dplyr
(and the tidyverse)

Matthew Flickinger, Ph.D.

CSG Tech Talk

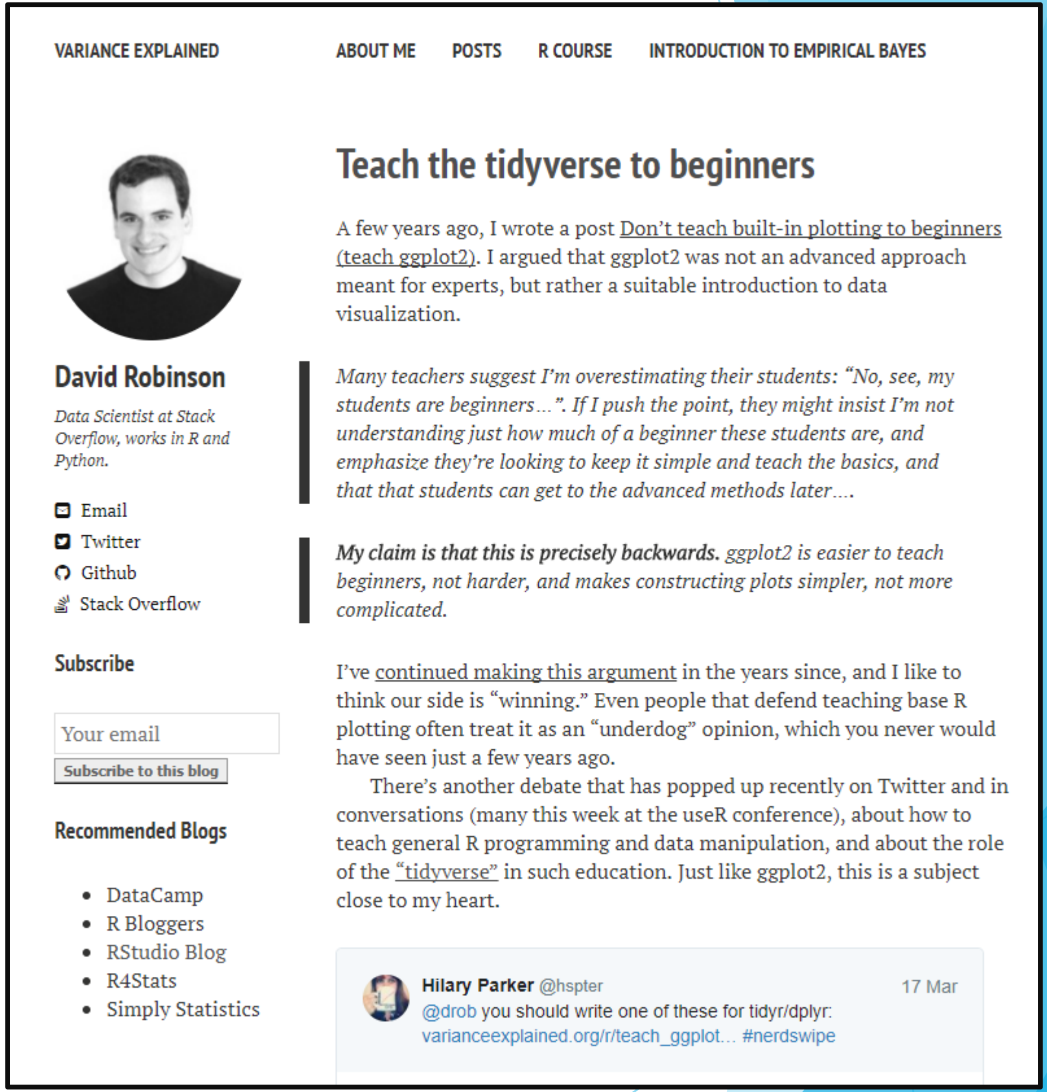
University of Michigan

July 12, 2017

The Tidyverse

Tidyverse

- Very popular, widely used
- Prioritize data analysis rather than computer science
- Enable learners to become productive more quickly
- Encourages readable code



VARIANCE EXPLAINED   ABOUT ME   POSTS   R COURSE   INTRODUCTION TO EMPIRICAL BAYES

David Robinson

Data Scientist at Stack Overflow, works in R and Python.

- Email
- Twitter
- Github
- Stack Overflow

Subscribe

Your email

Subscribe to this blog

Recommended Blogs

- DataCamp
- R Bloggers
- RStudio Blog
- R4Stats
- Simply Statistics

Teach the tidyverse to beginners

A few years ago, I wrote a post Don't teach built-in plotting to beginners (teach ggplot2). I argued that ggplot2 was not an advanced approach meant for experts, but rather a suitable introduction to data visualization.

> *Many teachers suggest I'm overestimating their students: "No, see, my students are beginners…". If I push the point, they might insist I'm not understanding just how much of a beginner these students are, and emphasize they're looking to keep it simple and teach the basics, and that that students can get to the advanced methods later….*

> ***My claim is that this is precisely backwards.** ggplot2 is easier to teach beginners, not harder, and makes constructing plots simpler, not more complicated.*

I've continued making this argument in the years since, and I like to think our side is "winning." Even people that defend teaching base R plotting often treat it as an "underdog" opinion, which you never would have seen just a few years ago.

