ST720 Data Science

Data Transformation with dplyr

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

Department of Statistics, Korea University

package : tidyverse

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

- Illustrate the `dplyr` package to `flights` data.
- dplyr overwrites some functions in base `R`.
- Use thier full names: `stats::filter()` and `stats:lag()`.

flights data in nycflights13

```
flights
```

```
## # A tibble: 336,776 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      517            515         2      830
##  2  2013     1     1      533            529         4      850
##  3  2013     1     1      542            540         2      923
##  4  2013     1     1      544            545        -1     1004
##  5  2013     1     1      554            600        -6      812
##  6  2013     1     1      554            558        -4      740
##  7  2013     1     1      555            600        -5      913
##  8  2013     1     1      557            600        -3      709
##  9  2013     1     1      557            600        -3      838
## 10  2013     1     1      558            600        -2      753
## # ... with 336,766 more rows, and 12 more variables: sched_arr_time
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <
## #   minute <dbl>, time_hour <dttm>
```

nycflights13

- The data looks a little different from the data frame.
- To see the whole data set, `view(flights)`.
- Abbreviations under the column names describe the variable type.
  - `int` for integer.
  - `dbl` for doubles or real numbers.
  - `chr` for characters or strings.
  - `dttm` for date-times.
  - `lgl` for logical values.
  - `fctr` for factors.
  - `date` for dates.

dplyr Basics

- `filter()`: Pick obsrvations by their value.
- `select()`: Pick variables by their names.
- `mutate()`": Create new variables with functions of erxisting variables.
- `arrange()`: Reorder the rows.
- `summarize()`: Collapse many values down to a single summary.
- All functions can be used in conjunction with `group_by()`.

dplyr Basics

- First argument is a data frame.
- Next ones describes what to do with dhata frame, using variable names (without quotes).
- Output is a new data frame.

Filter Rows with filter()

```r
jan1 <- filter(flights, month == 1, day == 1)
jan1
```

```
## # A tibble: 842 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      517            515         2      830
##  2  2013     1     1      533            529         4      850
##  3  2013     1     1      542            540         2      923
##  4  2013     1     1      544            545        -1     1004
##  5  2013     1     1      554            600        -6      812
##  6  2013     1     1      554            558        -4      740
##  7  2013     1     1      555            600        -5      913
##  8  2013     1     1      557            600        -3      709
##  9  2013     1     1      557            600        -3      838
## 10  2013     1     1      558            600        -2      753
## # ... with 832 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 <
## #   minute <dbl>, time_hour <dttm>
```

Logical Operations

- Equivalent code 1

```
filter(flights, month == 11 | month == 12)
filter(flights, month %in% c(11,12))
```

- Equivalent code 2

```
filter(flights, !(arr_delay > 120 | dep_delay > 12))
filter(flights, arr_delay <= 120,  dep_delay <= 12)
```

- `is.na()` and `near()` functions are useful.

Select Columns with select()

```r
select(flights, year, month, day)
select(flights, year:day)
select(flights, -(year:day))
```

There are some useful functions you can use within `select()`.

- `starts_with("abs")`: matches names that begin with “abc”.
- `ends_with("xyz)`: matches names that end with “xyz”.
- `contains("ijk")`: matches names that contain “ijk”.
- `num_range("x", 1:3)`: matches x1, x2, and x3

Select Columns with `select()`

- `rename()` is useful to change the variable name while keeping all other variables.

```r
rename(flights, tail_num = tailnum)
```

```
## # A tibble: 336,776 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      517            515         2      830
##  2  2013     1     1      533            529         4      850
##  3  2013     1     1      542            540         2      923
##  4  2013     1     1      544            545        -1     1004
##  5  2013     1     1      554            600        -6      812
##  6  2013     1     1      554            558        -4      740
##  7  2013     1     1      555            600        -5      913
##  8  2013     1     1      557            600        -3      709
##  9  2013     1     1      557            600        -3      838
## 10  2013     1     1      558            600        -2      753
## # ... with 336,766 more rows, and 12 more variables: sched_arr_time
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tail_num <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <
## #   minute <dbl>, time_hour <dttm>
```

