Managing and Manipulating Data Using R

Lecture 4

Ozan Jaquette

Introduction

Libraries we will use today

```
library(tidyverse)
#> -- Attaching packages -----------------------------------------------------
#> v ggplot2 3.0.0     v purrr   0.2.5
#> v tibble  1.4.2     v dplyr   0.7.6
#> v tidyr   0.8.1     v stringr 1.3.1
#> v readr   1.1.1     v forcats 0.3.0
#> -- Conflicts ------------------------------------------------------ tidyv
#> x dplyr::filter() masks stats::filter()
#> x dplyr::lag()    masks stats::lag()
```

Data we will use today

Data on off-campus recruiting events by public universities

- Object `df_event`
  - One observation per university, recruiting event
- Object `df_event`
  - One observation per high school (visited and non-visited)

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

#load dataset with one obs per recruiting event
load("../../data/recruiting/recruit_event_somevars.Rdata")

#load dataset with one obs per high school
load("../../data/recruiting/recruit_school_somevars.Rdata")

load("../../data/prospect_list/western_washington_college_board_list.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
- Average household income at visited versus non-visited high schools

Processing across variables vs. processing across observations

Visits by UC Berkely 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
- 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

`group_by()` and `summarise()`

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
- event type (e.g., public HS, private HS, hotel)
- state

group_by()

Group `df_event` data by university, event type, and event state

- group_by doesn't do much by itself; just prints data

```
group_by(df_event, univ_id, event_type, event_state)

df_event %>% group_by(univ_id, event_type, event_state) # using pipes
```

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)
```

`summarise()`

`summarise()` function performs calculations across rows of a data frame and then collapses the data frame to a single row

Basic syntax [see documentation]:

- `summarise(object, summarise functions separated by commas)`
- summarise functions include: `n()`, `mean()`, `first()`, etc.

Simple example (output omitted)

```
summarise(df_event, num_events=n())
df_event %>% summarise(num_events=n()) # using pipes
```

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      17976   88774.
rm(event_temp)
```

I'll explain `na.rm = TRUE` later

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

Number of recruiting events for each university

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

Number of recruiting events by event_type for each university

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

Number of events and avg. pct White by event_type for each university

```
df_event %>% group_by(instnm, event_type) %>%
  summarise(num_events=n(),
    mean_pct_white=mean(pct_white_zip, na.rm = TRUE)
  )
```

Combining `summarise()` and `group_by`

Recruiting events by UC Berkeley

```
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

The 0/1 variable `event_inst` equals 1 if event is in same state as the university

```
#event_berk %>% group_by(event_type, event_inst) %>% select(pid, event_date, eve
event_berk %>% arrange(event_date) %>% select(pid, event_date, event_type, event
#> # A tibble: 8 x 5
#>     pid event_date event_type  event_state event_inst
#>   <int> <date>     <fct>       <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
#> 7 13092 2017-04-25 other       NY          Out-State
#> 8 13099 2017-04-25 2yr college CA          In-State
```

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())
```

Object not retained unless we **assign**

```
berk_temp <- event_berk %>% group_by(event_type, event_inst) %>%
  summarise(num_events=n())
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()
event_berk %>% group_by(event_type) %>% count(event_inst) # same

berk_temp2 <- event_berk %>% group_by(event_type, event_inst) %>% count()
berk_temp == berk_temp2
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))
#> [1] 15
sum(c(TRUE,TRUE,FALSE,FALSE))
#> [1] 2
```

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

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

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

```
event_berk %>% group_by(event_type) %>%
  summarise(
    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))
  )
```

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)) %>% head(5)
#> # A tibble: 5 x 5
#> # Groups:   event_inst [1]
#>   event_inst event_type  n_events mean_inc mean_pct_white
#>   <chr>      <fct>          <int>    <dbl>          <dbl>
#> 1 In-State   public hs        260   87146.           39.8
#> 2 In-State   private hs        36   94133.           48.0
#> 3 In-State   2yr college      107   79144.           40.5
#> 4 In-State   4yr college       12  148587.           55.1
#> 5 In-State   other             50   73218.           35.9
```

I'll talk about na.rm = TRUE on next slide

`summarise()`: means and `NA` values

The default behavior of "aggregation functions" (e.g., `summarise()`) is if the **input** has any missing value (`NA`) than the output will be missing.

`na.rm` (in words "remove `NA`") is an option available in many functions.

- `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

