Formation RStudio niveau 2

Plan de la formation:

1- Importation
2- Indexation-Assignation
3- Mise en forme des variables
4- Manipuler les types de variables
5- Manipuler les données
6- Modifier les données
7- Représentation graphique
8- Exercices d'application (à voir si on a le temps)

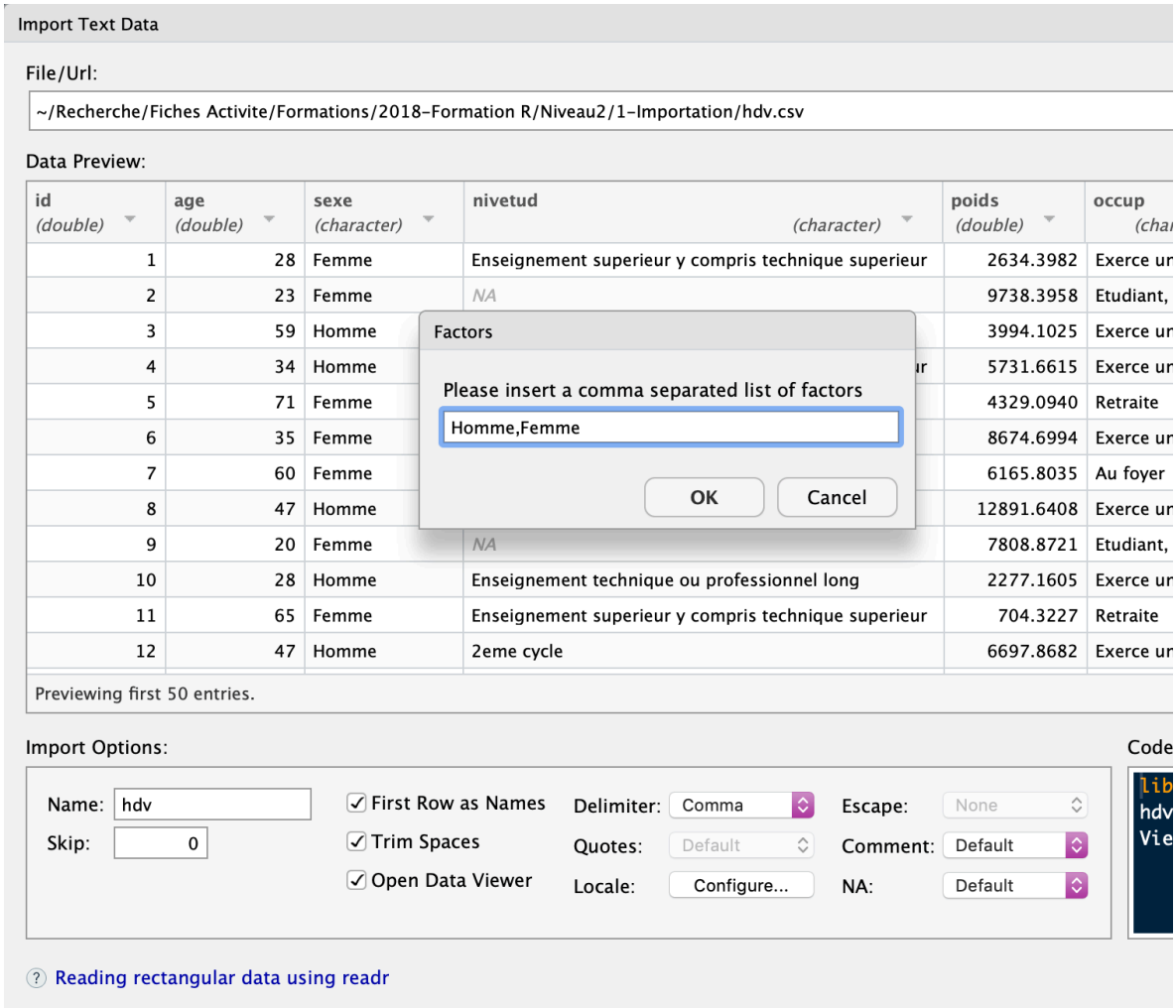
Rappel: Pour définir le dossier de travail par défaut faire: Tools -> Global options -> 1° ligne, chercher le dossier de travail souhaité.
Si l'on passe par la frame « files », le dossier de travail sera perdu au redémarrage.