There's another debate that has popped up recently on Twitter and in conversations (many this week at the useR conference), about how to teach general R programming and data manipulation, and about the role of the "tidyverse" in such education. Just like ggplot2, this is a subject close to my heart.

Hilary Parker @hspter   17 Mar
@drob you should write one of these for tidyr/dplyr: varianceexplained.org/r/teach_ggplot... #nerdswipe

http://varianceexplained.org/r/teach-tidyverse/

Dplyr motivation

- Analysists spend a lot of time manipulating and summarizing data
- Base R provides many functions for this, but
  - the syntax is sometimes verbose or "ugly"
  - the functions can be slow for big data
- dplyr exists to make code easier to read and faster

Install and load dplyr

- Install via tidyverse
  - `install.packages("tidyverse")`
  - `library(tidyverse)`
- OR install directly
  - `install.packages("dplyr")`
  - `library(dplyr)`
- This guide assumes you're running dplyr 0.7.1 (released June 22, 2017)

Sample data

- Examples use a data set containing all out-bound flights from NYC in 2013
- Available as an R package
  - `install.packages("nycflights13")`
  - `library(nycflights13)`

"flights" table

```
> flights
Source: local data frame [336,776 x 19]

    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
   (int) (int) (int)    (int)          (int)     (dbl)    (int)          (int)
1   2013     1     1      517            515         2      830            819
2   2013     1     1      533            529         4      850            830
3   2013     1     1      542            540         2      923            850
4   2013     1     1      544            545        -1     1004           1022
5   2013     1     1      554            600        -6      812            837
6   2013     1     1      554            558        -4      740            728
7   2013     1     1      555            600        -5      913            854
8   2013     1     1      557            600        -3      709            723
9   2013     1     1      557            600        -3      838            846
10  2013     1     1      558            600        -2      753            745
..   ...   ...   ...      ...            ...       ...      ...            ...
Variables not shown: arr_delay (dbl), carrier (chr), flight (int), tailnum
  (chr), origin (chr), dest (chr), air_time (dbl), distance (dbl), hour (dbl),
  minute (dbl), time_hour (time)
```

Basic single-table dplyr verbs

Basic dplyr verbs

- filter() – keep rows matching desired properties
- select() – choose which columns you want to extract
- arrange() – sort rows
- mutate() – create new columns
- summarize() – collapse rows into summaries
- group_by() – operate on subsets of rows at a time

dplyr verb properties

- Always take a data source as the first parameter
- Returns a new data object, never updates/replace original
- Specify columns as unquoted strings (symbols)

Filtering Rows

- Find all flights to Detroit (DTW) in June (2013)
- dplyr
  - `filter(flights, dest=="DTW" & month==6)`
- Base R
  - `flights[flights$dest=="DTW" & flights$month==6, ]`
  - `subset(flights, dest=="DTW" & month==6)`

Selecting columns

- Select specific columns
  - `select(flights, dep_time, arr_time, carrier)`
- Exclude columns
  - `select(flights, -year, -tailnum)`
- Select column range
  - `select(flights, month:dep_delay)`

Selecting columns … part 2

- `select(flights, starts_with("d"))`
- `select(flights, ends_with("time"))`
- `select(flights, contains("arr"))`
- `select(flights, -starts_with("d"))`
- `select(flights, flight, everything())`

- See "?select" for complete list and examples

Verb composition via magrittr

- What if we want to both filter and select?
- We could create variables for each intermediate step
  - `filtered <- filter(flights, dest=="DTW")`
  - `select(filtered, carrier)`
- Or we could nest the calls
  - `select(filter(flights, dest=="DTW"), carrier)`
- But these can get messy

Verb composition via magrittr

- The magrittr package introduces the "%>%" operator to "pipe" data into function
- Similar to the unix pipe operator
  - grep apple fruit.txt | head -50 | cut –f3 | sort | uniq –c
- The %>% operator passes the result from the left side to the first argument of the right side
  - `a(b(c(x)) ⬄ x %>% c() %>% b() %>% a()`
- `flights %>% filter(dest=="DTW") %>% select(carrier)`

Sort data

- Use arrange() to sort your rows
  - `flights %>% arrange(sched_dep_time)`
- Use desc() to reverse the sort order of a column
  - `flights %>% arrange(month, desc(day))`
- You can sort on functions of variables

  -
    ```
    flights %>%
     arrange(desc(dep_time-sched_dep_time ))
    ```

Create new variables

- Mutate allows you to create columns using existing values

  -
    ```
    flights %>%
     mutate(speed = distance/(air_time/60)) %>%
     arrange(desc(speed)) %>%
     select(flight, speed)
    ```
- Remember, changes are not saved to "flights", be sure to save the result if you want to use it later

  -
    ```
    new_flights <- flights %>% mutate(...)
    ```

Use new variables right away

- The parameters to mutate are processed in the order they appear

