Importation/Exportation de données R

Version 4.3.1 (2023-06-16)

Équipe de base R

Ce manuel concerne R, version 4.3.1 (2023-06-16).

Table des matières

Remerciements

La partie bases de données relationnelles de ce manuel est basée en partie sur un manuel antérieur de Douglas Bates et Saikat DebRoy. L'auteur principal de ce manuel était Brian Ripley.

De nombreux bénévoles ont contribué aux packages utilisés ici. Les principaux auteurs du les forfaits mentionnés sont

DBI (https://CRAN.R-project.org/package=DBI) : David A.James

dataframes2xls (https://CRAN.R-project.org/package=dataframes2xls) : Guido van Steen

étranger (https://CRAN.R-project.org/package=foreign) : Thomas Lumley, Saikat DebRoy, Douglas Bates, Duncan Murdoch et Roger Bivand

gdata (https://CRAN.R-project.org/package=gdata) : Gregory R.Warnes

ncdf4 (https://CRAN.R-project.org/package=ncdf4) : David Pierce

rJava (https://CRAN.R-project.org/package=rJava) : Simon Urbanek

RJDBC (https://CRAN.R-project.org/package=RJDBC) : Simon Urbanek

RMySQL (https://CRAN.R-project.org/package=RMySQL) : David James et Saikat DebRoy

RNetCDF (https://CRAN.R-project.org/package=RNetCDF) : Pavel Michna

RODBC (https://CRAN.R-project.org/package=RODBC) : Michael Lapsley et Brian Ripley ROracle (https://

CRAN.R-project.org/package=ROracle) : David A. James

RPostgreSQL (https://CRAN.R-project.org/package=RPostgreSQL) : Sameer Kumar Prayaga et Tomoaki Nishiyama

RSPerl : Duncan Temple Lang

RSPython : Duncan Temple Lang

RSQLite (https://CRAN.R-project.org/package=RSQLite) : David A. James

Java: John Chambers et Duncan Temple Lang

WriteXLS (https://CRAN.R-project.org/package=WriteXLS) : Marc Schwartz

XLConnect (https://CRAN.R-project.org/package=XLConnect) : Mirai Solutions GmbH

XML (https://CRAN.R-project.org/package=XML) : Duncan Temple Lang

Brian Ripley est l'auteur du support pour les connexions.

1. Introduction

Lire des données dans un système statistique à des fins d'analyse et exporter les résultats vers un autre système pour rédiger un rapport peut être une tâche frustrante qui peut prendre beaucoup plus de temps que l'analyse statistique elle-même, même si la plupart des lecteurs trouveront cette dernière beaucoup plus attrayante.

Ce manuel décrit les fonctionnalités d'importation et d'exportation disponibles soit dans R lui-même, soit via des packages disponibles auprès du CRAN ou ailleurs.

Sauf indication contraire, tout ce qui est décrit dans ce manuel est (au moins en principe) disponible sur toutes les plateformes exécutant R.

En général, les systèmes statistiques comme R ne sont pas particulièrement adaptés aux manipulations de données à grande échelle. Certains autres systèmes sont meilleurs que R dans ce domaine, et une partie de l'objectif de ce manuel est de suggérer qu'au lieu de dupliquer les fonctionnalités de R, nous pouvons confier le travail à un autre système ! (Par exemple, Therneau & Grambsch (2000) ont commenté qu'ils préféraient manipuler les données dans SAS, puis utiliser la survie des packages (https://CRAN.R-project.org/package=survival) dans S pour l'analyse.) Manipulation de base de données les systèmes sont souvent très adaptés à la manipulation et à l'extraction de données : plusieurs packages permettant d'interagir avec les SGBD sont abordés ici.

Il existe des packages permettant d'intégrer directement des fonctionnalités développées dans des langages tels que Java, Perl et Python au code R, ce qui rend l'utilisation des fonctionnalités de ces langages encore plus appropriée. (Voir le package rJava (https://CRAN.R-project.org/package=rJava) de CRAN.)

Il convient également de rappeler que R comme S sont issus de la tradition Unix des petits outils réutilisables, et qu'il peut être enrichissant d'utiliser des outils tels que awk et perl pour manipuler des données avant l'importation ou après l'exportation. L'étude de cas de Becker, Chambers & Wilks (1988, chapitre 9) en est un exemple, où des outils Unix ont été utilisés pour vérifier et manipuler les données avant leur saisie dans S. Les outils Unix traditionnels sont désormais beaucoup plus largement disponibles, y compris pour Les fenêtres.

Ce manuel a été rédigé pour la première fois en 2000, et le nombre de packages R a été multiplié par cent depuis. Pour les formats de données spécialisés, il vaut la peine de rechercher s'il existe déjà un package approprié.

1.1 Importations

La forme de données la plus simple à importer dans R est un simple fichier texte, et cela sera souvent acceptable pour des problèmes à petite ou moyenne échelle. La fonction principale à importer à partir d'un fichier texte est la numérisation, et elle est à la base de la plupart des fonctions les plus pratiques abordées au chapitre 2 [Données de type feuille de calcul], page 6.

Cependant, tous les consultants en statistiques sont habitués à ce qu'un client leur présente une clé USB (anciennement une disquette ou un CD-R) contenant des données dans un format binaire propriétaire, par exemple « une feuille de calcul Excel » ou « un fichier SPSS ». Souvent, la chose la plus simple à faire est d'utiliser l'application d'origine pour exporter les données sous forme de fichier texte (et les consultants en statistiques disposent à cet effet de copies des applications les plus courantes sur leur ordinateur). Cependant, cela n'est pas toujours possible et le chapitre 3 [Importation depuis d'autres systèmes statistiques], page 12, explique les fonctionnalités disponibles pour accéder à ces fichiers directement depuis R. Pour les feuilles de calcul Excel, les méthodes disponibles sont résumées au chapitre 9 [Lecture d'Excel feuilles de calcul], page 27.

Dans quelques cas, les données ont été stockées sous forme binaire pour des raisons de compacité et de rapidité d'accès. Une application que nous avons vue à plusieurs reprises concerne les données d'imagerie, qui sont normalement stockées sous forme de flux d'octets représentés en mémoire, éventuellement précédés d'un en-tête. Ces formats de données sont abordés au Chapitre 5 [Fichiers binaires], page 20, et à la Section 7.5 [Connexions binaires], page 24.

Pour les bases de données beaucoup plus volumineuses, il est courant de gérer les données à l'aide d'un système de gestion de base de données (SGBD). Il existe à nouveau la possibilité d'utiliser le SGBD pour extraire un fichier brut, mais

pour beaucoup de ces SGBD, l'opération d'extraction peut être effectuée directement à partir d'un package R : voir Chapitre 4 [Bases de données relationnelles], page 14. L'importation de données via des connexions réseau est abordée dans le Chapitre 8 [Interfaces réseau], page 26.

1.1.1 Encodages À moins

que le fichier à importer soit entièrement en ASCII, il est généralement nécessaire de savoir comment il a été encodé. Pour les fichiers texte, un bon moyen de savoir quelque chose sur leur structure est l'outil de ligne de commande de fichier (pour Windows, inclus dans Rtools). Cela rapporte quelque chose comme

```
text.Rd : texte anglais Unicode UTF-8 text2.dat :
texte anglais ISO-8859 text3.dat : données de
caractères anglais Unicode UTF-16 Little-endian, avec terminateurs de ligne CRLF

intro.dat : texte Unicode UTF-8
intro.dat : texte Unicode UTF-8 (avec nomenclature)
```

Les systèmes modernes de type Unix, y compris macOS, sont susceptibles de produire des fichiers UTF-8. Windows peut produire ce qu'il appelle des fichiers « Unicode » (UCS-2LE ou éventuellement UTF-16LE[1]). Sinon, la plupart des fichiers seront codés sur 8 bits, sauf s'ils proviennent d'un environnement local chinois/japonais/coréen (qui a une large gamme d'encodages couramment utilisés). Il n'est pas possible de détecter automatiquement avec certitude quel encodage 8 bits (bien que des suppositions puissent être possibles et que le fichier puisse le deviner comme dans l'exemple ci-dessus), vous devrez peut-être simplement demander quelques indices à l'auteur (par exemple « russe sur Les fenêtres').

Les « nomenclatures » (Byte Order Marks, https://en.wikipedia.org/wiki/Byte_order_mark) provoquent des problèmes pour les fichiers Unicode. Dans le monde Unix, les nomenclatures sont rarement utilisées, alors que dans le monde Windows, elles concernent presque toujours les fichiers UCS-2/UTF-16 et souvent les fichiers UTF-8. L'utilitaire de fichiers ne reconnaîtra même pas les fichiers UCS-2 sans nomenclature, mais de nombreux autres utilitaires refuseront de lire les fichiers avec une nomenclature et les normes IANA pour UTF-16LE et UTF-16BE l'interdisent. Nous en avons trop souvent été réduits à regarder le fichier avec l'utilitaire de ligne de commande od ou un éditeur hexadécimal pour déterminer son encodage.

Notez que utf8 n'est pas un nom de codage valide (UTF-8 l'est) et que Macintosh est le plus portable.
nom de ce qu'on appelle parfois l'encodage « Mac Roman ».

1.2 Exporter vers des fichiers texte

L'exportation des résultats depuis R est généralement une tâche moins controversée, mais il existe encore un certain nombre d'embûches. Il y aura une application cible à l'esprit, et souvent un fichier texte sera le moyen d'échange le plus pratique. (Si un fichier binaire est requis, voir Chapitre 5 [Fichiers binaires], page 20.)

La fonction cat sous-tend les fonctions d'exportation de données. Il prend un argument file, et l'argument append permet d'écrire un fichier texte via des appels successifs à cat. Il est préférable, surtout si cela doit être fait plusieurs fois, d'ouvrir une connexion de fichier pour l'écriture ou l'ajout, de connecter cette connexion, puis de la fermer.

La tâche la plus courante consiste à écrire une matrice ou un bloc de données à classer sous la forme d'une grille rectangulaire de nombres, éventuellement avec des étiquettes de lignes et de colonnes. Cela peut être fait par les fonctions write.table et write. La fonction write écrit simplement une matrice ou un vecteur dans un nombre spécifié de colonnes (et transpose une matrice). La fonction write.table est plus pratique et écrit un bloc de données (ou un objet qui peut être contraint à un bloc de données) avec des étiquettes de ligne et de colonne.

Un certain nombre de problèmes doivent être pris en compte lors de l'écriture d'un bloc de données dans un fichier texte.

---

[1] la distinction est subtile, https://en.wikipedia.org/wiki/UTF-16/UCS-2, et l'utilisation de paires de substitution est très rare.

1. Précision

   La plupart des conversions de nombres réels/complexes effectuées par ces fonctions sont en toute précision, mais celles par écriture sont régies par le réglage actuel des options (chiffres). Pour plus de contrôle, utilisez le format sur un bloc de données, éventuellement colonne par colonne.

