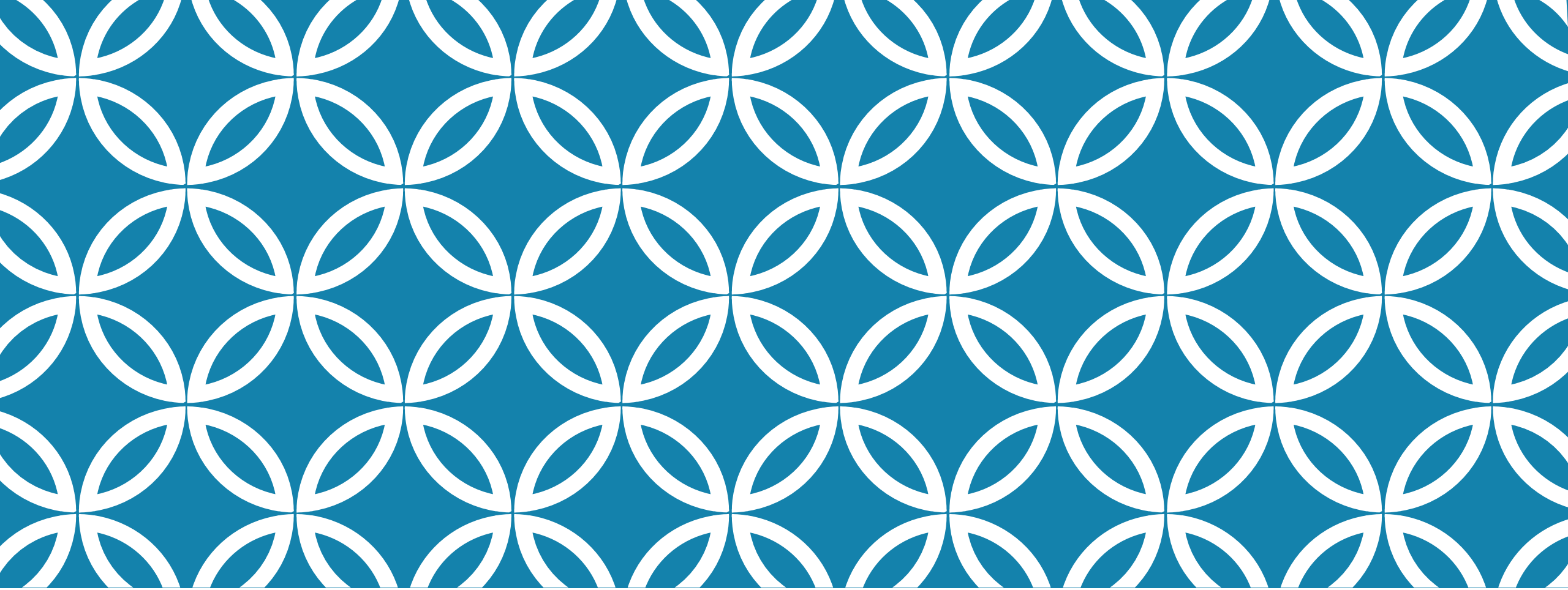


Data Transformation

Session 4

Patrick Mathias

September 26, 2021

September 26, 2021	**Session**	**Instructor**
8:30 am - 8:45 am	Course Introduction	Patrick Mathias
8:45 am – 9:30 am	Intro to R and Reproducible Reporting	Joseph Rudolf
9:45 am - 10:30 am	Coding Basics and Importing Data	Joseph Rudolf
10:45 am – 11:30 am	Data Visualization	Patrick Mathias
LUNCH		
12:30 pm - 1:30 pm	Data Transformation	Patrick Mathias
1:45 pm – 2:45 pm	Grouping and Summarizing Data	Joseph Rudolf
3:00 pm - 3:30 pm	Dashboard Demo and Course Wrap Up	Patrick Mathias

