

Welcome to the tidyverse

Week 2

AEM 2850 / 5850 : R for Business Analytics

Cornell Dyson

Fall 2025

Acknowledgements: Grant McDermott, Allison Horst

Announcements

If you are new to the class, please review canvas and the **course site**

Prelim dates: Oct 2 and Dec 4 at 7:30pm

Other reminders:

- Bookmark **the schedule** and visit often
- Submit assignments via canvas by Monday at 11:59pm going forward

Office hours

TA office hours:

- Mondays 2:00pm - 6:00pm in Warren 370
- Fridays 1:00pm - 2:00pm in Warren B05

My office hours:

- Tuesdays 11:30 - 12:30 in Warren 464
- Thursday afternoons by appointment: aem2850.youcanbook.me

Questions before we get started?

Plan for today

Generative AI policy and Prologue

Tidyverse basics

- data frames and pipes

Data transformation with dplyr

- select
- filter
- arrange
- mutate
- summarize
- other goodies

Generative AI policy

Generative AI policy

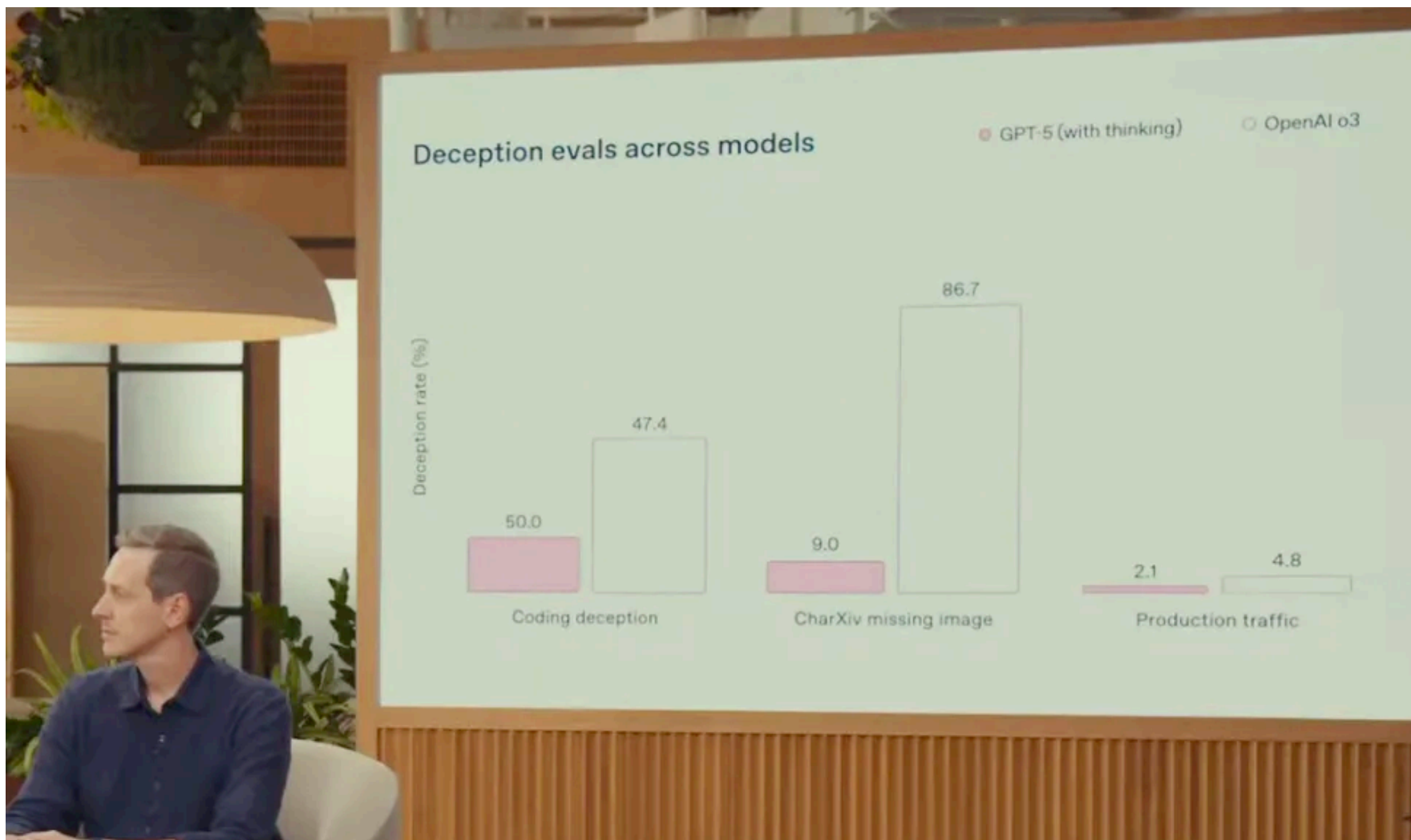
Generative AI (GenAI) tools are very powerful assistants for widely used programming languages like R

You may use GenAI tools in this course **unless explicitly forbidden** for a specific assignment

However, be aware of the limitations of GenAI tools:

- GenAI tools can be misleading
- GenAI tools often propose code that does not function
- GenAI code can be difficult to debug if you do not understand the fundamentals

Generative AI tools can be misleading



Generative AI example

Does anyone know where this quote is from?

A-B-C. A-always, B-be, C-closing. Always be closing! Always be closing!!

A-I-D-A. Attention, interest, decision, action. Attention -- do I have your attention? Interest -- are you interested? I know you are because it's fuck or walk. You close or you hit the bricks! Decision -- have you made your decision for Christ?!! And action.

A-I-D-A; get out there!! You got the prospects comin' in; you think they came in to get out of the rain? Guy doesn't walk on the lot unless he wants to buy. Sitting out there waiting to give you their money! Are you gonna take it?

