7
Wrangling and Data Verbs

When data are in glyph-ready form it is straightforward to construct data graphics using the concepts and techniques in Chapters 5. First, you choose an appropriate sort of glyph: dots, stars, bars, countries, etc. Next, select which variables are to be mapped to the various aesthetics for that glyph. Then let the computer do the drawing.

On occasion, data will arrive in glyph-ready form. More typically, you have to *wrangle* the data into the glyph-ready form appropriate for your own purpose.

WRANGLE: Transforming data from one or more data tables into a glyph-read format. The literal meaning of "wrangle" is to herd or round-up animals for some purpose such as bringing them to market. "Data wrangling" is a metaphor for putting data into a form suitable for the data visualization or analysis that's desired.

7.1 Examples of Wrangling

EXAMPLE: COUNTING THE BALLOTS. Table 7.1 records the choice of each of 80101 individual voters in the 2013 mayoral election in Minneapolis, MN, USA.

The primary use for data like Table 7.1 is to determine how many votes were given to each candidate Counting the votes for each candidate is a simple wrangling task that transforms the ballot data (Table 7.1) into a glyph-ready form (Table 7.2 right) that makes the answer obvious. The count information is still latent in the raw ballot data; it is not yet in a form where it is easily seen.

The instructions for carrying out the wrangling are simple to give in English: Count the number of ballots that each candidate received and sort from highest to lowest. To carry out the process on a computer, you need to express this idea in a form the computer can carry out.

Precinct	First	Ward
P-10	BETSY HODGES	W-7
P-06	BOB FINE	W-10
P-09	KURTIS W. HANNA	W-10
P-05	BETSY HODGES	W-13
P-01	DON SAMUELS	W-5
	... and so on for 80,101 rows	

Table 7.1: The choices of each voter in the mayoral election. See `Minneapolis2013` in the `DataComputing` package.

candidate	count
BETSY HODGES	28935
MARK ANDREW	19584
DON SAMUELS	8335
CAM WINTON	7511
JACKIE CHERRYHOMES	3524
... and so on for 38 rows	

Table 7.2: The number of votes for each candidate. This has been wrangled from Table 7.1 into a glyph-ready form, that is, a form in which the sought after information is readily seen.

7.2 Planning your wrangling

Before you start wrangling data, it's crucial to envision what the goal is to be. In general, the purpose of wrangling is to take the data you have at hand and put it into glyph-ready form. But glyph-ready for what? This is something you, the data analyst, need to determine based on your own objectives.

age	sex	smoker
2	female	no
77	male	no
10	female	no
1	male	no
49	male	yes
... and so on for 29,375 rows		

Table 7.3: The `NCHS` data from the `DataComputing` package. In `NCHS`, the case is an individual person.

Example: Demographic patterns of smoking Table 7.3 has variables indicating each person's age, sex, and whether he or she smokes..

Suppose you want to use `NCHS` to explore the links between smoking and age. Often you will have some kind of presentation graphic in mind. Making an informal sketch like Figure 7.1 can be a helpful way to chart a path for data wrangling. The sketch can be based on your imagination of what the data might show. Even so, the sketch contains important information. For instance, from the sketch you can determine what a single glyph represents. As always, each glyph will be one case in the glyph-ready data. In this sketch, the glyph describes a *group* of people — people of one sex in one age group. This differs from the cases in `NCHS`: individual people.

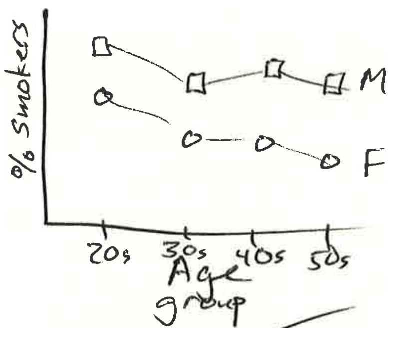
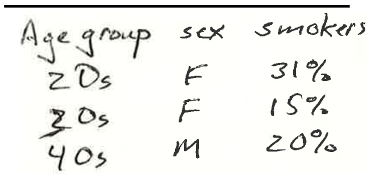


Figure 7.1: A made-up depiction of trends in the fraction of people of different ages who smoke.

To make your goal for data wrangling even more explicit, rough out the form of the glyph-ready data table, as in Table 7.4. Don't worry about calculating precise data values; the computer will do that after you have implemented your data wrangling plan.

The glyph-ready data for that graph will has a form like Table 7.4. That's glyph-ready form is your target.



