Tidy data

Hadley Wickham

September 23, 2011

Abstract

It's often said that 80% of the effort of analysis is in data cleaning. Despite the amount of time it takes, there has been little research on how to clean data well. This paper attempts to tackle a small, but important, subset of data cleaning: data "tidying".

"Tidy" data is easy to manipulate, model and visualise, and has a specific structure: variables are stored in columns, observations in rows, and one type of experimental unit per file. This structure makes it easy to tidy messy data, because only a small set of tools are needed to deal with a large number of messy data sets.

Tidy data is only useful in conjunction with tidy tools, tools that input and output tidy data. Tidy tools for manipulation, modelling, and visualisation are described, and the notion of tidy tools is used to critique existing messy tools.

1 Introduction

It's often said that 80% of the effort of analysis is spent just getting the data ready to analyse, that is, the process of data cleaning. 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 up, 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": getting the data in a format that is easy to manipulate, model, and visualise.

The principles of "tidy" data are inspired by databases and Codd's relational algebra (Codd, 1990), but the set algebra that powers SQL and relational databases is not always a good fit for statistical problems. The development of tidy data has been driven by my work in data analysis, where I have struggled with

many datasets organised in bizarre ways, and led to the creation of the plyr (Wickham, 2011) and reshape (Wickham, 2007) packages in R. Understanding tidy data has also been driven by my continued struggle to effectively teach data cleaning to my students.

While statisticians have done little formal work on data cleaning, computer scientists have contributed some very interesting papers. For example, Lakshmanan et al. (1996) defines a extension to SQL allowing it to operate on messy data, Raman and Hellerstein (2001) provides framework for cleaning data, and Kandel et al. (2011) develops an interactive tool that makes data cleaning easy and even enjoyable, automatically creating the code to clean data from a friendly user interface. These tools are useful, but they tend to emphasise intra-cell cleaning (i.e. separating "Smith, John" into first and last name columns) rather than the more structural approach of this paper.

The paper proceeds as follows. Section 2 begins by defining the three characteristics that make data tidy. Most real world data is not tidy, so Section 3 describes the operations needed to make messy data tidy, and illustrates the techniques with a range of real-life examples. Section 4 defines tidy tools, tools that input and output tidy data, 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 data sets.

Statistical data usually comes in a rectangular **table**, made up of **rows** and **columns**. It is usually labelled with column names, and sometimes with row names. Such data is a collection of **value**s, typically either numeric (if quantitative) or character (if qualitative). Values are grouped into **variables**, measurements of a single attribute. Multiple measurements made on the same **experimental unit** form an **observation**.

Table 1 illustrates a common data format, here used to store data about an imaginary experiment with two treatments. There are two columns and three rows (plus headers), 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 data presentation.

Data is messy or tidy depending on how rows, columns and tables are matched up with observations, variables and types of experimental unit. In tidy data:

1. Each variable forms a column,

2. Each observation forms a row,

3. Each table (or file) stores data about one class of experimental unit.

I will call other arrangements 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 John Smith's treatment a, but we don't, because generally rows should only be omitted if they are not possible due to 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 data. 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. **Measured** variables are what we actually measure in the study. Fixed variables should come first, followed by measured variables come next, each ordered so that related variables are contiguous. Rows can then be ordered so that the first id variable varies slowest, followed by the second, and so on. This is the

convention adopted by all tabular displays in this paper. This "Alabama first" (Wainer, 2000) alphabetical ordering should usually be abandoned in favour of a useful sorting when presenting data.

The following sections describe how to make messy data tidy, and then justify my definition of tidy data by showing how it is easier to work with.

3 Tidying messy data

Real data violates the three precepts of tidy data in almost every way imaginable (and many that you can't imagine in advance!). Surprisingly, most messy data, including types of messiness not explicitly described below, can be tidied with a small set of tools: "melting", string splitting, and "casting". In this section, I will focus on some of the most common problems, each illustrated with a real dataset that I have struggled with.

In my experience, the five most common problems are:

- 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 experimental unit stored in the same table
- experimental unit stored in multiple tables

The complete datasets and the code used to tidy them, are available online at `https://github.com/hadley/tidy-data`.

