Multivariate Analysis I

Alboukadel Kassambara

Practical Guide To Cluster Analysis in R

Unsupervised Machine Learning

sthda.com

Edition 1

Published by STHDA (http://www.sthda.com), Alboukadel Kassambara

Contact: Alboukadel Kassambara <alboukadel.kassambara@gmail.com>

For general information contact Alboukadel Kassambara <alboukadel.kassambara@gmail.com>.

0.1 Preface

Large amounts of data are collected every day from satellite images, bio-medical, security, marketing, web search, geo-spatial or other automatic equipment. Mining knowledge from these big data far exceeds human's abilities.

Clustering is one of the important data mining methods for discovering knowledge in multidimensional data. The goal of clustering is to identify pattern or groups of similar objects within a data set of interest.

In the litterature, it is referred as "pattern recognition" or "unsupervised machine learning" - "unsupervised" because we are not guided by a priori ideas of which variables or samples belong in which clusters. "Learning" because the machine algorithm "learns" how to cluster.

Cluster analysis is popular in many fields, including:

- In *cancer research* for classifying patients into subgroups according their gene expression profile. This can be useful for identifying the molecular profile of patients with good or bad prognostic, as well as for understanding the disease.
- In *marketing* for *market segmentation* by identifying subgroups of customers with similar profiles and who might be receptive to a particular form of advertising.
- In *City-planning* for identifying groups of houses according to their type, value and location.

This book provides a practical guide to unsupervised machine learning or cluster analysis using R software. Additionally, we developped an R package named *factoextra* to create, easily, a ggplot2-based elegant plots of cluster analysis results. Factoextra official online documentation: http://www.sthda.com/english/rpkgs/factoextra

0.2 About the author

Alboukadel Kassambara is a PhD in Bioinformatics and Cancer Biology. He works since many years on genomic data analysis and visualization. He created a bioinformatics tool named GenomicScape (www.genomicscape.com) which is an easy-to-use web tool for gene expression data analysis and visualization.

He developed also a website called STHDA (Statistical Tools for High-throughput Data Analysis, www.sthda.com/english), which contains many tutorials on data analysis and visualization using R software and packages.

He is the author of the R packages **survminer** (for analyzing and drawing survival curves), **ggcorrplot** (for drawing correlation matrix using ggplot2) and **factoextra** (to easily extract and visualize the results of multivariate analysis such PCA, CA, MCA and clustering). You can learn more about these packages at: http://www.sthda.com/english/wiki/r-packages

Recently, he published two books on data visualization:

1. Guide to Create Beautiful Graphics in R (at: https://goo.gl/vJ0OYb).
2. Complete Guide to 3D Plots in R (at: https://goo.gl/v5gwl0).

Contents

0.3 Key features of this book

Although there are several good books on unsupervised machine learning/clustering and related topics, we felt that many of them are either too high-level, theoretical or too advanced. Our goal was to write a practical guide to cluster analysis, elegant visualization and interpretation.

The main parts of the book include:

- *distance measures*,
- *partitioning clustering*,
- *hierarchical clustering*,
- *cluster validation methods*, as well as,
- *advanced clustering methods* such as fuzzy clustering, density-based clustering and model-based clustering.

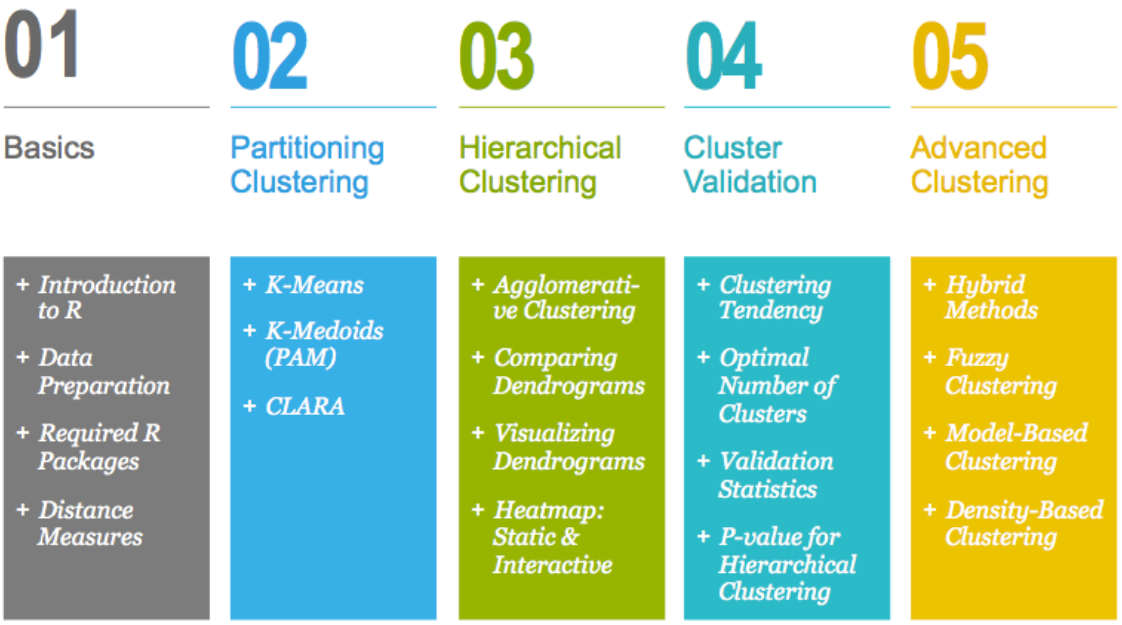
The book presents the basic principles of these tasks and provide many examples in R. This book offers solid guidance in data mining for students and researchers.

Key features:

- Covers clustering algorithm and implementation
- Key mathematical concepts are presented
- Short, self-contained chapters with practical examples. This means that, you don't need to read the different chapters in sequence.

> At the end of each chapter, we present R lab sections in which we systematically work through applications of the various methods discussed in that chapter.

