Lesson 5: Data Wrangling with Tidy Data, Part 2

Announcements

Schedule / Cheatsheet

- We are in Week 6, Lesson 5
- Video, Slide Links, and summaries of Lessons
- Please add to the cheatsheet

Last Day of Class

- Last day of class (with Pizza) will be November 13 (Hybrid) and November 14 (Remote)
- Remote folks are welcome to join us, make sure to RSVP
- RSVP Here with Dietary Restrictions/preferences

Data-thon

- November 22 (10 AM - 3 PM)
- Come for however long you like
- Explore 3 datasets with Python/R/Whatever tools people want
- Let us know your interest in the RSVP form (previous page)

Community Session

- Next week (November 6 12-1:30 PST)
- Suggest Topics Here

Last Week Today

Clear Points

> The in-class example problems as we went throughout the lesson were really helpful!

> The distinction between `filter()` and `select()`

> Everything you covered.

> The graphics were helpful in remembering filtering vs sorting

Muddy Points

> I'm curious when you would choose to use tidyverse vs. base R?

- It mostly comes down to the simplified syntax and pipes. Tidyverse expects data in a certain format (tidy format), and you can do a lot of things with it.
- There are also specialized packages for dealing with data types (`{lubridate}` for dates, `{stringr}` for characters, `{forcats}` for categorical data)
- If you don't like it, there is always Base R.

Muddy Points

> Unclear things started to be more clearer gradually, just need to find more time to practice.

Stay curious and ask questions if you can!

> couldn’t figure out how to put numeric and logical data together

Are you talking about filtering using both?

Back to Tidyverse and `{dplyr}` functions!

Function Name	Purpose	When
`select()`	Selects sets of columns in df	Lesson 4
`filter()`	Filters rows in df	Lesson 4
`mutate()`	Calculate a New Column in df	Lesson 5
`group_by()/summarize()`	Calculate summary statistics across groups	Lesson 5
`arrange()`	Sorts a df by one or more columns	Lesson 5
`_join()`	Functions to merge two tables together	Lesson 5
`|>`	Operation to build pipelines	Lesson 4

Modifying and creating new columns

The `mutate()` function takes in the following arguments: the first argument is the dataframe of interest, and the second argument is a *new or existing data variable* that is defined in terms of *other data variables*.

I think of them like formulas in Excel. . . .

```
metadata |>
  select(ModelID, Age) |>
  head()
```

	ModelID <chr>	Age <dbl>
1	ACH-000001	60
2	ACH-000002	36
3	ACH-000003	72
4	ACH-000004	30
5	ACH-000005	30
6	ACH-000006	64

6 rows

```
metadata2 <- metadata |>
             mutate(olderAge = Age + 10)
```

```
metadata2 |>
  select(ModelID, Age, olderAge) |>
  head()
```

	ModelID <chr>	Age <dbl>	olderAge <dbl>
1	ACH-000001	60	70
2	ACH-000002	36	46
3	ACH-000003	72	82
4	ACH-000004	30	40
5	ACH-000005	30	40
6	ACH-000006	64	74

6 rows

```
metadata |>
  mutate(olderAge = Age + 10)
```

ModelID <chr>	PatientID <chr>	CellLineName <chr>	StrippedCellLineName <chr>	Ag <dbl
ACH-000001	PT-gj46wT	NIH:OVCAR-3	NIHOVCAR3	6
ACH-000002	PT-5qa3uk	HL-60	HL60	3
ACH-000003	PT-puKIyc	CACO2	CACO2	7
ACH-000004	PT-q4K2cp	HEL	HEL	3
ACH-000005	PT-q4K2cp	HEL 92.1.7	HEL9217	3
ACH-000006	PT-ej13Dz	MONO-MAC-6	MONOMAC6	6
ACH-000007	PT-NOXwpH	LS513	LS513	6
ACH-000008	PT-fp8PeY	A101D	A101D	5
ACH-000009	PT-puKIyc	C2BBe1	C2BBE1	7
ACH-000011	PT-AR7W9o	253J	253J	5

