Traitement des données PMSI avec R

Guillaume Pressiat || SIMAP / DOMU / Assistance Publique - Hôpitaux de Paris

2019-01-22

Tables des matières

1

Introduction

Ce livret numérique présente des exemples de traitements de données PMSI avec R. L'objectif est de concentrer ici :

- une documentation permettant de débuter avec l'import de données via le package *pmeasyr*
- des exemples d'analyses PMSI :
  - requêtes sur les diagnostics et les actes
  - analyse des files actives pour une pathologie
  - statistiques élementaires sur des variables du PMSI
  - analyse du case-mix et de la dms par ghm
  - valorisation des rsa

2

Contexte et motivations

Les données du Programme de Médicalisation des Systèmes d'Information (PMSI) sont souvent traitées via des logiciels spécifiques au PMSI (ou des outils statistiques / bases de données du marché) ne permettant pas de réaliser des traitements statistiques et des infographies satisfaisantes. Les départements d'information médicale sont donc souvent amenés à retraiter ces données avec R.

L'évolution récente de R intègre la manipulation de bases de données de taille importante. Le package *pmeasyr* s'inscrit dans cette veine et permet de réaliser de façon autonome l'ensemble des traitements (de l'import des données à leur analyse) avec R.

2.1 Avantages de R

2.1.1 Un flux de travail unique

En travaillant uniquement avec R, on peut mettre en place un flux de travail épuré : un seul projet, un seul programme, un seul logiciel. La traçabilité, la reproductibilité et la mise à jour des opérations sont ainsi facilitées.

Le travail avec de multiples logiciels oblige à l'export / import de fichiers entre les différents logiciels, et chaque modification du début du flux de travail génère des fichiers exportés v1, v2, ...

Avec un flux complet dans R, toute nouvelle modification est intégrée au processus de travail global. La localisation de toutes les étapes d'une analyse en un seul point évite les erreurs et la confusion lorsque l'on reprend l'analyse ultérieurement.

2.1.2 R et le PMSI

L'utilisation de R confère aux données du PMSI la liberté proposée par le logiciel :

- les requêtes sur les diagnostics et les actes peuvent s'écrire de multiples façons et c'est l'utilisateur qui crée ses propres programmes
- les données sont dans R : prêtes pour des modèles linéaires, logistiques, des classifications...
- la confrontation des données in* (reflet du codage des établissements) aux données out* (reflet de la valorisation accordée à l'établissement) est facilitée par l'import du fichier tra, cela peut permettre aux équipes DIM d'améliorer leur recueil
- le reporting de l'activité vers différents formats (Excel, PDF, Word, HTML) ou en créant des applications (shiny)
- l'utilisation des graphiques pour représenter des volumes d'activités et des cartographies interactives pour visualiser la localisation d'activités, de patientèles, et les flux de patients

- le partage de projets RStudio, qui facilite et encourage les travaux en équipe.

NB: Données In / Out : données en entrée / sortie des logiciels de l'ATIH

2.1.3 Des outils performants

L'engouement autour de R est lié au développement de packages intuitifs et performants : *readr*, *dplyr*, *tidyr*, *magrittr*, pour n'en citer que quelques-uns. *pmeasyr* s'appuie sur ces packages pour proposer des imports de données rapides sur des fichiers de taille importante (l'entité juridique de l'AP-HP est prise en charge sans problème avec un ordinateur récent).

Dans le cas de *pmeasyr*, l'import de 100 000 RSA (partie fixe, parsing des passages unités médicales, des diagnostics associés et des actes) nécessite en moyenne 5 secondes avec un processeur i7 – 16Go de ram.

En dernier ressort, R travaillant en mémoire vive, les exécutions de requêtes sont très rapides.

2.2 Contenu du package

Le package contient des fonctions pour la gestion des archives PMSI en entrée / sortie des logiciels de l'ATIH : dézippage, suppression des archives, et des fonctions pour l'import des fichiers des champs PMSI MCO, SSR, HAD, PSY et RSF.

Il est utilisé depuis un an à l'AP-HP pour des analyses d'activité et la description des prises en charge.

2.3 Installation du package

```
devtools::install_github('IM-APHP/pmeasyr')
```

Cette commande lance l'installation du package et de ses dépendances.

3

Les archives PMSI

Cette partie aborde le point de départ des études PMSI : les archives PMSI. Ces archives sont les fichiers en entrées / sorties des logiciels de l'ATIH.

Les manuels techniques de ces logiciels, relatifs aux champs MCO, SSR, HAD, PSY et RSF, respectivement Genrsa, Genrha, Paprica, Pivoine et Preface sont disponibles dans l'espace de téléchargement sur le site de l'ATIH.

3.1 Arborescence des archives

Le package *pmeasyr* prend en charge les données des quatre champs PMSI MCO, SSR, HAD, PSY ainsi que les RSF.

Placer les archives dans un répertoire, par exemple ici dans `~/Documents/data/mco` :

Vous noterez que pour chaque champ PMSI il est conseillé d'utiliser un répertoire indépendant, ceci est nécessaire dans la mesure où le nom des archives PMSI ne contient pas l'information champ MCO, RSF, etc., il faut organiser l'archivage champ par champ, dans des répertoires différents.

```
# Créer l'arborescence à partir de R
champs = c('mco', 'ssr', 'had', 'psy', 'rsf')
emplacement <- "~/Documents/data"
sapply(champs, function(x){dir.create(file.path(emplacement, x))})
```

3.1.1 Sous Unix

Chaque utilisateur dispose de son path '~', qui équivaut par exemple à : '/home/gui/'.

3.1.2 Sous Windows

Chaque utilisateur dispose de son répertoire, exemple `C:/Users/gui/`. Et souvent, sans l'utilisation de projets RStudio, les chemins d'accès aux données sont pénibles à configurer dans chaque programme.

Mais dans R, le symbole '~' utilisé sur windows dans un chemin d'accès aux fichiers renvoie au répertoire Documents de l'utilisateur : `C:/Users/gui/Documents`.

```
path.expand('~')
```

