Summary: Unsupervised learning in R

Hannes Geldof

23 October 2018

0. Pre-process the data

As with every piece of data analysis, we need to pre-process the data in an adequate way. The following three actions are important before we are able to start clustering!

Action 1: Load and inspect the dataset

This is pretty standard.. Get a feel for the data.

```
str(myData) # View data structure
```

```
## 'data.frame':    800 obs. of  13 variables:
##  $ Number        : int  1 2 3 3 4 5 6 6 6 7 ...
##  $ Name          : Factor w/ 800 levels "Abomasnow","AbomasnowMega Abomasnow",..: 81 330 746 747 103
##  $ Type1         : Factor w/ 18 levels "Bug","Dark","Dragon",..: 10 10 10 10 7 7 7 7 7 18 ...
##  $ Type2         : Factor w/ 19 levels "","Bug","Dark",..: 15 15 15 15 1 1 9 4 9 1 ...
##  $ Total         : int  318 405 525 625 309 405 534 634 634 314 ...
##  $ HitPoints     : int  45 60 80 80 39 58 78 78 78 44 ...
##  $ Attack        : int  49 62 82 100 52 64 84 130 104 48 ...
##  $ Defense       : int  49 63 83 123 43 58 78 111 78 65 ...
##  $ SpecialAttack : int  65 80 100 122 60 80 109 130 159 50 ...
##  $ SpecialDefense: int  65 80 100 120 50 65 85 85 115 64 ...
##  $ Speed         : int  45 60 80 80 65 80 100 100 100 43 ...
##  $ Generation    : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ Legendary     : Factor w/ 2 levels "False","True": 1 1 1 1 1 1 1 1 1 1 ...
```

Action 2: Choose variables used for clustering

Browse for the relevant variables for clustering. Here, these are the features of the pokemons (e.g. speed, attack).

```
names(myData) # View data variables
```

```
##  [1] "Number"         "Name"           "Type1"          "Type2"
##  [5] "Total"          "HitPoints"      "Attack"         "Defense"
##  [9] "SpecialAttack"  "SpecialDefense" "Speed"          "Generation"
## [13] "Legendary"
```

Make sure to only keep the relevant variables for clustering. Here, these are included in columns 6 to 11.

```
pokemon <- myData[,c(6:11)] # Create a subset of the data
head(pokemon)
```

```
##   HitPoints Attack Defense SpecialAttack SpecialDefense Speed
## 1        45     49      49            65             65    45
## 2        60     62      63            80             80    60
## 3        80     82      83           100            100    80
## 4        80    100     123           122            120    80
## 5        39     52      43            60             50    65
## 6        58     64      58            80             65    80
```

Action 3: Scale variables before clustering

It is important to scale variables before conducting a clustering analysis if they have different distributions. This is intuitive in case your variables are of incomparable units (e.g. height in cm and weight in kg). Even if variables are of the same units but show quite different variances it is still a good idea to standardize variables.

To illustrate this, let's look at the distribution (mean and standard deviation) of each feature to see if scaling the data is necessary.

```
colMeans(pokemon) # View column means
```

```
##      HitPoints         Attack        Defense  SpecialAttack SpecialDefense
##       69.25875       79.00125       73.84250       72.82000       71.90250
##          Speed
##       68.27750
```

```
apply(pokemon,2,sd) # View column standard deviations
```

```
##      HitPoints         Attack        Defense  SpecialAttack SpecialDefense
##       25.53467       32.45737       31.18350       32.72229       27.82892
##          Speed
##       29.06047
```

R has an in-house command to standardize your data. Convenient, right?

```
pokemonScaled <- scale(pokemon) # Standardise the data by performing a z transformation
```

Note: As a general rule, you can put '?command' in the console to get directed to the documentation of the command.

1. Basics

1.1 K-means clustering

Fit a k-means model to this reduced dataset using 3 centers. Make sure to run the k-means algorithm 20 times and set an upper bound to 50 iterations.

Note: kmeans() is an iterative algorithm, repeating over and over until some stopping criterion is reached. The default number of iterations for kmeans() is 10, which is sometimes not enough for the algorithm to converge and reach its stopping criterion, so we'll set the number of iterations to 50 to overcome this issue.

Basics: Create a k-means model using 3 clusters, and print the output

```r
km <- kmeans(pokemonScaled, centers = 3, nstart = 20, iter.max = 50)
km
```

