Lecture 9 problem set

INSERT YOUR NAME HERE

December 7, 2018

Contents

Required reading and instructions

Required reading before next class

General Problem Set instructions

In this homework, you will specify `pdf_document` as the output format. You must have LaTeX installed in order to create pdf documents.

If you have not yet installed MiKTeX/MacTeX, I recommend installing TinyTeX, which is much simpler to install!

- Instructions for installation of TinyTeX can be found Here
- General Instructions for Problem Sets Here

Overview of problem set

This problem set will ask you to write several functions to perform specific tasks. The problem set has several "Parts." Detailed instructions are given at the beginning of each "part." Generally speaking, this problem set provides more hints and "walks you through" the task more than the last few problem sets. This is because writing functions is difficult (a lot of little things that can go wrong and cause the function not to work) and because it is the end of the quarter so we assume that you have lots of assignments in other courses.

For all questons that ask you to write a function you must do the following:

Required steps for all questions that ask you to write a function

1. Perform the task/operation outside of a function
    - Perform task for at least two different sets of "inputs"

2. Write the function
3. Call/test the function
    - Call function for at least two different sets of input values

Note: this is usually not a linear process; more often, it is a "two steps forward, one-step backwards" type process. For example:

- You may successfully perform the task outside of a function, but then find out that approach does not work inside a function. So you go back to step (1) and develop a different approach to performing the task outside a function, and then attempt to implement this approach within a function.
- You may write a function, and successfully call it for a set of input values, but then find the function doesn't work for another set of input values. When this happens you have to go back to revise **step 2** (write the function) and/or go back to **step 1** (perform task outside of function)

Load libraries

```
#install.packages("tidyverse") #uncomment if you haven't installed these packaged
#install.packages("haven")
#install.packages("labelled")
library(tidyverse)
library(haven)
library(labelled)
```

Part 1: identify and "fix" missing values of variables

The questions for **Part 1** are about writing functions for survey data that do these sorts of things:

- Count the number of observations with missing values (in survey data missing values often represented by negative numbers or very high numbers)
- Replace these (survey) missing values with `NA`

Note: These functions won't work for "character" variables; don't worry about character variables when writing these functions

Question 1. Load High School Longitudinal Study of 2009 (HSLS) data frame

Run the code chunk below to load HSLS data, changes variable names from upper-case to lower-case, keeps selected variables, and does some cursory investigation of the data frame

```
rm(list = ls()) # remove all objects
getwd()
#list.files() # list files in directory w/ NLS data

#Read Stata data into R using read_data() function from haven package
hsls_stu <- read_dta(file="https://github.com/ozanj/rclass/raw/master/data/hsls/hsls_stu_small.dta", en

#change variable names to lowercase
names(hsls_stu) <- tolower(names(hsls_stu))

#names(hsls_stu)
#hsls_stu %>% select(contains("s3")) %>% var_label() # variable labels
```

```r
#keep selected variables
hsls_stu <- hsls_stu %>% select(stu_id,contains("univ"),
  x2sex,x2race,x2paredu,s3classes,s3focus,s3clglvl,x4evratndclg,x4ps1sector)

#investigate data frame
str(hsls_stu)
hsls_stu %>% var_label() # variable labels
hsls_stu %>% val_labels() # value labels
```

Question 2: Write a function called `num_negative` that prints the number of observations that have negative values for a specific variable

Perform task outside of function for a few variables

Write function and call function

Question 3. Write function `num_missing` that counts number of missing observations for a variable and allows you to specify which values are associated with missing for that variable.

Additional details:

- This function will take two arguments:
  1. `x` the variable (e.g., `hsls_stu$s3clglvl`)
  2. `miss_vals`, the vector of values you want to associate with "missing" variable.
- Use the val_labels() function to identify the values associated with missing.
  - HINT: `data_frame_name %>% select(varn_name) %>% val_labels()`
- HINT for counting obs w/ particular values (Base R approach:
  - `sum(data_frame_name$var_name %in% c(list of values separated by commas))`

Perform task outside of function for a few variables

Write function and call function

Question 4. Write function `fix_missing` that replaces specific values of a variable (values associated with missing) with `NA`

This function will take two arguments:

1. `x` the variable you want to modify (e.g., `hsls_stu$s3clglvl_v2`)
2. `miss_vals`, the vector of values you want to associate with "missing" variable.

