R Workshop 5: Basic Data Wrangling

Teal Potter

10/22/2021

Table of Contents

By 'data wrangling', I mean keeping data tidy and easy to use (reformat, visualize, model). In my mind, this concept is natural to introduce with ways for looking at key information about your data in R so you can then do actions to make/keep your data tidy as you work with it.

We'll be working with the Soils dataset yet again. Many of the functions used in this tutorial are base R functions but there will also be use of tidyverse functions. Tidyverse largely has been designed to make data wrangling and graphing easier.

Setup

```
library(tidyverse)
library(car)
library(knitr) # using knitr::kable in this tutorial to make cleaner views of data
```

```
kable(head(Soils, 4))
```

Group	Contour	Depth	Gp	Block	pH	N	Dens	P	Ca	Mg	K	Na	Conduc
1	Top	0-10	T0	1	5.40	0.188	0.92	215	16.35	7.65	0.72	1.14	1.09
1	Top	0-10	T0	2	5.65	0.165	1.04	208	12.25	5.15	0.71	0.94	1.35
1	Top	0-10	T0	3	5.14	0.260	0.95	300	13.02	5.68	0.68	0.60	1.41
1	Top	0-10	T0	4	5.14	0.169	1.10	248	11.92	7.88	1.09	1.01	1.64

Wide vs Long data frames

The soils dataset is in the standard wide format where each sample is represented by just 1 row of data. See how the data is made longer in the first step below.

`pivot_longer()` and `pivot_wider()` are tidyverse's newest and best functions for transitioning your data between long and wide formats.

Wide to long

```
long <- Soils %>%
  pivot_longer(cols = pH:Conduc, names_to = 'nutrient', values_to = 'value')

kable(head(long))
```

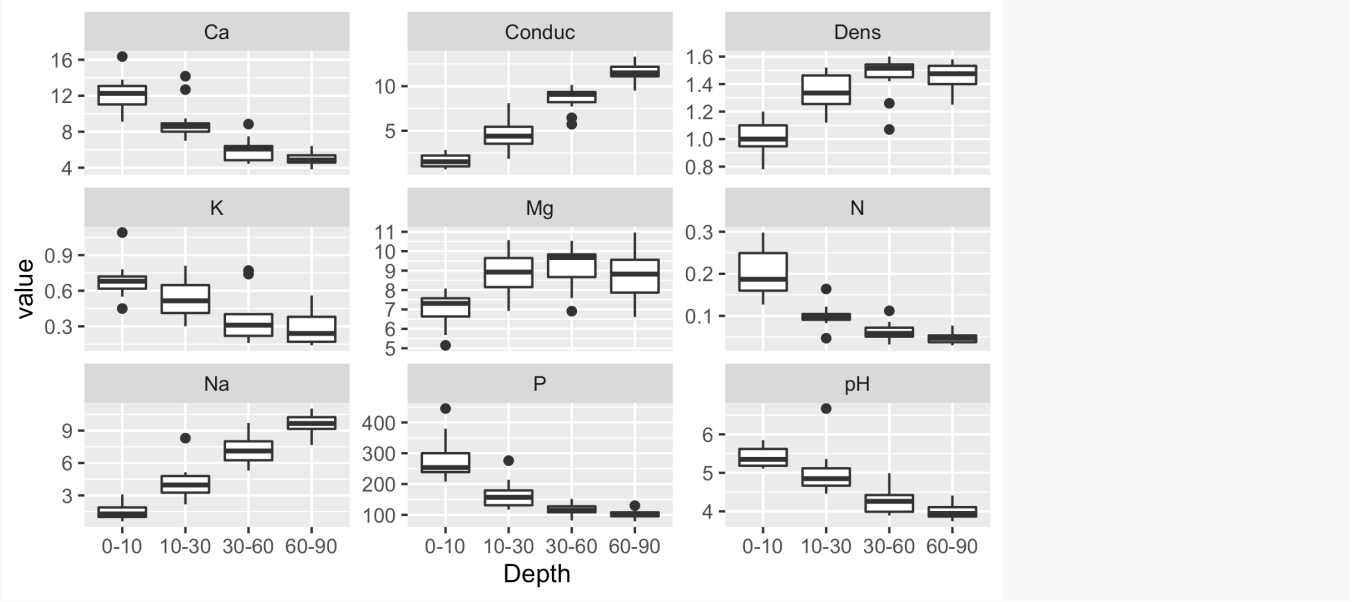
Group	Contour	Depth	Gp	Block	nutrient	value
1	Top	0-10	T0	1	pH	5.400
1	Top	0-10	T0	1	N	0.188
1	Top	0-10	T0	1	Dens	0.920
1	Top	0-10	T0	1	P	215.000
1	Top	0-10	T0	1	Ca	16.350
1	Top	0-10	T0	1	Mg	7.650

