

Data wrangling

Tame your data

Daniela Palleschi

2023-04-13

Table of contents

'wrangle' defined	3
Wrangler	3
Data Wrangling	3
Why tidy data?	4
What does tidy data look like?	4
the tidyverse	5
package versions	5
the magitrr pipe %>%	6
load our data	6
save dataframe as object <-	7
A note on annotation	7
Tidyverse verbs	8
rename()	8
Exercise	8
relocate	9
mutate()	9
if_else()	10
case_when()	10
Exercise	11
Extra exercise	12
group_by() and ungroup()	12
select()	13
select()	14
Exercise	15

filter()	15
filter()	16
.	16
Exercise	16
distinct()	17
arrange()	19
separate()	20
pivot_wider()	20
pivote_longer()	20
Save your tidy data	20
Summary	21
Important terms	21
Important functions	21
Session Info	22

```
knitr::opts_chunk$set(eval = T, # evaluate chunks
                        echo = T, # 'print code chunk?'
                        message = F, # 'print messages (e.g., warnings)?'
                        error = F, # stop when error encountered
                        warning = F) # don't print warnings
```

```
# Create references.json file based on the citations in this script
# make sure you have 'bibliography: references.json' in the YAML
rbbt::bbt_update_bib("_wrangling.qmd")
```

```
## play sound if error encountered
### from: https://sejohnston.com/2015/02/24/make-r-beep-when-r-markdown-finishes-or-when-i
options(error = function(){ # Beep on error
  beepr::beep(sound = "wilhelm")
  Sys.sleep(2) #
})
## and when knitting is complete
.Last <- function() { # Beep on exiting session
  beepr::beep(sound = "ping")
  Sys.sleep(6) # allow to play for 6 seconds
}
```

‘wrangle’ defined

/ ˈræŋ ɪ l/

noun

a dispute or argument, typically one that is long and complicated. “an insurance wrangle is holding up compensation payments”

verb

1. have a long, complicated dispute or argument. “the bureaucrats continue wrangling over the fine print”
2. NORTH AMERICAN round up, herd, or take charge of (livestock). “the horses were wrangled early”

Wrangler



Jeep Wrangler



Wrangler Jeans



Cowboys

Data Wrangling

- data wrangling = tidying + transforming
- an often long, arduous stage of analysis

Tidy

- re-shaping
 - e.g., from wide to long data
- outcome:
 - each column = a variable
 - each row = an observation

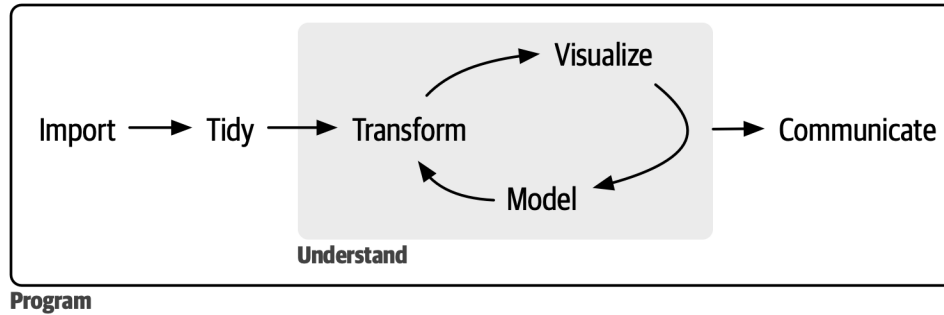


Figure 1: [Image source](#): Wickham et al. (n.d.) (all rights reserved)

Transform

- filtering
- creating new variables based on observations (e.g., reaction times)
- computing summary statistics (e.g., means)

Why tidy data?

- helps future you
 - and collaborators
- facilitates sharing your data *and* code (Laurinavichyute et al., 2022)
- in short: facilitates reproducibility!

What does tidy data look like?