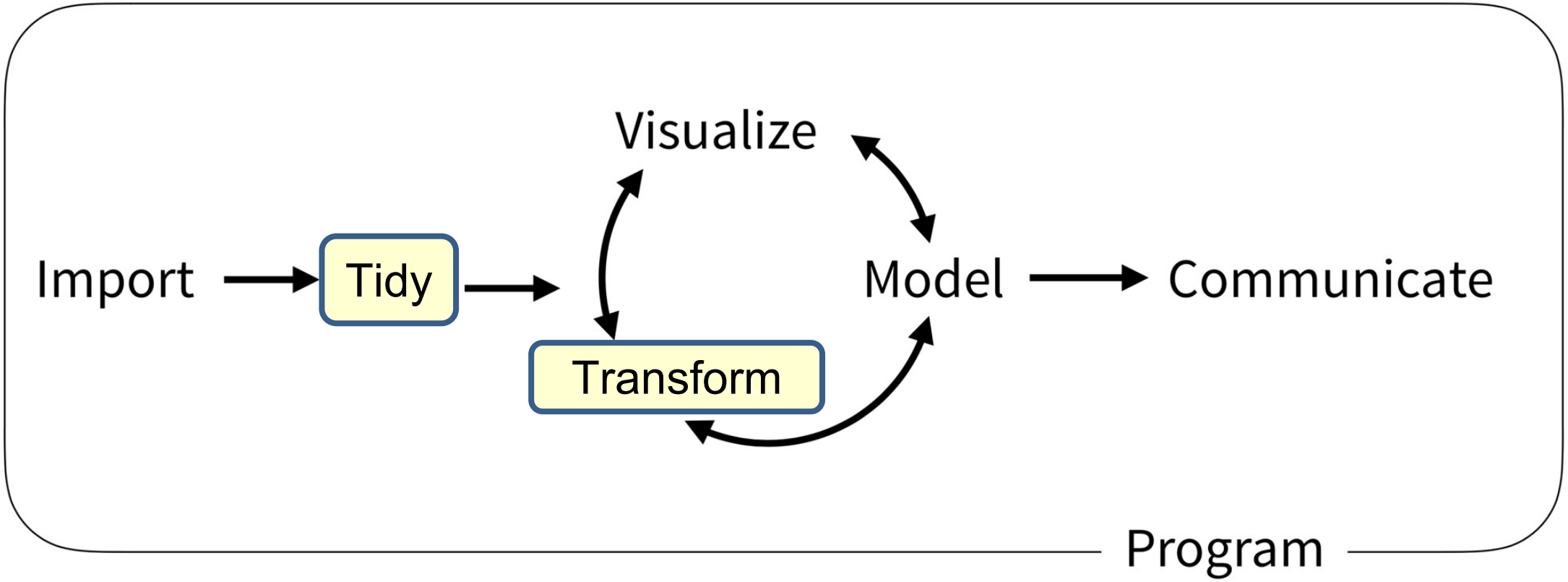
Goals

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

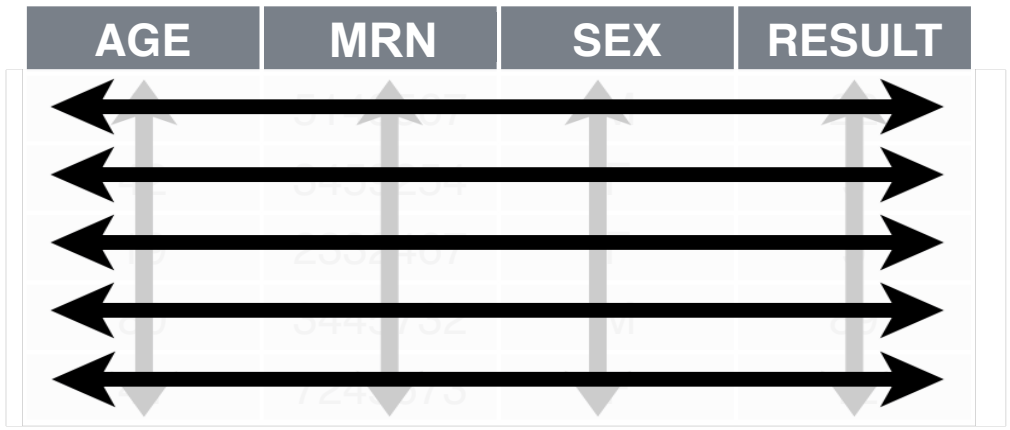
Objectives

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

Typical Data Science Pipeline



What is a "Tidy" Data Frame



A data set is **tidy** if:

1. Each **variable** is in its own **column**
2. Each **observation** is in its own **row**
3. Each **value** is in its own **cell**

Your Turn 1

Open "**04-Transform.Rmd**"

Run the setup chunk

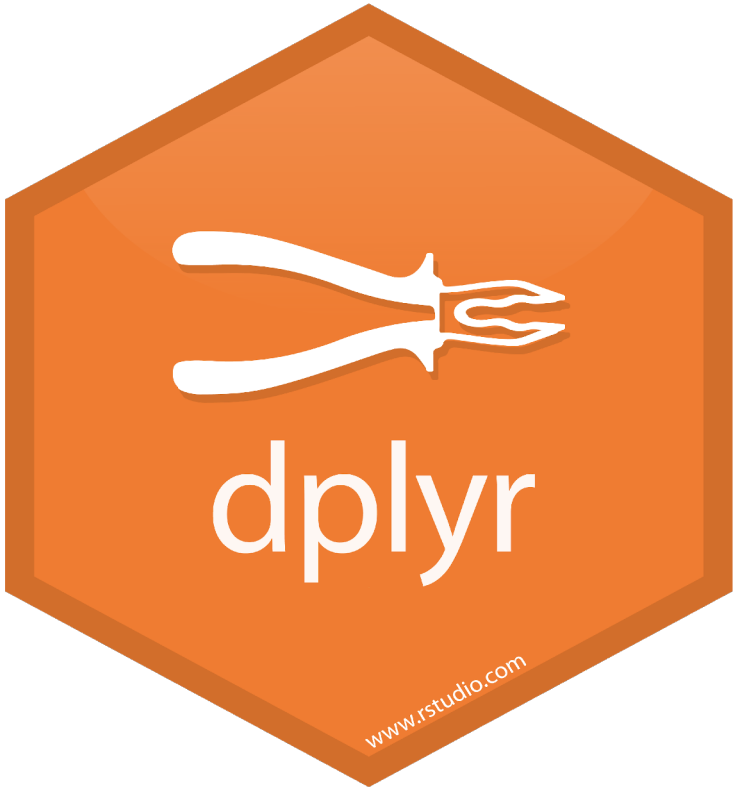
Pop Quiz

How can you confirm that you have successfully loaded the data file into RStudio?

1. The code that imported the data did not yield an error
2. Code that references the `covid_testing` object runs without errors
3. The `covid_testing` object is present in the environment pane
4. All of the above

Transform Data with

dplyr

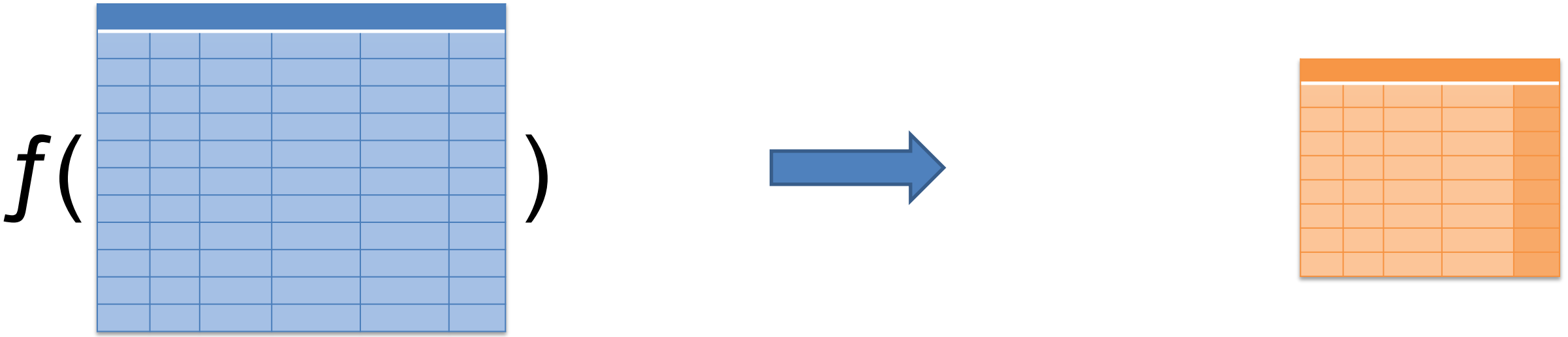


dplyr implements a *grammar* of data manipulation for transforming tabular data.