3.1 Column headers are values, not variable names

A common example of messy data is tabular data designed for presentation, where variables form both the rows and columns, and column headers are values, not variable names. While I call this type of data messy here, 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 typical dataset of this form, which explores the relationship between income and religion in the US. It comes from a pdf[1] produced by the Pew Research Center, an American think-tank that collects data (through surveys and other instruments) on attitudes to topics from religion to the internet.

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

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
Don't 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'

To tidy up this type of data we need to "melt", or stack it, turning columns into rows. This makes "wide" data "long" or "tall", but I will avoid those terms because they are imprecise. Melting is parameterised by a list of columns that are variables and should be kept in columns (`cvars`), or equivalently by the columns that should be stacked into rows (`crows`). It works by adding a new "variable" column to the dataset and stacking the repeated `cvar` values in a "value" column. I call this type of data **molten**.

We melt the Pew data with `cvar = religion` to yield Table 4. The variable column has been renamed to `income`, and the value column to `freq`, to better reflect their roles in this data. The molten form is tidy in this simple case: each column represents a variable and each row represents an experimental unit, 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 4: The first ten rows of the tidied Pew survey data on income and religion. The variable column has been renamed to `income`.

Another common use of this data format is recording regularly spaced observations over time. For example, the billboard dataset shown in Table 5 records the date a song first entered the billboard top 100. The rank of each song over time is recorded in 75 variables, `week1` to `week75`. This form of storage 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.inverted	track	time	genre	date.entered	week1
2000	2 Pac	Baby Don't Cry	4:22	Rap	2000-02-26	87
2000	2Ge+her	The Hardest Part Of Breaking Up	3:15	R&B	2000-09-02	91
2000	3 Doors Down	Kryptonite	3:53	Rock	2000-04-08	81
2000	98°	Give Me Just One Night	3:24	Rock	2000-08-19	51
2000	A*Teens	Dancing Queen	3:44	Pop	2000-07-08	97
2000	Aaliyah	I Don't Wanna	4:15	Rock	2000-01-29	84
2000	Aaliyah	Try Again	4:03	Rock	2000-03-18	59
2000	Adams, Yolanda	Open My Heart	5:30	Gospel	2000-08-26	76

Table 5: The first eight billboard top hits for 2000. Other columns not shown are `week2`, `week3`, ..., `week75`.

Melting the data with `cvar = (year, artist.inverted, track, time, genre, data.entered)` yields Table 6. This data has been cleaned a little in addition to the tidying: I have added a date variable that combines the date the song entered the billboard with the week number.

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

Table 6: Tidied billboard data. The `date` column does not appear in the original table, but can be computed from `date.entered` and `week`. Other columns not shown: `time`, `genre`.

3.2 Multiple variables stored in one column

After melting, it often happens that the "variable" column is a combination of multiple underlying variables. This is illustrated by the tb dataset, a sample of which is shown in Table 7. Each column represents the

counts of tb in each country in each year for a fixed value of `sex` (m, f) and `age` (0–14, 15–25, 25–34, 35–44, 45–54, 55–64, unknown).

iso2	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 7: Original tb data. 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.

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 data, 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 8 shows the results of melting the tb data then splitting the "variable" column into two real variables: `age` and `sex`.

iso2	year	variable	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

iso2	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

Table 8: (Left) Molten tb data. (Right) Tidy data with "variable" variable broken up in to sex and age variables.

Storing the data in this form resolves another problem in the original data: it's more informative to

compare rates, not counts, between countries, but to compute rates we need to know the population in each category. In the original data format, there is no easy way to record this information, and it has to be stored in a separate table. This makes it difficult to correctly match populations to disease counts. In tidy form, adding variables to record total population and disease 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. The weather data in Table 9 has variables in individual columns (`id`, `year`, `month`), spread across columns (`day`, d1–d31) and across rows (`tmin`, `tmax`) (minimum and maximum temperature). The `element` column is not a variable: it stores the names of variables. Melting the data with `cvar = (id, year, month, element)` gets us to Table 10. For presentation, we have dropped the missing values, making them implicit rather than explicit. It's ok to do this for daily data because we know how many days are in each month and can easily reconstruct the explicit missing values.

