Lab 1. Data Wrangling

PUBH 6199: Visualizing Data with R, Summer 2025

Xindi (Cindy) Hu, ScD

2025-05-22

Outline for today

- **Introduction to GitHub and Git**
- Data transformation
- Data tidying

About GitHub

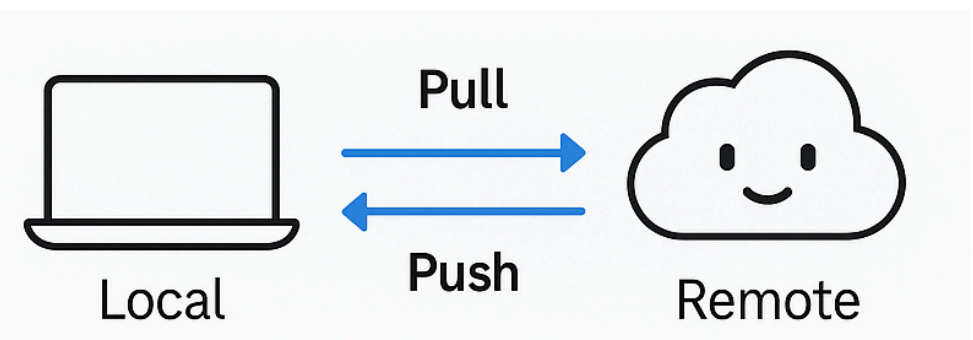
- GitHub is a cloud-based platform for version control and collaboration.
- Storing your code in a “repository” on GitHub allows you to:
  - Track changes to your code over time
  - Collaborate with others on projects, including your “future self”
  - Share your work with the world
- Made possible by the open-source software, Git

About Git

- Git is a version control system that allows you to track changes to files.
- A typical Git-based workflow includes:
  - **Clone** a repository from GitHub to your local machine
  - **Branch** off the main copy of the files that you are working on
  - **Edit** files independently and safely on your own branch
  - Let Git **keep track** of the changes you and others make
  - Let Git intelligently **merge** your changes back into the main copy of the files

How do Git and GitHub work together?

- What is a **Git repository**?
  - A collection of files and their history, can be local (on your computer) or **remote** (on GitHub)
  - When you make changes (or **commits**) to the files, Git keeps track of the changes
- Plenty to do in your browser
  - Create a Git repository, create branches, upload and edit files
- But, most people work locally, then continue to sync local changes with the **remote** repository on GitHub
  - Use Git commands in the terminal or GitHub Desktop
  - **Pull** the latest changes from the remote repository
  - **Push** back your own changes to the same remote repository



In-Class Activity:

GitHub and RStudio tutorial

Prerequisites

- You have a GitHub account
- You have downloaded and installed Git
- You have downloaded and installed RStudio

5 minutes to catch up on these if you haven't done so already!

05:00

Create the remote repository on GitHub

- Accept the invitation to join the GitHub Classroom, check your email
- Accept the assignment titled lab 1, check your email or use this link **https://classroom.github.com/a/XXXX**
- Navigate to GitHub, under the class organization, you should see a repository named `lab1-<your-github-username>`.

Clone the repository with RStudio

1. On GitHub, navigate to the **Code** tab of the repository
2. Click the green **<> Code** button
3. Clcik the **Copy to clipboard** button to copy the repository URL
4. Open RStudio on your local environment
5. Click **File**, **New Project**, **Version Control**, **Git**
6. Paste the URL you copied from GitHub into the **Repository URL** field and enter TAB to move to the **Project directory name** field
7. Click **Create Project**

Edit the lab notebook in RStudio

1. In RStudio, click **Files**, **Open File**, and select **1-lab1.qmd**
2. Update the header - put your name in the `author` argument and put today's date in the `date` argument.
3. Save the file, and click **Render** to generate the HTML file.

