ST720 Data Science

Tidy Data

Seung Jun Shin (sjshin@krea.ac.kr)

Department of Statistics, Korea University

Tibble?

- Tibbles are a version of dataframes.
- tibble package is a part of "tidverse" package
- Tibble is similar to `data.frame()` but does much less.
  - Never changes the type of inputs.
  - Never creates row names.

Tibble

```
as_tibble(iris)
```

```
## # A tibble: 150 x 5
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##           <dbl>       <dbl>        <dbl>       <dbl> <fct>
##  1          5.1         3.5          1.4         0.2 setosa
##  2          4.9         3            1.4         0.2 setosa
##  3          4.7         3.2          1.3         0.2 setosa
##  4          4.6         3.1          1.5         0.2 setosa
##  5          5           3.6          1.4         0.2 setosa
##  6          5.4         3.9          1.7         0.4 setosa
##  7          4.6         3.4          1.4         0.3 setosa
##  8          5           3.4          1.5         0.2 setosa
##  9          4.4         2.9          1.4         0.2 setosa
## 10          4.9         3.1          1.5         0.1 setosa
## # ... with 140 more rows
```

Tibble

- Tibble is also recycling argments.

```
tibble(
  x = 1:5,
  y = 1,
  z = x^2  + y
)
```

```
## # A tibble: 5 x 3
##       x     y     z
##   <int> <dbl> <dbl>
## 1     1     1     2
## 2     2     1     5
## 3     3     1    10
## 4     4     1    17
## 5     5     1    26
```

Tibble

- Tibble can have column names that are not valid R variable names.

```
tb <- tibble(
  `:)` = "simle",
  ` ` = "space",
  `2000` = "number"
)
tb
```

```
## # A tibble: 1 x 3
##   `:)`  ` `   `2000`
##   <chr> <chr> <chr>
## 1 simle space number
```

Tibble

- `tribble()` means transposed 'tibble'.

```
tribble(
  ~x, ~y, ~z,
 #--/--/----
 "a", 2, 3.6,
 "b", 1, 8.5
)
```

```
## # A tibble: 2 x 3
##   x         y     z
##   <chr> <dbl> <dbl>
## 1 a         2   3.6
## 2 b         1   8.5
```

Tibbles vs. Data.frame

- Tible have a refined print method.

```
tibble(
  a = lubridate::now() + runif(1e3) * 86400,
  b = lubridate::today() + runif(1e3) * 30,
  c = 1:1e3,
  d = runif(1e3),
  e = sample(letters, 1e3, replace = TRUE)
)
```

Tibbles vs. Data.frame

- Tible have a refined print method.

```
## # A tibble: 1,000 x 5
##    a                   b              c      d e
##    <dttm>              <date>     <int>  <dbl> <chr>
##  1 2019-09-19 20:56:15 2019-10-18     1 0.542  e
##  2 2019-09-20 06:33:59 2019-10-10     2 0.0171 b
##  3 2019-09-20 06:55:55 2019-10-07     3 0.594  i
##  4 2019-09-20 13:54:16 2019-09-27     4 0.795  e
##  5 2019-09-20 02:09:01 2019-09-26     5 0.0888 w
##  6 2019-09-19 16:15:19 2019-10-08     6 0.802  y
##  7 2019-09-20 13:27:16 2019-10-05     7 0.779  u
##  8 2019-09-19 19:35:43 2019-10-15     8 0.215  p
##  9 2019-09-20 02:15:43 2019-10-16     9 0.0438 j
## 10 2019-09-19 23:24:37 2019-10-09    10 0.409  s
## # ... with 990 more rows
```

Tibbles vs. Data.frame

```
library(nycflights13)
flights %>%
  print(n = 2, width = Inf)
```

```
## # A tibble: 336,776 x 19
##    year month   day dep_time sched_dep_time dep_delay arr_tim
##   <int> <int> <int>    <int>          <int>     <dbl>    <int
## 1  2013     1     1      517            515         2      83
## 2  2013     1     1      533            529         4      85
##   sched_arr_time arr_delay carrier flight tailnum origin dest
##            <int>     <dbl> <chr>    <int> <chr>   <chr>  <chr
## 1            819        11 UA        1545 N14228  EWR    IAH
## 2            830        20 UA        1714 N24211  LGA    IAH
##   distance  hour minute time_hour
##      <dbl> <dbl>  <dbl> <dttm>
## 1     1400     5     15 2013-01-01 05:00:00
## 2     1416     5     29 2013-01-01 05:00:00
## # ... with 3.368e+05 more rows
```

Interacting with Older Code

```
class(as.data.frame(tb))
```

```
## [1] "data.frame"
```

- makes problem.
- returns various objects, but tibble always returns tibble.

