dplyr

Matthew Flickinger, Ph.D.

University of Michigan

BDSI June 21, 2019

About Me

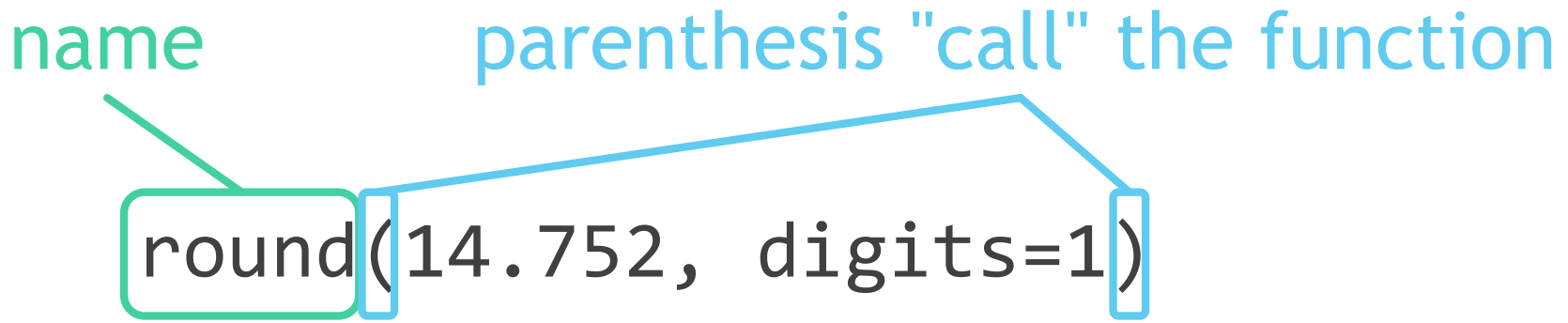
- Matthew Flickinger
- Ph.D. in Biostatistics, UM
- Application Programmer/Analyst Senior Center for Statistical Genetics
- Answer R questions on Stack Overflow
- RStudio certified "tidyverse" trainer

Quick R Facts

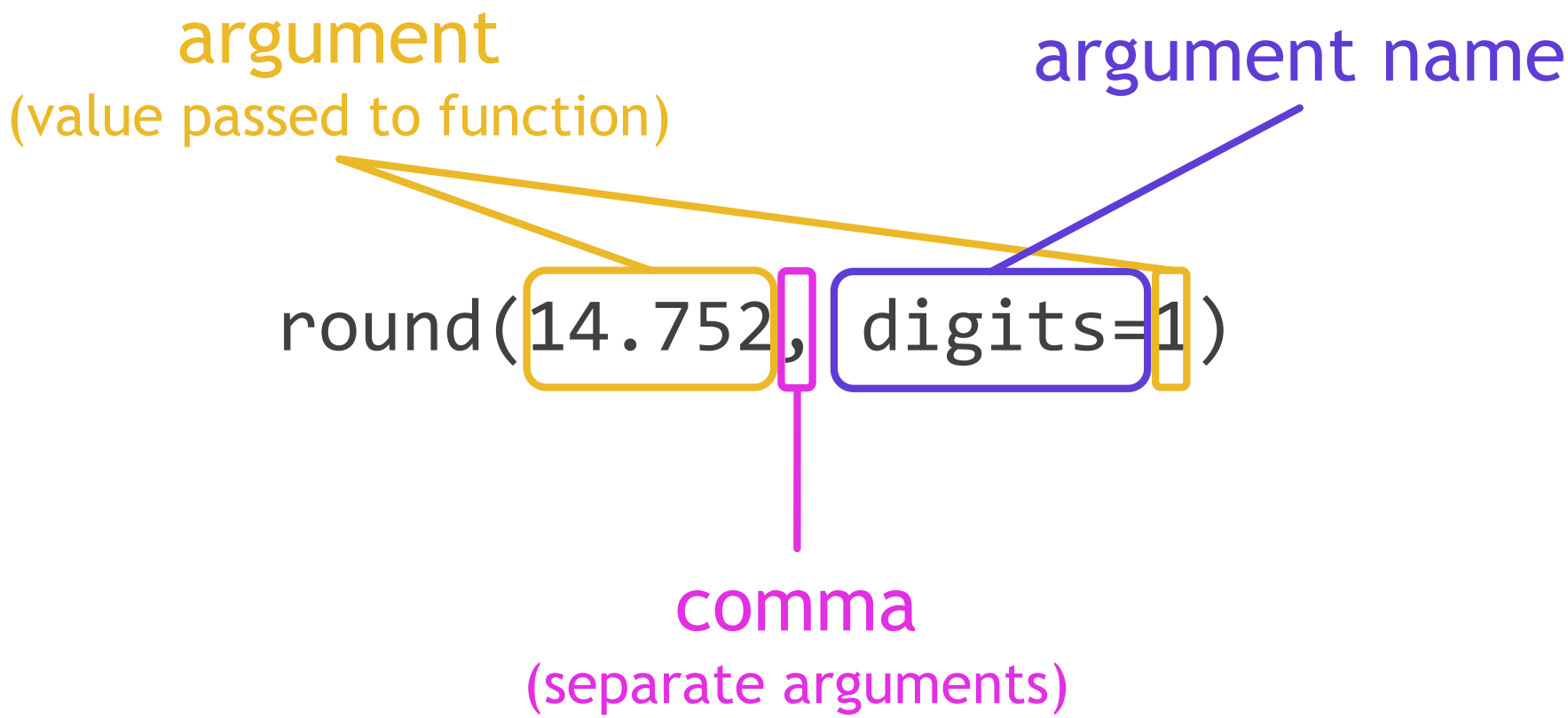
Function Syntax

```
round(14.752, digits=1)
```

