Montogue

Tutorial ST7
Cluster Analysis with R

Lucas Monteiro Nogueira

• Summary •	
Problem 1	***K*-Means clustering**
Problem 2	**Is a dataset clusterable?**
Problem 3	**Estimating the number of clusters**
Problem 4	***K*-Medoid clustering (PAM)**
Problem 5	**Hierarchical clustering and dendrograms**
Problem 6	**Selecting an algorithm**

PROBLEMS

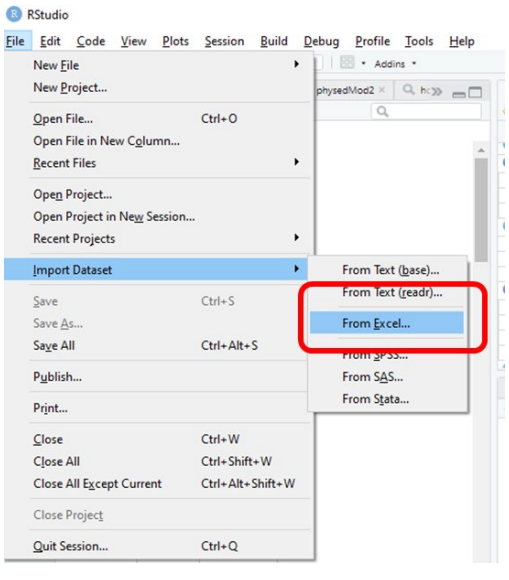
▶ Packages used

stats – *R's main statistical package*	**clValid** – *Validation of cluster results*
factoextra – *Cluster analysis visualization*	**cluster** – *Further cluster analysis*
clustertend – *Clustering tendency*	**dendextend** – *Further dendrogram tools*
NbClust – *Clustering tendency based on indices*	

▶ Problem 1 – K-Means clustering

We will be working with a simple dataset loosely based on the manual for version 13 of the Stata® statistical software. The data, named *physeduc*, can be found in our Google Drive folder. It describes the capability of 80 students in three skills of particular value for a physical education class: *flexibility*, *speed*, and *strength*. Each student has a skill level in these traits ranging from 0 to 10, with allowance for two decimal places. A P.E. teacher has arbitrarily assigned each student to one of four groups, but is seeking the help of a statistician to optimize his classes so that students with similar skill profiles are assigned to the same group. In the dataset, each student is described by a row; accordingly, there are 80 rows. Further, there are four columns: the first column, *grp*, is the ID of the student's group *before* the teacher sought the help of a statistician; the second column contains *flexibility* scores; the third column contains *speed* scores; lastly, the fourth column contains *strength* scores.

To load the dataset in RStudio, head to *File* → *Import Dataset* → *From Excel*, and browse to the folder in which you've saved *physeduc.xlsx*.

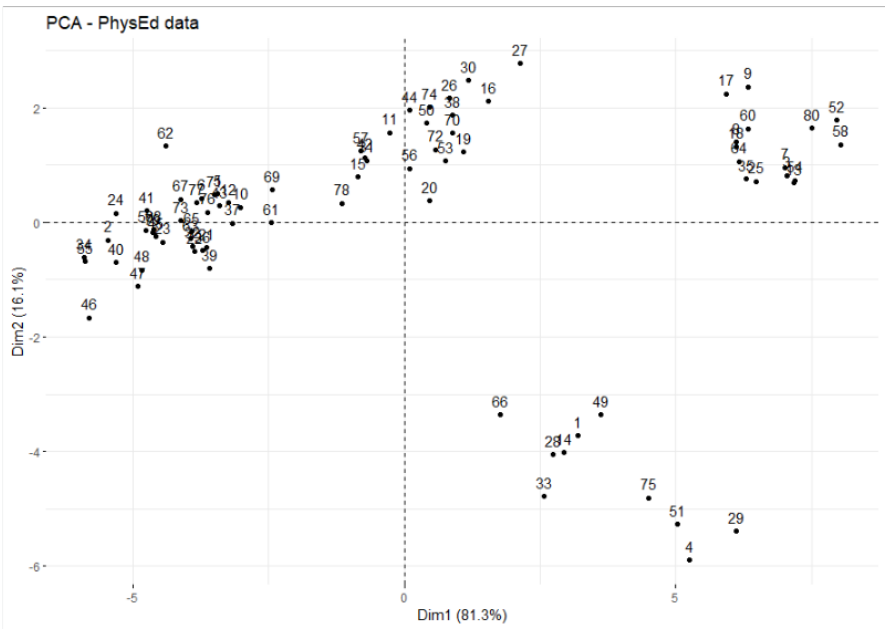


Next, we must eliminate the first column in the dataset, so as to get rid of the grouping scheme adopted by the P.E. teacher. Let's name the updated data frame *physedMod*:

```
> physedMod <- physeduc[,-1]
```

The first algorithm we'll work with is *k*-means clustering, a simple and well-known partitioning scheme. A reasonable first step, however, is to reduce the number of dimensions through a quick principal component analysis (PCA) and check for patterns; the PCA can be performed by dint of the *prcomp* (*stats* package) and *fviz_pca_ind* (*factoextra* package) commands:

```
> fviz_pca_ind(prcomp(physedMod), title = "PCA - PhysEd data")
```