-
  ```
  flights %>% mutate(
      dist_km = distance * 1.61,
      hours = air_time / 60,
      kph = dist_km/hours ) %>%
   select(flight, kph)
  ```
- Be careful! You can overwrite existing variables

Summarize data

- You generally use summarize() to reduce the number of rows in your data by specifying summary functions for each of the columns
-
```
flights %>%
 filter(!is.na(arr_delay)) %>%
 summarize(avg_arr_delay = mean(arr_delay))
```
- Useful summary functions:
  - mean(), median(), var(), sd(), min(), max(), first(), last(), n(), n_distinct()

Grouping data

- Often you want to perform summaries for groups of rows at a time
- The `group_by()` function allows your to specify columns that define groups
- Functions like `mutate()` and `summarize()` are performed for each group

group_by() + summarize() example

```
flights %>%
  filter(!is.na(arr_delay)) %>%
  group_by(carrier) %>%
  summarize(avg_arr_delay = mean(arr_delay))
```

```
# A tibble: 16 x 2
   carrier avg_arr_delay
     <chr>         <dbl>
 1      9E     7.3796692
 2      AA     0.3642909
 3      AS    -9.9308886
 4      B6     9.4579733
 ...
```

Combining group_by() with transformations

- `mutate()`
  - Will not change the number of rows
  - Functions like `max()` will return the max for each group
- `summarize()`
  - Returns one row per group
  - You must tell summarize how to collapse all non-grouped columns that you want

group_by() + mutate() example

```
flights %>%
  filter(!is.na(arr_delay)) %>%
  group_by(carrier) %>%
  mutate(avg_arr_delay = mean(arr_delay)) %>%
  select(carrier, arr_delay, avg_arr_delay)
```

```
# A tibble: 327,346 x 3
# Groups:   carrier [16]
   carrier arr_delay avg_arr_delay
     <chr>     <dbl>         <dbl>
 1      UA        11     3.5580111
 2      UA        20     3.5580111
 3      AA        33     0.3642909
 4      B6       -18     9.4579733
 ...
```

Merging data

What are these carrier codes?

```
flights %>%
   filter(!is.na(arr_delay)) %>%
   group_by(carrier) %>%
   summarize(avg_arr_delay = mean(arr_delay))
```

```
# A tibble: 16 x 2
   carrier avg_arr_delay
     <chr>         <dbl>
 1      9E     7.3796692
 2      AA     0.3642909
 3      AS    -9.9308886
 4      B6     9.4579733
 ...
```

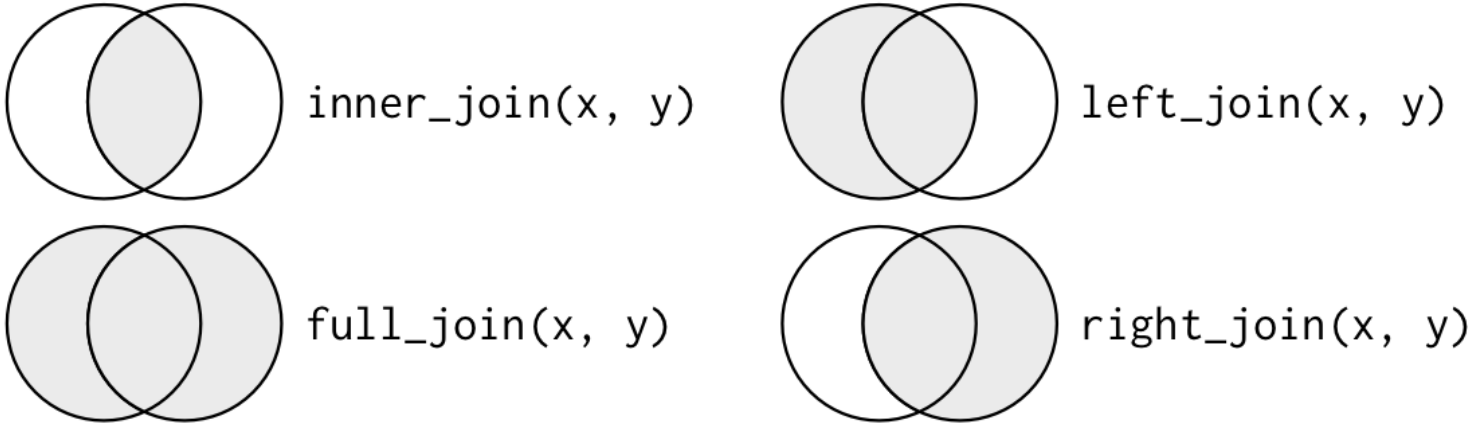
"airlines" table

```
> airlines
# A tibble: 16 x 2
   carrier                       name
     <chr>                      <chr>
 1      9E          Endeavor Air Inc.
 2      AA     American Airlines Inc.
 3      AS       Alaska Airlines Inc.
 4      B6            JetBlue Airways
 5      DL       Delta Air Lines Inc.
 ...
```

Join flights to airlines

