Lecture 3: Pipes and creating variables using `mutate()`

EDUC 263: Managing and Manipulating Data Using R

Ozan Jaquette

1 Introduction

What we will do today

1. Introduction
    - 1.1 Finish lecture 2, filter and arrange (i.e., sort)
    - 1.2 Data for lecture 3

2. Pipes

3. Creating variables using mutate
    - 3.1 Introduce mutate() function
    - 3.2 Using ifelse() function within mutate()
    - 3.3 Using recode() function within mutate()
    - 3.4 Using case_when() function within mutate()
    - 3.5 Alternative approach to creating 0/1 indicators

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)`

1.1 Finish lecture 2, filter and arrange (i.e., sort)

Load data for lecture 2

Data on off-campus recruiting events by public universities

```
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")
```

Object `df_event`

- Off-campus recruiting project; one obs per university, recruiting event

Object `df_event`

- Off-campus recruiting project; one obs per high school (visited and non-visited)

Work through lecture 2 slides on `filter()` and `arrange()`

1.2 Data for lecture 3

Lecture 3 data: prospects purchased by Western Washington U.

The "Student list" business

- Universities identify/target "prospects" by buying "student lists" from College Board/ACT (e.g., $.40 per prospect)
- Prospect lists contain contact info (e.g., address, email), academic achievement, socioeconomic, demographic characteristics
- Universities choose which prospects to purchase by filtering on criteria like zip-code, GPA, test score range, etc.

```
#load prospect list data
load("../../data/prospect_list/wwlist_merged.RData")
```

Object `wwlist`

- De-identified list of prospective students purchased by Western Washington University from College Board
- We collected these data using FOIA request
  - ASIDE: Become an expert on collecting data via FOIA requests and you will become a superstar!

Lecture 3 data: prospects purchased by Western Washington U.

Observations on `wwlist`

- each observation represents a prospective student

```
typeof(wwlist)
#> [1] "list"
dim(wwlist)
#> [1] 268396     31
```

Variables on `wwlist`

- some vars provide de-identified data on individual prospects
  - e.g., `psat_range`, `state`, `sex`, `ethn_code`
- some vars provide data about zip-code student lives in
  - e.g., `med_inc`, `pop_total`, `pop_black`
- some vars provide data about school student enrolled in
  - e.g., `fr_lunch` is number of students on free/reduced lunch
  - note: bad merge between prospect-level data and school-level data

```
names(wwlist)
str(wwlist)
```

2 Pipes

What are “pipes”, %>%

Pipes are a means of perfoming multiple steps in a single line of code

- Pipes are part of **tidyverse** suite of packages, not **base R**
- When writing code, the pipe symbol is `%>%`
- Basic flow of using pipes in code:
  - `object %>% some_function %>% some_function, \ldots`
- Pipes work from left to right:
  - The object/result from left of `%>%` pipe symbol is the input of function to the right of the `%>%` pipe symbol
  - In turn, the resulting output becomes the input of the function to the right of the next `%>%` pipe symbol

Intuitive mnemonic device for understanding pipes

- whenever you see a pipe `%>%` think of the words “**and then...**”
- Example: `wwlist %>% filter(firstgen == "Y")`
  - in words: start with object `wwlist` **and then** filter first generation students

Do task with and without pipes

Task:

- Using object `wwlist` print data for "first-generation" prospects (`firstgen == "Y"`)

```
filter(wwlist, firstgen == "Y") # without pipes
wwlist %>% filter(firstgen == "Y") # with pipes
```

Comparing the two approaches:

- In the "without pipes" approach, the object is the first argument `filter()` function
- In the "pipes" approach, you don't specify the object as the first argument of `filter()`
  - Why? Because `%>%` "pipes" the object to the left of the `%>%` operator into the function to the right of the `%>%` operator

Main takeaway:

- When writing code using pipes, functions to right of `%>%` pipe operator should not explicitly name object that is the input to the function.
- Rather, object to the left of `%>%` pipe operator is automatically the input.