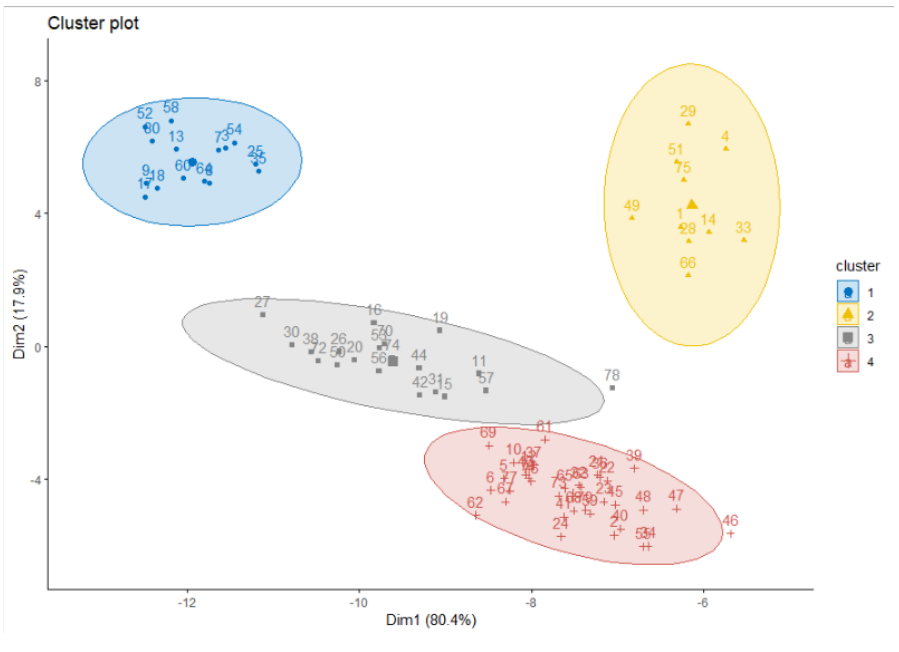
Clearly, there are 4 well-defined regions in the PCA plot, or 3 if we consider the dots in the upper-left quadrant as part of a single, slender cluster. Following the P.E. teacher's original arrangement, we'll work with 4 clusters. Once the number of clusters has been estimated, we can proceed with *k*-means clustering via the *kmeans* command (*stats* package). At this point, it is worth noting that a common cautionary step in a cluster analysis is to first scale the data with the function *scale()*. This is especially useful when working with measurements in distinct physical units. Since we are dealing with a simple 3-variable set, in which all 3 variables are contained within the same interval [0; 10], we can skip this step for the *k*-means partitioning, although, as we will see in later sections, scaling may be necessary before use is made of some visualization commands.

Scaling issues aside, we can now implement the *k*-means clustering algorithm:

```
> pe.km <- kmeans(physedMod, 4)
```

Next, we employ the *fviz_cluster() (factoextra* package) command to visualize the clusters:

```
> fviz_cluster(list(data = physedMod, cluster = pe.km$cluster),
ellipse.type = "norm", geom = "point", stand = FALSE,
palette="jco", ggtheme = theme_classic())
```



Note that *stand* = FALSE prevents the software from standardizing the data before processing it. Also, the palette *jco* stands for 'Journal of Clinical Oncology.' The four clusters are well-defined, as they should be.

In addition to *cluster*, the *kmeans* command yields several other measurements. The most useful one, arguably, is *centers*, which produces the cluster means:

```
> pe.km$centers

  flexibility  speed  strength
1    8.852000 8.743333 4.358000
2    3.157000 6.988000 1.641000
3    5.946500 3.448500 6.832500
4    1.969429 1.144857 8.478857
```

We can use these means to assist the P.E. teacher in tailoring his classes to different study groups. For example, students in group 4 excel at strength (score ~8.5) but have lackluster performance (score ~1 to 2) in flexibility and speed. On the other hand, students clustered in group 1 are skilled in terms of flexibility and speed, but not so in terms of strength.

▶ Problem 2 – Is a dataset clusterable?

The Hopkins statistic is used to assess the clustering tendency of a dataset. Simply put, the Hopkins statistic performs the following hypothesis test:

Null hypothesis: The dataset is uniformly distributed (i.e., there are no meaningful clusters).
Alternative hypothesis: The dataset is not uniformly distributed (i.e., it contains meaningful clusters).

If the value of the Hopkins statistic is sufficiently low, then we can reject the null hypothesis and conclude that the dataset is clusterable. The R function *hopkins()* (*clustertend* package) can be used to compute this statistic.

