Superviser un étudiant dans son travail de recherche

jcb

10 février 2015

Contents

1 Introduction **1**

2 Organiser son travail **1**

3 Saisir les données **2**

3.1 Anatomie d'un tableur 2

4 Divers **2**

4.1 Organiser le versionning 2

4.2 Les données manquantes 2

5 Data **2**

5.1 Données ordonnées (Tidy data) 2

5.2 Autre référence 5

5.3 Commentaires sur article de Wicham 13

6 Tidyr **13**

7 Alternatives **13**

8 Divers **13**

1 Introduction

A summary of the evidence that most published research is false

2 Organiser son travail

Stafford Noble W. A quick Guide to Organizing Computational Biology Projects. PLOS Comp. Biol. (2009) **5**(7). Accédé le 10/2/2015.

Un script pour implémenter automatiquement la structure proposée dans l'article de Stafford Noble.

Structuring a Data Analysis using R (Part 1 of 2) – Gathering, Organizing, Exploring and Munging Data
Structuring a Data Analysis using R (Part 2 of 2) – Analyzing, Modeling, and the Write-up

3 Saisir les données

Faut-il utiliser un tableur ou un logiciel de statistiques pour saisir les données ? Luis A. Apiolaza

3.1 Anatomie d'un tableur

Manuel Genstat

4 Divers

4.1 Organiser le versionning

Git et Github.

Git-les bases pourbbien gérer les versions de votre projet. Linux Pratique (2013) n°83 59-63.

Représenter les données: notion UTF8

Une des premières tâches dans un projet de recherche est de savoir comment recueillir et stocker ses données de manière intelligible et efficiente: - intelligible: comprendre de ce que l'on a fait plusieurs mois plus tard - efficiente: 80% du temps nécessaire au traitement statistique des données est consacré à nettoyer les données (#ref)

Règle 1: le support de saisie doit servir uniquement à cette tâche et rien d'autre.

Les données nettoyées: ce sont typiquement celles que l'on trouve dans les exemples des ouvrages de référence. Elles sont "parfaites" en ce sens qu"elles peuvent être importée dans un logiciel de statistique et traitées immédiatement sans que le progrmme de génère une erreur. L'objectf est que vos donnnées soient"parfaites". Hélas dans la vraie vie rien n'est parfait et il est rare q'un logiciels accepte sans broncher vos données.

4.2 Les données manquantes

Des données peuvent mnquer pour de nombreuses raisons. La plupart des logiciels acceptent le symbole NA (pour not avalaible) pour désigner une valeur manquante. Pour obtenir un vecteur de valeurs non manquantes: x[!is.na(x)].

Référence:

Sharing Clinical Research Data: Workshop Summary. Accédé le 7/2/2015

5 Data

Hadley Wickham Tidy Data

Une version alégée de cet article figure ci-après:

5.1 Données ordonnées (Tidy data)

Ce sont les tableaux de données que l'on touve dans les ouvrages de statistiques. Ils ont prêts à l'emploi, sans ereurs, sans données manquantes. Soumises à un logiciel de statistiques, elle produiront instantannément un résultat sans un arrêt intempestif du programme assorti de message d'erreur plus ou moins abscons.

Les données brutes (Raw data) dans le monde réel sont souvent désordonnés et mal formatées (Messy data). En outre, il peut manquer de détails appropriés de l'étude. La correction des données natives peut être un exercice périlleux puisque les données brutes originales risque d'être écrasées et il n'y aurait aucun moyen de vérifier ce processus ou de récupérer des erreurs commises pendant cette phase. Une bonne pratique serait de conserver les données d'origine, et d'utiliser un script programmatique pour nettoyer, corriger les erreurs et sauvegarder cet ensemble de données nettoyées pour une analyse ultérieure.

Question: combien de variable dans le tableau suivant:

lésions	hommes	femmes
oui	4	1
non	2	5

La façon dont le tableau est présenté, il semble que il ya seulement deux variables. La bonne réponse est trois: lésions?, sexe, nombre.

sujet	lésion	sexe	nombre
1	oui	H	4
2	oui	F	1
3	non	H	2
4	non	F	5

Comment définir des données bien ordonnées ? Ce sont des données répondant aux 3 caractéristiques suivantes:

- les observations (sujets) sont en lignes
- les variables sont en colonnes
- toutes les données sont dans le même tableau

Ce format est également appelé large (wide) et s'oppose au format long (long). La plupart des logiciels ont des instructions permettant de passer d'un format à l'autre (ex. format pivot dans excel).

Les données bien ordonnées facilitent l'analyse statistiques des données.

Les erreurs les plus fréquentes:

Répartition des revenus par Religion

Notre premier ensemble de données est basée sur une enquête réalisée par Pew Research qui examine la relation entre le revenu et l'appartenance religieuse.

```
religion  <$10k $10-20k $20-30k $30-40k $40-50k $50-75k $75-100k    $100-150k   >150k   Don't know/refus
1   Agnostic    27  34  60  81  76  137 122 109 84  96
2   Atheist 12  27  37  52  35  70  73  59  74  76
3   Buddhist    27  21  30  34  33  58  62  39  53  54
4   Catholic    418 617 732 670 638 1116    949 792 633 1489
5   Don't know/refused  15  14  15  11  10  35  21  17  18  116
6   Evangelical Prot    575 869 1064    982 881 1486    949 723 414 1529
7   Hindu   1   9   7   9   11  34  47  48  54  37
```

```
8   Historically Black Prot 228 244 236 238 197 223 131 81  78  339
9   Jehovah's Witness   20  27  24  24  21  30  15  11  6   37
10  Jewish  19  19  25  25  30  95  69  87  151 162
11  Mainline Prot   289 495 619 655 651 1107    939 753 634 1328
12  Mormon  29  40  48  51  56  112 85  49  42  69
13  Muslim  6   7   9   10  9   23  16  8   6   22
14  Orthodox    13  17  23  32  32  47  38  42  46  73
15  Other Christian 9   7   11  13  13  14  18  14  12  18
16  Other Faiths    20  33  40  46  49  63  46  40  41  71
17  Other World Religions   5   2   3   4   2   7   3   4   4   8
18  Unaffiliated    217 299 374 365 341 528 407 321 258 597
```

