Module 7: Working with Strings and Date/Time Variables

Managing and Manipulating Data Using R

Introduction

What we will do today

1. Introduction
2. Working with Strings
   2.1 String basics
3. Working with Dates and Times
   3.1 Creating Date/Times
   3.2 Using Date/Time Variables
4. Mini Lesson on Exploratory Data Analysis (EDA)
   4.1 Tools for EDA
   4.2 Guidelines for EDA
   4.3 Skip patterns in survey data
5. Problem Set 7

Load the packages we will use today (output omitted)

- **you must run this code chunk after installing these packages**

```r
library(tidyverse)
library(stringr)
library(lubridate)
library(nycflights13)
library(haven)
library(labelled)
```

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("stringr")`

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)`

Load data we will use today

- Western Washington University student list data

```
load(url("https://github.com/ksalazar3/HED696C_Rclass/raw/master/data/prospect_
```

Working with Strings

String basics

What are strings?

String refers to a "data type" used in programming to represent text rather than numbers (although it can include numbers)

- Strings have `character` types

```
string1<- "Apple"
typeof(string1) #type is charater
#> [1] "character"
```

- Create strings using `" "`

```
string2 <- "This is a string"
```

- If string contains a quotation, use `' " " '`

```
string3 <- 'example of a "quote" within a string'
```

- To print a string, use `writeLines()`

```
print(string3) #will print using \
#> [1] "example of a \"quote\" within a string"
writeLines(string3)
#> example of a "quote" within a string
```

Common uses of strings

Basic uses:

- Names of files and directories

```
acs_tract <- read_csv("https://raw.githubusercontent.com/ksalazar3/HED696C_Rcla
#> New names:
#> Rows: 11504 Columns: 30
#> -- Column specification
#> -------------------------------------------------------- Delimiter: "," chr
#> (4): tract_name, tract, race_brks_nonwhiteasian, inc_brks dbl (26): ...1,
#> pop_total, pop_total_25plus, median_household_income, pop_wh...
#> i Use `spec()` to retrieve the full column specification for this data. i
#> Specify the column types or set `show_col_types = FALSE` to quiet this messag
#> * `` -> `...1`
```

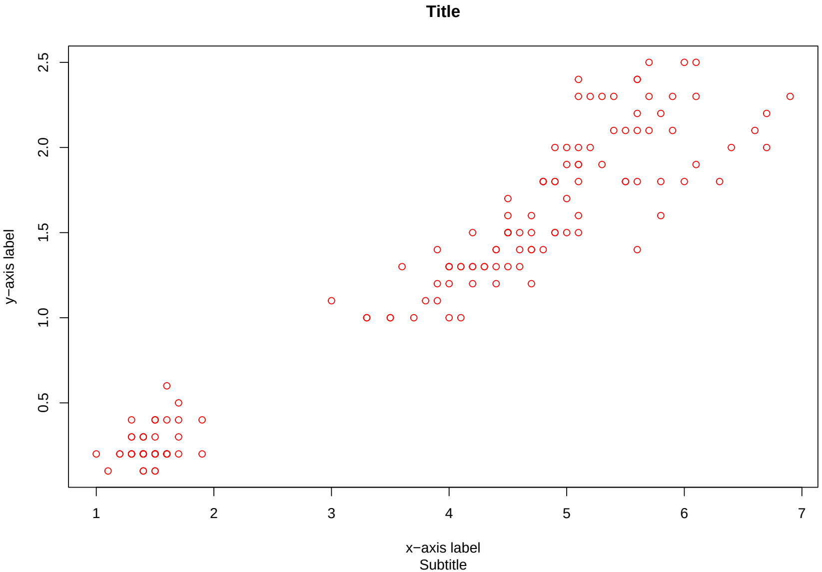
- Names of elements in data objects

```
num_vec <- 1:5
names(num_vec) <- c('uno', 'dos', 'tres', 'cuatro', 'cinco')
num_vec
#>    uno    dos   tres cuatro  cinco
#>      1      2      3      4      5
```

Common uses of strings

- Text elements displayed in plots, graphs, maps

```
plot(iris$Petal.Length, iris$Petal.Width, main = 'Title', sub = 'Subtitle',
     xlab = 'x-axis label', ylab = 'y-axis label', col = 'red')
```



String basics

We will use the `stringr` library for working with strings, rather than Base R

- `stringr` functions have intuitive names and all begin with `str_`
- Base R functions for working with strings can be inconsistent (avoid using them)

Basic functions:

- String length using `str_length()`

```r
#example 1
string2 <- "This is a string"
str_length(string2)
#> [1] 16

#example 2
str_length(c("a", "strings are fun", NA))
#> [1]  1 15 NA
```

Combining strings

- Combining strings using `str_c()`

```
#example 1
x_var <- "x"
y_var <- "y"

str_c(x_var, y_var)
#> [1] "xy"

#example 2
str_c("x", "y")
#> [1] "xy"
```