```
> flights %>%
   filter(!is.na(arr_delay)) %>%
   group_by(carrier) %>%
   summarize(avg_arr_delay = mean(arr_delay)) %>%
   left_join(airlines)
Joining, by = "carrier"
# A tibble: 16 x 3
   carrier avg_arr_delay                        name
     <chr>         <dbl>                       <chr>
 1      9E     7.3796692           Endeavor Air Inc.
 2      AA     0.3642909      American Airlines Inc.
 3      AS    -9.9308886        Alaska Airlines Inc.
 4      B6     9.4579733             JetBlue Airways
```

Types of joins (merges)

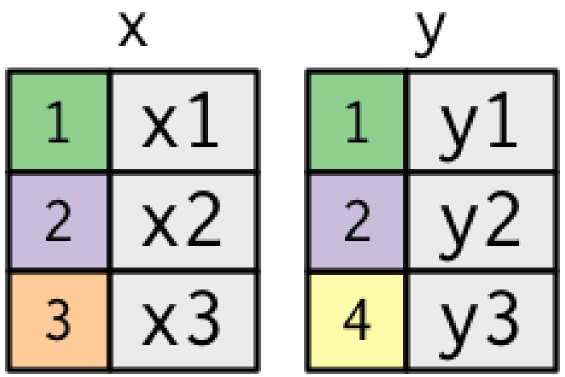


inner_join(x, y)

left_join(x, y)

full_join(x, y)

right_join(x, y)



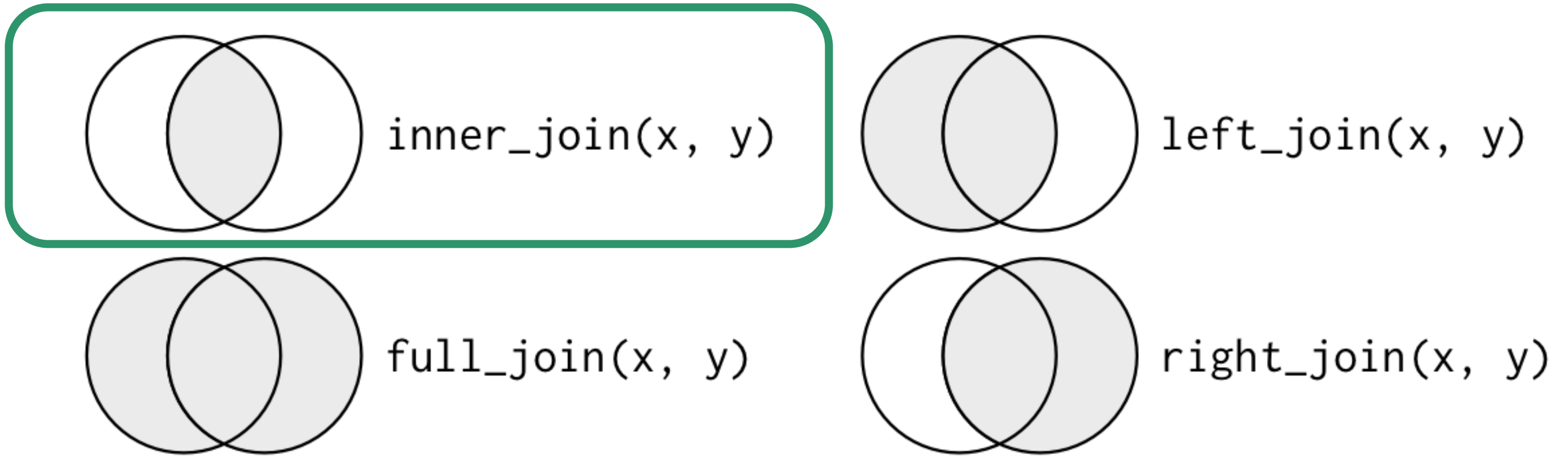
x

1	x1
2	x2
3	x3

y

1	y1
2	y2
4	y3

Types of joins (merges)

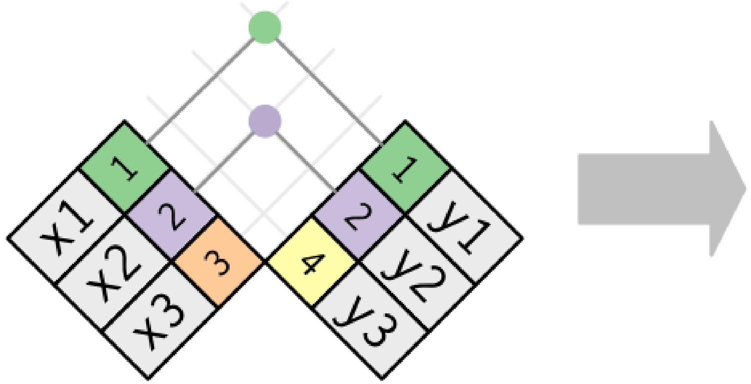


inner_join(x, y)

left_join(x, y)

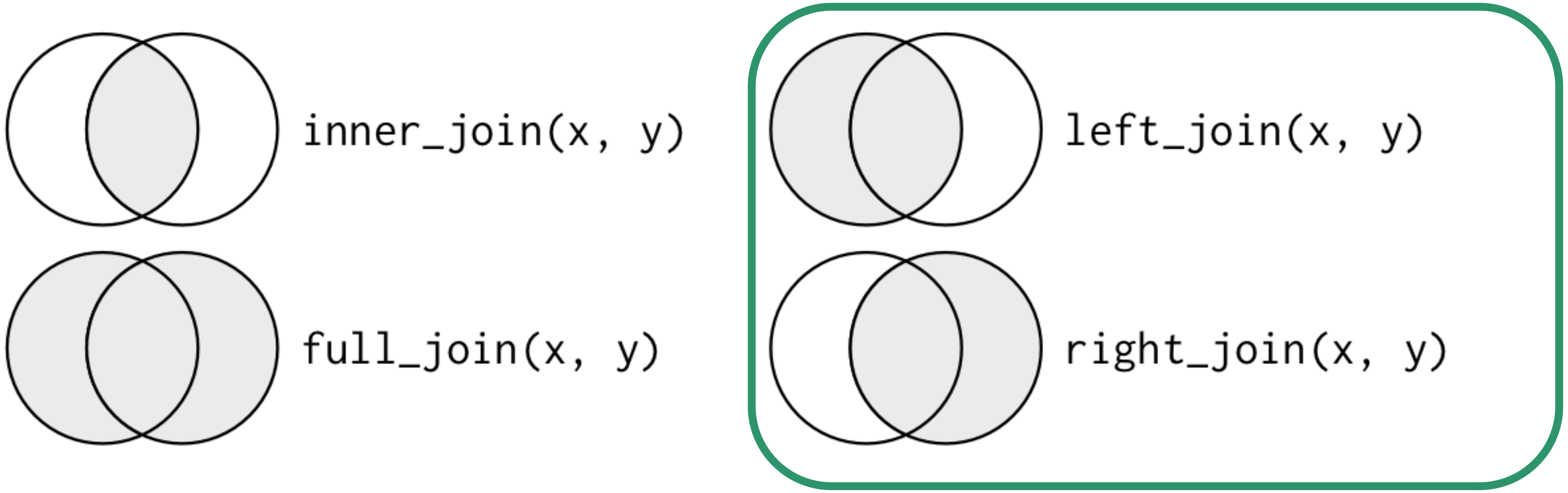
full_join(x, y)

right_join(x, y)



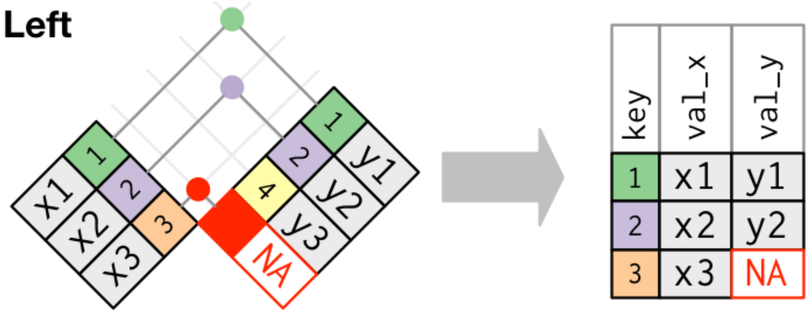
key	val_x	val_y
1	x1	y1
2	x2	y2

Types of joins (merges)



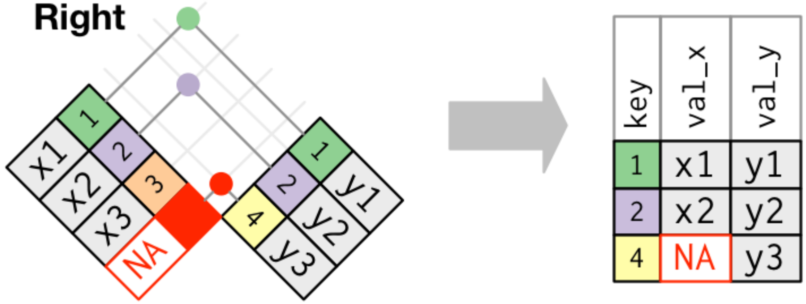
```
inner_join(x, y)
full_join(x, y)
left_join(x, y)
right_join(x, y)
```

Left



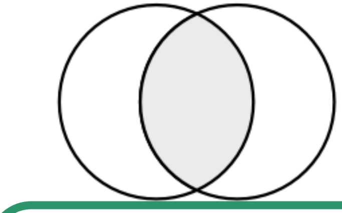
key	val_x	val_y
1	x1	y1
2	x2	y2
3	x3	NA

Right

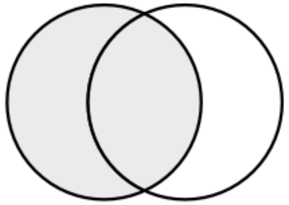


key	val_x	val_y
1	x1	y1
2	x2	y2
4	NA	y3

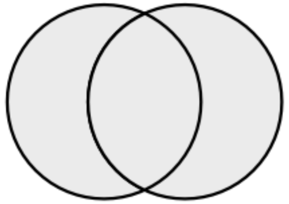
Types of joins (merges)



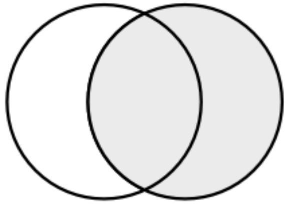
`inner_join(x, y)`



`left_join(x, y)`

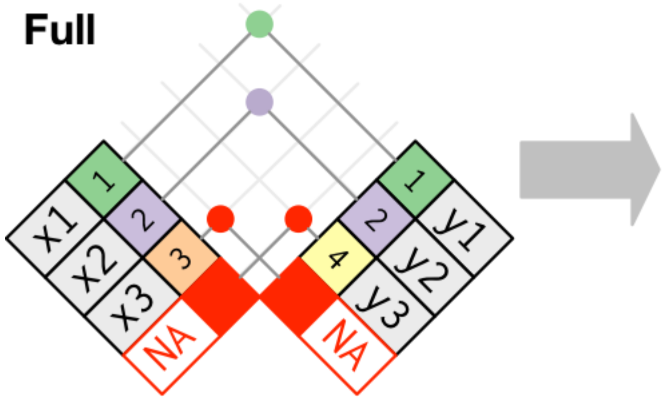


