Data Manipulation with dplyr

Pilsung Kang

School of Industrial Management Engineering

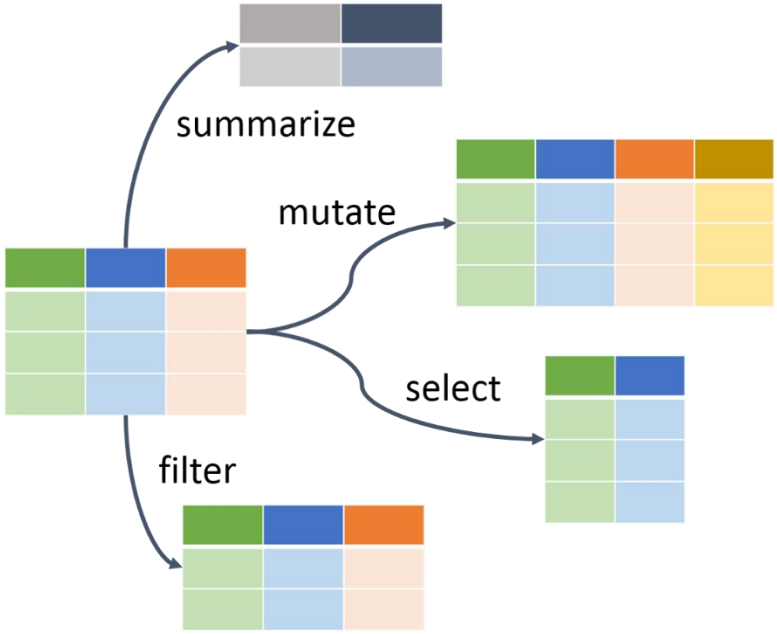
Korea University

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
```

dplyr: Filter( )

- Filter( )
  - ✓ Choose rows based on a specific criterion
  - ✓ Arguments:
    - Data frame
    - Relational expressions (returns true/false)
      - `x > y` is `TRUE` if `x` is greater than `y`
      - `x >= y` is `TRUE` if `x` is greater or equal than `y`
      - `x == y` is `TRUE` if `x` is equal to `y`
      - `is.na(x)` is `TRUE` if `x` is `NA`. **Warning:** never use `x == NA` to test if `x` is `NA`.
      - `x %in% c('a', 'b', 'c')` is `TRUE` if `x` is in the vector `c('a', 'b', 'c')`.
      - `!x` is `TRUE` if `x` is `FALSE` and viceversa.
  - ✓ Example: filter(surveys, year == 1995)

dplyr: Filter( )

- Filter( )
  - ✓ We keep only observations with arrival delay greater than 10 hours:

```
# We keep only observations with arrival delay greater than 10 hours:
delayed <- hflights2 %>% filter(ArrDelay > 600)
View(delayed)
```

Filter

	Year	Month	DayofMonth	DayOfWeek	DepTime	ArrTime	UniqueCarrier	FlightNum	TailNum	ActualElapsedTime	AirTime	ArrDelay	DepDelay
1	2011	1	20	4	635	807	CO	59	N74856	152	126	775	780
2	2011	5	20	5	858	1027	MQ	3328	N609MQ	89	55	822	803
3	2011	6	22	3	908	1040	CO	595	N75861	212	177	766	758
4	2011	6	21	2	2334	124	UA	855	N670UA	230	216	861	869
5	2011	6	9	4	2029	2243	MQ	3859	N6EAMQ	134	117	793	814
6	2011	8	1	1	156	452	CO	1	N69063	476	461	957	981
7	2011	10	25	2	2310	149	DL	1215	N764NC	99	92	701	730
8	2011	11	19	6	1752	1910	AA	1903	N495AA	78	40	685	677
9	2011	11	8	2	721	948	MQ	3786	N502MQ	147	120	918	931
10	2011	12	12	1	650	808	AA	1740	N473AA	78	49	978	970
11	2011	12	22	4	1728	1848	AA	1903	N580AA	80	40	663	653
12	2011	12	13	2	706	824	MQ	3328	N651MQ	78	56	704	691
13	2011	12	29	4	1928	2114	XE	4309	N16170	166	150	634	628

dplyr: Filter( )

- Filter( )
  - ✓ All flights flown by one of AA, FL, or XE

```
# Filter: All flights flown by one of AA, FL, or XE:
filter2 <- hflights2 %>% filter(UniqueCarrier %in% c("AA", "FL", "XE"))
table(UniqueCarrier)
```

```
> table(filter2$UniqueCarrier)

   AA    FL    XE
 3244  2139 73053
```

dplyr: Filter( )

- Filter( )
  - ✓ All flights where taxiing took longer than flying

```
# All flights where taxiing took longer than flying
filter3 <- hflights2 %>% filter(TaxiIn + TaxiOut > AirTime)
filter3[,c("TaxiIn", "TaxiOut", "AirTime")]
```

```
> filter3[,c("TaxiIn", "TaxiOut", "AirTime")]
# A tibble: 1,389 x 3
   TaxiIn TaxiOut AirTime
    <int>   <int>   <int>
 1     14      37      42
 2     10      40      43
 3     10      35      43
 4     27      20      45
 5      5      23      27
 6      7      25      30
 7      5      30      30
 8      5      29      32
 9      6      27      31
10     10      34      40
# ... with 1,379 more rows
```

dplyr: Filter( )

- Filter( )
  - ✓ Combining tests using boolean operators