- Use `sep` argument to control how strings are seperated when combined

```
str_c("x", "y", sep= ", ")
#> [1] "x, y"
```

- `NA` are still contagious, if you want a string "NA" rather than `NA` use `str_replace_na()`

```
street_dir<- c("East", "West", NA)
str_c("Direction: ", street_dir)
#> [1] "Direction: East" "Direction: West" NA
str_c("Direction: ", str_replace_na(street_dir))
#> [1] "Direction: East" "Direction: West" "Direction: NA"
```

Subsetting strings

-Extract parts of a string using `str_sub()` , which uses `start` and `end` arguments to extract the position of the substring wanted

```
fruits<- c("Apple", "Banana", "Orange")

#first three elements
str_sub(fruits, 1, 3) #end argument in inclusive
#> [1] "App" "Ban" "Ora"

#last three elements
str_sub(fruits, -3, -1) #neg nums count backwards from end
#> [1] "ple" "ana" "nge"
```

- **Task: extract 6-digit zip code from `zip9` in `wwlist`**

```
wwlist %>% mutate(
  zip=str_sub(zip9, 1, 5)
)
```

Lower-case and Upper-case functions

- Changing strings to lower or upper case

```
str_to_lower("HELLO")
#> [1] "hello"
str_to_upper("hello")
#> [1] "HELLO"
```

- **Task: lower-case `hs_name` in `wwlist`**

```
wwlist %>% select(receive_date, hs_name) %>%
  mutate(
  hs_name_lwr=str_to_lower(hs_name),
)
#> # A tibble: 268,396 x 3
#>    receive_date hs_name                     hs_name_lwr
#>    <date>       <chr>                       <chr>
#>  1 2016-05-31   Ingraham High School        ingraham high school
#>  2 2016-05-31   Kentwood Senior High School kentwood senior high school
#>  3 2016-05-31   Archbishop Thomas J Murphy HS archbishop thomas j murphy hs
#>  4 2016-05-31   Garfield High School        garfield high school
#>  5 2016-05-31   Lake Stevens High School    lake stevens high school
#>  6 2016-05-31   Franklin High School        franklin high school
#>  7 2016-05-31   Hockinson High School       hockinson high school
#>  8 2016-05-31   Nathan Hale High School     nathan hale high school
#>  9 2016-05-31   Sultan High School          sultan high school
#> 10 2016-05-31   Sandpoint High School       sandpoint high school
#> # i 268,386 more rows
```

Student Exercises

1. Combine `school_type` and `school_category` in the `wwlist` dataframe to create one school type + category varibale. Be sure to seperate type and category using a comma AND deal with contagious NAs by using string "NA" if `school_type` and/or `school_category` are `NA` .
2. The last four digits of `zip9` indicate the delivery route within the 5-digit zip code area. Create a new `route` variable that extracts the last four digits from `zip9` .

Student Excercises (Solutions)

1. Combine `school_type` and `school_category` in the `wwlist` dataframe to create one school type + category varibale. Be sure to seperate type and category using a comma AND deal with contagious NAs by using string "NA" if `school_type` and/or `school_category` are `NA` .

```
wwlist %>% select(school_type, school_category) %>%
  mutate(
    type_cat= str_c(str_replace_na(school_type), str_replace_na(school_category)
  )
#> # A tibble: 268,396 x 3
#>    school_type school_category type_cat
#>    <chr>       <chr>           <chr>
#>  1 public      Regular School  public, Regular School
#>  2 public      Regular School  public, Regular School
#>  3 <NA>        <NA>            NA, NA
#>  4 public      Regular School  public, Regular School
#>  5 public      Regular School  public, Regular School
#>  6 public      Regular School  public, Regular School
#>  7 public      Regular School  public, Regular School
#>  8 public      Regular School  public, Regular School
#>  9 public      Regular School  public, Regular School
#> 10 public      Regular School  public, Regular School
#> # i 268,386 more rows
```

Student Excercises (Solutions)

1. The last four digits of `zip9` indicate the delivery route within the 5-digit zip code area. Create a new `route` variable that extracts the last four digits from `zip9`.

```
wwlist %>% select(zip9) %>%
  mutate(
  route=str_sub(zip9, -4, -1)
)
#> # A tibble: 268,396 x 2
#>    zip9       route
#>    <chr>      <chr>
#>  1 98103-3528 3528
#>  2 98030-7964 7964
#>  3 98290-8659 8659
#>  4 98105-0002 0002
#>  5 98252-9327 9327
#>  6 98108-1809 1809
#>  7 98685-3135 3135
#>  8 98125-4543 4543
#>  9 98294-1529 1529
#> 10 83864-2304 2304
#> # i 268,386 more rows
```

Why are string manipulations useful?

