Social Sciences Intro to Statistics

Week 1.2 Basics of R

Week 1: Learning goal - Understand what and how to access R and R studio.

Introduction

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)`

tidyverse

The package we just downloaded, tidyverse, is a programming package in R that helps us transform data. This package is important for data mutation and visualization.

Investigating data patterns

Introduction to the `dplyr` library

`dplyr`, a package within the `tidyverse` suite of packages, provide tools for manipulating data frames

- Wickham describes functions within `dplyr` as a set of "verbs" that fall in the broader categories of **subsetting**, **sorting**, and **transforming**

-select() extracts columns and returns a tibble.

-arrange() changes the ordering of the rows.

-filter() picks cases based on their values.

-mutate() adds new variables that are functions of existing variables.

-rename() easily changes the name of a column(s).

-pull() extracts a single column as a vector.

Today	Upcoming weeks
Subsetting data	**Transforming data**
- `select()` variables	- `summarize()` calculates across rows
- `filter()` observations	- `group_by()` to calculate across rows within groups
- `mutate()` creates new variables	

- `pull()` variables **Sorting data** |
- `arrange()` | **Transforming data**
- `rename`() variables |

All `dplyr` verbs (i.e., functions) work as follows

1. first argument is a data frame
2. subsequent arguments describe what to do with variables and observations in data frame
    - refer to variable names without quotes
3. result of the function is a new data frame

Data for lecture sections on select(), arrange(), filter(), mutate(), rename(), and pull() functions

Lecture overview

- Introduction to Netflix data on IMDb score and votes
- Brief review of statistics (selected concepts)

Libraries we will use

```
#install.packages('tidyverse') # if you haven't installed already
#install.packages('labelled') # if you haven't installed already

library(tidyverse) # load tidyverse package
library(labelled) # load labelled package package
```

College Scorecard data

The College Scorecard is a tool created by the *US Department of Education* that seeks to help prospective students investigate/compare postsecondary institutions and degree programs

- We will use data from the *College Scorecard* in lecture and potentially for some assignments
- Relevant links
  - College Scorecard website
  - Data documentation
  - Data download

Recently, the *US Department of Education* released new *College Scorecard* data on debt and earnings associated with postsecondary degree programs

- (In theory) for each postsecondary degree program offered by a college/university, *College Scorecard* identifies average student debt associated with that degree program (for federal student loan programs) and average earnings of graduates
- Many debt and earnings variables have the value `'PrivacySuppressed'` because number of graduates is sufficiently small that there are concerns about being able to identify individual students
- Generally, academic programs with "large" enrollment are not suppressed
- We will focus on debt and earnings associated with master's (MA) programs

  - Hopefully, this information will be useful for students considering an MA program down the road

In the following sub-sections, we "load" the data, create modified datasets, investigate the data, and run some basic descriptive statistics

- **Your are not responsible for knowing the below code**
  - You will only be responsible for knowing code that we explicitly teach you during the quarter
- But try to follow the general logic of what the code is doing
- And try running the below "code chunks" on your own computer

Load and inspect data

Load data frame (i.e., dataset)

```
load(file = url('https://github.com/anyone-can-cook/educ152/raw/main/data/college_scorecar
## Need to update to new link with raw data on college scorecard

best_netflix <- read_csv("https://raw.githubusercontent.com/bcl96/Social-Sciences-Stats/ma
```

Load .Rdata data frames, `df_event` and `df_school`

Data on off-campus recruiting events by public universities

- Data frame object `df_event`
  - One observation per university, recruiting event
- Data frame object `df_school`
  - One observation per high school (visited and non-visited)

```
getwd()
#> [1] "/Users/bellelee/Downloads/SSS Lectures"
#load dataset with one obs per recruiting event
load(url("https://github.com/ozanj/rclass/raw/master/data/recruiting/recruit_event_somevar
#load("../../data/recruiting/recruit_event_somevars.Rdata")

#load dataset with one obs per high school
load(url("https://github.com/ozanj/rclass/raw/master/data/recruiting/recruit_school_someva
```

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

Data for lecture sections on pipes and mutate() function

Load .Rdata data frame `wwlist`, "prospects" purchased by Western Washington U.

Note: we won't use this data frame until the lecture section on "pipes"

- You can ignore `wwlist` data frame for lecture sections on select(), filter(), and arrange() functions

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(url("https://github.com/ozanj/rclass/raw/master/data/prospect_list/wwlist_merged.RDat
```

Object `wwlist`

- De-identified list of prospective students purchased by Western Washington University from College Board
- We collected these data using public records requests request

Data frame `wwlist`, "prospects" purchased by Western Washington U.

Observations on `wwlist`

- each observation represents a prospective student

```
#typeof(wwlist)
#dim(wwlist)

typeof(best_netflix)
#> [1] "list"
dim(best_netflix)
#> [1] 246   8
```

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)
glimpse(wwlist) # tidyverse function, similar to str()
```

Data frame `wwlist`, "prospects" purchased by Western Washington U.

