Lecture 3: Investigating data patterns using Base R

Managing and Manipulating Data Using R

1 Introduction

What we will do today

1. Introduction

2. Subsetting using subset() function

3. Subsetting using subsetting operators
   - 3.1 Subset atomic vectors using []
   - 3.2 Subsetting lists/data frames using []
   - 3.3 Subsetting lists/data frames using [[]] and $
   - 3.4 Subsetting data frames with [] combined with $

4. Sorting data

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   1.0.0     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
```

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

2 Subsetting using subset() function

Subset function

- The `subset()` is a base R function and easiest way to “filter” observations
  - can also used `subset()` to select variables
  - Like tidyverse `filter()`, `subset()` can be combined with:
    - with assignment (`<-`) to create new objects
    - with `count()` to count number of observations that satisfy criteria

```
?subset
```

- Syntax [when object is data frame]: **subset(x, subset, select, drop = FALSE)**
  - `x` is object to be subset
  - `subset` is the logical expression(s) (evaluates to `TRUE/FALSE`) indicating elements (rows) to keep
  - `select` indicates columns to select from data frame (if argument is not used default will keep all columns)
  - `drop` to preserve original **dimensions** [SKIP]
    - cane take values `TRUE` or `FALSE`; default is `FALSE`
    - only need to worry about dataframes when subset output is single column

Subset function, examples

Using `df_school`, show all public high schools that are at least 50% Latinx (var=`pct_hispanic`) student enrollment in California

Using tidyverse `filter()` [output omitted]

```r
filter(df_school, school_type == "public", pct_hispanic >= 50,
   state_code == "CA")

filter(df_school, school_type == "public" & pct_hispanic >= 50
   & state_code == "CA") # same as above
```

Using base R, `subset()` [output omitted]

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

Subset function, examples

Count all CA public high schools that are at least 50% Latinx

- Can wrap `filter()` or `subset()` within `count()` to count number of observations that satisfy criteria

```
#filter()
count(filter(df_school, school_type == "public", pct_hispanic >= 50,
   state_code == "CA"))
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1   713
count(filter(df_school, school_type == "public" & pct_hispanic >= 50
   & state_code == "CA"))
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1   713

#subset()
count(subset(df_school, school_type == "public" & pct_hispanic >= 50
     & state_code == "CA"))
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1   713
```

Subset function, examples

Note that both `filter()` and `subset()` identify the number of observations for which the condition is `TRUE`

```r
count(filter(df_school, TRUE))
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1 21301
count(subset(df_school, TRUE))
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1 21301

count(filter(df_school, FALSE))
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1     0
count(subset(df_school, FALSE))
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1     0
```

Subset function, examples

Count all CA public high schools that are at least 50% Latinx and received at least 1 visit from UC Berkeley (var= `visits_by_110635`)

```
#filter()
count(filter(df_school, school_type == "public", pct_hispanic >= 50,
   state_code == "CA", visits_by_110635 >= 1))
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1   100

#subset()
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
```

Subset function, examples

`subset()` can also use %in% operator, which is more efficient version of **OR** operator `|`

Count number of schools from MA, ME, or VT that received at least one visit from University of Alabama (var= `visits_by_100751`)

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

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

Subset function, examples

Use the `select` argument within `subset()` to keep selected variables

syntax: `select = c(var_name1,var_name2,...,var_name_n)`

Subset all CA public high schools that are at least 50% Latinx **AND** only keep variables `name` and `address`

```
subset(df_school, school_type == "public" & pct_hispanic >= 50
             & state_code == "CA", select = c(name, address))
#> # A tibble: 713 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.
#>  7 Sequoia High                1201 Brewster Ave.
#>  8 Santa Barbara Senior High 700 E. Anapamu St.
#>  9 Santa Paula High            404 N. Sixth St.
#> 10 Azusa High                  240 N. Cerritos Ave.
#> # ... with 703 more rows
```

Subset function, examples

Combine `subset()` with assignment (`<-`) to create a new data frame

Create a new date frame of all CA public high schools that are at least 50% Latinx **AND** only keep variables `name` and `address`

```
df_school_v2 <- subset(df_school, school_type == "public" & pct_hispanic >= 50
  & state_code == "CA", select = c(name, address))

head(df_school_v2, n=5)
#> # A tibble: 5 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.

