Chapter 6, Part 1: Tidy Data

- In statistics class examples, data may be presented to students in a form that is all ready for them to analyze.
- In the real world, data may be stored in formats that are not immediately ready for data analysis.
- In this chapter, we will learn some data wrangling skills that are valuable for working with data of unusual types or in unusual forms, so that they can be prepared for analysis.

Example: Long and Wide Formats

- The *Gapminder* database is a well-known collection of data measuring various variables on countries across time.
- Consider an example data set on HIV prevalence in several countries over a set of different years.
- We could view this data set in "wide" format, in which each row is a country, and the columns represent the prevalence of HIV in that country for the series of years.
- Or we could view the data set in "long" format, where there is a different row for each country-year combination, and there are fewer columns.
- See examples for the HIV dataset.

Pros and Cons of Long and Wide Formats

- The wide format is a "spreadsheet-style" format, in which we can more easily take in the whole data set at one glance.
- So visual inspection is easier with the wide format, since the long format is typically too long to fit on one screen – we'd have to keep scrolling down to see it all.
- However, it turns out that for many types of data analysis, it is more convenient for the input data to be in long format.
- And when working with multiple tables, it is easier if they are in long format.
- And it is easier to add variables to an existing data set if they are in long format rather than wide, since it just entails adding another column (as opposed to adding another sheet).
- In the era of big data, being able to see all the data at once is often not realistic anyway.

Data Management Using Programming Rather than Click-and-Drag Steps

- Don't do your preprocessing of raw data using click-and-drag operations within spreadsheet applications (like Excel).
- It's very important that when you manage and preprocess your data, that you do it through programming steps.
- This separates the raw data from the analysis.
- It makes your work reproducible — if the code is there, you and others can do those steps later on that raw data file and other similar/updated raw data files, without starting over from scratch.
- Getting your data into a *tidy data* format is an important concept in data preprocessing.

Tidy Data

- Tidy data are presented in an array of rows and columns.
- The rows, called cases or observations, each refer to a specific, unique, and similar sort of thing, which we might call an *item*. The columns, called variables, each have the same sort of value recorded for each row.
- The variables are characteristics which are measured or observed for each item.
- When data are in tidy form, it is easy to perform steps that summarize useful information about the data.
- See the simple example on the `babynames` data set.

Example of a Data Set Not in Tidy Form

- Look at the summarized Minneapolis election data in Figure 6.1.
- Why is it not in tidy form?
- We must put it in tidy form before we write code to extract summaries of information, produce graphical displays of the data, etc.
- *Data wrangling* involves transforming information in a table into a form where the information is explicitly seen.
- We learned several verbs for data wrangling in Chapter 4.
- Data that are coming from different sources can easily be combined (such as using the joins we learned) if they are in tidy form.

Figure 6.1

	A	B	E	F	G	H	I	J
1			City of Minneapolis Statistics					
2			General Election November 5, 2013					
3	Ward	Precinct	Voters Registering by Absentee	Total Registrations	Voters at Polls	Absentee Voters	Total Ballots Cast	Total Turnout
4	City-Wide Total		708	6,634	75,145	4,954	80,099	33.38%
5								
6	1	1	3	28	492	27	519	27.23%
7	1	2	1	44	836	56	892	31.71%
8	1	3	0	40	905	19	924	38.87%
9	1	4	5	29	768	26	794	36.62%
10	1	5	0	31	683	31	714	37.46%
11	1	6	0	69	739	20	759	32.62%
12	1	7	0	47	291	8	299	15.79%
13	1	8	0	43	415	5	420	30.55%
14	1	9	0	42	596	25	621	25.42%
15	Ward 1 Subtotal		9	373	5,725	217	5,942	30.93%
16								
17	2	1	1	63	1,011	39	1,050	36.42%
18	2	2	5	44	679	37	716	50.39%
19	2	3	4	48	324	18	342	18.88%
20	2	4	0	53	117	3	120	7.34%
21	2	5	2	50	495	26	521	25.49%
22	2	6	1	36	433	19	452	39.10%
23	2	7	0	39	138	7	145	13.78%
24	2	8	1	50	1,206	36	1,242	47.90%
25	2	9	2	39	351	16	367	30.56%
26	2	10	0	87	196	5	201	6.91%
27	Ward 2 Subtotal		16	509	4,950	206	5,156	27.56%
28								
29	3	1	0	52	165	1	166	7.04%

Figure: Figure 6.1 from MDSR textbook

Variables and Cases

- A *variable* is an observed characteristic of an observation that varies from case to case.
- Variables could be categorical or numerical.
- A case is a row in the data set.
- Often it represents an individual person or thing, but with *repeated-measures* data, measurements on the same individual could be listed as several cases.
- See the ballot data (Table 6.4) and the road-race data (Table 6.5) for examples.