Three rules (Wickham et al., n.d.):

1. Each variable is a column, each column is a variable
 2. Each observation is a row, each row is an observation
 3. Each value is a cell, each cell is a single value
- N.B., how you define a *variable* or *observation* is relative to what you want to do
 - for now, let's consider a single trial per participant as an observation

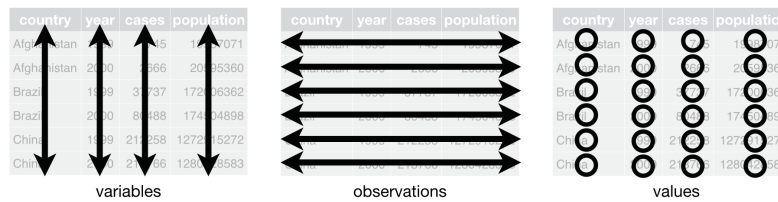


Figure 2: [Image source](#): Wickham et al. (n.d.) (all rights reserved)

the tidyverse

- a collection of R packages for tidy data
- you need to load a package at the beginning of every session
 - today we will mostly use functions from the `dplyr` package
 - * if you load the `tidyverse` you don't need to also load `dplyr`

```
# load tidyverse
library(tidyverse)
```

package versions

- you can check the package version with:

```
packageVersion("tidyverse")
```

```
[1] '2.0.0'
```

- need to update?

```
# update a single package
install.packages("tidyverse")
```

- what about your other packages?

```
# which packages need updating?
old.packages()
# update all old packages
update.packages()
```

the magitrr pipe %>%

- takes the object before it and feeds it into the next command
 - the pipe could be read as “and then”
 - N.B., there’s a new pipe in town! The R native |> (Ctrl/Cmd+Shift+M)

```
1 # take data frame and then...
2 iris %>%
3   # print the head
4   head()
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa



Figure 3: Image source: [magitrr documentation](#) (all rights reserved)

load our data

```
# load lifetime data
readr::read_csv(here::here("data/data_lifetime_pilot.csv"))
```

```
# A tibble: 4,431 x 28
  RECORDING_SESSION_LABEL TRIAL_INDEX EYE_USED IA_DWELL_TIME
  <chr>                  <dbl> <chr>          <dbl>
1 px3                    1 RIGHT            0
2 px3                    2 RIGHT            0
3 px3                    3 RIGHT            0
4 px3                    3 RIGHT            0
5 px3                    3 RIGHT            0
6 px3                    3 RIGHT            0
7 px3                    3 RIGHT            0
8 px3                    3 RIGHT            0
9 px3                    4 RIGHT            0
10 px3                   5 RIGHT            0
# i 4,421 more rows
# i 24 more variables: IA_FIRST_FIXATION_DURATION <dbl>,
#   IA_FIRST_RUN_DWELL_TIME <dbl>, IA_FIXATION_COUNT <dbl>, IA_ID <dbl>,
#   IA_LABEL <chr>, IA_REGRESSION_IN <dbl>, IA_REGRESSION_IN_COUNT <dbl>,
#   IA_REGRESSION_OUT <dbl>, IA_REGRESSION_OUT_COUNT <dbl>,
#   IA_REGRESSION_PATH_DURATION <dbl>, KeyPress <dbl>, rt <dbl>, bio <chr>,
#   critical <chr>, gender <chr>, item_id <dbl>, list <dbl>, match <chr>, ...
```

- was anything added to the Environment pane (top right box in RStudio)?

save dataframe as object <-

- `object_name <- code_output_to_be_saved_as_object_name`

```
1 # load lifetime data and store it under df_lifetime
2 df_lifetime <- readr::read_csv(here::here("data/data_lifetime_pilot.csv"),
3                               # for special characters
4                               locale = readr::locale(encoding = "latin1")
5                               )
```

- you should now see the object `df_lifetime` in the Environment pane

A note on annotation

- annotation as you go: provide useful comments to describe your code
- you always have at least one collaborator: future you!
 - try to write useful comments to help future you/collaborators follow

```
1 library(tidyverse) # for tidy data wrangling
```

Tidyverse verbs

- verbs are functions from the `tidyverse` package
- for data tidying and transforming we'll mostly use verbs from the `dplyr` package, which is part of the `tidyverse`
- check out [RLadies Freiburg](#) to see a [YouTube video](#) that covers most of these verbs

`rename()`

- one of the first things you'll often want to do is rename some variables
- let's start by re-naming some of our variables
 - e.g., `RECORDING_SESSION_LABEL` is a long way of saying 'participant'

