

# Enter the tidyverse: Processing across rows

## Managing and Manipulating Data Using R

# Introduction

# Lecture outline

1. Introduction
2. Introduce `group_by()` and `summarize()`
  - 2.1 `group_by`
  - 2.2 `summarize()`
3. Combining `group_by()` and `summarize()`
  - 3.1 `summarize()` and Counts
  - 3.2 `summarize()` and means
  - 3.3 `summarize()` and logical vectors, part II
4. Summarize multiple columns
5. Attach aggregate measures to your data frame

## Libraries we will use today

“Load” the package we will use today (output omitted)

▶ **you must run this code chunk**

```
library(tidyverse)
```

If package not yet installed, then must install before you load. Install in “console” rather than .Rmd file

▶ Generic syntax: `install.packages("package_name")`

▶ Install “tidyverse”: `install.packages("tidyverse")`

Note: when we load package, name of package is not in quotes; but when we install package, name of package is in quotes:

▶ `install.packages("tidyverse")`

▶ `library(tidyverse)`

# Data we will use today

Data on off-campus recruiting events by public universities

▶ Object `df_event`

▶ One observation per university, recruiting event

```
rm(list = ls()) # remove all objects
```

```
#load dataset with one obs per recruiting event
```

```
load(url("https://github.com/ozanj/rclass/raw/master/data/recruiting/recruit_ev
```

```
#load("../..//data/recruiting/recruit_event_allvars.Rdata")
```

## Processing across variables vs. processing across observations

Visits by UC Berkeley to public high schools

```
#> # A tibble: 5 x 6
#>   school_id    state tot_stu_pub fr_lunch pct_fr_lunch med_inc
#>   <chr>        <chr>    <dbl>    <dbl>    <dbl>    <dbl>
#> 1 340882002126 NJ          1846      29    0.0157 178732
#> 2 340147000250 NJ          1044      50    0.0479  62288
#> 3 340561003796 NJ          1505     298    0.198 100684.
#> 4 340165005124 NJ          1900      43    0.0226 160476.
#> 5 341341003182 NJ          1519     130    0.0856 144346
```

So far, we have focused on “processing across variables”

- ▶ Performing calculations across columns (i.e., vars), typically within a row (i.e., observation)
- ▶ Example: percent free-reduced lunch (above)

Processing across obs (focus of today’s lecture)

- ▶ Performing calculations across rows (i.e., obs), often within a column (i.e., variable)
- ▶ Example: Average household income of visited high schools, by state

## Why processing across observations

### Note

- ▶ In today's lecture, I'll use the terms "observations" and "rows" interchangeably

### Creation of analysis datasets often requires calculations across obs

#### Examples:

- ▶ You have a dataset with one observation per student-term and want to create a variable of credits attempted per term
- ▶ You have a dataset with one observation per student-term and want to create a variable of GPA for the semester or cumulative GPA for all semesters
- ▶ Number of off-campus recruiting events university makes to each state
- ▶ Average household income at visited versus non-visited high schools

### Creating graphs and tables of descriptive stats usually require calculations across obs

Example: Want to create a graph that shows number of recruiting events by event "type" (e.g., public HS, private HS) for each university

- ▶ Start with `df_event` dataset that has one observation per university, recruiting event
- ▶ Create new data frame object that has one observation per university and event type and has variable for number of events
  - ▶ this variable calculated by counting number of rows in each combination of university and event type
- ▶ This new data frame object is the input for creating desired graph

Introduce `group_by()` and `summarize()`



## Strategy for teaching processing across obs

In `tidyverse` the `group_by()` and `summarize()` functions are the primary means of performing calculations across observations

- ▶ Usually, processing across observations requires using `group_by()` and `summarize()` together
- ▶ `group_by()` typically not very useful by itself
- ▶ `summarize()` [with or without `group_by()`] can be helpful for creating summary statistics that are the inputs for tables or graphs you create

How we'll teach:

- ▶ introduce `group_by()` and `summarize()` separately
  - ▶ goal: you understand what each function does
- ▶ then we'll combine them

group\_by

## group\_by()

**Description:** “group\_by() takes an existing data frame and converts it into a grouped data frame where operations are performed”by group“. ungroup() removes grouping.”

- ▶ part of **dplyr** package within **tidyverse**; not part of **Base R**
- ▶ works best with pipes `%>%` and `summarize()` function [described below]

**Basic syntax:** `group_by(.data, ...)`

- ▶ `.data` argument refers to name of data frame
- ▶ `...` argument refers to names of “group\_by” variables, separated by commas
  - ▶ Can “group by” one or many variables
  - ▶ Typically, “group\_by” variables are character, factor, or integer variables

Possible “group by” variables in `df_event` data:

- ▶ university name/id; event type (e.g., public HS, private HS); state

**Example:** in `df_event`, create frequency count of `event_type` [output omitted]

```
names(df_event)
#without group_by()
df_event %>% count(event_type)
df_event %>% count(instnm)
#group_by() university
df_event %>% group_by(instnm) %>% count(event_type)
```

## group\_by()

By itself `group_by()` doesn't do much; it just prints data

- ▶ Below, group `df_event` data by university, event type, and event state

```
#print object
df_event
#group_by (without pipes)
group_by(df_event, univ_id, event_type, event_state)
#group_by (with pipes)
df_event %>% group_by(univ_id, event_type, event_state)
df_event %>% group_by(univ_id, event_type, event_state) %>% glimpse()
```

But once an object is grouped, all subsequent functions are run separately “by group”

- ▶ recall that `count()` counts number of observations by group

```
# count number of observations in group, ungrouped data
df_event %>% count()
#group by and then count obs
df_event %>% group_by(univ_id) %>% count()
df_event %>% group_by(univ_id) %>% count() %>% glimpse()

df_event %>% group_by(univ_id, event_type) %>% count()
df_event %>% group_by(univ_id, event_type) %>% count() %>% glimpse()

df_event %>% group_by(univ_id, event_type, event_state) %>% count()
```

## Grouping not retained unless you **assign** it

Below, we'll use `class()` function to show whether data frame is grouped

- ▶ will talk more about `class()` next week, but for now, just think of it as a function that provides information about an object
- ▶ similar to `typeof()`, but `class()` provides different info about object

Grouping is not retained unless you **assign** it

```
class(df_event)
#> [1] "tbl_df"      "tbl"        "data.frame"
```

```
df_event %>% group_by(univ_id, event_type, event_state)
df_event_grp <- df_event %>% group_by(univ_id, event_type, event_state) # using
```

```
class(df_event_grp)
#> [1] "grouped_df" "tbl_df"     "tbl"        "data.frame"
```

## Un-grouping an object

Use `ungroup(object)` to un-group grouped data

```
class(df_event_grp)
#> [1] "grouped_df" "tbl_df"      "tbl"        "data.frame"
df_event_grp <- ungroup(df_event_grp)
class(df_event_grp)
#> [1] "tbl_df"      "tbl"        "data.frame"
rm(df_event_grp)
```

1. Group by “instnm” and get a frequency count.
  - ▶ How many rows and columns do you have? What do the number of rows mean?
2. Now group by “instnm” **and** “event\_type” and get a frequency count.
  - ▶ How many rows and columns do you have? What do the number of rows mean?
3. **Bonus:** In the same code chunk, group by “instnm” and “event\_type”, but this time filter for observations where “med\_inc” is greater than 75000 and get a frequency count.

