

Part II. Wrangle

Spring 2019 - Statistical Graphics

```
library(tidyverse)

## -- Attaching packages ----- tidyverse 1.2.1 --

## √ ggplot2 3.1.0      √ purrr  0.2.5
## √ tibble  1.4.2      √ dplyr  0.7.8
## √ tidyr   0.8.2      √ stringr 1.3.1
## √ readr   1.2.1      √ forcats 0.3.0

## -- Conflicts ----- tidyverse_conflicts()
##
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

7. Tibbles with tibble

- data frame 을 좀 더 편하게 사용할 수 있도록 변형시켜 놓은 class
- data frame 과 상호교환 가능

tibble 만들기

- as_tibble(): data frame 을 tibble 로 바꿔주는 함수

```
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()`을 이용하여 만들기

```
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
```

- data frame 과의 차이점
- character 가 factor 로 바뀌지 않음
- R 에서 허용하지 않는 형태의 변수 이름 가능 (예: ':')
- printing 에서 처음 10 줄과 화면에 맞는 변수만을 보여줌

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

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

- SAS 의 cards 문과 같은 형태의 자료입력도 허용

```
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
```

Printing

- data frame: 모든 자료를 보여줌
- tibble: 처음 10 줄과 화면에 맞는 만큼의 변수만을 보여주며 변수의 type 도 함께 보여줌.

```
tibble(  
  # now()는 시간 포함  
  a = lubridate::now() + runif(1e3) * 86400,  
  b = lubridate::today() + runif(1e3) * 30,  
  c = 1:1e3,  
  d = runif(1e3),  
  e = sample(letters, 1e3, replace = TRUE)  
)  
  
## # A tibble: 1,000 x 5  
##       a                b                c                d e  
##   <dtm>          <date>          <int>      <dbl> <chr>  
## 1 2019-03-08 03:15:43 2019-03-26         1 0.787 y  
## 2 2019-03-07 23:45:05 2019-03-27         2 0.107 y  
## 3 2019-03-08 07:21:58 2019-03-31         3 0.908 c  
## 4 2019-03-08 11:50:23 2019-03-07         4 0.467 z  
## 5 2019-03-07 23:48:55 2019-03-13         5 0.639 u  
## 6 2019-03-07 21:57:20 2019-04-01         6 0.119 h  
## 7 2019-03-07 16:34:52 2019-04-03         7 0.476 l  
## 8 2019-03-07 15:50:38 2019-03-16         8 0.955 h  
## 9 2019-03-08 07:35:49 2019-03-19         9 0.00224 k  
## 10 2019-03-07 18:26:41 2019-03-30        10 0.775 f  
## # ... with 990 more rows
```

- n, width 옵션을 이용하여 자료 전체를 볼 수 있음.

```
nycflights13::flights %>%  
  print(n = 10, width = Inf)
```

- default print option 을 바꿀수도 있음 .
- options(tibble.print_max = n, tibble.print_min = m): 자료가 m 줄 이상인 경우 처음 n 줄만을 인쇄.
- options(dplyr.print_min = Inf): 항상 모든 자료를 인쇄.
- options(tibble.width = Inf): 항상 모든 변수를 인쇄
- View(): 자료 전체를 보여줌

```
nycflights13::flights %>%  
  View()
```

Subsetting

- data frame 과 동일한 방법 이용 가능

```
df <- tibble(  
  x = runif(5),  
  y = rnorm(5)  
)  
  
# Extract by name  
df$x  
  
## [1] 0.3506448 0.1151263 0.3695623 0.9724234 0.3645855  
  
df[["x"]]  
  
## [1] 0.3506448 0.1151263 0.3695623 0.9724234 0.3645855  
  
# Extract by position  
df[[1]]  
  
## [1] 0.3506448 0.1151263 0.3695623 0.9724234 0.3645855
```

- pipe 에서는 .을 이용.

```
df %>% .$x 앞에서 넘어온 자료의 x변수  
  
## [1] 0.3506448 0.1151263 0.3695623 0.9724234 0.3645855  
  
df %>% .[["x"]]  
  
## [1] 0.3506448 0.1151263 0.3695623 0.9724234 0.3645855
```

- partial matching 은 불가능 (data frame 에서는 가능).

```
df.tbl <- tibble(  
  xx = runif(5),  
  y = rnorm(5)  
)  
df.DF <- data.frame(  
  xx = runif(5),  
  y = rnorm(5)  
)  
df.tbl$x data프레임은 비슷한걸 뽑아오지 않는다.  
  
## Warning: Unknown or uninitialised column: 'x'.  
  
## NULL  
  
df.DF$x data프레임은 비슷한걸 뽑아온다.  
  
## [1] 0.9085554 0.3030430 0.8651596 0.5125517 0.8664857
```

- `as.data.frame()` : tibble 을 data frame 으로 바꾸기

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

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

8. Data import with readr

- readr package 의 flat file 을 부르는 함수들
- read_csv() : 자료가 ,로 분리된 형태의 파일을 읽는다.
- read_csv2() : 자료가 ;로 분리된 형태의 파일을 읽는다. (,가 소숫점을 대신 하는 나라의 경우 많이 이용)
- read_tsv() : 자료가 (tab)으로 분리된 형태의 파일을 읽는다
- read_delim() : delim 에 설정된 형태로 분리된 파일을 읽는다.
- read_fwf() : 고정폭으로 된 파일을 읽는다. field 를 fwf_widths()로 지정하거나 position 을 fwf_positions()로 지정
- read_table() : 공백으로 분리된 형태의 파일 읽기

```
#heights <- read_csv("./data/heights.csv")
```

- read_csv()를 실행시키면 parsing 결과로 나타나는 각 변수의 이름과 type 을 보여줌.
- inline 형태로도 이용 가능

```
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
```

- read_csv()의 첫줄의 자료를 column 이름으로 이용.
- skip=n 을 이용하여 n 번째 줄의 이름을 column 이름으로 하고 그 이후의 자료를 읽을 수도 있음.
- comment="문자"를 지정하여 특정 문자로 시작되는 줄을 빼고 읽을 수도 있음.

```
read_csv("The first line 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
```

- col_names = FALSE 를 이용하여 첫줄을 자료로 읽기.

```
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
```

- col_names 를 이용하여 column 의 이름을 지정 가능.

```
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
```

- na="문자"를 이용하여 특정 문자를 NA 로 처리.

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

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

- read_csv vs. read.csv()와 비교

1. read.csv 보다 10 배정도 빠르다. 파일이 큰 경우 progress bar 를 제공하여 상황을 알수 있게 한다.
2. tibble 자료를 생성. character 의 경우 factor 로 바뀌지 않고 character 로 남아있게 됨.
3. read.csv 은 OS 시스템마다 다르게 작동. read_csv 가 더 reproducible 하다.

Parsing a vector

- `parse_type` 함수: character vector 로 읽은 후 type 의 형태로 바꿔줌.

```
str(parse_logical(c("TRUE", "FALSE", "NA")))
```

```
## logi [1:3] TRUE FALSE NA
```

```
str(parse_integer(c("1", "2", "3")))
```

```
## int [1:3] 1 2 3
```

```
str(parse_date(c("2010-01-01", "1979-10-14")))
```

```
## Date[1:2], format: "2010-01-01" "1979-10-14"
```

- `na="문자"`를 이용하여 특정 문자를 NA 로 처리.

```
parse_integer(c("1", "231", ".", "456"), na = ".")
```

```
## [1] 1 231 NA 456
```

- parsing 에 실패하는 경우 warning 을 주고 NA 로 처리.
- `problems()`로 문제점 파악 가능

```
x <- parse_integer(c("123", "345", "abc", "123.45"))
```

```
## Warning: 2 parsing failures.
```

```
## row col expected actual
```

```
## 3 -- an integer abc
```

```
## 4 -- no trailing characters .45
```

```
x
```

```
## [1] 123 345 NA NA
```

```
## attr(,"problems")
```

```
## # A tibble: 2 x 4
```

```
## row col expected actual
```

```
## <int> <int> <chr> <chr>
```

```
## 1 3 NA an integer abc
```

```
## 2 4 NA no trailing characters .45
```

```
problems(x)
```

```
## # A tibble: 2 x 4
```

```
## row col expected actual
```

```
## <int> <int> <chr> <chr>
```

```
## 1 3 NA an integer abc
```

```
## 2 4 NA no trailing characters .45
```

- parser 함수들

1. `parse_logical()` : logical value 를 parsing
2. `parse_integer()`: integers 를 parsing.
3. `parse_double()`: 숫자에 대한 엄격한 parser
4. `parse_number()`: 숫자에 대한 좀 덜엄격한 parser. 나라마다의 다른 형태까지 가능
5. `parse_character()`: 문자열 parsing. 문제는 characer encodings.
6. `parse_factor()`: factor 만들기. R 에서 범주형 변수에 대응.
7. `parse_datetime()`: 날짜와 시간
8. `parse_date()`: 날짜
9. `parse_time()`: 시간

Numbers

- 숫자를 분석의 문제점

1. 숫자를 쓰는데에 소숫점을 ,로 쓰는 나라들도 있다.
 2. "\$1000" or "10%"와 같이 숫자 앞, 뒤에 단위 등의 문자가 올 수 있다.
 3. grouping 을 나타내는 문자가 있을 수 있고 이는 나라마다 다를 수 있다. 예) 1000 단위마다 ,를 표시 등등...
- 나라마다 다를 수 있는 소숫점을 지정하기 위하여 locale 을 이용. default 는 US-centric.

```
parse_double("1.23")
```

```
## [1] 1.23
```

```
parse_double("1,23", locale = locale(decimal_mark = ","))
```

```
## [1] 1.23
```

- `parse_number()`: 숫자 앞, 뒤의 특수문자들을 가려냄.

```
parse_number("$100")
```

```
## [1] 100
```

```
parse_number("20%")
```

```
## [1] 20
```

```
parse_number("It cost $123.45")
```

```
## [1] 123.45
```

- grouping_mark 옵션으로 grouping 문자 가려내기. default 는 grouping_mark=",".

```
# Used in America
```

```
parse_number("$123,456,789")
```

```
## [1] 123456789
```

```
# Used in many parts of Europe
```

```
parse_number("123.456.789", locale = locale(grouping_mark = "."))
```

```
## [1] 123456789
```

```
# Used in Switzerland
```

```
parse_number("123'456'789", locale = locale(grouping_mark = "'"))
```

```
## [1] 123456789
```

Strings

- charToRaw(): 문자를 16 진수로 표현

```
charToRaw("Hadley")
```

```
## [1] 48 61 64 6c 65 79
```

- readr 은 자료를 읽고 쓸 때에 UTF-8 encoding 을 이용.

```
x1 <- "El Ni\xf1o was particularly bad this year"
```

```
x2 <- "\x82\xb1\x82\xf1\x82\xc9\x82xbf\x82xcd"
```

```
x1
```

```
## [1] "El Ni\xf1o was particularly bad this year"
```

```
x2
```

```
## [1] "꺄꺀꺁꺂꺃꺄꺅"
```

- parse_character()를 이용하여 encoding 방식 지정

```
parse_character(x1, locale = locale(encoding = "Latin1"))
```

```
## [1] "El Nino was particularly bad this year"
```

```
parse_character(x2, locale = locale(encoding = "Shift-JIS"))
```

```
## [1] "こんにちハ"
```

- guess_encoding(): 정확한 encoding 방식을 모를 경우 이용.

```
guess_encoding(charToRaw(x1))
```

```
## # A tibble: 2 x 2
##   encoding confidence
##   <chr>          <dbl>
## 1 ISO-8859-1      0.46
## 2 ISO-8859-9      0.23
```

```
guess_encoding(charToRaw(x2))
```

```
## # A tibble: 1 x 2
##   encoding confidence
##   <chr>          <dbl>
## 1 KOI8-R          0.42
```

Factors

- `parse_factor()`: levels 를 지정하여 잘못입력된 범주가 있는 경우 warning 이 나오도록 해줌.

```
fruit <- c("apple", "banana")
parse_factor(c("apple", "banana", "bananana"), levels = fruit)
```

```
## Warning: 1 parsing failure.
## row col      expected actual
##   3  -- value in level set bananana

## [1] apple  banana <NA>
## attr(,"problems")
## # A tibble: 1 x 4
##   row col expected      actual
##   <int> <int> <chr>      <chr>
## 1     3    NA value in level set bananana
## Levels: apple banana
```

Dates, date-times, and times

- 날짜에 대한 3 가지 변환함수
- `date`: 1970 년 1 월 1 일을 기준으로 하여 기준 날로 부터의 날 수를 나타냄.
- `date-time`: 기준일부터의 시간을 초단위로 나타냄.
- `time`: 자정 이후부터의 시간을 초단위로 나타냄.

`parse_datetime()`: ISO8601 기준의 date-time 을 나타냄.

```
```r
parse_datetime("2010-10-01T2010")
```
```

```

```
[1] "2010-10-01 20:10:00 UTC"
```

```r
If time is omitted, it will be set to midnight
parse_datetime("20101010")
```

```
[1] "2010-10-10 UTC"
```

```

- `parse_date()`: 4 자리 연도와 두자리의 달, 날짜를 /나 -로 분리하여 표시하는 경우 이용.
- `parse_date()` expects a four digit year, a - or /, the month, a - or /, then the day:

```

parse_date("2010-10-01")
## [1] "2010-10-01"

```

`parse_time()`은 시간:분:초로 기록하는 경우 이용

```

library(hms)
parse_time("01:10 am")
## 01:10:00
parse_time("20:10:01")
## 20:10:01

```

- Year
 - %Y (4 digits).
 - %y (2 digits); 00-69 -> 2000-2069, 70-99 -> 1970-1999.
- Month
 - %m (2 digits).
 - %b (abbreviated name, like "Jan").
 - %B (full name, "January").
- Day
 - %d (2 digits).
 - %e (optional leading space).
- Time

- %H 0-23 hour.
- %I 0-12, must be used with %p.
- %p AM/PM indicator.
- %M minutes.
- %S integer seconds.
- %OS real seconds.
- %Z 타임존
- %z (as offset from UTC, e.g. +0800).
- Non-digits
- %. 숫자가 아닌 문자를 하나 skip.
- %* 숫자가 아닌 문자를 모두 skip.

```
parse_date("01/02/15", "%m/%d/%y")
## [1] "2015-01-02"

parse_date("01/02/15", "%d/%m/%y")
## [1] "2015-02-01"

parse_date("01/02/15", "%y/%m/%d")
## [1] "2001-02-15"
```

- 영어가 아닌 달의 이름은 %b, %B 와 함께 사용하려면 locale 내에 lang 옵션을 이용

```
parse_date("1 janvier 2015", "%d %B %Y", locale = locale("fr"))
## [1] "2015-01-01"
```

Parsing a file

- readr 은 처음 1000 개의 줄을 읽어서 밝혀낸 각 column 의 타입을 이용.
- guess_parser()와 parse_parser() 를 함께 이용하는 것임.

```
guess_parser("2010-10-01")
## [1] "date"

guess_parser("15:01")
## [1] "time"

guess_parser(c("TRUE", "FALSE"))
## [1] "logical"

guess_parser(c("1", "5", "9"))
```

```
## [1] "double"
guess_parser(c("12,352,561"))
## [1] "number"
str(parse_guess("2010-10-10"))
## Date[1:1], format: "2010-10-10"
```

- 아래의 type 중 하나를 시도. 어디에도 해당되지 않으면 character 그대로.
- logical: "F", "T", "FALSE", 혹은 "TRUE".
- integer: 숫자와 -만으로 표현
- double: 가능한 모든 실수 표현과 비교 (예: 4.5e-5).
- number: 모든 숫자 표현과 비교 (소숫점, grouping mark 포함).
- time: time_format 과 일치
- date: date_format 과 일치
- date-time: ISO8601 방식의 날짜.

위의 type 중 어디에도 해당이 되지 않는다면 string vector 로 그대로 놓아둔다.

Problems

- 큰 파일에서 발생가능한 문제들
 1. 처음 1000 개의 자료가 특별한 상황일 수 있으므로 이를 이용하여 guess 한 것을 그 이후의 자료에 대한 변환에 이용하는 readr 에 문제가 발생할 수 있다.
 2. missing 이 많은 column 이 있을 수 있다. 특정 column 이 처음 1000 개의 자료가 모두 NA 인 경우 type 을 지정할 수가 없게된다.

```
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
##                                file
## 1001   y 1/0/T/F/TRUE/FALSE 2015-01-16 'C:/Users/EKLee/Documents/R/win-lib
rary/3.5/readr/extdata/challenge.csv'
```

```
## 1002 y 1/0/T/F/TRUE/FALSE 2018-05-18 'C:/Users/EKLee/Documents/R/win-lib
rary/3.5/readr/extdata/challenge.csv'
## 1003 y 1/0/T/F/TRUE/FALSE 2015-09-05 'C:/Users/EKLee/Documents/R/win-lib
rary/3.5/readr/extdata/challenge.csv'
## 1004 y 1/0/T/F/TRUE/FALSE 2012-11-28 'C:/Users/EKLee/Documents/R/win-lib
rary/3.5/readr/extdata/challenge.csv'
## 1005 y 1/0/T/F/TRUE/FALSE 2020-01-13 'C:/Users/EKLee/Documents/R/win-lib
rary/3.5/readr/extdata/challenge.csv'
## ....
## See problems(...) for more details.
```

`readr_example()`: *readr* package 의 *example* 파일을 찾아줌. 처음 1000 줄에 대한 error message 와 그 다음 5 줄에 대한 결과를 함께 보여줌.

```
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~ 'C:/Users/EKLee/Documents/R/win-lib~
## 2 1002 y 1/0/T/F/TRUE~ 2018-05~ 'C:/Users/EKLee/Documents/R/win-lib~
## 3 1003 y 1/0/T/F/TRUE~ 2015-09~ 'C:/Users/EKLee/Documents/R/win-lib~
## 4 1004 y 1/0/T/F/TRUE~ 2012-11~ 'C:/Users/EKLee/Documents/R/win-lib~
## 5 1005 y 1/0/T/F/TRUE~ 2020-01~ 'C:/Users/EKLee/Documents/R/win-lib~
## 6 1006 y 1/0/T/F/TRUE~ 2016-04~ 'C:/Users/EKLee/Documents/R/win-lib~
## 7 1007 y 1/0/T/F/TRUE~ 2011-05~ 'C:/Users/EKLee/Documents/R/win-lib~
## 8 1008 y 1/0/T/F/TRUE~ 2020-07~ 'C:/Users/EKLee/Documents/R/win-lib~
## 9 1009 y 1/0/T/F/TRUE~ 2011-04~ 'C:/Users/EKLee/Documents/R/win-lib~
## 10 1010 y 1/0/T/F/TRUE~ 2010-05~ 'C:/Users/EKLee/Documents/R/win-lib~
## # ... with 990 more rows
```

- `read_csv` 함수 내에 `col_types` 옵션을 이용.

```
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
```

- `y = col_date()` 이용

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

```
## # A tibble: 6 x 2
##       x y
##   <dbl> <date>
## 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
```

Other strategies

- `guess_max`: guess 에 이용할 줄 수를 지정.