Function Syntax



Function Syntax

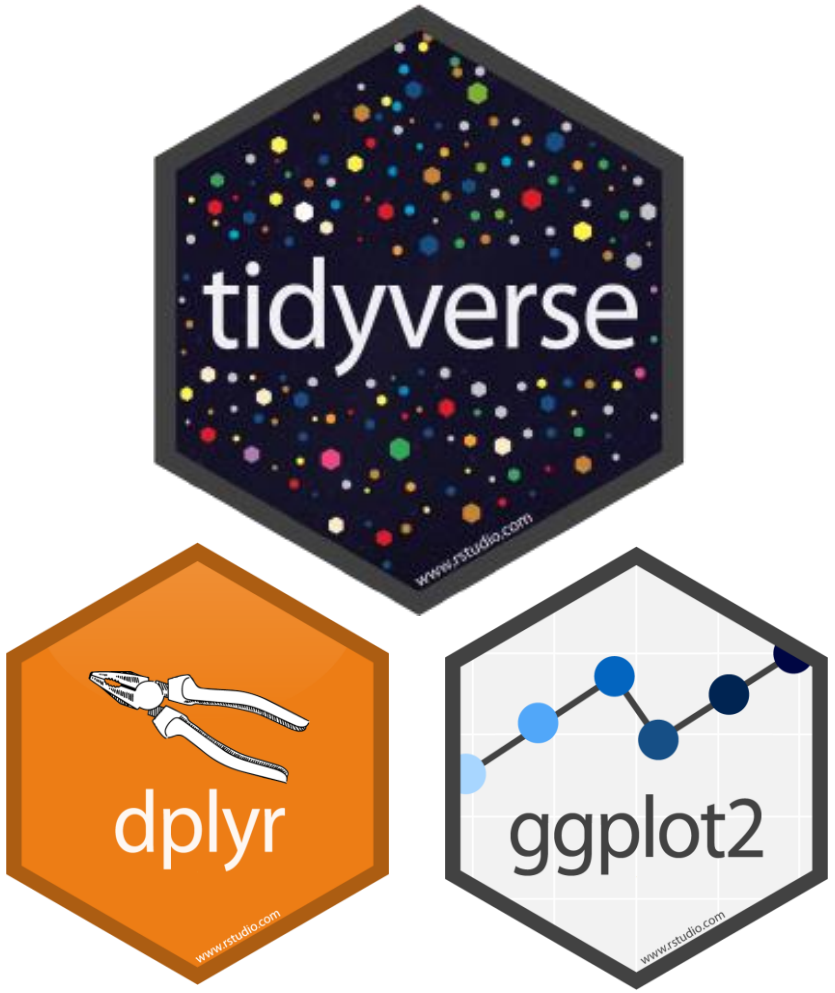


Creating variables

```
fruit <- c("apples", "oranges", "bananas")
primes <- c(1, 2, 3, 5, 7, 11)
numbers <- 1:10
age <- 22
age <- age + 1
```

"<-" assigns a value to a variable

#variable-syntax



Tidyverse

The tidyverse is an opinionated collection of R packages designed for data science. All packages share an underlying design philosophy, grammar, and data structures.

R Packages

Packages add new functions and data into R

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

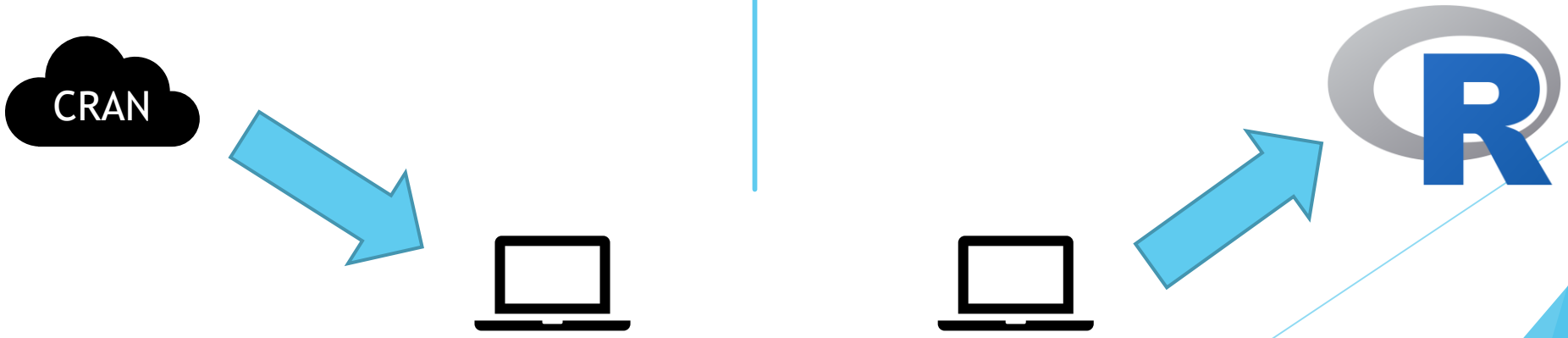
Run **once** per machine

Downloads package to your computer

```
library(dplyr)
```

Run **each time** you start R

Load package into R's "brain"



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 >0.8.1 (released May 14, 2019)