Age group	sex	smokers
20s	F	31%
30s	F	15%
40s	M	20%

Table 7.4: A sketch of the form of glyph-ready data corresponding to Figure 7.1. For each aesthetic in Figure 7.1 there is one variable in the glyph-ready table.

Once you have a target glyph-ready format in mind, use plain English to plan the individual data wrangling steps. For instance, a plan for wrangling Table 7.3 into Table 7.4 might look like this:

1. Turn the age in years into a new variable, the age group (in decades).
2. Drop the people under 20-years old.
3. Divide up the table into separate groups for each age decade and sex.
4. For each of those groups, count the number of people and the number of people who smoke. Divide one by the other to get the fraction who smoke.

Each of these step would be easy enough to do by hand (if you had the time and patience to work through 31126 cases in `NCHS`). To get the computer to do the work for you, you have to be able to describe each process to the computer. The next sections present a framework for describing wrangling operations that can works for both the human describing the wrangling and the computer that will carry out the calculations.

7.3 *A Framework for Data Wrangling*

Over the last half century, researchers have identified a small set of patterns that can be used to describe a wrangling process. Because this set is small, it's feasible for you to learn quickly what you need to describe the wrangling you have planned.

As you know, *doing something* in R is accomplished using functions. A function takes one or more inputs (the "arguments") and returns an output. In thinking about data wrangling, it helps to consider these potential forms for inputs and outputs:

- A **data table**, a collection of variables and cases.
- A **variable**, just a single one from the collection in a data table.
- A **scalar**, a single number or character string.

There are three broad families of functions involved in data wrangling. The families differ in what form of input they take in and what form they return.

1. **Reduction functions** take a variable as input and return a scalar.
2. **Transformation functions** take one or more existing variables as input and return a new variable.
3. **Data verbs** take an existing data table as input and return a new data table.

Each step in data wrangling involves a data verb and one or more reduction or transformation functions.

Reduction Functions

Reduction functions summarize or reduce a variable to scalar form. You are likely familiar with several reduction functions:

- `mean()` — find a single typical value
- `sum()` — add up numbers into a total
- `n()` — find how many cases there are

Some other frequently used reduction functions:

- `min()` and `max()` — find the smallest and largest value in a variable
- `median()`, `sd()` and other functions used in statistics
- `n_distinct()` — how many different levels are there among the cases

Transformation Functions

Transformation functions take one or more variables as input and return a new variable. In contrast to reduction functions, which produce a single number — a scalar — by combining all the cases, transformation functions produce a result for each individual case.

Often, transformations are mathematical operations, for instance:

- `weight / height`
- `log10( population )`
- `round( age )`

Other transformation functions include numerical and character comparisons, as in Table 7.5.

Function	Meaning	Example
`<`	less than	`age < 21`
`>`	greater than	`age > 65`
`==`	matches exactly	`age == 39`
`%in%`	is one of	`age %in% c(18, 19, 20)`
	For character strings	
`<`	alphabetically before	`name < "ad"`
`>`	alphabetically after	`name > "ac"`
`==`	matches exactly	`name == "Jon"`
`%in%`	is one of	`name %in% c("Abby", "Abe")`

Table 7.5: Some of the functions used to compare values.

Another important operation, `ifelse()` allows you to translate each value in a variable to one of two values, depending on the result of a comparison. For instance

```
ifelse(age >= 18, "voter", "non-voter")
```

Data Verbs

Data verbs carry out an operation on a data table and return a new data table. Some data verbs involve the modification of variables, the creation of new variables, or the deletion existing ones. Other data verbs involve changing the meaning of a case or adding or deleting cases.

In addition to a data table input, each data verb takes additional arguments that provide the specifics of the operation. Very often, these specifics involve reduction and transformation functions.

There are about a dozen data verbs that are commonly used. You can start with these two:

- summarise() — turns multiple cases into a single case using reduction functions. "Aggregate" is synonym for "summarise."
- group_by() — modifies the action of reduction functions so that they give a single value for different groups of cases in a data table.