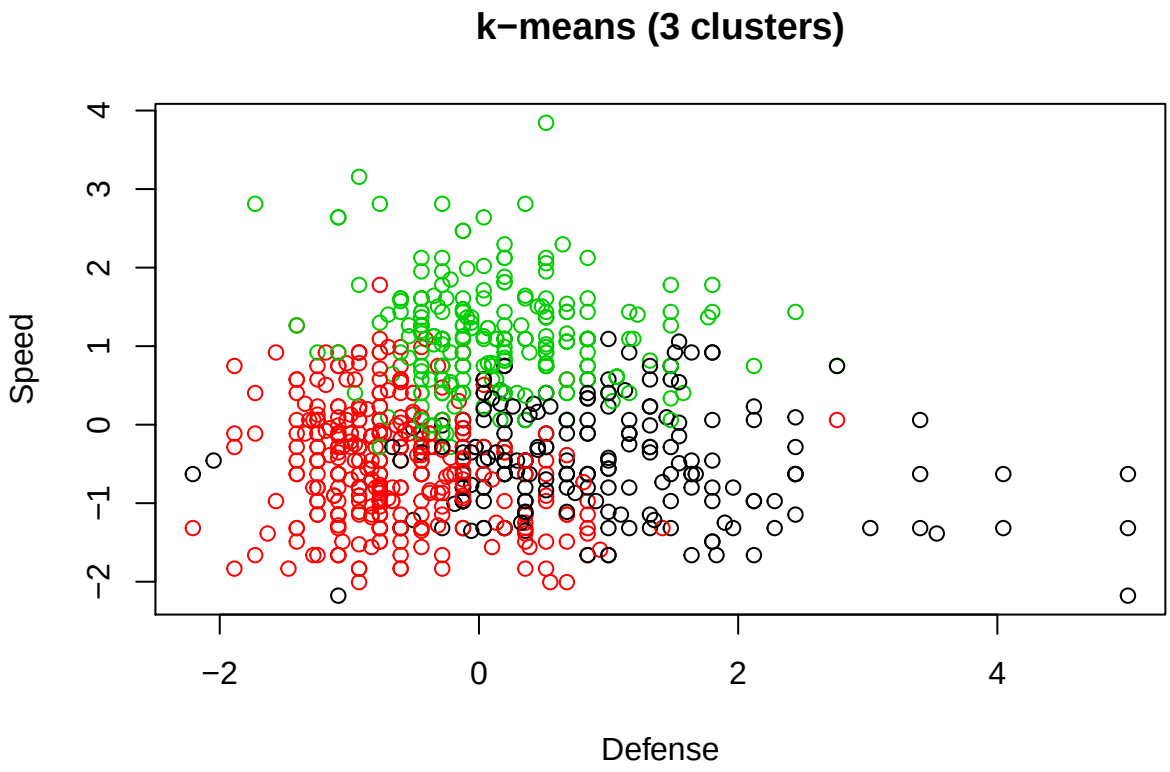
```
## K-means clustering with 3 clusters of sizes 214, 343, 243
##
## Cluster means:
##    HitPoints     Attack    Defense SpecialAttack SpecialDefense      Speed
## 1  0.6032896  0.3457782  0.8681959   0.005358032      0.5720643 -0.4538369
## 2 -0.6669891 -0.6910549 -0.6478138  -0.634570617     -0.7152782 -0.4900992
## 3  0.4101780  0.6709271  0.1498197   0.890992193      0.5058380  1.0914614
##
## Clustering vector:
##   [1] 2 2 3 3 2 2 3 3 3 2 2 1 3 2 2 2 2 2 2 3 2 2 3 3 2 2 2 3 2 3 2 3 2 1 2
##  [36] 2 1 2 2 3 2 1 2 3 2 1 2 3 2 2 1 2 1 2 3 2 2 2 3 2 3 2 3 2 3 2 2 1 2 2
##  [71] 3 3 2 1 1 2 2 3 2 3 2 1 1 2 3 2 1 1 2 3 2 2 3 2 1 2 1 2 1 2 2 3 3 2 2
## [106] 1 2 1 2 3 2 1 2 1 3 1 1 2 1 2 1 1 1 1 3 2 2 2 1 2 3 3 3 3 3 3 1 3 3 2
## [141] 1 1 1 2 2 1 3 3 2 2 1 2 1 3 3 1 3 3 3 2 2 3 3 3 3 3 2 2 1 2 2 3 2 2 1
## [176] 2 2 2 1 2 2 2 2 3 2 1 2 2 2 2 1 2 3 2 2 1 1 1 2 1 1 1 2 2 3 2 2 1 2 2
## [211] 1 3 1 2 1 3 2 1 3 2 1 1 1 1 1 2 1 2 1 1 1 1 1 3 2 1 2 1 2 1 2 2 1 2 1
## [246] 1 2 3 3 3 2 1 1 3 2 2 1 2 2 2 1 1 3 3 1 2 1 1 1 3 3 3 2 2 3 3 2 2 3 3
## [281] 2 1 1 1 2 2 2 2 2 2 2 2 2 2 2 1 2 2 3 2 3 2 1 2 2 3 3 2 2 2 1 2 1 3 2
## [316] 3 2 2 2 3 2 1 2 1 2 2 2 1 2 1 2 1 1 1 2 2 3 2 3 3 2 2 2 2 2 2 1 2 3 3
## [351] 2 1 2 1 1 1 2 3 2 2 2 3 2 3 2 1 3 3 3 3 1 2 1 2 1 2 1 2 1 2 1 2 3 2 1
## [386] 2 3 3 2 1 1 2 3 3 2 2 3 3 2 1 1 2 1 1 1 2 2 1 3 3 2 1 1 3 1 1 1 3 3 3
## [421] 3 3 3 3 3 3 3 3 3 3 1 3 2 1 1 2 2 3 2 2 1 2 2 3 2 2 2 2 2 2 2 3 2 3 2 1
## [456] 2 1 2 1 1 1 2 2 1 2 2 3 2 3 2 1 3 2 3 2 3 3 3 3 2 3 2 2 3 2 1 2 2 2 2
## [491] 1 2 2 3 3 1 2 3 3 2 1 2 1 2 3 1 2 3 2 2 1 1 3 1 1 1 1 3 3 3 3 1 1 1 3
## [526] 3 3 3 1 1 3 3 3 3 3 3 3 1 3 3 3 3 3 3 1 3 1 3 3 3 3 3 3 3 2 2 3 2 1 3
## [561] 2 2 3 2 2 2 2 1 2 3 2 3 2 3 2 3 2 1 2 2 3 2 3 2 1 1 2 3 2 3 1 1 2 1 1
## [596] 2 2 1 1 1 2 2 3 2 2 3 2 3 2 3 3 2 2 3 2 3 1 3 2 1 2 1 3 2 1 2 1 2 3 2
## [631] 1 2 3 2 3 2 2 1 2 2 1 2 3 2 2 3 2 3 3 2 1 2 1 2 1 1 2 3 2 1 2 1 1 2 1
## [666] 1 2 1 2 2 3 2 2 3 2 1 3 2 3 1 2 3 1 2 1 2 1 1 2 1 2 1 3 3 2 1 3 2 3 3
## [701] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 2 2 1 2 2 3 2 2 3 2 1 2 2 3 2 2 2
## [736] 2 3 2 2 3 2 3 2 1 3 2 3 3 2 1 3 1 2 1 2 1 2 1 2 1 2 1 2 1 2 3 2 1 2 1
## [771] 1 3 3 1 2 1 3 1 2 1 2 2 2 2 1 1 1 1 2 1 2 3 3 3 1 1 3 3 3 3
##
## Within cluster sum of squares by cluster:
## [1] 1106.2971  802.1416  950.0416
##  (between_SS / total_SS =  40.4 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"
## [5] "tot.withinss" "betweenss"    "size"         "iter"
## [9] "ifault"
```

Visualizing and interpreting results: create a 2D scatter plot

Create a 2D scatter plot (Defense vs. Speed) together with the cluster membership for each observation.

To create a scatter plot, you can pass data from two features (identified by the columns of the dataframe) to the plot() command. You will need an extra argument col = km.out$cluster, which assigns a color to each point in the scatter plot according to its cluster membership.

```r
plot(pokemonScaled[, c("Defense", "Speed")],
     col = km$cluster,
     main = "k-means (3 clusters)",
     xlab = "Defense", ylab = "Speed")
```



Intuition: Handling random algorithms

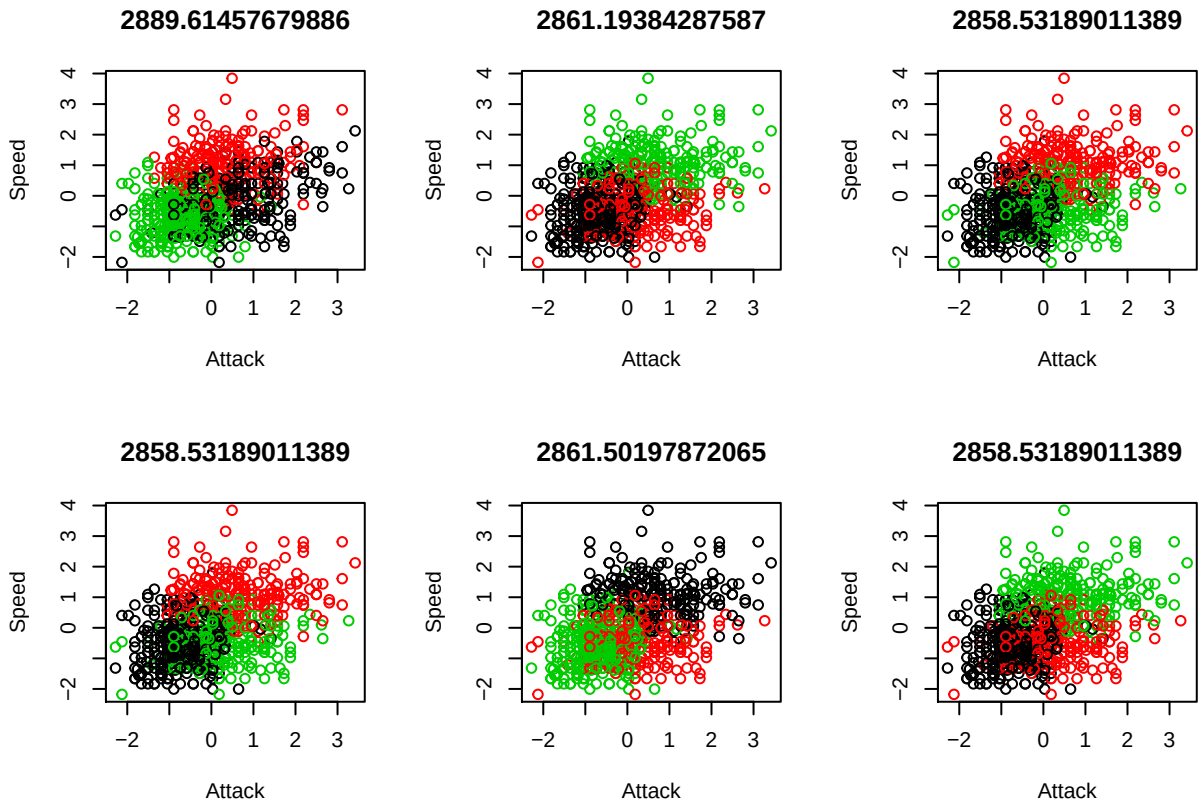
K-means() randomly initializes the centers of clusters. This random initialization can result in assigning observations to different cluster labels. Also, the random initialization can result in finding different local minima for the k-means algorithm.