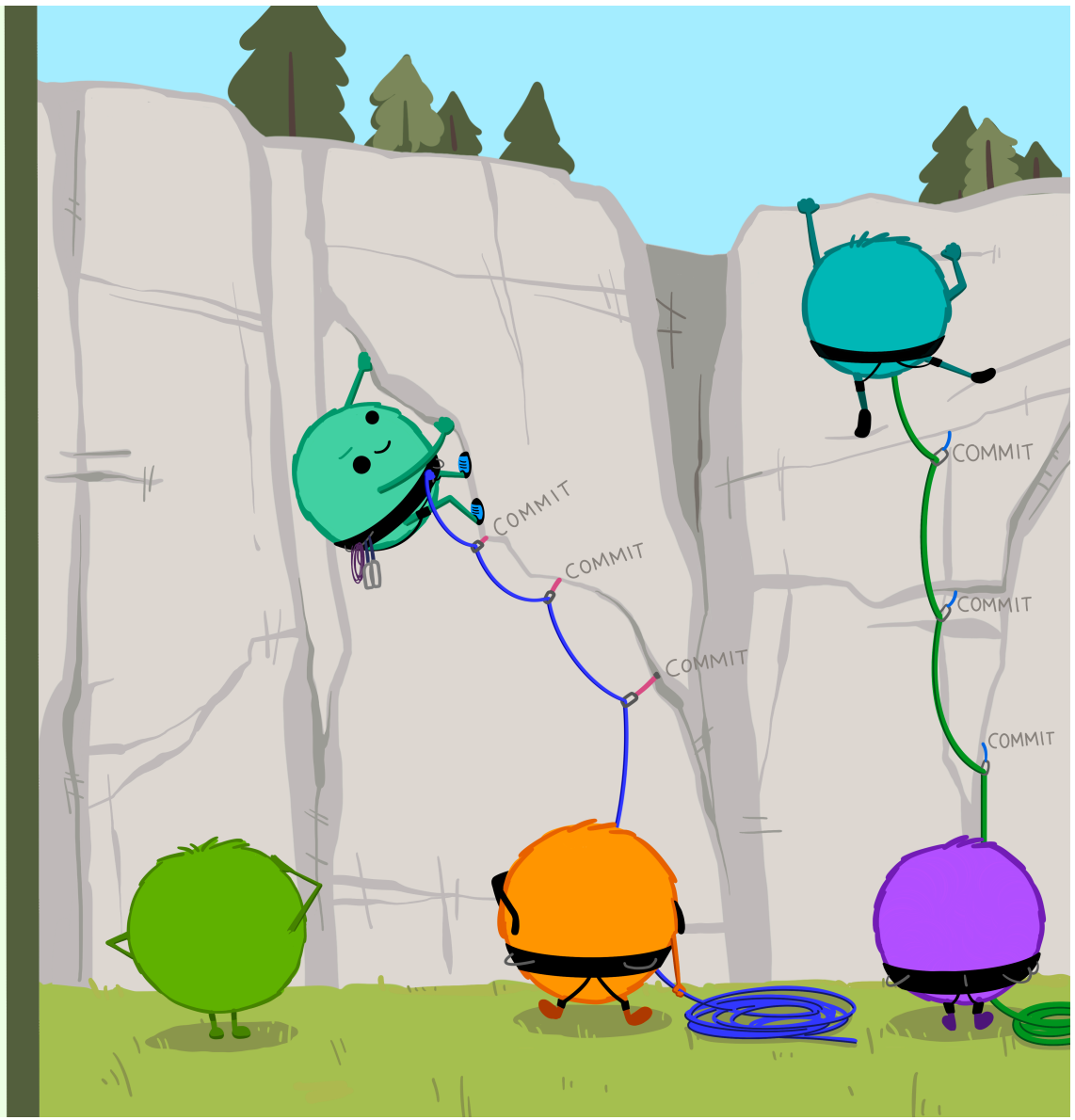
Commit and push the changes to GitHub

1. In RStudio, click the **Git** tab in the upper right pane
2. Click **Commit**
3. In the **Commit** window, check the box next to the file you want to commit (`1-lab1.qmd` and `1-lab1.html`)
4. Enter a commit message in the **Commit message** field (e.g., "Update lab notebook header")
5. Click the **Commit** button
6. Click the **Pull** button to fetch any remote changes
7. Click the **Push** button to push your changes to GitHub
8. Navigate to your GitHub repository in your browser and check that the changes have been pushed successfully

Congratulations!

> Using a Git commit is like using anchors and other protection when climbing…**if you make a mistake, you can't fall past the previous commit**.
>
> Commits are also helpful to others, because **they show your journey, not just the destination**.
>
> – HADLEY WICKHAM & JENNY BRYAN

Wickham & Bryan, RPackages (https://r-packages.org/preface.html)



Introducing GitHub Flow

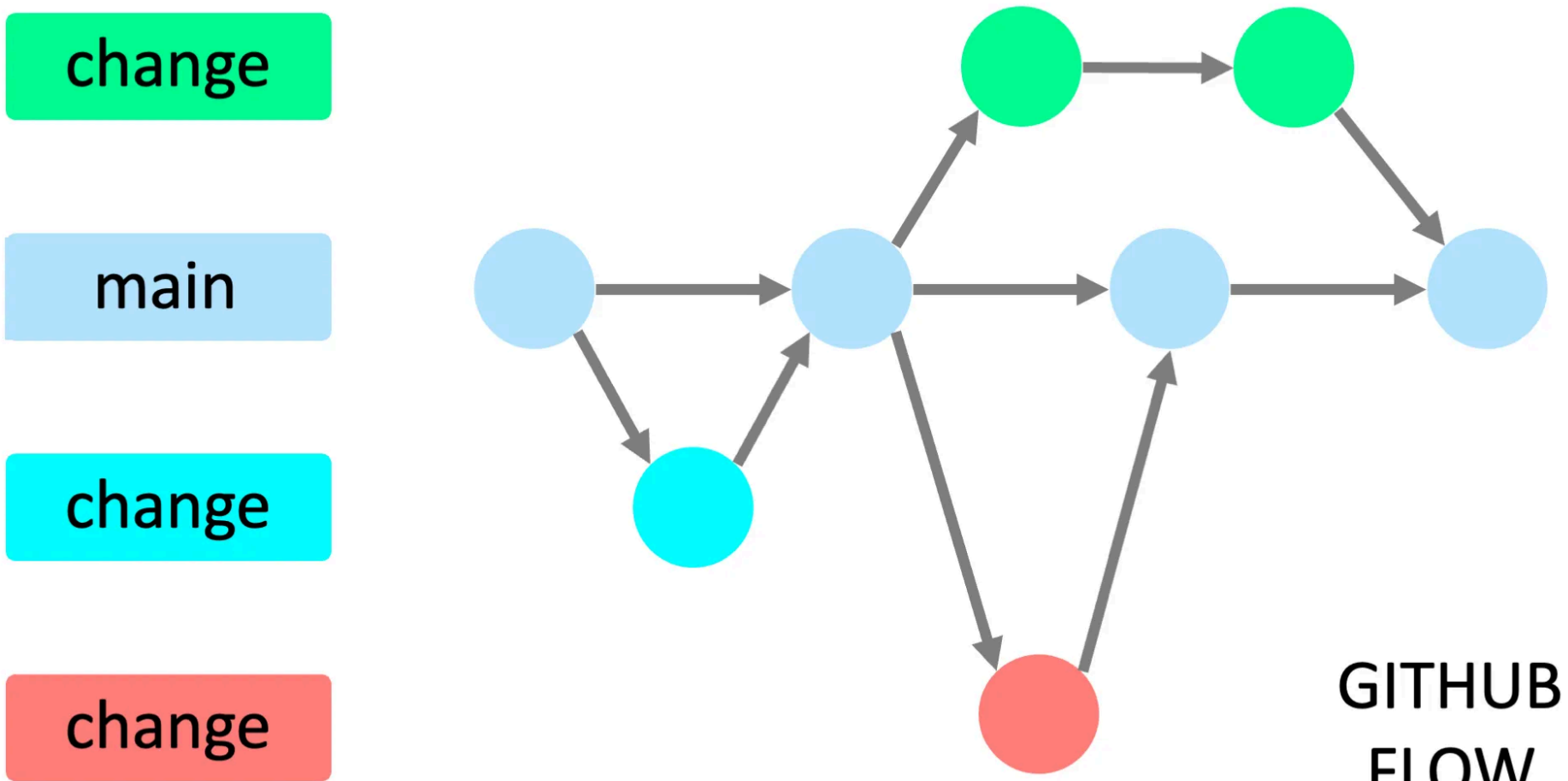


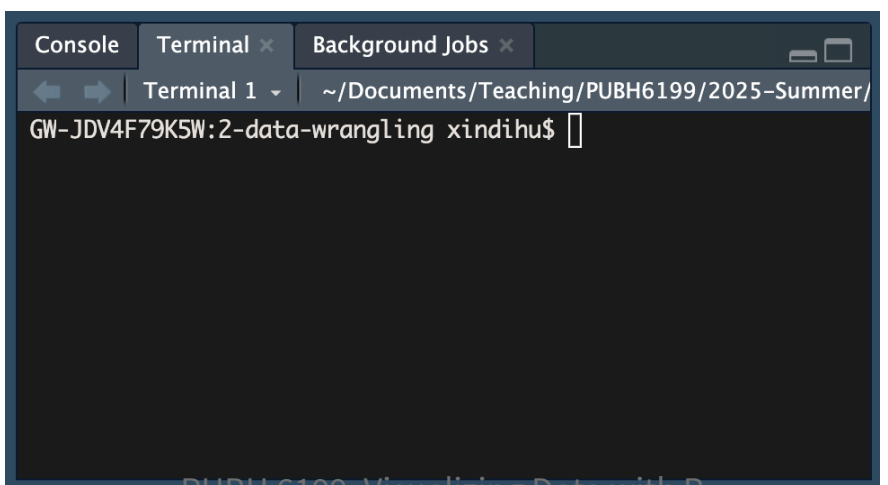
Image by Yan Min Thwin

Create local branches with Git

> **Tip**
>
> You can do these using the Git GUI in RStudio, I am showing you the command line version so you can learn a different method and choose what you prefer.

