# Package de clustering de variables sous R

**Master 2**
**Statistiques et informatique pour la Science des données**


Olivier BOROT
Léo-Paul KNOEPFFLER
Perrine IBOUROI

Novembre 2025

# Sommaire

# Introduction

Ce rapport présente le développement du package R `M2RClust`, dédié au clustering de variables. Le package implémente trois algorithmes distincts : une approche par réallocation dynamique (K-Means), une approche hiérarchique divisive (inspiré du PROC VARCLUS du logiciel SAS), et une approche hybride (ACM + CAH) pour le clustering de modalités. Une architecture orientée objet (R6) a été adoptée pour garantir robustesse et évolutivité. Le projet inclut également une application Shiny.

Le clustering de variables est une technique exploratoire essentielle en science des données, visant à réduire la dimensionnalité en regroupant les variables redondantes ou fortement liées. Contrairement au clustering d'individus, l'objectif est d'identifier des structures latentes explicatives.

Nous présenterons donc dans un premier temps l'architecture générale et techniques du package, suivi d'une explication de chaque algorithmes avant de présenter l'interface

# Architecture générale du package

Pour répondre aux exigences du projet, nous avons développé le package via un système de Programmation Orientée Objet **R6**, choix motivé par plusieurs considérations techniques et pratiques.

## 1.1 Le système R6

Le développement du package s'appuie sur le système de programmation orientée objet **R6**. Cette section présente les différents systèmes OOP disponibles en R et décrit les caractéristiques de R6 exploitées dans notre implémentation.

### 1.1.1 Les systèmes OOP en R

R propose plusieurs paradigmes de programmation orientée objet, chacun avec ses spécificités. Le tableau 1.1 présente une comparaison des principales approches :

| **Caractéristique** | **S3** | **S4** | **R6** | **RC** |
|---|---|---|---|---|
| Sémantique de référence | Non | Non | Oui | Oui |
| Encapsulation (privé/public) | Non | Non | Oui | Partielle |
| Héritage multiple | Non | Oui | Non | Non |
| Validation formelle | Non | Oui | Non | Non |
| Performance | Moyenne | Faible | Élevée | Moyenne |
| Syntaxe intuitive | Oui | Non | Oui | Moyenne |

Table 1.1: Comparaison des systèmes OOP en R

- **S3** : Système historique de R, simple et flexible mais sans encapsulation formelle
- **S4** : Extension formalisée de S3, avec validation des slots et héritage multiple
- **R6** : Système moderne à sémantique de référence, proche des langages OOP classiques
- **RC** : Reference Classes, précurseur de R6 intégré à R de base

### 1.1.2 Caractéristiques de R6 utilisées dans le package

Le système R6 offre plusieurs fonctionnalités que nous avons exploitées dans l'implémentation du package :

1. **Sémantique de référence (mutabilité)** : Les objets R6 sont modifiés *in place*, ce qui permet d'enchaîner les opérations sans réassignation :

   ```
   clusterer <- KMeansClusterer$new(data = mtcars, n_clusters = 3)
   clusterer$fit()  # Modifie l'objet directement
   ```

   À titre de comparaison, avec S3/S4 il faudrait écrire : `clusterer <- fit(clusterer)`.

2. **Encapsulation stricte** : La distinction entre méthodes/attributs `public` et `private` permet de :
   - Exposer une API claire et stable aux utilisateurs
   - Masquer les détails d'implémentation internes
   - Prévenir les modifications accidentelles de l'état interne

3. **Héritage** : Le mot-clé `inherit` permet une extension naturelle des classes. Dans notre package, cela facilite la factorisation du code commun dans la classe `BaseClusterer`, dont héritent les trois algorithmes.

4. **Performance** : R6 présente de bonnes performances pour l'instanciation et l'appel de méthodes, ce qui est pertinent pour les algorithmes itératifs de clustering.

5. **Syntaxe** : La notation `objet$methode()` est similaire à celle des langages OOP classiques (Python, Java, C++), facilitant la lisibilité du code.

## 1.2 Architecture des classes

### 1.2.1 Diagramme de classes

La figure 1.1 présente l'architecture hiérarchique du package :

### 1.2.2 Classe abstraite BaseClusterer

La classe `BaseClusterer` définit le contrat que doivent respecter toutes les implémentations de clustering. Elle encapsule :

**Attributs publics**

- `data` : DataFrame des données d'entrée (variables en colonnes)
- `n_clusters` : Nombre de clusters cible
- `clusters` : Vecteur d'assignations (NULL avant `fit()`)
- `fitted` : Booléen indiquant si le modèle est ajusté
- `standardize` : Booléen pour la standardisation des données
- `max_iter`, `tol` : Paramètres de convergence

**Méthodes publiques**

| Méthode | Description |
|---|---|
| `fit()` | Ajuste le modèle aux données (abstraite) |
| `predict(new_data)` | Prédit les clusters pour de nouvelles variables |
| `get_results()` | Retourne un DataFrame avec les assignations |
| `get_cluster_members(k)` | Retourne les noms des variables du cluster k |
| `get_cluster_sizes()` | Retourne le nombre de variables par cluster |
| `save_results(file)` | Exporte les résultats en CSV |
| `print()`, `summary()` | Affichage formaté des résultats |

Table 1.2: Méthodes de la classe BaseClusterer

**Méthodes privées**

Les méthodes privées (préfixe `private$`) incluent :

- `validate_data()` : Vérifie la cohérence des données d'entrée
- `check_fitted()` : Lève une erreur si le modèle n'est pas ajusté
- `standardize_data()` : Applique la standardisation si requise

### 1.2.3 Polymorphisme et extensibilité

L'architecture permet d'ajouter facilement de nouveaux algorithmes. Pour créer un nouveau clusterer, il suffit de :

```
MonClusterer <- R6::R6Class("MonClusterer",
  inherit = BaseClusterer,
  public = list(
    fit = function() {
      # Implémentation spécifique
      self$fitted <- TRUE
```

```
      invisible(self)
    },
    predict = function(new_data) {
      # Logique de prédiction
    }
  )
)
```

Cette approche garantit que toutes les fonctions de visualisation et de validation (modules 05 et 06) fonctionnent automatiquement avec tout nouveau clusterer respectant l'interface.

## 1.3 Organisation modulaire

### 1.3.1 Structure des fichiers source

Le package adopte une organisation modulaire favorisant la séparation des responsabilités :

```
R/
|-- 00_utils.R               # Fonctions utilitaires bas niveau
|                            # (standardize_data, euclidean_distance,
|                            #  get_correlation_distance_matrix)
|-- 01_base_clusterer.R      # Classe abstraite BaseClusterer
|                            # Définit l'interface commune
|
|-- 02_kmeans_clusterer.R    # KMeansClusterer
|                            # Algorithme de réallocation dynamique
|                            # ~1300 lignes (complexité: initialisations multiples)
|
|-- 03_mca_hclust_cluster.R  # MCA_HClusterer
|                            # Clustering de modalités (ACM + CAH)
|                            # ~ lignes (a completer)
|
|-- 04_PDDP_clusterer.R      # DivisiveClusterer
|                            # Algorithme hiérarchique divisif
|                            # ~1100 lignes (gestion de l'historique)
|
|-- 05_cluster_validator.R   # Méthodes de sélection de K
|                            # (elbow, silhouette, Calinski-Harabasz)
|                            # ~420 lignes
|
|-- 06_visualization.R       # Fonctions de visualisation
|                            # (cercle des corrélations, heatmap, réseau)
|                            # ~860 lignes
|
```

```
|-- 07_shiny_app.R           # Application Shiny (à développer)
```

### 1.3.2 Gestion des dépendances

Les dépendances du package sont classées en deux catégories dans le fichier `DESCRIPTION` :

**Imports (obligatoires)**

- `R6` : Système de classes orientées objet
- `PCAmixdata` : ACP sur données mixtes (quantitatives + qualitatives)
- `cluster` : Calcul des silhouettes

**Suggests (optionnels)**

- `igraph` : Visualisation en graphe de réseau
- `FactoMineR` : ACM pour MCA_HClusterer
- `ggplot2` : Graphiques avancés (vignettes)
- `testthat` : Tests unitaires
- `knitr`, `rmarkdown` : Documentation

Cette distinction permet une installation minimale tout en offrant des fonctionnalités étendues aux utilisateurs disposant des packages optionnels.