To illustrate this, we will plot a metric of model quality, namely the total within cluster sum of squares. Look for the model(s) with the lowest error to find models with the better model results.

```r
par(mfrow = c(2, 3)) # Set up 2 x 3 plotting grid

set.seed(1) # Set seed to obtain reproducible results

for(i in 1:6) {  # Run kmeans() on the scaled data with three clusters and one start, and plot clusters
  km2 <- kmeans(pokemonScaled, centers = 3, nstart = 1)
  plot(pokemonScaled[, c("Attack", "Speed")],
       col = km2$cluster,
       main = km2$tot.withinss)
}
```



Interesting! Because of the random initialization of the k-means algorithm, there is some slight variation in cluster assignments of observations among the six models.

1.2 Hierarchical clustering

Basics: Create a hierarchical clustering model

The first step to hierarchical clustering is to determine the similarity between observations, which you will do with the dist() function. This will create a distance matrix which will allow us to compare observations across multiple dimensions.

We use the **Euclidean distance** as the measure of distance to assess similarity between observations.

```
distPokemonScaled <- dist(pokemonScaled, method = "euclidian")
```

Then, we fit our hierachical clustering model.

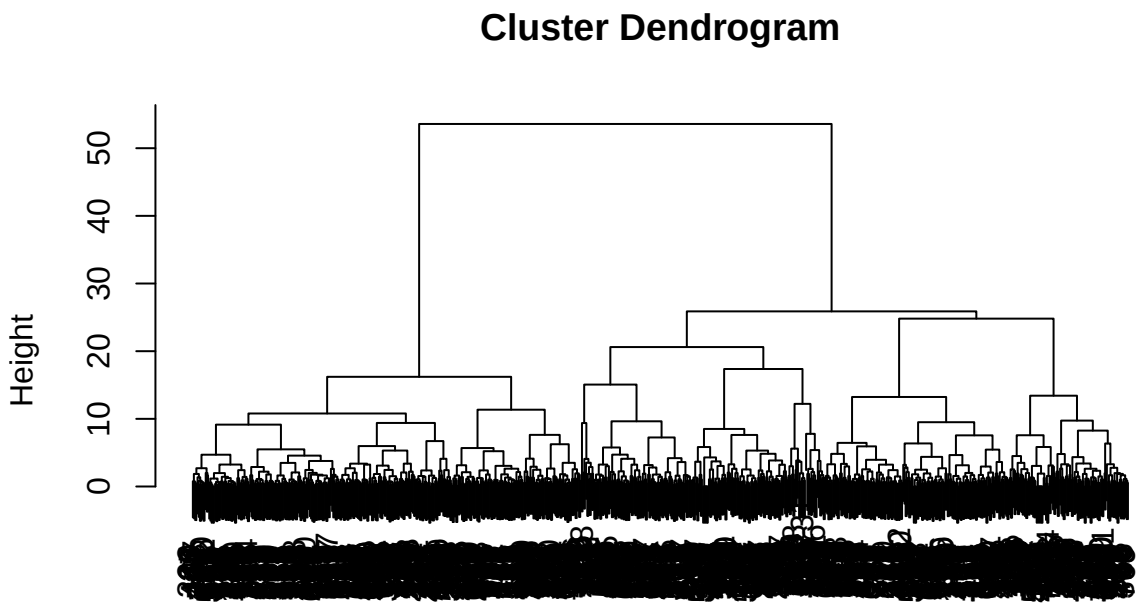
We use the popular **Ward.D2 method** for hierarchical clustering. This method is less susceptible to noise and outliers.

```
hc <- hclust(distPokemonScaled, method = "ward.D2")
```

Visualizing and interpreting results: create a dendrogram

Hierarchical clustering organizes objects into a dendrogram of which the branches are the desired clusters. The process of cluster detection is referred to as "cutting the tree" at a a certain height. Below, we build the dendrogram / tree of the hierarchical clustering model.

```
plot(hc)
```



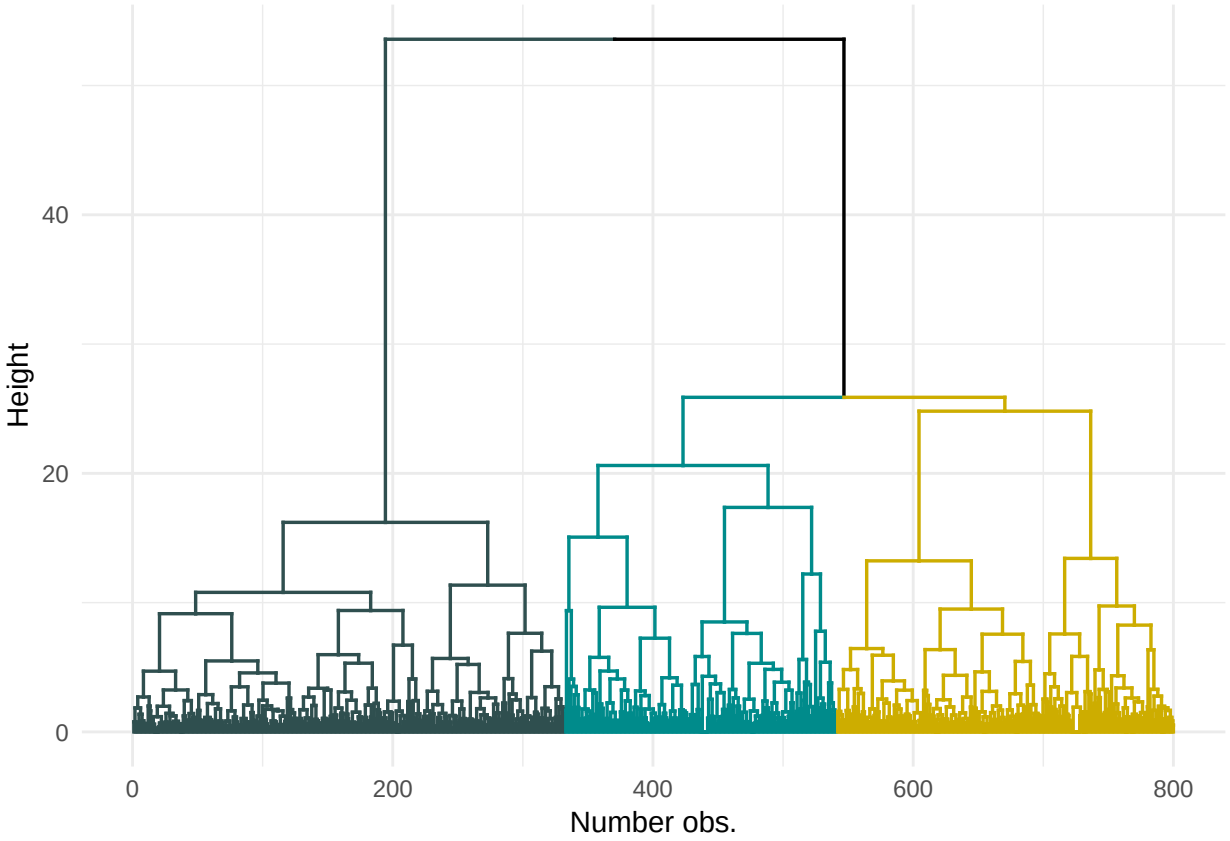
Note: Judging from the tree above, it looks like there are either two or tree clusters. Here, we will decide to 'cut' the tree at 3 clusters.

```
hcCut <- cutree(hc, k = 3)
```