This data is mostly tidy, but we have two variables stored in rows: `tmin` and `tmax`, the type of observation. 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. 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).

id	year	month	element	d1	d2	d3	d4	d5	d6	d7	d8	d9	d10
MX000017004	2010	1	tmax										
MX000017004	2010	1	tmin										
MX000017004	2010	2	tmax		273	241							
MX000017004	2010	2	tmin		144	144							
MX000017004	2010	3	tmax					321					345
MX000017004	2010	3	tmin					142					168
MX000017004	2010	4	tmax										
MX000017004	2010	4	tmin										
MX000017004	2010	5	tmax										
MX000017004	2010	5	tmin										

Table 9: Original weather data. There is a column for each possible day in the month. Columns `d11` to `d31` have been omitted to conserve space.

id	year	month	day	element	value
MX000017004	2010	1	30	tmax	278
MX000017004	2010	1	30	tmin	145
MX000017004	2010	2	2	tmax	273
MX000017004	2010	2	2	tmin	144
MX000017004	2010	2	3	tmax	241
MX000017004	2010	2	3	tmin	144
MX000017004	2010	2	11	tmax	297
MX000017004	2010	2	11	tmin	134
MX000017004	2010	2	23	tmax	299
MX000017004	2010	2	23	tmin	107

Table 10: Molten weather data. This is almost tidy, but the `element` column contains names of variables, not values. Missing values are dropped to conserve space.

id	year	month	day	tmax	tmin
MX000017004	2010	1	30	278	145
MX000017004	2010	2	2	273	144
MX000017004	2010	2	3	241	144
MX000017004	2010	2	11	297	134
MX000017004	2010	2	23	299	107
MX000017004	2010	3	5	321	142
MX000017004	2010	3	10	345	168
MX000017004	2010	3	16	311	176
MX000017004	2010	4	27	363	167
MX000017004	2010	5	27	332	182

Table 11: Tidy weather data. 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	98°	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 data split up into song (left) and rank (right) datasets. First 15 rows of each dataset shown; `genre` omitted from song data, `week` omitted from rank data.

3.4 Multiple types in one table

Datasets often involve data collected at multiple levels, on different types of experimental unit. During tidying, each type of experimental 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. If you're not familiar with normalisation, it can be worthwhile to learn a little about it. There are many good tutorials available online - I found `http://phlonx.com/resources/nf3/` after a few minutes of searching. You certainly don't need to become an expert, as most statistical databases only need a small amount of normalisation, but it is extremely helpful for identifying inconsistencies in your data.

For example, the billboard data described in Table 6 actually contains data on two experimental units: the song and its rank in each week. The easiest way to spot this is to note that facts about the song (e.g. `artist` and `time`) are repeated multiple times. The billboard data should be broken down into two datasets, a song dataset which stores artist, song name, and a ranking dataset which matches the song and week to the ranking, as shown in Table 12. You could also imagine a week dataset which would record background information about the week, maybe the total number of song sales 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 data back into one table. This is acceptable because each piece is consistent, and so the entire data frame will also be consistent.

3.5 One type in multiple tables

It's also common to find data about a single type of experimental unit spread out over multiple tables, or multiple files. These tables are often split up by some other variable, so each file represents a single year, person, or other condition. 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. This is three or four lines of R code. 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 names tables provided by the US Social Security Administration and combines them into a single file.

A more complicated situation occurs when the data storage format 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`, and shows 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 have been four major file formats, and many minor variations, making tidying this data a considerable challenge.

4 Tidy tools

Once you have tidy data, 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 data as input and return tidy data 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 types of data.

Tools can be messy for two reasons: either they take messy data as input (messy-input tools) or they produce messy data 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 data, 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, 2010), 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 variation (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.

In R, filtering and transforming are performed by the base R functions `subset` and `transform`. These are both 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 an alternative implementation of `transform`, `mutate`, as well as 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. 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.