`full_join(x, y)`



`right_join(x, y)`

Full



key	val_x	val_y
1	x1	y1
2	x2	y2
3	x3	NA
4	NA	y3

Non-merging joins

- These "joins" do not add any new columns to your data
- They useful for subsetting with multi-column matches
- `semi_join()`
  - Only keep rows in left table with matches in right
- `anti_join()`
  - Drop rows in left table with matches in right

Join on

- By default the join commands will join two tables based on all matching column names
- You can control the joining by specifying the column names
- `flights %>% inner_join(planes) %>% nrow`
- `flights %>% inner_join(planes, "tailnum") %>% nrow`

Other dplyr functions

Subsetting observations

- `distinct()`
  - Return unique rows
  - `flights %>% distinct(tailnum, carrier)`
- `count()`
  - count rows with unique values of selected columns
  - `flights %>% count(carrier)`
- `sample_n()`
  - Randomly choose n rows
  - `flights %>% sample_n(3)`

Transformation functions

- `summarize_all()/mutate_all()`
  - Apply function to all non-grouped columns
- `summarize_at()/mutate_at()`
  - Apply function to chosen columns
  - `flights %>% summarize_at(vars(ends_with("time")), mean, na.rm=T)`
- `summarize_if()/mutate_if()`
  - Apply function to matching columns
  - `flights %>% summarize_if(is.numeric, funs(mean, var), na.rm=T)`
- See `vars()` and `funs()` for passing multiple columns or functions

Other functions

- `lead()/lag()`
  - Get value just after/before current value
  - `x<-1:5; x; lead(x); lag(x)`

