Problem 1

1. The k-medoid or PAM (Partitioning Around Medoids) algorithm is a clustering algorithm similar to the k-means algorithm. A medoid can be defined as the object of a cluster whose average dissimilarity to all the objects in the cluster is minimal, that is, it is a most centrally located point in the cluster.

k-medoids chooses data points as centers (medoids or exemplars) and can be used with arbitrary distances, while in k-means the center of a clusters is not necessarily one of the input data points (it is the average between the points in the cluster).

Advantages of K medoid

- More robust to noise and outliers as compared to k means because it minimizes a sum of pairwise dissimilarities instead of a sum of squared Euclidean distances.
- K medoid is more flexible than k mean. eg. Absolute Pearson Correlation must not be used with k-means, but it works well with k medoid

Disadvantages of K medoid

- It requires precision
- Quite Complex Algorithm
- PAM takes much longer to run than k-means. As it involves computing all pairwise distances, it is O(n^2*k*i); whereas k-means runs in O(n*k*i) where usually, k times the number of iterations is k*i << n.

In addition to **"k-medoid"** and **"k-mean"**. **"k-median"** in another method that is robust to outlier data but it prone to high bias model.

2. There are numerous distance metrics that can be used in k-means algorithm in the following table they are discussed.

Distance Measure	Equation	Time complexity	Advantages	Disadvantages	Applications
Euclidean Distance	$d_{euc} = \left[\sum_{i=1}^{n}(x_i - y_i)^2\right]^{\frac{1}{2}}$	O(n)	Very common, easy to compute and works well with datasets with compact or isolated clusters [27,31].	Sensitive to outliers [27,31].	*K*-means algorithm, Fuzzy *c*-means algorithm [38].
Average Distance	$d_{ave} = \left(\frac{1}{n}\sum_{i=1}^{n}(x_i - y_i)^2\right)^{\frac{1}{2}}$	O(n)	Better than Euclidean distance [35] at handling outliers.	Variables contribute independently to the measure of distance. Redundant values could dominate the similarity between data points [37].	*K*-means algorithm
Weighted Euclidean	$d_{we} = \left(\sum_{i=1}^{n}w_i(x_i - y_i)^2\right)^{\frac{1}{2}}$	O(n)	The weight matrix allows to increase the effect of more important data points than less important one [37].	Same as Average Distance.	Fuzzy *c*-means algorithm [38]
Chord	$d_{chord} = \left(2 - 2\frac{\sum_{i=1}^{n}x_i y_i}{\|x\|_2\|y\|_2}\right)^{\frac{1}{2}}$	O(3n)	Can work with un-normalized data [27].	It is not invariant to linear transformation [33].	Ecological resemblance detection [35].
Mahalanobis	$d_{mah} = \sqrt{(x-y)S^{-1}(x-y)^T}$	O(3n)	Mahalanobis is a data-driven measure that can ease the distance distortion caused by a linear combination of attributes [35].	It can be expensive in terms of computation [33]	Hyperellipsoidal clustering algorithm [30].
Cosine Measure	$\text{Cosine}(x,y) = \frac{\sum_{i=1}^{n}x_i y_i}{\|x\|_2\|y\|_2}$	O(3n)	Independent of vector length and invariant to rotation [33].	It is not invariant to linear transformation [33].	Mostly used in document similarity applications [28,33].
Manhattan	$d_{man} = \sum_{i=1}^{n}(x_i - y_i)$	O(n)	Is common and like other Minkowski-driven distances it works well with datasets with compact or isolated clusters [27].	Sensitive to the outliers. [27,31]	*K*-means algorithm
Mean Character Difference	$d_{MCD} = \frac{1}{n}\sum_{i=1}^{n}\|x_i - y_i\|$	O(n)	*Results in accurate outcomes using the K-medoids algorithm.	*Low accuracy for high-dimensional datasets using K-means.	Partitioning and hierarchical clustering algorithms.
Index of Association	$d_{IOA} = \frac{1}{n}\sum_{i=1}^{n}\left\|\frac{x_i}{\sum_{i=1}^{n}x_i} - \frac{y_i}{\sum_{i=1}^{n}y_i}\right\|$	O(3n)	-	*Low accuracy using K-means and K-medoids algorithms.	Partitioning and hierarchical clustering algorithms.
Canberra Metric	$d_{canb} = \sum_{i=1}^{n}\frac{\|x_i - y_i\|}{(x_i + y_i)}$	O(n)	*Results in accurate outcomes for high-dimensional datasets using the K-medoids algorithm.	-	Partitioning and hierarchical clustering algorithms.
Czekanowski Coefficient	$d_{czekan} = 1 - \frac{2\sum_{i=1}^{n}min(x_i, y_i)}{\sum_{i=1}^{n}(x_i + y_i)}$	O(2n)	*Results in accurate outcomes for medium-dimensional datasets using the K-means algorithm.	-	Partitioning and hierarchical clustering algorithms.
Coefficient of Divergence	$d_{canb} = \left(\frac{1}{n}\sum_{i=1}^{n}\left(\frac{x_i - y_i}{x_i + y_i}\right)^2\right)^{\frac{1}{2}}$	O(n)	*Results in accurate outcomes using the K-means algorithm.	-	Partitioning and hierarchical clustering algorithms.
Pearson coefficient	$Pearson(x,y) = \frac{\sum_{i=1}^{n}(x_i - \mu_x)(y_i - \mu_y)}{\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}}$	O(2n)	*Results in accurate outcomes using the hierarchical single-link algorithm for high dimensional datasets.	-	Partitioning and hierarchical clustering algorithms.

