

Data Transformation

Patrick Mathias
Lesson 5
DLMP Fall 2021

Goal

1. Learn how to use `dplyr` to transform data frames
2. Appreciate the role of piping in facilitating data transformation

Objectives

1. List the major forms of data transformation implemented in `dplyr`
2. Use the pipe operator to pass the output of one function as an input to the next function
3. Create new calculated columns not found in the original data frame

Reordering Rows

arrange()

Order rows by values in a column



- = Number of rows
- = Number of Columns



arrange()

Order rows by values in a column

```
arrange(data, ... )
```

data frame to
transform

name(s) of columns to
arrange by



arrange()

Order rows by values in a column

```
arrange(covid_testing, first_name)
```

mrn	first_name	last_name
5000876	sarella	stark
5006017	alester	stark
5001412	jhezane	targaryen
5000533	penny	targaryen



mrn	first_name	last_name
5006017	alester	stark
5001412	jhezane	targaryen
5000533	penny	targaryen
5000876	sarella	stark



arrange()

Order rows by values in a column

```
arrange(covid_testing, desc(mrn))
```

mrn	first_name	last_name
5000876	sarella	stark
5006017	alester	stark
5001412	jhezane	targaryen
5000533	penny	targaryen



mrn	first_name	last_name
5006017	alester	stark
5001412	jhezane	targaryen
5000876	sarella	stark
5000533	penny	targaryen



Your Turn 1

The column `ct_value` contains the cycle threshold (C_t) for the real-time PCR that generated the final result.

How might you use `arrange()` to determine the highest and lowest Ct result in the dataset?



Pop Quiz

The default behavior of `arrange()` is to order from lower to higher values.

When might `arrange()` place "1000" before "50"?



The Pipe Operator: %>%

Data Analysis Steps

```
day_10 <- filter(covid_testing, pan_day <= 10)  
day_10 <- select(day_10, clinic_name)  
day_10 <- arrange(day_10 , clinic_name)
```

1. Filter tests to those on pandemic day less than 10
2. Select the column that contains ordering location
3. Arrange those columns by location



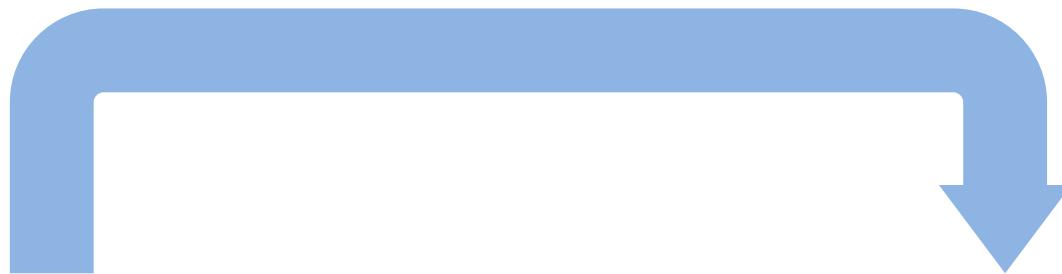
Data Analysis Steps

```
day_10 <- arrange(  
  select(  
    filter(  
      covid_testing,  
      pan_day <= 10  
    ),  
    clinic_name  
  ),  
  clinic_name  
)
```



The Pipe Operator %>%

Passes result on left into first argument of function on right.



```
covid_testing %>% filter(_____, pan_day <= 10)
```

```
filter(covid_testing, pan_day <= 10)  
covid_testing %>% filter(pan_day <= 10)
```



Data Analysis Steps

```
day_10 <- arrange(  
  select(  
    filter(  
      covid_testing,  
      pan_day <= 10  
    ),  
    clinic_name  
  ),  
  clinic_name  
)
```



Data Analysis Steps

```
covid_testing %>%  
  filter(pan_day <= 10) %>%  
  select(clinic_name) %>%  
  arrange(clinic_name)
```



Shortcut to type %>%

Cmd + Shift + M (Mac)

