clustering

```r
setwd("~/DataVis2/ex2/Méthode 2  Classification ascendante hiérarchique-20231218")
```

Importation des données

Ce code est utilisé pour importer les données.

```r
library(readxl)
autos <- read_excel("autos.xls")
summary(autos)
```

```
##     Modele               CYL           PUISS            LONG
##  Length:18          Min.   :1166   Min.   : 55.00   Min.   :393.0
##  Class :character   1st Qu.:1310   1st Qu.: 70.75   1st Qu.:424.0
##  Mode  :character   Median :1578   Median : 82.00   Median :434.5
##                     Mean   :1632   Mean   : 84.61   Mean   :433.5
##                     3rd Qu.:1798   3rd Qu.: 98.00   3rd Qu.:448.0
##                     Max.   :2664   Max.   :128.00   Max.   :469.0
##       LARG           POIDS          V-MAX         FINITION
##  Min.   :157.0   Min.   : 815   Min.   :140.0   Length:18
##  1st Qu.:162.2   1st Qu.:1020   1st Qu.:151.2   Class :character
##  Median :167.0   Median :1088   Median :160.0   Mode  :character
##  Mean   :166.7   Mean   :1079   Mean   :158.3
##  3rd Qu.:169.8   3rd Qu.:1127   3rd Qu.:165.0
##  Max.   :177.0   Max.   :1370   Max.   :180.0
##       PRIX        R-POID.PUIS
##  Min.   :22100   Min.   : 9.725
##  1st Qu.:29843   1st Qu.:11.219
##  Median :33345   Median :13.182
##  Mean   :34159   Mean   :13.181
##  3rd Qu.:38458   3rd Qu.:14.549
##  Max.   :47700   Max.   :18.364
```

Classification hierarchique ascendante

On cherche a créer une typologie des voitures en fonction de leurs caractéristiques.