2. Ligne d'en-tête

   R préfère que la ligne d'en-tête n'ait aucune entrée pour les noms de lignes, donc le fichier ressemble à

   ```
                          dist    temps de montée
   Manteau vert           2.5     650         16.083
       ...
   ```

   Certains autres systèmes nécessitent une entrée (éventuellement vide) pour les noms de lignes, ce que write.table fournira si l'argument col.names = NA est spécifié. Excel est l'un de ces systèmes.

3. Séparateur

   Un séparateur de champ commun à utiliser dans le fichier est la virgule, car il est peu probable qu'elle apparaisse dans les champs des pays anglophones. Ces fichiers sont appelés fichiers CSV (valeurs séparées par des virgules) et la fonction wrapper write.csv fournit les valeurs par défaut appropriées. Dans certaines valeurs locales, la virgule est utilisée comme point décimal (définissez-la dans write.table par dec = ",") et là, les fichiers CSV utilisent le point-virgule comme séparateur de champ : utilisez write.csv2 pour les valeurs par défaut appropriées. Il existe une norme IETF pour les fichiers CSV (qui impose les virgules et les fins de ligne CRLF, pour lesquelles utilisez eol = "\r\n"), RFC4180 (voir https://www.rfc-editor.org/rfc/rfc4180) , mais ce qui est plus important en pratique, c'est que le fichier soit lisible par l'application à laquelle il est destiné.

   L'utilisation d'un point-virgule ou d'une tabulation (sep = "\t") est probablement l'option la plus sûre.

4. Valeurs manquantes

   Par défaut, les valeurs manquantes sont affichées sous la forme NA, mais cela peut être modifié par l'argument na. Notez que les NaN sont traités comme NA par write.table, mais pas par cat ni write.

5. Citation de chaînes

   Par défaut, les chaînes sont entre guillemets (y compris les noms de lignes et de colonnes). L'argument quote contrôle si les variables de caractères et de facteurs sont citées : certains programmes, par exemple Mondrian (https://en.wikipedia.org/wiki/Mondrian_(software)), n'acceptent pas les chaînes entre guillemets.

   Une certaine prudence est nécessaire si les chaînes contiennent des guillemets intégrés. Trois formulaires utiles sont

   ```
   > df <- data.frame(a = I("a \" quote")) > write.table(df) "a"


   "1" "a \" quote" >
   write.table(df, qmethod = "double") "a"

   "1" "un "" quote ">
   write.table(df, quote = FALSE, sep = ",")
   un
   1,un "     quote
   ```

   La seconde est la forme d'échappement couramment utilisée par les feuilles de calcul.

6. Encodages

   Les fichiers texte ne contiennent pas de métadonnées sur leurs encodages, donc pour les données non-ASCII , le fichier doit être ciblé sur l'application destinée à le lire. Toutes ces fonctions peuvent écrire sur une connexion qui permet de spécifier un encodage pour le fichier, et write.table a un argument fileEncoding pour faciliter cela.

Le plus difficile est de savoir quel encodage de fichier utiliser. Pour une utilisation sous Windows, il est préférable d'utiliser ce que Windows appelle « Unicode »2, c'est-à-dire « UTF-16LE ». L'utilisation d'UTF-8 est un bon moyen de créer des fichiers portables qui ne seront pas facilement confondus avec un autre encodage, mais même les applications macOS (où UTF-8 est l'encodage du système) peuvent ne pas les reconnaître, et les applications Windows sont très peu susceptibles de le faire. Apparemment, Excel : mac 2004/8 attendait des fichiers .csv en codage "macroman" (le codage utilisé dans les versions bien antérieures de Mac OS).

La fonction write.matrix du package MASS (https://CRAN.R-project.org/package=MASS) fournit une interface spécialisée pour l'écriture de matrices, avec la possibilité de les écrire en blocs et de réduire ainsi l'utilisation de la mémoire.

Il est possible d'utiliser Sink pour détourner la sortie R standard vers un fichier, et ainsi capturer la sortie des instructions d'impression (éventuellement implicites). Ce n'est généralement pas l'itinéraire le plus efficace et le paramètre d'options (largeur) devra peut-être être augmenté.

La fonction write.foreign du package Foreign (https://CRAN.R-project.org/package=foreign) utilise write.table pour produire un fichier texte et écrit également un fichier de code qui lira ce fichier texte dans un autre package statistique. Il existe actuellement une prise en charge de l'exportation vers SAS, SPSS et Stata.

1.3XML

Lors de la lecture de données à partir de fichiers texte, il est de la responsabilité de l'utilisateur de connaître et de spécifier les conventions utilisées pour créer ce fichier, par exemple le caractère de commentaire, la présence ou non d'une ligne d'en-tête, le séparateur de valeurs, la représentation des valeurs manquantes (et ainsi de suite) décrit dans la Section 1.2 [Exportation vers des fichiers texte], page 3. Un langage de balisage qui peut être utilisé pour décrire non seulement le contenu mais aussi la structure du contenu peut rendre un fichier auto-descriptif, de sorte qu'il n'est pas nécessaire de fournir ces détails au logiciel qui lit les données.

Le langage de balisage extensible – plus communément appelé XML – peut être utilisé pour fournir une telle structure, non seulement pour des ensembles de données standard mais également pour des structures de données plus complexes. XML devient extrêmement populaire et apparaît comme une norme pour le balisage et l'échange général de données. Il est utilisé par différentes communautés pour décrire des données géographiques telles que des cartes, des affichages graphiques, des mathématiques, etc.

XML fournit un moyen de spécifier l'encodage du fichier, par exemple

```
<?xml version="1.0" encoding="UTF-8"?>
```

même si cela ne l'exige pas.

Le package XML (https://CRAN.R-project.org/package=XML) fournit des fonctionnalités générales pour lire et écrire des documents XML dans R. Package StatDataML (https://CRAN.
R-project.org/package=StatDataML) sur CRAN est un exemple de construction sur XML (https://CRAN.R-project.org/package=XML). Une autre interface vers la bibliothèque C libxml2 est fournie par le package xml2 (https://CRAN.R-project.org/package=xml2).

yaml est un autre système de structuration de données textuelles, mettant l'accent sur la lisibilité humaine : il s'agit pris en charge par le package yaml (https://CRAN.R-project.org/package=yaml).

[2] Même dans ce cas, les applications Windows peuvent s'attendre à une marque d'ordre d'octets que l'implémentation d'iconv utilisée par R peut ajouter ou non en fonction de la plate-forme.

2 Données de type feuille de calcul

Dans la section 1.2 [Exportation vers des fichiers texte], page 3, nous avons vu un certain nombre de variations sur le format d'un fichier texte de type feuille de calcul, dans lequel les données sont présentées dans une grille rectangulaire, éventuellement avec des étiquettes de lignes et de colonnes. Dans cette section, nous envisageons d'importer de tels fichiers dans R.

2.1 Variations sur read.table

La fonction read.table est le moyen le plus pratique de lire une grille rectangulaire de données.
En raison des nombreuses possibilités, il existe plusieurs autres fonctions qui appellent read.table mais modifient un groupe d'arguments par défaut.

Attention, read.table est un moyen inefficace de lire dans de très grandes matrices numériques : voir scan ci-dessous.

Certaines des questions à considérer sont :

1. Encodage Si

   le fichier contient des champs de caractères non-ASCII , assurez-vous qu'il est lu avec le codage correct.
   Il s'agit principalement d'un problème de lecture de fichiers Latin-1 dans une locale UTF-8, ce qui peut être fait par quelque chose comme

   ```
   read.table("file.dat", fileEncoding="latin1")
   ```

   Notez que cela fonctionnera dans n'importe quelle locale pouvant représenter des chaînes Latin-1, mais pas dans de nombreux Greek/Russ/Chinese/Japanese . . . locales.

2. Ligne d'en-tête

   Nous vous recommandons de spécifier explicitement l'argument d'en-tête. Par convention, la ligne d'en-tête contient des entrées uniquement pour les colonnes et non pour les étiquettes de ligne. Un champ est donc plus court que les lignes restantes. (Si R voit cela, il définit header = TRUE.) Si on lui présente un fichier qui a un champ d'en-tête (éventuellement vide) pour les étiquettes de ligne, lisez-le par quelque chose comme

   ```
   read.table("file.dat", en-tête = TRUE, row.names = 1)
   ```

   Les noms de colonnes peuvent être donnés explicitement via col.names ; les noms explicites remplacent la ligne d'en-tête (si présente).

3. Séparateur

   Normalement, l'examen du fichier déterminera le séparateur de champs à utiliser, mais avec les fichiers séparés par des espaces, il peut y avoir le choix entre la valeur par défaut sep = "" qui utilise n'importe quel espace blanc (espaces, tabulations ou nouvelles lignes) comme séparateur. séparateur, sep = " " et sep = "\t". Notez que le choix du séparateur affecte la saisie des chaînes entre guillemets.

   Si vous disposez d'un fichier délimité par des tabulations contenant des champs vides, assurez-vous d'utiliser sep = "\t".

4. Citations Par

   défaut, les chaînes de caractères peuvent être citées soit par "", soit par '', et dans chaque cas, tous les caractères jusqu'à un guillemet correspondant sont considérés comme faisant partie de la chaîne de caractères. L'ensemble des caractères de guillemets valides (qui pourraient be none) est contrôlé par l'argument quote. Pour sep = "\n", la valeur par défaut est remplacée par quote = "".

   Si aucun caractère séparateur n'est spécifié, les guillemets peuvent être échappés dans les chaînes entre guillemets en les précédant immédiatement par '\', style C.

   Si un caractère séparateur est spécifié, les guillemets peuvent être échappés dans les chaînes entre guillemets en les doublant comme cela est conventionnel dans les feuilles de calcul. Par

   ```
   exemple "Une chaîne n'est pas deux", "une de plus"
   ```

   peut être lu par

   ```
   read.table("testfile", sep = ",")
   ```

   Cela ne fonctionne pas avec le séparateur par défaut.

5. Valeurs manquantes

Par défaut, le fichier est supposé contenir la chaîne de caractères NA pour représenter les valeurs manquantes, mais cela peut être modifié par l'argument na.strings, qui est un vecteur d'une ou plusieurs représentations de caractères des valeurs manquantes.

Les champs vides dans les colonnes numériques sont également considérés comme des valeurs manquantes.

Dans les colonnes numériques, les valeurs NaN, Inf et -Inf sont acceptées.

6. Lignes non remplies

Il est assez courant qu'un fichier exporté à partir d'une feuille de calcul ait tous les champs vides de fin (et leurs séparateurs) omis. Pour lire de tels fichiers, définissez fill = TRUE.

7. Espace blanc dans les champs de caractères

Si un séparateur est spécifié, les espaces blancs de début et de fin dans les champs de caractères sont considérés comme faisant partie du champ. Pour supprimer l'espace, utilisez l'argument strip.white = TRUE.

8. Lignes vides

Par défaut, read.table ignore les lignes vides. Cela peut être modifié en définissant blank.lines.skip = FALSE, ce qui ne sera utile qu'en conjonction avec fill = TRUE, peut-être pour utiliser des lignes vides pour indiquer les cas manquants dans une mise en page normale.