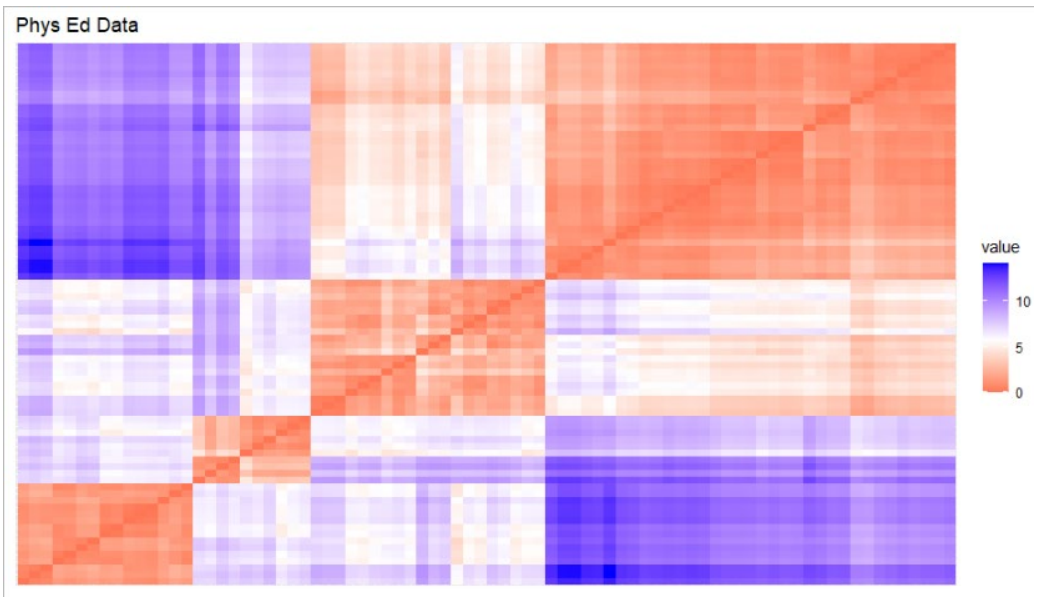
```
> hopkins(physedMod, nrow(physedMod)-1)
$H
[1] 0.1684756
```

A commonly used threshold value is 0.5; since $H < 0.5$ in the case at hand, we may reject the null hypothesis and surmise that the dataset *physedMod* is clusterable.

We can also assess clustering tendency by visualizing the dissimilarity matrix. The syntax in this case is to use *fuiz_dist (factoextra* package) wrapped in *dist()*, which is used to compute the dissimilarity matrix per se.

```
> fviz_dist(dist(physedMod), show_labels = FALSE) + labs(title
= "Phys Ed Data")
```

The plot is shown below. As can be seen, there are well-defined regions of high similarity (or low dissimilarity), as indicated by the large red rectangles; this suggests that the data is in fact clusterable.



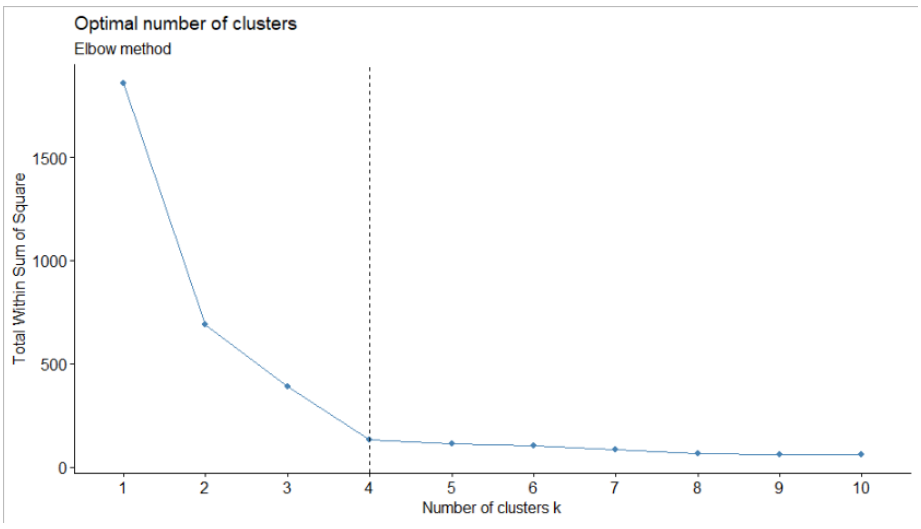
▶ Problem 3 – Estimating the number of clusters

As you know, partitioning methods such as the *k*-means algorithm define clusters such that the total intra-cluster variation, or within-cluster sum of squares (WSS), is minimized. The total WSS measures the compactness of the clustering, hance we aim to have it be as small as possible. Accordingly, the optimal number of clusters can be established as follows:

1. Compute the clustering algorithm (e.g., *k*-means clustering) for different values of *k* – say, using *k* from 1 to 10 clusters.
2. For each *k*, calculate the total within-cluster sum of squares (WSS).
3. Plot WSS as a function of the number of clusters *k*.
4. The location of a bend (knee) in the plot is generally considered as an indicator of the appropriate number of clusters.

To implement the graph mentioned in step 3, we may use the *fviz_nbclust (factoextra* package) command with *wss* as the method:

```
> fviz_nbclust(physedMod, kmeans, method="wss") +
geom_vline(xintercept = 4, linetype = 2) + labs(subtitle =
"Elbow method")
```



The vertical dashed line indicates the suggested number of clusters – in this case, 4.

