R class, topics by week

Ozan Jaquette and Patricia Martin

Contents

1 General

1.1 wish list for beamer slides

- smaller font-size; similar to fontsize from Christenson slides
- code in code chunks wraps to next row rather then bleeds off right margin
- reduce horizontal space between line of text and first item of list below that line of text
- when I run a code chunk RMarkdown that includes a print of data, the printout below the chunk looks the way printing data from console looks as opposed to having the df_print: paged style (HTML tables with support for pagination over rows and columns)

1.2 Stuff for students to do before class

1. Install R
2. Install RStudio
3. Install MikTeX
4. Knit following documents in RMarkdown
    - html
    - pdf_document
    - beamer_presentation
5. Watch videos on relative and absolute filepaths
    - the general one
    - the one in R
6. Complete intro to R tutorials
    - which ones

2 Alternative approaches

- current
  - class 1: objects
  - class 2: investigating data patterns
    - investigating objects
      - objectname$variable_name
    - printing data
    - selecting variables
    - sorting data
    - missing values
    - operators
    - filtering data
  - class 3: creating and labeling variables
    - pipes
    - creating variables using mutate
  - attributes and augmented vectors in general with an application to factors
    - review data structures
    - attributes and augmented vectors
  - class 4:
    - group_by
    - summarise()
    - combining group_by and summarise()
      - counts
      - means
      - logical vectors
      - na.rm
    - attach aggregate measures to data frame
  - class 5:
    - labelled/haven package and working with survey data
    - EDA for data quality
    - preparing for creation of GPA
- alternative
  - class 1: objects
  - class 2:
    - pipes
    - selecting variables
    - sorting data
    - filtering data
    - creating variables
  - class 3
    - group_by and summarise()
  - class 4: factors and investigating data patterns/EDA?4
    - data structures and factors
    - review data structures
    - attributes and augmented vectors
    - factors
    - labeling variables
    - EDA for data quality
  - class 5: review of core stuff?

3 Topics by week

3.1 Class 1. Introduction and data types

3.1.1 Big topics

- Course objectives
  - Broad
  - Specific topics we will learn
  - Stuff you can do in R that we will not learn
- What is R? What can R do?
  - Base R
  - R packages
- Complements to R
  - Rstudio
  - RMarkdown
- Introduction to R
  - Open a dataset and print some observations
  - Objects and assignment
    - R is an object oriented programming language
    - Objects in R
    - Assignment
    - Printing objects
  - Functions and help
    - basic syntax
    - help files
  - Data types
  - Packages
  - installing packages
  - loading packages
  - Filepaths

What Patricia says:

2. You might be planning to do this anyway, but I think it would be a good idea to explain what data management is during the first class meeting. Equally important, we should mention who this class is for. What I mean by this is that there may be folks who are experienced with R, but have no clue how to clean data or other folks who have experience with data management, but are new to R, or others who have no experience with either, etc.
3. It would also be a good idea to discuss other uses for R during the first class meeting (graphing, mapping, web scraping, etc.) to introduce students to the various uses of R so they could potentially invest in learning more advanced methods in the future.
4. I agree that strings require at least two weeks, but I also think we should briefly go over them if we can because they may be needed for the weeks on functions and iterations.

3.1.2 Learning objectives

- directories
  - understand difference between absolute and relative filepaths
  - can identify the current working directory
  - change the working directory using absolute filepath
  - change current working directory using relative filepath
- data types
  - Understand the different data types and how they fit into broader categories

  -

3.1.3 Data sources

3.1.4 Reading [after lecture]

3.1.5 Problem set ideas

3.1.6 What covered in Stata data mgt course

- lecture 1:
  - course objectives, why important
  - syllabus and logistics
    - core data manipulation skills you should know
    - homework
    - communication with instructor and classmates
    - course etiquette
  - understanding syntax of stata commands
  - executing stata commands
  - where to find resources and help
  - Stata do-files
  - changing directories
  - Stata log-files

3.1.7 What covered in Christenson class

1. What is data science
2. Course goals
3. Student introductions
4. Course logistics
   - course components
   - course outline
5. Tools for using R
   - what is R; R packages; why R
6. Complementary tools
   - RStudio
   - RMarkdown
7. R basics [review of material from Try R tutorials]
   - creating vectors
   - reading help page
   - indexing (e.g., grab row 3 and column 2)
   - logical expressions
   - Useful functions for data frames

3.2 Class X. Investigating data patterns

3.2.1 Big topics

- ?directories and filepaths?
- investigating objects
- printing data
- selecting variables

- sorting data
- missing values
- operators
- filtering data
- descriptive statistics [cut? move to later week?]