데이터 recog를 1001번까지, default: 1000

```
challenge2 <- read_csv(readr_example("challenge.csv"), guess_max = 1001)
```

```
## Parsed with column specification:
## cols(
##   x = col_double(),
##   y = col_date(format = "")
## )
```

```
challenge2
```

```
## # A tibble: 2,000 x 2
##       x y
##   <dbl> <date>
## 1  404 NA
## 2 4172 NA
## 3 3004 NA
## 4  787 NA
## 5   37 NA
## 6 2332 NA
## 7 2489 NA
## 8 1449 NA
## 9 3665 NA
```



```
## 10 3863 NA
## # ... with 1,990 more rows
```

- 모든 자료를 character 로 읽은 후 확인.

```
challenge2 <- read_csv(readr_example("challenge.csv"),
  col_types = cols(.default = col_character())
)
```

- type_convert() 를 함께 이용.

```
df <- tribble(
  ~x, ~y,
  "1", "1.21",
  "2", "2.32",
  "3", "4.56"
)
df

## # A tibble: 3 x 2
##   x     y
##   <chr> <chr>
## 1 1     1.21
## 2 2     2.32
## 3 3     4.56

# Note the column types
type_convert(df)

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

## # A tibble: 3 x 2
##       x     y
##   <dbl> <dbl>
## 1     1  1.21
## 2     2  2.32
## 3     3  4.56
```

- 큰 파일을 읽을 때에는 n_max 옵션을 이용하여 자료의 일부를 읽어 문제를 해결한 후 전체 자료를 읽는 것이 좋다.
- 심각한 parsing 문제가 있는 경우에는 read_lines()나 read_file()를 이용하여 character 로 자료를 읽은 후 format 등을 이용하여 parsing.

Writing to a file

- `write_csv()` 와 `write_tsv()`
- string 은 UTF-8 로 encoding
- date, date-time 은 ISO8601 format 으로 저장.
- `write_excel_csv()`: excel 파일로 저장
- 파일의 시작 부분에 UTF-8 로 encoding 을 하고 있다는 것을 excel 에 알려주는 특별 문자("byte order mark")를 써줌

```
write_csv(challenge, "challenge.csv")
```

- csv 로 저장할 때에는 type 에 대한 정보는 잃어버림

```
challenge
```

```
## # A tibble: 2,000 x 2
##       x y
##   <dbl> <date>
## 1   404 NA
## 2  4172 NA
## 3  3004 NA
## 4   787 NA
## 5    37 NA
## 6  2332 NA
## 7  2489 NA
## 8  1449 NA
## 9  3665 NA
## 10 3863 NA
## # ... with 1,990 more rows
```

```
write_csv(challenge, "challenge-2.csv")
#read_csv("challenge-2.csv")
```

- `write_rds()`와 `read_rds()`를 이용.
- R 의 기본함수인 `saveRDS()`와 `readRDS()`에 해당
- binary format 으로 저장.

```
write_rds(challenge, "challenge.rds")
#read_rds("challenge.rds")
```

- feather package 를 이용
- 프로그래밍 언어 간에 통용되는 형태의 binary file format 으로 저장

```
library(feather)
write_feather(challenge, "challenge.feather")
```

```
read_feather("challenge.feather")
```

```
#> # A tibble: 2,000 x 2
```

```
#>       x       y
```

```
#>   <dbl> <date>
```

```
#> 1   404   <NA>
```

```
#> 2  4172   <NA>
```

```
#> 3  3004   <NA>
```

```
#> 4   787   <NA>
```

```
#> 5    37   <NA>
```

```
#> 6  2332   <NA>
```

```
#> # ... with 1,994 more rows
```

9. Tidy data with dplyr

Introduction

Tidy data

- 같은 자료를 4 가지의 다른 방법으로 표현

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
```

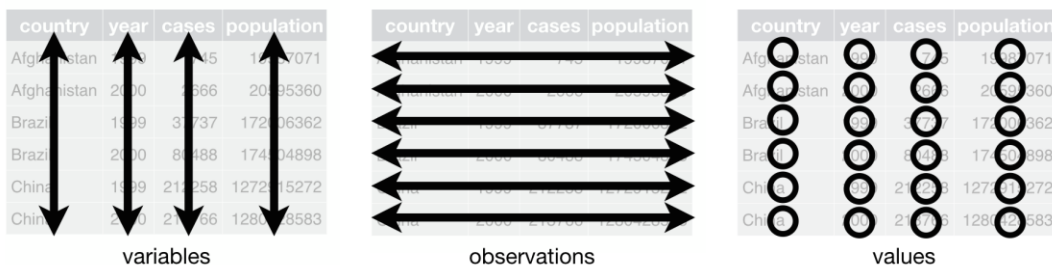
```
# Spread across two tibbles
table4a # cases

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

table4b # population

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

- dataset 을 tidy 하게 만들기 위한 3 가지의 기본 규칙.
1. 각 변수는 각자의 column 을 가지고 있어야 한다.
 2. 각 observation 은 각자의 row 를 가지고 있어야 한다.
 3. 각 값은 각자의 cell 을 가지고 있어야 한다.



Following three rules makes a dataset tidy: variables are in columns, observations are in rows, and values are in cells.

- tidy 자료의 장점.
1. 자료를 저장하는데에 일관된 방법을 제공하고 이후 작업을 위한 도구들을 이에 맞춰 개발하므로 배우기 쉽다.
 2. 변수를 column 으로 놓는 것은 특히 유용. R 의 내장된 함수들은 vector 를 기본으로 하고 있으므로 tidy data 를 vector 로 변환하는 것이 유용.

```

# Compute rate per 10,000
table1 %>%
  mutate(rate = cases / population * 10000)

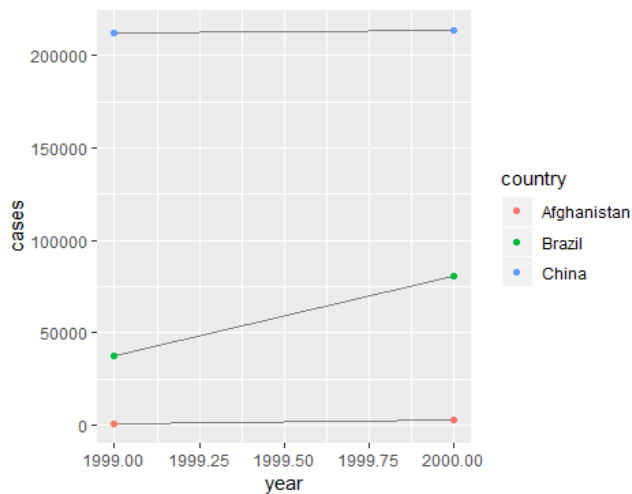
## # A tibble: 6 x 5
##   country    year  cases population  rate
##   <chr>      <int> <int>      <int> <dbl>
## 1 Afghanistan 1999    745    19987071 0.373
## 2 Afghanistan 2000   2666   20595360 1.29
## 3 Brazil       1999  37737  172006362 2.19
## 4 Brazil       2000  80488  174504898 4.61
## 5 China        1999 212258 1272915272 1.67
## 6 China        2000 213766 1280428583 1.67

# Compute cases per year
table1 %>%
  count(year, wt = cases)

## # A tibble: 2 x 2
##   year      n
##   <int> <int>
## 1 1999 250740
## 2 2000 296920

# Visualise changes over time
library(ggplot2)
ggplot(table1, aes(year, cases)) +
  geom_line(aes(group = country), colour = "grey50") +
  geom_point(aes(colour = country))

```



Spreading and gathering

- 대부분의 자료는 분석보다는 모아지기 편리한 형태로 구성되므로 tidy data 로 바꾸는 작업이 필요.

1. 하나의 변수를 multiple column 에 표현해야될 수 있다.
 2. 하나의 관측을 여러 줄에 표현해야될 수 있다.
- gather() 과 spread()의 함수를 이용.

Gathering

table4a

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

- 1999 와 2000 는 변수의 이름이 아니고 year 변수의 값이 됨.
- row 는 하나가 아니고 2 개의 관측을 나타낸다.
- tidy 자료로 만들기 위해서 column 을 새로운 변수의 pair 로 만들어야함. 이를 gathering 이라고 함.
- 이를 위하여 3 가지의 인자를 지정하는 것이 필요.
 1. 값을 나타내는 column 의 이름; table4a 에서는 1999 와 2000
 2. column 의 이름에 나타난 값을 위한 변수 이름 (key); 'year'
 3. column 에서 가지고 있는 값을 저장하기 위한 변수 이름 (value); 'cases'

```
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
```

| country | year | cases | country | 1999 | 2000 |
|-------------|------|--------|-------------|--------|--------|
| Afghanistan | 1999 | 745 | Afghanistan | 745 | 2666 |
| Afghanistan | 2000 | 2666 | Brazil | 37737 | 80488 |
| Brazil | 1999 | 37737 | China | 212258 | 213766 |
| Brazil | 2000 | 80488 | | | |
| China | 1999 | 212258 | | | |
| China | 2000 | 213766 | | | |

table4

Gathering table4 into a tidy form.

- table4b 도 같은 형태로 변형 가능.

```
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
```

- table4a 와 table4b 를 tidy data 로 바꾼 후 dplyr 패키지의 left_join()를 이용하여 하나의 tibble 로 나타내기. (자세한 내용은 relational data 에서)

```
tidy4a <- table4a %>%
  gather(`1999`, `2000`, key = "year", value = "cases")
tidy4b <- table4b %>%
  gather(`1999`, `2000`, key = "year", value = "population")
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

- spreading 은 gathering 의 반대로 observation 이 여러줄에 나타날 때 이용.

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
```

- 위의 자료를 tidy 자료로 바꾸기 위해서는 두가지의 인자가 필요.

1. 변수 이름이 저장되어 있는 key column: 여기서는 type
2. 변수의 값이 저장되어 있는 value column: 여기서는 value

```
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
```

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

Spreading table2 makes it tidy

Separating and uniting

- table3 의 “하나의 column 에 두변수의 값”을 가지고 있는 문제를 separate()과 unite()을 이용하여 해결.

Separate

- separate(): 하나의 column 을 여러 column 으로 바꿔주는 것으로 어떤 문자로 나누어져있던 간에 나 여러 column 으로 분할해 줌.

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 를 cases 와 population 으로 분리.

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
```



| 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 |

seperation을 하면 항상 캐릭터로 저장된다

Separating table3 makes it tidy

*sep 을 이용하여 특정 문자로 분할

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

## # 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
```

- convert 옵션을 이용하여 알맞는 type 으로 변경

```
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
```

각각 맞는 type으로 변환해준다.

```
## 4 Brazil      2000  80488 174504898
## 5 China       1999 212258 1272915272
## 6 China       2000 213766 1280428583
```

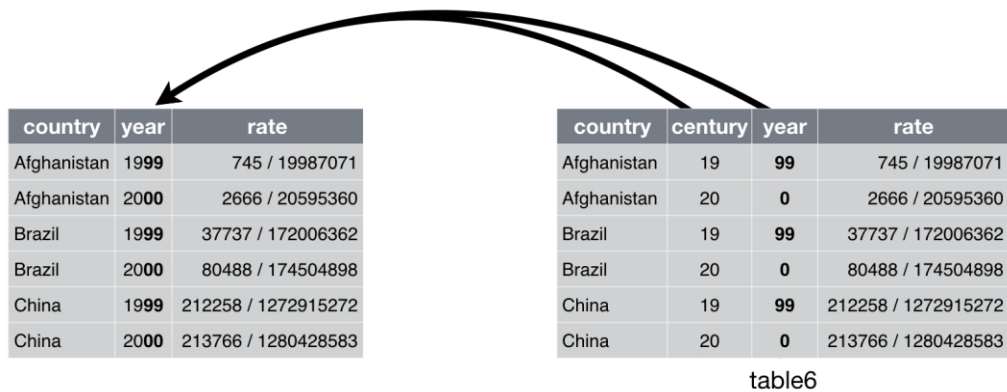
- sep=k 로 나누는 문자 개수 지정.

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

## # A tibble: 6 x 4                2개씩 끊기, -2라면 뒤에서 2개
##   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

*unite(): 여러개의 column 을 하나의 column 으로 합쳐줌.



Uniting table5 makes it tidy

unite()에서는 두번째 인자부터 나열된 변수들의 값을 “_”를 이용하여 연결한 후 첫번째 인자에 지정된 값의 변수에 저장한다.

```
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
```

- sep 옵션을 이용하여 연결문자 지정.

```
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
```

string 작업 후에는 늘 캐릭터로 바뀐다.

Missing values

- missing 의 종류
- explicitly missing: 자료는 있으나 파악되지 못하여 NA 로 표시
- implicitly missing: 자료가 존재하지 않는 경우

```
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)
)
```

- 2015 년 qtr 4 의 자료 - explicitly missing
- 2016 년 1 분기 - implicitly missing
- spread 를 이용하여 implicit missing 을 explicit missing 으로 만들 수 있음.

```
stocks %>%
  spread(year, return)

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

- explicit missing 을 제거하고 싶은 경우에는 na.rm=TRUE 옵션을 이용.

```
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
```

- complete() 함수를 이용하여 implicit missing 을 explicit missing 으로 변환.

```
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
```

- complete(): 모든 unique combination 을 찾아서 만들어줌. missing 은 NA 로 채우기.

```
treatment <- tribble(
  ~ person,      ~ treatment, ~response,
  "Derrick Whitmore", 1,      7,
  NA,                2,      10,
  NA,                3,      9,
  "Katherine Burke", 1,      4
)
```

- fill():missing 을 LOCF(Last Observation Carried Forward) 규칙에 따라 채워줌.

```
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
```

```
## 1 Derrick Whitmore      1      7
## 2 Derrick Whitmore      2     10
## 3 Derrick Whitmore      3      9
## 4 Katherine Burke       1      4
```

Case Study

- 1980 년부터 2013 년까지의 WHO 의 TB 원자료.
- country : 나라이름
- iso2,iso2 : 나라이름에 대한 code
- new_sp_m014 - new_rel_f65: 그룹별 새로운 결핵(TB:Tuberculosis)환자수
- rel: relapse, sn:negative pulmonary smear, sp:extrapulmonary
- f:female, m:male
- 014:0~14 세, 1524:15~24 세, ..., 65: 65 세이상

who

```
## # A tibble: 7,240 x 60
##   country iso2  iso3  year new_sp_m014 new_sp_m1524 new_sp_m2534
##   <chr>   <chr> <chr> <int>      <int>      <int>      <int>
## 1 Afghan~ AF    AFG   1980         NA         NA         NA
## 2 Afghan~ AF    AFG   1981         NA         NA         NA
## 3 Afghan~ AF    AFG   1982         NA         NA         NA
## 4 Afghan~ AF    AFG   1983         NA         NA         NA
## 5 Afghan~ AF    AFG   1984         NA         NA         NA
## 6 Afghan~ AF    AFG   1985         NA         NA         NA
## 7 Afghan~ AF    AFG   1986         NA         NA         NA
## 8 Afghan~ AF    AFG   1987         NA         NA         NA
## 9 Afghan~ AF    AFG   1988         NA         NA         NA
## 10 Afghan~ AF    AFG   1989         NA         NA         NA
## # ... with 7,230 more rows, and 53 more variables: new_sp_m3544 <int>,
## #   new_sp_m4554 <int>, new_sp_m5564 <int>, new_sp_m65 <int>,
## #   new_sp_f014 <int>, new_sp_f1524 <int>, new_sp_f2534 <int>,
## #   new_sp_f3544 <int>, new_sp_f4554 <int>, new_sp_f5564 <int>,
## #   new_sp_f65 <int>, new_sn_m014 <int>, new_sn_m1524 <int>,
## #   new_sn_m2534 <int>, new_sn_m3544 <int>, new_sn_m4554 <int>,
## #   new_sn_m5564 <int>, new_sn_m65 <int>, new_sn_f014 <int>,
## #   new_sn_f1524 <int>, new_sn_f2534 <int>, new_sn_f3544 <int>,
## #   new_sn_f4554 <int>, new_sn_f5564 <int>, new_sn_f65 <int>,
## #   new_ep_m014 <int>, new_ep_m1524 <int>, new_ep_m2534 <int>,
## #   new_ep_m3544 <int>, new_ep_m4554 <int>, new_ep_m5564 <int>,
## #   new_ep_m65 <int>, new_ep_f014 <int>, new_ep_f1524 <int>,
## #   new_ep_f2534 <int>, new_ep_f3544 <int>, new_ep_f4554 <int>,
## #   new_ep_f5564 <int>, new_ep_f65 <int>, newrel_m014 <int>,
## #   newrel_m1524 <int>, newrel_m2534 <int>, newrel_m3544 <int>,
## #   newrel_m4554 <int>, newrel_m5564 <int>, newrel_m65 <int>,
```

```
## # newrel_f014 <int>, newrel_f1524 <int>, newrel_f2534 <int>,
## # newrel_f3544 <int>, newrel_f4554 <int>, newrel_f5564 <int>,
## # newrel_f65 <int>
```

tidy 자료를 위해 필요한 변수들.

1. country
2. year
3. ?==> 일단 cases 로 new_sp_m014 부터 newrel_f65 변수를 합치기.