A second, perhaps better, approach to establish the number of clusters is the *NbClust* function located in the eponymous package. This command computes 30 indices available in the literature and recommends a number of clusters on the basis of which number was identified as the optimal choice in most indices. To try out this command in the situation at hand, we type the following:

```
> nb <- NbClust(physedMod, distance = "euclidean", min.nc = 2,
max.nc = 10, method = "kmeans")
Among all indices:
* 8 proposed 2 as the best number of clusters
* 7 proposed 3 as the best number of clusters
* 1 proposed 5 as the best number of clusters
* 4 proposed 6 as the best number of clusters
* 1 proposed 7 as the best number of clusters
* 3 proposed 8 as the best number of clusters

                   ***** Conclusion *****

* According to the majority rule, the best number of clusters
is 2
```

The output indicates that most indices favor a number of clusters equal to 2, which in turn is closely followed by 3. Thus, the graphical method suggests that we take 4 as the number of clusters, whereas the majority rule afforded by *NbClust()* suggests that we take only 2 clusters. In the sequel, we'll work with 4 clusters; the reader is invited to repeat the following analysis for a 2-cluster scheme and check whether it yields better results than the 4-cluster scheme.

▶ Problem 4 – K-medoid clustering (PAM)

K-Medoid clustering is a robust alternative to *k*-means clustering, as it is less sensitive to noise and outliers. The most common *k*-medoids technique is the PAM (**P**artitioning **A**round **M**edoids) algorithm. This algorithm can be implemented in R with the functions *pam* (*cluster* package), *pamk* (*fpc* package), and others; importantly, *pam* requires the user to specify the number of clusters beforehand, whereas *pamk* does not. In the present tutorial, we'll be working with *pam* only.

A *pam* call syntax is no different from a *kmeans* call:

```
> pe.pam <- pam(physedMod, 4)
```

This command yields an object with several components, including: *clustering*, which contains the cluster number of each data point (i.e., each student), and *medoids*, which contains the medoid values associated with each cluster. Accessing medoids, we obtain the following:

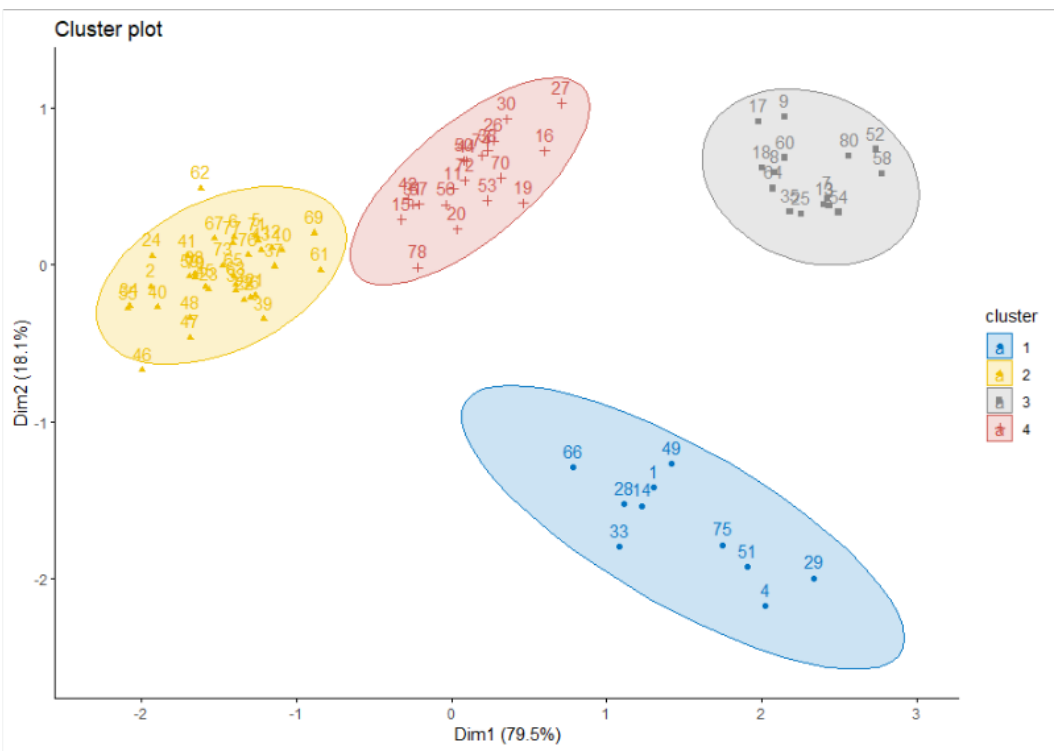
```
> pe.pam$medoids
```

```
     flexibility speed  strength
[1,]        3.26  5.95     1.97
[2,]        2.10  1.20     8.59
[3,]        8.79  8.83     3.91
[4,]        6.36  3.67     6.50
```

Similarly to the analysis with *k*-means, we can use *fviz_cluster* to visualize clusters:

```
> fviz_cluster(pe.pam, ellipse.type = "norm", stand = FALSE,
palette = "jco", ggtheme = theme_classic())
```

The clusters are shown below.



▶ Problem 5 – Hierarchical clustering and dendrograms

Turning our attention to hierarchical clustering, we first compute the dissimilarity matrix using the *dist* command (*stats* package); since we'll be working with Euclidean distances, we add the argument *method* = "*euclidean*."