Some aggregations occur so frequently they deserve their own optimised implementations. One such operation is (weighted) counting. Base R provides the `table` function for this, but it is not output-tidy: it returns a multidimensional array. An tidy alternative is the `count` function from `plyr`, which returns a tidy

dataset with a column for each of the input variables plus a new variable `freq`, which records the number of records in each category.

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 are very hard to detect.

4.2 Visualisation

Tidy visualisation tools need only to be input-tidy as their output is graphic, not data. Domain specific languages work particularly well for the visualisation of tidy data because they can describe a visualisation as a mapping between variables in the data 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: the base `plot` function, the lattice family of plots (Sarkar, 2008) and the plots of ggplot2 (Wickham, 2009). Some specialised tools exist for visualising messy data. Some base R functions like `barplot`, `matplot`, `dotchart`, and `mosaicplot`, work with messy data where variables are spread out over multiple columns. More generally, parallel coordinates plots (Wegman, 1990; Inselberg, 1985) can be used to create time series plots for messy data where each time is recorded in a column.

4.3 Modelling

Modelling is the inspiration that has driven much of this work, because most modelling tools work best with tidy data. Every statistical language has a way of describing a model as a connection between different variables, a domain specific language for connecting 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`

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 on to the appropriate basis functions.

There have been some attempts to adapt modelling functions for specific types of messy data. 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 `week1-week76 = track` instead of `rank = week * track` on the tidy data (provided week is labelled as a categorical variable).

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 data.

5 Case study

The following case study attempts to illustrate how tidy data makes a data analysis easier, combining manipulation, visualisation and modelling. The case study uses data on mortality in Mexico, with the goal of finding diseases that have notably different time course within a day. The original data and cleaning code can be found at `https://github.com/hadley/mexico-mortality`. It illustrates some of the many challenges apart from tidying that data cleaning requires: getting variables into the right formats, removing clearly impossible values, and finding good sources of data.

Table 13 shows where we want to end up. For each disease for each hour of the day (`hod`) we have the number of deaths (`freq`), the proportion of deaths from that disease in that hour (`prop`), and the overall deaths and proportion of death in that hour (`freq_all` and `prop_all`).

The case study uses R, but even if you're not familiar with R or the exact packages I use, try reading the code. It is consistent, and fairly simple, and you should be able to pick up on the important patterns. The code starts by aggregating the full dataset to the minimal data that will allow us to answer our question: the number of deaths per hour per disease.

```
# Count the the number of deaths in each hour of the day for each
```

hod	cod	disease	freq	prop	freq_all	prop_all
1	A01	Typhoid and paratyphoid fevers	3	0.06	20430	0.04
1	A02	Other salmonella infections	3	0.05	20430	0.04
1	A04	Other bacterial intestinal infections	7	0.05	20430	0.04
1	A05	Other bacterial foodborne intoxications, not...	1	0.05	20430	0.04
1	A06	Amebiasis	2	0.02	20430	0.04
1	A09	Diarrhea and gastroenteritis of infectious...	112	0.04	20430	0.04
1	A15	NA	2	0.01	20430	0.04
1	A16	Respiratory tuberculosis, not confirmed...	53	0.03	20430	0.04
1	A17	Tuberculosis of nervous system	2	0.02	20430	0.04
1	A18	Tuberculosis of other organs	5	0.05	20430	0.04
1	A19	Miliary tuberculosis	8	0.05	20430	0.04
1	A37	Whooping cough	1	0.10	20430	0.04
1	A41	Other septicemia	180	0.04	20430	0.04
1	A46	Erysipelas	1	0.04	20430	0.04
1	A48	Other bacterial diseases, not elsewhere...	1	0.07	20430	0.04

Table 13: First fifteen rows of the `hod2` data frame.

```
# cause of death.  Creates the hod, cod and freq columns.
hod2 <- count(deaths, c("hod", "cod"))

# Remove missing observations
hod2 <- subset(hod2, !is.na(hod))

# Combine with data set that gives disease names.  Adds the disease column.
hod2 <- join(hod2, codes)

# Convert from counts to proportions within each disease (cod).  Adds the
# prop column. ddply is a by operator, performing group-wise transformations,
# summaries or subsets depending on its third argument.
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 data.

