Module 5: Processing across rows

Managing and Manipulating Data Using R

Introduction

Logistics

Required reading for next week:

- GW 15.1 - 15.2 (factors) [this is like 2-3 pages]
- GW 20.6 - 20.7 (attributes and augmented vectors)
- GW 10 (tibbles) [this is like 3-4 pages]

What we will do today

1. Introduction

2. Introduce group_by() and summarise()
   - 2.1 group_by
   - 2.2 summarise()

3. Combining group_by() and summarise()
   - 3.1 summarise() and Counts (and logical vectors)
   - 3.2 summarise() and means
   - 3.3 summarise() and logical vectors, part II
   - 3.4 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/ksalazar3/HED696C_RClass/raw/master/data/recruitin
#load("../../data/recruiting/recruit_event_allvars.Rdata")
```

Processing across observations, introduction

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 and you want to create a variable for the average household income at visited versus non-visited high schools

Note

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

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         1.57 178732
#> 2 340147000250 NJ           1044       50         4.79  62288
#> 3 340561003796 NJ           1505      298        19.8  100684.
#> 4 340165005124 NJ           1900       43         2.26 160476.
#> 5 341341003182 NJ           1519      130         8.56 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 or rows" (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

Introduce group_by() and summarise()

Strategy for teaching processing across obs

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

- Usually, processing across observations requires using `group_by()` and `summarise()` together
- `group_by()` and `summarise()` usually aren't very useful by themselves (like peanut butter and jelly)

How I'll teach:

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

group_by

group_by()

`group_by()` converts a data frame object into groups. After grouping, functions performed on data frame are performed "by group"

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

Basic syntax:

- `group_by(object, vars to group by separated by commas)`

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`

```
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

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

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

```
df_event  %>% count()
df_event %>% group_by(univ_id) %>% count()
df_event %>% group_by(univ_id) %>% count() %>% str()
df_event %>% group_by(univ_id, event_type) %>% count()
df_event %>% group_by(univ_id, event_type) %>% count() %>% str()
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_grp <- df_event %>% group_by(univ_id, event_type, event_state) # using
class(df_event_grp)
#> [1] "grouped_df" "tbl_df"     "tbl"        "data.frame"
```

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

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

`group_by()` student exercise

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 CU Boulder   1439
#>  4 Cinci         679
#>  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
```

`group_by()` student exercise solutions

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
#> # i 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 %>%
  filter(med_inc > 75000) %>%
  group_by(instnm, event_type) %>%
  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
#> # i 70 more rows
```

summarise()

`summarise()` function

`summarise()` does calculations across rows; then collapses into single row

Usage (i.e., syntax): `summarise(.data, ...)`

Arguments

- `.data`: a data frame; omit if using `summarise()` after pipe `%>%`
- `...`: Name-value pairs of summary functions.
  - The name will be the name of the variable in the result.
  - Value should be expression that returns a single value like `min(x)`, `n()`

Value (what `summarise()` 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* `summarise()`, e.g., `mean()`, `n()`
- Count function `n()` takes no arguments; returns number of rows in group

Example: Count total number of events

```
summarise(df_event, num_events=n()) # without pipes
sum_object <- df_event %>% summarise(num_events=n()) # using pipes
df_event %>% summarise(num_events=n()) # using pipes
```

Investigate objects created by `summarise()`

Example: Count total number of events

```
df_event %>% summarise(num_events=n())
df_event %>% summarise(num_events=n()) %>% str()
```

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

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

- IMPORTANT NOTE: Many helper functions like `min()`, `max()`, `mean()` require us to explicitly deal with any missing values in the data
- If there are missing values, we need to specify R to "calculate" the helper function operation by ignoring the missing values: `na.rm = TRUE`
- If we have missing values and don't specify `na.rm = TRUE`, we will get all missing values on the new variable. Why? Missing values are contagious!
- We'll learn more on this later in elcture

Investigate objects created by `summarise()` cont...