More intuition on the pipe operator, `%>%`

The pipe operator "pipes" (verb) an object from left of `%>%` operator into the function to the right of the %>% operator

Example:

```
str(wwlist) # without pipe

wwlist %>% str() # with pipe
```

Do task with and without pipes

Task: Using object `wwlist`, print data for "first-gen" prospects for selected variables [output omitted]

```
#Without pipes
select(filter(wwlist, firstgen == "Y"), state, hs_city, sex)
#With pipes
wwlist %>% filter(firstgen == "Y") %>% select(state, hs_city, sex)
```

Comparing the two approaches:

- In the "without pipes" approach, code is written "inside out"
  - The first step in the task – identifying the object – is the innermost part of code
  - The last step in task – selecting variables to print – is the outermost part of code
- In "pipes" approach the left-to-right order of code matches how we think about the task
  - First, we start with an object ***and then*** (`%>%`) we use `filter()` to isolate first-gen students ***and then*** (`%>%`) we select which variables to print

Note the object "piped" into `select()` from `filter()`

```
wwlist %>% filter(firstgen == "Y") %>% str()
```

Do task with and without pipes

Task:

- Count the number “first-generation” prospects from the state of Washington

Without pipes

```
count(filter(wwlist, firstgen == "Y", state == "WA"))
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1 32428
```

With pipes

```
wwlist %>% filter(firstgen == "Y", state == "WA") %>% count()
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1 32428
```

Do task with and without pipes

Task: Create frequency table of `school_type` for non first-gen prospects from WA

without pipes

```
wwlist_temp <- filter(wwlist, firstgen == "N", state == "WA")
table(wwlist_temp$school_type, useNA = "always")
#>
#> private  public    <NA>
#>      11   46146   12489
rm(wwlist_temp) # cuz we don't need after creating table
```

With pipes

```
wwlist %>% filter(firstgen == "N", state == "WA") %>% count(school_type)
#> # A tibble: 3 x 2
#>   school_type     n
#>   <chr>       <int>
#> 1 private        11
#> 2 public      46146
#> 3 <NA>        12489
```

Comparison of two approaches

- without pipes, task requires multiple lines of code; this is quite common
  - first line creates object; second line analyzes object
- with pipes, task can be completed in one line of code and you aren't left with objects you don't care about

Student exercises with pipes

PATRICIA CAN YOU CREATE A COUPLE STUDENT EXERCISES FOR PIPES

3 Creating variables using mutate

Our plan for learning how to create new variables

Recall that `dplyr` package within `tidyverse` provide a set of functions that can be described as "verbs": **subsetting**, **sorting**, and **transforming**

What we've done	Where we're going
Subsetting data	**Transforming data**
- `select()` variables	- `mutate()` creates new variables
- `filter()` observations	- `summarize()` calculates across rows
Sorting data	- `group_by()` to calculate across rows within groups
- `arrange()`	

Today

- we'll use `mutate()` to create new variables based on calculations across columns within a row

Next week

- we'll combine `mutate()` with `summarize()` and `group_by()` to create variables based on calculations across rows

Create new data frame based on `df_school_all`

Data frame `df_school_all` has one obs per US high school and then variables identifying number of visits by particular universities

```
load("../../data/recruiting/recruit_school_allvars.Rdata")
names(df_school_all)
#>  [1] "state_code"         "school_type"        "ncessch"
#>  [4] "name"               "address"            "city"
#>  [7] "zip_code"           "pct_white"          "pct_black"
#> [10] "pct_hispanic"       "pct_asian"          "pct_amerindian"
#> [13] "pct_other"          "num_fr_lunch"       "total_students"
#> [16] "num_took_math"      "num_prof_math"      "num_took_rla"
#> [19] "num_prof_rla"       "avgmedian_inc_2564" "latitude"
#> [22] "longitude"          "visits_by_196097"   "visits_by_186380"
#> [25] "visits_by_215293"   "visits_by_201885"   "visits_by_181464"
#> [28] "visits_by_139959"   "visits_by_218663"   "visits_by_100751"
#> [31] "visits_by_199193"   "visits_by_110635"   "visits_by_110653"
#> [34] "visits_by_126614"   "visits_by_155317"   "visits_by_106397"
#> [37] "visits_by_149222"   "visits_by_166629"   "total_visits"
#> [40] "inst_196097"        "inst_186380"        "inst_215293"
#> [43] "inst_201885"        "inst_181464"        "inst_139959"
#> [46] "inst_218663"        "inst_100751"        "inst_199193"
#> [49] "inst_110635"        "inst_110653"        "inst_126614"
#> [52] "inst_155317"        "inst_106397"        "inst_149222"
#> [55] "inst_166629"
```

