



# **Grouping and Summarizing Data**

Patrick Mathias

Lesson 6

DLMP Fall 2021

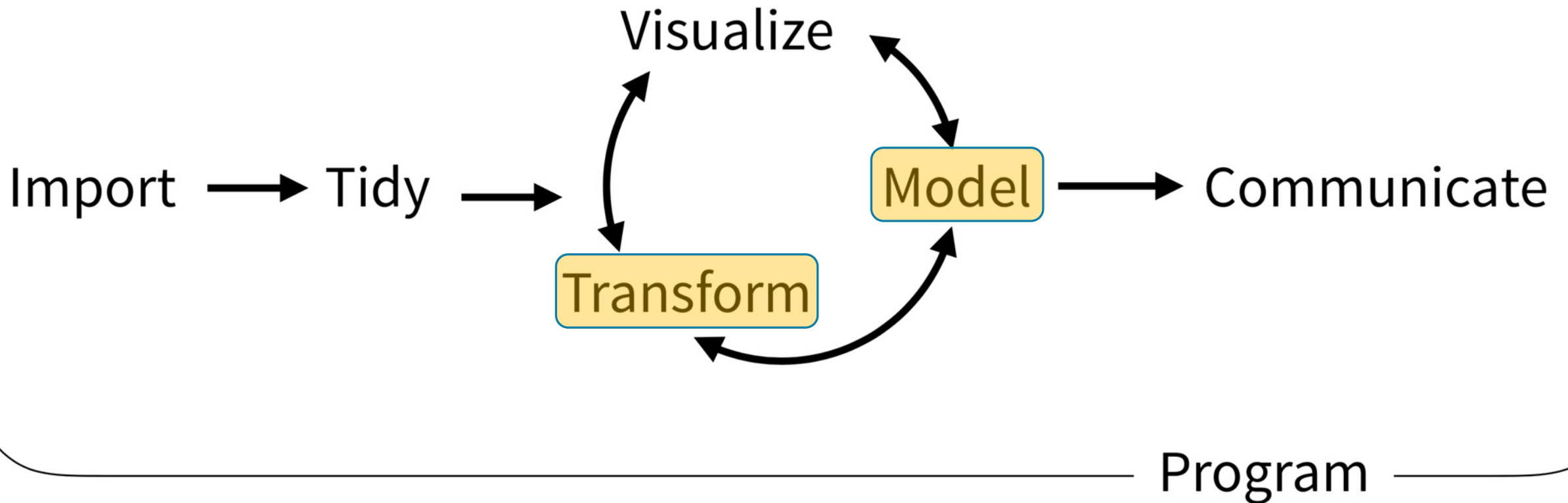
# Goals

1. Learn dplyr tools for grouping and summarizing data in R

# Objectives

1. Calculate a summary statistic for a variable using the `summarize()` function
2. Creates groupings of data using the `group_by()` function
3. Combine `group_by()` and `summarize()` functions to calculate summary statistics for groups of data