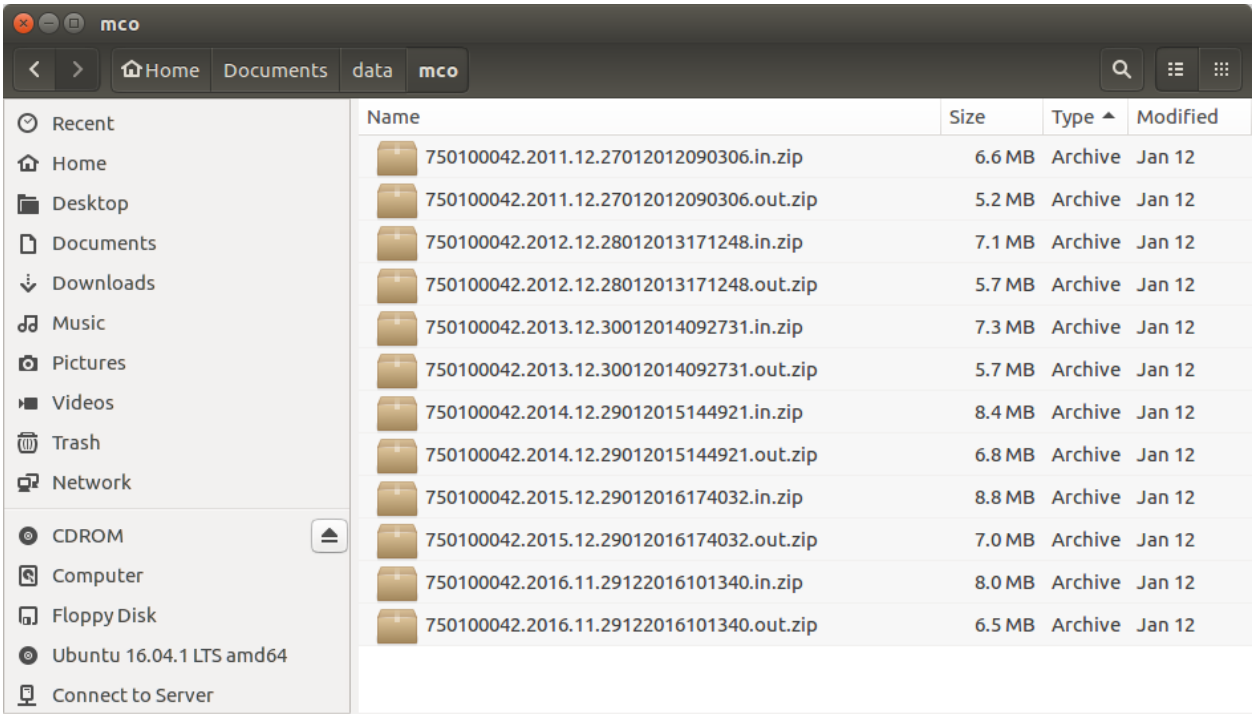

Figure 3.1: Archives MCO

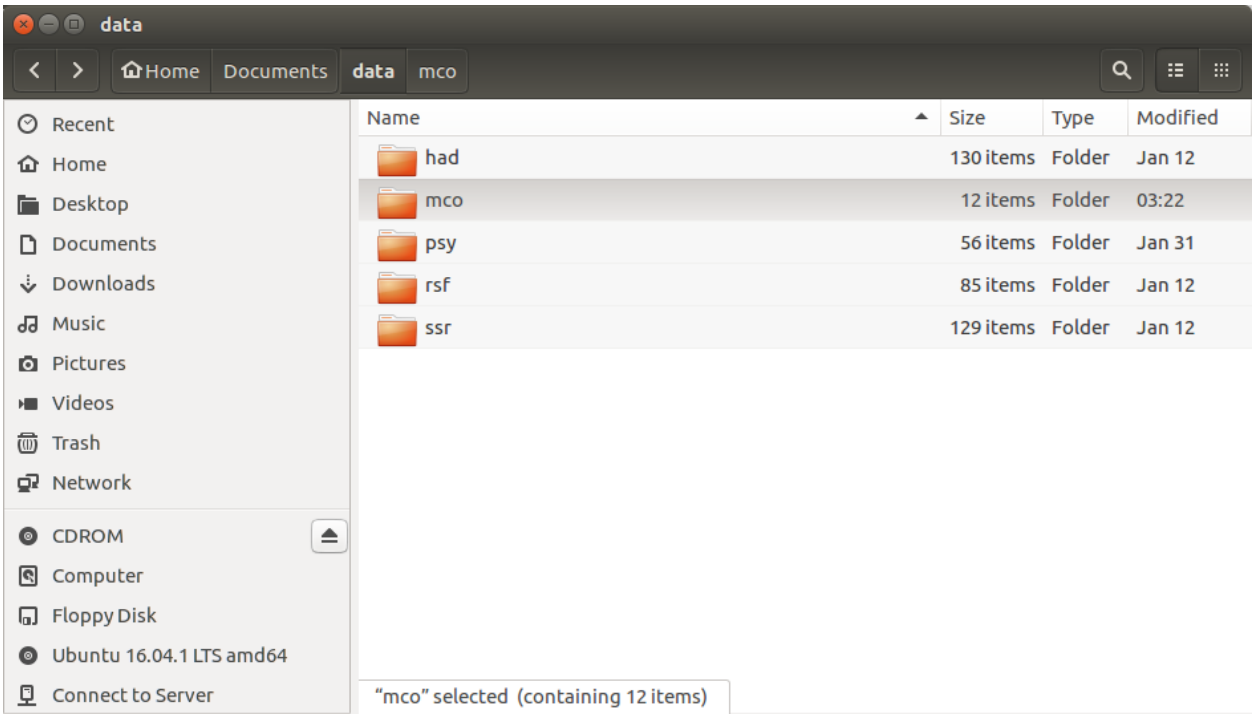

Figure 3.2: Un répertoire par champ PMSI

renvoie `C:/Users/gui/Documents/`.

Par conséquent sur Windows le répertoire à créer pour localiser les fichiers d'archives pmsi sera : `C:/Users/gui/Documents/Documents/data`.

N.B.: Nous proposons ici que les utilisateurs de `pmeasyr` utilisent ce chemin générique pour leurs programmes. L'avantage qui en découle : le partage de programme ne nécessite pas de changer le chemin d'accès, puisque c'est le même.

3.2 Informations sur les archives

Le nom des fonctions dont l'objectif est de manipuler les **a**rchives commence par **a**.

La fonction `astat` permet d'éditer des statistiques sommaires sur les fichiers contenus dans une archive.

Nom	Fonction
astat	~ *.zip - Liste et volume des fichiers d'une archive PMSI

```
# Informations sur les fichiers : Date de creation, Taille
pmeasyr::astat(path = '~/Documents/data/mco/',
               file = '750100042.2015.12.29012016174032.out.zip',
               view = F)
```

3.3 Dézippage

Cette partie du package facilite la manipulation des archives PMSI, fichiers de type :

- `finess.annee.mois.date_et_heure_de_creation.in.zip`
- `finess.annee.mois.date_et_heure_de_creation.out.zip`

Les fonctions permettent de dézipper les fichiers depuis `R` en ligne de commande, sans intervention manuelle de l'utilisateur. L'avantage est d'obtenir un processus ne relevant pas d'interventions externes au logiciel `R` (pour pouvoir garder trace des etapes, et faciliter la reproduction, tout est inscrit dans un programme, dans un flux de processus). Une fois que les traitements et analyses sur les fichiers sont faits, il est possible d'effacer les archives également en ligne de commande.

Nom	Fonction
adezip	~ *.zip - Dezippe des fichiers de l'archive PMSI
adezip2	~ *.zip - Dezippe des fichiers de l'archive PMSI, avec en parametre le nom de l'archive

```
# Dezippage uniquement des fichiers rsa, ano et tra du out 2015
# Ex: 750100042.2015.12.20160130.153012.out.zip
pmeasyr::adezip(finess = 750100042,
                annee = 2015,
                mois = 12,
                path = '~/Documents/data/mco',
                liste = c("rsa", "ano", "tra"),
                type = "out")
```