0.4 How this book is organized?



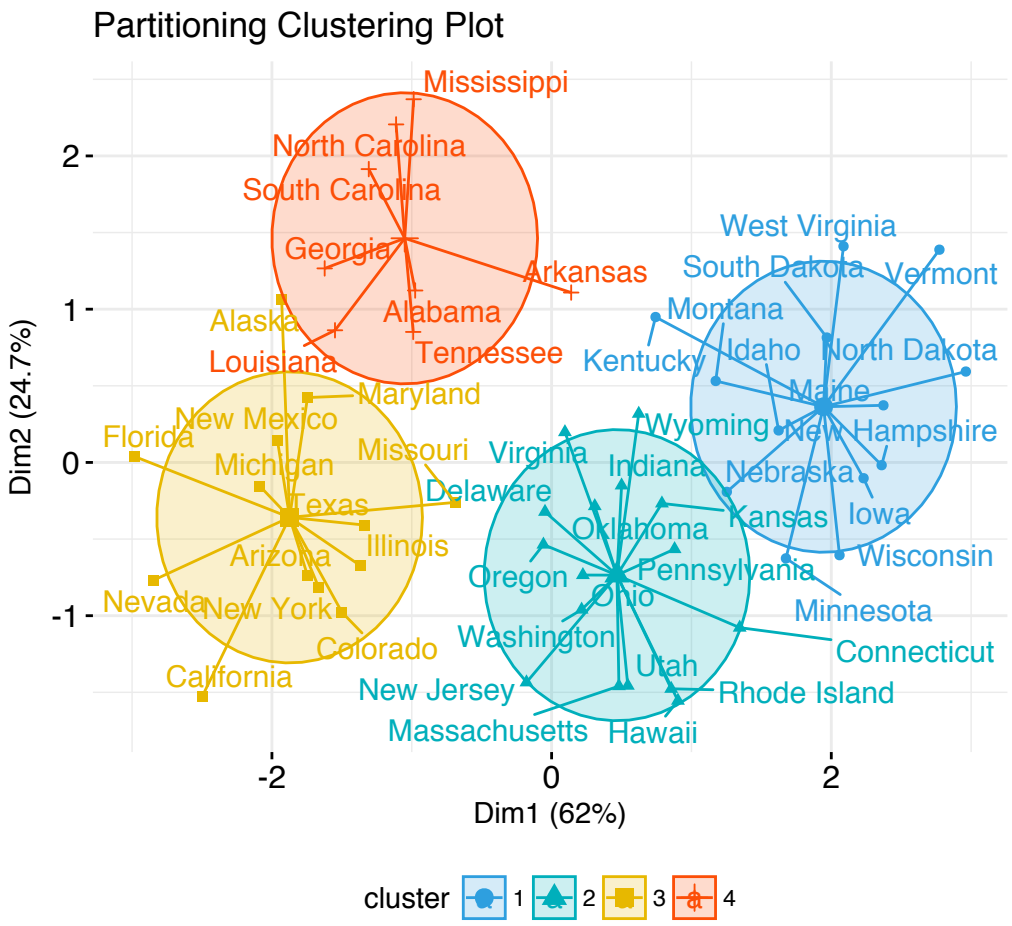
This book contains 5 parts. Part I (Chapter 1 - 3) provides a quick introduction to R (chapter 1) and presents required R packages and data format (Chapter 2) for clustering analysis and visualization.

The classification of objects, into clusters, requires some methods for measuring the distance or the (dis)similarity between the objects. Chapter 3 covers the common distance measures used for assessing similarity between observations.

Part II starts with partitioning clustering methods, which include:

- K-means clustering (Chapter 4),
- K-Medoids or PAM (partitioning around medoids) algorithm (Chapter 5) and
- CLARA algorithms (Chapter 6).

Partitioning clustering approaches subdivide the data sets into a set of k groups, where k is the number of groups pre-specified by the analyst.



In Part III, we consider agglomerative hierarchical clustering method, which is an alternative approach to partitionning clustering for identifying groups in a data set. It does not require to pre-specify the number of clusters to be generated. The result of hierarchical clustering is a tree-based representation of the objects, which is also known as *dendrogram* (see the figure below).

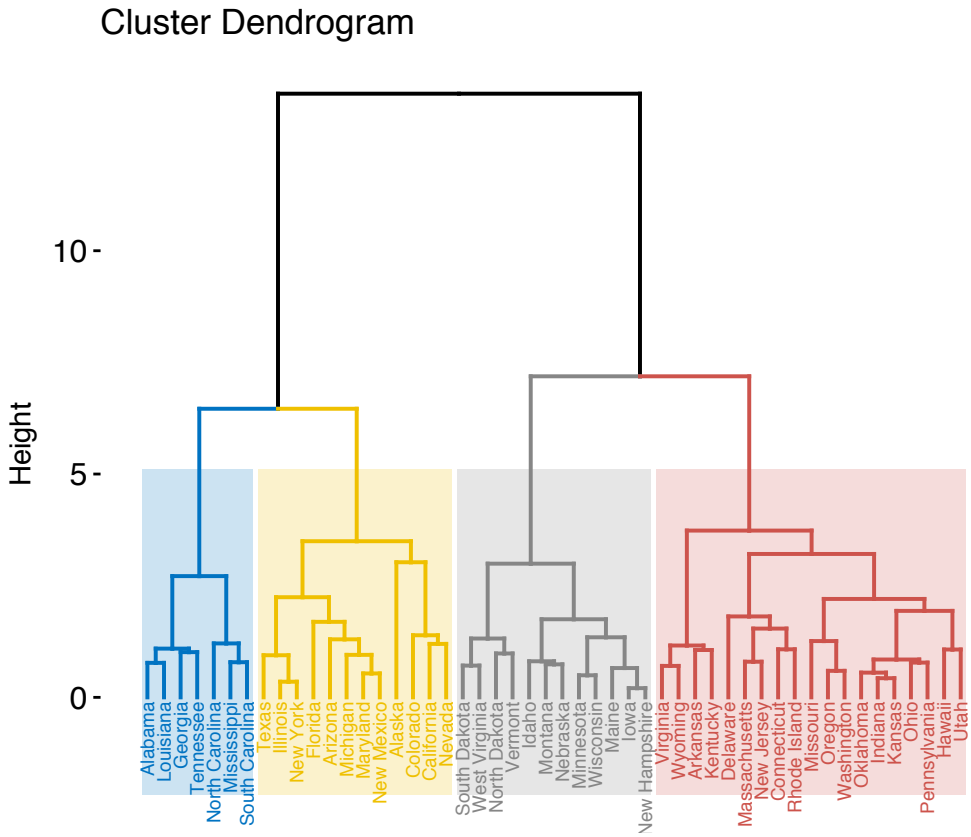
In this part, we describe how to compute, visualize, interpret and compare dendrograms:

- Agglomerative clustering (Chapter 7)
  - Algorithm and steps
  - Verify the cluster tree
  - Cut the dendrogram into different groups
- Compare dendrograms (Chapter 8)
  - Visual comparison of two dendrograms
  - Correlation matrix between a list of dendrograms

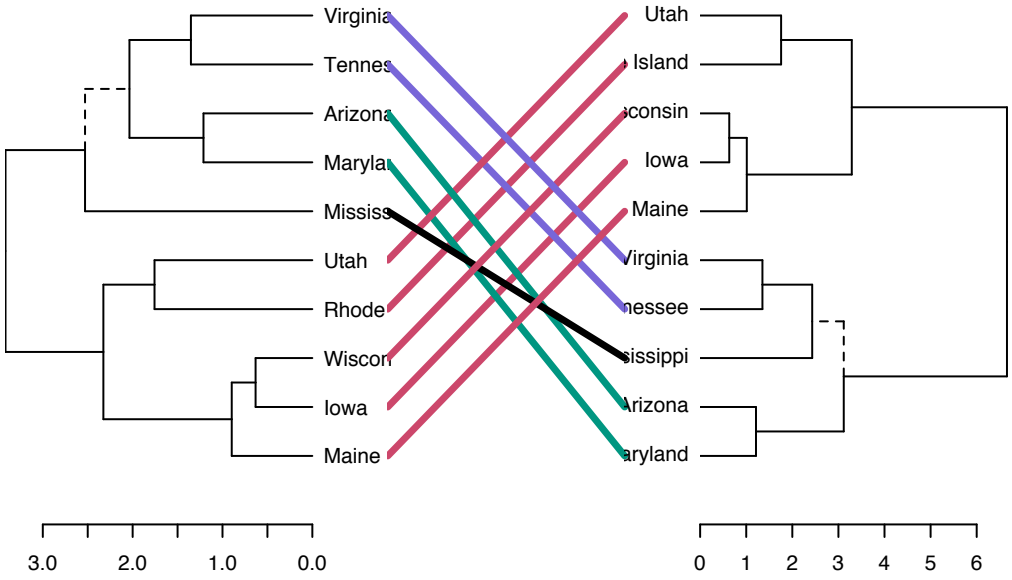
- Visualize dendrograms (Chapter 9)
  - Case of small data sets
  - Case of dendrogram with large data sets: zoom, sub-tree, PDF
  - Customize dendrograms using dendextend
- Heatmap: static and interactive (Chapter 10)
  - R base heat maps
  - Pretty heat maps
  - Interactive heat maps
  - Complex heatmap
  - Real application: gene expression data

