Journal of Statistical Software

MMMMMM YYYY, Volume VV, Issue II. http://www.jstatsoft.org/

Tidy Data

Hadley Wickham
RStudio

Abstract

A huge amount of effort is spent cleaning data to get it ready for data analysis, but there has been little research on how to make data cleaning as easy and effective as possible. This paper tackles a small, but important, subset of data cleaning: data "tidying". Tidy datasets are easy to manipulate, model and visualise, and have a specific structure: variables are stored in columns, observations in rows, and a single type of experimental unit per dataset. This framework makes it easy to tidy messy datasets because only a small set of tools are needed to deal with a wide range of un-tidy datasets. This structure also makes it easier to develop tidy tools for data analysis, tools that both take in and put out tidy datasets. The advantages of a consistent data structure and matching tools are demonstrated with a case study free from mundane data manipulation chores.

Keywords: data cleaning, data tidying, relational databases, R.

1. Introduction

It's often said that 80% of the effort in a data analysis is spent on data cleaning (Dasu and Johnson 2003), the process of getting the data ready to analyse. Data cleaning is not only a vital first step, but it is often repeated multiple times over the course of an analysis as new problems come to light. Despite the amount of time it takes, there has been little research on how to clean data well. Part of the challenge is the breadth of activities that cleaning encompasses, from outlier checking, to date parsing, to missing value imputation. To get a handle on the problem, this paper focusses on a small, but important, subset of data cleaning that I call data "tidying": structuring data values to make the process of data analysis as easy as possible.

The principles of tidy data provide a standard way of organising the data values within a dataset. This makes initial data cleaning easier because you have a *standard* that you can start with: you don't need to reinvent the wheel for each new dataset. It also makes it easier to develop data analysis tools that work together seamlessly. Current tools are often inconsistent, so you need to spend time munging the output from one tool into the input for 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 the uninteresting details of data structure.

The principles of tidy data are closely tied to relational databases and Codd's relational algebra (Codd 1990), but are framed in a language more familiar to statisticians. Computer scientists have also contributed much to the study of data cleaning. For example, Lakshmanan, Sadri, and Subramanian (1996) defines a extension to SQL allowing it to operate on messy datasets, Raman and Hellerstein (2001) provides a framework for cleaning datasets, and Kandel, Paepcke, Hellerstein, and Heer (2011) develops an interactive tool that makes data cleaning easy, automatically creating the code to clean datasets with a friendly user interface. These tools are useful, but are presented in language foreign to most statisticians, fail to give much advice on how datasets should be structured, and lack connections to the tools of data analysis.

The development of tidy data has been driven by my struggles working with real-world datasets, which are often organised in bizarre ways. I have spent countless hours struggling to get these datasets organised in a way that makes data analysis possible, let alone easy. I have also struggled to impart these skills to my students, so they can tackle problems in the real-world, where there are few constraints on how datasets are organised. In the course of these struggles I developed the **reshape** and **reshape2** (Wickham 2007) R packages, but while I could intuitively use the tools, I lacked the framework to make my understanding explicit. This paper provides that framework, and provides a consistent "philosophy of data" that underlies my work in the **plyr** (Wickham 2011) and **ggplot2** (Wickham 2009) packages.

The paper proceeds as follows. Section 2 begins by defining the three characteristics that make a dataset tidy. Most real world datasets are not tidy, so Section 3 describes the operations needed to make messy datasets tidy, and illustrates the techniques with a range of real-life examples. Section 4 defines tidy tools, tools that input and output tidy datasets, and discusses how together tidy data and tidy tools make it easier to do a data analysis. These principles are illustrated with a small case study in Section 5. Section 6 concludes with a discussion of what this framework misses and other approaches that might be fruitful to pursue.

2. Defining tidy data

> 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. Here we will define the essence of tidy data, and provide important vocabulary for describing datasets.

Statistical datasets usually come in the form of a rectangular table, made up of rows and columns. Tables are usually labelled with column names, and sometimes with row names.

The table is a collection of data values, typically either numeric (if quantitative) or character (if qualitative). Values are grouped into variables, measurements of a single quantifiable property. Multiple measurements made on the same observational unit form an observation.

There is a little circularity in the definition, which is resolved by defining an observational unit as the smallest constituent atom of the experimental unit. For example, in a longitudinal medical study, the experimental unit might be the patient, but the observational unit is the patient at each time point. In an agricultural experiment, the experimental units might be the cages given different diets, but the observational units are the pigs whose individual weights are measured. This definition of observational unit leads to variables which are physical properties, like blood pressure and weight, not conditional properties like blood pressure at time 1, or weight of pig 2. Structure in the variable names ('trtA', 'trtB', 'trtC'; 'jan1', 'feb1', 'mar1') is difficult to exploit, and suggests that the names are values of other variables.

Table 1 illustrates a data format commonly seen in the wild. Here it is used to store information about an imaginary experiment with two treatments. There are two columns and three rows (plus column and row names), defining three variables (name, treatment, result) and either five or six observations, depending on how you think about the missing value.

	treatmentA	treatmentB
John Smith	—	5
Jane Doe	1	4
Mary Johnson	2	3

Table 1: Typical presentation dataset.

It's not always obvious what qualifies as a variable, and a variable in one analysis may be a value in another. For example, if the columns in the example were 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 weight 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/columns (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 rows/observations (e.g., average of group a vs. average of group b) than between columns. If you find yourself fighting these constraints, you may not have correctly identified the appropriate variables for your problem.

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

A dataset is messy or tidy depending on how rows, columns and tables are matched up with observations, variables and types. In tidy data:

1. Each variable forms a column,

2. Each observation forms a row,

3. Each table (or file) stores information about one observational type.

This is Codd's 3rd normal form (Codd 1990), but with the constraints framed in statistical language, and the focus on a single dataset, not the many connected datasets common in relational databases. I will call other arrangements of data messy.

Table 2 shows the tidy version of Table 1. Now each row represents an observation, the result of one treatment on one person, and each column is a variable. We could also choose to drop the row representing the result of John Smith's treatment a, but we don't, because missing values should only be omitted if they are structural: impossible because of the design of the experiment.

name	treatment	result
Jane Doe	a	1
Jane Doe	b	4
John Smith	a	—
John Smith	b	5
Mary Johnson	a	2
Mary Johnson	b	3

Table 2: Tidied data.