9. Classes pour les variables

Sauf si vous effectuez une action spéciale, read.table lit toutes les colonnes en tant que vecteurs de caractères, puis essaie de sélectionner une classe appropriée pour chaque variable du bloc de données. Il essaie tour à tour les valeurs logiques, entières, numériques et complexes, en passant à autre chose si une entrée ne manque pas et ne peut pas être convertie.1 Si toutes ces réponses échouent, la variable est convertie en facteur.

Les arguments colClasses et as.is offrent un meilleur contrôle. Spécifier as.is = TRUE supprime la conversion des vecteurs de caractères en facteurs (uniquement). L'utilisation de colClasses permet de définir la classe souhaitée pour chaque colonne de l'entrée : elle sera plus rapide et utilisera moins de mémoire.

Notez que colClasses et as.is sont spécifiés par colonne, et non par variable, et incluent donc la colonne des noms de lignes (le cas échéant).

10. Commentaires

Par défaut, read.table utilise '#' comme caractère de commentaire, et si cela se produit (sauf dans les chaînes entre guillemets), le reste de la ligne est ignoré. Lignes contenant uniquement un espace blanc et un
Les commentaires sont traités comme des lignes vides.

Si l'on sait qu'il n'y aura aucun commentaire dans le fichier de données, c'est plus sûr (et peut-être plus rapide) utiliser comment.char = "".

11. Les évasions

De nombreux systèmes d'exploitation ont des conventions pour utiliser la barre oblique inverse comme caractère d'échappement dans les fichiers texte, mais pas Windows (et utilise la barre oblique inverse dans les noms de chemin). Il est facultatif dans R si de telles conventions sont appliquées aux fichiers de données.

read.table et scan ont tous deux un argument logique AllowEscapes. Ceci est faux par défaut, et les barres obliques inverses sont alors uniquement interprétées comme (dans les circonstances décrites ci-dessus) des guillemets d'échappement. Si cet ensemble est vrai, les échappements de style C sont interprétés, à savoir les caractères de contrôle \a, \b, \f, \n, \r, \t, \v et les représentations octales et hexadécimales comme \040 et \ 0x2A. Tout autre caractère d'échappement est traité comme lui-même, y compris la barre oblique inverse. Notez que les échappements Unicode tels que \uxxxx ne sont jamais interprétés.

12. Encodage Ceci

peut être spécifié par l'argument fileEncoding, par exemple

```
fileEncoding = "UCS-2LE"                    # Fichiers 'Unicode' Windows
```

[1] C'est normalement rapide car le fait de regarder la première entrée exclut la plupart des possibilités.

fichierEncodage = "UTF-8"

Si vous connaissez (correctement) l'encodage du fichier, cela fonctionnera presque toujours. Cependant, nous connaissons une exception : les fichiers UTF-8 avec une nomenclature. Certaines personnes prétendent que les fichiers UTF-8 ne devraient jamais avoir de nomenclature, mais certains logiciels (y compris apparemment Excel:mac) les utilisent, et de nombreux systèmes d'exploitation de type Unix ne les acceptent pas. Alors face à un dossier qui se présente comme

intro.dat : texte Unicode UTF-8 (avec nomenclature) pouvant être

lu sous Windows par

read.table("intro.dat", fileEncoding = "UTF-8")

mais sous Unix, il faudra peut-être

read.table("intro.dat", fileEncoding = "UTF-8-BOM")

(Cela fonctionnerait très probablement sans spécifier d'encodage dans une locale UTF-8.)

Les fonctions pratiques read.csv et read.delim fournissent des arguments à read.table appropriés pour les fichiers CSV et délimités par des tabulations exportés à partir de feuilles de calcul dans des langues anglophones.
Les variantes read.csv2 et read.delim2 conviennent aux environnements régionaux où la virgule est utilisée pour le point décimal et (pour read.csv2) aux feuilles de calcul qui utilisent des points-virgules pour séparer les champs.

Si les options de read.table ne sont pas spécifiées correctement, le message d'erreur sera généralement de la forme

Erreur lors de l'analyse (fichier = fichier, quoi = quoi, sep = sep, :
la ligne 1 n'avait pas 5 éléments

ou

Erreur dans read.table("files.dat", header = TRUE) :
plus de colonnes que de noms de colonnes

Cela peut donner suffisamment d'informations pour trouver le problème, mais la fonction auxiliaire count.fields peut être utile pour approfondir ses recherches.

L'efficacité peut être importante lors de la lecture de grandes grilles de données. Cela aidera à spécifier comment.char = "", colClasses comme l'un des types de vecteurs atomiques (logique, entier, numérique, complexe, caractère ou peut-être brut) pour chaque colonne, et à donner nrows, le nombre de lignes à lire. (et une légère surestimation vaut mieux que de ne pas le préciser du tout). Voir les exemples dans les sections suivantes.

2.2 Fichiers au format de largeur fixe

Parfois, les fichiers de données n'ont pas de délimiteurs de champs mais contiennent des champs dans des colonnes prédéfinies. C'était très courant à l'époque des cartes perforées et est encore parfois utilisé pour économiser de l'espace dans les fichiers.

La fonction read.fwf fournit un moyen simple de lire de tels fichiers, en spécifiant un vecteur de largeurs de champ.
La fonction lit le fichier en mémoire sous forme de lignes entières, divise les chaînes de caractères résultantes, écrit un fichier temporaire séparé par des tabulations, puis appelle read.table. Ceci est suffisant pour les petits fichiers, mais pour tout ce qui est plus compliqué, nous vous recommandons d'utiliser les fonctionnalités d'un langage comme Perl pour prétraiter le fichier.

La fonction read.fortran est une fonction similaire pour les fichiers au format fixe, utilisant des spécifications de colonnes de style Fortran.

2.3 Format d'échange de données (DIF)

Un ancien format parfois utilisé pour les données de type feuille de calcul est le format DIF, ou Data Interchange.

La fonction read.DIF fournit un moyen simple de lire de tels fichiers. Il prend des arguments similaires à read.table pour attribuer des types à chacune des colonnes.

Sous Windows, les tableurs stockent souvent les données des feuilles de calcul copiées dans le presse-papiers dans ce format ; read.DIF("clipboard") peut le lire directement à partir de là. Il est légèrement plus robuste que read.table("clipboard") dans la gestion des feuilles de calcul avec des cellules vides.

2.4 Utilisation directe du scan

read.table et read.fwf utilisent tous deux scan pour lire le fichier, puis traitent les résultats de l'analyse.
Ils sont très pratiques, mais il est parfois préférable d'utiliser directement le scan.

L'analyse des fonctions comporte de nombreux arguments, dont la plupart ont déjà été abordés sous read.table. L'argument le plus crucial est quoi, qui spécifie une liste de modes de variables à lire dans le fichier. Si la liste est nommée, les noms sont utilisés pour les composants de la liste renvoyée. Les modes peuvent être numériques, caractères ou complexes, et sont généralement spécifiés par un exemple, par exemple 0, "" ou 0i. Par exemple cat("2 3 5 7", "11 13 17 19", file="ex.dat",

```
sep="\n") scan(file="ex.dat", what=list(x=0 , y="", z=0), flush=VRAI)
```

renvoie une liste de trois composants et supprime la quatrième colonne du fichier.

Il existe une fonction readLines qui sera plus pratique si tout ce que vous voulez est de lire des lignes entières dans R pour un traitement ultérieur.

Une utilisation courante du scan consiste à lire dans une grande matrice. Supposons que le fichier Matrix.dat contienne simplement les nombres pour une matrice 200 x 2000. Nous pouvons alors utiliser

```
A <- matrice(scan("matrix.dat", n = 200*2000), 200, 2000, byrow = TRUE)
```

Sur un test, cela prenait 1 seconde (sous Linux, 3 secondes sous Windows sur la même machine) alors que

```
Un <- as.matrix(read.table("matrix.dat"))
```

a pris 10 secondes (et plus de mémoire), et

```
A <- as.matrix(read.table("matrix.dat", header = FALSE, nrows = 200,
                                        comment.char = "", colClasses = "numeric"))
```

a pris 7 secondes. La différence est presque entièrement due à la surcharge liée à la lecture de 2 000 colonnes courtes distinctes : si elles avaient une longueur de 2 000, l'analyse prenait 9 secondes alors que read.table en prenait 18 si elle était utilisée efficacement (en particulier, en spécifiant les colClasses) et 125 si elle était utilisée naïvement.

Notez que les délais peuvent dépendre du type de lecture et des données. Pensez à lire un million d'entiers distincts :

```
writeLines(as.character((1+1e6):2e6), "ints.dat") xi <- scan("ints.dat",
what=integer(0), n=1e6) # 0,77s xn <- scan ("ints.dat", what=numeric(0), n=1e6)
# 0,93s xc <- scan("ints.dat", what=character(0), n=1e6) # 0,85s xf <- as .facteur(xc)

                                                                    #2.2s
DF <- read.table("ints.dat")                                        #4.5s
```

et un million d'exemples d'un petit ensemble de codes :

```
code <- c("LMH", "SJC", "CHCH", "SPC", "SOM")
writeLines(sample(code, 1e6, replace=TRUE), " code.dat") y <- scan("code.dat",
what=character(0), n=1e6) # 0,44s yf <- as.factor(y) # 0,21s DF <- read.table("
code.dat")
                                                                    #4.9s
DF <- read.table("code.dat", nrows=1e6)                             # 3,6
```

Notez que ces délais dépendent fortement du système d'exploitation (les lectures de base sous Windows prennent au moins deux fois plus de temps que ces temps sous Linux) et de l'état précis des déchets.
collectionneur.

2.5 Remodeler les données

Parfois, les données d'une feuille de calcul sont dans un format compact qui donne les covariables pour chaque sujet suivi de toutes les observations à ce sujet. Les fonctions de modélisation de R nécessitent des observations dans un seule colonne. Considérez l'échantillon suivant de données provenant de mesures répétées du cerveau par IRM

Âge du statut	V1	V2	V3	V4
P23646	45190	50333	55166	56271
CC26174	35535	38227	37911	41184
CC27723	25691	25712	26144	26398
CC27193	30949	29693	29754	30772
CC24370	50542	51966	54341	54273
CC28359	58591	58803	59435	61292
CC25136	45801	45389	47197	47126

Il existe deux covariables et jusqu'à quatre mesures sur chaque sujet. Les données ont été exportées à partir d'Excel sous forme de fichier mr.csv.

Nous pouvons utiliser la pile pour aider à manipuler ces données afin de donner une réponse unique.

```
zz <- read.csv("mr.csv", strip.white = TRUE)
zzz <- cbind(zz[gl(nrow(zz), 1, 4*nrow(zz)), 1:2], stack(zz[, 3:6]))
```

avec résultat

	Statut	Âge	valeurs	ind
X1	P23646	45190	V1	
X2	CC26174	35535	V1	
X3	CC27723	25691	V1	
X4	CC27193	30949	V1	
X5	CC24370	50542	V1	
X6	CC28359	58591	V1	
X7	CC25136	45801	V1	
X11	P23646	50333	V2	
...				

