More "dplyr" and "tidyr"

Tidyverse

Gaston Sanchez

STAT 33B, Fall 2025

About

In this slides we review more functions from the R packages `"dplyr"`, and `"tidyr"`, which are part of the `"tidyverse"`.

Pipe Operators

Pipe Operators: |> and %>%

- This section introduces the pipe operators, denoted as |> and also as %>% , which allow you to write function calls in a more human-readable way.
- These operators are heavily used among the ecosystem of "tidyverse" packages, and they are becoming more common in traditional R code.
- Technically speaking, %>% is known as the *magrittr* pipe operator (from its homonym package, introduced in 2014).
- In turn, |> is the *base R* pipe operator (introduced in May 2021 with R version 4.1.0.)

About pipe operators

A pipe operator lets you write `f(x)` as:

```
x |> f()
```

Likewise, you can rewrite `g(x, y)` as:

```
x %>% g(y)
```

Example 1: use of pipes

```r
x = c(2, 4, 6)

# without the pipe
mean(x)
```

```
[1] 4
```

```r
# with the pipe
x |> mean()
```

```
[1] 4
```

Example 2: use of pipes

```r
x = c(2, 4, 6, NA)

# without the pipe
mean(x, na.rm = TRUE)
```

```
[1] 4
```

```r
# with the pipe
x |> mean(na.rm = TRUE)
```

```
[1] 4
```

Example 3: Multiple steps (no pipe)

- Generate n $= 10$ random numbers with `runif()`,
- Round them to 2 decimal digits,
- Take their absolute values,
- And add them all up.

Example 3: Multiple steps (no pipe)

- Generate n = 10 random numbers with `runif()`,
- Round them to 2 decimal digits,
- Take their absolute values,
- And add them all up.

```
# step-by-step (no pipe)
set.seed(12345)

n = 10
x1 = runif(n, min = -3, max = 3)
x2 = round(x1, 2)
x3 = abs(x2)
x4 = sum(x3)
x4
```

```
[1] 15.15
```

Example 3: Multiple steps (with pipes)

- Generate n = 10 random numbers with `runif()`,
- Round them to 2 decimal digits,
- Take their absolute values,
- And add them all up.

```
# pipeline
set.seed(12345)

10 |>
  runif(min = -3, max = 3) |>
  round(2) |>
  abs() |>
  sum()
```

```
[1] 15.15
```

Data pipelines

dplyr functional calls

An "ugly" side of dplyr is that if you want to do many operations at once, it does not lead to particularly elegant code.

You either have to do computations, step-by-step, with separate commands.

Or you have to wrap several function calls inside each other (making your code hard to read)

Step-by-step commands

dat

name	gender	height
Anakin	male	1.88
Padme	female	1.65
Luke	male	1.72
Leia	female	1.50

dat3

gender	avg	sd
male	1.8	0.113
female	1.58	0.106

```
dat1 = group_by(dat, gender)
dat2 = summarise(dat1,
  avg = mean(height), sd = sd(height))
dat3 = arrange(dat2, desc(avg))
dat3
```

Inside-out commands

dat

name	gender	height
Anakin	male	1.88
Padme	female	1.65
Luke	male	1.72
Leia	female	1.50

gender	avg	sd
male	1.8	0.113
female	1.58	0.106

```
arrange(
  summarise(group_by(dat, gender),
    avg = mean(height),
    sd = sd(height)),
  desc(avg))
```

What about using a pipeline?

