TP3 ACP, K-Means et HAC

Data Mining

Réalisé par :

LABAYBI OUTMANE

KARDATE HAMZA

Encadré par :

Prof El HARTI MOSTAFA

Table des matières

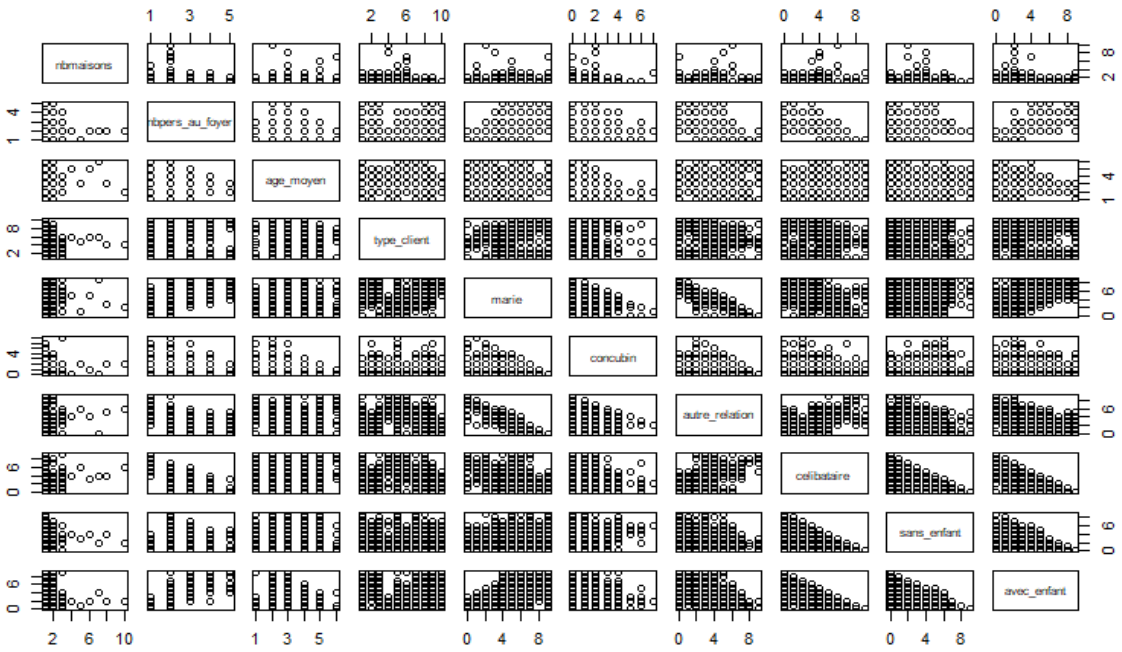
I-ANALYSE EN COMPOSANTES PRINCIPALES :

Dans la première partie de ce projet, on va effectuer une Analyse en Composantes Principales (ACP) sur le fichier banque qui est composé de 35 variables et 5822 individus.

On importe le fichier dans RStudio avec la commande suivant :

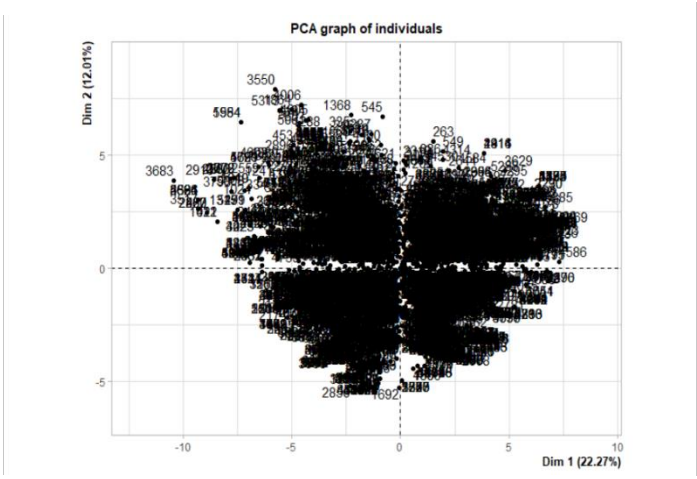
```
library(FactoMineR)
library(readxl)
library(corrp)
data<- read_excel(file.choose())
```

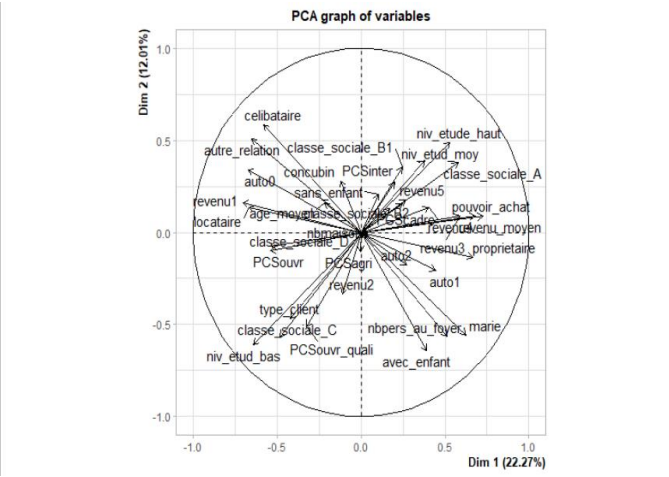
Puis on tape la commande : pairs(data[1:10]) pour afficher les graph de corrélation entre les variables, puisque le nombre des variables est grands on ne peut afficher tous, c'est pour ça on a choisi d'afficher les 10 premier variables



Après, on lance la procédure d'analyse en Composant Principales pour tous les données on tapant la commande suivant : **acp<-PCA(data)** on met la premier résultat dans la variable acp et on obtient :

```
**Results for the Principal Component Analysis (PCA)**
The analysis was performed on 5822 individuals, described by 35 variables
```





