Lecture 2: Investigating data patterns

Managing and Manipulating Data Using R

Introduction

What we will do today

1. Introduction

2. R Markdown

3. More R basics: functions and directories
   - 3.1 Introduction to using functions
   - 3.2 Directories and filepaths

4. Investigating objects
   - 4.1 Variables names
   - 4.2 View and print data
   - 4.3 Missing values

5. Investigating data patterns with tidyverse
   - 5.1 Select variables
   - 5.2 Filter rows
   - 5.3 Arrange rows

Libraries we will use today

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

```
library(tidyverse)
```

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

- Generic syntax: `install.packages("package_name")`
- Install "tidyverse": `install.packages("tidyverse")`

Note: when we load package, name of package is not in quotes; but when we install package, name of package is in quotes:

- `install.packages("tidyverse")`
- `library(tidyverse)`

R Markdown

What is R Markdown

Borrowing from Darin Christensen:

- R Markdown documents embed R code, the output associated with R code, and text into one document
- An R Markdown document is a " 'Living' document that updates every time you compile ["knit"] it"
- R Markdown documents have the extension .Rmd
  - can think of them as text files with the extension .Rmd rather than .txt
- At top of .Rmd file you specify the "output" style, which dictates what kind of formatted document will be created
- When you compile ["knit"] a .Rmd file, the resulting formatted document can be an HTML document, a PDF document, an MS Word document, or many other types

How we will be using R Markdown files in this class:

- homework you submit will be .Rmd files, with "output" style will be `html_document` or `pdf_document`
- lectures we write are .Rmd files, where we the output style will usually be `beamer_presentation`
  - this is essentially a PDF document, where each page is a slide

Creating RMarkdown documents

Do this with a partner

Approach for creating a RMarkdown document.

1. Point-and-click from within RStudio
    - Click on *File* » *New File* » *R Markdown* » *Document* » choose *HTML* » click *OK*
    - save the .Rmd file [any name, anywhere you can find it]
    - "Knit" the entire .Rmd file
        - point-and-click OR shortcut: **Cmd/Ctrl + Shift + k**

Components of a .Rmd file

An RMarkdown (.Rmd) file consists of several parts

1. **YAML header**
   - YAML stands for "yet another markup language"
   - controls settings that apply to the whole document (e.g., "output" should be html_document or pdf_document, whether to include table of contents, etc.)
   - YAML header goes at very top of document
   - starts with a line of three horizontal dashes `---`; ends with a line of three horizontal dashes `---`
2. **Text** in body of .Rmd file
   - e.g., headings; description of results, etc.
3. **R code chunks** in body of .Rmd file

```r
a <- c(2,4,6)
a
a-1
```

4. **R output** associated with code chunks

```
#> [1] 2 4 6
#> [1] 1 3 5
```

Comment: Running R code chunks vs. “knit” entire .Rmd file

Two ways to execute R commands in .Rmd file:

1. “Knit” entire .Rmd file
    - shortcut: **Cmd/Ctrl + Shift + k**
2. “Run” code chunk or selected lines within code chunk
    - Run selected line(s): **Cmd/Ctrl + Enter**
    - Run current chunk: **Cmd/Ctrl + Shift + Enter**

Comment on default settings for RStudio:

- When you knit entire .Rmd file, “objects” created within .Rmd file will not be available after file comples
- When you run code chunk (or selected lines in chunk), objects created by lines you run will be in your “environment” until you remove them or quit R session

Output types of .Rmd file

Common/important output types:

- **html_document**: R Markdown originally designed to create HTML documents
  - Most features/code in .Rmd files were written for html_document
  - many of these features are available in other output types
  - When learning R Markdown, best to start by learning html_document
- **pdf_document**: Requires installation of LaTeX (MiKTeX/MacTeX)
  - How it works:
    - You write .Rmd code;
    - When you compile, this .Rmd code is transformed into LaTeX code
    - LaTeX "engine" creates the formatted .pdf file
  - Can include some of the same features available for *html_document*
  - Can insert LaTeX commands in .Rmd file with *pdf_document* output
- **beamer_presentation**: Requires installation of LaTeX
  - "beamer" is the name for presentations written in LaTeX
  - essentially creates PDF of presentation slides
  - Lectures for this class created with *beamer_presentation* output
  - note: YAML header includes `beamer_header.tex` file, which creates some formatting rules and additional commands

Learning more about R Markdown

Resources

- Cheat sheets and quick reference:
  - Cheat Sheet
  - Quick Reference [I prefer the quick reference]
- Chapters/books
  - Chapter 27 of "R for Data Science" book
  - R Markdown: The Definative Guide book [I prefer this book]

How you will learn R Markdown

- Lectures written as .Rmd file
  - During class run "code chunks" and try to "knit" entire .Rmd file
- I'll assign **small** amount of reading on R Markdown
  - prior to next week:
    - spend 10-15 minutes familiarizing yourself with Quick Reference
    - Read section 3.1 of R Markdown: The Definative Guide, about creating html_document
- Homework must be written in .Rmd file
  - you submit .Rmd file AND output of compiled file
  - for next week, you will submit homework as html_document output

More R basics: functions and directories

Introduction to using functions

What are functions

Functions are pre-written bits of code that accomplish some task.