Commands in a pipeline

dat

name	gender	height
Anakin	male	1.88
Padme	female	1.65
Luke	male	1.72
Leia	female	1.50

dat3

gender	avg	sd
male	1.8	0.113
female	1.58	0.106

```
dat3 = dat |>
   group_by(gender) |>
   summarise(avg = mean(height),
             sd = sd(height)) |>
   arrange(desc(avg))
```

Pipeline example

dat

name	gender	height
Anakin	male	1.88
Padme	female	1.65
Luke	male	1.72
Leia	female	1.50

name	gender	height
Padme	female	1.65
Leia	female	1.50

```
filter(dat, gender == "female")

# is equivalent to
dat |> filter(gender == "female")
```

Pipeline example

dat

name	gender	height
Anakin	male	1.88
Padme	female	1.65
Luke	male	1.72
Leia	female	1.50

name	gender
Anakin	male
Padme	female
Luke	male
Leia	female

```
select(dat, name:gender)

# is equivalent to
dat |> select(name:gender)
```

Pipeline example

dat

name	gender	height
Anakin	male	1.88
Padme	female	1.65
Luke	male	1.72
Leia	female	1.50

name	gender	height
Anakin	male	1.88
Luke	male	1.72
Padme	female	1.65
Leia	female	1.50

```
arrange(dat, desc(height))

# is equivalent to
dat |> arrange(desc(height))
```

Pipeline example

dat

name	gender	height
Anakin	male	1.88
Padme	female	1.65
Luke	male	1.72
Leia	female	1.50

dat3

gender	avg	sd
male	1.8	0.113
female	1.58	0.106

```
dat3 = dat |>
   group_by(gender) |>
   summarise(avg = mean(height),
             sd = sd(height)) |>
   arrange(desc(avg))
```

Merging tables with `joins()`

Motivation

tbl1

id	year	coffee
Luke	1	no
Leia	3	yes
Han	4	yes

tbl2

id	gpa	lunch
Padme	3.9	pizza
Leia	4.0	tacos
Luke	3.7	burrito
Obi-Wan	3.8	pad thai

Toy Example

Consider the following tables `tbl1` and `tbl2` that share a common `id` column

```
tbl1 <- data.frame(
  id = c('Luke', 'Leia', 'Han'),
  year = c(1, 3, 4),
  coffee = c('no', 'yes', 'yes')
)

tbl2 <- data.frame(
  id = c('Padme', 'Leia', 'Luke', 'Obi-Wan'),
  gpa = c(3.9, 4.0, 3.7, 3.8),
  lunch = c('pizza', 'tacos', 'burrito', 'pad thai')
)
```

"dplyr" merging or join functions

- `full_join()`
- `inner_join()`
- `left_join()`
- `right_join()`
- `anti_join()`
- `semi_join()`

full_join()

tbl1

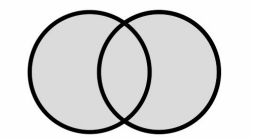
id	year	coffee
Luke	1	no
Leia	3	yes
Han	4	yes

tbl2

id	gpa	lunch
Padme	3.9	pizza
Leia	4.0	tacos
Luke	3.7	burrito
Obi-Wan	3.8	pad thai

```
# keeps all observations in tbl1 and tbl2
full_join(tbl1, tbl2, by = "id")
```

id	year	coffee	gpa	lunch
Luke	1	no	3.7	burrito
Leia	3	yes	4.0	tacos
Han	4	yes	NA	NA
Padme	NA	NA	3.9	pizza
Obi-Wan	NA	NA	3.8	pad thai



inner_join(): example

tbl1

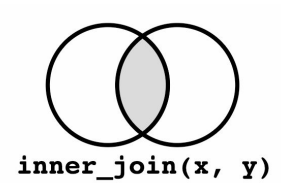
id	year	coffee
Luke	1	no
Leia	3	yes
Han	4	yes

tbl2

id	gpa	lunch
Padme	3.9	pizza
Leia	4.0	tacos
Luke	3.7	burrito
Obi-Wan	3.8	pad thai

```
# keeps obs in tbl1 that have matching key in tbl2
inner_join(tbl1, tbl2, by = "id")
```