Select Columns with select()

- To move some variables to the front, use everything() helper.

```
select(flights, time_hour, air_time, everything())
```

```
## # A tibble: 336,776 x 19
##    time_hour           air_time  year month   day dep_time sched_dep
##    <dttm>                 <dbl> <int> <int> <int>    <int>
##  1 2013-01-01 05:00:00      227  2013     1     1      517
##  2 2013-01-01 05:00:00      227  2013     1     1      533
##  3 2013-01-01 05:00:00      160  2013     1     1      542
##  4 2013-01-01 05:00:00      183  2013     1     1      544
##  5 2013-01-01 06:00:00      116  2013     1     1      554
##  6 2013-01-01 05:00:00      150  2013     1     1      554
##  7 2013-01-01 06:00:00      158  2013     1     1      555
##  8 2013-01-01 06:00:00       53  2013     1     1      557
##  9 2013-01-01 06:00:00      140  2013     1     1      557
## 10 2013-01-01 06:00:00      138  2013     1     1      558
## # ... with 336,766 more rows, and 12 more variables: dep_delay <dbl>
## #   arr_time <int>, sched_arr_time <int>, arr_delay <dbl>, carrier <
## #   flight <int>, tailnum <chr>, origin <chr>, dest <chr>, distance
## #   hour <dbl>, minute <dbl>
```

Add new variables with `mutate()`

- Useful to add new columns that are functions of exsiting columns.
- `mutate()` always adds new columns at the end of the data set.
- Create a small data set with less variables:

```r
flights_sml <- select(flights, year:day,
                      ends_with("delay"),
                      distance, air_time)
```

Add new variables with mutate()

```r
mutate(flights_sml,
       gain = arr_delay - dep_delay,
       speed = distance / air_time)
```

```
## # A tibble: 336,776 x 9
##     year month   day dep_delay arr_delay distance air_time  gain spe
##    <int> <int> <int>     <dbl>     <dbl>    <dbl>    <dbl> <dbl> <db
##  1  2013     1     1         2        11     1400      227     9  6.
##  2  2013     1     1         4        20     1416      227    16  6.
##  3  2013     1     1         2        33     1089      160    31  6.
##  4  2013     1     1        -1       -18     1576      183   -17  8.
##  5  2013     1     1        -6       -25      762      116   -19  6.
##  6  2013     1     1        -4        12      719      150    16  4.
##  7  2013     1     1        -5        19     1065      158    24  6.
##  8  2013     1     1        -3       -14      229       53   -11  4.
##  9  2013     1     1        -3        -8      944      140    -5  6.
## 10  2013     1     1        -2         8      733      138    10  5.
## # ... with 336,766 more rows
```

Add new variables with mutate()

- You can refer to columns that you've just created.

```
mutate(flights_sml,
       gain = arr_delay - dep_delay,
       hours = air_time/60,
       gain_per_hour = gain/hours)
```

```
## # A tibble: 336,776 x 10
##     year month   day dep_delay arr_delay distance air_time  gain hou
##    <int> <int> <int>     <dbl>     <dbl>    <dbl>    <dbl> <dbl> <db
##  1  2013     1     1         2        11     1400      227     9 3.7
##  2  2013     1     1         4        20     1416      227    16 3.7
##  3  2013     1     1         2        33     1089      160    31 2.6
##  4  2013     1     1        -1       -18     1576      183   -17 3.0
##  5  2013     1     1        -6       -25      762      116   -19 1.9
##  6  2013     1     1        -4        12      719      150    16 2.5
##  7  2013     1     1        -5        19     1065      158    24 2.6
##  8  2013     1     1        -3       -14      229       53   -11 0.8
##  9  2013     1     1        -3        -8      944      140    -5 2.3
## 10  2013     1     1        -2         8      733      138    10 2.3
## # ... with 336,766 more rows, and 1 more variable: gain_per_hour <db
```

