

# Data Transformation

Session 5  
Dan Herman  
December 13, 2020

*Presentation adapted from...*

# Amrom Obstfeld

Assistant Professor of Clinical Pathology  
and Laboratory Medicine

University of Pennsylvania Perelman  
School of Medicine

Director of Hematology and Coagulation  
Laboratories

Children's Hospital of Philadelphia



# Goal

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

# Objectives

1. Use the pipe operator to pass the output of one function as an input to the next function
2. Create new calculated columns not found in the original data frame

$\% > \%$

# 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)
```

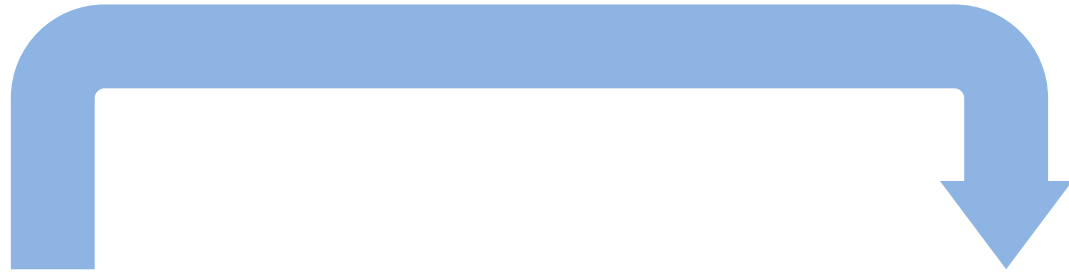
1. Filter tests to those from pandemic day 10 or earlier
2. Select the column that contains the ordering location
3. Arrange those columns by location

# 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(panday <= 10) %>%  
  select(clinic_name) %>%  
  arrange(clinic_name)
```

# Example: TAT investigation

The PICU would like a word with you because of a recent incident involving a delay in results for a patient who required an aerosol generating procedure.

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

You decide to investigate...WITH DATA



# Your Turn 1

Open '05– Transform.Rmd' and run the setup chunk.

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

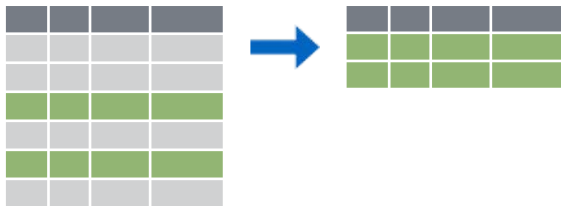
1. Filters to orders from the clinic (**clinic\_name**) of "picu"
2. Selects the columns with the receive to verify turnaround time (**rec\_ver\_tat**) and day from start of the pandemic (**pan\_day**)
3. Arrange the **pan\_day** from highest to lowest
4. Use <- to assign the result to a new variable and take a look.



# Isolating data



Extract variables with **select()**



Extract rows with **filter()**



Arrange rows, with **arrange()**.



# Deriving Data



What are the **average and standard deviation** of the total turnaround time for each ordering clinic?

# Breaking down the analytical question

1. Total TAT for each order
2. Group orders by clinic
3. Calculate average and standard deviation for each clinic

# Deriving data



Make new variables with **mutate()**



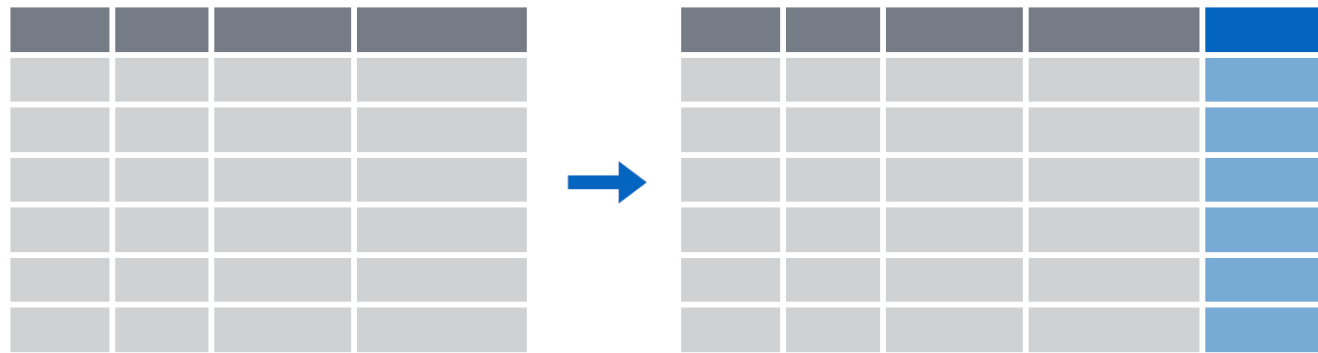
Make summaries of data with **summarize()**



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 2

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



# mutate()

## Replace columns

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

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

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

# 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()  
**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



## 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 uniques  
**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  
**sd()** - 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.  
**a <- 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

Use **bind\_cols()** to paste tables beside each other as they are.

#### bind\_cols(...)

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 data. Retain all values, all rows.

Use **by = c("col1", "col2", ...)** to specify one or more common columns to match on.

Use a named vector, **by = c("col1" = "col2")**, to match on columns that have different names in each table.

Use **suffix** to specify the suffix to give to unmatched columns that have the same name in both tables.

Use **left\_join(x, y, by = c("C" = "D"), suffix = c("1", "2"))**

### COMBINE CASES

Use **bind\_rows()** to paste tables below each other as they are.

#### bind\_rows(..., id = NULL)

Returns tables one on top of the other as a single table. Set id to a column name to add a column of the original table names (as pictured)

**intersect(x, y, ...)**  
Rows that appear in both x and y.

**setdiff(x, y, ...)**  
Rows that appear in x but not y.

**union(x, y, ...)**  
Rows that appear in x or y. (Duplicates removed). **union\_all()** retains duplicates.

Use **setequal()** to test whether two data sets contain the exact same rows (in any order).

### EXTRACT ROWS

Use a "Filtering Join" to filter one table against the rows of another.

**semi\_join(x, y, by = NULL, ...)**  
Return rows of x that have a match in y. USEFUL TO SEE WHAT WILL BE JOINED.

**anti\_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.



# Goal

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

# Objectives

1. Use the pipe operator to pass the output of one function as an input to the next function
2. Create new calculated columns not found in the original data frame