```
who1 <- who %>%
  gather(new_sp_m014:newrel_f65, key = "key", value = "cases", na.rm = TRUE)
who1
```

```
## # A tibble: 76,046 x 6
##   country iso2 iso3 year key cases
##   <chr>    <chr> <chr> <int> <chr> <int>
## 1 Afghanistan AF AFG 1997 new_sp_m014 0
## 2 Afghanistan AF AFG 1998 new_sp_m014 30
## 3 Afghanistan AF AFG 1999 new_sp_m014 8
## 4 Afghanistan AF AFG 2000 new_sp_m014 52
## 5 Afghanistan AF AFG 2001 new_sp_m014 129
## 6 Afghanistan AF AFG 2002 new_sp_m014 90
## 7 Afghanistan AF AFG 2003 new_sp_m014 127
## 8 Afghanistan AF AFG 2004 new_sp_m014 139
## 9 Afghanistan AF AFG 2005 new_sp_m014 151
## 10 Afghanistan AF AFG 2006 new_sp_m014 193
## # ... with 76,036 more rows
```

- key 변수 살펴보기

```
who1 %>%
  count(key)

## # A tibble: 56 x 2
##   key n
##   <chr> <int>
## 1 new_ep_f014 1032
## 2 new_ep_f1524 1021
## 3 new_ep_f2534 1021
## 4 new_ep_f3544 1021
## 5 new_ep_f4554 1017
## 6 new_ep_f5564 1017
## 7 new_ep_f65 1014
## 8 new_ep_m014 1038
## 9 new_ep_m1524 1026
## 10 new_ep_m2534 1020
## # ... with 46 more rows
```


- key 값의 의미
1. 첫부분 “new”
 2. 두번째 부분
 - rel :cases of relapse
 - ep : cases of extrapulmonary TB
 - sn : cases of pulmonary TB that could not be diagnosed by a pulmonary smear (smear negative)
 - sp : cases of pulmonary TB that could be diagnosed by a pulmonary smear (smear positive)
 3. 세번째 부분: 환자 성별 males (m) and females (f).
 4. 나머지 숫자들: age group
 - 014 = 0 – 14 years old
 - 1524 = 15 – 24 years old
 - 2534 = 25 – 34 years old
 - 3544 = 35 – 44 years old
 - 4554 = 45 – 54 years old
 - 5564 = 55 – 64 years old
 - 65 = 65 or older
- 먼저 newrel 을 new_rel 로 변환

```
who2 <- who1 %>%
  mutate(key = stringr::str_replace(key, "newrel", "new_rel"))
who2
```

```
## # A tibble: 76,046 x 6
##   country      iso2 iso3  year key      cases
##   <chr>        <chr> <chr> <int> <chr>    <int>
## 1 Afghanistan AF    AFG   1997 new_sp_m014    0
## 2 Afghanistan AF    AFG   1998 new_sp_m014   30
## 3 Afghanistan AF    AFG   1999 new_sp_m014    8
## 4 Afghanistan AF    AFG   2000 new_sp_m014   52
## 5 Afghanistan AF    AFG   2001 new_sp_m014  129
## 6 Afghanistan AF    AFG   2002 new_sp_m014   90
## 7 Afghanistan AF    AFG   2003 new_sp_m014  127
## 8 Afghanistan AF    AFG   2004 new_sp_m014  139
## 9 Afghanistan AF    AFG   2005 new_sp_m014  151
## 10 Afghanistan AF    AFG   2006 new_sp_m014  193
## # ... with 76,036 more rows
```

- key 를 separate() 함수를 이용하여 분리

```

who3 <- who2 %>%
  separate(key, c("new", "type", "sexage"), sep = "_")
who3

## # A tibble: 76,046 x 8
##   country      iso2 iso3   year new   type sexage cases
##   <chr>      <chr> <chr> <int> <chr> <chr> <chr> <int>
## 1 Afghanistan AF    AFG   1997 new   sp    m014     0
## 2 Afghanistan AF    AFG   1998 new   sp    m014    30
## 3 Afghanistan AF    AFG   1999 new   sp    m014     8
## 4 Afghanistan AF    AFG   2000 new   sp    m014    52
## 5 Afghanistan AF    AFG   2001 new   sp    m014   129
## 6 Afghanistan AF    AFG   2002 new   sp    m014    90
## 7 Afghanistan AF    AFG   2003 new   sp    m014   127
## 8 Afghanistan AF    AFG   2004 new   sp    m014   139
## 9 Afghanistan AF    AFG   2005 new   sp    m014   151
## 10 Afghanistan AF    AFG   2006 new   sp    m014   193
## # ... with 76,036 more rows

```

- 필요없는 변수 new, iso2, iso3 제거

```

who3 %>%
  count(new)

## # A tibble: 1 x 2
##   new      n
##   <chr> <int>
## 1 new   76046

who4 <- who3 %>%
  select(-new, -iso2, -iso3)

```

- sexage 를 sex 와 age 로 분리

```

who5 <- who4 %>%
  separate(sexage, c("sex", "age"), sep = 1)
who5

## # A tibble: 76,046 x 6
##   country      year type sex   age cases
##   <chr>      <int> <chr> <chr> <chr> <int>
## 1 Afghanistan 1997 sp    m    014     0
## 2 Afghanistan 1998 sp    m    014    30
## 3 Afghanistan 1999 sp    m    014     8
## 4 Afghanistan 2000 sp    m    014    52
## 5 Afghanistan 2001 sp    m    014   129
## 6 Afghanistan 2002 sp    m    014    90
## 7 Afghanistan 2003 sp    m    014   127
## 8 Afghanistan 2004 sp    m    014   139
## 9 Afghanistan 2005 sp    m    014   151

```

```
## 10 Afghanistan 2006 sp m 014 193
## # ... with 76,036 more rows
```

- pipe 를 이용하여 한번에 처리하기.

```
who %>%
  gather(code, value, new_sp_m014:newrel_f65, na.rm = TRUE) %>%
  mutate(code = stringr::str_replace(code, "newrel", "new_rel")) %>%
  separate(code, c("new", "var", "sexage")) %>%
  select(-new, -iso2, -iso3) %>%
  separate(sexage, c("sex", "age"), sep = 1)
```

10. Relational data with dplyr

Introduction

- 관계형 data: table 로 구성된 자료들 간의 관계를 포함.
- 관계형 data 에서 자료 변환
 1. mutation joins: 하나의 data frame 에 다른 data frame 을 matching 시켜 새로운 변수 만들기.
 2. filtering joins : 하나의 data frame 에서 다른 data frame 의 자료와 맞는 관측이 있는지를 확인하여 자료를 걸러내기.
 3. set operations: 관측을 마치 set element 인것처럼 다루기

```
library(tidyverse)
library(nycflights13)
```

nycflights13

- airlines: 비행사 code 자료

```
airlines

## # A tibble: 16 x 2
##   carrier name
##   <chr>   <chr>
## 1 9E      Endeavor Air Inc.
## 2 AA      American Airlines Inc.
## 3 AS      Alaska Airlines Inc.
## 4 B6      JetBlue Airways
## 5 DL      Delta Air Lines Inc.
## 6 EV      ExpressJet Airlines Inc.
## 7 F9      Frontier Airlines Inc.
## 8 FL      AirTran Airways Corporation
## 9 HA      Hawaiian Airlines Inc.
## 10 MQ     Envoy Air
## 11 OO     SkyWest Airlines Inc.
## 12 UA     United Air Lines Inc.
## 13 US     US Airways Inc.
## 14 VX     Virgin America
## 15 WN     Southwest Airlines Co.
## 16 YV     Mesa Airlines Inc.
```

- airports: 공항 code 자료

```
airports

## # A tibble: 1,458 x 8
##   faa   name          lat   lon   alt   tz dst  tzone
```

```
##      <chr> <chr>                <dbl> <dbl> <int> <dbl> <chr> <chr>
## 1 04G   Lansdowne Airport      41.1  -80.6  1044   -5 A   America/New_~
## 2 06A   Moton Field Municip~  32.5  -85.7   264   -6 A   America/Chic~
## 3 06C   Schaumburg Regional    42.0  -88.1   801   -6 A   America/Chic~
## 4 06N   Randall Airport        41.4  -74.4   523   -5 A   America/New_~
## 5 09J   Jekyll Island Airpo~  31.1  -81.4    11   -5 A   America/New_~
## 6 0A9   Elizabethton Munic~   36.4  -82.2  1593   -5 A   America/New_~
## 7 0G6   Williams County Air~  41.5  -84.5   730   -5 A   America/New_~
## 8 0G7   Finger Lakes Region~  42.9  -76.8   492   -5 A   America/New_~
## 9 0P2   Shoestring Aviation~  39.8  -76.6  1000   -5 U   America/New_~
## 10 0S9  Jefferson County In~  48.1 -123.    108   -8 A   America/Los_~
## # ... with 1,448 more rows
```

- planes: 각 비행기 정보

```
planes

## # A tibble: 3,322 x 9
##   tailnum year type      manufacturer model  engines seats speed engine
##   <chr>   <int> <chr>      <chr>          <chr>   <int> <int> <int> <chr>
## 1 N10156  2004 Fixed wi~ EMBRAER      EMB-1~     2    55    NA Turbo~
## 2 N102UW  1998 Fixed wi~ AIRBUS INDUS~ A320~-     2   182    NA Turbo~
## 3 N103US  1999 Fixed wi~ AIRBUS INDUS~ A320~-     2   182    NA Turbo~
## 4 N104UW  1999 Fixed wi~ AIRBUS INDUS~ A320~-     2   182    NA Turbo~
## 5 N10575  2002 Fixed wi~ EMBRAER      EMB-1~     2    55    NA Turbo~
## 6 N105UW  1999 Fixed wi~ AIRBUS INDUS~ A320~-     2   182    NA Turbo~
## 7 N107US  1999 Fixed wi~ AIRBUS INDUS~ A320~-     2   182    NA Turbo~
## 8 N108UW  1999 Fixed wi~ AIRBUS INDUS~ A320~-     2   182    NA Turbo~
## 9 N109UW  1999 Fixed wi~ AIRBUS INDUS~ A320~-     2   182    NA Turbo~
## 10 N110UW 1999 Fixed wi~ AIRBUS INDUS~ A320~-     2   182    NA Turbo~
## # ... with 3,312 more rows
```

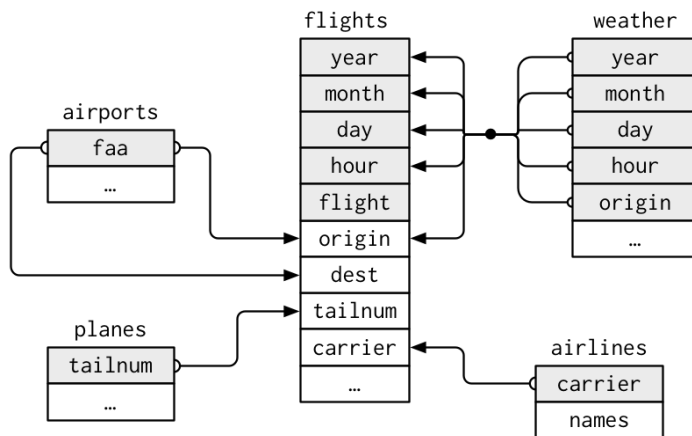
- weather: NYC 공항 날씨 정보

```
weather

## # A tibble: 26,115 x 15
##   origin year month  day  hour  temp  dewp humid wind_dir wind_speed
##   <chr>   <dbl> <dbl> <int> <int> <dbl> <dbl> <dbl>   <dbl>   <dbl>
## 1 EWR    2013     1     1     1  39.0  26.1  59.4     270     10.4
## 2 EWR    2013     1     1     2  39.0  27.0  61.6     250      8.06
## 3 EWR    2013     1     1     3  39.0  28.0  64.4     240     11.5
## 4 EWR    2013     1     1     4  39.9  28.0  62.2     250     12.7
## 5 EWR    2013     1     1     5  39.0  28.0  64.4     260     12.7
## 6 EWR    2013     1     1     6  37.9  28.0  67.2     240     11.5
## 7 EWR    2013     1     1     7  39.0  28.0  64.4     240     15.0
## 8 EWR    2013     1     1     8  39.9  28.0  62.2     250     10.4
## 9 EWR    2013     1     1     9  39.9  28.0  62.2     260     15.0
## 10 EWR   2013     1     1    10  41    28.0  59.6     260     13.8
```

```
## # ... with 26,105 more rows, and 5 more variables: wind_gust <dbl>,
## #   precip <dbl>, pressure <dbl>, visib <dbl>, time_hour <dtm>
```

- nycflights 자료구조



nycflight 자료는 실전에서 만나는 자료보다는 훨씬 간단. 그러나 연습하기에는 충분

- flights 는 planes 와 tailnum 으로 연결
- flights 는 airlines 와 carrier 로 연결
- flights 는 airports 와 origin, dest 로 연결
- flights 는 weather 과 origin, year, month, day, hour 로 연결

Keys

- key: 각 table 을 연결시켜주는 변수. 하나의 변수로 충분할 수도 있고 여러 변수가 필요할 수도 있음.
- 두가지 종류의 key 가 있음.
 1. primary key: table 의 observation 을 각각 구별하는 변수. tailnum 은 plane table 의 각 비행기를 구별
 2. foreign key: 다른 table 에서의 한 관측을 구별. flights 의 tailnum 은 foreign key.
- primary key 에 대해서는 count 를 이용하여 unique 한지 확인하는 것이 필요.

```
planes %>%
  count(tailnum) %>%
  filter(n > 1)
```

```
## # A tibble: 0 x 2
## # ... with 2 variables: tailnum <chr>, n <int>
```

```
weather %>%
  count(year, month, day, hour, origin) %>%
  filter(n > 1)
```

```
## # A tibble: 3 x 6
##   year month   day hour origin     n
##   <dbl> <dbl> <int> <int> <chr>  <int>
## 1  2013    11     3     1 EWR      2
## 2  2013    11     3     1 JFK      2
## 3  2013    11     3     1 LGA      2
```

- 때로는 명확한 primary key 를 갖지 않을수도 있음. 각 row 는 observation 이지만 변수의 combination 으로는 primary key 를 만들 수 없는 경우도 있음.

```
flights %>%
  count(year, month, day, flight) %>%
  filter(n > 1)
```

```
## # A tibble: 29,768 x 5
##   year month   day flight     n
##   <int> <int> <int>   <int> <int>
## 1  2013     1     1       1     2
## 2  2013     1     1       3     2
## 3  2013     1     1       4     2
## 4  2013     1     1     11     3
## 5  2013     1     1     15     2
## 6  2013     1     1     21     2
## 7  2013     1     1     27     4
## 8  2013     1     1     31     2
## 9  2013     1     1     32     2
## 10 2013     1     1     35     2
## # ... with 29,758 more rows
```

```
flights %>%
  count(year, month, day, tailnum) %>%
  filter(n > 1)
```

```
## # A tibble: 64,928 x 5
##   year month   day tailnum     n
##   <int> <int> <int>   <chr>  <int>
## 1  2013     1     1 NØEGMQ     2
## 2  2013     1     1 N11189     2
## 3  2013     1     1 N11536     2
## 4  2013     1     1 N11544     3
## 5  2013     1     1 N11551     2
## 6  2013     1     1 N12540     2
## 7  2013     1     1 N12567     2
## 8  2013     1     1 N13123     2
## 9  2013     1     1 N13538     3
```

```
## 10 2013 1 1 N13566 3
## # ... with 64,918 more rows
```

- 하루 한번이상 운행되는 비행기도 많음.
- surrogate key: table 에 prime key 가 없을 경우 mutate 나 row_number 함수를 이용하여 만드는 것.
- 1 대 다, 1 대 1, 다대다, 다대 1 의 관계가 모두 가능

Mutating joins

- 두 table 을 이용하여 변수를 합성하는 것.
- 먼저 key 를 이용하여 observation 을 맞추고 한 table 에서 다른 table 로 변수를 copy 함.
- mutate 함수와 같이 join 함수는 변수를 오른쪽에 추가.
- 새로운 변수는 인쇄되지 않음.

```
flights2 <- flights %>%
  select(year:day, hour, origin, dest, tailnum, carrier)
flights2
```

```
## # A tibble: 336,776 x 8
##   year month   day hour origin dest  tailnum carrier
##   <int> <int> <int> <dbl> <chr> <chr> <chr>   <chr>
## 1  2013     1     1     5 EWR   IAH   N14228 UA
## 2  2013     1     1     5 LGA   IAH   N24211 UA
## 3  2013     1     1     5 JFK   MIA   N619AA AA
## 4  2013     1     1     5 JFK   BQN   N804JB B6
## 5  2013     1     1     6 LGA   ATL   N668DN DL
## 6  2013     1     1     5 EWR   ORD   N39463 UA
## 7  2013     1     1     6 EWR   FLL   N516JB B6
## 8  2013     1     1     6 LGA   IAD   N829AS EV
## 9  2013     1     1     6 JFK   MCO   N593JB B6
## 10 2013     1     1     6 LGA   ORD   N3ALAA AA
## # ... with 336,766 more rows
```

(Remember, when you're in RStudio, you can also use View() to avoid this problem.)

- flights2 에 left_join()을 이용하여 airlines 의 정보를 추가.

```
flights2 %>%
  select(-origin, -dest) %>%
  left_join(airlines, by = "carrier")
```



```
## # A tibble: 336,776 x 7
##   year month   day hour tailnum carrier name
##   <int> <int> <int> <dbl> <chr>   <chr>   <chr>
## 1  2013     1     1     5 N14228   UA      United Air Lines Inc.
## 2  2013     1     1     5 N24211   UA      United Air Lines Inc.
## 3  2013     1     1     5 N619AA   AA      American Airlines Inc.
## 4  2013     1     1     5 N804JB   B6      JetBlue Airways
## 5  2013     1     1     6 N668DN   DL      Delta Air Lines Inc.
## 6  2013     1     1     5 N39463   UA      United Air Lines Inc.
## 7  2013     1     1     6 N516JB   B6      JetBlue Airways
## 8  2013     1     1     6 N829AS   EV      ExpressJet Airlines Inc.
## 9  2013     1     1     6 N593JB   B6      JetBlue Airways
## 10 2013     1     1     6 N3ALAA   AA      American Airlines Inc.
## # ... with 336,766 more rows
```

- 같은 작업을 mutate()를 이용해서도 할 수 있으니 훨씬 복잡함.