Import Data with `readr` package

`readr` package

- `readr` package is a part of `tidyverse` package.
  - `read_csv()`: comma
  - `read_csv2()`: semi-colon
  - `read_tsv()`: tab
  - `read_delim()`: any delimiter
  - `read_fwf()`: fixed-width file
  - `read_web()` Apche style log file. Check "webread" as well.
- functions are all similar.

Examples

```
read_csv("a,b,c
         1,2,3
         4,5,6")
```

```
## # A tibble: 2 x 3
##       a     b     c
##   <dbl> <dbl> <dbl>
## 1     1     2     3
## 2     4     5     6
```

Examples (skip lines)

```
read_csv("The first lin of metadata
         The second line of metadata
         x,y,z
         1,2,3", skip = 2)
```

```
## # A tibble: 1 x 3
##       x     y     z
##   <dbl> <dbl> <dbl>
## 1     1     2     3
```

```
read_csv("# A comment I want to skip
         x,y,z
         1,2,3", comment = "#")
```

```
## # A tibble: 1 x 3
##       x     y     z
##   <dbl> <dbl> <dbl>
## 1     1     2     3
```

Examples (headers)

```
read_csv("1,2,3\n4,5,6", col_names = FALSE)
```

```
## # A tibble: 2 x 3
##      X1    X2    X3
##   <dbl> <dbl> <dbl>
## 1     1     2     3
## 2     4     5     6
```

```
read_csv("1,2,3\n4,5,6", col_names = c("x", "y", "z"))
```

```
## # A tibble: 2 x 3
##       x     y     z
##   <dbl> <dbl> <dbl>
## 1     1     2     3
## 2     4     5     6
```

Examples (missing)

```
read_csv("a,b,c \n1,2,.", na = ".")
```

```
## # A tibble: 1 x 3
##       a     b c
##   <dbl> <dbl> <lgl>
## 1     1     2 NA
```

Compared to Base R

- much faster!
- always produce tibbles, not convert thier structures.
- more reproducible.

Problems

```
challenge <- read_csv(readr_example("challenge.csv"))
```

```
## Parsed with column specification:
## cols(
##   x = col_double(),
##   y = col_logical()
## )
```

```
## Warning: 1000 parsing failures.
##  row col           expected     actual
## 1001   y 1/0/T/F/TRUE/FALSE 2015-01-16 '/Library/Frameworks/R
## 1002   y 1/0/T/F/TRUE/FALSE 2018-05-18 '/Library/Frameworks/R
## 1003   y 1/0/T/F/TRUE/FALSE 2015-09-05 '/Library/Frameworks/R
## 1004   y 1/0/T/F/TRUE/FALSE 2012-11-28 '/Library/Frameworks/R
## 1005   y 1/0/T/F/TRUE/FALSE 2020-01-13 '/Library/Frameworks/R
## .... ... ................. .......... ....................
## See problems(...) for more details.
```

Problems

```
problems(challenge)
```

```
## # A tibble: 1,000 x 5
##       row col   expected        actual   file
##     <int> <chr> <chr>           <chr>    <chr>
##  1   1001 y     1/0/T/F/TRUE~ 2015-01~ '/Library/Frameworks/R.
##  2   1002 y     1/0/T/F/TRUE~ 2018-05~ '/Library/Frameworks/R.
##  3   1003 y     1/0/T/F/TRUE~ 2015-09~ '/Library/Frameworks/R.
##  4   1004 y     1/0/T/F/TRUE~ 2012-11~ '/Library/Frameworks/R.
##  5   1005 y     1/0/T/F/TRUE~ 2020-01~ '/Library/Frameworks/R.
##  6   1006 y     1/0/T/F/TRUE~ 2016-04~ '/Library/Frameworks/R.
##  7   1007 y     1/0/T/F/TRUE~ 2011-05~ '/Library/Frameworks/R.
##  8   1008 y     1/0/T/F/TRUE~ 2020-07~ '/Library/Frameworks/R.
##  9   1009 y     1/0/T/F/TRUE~ 2011-04~ '/Library/Frameworks/R.
## 10   1010 y     1/0/T/F/TRUE~ 2010-05~ '/Library/Frameworks/R.
## # ... with 990 more rows
```

