R et la fouille de données

M. Ledmi
m_ledmi@esi.dz

Département d'Informatique Khenchela

2020/2021

Plan

Vous êtes ici

Démarrer R

R est un logiciel pour l'analyse statistique. C'est un logiciel libre ; il est disponible gratuitement et tourne sur différent système (PC Linux, PC Windows, Mac).

Commandes

- Vous pouvez le télécharger ici : http ://cran.r-project.org/
- Le signe $>$ en début de ligne est l'invite de commande de R.
- Pour quitter R : $>$ **q()**

Astuce

Lorsque l'on utilise R, il est possible de reprendre la ligne de commande tapée précédemment en appuyant sur la flèche du haut du clavier. Appuyer plusieurs fois permet de récupérer des lignes plus anciennes.

Commentaires

Dans R, tout ce qui suit le caractère #(= dièse) est un commentaire et n'est pas pris en compte par R :

```
>#  Ceci est du baratin qui n'est pas pris en
compte par R !
```

Variables

R manipule que les tableaux de données. Ces tableaux sont stockés dans des *variables*, ce qui permet de leur donner un nom.

- Le nom des variables doit commencer par une lettre et peut contenir des lettres, des chiffres, des points et des caractères de soulignement (_), mais surtout pas d'espace.
- L'opérateur = est utilisé pour donner une valeur à une variable ; il peut se lire *prend la valeur de*
- on trouve aussi l'opérateur <- qui a exactement la même signification.

```
> age = 28
```

- Pour afficher la valeur de la variable âge, il suffit de taper le nom de la variable :

```
> age
[1] 28
```

Types de donnée

R peut manipuler des nombres entiers, des flottants (= nombres à virgule), des chaînes de caractère et des booléens (valeur vraie ou fausse) :

```
> age = 28 # Entier
> poids = 64.5 # Flottant
> nom = "Ngo bo" # Chaîne de caractère
> enseignant = TRUE # booléen
> etudiant = FALSE # booléen
> telephone = "01 48 38 73 34" # Chaîne de caractère !
```

La valeur spéciale *NA* (non available) est utilisée lorsqu'une donnée est manquante.

Tableaux-Vecteurs

Un vecteur est un tableau à une dimension. Toutes les cases du vecteur doivent contenir des données du même type (des entiers, des chaînes de caractère,...). *La fonction c() permet de créer un vecteur :*

```
> ages = c(28, 25, 23, 24, 26, 23, 21, 22, 24, 29, 24,
26, 31, 28, 27, 24, 23, 25, 27, 25, 24, 21, 24, 23, 25, 31,
28, 27, 24, 23)
> ages
[1] 28 25 23 24 26 23 21 22 24 29 24 26 31 28 27 24 23
[17] 25 27 25 24 21 24 23 25 31 28 27 24 23
```

[**1**] indique que ce qui suit est la première valeur du vecteur, et
[**17**] que la deuxième ligne commence à la 17ème valeur ;
[**1**] et [**17**] ne sont pas des éléments du vecteur.

Tableaux-Vecteurs

La fonction c() permet aussi de concaténer (mettre bout à bout) des vecteurs :

```
> poids_groupe_temoin = c(75.0, 69.2, 75.4, 87.3)
> poids_groupe_intervention = c(70.5, 64.2, 76.4, 81.6)
> poids = c(poids_groupe_temoin,
poids_groupe_intervention)
> poids
[1] 75.0 69.2 75.4 87.3 70.5 64.2 76.4 81.6
```

Il est possible d'accéder à un élément du vecteur avec des crochets. Par exemple pour accéder au second élément du vecteur poids :

```
> poids[2]
[1] 69.2
```

Tableaux-Vecteurs

Il est aussi possible d'accéder à l'ensemble des poids répondant à une condition, par exemple l'ensemble des poids supérieurs à 70.0 : `>poids[poids > 70.0]`

```
[1] 75.0 75.4 87.3 70.5 76.4 81.6
```

Enfin, la fonction **length()** permet de récupérer le nombre d'éléments d'un tableau :

```
> length(poids)
[1] 8
> length(poids[poids > 70.0])
[1] 6
```

Tableaux-Vecteurs

Il est possible de créer un vecteur contenant une suite de nombres entiers avec la fonction **seq** :

```
> seq(1, 10)
[1] 1 2 3 4 5 6 7 8 9 10
> seq(1, 10, by = 0.5)
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0
[14] 7.5 8.0 8.5 9.0 9.5 10.0
```

Tableaux-Matrices

Une matrice est un tableau à deux dimensions, toutes les cases doivent contenir des données du même type.
Une matrice est créée à partir d'un vecteur contenant les valeurs, et d'un nombre de ligne (nr, pour Number of Row) et de colonne (nc, pour Number of Column) :

```
> ma_matrice = matrix(c(1.5, 2.1, 3.2, 1.6, 1.4,
1.5),nr=3, nc=2)
> ma_matrice
[,1] [,2]
[1,] 1.5 1.6
[2,] 2.1 1.4
[3,] 3.2 1.5
```