*Points marked by asterisk are compiled based on this article's experimental results.

doi:10.1371/journal.pone.0144059.t001

3. If your variables are of incomparable units (e.g. height in cm and weight in kg) then you should standardize variables, of course. Even if variables are of the same units but show quite different variances it is still a good idea to standardize before K-means. You see, K means clustering is "isotropic" in all directions of space and therefore tends to produce more or less round (rather than elongated) clusters. In this situation leaving variances unequal is equivalent to putting more weight on variables with smaller variance, so clusters will tend to be separated along variables with greater variance.

A different thing also worth to remind is that K-means clustering results are potentially sensitive to the order of objects in the data set1. A justified practice would be to run the analysis several times, randomizing objects order; then average the cluster centres of those runs and input the centers as initial ones for one final run of the analysis.

Here is some general reasoning about the issue of standardizing features in cluster or other multivariate analysis.

Specifically, (1) some methods of centers initialization are sensitive to case order; (2) even when the initialization method isn't sensitive, results might depend sometimes on the order the initial centers are introduced to the program by (in particular, when there are tied, equal distances within data); (3) so-called running means version of k-means algorithm is naturally sensitive to case order (in this version - which is not often used apart from maybe online clustering - recalculation of centroids take place after each individual case is re-assigned to another cluster).

4. The linkage methods work by calculating the distances or similarities between all objects. Then the closest pair of clusters are combined into a single cluster, reducing the number of clusters remaining.

The process is then repeated until there is only a single cluster left.

- Method of **single linkage** or nearest neighbor. Proximity between two clusters is the proximity between their two closest objects. This value is one of values of the input matrix. The conceptual metaphor of this built of cluster, its archetype, is spectrum or chain. Chains could be straight or curvilinear, or could be like "snowflake" or "amoeba" view. Two most dissimilar cluster members can happen to be very much dissimilar in comparison to two most similar. Single linkage method controls only nearest neighbors similarity.

- Method of **complete linkage** or farthest neighbour. Proximity between two clusters is the proximity between their two most distant objects. This value is one of values of the input matrix. The metaphor of this built of cluster is circle (in the sense, by hobby or plot) where two most distant from each other members cannot be much more dissimilar than other quite dissimilar pairs (as in circle). Such clusters are "compact" contours by their borders, but they are not necessarily compact inside.

- Method of **between-group** average linkage (UPGMA). Proximity between two clusters is the arithmetic mean of all the proximities between the objects of one, on one side, and the objects of the other, on the other side. The metaphor of this built of cluster is quite generic, just united class or close-knit collective; and the method is frequently set the default one in hierarchical clustering packages. Clusters of miscellaneous shapes and outlines can be produced.

- **Simple average**, or method of equilibrious between-group average linkage (WPGMA) is the modified previous. Proximity between two clusters is the arithmetic mean of all the proximities between the objects of one, on one side, and the objects of the other, on the other side; while the sub clusters of which each of these two clusters were merged recently have equalized influence on that proximity – even if the sub clusters differed in the number of objects.

- Method of **within-group average** linkage (MNDIS). Proximity between two clusters is the arithmetic mean of all the proximities in their joint cluster. This method is an alternative to UPGMA. It usually will lose to it in terms of cluster density, but sometimes will uncover cluster shapes which UPGMA will not.

