in `kmeans` function and the `kmeans` addon (whose primary function is `kmeans_fit`). Built-in `kmeans` is fully specialized to squared Euclidean distance, and employs a C-coded version of algorithm ASA136 (Hartigan and Wong, 1979), which is known to be particularly efficient. The addon takes the squared Euclidean metric as default, and in fact `kmeans_fit` calls built-in `kmeans` to solve the basic K-means problem if that metric is in use. But the addon also supports a variety of alternative metrics, and it provides tools to visualize the results of clustering, to assess its effectiveness, and to gauge whether the assumed number of clusters is appropriate. Plus it offers a graphical interface (see section 4).

2 The algorithm

In general terms, the K-means algorithm can be described as follows.

1. Fix on K n-vectors to serve as initial centroids. This is usually done by selection from the m observations, either at random or by means of a deterministic method designed to ensure that the selected points are widely separated (more on this below). Set total within-cluster distance to some arbitrary huge value.
2. Assign each data-point to the cluster whose centroid is nearest according to the chosen metric.
3. Calculate the resulting centroid for each cluster; calculate the total within-cluster distance and compare it to its previous value.
4. Go to step 2, unless either (a) no points were reassigned in step 2 or (b) the reduction in within-cluster distance is smaller than some specified tolerance, in which case stop.

This algorithm will almost surely proceed to a locally optimal set of clusters but there's no guarantee that the global optimum will be found.[1] In most cases there exist multiple local optima, some better than others, and the stopping point of the algorithm depends on its starting point.

[1] It's generally reckoned that mounting a direct search for the global optimum is prohibitively expensive in computational terms.

Our recommendation is to start with a deterministic initializer, follow up with several random draws, and select the best solution. For built-in `kmeans` two deterministic initializers are supported, as follows:

- Principal Components (code `pc`): the data points are sorted by value of the first Principal Component of the n-dimensional data, then K evenly spaced points are selected from the sorted array.
- Hartigan–Wong (code `hw`): the method suggested in the last paragraph of their 1979 paper. The data points are sorted by Euclidean distance from the global centroid, and again evenly spaced points are selected.

At present `kmeans_fit` supports `pc` but not `hw`. If m and/or n are very large these initializers will be relatively expensive computationally. You can prevent their use by passing `random` for the initializer option (see below for details).

The number of randomized starting points is controlled by an `n_draws` option, with a default value of 8. Some researchers recommend 20 or higher. If you just want to examine the effects of a deterministic initializer you can set `n_draws` to 0. With the random initializer at least one random drawing is always performed; the `n_draws` value should be thought of as the number of additional draws.

3 Scripting usage

This section is by way of a brief "how to". Full details on the functions mentioned below can be found in section 5.

The first function you'll need is either built-in `kmeans` or (from the addon) `kmeans_fit`. These are very similar except that `kmeans` is specialized to Euclidean distance and uses a fast algorithm. In both cases you supply the data as a list of series,[2] specify how many clusters you want, and perhaps supply some optional settings. You get back a bundle whose primary contents are a series (or column vector), `cluster_id`, holding the ID number of the cluster to which each data point belongs, and a matrix, `centroids`, holding the cluster centroids.

Here's a simple example using R. A. Fisher's famous iris dataset and mostly accepting the default values of the settings supported by these functions, apart from adding more verbosity.

```
open iris.gdt
list X = 1 .. 4
bundle opts = _(verbosity = 2)

bundle b = kmeans(X, 3, opts)
# or
include kmeans.gfn
b = kmeans_fit(X, 3, opts)
```

You can get a printout of the basic information contained in the bundles returned by these functions using `kmeans_summary`: just pass the bundle as argument.

Then you may want to visualize the clustering and assess its effectiveness. Three plotting functions are available for this job.