Tableaux-Matrices

Les éléments de la matrice peuvent être accédé en donnant entre crochets le numéro de la ligne puis celui de la colonne :

```
> ma_matrice[1, 1]
[1] 1.5
```

Il est aussi possible de récupérer une ligne ou une colonne entière, en omettant le numéro correspondant :

```
> ma_matrice[1,]
[1] 1.5 1.6
```

Tableaux-Liste,Data frames

Une liste est un tableau à une dimension, qui peut contenir des données de différents types (contrairement au vecteur).

```
> ma_liste = list("JB", 28)
```

- Tableaux de données (data frame)

Un tableau de donnée est un tableau où chaque colonne correspond à un attribut différents (âge, taille, poids par exemple) et chaque ligne à un individu différent.
Il est possible de créer des tableaux de données dans R, cependant il est beaucoup plus facile de les charger à partir d'un fichier. On utilise pour cela la fonction **read.table()**.

Tableaux-Data frames

Voici le fichier **taille_poids.csv** :

```
nom,taille,poids
Ngo bo,1.70,64.0
Bo bo,1.80,63.0
M. X,1.67,70.5
M. Y,1.69,95.0
M. Z,1.75,NA
```

Ce fichier peut être chargé ainsi dans **R** :

```
t = read.table("taille_poids.csv", sep=",", header=TRUE)
```

Tableaux-Data frames

Les noms des colonnes sont disponibles via la fonction **names()** :

```
> names(t)
[1] "nom" "taille" "poids"
```

Comme pour les matrices, il est possible d'accéder aux cases, lignes et colonnes d'un tableau. Les noms des colonnes peuvent être utilisés à la place de leurs index :

```
> t[1, 2] # taille du premier individu
[1] 1.7
> t[1, "taille"] # Pareil
[1] 1.7
> t[1,] # Première ligne
       nom taille poids
1 NgoBaoCho 1.7 64
```

Tableaux-Data frames

```
 > t[,"taille"] # Colonne taille
[1] 1.70 1.80 1.67 1.69 1.75
> t$taille # Notation raccourci pour la colonne taille
[1] 1.70 1.80 1.67 1.69 1.75
```

Il est aussi possible d'indexer avec une condition : par exemple, pour obtenir un tableau avec seulement les individus dont la taille est supérieure à 1m70 (ne pas oublier la virgule, qui sert à indiquer que l'on veut récupérer toutes les colonnes !) :

```
 > t[t$taille > 1.7,]
     nom taille poids
2 NgoBaoCho 1.80 63
5 M. Z 1.75 NA
```

Opérateur et calcul

R permet de réaliser la plupart des opérations courantes à l'aide des opérateurs suivants : + (addition), - (soustraction), * (multiplication), / (division), $\hat{}$(puissance).

```
> 2 * 3 + 1
[1] 7
```

Les opérations sont réalisées sur chaque élément des tableaux. Par exemple, pour calculer l'Indice de Masse Corporelle (IMC) sur chaque individu du tableau de donnée chargé précédemment, en appliquant la formule $IMC = \frac{poid}{taille^2}$:

```
> t$poids / (t$taille^2)
[1] 22.14533 19.44444 25.27878 33.26214 NA
```

Opérateur et calcul

Il est possible d'ajouter une quatrième colonne avec l'IMC à notre tableau de la manière suivante : `> t$IMC = t$poids / (t$taille^2)`

```
> t
      nom taille poids IMC
1 NgoBaocho 1.70 64.0 22.14533
2 NgoBaoCho 1.80 63.0 19.44444
3 M. X 1.67 70.5 25.27878
4 M. Y 1.69 95.0 33.26214
5 M. Z 1.75 NA NA
```

Comparaison

Les comparaisons se font avec les opérateurs $<$, $>$, $<=$ (inférieur ou égal) , $>=$ (suppérieur ou égal), $==$ (égal), $!=$ (différent de). Ils retournent une (ou plusieurs) valeur(s) booléenne(s).

```
> 1 < 3
[1] TRUE
> t$IMC > 25
[1] FALSE FALSE TRUE TRUE NA
```

Il est possible de combiner plusieurs comparaisons avec des & (et) ou des |(ou)

Fonctions

R définit un grand nombre de fonctions ; nous en avons déjà vu quelques unes. La fonction **help()** permet d'obtenir de l'aide sur une fonction :

```
>help(mean)
```

Voici une liste des principales fonctions :

- **summary(tableau)** affiche un résumé du tableau
- **length(vecteur)** retourne le nombre de case d'un vecteur
- **ncol(tableau)** retourne le nombre de colonne d'un tableau
- **nrow(tableau)** retourne le nombre de ligne d'un tableau
- **q()** quitte R
- **min(vecteur)** retourne la plus petite valeur d'un vecteur
- **max(vecteur)** retourne la plus grande valeur d'un vecteur
- **sort(vecteur)** retourne une copie d'un vecteur après l'avoir trié

Fonctions

- **mean(vecteur)** calcule la moyenne
- **median(vecteur)** calcule la médianne
- **var(vecteur)** calcule la variance
- **sd(vecteur)** calcule la déviation standard
- **sqrt(nombre)** retourne la racine carré d'un nombre
- **sum(vecteur)** retourne la somme de toutes les valeurs du vecteur
- **round(flottant)** arrondit un nombre

Calculer la moyenne des poids ?