1- Importation

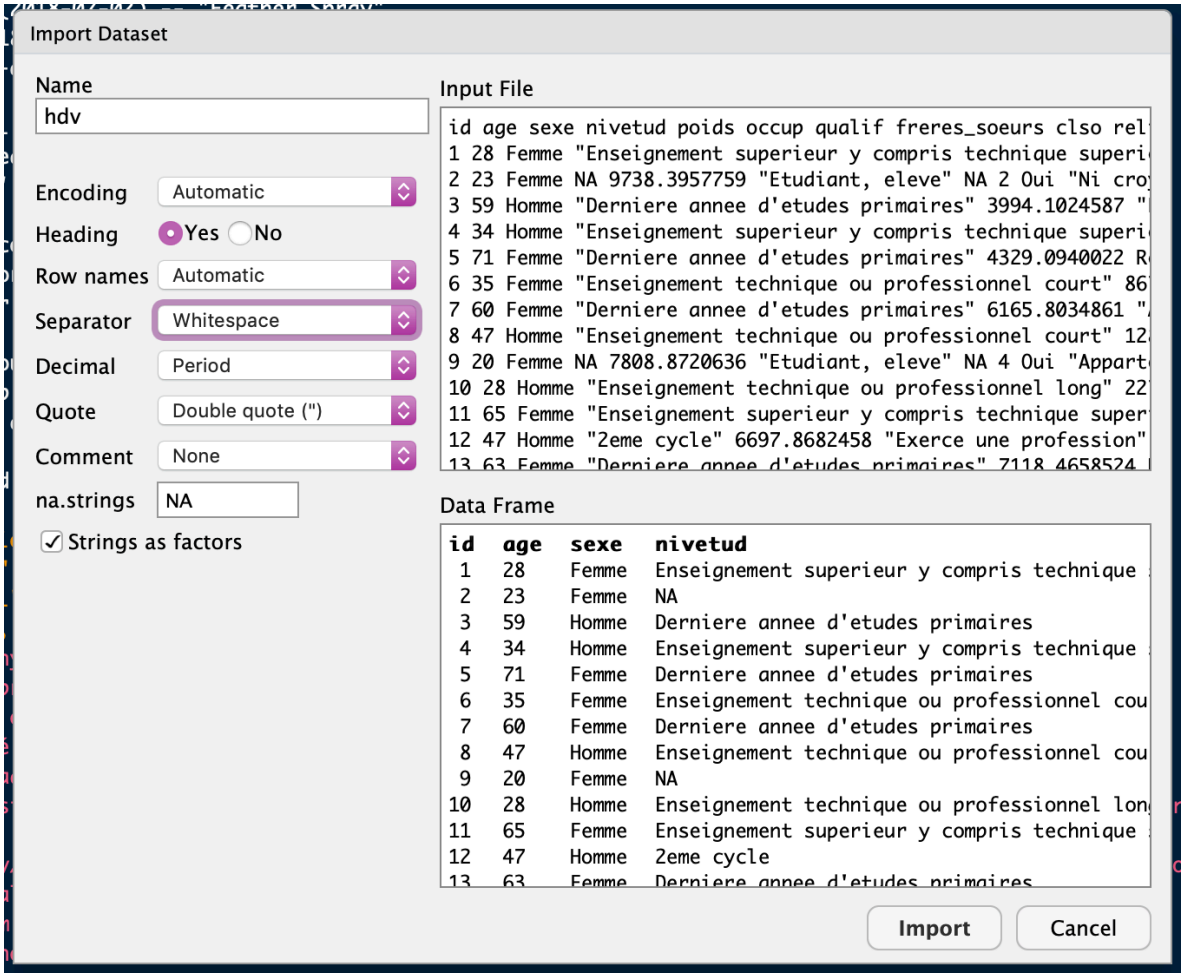
- Importer fichier SPSS: Frame Import -> fichier SDV. Dans Import Options (frame en B à D), le nom sera celui de l'objet une fois importé.
- Importer fichier excel, il faut attribuer le type de variable: dans la fenêtre importation, cliquer sur id et choisir le type (Date, Numéric ou Character). Dans import option, il y a « Skip »: permet d'ignorer les premières lignes (choisir le nombre)
Pour annuler l'objet importé, balayette dans la frame Environment.