3.2.2 Learning objectives

- Select desired observations, based on numeric and character variables
  - skills: operators; filtering; character variables
- Print desired desired variables and desired observations in desired order for a dataset
  - skills: printing; selecting variables; filtering observations; sorting observations
- Explicitly include or exclude missing observations
  - skills: missing values in quant or character variables
- Conduct basic descriptive statistics: one-way and two-way; categorical and continuous
  - skills:

3.2.3 Data sources

3.2.4 Reading [after lecture]

3.2.5 Problem set ideas

3.2.6 What covered in Stata data mgt course

- isolating variables
  - lookfor
- Isolating specific observations
  - tabulate command
    - with missing option
    - with numlabel
  - isolating observations using "operators"
  - isolating observations, string variables
  - inlist() function
  - missing values
- sorting and listing observations
  - list command [equivalent to print command]
  - assert command
  - combining _variables with bysort [I think save this topic until by-groups week]
  - combining bysort variables and assert [I think save this topic until by-groups week]
- checking variables and cleaning data [I think save this topic until week on data cleaning]
  - assert to check double data entry
  - checking categorical variables, one-way tabulation
  - checking categorical variables, two-way tabulation
  - checking continuous variables

3.2.7 What covered in Christenson class

3.3 Class X. Creating variables, factors, labeling variables

3.3.1 Big topics

- Pipes [DONE]
- Creating variables [DONE]
- class=Factors [DONE]
- class=labelled [or put this in EDA section]
  - labelling variables
  - labelling values

3.3.2 Learning objectives

- gain intuitive understanding of how "pipes" work
- Create variables that are based on calculations across observations and within rows
- be able to add labels to variables
- be able to add value labels to factor variables
- be able to turn string categorical variables into numeric factor variables

3.3.3 questions [for Ben/Crystal]

- How to introduce data types and data structures
  - Skinner introduces concept of data types (logical, numeric, character) and data structures (vectors, lists, etc.);
  - by contrast, Wickham introduces vectors [atomic vectors vs. lists] and then the data types as different types of atomic vectors
    - LINK TO FIGURE
  - which is correct?
- class vs. type

3.3.4 Reading [after lecture]

3.3.5 Data sources

3.3.6 Problem set ideas

- Create variables and add value labels

3.3.7 What covered in Stata data mgt course

- lecture 4:
  - label variables
    - add variable labels
    - add value labels
      - add num labels to value labels
    - view value labels
    - add notes to variables
  - variable storage type and display format
    - internal storage type

    - display format
  - create variables (part 1)
    - principles of variable creation [e.g., don't change input var]
    - Creating variables using different commands
      - introducing egen
    - other

3.3.8 What covered in Christenson class

- lecture 4 [manipulating data frames, 1]
  - introduction to dplyr
    - verbs
      - subsetting: filter, select
      - ordering: arrange
      - transforming: mutate, summarize, group_by
  - subsetting
    - filtering
    - select
    - sample_n, sample_frac, slice
  - ordering
    - arrange
  - transform
    - mutate
- lecture 5 [manipulating data frames, 1]
  - loading data [brief]
  - transform data
    - mutate to create new variables [using the pipe]
    - group_by
    - summarize (aggregate and collapse)
  - presenting summary statistics
    - creating tables
      - using summarize() function
    - plot and aggregate data

3.4 Class X. Processing across rows; exploratory data analysis for data quality

3.4.1 Big topics

- Processing across rows with summarize() and group by

3.4.2 Learning objectives

3.4.3 Order of lecture

1. Processing across obs
   1. introduce group_by() and summarize()
      - Introduce each command on its own and then together
      - ? INTRODUCE SUMMARISE() OR GROUP_BY() FIRST?
      - Common functions to use with summarize (e.g., n _, mean, median, sum, nth)
   2. How missing values handled by summarize be a missing value. Fortunately, all aggregation functions have an na.rm argument which removes the missing values prior to computation:"
   3. Means, counts, and sum

    - counts: n(), which takes no arguments, and returns the size of the current group.
        - talk about n() vs. count() function
    - Counts and proportions of logical values: sum(x > 10), mean(y == 0). When used with numeric functions, TRUE is converted to 1 and FALSE to 0. This makes sum() and mean() very useful: sum(x) gives the number of TRUEs in x, and mean(x) gives the proportion.

