# L06. Tidy data

Sim, Min Kyu, Ph.D., mksim@seoultech.ac.kr

SEOULTECH 서울과학기술대학교 데이터사이언스학과

1. I. Tidy data
2. II. Long $\longleftrightarrow$ Wide (`tidyr`)
3. III. Split vs Concatenate (`tidyr` or `stringr`)
4. IV. Joining (`dplyr`)
5. V. `NA`의 처리

Section 1

# I. Tidy data

```
library(dplyr)
library(tidyr)
```

## Tidy data.frame!

1. 개체 타입은 `data.frame` (or, `tibble`)
2. 각각의 row는 관찰값을 의미
3. 각각의 column은 변수를 의미

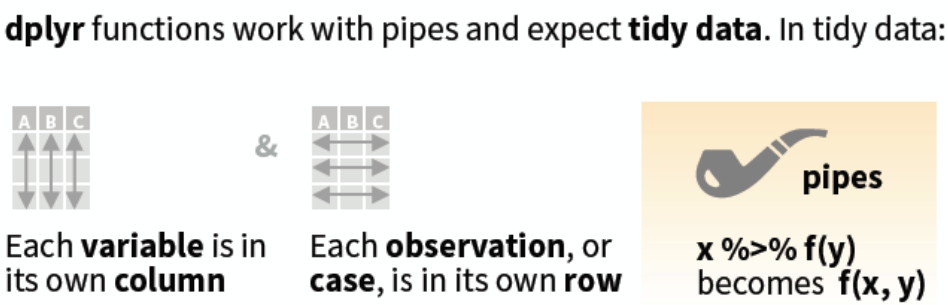



Figure 1: from `dplyr` Cheatsheet

## Advantages

1. 일관된 방식으로 저장되며, 이해하기 쉽다.
2. Vectorized programming의 속성이 발휘된다.
3. `ggplot2`를 비롯한 대부분의 `tidyverse` 패키지는 tidy 데이터에 작동하도록 설계되어있다.

# Tidy vs Non−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
```

```
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
```

```
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
```

```
table4a
```

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

```
table4b
```

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

Section 2

# II. Long ⟷ Wide (tidyr)

# Long to Wide (spread)

- Before (too long)

```
table2 %>% head()
```

```
## # A tibble: 6 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
```

- After (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
```

## spread: Long to Wide

```
table2 %>% spread(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
```

# Wide to Long (gather)

```
table4a
```

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

```
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
```

## gather: Wide to Long

```
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
```

```
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
```

```
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
```

- And the last piece of puzzle? How to put them **tidy4a** and **tidy4b** together?

Section 3

# III. Split vs Concatenate (tidyr or stringr)

# Splitting

- Before

```
table3 %>% head(3)
```

```
## # A tibble: 3 x 3
##   country      year rate
##   <chr>       <int> <chr>
## 1 Afghanistan  1999 745/19987071
## 2 Afghanistan  2000 2666/20595360
## 3 Brazil       1999 37737/172006362
```

- After

```
table1 %>% head(3)
```

```
## # A tibble: 3 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
```

## Using `tidyr::separate()`

```
table3 %>% separate(rate, into = c("cases", "population"), sep = "/") %>% head(3)
```

```
## # A tibble: 3 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
```

# Using stringr::str_split()

```
library(stringr)
temp <-
  str_split(string = table3$rate, pattern = "/")
temp
```

```
## [[1]]
## [1] "745"      "19987071"
##
## [[2]]
## [1] "2666"     "20595360"
##
## [[3]]
## [1] "37737"     "172006362"
##
## [[4]]
## [1] "80488"     "174504898"
##
## [[5]]
## [1] "212258"     "1272915272"
##
## [[6]]
## [1] "213766"     "1280428583"
```

- 결과물이 `list`의 형태로 나오며, 각 `list`의 element는 `vector`
- 해당 `list`의 각 element인 `vector`에서
  1. 첫 번째 element가 `cases`
  2. 두 번째 element가 `population`

```
sapply(temp, function(x) x[1])
```

```
## [1] "745"    "2666"   "37737"  "80488"  "2122
```

```
sapply(temp, function(x) x[2])
```

```
## [1] "19987071"   "20595360"   "172006362"  "1
## [6] "1280428583"
```

```
table3$cases <-
  str_split(string = table3$rate, pattern = "/") %>%
  sapply(function(x) x[1])
table3$population <-
  str_split(string = table3$rate, pattern = "/") %>%
  sapply(function(x) x[2])
table3 %>% select(-rate)
```

```
## # 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
```

- **stringr** 패키지의 함수는 사후 처리 때문에 어렵게 느껴질수도 있지만! 매우 빠른 처리를 해주는 장점이 있음!