Création de la matrice des distances

Avant de passer l'algorythme de distances, il faut : - Sélectionner les variables quantitatives - Mettre les variables à la même échelle.

```r
# Sélectionner les variables quantitatives
autos_quant <- autos[, sapply(autos, is.numeric)]

# Mettre les variables à la même échelle
autos_scaled <- scale(autos_quant)

# Calculate the Manhattan distance matrix
dist_man <- dist(autos_scaled, method = "manhattan")
dist_euc <- dist(autos_scaled, method = "euclidean")

print(dist_man)
```

```
##            1         2         3         4         5         6         7
## 2  11.848176                                                            
## 3   7.924276  9.049112                                                  
## 4   6.253908 12.851424  3.802312                                        
## 5   6.698670  6.425499  7.825418 10.122184                              
## 6   7.081297  5.150544  3.898569  7.700880  4.479753                    
## 7  10.161378  5.972479  4.919294  8.721606  6.118908  4.625820          
## 8  10.121108 12.126842  5.257617  5.257892  9.397602  8.409722  7.997023
## 9  18.287226 12.047671 18.143756 21.946068 11.823883 14.245187 13.224462
## 10  7.471669 16.121446  7.630090  4.115643 13.392206 10.970903 11.991628
## 11  8.418935  9.317278 10.533782 12.077776  4.440874  7.394128 10.559782
## 12 10.384815  4.619832  6.282638 10.084949  5.533549  4.040462  1.725214
## 13 14.949284  7.137995 13.176016 16.978328  8.250614  9.277448  8.256723
## 14  8.758455  6.861375  8.608046 12.410358  2.743717  4.852517  5.852139
## 15  9.314711  6.390092  5.007166  7.891966  5.871391  3.842355  4.768252
## 16  7.220644  6.528139  5.959305  8.221598  3.458101  4.246379  4.145937
## 17 10.848816  6.026908 10.537621 14.339933  4.887299  7.485120  6.202449
## 18  6.512938 13.469119  5.670769  3.789877 10.739879  8.318575  9.911768
##            8         9        10        11        12        13        14
## 2                                                                       
## 3                                                                       
## 4                                                                       
## 5                                                                       
## 6                                                                       
## 7                                                                       
## 8                                                                       
## 9  21.221485                                                            
## 10  7.168787 25.216090                                                  
## 11 11.729580 10.394475 15.347798                                        
## 12  9.360367 11.861118 13.354972  9.974423                              
## 13 16.253746  7.235841 20.248351  9.330766  8.022538                    
## 14 11.685775  9.535710 15.680380  6.137703  4.879697  6.896694          
## 15  6.672085 15.207278 10.008812  8.123544  5.038159 10.239539  7.738390
## 16  8.638294 13.830989 11.385101  7.752945  4.074915  9.686947  4.695259
## 17 13.615350  8.239408 17.609955  7.746347  5.231705  6.923929  2.868269
## 18  6.485459 22.563762  4.003601 12.695471 10.794163 17.596023 13.028052
##           15        16        17
## 2                               
## 3                               
## 4                               
## 5                               
```

```
## 6
## 7
## 8
## 9
## 10
## 11
## 12
## 13
## 14
## 15
## 16  5.764316
## 17  9.786872  6.224854
## 18  7.417706  8.732774 14.957628
```

```r
print(dist_euc)
```

```
##            1         2         3         4         5         6         7
## 2  5.2155450
## 3  3.2925919 3.5224139
## 4  2.8260984 5.0002819 1.7605733
## 5  3.0503911 3.0856803 3.0390576 3.7996470
## 6  2.9953869 2.5549105 1.6954248 2.9117145 1.7973003
## 7  4.2637895 2.3827013 2.0890194 3.4470290 2.3966755 1.9746255
## 8  4.4706454 4.8413467 2.1736613 2.0395062 4.4649630 3.5006817 3.0932134
## 9  7.0405267 4.6665710 6.8297957 7.9523002 4.6561064 5.5668342 5.1988251
## 10 3.1542881 6.2858605 3.2100172 1.6570792 4.8554516 4.2038486 4.7841732
## 11 3.3509720 4.0808520 4.4536038 4.9187378 1.8031577 3.0101353 3.9978735
## 12 4.2813256 1.8803984 2.3300874 3.8209516 2.3336166 1.8022626 0.7782753
## 13 6.1671428 3.1423274 5.1205207 6.3517101 3.2475964 3.9855484 3.3947123
## 14 3.6977304 2.6352146 3.5175529 4.5461765 1.5941628 2.3550645 2.4034885
## 15 3.7712945 3.0470572 2.0279900 2.9877058 2.5128039 1.7013296 1.9644786
## 16 3.1932778 3.2431363 2.3092450 3.1313800 1.6425789 1.9804391 1.6480232
## 17 4.6672107 2.3329432 4.0366181 5.2540935 2.3165683 2.9502793 2.6538478
## 18 2.9478443 5.5101785 2.3886446 1.6386117 4.0457811 3.4442470 3.8164209
##            8         9        10        11        12        13        14
## 2
## 3
## 4
## 5
## 6
## 7
## 8
## 9  7.8290756
## 10 2.9881036 9.0876557
## 11 5.7439431 4.4031213 5.7730035
## 12 3.6011896 4.7478493 5.1797813 3.7825736
## 13 6.0592381 2.9014206 7.5244113 3.7735007 3.1772392
## 14 4.9114423 3.7622164 5.6288150 2.4692491 2.0008354 3.0881661
## 15 2.7933130 5.8373167 4.0420455 3.6824500 2.1595508 3.8356915 3.1157820
## 16 3.6364943 5.3040807 4.3803321 3.2471094 1.8422008 3.9554587 1.9873900
## 17 5.5208152 3.6300433 6.4720117 3.2364831 2.2002336 2.9736520 1.3006502
## 18 2.8393567 8.2360254 1.6623788 5.2898664 4.2104704 6.6081436 4.7293363
##           15        16        17
## 2
```

```
## 3
## 4
## 5
## 6
## 7
## 8
## 9
## 10
## 11
## 12
## 13
## 14
## 15
## 16 2.7018207
## 17 3.8510348 2.4111347
## 18 3.4021327 3.4034639 5.4931169
```

Il existe de nombreuses distances mathématiques pour les variables quantitatives (euclidiennes, Manhattan). La plupart peuvent être calculées avec la fonction dist.