La fonction de dépilage va dans la direction opposée et peut être utile pour exporter des données.

Une autre façon de procéder est d'utiliser la fonction reshape, en

```
> reshape(zz, idvar="id",timevar="var",
    variant=list(c("V1","V2","V3","V4")),direction="long")
```

	Statut	Âge	var	ID	V1
1.1	P23646		1	45190	1
2.1	CC26174		1	35535	2
3.1	CC27723		1	25691	3
4.1	CC 27193		1	30949	4
5.1	CC24370		1	50542	5
6.1	CC 28359		1	58591	6
7.1	CC25136		1	45801	7
1.2	P 23646		2	50333	1
2.2	CC 26174		2	38227	2
...					

La fonction reshape a une syntaxe plus compliquée que celle de stack mais peut être utilisée pour les données où la forme « longue » comporte plus d'une colonne dans cet exemple. Avec direction="wide", reshape peut également effectuer la transformation inverse.

Certaines personnes préfèrent les outils des packages reshape (https://CRAN.R-project.org/package=reshape), reshape2 (https://CRAN.R-project.org/package=reshape2) et plyr (https://CRAN.R-project.org/package=plyr).

2.6 Tableaux de contingence plats

L'affichage de tableaux de contingence de plus grande dimension sous forme de tableau est généralement plutôt gênant. Dans l'analyse de données catégorielles, ces informations sont souvent représentées sous la forme de tableaux bidimensionnels bordés avec des lignes et des colonnes principales spécifiant la combinaison de niveaux de facteurs correspondant au nombre de cellules. Ces lignes et colonnes sont généralement « irrégulières » dans le sens où les étiquettes ne sont affichées que lorsqu'elles changent, avec la convention évidente selon laquelle les lignes sont lues de haut en bas et les colonnes de gauche à droite. Dans R, de tels tableaux de contingence « plats » peuvent être créés à l'aide de ftable, qui crée des objets de classe « ftable » avec une méthode d'impression appropriée.

À titre d'exemple simple, considérons l'ensemble de données standard R UCBAdmissions, qui est un tableau de contingence tridimensionnel résultant de la classification des candidats aux études supérieures à l'UC Berkeley pour les six plus grands départements en 1973, classés par admission et par sexe.

```
> données (UCBAdmissions) >
ftable (UCBAdmissions)
                     Département ABCDEF
Admettre Genre
Admis Homme                512 353 120 138 53 22
         Femelle            89 17 202 131 94 24
Homme rejeté               313 207 205 279 138 351
         Femelle            19 8 391 244 299 317
```

La représentation imprimée est clairement plus utile que l'affichage des données sous forme tridimensionnelle.
tableau.

Il existe également une fonction read.ftable pour lire des tableaux de contingence plats à partir de fichiers.
Cela comporte des arguments supplémentaires pour traiter les variantes sur la façon dont les informations sur les noms et les niveaux des variables de ligne et de colonne sont représentées exactement. La page d'aide de read.ftable contient quelques exemples utiles. Les tableaux plats peuvent être convertis en tableaux de contingence standard sous forme de tableau à l'aide de as.table.

Notez que les tableaux plats se caractérisent par leur affichage « irrégulier » des étiquettes de lignes (et peut-être aussi de colonnes). Si la grille complète des niveaux des variables de ligne est donnée, il faut plutôt utiliser read.table pour lire les données et créer le tableau de contingence à partir de cela à l'aide de xtabs.

3 Importation depuis d'autres systèmes statistiques

Dans ce chapitre, nous considérons le problème de la lecture d'un fichier de données binaires écrit par un autre système statistique. Il est souvent préférable d'éviter cette situation, mais elle peut s'avérer inévitable si le système d'origine n'est pas disponible.

Dans tous les cas, les fonctionnalités décrites ont été écrites pour des fichiers de données provenant de versions spécifiques de l'autre système (souvent au début des années 2000), et n'ont pas nécessairement été mises à jour pour les versions les plus récentes de l'autre système.

3.1 EpiInfo, Minitab, S-PLUS, SAS, SPSS, Stata, Systat Le package Foreign recommandé (https://CRAN.R-

project.org/package=foreign) fournit des fonctionnalités d'importation pour les fichiers produits par ces systèmes statistiques, et pour l'exportation vers Stata.
Dans certains cas, ces fonctions peuvent nécessiter beaucoup moins de mémoire que read.table. write.foreign (voir Section 1.2 [Exporter vers des fichiers texte], page 3) fournit un mécanisme d'exportation prenant actuellement en charge SAS, SPSS et Stata.

Les versions 5 et 6 d'EpiInfo stockaient les données dans un format de texte auto-descriptif à largeur fixe. read.epiinfo lira ces fichiers .REC dans une trame de données R. EpiData produit également des données dans ce format.

La fonction read.mtp importe une « feuille de calcul portable Minitab ». Cela renvoie les composants de la feuille de calcul sous forme de liste R.

La fonction read.xport lit un fichier au format SAS Transport (XPORT) et renvoie une liste de trames de données. Si SAS est disponible sur votre système, la fonction read.ssd peut être utilisée pour créer et exécuter un script SAS qui enregistre un ensemble de données permanent SAS (.ssd ou .sas7bdat) au format Transport.
Il appelle ensuite read.xport pour lire le fichier résultant. (Le package Hmisc (https://CRAN.R-project.org/package=Hmisc) a une fonction similaire sas.get, exécutant également SAS.) Pour ceux qui n'ont pas accès à SAS mais fonctionnent sous Windows, SAS System Viewer (un téléchargement gratuit) peut être utilisé pour ouvrir des ensembles de données SAS et les exporter, par exemple, au format .csv.

Fonction read.S qui peut lire des objets binaires produits par S-PLUS 3.x, 4.x ou 2000 sous Unix (32 bits) ou Windows (et peut les lire sur un autre système d'exploitation). Celui-ci est capable de lire de nombreux objets S, mais pas tous : en particulier, il peut lire des vecteurs, des matrices et des trames de données ainsi que des listes les contenant.

La fonction data.restore lit les dumps de données S-PLUS (créés par data.dump) avec les mêmes restrictions (sauf que les dumps de la plateforme Alpha peuvent également être lus). Il devrait être possible de lire les dumps de données de S-PLUS 5.x et versions ultérieures écrites avec data.dump(oldStyle=T).

Si vous avez accès à S-PLUS, il est généralement plus fiable de vider le ou les objets dans S-PLUS et de sourcer le fichier de vidage dans R. Pour S-PLUS 5.x et versions ultérieures, vous devrez peut-être utiliser dump(. .., oldStyle=T), et pour lire des objets très volumineux, il peut être préférable d'utiliser le fichier dump comme script batch plutôt que d'utiliser la fonction source.

La fonction read.spss peut lire les fichiers créés par les commandes « enregistrer » et « exporter » dans SPSS. Il renvoie une liste avec un composant pour chaque variable de l'ensemble de données enregistré. Les variables SPSS avec des étiquettes de valeur sont éventuellement converties en facteurs R.

SPSS Data Entry est une application permettant de créer des formulaires de saisie de données. Par défaut, il crée des fichiers de données avec des informations de formatage supplémentaires que read.spss ne peut pas gérer, mais il est possible d'exporter les données dans un format SPSS ordinaire .

Certaines applications tierces prétendent produire des données « au format SPSS », mais avec des différences dans les formats : read.spss peut ou non être capable de les gérer.

Les fichiers Stata .dta sont un format de fichier binaire. Les fichiers des versions 5 à 12 de Stata peuvent être lus et écrits par les fonctions read.dta et write.dta. Les variables Stata avec des étiquettes de valeur sont éventuellement converties en (et à partir de) facteurs R. Pour les versions Stata 13 et ultérieures

voir les packages CRAN readstata13 (https://CRAN.R-project.org/package=readstata13) et
refuge (https://CRAN.R-project.org/package=haven).

read.systat lit les fichiers Systat SAVE qui sont des fichiers de données rectangulaires (mtype = 1) écrits sur des machines Little-Endian (comme celles de Windows). Ceux-ci ont l'extension .sys ou (plus récemment) .sud.

3.2 Octaves

Octave est un système d'algèbre linéaire numérique (https://octave.org/) et une fonction read.octave dans le package Foreign (https://CRAN.R-project.org/package=foreign) peut lire des fichiers au format de données texte Oc-tave créés à l'aide de la commande Octave save -ascii, avec prise en charge de la plupart des types courants de variables, y compris les variables standard atomiques (scalaires réels et complexes, matrices et tableaux Nd, chaînes, plages et scalaires et matrices booléens) et récursives (structures, cellules et listes).

4 Bases de données relationnelles

4.1 Pourquoi utiliser une base de données ?

Il existe des limites quant aux types de données que R gère correctement. Étant donné que toutes les données manipulées par R résident en mémoire et que plusieurs copies des données peuvent être créées lors de l'exécution d'une fonction, R n'est pas bien adapté aux ensembles de données extrêmement volumineux. Les objets de données d'une taille supérieure à (quelques) centaines de mégaoctets peuvent entraîner un manque de mémoire dans R, en particulier sur un système d'exploitation 32 bits.

R ne prend pas facilement en charge l'accès simultané aux données. Autrement dit, si plusieurs utilisateurs accèdent aux mêmes données, et éventuellement les mettent à jour, les modifications apportées par un utilisateur ne seront pas visibles par les autres.

R prend en charge la persistance des données, dans la mesure où vous pouvez enregistrer un objet de données ou une feuille de calcul entière à partir d'une session et le restaurer lors de la session suivante, mais le format des données stockées est spécifique à R et n'est pas facilement manipulable par d'autres systèmes.

Systèmes de gestion de bases de données (SGBD) et en particulier SGBD relationnels (SGBDR) sont conçus pour bien faire toutes ces choses. Leurs points forts sont 1. Fournir un

accès rapide à des parties sélectionnées de grandes bases de données.

2. Des moyens puissants pour résumer et croiser les colonnes des bases de données.
3. Stockez les données de manière plus organisée que le modèle de grille rectangulaire des feuilles de calcul et R trames de données.
4. Accès simultané à partir de plusieurs clients exécutés sur plusieurs hôtes tout en renforçant la sécurité contraintes d'accès aux données.
5. Capacité à agir en tant que serveur auprès d'un large éventail de clients.

Les types d'applications statistiques pour lesquelles le SGBD pourrait être utilisé consistent à extraire un échantillon de 10 % des données, à croiser les données pour produire un tableau de contingence multidimensionnel et à extraire les données groupe par groupe d'une base de données pour une analyse séparée.

De plus en plus de systèmes d'exploitation utilisent eux-mêmes des SGBD pour ces raisons. Il est donc probable qu'un tel système soit déjà installé sur votre système d'exploitation (non Windows). Akonadi (https:// en.wikipedia.org/wiki/Akonadi) est utilisé par KDE4 pour stocker des informations personnelles. Plusieurs applications macOS, notamment Mail et Carnet d'adresses, utilisent SQLite.