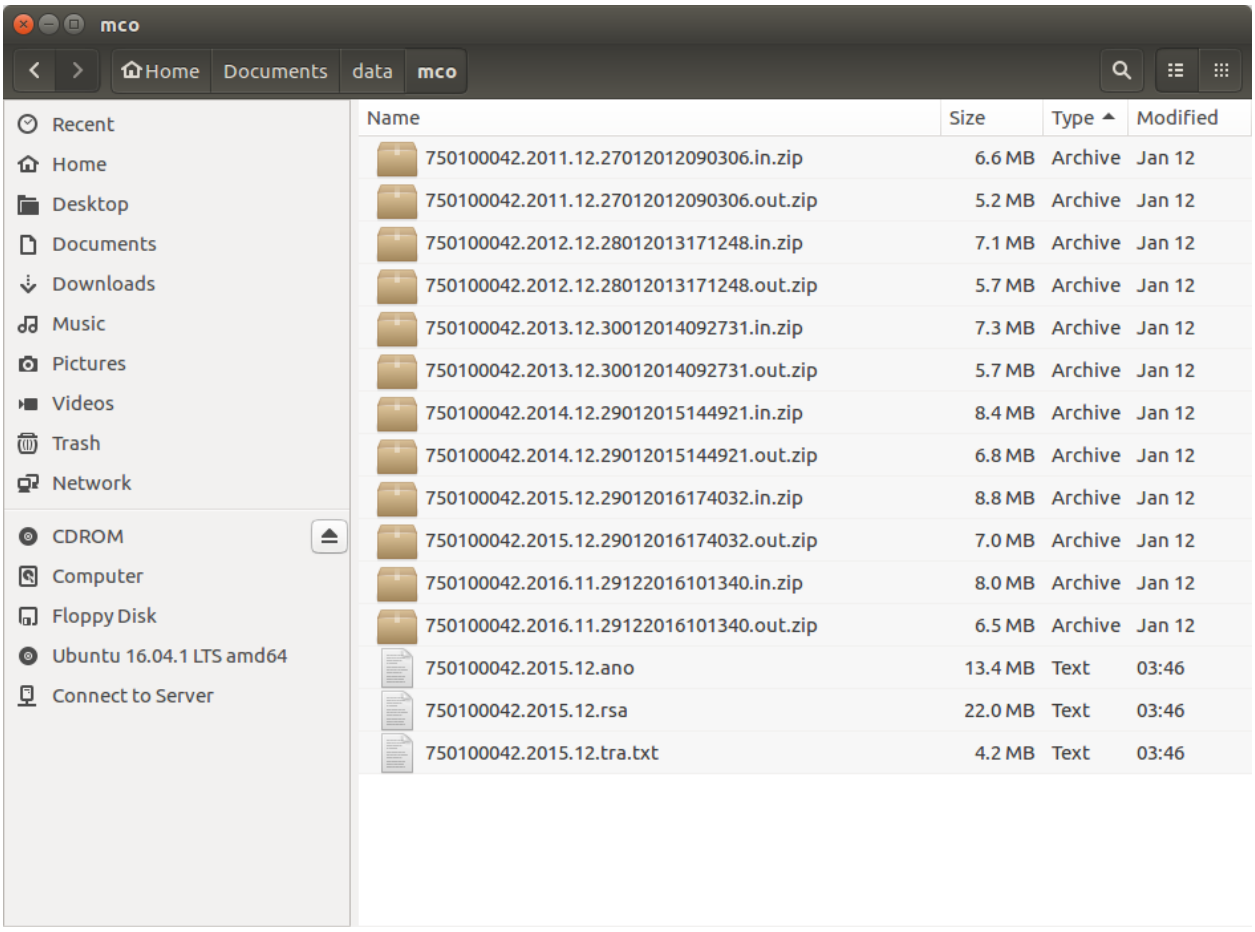

Figure 3.3: Après éxécution de `adezip()` sur des fichiers du *out*

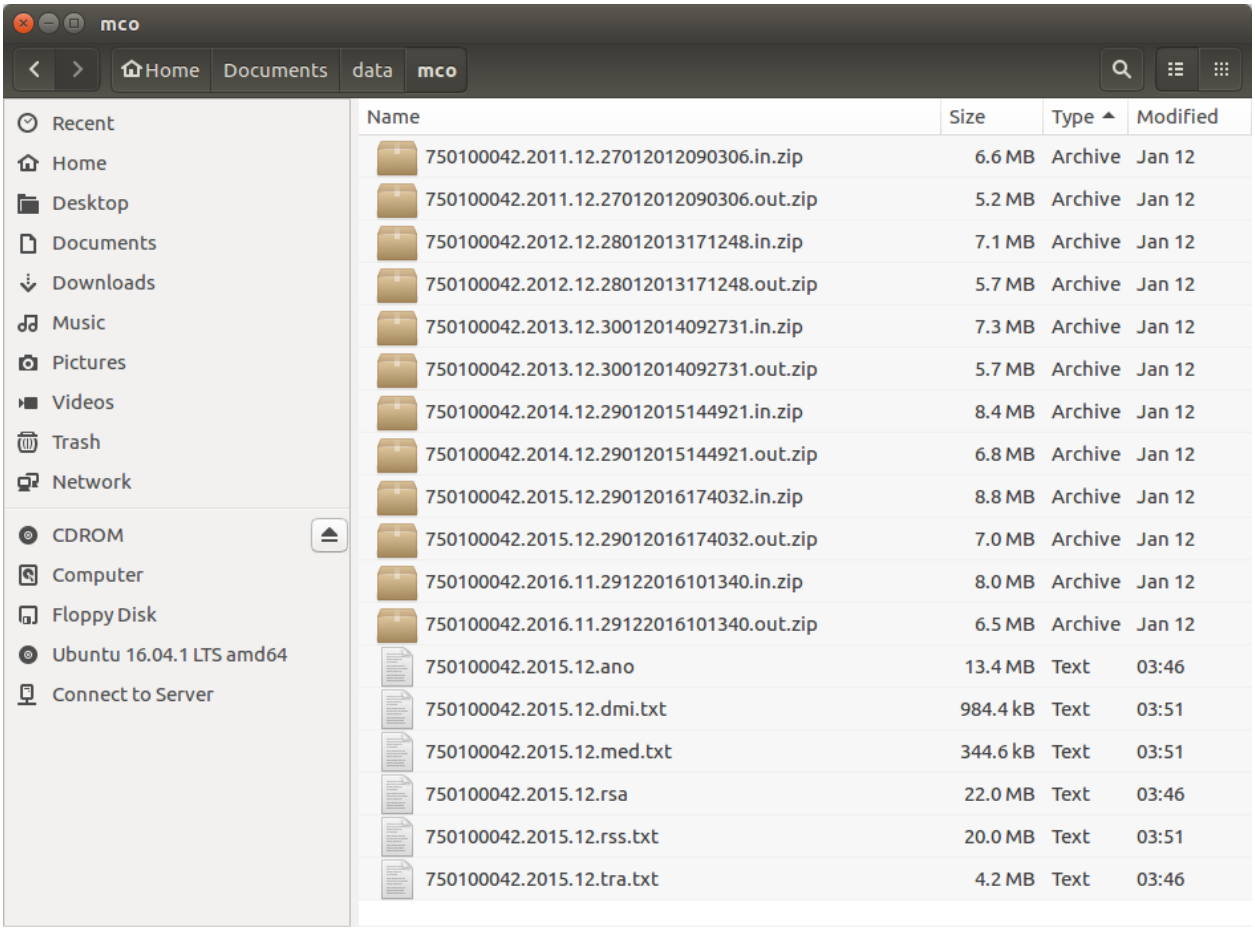

Figure 3.4: Après éxécution de `adezip()` sur des fichiers du *in*

```
# Dezippage uniquement des fichiers rss, dmi et med du in 2015
# Ex: 750100042.2015.12.20160130.153012.out.zip
pmeasyr::adezip(finess = 750100042,
                annee = 2015,
                mois = 12,
                path = '~/Documents/data/mco',
                liste = c("rss", "dmi", "med"),
                type = "in")
```

3.4 Suppression

À la fin d'une étude, il est inutile de garder les fichiers dézippés hors de l'archive, on peut les effacer : c'est ce que permet la fonction `adelete()`.

```
# Effacer les fichiers
pmeasyr::adelete(finess = 750100042,
                 annee = 2015,
                 mois = 12,
                 path = '~/Documents/data/mco',
```

