Data Wranging with dplyr

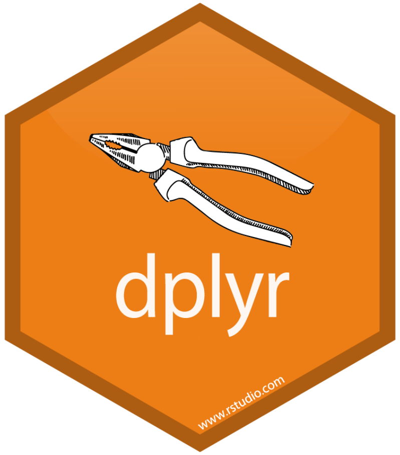
The R Bootcamp
Twitter: @therbootcamp

September 2017

dplyr

dplyr is a package for managing dataframes.

Anytime you want to slice, dice, aggregate, or manipulate a dataframe, there is almost certainly a way to do it in dplyr.

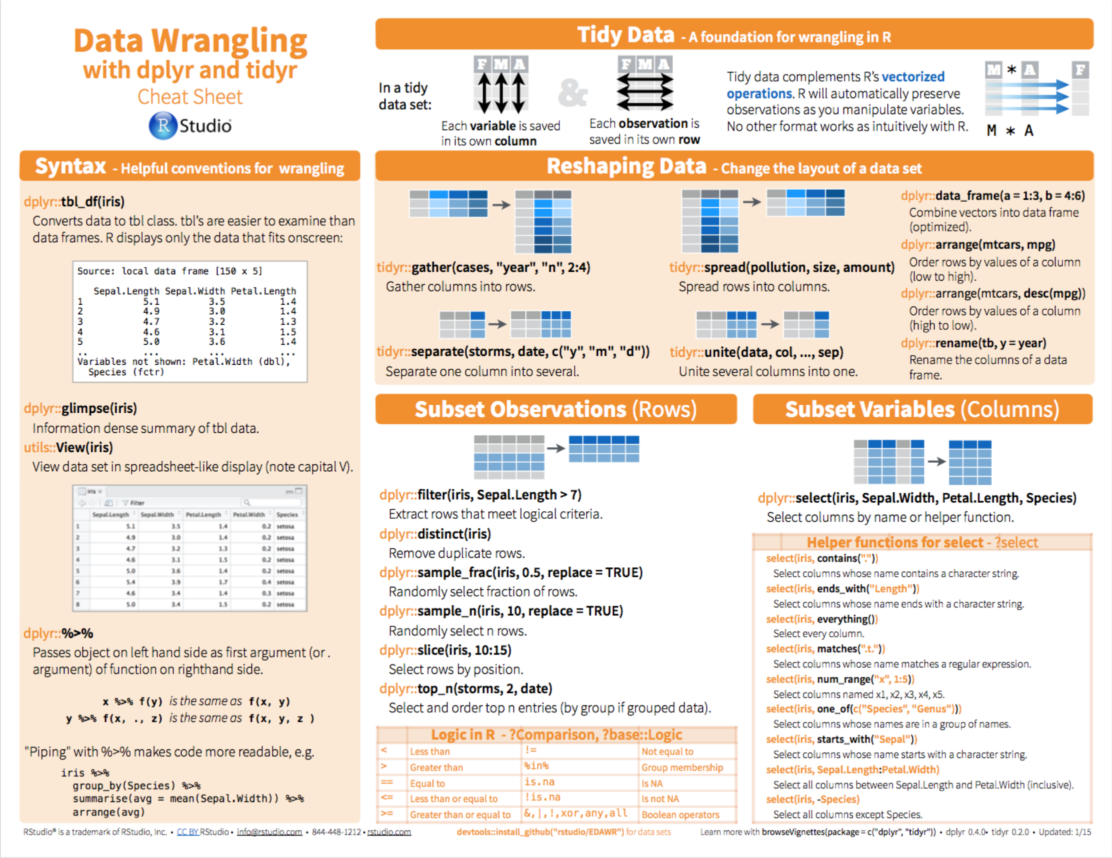


Questions you can answer with dplyr

> *Can you calculate the mean survival times for each treatment separated by gender and time?*
>
> *I need to know the mean birth rate only for countries in Africa from 1980 to 1980.*
>
> *What percent of female patients had adverse events to drug X during weeks 5 through 10?*

dplyr CheatSheet!