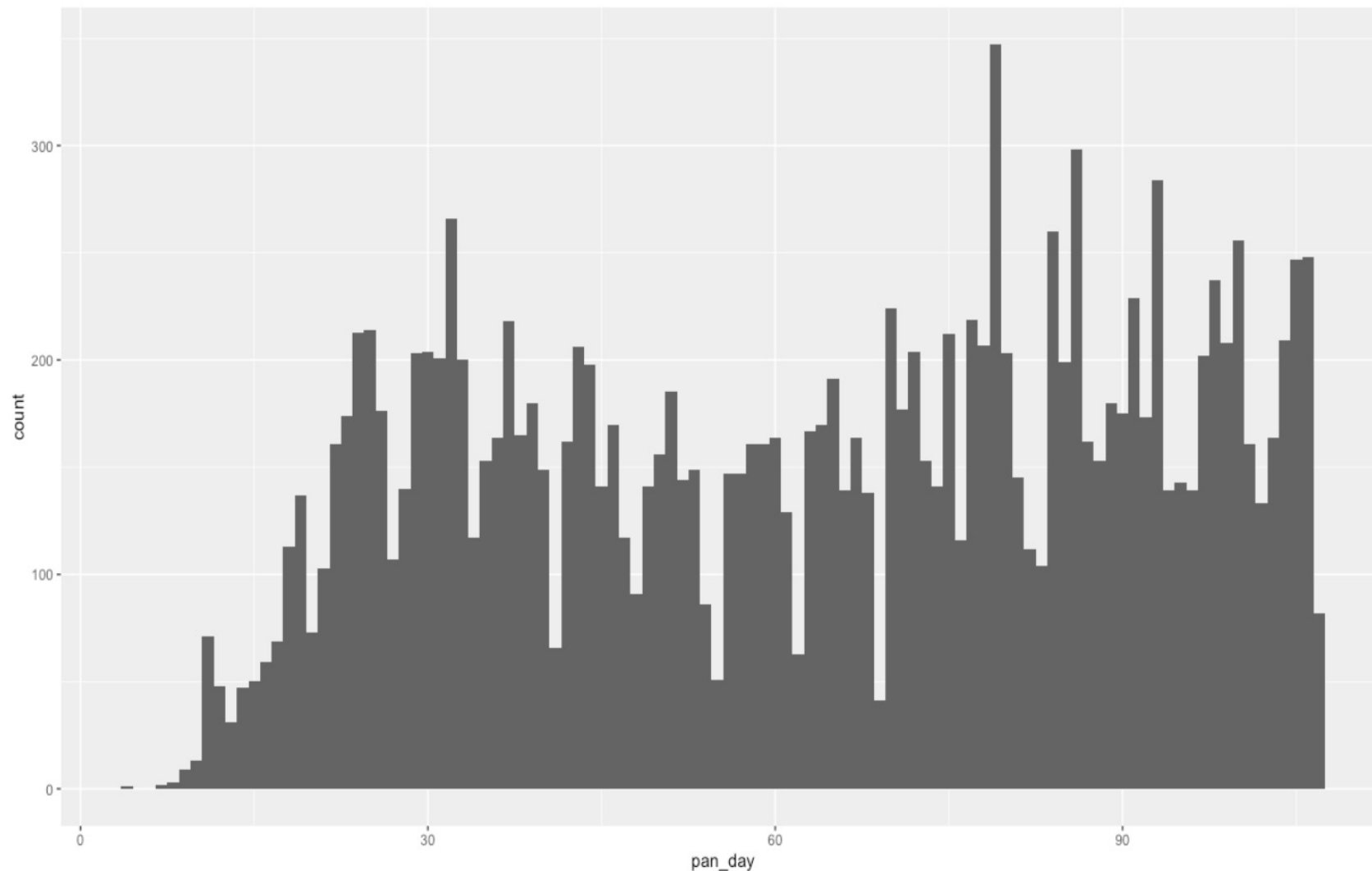
# Typical Data Science Pipeline





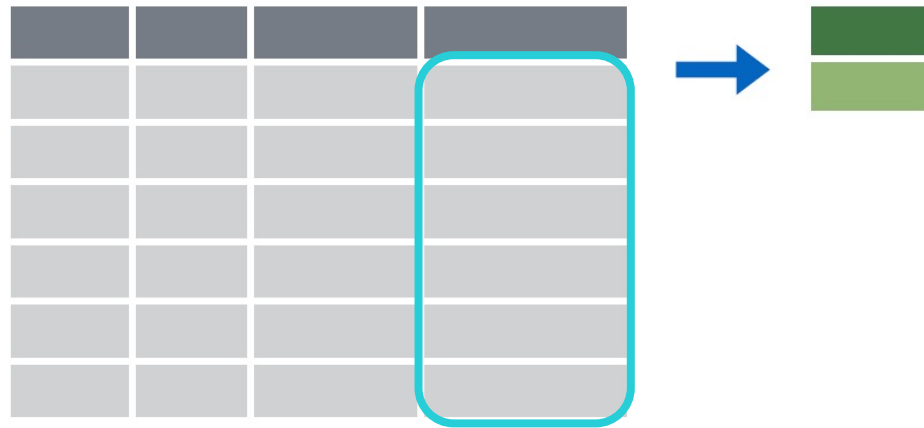
**Summarize the data set**

**Q: How many tests are ordered per day?**



# summarize()

- Make summaries of your data



# summarize()

- Make summaries of your data

```
covid_testing %>%  
  summarize(new_variable = calculation)
```

name for new  
variable

Value or  
function

Performs calculation across all rows of data frame



# summarize()

- Make summaries of your data

```
covid_testing %>%  
  select(mrn, pan_day) %>%  
  head(4) %>%  
  summarize(order_count = n())
```

function that returns  
number of observations

mrn	pan_day
5001412	4
5000533	7
5009134	7
5008518	8



order_count
4



# summarize()

- Additional summaries = new columns

```
covid_testing %>%  
  select(mrn, pan_day) %>%  
  head(4) %>%  
  summarize(order_count = n(),  
            day_count = n_distinct(pan_day))
```

function that returns  
number of distinct values

mrn	pan_day
5001412	4
5000533	7
5009134	7
5008518	8



order_count	day_count
4	3

# Your Turn #1

- Open “05 – Group by and Summarize.Rmd”
- Run the setup chunk
- Fill-in gaps to calculate:
  - a) Mean count of orders per `pan\_day`
  - b) Mean count of orders per clinic