## group\_by() student exercise solutions

1. Group by “instnm” and get a frequency count.

▶ How many rows and columns do you have? What do the number of rows mean?

```
df_event %>%  
  group_by(instnm) %>%  
  count()  
  
#> # A tibble: 16 x 2  
#>   instnm      n  
#>   <chr>    <int>  
#> 1 Arkansas    994  
#> 2 Bama      4258  
#> 3 Cinci     679  
#> 4 CU Boulder 1439  
#> 5 Kansas    1014  
#> 6 NC State   640  
#> 7 Pitt      1225  
#> 8 Rutgers   1135  
#> 9 S Illinois 549  
#> 10 Stony Brook 730  
#> 11 UC Berkeley 879  
#> 12 UC Irvine  539  
#> 13 UGA       827  
#> 14 UM Amherst 908  
#> 15 UNL      1397  
#> 16 USCC     1467
```



2. Now group by “instnm” and “event\_type” and get a frequency count.

▶ How many rows and columns do you have? What do the number of rows mean?

```
df_event %>%  
  group_by(instnm, event_type) %>%  
  count()  
#> # A tibble: 80 x 3  
#>   instnm event_type      n  
#>   <chr>  <chr>      <int>  
#> 1 Arkansas 2yr college    32  
#> 2 Arkansas 4yr college    14  
#> 3 Arkansas other         112  
#> 4 Arkansas private hs    222  
#> 5 Arkansas public hs    614  
#> 6 Bama    2yr college    127  
#> 7 Bama    4yr college    158  
#> 8 Bama    other         608  
#> 9 Bama    private hs    963  
#> 10 Bama   public hs   2402  
#> # ... with 70 more rows
```

## group\_by() student exercise solutions

3. **Bonus:** Group by “instnm” and “event\_type”, but this time filter for observations where “med\_inc” is greater than 75000 and get a frequency count.

```
df_event %>%  
  group_by(instnm, event_type) %>%  
  filter(med_inc > 75000) %>%  
  count()  
#> # A tibble: 80 x 3  
#>   instnm event_type      n  
#>   <chr>  <chr>      <int>  
#> 1 Arkansas 2yr college    7  
#> 2 Arkansas 4yr college    3  
#> 3 Arkansas other        30  
#> 4 Arkansas private hs   99  
#> 5 Arkansas public hs  303  
#> 6 Bama    2yr college   21  
#> 7 Bama    4yr college   42  
#> 8 Bama    other       249  
#> 9 Bama    private hs  477  
#> 10 Bama   public hs  1478  
#> # ... with 70 more rows
```

summarize()

## summarize() function

**Description:** `summarize()` calculates across rows; then collapses into single row

- ▶ `summarize()` create scalar vars summarizing variables of existing data frame
- ▶ if you first group data frame using `group_by()`, `summarize()` creates summary vars separately for each group, returning object with one row per group
- ▶ if data frame not grouped, `summarize()` will result in one row.

**Syntax:** `summarize(.data, ...)`

- ▶ `.data`: a data frame; omit if using `summarize()` after pipe `%>%`
- ▶ `...`: Name-value pairs of summary functions separated by commas
  - ▶ “name” will be the name of new variable you will create
  - ▶ “value” should be expression that returns a single value like `min(x)`, `n()`
  - ▶ variable names do not need to be placed within quotes

**Value** (what `summarize()` returns/creates)

- ▶ Object of same class as `.data`; object will have one obs per “by group”

**Useful functions** (i.e., “helper functions”)

- ▶ Standalone functions called *within* `summarize()`, e.g., `mean()`, `n()`
- ▶ e.g., count function `n()` takes no arguments; returns number of rows in group

## Investigate objects created by `summarize()`

**Example:** Count total number of events

- ▶ function `n()` from `dplyr` package (`dplyr::n`) returns the size of the current group

```
##?n # dplyr function n() gives current group size  
summarize(df_event, num_events=n()) # without pipes
```

```
df_event %>% summarize(num_events=n()) # with pipes
```

```
df_event %>% summarize(num_events=n()) %>% str() # use str to see what pipe returns
```

**Example:** What is max value of `med_inc` across all events

- ▶ function `max()` from `base` package (`base::max`) returns max value

```
##?max # base R function max() returns max value
```

```
df_event %>% summarize(max_inc=max(med_inc, na.rm = TRUE))
```

```
df_event %>% summarize(max_inc=max(med_inc, na.rm = TRUE)) %>% str()
```

## Investigate objects created by `summarize()`

**Example:** Count total number of events AND max value of median income

```
df_event %>% summarize(num_events=n(),
                       max_inc=max(med_inc, na.rm = TRUE))
#> # A tibble: 1 x 2
#>   num_events max_inc
#>   <int>     <dbl>
#> 1     18680 250001
df_event %>% summarize(num_events=n(), # show object returned by pipe
                       max_inc=max(med_inc, na.rm = TRUE)) %>% str()
#> tibble [1 x 2] (S3: tbl_df/tbl/data.frame)
#> $ num_events: int 18680
#> $ max_inc    : num 250001
```

We can use assignment to keep the object created by `summarize()`

```
df_event_temp <- df_event %>% summarize(num_events=n(),
                                         max_inc=max(med_inc, na.rm = TRUE))
df_event_temp
rm(df_event_temp)
```

What if we forgot `na.rm = TRUE` ?