Functions generally follow three sequential steps:

1. take in an **input** object(s)
2. **process** the input.
3. **return** (A) a new object or (B) a visualizatoin (e.g., plot)

For example, `sum()` function calcualtes sum of elements in a vector

1. **input**. takes in a vector of elements (numeric or logical)
2. **processing**. Calculates the sum of elements
3. **return**. Returns numeric vector of length=1; value is sum of input vector

```
sum(c(1,2,3))
#> [1] 6
typeof(sum(c(1,2,3)))
#> [1] "double"
length(sum(c(1,2,3)))
#> [1] 1

sum(c(TRUE,TRUE,FALSE))
#> [1] 2
typeof(sum(c(TRUE,TRUE,FALSE))); length(sum(c(TRUE,TRUE,FALSE)))
#> [1] "integer"
#> [1] 1
```

Function syntax

Components of a function

- function name (e.g., `sum()`, `length()`, `seq()`)
- function arguments
  - Inputs that the function takes, which determine what function does
    - can be vectors, data frames, logical statements, etc.
  - In "function call" you specify values to assign to these function arguments
    - e.g., `sum(c(1,2,3))`
  - Separate arguments with a comma `,`
    - e.g., `seq(10,15)` Example: the sequence function, `seq()`

```
seq(10,15)
#> [1] 10 11 12 13 14 15
```

Function syntax: More on function arguments

Usually, function arguments have names

- e.g., the `seq()` function includes the arguments `from` , `to` , `by`
- when you call the function, you need to assign values to these arguments; but you usually don't have to specify the name of the argument

```
seq(from=10, to=20, by=2)
#> [1] 10 12 14 16 18 20
seq(10,20,2)
#> [1] 10 12 14 16 18 20
```

Many function arguments have "default values", set by whoever wrote function

- if you don't specify a value for that argument, the default value is inserted
- e.g., partial list of default values for `seq()` : `seq(from=1, to=1, by=1)`

```
seq()
#> [1] 1
seq(to=10)
#>  [1]  1  2  3  4  5  6  7  8  9 10
seq(10) # R assigned value of 10 to "to" rather than "from" or "by"
#>  [1]  1  2  3  4  5  6  7  8  9 10
```

Function arguments, the `na.rm` argument

When R performs calculation and an input has value `NA`, output value is `NA`

```
5+4+NA
#> [1] NA
```

R functions that perform calculations often have argument named `na.rm`

- `na.rm` argument asks whether to remove `NA` values prior to calculation
- For most functions, default value is `na.rm = FALSE`
    - This means "do not remove `NAs`" prior to calculation
    - e.g., default values for `sum()` function: `sum(..., na.rm = FALSE)`

```
sum(c(1,2,3,NA), na.rm = FALSE) # default value
#> [1] NA
sum(c(1,2,3,NA))
#> [1] NA
```

- if you specify, `na.rm = TRUE`, `NA` values removed prior to calculation

```
sum(c(1,2,3,NA), na.rm = TRUE)
#> [1] 6
```

Help files for functions

To see help file on a function, type `?function_name` without parentheses

```
?sum
?seq
```

Contents of help files

- **Description**. What the function does
- **Usage**. Syntax, including default values for arguments
- **Arguments**. Description of function arguments
- **Details**. Details and idiosyncracies of about how the function works.
- **Value**. What (object) the function “returns”
  - e.g., `sum()` returns vector of length 1 whose value is sum of input vector
- **References**. Additional reading
- **See Also**. Related functions
- **Examples**. Examples of function in action
- Bottom of help file identifies the package the function comes from

Practice!