Tuberculose

```
##   iso2 year new_sp new_sp_m04 new_sp_m514 new_sp_m014 new_sp_m1524
## 1   AD 1989     NA         NA          NA          NA           NA
## 2   AD 1990     NA         NA          NA          NA           NA
## 3   AD 1991     NA         NA          NA          NA           NA
## 4   AD 1992     NA         NA          NA          NA           NA
## 5   AD 1993     15         NA          NA          NA           NA
## 6   AD 1994     24         NA          NA          NA           NA
##   new_sp_m2534 new_sp_m3544 new_sp_m4554 new_sp_m5564 new_sp_m65 new_sp_mu
## 1           NA           NA           NA           NA         NA        NA
## 2           NA           NA           NA           NA         NA        NA
## 3           NA           NA           NA           NA         NA        NA
## 4           NA           NA           NA           NA         NA        NA
## 5           NA           NA           NA           NA         NA        NA
## 6           NA           NA           NA           NA         NA        NA
##   new_sp_f04 new_sp_f514 new_sp_f014 new_sp_f1524 new_sp_f2534
## 1         NA          NA          NA           NA           NA
## 2         NA          NA          NA           NA           NA
## 3         NA          NA          NA           NA           NA
## 4         NA          NA          NA           NA           NA
## 5         NA          NA          NA           NA           NA
## 6         NA          NA          NA           NA           NA
##   new_sp_f3544 new_sp_f4554 new_sp_f5564 new_sp_f65 new_sp_fu
## 1           NA           NA           NA         NA        NA
## 2           NA           NA           NA         NA        NA
## 3           NA           NA           NA         NA        NA
## 4           NA           NA           NA         NA        NA
## 5           NA           NA           NA         NA        NA
## 6           NA           NA           NA         NA        NA
```

Sauf pour ISO2 et l'année, le reste des en-têtes de colonnes sont en fait des valeurs d'une variable qui se cache, en fait la combinaison de deux variables, le sexe et l'âge.

Le temps

```
##            id year month element d1  d2  d3 d4  d5 d6 d7 d8 d9 d10 d11 d12
## 1 MX000017004 2010     1    TMAX NA  NA  NA NA  NA NA NA NA NA  NA  NA  NA
## 2 MX000017004 2010     1    TMIN NA  NA  NA NA  NA NA NA NA NA  NA  NA  NA
## 3 MX000017004 2010     2    TMAX NA 273 241 NA  NA NA NA NA NA  NA 297  NA
## 4 MX000017004 2010     2    TMIN NA 144 144 NA  NA NA NA NA NA  NA 134  NA
## 5 MX000017004 2010     3    TMAX NA  NA  NA NA 321 NA NA NA NA 345  NA  NA
```

```
## 6 MX000017004 2010     3    TMIN NA  NA  NA NA 142 NA NA NA NA 168  NA  NA
##   d13 d14 d15 d16 d17 d18 d19 d20 d21 d22 d23 d24 d25 d26 d27 d28 d29 d30
## 1  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA 278
## 2  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA 145
## 3  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA 299  NA  NA  NA  NA  NA  NA  NA
## 4  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA 107  NA  NA  NA  NA  NA  NA  NA
## 5  NA  NA  NA 311  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA
## 6  NA  NA  NA 176  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA
##   d31
## 1  NA
## 2  NA
## 3  NA
## 4  NA
## 5  NA
## 6  NA
```

Cet ensemble de données semble avoir deux problèmes. Premièrement, il a variables dans les lignes de l'élément de colonne. Deuxièmement, il a un d variable dans l'en-tête de colonne répartis sur plusieurs colonnes.

Les erreurs les plus fréquentes:

- les entêtes de colonnes sont des valeurs, pas les noms de variables
- Plusieurs variables sont stockées dans une colonne
- Les variables sont stockées dans les lignes et les colonnes
- Plusieurs types de unité expérimentale stockés dans la même table
- Un type d'appareil d'essai stockées dans plusieurs tables

suite

Rmodeler (reshaping)

Une façon de données est bien rangé pour remodeler pour qu'il adhère aux trois règles de données bien rangé. Bien que la base R a plusieurs fonctions visant à remodeler les données, nous allons utiliser le package reshape2 par Hadley Wickham, car il fournit un ensemble simple et cohérente des fonctions pour remodeler données. Notions de base

En termes simples, le remodelage des données est comme faire un tableau croisé dynamique (pivot) dans Excel, où vous mélangez colonnes, des lignes et des valeurs. Commençons par nous ranger l'ensemble de données de pew.