- **Centroid** method (UPGMC). Proximity between two clusters is the proximity between their geometric centroids: [squared] Euclidean distance between those. The metaphor of this built of cluster is proximity of platforms (politics). Like in political parties, such clusters can have fractions or "factions", but unless their central figures are apart from each other the union is consistent. Clusters can be various by outline.

- **Ward's** method, or minimal increase of sum-of-squares (MISSQ), sometimes incorrectly called "minimum variance" method. Proximity between two clusters

is the magnitude by which the summed square in their joint cluster will be greater than the combined summed square in these two clusters: $SS_{12} - (SS_1 + SS_2)$. (Between two singleton objects this quantity = squared Euclidean distance / 2.) The metaphor of this built of cluster is type. Intuitively, a type is a cloud denser and more concentric towards its middle, whereas marginal points are few and could be scattered relatively freely.

- Method of minimal **sum-of-squares** (MNSSQ). Proximity between two clusters is the summed square in their joint cluster: SS_{12}. (Between two singleton objects this quantity = squared Euclidean distance / 2.)

- Method of minimal increase of variance (MIVAR). Proximity between two clusters is the magnitude by which the mean square in their joint cluster will be greater than the weightily (by the number of objects) averaged mean square in these two clusters:$MS_{12} - \frac{n_1 MS_1 + n_2 MS_2}{n_1 + n_2} = \frac{[SS_{12} - (SS_1 + SS_2)]}{(n_1 + n_2)}$. (Between two singleton objects this quantity = squared Euclidean distance / 4.)

- Method of minimal variance (MNVAR). Proximity between two clusters is the mean square in their joint cluster:$MS_{12} = SS_{12}/(n_1 + n_2)$. (Between two singleton objects this quantity = squared Euclidean distance / 4.)

First 5 methods permit any proximity measures (any similarities or distances) and results will, naturally, depend on the measure chosen.

Last 6 methods require distances; and fully correct will be to use only squared Euclidean distances with them, because these methods compute centroids in Euclidean space. Therefore, distances should be Euclidean for the sake of geometric correctness (these 6 methods are called together geometric linkage methods). At worst case, you might input other metric distances at admitting more heuristic, less rigorous analysis. Now about that "squared". Computation of centroids and deviations from them are most convenient mathematically/programmically to perform on squared distances, that's why HAC packages usually require to input and are tuned to process the squared ones. However, there exist implementations - fully equivalent yet a bit slower - based on no squared distances input and requiring those; see for example "Ward-2" implementation for Ward's method. You should consult with the documentation of you clustering program to know which - squared or not - distances it expects at input to a "geometric method" in order to do it right.

Methods MNDIS, MNSSQ, and MNVAR require on steps, in addition to just update the Lance-Williams formula, to store a within-cluster statistic (which depends on the method).

Methods which are most frequently used in studies where clusters are expected to be solid more or less round clouds, - are methods of average linkage, complete linkage method, and Ward's method.

Ward's method is the closest, by it properties and efficiency, to K-means clustering; they share the same objective function - minimization of the pooled within-cluster SS "in the end". Of course, K-means (being iterative and if provided with decent initial centroids) is usually a better minimizer of it than Ward. However, Ward seems to me a bit more accurate than K-means in uncovering clusters of uneven physical sizes (variances) or clusters thrown about space very irregularly. MIVAR method is weird to me, I can't imagine when it could be recommended, it doesn't produce dense enough clusters.

Methods centroid, median, minimal increase of variance – may give sometimes the so-called reversals: a phenomenon when the two clusters being merged at some step appear closer to each other than pairs of clusters merged earlier. That is because these methods do not belong to the so called ultra-metric. This situation is inconvenient but is theoretically OK.

Methods of single linkage and centroid belong to so called space contracting, or “chaining”. That means - roughly speaking - that they tend to attach objects one by one to clusters, and so they demonstrate relatively smooth growth of curve “% of clustered objects”. On the contrary, methods of complete linkage, Ward’s, sum-of-squares, increase of variance, and variance commonly get considerable share of objects clustered even on early steps, and then proceed merging yet those – therefore their curve “% of clustered objects” is steep from the first steps. These methods are called space dilating. Other methods fall in-between.