```
# Work out number of people dying each hour by breaking down hod2
# by hod, and summarising by computing total over each cause of death
overall <- ddply(hod2, "hod", summarise, freq_all = sum(freq))
```

```
# Work out overall proportion of people dying in each hour
overall <- mutate(overall, prop_all = freq_all / sum(freq_all))

# Join overall data with individual data to make it easier to
# compare the two.  Adds the freq_all and prop_all columns.
hod2 <- join(hod2, overall, by = "hod")
```

Next we compute a measure of deviation between each disease and the overall trend: a mean squared deviation, summarising the hod dataset by cause of death. We don't know the variance characteristics of this estimator, so we explore it empirically in Figure 1 by plotting n vs. deviation. On the log-log scale there is a clear linear relationship with slope -0.4, suggesting that variance is proportional to $n^{-0.4}$. We are interested in the outliers who have relatively high values relative to the variability: these are diseases which are different from the norm.

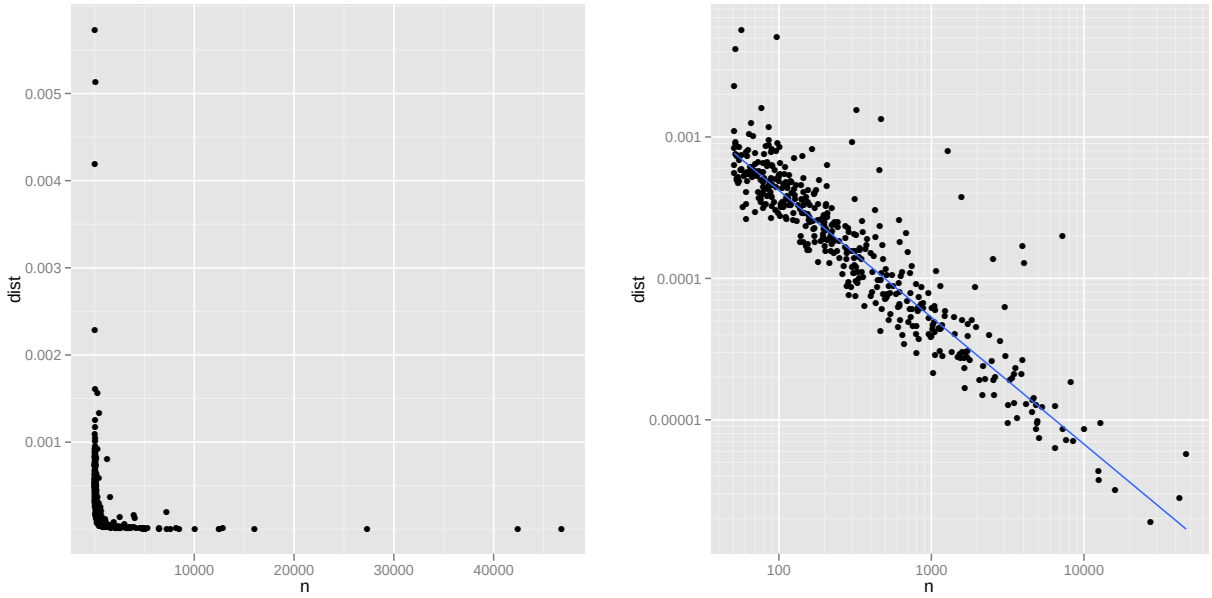

Figure 1: (Left) Plot of n vs deviation. Variability of deviation dominated by the sample size: small samples have large variability. (Right) Log-log plot makes it easy to see pattern of variation as well as unusually high values. Blue line is a robust line of best fit.