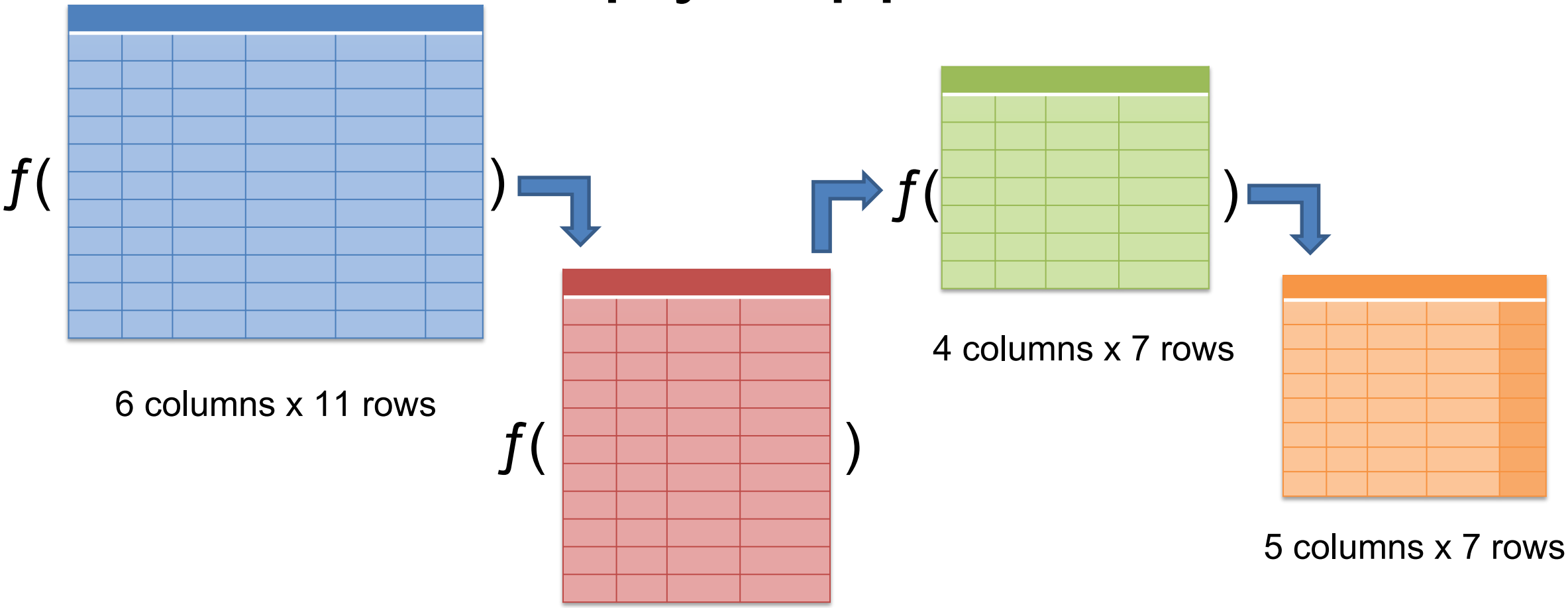
dplyr: a grammar for transforming data

1. **Choose** columns. — **select()**
2. **Extract** rows. — **filter()**
3. **Derive** new columns. — **mutate()**
4. **Change** the unit of analysis. — **summarize()**

Dplyr Approach



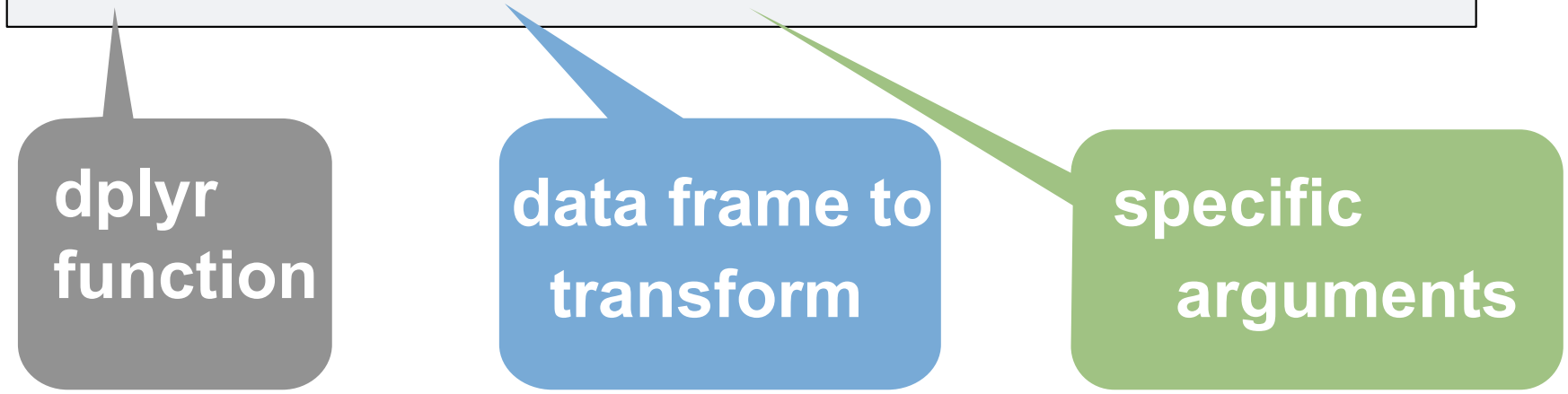
Dplyr Approach



Common syntax

Each function takes a data frame as its first argument and returns a data frame as its output.

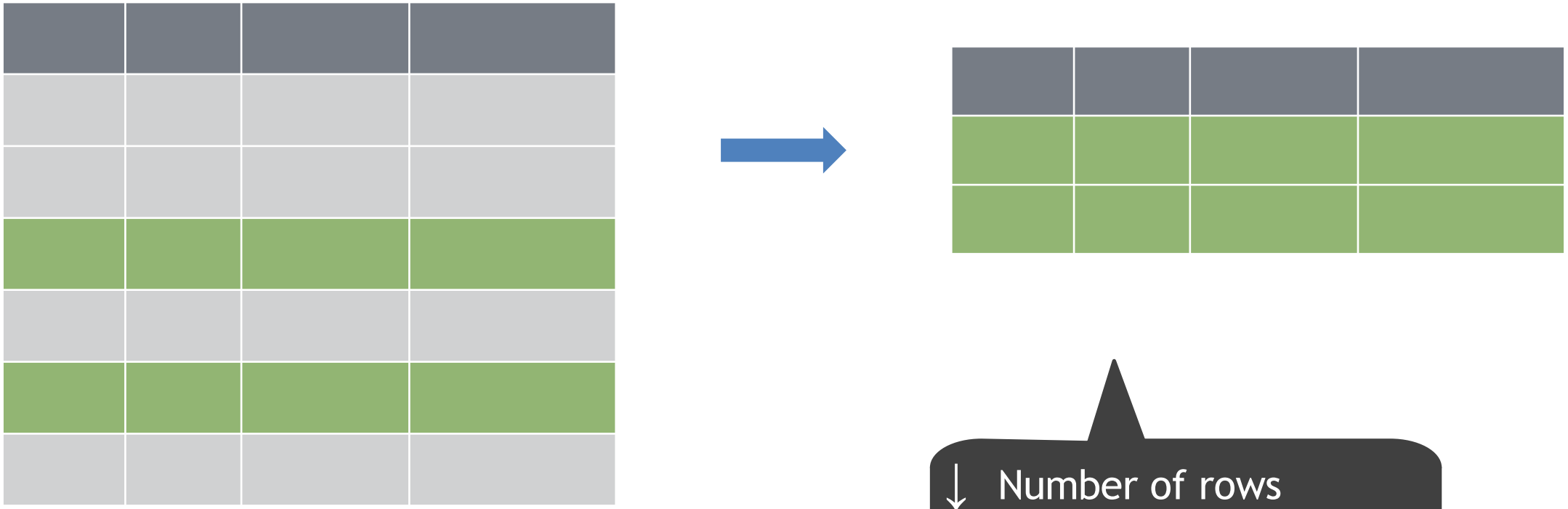
```
function(data, ...)
```



Filtering a Subset of Rows

filter()