nrow(df_school_v2)
#> [1] 713
```

Student Exercises

Compare tidyverse to subset() from base R in extracting columns (variables), observations:

1. Use both base R and tidyverse to create a new dataframe by extracting the columns `instnm`, `event_date`, `event_type` from df_event. And show what columns (variables) are in the newly created dataframe.

2. Use both base R and tidyverse to create a new dataframe from df_school 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 (var= visits_by_110635). And count the number of observations.

3. Use both base R and tidyverse to count the number of public schools from CA, FL or MA that received one or two visits from UC Berkeley from df_school.

4. Use base R to subset all public out-of-state high schools visited by University of California Berkeley that enroll at least 50% Black students, and only keep variables “state_code”, “name” and “zip_code” .

Solution to Student Exercises

Solution to 1

base R using `subset()` function

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

tidyverse using `select()` function

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

Solution to 2

base R using `subset()` function

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

tidyverse using `filter()` function

```
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
```

Solution to Student Exercises

Solution to 3

base R using `subset()` function

```
count(subset(df_school, state_code %in% c("CA", "FL", "MA")
             & school_type == "public" & visits_by_110635 %in% c(1,2) ))
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1   246
```

tidyverse using `filter()` function

```
count(filter(df_school, state_code %in% c("CA", "FL", "MA")
             & school_type == "public" & visits_by_110635 %in% c(1,2) ))
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1   246
```

Solution to 4

base R using `subset()` function

```
subset(df_school, school_type == "public" & state_code != "CA"
       & visits_by_100751 >= 1 & pct_hispanic >= 50,
       select = c(state_code, name, zip_code))
#> # A tibble: 73 x 3
#>    state_code name                                  zip_code
#>    <chr>      <chr>                                 <chr>
#>  1 AZ         Agua Fria High School                 85323
```

3 Subsetting using subsetting operators

Subsetting to Extract Elements

"Subsetting" refers to isolating particular elements of an object

Subsetting operators can be used to select/exclude elements (e.g., variables, observations)

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

Wichham refers to number of “dimensions” in R objects

An atomic vector is a 1-dimensional object that contains n elements

```
x <- c(1.1, 2.2, 3.3, 4.4, 5.5)
str(x)
#>  num [1:5] 1.1 2.2 3.3 4.4 5.5
```

Lists are multi-dimensional objects

Contains n elements; each element may contain a 1-dimensional atomic vector or a multi-dimensional list. Below list contains 3 dimensions

```
list <- list(c(1,2), list("apple", "orange"))
str(list)
#> List of 2
#>  $ : num [1:2] 1 2
#>  $ :List of 2
#>   ..$ : chr "apple"
#>   ..$ : chr "orange"
```

Data frames are 2-dimensional lists

each element is a variable (dimension=columns)

within each variable, each element is an observation (dimension=rows)

```
ncol(df_school)
#> [1] 26
nrow(df_school)
#> [1] 21301
```

3.1 Subset atomic vectors using []

Subsetting elements of atomic vectors

“Subsetting” a vector refers to isolating particular elements of a vector

- I sometimes refer to this as “accessing elements of a vector”
- subsestting elements of a vector is similar to “filtering” rows of a data-frame
- `[]` is the subsetting function for vectors

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

1. Using positive integers to return elements at specified positions
2. Using negative integers to exclude elements at specified positions
3. Using logicals to return elements where corresponding logical is `TRUE`
4. Empty `[]` returns original vector (useful for dataframes)
5. Zero vector [0], useful for testing data
6. If vector is “named,” use character vectors to return elements with matching names

1. Using positive integers to return elements at specified positions (subset atomic vectors using [])

Create atomic vector `x`

```r
(x <- c(1.1, 2.2, 3.3, 4.4, 5.5))
#> [1] 1.1 2.2 3.3 4.4 5.5
str(x)
#>  num [1:5] 1.1 2.2 3.3 4.4 5.5
```

`[]` is the subsetting function for vectors

contents inside `[]` can refer to element number (also called "position").

e.g., `[3]` refers to contents of 3rd element (or position 3)