The `kmeans_plot` function lets you visualize the relationship between each pair of features, in an array of 2-D plots with cluster membership indicated by specific colors and point-types. Here's how usage of that function goes:

```
bundle plot_opts = _(centroid = "mean")
kmeans_plot(X, b.cluster_id, plot_opts)
```

Here we exercise one option, to show the cluster centroids along with the data points. This function is actually a wrapper for the `PairPlot` function, provided by the gretl addon of the same name. For details on the available options see the `PairPlot` documentation.

[2] In the case of `kmeans` a matrix is acceptable instead of a list.

The **silhouette value** (Rousseeuw, 1987) for an observation is a composite measure of its similarity to the members of its own cluster and to the members of its nearest neighboring cluster. These values are by construction in $[-1, 1]$, where 1 is "best" (the point is firmly located in its own cluster) and -1 is "worst" (the point appears to belong elsewhere). Values near 0 indicate overlapping clusters. The `kmeans_sil_plot` function plots these points, with a separate panel for each cluster, along with the overall mean silhouette value. You pass a bundle obtained via `kmeans` or `kmeans_fit` along with the list of features. If the clustering is effective most of the points should be in the positive zone. Example:

```
kmeans_sil_plot(b, X)
```

In the K-means context a **scree plot** provides a perspective on whether the optimal number of clusters has been specified. For 1 to some chosen maximum value of K the minimized within-cluster distance is computed and plotted against number of clusters. The idea is to look for a so-called "elbow" where the slope of the curve becomes markedly less steep. A scree plot for $K = 1, 2, \ldots, 5$, accepting the default values of the associated options, can be produced as follows.

```
kmeans_screeplot(X, 5)
```

You can also use the `kmeans_scree` function to obtain the scree data as a two-column matrix:

```
matrix S = kmeans_scree(X, 5)
print S
```

4 GUI access

K-means functionality can be accessed via the item K-Means under the View menu in gretl's main window. The dialog box is shown in Figure 1. On clicking OK you get a results window as shown in Figure 2. The "Graph" button in the toolbar pops down a menu giving access to the three sorts of plot described in section 3.

5 Reference: public functions

In the function signatures below bundles named `kmb` are assumed to have been obtained from `kmeans()` or `kmeans_fit()`. Lists named `X` are assumed to be lists of "feature" series.

```
bundle kmeans_fit (list X, numeric clusters, bundle opts[null])
```

Executes the K-means algorithm and estimates the clusters. The `X` argument, which specifies the features on which clustering is to be based, must not contain any missing values, so be sure to clean your data before calling this function. The `clusters` argument must be either a positive integer, specifying the assumed number of clusters, K, or a $K \times n$ matrix holding initial centroids. Table 1 lists the parameters that can be passed via the optional `opts` argument.

The bundle returned by `kmeans_fit` contains the items shown in Table 2.

```
void kmeans_summary (bundle kmb)
```

Prints summary information given a bundle obtained from `kmeans_fit`.

```
series kmeans_predict (list X, bundle kmb)
```

