dplyr

Tidying and manipulating data

Download the section 5 .Rmd handout to `STAT240/lecture/sect05-dplyr`.

Download three data files: `grocery-list.csv`, `grocery-prices.csv`, `madison-weather-official-1869-2023.csv` to `STAT240/data`.

Install `magrittr`.

Material in this section is covered by Chapters 7-8 on the notes website.

`dplyr` is a collection of simple commands for performing powerful data tidying tasks.

- Most data does not start in a convenient form
- Grammar for data manipulation

Here is a cheatsheet of `dplyr` commands.

The **pipe** operator %>% takes an object on the left, and passes it into the command on the right.

- Always passes as the *first* element
- Main use case: nested functions
- Works exactly like the word "then"

Let's demonstrate with some arithmetic functions.

Basic dplyr command:

- First argument is a df
- Then say what we want to do to the df
- Output is the new df

Commands can be chained together with the pipe.

Column commands act on the columns.

- `mutate` for a new column
- `select` subsets the columns
- `relocate` move columns around
- `rename` changes column names

Rows are left unchanged.

Row commands act on the rows.

- `arrange` to order the rows
- `filter` subsets rows by a condition
- `drop_na` removes rows with missing values

Columns are left unchanged.

An important skill is to translate ordinary language into `dplyr` verbs.

Write your own code to respond to five English requests.

`summarize` reduces a column to a single value.

- For example: `sum`, `mean`
- These functions go into `summarize`
- Much of the information is lost

`summarize` is particularly powerful when combined with `group_by`.

`group_by` is a special command that sets the "grouping" property of the dataframe.

- Does not change the values
- Gives instructions for future commands

Let's find the average price of fruits and vegetables.

By using `group_by`, we can do fewer total operations.

Reducing functions in `summarize` are given separately to each group.

Useful functions to use with `group_by`:

- `slice_min` to get the lowest values
- `slice_max` to get the highest values
- `n` to get the number of rows

We can also use `count` as a shortcut for `n`.

`group_by` has useful applications to `mutate`.

- We can `mutate` the result of a function
- `group_by` first to reduce based on group

We can do sophisticated operations. Which fruits and vegetables make up the highest % cost?

We can remove the grouping instruction with `ungroup`.

This is useful if you want to perform a grouping operation at first, then go back to looking at the entire df at once.

Also lets us group by multiple grouping schemes!

Let's see how it works. Each code chunk has `group_by` commented out.

- Run the chunk as-is
- Predict what happens if you include `group_by`
- Uncomment the line and see what happens

The presence of `group_by` can drastically change the output of your workflow.

We've learned how to manipulate columns with basic tools and functions.

Another thing we might want to do is create a column based on a **condition**.

The value of each row depends on other values found in the df.

The command `case_when` is used to define a column in terms of the values in other columns.

- Use ~ to assign values
- Can be used within `mutate`

Let's look at a few examples with different levels of complexity.