05:00

## Vector Functions

TO USE WITH MUTATE ()

### 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

## 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

x + y =

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 the column(s) to match on.  
`left_join(x, y, by = "A")`

Use a named vector, `by = c("col1" = "col2")`, to match on columns with different names in each data set.  
`left_join(x, y, by = c("C" = "D"))`

Use `suffix` to specify suffix to give to duplicate column names.  
`left_join(x, y, by = c("C" = "D"), suffix = c("1", "2"))`

COMBINE CASES

x + y =

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

x + y =

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.



# Output the last day

```
covid_testing %>%  
  summarize(last_day = last(pan_day))
```

mrn	pan_day
5001412	4
5000533	7
5009134	7
5008518	8



last_day
8

# Calculate the median turnaround time

```
covid_testing %>%  
  mutate(col_ver_tat = col_rec_tat + rec_ver_tat) %>%  
  summarize(col_ver_tat_mean = mean(col_ver_tat))
```

mrn	pan_day	col_ver_tat
5001412	4	6
5000533	7	8
5009134	7	10
5008518	8	11



col_ver_tat_mean
8.75

# Your Turn #2

For the covid\_testing data frame, calculate the median collect-to-verify turnaround time.

03:00

# Pop Quiz

How would you calculate the median number of orders per day?

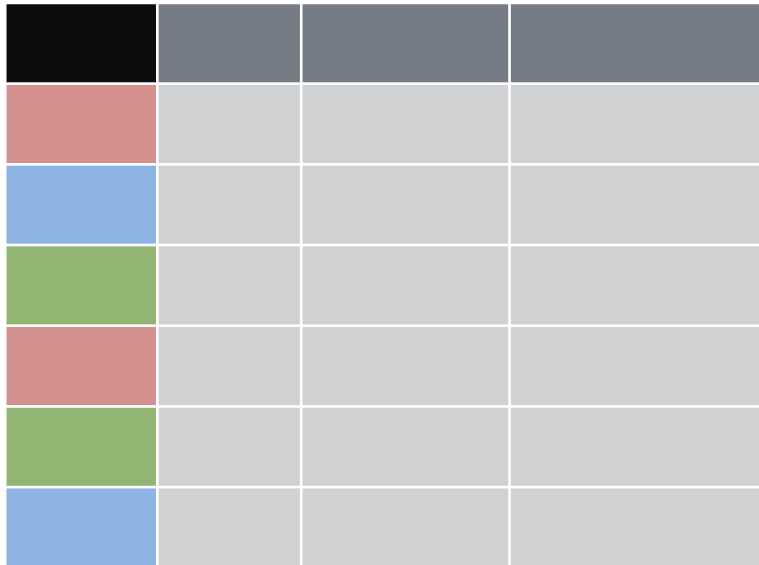


# Grouping your data





# group\_by()



# group\_by()

- *Grouping observations based on a specific **variable**'s values*

```
covid_testing %>%  
  group_by(variable)
```

name of variable  
to group by

# group\_by()

- Group observations by *pan\_day*

```
covid_testing %>%  
  group_by(pan_day)
```