Basic examples:

- Dealing with identification numbers (leading or trailing zeros)

```
typeof(acs_tract$fips_county_code)
#> [1] "double"

acs_tract <- acs_tract %>%
  mutate(char_county=
  str_pad(as.character(fips_county_code), side = "left" ,3, pad="0"))
```

- Complex reshaping (tidying) of data [We will learn this next week!!!]
    - Problem: multiple variables crammed into the column names
        - new_ prefix = new cases
        - sp/rel/sp/ep describe how the case was diagnosed
        - m/f gives the gender
        - digits are age ranges

```
who %>% pivot_longer(
  cols = new_sp_m014:newrel_f65,
  names_to = c("diagnosis", "gender", "age"),
  names_pattern = "new_?(.*)_(.)(.*)",
  values_to = "count"
)
```

Why are string manipulations useful?

Advanced examples:

- Web-scraping
  - Find and scrape all linked pages of recruiters assigned by states: (https://gobama.ua.edu/staff/)
  - Parsing raw HTML to convert it into tabular data
- Natural Language Processing
  - Analyzing university president speeches for promotion of interdisciplinary research (IDR)
  - Predict sentiment of promotion of IDR

Working with Dates and Times

Working with date/time variables

Working with dates and times in data management seems simpler than it really is!

- Does every year have 365 days?
- Does every day have 24 hours?
- Does every minute have 60 seconds?

These details matter for:

- Calculating changes over time
- Analyzing longitudinal data
- Predicting the occurance/timing of events

There are three ways you're likely to create a date/time variable:

- From a string (most common)
- From date and time individual components
- From an existing date/time object

Creating Date/Times

Creating Date/Times from strings

The most common way you're likely to create Date/Time variables is from primary/secondary data where dates and times are recorded and/or stores as strings.

For Dates:

- Use `lubridate` "helpers" to identify the order of year/month/day

```
ymd("2017/01/31")
#> [1] "2017-01-31"

mdy("January 31st, 2017")
#> [1] "2017-01-31"

dmy("31-01-2017")
#> [1] "2017-01-31"

ymd(20170131)
#> [1] "2017-01-31"
```

For Dates:

- Use `lubridate` "helpers" to identify the order of year/month/day **AND** hours/minutes/seconds

```
ymd_hms("2017-01-31 20:11:59")
#> [1] "2017-01-31 20:11:59 UTC"

mdy_hm("01/31/2017 08:01")
#> [1] "2017-01-31 08:01:00 UTC"
```

Creating Date/Times from individual variables

What if your dates and times are recorded across multiple columns/variables?

EX: NYC flights data

```
flights %>%
  select(year, month, day, hour, minute)
#> # A tibble: 336,776 x 5
#>     year month   day  hour minute
#>    <int> <int> <int> <dbl>  <dbl>
#>  1  2013     1     1     5     15
#>  2  2013     1     1     5     29
#>  3  2013     1     1     5     40
#>  4  2013     1     1     5     45
#>  5  2013     1     1     6      0
#>  6  2013     1     1     5     58
#>  7  2013     1     1     6      0
#>  8  2013     1     1     6      0
#>  9  2013     1     1     6      0
#> 10  2013     1     1     6      0
#> # i 336,766 more rows
```

- Create a date variable using `make_date()`

```
flights1<- flights %>%
  select(year, month, day) %>%
  mutate(
    depart= make_date(year, month, day)

  )
```

Using Date/Time Variables

Time spans

Arithmatic with dates works differently than with any numeric type!

There are three date/time classes that represent time spans:

- Durations: represent the duration of time to an exact number of seconds
- Periods: represent the period of time such as weeks/months/years
- Intervals: represent a starting and end point in time

Durations

When you subtract two dates, the result is a `difftime` object

- A `difftime` object records time span as seconds (not intuitive)
- Use `as.duration` to make the difftime object more intuitive (but records time span in seconds)

```
# How old is Karina?
k_age <- today() - ymd(19890321)
k_age
#> Time difference of 13145 days

typeof(k_age)
#> [1] "double"
class(k_age)
#> [1] "difftime"

as.duration(k_age)
#> [1] "1135728000s (~35.99 years)"
```

- Durations uses "constructers"

Time spans

Periods

Periods don't record time spans in exact seconds and are more intuitive to the way we think about time!

```
one_pm <- ymd_hms("2016-03-12 13:00:00", tz = "America/New_York")

one_pm # 1pm
#> [1] "2016-03-12 13:00:00 EST"
one_pm + days(1) #1pm
#> [1] "2016-03-13 13:00:00 EDT"

one_pm + years(1)
#> [1] "2017-03-12 13:00:00 EDT"
```

Mini Lesson on Exploratory Data Analysis (EDA)

New Data: High School Longitudinal Study (HSLS)

Use `read_dta()` function from `haven` to import Stata dataset into R

```
hsls <- read_dta(file="https://github.com/ksalazar3/HED696C_RClass/raw/master/da
```

Let's examine the data [you **must** run this code chunk]

```
hsls %>% names()
hsls %>% names() %>% str()
hsls %>% names() %>% tolower() %>% str()

names(hsls) <- tolower(names(hsls)) # convert names to lowercase
names(hsls)

str(hsls) # ugh

str(hsls$s3classes)
attributes(hsls$s3classes)
typeof(hsls$s3classes)
class(hsls$s3classes)
```

Download the HSLS Codebook:
https://nces.ed.gov/pubs2014/2014361_AppendixI.pdf

What is exploratory data analysis (EDA)?