Codebooks for Data Sets

- A *codebook* is a documentation for a data set.
- It gives information about what the variables mean, how the data were collected, what the different categories are for categorical variables, etc.
- It gives more information than simple variable names can.
- It's always good practice to include such documentation about data sets so that humans can understand them.
- Codebooks for data tables in built-in packages are available from the `help` function in R.

Reshaping Data

- Recall the differences between data in *wide* format and in *long* format, and the advantages of each.
- With data in wide format, it is simple to create variables that compare response values across time periods.
- With data in long format, it can be easier to add new variables or combine information in multiple tables.
- There are some useful R functions for changing formats from wide to long or from long to wide.

Blood Pressure Data Set Example

- See the blood pressure data sets in two formats: `BP_wide` and `BP_narrow`.
- The `pivot_longer` function in the `tidyr` package takes a wide-format data set and converts it to a long-format data set.
- The `pivot_wider` function in the `tidyr` package takes a long-format data set and converts it to a wide-format data set.

Arguments in `pivot_wider`

- There are two key arguments in the `pivot_wider` function:
- `values_from` argument: name of the variable in the narrow format to be divided up into multiple variables in the resulting wide format.
- `names_from` argument: name of the variable in the narrow format that identifies, for each case, which column in the wide format will receive the value.

Arguments in `pivot_longer`

- The key arguments in the `pivot_longer` function:
- `names_to` argument: defines variables from the wide form that will become the categorical levels in the narrow form.
- `values_to` argument: the variable that is to hold the values in the variables being gathered – it should reflect what those values actually represent.
- With `pivot_longer`, we can specify which variables (like subject IDs or names) should remain a variable in the narrow format.
- See examples of transforming the blood pressure data set.

Working with List-Columns

- A common task with repeated-measures data is to perform summary statistics on each subject (e.g., calculating within-subject means for each subject).
- Using *list-columns*, we can obtain a data set whose rows are the groups (i.e., subjects), but which also have the individual measurement values, which we could do further analysis on.
- The `nest` function will collapse all the *ungrouped* variables into a *tibble*, which is a simple type of data frame.
- A new variable is created which is a *list* – note that list objects in R can be made up of elements that differ in dimension and type.
- A variable in a data frame that has the type *list* is called a *list-column*.
- The elements in the created list column are tibbles (which may differ in size) containing the ungrouped variable values.

Getting Information Out of List-Columns

- How can we access the ungrouped variable values in the list-column?
- The `pull` function can pull out these values, and using the `map` function, we can apply the `pull` function to a specific measured variable.
- The result will be a list object, and we can again use the `map` function to perform calculations on **this** list.
- The `unnest` function can convert the nested list back into numeric or ungrouped data.
- See the example of pulling out systolic blood pressure values on the course website.
- The textbook has a short example of using `pivot_wider` on the `babynames` data frame when analyzing gender-neutral names.

Naming Conventions in R

- When we create an object in R, like a vector, a data frame, its columns, a function, etc., we give it a name.
- R has some rules about what kinds of names are allowed to be given to objects.
  1. The name cannot start with a digit.
  2. The name cannot have any punctuation except the underscore (_) or the period (.).
  3. R names are case-sensitive, so `Mydata` and `mydata` would **NOT** refer to the same object.
- It is recommended to use underscores rather than periods in user-defined names, to avoid conflicting with internal R functions, which often have periods.
- The creators of the tidyverse developed a *style guide* with recommended usages and conventions, which the book typically follows.
- For multiword object names, they use *snake_case*, which uses all lowercase letters and separates words with underscores.