## 1.4 Patrons de conception utilisés

### 1.4.1 Template Method (Méthode Gabarit)

Le patron *Template Method* est utilisé dans `BaseClusterer` pour définir le squelette des algorithmes :

```
# Dans BaseClusterer (simplifié)
initialize = function(data, n_clusters, ...) {
  private$validate_data()      # Hook 1 : validation
  if (self$standardize) {
    self$data <- private$standardize_data()  # Hook 2
  }
  # ... initialisation commune
}
```

Les sous-classes peuvent surcharger `validate_data()` pour ajouter des validations spécifiques (ex: `KMeansClusterer` accepte les facteurs, `BaseClusterer` non).

### 1.4.2 Strategy (Stratégie)

Le patron *Strategy* est utilisé pour les méthodes d'initialisation dans `KMeansClusterer` :

```
# Sélection dynamique de la stratégie d'initialisation
fit = function() {
  initial_clusters <- switch(private$init_method,
    "homogeneity++" = private$initialize_homogeneitypp(),
    "correlation"   = private$initialize_correlation_based(),
    "random"        = private$initialize_random()
  )
  # ... suite de l'algorithme
}
```

### 1.4.3 Lazy Loading (Chargement paresseux)

Pour optimiser les performances, certains calculs coûteux sont différés :

```
# Dans KMeansClusterer
get_plot_data = function() {
  if (!private$pca_global_computed) {
    private$compute_global_pca_if_needed()  # Calcul à la demande
  }
  return(list(coords = private$coords_fit, ...))
}
```

L'ACP globale pour la visualisation n'est calculée que si l'utilisateur appelle une fonction de visualisation, évitant un surcoût inutile si seules les assignations sont requises.

## 1.5 Gestion des erreurs et validation

### 1.5.1 Validation des entrées

Chaque classe implémente une validation rigoureuse des paramètres :

```
# Exemple dans BaseClusterer$initialize()
if (!is.data.frame(data) && !is.matrix(data)) {
  stop("data must be a data frame or matrix")
}
if (n_clusters < 2) {
  stop("n_clusters must be at least 2")
}
if (n_clusters >= ncol(data)) {
  stop("n_clusters must be strictly less than number of variables")
}
```

### 1.5.2 Vérification de l'état

La méthode privée `check_fitted()` est appelée systématiquement avant toute opération nécessitant un modèle ajusté :

```
get_homogeneity = function() {
  private$check_fitted()  # Lève une erreur si !self$fitted
  return(private$global_homogeneity)
}
```

### 1.5.3 Messages informatifs

Le package utilise `cat()` pour les messages de progression et `warning()` pour les situations non bloquantes :

```
# Progression pendant fit()
cat(sprintf("Fitting %s with K=%d clusters...\n",
            class(self)[1], self$n_clusters))

# Avertissement si convergence lente
if (iter == self$max_iter) {
  warning("Maximum iterations reached. Consider increasing max_iter.")
}
```

## 1.6 Tests et qualité du code

### 1.6.1 Couverture des tests

Le package inclut une suite de tests unitaires via `testthat` :

```
tests/testthat/
|-- test-01_base_clusterer.R      # ~290 lignes
|-- test-02_kmeans_clusterer.R    # ~450 lignes
|-- test-04_PDDP_clusterer.R      # ~700 lignes
|-- test-05_cluster_validator.R   # ~280 lignes
|-- test-06_visualization.R       # ~330 lignes
|-- test-integration.R            # ~270 lignes
```

Les tests couvrent :

- Instanciation avec paramètres valides/invalides
- Convergence des algorithmes
- Cohérence des résultats (reproductibilité avec `seed`)
- Cas limites (K=2, K=n_vars-1, données mixtes)
- Intégration entre modules

### 1.6.2 Standards de codage

Le code respecte le *Tidyverse Style Guide*, appliqué automatiquement via le package `styler` :

- Indentation : 2 espaces
- Longueur de ligne : max 80 caractères
- Nommage : `snake_case` pour fonctions et variables
- Documentation : Roxygen2 pour toutes les fonctions exportées

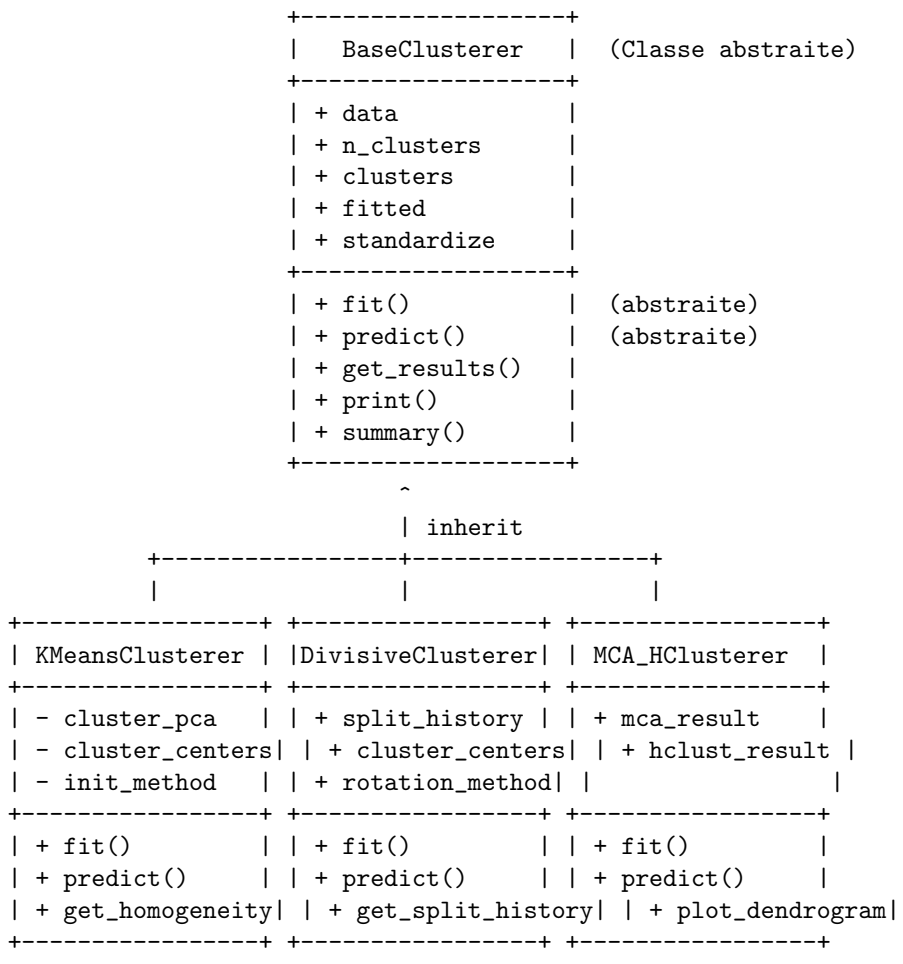


Figure 1.1: Diagramme de classes simplifié du package M2RClust

# 2.Méthodes de clustering de variables utilisées

Ce chapitre présente les trois algorithmes de clustering de variables implémentés dans le package. Chaque méthode répond à des besoins spécifiques selon le type de données et l'objectif de l'analyse.