In this section, you will learn how to generate and interpret the following plots.

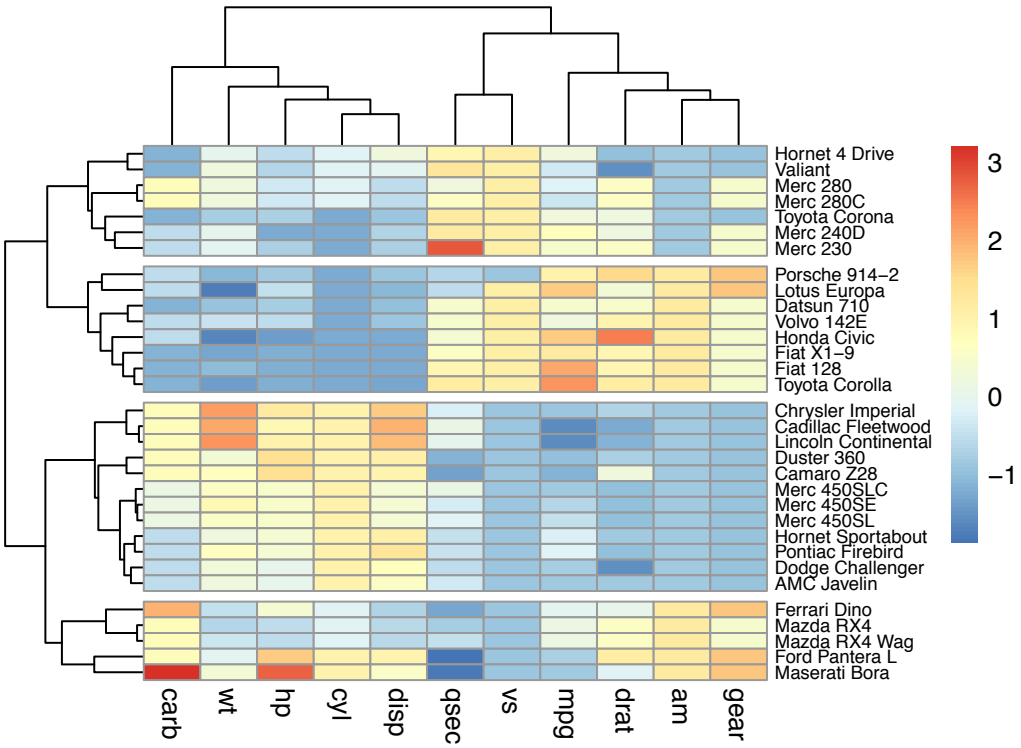
- **Standard dendrogram with filled rectangle around clusters**:



- **Compare two dendrograms**:



- **Heatmap**:



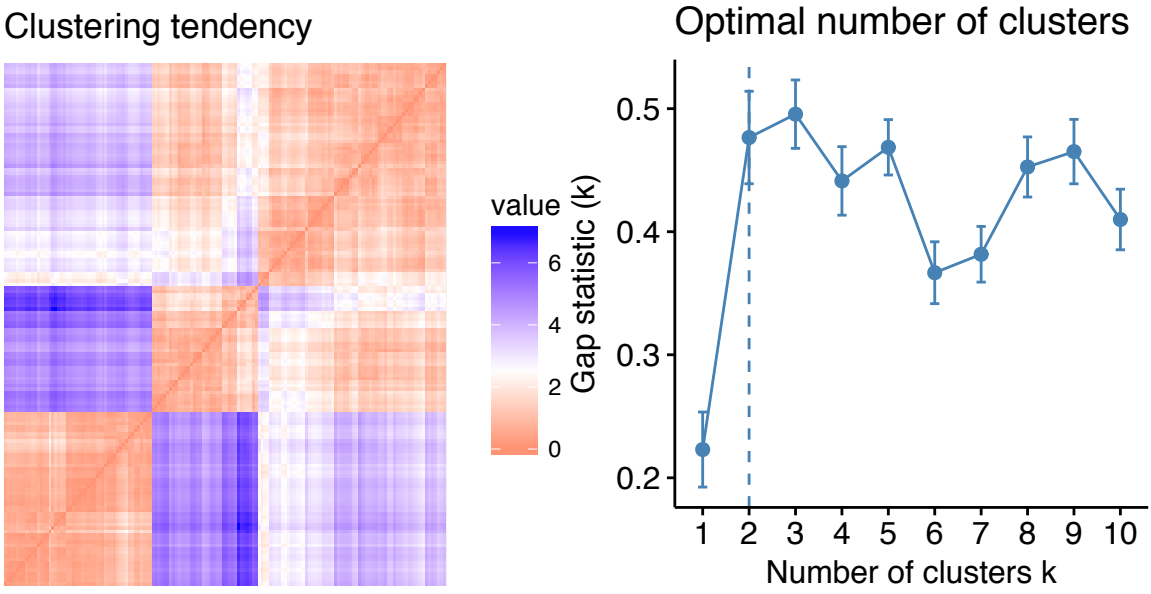
Part IV describes clustering validation and evaluation strategies, which consists of measuring the goodness of clustering results. Before applying any clustering algorithm to a data set, the first thing to do is to assess the *clustering tendency*. That is, whether applying clustering is suitable for the data. If yes, then how many clusters are there. Next, you can perform hierarchical clustering or partitioning clustering (with a pre-specified number of clusters). Finally, you can use a number of measures, described in this chapter, to evaluate the goodness of the clustering results.