```
flights2 %>%
  select(-origin, -dest) %>%
  mutate(name = airlines$name[match(carrier, airlines$carrier)])

## # A tibble: 336,776 x 7
##   year month   day hour tailnum carrier name
##   <int> <int> <int> <dbl> <chr>   <chr>   <chr>
## 1  2013     1     1     5 N14228   UA      United Air Lines Inc.
## 2  2013     1     1     5 N24211   UA      United Air Lines Inc.
## 3  2013     1     1     5 N619AA   AA      American Airlines Inc.
## 4  2013     1     1     5 N804JB   B6      JetBlue Airways
## 5  2013     1     1     6 N668DN   DL      Delta Air Lines Inc.
## 6  2013     1     1     5 N39463   UA      United Air Lines Inc.
## 7  2013     1     1     6 N516JB   B6      JetBlue Airways
## 8  2013     1     1     6 N829AS   EV      ExpressJet Airlines Inc.
## 9  2013     1     1     6 N593JB   B6      JetBlue Airways
## 10 2013     1     1     6 N3ALAA   AA      American Airlines Inc.
## # ... with 336,766 more rows
```

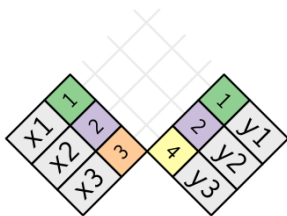
join 이해하기

| x | | y | |
|---|----|---|----|
| 1 | x1 | 1 | y1 |
| 2 | x2 | 2 | y2 |
| 3 | x3 | 4 | y3 |

```
x <- tribble(
  ~key, ~val_x,
  1, "x1",
  2, "x2",
  3, "x3"
)
```

```
y <- tribble(
  ~key, ~val_x,
    1, "x1",
    2, "x2",
    4, "x3"
)
```

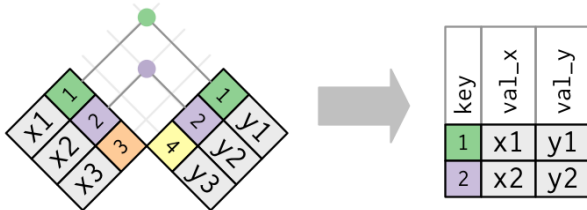
- 색으로 표현된 부분이 같은 key 를 나타냄.
- join 은 아래의 그림에서 선으로 나타난 연결 중 필요한 부분을 연결하는 것.
- 즉, join 을 key 를 기준으로 연결하는 것. 옆의 값(val_x, val_y)들은 따라서 간다.



- join 은 위의 그림에서 회색선이 만나는 점에서 일어남.
- 점의 갯수 = match 의 갯수 = 결과 자료의 관측수

Inner join

- join 의 가장 단순한 형태. key 가 같은 pair 찾기



- 정확하게 말하자면 equality operator 를 이용하는 equijoin 임.
- inner join 의 결과는 key, val_x, val_y 의 세 변수를 갖는다.
- by 를 이용하여 key 지정

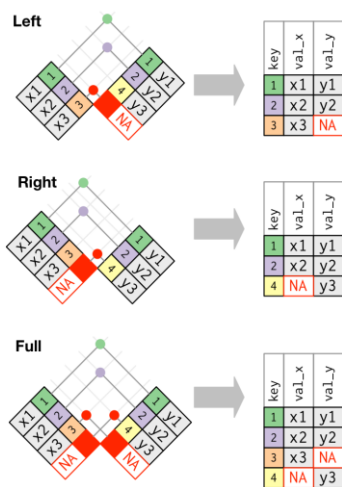
```
x %>%
  inner_join(y, by = "key")

## # A tibble: 2 x 3
##   key val_x val_y
##   <dbl> <chr> <chr>
## 1     1  x1    y1
## 2     2  x2    y2
```

- inner join에서는 키가 맞지 않는 자료는 모두 제외됨

Outer joins

- 모든 observation 을 유지.
- left join: 첫번째 table 의 모든 observation 유지
 - right join: 두번째 table 에 있는 모든 observation 유지
 - full join: 두 table 의 모든 observation 유지
- 아래의 그림 참고



```
x %>%
  left_join(y, by = "key")

## # A tibble: 3 x 3
##   key val_x val_y
##   <dbl> <chr> <chr>
## 1     1 x1    y1
## 2     2 x2    y2
## 3     3 x3    <NA>

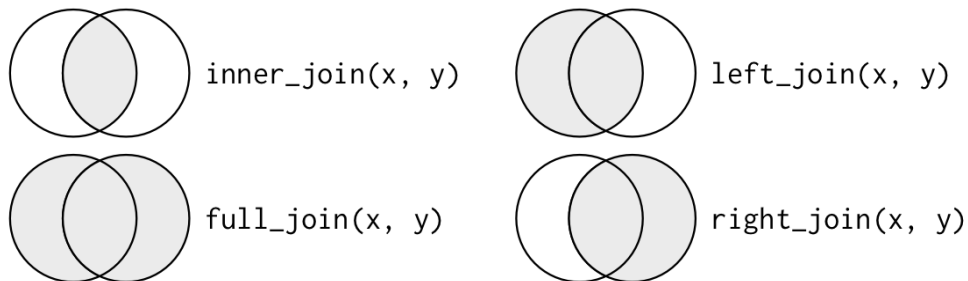
x %>%
  right_join(y, by = "key")

## # A tibble: 3 x 3
##   key val_x val_y
##   <dbl> <chr> <chr>
## 1     1 x1    y1
## 2     2 x2    y2
## 3     4 <NA> y3

x %>%
  full_join(y, by = "key")
```

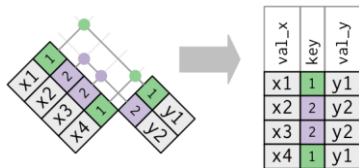
```
## # A tibble: 4 x 3
##   key val_x val_y
##   <dbl> <chr> <chr>
## 1     1 x1    y1
## 2     2 x2    y2
## 3     3 x3    <NA>
## 4     4 <NA> y3
```

- 가장 많이 쓰이는 형태의 join.
- 다른 table 로 부터 자료를 추가할 대에 많이 이용
- left_join 을 default join 으로 사용해야함
- 벤다이어그램을 이용한 join 그림



Duplicate keys

- key 가 unique 하지 않은 경우
- 한 table 에 duplicated key 를 가진 경우. 1:다 대응의 관계



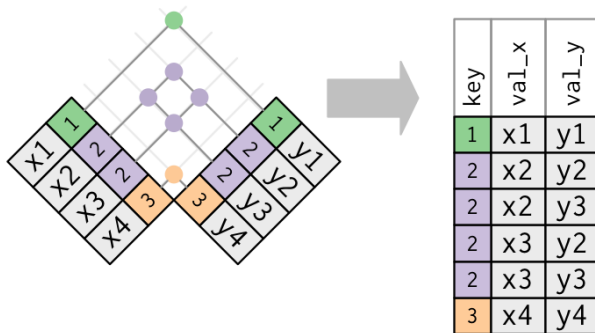
- y 에서는 primary key, x 에서는 foreign key 로 쓰임

```
x <- tribble(
  ~key, ~val_x,
  1, "x1",
  2, "x2",
  2, "x3",
  1, "x4"
)
y <- tribble(
  ~key, ~val_y,
  1, "y1",
  2, "y2"
```

```
)
left_join(x, y, by = "key")

## # A tibble: 4 x 3
##   key val_x val_y
##   <dbl> <chr> <chr>
## 1     1  x1    y1
## 2     2  x2    y2
## 3     2  x3    y2
## 4     1  x4    y1
```

2. 두 table 모두 duplicated key 를 갖는 경우. 일반적으로는 error 처리. 여기에서는 모든 가능한 조합을 모두 표시.



```
x <- tribble(
  ~key, ~val_x,
  1, "x1",
  2, "x2",
  2, "x3",
  3, "x4"
)
y <- tribble(
  ~key, ~val_y,
  1, "y1",
  2, "y2",
  2, "y3",
  3, "y4"
)
left_join(x, y, by = "key")

## # A tibble: 6 x 3
##   key val_x val_y
##   <dbl> <chr> <chr>
## 1     1  x1    y1
## 2     2  x2    y2
## 3     2  x2    y3
## 4     2  x3    y2
## 5     2  x3    y3
## 6     3  x4    y4
```

```
## 5      2 x3      y3
## 6      3 x4      y4
```

key 를 정의하기

- by="key"에 의해 key 를 정의.
- 기본값은 NULL 로 양쪽 table 에 나타나는 모든 변수를 key 로 이용. (natrual join)

```
flights2 %>%
  left_join(weather)

## Joining, by = c("year", "month", "day", "hour", "origin")

## # A tibble: 336,776 x 18
##   year month   day hour origin dest tailnum carrier temp dewp humid
##   <dbl> <dbl> <int> <dbl> <chr> <chr> <chr>   <chr>   <dbl> <dbl> <dbl>
## 1  2013     1     1     5 EWR   IAH   N14228 UA      39.0  28.0  64.4
## 2  2013     1     1     5 LGA   IAH   N24211 UA      39.9  25.0  54.8
## 3  2013     1     1     5 JFK   MIA   N619AA AA      39.0  27.0  61.6
## 4  2013     1     1     5 JFK   BQN   N804JB B6      39.0  27.0  61.6
## 5  2013     1     1     6 LGA   ATL   N668DN DL      39.9  25.0  54.8
## 6  2013     1     1     5 EWR   ORD   N39463 UA      39.0  28.0  64.4
## 7  2013     1     1     6 EWR   FLL   N516JB B6      37.9  28.0  67.2
## 8  2013     1     1     6 LGA   IAD   N829AS EV      39.9  25.0  54.8
## 9  2013     1     1     6 JFK   MCO   N593JB B6      37.9  27.0  64.3
## 10 2013     1     1     6 LGA   ORD   N3ALAA AA      39.9  25.0  54.8
## # ... with 336,766 more rows, and 7 more variables: wind_dir <dbl>,
## #   wind_speed <dbl>, wind_gust <dbl>, precip <dbl>, pressure <dbl>,
## #   visib <dbl>, time_hour <dtm>
```

- flights 와 planes 모두 year 변수를 가지고 있으나 key 로는 tailnum 만 이용.
- 이 경우 year.x 는 flights 로 부터의 year, year.y 는 planes 로 부터의 year 를 나타냄.

```
flights2 %>%
  left_join(planes, by = "tailnum")

## # A tibble: 336,776 x 16
##   year.x month   day hour origin dest tailnum carrier year.y type
##   <int> <int> <int> <dbl> <chr> <chr> <chr>   <chr>   <int> <chr>
## 1  2013     1     1     5 EWR   IAH   N14228 UA      1999 Fixe~
## 2  2013     1     1     5 LGA   IAH   N24211 UA      1998 Fixe~
## 3  2013     1     1     5 JFK   MIA   N619AA AA      1990 Fixe~
## 4  2013     1     1     5 JFK   BQN   N804JB B6      2012 Fixe~
## 5  2013     1     1     6 LGA   ATL   N668DN DL      1991 Fixe~
## 6  2013     1     1     5 EWR   ORD   N39463 UA      2012 Fixe~
## 7  2013     1     1     6 EWR   FLL   N516JB B6      2000 Fixe~
## 8  2013     1     1     6 LGA   IAD   N829AS EV      1998 Fixe~
## 9  2013     1     1     6 JFK   MCO   N593JB B6      2004 Fixe~
## 10 2013     1     1     6 LGA   ORD   N3ALAA AA          NA <NA>
```

```
## # ... with 336,766 more rows, and 6 more variables: manufacturer <chr>,
## #   model <chr>, engines <int>, seats <int>, speed <int>, engine <chr>
```

- 양쪽 table 에서 변수의 이름이 다른 경우 `by = c("a" = "b")`와 같이 사용
- flights 의 dest, origin 은 airports 의 faa code 임

```
flights2 %>%
  left_join(airports, c("dest" = "faa"))

## # A tibble: 336,776 x 15
##   year month   day hour origin dest tailnum carrier name lat lon
##   <int> <int> <int> <dbl> <chr> <chr> <chr> <chr> <chr> <dbl> <dbl>
## 1  2013     1     1     5 EWR   IAH  N14228 UA      Geor~  30.0 -95.3
## 2  2013     1     1     5 LGA   IAH  N24211 UA      Geor~  30.0 -95.3
## 3  2013     1     1     5 JFK   MIA  N619AA AA      Miam~  25.8 -80.3
## 4  2013     1     1     5 JFK   BQN  N804JB B6      <NA>   NA    NA
## 5  2013     1     1     6 LGA   ATL  N668DN DL      Hart~  33.6 -84.4
## 6  2013     1     1     5 EWR   ORD  N39463 UA      Chic~  42.0 -87.9
## 7  2013     1     1     6 EWR   FLL  N516JB B6      Fort~  26.1 -80.2
## 8  2013     1     1     6 LGA   IAD  N829AS EV      Wash~  38.9 -77.5
## 9  2013     1     1     6 JFK   MCO  N593JB B6      Orla~  28.4 -81.3
## 10 2013     1     1     6 LGA   ORD  N3ALAA AA      Chic~  42.0 -87.9
## # ... with 336,766 more rows, and 4 more variables: alt <int>, tz <dbl>,
## #   dst <chr>, tzone <chr>
```

```
flights2 %>%
  left_join(airports, c("origin" = "faa"))

## # A tibble: 336,776 x 15
##   year month   day hour origin dest tailnum carrier name lat lon
##   <int> <int> <int> <dbl> <chr> <chr> <chr> <chr> <chr> <dbl> <dbl>
## 1  2013     1     1     5 EWR   IAH  N14228 UA      Newa~  40.7 -74.2
## 2  2013     1     1     5 LGA   IAH  N24211 UA      La G~  40.8 -73.9
## 3  2013     1     1     5 JFK   MIA  N619AA AA      John~  40.6 -73.8
## 4  2013     1     1     5 JFK   BQN  N804JB B6      John~  40.6 -73.8
## 5  2013     1     1     6 LGA   ATL  N668DN DL      La G~  40.8 -73.9
## 6  2013     1     1     5 EWR   ORD  N39463 UA      Newa~  40.7 -74.2
## 7  2013     1     1     6 EWR   FLL  N516JB B6      Newa~  40.7 -74.2
## 8  2013     1     1     6 LGA   IAD  N829AS EV      La G~  40.8 -73.9
## 9  2013     1     1     6 JFK   MCO  N593JB B6      John~  40.6 -73.8
## 10 2013     1     1     6 LGA   ORD  N3ALAA AA      La G~  40.8 -73.9
## # ... with 336,766 more rows, and 4 more variables: alt <int>, tz <dbl>,
## #   dst <chr>, tzone <chr>
```

Other implementations

- dplyr 과 merge 의 관계

| dplyr | merge |
|-------------------------------|--------------------------|
| <code>inner_join(x, y)</code> | <code>merge(x, y)</code> |

```

left_join(x, y)  merge(x, y, all.x = TRUE)
right_join(x, y) merge(x, y, all.y = TRUE),
full_join(x, y)  merge(x, y, all.x = TRUE, all.y = TRUE)

```

- dplyr 과 SQL 의 관계

| dplyr | SQL |
|----------------------------|--|
| inner_join(x, y, by = "z") | SELECT * FROM x INNER JOIN y USING (z) |
| left_join(x, y, by = "z") | SELECT * FROM x LEFT OUTER JOIN y USING (z) |
| right_join(x, y, by = "z") | SELECT * FROM x RIGHT OUTER JOIN y USING (z) |
| full_join(x, y, by = "z") | SELECT * FROM x FULL OUTER JOIN y USING (z) |

Filtering joins

- semi_join(x, y): y 와 맞는 x 에 있는 모든 자료를 유지.
- anti_join(x, y): y 와 맞는 x 에 있는 모든 자료를 삭제.

```

top_dest <- flights %>%
  count(dest, sort = TRUE) %>%
  head(10)
top_dest

## # A tibble: 10 x 2
##   dest      n
##   <chr> <int>
## 1 ORD    17283
## 2 ATL    17215
## 3 LAX    16174
## 4 BOS    15508
## 5 MCO    14082
## 6 CLT    14064
## 7 SFO    13331
## 8 FLL    12055
## 9 MIA    11728
## 10 DCA     9705

```

- 특정 dest 에 해당되는 flights 만 선택

```

flights %>%
  filter(dest %in% top_dest$dest)

## # A tibble: 141,145 x 19
##   year month   day dep_time sched_dep_time dep_delay arr_time
##   <int> <int> <int>   <int>         <int>         <dbl>   <int>
## 1  2013     1     1     542             540           2     923
## 2  2013     1     1     554             600          -6     812
## 3  2013     1     1     554             558          -4     740

```



```
## 4 2013 1 1 555 600 -5 913
## 5 2013 1 1 557 600 -3 838
## 6 2013 1 1 558 600 -2 753
## 7 2013 1 1 558 600 -2 924
## 8 2013 1 1 558 600 -2 923
## 9 2013 1 1 559 559 0 702
## 10 2013 1 1 600 600 0 851
## # ... with 141,135 more rows, and 12 more variables: sched_arr_time <int>,
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dtm>
```

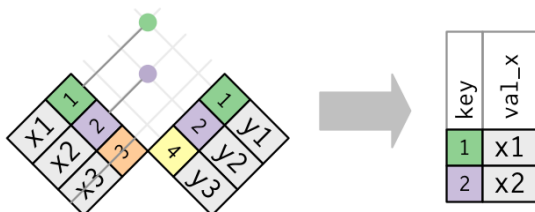
- semi_join 을 이용

```
flights %>%
  semi_join(top_dest)

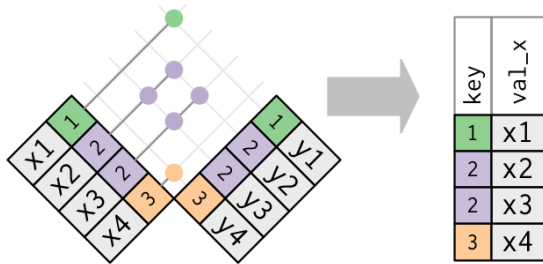
## Joining, by = "dest"

## # A tibble: 141,145 x 19
##   year month   day dep_time sched_dep_time dep_delay arr_time
##   <int> <int> <int>   <int>         <int>         <dbl>   <int>
## 1 2013     1     1     542           540           2     923
## 2 2013     1     1     554           600          -6     812
## 3 2013     1     1     554           558          -4     740
## 4 2013     1     1     555           600          -5     913
## 5 2013     1     1     557           600          -3     838
## 6 2013     1     1     558           600          -2     753
## 7 2013     1     1     558           600          -2     924
## 8 2013     1     1     558           600          -2     923
## 9 2013     1     1     559           559           0     702
## 10 2013     1     1     600           600           0     851
## # ... with 141,135 more rows, and 12 more variables: sched_arr_time <int>,
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dtm>
```

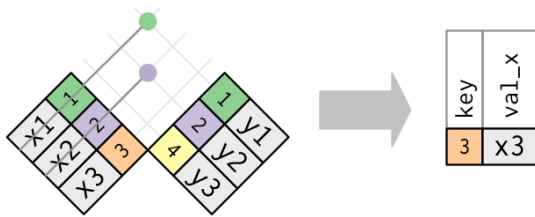
- semi_join 이해하기



- semi_join 에서 중요한 것은 match 의 존재 여부 뿐, 어떤 관측과 match 되었는가는 중요하지 않음



- anti_join : semi_join 의 반대



- mismatch 를 찾는데 중요
- plane 정보가 없는 flights 찾기

```
flights %>%
  anti_join(planes, by = "tailnum") %>%
  count(tailnum, sort = TRUE)
```

```
## # A tibble: 722 x 2
##   tailnum      n
##   <chr>    <int>
## 1 <NA>     2512
## 2 N725MQ     575
## 3 N722MQ     513
## 4 N723MQ     507
## 5 N713MQ     483
## 6 N735MQ     396
## 7 N0EGMQ     371
## 8 N534MQ     364
## 9 N542MQ     363
## 10 N531MQ    349
## # ... with 712 more rows
```

Join problems

- join 을 사용하기 전에 지켜야할 사항들
1. 먼저 각 table 에서 primary key 가 되는 변수들을 지정할 것. 이는 자료에 대한 이해를 바탕으로 해야하며 변수들의 조합을 살펴보는 등의 경험적 이해로 진행해서는 안된다. 의미를 생각하지 않고 변수들을 본다면 운이 좋게 혹은 운이 나쁘게도 일반적이지

않은, 현재 자료에서만 나타나는 unique 한 조합을 발견하게 될 수도 있다.

예를 들어 각 공항마다 위도, 경도는 unique 하지만 좋은 identifier 는 아니다.