## 2.1 Clustering par réallocation dynamique (KMeansClusterer)

L'algorithme `KMeansClusterer` est inspiré de la fonction `kmeansvar()` du package `ClustOfVar` [2]. Il s'agit d'une adaptation du K-Means classique au clustering de variables, où l'objectif est de maximiser l'**homogénéité** au sein de chaque cluster.

### 2.1.1 Principe fondamental

Contrairement au K-Means classique qui minimise l'inertie intra-cluster sur des individus, cette méthode travaille sur les **variables** et maximise l'homogénéité de chaque groupe. L'homogénéité d'un cluster est définie comme la proportion de variance expliquée par sa première composante principale :

$$H_k = \frac{\lambda_1^{(k)}}{\sum_{j=1}^{p_k} \lambda_j^{(k)}} \tag{2.1}$$

où $\lambda_j^{(k)}$ représente la $j$-ème valeur propre de l'ACP réalisée sur les $p_k$ variables du cluster $k$.

Le critère global à maximiser est l'homogénéité pondérée :

$$H_{global} = \sum_{k=1}^{K} \frac{p_k}{p} \cdot H_k \tag{2.2}$$

où $p$ est le nombre total de variables et $p_k$ le nombre de variables dans le cluster $k$.

### 2.1.2 Description de l'algorithme

L'algorithme procède par itérations successives en deux étapes (figure 2.1) :

**Algorithme : KMeansVar pour le clustering de variables**

**Entrée :** Données $\mathbf{X}$ $(n \times p)$, nombre de clusters $K$
**Sortie :** Partition $\mathcal{P} = \{C_1, \ldots, C_K\}$ des variables

1. **Initialisation :** Créer une partition initiale des $p$ variables en $K$ clusters
2. **Répéter** jusqu'à convergence :
   (a) **Étape de représentation :** Pour chaque cluster $C_k$, calculer la première composante principale $\mathbf{c}_k$ (centre du cluster)
   (b) **Étape d'affectation :** Pour chaque variable $X_j$, l'affecter au cluster $k^*$ dont le centre maximise l'association :
   $$k^* = \arg\max_k \rho^2(X_j, \mathbf{c}_k)$$
3. **Retourner** la partition finale et l'homogénéité globale

Figure 2.1: Pseudo-code de l'algorithme KMeansVar

#### Étape de représentation

Pour chaque cluster $C_k$, on réalise une ACP sur les variables qu'il contient. Le **centre du cluster** est défini comme le vecteur des scores de la première composante principale (PC1). Ce centre représente la direction de variance maximale commune aux variables du cluster.

```
# Pseudo-code R
for (k in 1:K) {
  data_cluster <- data[, clusters == k]
  pca_k <- prcomp(data_cluster, scale. = TRUE)
  center_k <- pca_k$x[, 1]  # Scores PC1
}
```

#### Étape d'affectation

Chaque variable est réaffectée au cluster dont le centre présente la plus forte association avec elle. La mesure d'association dépend du type de variable :

- **Variable quantitative** : Coefficient de détermination $R^2$ (corrélation au carré)
  $$\rho^2(X_j, \mathbf{c}_k) = \text{cor}(X_j, \mathbf{c}_k)^2$$

- **Variable qualitative** : Rapport de corrélation $\eta^2$

$$\eta^2(X_j, \mathbf{c}_k) = \frac{SS_{between}}{SS_{total}} = \frac{\sum_m n_m(\bar{c}_{k,m} - \bar{c}_k)^2}{\sum_i (c_{k,i} - \bar{c}_k)^2}$$

où $m$ parcourt les modalités de $X_j$ et $\bar{c}_{k,m}$ est la moyenne de $\mathbf{c}_k$ sur les individus de la modalité $m$.

### 2.1.3 Support des données mixtes

Une originalité de notre implémentation est le support natif des **données mixtes** (variables quantitatives et qualitatives simultanément). Pour cela, nous utilisons le package `PCAmixdata` [1] qui généralise l'ACP aux données mixtes.

| Type de paire | Mesure d'association | Formule | Interprétation |
|---|---|---|---|
| Quanti - Quanti | $R^2$ de Pearson | $\text{cor}(X_i, X_j)^2$ | Corrélation linéaire |
| Quali - Quali | $V^2$ de Cramér | $\frac{\chi^2}{n \cdot \min(r-1, s-1)}$ | Association catégorielle |
| Quanti - Quali | $\eta^2$ | $\frac{SS_{between}}{SS_{total}}$ | Rapport de corrélation |

Table 2.1: Mesures d'association utilisées selon le type de variables

### 2.1.4 Stratégies d'initialisation

La qualité de la solution finale dépend fortement de l'initialisation. Trois stratégies sont proposées :

#### Initialisation aléatoire (`random`)

Les variables sont affectées aléatoirement aux $K$ clusters. Simple mais sensible aux minima locaux.

#### Initialisation par corrélation (`correlation`)

Une CAH (Classification Ascendante Hiérarchique) est d'abord réalisée sur la matrice de dissimilarité des variables, puis coupée en $K$ groupes. La dissimilarité est calculée comme :

$$d(X_i, X_j) = \sqrt{1 - \rho^2(X_i, X_j)}$$

Cette méthode est **déterministe** et produit toujours la même partition initiale.

#### Initialisation homogeneity++ (`homogeneity++`)

Inspirée de la méthode K-Means++, cette stratégie sélectionne les centres initiaux de manière à maximiser leur diversité. La première variable est choisie aléatoirement, puis chaque variable suivante est sélectionnée avec une probabilité proportionnelle au carré de sa distance au centre le plus proche.

### 2.1.5 Exécutions multiples et arrêt précoce

Pour éviter les minima locaux, l'algorithme est exécuté `n_init` fois avec des initialisations différentes. Un mécanisme d'**arrêt précoce** stoppe les exécutions si le même score optimal est atteint deux fois consécutivement, supposant que l'optimum global a été trouvé.

```
# Stratégie d'arrêt précoce
for (run in 1:n_init) {
  result <- run_clustering_once()
  if (result$homogeneity > best_homogeneity) {
    best_homogeneity <- result$homogeneity
    best_clusters <- result$clusters
    times_found <- 1
  } else if (result$homogeneity == best_homogeneity) {
    times_found <- times_found + 1
    if (times_found >= 2) break  # Arrêt précoce
  }
}
```

### 2.1.6 Critère de convergence

L'algorithme converge lorsque l'une des conditions suivantes est satisfaite :

1. **Stabilité de la partition :** Aucune variable ne change de cluster entre deux itérations
2. **Stabilité de l'homogénéité :** $|H_{global}^{(t)} - H_{global}^{(t-1)}| < \epsilon$ (tolérance)
3. **Nombre maximal d'itérations :** `max_iter` atteint

### 2.1.7 Complexité algorithmique

La complexité par itération est dominée par les ACP locales :

- **Étape de représentation :** $O(K \cdot n \cdot \bar{p}^2)$ où $\bar{p} = p/K$ est le nombre moyen de variables par cluster
- **Étape d'affectation :** $O(p \cdot K \cdot n)$
- **Complexité totale :** $O(T \cdot R \cdot (K \cdot n \cdot p^2/K^2 + p \cdot K \cdot n))$

où $T$ est le nombre d'itérations, $R$ le nombre d'exécutions (`n_init`), $n$ le nombre d'individus, $p$ le nombre de variables et $K$ le nombre de clusters.