Generative AI example

ChatGPT >  

Translate the text below from English to Mandarin, without using any characters from the English alphabet:

A-B-C. A-always, B-be, C-closing. Always be closing! Always be closing!! A-I-D-A. Attention, interest, decision, action. Attention -- do I have your attention? Interest -- are you interested? I know you are because it's fuck or walk. You close or you hit the bricks! Decision -- have you made your decision for Christ?!! And action. A-I-D-A; get out there!! You got the prospects comin' in; you think they came in to get out of the rain? Guy doesn't walk on the lot unless he wants to buy. Sitting out there waiting to give you their money! Are you gonna take it?

 甲-乙-丙。甲-始终，乙-必须，丙-成交。始终要成交！始终要成交！！

甲-戊-丁-己。注意、兴趣、决定、行动。

注意——我吸引你的注意力了吗？

兴趣——你感兴趣吗？我知道你感兴趣，因为这不是成功就是失败！要么成交，要么滚蛋！

决定——你已经做出最终决定了吗？！！

行动！

甲-戊-丁-己，马上行动！！客户已经来了，你以为他们是为了避雨才进来的？没人走进店里，除非他想买东西！他们就坐在那里，等着把钱交给你！你要不要拿下这笔交易？

Generative AI example

ChatGPT > ↑ 🔄

Translate the text below from English to Mandarin, without using any characters from the English alphabet:

A-B-C. A-always, B-be, C-closing. Always be closing! Always be closing!! A-I-D-A. Attention, interest, decision, action. Attention -- do I have your attention? Interest -- are you interested? I know you are because it's fuck or walk. You close or you hit the bricks! Decision -- have you made your decision for Christ?!! And action. A-I-D-A; get out there!! You got the prospects comin' in; you think they came in to get out of the rain? Guy doesn't walk on the lot unless he wants to buy. Sitting out there waiting to give you their money! Are you gonna take it?

 甲-乙-丙。甲-始终，乙-必须，丙-成交。始终要成交！始终要成交！！

甲-戊-丁-己。注意、兴趣、决定、行动。

注意——我吸引你的目光了吗？

兴趣——你有兴趣吗？我知道你有，因为不是赢就是输！要么成交，要么走人！

决定——你已经下定决心了吗？！！

行动！

甲-戊-丁-己，马上去做！！机会已经来了，你以为他们是为了避雨才进来的？没有人踏进这里，除非他想买东西！他们就在那儿，等着把钱交给你！你会接受吗？

Which answer is better?

ChatGPT >  

Translate the text below from English to Mandarin, without using any characters from the English alphabet:

A-B-C. A-always, B-be, C-closing. Always be closing! Always be closing!! A-I-D-A. Attention, interest, decision, action. Attention -- do I have your attention? Interest -- are you interested? I know you are because it's fuck or walk. You close or you hit the bricks! Decision -- have you made your decision for Christ?!! And action. A-I-D-A; get out there!! You got the prospects comin' in; you think they came in to get out of the rain? Guy doesn't walk on the lot unless he wants to buy. Sitting out there waiting to give you their money! Are you gonna take it?

 甲-乙-丙。甲-始终，乙-必须，丙-成交。始终要成交！始终要成交！！

甲-戊-丁-己。注意、兴趣、决定、行动。

注意——我吸引|你的注意力了吗？
兴趣——你感兴趣吗？我知道你感兴趣，因为这不是成功就是失败！要么成交，要么滚蛋！
决定——你已经做出最终决定了吗？！！
行动！

甲-戊-丁-己，马上去行动！！客户已经来了，你以为他们是为了避雨才进来的？没人走进店里，除非他想买东西！他们就坐在那里，等着把钱交给你！你要不要拿下这笔交易？

ChatGPT >  

Translate the text below from English to Mandarin, without using any characters from the English alphabet:

A-B-C. A-always, B-be, C-closing. Always be closing! Always be closing!! A-I-D-A. Attention, interest, decision, action. Attention -- do I have your attention? Interest -- are you interested? I know you are because it's fuck or walk. You close or you hit the bricks! Decision -- have you made your decision for Christ?!! And action. A-I-D-A; get out there!! You got the prospects comin' in; you think they came in to get out of the rain? Guy doesn't walk on the lot unless he wants to buy. Sitting out there waiting to give you their money! Are you gonna take it?

 甲-乙-丙。甲-始终，乙-必须，丙-成交。始终要成交！始终要成交！！

甲-戊-丁-己。注意、兴趣、决定、行动。

注意——我吸引|你的目光了吗？
兴趣——你有兴趣吗？我知道你有，因为不是赢就是输！要么成交，要么走人！
决定——你已经下定决心了吗？！！
行动！

甲-戊-丁-己，马上去做！！机会已经来了，你以为他们是为了避雨才进来的？没有人踏进这里，除非他想买东西！他们就在那儿，等着把钱交给你！你会接受吗？

I **strongly** encourage you to focus on learning essential syntax and then, later, consider using these tools as helpers (or not at all), rather than relying on them as replacements for your own knowledge and function

Prologue

How we feel about the course, in a few words

I'm excited to learn more about R!

I think the pace is just right!

Slightly nervous

Where we're from

How can we use your 89 survey responses to construct our origin story without having to read them one by one?

- **Caveat:** This is based on submissions as of 13:25 Monday

We can use R for this!