```r
x[5] #return 5th element
#> [1] 5.5

x[c(3, 1)] #return 3rd and 1st element
#> [1] 3.3 1.1

x[c(4,4,4)] #return 4th element, 4th element, and 4th element
#> [1] 4.4 4.4 4.4

#Return 3rd through 5th element
str(x)
#>  num [1:5] 1.1 2.2 3.3 4.4 5.5
x[3:5]
#> [1] 3.3 4.4 5.5
```

2. Using negative integers to exclude elements at specified positions (subset atomic vectors using [])

Before excluding elements based on position, investigate object

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

length(x)
#> [1] 5
str(x)
#>  num [1:5] 1.1 2.2 3.3 4.4 5.5
```

Use negative integers to exclude elements based on element position

```
x[-1] # exclude 1st element
#> [1] 2.2 3.3 4.4 5.5

x[-c(3,1)] # exclude 3rd and 1st element
#> [1] 2.2 4.4 5.5
```

3. Using logicals to return elements where corresponding logical is `TRUE` (subset atomic vectors using [])

```r
x
#> [1] 1.1 2.2 3.3 4.4 5.5
```

When using `x[y]` to subset `x`, good practice to have `length(x)==length(y)`

```r
length(x) # length of vector x
#> [1] 5
length(c(TRUE,FALSE,TRUE,FALSE,TRUE)) # length of y
#> [1] 5
length(x) == length(c(TRUE,FALSE,TRUE,FALSE,TRUE)) # condition true
#> [1] TRUE
x[c(TRUE,TRUE,FALSE,FALSE,TRUE)]
#> [1] 1.1 2.2 5.5
```

Recycling rules:

> in `x[y]`, if `x` is different length than `y`, R “recycles” length of shorter to match length of longer

```r
length(c(TRUE,FALSE))
#> [1] 2
x[c(TRUE,FALSE)]
#> [1] 1.1 3.3 5.5
```

3. Using logicals to return elements where corresponding logical is `TRUE` (subset atomic vectors using [])

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

Note that a missing value (`NA`) in the index always yields a missing value in the output

```
x[c(TRUE, FALSE, NA, TRUE, NA)]
#> [1] 1.1  NA 4.4  NA
```

Return all elements of object `x` where element is greater than 3

```
x[x>3]
#> [1] 3.3 4.4 5.5
```

4. Empty `[]` returns original vector (subset atomic vectors using [])

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

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

This is useful for sub-setting data frames, as we will show below

5. Zero vector [0] (subset atomic vectors using [])

Zero vector, `x[0]`

> R interprets this as returning element 0

```
x[0]
#> numeric(0)
```

Wickham states:

> "This is not something you usually do on purpose, but it can be helpful for generating test data."

6. If vector is named, character vectors to return elements with matching names (subset atomic vectors using [])

Create vector `y` that has values of vector `x` but each element is named

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

(y <- c(a=1.1, b=2.2, c=3.3, d=4.4, e=5.5))
#>   a   b   c   d   e
#> 1.1 2.2 3.3 4.4 5.5
```

Return elements of vector based on name of element

enclose element names in single `''` or double `""` quotes

```
#show element named "a"
y["a"]
#>   a
#> 1.1

#show elements "a", "b", and "d"
y[c("a", "b", "d" )]
#>   a   b   d
#> 1.1 2.2 4.4
```

3.2 Subsetting lists/data frames using []

Subsetting lists using []

Using `[]` operator to subset lists works the same as subsetting atomic vector

Using `[]` with a list always returns a list

```r
list_a <- list(list(1,2),3,"apple")
str(list_a)
#> List of 3
#>  $ :List of 2
#>   ..$ : num 1
#>   ..$ : num 2
#>  $ : num 3
#>  $ : chr "apple"

#create new list that consists of elements 3 and 1 of list_a
list_b <- list_a[c(3, 1)]
str(list_b)
#> List of 2
#>  $ : chr "apple"
#>  $ :List of 2
#>   ..$ : num 1
#>   ..$ : num 2