id	year	coffee	gpa	lunch
Luke	1	no	3.7	burrito
Leia	3	yes	4.0	tacos



left_join(): example

tbl1

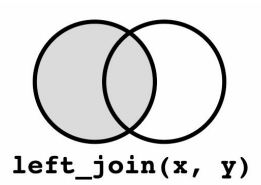
id	year	coffee
Luke	1	no
Leia	3	yes
Han	4	yes

tbl2

id	gpa	lunch
Padme	3.9	pizza
Leia	4.0	tacos
Luke	3.7	burrito
Obi-Wan	3.8	pad thai

```
# keeps all observations in tbl1
left_join(tbl1, tbl2, by = "id")
```

id	year	coffee	gpa	lunch
Luke	1	no	3.7	burrito
Leia	3	yes	4.0	tacos
Han	4	yes	NA	NA



right_join(): example

tbl1

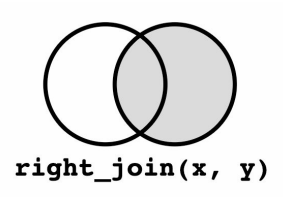
id	year	coffee
Luke	1	no
Leia	3	yes
Han	4	yes

tbl2

id	gpa	lunch
Padme	3.9	pizza
Leia	4.0	tacos
Luke	3.7	burrito
Obi-Wan	3.8	pad thai

```
# keeps all observations in tbl2
right_join(tbl1, tbl2, by = "id")
```

id	year	coffee	gpa	lunch
Luke	1	no	3.7	burrito
Leia	3	yes	4.0	tacos
Padme	NA	NA	3.9	pizza
Obi-Wan	NA	NA	3.8	pad thai



anti_join(): example 1

tbl1

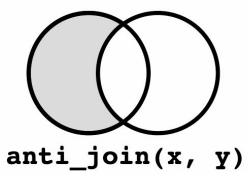
id	year	coffee
Luke	1	no
Leia	3	yes
Han	4	yes

tbl2

id	gpa	lunch
Padme	3.9	pizza
Leia	4.0	tacos
Luke	3.7	burrito
Obi-Wan	3.8	pad thai

```
# return rows from tbl1 without a match in tbl2
anti_join(tbl1, tbl2, by = "id")
```

id	year	coffee
Han	4	yes



anti_join(): example 2

tbl1

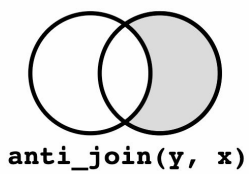
id	year	coffee
Luke	1	no
Leia	3	yes
Han	4	yes

tbl2

id	gpa	lunch
Padme	3.9	pizza
Leia	4.0	tacos
Luke	3.7	burrito
Obi-Wan	3.8	pad thai

```
# return rows from tbl2 without a match in tbl1
anti_join(tbl2, tbl1, by = "id")
```

id	gpa	lunch
Padme	3.9	pizza
Obi-Wan	3.8	pad thai



semi_join(): example 1

tbl1

id	year	coffee
Luke	1	no
Leia	3	yes
Han	4	yes

tbl2

id	gpa	lunch
Padme	3.9	pizza
Leia	4.0	tacos
Luke	3.7	burrito
Obi-Wan	3.8	pad thai

```
# return rows from tbl1 with a match in tbl2
semi_join(tbl1, tbl2, by = "id")
```

