Lecture 2: Investigating data patterns

EDUC 263: Managing and Manipulating Data Using R

Ozan Jaquette

1. Directories and filepaths [?lec1 or lec2?]

2. Investigating objects

3. Variables names

4. Selecting variables and printing data

5. Filtering data

6. Missing values

7. Arrange rows

Libraries we will use today

```
library(tidyverse)
#> -- Attaching packages ---------------------------------------------
#> v ggplot2 3.0.0     v purrr   0.2.5
#> v tibble  1.4.2     v dplyr   0.7.6
#> v tidyr   0.8.1     v stringr 1.3.1
#> v readr   1.1.1     v forcats 0.3.0
#> -- Conflicts ------------------------------------------------------
#> x dplyr::filter() masks stats::filter()
#> x dplyr::lag()    masks stats::lag()
```

Directories and filepaths [?lec1 or lec2?]

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] "C:/Users/ozanj/Documents/rclass/lectures/lecture2"
```

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

```
getwd()
#> [1] "C:/Users/ozanj/Documents/rclass/lectures/lecture2"
list.files()
#> [1] "lecture2.pdf"          "lecture2.Rmd"          "lecture2.tex"
#> [4] "text"                  "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] "C:/Users/ozanj/Documents/rclass/lectures/lecture2"
list.files()
#> [1] "lecture2.pdf"          "lecture2.Rmd"          "lecture2.tex"
#> [4] "text"                  "transform-logical.png"
```

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

I find this very annoying

Command `getwd()` shows current working directory

```
getwd()
#> [1] "C:/Users/ozanj/Documents/rclass/lectures/lecture2"
```

Absolute vs. relative filepath

Investigating objects

Data on off-campus recruiting events by public universities

```
rm(list = ls()) # remove all objects
getwd()
#> [1] "C:/Users/ozanj/Documents/rclass/lectures/lecture2"

#load dataset with one obs per recruiting event
load("../../data/recruiting/recruit_event_somevars.Rdata")

#load dataset with one obs per high school
load("../../data/recruiting/recruit_school_somevars.Rdata")
```

Object \hlgc{df_event}

- One observation per university, recruiting event

Object \hlgc{df_event}

- One observation per high school (visited and non-visited)

Listing objects

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

- This is different from files in current working directory

```
x <- "hello!"
ls() # Objects open in R
#> [1] "df_event"  "df_school" "x"
list.files() # files in working directory
#> [1] "lecture2.pdf"           "lecture2.Rmd"           "lecture2.tex"
#> [4] "text"                   "transform-logical.png"
```

`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**

Object **type** and **length**

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

Number of **rows** and **columns**

```
nrow(df_event) # num rows = num observations
#> [1] 17976
ncol(df_event) # num columns = num variables
#> [1] 33
dim(df_event) # shows number rows by columns
#> [1] 17976    33
```

Variables names

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

Variable names

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

- when object is a data frame, each element name is a variable name

```
#Output omitted
names(df_event)
```

Refer to named elements in object using `obj_name$element_name`

- when object is a dataframe: `obj_name$varname`

```
typeof(df_event$instnm)
#> [1] "character"
typeof(df_event$avgmedian_inc_2564)
#> Warning: Unknown or uninitialised column: 'avgmedian_inc_2564'.
#> [1] "NULL"
```

This approach to isolating variables is very useful for investigating and manipulating data

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_statu
names(rename_event)
rm(rename_event)
```

Selecting variables and printing data

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

- See syntax in help file

Select **without assignment** simply prints data for selected variables

```
select(df_event,instnm,event_date,event_type,event_state,med_inc)
#> # A tibble: 17,976 x 5
#>    instnm      event_date event_type event_state med_inc
#>    <chr>       <date>     <fct>      <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  2017-10-26 public hs  MA           70136.
#>  4 UM Amherst  2017-10-25 public hs  MA           70136.
#>  5 USCC        2017-09-18 private hs MA           71024.
#>  6 UM Amherst  2017-09-18 private hs MA           71024.
#>  7 Stony Brook 2017-10-02 public hs  MA           71024.
#>  8 UM Amherst  2017-09-26 private hs MA           97225
#>  9 UM Amherst  2017-09-26 public hs  MA           97225
#> 10 UM Amherst  2017-10-12 public hs  MA           77800.
#> # ... with 17,966 more rows
```

