Data Manipulation in R with *dplyr*

Sean Hellingman ©

Data Visualization and Manipulation through Scripting (ADSC1010)

shellingman@tru.ca

Fall 2024

THOMPSON RIVERS UNIVERSITY

Topics

2. Introduction
3. The select() Function
4. The filter() Function
5. The mutate() Function
6. The arrange() Function
7. The summarize() Function
8. Sequential Operations
9. The group_by() Function

Introduction

- The *dplyr* package is the primary data wrangling tool in R.
- Intended to have intuitive vocabulary.
- Once mastered, it will make your data processing much faster.

dplyr Functions (Verbs)

- The package is written in a way that you should be able to use specific functions that match your task description:
  - **Select** specific columns (features) from a dataset.
  - **Filter** out any unwanted observations (rows) from your dataset.
  - **Mutate** a dataset by adding more columns (features).
  - **Arrange** observations (rows) in a particular order.
  - **Summarize** data in aggregates such as *mean*, *median*, or *minimum*.
  - **Join** multiple datasets together into a single data frame.

Package Information

- The *dplyr* package is part of the *tidyverse* collection of R packages.
- We need to install and load the package prior to use:

  -
    ```
    if(!require(dplyr)) install.packages("dplyr")
    library(dplyr)
    ```
- Once the package is loaded we can use the existing functions as we please.

select()

- The `select()` function takes the data frame and which columns you want to select as arguments.
  - `select(df.name, variable1.name, variable2.name,...)`
  - `select(df.name, - variable1.name, - variable2.name,...)` *selects all columns apart from those indicated with -*
- Less involved than using base R.
- Returns a data frame even if you only ask for one column.

Example 1

- Import the the *starwars* data from the *dplyr* package.
- Use the `select()` function to create a data frame with the *name*, *height*, and *homeworld* of characters from the *starwars* data.
- Use the `select()` function to create a data frame **without** the *vehicles*, and *starships* of characters from the *starwars* data.

`filter()`

- The `filter()` function takes the data frame and allows you to extract specific rows.
  - `filter(df.name, variable1.name == Value)`
  - `filter(df.name, variable2.name <= Value)`
- Takes a list of conditions and returns a data frame.

Example 2

- Use the `filter()` function to create a data frame of all the humans from the *species* variable in the *starwars* data from the *dplyr* package.

mutate()

- The `mutate()` function takes the data frame and allows you to create new columns.
    - `mutate(df.name, New.variable.name = Value)`
    - `mutate(df.name, New.variable.name = variable1.name + variable2.name)`
- Takes a list of new columns to create and returns a data frame.
- Note: the function does not actually add the new columns. Instead you need to replace your old data frame with the *new* one.

Example 3

- Use the `mutate()` function to add the Body Mass Index (BMI) of the characters to the *starwars* data from the *dplyr* package.
- BMI: kg/m^2

`arrange()`

- The `arrange()` function takes the data frame and allows you to sort the rows of your data by some column.
  - `arrange(df.name, -variable2.name)` decreasing order
  - `arrange(df.name, variable1.name)` increasing order
- Takes a list of columns to arrange by and returns a data frame.
- The order of your list of columns matters.
- Note: you again need to replace your old data frame with the *new* one.

Example 4

- Use the `arrange()` function to arrange the characters from the *starwars* data from the *dplyr* package by *height* in decreasing order.

summarize()

- The `summarize()`/`summarise()` function acts as an aggregation operation for one or more columns from your data frame.
- It reduces the entire column into a single value.
  - `summarize(df.name, Var.2_mean = mean(variable2.name))` mean of *variable2.name*
  - `summarize(df.name, Var.1_max = max(variable1.name))` maximum value of *variable1.name*
- Takes a list of arguments and returns a data frame with one row.
- You can also write your own summary functions and use them within the `summarize()` function.
- Not usually ideal to replace your existing data with these results.

Example 5

- Use the `summarize()` function to obtain the maximum, minimum, average, and median BMI of the characters from the *starwars* data from the *dplyr* package by *height* in decreasing order.
- You will need to omit the missing observations.

Sequential Operations

- In order to conduct more complex analysis you may need to combine some of these functions.
- One approach is to create *intermediary* variables and pass them from one function to another.
- A more efficient approach is to use *anonymous* variables and nest the statements within other functions.

Example 6