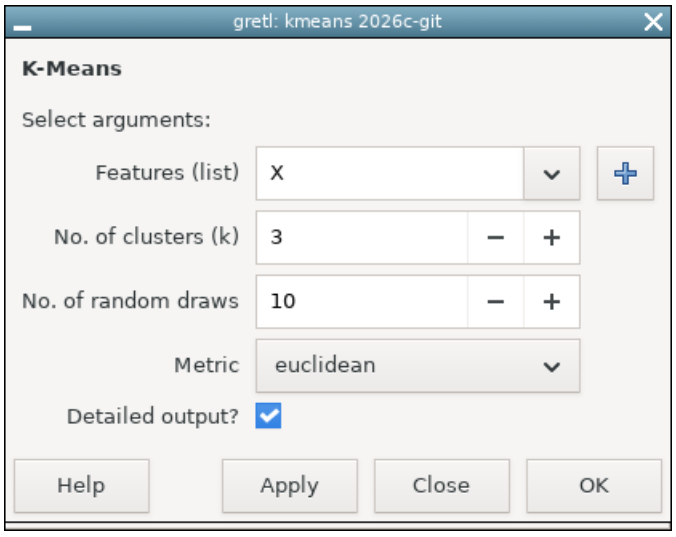


Figure 1: K-Means dialog

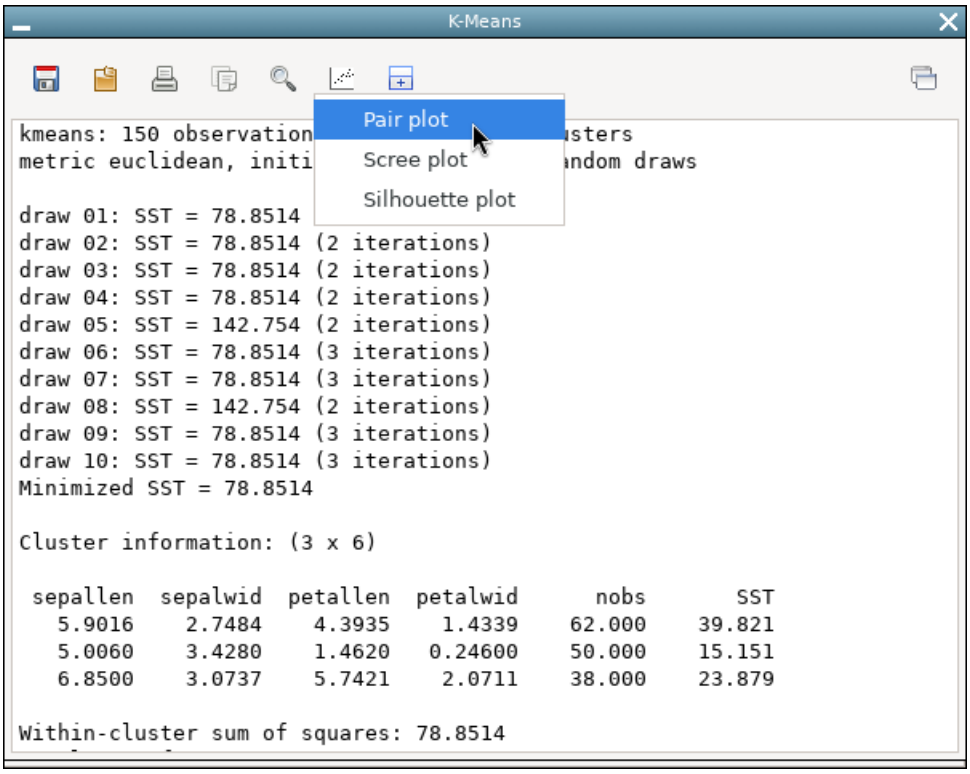


Figure 2: K-Means results window

`metric`	string	specifies the distance metric, default `euclidean`. Alternatives includes `manhattan` and `chebyshev`; see the help for gretl's `distance()` function for a full list.
`init`	string	specifies the initializer for the algorithm: `random` (the default) or `pc`
`n_draws`	scalar	integer specifying how many times the K-means algorithm will be run with different randomly selected initial centroids, default 10.
`tolerance`	scalar	value for the improvement in the objective function below which the algorithm is taken to have converged, default 0.0001.
`verbosity`	scalar	integer governing printed output: 0 means don't print anything; 1, print some details; 2, print more details.

Table 1: Summary of optional parameters for `kmeans_fit`

`within_variation`	scalar	sum of within-cluster variation
`total_variation`	scalar	sum of variation around the global mean
`centroids`	matrix	the optimized centroids, $K \times n$.
`cluster_id`	matrix	m-vector indicating for each data-point the cluster to which it is assigned
`distances`	matrix	m-vector indicating for each data-point the distance to its associated centroid
`error`	scalar	error code: 0 for no error, or a positive integer
`nobs`	scalar	the number of observations used
`maxiter`	scalar	the maximum number ofiterations to allow
`tolerance`	scalar	threshold for change in objective function below which the algorithm is taken to have converged
`init`	string	the initializer that was used
`metric`	string	the metric that was employed
`xlist`	list	the list of features