```
# Compute deviation for each disease by summarising with the number
# of deaths, and mean squared deviation from the overall trend.
```

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

# Only look at diseases with more than 50 deaths (~2/hour)
devi <- subset(devi, n > 50)
```

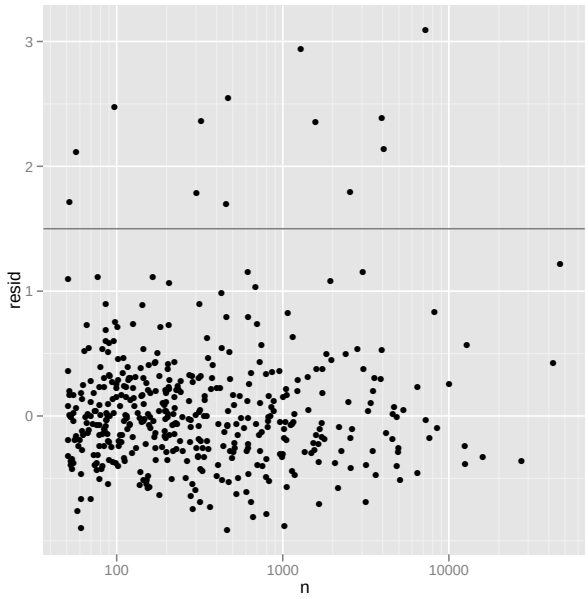

Figure 2: Residuals from a robust linear model predicting $\log(dist)$ by $\log(n)$. Horizontal line at 1.5 shows threshold for further exploration.

Figure 2 shows a plot of sample size vs. residual for each disease. The plot shows a break around 1.5, so somewhat arbitrarily we select those diseases with a residual greater than 1.5 and explore their time courses 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 following code performs the cycle of modelling, manipulating and visualising the produces the final results.

```
# MODEL: Compute residuals from robust linear model on log-log data
devi$resid <- resid(rlm(log(dist) ~ log(n), data = devi))
# Select diseases with residuals bigger than 1.5
```