Structures de contrôle

Structures de contrôle : commandes qui contrôlent l'ordre dans lequel les différentes instructions d'un programme informatique sont exécutées.

- Alternatives (énoncés conditionnels) : **if ... else**,
- Boucles (énoncés itératifs) : **for**, **while**, etc.

Structures de contrôle

Les alternatives ont pour but d'exécuter des instructions seulement si une certaine condition est satisfaite.

Écriture générale

```
if (condition) {
instructions
} else {
instructions
}
```

Exemple

```
if (x>0) y=x*log(x) else y=0
```

Structures de contrôle

Les boucles ont pour but de répéter des instructions à plusieurs reprises.

Écriture générale

```
for (i in ensemble ) {
instructions
}
```

Exemple

```
for (i in 1:10 ) {
print(i)
}
```

Structures de contrôle

Les boucles ont pour but de répéter des instructions à plusieurs reprises.

Écriture générale

```
while (condition ) {
instructions
}
```

Exemple

```
while (i <11 ) {
print(i)
i=i+1
}
```

Structures de contrôle

Les boucles ont pour but de répéter des instructions à plusieurs reprises.

Écriture générale

```
repeat {
instructions
if (condition) break
}
```

Exemple

```
repeat { print(i)
if (i<10) { i=i+1} else {break}
}
```

Fonctions

Écriture générale

Pour créer une fonction en R, on utilise la fonction nommée **function** en respectant la syntaxe suivante :

```
nomFonction <- function(arg1, arg2, arg3) {
# corps de la fonction }
```

- Les accolades { et } définissent le début et la fin de la fonction.
- La dernière instruction contient le nom de l'objet retourné par la fonction.
- Exécuter la fonction : `myname(...)`
- On peut donner des valeurs par défaut aux paramètre

Fonctions

Exemple

```
 exemple=function(n=10)
{ sample=runif(n)
m = mean(sample) ; v =var(sample)
list(serie=sample,moyenne=m, variance=v)
}
```

Les trois commandes suivantes :
`exemple(10)` ou `exemple(n=10)` ou `exemple()` retournent le même résultat.

Vous êtes ici

Arbre de décision

- Installer le package rpart : `install.packages("rpart", dep=TRUE)`
- Charger le package rpart : `library(rpart)`
- Charger les données kyphosis : `data(kyphosis)`
  - La variable Kyphosis indique si l'enfant a subi une déformation de la colonne vertébrale après une opération chirurgicale ;
  - La variable Age donne l'âge de l'enfant (en mois) ;
  - La variable Number indique le nombre de vertèbres concernées par l'opération ;
  - La variable Start indique le numéro de la première vertèbre opérée.

Arbre de décision

- Installer le package rpart : `install.packages("rpart", dep=TRUE)`
- Charger le package rpart : `library(rpart)`
- Charger les données kyphosis : `data(kyphosis)`
  - La variable Kyphosis indique si l'enfant a subi une déformation de la colonne vertébrale après une opération chirurgicale ;
  - La variable Age donne l'âge de l'enfant (en mois) ;
  - La variable Number indique le nombre de vertèbres concernées par l'opération ;
  - La variable Start indique le numéro de la première vertèbre opérée.

Arbre de décision

- Installer le package rpart : `install.packages("rpart", dep=TRUE)`
- Charger le package rpart : `library(rpart)`
- Charger les données kyphosis : `data(kyphosis)`
  - La variable Kyphosis indique si l'enfant a subi une déformation de la colonne vertébrale après une opération chirurgicale ;
  - La variable Age donne l'âge de l'enfant (en mois) ;
  - La variable Number indique le nombre de vertèbres concernées par l'opération ;
  - La variable Start indique le numéro de la première vertèbre opérée.

Arbre de décision

- Installer le package rpart : `install.packages("rpart", dep=TRUE)`
- Charger le package rpart : `library(rpart)`
- Charger les données kyphosis : `data(kyphosis)`
  - La variable Kyphosis indique si l'enfant a subi une déformation de la colonne vertébrale après une opération chirurgicale ;
  - La variable Age donne l'âge de l'enfant (en mois) ;
  - La variable Number indique le nombre de vertèbres concernées par l'opération ;
  - La variable Start indique le numéro de la première vertèbre opérée.