- Rappel: pour assigner un fichier (exemple fichier dbf) à un objet, il faut écrire d'abord le nom du futur objet suivi de la flèche pour assignation:
> happy <- read.dbf(file = "~/Recherche/Fiches Activite/Formations/2018-Formation R/Niveau2/1-Importation/happy.dbf",as.is = FALSE)
La mention as.if = false permet de dire qu'il doit convertir des character en nombre (exemple: homme sera codé par 2)

- Importer fichier CSV. On peut transformer Character en facteur (cliquer sur sexe/ character et sélectionner factors. Suite voir image)



- Importation txt: On peut cocher la case String as factors pour transformer toutes les variables character en facteurs numériques



2- Indexation/Assignation (scripts_1)

Une indexation est une sélection dans un tableau (par position, Nom ou Condition)
Syntaxe générale: objet_BDD [x,y] *Remarque: pour faire un crochet sur Mac, faire Alt+Maj+(*
x correspond aux lignes, y correspond à colonnes.

a- Indexation par position:

- Pour sélectionner la 3° colonne: **hdv [,3]** va sélectionner la 3° colonne (autre méthode: hdv$sexe par contre le $ permet de ne sélectionner qu'une variable)
- Pour exclure une variable: **hdv [,-3]** Va exclure la 3° colonne. Pour exclure plusieurs colonnes: **hdv [,-c(6,7,9,11,12)]**

Idem pour ligne.

- Pour sélectionner une case: remplir x ET y: **hdv [72,3]**
- Pour sélectionner un ensemble de ligne et de colonnes: **hdv [,1:4]** va sélectionner les 4 premières colonnes. **hdv [50:145,5:8]** va sélectionner les lignes 50 à 145 et les colonnes 5 à 8
- Pour vérifier si la plage sélectionnée est bonne, on peut ajouter dim pour dimension: **dim(hdv [50:145,5:8])** Il va renvoyer 94 4 c.a.d 96 lignes et 4 colonnes
- Pour sélectionner des colonnes non contigues, il faut mettre les plages de variable dans une parenthèse avec « c » devant (c pour concaténation):

dim(hdv[,c(4,8,9,14)])
- Pour sélectionner des lignes non contigues: **dim(hdv[c(14,80,169,1568),])**
- Pour sélectionner en excluant, c'est pareil mais on mets le signe - devant.
dim(hdv[,-c(1:4)]) va exclure les 4 premières colonnes. **dim(hdv[,c(2:4,6:8)])** va sélectionner les colonnes 2 à 4 et 6 à 8.
Remarque: on ne peut pas mettre ensemble des positifs et des négatifs
- Fonction similaire: Slice. Permet de ne sélectionner que des lignes (évite l'emploi des crochets) **dim(slice(hdv,c(1,2,4:15,80,90,1284,1547)))**. Sélectionnera donc toutes les colonnes

Select: Permet de ne sélectionner que des colonnes

b- Indexation par condition:

Les conditions doivent toujours être entre crochet
Opérateurs: Egal == ; Différent !=; >; >=; <; <=
- Pour sélectionner les lignes « Homme » et toutes les colonnes:
dim(hdv[hdv$sexe=="Homme",])
- Pour sélectionner tout sauf une catégorie: **dim(hdv[hdv$qualif !="Cadre",])** sélectionne tout le monde sauf les cadres
- Pour mettre plusieurs conditions:
Exemple: dim(hdv[hdv$age >45,c(4,8)]) va sélectionner les individus de plus de 45 ans et les colonnes 4 et 8
Si 2 conditions, utiliser &: hdv[(hdv$sexe==« Femme »)&(hdv$age >45)]

OPERATEUR ET: **&**

##Sélectionner les femmes étudiantes entre 18 ans (compris) ***et*** *20 ans (compris)*

```
dim(hdv[(hdv$sexe=="Femme")
      &(hdv$occup=="Etudiant, eleve")
      &(hdv$age>=18)
      &(hdv$age<=20),]
    )
```

OPERATEUR OU: **I**

##Sélectionner les hommes chômeur ***ou*** *au foyer entre 30* ***et*** *40 ans*

```
dim(hdv[(hdv$sexe=="Homme")
      &(hdv$age>=30)
      &(hdv$age<=40)
      &((hdv$occup=="Chomeur")
      I(hdv$occup=="Au foyer")),]
)
```

Autre possibilité si plusieurs opérations OU: lister toutes les conditions entre parenthèse avec la fonction %in%:

##Sélectionner les individus qui sont chômeurs ***ou*** *retraités*

```
dim(hdv[hdv$occup %in% c("Chomeur", "Retraite"), ])
```

Autre possibilité, la fonction filtre:

##Sélectionner les hommes chomeur ***ou*** *au foyer entre 30* ***et*** *40 ans avec la fonction "Filter"*

```
dim(filter(hdv,sexe=="Homme",
      age>=30,
      age<=40,
```

