Data Manipulation: Select( )

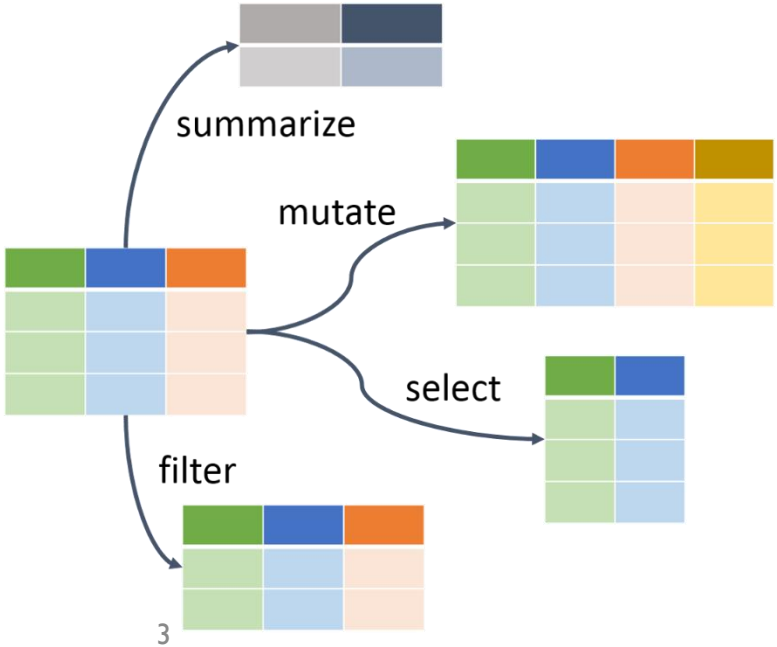
Pilsung Kang
School of Industrial Management Engineering
Korea University

dplyr Package

- The data frame is an important data structure in statistics and in R
  - ✓ The basic structure is one observation per row and each column represents a variable
- We have learned tools like [ ] and $ operator to extract subsets of data frames
- However, the dplyr package allows further operations such as filtering, re-ordering, and collapsing
  - ✓ Everything dplyr does could already be done with base R, but it greatly simplifies existing functionality in R
  - ✓ It makes the data frames management easier

dplyr

- dplyr
  - ✓ A package developed by Hadley Wickham to help transform tabular data
    - Unified, intuitive syntax
    - Fast implementation in C++
    - Support various data backends (dataframe, RDB, etc.)
    - Can work with directly with external DBs: eliminates the limitation that all data must be loaded into working memory



dplyr: Key Verbs

- The key verbs of dplyr
  - ✓ select( ): returns a subset of the columns of a data frame, using a flexible notation
  - ✓ filter( ): extract a subset of rows from a data frame based on logical conditions
  - ✓ arrange( ): reorder rows of a data frame
  - ✓ mutate( ): add new variables/columns or transform existing variables
  - ✓ summarize( ): generate summary statistics of different variables in the data frame
  - ✓ group_by( ): generate summary statistics from the data frame within strata defined by a variable
  - ✓ inner_join( ) and full_join( ): merge or join two data frames

dplyr: Common Properties

- Common dplyr function properties
  - ✓ The first argument is a data frame
  - ✓ The subsequent arguments describe what to do with the data frame specified in the first argument, and you can refer to column names in the data frame directly without using the $ operator
  - ✓ The return result of a function is a new data frame
  - ✓ Data frames must be properly formatted and annotated for this to all be useful. i.e. there should be one observation per row, and one variable per column

dplyr: Simple Example

- Install packages and load "mtcars" dataframe

```
install.packages("dplyr")
library(dplyr)

# load data "mtcars"
data(mtcars)
View(mtcars)
head(mtcars)
```

```
> head(mtcars)
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb              model
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4          Mazda RX4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4      Mazda RX4 Wag
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1         Datsun 710
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1     Hornet 4 Drive
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2 Hornet Sportabout
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1            Valiant
```

dplyr: Simple Example

- Question
  - ✓ What are the car models with fewer than 6 cylinders and a consumption less than 20 miles/gallon?
    - Using which( ) function

```
index <- which(mtcars$cyl <= 6 & mtcars$mpg < 20)
mtcars$model[index]
```

```
> mtcars$model[index]
[1] "Valiant"      "Merc 280"     "Merc 280C"    "Ferrari Dino"
```