Ctrl + Shift + M (Windows)

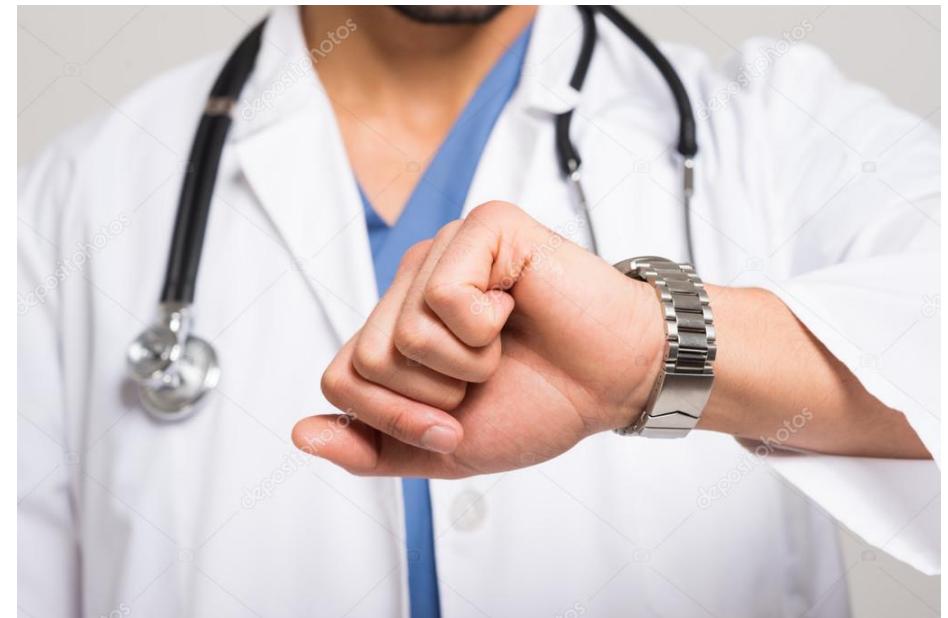


Scene

The PICU would like a word with you because of a recent incident involving a delay in results for a patient who required a AGP

They had to wait over 10 hours before the procedure could begin

You decide to investigate... WITH DATA



Your Turn 2

Use %>% to write a sequence of three functions that:

1. Filters to tests from the clinic (**clinic_name**) of "picu"
2. Selects the column with the receive to verify turnaround time (**rec_ver_tat**) as well as the day from start of the pandemic (**pan_day**)
3. Arrange the `pan_day` from highest to lowest

Using <- , assign the result to a new variable, call it whatever you want.



Isolating data



Extract variables with `select()`

Extract rows with `filter()`

Arrange rows, with `arrange()`.



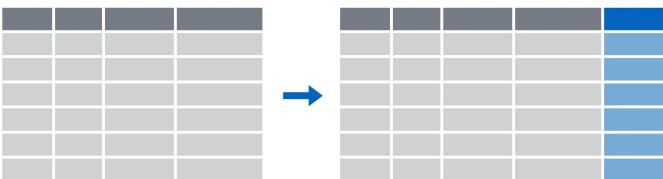
Creating New Columns

What is the mean and median collect to verify turnaround time by clinic?

