Lecture 3: Investigating data patterns using Base R

Managing and Manipulating Data Using R

1 Introduction

What we will do today

Load libraries and .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)

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

library(tidyverse) #load tidyverse library
```

```
## -- Attaching packages ---------------------------------------------

## v ggplot2 3.2.1     v purrr   0.3.2
## v tibble  2.1.3     v dplyr   0.8.3
## v tidyr   0.8.3     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0

## -- Conflicts ------------------------------------------------------
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
#load dataset with one obs per recruiting event
load(url("https://github.com/ozanj/rclass/raw/master/data/recruiting/recruit_eve

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

2 [Finish] Investigating data patterns with tidyverse

2.1 Select variables

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)
```

```
## Observations: 18,680
## Variables: 5
## $ instnm      <chr> "UM Amherst", "UM Amherst", "UM Amherst", "UM Amhe...
## $ event_date  <date> 2017-10-12, 2017-10-04, 2017-10-25, 2017-10-26, 2...
## $ event_type  <chr> "public hs", "public hs", "public hs", "public hs"...
## $ event_state <chr> "MA", "MA", "MA", "MA", "MA", "MA", "MA", "MA", "M...
## $ med_inc     <dbl> 71713.5, 89121.5, 70136.5, 70136.5, 71023.5, 71023...
```

`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 ~
```

2.2 Filter rows

The `filter()` function

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

What is the result of a `filter()` command? - `filter()` returns a data frame consisting of rows where the condition is `TRUE`

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

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

```
## # A tibble: 528 x 26
##    state_code school_type ncessch name  address city  zip_code pct_white
##    <chr>      <chr>       <chr>   <chr> <chr>   <chr> <chr>        <dbl>
##  1 AZ         public      040081~ Grea~ 39808 ~ Anth~ 85086         81.0
##  2 AZ         public      040187~ Chan~ 350 N.~ Chan~ 85225         36.0
##  3 AZ         public      040834~ Dese~ 16440 ~ Phoe~ 85048         63.2
##  4 AZ         private     000312~ XAVI~ 4710 N~ PHOE~ 85012         83.3
##  5 AZ         private     A97001~ GILB~ 3632 E~ GILB~ 85296         85.5
##  6 AZ         public      040082~ BASI~ 25950 ~ Peor~ 85383         46.5
##  7 AZ         public      040834~ Coro~ 1001 E~ Tempe 85284         59.0
##  8 AZ         private     000321~ PHOE~ 3901 E~ PARA~ 85253         70.2
##  9 CA         public      062724~ Coro~ 2101 E~ Newp~ 92660         82.6
## 10 CA         public      063386~ Trab~ 27501 ~ Miss~ 92691         57.2
## # ... with 518 more rows, and 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>,
```

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
```

2.3 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>
```

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
##    <chr>    <int> <chr>  <int> <date>     <chr>      <chr> <chr>
##  1 Bama    100751 AL      7115 2017-12-18 private hs 77089 A9106483
##  2 Bama    100751 AL      7121 2017-12-18 other      <NA>  <NA>
##  3 Bama    100751 AL      7114 2017-12-15 public hs  75165 48447300~
##  4 Bama    100751 AL      7100 2017-12-15 public hs  93012 06292700~
##  5 Bama    100751 AL      7073 2017-12-15 other      98027 <NA>
##  6 Bama    100751 AL      7072 2017-12-14 other      98007 <NA>
##  7 Bama    100751 AL      7118 2017-12-13 public hs  31906 13038700~
##  8 Bama    100751 AL      7099 2017-12-13 private hs 90293 00071151
##  9 Bama    100751 AL      7109 2017-12-13 public hs  92630 06338600~
## 10 Bama    100751 AL      7071 2017-12-13 other      98032 <NA>
## # ... with 18,670 more rows, and 25 more variables: ipeds_id <int>,
## #   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
```

3 Investigating data patterns using Base R

Why learn to "wrangle" data both via tidyverse and base R?

Tidyverse has become the leading way many people clean and manipulate data in R

- these packages make data wrangling simpler than core base R commands (most times)
- tidyverse commands can be more more efficient (less lines of code, consolidate steps)

But you will inevitably run into edge cases where tidyverse commands don't work the way you expect them to and you'll need to use **base R**

It's good to have a basic foundation on both approaches and then decide which you prefer for most data tasks!

- this class will primarily use tidyverse approach
- future data science seminar will provide examples of edge cases where base R is necessary

Tidyverse vs. base R functions

tidyverse	base R	operation
`select()`	`[ ]` + `c()` **OR** `subset()`	"extract" variables
`filter()`	`[ ]` + `$` **OR** `subset()`	"extract" observations
`arrange()`	`order()`	sorting data

3.1 Subsetting using subsetting operators

Subsetting to Extract Elements

Subsetting is the R word for accessing object elements.

Subsetting features can be used to select/exclude elements (i.e., variables and observations)

- there are three subsetting operators: `[]` , `$` , `[[]]`
- these operators function differently based on vector types (e.g, atomic vectors, lists, data frames)

Subsetting Atomic Vectors via operators

Six ways to subset an atomic vector using `[]`

1. Using positive integers to return elements at specified positions