Choix du nombre d'axes à retenir

On tape la commande : **acp$eig** pour avoir la matrice des valeurs propres et pourcentage des variances

acp$eig

	eigenvalue	percentage of variance	cumulative percentage of variance
comp 1	7.7945744224	22.270212635	22.27021
comp 2	4.2035570684	12.010163053	34.28038
comp 3	2.5962220441	7.417777269	41.69815
comp 4	1.9095631138	5.455894611	47.15405
comp 5	1.8350018230	5.242862351	52.39691
comp 6	1.5050527790	4.300150797	56.69706
comp 7	1.3422657321	3.835044949	60.53211
comp 8	1.2549728144	3.585636613	64.11774
comp 9	1.1762162924	3.360617978	67.47836
comp 10	1.1220224131	3.205778323	70.68414
comp 11	1.0271241946	2.934640556	73.61878
comp 12	0.9825762152	2.807360615	76.42614
comp 13	0.8811708244	2.517630927	78.94377
comp 14	0.8615825973	2.461664564	81.40544
comp 15	0.8046838564	2.299096733	83.70453
comp 16	0.7312860821	2.089388806	85.79392
comp 17	0.6862446240	1.960698926	87.75462
comp 18	0.6687403723	1.910686778	89.66531
comp 19	0.6113618750	1.746748214	91.41205
comp 20	0.5647928444	1.613693841	93.02575
comp 21	0.4674278954	1.335508273	94.36126
comp 22	0.4286778306	1.224793802	95.58605
comp 23	0.3295760525	0.941645864	96.52770
comp 24	0.3229651055	0.922757444	97.45045
comp 25	0.2262895284	0.646541510	98.09700
comp 26	0.1904887071	0.544253449	98.64125
comp 27	0.1716144007	0.490326859	99.13158
comp 28	0.1151507327	0.329002094	99.46058
comp 29	0.0649366999	0.185533428	99.64611

comp 30 0.0311969812	0.089134232	99.73525
comp 31 0.0287073986	0.082021139	99.81727
comp 32 0.0267260092	0.076360026	99.89363
comp 33 0.0198182798	0.056623657	99.95025
comp 34 0.0170121289	0.048606082	99.99886
comp 35 0.0004002607	0.001143602	100.00000

Tableau des valeurs propres et pourcentage des variances

Pour choisir le nombre d'axes à retenir on applique le critère de Kaiser qui permet de choisir les axes dont l'inertie est supérieure à l'inertie moyenne I/p=1 (ACP normée). Ici, le critère de Kaiser nous conduit à sélectionner 11 axes (dont les valeurs propres > 1) cela permet d'avoir 73,61% d'inertie total.

Maintenant on va relancer la commande PCA mais on va fixer le nombre d'axes a 11.

```
resultat_acp <- PCA(data,ncp = 11)
```

On gardre les resultat d'ACP dans une variable nommé resultat_acp et on termine la partie d'ACP parce que l'objectif de cette étape c'était la préparation des données et minimiser le nombre d'axes.

K-MEANS : FONCTION KMEANS

Dans cette partie on va traiter le problème de classification. Puisque le nombre d'individus est grands, le résultat de l'analyse en composantes principales n'a pas donné des bonnes interprétations. La solution est donc et de regrouper ces individus en classes pour diminuer la taille des données

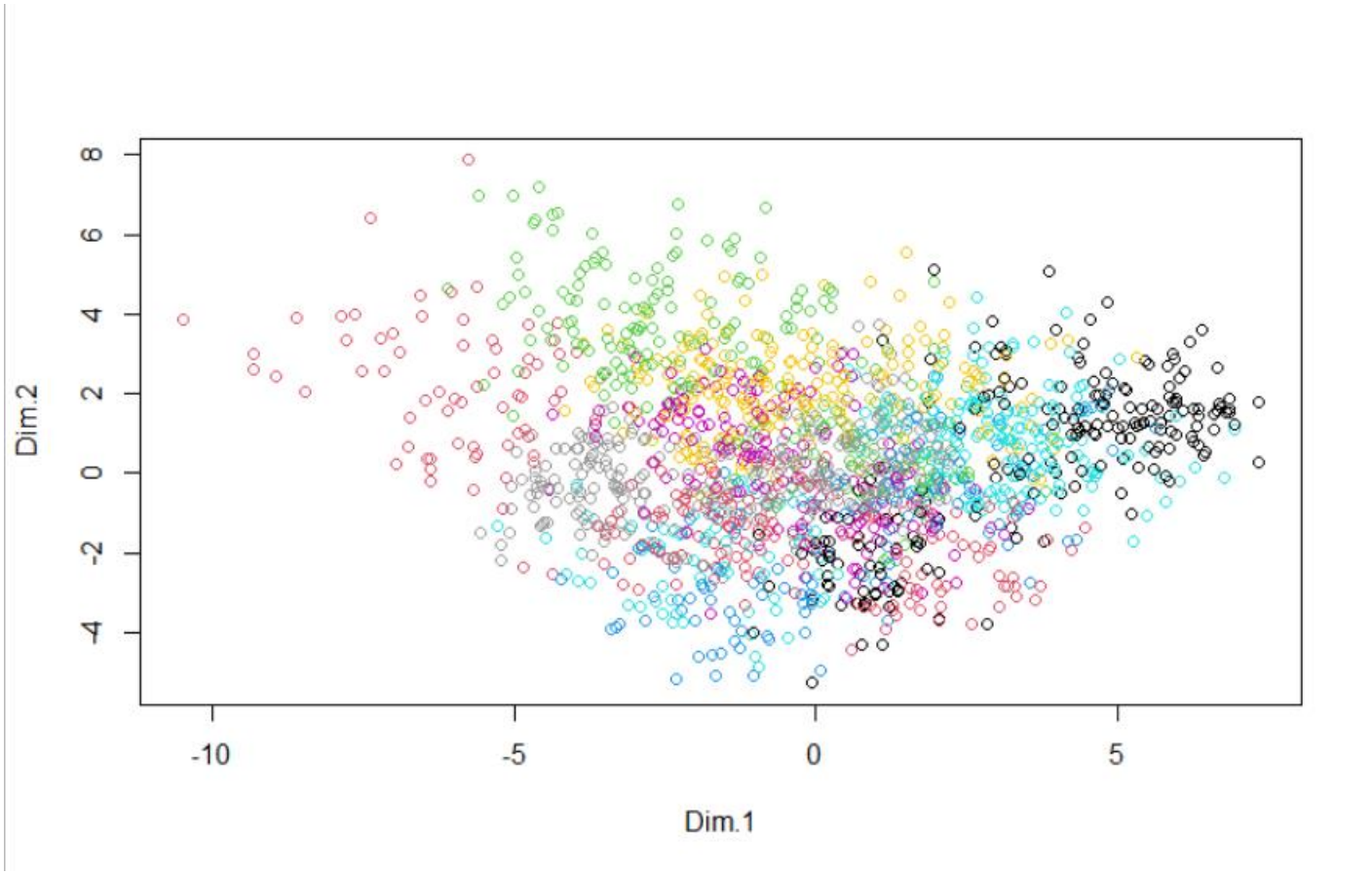
Donc on va suivre le processus de classification on utilisant l'algorithme de K-Means . on prend en entrée résultat de l' ACP qui été faite dans la partie précédente avec 11 axes (resultat_acpindcoord) et a comme paramètre le nombre maximum des classes, on utilisant la formule de Wrong qui est k= $n^{0,3}$ = $5822^{0,3}$ =13.47 , donc on prend k=20 (nombre de classe) pour avoir des bons résultat.