#load-tidyverse

Sample data

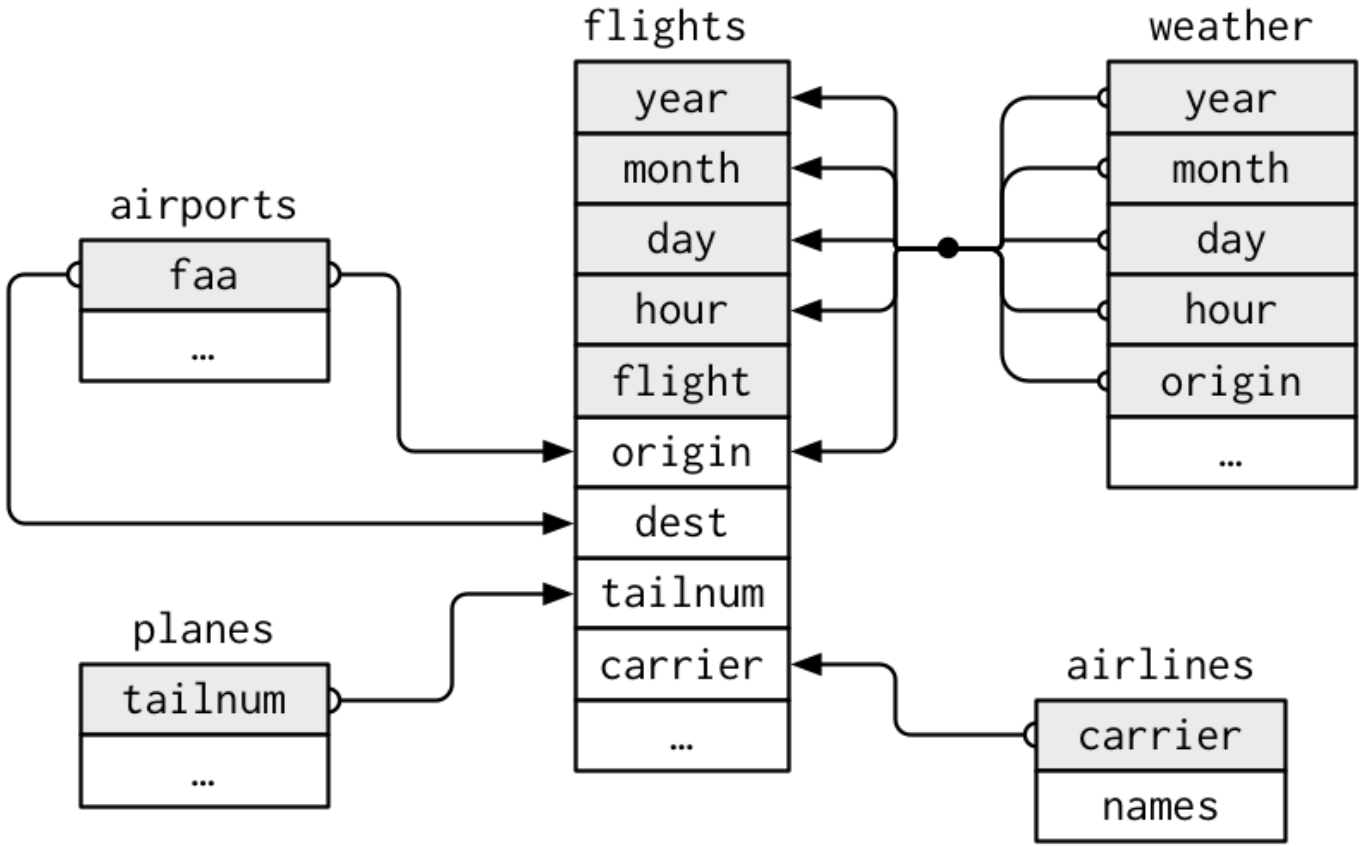
We will be using a data set containing all out-bound flights from NYC in 2013

Available as an R package

```
install.packages("nycflights13")
library(nycflights13)
```

#load-flights

nycflights13 tables



"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 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 argument
- Returns a new data object (never updates/replaces original)
- Specify columns as unquoted strings (symbols)

Filtering Rows

Find all flights to Detroit (DTW) in June (2013)

```
filter(flights, dest=="DTW" & month==6)
```

Base R

```
flights[flights$dest=="DTW" & flights$month==6, ]
subset(flights, dest=="DTW" & month==6)
```

#filter

How about your home airport?

How many flights went to your “home” airport?

```
filter(flights, dest="   ")
                      # ^ put airport code here
```

Or try these

- "HNL" (Honolulu, Hawaii)
- "SLC" (Salt Lake City, Utah)

#filter-poll

Comparisons

`==`	Equal	`month == 6`
`!=`	Not Equal	`month != 6`
`<, <=`	Less than (or equal)	
`>, >=`	Greater than (or equal)	
`&`	And	`month > 2 & month < 7`
`|`	Or	`month < 3 | month > 6`
`!`	Not	
`%in%`	Matches one of	`month %in% c(1, 12)`

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

Can we select AND filter?

```
# Temporary variables
   filtered <- filter(flights, dest=="DTW")
   select(filtered, carrier)
```

```
# Nested calls
   select(filter(flights, dest=="DTW"), carrier)
```

```
# But these can get messy
```

Verb composition with pipes

```
flights %>%
  filter(dest == "DTW") %>%
  select(carrier)
```

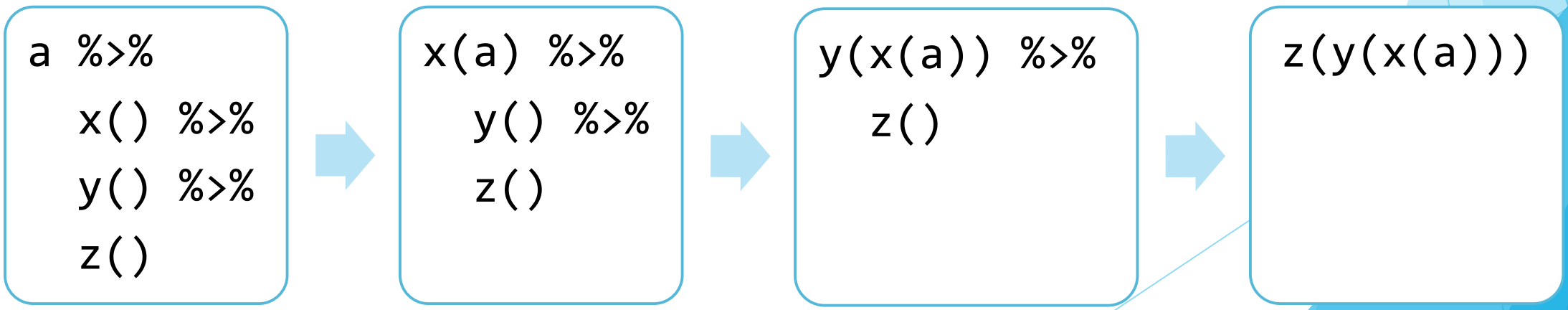
%>% is the "pipe" operator (often read as "then")

RStudio: ctrl-shift-M (win) cmd-shift-M (mac)

Verb composition with pipes

The %>% operator passes the result from the left side to the first argument of the right side

```
a %>% x() %>% y() %>% z()      <->     z(y(x(a))
```



Try using chains

Rewrite the following as a chain

```
round(exp(sin(.5)),2)
# [1] 1.62
```

#pipe-poll

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))
```

Final flight

What is the last day that flight 4401 arrived in Detroit?