The different chapters included in part IV are organized as follow:

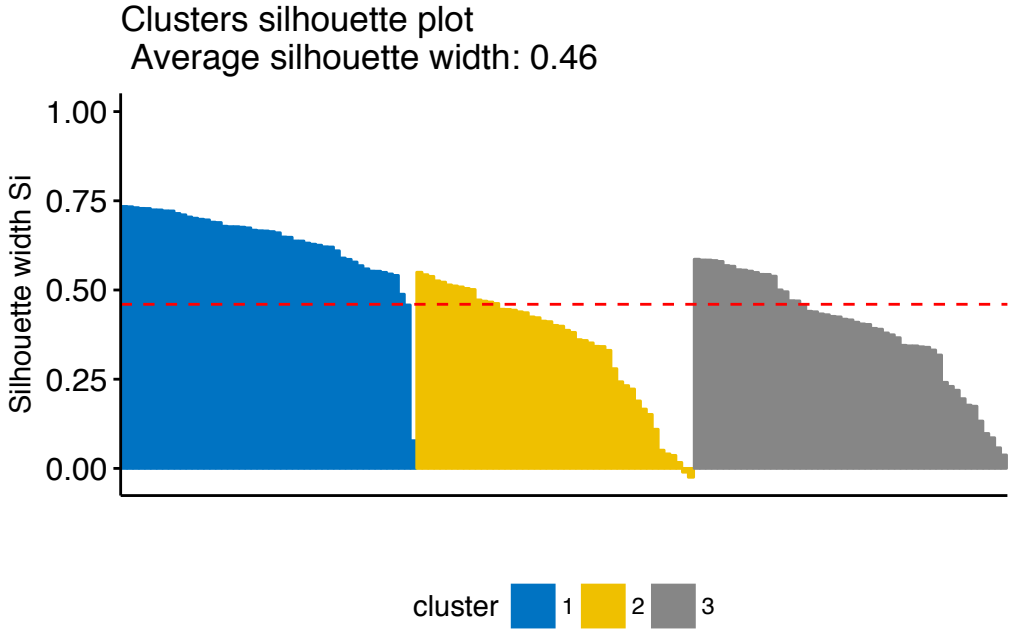
- Assessing clustering tendency (Chapter 11)
- Determining the optimal number of clusters (Chapter 12)
- Cluster validation statistics (Chapter 13)
- Choosing the best clustering algorithms (Chapter 14)
- Computing p-value for hierarchical clustering (Chapter 15)

In this section, you'll learn how to create and interpret the plots hereafter.

- **Visual assessment of clustering tendency** (left panel): Clustering tendency is detected in a visual form by counting the number of square shaped dark blocks along the diagonal in the image.
- **Determine the optimal number of clusters** (right panel) in a data set using the gap statistics.



- Cluster validation using the *silhouette coefficient* (Si): A value of Si close to 1 indicates that the object is well clustered. A value of Si close to -1 indicates that the object is poorly clustered. The figure below shows the silhouette plot of a k-means clustering.



Part V presents advanced clustering methods, including:

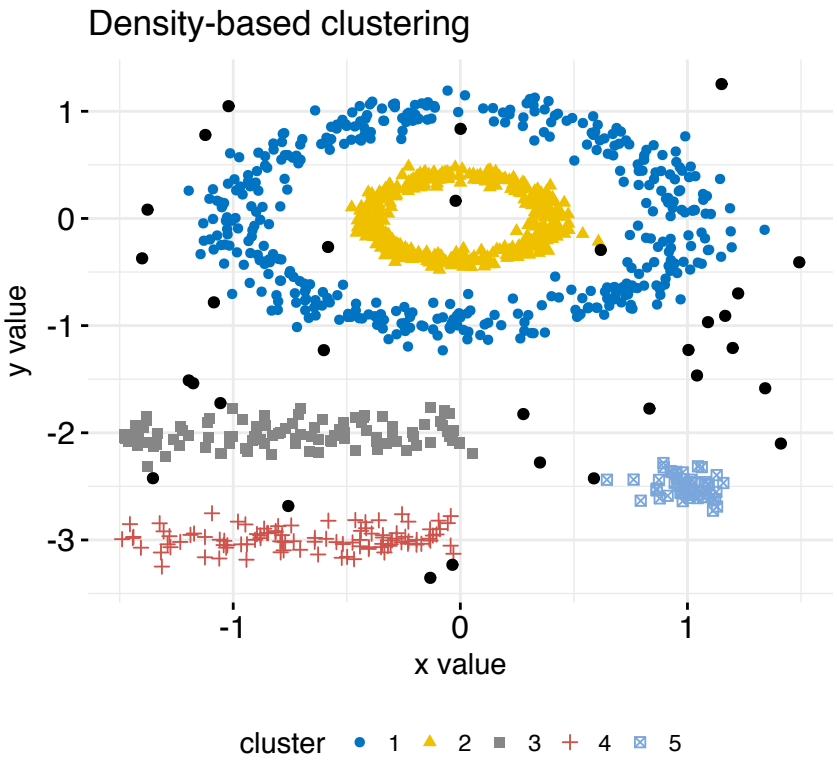
- Hierarchical k-means clustering (Chapter 16)
- Fuzzy clustering (Chapter 17)
- Model-based clustering (Chapter 18)
- DBSCAN: Density-Based Clustering (Chapter 19)

The *hierarchical k-means clustering* is an hybrid approach for improving k-means results.

In *Fuzzy clustering*, items can be a member of more than one cluster. Each item has a set of membership coefficients corresponding to the degree of being in a given cluster.

In *model-based clustering*, the data are viewed as coming from a distribution that is mixture of two ore more clusters. It finds best fit of models to data and estimates the number of clusters.

The *density-based clustering* (DBSCAN is a partitioning method that has been introduced in Ester et al. (1996). It can find out clusters of different shapes and sizes from data containing noise and outliers.



0.5 Book website

The website for this book is located at : http://www.sthda.com/english/. It contains number of ressources.

0.6 Executing the R codes from the PDF

For a single line R code, you can just copy the code from the PDF to the R console.

For a multiple-line R codes, an error is generated, sometimes, when you copy and paste directly the R code from the PDF to the R console. If this happens, a solution is to:

- Paste firstly the code in your R code editor or in your text editor
- Copy the code from your text/code editor to the R console

Part I

Basics

Chapter 1

Introduction to R

R is a free and powerful statistical software for **analyzing** and **visualizing** data. If you want to learn easily the essential of R programming, visit our series of tutorials available on STHDA: http://www.sthda.com/english/wiki/r-basics-quick-and-easy.

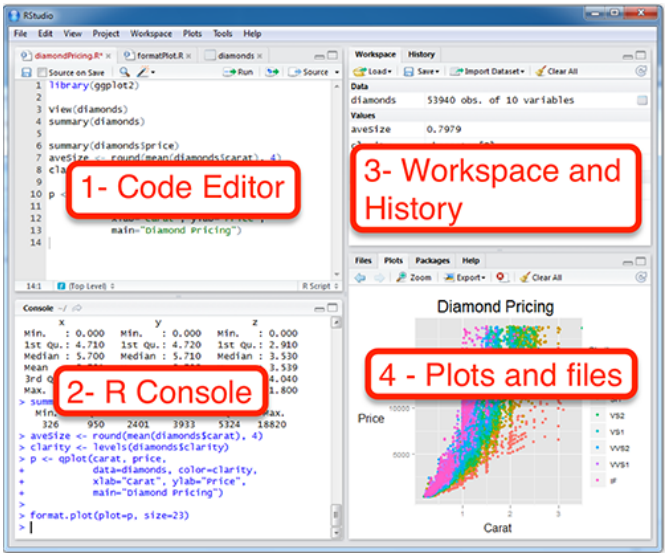
In this chapter, we provide a very brief introduction to **R**, for installing R/RStudio as well as importing your data into R.