4. Other summary functions [5.6.4]
    - measures of position e.g., first(x) x[1]
        - "Measures of position: first(x), nth(x, 2), last(x). These work similarly to x[1], x[2], and x[length(x)] but let you set a default value if that position does not exist (i.e. you're trying to get the 3rd element from a group that only has two elements). For example, we can find the first and last departure for each day:"
5. combining processing across obs with filtering
6. grouping by multiple variables [I haven't re-read this section of wickham yet]
7. appears that you can skip 5.7 grouped/mutates and filters

3.4.4 Reading [after lecture]

- Wickham 5.6 Grouped summaries with summarise()
- Wickham 5.7 Grouped mutates (and filters)

3.4.5 Data sources

- decision
    - start with df_event
    - then do some stuff with df_school
- df_event [focus: how characteristics of visited places [and number of visits] differ by group]
    - one obs per university*recruiting_event
    - potential group_by vars
        - university
        - event type
        - state
        - zip-code
        - in-state vs. out-of-state visits
    - potential things to calculate using summarize() and group_by()
        - number of visits by event_type
        - number of visits in each by_group [in_stae vs. out-of-state, public vs. private]
        - mean income for in-state vs. out-of-state visits
- df_school [focus: how characteristics of visited places differ from non-visited places]
    - one obs per high school [both visted and non-visited]
    - potential group_by vars
        - visited vs. non-visited schools
        - in-state vs. out-of-state
        - public vs. private
        - state
    - potential things to calculate using summarize() and group_by()
        - mean enrollment at visited schools vs. non-visited [and state]
        - mean income at visited vs. non-visited
        - number of visited, number of non-visited
        - mean race at visited vs. non-visited schools
        - academic achievement at visited vs. non-visited
- wwlist [not that many things to calculate besides counts]
    - one obs per prospect

- potential group_by vars
    - state
    - high school
    - zip code
    - gender
    - race
- potential things to calculate using summarize() and group_by()
    - number of prospects purchased by:
        - state, high school, zip-code, country

3.4.6 Problem set ideas

3.4.7 What covered in Stata data mgt course

- Lecture 8: processing across observations, part 1
    - processing across observations using egen command
        - egen to count number of obs in a group; to add all obs in a group for a variable
        - going through some of the sub-commands for egen; count(var) total(var) mean(var)
        - combine processing across obs w/ bysort and egen
    - processing across obs by combining bysort with _variables
        - introduce _n _N
        - introduce sum() function
    - collapsing obs using _variables
        - bysort id: keep if n== N
        - Collapsing obs using variables, then merging
- Lecture 9: processing across observations, part 2 [lots of student exercises]
    - Processing across obs using _variables
        - table of _variables. column 1= _variable (e.g., _n, _n-1, 1, 2, _N); column 2=meaning of _var
        - calculations within subgroups using _vars
        - various student exercises
    - Filling down obs using _variables
        - filling down using _vars and doing same thing using egen
        - filling down to create imputation vars
    - time series operators for lag and lead obs

3.4.8 What covered in Christenson class

- lecture 5 [manipulating data frames, 1]
    - transform data
        - mutate to create new variables [using the pipe]
        - group_by
            - identifies potential groups in stop-and-frisk data
            - group_by example: is an individual above or below the average age in their borough
        - summarize (aggregate and collapse)
    - presenting summary statistics
        - creating tables
            - using summarize() function
        - plot and aggregate data

3.5 Class 5. Survey data and Exploratory data analysis [for data quality]

3.5.1 Big topics

- explaining haven and labelled package; class=labelled [DONE]
  - difference between class=labelled vs. class=factors [DONE]
- EDA for data quality [focus on what commands to use] [DONE]
- Rules for investigating data, creating variables, checking variables [DONE]
  - How to be a good RA/researcher [NOT DONE]
- Dealing with skip patterns in survey data [DONE]
- Brain-storming how to attack create_gpa assignment [DONE]