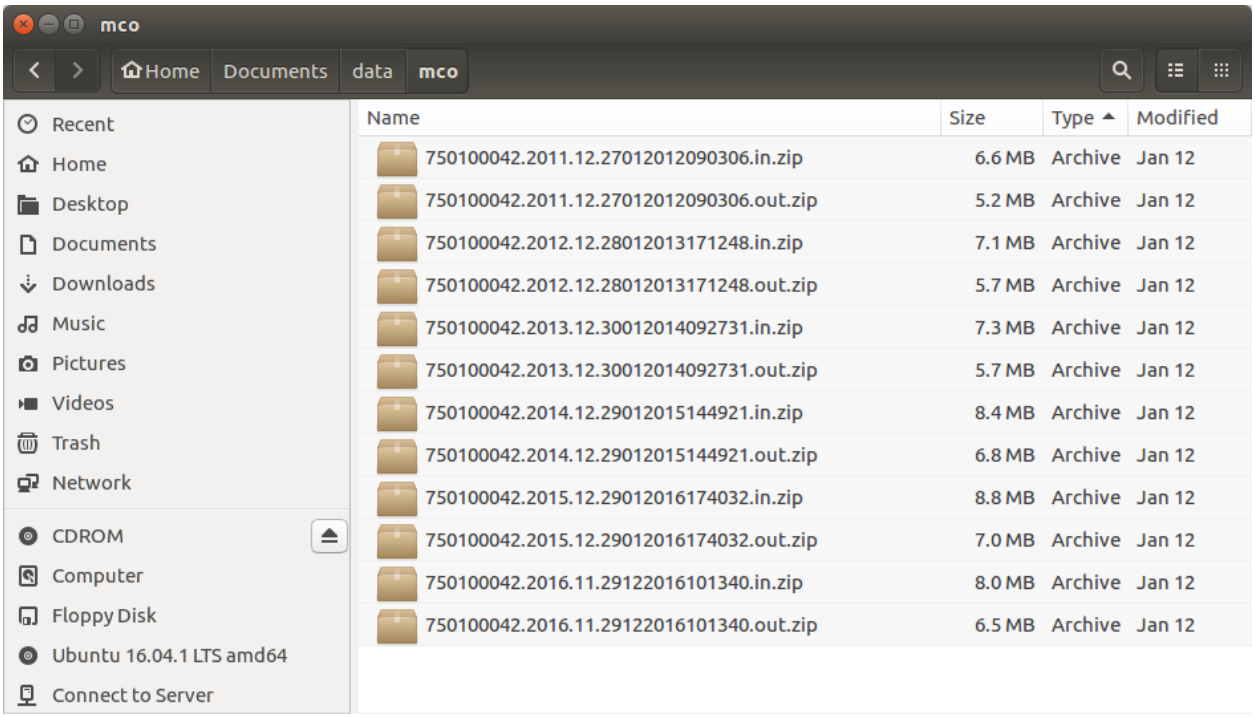

Figure 3.5: Après éxécution de `adelete()`

```
                liste = c("rsa", "ano", "tra"),
                type = "out")

pmeasyr::adelete(finess = 750100042,
                annee = 2015,
                mois = 12,
                path = '~/Documents/data/mco',
                liste = c("rss", "med", "dmi"),
                type = "in")
```

4

Import des données

4.1 MCO

Nom	Fonction
irsa	~ MCO - Import des RSA
irum	~ MCO - Import des RUM
idiap	~ MCO - Import des DIAP
idmi_mco	~ MCO - Import des DMI
iium	~ MCO - Import des donnees UM
ileg_mco	~ MCO - Import des erreurs Leg
imed_mco	~ MCO - Import des Med
ipo	~ MCO - Import des PO
iano_mco	~ MCO - Import des Anohosp

Les données in / out sont prises en charge.

4.1.1 RSA

Selon la nature des analyses à produire, plusieurs types d'imports sont possibles :

Type	Import
1	Light : Partie fixe
2	Light+ : Partie fixe + stream en ligne (+) actes et das
3	Light++ : Partie fixe + stream en ligne (++) actes, das, typaut um et dpdr des um
4	Standard : Partie fixe + creation des tables actes, das et rsa_um
5	Standard+ : Partie fixe + creation des tables actes, das et rsa_um + stream (+)
6	Standard++ : Partie fixe + creation des tables actes, das et rsa_um + stream (++)

```
library(pmeasyr)
# Import des rsa 2015 type 6
irsa(finess = 750100042,
     annee = 2015,
     mois = 12,
```

NOFINESS Finess de l'établissement	NOVRSA Numéro de version du RSA	CLE_RSA Clé RSA	NOVRSS Numéro de version du RSS groupé	NOSEQTA Numéro séquentiel des tarifs
750100042	222	0000000001	116	002
750100042	222	0000000002	116	002
750100042	222	0000000003	116	002
750100042	222	0000000004	116	002
750100042	222	0000000005	116	002
750100042	222	0000000006	116	002
750100042	222	0000000007	116	002
750100042	222	0000000008	116	002
750100042	222	0000000009	116	002
750100042	222	0000000010	116	002
750100042	222	0000000011	116	002
750100042	222	0000000012	116	002
750100042	222	0000000013	116	002
750100042	222	0000000014	116	002
750100042	222	0000000015	116	002
750100042	222	0000000016	116	002
750100042	222	0000000017	116	002

Figure 4.1: Capture d'une portion de la table `rsa15$rsa`

```
     path = '~/Documents/data/mco',
     typi = 6) -> rsa15
View(rsa15$rsa)
View(rsa15$rsa_um)
View(rsa15$actes)
View(rsa15$das)
```

Les tables sont par défaut avec des libellés :

4.1.2 RUM

```
# Import des rum 2015
irum(finess = 750100042,
     annee = 2015,
     mois = 12,
     path = '~/Documents/data/mco')
```

Selon la nature des analyses à produire, plusieurs types d'imports sont possibles :

Type	Import
1	XLight : Partie fixe
2	Light : Partie fixe + stream en ligne des actes, das et dad
3	Standard : Partie fixe + table actes, das, dad
4	Standard+ : Partie fixe + stream + table actes, das, dad

4.1.3 Colonnes stream

Exemples sur quelques rsa :

- actes : Actes CCAM du Rsa

actes Stream Actes	um Parcours Typaut UM	dpdrum Stream DP/DR des UM	das Stream Das
JDLD001, DEQP003	07AC	N179	
DEQP003, DEQP003, DEQP003	07AC	I480	
	07AC	F100	
	07AC	T406	
YYYY600, ACQJ002	07AC	R410	B230, B582
DEQP003	07AC	T509	F603
YYYY009, DEQP007, ZZEP004	04 C	P599	
LFQK002, NEQK010, NAQK023	07AC	M796	
EAQH002, DEQP003, ZBQK001, YYYY600	07AC	J449	Z030, I618
BGQP003, BLQP010, DEQP003, BJQP005, EQQP008, ...	18 C	G459	
	07AC	F119	F209
DEQP003, JAQM003	07AC	N179	
DEQP003	07AC	I509	
DEQP003, NAQK023	07AC	R53+0	F023, G20, I480
DEQP003, LFQK002, LEQK001	07AC	I501	
	07AC	J189	J458
DEQP003, AFHB002, YYYY600, EBQH004	07AC	A879	
DEQP003, DEQP003	07AC	J209	
QAGA003, ZZLP054, ZZLP054, ZZQX180	53 P	M7958	
	29 P	Z502	
EEAF003, EEAF003, EEAF003, EEAF003, YYYY200, N...	29 C	I7021	E1140, E1150, Z713, Z718, L039, L030, Z740, E440...
FCFA008, FCFA008, GDFA011, GDFA011, ZZQX175	53 C	C328	I10, Z921, J384, T814, B956
ZCQM001, JAQM003	29 C	E1120	Z713, N083, E559
NDQK002, DEQP003	29 C	R02	R2630, E1170, Z713

Figure 4.2: Capture des zones *stream* de la table `rsa15$rsa`

- um : types autorisations T2A des um de passage par ordre chronologique

Cle RSA	um
0000000009	01AC, 53 C
0000000010	51 C
0000000011	71 C, 04 C, 71 C

Pour les quatre autres champs PMSI, seules les données du *out* sont prises en charge par le package pour le moment.