```
airports %>% count(alt, lon) %>% filter(n > 1)

## # A tibble: 0 x 3
## # ... with 3 variables: alt <int>, lon <dbl>, n <int>
```

2. primary key 값이 missing 이 없는지 확인해야한다. 만약에 missing 값이 없으면 그 값에 대한 identify 는 불가능해진다.
3. foreign key 와 primary key 가 잘 맞는지 확인. 이를 확인하기 위해서는 anti_join 함수를 이용하면 됨. missing key 가 있는 경우 inner, outer join 이용에 주의해야한다. 맞지 않는 자료를 제외하는 것을 원하는지 아닌지를 생각해야함.
- join 이 제대로 되었는지를 확인하기 위해 join 하기 전, 후의 자료 수를 확인하는 것만으로는 불충분하다는 것을 명심해야한다. duplicate key 를 이용한 inner join 인 경우 운나쁘게도 drop 한 row 의 숫자와 duplicated 된 row 의 수가 우연히 같을수도 있다!

Set operations

- 두 개의 table 에 대하여 사용.
- 일반적으로 자주사용되지는 않지만 하나의 복잡한 filter 를 좀 더 단순한 여러 조각으로 나누고 싶은 경우 종종 사용.
- 모든 operator 들은 모든 변수에 대한 값과 비교하여 complete row 에 대하여 수행.
- 이는 두 table 이 같은 변수를 갖고 관측에 대해 다음과 같이 처리.
- intersect(x, y): x, y table 양쪽 모두에 있는 observation 만
- union(x, y): x, y table 모두에 있는 모든 observation
- setdiff(x, y): x 에만 있고 y 에는 없는 observation

```
df1 <- tribble(
  ~x, ~y,
  1, 1,
  2, 1
)
df2 <- tribble(
  ~x, ~y,
```

```

  1, 1,
  1, 2
)

intersect(df1, df2)

## # A tibble: 1 x 2
##       x     y
##   <dbl> <dbl>
## 1     1     1

# Note that we get 3 rows, not 4
union(df1, df2)

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

setdiff(df1, df2)

## # A tibble: 1 x 2
##       x     y
##   <dbl> <dbl>
## 1     2     1

setdiff(df2, df1)

## # A tibble: 1 x 2
##       x     y
##   <dbl> <dbl>
## 1     1     2

```

11. Strings with stringr

Introduction

- string 을 다루는 방법 소개
- regular expression (regexp) 소개

Prerequisites

- stringr 패키지가 필요

```
library(tidyverse)
library(stringr)
```

String basics

- string 을 만들기 위해서는 작은 따옴표나 큰 따옴표를 사용
- 다른 언어들과는 달리 두 따옴표의 차이는 없음.
- " 사용을 권고.

```
string1 <- "This is a string"
string2 <- 'If I want to include a "quote" inside a string, I use single quotes'
```

- string 내에 " 를 쓰고 싶은 경우 함께 써야 함. (특수문자는 모두 함께)

```
double_quote <- "\"\" # or '\"'
single_quote <- '\'' # or '\"'
```

- cat 이나 writeLines 를 이용하여 확인

```
cat(double_quote)
```

```
## "
```

```
x <- c("\"", "\\")
```

```
x
```

```
## [1] "\"" "\\\""
```

```
writeLines(x)
```

```
## "
```

```
## \
```

- 유용한 특수 문자들: "\n", "\t", ":?'\"', or ?'\"'

```
x <- "\u00b5"
```

```
x
```

```
## [1] "μ"
```

String length

- stringr 의 string 을 다루는 함수는 모두 str_로 시작
- str_length: string 길이

```
str_length(c("a", "R for data science", NA))
```

```
## [1]  1 18 NA
```

- Rstudio 의 autocomplete 기느

Combining strings

- str_c(): 두 개 이상의 string 을 결합

```
str_c("x", "y")
```

```
## [1] "xy"
```

```
str_c("x", "y", "z")
```

```
## [1] "xyz"
```

- sep 옵션을 이용하여 결합할 때 연결부분의 문자들을 정의할 수 있음.

```
str_c("x", "y", sep = ", ")
```

```
## [1] "x, y"
```

- str_replace_na() : 결측치가 있는경우 사용

```
x <- c("abc", NA)
```

```
str_c("|-", x, "-|")
```

```
## [1] "|-abc-|" NA
```

```
str_c("|-", str_replace_na(x), "-|")
```

```
## [1] "|-abc-|" "|-NA-|"
```

- str_c()에서 길이가 다른 vector 를 사용하는 경우 recycle 원칙을 적용.

```
str_c("prefix-", c("a", "b", "c"), "-suffix")
```

```
## [1] "prefix-a-suffix" "prefix-b-suffix" "prefix-c-suffix"
```

- 길이가 0 인 object 의 경우 제외시킴.(이는 if 문을 함께 이용할 때 매우 유용)

```
name <- "Hadley"
```

```
time_of_day <- "morning"
```

```
birthday <- FALSE
```

```
str_c(
  "Good ", time_of_day, " ", name,
  if (birthday) " and HAPPY BIRTHDAY",
  "."
)
## [1] "Good morning Hadley."
```

- collapse 옵션: 하나의 벡터 내에 정의되어 있는 string 들을 하나의 string 으로 결합하고자 할 때에 이용.

```
str_c(c("x", "y", "z"), collapse = ", ")
## [1] "x, y, z"
```

Subsetting strings

- str_sub(): string 의 일부를 발췌.
- start 와 end 인자를 이용.
- 음수를 이용할 경우 string 의 뒤에서부터의 위치를 의미.

```
x <- c("Apple", "Banana", "Pear")
str_sub(x, 1, 3)
## [1] "App" "Ban" "Pea"

# negative numbers count backwards from end
str_sub(x, -3, -1)
## [1] "ple" "ana" "ear"
```

- start 와 end 에 지정된 값이 string 의 길이를 넘어가는 경우에는 start 위치로부터 값이 있는 만큼을 보여줌.

```
str_sub("a", 1, 5)
## [1] "a"

str_sub("a", 4, 5)
## [1] ""

str_sub("a", -3, -1)
## [1] "a"

str_sub("a", -3, -2)
## [1] ""
```

- `str_sub()`를 활용하여 string 의 일부를 수정.

```
str_sub(x, 1, 1) <- str_to_lower(str_sub(x, 1, 1))
x
## [1] "apple" "banana" "pear"
```

Locales

- `str_to_lower()`, `str_to_upper()`, `str_to_title()`: locale 과 함께 이용하여 각 나라의 언어에 맞게 바꿀 수 있음.

*# Turkish has two i's: with and without a dot, and it
has a different rule for capitalising them:*

```
str_to_upper(c("i", "ı"))
## [1] "I" "I"

str_to_upper(c("i", "ı"), locale = "tr")
## [1] "<U+0130>" "I"
```

- ISO 639-1 locale codes : https://en.wikipedia.org/wiki/List_of_ISO_639-1_codes 참고
- `order()`, `sort()`: 현재의 locale 에 대하여 정렬
- `str_order()`, `str_sort()`: locale 지정 가능

```
x <- c("apple", "eggplant", "banana")

str_sort(x, locale = "en") # English
## [1] "apple" "banana" "eggplant"

str_sort(x, locale = "haw") # Hawaiian
## [1] "apple" "eggplant" "banana"

str_sort(LETTERS, locale = "haw") # Hawaiian
## [1] "A" "E" "I" "O" "U" "B" "C" "D" "F" "G" "H" "J" "K" "L" "M" "N" "P"
## [18] "Q" "R" "S" "T" "V" "W" "X" "Y" "Z"
```

Matching patterns with regular expressions

- `regexp`: 매우 간결한 언어로 string 에 대한 pattern 을 기술할 수 있게 해줌.
- `str_view()`와 `str_view_all()`: 문자 벡터와 regular expression 이 어떻게 match 되는지를 보여줌.

Basic matches

- string 중 exact match 가 되는 부분 찾기.

```
x <- c("apple", "banana", "pear")
#str_view(x, "an")
```

- `::` 하나의 임의의 문자를 나타냄.

```
#str_view(x, ".a.")
```

- 마침표를 찾기 위해서는 `\.`을 이용.

```
# To create the regular expression, we need \\
dot <- "\\."
```

```
# But the expression itself only contains one:
writeLines(dot)
```

```
## \.
```

```
# And this tells R to look for an explicit .
#str_view(c("abc", "a.c", "bef"), "a\\.c")
```

- 찾기 위해서는 `\\`를 이용.

```
x <- "a\\b"
writeLines(x)
```

```
## a\b
```

```
#str_view(x, "\\|\\|")
```

Anchors

- string 의 첫 부분 부터 혹은 끝부분을 맞추기 위한 것.
- `^`: string 의 시작부분 맞추기
- `$`: string 의 끝부분 맞추기

```
x <- c("apple", "banana", "pear")
#str_view(x, "^a")
#str_view(x, "a$")
```

- string 전체가 똑같은 것만을 찾고 싶은 경우 string 앞뒤에 `^`와 `$`를 붙여주면 됨.

```
x <- c("apple pie", "apple", "apple cake")
#str_view(x, "apple")
#str_view(x, "^apple$")
```

- `\\b`: 단어 사이의 경계 You can also match the boundary between words with `.` I don't often use this in R, but I will sometimes use it when I'm doing a search in RStudio

when I want to find the name of a function that's a component of other functions. For example, I'll search for `\bsum\b` to avoid matching `summarise`, `summary`, `rowsum` and so on.

```
x <- c("summarise", "summary", "rowsum", "add sum add")
#str_view(x, "\\bsum\\b")
#str_view(x, "\\bsum")
#str_view(x, "sum\\b")
```

Character classes and alternatives

- `\d`: 숫자. regular expression 에서는 `\d`
- `\s`: space, tab, newline `\s`
- `[abc]`: matches a, b, or c. string 의 앞부터 비교하여 첫번째 나오는 것 찾기
- `[^abc]`: a, b, or c 를 제외한 나머지 찾기
- 하나 이상의 pattern 들 중 고르기 위해서는 `|`를 이용.

```
#str_view(c("grey", "gray"), "gr(e/a)y")
```

Repetition

- pattern match 의 반복이 있는 경우
- `?:` 0 or 1
- `+:` 1 or more
- `*:` 0 or more
- 이 연산자의 우선순위가 높기 때문에 `colou?r` 로 match 시키면 영국/미국식 spelling 을 모두 찾을 수 있음.

```
x <- "1888 is the longest year in Roman numerals: MDCCCLXXXVIII"
#str_view(x, "CC?")
#str_view(x, "CC+")
#str_view(x, 'C[ LX ]+')
```

- 반복 숫자 지정
- `{n}`: exactly n
- `{,m}`: at most m
- `{n,m}`: between n and m

```
#str_view(x, "C{2}")
#str_view(x, "C{2,}")
#str_view(x, "C{2,3}")
```

- 기본적으로는 가장 긴 string 을 찾음.
- `?`를 이용하여 가장 짧은 string 을 찾을수도 있음.

```
#str_view(x, 'C{2,3}?')
#str_view(x, 'C[LX]+?')
```

Grouping and backreferences

- `()`: 복잡한 표현의 모호함을 없애기 위하여 사용. `group` 을 정의하는데에 쓰일수도 있음.

```
#str_view(fruit, "(.)\\1", match = TRUE)
```

Detect matches

- `str_detect()`: character vector 가 정의한 pattern 을 가지고 있는지를 확인

```
x <- c("apple", "banana", "pear")
str_detect(x, "e")
```

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

```
# How many common words start with t?
sum(str_detect(words, "^t"))
```

```
## [1] 65
```

```
# What proportion of common words end with a vowel?
mean(str_detect(words, "[aeiou]$"))
```

```
## [1] 0.2765306
```

- 복잡한 logical condition 이 있는 경우 `str_detect()` 함수를 결합하여 사용

```
# Find all words containing at least one vowel, and negate
no_vowels_1 <- !str_detect(words, "[aeiou]")
# Find all words consisting only of consonants (non-vowels)
no_vowels_2 <- str_detect(words, "^[^aeiou]+$")
identical(no_vowels_1, no_vowels_2)
```

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

- regular expression 이 복잡해진다면 작은 piece 로 나누어 시도
- `str_detect()` 대신 `str_subset()` 을 이용할수도 있다.

```
words[str_detect(words, "x$")]
```

```
## [1] "box" "sex" "six" "tax"
```

```
str_subset(words, "x$")
```

```
## [1] "box" "sex" "six" "tax"
```

- 일반적으로 string 은 data frame 의 한 column 인 경우가 많으므로 다음과 같은 형태의 filter 를 사용하는 것이 더 좋다.

```
df <- tibble(
  word = words,
  i = seq_along(word)
)
df %>%
  filter(str_detect(words, "x$"))

## # A tibble: 4 x 2
##   word      i
##   <chr> <int>
## 1 box    108
## 2 sex    747
## 3 six    772
## 4 tax    841
```

- str_count(): str_detect()의 변형. string 내에 pattern 이 몇개나 있는지를 알려줌

```
x <- c("apple", "banana", "pear")
str_count(x, "a")

## [1] 1 3 1

# On average, how many vowels per word?
mean(str_count(words, "[aeiou]"))

## [1] 1.991837
```

- str_count()는 mutate()와 함께 쓰임

```
df %>%
  mutate(
    vowels = str_count(word, "[aeiou]"),
    consonants = str_count(word, "^[aeiou]")
  )

## # A tibble: 980 x 4
##   word      i vowels consonants
##   <chr> <int> <int>      <int>
## 1 a         1     1         0
## 2 able      2     2         2
## 3 about     3     3         2
## 4 absolute  4     4         4
## 5 accept    5     2         4
## 6 account   6     3         4
## 7 achieve   7     4         3
## 8 across    8     2         4
## 9 act       9     1         2
```

```
## 10 active      10      3      3
## # ... with 970 more rows
```

- matches 는 결코 overlap 되지 않는다!

```
str_count("abababa", "aba")
```

```
## [1] 2
```

```
#str_view_all("abababa", "aba")
```

Extract matches

- str_extract(): match 가 되는 text 뽑아내기

```
length(sentences)
```

```
## [1] 720
```

```
head(sentences)
```

```
## [1] "The birch canoe slid on the smooth planks."
## [2] "Glue the sheet to the dark blue background."
## [3] "It's easy to tell the depth of a well."
## [4] "These days a chicken leg is a rare dish."
## [5] "Rice is often served in round bowls."
## [6] "The juice of lemons makes fine punch."
```

- sentences 에 포함된 color 찾기

```
colours <- c("red", "orange", "yellow", "green", "blue", "purple")
colour_match <- str_c(colours, collapse = "|")
colour_match
```

```
## [1] "red|orange|yellow|green|blue|purple"
```

```
has_colour <- str_subset(sentences, colour_match)
matches <- str_extract(has_colour, colour_match)
head(matches)
```

```
## [1] "blue" "blue" "red"  "red"  "red"  "blue"
```

```
length(matches)
```

```
## [1] 57
```

```
table(matches)
```

```
## matches
##   blue green orange purple   red yellow
##     8     8     1     1    37     2
```

- str_extract()는 문장에서 처음 만나는 색의 이름만을 뽑아냄.

- 문장에 2 개 이상의 색이 나타나는 경우 아래와 같이 추출

```
more <- sentences[str_count(sentences, colour_match) > 1]
#str_view_all(more, colour_match)
more

## [1] "It is hard to erase blue or red ink."
## [2] "The green light in the brown box flickered."
## [3] "The sky in the west is tinged with orange red."

str_extract(more, colour_match)

## [1] "blue" "green" "orange"
```

- str_extract_all(): 모든 match 찾기

```
str_extract_all(more, colour_match)

## [[1]]
## [1] "blue" "red"
##
## [[2]]
## [1] "green" "red"
##
## [[3]]
## [1] "orange" "red"
```

- simplify = TRUE 옵션: matrix 형태로 결과를 나타냄.

```
str_extract_all(more, colour_match, simplify = TRUE)

##      [,1]      [,2]
## [1,] "blue"   "red"
## [2,] "green"  "red"
## [3,] "orange" "red"

x <- c("a", "a b", "a b c")
str_extract_all(x, "[a-z]", simplify = TRUE)

##      [,1] [,2] [,3]
## [1,] "a"  ""   ""
## [2,] "a"  "b"  ""
## [3,] "a"  "b"  "c"
```

Grouped matches

- 문장에서 명사 뽑아내기
- a 나 the 뒤의 단어는 대부분 명사이므로 이를 이용해보자

```
noun <- "(a|the) ([^ ]+)"
has_noun <- sentences %>%
```

```

str_subset(noun) %>%
head(10)
has_noun %>%
  str_extract(noun)

## [1] "the smooth" "the sheet" "the depth" "a chicken" "the parked"
## [6] "the sun"     "the huge"   "the ball"  "the woman" "a helps"

```

- str_extract(): complete match 찾기
- str_match(): individual component 를 matrix 형태로 찾기

```

has_noun %>%
  str_match(noun)

##      [,1]      [,2] [,3]
## [1,] "the smooth" "the" "smooth"
## [2,] "the sheet"  "the" "sheet"
## [3,] "the depth"  "the" "depth"
## [4,] "a chicken"  "a"   "chicken"
## [5,] "the parked" "the" "parked"
## [6,] "the sun"     "the" "sun"
## [7,] "the huge"    "the" "huge"
## [8,] "the ball"    "the" "ball"
## [9,] "the woman"   "the" "woman"
## [10,] "a helps"    "a"   "helps"

```

- tidyr::extract():tidy data 인 경우 이용

```

tibble(sentence = sentences) %>%
  tidyr::extract(
    sentence, c("article", "noun"), "(a|the) ([^ ]+)",
    remove = FALSE
  )

## # A tibble: 720 x 3
##   sentence                                article noun
##   <chr>                                <chr>  <chr>
## 1 The birch canoe slid on the smooth planks. the    smooth
## 2 Glue the sheet to the dark blue background. the    sheet
## 3 It's easy to tell the depth of a well.    the    depth
## 4 These days a chicken leg is a rare dish.  a      chicken
## 5 Rice is often served in round bowls.      <NA>   <NA>
## 6 The juice of lemons makes fine punch.     <NA>   <NA>
## 7 The box was thrown beside the parked truck. the    parked
## 8 The hogs were fed chopped corn and garbage. <NA>   <NA>
## 9 Four hours of steady work faced us.       <NA>   <NA>
## 10 Large size in stockings is hard to sell.  <NA>   <NA>
## # ... with 710 more rows

```

Replacing matches

- `str_replace()`, `str_replace_all()`: match 되는 string 바꾸기

```
x <- c("apple", "pear", "banana")
str_replace(x, "[aeiou]", "-")

## [1] "-pple" "p-ar" "b-nana"

str_replace_all(x, "[aeiou]", "-")

## [1] "-ppl-" "p--r" "b-n-n-"
```

- `str_replace_all()`에서는 multiple replacement 가능

```
x <- c("1 house", "2 cars", "3 people")
str_replace_all(x, c("1" = "one", "2" = "two", "3" = "three"))

## [1] "one house" "two cars" "three people"
```

- 단어 순서바꾸기도 가능

```
sentences[1:5]

## [1] "The birch canoe slid on the smooth planks."
## [2] "Glue the sheet to the dark blue background."
## [3] "It's easy to tell the depth of a well."
## [4] "These days a chicken leg is a rare dish."
## [5] "Rice is often served in round bowls."

sentences %>%
  str_replace("([ ^ ]+) ([ ^ ]+) ([ ^ ]+)", "\\1 \\3 \\2") %>%
  head(5)

## [1] "The canoe birch slid on the smooth planks."
## [2] "Glue sheet the to the dark blue background."
## [3] "It's to easy tell the depth of a well."
## [4] "These a days chicken leg is a rare dish."
## [5] "Rice often is served in round bowls."
```

Splitting

- `str_split()`: string 을 여러조각으로 분할. 결과는 list 형태로 저장

```
sentences %>%
  head(5) %>%
  str_split(" ")

## [[1]]
## [1] "The" "birch" "canoe" "slid" "on" "the" "smooth"
## [8] "planks."
##
## [[2]]
```



```
## [1] "Glue"      "the"      "sheet"    "to"      "the"
## [6] "dark"      "blue"     "background."
##
## [[3]]
## [1] "It's" "easy" "to" "tell" "the" "depth" "of" "a" "well."
##
## [[4]]
## [1] "These" "days" "a" "chicken" "leg" "is" "a"
## [8] "rare" "dish."
##
## [[5]]
## [1] "Rice" "is" "often" "served" "in" "round" "bowls."