https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf



Data Wrangling with dplyr and tidyr Cheat Sheet

RStudio

Syntax - Helpful conventions for wrangling

dplyr::**tbl_df(iris)**
Converts data to tbl class. tbl's are easier to examine than data frames. R displays only the data that fits onscreen:

```
Source: local data frame [150 x 5]

   Sepal.Length Sepal.Width Petal.Length
1           5.1         3.5          1.4
2           4.9         3.0          1.4
3           4.7         3.2          1.3
4           4.6         3.1          1.5
5           5.0         3.6          1.4
..          ...         ...          ...
Variables not shown: Petal.Width (dbl),
  Species (fctr)
```

dplyr::**glimpse(iris)**
Information dense summary of tbl data.

utils::**View(iris)**
View data set in spreadsheet-like display (note capital V).

iris × Filter

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa
7	4.6	3.4	1.4	0.3	setosa
8	5.0	3.4	1.5	0.2	setosa

dplyr::**%>%**
Passes object on left hand side as first argument (or . argument) of function on righthand side.

```
    x %>% f(y)  is the same as  f(x, y)
y %>% f(x, ., z)  is the same as  f(x, y, z )
```

"Piping" with %>% makes code more readable, e.g.

```
iris %>%
  group_by(Species) %>%
  summarise(avg = mean(Sepal.Width)) %>%
  arrange(avg)
```

Tidy Data - A foundation for wrangling in R

In a tidy data set:

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

&

Each **observation** is saved in its own **row**

Tidy data complements R's **vectorized operations.** R will automatically preserve observations as you manipulate variables. No other format works as intuitively with R.

Reshaping Data - Change the layout of a data set

tidyr::**gather(cases, "year", "n", 2:4)**
Gather columns into rows.

tidyr::**spread(pollution, size, amount)**
Spread rows into columns.

tidyr::**separate(storms, date, c("y", "m", "d"))**
Separate one column into several.

tidyr::**unite(data, col, ..., sep)**
Unite several columns into one.

dplyr::**data_frame(a = 1:3, b = 4:6)**
Combine vectors into data frame (optimized).

dplyr::**arrange(mtcars, mpg)**
Order rows by values of a column (low to high).

dplyr::**arrange(mtcars, desc(mpg))**
Order rows by values of a column (high to low).

dplyr::**rename(tb, y = year)**
Rename the columns of a data frame.

Subset Observations (Rows)

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

dplyr::**distinct(iris)**
Remove duplicate rows.

dplyr::**sample_frac(iris, 0.5, replace = TRUE)**
Randomly select fraction of rows.

dplyr::**sample_n(iris, 10, replace = TRUE)**
Randomly select n rows.

dplyr::**slice(iris, 10:15)**
Select rows by position.

dplyr::**top_n(storms, 2, date)**
Select and order top n entries (by group if grouped data).

Logic in R - ?Comparison, ?base::Logic

<	Less than	!=	Not equal to
>	Greater than	%in%	Group membership
==	Equal to	is.na	Is NA
<=	Less than or equal to	!is.na	Is not NA
>=	Greater than or equal to	&,|,!,xor,any,all	Boolean operators

Subset Variables (Columns)

dplyr::**select(iris, Sepal.Width, Petal.Length, Species)**
Select columns by name or helper function.

Helper functions for select - ?select

select(iris, **contains(".")**)
Select columns whose name contains a character string.

select(iris, **ends_with("Length")**)
Select columns whose name ends with a character string.

select(iris, **everything()**)
Select every column.

select(iris, **matches(".t.")**)
Select columns whose name matches a regular expression.

select(iris, **num_range("x", 1:5)**)
Select columns named x1, x2, x3, x4, x5.

select(iris, **one_of(c("Species", "Genus"))**)
Select columns whose names are in a group of names.

select(iris, **starts_with("Sepal")**)
Select columns whose name starts with a character string.

select(iris, Sepal.Length:Petal.Width)
Select all columns between Sepal.Length and Petal.Width (inclusive).

select(iris, -Species)
Select all columns except Species.

RStudio® is a trademark of RStudio, Inc. • CC BY RStudio • info@rstudio.com • 844-448-1212 • rstudio.com

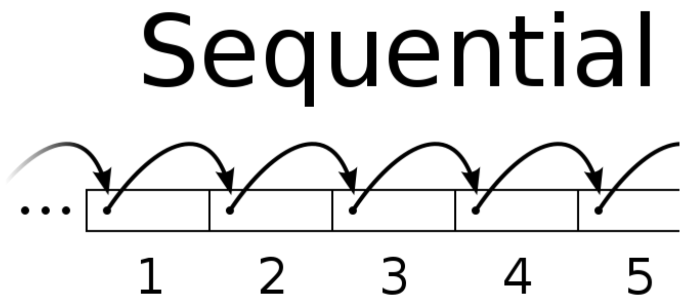
devtools::install_github("rstudio/EDAWR") for data sets

Learn more with browseVignettes(package = c("dplyr", "tidyr")) • dplyr 0.4.0• tidyr 0.2.0 • Updated: 1/15

dplyr

dplyr is a combination of 3 things:

1. **`objects`** like dataframes
2. **`verbs`** that **`do`** things to objects.
3. **`pipes`** `%>%` that string together objects and verbs



dplyr is meant to be sequential and work like language

> Take data X, then do Y, then do Z...