3.5.2 to do [8/29/2018]

- make slides for reading in data from other formats/labelled package [done]
  - slides for labelled package; difference between class=labelled vs. class=factors
- Go through steps in EDA for data quality
  - first, go through all the different checks;
  - second, rules for investigating and creating variables
    - focus on missing values for input variables
    - do some examples where analysis variable has only one input variable [and you can do simple recode]
- skip patterns:
  - make slides for rules/explanations
  - make slides for example
- Brain-storming how to attack the create_gpa assignment
  - you walk them through EDA to investigate data
  - they brain-storm about how to attack the problem

3.6 Learning objectives

3.6.1 Order of lecture

- EDA for data quality [focus on what commands to use], the tidyverse way [ignore Base R]
  - Investigate objects
  - investigate variable labels/value labels
  - summary stats for continuous vars
  - summary stats for vars w/ discrete values
  - two-way table when both vars have discrete values

3.6.2 Data sources for lecture

- one of the recruiting datasets for basics of EDA for data quality analysis?
- maybe use NLS data for everything?
- NLS dataset for
  - skip patterns
  - brain_storming how to tackle problem set

3.6.3 Problem set ideas

- create GPA from NLS data

3.6.4 What Ben skinner covers

```
- reading in data from Stata using read_dta
- labels
    - variable labels
    - value labels
- summary statistics for continuous variables
    - Base R
        - mean() and sd() functions...
    - tidyverse
        - summarise_at()
- One-way Discrete/categorical variables
    - "as_factor()"
    - base R
        - table() function
    - tidyverse
        - using count() to create a table;
            - benefit is that it automatically includes NAs
- Two-way table
    - base R
        - table(as_factor(df$bypared), as_factor(df$bysex))
    - tidyverse
        -
        - df %>% group_by(bysex) %>% count(bypared) %>% as_factor()
        - to have one variable as columns and one variable as rows
            - df %>% group_by(bysex) %>% count(bypared) %>% as_factor() %>% spread(bysex, n)
- conditional mean [good for categorical by continuous]
    - tidyverse
        - mean of continuous var for each value of categorical
            - df %>% group_by(bypared) %>% summarise_at(.vars = vars(bynels2m), .funs = funs(mean, .arg
        - mean of continuous var for each value of categorical by categorical
        - use "spread" to make it look like published table
```

3.6.5 What covered in Stata data mgt course

3.6.6 Reading [after lecture]

3.7 Class 6. tidying/reshaping data

3.7.1 Big topics

- what is tidy data;
  - key concepts/vocabulary
  - rules of tidy data
  - how to diagnose tidy data
  - steps in tidying data
    - figure out what the variables and observations are
    - resolve problems (via spreading+gathering; via separating+uniting:
- spreading and gathering
- Separating and uniting
- missing values
- tibbles [put this last because tidying is main event?]

3.7.2 Learning objectives

- understand concepts of unit of analysis, variable, etc.
- understand rules of tidy data
- how to diagnose whether data is tidy and what principle the data violates

3.7.3 Order of lecture

- what is tidy data and why we want tidy data?

3.7.4 Data sources for lecture

- will have to figure out data sources for each type of untidy data covered by wickham chapter 9

3.7.5 Problem set ideas

- IPEDS data task? look at tast from Stata data mgt class

3.7.6 What covered in Stata data mgt course

- stuff from stata data mgt course that could go here
  - slide title: Reshaping: Changing the organizational structure of data
    - Creating analysis datasets often require changing the organizational structure of data
    - Examples:
      - You want analysis dataset to have one obs per student, but your data has one obs per student-course
      - You want analysis dataset to have one obs per institution, but enrollment data has one obs per institution-enrollment level
    - Two common ways to change organizational structure of data
      - Perform calcs across obs and then, keep if n== N
      - We did this in previous lecture
      - Use the reshape command
  - what is "long" form data
    - defines conceptually
    - shows what it looks like
  - what is "wide" form data
    - defines conceptually
    - shows what it looks like
  - why reshape?
    - Statistical methods often require the data to be in specific form
      - For example, most longitudinal regression models require data to have one obs per each combination of unit and time-period (e.g., one obs per each organization-year) - reshaping from long to wide:
    - important step is to write it out conceptually; what form of data do you have? what form of data do you want?
    - Step 1: conceptualize current and desired data structure
      - First, have a conceptual understanding of what you have and what you want
      - What is the current organizational structure of your (long) data?
      - Investigate organizational structure in Stata
      - Draw a simplified picture
      - What is the desired organizational structure of your (wide) data?