I won't go through details now

By the end of the course you should be able to do this kind of thing easily

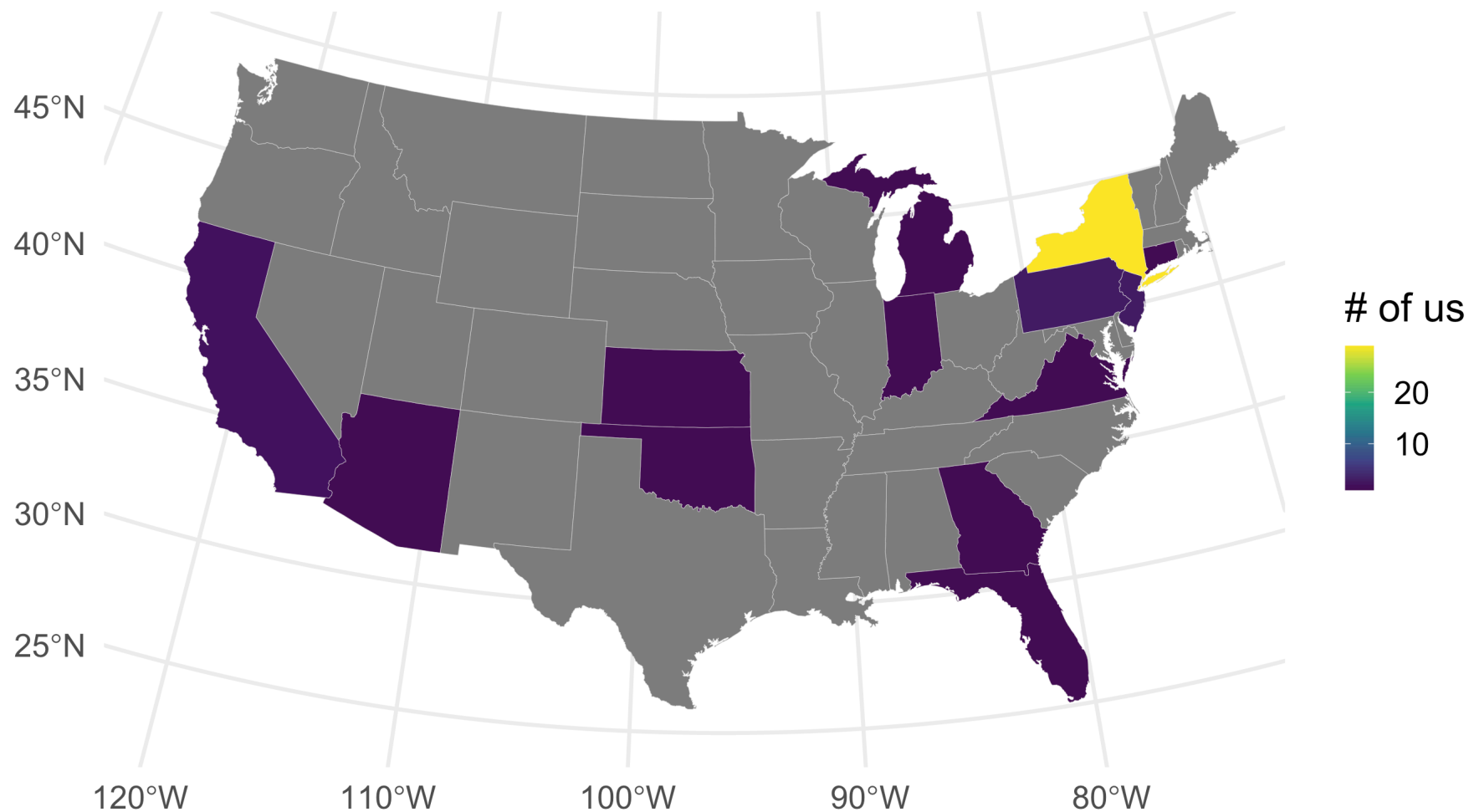
Are we all from New York?

What share of us are from the state of New York?

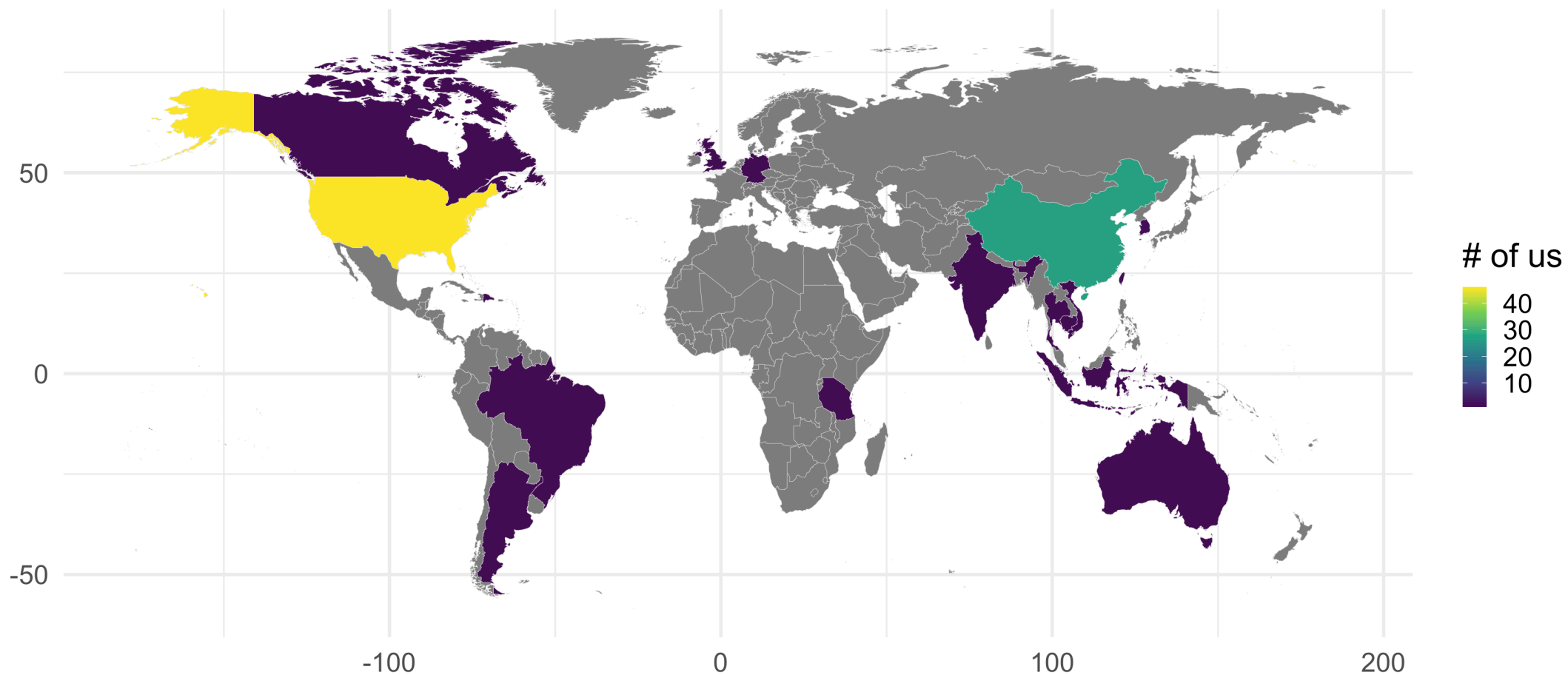
```
# step 1: do a bunch of stuff that's omitted for clarity  
# step 2: show us the answer!  
ny_share
```

```
## # A tibble: 1 × 2  
##   origin    percent  
##   <chr>      <dbl>  
## 1 new york      32
```