```
1 # rename variables
2 df_lifetime <- df_lifetime %>% # make df_lifetime from df_lifetime BUT THEN
3   rename("px" = RECORDING_SESSION_LABEL, # rename a variable and (comma = 'and')
4         "trial" = TRIAL_INDEX) # another variable
```

Exercise

Change the following names:

- `EYE_USED` to `eye`
- `IA_DWELL_TIME` to `tt`
- `IA_FIRST_FIXATION_DURATION` to `ff`
- `IA_FIXATION_COUNT` to `fix_count`
- `IA_FIRST_RUN_DWELL_TIME` to `fp`
- `IA_ID` to `region_n`
- `IA_LABEL` to `region_text`

- `IA_REGRESSION_IN` to `reg_in`
- `IA_REGRESSION_IN_COUNT` to `reg_in_count`
- `IA_REGRESSION_OUT` to `reg_out`
- `IA_REGRESSION_OUT_COUNT` to `reg_out_count`
- `IA_REGRESSION_PATH_DURATION` to `rpd`
- `name_vital_status` to `lifetime`


```
# the names should then look like this:
names(df_lifetime)
```

```
[1] "px"          "trial"       "eye"         "tt"
[5] "ff"          "fp"          "fix_count"   "region_n"
[9] "region_text" "reg_in"      "reg_in_count" "reg_out"
[13] "reg_out_count" "rpd"        "KeyPress"    "rt"
[17] "bio"         "critical"    "gender"      "item_id"
[21] "list"        "match"      "condition"   "name"
[25] "lifetime"    "tense"      "type"        "yes_press"
```

relocate

- the second step thing you might want to do is reorder your variables so the most important/relevant are near the beginning and ordered logically
 - let's order our continuous reading time variables from 'earliest' to 'latest' measure

```
df_lifetime <- df_lifetime %>%
  relocate(ff,fp,rpd,tt, .after="eye") %>%
  relocate(region_n, region_text, .after="trial")
```

```
names(df_lifetime[1:10])
```

```
[1] "px"          "trial"       "region_n"    "region_text" "eye"
[6] "ff"          "fp"          "rpd"         "tt"          "fix_count"
```

mutate()

Make some change

- new columns

```
1 df_lifetime <- df_lifetime %>%
2   mutate(new_column = "new")
```

- change existing column

```

1 df_lifetime <- df_lifetime %>%
2   mutate(new_column = px,
3          trial = trial + 5)

```

- but let's undo that...

```

1 df_lifetime <- df_lifetime %>%
2   mutate(trial = trial - 5)

```

if_else()

- can be used inside mutate()
 - change values based on some logical condition
 - can be used to change an existing column, or create a new one
- ifelse(condition, output_if_true, output_if_false)

```

1 df_lifetime <- df_lifetime %>%
2   mutate(new_column = if_else(name=="Aaliyah","name is Aaliyah","name is not Aaliyah"))

```

case_when()

- can be used inside mutate()
 - change values based on multiple logical conditions
 - can be used to change an existing column, or create a new one
- case_when(condition & other_condition | other_condition ~ output, TRUE ~ output_otherwise)
 - if you don't include TRUE ~ output then NAs will be created

```

1 df_lifetime <- df_lifetime %>%
2   mutate(newer_column = case_when(
3     name=="Aaliyah" & trial > 104 ~ "Aaliyah 2nd half",
4     name=="Beyoncé" & (px == "px01" | px == "px04") ~ "Beyoncé px04 or px06",
5     TRUE ~ "otherwise"))

```

Exercise

1. Create a new variable `accept` that checks whether the button pressed (`KeyPress`) equals the button that corresponds to an acceptance (`yes_press`)
 - if `KeyPress` and `yes_press` are the same, `accept` should be 1. If not, `accept` should be 0
 - hint: you will need `if_else()` or `case_when()`
2. Create a new variable `accuracy` where:
 - if `match` is `yes` and `accept` is 1, `accuracy` is 1
 - if `match` is `no` and `accept` is 0, `accuracy` is 1
 - if `match` is `yes` and `accept` is 0, `accuracy` is 0
 - if `match` is `no` and `accept` is 1, `accuracy` is 0
 - the means and summaries should look like this:

```
mean(df_lifetime$accept)
```

```
[1] 0.6068608
```

```
summary(as_factor(df_lifetime$accept))
```

```
  0    1  
1742 2689
```

```
mean(df_lifetime$accuracy)
```

```
[1] 0.6267208
```

```
summary(as_factor(df_lifetime$accuracy))
```

```
  0    1  
1654 2777
```

Extra exercise

3. Create a new variable `region`, that has the following values based on `region_n`

- `region_n` 1 is `region verb-1`
- `region_n` 2 is `region verb`
- `region_n` 3 is `region verb+1`
- `region_n` 4 is `region verb+2`
- `region_n` 5 is `region verb+3`
- `region_n` 6 is `region verb+4`

```
summary(as_factor(df_lifetime$region))
```

```
filler verb-1   verb verb+1 verb+2 verb+3 verb+4
1024    639    639    639    639    639    212
```

4. Now relocate our new variables so that:

- `region` is before `region_n`
- `KeyPress` is after `yes_press`

```
names(df_lifetime)
```

```
[1] "px"           "trial"        "region"       "region_n"
[5] "region_text"  "eye"          "ff"           "fp"
[9] "rpd"          "tt"           "fix_count"    "reg_in"
[13] "reg_in_count" "reg_out"      "reg_out_count" "rt"
[17] "bio"          "critical"     "gender"       "item_id"
[21] "list"         "match"        "condition"    "name"
[25] "lifetime"     "tense"        "type"         "yes_press"
[29] "KeyPress"     "new_column"   "newer_column" "accept"
[33] "accuracy"
```

`group_by()` and `ungroup()`

Group data by certain variable(s)

- then perform some mutation
- then ungroup the data

```
df_lifetime <- df_lifetime |>
  group_by(px) |>
  mutate(px_accuracy = mean(accuracy)) %>%
  ungroup()
```

```
round(
  range(df_lifetime$px_accuracy),
  2)
```

```
[1] 0.26 0.90
```

select()

- keep only certain column(s)

```
df_lifetime %>%
  select(px)
```

```
# A tibble: 4,431 x 1
```

```
  px
<chr>
1 px3
2 px3
3 px3
4 px3
5 px3
6 px3
7 px3
8 px3
9 px3
10 px3
```

```
# i 4,421 more rows
```

```
df_lifetime %>%
  select(px, trial)
```

```
# A tibble: 4,431 x 2
```

```
  px    trial
```

```

      <chr> <dbl>
1 px3      1
2 px3      2
3 px3      3
4 px3      3
5 px3      3
6 px3      3
7 px3      3
8 px3      3
9 px3      4
10 px3     5
# i 4,421 more rows

```

`select()`

- or remove certain columns

```

df_lifetime %>%
  select(-px, -trial)

```

A tibble: 4,431 x 32

	region	region_n	region_text	eye	ff	fp	rp	tt	fix_count	reg_in
	<chr>	<dbl>	<chr>	<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	filler	1	He owned innu~	RIGHT	0	0	0	0	0	0
2	filler	1	She is a moth~	RIGHT	0	0	0	0	0	0
3	verb-1	1	She	RIGHT	0	0	0	0	0	0
4	verb	2	will perform	RIGHT	0	0	0	0	0	0
5	verb+1	3	in prestigiou~	RIGHT	0	0	0	0	0	0
6	verb+2	4	in the future,	RIGHT	0	0	0	0	0	0
7	verb+3	5	as reported i~	RIGHT	0	0	0	0	0	0
8	verb+4	6	as reported i~	RIGHT	0	0	0	0	0	0
9	filler	1	He interviewe~	RIGHT	0	0	0	0	0	0
10	verb-1	1	She	RIGHT	0	0	0	0	0	0

i 4,421 more rows

i 22 more variables: reg_in_count <dbl>, reg_out <dbl>, reg_out_count <dbl>,
rt <dbl>, bio <chr>, critical <chr>, gender <chr>, item_id <dbl>,
list <dbl>, match <chr>, condition <chr>, name <chr>, lifetime <chr>,
tense <chr>, type <chr>, yes_press <dbl>, KeyPress <dbl>, new_column <chr>,
newer_column <chr>, accept <dbl>, accuracy <dbl>, px_accuracy <dbl>

Exercise

Remove the example variables we created with `mutate`:

- `new_column` and `newer_column`