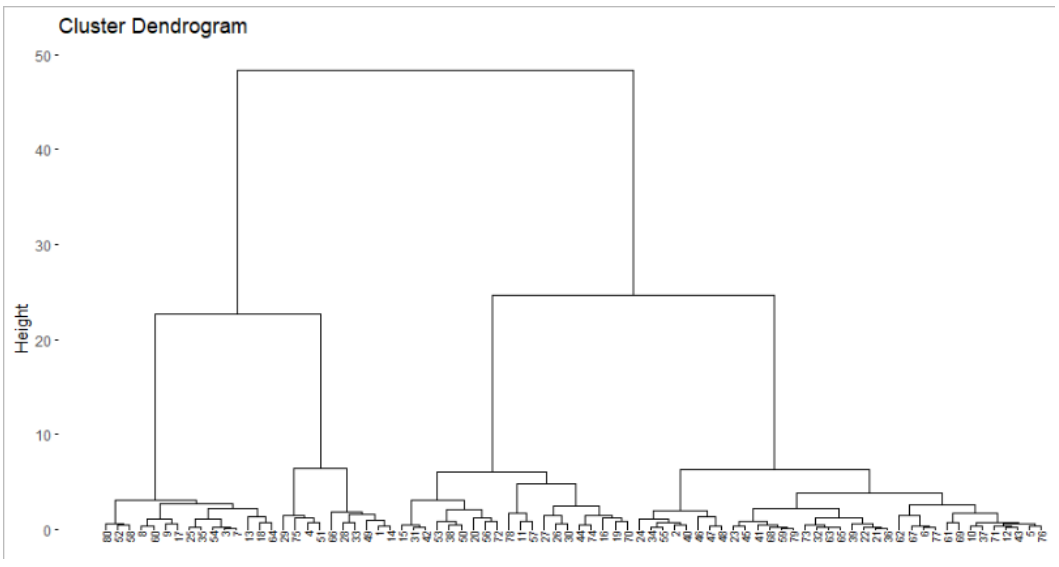
```
> distances = dist(physedMod, method="euclidean")
```

Next, we make use of the *hclust* command (*stats* package) with *ward.D2* as the linkage method:

```
> hc.physed <- hclust(d = distances, method="ward.D2")
```

Once the hierarchical clustering algorithm has been implemented, we can visualize the dendrogram with the *fviz_dend* command (*factoextra* package); the *cex* argument controls label size.

```
> fviz_dend(hc.physed, cex = 0.5)
```



Now that the objects have been grouped into a hierarchical tree, an obvious next step is to check whether the distances (i.e., heights) in the tree reflect the original distances accurately. This is where *cophenetic distances* come in; these can be computed with the *cophenetic* command (*stats* package):

```
> cph.distances <- cophenetic(hc.physed)
```

Then, we compute the correlation between *cph.distances* and *distances*:

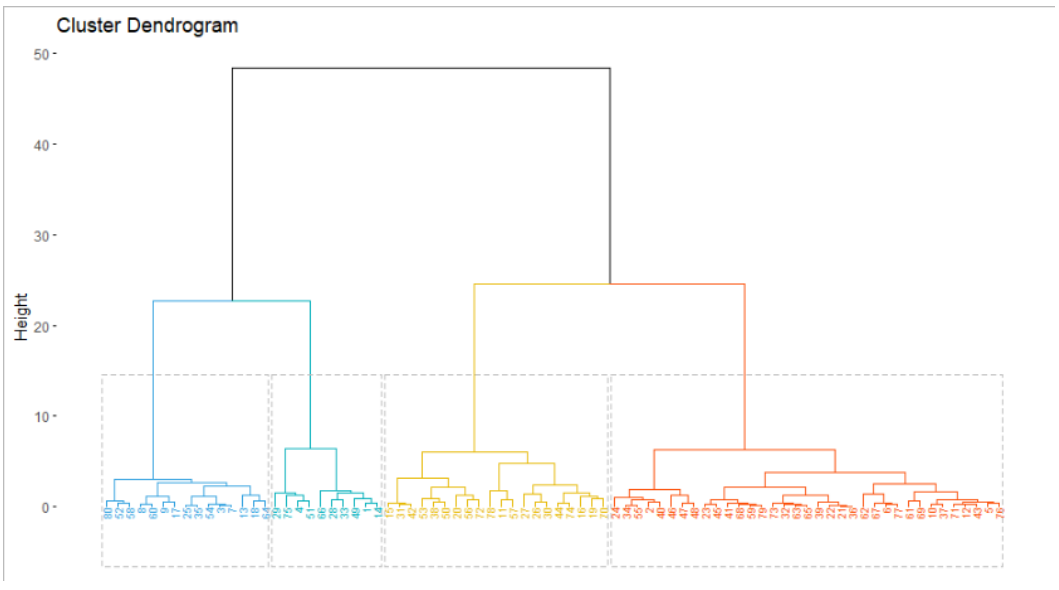
```
> cor(distances,cph.distances)
[1] 0.8807885
```

The closer the correlation is to unity, the more accurately the clustering scheme reproduces the relationships between the corresponding data. Our correlation is close to 0.881, which indicates a good agreement.