To illustrate the uses of summarise(), look at the WorldCities data table (in the DataComputing package) with information about the most populous cities in each country. Table 7.7 is a subset of the variables:

name	population	country	latitude	longitude
Shanghai	14608512	CN	31.22	121.46
Buenos Aires	13076300	AR	-34.61	-58.38
Mumbai	12691836	IN	19.07	72.88
Mexico City	12294193	MX	19.43	-99.13
Karachi	11624219	PK	24.91	67.08
Istanbul	11174257	TR	41.01	28.95
Delhi	10927986	IN	28.65	77.23
Manila	10444527	PH	14.60	120.98
Moscow	10381222	RU	55.75	37.62
Dhaka	10356500	BD	23.71	90.41
	... *and so on for 23,018 rows*			

Table 7.7: World Cities

A simple summary of the data is a count of the number of cities.

```
WorldCities %>%
  summarise(count = n())
```

count
23018

The reduction verb is n(). The expression count = n() means that the output data table should have a variable named count containing the results of n().

Perhaps you want to know the total population in these cities, or the average population per city or the smallest city in the table, as in Table 7.8.

```
WorldCities %>%
  summarise(averPop  = mean(population, na.rm=TRUE),
            totalPop = sum(population, na.rm=TRUE),
            smallest = min(population, na.rm=TRUE))
```

averPop	totalPop	smallest
112624.76	NA	0

Table 7.8: Summary statistics of the population of world cities.

The average city on the list has about 110,000 people. The total population of people living in these cities is about 2.6 billion — a bit more than one-third of the world population. The smallest city has . . . zero people! Evidently, the WorldCities data table has cases that are not really cities.

Some things to notice about the use of summarise():

The n() function doesn't take any arguments.

- The chaining syntax, `%>%` has been used to pass the first argument to `summarize()`. This will be useful later on, when there is more than one step in a transfiguration. It's OK to *end* a line with `%>%`, but **never** start a line with it.
- The output of `summarise()` is a data table. (In this example the output has only one case, but it is still a data table.)
- `summarise()` takes named arguments. The name of an argument is taken as the name of the corresponding variable created by `summarise()`.

group_by

The `group_by()` data verb sets things up so that other data verbs will perform their action on a group-by-group basis. For instance, Figure 7.9 shows the number of cities in `WorldCities` broken down by country.

```
WorldCities %>%
  group_by( country ) %>%
  summarise( count=n() )
```

country	count
AD	2
AE	12
AF	50
AG	1
AI	1
... and so on for 243 rows	

Table 7.9: The number of cities in each country listed in `WorldCities`.

The `group_by()` verb should always be followed by another verb — `summarise()` in the above example. `group_by()` is the way to indicate to the following verbs that reduction operations should be performed on a group-wise basis.

Using `group_by()` along with `summarise()` and `n()`, provides a basis for counting up the number of cases in groups and subgroups, or for calculating group-wise statistics.

Note that the functions used within arguments to `summarise()` — functions such as `n()`, `max()`, `sum()`, etc. — are **not** data verbs. A data verb takes a data table as input and returns a transfigured data table as output. In contrast, the reduction functions, `n()`, `sum()`, and so on, take a variable as input and return a *single number* as output.

Is That All?

The `summarise()` and `group_by()` data verbs are team players; they work best in combination with other data verbs. Once you learn those data verbs, particularly `filter()` and `mutate()`, you'll be able to carry out many more operations. The richness of the data-verb system comes from the ways the different verbs can be combined together.

EXAMPLE: SUMMARIES IN HEALTH-RELATED DATA. `NCHS` contains records of body shape, health, and mortality of 31126 people.

```
data(NCHS, package = "DataComputing")
```

One of the variables in `NCHS` is `hdl` — HDL cholesterol. (HDL is the "good" kind of cholesterol, as opposed to LDL cholesterol, which is reported in the `chol` variable).

Suppose you want to know a typical HDL level for the people enrolled in NCHS. A typical value will be a fair representative of all the cases. It combines together information found in the individual cases. Since all the cases are being combined, `summarise()` is appropriate.

```
NCHS %>%
  summarise(typical = mean(hdl, na.rm = TRUE))
```

typical
52.37

Or, suppose for some reason you want the typical HDL, the tallest height, and the number of cases.

```
NCHS %>%
  summarise(typical = mean(hdl, na.rm = TRUE),
            tallest = max(height, na.rm = TRUE))
```

typical	tallest
52.37	2.04

`na.rm = TRUE` instructs the reduction function to ignore missing data. Table 7.10 shows what happens without `na.rm=TRUE` — the missing data shapes the result and no useful information is produced.

```
NCHS %>%
  summarise(typical = mean( hdl ),
            tallest = max( height ))
```

typical	tallest
NA	NA

Table 7.10: When there is missing data (`NA`), functions like `mean()`, `max()`, etc. will indicate that the result is `NA`. This is rarely what you want, so use the named argument `na.rm` to instruct the functions to ignore the missing data: `na.rm = TRUE`

Using `group_by()` in conjunction with `summarise()` lets you calculate a summary for each of several groups. The groups can be defined by a single variable or by two or more variables. Table 7.11 shows a division into groups by `sex`:

```
NCHS %>%
  group_by(sex) %>%
  summarise(typical = mean(hdl, na.rm = TRUE),
            tallest = max(height, na.rm = TRUE))
```

sex	typical	tallest
female	55.70	1.87
male	48.91	2.04

Table 7.11: Dividing `NCHS` into groups by sex.