Here's an example of the power of R in terms of visualisations - almost everything can be customised. We will highlight the three clusters by assigning a separate colour to them.

```
dendro <- as.dendrogram(hc)
dendroCol <- dendro %>%
  set("branches_k_color", k = 3, value =   c("darkslategray", "darkcyan", "gold3")) %>%
  set("branches_lwd", 0.6) %>%
  set("labels_cex", 0.5)
ggd <- as.ggdend(dendroCol)
ggplot(ggd, labels = F, theme = theme_minimal()) +
  labs(x = "Number obs.", y = "Height")
```

```
## Warning: Removed 1599 rows containing missing values (geom_point).
```



Intuition: Compare the outcomes generated under k-means and hierarchical clustering

```
table(hcCut, km$cluster)
```

```
##
## hcCut   1   2   3
##     1  26 305   1
##     2 154   3  52
##     3  34  35 190
```

Looking at the table, it looks like the hierarchical clustering model and the k-means model distribute the observations relatively evenly among all clusters. Both models have one big cluster and two smaller clusters of similar size.

Note: It's important to note that there's ***no consensus*** *on which method produces better clusters. The job of the analyst in unsupervised clustering is to observe the cluster assignments and make a judgment call as to which method provides more insights into the data.*

2. Advanced methods

2.1. What is the optimal number of clusters?

Conceptually speaking, clustering analysis is all about:

- **Intra-cluster compactness**: you want the distance between data points within clusters to be minimal.
- **Inter-cluster separation**: you want the distance between clusters to be as large as possible.

Step 0: 'It is not a blind exercise'

It is worth keeping in mind that this exercise remains to be subject to important judgement calls, based on insights in the data. However, the following tools exist that may help you inform your decisions along the way.

These can be used both for k-means and hierarchical clustering. In what follows, we use hierarchical clustering as an example.

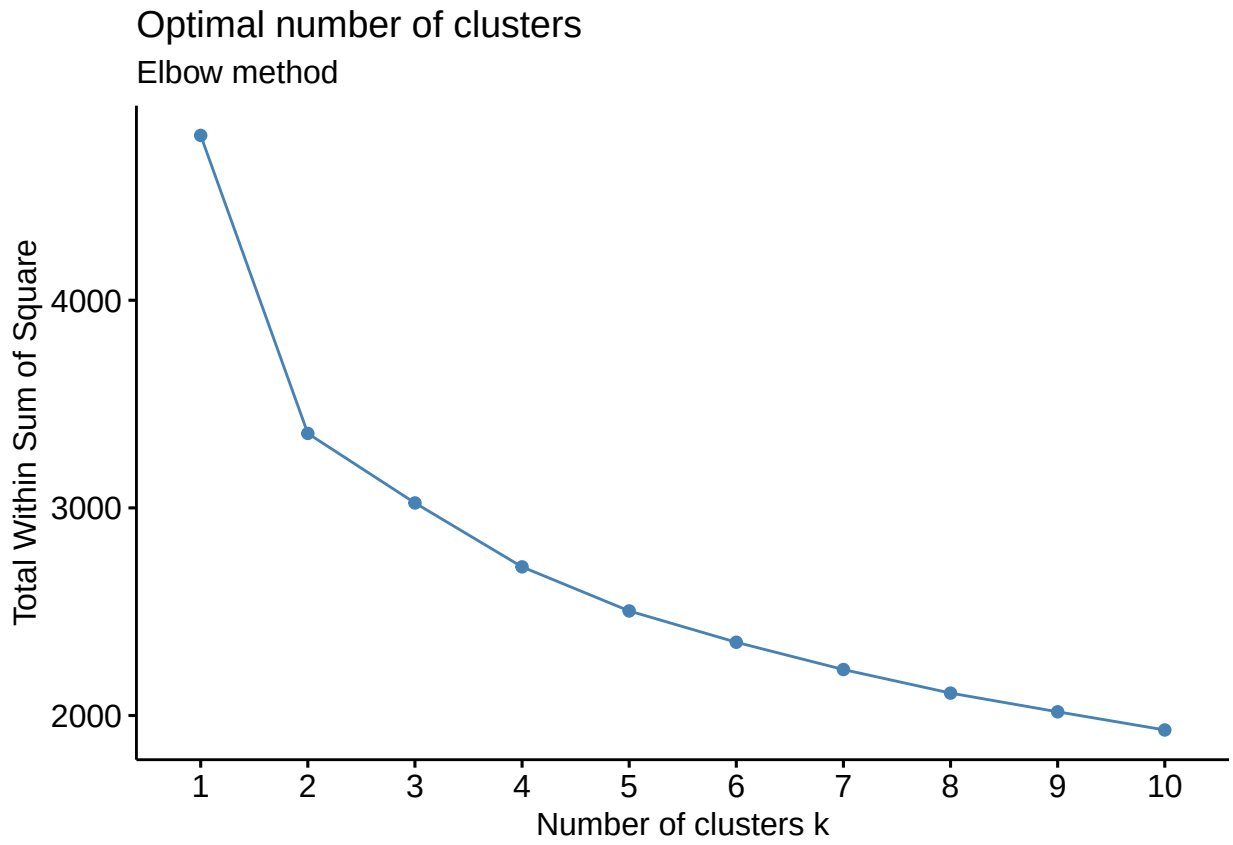
Step 1: 'Elbow plot'

Most importantly, we care about intra-cluster compactness. This will maximise the similarities between observations within clusters.