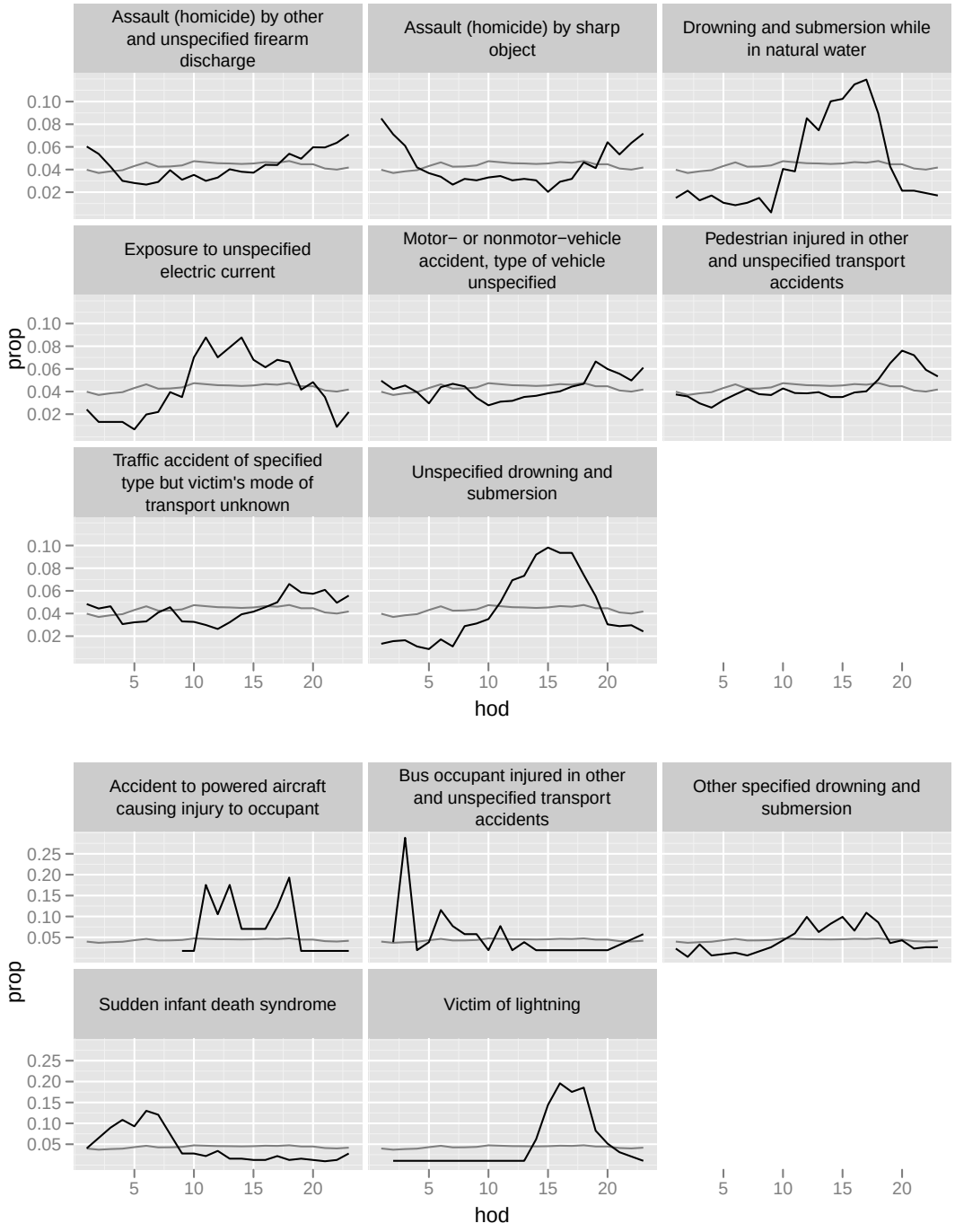

Figure 3: Causes of death with unusual time courses. Overall hourly death rate shown in grey. (Top) Causes of death with more than 350 deaths over a year. (Bottom) Causes of death with less than 350 deaths. Note that different scales are used on the y axis.

```
unusual <- subset(devi, resid > 1.5)

# MANIPULATE: Extract matching time course data for unsual diseases, broken
# down into two groups: common diseases and rare disease.
hod_unusual_big <- match_df(hod2, subset(unusual, n > 350))
hod_unusual_sml <- match_df(hod2, subset(unusual, n <= 350))

# VISUALISE: hod on x-axis, prop on y-axis, display by line, with pale gray
# line showing overall death rate.
ggplot(hod_unusual_big, aes(hod, prop)) +
  geom_line(aes(y = prop_all), data = overall, colour = "grey50") +
  geom_line() +
  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 data 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. My experience is that the tools we use strongly influence the way we think, and my work developing this tidy framework has been slow and incremental, building over the up seven years I have been seriously working with data. Each improvement in tools helped me think better about how data should be stored, and each insight into how data should be stored helped me create better tools. I expect this process to continue in the future.

Other formulations of tidy data are possible. For example, it would be possible to construct a set of tools for dealing with data 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. 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?

References

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

A. Inselberg. The plane with parallel coordinates. *The Visual Computer*, 1:69–91, 1985.

Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. Wrangler: Interactive visual specification of data transformation scripts. In *ACM Human Factors in Computing Systems (CHI)*, 2011. URL `http://vis.stanford.edu/papers/wrangler`.

L.V.S. Lakshmanan, F. Sadri, and I.N. Subramanian. SchemaSQL-a language for interoperability in relational multi-database systems. In *Proceedings of the International Conference on Very Large Data Bases*, pages 239–250, 1996.

R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2010. URL `http://www.R-project.org`. ISBN 3-900051-07-0.

V. Raman and J.M. Hellerstein. Potter's wheel: An interactive data cleaning system. In *Proceedings of the International Conference on Very Large Data Bases*, pages 381–390, 2001.

Deepayan Sarkar. *Lattice: Multivariate Data Visualization with R*. Springer, 2008.

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

Edward J. Wegman. Hyperdimensional data analysis using parallel coordinates. *Journal of the American Statistical Association*, 85(411):664–675, 1990.

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

Hadley Wickham. *ggplot2: Elegant graphics for data analysis*. useR. Springer, July 2009.

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

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

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