Extract rows that meet logical criteria



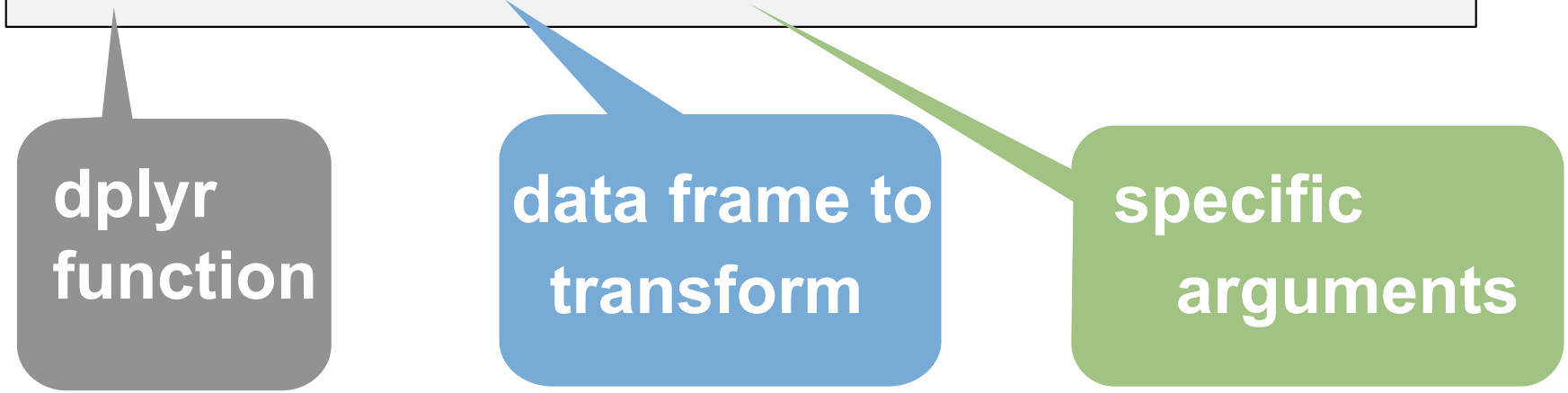
↓ Number of rows

= Number of Columns

Common syntax

Each function takes a data frame as its first argument and returns a data frame as its output.

```
function(data, ...)
```



filter()

Extract rows that meet logical criteria

```
filter(data, ... )
```

data frame to transform

one or more logical tests (filter returns each row for which the test is TRUE)

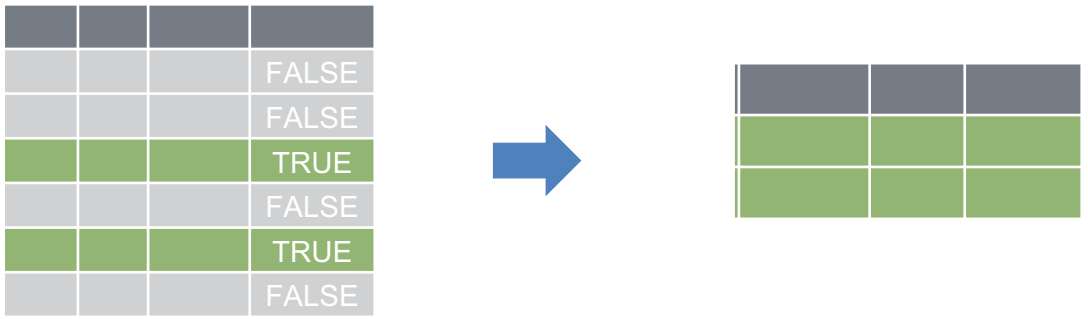
dplyr

filter()

Extract rows that meet logical criteria

```
filter(data, column_name == criteria )
```

one or more logical tests
(filter returns each row for which the test is TRUE)



filter()

Extract rows that meet logical criteria

```
filter(covid_testing, mrn == 5000083)
```

	mrn	first_name	last_name
FALSE	5000876	sarella	stark
FALSE	5006017	alester	stark
FALSE	5001412	jhezane	westerling
TRUE	5000083	lollys	clegane

mrn	first_name	last_name
5000083	lollys	clegane

filter()

Extract rows that meet logical criteria

```
filter(covid_testing, mrn == 5000083)
```

mrn	first_name	last_name
5000876	sarella	stark
5006017	alester	stark
5001412	jhezane	westerling
5000083	lollys	clegane

= sets
(returns nothing)
== tests if equal
(returns TRUE or FALSE)

filter()

Values coded as character strings must be surrounded by quotes

Extract rows that meet logical criteria.

```
filter(covid_testing, last_name == "stark")
```

mrn	first_name	last_name	
5000876	sarella	stark	TRUE
5006017	alester	stark	TRUE
5001412	jhezane	westerling	FALSE
5000083	lollys	clegane	FALSE

→

mrn	first_name	last_name
5000876	sarella	stark
5006017	alester	stark

filter()

Extract rows that meet logical criteria

```
filter(data, ...)
```

data frame to transform

one or more logical tests (filter returns each row for which the test is TRUE)

Logical tests

`x < y`	Less than
`x > y`	Greater than
`x == y`	Equal to
`x <= y`	Less than or equal to
`x >= y`	Greater than or equal to
`x != y`	Not equal to
`x %in% y`	Group membership
`is.na(x)`	Is NA
`!is.na(x)`	Is not NA

Pop Quiz

What is the result?

```
1 == 1
```

Pop Quiz

What is the result?

```
3 != 1
```

Your Turn 2

Use filter() with the logical operators to find:

- Every test for patients **over age 80**
- All of the covid testing where the demographic group (demo_group) is **equal to "client"**

filter()

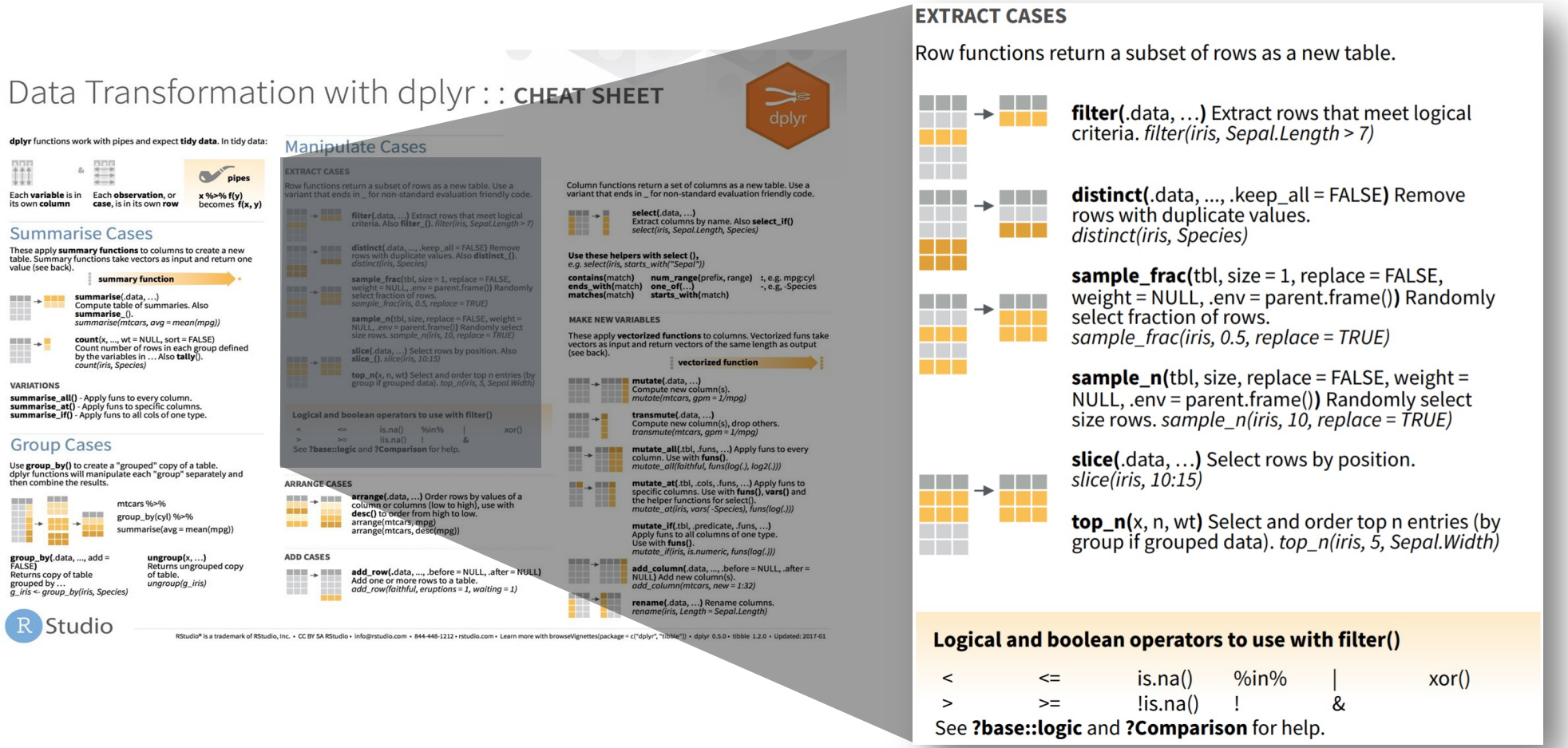
Filter to multiple matches

```
covid_testing %>%
    filter(first_name %in% c("jon","daenerys"))
```