#show elements 3 and 1 of object list_a
#str(list_a[c(3, 1)])
```

Subsetting data frames using []

Recall that a data frame is just a particular kind of list

- each element = a column = a variable

Using `[]` with a list always returns a list

- Using `[]` with a data frame always returns a data frame

Two ways to use `[]` to extract elements of a data frame

1. use "single index" `df_name[<columns>]` to extract columns (variables) based on element position number (i.e., column number)
2. use "double index" `df_name[<rows>, <columns>]` to extact particular rows and columns of a data frame

Subsetting data frames using [] to extract columns (variables) based on element position

Use “single index” `df_name[<columns>]` to extract columns (variables) based on element number (i.e., column number)

Examples [output omitted]

```r
names(df_event)

#extract elements 1 through 4 (elements=columns=variables)
df_event[1:4]
df_event[c(1,2,3,4)]

#extract columns 13 and 7
df_event[c(13,7)]
```

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

use “double index” syntax `df_name[<rows>, <columns>]` to extact particular rows and columns of a data frame

- often combined with sequences (e.g., `1:10`)

```
#Return rows 1-3 and columns 1-4
df_event[1:3, 1:4]
#> # A tibble: 3 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

#Return rows 50-52 and columns 10 and 20
df_event[50:52, c(10,20)]
#> # A tibble: 3 x 2
#>   event_state pct_tworaces_zip
#>   <chr>                  <dbl>
#> 1 MA                      1.98
#> 2 MA                      1.98
#> 3 MA                      1.98
```

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

use “double index” syntax `df_name[<rows>, <columns>]` to extact particular rows and columns of a data frame

recall that empty `[]` returns original object (output omitted)

```
#return original data frame
df_event[]

#return specific rows and all columns (variables)
df_event[1:5, ]

#return all rows and specific columns (variables)
df_event[, c(1,2,3)]
```

Use [] to extract data frame columns based on variable names

Selecting columns from a data frame by subsetting with `[]` and list of element names (i.e., variable names) enclose in quotes

“single index” approach extracts specific variables, all rows (output omittted)

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

“Double index” approach extracts specific variables and specific rows

syntax `df_name[<rows>, <columns>]`

```
df_event[1:5, c("instnm", "event_state", "event_type")]
#> # A tibble: 5 x 3
#>   instnm      event_state event_type
#>   <chr>       <chr>       <chr>
#> 1 UM Amherst  MA          public hs
#> 2 UM Amherst  MA          public hs
#> 3 UM Amherst  MA          public hs
#> 4 UM Amherst  MA          public hs
#> 5 Stony Brook MA          public hs
```

Student exercises

Use subsetting operators from base R in extracting columns (variables), observations:

1. Use both “single index” and “double index” in subsetting to create a new dataframe by extracting the columns `instnm`, `event_date`, `event_type` from df_event. And show what columns (variables) are in the newly created dataframe.
2. Use subsetting to return rows 1-5 of columns `state_code`, `name`, `address` from df_school.

Solution to Student Exercises

Solution to 1

base R using subsetting operators

```
# single index
df_event_br <- df_event[c("instnm", "event_date", "event_type")]
#double index
df_event_br <- df_event[, c("instnm", "event_date", "event_type")]
names(df_event_br)
#> [1] "instnm"     "event_date" "event_type"
```

Solution to 2

base R using subsetting operators

```
df_school[1:5, c("state_code", "name", "address")]
#> # A tibble: 5 x 3
#>   state_code name                        address
#>   <chr>      <chr>                       <chr>
#> 1 AK         Bethel Regional High School 1006 Ron Edwards Memorial Dr
#> 2 AK         Ayagina'ar Elitnaurvik      106 Village Road
#> 3 AK         Kwigillingok School         108 Village Road
#> 4 AK         Nelson Island Area School   118 Village Road
#> 5 AK         Alakanuk School             9 School Road
```

3.3 Subsetting lists/data frames using [[]] and $

Subset single element from object using [[]] operator

- So far we have used `[]` to excract elements from an object
    - Applying `[]` to an atomic vector returns an atomic vector with specific elements you requested
    - Applying `[]` to a list returns a shorter list that contains the specific elements you requested
- `[[]]` also extract elements from an object
    - Applying `[[]]` gives same result as `[]`; that is, an atomic vector with element you request

```
x <- c(1.1, 2.2, 3.3, 4.4, 5.5)
str(x[3])
#>  num 3.3
str(x[[3]])
#>  num 3.3
```

- Applying `[[]]` to list gives the "contents" of the list, rather than list itself

