3_data_manip_tidyverse

December 21, 2024

1 Data Manipulation with Tidyverse, Part I

In this lecture and the next, we're going to go through a lot of different concepts. Remember, you can use the Complete Notebook as a reference fo all that we will cover in this lecture.

The goal of this lecture and the next is:

1) Learn about packages, the tidyverse
2) Learn how to manipulate and clean data with tidyverse

2 Why is data manipulation important

Presumably, we are all planning to do Environmental Data Science

- We will have a hypothesis of how the world works
- We will want to construct a model that approximates that
- We will need data from real world to build that model that approximates the world
- The data we have may not be set up to be plugged into the model we'd like to run
- However, it could be *manipulated* so that we can use the data we have with the model we want

My personal example: American Time Use Survey and Travel Cost Model, Cell Photo Data and green space

3 What are packages and how can I get them?

What are they:

- A package contains a bunch of pre-built functions
- Anyone can load and use them
- Saves you a ton of time because someone already figured out how to do it

Tidyverse

- Collection of R packages
- All are meant for data science
- Have shared syntax
- Makes it easier to import, tidy, transform, visualize, and model data in R
- Shout out to Hadley Wickham and co

Installing a package vs. loading a pacakge

- You only need to install a package on your local computer once (your lab has all the packages pre-installed)
- You then "load" that package in a script when you want to use it using the `library()` command

For reference, the code to install a package on you local computer is below. We do not need to run this code on each of our local machines.

```
# installing packages that are a part of the tidyverse using r code
install.packages("dplyr")
install.packages("tidyr")
install.packages("ggplot2")
```

You can load packages after you've installed them with the library function

[1]:
```
# load dplyr, the package we will focus on today
library(dplyr)
```

```
Attaching package: 'dplyr'


The following objects are masked from 'package:stats':

    filter, lag


The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union


The following objects are masked from 'package:stats':

    filter, lag


The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

4 Manipulating and cleaning with dplyr

- dplyr is my most used package for data cleaning and manipulation
- Going to introduce the primary functions in the dplyr package
  - `mutate()`
  - `if_else()`

  - `filter()`
  - `select()`
  - `group_by()`
  - `summarise()`
- There are a million ways to implement these functions
- Important for your solution strategy to know that these are the functions you can build around
- I would say 90% of my data cleaning is different combinations of these functions
- Why use dplyr instead of base R?
  - It's faster and more memory efficient (good for large data sets)
  - It's easier to read
- Let's go through some examples of what we did in the Base R Lectures , but redo with dplyr code

5 `mutate()`

Last lecture, everyone moved 1 mile futher from the park (bummer). We used a loop, like this

[2]:
```
# Build our trusty dataset, but let's call this one myBase
myBase <- data.frame(
   name = c("Andie", "Bridger", "Scott"),
   gender = c("Female", "non-binary", "Male"),
   male = c(FALSE, FALSE, TRUE),
   income_cat = c("middle", "poor", "rich"),
   park_dist = c(1, 0.5, 0.1)
)
myBase

# create a second dataset that is identical to the first
myDplyr <- myBase

# check to make sure they're the same
identical(myBase, myDplyr)
```

A data.frame: 3 x 5

name <chr>	gender <chr>	male <lgl>	income_cat <chr>	park_dist <dbl>
Andie	Female	FALSE	middle	1.0
Bridger	non-binary	FALSE	poor	0.5
Scott	Male	TRUE	rich	0.1

TRUE

[3]:
```
# Base R
for (i in seq_along(myBase$park_dist)) {
  myBase$park_dist[i] <- myBase$park_dist[i] + 1 # everyone aged one year
}

myBase
```

	name	gender	male	income_cat	park_dist
	<chr>	<chr>	<lgl>	<chr>	<dbl>
A data.frame: 3 x 5	Andie	Female	FALSE	middle	2.0
	Bridger	non-binary	FALSE	poor	1.5
	Scott	Male	TRUE	rich	1.1

Let's perform the same command with the `mutate()` function from the `dplyr` package.

[4]:
```
# add mile to park dist using dplyr
myDplyr <- myDplyr %>%
  mutate(park_dist = park_dist + 1)