1.1 Install R and RStudio

R and RStudio can be installed on Windows, MAC OSX and Linux platforms. RStudio is an integrated development environment for R that makes using R easier. It includes a console, code editor and tools for plotting.

1. R can be downloaded and installed from the Comprehensive R Archive Network (CRAN) webpage (http://cran.r-project.org/).
2. After installing R software, install also the RStudio software available at: http://www.rstudio.com/products/RStudio/.
3. Launch RStudio and start use R inside R studio.

RStudio screen:



1.2 Installing and loading R packages

An **R package** is an extension of R containing data sets and specific R functions to solve specific questions.

For example, in this book, you'll learn how to compute easily clustering algorithm using the *cluster* R package.

There are thousands other R packages available for download and installation from CRAN, Bioconductor(biology related R packages) and GitHub repositories.

1. How to install packages from CRAN? Use the function *install.packages*():

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

2. How to install packages from GitHub? You should first install *devtools* if you don't have it already installed on your computer:

For example, the following R code installs the latest version of *factoextra* R package developed by A. Kassambara (https://github.com/kassambara/facoextra) for multivariate data analysis and elegant visualization..

```
install.packages("devtools")
devtools::install_github("kassambara/factoextra")
```

Note that, GitHub contains the developmental version of R packages.

3. After installation, you must first load the package for using the functions in the package. The function *library*() is used for this task.

```
library("cluster")
```

Now, we can use R functions in the cluster package for computing clustering algorithms, such as PAM (Partitioning Around Medoids).

1.3 Getting help with functions in R

If you want to learn more about a given function, say kmeans(), type this:

```
?kmeans
```

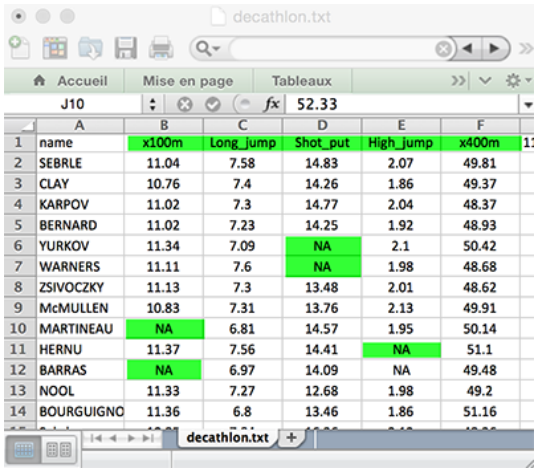
1.4 Importing your data into R

1. **Prepare your file** as follow:

- Use the first row as **column names**. Generally, columns represent **variables**
- Use the first column as **row names**. Generally rows represent **observations**.
- Each row/column name should be unique, so remove duplicated names.
- Avoid names with blank spaces. Good column names: *Long_jump* or *Long.jump*. Bad column name: *Long jump*.
- Avoid names with special symbols: ?, $, *, +, #, (,), -, /, }, {, |, >, < etc. Only underscore can be used.
- Avoid beginning variable names with a number. Use letter instead. Good column names: sport_100m or x100m. Bad column name: 100m
- R is case sensitive. This means that Name is different from Name or NAME.
- Avoid blank rows in your data
- Delete any comments in your file

- Replace missing values by **NA** (for not available)
- If you have a column containing date, use the four digit format. Good format: 01/01/2016. Bad format: 01/01/16

2. Our **final file** should look like this:



	A	B	C	D	E	F
1	name	x100m	Long_jump	Shot_put	High_jump	x400m
2	SEBRLE	11.04	7.58	14.83	2.07	49.81
3	CLAY	10.76	7.4	14.26	1.86	49.37
4	KARPOV	11.02	7.3	14.77	2.04	48.37
5	BERNARD	11.02	7.23	14.25	1.92	48.93
6	YURKOV	11.34	7.09	NA	2.1	50.42
7	WARNERS	11.11	7.6	NA	1.98	48.68
8	ZSIVOCZKY	11.13	7.3	13.48	2.01	48.62
9	McMULLEN	10.83	7.31	13.76	2.13	49.91
10	MARTINEAU	NA	6.81	14.57	1.95	50.14
11	HERNU	11.37	7.56	14.41	NA	51.1
12	BARRAS	NA	6.97	14.09	NA	49.48
13	NOOL	11.33	7.27	12.68	1.98	49.2
14	BOURGUIGNO	11.36	6.8	13.46	1.86	51.16

3. **Save your file**

We recommend to save your file into **.txt** (tab-delimited text file) or **.csv** (comma separated value file) format.

4. **Get your data into R**:

Use the R code below. You will be asked to choose a file:

```
# .txt file: Read tab separated values
my_data <- read.delim(file.choose())

# .csv file: Read comma (",") separated values
my_data <- read.csv(file.choose())

# .csv file: Read semicolon (";") separated values
my_data <- read.csv2(file.choose())
```

You can read more about how to import data into R at this link: http://www.sthda.com/english/wiki/importing-data-into-r

1.5 Demo data sets

R comes with several *built-in data sets*, which are generally used as demo data for playing with R functions. The most used R demo data sets include: **USArrests**, **iris** and **mtcars**. To load a demo data set, use the function **data**() as follow:

```
data("USArrests")   # Loading
head(USArrests, 3)  # Print the first 3 rows
```

```
##          Murder Assault UrbanPop Rape
## Alabama    13.2     236       58 21.2
## Alaska     10.0     263       48 44.5
## Arizona     8.1     294       80 31.0
```

If you want learn more about USArrests data sets, type this:

```
?USArrests
```

USArrests data set is an object of class **data frame**.

To select just certain columns from a data frame, you can either refer to the columns by name or by their location (i.e., column 1, 2, 3, etc.).

```
# Access the data in 'Murder' column
# dollar sign is used
head(USArrests$Murder)
```

```
## [1] 13.2 10.0  8.1  8.8  9.0  7.9
```

```
# Or use this
USArrests[, 'Murder']
```

1.6 Close your R/RStudio session

Each time you close R/RStudio, you will be asked whether you want to save the data from your R session. If you decide to save, the data will be available in future R sessions.

Chapter 2

Data Preparation and R Packages

2.1 Data preparation

To perform a cluster analysis in R, generally, the data should be prepared as follow:

1. Rows are observations (individuals) and columns are variables
2. Any missing value in the data must be removed or estimated.
3. The data must be standardized (i.e., scaled) to make variables comparable. Recall that, standardization consists of transforming the variables such that they have mean zero and standard deviation one. Read more about data standardization in chapter 3.