1. In RStudio click the **Terminal** tab in the lower left pane, next to the **Console** tab

> **Note**
>
> If you cannot find the **Terminal** tab, you can also open a terminal window by clicking on the **Tools** menu and selecting **Terminal** > **New Terminal**. If that doesn't work, check if your RStudio is out of date. Click **Help**, About RStudio to check the current version.



Create local branches with Git

2. In the terminal, type the following command to create a new branch called `feat/clean-data`:

```
git checkout -b feat/clean-data
```

3. Type the following command to check that you are on the new branch:

```
git status
```

You should see a message that says “On branch feat/clean-data” and “nothing to commit, working tree clean”.

You are ready to start making changes to your files!

Make local changes with Git

In RStudio, open the `1-lab1.qmd` file and make some changes to the text.

For example, you can add a new section called "Data Wrangling" and write a few sentences about what tidy data is about.

You can also add a new code chunk to the file and write some R code to load the `tidyverse` package and read in a CSV file.

```
library(tidyverse)
raw_data <- read_csv("raw_data.csv")
```

After you are satisfied with your changes, save the file and knit the `1-lab1.qmd` file to generate the HTML file.

Commit local changes with Git

1. Determine your file's status.

```
1 git status
```

You should see a message that says **"On branch feat/clean-data"** and **"Changes not staged for commit"**.

2. Add the changes to the staging area.

```
1 git add .
```

3. See your file's current status.

```
1 git status
```

Your files should be listed under **Changes to be committed**.

4. Commit the changes with a message. Replace with a log message describing the changes.

```
1 git commit -m "<COMMIT-MESSAGE>"
```


Open a pull request on GitHub

1. Push the changes to the remote repository, replace with the name of your branch, in this case **feat/clean-data**

```
git push origin <BRANCH-NAME>
```

2. Navigate to your GitHub repository in your browser
3. Click the **Compare & pull requests** button, if you don't see it, navigate to the **"Pull requests"** tab and click the **New pull reques** button.
4. In the **"Open a pull request"** page, enter a title and description for your pull request. You can add a reviewer, for example your teammate on this pull request.

Merge your pull request on GitHub

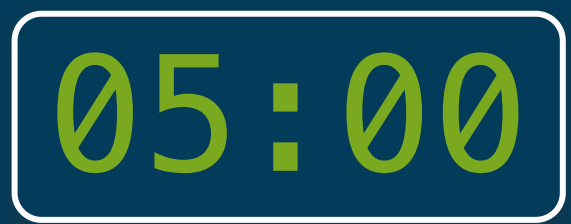
> **Note**
>
> Since this is your repository, you probably don't have anyone to collaborate with (yet). Go ahead and merge your Pull Request now. Later in the semester you may want your teammate to look over your code before they merge.

1. On GitHub, navigate to the Pull Request that you just opened.
2. Scroll down and click the big green Merge Pull Request button.
3. Click Confirm Merge.
4. Delete the branch .

Reference: GitHub and RStudio

☕ Take a Break

~ This is the end of part 1 ~

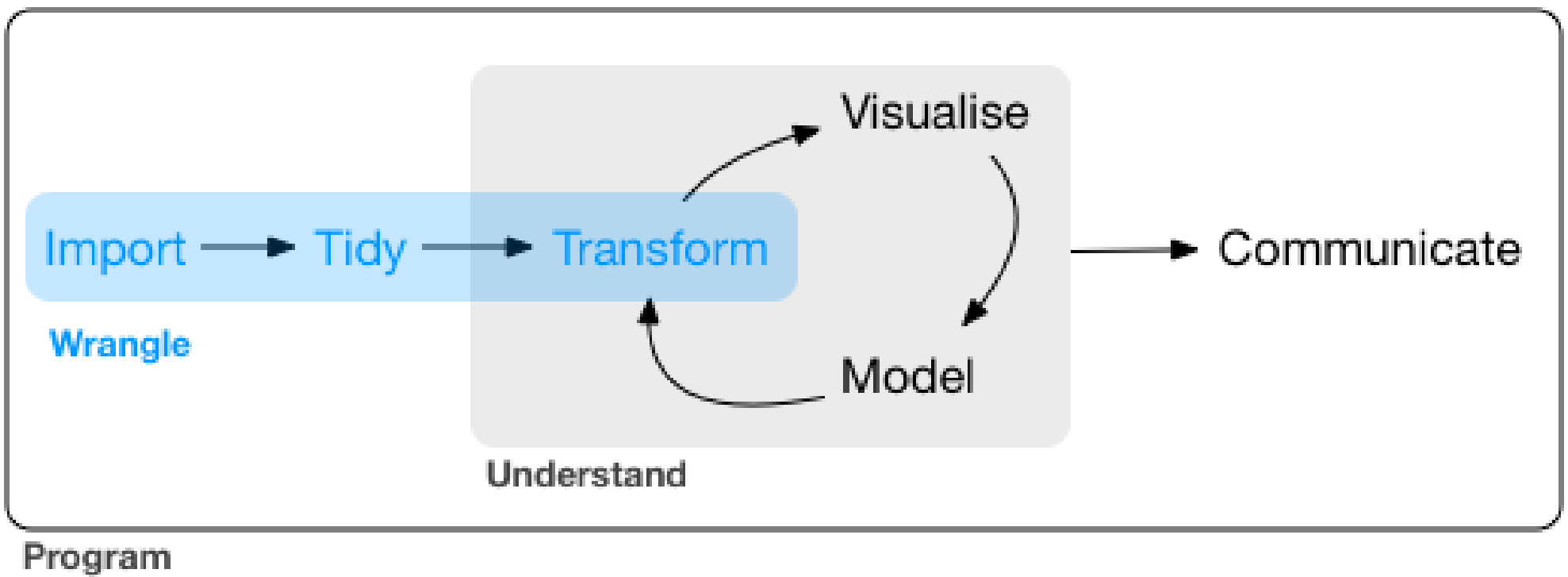


Outline for today

- Introduction to GitHub and Git
- **Data transformation**
- Data tidying

“80% of data scientists’ time is spent on data wrangling”

Data wrangling: also known as data cleaning or data preparation, is the process of collecting, cleaning, transforming and organizing data from one “raw” form into another format with the intent of making it more appropriate for analysis.