Are we all from New York?



Are we all from the U.S.?



Tidyverse basics

R "dialects"

Last week we *briefly* introduced several R programming concepts:

- logical expressions
- assignment
- object types
- [object names, namespace conflicts]
- [indexing]

We did this using base R

- **pro:** comes in the box, close parallels to other programming languages
- **con:** can be confusing for new programmers (esp. indexing operations)

R "dialects"

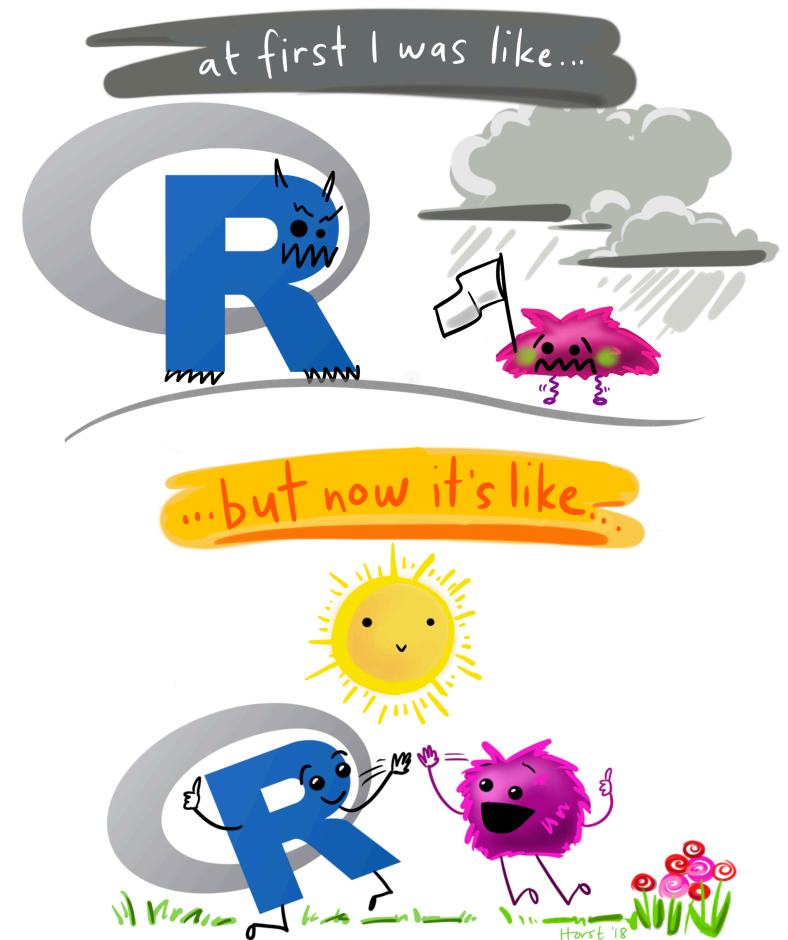
This course will primarily use the tidyverse

You can think of it as a particular "dialect" of R

Why we're using the tidyverse:

- Intuitive if you're new to programming
- Consistent philosophy and syntax
- Great documentation and community support
- For data wrangling and plotting, it's 🔥

But not representative of all R programming



Let's load the tidyverse meta-package

```
library(tidyverse)
```

```
## — Attaching core tidyverse packages — tidyverse 2.0.0 —
## ✓ dplyr      1.1.4      ✓ readr      2.1.5
## ✓ forcats    1.0.0      ✓ stringr    1.5.1
## ✓ ggplot2    3.5.2      ✓ tibble     3.3.0
## ✓ lubridate  1.9.4      ✓ tidyr      1.3.1
## ✓ purrr      1.1.0
## — Conflicts — tidyverse_conflicts() —
## ✗ dplyr::filter() masks stats::filter()
## ✗ dplyr::lag()     masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

We just loaded a bunch of packages: **ggplot2**, **dplyr**, **tidyr**, **tibble**, etc.

We also see some **namespace conflicts**

Tidyverse packages

When you install the tidyverse, you actually get a lot more packages:

```
tidyverse_packages()
```

```
## [1] "broom"          "conflicted"    "cli"           "dbplyr"
## [5] "dplyr"          "dtplyr"        "forcats"       "ggplot2"
## [9] "googledrive"    "googlesheets4" "haven"         "hms"
## [13] "httr"           "jsonlite"      "lubridate"     "magrittr"
## [17] "modelr"         "pillar"        "purrr"         "ragg"
## [21] "readr"          "readxl"        "reprex"        "rlang"
## [25] "rstudioapi"     "rvest"         "stringr"       "tibble"
## [29] "tidyr"          "xml2"          "tidyverse"
```

These other packages have to be loaded separately

Tidyverse packages

Today we'll focus on **dplyr**

But first let's talk about data frames and pipes

Review: What are objects?

There are many different *types* (or *classes*) of objects

What are some examples from last week?

Here are some objects that we'll be working with regularly:

- vectors
- matrices
- data frames
- lists
- functions

Data frames

The most important object we'll work with is the **data frame**

```
# Create a small data frame called "d"
d <- tibble(x = c("a", "b", "c"), y = 7:9)
d
```

```
## # A tibble: 3 × 2
##   x         y
##   <chr> <int>
## 1 a         7
## 2 b         8
## 3 c         9
```

Think of it as a spreadsheet

Tibbles are the tidyverse term for data frame

We'll treat "tibbles" and "data frames" as synonymous

This is essentially just a table with columns named **x** and **y**

Each row is an observation telling us the values of both **x** and **y**

Pipes

Generic programming question: Does anyone know what pipes do?

Pipes allow us to pass information through a sequence of operations

In this class, we will use R's native pipe: `|>`

Keyboard shortcut: use `Ctrl/Cmd+Shift+m` to insert the pipe (with spaces)

You might also see the `magrittr` pipe, denoted `%>%`

They are identical for simple cases (but have some differences)

Pipe usage

To see why using pipes is so powerful, consider this contrived example based on what we did this morning:

1. Wake up
2. Get out of bed
3. Get dressed
4. Drink coffee
5. Go to class

A day without pipes

You could code this through a series of assignment operations:

```
me <- wake_up(me)
me <- get_out_of_bed(me)
me <- get_dressed(me)
me <- drink(me, "coffee")
me <- go(me, "class")
```

Or by putting all these operations in a single line of code:

```
me <- go(drink(get_dressed(get_out_of_bed(wake_up(me))), "coffee"), "class")
```

Neither is ideal

Pipes are the best of both worlds

We can do everything at once, in order:

```
me |>  
  wake_up() |>  
  get_out_of_bed() |>  
  get_dressed() |>  
  drink("coffee") |>  
  go("class")
```

Emphasis is on verbs (e.g., `wake_up`, `get_out_of_bed`, etc.), not nouns (i.e., `me`)

Pipes: Best practices

Write linear code

Spread code out for readability: use one line per verb

Don't use pipes...

- when you need more than 10 (use intermediate objects instead)
- for multiple inputs or outputs
- for non-linear relationships

Reminder: `|>` and `%>%` can both be used, though they have some important differences for more advanced work. We will use `|>` in this class.

dplyr

What is dplyr?

dplyr : go wrangling



What is dplyr?

The dplyr package provides a **grammar of data manipulation**

dplyr's functions are **verbs** that correspond to things we want to **do**

dplyr functions all have these things in common:

1. their first argument is a data frame
2. their other arguments describe what you want to do
3. their output is a new data frame

1 + 3 + pipes allow us to combine simple steps to achieve complex results

dplyr has five key verbs we will use

1. `select`: select columns (i.e., variables)
2. `filter`: subset rows based on their values
3. `arrange`: reorder rows based on their values
4. `mutate`: create new columns
5. `summarize`: collapse multiple rows into a single summary value

dplyr verbs operate on different things

- Rows:
 - `filter`: subset rows based on their values
 - `arrange`: reorder rows based on their values
- Columns:
 - `select`: select columns (i.e., variables)
 - `mutate`: create new columns
- Groups of rows:
 - `summarize`: collapse multiple rows into a single summary value

Let's study these commands together using the `starwars` data frame that comes pre-packaged with dplyr

1) dplyr::select

Recall what the `starwars` data frame looks like:

```
starwars
```

```
## # A tibble: 87 × 14
##   name      height  mass hair_color skin_color eye_color birth_year sex  gender
##   <chr>      <int> <dbl> <chr>      <chr>      <chr>      <dbl> <chr> <chr>
## 1 Luke Sk...    172    77 blond      fair        blue        19    male masculi...
## 2 C-3PO        167    75 <NA>       gold        yellow      112    none masculi...
## 3 R2-D2         96    32 <NA>       white, bl... red         33    none masculi...
## 4 Darth V...   202   136 none       white        yellow      41.9  male masculi...
## 5 Leia Or...   150    49 brown      light        brown       19    fema... femin...
## 6 Owen La...   178   120 brown, gr... light        blue        52    male masculi...
## 7 Beru Wh...   165    75 brown      light        blue        47    fema... femin...
## 8 R5-D4         97    32 <NA>       white, red  red         NA     none masculi...
## 9 Biggs D...   183    84 black      light        brown       24    male masculi...
## 10 Obi-Wan...   182    77 auburn, w... fair        blue-gray   57    male masculi...
## # i 77 more rows
## # i 5 more variables: homeworld <chr>, species <chr>, films <list>,
## #   vehicles <list>, starships <list>
```

1) dplyr::select

`select` allows us to select columns / variables:

```
starwars |> select(name)
```

```
## # A tibble: 87 × 1
##   name
##   <chr>
## 1 Luke Skywalker
## 2 C-3PO
## 3 R2-D2
## 4 Darth Vader
## 5 Leia Organa
## 6 Owen Lars
## 7 Beru Whitesun Lars
## 8 R5-D4
## 9 Biggs Darklighter
## 10 Obi-Wan Kenobi
## # i 77 more rows
```

1) dplyr::select

You can use commas to select multiple columns, deselect a column using "-"