Add new variables with transmute()

- If you only want to keep the new variables, use transmute().

```r
transmute(flights_sml,
       gain = arr_delay - dep_delay,
       hours = air_time/60,
       gain_per_hour = gain/hours)
```

```
## # A tibble: 336,776 x 3
##     gain hours gain_per_hour
##    <dbl> <dbl>         <dbl>
##  1     9 3.78           2.38
##  2    16 3.78           4.23
##  3    31 2.67          11.6
##  4   -17 3.05          -5.57
##  5   -19 1.93          -9.83
##  6    16 2.5            6.4
##  7    24 2.63           9.11
##  8   -11 0.883        -12.5
##  9    -5 2.33          -2.14
## 10    10 2.3            4.35
## # ... with 336,766 more rows
```

Useful Creation Functions

- There are many functions you can use with `mutate()`.
- Key is that the function must be vectorized: Both input and output should be vectors.
  - Arithmetic Operators
  - Modular Arithmetic
  - Logs
  - Offsets such as `lead()`, `lag()`
  - Cumulative and rolling aggregates
  - Logical Comparisons
  - Ranking

arrange()

- sort() will sort a vector, but not a data frame.
- arrange() is for it.
- Specify the data frame and the column by which you want it to be sorted.

```
arrange(flights_sml, distance)
```

```
## # A tibble: 336,776 x 7
##     year month   day dep_delay arr_delay distance air_time
##    <int> <int> <int>     <dbl>     <dbl>    <dbl>    <dbl>
##  1  2013     7    27        NA        NA       17       NA
##  2  2013     1     3        -2        -2       80       30
##  3  2013     1     4        40        27       80       30
##  4  2013     1     4       134       136       80       28
##  5  2013     1     4        -1        -6       80       32
##  6  2013     1     5        -5       -25       80       29
##  7  2013     1     6        -4         0       80       22
##  8  2013     1     7        -5       -12       80       25
##  9  2013     1     8        -3        39       80       30
## 10  2013     1     9        -3        -7       80       27
## # ... with 336,766 more rows
```

arrange()

- To break ties, we can further sort by additional variables.

```
arrange(flights_sml, distance, year, month, day)
```

```
## # A tibble: 336,776 x 7
##     year month   day dep_delay arr_delay distance air_time
##    <int> <int> <int>     <dbl>     <dbl>    <dbl>    <dbl>
##  1  2013     7    27        NA        NA       17       NA
##  2  2013     1     3        -2        -2       80       30
##  3  2013     1     4        40        27       80       30
##  4  2013     1     4       134       136       80       28
##  5  2013     1     4        -1        -6       80       32
##  6  2013     1     5        -5       -25       80       29
##  7  2013     1     6        -4         0       80       22
##  8  2013     1     7        -5       -12       80       25
##  9  2013     1     8        -3        39       80       30
## 10  2013     1     9        -3        -7       80       27
## # ... with 336,766 more rows
```

Grouped Summaries with summarize()

- summarize() collapses a data frame to a single row.

```
summarize(flights, N = n(), delay = mean(dep_delay, na.rm = TRUE))
```

```
## # A tibble: 1 x 2
##          N delay
##      <int> <dbl>
## 1 336776  12.6
```

- You can specify a list of functions to summarize the observations.

Grouped Summaries with summarize()

- summarize() is much useful with group_by()

```
by_day <- group_by(flights, year, month, day)
```

- by_day is now grouped data frame.
- Following code yields grouped summaries.

```
summarize(by_day, N = n(), delay = mean(dep_delay, na.rm = FALSE))
```

- Together group_by() and summarize() are one the most popular tools in **dplyr**.