The elbow plot looks at the closeness between observations in a cluster (measured by the 'total within-cluster sum of squares') and looks how this changes depending on the number of clusters. The optimal number of clusters is then identified as the elbow of the plot, where adding another cluster does not improve the measure of closeness much better.

```
#install.packages("factoextra")
suppressPackageStartupMessages(library(factoextra))
```

```
fviz_nbclust(pokemonScaled, hcut, method = "wss", hc_method = "ward.D2") +
 labs(subtitle = "Elbow method")
```

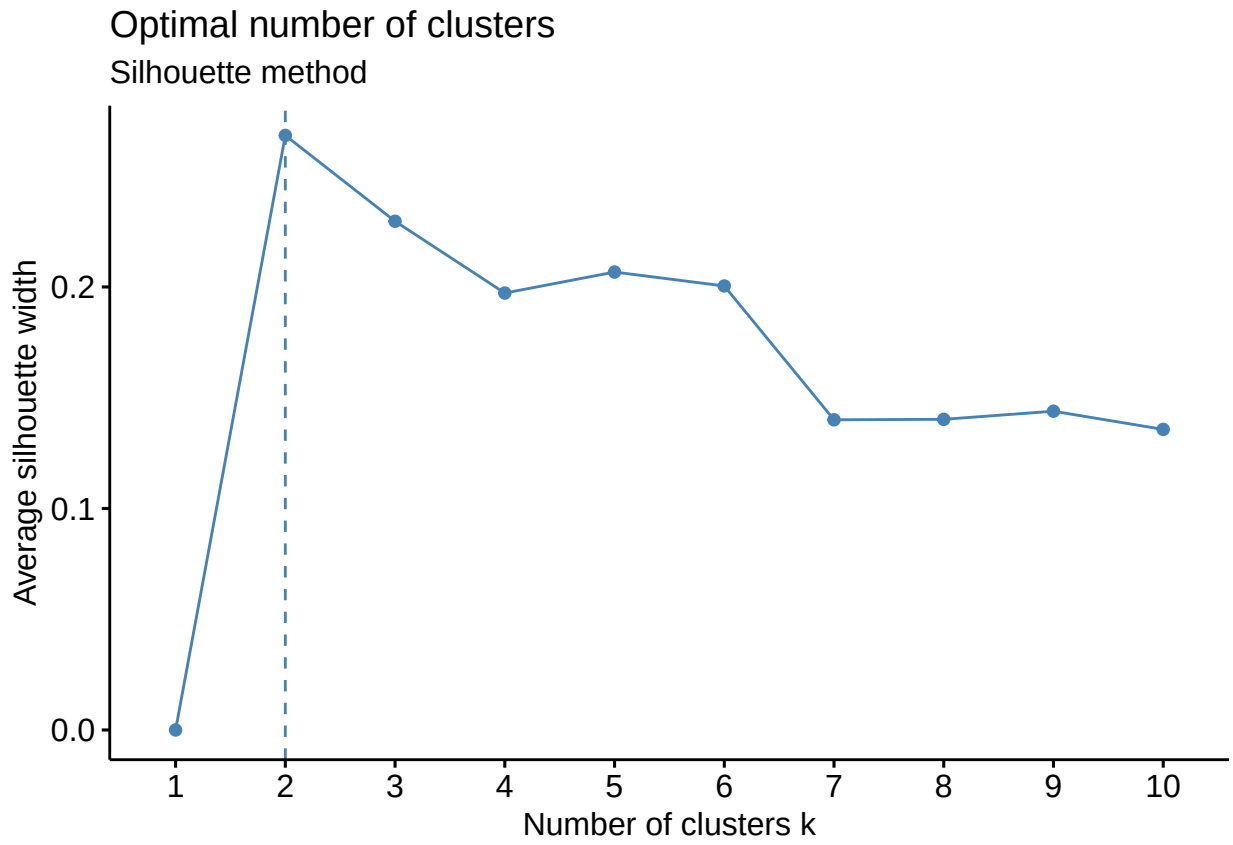


Note: The figure shows how the within sum of squares changes with the number of clusters. The lower it is, the closer the observations within the clusters are. Ideally, we should see a distinctive "bend" in the elbow where splitting clusters further only gives a minor decrease in the metric of closeness.

Step 2: 'Average silhouette plot'

The elbow method can sometimes deliver ambiguous results. It can be complemented by the average silhouette plot. This tool also takes into account inter-cluster separation, alongside intra-cluster compactness.

```
fviz_nbclust(pokemonScaled,hcut, method = "silhouette", hc_method = "ward.D2") + labs(subtitle = "Silhou
```



Note: Silhouette analysis measures how well an observation is clustered and it estimates the average distance between clusters.The silhouette plot displays a measure of how close each point in one cluster is compared to points in the neighboring clusters. A high average silhouette width indicates a good clustering.

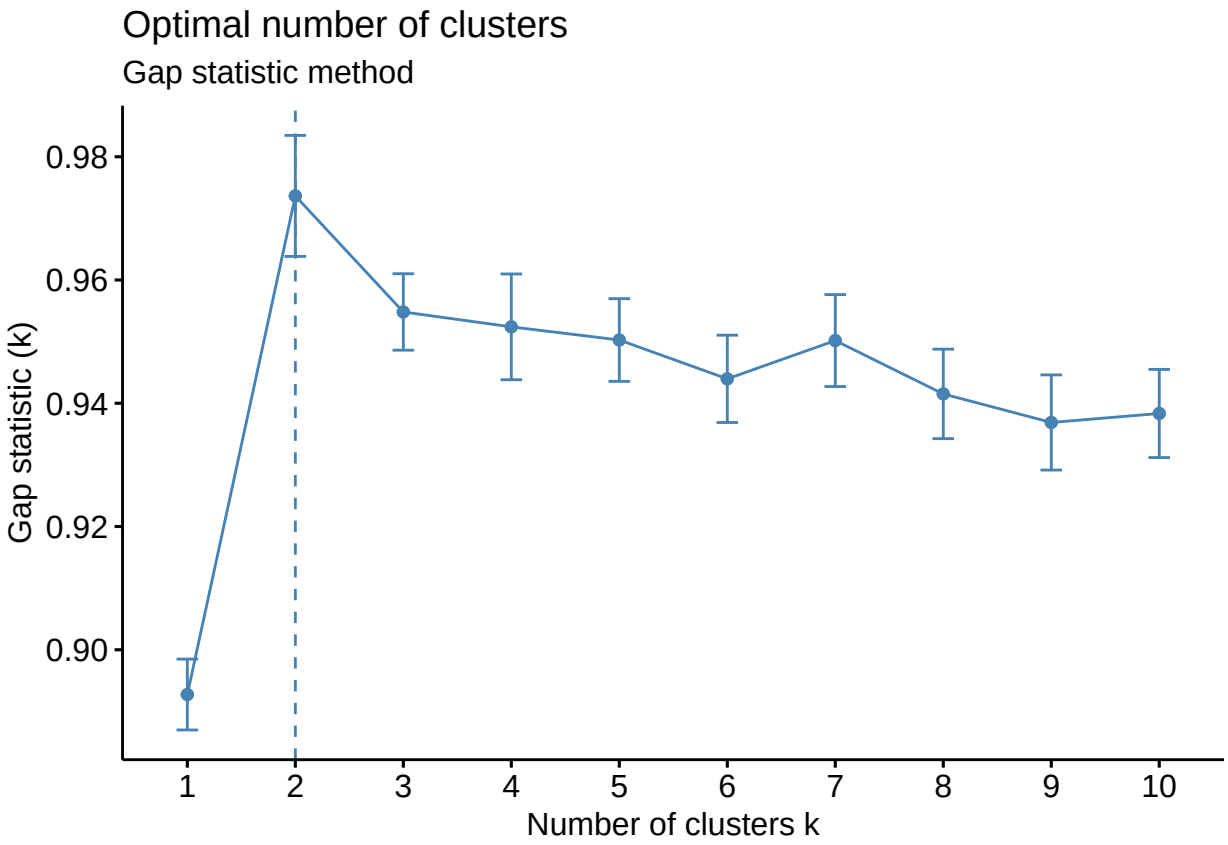
Step 3: 'Gap statistic method

The disadvantage of the two previous methods is that they measure a global clustering characteristic. A more sophisticated method is the gap statistic.