Et on lance la procédure de KMeans , le fonction de kmeans utilise une fonction aléatoire pour le choix des ponts centraux. Pour avoir le même résultat d'une exécution à l'autre on utilise set.sed(10)

```
nombre_class=20
res_km<- kmeans(resultat_acp$ind$coord,centers = nombre_class,iter.max = 40,nstart = 10)
```

On stock les résultats obtenu dans une variables nommé res_km

On peut afficher les individus de chaque classe pour l'axe1 et l'axes2, on tapant la commande **plot(resultat_acpindcoord,col=res_km$cluster)** et on obtient le graphe suivant :



Après on lance la commande **head(res_km$cluster,50)** pour voir affectation de chaque individu à la classe convenable et on obtient les résultats suivants :

```
> head(res_km$cluster,50)
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
 6  8 11 19  9  3 18 17  5 10  1 19 14  8  3 17 19  4 13  5  4  6 13 17 10  6 19 20 18 16 14  3 17 14 10 13
37 38 39 40 41 42 43 44 45 46 47 48 49 50
12  8 18 17 18 12 13  8  8 18  7 18 19  4
```

Vu le nombre des individus est grands on a affiché seulement les premiers 50.

Après on utilise la commande **res_km$centers** pour afficher les des classes

```
res_km$centers
      Dim.1      Dim.2      Dim.3      Dim.4      Dim.5      Dim.6      Dim.7      Dim.8
```

1	4.8829063	1.278716485	1.7801167	0.32799817	-0.99039353	0.55191777	1.3629459	0.36833899
2	-5.8876173	1.650247102	1.6826110	0.42016179	-0.63979570	-0.99702196	0.3273723	0.32366111
3	-3.0060773	3.670262649	-1.2484742	0.94938969	-0.04337010	0.75444515	-0.5454737	-0.31418344
4	-1.3740805	-3.182352903	0.1899808	-0.19058376	-1.02856301	0.22209401	0.6802477	-0.38104063
5	-1.7471445	-2.359249574	-0.2035128	-2.41647492	-0.09411837	0.48944938	-0.3972238	0.96131343
6	0.7148184	-1.289387591	0.3276584	0.08037139	0.58314991	-0.96192056	0.3279577	-0.31739519
7	-1.2121721	1.916702643	0.8205166	-0.48622495	0.11083945	-0.50907434	-0.1773387	0.76213555
8	0.4777793	0.007424612	0.4616129	0.36649520	0.83230431	0.49637110	0.5810114	0.26662854
9	5.0213217	1.915037051	3.5123377	-1.68334558	-0.77514250	0.91207807	-2.3230181	-1.83355670
10	2.1483361	-2.598422090	-1.5037797	0.55643080	-0.64500144	-0.80980984	-0.5853457	0.87249848
11	-0.9628561	2.861126826	-2.7188745	-0.21714533	-0.03571493	-0.38341284	-0.2926956	1.20270450
12	2.8826003	0.501126266	-2.7186868	0.13074422	1.82565715	-0.61082788	0.2103475	0.03490447
13	3.2300512	0.754766845	-0.7117559	0.53729131	-0.83762406	-0.34005042	-0.1403977	0.06170366
14	-1.5146040	1.146093856	-1.4368822	-0.69916372	-0.00519427	1.61716123	-0.1952913	-0.28760454
15	2.6012326	2.018613756	1.0939692	-2.22057288	-0.25954113	-0.93268990	-0.8110531	-0.54750239
16	-3.2195681	-0.537193476	0.8089141	0.25586603	0.13963066	-0.48834548	0.2617302	-0.22611692
17	0.7763298	-2.459796690	0.7622133	0.08904995	1.07735275	1.68254984	-0.5897886	0.44837882
18	-1.7421508	-1.207539282	-0.5938177	0.95267283	-0.98877657	-0.03310619	-0.1500441	-0.82175068
19	1.0436077	0.357107619	-2.0245470	-1.21543065	0.38795282	-0.68693270	0.1308366	-1.10955412
20	1.5249133	-0.738658552	1.7562209	1.98513901	3.73499190	-0.85542301	-1.2803328	-0.98981368

	Dim.9	Dim.10	Dim.11
1	0.342404719	0.42183065	-0.72581501
2	0.161354200	-0.04243684	-0.02893954
3	-0.419279816	0.16698798	-0.60326611
4	0.223015608	-0.73463450	0.19757310
5	-0.116100830	0.78735822	-0.73039865
6	-1.634919575	-0.24552465	0.74189255
7	0.218443013	0.11706278	0.36530675
8	-0.177089875	0.39812739	0.04668907
9	0.491836489	-1.31182954	1.57662647
10	0.007936731	-0.08126919	-0.01721780
11	0.887907863	-0.28893976	1.23991581
12	0.726815029	-0.50844351	-0.80718771

```
13 -0.253541570 -0.10621295  0.02550748
14 -0.617511521 -0.19248345  0.09488649
15  0.083843232 -0.42862489 -0.97386756
16 -0.086529089 -0.37617879 -0.09546827
17  0.398232531 -0.27898256 -0.03094535
18  0.752876420  0.36423899 -0.12404086
19  0.816366172  0.61039283  0.54836671
20 -0.273421847  0.43425463  0.59997897
```

Nous allons mettre les centres de k-maens dans une variable pour l'utilise dans la prochaine partie de la classification hiérarchique ascendante (HAC)

On peut afficher autres informations à propos la classification K-means par exemple la distance intra_classe :

```
> res_km$tot.withinss
[1] 62595.62
```

La distance inter-classe :

```
> res_km$withinss
 [1] 4637.732 2897.101 2698.105 3054.205 3388.545 3272.285 4698.487 4998.190 1306.576 3731.508 1395.904
[12] 1390.494 5175.347 2851.222 1862.338 5179.926 2535.373 3283.491 2641.942 1596.852
```

Le nombre des individus de chaque classe :

```
> res_km$size
 [1] 322 214 271 301 249 313 407 594  81 284 105 141 518 307 168 508 299 379 244 117
```

CLASSIFICATION ASCENDANTE HIÉRARCHIQUE (HAC)

Après avoir faire la classification K-Means nous avons obtenu 20 classes, la classification hiérarchique se base sur les résultats de la classification K-Means