### 2.1.8 Exemple d'utilisation

```
library(M2RClust)

# Chargement des données
data(mtcars)

# Création du clusterer
clusterer <- KMeansClusterer$new(
  data = mtcars,
  n_clusters = 3,
  standardize = TRUE,
  init_method = "homogeneity++",
  n_init = 10,
  seed = 42
)

# Ajustement
clusterer$fit()

# Résultats
clusterer$summary()
# Global Homogeneity: 0.847
# Cluster 1 (4 vars): mpg, drat, am, gear - Homogeneity: 0.72
# Cluster 2 (3 vars): cyl, disp, hp - Homogeneity: 0.91
# Cluster 3 (4 vars): wt, qsec, vs, carb - Homogeneity: 0.65

# Visualisation
plot_clustering_2d(clusterer)
```

## 2.2 Clustering hiérarchique divisif (DivisiveClusterer)

L'algorithme `DivisiveClusterer` est inspiré de la procédure `PROC VARCLUS` du logiciel SAS [3]. Contrairement à l'approche agglomérative (ascendante) classique, cette méthode utilise une stratégie **divisive** (descendante) où l'on part d'un unique cluster contenant toutes les variables, puis on le divise itérativement.

### 2.2.1 Principe de l'algorithme divisif

L'approche divisive présente plusieurs avantages pour le clustering de variables :

- **Déterminisme** : Pas d'initialisation aléatoire, résultats 100% reproductibles

- **Interprétabilité** : Hiérarchie explicite des divisions
- **Critères d'arrêt naturels** : Basés sur les valeurs propres (règle de Kaiser)
- **Stabilité** : Contrairement au K-Means, chaque exécution produit le même résultat

Le principe repose sur l'analyse des valeurs propres de l'ACP locale :

- $\lambda_1$ : Première valeur propre, représente la variance expliquée par la direction principale
- $\lambda_2$ : Seconde valeur propre, indique l'hétérogénéité résiduelle du cluster

Un cluster avec $\lambda_2$ élevé est **hétérogène** : il contient des variables qui ne partagent pas la même direction principale et mérite d'être divisé.

### 2.2.2 Description détaillée de l'algorithme

#### Étape de sélection du cluster à diviser

À chaque itération, on sélectionne le cluster le plus **hétérogène**, c'est-à-dire celui dont la seconde valeur propre est maximale :

$$k^* = \arg\max_k \lambda_2^{(k)} \tag{2.3}$$

Une seconde valeur propre élevée indique qu'une direction secondaire capture une part significative de la variance, suggérant la présence de deux sous-groupes distincts.

#### Rotation Varimax et division

Après extraction des deux premières composantes, une **rotation Varimax** est appliquée pour maximiser la séparation des variables entre les deux facteurs rotés. La rotation Varimax maximise la variance des carrés des loadings :

$$V = \sum_{j=1}^{2} \left[ \frac{1}{p_k} \sum_{i=1}^{p_k} a_{ij}^4 - \left( \frac{1}{p_k} \sum_{i=1}^{p_k} a_{ij}^2 \right)^2 \right] \tag{2.4}$$

où $a_{ij}$ est le loading de la variable $i$ sur le facteur rotaté $j$.

Chaque variable est ensuite affectée au sous-groupe correspondant au facteur rotaté avec lequel elle présente la plus forte corrélation carrée :

$$\text{Groupe}(X_i) = \begin{cases} 1 & \text{si } a_{i1}^2 > a_{i2}^2 \\ 2 & \text{sinon} \end{cases} \tag{2.5}$$

**Algorithme : Clustering divisif de variables (VARCLUS)**

**Entrée :** Données $\mathbf{X}$ $(n \times p)$, nombre de clusters cible $K$
**Sortie :** Partition hiérarchique $\mathcal{P} = \{C_1, \ldots, C_K\}$ des variables

1. **Initialisation :** Placer toutes les $p$ variables dans un unique cluster $C_1$
2. **Répéter** tant que le nombre de clusters $< K$ :
   (a) **Sélection :** Identifier le cluster $C_k$ avec la plus grande seconde valeur propre $\lambda_2^{(k)}$
   (b) **Vérification des critères d'arrêt :**
      - Si $\lambda_2^{(k)} < 1$ (critère de Kaiser) : STOP
      - Si $\lambda_2^{(k)}/\lambda_1^{(k)} < \theta$ (ratio minimum) : STOP
   (c) **ACP locale :** Calculer les deux premières composantes principales sur $C_k$
   (d) **Rotation Varimax :** Appliquer une rotation orthogonale aux deux composantes
   (e) **Division :** Affecter chaque variable au sous-groupe dont le facteur rotaté a la plus forte corrélation carrée
3. **Retourner** la partition finale et l'historique des divisions

Figure 2.2: Pseudo-code de l'algorithme de clustering divisif

### 2.2.3 Critères d'arrêt

L'algorithme propose plusieurs critères pour déterminer automatiquement le nombre optimal de clusters :

#### Critère de Kaiser

Le critère de Kaiser stipule qu'une composante est significative si sa valeur propre est supérieure à 1 (pour des données standardisées). Appliqué au clustering :

$$\text{Si } \max_k \lambda_2^{(k)} < 1 \Rightarrow \text{Arrêt} \tag{2.6}$$

Ce critère indique que plus aucun cluster ne présente d'hétérogénéité suffisante pour justifier une division.

**Critère du ratio de valeurs propres**

Un critère alternatif compare le ratio $\lambda_2/\lambda_1$ :

$$\text{Si } \frac{\lambda_2^{(k^*)}}{\lambda_1^{(k^*)}} < \theta \Rightarrow \text{Arrêt} \tag{2.7}$$

où $\theta$ est un seuil configurable (par défaut 0.1). Ce critère est plus flexible que Kaiser et s'adapte mieux aux données non standardisées.

### 2.2.4 Implémentation hybride

Notre implémentation détecte automatiquement le type de données et utilise le chemin de calcul optimal :

| Type de données | Méthode de calcul | Complexité |
|---|---|---|
| Purement quantitatif | `eigen(cor(X))` | $O(p^2)$ par cluster |
| Mixte (quanti + quali) | `PCAmix` | $O(p^2 \cdot m)$ par cluster |

Table 2.2: Chemins de calcul selon le type de données

Pour les données purement numériques, la matrice de corrélation globale est pré-calculée une seule fois, puis on extrait les sous-matrices pour chaque cluster :

```
# Chemin rapide (données numériques)
global_cor <- cor(data)  # Calculé une fois

# Pour diviser le cluster k
cor_sub <- global_cor[vars_in_k, vars_in_k]
eigen_result <- eigen(cor_sub, symmetric = TRUE)
```

### 2.2.5 Historique des divisions

Un avantage majeur de l'approche divisive est la conservation de l'**historique des divisions**, permettant une interprétation hiérarchique des résultats :

```
# Accès à l'historique
history <- clusterer$get_split_history()

# Chaque entrée contient :
# - iteration       : numéro de la division
# - split_cluster   : cluster qui a été divisé
# - eigenvalue_1    : lambda1 avant division
# - eigenvalue_2    : lambda2 avant division
# - eigenvalue_ratio: lambda2/lambda1
# - variables_group1: variables du premier sous-groupe
# - variables_group2: variables du second sous-groupe
# - split_quality   : score de qualité de la division
```

### 2.2.6 Mesure de qualité de division

Pour chaque division, un score de qualité similaire au silhouette est calculé :

$$Q = \frac{\bar{r}_{intra} - \bar{r}_{inter}}{\bar{r}_{intra} + \bar{r}_{inter}} \tag{2.8}$$

où $\bar{r}_{intra}$ est la corrélation moyenne intra-groupe et $\bar{r}_{inter}$ la corrélation moyenne inter-groupe. Un score proche de 1 indique une division de haute qualité (forte cohésion interne, faible liaison entre groupes).