    - How we can do the same thing with "dplyr"

```
x <- filter(mtcars, cyl <= 6, mpg < 20)
x
select(x, model)
```

```
> select(x, model)
          model
1       Valiant
2      Merc 280
3     Merc 280C
4 Ferrari Dino
```

dplyr: Simple Example

- Question
  - ✓ What are the car models with fewer than 6 cylinders and a consumption less than 20 miles/gallon?
    - Do the same thing in one line

```
# Do the same thing in one line
select( filter(mtcars, cyl <= 6, mpg < 20), model)
```

```
> select( filter(mtcars, cyl <= 6, mpg < 20), model)
          model
1       Valiant
2     Merc 280
3     Merc 280C
4 Ferrari Dino
```

    - Do the same thing in one line using pipeline

```
# Do the same thing in one line using pipeline
mtcars %>% filter(cyl <= 6, mpg < 20) %>% select(model)
```

dplyr: Tibble

- Dataframe vs. Tibble
  - ✓ Tibble is a more advanced version of data frames
  - ✓ It is particularly useful for large datasets

```
# Dataframes and tibbles
install.packages("hflights")
library(hflights)
str(hflights)
dim(hflights)
hflights # Not recommended
head(hflights) # Recommended
```

```
6345        0                         0
6346        0                         0
6347        0                         0
6348        0                         0
6349        0                         0
6350        0                         0
6351        0                         0
6352        0                         0
6353        0                         0
6354        0                         0
6355        0                         0
6356        0                         0
6357        0                         0
6358        0                         0
 [ reached 'max' / getOption("max.print") -- omitted 227449 rows ]
```

dplyr: Tibble

- Dataframe vs. Tibble

```
# Tibble
hflights2 <- tbl_df(hflights)
hflights2
glimpse(hflights2) # to catch a glimpse
```

```
> hflights2
# A tibble: 227,496 x 21
    Year Month DayofMonth DayOfWeek DepTime ArrTime UniqueCarrier FlightNum TailNum
   <int> <int>      <int>     <int>   <int>   <int> <chr>             <int> <chr>
 1  2011     1          1         6    1400    1500 AA                  428 N576AA
 2  2011     1          2         7    1401    1501 AA                  428 N557AA
 3  2011     1          3         1    1352    1502 AA                  428 N541AA
 4  2011     1          4         2    1403    1513 AA                  428 N403AA
 5  2011     1          5         3    1405    1507 AA                  428 N492AA
 6  2011     1          6         4    1359    1503 AA                  428 N262AA
 7  2011     1          7         5    1359    1509 AA                  428 N493AA
 8  2011     1          8         6    1355    1454 AA                  428 N477AA
 9  2011     1          9         7    1443    1554 AA                  428 N476AA
10  2011     1         10         1    1443    1553 AA                  428 N504AA
# ... with 227,486 more rows, and 12 more variables: ActualElapsedTime <int>,
#   AirTime <int>, ArrDelay <int>, DepDelay <int>, Origin <chr>, Dest <chr>,
#   Distance <int>, TaxiIn <int>, TaxiOut <int>, Cancelled <int>,
#   CancellationCode <chr>, Diverted <int>
```

dplyr: Select( )

- Select( )
  - ✓ Selects columns from a data frame
  - ✓ Arguments
    - Data frame
    - The columns you would like to keep
  - ✓ Example: select(surveys, plot_id, species_id, weight)

dplyr: Select( )

- Select( )

```
# Select()
select(hflights2, Origin, Dest)
# Note that select() does not change the data frame it is called on:
dim(hflights2)
orig_dest <- select(hflights2, Origin, Dest)
dim(orig_dest)
```

```
> select(hflights2, Origin, Dest)
# A tibble: 227,496 x 2
   Origin Dest
   <chr>  <chr>
 1 IAH    DFW
 2 IAH    DFW
 3 IAH    DFW
 4 IAH    DFW
 5 IAH    DFW
 6 IAH    DFW
 7 IAH    DFW
 8 IAH    DFW
 9 IAH    DFW
10 IAH    DFW
# ... with 227,486 more rows
> # Note that select() does not change the data frame it is called on:
> dim(hflights2)
[1] 227496     21
> orig_dest <- select(hflights2, Origin, Dest)
> dim(orig_dest)
[1] 227496      2
```

dplyr: Select( )