Arbre de décision

- Installer le package rpart : `install.packages("rpart", dep=TRUE)`
- Charger le package rpart : `library(rpart)`
- Charger les données kyphosis : `data(kyphosis)`
  - La variable Kyphosis indique si l'enfant a subi une déformation de la colonne vertébrale après une opération chirurgicale ;
  - La variable Age donne l'âge de l'enfant (en mois) ;
  - La variable Number indique le nombre de vertèbres concernées par l'opération ;
  - La variable Start indique le numéro de la première vertèbre opérée.

Arbre de décision

- Installer le package rpart : `install.packages("rpart", dep=TRUE)`
- Charger le package rpart : `library(rpart)`
- Charger les données kyphosis : `data(kyphosis)`
  - La variable Kyphosis indique si l'enfant a subi une déformation de la colonne vertébrale après une opération chirurgicale ;
  - La variable Age donne l'âge de l'enfant (en mois) ;
  - La variable Number indique le nombre de vertèbres concernées par l'opération ;
  - La variable Start indique le numéro de la première vertèbre opérée.

Arbre de décision

- Installer le package rpart : `install.packages("rpart", dep=TRUE)`
- Charger le package rpart : `library(rpart)`
- Charger les données kyphosis : `data(kyphosis)`
  - La variable Kyphosis indique si l'enfant a subi une déformation de la colonne vertébrale après une opération chirurgicale ;
  - La variable Age donne l'âge de l'enfant (en mois) ;
  - La variable Number indique le nombre de vertèbres concernées par l'opération ;
  - La variable Start indique le numéro de la première vertèbre opérée.

Exemple

On peut se demander si l'âge de l'enfant, le nombre de vertèbres opérées et la position des vertèbres opérées permettent d'évaluer le risque de survenue d'une cyphose à l'issue de l'opération.

Arbre de décision : Constrcution

On va construire un modèle d'arbre de décision grâce à la commande :
`arbre = rpart(Kyphosis ∼ ., method="class", minsplit=20, xval=81, data=kyphosis)`

- **arbre** est le nom choisi pour l'objet R qui contiendra les résultats ;
- **Kyphosis ∼** : La **tilde** signifie « est à expliquer en fonction de » : on met donc à gauche de la tilde la variable que l'on souhaite expliquer (ici, Kyphosis) et à droite les variables explicatives séparées par des + ;
- **method = "class"** : optionnel, mais il permet de rappeler à R que la variable à expliquer est qualitative ;
- **minsplit=20** signifie qu'il faut au moins 20 individus dans un nœud pour tenter un split ;
- l'argument **xval** permet de régler la stratégie de validation croisée ;
- L'argument **data=kyphosis** indique le nom du jeu de données utilisé.

Arbre de décision : Constrcution

On va construire un modèle d'arbre de décision grâce à la commande :
`arbre = rpart(Kyphosis ~ ., method="class", minsplit=20, xval=81, data=kyphosis)`

- **arbre** est le nom choisi pour l'objet R qui contiendra les résultats ;
- **Kyphosis** $\sim$: La **tilde** signifie « est à expliquer en fonction de » : on met donc à gauche de la tilde la variable que l'on souhaite expliquer (ici, Kyphosis) et à droite les variables explicatives séparées par des $+$;
- **method = "class"** : optionnel, mais il permet de rappeler à R que la variable à expliquer est qualitative ;
- **minsplit=20** signifie qu'il faut au moins 20 individus dans un nœud pour tenter un split ;
- l'argument **xval** permet de régler la stratégie de validation croisée ;
- L'argument **data=kyphosis** indique le nom du jeu de données utilisé.

Arbre de décision : Constrcution

On va construire un modèle d'arbre de décision grâce à la commande :
`arbre = rpart(Kyphosis ~ ., method="class", minsplit=20, xval=81, data=kyphosis)`

- **arbre** est le nom choisi pour l'objet R qui contiendra les résultats ;
- **Kyphosis** $\sim$: La **tilde** signifie « est à expliquer en fonction de » : on met donc à gauche de la tilde la variable que l'on souhaite expliquer (ici, Kyphosis) et à droite les variables explicatives séparées par des $+$;
- **method = "class"** : optionnel, mais il permet de rappeler à R que la variable à expliquer est qualitative ;
- **minsplit=20** signifie qu'il faut au moins 20 individus dans un nœud pour tenter un split ;
- l'argument **xval** permet de régler la stratégie de validation croisée ;
- L'argument **data=kyphosis** indique le nom du jeu de données utilisé.

Arbre de décision : Constrcution

On va construire un modèle d'arbre de décision grâce à la commande :
`arbre = rpart(Kyphosis ~ ., method="class", minsplit=20, xval=81, data=kyphosis)`

- **arbre** est le nom choisi pour l'objet R qui contiendra les résultats ;
- **Kyphosis** $\sim$: La **tilde** signifie « est à expliquer en fonction de » : on met donc à gauche de la tilde la variable que l'on souhaite expliquer (ici, Kyphosis) et à droite les variables explicatives séparées par des $+$;
- **method = "class"** : optionnel, mais il permet de rappeler à R que la variable à expliquer est qualitative ;
- **minsplit=20** signifie qu'il faut au moins 20 individus dans un nœud pour tenter un split ;
- l'argument **xval** permet de régler la stratégie de validation croisée ;
- L'argument **data=kyphosis** indique le nom du jeu de données utilisé.