4.2 Présentation des SGBDR

Traditionnellement, il existait des SGBDR commerciaux volumineux (et coûteux) (Informix (https://www.ibm.com/products/informix) ; Oracle (https://www.oracle.com) ; Sybase ; DB2 d'IBM (https:/ /www.ibm.com/products/db2); Microsoft SQL Server (https://www.microsoft.com/sql-server/) sous Windows) et bases de données académiques et pour petits systèmes (telles que MySQL[1] , greSQL, Microsoft Access , . .), les premiers se caractérisent par une importance beaucoup plus grande Poste- accordée aux fonctionnalités de sécurité des données. La frontière s'estompe, MySQL et PostgreSQL ayant de plus en plus de fonctionnalités haut de gamme et des versions « express » gratuites étant mises à disposition pour les SGBD commerciaux.

Il existe d'autres sources de données couramment utilisées, notamment des feuilles de calcul, des bases de données non relationnelles et même des fichiers texte (éventuellement compressés). Open Database Connectivity (ODBC) est une norme pour utiliser toutes ces sources de données. Il est originaire de Windows (voir https://docs.microsoft.com/en-us/sql/odbc/microsoft-open-database-connectivity-odbc) mais est également implémenté sous Linux/Unix/macOS.

[1] et des forks, notamment MariaDB.

Tous les packages décrits plus loin dans ce chapitre fournissent des clients aux bases de données client/serveur. La base de données peut résider sur la même machine ou (plus souvent) à distance. Il existe une norme ISO (en fait plusieurs : SQL92 est ISO/IEC 9075, également connu sous le nom d' ANSI X3.135-1992, et SQL99 est en cours d'utilisation) pour un langage d'interface appelé SQL (Structured Query Language, parfois prononcé « suite »). : voir Bowman et al. 1996 et Kline et Kline 2001) que ces SGBD prennent en charge à des degrés divers.

4.2.1 Requêtes SQL

Les interfaces R plus complètes génèrent du SQL en arrière-plan pour les opérations courantes, mais l'utilisation directe de SQL est nécessaire pour les opérations complexes. Traditionnellement, SQL est écrit en majuscules, mais de nombreux utilisateurs trouveront plus pratique d'utiliser des minuscules dans les fonctions de l'interface R.

Un SGBD relationnel stocke les données sous la forme d'une base de données de tables (ou relations) assez similaires aux trames de données R, dans le sens où elles sont constituées de colonnes ou de champs d'un même type (numérique, caractère, date, devise, . . .) et lignes ou enregistrements contenant les observations d'une entité.

Les « requêtes » SQL sont des opérations assez générales sur une base de données relationnelle. La requête classique est une Instruction SELECT du type

```
SELECT Etat, Meurtre FROM USAArrestations O Viol > 30 ORDER BY Meurtre

SELECT t.sch, c.moyens, t.sex, t.achieve
    DE l'élève comme t, de l'école comme c OÙ t.sch = c.id

SELECT le sexe, COUNT(*) DU GROUPE d'étudiants PAR sexe

SELECT sch, AVG(sestat) FROM student GROUP BY sch LIMITE 10
```

Le premier d'entre eux sélectionne deux colonnes de la trame de données R USArrests qui a été copiée dans une table de base de données, des sous-ensembles sur une troisième colonne et demande que les résultats soient triés. La seconde effectue une jointure de base de données sur deux tables étudiant et école et renvoie quatre colonnes. Les troisième et quatrième requêtes effectuent des tableaux croisés et renvoient des décomptes ou des moyennes. (Les cinq fonctions d'agrégation sont COUNT(*) et SUM, MAX, MIN et AVG, chacune appliquée à une seule colonne.)

Les requêtes SELECT utilisent FROM pour sélectionner la table, WHERE pour spécifier une condition d'inclusion (ou plusieurs conditions séparées par AND ou OR) et ORDER BY pour trier le résultat.
Contrairement aux trames de données, il est préférable de considérer les lignes des tables SGBDR comme non ordonnées, et sans instruction ORDER BY, l'ordre est indéterminé. Vous pouvez trier (par ordre lexicographique) sur plusieurs colonnes en les séparant par des virgules. Placer DESC après un ORDER BY place le tri par ordre décroissant.

Les requêtes SELECT DISTINCT ne renverront qu'une copie de chaque ligne distincte de la table sélectionnée.

La clause GROUP BY sélectionne des sous-groupes de lignes selon le critère. Si plusieurs colonnes sont spécifiées (séparées par des virgules), les classifications croisées multidirectionnelles peuvent être résumées par l'une des cinq fonctions d'agrégation. Une clause HAVING permet à la sélection d'inclure ou d'exclure des groupes en fonction de la valeur agrégée.

Si l'instruction SELECT contient une instruction ORDER BY qui produit un classement unique, une clause LIMIT peut être ajoutée pour sélectionner (par numéro) un bloc contigu de lignes de sortie. Cela peut être utile pour récupérer des lignes bloc par bloc. (Il peut ne pas être fiable à moins que l'ordre soit unique, car la clause LIMIT peut être utilisée pour optimiser la requête.)

Il existe des requêtes pour créer une table (CREATE TABLE, mais généralement on copie une trame de données dans la base de données dans ces interfaces), INSERT ou DELETE ou UPDATE données. Une table est détruite par une 'requête' DROP TABLE.

Kline et Kline (2001) discutent des détails de l'implémentation de SQL dans Microsoft SQL Serveur 2000, Oracle, MySQL et PostgreSQL.

4.2.2 Types de données Les

données peuvent être stockées dans une base de données sous différents types de données. La gamme de types de données est spécifique au SGBD, mais la norme SQL définit de nombreux types, notamment les suivants qui sont largement implémentés (souvent sous le nom SQL).

float(p) Nombre réel, avec précision facultative. Souvent appelé réel ou double ou double précision.

entier Entier de 32 bits. Souvent appelé int. smallint entier

de 16 bits

caractère(n)
chaîne de caractères de longueur fixe. Souvent appelé char.

caractère variable(n) chaîne
de caractères de longueur variable. Souvent appelé varchar. A presque toujours une limite de 255 caractères.

booléen vrai ou faux. Parfois appelé bool ou bit.

date date du calendrier

temps moment de la journée

horodatage
date et l'heure

Il existe des variantes d'heure et d'horodatage, avec fuseau horaire. D'autres types largement implémentés sont le texte et le blob, pour les gros blocs de texte et de données binaires, respectivement.

Le plus complet des packages d'interface R masque les problèmes de conversion de type au utilisateur.

4.3 Packages d'interface R Il existe plusieurs

packages disponibles sur CRAN pour aider R à communiquer avec les SGBD. Ils offrent différents niveaux d'abstraction. Certains fournissent des moyens de copier des trames de données entières vers et depuis des bases de données. Tous ont des fonctions pour sélectionner des données dans la base de données via des requêtes SQL et pour récupérer le résultat dans son ensemble sous forme de bloc de données ou par morceaux (généralement sous forme de groupes de lignes).

Tous sauf RODBC (https://CRAN.R-project.org/package=RODBC) sont liés à un seul SGBD, mais il y a eu une proposition pour un package « front-end » unifié DBI (https://CRAN.R-project.org/package=DBI) (https://developer.r-project.org/db/) dans en conjonction avec un 'back-end' dont le plus développé est RMySQL (https://CRAN.

R-project.org/package=RMySQL). Sur CRAN se trouvent également les back-ends ROracle (https:// CRAN.R-project.org/package=ROracle), RPostgreSQL (https://CRAN.R-project.org/package=RPostgreSQL) et RSQLite (https : //CRAN.R-project.org/package=RSQLite) (qui fonctionne avec le SGBD SQLite fourni, https://www.sqlite.org/index.html) et RJDBC (https:// CRAN.R-project. org/package=RJDBC) (qui utilise Java et peut se connecter à n'importe quel SGBD doté d'un pilote JDBC).

PL/R (https://github.com/postgres-plr/plr) est un projet visant à intégrer R dans PostgreSQL.

Le package RMongo (https://CRAN.R-project.org/package=RMongo) fournit une interface R à un client Java pour les bases de données 'MongoDB' (https://en.wikipedia.org/wiki/MongoDB), qui sont interrogé en utilisant JavaScript plutôt que SQL. Le package mongolite (https://CRAN.R-project.org/package=mongolite) est un autre client utilisant le pilote C de mongodb.

4.3.1 Packages utilisant DBI

Le package RMySQL (https://CRAN.R-project.org/package=RMySQL) sur CRAN fournit une interface au système de base de données MySQL (voir https://www.mysql.com et Dubois, 2000) ou son fork MariaDB (voir https://mariadb.org/). La description ici s'applique aux versions 0.5-0 et plus tard : les versions antérieures avaient une interface sensiblement différente. La version actuelle nécessite le package DBI (https://CRAN.R-project.org/package=DBI), et cette description s'appliquera avec des modifications mineures à tous les autres back-ends de DBI (https://CRAN.R-project .org/package=DBI).

MySQL existe sur Unix/Linux/macOS et Windows : une 'Community Edition' est disponible sous GPL mais des licences commerciales sont également disponibles. MySQL était à l'origine un système « léger et simple » base de données. (Il préserve la casse des noms pour lesquels le système de fichiers d'exploitation est sensible à la casse, donc pas sous Windows.)

L'appel dbDriver("MySQL") renvoie un objet gestionnaire de connexions à la base de données, puis un l'appel à dbConnect ouvre une connexion à la base de données qui peut ensuite être fermée par un appel à la fonction générique dbDisconnect. Utilisez dbDriver("Oracle"), dbDriver("PostgreSQL") ou dbDriver("SQLite") avec ces SGBD et packages ROracle (https://CRAN.R-project.org/package=ROracle), RPostgreSQL (https://CRAN.R-project.org/package=RPostgreSQL) ou RSQLite (https://CRAN.R-project.org/package=RSQLite) respectivement.

Les requêtes SQL peuvent être envoyées par dbSendQuery ou dbGetQuery. dbGetquery envoie la requête et récupère les résultats sous forme de trame de données. dbSendQuery envoie la requête et renvoie un objet de classe héritant de "DBIResult" qui peut être utilisée pour récupérer les résultats, et par la suite utilisé dans un appel à dbClearResult pour supprimer le résultat.

La fonction fetch est utilisée pour récupérer tout ou partie des lignes du résultat de la requête, sous forme de liste. La fonction dbHasCompleted indique si toutes les lignes ont été récupérées, et dbGetRowCount renvoie le nombre de lignes dans le résultat.