myDplyr

# use a boolean operator to show the two dfs are the same
identical(myBase, myDplyr) # base R command
```

	name	gender	male	income_cat	park_dist
	<chr>	<chr>	<lgl>	<chr>	<dbl>
A data.frame: 3 x 5	Andie	Female	FALSE	middle	2.0
	Bridger	non-binary	FALSE	poor	1.5
	Scott	Male	TRUE	rich	1.1

TRUE

New things introduced

- Pipes %>%: pipes take input (our dataframe) and passes it onto the next function. You can chain pipes together.
- `mutate()` mutate is a function from the dplyr package
- A data frame is piped to the function `mutate()` and then the function executes some small operation you gave it (park_dist +1) and returns the new value

[5]:
```
# create a new column with mutate (rather than just changing original)
myDplyr <- myDplyr %>%
  mutate(park_dist_new = park_dist + 1)

  myDplyr
```

	name	gender	male	income_cat	park_dist	park_dist_new
	<chr>	<chr>	<lgl>	<chr>	<dbl>	<dbl>
A data.frame: 3 x 6	Andie	Female	FALSE	middle	2.0	3.0
	Bridger	non-binary	FALSE	poor	1.5	2.5
	Scott	Male	TRUE	rich	1.1	2.1

New things introduced

- `mutate()` can also construct a new a new column

6 if_else()

Last lecture, we said that women and non-binary people had their distances from parks recorded wrong. Non-male people were a quarter mile closer to the park than orginally recorded.

[6]:
```
# goes through each row and changes distance if someone is not male
for (i in seq_along(myBase$male)) {
    if (myBase$male[i] == FALSE) { # check if someone is "not male"
        myBase$park_dist_correct[i] <- myBase$park_dist[i] - 0.25 # adjust
    }else{
        myBase$park_dist_correct[i] <- myBase$park_dist[i]
    }
}

myBase
```

A data.frame: 3 x 6

name <chr>	gender <chr>	male <lgl>	income_cat <chr>	park_dist <dbl>	park_dist_correct <dbl>
Andie	Female	FALSE	middle	2.0	1.75
Bridger	non-binary	FALSE	poor	1.5	1.25
Scott	Male	TRUE	rich	1.1	1.10

`dplyr` has an `if_else()` function

[7]:
```
# correct distance using if_else
myDplyr <- myDplyr %>%
  mutate(park_dist_correct = if_else(male == FALSE, park_dist - 0.25,␣
 ↪park_dist))

myDplyr
```

A data.frame: 3 x 7

name <chr>	gender <chr>	male <lgl>	income_cat <chr>	park_dist <dbl>	park_dist_new <dbl>	park_dist_cor <dbl>
Andie	Female	FALSE	middle	2.0	3.0	1.75
Bridger	non-binary	FALSE	poor	1.5	2.5	1.25
Scott	Male	TRUE	rich	1.1	2.1	1.10

New things introduced

- `if_else()` combined with `mutate()`
  - the first part is what you're evaluating to check if it's true (`male == FALSE`)
  - Next entry is what to return if true (`park_dist - 0.25`)
  - Final part is what to return if false (`park_dist`)

7 filter()

We haven't seen something like a filter command yet. The filter command is used when you have a data set, and want subset to only some observations conditioned on something.

7.1 Example One: Filter for a characteristicc

Let's say we have observations of different ecosystems

[8]:
```
# ecosystem dataset
df_env_data <- data.frame(
    ecosystem = c("Forest", "Desert", "Wetland", "Grassland", "Urban"),
    species_richness = c(120, 45, 80, 60, 30),
    pollution_level = c("Low", "High", "Medium", "Low", "High")
)

# display
df_env_data

# filter to include only locations with low pollution levels
low_pollution_data <- df_env_data %>%
    filter(pollution_level == "Low")

# display the filtered dataset
low_pollution_data
```

A data.frame: 5 x 3

ecosystem <chr>	species_richness <dbl>	pollution_level <chr>
Forest	120	Low
Desert	45	High
Wetland	80	Medium
Grassland	60	Low
Urban	30	High

A data.frame: 2 x 3

ecosystem <chr>	species_richness <dbl>	pollution_level <chr>
Forest	120	Low
Grassland	60	Low

7.2 Example two: drop NAs

You will frequently have datasets that have missing data (*i.e.,* the cell has a `NA` value). Many functions and models won't work with NAs, so they have to be cleaned out of the data set.

Let's imagine we're missing a location for one of our ecosystem observations

[9]:
```
# ecosystem dataset but with NAs
df_env_data_na <- data.frame(
    ecosystem = c("Forest", "Desert", "Wetland", NA, "Urban"),
    species_richness = c(120, 45, 80, 60, 30),
    pollution_level = c("Low", "High", "Medium", "Low", "High")
)