(occup=="Chomeur" | occup=="Au foyer")
))

Fonctions pour gérer les valeurs manquantes dans les sélections:
is.na(): renvoie TRUE si c'est une valeur manquante (sinon false)
!is.na: renvoie TRUE si ce n'est pas une valeur manquante (à ajouter si l'on ne veut garder QUE les valeurs non manquantes)

c- Assignation:

syntaxe générale: objet_à _créer <- objet/variable/selection
> sexe<-hdv$sexe Va créer un objet Sexe à partir de la colonne sexe du fichier hdv

Pour créer un nouvel objet (resume) contenant les colonnes 1,2,3,6,10
resume<-hdv[,c(1,2,3,6,10)]
Pour créer un nouvel objet (resume) ne contenant pas la colonne 5 (c'était la 10° dans le fichier hdv, mais c'est maintenant la 5°)
resume<-resume[,-5]

Créer une nouvelle variable
resume$age_2018 <-resume$age+15
resume$age_2018 <-resume[,2]+15
resume<-mutate(resume,age_2018=age+15)
Ces trois formules sont équivalentes. La fonction mutate est la seule pour laquelle on n'est pas obligé de préciser $colonne car il va ajouter une colonne. Dans TOUS les autres cas, ne pas préciser $colonne va écraser le fichier avec le nouveau !

Créer plusieurs nouvelles variables (âge des individus en 2025 et 2050)
resume<-mutate(resume,age_2025=age+22,age_2025=age+47)

Créer une nouvelle variable sexe2 avec la valeur H pour les Hommes
resume$sexe2[resume$sexe=="Homme"]<-"H"
ATTENTION: pas de virgule avant le crochet car une seule colonne (ce n'est pas une matrice)

##Créer une colonne sexe_age avec la valeur 0 pour toutes les lignes
resume$sexe_age<-0

##Pour les hommes de moins de 40 ans (compris), mettre la valeur "Hommes de moins de 40 ans"
resume$sexe_age[(resume$sexe=="Homme")
&resume$age<=40]<-"Homme de moins de 40 ans"

##Pour les hommes de moins de 40 ans (compris), mettre la valeur "Hommes de moins de 40 ans"
resume$sexe_age[(resume$sexe=="Femme")
&resume$age<=40]<-"Femmes de moins de 40 ans"

Pour vérifier les résultats: **freq(resume$sexe_age)**

3- Mise en forme des variables (scripts_2)

Importer le fichier INSEE en dbf. En dbf, on ne peut importer directement. Il faut l'assigner à un objet (ici mariage) et supprimer les valeurs manquantes. Pour récupérer rapidement l'adresse, faire une importation bidon et copier l'adresse du fichier dans la barre d'adresse (puis annuler l'import).
mariage<-read.dbf(file="~/Recherche/Fiches Activite/Formations/2018-Formation R/Niveau2/2-BDD_travail/mar2016_1.dbf",as.is=TRUE)
as.is=TRUE permet de tout mettre en caractères. Nous on s'en est servi juste pour apprendre à recoder si notre fichier de base est tout en caractères (la situation la pire).
Le nom de la variable est toujours court (nom de la colonne). Le label est parfois long (peut expliquer en détail la variable). Ne pas confondre avec « les labels » qui sont des étiquettes (ex: Urbain: 1 et Rural:2).

a- Renommer une variable: fonction Rename du package dplyr
syntaxe: rename(objet, nouvelle_variable=ancienne_variable
##Changer le nom de la variable "DEPMAR" par "DEP" et "NBENFCOM" par "NBEF"
mariage<-rename(mariage,DEP=DEPMAR, NBENF=NBENFCOM)
Il ne faut pas oublier de stocker le résultat dans mariage, sinon c'est juste un affichage dans la console.

b- Changer l'ordre des variables
utiliser la fonction select (package dplyr)
Pour changer l'ordre des colonnes ou ne garder QUE certaines colonnes
mariage<-select(mariage,DEP,SEXE1,SEXE2,ANAIS1,ANAIS2,NBENFCOM,ID)
Les colonnes n'apparaissant pas dans la parenthèse seront perdues.
Fonction everything: permet de ne reclasser qu'une partie des colonnes (ensuite, il laisse comme c'était) et garder les autres colonnes non triées quand même.
Par exemple, si on veut remettre ID en première position: **mariage<-select(mariage,ID,everything())**

c- Changer le label des variables (package « labelled »)
var_label(objet$variable)<-« nouveau_label »
Changer le label des variables
var_label(mariage$ANAIS1)<-"Année de naissance du conjoint 1"