Fonction HCPC

La fonction a d'abord construit un arbre hiérarchique. Ensuite, la somme de l'inertie intra-cluster est calculée pour chaque partition. La partition suggérée est celle avec la perte relative d'inertie la plus élevée

Pour utiliser la fonction HCPC de Factominer on la lance directement en utilisant les centres de K-means qu'on a obtenu :

```
resultat_hcpc<-HCPC(centres)
```

Cette fonction Renvoie une liste comprenant :

resultat_hcpc$data.clust

Les données d'origine avec une colonne supplémentaire appelée clust contenant la partition.

desc.var

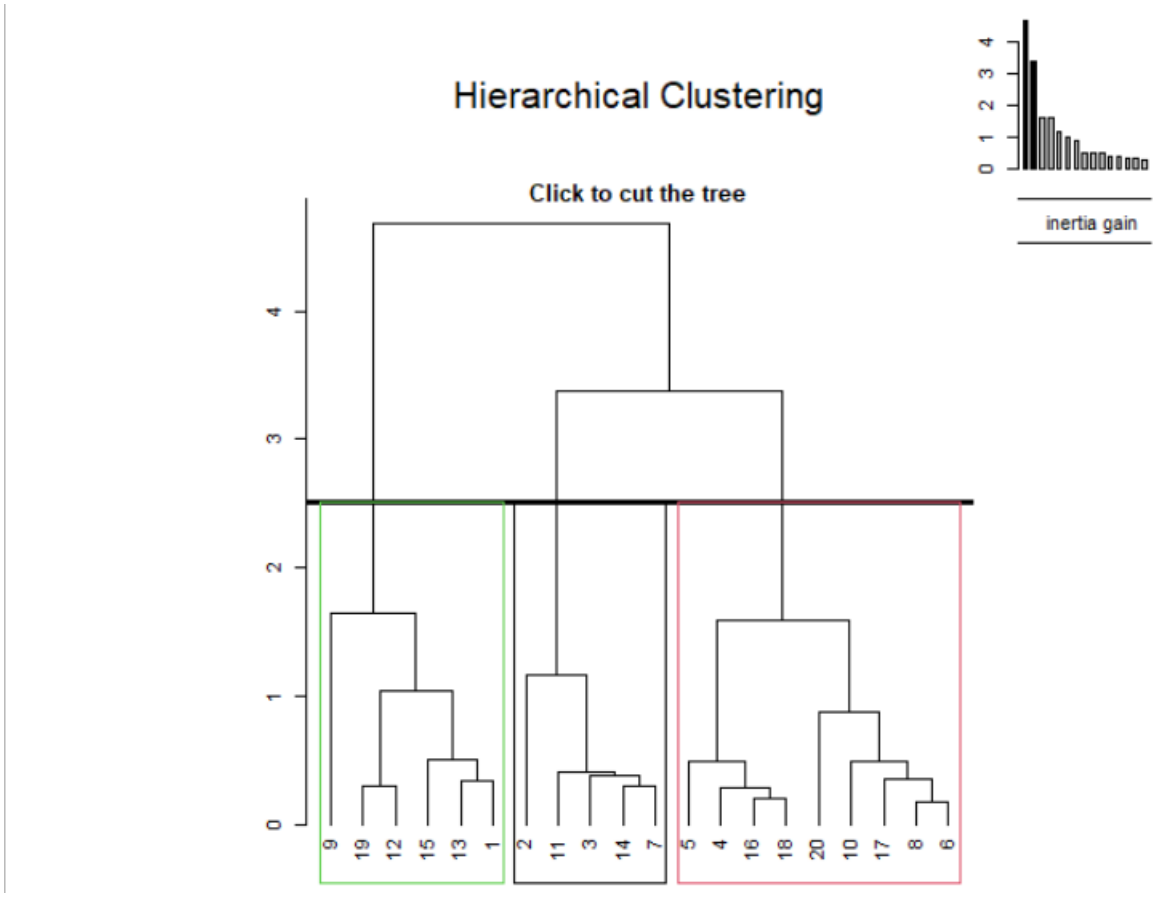
La description des classes par les variables.

desc.axes

La description des classes par les facteurs (axes).

Et pour la partie graphique on obtient le graphe suivant qui permet de couper pour former le nombre de classe à retenir, par défaut l'algorithme donne 3 classe basant sur le petit graphe