id	year	coffee
Luke	1	no
Leia	3	yes

semi_join(): example 2

tbl1

id	year	coffee
Luke	1	no
Leia	3	yes
Han	4	yes

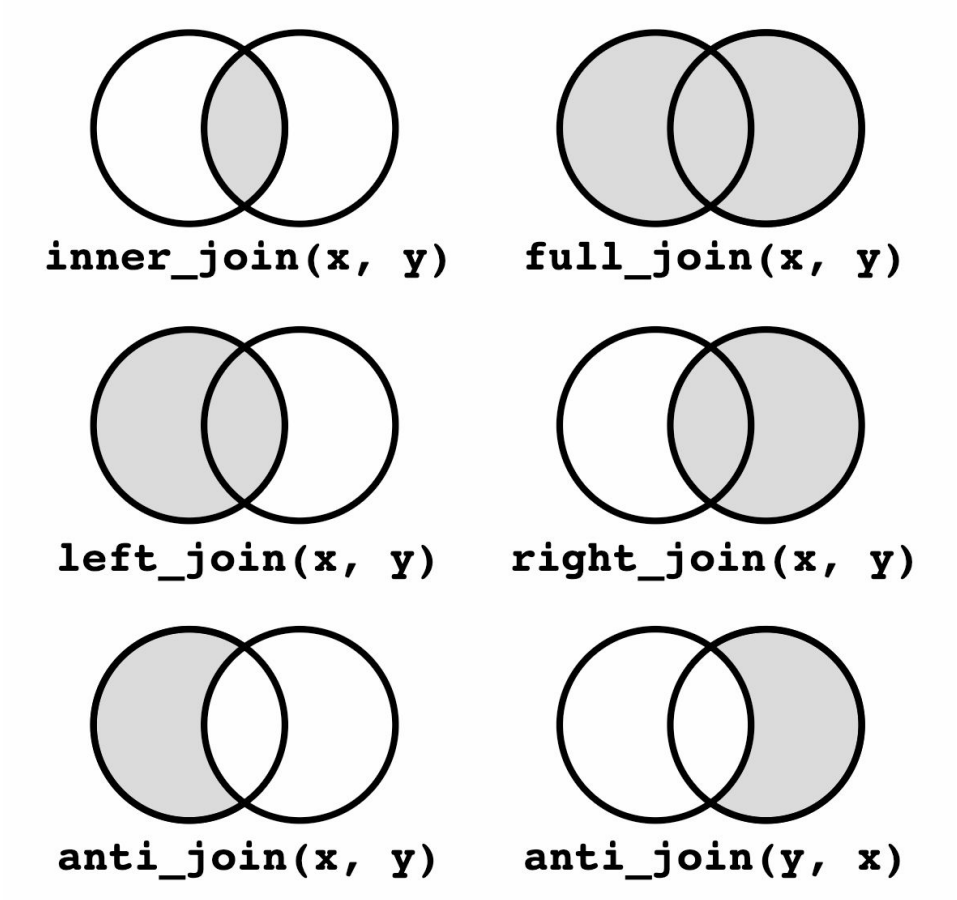
tbl2

id	gpa	lunch
Padme	3.9	pizza
Leia	4.0	tacos
Luke	3.7	burrito
Obi-Wan	3.8	pad thai

```
# return rows from tbl2 with a match in tbl1
semi_join(tbl2, tbl1, by = "id")
```

id	gpa	lunch
Leia	4.0	tacos
Luke	3.7	burrito

Joins


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

What if tables had keys with different names?

Tables with different key names

tbl1

id1	year	coffee
Luke	1	no
Leia	3	yes
Han	4	yes

tbl2

id2	gpa	lunch
Padme	3.9	pizza
Leia	4.0	tacos
Luke	3.7	burrito
Obi-Wan	3.8	pad thai

Tables with different key (`id`) names

```
tbl1 <- data.frame(
  id1 = c('Luke', 'Leia', 'Han'),
  year = c(1, 3, 4),
  coffee = c('no', 'yes', 'yes')
)

tbl2 <- data.frame(
  id2 = c('Padme', 'Leia', 'Luke', 'Obi-Wan'),
  gpa = c(3.9, 4.0, 3.7, 3.8),
  lunch = c('pizza', 'tacos', 'burrito', 'pad thai')
)
```

