Module 3: Investigating objects via Base R

Managing and Manipulating Data Using R

Logistics

Download Module 3 Rmd and knit!

- From the class website, download themodule3.Rmd and module3.R files; move the files from the downloads folder to you `HED696C_Rclass/Modules/Module3` subfolder
- Open R Studio via `HED696C_Rclass.rproj`
- Once in R Studio, go to File » Open File...» Navigate to and click on `Module3.Rmd`
- Try to knit `Module3.Rmd` to pdf

What we will do today

1. Logistics

2. Investigating data patterns via Tidyverse [some more practice]

3. Tidyverse vs. Base R

4. Investigating data patterns using Base R
   - 4.1 Subsetting using subsetting operators
   - 4.2 Subsetting using the subset function
   - 4.3 Sorting data

5. Tidyverse vs base R examples [resource for you]

Tips for R Success: Cumulative Learning...

- The tasks we will be working on in class and in assignments going forward will require you to use skills/functions you have learned over the past several weeks (not only material from the current module/week)
- Working with new data and new variables...
- Investigate the object!
  - `length()`
  - `typeof()`
  - `str()`
  - `names()`
  - `nrow()`
  - `ncol()`
  - `count()`
- Check for missing observations
  - `table(, useNA="always")`
  - `is.na()`
  - `count(is.na())`
- Print!
  - `View()`
  - `head()`

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)`

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)

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

getwd()
#> [1] "/Users/karinasalazar/Library/CloudStorage/Dropbox/HED696C_RClass/modules
#load dataset with one obs per recruiting event
load(url("https://github.com/ksalazar3/HED696C_RClass/raw/master/data/recruitin

#load dataset with one obs per high school
load(url("https://github.com/ksalazar3/HED696C_RClass/raw/master/data/recruitin
```

Investigating data patterns via Tidyverse [some more practice]

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	Upcoming 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

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

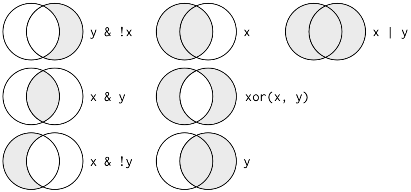

Figure 1: "Boolean" operations, x=left circle, y=right circle, from Wichkam (2018)

Some More Practice with Tidyverse

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

1. Investigate the dataframe `df_event`, including printing variable names (you'll need these to answer the rest of the questions)
2. Count the number of events attended by the University of Pittsburgh (Pitt). Universities have unique ID numbers contained in the variable `univ_id.` Pitt's unique ID is 215293.
3. Count the number of recruiting events by Pitt at public or private high schools
4. Count the number of recruiting events by Pitt at public or private high schools located in the state of PA
5. Count the number of recruiting events by Pitt at public high schools not located in PA where median income is less than 100,000
6. 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
7. 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. Investigate the dataframe `df_event`, including printing variable names (you'll need these to answer the rest of the questions)

```
typeof(df_event)
#> [1] "list"
str(df_event)
#> tibble [18,680 x 33] (S3: tbl_df/tbl/data.frame)
#>  $ instnm               : chr [1:18680] "UM Amherst" "UM Amherst" "UM Amherst"
#>  $ univ_id              : int [1:18680] 166629 166629 166629 166629 196097 218
#>  $ instst               : chr [1:18680] "MA" "MA" "MA" "MA" ...
#>  $ pid                  : int [1:18680] 57570 56984 57105 57118 16281 8608 568
#>  $ event_date           : Date[1:18680], format: "2017-10-12" "2017-10-04" ...
#>  $ event_type           : chr [1:18680] "public hs" "public hs" "public hs" "p
#>  $ zip                  : chr [1:18680] "01002" "01007" "01020" "01020" ...
#>  $ school_id            : chr [1:18680] "250192000042" "250243000134" "2503660
#>  $ ipeds_id             : int [1:18680] NA NA NA NA NA NA NA NA NA NA ...
#>  $ event_state          : chr [1:18680] "MA" "MA" "MA" "MA" ...
#>  $ event_inst           : chr [1:18680] "In-State" "In-State" "In-State" "In-S
#>  $ med_inc              : num [1:18680] 71714 89122 70136 70136 71024 ...
#>  $ pop_total            : num [1:18680] 29970 14888 30629 30629 17872 ...
#>  $ pct_white_zip        : num [1:18680] 73.7 91.4 79.4 79.4 88.7 ...
#>  $ pct_black_zip        : num [1:18680] 5.27 0.84 3.03 3.03 1.76 ...
#>  $ pct_asian_zip        : num [1:18680] 11.69 2.98 1.26 1.26 1.34 ...
#>  $ pct_hispanic_zip     : num [1:18680] 6.24 2.05 14.64 14.64 6.59 ...
#>  $ pct_amerindian_zip   : num [1:18680] 0.2469 0 0.1339 0.1339 0.0168 ...
#>  $ pct_nativehawaii_zip: num [1:18680] 0.0567 0 0 0 0 0 0 0 0 0 ...
#>  $ pct_tworaces_zip     : num [1:18680] 2.59 2.55 1.51 1.51 1.64 ...
```

Solution

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