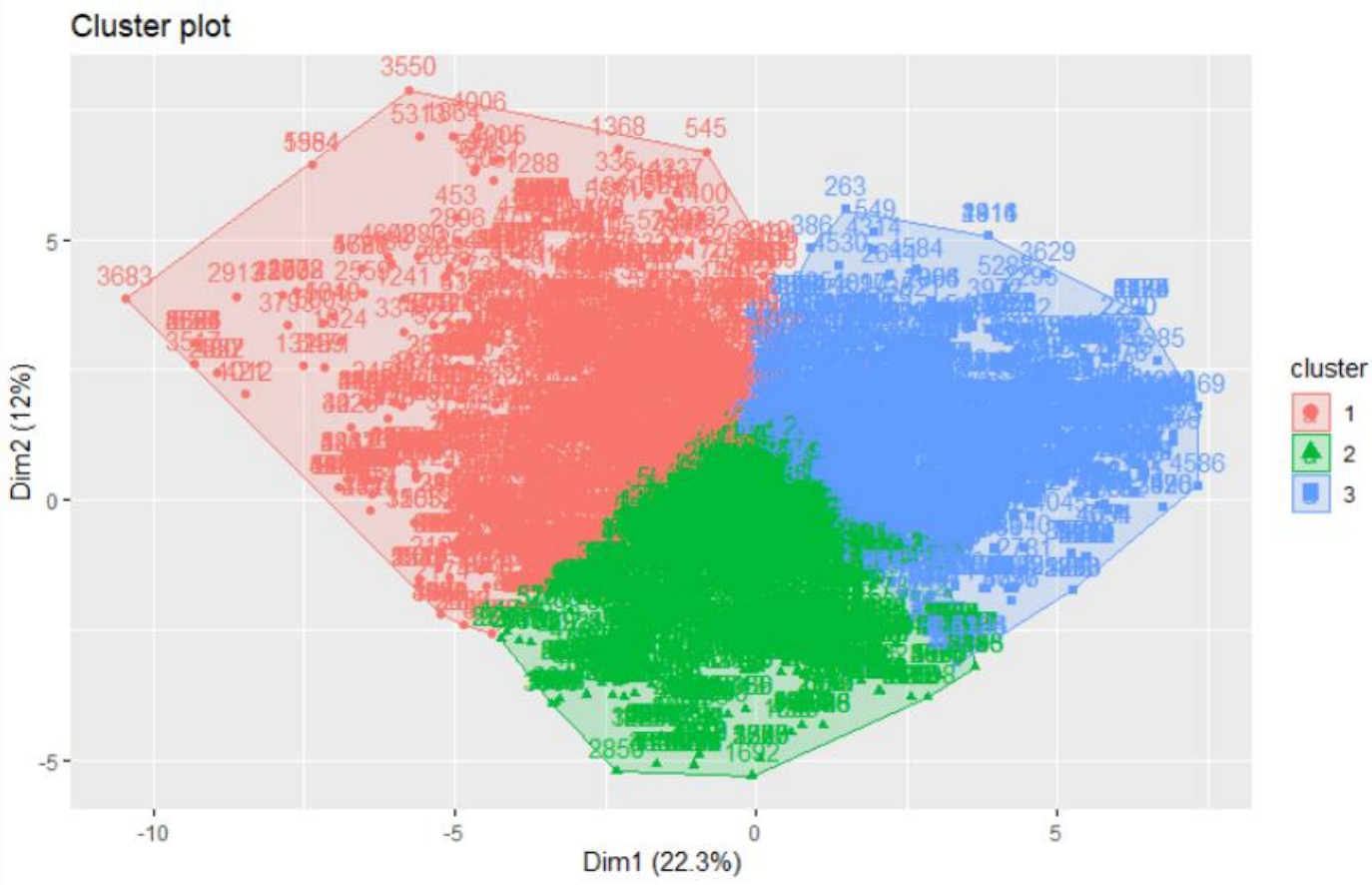
en haut de inertia gain

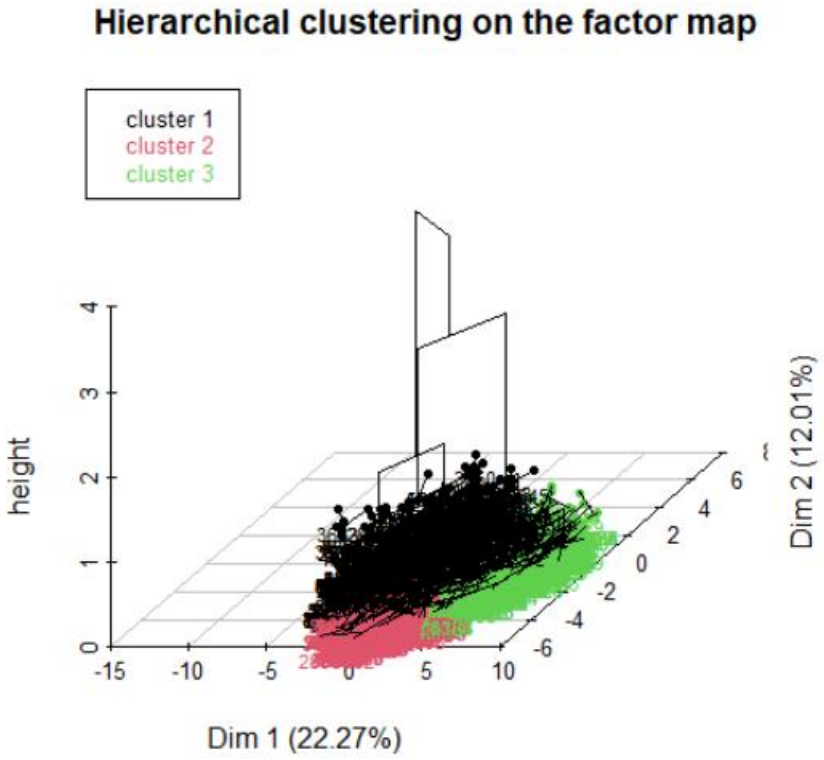


Et pour afficher les indivudaus par classe on utilise les commandes suivantes :

```
fviz_cluster(resultat_hcpc)
color=as.integer(resultat_hcpc$data.clust$clust)
plot(resultat_hcpc, choix="ind",habillage="quali",col.ind=color)
```

Et on obtient les graphes suivants


Cluster plot
Dim2 (12%)
Dim1 (22.3%)
cluster
1
2
3


Hierarchical clustering on the factor map
cluster 1
cluster 2
cluster 3
height
Dim 1 (22.27%)
Dim 2 (12.01%)

Fonction hclust

Hclust est une fonction permet d'effectuer une analyse de cluster hiérarchique en utilisant un ensemble de dissemblances pour les n objets des centres de cluster K-Means. Initialement, chaque objet est assigné à son propre cluster, puis l'algorithme procède de manière itérative, à chaque étape joignant les deux clusters les plus similaires, en continuant jusqu'à ce qu'il n'y ait qu'un seul cluster.