- when you encounter new function, spend two minutes reading help file
- over time, help files will feel less cryptic and will start to feel helpful

Function arguments, the dot-dot-dot (`...`) argument

On help file for many functions, you will see an argument called `...`, referred to as the “dot-dot-dot” argument

```
?sum
?seq
```

“Dot-dot-dot” arguments have several uses. What you should know for now:

- `...` refers to arguments that are “un-named”; but user can specify values
  - e.g., default syntax for `sum()` : `sum(..., na.rm = FALSE)`
    - argument `na.rm` is “named” (name is `na.rm`); argument `...` un-named
- `...` used to allow a function to take an arbitrary number of arguments:

```
sum(c(10,5,NA),na.rm=TRUE)
#> [1] 15

#Here the sum function takes 3 un-named arguments
sum(10,5,NA,na.rm=TRUE)
#> [1] 15

#Here the sum function takes 5 un-named arguments
sum(10,5,10,20,NA,na.rm=TRUE)
#> [1] 45
```

Directories and filepaths

Working directory

(Current) Working directory

- the folder/directory in which you are currently working
- this is where R looks for files
- Files located in your current working directory can be accessed without specifying a filepath because R automatically looks in this folder

Function `getwd()` shows current working directory

```
getwd()
#> [1] "/Users/karinasalazar/rclass/lectures/lecture2"
```

Command `list.files()` lists all files located in working directory

```
getwd()
#> [1] "/Users/karinasalazar/rclass/lectures/lecture2"
list.files()
#>  [1] "fp1.JPG"                        "fp2.JPG"
#>  [3] "lecture2_test.Rmd"              "lecture2_ucla.pdf"
#>  [5] "lecture2_ucla.Rmd"              "lecture2.1_ucla.pdf"
#>  [7] "lecture2.1_ucla.Rmd"            "lecture2.pdf"
#>  [9] "lecture2.Rmd"                   "problemset2_solutions.html"
#> [11] "problemset2_solutions.html.zip" "problemset2_solutions.Rmd"
#> [13] "problemset2.html"               "problemset2.Rmd"
#> [15] "sample_simple_rmarkdown.txt"    "sample.html"
#> [17] "sample.Rmd"                     "text"
#> [19] "transform-logical.png"
```

Working directory, “Code chunks” vs. “console” and “R scripts”

When you run **code chunks** in RMarkdown files (.Rmd), the working directory is set to the filepath where the .Rmd file is stored

```
getwd()
#> [1] "/Users/karinasalazar/rclass/lectures/lecture2"
list.files()
#>  [1] "fp1.JPG"                        "fp2.JPG"
#>  [3] "lecture2_test.Rmd"              "lecture2_ucla.pdf"
#>  [5] "lecture2_ucla.Rmd"              "lecture2.1_ucla.pdf"
#>  [7] "lecture2.1_ucla.Rmd"            "lecture2.pdf"
#>  [9] "lecture2.Rmd"                   "problemset2_solutions.html"
#> [11] "problemset2_solutions.html.zip" "problemset2_solutions.Rmd"
#> [13] "problemset2.html"               "problemset2.Rmd"
#> [15] "sample_simple_rmarkdown.txt"    "sample.html"
#> [17] "sample.Rmd"                     "text"
#> [19] "transform-logical.png"
```

When you run code from the **R Console** or an **R Script**, the working directory is....

Command `getwd()` shows current working directory

```
getwd()
#> [1] "/Users/karinasalazar/rclass/lectures/lecture2"
```

Absolute vs. relative filepath

Absolute file path: The absolute file path is the complete list of directories needed to locate a file or folder.

```
setwd("/Users/pm/Desktop/rclass/lectures/lecture2")
```

Relative file path: The relative file path is the path relative to your current location/directory. Assuming your current working directory is in the "lecture2" folder and you want to change your directory to the data folder, your relative file path would look something like this:

```
setwd("../../data")
```

`File path shortcuts`

Key	Description
~	tilde is a shortcut for user's home directory (mine is my name pm)
../	moves up a level
../../	moves up two level

Exercise

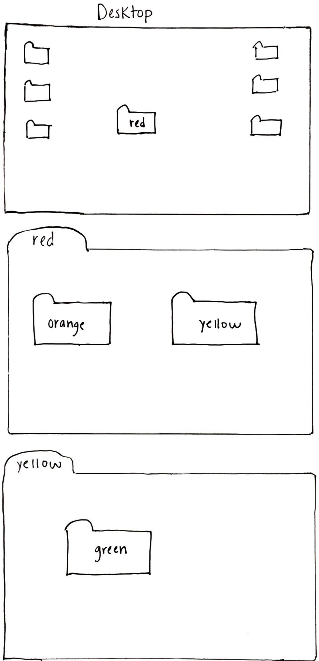
1. Let's create a folder on our desktop and name it red
2. Inside the red folder, create two subfolders named orange and yellow
3. Inside the yellow folder create another subfolder named green

Make sure to name these folders in lowercase.

You should have 1 folder on your desktop called red. Inside the red folder you have two folders called orange and yellow. Inside the yellow folder you have a folder called green.