While 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. These are often called dimensions by computer scientists, and typically denoted by 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 variables. This is the convention adopted by all tabular displays in this paper. When using tables for communication, this so-called "Alabama first" (Wainer 2000) ordering should be abandoned, and replaced with an ordering based on a meaningful variable.

3. Tidying messy datasets

Real datasets can violate the three precepts of tidy data in almost every way imaginable, and often do. While occasionally you do get a dataset that you can start analysing immediately, this is the exception, not the norm. This section describes the the five most common problems that make datasets messy, 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 unit 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: "melting", string splitting, and "casting". The following sections illustrate each problem with a real dataset that I have encountered in practice, and show how to tidy it. The complete datasets and the R code used to tidy them are available online at https://github.com/hadley/tidy-data, and in the online supplementary materials for this paper.

3.1. 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 in this paper I call this arrangement messy, in some cases it can be extremely useful. It provides efficient storage for completely crossed designs, and it can provide extremely efficient computation if desired operations can be expressed as matrix operations. This issue is discussed in more depth in Section 6.

Table 3 shows a subset of a typical dataset of this form. This dataset explores the relationship between income and religion in the US, and comes from a report[1] produced by the Pew Research Center. The Pew Center is an American think-tank that collects data on attitudes to topics from religion to the internet, and produces many reports that contain datasets in the format shown.

religion	<$10k	$10-20k	$20-30k	$30-40k	$40-50k	$50-75k	$75-100k
Agnostic	27	34	60	81	76	137	122
Atheist	12	27	37	52	35	70	73
Buddhist	27	21	30	34	33	58	62
Catholic	418	617	732	670	638	1116	949
DonâĂŹt know/refused	15	14	15	11	10	35	21
Evangelical Prot	575	869	1064	982	881	1486	949
Hindu	1	9	7	9	11	34	47
Historically Black Prot	228	244	236	238	197	223	131
Jehovah's Witness	20	27	24	24	21	30	15
Jewish	19	19	25	25	30	95	69

Table 3: The first ten rows of data on income and religion from the Pew Forum. Two columns have been omitted to save space: '$100-150k' and '$150k'

This dataset has three variables, religion, income and frequency, and to tidy it we need to "melt", or stack it, turning columns into rows. This makes "wide" datasets "long" or "tall", but I will avoid those terms because they are imprecise. Melting is parameterised by a list of columns that are already variables, or *colvars* for short. The other columns are converted into two variables: a new "column" variable that contains repeated column headings and

[1] http://religions.pewforum.org/pdf/comparison-Income%20Distribution%20of%20Religious%20Traditions.pdf

row	a	b	c
a	1	4	7
b	2	5	8
c	3	6	9

(a) Raw data

row	column	value
a	a	1
b	a	2
c	a	3
a	b	4
b	b	5
c	b	6
a	c	7
b	c	8
c	c	9

(b) Molten data

Table 4: A simple example of melting. (a) is melted with one colvar, row, yielding the molten dataset (b). The information in each table is exactly the same, just stored in a different way.

a "value" variable that contains a vector of data values concatenated from the previously separate columns. This is illustrated in Table 4 with a toy dataset. The result of melting is a "molten" dataset.

The Pew dataset has one colvar, religion, and melting yields Table 5. The column variable has been renamed to `income`, and the value column to `freq`, to better reflect their roles in this dataset. This form is tidy: 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.

religion	income	freq
Agnostic	<$10k	27
Agnostic	$10-20k	34
Agnostic	$20-30k	60
Agnostic	$30-40k	81
Agnostic	$40-50k	76
Agnostic	$50-75k	137
Agnostic	$75-100k	122
Agnostic	$100-150k	109
Agnostic	>150k	84
Agnostic	Don't know/refused	96

Table 5: The first ten rows of the tidied Pew survey dataset on income and religion. The variable "column" has been renamed to income, and "value" to freq.

Another common use of this data format is to record regularly spaced observations over time. For example, the billboard dataset shown in Table 6 records the date a song first entered the billboard top 100. It has variables 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`. If a song is in the top 100 for less than 75 weeks the remaining columns are filled with missing values. This form of storage is not tidy, but is useful for data entry because it reduces duplication:

otherwise each song in each week would need its own row, and song metadata like title and artist would need to be repeated. This issue will be discussed in more depth in Section 3.4.

year	artist	track	time	date.entered	wk1	wk2	wk3
2000	2 Pac	Baby Don't Cry	4:22	2000-02-26	87	82	72
2000	2Ge+her	The Hardest Part Of Breaking Up	3:15	2000-09-02	91	87	92
2000	3 Doors Down	Kryptonite	3:53	2000-04-08	81	70	68
2000	98Âř	Give Me Just One Night	3:24	2000-08-19	51	39	34
2000	A*Teens	Dancing Queen	3:44	2000-07-08	97	97	96
2000	Aaliyah	I Don't Wanna	4:15	2000-01-29	84	62	51
2000	Aaliyah	Try Again	4:03	2000-03-18	59	53	38
2000	Adams, Yolanda	Open My Heart	5:30	2000-08-26	76	76	74

Table 6: The first eight billboard top hits for 2000. Other columns not shown are `wk4`, `wk5`, ..., `wk75`.

This dataset has colvars year, artist, track, time, and date.entered. Melting yields Table 7. I have also done a little cleaning as well as tidying: the column variable has been converted to a week variable by extracting the week number, and I have added a date variable that combines the date the song entered the billboard with the week number.

year	artist	time	track	date	week	rank
2000	2 Pac	4:22	Baby Don't Cry	2000-02-26	1	87
2000	2 Pac	4:22	Baby Don't Cry	2000-03-04	2	82
2000	2 Pac	4:22	Baby Don't Cry	2000-03-11	3	72
2000	2 Pac	4:22	Baby Don't Cry	2000-03-18	4	77
2000	2 Pac	4:22	Baby Don't Cry	2000-03-25	5	87
2000	2 Pac	4:22	Baby Don't Cry	2000-04-01	6	94
2000	2 Pac	4:22	Baby Don't Cry	2000-04-08	7	99
2000	2Ge+her	3:15	The Hardest Part Of Breaking Up	2000-09-02	1	91
2000	2Ge+her	3:15	The Hardest Part Of Breaking Up	2000-09-09	2	87
2000	2Ge+her	3:15	The Hardest Part Of Breaking Up	2000-09-16	3	92
2000	3 Doors Down	3:53	Kryptonite	2000-04-08	1	81
2000	3 Doors Down	3:53	Kryptonite	2000-04-15	2	70
2000	3 Doors Down	3:53	Kryptonite	2000-04-22	3	68
2000	3 Doors Down	3:53	Kryptonite	2000-04-29	4	67
2000	3 Doors Down	3:53	Kryptonite	2000-05-06	5	66

Table 7: First fifteen rows of the tidied billboard dataset. The `date` column does not appear in the original table, but can be computed from `date.entered` and `week`.