Here's the basic structure of `dplyr` in action

```
data %>%              # Start with data, and THEN
     VERB1 %>%        # Do VERB1, (and THEN)
     VERB2 %>% ...    # Do VERB2, (and THEN)
```

dplyr

Question:

> From the ChickWeight dataframe, calculate the mean weight and time for each diet

Answer:

```
library(dplyr)

x <- ChickWeight %>%  # Start with ChickWeight
  group_by(Diet) %>%  # Group by Diet
  summarise(          # Get ready to summarise....
    weight.mean = mean(weight), # Mean weight
    time.mean = mean(Time),     # Mean time
    N = n()                     # Number of cases
  )


x
```

```
## # A tibble: 4 x 4
##     Diet weight.mean time.mean     N
##   <fctr>       <dbl>     <dbl> <int>
## 1      1       102.6     10.48   220
## 2      2       122.6     10.92   120
## 3      3       142.9     10.92   120
## 4      4       135.3     10.75   118
```

Common dplyr verbs

verb	action	example
`filter()`	Select rows based on some criteria	`filter(age > 40 & sex == "m")`
`arrange()`	Sort rows	`arrange(date, group)`
`select()`	Select columns (and ignore all others)	`select(age, sex)`
`rename()`	Rename columns	`rename(DATE_MONTHS_X24, date)`
`mutate()`	Add new columns	`mutate(height.m = height.cm / 100)`
`case_when()`	Recode values of a column	`sex.n = case_when(sex == 0 ~ "m", sex == 1 ~ "f")`
`group_by()`, `summarise()`	Group data and then calculate summary statistics	`group_by(treatment) %>% summarise(...)`

Example 1

Add a column called `weight_d_time` that is weight divided by time

```r
library(dplyr)

x <- ChickWeight %>%          # Start with the ChickWeight data
      mutate(                 # Create new columns...
            weight_d_time = weight / Time
            )

head(x)   # Print the result
```

```
##   weight Time Chick Diet weight_d_time
## 1     42    0     1    1           Inf
## 2     51    2     1    1         25.50
## 3     59    4     1    1         14.75
## 4     64    6     1    1         10.67
## 5     76    8     1    1          9.50
## 6     93   10     1    1          9.30
```

Example 2

Add a column called `weight_d_time` that is weight divided by time AND `time_d` that is time in days

```
x <- ChickWeight %>%          # Start with the ChickWeight data
      mutate(                 # Create new columns...
            weight_d_time = weight / Time,  # weight_d_time is weight divided by Time
            time_d = Time * 7               # time_d is Time times 7
            )

head(x)   # Print the result
```

```
##   weight Time Chick Diet weight_d_time time_d
## 1     42    0     1    1           Inf      0
## 2     51    2     1    1         25.50     14
## 3     59    4     1    1         14.75     28
## 4     64    6     1    1         10.67     42
## 5     76    8     1    1          9.50     56
## 6     93   10     1    1          9.30     70
```

Recoding values with case_when()

Recoding values is a common data wrangling task. You can easily do this with `case_when()`:

```
data %>%
  mutate(
  var_new = case_when(
    var_old == OLD_A ~ NEW_A,
    var_old == OLD_B ~ NEW_B
  )
```

For example, in a dataset, the column `sex` might be coded with 1s and 0s.

You might want to create a new column `sex_new` where 1 = "female" and 0 = "male":

sex	sex_new
1	"female"
0	"male"

To change the value of 1 to `"female"`, and 0 to `"male"`, you can use `case_when()`:

```
# Add a column sex_new to data

data <- data %>%
  mutate(
        sex_new = case_when(
          sex == 1 ~ "female",
          sex == 0 ~ "male"
        )
  )
```

You can think about the code above as follows:

- Create a new column `sex_new` where
  - If `sex == 1`, then set the value to `"female"`
  - If `sex == 0`, then set the value to `"male"`

Example 3

Create a new variable Diet_name which shows Diet in text format. Here is a table of the values