```r
#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))
#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),
  n_miss_frlunch = sum(is.na(fr_lunch)),
  mean_fr_lunch=mean(fr_lunch, na.rm = TRUE))
```

Student exercise

summarise(): counts with logical vectors, part II

Application: count number in a group that satisfy some condition

Task: For each combination of `event_type` and `event_inst`, how many visits to communities that are majority Latinx or Black?

```
#event_berk %>% select(pct_black_zip, pct_hispanic_zip)
event_berk %>% group_by (event_inst, event_type) %>% summarise(
  n_events=n(), # number of events by group
  n_nonmiss_latbl = sum(!is.na(pct_black_zip) & !is.na(pct_hispanic_zip)), # w/
  n_majority_latbl= sum(pct_black_zip+ pct_hispanic_zip>50, na.rm = TRUE)) # num
#> # A tibble: 10 x 5
#> # Groups:   event_inst [?]
#>    event_inst event_type  n_events n_nonmiss_latbl n_majority_latbl
#>    <chr>      <fct>          <int>           <int>            <int>
#>  1 In-State   public hs        260             259               88
#>  2 In-State   private hs        36              36                7
#>  3 In-State   2yr college      107             102               27
#>  4 In-State   4yr college       12              10                0
#>  5 In-State   other             50              49               17
#>  6 Out-State  public hs        184             184               27
#>  7 Out-State  private hs       135             133               20
#>  8 Out-State  2yr college        1               1                0
#>  9 Out-State  4yr college        3               2                0
#> 10 Out-State  other             90              87               20
```

`summarise()`: proportions with logical values

Application: count proportion of obs in group that satisfy some condition

- 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 $100K
#> # A tibble: 2 x 4
#>   event_inst n_events n_nonmiss_inc p_incgt100k
#>   <chr>         <int>         <int>       <dbl>
#> 1 In-State        260           257       0.272
#> 2 Out-State       184           184       0.516
```

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

Student exercise

`summarise()`: Other functions

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))
#> # A tibble: 10 x 3
#> # Groups:   event_type [?]
#>    event_type  event_inst max_inc
#>    <fct>       <chr>        <dbl>
#>  1 public hs   In-State        NA
#>  2 public hs   Out-State  223556.
#>  3 private hs  In-State   250001
#>  4 private hs  Out-State       NA
#>  5 2yr college In-State        NA
#>  6 2yr college Out-State  153070.
#>  7 4yr college In-State        NA
#>  8 4yr college Out-State       NA
#>  9 other       In-State        NA
#> 10 other       Out-State       NA
```

What did we do wrong here?

`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 `event_berk` by [`arrange()`] by `event_type` and `event_datetime_start`, what is the value of `event_date` for:

- first event for each event type?
- the last eventfor each event type?
- the `10`th 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

something that involves whether visits adhered to a certain pattern? e.g., visited org of type 1 and then org of type 2 in succession?

Attach aggregate measures to your data frame

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

Example 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

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)
```

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

event_berk_temp %>% head(5)
```

Attach aggregate measures to your data frame

Task: 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 measure that compares income to average income for event 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
```

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)
#> # A tibble: 4 x 5
#> # Groups:   event_type [3]
#>   event_date event_type med_inc avg_inc_type gt_avg_inc_type
#>   <date>     <fct>        <dbl>        <dbl>           <int>
#> 1 2017-04-11 other       62022.       70922.               0
#> 2 2017-04-14 public hs  125056.       93956.               1
#> 3 2017-04-23 private hs  78897        88695.               0
#> 4 2017-04-23 other       55127        70922.               0
```

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_zup is higher or lower than this average