3.2. Multiple variables stored in one column

After melting, it often happens that the "column" variable is a combination of multiple underlying variables. This is illustrated by the tuberculosis (TB) dataset, a sample of which is shown in Table 8. 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).

country	year	m014	m1524	m2534	m3544	m4554	m5564	m65	mu	f014
AD	2000	0	0	1	0	0	0	0	—	—
AE	2000	2	4	4	6	5	12	10	—	3
AF	2000	52	228	183	149	129	94	80	—	93
AG	2000	0	0	0	0	0	0	1	—	1
AL	2000	2	19	21	14	24	19	16	—	3
AM	2000	2	152	130	131	63	26	21	—	1
AN	2000	0	0	1	2	0	0	0	—	0
AO	2000	186	999	1003	912	482	312	194	—	247
AR	2000	97	278	594	402	419	368	330	—	121
AS	2000	—	—	—	—	1	1	—	—	—

Table 8: Original TB dataset. Corresponding to each 'm' column for males, there is also an 'f' column for females: `f1524`, `f2534` and so on. These are not shown to conserve space. Note the mixture of 0s and missing values (—): this is due to the data collection process and the distinction is important for this dataset.

Column headers in this format are often separated by some character (`.`, `-`, `_`, `:`) and the string can be broken into pieces using that character as a divider. In other cases, such as for this dataset, more careful string processing is required, or the variable names need to be matched to a lookup table that converts single compound values to values of multiple variables.

Table 9(a) shows the results of melting the TB dataset, and Table 9(b) the results of splitting the column "column" into two real variables: age and sex.

Storing the values in this form resolves another problem in the original data: we want to compare rates, not counts, but to compute rates we need to know the population. In the original format, there is no easy way to add a population variable, and it has to be stored in a separate table, which makes it hard to correctly match populations to counts. In tidy form, adding variables to store population and rate is easy: they just become additional columns.

3.3. Variables are stored in both rows and columns

The most complicated form of messy data is when variables have been stored in both rows and columns. Table 10 shows daily weather data from the Global Historical Climatology Network for one weather station (MX17004) in Mexico for five months in 2010. 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. The element column is not a variable: it stores the names of variables.

To tidy this dataset we first melt it with colvars id, year, month and the column that contains variable names, element, yielding Table 11(a). For presentation, we have dropped the missing values, making them implicit rather than explicit. This is permissible because we know how many days are in each month and can easily reconstruct the explicit missing values.

country	year	column	cases
AD	2000	m014	0
AD	2000	m1524	0
AD	2000	m2534	1
AD	2000	m3544	0
AD	2000	m4554	0
AD	2000	m5564	0
AD	2000	m65	0
AE	2000	m014	2
AE	2000	m1524	4
AE	2000	m2534	4
AE	2000	m3544	6
AE	2000	m4554	5
AE	2000	m5564	12
AE	2000	m65	10
AE	2000	f014	3

(a) Molten data

country	year	sex	age	cases
AD	2000	m	0-14	0
AD	2000	m	15-24	0
AD	2000	m	25-34	1
AD	2000	m	35-44	0
AD	2000	m	45-54	0
AD	2000	m	55-64	0
AD	2000	m	65+	0
AE	2000	m	0-14	2
AE	2000	m	15-24	4
AE	2000	m	25-34	4
AE	2000	m	35-44	6
AE	2000	m	45-54	5
AE	2000	m	55-64	12
AE	2000	m	65+	10
AE	2000	f	0-14	3

(b) Tidy data

Table 9: Tidying the TB dataset requires first melting, and then splitting the "column" column into two variables: sex and age.

This dataset is mostly tidy, but we have two variables stored in rows: `tmin` and `tmax`, the type of observation. Not shown in this example are other meteorological variables prcp (precipitation) and snow (snowfall). Fixing this requires the cast, or unstack, operation, which performs the inverse of melting, rotating the `element` variable back out into the columns to give Table 11(b). This form is tidy: there is one variable in each column, and each row represents a day's observations. The cast operation is described in depth in Wickham (2007).

3.4. Multiple types in one table

Datasets often involve values collected at multiple levels, on different types of observational unit. 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; if not, it's possible for inconsistencies to occur.

The billboard dataset described in Table 7 actually contains observations on two types of observational unit: the song, and its rank in each week. This is revealed through the duplication of facts about the song: the artist and time are repeated for every song in each week. The billboard dataset needs to be broken down into two datasets: a song dataset which stores artist, song name and time; and a ranking dataset which gives the rank of the song in each week. Table 12 shows these two datasets. 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, but there are few data analysis tools that work directly with relational data, so analysis usually requires denormalisation, merging the datasets back into one table.

id	year	month	element	d1	d2	d3	d4	d5	d6	d7	d8	d9	d10
MX17004	2010	1	tmax	—	—	—	—	—	—	—	—	—	—
MX17004	2010	1	tmin	—	—	—	—	—	—	—	—	—	—
MX17004	2010	2	tmax	—	27.3	24.1	—	—	—	—	—	—	—
MX17004	2010	2	tmin	—	14.4	14.4	—	—	—	—	—	—	—
MX17004	2010	3	tmax	—	—	—	—	32.1	—	—	—	—	34.5
MX17004	2010	3	tmin	—	—	—	—	14.2	—	—	—	—	16.8
MX17004	2010	4	tmax	—	—	—	—	—	—	—	—	—	—
MX17004	2010	4	tmin	—	—	—	—	—	—	—	—	—	—
MX17004	2010	5	tmax	—	—	—	—	—	—	—	—	—	—
MX17004	2010	5	tmin	—	—	—	—	—	—	—	—	—	—