Example graph using long formatted data

With long data you can choose your x and y to plot and then divide data into panels based on a column that contains group info like the new `nutrient` column. This method is called facetting.

```
ggplot(long, aes(x = Depth, y = value))+
  geom_boxplot()+
  facet_wrap(~nutrient, scales = 'free_y')  # scales = 'free' makes a unique

#x and y axis range for each panel based on the data included in each panel
```



Long to wide

In this example I reverse the previous action and separate the nutrient column into unique columns and place the values in them.

```
wide <- long %>%
pivot_wider(id_cols = Group:Block, names_from = nutrient, values_from = value)

kable(head(wide))
```

Group	Contour	Depth	Gp	Block	pH	N	Dens	P	Ca	Mg	K	Na	Conduc
1	Top	0-10	T0	1	5.40	0.188	0.92	215	16.35	7.65	0.72	1.14	1.09
1	Top	0-10	T0	2	5.65	0.165	1.04	208	12.25	5.15	0.71	0.94	1.35
1	Top	0-10	T0	3	5.14	0.260	0.95	300	13.02	5.68	0.68	0.60	1.41
1	Top	0-10	T0	4	5.14	0.169	1.10	248	11.92	7.88	1.09	1.01	1.64
2	Top	10-30	T1	1	5.14	0.164	1.12	174	14.17	8.12	0.70	2.17	1.85
2	Top	10-30	T1	2	5.10	0.094	1.22	129	8.55	6.92	0.81	2.67	3.18

As demonstrated here, it is often the case that longer datasets are good for copmlex graphs and some statistical procedures, whereas wide format is commonly used for simple graphs and some statistics.

Subsetting data frames

Selecting columns to keep or remove

Option 1: Indexing method

Remember when using square brackets, the first arugement inside the brakets refers to rows and the second argument refers to columns

```
# first assigning Soils to become a new object, soils, so it can be altered
soils <- Soils

names(soils) # the column names in the data frame

##  [1] "Group"   "Contour" "Depth"   "Gp"      "Block"   "pH"      "N"
##  [8] "Dens"    "P"       "Ca"      "Mg"      "K"       "Na"      "Conduc"

ncol(soils) # number of columns in data frame

## [1] 14

dat <- soils[, 6:length(soils)] # keeps columns through the last column
#or
dat <- soils[, c(3, 6:14)] # keeps column 3 and column 6 through the last column

kable(head(dat))
```

Depth	pH	N	Dens	P	Ca	Mg	K	Na	Conduc
0-10	5.40	0.188	0.92	215	16.35	7.65	0.72	1.14	1.09
0-10	5.65	0.165	1.04	208	12.25	5.15	0.71	0.94	1.35
0-10	5.14	0.260	0.95	300	13.02	5.68	0.68	0.60	1.41
0-10	5.14	0.169	1.10	248	11.92	7.88	1.09	1.01	1.64
10-30	5.14	0.164	1.12	174	14.17	8.12	0.70	2.17	1.85
10-30	5.10	0.094	1.22	129	8.55	6.92	0.81	2.67	3.18

Option 2: Tidyverse::select method

`select()` allows you to list all the columns you want to keep or place a minus/hypen in front of column names remove columns by name. Note that `select()` and a set of related functions offer more arguments and functionality to specify criteria you want to use to determine which columns to keep.

```
meta <- select(soils, Group, Block, Gp)
meta <- select(meta, -Group)
```

```
kable(head(meta))
```

Block	Gp
1	T0
2	T0
3	T0
4	T0
1	T1
2	T1

Selecting rows to keep or remove

You can filter or keep rows by specifiying criteria in a column of your dataset using *logical operators* like == (equivalent), != (not equivalent), <= (less than or equal to), etc. See the three examples below.

```
kable(head(filter(soils, Depth != "10-30"))) # keep rows where depth is not 10-30 cm
```

	Group	Contour	Depth	Gp	Block	pH	N	Dens	P	Ca	Mg	K	Na	Conduc
1	1	Top	0-10	T0	1	5.40	0.188	0.92	215	16.35	7.65	0.72	1.14	1.09
2	1	Top	0-10	T0	2	5.65	0.165	1.04	208	12.25	5.15	0.71	0.94	1.35
3	1	Top	0-10	T0	3	5.14	0.260	0.95	300	13.02	5.68	0.68	0.60	1.41
4	1	Top	0-10	T0	4	5.14	0.169	1.10	248	11.92	7.88	1.09	1.01	1.64
9	3	Top	30-60	T3	1	4.37	0.112	1.07	121	8.85	10.35	0.74	5.74	5.73
10	3	Top	30-60	T3	2	4.39	0.058	1.54	115	4.73	6.91	0.77	5.85	6.45

```
kable(head(filter(soils, pH <= 5))) # keep rows where pH is less than or equal to 5
```