            - Draw a simplified picture

        - i skipped discussion of reshape command
    - reshaping from wide to long
    - final comments on reshaping
        - Reshaping is conceptually challenging; takes practice to feel comfortable
        - On scrap paper, draw how dataset currently organized and draw how you want the reshaped data to be organized
        - Often takes trial and error to get reshape to work correctly
        - Often dataset is not purely "wide" or "long"; important thing is that you know who data are currently structured and how you want it to be structured

3.7.7 What is in Wickham book?

- no need to take detailed notes because basically you are going to work through this chapter

3.7.8 What Ben skinner covers

- nothing that is not covered in wickham

3.7.9 What covered in Christenson course [lecture 7, tidy data]

- What is tidy data?
    - qualities of tidy data [from wickham]
    - example of tidy data [from wickham]
    - examples of untidy data [several slides]
    - note that he mentions "wide" vs. "long" data
    - why tidy data? [this is good, and do ths slide after you define tidy vs. untidy]
        1. Consistency & Efficiency: > If you have a consistent data structure, it's easier to learn the tools that work with it because they have an underlying uniformity.
        2. Optimized for R: Both `dplyr` and base R functions work with vectors of values.
           In a tidy dataset, each column is the vector of values for a given variable.
    - real world examples of untidy data [now he uses his own data, performance metrics for LA city departments]
- how do we create tidy data?
    - sketching tidy data [does this in two separate slides; one with questions and one with answers]
        - In our recipe for visualization, we started by sketching the figure we wanted to create.

        - I find it useful to do the same for cleaning data.
        - What makes `untidy_df` untidy (i.e., what principle(s) is violated)?
        - What would a tidy version of this data look like?
- cleaning data with tidyr: gather and spread
    - first goes over gather for his real world example
    - then goes over spread for his real world example
    - shows the benefit of tidy data: do some task with untidy data; now do this task with tidy data
    - student task: practice gathering, spreading
- cleaning data with tidyr: separate and unite
    - separate: a column sometimes contains two pieces of information
        - A column sometimes contains two pieces of information.
        - The `country_year` column includes both the country and year for a given observation.
        - These are separate variables and, as such, deserve their own columns.
    - unite

        * You won't often have a use for `unite`, but you can use it combine multiple columns.
- task: more practice tidying data:
    - What principle of tidy data does each table violate?
    - What is the unit of analysis? What are the variables?
    - Use the functions in `tidyr` to tidy these datasets.

3.7.10 Reading [after lecture]

- Wickham chapter 12

3.8 Class 7. relational data

- Maybe put class on relational data before class on acquiring data? Why:
    - the earlier we put merging in quarter, the more practice students can get merging in subsequent weeks
    - this way in relational data class can just focus on merging data sources.
    - then in acquiring data class, can do problem sets that read in data and towards the end of problem set have questions that ask them to merge data

3.8.1 Big topics

- concepts/terminology of (SQL) merging

- investigate data structure of both tables prior to merging

- appending data [at the end]

3.8.2 Learning objectives

3.8.3 Order of lecture

- introduction
- introduce NLS datasets we are working with
- keys
    - primary key
    - foreign key
    - "relation"
    - surrogate key
    - when a table doesn't have a primary key, create one. called surrogate key
    - exercise
        * identify the primary and foreign keys for each dataset
- mutating joins
- filtering joins [? cover this?]
- appending data [at the end]