Create new data frame based on `df_school_all`

Let's create new version of this data frame, called `school_v2`, which we'll use to introduce how to create new variables

```
school_v2 <- df_school_all %>%
  select(-contains("inst_")) %>% # remove vars that start with "inst_"
  rename(
    visits_by_berkeley = visits_by_110635,
    visits_by_boulder = visits_by_126614,
    visits_by_bama = visits_by_100751,
    visits_by_stonybrook = visits_by_196097,
    visits_by_rutgers = visits_by_186380,
    visits_by_pitt = visits_by_215293,
    visits_by_cinci = visits_by_201885,
    visits_by_nebraska = visits_by_181464,
    visits_by_georgia = visits_by_139959,
    visits_by_scarolina = visits_by_218663,
    visits_by_ncstate = visits_by_199193,
    visits_by_irvine = visits_by_110653,
    visits_by_kansas = visits_by_155317,
    visits_by_arkansas = visits_by_106397,
    visits_by_sillinois = visits_by_149222,
    visits_by_umass = visits_by_166629,
    num_took_read = num_took_rla,
    num_prof_read = num_prof_rla,
    med_inc = avgmedian_inc_2564)

names(school_v2)
```

3.1 Introduce mutate() function

Introduce `mutate()` function

`mutate()` is **tidyverse** approach to creating variables (not **Base R** approach)

Description of `mutate()`

- `mutate()` creates new columns (variables) that are functions of existing columns
- After creating a new variable using `mutate()`, every row of data is retained
- `mutate()` works best with pipes `%>%`

Task:

- Using data frame `school_v2` create new variable that measures the pct of students on free/reduced lunch (output omitted)

```
school_sml <- school_v2 %>% # create new dataset with fewer vars; not necessary
  select(ncessch, school_type, num_fr_lunch, total_students)

school_sml %>%
  mutate(pct_fr_lunch = num_fr_lunch/total_students) # create new var

rm(school_sml)
```

Syntax for `mutate()`

Let's spend a couple minutes looking at help fule for `mutate()`

```
?mutate
```

Usage (i.e., syntax)

- `mutate(.data,...)`

Arguments

- `.data` : a data frame
  - if using `mutate()` after pipe operator `%>%` , then this argument can be omitted
    - Why? Because data frame object to left of `%>%` "piped in" to first argument of `mutate()`
- `...` : expressions used to create new variables
  - Can create multiple variables at once

Value

- returns an object that contains the original input data frame and new variables that were created by `mutate()`

Useful functions (i.e., "helper functions")

- These are standalone functions can be called *within* `mutate()`
  - e.g., `if_else()` , `recode()` , `case_when()`
- will show examples of this in subsequent slides

Introduce `mutate()` function

New variable not retained unless we **assign** `<-` it to an object (existing or new)

`mutate()` without assignment

```
school_v2 %>% mutate(pct_fr_lunch = num_fr_lunch/total_students)

names(school_v2)
```

`mutate()` with assignment

```
school_v2_temp <- school_v2 %>%
  mutate(pct_fr_lunch = num_fr_lunch/total_students)

names(school_v2_temp)
rm(school_v2_temp)
```

Aside: Base R approach to creating new variables

Task:

- Create measure of percent of students on free-reduced lunch

dplyr/tidyverse approach

```
school_v2_temp <- school_v2 %>%
  mutate(pct_fr_lunch = num_fr_lunch/total_students)
```