Variable `firstgen` identifies whether prospect is a first-generation college student

Imagine we want to isolate all the first-generation prospects

1. Investigate variable type/structure.

- A dichotomous var, but stored as character in `wwlist`. So must use quotes (`''` or `""`) to filter/subset based on values of `firstgen`

```
str(wwlist$firstgen)
#>  chr [1:268396] NA "N" "N" "N" NA "N" "N" "Y" "Y" "N" "N" "N" "N" "N" "N" ...
```

2. Create frequency table to identify possible values of `firstgen`

```
table(wwlist$firstgen, useNA = "always")
#>
#>      N      Y   <NA>
#> 193333  65046  10017
```

3. Isolate all the first-gen prospects (output omitted)

```
filter(wwlist, firstgen == "Y")
```

select() variables

Select variables using `select()` function

Printing observations is key to investigating data, but datasets often have hundreds, thousands of variables

`select()` function selects **columns** of data (i.e., variables) you specify

- first argument is the name of data frame object
- remaining arguments are variable names, which are separated by commas and without quotes

Without **assignment** (`<-`), `select()` by itself simply prints selected vars

```
#?select
select(df_event,instnm,event_date,event_type,event_state,med_inc)
#> # A tibble: 18,680 x 5
#>    instnm      event_date event_type event_state med_inc
#>    <chr>       <date>     <chr>      <chr>         <dbl>
#>  1 UM Amherst  2017-10-12 public hs  MA           71714.
#>  2 UM Amherst  2017-10-04 public hs  MA           89122.
#>  3 UM Amherst  2017-10-25 public hs  MA           70136.
#>  4 UM Amherst  2017-10-26 public hs  MA           70136.
#>  5 Stony Brook 2017-10-02 public hs  MA           71024.
#>  6 USCC        2017-09-18 private hs MA           71024.
#>  7 UM Amherst  2017-09-18 private hs MA           71024.
#>  8 UM Amherst  2017-09-26 public hs  MA           97225
#>  9 UM Amherst  2017-09-26 private hs MA           97225
#> 10 UM Amherst  2017-10-12 public hs  MA           77800.
#> # i 18,670 more rows
```

Select variables using `select()` function

Recall that all `dplyr` functions (e.g., `select()`) return a new data frame object

- **type** equals "list"
- **length** equals number of vars you select

```
typeof(select(df_event,instnm,event_date,event_type,event_state,med_inc))
#> [1] "list"
length(select(df_event,instnm,event_date,event_type,event_state,med_inc))
```

```
#> [1] 5
```

`glimpse()`: tidyverse function for viewing data frames

- a cross between `str()` and simply printing data

```
?glimpse
glimpse(df_event)
```

`glimpse()` a `select()` set of variables

```
glimpse(select(df_event,instnm,event_date,event_type,event_state,med_inc))
#> Rows: 18,680
#> Columns: 5
#> $ instnm      <chr> "UM Amherst", "UM Amherst", "UM Amherst", "UM Amherst", "S~
#> $ event_date  <date> 2017-10-12, 2017-10-04, 2017-10-25, 2017-10-26, 2017-10-0~
#> $ event_type  <chr> "public hs", "public hs", "public hs", "public hs", "publi~
#> $ event_state <chr> "MA", "MA", "MA", "MA", "MA", "MA", "MA", "MA", "MA", "MA"~
#> $ med_inc     <dbl> 71713.5, 89121.5, 70136.5, 70136.5, 71023.5, 71023.5, 7102~
```

Select variables using `select()` function

With **assignment** (`<-`), `select()` creates a new object containing only the variables you specify

```
event_small <- select(df_event,instnm,event_date,event_type,event_state,
  med_inc)

glimpse(event_small)
#> Rows: 18,680
#> Columns: 5
#> $ instnm      <chr> "UM Amherst", "UM Amherst", "UM Amherst", "UM Amherst", "S~
#> $ event_date  <date> 2017-10-12, 2017-10-04, 2017-10-25, 2017-10-26, 2017-10-0~
#> $ event_type  <chr> "public hs", "public hs", "public hs", "public hs", "publi~
#> $ event_state <chr> "MA", "MA", "MA", "MA", "MA", "MA", "MA", "MA", "MA", "MA"~
#> $ med_inc     <dbl> 71713.5, 89121.5, 70136.5, 70136.5, 71023.5, 71023.5, 7102~
```

Select

`select()` can use "helper functions" `starts_with()`, `contains()`, and `ends_with()` to choose columns

```
?select
```

Example:

```
#names(df_event)

select(df_event,instnm,starts_with("event"))
#> # A tibble: 18,680 x 8
#>    instnm      event_date event_type event_state event_inst event_name
#>    <chr>       <date>     <chr>      <chr>       <chr>      <chr>
#>  1 UM Amherst  2017-10-12 public hs  MA          In-State   Amherst-Pelham Regi~
#>  2 UM Amherst  2017-10-04 public hs  MA          In-State   Hampshire County Co~
#>  3 UM Amherst  2017-10-25 public hs  MA          In-State   Chicopee High Schoo~
#>  4 UM Amherst  2017-10-26 public hs  MA          In-State   Chicopee Comprehens~
#>  5 Stony Brook 2017-10-02 public hs  MA          Out-State  Easthampton High Sc~
#>  6 USCC        2017-09-18 private hs MA          Out-State  Williston Northampt~
#>  7 UM Amherst  2017-09-18 private hs MA          In-State   Williston-Northampt~
#>  8 UM Amherst  2017-09-26 public hs  MA          In-State   Granby Jr Sr High S~
#>  9 UM Amherst  2017-09-26 private hs MA          In-State   MacDuffie School Vi~
#> 10 UM Amherst  2017-10-12 public hs  MA          In-State   Smith Academy Visit
#> # i 18,670 more rows
#> # i 2 more variables: event_location_name <chr>, event_datetime_start <dttm>
```

Rename variables

`rename()` function renames variables within a data frame object

Syntax:

- `rename(obj_name, new_name = old_name,...)`

```
rename(df_event, g12_offered = g12offered,
       titlei = titlei_status_pub)
names(df_event)
```