```
# should look like this after
names(df_lifetime)
```

```
[1] "px"           "trial"        "region"        "region_n"
[5] "region_text"  "eye"          "ff"            "fp"
[9] "rpd"          "tt"           "fix_count"     "reg_in"
[13] "reg_in_count" "reg_out"      "reg_out_count" "rt"
[17] "bio"          "critical"     "gender"        "item_id"
[21] "list"         "match"        "condition"     "name"
[25] "lifetime"     "tense"        "type"          "yes_press"
[29] "KeyPress"     "accept"       "accuracy"      "px_accuracy"
```

`filter()`

- select certain rows based on certain criteria (`==`, `!=`, `>`, `<`, `|`)
 - N.B. when testing logical conditions `==` is needed

```
1 df_lifetime %>%
2   filter(trial == 1)
```

```
# A tibble: 8 x 32
```

	px	trial	region	region_n	region_text	eye	ff	fp	rpd	tt
	<chr>	<dbl>	<chr>	<dbl>	<chr>	<chr>	<dbl>	<dbl>	<dbl>	<dbl>
1	px3	1	filler	1	He owned innumerabl~	RIGHT	0	0	0	0
2	px5	1	filler	1	She is a mother of ~	RIGHT	145	1603	1603	1603
3	px6	1	filler	1	He is a father of t~	RIGHT	147	1224	1224	1224
4	px2	1	filler	1	She made innumerabl~	RIGHT	84	1829	1829	1829
5	px7	1	filler	1	In the '70s, he own~	RIGHT	138	2456	2456	2456
6	px1	1	filler	1	Beloved morning sho~	RIGHT	160	1708	1708	1708
7	px8	1	filler	1	She was a mother of~	RIGHT	220	806	806	806
8	px4	1	filler	1	In the '70s, he own~	LEFT	171	3557	3557	3557

```
# i 22 more variables: fix_count <dbl>, reg_in <dbl>, reg_in_count <dbl>,
#   reg_out <dbl>, reg_out_count <dbl>, rt <dbl>, bio <chr>, critical <chr>,
#   gender <chr>, item_id <dbl>, list <dbl>, match <chr>, condition <chr>,
```

```
# name <chr>, lifetime <chr>, tense <chr>, type <chr>, yes_press <dbl>,  
# KeyPress <dbl>, accept <dbl>, accuracy <dbl>, px_accuracy <dbl>
```

`filter()`

What are these code chunks doing?

```
1 df_lifetime %>%  
2   filter(px_accuracy > .5)
```

```
1 df_lifetime %>%  
2   filter(px == "px3")
```

```
1 df_lifetime %>%  
2   filter(px == "px3" | trial == "3")
```

```
1 df_lifetime %>%  
2   filter(px == "px3" & trial != "3")
```

💡 Logical operators

- symbols used to describe a logical condition
- `==` is identical (`1 == 1`)
- `!=` is not identical (`1 != 2`)
- `>` is greater than (`2 > 1`)
- `<` is less than (`1 < 2`)
- `&` and also (for multiple conditions)
- `|` or (for multiple conditions)

Exercise

1. Create a new dataframe `df_crit` that includes only critical trials

2. Create a new dataframe `df_fill` that includes only filler trials

- Tip: trial type is stored in the column `type`

```
df_crit |> select(type) |> head()
```

```
# A tibble: 6 x 1
  type
<chr>
1 critical
2 critical
3 critical
4 critical
5 critical
6 critical
```

```
df_fill |> select(type) |> head()
```

```
# A tibble: 6 x 1
  type
<chr>
1 filler
2 filler
3 filler
4 filler
5 filler
6 filler
```

distinct()

- like `filter()`, but for *distinct values* of a variable
 - “select rows with distinct values for some row(s)”

