Data Cleaning and Manipulation

In The Tidyverse

Josh Allen

Department of Political Science

Georgia State University

2022-08-16

Packages You Will Need

```
install.packages("pacman")

pacman::p_load("palmerpenguins", "knitr", "kableExtra", "tidyverse", install = TRUE)

data("starwars")

penguins = penguins
```

Cleaning Data in the Tidyverse

> Like families, tidy datasets are all alike but every messy dataset is messy in its own way.
>
> Hadley Wickham

What is tidy data?

- Every variable is a column
- Every row is an observation
- Each type of observational unit forms a table.
- Most of the time this looks like what is known as "long data"

Pros of the Tidyverse

One of the most furious debates in R is base R versus tidyverse. I will not go into great detail about them because many people who are better at R than me have

- Similar syntax and philosophy which makes it more intuitive to learn
- Extremely nice community
- The core syntax of `dplyr` provides a nice front end for many other big data tools i.e. SQL
- Generally pretty intuitive and easy to figure out what past you has done.

Pros of the Tidyverse

One of the most furious debates in R is base R versus tidyverse. I will not go into great detail about them because many people who are better at R than me have

- Similar syntax and philosophy which makes it more intuitive to learn
- Extremely nice community
- The core syntax of `dplyr` provides a nice front end for many other big data tools i.e. SQL
- Generally pretty intuitive and easy to figure out what past you has done.However,

do not avoid some base R solutions

Base R is more stable so it limits the fragility of your code

A mix of base R and the tidyverse never hurt nobody

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

What is loaded

```
library(tidyverse)
```

```
## ── Attaching packages ─────────────────────────────────────── tidyverse 1.3.2 ──
## ✔ ggplot2 3.3.6     ✔ purrr   0.3.4
## ✔ tibble  3.1.8     ✔ dplyr   1.0.9
## ✔ tidyr   1.2.0     ✔ stringr 1.4.0
## ✔ readr   2.1.2     ✔ forcats 0.5.1
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
## ✖ dplyr::filter()     masks stats::filter()
## ✖ dplyr::group_rows() masks kableExtra::group_rows()
## ✖ dplyr::lag()        masks stats::lag()
```

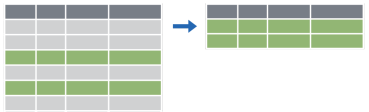
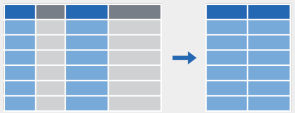
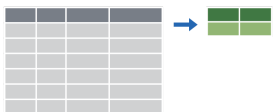
- However you can load in specific packages. There is something called the tinyverse that loads in `ggplot2` and `data.table`

Namespace Conflicts

- Since R is open source you can name your functions just about anything
    - this results in lots of packages having similarly named functions
- `Dplyr` is just warning us that if we use `filter` or `lag` it will use `dplyr`'s version of the function
- Whenever `R` runs into a namespace conflict it will default to the last package that was loaded
- It is generally best practice to load the `tidyverse` last
- You can also use `packagename::function you want to use` to get around it.
    - This is called a namespace call
    - explicitly tells R which function you are using


dplyr
dplyr.tidyverse.org

The Big 5 Dplyr Verbs

Extract rows with `filter()`	
Extract columns with `select()`	
Arrange/sort rows with `arrange()`	
Make new columns with `mutate()`	
Make group summaries with `group_by() %>% summarize()`	

A Common Syntax

```
VERB(DATA, ...)
```

- `VERB` = dplyr function/verb
- `DATA` = Data frame to transform
- `...` = Stuff the verb does

Piping

- The pipe is wildly convenient and makes your code much easier to debug
- It is a lot more intuitive than huge nested calls not only to read but to write
- It is how most people do things in the tidyverse

Magrittr Piping

The tidyverse has its own pipe %>% and used to be the only game in town.