```r
typeof(df_event$univ_id)
#> [1] "integer"
table(df_event$univ_id, useNA = "always")
#> 
#> 100751 106397 110635 110653 126614 139959 149222 155317 166629 181464 186380
#>   4258    994    879    539   1439    827    549   1014    908   1397   1135
#> 196097 199193 201885 215293 218663   <NA>
#>    730    640    679   1225   1467      0

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

Solution

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

```
str(df_event$event_type)
#>  chr [1:18680] "public hs" "public hs" "public hs" "public hs" "public hs" ..
typeof(df_event$event_type)
#> [1] "character"
table(df_event$event_type, useNA = "always")
#> 
#> 2yr college 4yr college       other  private hs   public hs        <NA> 
#>         951         531        2001        3774       11423           0


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

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

```
str(df_event$event_state)
#>  chr [1:18680] "MA" "MA" "MA" "MA" "MA" "MA" "MA" "MA" "MA" "MA" "MA" "MA" ..
typeof(df_event$event_state)
#> [1] "character"
table(df_event$event_state, useNA = "always")
#> 
#>   AL   AR   AZ   CA  CA    CO   CT  CT    DC   DE  DE    FL  FL    GA   HI
#>  395  248  138 2039   46  834  341    4   76   50   14  718   17  881   30
#>   ID   IL  IL     IN   KS   KY   LA   MA  MA     MD  MD   MD     ME   MI  MI    M
#>    1 1420   12  125  427  173   68  619   12  602    1   38   45  128    1
#>   MN   MO   MS   NC   ND   NE   NH  NH    NJ  NJ   NJ    NM   NV   NY  NY    N
#>  247  490   42  888    5  595   87    3  864    1   39   34   52 1010   16
#>   OH   OK   OR   PA  PA    RI  RI    SC   SD   TN   TX  TX    UT   VA  VA
#>  392  127   97  654   21   37    2  423   43  283 1558   10   10  572   19
#>   WA   WI   WV <NA>
#>  247  136    7    0

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
```

Solution

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

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

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

6. 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
```

7. 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
```

Tidyverse vs. Base R

Why learn to “wrangle” data both via tidyverse and Base R?

- **Base R**: “core” R commands for cleaning and manipulating data that are not part of any external package/library
- **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

Investigating data patterns using Base R

Tidyverse vs. base R functions

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

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.

```r
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 UM Amherst   166629 MA     57570
#>  2 UM Amherst   166629 MA     56984
#>  3 UM Amherst   166629 MA     57105
#>  4 UM Amherst   166629 MA     57118
#>  5 Stony Brook  196097 NY     16281
#>  6 USCC         218663 SC      8608
#>  7 UM Amherst   166629 MA     56898
#>  8 UM Amherst   166629 MA     56933
#>  9 UM Amherst   166629 MA     56940
#> 10 UM Amherst   166629 MA     57030
#> # i 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 UM Amherst   166629 MA
#> 2 UM Amherst   166629 MA
#> 3 UM Amherst   166629 MA
#> 4 UM Amherst   166629 MA
#> 5 Stony Brook  196097 NY
```