- ▶ then `max_inc` equals `NA` cuz can't perform calculation on `NA` values

```
df_event %>% summarize(num_events=n(),
                       max_inc=max(med_inc, na.rm = FALSE))
```

## Takeaways

- ▶ by default, objects created by `summarize()` are data frames that contain variables created within `summarize()` and one observation [per “by group”]
- ▶ most “helper” functions (e.g., `max()`, `mean()`) have option `na.rm` to keep/remove missing obs before performing calculations
  - ▶ `na.rm = FALSE` (default); don't remove `NA`s prior to calculation
    - ▶ if any obs missing, then result of calculation is `NA`
  - ▶ `na.rm = TRUE` (default); remove `NA`s prior to calculation

## Retaining objects created by `summarize()`

Object created by `summarize()` not retained unless you **assign** it

```
event_temp <- df_event %>% summarize(num_events=n(),  
  mean_inc=mean(med_inc, na.rm = TRUE))
```

```
event_temp
```

```
#> # A tibble: 1 x 2
```

```
#>   num_events mean_inc
```

```
#>   <int>     <dbl>
```

```
#> 1     18680  89089.
```

```
rm(event_temp)
```



## Using `[]` operator to filter observations within summarize

Imagine we want to calculate avg. income, separately for in-state vs. out-of-state visits

- ▶ first, let's use `filter()` to make sure we can identify in-state vs. out-of-state

```
#in state
df_event %>% filter(event_state == instst) %>% count() %>% as_vector()
#>      n
#> 5425
#out state
df_event %>% filter(event_state != instst) %>% count() %>% as_vector()
#>      n
#> 13255
```

- ▶ calculate mean income for: all events; in-state events; out-of-state events

```
df_event %>%
  summarize(avg_inc = mean(med_inc, na.rm = TRUE), # all events
            avg_inc_inst = mean(med_inc[event_state == instst], na.rm = TRUE), # in-sta
            avg_inc_outst = mean(med_inc[event_state != instst], na.rm = TRUE) # out-st
  )
#> # A tibble: 1 x 3
#>   avg_inc avg_inc_inst avg_inc_outst
#>   <dbl>     <dbl>     <dbl>
#> 1  89089.     71589.     96162.
```

## Using `summarize()` to create descriptive statistics table

Often helpful to use `summarize()` to calculate summary statistics that are the basis for a table of descriptive statistics

**Task:** create a table of descriptive statistics about variable `med_inc`

▶ want these measures: number of non-missing obs; mean; standard deviation

```
df_event %>% mutate(non_miss_inc = is.na(med_inc)==0) %>%  
  summarize(  
    n = sum(non_miss_inc, na.rm = TRUE), #SAMPLE SIZE all  
    avg_inc = mean(med_inc, na.rm = TRUE), # MEAN  
    std_inc = sd(med_inc, na.rm = TRUE) # STANDARD DEVIATION all events  
  )
```

**Task:** same as above but separate measures for: all events; in-state; out-of-state

```
df_event %>% mutate(non_miss_inc = is.na(med_inc)==0) %>%  
  summarize(  
    n = sum(non_miss_inc, na.rm = TRUE), #SAMPLE SIZE  
    n_inst = sum(non_miss_inc[event_state == instst], na.rm = TRUE),  
    n_outst = sum(non_miss_inc[event_state != instst], na.rm = TRUE),  
    avg_inc = mean(med_inc, na.rm = TRUE), # MEAN  
    avg_inc_inst = mean(med_inc[event_state == instst], na.rm = TRUE),  
    avg_inc_outst = mean(med_inc[event_state != instst], na.rm = TRUE),  
    std_inc = sd(med_inc, na.rm = TRUE), # STANDARD DEVIATION  
    std_inc_inst = sd(med_inc[event_state == instst], na.rm = TRUE),  
    std_inc_outst = sd(med_inc[event_state != instst], na.rm = TRUE)
```

## summarize() student exercise

1. What is the min value of `med_inc` across all events?
  - ▶ Hint: Use `min()`
2. What is the mean value of `fr_lunch` across all events?
  - ▶ Hint: Use `mean()`

## summarize() student exercise

1. What is min value of `med_inc` across all events?

```
df_event %>%  
  summarize(min_med_income = min(med_inc, na.rm = TRUE))  
#> # A tibble: 1 x 1  
#>   min_med_income  
#>   <dbl>  
#> 1         12894.
```

## summarize() student exercise

2. What is the mean value of `fr_lunch` across all events?

▶ Hint: Use `mean()`

```
df_event %>%  
  summarize(mean_fr_lunch = mean(fr_lunch, na.rm = TRUE))  
#> # A tibble: 1 x 1  
#>   mean_fr_lunch  
#>   <dbl>  
#> 1           475.
```

Combining `group_by()` and `summarize()`

## Combining `summarize()` and `group_by`

`summarize()` on ungrouped vs. grouped data:

- ▶ By itself, `summarize()` performs calculations across all rows of data frame then collapses the data frame to a single row
- ▶ When data frame is grouped, `summarize()` performs calculations across rows within a group and then collapses to a single row for each group

**Example:** Count the number of events for each university

- ▶ remember: `df_event` has one observation per university, recruiting event

```
df_event %>% summarize(num_events=n())
df_event %>% group_by(instnm) %>% summarize(num_events=n())
#> `summarise()` ungrouping output (override with `.groups` argument)
```

- ▶ Investigate the object created above

```
df_event %>% group_by(instnm) %>% summarize(num_events=n()) %>% str()
#> `summarise()` ungrouping output (override with `.groups` argument)
```

- ▶ Or we could retain object for later use

```
event_by_univ <- df_event %>% group_by(instnm) %>% summarize(num_events=n())
#> `summarise()` ungrouping output (override with `.groups` argument)
str(event_by_univ)
event_by_univ # print
rm(event_by_univ)
```

## Combining `summarize()` and `group_by`

### Task

- ▶ Count number of recruiting events by institution and event\_type

```
df_event %>% group_by(instnm, event_type) %>% summarize(num_events=n())  
#> `summarise()` regrouping output by 'instnm' (override with ` .groups ` argument)  
  
#investigate object created  
df_event %>% group_by(instnm, event_type) %>% summarize(num_events=n()) %>% glim  
#> `summarise()` regrouping output by 'instnm' (override with ` .groups ` argument)
```

Note that data frame object created by `group_by()` and `summarize()` can be input to graph

```
#bar chart of number of events, all universities combined  
df_event %>% group_by(instnm, event_type) %>%  
  summarize(num_events=n()) %>%  
  ggplot(aes(x=event_type, y=num_events)) + # plot  
  ylab("Number of events") + xlab("Event type") +geom_col()  
  
#bar chart of number of events, separate chart for each university  
df_event %>% group_by(instnm, event_type) %>%  
  summarize(num_events=n()) %>%  
  ggplot(aes(x=event_type, y=num_events)) + # plot  
  ylab("Number of events") + xlab("Event type") + geom_col() +  
  coord_flip() + facet_wrap(~ instnm)
```



## Combining `summarize()` and `group_by`

**Task.** Count number of recruiting events by institution, event\_type, and whether event is in- or out-of-state (var= `event_inst` )