```
[1]  1  2  3  4  5
[1]  2  3  4  5 NA
[1] NA  1  2  3  4
```

- `coalesce()`
  - Returns first non-NA value from vectors
  - `coalesce(c(NA,2,NA), c(1, NA, NA), 3)`

```
[1] 1 2 3
```

- `n()/n_distinct()`
  - Number of (distinct) values is a vector
  - Can only be used within summarize(), mutate(), filter()
  - `flights %>% group_by(tailnum) %>% summarize(flights=n(), routes=n_distinct(flight))`

Other functions

- `recode()`
  - Replace values in vector with different values
  - `recode(letters[1:5], b="boo")`

```
[1] "a" "boo" "c" "d" "e"
```

- `case_when()`
  - Alternative to nested ifelse() calls

```
x <- 1:50
case_when(
  x %% 35 == 0 ~ "fizz buzz",
  x %% 5 == 0 ~ "fizz",
  x %% 7 == 0 ~ "buzz",
  TRUE ~ as.character(x)
)
```

```
 [1] "1"      "2"      "3"      "4"
 [5] "fizz"   "6"      "buzz"   "8"
 [9] "9"      "fizz"   "11"     "12"
[13] "13"     "buzz"   "fizz"   "16"
```

Combining data frames

- `bind_rows()`
  - Stack two data frames on top of each other (should have the same number of columns)
- `bind_columns()`
  - Place two data frames next to each other (should have the same number of rows) – no merge-able columns
- `intersect(), union(), setdiff()`
  - For rows shared or exclusive to data frames

Other benefits of dplyr

Generic data interface

- In all these examples we've been working with basic R data.frames
- However, dplyr can act as a front end to other data types
- This include databases (via SQL) and data.tables

"Tibbles"