Breaking down the analytical question

1. Total TAT for each test
2. Group tests by clinic
3. Calculate mean and median for each clinic

Deriving data



Make new variables with **mutate()**



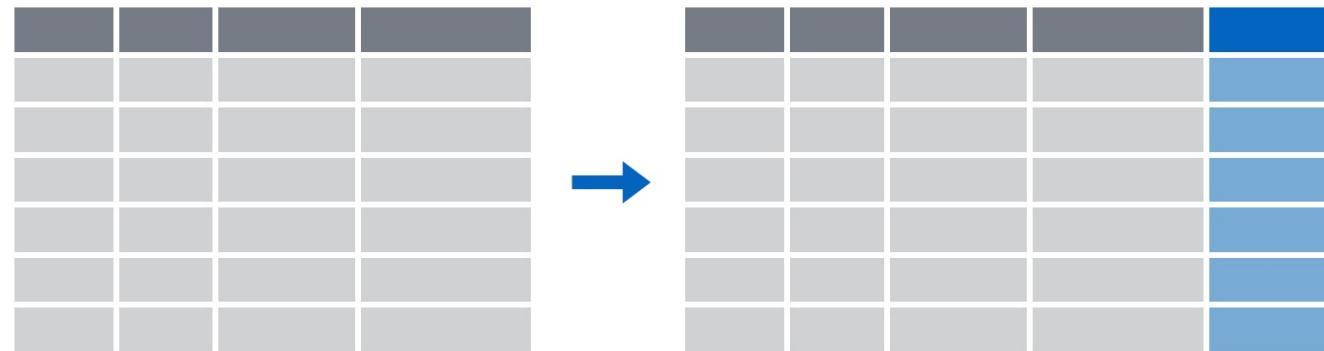
Make summaries of data with
summarize()



mutate()

mutate()

Creating new calculated columns



= Number of rows
↑ Number of Columns



mutate()

Creating new calculated columns

```
Covid_testing %>%  
  mutate(new_column = calculation)
```

name for new column

equals

function whose results will populate columns



mutate()

Creating new calculated columns

```
covid_testing %>%  
  mutate(c_r_tat_mins = col_rec_tat * 60)
```

mrn	col_rec_tat	rec_ver_tat
5000876	29.5	11.5
5006017	3.6	5
5001412	1.4	5.2
5000533	2.3	5.8



mrn	col_rec_tat	rec_ver_tat	c_r_tat_mins
5000876	29.5	11.5	1770
5006017	3.6	5	216
5001412	1.4	5.2	84
5000533	2.3	5.8	138

Your Turn 3

Create a new column using the `mutate()` function that contains the total TAT (sum of `col_rec_tat` and `rec_ver_tat`)



Functions to use in mutate()

Vector Functions

TO USE WITH MUTATE ()

mutate() and **transmute()** apply vectorized functions to columns to create new columns. Vectorized functions take vectors as input and return vectors of the same length as output.

vectorized function

OFFSETS

dplyr::lag() - Offset elements by 1
dplyr::lead() - Offset elements by -1

CUMULATIVE AGGREGATES

dplyr::cumall() - Cumulative all()
dplyr::cumany() - Cumulative any()
 cummax() - Cumulative max()
dplyr::cummean() - Cumulative mean()
 cummin() - Cumulative min()
 cumprod() - Cumulative prod()
 cumsum() - Cumulative sum()

RANKINGS

dplyr::cume_dist() - Proportion of all values <=
dplyr::dense_rank() - rank with ties = min, no gaps
dplyr::min_rank() - rank with ties = min
dplyr::ntile() - bins into n bins
dplyr::percent_rank() - min_rank scaled to [0,1]
dplyr::row_number() - rank with ties = "first"

MATH

+, -, *, /, ^, %/%, %% - arithmetic ops
log(), log2(), log10() - logs
<, <=, >, >=, !=, == - logical comparisons
dplyr::between() - x >= left & x <= right
dplyr::near() - safe == for floating point numbers

MISC

dplyr::case_when() - multi-case if_else()
dplyr::coalesce() - first non-NA values by element across a set of vectors
dplyr::if_else() - element-wise if() + else()
dplyr::na_if() - replace specific values with NA
 pmax() - element-wise max()
 pmin() - element-wise min()
dplyr::recode() - Vectorized switch()
dplyr::recode_factor() - Vectorized switch() for factors

Vector Functions

TO USE WITH MUTATE ()

mutate() and **transmute()** apply vectorized functions to columns to create new columns. Vectorized functions take vectors as input and return vectors of the same length as output.

vectorized function

OFFSETS

dplyr::lag() - Offset elements by 1
dplyr::lead() - Offset elements by -1

CUMULATIVE AGGREGATES

dplyr::cumall() - Cumulative all()
dplyr::cumany() - Cumulative any()
 cummax() - Cumulative max()