```
vitals <- starwars |>
  select(name, species, mass, height)
vitals
```

```
## # A tibble: 87 × 4
##   name                species  mass height
##   <chr>              <chr>   <dbl> <int>
## 1 Luke Skywalker     Human      77    172
## 2 C-3P0              Droid      75    167
## 3 R2-D2              Droid      32     96
## 4 Darth Vader        Human     136    202
## 5 Leia Organa         Human      49    150
## 6 Owen Lars          Human     120    178
## 7 Beru Whitesun Lars Human      75    165
## 8 R5-D4              Droid      32     97
## 9 Biggs Darklighter  Human      84    183
## 10 Obi-Wan Kenobi     Human      77    182
## # i 77 more rows
```

```
new_vitals <- vitals |>
  select(-height)
new_vitals
```

```
## # A tibble: 87 × 3
##   name                species  mass
##   <chr>              <chr>   <dbl>
## 1 Luke Skywalker     Human      77
## 2 C-3P0              Droid      75
## 3 R2-D2              Droid      32
## 4 Darth Vader        Human     136
## 5 Leia Organa         Human      49
## 6 Owen Lars          Human     120
## 7 Beru Whitesun Lars Human      75
## 8 R5-D4              Droid      32
## 9 Biggs Darklighter  Human      84
## 10 Obi-Wan Kenobi     Human      77
## # i 77 more rows
```

1) dplyr::rename

If you just want to rename columns without subsetting them, use `rename`:

```
starwars |>
  rename(alias = name, crib = homeworld)
```

```
## # A tibble: 87 × 14
##   alias      height  mass hair_color skin_color eye_color birth_year sex  gender
##   <chr>      <int> <dbl> <chr>      <chr>      <chr>      <dbl> <chr> <chr>
## 1 Luke Sk...    172    77 blond      fair        blue         19  male  mascu...
## 2 C-3PO        167    75 <NA>      gold        yellow       112  none  mascu...
## 3 R2-D2         96    32 <NA>      white, bl... red          33  none  mascu...
## 4 Darth V...   202   136 none      white       yellow       41.9  male  mascu...
## 5 Leia Or...   150    49 brown     light      brown        19  fema... femin...
## 6 Owen La...   178   120 brown, gr... light      blue         52  male  mascu...
## 7 Beru Wh...   165    75 brown     light      blue         47  fema... femin...
## 8 R5-D4         97    32 <NA>      white, red  red          NA  none  mascu...
## 9 Biggs D...   183    84 black     light      brown        24  male  mascu...
## 10 Obi-Wan...   182    77 auburn, w... fair        blue-gray    57  male  mascu...
## # i 77 more rows
## # i 5 more variables: crib <chr>, species <chr>, films <list>, vehicles <list>,
## #   starships <list>
```

Let's work through an example on Posit Cloud

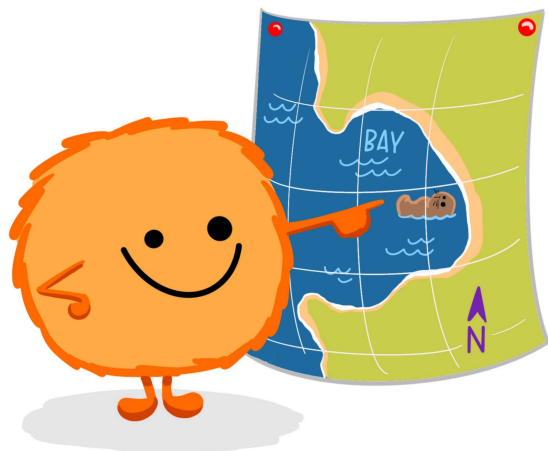
2) dplyr::filter

dplyr::filter()

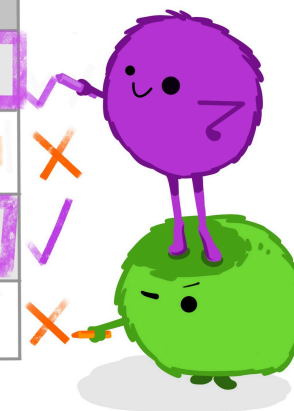
KEEP ROWS THAT
satisfy
your CONDITIONS

keep rows from... this data... ONLY IF... type is "otter" AND site is "bay"

```
filter(df, type == "otter" & site == "bay")
```



type	food	site
otter	urchin	bay
shark	seal	channel
otter	abalone	bay
otter	crab	wharf



@allison_horst

2) dplyr::filter

`filter` allows us to focus on certain rows / observations

For example, you may only be interested in droids:

```
starwars |>
  filter(species == "Droid")
```

```
## # A tibble: 6 × 14
##   name      height  mass hair_color skin_color eye_color birth_year sex  gender
##   <chr>    <int> <dbl> <chr>      <chr>      <chr>      <dbl> <chr> <chr>
## 1 C-3P0      167    75 <NA>      gold        yellow        112 none masculi...
## 2 R2-D2      96    32 <NA>      white, blue red          33 none masculi...
## 3 R5-D4      97    32 <NA>      white, red  red          NA none masculi...
## 4 IG-88     200   140 none      metal       red          15 none masculi...
## 5 R4-P17      96    NA none      silver, red red, blue     NA none feminine
## 6 BB8       NA    NA none      none        black         NA none masculi...
## # i 5 more variables: homeworld <chr>, species <chr>, films <list>,
## #   vehicles <list>, starships <list>
```

2) dplyr::filter

Or perhaps you want to subset the humans that are taller than I am:

```
starwars |>
  select(name, height, species) |>
  filter(
    species == "Human",
    height >= 194
  )
```

Can you name one? *Hint: there's only one*

```
## # A tibble: 1 × 3
##   name      height species
##   <chr>      <int> <chr>
## 1 Darth Vader    202 Human
```


2) dplyr::filter

A useful `filter` use case is identifying/removing missing data using `is.na`:

```
starwars |>
  filter(is.na(height))
```

```
## # A tibble: 6 × 14
##   name      height  mass hair_color skin_color eye_color birth_year sex  gender
##   <chr>      <int> <dbl> <chr>      <chr>      <chr>      <dbl> <chr> <chr>
## 1 Arvel Cr...    NA     NA brown      fair        brown          NA male  mascu...
## 2 Finn          NA     NA black      dark        dark          NA male  mascu...
## 3 Rey           NA     NA brown      light       hazel          NA fema... femin...
## 4 Poe Dame...    NA     NA brown      light       brown          NA male  mascu...
## 5 BB8           NA     NA none       none        black          NA none  mascu...
## 6 Captain ...    NA     NA none       none        unknown        NA fema... femin...
## # i 5 more variables: homeworld <chr>, species <chr>, films <list>,
## #   vehicles <list>, starships <list>
```

2) dplyr::filter

How could you build on this to remove rows with missing data?