Base R approach

```
school_v2_temp <- school_v2 # create copy of dataset; not necessary

school_v2_temp$pct_fr_lunch <- school_v2_temp$num_fr_lunch/school_v2_temp$total_

names(school_v2_temp)
rm(school_v2_temp)
```

Good to know both Base R and tidyverse approaches; sometimes you need to use one or the other

- But overwhelming to learn both approaches at once
- We'll focus mostly on learning tidyverse approaches
- But I'll try to work-in opportunities to learn Base R approach

`mutate()` can create multiple variables at once

`mutate()` can create multiple variables at once

```
school_v2 %>%
  mutate(pct_fr_lunch = num_fr_lunch/total_students,
         pct_prof_math= num_prof_math/num_took_math) %>%
  select(num_fr_lunch, total_students, pct_fr_lunch,
         num_prof_math, num_took_math, pct_prof_math)
```

Or we could write code this way:

```
school_v2 %>%
  select(num_fr_lunch, total_students, num_prof_math, num_took_math) %>%
  mutate(pct_fr_lunch = num_fr_lunch/total_students,
         pct_prof_math= num_prof_math/num_took_math)
```

Student exercise using mutate()

PATRICIA ADD?

3.2 Using ifelse() function within mutate()

Using `ifelse()` function within `mutate()`

```
?if_else
```

Description

- if `condition` `TRUE`, assign a value; if `condition` `FALSE` assign a value

Usage (i.e., syntax)

- `if_else(logical condition, true, false, missing = NULL)`

Arguments

- `logical condition`: a condition that evaluates to `TRUE` or `FALSE`
- `true`: value to assign if condition `TRUE`
- `false`: value to assign if condition `FALSE`

Value

- "Where condition is TRUE, the matching value from true, where it's FALSE, the matching value from false, otherwise NA."
- missing values from "input" var are assigned missing values in "output var", unless you specify otherwise

Example: Create 0/1 indicator of whether got at least one visit from Berkeley

```
school_v2 %>%
  mutate(got_visit_berkeley = ifelse(visits_by_berkeley>0,1,0)) %>%
  count(got_visit_berkeley)
```

Using `ifelse()` function within `mutate()`

Task

- Create 0/1 indicator of school has median income greater than $100,000

Usually a good idea to investigate "input" variables before creating analysis vars

```
school_v2 %>% count(med_inc) # this isn't very helpful

school_v2 %>% filter(is.na(med_inc)) %>% count(med_inc) # shows number of obs w/
```

Create variable

```
school_v2 %>% select(med_inc) %>%
  mutate(inc_gt_100k= ifelse(med_inc>100000,1,0)) %>%
  count(inc_gt_100k) # note how NA values of med_inc treated
#> # A tibble: 3 x 2
#>   inc_gt_100k     n
#>         <dbl> <int>
#> 1           0 18632
#> 2           1  2045
#> 3          NA   624
```

Using `ifelse()` function within `mutate()`

Task

- Create 0/1 indicator variable `nonmiss_math` which indicates whether school has non-missing values for the variable `num_took_math`
  - note: `num_took_math` refers to number of students at school that took state math proficiency test

Usually a good to investigate "input" variables before creating analysis vars

```
school_v2 %>% count(num_took_math) # this isn't very helpful
school_v2 %>% filter(is.na(num_took_math)) %>% count(num_took_math) # shows numb
```

Create variable

```
school_v2 %>% select(num_took_math) %>%
  mutate(nonmiss_math= ifelse(!is.na(num_took_math),1,0)) %>%
  count(nonmiss_math) # note how NA values treated
#> # A tibble: 2 x 2
#>   nonmiss_math     n
#>          <dbl> <int>
#> 1            0  4103
#> 2            1 17198
```

Student exercises `ifelse()`

PATRICIA FINISH CREATE STUDENT EXERCISES FOR IFELSE

Task

- Create 0/1 indicator variable `in_state_berkeley` that equals `1` if the high school is in the same state as UC Berkeley (i.e., `state_code=="CA"`)

```
school_v2 %>% mutate(in_state_berkeley=ifelse(state_code=="CA",1,0)) %>%
  count(in_state_berkeley)
```