3.8.4 Data sources for lecture

- NLS and datasets created on the fly for simple examples

3.8.5 Problem set ideas

3.8.6 Wickham chapter [13: relational data]

- Intro
  - Mutating joins, which add new variables to one data frame from matching observations in another.
  - Filtering joins, which filter observations from one data frame based on whether or not they match an observation in the other table.
  - Set operations, which treat observations as if they were set elements.
- Keys
  - primary key
  - foreign key
- mutating joins
  - understanding joins [shows visual representation]
  - inner join [An inner join matches pairs of observations whenever their keys are equal:]
  - outer join:
    - An inner join keeps observations that appear in both tables. An outer join keeps observations that appear in at least one of the tables. There are three types of outer joins:
    - three types of outer joins
      - A left join keeps all observations in x.
      - A right join keeps all observations in y.
      - A full join keeps all observations in x and y.
  - duplicate keys
    - So far all the diagrams have assumed that the keys are unique. But that's not always the case. This section explains what happens when the keys are not unique. There are two possibilities:
    - first, One table has duplicate keys. This is useful when you want to add in additional information as there is typically a one-to-many relationship.
    - second, Both tables have duplicate keys. This is usually an error because in neither table do the keys uniquely identify an observation
  - defining the key columns
    - So far, the pairs of tables have always been joined by a single variable, and that variable has the same name in both tables. That constraint was encoded by by = "key". You can use other values for by to connect the tables in other ways:
- filtering joins
  - Filtering joins match observations in the same way as mutating joins, but affect the observations, not the variables.
  - There are two types:
    - semi_join(x, y) keeps all observations in x that have a match in y.
    - anti_join(x, y) drops all observations in x that have a match in y.
  - Basic difference between filtering joins and mutating joins
    - in filtering joins, only the existence of a match is important; it doesn't matter which observation is matched. This means that filtering joins never duplicate rows like mutating joins do:
  - anti-joins useful for diagnosing mismatches in mutating joins
- join problems [IMPORTANT]
  - 3 things to do to make sure your join goes smoothly
- set operations [skip]

3.8.7 What Ben skinner covers

- not really covered

3.8.8 What covered in Stata data mgt course [lecture 7]

- uniquely identify observations in the dataset
- Jargon of merging
  - master
  - using
  - key
  - rule: "key" variable(s) must uniquely identify observations in the "master" dataset or the "using" dataset
- one-to-one merges
  - investigate data structure
  - merge
- one-to-many merges
- merges with more than one "key" variable
- recommendations for merging
  - Can do "one-to-one," "one-to-many," or "many-to-one" merge, but Never do a "many-to-many" merge
  - Investigate data structure of "master" and "using" datasets before merging
  - Make sure to change directories prior to merging; or specify filepath of "using" dataset within the merge statement
  - Investigate observations that don't merge
    - Sort and list specific observations
- common merging problems
  - Same variables across datasets
    - Except for the "key" variable(s), all variable names should be different across datasets
    - If "master" and "using" datasets have same non-key variables, merged dataset retains values of "master"
  - Conflicts in "key" variable(s)
    - "key" variable (e.g., id) should have same name across datasets
    - "key" variable cannot be numeric in "master" dataset but string in "using," and vice versa
    - Use tostring, destring, and/or string functions (e.g.,substr) to modify "key" variables

3.8.9 What covered in Christenson course [lecture 10]

- keys:
  - primary key
  - foreign key
- types of joins
  - inner
  - left
  - full
  - right
- what can go wrong
  - columns w/ same name but different info
  - observations with missing keys
  -
- how do you know if a merge worked correctly?
  - does your data have correct dimensions? if you have no missing observations, then an inner_ and left_ join should return the same thing
  - print out results for a sample of your data
- Appending
  - note: he uses `bind_rows()` function, which is in dplyr package

3.8.10 Reading [after lecture]

3.9 Class 8. acquiring data [Patricia writes]

3.9.1 Big topics

3.9.2 Learning objectives

3.9.3 Order of lecture

3.9.4 Data sources for lecture

- NLS datasets

3.9.5 Problem set ideas

3.9.6 What Ben skinner covers

3.9.7 What covered in Stata data mgt course

3.9.8 What covered in Christenson course

3.9.9 Reading [after lecture]

3.10 Class 9. functions

3.10.1 Big topics

-

3.10.2 Order of lecture

- what are functions [DONE]
- when to write a function vs. not [DONE]
- basics of how writing functions
  - how to write a function; how to improve a function
    - basics of functional arguments (19.5); put stuff from 19.5 function arguments here
  - non-practical example (e.g., namage)
  - practical exampless
- conditional execution [19.4]
  - how to
  - practical examples
- [more advanced] functional arguments, more advanced part of 19.5; [OMIT SOME OF THIS STUFF]
  - how to [don't need to show everything]
    - default values
    - choosing names
    - checking values
    - dot-dot-dot
    - pipeable functions?
  - practical examples

potential function examples

- calculate mean
- calculate the proportion missing
- calculate z score for each observation [number of standard deviations from mean] z = (x_i - xbar)/sd(x)
- count the number of NAs;
- count the number of negative values