The Towards Data Science website has a nice definition of EDA:

> *"Exploratory Data Analysis refers to the critical process of performing initial investigations on data so as to discover patterns, to spot anomalies, to test hypothesis, and to check assumptions with the help of summary statistics"*

This course focuses on "data management":

- Investigating and cleaning data for the purpose of creating analysis variables
- Basically, everything that happens **before** you conduct analyses

I think about "exploratory data analysis for data quality"

- Investigating values and patterns of variables from "input data"
- Identifying and cleaning errors or values that need to be changed
- Creating analysis variables
- Checking values of analysis variables agains values of input variables

How we will teach exploratory data analysis

Will teach exploratory data analysis (EDA) in two sub-sections:

1. Introduce "Tools of EDA":
   - Demonstrate code to investigate variables and relatioship between variables
   - Most of these tools are just the application of programming skills you have already learned
2. Provide "Guidelines for EDA"
   - Less about coding, more about practices you should follow and mentality necessary to ensure high data quality

Tools for EDA

Tools of EDA

To do EDA for data quality, must master the following tools:

- **Select, sort, filter, and print** in order to see data patterns, anomolies
  - Select and sort particular values of particular variables
  - Print particular values of particular variables
- **One-way descriptive analyses** (i.e,. focus on one variable)
  - Descriptive analyses for continuous variables
  - Descriptive analyses for discreet/categorical variables
- **Two-way descriptive analyses** (relationship between two variables)
  - Categorical by categorical
  - Categorical by continuous
  - Continuous by continuous

Whenever using any of these tools, **pay close attention to missing values and how they are coded**

- Often, the "input" variables don't code missing values as `NA`
- Especially when working with survey data, missing values coded as a negative number (e.g., `-9`, `-8`, `-4`) with different negative values representing different reasons for data being missing
- Sometimes missing values coded as very high positive numbers
- Therefore, important to investigate input vars prior to creating analysis vars

Tools of EDA

First, Let's create a smaller version of the HSLS:09 dataset

```
#hsls %>% var_label()
hsls_small <- hsls %>%
  select(stu_id,x3univ1,x3sqstat,x4univ1,x4sqstat,s3classes,
         s3work,s3focus,s3clgft,s3workft,s3clgid,s3clgcntrl,
         s3clglvl,s3clgsel,s3clgstate,s3proglevel,x4evrappclg,
         x4evratndclg,x4atndclg16fb,x4ps1sector,x4ps1level,
         x4ps1ctrl,x4ps1select,x4refsector,x4reflevel,x4refctrl,
         x4refselect, x2sex,x2race,x2paredu,x2txmtscor,x4x2ses,x4x2sesq5)
```

```
names(hsls_small)
hsls_small %>% var_label()
```

Tools of EDA: select, sort, filter, and print

We've already know `select()`, `arrange()`, `filter()`

Select, sort, and print specific vars

```
#sort and print
hsls_small %>% arrange(desc(stu_id)) %>%
  select(stu_id,x3univ1,x3sqstat,s3classes,s3clglvl)

#investigate variable attributes
hsls_small %>% arrange(desc(stu_id)) %>%
  select(stu_id,x3univ1,x3sqstat,s3classes,s3clglvl) %>% str()

#investigate variable attributes
hsls_small %>%
  select(stu_id,x3univ1,x3sqstat,s3classes,s3clglvl) %>% var_label()

hsls_small %>%
  select(s3clglvl) %>% val_labels()

#print observations with value labels rather than variable values
hsls_small %>% arrange(desc(stu_id)) %>%
  select(stu_id,x3univ1,x3sqstat,s3classes,s3clglvl) %>% as_factor()
```

Sometimes helpful to increase the number of observations printed

```
class(hsls_small) #it's a tibble, which is the "tidyverse" version of a data fra
options(tibble.print_min=50)
# execute this in console
```

One-way descriptive stats for continuous vars, Tidyverse approach

Use `summarise_at()`, a variation of `summarise()`, to make descriptive stats

- `.args=list(na.rm=TRUE)` = a named list of additional arguments to be added to all function calls

Task:

- calculate descriptive stats for `x2txmtscor`, math test score

```
#?summarise_at
hsls_small %>% select(x2txmtscor) %>% var_label()
#> $x2txmtscor
#> [1] "X2 Mathematics standardized theta score"
hsls_small %>% #old way, still works
  summarise_at(
    .vars = vars(x2txmtscor),
    .funs = funs(mean, sd, min, max, .args=list(na.rm=TRUE))
  )
#> # A tibble: 1 x 4
#>    mean    sd   min   max
#>   <dbl> <dbl> <dbl> <dbl>
#> 1  44.1  21.8    -8  84.9

hsls_small %>%  #this also works
  summarise(across(c(x2txmtscor),
  list(mean= ~mean(.x), stdv= ~sd(.x), min= ~min(.x), max= ~max(.x))))
#> # A tibble: 1 x 4
#>   x2txmtscor_mean x2txmtscor_stdv x2txmtscor_min x2txmtscor_max
#>             <dbl>           <dbl>          <dbl>          <dbl>
```

One-way descriptive stats for continuous vars, Tidyverse approach

Can calculate descriptive stats for more than one variable at a time

Task:

- calculate descriptive stats for `x2txmtscor`, math test score, and `x4x2ses`, socioeconomic index score

```
hsls_small %>% select(x2txmtscor,x4x2ses) %>% var_label()
#> $x2txmtscor
#> [1] "X2 Mathematics standardized theta score"
#>
#> $x4x2ses
#> [1] "X4 Revised X2 Socio-economic status composite"

hsls_small %>% #still works
  summarise_at(
    .vars = vars(x2txmtscor,x4x2ses),
    .funs = funs(mean, sd, min, max, .args=list(na.rm=TRUE))
  )
#> # A tibble: 1 x 8
#>   x2txmtscor_mean x4x2ses_mean x2txmtscor_sd x4x2ses_sd x2txmtscor_min
#>             <dbl>        <dbl>         <dbl>      <dbl>          <dbl>
#> 1            44.1       -0.802          21.8       2.63             -8
#> # i 3 more variables: x4x2ses_min <dbl>, x2txmtscor_max <dbl>,
#> #   x4x2ses_max <dbl>

hsls_small %>%  #this also works
  summarise(across(c(x2txmtscor, x4x2ses),
  list(mean = mean(.x), stdv = sd(.x), min = min(.x), max = max(.x))))
```

One-way descriptive stats for continuous vars, Tidyverse approach

"Input vars" in survey data often have negative values for missing/skips

```
hsls_small %>% filter(x2txmtscor<0) %>% count(x2txmtscor)
```

R includes those negative values when calculating stats; you don't want this

- Solution: create version of variable that replaces negative values with `NA`

```
hsls_small %>% mutate(x2txmtscor_na=ifelse(x2txmtscor<0,NA,x2txmtscor)) %>%
  summarise_at(
    .vars = vars(x2txmtscor_na),
    .funs = funs(mean, sd, min, max, .args=list(na.rm=TRUE))
  )
#> # A tibble: 1 x 4
#>    mean    sd   min   max
#>   <dbl> <dbl> <dbl> <dbl>
#> 1  51.5  10.2  22.2  84.9
```

What if you didn't include `.args=list(na.rm=TRUE)` ?

```
hsls_small %>% mutate(x2txmtscor_na=ifelse(x2txmtscor<0,NA,x2txmtscor)) %>%
  summarise_at(
    .vars = vars(x2txmtscor_na),
    .funs = funs(mean, sd, min, max))
#> # A tibble: 1 x 4
#>    mean    sd   min   max
#>   <dbl> <dbl> <dbl> <dbl>
#> 1    NA    NA    NA    NA
```

One-way descriptive stats for continuous vars, Tidyverse approach

How to identify these missing/skip values if you don't have a codebook?

- `count()` combined with `filter()` helpful for finding extreme values of continuous vars, which are often associated with missing or skip

```
#variable x2txmtscor
hsls_small %>% filter(x2txmtscor<0) %>%
  count(x2txmtscor)
#> # A tibble: 1 x 2
#>   x2txmtscor     n
#>        <dbl> <int>
#> 1         -8  2909

#variable s3clglvl
hsls_small %>% select(s3clglvl) %>% var_label()
#> $s3clglvl
#> [1] "S3 Enrolled college IPEDS level"

hsls_small %>% filter(s3clglvl<0) %>%
  count(s3clglvl)
#> # A tibble: 3 x 2
#>   s3clglvl                          n
#>   <dbl+lbl>                     <int>
#> 1 -9 [Missing]                    487
#> 2 -8 [Unit non-response]         4945
#> 3 -7 [Item legitimate skip/NA]   5022
```

One-way descriptive stats student exercise

1. Using the object `hsls`, identify variable type, variable class, and check the variable values and value labels of `x4ps1start`
   - variable `x4ps1start` identifies month and year student first started postsecondary education
   - **Note**: This variable is a bit counterintuitive.
     - e.g., the value `201105` refers to May 2011
2. Get a frequency count of the variable `x4ps1start`
3. Get a frequency count of the variable, but this time only observations that have negative values **hint**: use filter()
4. Create a new version of the variable `x4ps1start_na` that replaces negative values with NAs and use `summarise_at()` to get the min and max value.