- ▶ Note: in `group_by()` , the optional `drop` argument controls whether empty groups dropped. default is `drop = TRUE`

```
df_event %>% group_by(instnm, event_type, event_inst) %>%  
  summarize(num_events=n())  
#> `summarise()` regrouping output by 'instnm', 'event_type' (override with `.groups`)
```

```
df_event %>% group_by(instnm, event_type, event_inst, .drop = TRUE) %>%  
  summarize(num_events=n())  
#> `summarise()` regrouping output by 'instnm', 'event_type' (override with `.groups`)
```

```
df_event %>%  
  group_by(as.factor(instnm), as.factor(event_type), as.factor(event_inst),  
          .drop = FALSE) %>% summarize(num_events=n()) %>% arrange(num_events)  
#> `summarise()` regrouping output by 'as.factor(instnm)', 'as.factor(event_type)  
# .drop=FALSE affects only grouping columns that are coded as factors  
# combinations that include non-factor grouping variables are still  
# silently dropped even with .drop=FALSE.
```

## Combining `summarize()` and `group_by`

Make a graph, showing in/out state as fill color of bar

```
df_event %>% group_by(instnm, event_type, event_inst) %>%  
  summarize(num_events=n()) %>%  
  ggplot(aes(x=event_type, y=num_events, fill = event_inst)) + # plot  
  ylab("Number of events") + xlab("Event type") + geom_col() +  
  coord_flip() + facet_wrap(~ instnm)
```

## Combining `summarize()` and `group_by`

### Task

- ▶ By university, event type, event\_inst count the number of events and calculate the avg. pct white in the zip-code

```
df_event %>% group_by(instnm, event_type, event_inst) %>%  
  summarize(num_events=n(),  
            mean_pct_white=mean(pct_white_zip, na.rm = TRUE)  
  )  
#> `summarise()` regrouping output by 'instnm', 'event_type' (override with `group_by()`)  
  
#investigate object you created  
df_event %>% group_by(instnm, event_type, event_inst) %>%  
  summarize(num_events=n(),  
            mean_pct_white=mean(pct_white_zip, na.rm = FALSE)  
  ) %>% glimpse()  
#> `summarise()` regrouping output by 'instnm', 'event_type' (override with `group_by()`)
```

## Combining `summarize()` and `group_by`

Recruiting events by UC Berkeley

```
df_event %>% filter(univ_id == 110635) %>%  
  group_by(event_type) %>% summarize(num_events=n())  
#> `summarise()` ungrouping output (override with `.groups` argument)
```

Let's create a dataset of recruiting events at UC Berkeley

```
event_berk <- df_event %>% filter(univ_id == 110635)  
  
event_berk %>% count(event_type)
```

summarize() and Counts

## summarize() : Counts

The count function `n()` takes no arguments and returns the size of the current group

```
event_berk %>% group_by(event_type, event_inst) %>%  
  summarize(num_events=n())  
#> `summarise()` regrouping output by 'event_type' (override with `groups` argu
```

Because counts are so important, `dplyr` package includes separate `count()` function that can be called outside `summarize()` function

```
event_berk %>% group_by(event_type, event_inst) %>% count()
```

## `summarize()` : count with logical vectors and `sum()`

Logical vectors have values `TRUE` and `FALSE` .

▶ When used with numeric functions, `TRUE` converted to 1 and `FALSE` to 0.

`sum()` is a numeric function that returns the sum of values

```
sum(c(5,10))
sum(c(TRUE,TRUE,FALSE,FALSE))
```

`is.na()` returns `TRUE` if value is `NA` and otherwise returns `FALSE`

```
is.na(c(5,NA,4,NA))
#> [1] FALSE TRUE FALSE TRUE
```

```
sum(is.na(c(5,NA,4,NA,5)))
#> [1] 2
sum(!is.na(c(5,NA,4,NA,5)))
#> [1] 3
```

Application: How many missing/non-missing obs in variable [**very important**]

```
event_berk %>% group_by(event_type) %>%
  summarize(
    n_events = n(),
    n_miss_inc = sum(is.na(med_inc)),
    n_nonmiss_inc = sum(!is.na(med_inc)),
    n_nonmiss_fr_lunch = sum(!is.na(fr_lunch))
  )
```

## summarize() and count student exercise

Use one code chunk for this exercise. You could tackle this a step at a time and run the entire code chunk when you have answered all parts of this question. Create your own variable names.

1. Using the `event_berk` object, filter observations where `event_state` is VA and group by `event_type`.
  - 1.1 Using the `summarize` function to create a variable that represents the count for each `event_type`.
  - 1.2 Create a variable that represents the sum of missing obs for `med_inc`.
  - 1.3 Create a variable that represents the sum of non-missing obs for `med_inc`.
  - 1.4 **Bonus:** Arrange variable you created representing the count of each `event_type` in descending order.



## summarize() and count student exercise SOLUTION

- Using the `event_berk` object filter observations where `event_state` is VA and group by `event_type`.
  - Using the `summarize` function, create a variable that represents the count for each `event_type`.
  - Now get the sum of missing obs for `med_inc`.
  - Now get the sum of non-missing obs for `med_inc`.

```
event_berk %>%
  filter(event_state == "VA") %>%
  group_by(event_type) %>%
  summarize(
    n_events = n(),
    n_miss_inc = sum(is.na(med_inc)),
    n_nonmiss_inc = sum(!is.na(med_inc))) %>%
  arrange(desc(n_events))
#> `summarise()` ungrouping output (override with `.groups` argument)
#> # A tibble: 3 x 4
#>   event_type n_events n_miss_inc n_nonmiss_inc
#>   <chr>      <int>      <int>      <int>
#> 1 public hs         20          0          20
#> 2 private hs        13          0          13
#> 3 other             3          0           3
```

summarize() and means

## summarize() : means

The `mean()` function within `summarize()` calculates means, separately for each group

```
event_berk %>% group_by(event_inst, event_type) %>% summarize(  
  n_events=n(),  
  mean_inc=mean(med_inc, na.rm = TRUE),  
  mean_pct_white=mean(pct_white_zip, na.rm = TRUE))  
#> `summarise()` regrouping output by 'event_inst' (override with ``.groups` argument)  
#> # A tibble: 10 x 5  
#>   event_inst event_type  n_events mean_inc mean_pct_white  
#>   <chr>      <chr>         <int>   <dbl>   <dbl>  
#> 1 In-State  2yr college    111  78486.   40.1  
#> 2 In-State  4yr college    14 131691.   58.0  
#> 3 In-State  other          49  75040.   37.6  
#> 4 In-State  private hs     35  95229.   48.4  
#> 5 In-State  public hs     259  87097.   39.6  
#> 6 Out-State 2yr college     1 153070.   89.7  
#> 7 Out-State 4yr college     4  76913.   65.8  
#> 8 Out-State other          89  69004.   56.5  
#> 9 Out-State private hs    134  87654.   64.3  
#> 10 Out-State public hs    183 103603.   62.0
```

## `summarize()` : means and `na.rm` argument

Default behavior of “aggregation functions” (e.g., `summarize()` )

- ▶ if *input* has any missing values ( `NA` ), then output will be missing.