Source: R for Data Science

Manipulate data in R using `dplyr`

Commonality:

- The first argument is always a data frame
- The sebsequent arguments are the columns of the data frame (without quotes)
- The output is a new data frame

Individuality:

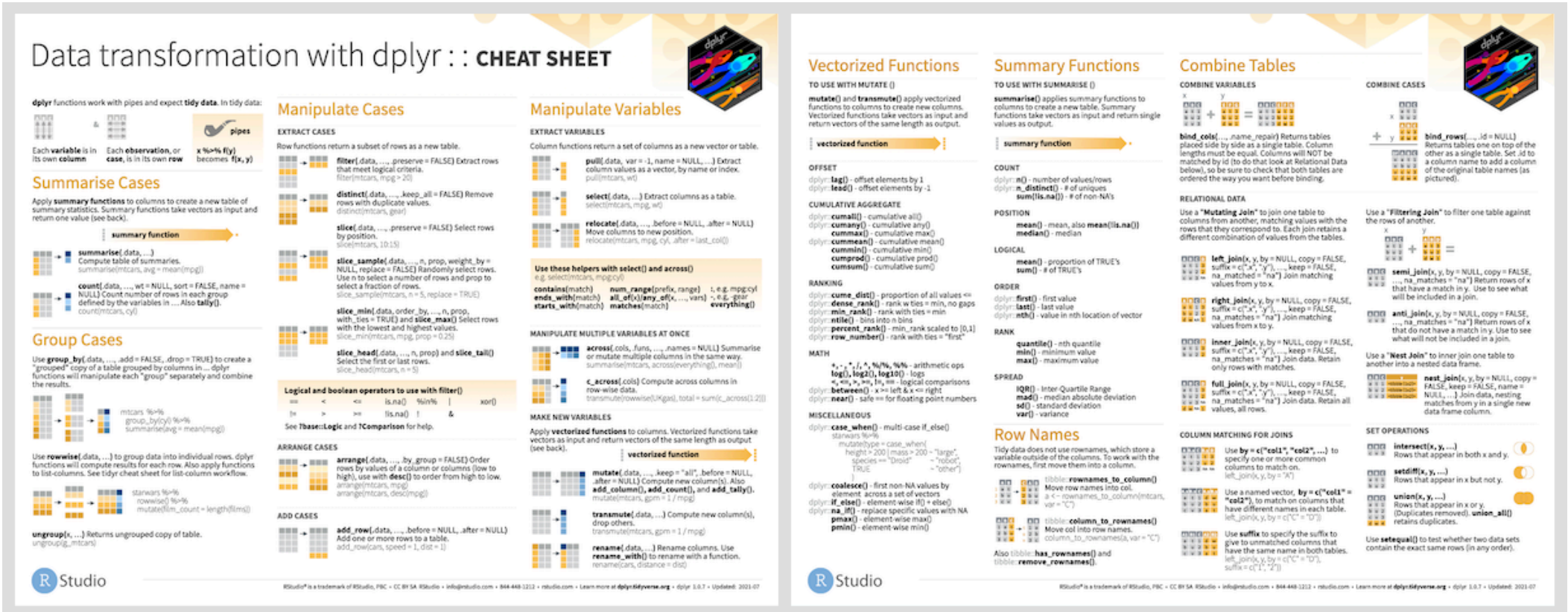
A word on pipe

`%>%` in `{magrittr}` or `|>` in base R

- Pipe is a tool to combine multiple verbs.
- It takes the thing on the left and passes it to the function on the right.
- `x |> f(y)` is equivalent to `f(x, y)`
- `x |> f(y) |> g(z)` is equivalent to `g(f(x, y), z)`.
- Pronounces as “then”
- Add pipe to your code using keyboard shortcut Ctrl/Cmd + Shift + M

```
flights |>
  filter(dest == "IAH") |>
  group_by(year, month, day) |>
  summarize(
    arr_delay = mean(arr_delay, na.rm = TRUE)
  )
```


Lots of verbs to remember!



Data transformation with dplyr : : CHEAT SHEET

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

Apply summary functions to columns to create a new table of summary statistics. Summary functions take vectors as input and return one value (see back).

summary function

summarise(.data, ...) Compute table of summaries.

count(.data, ..., wt = NULL, sort = FALSE, name = NULL) Count number of rows in each group defined by the variables in ... Also tally().

Group Cases

Use group_by(.data, ..., .add = FALSE, .drop = TRUE) to create a "grouped" copy of a table grouped by columns in ... dplyr functions will manipulate each "group" separately and combine the results.

Use rowwise(.data, ...) to group data into individual rows. dplyr functions will compute results for each row. Also apply functions to list-columns. See tidyr cheat sheet for list-column workflow.

ungroup(x, ...) Returns ungrouped copy of table.

Manipulate Cases

EXTRACT CASES

Row functions return a subset of rows as a new table.

filter(.data, ..., .preserve = FALSE) Extract rows that meet logical criteria.

distinct(.data, ..., .keep_all = FALSE) Remove rows with duplicate values.

slice(.data, ..., .preserve = FALSE) Select rows by position.

slice_sample(.data, ..., n, prop, weight_by = NULL, replace = FALSE) Randomly select rows. Use n to select a number of rows and prop to select a fraction of rows.

slice_min(.data, order_by, ..., n, prop, with_ties = TRUE) and slice_max() Select rows with the lowest and highest values.

slice_head(.data, ..., n, prop) and slice_tail() Select the first or last rows.

Logical and boolean operators to use with filter()

See ?base::Logic and ?Comparison for help.

ARRANGE CASES

arrange(.data, ..., .by_group = FALSE) Order rows by values of a column or columns (low to high), use with desc() to order from high to low.

ADD CASES

add_row(.data, ..., .before = NULL, .after = NULL) Add one or more rows to a table.

Manipulate Variables

EXTRACT VARIABLES

Column functions return a set of columns as a new vector or table.

pull(.data, var = -1, name = NULL, ...) Extract column values as a vector, by name or index.

select(.data, ...) Extract columns as a table.

relocate(.data, ..., .before = NULL, .after = NULL) Move columns to new position.

Use these helpers with select() and across()

MANIPULATE MULTIPLE VARIABLES AT ONCE

across(.cols, .funs, ..., .names = NULL) Summarise or mutate multiple columns in the same way.

c_across(.cols) Compute across columns in row-wise data.

MAKE NEW VARIABLES

Apply vectorized functions to columns. Vectorized functions take vectors as input and return vectors of the same length as output (see back).

vectorized function

mutate(.data, ..., .keep = "all", .before = NULL, .after = NULL) Compute new column(s). Also add_column(), add_count(), and add_tally().

transmute(.data, ...) Compute new column(s), drop others.

rename(.data, ...) Rename columns. Use rename_with() to rename with a function.

Vectorized Functions

TO USE WITH MUTATE ()

mutate() and transmute() apply vectorized functions to columns to create new columns. Vectorized functions take vectors as input and return vectors of the same length as output.

vectorized function

OFFSET

dplyr::lag() - offset elements by 1
dplyr::lead() - offset elements by -1

CUMULATIVE AGGREGATE

dplyr::cumall() - cumulative all()
dplyr::cumany() - cumulative any()
cummax() - cumulative max()
dplyr::cummean() - cumulative mean()
cummin() - cumulative min()
cumprod() - cumulative prod()
cumsum() - cumulative sum()

RANKING

dplyr::cume_dist() - proportion of all values <=
dplyr::dense_rank() - rank w ties = min, no gaps
dplyr::min_rank() - rank with ties = min
dplyr::ntile() - bins into n bins
dplyr::percent_rank() - min_rank scaled to [0,1]
dplyr::row_number() - rank with ties = "first"

MATH

+, -, *, /, ^, %/%, %% - arithmetic ops
log(), log2(), log10() - logs
<, <=, >, >=, !=, == - logical comparisons
dplyr::between() - x >= left & x <= right
dplyr::near() - safe == for floating point numbers