	Group	Contour	Depth	Gp	Block	pH	N	Dens	P	Ca	Mg	K	Na	Conduc
7	2	Top	10-30	T1	3	4.70	0.100	1.52	117	8.74	8.16	0.39	3.32	4.16
8	2	Top	10-30	T1	4	4.46	0.112	1.47	170	9.49	9.16	0.70	3.76	5.14
9	3	Top	30-60	T3	1	4.37	0.112	1.07	121	8.85	10.35	0.74	5.74	5.73
10	3	Top	30-60	T3	2	4.39	0.058	1.54	115	4.73	6.91	0.77	5.85	6.45
11	3	Top	30-60	T3	3	4.17	0.078	1.26	112	6.29	7.95	0.26	5.30	8.37
12	3	Top	30-60	T3	4	3.89	0.070	1.42	117	6.61	9.76	0.41	8.30	9.21

Combining and separating

`paste()` and `paste0()` can be used to combine text. The first 2 arguments are the objects you want to paste together, and `sep =` specifies what goes between the two items. For example `sep = " "` would add a space, `sep = ""` means no space (aka no separator), and in the example below "_" adds and underscore separator.

```
rownames(dat) <- paste(soils$Gp, soils$Block, sep = "_") # newly combined columns become rownames

kable(head(dat), row.names=TRUE)
```

	Depth	pH	N	Dens	P	Ca	Mg	K	Na	Conduc
T0_1	0-10	5.40	0.188	0.92	215	16.35	7.65	0.72	1.14	1.09
T0_2	0-10	5.65	0.165	1.04	208	12.25	5.15	0.71	0.94	1.35
T0_3	0-10	5.14	0.260	0.95	300	13.02	5.68	0.68	0.60	1.41
T0_4	0-10	5.14	0.169	1.10	248	11.92	7.88	1.09	1.01	1.64
T1_1	10-30	5.14	0.164	1.12	174	14.17	8.12	0.70	2.17	1.85
T1_2	10-30	5.10	0.094	1.22	129	8.55	6.92	0.81	2.67	3.18

```
meta$id <- paste(soils$Gp, soils$Block, sep = "_") # newly combined columns now become a new id column for the soils dataset or the meta and data data frames we made made in the previous section

dat$id <- paste(soils$Gp, soils$Block, sep = "_") # I'm making the same id column for the dat data set so we have a column we can use to merge them back together later in this tutorial
```

`separate()` splits something into 2 or more groups/columns. `sep =` is used here to specify where in a string separation should occur. In this case I've separated Gp and Block back into separate columns.

```
kable(head(dat))
```

	Depth	pH	N	Dens	P	Ca	Mg	K	Na	Conduc	id
T0_1	0-10	5.40	0.188	0.92	215	16.35	7.65	0.72	1.14	1.09	T0_1
T0_2	0-10	5.65	0.165	1.04	208	12.25	5.15	0.71	0.94	1.35	T0_2
T0_3	0-10	5.14	0.260	0.95	300	13.02	5.68	0.68	0.60	1.41	T0_3
T0_4	0-10	5.14	0.169	1.10	248	11.92	7.88	1.09	1.01	1.64	T0_4
T1_1	10-30	5.14	0.164	1.12	174	14.17	8.12	0.70	2.17	1.85	T1_1
T1_2	10-30	5.10	0.094	1.22	129	8.55	6.92	0.81	2.67	3.18	T1_2

```
temp <- separate(data = dat, col = id, into = c("Gp", "Block"), sep = "_")

kable(head(temp))
```

	Depth	pH	N	Dens	P	Ca	Mg	K	Na	Conduc	Gp	Block
T0_1	0-10	5.40	0.188	0.92	215	16.35	7.65	0.72	1.14	1.09	T0	1
T0_2	0-10	5.65	0.165	1.04	208	12.25	5.15	0.71	0.94	1.35	T0	2
T0_3	0-10	5.14	0.260	0.95	300	13.02	5.68	0.68	0.60	1.41	T0	3
T0_4	0-10	5.14	0.169	1.10	248	11.92	7.88	1.09	1.01	1.64	T0	4
T1_1	10-30	5.14	0.164	1.12	174	14.17	8.12	0.70	2.17	1.85	T1	1
T1_2	10-30	5.10	0.094	1.22	129	8.55	6.92	0.81	2.67	3.18	T1	2