Many functions have argument `na.rm` (means “remove `NAs` ”)

- ▶ `na.rm = FALSE` [the default for `mean()` ]
  - ▶ Do not remove missing values from input before calculating
  - ▶ Therefore, missing values in input will cause output to be missing
- ▶ `na.rm = TRUE`
  - ▶ Remove missing values from input before calculating
  - ▶ Therefore, missing values in input will not cause output to be missing

*#na.rm = FALSE; the default setting*

```
event_berk %>% group_by(event_inst, event_type) %>% summarize(  
  n_events=n(),  
  n_miss_inc = sum(is.na(med_inc)),  
  mean_inc=mean(med_inc, na.rm = FALSE),  
  n_miss_frlunch = sum(is.na(fr_lunch)),  
  mean_fr_lunch=mean(fr_lunch, na.rm = FALSE))
```

*#> `summarise()` regrouping output by 'event\_inst' (override with `groups` argument)*

*#na.rm = TRUE*

```
event_berk %>% group_by(event_inst, event_type) %>% summarize(  
  n_events=n(),  
  n_miss_inc = sum(is.na(med_inc)),  
  mean_inc=mean(med_inc, na.rm = TRUE),  
  n_miss_frlunch = sum(is.na(fr_lunch)),  
  mean_fr_lunch=mean(fr_lunch, na.rm = TRUE))
```

## Student exercise

1. Using the `event_berk` object, group by `instnm`, `event_inst`, & `event_type`.
  - 1.1 Create vars for number non\_missing for these racial/ethnic groups ( `pct_white_zip`, `pct_black_zip`, `pct_asian_zip`, `pct_hispanic_zip`, `pct_amerindian_zip`, `pct_nativehawaii_zip` )
  - 1.2 Create vars for mean percent for each racial/ethnic group

## Student exercise solutions

```
event_berk %>% group_by(instnm, event_inst, event_type) %>%
  summarize(
    n_events=n(),
    n_miss_white = sum(!is.na(pct_white_zip)),
    mean_white = mean(pct_white_zip, na.rm = TRUE),
    n_miss_black = sum(!is.na(pct_black_zip)),
    mean_black = mean(pct_black_zip, na.rm = TRUE),
    n_miss_asian = sum(!is.na(pct_asian_zip)),
    mean_asian = mean(pct_asian_zip, na.rm = TRUE),
    n_miss_lat = sum(!is.na(pct_hispanic_zip)),
    mean_lat = mean(pct_hispanic_zip, na.rm = TRUE),
    n_miss_na = sum(!is.na(pct_amerindian_zip)),
    mean_na = mean(pct_amerindian_zip, na.rm = TRUE),
    n_miss_nh = sum(!is.na(pct_nativehawaii_zip)),
    mean_nh = mean(pct_nativehawaii_zip, na.rm = TRUE)) %>%
  head(6)
```

```
#> `summarise()` regrouping output by 'instnm', 'event_inst' (override with `group_by()`)
```

```
#> # A tibble: 6 x 16
```

```
#>   instnm event_inst event_type n_events n_miss_white mean_white n_miss_black
#>   <chr> <chr> <chr> <int> <int> <dbl> <int>
#> 1 UC Be~ In-State 2yr colle~ 111 106 40.1 106
#> 2 UC Be~ In-State 4yr colle~ 14 12 58.0 12
#> 3 UC Be~ In-State other 49 48 37.6 48
#> 4 UC Be~ In-State private hs 35 35 48.4 35
#> 5 UC Be~ In-State public hs 259 258 39.6 258
#> 6 UC Be~ Out-State 2yr colle~ 1 1 89.7 1
#> # with 9 more variables: mean_black <dbl>, n_miss_asian <int>
```

summarize() and logical vectors, part II

## summarize() : counts with logical vectors, part II

Logical vectors (e.g., `is.na()`) useful for counting obs that satisfy some condition

```
is.na(c(5,NA,4,NA))
#> [1] FALSE TRUE FALSE TRUE
typeof(is.na(c(5,NA,4,NA)))
#> [1] "logical"
sum(is.na(c(5,NA,4,NA)))
#> [1] 2
```

**Task:** Using object `event_berk`, calculate the following measures for each combination of `event_type` and `event_inst`:

- ▶ count of number of rows for each group
- ▶ count of rows non-missing for both `pct_black_zip` and `pct_hispanic_zip`
- ▶ count of number of visits to communities where the `sum` of Black and Latinx people comprise more than 50% of the total population

```
event_berk %>% group_by (event_inst, event_type) %>%
  summarize(
    n_events=n(),
    n_nonmiss_latbl = sum(!is.na(pct_black_zip) & !is.na(pct_hispanic_zip)),
    n_majority_latbl= sum(pct_black_zip+ pct_hispanic_zip>50, na.rm = TRUE)
  )
#> `summarise()` regrouping output by 'event_inst' (override with `groups` argu
```



## summarize() : logical vectors to count *proportions*

Syntax: `group_by(vars) %>% summarize(prop = mean(TRUE/FALSE conditon))`

**Task:** separately for in-state/out-of-state, what proportion of visits to public high schools are to communities with median income greater than \$100,000?

Steps:

1. Filter public HS visits
2. group by in-state vs. out-of-state
3. Create measure

```
event_berk %>% filter(event_type == "public hs") %>% # filter public hs visits
  group_by (event_inst) %>% # group by in-state vs. out-of-state
  summarize(
    n_events=n(), # number of events by group
    n_nonmiss_inc = sum(!is.na(med_inc)), # w/ nonmissings values median inc,
    p_incgt100k = mean(med_inc>100000, na.rm=TRUE)) # proportion visits to $100k
#> `summarise()` ungrouping output (override with `.groups` argument)
#> # A tibble: 2 x 4
#>   event_inst n_events n_nonmiss_inc p_incgt100k
#>   <chr>      <int>      <int>      <dbl>
#> 1 In-State    259        256      0.273
#> 2 Out-State   183        183      0.519
```

## summarize() : logical vectors to count *proportions*

What if we forgot to put `na.rm=TRUE` in the above task?