Au départ on utilise la fonction scale qui permet de standardiser les centres

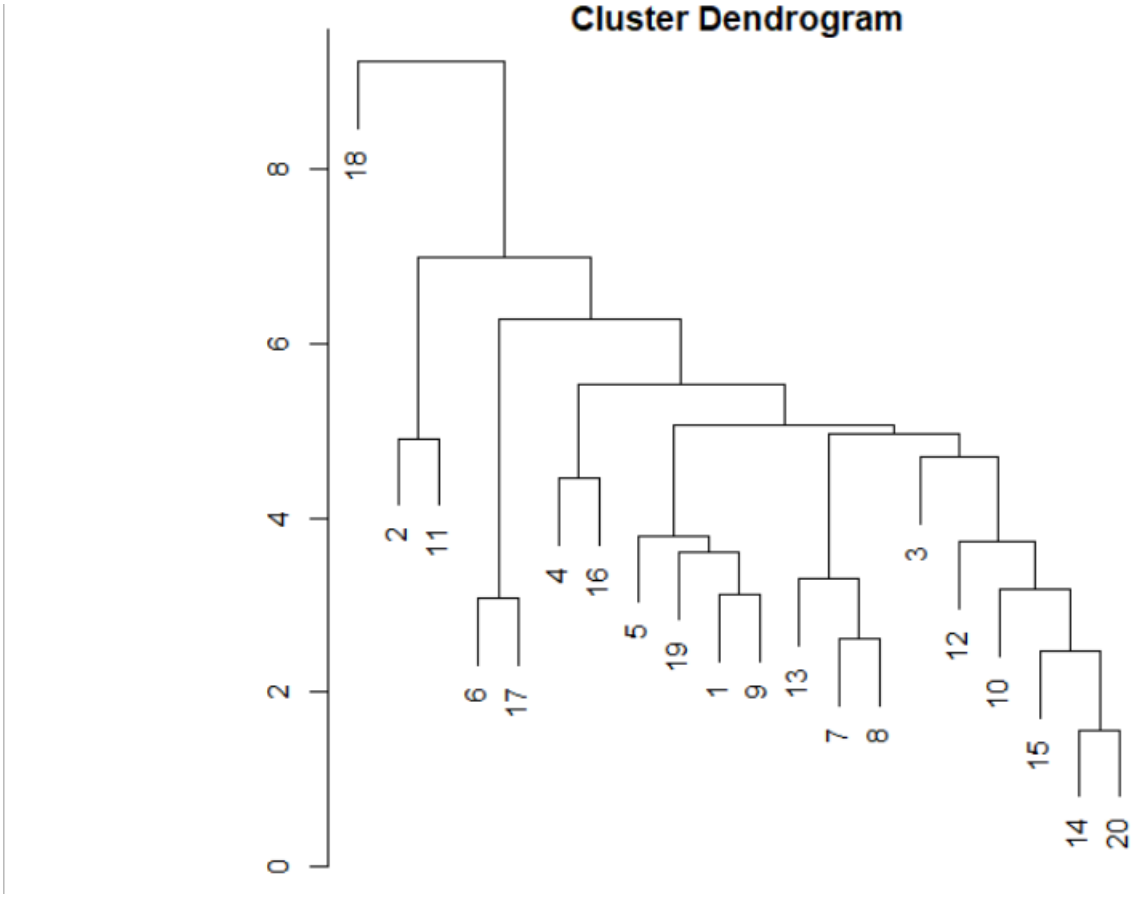
```
res_scale<-scale(centres,center = TRUE,scale = TRUE)
```

Nous calculons après les valeurs de dissimilarité avec dist, puis introduisons ces valeurs dans hclustet spécifions la méthode d'agglomération à utiliser (c'est-à-dire « complète », « moyenne », « unique », « ward.D2»)

Dans notre exemple on va utiliser la méthode ward.D2.

```
res_distance<- dist(res_scale)
res_hclust<-hclust(res_distance,method = "ward.D2")
```

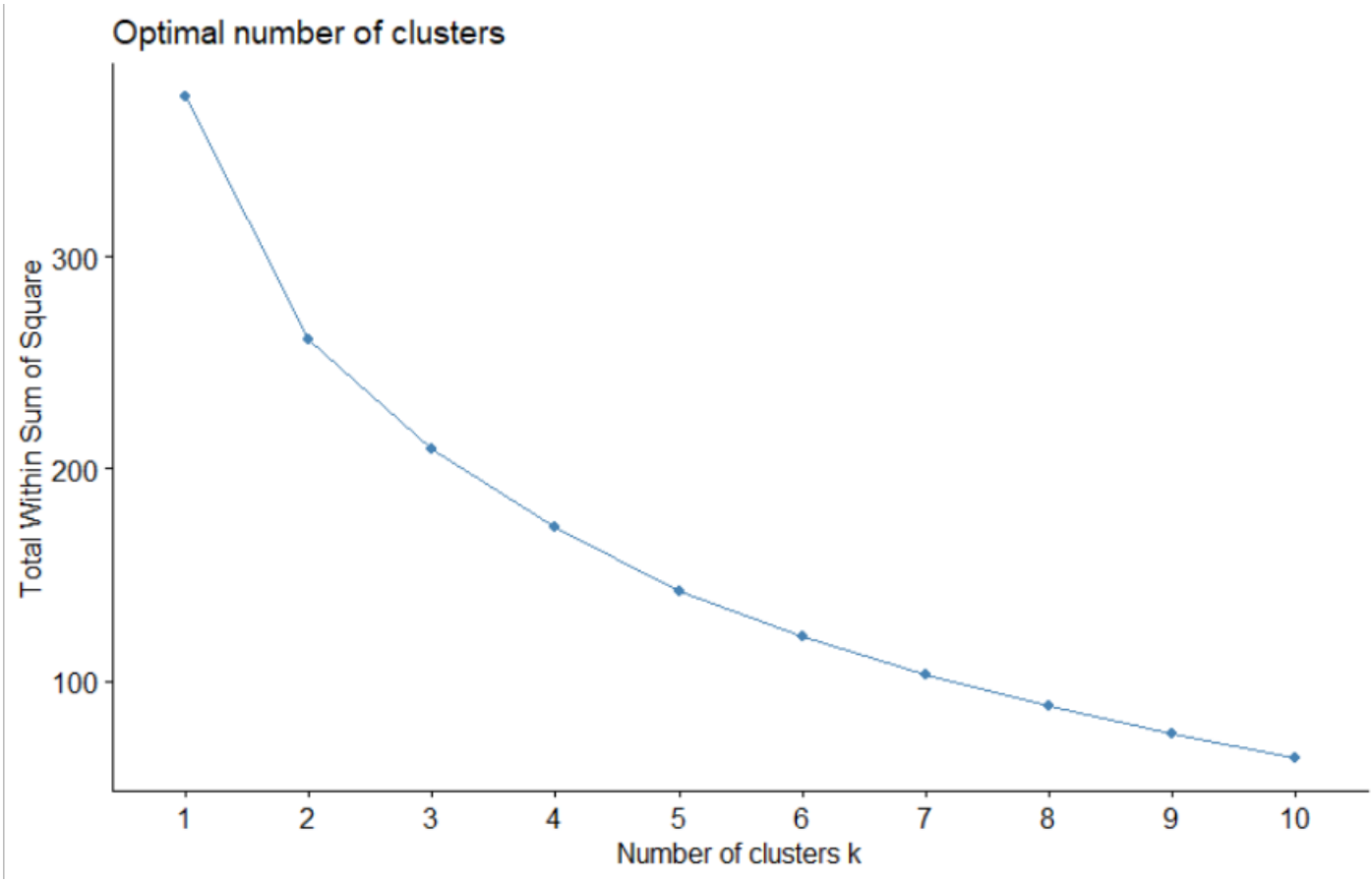
Il est possible de dessiner le dendrogramme :



Pour bien choisir le nombre de classe à retenir en plus d'endogramme on peut utiliser la méthode de coude, nous tapons la commande suivant

```
fviz_nbclust(centres, FUN = hcut, method = "wss")
```

Et on obtient les résultat le graphe suivant



On remarque dans le graphe de coude qu'il y a une distance important entre 4 premières points et aussi dans le graphe d'endogramme donc on choisit 4 classe.

