



Data Transformation

Session 4
Amrom Obstfeld

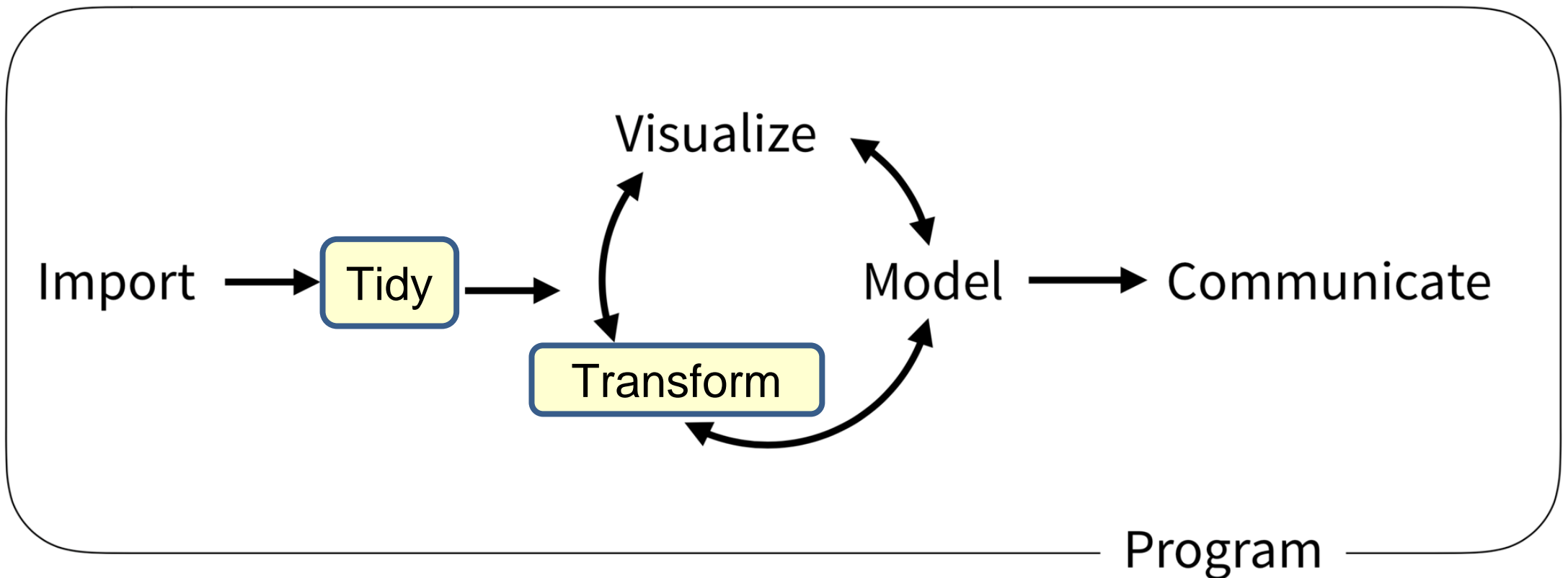
Goal

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 code templates with dplyr functions to tidy a raw data set
3. Use the pipe operator to pass the output of one function as an input to the next function
4. Create new calculated columns not found in the original data frame

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**

AGE	MRN	SEX	RESULT
45	0130304	F	0
55	0130304	F	0
25	0130304	F	0
35	0130302	M	0
124	0130302	M	0

Your Turn 1

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

Run the setup chunk

```
```{r setup}
library(tidyverse) # Provides functions used throughout this session
covid_testing <- read_csv("covid_testing.csv")
```
```



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* for transforming tabular data.