Removing or replacing column content

`gsub()` searches for matches to the `pattern` provided within each element of a character vector. There is a set of related functions you can learn about by searching:

?gsub

```
rownames(dat) <- gsub(pattern = "_", replacement = ".", x = rownames(dat))

kable(head(dat), row.names=TRUE)
```

	Depth	pH	N	Dens	P	Ca	Mg	K	Na	Conduc	id
T0.1	0-10	5.40	0.188	0.92	215	16.35	7.65	0.72	1.14	1.09	T0_1
T0.2	0-10	5.65	0.165	1.04	208	12.25	5.15	0.71	0.94	1.35	T0_2
T0.3	0-10	5.14	0.260	0.95	300	13.02	5.68	0.68	0.60	1.41	T0_3
T0.4	0-10	5.14	0.169	1.10	248	11.92	7.88	1.09	1.01	1.64	T0_4

Regular expressions

Regular expressions are a whole different set of codes used in many programing languages that can allow you to get more specific with what part of a string or text and/or numbers you want to replace or remove. In this example "*.*-" means select everything after the hyphen and replace it with nothing.

```
dat$max_depth <- gsub(pattern = ".*-", replacement = "", x = dat$Depth)

kable(head(dat))
```

	Depth	pH	N	Dens	P	Ca	Mg	K	Na	Conduc	id	max_depth
T0.1	0-10	5.40	0.188	0.92	215	16.35	7.65	0.72	1.14	1.09	T0_1	10

T0.2	0-10	5.65	0.165	1.04	208	12.25	5.15	0.71	0.94	1.35	T0_2	10
T0.3	0-10	5.14	0.260	0.95	300	13.02	5.68	0.68	0.60	1.41	T0_3	10
T0.4	0-10	5.14	0.169	1.10	248	11.92	7.88	1.09	1.01	1.64	T0_4	10
T1.1	10-30	5.14	0.164	1.12	174	14.17	8.12	0.70	2.17	1.85	T1_1	30
T1.2	10-30	5.10	0.094	1.22	129	8.55	6.92	0.81	2.67	3.18	T1_2	30

Merge data frames

If you have 2 datasets that each contain a column with the same row identifiers names you can use `merge()` to attach them side-by-side into one dataframe with the rows correctly matched. If you have a more complicated merge, check out the join functions in the dplyr/tidyverse packages: inner_join(), left_join(), right_join(), full_join().

In this example I'll merge meta (metadata data frame made in a previous section) and dat (the measurment columns data frame made in the previous section). We conveniently made unique identifiers for each row in a column named `id` for both these datasets. Therefore we can merge by this column.

```
names(dat)

##  [1] "Depth"     "pH"        "N"         "Dens"      "P"         "Ca"
##  [7] "Mg"        "K"         "Na"        "Conduc"    "id"        "max_depth"

merged <- merge(meta, dat, by = "id")

kable(head(merged))
```

id	Block	Gp	Depth	pH	N	Dens	P	Ca	Mg	K	Na	Conduc	max_depth
D0_1	1	D0	0-10	5.24	0.194	1.00	445	12.27	6.27	0.72	1.02	0.75	10
D0_2	2	D0	0-10	5.20	0.256	0.78	380	11.39	7.55	0.78	1.63	2.20	10
D0_3	3	D0	0-10	5.30	0.136	1.00	259	9.96	8.08	0.45	1.97	2.27	10
D0_4	4	D0	0-10	5.67	0.127	1.13	248	9.12	7.04	0.55	1.43	0.67	10
D1_1	1	D1	10-30	4.46	0.087	1.24	276	7.24	9.40	0.43	4.17	5.08	30

Transposing (flipping dataset on its side)

Sometimes it is necessary to transpose matrices in order to peform actions on the data. I would not recommend transposing most data frames (nor matrices that contain multiple types of measurements like this one) because the best practice is to have columns = measurements and metadata columns and rows = sample units.

```
flipped <- t(dat)

# take a look
kable(flipped[1:4,1:8]) # no need to include the kable function. This makes a
```

```
cleaner output. This time, instead of using head to look at the data I specified
the rows and columns using indexing.
```

	T0.1	T0.2	T0.3	T0.4	T1.1	T1.2	T1.3	T1.4
Depth	0-10	0-10	0-10	0-10	10-30	10-30	10-30	10-30
pH	5.40	5.65	5.14	5.14	5.14	5.10	4.70	4.46
N	0.188	0.165	0.260	0.169	0.164	0.094	0.100	0.112
Dens	0.92	1.04	0.95	1.10	1.12	1.22	1.52	1.47