We can tidy this data using the melt function in the reshape2 package.

```
library(reshape2)
pew_tidy <- melt(
  data = pew,
  id = "religion",
  variable.name = "income",
  value.name = "frequency"
)
```

5.2 Autre référence

Data tidying

It is often said that 80% of data analysis is spent on the cleaning and preparing data. And it's not just a first step, but it must be repeated many over the course of analysis as new problems come to light or new data is collected. To get a handle on the problem, this paper focuses on a small, but important, aspect of data cleaning that I call data tidying: structuring datasets to facilitate analysis.

The principles of tidy data provide a standard way to organise data values within a dataset. A standard makes initial data cleaning easier because you don't need to start from scratch and reinvent the wheel every time. The tidy data standard has been designed to facilitate initial exploration and analysis of the data, and to simplify the development of data analysis tools that work well together. Current tools often require translation. You have to spend time munging the output from one tool so you can input it into another. Tidy datasets and tidy tools work hand in hand to make data analysis easier, allowing you to focus on the interesting domain problem, not on the uninteresting logistics of data. Defining tidy data {#sec:defining}

```
Happy families are all alike; every unhappy family is unhappy in its own way - Leo Tolstoy
```

Like families, tidy datasets are all alike but every messy dataset is messy in its own way. Tidy datasets provide a standardized way to link the structure of a dataset (its physical layout) with its semantics (its meaning). In this section, I'll provide some standard vocabulary for describing the structure and semantics of a dataset, and then use those definitions to define tidy data. Data structure

Most statistical datasets are data frames made up of rows and columns. The columns are almost always labeled and the rows are sometimes labeled. The following code provides some data about an imaginary experiment in a format commonly seen in the wild. The table has two columns and three rows, and both rows and columns are labeled.

preg <- read.csv("preg.csv", stringsAsFactors = FALSE) preg #> name treatmenta treatmentb #> 1 John Smith NA 18 #> 2 Jane Doe 4 1 #> 3 Mary Johnson 6 7

There are many ways to structure the same underlying data. The following table shows the same data as above, but the rows and columns have been transposed.

read.csv("preg2.csv", stringsAsFactors = FALSE) #> treatment John.Smith Jane.Doe Mary.Johnson #> 1 a NA 4 6 #> 2 b 18 1 7

The data is the same, but the layout is different. Our vocabulary of rows and columns is simply not rich enough to describe why the two tables represent the same data. In addition to appearance, we need a way to describe the underlying semantics, or meaning, of the values displayed in table. Data semantics

A dataset is a collection of values, usually either numbers (if quantitative) or strings (if qualitative). Values are organised in two ways. Every value belongs to a variable and an observation. A variable contains all values that measure the same underlying attribute (like height, temperature, duration) across units. An observation contains all values measured on the same unit (like a person, or a day, or a race) across attributes.

A tidy version of the pregnancy data looks like this: (you'll learn how the functions work a little later)

library(tidyr) library(dplyr) preg2 <- preg %>% gather(treatment, n, treatmenta:treatmentb) %>% mutate(treatment = gsub("treatment", "", treatment)) %>% arrange(name, treatment) preg2 #> name treatment n #> 1 Jane Doe a 4 #> 2 Jane Doe b 1 #> 3 John Smith a NA #> 4 John Smith b 18 #> 5 Mary Johnson a 6 #> 6 Mary Johnson b 7

This makes the values, variables and observations more clear. The dataset contains 18 values representing three variables and six observations. The variables are:

```
name, with three possible values (John, Mary, and Jane).

treatment, with two possible values (a and b).

n, with five or six values depending on how you think of the missing value (1, 4, 6, 7, 18, NA)
```

The experimental design tells us more about the structure of the observations. In this experiment, every combination of of person and treatment was measured, a completely crossed design. The experimental design also determines whether or not missing values can be safely dropped. In this experiment, the missing value represents an observation that should have been made, but wasn't, so it's important to keep it. Structural missing values, which represent measurements that can't be made (e.g., the count of pregnant males) can be safely removed.

For a given dataset, it's usually easy to figure out what are observations and what are variables, but it is surprisingly difficult to precisely define variables and observations in general. For example, if the columns in the pregnancy data were height and weight we would have been happy to call them variables. If the columns were height and width, it would be less clear cut, as we might think of height and width as values of a dimension variable. If the columns were home phone and work phone, we could treat these as two variables, but in a fraud detection environment we might want variables phone number and number type because the use of one phone number for multiple people might suggest fraud. A general rule of thumb is that it is easier to describe functional relationships between variables (e.g., z is a linear combination of x and y, density is the ratio of weight to volume) than between rows, and it is easier to make comparisons between groups of observations (e.g., average of group a vs. average of group b) than between groups of columns.

In a given analysis, there may be multiple levels of observation. For example, in a trial of new allergy medication we might have three observational types: demographic data collected from each person (age, sex, race), medical data collected from each person on each day (number of sneezes, redness of eyes), and meteorological data collected on each day (temperature, pollen count).

Variables may change over the course of analysis. Often the variables in the raw data are very fine grained, and may add extra modelling complexity for little explanatory gain. For example, many surveys ask variations on the same question to better get at an underlying trait. In early stages of analysis, variables correspond to questions. In later stages, you change focus to traits, computed by averaging together multiple questions. This considerably simplifies analysis because you don't need a hierarchical model, and you can often pretend that the data is continuous, not discrete. Tidy data

Tidy data is a standard way of mapping the meaning of a dataset to its structure. A dataset is messy or tidy depending on how rows, columns and tables are matched up with observations, variables and types. In tidy data:

```
Each variable forms a column.

Each observation forms a row.

Each type of observational unit forms a table.
```

This is Codd's 3rd normal form, but with the constraints framed in statistical language, and the focus put on a single dataset rather than the many connected datasets common in relational databases. Messy data is any other other arrangement of the data.