**Task:** separately for in-state/out-of-state, what proportion of visits to public high schools are to communities with median income greater than \$100,000?

```
event_berk %>% filter(event_type == "public hs") %>% # filter public hs visits
  group_by(event_inst) %>% # group by in-state vs. out-of-state
  summarize(
    n_events=n(), # number of events by group
    n_nonmiss_inc = sum(!is.na(med_inc)), # w/ nonmissings values median inc,
    p_incgt100k = mean(med_inc>100000, na.rm=TRUE)) # proportion visits to $100k
#> `summarise()` ungrouping output (override with `.groups` argument)
#> # A tibble: 2 x 4
#>   event_inst n_events n_nonmiss_inc p_incgt100k
#>   <chr>      <int>      <int>      <dbl>
#> 1 In-State    259          256      0.273
#> 2 Out-State   183          183      0.519
```

## `summarize()` : Other “helper” functions

Lots of other functions we can use within `summarize()`

Common functions to use with `summarize()` :

Function	Description
<code>n</code>	count
<code>n_distinct</code>	count unique values
<code>mean</code>	mean
<code>median</code>	median
<code>max</code>	largest value
<code>min</code>	smallest value
<code>sd</code>	standard deviation
<code>sum</code>	sum of values
<code>first</code>	first value
<code>last</code>	last value
<code>nth</code>	nth value
<code>any</code>	condition true for at least one value

*Note: These functions can also be used on their own or with `mutate()`*

## summarize() : Other functions

Maximum value in a group

```
max(c(10,50,8))  
#> [1] 50
```

**Task:** For each combination of in-state/out-of-state and event type, what is the maximum value of `med_inc` ?

```
event_berk %>% group_by(event_type, event_inst) %>%  
  summarize(max_inc = max(med_inc)) # oops, we forgot to remove NAs!  
#> `summarise()` regrouping output by 'event_type' (override with `groups` argument)  
#> # A tibble: 10 x 3  
#>   event_type event_inst max_inc  
#>   <chr>      <chr>      <dbl>  
#> 1 2yr college In-State      NA  
#> 2 2yr college Out-State 153070.  
#> 3 4yr college In-State      NA  
#> 4 4yr college Out-State      NA  
#> 5 other      In-State      NA  
#> 6 other      Out-State      NA  
#> 7 private hs In-State 250001  
#> 8 private hs Out-State      NA  
#> 9 public hs  In-State      NA  
#> 10 public hs Out-State 223556.
```

```
event_berk %>% group_by(event_type, event_inst) %>%  
  summarize(max_inc = max(med_inc, na.rm = TRUE))
```

## summarize() : Other functions

Isolate first/last/nth observation in a group

```
x <- c(10,15,20,25,30)
first(x)
last(x)
nth(x,1)
nth(x,3)
nth(x,10)
```

**Task:** after sorting object `event_berk` by `event_type` and `event_datetime_start`, what is the value of `event_date` for:

- ▶ first event for each event type?
- ▶ the last event for each event type?
- ▶ the 50th event for each event type?

```
event_berk %>% arrange(event_type, event_datetime_start) %>%
  group_by(event_type) %>%
  summarize(
    n_events = n(),
    date_first= first(event_date),
    date_last= last(event_date),
    date_50th= nth(event_date, 50)
  )
#> `summarise()` ungrouping output (override with `groups` argument)
```

## Student exercise

Identify value of `event_date` for the *n*th event in each by group

### Specific task:

- ▶ arrange (i.e., sort) by `event_type` and `event_datetime_start`, then group by `event_type`, and then identify the value of `event_date` for:
  - ▶ the first event in each by group (`event_type`)
  - ▶ the second event in each by group
  - ▶ the third event in each by group
  - ▶ the fourth event in each by group
  - ▶ the fifth event in each by group

## Student exercise solution

```
event_berk %>% arrange(event_type, event_datetime_start) %>%
  group_by(event_type) %>%
  summarize(
    n_events = n(),
    date_1st= first(event_date),
    date_2nd= nth(event_date,2),
    date_3rd= nth(event_date,3),
    date_4th= nth(event_date,4),
    date_5th= nth(event_date,5))
#> `summarise()` ungrouping output (override with `.groups` argument)
#> # A tibble: 5 x 7
#>   event_type n_events date_1st   date_2nd   date_3rd   date_4th   date_5th
#>   <chr>         <int> <date>     <date>     <date>     <date>     <date>
#> 1 2yr college     112 2017-04-25 2017-09-05 2017-09-05 2017-09-06 2017-09-06
#> 2 4yr college      18 2017-04-30 2017-05-01 2017-05-06 2017-09-13 2017-09-14
#> 3 other          138 2017-04-11 2017-04-23 2017-04-25 2017-04-29 2017-05-14
#> 4 private hs     169 2017-04-23 2017-04-24 2017-04-29 2017-04-30 2017-09-05
#> 5 public hs      442 2017-04-14 2017-04-24 2017-04-26 2017-04-27 2017-04-27
```

Summarize multiple columns



## What are “scoped” variants of a function?

“Scoped” variants of a function apply the function to a selection of variables.

Three kinds of scoped variants exist:

1. Verbs (i.e., functions) suffixed with `_all()` apply an operation on all variables.
  - ▶ e.g.: `summarize_all()`, `mutate_all()`
2. Verbs suffixed with `_at()` (e.g., `summarize_at()`) apply an operation on a subset of variables specified with quoting function `vars()`.
  - ▶ This quoting function accepts helpers functions like `starts_with()`
3. Verbs suffixed with `_if()` apply an operation on the subset of variables for which a predicate function returns TRUE.

Arguments of scoped variants

- ▶ `.tbl` A tbl object (data frame)
- ▶ `.funs` specifies which function(s) to perform (e.g., calculate mean)
  - ▶ Argument values: A function `fun`; a quosure style lambda `~ fun(.)`; or a list of either form (e.g., `'list(mean,min,max)`).
- ▶ `.vars` which variables to apply function to:
  - ▶ argument values: A list of columns generated by `vars()`, a character vector of column names, a numeric vector of column positions, or `NULL`.
- ▶ `.predicate` A predicate function to be applied to the columns or a logical vector. The variables for which `.predicate` is or returns `TRUE` are selected.
- ▶ `...` Additional arguments for function calls in `.funs`, evaluated once w/ tidy dots support

## What are “scoped” variants of a function?

Why/when use “scoped” variants of a function

- ▶ When you want to perform an operation on multiple variables without naming each individual variable

“verbs” (i.e., functions) from the `dplyr` package that have scoped variants `_all()`, `_at()`, and `_if()`

- ▶ `mutate()` and `transmute()` [see `?mutate_all`]
- ▶ `summarize()` [see `?summarize_all`]
- ▶ `filter()`
- ▶ `group_by()`
- ▶ `rename()` and `select()`
- ▶ `arrange()`

