Lecture 12: Wrangling Data with `dplyr` for single data frames

UNIVERSITY OF
SAN FRANCISCO

Abbie M. Popa

BSDS 100 - Intro to Data Science with R

Table of Contents

- New Features from `dplyr`
- Doing old things a new way with `dplyr`

New Features from `dplyr`

`dplyr::select()` column selection

- Much like traditional subsetting, `dplyr` can be used to output individual columns
- First, import the library with `library(dplyr)`
- Then use `select(data frame, columns)`
- e.g., `select(mtcars, mpg, cyl)`

`dplyr::select()` provides more control

- `select(iris, contains("."))`
  select columns whose name contain a particular string
- `select(iris, ends_with("Length"))`
  select columns whose name ends with a particular string
- `select(iris, num_range("x", 1:5))`
  select columns whose named x1, x2, x3, x4, x5
- `select(iris, one_of(c("Species", "Genus")))`
  select columns whose names are in a group of possible names
- `select(iris, starts_with("Sepal"))`
  select columns whose name starts with a particular string
- `select(iris, "Sepal.Length":"Petal.Width")`
  select columns between two named columns (inclusive)
- `select(iris, -Species)`
  select columns except those with the minus

Summarizing data using `dplyr`

The main `dplyr` functions used to summarize data are:

- `summarise()`, which summarizes data into a single row of values
- `summarise_all()`, which applies multiple summary functions to each column

`summarise()` examples

- Load the `ggplot2` library to gain access to the msleep data set
- `summarise(msleep, avg_sleep = mean(sleep_total))` outputs a tibble with one column, "avg_sleep" reporting the mean of "sleep_total" from the msleep dataset
- `summarise(msleep, avg_sleep = mean(sleep_total), med_sleep = median(sleep_total))` outputs a tibble with two columns, one for the mean total sleep and one for the median total sleep
- `summarise(msleep, avg_sleep = mean(sleep_total), avg_rem = mean(sleep_rem, na.rm = T))` outputs a tibble with two columns, on for mean total sleep and one for mean REM sleep

summarise_all()

- Applies the same function(s) to all columns of data
- `summarise_all(iris, funs(mean))` reports the mean of each column of the iris data set, in a tibble
- You can also pass arguments like `na.rm = T` using the `.` placeholder

  `summarise_all(msleep, funs(mean(., na.rm = T)))`
- You can also pass more than one function, e.g.,

  `summarise_all(select(iris, -Species), funs(mean, median, sd))`

  note also how we removed the "Species" column since it is not numeric!

group_by()

- The real power of summarise() and summarise_all() comes when we add a group_by() clause
- This makes one row for each of the groups
- Example with summarise()

  ```
  summarise(group_by(msleep, vore), mean_sleep =
  mean(sleep_total))
  ```

  gives the mean sleep total for each of the groups in vore from msleep
- Example with summarise_all()

  ```
  summarise_all(group_by(iris, Species),
  funs(mean))
  ```

  gives the mean of all columns for each species in the iris data set

Using `group_by()` for plotting

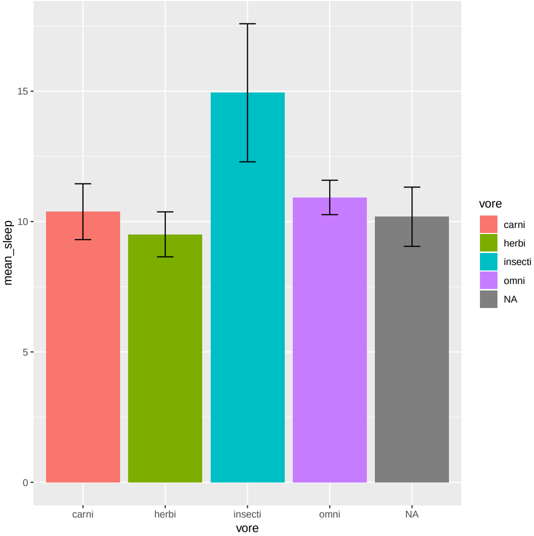
- We wish to make a boxplot with bars for each of the animal group's mean sleep times with standard deviation error bars
- First, make a useful table with `summarise()`:

  ```
  sum_sleep <- summarise(group_by(msleep, vore), mean_sleep =
  mean(sleep_total), se_sleep =
  sd(sleep_total)/sqrt(length(sleep_total)))
  ```

- Then, we use our new table to make a plot:

  ```
  ggplot(sum_sleep, aes(x = vore, y = mean_sleep, fill = vore)) +
  geom_bar(stat = "identity") + geom_errorbar(aes(ymin = mean_sleep
  - se_sleep, ymax = mean_sleep + se_sleep), width = 0.2)
  ```

The resulting plot:



`count()`

- The function `count()` counts the number of rows containing each unique value of a variable
- For example, `(count(msleep, vore)` provides a count for each unique value of vore
- You can also combine this with `interaction()` to provide a count of each unique combination of 2 or more variables

  e.g., `count(mtcars, interaction(cyl, vs))`

`count()` versus `summarise(group_by)`

- When you use `summarise` with a `group_by` clause it also breaks up your rows by unique values of a variable, but it then applies whichever function you choose to those groups
- `count()` breaks the column down into unique values, but the only "function" it performs is a count of how many observations (rows) have each value

Practice

- Find how many diamonds there are for each cut by filling in the blanks:

  `count(_____, _____)`

  (hint, one blank is the name of the dataset, and the other is a name of a column in that data set, if you don't remember the names of the columns, try `colnames(diamonds)` or `str(diamonds)`)

- Find the mean price for each cut of diamond by filling in the blanks:

  `summarise(group_by(_____, _____), mean_price = _____(_____))`

  (hint, the first blank is a dataset, the second is a column, the third is a function, and the fourth is a column!)

Doing old things a new way with `dplyr`

Old functions in `dplyr`

- `dplyr` provides syntax for a lot of our traditional data frame operations
- Examples include conditional subsetting and adding new columns
- While you don't need to do these functions with `dplyr` some of you may find you prefer the `dplyr` syntax
- The `dplyr` syntax also plays nicely with pipes (`%>%`) from `magrittr`

slice()

- slice() is a way to select rows from a data frame
- slice(iris, 3:5) is equivalent to iris[3:5,]

filter()

- `filter()` is a way to conditional subset
- `filter(mtcars, mpg > 32)` is equivalen to `mtcars[mtcars$mpg > 32, ]`
- As with traditional conditional subsetting, you can have multiple conditions. With `filter()` you can separate these by commas e.g., `filter(mtcars, mpg > 30, mpg < 33, wt <2)`
- To use the "or" operator you must still use the |, e.g., `filter(mtcars, mpg < 30 | mpg > 33, wt < 2)`

mutate()

- mutate() can be used to add a column to the data frame
- first, make a new copy of the data frame we can modify with

  ```
  my_msleep <- msleep
  ```

-
  ```
  my_msleep <- mutate(msleep, rem_per_sleep = sleep_rem/sleep_total
  ```

  is equivalent to

  ```
  my_msleep$rem_per_sleep <- my_msleep$sleep_rem/my_msleep$sleep_total
  ```

End of Class

- Find the data wrangling lab on the github page https://www.github.com/abbiepopa/bsds100
- Complete and upload to canvas for today's participation
- You can complete together, but everyone must upload something