

Data Wrangling

Wrangling Tools (The Big Six)

- ☐ arrange: **arrange** the rows according to some *column*
- ☐ filter: **filter** out or obtain a subset of the *rows*
- ☐ select: **select** a subset of *columns*
- ☐ mutate: **mutate** or create a *column*
- ☐ summarize: calculate a numerical **summary** of a *column*
- ☐ group_by: **group** the rows by a specified *column*

Example: `x > select(y,z,a) > filter (b) > arrange(c, desc) > mutate`

`filter(!is.na(column_name))` - to remove any columns that include NA values

`filter(complete.cases(column_name))` - to remove any rows that include NA values in that column

Wrangling (Dates)

Key Takeaways

- ✓ Use (lubridate) functions (`year()`, `month()`, `yday()`) for easy date extraction.
- ✓ Store dates as "Date" type rather than strings.
- ✓ Compare dates directly with logical operators (`>=`, `<`).
- ✓ Convert date formats correctly using `ymd()`, `dmy()`, etc.
- ✓ Filter and aggregate data based on date components for insights.

less

Types of Reshaped Data

1. **Aggregate Data:** Summarizes observations but loses individual-level data (e.g., using `group_by()` with `summarize()`).

2. **Raw Data, Reshaped:** Retains individual observations but is structured differently for analysis.

Join Type	Keeps All Left?	Keeps All Right?	Keeps Only Matches?	Adds Columns?
left_join ()	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
inner_join	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
full_join	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
semi_join	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
anti_join	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Final Notes

Mutating joins (`left_join`, `inner_join`, `full_join`) are best for combining datasets.

Filtering joins (`semi_join`, `anti_join`) are best for selecting relevant rows.

• Always consider **which table should be the left table**—this affects the output.

- ❖ `left_join()` → Keep all students, add enrollment if available.
- ❖ `inner_join()` → Keep only students with enrollment data.
- ❖ `full_join()` → Keep everything, even if some values are missing.
- ❖ `semi_join()` → Keep students whose class has enrollment data, but don't add the extra column.
- ❖ `anti_join()` → Find students whose class has no enrollment data.

Categorical Variables: Characters and Factors

• Categorical variables (character or factor types) require special tools and considerations.

Two main considerations:

1. **Regular expressions:** Used to detect, replace, or extract patterns in character strings.

2. **Converting characters to factors:** Helps in organizing categorical data meaningfully

Regular Expressions & Character Manipulation

• Example dataset: Course sessions data with variables like `sessionID`, `dept`, `level`, `sem`, `enroll`, and `iid`.

• Replacing text (e.g., changing FA to fall, SP to spring).

• Filtering (e.g., keeping only courses taught in the fall).

• Splitting variables (e.g., extracting semester and year separately).

Converting Characters to Factors

Example 1: Default Order Issue

• Dataset: U.S. presidential election results by county.

• Created a new categorical variable, `dem_support_20`, based on Republican-Democrat vote differences.

• Problem: The variable is stored as a **character** type, causing alphabetical ordering.

Example 2: Reordering Levels using `fct_relevel`

• **Solution:** Convert `dem_support_20` into a **factor** with a meaningful order.

• Used `fct_relevel()` to define the order: "low", "medium", "high".

• This ensures that categorical data is displayed meaningfully in summaries and visualizations.

Example 3: Changing Labels using `fct_recode`

• **Problem:** Existing category labels (low, medium, high) may not be descriptive enough.

• **Solution:** Used `fct_recode()` to rename categories:

• "low" → "strong republican"

• "medium" → "close race"

• "high" → "strong democrat"

• This improves clarity while maintaining meaningful order.

Key Takeaways

1. **Factors** help store categorical data with meaningful order.

2. `fct_relevel()` reorders factor levels to avoid default alphabetical sorting.

3. `fct_recode()` renames factor levels for clarity.

4. **Regular expressions** help manipulate character data efficiently.

5. **Proper factor handling** leads to clearer visualizations and analysis.