Data Analytics
CS301
Tidy Data and Import

Week 6
Fall 2018
Oliver Bonham-Carter

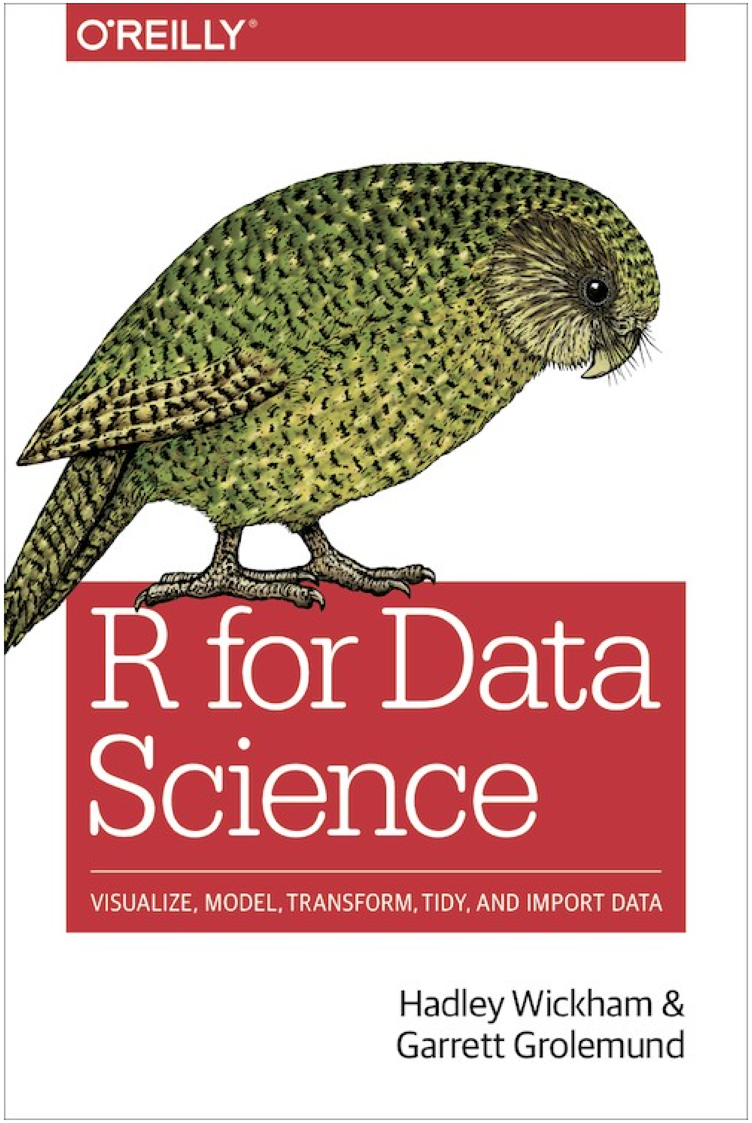
On Exam 1, 2:30pm, 12th Oct 2018

- Online exam: multiple choice, matching, True and False.
- Up through week 6: *Exploratory Data Analysis* Material, and *Tidy Data and Import*
- **Study your slides and notes.**
- **Turn to the book to provide further detail as needed.**
- Be familiar with the graphing code in R. You do not have to write code, but you will have to recognize working code.
- Be familiar with concepts discussed in class such as problems using bin-widths, working with outliers or missing information, categorical variables and others discussion points.

On Exam 1, 2:30pm, 12th Oct 2018

- Topics
  - Google Analytics
    - Web traffic Information: understanding terms and reading plots
  - Data Visualizations: types and meanings
  - R Statistics: recognizing the code to make particular plots
    - Basic syntax and methods
  - Library features:
    - Tidyverse, nycflights13, lubridate
  - Concepts:
    - Exploratory data analysis
    - Tidy data manipulation
    - Managing date and time
    - Others from recent lessons

