Clustering and implementation in R

Overview of clustering
The goal of clustering is to recognized discontinuous subsets in an environment that is often continuous in nature. This again creates a loss of information or some level of abstraction, but the benefit is a degree of simplification. Therefore, clustering aligns more strongly with "data mining" and visualization practices than with traditional statistics.

Most clustering algorithms rely on an association matrix!!!

Classifying clustering methods
Clustering algorithms can be *sequential* or *simultaneous*, but most are sequential. Sequential algorithms arrive at a final clustering based upon repetitive application of a set of rules. Simultaneous algorithms find the solution in a single step

Sequential algorithms can be either *agllomerative* or *divisive*. Agglomerative algorithms begin with a collection of objects that are successively grouped into larger and larger clusters until a single, all-encompassing cluster is obtained. Divisive methods start with the full collection and repetitively identify subgroups.

Clustering algorithms can be *monothetic* (use a single descriptor at each partitioning step) or *polythetic* (use all descriptors all the time, often as an association matrix).

Clustering algorithms can be *hierarchical* or *non-hierarchical*. Hierarchical methods generate clusters where inferior-ranking clusters are members of larger, higher-ranking clusters. Non-hierarchical methods create partitions without any hierarchy among the groups.

Common hierarchical clustering approaches and their implementation in R
The most common class of clustering algorithms used in biology are sequential, agglomerative and hierarchical. The results of these algorithms are usually presented as dendrograms. These algorithms can be implemented with the `hclust()` function in the base package of R.

-Single linkage agglomerative clustering (nearest neighbor)
Agglomerates objects on the basis of their shortest pairwise distances (or greatest similarities). The fusion of an object (or group) with a group at a given similarity level only requires one member of the group be linked at that level of similarity. This is implemented with the method argument `"single"` in the `hclust()` function.

-Complete linkage agglomerative clustering (furthest neighbor)
This method is the opposite of nearest neighbor. It requires that the most distant pair of objects or objects within groups meet the required distance to fuse the objects or groups. This is implemented with the method argument `"complete"` in the `hclust()` function.

-Average Agglomerative Clustering
This represents a family of algorithms that use average distances or similarities for identifying groups. The algorithms can differ in whether averages or centroids of clusters are used and whether groups are weighted or not by the number of objects in the groups. The most common algorithm in this family is Unweighted Pair Group Method with Arithmetic Mean (UPGMA). This is implemented with the method argument `"average"` in the `hclust()` function.

Evaluating alternative clustering methods
Because the outcome of a clustering application is dependent on the association metric and clustering algorithm chose, some metrics exist to compare clustering results. A couple common methods are described below.

*Cophenetic Correlation – this evaluates the degree to which the clustering dendrogram reflects the association matrix. This is implemented with the function `cophenetic()`.

*Identifying interpretable clusters with fusion level values – This allows you to visualize the number of groups that exist at a given node height and comparisons can be made across clustering results.

*Heatmaps – This allows you to look at descriptor abundances within object clusters in a two-dimensional visualization.

A useful non-hierarchical clustering algorithm: k-Means Partitioning
This method iteratively minimizes the total error sum of squares, which is the sum of the within-group sums of squares (sums of distances among the objects in the group divided by the number of objects in the group). A pre-determined number of groups can be defined and implemented using the `kmeans()` function. You can also use a function in the `vegan` package, `cascadeKM()`, to identify the "optimal" number of partitions. This however requires specification of an optimization function. A common one is the 'Simple Structure Index", which is maximized.