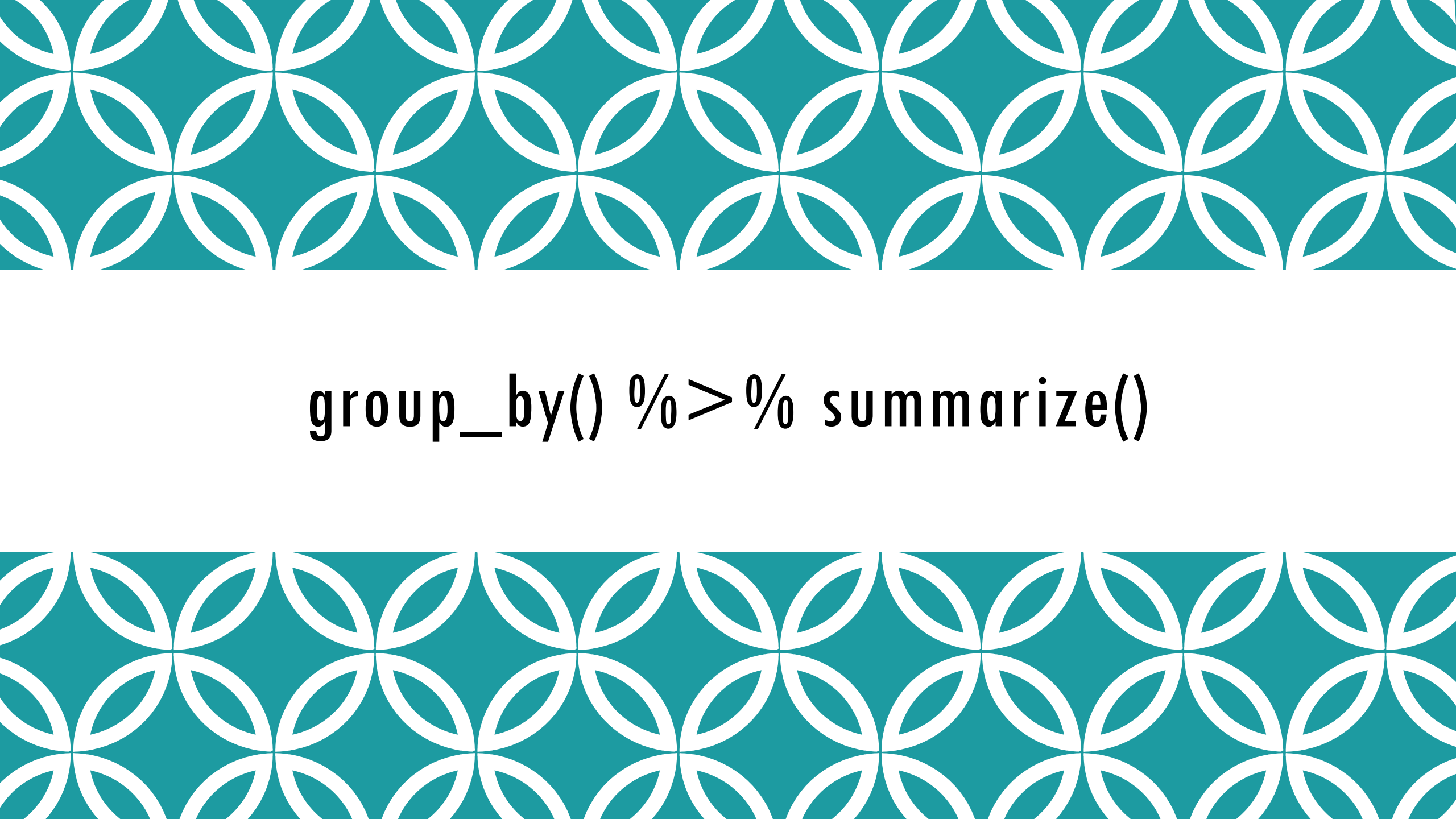
```
# A tibble: 15,524 x 17  
# Groups:   pan_day [102]  
   mrn first_name last_name gender pan_day  
   <dbl> <chr>      <chr>    <chr>    <dbl>  
1 5.00e6 jhezane    westerli... female     4  
2 5.00e6 penny      targaryen female     7  
3 5.01e6 grunt      rivers    male      7  
4 5.01e6 melisandre swyft     female     8  
5 5.01e6 roley      karstark  male      8
```

# group\_by()

- Group observations by `pan_day` and `clinic_name`

```
covid_testing %>%  
  select(mrn, pan_day, clinic_name) %>%  
  group_by(pan_day, clinic_name)
```