- create "v2" version of a variable that converts values you specify to NA
- convert negative values to NA [from skinner]
  - note: this requires you to return(x)

potential practical examples - reading in the admissions data for 2014-15 to 2016-17? [very simple one] - functions for the recruiting research data - from skinner: https://www.btskinner.me/rworkshop/modules/programming_one.html - Moving to a more realistic example, we could make a function that filled in missing values, a common task we've had. First, we'll generate some fake data with missing values.

3.10.3 Questions:

- ? include stuff about vectors from chapter 20?

3.10.4 Learning objectives

3.10.5 Data sources for lecture

- recruiting data for functions [from Karina]
- IPES for reading in data
- IPEDS/HLS/recruiting for fixing missing values
-

3.10.6 What covered in Wickham chapter

- from chapter 17 [intro to the "programming" unit of the book]
  - stuff they say about purpose of programming is helpful
  - also second paragraph about revising;
  - overview of chapter 19: functions
    - Copy-and-paste is a powerful tool, but you should avoid doing it more than twice. Repeating yourself in code is dangerous because it can easily lead to errors and inconsistencies. Instead, in functions, you'll learn how to write functions which let you extract out repeated code so that it can be easily reused
  - overview of chapter 20: vectors
    - As you start to write more powerful functions, you'll need a solid grounding in R's data structures, provided by vectors. You must master the four common atomic vectors, the three important S3 classes built on top of them, and understand the mysteries of the list and data frame

chapter 19: functions - 19.2 When you should write a function - whenever you've copied and pasted a block of code more than twice - steps in writing a function: 1. name 2. arguments 3. body - 19.3 functions are humans and computers - writing functions that humans can understand - names should be verbs generally - 19.4 conditional execution

```
if (condition) {
  # code executed when condition is TRUE
} else {
  # code executed when condition is FALSE
}
```

```
- conditions must evaluate to TRUE or FALSE
- multiple conditions using this:
```

```
if (this) {
  # do that
} else if (that) {
  # do something else
} else {
  #
}
```

```
- style guidelines for writing code
```

- 19.5 function arguments
  - skip dot-dot-dot?
  - skip lazy evaluation
- 19.6 return values
  - explicit return statements [skip? omit?]
  - writing pipeable functions [skip? omit?]
    - this is important; but may feel a little too advanced
- 19.7 environment [skip]

3.10.7 What covered in Stata data mgt course [lecture 12]

lecture 12: creating user-defined programs

- Introduction to user-defined programs
  - difference between user defined programs and loops; when to use one vs. the other
  - think of built in functions like sum() as user-defined programs written by creators of Stata;
  - you write functions to do something you want to do
  - basic steps of user-defined programs
    - define
    - load
    - call
    - Show in Stata do-file, create "Hello" program
  - Core concept of programs: parameters and arguments
    - Most programs (not the "hello" program) perform some task based on one or more "inputs" supplied by you in program call statement
    - What program does depends on values you assign to inputs Parameters
    - A parameter is a name for an input taken in by the program
    - A parameter is a name used inside the program definition to refer to the value passed into the program
    - Arguments: An argument is the value (or contents) we assign to the
    - parameter name
  - lecture writes a namage function that takes input as name and takes date of birth as name
  - discusses using macro list and trace commands to understand program parameters We specify this value in the program call statement

- Practical example: write exible program to read in data

    - Practical example: create program to import IPEDS enrollment data
- Practical example: write "utility" program to rename all variables to uppercase or lowercase
- Practical example: write program to run regression models using different model specifications