mrn <dbl>	first_name <chr>
5001412	jhezane
5000533	penny
5009134	grunt
5008518	melisandre
5008967	rolley

mrn <dbl>	first_name <chr>
5002427	daenerys
5011120	jon
5001092	jon
5004082	jon
5005197	daenerys

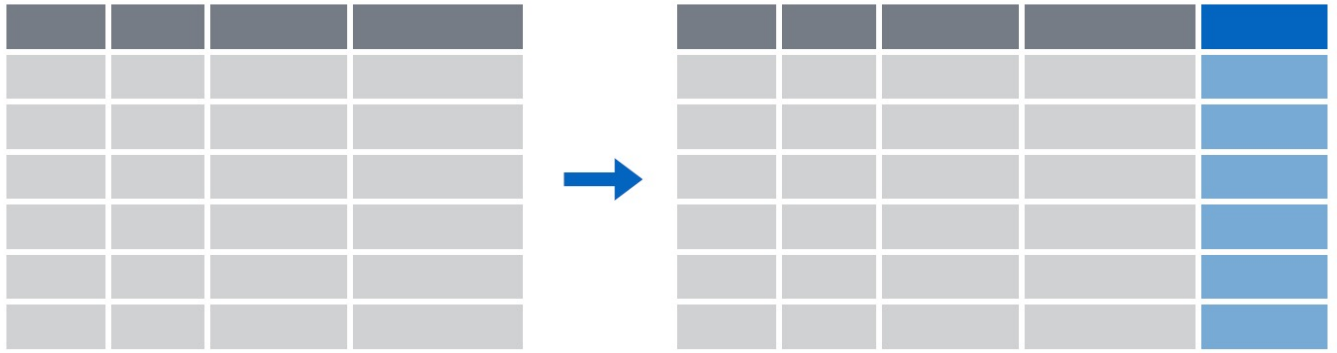
filter() variants



Creating a New Column

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

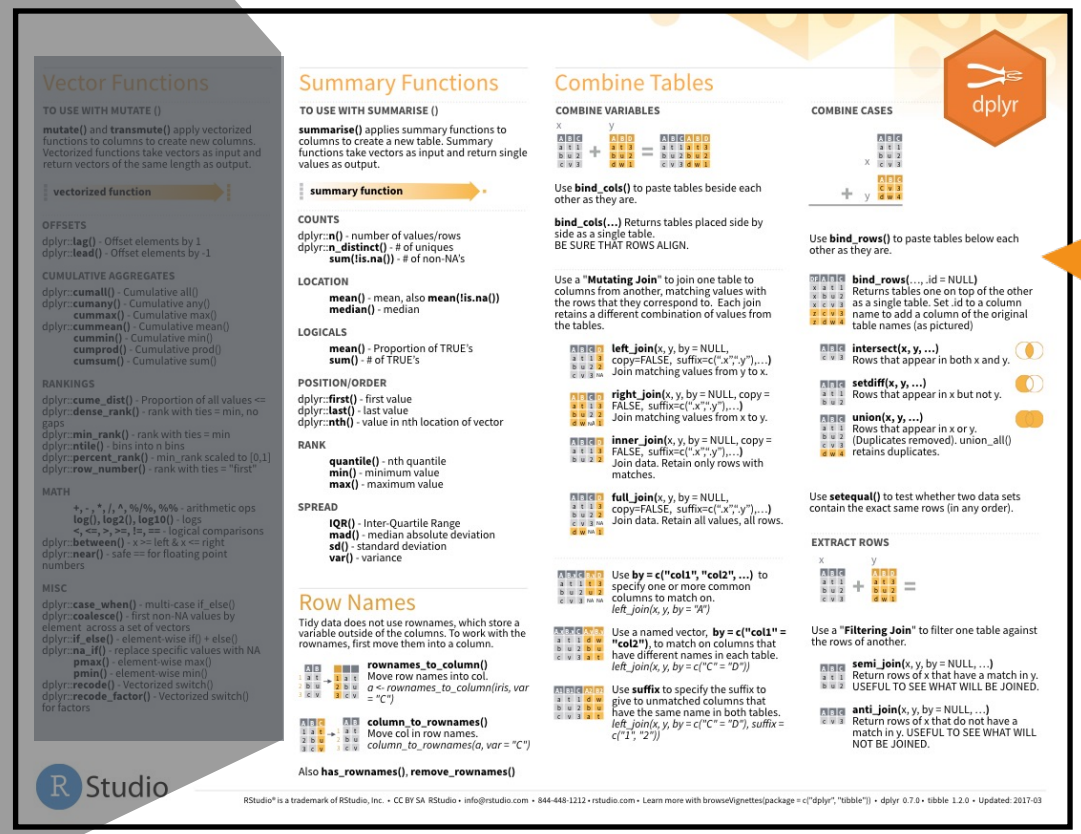
dplyr::**cume_dist()** - Proportion of all values <=
dplyr::**dense_rank()** - rank with ties = min, no gaps
dplyr::**min_rank()** - rank with ties = min
dplyr::**ntile()** - bins into n bins
dplyr::**percent_rank()** - min_rank scaled to [0,1]
dplyr::**row_number()** - rank with ties = "first"

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



on back

dplyr

mutate()