Select variables using `select()` function

Select **with assignment** creates a new object containing only the variables you specify

```
event_small <- select(df_event,instnm,event_date,event_type,event_state,med_inc)
event_small
#> # A tibble: 17,976 x 5
#>    instnm      event_date event_type event_state med_inc
#>    <chr>       <date>     <fct>      <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  2017-10-26 public hs  MA           70136.
#>  4 UM Amherst  2017-10-25 public hs  MA           70136.
#>  5 USCC        2017-09-18 private hs MA           71024.
#>  6 UM Amherst  2017-09-18 private hs MA           71024.
#>  7 Stony Brook 2017-10-02 public hs  MA           71024.
#>  8 UM Amherst  2017-09-26 private hs MA           97225
#>  9 UM Amherst  2017-09-26 public hs  MA           97225
#> 10 UM Amherst  2017-10-12 public hs  MA           77800.
#> # ... with 17,966 more rows
```

Select

We can also use "helper functions" `starts_with()`, `contains()`, and `ends_with()` to choose columns

```
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"
select(df_event,instnm,starts_with("event"))
#> # A tibble: 17,976 x 8
#>    instnm event_date event_type event_state event_inst event_name
#>    <chr>  <date>     <fct>      <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~ 2017-10-26 public hs  MA          In-State   Chicopee ~
#>  4 UM Am~ 2017-10-25 public hs  MA          In-State   Chicopee ~
#>  5 USCC   2017-09-18 private hs MA          Out-State  Williston~
#>  6 UM Am~ 2017-09-18 private hs MA          In-State   Williston~
#>  7 Stony~ 2017-10-02 public hs  MA          Out-State  Easthampt~
#>  8 UM Am~ 2017-09-26 private hs MA          In-State   MacDuffie~
```

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 the data frame
2. Practice using the `contains()` and `ends_with()` helper functions to to choose variables

Viewing and printing data

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

```
View(df_event)
```

- `head()` to show the first *n* rows

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

Viewing and printing data

Use `{obj_name[<rows>,<cols>]` to print specific rows and columns of a 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 obs for a variable

```
df_event$event_state
```

- can be combined with sequences

```
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 public hs  private hs public hs  public hs
#> Levels: public hs private hs 2yr college 4yr college other
```

- 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" "1"  "1"  "1"  "1"  "2"
```

Exercise

Create a printing exercise using the df_school

1. Use head() to print first 5 observations
2. Use select to print observations of variables of your choosing
3. Use `obj_name$var_name[1:10]` to print the first 10 observations of a variable
4. use combine()

Filtering data

Filter

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

- Arguments
  - first argument is name of data frame
  - subsequent arguments are *expressions* to filter the data frame
- What is the result of a `filter()` command?
  - `filter()` returns the 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]

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

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

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

Filter, character variables

- Use quotes `'` or `""` to refer to character variables

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

#Berkeley and Bama
filter(df_school,visits_by_100751 == 1, visits_by_110635 == 1, school_type == "p
```

Logical operators for comparisons

Table 2: Logical operators

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

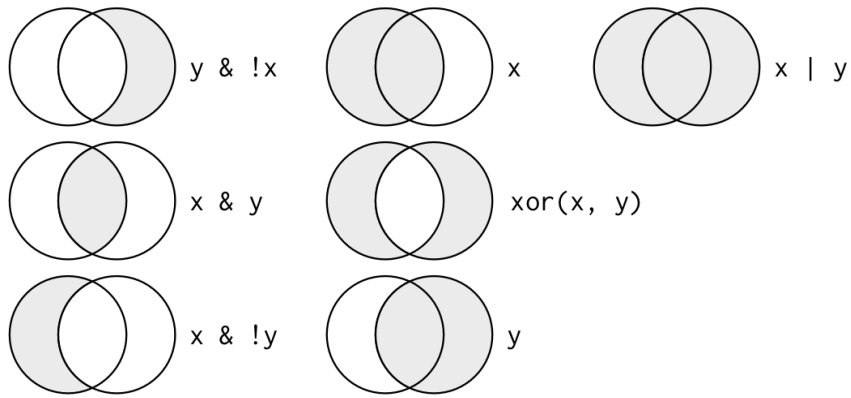

Figure 1: “Boolean” operations, x=left circle, y=right circle, from Wichkam (2018)