```
1 df_crit %>%
2   distinct(px)
```

```
# A tibble: 8 x 1
```

```
  px  
  <chr>
```

```
1 px3  
2 px5  
3 px6  
4 px2  
5 px7  
6 px1  
7 px8  
8 px4
```

```
1 df_crit %>%  
2   distinct(px, name)
```

```
# A tibble: 639 x 2
```

```
  px    name  
  <chr> <chr>  
1 px3 Edith Piaf  
2 px3 Aaliyah  
3 px3 David Beckham  
4 px3 Jana Novotna  
5 px3 Grace Kelly  
6 px3 Nigella Lawson  
7 px3 Coco Chanel  
8 px3 Ben Kingsley  
9 px3 Jim Carrey  
10 px3 Judy Garland  
# i 629 more rows
```

```
1 df_crit %>%  
2   distinct(px, name,  
3             .keep_all=T)
```

```
# A tibble: 639 x 32
```

	px	trial	region	region_n	region_text	eye	ff	fp	rp	pd	tt
	<chr>	<dbl>	<chr>	<dbl>	<chr>	<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	px3	3	verb-1	1	She	RIGHT	0	0	0	0	0
2	px3	5	verb-1	1	She	RIGHT	0	0	0	0	0
3	px3	8	verb-1	1	He	RIGHT	0	0	0	0	0

```

4 px3      10 verb-1      1 She      RIGHT      0      0      0      0
5 px3      13 verb-1      1 She      RIGHT      0      0      0      0
6 px3      16 verb-1      1 She      RIGHT      0      0      0      0
7 px3      18 verb-1      1 She      RIGHT      0      0      0      0
8 px3      21 verb-1      1 He       RIGHT      0      0      0      0
9 px3      23 verb-1      1 He       RIGHT      0      0      0      0
10 px3     26 verb-1      1 She      RIGHT      0      0      0      0
# i 629 more rows
# i 22 more variables: fix_count <dbl>, reg_in <dbl>, reg_in_count <dbl>,
#   reg_out <dbl>, reg_out_count <dbl>, rt <dbl>, bio <chr>, critical <chr>,
#   gender <chr>, item_id <dbl>, list <dbl>, match <chr>, condition <chr>,
#   name <chr>, lifetime <chr>, tense <chr>, type <chr>, yes_press <dbl>,
#   KeyPress <dbl>, accept <dbl>, accuracy <dbl>, px_accuracy <dbl>

```

`arrange()`

- sort column(s) in ascending or descending order
 - this is really just for ease of reading

```

# default: ascending order (A-Z)
df_crit %>%
  distinct(px, trial, name, condition) %>%
  arrange(px, trial)

```

```

# A tibble: 639 x 4
   px      trial name      condition
<chr> <dbl> <chr>      <chr>
1 px1         3 Amy Winehouse deadPP
2 px1         5 John Wayne   deadPP
3 px1         8 Abraham Lincoln deadPP
4 px1        10 Helen Mirren   livingSF
5 px1        13 Paul McCartney livingSF
6 px1        16 Ariana Grande livingPP
7 px1        18 Kate Middleton livingSF
8 px1        21 Johan Cruyff  deadSF
9 px1        23 Marilyn Monroe deadPP
10 px1        26 Biggie Smalls deadSF
# i 629 more rows

```

```
# descending order (Z-A)
df_crit %>%
  distinct(px, trial, name, condition) %>%
  arrange(desc(px), trial)
```

```
# A tibble: 639 x 4
  px      trial name      condition
<chr> <dbl> <chr>      <chr>
1 px8      3 Whitney Houston deadPP
2 px8      5 Elton John    livingSF
3 px8      8 Jackie Chan    livingPP
4 px8     10 Romy Schneider deadPP
5 px8     13 James Cameron livingSF
6 px8     16 Ella Fitzgerald deadSF
7 px8     18 Kathryn Hepburn deadPP
8 px8     21 Kate Middleton livingPP
9 px8     23 Janis Joplin  deadPP
10 px8     26 Serena Williams livingSF
# i 629 more rows
```

separate()

- create new columns from a single column