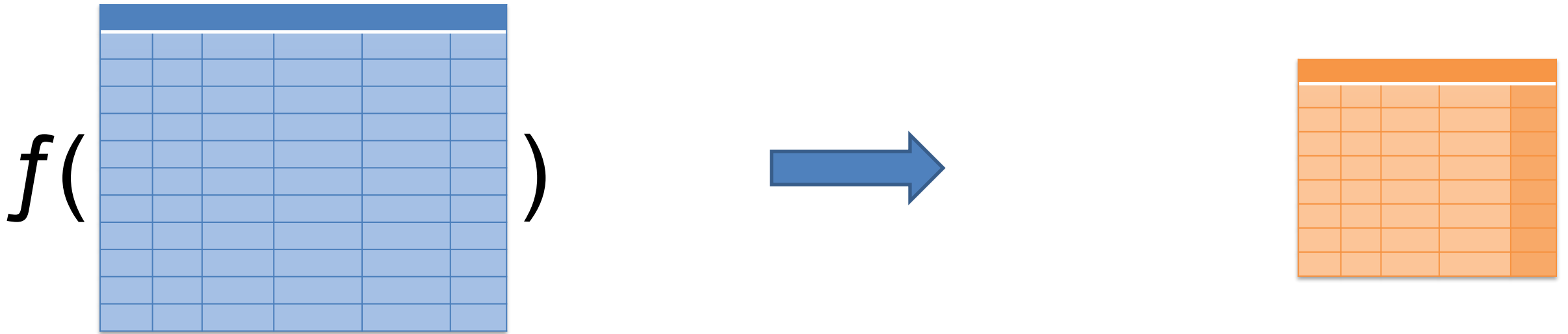


dplyr: a grammar for transforming data

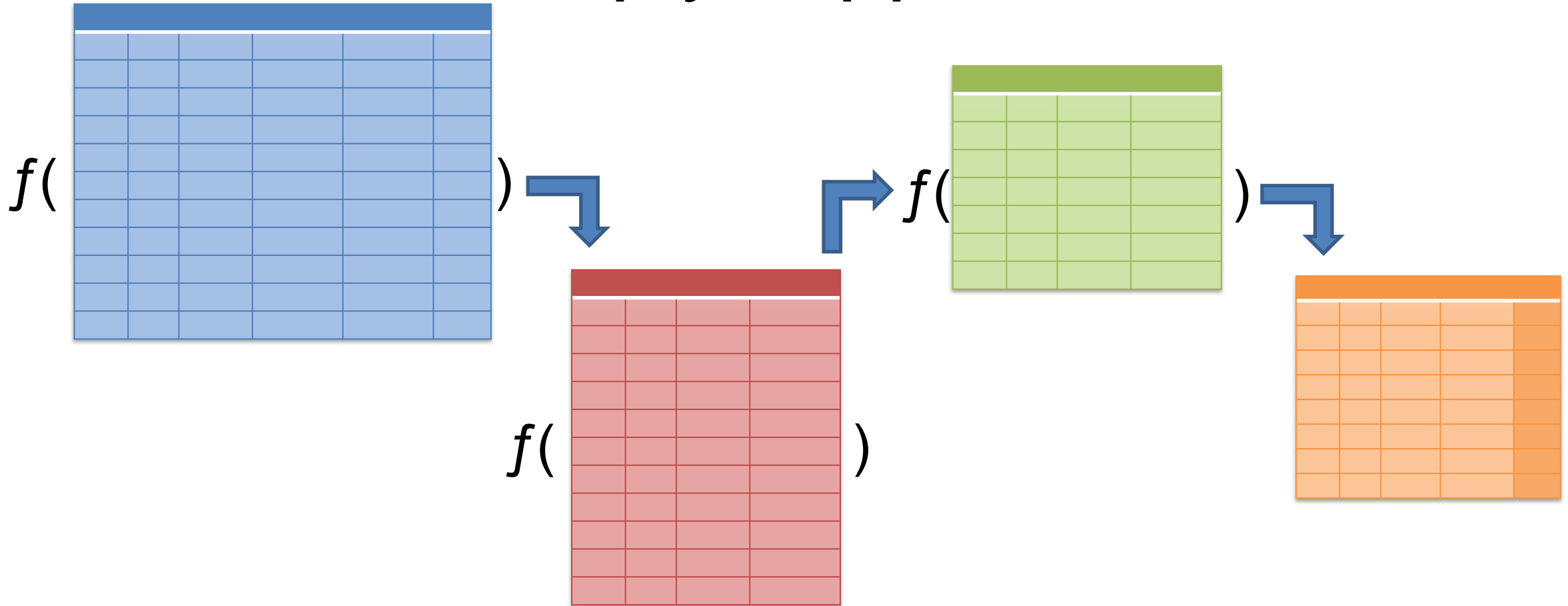
- 1 Choose** columns. `select()`
- 2 Extract** rows. `filter()`
- 3 Derive** new columns. `mutate()`
- 4 Change** the unit of analysis. `summarize()`