One-way descriptive stats student exercise solutions

1. Using the object `hsls`, identify variable type, variable class, and check the variable vakyes and value labels of `x4ps1start`

```
typeof(hsls$x4ps1start)
#> [1] "double"
class(hsls$x4ps1start)
#> [1] "haven_labelled" "vctrs_vctr"     "double"

hsls %>% select(x4ps1start) %>% var_label()
#> $x4ps1start
#> [1] "X4 Month and year of enrollment at first postsecondary institution"

hsls %>% select(x4ps1start) %>% val_labels()
#> $x4ps1start
#>                                        Missing
#>                                             -9
#>                               Unit non-response
#>                                             -8
#>                         Item legitimate skip/NA
#>                                             -7
#>                         Component not applicable
#>                                             -6
#> Item not administered: abbreviated interview
#>                                             -4
#>                           Carry through missing
#>                                             -3
#>                                      Don't know
```

One-way descriptive stats student exercise solutions

2. Get a frequency count of the variable `x4ps1start`

```
hsls %>%
  count(x4ps1start)
#> # A tibble: 9 x 2
#>   x4ps1start                                 n
#>   <dbl+lbl>                              <int>
#> 1     -9 [Missing]                         107
#> 2     -8 [Unit non-response]              6168
#> 3     -7 [Item legitimate skip/NA]        4281
#> 4 201100                                    57
#> 5 201200                                   206
#> 6 201300                                 10800
#> 7 201400                                  1295
#> 8 201500                                   471
#> 9 201600                                   118
```

One-way descriptive stats student exercise solutions

3. Get a frequency count of the variable, but this time only observations that have negative values **hint**: use filter()

```
hsls %>%
  filter(x4ps1start<0) %>%
  count(x4ps1start)
#> # A tibble: 3 x 2
#>   x4ps1start                        n
#>   <dbl+lbl>                     <int>
#> 1 -9 [Missing]                    107
#> 2 -8 [Unit non-response]         6168
#> 3 -7 [Item legitimate skip/NA]   4281
```

One-way descriptive stats student exercise solutions

4. Create a new version `x4ps1start_na` of the variable `x4ps1start` that replaces negative values with NAs and use `summarise_at()` to get the min and max value.

```
hsls %>% mutate(x4ps1start_na=ifelse(x4ps1start<0,NA,x4ps1start)) %>%
  summarise_at(
    .vars = vars(x4ps1start_na),
    .funs = funs(min, max, .args=list(na.rm=TRUE))
  )
#> # A tibble: 1 x 2
#>      min    max
#>    <dbl>  <dbl>
#> 1 201100 201600
```

One-way descriptive stats for discrete/categorical vars, Tidyverse approach

Use `count()` to investigate values of discrete or categorical variables

For variables where `class==labelled`

```
class(hsls_small$s3classes)
attributes(hsls_small$s3classes)
#show counts of variable values
hsls_small %>% count(s3classes) #print in console to show both
#show counts of value labels
hsls_small %>% count(s3classes) %>% as_factor()
```

- I like `count()` because the default setting is to show `NA` values too!

```
hsls_small %>% mutate(s3classes_na=ifelse(s3classes<0,NA,s3classes)) %>%
  count(s3classes_na)
```

Relationship between variables, categorical by categorical

Two-way frequency table, called "cross tabulation", important for data quality

- When you create categorical analysis var from single categorical "input" var
  - Two-way tables show us whether we did this correctly
- Two-way tables helpful for understanding skip patterns in surveys

key to syntax

- `df_name %>% group_by(var1) %>% count(var2)` **OR**
- `df_name %>% count(var1,var2)`
- play around with which variable is `var1` and which variable is `var2`

Relationship between variables, categorical by categorical

Task: Create a two-way table between `s3classes` and `s3clglvl`

- Investigate variables

```
hsls_small %>% select(s3classes,s3clglvl) %>% var_label()
hsls_small %>% select(s3classes,s3clglvl) %>% val_labels()
```

- Create two-way table

```
hsls_small %>% group_by(s3classes) %>% count(s3clglvl) # show values
hsls_small %>% count(s3classes,s3clglvl)
#hsls_small %>% group_by(s3classes) %>% count(s3clglvl) %>% as_factor() # show v
```