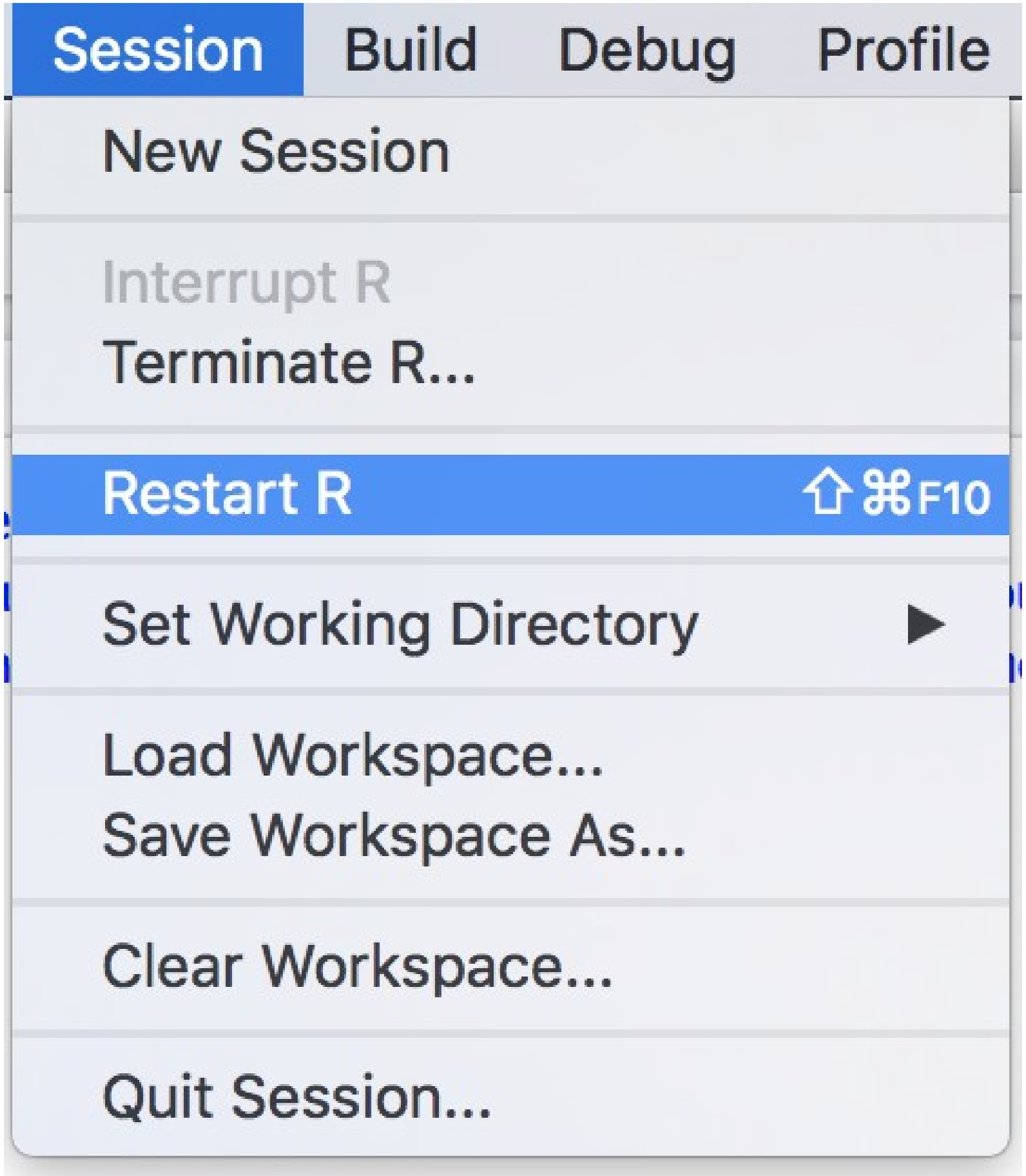
Where in the Web?
Where in the Book?



- Note the chapter differences!
- Book:
  - Chap 8
- Web:
  - Chap 11
- Tidy Data and Import

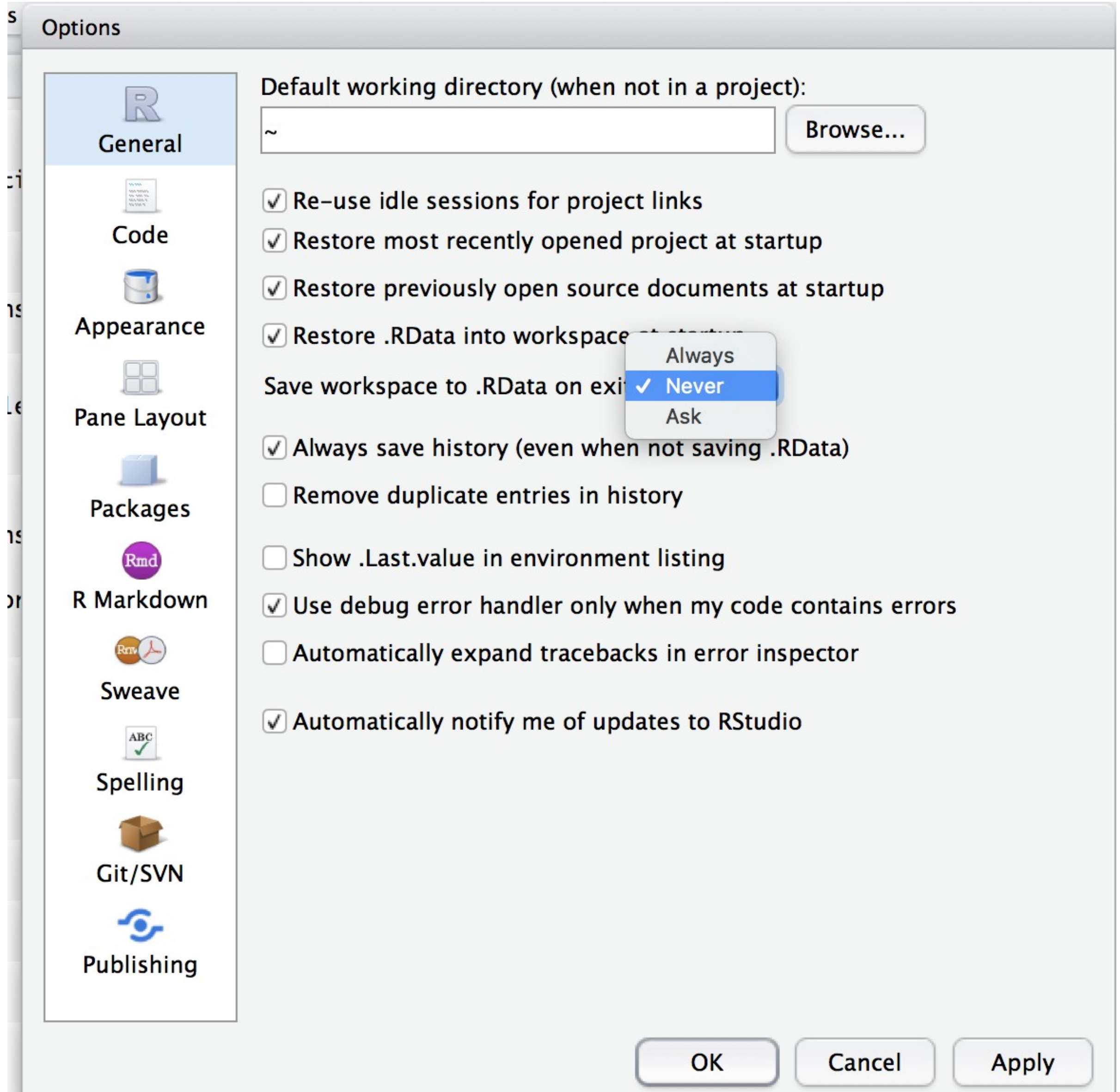
Now That We Are RStudio Programmers...