Arbre de décision : Constrcution

On va construire un modèle d'arbre de décision grâce à la commande :
`arbre = rpart(Kyphosis ~ ., method="class", minsplit=20, xval=81, data=kyphosis)`

- **arbre** est le nom choisi pour l'objet R qui contiendra les résultats ;
- **Kyphosis** $\sim$: La **tilde** signifie « est à expliquer en fonction de » : on met donc à gauche de la tilde la variable que l'on souhaite expliquer (ici, Kyphosis) et à droite les variables explicatives séparées par des $+$;
- **method = "class"** : optionnel, mais il permet de rappeler à R que la variable à expliquer est qualitative ;
- **minsplit=20** signifie qu'il faut au moins 20 individus dans un nœud pour tenter un split ;
- l'argument **xval** permet de régler la stratégie de validation croisée ;
- L'argument **data=kyphosis** indique le nom du jeu de données utilisé.

Arbre de décision : Constrcution

On va construire un modèle d'arbre de décision grâce à la commande :
`arbre = rpart(Kyphosis ~ ., method="class", minsplit=20, xval=81, data=kyphosis)`

- **arbre** est le nom choisi pour l'objet R qui contiendra les résultats ;
- **Kyphosis** $\sim$: La **tilde** signifie « est à expliquer en fonction de » : on met donc à gauche de la tilde la variable que l'on souhaite expliquer (ici, Kyphosis) et à droite les variables explicatives séparées par des $+$;
- **method = "class"** : optionnel, mais il permet de rappeler à R que la variable à expliquer est qualitative ;
- **minsplit=20** signifie qu'il faut au moins 20 individus dans un nœud pour tenter un split ;
- l'argument **xval** permet de régler la stratégie de validation croisée ;
- L'argument **data=kyphosis** indique le nom du jeu de données utilisé.

Arbre de décision : Affichage

Les commandes suivantes permettent d'afficher l'arbre de décision sous forme graphique :

- `> plot(arbre, uniform=TRUE, margin=0.1, main="Arbre de décision")`
- `> text(arbre, fancy=TRUE, use.n=TRUE, pretty=0, all=TRUE)`

Arbre de décision : Affichage

Les commandes suivantes permettent d'afficher l'arbre de décision sous forme graphique :

- `> plot(arbre, uniform=TRUE, margin=0.1, main="Arbre de décision")`
- `> text(arbre, fancy=TRUE, use.n=TRUE, pretty=0, all=TRUE)`

On peut souhaiter élaguer l'arbre de décision. Pour cela, il est conseillé d'exécuter la commande **plotcp(arbre)** pour déterminer la taille optimale (ou **printcp(arbre)**).

- `> arbre2 = rpart(Kyphosis ~ ., cp=0.059, data=kyphosis)`

Arbre de décision : Affichage

Les commandes suivantes permettent d'afficher l'arbre de décision sous forme graphique :

- `> plot(arbre, uniform=TRUE, margin=0.1, main="Arbre de décision")`
- `> text(arbre, fancy=TRUE, use.n=TRUE, pretty=0, all=TRUE)`

On peut souhaiter élaguer l'arbre de décision. Pour cela, il est conseillé d'exécuter la commande **plotcp(arbre)** pour déterminer la taille optimale (ou **printcp(arbre)**).

- `> arbre2 = rpart(Kyphosis ~ ., cp=0.059, data=kyphosis)`

Arbre de décision : Prédiction

On peut être conduit à appliquer ultérieurement ces règles sur de nouveaux enfants qui vont être opérés, de manière à estimer avant l'opération leur risque de développer une cyphose. On construit arbitrairement un tableau de données correspondant à trois enfants sur le point d'être opérés :

- `> inco = data.frame(c(60,30,90), c(1,2,3), c(5,12,16))`
- `> colnames(inco) = c("Age", "Number", "Start")`

Arbre de décision : Prédiction

On peut être conduit à appliquer ultérieurement ces règles sur de nouveaux enfants qui vont être opérés, de manière à estimer avant l'opération leur risque de développer une cyphose. On construit arbitrairement un tableau de données correspondant à trois enfants sur le point d'être opérés :

- `> inco = data.frame(c(60,30,90), c(1,2,3), c(5,12,16))`
- `> colnames(inco) = c("Age", "Number", "Start")`

La prédiction de la variable Kyphosis pour ces trois individus s'opère alors par la commande **predict** (dont le premier argument est le nom du modèle utilisé, et le second, newdata, est le nom du data frame contenant les nouveaux individus) :