```
# Combining tests using boolean operators
# all flights that departed before 5am or arrived after 10pm.
filter4 <- filter(hflights2, DepTime < 500 | ArrTime > 2200)
filter4[,1:7]
```

```
> filter4[,1:7]
# A tibble: 27,799 x 7
    Year Month DayofMonth DayOfWeek DepTime ArrTime UniqueCarrier
   <int> <int>      <int>     <int>   <int>   <int> <chr>
 1  2011     1          4         2    2100    2207 AA
 2  2011     1         14         5    2119    2229 AA
 3  2011     1         10         1    1934    2235 AA
 4  2011     1         26         3    1905    2211 AA
 5  2011     1         30         7    1856    2209 AA
 6  2011     1          9         7    1938    2228 AS
 7  2011     1         31         1    1919    2231 CO
 8  2011     1         31         1    2116    2344 CO
 9  2011     1         31         1    1850    2211 CO
10  2011     1         31         1    2102    2216 CO
# ... with 27,789 more rows
```

dplyr: Filter( )

- Filter( )
  - ✓ Combining tests using boolean operators

```
# all flights that departed late but arrived ahead of schedule
filter5 <- filter(hflights2, DepDelay > 0, ArrDelay < 0)
filter5[,11:15]
```

```
> filter5[,11:15]
# A tibble: 27,712 x 5
   AirTime ArrDelay DepDelay Origin Dest
     <int>    <int>    <int> <chr>  <chr>
 1      45       -9        1 IAH    DFW
 2      44       -3        5 IAH    DFW
 3      42       -2        8 IAH    DFW
 4      46       -8        1 IAH    DFW
 5      39       -7       10 IAH    DFW
 6      44       -4       15 IAH    DFW
 7      39      -17        4 IAH    DFW
 8      37       -9        1 IAH    DFW
 9      41       -5        9 IAH    DFW
10      44       -6        1 IAH    DFW
# ... with 27,702 more rows
```

dplyr: Filter( )

- Filter( )
  - ✓ Combining tests using boolean operators

```
# all cancelled weekend flights
filter6 <- filter(hflights2, DayOfWeek %in% c(6,7), Cancelled == 1)
filter6[,c(1:4,18:21)]
```

```
> filter6[,c(1:4,18:21)]
# A tibble: 585 x 8
    Year Month DayofMonth DayOfWeek TaxiOut Cancelled CancellationCode Diverted
   <int> <int>      <int>     <int>   <int>     <int> <chr>                <int>
 1  2011     1          9         7      NA         1 B                        0
 2  2011     1         29         6      NA         1 A                        0
 3  2011     1          9         7      NA         1 B                        0
 4  2011     1          9         7      NA         1 B                        0
 5  2011     1          9         7      NA         1 B                        0
 6  2011     1          2         7      NA         1 A                        0
 7  2011     1         29         6      NA         1 A                        0
 8  2011     1          9         7      NA         1 A                        0
 9  2011     1          1         6      NA         1 A                        0
10  2011     1          9         7      NA         1 B                        0
# ... with 575 more rows
```

dplyr: Filter( )

- Filter( )
  - ✓ Combining tests using boolean operators

```
# all flights that were cancelled after being delayed
filter7 <- filter(hflights2, Cancelled == 1, DepDelay > 0)
filter6[,c(1:4,13,19)]
```

```
> filter7[,c(1:4,13,19)]
# A tibble: 40 x 6
    Year Month DayofMonth DayOfWeek DepDelay Cancelled
   <int> <int>      <int>     <int>    <int>     <int>
 1  2011     1         26         3       26         1
 2  2011     1         11         2      135         1
 3  2011     1         19         3        6         1
 4  2011     1          7         5       73         1
 5  2011     2          4         5        8         1
 6  2011     2          8         2      187         1
 7  2011     2          2         3        2         1
 8  2011     2          9         3        4         1
 9  2011     2          1         2       28         1
10  2011     3         31         4      156         1
# ... with 30 more rows
```

dplyr: Arrange( )

- Arrange( )
  - ✓ Q) For all flights with arrival delay greater than 10 hours, give the variables Year, Month,bDayofMonth, UniqueCarrier, FlightNum and ArrDelay
  - ✓ Sort the observations in the result according to variable ArrDelay

```
# filter, select, and arrange
hflights2 %>% filter(ArrDelay > 600) %>%
    select(Year, Month,DayofMonth, UniqueCarrier, FlightNum, ArrDelay) %>%
    arrange(ArrDelay)

hflights2 %>% filter(ArrDelay > 600) %>%
    select(Year, Month,DayofMonth, UniqueCarrier, FlightNum, ArrDelay) %>%
    arrange(desc(ArrDelay))
```

dplyr:Arrange( )

- Arrange( )
  - ✓Arranged in an ascending order