Here is a visual of how it should look...

File path visual



Exercise continued

Let's say we want to get to the green folder using the absolute file path.

1. View your current working directory `getwd()`
2. Set your working directory to the green folder using the absolute file path
3. Now set your working directory to the orange folder using the relative file path (hint: ../)

Solution

```r
getwd()
setwd("~/Desktop/red/yellow/green")
getwd()
setwd("../../orange")
getwd()
```

Investigating objects

Load .Rdata data frames we will use today

Data on off-campus recruiting events by public universities

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

```r
rm(list = ls()) # remove all objects in current environment

getwd()
#> [1] "/Users/karinasalazar/rclass/lectures/lecture2"
#load dataset with one obs per recruiting event
load(url("https://github.com/ozanj/rclass/raw/master/data/recruiting/recruit_ev
#load("../../data/recruiting/recruit_event_somevars.Rdata")

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

Listing objects

Files in your working directory

`list.files()` function lists files in your current working directory

- if you run this code from .Rmd file, working directory is location .Rmd file is stored

```
getwd() # what is your current working directory
#> [1] "/Users/karinasalazar/rclass/lectures/lecture2"
list.files()
#>  [1] "fp1.JPG"                        "fp2.JPG"
#>  [3] "lecture2_test.Rmd"              "lecture2_ucla.pdf"
#>  [5] "lecture2_ucla.Rmd"              "lecture2.1_ucla.pdf"
#>  [7] "lecture2.1_ucla.Rmd"            "lecture2.pdf"
#>  [9] "lecture2.Rmd"                   "problemset2_solutions.html"
#> [11] "problemset2_solutions.html.zip" "problemset2_solutions.Rmd"
#> [13] "problemset2.html"               "problemset2.Rmd"
#> [15] "sample_simple_rmarkdown.txt"    "sample.html"
#> [17] "sample.Rmd"                     "text"
#> [19] "transform-logical.png"
```

Objects currently open in your R session

`ls()` function lists objects currently open in R

```
x <- "hello!"
ls() # Objects open in R
#> [1] "df_event"  "df_school" "x"
```

Removing objects

`rm()` function removes specified objects open in R

```
rm(x)
ls()
#> [1] "df_event"  "df_school"
```

Command to remove all objects open in R (I don't run it)

```
rm(list = ls())
```

Describing objects, focus on **data frames**

type and **length** of a data frame object

- Recall that a data frame is an object where **type** is a list
- **Length** of an object is the number of elements
  - When object is a data frame, number of elements = number of variables

```
typeof(df_event)
#> [1] "list"
length(df_event) # = num elements = num columns
#> [1] 33
```

Number of **columns** and **rows** of data frame object

- number of columns = number of elements = number of variables
- number of rows = number of observations

```
ncol(df_event) # num columns = num variables
#> [1] 33
nrow(df_event) # num rows = num observations
#> [1] 18680
dim(df_event) # shows number rows by columns
#> [1] 18680    33
```

`str()` provides compact information on structure any object (output omitted)

```
str(df_event)
```

Variables names

Variable names

`names()` function lists names of elements in an object

```
?names
```

When object is a data frame:

- each element is a variable
- each element name is a variable name

```
names(df_event)
#>  [1] "instnm"               "univ_id"              "instst"
#>  [4] "pid"                  "event_date"           "event_type"
#>  [7] "zip"                  "school_id"            "ipeds_id"
#> [10] "event_state"          "event_inst"           "med_inc"
#> [13] "pop_total"            "pct_white_zip"        "pct_black_zip"
#> [16] "pct_asian_zip"        "pct_hispanic_zip"     "pct_amerindian_zip"
#> [19] "pct_nativehawaii_zip" "pct_tworaces_zip"     "pct_otherrace_zip"
#> [22] "fr_lunch"             "titlei_status_pub"    "total_12"
#> [25] "school_type_pri"      "school_type_pub"      "g12offered"
#> [28] "g12"                  "total_students_pub"   "total_students_pri"
#> [31] "event_name"           "event_location_name"  "event_datetime_start"
```

Variable names

Refer to specific named elements of an object using this syntax:

- `obj_name$element_name`

When object is data frame, refer to specific variables using this syntax:

- `data_fram_name$varname`
- **This approach to isolating variables very useful for investigating data**

```r
typeof(df_event$instnm)
#> [1] "character"
typeof(df_event$med_inc)
#> [1] "double"
```

Variable names

Recall that data frames are lists with following criteria:

- each element of the list is a vector
    - each element of list is a variable; length of data frame = number of variables

```
length(df_event)
#> [1] 33
nrow(df_event)
#> [1] 18680
#str(df_event)
```

- each element of the list (i.e., variable) has the same length
    - Length of each variable is equal to number of observations in data frame

```
typeof(df_event$event_state)
#> [1] "character"
length(df_event$event_state)
#> [1] 18680
str(df_event$event_state)
#>  chr [1:18680] "MA" "MA" "MA" "MA" "MA" "MA" "MA" "MA" "MA" "MA" "MA" "MA" ..