3.10.8 What covered in Christenson course

- why write functions
    - DRY: do not repeat yourself
- example of doing something without and with a function
    - center a set of variables: for each variable, subtract the mean from each observation
- three components of a function
    - name
    - arguments
    - body
- recipe for writing a function
    - perform the operation without function
    - get operation to work on a separate set of inputs
    - write function
    - perform (quick) tests
- demonstrates functions:
    - center a set of variables: for each variable, subtract the mean from each observation
    - write percentage change function
    - renaming columns
- student exercises: tasks from wickham text and wickham exercises
    - function that calculates mean and sd
    - function calculates proportion missing
    - function converts Fahrenheit to celsius
    - function extracts initials from a name
    - function counts how often X greater than some constant
- conditional execution
    - basics
    - example
    - conditional execution: multiple conditions

3.10.9 What Ben skinner covers

- module on programming 1 has functions at the end
    - https://www.btskinner.me/rworkshop/modules/programming_one.html

3.10.10 Reading [after lecture]

3.10.11 Problem set ideas

3.11 Class 10. iteration

3.11.1 Learning objectives

3.11.2 big topics/Order of lecture

- [maybe] start w/ a refresher on length and indices?

3.11.3 Data sources for lecture

3.11.4 Problem set ideas

3.11.5 what Wickham covers

3.11.6 What Ben skinner covers

(https://www.btskinner.me/rworkshop/modules/programming_one.html)

- for() function allows you to build loops
  - [like his approach] The for() function allows you to build loops. There are few ways to use for(), but its construction is the same: for(variable in sequence).
  - Reading it backwards, the sequence is just the set of numbers or objects that we're going to work through. The variable is a new variable that will temporarily hold a value from the sequence in each run through the loop. When the sequence is finished, so is the loop.
  - First, let's loop through a sequence of 10 numbers, printing each one at a time as we work through the loop.

```r
## make vector of numbers between 1 and 10
num_sequence <- 1:10

## loop through, printing each num_sequence value, one at a time
for (i in num_sequence) {
    print(i)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
```

```
- Notice the braces {} that come after for(). This is the code in the loop that will be repeated as lon
- Let's do it again, but this time with characters.
```

```r
## character vector using letters object from R base
chr_sequence <- letters[1:10]

## loop through, printing each chr_sequence value, one at a time
for (i in chr_sequence) {
    print(i)
}
```

```
## [1] "a"
## [1] "b"
## [1] "c"
## [1] "d"
## [1] "e"
## [1] "f"
## [1] "g"
## [1] "h"
## [1] "i"
## [1] "j"
```

- loop that works through a vector by its indices
  - explains how this code works:

```
## for loop by indices
for (i in 1:length(chr_sequence)) {
    print(chr_sequence[i])
}
```

```
## [1] "a"
## [1] "b"
## [1] "c"
## [1] "d"
## [1] "e"
## [1] "f"
## [1] "g"
## [1] "h"
## [1] "i"
## [1] "j"
```

```
- Inside the for() parentheses, we have i in 1:length(chr_sequence). We know what i in means since it's

- Inside the braces ({}), we have print(chr_sequence[i]). From the first module, we know that brackets
```

- while

3.11.7 What covered in Stata data mgt course [lecture 11]

lecture 11: macros and looping

- intro to automating your work
- global macro vars
- local macro vars
- looping
  - explanation of general concepts
    - iteration
    - explanation of how "hello" program works
    - what happens at each iteration
    - trace command
  - first example is essentially, the "hello" program
  - foreach y in 2013 2014 2015 {
    2. display as result "My age in y was" y-1979
    3. }
  - task: write hello loop for people in your family
  - practical example:
    - import IPEDS 12-month enrollment for 2012,2013, and 2014
    -
  - practical example: create new variables using loops
    - use loops to create 0/1 versions of imputation variables
  - Practical example 3: Creating counter variable and running regressions in a loop
    - Show example of creating counter, without using dataset
    - Show example of running regression in loop and using counter to save model results
  - show forvalues [as opposed to foreach]
  - debugging loops

        * use “display” command to check whether macro variable contents are what you want them to be
            · Long: “the first thing I do when I have a problem with a loop is to use display to show the value of the local created by foreach or forvalues”
        * use trace command to see what loop is actually doing

3.11.8 What covered in Christenson course [lecture 11]

Lecture 11: functions

- why write functions [from wickham]

3.11.9 Reading [after lecture]