Les fonctions d'imports pour ces champs PMSI reposent sur le même principe qu'en MCO.

4.2 HAD

Nom	Fonction
iano_had	~ HAD - Import des Anohosp
imed_had	~ HAD - Import des Med
irapss	~ HAD - Import des RAPSS
ileg_had	~ HAD - Import des erreurs LEG

```
library(pmeasyr)
# Import des rapss 2015
irapss(finess = 750712184,
       annee = 2015,
       mois = 12,
       path = '~/Documents/data/had') -> data_had
```

4.4 PSY

Nom	Fonction
iano_psy	~ PSY - Import des Anohosp
ir3a	~ PSY - Import des R3A
irpsa	~ PSY - Import des RPSA

```
# Import des rpsa 2015
irpsa(finess = 750803454,
      annee = 2015,
      mois = 12,
      path = '~/Documents/data/psy') -> rpsa_psy

# Import des r3a 2015
ir3a(finess = 750803454,
      annee = 2015,
      mois = 12,
      path = '~/Documents/data/psy') -> r3a_psy
```

4.5 RSF

Nom	Fonction
irafael	~ RSF - Import des RSFA / Rafael
iano_rafael	~ RSF - Import des RSFA / ANO

```
# Import des rsfa 2015
irafael(finess = 750712184,
        annee = 2015,
        mois = 12,
        path = '~/Documents/data/rsf') -> rsfa
```

4.6 Dictionnaire de variables

```
# Obtenir les noms, labels et types de variables (character, numeric, integer, date, ...)
dico(rsa15$rsa)
```

```
# Charger les formats de toutes les tables prises en charge par le package
pmeasyr::formats
```

4.7 Labels

```r
# Obtenir le libelle d'une variable du PMSI
labeleasier(rsa15$rsa$SEXE, Sexe = T)
labeleasier(rsa15$rsa$ECHPMSI, Mode_entree = T)
```

5

Requêtes sur des pathologies / actes

5.1 Transposition des codes diagnostics

Les analyses sur les diagnostics CIM-10 sont parfois fastidieuses du fait des multiples positions de diagnostics : DP principal du séjour, DR principal du séjour, DPUM, DRUM, DAS. La fonction *tdiag* permet de rassembler tous les diagnostics dans une seule table.

```
# Pour les objets rsa et rum du MCO
# Transbahuter tous les diagnostics dans une seule table
tdiag(rsa15) -> rsa15 # "Tidy diagnostics"
View(rsa15$diags)
# Tous les diagnostics sont dans une table, avec un numero selon leur position
# 1:DP, 2:DR, 3:DPUM, 4:DRUM, 5:DAS
```

Exemple de résultat :

CLE_RSA	NSEQRUM	position	diag
0000000001	01	1	Z511
0000000001	01	2	C18
0000000002	01	1	C501
0000000002	01	3	C501
0000000002	02	1	D051
0000000002	02	5	E109

5.2 Recherche de codes diagnostics

L'objectif est de récupérer les séjours présentant un code diagnostic de la liste

```
# Liste D-0103 de la fonction groupage 2016 : Epilepsies
liste_diag = c('F803', 'G400', 'G401', 'G402', 'G403', 'G404',
               'G405', 'G406', 'G407', 'G408', 'G409', 'G410',
               'G411', 'G412', 'G418', 'G419', 'R568')

# En passant par la table diags
tdiag(rsa15) -> rsa15
```

```r
library(dplyr)
# quelle que soit la position du diagnostic
rsa15$diags %>% filter(diag %in% liste_diag)
# position en das
rsa15$diags %>% filter(diag %in% liste_diag, position == 5)
# position en dp dr
rsa15$diags %>% filter(diag %in% liste_diag, position < 5)

# En passant par les zones stream
string_diags =
  'F803|G400|G401|G402|G403|G404|G405|G406|G407|G408|G409|G410|G411|G412|G418|G419|R568'

# quelle que soit la position du diagnostic
rsa15$rsa %>% filter(grepl(string_diags, dpdrum)|grepl(string_diags, das))
# position en das
rsa15$rsa %>% filter(grepl(string_diags, das))
# position en dpdr
rsa15$rsa %>% filter(grepl(string_diags, dpdrum))
```

5.3 Recherche de codes actes

```r
# Code EBLA003

library(dplyr)
# En passant par la table actes
rsa15$actes %>% filter(CDCCAM == 'EBLA003')
# En passant par la zone stream
rsa15$rsa %>% filter(grepl('EBLA003', actes))
```

6

Étude des files actives

6.1 Import des données Anohosp

```
# Import des données anohosp du out
iano_mco(finess = 750100042,
         annee = 2015,
         mois = 12,
         path = '~/Documents/data/mco') -> ano

# Filtrer sur les patients chainables avec la variable cok
library(dplyr)
ano %>% filter(cok) -> ano

# File active globale établissement
distinct(ano, NOANON) %>% nrow()
```

6.2 File active d'une pathologie

```
# Codes diagnostics obésité
string_diags = 'E66'

library(dplyr)
# position en dpdr
rsa15$rsa %>% filter(grepl(string_diags, dpdrum)) -> ob

# File active obésité globale établissement
inner_join(ano, ob, by = c('CLE_RSA')) -> patients_ob
distinct(patients_ob, NOANON) %>% nrow()
```

6.3 File active d'une chirurgie

```
# Codes actes chirurgie bariatrique
liste_actes = c('HFCA001', 'HFCC003', 'HFFC004', 'HFFA001', 'HFMA009',
```

```
                'HFMC007', 'HFKA001', 'HFKC001', 'HFKA002', 'HFMA011',
                'HFMC008', 'HFMA010', 'HFMC006', 'HFLE002', 'HFGC900',
                'HFLC900', 'HFFA011', 'HFFC018', 'HGCA009', 'HGCC027')

library(dplyr)
# acte codé activité 1 (chirurgical)
rsa15$actes %>% filter(CDCCAM %in% liste_actes, ACT == '1') -> cob

# File active chirurgie bariatrique globale établissement
inner_join(ano, cob, by = c('CLE_RSA')) -> patients_cob
distinct(patients_cob, NOANON) %>% nrow()
```

7

Fichier TRA

Le fichier TRA est un fichier du *out* qui permet de relier les données anonymes du *out* aux données du *in*, il comprend un lien entre :

- MCO : clé rsa, numéro de rss, numéro de sejour (nas), date d'entrée et date de sortie du séjour
- SSR : numéro séquentiel du séjour + noseqrhs et numéro de séjour + numéro de semaine
- PSY RPSA : ipp, date d'entrée et de fin du sejour, numéro séquentiel du séjour, numéro de séquence et numéro de séjour, dates de début et fin de sequence
- PSY R3A : ipp, date de l'acte, numéro d'ordre, forme activité, um, nature et lieu de l'acte
- HAD : numéro séquentiel de séjour, numéro de séquence, sous-sequence et numéro de séjour, dates de début et fin des séquences et sous-séquences, dates d'entrée et de sortie du séjour, modes d'entrée sortie provenance destination

Type	Import
itra	~ TRA - Import du TRA
inner_tra	~ TRA - Ajout du TRA aux données Out