Table 10: Original weather dataset. There is a column for each possible day in the month. Columns `d11` to `d31` have been omitted to conserve space.

id	date	element	value
MX17004	2010-01-30	tmax	27.8
MX17004	2010-01-30	tmin	14.5
MX17004	2010-02-02	tmax	27.3
MX17004	2010-02-02	tmin	14.4
MX17004	2010-02-03	tmax	24.1
MX17004	2010-02-03	tmin	14.4
MX17004	2010-02-11	tmax	29.7
MX17004	2010-02-11	tmin	13.4
MX17004	2010-02-23	tmax	29.9
MX17004	2010-02-23	tmin	10.7

(a) Molten data

id	date	tmax	tmin
MX17004	2010-01-30	27.8	14.5
MX17004	2010-02-02	27.3	14.4
MX17004	2010-02-03	24.1	14.4
MX17004	2010-02-11	29.7	13.4
MX17004	2010-02-23	29.9	10.7
MX17004	2010-03-05	32.1	14.2
MX17004	2010-03-10	34.5	16.8
MX17004	2010-03-16	31.1	17.6
MX17004	2010-04-27	36.3	16.7
MX17004	2010-05-27	33.2	18.2

(b) Tidy data

Table 11: (a) Molten weather dataset. This is almost tidy, but the `element` column contains names of variables, not values. Missing values are dropped to conserve space. (b) Tidy weather dataset. Each row representation the meteorological measurements for a single day. There are two measured variables, minimum (`tmin`) and maximum (`tmax`) temperature; all other variables are fixed.

id	artist	track	time
1	2 Pac	Baby Don't Cry	4:22
2	2Ge+her	The Hardest Part Of Breaking Up	3:15
3	3 Doors Down	Kryptonite	3:53
4	3 Doors Down	Loser	4:24
5	504 Boyz	Wobble Wobble	3:35
6	98Âř	Give Me Just One Night	3:24
7	A*Teens	Dancing Queen	3:44
8	Aaliyah	I Don't Wanna	4:15
9	Aaliyah	Try Again	4:03
10	Adams, Yolanda	Open My Heart	5:30
11	Adkins, Trace	More	3:05
12	Aguilera, Christina	Come On Over Baby	3:38
13	Aguilera, Christina	I Turn To You	4:00
14	Aguilera, Christina	What A Girl Wants	3:18
15	Alice Deejay	Better Off Alone	6:50

id	date	rank
1	2000-02-26	87
1	2000-03-04	82
1	2000-03-11	72
1	2000-03-18	77
1	2000-03-25	87
1	2000-04-01	94
1	2000-04-08	99
2	2000-09-02	91
2	2000-09-09	87
2	2000-09-16	92
3	2000-04-08	81
3	2000-04-15	70
3	2000-04-22	68
3	2000-04-29	67
3	2000-05-06	66

Table 12: Normalised billboard dataset split up into song dataset (left) and rank dataset (right). First 15 rows of each dataset shown; `genre` omitted from song dataset, `week` omitted from rank dataset.

3.5. 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 multiple files. These tables are often split up by another variable, so each file 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 in all files, add a new column that records the original file name (because the filename is often the value of an important variable), then combine all tables into a single table. 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 has changed over time: different variables, variable names, data storage formats, or different conventions for missing values. This may require each file to tidied individually (or hopefully in small groups) and then combined together. 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.

4. Tidy tools

Once you have a tidy dataset, what can you do with it? Tidy data is only worthwhile if it makes analysis easier. This section discusses tidy tools, tools that take tidy datasets as input and return tidy datasets as output. Tidy tools are useful because the output of one tool can be used as the input to another, making it straightforward to compose multiple tools to solve a problem. Tidy data also ensures that variables are stored in a consistent, explicit manner. This makes each tool simpler, because it doesn't need a Swiss Army knife of parameters for dealing with different dataset structures.

Tools can be messy for two reasons: either they take messy datasets as input (messy-input tools) or they produce messy datasets as output (messy-output tools). Messy-input tools are typically more complicated than tidy-input tools because they need to include some parts of the tidying process. This can be useful for common types of messy datasets, but it typically makes the function more complex, harder to use and harder to maintain. Messy-output tools are frustrating and slow down analysis because they can not be easily composed and you must constantly think about how to convert from one format to another. We'll see examples of both in the following sections.

Next, I give examples of tidy and messy tools for three important components of analysis: data manipulation, visualisation and modelling. I will focus particularly on tools provided by R (R Development Core Team 2011), because it has many existing tidy tools, but I will also touch on other statistical programming environments.

4.1. Manipulation

Data manipulation includes variable-by-variable transformation (e.g., `log` or `sqrt`), as well as aggregation, filtering and reordering. In my experience, there are four extremely common operations that are performed over and over again in the course of an analysis. These are the four fundamental verbs of data manipulation:

- Filtering, or subsetting, where observations are removed based on some condition.
- Transforming, where new variables are added or existing variables modified. These modifications can involve either a single variable (e.g., log-transforming a variable), or involve multiple variables (e.g., computing density from weight and volume).
- Aggregating, where multiple values are collapsed into a single value, e.g., by summing or taking means.
- Sorting, where the order of observations is changed.

All these operations are made easier when there is a consistent way to refer to variables. Tidy data provides this because each variable lives in its own column.

In R, filtering and transforming are performed by the base R functions `subset` and `transform`. These are input and output-tidy. The `aggregate` function performs group-wise aggregation, is input-tidy, and is output-tidy providing a single aggregation method is used. The **plyr** package provides a tidy `summarise` and `arrange` function for aggregation and sorting.

The four verbs can be, and often are, modified by the by preposition. We often need group-wise aggregates, transformations or subsets: pick the biggest in each group, average over replicates and so on. Combining each of the four verbs with a by operator allows them to operate on subsets of a data frame at a time. Many SAS PROCs possess a BY statement which allows the operation to be performed by group, and are generally input-tidy. Base R possesses a `by` function, which is input-tidy, but not output-tidy, because it produces a list. The `ddply` function from the **plyr** package is a tidy alternative.