Ce sont des interfaces pratiques pour lire/écrire/tester/supprimer des tables dans la base de données. dbReadTable et dbWriteTable copient vers et depuis une trame de données R, mappant les noms de lignes du bloc de données au champ row_names dans la table MySQL.

```
> library(RMySQL) # chargera également DBI
## ouvrir une connexion à une base de données MySQL
> avec <- dbConnect(dbDriver("MySQL"), dbname = "test")
## lister les tables de la base de données
> dbListTables(con)
## charge un bloc de données dans la base de données, en supprimant toute copie existante
> données (USAArrestes)
> dbWriteTable(con, "arrestations", USArrests, overwrite = TRUE)
VRAI
> dbListTables(con)
 [1] "arrestations"
## récupère toute la table
> dbReadTable(con, "arrestations")
                        Meurtre, Agression, UrbanPop, Viol
Alabama                   13,2      236 58 21,2
Alaska                    10,0      263 48 44,5
Arizona                    8,1      294          80 31,0
Arkansas                   8,8      190          50 19,5
 ...
## Sélectionnez dans la table chargée
> dbGetQuery(con, paste("select row_names, Meurtre suite à des arrestations",
                                    "où Viol > 30 ordre par Meurtre"))
      row_names Meurtre
1 Colorado 7,9
2         Arizona      8.1
3 Californie           9.0
4          Alaska 10,0
```

```
5 Nouveau-Mexique 11,4
6 Michigan 12,1
7 Nevada 12.2
8 Floride 15.4 >
dbRemoveTable(con, "arrestations") >
dbDisconnect(con)
```

4.3.2 Package RODBC Le package RODBC

(https://CRAN.R-project.org/package=RODBC) sur CRAN fournit une interface aux sources de bases de données prenant en charge une interface ODBC . Ceci est très largement disponible et permet au même code R d'accéder à différents systèmes de bases de données. RODBC (https://CRAN.R-project.org/package=RODBC) fonctionne sous Unix/Linux, Windows et macOS, et presque tous les systèmes de bases de données prennent en charge ODBC. Nous avons testé Microsoft SQL Server, Access, MySQL, PostgreSQL, Oracle et IBM DB2 sous Windows et MySQL, MariaDB, Oracle, PostgreSQL et SQLite sous Linux.

ODBC est un système client-serveur, et nous nous sommes connectés avec plaisir à un SGBD fonctionnant sur un serveur Unix à partir d'un client Windows, et vice versa.

Sous Windows, la prise en charge ODBC fait partie du système d'exploitation. Sous Unix/Linux, vous aurez besoin d'un gestionnaire de pilotes ODBC tel que unixODBC (https://www.unixodbc.org/) ou iOBDC (https://www.iodbc. org/ : il est préinstallé dans macOS) et d'un pilote installé pour votre système de base de données.

Windows fournit des pilotes non seulement pour les SGBD, mais également pour les feuilles de calcul Excel (.xls), les fichiers DBase (.dbf) et même les fichiers texte. (Les applications nommées n'ont pas besoin d'être installées. Les formats de fichiers pris en charge dépendent des versions des pilotes.) Il existe des versions pour Excel et Access 2007/2010 (allez sur https://www.microsoft.com/en- us/download/default.aspx, et recherchez « Office ODBC », qui mènera à AccessDatabaseEngine.exe), le « 2007 Office System Driver » (ce dernier a une version pour Windows 64 bits, et cela sera également lu plus tôt versions).

Sur macOS, les pilotes Actual Technologies (https://www.actualtech.com/product_access.php) fournissent des interfaces ODBC aux bases de données Access et aux feuilles de calcul Excel (hors Excel 2007/2010).

De nombreuses connexions simultanées sont possibles. Une connexion est ouverte par un appel à odbcConnect ou odbcDriverConnect (qui sur l'interface graphique Windows permet de sélectionner une base de données via des boîtes de dialogue) qui renvoie un handle utilisé pour un accès ultérieur à la base de données. L'impression d'une connexion fournira des détails sur la connexion ODBC, et l'appel de odbcGetInfo donnera des détails sur le client et le serveur.

Une connexion est fermée par un appel à close ou odbcClose, et également (avec un avertissement) lorsqu'aucun objet R n'y fait référence et à la fin d'une session R.

Les détails des tables sur une connexion peuvent être trouvés à l'aide de sqlTables.

La fonction sqlSave copie une trame de données R dans une table de la base de données et sqlFetch copie une trame de données R. table de la base de données vers une trame de données R.

Une requête SQL peut être envoyée à la base de données par un appel à sqlQuery. Cela renvoie le résultat dans une trame de données R. (sqlCopy envoie une requête à la base de données et enregistre le résultat sous forme de table dans la base de données.) Un niveau de contrôle plus précis est atteint en appelant d'abord odbcQuery, puis sqlGetResults pour récupérer les résultats. Ce dernier peut être utilisé au sein d'une boucle pour récupérer un nombre limité de lignes à la fois, tout comme la fonction sqlFetchMore.

Voici un exemple utilisant PostgreSQL, pour lequel le pilote ODBC mappe les noms de colonnes et de blocs de données en minuscules. Nous utilisons une base de données testdb que nous avons créée précédemment et dont le DSN (nom de la source de données) est configuré dans ~/.odbc.ini sous unixODBC. Exactement le même code fonctionnait avec MyODBC pour accéder à une base de données MySQL sous Linux ou Windows (où MySQL également

mappe les noms en minuscules). Sous Windows, les DSN sont paramétrés dans l' applet ODBC du Control Panel (« Sources de données (ODBC) » dans la section « Outils d'administration »).

```
> bibliothèque (RODBC)
## dites-lui de mapper les noms sur l/case
> canal <- odbcConnect("testdb", uid="ripley", case="tolower")
## charger un bloc de données dans la base de données
> données (USAArrestes)
> sqlSave (canal, USArrests, rownames = "state", addPK = TRUE)
> rm(USAArrestations)
## lister les tables de la base de données
> sqlTables (canal)
    TABLE_QUALIFIER TABLE_OWNER TABLE_NAME TABLE_TYPE REMARQUES
1                                   usarrests          TABLEAU
## Listez-le
> sqlFetch (canal, "USArrests", noms de lignes = "état")
                meurtre agression pop urbain rap
Alabama          13,2       236 58 21,2
Alaska           10,0       263 48 44,5
        ...
## une requête SQL, à l'origine sur une seule ligne
> sqlQuery (canal, "sélectionner l'état, meurtre depuis USArrests
                où viol > 30 ordre par meurtre")
          meurtre d'État
1 Colorado          7.9
2Arizona            8.1
3 Californie        9.0
4Alaska            10,0
5 Nouveau-Mexique 11,4
6Michigan 12,1
7 Nevada 8          12.2
Floride             15.4
## supprimer le tableau
> sqlDrop(canal, "USArrests")
## ferme la connexion
> odbcFermer (canal)
```

Comme exemple simple d'utilisation d'ODBC sous Windows avec une feuille de calcul Excel, on peut lire à partir d'une feuille de calcul par

```
> bibliothèque (RODBC)
> canal <- odbcConnectExcel("bdr.xls")
## lister les feuilles de calcul
> sqlTables (canal)
  TABLE_CAT TABLE_SCHEM                TABLE_NAME TABLE_TYPE REMARQUES
1 C:\\bdr          QUE          Feuille1$ TABLE DU SYSTÈME      QUE
2 C:\\bdr          QUE          Feuille2$ TABLE DU SYSTÈME      QUE
3 C:\\bdr          QUE          Fiche3$ TABLE DU SYSTÈME        QUE
4 C:\\bdr ##        NA Feuille1$Print_Area          TABLEAU      QUE
récupérer le contenu de la feuille 1, par l'un des
> sh1 <- sqlFetch(canal, "Feuil1")
> sh1 <- sqlQuery(canal, "select * from [Sheet1$]")
```

Notez que la spécification de la table est différente du nom renvoyé par sqlTables : sqlFetch est capable de cartographier les différences.

5 fichiers binaires

Les connexions binaires (Chapitre 7 [Connexions], page 22) sont désormais le moyen privilégié pour gérer les fichiers binaires.

5.1 Formats de données binaires

Packages h5 (https://CRAN.R-project.org/package=h5), rhdf5 de Bioconductor, RNetCDF (https://CRAN.R-project.org/package=RNetCDF) et ncdf4 (https://CRAN. R-project.org/package=ncdf4) sur CRAN fournissent des interfaces au HDF5 de la NASA (format de données hiérarchique, voir https://www.hdfgroup.org/HDF5/) et aux fichiers de données netCDF d'UCAR (formulaire de données commun du réseau, voir https ://www.unidata.ucar.edu/software/netcdf/).

Ces deux systèmes permettent de stocker des données scientifiques sous forme de tableaux, notamment des descriptions, des étiquettes, des formats, des unités, etc.. HDF5 autorise également des groupes de tableaux, et l'interface R mappe les listes aux groupes HDF5 et peut écrire des vecteurs et des matrices numériques et de caractères.

Le format version 4 de NetCDF (implémenté de manière confuse dans netCDF 4.1.1 et versions ultérieures, mais pas dans 4.0.1) inclut l'utilisation de divers formats HDF5. Ceci est géré par le package ncdf4 (https:// CRAN.R-project.org/package=ncdf4) tandis que RNetCDF (https://CRAN.R-project.org/ package=RNetCDF) gère les fichiers de la version 3.

La disponibilité des logiciels prenant en charge ces formats est quelque peu limitée par la plate-forme, en particulier sous Windows.

5.2 Fichiers dBase (DBF) dBase

était un programme DOS écrit par Ashton-Tate et appartenant plus tard à Borland. Il possède un format de fichier plat binaire devenu populaire, avec l'extension de fichier .dbf. Il a été adopté pour la famille de bases de données « Xbase », couvrant dBase, Clipper, FoxPro et leurs équivalents Windows Visual dBase, Visual Objects et Visual FoxPro (voir https://www.clicketyclick.dk/databases/xbase/format /). Un fichier dBase contient un en-tête puis une série de champs et ressemble donc beaucoup à une trame de données R. Les données elles-mêmes sont stockées au format texte et peuvent inclure des champs de caractères, logiques et numériques, ainsi que d'autres types dans les versions ultérieures (voir par exemple https://www.loc.gov/preservation/digital/formats/fdd/fdd000325.shtml et https://www.clicketyclick.dk/databases/xbase/format/index.html).

Les fonctions read.dbf et write.dbf fournissent des moyens de lire et d'écrire des fichiers DBF de base sur toutes les plateformes R. Pour les utilisateurs Windows, odbcConnectDbase dans le package RODBC (https://CRAN.R-project.org/package=RODBC) fournit des fonctionnalités plus complètes pour lire les fichiers DBF via le pilote ODBC dBase de Microsoft (et le pilote Visual FoxPro peut également être utilisé via odbcDriverConnect).

6 Fichiers images

Une classe particulière de fichiers binaires est celle qui représente des images, et une demande courante consiste à lire un tel fichier dans R sous forme de matrice.

Il existe de nombreux formats de fichiers image (la plupart avec de nombreuses variantes), et il peut être nécessaire d'utiliser un logiciel de conversion externe pour convertir d'abord l'image dans l'un des formats pour lesquels un package fournit actuellement un lecteur R. Un exemple polyvalent d'un tel logiciel est ImageMagick et son fork GraphicsMagick. Ceux-ci fournissent des programmes en ligne de commande convert et gm convert pour convertir des images d'un format à un autre : les formats qu'ils peuvent saisir sont déterminés lors de leur compilation, et les formats pris en charge peuvent être répertoriés par exemple par format convert -list.