### 2.2.7 Comparaison avec KMeansClusterer

| **Caractéristique** | **KMeansClusterer** | **DivisiveClusterer** |
|---|---|---|
| Stratégie | Partitionnement direct | Hiérarchique divisive |
| Déterminisme | Non (sauf `correlation`) | Oui |
| Sensibilité à l'initialisation | Forte | Aucune |
| Arrêt automatique | Non | Oui (Kaiser, ratio) |
| Historique des étapes | Non | Oui |
| Visualisation dendrogramme | Non | Oui |
| Complexité par itération | $O(K \cdot n \cdot p)$ | $O(p^2)$ |
| Optimum | Local (multi-start) | Hiérarchique |

Table 2.3: Comparaison des deux algorithmes de clustering

### 2.2.8 Exemple d'utilisation

```
library(M2RClust)

# Chargement des données
data(mtcars)

# Clustering divisif avec critère de Kaiser
clusterer <- DivisiveClusterer$new(
  data = mtcars,
  n_clusters = 10,        # Maximum (arrêt automatique probable avant)
  standardize = TRUE,
  rotation_method = "varimax",
  stop_at_kaiser = TRUE,  # Arrêt si lambda2 < 1
  min_cluster_size = 2
)

# Ajustement
clusterer$fit()
# Data mode: NUMERIC (fast correlation path)
# Kaiser criterion: Stopping at 4 clusters (max lambda2 = 0.892 < 1.0)
```

```
# Divisive clustering completed: 4 clusters formed in 3 iterations

# Résultats
clusterer$summary()

# Historique des divisions
history <- clusterer$get_split_history()
print(history[[1]])
# $iteration: 1
# $split_cluster: 1
# $eigenvalue_1: 6.12
# $eigenvalue_2: 2.41
# $eigenvalue_ratio: 0.394
# $variables_group1: c("mpg", "cyl", "disp", "hp", "wt")
# $variables_group2: c("drat", "qsec", "vs", "am", "gear", "carb")

# Visualisation du dendrogramme divisif
clusterer$plot_split_dendrogram()

# Cercle des corrélations
plot_clustering_2d(clusterer)
```

### 2.2.9 Visualisations spécifiques

L'algorithme divisif offre des visualisations spécifiques :

- **Dendrogramme divisif** : Arbre descendant montrant l'historique des divisions avec les ratios $\lambda_2/\lambda_1$
- **Cercle des corrélations** : Variables colorées par cluster avec indication de l'homogénéité
- **Heatmap ordonnée** : Matrice de corrélation réordonnée par cluster

## 2.3 Clustering de modalités (MCA_HClusterer)

L'algorithme `MCA_HClusterer` (également appelé `ModalitiesDiceClusterer`) propose une approche originale en se concentrant sur les **modalités** (niveaux des variables catégorielles) plutôt que sur les variables elles-mêmes. Cette méthode combine l'Analyse des Correspondances Multiples (ACM) avec une Classification Ascendante Hiérarchique (CAH) pour regrouper les modalités similaires.

### 2.3.1 Principe fondamental

Contrairement aux deux algorithmes précédents qui groupent des variables, cette méthode travaille au niveau des modalités individuelles. L'objectif est

d'identifier des profils de modalités qui co-occurrent fréquemment ou qui sont structurellement similaires dans leur distribution.

#### Représentation disjonctive

La première étape consiste à transformer les données catégorielles en une **matrice disjonctive complète** (ou tableau disjonctif). Pour une variable $X$ avec $m$ modalités, on crée $m$ colonnes binaires :

$$Y_{ij}^{(k)} = \begin{cases} 1 & \text{si l'individu } i \text{ possède la modalité } k \text{ de la variable } j \\ 0 & \text{sinon} \end{cases} \tag{2.9}$$

Par exemple, une variable "Couleur" avec les modalités {Rouge, Bleu, Vert} génère trois colonnes : `Couleur.Rouge`, `Couleur.Bleu`, `Couleur.Vert`.

#### Discrétisation automatique

Pour traiter des données mixtes, l'algorithme propose une **discrétisation automatique** des variables quantitatives en les transformant en `n_bins` classes basées sur les quantiles. Cette approche permet d'appliquer la méthode à des jeux de données comportant à la fois des variables qualitatives et quantitatives.

### 2.3.2 Mesures de dissimilarité entre modalités

L'algorithme propose deux mesures de dissimilarité pour quantifier la distance entre modalités :

#### Dissimilarité de Dice (par défaut)

La dissimilarité de Dice est calculée à partir des vecteurs binaires de la matrice disjonctive. Pour deux modalités $j$ et $k$ représentées par les colonnes binaires $\mathbf{y}_j$ et $\mathbf{y}_k$, la distance de Dice au carré est définie comme :

$$d^2_{Dice}(j,k) = \frac{1}{2}\left(\frac{1}{n_j} + \frac{1}{n_k}\right)\sum_{i=1}^{n}(y_{ij} - y_{ik})^2 \tag{2.10}$$

où $n_j = \sum_{i=1}^{n} y_{ij}$ est l'effectif de la modalité $j$.

Cette mesure présente l'avantage de pondérer les différences par les fréquences des modalités, donnant plus d'importance aux modalités rares.

#### Dissimilarité basée sur le V de Cramér

Alternativement, on peut utiliser le coefficient de Cramér transformé en dissimilarité. Pour deux modalités, on construit un tableau de contingence $2 \times 2$ et on calcule :

$$V^2 = \frac{\chi^2}{n \cdot \min(r-1, c-1)} \tag{2.11}$$

La dissimilarité est alors définie comme :

$$d^2_{Cramer}(j,k) = 1 - V^2(j,k) \tag{2.12}$$

### 2.3.3 Classification Ascendante Hiérarchique

Une fois la matrice de dissimilarité $\mathbf{D}$ calculée entre toutes les paires de modalités, l'algorithme applique une CAH classique avec la méthode de Ward (ou toute autre méthode de linkage spécifiée). La figure 2.3 présente l'algorithme complet.

**Algorithme : Clustering de modalités (MCA_HClusterer)**

**Entrée :** Données catégorielles $\mathbf{X}$ $(n \times p)$, nombre de groupes $K$, méthode de linkage
**Sortie :** Partition $\mathcal{P} = \{G_1, \ldots, G_K\}$ des modalités

1. **Transformation disjonctive :** Si `auto_discretize = TRUE`, discrétiser les variables numériques. Créer la matrice disjonctive $\mathbf{Y}$ $(n \times m)$, où $m$ est le nombre total de modalités.
2. **Calcul de la dissimilarité :** Pour chaque paire de modalités $(j,k)$, calculer $d^2(j,k)$ selon la mesure choisie (Dice ou Cramér). Construire la matrice $\mathbf{D}$ $(m \times m)$.
3. **Clustering hiérarchique :** Appliquer `hclust()` sur $\mathbf{D}$ avec la méthode de linkage spécifiée. Couper le dendrogramme en $K$ groupes.
4. **Calcul des statistiques :** Pour chaque groupe $G_k$, calculer le centre, la décomposition de l'inertie et les contributions des modalités.
5. **Retourner** la partition des modalités, les statistiques et l'objet `hclust`.

Figure 2.3: Pseudo-code de l'algorithme de clustering de modalités

### 2.3.4 Statistiques et diagnostics

L'implémentation calcule et met en cache plusieurs statistiques pour faciliter l'interprétation des résultats.

**Centres des groupes**

Pour chaque groupe $G_k$, le centre est défini comme le barycentre des vecteurs binaires des modalités du groupe :

$$\mathbf{c}_k = \frac{1}{|G_k|} \sum_{j \in G_k} \mathbf{y}_j \tag{2.13}$$

Ce vecteur représente le "profil moyen" des individus possédant les modalités du groupe.