```
starwars |>
  filter(!is.na(height)) # use ! for negation
```

```
## # A tibble: 81 × 14
##   name      height  mass hair_color skin_color eye_color birth_year sex  gender
##   <chr>      <int> <dbl> <chr>      <chr>      <chr>      <dbl> <chr> <chr>
## 1 Luke Sk...    172    77 blond      fair        blue         19  male  mascu...
## 2 C-3PO        167    75 <NA>      gold        yellow        112 none  mascu...
## 3 R2-D2         96    32 <NA>      white, bl... red          33  none  mascu...
## 4 Darth V...   202   136 none      white        yellow        41.9 male  mascu...
## 5 Leia Or...   150    49 brown      light        brown         19  fema... femin...
## 6 Owen La...   178   120 brown, gr... light        blue          52  male  mascu...
## 7 Beru Wh...   165    75 brown      light        blue          47  fema... femin...
## 8 R5-D4         97    32 <NA>      white, red  red          NA   none  mascu...
## 9 Biggs D...   183    84 black      light        brown         24  male  mascu...
## 10 Obi-Wan...   182    77 auburn, w... fair        blue-gray     57  male  mascu...
## # i 71 more rows
## # i 5 more variables: homeworld <chr>, species <chr>, films <list>,
## #   vehicles <list>, starships <list>
```

3) dplyr::arrange

arrange sorts the data frame based on the variable(s) you supply:

```
starwars |>
  arrange(birth_year)
```

```
## # A tibble: 87 × 14
##   name      height  mass hair_color skin_color eye_color birth_year sex  gender
##   <chr>      <int> <dbl> <chr>      <chr>      <chr>      <dbl> <chr> <chr>
## 1 Wicket ...    88   20  brown      brown      brown         8  male  mascu...
## 2 IG-88         200  140  none       metal      red          15  none  mascu...
## 3 Luke Sk...   172   77  blond      fair       blue         19  male  mascu...
## 4 Leia Or...   150   49  brown      light      brown         19  fema... femin...
## 5 Wedge A...   170   77  brown      fair       hazel         21  male  mascu...
## 6 Plo Koon    188   80  none       orange     black         22  male  mascu...
## 7 Biggs D...   183   84  black      light      brown         24  male  mascu...
## 8 Han Solo    180   80  brown      fair       brown         29  male  mascu...
## 9 Lando C...   177   79  black      dark       brown         31  male  mascu...
## 10 Boba Fe...   183  78.2  black      fair       brown        31.5  male  mascu...
## # i 77 more rows
## # i 5 more variables: homeworld <chr>, species <chr>, films <list>,
## #   vehicles <list>, starships <list>
```

3) dplyr::arrange

We can also arrange items in descending order using `arrange(desc())`:

```
starwars |>
  arrange(desc(birth_year))
```

```
## # A tibble: 87 × 14
##   name      height  mass hair_color skin_color eye_color birth_year sex  gender
##   <chr>      <int> <dbl> <chr>      <chr>      <chr>      <dbl> <chr> <chr>
## 1 Yoda         66    17 white      green      brown         896 male  mascu...
## 2 Jabba D...   175  1358 <NA>      green-tan... orange         600 herm... mascu...
## 3 Chewbac...   228   112 brown     unknown    blue          200 male  mascu...
## 4 C-3PO       167    75 <NA>      gold       yellow         112 none  mascu...
## 5 Dooku       193    80 white     fair       brown          102 male  mascu...
## 6 Qui-Gon...   193    89 brown     fair       blue           92 male  mascu...
## 7 Ki-Adi-...   198    82 white     pale       yellow          92 male  mascu...
## 8 Finis V...   170    NA blond     fair       blue           91 male  mascu...
## 9 Palpati...   170    75 grey      pale       yellow          82 male  mascu...
## 10 Cliegg ...   183    NA brown     fair       blue           82 male  mascu...
## # i 77 more rows
## # i 5 more variables: homeworld <chr>, species <chr>, films <list>,
## #   vehicles <list>, starships <list>
```

4) dplyr::mutate



4) dplyr::mutate

You can create new columns from scratch or by transforming existing columns:

```
starwars |>
  select(name, height) |>
  mutate(height_in_inches = height / 2.54)
```

```
## # A tibble: 87 × 3
##   name                height height_in_inches
##   <chr>                <int>         <dbl>
## 1 Luke Skywalker      172          67.7
## 2 C-3PO               167          65.7
## 3 R2-D2               96          37.8
## 4 Darth Vader        202          79.5
## 5 Leia Organa        150          59.1
## 6 Owen Lars          178          70.1
## 7 Beru Whitesun Lars 165          65.0
## 8 R5-D4              97          38.2
## 9 Biggs Darklighter  183          72.0
## 10 Obi-Wan Kenobi     182          71.7
## # i 77 more rows
```

4) dplyr::mutate

You can create new columns from scratch or by transforming existing columns:

```
starwars |>
  select(name, height) |>
  mutate(height_in_inches = height / 2.54) |>
  mutate(height_in_feet = height_in_inches / 12)
```