```
x <- c(1.1, 2.2, 3.3, 4.4, 5.5)
x[c(3, 1)]
```

```
## [1] 3.3 1.1
```

2. Using negative integers to exclude elements at specified positions

```
x[-c(3,1)]
```

```
## [1] 2.2 4.4 5.5
```

3. Using logicals to return elements where corresponding logical is `TRUE`

```
x[x>3] #3
```

```
## [1] 3.3 4.4 5.5
```

Subsetting Atomic Vectors via operators

Six ways to subset an atomic vector using `[]` continued...

4. Empty `[]` returns original vector (useful for dataframes)

```
x[] #4
```

```
## [1] 1.1 2.2 3.3 4.4 5.5
```

5. Zero vector (useful for testing data)

```
x[0]
```

```
## numeric(0)
```

6. Returning character elements with matching names

```
y<- setNames(x, letters[1:5]) #6
y[c("a", "b", "d" )] #6
```

```
##   a   b   d
## 1.1 2.2 4.4
```

Subsetting Lists and Matrices via operators

Subsetting lists (arrays and matrices too) via `[]` operator works the same as subsetting an atomic vector

`[]` simplifies output to the lowest possible dimensionality (i.e.,if you subset a (2D) matrix it will return a 1D vector with however many elements you subset)

```
x <- list(1,2,"apple")
y <- x[c(3, 1)]
typeof(y)
```

```
## [1] "list"
```

```
a <- matrix(1:9, nrow = 3)
a #this is a 3X3 matrix
```

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

```
b <- a[c(1,5 )]
b #returns an integer vector with two elements
```

```
## [1] 1 5
```

Subsetting Single Elements from Vectors, Lists, and Matrices via operators

Two other subsetting operators are used for extracting single elements, since subsetting lists with `[]` returns a smaller list

`[[]]`, `$`

`$` is shorthand operator equivalent to `x[["y"]]` and is used to access variables in a dataframe (will show this in upcoming slides)

Example from Hadley: If x is a train carrying objects, then `x[[5]]` is the object in car 5 and `x[4:6]` is a smaller train made up of cars 4, 5, & 6.

```
x <- list(1:3, "a", 4:6)

y <- x[1] #this returns a list
typeof(y)
```

```
## [1] "list"
```

```
z <- x[[1]] #this is not a list
typeof(z)
```

```
## [1] "integer"
```

Subsetting Data Frames to extract columns (variables) based on positionality

Selecting columns from a data frame by subsetting with `[]` and a single index based on column positionality

```
df_event[1:4]
```

```
## # A tibble: 18,680 x 4
##    instnm      univ_id instst   pid
##    <chr>         <int> <chr>  <int>
##  1 UNL          181464 NE     11052
##  2 Rutgers      186380 NJ     64786
##  3 Rutgers      186380 NJ     64727
##  4 Stony Brook  196097 NY     16005
##  5 Bama         100751 AL      2667
##  6 UGA          139959 GA     21008
##  7 Kansas       155317 KS     59772
##  8 Bama         100751 AL      2674
##  9 Bama         100751 AL      2675
## 10 Kansas       155317 KS     59853
## # ... with 18,670 more rows
```

Subsetting Data Frames to extract columns (variables) and rows (observations) based on positionality

Selecting rows and columns from a data frame by subsetting with `[]` and a double index based on row/column positionality

```
#this returns the first 5 rows and first 3 columns
df_event[1:5, 1:3]
```

```
## # A tibble: 5 x 3
##   instnm      univ_id instst
##   <chr>         <int> <chr>
## 1 UNL          181464 NE
## 2 Rutgers      186380 NJ
## 3 Rutgers      186380 NJ
## 4 Stony Brook  196097 NY
## 5 Bama         100751 AL
```

```
#this returns the first 5 rows and all columns [output omitted]
df_event[1:5, ]
```

Subsetting Data Frames to extract columns (variables) based on names

Selecting columns from a data frame by subsetting with `[]` and list of column names

```
df_event[c("instnm", "univ_id", "event_state")]
```

```
## # A tibble: 18,680 x 3
##    instnm      univ_id event_state
##    <chr>         <int> <chr>
##  1 UNL          181464 TX
##  2 Rutgers      186380 NJã
##  3 Rutgers      186380 NJã
##  4 Stony Brook  196097 NY
##  5 Bama         100751 TX
##  6 UGA          139959 CT
##  7 Kansas       155317 KS
##  8 Bama         100751 AL
##  9 Bama         100751 AL
## 10 Kansas       155317 TX
## # ... with 18,670 more rows
```

Subsetting Data Frames with [] and $

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

```
x <- df_school[df_school$visits_by_110635 == 1, ]
```

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