- Consider starting with a clean-slate, without a bunch of old data tables.
- Consider not saving your R-environment after each session.
- Instead, your work and code should come source files and not be text-mined from the command history.



Now That We Are RStudio Programmers...

- Consider stopping the workspace from being saved each time.
- This move will encourage you to begin writing code to be opened in RStudio.
- Better command archive for future works.



Entering Data as a Table

Your own data typed in:

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

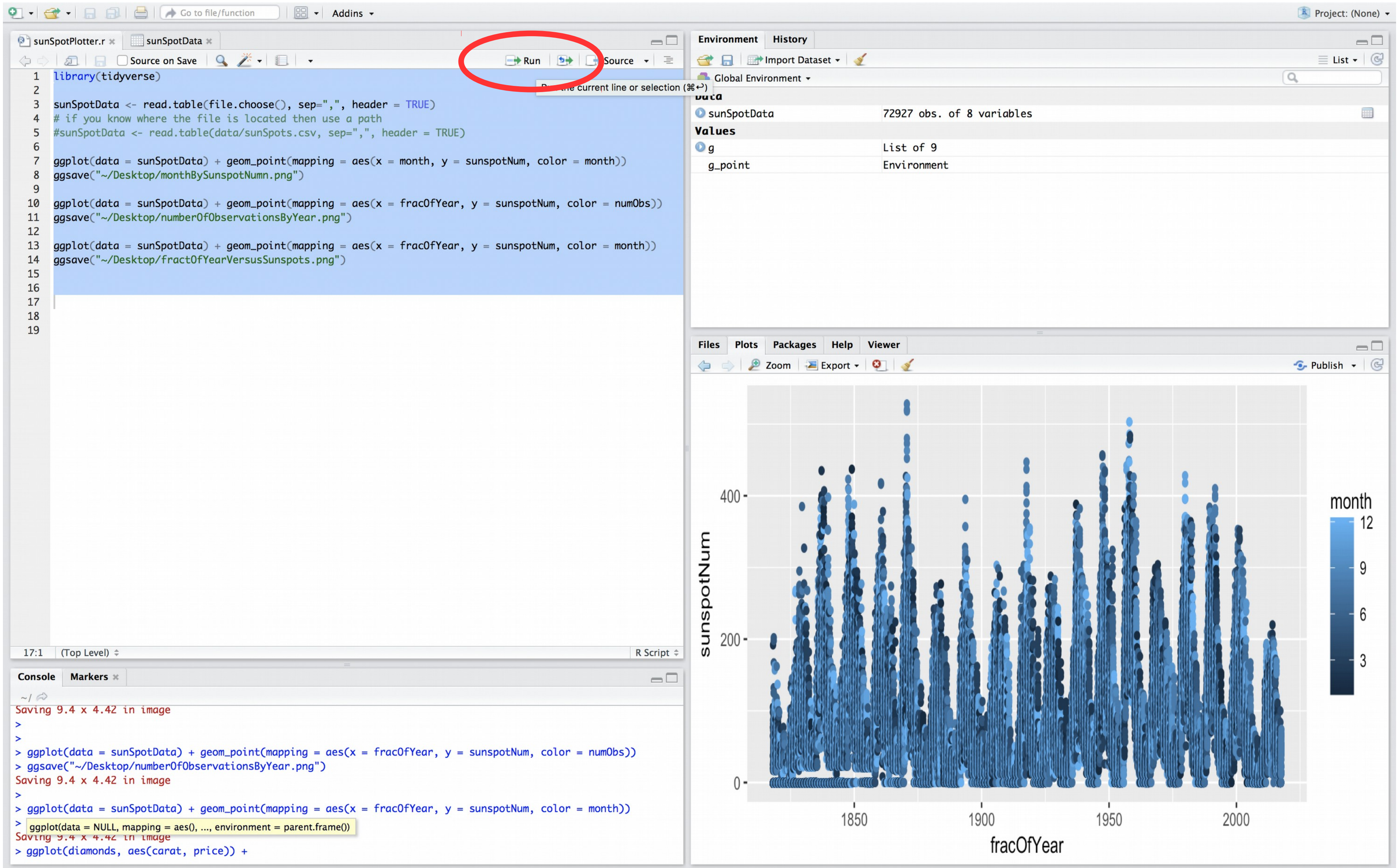
```
> read_csv("a,b,c
+ 1,2,3
+ 4,5,6")
# A tibble: 2 x 3
      a     b     c
  <int> <int> <int>
1     1     2     3
2     4     5     6
```