- Select( )
  - ✓ Drop variables

```
# Select(): drop operator
colnames(hflights2)
drop_hflights2 <- select(hflights2, -c("Year", "Month", "UniqueCarrier"))
drop_hflights2
```

```
> colnames(hflights2)
 [1] "Year"              "Month"             "DayofMonth"        "DayOfWeek"         "DepTime"
 [6] "ArrTime"           "UniqueCarrier"     "FlightNum"         "TailNum"           "ActualElapsedTime"
[11] "AirTime"           "ArrDelay"          "DepDelay"          "Origin"            "Dest"
[16] "Distance"          "TaxiIn"            "TaxiOut"           "Cancelled"         "CancellationCode"
[21] "Diverted"
> drop_hflights2 <- select(hflights2, -c("Year", "Month","UniqueCarrier"))
> drop_hflights2
# A tibble: 227,496 x 18
   DayofMonth DayOfWeek DepTime ArrTime FlightNum TailNum ActualElapsedTi~ AirTime ArrDelay DepDelay Origin Dest
        <int>     <int>   <int>   <int>     <int> <chr>              <int>   <int>    <int>    <int> <chr>  <chr>
 1          1         6    1400    1500       428 N576AA                60      40      -10        0 IAH    DFW
 2          2         7    1401    1501       428 N557AA                60      45       -9        1 IAH    DFW
 3          3         1    1352    1502       428 N541AA                70      48       -8       -8 IAH    DFW
 4          4         2    1403    1513       428 N403AA                70      39        3        3 IAH    DFW
 5          5         3    1405    1507       428 N492AA                62      44       -3        5 IAH    DFW
 6          6         4    1359    1503       428 N262AA                64      45       -7       -1 IAH    DFW
 7          7         5    1359    1509       428 N493AA                70      43       -1       -1 IAH    DFW
 8          8         6    1355    1454       428 N477AA                59      40      -16       -5 IAH    DFW
 9          9         7    1443    1554       428 N476AA                71      41       44       43 IAH    DFW
10         10         1    1443    1553       428 N504AA                70      45       43       43 IAH    DFW
# ... with 227,486 more rows, and 6 more variables: Distance <int>, TaxiIn <int>, TaxiOut <int>, Cancelled <int>,
#   CancellationCode <chr>, Diverted <int>
```

dplyr: Select( )

- Helper functions
  - ✓ select( ) is often used in combination with very flexible *helper* functions that help to identify the variables of interest
  - ✓ dplyer provides 6 helper functions, each of which only works when used inside select ( )

- `starts_with("X")` : every name that starts with `"X"` ,
- `ends_with("X")` : every name that ends with `"X"` ,
- `contains("X")` : every name that contains `"X"` ,
- `matches("X")` : every name that matches `"X"` , which can be a regular expression,
- `num_range("x", 1:5)` : the variables named `x01` , `x02` , `x03` , `x04` and `x05` ,
- `one_of(x)` : every name that appears in `x` , which should be a character vector.

dplyr: Select( )

- Helper functions
  - ✓ Select variable names based on patterns
    - Select the variables starting with "D"

```
# Select(): select variable names based on patterns
colnames(hflights2)
# Let's select the variables starting with "D"
select(hflights2, starts_with("D"))
```

```
> colnames(hflights2)
 [1] "Year"              "Month"             "DayofMonth"        "DayOfWeek"
 [5] "DepTime"           "ArrTime"           "UniqueCarrier"     "FlightNum"
 [9] "TailNum"           "ActualElapsedTime" "AirTime"           "ArrDelay"
[13] "DepDelay"          "Origin"            "Dest"              "Distance"
[17] "TaxiIn"            "TaxiOut"           "Cancelled"         "CancellationCode"
[21] "Diverted"
```

dplyr: Select( )