```
list_a <- list(1:3, "a", 4:6)
str(list_a[1])
#> List of 1
#>  $ : int [1:3] 1 2 3
str(list_a[[1]])
#>  int [1:3] 1 2 3
```

Subset single element from object using [[]] operator

Wickham “Advanced R” chapter 4.3 [LINK HERE] uses “Train Metaphor” to differentiate list vs. contents of list

The list is the entire train. Create a list with three elements (three “carriages”)

```r
list_a <- list(1:3, "a", 4:6)
str(list_a)
#> List of 3
#>  $ : int [1:3] 1 2 3
#>  $ : chr "a"
#>  $ : int [1:3] 4 5 6
```

When extracting element(s) of a list you have two options:

1. Extracting elements using `[]` always returns a smaller list (smaller train)

```r
str(list_a[1]) # returns a list
#> List of 1
#>  $ : int [1:3] 1 2 3
```

2. Extracting element using `[[]]` returns contents of particular carriage

    I say applying `[[]]` to a list or data frame returns a simpler object that moves up one level of hierarchy

```r
str(list_a[[1]]) # returns an atomic vector
#>  int [1:3] 1 2 3
```

Subset single element from object using [[]] operator

In contrast to `[]`, we use `[[]]` to extract individual elements rather than multiple elements

- we could write `x[4]` or `x[4:6]`
- we could write `x[[4]]` but not `x[[4:6]]`

Subset single element from object using [[]] operator

Just like `[]` can use `[[]]` to return contents of **named** elements, specified using quotes

syntax: `obj_name[["element_name"]]`

```
list_b <- list(var1=1:3, var2="a", var3=4:6)
str(list_b)
#> List of 3
#>  $ var1: int [1:3] 1 2 3
#>  $ var2: chr "a"
#>  $ var3: int [1:3] 4 5 6

str(list_b["var1"])
#> List of 1
#>  $ var1: int [1:3] 1 2 3

str(list_b[["var1"]])
#>  int [1:3] 1 2 3
```

Works the same with data frames

```
str(df_event["zip"])
#> Classes 'tbl_df', 'tbl' and 'data.frame':    18680 obs. of  1 variable:
#>  $ zip: chr  "01002" "01007" "01020" "01020" ...

str(df_event[["zip"]])
#>  chr [1:18680] "01002" "01007" "01020" "01020" "01027" "01027" "01027" ...
```

Subset lists/data frames using $

`obj_name$element_name` shorthand operator for `obj_name[["element_name"]]`

```
str(list_b)
#> List of 3
#>  $ var1: int [1:3] 1 2 3
#>  $ var2: chr "a"
#>  $ var3: int [1:3] 4 5 6

list_b[["var1"]]
#> [1] 1 2 3
list_b$var1
#> [1] 1 2 3

str(list_b[["var1"]])
#>  int [1:3] 1 2 3
str(list_b$var1)
#>  int [1:3] 1 2 3
```

`df_name$var_name` : easiest way in base R to refer to variable in a data frame

```
str(df_event[["zip"]])
#>  chr [1:18680] "01002" "01007" "01020" "01020" "01027" "01027" "01027" ...
str(df_event$zip)
#>  chr [1:18680] "01002" "01007" "01020" "01020" "01027" "01027" "01027" ...
```

3.4 Subsetting data frames with [] combined with $

Subsetting Data Frames with [] combined with $

Combine `[]` with `$` to subset data frame same as `filter()` or `subset()`

Syntax: `df_name[df_name$var_name <condition>, ]`

- Note: Uses “double index” `df_name[<rows>, <columns>]` syntax
- **Cannot** use “single index” `df_name[<columns>]`

Examples (output omitted)

- All observations where the hich school received at least 1 visit from UC Berkeley (var= `visits_by_110635`) and all columns

```
df_school[df_school$visits_by_110635 >= 1, ]
```

- All obs where the high school received at least 1 visit from UC Berkeley and the first three columns

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

- All obs where the high school received at least 1 visit from UC Berkeley and variables “state_code” “school_type” “name”