Variable names do not change permanently unless we combine rename with assignment

```
rename_event <- rename(df_event, g12_offered = g12offered, titlei = titlei_status_pub)
names(rename_event)
rm(rename_event)
```

filter() rows

The `filter()` function

`filter()` allows you to **select observations** based on values of variables

- Arguments
  - first argument is name of data frame
  - subsequent arguments are *logical expressions* to filter the data frame
  - Multiple expressions separated by commas work as **AND** operators (e.g., condtion 1 `TRUE` AND condition 2 `TRUE`)
- What is the result of a `filter()` command?
  - `filter()` returns a data frame consisting of rows where the condition is `TRUE`

```
?filter
```

Example from data frame object `df_school`, each obs is a high school

- Show all obs where the high school received 1 visit from UC Berkeley (110635) [output omitted]

```
filter(df_school,visits_by_110635 == 1)
```

Note that resulting object is list, consisting of obs where condition `TRUE`

```
nrow(df_school)
#> [1] 21301
nrow(filter(df_school,visits_by_110635 == 1))
#> [1] 528
```

The `filter()` function, base R equivalents

Task: Count the number of high schools that received 1 visit from UC Berkeley.

tidyverse Using `filter()`:

```
nrow(filter(df_school, visits_by_110635 == 1))
#> [1] 528
```

[**base R**] Using `[]` and `$`:

```
nrow(df_school[df_school$visits_by_110635 == 1, ])
#> [1] 528
```

[**base R**] Using `subset()`:

```
nrow(subset(df_school, visits_by_110635 == 1))
#> [1] 528
```

Filter, character variables

Use single quotes `''` or double quotes `""` to refer to values of character variables

```
glimpse(select(df_school, school_type, state_code))
#> Rows: 21,301
#> Columns: 2
#> $ school_type <chr> "public", "public", "public", "public", "public", "public"~
#> $ state_code  <chr> "AK", "AK", "AK", "AK", "AK", "AK", "AK", "AK", "AK", "AK"~
```

Identify all private high schools in CA that got 1 visit by particular universities

- Visited once by UC Berkeley (ID=110635)

```
filter(df_school,visits_by_110635 == 1, school_type == "private",
       state_code == "CA")
```

- Visited once by University of Alabama (ID=100751)

```
filter(df_school,visits_by_100751 == 1, school_type == "private",
       state_code == "CA")
```

- Visited once by Berkeley and University of Alabama

```
filter(df_school,visits_by_100751 == 1, visits_by_110635 == 1,
       school_type == "private", state_code == "CA")
```

Filter by multiple conditions, base R equivalents

Task: Count the number of private high schools in CA that received 1 visit each from UC Berkeley and University of Alabama.

tidyverse Using `filter()`:

```
nrow(filter(df_school, visits_by_100751 == 1, visits_by_110635 == 1,
            school_type == "private", state_code == "CA"))
#> [1] 9
```

[**base R**] Using `[]` and `$`:

```
nrow(df_school[df_school$visits_by_100751 == 1 &
                 df_school$visits_by_110635 == 1 &
                 df_school$school_type == "private" &
                 df_school$state_code == "CA", ])
#> [1] 9
```

[**base R**] Using `subset()`:

```
nrow(subset(df_school, visits_by_100751 == 1 & visits_by_110635 == 1 &
            school_type == "private" & state_code == "CA"))
#> [1] 9
```

Logical operators for comparisons

logical operators useful for: filter obs w/ `filter()`; create variables w/ `mutate()`

- logical operators also work when using Base R functions

Operator symbol	Operator meaning
`==`	Equal to
`!=`	Not equal to
`>`	greater than
`>=`	greater than or equal to
`<`	less than
`<=`	less than or equal to
`&`	AND
`|`	OR

Operator symbol	Operator meaning
`%in%`	includes

- Visualization of "Boolean" operators (e.g., AND, OR, AND NOT)

["Boolean" operations, x=left circle, y=right circle, from Wichkam (2018)]{width=40%}

Aside: `count()` function

`count()` function from `dplyr` package counts the number of obs by group

Syntax [see help file for full syntax]

- `count(x,...)`

Arguments [see help file for full arguments]

- `x`: an object, often a data frame
- `...`: variables to group by

Examples of using `count()`

- Without vars in `...` argument, counts number of obs in object

```
count(df_school)
  # df_school %>% count() # same as above but using pipes
str(count(df_school))
  # #df_school %>% count() %>% str() # same as above but using pipes
```

- With vars in `...` argument, counts number of obs per variable value
  - This is the best way to create frequency table, better than `table()`
  - note: by default, `count()` always shows `NAs` [this is good!]

```
count(df_school,school_type)
  # df_school %>% count(school_type) # same as above but using pipes
str(count(df_school,school_type))
  # df_school %>% count(school_type) %>% str() # same as above but using pipes
```

Filters and comparisons, Demonstration

Schools visited by Bama (100751) and/or Berkeley (110635)

```
# Berkeley AND Bama
filter(df_school,visits_by_100751 >= 1, visits_by_110635 >= 1)
filter(df_school,visits_by_100751 >= 1 & visits_by_110635 >= 1) # same same

df_school[df_school$visits_by_100751 >= 1 &
            df_school$visits_by_110635 >= 1, ] # using [] and $

subset(df_school,visits_by_100751 >= 1 &
         visits_by_110635 >= 1) # using subset()
```

```
# Berkeley OR Bama
filter(df_school,visits_by_100751 >= 1 | visits_by_110635 >= 1)

df_school[df_school$visits_by_100751 >= 1 |
            df_school$visits_by_110635 >= 1, ] # using [] and $

subset(df_school,visits_by_100751 >= 1 |
         visits_by_110635 >= 1) # using subset()
```

Filters and comparisons, Demonstration (cont.)