"a|b|c|d" %>%
  str_split("\\|") %>%
  .[[1]]

## [1] "a" "b" "c" "d"
```

- simplify=TRUE 옵션을 이용하면 결과가 matrix 형태로 저장

```
sentences %>%
  head(5) %>%
  str_split(" ", simplify = TRUE)

##      [,1] [,2] [,3] [,4] [,5] [,6] [,7]
## [1,] "The" "birch" "canoe" "slid" "on" "the" "smooth"
## [2,] "Glue" "the" "sheet" "to" "the" "dark" "blue"
## [3,] "It's" "easy" "to" "tell" "the" "depth" "of"
## [4,] "These" "days" "a" "chicken" "leg" "is" "a"
## [5,] "Rice" "is" "often" "served" "in" "round" "bowls."
##      [,8] [,9]
## [1,] "planks." ""
## [2,] "background." ""
## [3,] "a" "well."
## [4,] "rare" "dish."
## [5,] "" ""
```

- n=: 가능한 piece 의 최대값을 지정.

```
fields <- c("Name: Hadley", "Country: NZ", "Age: 35 :11")
fields %>% str_split(":", n = 2, simplify = TRUE)

##      [,1] [,2]
## [1,] "Name" "Hadley"
## [2,] "Country" "NZ"
## [3,] "Age" "35 :11"
```

- boundary 함수를 이용하면 character, line, sentence, word 별로 나눌 수 있다.

```
x <- "This is a sentence. This is another sentence."
#str_view_all(x, boundary("word"))

str_split(x, " ")[[1]]

## [1] "This"      "is"      "a"      "sentence." ""      "This"

## [7] "is"      "another" "sentence."

str_split(x, boundary("word"))[[1]]

## [1] "This"      "is"      "a"      "sentence" "This"      "is"
## [7] "another"   "sentence"
```

Find matches

- `str_locate()`, `str_locate_all()`: 각 match 의 시작과 끝 위치를 지정

```
fruit <- c("apple", "banana", "pear", "pineapple")
str_locate(fruit, "$")
```

```
##      start end
## [1,]      6  5
## [2,]      7  6
## [3,]      5  4
## [4,]     10  9
```

```
str_locate(fruit, "a")
```

```
##      start end
## [1,]      1  1
## [2,]      2  2
## [3,]      3  3
## [4,]      5  5
```

```
str_locate_all(fruit, "a") # list 형태
```

```
## [[1]]
##      start end
## [1,]      1  1
##
## [[2]]
##      start end
## [1,]      2  2
## [2,]      4  4
## [3,]      6  6
##
## [[3]]
##      start end
## [1,]      3  3
##
```

```
## [[4]]
##      start end
## [1,]      5  5
```

Other types of pattern

- `regex()`를 이용하면 여러가지 option 사용 가능

```
# The regular call:
#str_view(fruit, "nana")
# Is shorthand for
#str_view(fruit, regex("nana"))
str_extract_all("The Cat in the Hat", "[a-z]+")
str_extract_all("The Cat in the Hat", regex("[a-z]+"))
```

- `ignore_case = TRUE`: 대/소문자 구별 없이 match 를 찾기.

```
str_extract_all("The Cat in the Hat", regex("[a-z]+", ignore_case = TRUE))

## [[1]]
## [1] "The" "Cat" "in"  "the" "Hat"
```

- `multiline = TRUE`: 여러줄로 된 경우 각 줄에서 찾기

```
x <- "Line 1\nLine 2\nLine 3"
str_extract_all(x, "^Line")[[1]]

## [1] "Line"

str_extract_all(x, regex("^Line", multiline = TRUE))[[1]]

## [1] "Line" "Line" "Line"
```

- `comments = TRUE`: #으로 시작하는 문장을 comment 로 처리

```
phone <- regex("
  \\(?      # optional opening parens
  (\\d{3})  # area code
  [- ]?    # optional closing parens, dash, or space
  (\\d{3})  # another three numbers
  [- ]?    # optional space or dash
  (\\d{4})  # three more numbers
", comments = TRUE)

str_match("514-791-8141", phone)

##      [,1]      [,2] [,3] [,4]
## [1,] "514-791-8141" "514" "791" "8141"
```

- `fixed()`: `regex()`보다 빠르다.

```
microbenchmark::microbenchmark(
  fixed = str_detect(sentences, fixed("the")),
```

```

    regex = str_detect(sentences, "the"),
    times = 20
)

## Unit: microseconds
##   expr      min       lq     mean   median      uq     max neval
## fixed  77.669  81.2465 106.5523  84.6950  92.232 411.848    20
## regex 214.867 216.0165 253.6243 218.9545 232.623 502.036    20

```

- fixed(), coll()


```

a1 <- "\u00e1"
a2 <- "a\u0301"
c(a1, a2)

## [1] "a" "a<U+0301>"

a1 == a2

## [1] FALSE

str_detect(a1, fixed(a2))

## [1] FALSE

str_detect(a1, coll(a2))

## [1] TRUE

```

*coll(): case insensitive match 에 유용. locale 지정 가능. 그러나 상대적으로 느리다.

```

# That means you also need to be aware of the difference
# when doing case insensitive matches:
i <- c("I", "İ", "i", "ı")
i

## [1] "I"          "<U+0130>" "i"          "ı"

str_subset(i, coll("i", ignore_case = TRUE))

## [1] "I" "i"

str_subset(i, coll("i", ignore_case = TRUE, locale = "tr"))

## [1] "i"

```

- boundary 함수를 이용하여 단어 찾아내기

```

x <- "This is a sentence."
#str_view_all(x, boundary("word"))
str_extract_all(x, boundary("word"))

```

```
## [[1]]  
## [1] "This"      "is"      "a"      "sentence"
```

Other uses of regular expressions

- `apropos()`: global environment 중 match 가 있는 모든 object 찾기

```
apropos("replace")
```

```
## [1] "%+replace%"      "replace"          "replace_na"  
## [4] "setReplaceMethod" "str_replace"      "str_replace_all"  
## [7] "str_replace_na"   "theme_replace"
```

* ``dir()``: directory 내의 모든 file 중에서 찾기

```
head(dir(pattern = "\\..Rmd$"))
```

```
## [1] "Part 1-Explore.Rmd" "Part 2-Wrangle.Rmd" "Part_2-Wrangle.Rmd"
```

stringi

- `stringr`: `stringi` 위에 만들어진 package 로 꼭 필요한 최소한의 함수들로 구성되어 있어서 처음 배우기는 매우 좋음.
- `stringi`: 대부분의 string manipulation 함수들을 가지고 있으므로 모든 조작이 가능. 함수의 이름은 `stri_`로 시작.

12. Factors with forcats

Introduction

- forcats: 범주형 자료를 다루기 위한 다양한 함수 제공

```
library(tidyverse)
library(forcats)
```

범주형 변수 만들기

- factor 로 변환하는데에 두가지 문제점
 1. “월”을 입력하였으나 알파벳순서로 level 이 지정되어 원하는 순서대로 정렬되지 않음
 2. typo 가 있는 경우에도 그대로 입력됨

```
x1 <- c("Dec", "Apr", "Jan", "Mar")
factor(x1)
```

```
## [1] Dec Apr Jan Mar
## Levels: Apr Dec Jan Mar
```

```
sort(x1)
```

```
## [1] "Apr" "Dec" "Jan" "Mar"
```

```
x2 <- c("Dec", "Apr", "Jam", "Mar")
factor(x2)
```

```
## [1] Dec Apr Jam Mar
## Levels: Apr Dec Jam Mar
```

- levels 를 이용하여 level 지정하여야 올바르게 입력됨

```
month_levels <- c(
  "Jan", "Feb", "Mar", "Apr", "May", "Jun",
  "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"
)
```

```
y1 <- factor(x1, levels = month_levels)
y1
```

```
## [1] Dec Apr Jan Mar
## Levels: Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
```

```
sort(y1)
```

```
## [1] Jan Mar Apr Dec
## Levels: Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
```

```
y2 <- factor(x2, levels = month_levels)
y2

## [1] Dec Apr <NA> Mar
## Levels: Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
```

- readr::parse_factor()을 이용하면 parsing error 를 찾을 수 있음.

```
y2 <- parse_factor(x2, levels = month_levels)

## Warning: 1 parsing failure.
## row col expected actual
## 3 -- value in level set Jam

y2

## [1] Dec Apr <NA> Mar
## attr(,"problems")
## # A tibble: 1 x 4
## row col expected actual
## <int> <int> <chr> <chr>
## 1 3 NA value in level set Jam
## Levels: Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
```

- unique(x), fct_inorder(): 자료에서 나오는 순서에 따라 level 을 정함

```
f1 <- factor(x1, levels = unique(x1))
f1

## [1] Dec Apr Jan Mar
## Levels: Dec Apr Jan Mar

f2 <- x1 %>% factor() %>% fct_inorder()
f2

## [1] Dec Apr Jan Mar
## Levels: Dec Apr Jan Mar
```

- levels(): factor class 의 범주를 알려줌.

```
levels(f2)

## [1] "Dec" "Apr" "Jan" "Mar"
```

General Social Survey

- forcats::gss_cat: General Social Survey(<http://gss.norc.org>)의 sample data.
- year, age, marital, race, rincome, partyid, relig, denom, tvhours 의 9 개 변수가 있음

```
gss_cat
```

```
## # A tibble: 21,483 x 9
##   year marital    age race  rincome  partyid  relig  denom  tvhours
##   <int> <fct>    <int> <fct> <fct>    <fct>    <fct> <fct>    <int>
## 1  2000 Never ma~    26 White $8000 to~ Ind,near~ Protes~ Southe~    12
## 2  2000 Divorced    48 White $8000 to~ Not str ~ Protes~ Baptis~    NA
## 3  2000 Widowed     67 White Not appl~ Independ~ Protes~ No den~     2
## 4  2000 Never ma~    39 White Not appl~ Ind,near~ Orthod~ Not ap~     4
## 5  2000 Divorced    25 White Not appl~ Not str ~ None    Not ap~     1
## 6  2000 Married     25 White $20000 ~ Strong d~ Protes~ Southe~    NA
## 7  2000 Never ma~    36 White $25000 o~ Not str ~ Christ~ Not ap~     3
## 8  2000 Divorced    44 White $7000 to~ Ind,near~ Protes~ Luther~    NA
## 9  2000 Married     44 White $25000 o~ Not str ~ Protes~ Other     0
## 10 2000 Married     47 White $25000 o~ Strong r~ Protes~ Southe~     3
## # ... with 21,473 more rows
```

- count()와 table()의 차이

```
levels(gss_cat$race)
```

```
## [1] "Other"          "Black"          "White"          "Not applicable"
```

```
table(gss_cat$race)
```

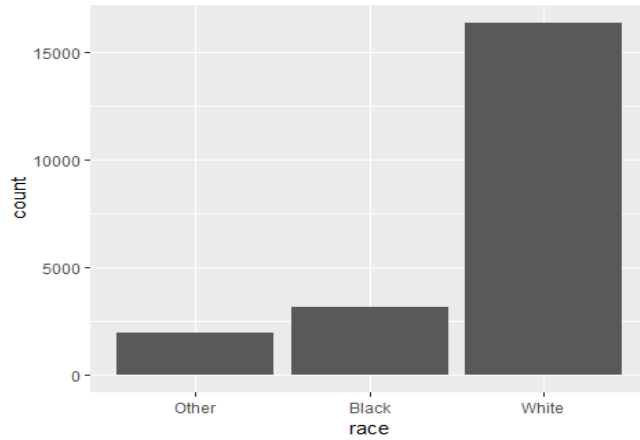
```
##
##      Other      Black      White Not applicable
##      1959      3129      16395             0
```

```
gss_cat %>%
  count(race)
```

```
## # A tibble: 3 x 2
##   race      n
##   <fct> <int>
## 1 Other  1959
## 2 Black  3129
## 3 White 16395
```

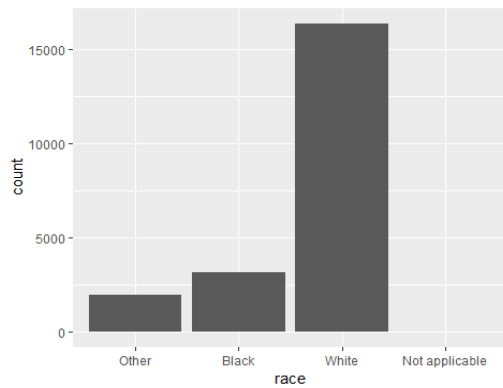

- barchart

```
ggplot(gss_cat, aes(race)) +  
  geom_bar()
```



- 자료는 없지만 level 에 지정된 값에 대하여 모두 그리기

```
ggplot(gss_cat, aes(race)) +  
  geom_bar() +  
  scale_x_discrete(drop = FALSE)
```

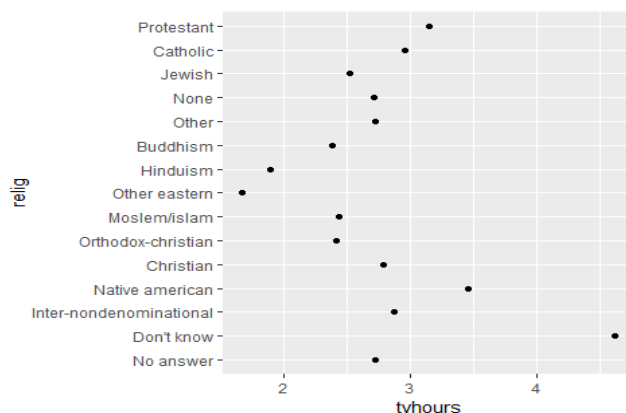


factor 순서 바꾸기

- gss_cat 자료를 이용하여 region 별 하루 평균 TV 시청 시간을 살펴보기

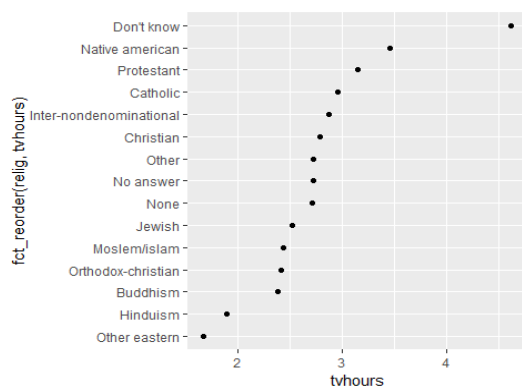
```
relig <- gss_cat %>%  
  group_by(relig) %>%  
  summarise(  
    age = mean(age, na.rm = TRUE),  
    tvhours = mean(tvhours, na.rm = TRUE),  
    n = n()  
  )
```

```
ggplot(relig, aes(tvhours, relig)) + geom_point()
```



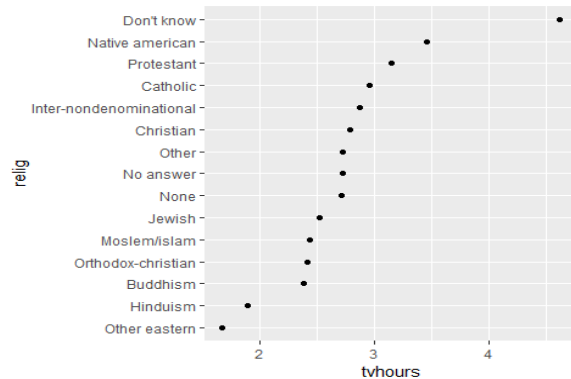
- fct_reorder()를 이용하여 level 순서 바꾸기
- f: 범주순서를 바꾸고자 하는 factor 지정
- x: level 순서를 바꾸는데 사용할 수치변수 *fun: x 가 여러 값일 경우 이용할 함수.
기본은 median

```
ggplot(relig, aes(tvhours, fct_reorder(relig, tvhours))) +  
  geom_point()
```



- `fct_reorder()`을 이용하여 변수 생성 후 그림을 그려도 됨.

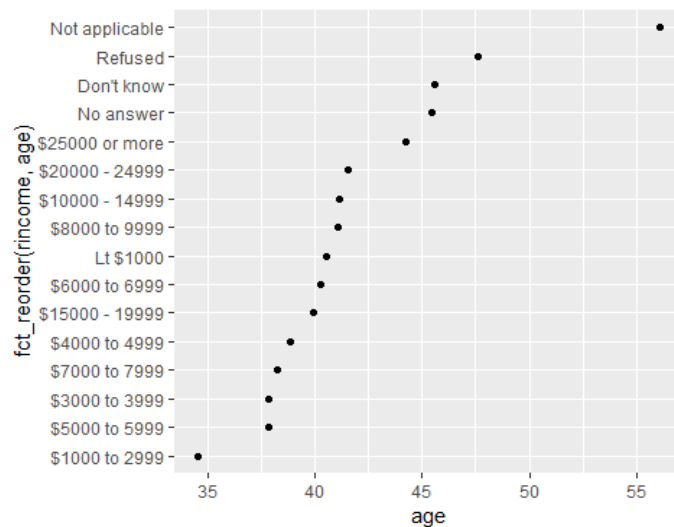
```
relig %>%
  mutate(relig = fct_reorder(relig, tvhours)) %>%
  ggplot(aes(tvhours, relig)) +
    geom_point()
```



- average reported income 을 평균 나이에 따라 순서 바꾸기

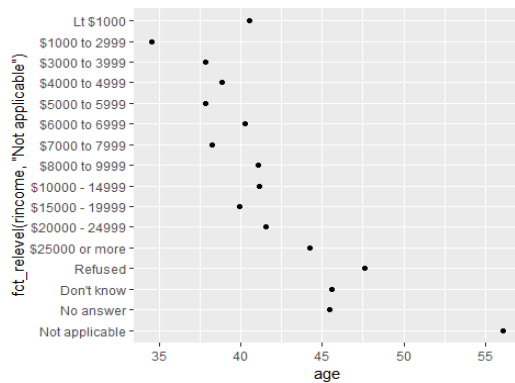
```
rincome <- gss_cat %>%
  group_by(rincome) %>%
  summarise(
    age = mean(age, na.rm = TRUE),
    tvhours = mean(tvhours, na.rm = TRUE),
    n = n()
  )
```

```
ggplot(rincome, aes(age, fct_reorder(rincome, age))) + geom_point()
```



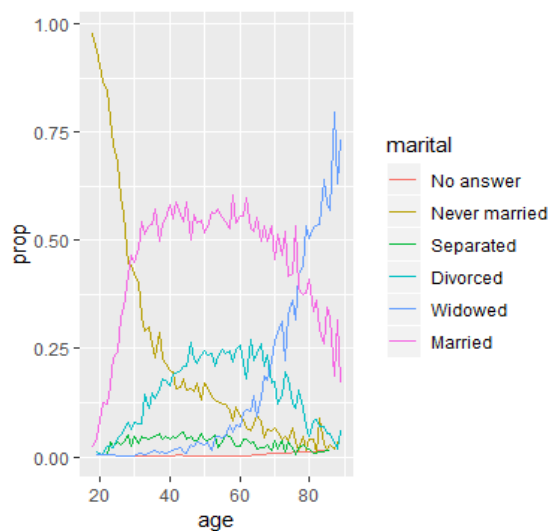
- rincome 의 경우 이미 순서가 있는 범주이므로 이를 age 순서에 따라 순서화하는 것은 별 의미 없음. 단, “Not applicable” 위치 조정은 필요.