MISCELLANEOUS

dplyr::case_when() - multi-case if_else()
dplyr::coalesce() - first non-NA values by element across a set of vectors
dplyr::if_else() - element-wise if() + else()
dplyr::na_if() - replace specific values with NA
pmax() - element-wise max()
pmin() - element-wise min()

Summary Functions

TO USE WITH SUMMARISE ()

summarise() applies summary functions to columns to create a new table. Summary functions take vectors as input and return single values as output.

summary function

COUNT

dplyr::n() - number of values/rows
dplyr::n_distinct() - # of uniques
sum(!is.na()) - # of non-NA's

POSITION

mean() - mean, also mean(!is.na())
median() - median

LOGICAL

mean() - proportion of TRUE's
sum() - # of TRUE's

ORDER

dplyr::first() - first value
dplyr::last() - last value
dplyr::nth() - value in nth location of vector

RANK

quantile() - nth quantile
min() - minimum value
max() - maximum value

SPREAD

IQR() - Inter Quartile Range
mad() - median absolute deviation
sd() - standard deviation
var() - variance

Row Names

Tidy data does not use rownames, which store a variable outside of the columns. To work with the rownames, first move them into a column.

tibble::rownames_to_column() Move row names into col.

tibble::column_to_rownames() Move col into row names.

Also tibble::has_rownames() and tibble::remove_rownames().

Combine Tables

COMBINE VARIABLES

bind_cols(..., .name_repair) Returns tables placed side by side as a single table. Column lengths must be equal. Columns will NOT be matched by id (to do that look at Relational Data below), so be sure to check that both tables are ordered the way you want before binding.

COMBINE CASES

bind_rows(..., .id = NULL) Returns tables one on top of the other as a single table. Set .id to a column name to add a column of the original table names (as pictured).

RELATIONAL DATA

Use a "Mutating Join" to join one table to columns from another, matching values with the rows that they correspond to. Each join retains a different combination of values from the tables.

left_join(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), ..., keep = FALSE, na_matches = "na") Join matching values from y to x.

right_join(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), ..., keep = FALSE, na_matches = "na") Join matching values from x to y.

inner_join(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), ..., keep = FALSE, na_matches = "na") Join data. Retain only rows with matches.

full_join(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), ..., keep = FALSE, na_matches = "na") Join data. Retain all values, all rows.

Use a "Filtering Join" to filter one table against the rows of another.

semi_join(x, y, by = NULL, copy = FALSE, ..., na_matches = "na") Return rows of x that have a match in y. Use to see what will be included in a join.

anti_join(x, y, by = NULL, copy = FALSE, ..., na_matches = "na") Return rows of x that do not have a match in y. Use to see what will not be included in a join.

Use a "Nest Join" to inner join one table to another into a nested data frame.

nest_join(x, y, by = NULL, copy = FALSE, keep = FALSE, name = NULL, ...) Join data, nesting matches from y in a single new data frame column.

COLUMN MATCHING FOR JOINS

Use by = c("col1", "col2", ...) to specify one or more common columns to match on.

Use a named vector, by = c("col1" = "col2"), to match on columns that have different names in each table.

Use suffix to specify the suffix to give to unmatched columns that have the same name in both tables.

SET OPERATIONS

intersect(x, y, ...) Rows that appear in both x and y.

setdiff(x, y, ...) Rows that appear in x but not y.

union(x, y, ...) Rows that appear in x or y. (Duplicates removed). union_all() retains duplicates.

Use setequal() to test whether two data sets contain the exact same rows (in any order).

RStudio® is a trademark of RStudio, PBC • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more at dplyr.tidyverse.org • dplyr 1.0.7 • Updated: 2021-07

Refer to this cheat sheet

🎯 Practice makes perfect!

~ Head over to lab1 notebook! ~

Outline for today

- Introduction to GitHub and Git
- Data transformation
- **Data tidying**

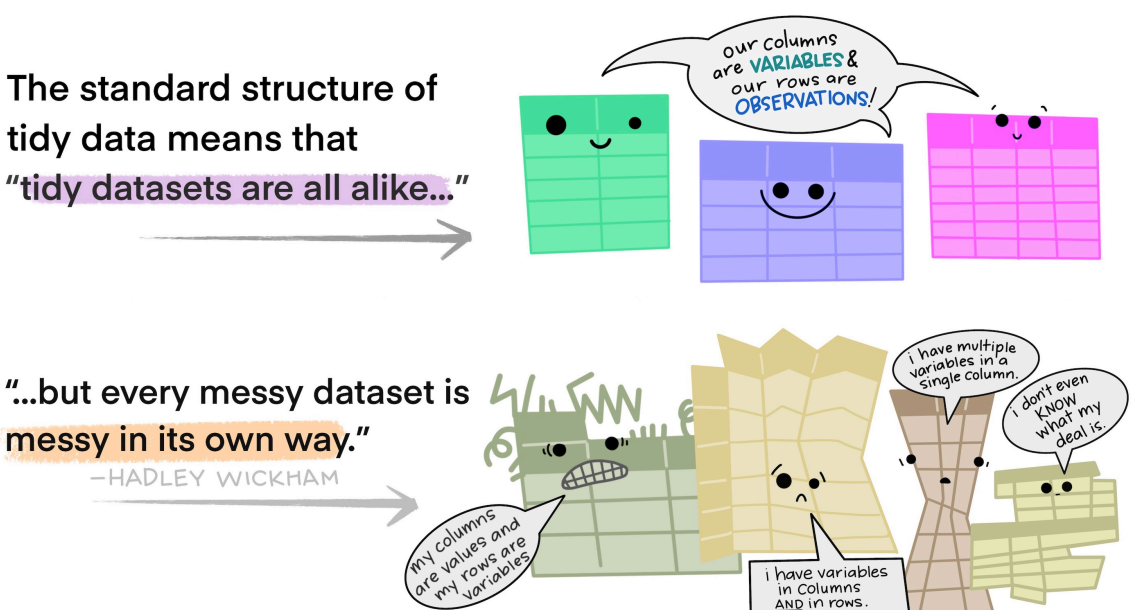
Introduction to tidy data

> "Happy families are all alike; every unhappy family is unhappy in its own way."

- Leo Tolstoy, *Anna Karenina*

> "Tidy datasets are all alike, but every messy dataset is messy in its own way."

- Hadley Wickham, *Tidy Data*

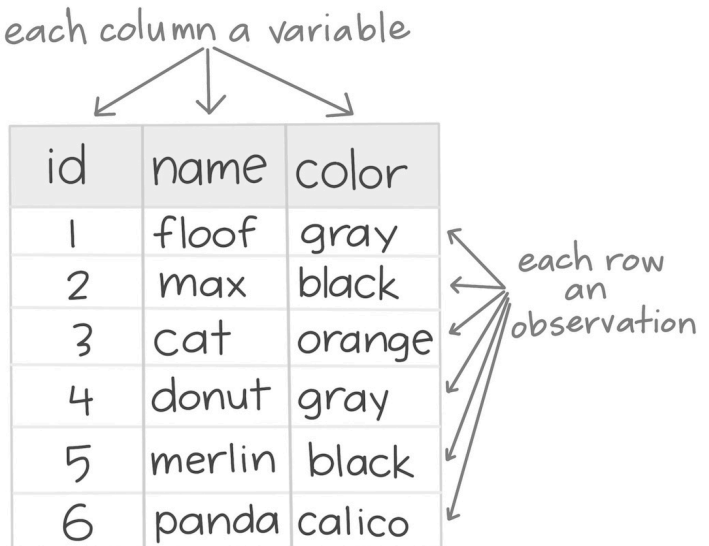


What is tidy data?