```
flights %>%
   filter(       ) %>%
   arrange(       ) %>%
   select(       )
```

#sort-poll

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 arguments to mutate are run 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()`

Try your own summary

Create one statement to calculate the minimum (`min`) and maximum (`max`) departure delay (dep_delay). Be sure to remove rows that have missing values.

```
flights %>%
   filter(      ) %>%
   mutate(   ,   )
```

#mutate-poll

Grouping data

- Often you want to perform summaries for groups of rows at a time
- The `group_by()` function allows you 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
 ...
```

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
 ...
```

Combining group_by() with transformations

`mutate()`

- Will not change the number of rows
- Collapsing functions like max() will return the max for each group
- Keep all existing columns and adds new ones

`summarize()`

- Returns one row per group
- Only returns columns that are used as groups and those new values created from collapsing functions

Count summaries

The `count()` function is a special combination of `summarize()` + `group_by()` to see how often certain values are repeated

```
flights %>%
  count(carrier)
```

```
# A tibble: 16 x 2
   carrier     n
   <chr>   <int>
 1 9E      18460
 2 AA      32729
 3 AS        714
 4 B6      54635
 5 DL      48110
 6 EV      54173
...
```

#count

Special Shortcuts

`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)
```

Summarization exercises

Calculate was the longest arrival delay for each carrier.

Which carrier's longest delay was the shortest compared to all the other carriers?

```
flights %>%
  filter() %>%
  group_by() %>%
  summarize() %>%
  arrange()
```

#summarize-poll

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

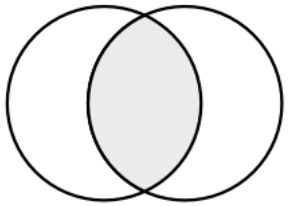
```
> airlines
Source: local data frame [16 x 2]

   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.
6       EV    ExpressJet Airlines Inc.
...
```

Join flights to airlines

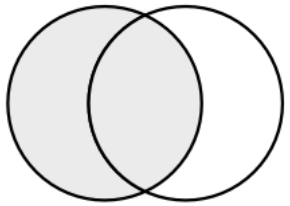
```
> flights %>%
    filter(!is.na(arr_delay)) %>%
    group_by(carrier) %>%
    summarize(avg_arr_delay = mean(arr_delay)) %>%
    left_join(airlines)

Joining by: "carrier"

Source: local data frame [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
5       DL     1.6443409       Delta Air Lines Inc.
```

#left_join

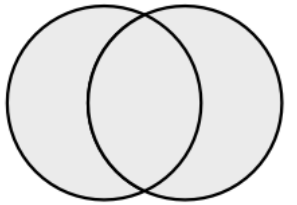
Types of joins (merges)



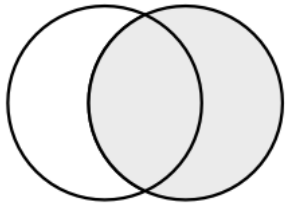
inner_join(x, y)



left_join(x, y)



full_join(x, y)



right_join(x, y)