- Helper functions
  - ✓ Select variable names based on patterns
    - Select the variables starting with "D"

```
# Select(): select variable names based on patterns
colnames(hflights2)
# Let's select the variables starting with "D"
select(hflights2, starts_with("D"))
```

```
> select(hflights2, starts_with("D"))
# A tibble: 227,496 x 7
   DayofMonth DayOfWeek DepTime DepDelay Dest  Distance Diverted
        <int>     <int>   <int>    <int> <chr>    <int>    <int>
 1          1         6    1400        0 DFW        224        0
 2          2         7    1401        1 DFW        224        0
 3          3         1    1352       -8 DFW        224        0
 4          4         2    1403        3 DFW        224        0
 5          5         3    1405        5 DFW        224        0
 6          6         4    1359       -1 DFW        224        0
 7          7         5    1359       -1 DFW        224        0
 8          8         6    1355       -5 DFW        224        0
 9          9         7    1443       43 DFW        224        0
10         10         1    1443       43 DFW        224        0
# ... with 227,486 more rows
```

dplyr: Select( )

- Helper functions
  - ✓ Select variable names based on patterns
    - Select the variables ending with "e"

```
# Let's select the variables ending with "e"
select(hflights2, ends_with("e"))
```

```
> select(hflights2, ends_with("e"))
# A tibble: 227,496 x 6
   DepTime ArrTime ActualElapsedTime AirTime Distance CancellationCode
     <int>   <int>             <int>   <int>    <int> <chr>
 1    1400    1500                60      40      224 ""
 2    1401    1501                60      45      224 ""
 3    1352    1502                70      48      224 ""
 4    1403    1513                70      39      224 ""
 5    1405    1507                62      44      224 ""
 6    1359    1503                64      45      224 ""
 7    1359    1509                70      43      224 ""
 8    1355    1454                59      40      224 ""
 9    1443    1554                71      41      224 ""
10    1443    1553                70      45      224 ""
# ... with 227,486 more rows
```

dplyr: Select( )

- Helper functions
  - ✓ We select the variable FlightNum together with the variables DepTime, AirTime, ActualElapsedTime and ArrTime,

```
# Let's select the variables ending with "Time"
select(hflights2, ends_with("Time"))
```

```
> select(hflights2, ends_with("Time"))
# A tibble: 227,496 x 4
   DepTime ArrTime ActualElapsedTime AirTime
     <int>   <int>             <int>   <int>
 1    1400    1500                60      40
 2    1401    1501                60      45
 3    1352    1502                70      48
 4    1403    1513                70      39
 5    1405    1507                62      44
 6    1359    1503                64      45
 7    1359    1509                70      43
 8    1355    1454                59      40
 9    1443    1554                71      41
10    1443    1553                70      45
# ... with 227,486 more rows
```

dplyr: Select( )

- Helper functions
  - ✓ Select variable names based on patterns
    - Select the variables containing "n"

```
# Let's select the variables containing "n"
select(hflights2, contains("n"))
```

```
> select(hflights2, contains("n"))
# A tibble: 227,496 x 10
   Month DayofMonth UniqueCarrier FlightNum TailNum Origin Distance TaxiIn Cancelled CancellationCode
   <int>      <int> <chr>             <int> <chr>   <chr>      <int>  <int>     <int> <chr>
 1     1          1 AA                  428 N576AA  IAH          224      7         0 ""
 2     1          2 AA                  428 N557AA  IAH          224      6         0 ""
 3     1          3 AA                  428 N541AA  IAH          224      5         0 ""
 4     1          4 AA                  428 N403AA  IAH          224      9         0 ""
 5     1          5 AA                  428 N492AA  IAH          224      9         0 ""
 6     1          6 AA                  428 N262AA  IAH          224      6         0 ""
 7     1          7 AA                  428 N493AA  IAH          224     12         0 ""
 8     1          8 AA                  428 N477AA  IAH          224      7         0 ""
 9     1          9 AA                  428 N476AA  IAH          224      8         0 ""
10     1         10 AA                  428 N504AA  IAH          224      6         0 ""
# ... with 227,486 more rows
```

dplyr: Select( )