> "TIDY DATA is a standard way of mapping the meaning of a dataset to its structure."
>
> –HADLEY WICKHAM

In tidy data:

- each variable forms a column
- each observation forms a row
- each cell is a single measurement



Wickham, H. (2014). Tidy Data. Journal of Statistical Software *59* (10). DOI: 10.18637/jss.v059.i10

By Julia Lowndes and Allison Horst

What is an example of untidy data?

AtlasGroveCOMPLETE.xls

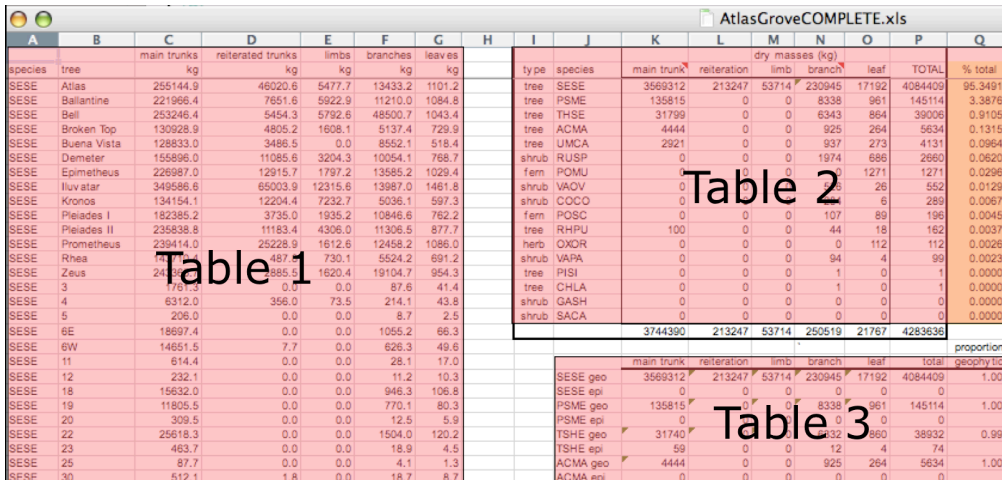
species	tree	main trunks kg	reiterated trunks kg	limbs kg	branches kg	leaves kg
SESE	Atlas	255144.9	46020.6	5477.7	13433.2	1101.2
SESE	Ballantine	221966.4	7651.6	5922.9	11210.0	1084.8
SESE	Bell	253246.4	5454.3	5792.6	48500.7	1043.4
SESE	Broken Top	130928.9	4805.2	1608.1	5137.4	729.9
SESE	Buena Vista	128833.0	3486.5	0.0	8552.1	518.4
SESE	Demeter	155896.0	11085.6	3204.3	10054.1	768.7
SESE	Epimetheus	226987.0	12915.7	1797.2	13585.2	1029.4
SESE	Iluvatar	349586.6	65003.9	12315.6	13987.0	1461.8
SESE	Kronos	134154.1	12204.4	7232.7	5036.1	597.3
SESE	Pleiades I	182385.2	3735.0	1935.2	10846.6	762.2
SESE	Pleiades II	235838.8	11183.4	4306.0	11306.5	877.7
SESE	Prometheus	239414.0	25228.9	1612.6	12458.2	1086.0
SESE	Rhea	143710.4	487.8	730.1	5524.2	691.2
SESE	Zeus	243365.7	2885.5	1620.4	19104.7	954.3
SESE	3	1761.3	0.0	0.0	87.6	41.4
SESE	4	6312.0	356.0	73.5	214.1	43.8
SESE	5	206.0	0.0	0.0	8.7	2.5
SESE	6E	18697.4	0.0	0.0	1055.2	66.3
SESE	6W	14651.5	7.7	0.0	626.3	49.6
SESE	11	614.4	0.0	0.0	28.1	17.0
SESE	12	232.1	0.0	0.0	11.2	10.3
SESE	18	15632.0	0.0	0.0	946.3	106.8
SESE	19	11805.5	0.0	0.0	770.1	80.3
SESE	20	309.5	0.0	0.0	12.5	5.9
SESE	22	25618.3	0.0	0.0	1504.0	120.2
SESE	23	463.7	0.0	0.0	18.9	4.5
SESE	25	87.7	0.0	0.0	4.1	1.3
SESE	30	512.1	1.8	0.0	18.7	8.7

type	species	dry masses (kg) main trunk	reiteration	limb	branch	leaf	TOTAL	% total
tree	SESE	3569312	213247	53714	230945	17192	4084409	95.3491
tree	PSME	135815	0	0	8338	961	145114	3.3876
tree	THSE	31799	0	0	6343	864	39006	0.9105
tree	ACMA	4444	0	0	925	264	5634	0.1315
tree	UMCA	2921	0	0	937	273	4131	0.0964
shrub	RUSP	0	0	0	1974	686	2660	0.0620
fern	POMU	0	0	0	0	1271	1271	0.0296
shrub	VAOV	0	0	0	526	26	552	0.0129
shrub	COCO	0	0	0	284	6	289	0.0067
fern	POSC	0	0	0	107	89	196	0.0045
tree	RHPU	100	0	0	44	18	162	0.0037
herb	OXOR	0	0	0	0	112	112	0.0026
shrub	VAPA	0	0	0	94	4	99	0.0023
tree	PISI	0	0	0	1	0	1	0.0000
tree	CHLA	0	0	0	1	0	1	0.0000
shrub	GASH	0	0	0	0	0	0	0.0000
shrub	SACA	0	0	0	0	0	0	0.0000
		3744390	213247	53714	250519	21767	4283636	