Here, we'll use the built-in R data set "USArrests", which contains statistics in arrests per 100,000 residents for assault, murder, and rape in each of the 50 US states in 1973. It includes also the percent of the population living in urban areas.

```
data("USArrests")  # Load the data set
df <- USArrests    # Use df as shorter name
```

1. To remove any missing value that might be present in the data, type this:

```
df <- na.omit(df)
```

2. As we don't want the clustering algorithm to depend to an arbitrary variable unit, we start by scaling/standardizing the data using the R function *scale*():

```
df <- scale(df)
head(df, n = 3)
```

```
##              Murder   Assault   UrbanPop         Rape
## Alabama 1.24256408 0.7828393 -0.5209066 -0.003416473
## Alaska  0.50786248 1.1068225 -1.2117642  2.484202941
## Arizona 0.07163341 1.4788032  0.9989801  1.042878388
```

2.2 Required R Packages

In this book, we'll use mainly the following R packages:

- **cluster** for computing clustering algorithms, and
- **factoextra** for ggplot2-based elegant visualization of clustering results. The official online documentation is available at: http://www.sthda.com/english/rpkgs/factoextra.

factoextra contains many functions for cluster analysis and visualization, including:

Functions	Description
dist(fviz_dist, get_dist)	Distance Matrix Computation and Visualization
get_clust_tendency	Assessing Clustering Tendency
fviz_nbclust(fviz_gap_stat)	Determining the Optimal Number of Clusters
fviz_dend	Enhanced Visualization of Dendrogram
fviz_cluster	Visualize Clustering Results
fviz_mclust	Visualize Model-based Clustering Results
fviz_silhouette	Visualize Silhouette Information from Clustering
hcut	Computes Hierarchical Clustering and Cut the Tree
hkmeans	Hierarchical k-means clustering
eclust	Visual enhancement of clustering analysis

To install the two packages, type this:

```
install.packages(c("cluster", "factoextra"))
```

Chapter 3

Clustering Distance Measures

The classification of observations into groups requires some methods for computing the **distance** or the (dis)**similarity** between each pair of observations. The result of this computation is known as a dissimilarity or **distance matrix**.

There are many methods to calculate this distance information. In this article, we describe the common distance measures and provide R codes for computing and visualizing distances.

3.1 Methods for measuring distances

The choice of distance measures is a critical step in clustering. It defines how the similarity of two elements (x, y) is calculated and it will influence the shape of the clusters.

The classical methods for distance measures are *Euclidean* and *Manhattan distances*, which are defined as follow:

1. *Euclidean distance*:

$$d_{euc}(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

2. *Manhattan distance*:

$$d_{man}(x, y) = \sum_{i=1}^{n} |(x_i - y_i)|$$

Where, x and y are two vectors of length n.

Other dissimilarity measures exist such as **correlation-based distances**, which is widely used for gene expression data analyses. Correlation-based distance is defined by subtracting the correlation coefficient from 1. Different types of correlation methods can be used such as:

1. **Pearson correlation distance**:

$$d_{cor}(x, y) = 1 - \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

Pearson correlation measures the degree of a linear relationship between two profiles.

2. **Eisen cosine correlation distance** (Eisen et al., 1998):

It's a special case of Pearson's correlation with $\bar{x}$ and $\bar{y}$ both replaced by zero:

$$d_{eisen}(x, y) = 1 - \frac{\left|\sum_{i=1}^{n} x_i y_i\right|}{\sqrt{\sum_{i=1}^{n} x_i^2 \sum_{i=1}^{n} y_i^2}}$$

3. **Spearman correlation distance**:

The spearman correlation method computes the correlation between the rank of x and the rank of y variables.

$$d_{spear}(x, y) = 1 - \frac{\sum_{i=1}^{n}(x'_i - \bar{x'})(y'_i - \bar{y'})}{\sqrt{\sum_{i=1}^{n}(x'_i - \bar{x'})^2 \sum_{i=1}^{n}(y'_i - \bar{y'})^2}}$$

Where $x'_i = rank(x_i)$ and $y'_i = rank(y)$.

4. **Kendall correlation distance**:

Kendall correlation method measures the correspondence between the ranking of x and y variables. The total number of possible pairings of x with y observations is $n(n-1)/2$, where n is the size of x and y. Begin by ordering the pairs by the x values. If x and y are correlated, then they would have the same relative rank orders. Now, for each y_i, count the number of $y_j > y_i$ (concordant pairs (c)) and the number of $y_j < y_i$ (discordant pairs (d)).

Kendall correlation distance is defined as follow:

$$d_{kend}(x, y) = 1 - \frac{n_c - n_d}{\frac{1}{2}n(n-1)}$$

Where,

- n_c: total number of concordant pairs
- n_d: total number of discordant pairs
- n: size of x and y

Note that,

- Pearson correlation analysis is the most commonly used method. It is also known as a parametric correlation which depends on the distribution of the data.
- Kendall and Spearman correlations are non-parametric and they are used to perform rank-based correlation analysis.

In the formula above, x and y are two vectors of length n and, means $\bar{x}$ and $\bar{y}$, respectively. The distance between x and y is denoted $d(x, y)$.

3.2 What type of distance measures should we choose?

The choice of distance measures is very important, as it has a strong influence on the clustering results. For most common clustering software, the default distance measure is the Euclidean distance.

Depending on the type of the data and the researcher questions, other dissimilarity measures might be preferred. For example, correlation-based distance is often used in gene expression data analysis.

Correlation-based distance considers two objects to be similar if their features are highly correlated, even though the observed values may be far apart in terms of Euclidean distance. The distance between two objects is 0 when they are perfectly correlated. Pearson's correlation is quite sensitive to outliers. This does not matter when clustering samples, because the correlation is over thousands of genes. When clustering genes, it is important to be aware of the possible impact of outliers. This can be mitigated by using Spearman's correlation instead of Pearson's correlation.

If we want to identify clusters of observations with the same overall profiles regardless of their magnitudes, then we should go with *correlation-based distance* as a dissimilarity measure. This is particularly the case in gene expression data analysis, where we might want to consider genes similar when they are "up" and "down" together. It is also the case, in marketing if we want to identify group of shoppers with the same preference in term of items, regardless of the volume of items they bought.

If Euclidean distance is chosen, then observations with high values of features will be clustered together. The same holds true for observations with low values of features.

3.3 Data standardization

The value of distance measures is intimately related to the scale on which measurements are made. Therefore, variables are often scaled (i.e. standardized) before measuring the inter-observation dissimilarities. This is particularly recommended when variables are measured in different scales (e.g: kilograms, kilometers, centimeters, ...); otherwise, the dissimilarity measures obtained will be severely affected.

The goal is to make the variables comparable. Generally variables are scaled to have i) standard deviation one and ii) mean zero.

The standardization of data is an approach widely used in the context of gene expression data analysis before clustering. We might also want to scale the data when the mean and/or the standard deviation of variables are largely different.

When scaling variables, the data can be transformed as follow:

$$\frac{x_i - center(x)}{scale(x)}$$

Where $center(x)$ can be the mean or the median of x values, and $scale(x)$ can be the standard deviation (SD), the interquartile range, or the MAD (median absolute deviation).

The R base function *scale*() can be used to standardize the data. It takes a numeric matrix as an input and performs the scaling on the columns.

Standardization makes the four distance measure methods - Euclidean, Manhattan, Correlation and Eisen - more similar than they would be with non-transformed data.