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

Loading Data and Saving Plots

```
#Rscript:

library(tidyverse)

sunSpotData1 <- read.table(file.choose(),
sep=",",header = TRUE)

#sunSpotData2 <- read.table(data/sunSpots.csv,
sep=",",header = TRUE)

sunSpotData3 <- read_csv("PATH/sunSpots.csv")

ggplot(data = sunSpotData1) + geom_point(mapping =
aes(x = fracOfYear, y = sunspotNum, color = month))

#save the plot to file

ggsave("~/Desktop/fractOfYearVersusSunspots.png")
```

Save only good code and then have it to run later.



How Do We Deal With Messy Data?

- We may try to use a data table only to find:
  - There are numbers mixed with characters
  - Different types of entries are mixed in a column
  - Mixed makes things messy.

The Organization of Data

#Naturally tidy data:

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

What are the qualities that make data tidy?!

```
library(tidyverse)
```

The same data displayed in multiple ways; each data set below organizes the values in a different way

```
table1 # country  year  cases population
table2 # country  year  type   count
table3 # country  year rate
table4a # country `1999` `2000`
table4b # country `1999` `2000`
```

Tidy Data

- What does tidy data look like?
  - A column should be of all same types and description
- There are three interrelated rules which make a data set *tidy*:
  - Each variable must have its own column.
  - Each observation must have its own row.
  - Each value must have its own cell.

Tidy Data

- Be tidy: it matters how your data is arranged
- Trends could be *missed due to mess*
- Code is easiest to implement when data from a column is same

Figure 9-1 shows the rules visually.

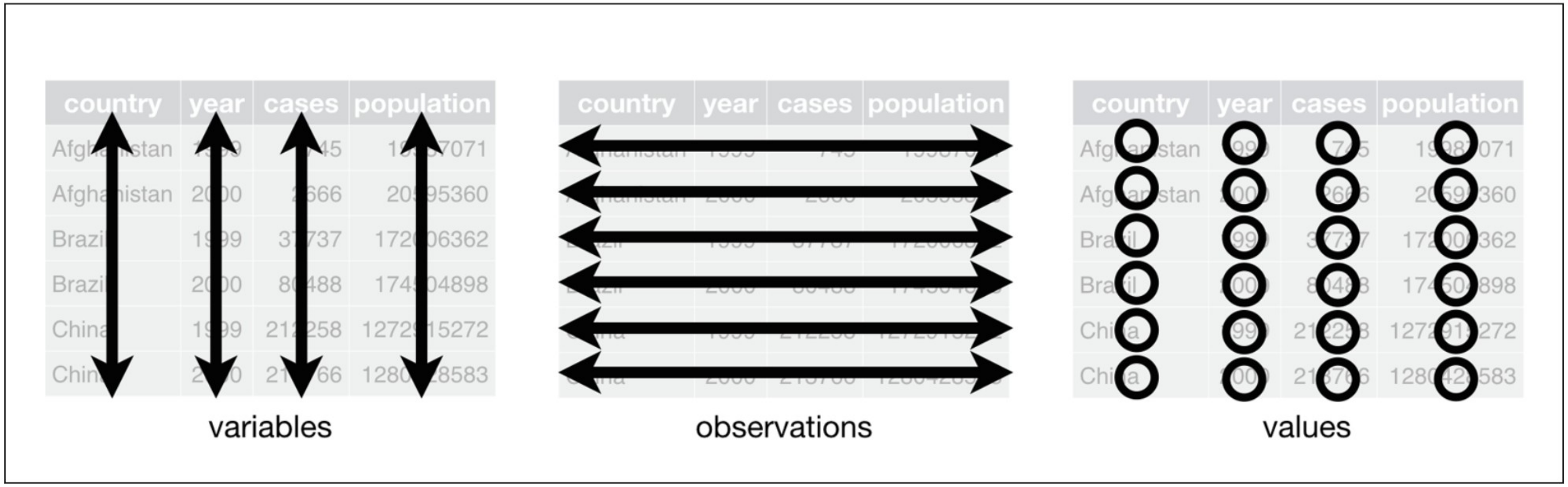


Figure 9-1. The following three rules make a dataset tidy: variables are in columns, observations are in rows, and values are in cells

Which Table is Most Tidy?