4- Manipuler les types de variables (scripts_3)

Pour transformer une variable (ex: passer de character à integer) ce sera toujours **as.**
as.interger(): transforme en intégrer (nombre entier)
as.integer(): nbr entier
as.numeric(): nbr réel
as.character(): chaîne de caractères
as.logical(): booléen
as.factor(): catégorielle. Exemple Hommes (1), Femme (2) et valeur ,manquante

(99). 1 ne vaut pas 1. C'est une valeur codée. 1, 2, 99 sont des « levels »
as.ordered(): catégoriel ordonné ou ordinale (ex: mentions au bac. C'est du texte mais il y aune hiérarchie)

a- Créer un facteur (fonction du package labelled)

- variable de type character transformée en var. type facteur par la fonction **to_factor()**

La fonction levels(variable): permet d'afficher les modalités/niveaux

Transformer la variable SEXE1 en facteur

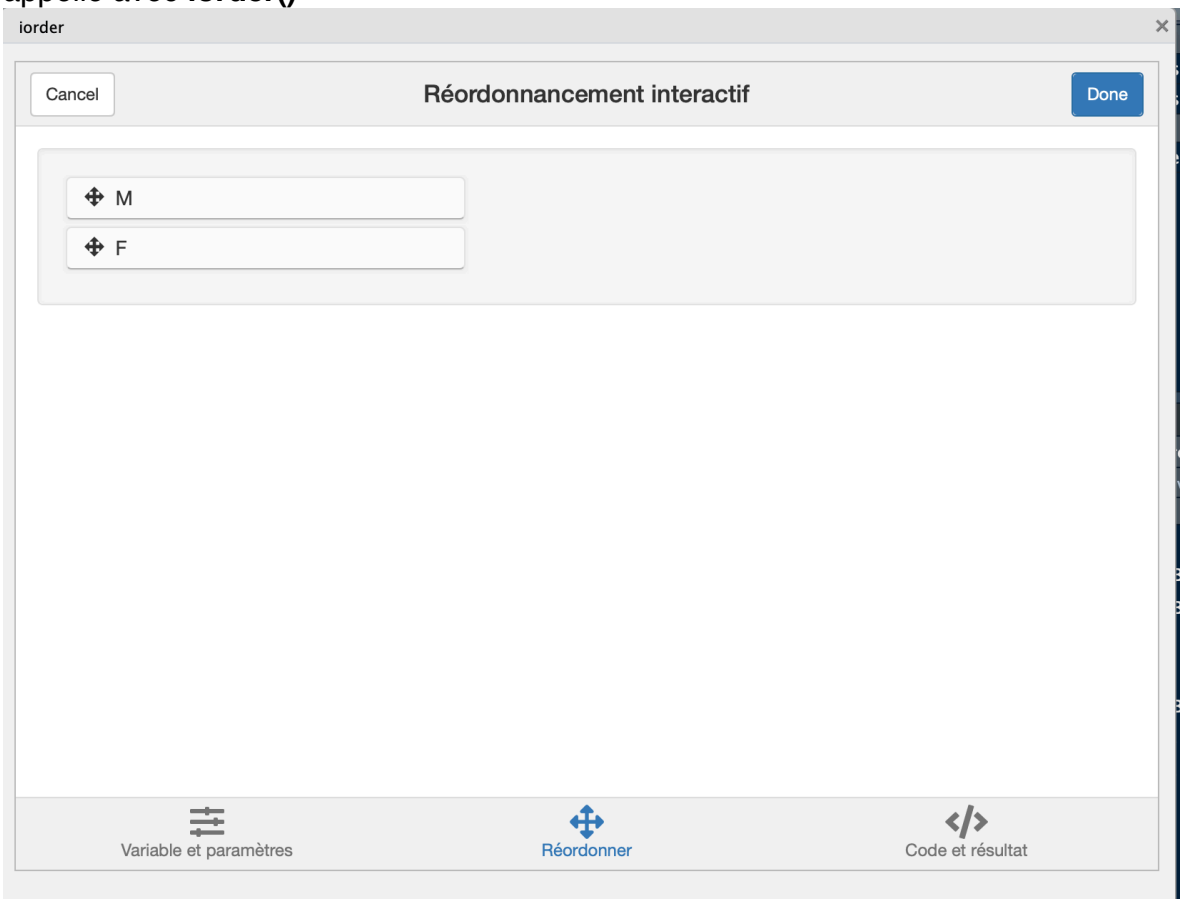
mariage1$SEXE1<-to_factor(mariage1$SEXE1)

La variable sexe était un char. C'est devenu un facteur (1 ou 2 pour H ou F)

Vérifier les niveaux créés

levels(mariage1$SEXE1)

Transformer les variables peut être fait grâce à une console interactive que l'on appelle avec **iorder()**



Pour changer les étiquettes des variables avec la console interactive **irec()**: il faut appeler la console grec en tapant **irec()**. Il suffit ensuite de copier le code dans la console ou le script.