Donc on va découper les résultat obtenu dans hclust en 4 classe

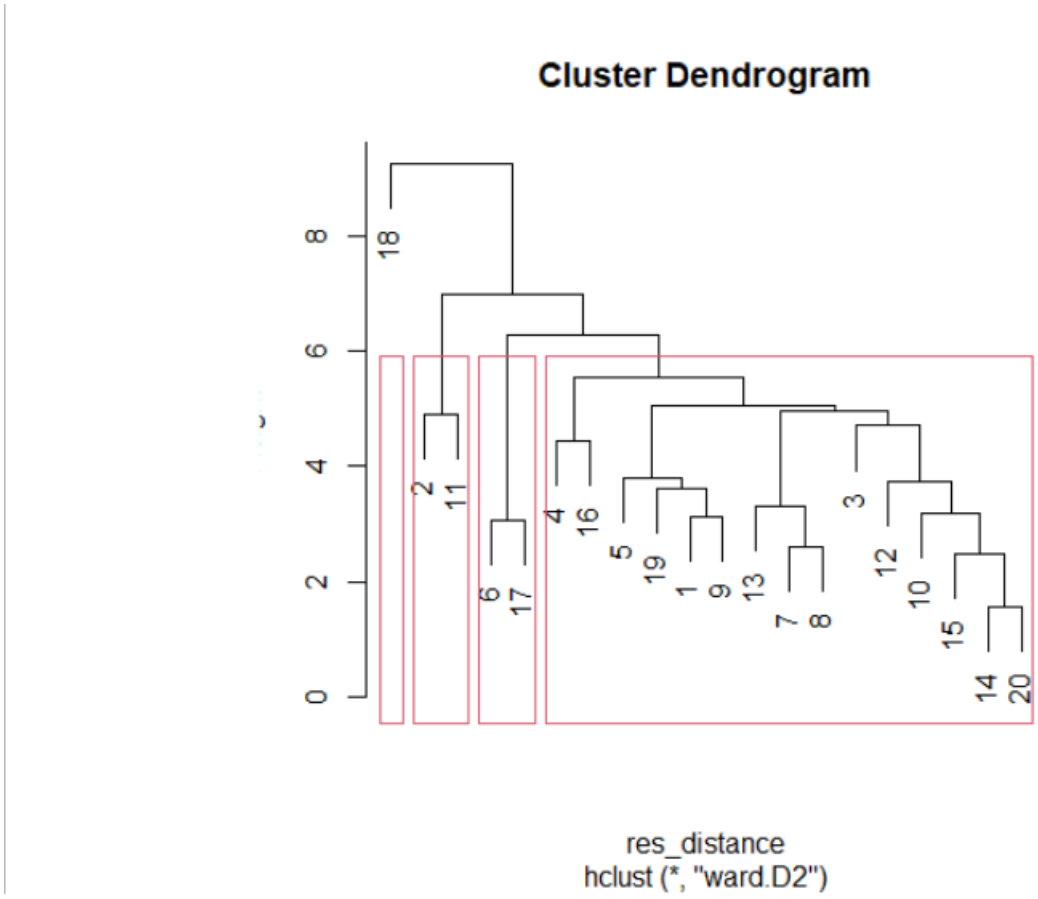
```
class_hclust<-cutree(res_hclust,k=4)
print(sort(class_hclust))
table(class_hclust)
```

Et on obtient la classe de chaque centre de classe K-Means

```
> print(sort(class_hclust))
 1  3  4  5  7  8  9 10 12 13 14 15 16 19 20  2 11  6 17 18
 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  2  2  3  3  4
```

Et aussi le nombre de classe de K-Means dans chaque classe de hclust

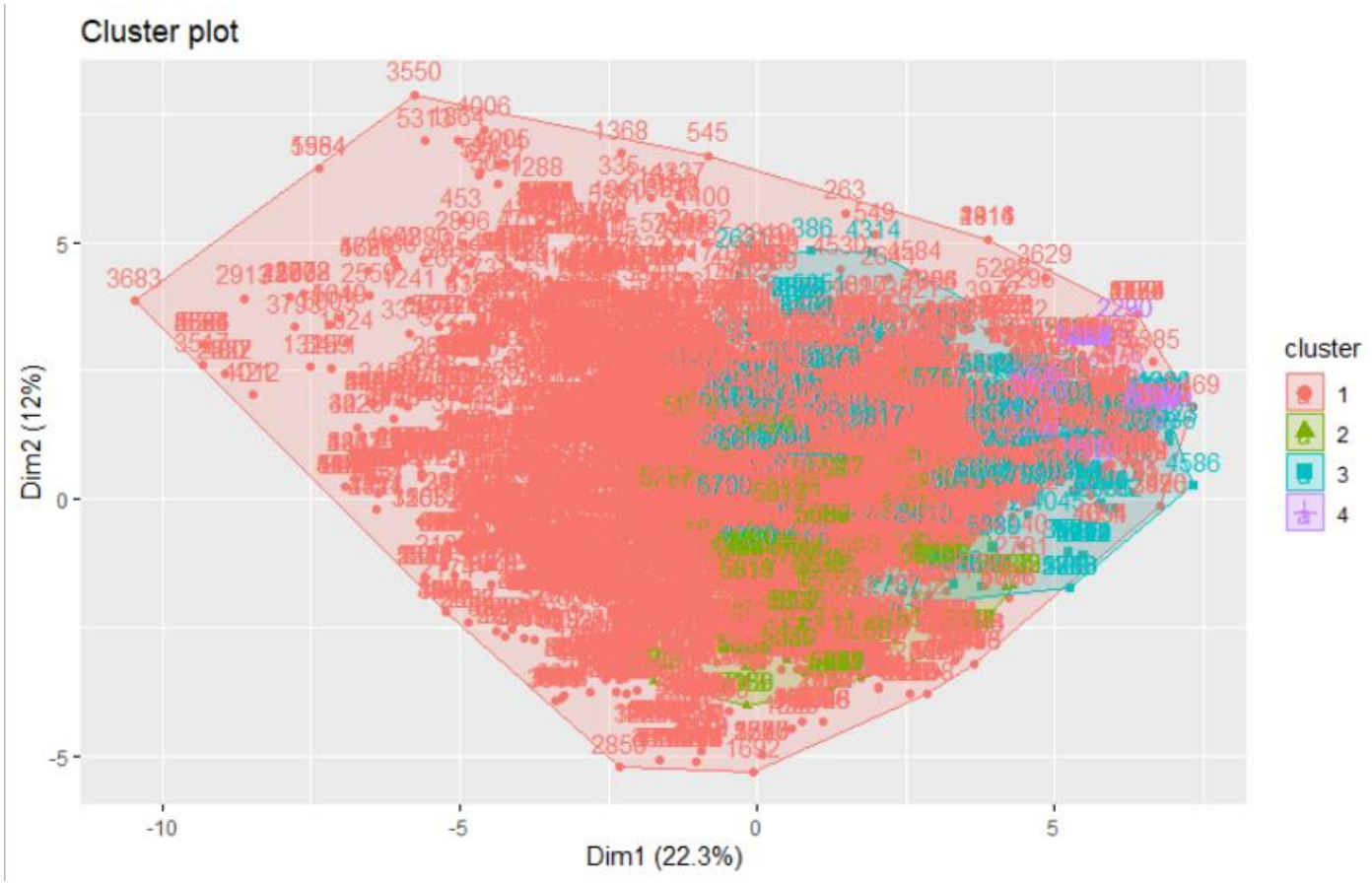
```
> table(class_hclust)
class_hclust
 1  2  3  4
15  2  2  1
```