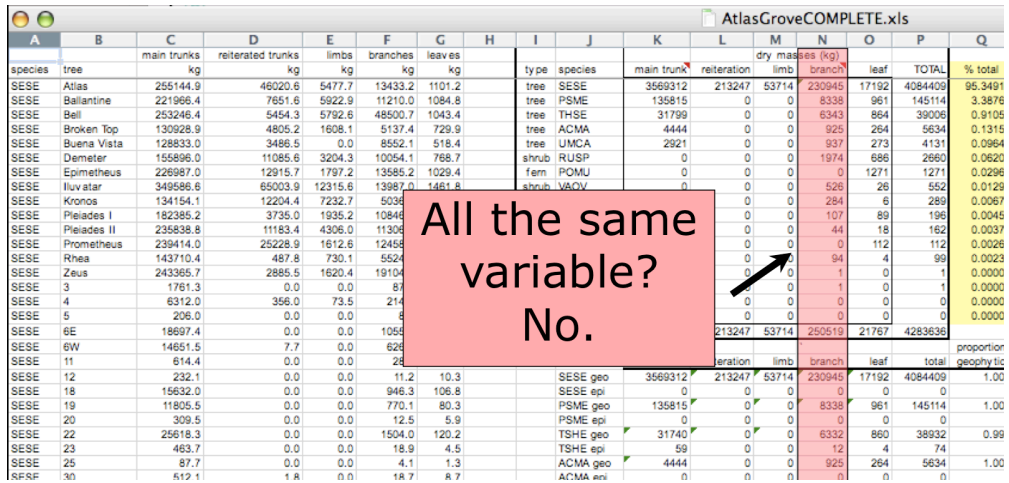
	main trunk	reiteration	limb	branch	leaf	total	proportion geophytic
SESE geo	3569312	213247	53714	230945	17192	4084409	1.00
SESE epi	0	0	0	0	0	0	
PSME geo	135815	0	0	8338	961	145114	1.00
PSME epi	0	0	0	0	0	0	
TSHE geo	31740	0	0	6332	860	38932	0.99
TSHE epi	59	0	0	12	4	74	
ACMA geo	4444	0	0	925	264	5634	1.00
ACMA epi	0	0	0	0	0	0	