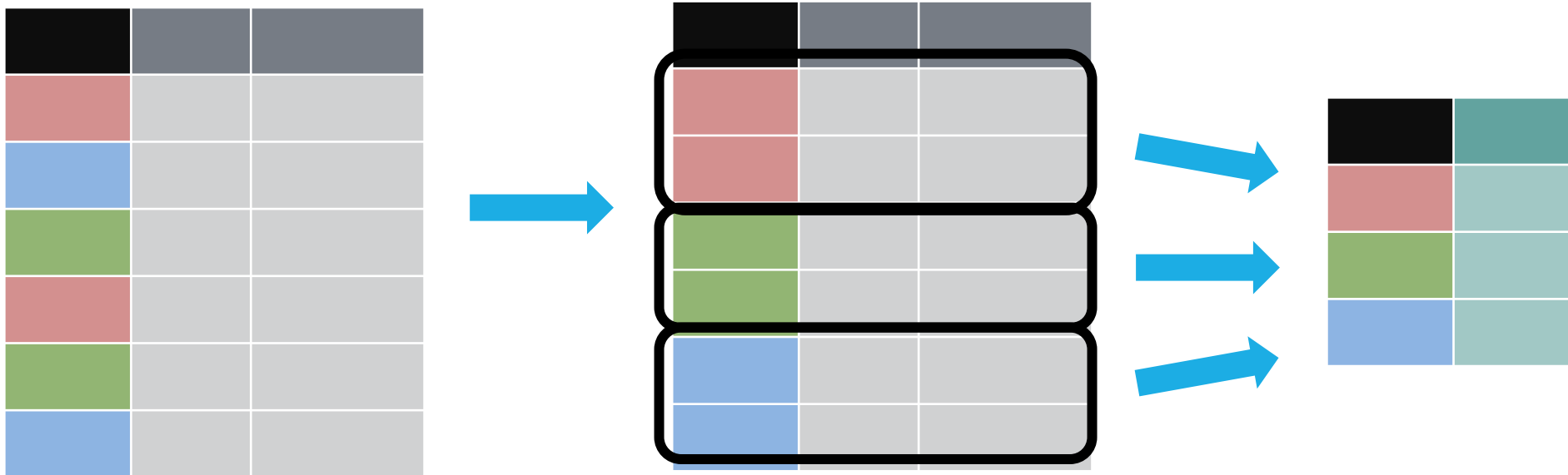
```
# A tibble: 15,524 x 3  
# Groups:   pan_day, clinic_name [2,526]  
  mrn pan_day clinic_name  
  <dbl> <dbl> <chr>  
1 5001412      4 inpatient ward a  
2 5000533      7 clinical lab  
3 5009134      7 clinical lab  
4 5008518      8 clinical lab  
5 5008967      8 emergency dept
```



```
group_by() %>% summarize()
```

# `group_by()` %>% `summarize()`

Make summaries of your data *by group*



# group\_by() %>% summarize()

- Make summaries of your data

```
covid_testing %>%
```

```
  summarize(order_count = n())
```

mrn	pan_day
5001412	4
5000533	7
5009134	7
5008518	8



order_count
15524

# group\_by() %>% summarize()

- Make summaries of your data

```
covid_testing %>%  
  group_by(pan_day) %>%  
  summarize(order_count = n())
```

mrn	pan_day
5001412	4
5000533	7
5009134	7
5008518	8



pan_day	order_count
4	1
7	2
8	3
9	9





# Your Turn #3

Calculate:

- a) The median turnaround time for each day
- b) The median number of orders per day

05:00

# Calculate the 95th percentile of turnaround time

```
covid_testing %>%  
  group_by(clinic_name) %>%  
  summarize(tat_95th_pctile = quantile(rec_ver_tat, 0.95))
```

clinic_name	rec_ver_tat
inpatient ward a	4.7
clinical lab	6.4
picu	5.6
emergency dept	7.1