```
> hflights2 %>% filter(ArrDelay > 600) %>%
+   select(Year, Month,DayofMonth, UniqueCarrier, FlightNum, ArrDelay) %>%
+   arrange(ArrDelay)
# A tibble: 13 x 6
    Year Month DayofMonth UniqueCarrier FlightNum ArrDelay
   <int> <int>      <int> <chr>             <int>    <int>
 1  2011    12         29 XE                4309      634
 2  2011    12         22 AA                1903      663
 3  2011    11         19 AA                1903      685
 4  2011    10         25 DL                1215      701
 5  2011    12         13 MQ                3328      704
 6  2011     6         22 CO                 595      766
 7  2011     1         20 CO                  59      775
 8  2011     6          9 MQ                3859      793
 9  2011     5         20 MQ                3328      822
10  2011     6         21 UA                 855      861
11  2011    11          8 MQ                3786      918
12  2011     8          1 CO                   1      957
13  2011    12         12 AA                1740      978
```

dplyr:Arrange( )

- Arrange( )
  - ✓Arranged in a descending order

```
> hflights2 %>% filter(ArrDelay > 600) %>%
+   select(Year, Month,DayofMonth, UniqueCarrier, FlightNum, ArrDelay) %>%
+   arrange(desc(ArrDelay))
# A tibble: 13 x 6
    Year Month DayofMonth UniqueCarrier FlightNum ArrDelay
   <int> <int>      <int> <chr>             <int>    <int>
 1  2011    12         12 AA                1740      978
 2  2011     8          1 CO                   1      957
 3  2011    11          8 MQ                3786      918
 4  2011     6         21 UA                 855      861
 5  2011     5         20 MQ                3328      822
 6  2011     6          9 MQ                3859      793
 7  2011     1         20 CO                  59      775
 8  2011     6         22 CO                 595      766
 9  2011    12         13 MQ                3328      704
10  2011    10         25 DL                1215      701
11  2011    11         19 AA                1903      685
12  2011    12         22 AA                1903      663
13  2011    12         29 XE                4309      634
```

dplyr:Arrange( )

- Arrange( )
  - ✓Arrange with more than two variables

```
# Arrange with more than two variables
arrange1 <- arrange(hflights2, UniqueCarrier, DepDelay)
arrange1[,c(1:4,7,13)]
```

```
> arrange1[,c(1:4,7,13)]
# A tibble: 227,496 x 6
    Year Month DayofMonth DayOfWeek UniqueCarrier DepDelay
   <int> <int>      <int>     <int> <chr>             <int>
 1  2011     2         13         7 AA                  -15
 2  2011    10          5         3 AA                  -15
 3  2011    11         24         4 AA                  -15
 4  2011     2          6         7 AA                  -14
 5  2011    12          5         1 AA                  -14
 6  2011     5          7         6 AA                  -13
 7  2011     6          1         3 AA                  -13
 8  2011     8         13         6 AA                  -13
 9  2011    11         25         5 AA                  -13
10  2011     1         11         2 AA                  -12
# ... with 227,486 more rows
```

dplyr: Mutate( )

- Mutate( )
  - ✓ Create a new column, assigns a value
  - ✓ Arguments:
    - Data frame
    - Name of new column = value
  - ✓ Example: mutate(surveys, weight_kg = weight/1000)
  - ✓ Imagine to have a data frame df with three columns: Id (the identifier), w (weight in Kg) and h (height in m)
  - ✓ We want to create a fourth variable bmi with the Body Mass Index: bmi = w/h^2. This can be easily done with the mutate() function:

```
mutate(df, bmi = w/h^2)
```

dplyr: Mutate( )

- Mutate( )
  - ✓ Similarly, we create a new variable TotalTime measuring the total flight time, as the sum of TaxiIn (time spent on ground before taking off), TaxiOut (ground time after landing) and AirTime:

```
# Mutate example
mutate1 <- hflights2 %>% mutate(TotalTime = TaxiIn + AirTime + TaxiOut)

# Compare with the original value
mutate1 %>% select(TotalTime, ActualElapsedTime) %>% head
```

```
> mutate1 %>% select(TotalTime, ActualElapsedTime) %>% head
# A tibble: 6 x 2
  TotalTime ActualElapsedTime
      <int>             <int>
1        60                60
2        60                60
3        70                70
4        70                70
5        62                62
6        64                64
```

dplyr: Mutate( )

- Mutate( )
  - ✓ Add multiple variables using mutate

```
# Add multiple variables
mutate2 <- mutate(hflights,
                  loss = ArrDelay - DepDelay,
                  loss_percent = (ArrDelay - DepDelay)/DepDelay * 100)
glimpse(mutate2)
```