Source: National Center for Ecological Analysis & Synthesis

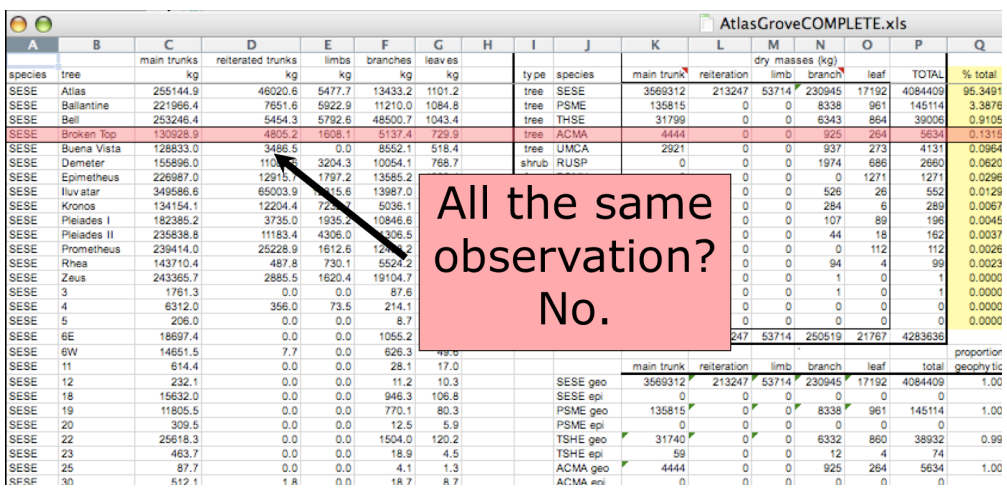
Multiple tables, not machine-readable



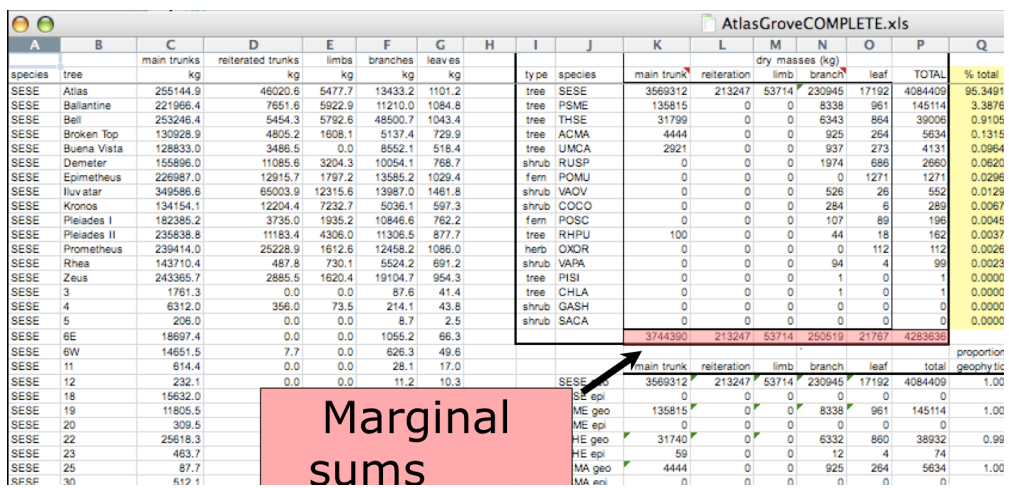
Inconsistent columns



Inconsistent rows

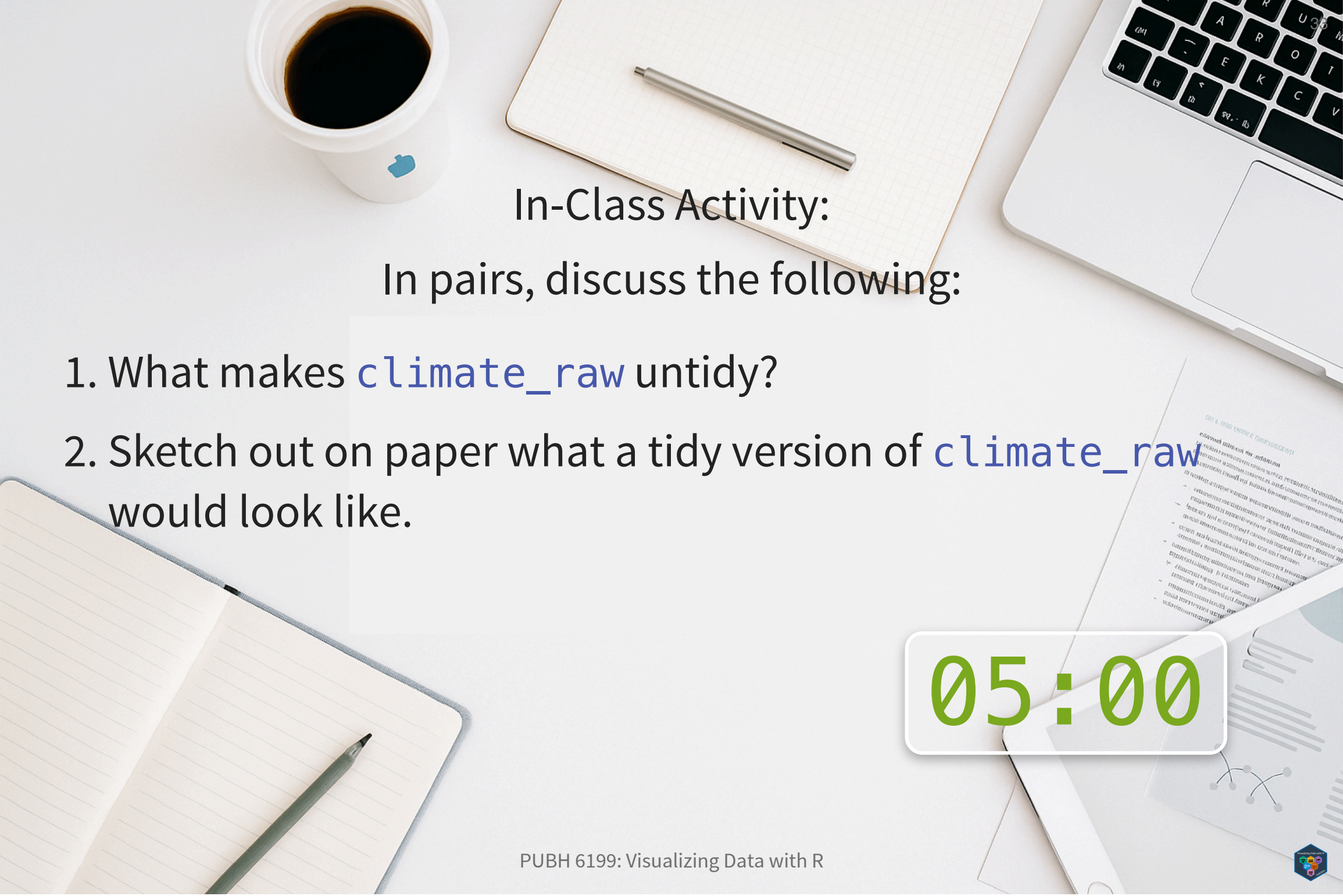


Marginal sums and statistics



A single untidy table, `climate_raw`

date	city	zone	temp_morning	temp_afternoon	humid_morning	humid_afternoon
2022-07-01	Phoenix	urban	84	108	79	34
2022-07-02	Phoenix	urban	82	107	36	50
2022-07-03	Phoenix	urban	90	108	43	17
2022-07-04	Phoenix	urban	79	97	81	34
2022-07-05	Phoenix	urban	83	95	17	55
2022-07-01	Miami	coastal	87	104	70	67
2022-07-02	Miami	coastal	78	104	71	71
2022-07-03	Miami	coastal	85	104	52	53
2022-07-04	Miami	coastal	84	108	83	78
2022-07-05	Miami	coastal	84	103	26	66



In-Class Activity:

In pairs, discuss the following:

1. What makes `climate_raw` untidy?
2. Sketch out on paper what a tidy version of `climate_raw` would look like.

05:00

Why do untidy data exist and what to do about it?

- Data is collected in a way that is convenient for the collector, not the analyst
- Most people aren't familiar with the principles of tidy data unless you are a data professional

- To tidy data:
  - Begin by figuring out what are the variables and observations
  - Talk to the data curator if needed
  - **pivot** your data into a tidy form

pivot_longer()

Suppose we have three patients with ids A, B, and C. Each patient has two blood pressure measurements: bp1 and bp2. The data is in wide format:

```
df <- tibble::tribble(
  ~id,  ~bp1, ~bp2,
   "A",  100,  120,
   "B",  140,  115,
   "C",  120,  125
)
```

We want our new dataset to have three variables: id (already exists), measurement (the column names), and value (the cell values). To achieve this, we pivot df longer:

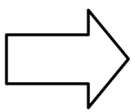
```
df |>
  tidyr::pivot_longer(
    cols = bp1:bp2,
    names_to = "measurement",
    values_to = "value"
  )
```

```
# A tibble: 6 × 3
  id    measurement value
  <chr> <chr>       <dbl>
1 A     bp1           100
2 A     bp2           120
3 B     bp1           140
4 B     bp2           115
5 C     bp1           120
6 C     bp2           125
```


How does `pivot_longer()` work?

Repeat `id` twice

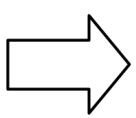
id	bp1	bp2
A	100	120
B	140	115
C	120	125



id	measurement	value
A	bp1	100
A	bp2	120
B	bp1	140
B	bp2	115
C	bp1	120
C	bp2	125

`bp1` and `bp2` become values in a new column

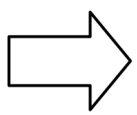
id	bp1	bp2
A	100	120
B	140	115
C	120	125



id	measurement	value
A	bp1	100
A	bp2	120
B	bp1	140
B	bp2	115
C	bp1	120
C	bp2	125

The number of values is preserved and unwound row-by-row.

id	bp1	bp2
A	100	120
B	140	115
C	120	125



id	measurement	value
A	bp1	100
A	bp2	120
B	bp1	140
B	bp2	115
C	bp1	120
C	bp2	125

pivot_wider()

Suppose we have two patients with ids A and B. We have three blood measurements on patient A and two on patient B. The data is in long format:

```
df <- tribble(
  ~id, ~measurement, ~value,
  "A",        "bp1",    100,
  "B",        "bp1",    140,
  "B",        "bp2",    115,
  "A",        "bp2",    120,
  "A",        "bp3",    105
)
```

We'll take the values from the value column and the names from the measurement column:

```
df |>
  tidyr::pivot_wider(
    names_from = measurement,
    values_from = value
  )
```

```
# A tibble: 2 × 4
  id      bp1   bp2   bp3
  <chr> <dbl> <dbl> <dbl>
1 A       100   120   105
2 B       140   115    NA
```

`pivot_wider()` can make missing values.

How does `pivot_wider()` work?

First, figure out what will be the new column names, taken from `measurement`.

```
1 library(tidyverse)
2 df |>
3   distinct(measurement) |>
4   pull()
```

```
[1] "bp1" "bp2" "bp3"
```

Then, figure out what will be the rows in the output, determined by all the variables that aren't going into the new names or values. Can be one or many.

```
1 df |>
2   select(-measurement, -value) |>
3   distinct()
```

```
# A tibble: 2 × 1
  id
  <chr>
1 A
2 B
```

`pivot_wider()` then combine the columns and rows to generate an empty data frame, then fill it with `value` in the input.

```
1 df |>
2   select(-measurement, -value) |>
3   distinct() |>
4   mutate(bp1 = NA, bp2 = NA, bp3 = NA)
```

```
# A tibble: 2 × 4
  id    bp1   bp2   bp3
  <chr> <lgl> <lgl> <lgl>
1 A     NA    NA    NA
2 B     NA    NA    NA
```

`pivot_wider()` can make missing values.

Why do we need `pivot_wider()`?

Isn't tidy data long?

- Yes — **tidy data often means long format**, especially for:
  - plotting
  - filtering
  - grouping
- But tidy ≠ always long!

Tidy = Structure

- Each variable in a column, each observation in a row
- Sometimes wide format **is** tidy — it depends on context.

When do we need `pivot_wider()`?

- ✅ For **modeling**:
  - `lm(bp1 ~ bp2)` needs one column per variable
- ✅ For **presentation**:
  - Easier to read tables with 1 row per subject
- ✅ For **joining**:
  - Merge with spatial data or metadata
- ✅ To **undo a pivot_longer()**

🖌 Let's tidy `climate_raw`

~ Head over to lab1 notebook! ~

End-of-Class Survey

Fill out the end-of-class survey

~ This is the end of Lab 1 ~

10:00