# Another usage of tidyr::separate

```
table1 %>% separate(year, into = c("century", "year"), sep = 2)
```

```
## # A tibble: 6 x 5
##   country     century year   cases population
##   <chr>       <chr>   <chr>  <int>      <int>
## 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
```

# Concatenating

- Before

```
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
```

- After

```
table3
```

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

## Concatenating using `tidyr::unite()`

```
table1 %>% unite(rate, cases, population, sep = "/")
```

```
## # 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
```

# Or, simply use base::paste()

```
table1$rate <- paste(table1$cases, table1$population, sep="/")
table1
```

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

Section 4

# IV. Joining (dplyr)

# Dataset

```
df1 <- data.frame(CustomerId = c(1:5), Product = c(rep("Toaster", 3), rep("Radio", 2)))
df2 <- data.frame(CustomerId = c(2, 4, 6), State = c(rep("Seoul", 2), rep("Busan", 1)))
df1
```

```
##   CustomerId Product
## 1          1 Toaster
## 2          2 Toaster
## 3          3 Toaster
## 4          4   Radio
## 5          5   Radio
```

```
df2
```

```
##   CustomerId State
## 1          2 Seoul
## 2          4 Seoul
## 3          6 Busan
```

- df1과 df2를 어떻게 합할까?
- 어느쪽이 primary?
- 결측치가 생겨도 상관없음?

```
library(dplyr)
library(sqldf) # Use SQL syntax in R!
```

```
## Loading required package: gsubfn
## Loading required package: proto
## Loading required package: RSQLite
```

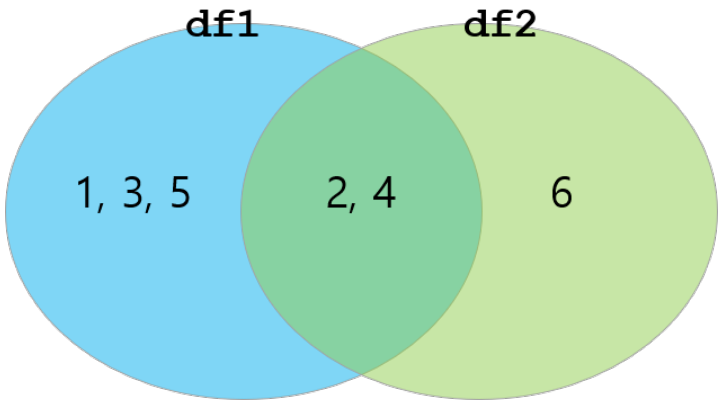
# Join(Merge)의 4가지 모드

```
df1
```

```
##   CustomerId Product
## 1          1 Toaster
## 2          2 Toaster
## 3          3 Toaster
## 4          4   Radio
## 5          5   Radio
```

```
df2
```

```
##   CustomerId State
## 1          2 Seoul
## 2          4 Seoul
## 3          6 Busan
```




1. inner
2. left
3. full (outer)
4. right

# 1. Inner join

```
inner_join(df1, df2) # dplyr
merge(x = df1, y = df2, by = "CustomerId") # base
sqldf("SELECT * FROM df1 JOIN df2 USING(CustomerID)") # sqldf
```

```
## Joining, by = "CustomerId"
##   CustomerId Product State
## 1          2 Toaster Seoul
## 2          4   Radio Seoul
```

# 2. Left join

```
left_join(df1, df2) # dplyr
merge(x = df1, y = df2, by = "CustomerId", all.x = TRUE) # base
sqldf("SELECT * FROM df1 LEFT JOIN df2 USING(CustomerID)") # sqldf
```

```
## Joining, by = "CustomerId"
##   CustomerId Product State
## 1          1 Toaster  <NA>
## 2          2 Toaster Seoul
## 3          3 Toaster  <NA>
## 4          4   Radio Seoul
## 5          5   Radio  <NA>
```

# 3. Outer join (full)

```
full_join(df1, df2) # dplyr
merge(x = df1, y = df2, by = "CustomerId", all = TRUE) # base
# sqldf does not support FULL JOIN, needing three lines.
a <- sqldf("SELECT * FROM df1 LEFT JOIN df2 USING(CustomerID)")
b <- sqldf("SELECT * FROM df2 LEFT JOIN df1 USING(CustomerID)")
union(a,b)
```

```
## Joining, by = "CustomerId"
##   CustomerId Product State
## 1          1 Toaster  <NA>
## 2          2 Toaster Seoul
## 3          3 Toaster  <NA>
## 4          4   Radio Seoul
## 5          5   Radio  <NA>
## 6          6    <NA> Busan
```

- Set operations
  - `union(a,b)`
  - `intersect(a,b)`
  - `setdiff(a,b)`
  - `setdiff(b,a)`