View(table1)

- There are three interrelated rules which make a data set tidy:
  - Each variable must have its own column.
  - Each observation must have its own row.
  - Each value must have its own cell.
- Table 1 is the most tidy for for data-organization

```
> 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
```

All same types and descriptions in columns, but it seems that two sets are mixed

Not Tidy!!

- View(table2)
- Not tidy
- The Cases are easily confused

```
> 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
```

Use Tibble to find out how many different *years* there are mixed together

- Use the functions that we already know
  - #**Mutate()** to Compute rate (new column) per 10,000

```
table1 %>%
mutate(rate = cases / population * 10000)
```

  - #**Quick Computations** of cases per year

```
table1 %>% count(year, wt = cases)
```

<int> <int>
1 1999 250740
2 2000 296920

1999: 745 + 37737 + 212258
2000: 213766 + 80488 + 2666

```
table1 %>% count(country, wt =
      as.numeric(population))
```

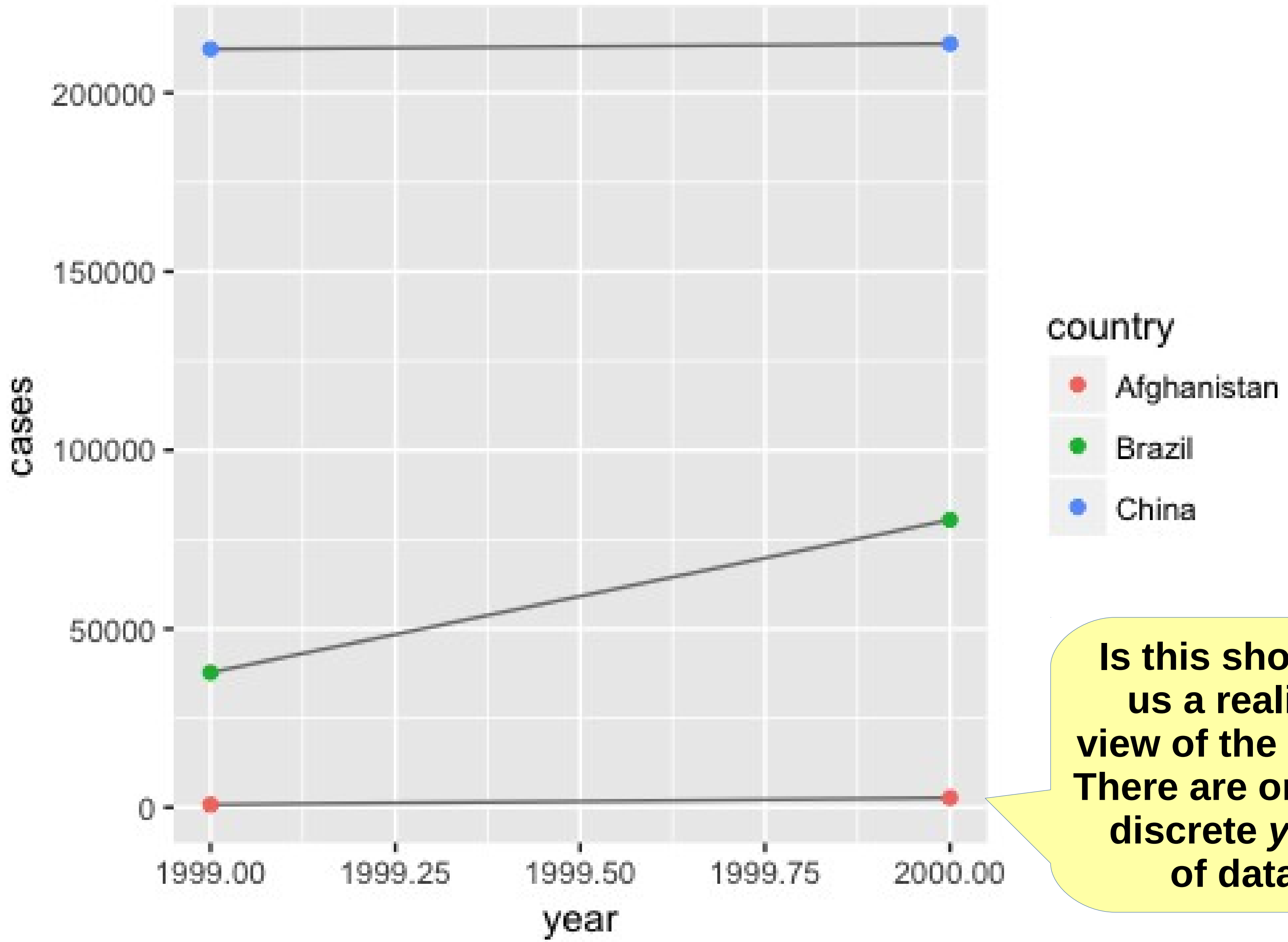
count the populations, aggregated by country

Implement Some Code

Visualize changes over time on *table1*

```
library(ggplot2)

ggplot(table1, aes(year, cases)) +
geom_line(aes(group = country),colour =
"grey50") + geom_point(aes(colour = country))
```

Output From The Code



Is this showing us a realistic view of the years? There are only two discrete *years* of data.