Changer M et F de la variable SEXE1 en H et F avec la console interactive irec()

Recodage de mariage1$SEXE1 en mariage1$SEXE1_rec

mariage1$SEXE1_rec <- fct_recode(mariage1$SEXE1,
"H" = "M")

Le problème c'est que dans les variables de type **factor** ce sont les **labels** qui sont

appelés dans les traitements. Si les labels (modalités/niveaux) sont longs pour être explicites, ils prennent trop de temps à écrire. S'ils sont court, ils ne sont plus informatifs.
Donc on peut transformer le **type factor** en **type labelled** car on pourra utiliser des « label (modalité/niveau) explicites car on va sélectionner sur les valeurs (et plus les label).

b- Transformation des variables en labelled

La méthode dépend du type de variable au départ. Certaines fonctions de R ne marchent pas avec les labelled. Il est donc préférable de dupliquer les colonnes.
La fonction labelled transforme la VALEUR en LABEL. Si la variable est char., après labelled on aura des char., si la variable est factor (1,2,…) le labelled sera 1,2,…

- Création à partir d'un character, integer ou numeric:

Transformer SEXE2 en labelled en mettant "Homme" pour "M" et "Femmes" pour "F" en écrasant l'ancienne variable

mariage1$SEXE2<-labelled(mariage1$SEXE2,c(Homme="M",Femmes="F"))

- Création à partir d'un factor:

Syntaxe: **to_labelled(variable)**

Transformer SEXE1 et NBENFCOM en labelled

mariage1$SEXE1<-to_labelled(mariage1$SEXE1)
mariage1$NBENFCOM<-to_labelled(mariage1$NBENFCOM)

c- Transformer les étiquettes de valeurs, c.a.d les labels des valeurs (=les changer): fonction val_labels() et val_label()

- Si on veut changer toutes les étiquettes: **val_labels(variable)<-c(nouvelle_etiquette=valeur)**. Par exemple la valeur peut être 1 (pour H), ou à l'inverse H (voir le tableau).
Dans ce cas, on ne peut pas changer qu'une étiquette. Il faut les changer toutes.

##Changer les labels de SEXE2 pour que ce soit homme pour H et femme pour F

val_labels(mariage1$SEXE2)<-c("homme"="M","femme"="F")

Les valeurs « M » auront comme étiquette « homme » et « F » l'étiquette « femme »

- Si on veut changer une seule étiquette il faut utiliser la fonction **val_label(x=variable, v=valeur)<-**« **nouvelle_etiquette** ». « valeur » est la valeur que l'on veut qu'il trouve dans la variable.

Changer le label homme par le label Homme

val_label(x=mariage1$SEXE2,v="M")<-"Homme"

- Si on veut supprimer une étiquette: **val_label(x=mariage1$SEXE2, v=**« **homme** »**) <-NULL**

d- Transformer un labelled en facteur: fonction to_factor(variable)

On peut préciser si pour les labels du factor il prend les valeurs de labelled ou les labels de labelled.

Changer la variable SEXE2 en factor en utilisant les labels de labelled comme étiquette

mariage1$SEXE2 <- to_factor(mariage1$SEXE2, levels = "labels")

Changer la variable SEXE2 en factor en utilisant les valeurs de labelled comme

étiquette
mariage1$SEXE2 <- to_factor(mariage1$SEXE2, levels = "values")

Rmq: on peut transformer un ensemble de colonnes d'un seul coup !
to_factor(mariageF [,1:100],levels=« labels »)

4- Exercice d'application (Voir Scripts_4)

5- Manipuler les données

a- Trier un tableau de données

Si je trie en cliquant en haut des colonnes du tableau, c'est juste de l'affichage. Pour le faire « en dur »:
arrange(objet, variable1, variable2,…)
Par défaut, le tri se fait par ordre croissant.
Si on veut un tri décroissant: **arrange(objet, desc(variable1), variable2,…)**