Other tools are needed when we have multiple datasets. An advantage of tidy data is the ease with which it can be combined with other tidy datasets: we just need a join operator which works by matching common variables and adding new columns. This is implemented in the `merge` function in base R, or the `join` function in **plyr**. Compare this to the difficulty of combining datasets stored in arrays; these typically require painstaking alignment before matrix operations can be used, and errors can be very hard to detect.

4.2. Visualisation

Tidy visualisation tools need only to be input-tidy as their output is visual. Domain specific languages work particularly well for the visualisation of tidy datasets because they can describe a visualisation as a mapping between variables in the dataset and aesthetic properties of the graphic like position, size, shape and colour. This is the idea behind the grammar of graphics (Wilkinson 2005), and the layered grammar of graphics (Wickham 2010), an extension tailored specifically for R.

Most graphical tools in R are input tidy, including the **base** `plot` function, the **lattice** family of plots (Sarkar 2008) and **ggplot2** (Wickham 2009). Some specialised tools exist for visualising messy datasets. Some base R functions like `barplot`, `matplot`, `dotchart`, and `mosaicplot`, work with messy datasets where variables are spread out over multiple columns. Similarly, the parallel coordinates plot (Wegman 1990; Inselberg 1985) can be used to create time series plots for messy datasets where each time point is a column.

4.3. Modelling

Modelling is the inspiration that has driven much of this work, because most modelling tools work best with tidy datasets. Every statistical language has a way of describing a model as a connection between different variables, a domain specific language that connects responses to predictors:

- R (`lm`): `y ~ a + b + c * d`
- SAS (`PROC GLM`): `y = a + b + c * d`
- SPSS (`glm`): `y BY a b c d / DESIGN a b c d c*d`

This is not to say that tidy data is the format used internally to compute the regression. Significant transformations take place to produce a numeric matrix that can easily be fed to standard linear algebra routines. Common transformations include adding an intercept column (column of all ones), turning categorical variables into multiple binary dummy variables, and for smooth terms like splines, projecting the data values on to the appropriate basis functions.

There have been some attempts to adapt modelling functions for specific types of messy datasets. For example, in SAS's `proc glm`, multiple variables on the response side of the equation will be interpreted as repeated measures if the REPEATED keyword is present. For the raw billboard data, we could construct a model of the form `wk1-wk76 = track` instead of `rank = week * track` on the tidy data (provided week is labelled as a categorical variable).

Another interesting example is the classic paired t-test, which can be computed in two ways depending on how the data is stored. If the data is stored as in Table 13(a), then a paired t-test is just a t-test applied to the mean difference between x and y. If it is stored in the form of Table 13(b), then an equivalent model is to fit a mixed effects model, with a fixed dummy variable representing the "variable", and a random intercept for each id. (In R's lmer4 notation, this is `value ~ variable + (1 | id)`). Either way of modelling the data yields the same result. Without more information we can't say which form of the data is tidy: if x and y represent length of left and right arms, then Table 13(a) is tidy, if x and y represent measurements on day 1 and day 10, then Table 13(b) is tidy.

While model inputs usually require tidy inputs, model outputs, such as predictions and estimated coefficients, aren't always tidy. This makes it more difficult to combine results from multiple models. For example, in R, the default representation of model coefficients is not tidy because it does not have an explicit variable that records the variable name for each estimate: these are instead recorded as row names. In R, row names must be unique, so combining coefficients from many models (e.g., from bootstrap resamples, or subgroups) requires workarounds to avoid losing important information. This knocks you out of the flow of analysis and makes it harder to combine the results from multiple models.

5. Case study

id	x	y
1	22.19	24.05
2	19.82	22.91
3	19.81	21.19
4	17.49	18.59
5	19.44	19.85

(a) Data for paired t-test

id	variable	value
1	x	22.19
2	x	19.82
3	x	19.81
4	x	17.49
5	x	19.44
1	y	24.05
2	y	22.91
3	y	21.19
4	y	18.59
5	y	19.85

(b) Data for mixed effects model

Table 13: Two data sets for performing the same test.

The following case study illustrates how tidy data and tidy tools make data analysis easier by easing the transitions between manipulation, visualisation and modelling: you will not see any code that exists solely to get the output of one function into the right format for input to another. I'll show the R code that performs the analysis, but even if you're not familiar with R or the exact idioms I use, I've tried to make it easy to understand by tightly interleaving code, results and explanation.

The case study uses individual-level mortality data from Mexico, with the goal of finding causes of death that have notably different time patterns within a day. The full dataset has information on 539,530 deaths in Mexico in 2008 and 55 variables, including the the location and time of death, the cause of death, and demographics of the deceased. Table 14 shows a small sample of the dataset, focussing on variables related to time of death (year, month, day and hour), and cause (cod).

To achieve our goal of finding unusual time courses, we'll start by counting the number of deaths in each hour (`hod`) for each cause (`cod`) with the tidy `count` function. Then we remove missing (and hence uninformative for our purpose) values with `subset`. This gives Table 15(a).

First, we count the the number of deaths in each hour of the day for each cause of death, and remove missing observations:

```
R> hod2 <- count(deaths, c("hod", "cod"))
R> hod2 <- subset(hod2, !is.na(hod))
```

Next we join the dataset to `codes` dataset that gives informative labels for the disease. These two datasets are connected by the `cod` variable. This adds a new variable, `disease`, shown in Table 15(b)).

```
R> hod2 <- join(hod2, codes, by = "cod")
```

The total deaths for each cause vary over several orders of magnitude: there were 46,794 deaths from heart attack, and 10 deaths from avalanche. This means that it makes more

yod	mod	dod	hod	cod
2008	1	1	1	B20
2008	1	2	4	I67
2008	1	3	8	I50
2008	1	4	12	I50
2008	1	5	16	K70
2008	1	6	18	I21
2008	1	7	20	I21
2008	1	8	—	K74
2008	1	10	5	K74
2008	1	11	9	I21
2008	1	12	15	I25
2008	1	13	20	R54
2008	1	15	2	I61
2008	1	16	7	I21
2008	1	17	13	I21

Table 14: A sample of 16 rows and 5 columns from the original dataset of 539,530 rows and 55 columns.

sense to compare the proportion of deaths in each hour, rather than the total number. We compute this by breaking the dataset down by `cod`, and then `transform`ing to add a new `prop` column, the hourly frequency divided by the total number of deaths from that cause. This new columns is Table 15(c).