Now, the *fuiz_dend* command (*factoextra* package) introduced just now can be customized for better visualization. For example, we can employ the following code to highlight and color different branches:

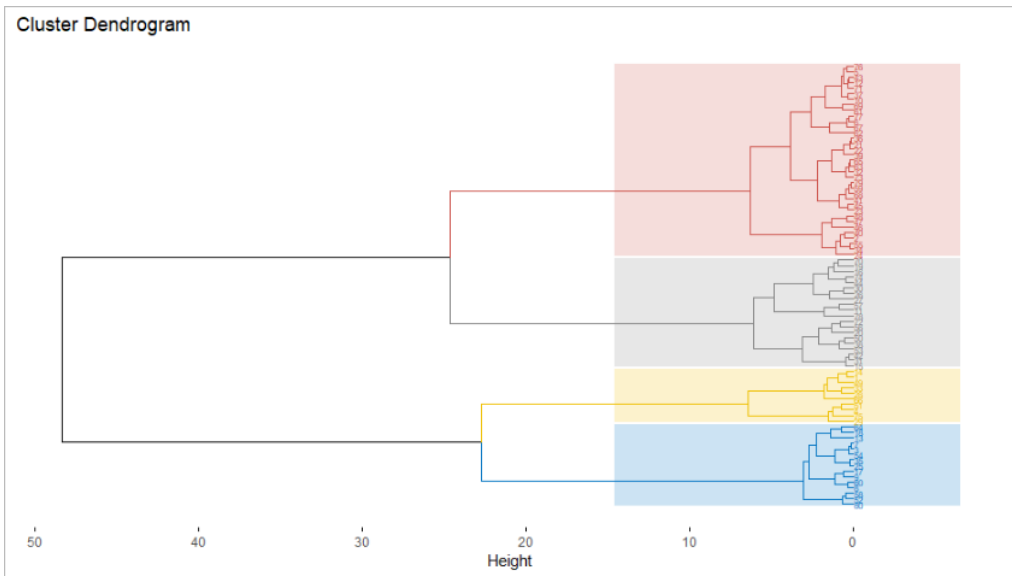
```
> fviz_dend(hc.physed, k = 4, cex = 0.5,
k_colors=c("#2E9FDF","#00AFBB","#E7B800","#FC4E07"),
color_labels_by_k = TRUE, rect = TRUE)
```

The output is shown below. Note that *k_colors* is used to select colors for the branches; *color_labels_by_k = TRUE* is used to, you guessed it, color labels according to their branch colors; *rect = TRUE* is used to enclose the clusters in rectangles, as shown.



We may also plot the dendogram horizontally, or with a slightly modified styling:

```
> fviz_dend(hc.physed, k = 4, cex = 0.4, horiz = TRUE,
k_colors = "jco", rect = TRUE, rect_border = "jco", rect_fill
= TRUE)
```



Our dendrogram plots, while visually neat, make it impossible to visualize leaf labels and smaller branches. Fortunately, we can easily plot subtrees in greater detail. We begin by programming the entire dendrogram as usual:

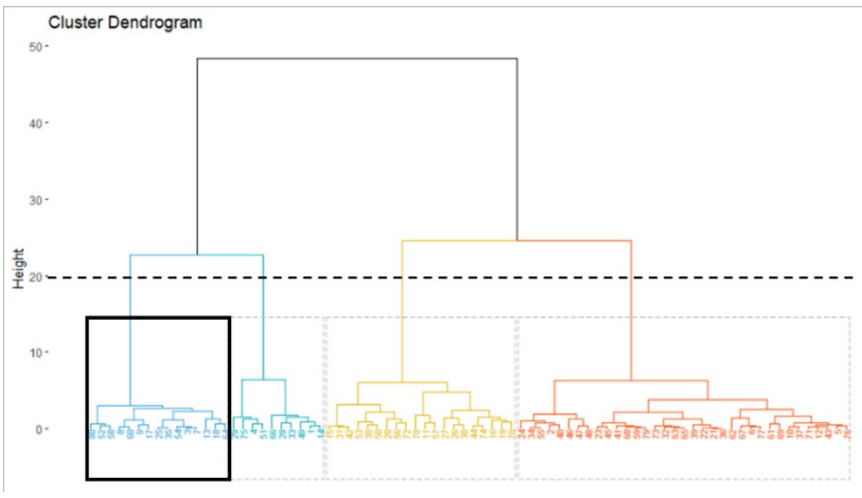
```
> dendPlot <- fviz_dend(hc.physed, k = 4, cex = 0.5, k_colors
= "jco")
```

Then, we extract the dendrogram data via the *attr* command (*base* package):

```
> dendData <- attr(dendPlot, "dendrogram")
```

Next, we 'cut' (command *cut*, also from *base* package) the dendrogram at some height *h* of interest. Assuming we want to see the blue cluster highlighted below in greater detail, we may choose the height *h* = 20, as shown.