Additional details and hints:

- Prior to performing task outside of the function for a specific variable, create a second version ("`v2`") that is an exact copy of the original variable; and then perform task/call function on this "`v2`" variable rather than the original variable
  - HINT for creating "`v2`" variables:
    - `hsls_stu <- hsls_stu %>% (mutate(s3classes_v2 = s3classes))`
  - Why create these "`v2`" variables?
    - Generally, it is bad practice to change the value of "input" variables
    - Creating these "`v2`" variables will allow you to experiment with completing this task without reading in dataset again

  - **Note**: after you perform task outside the function, you may need to recreate these "`v2`" variables again before calling the function
- Use the val_labels() function to identify the values associated with missing.
  - HINT: `data_frame_name %>% select(var_name) %>% val_labels()`
- Recommend using "Base R" approach rather than "Tidyverse/dplyr" approach when writing code to (A) perform task outside the function and (B) to perform task within function
  - Why? Using dplyr functions within user-written-functions requires additional programming concepts we have not covered
  - HINT for "Base R" approach to replacing values
    - `data_frame_name$var_name_v2 <- ifelse(data_frame_name$var_name_v2 %in% c(list of values separated by commas),NA,data_frame_name$var_name_v2)`
- Hints for calling the function:
  - Use this approach:
    - `data_frame_name$var_name_v2 <- fix_missing(data_frame_name$var_name_v2, c(list of values separated by commas))`
  - Don't use this approach [will not replace values of the variables]
    - `fix_missing(data_frame_name$var_name_v2, c(list of values separated by commas))`
- After you call the function, check that it worked correctly by performing the following descriptive checks:
  - frequency count (i.e., tabulation) of the original variable
  - frequency count (i.e., tabulation) of the "`v2`" variable (whose values you just changed)
  - two-way tabulation of the original variable by the "`v2`" variable
    - HINT: `data_frame_name %>% group_by(var_name) %>% count(var_name_v2)`

Create `v2` versions of three variables

Perform task outside of function for at least two variables

Write and call function

Part 2: Create variables for percent of people in each race/ethnicity group

In Part 2, you will load a data frame called `zip_data` that contains characteristics of each zip code and then you will write a function `pct_race()` that creates variables that measure the percent of people in each zip code who identify with a particular race/ethnicity group (e.g., create `pop_white_pct` from the input variables `pop_white` and `pop_total`)

- The function `pct_race()` should take three arguments:
  1. `pct_var`: the name of the variable you will create (e.g., `zip_data$pop_white_pct`)
     - this variable does not yet exist in the data frame `zip_data`
  2. `pop_var`: name of the variable that is the numerator for the percent race variable (e.g., `zip_data$pop_white`)
     - this variable already exists in the data frame `zip_data`
  3. `total_var`: name of the variable that is the denominator for the percent race variable (e.g., `zip_data$pop_total`)
     - note: the variable `pop_toal` already exists in the data frame `zip_data` and will be the denominator for all percent race variables
- Before creating function, create the percent race variable outside of a function for at least two race/ethnicity groups
  - after you create a percent race variable, run some simple descriptive statistics to make sure
- Recommend using "Base R" approach to creating percent race variables (both when creating the variable outside of a function and creating variables within a function)

  - HINT: `data_frame_name$var_name_pct <- (data_frame_name$var_name/zip_data$pop_total)*100`

  - If the new variable looks ok; for this problem set, just printing a few observations for the input variables and the output variable is sufficient
  - If you want to remove a variable you previously created:
    * SYNTAX: `data_fram_name$var_name <- NULL`
- Hints for calling the function:
  - Use this approach:
    * `data_frame_name$var_name_pct <- pct_race(data_frame_name$var_name_pct,data_frame_name$var_n`
  - Don't use this approach [will not replace values of the variables]
    * `pct_race(data_frame_name$var_name_pct,data_frame_name$var_name,data_frame_name$pop_total)`
- After writing the function, call the function for at least three race/ethnicity groups
  - after you create a percent race variable, run some simple descriptive statistics to make sure the new variable looks ok; for this problem set, just printing a few observations for the input variables and the output variable is sufficient