Replacing columns

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

```
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

Piping Data from One Row to the Next

Answering the age old lab question

There have been some complaints from a few inpatient units that turnaround times are longer than expected. You would like to a take a closer look at the tests that took longer than expected. For these inpatient samples turnaround times should rarely be longer than 18 hours, so you would specifically like to filter rows that are from inpatients and have a total turnaround time (collect to verify) of more than 18 hours.

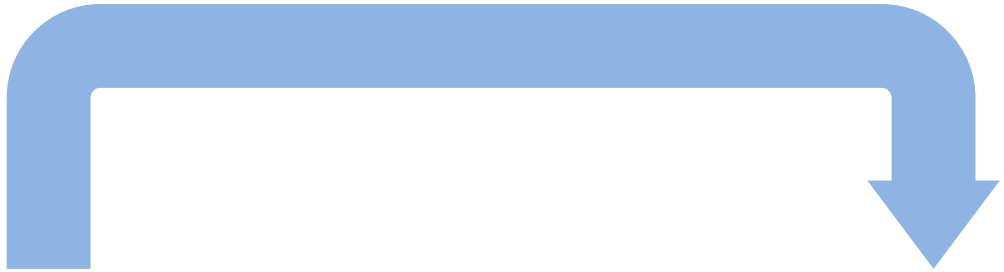
Data Analysis Steps

```
tat_outliers <- filter(covid_testing, patient_class == 'inpatient')

tat_outliers <- mutate(tat_outliers,
                       total_tat = col_rec_tat + rec_ver_tat)

tat_outliers <- filter(tat_outliers, total_tat > 18)
```

1. Filter to extract a subset of inpatient results
2. Mutate to add a column for total turnaround time
3. Filter to select only the results with a total turnaround time greater than 18 hours

The Pipe Operator %>%

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



```
covid_testing  %>% filter(____, patient_class == 'inpatient')
```

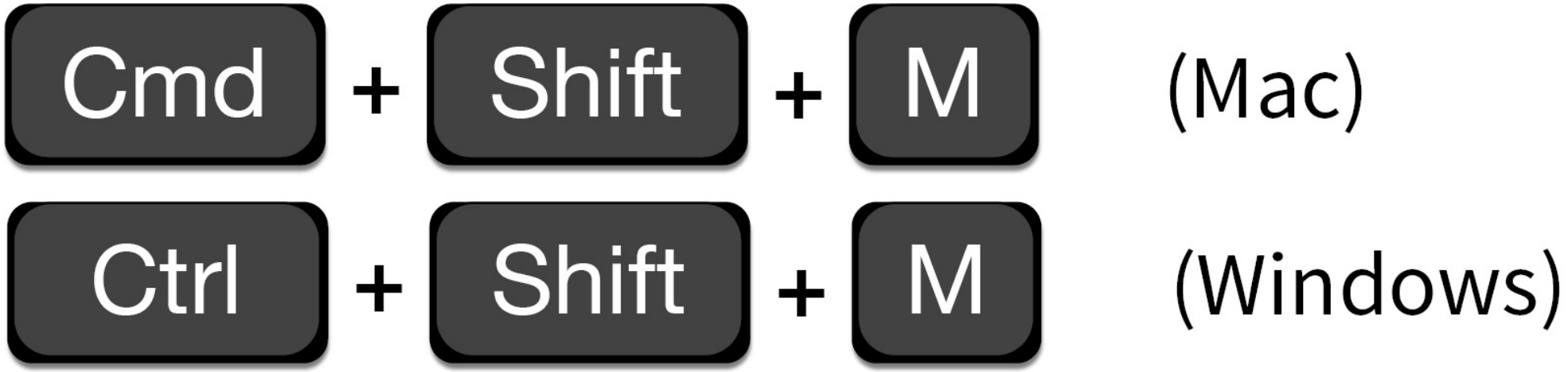
```
filter(covid_testing, patient_class == 'inpatient')
covid_testing %>% filter(patient_class == 'inpatient')
```

Data Analysis Steps

```
tat_outliers <- covid_testing %>%
     filter(patient_class == 'inpatient') %>%
     mutate(total_tat = col_rec_tat + rec_ver_tat) %>%
     filter(total_tat > 18)
```

1. Filter to extract a subset of inpatient results
2. Mutate to add a column for total turnaround time
3. Filter to select only the results with a total turnaround time greater than 18 hours

Shortcut to type %>%

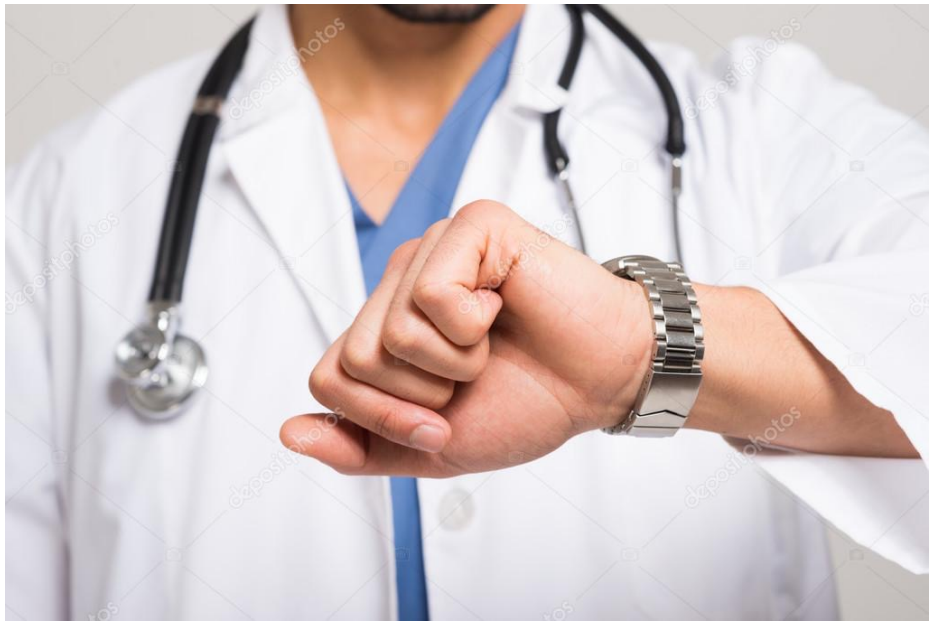


Scene

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 4

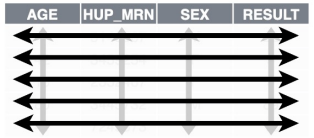
Use %>% to write a sequence of three functions that:

1. Filters to tests from the clinic (**`clinic_name`**) of "picu"
2. Adds a column with the total turnaround time (sum of **`col_ver_tat`** and **`rec_ver_tat`**)
3. Filters to include only the turnaround times greater than 10 hours

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

Recap

dplyr is a package that provides a **grammar of data manipulation**. Common operations include extracting a subset of your data set and adding new columns.



1. Fully utilizing **dplyr** requires a **tidy data frame**, in which each **variable** is in its own **column**, each **observation** is in its own **row**, each **value** is in its own **cell**;

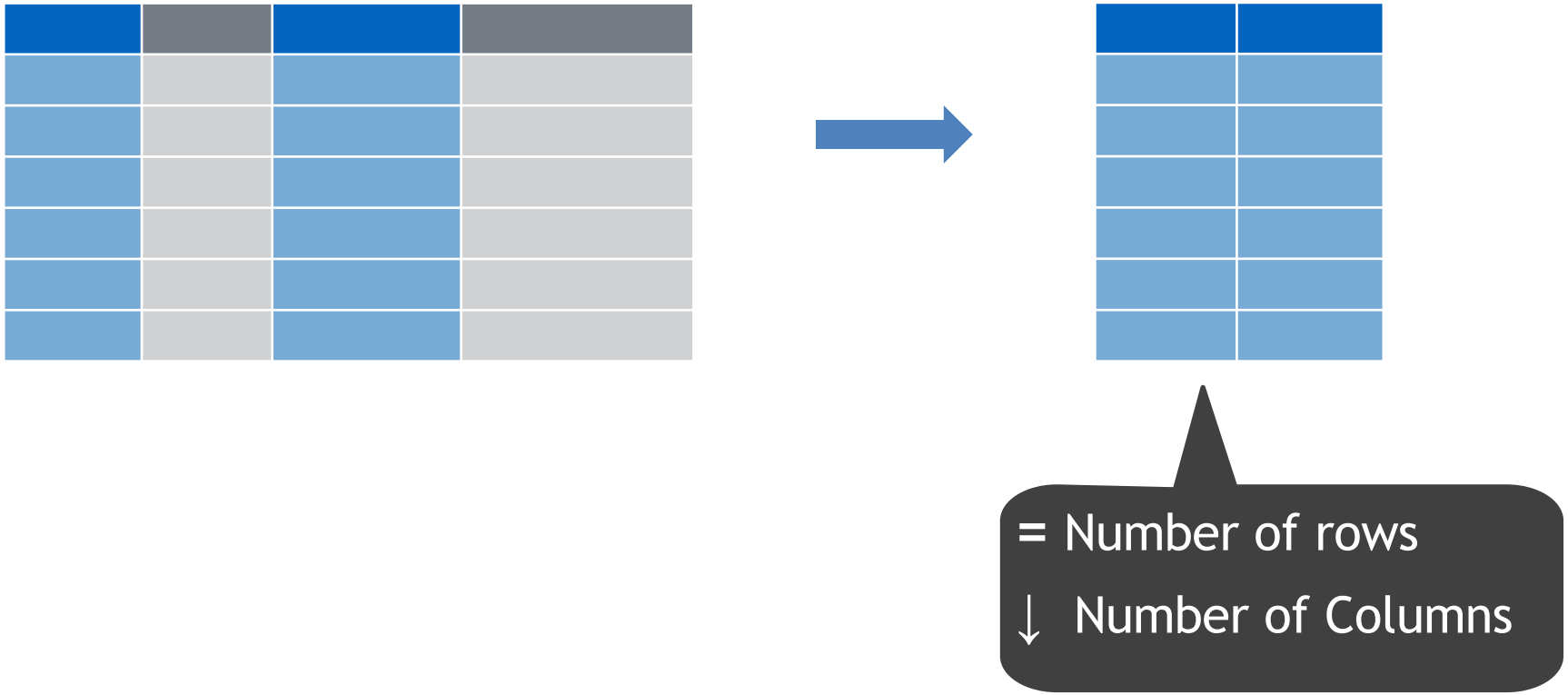
2. The **filter** function allows you to extract a subset of your data using logical conditions

3. The **mutate** function allows you to add new columns to your data frame that can apply calculations across columns in your data frame

What Else?

select()

Extract columns from a data frame



select()

Extract columns from a data frame