- Use the functions in the *dplyr* package to determine the maximum BMI of the humans with blue eyes in the *starwars* data.
    - Use *intermediary* variables.
    - Use *anonymous* variables and nest the statements.

The Pipe Operator %>%

- Built in *dplyr* functionality that makes passing results of functions easier.
- Formally, the pipe operator %>% takes the result from one function and passes as the first argument to the next function.
- Works well as all of the *dplyr* functions take a data frame as the first argument.
- In R Studio: ctrl+shift+m results in %>%

Example 7

- Use the pipe operator %>% and functions in the *dplyr* package to determine the maximum BMI of the humans with blue eyes in the *starwars* data.
- Which approach seemed easiest?

`group_by()`

- The `summarize()` function works on all observations in a column.

- The `group_by()` function allows you to group data with the same variable value together.

- Returns a **tibble** which is a special version of a data frame used primarily in the *tidyverse* packages.

- Functions (*verbs*) applied directly to the **tibble** will be applied to each group separately.

Example 7

- Use the `group_by()` function to summarize *height* and *mass* by *eye_color* groups in the *starwars* data.

Data in Multiple Files

- Data is often stored in multiple multiple locations.
- Very typical of relational databases.
- Often more efficient to store and update data in this format.
- We will need to **join** data frames together to work with them.

Joining Data Frames

- Very similar approach to SQL.
- Use columns that are present in both data frames to *match* corresponding rows together.
- Columns used in the matching process are called **identifiers** (keys).
- The identifiers are combined in one row when the data frames are joined.

`left_join()`

- Looks for matching columns between two data frames.
- Returns a new data frame that is the first (*left*) argument with extra columns from the second (”right”) added on.
- The resulting table is a *merged* table of the two arguments.
- Matching occurs using the `by` argument which takes a vector of column names (strings).
- Left rows without a match will have `NA` in the right columns.

right_join()

- Looks for matching columns between two data frames.
- Returns a new data frame that is generated in the opposite direction of the `left_join()` function.
- In other words, simply reversing the arguments from the `left_join()` function.
- *You really only need to remember how to use* `left_join()` *OR* `right_join()`.

`inner_join()`

- Only rows present in *both* data frames are returned.
- Returns a new data frame that contains only observations that had matches in both data frames.
- Observations without matches will not be included (no `NA` values).
- The order of the arguments does not matter.

full_join()

- All the rows present in *both* data frames are returned.
- A row for every single observation is returned.
- Observations without matches will have `NA` values in the columns from the other data frame.
- Can lead to very messy data.
- The order of the arguments does not matter.

Joins

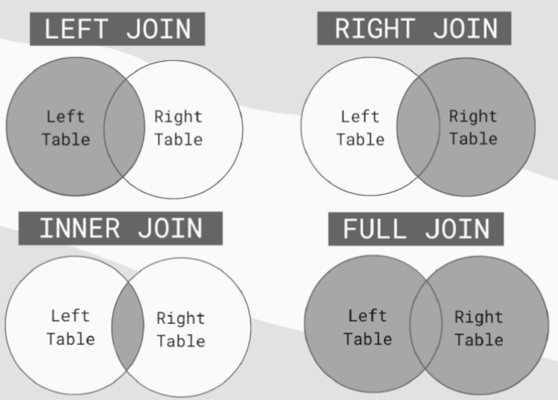

Figure: *source:* (4)

Example 7

- Import the `Clients.csv` and `Purchases.csv` data into your workspace.
- Take some time to look through the data.
- Practice joining the data using the functions in the *dplyr* package.

Exercise 1

- Import the *mpg* dataset from the *ggplot2* package.
- Take some time to get to know the data.
- Using the functionality of the *dplyr* package conduct the following analysis:
    - Identify the average highway miles per gallon (*hwy*) of ford vehicles newer than 1999.
    - Find the average city (*cty*) and highway miles per gallon (*hwy*) for each car *class*.
    - Create a data frame ordering the highway miles per gallon (*hwy*) from greatest to smallest for each *manufacturer*.

Exercise 2

- Create a data frame (in R or Excel) with overlapping client IDs from Exercise 7.
- In the new data frame (randomly) include:
  - The clients' monthly incomes.
  - The clients' preferred payment method (debit or credit).
- Join your new data to the existing data and utilise the functionality in the *dplyr* package to appropriately summarise the information contained in the clients list.

References & Resources

1. Douglas, A., Roos, D., Mancini, F., Couto, A., & Lusseau, D. (2023). *An introduction to R*. Retrieved from https://intro2r.com/
2. Michael Freeman, Joel Ross, *Programming Skills for Data Science: Start Writing Code to Wrangle, Analyze, and Visualize Data with R*, 2019, ISBN-13: 978-0-13-513310-1
3. Baumer, Kaplan, Horton. *Modern Data Science with R.* Retrieved from https://mdsr-book.github.io/mdsr3e/.
4. https://learnsql.com/blog/learn-and-practice-sql-joins/

- https://www.tidyverse.org/
- https://tibble.tidyverse.org/
- https://statisticsglobe.com/r-dplyr-join-inner-left-right-full-semi-anti