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

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

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

Takeaway

- by default, objects created by `summarise()` are data frames that contain variables created within `summarise()` and one observation [per "by group"]

Retaining objects created by `summarise()`

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

```
event_temp <- df_event %>% summarise(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)
```

`summarise()` student exercise

1. What is the min value of `med_inc` across all events?
   - Hint: Use min(), don't forget to use `na.rm = TRUE`
   - Name the new variable you are creating "min_med_income"
2. What is the mean value of `fr_lunch` across all events?
   - Hint: Use mean(), don't forget to use `na.rm = TRUE`
   - Name the new variable you are creating "mean_fr_lunch"

`summarise()` student exercise [solutions]

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

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

`summarise()` student exercise [solutions]

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

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

Combining group_by() and summarise()

Combining `summarise()` and `group_by`

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

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

Example: Count the number of events across all rows (all universities); then within groups of `instnm` (for each university)

```
df_event %>% summarise(num_events=n())
df_event %>% group_by(instnm) %>% summarise(num_events=n())
```

- Investigate the object created above

```
df_event %>% group_by(instnm) %>% summarise(num_events=n()) %>% str()
```

- Or we could retain object for later use

```
event_by_univ <- df_event %>% group_by(instnm) %>% summarise(num_events=n())
str(event_by_univ)
event_by_univ # print
rm(event_by_univ)
```

Combining `summarise()` and `group_by`

Task

- Count number of recruiting events by event_type for each university

```
df_event %>% group_by(instnm, event_type) %>%
  summarise(num_events=n())
#> `summarise()` has grouped output by 'instnm'. You can override using the
#> `.groups` argument.

df_event %>% group_by(instnm, event_state, event_type) %>%
  summarise(num_events=n())
#> `summarise()` has grouped output by 'instnm', 'event_state'. You can override
#> using the `.groups` argument.

#investigate object created
df_event %>% group_by(instnm, event_type) %>%
  summarise(num_events=n()) %>% str()
#> `summarise()` has grouped output by 'instnm'. You can override using the
#> `.groups` argument.
```

Combining `summarise()` and `group_by`

Task

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

```
df_event %>% group_by(instnm, event_type) %>%
  summarise(num_events=n(),
    mean_pct_white=mean(pct_white_zip, na.rm = TRUE)
  )
#> `summarise()` has grouped output by 'instnm'. You can override using the
#> `.groups` argument.

#investigate object you created
df_event %>% group_by(instnm, event_type) %>%
  summarise(num_events=n(),
    mean_pct_white=mean(pct_white_zip, na.rm = TRUE)
  ) %>% str()
#> `summarise()` has grouped output by 'instnm'. You can override using the
#> `.groups` argument.
```

Combining `summarise()` and `group_by`

- Task: For recruiting events by UC Berkeley, count number of recruiting events by event_type

```
df_event %>% count(instnm)

df_event %>% filter(univ_id == 110635) %>%
  group_by(event_type) %>% summarise(num_events=n())
```

Let's create a dataset of recruiting events at UC Berkeley [we'll use this later]

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

event_berk %>% count(event_type)
```

The "char" variable `event_inst` equals "In-State" if event is in same state as the university

```
event_berk %>% arrange(event_date) %>%
  select(pid, event_date, event_type, event_state, event_inst) %>%
  slice(1:8)
#> # A tibble: 8 x 5
#>     pid event_date event_type  event_state event_inst
#>   <int> <date>     <chr>       <chr>       <chr>
#> 1 13100 2017-04-11 other       HI          Out-State
#> 2 13089 2017-04-14 public hs   GA          Out-State
#> 3 13088 2017-04-23 private hs  CT          Out-State
#> 4 13086 2017-04-23 other       CA          In-State
#> 5 13091 2017-04-24 private hs  NY          Out-State
#> 6 13087 2017-04-24 public hs   CA          In-State
```

summarise() and Counts (and logical vectors)

`summarise()` : Counts

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

```
event_berk %>% group_by(event_type, event_inst) %>%
  summarise(num_events=n())
#> `summarise()` has grouped output by 'event_type'. You can override using the
#> `.groups` argument.
```

Object not retained unless we **assign**

```
berk_temp <- event_berk %>% group_by(event_type, event_inst) %>%
  summarise(num_events=n())
#> `summarise()` has grouped output by 'event_type'. You can override using the
#> `.groups` argument.
berk_temp
typeof(berk_temp)
str(berk_temp)
```

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

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

berk_temp2 <- event_berk %>% group_by(event_type, event_inst) %>% count()

berk_temp == berk_temp2 # TAKEAWAY: these two objects are identical!
rm(berk_temp,berk_temp2)
```

`summarise()` : 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) %>%
  summarise(
    n_events = n(),
    n_miss_fr_lunch = sum(is.na(fr_lunch)),
    n_nonmiss_fr_lunch = sum(!is.na(fr_lunch))
  )
```

`summarise()` and count student exercise

Use one code chunk for this exercise. Tackle this one step at a time, building your code by adding piped operations for each step, and then 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 summarise 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.

`summarise()` and count student exercise SOLUTION

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