On peut afficher les individus de chaque classe HAC on utilisant la fonction fviz_cluster de packages Factoextra , on tapons la commande suivant :

```
fviz_cluster(list(data = data, cluster = class_final))
```

Et on obtient le graphe suivant sur l'axes (1,2)



Affectation des classes aux individus

Dans cette étape on affecte chaque individus à une classe de HAC, pour faire ça on se basant sur les résultats de classification K-Means , et vois le code :

```
class_final<-res_km$cluster
 for (i in l:length(class_final)){
   k=res_km$cluster[i]
    class_final[i]=class_hclust[k]
   }
class_final <- as.factor(class_final)
```

Pour voir la classe de chaque individus on tape print(class_final) et on obtient

```
class_final
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21
 2  3  1  1  4  1  1  1  1  1  3  1  1  1  1  1  3  1  1  1  1
22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42
 2  1  1  1  2  1  2  1  1  1  1  1  1  1  1  1  1  1  1  1  1
43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63
```

1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 3 1 1 1
64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84
1 1 1 1 1 2 1 1 1 1 1 3 2 1 1 1 1 2 1 1 2
85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105
1 1 1 1 1 1 1 1 2 1 3 1 1 1 1 1 1 1 1 1 1
106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126
1 4 1 1 1 3 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1
127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147
1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 3 1 1 1 1
148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168
1 2 1 3 1 1 1 1 3 1 2 1 1 1 3 1 2 1 1 1 3
169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189
3 1 1 1 1 2 1 1 1 3 1 1 3 1 1 1 1 1 1 1 1
190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210
1 3 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1
211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231
1 1 1 1 1 2 3 1 1 1 1 1 2 1 1 3 3 1 1 1 1
232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252
1 1 1 1 1 1 1 3 3 1 1 2 1 1 1 1 1 1 1 1 4
253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273
1 1 1 1 1 1 3 1 3 1 1 1 1 1 1 1 1 1 1 1 1
274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294
3 3 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 3 1 1
295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315
4 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336
1 1 1 1 1 1 3 2 3 1 1 3 1 1 1 1 1 2 1 1 1
337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357
1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1
358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378
1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 4 3 1 1 1 1
379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399
1 1 1 1 1 1 1 3 1 2 1 3 1 1 1 1 1 1 2 1 1
400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420
1 1 2 4 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1
421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441
1 1 3 1 1 1 2 1 1 1 1 1 1 1 1 1 1 3 1 1 4 1
442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483
1 1 1 1 2 1 3 1 1 1 1 1 1 1 3 1 1 1 1 1 3
484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504
1 1 1 1 1 3 1 2 1 1 1 1 1 3 1 1 1 1 1 1 1
505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525
1 1 1 2 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546
1 4 1 1 1 1 2 1 1 1 1 4 1 1 1 1 1 3 1 1 1
547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567
1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1
568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588
1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1
589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609

```
  1   1   1   4   1   1   1   1   1   3   2   1   1   2   2   1   1   1   1   1   1
610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629 630
  1   1   1   1   1   1   2   1   1   1   1   1   1   1   1   3   1   1   1   1   1
631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651
  1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   2   1   2   1
652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672
  1   1   1   1   1   2   1   4   1   1   3   1   1   1   1   1   3   1   1   1   1
673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693
  1   1   2   1   1   1   1   3   1   3   1   1   1   1   1   1   1   1   1   1   1
694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714
  1   3   1   1   1   1   1   1   1   1   1   1   3   1   1   1   1   1   1   1   1
715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735
  1   1   1   1   1   1   1   2   1   1   1   1   1   1   1   1   1   1   1   1   1
736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755 756
  1   1   1   3   1   1   1   1   1   1   1   3   4   1   1   1   1   1   1   1   1
757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777
  1   1   1   2   1   1   1   1   3   1   1   1   1   2   1   2   1   1   1   1   1
778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798
  1   3   1   1   1   1   1   1   1   3   1   1   1   1   1   1   3   1   1   1   1
799 800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819
  1   1   1   1   1   1   3   3   1   1   1   1   3   1   2   1   1   1   1   1   2
```

Et pour la correspandance entre les classe de K-Means et les classe de HAC on tape la commande suivante :

```
table(class_final,res_km$cluster)
```

Et on obtient le tableau suivant :

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	283	0	272	215	323	0	228	402	220	338	0	382	391	640	297	169	0	0	370	449
2	0	131	0	0	0	0	0	0	0	0	250	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	198	0	0	0	0	0	0	0	0	0	0	199	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	65	0	0

D’après ce tableau on remarque que :

- La classe 1 de HAC coïncide avec les classes (1,3,4,5,7,8,9,10,12,13,14,15,16,19,20) de K-Means
- La classe 2 de HAC coïncide avec les classes (2,11) de K-Means
- La classe 3 de HAC coïncide avec les classes (6,17) de K-Means
- La classe 4 de HAC coïncide avec les classes (18) de K-Means

CONCLUSION

Le clustering peut être un outil très utile pour l'analyse des données dans un environnement non supervisé. Cependant, un certain nombre de problèmes surviennent lors de la mise en cluster par exemple comment choisir le meilleur k nombre de classe, Quelle mesure de dissimilarité faut-il utiliser, Où doit-on couper le dendrogramme pour obtenir des classes.

Chaque décision peut avoir un fort impact sur les résultats obtenus. En pratique, nous essayons plusieurs choix différents, et cherchons celui avec la solution la plus utile ou interprétable.