Tidy data makes it easy for an analyst or a computer to extract needed variables because it provides a standard way of structuring a dataset. Compare the different versions of the pregnancy data: in the messy version you need to use different strategies to extract different variables. This slows analysis and invites errors. If you consider how many data analysis operations involve all of the values in a variable (every aggregation function), you can see how important it is to extract these values in a simple, standard way. Tidy data is particularly well suited for vectorised programming languages like R, because the layout ensures that values of different variables from the same observation are always paired.

While the order of variables and observations does not affect analysis, a good ordering makes it easier to scan the raw values. One way of organising variables is by their role in the analysis: are values fixed by the design of the data collection, or are they measured during the course of the experiment? Fixed variables describe the experimental design and are known in advance. Computer scientists often call fixed variables dimensions, and statisticians usually denote them with subscripts on random variables. Measured variables are what

we actually measure in the study. Fixed variables should come first, followed by measured variables, each ordered so that related variables are contiguous. Rows can then be ordered by the first variable, breaking ties with the second and subsequent (fixed) variables. This is the convention adopted by all tabular displays in this paper. Tidying messy datasets {#sec:tidying}

Real datasets can, and often do, violate the three precepts of tidy data in almost every way imaginable. While occasionally you do get a dataset that you can start analysing immediately, this is the exception, not the rule. This section describes the five most common problems with messy datasets, along with their remedies:

```
Column headers are values, not variable names.

Multiple variables are stored in one column.

Variables are stored in both rows and columns.

Multiple types of observational units are stored in the same table.

A single observational unit is stored in multiple tables.
```

Surprisingly, most messy datasets, including types of messiness not explicitly described above, can be tidied with a small set of tools: gathering, separating and spreading. The following sections illustrate each problem with a real dataset that I have encountered, and show how to tidy them. Column headers are values, not variable names

A common type of messy dataset is tabular data designed for presentation, where variables form both the rows and columns, and column headers are values, not variable names. While I would call this arrangement messy, in some cases it can be extremely useful. It provides efficient storage for completely crossed designs, and it can lead to extremely efficient computation if desired operations can be expressed as matrix operations.

The following code shows a subset of a typical dataset of this form. This dataset explores the relationship between income and religion in the US. It comes from a report1 produced by the Pew Research Center, an American think-tank that collects data on attitudes to topics ranging from religion to the internet, and produces many reports that contain datasets in this format.

pew <- tbl_df(read.csv("pew.csv", stringsAsFactors = FALSE, check.names = FALSE)) pew #> Source: local data frame [18 x 11] #> #> religion <$10k $10-20k $20-30k $30-40k $40-50k $50-75k #> 1 Agnostic 27 34 60 81 76 137 #> 2 Atheist 12 27 37 52 35 70 #> 3 Buddhist 27 21 30 34 33 58 #> 4 Catholic 418 617 732 670 638 1116 #> 5 Don't know/refused 15 14 15 11 10 35 #> 6 Evangelical Prot 575 869 1064 982 881 1486 #> 7 Hindu 1 9 7 9 11 34 #> 8 Historically Black Prot 228 244 236 238 197 223 #> 9 Jehovah's Witness 20 27 24 24 21 30 #> 10 Jewish 19 19 25 25 30 95 #> .. .. #> Variables not shown: $75-100k (int), $100-150k (int), >150k (int), Don't #> know/refused (int)

This dataset has three variables, religion, income and frequency. To tidy it, we need to gather the non-variable columns into a two-column key-value pair. This action is often described as making a wide dataset long (or tall), but I'll avoid those terms because they're imprecise.

When gathering variables, we need to provide the name of the new key-value columns to create. The first argument, is the name of the key column, which is the name of the variable defined by the values of the column headings. In this case, it's income. The second argument is the name of the value column, frequency. The third argument defines the columns to gather, here, every column except religion.

pew %>% gather(income, frequency, -religion) #> Source: local data frame [180 x 3] #> #> religion income frequency #> 1 Agnostic <$10k 27 #> 2 Atheist <$10k 12 #> 3 Buddhist <$10k 27 #> 4 Catholic <$10k 418 #> 5 Don't know/refused <$10k 15 #> 6 Evangelical Prot <$10k 575 #> 7 Hindu <$10k 1 #> 8 Historically Black Prot <$10k 228 #> 9 Jehovah's Witness <$10k 20 #> 10 Jewish <$10k 19 #> .. ..

This form is tidy because each column represents a variable and each row represents an observation, in this case a demographic unit corresponding to a combination of religion and income.

This format is also used to record regularly spaced observations over time. For example, the Billboard dataset shown below records the date a song first entered the billboard top 100. It has variables for artist, track, date.entered, rank and week. The rank in each week after it enters the top 100 is recorded in 75 columns, wk1 to wk75. This form of storage is not tidy, but it is useful for data entry. It reduces duplication since otherwise each song in each week would need its own row, and song metadata like title and artist would need to be repeated. This will be discussed in more depth in (multiple types)[#multiple-types].