expression data.frame

```
expression$KRAS_Exp[1:10]
```

```
[1] 4.634012 4.638653 4.032101 5.503031 3.713696 3.972693 3.235727 4.135042
[9] 9.017365 3.940167
```

```
expression2 = expression |>
  mutate(log_KRAS_Exp = log(KRAS_Exp))
```

```
expression2$log_KRAS_Exp[1:10]
```

```
[1] 1.533423 1.534424 1.394288 1.705299 1.312028 1.379444 1.174254 1.419498
[9] 2.199152 1.371223
```

Try it Out 1

One thing we often need to do is *scale* or *normalize* data. Let's try scaling by the maximum value in a column. Try calculating a new variable `KRAS_scaled` by calculating:

```
KRAS_Exp / max(KRAS_Exp)
```

R Code | Start Over | Run Code

```
expression2 = expression |>
  mutate(KRAS_scaled = --------) |>
  select(ModelID, KRAS_Exp, KRAS_scaled)

head(expression2)
```

Try it Out 2

Try calcluating a new variable in `mutation` called `TP53_CDKN2A` which is equal to:

- `TP53_Mut | CDKN2A_Mut`

Then filter `new_mutation` by `TP53_CDKN2A == TRUE`. How many rows did you return?

R Code | Start Over | Run Code

```
new_mutation <- mutation |>
  mutate(TP53_CDKN2A = --------) |>
  select(ModelID, TP53_CDKN2A, TP53_Mut, CDKN2A_Mut)

new_mutation |>
  filter(TP53_CDKN2A == TRUE)
```

Grouping and summarizing dataframes

```
metadata_grouped = group_by(metadata, OncotreeLineage)
metadata_type = summarise(metadata_grouped, MeanAge = mean(Age, na.rm = TRUE), Count = n())
```

```
head(metadata_type)
```

OncotreeLineage <chr>	MeanAge <dbl>	Count <int>
Adrenal Gland	55.00000	1
Ampulla of Vater	65.50000	4
Biliary Tract	58.45000	40
Bladder/Urinary Tract	65.16667	39
Bone	20.85455	75
Bowel	58.61111	87

6 rows

Grouping and summarizing dataframes

The `group_by()` function returns the identical input dataframe but remembers which variable(s) have been marked as grouped.

The `summarise()` (you can also use `summarize()`) returns one row for each combination of grouping variables, and one column for each of the summary statistics that you have specified.

Functions you can use for `summarise()` must take in a vector and return a simple data type, such as any of our summary statistics functions: `mean()`, `median()`, `min()`, `max()`, etc.

The exception is `n()`, which returns the number of entries for each grouping variable's value.

Try it Out 1

First group by `OncotreeLineage`. Then try calculating the maximum age as `max_age` within `metadata` when it is grouped by `OncotreeLineage`:

R Code | Start Over | Run Code

```
1 metadata |>
2   group_by(-----) }>
3   summarize(max_age = max(-----))
```

Quick Note

Our summarization has a lot of NAs. We can use the `na.rm` argument in `max()` to drop `NA` values in our calculation.

R Code | Start Over | Run Code

```
metadata |>
  group_by(OncotreeLineage) |>
  summarize(max_age = max(Age, na.rm=TRUE))
```

Try it Out 2

First group by `OncotreeLineage`. Then try calculating the median age as `median_age` within `metadata` when it is grouped by `OncotreeLineage`. (Hint: use the `median()` function. Bonus points if you use the `na.rm` argument)

R Code | Start Over | Run Code

```
metadata |>
  group_by(------) }>
  summarize(median_age = -------)
```

Think about this:

What is the difference between these two pipelines?