Filters and comparisons

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

```
count(filter(df_school,visits_by_100751 >= 1 & visits_by_110635 >= 1))
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1   247
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 ==
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1    86
count(filter(df_school, school_type == "public", pct_black >= 50, state_code ==
#> # 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 ==
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1   238
count(filter(df_school, school_type == "public", pct_white >= 50, state_code ==
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1    82
```

Filters and comparisons, not equals (!=)

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 ==
#> # 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$event_type)
#> [1] "factor"
str(df_event$event_type)
#>  Factor w/ 5 levels "public hs","private hs",..: 1 1 1 1 2 2 1 2 1 1 ...
table(df_event$event_type)
#>
#>   public hs  private hs 2yr college 4yr college       other
#>       11025        3644         769         431        2107

class(df_event$event_state)
#> [1] "character"
str(df_event$event_state) # double quotes indicate character
#>  chr [1:17976] "MA" "MA" "MA" "MA" "MA" "MA" "MA" "MA" "MA" "MA" "MA" ...

class(df_event$med_inc)
#> [1] "numeric"
str(df_event$med_inc)
#>  num [1:17976] 71714 89122 70137 70137 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
```

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

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(5)
#> [1] FALSE
is.na(NA)
#> [1] TRUE
is.na("NA")
#> [1] FALSE

nvector <- c(10,5,NA)
nvector
#> [1] 10  5 NA
is.na(nvector)
#> [1] FALSE FALSE  TRUE

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

Missing values are “contageous”

What does “contageous” 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

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

`table()` function useful for investigating categorical variables

- by default `table()` ignores `NA` values
- the `useNA` argument to include `NA` values
  - from help file: "useNA controls if the table includes counts of NA values: the allowed values correspond to never ("no"), only if the count is positive ("ifany") and even for zero counts ("always")"

```
table(df_event$g12offered)
#>
#>     1
#> 11025
nrow(df_event)
#> [1] 17976
table(df_event$g12offered, useNA="always")
#>
#>     1  <NA>
#> 11025  6951
```

Broader point:

- Most functions in R ignor missing values by default, but have the option to include missing values
- When investigating data, generally a good idea to always include missing values

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   890
#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   916
```

Exercises, missing values

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

```
#by university, date, ascending school id
select(arrange(df_event, univ_id, desc(event_date), school_id),instnm,event_date
#> # A tibble: 17,976 x 4
#>    instnm event_date event_type school_id
#>    <chr>  <date>     <fct>      <chr>
#>  1 Bama   2017-12-18 private hs A9106483
#>  2 Bama   2017-12-18 other      <NA>
#>  3 Bama   2017-12-15 public hs  062927004516
#>  4 Bama   2017-12-15 public hs  484473005095
#>  5 Bama   2017-12-15 other      <NA>
#>  6 Bama   2017-12-14 other      <NA>
#>  7 Bama   2017-12-13 private hs 00071151
#>  8 Bama   2017-12-13 public hs  063386005296
#>  9 Bama   2017-12-13 public hs  130387001439
#> 10 Bama   2017-12-13 other      <NA>
#> # ... with 17,966 more rows

#by university, date, descending school id
select(arrange(df_event, univ_id, desc(event_date), desc(school_id)),instnm,even
#> # A tibble: 17,976 x 4
#>    instnm event_date event_type school_id
#>    <chr>  <date>     <fct>      <chr>
#>  1 Bama   2017-12-18 private hs A9106483
#>  2 Bama   2017-12-18 other      <NA>
```

Exercises, arranging