Problems

```
challenge <- read_csv(readr_example("challenge.csv"),
                      col_types = cols(
                      x = col_double(),
                      y = col_character())
                      )
tail(challenge)
```

```
## # A tibble: 6 x 2
##       x y
##   <dbl> <chr>
## 1 0.805 2019-11-21
## 2 0.164 2018-03-29
## 3 0.472 2014-08-04
## 4 0.718 2015-08-16
## 5 0.270 2020-02-04
## 6 0.608 2019-01-06
```

writing to a file

- `wirte_csv()`, `read_csv()`
- `wirte_tsv()`, `read_tsv()`
- `wirte_rds()`, `read_rds()`

  ...

Tidy Data

Tidy Data

- Tidy Data is a consistent way to organize data in R
- Tidy Data:
  - Each variable must have its own column.
  - Each observation must have its own row.
  - Each value must have its own cell.

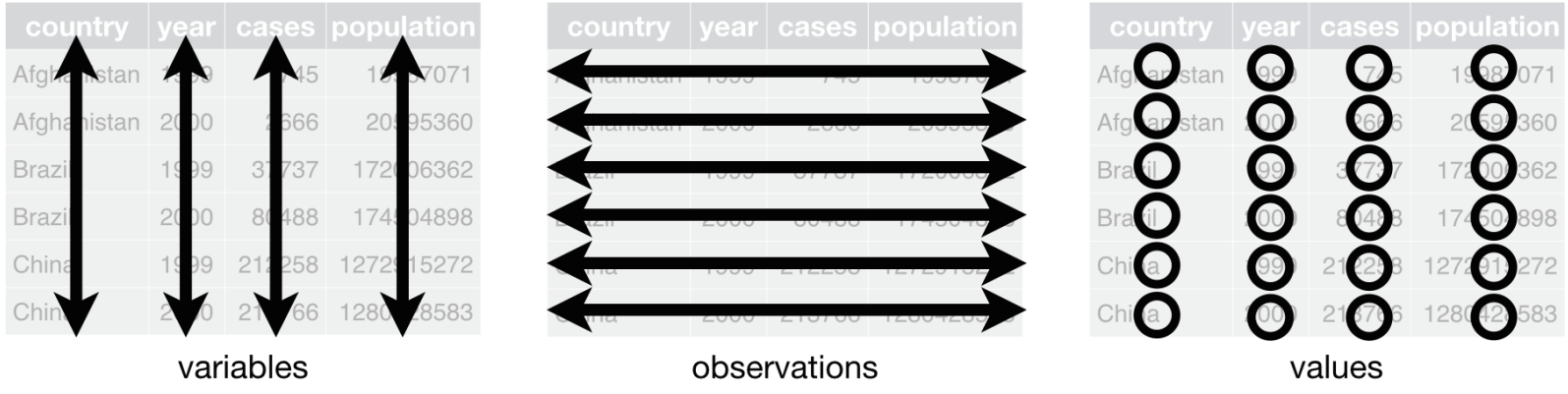


Figure 1: visualization of three rules

Practical Instruction

- Put each dataset in a tibble.
- Put each variable in a column.

Advantageous of Tidy Data

- Consistent way of storing data.
- Placing variable in columns is natural due to R's vectorized nature.
- **dplyr** and **ggplot2** are designed to work with tidy data.

some examples (built in tidyr)

- `table1` is tidy.

```
table1
```

```
## # A tibble: 6 x 4
##   country      year  cases population
##   <chr>       <int>  <int>      <int>
## 1 Afghanistan  1999    745   19987071
## 2 Afghanistan  2000   2666   20595360
## 3 Brazil       1999  37737  172006362
## 4 Brazil       2000  80488  174504898
## 5 China        1999 212258 1272915272
## 6 China        2000 213766 1280428583
```

What to do with untidy data?

- Most data are untidy.
  - people are not familiar with tidy data.
  - data are often organized to facilitate the use other than analysis.
- The first step is always to figure out what the variables and observations are.
- The second step is to resolve one of two common problems:
  1. One variable might be spread across multiple columns.
  2. One observation might be scattered across multiple rows.
- Two most important function in `tidyr` package are `gather()` and `spread()`.