**Décomposition de l'inertie**

L'inertie totale est décomposée en inertie intra-groupes et inter-groupes :

$$I_{total} = \sum_{j=1}^{m} n_j \cdot ||\mathbf{y}_j - \bar{\mathbf{y}}||^2 \tag{2.14}$$

$$I_{intra} = \sum_{k=1}^{K} \sum_{j \in G_k} n_j \cdot ||\mathbf{y}_j - \mathbf{c}_k||^2 \tag{2.15}$$

$$I_{inter} = \sum_{k=1}^{K} |G_k| \cdot ||\mathbf{c}_k - \bar{\mathbf{y}}||^2 \tag{2.16}$$

où $\bar{\mathbf{y}}$ est le barycentre global. Le critère de qualité est $R^2 = I_{inter}/I_{total}$. Un $R^2$ élevé indique que les groupes sont bien séparés.

**Contributions des modalités**

La contribution d'une modalité $j$ à l'inertie de son groupe $G_k$ est :

$$CTR_j = \frac{n_j \cdot ||\mathbf{y}_j - \mathbf{c}_k||^2}{I_{intra}(G_k)} \tag{2.17}$$

Les modalités avec les contributions les plus élevées sont les plus "typiques" de leur groupe.

### 2.3.5 Analyse des Correspondances Multiples (ACM)

Pour faciliter la visualisation, l'algorithme intègre une ACM sur la matrice disjonctive. L'ACM généralise l'Analyse Factorielle des Correspondances aux tableaux à plus de deux variables.

**Principe de l'ACM**

L'ACM réalise une décomposition en valeurs propres de la matrice :

$$\mathbf{M} = \mathbf{D}_n^{-1}\mathbf{Y}^T\mathbf{D}_n^{-1}\mathbf{Y}\mathbf{D}_m^{-1} \tag{2.18}$$

où $\mathbf{D}_n$ et $\mathbf{D}_m$ sont des matrices diagonales de pondération.

Les coordonnées des modalités sur les axes factoriels permettent une visualisation en 2D ou 3D qui préserve au mieux les distances entre modalités, ce qui n'est pas le cas avec une éventuelle projection de Dice.

**Projection de variables illustratives**

L'algorithme permet de projeter des variables illustratives (supplémentaires) dans l'espace factoriel sans modifier la solution. Pour une modalité illustrative $\mathbf{y}_{illus}$, les coordonnées sont calculées comme :

$$\mathbf{coord}_{illus} = \mathbf{D}_n^{-1}\mathbf{Y}^T\mathbf{y}_{illus} \tag{2.19}$$

Cette fonctionnalité est utile pour :

- Valider la stabilité du clustering sur de nouvelles données
- Identifier à quel groupe une nouvelle modalité serait associée
- Analyser la position d'une variable non incluse dans le clustering

On peut aussi les projeter sur un plan factoriel pour observer leurs similarités avec les autres modalités.

### 2.3.6 Méthodes de recoupage

Une particularité intéressante de cette implémentation est la méthode `cut_tree()` qui permet de **recouper** le dendrogramme en un nombre différent de groupes sans recalculer les distances ni refaire la CAH :

```
# Recoupage rapide
clusterer$fit(data)         # K = 3 (par défaut)
clusterer$cut_tree(5)       # Passer à K = 5
clusterer$summary()         # Nouvelles statistiques calculées
```

Cette approche est très efficace pour explorer différentes valeurs de $K$ et identifier le nombre optimal de groupes.

### 2.3.7 Complexité algorithmique

La complexité totale de l'algorithme, bien qu'efficace sur la plupart des jeux de données, peut s'avérer lent sur de très grand jeux de données, et est dominée par plusieurs étapes :

- **Création de la matrice disjonctive :** $O(n \cdot p)$ où $n$ est le nombre d'individus et $p$ le nombre de variables
- **Calcul de la matrice de dissimilarité :** $O(m^2 \cdot n)$ où $m$ est le nombre total de modalités
- **CAH :** $O(m^2 \log m)$ avec des structures de données optimisées
- **ACM (si visualisation) :** $O(n \cdot m^2)$ pour la décomposition SVD

La complexité globale est donc $O(n \cdot m^2 + m^2 \log m)$, ce qui reste raisonnable pour des jeux de données de taille moyenne, mais peut s'avérer lent pour un jeu de données très large.

### 2.3.8 Exemple d'utilisation

```
library(M2RClust)

# Données mixtes (catégorielles + numériques)
data(mtcars)
df <- data.frame(
  cyl = factor(mtcars$cyl),
  am = factor(mtcars$am),
  vs = factor(mtcars$vs),
  mpg = mtcars$mpg,
  hp = mtcars$hp
)

# Création du clusterer avec discrétisation automatique
clusterer <- ModalitiesDiceClusterer$new(
  n_groups = 4,
  linkage = "ward.D2",
  dissimilarity = "dice",
  auto_discretize = TRUE,
  n_bins = 4
)

# Ajustement
clusterer$fit(df)

# Résultats
clusterer$summary()
# Groupe 1 (5 modalités): cyl.4, am.1, mpg_Q4, hp_Q1, vs.1
#   - Inertie intra: 0.15
#   - Contributions principales: cyl.4 (35%), am.1 (28%)
#
# Groupe 2 (4 modalités): cyl.8, mpg_Q1, hp_Q4, vs.0
```

```
#   - Inertie intra: 0.12
#   - Contributions principales: cyl.8 (42%), hp_Q4 (31%)
# ...

# Visualisations
clusterer$plot_dendrogram()              # Dendrogramme avec rectangles
clusterer$plot_mca()                     # Projection ACM des modalités
clusterer$plot_clusters(add_ellipses = TRUE)  # Modalités colorées par groupe

# Table des résultats
results <- clusterer$get_cluster_table()
print(results)
#   modality       cluster frequency
# 1 cyl.4              1        11
# 2 cyl.6              3         7
# 3 cyl.8              2        14
# ...

# Projection d'une variable illustrative
illus_var <- factor(ifelse(mtcars$gear > 3, "high_gear", "low_gear"))
proj <- clusterer$predict_illustrative(illus_var)
print(proj$by_group)  # Distance moyenne par groupe

# Explorer différentes valeurs de K
clusterer$cut_tree(6)
clusterer$summary()
```

### 2.3.9 Implémentation R6 et architecture

L'implémentation suit les mêmes principes architecturaux que les autres clusterers du package :

```
ModalitiesDiceClusterer <- R6::R6Class(
  "ModalitiesDiceClusterer",
  public = list(
    # Champs publics
    n_groups = NULL,
    linkage = NULL,
    dissimilarity = NULL,
    disj = NULL,      # Matrice disjonctive
    d2 = NULL,        # Matrice de dissimilarité au carré
    hclust = NULL,    # Objet hclust
    groups = NULL,    # Assignation des modalités
    fitted = FALSE,

    # Méthodes principales
```

```
    initialize = function(...) { ... },
    fit = function(data) { ... },
    cut_tree = function(k) { ... },
    predict_illustrative = function(illus) { ... },

    # Méthodes de visualisation
    plot_dendrogram = function() { ... },
    plot_mca = function(...) { ... },
    plot_clusters = function(...) { ... },

    # Accesseurs
    get_cluster_table = function() { ... },
    get_dice_matrix = function(...) { ... },
    summary = function() { ... }
  ),
  private = list(
    # Méthodes privées pour calculs internes
    compute_dice_matrix = function() { ... },
    compute_cramer_matrix = function() { ... },
    discretize_numeric = function(...) { ... },
    compute_group_stats = function() { ... },
    compute_mca = function() { ... }
  )
)
```

# 3.Fonctions transverses

Ce chapitre présente les modules auxiliaires du package qui fournissent des fonctionnalités communes à tous les algorithmes de clustering : fonctions utilitaires bas niveau, méthodes de sélection du nombre de clusters, et outils de visualisation.

## 3.1 Fonctions utilitaires (module 00_utils)

Le module `00_utils.R` regroupe les fonctions mathématiques et de manipulation de données utilisées par les algorithmes de clustering.