- Are these objects the same?

```
hsls_small %>% group_by(s3classes) %>% count(s3clglvl) %>% glimpse()
#> Rows: 8
#> Columns: 3
#> Groups: s3classes [5]
#> $ s3classes <dbl+lbl> -9, -8,  1,  1,  1,  1,  2,  3
#> $ s3clglvl  <dbl+lbl> -9, -8, -9,  1,  2,  3, -7, -7
#> $ n         <int> 59, 4945, 428, 8894, 3929, 226, 3401, 1621
hsls_small %>% count(s3classes,s3clglvl) %>% glimpse()
#> Rows: 8
#> Columns: 3
#> $ s3classes <dbl+lbl> -9, -8,  1,  1,  1,  1,  2,  3
#> $ s3clglvl  <dbl+lbl> -9, -8, -9,  1,  2,  3, -7, -7
#> $ n         <int> 59, 4945, 428, 8894, 3929, 226, 3401, 1621
```

Relationship between variables, categorical by categorical

Two-way frequency table, also called "cross tabulation"

Task:

- Create a version of `s3classes` called `s3classes_na` that changes negative values to `NA`
- Create a two-way table between `s3classes_na` and `s3clglvl`

```
hsls_small %>%
  mutate(s3classes_na=ifelse(s3classes<0,NA,s3classes)) %>%
  group_by(s3classes_na) %>% count(s3clglvl)

hsls_small %>%
  mutate(s3classes_na=ifelse(s3classes<0,NA,s3classes)) %>%
  count(s3classes_na, s3clglvl)


#example where we create some NA obs in the second variable
hsls_small %>%
  mutate(s3classes_na=ifelse(s3classes<0,NA,s3classes),
         s3clglvl_na=ifelse(s3clglvl==-7,NA,s3clglvl)) %>%
  group_by(s3classes_na) %>% count(s3clglvl_na)


hsls_small %>%
  mutate(s3classes_na=ifelse(s3classes<0,NA,s3classes),
         s3clglvl_na=ifelse(s3clglvl==-7,NA,s3clglvl)) %>%
  count(s3classes_na, s3clglvl_na)
```

Relationship between variables, categorical by continuous

Investigating relationship between multiple variables is a little tougher when at least one of the variables is continuous

Conditional mean (like regression with continuous Y and one categorical X):

- Shows average values of continous variables within groups
- Groups are defined by your categorical variable(s)

key to syntax

- ```
  group_by(categorical_var) %>% summarise_at(.vars = vars(continuous_var)
  ```
  ```

Relationship between variables, categorical by continuous

Task

- Calculate mean math score, `x2txmtscor`, for each value of parental education, `x2paredu`

```
#first, investigate parental education [print in console]
hsls_small %>% count(x2paredu)
hsls_small %>% count(x2paredu) %>% as_factor()
hsls_small %>% select(x2paredu)  %>% val_labels()
```

```
# using dplyr to get average math score by parental education level [print in co
hsls_small %>% group_by(x2paredu) %>%
    summarise_at(.vars = vars(x2txmtscor),
                 .funs = funs(mean, .args = list(na.rm = TRUE)))
#> # A tibble: 8 x 2
#>   x2paredu                                                          x2txmts
#>   <dbl+lbl>                                                           <d
#> 1 -8 [Unit non-response]                                               -
#> 2  1 [Less than high school]                                           4
#> 3  2 [High school diploma or GED or alterntive HS credential]          4
#> 4  3 [Certificate/diploma from school providing occupational trainin~  4
#> 5  4 [Associate's degree]                                              4
#> 6  5 [Bachelor's degree]                                               5
#> 7  6 [Master's degree]                                                 5
#> 8  7 [Ph.D/M.D/Law/other high lvl prof degree]                         5
```

Relationship between variables, categorical by continuous

Task

- Calculate mean math score, `x2txmtscor`, for each value of `x2paredu`

For checking data quality, helpful to calculate other stats besides mean

```
hsls_small %>% group_by(x2paredu) %>% #[print in console]
    summarise_at(.vars = vars(x2txmtscor),
                 .funs = funs(mean, min, max, .args = list(na.rm = TRUE)))
#> # A tibble: 8 x 4
#>   x2paredu                                                    mean   min
#>   <dbl+lbl>                                                  <dbl> <dbl> <d
#> 1 -8 [Unit non-response]                                     -8       -8  -
#> 2  1 [Less than high school]                                 44.3     -8  7
#> 3  2 [High school diploma or GED or alterntive HS credential] 47.2    -8  8
#> 4  3 [Certificate/diploma from school providing occupational ~ 46.4   -8  7
#> 5  4 [Associate's degree]                                    48.9     -8  7
#> 6  5 [Bachelor's degree]                                     53.3     -8  8
#> 7  6 [Master's degree]                                       55.6     -8  8
#> 8  7 [Ph.D/M.D/Law/other high lvl prof degree]               58.9     -8  8
```

Always Investigate presence of missing/skip values

```
hsls_small %>% filter(x2paredu<0) %>% count(x2paredu)
hsls_small %>% filter(x2txmtscor<0) %>% count(x2txmtscor)

hsls_small %>% select(x2paredu) %>% val_labels()
```

Replace `-8` with `NA` and re-calculate conditional stats

Student exercise

Can use same approach to calculate conditional mean by multiple `group_by()` variables

- Just add additional variables within `group_by()`

1. Calculate mean math test score (`x2txmtscor`), for each combination of parental education (`x2paredu`) and sex (`x2sex`).

Student exercise solution

1. Calculate mean math test score (`x2txmtscor`), for each combination of parental education (`x2paredu`) and sex (`x2sex`)