billboard <- tbl_df(read.csv("billboard.csv", stringsAsFactors = FALSE)) billboard #> Source: local data frame [317 x 81] #> #> year artist track time date.entered wk1 wk2 #> 1 2000 2 Pac Baby Don't Cry (Keep... 4:22 2000-02-26 87 82 #> 2 2000 2Ge+her The Hardest Part Of ... 3:15 2000-09-02 91 87 #> 3 2000 3 Doors Down Kryptonite 3:53 2000-04-08 81 70 #> 4 2000 3 Doors Down Loser 4:24 2000-10-21 76 76 #> 5 2000 504 Boyz Wobble Wobble 3:35 2000-04-15 57 34 #> 6 2000 98ˆ0 Give Me Just One Nig... 3:24 2000-08-19 51 39 #> 7 2000 A*Teens Dancing Queen 3:44 2000-07-08 97 97 #> 8 2000 Aaliyah I Don't Wanna 4:15 2000-01-29 84 62 #> 9 2000 Aaliyah Try Again 4:03 2000-03-18 59 53 #> 10 2000 Adams, Yolanda Open My Heart 5:30 2000-08-26 76 76 #> .. .. #> Variables not shown: wk3 (int), wk4 (int), wk5 (int), wk6 (int), wk7 #> (int), wk8 (int), wk9 (int), wk10 (int), wk11 (int), wk12 (int), wk13 #> (int), wk14 (int), wk15 (int), wk16 (int), wk17 (int), wk18 (int), wk19 #> (int), wk20 (int), wk21 (int), wk22 (int), wk23 (int), wk24 (int), wk25 #> (int), wk26 (int), wk27 (int), wk28 (int), wk29 (int), wk30 (int), wk31 #> (int), wk32 (int), wk33 (int), wk34 (int), wk35 (int), wk36 (int), wk37 #> (int), wk38 (int), wk39 (int), wk40 (int), wk41 (int), wk42 (int), wk43 #> (int), wk44 (int), wk45 (int), wk46 (int), wk47 (int), wk48 (int), wk49 #> (int), wk50 (int), wk51 (int), wk52 (int), wk53 (int), wk54 (int), wk55 #> (int), wk56 (int), wk57 (int), wk58 (int), wk59 (int), wk60 (int), wk61 #> (int), wk62 (int), wk63 (int), wk64 (int), wk65 (int), wk66 (lgl), wk67 #> (lgl), wk68 (lgl), wk69 (lgl), wk70 (lgl), wk71 (lgl), wk72 (lgl), wk73 #> (lgl), wk74 (lgl), wk75 (lgl), wk76 (lgl)

To tidy this dataset, we first gather together all the wk columns. The column names give the week and the values are the ranks:

billboard2 <- billboard %>% gather(week, rank, wk1:wk76, na.rm = TRUE) billboard2 #> Source: local data frame [5,307 x 7] #> #> year artist track time date.entered week rank #> 1 2000 2 Pac Baby Don't Cry (Keep... 4:22 2000-02-26 wk1 87 #> 2 2000 2Ge+her The Hardest Part Of ... 3:15 2000-09-02 wk1 91 #> 3 2000 3 Doors Down Kryptonite 3:53 2000-04-08 wk1 81 #> 4 2000 3 Doors Down Loser 4:24 2000-10-21 wk1 76 #> 5 2000 504 Boyz Wobble Wobble 3:35 2000-04-15 wk1 57 #> 6 2000 98ˆ0 Give Me Just One Nig... 3:24 2000-08-19 wk1 51 #> 7 2000 A*Teens Dancing Queen 3:44 2000-07-08 wk1 97 #> 8 2000 Aaliyah I Don't Wanna 4:15 2000-01-29 wk1 84 #> 9 2000 Aaliyah Try Again 4:03 2000-03-18 wk1 59 #> 10 2000 Adams, Yolanda Open My Heart 5:30 2000-08-26 wk1 76 #> .. ..

Here we use na.rm to drop any missing values from the gather columns. In this data, missing values represent weeks that the song wasn't in the charts, so can be safely dropped.

In this case it's also nice to do a little cleaning, converting the week variable to a number, and figuring out the date corresponding to each week on the charts:

billboard3 <- billboard2 %>% mutate(week = extract_numeric(week), date = as.Date(date.entered) + 7 * (week - 1)) %>% select(-date.entered) billboard3 #> Source: local data frame [5,307 x 7] #> #> year artist track time week rank date #> 1 2000 2 Pac Baby Don't Cry (Keep... 4:22 1 87 2000-02-26 #> 2 2000 2Ge+her The Hardest Part Of ... 3:15 1 91 2000-09-02 #> 3 2000 3 Doors Down Kryptonite 3:53 1 81 2000-04-08 #> 4 2000 3 Doors Down Loser 4:24 1 76 2000-10-21 #> 5 2000 504 Boyz Wobble Wobble 3:35 1 57 2000-04-15 #> 6 2000 98ˆ0 Give Me Just One Nig... 3:24 1 51 2000-08-19 #> 7 2000 A*Teens Dancing Queen 3:44 1 97 2000-07-08 #> 8 2000 Aaliyah I Don't Wanna 4:15 1 84 2000-01-29 #> 9 2000 Aaliyah Try Again 4:03 1 59 2000-03-18 #> 10 2000 Adams, Yolanda Open My Heart 5:30 1 76 2000-08-26 #> .. ..