- %>% is just % followed by > followed by %>%
- A pipe takes whats on the left hand side of the pipe and evaluates it as the *first* argument on the right hand side

```
# These do the same thing
summarise(group_by(filter(penguins,species == "Adelie"), island), mean(body_mass_g, na.rm = TRUE))

penguins %>%  filter(species == "Adelie") %>% group_by(island) %>% summarise(mean(body_mass_g, na.rm = TRUE))
```

Example provided by Grant McDermot

Magrittr Piping(cont)

- When you pipe stuff it is easier to think of the pipe as saying and then
- If we use an example from your morning routine we can start to figure it our

```
me %>%
wake_up(time = "8.00am") %>%
get_out_of_bed(side = "correct") %>%
get_dressed(pants = "TRUE", shirt = "TRUE") %>%
leave_house(car = TRUE, bike = FALSE, MARTA = FALSE) %>%
am_late(traffic = TRUE)
```

Example derived from Andrew Heiss

Base R Pipe

- The pipe caught on and the team behind `R` added a native pipe |>
    - this is just `|` followed by >
- If you are have a version of R that is older than `4.2.0` it should come with the native pipe
- The base versus magrittr pipe differ slightly and it is worth knowing some of the differences
- The base R pipe is pretty flexible and supports some cool computer sciency stuff for more check out this page
    - I tend to use the base R pipe

filter()

filter()

filter extracts a set of rows that meet a test that you give it

```
filter(.data = DATA, ...)
```

- `DATA` = Data frame to transform
- `...` = One or more tests
  `filter()` returns each row for which the test is TRUE

What Kind of Tests Can I Do?

Test	Meaning	Test	Meaning
`x < y`	Less than	`x %in% y`	In (group membership)
`x > y`	Greater than	`is.na(x)`	Is missing
`==`	Equal to	`!is.na(x)`	Is not missing
`x <= y`	Less than or equal to		
`x >= y`	Greater than or equal to		
`x != y`	Not equal to		

filter()

```
penguins |>
  filter(species == "Adelie",
         body_mass_g < 6200,
         bill_length_mm < 59.6)
```

species	body_mass_g	bill_length_mm
Adelie	3750	39.1
Adelie	3800	39.5
Adelie	3250	40.3
Adelie	3450	36.7
Adelie	3650	39.3
Adelie	3625	38.9
…	…	…

filter()

regular expressions also work and can be pretty handy.

```
starwars |>
  filter(grepl("Skywalker", name)|
           grepl("Palp", name) |
           grepl("Obi", name))
```

name	height	mass	homeworld
Luke Skywalker	172	77	Tatooine
Obi-Wan Kenobi	182	77	Stewjon
Anakin Skywalker	188	84	Tatooine
Palpatine	170	75	Naboo
Shmi Skywalker	163	NA	Tatooine
...	...	...	...

filter()

You can also supply a list to filter to subset your data

```
starwars |>
  filter(homeworld %in% c("Naboo", "Tatooine")
         & !grepl(c("ars"),name))
```

name	homeworld
Luke Skywalker	Tatooine
C-3PO	Tatooine
R2-D2	Naboo
Darth Vader	Tatooine
R5-D4	Tatooine
Biggs Darklighter	Tatooine
...	...

filter()

One of the most common uses is to find and remove missing values or not in a list.

```
penguins |>
  filter(is.na(sex))

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

Your Turn

- Using the sample code from the last slide filter out the NA values. Hint use !
- Using either the star wars or penguins data use `%in%` to get a set of home worlds or islands
- subset the penguin data where body mass is less than `4202` and not on the Dream Island.

Select()

Select()

For various reasons we may want to just keep a few columns.

- For this task we use our friend select
- You can feed it one column or a range of columns with :
- You can also leave columns out using -