La distance de Gower qui peut s'appliquer à un ensemble de variables à la fois qualitatives et quantitatives et qui se calcule avec la fonction `daisy` du package {cluster}.

```r
library(cluster)
# Calcul de la distance de Gower
distance_gower <- daisy(autos_scaled, metric = "gower")
print(distance_gower)
```

```
## Dissimilarities :
##             1          2          3          4          5          6          7
## 2  0.39877117
## 3  0.26822563 0.30597809
## 4  0.21349999 0.43228018 0.12630209
## 5  0.22442546 0.21886626 0.26448966 0.34079175
## 6  0.23689321 0.17396160 0.13201650 0.25831859 0.15063723
## 7  0.34057499 0.19994942 0.16326414 0.28956623 0.20660865 0.15385557
## 8  0.34552719 0.41395165 0.17663803 0.17604527 0.32246322 0.28471636 0.27123770
## 9  0.60758964 0.40191400 0.60526051 0.73156260 0.39077085 0.47324401 0.44199637
## 10 0.25657223 0.54514261 0.25719335 0.14019617 0.45365418 0.37118102 0.40242867
## 11 0.28396059 0.32025884 0.35663146 0.40793355 0.15109962 0.24941345 0.35770827
## 12 0.34602882 0.15428327 0.20926408 0.33556617 0.18528460 0.13253152 0.05915783
## 13 0.49746018 0.23184672 0.43782481 0.56412690 0.27303472 0.30580831 0.27456067
## 14 0.29179140 0.23159044 0.28923799 0.41554008 0.08929907 0.16181134 0.19819410
## 15 0.31531008 0.22251198 0.16702467 0.26207676 0.20270939 0.13161131 0.16101379
## 16 0.23937314 0.21955550 0.20198110 0.27759960 0.11310720 0.14052428 0.14097470
## 17 0.36641562 0.20348359 0.35866501 0.48496710 0.16565972 0.25379695 0.21341888
## 18 0.22409086 0.45792559 0.19237696 0.12947898 0.36643716 0.28396399 0.33371600
##             8          9         10         11         12         13         14
## 2
## 3
## 4
## 5
## 6
## 7
```

```
## 8
## 9  0.71323407
## 10 0.23379282 0.84442503
## 11 0.40210502 0.34063734 0.52079598
## 12 0.31723764 0.39599643 0.44842860 0.33638422
## 13 0.54579837 0.24587927 0.67698933 0.31326974 0.26606073
## 14 0.39721155 0.31602252 0.52840251 0.20489134 0.16403627 0.23066878
## 15 0.22052560 0.51323584 0.33118919 0.28055629 0.17153906 0.34580014 0.26264477
## 16 0.29589216 0.45738097 0.38704406 0.25970232 0.13615455 0.32119527 0.15469416
## 17 0.46663857 0.26962182 0.59782953 0.25772484 0.18001661 0.23104456 0.09951926
## 18 0.21349697 0.75720801 0.13057640 0.43357896 0.36416983 0.58977231 0.44118549
##            15         16         17
## 2
## 3
## 4
## 5
## 6
## 7
## 8
## 9
## 10
## 11
## 12
## 13
## 14
## 15
## 16 0.19646315
## 17 0.33661236 0.21078547
## 18 0.24595108 0.29982704 0.51061251
##
## Metric :  mixed ;  Types = I, I, I, I, I, I, I, I
## Number of objects : 18
```