RANKINGS

dplyr::cume_dist() - Proportion of all values <= min
dplyr::dense_rank() - rank with ties = min, no gaps
dplyr::min_rank() - rank with ties = min
dplyr::ntile() - bins into n bins
dplyr::percent_rank() - min_rank scaled to [0,1]
dplyr::row_number() - rank with ties = "first"

MATH

+, -, *, /, ^, %/%, %% - arithmetic ops
log(), log2(), log10() - logs
<, <=, >, >=, !=, == - logical comparisons
dplyr::between() - x >= left & x <= right
dplyr::near() - safe == for floating point numbers

MISC

dplyr::case_when() - multi-case if_else()
dplyr::coalesce() - first non-NA values by element across a set of vectors
dplyr::if_else() - element-wise if() + else()
dplyr::na_if() - replace specific values with NA
 pmax() - element-wise max()
 pmin() - element-wise min()
dplyr::recode() - Vectorized switch()
dplyr::recode_factor() - Vectorized switch() for factors

Summary Functions

TO USE WITH SUMMARISE ()

summarise() applies summary functions to columns to create a new table. Summary functions take vectors as input and return single values as output.

summary function

COUNTS

dplyr::n() - number of values/rows
dplyr::n_distinct() - # of uniqueness
 sum(is.na()) - # of non-NA's

LOCATION

mean() - mean, also **mean(is.na())**
median() - median

LOGICALS

mean() - proportion of TRUE's
sum() - # of TRUE's

POSITION/ORDER

dplyr::first() - first value
dplyr::last() - last value
dplyr::nth() - value in nth location of vector

RANK

quantile() - nth quantile
min() - minimum value
max() - maximum value

SPREAD

IQR() - Inter-Quartile Range
mad() - median absolute deviation
sdi() - standard deviation
var() - variance

Row Names

Tidy data does not use rownames, which store a variable outside of the columns. To work with the rownames, first move them into a column.

rownames_to_column()

Move row names into col.
o <- rownames_to_column(iris, var = "C")

column_to_rownames()

Move col in row names.
 column_to_rownames(a, var = "C")

Also has **rownames()**, **remove_rownames()**

Combine Tables

COMBINE VARIABLES

x + **y** = **bind_cols()**
Use **bind_cols()** to paste tables beside each other as they are.

x + **y** = **bind_col()**
Returns tables placed side by side as a single table.
BE SURE THAT ROWS ALIGN.

Use a "Mutating Join" to join one table to columns from another, matching values with the rows that they correspond to. Each join retains a different combination of values from the tables.

left_join(x, y, by = NULL, copy = FALSE, suffix = c("x", "y"))
Join matching values from y to x.

right_join(x, y, by = NULL, copy = FALSE, suffix = c("x", "y"))
Join matching values from x to y.

inner_join(x, y, by = NULL, copy = FALSE, suffix = c("x", "y"))
Join data. Retain only rows with matches.

full_join(x, y, by = NULL, copy = FALSE, suffix = c("x", "y"))
Join all values, all rows.

use_by = c("col1", "col2", ...) to specify one or more common columns to match on.
left_join(x, y, by = "A")

use_by = c("col1", "col2", ...) to match on columns that have different names in each table.
left_join(x, y, by = "C" = "D")

use_by = c("col1", "col2", ...) Returns rows of x that have a match in y.
USEFUL TO SEE WHAT WILL BE JOINED.

semi_join(x, y, by = NULL, ...)
Return rows of x that do not have a match in y.
USEFUL TO SEE WHAT WILL NOT BE JOINED.

anti_join(x, y, by = NULL, ...)
Return rows of x that do not have a match in y.