```
penguins |>
select(species,
     island:bill_depth_mm,
     -bill_length_mm)
```

species	island	bill_depth_mm
Adelie	Torgersen	18.7
Adelie	Torgersen	17.4
Adelie	Torgersen	18
Adelie	Torgersen	NA
Adelie	Torgersen	19.3
Adelie	Torgersen	20.6
...	...	...

Helpful Helpers

- Dplyr comes really useful functions that help you when there are common patterns in your variable names
- the syntax usually goes `select(contains("pattern"))`, `select(starts_with("pattern"))`, or `select(ends_with("pattern"))`

```
starwars |>
  select(name, ends_with("color"), homeworld)
```

name	hair_color	skin_color	eye_color	homeworld
Luke Skywalker	blond	fair	blue	Tatooine
C-3PO	NA	gold	yellow	Tatooine
R2-D2	NA	white, blue	red	Naboo
Darth Vader	none	white	yellow	Tatooine
Leia Organa	brown	light	brown	Alderaan
Owen Lars	brown, grey	light	blue	Tatooine
...	...	...	...	...

Mutate

- Mutate creates new columns in your dataset
- You can use logical and boolean operators as well as various mathematical transformations

```
penguins |>
  filter(!is.na(sex)) |>
  select(species, sex) |>
  mutate(female = ifelse(sex == "female", TRUE, FALSE
```

species	sex	female
Adelie	male	FALSE
Adelie	female	TRUE
Adelie	female	TRUE
Adelie	female	TRUE
Adelie	male	FALSE
Adelie	female	TRUE
...	...	...

Mutate()

- Mutate is order aware so you don't have to use a new `mutate` for each new variable you want to create

```
penguins |>
 filter(!is.na(bill_length_mm) & species == "Gentoo"
 mutate(long_bill = bill_length_mm * 2,
        long_bill_logical =
          ifelse(long_bill >= 100
                 & long_bill <= 119.20,
                 "Long",
                 "Short")) |>
 select(species, bill_length_mm, long_bill, long_bil
```

species	bill_length_mm	long_bill	long_bill_logical
Gentoo	46.1	92.2	Short
Gentoo	50	100	Long
Gentoo	48.7	97.4	Short
Gentoo	50	100	Long
Gentoo	47.6	95.2	Short
Gentoo	46.5	93	Short
...	...	...	...

Group_by() and Summarize()

group_by()

- `group_by()` simply puts rows into groups based on values of a column
- Not necessarily the most useful function because nothing really happens when called by itself

```
penguins |>
  group_by(species)
```

group_by()

- `group_by()` simply puts rows into groups based on values of a column
- Not necessarily the most useful function because nothing really happens when called by itself

```
penguins |>
  group_by(species)
```

- Unless you combine it with `summarize()`

You may see me use `summarise` this is the same thing as `summarize`.

```
penguins |>
  group_by(species) |>
  summarize(total_penguins = n())
```

species	total_penguins
Adelie	152
Chinstrap	68
Gentoo	124

```
penguins |>
  group_by(species) |>
  summarise(mean_bill_length = mean(bill_length_mm, r
```

species	mean_bill_length
Adelie	38.7913907284768
Chinstrap	48.8338235294118
Gentoo	47.5048780487805

Your Turn

- Calculate the minimum, maximum, and median for each species of penguin
- what happens if you remove `group_by()`?
- Do the same for the star wars data

`03:00`

Other useful dplyr stuff

Joins

- Often times we need to get data from another dataset
- In `Dplyr` we use join operations
    - What each of the below joins are doing are elaborated more in R for Data Science
- `inner_join(df1, df2)`
- `left_join(df1, df2)`
- `right_join(df1, df2)`
- `full_join(df1, df2)`
- `semi_join(df1, df2)`
- `anti_join(df1, df2)`

Joins

- The basic syntax for each join is the same `_join(df1, df2, by = "var I want to join on)`
- The by argument can take a list of variables or you can just let dplyr guess(bad idea)
- Each join does something different and some are more cautious than others
- I tend to use `left_join` the most and is handy when you are trying to fill in gaps in panel data