Bad Organization, Bad Luck!!

- **We can apply code to data when in the right format (integers, strings, etc.)**
- **What happens when the data is badly stored; messy, and without any organization??!**

Gather(): Table4a

- *The Gather()* function takes multiple columns and collapses into key-value pairs, duplicating all other columns as needed.
- Use *gather()* when you notice that you have columns that are not variables.

These variables could be better ordered as elements of “Year”

```
> table4a
# A tibble: 3 x 3
      country `1999` `2000`
*       <chr>  <int>  <int>
1 Afghanistan    745   2666
2      Brazil  37737  80488
3       China 212258 213766
```

Reordering of Data: table4a

	country	1999	2000
1	Afghanistan	745	2666
2	Brazil	37737	80488
3	China	212258	213766

```
> table4a %>% gather(`1999`, `2000`, key = `year`, value = `cases`)
# 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
```

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

How did we do that?

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

Here's how:
Reorganize the data
in the columns

Figure 12.2: Gathering `table4` into a tidy form.

Reordering of Data: table4b

	country	1999	2000
1	Afghanistan	19987071	20595360
2	Brazil	172006362	174504898
3	China	1272915272	1280428583

```
> table4b %>% gather(`1999`, `2000`,
key = `year`, value = `population`)
# 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
```

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

spread(): table2

- Dealing with mixed values in the same column

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

Here's how:
Reorganize the data
Into two columns

spread(): table2

	country	year	type	count
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

ng 1 to 8 of 12 entries

```
> spread(table2, key = type,
value = count)
# A tibble: 6 x 4
      country  year  cases
*       <chr> <int>  <int>
1 Afghanistan  1999    745
2 Afghanistan  2000   2666
3      Brazil  1999  37737
4      Brazil  2000  80488
5       China  1999 212258
6       China  2000 213766
# ... with 1 more variables:
#   population <int>
```

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

separate(): table3

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

Here's how:
Push the data
into two columns

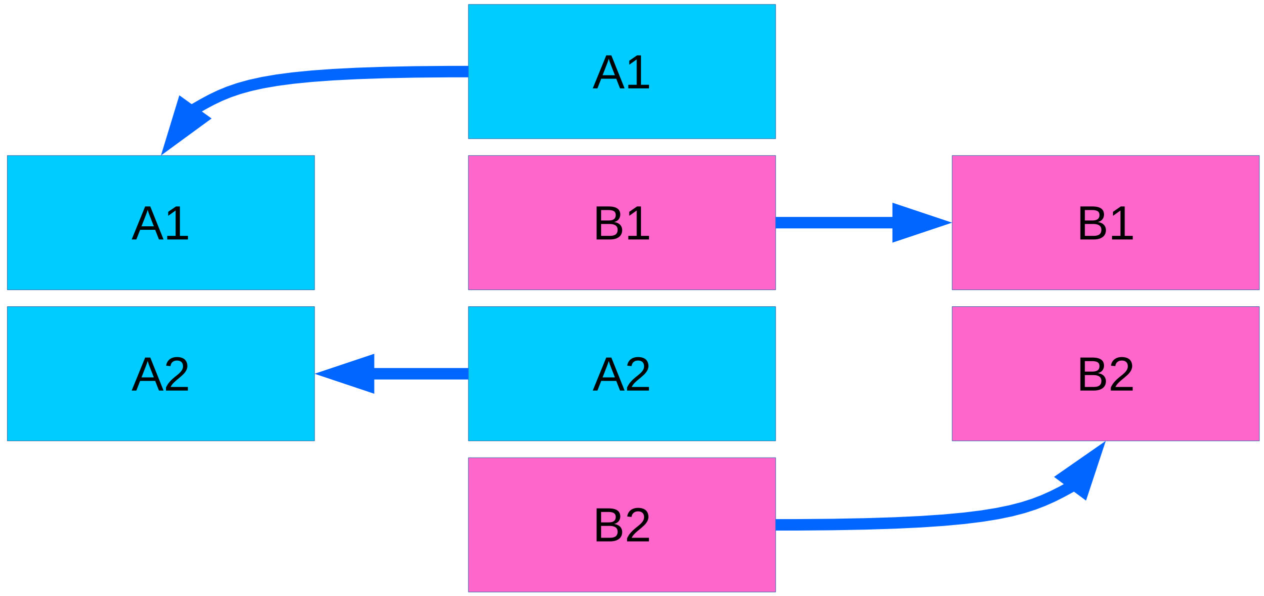
country	year	rate
Afghanistan	1999	**745** / 19987071
Afghanistan	2000	**2666** / 20595360
Brazil	1999	**37737** / 172006362
Brazil	2000	**80488** / 174504898
China	1999	**212258** / 1272915272
China	2000	**213766** / 1280428583

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

separate(): table3

- What do I do if I know that my column contains mixed data entries?
- Given a regular expression of a vector is character positions, *separate()* turns a single character column into multiple columns.