The gap statistic compares the total within intra-cluster variation for different numbers of clusters with their expected values under a random uniform distribution of the data.

```
#install.packages("cluster")
library(cluster)
```

```
set.seed(123)
gap_stat <- clusGap(pokemonScaled, hcut, K.max = 10, B = 20)
fviz_gap_stat(gap_stat) + labs(subtitle = "Gap statistic method")
```



Note: The estimate of the optimal number of clusters will be the value that maximizes the gap statistic. This means that the clustering structure is far away from the random uniform distribution of points.

Step 4: Pseudo F-index

Another methodology that is found in the literature is the pseudo F-index. This index balances parsimony (i.e. a small number of groups) with the ability to discriminate (i.e. groups have sufficiently distinct characteristics from each other).

It increases when observations are more alike within a group, but more distinct across groups. It does however impose a penalty as the number of groups get larger in the similar fashion of the Akaike and Schwarz information criteria. Those criteria are used in modelling selection to select the appropriate number of lags in time series regressions.

How can we code this method? Try it.

Conclusion:

- **Elbow method**: 2 / 3 clusters suggested
- **Silhouette method**: 2 clusters suggested
- **Gap statistic method**: 2 clusters suggested

This seems to indicate that 2 clusters is most appropriate. Nevertheless, make sure to use these tools critically, and complement those with **judgement calls**, based on insights in the data.

2.2. Hierarchical clustering: alternative methods?

How much 'overlap' is there between clusters found using different hierarchical clustering methods?

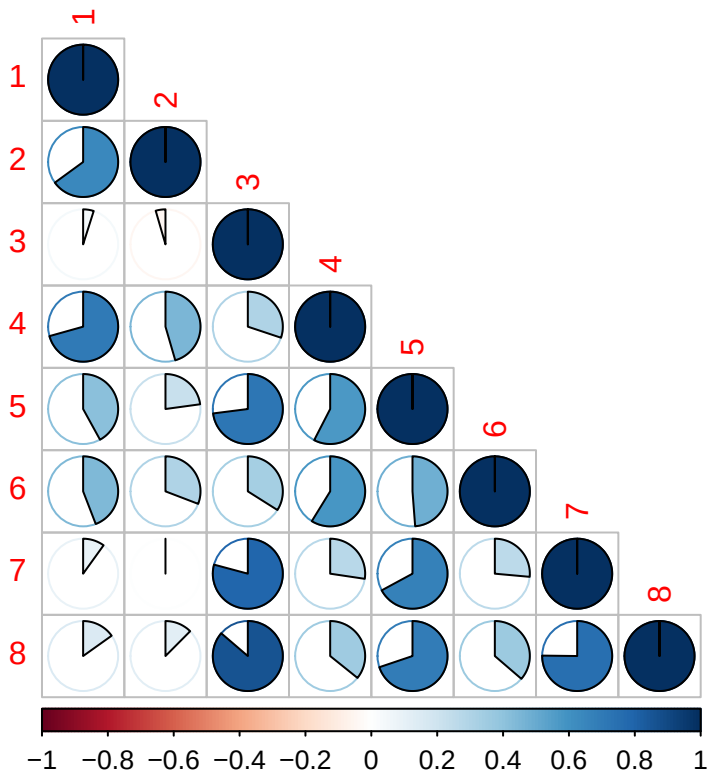
One can check this by assessing the statistical similarity of clustering outcomes achieved by other methods based on the correlation between the distance matrices of two trees.

```
#install.packages("corrplot")
suppressPackageStartupMessages(library(corrplot))
```

```
hclustMethods <- c("ward.D2", "ward.D", "single", "complete", "average","mcquitty", "median", "centroid
options(expressions = 10000)
hcList <- dendlist()

for(i in seq_along(hclustMethods)) {
  hcAlt <- hclust(distPokemonScaled, method = hclustMethods[i])
  hcList <- dendlist(hcList, as.dendrogram(hcAlt))
}

hcListCorr <- cor.dendlist(hcList)
corrplot(hcListCorr, "pie", "lower")
```



As can be seen in the figure, different clustering methods yield rather different results, indicated by some low correlations. There seem to be two groups of methods which produce similar results:

- Group 1: method 1, 2 and 4

- Group 2: method 3, 5, 7 and 8.

Group 1 includes our preferred method **Ward.D2**, which seems to compare best with two other methods, namely **Ward.D** and **Complete linkage**. This may give us additional comfort in our approach.

Group 2 includes *inter alia* the **Single linkage**, which is rather sensitive to noise and outliers.

Note: 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. 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.

2.3. Hierarchical clustering: can I improve my clustering results using Principal Component Analysis?

Introduction: PCA

Principal Component Analysis ('PCA') is a dimensionality-reduction technique that is often used to transform a high-dimensional dataset into a smaller-dimensional subspace prior to running a machine learning algorithm on the data.

Principal Component Analysis does just what it advertises: it finds the principal components of the dataset. PCA transforms the data into a new, lower-dimensional subspace. In the new coordinate system, the first axis corresponds to the first principal component, which is the component that explains the *greatest amount of the variance* in the data.

When should you use PCA?

It is often helpful to use a dimensionality-reduction technique such as PCA prior to performing machine learning because:

- Reducing the dimensionality of the dataset reduces the size of the space on which 'distance-based algorithms' (e.g. hierarchical clustering) must calculate measures of distance, which improve the performance of those algorithms.
- Reducing the dimensionality of the dataset reduces the number of degrees of freedom of the hypothesis, which reduces the risk of overfitting.
- Most algorithms will run significantly faster if they have fewer dimensions they need to look at.
- Reducing the dimensionality via PCA can simplify the dataset, facilitating description, visualization, and insight.