```
df_crit<- df_crit %>%
  separate(name,
    sep=" ",
    into = c("First","Last"),
    remove = F, # don't remove original column (name)
    extra = "merge") # if extra chunks, combine in 'Last' (von der...)
```

- opposite: unite()

pivot_wider()

pivot_longer()

Save your tidy data

- once your data is nice and tidy, save it with a **new filename**

- this way you always have the same starting point for your data exploration/analyses

```
# run this manually!  
write.csv(df_lifetime, here::here("data/tidy_data_lifetime_pilot.csv"),row.names=FALSE)
```

Summary

- we saw that the equation for a straight line boils down to its intercept and slope
- we fit our first linear model with a categorical predictor
- next, we'll look at a case with more than one predictor: **multiple** regression

Important terms

wrangle	have a long dispute
data wrangling	tidying and transforming your data
tidy data	data where each column is a variable and each row is an observation
the tidyverse	a group of packages for tidy data
dplyr	a package within the tidyverse for data wrangling
pipe operator (%>% or >)	operational function, passes the result of one function/argument to the next
logical operators	compare values of two arguments: &, , ==, !=, >, <

Important functions

read_csv()	log-transform a continuous variable
rename()	rename variables
relocate()	move variables
mutate()	change or create new variables
if_else()	condition for 'mutate()'
case_when()	can handle multiple conditions for 'mutate()'
group_by()	group by a certain variable
select()	keep (or exclude) certain variables
filter()	keep (or exclude) rows based on some criteria
distinct()	keep rows with distinct value of given variable(s)
arrange()	sort variable(s) in ascending or descending order
separate()	split a variable into multiple variables

<code>pivot_longer()</code>	make wide data longer
<code>pivot_wider()</code>	make long data wider

Session Info

```
sessionInfo()
```

```
R version 4.2.3 (2023-03-15)
Platform: aarch64-apple-darwin20 (64-bit)
Running under: macOS Ventura 13.2.1
```

```
Matrix products: default
```

```
BLAS:   /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/lib/libRblas.0.dylib
```

```
LAPACK: /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/lib/libRlapack.dylib
```

```
locale:
```

```
[1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
```

```
attached base packages:
```

```
[1] stats      graphics  grDevices  utils      datasets  methods    base
```

```
other attached packages:
```

```
[1] lubridate_1.9.2 forcats_1.0.0  stringr_1.5.0  dplyr_1.1.1
[5] purrr_1.0.1     readr_2.1.4    tidyr_1.3.0    tibble_3.2.1
[9] ggplot2_3.4.2   tidyverse_2.0.0
```

```
loaded via a namespace (and not attached):
```

```
[1] pillar_1.9.0      compiler_4.2.3  tools_4.2.3     bit_4.0.5
[5] digest_0.6.31     timechange_0.2.0 jsonlite_1.8.4  evaluate_0.20
[9] lifecycle_1.0.3   gtable_0.3.3    pkgconfig_2.0.3 png_0.1-8
[13] rlang_1.1.0        cli_3.6.1       rstudioapi_0.14 parallel_4.2.3
[17] curl_5.0.0         yaml_2.3.7      xfun_0.38       fastmap_1.1.1
[21] withr_2.5.0        httr_1.4.5      knitr_1.42      generics_0.1.3
[25] vctrs_0.6.1        fs_1.6.1        hms_1.1.3       bit64_4.0.5
[29] tidyselect_1.2.0   rprojroot_2.0.3 grid_4.2.3       glue_1.6.2
[33] here_1.0.1         R6_2.5.1        fansi_1.0.4     vroom_1.6.1
[37] rmarkdown_2.21     tzdb_0.3.0      magrittr_2.0.3  scales_1.2.1
[41] htmltools_0.5.5    rbbt_0.0.0.9000 colorspace_2.1-0 utf8_1.2.3
[45] stringi_1.7.12     munsell_0.5.0   crayon_1.5.2
```

References

- Laurinavichyute, A., Yadav, H., & Vasishth, S. (2022). Share the code, not just the data: A case study of the reproducibility of articles published in the Journal of Memory and Language under the open data policy. *Journal of Memory and Language*, 125, 12.
- Wickham, H., Çetinkaya-Rundel, M., & Grolemund, G. (n.d.). *R for Data Science* (2nd ed.). <https://r4ds.hadley.nz/>