```
> glimpse(mutate2)
Observations: 227,496
Variables: 23
$ Year              <int> 2011, 2011, 2011, 2011, 2011, 2011, 2011, 2011, 2011, 2011, 2011, 2011...
$ Month             <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1...
$ DayofMonth        <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,...
$ DayOfWeek         <int> 6, 7, 1, 2, 3, 4, 5, 6, 7, 1, 2, 3, 4, 5, 6, 7, 1, 2, 3, 4, 5, 6, 7, 1...
$ DepTime           <int> 1400, 1401, 1352, 1403, 1405, 1359, 1359, 1355, 1443, 1443, 1429, 1419...
$ ArrTime           <int> 1500, 1501, 1502, 1513, 1507, 1503, 1509, 1454, 1554, 1553, 1539, 1515...
$ UniqueCarrier     <chr> "AA", "AA", "AA", "AA", "AA", "AA", "AA", "AA", "AA", "AA", "AA", "AA"...
$ FlightNum         <int> 428, 428, 428, 428, 428, 428, 428, 428, 428, 428, 428, 428, 428, 428, ...
$ TailNum           <chr> "N576AA", "N557AA", "N541AA", "N403AA", "N492AA", "N262AA", "N493AA", ...
$ ActualElapsedTime <int> 60, 60, 70, 70, 62, 64, 70, 59, 71, 70, 70, 56, 63, 67, 60, 70, 64, 60...
$ AirTime           <int> 40, 45, 48, 39, 44, 45, 43, 40, 41, 45, 42, 41, 44, 47, 44, 41, 48, 42...
$ ArrDelay          <int> -10, -9, -8, 3, -3, -7, -1, -16, 44, 43, 29, 5, -9, -6, -11, -1, 84, -...
$ DepDelay          <int> 0, 1, -8, 3, 5, -1, -1, -5, 43, 43, 29, 19, -2, -3, -1, -1, 90, 8, -4,...
$ Origin            <chr> "IAH", "IAH", "IAH", "IAH", "IAH", "IAH", "IAH", "IAH", "IAH", "IAH", ...
$ Dest              <chr> "DFW", "DFW", "DFW", "DFW", "DFW", "DFW", "DFW", "DFW", "DFW", "DFW", ...
$ Distance          <int> 224, 224, 224, 224, 224, 224, 224, 224, 224, 224, 224, 224, 224, 224, ...
$ TaxiIn            <int> 7, 6, 5, 9, 9, 6, 12, 7, 8, 6, 8, 4, 6, 5, 6, 12, 8, 7, 10, 9, 6, 9, 7...
$ TaxiOut           <int> 13, 9, 17, 22, 9, 13, 15, 12, 22, 19, 20, 11, 13, 15, 10, 17, 8, 11, 1...
$ Cancelled         <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
$ CancellationCode  <chr> "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", ""...
$ Diverted          <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
$ loss              <int> -10, -10, 0, 0, -8, -6, 0, -11, 1, 0, 0, -14, -7, -3, -10, 0, -6, -10,...
$ loss_percent      <dbl> -Inf, -1000.000000, 0.000000, 0.000000, -160.000000, 600.000000, 0.000...
```

dplyr: Summarize( )

- Summarize( )
  - ✓ Applies a function to variable
  - ✓ Arguments
    - Data frame
    - Definition of a summary statistic
  - ✓ Example: summarize(data*, mean_weight = mean(weight))
    - data* must be a tibble
  - ✓ Creating summary statistics from a complex data set is obviously a crucial task in data analysis
  - ✓ In dplyr this is done with the function summarise() that creates a new data frame with a single row with statistics
  - ✓ The syntax is the same as mutate:

```
summarise(df, AverageBmi = mean(bmi))
```

dplyr: Summarize( )

- Summarize( )
  - ✓ Determine the shortest and longest distance flown and save statistics to min_dist and max_dist
  - ✓ Determine the longest distance for diverted flights, save statistic to max_div

```
# Determine the shortest and longest distance flown and save statistics to
min_dist and max_dist
summarize1 <- summarize(hflights, min_dist = min(Distance),
                        max_dist = max(Distance))
summarize1
# Determine the longest distance for diverted flights, save statistic to
max_div
summarize2 <- summarize(filter(hflights, Diverted==1),
                        max_div = max(Distance))
summarize2
```

```
> summarize1
  min_dist max_dist
1       79     3904
```

```
> summarize2
  max_div
1    3904
```

dplyr: Summarize( )

- Summarize( )
  - ✓ Aggregate functions
    - We can use any function so long as the function can take a vector of data and return a single number

      - `min(x)` - minimum value of vector `x`.
      - `max(x)` - maximum value of vector `x`.
      - `mean(x)` - mean value of vector `x`.
      - `median(x)` - median value of vector `x`.
      - `quantile(x, p)` - pth quantile of vector `x`.
      - `sd(x)` - standard deviation of vector `x`.
      - `var(x)` - variance of vector `x`.
      - `IQR(x)` - Inter Quartile Range (IQR) of vector `x`.
      - `diff(range(x))` - total range of vector `x`.

dplyr: Summarize( )

- Summarize( )
  - ✓ Aggregate functions

```
# Aggregate functions in basic R
agg1 <- filter(hflights2, !is.na(ArrDelay))
agg2 <- summarize(agg1,
                  earliest = min(ArrDelay),
                  average = mean(ArrDelay),
                  latest = max(ArrDelay),
                  sd = sd(ArrDelay))
agg2
```