`ddply` breaks down its first argument (`hod2`) by its second argument (the `cod` variable), and applies its third argument (`transform`) to each piece. The fourth argument (`prop = freq / sum(freq)`) is passed on to transform.

```
R> hod2 <- ddply(hod2, "cod", transform, prop = freq / sum(freq))
```

We then compute the overall average death rate for each hour, and merge that back into the original dataset. This yields Table 15(d) and allows us to easily compare each disease with the overall pattern by comparing `prop` to `prop_all`.

First, we work out the number of people dying each hour by breaking down `hod2` by `hod`, summarising with a total for over each cause of death.

```
R> overall <- ddply(hod2, "hod", summarise, freq_all = sum(freq))
```

Next we work out overall proportion of people dying in each hour:

```
R> overall <- transform(overall, prop_all = freq_all / sum(freq_all))
```

Then finally join the overall dataset with the individual dataset to make it easier to compare the two:

```
R> hod2 <- join(hod2, overall, by = "hod")
```

hod	cod	freq	disease	prop	freq_all	prop_all
8	B16	4	Acute hepatitis B	0.04	21915	0.04
8	E84	3	Cystic fibrosis	0.03	21915	0.04
8	I21	2205	Acute myocardial infarction	0.05	21915	0.04
8	N18	315	Chronic renal failure	0.04	21915	0.04
9	B16	7	Acute hepatitis B	0.07	22401	0.04
9	E84	1	Cystic fibrosis	0.01	22401	0.04
9	I21	2209	Acute myocardial infarction	0.05	22401	0.04
9	N18	333	Chronic renal failure	0.04	22401	0.04
10	B16	10	Acute hepatitis B	0.10	24321	0.05
10	E84	7	Cystic fibrosis	0.07	24321	0.05
10	I21	2434	Acute myocardial infarction	0.05	24321	0.05
10	N18	343	Chronic renal failure	0.04	24321	0.05
11	B16	6	Acute hepatitis B	0.06	23843	0.05
11	E84	3	Cystic fibrosis	0.03	23843	0.05
11	I21	2128	Acute myocardial infarction	0.05	23843	0.05
(a)			(b)	(c)	(d)	

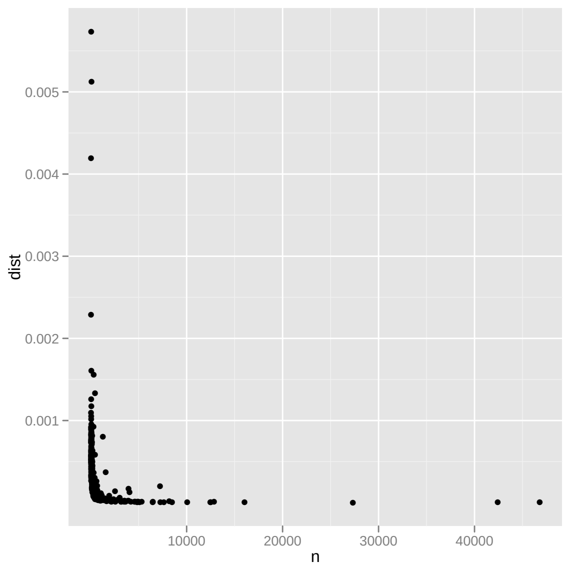
Table 15: A sample of four diseases and four hours from `hod2` data frame.

Next we compute a distance between the time course of each cause of death and the overall time course. Here we use a simple mean squared deviation. We also record the sample size, the total number of deaths from that cause. To ensure that the disease we work with have a decent amount of underlying data we'll only work with diseases with more than 50 deaths in total (~2/hour).

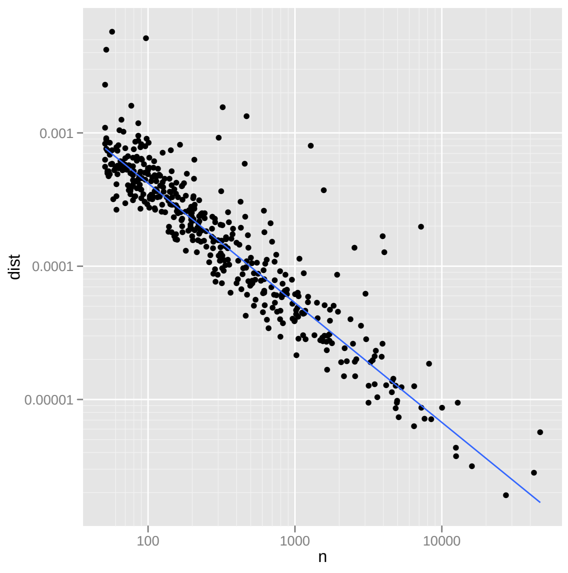
```
R> devi <- ddply(hod2, "cod", summarise,
R>   n = sum(freq),
R>   dist = mean((prop - prop_all)^2))
R>
R> devi <- subset(devi, n > 50)
```

We don't know the variance characteristics of this estimator, so we explore it visually by plotting n vs. deviation, Figure 1a. The linear scale shows little, except the variability decreases with sample size, but on the log-log scale, Figure 1b there is a clear pattern. It's particularly easy to see the pattern when we add the line of best fit from a robust linear model.

```
R> ggplot(data = devi, aes(x = n, y = dist) + geom_point()
R>
R> last_plot() +
R>   scale_x_log10() +
R>   scale_y_log10() +
R>   geom_smooth(method = "rlm", se = F)
```



(a) Linear scales



(b) Log scales

Figure 1: (Left) Plot of n vs deviation. Variability of deviation dominated by the sample size: small samples have large variability. (Right) Log-log plot makes it easy to see pattern of variation as well as unusually high values. Blue line is a robust line of best fit.