7.1 Ajout du TRA en MCO

```
# lecture du fichier tra et jointure aux rsa
itra(750100042, 2015, 12, '~/Documents/data/mco') -> tra

# Ajout du tra aux rsa :
inner_tra(rsa15$rsa, tra) -> rsa15$rsa
# Ajout du tra à la partie um des rsa :
inner_tra(rsa15$rsa_um, tra) -> rsa15$rsa_um
# Ajout du tra à la partie actes des rsa :
inner_tra(rsa15$actes, tra) -> rsa15$actes
# Ajout du tra à la partie das des rsa :
inner_tra(rsa15$das, tra) -> rsa15$das
```

7.2 Ajout du TRA en HAD

```
# Import du TRA HAD
itra(finess = 750712184,
     annee = 2015,
     mois = 12,
     path = '~/Documents/data/had',
     champ = "had") -> tra

# Ajout du tra
inner_tra(data_had$rapss, tra, champ = "had") -> data_had$rapss
inner_tra(data_had$acdi,  tra, champ = "had") -> data_had$acdi
inner_tra(data_had$ght,   tra, champ = "had") -> data_had$ght
```

7.3 Ajout du TRA en SSR

```
# Import du TRA SSR
itra(finess = 750712184,
     annee = 2015,
     mois = 12,
     path = '~/Documents/data/ssr',
     champ = "ssr") -> tra

# Ajout du tra
inner_tra(data_ssr$rha,  tra, champ = "ssr") -> data_ssr$rha
inner_tra(data_ssr$acdi, tra, champ = "ssr") -> data_ssr$acdi
```

7.4 Ajout du TRA en PSY

```
# Import du TRA PSY : fichiers RPSA
itra(finess = 750803454,
     annee = 2015,
     mois = 12,
     path = '~/Documents/data/psy',
     champ = "tra_psy_rpsa") -> tra

# Ajout du tra
inner_tra(rpsa_psy$rpsa, tra, champ = "psyrpsa") -> rpsa_psy$rpsa
inner_tra(rpsa_psy$das,  tra, champ = "psyrpsa") -> rpsa_psy$das
```

```
# Import du TRA PSY : fichiers R3A
itra(finess = 750803454,
     annee = 2015,
     mois = 12,
     path = '~/Documents/data/psy',
     champ = "tra_psy_r3a") -> tra

# Ajout du tra
```

```
inner_tra(r3a_psy$r3a, tra, champ = "psyr3a") -> r3a_psy$r3a
inner_tra(r3a_psy$das, tra, champ = "psyr3a") -> r3a_psy$das
```

8

Statistiques du PMSI

8.1 Âge et durée de séjour

```
library(dplyr)

# Age moyen et DMS sur les plus de 0 jour
rsa15$rsa %>%
  summarise(age_moyen = mean(AGEAN, na.rm = T),
            dms = mean(DUREE[DUREE > 0]),
            effectif = n(),
            effectif_sup0 = sum(DUREE > 0))

# Age moyen en prenant en compte les séjours des bébés
# (variable age en jour)
rsa15$rsa %>%
  mutate(Age = if_else(is.na(AGEAN), as.integer(AGEJR) / 365.25, as.numeric(AGEAN))) %>%
  summarise(age_moyen = mean(Age),
            effectif = n())
```

8.2 Nombre de séjours par catégorie majeure de diagnostics

```
# Nombre de séjours par catégorie majeure de diagnostics
rsa15$rsa %>% count(RSACMD)
```

8.3 Case-mix MCO, DMS par GHM / GHS

```
# Construire la variable GHM
rsa15$rsa %>% tidyr::unite(GHM,
                           RSACMD, RSATYPE, RSANUM, RSACOMPX,
                           sep = "") -> rsa15$rsa

# Case-mix par GHM
rsa15$rsa %>% count(GHM)
```

```r
# Case-mix par GHM / GHS
rsa15$rsa %>% count(GHM, NOGHS)

# DMS par GHM / GHS
rsa15$rsa %>% group_by(GHM, NOGHS) %>%
  summarise(dms = mean(DUREE[DUREE > 0]),
            effectif = n(),
            effectif_sup0 = sum(DUREE > 0))
```

9

Help

Toutes les fonctions du package ont une page d'aide :

```
# Exemple : aide sur la fonction d'import des rsa
?irsa
```

L'aide en ligne est disponible ici.

10

Noyau de paramètres

10.1 Quatre paramètres

Les fonctions de dézippage et d'import de pmeasyr ont en commun quatre paramètres :

- le finess du fichier
- l'année pmsi
- le mois pmsi
- le path, repértoire où se trouve le fichier

Ces paramètres sont toujours nécessaires : définir un noyau de paramètres permet de ne les préciser une seule fois, allège le code pour plus de lisibilité.

Exemple de redondance des paramètres :

```
# Exemple
library(pmeasyr)

# Dézipper
adezip(finess = '750712184',
       annee  = 2015,
       mois   = 12,
       path   = '~/Documents/data/mco/',
       type   = "out",
       liste  = c("rsa", "ano", "tra"))

# Table ano
iano_mco(finess = '750712184',
         annee  = 2015,
         mois   = 12,
         path   = '~/Documents/data/mco/') -> ano

# Tables rsa
irsa(finess = '750712184',
     annee  = 2015,
     mois   = 12,
     path   = '~/Documents/data/mco/',
     typi   = 4) -> rsa

# Table tra
```

```
itra(finess = '750712184',
     annee  = 2015,
     mois   = 12,
     path   = '~/Documents/data/mco/') -> tra
```

10.2 Solution avec `noyau_pmeasyr()`

10.2.1 Définir le noyau de paramètres

```
library(pmeasyr)
p <- noyau_pmeasyr(finess = '750712184',
                   annee = 2015,
                   mois = 12,
                   path   = '~/Documents/data/mco/')
```

10.2.2 Utiliser le noyau de paramètres

```
# Tout dézipper
# out
p %>% adezip(type = "out")
# in
p %>% adezip(type = "in")
```

```
p %>% irsa()     -> rsa
p %>% iano_mco() -> ano_out
p %>% itra()     -> tra
```

La syntaxe est beaucoup plus lisible désormais.

10.3 `noyau_skeleton()`

La fonction `noyau_skeleton()` permet d'obtenir dans la console R le squelette pour définir les quatre paramètres d'un noyau `pmeasyr`.

```
library(pmeasyr)
noyau_skeleton()
```

```
##
## noyau_pmeasyr(
##   finess = '.........',
##   annee  = ....,
##   mois   = ..,
##   path   = ''
## ) -> p
```

11

Restructurer des listes de codes : enrobeur

La fonction `enrobeur()` permet de structurer une liste de diagnostics, de ghm ou de codes CCAM sous différents formats, selon le type de requêtes que l'on souhaite effectuer ensuite.