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

Variable names

Recall that object `df_school` has one obs per high school

- the variable `visits_by_100751` shows number of visits by University of Alabama to each high school
- like all variables in a data frame, the var `visits_by_100751` is just a vector

```
typeof(df_school$visits_by_100751)
#> [1] "integer"
length(df_school$visits_by_100751) # num elements in vector
#> [1] 21301
sum(df_school$visits_by_100751)
#> [1] 3338
```

Sp we perform calculations on a variable, just like we would any numeric vector

```
v <- c(2,4,6)
typeof(v)
#> [1] "double"
length(v)
#> [1] 3
sum(v)
#> [1] 12
```

View and print data

Viewing and printing data frames

Three ways to view/print a data frame object

1. Simply type the object name (output omitted)
    - number of observations and rows printed depend on YAML header settings and on attributes (discussed next week) of the object

```
df_event
```

2. Use the `View()` function to view data in a browser

```
View(df_event)
```

3. `head()` to show the first *n* rows

```
#?head
head(df_event, n=5)
```

Viewing and printing data frames

`obj_name[<rows>,<cols>]` to print specific rows and columns of data frame

- particularly powerful when combined with sequences (e.g., `1:10`)

Examples:

- Print first five rows

```
df_event[1:5, ]
```

- Print first five rows and first three columns

```
df_event[1:5, 1:3]
```

- Print first three columns of the 100th observation

```
df_event[100, 1:3]
```

- Print the 50th observation, all variables

```
df_event[50,]
```

Viewing and printing data

type `obj_name$var_name` to print specific elements (i.e., vars) in data frame

```
df_event$zip
```

- recall that these elements are vectors, with length = number of obs

```
typeof(df_event$zip)
#> [1] "character"
length(df_event$zip)
#> [1] 18680
```

- `obj_name$var_name` syntax can be combined with sequences
  - vectors don't have "rows" or "columns"; they just have elements
  - so use sequence to identify which elements you want to print

```
df_event$event_state[1:10]
#>  [1] "MA" "MA" "MA" "MA" "MA" "MA" "MA" "MA" "MA" "MA"
df_event$event_type[6:10]
#> [1] "private hs" "private hs" "public hs"  "private hs" "public hs"
```

Can also print multiple variables using `combine()` function

```
c(df_event$event_state[1:5],df_event$event_type[1:5])
#>  [1] "MA"         "MA"         "MA"         "MA"         "MA"         "public hs"
#>  [7] "public hs" "public hs" "public hs" "public hs"
```

Exercise

Create a printing exercise using the df_school data frame

1. Use `obj_name[<rows>,<cols>]` to print the first 5 rows and 3 columns of data frame
2. Use head() to print first 4 observations
3. Use `obj_name$var_name[1:10]` to print the first 10 observations of a variable
4. Use combine() to print the first 3 observations of variables "school_type" & "name"

Solution

1. Use `obj_name[<rows>,<cols>]` to print the first 5 rows and 3 columns of data frame

```
df_school[1:5,1:3]
#> # A tibble: 5 x 3
#>   state_code school_type ncessch
#>   <chr>      <chr>       <chr>
#> 1 AK         public      020000100208
#> 2 AK         public      020000100211
#> 3 AK         public      020000100212
#> 4 AK         public      020000100213
#> 5 AK         public      020000300216
```

Solution

2. Use head() to print first 4 observations

```
head(df_school, n=4)
#> # A tibble: 4 x 26
#>   state_code school_type ncessch name  address city  zip_code pct_white
#>   <chr>      <chr>       <chr>   <chr> <chr>   <chr> <chr>        <dbl>
#> 1 AK         public      020000~ Beth~ 1006 R~ Beth~ 99559         11.8
#> 2 AK         public      020000~ Ayag~ 106 Vi~ Kong~ 99559          0
#> 3 AK         public      020000~ Kwig~ 108 Vi~ Kwig~ 99622          0
#> 4 AK         public      020000~ Nels~ 118 Vi~ Toks~ 99637          0
#> # ... with 18 more variables: pct_black <dbl>, pct_hispanic <dbl>,
#> #   pct_asian <dbl>, pct_amerindian <dbl>, pct_other <dbl>, num_fr_lunch <dbl
#> #   total_students <dbl>, num_took_math <dbl>, num_prof_math <dbl>,
#> #   num_took_rla <dbl>, num_prof_rla <dbl>, avgmedian_inc_2564 <dbl>,
#> #   visits_by_110635 <int>, visits_by_126614 <int>, visits_by_100751 <int>,
#> #   inst_110635 <chr>, inst_126614 <chr>, inst_100751 <chr>
```

