Social Sciences Intro to Statistics

Pset 1: Due MONTH, DATE, YEAR at 11:59pm

Belle Lee

2024-06-15

Overview

Welcome to Social Sciences Introduction to Statistics using R. This problem set is intended to give you some practice becoming familiar with using R. In this problem set, we are asking you to: create an R project, render your file, load and investigate an R data frame that is stored on the web, and apply some basic functions to atomic vectors.

- Note: Change the values of the YAML header above to your name and the date.

Question 1: Creating an R project

Create an R project

- Create a folder where you want to save files associated with problem set 1. Let's call that folder "problemset1", but you can name it whatever you want.
  - For instance, it could be SSS » problem_sets » problemset1.
- In RStudio, click on "File" » "New Project" » "Existing Directory" » "Browse".
- Browse to find and select your problem set 1 folder.
- Click on "Create Project".
  - An R project file has the extension ".Rproj".
  - The name of the file should be "problemset1.Rproj", or whatever you named the folder.

Save this problemset1.Rmd file anywhere in the folder named problemset1.

- Use this naming convention "lastname_firstname_ps#" for your .qmd files (e.g. lee_belle_ps1.qmd).

    - If you want, you can change the name of this file to include your first and last name.

- Run the `getwd()` function and the `list.files()` function in the code chunk below.
- What is the output? Why?

```r
getwd()
list.files()
```

ANSWER:

ANSWER KEY: The output shows "/Users/bellelee/Documents/SSS, Fall 2024/problem_sets/problemset1" since that is the working directory I'm currently in. The getwd() code asks r studio to get or show the working directory. The output for list.files() shows "lee_belle_ps1.qmd" "problemset1.Rproj" since those are the two files in working directory.

Question 2: Render to pdf

- At the top of this .qmd file, type in your first and last name in the appropriate place in the YAML header (e.g. "Belle Lee").
- in the date field of the YAML header, insert the date within quotations (any date format is fine).
- Now click the "Render" button near the top of your RStudio window (icon with blue arrow sign) or drop down "File" and select "Render Document".
    - Alternatively you can use the shortcut: **Cmd/Ctrl + Shift + k**.
    - *Note*: One goal of this assignment is to make sure you are able to knit to a PDF without running into errors.

"Load" the package we will use today (output omitted)

- **you must run this code chunk**

```r
library(tidyverse)
```

If package not yet installed, then must install before you load. Install in "console" rather than .qmd 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.

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 w
```

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)
```

Exerciseif_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_somevars

#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
```