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, ...)
```

**dplyr
function**

**data frame to
transform**

**specific
arguments**



Isolating data



select()

select()

Extract columns from a data frame



= Number of rows
↓ Number of Columns

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

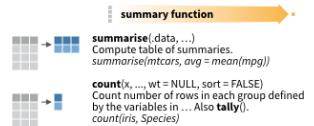


dplyr functions work with pipes and expect tidy data. In tidy data:



Summarise Cases

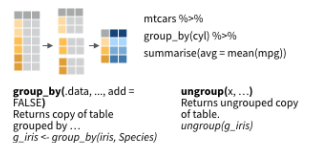
These apply **summary functions** to columns to create a new table of summary statistics. Summary functions take vectors as input and return one value (see back).



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

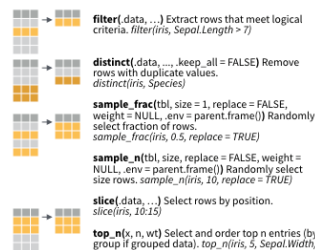
Use **group_by()** to create a "grouped" copy of a table. dplyr functions will manipulate each "group" separately and then combine the results.



Manipulate Cases

EXTRACT CASES

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



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))

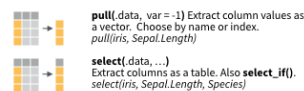
ADD CASES

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.



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).

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

contains(match)
ends_with(match)
matches(match)

num_range(prefix, range) ; e.g. mpg:cyl
one_of(...)
starts_with(match) -, e.g. -Species



RStudio® is a trademark of RStudio, Inc. • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more with browseVignettes(package = "dplyr", "tidyverse") • dplyr 0.7.0 • tidyverse 1.2.0 • Updated: 2017-03



Your Turn 2

- Alter the code to select just the `first_name` column from `covid_testing`
- Use the second code chunk to see if you can remove the `first_name` column

```
covid_testing_2 <- select(covid_testing, _____)
```

filter()

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, ...)
```

dplyr
function

data frame to
transform

specific
arguments

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)

| | | | FALSE |
|--|--|--|-------|
| | | | FALSE |
| | | | TRUE |
| | | | FALSE |
| | | | TRUE |
| | | | FALSE |



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)

| | | | FALSE |
|--|--|--|-------|
| | | | FALSE |
| | | | TRUE |
| | | | FALSE |
| | | | TRUE |
| | | | FALSE |



filter()

Extract rows that meet logical criteria

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

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

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

Pop Quiz

What is the result?

`1 == 1`

Pop Quiz

What is the result?

$3 \neq 1$

filter() variants

Data Transformation with dplyr : : CHEAT SHEET

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
x %>% f(y) becomes f(x, y)

Summarise Cases

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

summary function
→ **summarise(data, ...)**
Compute table of summaries. Also **summarise_at()**, **summarise_if()**
→ **count(x, ...)**
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, ...)
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. Use a variant that ends in **_()** for non-standard evaluation friendly code.

→ **filter(data, ...)** Extract rows that meet logical criteria. Also **filter_()**, **filter_if()**, **filter_at()**

→ **distinct(data, ...)** Remove rows with duplicate values. Also **distinct_()**, **distinct_if()**

→ **sample_frac(tbl, size = 1, replace = FALSE, weight = NULL, env = parent.frame())** Randomly select fraction of rows. Also **sample_n()**

→ **sample_n(tbl, size = 1, replace = FALSE, weight = NULL, env = parent.frame())** Randomly select size rows. Also **sample_n()**

→ **slice(data, ...)** Select rows by position. Also **slice_()**, **slice_if()**

→ **top_n(x, n, wt)** Select and order top n entries (by group if grouped data). Also **top_n()**

Logical and boolean operators to use with filter()

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

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(mtcars, new = 1:32)**

Column functions return a set of columns as a new table. Use a variant that ends in **_()** for non-standard evaluation friendly code.