Solution

3. Use `obj_name$var_name[1:10]` to print the first 10 observations of a variable

```r
df_school$name[1:10]
#>  [1] "Bethel Regional High School" "Ayagina'ar Elitnaurvik"
#>  [3] "Kwigillingok School"         "Nelson Island Area School"
#>  [5] "Alakanuk School"             "Emmonak School"
#>  [7] "Hooper Bay School"           "Ignatius Beans School"
#>  [9] "Pilot Station School"        "Kotlik School"
```

Solution

4. Use combine() to print the first 3 observations of variables “school_type” & “name”

```
c(df_school$school_type[1:3],df_school$name[1:3])
#> [1] "public"                      "public"
#> [3] "public"                      "Bethel Regional High School"
#> [5] "Ayagina'ar Elitnaurvik"      "Kwigillingok School"
```

Missing values

Missing values

Missing values have the value `NA`

- `NA` is a special keyword, not the same as the character string `"NA"`

use `is.na()` function to determine if a value is missing

- `is.na()` returns a logical vector

```r
is.na(5)
#> [1] FALSE
is.na(NA)
#> [1] TRUE
is.na("NA")
#> [1] FALSE
typeof(is.na("NA")) # example of a logical vector
#> [1] "logical"

nvector <- c(10,5,NA)
is.na(nvector)
#> [1] FALSE FALSE  TRUE
typeof(is.na(nvector)) # example of a logical vector
#> [1] "logical"

svector <- c("e","f",NA,"NA")
is.na(svector)
#> [1] FALSE FALSE  TRUE FALSE
```

Missing values are “contagious”

What does “contagious” mean?

- operations involving a missing value will yield a missing value

```
7>5
#> [1] TRUE
7>NA
#> [1] NA
0==NA
#> [1] NA
2*c(0,1,2,NA)
#> [1]  0  2  4 NA
NA*c(0,1,2,NA)
#> [1] NA NA NA NA
```

Function and missing values, the `table()` function

`table()` function useful for investigating categorical variables

```
table(df_event$g12offered)
#> 
#>     1 
#> 11423
```

By default `table()` ignores `NA` values

- `useNA` argument determines whether to include `NA` values
  - “allowed values correspond to never (”no“); only if count is positive (”ifany“); and even for zero counts (”always“)”

```
nrow(df_event)
#> [1] 18680
table(df_event$g12offered, useNA="always")
#> 
#>     1  <NA> 
#> 11423  7257
```

Broader point:

- Most functions that create descriptive statistics have options about how to treat missing values
- When investigating data, good practice to *always* show missing values

Tip:

- command `str(df_event)` shows which variables have missing values

Investigating data patterns with tidyverse

Introduction to the `dplyr` library

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

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

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

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

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

Select variables

Select variables using `select()` function

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

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

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

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

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

Select variables using `select()` function

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

- **type** equals “list”
- **length** equals number of vars you select

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

length(select(df_event,instnm,event_date,event_type,event_state,med_inc))
#> [1] 5
```

`glimpse()` function – a tidyverse function for viewing data frames – is a cross between `str()` and simply printing data

```
#?glimpse
glimpse(select(df_event,instnm,event_date,event_type,event_state,med_inc))
#> Rows: 18,680
#> Columns: 5
#> $ instnm      <chr> "UM Amherst", "UM Amherst", "UM Amherst", "UM Amherst",..
#> $ event_date  <date> 2017-10-12, 2017-10-04, 2017-10-25, 2017-10-26, 2017-1..
#> $ event_type  <chr> "public hs", "public hs", "public hs", "public hs", "pu..
#> $ event_state <chr> "MA", "MA", "MA", "MA", "MA", "MA", "MA", "MA", "MA", "..
#> $ med_inc     <dbl> 71713.5, 89121.5, 70136.5, 70136.5, 71023.5, 71023.5, 7..
```

Select variables using `select()` function

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