Gathering

- Some of column names are not names of variables, but values of variables

```
table4a
```

```
## # A tibble: 3 x 3
##   country     `1999` `2000`
## * <chr>        <int>  <int>
## 1 Afghanistan    745   2666
## 2 Brazil       37737  80488
## 3 China       212258 213766
```

- You need to gather those columns into a new pair of variables.

Three parameters for gathering

- The set of columns that represents values, not variables.
- `key`: Name of the variable whose values from from the column names.
- `value`: Name of the variable whose values are spread over the cells.

Visualization of gathering

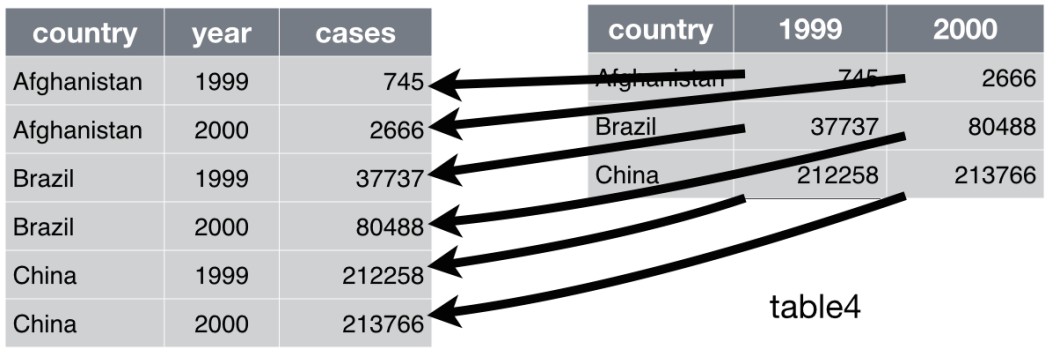


Figure 2: Gathering table4 into a tidy form

tidy table4a via gather()

```
tidy4a <- table4a %>%
  gather(`1999`, `2000`, key = "year", value = "cases")
tidy4a
```

```
## # A tibble: 6 x 3
##       country     year   cases
##         <chr>      <chr>  <int>
## 1 Afghanistan 1999     745
## 2 Brazil      1999   37737
## 3 China       1999  212258
## 4 Afghanistan 2000    2666
## 5 Brazil      2000   80488
## 6 China       2000  213766
```

Tidy table4b via gather()

```
tidy4b <- table4b %>%
  gather(`1999`, `2000`, key = "year", value = "population")
tidy4b
```

```
## # A tibble: 6 x 3
##       country     year  population
##         <chr>      <chr>       <int>
## 1 Afghanistan 1999    19987071
## 2 Brazil      1999   172006362
## 3 China       1999  1272915272
## 4 Afghanistan 2000    20595360
## 5 Brazil      2000   174504898
## 6 China       2000  1280428583
```

Combine via dplyr::left_join()

```
left_join(tidy4a, tidy4b)
```

```
## Joining, by = c("country", "year")
```

```
## # A tibble: 6 x 4
##   country     year   cases population
##   <chr>       <chr>  <int>      <int>
## 1 Afghanistan 1999     745   19987071
## 2 Brazil      1999   37737  172006362
## 3 China       1999  212258 1272915272
## 4 Afghanistan 2000    2666   20595360
## 5 Brazil      2000   80488  174504898
## 6 China       2000  213766 1280428583
```

Spreading

- Spreadning is the opposite of gathering.
- Use when an observation is scattered across multiple rows.

```
table2
```

```
## # A tibble: 12 x 4
##    country      year type            count
##    <chr>       <int> <chr>           <int>
##  1 Afghanistan  1999 cases             745
##  2 Afghanistan  1999 population   19987071
##  3 Afghanistan  2000 cases            2666
##  4 Afghanistan  2000 population   20595360
##  5 Brazil       1999 cases           37737
##  6 Brazil       1999 population  172006362
##  7 Brazil       2000 cases           80488
##  8 Brazil       2000 population  174504898
##  9 China        1999 cases          212258
## 10 China        1999 population 1272915272
## 11 China        2000 cases          213766
## 12 China        2000 population 1280428583
```