```
> class(flights)
[1] "tbl_df"      "tbl"          "data.frame"
```

- dplyr works with objects of class "tbl" (pronounced tibble)
- Here we have a wrapped data.frame

Data tables

- The "data.table" package exist to make data frame like structures that are faster and more efficient to work with
- The "data.table" package overload the subset operator "[" to allow for grouping and subsetting in a non-standard way
- If you load the "dtplyr" package, you can use the nicer dplyr functions to work with data tables as well
- (Prior to dplyr 0.5.0, the data tables functions were in the same package and didn't need to be loaded separately)

Programming with dplyr

Writing functions with dplyr

```
# WORKS
flights %>%
  group_by(carrier) %>%
  summarize(delay=mean(arr_delay, na.rm=T))

# DOESN’T WORK
f <- function(x) {
  flights %>%
    group_by(x) %>%
    summarize(delay=mean(arr_delay, na.rm=T))
}
f(carrier)  # ERROR: Column `x` is unknown
```

Quosures

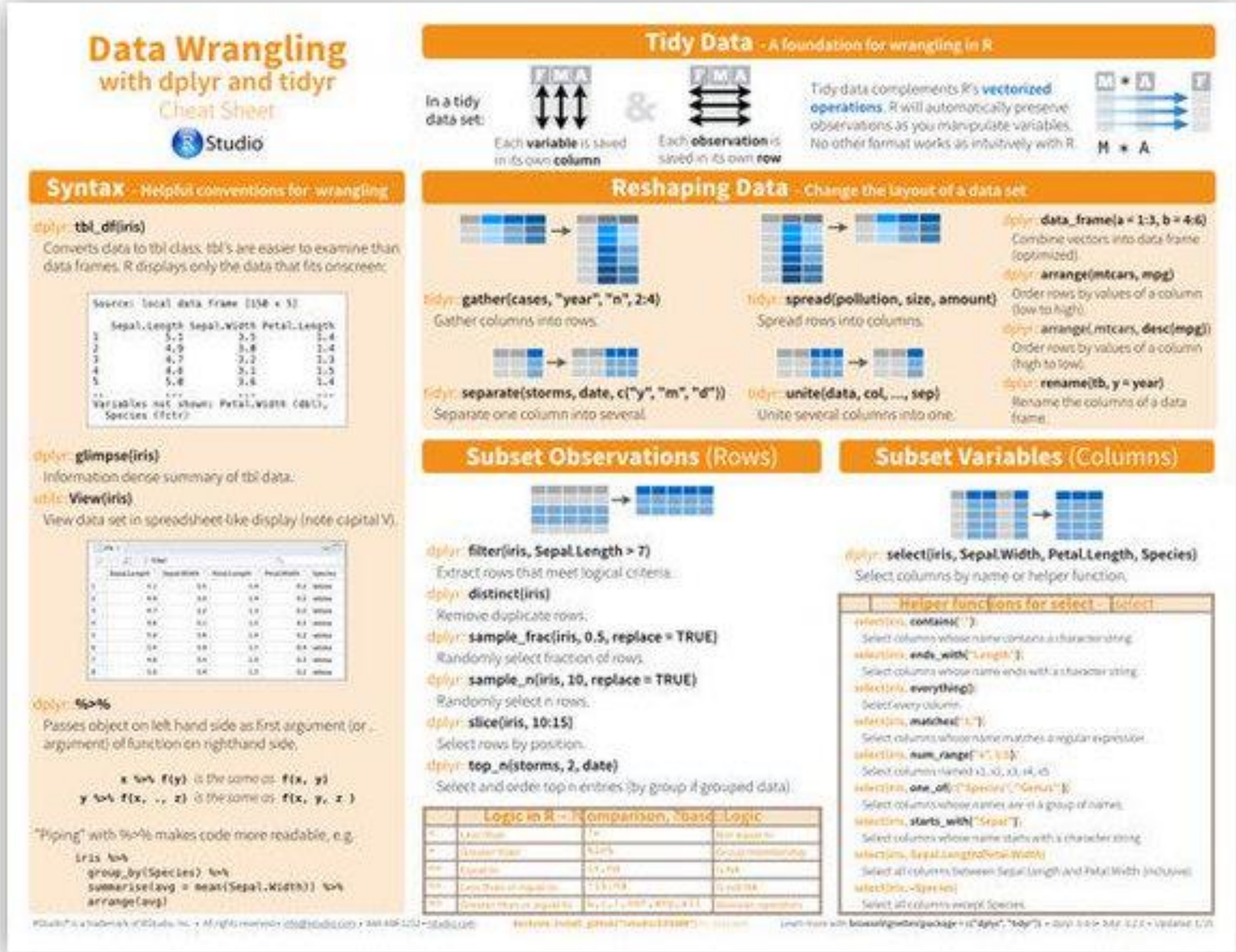
```
f <- function(x) {
  flights %>% group_by(!!x) %>%
  summarize(delay = mean(arr_delay, na.rm=T))
}
f(quo(carrier))

g <- function(x) {
  x <- enquo(x)
  flights %>% group_by(!!x) %>%
  summarize(delay = mean(arr_delay, na.rm=T))
}
g(carrier)
```