```
event_small <- select(df_event,instnm,event_date,event_type,event_state,med_inc
glimpse(event_small)
#> Rows: 18,680
#> Columns: 5
#> $ instnm      <chr> "UM Amherst", "UM Amherst", "UM Amherst", "UM Amherst",..
#> $ event_date  <date> 2017-10-12, 2017-10-04, 2017-10-25, 2017-10-26, 2017-1..
#> $ event_type  <chr> "public hs", "public hs", "public hs", "public hs", "pu..
#> $ event_state <chr> "MA", "MA", "MA", "MA", "MA", "MA", "MA", "MA", "MA", "..
#> $ med_inc     <dbl> 71713.5, 89121.5, 70136.5, 70136.5, 71023.5, 71023.5, 7..
```

Select

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

Example:

```
#names(df_event)
select(df_event,instnm,starts_with("event"))
#> # A tibble: 18,680 x 8
#>    instnm event_date event_type event_state event_inst event_name
#>    <chr>  <date>     <chr>      <chr>       <chr>      <chr>
#>  1 UM Am~ 2017-10-12 public hs  MA          In-State   Amherst-P~
#>  2 UM Am~ 2017-10-04 public hs  MA          In-State   Hampshire~
#>  3 UM Am~ 2017-10-25 public hs  MA          In-State   Chicopee ~
#>  4 UM Am~ 2017-10-26 public hs  MA          In-State   Chicopee ~
#>  5 Stony~ 2017-10-02 public hs  MA          Out-State  Easthampt~
#>  6 USCC   2017-09-18 private hs MA          Out-State  Williston~
#>  7 UM Am~ 2017-09-18 private hs MA          In-State   Williston~
#>  8 UM Am~ 2017-09-26 public hs  MA          In-State   Granby Jr~
#>  9 UM Am~ 2017-09-26 private hs MA          In-State   MacDuffie~
#> 10 UM Am~ 2017-10-12 public hs  MA          In-State   Smith Aca~
#> # ... with 18,670 more rows, and 2 more variables: event_location_name <chr>,
#> #   event_datetime_start <dttm>
```

Exercise

The data frame `df_school` has one observation for each high school and indicators for whether the high school received a recruiting visit.

```
names(df_school)
```

1. Use `select()` to familiarize yourself with variables in the data frame
2. Practice using the `contains()` and `ends_with()` helper functions to to choose variables

Rename variables

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

Syntax:

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

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

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

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

Filter rows

The `filter()` function

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

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

Example using data frame object `df_school`, where each observation is a high school

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

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

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

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

Exercise

Task

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

Solution

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

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

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

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

Filter, character variables

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

Below, we identify all private high schools in CA that got visit by particular universities

```r
glimpse(df_school)
#Berkeley
filter(df_school,visits_by_110635 == 1, school_type == "private",
       state_code == "CA")
#Bama
filter(df_school,visits_by_100751 == 1, school_type == "private",
       state_code == "CA")

#Berkeley and Bama
filter(df_school,visits_by_100751 == 1, visits_by_110635 == 1,
       school_type == "private", state_code == "CA")
```

Logical operators for comparisons

Symbol	Meaning
`==`	Equal to
`!=`	Not equal to
`>`	greater than
`>=`	greater than or equal to
`<`	less than
`<=`	less than or equal to
`&`	AND
`|`	OR
`%in`	includes

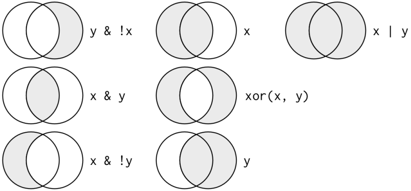


Figure 1: "Boolean" operations, x=left circle, y=right circle, from Wichkam (2018)

Filters and comparisons, Demonstration

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

```
#berkeley and bama
filter(df_school,visits_by_100751 >= 1, visits_by_110635 >= 1)
filter(df_school,visits_by_100751 >= 1 & visits_by_110635 >= 1) # same same
#berkeley or bama
filter(df_school,visits_by_100751 >= 1 | visits_by_110635 >= 1)
```

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

- Avoids printing individual observations

```
#Number of schools that git visit by Berkeley AND Bama
count(filter(df_school,visits_by_100751 >= 1 & visits_by_110635 >= 1))
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1   247
#Number of schools that git visit by Berkeley OR Bama
count(filter(df_school,visits_by_100751 >= 1 | visits_by_110635 >= 1))
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1  2763
```

Filters and comparisons, >=

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

```
#at least 50% black
count(filter(df_school, school_type == "public", pct_black >= 50,
             state_code == "AL"))
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1    86
count(filter(df_school, school_type == "public", pct_black >= 50,
             state_code == "AL", visits_by_100751 >= 1))
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1    21
#at least 50% white
count(filter(df_school, school_type == "public", pct_white >= 50,
             state_code == "AL"))
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1   238
count(filter(df_school, school_type == "public", pct_white >= 50,
             state_code == "AL", visits_by_100751 >= 1))
#> # A tibble: 1 x 1
#>       n
```

Filters and comparisons, not equals (!=)

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

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

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

Filters and comparisons, %in% operator

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

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

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

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

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

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

Identifying data type and possible values of variable is helpful for filtering

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