Finally, it's always a good idea to sort the data. We could do it by artist, track and week:

billboard3 %>% arrange(artist, track, week) #> Source: local data frame [5,307 x 7] #> #> year artist track time week rank date #> 1 2000 2 Pac Baby Don't Cry (Keep... 4:22 1 87 2000-02-26 #> 2 2000 2 Pac Baby Don't Cry (Keep... 4:22 2 82 2000-03-04 #> 3 2000 2 Pac Baby Don't Cry (Keep... 4:22 3 72 2000-03-11 #> 4 2000 2 Pac Baby Don't Cry (Keep... 4:22 4 77 2000-03-18 #> 5 2000 2 Pac Baby Don't Cry (Keep... 4:22 5 87 2000-03-25 #> 6 2000 2 Pac Baby Don't Cry (Keep... 4:22 6 94 2000-04-01 #> 7 2000 2 Pac Baby Don't Cry (Keep... 4:22 7 99 2000-04-08 #> 8 2000 2Ge+her The Hardest Part Of ... 3:15 1 91 2000-09-02 #> 9 2000 2Ge+her The Hardest Part Of ... 3:15 2 87 2000-09-09 #> 10 2000 2Ge+her The Hardest Part Of ... 3:15 3 92 2000-09-16 #> .. ..

Or by date and rank:

billboard3 %>% arrange(date, rank) #> Source: local data frame [5,307 x 7] #> #> year artist track time week rank date #> 1 2000 Lonestar Amazed 4:25 1 81 1999-06-05 #> 2 2000 Lonestar Amazed 4:25 2 54 1999-06-12 #> 3 2000 Lonestar Amazed 4:25 3 44 1999-06-19 #> 4 2000 Lonestar Amazed 4:25 4 39 1999-06-26 #> 5 2000 Lonestar Amazed 4:25 5 38 1999-07-03 #> 6 2000 Lonestar Amazed 4:25 6 33 1999-07-10 #> 7 2000 Lonestar Amazed 4:25 7 29 1999-07-17 #> 8 2000 Amber Sexual 4:38 1 99 1999-07-17 #> 9 2000 Lonestar Amazed 4:25 8 29 1999-07-24 #> 10 2000 Amber Sexual 4:38 2 99 1999-07-24 #> .. ..

Multiple variables stored in one column

After gathering columns, the key column is sometimes a combination of multiple underlying variable names. This happens in the tb (tuberculosis) dataset, shown below. This dataset comes from the World Health Organisation, and records the counts of confirmed tuberculosis cases by country, year, and demographic group. The demographic groups are broken down by sex (m, f) and age (0-14, 15-25, 25-34, 35-44, 45-54, 55-64, unknown).

tb <- tbl_df(read.csv("tb.csv", stringsAsFactors = FALSE)) tb #> Source: local data frame [5,769 x 22] #> #> iso2 year m04 m514 m014 m1524 m2534 m3544 m4554 m5564 m65 mu f04 f514 #> 1 AD 1989 NA NA NA NA NA NA NA NA NA NA NA NA #> 2 AD 1990 NA NA NA NA NA NA NA NA NA NA NA NA #> 3 AD 1991 NA NA NA NA NA NA NA NA NA NA NA NA #> 4 AD 1992 NA NA NA NA NA NA NA NA NA NA NA NA #> 5 AD 1993 NA NA NA NA NA NA NA NA NA NA NA NA #> 6 AD 1994 NA NA NA NA NA NA NA NA NA NA NA NA #> 7 AD 1996 NA NA 0 0 0 4 1 0 0 NA NA NA #> 8 AD 1997 NA NA 0 0 1 2 2 1 6 NA NA NA #> 9 AD 1998 NA NA 0 0 0 1 0 0 0 NA NA NA #> 10 AD 1999 NA NA 0 0 0 1 1 0 0 NA NA NA #> .. .. #> Variables not shown: f014 (int), f1524 (int), f2534 (int), f3544 (int), #> f4554 (int), f5564 (int), f65 (int), fu (int)

First we gather up the non-variable columns:

tb2 <- tb %>% gather(demo, n, -iso2, -year, na.rm = TRUE) tb2 #> Source: local data frame [35,750 x 4] #> #> iso2 year demo n #> 1 AD 2005 m04 0 #> 2 AD 2006 m04 0 #> 3 AD 2008 m04 0 #> 4 AE 2006 m04 0 #> 5 AE 2007 m04 0 #> 6 AE 2008 m04 0 #> 7 AG 2007 m04 0 #> 8 AL 2005 m04 0 #> 9 AL 2006 m04 1 #> 10 AL 2007 m04 0 #> .. ..

Column headers in this format are often separated by a non-alphanumeric character (e.g. ., -, _, :), or have a fixed width format, like in this dataset. separate() makes it easy to split a compound variables into individual variables. You can either pass it a regular expression to split on (the default is to split on non-alphanumeric columns), or a vector of character positions. In this case we want to split after the first character:

tb3 <- tb2 %>% separate(demo, c("sex", "age"), 1) tb3 #> Source: local data frame [35,750 x 5] #> #> iso2 year sex age n #> 1 AD 2005 m 04 0 #> 2 AD 2006 m 04 0 #> 3 AD 2008 m 04 0 #> 4 AE 2006 m 04 0 #> 5 AE 2007 m 04 0 #> 6 AE 2008 m 04 0 #> 7 AG 2007 m 04 0 #> 8 AL 2005 m 04 0 #> 9 AL 2006 m 04 1 #> 10 AL 2007 m 04 0 #> .. ..