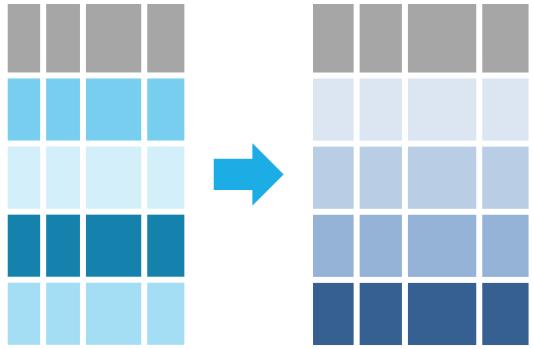


dplyr



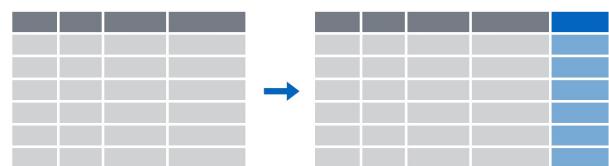
29

Recap

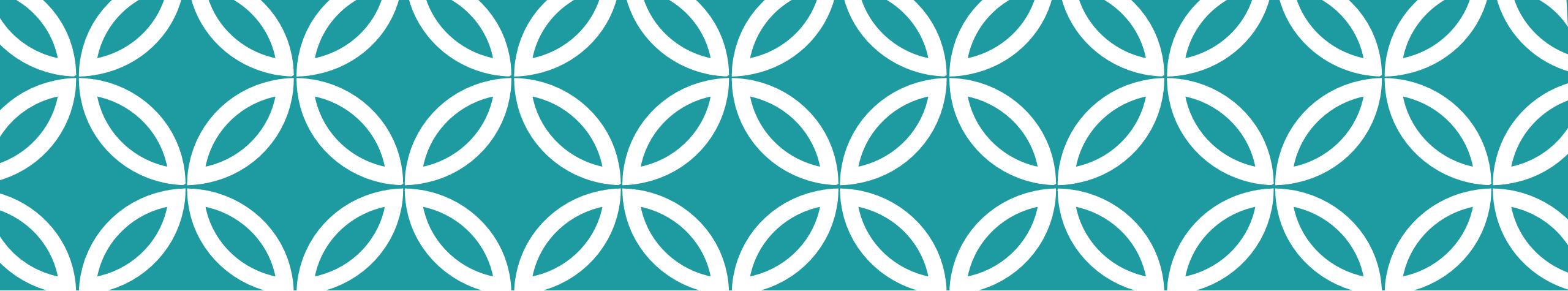


arrange() sorts data based on the data type of the column(s) used to sort

Pipes (%>%) allow you to sequentially apply functions to a data frame efficiently



mutate() adds a column to the data frame that may be based on calculations on one or more other columns



What else?

mutate()

Function to "coerce" one type of data into another type of data

Replacing columns

```
covid_testing %>%  
  mutate(mrn = as.character(mrn))
```

mrn <dbl>	first_name <chr>	last_name <chr>
5000876	sarella	stark
5006017	alester	stark
5001412	jhezane	westerling
5000533	penny	targaryen

mrn <chr>	first_name <chr>	last_name <chr>
5000876	sarella	stark
5006017	alester	stark
5001412	jhezane	westerling
5000533	penny	targaryen



mutate()

Conditionally replacing values

```
covid_testing %>%  
  mutate(last_name = if_else(last_name=="targaryen",  
                            "TARGARYEN",last_name))
```

mrn <dbl>	first_name <chr>	last_name <chr>
5001412	jhezane	westerling
5000533	penny	targaryen
5009134	grunt	rivers
5008518	melisandre	swyft



mrn <dbl>	first_name <chr>	last_name <chr>
5001412	jhezane	westerling
5000533	penny	TARGARYEN
5009134	grunt	rivers
5008518	melisandre	swyft



Goal

1. Learn how to use `dplyr` to transform data frames
2. Appreciate the role of piping in facilitating data transformation

Objectives

1. List the major forms of data transformation implemented in `dplyr`
2. Use the pipe operator to pass the output of one function as an input to the next function
3. Create new calculated columns not found in the original data frame