```
> agg2
# A tibble: 1 x 4
  earliest average latest    sd
     <int>   <dbl>  <int> <dbl>
1      -70    7.09    978  30.7
```

dplyr: Summarize( )

- Summarize( )
  - ✓ Aggregate functions
    - dplyr provides several helpful aggregate functions of its own, in addition to the ones that are already defined in R

- `first(x)` - The first element of vector `x`.
- `last(x)` - The last element of vector `x`.
- `nth(x, n)` - The nth element of vector `x`.
- `n()` - The number of rows in the data.frame or group of observations that summarise() describes.
- `n_distinct(x)` - The number of unique values in vector `x`.

dplyr: Summarize( )

- Summarize( )
  - ✓ Aggregate functions: dplyr provides several helpful aggregate functions of its own, in addition to the ones that are already defined in R

```
# Additional aggregate functions provided by dplyr
agg3 <- summarise(hflights2, n_obs = n(),
                  n_carrier = n_distinct(UniqueCarrier),
                  n_dest = n_distinct(Dest),
                  dest100 = nth(Dest, 100))
agg3
```

```
> agg3
# A tibble: 1 x 4
   n_obs n_carrier n_dest dest100
   <int>     <int>  <int> <chr>
1 227496        15    116 DFW
```

dplyr: Summarize( )

- Summarize( )
  - ✓ Aggregate functions: dplyr provides several helpful aggregate functions of its own, in addition to the ones that are already defined in R

```
# Calculate the summarizing statistics for flights flown by American
Airlines (carrier code AA)
AA <- filter(hflights2, UniqueCarrier == "AA")
agg4 <- summarise(AA, n_flights = n(),
                      n_canc = sum(Cancelled == 1),
                      p_canc = mean(Cancelled == 1) * 100,
                      avg_delay = mean(ArrDelay, na.rm = TRUE))
agg4
```

```
> agg4
# A tibble: 1 x 4
  n_flights n_canc p_canc avg_delay
      <int>  <int>  <dbl>     <dbl>
1      3244     60   1.85     0.892
```

dplyr: Group by( )

- Group by( )
  - ✓ Groups data in the table by an attribute
  - ✓ Arguments
    - Data frame
    - Factor variable to group by
  - ✓ Example: group_by(surveys, sex)
  - ✓ Very often we are interested in computing summary statistics for each value of a given variable
  - ✓ For instance, we might want to compute the average bmi separately for men and women or for each age category
  - ✓ In this case we can use group_by() to create a *grouped* data frame in which any following operations will be done accordingly *by group*:

```
group_by(df, sex) %>% summarise(mean(bmi))
```

dplyr: Group by( )

- Group by( )
  - ✓ The average departure and arrival delays for each day of the week

```
# The average departure and arrival delays for each day of the week
hflights2 %>% group_by(DayOfWeek) %>%
    summarise(AverageArrDelay = mean(ArrDelay, na.rm = TRUE),
              AverageDepDelay = mean(DepDelay, na.rm = TRUE))
```

```
> hflights2 %>% group_by(DayOfWeek) %>%
+   summarise(AverageArrDelay = mean(ArrDelay, na.rm = TRUE),
+             AverageDepDelay = mean(DepDelay, na.rm = TRUE))
# A tibble: 7 x 3
  DayOfWeek AverageArrDelay AverageDepDelay
      <int>           <dbl>           <dbl>
1         1            8.26            10.0
2         2            5.55             7.59
3         3            5.53             8.08
4         4            9.80            12.4
5         5            7.29             9.88
6         6            5.75             7.77
7         7            6.95             9.78
```

dplyr: Group by( )

- Group by( )
  - ✓ The average departure and arrival delays for each day of the week
  - ✓ With basic R syntax without dplyr

```
# Note how more immediate is feels compared to basic R
AverageArrDelay <- tapply(hflights$ArrDelay, hflights$DayOfWeek,
                          mean, na.rm = TRUE)
AverageDepDelay <- tapply(hflights$DepDelay, hflights$DayOfWeek,
                          mean, na.rm = TRUE)
cbind(sort(unique(hflights$DayOfWeek)), AverageArrDelay, AverageDepDelay)
```

```
> cbind(sort(unique(hflights$DayOfWeek)), AverageArrDelay, AverageDepDelay)
  AverageArrDelay AverageDepDelay
1 1        8.255831       10.025682
2 2        5.551781        7.591971
3 3        5.533013        8.083891
4 4        9.797332       12.404041
5 5        7.291188        9.877408
6 6        5.746582        7.772742
7 7        6.950572        9.777305
```

dplyr: Group by( )

- Group by( )
  - ✓ We rank airline companies according to their average departure delay

```
hflights2 %>% filter(!is.na(DepDelay), DepDelay > 0) %>%
    # we keep only flights with a departure delay
    group_by(UniqueCarrier) %>%
    summarise(avg = mean(DepDelay)) %>%
    # average departure delay for each company
    mutate(rank = rank(avg)) %>%
    arrange(rank)
```

```
# A tibble: 15 x 3
   UniqueCarrier   avg  rank
   <chr>         <dbl> <dbl>
 1 CO             17.9     1
 2 AS             20.8     2
 3 WN             21.9     3
 4 F9             22.7     4
 5 YV             24.5     5
 6 OO             24.6     6
 7 AA             24.7     7
 8 US             26.5     8
 9 XE             26.9     9
10 UA             28.8    10
11 DL             32.4    11
12 FL             33.4    12
13 MQ             37.9    13
14 B6             43.5    14
15 EV             49.3    15
```

dplyr: Group by( )

- Group by( )
  - ✓ Note how complicate it would have been not to use the %>% operator in the previous example:

```
hflights2 <- group_by(filter(hflights2, !is.na(DepDelay), DepDelay > 0),
             UniqueCarrier)

arrange( mutate(summarise(hflights2, avg = mean(DepDelay)),
rank = rank(avg)
  ),
  rank
)
```

dplyr: Group by( )

- Group by( )
  - ✓ Arrange the UniqueCarrier with the delay proportion and their rank

```
# Arrange the UniqueCarrier with the delay proportion and their rank
hflights2 %>%
    group_by(UniqueCarrier) %>%
    filter(!is.na(ArrDelay)) %>%
    summarise(p_delay = mean(ArrDelay > 0)) %>%
    mutate(rank = rank(p_delay)) %>%
    arrange(rank)
```

```
# A tibble: 15 x 3
   UniqueCarrier p_delay  rank
   <chr>           <dbl> <dbl>
 1 AA              0.303     1
 2 FL              0.311     2
 3 US              0.327     3
 4 EV              0.368     4
 5 MQ              0.370     5
 6 DL              0.387     6
 7 B6              0.395     7
 8 AS              0.437     8
 9 WN              0.464     9
10 YV              0.474    10
11 CO              0.491    11
12 XE              0.494    12
13 UA              0.496    13
14 OO              0.535    14
15 F9              0.556    15
```

dplyr: Group by( )

- Group by( )
  - ✓ Arrange the UniqueCarrier with the average arrival delay time with their rank

```
# Arrange the UniqueCarrier with the average arrival delay time with their
rank
hflights2 %>%
    group_by(UniqueCarrier) %>%
    filter(!is.na(ArrDelay), ArrDelay > 0) %>%
    summarise(avg = mean(ArrDelay)) %>%
    mutate(rank = rank(avg)) %>%
    arrange(rank)
```

```
# A tibble: 15 x 3
   UniqueCarrier   avg  rank
   <chr>         <dbl> <dbl>
 1 YV             18.7     1
 2 F9             18.7     2
 3 US             20.7     3
 4 CO             22.1     4
 5 AS             22.9     5
 6 OO             24.1     6
 7 XE             24.2     7
 8 WN             25.3     8
 9 FL             27.9     9
10 AA             28.5    10
11 DL             32.1    11
12 UA             32.5    12
13 MQ             38.8    13
14 EV             40.2    14
15 B6             45.5    15
```

dplyr: Group by( )

- Group by( )
  - ✓ Which plane (by tail number) flew out of Houston the most times? How many times?

```
# Which plane (by tail number) flew out of Houston the most times? How many
times?
hflights2 %>%
    group_by(TailNum) %>%
    summarise(n = n()) %>%
    filter(n == max(n))
```

```
# A tibble: 1 x 2
  TailNum     n
  <chr>   <int>
1 N14945    971
```

dplyr: Group by( )

- Group by( )
  - ✓ How many airplanes only flew to one destination from Houston?

```
# How many airplanes only flew to one destination from Houston?
hflights2 %>%
    group_by(TailNum) %>%
    summarise(ndest = n_distinct(Dest)) %>%
    filter(ndest == 1) %>%
    summarise(nplanes = n())
```

```
# A tibble: 1 x 1
  nplanes
    <int>
1    1526
```

dplyr: Group by( )

- Group by( )
  - ✓ Find the most visited destination for each carrier

```
# Find the most visited destination for each carrier
hflights2 %>%
    group_by(UniqueCarrier, Dest) %>%
    summarise(n = n()) %>%
    mutate(rank = rank(desc(n))) %>%
    filter(rank == 1)
```

```
# A tibble: 15 x 4
# Groups:   UniqueCarrier [15]
   UniqueCarrier Dest      n  rank
   <chr>         <chr> <int> <dbl>
 1 AA            DFW    2105     1
 2 AS            SEA     365     1
 3 B6            JFK     695     1
 4 CO            EWR    3924     1
 5 DL            ATL    2396     1
 6 EV            DTW     851     1
 7 F9            DEN     837     1
 8 FL            ATL    2029     1
 9 MQ            DFW    2424     1
10 OO            COS    1335     1
11 UA            SFO     643     1
12 US            CLT    2212     1
13 WN            DAL    8243     1
14 XE            CRP    3175     1
15 YV            CLT      71     1
```

dplyr: Group by( )

- Group by( )
  - ✓ Find the carrier that travels to each destination the most

```
# Find the carrier that travels to each destination the most
hflights2 %>%
    group_by(Dest, UniqueCarrier) %>%
    summarise(n = n()) %>%
    mutate(rank = rank(desc(n))) %>%
    filter(rank == 1)
```