## Scoped variants of `summarize()`

**Description.** The “scoped variants” of `summarize()` apply `summarize()` to multiple variables. Three variants:

- ▶ `summarize_all()` affects every variable
- ▶ `summarize_at()` affects variables selected with a character vector or `vars()`
- ▶ `summarize_if()` affects variables selected with a predicate function

### Syntax

- ▶ `summarize_all(.tbl, .funs, ...)`
- ▶ `summarize_at(.tbl, .vars, .funs, ...)`
- ▶ `summarize_if(.tbl, .predicate, .funs, ...)`

### Arguments

- ▶ `.tbl` A `tbl` object (data frame)
- ▶ `.funs` specifies which function(s) to perform (e.g., calculate mean)
  - ▶ Argument values: A function `fun`; a quosure style lambda `~ fun(.)`; or a list of either form (e.g., `'list(mean,min,max)`).
- ▶ `.vars` which variables to apply function to:
  - ▶ argument values: A list of columns generated by `vars()`, a character vector of column names, a numeric vector of column positions, or `NULL`.
- ▶ `.predicate` A predicate function to be applied to the columns or a logical vector. The variables for which `.predicate` is or returns `TRUE` are selected.
- ▶ `...` Additional arguments for the function calls in `.funs`.
  - ▶ These are evaluated only once, with `tidy dots` support.

## summarize\_all() affects every variable

Syntax: `summarize_all(.tbl, .funs, ...)`

- ▶ `.tbl` A tbl object (data frame)
- ▶ `.funs` specifies which function(s) to perform. Argument values:
  - ▶ A function `fun` ; a quosure style lambda `~ fun(.)`; a list (e.g., `list(mean,min)`)
- ▶ `...` Additional arguments for function calls in `.funs` . These are evaluated once

### Task:

- ▶ For U. Pittsburgh (`univ_id = 215293`) events at public and private high schools, calculate the **mean** value of `med_inc` and `pct_white_zip` for each combination of `event_type` and `event_inst`

```
df_event %>%  
  filter(univ_id == 215293, event_type %in% c("private hs", "public hs")) %>%  
  select(event_type, event_inst, med_inc, pct_white_zip) %>%  
  group_by(event_type, event_inst) %>%  
  summarize_all(.funs = mean)
```

Try again, this time applying `na.rm = TRUE`

- ▶ this is an example of a `...` argument “for the function calls in `.funs` .”

```
df_event %>%  
  filter(univ_id == 215293, event_type %in% c("private hs", "public hs")) %>%  
  select(event_type, event_inst, med_inc, pct_white_zip) %>%  
  group_by(event_type, event_inst) %>%
```

## summarize\_all() affects every variable

Syntax: `summarize_all(.tbl, .funs, ...)`

- ▶ `.tbl` A tbl object (data frame)
- ▶ `.funs` specifies which function(s) to perform. Argument values:
  - ▶ A function `fun` ; a quosure style lambda `~ fun(.)`; a list (e.g., `list(mean,min)`)
- ▶ `...` Additional arguments for function calls in `.funs` . These are evaluated once

### Task:

- ▶ For U. Pittsburgh (`univ_id = 215293`) events at public and private high schools, calculate **mean** and **standard deviation** of `med_inc` and `pct_white_zip` for each combination of `event_type` and `event_inst`

```
df_event %>%  
  filter(univ_id == 215293, event_type %in% c("private hs", "public hs")) %>%  
  select(event_type, event_inst, med_inc, pct_white_zip) %>%  
  group_by(event_type, event_inst) %>%  
  summarize_all(.funs = list(mean, sd), na.rm = TRUE)
```

Use this syntax to control variable name suffixes:

- ▶ `.funs = list(var_name_suffix = function_name, ...)`

```
df_event %>%  
  filter(univ_id == 215293, event_type %in% c("private hs", "public hs")) %>%  
  select(event_type, event_inst, med_inc, pct_white_zip) %>%  
  group_by(event_type, event_inst) %>%
```

## summarize\_all() affects every variable

### Task:

- ▶ Same task as before, but now calculate **mean, standard deviation, min, and max** of `med_inc` and `pct_white_zip` for each combination of `event_type` and `event_inst`

```
df_event %>%
  filter(univ_id == 215293, event_type %in% c("private hs", "public hs")) %>%
  select(event_type, event_inst, med_inc, pct_white_zip) %>%
  group_by(event_type, event_inst) %>%
  summarize_all(.funs = list(avg = mean, std = sd, low = min, high = max),
    na.rm = TRUE)
#> # A tibble: 4 x 10
#>   event_type event_inst med_inc_avg pct_white_zip_a~ med_inc_std
#>   <chr>      <chr>          <dbl>          <dbl>          <dbl>
#> 1 private hs In-State          77115.          78.9           36559.
#> 2 private hs Out-State       103915.         63.3           44220.
#> 3 public hs  In-State          78408.          83.0           25841.
#> 4 public hs  Out-State       114212.         67.5           39745.
#> # ... with 5 more variables: pct_white_zip_std <dbl>, med_inc_low <dbl>,
#> #   pct_white_zip_low <dbl>, med_inc_high <dbl>, pct_white_zip_high <dbl>
```

## summarize\_all(), quosure style lambdas ~ func\_name(.)

Syntax: `summarize_all(.tbl, .funs, ...)`

- ▶ `.funs` specifies which function(s) to perform. Argument values:
  - ▶ A function `fun`; a quosure style lambda `~ fun(.)`; a list (e.g., `list(mean,min)`)

**Task:** Calculate mean, number of obs, and number of non-missing obs for variables

- ▶ Functions you specify within `.funs` require different options (e.g., some require `na.rm = TRUE` but others don't take arguments)
- ▶ Within `.funs` argument, specify functions using "quosure style lambda"
  - ▶ Syntax: `.funs = list(~ func_name(., options), ~ func_name(., options))`

```
df_event %>%  
  filter(univ_id == 215293, event_type %in% c("private hs","public hs")) %>%  
  select(event_type, event_inst,med_inc,pop_total) %>%  
  group_by(event_type,event_inst) %>%  
  summarize_all(.funs = list(~ mean(., na.rm = TRUE), ~ n(), ~ sum(!is.na(.))))
```

Specify suffix of variable name

```
df_event %>%  
  filter(univ_id == 215293, event_type %in% c("private hs","public hs")) %>%  
  select(event_type, event_inst,med_inc,pop_total) %>%  
  group_by(event_type,event_inst) %>%  
  summarize_all(.funs = list(avg = ~ mean(., na.rm = TRUE), nrow = ~ n(),  
    n_nonmiss = ~ sum(!is.na(.))))
```