x

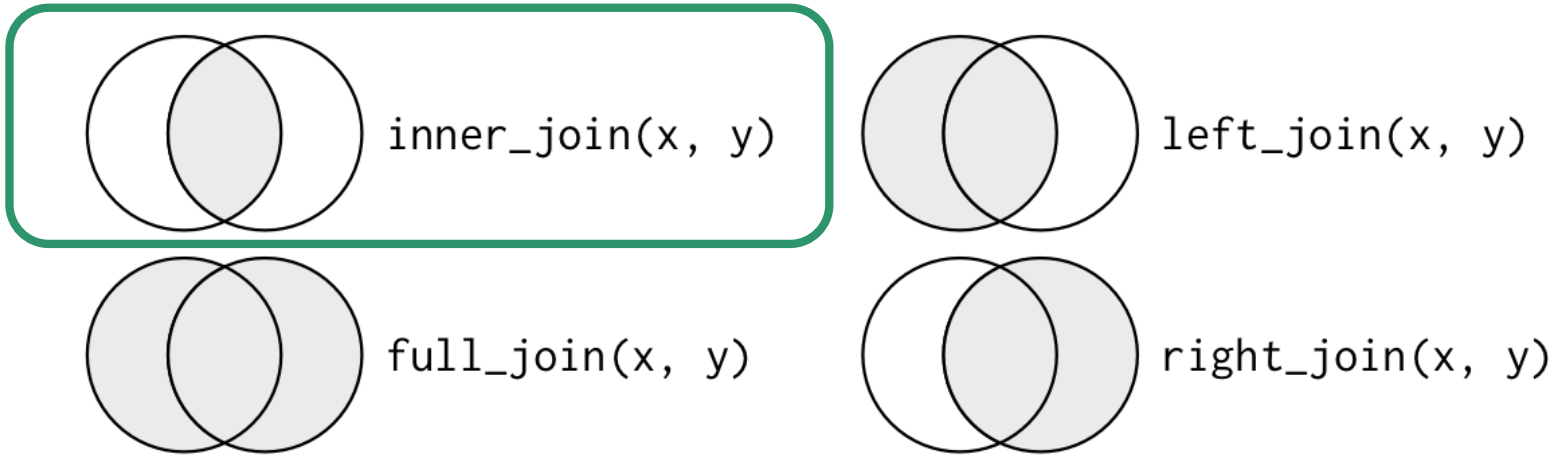
1	x1
2	x2
3	x3

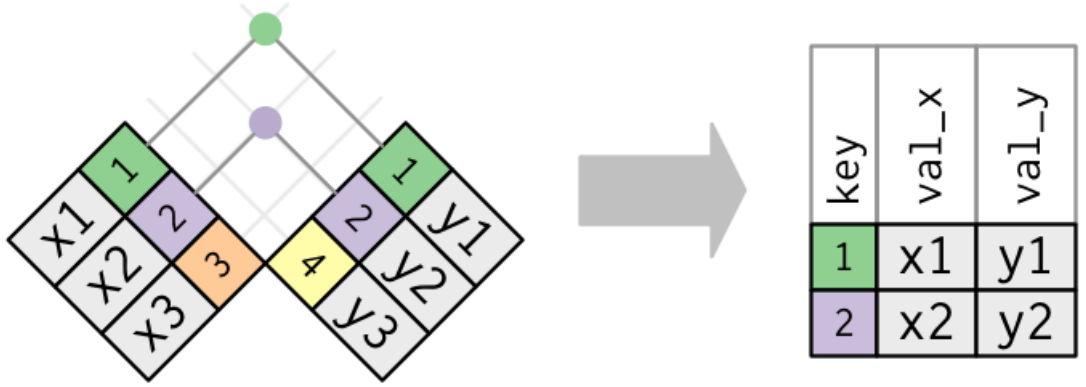
y

1	y1
2	y2
4	y3

Table join images courtesy http://r4ds.had.co.nz/relational-data.html

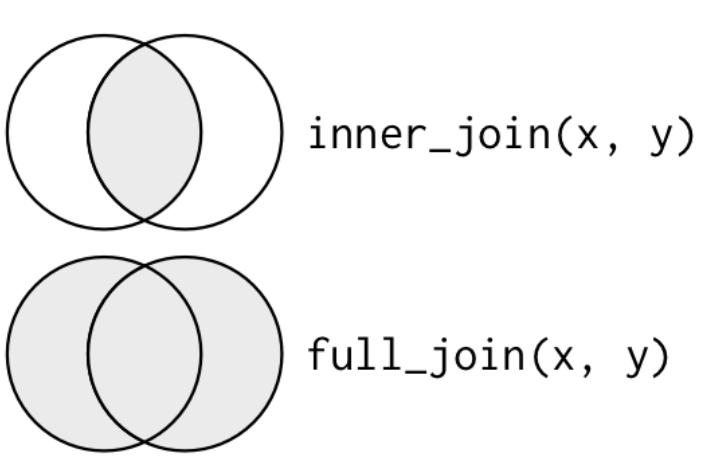
Types of joins (merges)





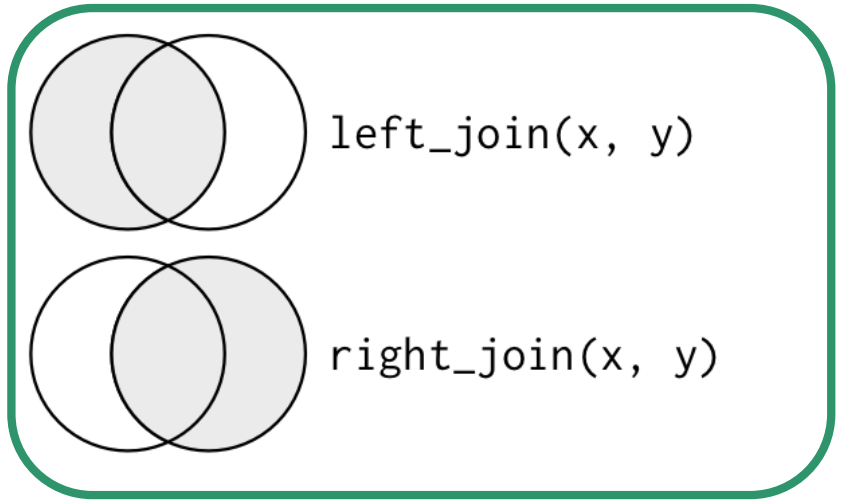
key	val_x	val_y
1	x1	y1
2	x2	y2

Types of joins (merges)



inner_join(x, y)

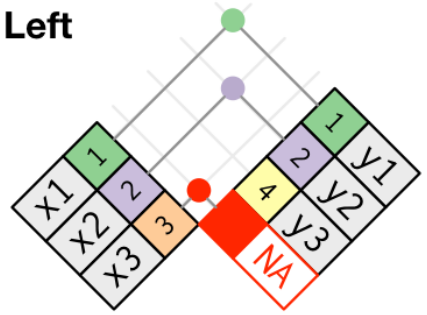
full_join(x, y)

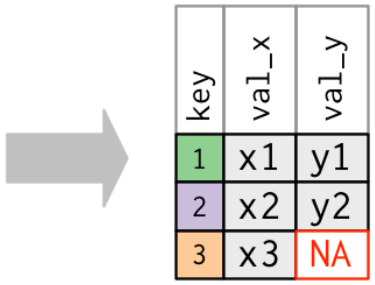


left_join(x, y)

right_join(x, y)