Storing the values in this form resolves a problem in the original data. We want to compare rates, not counts, which means we need to know the population. In the original format, there is no easy way to add a population variable. It has to be stored in a separate table, which makes it hard to correctly match populations to counts. In tidy form, adding variables for population and rate is easy because they're just additional columns. Variables are stored in both rows and columns

The most complicated form of messy data occurs when variables are stored in both rows and columns. The code below loads daily weather data from the Global Historical Climatology Network for one weather station (MX17004) in Mexico for five months in 2010.

weather <- tbl_df(read.csv("weather.csv", stringsAsFactors = FALSE)) weather #> Source: local data frame [22 x 35] #> #> id year month element d1 d2 d3 d4 d5 d6 d7 d8 d9 d10 d11 #> 1 MX17004 2010 1 tmax NA NA NA NA NA NA NA NA NA NA NA #> 2 MX17004 2010 1 tmin NA NA NA NA NA NA NA NA NA NA NA #> 3 MX17004 2010 2 tmax NA 27.3 24.1 NA NA NA NA NA NA NA 29.7 #> 4 MX17004 2010 2 tmin NA 14.4 14.4 NA NA NA NA NA NA NA 13.4 #> 5 MX17004 2010 3 tmax NA NA NA NA 32.1 NA NA NA NA 34.5 NA #> 6 MX17004 2010 3 tmin NA NA NA NA 14.2 NA NA NA NA 16.8 NA #> 7 MX17004 2010 4 tmax NA NA NA NA NA NA NA NA NA NA NA #> 8 MX17004 2010 4 tmin NA NA NA NA NA NA NA NA NA NA NA #> 9 MX17004 2010 5 tmax NA NA NA NA NA NA NA NA NA NA NA #> 10 MX17004 2010 5 tmin NA NA NA NA NA NA NA NA NA NA NA #> .. .. #> Variables not shown: d12 (lgl), d13 (dbl), d14 (dbl), d15 (dbl), d16 #> (dbl), d17 (dbl), d18 (lgl), d19 (lgl), d20 (lgl), d21 (lgl), d22 (lgl), #> d23 (dbl), d24 (lgl), d25 (dbl), d26 (dbl), d27 (dbl), d28 (dbl), d29 #> (dbl), d30 (dbl), d31 (dbl)

It has variables in individual columns (id, year, month), spread across columns (day, d1-d31) and across rows (tmin, tmax) (minimum and maximum temperature). Months with less than 31 days have structural missing values for the last day(s) of the month.

To tidy this dataset we first gather the day columns:

weather2 <- weather %>% gather(day, value, d1:d31, na.rm = TRUE) weather2 #> Source: local data frame [66 x 6] #> #> id year month element day value #> 1 MX17004 2010 12 tmax d1 29.9 #> 2 MX17004 2010 12 tmin d1 13.8 #> 3 MX17004 2010 2 tmax d2 27.3 #> 4 MX17004 2010 2 tmin d2 14.4 #> 5 MX17004 2010 11 tmax d2 31.3 #> 6 MX17004 2010 11 tmin d2 16.3 #> 7 MX17004 2010 2 tmax d3 24.1 #> 8 MX17004 2010 2 tmin d3 14.4 #> 9 MX17004 2010 7 tmax d3 28.6 #> 10 MX17004 2010 7 tmin d3 17.5 #> .. ..

For presentation, I've dropped the missing values, making them implicit rather than explicit. This is ok because we know how many days are in each month and can easily reconstruct the explicit missing values.

We'll also do a little cleaning:

weather3 <- weather2 %>% mutate(day = extract_numeric(day)) %>% select(id, year, month, day, element, value) %>% arrange(id, year, month, day) weather3 #> Source: local data frame [66 x 6] #> #> id year month day element value #> 1 MX17004 2010 1 30 tmax 27.8 #> 2 MX17004 2010 1 30 tmin 14.5 #> 3 MX17004 2010 2 2 tmax 27.3 #> 4 MX17004 2010 2 2 tmin 14.4 #> 5 MX17004 2010 2 3 tmax 24.1 #> 6 MX17004 2010 2 3 tmin 14.4 #> 7 MX17004 2010 2 11 tmax 29.7 #> 8 MX17004 2010 2 11 tmin 13.4 #> 9 MX17004 2010 2 23 tmax 29.9 #> 10 MX17004 2010 2 23 tmin 10.7 #> .. ..

This dataset is mostly tidy, but the element column is not a variable; it stores the names of variables. (Not shown in this example are the other meteorological variables prcp (precipitation) and snow (snowfall)). Fixing this requires the spread operation. This performs the inverse of gathering by spreading the element and value columns back out into the columns:

weather3 %>% spread(element, value) #> Source: local data frame [33 x 6] #> #> id year month day tmax tmin #> 1 MX17004 2010 1 30 27.8 14.5 #> 2 MX17004 2010 2 2 27.3 14.4 #> 3 MX17004 2010 2 3 24.1 14.4 #> 4 MX17004 2010 2 11 29.7 13.4 #> 5 MX17004 2010 2 23 29.9 10.7 #> 6 MX17004 2010 3 5 32.1 14.2 #> 7 MX17004 2010 3 10 34.5 16.8 #> 8 MX17004 2010 3 16 31.1 17.6 #> 9 MX17004 2010 4 27 36.3 16.7 #> 10 MX17004 2010 5 27 33.2 18.2 #> .. ..

This form is tidy: there's one variable in each column, and each row represents one day. Multiple types in one table {#multiple-types}

Datasets often involve values collected at multiple levels, on different types of observational units. During tidying, each type of observational unit should be stored in its own table. This is closely related to the idea of database normalisation, where each fact is expressed in only one place. It's important because otherwise inconsistencies can arise.

The billboard dataset actually contains observations on two types of observational units: the song and its rank in each week. This manifests itself through the duplication of facts about the song: artist, year and time are repeated many times.