11.0.1 Pour du SQL %like% %..% (left & right)

```
library(magrittr)
li <- c('QEFA003', 'QEFA005', 'QEFA010', 'QEFA013', 'QEFA015', 'QEFA019', 'QEFA020')

enrobeur(li, robe="\'%", interstice="\n", symetrique = T) %>% cat()
```

```
## '%QEFA003%'
## '%QEFA005%'
## '%QEFA010%'
## '%QEFA013%'
## '%QEFA015%'
## '%QEFA019%'
## '%QEFA020%'
```

11.0.2 SQL %like% ..%

à venir

11.0.3 SQL %like% %..

à venir

11.0.4 Pour du grepl

```
li <- c('QEFA003', 'QEFA005', 'QEFA010', 'QEFA013', 'QEFA015', 'QEFA019', 'QEFA020')

enrobeur(li, robe="", interstice="|") %>% cat()
```

```
## QEFA003|QEFA005|QEFA010|QEFA013|QEFA015|QEFA019|QEFA020
```

11.0.5 Pour du listing

```
li <- c('QEFA003', 'QEFA005', 'QEFA010', 'QEFA013', 'QEFA015', 'QEFA019', 'QEFA020')

enrobeur(li, robe="\'", interstice=", ") %>% cat()
```

```
## 'QEFA003', 'QEFA005', 'QEFA010', 'QEFA013', 'QEFA015', 'QEFA019', 'QEFA020'
```

12

Noms de colonnes en minuscules

Dans chaque fonction, un nouveau paramètre est disponible : `tolower_names`. Il permet à l'utlisateur de choisir entre minuscules et majuscules pour les noms des variables dans les tables.

Par homogénéité avec le format jusqu'alors, la valeur par défaut de ce paramètre reste `FALSE`, les noms de variables restent donc en majuscule sans modifier ce paramètre.

N.B. : Pour utiliser les fonctions `requete`, `requete_db`, `lancer_requete` et `lancer_requete_db`, il est nécessaire d'avoir choisi un import avec noms de variables en minuscule : `tolower_names = TRUE`.

13

Vers la base de données avec pmeasyr

Constatant que chaque étude avec pmeasyr commence par un import de données, nous proposons une nouvelle approche : intégrer une fois pour toutes les données M12 dans une base de données managée et accessible avec pmeasyr. Les données en cours d'année pouvant également y être intégrées.

Nous décrivons ici l'import de données MCO 2016 dans une base de données tel qu'il est possible de le réaliser à l'aide de pmeasyr.

13.1 Connexion à une base de données

13.1.1 monetdb

```
dbdir <- "~/Documents/data/monetdb"
con <- MonetDBLite::src_monetdblite(dbdir)
```

13.1.2 SQLite

```
dbdir <- "~/Documents/data/sqlite/pmsi.sqlite"
con <- dplyr::src_sqlite(dbdir)
```

13.1.3 PostGres

```
con <- dplyr::src_postgres(user = "gui", password = "gui", dbname = "aphp",
                   host = "localhost", port = 5432)
```

13.2 Intégration en base

13.2.1 Définition des paramètres

```
# noyau_skeleton()
p <- noyau_pmeasyr(
  finess   = '750712184',
  mois     = 12,
  path     = '~/Documents/data/mco',
  progress = F,
  tolower_names = T, # choix de noms de colonnes minuscules : T / F
  n_max = 1e4, # on limite la lecture a un petit nombre de lignes pour tester d'abord
  # on importera tout dans un second temps
  lib = F) # pas de libellés de colonnes (inutile ici)
```

13.2.2 Exemple d'import

```
purrr::quietly(db_mco_out)(con,  p, annee = 2016) -> ok
```

`ok` contient toutes les sorties de l'éxécution de la fonction : vérifier qu'il n'y a pas d'erreur ou d'avertissement anormal avant de poursuivre. Ok ? Voyons ce que contient `ok` :

```
ok
```

Nous avons présenté la ligne pour importer les données MCO out 2016, mais il est possible de la faire pour le MCO in (`db_mco_in`), le SSR out (`db_ssr_out`), l'HAD (`db_had_out`), la Psy (`db_psy_out`) et les rsf (`db_rsf_out`).

Une fonction pour importer une autre table est également disponible (`db_generique`).

Après plusieurs imports dans la base de données, se déconnecter :

```
DBI::dbDisconnect(con$con)
```

13.3 Accéder aux données dans la base

Se reconnecter à la base en redéfinissant l'objet `con`.

Poursuivons l'exemple sur le mco out 2016.

13.3.1 Lister les tables dans la base

```
db_liste_tables(con)
```

13.3.2 Accéder aux tables mco out

Les lignes suivantes permettent de se connecter aux tables dans la base. Pour la suite il pourra être intéressant de lire l'aide du package dplyr.

```
tbl_mco(con, 16, 'rsa_rsa') # Table rsa
tbl_mco(con, 16, 'rsa_actes') # Table actes des rsa
tbl_mco(con, 16, 'rsa_diags') # Table diags des rsa (après `tdiag`)
tbl_mco(con, 16, 'rsa_um') # Table um des rsa
```

D'autres fonctions `tbl_ssr`, `tbl_had`, `tbl_had`, `tbl_psy`, `tbl_rsf` facilitent la connexion à ces "remote table".

14

requetr : rédiger, exécuter des requêtes pmsi

Cette partie de `pmeasyr` concerne la rédaction et l'exécution de requêtes pmsi. Quelques exemples d'abord puis une présentation de ce qu'il est possible de faire avec (et, de manière complémentaire, de ce qu'il n'est pas possible de faire).

14.1 Définir une requête

14.1.1 Exemple simple

```
requete_simple <- list(
  actes = "EPLF002",
  activites_actes = "1"
)
```

Ici on cherchera tous les séjours avec un acte EPLF002 codé en activité 1.

14.1.2 Exemple plus complexe

```
requete_complexe = list(diags = "E66",
                        positions_diags = 5,
                        dureemax = 0,
                        ghm = '28Z',
                        agemax = 18,
                        diags_exclus = "C")

# positions_diags : 1 dp rsa, 2 dr rsa, 3 dp um, 4 dr um, 5 das
```

Ici on cherchera tous les séjours avec un diagnostic E66 en das, d'une durée de 0 jour, avec un ghm en 28Z, où l'âge du patient est au maximum 18 ans, et sans aucun diagnostic commençant par C.

14.2 Requêtes sur des rsa importés avec `irsa`

Il faut d'abord importer les rsa avec un type d'import 6 et "préparer les rsa" avec la fonction `prepare_rsa()` :

```
irsa(p, typi = 6) -> rsa
prepare_rsa(rsa) -> rsa
```

Exemple de requêtes :

```
exemple_requete <- list(
  actes = c('QEFA003','QEFA005','QEFA010','QEFA013','QEFA015','QEFA019','QEFA020')
)
```

La commande suivante renvoie un `tibble` (tableau) contenant les clés rsa correspondants à cette requête :

```
requete(rsa, exemple_requete)
```

Il est possible de retourner d'autres variables que la clé rsa uniquement, en les spécifiant :

```
requete(rsa, exemple_requete, vars = c('nas', 'agean', 'actes', 'diags', 'ghm'))
```

14.3 Requêtes sur des rsa dans la base de données

C'est le même principe seulement il faut spécifier la connexion à la table et l'année de la requête plutôt que l'objet rsa :

```
requete_db(con, 16, exemple_requete, vars = c('nas', 'agean', 'actes', 'diags', 'ghm'))
```

14.4 Éléments possibles pour une requête

Élément	Commentaire
agemin	Âge minimum en années (inclus)
agejrmin	Âge minimum en jours (inclus)
agejrmax	Âge maximum en jours (inclus)
agemax	Âge maximum en années (inclus)
dureemin	Durée minimum en jours (inclus)
dureemax	Durée maximum en jours (inclus)
ghm	Liste de ghm ou racine de ghm : exemple : 08C24, 08C241
diags_exclus	Diagnostics à exclure en toutes positions
poidsmin	Poids minimum (inclus)
poidsmax	Poids maximum (inclus
ghm_exclus	ghm à exclure en toutes positions
autres	Autres éléments de requête au niveau rsa partie fixe, exemple 'agegest $<$ 37'
diags	Diagnostic ou liste de diagnostics à rechercher
positions_diags	Positions des diagnostics à chercher, voir les positions de la table diags (de 1 à 5)
actes	Acte ou liste d'actes à rechercher
activites_actes	Activité pour laquelle chercher les actes

14.5 Lancer plusieurs requêtes

Les fonctions `lancer_requete()` et `lancer_requete_db()` permettent de lancer une ou plusieurs requêtes lorsque l'on a défini une liste de requêtes.