Combining Multiple Operations with the Pipe

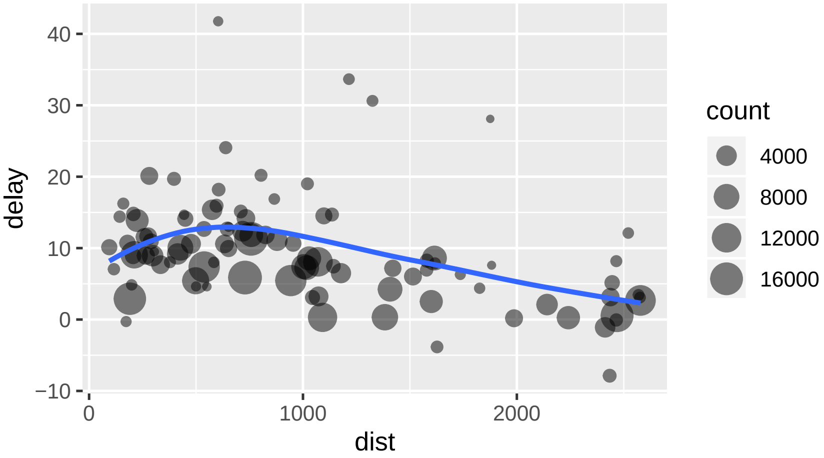
- You want to explore the relationship between the distance and average delay for each location.

```r
by_dest <- group_by(flights, dest)
delay <- summarize(by_dest,
                   count = n(),
                   dist = mean(distance, na.rm = TRUE),
                   delay = mean(arr_delay, na.rm = TRUE)
                   )
delay <- filter(delay, count > 20, dest != "HNL")
```

Combining Multiple Operations with the Pipe

```r
ggplot(data = delay, mapping = aes(x = dist, y = delay)) +
  geom_point(aes(size = count), alpha = 1/2) +
  geom_smooth(se = FALSE)
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Combining Multiple Operations with the Pipe

- Equivalent expression with the pipe %>%.

```
delays <- flights %>%
  group_by(dest) %>%
  summarize(
    count = n(),
    dist = mean(distance, na.rm = TRUE),
    delay = mean(arr_delay, na.rm = TRUE)
  ) %>%
  filter(count > 20, dest !="HNL")
```

Missing Values

```
not_cancelled <- flights %>%
  filter(!is.na(dep_delay), !is.na(arr_delay))
not_cancelled %>%
  group_by(year, month, day) %>%
  summarize(mean = mean(dep_delay))
```

```
## # A tibble: 365 x 4
## # Groups:   year, month [12]
##     year month   day  mean
##    <int> <int> <int> <dbl>
##  1  2013     1     1 11.4
##  2  2013     1     2 13.7
##  3  2013     1     3 10.9
##  4  2013     1     4  8.97
##  5  2013     1     5  5.73
##  6  2013     1     6  7.15
##  7  2013     1     7  5.42
##  8  2013     1     8  2.56
##  9  2013     1     9  2.30
## 10  2013     1    10  2.84
## # ... with 355 more rows
```

Useful Summary Functions (location)

```
not_cancelled %>%
  group_by(year, month, day) %>%
  summarize(
    # averaged delay:
    ave_delay1 = mean(arr_delay),
    # averaged positive delay:
    ave_delay2 = mean(arr_delay[arr_delay > 0])
  )
```

```
## # A tibble: 365 x 5
## # Groups:   year, month [12]
##     year month   day ave_delay1 ave_delay2
##    <int> <int> <int>      <dbl>      <dbl>
##  1  2013     1     1     12.7         32.5
##  2  2013     1     2     12.7         32.0
##  3  2013     1     3      5.73        27.7
##  4  2013     1     4     -1.93        28.3
##  5  2013     1     5     -1.53        22.6
##  6  2013     1     6      4.24        24.4
##  7  2013     1     7     -4.95        27.8
##  8  2013     1     8     -3.23        20.8
##  9  2013     1     9     -0.264       25.6
## 10  2013     1    10     -5.90        27.3
```

Useful Summary Functions (dispersion)