```
class(df_event$event_type)
#> [1] "character"
str(df_event$event_type)
#>  chr [1:18680] "public hs" "public hs" "public hs" "public hs" "public hs" ..
table(df_event$event_type)
#> 
#> 2yr college 4yr college       other  private hs   public hs 
#>         951         531        2001        3774       11423

class(df_event$event_state)
#> [1] "character"
str(df_event$event_state) # double quotes indicate character
#>  chr [1:18680] "MA" "MA" "MA" "MA" "MA" "MA" "MA" "MA" "MA" "MA" "MA" "MA" ..

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

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

```
count(filter(df_event, event_type == "public hs", event_state =="CA"))
#> # A tibble: 1 x 1
#>       n
#>   <int>
```

Exercises

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

1. Count the number of events attended by the University of Pittsburgh (Pitt) `univ_id == 215293`
2. Count the number of recruiting events by Pitt at public or private high schools
3. Count the number of recruiting events by Pitt at public or private high schools located in the state of PA
4. Count the number of recruiting events by Pitt at public high schools not located in PA where median income is less than 100,000
5. Count the number of recruiting events by Pitt at public high schools not located in PA where median income is greater than or equal to 100,000
6. Count the number of out-of-state recruiting events by Pitt at private high schools or public high schools with median income of at least 100,000

Solution

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

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

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

```
count(filter(df_event, univ_id == 215293, event_type == "private hs" |
                event_type == "public hs"))
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1  1030
```

Solution

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

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

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

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

Solution

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

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

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

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

Filtering and missing values

Wickham (2018) states:

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

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

- only available for visits to public high schools

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

Arrange rows

`arrange()` function

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

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

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

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

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

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

`arrange()` function

Ascending and descending order

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

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

Can sort by multiple variables

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

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

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

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

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

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

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

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

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

Exercise, arranging

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

1. Sort ascending by “univ_id” and descending by “event_date”
2. Select four variables in total and sort ascending by “univ_id” and descending by “event_date”
3. Now using the same variables from above, sort by `is.na` to put missing values in “school_id” first

Solution

1. Sort ascending by “univ_id” and descending by “event_date”

```
arrange(df_event, univ_id, desc(event_date))
#> # A tibble: 18,680 x 33
#>    instnm univ_id instst   pid event_date event_type zip   school_id ipeds_id
#>    <chr>    <int> <chr>  <int> <date>     <chr>      <chr> <chr>        <int>
#>  1 Bama    100751 AL      7115 2017-12-18 private hs 77089 A9106483        NA
#>  2 Bama    100751 AL      7121 2017-12-18 other      <NA>  <NA>            NA
#>  3 Bama    100751 AL      7114 2017-12-15 public hs  75165 48447300~       NA
#>  4 Bama    100751 AL      7100 2017-12-15 public hs  93012 06292700~       NA
#>  5 Bama    100751 AL      7073 2017-12-15 other      98027 <NA>            NA
#>  6 Bama    100751 AL      7072 2017-12-14 other      98007 <NA>            NA
#>  7 Bama    100751 AL      7118 2017-12-13 public hs  31906 13038700~       NA
#>  8 Bama    100751 AL      7099 2017-12-13 private hs 90293 00071151        NA
#>  9 Bama    100751 AL      7109 2017-12-13 public hs  92630 06338600~       NA
#> 10 Bama    100751 AL      7071 2017-12-13 other      98032 <NA>            NA
#> # ... with 18,670 more rows, and 24 more variables: event_state <chr>,
#> #   event_inst <chr>, med_inc <dbl>, pop_total <dbl>, pct_white_zip <dbl>,
#> #   pct_black_zip <dbl>, pct_asian_zip <dbl>, pct_hispanic_zip <dbl>,
#> #   pct_amerindian_zip <dbl>, pct_nativehawaii_zip <dbl>,
#> #   pct_tworaces_zip <dbl>, pct_otherrace_zip <dbl>, fr_lunch <dbl>,
#> #   titlei_status_pub <fct>, total_12 <dbl>, school_type_pri <int>,
#> #   school_type_pub <int>, g12offered <dbl>, g12 <dbl>,
#> #   total_students_pub <dbl>, total_students_pri <dbl>, event_name <chr>,
#> #   event_location_name <chr>, event_datetime_start <dttm>
```

Solution

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

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

Solution

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

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