- Helper functions
  - ✓ Select variable names based on patterns
    - Select the variables with certain names if they exist

```
# Let's select the variables with certain names if they exist
select(hflights2, FlightNum, Distance, Cancelled, Pilsung)

# Let's select the variables with certain names if they exist
select(hflights2, one_of(c("FlightNum", "Distance", "Cancelled", "Pilsung")))
```

```
> select(hflights2, FlightNum, Distance, Cancelled, Pilsung)
Error in .f(.x[[i]], ...) : object 'Pilsung' not found
```

dplyr: Select( )

- Helper functions
  - ✓ Select variable names based on patterns
    - Select the variables with certain names if they exist

```
# Let's select the variables with certain names if they exist
select(hflights2, FlightNum, Distance, Cancelled, Pilsung)

# Let's select the variables with certain names if they exist
select(hflights2, one_of(c("FlightNum", "Distance", "Cancelled", "Pilsung")))
```

```
> select(hflights2, one_of(c("FlightNum", "Distance", "Cancelled", "Pilsung")))
# A tibble: 227,496 x 3
   FlightNum Distance Cancelled
       <int>    <int>     <int>
 1       428      224         0
 2       428      224         0
 3       428      224         0
 4       428      224         0
 5       428      224         0
 6       428      224         0
 7       428      224         0
 8       428      224         0
 9       428      224         0
10       428      224         0
# ... with 227,486 more rows
Warning message:
Unknown columns: `Pilsung`
```

dplyr: Pipe Operator %>%

- Pipe operator %>%
  - ✓ Allows you to combine multiple "verb" operations
  - ✓ Syntax: %>% at the end of the line
  - ✓ Output of the first line becomes the input of next line
  - ✓ Final output to the screen or a variable
  - ✓ Example: surveys %>%

```
filter(weight < 5) %>%
select(species_id, sex, weight)
```

dplyr: Pipe Operator %>%

- Pipe operator %>%
  - ✓ We are interested in the number of different destinations of flights departing from Houston
    - We can use unique() to eliminates multiple values in Dest and then nrow() to compute the number of (now distinct) observations in the resulting column:

```
# Pipe operator %>%
nrow(unique(select(hflights2, Dest)))
```

```
> nrow(unique(select(hflights2, Dest)))
[1] 116
```

  - ✓ This is not very easy to read nor to write.

dplyr: Pipe Operator %>%

- Pipe operator %>%
  - ✓ Instead we can use the pipe operator %>% to concatenate the different steps of our analysis into a pipeline
    - we take hflights, then we select Dest, then we take its values without considering repetitions, and at last we count the number of resulting values:

```
# With pipe operator
hflights2 %>% select(Dest) %>% unique %>% nrow()

# With pipe operator and n_distinct() function
hflights2 %>% select(Dest) %>% n_distinct()
```

```
> hflights2 %>% select(Dest) %>% unique %>% nrow()
[1] 116
> # With pipe operator and n_distnct() function
> hflights2 %>% select(Dest) %>% n_distinct()
[1] 116
```

dplyr: Pipe Operator %>%

- Pipe operator %>%
  - ✓ The %>% operators passes the object on the left to the first argument of the function on the right:

```
x %>% f(y) gives f(x,y)
```

  - ✓ This corresponds to our way of thinking and makes it possible to code in a progressively and more readable fashion
  - ✓ In other words, when coding we do not have to start from the last function and then go backward, as we would normally do using basic R
  - ✓ Instead we are now free to build our sequence of instructions from the very first object, that is data.
  - ✓ This approach is much more flexible and allows to change very quickly our queries to explore data.

dplyr: Pipe Operator %>%

- Pipe operator %>%
  - ✓ The %>% operator can also be used to pass the object on the left to any argument of the function on the right, not only the first one.
  - ✓ In this case, the argument position is to be indicated with the placeholder .

`x %>% f(y, .) gives f(y,x)`

```
# Placeholder (.) example
ratio <- function(x,y) x/y
1 %>% ratio(2)
2 %>% ratio(1, .)
```

```
> 1 %>% ratio(2)
[1] 0.5
> 2 %>% ratio(1, .)
[1] 0.5
```


ANY
questions?