Run below code to load zip-code level data from the Census American Community Survey (ACS) and keep selected variables

```
#options(tibble.print_min=90)
#options(tibble.print_min=10)

zip_data <- read.csv('https://github.com/ozanj/rclass/raw/master/data/acs/zip_to_state.csv', na.strings=
  filter(!(state_code %in% c("PR"))) %>%
  select(state_code,zip_code,pop_total, pop_white, pop_black, pop_amerindian, pop_asian, pop_nativehawa
  rename(pop_nativeamer = pop_amerindian, pop_latinx = pop_hispanic)

names(zip_data)
#>  [1] "state_code"       "zip_code"         "pop_total"
#>  [4] "pop_white"        "pop_black"        "pop_nativeamer"
#>  [7] "pop_asian"        "pop_nativehawaii" "pop_otherrace"
#> [10] "pop_tworaces"     "pop_latinx"

#Check if race/ethnicity variables add up to race total "pop_total"
zip_data %>%
  mutate(pop_totalv2 = pop_white + pop_black + pop_nativeamer + pop_asian + pop_nativehawaii + pop_othe
  filter(pop_totalv2 == pop_total) %>%
  count()
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1 32989
```

Question 1: Create percent race variables outside of function

Question 2: Create percent race variables within function

Part 3: Think of a task you might want to write a function for

Questions

- What is some task from your coursework, research assistantship, job, or your own research that you might write a function to complete?

- Why would you want to accomplish this task with a function rather than copy and paste?
- How would you go about developing the function?

Don't spend more than 20 minutes answering Part 3. Your answer can be brief and you don't need to go into any technical details. I will ask students to share their ideas at the beginning of next class.

Part 4: Write a function to read-in multiple years of data

In my research, I often read in ten to twenty years of annual data. I would rather write a function that I call 20 times then use writing separate code for each of the 20 years of data (i.e., the "copy-and-paste" method).

In Part 4, you will create a function to read-in annual IPEDS data on admissions characteristics. We have provided ALL the code here. YOU WILL ONLY NEED TO DOWNLOAD AND SAVE THE DATA IN THE CORRECT FOLDER, AND CHANGE THE FILE PATH BUT NOTHING ELSE.

Goal here is to walk you through the process I went through over the summer to figure out how to write this function. As you will see below, there was a lot of trial and error.

The copy and paste method

Copy this link and paste it in your web browser.

- You should have a zip file called "ic_admit.zip"
- Unzip the file and save each stata data file in the ic folder (e.g. data/ipeds/ic)
  - Do not save the ic_admit folder, only the three data files

Here is an example of writing separate code to read in each year of data (as opposed to writing a function)

- NOTE: Change your file path to where the data is saved on your desktop
  - We recommend using the relative file path instead of the absolute file path.

```
#read in data
getwd()
#> [1] "/Users/patriciamartin/Dropbox/r_class_problem_sets/lecture9"
admit_16_17 <- read_dta(file="~/Desktop/rclass/data/ipeds/ic/ic16_17_admit.dta") %>%
  select(unitid,endyear,sector,contains("admcon"),
         contains("numapply"),contains("numadmit"))

admit_15_16 <- read_dta(file="~/Desktop/rclass/data/ipeds/ic/ic15_16_admit.dta") %>%
  select(unitid,endyear,sector,contains("admcon"),
         contains("numapply"),contains("numadmit"))

admit_14_15 <- read_dta(file="~/Desktop/rclass/data/ipeds/ic/ic14_15_admit.dta") %>%
  select(unitid,endyear,sector,contains("admcon"),
         contains("numapply"),contains("numadmit"))
```

Lots of repeated code here. Better to write a function

Process I went through to write a function that could read in annual IPEDS admissions data

Task:

- Write a function to read in annual IPEDS admissions data

We already performed the task outside of a function, so let's try writing a function

- NOTE: Change your file path to where the data is saved on your desktop
  - We recommend using the relative file path instead of the absolute file path.

```
read_admit <- function(ayear) {
  admit_ayear <- read_dta(file="~/Desktop/rclass/data/ipeds/ic/ayear_admit.dta") %>% #Change to your fi
  select(unitid,endyear,sector,contains("admcon"),
         contains("numapply"),contains("numadmit"))
}
read_admit(ayear=16_17)
#rm(list = ls())
```

What went wrong?