Apply `count()` function on top of `filter()` function to count the number of observations that satisfy criteria

- Avoids printing individual observations

```
# Number of schools that get visit by Berkeley AND Bama
count(filter(df_school, visits_by_100751 >= 1 & visits_by_110635 >= 1))
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1   247

# Number of schools that get visit by Berkeley OR Bama
count(filter(df_school, visits_by_100751 >= 1 | visits_by_110635 >= 1))
```

```
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1  2763
```

- Note: You could also use any of the base R equivalents from the previous slide

Filters and comparisons, >=

Number of public high schools that are at least 50% Black in Alabama compared to number of schools that received visit by Bama:

```
# at least 50% black
count(filter(df_school, school_type == "public", pct_black >= 50,
             state_code == "AL"))
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1    86

# at least 50% black and received visit by Bama
count(filter(df_school, school_type == "public", pct_black >= 50,
             state_code == "AL", visits_by_100751 >= 1))
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1    21
```

Filters and comparisons, >= (cont.)

Number of public high schools that are at least 50% White in Alabama compared to number of schools that received visit by Bama:

```
# at least 50% white
count(filter(df_school, school_type == "public", pct_white >= 50,
             state_code == "AL"))
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1   238
```

```
# at least 50% white and received visit by Bama
count(filter(df_school, school_type == "public", pct_white >= 50,
             state_code == "AL", visits_by_100751 >= 1))
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1    82
```

Filters and comparisons, not equals (!=)

Count the number of high schools visited by University of Colorado (126614) that are not located in CO

```
#number of high schools visited by U Colorado
count(filter(df_school, visits_by_126614 >= 1))
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1  1056

#number of high schools visited by U Colorado not located in CO
count(filter(df_school, visits_by_126614 >= 1, state_code != "CO"))
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1   873

#number of high schools visited by U Colorado located in CO
#count(filter(df_school, visits_by_126614 >= 1, state_code == "CO"))
```

Filters and comparisons, %in% operator

What if you wanted to count the number of schools visited by Bama (100751) in a group of states?

```
count(filter(df_school,visits_by_100751 >= 1, state_code == "MA" |
               state_code == "VT" | state_code == "ME"))
#> # A tibble: 1 x 1
#>       n
```

```
#>   <int>
#> 1   108
```

Easier way to do this is with `%in%` operator

```
count(filter(df_school,visits_by_100751 >= 1, state_code %in% c("MA","ME","VT")))
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1   108
```

Select the private high schools that got either 2 or 3 visits from Bama

```
count(filter(df_school, visits_by_100751 %in% 2:3, school_type == "private"))
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1   183
```

Identifying data type and possible values helpful for filtering

- `typeof()` and `str()` shows internal data type of a variable
- `table()` to show potential values of categorical variables

```
typeof(df_event$event_type)
#> [1] "character"
str(df_event$event_type)  # double quotes indicate character
#>  chr [1:18680] "public hs" "public hs" "public hs" "public hs" "public hs" ...
table(df_event$event_type, useNA="always")
#>
#> 2yr college 4yr college       other  private hs   public hs        <NA>
#>         951         531        2001        3774       11423           0

typeof(df_event$med_inc)
#> [1] "double"
str(df_event$med_inc)
#>  num [1:18680] 71714 89122 70136 70136 71024 ...
```

Now that we know `event_type` is a character, we can filter values

```
count(filter(df_event, event_type == "public hs", event_state =="CA"))
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1  1100
#below code would return an error because variables are character
#count(filter(df_event, event_type == public hs, event_state ==CA))
```

Filtering and missing values

Wickham (2018) states:

- "`filter()` only includes rows where condition is TRUE; it excludes both `FALSE` and `NA` values. To preserve missing values, ask for them explicitly:"

Investigate var `df_event$fr_lunch`, number of free/reduced lunch students

- only available for visits to public high schools

```
#visits to public HS with less than 50 students on free/reduced lunch
count(filter(df_event,event_type == "public hs", fr_lunch<50))
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1   910
#visits to public HS, where free/reduced lunch missing
count(filter(df_event,event_type == "public hs", is.na(fr_lunch)))
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1    26
#visits to public HS, where free/reduced is less than 50 OR is missing
count(filter(df_event,event_type == "public hs", fr_lunch<50 | is.na(fr_lunch)))
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1   936
```

Exercise

Task

- Create a filter to identify all the high schools that recieved 1 visit from UC Berkeley (110635) AND 1 visit from CU Boulder (126614)[output omitted]

Solution

```
filter(df_school,visits_by_110635 == 1, visits_by_126614==1)

nrow(filter(df_school,visits_by_110635 == 1, visits_by_126614==1))
count(filter(df_school,visits_by_110635 == 1, visits_by_126614==1))
```

- Must **assign** to create new object based on filter

```
berk_boulder <- filter(df_school,visits_by_110635 == 1, visits_by_126614==1)
count(berk_boulder)
```

Exercises

Use the data from df_event, which has one observation for each off-campus recruiting event a university attends

1. Count the number of events attended by the University of Pittsburgh (Pitt) `univ_id == 215293`

2. Count the number of recruiting events by Pitt at public or private high schools

3. Count the number of recruiting events by Pitt at public or private high schools located in the state of PA

4. Count the number of recruiting events by Pitt at public high schools not located in PA where median income is less than 100,000