```
ggplot(rincome, aes(age, fct_relevel(rincome, "Not applicable")) +  
  geom_point())
```

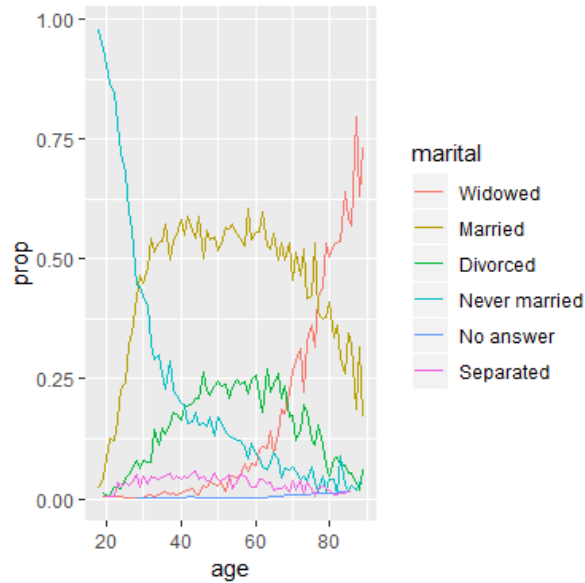


- “Not applicable”의 나이가 왜 높을까?
- fct_reorder2(): x 의 가장 큰 값에 대한 y 값 순서로 범주 순서 조정.

```
by_age <- gss_cat %>%  
  filter(!is.na(age)) %>%  
  group_by(age, marital) %>%  
  count() %>% group_by(age)%>%  
  mutate(prop = n / sum(n))  
  
ggplot(by_age, aes(age, prop, colour = marital)) +  
  geom_line(na.rm = TRUE)
```

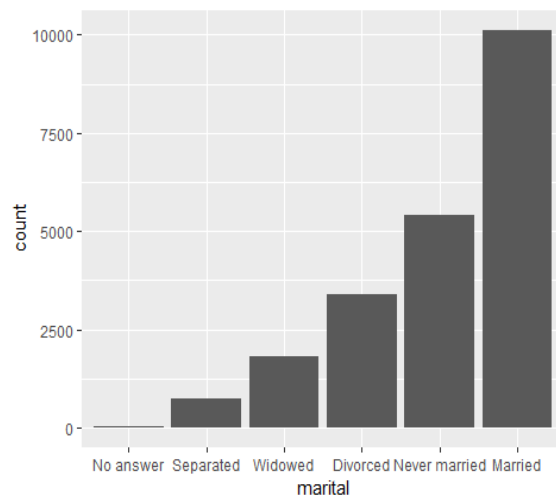


```
ggplot(by_age, aes(age, prop, colour = fct_reorder2(marital, age, prop))) +  
  geom_line() +  
  labs(colour = "marital")
```



- `fct_infreq()`: frequency 순으로 정렬
- `fct_rev()`: 거꾸로 정렬

```
gss_cat %>%  
  mutate(marital = marital %>% fct_infreq() %>% fct_rev()) %>%  
  ggplot(aes(marital)) +  
  geom_bar()
```



factor level 바꾸기

- `fct_recode()`: factor 의 level 을 조정하는 함수

```
gss_cat %>% count(partyid)

## # A tibble: 10 x 2
##   partyid          n
##   <fct>         <int>
## 1 No answer      154
## 2 Don't know      1
## 3 Other party    393
## 4 Strong republican 2314
## 5 Not str republican 3032
## 6 Ind,near rep    1791
## 7 Independent    4119
## 8 Ind,near dem    2499
## 9 Not str democrat 3690
## 10 Strong democrat 3490
```

- 범주의 이름 바꾸기

```
gss_cat %>%
  mutate(partyid = fct_recode(partyid,
    "Republican, strong" = "Strong republican",
    "Republican, weak" = "Not str republican",
    "Independent, near rep" = "Ind,near rep",
    "Independent, near dem" = "Ind,near dem",
    "Democrat, weak" = "Not str democrat",
    "Democrat, strong" = "Strong democrat"
  )) %>%
  count(partyid)

## # A tibble: 10 x 2
##   partyid          n
##   <fct>         <int>
## 1 No answer      154
## 2 Don't know      1
## 3 Other party    393
## 4 Republican, strong 2314
## 5 Republican, weak 3032
## 6 Independent, near rep 1791
## 7 Independent    4119
## 8 Independent, near dem 2499
## 9 Democrat, weak 3690
## 10 Democrat, strong 3490
```

- 몇개의 범주를 통합하는것도 가능

```

gss_cat %>%
  mutate(partyid = fct_recode(partyid,
    "Republican, strong" = "Strong republican",
    "Republican, weak" = "Not str republican",
    "Independent, near rep" = "Ind,near rep",
    "Independent, near dem" = "Ind,near dem",
    "Democrat, weak" = "Not str democrat",
    "Democrat, strong" = "Strong democrat",
    "Other" = "No answer",
    "Other" = "Don't know",
    "Other" = "Other party"
  )) %>%
  count(partyid)

## # A tibble: 8 x 2
##   partyid      n
##   <fct>    <int>
## 1 Other      548
## 2 Republican, strong 2314
## 3 Republican, weak 3032
## 4 Independent, near rep 1791
## 5 Independent      4119
## 6 Independent, near dem 2499
## 7 Democrat, weak 3690
## 8 Democrat, strong 3490

```

- fct_collapse()를 이용해도 됨.

```

gss_cat %>%
  mutate(partyid = fct_collapse(partyid,
    other = c("No answer", "Don't know", "Other party"),
    rep = c("Strong republican", "Not str republican"),
    ind = c("Ind,near rep", "Independent", "Ind,near dem"),
    dem = c("Not str democrat", "Strong democrat")
  )) %>%
  count(partyid)

## # A tibble: 4 x 2
##   partyid      n
##   <fct>    <int>
## 1 other      548
## 2 rep      5346
## 3 ind      8409
## 4 dem      7180

```

- fct_lump(): sample size 가 작은 그룹을 자동으로 통합

```

gss_cat %>%
  mutate(relig = fct_lump(relig)) %>%
  count(relig)

```

```
## # A tibble: 2 x 2
##   relig      n
##   <fct>    <int>
## 1 Protestant 10846
## 2 Other      10637
```

- n 개의 그룹으로 통합

```
gss_cat %>%
  mutate(relig = fct_lump(relig, n = 10)) %>%
  count(relig, sort = TRUE) %>%
  print(n = Inf)
```

```
## # A tibble: 10 x 2
##   relig      n
##   <fct>    <int>
## 1 Protestant 10846
## 2 Catholic    5124
## 3 None        3523
## 4 Christian    689
## 5 Other        458
## 6 Jewish       388
## 7 Buddhism     147
## 8 Inter-nondenominational 109
## 9 Moslem/islam  104
## 10 Orthodox-christian    95
```


13. Dates and times with lubridate

서론

- 날짜, 시간을 다룰 때에 문제가 되는 점들은 아래의 세 질문에 대한 것임.
- 1 년은 모두 365 일 인가?: 윤년을 고려해야 함
- 하루는 모두 24 시간 인가?: 지역에 따른 summer time 을 고려해야함.
- 1 분은 모두 60 초 인가?: 때때로 시간을 맞추기 위하여 1 초씩 조정하는 것을 고려.
- 필요한 사항들 *lubridate package

```
library(tidyverse)
```

```
library(lubridate)  
library(nycflights13)
```

날짜/시간 만들기

- tibble 에서 인식하는 날짜/시간의 3 가지 형태
 1. 날짜 : "Date" class
 2. 시간
 3. 날짜와 시간 : "POSIXct", "POSIXt" class

```
a<-today()  
class(a)
```

```
## [1] "Date"
```

```
a
```

```
## [1] "2019-03-07"
```

```
b<-now()  
class(b)
```

```
## [1] "POSIXct" "POSIXt"
```

```
b
```

```
## [1] "2019-03-07 12:04:06 KST"
```

- 입력 type 에 따른 날짜/시간 만들기
 1. 문자열 형태로 입력하는 경우

- 연(y), 월(m), 일(d)의 순서를 지정해야 함.

```
ymd("2017-01-31")
## [1] "2017-01-31"

mdy("January 31st, 2017")
## [1] "2017-03-01"

dmy("31-Jan-2017")
## [1] "2017-01-31"

ymd(20170131)
## [1] "2017-01-31"

ymd(170131)
## [1] "2017-01-31"
```

- 화면상에는 string 처럼 보이지만 Date 의 속성을 가지고 있음.

```
class(ymd("2017-01-31"))
## [1] "Date"
```

- 시간 입력을 위해서는 시(h), 분(m), 초(s)를 지정해야함.

```
ymd_hms("2017-01-31 20:11:59")
## [1] "2017-01-31 20:11:59 UTC"

mdy_hm("01/31/2017 08:01")
## [1] "2017-01-31 08:01:00 UTC"
```

- 시간 입력 시 timezone 설정 가능. default timezone 은 UTC

```
ymd(20170131, tz = "UTC")
## [1] "2017-01-31 UTC"
```

2. 날짜, 시간을 따로 입력하는 경우

- flights 자료와 같이 연, 월, 일, 시, 분이 따로 입력되어 있는 경우 make_date(), make_datetime()를 이용.

```
flights %>%
  select(year, month, day, hour, minute, sched_dep_time) %>%
  mutate(departure = make_datetime(year, month, day, hour, minute))
```

```
## # A tibble: 336,776 x 7
##   year month   day hour minute sched_dep_time departure
##   <int> <int> <int> <dbl>  <dbl>         <int> <dtm>
## 1  2013     1     1     5     15           515 2013-01-01 05:15:00
## 2  2013     1     1     5     29           529 2013-01-01 05:29:00
## 3  2013     1     1     5     40           540 2013-01-01 05:40:00
## 4  2013     1     1     5     45           545 2013-01-01 05:45:00
## 5  2013     1     1     6     0            600 2013-01-01 06:00:00
## 6  2013     1     1     5     58           558 2013-01-01 05:58:00
## 7  2013     1     1     6     0            600 2013-01-01 06:00:00
## 8  2013     1     1     6     0            600 2013-01-01 06:00:00
## 9  2013     1     1     6     0            600 2013-01-01 06:00:00
## 10 2013     1     1     6     0            600 2013-01-01 06:00:00
## # ... with 336,766 more rows
```

- flights 자료와 같이 특이한 형태로 시간이 저장되어 있는 경우 이를 시, 분으로 나누는 함수를 만들어 이용.

```
make_datetime_100 <- function(year, month, day, time) {
  make_datetime(year, month, day, time %/% 100, time %% 100)
}

flights_dt <- flights %>%
  filter(!is.na(dep_time), !is.na(arr_time)) %>%
  mutate(
    dep_time = make_datetime_100(year, month, day, dep_time),
    arr_time = make_datetime_100(year, month, day, arr_time),
    sched_dep_time = make_datetime_100(year, month, day, sched_dep_time),
    sched_arr_time = make_datetime_100(year, month, day, sched_arr_time)
  ) %>%
  select(origin, dest, ends_with("delay"), ends_with("time"))

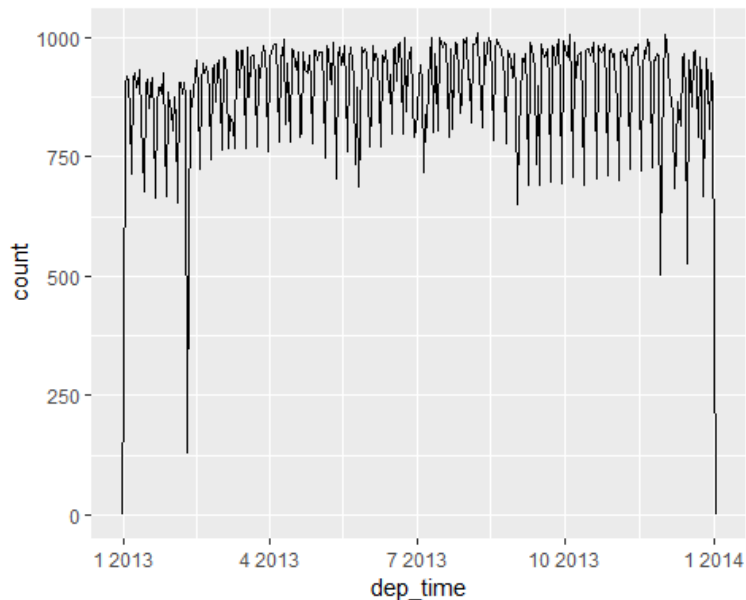
flights_dt

## # A tibble: 328,063 x 9
##   origin dest dep_delay arr_delay dep_time sched_dep_time
##   <chr>  <chr>    <dbl>    <dbl> <dtm>         <dtm>
## 1 EWR    IAH         2        11 2013-01-01 05:17:00 2013-01-01 05:15:00
## 2 LGA    IAH         4        20 2013-01-01 05:33:00 2013-01-01 05:29:00
## 3 JFK    MIA         2        33 2013-01-01 05:42:00 2013-01-01 05:40:00
## 4 JFK    BQN        -1       -18 2013-01-01 05:44:00 2013-01-01 05:45:00
## 5 LGA    ATL        -6       -25 2013-01-01 05:54:00 2013-01-01 06:00:00
```

```
## 6 EWR    ORD      -4      12 2013-01-01 05:54:00 2013-01-01 05:58:00
## 7 EWR    FLL      -5      19 2013-01-01 05:55:00 2013-01-01 06:00:00
## 8 LGA    IAD      -3     -14 2013-01-01 05:57:00 2013-01-01 06:00:00
## 9 JFK    MCO      -3      -8 2013-01-01 05:57:00 2013-01-01 06:00:00
## 10 LGA   ORD      -2       8 2013-01-01 05:58:00 2013-01-01 06:00:00
## # ... with 328,053 more rows, and 3 more variables: arr_time <dtm>,
## #   sched_arr_time <dtm>, air_time <dbl>
```

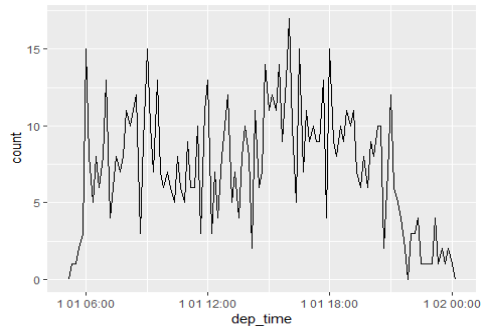
- 일별 비행횟수 비교를 위한 그림
- 형태로 바꾸고 나면 연속적인 시간(초단위)으로 인식, ggplot2 에서 연속변수와 같은 형태로 처리가 가능.

```
flights_dt %>%
  ggplot(aes(dep_time)) +
  geom_freqpoly(binwidth = 86400) # 86400 seconds = 1 day
```



- 2013 년 1 월 2 일의 시간별(10 분단위) 비행 횟수

```
flights_dt %>%
  filter(dep_time < ymd(20130102)) %>%
  ggplot(aes(dep_time)) +
  geom_freqpoly(binwidth = 600) # 600 s = 10 minutes
```



3. R 의 date/time object 형태

- as_datetime() and as_date():, , 으로 지정되어있는 자료를 다른 형태로 바꾸기 위해서 사용.

```
today()
```

```
## [1] "2019-03-07"
```

```
as_datetime(today())
```

```
## [1] "2019-03-07 UTC"
```

```
now()
```

```
## [1] "2019-03-07 12:04:07 KST"
```

```
as_date(now())
```

```
## [1] "2019-03-07"
```

- 정수를 입력하는 경우 1970 년 1 월 1 일을 기준으로 계산
- as_datetime(): 초단위로 입력
- as_date(): 일단위로 입력

```
as_datetime(60 * 60 * 10) # 1970 년 1 월 1 일 0 시 0 분 이후 36000 초=10 시간
```

```
## [1] "1970-01-01 10:00:00 UTC"
```

```
as_date(365 * 10 + 2) # 1970 년 1 월 1 일 이후 3652 일
```

```
## [1] "1980-01-01"
```

Date-time component 나누기

- year(), month(), mday() (day of the month), yday() (day of the year), wday() (day of the week), hour(), minute(), second() 등의 함수 이용.

```
datetime <- ymd_hms("2019-03-04 12:30:52")
year(datetime)
## [1] 2019

month(datetime)
## [1] 3

mday(datetime) # 2016 년 7 월 1 일부터의 날짜수
## [1] 4

yday(datetime) # 2016 년 1 월 1 일부터의 날짜수
## [1] 63

wday(datetime) # 요일 일요일(1) ~ 토요일(7)
## [1] 2
```

- month()와 wday()의 옵션: label = TRUE, abbr = TRUE

```
month(datetime, label=TRUE, abbr=TRUE)
## [1] 3
## Levels: 1 < 2 < 3 < 4 < 5 < 6 < 7 < 8 < 9 < 10 < 11 < 12

month(datetime, label=TRUE, abbr=FALSE, locale="US")
## [1] March
## 12 Levels: January < February < March < April < May < June < ... < December

month(datetime, label=TRUE, abbr=TRUE, locale="US")
## [1] Mar
## 12 Levels: Jan < Feb < Mar < Apr < May < Jun < Jul < Aug < Sep < ... < Dec

wday(datetime, label=TRUE, abbr=TRUE)
## [1] 월
## Levels: 일 < 월 < 화 < 수 < 목 < 금 < 토

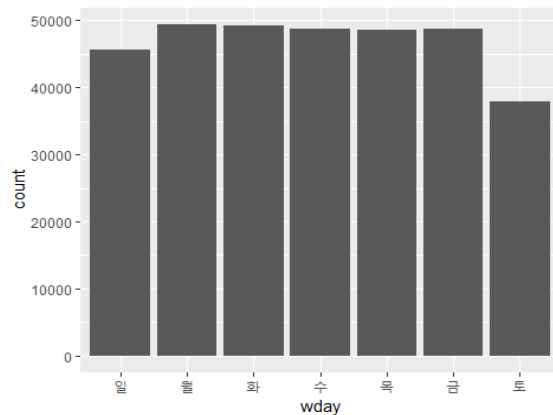
wday(datetime, label=TRUE, abbr=FALSE, locale="US")
```

```
## [1] Monday
## 7 Levels: Sunday < Monday < Tuesday < Wednesday < Thursday < ... < Saturday
wday(datetime,label=TRUE,abbr=TRUE,locale="US")

## [1] Mon
## Levels: Sun < Mon < Tue < Wed < Thu < Fri < Sat
```

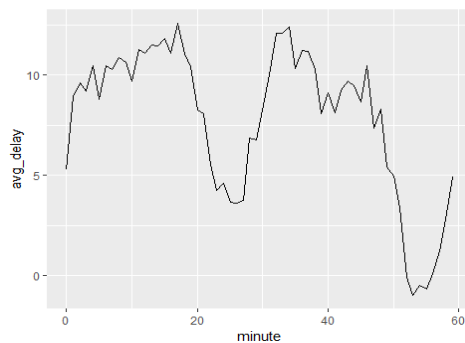
- 요일별 비행 건수 비교

```
flights_dt %>%
  mutate(wday = wday(dep_time, label = TRUE)) %>%
  ggplot(aes(x = wday)) +
    geom_bar()
```



- 매 시간대의 0~59 분 별 평균 출발지연시간 살펴보기

```
flights_dt %>%
  mutate(minute = minute(dep_time)) %>%
  group_by(minute) %>%
  summarise(
    avg_delay = mean(arr_delay, na.rm = TRUE),
    n = n()) %>%
  ggplot(aes(minute, avg_delay)) +
    geom_line()
```



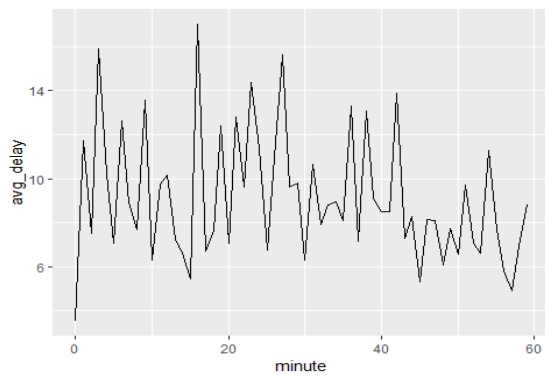
- 실제 출발시간을 이용한 경우 20~30 분, 그리고 50~60 분에 출발지연이 짧은 경향이 있다.