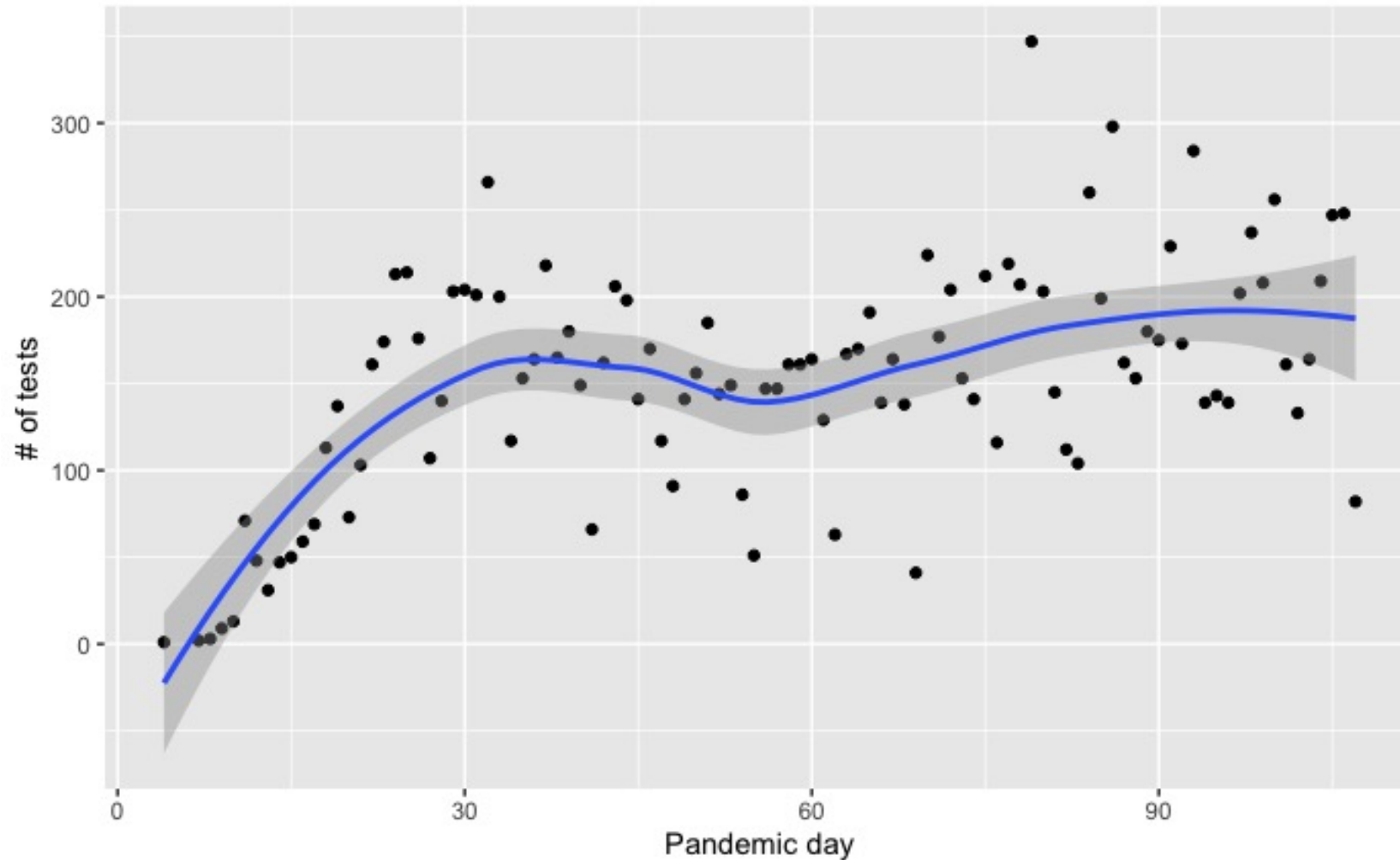
clinic_name	tat_95 <sup>th</sup> _pctile
clinical lab	8.9
emergency dept	9.6
inpatient ward a	9.3
picu	9.1

# Count tests per day, then visualize

```
tmp <- covid_testing %>%  
  group_by(pan_day) %>%  
  summarize(n_tests = n())
```

```
ggplot(data = tmp) +  
  geom_point(aes(x = pan_day, y = n_tests)) +  
  geom_smooth(aes(x = pan_day, y = n_tests), method = 'loess') +  
  ylab("# of tests") +  
  xlab("Pandemic day")
```

# `group_by() %>% summarize(): Example`



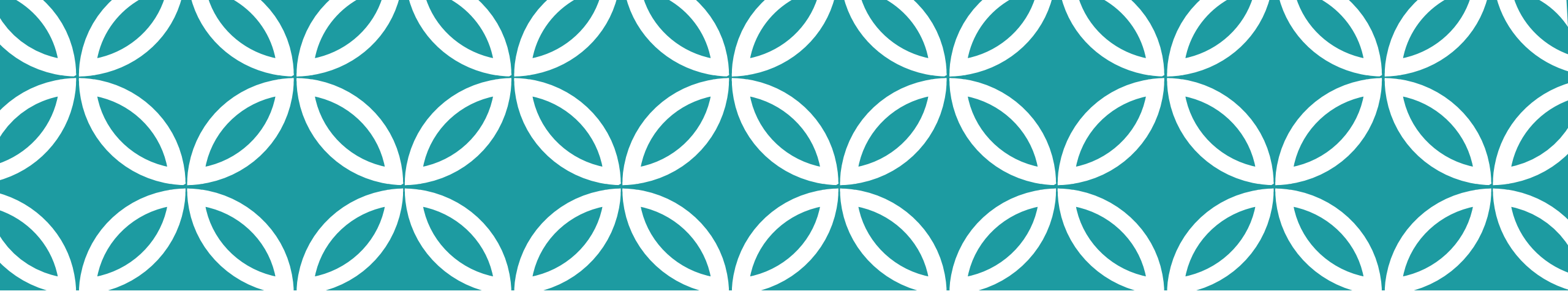
# Recap

**Summarize()** is a function that enables us to calculate summaries of variables (columns).

Common summary activities include counting observations using **n()**, counting unique observations using **n\_distinct()**, and calculating means using **mean()**.