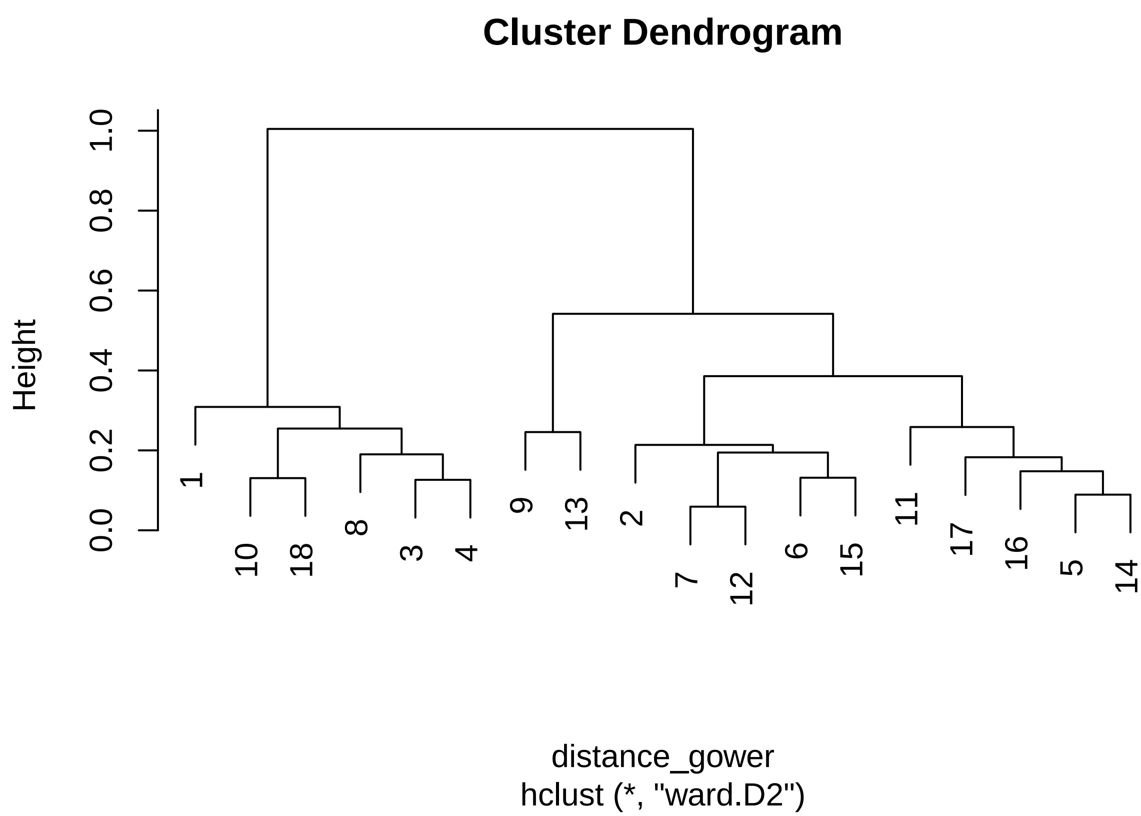
Création des clusters

On crée les clusters avec la fonction `hclust()`. On peut choisir plusieurs méthodes pour la création des clusters, elles sont disponibles dans l'aide de la fonction hclust().

```r
# Création des clusters avec hclust
clust_ward <- hclust(distance_gower, method = "ward.D2")
```