```
data1 = data.frame(ID = 1:2,
            X1 = c(&quot;a1&quot;, &quot;a2&quot;),
            stringsAsFactors = FALSE)
```

ID	X1
1	a1
2	a2

```
data2 = data.frame(ID = 2:3,
                   X2 = c("b1", "b2"),
                   stringsAsFactors = FALSE)
```

ID	X2
2	b1
3	b2

left_join()

```
left_join(data1, data2, by = "ID")
```

ID	X1	X2
1	a1	NA
2	a2	b1

Using "Real" Data

```
national_data
```

state	year	unemployment	inflation	population
GA	2018	5.0	2.0	100
GA	2019	5.3	1.8	200
GA	2020	5.2	2.5	300
NC	2018	6.1	1.8	350
NC	2019	5.9	1.6	375
NC	2020	5.3	1.8	400
CO	2018	4.7	2.7	200
CO	2019	4.4	2.6	300
CO	2020	5.1	2.5	400

```
national_libraries
```

state	year	libraries	schools
CO	2018	230	470
CO	2019	240	440
CO	2020	270	510
NC	2018	200	610
NC	2019	210	590
NC	2020	220	530

Joins(cont)

```
national_combined = left_join(national_data, national_libraries,
                              by = c("state", "year"))

national_combined
```

state	year	unemployment	inflation	population	libraries	schools
GA	2018	5.0	2.0	100	NA	NA
GA	2019	5.3	1.8	200	NA	NA
GA	2020	5.2	2.5	300	NA	NA
NC	2018	6.1	1.8	350	200	610
NC	2019	5.9	1.6	375	210	590
NC	2020	5.3	1.8	400	220	530
CO	2018	4.7	2.7	200	230	470
CO	2019	4.4	2.6	300	240	440
CO	2020	5.1	2.5	400	270	510

```
national_combined = national_data |>
  left_join(national_libraries, by = c("state", "year"))


national_combined
```

state	year	unemployment	inflation	population	libraries	schools
GA	2018	5.0	2.0	100	NA	NA
GA	2019	5.3	1.8	200	NA	NA
GA	2020	5.2	2.5	300	NA	NA
NC	2018	6.1	1.8	350	200	610
NC	2019	5.9	1.6	375	210	590
NC	2020	5.3	1.8	400	220	530
CO	2018	4.7	2.7	200	230	470
CO	2019	4.4	2.6	300	240	440
CO	2020	5.1	2.5	400	270	510

Joins continued

state	year	unemployment	inflation	population
GA	2018	5.0	2.0	100
GA	2019	5.3	1.8	200
GA	2020	5.2	2.5	300
NC	2018	6.1	1.8	350
NC	2019	5.9	1.6	375
NC	2020	5.3	1.8	400
CO	2018	4.7	2.7	200
CO	2019	4.4	2.6	300
CO	2020	5.1	2.5	400

statename	year	libraries	schools
CO	2018	230	470
CO	2019	240	440
CO	2020	270	510
NC	2018	200	610
NC	2019	210	590
NC	2020	220	530

Joins(cont)

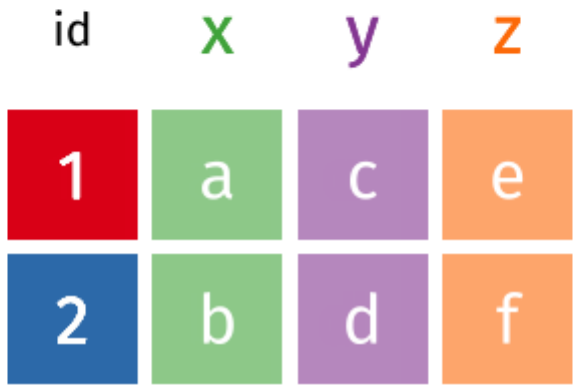
```
national_data |>
  left_join(national_libraries, by = c("state" = "statename", "year"))
```

state	year	unemployment	inflation	population	libraries	schools
GA	2018	5.0	2.0	100	NA	NA
GA	2019	5.3	1.8	200	NA	NA
GA	2020	5.2	2.5	300	NA	NA
NC	2018	6.1	1.8	350	200	610
NC	2019	5.9	1.6	375	210	590
NC	2020	5.3	1.8	400	220	530
CO	2018	4.7	2.7	200	230	470
CO	2019	4.4	2.6	300	240	440
CO	2020	5.1	2.5	400	270	510


tidyr
tidyr.tidyverse.org

Reshaphing Data

wide



What does this look like in practice?

religion	<$10k	$10-20k	$20-30k	$30-40k	$40-50k	$50-75k	$75-100k	$100-150k	>150k	Don't know/refused
Agnostic	27	34	60	81	76	137	122	109	84	96
Atheist	12	27	37	52	35	70	73	59	74	76
Buddhist	27	21	30	34	33	58	62	39	53	54
Catholic	418	617	732	670	638	1116	949	792	633	1489
Don’t know/refused	15	14	15	11	10	35	21	17	18	116
Evangelical Prot	575	869	1064	982	881	1486	949	723	414	1529
Hindu	1	9	7	9	11	34	47	48	54	37
Historically Black Prot	228	244	236	238	197	223	131	81	78	339
Jehovah's Witness	20	27	24	24	21	30	15	11	6	37
Jewish	19	19	25	25	30	95	69	87	151	162
Mainline Prot	289	495	619	655	651	1107	939	753	634	1328

Making Data Longer