Task

- create 0/1 indicator `berkeley_and_irvine` of whether a school got at least one visit from UC Berkeley **AND** from UC Irvine

```
school_v2 %>%
  mutate(berkeley_and_irvine=ifelse(visits_by_berkeley>0 & visits_by_irvine>0,1,
  count(berkeley_and_irvine)
```

Task

- create 0/1 indicator `berkeley_or_irvine` of whether a school got at least one visit from UC Berkeley **OR** from UC Irvine

```
school_v2 %>%
  mutate(berkeley_or_irvine=ifelse(visits_by_berkeley>0 | visits_by_irvine>0,1,0
  count(berkeley_or_irvine)
```

3.3 Using recode() function within mutate()

Using `recode()` function within `mutate()`

Description: Recode values of a variable

Usage (i.e., syntax)

- recode(.x, ..., .default = NULL, .missing = NULL)

Arguments [see help file for further details]

- `.x` A vector (e.g., variable) to modify
- `...` Specifications for recode, of the form `current_value = new recoded value`
- `.default` : If supplied, all values not otherwise matched given this value.
- `.missing` : If supplied, any missing values in .x replaced by this value.

Example: Using data frame `wwlist` , create new 0/1 indicator `public_school` from variable `school_type`

```
str(wwlist$school_type)
wwlist %>% count(school_type)

wwlist_temp <- wwlist %>% select(school_type) %>%
  mutate(public_school = recode(school_type,"public" = 1, "private" = 0))

wwlist_temp %>% head(n=10)
str(wwlist_temp$public_school)
wwlist_temp %>% count(public_school)
rm(wwlist_temp)
```

Using `recode()` function within `mutate()`

Recoding `school_type` could have been accomplished using `if_else()`

- Use `recode()` when new variable has more than two categories

Task: Create `school_catv2` based on `school_category` with these categories:

- "regular"; "alternative"; "special"; "vocational"

Investigate input var

```
str(wwlist$school_category)
wwlist %>% count(school_category)
```

Recode

```
wwlist_temp <- wwlist %>% select(school_category) %>%
  mutate(school_catv2 = recode(school_category,
    "Alternative Education School" = "alternative",
    "Alternative/other" = "alternative",
    "Regular elementary or secondary" = "regular",
    "Regular School" = "regular",
    "Special Education School" = "special",
    "Special program emphasis" = "special",
    "Vocational Education School" = "vocational")
  )
str(wwlist_temp$school_catv2)
wwlist_temp %>% count(school_catv2)
wwlist %>% count(school_category)
```

Using `recode()` within `mutate()` [do in pairs/groups]

Task: Create `school_catv2` based on `school_category` with these categories:

- "regular"; "alternative"; "special"; "vocational"
- This time use the `.missing` argument to recode `NAs` to "unknown"

```
wwlist_temp <- wwlist %>% select(school_category) %>%
  mutate(school_catv2 = recode(school_category,
    "Alternative Education School" = "alternative",
    "Alternative/other" = "alternative",
    "Regular elementary or secondary" = "regular",
    "Regular School" = "regular",
    "Special Education School" = "special",
    "Special program emphasis" = "special",
    "Vocational Education School" = "vocational",
    .missing = "unknown")
  )
str(wwlist_temp$school_catv2)
wwlist_temp %>% count(school_catv2)
wwlist %>% count(school_category)
rm(wwlist_temp)
```

Using `recode()` within `mutate()` [do in pairs/groups]

Task: Create `school_catv2` based on `school_category` with these categories:

- "regular"; "alternative"; "special"; "vocational"
- This time use the `.default` argument to assign the value "regular"

```
wwlist_temp <- wwlist %>% select(school_category) %>%
  mutate(school_catv2 = recode(school_category,
    "Alternative Education School" = "alternative",
    "Alternative/other" = "alternative",
    "Special Education School" = "special",
    "Special program emphasis" = "special",
    "Vocational Education School" = "vocational",
    .default = "regular")
  )
str(wwlist_temp$school_catv2)
wwlist_temp %>% count(school_catv2)
wwlist %>% count(school_category)
rm(wwlist_temp)
```