# display the dataset with NAs
df_env_data_na
```

```
# drop rows with NAs in the 'location' column using filter
df_env_data_clean <- df_env_data_na %>%
    filter(!is.na(ecosystem))

# display the cleaned dataset
df_env_data_clean
```

A data.frame: 5 x 3

ecosystem <chr>	species_richness <dbl>	pollution_level <chr>
Forest	120	Low
Desert	45	High
Wetland	80	Medium
NA	60	Low
Urban	30	High

A data.frame: 4 x 3

ecosystem <chr>	species_richness <dbl>	pollution_level <chr>
Forest	120	Low
Desert	45	High
Wetland	80	Medium
Urban	30	High

8 select()

You have a data set with more columns than you need. Sometimes you want to only work with some of the variables, and it is space efficient to get rid of the rest.

For example, let's say you're only interested in the polution at diffent locations, not the species richness.

[10]:
```
# select only the ecosystem and pollution_level columns
pollution_data <- df_env_data %>%
    select(ecosystem, pollution_level)

# display the selected columns
pollution_data
```

A data.frame: 5 x 2

ecosystem <chr>	pollution_level <chr>
Forest	Low
Desert	High
Wetland	Medium
Grassland	Low
Urban	High

9 group_by()

Group by is useful for when you want to aggregate information up to a higher level. So, if you need a varaibel that is the average species richness by ecosystem rather than the species richness at individual sites.

Let's work with a longer version of our ecosystem dataset

[11]:
```
# create an environmental dataset (don't clear this!)
df_env_long <- data.frame(
    ecosystem = c("Forest", "Desert", "Wetland", "Grassland", "Urban",␣
 ↪"Forest", "Desert", "Wetland", "Grassland", "Urban"),
    species_richness = c(120, 45, 80, 60, 30, 110, 50, 85, 65, 35),
    pollution_level = c("Low", "High", "Medium", "Low", "High", "Low", "High",␣
 ↪"Medium", "Low", "High")
)

# display the dataset
df_env_long

# group by ecosystem and calculate the mean species richness
df_env_grouped <- df_env_long %>%
    group_by(ecosystem) %>%
    mutate(mean_species_richness = mean(species_richness))

# display the updated dataset with mean species richness
df_env_grouped
```

A data.frame: 10 x 3

ecosystem <chr>	species_richness <dbl>	pollution_level <chr>
Forest	120	Low
Desert	45	High
Wetland	80	Medium
Grassland	60	Low
Urban	30	High
Forest	110	Low
Desert	50	High
Wetland	85	Medium
Grassland	65	Low
Urban	35	High

A grouped_df: 10 x 4

ecosystem <chr>	species_richness <dbl>	pollution_level <chr>	mean_species_richness <dbl>
Forest	120	Low	115.0
Desert	45	High	47.5
Wetland	80	Medium	82.5
Grassland	60	Low	62.5
Urban	30	High	32.5
Forest	110	Low	115.0
Desert	50	High	47.5
Wetland	85	Medium	82.5
Grassland	65	Low	62.5
Urban	35	High	32.5

10 `summarise()`

Finally, the `summarise()` function can help you create summary tables. These are useful for getting a quick snapshot of data. Using them is particularly important when you have a large enough datset that you're unable to look at the dataset and glean insight (which, for me, happens if the dataset is longer than 5 observations).

[12]:
```
# group by ecosystem and calculate the mean and total species richness
summary_table <- df_env_long %>%
    group_by(ecosystem) %>%
    summarise(
        mean_species_richness = mean(species_richness),
        total_species_richness = sum(species_richness)
    )

# display the summary table
summary_table
```

A tibble: 5 x 3

ecosystem <chr>	mean_species_richness <dbl>	total_species_richness <dbl>
Desert	47.5	95
Forest	115.0	230
Grassland	62.5	125
Urban	32.5	65
Wetland	82.5	165