Visualisation

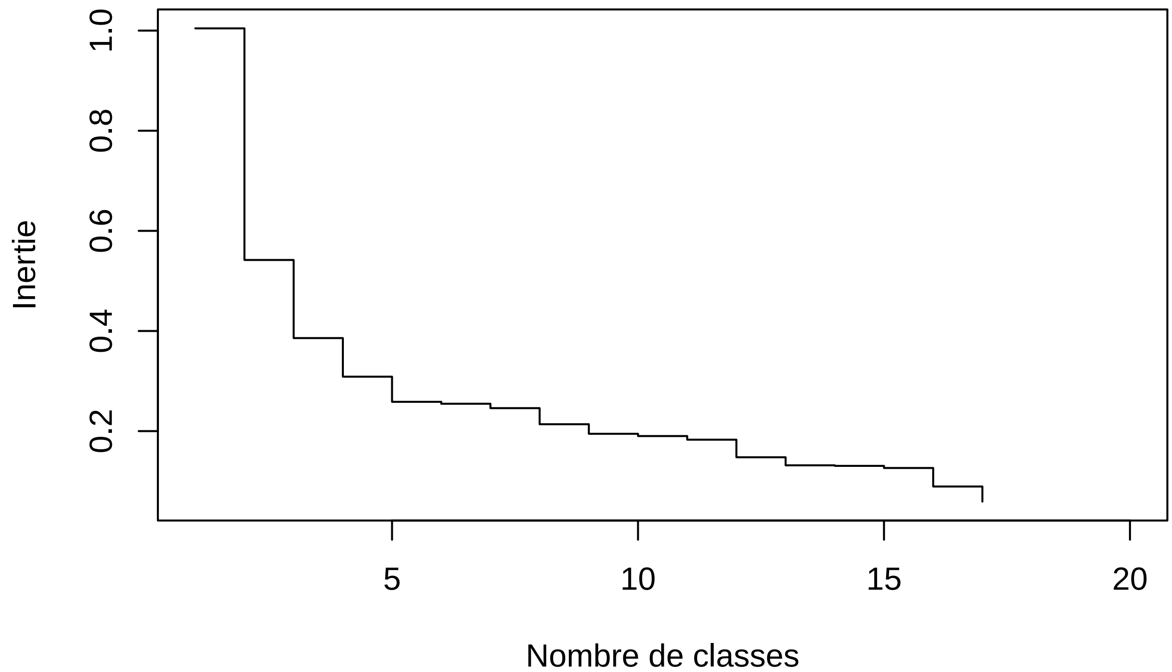
```r
# Visualisation du dendrogramme
plot(clust_ward)
```



Calcul de l'inertie

Pour déterminer à quelle hauteur il faut découper le dendrogramme, on regarde les sauts d'inertie :

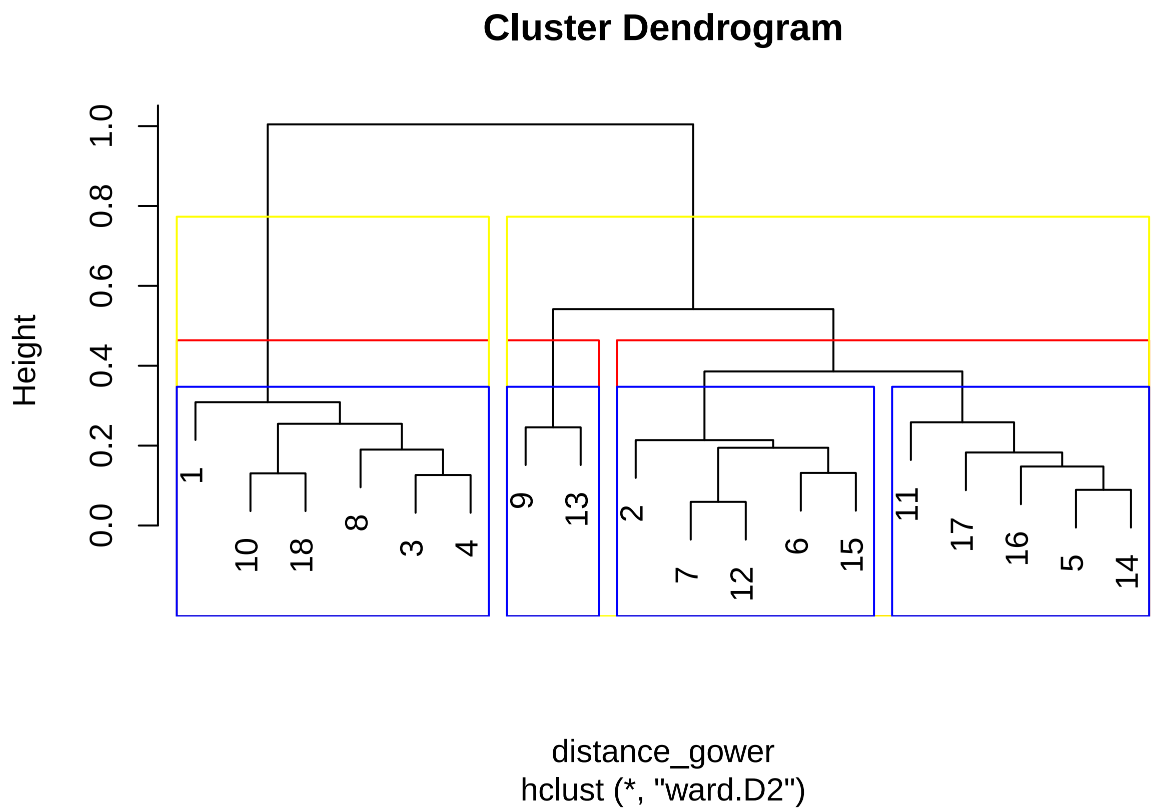
```
inertie <- sort(clust_ward$height, decreasing = TRUE)
plot(inertie[1:20], type = "s", xlab = "Nombre de classes", ylab = "Inertie")
```