→ **select(data, ...)** Extract columns by name. Also **select_if()**, **select_at()**

→ **contains(match)**, **ends_with(match)**, **one_of(match)**, **starts_with(match)**

→ **num_range(prefix, range)** z, e.g. mpg:cyl
→ **starts_with(match)** z, e.g. Species

→ **MAKE NEW VARIABLES**
These apply **vectorized functions** to columns. Vectorized funs take vectors as input and return vectors of the same length as output (see back).

→ **mutate(data, ...)** Compute new column(s). Also **mutate_at()**, **mutate_if()**

→ **transmute(data, ...)** Compute new column(s), drop others. Also **transmute_at()**, **transmute_if()**

→ **mutate_all(tbl, funs, ...)** Apply funs to every column. Use with **funs()**

→ **mutate_at(tbl, cols, funs, ...)** Apply funs to specific columns. Use with **funs()**, **vars()** and the helper functions for **select()**

→ **mutate_if(tbl, predicate, funs, ...)** Apply funs to all columns of one type. Use with **funs()**

→ **add_column(data, ...)** before = NULL, after = NULL
Add new column(s). Also **add_column_at()**, **add_column_after()**

→ **rename(data, ...)** Rename columns. Also **rename_at()**, **rename_if()**

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.

Your Turn 3

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"**

arrange()

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 4

The column `ct_value` contains the cycle threshold (Ct) 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"?

$\% > \%$

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)
```

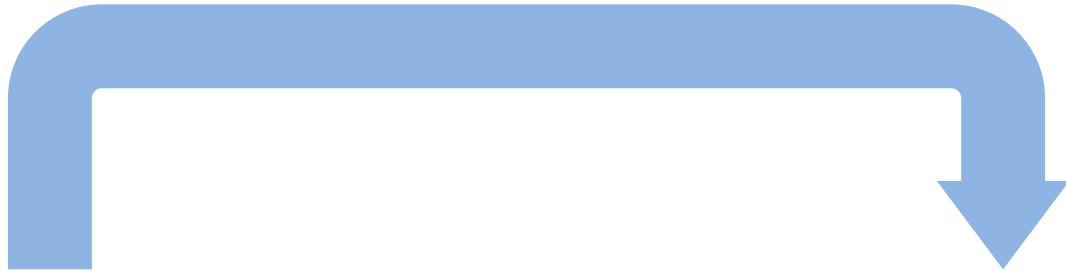
1. Filter tests to those on pandemic day less than 10
2. Select the column that contains ordering location
3. Arrange those columns by location

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(panday <= 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 5

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

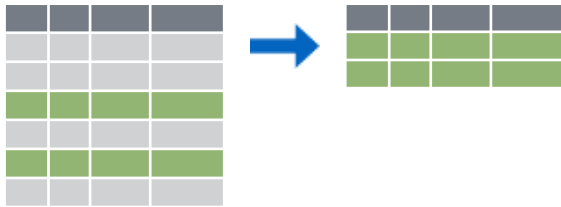
1. Filters to tests from the clinic (**clinic_name**) of "picu"
2. Selects the column with the receive to verify turnaround time (**rec_ver_tat**) as well as the day from start of the pandemic (**pan_day**)
3. Arrange the **pan_day** from highest to lowest

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()**.



Deriving Data



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

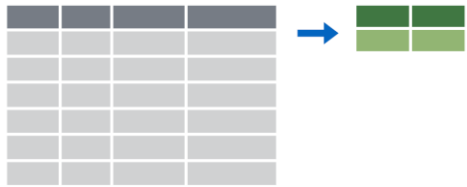
Breaking down the analytical question

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 6

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()
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

bind_cols() to paste tables beside each other as they are.

bind_rows() 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.

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).

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. List the major forms of data transformation implemented in dplyr
2. Use code templates containing dplyr functions to tidy a raw data set
3. Use the pipe operator to pass the output of one function as an input to the next function
4. Create new calculated columns not found in the original data frame



Dplyr tips and tricks



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 |

filter()

Filter to multiple matches

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

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



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

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> | | mrn
<chr> | first_name
<chr> | last_name
<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 |

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 |