**Group\_by()** is a function that enables us to create subsets of data by a variable. Data can also be grouped by multiple variables.

Combining the **group\_by()** and **summarize()** functions is a powerful way to look at summarizations across groups.



What else?



# 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**



pipes

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

## Summarise Cases

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



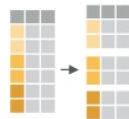
**summarise(.data, ...)**  
Compute table of summaries.  
`summarise(mtcars, avg = mean(mpg))`



**count(.data, ..., wt = NULL, sort = FALSE, name = NULL)** Count number of rows in each group defined by the variables in ... Also **tally()**.  
`count(mtcars, cyl)`

## Group Cases

Use **group\_by(.data, ..., add = FALSE, drop = TRUE)** to create a "grouped" copy of a table grouped by columns in ... dplyr functions will manipulate each "group" separately and combine the results.



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

Use **rowwise(.data, ...)** to group data into individual rows. dplyr functions will compute results for each row. Also apply functions to list-columns. See tidyverse cheat sheet for list-column workflow.



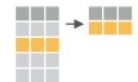
`starwars %>%  
rowwise() %>%  
mutate(film_count = length(films))`

**ungroup(x, ...)** Returns ungrouped copy of table.  
`ungroup(g_mtcars)`

## Manipulate Cases

### EXTRACT CASES

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



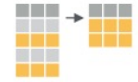
**filter(.data, ..., .preserve = FALSE)** Extract rows that meet logical criteria.  
`filter(mtcars, mpg > 20)`



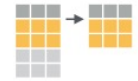
**distinct(.data, ..., .keep\_all = FALSE)** Remove rows with duplicate values.  
`distinct(mtcars, gear)`



**slice(.data, ..., .preserve = FALSE)** Select rows by position.  
`slice(mtcars, 10:15)`



**slice\_sample(.data, ..., n, prop, weight\_by = NULL, replace = FALSE)** Randomly select rows. Use `n` to select a number of rows and `prop` to select a fraction of rows.  
`slice_sample(mtcars, n = 5, replace = TRUE)`



**slice\_min(.data, order\_by, ..., n, prop, with\_ties = TRUE)** and **slice\_max()** Select rows with the lowest and highest values.  
`slice_min(mtcars, mpg, prop = 0.25)`



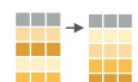
**slice\_head(.data, ..., n, prop)** and **slice\_tail()** Select the first or last rows.  
`slice_head(mtcars, n = 5)`

### Logical and boolean operators to use with filter()

<code>==</code>	<code>&lt;</code>	<code>&lt;=</code>	<code>is.na()</code>	<code>%in%</code>	<code> </code>	<code>xor()</code>
<code>!=</code>	<code>&gt;</code>	<code>&gt;=</code>	<code>!is.na()</code>	<code>!</code>	<code>&amp;</code>	

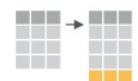
See `?base::Logic` and `?Comparison` for help.

### ARRANGE CASES



**arrange(.data, ..., .by\_group = FALSE)** 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

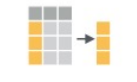


**add\_row(.data, ..., .before = NULL, .after = NULL)** Add one or more rows to a table.  
`add_row(cars, speed = 1, dist = 1)`

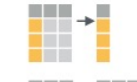
## Manipulate Variables

### EXTRACT VARIABLES

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



**pull(.data, var = -1, name = NULL, ...)** Extract column values as a vector, by name or index.  
`pull(mtcars, wt)`



**select(.data, ...)** Extract columns as a table.  
`select(mtcars, mpg, wt)`



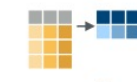
**relocate(.data, ..., .before = NULL, .after = NULL)** Move columns to new position.  
`relocate(mtcars, mpg, cyl, .after = last_col())`

### Use these helpers with select() and across()

e.g. `select(mtcars, mpg:cyl)`

<b>contains(match)</b>	<b>num_range(prefix, range)</b>	∴ e.g. <code>mpg:cyl</code>
<b>ends_with(match)</b>	<b>all_of(x)/any_of(x, ..., vars)</b>	∴ e.g. <code>gear</code>
<b>starts_with(match)</b>	<b>matches(match)</b>	<b>everything()</b>

### MANIPULATE MULTIPLE VARIABLES AT ONCE



**across(.cols, .funs, ..., .names = NULL)** Summarise or mutate multiple columns in the same way.  
`summarise(mtcars, across(everything(), mean))`

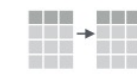


**c\_across(.cols)** Compute across columns in row-wise data.  
`transmute(rowwise(UKgas), total = sum(c_across(1:2)))`

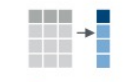
### MAKE NEW VARIABLES

Apply **vectorized functions** to columns. Vectorized functions take vectors as input and return vectors of the same length as output (see back).