Le package pixmap (https://CRAN.R-project.org/package=pixmap) a une fonction read.pnm pour lire les images 'portable anymap' en PBM (noir/blanc), PGM (gris) et PPM (couleur RVB) formats. Ceux-ci sont également connus sous le nom de formats « netpbm ».

Forfaits bmp (https://CRAN.R-project.org/package=bmp), jpeg (https://CRAN.
R-project.org/package=jpeg) et png (https://CRAN.R-project.org/package=png) lisent les formats d'après lesquels ils sont nommés. Voir également les packages biOps (https:// CRAN.R-project.org/package=biOps) et Momocs (https://CRAN.R-project.org/ package=Momocs) et le package Bioconductor EBImage.

TIFF est davantage un métaformat, un wrapper dans lequel une très grande variété de formats d'images peuvent être intégrés. Les packages rtiff (https://CRAN.R-project.org/package=rtiff) et tiff (https://CRAN.R-project.org/ package=tiff) peuvent lire certains des sous-formats (en fonction sur le logiciel libtiff externe avec lequel ils sont compilés). Il existe des fonctionnalités pour les sous-formats spécialisés, par exemple dans le package Bioconductor beadarray.

Les fichiers raster sont courants dans les sciences géographiques et le package rgdal (https://CRAN.
R-project.org/package=rgdal) fournit une interface à GDAL qui fournit ses propres fonctionnalités pour lire les fichiers raster et des liens vers de nombreux autres. Les formats qu'il prend en charge sont déterminés lorsque GDAL est compilé : utilisez gdalDrivers() pour voir de quoi il s'agit pour la version que vous utilisez. Il peut être utile pour les formats peu courants tels que JPEG 2000 (qui est un format différent de JPEG et qui n'est actuellement pas pris en charge dans les versions binaires macOS ni Windows de rgdal (https://CRAN.R-project.org/package=rgdal)).

7 connexions

Les connexions sont utilisées dans R au sens de Chambers (1998) et Ripley (2001), un ensemble de fonctions pour remplacer l'utilisation de noms de fichiers par une interface flexible vers des objets de type fichier.

7.1 Types de connexions

Le type de connexion le plus familier sera un fichier, et les connexions de fichiers sont créées par un fichier de fonction. Les connexions de fichiers peuvent (si le système d'exploitation le permet pour le fichier particulier) être ouvertes en lecture, en écriture ou en ajout, en mode texte ou binaire. En fait, les fichiers peuvent être ouverts à la fois en lecture et en écriture, et R conserve une position de fichier distincte pour la lecture et l'écriture.

Notez que par défaut une connexion n'est pas ouverte lors de sa création. La règle est qu'une fonction utilisant une connexion doit ouvrir une connexion (nécessaire) si la connexion n'est pas déjà ouverte, et fermer une connexion après utilisation si elle l'a ouverte. En bref, laissez la connexion dans l'état dans lequel vous l'avez trouvée. Il existe des fonctions génériques d'ouverture et de fermeture avec des méthodes pour ouvrir et fermer explicitement les connexions.

Les fichiers compressés via l'algorithme utilisé par gzip peuvent être utilisés comme connexions créées par le fonction gzfile, alors que les fichiers compressés par bzip2 peuvent être utilisés via bzfile.

Les programmeurs Unix sont habitués à gérer des fichiers spéciaux stdin, stdout et stderr. Ceux-ci existent sous forme de connexions de terminal dans R. Il peut s'agir de fichiers normaux, mais ils peuvent également faire référence à des entrées et des sorties vers une console GUI. (Même avec l'interface standard Unix R, stdin fait référence aux lignes soumises depuis readline plutôt qu'à un fichier.)

Les trois connexions des bornes sont toujours ouvertes et ne peuvent pas être ouvertes ou fermées. stdout et stderr sont classiquement utilisés respectivement pour la sortie normale et les messages d'erreur. Ils peuvent normalement aller au même endroit, mais alors que la sortie normale peut être redirigée par un appel à Sink, la sortie d'erreur est envoyée à stderr à moins qu'elle ne soit redirigée par Sink, type="message"). Notez attentivement le langage utilisé ici : les connexions ne peuvent pas être redirigées, mais la sortie peut être envoyée vers d'autres connexions.

Les connexions textuelles sont une autre source d'entrée. Ils permettent de lire les vecteurs de caractères R comme si les lignes étaient lues à partir d'un fichier texte. Une connexion texte est créée et ouverte par un appel à textConnection, qui copie le contenu actuel du vecteur de caractères dans un tampon interne au moment de la création.

Les connexions de texte peuvent également être utilisées pour capturer la sortie R dans un vecteur de caractères. textConnection peut être invité à créer un nouvel objet personnage ou à l'ajouter à un objet existant, dans les deux cas dans l'espace de travail de l'utilisateur. La connexion est ouverte par l'appel à textConnection, et à tout moment les lignes complètes sorties vers la connexion sont disponibles dans l'objet R. La fermeture de la connexion écrit toute sortie restante dans un élément final du vecteur de caractères.

Les tuyaux sont une forme spéciale de fichier qui se connecte à un autre processus, et les connexions de tuyaux sont créées par la fonction pipe. L'ouverture d'une connexion de canal pour l'écriture (cela n'a aucun sens de l'ajouter à un canal) exécute une commande du système d'exploitation et connecte son entrée standard à tout ce que R écrit ensuite sur cette connexion. À l'inverse, l'ouverture d'une connexion de canal pour l'entrée exécute une commande du système d'exploitation et rend sa sortie standard disponible pour l'entrée R à partir de cette connexion.

Les URL de types « http:// », « https:// », « ftp:// » et « file:// » peuvent être lues à partir de la fonction url. Pour plus de commodité, file les acceptera également comme spécification de fichier et URL d'appel.

Les sockets peuvent également être utilisées comme connexions via la fonction socketConnection sur les plates-formes prenant en charge les sockets de type Berkeley (la plupart des systèmes Unix, Linux et Windows). Les sockets peuvent être écrits ou lus, et les sockets client et serveur peuvent être utilisés.

7.2 Sortie vers les connexions

Nous avons décrit les fonctions cat, write, write.table et Sink comme écrivant dans un fichier, éventuellement ajoutées à un fichier si l'argument append = TRUE, et c'est ce qu'elles faisaient avant la version R 1.2.0.

Le comportement actuel est équivalent, mais ce qui se passe réellement, c'est que lorsque l'argument fichier est une chaîne de caractères, une connexion fichier est ouverte (pour écriture ou ajout) et fermée à nouveau à la fin de l'appel de fonction. Si nous voulons écrire à plusieurs reprises dans le même fichier, il est plus efficace de déclarer et d'ouvrir explicitement la connexion, et de transmettre l'objet de connexion à chaque appel à une fonction de sortie. Cela permet également d'écrire dans des tubes, ce qui était implémenté auparavant de manière limitée via le fichier de syntaxe = "| cmd" (qui peut encore être utilisé).

Il existe une fonction writeLines pour écrire des lignes de texte complètes dans une connexion.

Quelques exemples simples sont

```
zz <- file("ex.data", "w") # ouvrir une connexion de fichier de sortie cat("TITLE extra line", "2
3 5 7", "", "11 13 17", file = zz, sep = "\n")

cat("Une ligne de plus\n", file = zz) close(zz)


## convertir le point décimal en virgule dans la sortie, en utilisant un tube (Unix) ## les deux
chaînes R et (probablement) le shell ont besoin de \ doublé zz <- pipe(paste("sed s/\\\
\./,/ > ", "outfile"), "w") cat(format(round(rnorm(100), 4)), sep = "\n", file = zz)
close(zz) ## regardez maintenant le fichier de sortie : fichier.show("outfile", delete.file
= TRUE)



## capture la sortie R : utilisez les exemples de help(lm) zz <-
textConnection("ex.lm.out", "w") Sink(zz) example(lm,
prompt.echo
= "> ") Sink() close (zz) ## maintenant
'ex.lm.out'
contient la
sortie pour un traitement ultérieur.
## Regardez-le par, par exemple,
cat(ex.lm.out, sep = "\n")
```

7.3 Entrée depuis les connexions

Les fonctions de base pour lire à partir des connexions sont scan et readLines. Ceux-ci prennent un argument de chaîne de caractères et ouvrent une connexion de fichier pendant la durée de l'appel de fonction, mais l'ouverture explicite d'une connexion de fichier permet à un fichier d'être lu séquentiellement dans différents formats.

D'autres fonctions qui appellent scan peuvent également utiliser des connexions, notamment read.table.

Quelques exemples simples sont

```
## lecture du fichier créé dans les derniers exemples
readLines("ex.data")
unlink("ex.data")

## lire la liste du répertoire courant (Unix) readLines(pipe("ls -1"))
```

```
# supprime les virgules de fin d'un fichier d'entrée.
# Supposons que nous recevions un fichier 'data' contenant 450, 390,
467, 654, 30, 542, 334, 432, 421, 357, 497, 493, 550, 549, 467,
575, 578, 342, 446, 547 , 534, 495, 979, 479 # Ensuite, lisez ceci
en scan(pipe("sed -es/,$// data"), sep=",")
```

Pour plus de commodité, si l'argument file spécifie une URL FTP, HTTP ou HTTPS , l' URL est ouvert en lecture via l'URL. La spécification de fichiers via 'file://foo.bar' est également autorisée.

7.3.1 Repoussage

Les programmeurs C sont peut-être familiers avec la fonction ungetc pour repousser un caractère dans un flux de saisie de texte. Les connexions R ont la même idée de manière plus puissante, dans la mesure où un nombre (essentiellement) arbitraire de lignes de texte peut être repoussé sur une connexion via un appel à pushBack.

Les refoulements fonctionnent comme une pile, donc une demande de lecture utilise d'abord chaque ligne du texte refoulé le plus récemment, puis celles des refoulements précédents et enfin lit à partir de la connexion elle-même. Une fois qu'une ligne repoussée est lue complètement, elle est effacée. Le nombre de lignes en attente repoussées peut être trouvé via un appel à pushBackLength.

Un exemple simple montrera l'idée.

```
> zz <- textConnection(LETTRES) >
readLines(zz, 2)
 [1] "A" "B" >
scanner(zz, "", 4)
Lire 4 articles
 [1] "C" "D" "E" "F" >
pushBack(c("aa", "bb"), zz) > scan(zz, "", 4)

Lire 4 articles
 [1] "aa" "bb" "G" "H" > fermer(zz)
```

Le pushback n'est disponible que pour les connexions ouvertes pour la saisie en mode texte.

7.4 Lister et manipuler les connexions

Un résumé de toutes les connexions actuellement ouvertes par l'utilisateur peut être trouvé par showConnections(), et un résumé de toutes les connexions, y compris les connexions fermées et terminales, par showConnections(all = TRUE)

La fonction générique seek peut être utilisée pour lire et (sur certaines connexions) réinitialiser la position actuelle en lecture ou en écriture. Malheureusement, cela dépend des fonctionnalités du système d'exploitation qui peuvent ne pas être fiables (par exemple avec des fichiers texte sous Windows). La fonction isSeekable signale si la recherche peut changer la position sur la connexion donnée par son argument.