```
hsls_small %>% count(x2sex)

hsls_small %>%
  group_by(x2paredu,x2sex) %>%
  summarise_at(.vars = vars(x2txmtscor),
               .funs = funs(mean, .args = list(na.rm = TRUE))) %>%
  as_factor()
```

Guidelines for EDA

Guidelines for "EDA for data quality"

Assme that your goal in "EDA for data quality" is to investigate "input" data sources and create "analysis variables"

- Usually, your analysis dataset will incorporate multiple sources of input data, including data you collect (primary data) and/or data collected by others (secondary data)

While this is not a linear process, these are the broad steps I follow

1. Understand how input data sources were created
    - e.g., when working with survey data, have survey questionnaire and codebooks on hand
2. For each input data source, identify the "unit of analysis" and which combination of variables uniquely identify observations
3. Investigate patterns in input variables
4. Create analysis variable from input variable(s)
5. Verify that analysis variable is created correctly through descriptive statistics that compare values of input variable(s) against values of the analysis variable

Always be aware of missing values

- They will not always be coded as `NA` in input variables

“Unit of analysis” and which variables uniquely identify observations

“Unit of analysis” refers to “what does each observation represent” in an input data source

- If each obs represents a student, you have “student level data”
- If each obs represents a student-course, you have “student-course level data”
- If each obs represents a school, you have “school-level data”
- If each obs represents a school-year, you have “school-year level data”

How to identify unit of analysis

- data documentation
- investigating the data set

We will go over syntax for identifying unit of analysis in subsequent weeks

Rules for variable creation

Rules I follow for variable creation

1. Never modify "input variable"; instead create new variable based on input variable(s)
   - Always keep input variables used to create new variables
2. Investigate input variable(s) and relationship between input variables
3. Developing a plan for creation of analysis variable
   - e.g., for each possible value of input variables, what should value of analysis variable be?
4. Write code to create analysis variable
5. Run descriptive checks to verify new variables are constructed correctly
   - Can "comment out" these checks, but don't delete them
6. Document new variables with notes and labels

Rules for variable creation

Task:

- Create analysis for variable ses qunitile called `sesq5` based on `x4x2sesq5` that converts negative values to `NAs`

```
#investigate input variable
hsls_small %>% select(x4x2sesq5) %>% var_label()
hsls_small %>% select(x4x2sesq5) %>% val_labels()
hsls_small %>% select(x4x2sesq5) %>% count(x4x2sesq5)
hsls_small %>% select(x4x2sesq5) %>% count(x4x2sesq5) %>% as_factor()

#create analysis variable
hsls_small <- hsls_small %>%
  mutate(sesq5=ifelse(x4x2sesq5==-8,NA,x4x2sesq5)) # approach 1


hsls_small_temp <- hsls_small %>%
  mutate(sesq5=ifelse(x4x2sesq5<0,NA,x4x2sesq5)) # approach 2

#verify
hsls_small_temp %>% group_by(x4x2sesq5) %>% count(sesq5)
```

Skip patterns in survey data

What are skip patterns

Pretty easy to create an analysis variable based on a single input variable

Harder to create analysis variables based on multiple input variables

- When working with survey data, even seemingly simple analysis variables require multiple input variables due to "skip patterns"

What are "skip patterns"?

- Response on a particular survey item determines whether respondent answers some set of subsequent questions
- What are some examples of this?

Key to working with skip patterns

- Have the survey questionnaire on hand
- Sometimes it appears that analysis variable requires only one input variable, but really depends on several input variables because of skip patterns
  - Don't just blindly turn "missing" and "skips" from survey data to `NAs` in your analysis variable
  - Rather, trace why these "missing" and "skips" appear and decide how they should be coded in your analysis variable

Problem Set 7

Overview of problem set due next week

Assignment:

- create GPA from postsecondary transcript student-course level data

Data source: National Longitudinal Study of 1972 (NLS72)

- Follows 12th graders from 1972
  - Base year: 1972
  - Follow-up surveys in: 1973, 1974, 1976, 1979, 1986
  - Postsecondary transcripts collected in 1984

Why use such an old survey for this assignment?

- NLS72 predates data privacy agreements; transcript data publicly available

What we do to make assignment more manageable

- last week's problem set created the input var: numgrade
- we give you some hints/guidelines
- but you are responsible for developing plan to create GPA vars and for executing plan (rather than us giving you step-by-step questions)

Why this assignment?

1. Give you more practice investigating data, cleaning data, creating variables that require processing across rows
2. Real world example of "simple" task with complex data management needs