Ex: Separating Compounded Entries: table3

	country	year	rate
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

Break the string into length 2 chunks and place left in new col *century* and other in col *year*.

```
table3  %>%
separate(year, into = c("century", "year"), sep = 2)

table3 %>%
separate(rate, into = c("cases", "pop"), sep = "/")
```

unite(): table6

```
table5 %>%

unite(new,
century,year)
```

Here's how:
Pull the data
from two columns

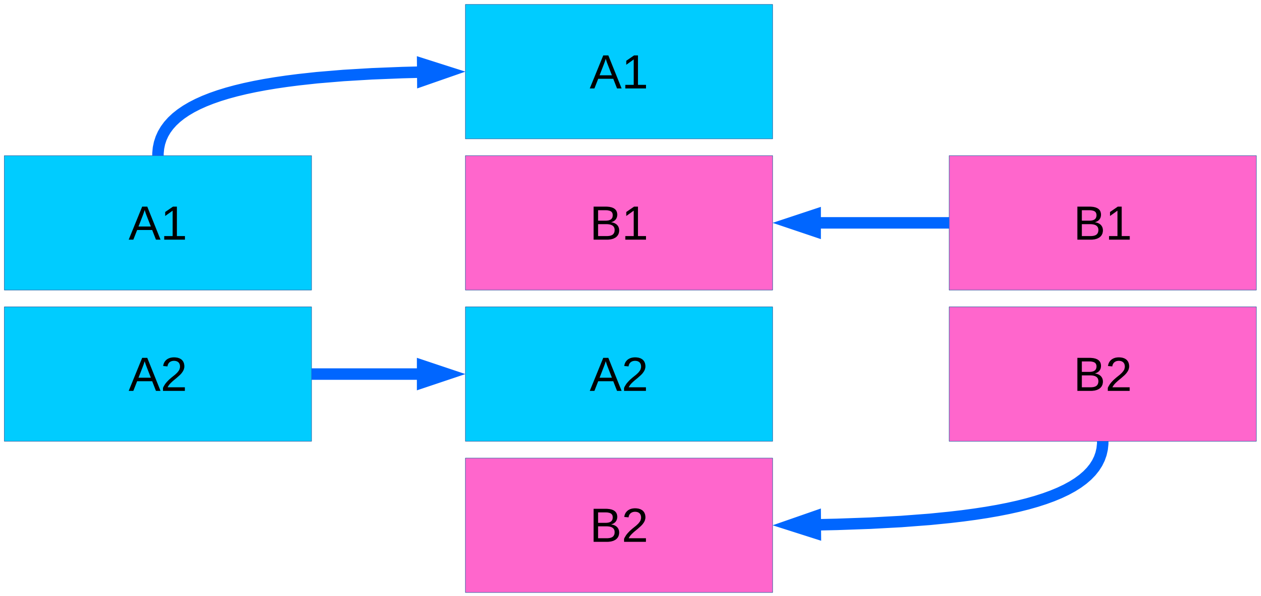
country	year	rate
Afghanistan	1999	745 / 19987071
Afghanistan	2000	2666 / 20595360
Brazil	1999	37737 / 172006362
Brazil	2000	80488 / 174504898
China	1999	212258 / 1272915272
China	2000	213766 / 1280428583

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

unite(): table3

- What do I do if I know that two columns contains data that could go into one column?
- Given a regular expression of a vector is character positions, separate() turns a single character column into multiple columns.