```
event_berk %>%
  filter(event_state == "VA") %>%
  group_by(event_type) %>%
  summarise(
    n_events = n(),
    n_miss_inc = sum(is.na(med_inc)),
    n_nonmiss_inc = sum(!is.na(med_inc))) %>%
  arrange(desc(n_events))
#> # 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
```

summarise() and means

`summarise()` : means

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

```
event_berk %>% group_by(event_inst, event_type) %>% summarise(
  n_events=n(),
  mean_inc=mean(med_inc, na.rm = TRUE),
  mean_pct_white=mean(pct_white_zip, na.rm = TRUE))
#> `summarise()` has grouped output by 'event_inst'. You can override using the
#> `.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
```

`summarise()` : means and `na.rm` argument

Default behavior of "aggregation functions" (e.g., `summarise()`)

- if *input* has any missing values (`NA`), than output will be missing. Missing values are contagious!

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) %>% summarise(
  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()` has grouped output by 'event_inst'. You can override using the
#> `.groups` argument.
#na.rm = TRUE
event_berk %>% group_by(event_inst, event_type) %>% summarise(
  n_events=n(),
  n_miss_inc = sum(is.na(med_inc)),
  mean_inc=mean(med_inc, 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`)
   1.2 Create vars for mean percent for each racial/ethnic group

Student exercise solutions

```
event_berk %>% group_by(instnm, event_inst, event_type) %>%
  summarise(
  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)) %>%
  head(6)
#> `summarise()` has grouped output by 'instnm', 'event_inst'. You can override
#> using the `.groups` argument.
#> # A tibble: 6 x 8
#>   instnm     event_inst event_type n_events n_miss_white mean_white n_miss_bl
#>   <chr>      <chr>      <chr>          <int>        <int>      <dbl>        <i
#> 1 UC Berkel~ In-State   2yr colle~       111          106       40.1
#> 2 UC Berkel~ In-State   4yr colle~        14           12       58.0
#> 3 UC Berkel~ In-State   other             49           48       37.6
#> 4 UC Berkel~ In-State   private hs        35           35       48.4
#> 5 UC Berkel~ In-State   public hs        259          258       39.6
#> 6 UC Berkel~ Out-State  2yr colle~        1            1       89.7
#> # i 1 more variable: mean_black <dbl>
```

summarise() and logical vectors, part II

`summarise()` : 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`, create a new object `gt50p_lat_bl` with 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

```
gt50p_lat_bl <- event_berk %>% group_by (event_inst, event_type) %>%
  summarise(
    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()` has grouped output by 'event_inst'. You can override using the
#> `.groups` argument.
gt50p_lat_bl # print object
str(gt50p_lat_bl)
```

`summarise()` : logical vectors to count *proportions*

Synatx: `group_by(vars) %>% summarise(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
  summarise(
    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 $100
#> # 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
```

`summarise()` : 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
  summarise(
    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)) # proportion visits to $100K+ commmuniti
#> # A tibble: 2 x 4
#>   event_inst n_events n_nonmiss_inc p_incgt100k
#>   <chr>         <int>         <int>       <dbl>
#> 1 In-State        259           256      NA
#> 2 Out-State       183           183       0.519
```

`summarise()` : Other "helper" functions

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

Common functions to use with `summarise()` :

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

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

`summarise()` : 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) %>%
  summarise(max_inc = max(med_inc))
#> `summarise()` has grouped output by 'event_type'. You can override using the
#> `.groups` argument.

event_berk %>% group_by(event_type, event_inst) %>%
  summarise(max_inc = max(med_inc, na.rm = TRUE))
#> `summarise()` has grouped output by 'event_type'. You can override using the
#> `.groups` argument.
```

What did we do wrong in the first attempt?

`summarise()` : 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) %>%
  summarise(
    n_events = n(),
    date_first= first(event_date),
    date_last= last(event_date),
    date_50th= nth(event_date, 50)
  )
```

Student exercise

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

Specific task:

- arrange (i.e., sort) by `event_type` and `event_datetme_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) %>%
  summarise(
    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))
#> # 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
```

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 summarise()

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 `summarise()` and assign as (new) object
2. Using object from previous step, create measure of avg. income across by event type using `group_by()` without `summarise()` 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 (optio
  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
```