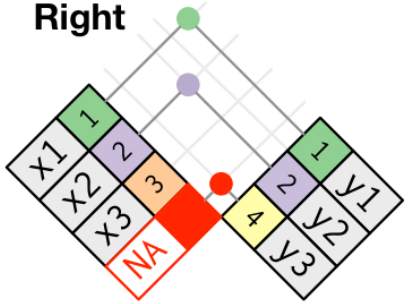
Left





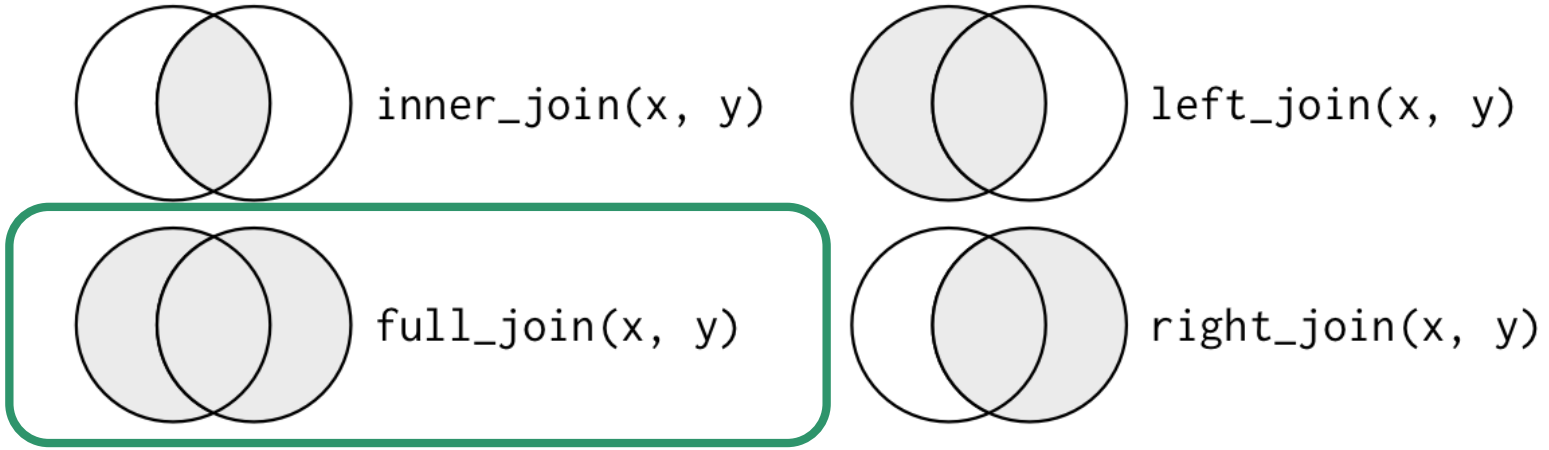
key	val_x	val_y
1	x1	y1
2	x2	y2
3	x3	NA

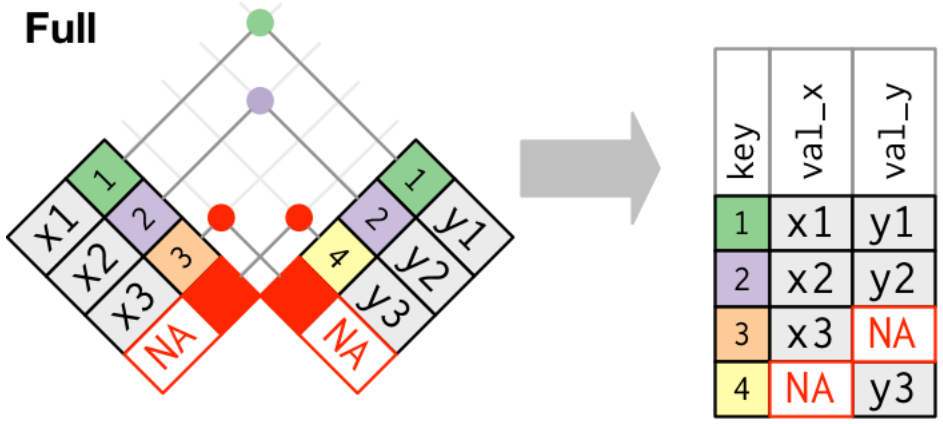
Right



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

Types of joins (merges)





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

What type of join is this?

x

```
# A tibble: 4 x 2
     id count
  <dbl> <dbl>
1     1     5
2     2    12
3     4    14
4     5    16
```

y

```
# A tibble: 4 x 2
     id color
  <dbl> <chr>
1     1 red
2     3 blue
3     4 green
4     5 orange
```

??

```
# A tibble: 4 x 3
     id count color
  <dbl> <dbl> <chr>
1     1     5 red
2     3    NA blue
3     4    14 green
4     5    16 orange
```

A)left_join(x, y)
B)right_join(x, y)
C)full_join(x, y)
D)inner_join(x, y)

Non-merging joins

These "joins" do not add any new columns to your data but are subsetting with multi-column matches (like a multi-column `%in%`)

semi_join()

Only keep rows in left table with matches in right

anti_join()

Drop rows in left table with matches in right

Join by

By default the join commands will join two tables based on all matching column names

```
flights %>% inner_join(planes)
# returns many fewer rows!
```

You can control the joining by specifying the column names

```
flights %>% inner_join(planes, by = "tailnum")
```

Other dplyr functions

Subsetting observations

`top_n()`

Return top or bottom entries

```
flights %>% top_n(3, air_time)
```

`sample_n()`

Randomly choose n rows

```
flights %>% sample_n(3)
```

`distinct()`

Returns unique combinations of values (like `count()` without the count)

```
flights %>% distinct(year, month)
```