- `> predict(arbre2, newdata=inco, type="class")`

Arbre de décision : Prédiction

On peut être conduit à appliquer ultérieurement ces règles sur de nouveaux enfants qui vont être opérés, de manière à estimer avant l'opération leur risque de développer une cyphose. On construit arbitrairement un tableau de données correspondant à trois enfants sur le point d'être opérés :

- `> inco = data.frame(c(60,30,90), c(1,2,3), c(5,12,16))`
- `> colnames(inco) = c("Age", "Number", "Start")`

La prédiction de la variable Kyphosis pour ces trois individus s'opère alors par la commande **predict** (dont le premier argument est le nom du modèle utilisé, et le second, newdata, est le nom du data frame contenant les nouveaux individus) :

- `> predict(arbre2, newdata=inco, type="class")`

Vous êtes ici

Package arules

Nous allons utiliser une implantation en R de l'algorithme Apriori disponible dans le package arules de R.

- Installer le package **arules** via la commande : `install.packages("arules", dep=TRUE)`
- Charger le package **arules** : `library(arules)`

Il peut être nécessaire d'installer aussi le package **Matrix**.

Package arules

Nous allons utiliser une implantation en R de l'algorithme Apriori disponible dans le package arules de R.

- Installer le package **arules** via la commande : `install.packages("arules", dep=TRUE)`
- Charger le package **arules** : `library(arules)`

Il peut être nécessaire d'installer aussi le package **Matrix**.

Jeu de transactions

Nous allons utiliser une implantation en R de l'algorithme **Apriori** disponible dans le package **arules** de R. Nous allons travailler sur le jeu de transaction **Adult** :

```
data("Adult")
inspect(Adult[1:2])
```

- Combien y a-t-il de transactions ?
- Combien y a-t-il d'items ?
- Quel est la longueur des transactions ?
- Pourquoi certaines transactions ont-elles moins de 13 items ?

Jeu de transactions

Nous allons utiliser une implantation en R de l'algorithme **Apriori** disponible dans le package **arules** de R. Nous allons travailler sur le jeu de transaction **Adult** :

```
data("Adult")
inspect(Adult[1:2])
```

- Combien y a-t-il de transactions ?
- Combien y a-t-il d'items ?
- Quel est la longueur des transactions ?
- Pourquoi certaines transactions ont-elles moins de 13 items ?

Jeu de transactions

Nous allons utiliser une implantation en R de l'algorithme **Apriori** disponible dans le package **arules** de R. Nous allons travailler sur le jeu de transaction **Adult** :

```
data("Adult")
inspect(Adult[1:2])
```

- Combien y a-t-il de transactions ?
- Combien y a-t-il d'items ?
- Quel est la longueur des transactions ?
- Pourquoi certaines transactions ont-elles moins de 13 items ?

Jeu de transactions

Nous allons utiliser une implantation en R de l'algorithme **Apriori** disponible dans le package **arules** de R. Nous allons travailler sur le jeu de transaction **Adult** :

```
data("Adult")
inspect(Adult[1:2])
```

- Combien y a-t-il de transactions ?
- Combien y a-t-il d'items ?
- Quel est la longueur des transactions ?
- Pourquoi certaines transactions ont-elles moins de 13 items ?

Jeu de transactions

Nous allons utiliser une implantation en R de l'algorithme **Apriori** disponible dans le package **arules** de R. Nous allons travailler sur le jeu de transaction **Adult** :

```
data("Adult")
inspect(Adult[1:2])
```

- Combien y a-t-il de transactions ?
- Combien y a-t-il d'items ?
- Quel est la longueur des transactions ?
- Pourquoi certaines transactions ont-elles moins de 13 items ?

Regardez l'aide sur les fonctions **itemFrequency()** et **itemFrequencyPlot()**.

- Affichez le graphique des items de support supérieur à 0.2.
- Affichez le graphique des 10 items les plus fréquents.

Jeu de transactions

Nous allons utiliser une implantation en R de l'algorithme **Apriori** disponible dans le package **arules** de R. Nous allons travailler sur le jeu de transaction **Adult** :

```
data("Adult")
inspect(Adult[1:2])
```

- Combien y a-t-il de transactions ?
- Combien y a-t-il d'items ?
- Quel est la longueur des transactions ?
- Pourquoi certaines transactions ont-elles moins de 13 items ?

Regardez l'aide sur les fonctions **itemFrequency()** et **itemFrequencyPlot()**.

- Affichez le graphique des items de support supérieur à 0.2.
- Affichez le graphique des 10 items les plus fréquents.