```
## # A tibble: 87 × 4
```

	name	height	height_in_inches	height_in_feet
	<chr>	<int>	<dbl>	<dbl>
## 1	Luke Skywalker	172	67.7	5.64
## 2	C-3PO	167	65.7	5.48
## 3	R2-D2	96	37.8	3.15
## 4	Darth Vader	202	79.5	6.63
## 5	Leia Organa	150	59.1	4.92
## 6	Owen Lars	178	70.1	5.84
## 7	Beru Whitesun Lars	165	65.0	5.41
## 8	R5-D4	97	38.2	3.18
## 9	Biggs Darklighter	183	72.0	6.00
## 10	Obi-Wan Kenobi	182	71.7	5.97

```
## # i 77 more rows
```

4) dplyr::mutate

mutate creates variables in order, so we can chain them together in a single call:

```
starwars |>
  select(name, height) |>
  mutate(height_in_inches = height / 2.54, # separate with a comma
         height_in_feet = height_in_inches / 12)
```

```
## # A tibble: 87 × 4
```

	name	height	height_in_inches	height_in_feet
	<chr>	<int>	<dbl>	<dbl>
## 1	Luke Skywalker	172	67.7	5.64
## 2	C-3PO	167	65.7	5.48
## 3	R2-D2	96	37.8	3.15
## 4	Darth Vader	202	79.5	6.63
## 5	Leia Organa	150	59.1	4.92
## 6	Owen Lars	178	70.1	5.84
## 7	Beru Whitesun Lars	165	65.0	5.41
## 8	R5-D4	97	38.2	3.18
## 9	Biggs Darklighter	183	72.0	6.00
## 10	Obi-Wan Kenobi	182	71.7	5.97

```
## # i 77 more rows
```


4) dplyr::mutate

Boolean, logical, and conditional operators all work well with `mutate`

For example:

```
starwars |>
  filter(name=="Luke Skywalker" | name=="Anakin Skywalker") |>
  select(name, height) |>
  mutate(tall = height > 180)
```

How many rows do you think this will return? How many columns?

```
## # A tibble: 2 × 3
##   name          height tall
##   <chr>         <int> <lgl>
## 1 Luke Skywalker    172 FALSE
## 2 Anakin Skywalker   188  TRUE
```

5) dplyr::summarize

Often we want to get summary statistics or *collapse* our data

`summarize` provides an easy way to do this

Example: "What are the minimum, maximum, and average prices of `diamonds`?"

```
diamonds |>
  summarize(
    min = min(price),      # calculate the min price
    max = max(price),      # calculate the max price
    average = mean(price) # calculate the mean price
  )
```

```
## # A tibble: 1 × 3
##   min    max average
##   <int> <int>   <dbl>
## 1   326 18823   3933.
```

5) dplyr::summarize

Including `na.rm = TRUE` keeps NAs from propagating to the end result:

```
# probably not what we want
starwars |>
  summarize(mh = mean(height))
```

```
## # A tibble: 1 × 1
##       mh
##   <dbl>
## 1    NA
```

```
# better
starwars |>
  summarize(mh = mean(height, na.rm = TRUE))
```

```
## # A tibble: 1 × 1
##       mh
##   <dbl>
## 1  175.
```

Is this a feature or a bug?

It depends on the context!

Bonus: dplyr::group_by

`summarize` is particularly useful in combination with `group_by`:

```
starwars |>
  group_by(species) |> # for each species
  summarize(mean_height = mean(height, na.rm = TRUE)) # calculate the mean height
```

```
## # A tibble: 38 × 2
##   species    mean_height
##   <chr>         <dbl>
## 1 Aleena          79
## 2 Besalisk       198
## 3 Cerean         198
## 4 Chagrian       196
## 5 Clawdite       168
## 6 Droid          131.
## 7 Dug            112
## 8 Ewok            88
## 9 Geonosian      183
## 10 Gungan        209.
## # i 28 more rows
```

Bonus: `dplyr::ungroup`



Groups are persistent (sort of), `ungroup` removes them

Let's work through an example on Posit Cloud

(the remaining slides contain other dplyr goodies for your reference)

Other dplyr goodies: distinct

`distinct` to isolate unique observations:

```
starwars |> distinct(species)
```

```
## # A tibble: 38 × 1
##   species
##   <chr>
## 1 Human
## 2 Droid
## 3 Wookiee
## 4 Rodian
## 5 Hutt
## 6 <NA>
## 7 Yoda's species
## 8 Trandoshan
## 9 Mon Calamari
## 10 Ewok
## # i 28 more rows
```

Other dplyr goodies: count

count to get the number of observations:

```
starwars |>  
  count(species)
```

```
## # A tibble: 38 × 2  
##   species      n  
##   <chr>    <int>  
## 1 Aleena      1  
## 2 Besalisk    1  
## 3 Cerean      1  
## 4 Chagrian    1  
## 5 Clawdite    1  
## 6 Droid       6  
## 7 Dug         1  
## 8 Ewok        1  
## 9 Geonosian   1  
## 10 Gungan     3  
## # i 28 more rows
```

```
starwars |>  
  group_by(species) |> summarize(num = n())
```

```
## # A tibble: 38 × 2  
##   species      num  
##   <chr>    <int>  
## 1 Aleena      1  
## 2 Besalisk    1  
## 3 Cerean      1  
## 4 Chagrian    1  
## 5 Clawdite    1  
## 6 Droid       6  
## 7 Dug         1  
## 8 Ewok        1  
## 9 Geonosian   1  
## 10 Gungan     3  
## # i 28 more rows
```


Other dplyr goodies: window functions

Window functions make it easy to get leads and lags, percentiles, cumulative sums, etc.

- These are often used in conjunction with grouped mutates
- See `vignette("window-functions")`

The final set of dplyr "goodies" worth mentioning are the family of join operations, but we'll come back to those later