```
# A tibble: 16 x 2
   carrier      delay
     <chr>      <dbl>
 1      9E  7.3796692
 2      AA  0.3642909
 ...
```

Rename output (:=)

```
h <- function(x) {
  x <- enquo(x)
  outname <- paste(quo_name(x), "delay", sep="_")
  flights %>% group_by(!!x) %>%
  summarize(!!outname := mean(arr_delay, na.rm=T))
}
h(carrier)
```

```
# A tibble: 16 x 2
   carrier carrier_delay
     <chr>         <dbl>
 1      9E     7.3796692
 2      AA     0.3642909
 ...
```



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).

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.

M * A

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. • All rights reserved • info@rstudio.com • 844-448-1212 • rstudio.com

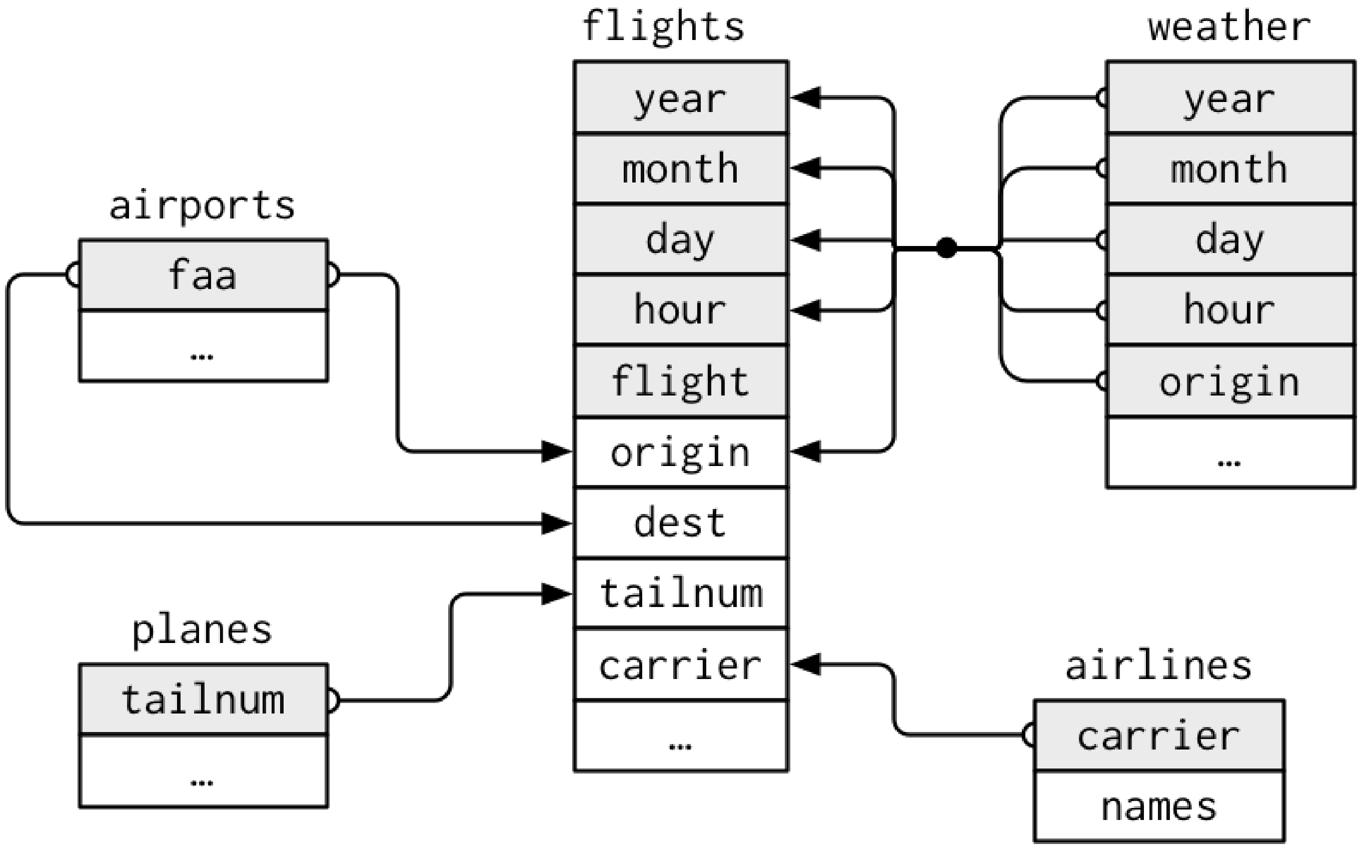
devtools::install_github("rstudio/EDAWR") for data sets

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

"Data Wrangling Cheat Sheet"

```
you %>%
  select(interesting_dataset) %>%
  summarize(features) %>%
  test(hypothesis) %>%
  profit() %>%
  the_end()
```

nycflights13 tables



flights
year
month
day
hour
flight
origin
dest
tailnum
carrier
…
weather
year
month
day
hour
origin
…
airports
faa
…
planes
tailnum
…
airlines
carrier
names