1. In assignment statement `admit_ayear <-`, we wanted text "16_17" to be substituted for text "ayear"
2. In file-path, we wanted the text "16_17" to be substituted for text "ayear"

Before attempting to revise the function, let's develop an approach for completing this task outside a function that is more conducive to something that we would place in the body of a function

Perform task outside of a function

- First, create an object for desired name for data frame (admit_16_17)

```
x <- "16_17" #create an object for academic year
x #print the object
#> [1] "16_17"

#use cat() to create object w/ desired name for data frame
cat("admit_",x)  #we need to remove spaces
#> admit_ 16_17
#?cat
cat("admit_",x, sep="") #remove spaces with the sep argument
#> admit_16_17

dfname <- cat("admit_",x, sep="")
#> admit_16_17
dfname  #problem is that cat() function can't be used for variable assignment because it just prints st
#> NULL

#use paste() to create object w/ desired name for data frame
paste("admit_",x, sep="")
#> [1] "admit_16_17"
dfname <- paste("admit_",x, sep="")
dfname #print object
#> [1] "admit_16_17"
```

Perform task outside of a function

- Next, let's try to a different way to specify file-path and name of the dataset as objects
- NOTE: Change your file path to where the data is saved on your desktop
  - We recommend using the relative file path instead of the absolute file path.

```
#name of the dataset we are reading, eg ic16_17_admit.dta
paste("ic",x,"_admit.dta",sep="")
#> [1] "ic16_17_admit.dta"
dtaname=paste("ic",x,"_admit.dta",sep="")
```

```
dtaname
#> [1] "ic16_17_admit.dta"

#Create object for filepath, eg. "~/Desktop/rclass/data/ipeds/ic/ic15_16_admit.dta"
  #?file.path # this function creates a file path
dir <- file.path("~","Desktop","rclass","data","ipeds","ic") #Change to your file path
dir #print object
#> [1] "~/Desktop/rclass/data/ipeds/ic"
setwd(dir) #Run line 476 and 477 together or run the entire code chunk
list.files()
#> [1] "ic14_15_admit.dta" "ic15_16_admit.dta" "ic16_17_admit.dta"
```

Perform task outside of a function.

- Now, we can put it all together
- NOTE: Change your file path to where the data is saved on your desktop
  - We recommend using the relative file path instead of the absolute file path.

```
#rm(list = ls())
x <- "16_17" # academic year
dfname <- paste("admit_",x, sep="") # name we want to assign to data frame
dtaname=paste("ic",x,"_admit.dta",sep="") # name of Stata dataset we are reading in
dir <- file.path("~","Desktop","rclass","data","ipeds","ic") # name of filepath to Stata datasets; Chan

#read data
getwd()
#> [1] "/Users/patriciamartin/Dropbox/r_class_problem_sets/lecture9"
setwd(dir) #Run line 492 and 493 together or run the entire code chunk
df <- read_dta(file=dtaname) %>% # read in data and assign default name df
  select(unitid,endyear,sector,contains("admcon"),
         contains("numapply"),contains("numadmit"))

#assign() function assigns a value to an object name; took me some googling to find this function and f
assign(dfname,df) # create new object that has values of df; object name is value of object dfname
rm(df)
```

Task: write a function to read in annual IPEDS admissions data

Now we can create function for task

- NOTE: Change your file path to where the data is saved on your desktop
  - We recommend using the relative file path instead of the absolute file path.

```
#rm(list = ls())
read_admit <- function(ayear) {

  dfname <- paste("admit_",ayear, sep="") # name we want to assign to data frame
  dtaname=paste("ic",ayear,"_admit.dta",sep="") # name of Stata dataset we are reading in
  dir <- file.path("~","Desktop","rclass","data","ipeds","ic") # name of filepath to Stata datasets; Ch

  #read data
  getwd()
  setwd(dir) #set working directory
  df <- read_dta(file=dtaname) %>% # read in data and assign default name df
    select(unitid,endyear,sector,contains("admcon"), #select a few variables
```

```
           contains("numapply"),contains("numadmit"))

  #default is that the return value is the last statement executed so we don't need to specify return s
  return(df)
}

admit_16_17 <- read_admit(ayear="16_17")
admit_15_16 <- read_admit(ayear="15_16")
admit_14_15 <- read_admit(ayear="14_15")
```

Once finished, knit to (pdf) and upload both .Rmd and pdf files to class website under the week 9 tab
Remeber to use this naming convention "lastname_firstname_ps9"