Using `recode()` within `mutate()` [do in pairs/groups]

Task: Create `school_catv2` based on `school_category` with these categories:

- This time create a numeric variable rather than character:
  - `1` for “regular”; `2` for “alternative”; `3` for “special”; `4` for “vocational”

```
wwlist_temp <- wwlist %>% select(school_category) %>%
  mutate(school_catv2 = recode(school_category,
    "Alternative Education School" = 2,
    "Alternative/other" = 2,
    "Regular elementary or secondary" = 1,
    "Regular School" = 1,
    "Special Education School" = 3,
    "Special program emphasis" = 3,
    "Vocational Education School" = 4)
  )
str(wwlist_temp$school_catv2)
wwlist_temp %>% count(school_catv2)
wwlist %>% count(school_category)
rm(wwlist_temp)
```

Student exercise using `recode()` within `mutate()`

PATRICIA CREATE STUDENT EXERCISE

A COUPLE CHOICES [OR CHOOSE YOUR OWN]

- in wwlist object, recode the variable psat_range or recode the variable ethn_code

```
str(wwlist$psat_range)
#>  chr [1:268396] "930-1160" "1270-1520" "990-1260" "1170-1520" ...
wwlist %>% count(psat_range)
#> # A tibble: 8 x 2
#>   psat_range     n
#>   <chr>      <int>
#> 1 1030-1160  45708
#> 2 1030-1520  67192
#> 3 1170-1520  48982
#> 4 1270-1520   8348
#> 5 930-1160   17387
#> 6 930-1260   15660
#> 7 990-1260   27628
#> 8 <NA>       37491
str(wwlist$ethn_code)
#>  Factor w/ 11 levels "American Indian or Alaska Native",..: 8 11 11 8 11 8 8
wwlist %>% count(ethn_code)
#> # A tibble: 11 x 2
#>    ethn_code                                        n
#>    <fct>                                        <int>
#>  1 American Indian or Alaska Native               202
```

3.4 Using case_when() function within mutate()

Using `case_when()` function within `mutate()`

Description Useful when the variable you want to create is more complicated than variables that can be created using `ifelse()` or `recode()`

- For example, useful when new variable is a function of multiple "input" variables

Usage (i.e., syntax): `case_when(...)`

Arguments [from help file; see help file for more details]

- `...`: A sequence of two-sided formulas.
  - The left hand side (LHS) determines which values match this case.
    - LHS must evaluate to a logical vector.
  - The right hand side (RHS) provides the replacement value.

Example task: Using data frame `wwlist` and input vars `state` and `firstgen`, create a 4-category var with following categories:

- "instate_firstgen"; "instate_nonfirstgen"; "outstate_firstgen"; "outstate_nonfirstgen"

```
wwlist_temp <- wwlist %>% select(state,firstgen) %>%
  mutate(state_gen = case_when(
    state == "WA" & firstgen =="Y" ~ "instate_firstgen",
    state == "WA" & firstgen =="N" ~ "instate_nonfirstgen",
    state != "WA" & firstgen =="Y" ~ "outstate_firstgen",
    state != "WA" & firstgen =="N" ~ "outstate_nonfirstgen")
  )
str(wwlist_temp$state_gen)
```

Using `case_when()` function within `mutate()`

Task: Using data frame `wwlist` and input vars `state` and `firstgen`, create a 4-category var with following categories:

- "instate_firstgen"; "instate_nonfirstgen"; "outstate_firstgen"; "outstate_nonfirstgen"

Let's take a closer look at how values of inputs are coded into values of outputs

```
wwlist %>% select(state,firstgen) %>% str()
count(wwlist,state)
count(wwlist,firstgen)

wwlist_temp <- wwlist %>% select(state,firstgen) %>%
  mutate(state_gen = case_when(
    state == "WA" & firstgen =="Y" ~ "instate_firstgen",
    state == "WA" & firstgen =="N" ~ "instate_nonfirstgen",
    state != "WA" & firstgen =="Y" ~ "outstate_firstgen",
    state != "WA" & firstgen =="N" ~ "outstate_nonfirstgen")
  )

wwlist_temp %>% count(state_gen)
wwlist_temp %>% filter(is.na(state)) %>% count(state_gen)
wwlist_temp %>% filter(is.na(firstgen)) %>% count(state_gen)
```