La fonction apriori()

- Regardez l'aide sur cette fonction **apriori()**.
- Par exemple pour obtenir les règles ayant un support d'au moins 1% et une confiance supérieure à 60%, il suffit de lancer la commande :
  `rules<-apriori(Adult,parameter=list(support=0.01,confidence=0.6)`
- Comprenez les messages affichés lors de l'exécution de l'algorithme. Vous pouvez utiliser les fonctions **summary()** et **inspect()** pour répondre aux questions suivantes :
  - Combien de règles ?
  - Quel nombre de transactions ?
  - Afficher les dix premières règles générées.
- Diminuez la valeur de la confiance (par exemple 0.1) et du support (par exemple 0.001, 0.0001, 0.00001) et regardez l'évolution du nombre de règles.

La fonction apriori()

- Regardez l'aide sur cette fonction **apriori()**.
- Par exemple pour obtenir les règles ayant un support d'au moins 1% et une confiance supérieure à 60%, il suffit de lancer la commande :
  `rules<-apriori(Adult,parameter=list(support=0.01,confidence=0.6)`
- Comprenez les messages affichés lors de l'exécution de l'algorithme. Vous pouvez utiliser les fonctions **summary()** et **inspect()** pour répondre aux questions suivantes :
  - Combien de règles ?
  - Taille moyenne des transactions ?
  - Afficher les dix premières règles obtenues.
- Diminuez la valeur de la confiance (par exemple 0.1) et du support (par exemple 0.001, 0.0001, 0.00001) et regardez l'évolution du nombre de règles.

La fonction apriori()

- Regardez l'aide sur cette fonction **apriori()**.
- Par exemple pour obtenir les règles ayant un support d'au moins 1% et une confiance supérieure à 60%, il suffit de lancer la commande :
  `rules<-apriori(Adult,parameter=list(support=0.01,confidence=0.6)`
- Comprenez les messages affichés lors de l'exécution de l'algorithme. Vous pouvez utiliser les fonctions **summary()** et **inspect()** pour répondre aux questions suivantes :
  - Combien de règles ?
  - Pour combien de transactions ?
  - Affichez les dix premières règles obtenues.
- Diminuez la valeur de la confiance (par exemple 0.1) et du support (par exemple 0.001, 0.0001, 0.00001) et regardez l'évolution du nombre de règles.

La fonction apriori()

- Regardez l'aide sur cette fonction **apriori()**.
- Par exemple pour obtenir les règles ayant un support d'au moins 1% et une confiance supérieure à 60%, il suffit de lancer la commande :
  `rules<-apriori(Adult,parameter=list(support=0.01,confidence=0.6)`
- Comprenez les messages affichés lors de l'exécution de l'algorithme. Vous pouvez utiliser les fonctions **summary()** et **inspect()** pour répondre aux questions suivantes :
  - Combien de règles ?
  - Pour combien de transactions ?
  - Affichez les dix premières règles obtenues.
- Diminuez la valeur de la confiance (par exemple 0.1) et du support (par exemple 0.001, 0.0001, 0.00001) et regardez l'évolution du nombre de règles.

La fonction apriori()

- Regardez l'aide sur cette fonction **apriori()**.
- Par exemple pour obtenir les règles ayant un support d'au moins 1% et une confiance supérieure à 60%, il suffit de lancer la commande :
  `rules<-apriori(Adult,parameter=list(support=0.01,confidence=0.6)`
- Comprenez les messages affichés lors de l'exécution de l'algorithme. Vous pouvez utiliser les fonctions **summary()** et **inspect()** pour répondre aux questions suivantes :
  - Combien de règles ?
  - Pour combien de transactions ?
  - Affichez les dix premières règles obtenues.
- Diminuez la valeur de la confiance (par exemple 0.1) et du support (par exemple 0.001, 0.0001, 0.00001) et regardez l'évolution du nombre de règles.

La fonction apriori()

- Regardez l'aide sur cette fonction **apriori()**.
- Par exemple pour obtenir les règles ayant un support d'au moins 1% et une confiance supérieure à 60%, il suffit de lancer la commande :
  `rules<-apriori(Adult,parameter=list(support=0.01,confidence=0.6)`
- Comprenez les messages affichés lors de l'exécution de l'algorithme. Vous pouvez utiliser les fonctions **summary()** et **inspect()** pour répondre aux questions suivantes :
  - Combien de règles ?
  - Pour combien de transactions ?
  - Affichez les dix premières règles obtenues.
- Diminuez la valeur de la confiance (par exemple 0.1) et du support (par exemple 0.001, 0.0001, 0.00001) et regardez l'évolution du nombre de règles.

La fonction apriori()

- Regardez l'aide sur cette fonction **apriori()**.
- Par exemple pour obtenir les règles ayant un support d'au moins 1% et une confiance supérieure à 60%, il suffit de lancer la commande :
  `rules<-apriori(Adult,parameter=list(support=0.01,confidence=0.6)`
- Comprenez les messages affichés lors de l'exécution de l'algorithme. Vous pouvez utiliser les fonctions **summary()** et **inspect()** pour répondre aux questions suivantes :
  - Combien de règles ?
  - Pour combien de transactions ?
  - Affichez les dix premières règles obtenues.
- Diminuez la valeur de la confiance (par exemple 0.1) et du support (par exemple 0.001, 0.0001, 0.00001) et regardez l'évolution du nombre de règles.