Spreading

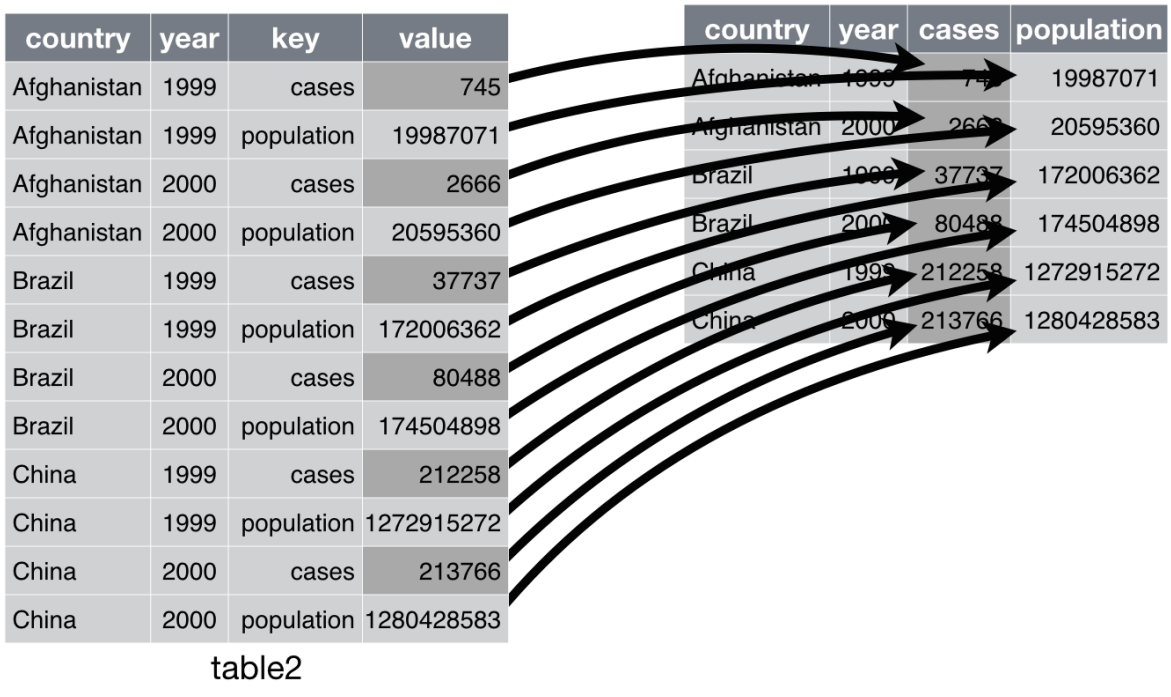
- We need to parameters
  - `key` column: contains variable names.
  - `value` column: contains values from multiple variables.

Spreading

```
spread(table2, key = type, value = count)
```

```
## # A tibble: 6 x 4
##   country      year  cases population
##   <chr>       <int>  <int>      <int>
## 1 Afghanistan  1999    745   19987071
## 2 Afghanistan  2000   2666   20595360
## 3 Brazil       1999  37737  172006362
## 4 Brazil       2000  80488  174504898
## 5 China        1999 212258 1272915272
## 6 China        2000 213766 1280428583
```

Visualization of Spreading



country	year	key	value
Afghanistan	1999	cases	745
Afghanistan	1999	population	19987071
Afghanistan	2000	cases	2666
Afghanistan	2000	population	20595360
Brazil	1999	cases	37737
Brazil	1999	population	172006362
Brazil	2000	cases	80488
Brazil	2000	population	174504898
China	1999	cases	212258
China	1999	population	1272915272
China	2000	cases	213766
China	2000	population	1280428583

table2

country	year	cases	population
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583

Figure 3: Spreading table2 makes it tidy

Separate

- What about the following example?

```
table3
```

```
## # A tibble: 6 x 3
##   country      year rate
## * <chr>       <int> <chr>
## 1 Afghanistan  1999 745/19987071
## 2 Afghanistan  2000 2666/20595360
## 3 Brazil       1999 37737/172006362
## 4 Brazil       2000 80488/174504898
## 5 China        1999 212258/1272915272
## 6 China        2000 213766/1280428583
```