5. Count the number of recruiting events by Pitt at public high schools not located in PA where median income is greater than or equal to 100,000

6. Count the number of out-of-state recruiting events by Pitt at private high schools or public high schools with median income of at least 100,000

Solution

1. Count the number of events attended by the University of Pittsburgh (Pitt) `univ_id == 215293`

```
count(filter(df_event, univ_id == 215293))
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1  1225
```

2. Count the number of recruiting events by Pitt at public or private high schools

```
str(df_event$event_type)
#>  chr [1:18680] "public hs" "public hs" "public hs" "public hs" "public hs" ...
table(df_event$event_type, useNA = "always")
#>
#> 2yr college 4yr college       other  private hs   public hs        <NA>
#>         951         531        2001        3774       11423           0
count(filter(df_event, univ_id == 215293, event_type == "private hs" |
               event_type == "public hs"))
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1  1030
```

Solution

3. Count the number of recruiting events by Pitt at public or private high schools located in the state of PA

```
count(filter(df_event, univ_id == 215293, event_type == "private hs" |
               event_type == "public hs", event_state == "PA"))
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1   262
```

4. Count the number of recruiting events by Pitt at public high schools not located in PA where median income is less than 100,000

```
count(filter(df_event, univ_id == 215293, event_type == "public hs",
             event_state != "PA", med_inc < 100000))
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1   213
```

Solution

5. Count the number of recruiting events by Pitt at public high schools not located in PA where median income is greater than or equal to 100,000

```
count(filter(df_event, univ_id == 215293, event_type == "public hs",
             event_state != "PA", med_inc >= 100000))
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1   344
```

6. Count the number of out-of-state recruiting events by Pitt at private high schools or public high schools with median income of at least 100,000

```
count(filter(df_event, univ_id == 215293, event_state != "PA",
             (event_type == "public hs" & med_inc >= 100000) |
               event_type == "private hs"))
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1   553
```

arrange() rows (i.e., sort rows)

`arrange()` function

`arrange()` function "arranges" rows in a data frame; said different, it sorts observations

Syntax: `arrange(x,...)`

- First argument, `x`, is a data frame
- Subsequent arguments are a "comma separated list of unquoted variable names"

```
df_event
arrange(df_event, event_date)
```

Data frame goes back to previous order unless you **assign** the new order

```
df_event
df_event <- arrange(df_event, event_date)
df_event
```

`arrange()` function

Ascending and descending order

- `arrange()` sorts in **ascending** order by default
- use `desc()` to sort a column by descending order

```
arrange(df_event, desc(event_date))
```

Can sort by multiple variables

```
arrange(df_event, univ_id, desc(event_date), desc(med_inc))

#sort by university and descending by size of 12th grade class; combine with select
select(arrange(df_event, univ_id, desc(g12)),instnm,event_type,event_date,g12)
```

`arrange()`, missing values sorted at the end

Missing values automatically sorted at the end, regardless of whether you sort ascending or descending

Below, we sort by university, then by date of event, then by ID of high school

```
#by university, date, ascending school id
select(arrange(df_event, univ_id, desc(event_date), school_id),
       instnm,event_date,event_type,school_id)

#by university, date, descending school id
select(arrange(df_event, univ_id, desc(event_date), desc(school_id)),
       instnm,event_date,event_type,school_id)
```

Can sort by `is.na` to put missing values first

```
select(arrange(df_event, univ_id, desc(event_date), desc(is.na(school_id))),
       instnm,event_date,event_type,school_id)
#> # A tibble: 18,680 x 4
#>    instnm event_date event_type school_id
#>    <chr>  <date>     <chr>      <chr>
#>  1 Bama   2017-12-18 other      <NA>
#>  2 Bama   2017-12-18 private hs A9106483
#>  3 Bama   2017-12-15 other      <NA>
#>  4 Bama   2017-12-15 public hs  484473005095
#>  5 Bama   2017-12-15 public hs  062927004516
#>  6 Bama   2017-12-14 other      <NA>
#>  7 Bama   2017-12-13 other      <NA>
#>  8 Bama   2017-12-13 public hs  130387001439
#>  9 Bama   2017-12-13 private hs 00071151
#> 10 Bama   2017-12-13 public hs  063386005296
#> # i 18,670 more rows
```

Exercise, arranging

Use the data from df_event, which has one observation for each off-campus recruiting event a university attends