```
# A tibble: 116 x 4
# Groups:   Dest [116]
   Dest  UniqueCarrier     n  rank
   <chr> <chr>         <int> <dbl>
 1 ABQ   WN             1019     1
 2 AEX   XE              724     1
 3 AGS   CO                1     1
 4 AMA   XE             1297     1
 5 ANC   CO              125     1
 6 ASE   OO              125     1
 7 ATL   DL             2396     1
 8 AUS   CO             2645     1
 9 AVL   XE              350     1
10 BFL   OO              504     1
# ... with 106 more rows
```

Data Transformation with dplyr : : CHEAT SHEET

dplyr functions work with pipes and expect **tidy data**. In tidy data:

Each **variable** is in its own **column**

&

Each **observation**, or **case**, is in its own **row**

pipes

x %>% f(y) becomes **f(x, y)**

Summarise Cases

These apply **summary functions** to columns to create a new table of summary statistics. Summary functions take vectors as input and return one value (see back).

summary function

summarise(.data, …)
Compute table of summaries.
summarise(mtcars, avg = mean(mpg))

count(x, ..., wt = NULL, sort = FALSE)
Count number of rows in each group defined by the variables in … Also **tally**().
count(iris, Species)

VARIATIONS

summarise_all() - Apply funs to every column.
summarise_at() - Apply funs to specific columns.
summarise_if() - Apply funs to all cols of one type.

Group Cases

Use **group_by()** to create a "grouped" copy of a table. dplyr functions will manipulate each "group" separately and then combine the results.

```
mtcars %>%
group_by(cyl) %>%
summarise(avg = mean(mpg))
```

group_by(.data, ..., add = FALSE)
Returns copy of table grouped by …
g_iris <- group_by(iris, Species)

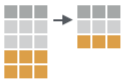
ungroup(x, …)
Returns ungrouped copy of table.
ungroup(g_iris)

Manipulate Cases

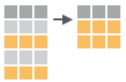
EXTRACT CASES

Row functions return a subset of rows as a new table.

filter(.data, …) Extract rows that meet logical criteria. *filter(iris, Sepal.Length > 7)*

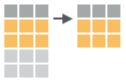


distinct(.data, ..., .keep_all = FALSE) Remove rows with duplicate values.
distinct(iris, Species)



sample_frac(tbl, size = 1, replace = FALSE, weight = NULL, .env = parent.frame()) Randomly select fraction of rows.
sample_frac(iris, 0.5, replace = TRUE)

sample_n(tbl, size, replace = FALSE, weight = NULL, .env = parent.frame()) Randomly select size rows. *sample_n(iris, 10, replace = TRUE)*



slice(.data, …) Select rows by position.
slice(iris, 10:15)

top_n(x, n, wt) Select and order top n entries (by group if grouped data). *top_n(iris, 5, Sepal.Width)*

Logical and boolean operators to use with filter()

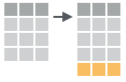
<	<=	is.na()	%in%	|	xor()
>	>=	!is.na()	!	&	

See **?base::Logic** and **?Comparison** for help.

ARRANGE CASES

arrange(.data, …) Order rows by values of a column or columns (low to high), use with **desc()** to order from high to low.
arrange(mtcars, mpg)
arrange(mtcars, desc(mpg))

ADD CASES



add_row(.data, ..., .before = NULL, .after = NULL) Add one or more rows to a table.
add_row(faithful, eruptions = 1, waiting = 1)

Manipulate Variables

EXTRACT VARIABLES

Column functions return a set of columns as a new vector or table.

pull(.data, var = -1) Extract column values as a vector. Choose by name or index.
pull(iris, Sepal.Length)

select(.data, …)
Extract columns as a table. Also **select_if()**.
select(iris, Sepal.Length, Species)

Use these helpers with select (),
e.g. select(iris, starts_with("Sepal"))

contains(match)	**num_range**(prefix, range)	**:**, e.g. mpg:cyl
ends_with(match)	**one_of**(…)	**-**, e.g, -Species
matches(match)	**starts_with**(match)	

MAKE NEW VARIABLES

These apply **vectorized functions** to columns. Vectorized funs take vectors as input and return vectors of the same length as output (see back).

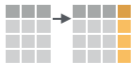
vectorized function

mutate(.data, …)
Compute new column(s).
mutate(mtcars, gpm = 1/mpg)

transmute(.data, …)
Compute new column(s), drop others.
transmute(mtcars, gpm = 1/mpg)

mutate_all(.tbl, .funs, …) Apply funs to every column. Use with **funs()**. Also **mutate_if()**.
mutate_all(faithful, funs(log(.), log2(.)))
mutate_if(iris, is.numeric, funs(log(.)))

mutate_at(.tbl, .cols, .funs, …) Apply funs to specific columns. Use with **funs()**, **vars()** and the helper functions for select().
mutate_at(iris, vars(-Species), funs(log(.)))



add_column(.data, ..., .before = NULL, .after = NULL) Add new column(s). Also **add_count()**, **add_tally()**. *add_column(mtcars, new = 1:32)*

rename(.data, …) Rename columns.
rename(iris, Length = Sepal.Length)

Vector Functions

TO USE WITH MUTATE ()

mutate() and **transmute()** apply vectorized functions to columns to create new columns. Vectorized functions take vectors as input and return vectors of the same length as output.

vectorized function

OFFSETS

dplyr::**lag()** - Offset elements by 1
dplyr::**lead()** - Offset elements by -1

CUMULATIVE AGGREGATES

dplyr::**cumall()** - Cumulative all()
dplyr::**cumany()** - Cumulative any()
cummax() - Cumulative max()
dplyr::**cummean()** - Cumulative mean()
cummin() - Cumulative min()
cumprod() - Cumulative prod()
cumsum() - Cumulative sum()

RANKINGS

dplyr::**cume_dist()** - Proportion of all values <=
dplyr::**dense_rank()** - rank w ties = min, no gaps
dplyr::**min_rank()** - rank with ties = min
dplyr::**ntile()** - bins into n bins
dplyr::**percent_rank()** - min_rank scaled to [0,1]
dplyr::**row_number()** - rank with ties = "first"

MATH

+, - , *, /, ^, %/%, %% - arithmetic ops
log(), log2(), log10() - logs
<, <=, >, >=, !=, == - logical comparisons
dplyr::**between()** - x >= left & x <= right
dplyr::**near()** - safe == for floating point numbers

MISC

dplyr::**case_when()** - multi-case if_else()