Example

tbl1

id1	year	coffee
Luke	1	no
Leia	3	yes
Han	4	yes

tbl2

id2	gpa	lunch
Padme	3.9	pizza
Leia	4.0	tacos
Luke	3.7	burrito
Obi-Wan	3.8	pad thai

```
# keeps all observations in tbl1 and tbl2
full_join(tbl1, tbl2, join_by("id1" == "id2"))
```

id1	year	coffee	gpa	lunch
Luke	1	no	3.7	burrito
Leia	3	yes	4.0	tacos
Han	4	yes	NA	NA
Padme	NA	NA	3.9	pizza
Obi-Wan	NA	NA	3.8	pad thai

Example

tbl1

id1	year	coffee
Luke	1	no
Leia	3	yes
Han	4	yes

tbl2

id2	gpa	lunch
Padme	3.9	pizza
Leia	4.0	tacos
Luke	3.7	burrito
Obi-Wan	3.8	pad thai

```
# equivalent command
full_join(tbl1, tbl2, by = c("id1" = "id2"))
```

id1	year	coffee	gpa	lunch
Luke	1	no	3.7	burrito
Leia	3	yes	4.0	tacos
Han	4	yes	NA	NA
Padme	NA	NA	3.9	pizza
Obi-Wan	NA	NA	3.8	pad thai

Pivot Tables with `tidyr`

Pivot table functions

`"tidyr"` provides two pivoting table functions:

- `pivot_longer()`
- `pivot_wider()`

More information at:

https://tidyr.tidyverse.org/articles/pivot.html

Pivot Table to Long Format

Wide Table (to be pivoted to long format)

```
# Name, Major and GPAs of three students
gpa_wide = tibble(
  name = c("Luke", "Leia", "Han"),
  major = c("Jedi studies", "Imperial Politics", "Galactic Commerce"),
  freshman = c(3.5, 3.9, 3.0),
  sophomore = c(3.7, 4.0, 2.9),
  junior = c(3.8, 4.0, 2.8))

gpa_wide
```

```
# A tibble: 3 x 5
  name  major              freshman sophomore junior
  <chr> <chr>                 <dbl>     <dbl>  <dbl>
1 Luke  Jedi studies            3.5       3.7    3.8
2 Leia  Imperial Politics       3.9       4      4
3 Han   Galactic Commerce       3         2.9    2.8
```

Goal: Stack all the gpa values from columns `freshman`, `sophomore`, and `junior` into a single column.

Pivot to long ("tall") table

Stack all the gpa values from columns `freshman`, `sophomore`, and `junior` into a single column.

```
# A tibble: 9 x 4
  name  major              year        gpa
  <chr> <chr>              <chr>     <dbl>
1 Luke  Jedi studies       freshman    3.5
2 Luke  Jedi studies       sophomore   3.7
3 Luke  Jedi studies       junior      3.8
4 Leia  Imperial Politics  freshman    3.9
5 Leia  Imperial Politics  sophomore   4
6 Leia  Imperial Politics  junior      4
7 Han   Galactic Commerce  freshman    3
8 Han   Galactic Commerce  sophomore   2.9
9 Han   Galactic Commerce  junior      2.8
```

Pivot table to long ("tall") format

```
gpa_long_format = pivot_longer(
  data = gpa_wide, # table to pivot
  cols = c(freshman, sophomore, junior), # columns to pivot to long format
  names_to = "year", # name of new column from 'cols'
  values_to =  "gpa") # name of column with stacked values

gpa_long_format
```

Pivot Table to Wide Format

What if we want to pivot to wide format?

Goal: Revert the long table into wide format.

```
gpa_wide_format = pivot_wider(
  data = gpa_long_format,
  names_from = year,
  values_from = gpa)

gpa_wide_format
```

```
# A tibble: 3 x 5
  name  major             freshman sophomore junior
  <chr> <chr>                <dbl>     <dbl>  <dbl>
1 Luke  Jedi studies           3.5       3.7    3.8
2 Leia  Imperial Politics      3.9       4      4
3 Han   Galactic Commerce      3         2.9    2.8
```