```
select(covid_testing, mrn, last_name)
```

dplyr function

data frame to transform

name(s) of columns to extract
(or a select helper)

select()

Extract columns from a data frame **by name**

```
select(covid_testing, mrn, last_name)
```

covid_testing

mrn	first_name	last_name	gender
5000876	sarella	stark	female
5006017	alester	stark	male
5001412	jhezane	westerling	female
5000533	penny	targaryen	female
...			

mrn	last_name
5000876	stark
5006017	stark
5001412	westerling
5000533	targaryen

select()

Extract columns from a data frame **by name**

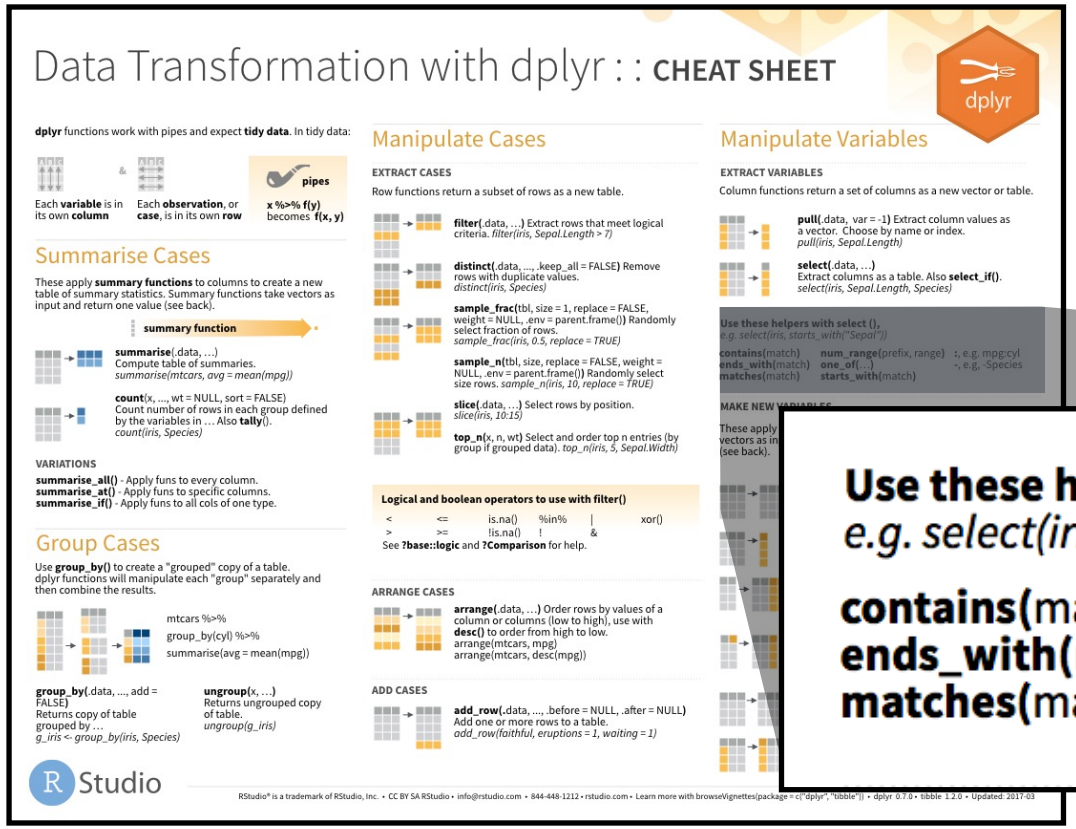
```
select(covid_testing, -mrn, -last_name)
```

covid_testing

mrn	first_name	last_name	gender
5000876	sarella	stark	female
5006017	alester	stark	male
5001412	jhezane	westerling	female
5000533	penny	targaryen	female
...			

first_name	gender
sarella	female
alester	male
jhezane	female
penny	female

select() helpers



Data Transformation with dplyr : : **CHEAT SHEET**

dplyr

dplyr functions work with pipes and expect **tidy data**. In tidy data:

Each **variable** is in its own **column**. Each **observation**, or **case**, is in its own **row**.

pipes: **x %>% f(y)** becomes **f(x, y)**

Summarise Cases

These apply **summary functions** to columns to create a new table of summary statistics. Summary functions take vectors as input and return one value (see back).

summary function

summarise(.data, …) Compute table of summaries. *summarise(mtcars, avg = mean(mpg))*

count(x, …, wt = NULL, sort = FALSE) Count number of rows in each group defined by the variables in … Also **tally**(). *count(iris, Species)*

VARIATIONS

summarise_all() - Apply funs to every column.
summarise_at() - Apply funs to specific columns.
summarise_if() - Apply funs to all cols of one type.

Group Cases

Use **group_by()** to create a "grouped" copy of a table. dplyr functions will manipulate each "group" separately and then combine the results.

mtcars %>%
group_by(cyl) %>%
summarise(avg = mean(mpg))

group_by(.data, …, add = FALSE) Returns copy of table grouped by … *g_iris <- group_by(iris, Species)*

ungroup(x, …) Returns ungrouped copy of table. *ungroup(g_iris)*

Manipulate Cases

EXTRACT CASES

Row functions return a subset of rows as a new table.

filter(.data, …) Extract rows that meet logical criteria. *filter(iris, Sepal.Length > 7)*

distinct(.data, …, .keep_all = FALSE) Remove rows with duplicate values. *distinct(iris, Species)*

sample_frac(tbl, size = 1, replace = FALSE, weight = NULL, .env = parent.frame()) Randomly select fraction of rows. *sample_frac(iris, 0.5, replace = TRUE)*