- rate column contains both cases and population

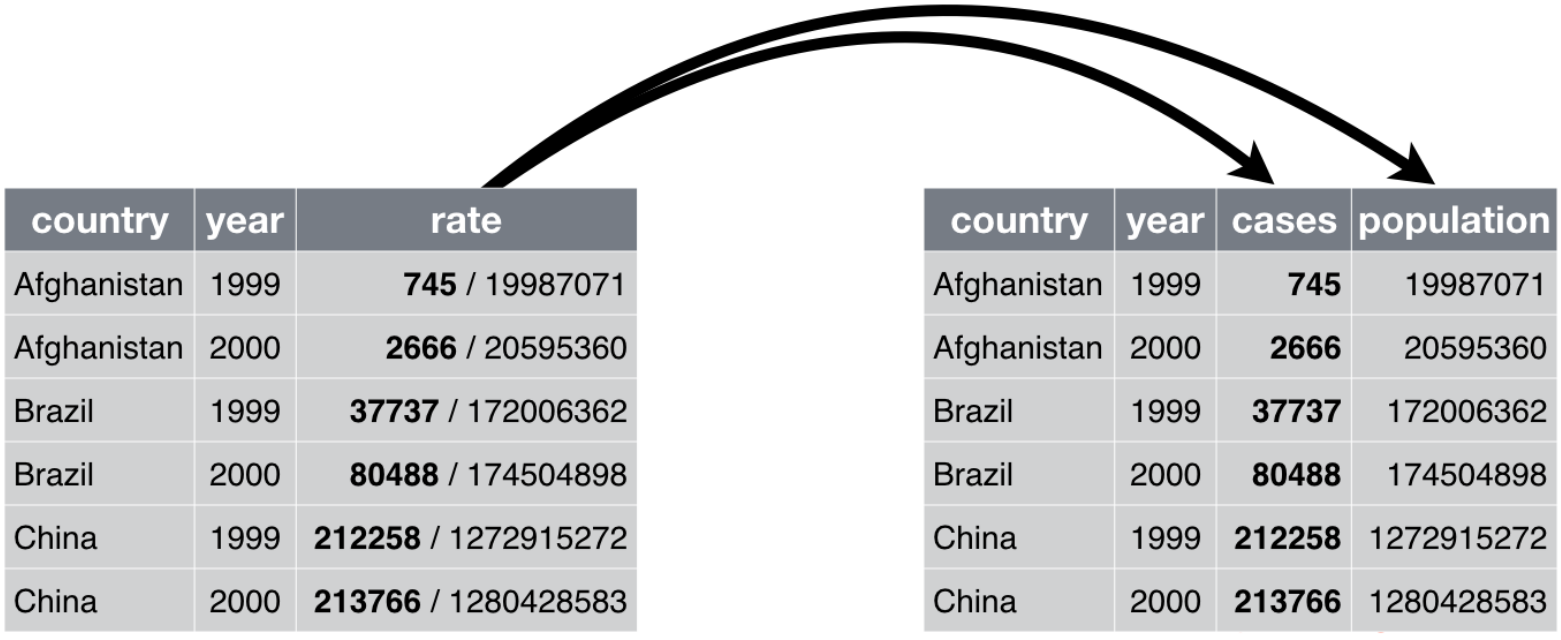
separate()

- By default it split values wherever it sees a non-alphanumeric character.

```
table3 %>%
  separate(rate, into = c("cases", "population"))
```

```
## # A tibble: 6 x 4
##   country      year cases  population
##   <chr>       <int> <chr>  <chr>
## 1 Afghanistan  1999 745    19987071
## 2 Afghanistan  2000 2666   20595360
## 3 Brazil       1999 37737  172006362
## 4 Brazil       2000 80488  174504898
## 5 China        1999 212258 1272915272
## 6 China        2000 213766 1280428583
```

Visualization of `separate()`



country	year	rate
Afghanistan	1999	**745** / 19987071
Afghanistan	2000	**2666** / 20595360
Brazil	1999	**37737** / 172006362
Brazil	2000	**80488** / 174504898
China	1999	**212258** / 1272915272
China	2000	**213766** / 1280428583

table3

country	year	cases	population
Afghanistan	1999	**745**	19987071
Afghanistan	2000	**2666**	20595360
Brazil	1999	**37737**	172006362
Brazil	2000	**80488**	174504898
China	1999	**212258**	1272915272
China	2000	**213766**	1280428583

Figure 4: Separating table3 makes it tidy

convert argument

```
table3 %>%
  separate(rate, into = c("cases", "population"), convert = TRUE
```

```
## # A tibble: 6 x 4
##   country      year  cases population
##   <chr>       <int>  <int>      <int>
## 1 Afghanistan  1999    745   19987071
## 2 Afghanistan  2000   2666   20595360
## 3 Brazil       1999  37737  172006362
## 4 Brazil       2000  80488  174504898
## 5 China        1999 212258 1272915272
## 6 China        2000 213766 1280428583
```