```

sched_dep <- flights_dt %>%
  mutate(minute = minute(sched_dep_time)) %>%
  group_by(minute) %>%
  summarise(
    avg_delay = mean(arr_delay, na.rm = TRUE),
    n = n())

ggplot(sched_dep, aes(minute, avg_delay)) +
  geom_line()

```

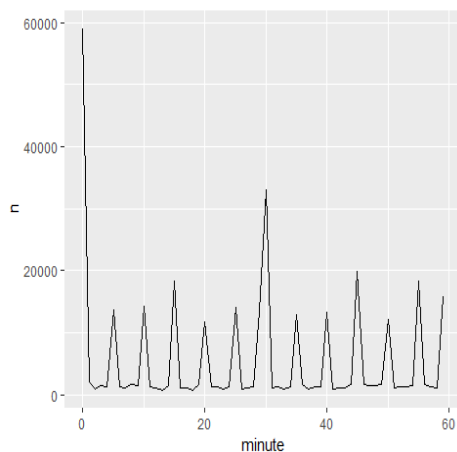


- 예정 출발시간을 이용한 경우에는 별다른 패턴이 보이지 않는다.
- 5 분 간격으로 피크를 보이는 것은 예정출발시간이 대부분 5 분 단위로 되어있기 때문.

```

ggplot(sched_dep, aes(minute, n)) +
  geom_line()

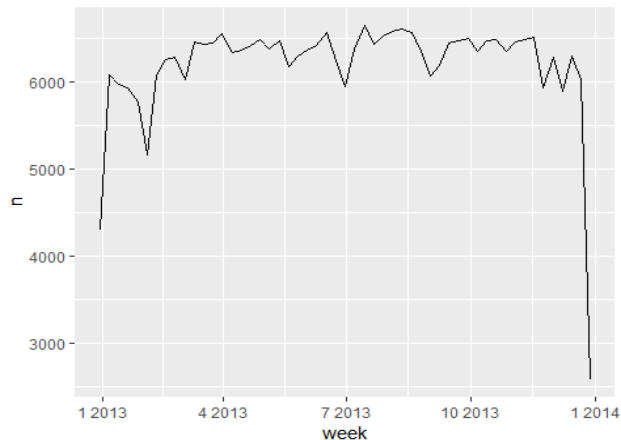
```



시간을 이용한 반올림

- `floor_date()`, `round_date()`, `ceiling_date()`: 시간을 이용한 내림, 반올림, 올림.
- `unit`: second, minute, hour, day, week, month, bimonth, quarter, season, halfyear, year

```
flights_dt %>%  
  count(week = floor_date(dep_time, "week")) %>%  
  ggplot(aes(week, n)) +  
    geom_line()
```



Setting components

- `year`, `month`, `hour` 함수를 이용하여 연도, 달, 시간 등을 바꿀 수 있음. You can also use each accessor function to set the components of a date/time:

```
(datetime <- ymd_hms("2016-07-08 12:34:56"))
```

```
## [1] "2016-07-08 12:34:56 UTC"
```

```
year(datetime) <- 2020  
datetime
```

```
## [1] "2020-07-08 12:34:56 UTC"
```

```
month(datetime) <- 01  
datetime
```

```
## [1] "2020-01-08 12:34:56 UTC"
```

```
hour(datetime) <- hour(datetime) + 1
```

- `update` 함수를 이용하여 동시에 바꿀 수 있음.

```
datetime
```

```
## [1] "2020-01-08 13:34:56 UTC"
```

```
update(datetime, year = 2020, month = 2, mday = 2, hour = 2)
```

```
## [1] "2020-02-02 02:34:56 UTC"
```

- 큰 값이 입력되는 경우 입력된 수 만큼의 시간이 흐른 날짜, 시간을 계산

```
ymd("2015-02-01") %>%  
  update(mday = 30)
```

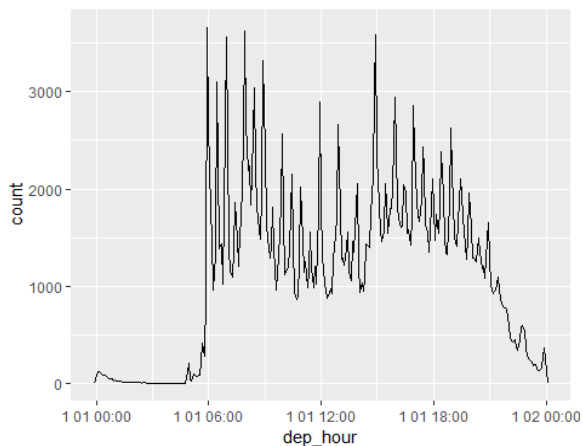
```
## [1] "2015-03-02"
```

```
ymd("2015-02-01") %>%  
  update(hour = 400)
```

```
## [1] "2015-02-17 16:00:00 UTC"
```

- 1 년동안 각 출발시간별 비행건수를 알아보기 위하여 update 함수를 이용, 모든 비행의 날짜를 1 월 1 일로 변경한 후 dep_time 에 대한 그림을 그림.

```
flights_dt %>%  
  mutate(dep_hour = update(dep_time, yday = 1)) %>%  
  ggplot(aes(dep_hour)) +  
    geom_freqpoly(binwidth = 300)
```



- 큰 단위(month, day)를 상수로 고정시키는 기법은 작은 단위(시, 분)별로 자료를 살펴보는 데에 매우 유용한 방법임.

시간 간격

Next you'll learn about how arithmetic with dates works, including subtraction, addition, and division. Along the way, you'll learn about three important classes that represent time spans:

- **durations** : 초단위의 시간간격
- **periods**: 주, 월 단위의 시간간격

- **intervals:** 시간과 끝을 나타냄

Durations

- 시간차이 계산은 Date, time 등의 시간 관련 object 를 이용하면 된다.

```
# How old is Hadley?
h_age <- today() - ymd(19791014)
h_age

## Time difference of 14389 days

class(h_age)

## [1] "difftime"
```

- difftime class 는 시간 간격을 초, 분, 시간, 날짜, 주 단위로 나타낸다.
- lubridate library 의 duration 함수를 이용하면 모두 초단위로 나타냄.

```
as.duration(h_age)

## [1] "1243209600s (~39.39 years)"
```

- duration 을 활용한 다양한 함수들.

```
dseconds(15)

## [1] "15s"

dminutes(10)

## [1] "600s (~10 minutes)"

dhours(c(12, 24))

## [1] "43200s (~12 hours)" "86400s (~1 days)"

ddays(0:5)

## [1] "0s" "86400s (~1 days)" "172800s (~2 days)"
## [4] "259200s (~3 days)" "345600s (~4 days)" "432000s (~5 days)"

dweeks(3)

## [1] "1814400s (~3 weeks)"

dyears(1)

## [1] "31536000s (~52.14 weeks)"
```

- +, -, * 모두 사용가능

```
2 * dyears(1)
```

```
## [1] "63072000s (~2 years)"
dyears(1) + dweeks(12) + dhours(15)
## [1] "38847600s (~1.23 years)"
dyears(1) - ddays(364)

## Note: method with signature 'Duration#ANY' chosen for function '-',
## target signature 'Duration#Duration'.
## "ANY#Duration" would also be valid

## [1] "86400s (~1 days)"
```

- Date 로 부터의 덧셈, 뺄셈 모두 가능

```
tomorrow <- today() + ddays(1)
last_year <- today() - dyears(1)
```

- duration 의 시간 계산에서는 time zone 에 따른 시간 차이가 포함되므로 이를 고려할 것.

```
one_pm <- ymd_hms("2016-03-12 13:00:00", tz = "America/New_York")

one_pm
## [1] "2016-03-12 13:00:00 EST"
one_pm + ddays(1)
## [1] "2016-03-13 14:00:00 EDT"
```

Periods

- time zone 에 다른 시간 차이에 대한 문제를 해결하기 위하여 lubridate 에서는 periods 함수를 제공

```
one_pm
## [1] "2016-03-12 13:00:00 EST"
one_pm + days(1)
## [1] "2016-03-13 13:00:00 EDT"
```

- duration 과 마찬가지로 period 에서도 다양한 함수 제공.

```
seconds(15)
## [1] "15S"
minutes(10)
```

```
## [1] "10M 0S"
hours(c(12, 24))
## [1] "12H 0M 0S" "24H 0M 0S"
days(7)
## [1] "7d 0H 0M 0S"
months(1:6)
## [1] "1m 0d 0H 0M 0S" "2m 0d 0H 0M 0S" "3m 0d 0H 0M 0S" "4m 0d 0H 0M 0S"
## [5] "5m 0d 0H 0M 0S" "6m 0d 0H 0M 0S"
weeks(3)
## [1] "21d 0H 0M 0S"
years(1)
## [1] "1y 0m 0d 0H 0M 0S"
```

- +, -, * 가능

```
10 * (months(6) + days(1))
## [1] "60m 10d 0H 0M 0S"
days(50) + hours(25) + minutes(2)
## [1] "50d 25H 2M 0S"
days(50) + hours(25) - minutes(2)
## Note: method with signature 'Period#ANY' chosen for function '-',
## target signature 'Period#Period'.
## "ANY#Period" would also be valid
## [1] "50d 25H -2M 0S"
```

- Date 와 함께 연산 가능

```
# A Leap year
ymd("2016-01-01") + dyears(1)
## [1] "2016-12-31"
ymd("2016-01-01") + years(1)
## [1] "2017-01-01"
# Daylight Savings Time
one_pm + ddays(1)
```

```
## [1] "2016-03-13 14:00:00 EDT"
```

```
one_pm + days(1)
```

```
## [1] "2016-03-13 13:00:00 EDT"
```

- 시간 연산을 이용하여 dep_time 이 arr_time 보다 앞서는 이상한 비행 찾아내기.

```
flights_dt %>%
```

```
  filter(arr_time < dep_time)
```

```
## # A tibble: 10,633 x 9
```

```
##   origin dest  dep_delay arr_delay dep_time          sched_dep_time
```

```
##   <chr>  <chr>    <dbl>    <dbl> <dtm>          <dtm>
```

```
## 1 EWR    BQN        9      -4 2013-01-01 19:29:00 2013-01-01 19:20:00
```

```
## 2 JFK    DFW       59      NA 2013-01-01 19:39:00 2013-01-01 18:40:00
```

```
## 3 EWR    TPA       -2        9 2013-01-01 20:58:00 2013-01-01 21:00:00
```

```
## 4 EWR    SJU       -6     -12 2013-01-01 21:02:00 2013-01-01 21:08:00
```

```
## 5 EWR    SFO       11     -14 2013-01-01 21:08:00 2013-01-01 20:57:00
```

```
## 6 LGA    FLL     -10      -2 2013-01-01 21:20:00 2013-01-01 21:30:00
```

```
## 7 EWR    MCO       41      43 2013-01-01 21:21:00 2013-01-01 20:40:00
```

```
## 8 JFK    LAX       -7     -24 2013-01-01 21:28:00 2013-01-01 21:35:00
```

```
## 9 EWR    FLL       49      28 2013-01-01 21:34:00 2013-01-01 20:45:00
```

```
## 10 EWR   FLL       -9     -14 2013-01-01 21:36:00 2013-01-01 21:45:00
```

```
## # ... with 10,623 more rows, and 3 more variables: arr_time <dtm>,
```

```
## #   sched_arr_time <dtm>, air_time <dbl>
```

- 이들은 밤 비행기로 출발과 비행에 같은 날짜를 이용하고 있으나 비행기는 다음날 도착한 것이다. 이를 days(1)를 이용하여 수정.

```
flights_dt <- flights_dt %>%
```

```
  mutate(
```

```
    overnight = arr_time < dep_time,
```

```
    arr_time = arr_time + days(overnight * 1),
```

```
    sched_arr_time = sched_arr_time + days(overnight * 1)
```

```
  )
```

```
flights_dt %>%
  filter(overnight, arr_time < dep_time)

## # A tibble: 0 x 10
## # ... with 10 variables: origin <chr>, dest <chr>, dep_delay <dbl>,
## #   arr_delay <dbl>, dep_time <dtm>, sched_dep_time <dtm>,
## #   arr_time <dtm>, sched_arr_time <dtm>, air_time <dbl>,
## #   overnight <lgl>
```

Intervals

- duration 에서는 1 년을 365 일로 계산, interval 에서는 윤년을 고려

```
dyears(1) / ddays(365)
```

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

```
years(1) / days(1)
```

```
## estimate only: convert to intervals for accuracy
```

```
## [1] 365.25
```

If you want a more accurate measurement, you'll have to use an **interval**. An interval is a duration with a starting point: that makes it precise so you can determine exactly how long it is:

```
next_year <- today() + years(1)
today() %--% next_year

## [1] 2019-03-07 UTC--2020-03-07 UTC

(today() %--% next_year) / ddays(1)

## [1] 366

last_year <- today() - years(1)
last_year %--% today()

## [1] 2018-03-07 UTC--2019-03-07 UTC

(last_year %--% today()) / ddays(1)

## [1] 365
```

- 구간 내에 몇주가 있는지를 알기 위해서는 %/%를 이용

```
(today() %--% next_year) %/% weeks(1)
```

```
## Note: method with signature 'Timespan#Timespan' chosen for function '%/%',
## target signature 'Interval#Period'.
## "Interval#ANY", "ANY#Period" would also be valid
```

```
## [1] 52
```

```
(today() %--% next_year) / weeks(1)
```

```
## [1] 52.28571
```

Summary

- duration, period, interval 중 어떤 것을 사용할 것인가?
- 단순히 physical time 에 관심이 있다면 duration 이용
- human time 을 추가하려면 period 이용
- human unit 에서의 기간을 따지려면 period 이용
- 아래의 표 참고

| | date | | | date time | | | duration | | | period | | | interval | | | number | | |
|-----------|------|---|--|-----------|---|--|----------|---|---|--------|---|---|----------|---|---|--------|---|---|
| date | - | | | | | | - | + | | - | + | | | | | - | + | |
| date time | | | | - | | | - | + | | - | + | | | | | - | + | |
| duration | - | + | | - | + | | - | + | / | | | | | | | - | + | × |
| period | - | + | | - | + | | | | | - | + | | | | | - | + | × |
| interval | | | | | | | | | / | | | / | | | | | | |
| number | - | + | | - | + | | - | + | × | - | + | × | - | + | × | - | + | × |

date/time class 들 간의 가능한 산술 연산

Time zones

- timezone 은 매우 복잡
- timezone 알아보기

```
Sys.timezone()
```

```
## [1] "Asia/Seoul"
```

- 가능한 timezone list

```
length(OlsonNames())
```

```
## [1] 592
```

```
head(OlsonNames())
```

```
## [1] "Africa/Abidjan"
```

```
"Africa/Accra"
```

```
"Africa/Addis_Ababa"
```

```
## [4] "Africa/Algiers"
```

```
"Africa/Asmara"
```

```
"Africa/Asmera"
```


- R에서는 date-time 의 attribute 중 하나로 시간 옆에 써줌
- 아래의 시간은 동일 시간, timezone 만 다르게 표시됨

```
(x1 <- ymd_hms("2015-06-01 12:00:00", tz = "America/New_York"))
## [1] "2015-06-01 12:00:00 EDT"

(x2 <- ymd_hms("2015-06-01 18:00:00", tz = "Europe/Copenhagen"))
## [1] "2015-06-01 18:00:00 CEST"

(x3 <- ymd_hms("2015-06-02 04:00:00", tz = "Pacific/Auckland"))
## [1] "2015-06-02 04:00:00 NZST"
```

- 빼기를 이용하여 같은 시간임을 확인할 수 있음

```
x1 - x2
## Time difference of 0 secs

x1 - x3
## Time difference of 0 secs
```

- lubridate에서는 UTC (Coordinated Universal Time)를 기준으로 계산.
- 여러 시간대를 합쳐서 만든 벡터의 경우 첫번째 값을 기준으로 시간대를 설정

```
x4 <- c(x1, x2, x3)
x4

## [1] "2015-06-01 12:00:00 EDT" "2015-06-01 12:00:00 EDT"
## [3] "2015-06-01 12:00:00 EDT"
```

You can change the time zone in two ways:

- timezone 을 맞춰서 바꾸기
- “2015-06-01 12:00:00 EDT”를 “Australia/Lord_Howe” 시간대의 시간 “2015-06-02 02:30:00 +1030”로 바꾸기

```
x4a <- with_tz(x4, tzone = "Australia/Lord_Howe")
x4a

## [1] "2015-06-02 02:30:00 +1030" "2015-06-02 02:30:00 +1030"
## [3] "2015-06-02 02:30:00 +1030"

x4a - x4

## Time differences in secs
## [1] 0 0 0
```

(This also illustrates another challenge of times zones: they're not all integer hour offsets!)

- timezone 이 잘못 입력된 경우 강제로 바꾸기
- “2015-06-01 12:00:00 +1030”이 “2015-06-01 12:00:00 EDT”로 잘못 입력된 경우

```
x4b <- force_tz(x4, tzone = "Australia/Lord_Howe")
```

```
x4b
```

```
## [1] "2015-06-01 12:00:00 +1030" "2015-06-01 12:00:00 +1030"
```

```
## [3] "2015-06-01 12:00:00 +1030"
```

```
x4b - x4
```

```
## Time differences in hours
```

```
## [1] -14.5 -14.5 -14.5
```