#subsetting

Counting functions

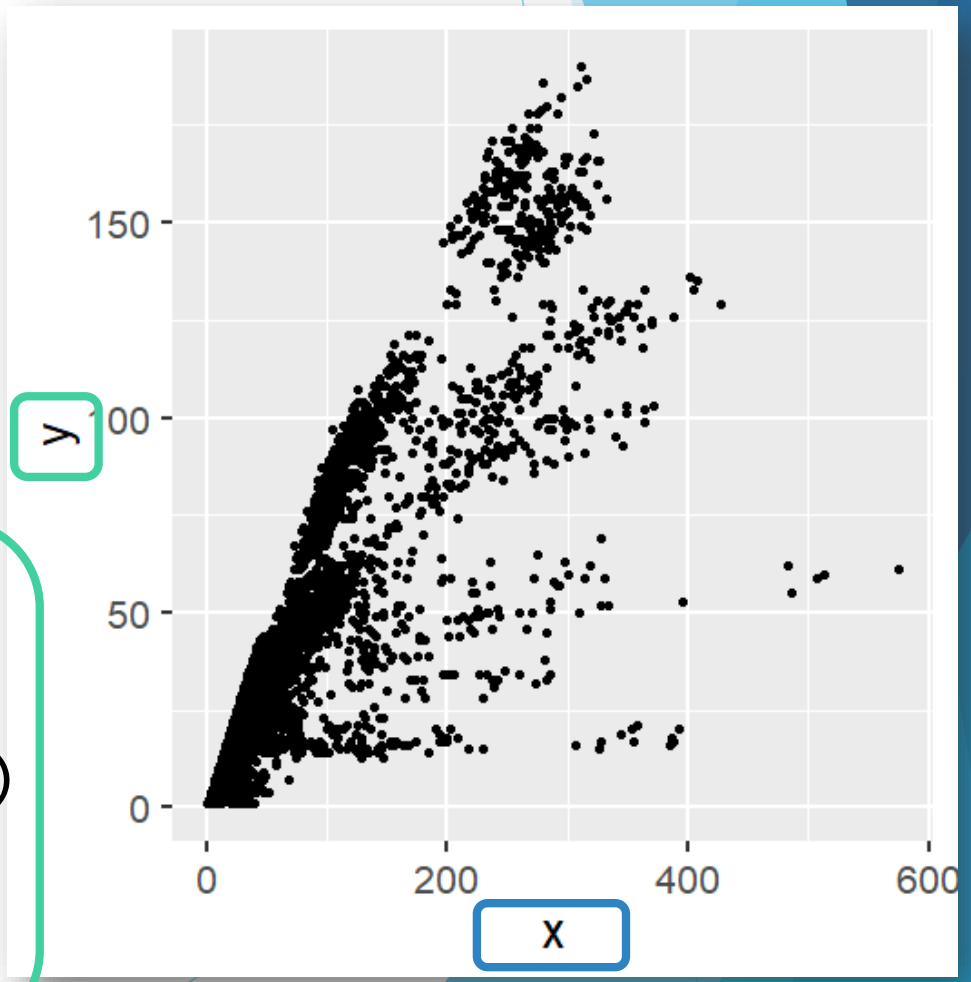
`n()` and `n_distinct()`

- Number of (distinct) values is a vector
- Can only be used within `summarize()`, `mutate()`, `filter()`

```
flights %>%
  group_by(tailnum) %>%
  summarize(
    routes = n_distinct(flight),
    flights = n()
)
```

What values made this plot?

```
flights %>%
  filter(!is.na(tailnum)) %>%
  group_by(tailnum) %>%
  summarize(x=??,
            y=??)
```



x=

A) n_distinct(flight)
B) n()
C) n()
D) n(tailnum)

y=

n()
n_distinct(tailnum)
n_distinct(flight)
n(flight)

Lead/lag functions

"Shift" values so you can compare current value to a previous or following value

```
growth <- tibble(
  age = 2:9,
  height = c(33.7, 37.0, 39.4, 42.2,
             45.5, 47.7, 50.6, 52.7))
growth %>%
  mutate(
    prevh = lag(height),
    nexth = lead(height),
    growth = height-prevh)
```

	age	height	prevh	nexth	growth
	<int>	<dbl>	<dbl>	<dbl>	<dbl>
1	2	33.7	NA	37	NA
2	3	37	33.7	39.4	3.30
3	4	39.4	37	42.2	2.40
4	5	42.2	39.4	45.5	2.8
5	6	45.5	42.2	47.7	3.30
6	7	47.7	45.5	50.6	2.2
7	8	50.6	47.7	52.7	2.90
8	9	52.7	50.6	NA	2.1

#lead-lag

Conditional replacement

The `if_else()` function helps to only change values in certain cases.

```
if_else(<condition>, <if true>, <if false>)
```

```
flights %>%
  mutate(
    real_delay = if_else(arr_delay<0, 0, arr_delay)
  )
```