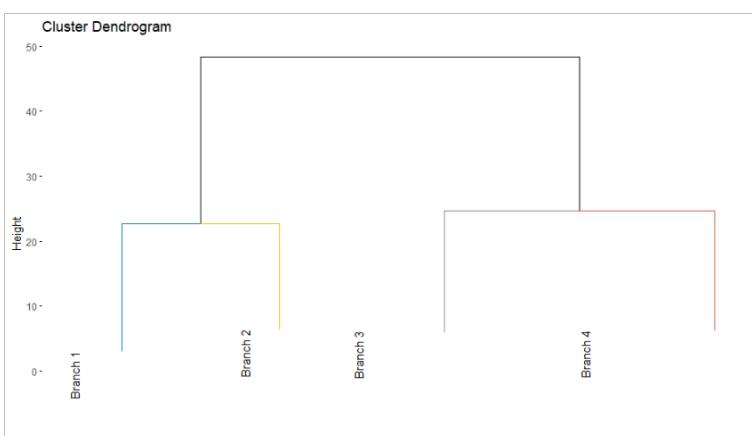
```
> dendCuts <- cut(dendData, h = 20)
```



Finally, the dendrogram segments can be visualized with *fviz_dend*, as we have been doing so far, one difference being that we must specify whether we want to plot the upper or lower regions of the segmented dendrogram. For instance, to obtain the upper part:

```
> fviz_dend(dendCuts$upper)
```

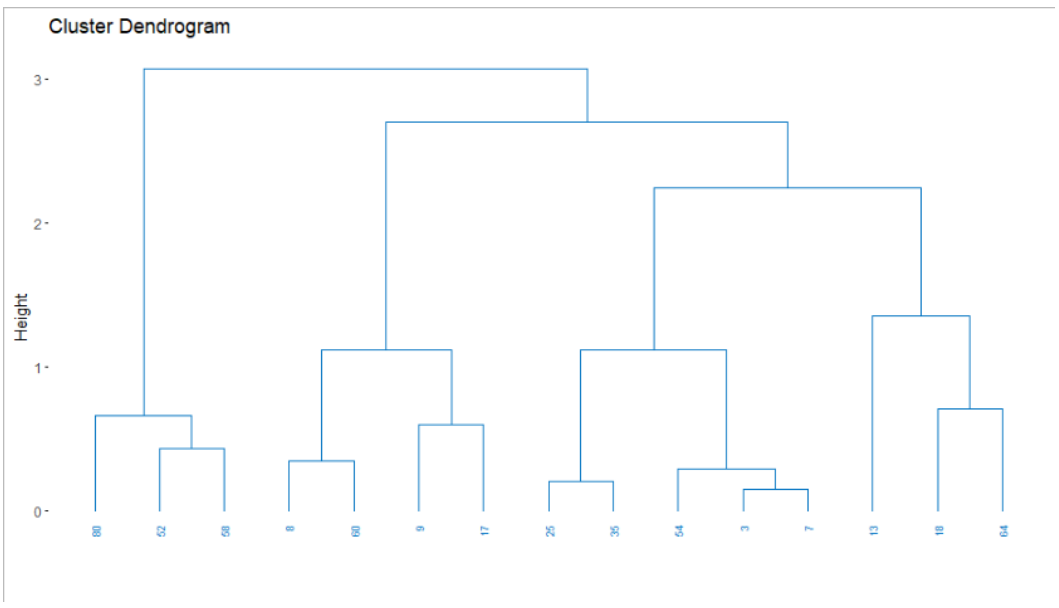
which, as shown below, is useless.



To plot the branch that we're *actually* looking for, we type the following:

```
> fviz_dend(dendCuts$lower[[1]])
```

The blue branch is shown in greater detail, as intended.



We may also visualize clusters in a scatterplot fashion. We first use the *cutree* command (*dendextend* package) to create a four-branch scheme, *grp*, akin to the one that we've obtained earlier.

```
> grp <- cutree(hc.physed, k=4)
```

Then, we appeal to the *fviz_cluster* command.

```
 > fviz_cluster(list(data = physedMod, cluster = grp),
palette=c("#2E9FDF","#00AFBB","#E7B800","#FC4E07"),
ellipse.type = "convex", repel = TRUE, show.clust.cent =
FALSE, ggtheme = theme_minimal())
```

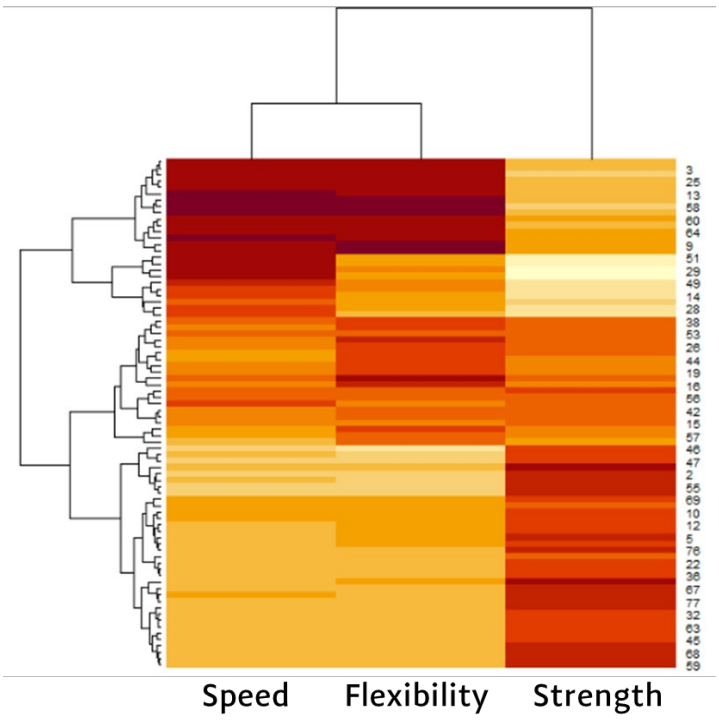
The output is shown below. Note that *palette* defines the colors to be used; *ellipse.type* defines the concentration ellipse; *repel* = *TRUE* ensures that no point label superposes another, albeit at a cost of reduced speed; finally, *show.clust.cent* = *FALSE* prevents the program from showing the cluster centroids. The output is shown below.