# 4. Right join

```r
right_join(df1, df2) # dplyr
merge(x = df1, y = df2, by = "CustomerId", all.y = TRUE) # base
# sqldf does not support RIGHT JOIN, just swapping the input order.
sqldf("SELECT * FROM df2 LEFT JOIN df1 USING(CustomerID)") # sqldf
```

```
## Joining, by = "CustomerId"
##   CustomerId Product State
## 1          2 Toaster Seoul
## 2          4   Radio Seoul
## 3          6    <NA> Busan
```

- join할때 사용할 key 변수의 지정
- by = argument로 key 변수를 아래처럼 지정한다.
- by = 을 입력하지 않으면 같은 이름의 변수를 찾아서 자동으로 key로 사용한다.

```
inner_join(df1, df2)
inner_join(x=df1, y=df2)
inner_join(x=df1, y=df2, by = "CustomerId")
inner_join(x=df1, y=df2, by = c("CustomerId"))
inner_join(x=df1, y=df2, by = c("CustomerId"="CustomerId"))
```

- vlookup()이나 index()-match()함수를 이용해서 엑셀 파일 합해본 경험있나요?
- R에서는 이게 정말 끝입니다.

- 데이터의 각 관찰값들에 대해 1개 혹은 2개의 key 변수를 가지고 관리되고 있습니까?
- 아니라면 직관적인 처리를 위해서 만드는 것이 좋다.
- key 변수는 알기 쉽고, 중복되지 않고, 체계적인 규칙을 가지고 있어야 한다.

Section 5

# V. NA의 처리

# Motivation

```
df3 <- full_join(df1, df2)
df3$Population <- c(NA, 1000, NA, 1000, NA, 200)
df3
```

```
##   CustomerId Product State Population
## 1          1 Toaster  <NA>         NA
## 2          2 Toaster Seoul       1000
## 3          3 Toaster  <NA>         NA
## 4          4   Radio Seoul       1000
## 5          5   Radio  <NA>         NA
## 6          6    <NA> Busan        200
```

- 여러가지 이유로 위처럼 NA (결측치)가 생긴다.
- 주로 3가지 방법으로 해결한다.
  1. 결측치가 있는 관찰값을 제거
  2. 그대로 두고 함수를 적용할 때 주의해서 분석
  3. 결측치를 다른 수치로 대체 (평균, 0 등의 값)

# 1. 결측치가 있는 관찰값을 제거하기

**dplyr**

```
df3 %>% filter(!is.na(State))
```

```
##   CustomerId Product State Population
## 1          2 Toaster Seoul       1000
## 2          4   Radio Seoul       1000
## 3          6    <NA> Busan        200
```

**base**

```
is.na(df3$State)
```

```
## [1]  TRUE FALSE  TRUE FALSE  TRUE FALSE
```

```
is.na(df3$State) %>% which()
```

```
## [1] 1 3 5
```

```
df3[-(is.na(df3$State) %>% which()),]
```

```
##   CustomerId Product State Population
## 2          2 Toaster Seoul       1000
## 4          4   Radio Seoul       1000
## 6          6    <NA> Busan        200
```

- 아래 명령들도 같은 결과를 만들어 낸다.

```
df3[!is.na(df3$State),]
df3[which(!is.na(df3$State)),]
```

# 2. 그대로 두고 함수를 적용할 때 주의해서 분석

- 많은 함수들에서 na.rm=TRUE의 옵션 사용이 가능하다.

```
mean(df3$Population)
```

```
## [1] NA
```

```
mean(df3$Population, na.rm = TRUE)
```

```
## [1] 733.3333
```

```
sd(df3$Population)
```

```
## [1] NA
```

```
sd(df3$Population, na.rm = TRUE)
```

```
## [1] 461.8802
```

# 3. 결측치를 다른 수치로 대체하기 (평균, 0등의 값)

**dplyr**

```
df3 %>% mutate(
  Population =
    if_else(
      is.na(Population), # Condition
      mean(Population, na.rm = TRUE), # If TRUE
      Population)) # If FALSE
```

```
##   CustomerId Product State Population
## 1          1 Toaster  <NA>   733.3333
## 2          2 Toaster Seoul  1000.0000
## 3          3 Toaster  <NA>   733.3333
## 4          4   Radio Seoul  1000.0000
## 5          5   Radio  <NA>   733.3333
## 6          6    <NA> Busan   200.0000
```

**base**

```
is.na(df3$Population)
```

```
## [1]  TRUE FALSE  TRUE FALSE  TRUE FALSE
```

```
df3$Population[is.na(df3$Population)] <-
  mean(df3$Population, na.rm = TRUE)
df3
```

```
##   CustomerId Product State Population
## 1          1 Toaster  <NA>   733.3333
## 2          2 Toaster Seoul  1000.0000
## 3          3 Toaster  <NA>   733.3333
## 4          4   Radio Seoul  1000.0000
## 5          5   Radio  <NA>   733.3333
## 6          6    <NA> Busan   200.0000
```

'I always like to hire a lazy person to do a difficult job, because a lazy person will find an easy way to do it.' – Bill Gates