Vous êtes ici

Package cluster

Nous allons utiliser une implantation en R de l'algorithme k-moyennes disponible dans le package cluster de R.

- Installer le package **cluster** via la commande : `install.packages("cluster", dep=TRUE)`
- Charger le package **cluster** : `library(cluster)`
- Regardez l'aide sur cette fonction **kmeans()**. Que fournit cette commande ?
- Visualisez les données clustering fournies avec ce TP, combien de clusters pensez-vous observer ?
- Visualiser le résultat de la segmentation avec k=2, à l'aide de couleurs.

Package cluster

Nous allons utiliser une implantation en R de l'algorithme k-moyennes disponible dans le package cluster de R.

- Installer le package **cluster** via la commande : `install.packages("cluster", dep=TRUE)`
- Charger le package **cluster** : `library(cluster)`
- Regardez l'aide sur cette fonction **kmeans()**. Que fournit cette commande ?
- Visualisez les données clustering fournies avec ce TP, combien de clusters pensez-vous observer ?
- Visualiser le résultat de la segmentation avec k=2, à l'aide de couleurs.

Package cluster

Nous allons utiliser une implantation en R de l'algorithme k-moyennes disponible dans le package cluster de R.

- Installer le package **cluster** via la commande : `install.packages("cluster", dep=TRUE)`
- Charger le package **cluster** : `library(cluster)`
- Regardez l'aide sur cette fonction **kmeans()**. Que fournit cette commande ?
- Visualisez les données clustering fournies avec ce TP, combien de clusters pensez-vous observer ?
- Visualiser le résultat de la segmentation avec k=2, à l'aide de couleurs.

Package cluster

Nous allons utiliser une implantation en R de l'algorithme k-moyennes disponible dans le package cluster de R.

- Installer le package **cluster** via la commande : `install.packages("cluster", dep=TRUE)`
- Charger le package **cluster** : `library(cluster)`
- Regardez l'aide sur cette fonction **kmeans()**. Que fournit cette commande ?
- Visualisez les données clustering fournies avec ce TP, combien de clusters pensez-vous observer ?
- Visualiser le résultat de la segmentation avec k=2, à l'aide de couleurs.

Package cluster

Nous allons utiliser une implantation en R de l'algorithme k-moyennes disponible dans le package cluster de R.

- Installer le package **cluster** via la commande : `install.packages("cluster", dep=TRUE)`
- Charger le package **cluster** : `library(cluster)`
- Regardez l'aide sur cette fonction **kmeans()**. Que fournit cette commande ?
- Visualisez les données clustering fournies avec ce TP, combien de clusters pensez-vous observer ?
- Visualiser le résultat de la segmentation avec k=2, à l'aide de couleurs.

Trouver K optimal

On s'intéresse maintenant à trouver le nombre de groupes optimal. Pour cela, on va essayer plusieurs valeurs de K qui semblent raisonnables.

- Essayez toutes les valeurs entre 2 et 10.
- Placez ces segmentations dans une liste.
- Calculez l'inertie intraclasse de ces 9 segmentations.
- Faites-en un graphe.
- Quelle segmentation est la meilleure ?

Trouver K optimal

On s'intéresse maintenant à trouver le nombre de groupes optimal. Pour cela, on va essayer plusieurs valeurs de K qui semblent raisonnables.

- Essayez toutes les valeurs entre 2 et 10.
- Placez ces segmentations dans une liste.
- Calculez l'inertie intraclasse de ces 9 segmentations.
- Faites-en un graphe.
- Quelle segmentation est la meilleure ?

Trouver K optimal

On s'intéresse maintenant à trouver le nombre de groupes optimal. Pour cela, on va essayer plusieurs valeurs de K qui semblent raisonnables.

- Essayez toutes les valeurs entre 2 et 10.
- Placez ces segmentations dans une liste.
- Calculez l'inertie intraclasse de ces 9 segmentations.
- Faites-en un graphe.
- Quelle segmentation est la meilleure ?

Trouver K optimal

On s'intéresse maintenant à trouver le nombre de groupes optimal. Pour cela, on va essayer plusieurs valeurs de K qui semblent raisonnables.

- Essayez toutes les valeurs entre 2 et 10.
- Placez ces segmentations dans une liste.
- Calculez l'inertie intraclasse de ces 9 segmentations.
- Faites-en un graphe.
- Quelle segmentation est la meilleure ?

Trouver K optimal

On s'intéresse maintenant à trouver le nombre de groupes optimal. Pour cela, on va essayer plusieurs valeurs de K qui semblent raisonnables.

- Essayez toutes les valeurs entre 2 et 10.
- Placez ces segmentations dans une liste.
- Calculez l'inertie intraclasse de ces 9 segmentations.
- Faites-en un graphe.
- Quelle segmentation est la meilleure ?