In Table 7.12, the division is into smokers and non-smokers of each sex.

```
NCHS %>%
  group_by( sex, smoker ) %>%
  summarise(typical = mean(hdl, na.rm=TRUE),
            tallest = min(height, na.rm=TRUE))
```

sex	smoker	typical	tallest
female	no	56.06	0.80
female	yes	53.51	1.40
female	NA	53.12	0.93
male	no	49.19	0.79
male	yes	47.80	1.52
... and so on for 6 rows			

Table 7.12: Dividing NCHS into groups by smoker and sex.

EXAMPLE: COUNTING BIRTHS Suppose you want to create a graph like Figure 7.2 to examine the relative number of male and female births over time.

The `BabyNames` data contains this information implicitly. Before the information can be graphed, you need to wrangle the existing data table to one in glyph-ready form.

The particular wrangling needed here is to calculate the sum of `count` for each sex in each year. Here's one way to do this.

```
YearlyBirths <-
  BabyNames %>%
  group_by(year, sex) %>%
  summarise(births = sum(count))
```

`YearlyBirths` is in glyph-ready form. Using `YearlyBirths`, drawing Figure 7.2 is a matter of assigning variables to graphical aesthetics: `year` to the x-axis, `births` to the y-axis, and `sex` to dot color.

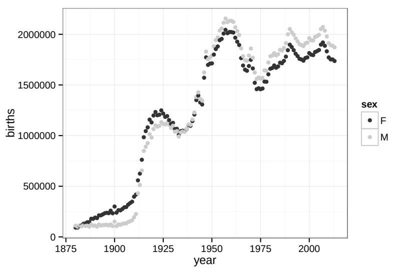


Figure 7.2: Births registered by the US Social Security Administration over the years.

`BabyNames`

name	sex	count	year
Tyshia	F	11	1990
Shacourtney	F	5	1990
Almira	F	7	1990
Elliotte	F	24	2010
Angelmiguel	M	5	2010
... and so on for 1,792,091 rows			

Table 7.13: `BabyNames`

`YearlyBirths`

year	sex	births
1990	F	1897572
1990	M	2052434
2000	F	1814474
2000	M	1962248
2010	F	1771846
... and so on for 268 rows		

Table 7.14: `BabyNames` wrangled into counts of births each year.

7.4 *Exercises*

Problem 7.1
Each of these tasks can be performed using a single data verb. For each task, say which verb it is:

a. Find the average of one of the variables.
b. Add a new column that is the ratio between two variables.
c. Sort the cases in descending order of a variable.
d. Create a new data table that includes only those cases that meet a criterion.
e. From a data table with three categorical variables A, B, & C, and a quantitative variable X, produce an output that has the same cases but only the variables A and X.
f. From a data table with three categorical variables A, B, & C, and a quantitative variable X, produce an output that has a separate case for each of the combinations of the levels of A and B. (Hint: It might be easier to see the answer if the problem statement added, "and gives the maximum value of X over all the cases that have a given combination of A and B.")

Problem 7.2
These questions refer to the `diamonds` data table in the `ggplot2` package. Take a look at the codebook (using `help()`) so that you'll understand the meaning of the tasks. (Motivated by Garrett Grolemund.)

Each of the following tasks can be accomplished by a statement of the form

```
diamonds %>%
  verb1( args1 ) %>%
  verb2( args2 ) %>%
  arrange(desc( args3 )) %>%
  head( 1 )
```

For each task, give appropriate R functions or arguments to substitute in place of `verb1`, `verb2`, `args1`, `args2`, and `args3`.

1. Which color diamonds seem to be largest on average (in terms of carats)?
2. Which clarity of diamonds has the largest average "table" per carat?