1. Sort ascending by "univ_id" and descending by "event_date"

2. Select four variables in total and sort ascending by "univ_id" and descending by "event_date"

3. Now using the same variables from above, sort by `is.na` to put missing values in "school_id" first

Solution

1. Sort ascending by "univ_id" and descending by "event_date"

```
arrange(df_event, univ_id, desc(event_date))
#> # A tibble: 18,680 x 33
#>    instnm univ_id instst   pid event_date event_type zip   school_id    ipeds_id
#>    <chr>    <int> <chr>  <int> <date>     <chr>      <chr> <chr>           <int>
#>  1 Bama    100751 AL      7115 2017-12-18 private hs 77089 A9106483           NA
#>  2 Bama    100751 AL      7121 2017-12-18 other      <NA>  <NA>               NA
```

```
#>  3 Bama    100751 AL      7114 2017-12-15 public hs  75165 484473005095       NA
#>  4 Bama    100751 AL      7100 2017-12-15 public hs  93012 062927004516       NA
#>  5 Bama    100751 AL      7073 2017-12-15 other      98027 <NA>               NA
#>  6 Bama    100751 AL      7072 2017-12-14 other      98007 <NA>               NA
#>  7 Bama    100751 AL      7118 2017-12-13 public hs  31906 130387001439       NA
#>  8 Bama    100751 AL      7099 2017-12-13 private hs 90293 00071151           NA
#>  9 Bama    100751 AL      7109 2017-12-13 public hs  92630 063386005296       NA
#> 10 Bama    100751 AL      7071 2017-12-13 other      98032 <NA>               NA
#> # i 18,670 more rows
#> # i 24 more variables: event_state <chr>, event_inst <chr>, med_inc <dbl>,
#> #   pop_total <dbl>, pct_white_zip <dbl>, pct_black_zip <dbl>,
#> #   pct_asian_zip <dbl>, pct_hispanic_zip <dbl>, pct_amerindian_zip <dbl>,
#> #   pct_nativehawaii_zip <dbl>, pct_tworaces_zip <dbl>,
#> #   pct_otherrace_zip <dbl>, fr_lunch <dbl>, titlei_status_pub <fct>,
#> #   total_12 <dbl>, school_type_pri <int>, school_type_pub <int>, ...
```

Solution

2. Select four variables in total and sort ascending by "univ_id" and descending by "event_date"

```
select(arrange(df_event, univ_id, desc(event_date)), univ_id, event_date,
       instnm, event_type)
#> # A tibble: 18,680 x 4
#>    univ_id event_date instnm event_type
#>      <int> <date>     <chr>  <chr>
#>  1  100751 2017-12-18 Bama   private hs
#>  2  100751 2017-12-18 Bama   other
#>  3  100751 2017-12-15 Bama   public hs
#>  4  100751 2017-12-15 Bama   public hs
#>  5  100751 2017-12-15 Bama   other
#>  6  100751 2017-12-14 Bama   other
#>  7  100751 2017-12-13 Bama   public hs
#>  8  100751 2017-12-13 Bama   private hs
#>  9  100751 2017-12-13 Bama   public hs
#> 10  100751 2017-12-13 Bama   other
#> # i 18,670 more rows
```

Solution

3. Select the variables "univ_id", "event_date", and "school_id" and sort by `is.na` to put missing values in "school_id" first.

```
select(arrange(df_event, univ_id, desc(event_date), desc(is.na(school_id))),
       univ_id, event_date, school_id)
#> # A tibble: 18,680 x 3
#>    univ_id event_date school_id
#>      <int> <date>     <chr>
#>  1  100751 2017-12-18 <NA>
#>  2  100751 2017-12-18 A9106483
#>  3  100751 2017-12-15 <NA>
#>  4  100751 2017-12-15 484473005095
#>  5  100751 2017-12-15 062927004516
#>  6  100751 2017-12-14 <NA>
#>  7  100751 2017-12-13 <NA>
#>  8  100751 2017-12-13 130387001439
#>  9  100751 2017-12-13 00071151
#> 10  100751 2017-12-13 063386005296
#> # i 18,670 more rows
```

Introduce mutate() function

Introduce `mutate()` function

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

Description of `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)

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

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

# remove data frame object
#rm(school_sml)
```

Investigate `mutate()` syntax

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
  - "Name-value pairs of expressions"
  - "The name of each argument will be the name of a new variable, and the value will be its corresponding value."
  - "Use a `NULL` value in mutate to drop a variable."
  - "New variables overwrite existing variables of the same name"

Value

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

Investigate `mutate()` syntax

Can create variables using standard mathematical or logical operators [output omitted]

```
#glimpse(school_v2)
#school_v2 %>%
  #select(state_code,school_type,ncessch,med_inc,num_fr_lunch,total_students,num_took_math
  #mutate( # each argument creates a new variable, name of argument is name of variable
```

```
    #one = 1,
    #med_inc000 = med_inc/1000,
    #pct_fr_lunch = num_fr_lunch/total_students*100,
    #took_math_na = is.na(num_took_math)==1
  #) %>%
  #select(state_code,school_type,ncessch,one,med_inc,med_inc000,num_fr_lunch,total_student
```

Can create variables using "helper functions" called within `mutate()` [output omitted]

- These are standalone functions can be called *within* `mutate()`
  - e.g., `if_else()`, `recode()`, `case_when()`
- will walk through helper functions in more detail in subsequent sections of lecture

```
#school_v2 %>%
  #select(state_code,ncessch,name,school_type) %>%
  #mutate(public = if_else(school_type == "public", 1, 0))
```

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)
```

`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)
```

`mutate()` can use variables previously created within `mutate()`

```
#school_v2 %>%
  #select(num_prof_math, num_took_math, num_took_read,num_prof_read) %>%
  #mutate(pct_prof_math = num_prof_math/num_took_math,
         #pct_prof_read = num_prof_read/num_took_read,
         #avg_pct_prof_math_read = (pct_prof_math + pct_prof_read)/2)
```

`mutate()`, removing variables created by `mutate()`

Within `mutate()` use syntax `var_name = NULL` to remove variable from data frame

- note: Variable not permanently removed from data frame unless you use assignment `<-` to create new data frame or overwrite existing data frame

```
#ncol(school_v2)
#school_v2 %>%
  #select(num_prof_math, num_took_math, num_took_read,num_prof_read) %>% glimpse()

#school_v2 %>%
  #select(num_prof_math, num_took_math, num_took_read,num_prof_read) %>%
  #mutate(num_prof_math = NULL, num_took_math = NULL) %>% glimpse()
#But variables not permanently removed because we didn't use assignment
#ncol(school_v2)
```

Why would we remove variables within `mutate()` rather `select()`?

- remove temporary "work" variables used to create desired variable
- Example: measure of average of pct who passed math and pct who passed reading