```
#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 UM Amherst   166629 MA
#>  2 UM Amherst   166629 MA
#>  3 UM Amherst   166629 MA
#>  4 UM Amherst   166629 MA
#>  5 Stony Brook  196097 MA
#>  6 USCC         218663 MA
#>  7 UM Amherst   166629 MA
#>  8 UM Amherst   166629 MA
#>  9 UM Amherst   166629 MA
#> 10 UM Amherst   166629 MA
#> # i 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_wh
#>   <chr>      <chr>       <chr>        <chr>     <chr>   <chr> <chr>         <d
#> 1 CA         public      064015006647 Tustin H~ 1171 E~ Tust~ 92780        13.
#> 2 CA         public      062547003790 Bell Gar~ 6119 A~ Bell~ 90201         0.
#> 3 CA         public      063531006005 Santa An~ 520 W.~ Sant~ 92701         0.
#> # i 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
```

Subsetting Data Frames with [] and $, NA Observations

- When extracting observations via subsetting operators, resulting dataframe will include rows where condition is `TRUE`; **as well as** `NA` values.
- To remove missing values, ask for values that only evaluate to `TRUE` explicitly via `which()`
- Task: Show all public high schools with at least $50k median household incomes

tidyverse

```
df_tv <- filter(df_event, event_type == "public hs" & med_inc>=50000)
nrow(df_tv) #9,941 obs
```

base R without `which()`

```
df_b1 <- df_event[df_event$event_type == "public hs" & df_event$med_inc>=50000,
nrow(df_b1) #10,016 obs
view(df_b1) #NAs sorted at the end of column
```

base R with `which()`

```
df_b2 <- df_event[which(df_event$event_type == "public hs" & df_event$med_inc>=
nrow(df_b2) #9,941 obs, same as tidyverse way
```

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
```

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

```r
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), ]
```

Tidyverse vs base R examples [resource for you]

Extracting columns (variables)

-Create a new dataframe by extracting the columns `instnm`, `event_date`, `event_type` from df_event. Use the `names()` function to show what columns/variables are in the newly created dataframe.

tidyverse

```
df_event_tv <- select(df_event, instnm, event_date, event_type)
names(df_event_tv)
#> [1] "instnm"     "event_date" "event_type"
```

base R using subsetting operators

```
df_event_br1 <- df_event[, c("instnm", "event_date", "event_type")]
names(df_event_br1)
#> [1] "instnm"     "event_date" "event_type"
```

base R using `subset()` function

```
df_event_br2 <- subset(df_event, select=c(instnm, event_date, event_type))
names(df_event_br2)
#> [1] "instnm"     "event_date" "event_type"
```

Extracting observations

-Create a new dataframe from df_schools that includes out-of-state public high schools with 50%+ Latinx student enrollment that received at least one visit by the University of California Berkeley.

tidyverse

```
df_school_tv <- filter(df_school, state_code != "CA"
                       & school_type == "public" &
                         pct_hispanic >= 50 & visits_by_110635 >=1 )
nrow(df_school_tv)
#> [1] 10
```

base R using subsetting operators

```
df_school_br1 <- df_school[which(df_school$state_code != "CA"
                  & df_school$school_type == "public"
                  & df_school$pct_hispanic >= 50
                  & df_school$visits_by_110635 >=1), ]
nrow(df_school_br1)
#> [1] 10
```

base R using `subset()` function

```
df_school_br2 <- subset(df_school, state_code != "CA" & school_type == "public"
                        & pct_hispanic >= 50
                        & visits_by_110635 >=1 )
nrow(df_school_br2)
#> [1] 10
```

Sorting observations

-Create a new dataframe from df_events that sorts by ascending by `event_date`, ascending `event_state`, and descending `pop_total` .

tidyverse

```
df_event_tv <- arrange(df_event, event_date, event_state, desc(pop_total))
```

base R using `order()` function

```
df_event_br1 <- df_event[order(df_event$event_date, df_event$event_state,
                               -df_event$pop_total), ]
```