```
relig_income |>
  pivot_longer(!religion, names_to = "income", values_to = "count" )
```

religion	income	count
Agnostic	<$10k	27
Agnostic	$10-20k	34
Agnostic	$20-30k	60
Agnostic	$30-40k	81
Agnostic	$40-50k	76
Agnostic	$50-75k	137
Agnostic	$75-100k	122
Agnostic	$100-150k	109
Agnostic	>150k	84
Agnostic	Don't know/refused	96

More Complex Example

```
data("billboard")
```

artist	track	date.entered	wk1	wk2	wk3	wk4	wk5
2 Pac	Baby Don't Cry (Keep...	2000-02-26	87	82	72	77	87
2Ge+her	The Hardest Part Of ...	2000-09-02	91	87	92	NA	NA
3 Doors Down	Kryptonite	2000-04-08	81	70	68	67	66
3 Doors Down	Loser	2000-10-21	76	76	72	69	67
504 Boyz	Wobble Wobble	2000-04-15	57	34	25	17	17
98^0	Give Me Just One Nig...	2000-08-19	51	39	34	26	26
A*Teens	Dancing Queen	2000-07-08	97	97	96	95	100
Aaliyah	I Don't Wanna	2000-01-29	84	62	51	41	38
Aaliyah	Try Again	2000-03-18	59	53	38	28	21
Adams, Yolanda	Open My Heart	2000-08-26	76	76	74	69	68
Adkins, Trace	More	2000-04-29	84	84	75	73	73

Using Your Helpers

```
billboard |>
  pivot_longer(
    cols = starts_with("wk"),
    names_to = "week",
    names_prefix = "wk",
    values_to = "rank",
    values_drop_na = TRUE
  )
```

artist	track	date.entered	week	rank
2 Pac	Baby Don't Cry (Keep...	2000-02-26	1	87
2 Pac	Baby Don't Cry (Keep...	2000-02-26	2	82
2 Pac	Baby Don't Cry (Keep...	2000-02-26	3	72
2 Pac	Baby Don't Cry (Keep...	2000-02-26	4	77
2 Pac	Baby Don't Cry (Keep...	2000-02-26	5	87
2 Pac	Baby Don't Cry (Keep...	2000-02-26	6	94
2 Pac	Baby Don't Cry (Keep...	2000-02-26	7	99
2Ge+her	The Hardest Part Of ...	2000-09-02	1	91