14.5.1 Exemple de liste de requêtes

```
deux_requetes <- list(requete_1 = list(actes = "EBLA003", nom = "pac", thematique = "pac"),
                      requete_2 = list(ghm = "28Z", nom = "seances", thematique = "CMD 28")
)
```

14.5.2 Lancement

```
# Pour des rsa importés avec irsa
lancer_requete(rsa, deux_requetes, vars = c('nas', 'duree', 'ghm'))

# Pour des rsa dans une base de données
lancer_requete_db(con, 16, deux_requetes, vars = c('nas', 'duree', 'ghm'))
```

La commande suivante renvoie un `tibble` (tableau) contenant les clés rsa correspondants à ces requêtes avec deux variables identifiant le nom et la thématique de la requête correspondante.

15

vvr : Valoriser les rsa

Valorisation, cf wikipédia

- *processus de détermination de la valeur d'un objet, d'un actif, d'une entité. L'objectif est d'établir un prix.*
- *processus visant à améliorer la valeur de cet objet, actif, entité*

Cette partie de `pmeasyr` permet de constituer des tables contenant les informations de valorisation de chaque rsa, en accord avec la valorisation epmsi, donc avec la valorisation VisualValoSej (VVS, d'où le choix du nom vvr).

Trois étapes sont distinguées pour reproduire la valorisation d'un rsa :

- disposer des référentiels de tarifs
- une étape autour du séjour : la valorisation GHS + suppléments, etc
- une étape autour de la facture : le caractère facturable ou non du séjour est déterminé.

Pour ces deux dernières étapes, la logique retenue a été la suivante : deux fonctions permettent de constituer une table séjour (rsa) et une table facture (ano) contenant les variables utiles.

15.1 Accéder aux tarifs depuis R

Le package intitulé nomensland contient un ensemble de référentiels utiles dans le PMSI, parmi lesquels les tarifs GHS et suppléments du MCO pour les établissements publics.

Il est nécessaire d'installer ce package ou de disposer de tables dont la structure et les noms de colonnes sont identiques à celles de ce package pour pouvoir continuer.

```
tarifs      <- nomensland::get_table('tarifs_mco_ghs') %>% distinct(ghs, anseqta, .keep_all = TRUE)
supplements <- nomensland::get_table('tarifs_mco_supplements') %>% mutate_if(is.numeric, tidyr::replace_
```

15.2 Étape séjour

Il y a plusieurs possibilités pour cette étape : valoriser uniquement les GHS + bornes extrêmes, ou bien valoriser l'intégralité des rubriques de valorisation.

Mais il faut d'abord préparer la table rsa spécifique à la valorisation.

15.2.1 Préparation des rsa

```
# avec un noyau pmeasyr (importer les données)
annee <- 18
p <- noyau_pmeasyr(
  finess   = '750712184',
  annee    = 2000 + annee,
  mois     = 6,
  path     = '~/Documents/data/mco',
  progress = FALSE,
  n_max    = Inf,
  lib      = FALSE,
  tolower_names = TRUE)

vrsa <- vvr_rsa(p)

# depuis une base de données (collecter les données)
dbdir <- "~/Documents/data/monetdb"
con <- src_monetdblite(dbdir)

vrsa <- vvr_rsa(con, annee)
```

15.2.2 Calculer la valorisation GHS borne extrême haut + extrême bas

```
# Recette GHS de base et suppléments EXB, EXH
rsa_v <- vvr_ghs_supp(rsa = vrsa, tarifs = tarifs)
```

15.2.3 Calculer la valorisation GHS borne extrême haut + extrême bas et suppléments

```
# Recette GHS de base, EXB, EXH, et suppléments
rsa_v <- vvr_ghs_supp(vrsa, tarifs, supplements, vano, ipo(p), idiap(p), pie = ipie(p), bee = FALSE)
```

15.3 Étape facture

15.3.1 Préparation des ano

```
## Not run:
# avec un noyau pmeasyr (importer les données)
p <- noyau_pmeasyr(
  finess   = '750712184',
  annee    = 2000 + 18,
  mois     = 4,
  path     = '~/Documents/data/mco',
  progress = FALSE,
  n_max    = Inf,
  lib      = FALSE,
  tolower_names = TRUE)
```

```
vano <- vvr_ano_mco(p)

# depuis une base de données (collecter les données)
dbdir <- "~/Documents/data/monetdb"
con <- src_monetdblite(dbdir)

vano <- vvr_ano_mco(con, 18)
```

15.3.2 Attribuer le caractère facturable du séjour tel que sur epmsi (tableau sv)

```
## Not run:
# Tenir compte des porg
ano_sv <- vvr_mco_sv(vrsa, vano, porg = ipo(p))

# ne pas tenir compte des porg
ano_sv <- vvr_mco_sv(vrsa, vano)

## End(Not run)
```

15.4 Conjuguer la valorisation des séjours à leur caractère facturable

Une fois que les éléments sont prêts, on peut les rassembler :

```
valo_rsa <- vvr_mco(rsa_v, ano_sv)
```

Ou tout calculer en un seul appel à ces différentes fonctions :

```
resu <- vvr_mco(
vvr_ghs_supp(rsa = vrsa,
             tarifs = tarifs_ghs,
             supplements =  tarifs_supp,
             ano = vano,
             porg = ipo(p),
             diap = idiap(p),
             pie = ipie(p),
             full = FALSE,
             cgeo = 1.07,
             prudent = NULL,
             bee = FALSE),
vvr_mco_sv(vrsa, vano, ipo(p))
)
```

15.5 Concordance avec epmsi

Deux fonctions permettent de produire des tableaux importants sur epmsi : cela permet de s'assurer que le résultat est ok.

- Tableau [1.V.1.SV]: Séjours valorisés
- Tableau [1.V.1.RAV]: Récapitulation activité - valorisation & Tableau [1.V.1.RAE]: Récapitulation activité - effectifs

```
epmsi_mco_sv(valo_rsa, knit = TRUE)
```

```
epmsi_mco_rav(valo, knit = TRUE)
```

Ce dernier tableau contient à la fois le RAE et le RAV.

15.6 Vignette présentant ces fonctions

Un concentré de l'utilisation de ces fonctions, ainsi qu'un dictionnaire des variables présentes dans les tables produites sont disponibles dans l'une des vignettes du package.

16

Exercices (béta)

Cette partie contient quelques exercices pour se familiariser à l'utilisation de `dplyr` avec les données du PMSI importées avec `pmeasyr`. Les données sont fictives et très simples, il s'agit simplement d'exercices autour de R.

16.0.0.1 1. Filtrer des rsa en fonction d' actes

Imaginez que vous disposez de rsa importés avec pmeasyr.

Instructions:

- charger le package dplyr
- Afficher la table actes des rsa$actes
- Filtrer cette table pour ne retenir que les lignes contenant l'acte NFKA015.

16.0.0.2 2. Filtrer sur les actes, les diagnostics et croiser les deux informations

Instructions:

- charger le package dplyr
- Afficher la table actes des rsa$actes
- Créer une table a en filtrer les actes des rsa pour ne retenir que les lignes contenant l'acte NFKA015.
- Créer une table b en filtrant les diagnostics des rsa pour ne retenir que les lignes contenant le code E66.
- Croiser ces deux tables