### 3.1.1 Standardisation des données

La standardisation est une étape cruciale pour le clustering de variables, car elle permet de rendre comparables des variables mesurées dans des unités différentes.

**Fonction `standardize_data()`**

Cette fonction centre et réduit les données :

$$z_{ij} = \frac{x_{ij} - \bar{x}_j}{s_j} \tag{3.1}$$

où $\bar{x}_j$ est la moyenne de la variable $j$ et $s_j$ son écart-type.

```
standardize_data <- function(data, center = TRUE, scale = TRUE) {
  scaled_data <- scale(data, center = center, scale = scale)

  # Sauvegarde des paramètres pour application ultérieure
  result <- as.data.frame(scaled_data)
  attr(result, "centers") <- attr(scaled_data, "scaled:center")
  attr(result, "scales") <- attr(scaled_data, "scaled:scale")

  return(result)
}
```

**Fonction** `apply_standardization()`

Permet d'appliquer les paramètres de standardisation calculés sur les données d'entraînement à de nouvelles données (variables supplémentaires) :

```
apply_standardization <- function(data, centers, scales)
```

Cette fonction est essentielle pour la méthode `predict()` des clusterers, garantissant que les nouvelles variables sont transformées de manière cohérente.

### 3.1.2 Matrice de distances basée sur les corrélations

La fonction `get_correlation_distance_matrix()` calcule une matrice de distances euclidiennes entre variables, adaptée au clustering de variables.

#### Cas des données quantitatives

Pour deux variables quantitatives $X_i$ et $X_j$, la distance est définie par :

$$d(X_i, X_j) = \sqrt{2(1 - r_{ij})} \tag{3.2}$$

où $r_{ij}$ est le coefficient de corrélation de Pearson. Cette transformation garantit que la distance est euclidienne et varie entre 0 (corrélation parfaite) et $\sqrt{2}$ (corrélation parfaitement négative).

#### Cas des données mixtes

Pour les données mixtes, la fonction utilise des mesures d'association appropriées :

| **Type de paire** | **Mesure** | **Formule de similarité** |
|---|---|---|
| Quanti-Quanti | Corrélation de Pearson | $sim = r^2$ |
| Quali-Quali | V de Cramér | $sim = V^2 = \frac{\chi^2}{n \cdot \min(r-1, c-1)}$ |
| Quanti-Quali | Rapport de corrélation | $sim = \eta^2 = \frac{SS_{between}}{SS_{total}}$ |

Table 3.1: Mesures d'association pour données mixtes

La distance finale est calculée par $d = \sqrt{2(1 - sim)}$, garantissant une métrique euclidienne cohérente quel que soit le type de variables.

### 3.1.3 Distance euclidienne classique

La fonction `euclidean_distance()` calcule les distances entre des points et des centres de clusters :

```
euclidean_distance <- function(points, centers) {
  # Retourne une matrice [n_centers x n_points]
  # Element [i,j] = distance du centre i au point j
}
```

Cette fonction est utilisée en interne par certaines méthodes d'initialisation.

## 3.2 Sélection du nombre de clusters (module 05)

Le choix du nombre optimal de clusters $K$ est un problème fondamental en clustering. Le module `05_cluster_validator.R` implémente trois méthodes complémentaires.

### 3.2.1 Méthode du coude (Elbow)

#### Principe

La méthode du coude trace l'évolution d'un critère de qualité en fonction de $K$ et identifie le point d'inflexion ( coude ) où l'amélioration marginale devient négligeable.

Pour le clustering de variables, le critère utilisé est $(1-H_{global})$ où $H_{global}$ est l'homogénéité globale. Ce critère décroît avec $K$ (plus de clusters = meilleure homogénéité).

#### Détection automatique du coude

L'algorithme identifie le coude comme le point le plus éloigné de la droite reliant le premier et le dernier point de la courbe :

$$K^* = \arg\max_k \frac{|(y_n - y_1) \cdot k - (K_n - K_1) \cdot y_k + K_n \cdot y_1 - y_n \cdot K_1|}{\sqrt{(y_n - y_1)^2 + (K_n - K_1)^2}} \tag{3.3}$$

```
elbow_method <- function(clusterer_class, data, k_range = 2:10,
                         plot = TRUE, ...)
```

### 3.2.2 Méthode de la silhouette

#### Principe

Le coefficient de silhouette mesure la cohésion intra-cluster et la séparation inter-cluster. Pour chaque variable $i$ :

$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))} \tag{3.4}$$

où :

- $a(i)$ : distance moyenne de $i$ aux autres variables de son cluster
- $b(i)$ : distance moyenne minimale de $i$ aux variables des autres clusters

**Adaptation au clustering de variables**

La silhouette utilise la matrice de distances basée sur les corrélations (équation 3.2), ce qui est plus approprié pour le clustering de variables que les distances euclidiennes classiques.

```
silhouette_method <- function(clusterer_class, data, k_range = 2:10,
                              plot = TRUE, ...)
```

Le $K$ optimal maximise la silhouette moyenne. Un coefficient proche de 1 indique des clusters bien définis, proche de 0 des clusters chevauchants, et négatif des assignations incorrectes.

### 3.2.3 Indice de Calinski-Harabasz (Pseudo-F)

**Principe**

L'indice de Calinski-Harabasz équilibre qualité du clustering et parcimonie en pénalisant le nombre de clusters :

$$CH(K) = \frac{(H_{total} - 1)/(K - 1)}{(p - H_{total})/(p - K)} \tag{3.5}$$

où $H_{total} = H_{global} \times p$ est l'inertie expliquée totale et $p$ le nombre de variables.

```
calinski_harabasz_method <- function(clusterer_class, data,
                                     k_range = 2:10, plot = TRUE, ...)
```

Le $K$ optimal maximise cet indice. Cette méthode tend à favoriser des partitions plus parcimonieuses.

### 3.2.4 Comparaison et consensus

La fonction `compare_k_selection_methods()` exécute les trois méthodes et détermine un consensus par vote majoritaire :

```
results <- compare_k_selection_methods(
  KMeansClusterer,
  data,
  k_range = 2:8
)
# Affiche 3 graphiques et retourne:
# - results$consensus_k : K recommandé (majorité 2/3)
# - results$elbow$suggested_k
# - results$silhouette$suggested_k
# - results$homogeneity$suggested_k
```

Si les trois méthodes divergent, la méthode du coude est utilisée par défaut.

## 3.3 Visualisation des résultats (module 06)

Le module `06_visualization.R` fournit des fonctions génériques compatibles avec tous les clusterers du package.

### 3.3.1 Cercle des corrélations

**Fonction** `plot_clustering_2d()`

Projette les variables sur les deux premières composantes principales d'une ACP globale :

```
plot_clustering_2d(clusterer, main = NULL,
                   show_centers = TRUE, show_labels = TRUE)
```

- Les variables sont représentées par des **flèches** partant de l'origine
- La **longueur** indique la qualité de représentation dans le plan
- La **couleur** indique l'appartenance au cluster
- Le **cercle unité** aide à identifier les variables bien représentées
- Le ratio d'aspect `asp=1` préserve les angles (corrélations)

**Fonction** `plot_clustering_with_supp()`

Variante distinguant les variables actives (trait plein) des variables supplémentaires (trait pointillé) après appel à `predict()`.

### 3.3.2 Heatmap des corrélations

```
plot_correlation_heatmap(clusterer, main = "...", reorder = TRUE)
```

Affiche la matrice de corrélation avec :

- Réordonnancement des variables par cluster (si `reorder = TRUE`)
- Séparateurs noirs entre clusters
- Échelle de couleurs : bleu (corrélation négative) → blanc → rouge (positive)

### 3.3.3 Graphe de réseau

```
plot_network_graph(clusterer, threshold = 0.3,
                   layout = "fruchterman.reingold")
```

Représentation en graphe où :

- Les **nœuds** sont les variables, colorées par cluster
- Les **arêtes** relient les variables dont l'association dépasse le seuil
- Trois algorithmes de disposition : Fruchterman-Reingold, cercle, Kamada-Kawai

Nécessite le package optionnel `igraph`. Sans lui, un graphique alternatif est affiché.