If you have more cases than TRUE/FALSE, check out the `case_when()` function

#if_else

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
```

#bad-function

Quosures

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

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

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

#quosures

Rename output (:=)

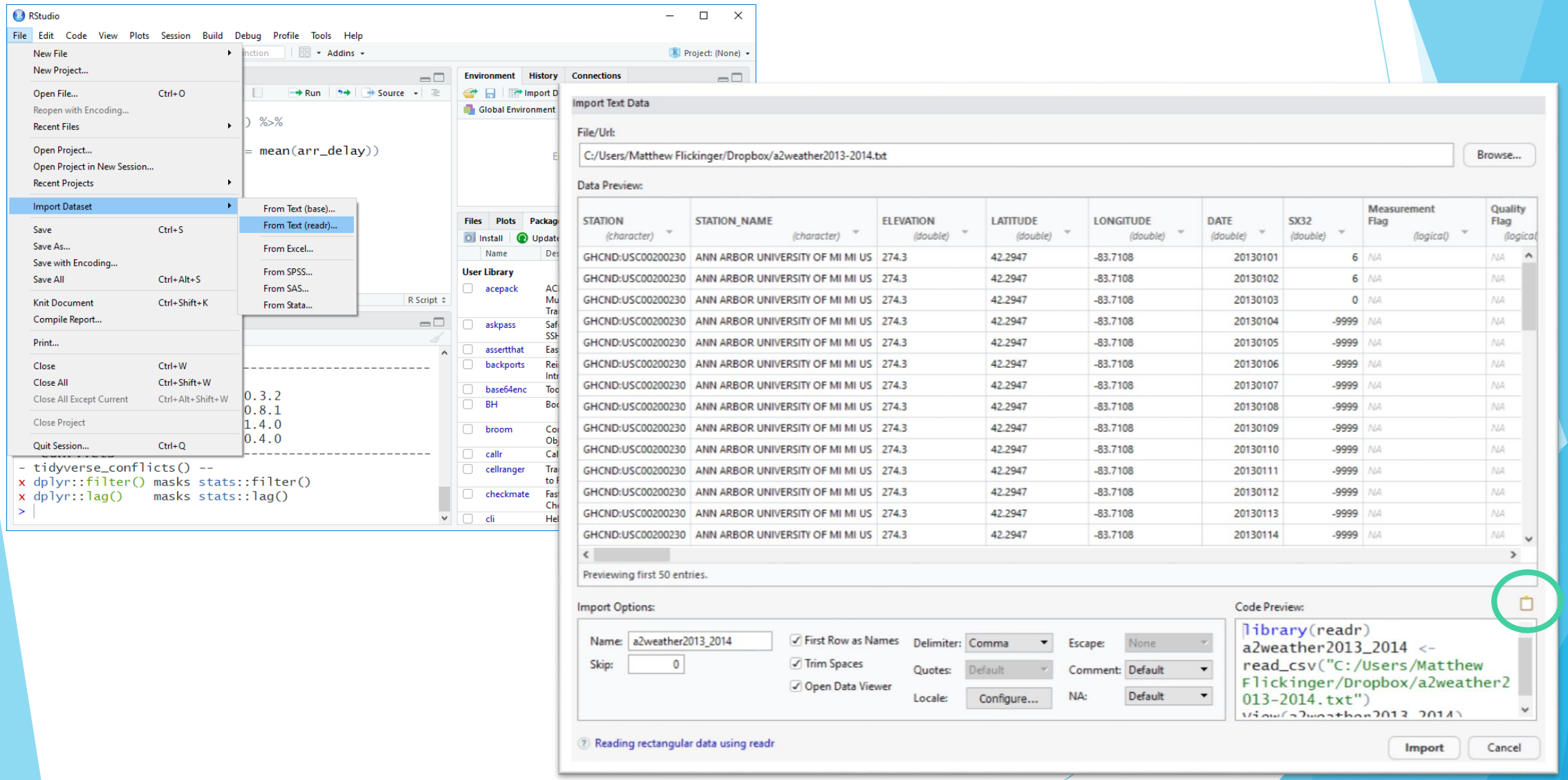
```
h <- function(x) {
  x <- enquo(x)
  outname <- paste(quo_name(x), "delay", sep="_")
  flights %>% group_by(!!x) %>%
  summarize(!!outname := mean(arr_delay, na.rm=T))
}
h(carrier)
```

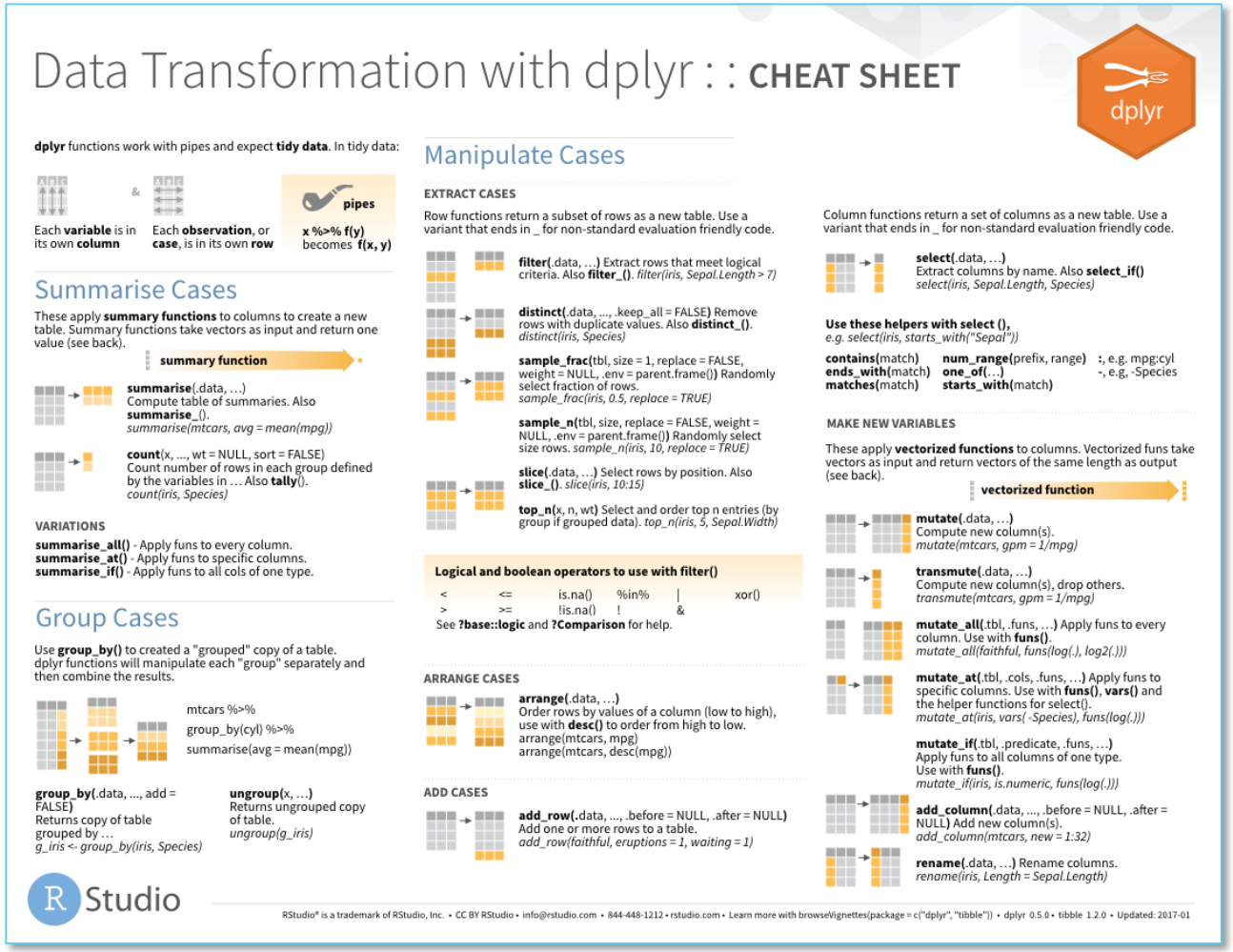
```
# A tibble: 16 x 2
   carrier carrier_delay
     <chr>         <dbl>
 1      9E     7.3796692
 2      AA     0.3642909
 ...
```

#quo_name

Loading your own data

RStudio will write your import code





Data Transformation with dplyr : : CHEAT SHEET

dplyr

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. Summary functions take vectors as input and return one value (see back).

summary function

summarise(.data, …)
Compute table of summaries. Also **summarise_**().
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 created 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. Use a variant that ends in _ for non-standard evaluation friendly code.

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

distinct(.data, ..., .keep_all = FALSE) Remove rows with duplicate values. Also **distinct_()**. *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. Also **slice_()**. *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 (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)

Column functions return a set of columns as a new table. Use a variant that ends in _ for non-standard evaluation friendly code.

select(.data, …)
Extract columns by name. Also **select_if()**
select(iris, Sepal.Length, Species)

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

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

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()**.
mutate_all(faithful, funs(log(.), log2(.)))

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(.)))

mutate_if(.tbl, .predicate, .funs, …)
Apply funs to all columns of one type. Use with **funs()**.
mutate_if(iris, is.numeric, funs(log(.)))

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

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

RStudio

RStudio® is a trademark of RStudio, Inc. • CC BY RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more with browseVignettes(package = c("dplyr", "tibble")) • dplyr 0.5.0 • tibble 1.2.0 • Updated: 2017-01

RStudio "Data Transformation Cheat Sheet"

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

nycflights13 tables