Unite the columns

- `unite()` is the inverse of `separate()`

country	year	rate
Afghanistan	19**99**	745 / 19987071
Afghanistan	20**00**	2666 / 20595360
Brazil	19**99**	37737 / 172006362
Brazil	20**00**	80488 / 174504898
China	19**99**	212258 / 1272915272
China	20**00**	213766 / 1280428583

country	century	year	rate
Afghanistan	19	**99**	745 / 19987071
Afghanistan	20	**0**	2666 / 20595360
Brazil	19	**99**	37737 / 172006362
Brazil	20	**0**	80488 / 174504898
China	19	**99**	212258 / 1272915272
China	20	**0**	213766 / 1280428583

table6

Figure 5: Separating table3 makes it tidy

table5

```
table5
```

```
## # A tibble: 6 x 4
##   country     century year  rate
## * <chr>       <chr>   <chr> <chr>
## 1 Afghanistan 19      99    745/19987071
## 2 Afghanistan 20      00    2666/20595360
## 3 Brazil      19      99    37737/172006362
## 4 Brazil      20      00    80488/174504898
## 5 China       19      99    212258/1272915272
## 6 China       20      00    213766/1280428583
```

unite()

```
table5 %>%
  unite(new, century, year)
```

```
## # A tibble: 6 x 3
##   country     new   rate
##   <chr>       <chr> <chr>
## 1 Afghanistan 19_99 745/19987071
## 2 Afghanistan 20_00 2666/20595360
## 3 Brazil      19_99 37737/172006362
## 4 Brazil      20_00 80488/174504898
## 5 China       19_99 212258/1272915272
## 6 China       20_00 213766/1280428583
```

- By default underscore(_) is placed between the values from different columns.

unite()

```
table5 %>%
  unite(new, century, year, sep="")
```

```
## # A tibble: 6 x 3
##   country     new   rate
##   <chr>       <chr> <chr>
## 1 Afghanistan 1999  745/19987071
## 2 Afghanistan 2000  2666/20595360
## 3 Brazil      1999  37737/172006362
## 4 Brazil      2000  80488/174504898
## 5 China       1999  212258/1272915272
## 6 China       2000  213766/1280428583
```

Handling Missing Values

- A value can be missing in one of two possible ways:
  - **Explicitly**, i.e. flagged with NA.
  - **Implicitly**, i.e. simply not present in the data.

stock Data

```
## # A tibble: 7 x 3
##    year   qtr return
##   <dbl> <dbl>  <dbl>
## 1  2015     1   1.88
## 2  2015     2   0.59
## 3  2015     3   0.35
## 4  2015     4  NA
## 5  2016     2   0.92
## 6  2016     3   0.17
## 7  2016     4   2.66
```

- 2015 4th quarter : explicitly missing
- 2016 1st quarter : implicitly missing

complete() : Make implicit explicit

```
stocks %>%
  complete(year, qtr)
```

```
## # A tibble: 8 x 3
##    year   qtr return
##   <dbl> <dbl>  <dbl>
## 1  2015     1   1.88
## 2  2015     2   0.59
## 3  2015     3   0.35
## 4  2015     4  NA
## 5  2016     1  NA
## 6  2016     2   0.92
## 7  2016     3   0.17
## 8  2016     4   2.66
```

- takes a set of columns, and finds all unique combinations
- ensures the original dataset contains all those values
- fill in explicit NAs where necessary.

`fill()`: Fill missing values based on the previous one

- when a data source has primarily been used for data entry, missing values indicate that the previous value should be carried forward

```
## # A tibble: 4 x 3
##   person           treatment response
##   <chr>                <dbl>    <dbl>
## 1 Derrick Whitmore         1        7
## 2 <NA>                     2       10
## 3 <NA>                     3        9
## 4 Katherine Burke          1        4
```

fill()

```
treatment %>%
  fill(person)
```

```
## # A tibble: 4 x 3
##   person           treatment response
##   <chr>                <dbl>    <dbl>
## 1 Derrick Whitmore         1        7
## 2 Derrick Whitmore         2       10
## 3 Derrick Whitmore         3        9
## 4 Katherine Burke          1        4
```

Reference

- Wickham, H. and Grolemund, G. (2017) R for Data Science, O'reilly Media Inc., Chapter 7–9.