```
df_school[df_school$visits_by_110635 == 1, 1:3]
```

Show all obs where high schools received 1 visit by Bama (100751) and Berkeley (110635)

```
df_school[df_school$visits_by_110635 == 1 & df_school$visits_by_100751 == 1, ]
```

Subsetting Data Frames with [] and $

Show all public high schools with at least 50% Latinx (hispanic in data) student enrollment

```
#public high schools with at least 50% Latinx student enrollment
df_CA<- df_school[df_school$school_type == "public"
                  & df_school$pct_hispanic >= 50
                  & df_school$state_code == "CA", ]

head(df_CA, n=3)
```

```
## # A tibble: 3 x 26
##   state_code school_type ncessch name  address city  zip_code pct_white
##   <chr>      <chr>       <chr>   <chr> <chr>   <chr> <chr>        <dbl>
## 1 CA         public      064015~ Tust~ 1171 E~ Tust~ 92780        13.3
## 2 CA         public      062547~ Bell~ 6119 A~ Bell~ 90201         0.402
## 3 CA         public      063531~ Sant~ 520 W.~ Sant~ 92701         0.547
## # ... 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>
```

```
nrow(df_CA)
```

```
## [1] 713
```

3.2 Subsetting using the subset function

Subset function

The `subset()` is a base R function and easiest way to "filter" observations

- can be combined with `select()` base R function to select variables
- can be combined with `count()` for quick comparisons or assignment to create new objects

```
?subset
```

Syntax: **subset(x, subset, select, drop = FALSE)**

- x is object to be subsetted
- subset is the logical expression(s) indicating elements (rows) to keep
- select indicates columns to select from data frame (if argument is not used default will keep all columns)
- drop takes `TRUE` or `FALSE` if you want to preserve the original dimensions (only need to worry about dataframes when your subset output is a single column)

Subset function, examples

Show all public high schools that are at least 50% Latinx (hispanic in data) student enrollment in California compared to number of schools that received visit by UC Berkeley

```
#public high schools with at least 50% Latinx student enrollment
count(subset(df_school, school_type == "public" & pct_hispanic >= 50
             & state_code == "CA"))
```

```
## # A tibble: 1 x 1
##       n
##   <int>
## 1   713
```

```
count(subset(df_school, school_type == "public" & pct_hispanic >= 50
             & state_code == "CA" & visits_by_110635 >= 1))
```

```
## # A tibble: 1 x 1
##       n
##   <int>
## 1   100
```

Can also use the %in% operator... -Show visits by Bama in multiple states

```
count(subset(df_school, visits_by_100751 >= 1 & state_code %in% c("MA","ME","VT"
```

```
## # A tibble: 1 x 1
##       n
##   <int>
## 1   108
```

Subset function, examples

Create new df with all public high schools that are at least 50% Latinx student enrollment in California **AND** only keep variables `name` and `address`

```
#public high schools with at least 50% Latinx student enrollment
df_CA2 <- subset(df_school, school_type == "public" & pct_hispanic >= 50
              & state_code == "CA", select = c(name, address))
head(df_CA2)
```

```
## # A tibble: 6 x 2
##   name                  address
##   <chr>                 <chr>
## 1 Tustin High           1171 El Camino Real
## 2 Bell Gardens High     6119 Agra St.
## 3 Santa Ana High        520 W. Walnut
## 4 Warren High           8141 De Palma St.
## 5 Hollywood Senior High 1521 N. Highland Ave.
## 6 Venice Senior High    13000 Venice Blvd.
```

```
nrow(df_CA2)
```

```
## [1] 713
```

3.3 Sorting data

Base R `sort()` for vectors

`sort()` is a base R function that sorts vectors - Syntax: `sort(x, decreasing=FALSE, ...)` ; where x is object being sorted - By default it sorts in ascending order (low to high) - Need to set decreasing argument to `TRUE` to sort from high to low

```
?sort()
x<- c(31, 5, 8, 2, 25)
sort(x)
```

```
## [1]  2  5  8 25 31
```

```
sort(x, decreasing = TRUE)
```

```
## [1] 31 25  8  5  2
```

Base R `order()` for dataframes

`order()` is a base R function that sorts vectors

- Syntax: `order(..., na.last = TRUE, decreasing = FALSE)`
- where `...` are variable(s) to sort by
- By default it sorts in ascending order (low to high)
- Need to set decreasing argument to `TRUE` to sort from high to low

Descending argument only works when we want either one (and only) variable descending or all variables descending (when sorting by multiple vars)

- use `-` when you want to indicate which variables are descending while using the default ascending sorting

```
df_event[order(df_event$event_date), ]
df_event[order(df_event$event_date, df_event$total_12), ]

#sort descending via argument
df_event[order(df_event$event_date, decreasing = TRUE), ]
df_event[order(df_event$event_date, df_event$total_12, decreasing = TRUE), ]

#sorting by both ascending and descending variables
df_event[order(df_event$event_date, -df_event$total_12), ]
```