Ex: Unite Compounded Entries

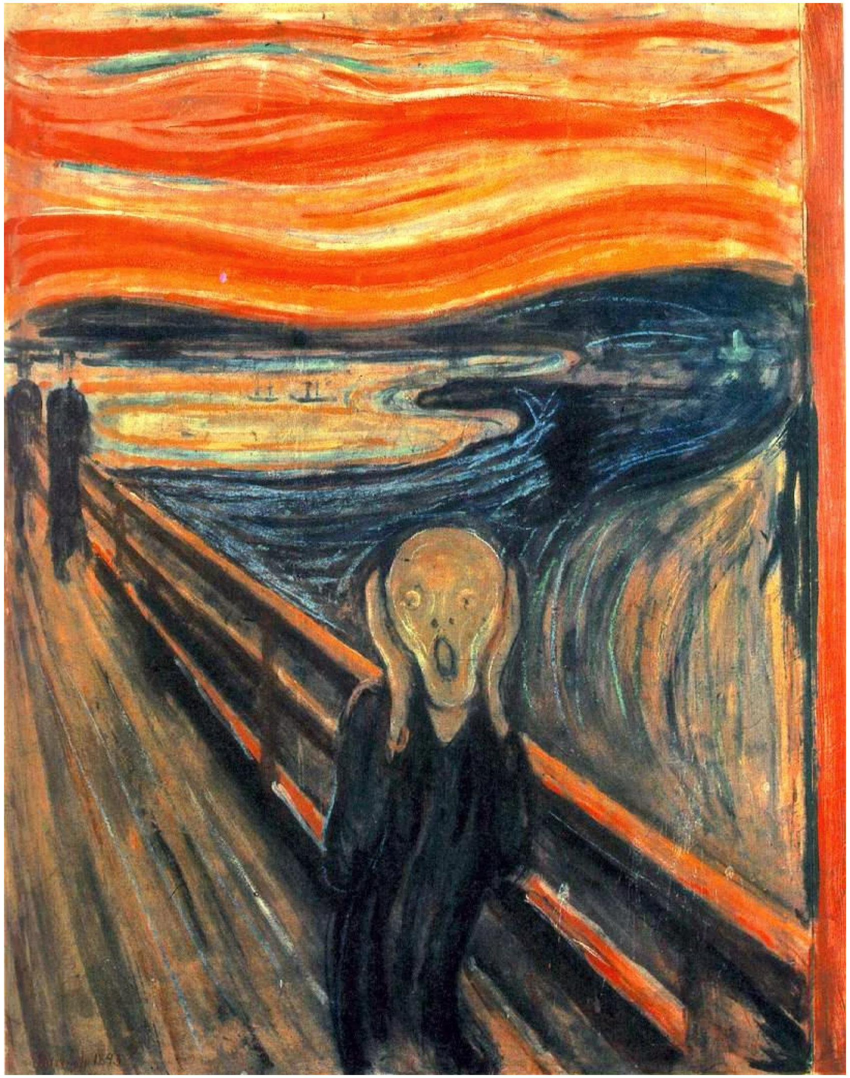
	country	century	year	rate
1	Afghanistan	19	99	745/19987071
2	Afghanistan	20	00	2666/20595360
3	Brazil	19	99	37737/172006362
4	Brazil	20	00	80488/174504898
5	China	19	99	212258/1272915272
6	China	20	00	213766/1280428583

What is the output of this?!

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

Missing Values!?

- We may find that table entries are missing
- Two types of missing entries
  - **Explicitly**, i.e., flagged with *NA*.
  - **Implicitly**, i.e., simply not present in the data.



Missing Data Illustrated

```
# Make a table with a missing entry (NA).
stocks <- tibble(
  year   = c(2015, 2015, 2015, 2015, 2016, 2016, 2016),
  qtr    = c(   1,    2,    3,    4,    2,    3,    4),
  return = c(1.88, 0.59, 0.35,   NA, 0.92, 0.17, 2.66))
```

Missing qtr: "1"

- Two missing values in this dataset:
  - The return for the **fourth** quarter of 2015 is explicitly missing, there is an entry of NA
  - The return for the first quarter of 2016 is implicitly missing, because it simply does not appear in the dataset.

Missing Data In Table

	year	qtr	return
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

Missing “1”

Missing

```
# Make a table with a missing entry (NA).
stocks <- tibble(
  year   = c(2015, 2015, 2015, 2015, 2016, 2016, 2016),
  qtr    = c(   1,    2,    3,    4,    2,    3,    4),
  return = c(1.88, 0.59, 0.35,   NA, 0.92, 0.17, 2.66))
```

Spread the Missing Data

	qtr	2015	2016
1	1	1.88	NA
2	2	0.59	0.92
3	3	0.35	0.17
4	4	NA	2.66

```
# Make the implicit missing values explicit.
# Use spread() to place both years into own column.
stocks %>%
  spread(year, return)
```

Removing Missing Entries

```
# Remove the rows having holes in the data

# Create two cols for years 2015 and 2016

# Place years back into the same col again,
removing the missing entries.

stocks %>%

spread(year, return) %>%  gather(year, return,
`2015`:`2016`, na.rm = TRUE)
```

```
> stocks %>%
+     spread(year, return) %>%  gather(year,
return, `2015`:`2016`, na.rm = TRUE)
# A tibble: 6 x 3
    qtr  year return
* <dbl> <chr>  <dbl>
1     1  2015   1.88
2     2  2015   0.59
3     3  2015   0.35
4     2  2016   0.92
5     3  2016   0.17
6     4  2016   2.66
```

The progression of the tables as the missing values are removed

	year	qtr	return
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

1

	qtr	2015	2016
1	1	1.88	NA
2	2	0.59	0.92
3	3	0.35	0.17
4	4	NA	2.66

2

	qtr	year	return
1	1	2015	1.88
2	2	2015	0.59
3	3	2015	0.35
6	2	2016	0.92
7	3	2016	0.17
8	4	2016	2.66

3

Let's Just Guess About The Missing Stuff...

```
#Create a table with missing entries
treatment <- tribble(
  ~ person, ~ treatment, ~response,
  "Derrick Whitmore", 1, 7,
  NA, 2, 10,
  NA, 3, 9,
  "Katherine Burke",  1,  4
)
```

Treatments Table With Missing Entries

- We assume that Derrick Whitmore's name makes up the missing entries.

	person	treatment	response
1	Derrick Whitmore	1	7
2	NA	2	10
3	NA	3	9
4	Katherine Burke	1	4

Whitmore To The Rescue?

	person	treatment	response
1	Derrick Whitmore	1	7
2	Derrick Whitmore	2	10
3	Derrick Whitmore	3	9
4	Katherine Burke	1	4

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

Can anything go wrong with this?!