Problem 7.3
For each of the operations listed here, say whether it involves a transformation function or a summary function or neither.

a. Determine the 3rd largest.
b. Determine the 3rd and 4th largest values.
c. Determine the number of cases.
d. Determine whether a year is a leap year.
e. Determine whether a date is a legal holiday.
f. Determine the range of a set, that is, the max minus the min.
g. Determine which day of the week (e.g., Sun, Mon, . . .) a given date is.
h. Find the time interval in days spanned by a set of dates.

Problem 7.4
Each of these statements have an error. It might be an error in syntax or an error in the way the data tables are used, etc. Tell what are the error(s) in these expressions.

a)
```
BabyNames %>%
  group_by( "First" ) %>%
  summarise( votesReceived=n() )
```

b)
```
Tmp <- group_by(BabyNames, year, sex ) %>%
  summarise( Tmp, totalBirths=sum(count))
```

c)
```
Tmp <- group_by(BabyNames, year, sex)
summarise( BabyNames, totalBirths=sum(count) )
```

Problem 7.5
Here is a small data table based on `BabyNames`. Take this table as the input.

name	sex	count	year
Christina	M	22	1967
Rotha	F	7	1907
Wayman	M	9	1997
Song	F	11	1994
Julian	M	535	1948
... and so on for 1,792,091 rows			

For each of the following outputs, identify the operation linking the input to the output and write down the details (i.e., arguments) of the operation.

a) Output Table A

name	sex	count	year
Rotha	F	7	1907
Song	F	11	1994
Kalia	F	46	1989
Lissa	F	102	1962
Vicky	F	2945	1957
... and so on for 1,792,091 rows			

b) Output Table B

name	sex	count	year
Rotha	F	7	1907
Song	F	11	1994
Vicky	F	2945	1957
Kalia	F	46	1989
Lissa	F	102	1962
... and so on for 896,046 rows			

c) Output Table C

name	sex	count	year
Christina	M	22	1967
Julian	M	535	1948
... and so on for 416,765 rows			

d) Output Table D

total
333417770

e) Output Table E

name	count
Christina	22
Rotha	7
Wayman	9
Song	11
Julian	535
... and so on for 1,792,091 rows	

Problem 7.6
Using the `Minneapolis2013` data table, answer these questions:

1. How many cases are there?
2. Who were the top 5 candidates in the `Second` vote selections.
3. How many ballots are marked "undervote" in
   - `First` choice selections?
   - `Second` choice selections?
   - `Third` choice selections?
4. What are the top 3 combinations of `First` and `Second` vote selections? (That is, of all the possible ways a voter might have marked his or her first and second choices, which received the highest number of votes?)
5. Which `Precinct` had the highest number of ballots cast?

Problem 7.7
Each of these statements has an error. It might be an error in syntax or an error in the way the data tables are used, etc. Write down a correct version of the statement.

a)
```
BabyNames %>%
  group_by(BabyNames, year, sex) %>%
  summarise(BabyNames, total = sum(count))
```

b)
```
ZipGeography <-
  group_by(State) %>%
  summarise(pop = sum(Population))
```

c)
```
Minneapolis2013 %>%
  group_by(First) ->
  summarise(voteReceived = n())
```

d)
```
summarise(votesReceived = n()) %<%
  group_by(First) <- Minneapolis2013
```

Problem 7.9

a. There's only one data verb that takes a single data table as input and produce an output that (in general) has a different meaning to the case. Which one?

b. There's only one operation that takes two data tables as input rather than just a single data table. Which one?

Problem 7.10

Using the `ZipGeography` data

Find the total land area and population in each state.

- Make a scatter plot showing the relationship between land area and population for each state.
- Make a choropleth map showing the population of each state.
- Make a choropleth map showing the population per unit area of each state.

Problem 7.11

Imagine a data table, `Patients`, with categorical variables `name`, `diagnosis`, `sex`, and quantitative variable `age`.

You have a statement in the form

```
Patients %>%
  group_by(  **some variables** ) %>%
  summarise(count=n(), meanAge = mean(age))
```

Replacing `**some variables**` with each of the following, tell what variables will appear in the output

a. `sex`

b. `diagnosis`

c. `sex`, `diagnosis`

d. `age`, `diagnosis`

e. `age`

Problem 7.12

For each of these computations, say what R function is the most appropriate:

1. Count the number of cases in a data table.
2. List the names of the variables in a data table.
3. For data tables in an R package, display the documentation ("codebook") for the data table.
4. Load a package into your R session.
5. Mark a data table as grouped by one or more categorical variables.
6. Add up, group-by-group, a quantitative variable in a data table.