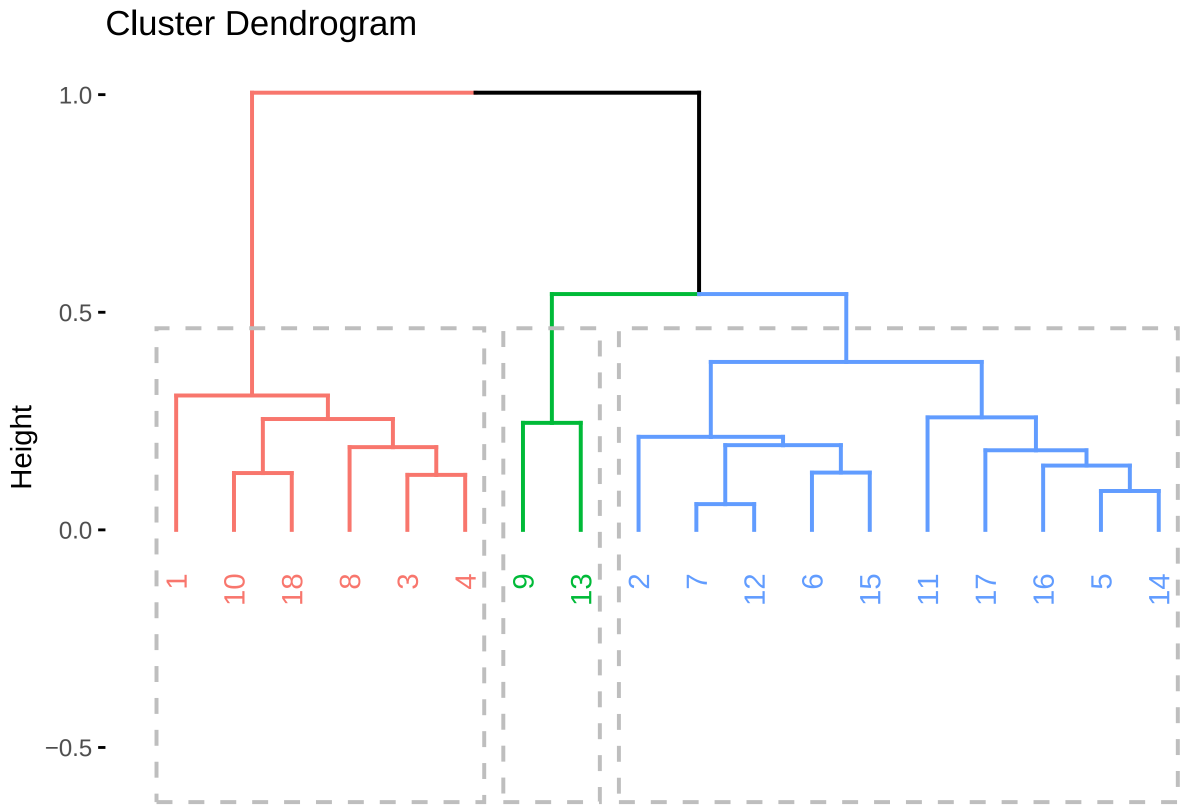
il y a un saut à 2, 3 et 4 ; on représente ces partitions directement sur le dendrogramme

```r
# Create the dendrogram plot
plot(clust_ward)

# Découpage et coloration des clusters sur le dendrogramme
rect.hclust(clust_ward, k = 3, border = "red")
rect.hclust(clust_ward, k = 2, border = "yellow")
rect.hclust(clust_ward, k = 4, border = "blue")
```