```r
not_cancelled %>%
  group_by(dest) %>%
  summarize(distance_sd = sd(distance)) %>%
  arrange(desc(distance_sd))
```

```
## # A tibble: 104 x 2
##    dest  distance_sd
##    <chr>       <dbl>
##  1 EGE         10.5
##  2 SAN         10.4
##  3 SFO         10.2
##  4 HNL         10.0
##  5 SEA          9.98
##  6 LAS          9.91
##  7 PDX          9.87
##  8 PHX          9.86
##  9 LAX          9.66
## 10 IND          9.46
## # ... with 94 more rows
```

Useful Summary Functions (others)

- rank: `min(x)`, `quantile(x, 0.25)`, `max(x)`
- posistion: `first(x)`, `nth(x, 2)`, `last(x)`
- counts: `n()`, `n_distinct()`
- Others: `sum(x > 10)`, `mean(y == 0)`

Grouping by Multiple Variables

```r
daily <- group_by(flights, year, month, day)
per_day <- summarize(daily, flights = n())
per_day
```

```
## # A tibble: 365 x 4
## # Groups:   year, month [12]
##      year month   day flights
##     <int> <int> <int>   <int>
##  1   2013     1     1     842
##  2   2013     1     2     943
##  3   2013     1     3     914
##  4   2013     1     4     915
##  5   2013     1     5     720
##  6   2013     1     6     832
##  7   2013     1     7     933
##  8   2013     1     8     899
##  9   2013     1     9     902
## 10   2013     1    10     932
## # ... with 355 more rows
```

Ungrouping

```r
daily %>%
  ungroup() %>%
  summarize(flights = n())
```

```
## # A tibble: 1 x 1
##   flights
##     <int>
## 1  336776
```

Grouped Mutates (and Filters)

- Find the worst members of each group

```r
flights_sml %>%
  group_by(year, month, day) %>%
  filter(rank(desc(arr_delay)) < 10)
```

```
## # A tibble: 3,306 x 7
## # Groups:   year, month, day [365]
##     year month   day dep_delay arr_delay distance air_time
##    <int> <int> <int>     <dbl>     <dbl>    <dbl>    <dbl>
##  1  2013     1     1       853       851      184       41
##  2  2013     1     1       290       338     1134      213
##  3  2013     1     1       260       263      266       46
##  4  2013     1     1       157       174      213       60
##  5  2013     1     1       216       222      708      121
##  6  2013     1     1       255       250      589      115
##  7  2013     1     1       285       246     1085      146
##  8  2013     1     1       192       191      199       44
##  9  2013     1     1       379       456     1092      222
## 10  2013     1     2       224       207      550       94
## # ... with 3,296 more rows
```

Grouped Mutates (and Filters)

- Find all groups bigger than a threshold.

```
flights %>%
  group_by(dest) %>%
  filter(n() > 365)
```

```
## # A tibble: 332,577 x 19
## # Groups:   dest [77]
##     year month   day dep_time sched_dep_time dep_delay arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1  2013     1     1      517            515         2      830
##  2  2013     1     1      533            529         4      850
##  3  2013     1     1      542            540         2      923
##  4  2013     1     1      544            545        -1     1004
##  5  2013     1     1      554            600        -6      812
##  6  2013     1     1      554            558        -4      740
##  7  2013     1     1      555            600        -5      913
##  8  2013     1     1      557            600        -3      709
##  9  2013     1     1      557            600        -3      838
## 10  2013     1     1      558            600        -2      753
## # ... with 332,567 more rows, and 12 more variables: sched_arr_time
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <
## #   minute <dbl>, time_hour <dttm>
```

Reference

- Wickham, H. and Grolemund, G. (2017) R for Data Science, O'reilly Media Inc., Chapter 4.