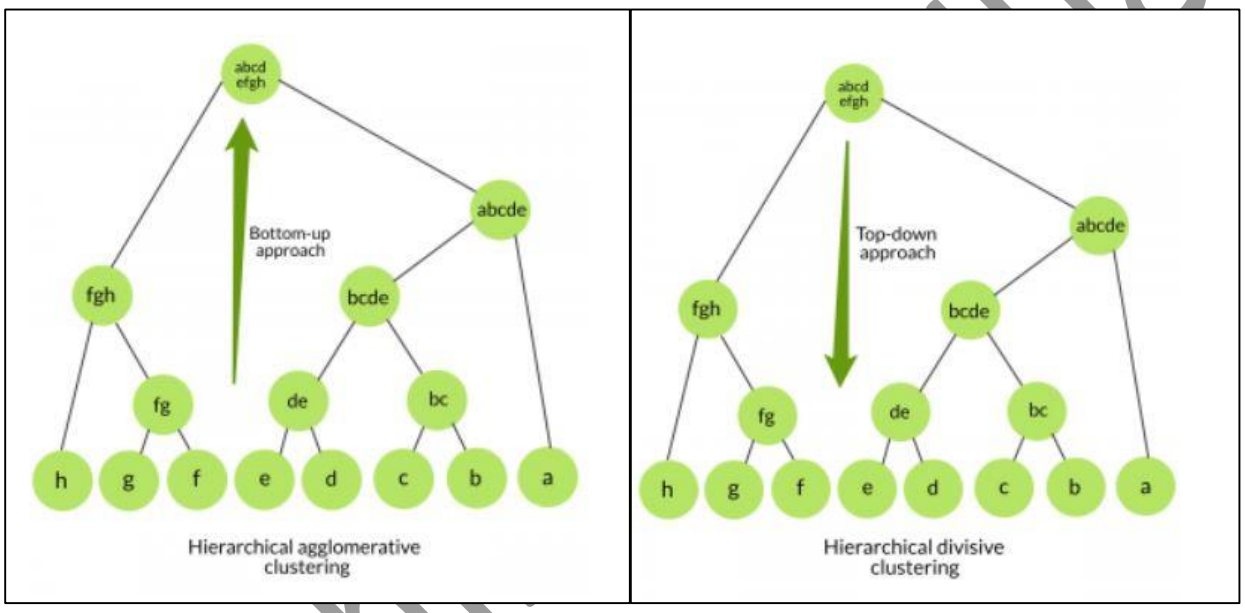
5. Hierarchical Agglomerative vs Divisive clustering

- Divisive clustering is more complex as compared to agglomerative clustering, as in case of divisive clustering we need a flat clustering method as “subroutine” to split each cluster until we have each data having its own singleton cluster.

- Divisive clustering is more efficient if we do not generate a complete hierarchy all the way down to individual data leaves. Time complexity of a naive agglomerative clustering is O(n3) because we exhaustively scan the N x N matrix dist_mat for the

lowest distance in each of N-1 iterations. Using priority queue data structure, we can reduce this complexity to O(n2logn). By using some more optimizations, it can be brought down to O(n2). Whereas for divisive clustering given a fixed number of top levels, using an efficient flat algorithm like K-Means, divisive algorithms are linear in the number of patterns and clusters.

- Divisive algorithm is also more accurate. Agglomerative clustering makes decisions by considering the local patterns or neighbor points without initially taking into account the global distribution of data. These early decisions cannot be undone. whereas divisive clustering takes into consideration the global distribution of data when making top-level partitioning decisions.



6.

"Eps" and "minimum number of parents" (minpts) are both considered hyper parameters. There are no algorithms to determine the perfect values for these, given a dataset. Instead, they must be optimized largely based on the problem you are trying to solve. Some ideas on how to optimize:
minpts should be larger as the size of the dataset increases.
eps is a value that deals with the radius of the clusters you are trying to find. To choose a value, we can perform a sort of elbowing technique (a similar technique that is often used to determine an optimal k in K-Means clustering).

- Let k = the number of nearest neighbors
- For a value of k, for each point in a dataset, calculate the average distance between each point and its k-nearest neighbors (some packages have this function built in somewhere)

- Plot number of points on the X axis and average distances on the y axis that you calculated.
- The resulting graph should be increasing (as long as you sort your arrays increasingly by average distance) and concave up. There should be a point where the rate of increase jumps drastically, this point is called the elbow point and contains your optimal eps, which is the y value of the elbow point.
- Run this algorithm using different values of k and compare results.