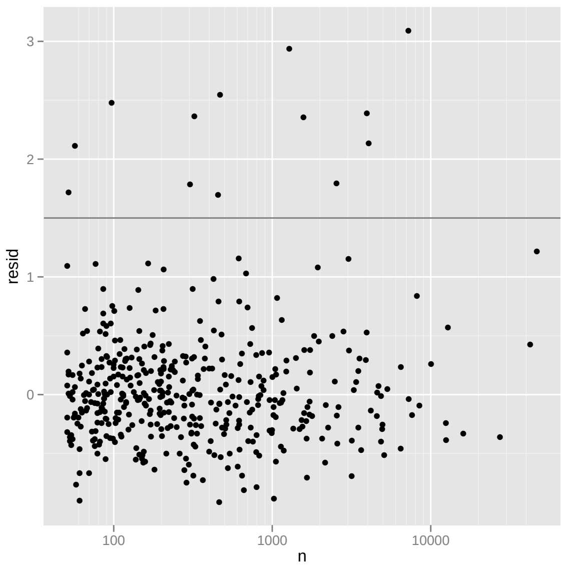


Figure 2: Residuals from a robust linear model predicting $\log(dist)$ by $\log(n)$. Horizontal line at 1.5 shows threshold for further exploration.

We are interested in points that are high relative to other causes with the same number of deaths: these are diseases most different from the overall pattern. To find these unusual points, we fit a robust linear model and plot the residuals, as shown in Figure 2. The plot shows an empty region around a residual of 1.5, so somewhat arbitrarily we select those diseases with a residual greater than 1.5. We do that in two steps: first selecting the appropriate rows from `devi` (one row per disease), and then finding the matching time course information in the original summary dataset (24 rows per disease).

```
R> devi$resid <- resid(rlm(log(dist) ~ log(n), data = devi))
R> unusual <- subset(devi, resid > 1.5)
R> hod_unusual <- match_df(hod2, unusual)
```

Finally, we plot the time courses of the unusual causes in Figure 3. We break the diseases into two plots because of the differences in variability: the top plot shows diseases with over 350 deaths and the bottom with less than 350. The causes of death fall in to three main groups: murders, drowning, and transportation related. Murders are more common at night, drowning in the afternoon, and transportation related during commute times. The pale gray line in the background shows the time course across all diseases.

```
R> ggplot(data = subset(hod_unusual, n > 350), aes(x = hod, y = prop)) +
R>   geom_line(aes(y = prop_all), data = overall, colour = "grey50") +
R>   geom_line() +
R>   facet_wrap(~ disease, ncol = 3)
```

6. Discussion

Data cleaning is an important problem, but it is an uncommon subject of study in statistics. This paper carves out a small but important subset of data cleaning that I've called data tidying: getting the dataset in the right structure to make further manipulation, visualisation and modelling easy. There is still much work to be done: as well as incremental improvements as my understanding of tidy data and tidy tools improves, there are big improvements possible from exploring alternative formulations of tidy data and tackling other data cleaning problems.

There is a chicken-and-egg problem with tidy data: if tidy data is only as useful as the tools that work with it, then tidy tools are inextricably linked to tidy data. This makes it is easy to get stuck in a local maxima where independently changing data structures or data tools does not result in an improved workflow. Breaking out of this local maxima is hard, and requires long-term concerted effort with many false starts. While I hope that this framework for tidy data is not a false start, equally, I don't see it as the final solution and I hope others will build on this framework to develop even better ways of storing data and better tools for working with it.

I have found few principles to guide the design of tidy data, which must acknowledge both statistical and cognitive factors. To date, my work has been driven by my experience doing data analysis, connections to relational database design, and introspection on the tools of data analysis. The human factors, user-centred design, and human-computer interaction communities seem like they should be able to add to this conversation, but the design of data