R Code Start Over Run Code

```r
metadata |>
   mutate(max_age=max(Age, na.rm=TRUE)) |>
   select(ModelID, OncotreeLineage, max_age)
```

R Code Start Over Run Code

```r
metadata |>
   group_by(OncotreeLineage) |>
   summarize(max_age=max(Age, na.rm=TRUE))
```

arrange()

If we need to sort by a column, we can use `arrange()`:

```
metadata |>
  arrange(Age)
```

ModelID <chr>	PatientID <chr>	CellLineName <chr>	StrippedCellLineName <chr>	Age <dbl>
ACH-000096	PT-ABKGjf	G-401	G401	0
ACH-000160	PT-0GrZ0k	BT-12	BT12	0
ACH-000203	PT-9PJrBN	NH-6	NH6	0
ACH-000206	PT-vEZ2Aw	C8166	C8166	0
ACH-000375	PT-U7nXm3	G-402	G402	0
ACH-000597	PT-XDzUk4	TTC-709	TTC709	0
ACH-000602	PT-T0am5q	M-07e	M07E	0
ACH-000607	PT-UnVDKb	KYM-1	KYM1	0
ACH-000641	PT-XYdNvp	CMK	CMK	0
ACH-001018	PT-xU6CEy	BJ hTERT	BJHTERT	0

1-10 of 1,864 rows | 1-5 of 30 columns

Previous 1 2 3 4 5 6 … 187 Next

Descending order

If we want to sort by descending order, then we can wrap our column in `desc()` (short for descending):

```
metadata |>
  arrange(desc(Age))
```

ModelID <chr>	PatientID <chr>	CellLineName <chr>	StrippedCellLineName <chr>	Age <dbl>
ACH-000923	PT-6XYxNF	BCP-1	BCP1	94
ACH-001654	PT-2qao55	SK-GT-4	SKGT4	89
ACH-000237	PT-wwKJYr	JHOM-1	JHOM1	88
ACH-000794	PT-zkbKhd	BICR 22	BICR22	88
ACH-002100	PT-wu5yKY	DJM-1	DJM1	87
ACH-001400	PT-2KkY6l	SW 954	SW954	86
ACH-001442	PT-QwG6Ej	A388	A388	86
ACH-002120	PT-dBVaTX	GAK	GAK	86
ACH-002512	PT-YwNqCa	MM160113	MM160113	86
ACH-000417	PT-9CcA8P	Panc 08.13	PANC0813	85

1-10 of 1,864 rows | 1-5 of 30 columns

Previous 1 2 3 4 5 6 … 187 Next

Sorting by Multiple Columns

We can sort by multiple columns by passing in multiple variables:

```
metadata |>
  arrange(desc(Age), ModelID)
```

ModelID <chr>	PatientID <chr>	CellLineName <chr>	StrippedCellLineName <chr>	Age <dbl>
ACH-000923	PT-6XYxNF	BCP-1	BCP1	94
ACH-001654	PT-2qao55	SK-GT-4	SKGT4	89
ACH-000237	PT-wwKJYr	JHOM-1	JHOM1	88
ACH-000794	PT-zkbKhd	BICR 22	BICR22	88
ACH-002100	PT-wu5yKY	DJM-1	DJM1	87
ACH-001400	PT-2KkY6l	SW 954	SW954	86
ACH-001442	PT-QwG6Ej	A388	A388	86
ACH-002120	PT-dBVaTX	GAK	GAK	86
ACH-002512	PT-YwNqCa	MM160113	MM160113	86
ACH-000417	PT-9CcA8P	Panc 08.13	PANC0813	85

1-10 of 1,864 rows | 1-5 of 30 columns

Previous 1 2 3 4 5 6 … 187 Next

Try it Out

Try sorting by `desc(Age)` and `OncotreeLineage`, and then by `OncotreeLineage` and `desc(Age)`. Does order matter?