## summarize\_at() affects selected variables

Syntax: `summarize_at(.tbl, .vars, .funs, ...)`

- ▶ `.tbl` A tbl object (data frame)
- ▶ `.vars` which variables to operate on. Argument values:
  - ▶ A list of columns generated by `vars()`, a character vector of column names, a numeric vector of column positions, or `NULL`.
- ▶ `.funs` specifies which function(s) to perform. Argument values:
  - ▶ A function `fun`; a quosure style lambda `~ fun(.)`; a list (e.g., `list(mean,min)`)
- ▶ `...` Additional arguments for function calls in `.funs`. These are evaluated once

**Task:** For U. Pittsburgh events at public and private high schools, calculate **mean**, **min**, and **max** of variables `med_inc` and `event_date` for each combination of `event_type` and `event_inst`

```
df_event %>%  
  filter(univ_id == 215293, event_type %in% c("private hs", "public hs")) %>%  
  group_by(event_type, event_inst) %>%  
  summarize_at(.vars = vars(med_inc, event_date),  
               .funs = list(avg = mean, low = min, high = max), na.rm = TRUE)
```

Alternative:

```
df_event %>%  
  filter(univ_id == 215293, event_type %in% c("private hs", "public hs")) %>%  
  group_by(event_type, event_inst) %>%  
  summarize_at(.vars = c("med_inc", "event_date"),  
               .funs = list(avg = mean, low = min, high = max), na.rm = TRUE)
```



## summarize\_if() affects variables that satisfy some condition

Useful if you want to apply functions to variables that are particular `type` or `class`

Syntax: `summarize_if(.predicate, .tbl, .funs, ...)`

- ▶ `.tbl` A `tbl` object (data frame)
- ▶ `.predicate` A predicate function to be applied to columns or a logical vector. The variables for which `.predicate` is or returns `TRUE` are selected.
- ▶ `.funs` specifies which function(s) to perform.
- ▶ `...` Additional arguments for function calls in `.funs`.

**Task:** For events by U. Pittsburgh at public and private high schools, calculate mean and standard deviation for **numeric variables**

*#First, which vars are numeric*

```
df_event %>%
  select(event_type, event_inst, instnm, school_id, med_inc, pct_white_zip) %>%
  glimpse()

df_event %>%
  filter(univ_id == 215293, event_type %in% c("private hs", "public hs")) %>%
  select(event_type, event_inst, instnm, school_id, med_inc, pct_white_zip) %>%
  group_by(event_type, event_inst) %>%
  summarize_if(.predicate = is.numeric, .funs = list(avg = mean, std = sd),
              na.rm = TRUE)
```

Attach aggregate measures to your data frame

## Attach aggregate measures to your data frame

We can attach aggregate measures to a data frame by using `group_by` without `summarize()`

What do I mean by “attaching aggregate measures to a data frame”?

- ▶ Calculate measures at the `by_group` level, but attach them to original object rather than creating an object with one row for each `by_group`

**Task:** Using `event_berk` data frame, create (1) a measure of average income across all events and (2) a measure of average income for each event type

- ▶ resulting object should have same number of observations as `event_berk`

Steps:

1. create measure of avg. income across all events without using `group_by()` or `summarize()` and assign as (new) object
2. Using object from previous step, create measure of avg. income across by event type using `group_by()` without `summarize()` and assign as new object

## Attach aggregate measures to your data frame

**Task:** Using `event_berk` data frame, create (1) a measure of average income across all events and (2) a measure of average income for each event type

1. Create measure of average income across all events

```
event_berk_temp <- event_berk %>%  
  arrange(event_date) %>% # sort by event_date (optional)  
  select(event_date, event_type, med_inc) %>% # select vars to be retained (optional)  
  mutate(avg_inc = mean(med_inc, na.rm=TRUE)) # create avg. inc measure  
  
dim(event_berk_temp)  
event_berk_temp %>% head(5)
```

2. Create measure of average income by event type

```
event_berk_temp <- event_berk_temp %>%  
  group_by(event_type) %>% # grouping by event type  
  mutate(avg_inc_type = mean(med_inc, na.rm=TRUE)) # create avg. inc measure  
  
str(event_berk_temp)  
event_berk_temp %>% head(5)
```

## Attach aggregate measures to your data frame

**Task:** Using `event_berk_temp` from previous question, create a measure that identifies whether `med_inc` associated with the event is higher/lower than average income for all events of that type

Steps:

1. Create measure of average income for each event type [already done]
2. Create 0/1 indicator that identifies whether median income at event location is higher than average median income for events of that type

```
# average income at recruiting events across all universities
event_berk_tempv2 <- event_berk_temp %>%
  mutate(gt_avg_inc_type = med_inc > avg_inc_type) %>%
  select(-(avg_inc)) # drop avg_inc (optional)
event_berk_tempv2 # note how med_ic = NA are treated
```

Same as above, but this time create integer indicator rather than logical

```
event_berk_tempv2 <- event_berk_tempv2 %>%
  mutate(gt_avg_inc_type = as.integer(med_inc > avg_inc_type))
event_berk_tempv2 %>% head(4)
```

## Student exercise

Task: is `pct_white_zip` at a particular event higher or lower than the average `pct_white_zip` for that `event_type` ?

- ▶ Note: all events attached to a particular `zip_code`
- ▶ `pct_white_zip` : pct of people in that `zip_code` who identify as white

Steps in task:

- ▶ Create measure of average pct white for each `event_type`
- ▶ Compare whether `pct_white_zip` is higher or lower than this average

## Student exercise solution

Task: is `pct_white_zip` at a particular event higher or lower than the average `pct_white_zip` for that `event_type` ?

```
event_berk_tempv3 <- event_berk %>%  
  arrange(event_date) %>% # sort by event_date (optional)  
  select(event_date, event_type, pct_white_zip) %>% #optional  
  group_by(event_type) %>% # grouping by event type  
  mutate(avg_pct_white = mean(pct_white_zip, na.rm=TRUE),  
         gt_avg_pctwhite_type = as.integer(pct_white_zip > avg_pct_white))  
event_berk_tempv3 %>% head(4)  
#> # A tibble: 4 x 5  
#>   event_date event_type pct_white_zip avg_pct_white gt_avg_pctwhite_type  
#>   <date>      <chr>          <dbl>          <dbl>          <int>  
#> 1 2017-04-11 other            37.2           49.7            0  
#> 2 2017-04-14 public hs         78.3           48.9            1  
#> 3 2017-04-23 private hs       84.7           61.0            1  
#> 4 2017-04-23 other            20.9           49.7            0
```