Trier le tableau mariage1 par le département (décroissant), puis par le sexe du conjoint 1 (croissant)
mariage1<-arrange(mariage1, desc(DEP), SEXE1)

b- Faire des sous-ensembles

C'est sélectionner dans le dataframe un certain nombre de variables avec des conditions (les hommes de 45 ans)
On va imbriquer la fonction **select** (va sélectionner des colonnes selon un critère) dans la fonction **filter** (va sélectionner des lignes).
Syntaxe: **filter(select(data frame, variables),condition)**

c- Faire des jointures

C'est joindre 2 tables entre elles. Jointure gauche: le tableau de G est collé dans celui de droite. Il s'agit donc de rapatrier des infos d'une table sur un autre.
Il faut au moins une colonne commune.
left_join(table1,table2)
right_join(table1,table2) c'est l'inverse
inner_join(table1,table2): va limiter aux lignes en commun et ajouter les colonnes manquantes
full_join (table1,table2): ajoute les lignes ET les colonnes
semi-join: ne garde que les lignes communes aux deux tables mais ne rapatrie pas de colonnes
anti_join: sort les lignes de la table 1 qui ne sont pas présentes dans la table 2
Il ne va pas réellement « mixer » 2 colonnes au même intitulé (ex: SEXE). Il va créer 2 colonnes dans la même table (SEXE1 et SEXE2).
Syntaxe: **left_join(table1,table2,by=c(colonne_ID_table1,colonne_ID_table2)**
Colonne ID donne la colonne de référence entre les deux tables (colonne commune)
Si 2 colonnes SEXE, il y aura SEXE.x et SEXE.y (x:table 1, y:table 2)
Si on veut nommer le suffixe (au lieu de x et y):
left_join(table1,table2,by=c(colonne_ID_table1,colonne_ID_table2), suffix=c(« suffixe_table1 »,suffixe_table2 »)

d- Exercice d’application

Ramener les colonnes manquantes dans data depuis mariage1, sans suffixe pour la table data et en identifiant les colonnes provenant de mariage1 par un suffixe. Supprimer ensuite les colonnes en doublons provenant de mariage1

data<-left_join(data,mariage1,by=c("ID","ID"), suffix=c("","mar1"))
data<-data [,-c(6,7,9,11,12)]

En ne mettant pas de suffixe à la table 1 ("") cela évite d’avoir à les enlever après (au final, on veut les colonnes à l’identique, pas avec des suffixes en plus).
Au lieu d’enlever les colonnes en doublon, on peut aussi ne sélectionner QUE les colonnes qui nous intéressent avec la fonction select (permet aussi de les trier du coup)

data<-select(data,ID,DEP,SEXE1,SEXE2,ANAIS1,ANAIS2,NBENFCOM)

e- Exercice d’application: voir Script_5

 # Exercice d'application
Transformer ANAIS1 et ANAIS2 en integer

mariage1$ANAIS1<-as.integer(mariage1$ANAIS1)
mariage1$ANAIS2<-as.integer(mariage1$ANAIS2)

Calculer l'âge du conjoint 1 (nouvelle variable age1) et du conjoint 2 (nouvelle variable age2) au moment du mariage en 2016
Méthode 1