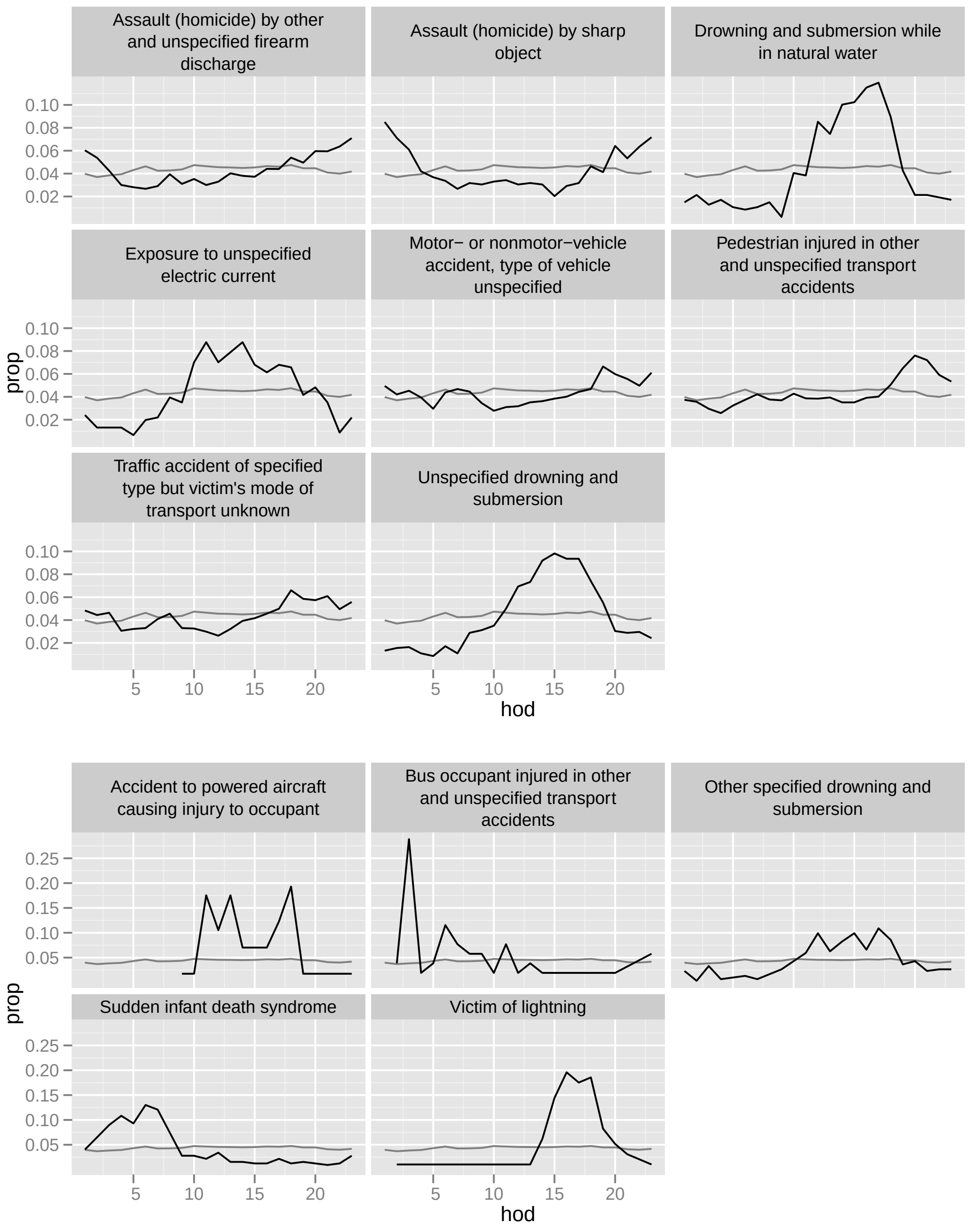


Figure 3: Causes of death with unusual time courses. Overall hourly death rate shown in grey. (Top) Causes of death with more than 350 deaths over a year. (Bottom) Causes of death with less than 350 deaths. Note that different scales are used on the y axis.

and tools to work with it has not been an active research topic. In the future, I hope to apply methodologies from these fields (user-testing, ethnography, talk-aloud protocols) to my work to build a better understanding of the cognitive side of data analysis, and how we can best design tools to support the process.

Other formulations of tidy data are possible. For example, it would be possible to construct a set of tools for dealing with values stored in multidimensional arrays. This is a common data storage format for large biomedical datasets such as microarray or fMRI data. It is also necessary for many multivariate methods whose underlying theory is based on matrix manipulation. This array-tidy format would be compact and efficient, because there are many efficient tools for working with high-dimensional arrays, even when sparse, and the format would connect more closely with the mathematical basis of statistics. This approach is taken by the Pandas python data analysis library (McKinney 2010). The biggest challenge is to connect different datasets, requiring the development of an efficient equivalent to join, and to connect to existing modelling tools without an expensive re-expression in another format. Even more interestingly, we could consider tidy tools that can ignore the underlying data representation, automatically choosing between array-tidy and dataframe-tidy formats to optimise memory usage and performance.

Apart from tidying, there are many other tasks involved in cleaning data: parsing dates and numbers, identifying missing values, correcting character encodings (for international data), matching similar but not identical values (created by typos), verifying experimental design, filling in structural missing values, not to mention model-based data cleaning that identifies suspicious values. Can we develop other frameworks to make these tasks easier?

7. Acknowledgements

This work wouldn't be possible without the many conversations I've had about data and how to deal with it statistically. I'd particularly like to thank Phil Dixon, Di Cook, and Heike Hofmann, who have put up with numerous questions over the years. I'd also like to thank the users of the **reshape** package who have provided many challenging problems, and my students who continue to challenge me to explain what I know in a way that they can understand. I'd also like to thank Bob Muenchen, Burt Gunter, Nick Horton and Garrett Grolemund who gave detailed comments on earlier drafts, and to particularly thank Ross Gayler who provided the nice example of the challenges of defining what a variable is and Ben Bolker who showed me the natural equivalent between a paired t-test and a mixed effects model.

References

Codd EF (1990). *The relational model for database management: version 2.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA. ISBN 0-201-14192-2.

Dasu T, Johnson T (2003). *Exploratory data mining and data cleaning.* Wiley-IEEE.

Inselberg A (1985). "The Plane with Parallel Coordinates." *The Visual Computer*, **1**, 69–91.

Kandel S, Paepcke A, Hellerstein J, Heer J (2011). "Wrangler: Interactive Visual Specification of Data Transformation Scripts." In *ACM Human Factors in Computing Systems (CHI)*. URL http://vis.stanford.edu/papers/wrangler.

Lakshmanan L, Sadri F, Subramanian I (1996). "SchemaSQL-a language for interoperability in relational multi-database systems." In *Proceedings of the International Conference on Very Large Data Bases*, pp. 239–250. ISSN 1047-7349.

McKinney W (2010). "Data Structures for Statistical Computing in Python." In S van der Walt, J Millman (eds.), *Proceedings of the 9th Python in Science Conference*, pp. 51 – 56.

R Development Core Team (2011). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org/.

Raman V, Hellerstein J (2001). "Potter's wheel: An interactive data cleaning system." In *Proceedings of the International Conference on Very Large Data Bases*, pp. 381–390. ISSN 1047-7349.

Sarkar D (2008). *Lattice: Multivariate Data Visualization with R*. Springer.

Wainer H (2000). *Visual revelations: Graphical tales of fate and deception from Napoleon Bonaparte to Ross Perot*. Lawrence Erlbaum.

Wegman EJ (1990). "Hyperdimensional Data Analysis Using Parallel Coordinates." *Journal of the American Statistical Association*, **85**(411), 664–675.

Wickham H (2007). "Reshaping data with the reshape package." *Journal of Statistical Software*, **21**(12), 1–20. URL http://www.jstatsoft.org/v21/i12/paper.

Wickham H (2009). *ggplot2: Elegant graphics for data analysis*. useR. Springer.

Wickham H (2010). "A layered grammar of graphics." *Journal of Computational and Graphical Statistics*, **19**(1), 3–28.

Wickham H (2011). "The split-apply-combine strategy for data analysis." *Journal of Statistical Software*, **40**(1), 1–29. URL http://www.jstatsoft.org/v40/i01/.

Wilkinson L (2005). *The Grammar of Graphics*. Statistics and Computing, 2nd edition. Springer.

Affiliation:

Hadley Wickham
Chief Scientist, RStudio
Adjunct Assitant Professor, Rice University
E-mail: h.wickham@gmail.com
URL: http://had.co.nz

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/

Volume VV, Issue II *Submitted:* yyyy-mm-dd
MMMMMM YYYY *Accepted:* yyyy-mm-dd