Note that, when the data are standardized, there is a functional relationship between the Pearson correlation coefficient $r(x, y)$ and the Euclidean distance.

With some maths, the relationship can be defined as follow:

$$d_{euc}(x, y) = \sqrt{2m[1 - r(x, y)]}$$

Where x and y are two standardized m-vectors with zero mean and unit length.

Therefore, the result obtained with Pearson correlation measures and standardized Euclidean distances are comparable.

3.4 Distance matrix computation

3.4.1 Data preparation

We'll use the USArrests data as demo data sets. We'll use only a subset of the data by taking 15 random rows among the 50 rows in the data set. This is done by using the function *sample*(). Next, we standardize the data using the function *scale*():

```
# Subset of the data
set.seed(123)
ss <- sample(1:50, 15)    # Take 15 random rows
df <- USArrests[ss, ]     # Subset the 15 rows
df.scaled <- scale(df)    # Standardize the variables
```

3.4.2 R functions and packages

There are many R functions for computing distances between pairs of observations:

1. *dist*() R base function [*stats* package]: Accepts only numeric data as an input.
2. *get_dist*() function [*factoextra* package]: Accepts only numeric data as an input. Compared to the standard dist() function, it supports correlation-based distance measures including "pearson", "kendall" and "spearman" methods.
3. *daisy()* function [*cluster* package]: Able to handle other variable types (e.g. nominal, ordinal, (a)symmetric binary). In that case, the Gower's coefficient will be automatically used as the metric. It's one of the most popular measures of proximity for mixed data types. For more details, read the R documentation of the *daisy*() function (*?daisy*).

All these functions compute distances between rows of the data.

3.4.3 Computing euclidean distance

To compute Euclidean distance, you can use the R base *dist*() function, as follow:

```
dist.eucl <- dist(df.scaled, method = "euclidean")
```

Note that, allowed values for the option method include one of: "euclidean", "maximum", "manhattan", "canberra", "binary", "minkowski".

To make it easier to see the distance information generated by the *dist*() function, you can reformat the distance vector into a matrix using the *as.matrix*() function.

```
# Reformat as a matrix
# Subset the first 3 columns and rows and Round the values
round(as.matrix(dist.eucl)[1:3, 1:3], 1)
```

```
##              Iowa Rhode Island Maryland
## Iowa          0.0          2.8      4.1
## Rhode Island  2.8          0.0      3.6
## Maryland      4.1          3.6      0.0
```

In this matrix, the value represent the distance between objects. The values on the diagonal of the matrix represent the distance between objects and themselves (which are zero).

In this data set, the columns are variables. Hence, if we want to compute pairwise distances between variables, we must start by transposing the data to have variables in the rows of the data set before using the *dist*() function. The function *t*() is used for transposing the data.

3.4.4 Computing correlation based distances

Correlation-based distances are commonly used in gene expression data analysis.

The function *get_dist*()[*factoextra* package] can be used to compute correlation-based distances. Correlation method can be either *pearson*, *spearman* or *kendall*.

```
# Compute
library("factoextra")
dist.cor <- get_dist(df.scaled, method = "pearson")

# Display a subset
round(as.matrix(dist.cor)[1:3, 1:3], 1)
```

```
##              Iowa Rhode Island Maryland
## Iowa          0.0          0.4      1.9
## Rhode Island  0.4          0.0      1.5
## Maryland      1.9          1.5      0.0
```

3.4.5 Computing distances for mixed data

The function *daisy*() [*cluster* package] provides a solution (*Gower's metric*) for computing the distance matrix, in the situation where the data contain no-numeric columns.

The R code below applies the *daisy*() function on *flower* data which contains *factor*, *ordered* and *numeric* variables:

```
library(cluster)
# Load data
data(flower)
head(flower, 3)
```

```
##   V1 V2 V3 V4 V5 V6  V7 V8
## 1  0  1  1  4  3 15  25 15
## 2  1  0  0  2  1  3 150 50
## 3  0  1  0  3  3  1 150 50
```

```
# Data structure
str(flower)
```

```
## 'data.frame':    18 obs. of  8 variables:
##  $ V1: Factor w/ 2 levels "0","1": 1 2 1 1 1 1 1 1 2 2 ...
##  $ V2: Factor w/ 2 levels "0","1": 2 1 2 1 2 2 1 1 2 2 ...
##  $ V3: Factor w/ 2 levels "0","1": 2 1 1 2 1 1 1 2 1 1 ...
##  $ V4: Factor w/ 5 levels "1","2","3","4",..: 4 2 3 4 5 4 4 2 3 5 ...
##  $ V5: Ord.factor w/ 3 levels "1"<"2"<"3": 3 1 3 2 2 3 3 2 1 2 ...
##  $ V6: Ord.factor w/ 18 levels "1"<"2"<"3"<"4"<..: 15 3 1 16 2 12 13 7 4 14 ...
##  $ V7: num  25 150 150 125 20 50 40 100 25 100 ...
##  $ V8: num  15 50 50 50 15 40 20 15 15 60 ...
```

```
# Distance matrix
dd <- daisy(flower)
round(as.matrix(dd)[1:3, 1:3], 2)
```

```
##      1    2    3
## 1 0.00 0.89 0.53
## 2 0.89 0.00 0.51
## 3 0.53 0.51 0.00
```

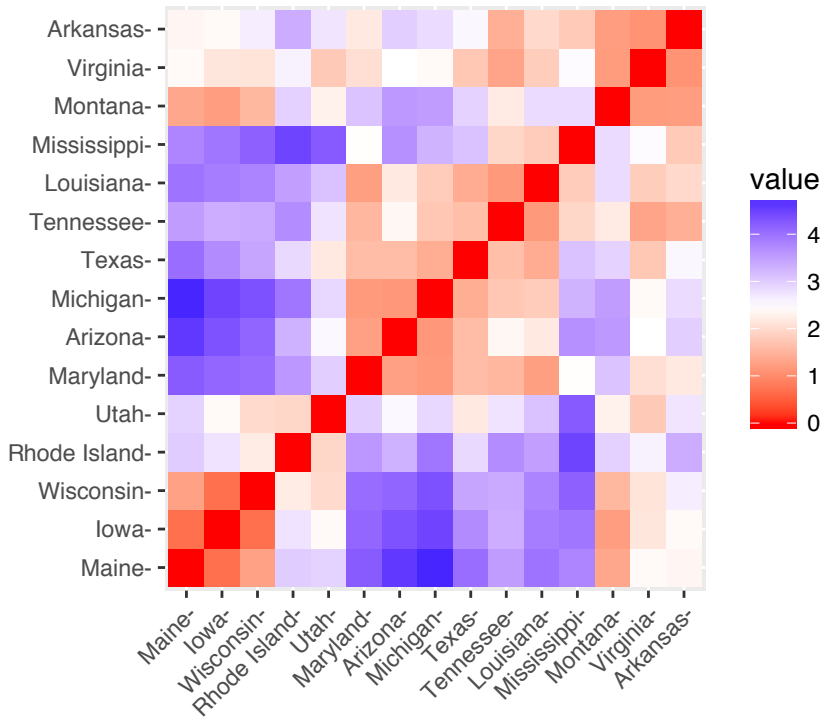
3.5 Visualizing distance matrices

A simple solution for visualizing the distance matrices is to use the function *fviz_dist*() [*factoextra* package]. Other specialized methods, such as agglomerative hierarchical clustering (Chapter 7) or heatmap (Chapter 10) will be comprehensively described in

the dedicated chapters.