```
#school_v2 %>%
  #select(num_prof_math, num_took_math, num_took_read,num_prof_read) %>%
  #mutate(pct_prof_math = num_prof_math/num_took_math, # create work var
         #pct_prof_read = num_prof_read/num_took_read, # create work var
         #avg_pct_prof_math_read = (pct_prof_math + pct_prof_read)/2, #create analysis var
         #pct_prof_math = NULL, # remove work var
         #pct_prof_read = NULL) %>% # remove work var
  #glimpse()
```

Student exercise using mutate()

1. Using the object `school_v2`, select the following variables (`num_prof_math`, `num_took_math`, `num_prof_read`, `num_took_read`) and create a measure of percent proficient in math `pct_prof_math` and percent proficient in reading `pct_prof_read`.
2. Now using the code for question 1, filter schools where at least 50% of students are proficient in math **&** reading.
3. Count the number of schools from question 2.
4. Using `school_v2`, using `mutate()` combined with `is.na()` create a dichotomous indicator variable `med_inc_na` that identifies whether `med_inc` is missing (`NA`) or not. And then use syntax `count(var_name)` to create frequency table of variable `med_inc_na`. How many observations are missing?

Solutions for exercise using mutate()

1. Using the object `school_v2`, select the following variables (`num_prof_math`, `num_took_math`, `num_prof_read`, `num_took_read`) and create a measure of percent proficient in math `pct_prof_math` and percent proficient in reading `pct_prof_read`.

```
#school_v2 %>%
  #select(num_prof_math, num_took_math, num_prof_read, num_took_read) %>%
  #mutate(pct_prof_math = num_prof_math/num_took_math,
         #pct_prof_read = num_prof_read/num_took_read)
```

Solutions for exercise using mutate()

2. Now using the code for question 1, filter schools where at least 50% of students are proficient in math & reading.

```
#school_v2 %>%
  #select(num_prof_math, num_took_math, num_prof_read, num_took_read) %>%
  #mutate(pct_prof_math = num_prof_math/num_took_math,
         #pct_prof_read = num_prof_read/num_took_read) %>%
  #filter(pct_prof_math >= 0.5 & pct_prof_read >= 0.5)
```

Solutions for exercise using mutate()

3. Count the number of schools from question 2.

```
#school_v2 %>%
  #select(num_prof_math, num_took_math, num_prof_read, num_took_read) %>%
  #mutate(pct_prof_math = num_prof_math/num_took_math,
         #pct_prof_read = num_prof_read/num_took_read) %>%
  #filter(pct_prof_math >= 0.5 & pct_prof_read >= 0.5) %>%
  #count()
```

Solutions for exercise using mutate()

4. Using `school_v2`, using `mutate()` combined with `is.na()` create a dichotomous indicator variable `med_inc_na` that identifies whether `med_inc` is missing (`NA`) or not. And then use syntax `count(var_name)` to create frequency table of variable `med_inc_na`. How many observations are missing?

```
#school_v2 %>%
  #mutate(med_inc_na = is.na(med_inc)) %>%
  #count(med_inc_na)
```

Using if_else() function within mutate()

Using `if_else()` function within `mutate()`

Description

- if `<condition>` `TRUE`, assign value; if `<condition>` `FALSE` assign 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`
- `missing`: value to assign to rows that have value `NA` for condition
  - default is `missing = NULL`; means that if condition is `NA`, then new_var == `NA`
  - But can assign different values to `NA`s, e.g., `missing = -9`

Value

- "Where condition is TRUE, the matching value from true, where it's FALSE, the matching value from false, otherwise NA."
- Unless otherwise specified, `NA`s in "input" var(s) assigned `NA` in "output var"

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

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

`if_else()` within `mutate()` to create 0/1 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

Using `if_else()` within `mutate()`

Task

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

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

```
#str(school_v2$med_inc) # investigate variable type
#school_v2 %>% count(med_inc) # frequency count, but this isn't very helpful

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

Create variable

```
#school_v2 %>% select(med_inc) %>%
  #mutate(inc_gt_100k= if_else(med_inc>100000,1,0)) %>%
  #count(inc_gt_100k) # note how NA values of med_inc treated
```

Using `if_else()` within `mutate()`

Task:

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

This time, let's experiment with the `missing` argument of `if_else()`

```
#what we wrote before
#school_v2 %>% select(med_inc) %>%
  #mutate(inc_gt_100k= if_else(med_inc>100000,1,0)) %>%
  #count(inc_gt_100k)

#manually write out the default value for `missing`
#school_v2 %>% select(med_inc) %>%
  #mutate(inc_gt_100k= if_else(med_inc>100000,1,0, missing = NULL)) %>%
  #count(inc_gt_100k) # note how NA values of med_inc treated

#school_v2 %>% select(med_inc) %>%
  #mutate(inc_gt_100k= if_else(med_inc>100000,1,0, missing = NA_real_)) %>%
  #count(inc_gt_100k) # note how NA values of med_inc treated
# NA can be coerced to any other vector type except raw:
```

```
# NA_integer_, NA_real_, NA_complex_ and NA_character_

# Here we give missing values in condition the value of -9 in new variable
#school_v2 %>% select(med_inc) %>%
  #mutate(inc_gt_100k= if_else(med_inc>100000,1,0, missing = -9)) %>%
  #count(inc_gt_100k)
```

Using `if_else()` 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 number of obs
```

Create variable

```
#school_v2 %>% select(num_took_math) %>%
  #mutate(nonmiss_math= if_else(!is.na(num_took_math),1,0)) %>%
  #count(nonmiss_math) # note how NA values treated
```

Student exercises `if_else()`

1. Using the object `school_v2`, 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"`).

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

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

Exercise`if_else()` solutions

1. Using the object `school_v2`, 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"`).