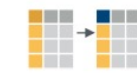
vectorized function



**mutate(.data, ..., .keep = "all", .before = NULL, .after = NULL)** Compute new column(s). Also **add\_column()**, **add\_count()**, and **add\_tally()**.  
`mutate(mtcars, gpm = 1 / mpg)`



**transmute(.data, ...)** Compute new column(s), drop others.  
`transmute(mtcars, gpm = 1 / mpg)`



**rename(.data, ...)** Rename columns. Use **rename\_with()** to rename with a function.  
`rename(cars, distance = dist)`

# Goals

1. Learn dplyr tools for grouping and summarizing data in R

# Objectives

1. Calculate a summary statistic for a variable using the `summarize()` function
2. Creates groupings of data using the `group_by()` function
3. Combine `group_by()` and `summarize()` functions to calculate summary statistics for groups of data



Please fill out an eval



<https://forms.office.com/r/rrMn3hevpY>



# Tests for Association



# Q: Is there an association between insurance product and SARS-CoV-2 RT-PCR positivity?

payor_group_fac <chr>	negative <int>	positive <int>
commercial	3549	86
government	3318	242
other	309	17
unassigned	7182	520

4 rows



```
data %>%  
  fisher.test(simulate.p.value = T)
```

# Data wrangling - 1

function that flexibly  
assigns values

```
covid_testing_2 <- covid_testing %>%  
  mutate(payor_group_fac = case_when(  
    is.na(payor_group) ~ "unassigned",  
    payor_group %in% c("charity care",  
                      "medical assistance",  
                      "self pay",  
                      "other") ~ "other",  
    TRUE ~ payor_group)  
  ) %>%  
  filter(result %in% c("positive", "negative"))
```

# Data wrangling - 2

```
# Generate counts
tmp_table_tall <- covid_testing_2 %>%
  group_by(payor_group_fac, result) %>%
  summarize(n = n()) %>%
  ungroup()
tmp_table_tall

# Pivot from tall to wide table
tmp_table_wide <- tmp_table_tall %>%
  spread(key = "result", value = "n")
tmp_table_wide
```

Remove groupings

Maps key values to  
separate columns

# Testing for association

payor_group_fac <chr>	negative <int>	positive <int>
commercial	3549	86
government	3318	242
other	309	17
unassigned	7182	520

4 rows



```
data %>%  
  fisher.test(simulate.p.value = T)
```



Fisher's Exact Test for Count Data with simulated p-value (based on 2000 replicates)

```
data: .  
p-value = 0.0004998  
alternative hypothesis: two.sided
```





# Regression Modeling

Q: Is the association between test positivity and a government insurance product explained by the age of the patient?

```
tmp <- covid_testing_2 %>%  
  filter(payor_group_fac %in% c("commercial", "government")) %>%  
  mutate(result_fac = factor(result,  
                              levels=c("negative", "positive"),  
                              ordered=T),  
         payor_group_fac = (payor_group == "government"))  
tmp_fit <- glm(result_fac ~ payor_group_fac + age,      # model formula  
              data = tmp,                             # dataset  
              family = "binomial"                     # type of model  
              )  
summary(tmp_fit)  
exp(coefficients(tmp_fit))                           # odds
```





# Output for logistic regression

```
Call:
glm(formula = result_fac ~ payor_group_fac + age, family = "binomial",
    data = tmp)
```

Deviance Residuals:

Min	1Q	Median	3Q	Max
-1.6365	-0.3468	-0.2532	-0.1985	2.8393

Coefficients:

	Estimate	Std. Error	z value	Pr(> z )	
(Intercept)	-4.016195	0.119611	-33.577	< 2e-16	***
payor_group_facTRUE	1.136566	0.128761	8.827	< 2e-16	***
age	0.032897	0.004436	7.416	1.21e-13	***

---

Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 2666.6 on 7194 degrees of freedom  
Residual deviance: 2535.2 on 7192 degrees of freedom  
AIC: 2541.2

Number of Fisher Scoring iterations: 6

(Intercept)	payor_group_facTRUE	age
0.01802141	3.11604971	1.03344368

Odds for Payor Group  
and Age