This dataset needs to be broken down into two pieces: a song dataset which stores artist, song name and time, and a ranking dataset which gives the rank of the song in each week. We first extract a song dataset:

song <- billboard3 %>% select(artist, track, year, time) %>% unique() %>% mutate(song_id = row_number()) song #> Source: local data frame [317 x 5] #> #> artist track year time song_id #> 1 2 Pac Baby Don't Cry (Keep... 2000 4:22 1 #> 2 2Ge+her The Hardest Part Of ... 2000 3:15 2 #> 3 3 Doors Down Kryptonite 2000 3:53 3 #> 4 3 Doors Down Loser 2000 4:24 4 #> 5 504 Boyz Wobble Wobble 2000 3:35 5 #> 6 98ˆ0 Give Me Just One Nig... 2000 3:24 6 #> 7 A*Teens Dancing Queen 2000 3:44 7 #> 8 Aaliyah I Don't Wanna 2000 4:15 8 #> 9 Aaliyah Try Again 2000 4:03 9 #> 10 Adams, Yolanda Open My Heart 2000 5:30 10 #> .. ..

Then use that to make a rank dataset by replacing repeated song facts with a pointer to song details (a unique song id):

rank <- billboard3 %>% left_join(song, c("artist", "track", "year", "time")) %>% select(song_id, date, week, rank) %>% arrange(song_id, date) rank #> Source: local data frame [5,307 x 4] #> #> song_id date week rank #> 1 1 2000-02-26 1 87 #> 2 1 2000-03-04 2 82 #> 3 1 2000-03-11 3 72 #> 4 1 2000-03-18 4 77 #> 5 1 2000-03-25 5 87 #> 6 1 2000-04-01 6 94 #> 7 1 2000-04-08 7 99 #> 8 2 2000-09-02 1 91 #> 9 2 2000-09-09 2 87 #> 10 2 2000-09-16 3 92 #> .. ..

You could also imagine a week dataset which would record background information about the week, maybe the total number of songs sold or similar "demographic" information.

Normalisation is useful for tidying and eliminating inconsistencies. However, there are few data analysis tools that work directly with relational data, so analysis usually also requires denormalisation or the merging the datasets back into one table. One type in multiple tables

It's also common to find data values about a single type of observational unit spread out over multiple tables or files. These tables and files are often split up by another variable, so that each represents a single year, person, or location. As long as the format for individual records is consistent, this is an easy problem to fix:

```
Read the files into a list of tables.

For each table, add a new column that records the original file name (the file name is often the value

Combine all tables into a single table.
```

Plyr makes this straightforward in R. The following code generates a vector of file names in a directory (data/) which match a regular expression (ends in .csv). Next we name each element of the vector with the name of the file. We do this because will preserve the names in the following step, ensuring that each row in the final data frame is labeled with its source. Finally, ldply() loops over each path, reading in the csv file and combining the results into a single data frame.

library(plyr) paths <- dir("data", pattern = "\.csv$", full.names = TRUE) names(paths) <- basename(paths) ldply(paths, read.csv, stringsAsFactors = FALSE)

Once you have a single table, you can perform additional tidying as needed. An example of this type of cleaning can be found at https://github.com/hadley/data-baby-names which takes 129 yearly baby name tables provided by the US Social Security Administration and combines them into a single file.

A more complicated situation occurs when the dataset structure changes over time. For example, the datasets may contain different variables, the same variables with different names, different file formats, or different conventions for missing values. This may require you to tidy each file to individually (or, if you're lucky, in small groups) and then combine them once tidied. An example of this type of tidying is illustrated in https://github.com/hadley/data-fuel-economy, which shows the tidying of epa fuel economy data for over

50,000 cars from 1978 to 2008. The raw data is available online, but each year is stored in a separate file and there are four major formats with many minor variations, making tidying this dataset a considerable challenge.

‘‘‘

5.3 Commentaires sur article de Wicham

commentaire 1: petit exemple d'utilisation avec dipplyr, notamment summarize.

autre commentaire

stackoverflow Some points (from my experiences):

```
Some people like colorful spreadsheets and make abundant use of formatting options. This is all fine, i
Sometimes I get some nicely formatted data (after I told people how to prepare it), but despite asking
Spreadsheets with several tables in them, multiple header lines or connected cells result in manual worl
Never, ever hide columns in Excel. If they are not needed, delete them. If they are needed, show them.
xls and its descendants are not suitable file formats for exchanging data with others or archiving it.
If you want to make life easy for others, adhere to the principles given in Hadley's article. Have a va
```

traduction (ggole)

Quelques points (à partir de mes expériences):

```
Certaines personnes aiment tableurs colorés et font un usage abondant d'options de formatage. Tout cela
Parfois, je reçois des données correctement formatées (après que je ai dit aux gens comment préparer),
Feuilles de calcul avec plusieurs tables en eux, plusieurs lignes d'en-tête ou cellules connectées entra
Jamais, jamais masquer des colonnes dans Excel. Se ils ne sont pas nécessaires, supprimez-les. Si elles
xls et ses descendants ne sont pas des formats adaptés pour échanger des données avec d'autres ou de les
Si vous voulez rendre la vie facile pour les autres, respecter les principes énoncés dans l'article de
```

Handling missing and messy data

6 Tidyr

nouveau package tidyr: page spécifique sur stackoverflow

Jeffrey Leek How to share data with a statistician

Data cleansing article de wikipédia. Pas d'équivalent en français.

Data manipulation

tidyr and pandas: Gather and Melt

7 Alternatives

How to merge and tidy data with Excel accédé le 1/3/2015.

8 Divers

Anatomie d'une table