```
#str(school_v2$state_code) # investigate input variable
#school_v2 %>% filter(is.na(state_code)) %>% count() # investigate input var

#Create var
#school_v2 %>% mutate(in_state_berkeley=if_else(state_code=="CA",1,0)) %>%
  #count(in_state_berkeley)
```

Exercise`if_else()` solutions

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

```
#investigate input vars
#school_v2 %>% select(visits_by_berkeley, visits_by_irvine) %>% str()
#school_v2 %>% filter(is.na(visits_by_berkeley)) %>% count()
#school_v2 %>% filter(is.na(visits_by_irvine)) %>% count()

#create variable
#school_v2 %>%
  #mutate(berkeley_and_irvine=if_else(visits_by_berkeley>0
   # & visits_by_irvine>0,1,0)) %>%
  #count(berkeley_and_irvine)
```

Exercise`if_else()` solutions

3. 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=if_else(visits_by_berkeley>0 | visits_by_irvine>0,1,0)) %>%
  #count(berkeley_or_irvine)
```

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 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) # note: numeric variable
#wwlist_temp %>% count(public_school) # note the NAs
#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) # character variable
#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) # character variable created
#wwlist_temp %>% count(school_catv2)
#rm(wwlist_temp)
```

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

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()`

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()`

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) # note: numeric variable now
#wwlist_temp %>% count(school_catv2)
#wwlist %>% count(school_category)
```

```
#rm(wwlist_temp)
```

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

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

#names(df_event)
```

1. Using object `df_event`, assign new object `df_event_temp` and a numeric variable create `event_typev2` based on `event_type` with these categories:
    - `1` for "2yr college"; `2` for "4yr college"; `3` for "other"; `4` for "private hs"; `5` for "public hs"
2. This time use the `.default` argument to assign the value `5` for "public hs"

Exercise using `recode()` within `mutate()` solutions

Check input variable

```
#names(df_event)
#str(df_event$event_type)
#df_event %>% count(event_type)
```

Exercise using `recode()` within `mutate()` solutions

1. Using object `df_event`, assign new object `df_event_temp` and create a numeric variable `event_typev2` based on `event_type` with these categories:
    - `1` for "2yr college"; `2` for "4yr college"; `3` for "other"; `4` for "private hs"; `5` for "public hs"

```
#df_event_temp <- df_event %>%
 # select(event_type) %>%
#  mutate(event_typev2 = recode(event_type,
                            #   "2yr college" = 1,
                             #  "4yr college" = 2,
                             #  "other" = 3,
                             #  "private hs" = 4,
                             #  "public hs" = 5)
```

```
       #  )
#str(df_event_temp$event_typev2)
#df_event_temp %>% count(event_typev2)
#df_event %>% count(event_type)
```

Exercise using `recode()` within `mutate()` solutions

2. This time assign the value use the `.default` argument to assign the value `5` for "public hs"

```
#df_event_temp <- df_event %>% select(event_type) %>%
#  mutate(event_typev2 = recode(event_type,
  #  "2yr college" = 1,
  #  "4yr college" = 2,
  #  "other" = 3,
  #  "private hs" = 4,
   # .default = 5)
 # )
#str(df_event_temp$event_typev2)
#df_event_temp %>% count(event_typev2)
#df_event %>% count(event_type)
```

Using case_when() function within mutate()

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

`case_when()` useful for creating variable that 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)
#wwlist_temp %>% count(state_gen)
```

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

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

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)
```

Create variable

```
#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")
#  )
```

Compare values of input vars to value of output var

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

Take-away: by default var created by `case_when()` equals `NA` for obs where one of the inputs equals `NA`

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

1. Using the object `school_v2` and input vars `school_type`, and `state_code` , create a 4-category var `state_type` with following categories:
    - "instate_public"; "instate_private"; "outstate_public"; "outstate_private"
    - Note: We are referring to CA as in-state for this example

Exercise using `case_when()` within `mutate()` solution

Investigate

```
#school_v2 %>% select(state_code,school_type) %>% str()
#count(school_v2,state_code)
#school_v2 %>% filter(is.na(state_code)) %>% count()

#count(school_v2,school_type)
#school_v2 %>% filter(is.na(school_type)) %>% count()
```

Exercise using `case_when()` within `mutate()` solution

1. Using the object `school_v2` and input vars `school_type`, and `state_code` , create a 4-category var `state_type` with following categories:
    - "instate_public"; "instate_private"; "outstate_public"; "outstate_private"

```
#school_v2_temp <- school_v2 %>% select(state_code,school_type) %>%
 # mutate(state_type = case_when(
  #  state_code == "CA" & school_type == "public"  ~ "instate_public",
 #   state_code == "CA" & school_type == "private" ~ "instate_private",
 #   state_code != "CA" & school_type == "public" ~ "outstate_public",
 #   state_code != "CA" & school_type == "private" ~ "outstate_private")
 # )

#school_v2_temp %>% count(state_type)
#school_v2_temp %>% filter(is.na(state_code)) %>% count(state_type) #no missing
#school_v2_temp %>% filter(is.na(school_type)) %>% count(state_type) #no missing
```