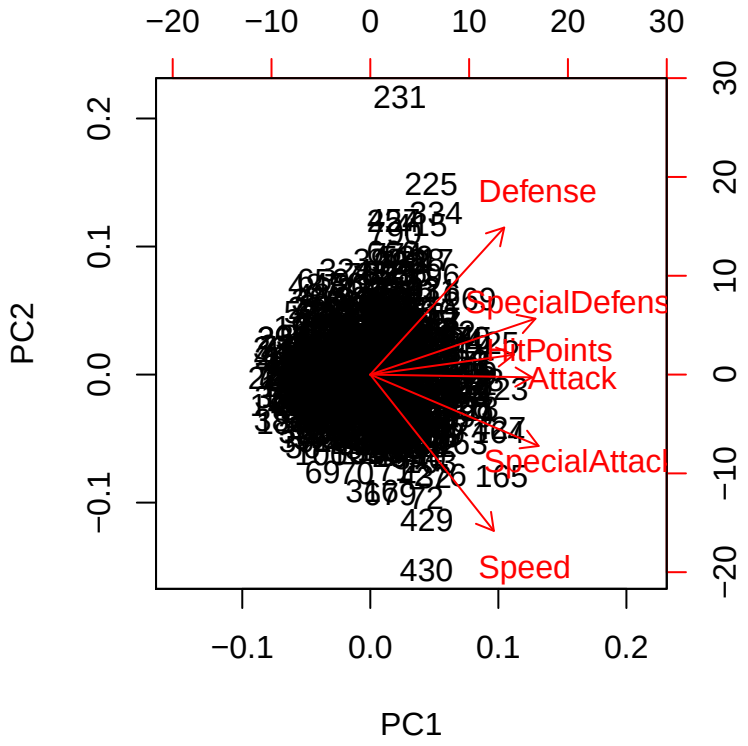
Create a scaled PCA model and inspect output

```
pca <- prcomp(pokemon, scale = TRUE, center = TRUE) # PCA has built-in scaling functionality
summary(pca)
```

```
## Importance of components:
##                           PC1    PC2    PC3    PC4     PC5     PC6
## Standard deviation     1.6466 1.0457 0.8825 0.8489 0.65463 0.51681
## Proportion of Variance 0.4519 0.1822 0.1298 0.1201 0.07142 0.04451
## Cumulative Proportion  0.4519 0.6342 0.7640 0.8841 0.95549 1.00000
```

Visualising and interpreting output

```
biplot(pca)
```



Seeing the chart, the variables HitPoints and Attack have approximately the same loadings in the first two principal components.

Fit a hierarchical clustering model on the PCA output

In addition to normalizing data and potentially avoiding overfitting, PCA also uncorrelates the variables, sometimes improving the performance of other modeling techniques.

```
hcPCA <- hclust(dist(pca$x[,1:6]), method = "ward.D2")
```

Compare results

```
hcPCACut <- cutree(hcPCA,3) # Cut model into 3 clusters
table(hcCut, hcPCACut)
```

```
##       hcPCACut
## hcCut   1   2   3
##     1 312  20   0
##     2   0   0 209
##     3   0 157 102
```

The clustering outcomes across both methods look to be similar, however not identical.

Conclusion

In this specific case, PCA does not add too much value as the six variables we use for clustering is still manageable. However, it is useful to have PCA as part of your toolkit when working with larger datasets.

3. Post cluster analysis

3.1 Summary stats

What are my **outside variables**? These are the variables that have not been used to inform the clustering algorithm - the so-called **input variables** - but may help to investigate the clusters.

```
names(myData) # View data variables
```

```
##  [1] "Number"         "Name"           "Type1"          "Type2"
##  [5] "Total"          "HitPoints"      "Attack"         "Defense"
##  [9] "SpecialAttack"  "SpecialDefense" "Speed"          "Generation"
## [13] "Legendary"
```

In which clusters are the legendary pokemons allocated?

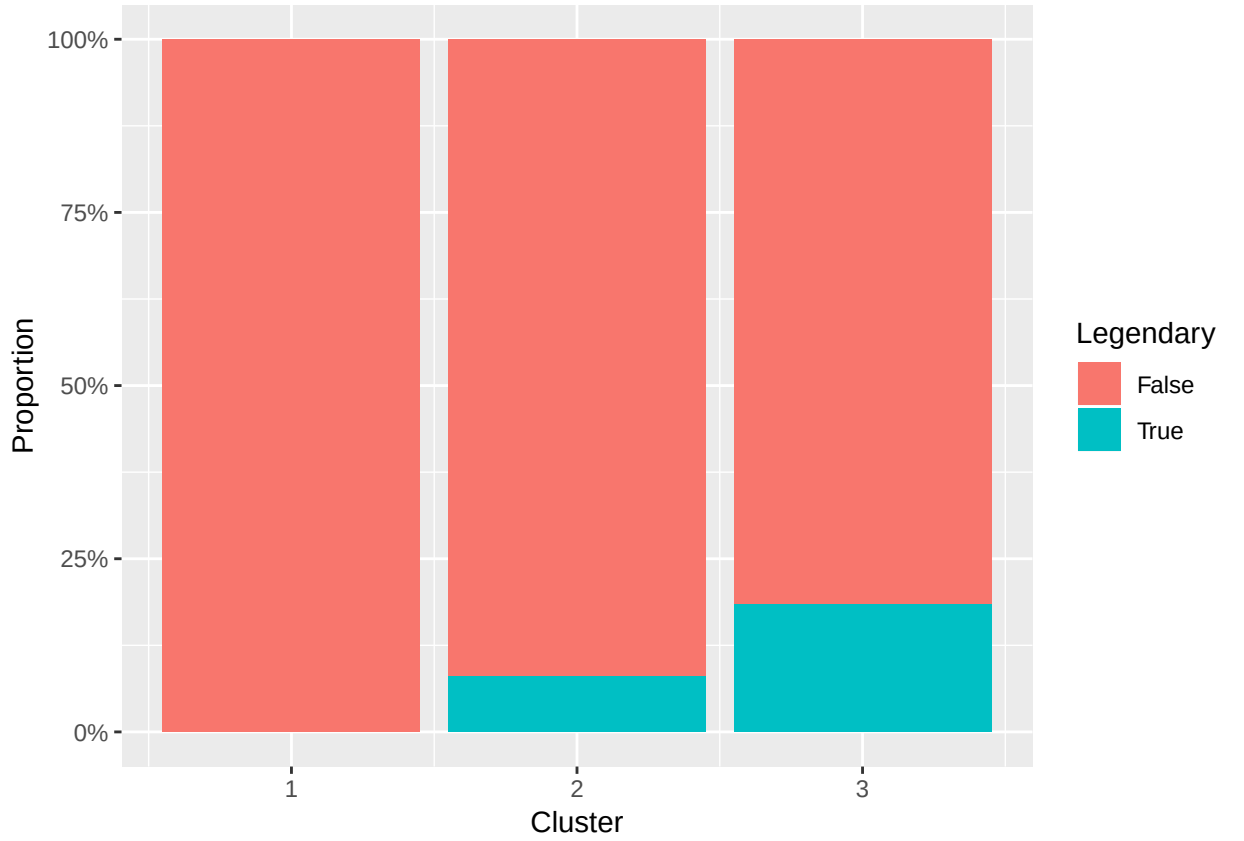
The power of clustering is real..

For context, many people thrive to collect all Pokemons; however, catching all of them is not a simple task, especially the Legendary Pokemon. All Legendary Pokemon are majestic, glorious, and extremely powerful. For more detail, see http://freeaddon.com/ultimate-list-of-legendary-pokemon/.

```
#install.packages("CGPfunctions")
suppressPackageStartupMessages(library(CGPfunctions))
```

```
myDataNew = mutate(myData, Cluster = hcCut)

ggplot(myDataNew,aes(x = Cluster,fill = Legendary)) +
    geom_bar(position = "fill") +
    scale_y_continuous(name = "Proportion", labels = scales::percent)
```



The pattern is quite clear. Cluster 1 does not contain *any* legendary pokemons. Cluster 2 contains some legendary pokemons. Cluster 3 contains most of the legendary pokemons (almost 75%).

As the 'Legendary' variable was not used for clustering, this result illustrates the power of clustering. This seems to indicate that the algorithm clusters pokemons according to their overall power across multiple features. Just as we expected on an ex ante basis.

3.2 Pruned decision trees

Now, let's flip things around! First, we allocated observations to clusters on the basis of their similarity across different variables. Now, let's investigate which variables (and what levels) actually have the largest impact on an observation ending up in a certain cluster.