```
iris %>% mutate(Species = case_when(
                Species == "versicolor" ~ "versi",
                Species == "virginica"  ~ "virgi",
                TRUE ~ Species))
```

dplyr::**coalesce()** - first non-NA values by element across a set of vectors
dplyr::**if_else()** - element-wise if() + else()
dplyr::**na_if()** - replace specific values with NA
pmax() - element-wise max()
pmin() - element-wise min()
dplyr::**recode()** - Vectorized switch()
dplyr::**recode_factor()** - Vectorized switch() for factors

Summary Functions

TO USE WITH SUMMARISE ()

summarise() applies summary functions to columns to create a new table. Summary functions take vectors as input and return single values as output.

summary function

COUNTS

dplyr::**n()** - number of values/rows
dplyr::**n_distinct()** - # of uniques
sum(!is.na()) - # of non-NA's

LOCATION

mean() - mean, also **mean(!is.na())**
median() - median

LOGICALS

mean() - Proportion of TRUE's
sum() - # of TRUE's

POSITION/ORDER

dplyr::**first()** - first value
dplyr::**last()** - last value
dplyr::**nth()** - value in nth location of vector

RANK

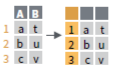
quantile() - nth quantile
min() - minimum value
max() - maximum value

SPREAD

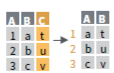
IQR() - Inter-Quartile Range
mad() - median absolute deviation
sd() - standard deviation
var() - variance

Row Names

Tidy data does not use rownames, which store a variable outside of the columns. To work with the rownames, first move them into a column.



rownames_to_column()
Move row names into col.
a <- rownames_to_column(iris, var = "C")

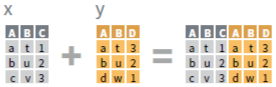


column_to_rownames()
Move col in row names.
column_to_rownames(a, var = "C")

Also **has_rownames()**, **remove_rownames()**

Combine Tables

COMBINE VARIABLES



Use **bind_cols()** to paste tables beside each other as they are.

bind_cols(...) Returns tables placed side by side as a single table.
BE SURE THAT ROWS ALIGN.

Use a "**Mutating Join**" to join one table to columns from another, matching values with the rows that they correspond to. Each join retains a different combination of values from the tables.

left_join(x, y, by = NULL, copy=FALSE, suffix=c(".x",".y"),...)
Join matching values from y to x.

right_join(x, y, by = NULL, copy = FALSE, suffix=c(".x",".y"),...)
Join matching values from x to y.

inner_join(x, y, by = NULL, copy = FALSE, suffix=c(".x",".y"),...)
Join data. Retain only rows with matches.

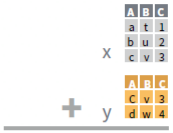
full_join(x, y, by = NULL, copy=FALSE, suffix=c(".x",".y"),...)
Join data. Retain all values, all rows.

Use **by = c("col1", "col2", ...)** to specify one or more common columns to match on.
left_join(x, y, by = "A")

Use a named vector, **by = c("col1" = "col2")**, to match on columns that have different names in each table.
left_join(x, y, by = c("C" = "D"))

Use **suffix** to specify the suffix to give to unmatched columns that have the same name in both tables.
left_join(x, y, by = c("C" = "D"), suffix = c("1", "2"))

COMBINE CASES



Use **bind_rows()** to paste tables below each other as they are.

bind_rows(..., .id = NULL)
Returns tables one on top of the other as a single table. Set .id to a column name to add a column of the original table names (as pictured)

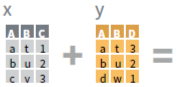
intersect(x, y, ...)
Rows that appear in both x and y.

setdiff(x, y, ...)
Rows that appear in x but not y.

union(x, y, ...)
Rows that appear in x or y. (Duplicates removed). union_all() retains duplicates.

Use **setequal()** to test whether two data sets contain the exact same rows (in any order).

EXTRACT ROWS



Use a "**Filtering Join**" to filter one table against the rows of another.

semi_join(x, y, by = NULL, ...)
Return rows of x that have a match in y. USEFUL TO SEE WHAT WILL BE JOINED.

anti_join(x, y, by = NULL, ...)
Return rows of x that do not have a match in y. USEFUL TO SEE WHAT WILL NOT BE JOINED.


ANY
questions?