R Code | Start Over | Run Code

```
metadata |>
    arrange(desc(Age), ------) |>
    select(ModelID, Age, OncotreeLineage) |>
    head()
```

R Code | Start Over | Run Code

```
metadata |>
    arrange(OncotreeLineage, ------) |>
    select(ModelID, Age, OncotreeLineage) |>
    head()
```

Merging two dataframes together

`metadata`

ModelID	OncotreeLineage	Age
“ACH-001113”	“Lung”	69
“ACH-001289”	“CNS/Brain”	NA
“ACH-001339”	“Skin”	14

`expression`

ModelID	PIK3CA_Exp	log_PIK3CA_Exp
“ACH-001113”	5.138733	1.636806
“ACH-001289”	3.184280	1.158226
“ACH-001339”	3.165108	1.152187

I want to compare the relationship between `OncotreeLineage` and `PIK3CA_Exp`:

ModelID	PIK3CA_Exp	log_PIK3CA_Exp	OncotreeLineage	Age
“ACH-001113”	5.138733	1.636806	“Lung”	69
“ACH-001289”	3.184280	1.158226	“CNS/Brain”	NA
“ACH-001339”	3.165108	1.152187	“Skin”	14

Merging two dataframes together

One strategy we can use is to only merge on common ids between the two tables. If one table has ids that aren't in the second table, then we'll remove those rows.

We see that in both dataframes, the rows (observations) represent cell lines with a common column `ModelID`, so let's merge these two dataframes together, using `inner_join()`:

```
1 merged = inner_join(metadata, expression, by = "ModelID")
```

Let's take a look at the dimensions:

```
1 dim(metadata)
```

```
[1] 1864   30
```

```
1 dim(expression)
```

```
[1] 1450  536
```

```
1 dim(merged)
```

```
[1] 1450  565
```

Which rows do we keep from each table?

`inner_join()` keeps all observations common to both dataframes based on the common column defined via the `by` argument.

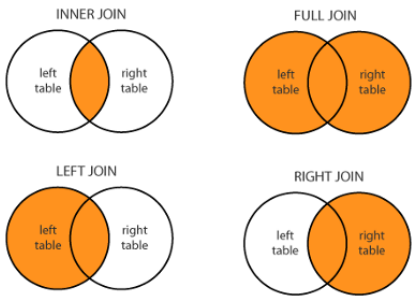
```
merged |>
  select(ModelID, KRAS_Exp, OncotreeLineage)
```

ModelID <chr>	KRAS_Exp <dbl>	OncotreeLineage <chr>
ACH-000001	5.800123	Ovary/Fallopian Tube
ACH-000002	2.625270	Myeloid
ACH-000003	5.284662	Bowel
ACH-000004	4.235727	Myeloid
ACH-000005	4.887525	Myeloid
ACH-000006	3.947666	Myeloid
ACH-000007	3.814550	Bowel
ACH-000008	3.604071	Skin
ACH-000009	5.725196	Bowel
ACH-000011	2.918386	Bladder/Urinary Tract

1-10 of 1,450 rows

Previous 1 2 3 4 5 6 … 145 Next

Merging two dataframes together: variations



Given `xxx_join(x, y, by = "common_col")`,

- `full_join()` keeps all observations.
- `left_join()` keeps all observations in `x`.
- `right_join()` keeps all observations in `y`.
- `inner_join()` keeps observations common to both `x` and `y`.

Try it Out

Try out `inner_join()` on the two tables and compare the number of rows in the merged table.

R Code | Start Over | Run Code

```r
merged_table_inner <- metadata |>
  inner_join(y=expression, by=ModelID) |>
  select(ModelID, KRAS_Exp, OncotreeLineage)

nrow(merged_table_inner)
```

Now try `full_join()` on the two tables and compare the number of rows in the merged table.

R Code | Start Over | Run Code

```r
merged_table_full <- metadata |>
  full_join(y=expression, by=ModelID) |>
  select(ModelID, KRAS_Exp, OncotreeLineage)

nrow(merged_table_full)
merged_table_full
```

Next Week

Learning Community Week!

How did we do?

Weekly Checkin