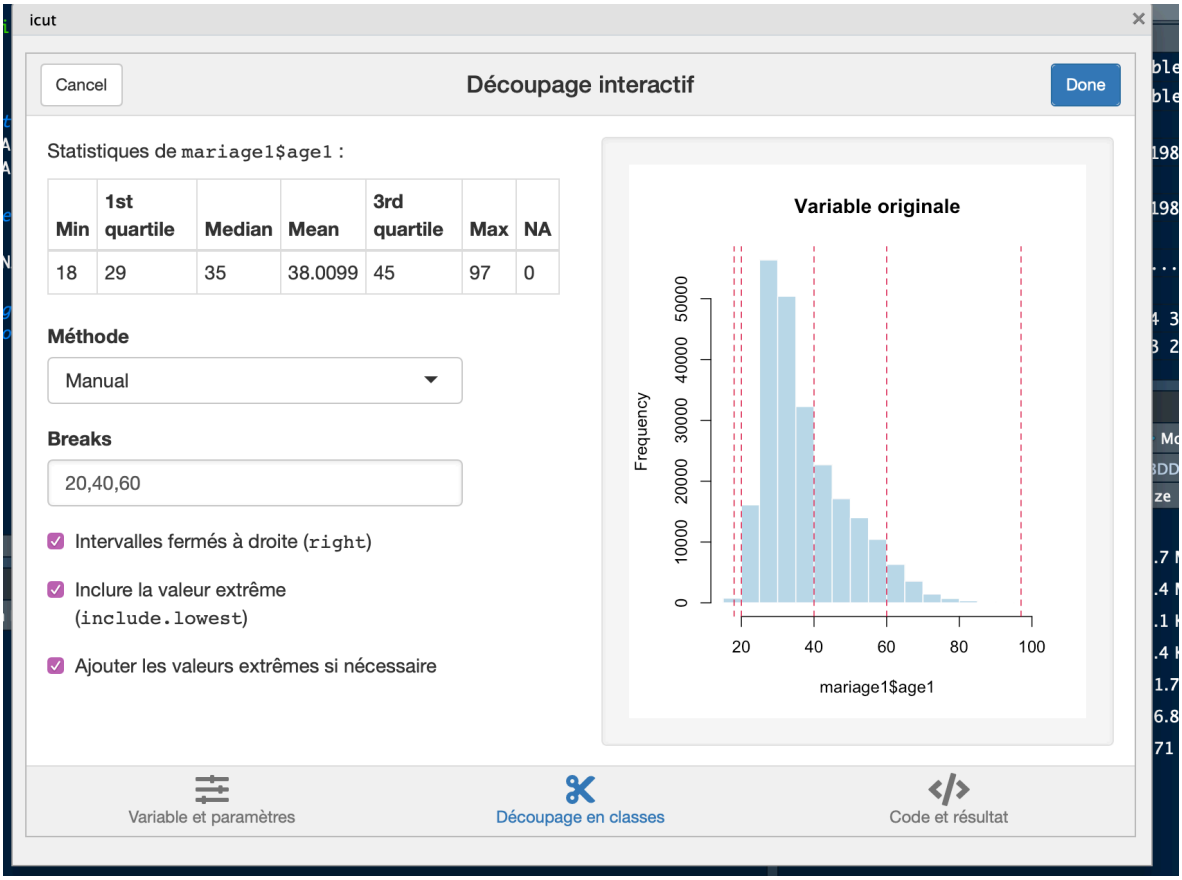
mariage1$age1<-2016-mariage1$ANAIS1
mariage1$age2<-2016-mariage1$ANAIS2

Méthode 2

mariage1<-mutate(mariage1,age1=2016-ANAIS1,age2=2016-ANAIS2)

Discrétiser age1 et age2 avec les groupes d'âge suivants: -20ans, 20-40ans, 40-60ans, 60ans et +
Les nouvelles variables se nommeront age1d et age2d (dans mariage1). Utiliser la console interactive icut()
Recodage de mariage1$age1 en mariage1$age1d

```
mariage1$age1d <- cut(mariage1$age1, include.lowest=TRUE,  right=TRUE,
                breaks=c(18, 20, 40, 60, 97))
mariage1$age2d <- cut(mariage1$age2, include.lowest=TRUE,  right=TRUE,
               breaks=c(15, 20, 40, 60, 99))
```




```
## Changer les labels des modalités en -20, 20-40, 40-60, 60+. Il faut utiliser  irec()
## Recodage de mariage1$age1d en mariage1$age1d_rec
mariage1$age1d <- fct_recode(mariage1$age1d,
                             "-20" = "[18,20]",
                             "20-40" = "(20,40]",
                             "40-60" = "(40,60]",
                             "+60" = "(60,97]")
mariage1$age2d <- fct_recode(mariage1$age2d,
                             "-20" = "[15,20]",
                             "20-40" = "(20,40]",
                             "40-60" = "(40,60]",
                             "+60" = "(60,99]")
```

f - Recodage des variables: voir Script_5

- Regrouper les modalités d'une variable qualitative (type factor):
    1° on peut utiliser irec(). On met le même nom à deux classes différentes.
    2° combinaison: revient à concaténer 2 colonnes (fonction **interaction**)

g- Recodage complexe des variables (fonctions logiques): fonction ifelse() du package dplyr. Script_6

- Fonctions logiques simples avec if else:

Objet$Var<-ifelse(Objet$Var1==Objet$Var2, valeur si vrai, valeur si faux)

- Fonctions logiques imbriquées avec case_when (package dplyr): permet d'imbriquer plusieurs conditions.
case_when (condition1 ~ valeur1, condition2 ~ valeur2, condition n~valeur n, TRUE ~ valeur si tout est faux)