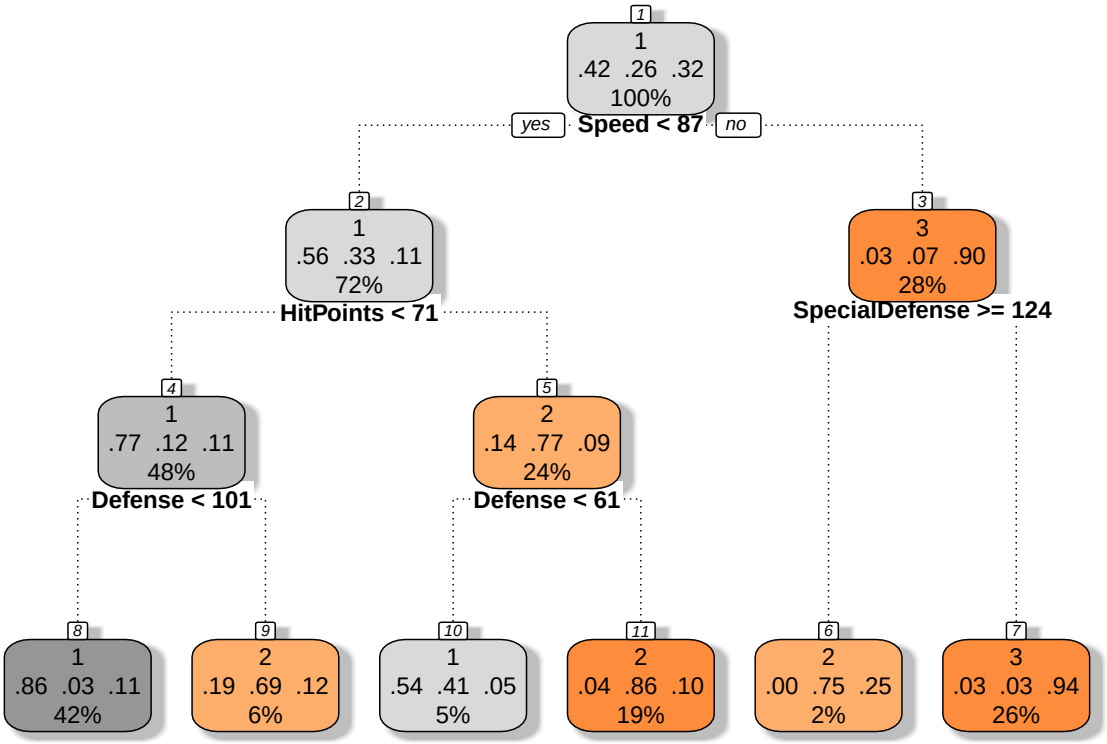
We will make things visual with a decision tree. That means, at each point, we consider a set of questions that can partition the data set. We choose the question that provides the best split and again find the best questions for the partitions. We stop once all the points you are considering are of the same class. Then the task of cluster classication is easy. You can simply grab a point, and chuck it down the tree.

Note: Here, we choose to restrict the level of the tree to only look at the most relevant questions for clustering.

```
#install.packages("rpart")
#install.packages("rattle")
#install.packages("tabplot")
suppressPackageStartupMessages(library(rpart))
suppressPackageStartupMessages(library(rattle))
suppressPackageStartupMessages(library(tabplot))
```

```
ctrl = rpart.control(maxdepth = 3) # Set the tree depth to three levels
DTmodel <- rpart(Cluster ~ HitPoints + Attack + Defense +
                   SpecialAttack + SpecialDefense + Speed,
                 data = myDataNew,
                 method = "class",
                 control = ctrl)

fancyRpartPlot(DTmodel, cex = 0.75, palettes=c("Greys", "Oranges"))
```

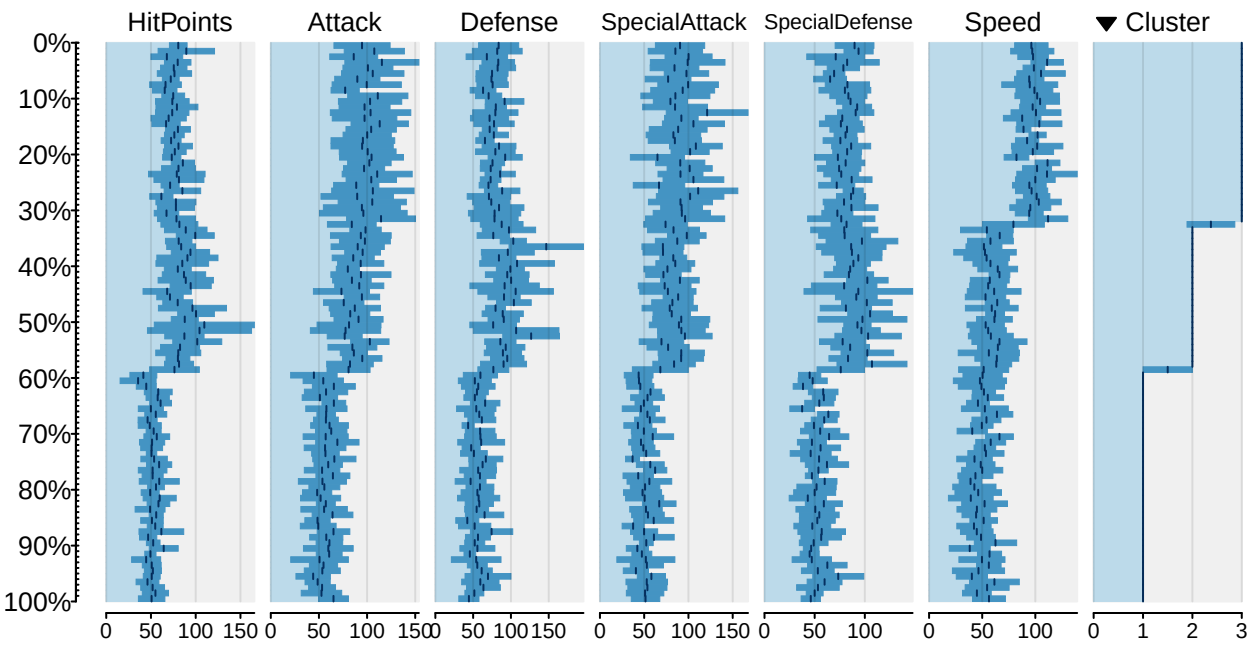


This already gives a first indication that **Speed**, **HitPoints** and **SpecialDefense** are the most important variables for clustering. For instance, Cluster 3 seems to include (legendary) pokemons with high Speed and high SpecialDefense.

3.3 Snake charts

Note: Credits to Luis for coming up with the name 'snake charts'.

```
var <- c("HitPoints","Attack","Defense", "SpecialAttack", "SpecialDefense", "Speed")
tableplot(myDataNew[, c(var, "Cluster")], sortCol = Cluster)
```



row bins: 100

objects:
800
8 (per bin)

Snake charts are able to tell you the whole story:

- Cluster 1: **Your average Joe**: These pokemons seem to be average across the board.
- Cluster 2: **It's all about defense**: These pokemons seem to be strong in defense.
- Cluster 3: **Agile in attack**: These pokemons seem to be very quick and forceful in attack.