To use *fviz_dist*() type this:

```
library(factoextra)
fviz_dist(dist.eucl)
```



- **Red**: high similarity (ie: low dissimilarity) | **Blue**: low similarity

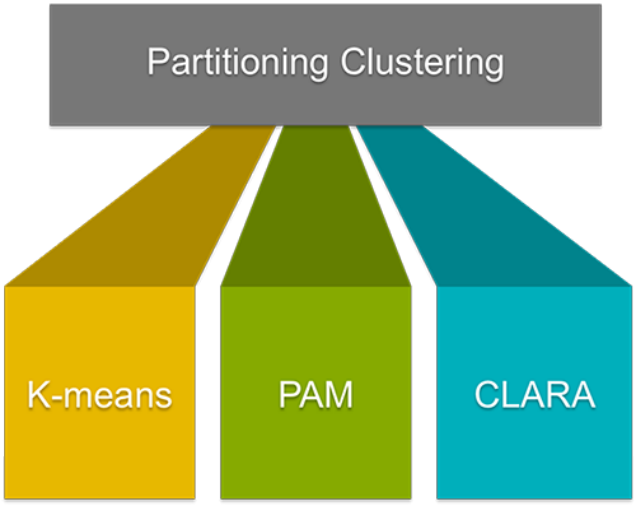
The color level is proportional to the value of the dissimilarity between observations: pure red if $dist(x_i, x_j) = 0$ and pure blue if $dist(x_i, x_j) = 1$. Objects belonging to the same cluster are displayed in consecutive order.

3.6 Summary

We described how to compute distance matrices using either Euclidean or correlation-based measures. It's generally recommended to standardize the variables before distance matrix computation. Standardization makes variable comparable, in the situation where they are measured in different scales.

Part II

Partitioning Clustering



Partitioning clustering are clustering methods used to classify observations, within a data set, into multiple groups based on their similarity. The algorithms require the analyst to specify the number of clusters to be generated.

This chapter describes the commonly used partitioning clustering, including:

- **K-means clustering** (MacQueen, 1967), in which, each cluster is represented by the center or means of the data points belonging to the cluster. The K-means method is sensitive to anomalous data points and outliers.
- **K-medoids clustering** or **PAM** (*Partitioning Around Medoids*, Kaufman & Rousseeuw, 1990), in which, each cluster is represented by one of the objects in the cluster. PAM is less sensitive to outliers compared to k-means.
- **CLARA algorithm** (*Clustering Large Applications*), which is an extension to PAM adapted for large data sets.

For each of these methods, we provide:

- the basic idea and the key mathematical concepts
- the clustering algorithm and implementation in R software
- R lab sections with many examples for cluster analysis and visualization

The following R packages will be used to compute and visualize partitioning clustering:

- *stats* package for computing K-means
- *cluster* package for computing PAM and CLARA algorithms
- *factoextra* for beautiful visualization of clusters

Chapter 4

K-Means Clustering

K-means clustering (MacQueen, 1967) is the most commonly used unsupervised machine learning algorithm for partitioning a given data set into a set of k groups (i.e. *k clusters*), where k represents the number of groups pre-specified by the analyst. It classifies objects in multiple groups (i.e., clusters), such that objects within the same cluster are as similar as possible (i.e., high *intra-class similarity*), whereas objects from different clusters are as dissimilar as possible (i.e., low *inter-class similarity*). In k-means clustering, each cluster is represented by its center (i.e, *centroid*) which corresponds to the mean of points assigned to the cluster.

In this article, we'll describe the **k-means algorithm** and provide practical examples using **R** software.

4.1 K-means basic ideas

The basic idea behind k-means clustering consists of defining clusters so that the total intra-cluster variation (known as total within-cluster variation) is minimized.

There are several k-means algorithms available. The standard algorithm is the Hartigan-Wong algorithm (1979), which defines the total within-cluster variation as the sum of squared distances Euclidean distances between items and the corresponding centroid:

$$W(C_k) = \sum_{x_i \in C_k} (x_i - \mu_k)^2$$

- x_i design a data point belonging to the cluster C_k
- μ_k is the mean value of the points assigned to the cluster C_k

Each observation (x_i) is assigned to a given cluster such that the sum of squares (SS) distance of the observation to their assigned cluster centers μ_k is a minimum.

We define the total within-cluster variation as follow:

$$tot.withinss = \sum_{k=1}^{k} W(C_k) = \sum_{k=1}^{k} \sum_{x_i \in C_k} (x_i - \mu_k)^2$$

The *total within-cluster sum of square* measures the compactness (i.e *goodness*) of the clustering and we want it to be as small as possible.

4.2 K-means algorithm

The first step when using k-means clustering is to indicate the number of clusters (k) that will be generated in the final solution.

The algorithm starts by randomly selecting k objects from the data set to serve as the initial centers for the clusters. The selected objects are also known as cluster means or centroids.

Next, each of the remaining objects is assigned to it's closest centroid, where closest is defined using the Euclidean distance (Chapter 3) between the object and the cluster mean. This step is called "cluster assignment step". Note that, to use correlation distance, the data are input as z-scores.

After the assignment step, the algorithm computes the new mean value of each cluster. The term cluster "centroid update" is used to design this step. Now that the centers have been recalculated, every observation is checked again to see if it might be closer to a different cluster. All the objects are reassigned again using the updated cluster means.

The cluster assignment and centroid update steps are iteratively repeated until the cluster assignments stop changing (i.e until *convergence* is achieved). That is, the

clusters formed in the current iteration are the same as those obtained in the previous iteration.

K-means algorithm can be summarized as follow:

1. Specify the number of clusters (K) to be created (by the analyst)
2. Select randomly k objects from the data set as the initial cluster centers or means
3. Assigns each observation to their closest centroid, based on the Euclidean distance between the object and the centroid
4. For each of the k clusters update the *cluster centroid* by calculating the new mean values of all the data points in the cluster. The centoid of a K_{th} cluster is a vector of length p containing the means of all variables for the observations in the k_{th} cluster; p is the number of variables.
5. Iteratively minimize the total within sum of square. That is, iterate steps 3 and 4 until the cluster assignments stop changing or the maximum number of iterations is reached. By default, the R software uses 10 as the default value for the maximum number of iterations.

4.3 Computing k-means clustering in R

4.3.1 Data

We'll use the demo data sets "USArrests". The data should be prepared as described in chapter 2. The data must contains only continuous variables, as the k-means algorithm uses variable means. As we don't want the k-means algorithm to depend to an arbitrary variable unit, we start by scaling the data using the R function *scale()* as follow:

```
data("USArrests")      # Loading the data set
df <- scale(USArrests) # Scaling the data

# View the firt 3 rows of the data
head(df, n = 3)
```