La fonction truncate peut être utilisée pour tronquer un fichier ouvert en écriture à sa position actuelle. Il ne fonctionne que pour les connexions de fichiers et n'est pas implémenté sur toutes les plateformes.

7.5 Connexions binaires Les fonctions

readBin et writeBin lisent et écrivent à partir de connexions binaires. Une connexion est ouverte en mode binaire en ajoutant « b » à la spécification du mode, c'est-à-dire en utilisant le mode « rb » pour la lecture et le mode « wb » ou « ab » (le cas échéant) pour l'écriture. Les fonctions ont des arguments readBin(con, what, n = 1, size = NA, endian = .Platform$endian) writeBin(object, con, size = NA, endian = .Platform$endian)

Dans chaque cas con est une connexion qui sera ouverte si nécessaire pendant la durée du appel, et si une chaîne de caractères est donnée, elle est supposée spécifier un nom de fichier.

Il est légèrement plus simple de décrire l'écriture, nous allons donc le faire en premier. L'objet doit être un objet vectoriel atomique, c'est-à-dire un vecteur de mode numérique, entier, logique, caractère, complexe ou brut, sans attributs. Par défaut, cela est écrit dans le fichier sous forme de flux d'octets exactement tel qu'il est représenté en mémoire.

readBin lit un flux d'octets du fichier et les interprète comme un vecteur de mode donné par quoi. Cela peut être soit un objet du mode approprié (par exemple what=integer()), soit une chaîne de caractères décrivant le mode (un des cinq donnés dans le paragraphe précédent ou "double" ou "int"). L'argument n spécifie le nombre maximum d'éléments vectoriels à lire à partir de la connexion : s'il y en a moins, un vecteur plus court sera renvoyé. L'argument signé permet de lire des entiers de 1 et 2 octets comme des entiers signés (par défaut) ou non signés.

Les deux arguments restants sont utilisés pour écrire ou lire des données à échanger avec un autre programme ou une autre plateforme. Par défaut, les données binaires sont transférées directement de la mémoire vers la connexion ou vice versa. Cela ne suffira pas si les données doivent être transférées vers une machine avec une architecture différente, mais entre presque toutes les plates-formes R, le seul changement nécessaire est celui de l'ordre des octets. Les PC courants (machines basées sur 'ix86' et 'x86_64'), Compaq Alpha et Vaxen sont en petit-boutiste, alors que Sun Sparc, la série mc680x0, IBM R6000, SGI et la plupart des autres sont en gros-boutiste. (L'ordre des octets du réseau (tel qu'utilisé par XDR, eXternal Data Representation) est big-endian.) Pour transférer vers ou depuis d'autres programmes, nous devrons peut-être en faire plus, par exemple pour lire des entiers de 16 bits ou écrire des nombres réels en simple précision. . Cela peut être fait en utilisant l'argument size, qui autorise (généralement) les tailles 1, 2, 4, 8 pour les entiers et les logiques, et les tailles 4, 8 et peut-être 12 ou 16 pour les réels. Le transfert à différentes tailles peut perdre en précision et ne doit pas être tenté pour les vecteurs contenant des NA.

Les chaînes de caractères sont lues et écrites au format C, c'est-à-dire sous la forme d'une chaîne d'octets terminée par un octet zéro. Les fonctions readChar et writeChar offrent une plus grande flexibilité.

7.5.1 Valeurs spéciales Les

fonctions readBin et writeBin transmettront les valeurs manquantes et spéciales, bien que cela ne doive pas être tenté si un changement de taille est impliqué.

La valeur manquante pour les types logiques et entiers R est INT_MIN, le plus petit entier représentable défini dans l'en-tête C limit.h, correspondant normalement au modèle binaire 0x80000000.

La représentation des valeurs spéciales pour les types numériques et complexes R dépend de la machine, et éventuellement aussi du compilateur. Le moyen le plus simple de les utiliser est de lier une application externe à la bibliothèque autonome Rmath qui exporte les doubles constantes NA_REAL, R_PosInf et R_NegInf, et d'inclure l'en-tête Rmath.h qui définit les macros ISNAN et R_FINITE.

Si cela n'est pas possible, sur toutes les plates-formes actuelles, l'arithmétique CEI 60559 (alias IEEE 754) est utilisée, de sorte que les fonctionnalités standard C peuvent être utilisées pour tester ou définir les valeurs Inf, -Inf et NaN. Sur de telles plates-formes, NA est représenté par la valeur NaN avec le mot faible 0x7a2 (1954 en décimal).

Les valeurs de caractères manquantes sont écrites sous la forme NA, et il n'existe aucune disposition permettant de reconnaître les valeurs de caractères comme manquantes (car cela peut être fait en les réattribuant une fois lues).

8 interfaces réseau

Certaines fonctionnalités limitées sont disponibles pour échanger des données à un niveau inférieur sur les connexions réseau.

8.1 Lecture depuis les sockets

Base R est livré avec certaines fonctionnalités permettant de communiquer via des sockets BSD sur les systèmes qui les prennent en charge (y compris les ports Linux, Unix et Windows communs de R). Un problème potentiel lié à l'utilisation des sockets est que ces fonctionnalités sont souvent bloquées pour des raisons de sécurité ou pour forcer l'utilisation de caches Web. Ces fonctions peuvent donc être plus utiles sur un intranet qu'en externe. Pour les nouveaux projets, il est suggéré d'utiliser plutôt des connexions socket.

L'interface de bas niveau précédente est donnée par les fonctions make.socket, read.socket, write.socket et close.socket.

8.2 Utilisation du fichier download.file

La fonction download.file est fournie pour lire un fichier à partir d'une ressource Web via FTP ou HTTP (y compris HTTPS) et l'écrire dans un fichier. Cela peut souvent être évité, car des fonctions telles que read.table et scan peuvent lire directement à partir d'une URL, soit en utilisant explicitement l'url pour ouvrir une connexion, soit en l'utilisant implicitement en donnant une URL comme argument de fichier.

9 Lire des feuilles de calcul Excel

La question d'importation/exportation de données R la plus courante semble être « comment lire une feuille de calcul Excel ». Ce chapitre rassemble les conseils et les options donnés précédemment. Notez que la plupart des conseils concernent les feuilles de calcul antérieures à Excel 2007 et non le format .xlsx ultérieur.

Le premier conseil est d'éviter de le faire si possible ! Si vous avez accès à Excel, exportez les données souhaitées depuis Excel sous forme délimitée par des tabulations ou séparées par des virgules, et utilisez read.delim ou read.csv pour les importer dans R. (Vous devrez peut-être utiliser read.delim2 ou read .csv2 dans des paramètres régionaux qui utilisent la virgule comme point décimal.) L'exportation d'un fichier DIF et sa lecture à l'aide de read.DIF sont une autre possibilité.

Si vous n'avez pas Excel, de nombreux autres programmes sont capables de lire de telles feuilles de calcul et de les exporter au format texte sous Windows et Unix, par exemple Gnumeric (http://www.gnumeric. org) et OpenOffice (https:// www. .openoffice.org). Vous pouvez également faire un copier-coller entre l'affichage d'un tableur dans un tel programme et R : read.table lira depuis la console R ou, sous Windows, depuis le presse-papier (via file = "clipboard" ou readClipboard) . La fonction read.DIF peut également lire à partir du presse-papiers.

Notez qu'un fichier Excel .xls n'est pas qu'une simple feuille de calcul : ces fichiers peuvent contenir de nombreuses feuilles, et les feuilles peuvent contenir des formules, des macros, etc. Tous les lecteurs ne peuvent pas lire autre chose que la première feuille et peuvent être déroutés par d'autres contenus du fichier.

Les utilisateurs Windows (de R 32 bits) peuvent utiliser odbcConnectExcel dans le package RODBC (https://CRAN.R-project.org/package=RODBC). Cela peut sélectionner des lignes et des colonnes de n'importe quelle feuille d'un fichier de feuille de calcul Excel (au moins à partir d'Excel 97–2003, en fonction de vos pilotes ODBC : en appelant directement odbcConnect, les versions vers Excel 3.0 peuvent être lues). La version odbcConnectExcel2007 lira les formats Excel 2007 ainsi que les formats antérieurs (à condition que les pilotes soient installés, y compris avec Windows R 64 bits : voir Section 4.3.2 [RODBC], page 18). Les utilisateurs de macOS peuvent également utiliser RODBC (https://CRAN.R-project.org/package=RODBC) s'ils disposent d'un pilote approprié (par exemple celui d'Actual Technologies).

Les utilisateurs de Perl ont contribué à un module OLE :: SpreadSheet :: ParseExcel et à un programme xls2csv.pl pour convertir les feuilles de calcul Excel 95-2003 en fichiers CSV. Le package gdata (https://CRAN.R-project.org/ package=gdata) fournit un wrapper de base dans sa fonction read.xls. Avec les modules Perl appropriés installés, cette fonction peut également lire les feuilles de calcul Excel 2007.

Les packages dataframes2xls (https://CRAN.R-project.org/package=dataframes2xls) et WriteXLS (https://CRAN.R-project.org/package=WriteXLS) contiennent chacun une fonction permettant d'écrire une ou plusieurs trames de données dans un fichier .xls, utilisant respectivement Python et Perl.

Le package xlsx (https://CRAN.R-project.org/package=xlsx) peut lire et manipuler des feuilles de calcul Ex-cel 2007 et versions ultérieures : il nécessite Java.

Le package XLConnect (https://CRAN.R-project.org/package=XLConnect) peut lire, écrire et manipuler les feuilles de calcul Excel 97-2003 et Excel 2007/10 à l'aide de Java.

Le package readxl (https://CRAN.R-project.org/package=readxl) peut lire les feuilles de calcul Excel 97-2003 et Excel 2007/10, à l'aide d'une bibliothèque C incluse.

Annexe A Références

RA Becker, JM Chambers et AR Wilks (1988) Le nouveau langage S. Un environnement de programmation pour l'analyse des données et les graphiques. Wadsworth et Brooks/Cole.

J. Bowman, S. Emberson et M. Darnovsky (1996) Le manuel pratique de SQL . Utilisation d'un langage de requête structuré. Addison-Wesley.

JM Chambers (1998) Programmation avec des données. Un guide du langage S. Springer-Verlag.

P. Dubois (2000) MySQL. Nouveaux cavaliers.

M. Henning et S. Vinoski (1999) Programmation CORBA avancée avec C++. Addison-Wesley.

K. Kline et D. Kline (2001) SQL en bref. O'Reilly.

B. Momjian (2000) PostgreSQL : introduction et concepts. Addison-Wesley. Également disponible sur https://momjian.us/main/writings/pgsql/aw_pgsql_book/.

BD Ripley (2001) Connexions. R Actualités, 1/1, 16-7. https://www.r-project.org/doc/ Rnews/Rnews_2001-1.pdf TM Therneau et PM Grambsch

(2000) Modélisation des données de survie. Extension du modèle Cox.
Éditions Springer.

EJ Yarger, G. Reese et T. King (1999) MySQL et mSQL. O'Reilly.

Fonction et index variable

T

DANS

DANS

X

Index des concepts