Student exercise using `case_when()` within `mutate()`

PATRICIA CREATE STUDENT EXERCISE

Mutate to create indicator variables

We often create dichotomous (0/1) indicator variables of whether something happened (or whether something is TRUE)

- Variables that are of substantive interest to project
  - e.g., did student graduate from college
- Variables that help you investigate data, check quality
  - e.g., indicator of whether an observation is missing/non-missing for a particular variable

Let's conduct some investigations of `df_school`, which has one observation for each high school

Creating indicators for `df_schoolv2` data frame

Create TRUE/FALSE indicator that median household income greater than $50,000

```
school_v2_temp <- school_v2 %>% mutate(incgt50k = med_inc>50000)

school_v2_temp %>% select(med_inc, incgt50k) %>% head(n=3)
#> # A tibble: 3 x 2
#>   med_inc incgt50k
#>     <dbl> <lgl>
#> 1   76160 TRUE
#> 2   76160 TRUE
#> 3      NA NA

school_v2_temp %>% filter(is.na(med_inc)) %>% count(incgt50k)
#> # A tibble: 1 x 2
#>   incgt50k     n
#>   <lgl>    <int>
#> 1 NA         624
```

Important takeaway:

- Variable created by `mutate()` equals `NA` for obs if input variable to `mutate()` is missing for that obs. This is a good thing!

3.5 Alternative approach to creating 0/1 indicators

Creating indicators for `school_v2` data frame

PATRICIA - I WROTE THESE BELOW SLIDES IN SUMMER BEFORE I KNEW ABOUT `ifelse()`; YOU MAKE A RECOMMENDATION TO ME ABOUT WHETHER THESE SLIDES SHOULD BE CUT Create TRUE/FALSE indicator that school is less than 50 percent white

```
school_v2_temp <- school_v2 %>% mutate(lt50pctwhite = pct_white<50)
school_v2_temp %>% select(pct_white,lt50pctwhite) %>% head(n=3)
#> # A tibble: 3 x 2
#>   pct_white lt50pctwhite
#>       <dbl> <lgl>
#> 1      11.8 TRUE
#> 2       0   TRUE
#> 3       0   TRUE
str(school_v2_temp$lt50pctwhite)
#>  logi [1:21301] TRUE TRUE TRUE TRUE TRUE TRUE ...
```

Create 0/1 integer indicator rather than logical indicator

```
df_schoolv2_temp <- school_v2 %>% mutate(lt50pctwhite = as.integer(pct_white<50))
df_schoolv2_temp %>% select(pct_white,lt50pctwhite) %>% head(n=3)
#> # A tibble: 3 x 2
#>   pct_white lt50pctwhite
#>       <dbl>        <int>
#> 1      11.8            1
#> 2       0              1
#> 3       0              1
str(df_schoolv2_temp$lt50pctwhite)
```

Student exercises

0/1 indicators of whether school received visit from each university

```
school_v2 %>% count(visits_by_berkeley)
#> # A tibble: 4 x 2
#>   visits_by_berkeley     n
#>                <int> <int>
#> 1                  0 20732
#> 2                  1   528
#> 3                  2    36
#> 4                  3     5
school_v2_temp <- school_v2 %>% mutate(yesvis_berkeley = visits_by_berkeley>0)

school_v2_temp %>% filter(visits_by_berkeley>0) %>% select(visits_by_berkeley,vi
#> # A tibble: 569 x 1
#>    visits_by_berkeley
#>                 <int>
#>  1                  2
#>  2                  2
#>  3                  2
#>  4                  1
#>  5                  1
#>  6                  1
#>  7                  1
#>  8                  1
#>  9                  1
#> 10                  1
#> # ... with 559 more rows
```