Diet	Diet_name
1	"fruit"
2	"vegetables"
3	"meat"
4	"grains"

```
ChickWeight <- ChickWeight %>%           # Start
                 mutate(
                   Diet_name = case_when(
                     Diet == 1 ~ "fruit",
                     Diet == 2 ~ "vegetables",
                     Diet == 3 ~ "meat",
                     Diet == 4 ~ "grains"
                   )

 )

head(ChickWeight)
```

```
##   weight Time Chick Diet Diet_name
## 1     42    0     1    1     fruit
## 2     51    2     1    1     fruit
## 3     59    4     1    1     fruit
## 4     64    6     1    1     fruit
## 5     76    8     1    1     fruit
## 6     93   10     1    1     fruit
```

Example 4

For each Diet, calculate the mean weight

```
ChickWeight %>%           # Start with the ChickWeight data
  group_by(Diet) %>%      # Group the data by Diet
  summarise(              # Now summarise....
    weight.mean = mean(weight) # Mean weight
  )
```

```
## # A tibble: 4 x 2
##     Diet weight.mean
##   <fctr>       <dbl>
## 1      1       102.6
## 2      2       122.6
## 3      3       142.9
## 4      4       135.3
```

Example 5

For each time period less than 10, calculate the mean weight

```
ChickWeight %>%                        # Start with the ChickWeight data
  filter(Time < 10) %>%                # Only Time periods less than 10
  group_by(Time) %>%                   # Group the data by Diet
  summarise(                           # Now summarise....
    weight.mean = mean(weight) # Mean weight
  )
```

```
## # A tibble: 5 x 2
##    Time weight.mean
##   <dbl>       <dbl>
## 1     0       41.06
## 2     2       49.22
## 3     4       59.96
## 4     6       74.31
## 5     8       91.24
```

Example 6

For each Diet, calculate the mean weight, maximum time, and the number of chicks on each diet:

```
ChickWeight %>%           # Start with the ChickWeight data
  group_by(Diet) %>%      # Group the data by Diet
  summarise(              # Now summarise....
    weight.mean = mean(weight), # Mean weight
    time.max = max(Time),       # Max time
    N = n()                     # Number of observations
  )
```

```
## # A tibble: 4 x 4
##     Diet weight.mean time.max     N
##   <fctr>       <dbl>    <dbl> <int>
## 1      1       102.6       21   220
## 2      2       122.6       21   120
## 3      3       142.9       21   120
## 4      4       135.3       21   118
```

Other dplyr verbs

verb	action	example
`sample_n()`	Select a random sample of n rows	`sample_n(10)`
`sample_frac()`	Select a random fraction of rows	`sample_frac(.20)`
`first(), last()`	Give the first (or last) observation	`first(), last()`

Example 7

Give me a random sample of 10 rows from the ChickWeight dataframe, but only show me the values for Chick and weight

```
# Give me a random sample of 10 rows, but only show me columns Chick and weight

ChickWeight %>%
  select(Chick, weight) %>%
  sample_n(10)
```

```
##     Chick weight
## 46      4    154
## 343    31     62
## 61      6     41
## 429    38     98
## 113    10     81
## 466    41    124
## 547    48    104
## 482    42    234
## 335    30    115
## 33      3    163
```

dplyr

dplyr operations (almost) always return a dataframe which you can assign to a new object:

> *Create a dataframe with the average weight for each time period and nothing else!!*

```
# Create a new object called time_agg

time_agg <- ChickWeight %>%
  group_by(Time) %>%
  summarise(
    weight.mean = mean(weight)
  )

head(time_agg)
```

```
## # A tibble: 6 x 2
##    Time weight.mean
##   <dbl>       <dbl>
## 1     0       41.06
## 2     2       49.22
## 3     4       59.96
## 4     6       74.31
## 5     8       91.24
## 6    10      107.84
```

dplyr summary

dplyr is great for elegantly performing sequential operations on data.

The 'pipe' operator `%>%` helps you string multiple *objects* (like dataframes) and *verbs* (summarise, order, aggregate...) together.

Basic structure of dplyr commands:

```
data %>%      # Start with data, AND THEN...
  VERB1 %>% # Do VERB1, AND THEN...
  VERB2 %>% # Do VERB2, AND THEN...
  VERB3 %>% # Do VERB3, AND THEN...
  group_by(x, y) %>%  # Group by variables x, y
    summarise(
      VAR_A_New = fun(X),
      VAR_B_New = fun(Y)
    )
  )
```

Questions?

Wrangling Pratical

Link to Wrangling practical

<!--

-->

<!--

-->

<!--

-->

<!--

-->