### 3.3.4 Métriques de qualité

**Fonction** `plot_cluster_quality()`

Affiche deux graphiques :

1. **Taille des clusters** : nombre de variables par cluster
2. **Homogénéité par cluster** : variance expliquée par PC1

**Fonction** `plot_variable_contributions()`

Visualise les contributions des variables à chaque cluster :

```
plot_variable_contributions(clusterer, cluster_id = NULL, top_n = 10)
```

Affiche les `top_n` variables les plus contributives (loadings PC1 en valeur absolue) pour chaque cluster ou un cluster spécifique.

**Fonction** `plot_scree_by_cluster()`

Crée un scree plot par cluster montrant :

- Barres : pourcentage de variance par composante principale
- Ligne rouge : variance cumulée

Utile pour évaluer si un cluster est bien résumé par sa première composante ou s'il est intrinsèquement multidimensionnel.

### 3.3.5 Dendrogramme

```
plot_dendrogram(clusterer, main = "Dendrogram", ...)
```

Détecte automatiquement le type de clusterer :

- **DivisiveClusterer** : appelle `plot_split_dendrogram()` (arbre descendant)
- **MCA_HClusterer** : utilise le dendrogramme standard de la CAH

### 3.3.6 Synthèse des fonctions de visualisation

| Fonction | Usage | Dépendance |
|---|---|---|
| `plot_clustering_2d` | Cercle des corrélations | Base R |
| `plot_clustering_with_supp` | Avec variables supplémentaires | Base R |
| `plot_correlation_heatmap` | Matrice de corrélation | Base R |
| `plot_network_graph` | Graphe de réseau | igraph (optionnel) |
| `plot_cluster_quality` | Tailles et homogénéités | Base R |
| `plot_variable_contributions` | Contributions à PC1 | Base R |
| `plot_scree_by_cluster` | Variance par composante | Base R |
| `plot_dendrogram` | Arbre hiérarchique | Base R |

Table 3.2: Récapitulatif des fonctions de visualisation

# 4.Application R Shiny

Ce chapitre présente l'application Shiny développée pour exploiter le package M2RClust de manière interactive. L'interface permet aux utilisateurs de charger des données, de configurer et d'exécuter des analyses de clustering, puis de visualiser les résultats sans nécessiter de compétences en programmation R.

## 4.1 Architecture et fonctionnalités globales

L'application suit une architecture modulaire avec séparation des composants UI et serveur pour chaque page (Accueil, Upload, Clustering). Cette organisation facilite la maintenance et l'extensibilité du code.

```
shinyR/
|-- main.R                    # Point d'entrée de l'application
|-- ui.R                      # Définition de l'interface utilisateur
|-- server.R                  # Logique serveur principale
|-- ui/                       # Modules UI par page
|   |-- home.R
|   |-- upload.R
|   |-- cluster.R
|-- server/                   # Modules serveur par page
|   |-- home_server.R
|   |-- upload_server.R
|   |-- cluster_server.R
|-- texts/                    # Ressources textuelles
    |-- dictionnary.csv       # Traductions FR/EN
    |-- markdowns/
        |-- home_en.md
        |-- home_fr.md
```

### 4.1.1 Fonctionnalités transversales

Deux fonctionnalités sont accessibles depuis l'en-tête de l'application :

- **Changement de thème** : Basculement entre thème clair (Flatly) et sombre (Darkly) via `bslib`

- **Internationalisation** : Support bilingue français/anglais géré par un système de traduction réactif basé sur un fichier CSV

### 4.1.2 Programmation réactive

L'application utilise intensivement le paradigme réactif de Shiny pour assurer une interface fluide et performante :

- **Mise en cache** : Les données chargées sont conservées en mémoire et rechargées uniquement si le fichier source est modifié
- **Valeurs réactives** : Les états (thème, langue, données, résultats) sont gérés via `reactiveValues()`, permettant une synchronisation automatique de l'interface
- **Isolation des calculs** : Utilisation de `observeEvent` et `reactive()` pour limiter les recalculs aux modifications pertinentes

## 4.2 Workflow de l'application

### 4.2.1 Page d'accueil

Présente le contexte du projet, les objectifs et les instructions d'utilisation. Le contenu est chargé dynamiquement depuis des fichiers Markdown selon la langue sélectionnée.

### 4.2.2 Page de chargement des données

Cette page permet de :

1. **Importer un fichier** : Formats CSV et XLSX supportés (jusqu'à 500 Mo)
2. **Configurer les variables** :
   - Détection automatique des types (quantitative/qualitative)
   - Assignation des rôles : *Active* (utilisée pour le clustering), *Illustrative* (projetée a posteriori), *Excluded* (ignorée)
   - Édition des noms de colonnes si souhaité
3. **Prévisualiser les données** : Aperçu tabulaire des premières lignes

Les variables configurées sont stockées dans un objet réactif qui alimente ensuite les algorithmes de clustering.

### 4.2.3 Page de clustering

#### Configuration

L'utilisateur sélectionne un algorithme parmi trois options :

- **K-Means** : Réallocation dynamique avec choix de la méthode d'initialisation (random, k-means++, PCA)
- **Hierarchical** : Division hiérarchique (PDDP) avec option de rotation des axes
- **Mixed** : Approche hybride ACM + CAH pour variables qualitatives avec choix de la liaison

Les paramètres principaux incluent le nombre de clusters souhaités, le nombre maximal d'itérations et le seuil de convergence. Des paramètres avancés spécifiques à chaque algorithme sont accessibles via un panneau dépliable.

#### Exécution et résultats

Chaque exécution génère dynamiquement un nouvel onglet de résultats contenant :

- **Tableau d'assignation** : Liste des variables avec leur cluster et degré d'appartenance
- **Métriques de qualité** : Taille des clusters, homogénéité, variance expliquée
- **Visualisations interactives** : Cercle des corrélations, dendrogramme ou plan factoriel selon l'algorithme (via `plotly`)
- **Options d'export** : Téléchargement des résultats en CSV et des graphiques en PNG/HTML

L'utilisateur peut lancer plusieurs analyses successives et comparer les résultats via les différents onglets créés dynamiquement.

# Conclusion

Ce projet a permis de développer un package R complet dédié au clustering de variables. Les trois algorithmes (K-Means, hiérarchique divisif, et approche hybride ACM+CAH) créées permettent d'analyser efficacement des jeux de données selon des approches diverses et variées. L'architecture R6 garantit maintenabilité et extensibilité, si l'on veut rajouter des méthodes aux différentes classes. Les tables et visualisations riches que les algorithmes retournent simplifient l'utilisation et l'exploitation des classes dans un contexte plus opérationnel que développeur. Enfin, l'interface Shiny rajoute une dernière couche à l'utilisateur pour qu'il puisse réaliser ces clustering en No-Code.

# Bibliographie

[1] Marie Chavent, Vanessa Kuentz-Simonet, Amaury Labenne, and Jérôme Saracco. Multivariate analysis of mixed data: The r package pcamixdata. *arXiv preprint arXiv:1411.4911*, 2014.

[2] Marie Chavent, Vanessa Kuentz-Simonet, Benoit Liquet, and Jérôme Saracco. Clustofvar: An r package for the clustering of variables. *Journal of Statistical Software*, 50(13):1–16, 2012.

[3] SAS Institute Inc. *The VARCLUS Procedure*. SAS Institute Inc., Cary, NC, 2020. SAS/STAT 15.2 User's Guide.