sample_n(tbl, size, replace = FALSE, weight = NULL, .env = parent.frame()) Randomly select size rows. *sample_n(iris, 10, replace = TRUE)*

slice(.data, …) Select rows by position. *slice(iris, 10:15)*

top_n(x, n, wt) Select and order top n entries (by group if grouped data). *top_n(iris, 5, Sepal.Width)*

Logical and boolean operators to use with filter()

<	<=	is.na()	%in%	| xor()
>	>=	!is.na()	!	&

See **?base::logic** and **?Comparison** for help.

ARRANGE CASES

arrange(.data, …) Order rows by values of a column or columns (low to high), use with **desc()** to order from high to low. arrange(mtcars, mpg) arrange(mtcars, desc(mpg))

ADD CASES

add_row(.data, …, .before = NULL, .after = NULL) Add one or more rows to a table. *add_row(faithful, eruptions = 1, waiting = 1)*

Manipulate Variables

EXTRACT VARIABLES

Column functions return a set of columns as a new vector or table.

pull(.data, var = -1) Extract column values as a vector. Choose by name or index. *pull(iris, Sepal.Length)*

select(.data, …) Extract columns as a table. Also **select_if()**. *select(iris, Sepal.Length, Species)*

Use these helpers with select (),
e.g. select(iris, starts_with("Sepal"))

contains(match) **num_range**(prefix, range) **:**, e.g. mpg:cyl
ends_with(match) **one_of**(…) **-**, e.g, -Species
matches(match) **starts_with**(match)

MAKE NEW VARIABLES

These apply … vectors as in … (see back).

RStudio

Use these helpers with select (),
e.g. select(iris, starts_with("Sepal"))

contains(match) **num_range**(prefix, range) **:**, e.g. mpg:cyl
ends_with(match) **one_of**(…) **-**, e.g, -Species
matches(match) **starts_with**(match)

dplyr

select()

Renaming columns

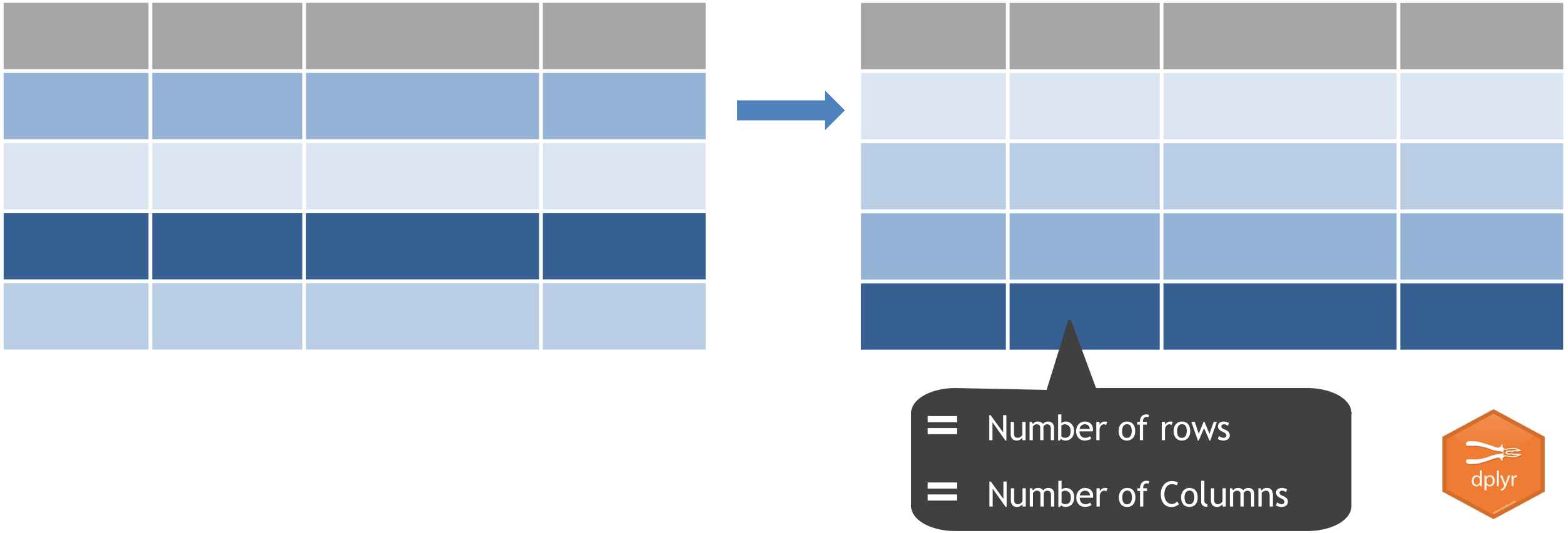
```
covid_testing %>%
    select(MRN = mrn, first_name, last_name)
```

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

arrange()

Order rows by values in a column



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

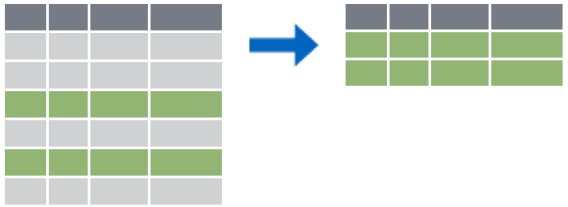
Pop Quiz

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

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

Transforming data

Extract variables with **select()**



Extract rows with **filter()**

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

Add calculated columns with **mutate()**

Example: Applying the pipe

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

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. Extract columns meeting inclusion criteria from a data frame
3. Create new calculated columns not found in the original data frame
4. Use the pipe operator to pass the output of one function as an input to the next function