Table 2: Members of the bundle returned by `kmeans_fit`

The first argument should be a list of features, as in `kmeans_fit`, and the second should be a bundle obtained from `kmeans_fit`. The returned series holds the predicted cluster ID for each observation of `xlist` in the current sample range.

```
matrix kmeans_crosstab (series cluster_id, series actual)
```

Relevant only if the dataset in use contains a series which identifies underlying clusters, for example the `species` series in the iris dataset (`iris.gdt`). It can then be of interest to see the degree of match between the clusters obtained via the K-means algorithm (`cluster_id`) and the underlying ones (generically, `actual`). Since the numeric coding of the clusters is essentially arbitrary, a simple cross-tabulation is likely to show garbled results. This function searches for the most appropriate mapping between the estimated clusters and the "actual" ones, namely that which maximizes the number of matching observations (the trace of the cross-tabulation). The crosstab is printed using gretl's `xtab` command, and is also returned in matrix form.

```
void kmeans_plot (list X, bundle kmb)
```

Produces an array of scatter plots, one per pair of features in `X`, factorized by the `cluster_id` series in `kmb`. MORE NEEDED on the options?

```
void kmeans_screeplot (list X, int Kmax[3::5],
                       string filename[null], bundle opts[null])
```

Produces a so-called scree plot, showing total within-cluster distance as a function of the number of assumed clusters, from 1 to `Kmax`. Adding a cluster is bound to reduce the within-cluster variation, but if the plot shows an "elbow" beyond which the slope becomes markedly flatter this gives an informal indication of the most appropriate number of clusters.

If the `filename` argument is given it should be the name of a PNG file to which the graphic should be written (otherwise the plot is shown on-screen). The last argument can contain ... TO BE WRITTEN.

```
matrix kmeans_scree (list X, int Kmax, bundle opts[null])
```

Produces a two-column matrix representing the "scree" field for a K-means problem. The first column holds the number of clusters, K, from 1 to `Kmax` and the second holds the minimized within-cluster distance. The `opts` bundle can hold options pertaining to `kmeans_fit`, which is called `Kmax` times to fill the matrix.

```
series kmeans_sil_samples (bundle kmb, list X)
```

Returns a series holding a silhouette value for each observation on the features in `X`, based on the cluster information in `kmb`. See section 3 and https://en.wikipedia.org/wiki/Silhouette_(clustering) for details.

```
numeric kmeans_sil_score (bundle kmb, list X, bool global[TRUE])
```

If the `global` argument has its default value of `TRUE` or 1 this function returns a scalar value, the mean silhouette score for all the observations in `X`, based on the cluster information in `kmb`. If `global` is set to `FALSE` or 0 the return value is a vector holding the mean silhouette score for each of the clusters.

```
void kmeans_sil_plot (bundle kmb, list X, bundle opts[null])
```

Creates a silhouette plot. Each cluster is plotted in a separate panel, showing the silhouette value for each included observation. Plot options can be passed via the `opts` bundle. The dashed horizontal line shows the global mean silhouette score.

References

Hartigan, J. A. and M. A. Wong (1979) 'A K-means clustering algorithm', *Journal of the Royal Statistical Society Series C: Applied Statistics* 28(1): 100–108. URL https://doi.org/10.2307/2346830.

James, G., D. Witten, T. Hastie and R. Tibshirani (2023) *An Introduction to Statistical Learning with Applications in R*, New York: Springer. URL https://www.statlearning.com/.

Rousseeuw, P. J. (1987) 'Silhouettes: a graphical aid to the interpretation and validation of cluster analysis', *Journal of Computational and Applied Mathematics* 20: 53–65. URL https://doi.org/10.1016/0377-0427(87)90125-7.

6 Changelog

- October 2026: fix a bug whereby iteration got aborted when random centroids were not unqiue.
- September 2026: first release as an official gretl addon.