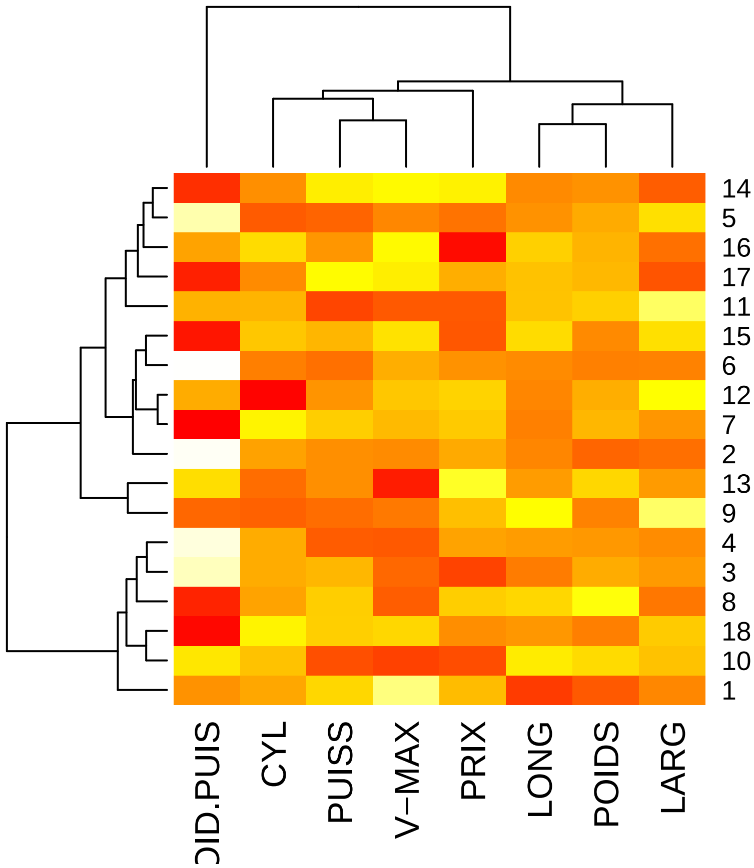



```
library(factoextra)
fviz_dend(clust_ward,k = 3, show_labels = TRUE, rect = TRUE)
```




```r
# Ordering the rows according to the clustering
ordered_data <- autos_scaled[order.dendrogram(as.dendrogram(clust_ward)), ]

# Creating the heatmap
heatmap(ordered_data, Rowv = as.dendrogram(clust_ward), col = heat.colors(256))
```




```
# # Calculate total inertia (sum of squared distances)
# total_inertia <- sum(as.matrix(dist_matrix)^2)
#
# # Function to calculate R^2 for a given number of clusters
# calc_R2 <- function(k, dist_matrix) {
#   clust <- hclust(dist_matrix, method = "ward.D2")
#   groups <- cutree(clust, k)
#   aggregate_inertia <- sum(tapply(dist_matrix, INDEX = groups, FUN = function(x) sum(x^2)))
#   R2 <- (total_inertia - aggregate_inertia) / total_inertia
#   return(R2)
# }
#
# # Test a range of cluster numbers and compute R^2
# R2_values <- sapply(2:10, calc_R2, dist_matrix = dist_matrix)
#
# # Find the number of clusters with the highest R^2 value
# optimal_clusters <- which.max(R2_values)
#
# print(R2_values)
# print(optimal_clusters)
```