Yet another way to visualize hierarchical clustering schemes is to use heatmaps. For this purpose, the default R package *stats* contains the function *heatmap*; the *gplots* package contains the function *heatmap.2*; the *pheatmap* package contains the eponymous function *pheatmap*; and so forth. Most of these commands require the data to be scaled first, so we begin by typing:

```
> scaled_physed <- scale(physedMod)
```

Then, we can try *heatmap*:

```
> heatmap(scaled_physed, scale = "none")
```



In the default color scheme, red indicates high values and yellow indicates low values.

■ **Note**
Before we proceed, note that a typical hierarchical cluster analysis procedure such as the one we've implemented here may involve three steps, namely (1) scaling data with the *scale* command (which we skipped at first because there were no unit discrepancies in our dataset); (2) establishing distances with the *dist* command; and (3) performing the desired cluster analysis through *hclust()*. Fortunately, this process can be made quicker if we resort to the *cluster* R package, which offers the *agnes()* and *diana()* commands. The former is used for agglomerative clustering, whereas the latter is used for divisive clustering. An *agnes()* or *diana()* call performs all necessary hierarchical clustering operations, and there is no need for additional steps; the user can immediately move on to data visualization and post-processing.

▶ Problem 6 – Selecting an algorithm

We have now introduced considered 3 types of clustering analysis – partitioning *k*-means, PAM, and hierarchical. One question that lingers, then, is which algorithm to use in a given situation. The *clValid* command in the eponymous package can be used for this purpose. The basic syntax is:

```
clValid(obj, numClust, clMethods = "hierarchical", validation
= "stability", maxitems = 600, metric = "euclidean", method =
"average")
```

The arguments are:

→ *obj:* A numeric matrix or data frame.
→ *numClust:* The range of number of clusters to be considered (e.g., type 2:10 if you want *clValid* to perform the analysis for 2 to 10 clusters).
→ *clMethods:* The methods to be used by *clValid*.
→ *validation:* The type of validation measures to be used. The available options are "internal," "stability," and "biological."
→ *maxitems:* The maximum number of items (rows) that can occupy a single cluster.
→ *metric:* The metric used to determine the distance matrix. Available options are "euclidean," "correlation," and "manhattan."
→ *method:* Applicable to hierarchical clustering (*hclust* and *agnes*), defines the agglomeration method to be used. Available choices are "ward," "single," "complete," and "average."

Let us try out this command in the context of our *physeduc* dataset. Following Kassambara (2017), we first standardize the data (in case you haven't already done so in an earlier section):

```
> scaled_physed <- scale(physedMod)
```

We proceed to specify an object containing the three clustering algorithms we are considering:

```
> algList <- c("hierarchical","kmeans","pam")
```

Then, we apply *clValid* and summarize the model:

```
> clusterAlgs <- clValid(scaled_physed, nClust = 2:8,
clMethods = algList, validation = "internal")

> summary(clusterAlgs)
```

The command above outputs the following optimal scores:

```
Optimal Scores:
             Score  Method       Clusters
Connectivity 0.4000 hierarchical 2
Dunn         0.4206 hierarchical 2
Silhouette   0.6886 hierarchical 4
```

As can be seen, the 3 internal coefficients are optimized for a hierarchical approach. On the other hand, the recommended number of clusters is not unanimous, in that two coefficients, *connectivity* and *Dunn*, become optimal for a number of clusters equal to 2, whereas one coefficient, *silhouette*, becomes optimal for a number of clusters equal to 4.

We could just as well have based our selection on stability measures. To obtain them, all we have to do is set *validation* to *stability*, and then access the results with *summary*:

```
> clusterAlgsStab <- clValid(scaled_physed, nClust = 2:8,
clMethods = algList, validation = "stability")

> summary(clusterAlgStab)

Optimal Scores:

Score      Method       Clusters
APN 0.0079 hierarchical 4
AD  0.4596 pam          8
ADM 0.0230 hierarchical 4
FOM 0.2638 hierarchical 8
```

As can be seen, hierarchical approaches are recommended for 3 of the 4 measures; the recommended number of clusters is 4 or 8. Combining the recommendations of this *clValid* call with the previous one, the best approach to tackle the *physed* dataset is to use a hierarchical clustering algorithm; the optimum number of clusters, in turn, can be 2, 4, or 8.

REFERENCES

- KASSAMBARA, A. (2017). *A Practical Guide to Cluster Analysis in R*. STHDA.
- SUZUKI, J. (2020). *Statistical Learning with Math and R*. Springer.