```
df_school[df_school$visits_by_110635 == 1, c("state_code","school_type","name")]
```

Subsetting Data Frames with [] combined with $

- Combine `[]` with `$` to subset data frame same as `filter()` or `subset()`
    - Syntax: `df_name[df_name$var_name <condition>, ]`
    - Can be combined with `count()` or `nrow()` to avoid printing many rows
- Count obs where high schools received at least 1 visit by Bama (100751) and at least one visit by Berkeley (110635)
    - compare with `filter()` and `subset()` approaches

```
#[] combined with $ approach
count(df_school[df_school$visits_by_110635 >= 1
  & df_school$visits_by_100751 >= 1, ])
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1   247

#filter() approach
nrow(filter(df_school, visits_by_110635 >= 1, visits_by_100751 >= 1))
#> [1] 247

#subset() approach
nrow(subset(df_school, visits_by_110635 >= 1 & visits_by_100751 >= 1))
#> [1] 247
```

Subsetting Data Frames with [] and $, NA Observations

When sub-setting via `[]` combined with `$`, result will include:

- rows where condition is `TRUE`
- **as well as** rows with `NA` (missing) values for condition.

Task: How many events at public high schools with at least $50k median household income

- extracting observations via `[]` combined with `$`

```r
#num obs event_type=="public hs" and med_inc is missing
nrow(df_event[df_event$event_type == "public hs"
  & is.na(df_event$med_inc)==1 , ])
#> [1] 75

#num obs event_type=="public hs" & med_inc is not NA & med_inc >= $50,000
nrow(df_event[df_event$event_type == "public hs"
  & is.na(df_event$med_inc)==0 & df_event$med_inc>=50000 , ])
#> [1] 9941

#num obs event_type=="public hs" and med_inc >= $50,000
nrow(df_event[df_event$event_type == "public hs"
  & df_event$med_inc>=50000 , ])
#> [1] 10016
```

Subsetting Data Frames with [] and $, NA Observations

subset using `[]` combined with `$`, result includes:

rows where condition `TRUE`; **AND** rows with `NA` for condition

Base R filter using `subset()` excludes rows with `NA` for condition

```r
#num obs event_type=="public hs" and med_inc is missing
nrow(subset(df_event, event_type == "public hs" & is.na(med_inc)==1))
#> [1] 75
#num obs event_type=="public hs" & med_inc is not NA & med_inc >= $50,000
nrow(subset(df_event, event_type == "public hs" & is.na(med_inc)==0 & med_inc>=5
#> [1] 9941
#num obs event_type=="public hs" & med_inc >= $50,000
nrow(subset(df_event, event_type == "public hs" & med_inc>=50000))
#> [1] 9941
```

Tidyverse `filter()` excludes rows with `NA` for condition.

```r
#num obs event_type=="public hs" and med_inc is missing
nrow(filter(df_event, event_type == "public hs", is.na(med_inc)==1))
#> [1] 75
#num obs event_type=="public hs" & med_inc is not NA & med_inc >= $50,000
nrow(filter(df_event, event_type == "public hs", is.na(med_inc)==0, med_inc>=500
#> [1] 9941
#num obs event_type=="public hs" & med_inc >= $50,000
nrow(filter(df_event, event_type == "public hs", med_inc>=50000))
#> [1] 9941
```

Subsetting Data Frames with [] and $, NA Observations

To exclude rows where condition is `NA` if subset using `[]` combined w/ `$`

- use `which()` to ask only for values where condition evaluates to `TRUE`
- `which()` returns position numbers for elements where condition is `TRUE`

```
#?which
c(TRUE,FALSE,NA,TRUE)
#> [1]  TRUE FALSE    NA  TRUE
str(c(TRUE,FALSE,NA,TRUE))
#>  logi [1:4] TRUE FALSE NA TRUE
which(c(TRUE,FALSE,NA,TRUE))
#> [1] 1 4
```

Task: Count events at public HS with at least $50k median household income?

```
#Tidyverse, filter()
nrow(filter(df_event, event_type == "public hs" & med_inc>=50000))
#> [1] 9941

#Base R, `[]` combined with `$`; without which()
nrow(df_event[df_event$event_type == "public hs" & df_event$med_inc>=50000, ])
#> [1] 10016

#Base R, `[]` combined with `$`; with which()
nrow(df_event[which(df_event$event_type == "public hs"
  & df_event$med_inc>=50000), ])
#> [1] 9941
```

Student Exercises

Subsetting Data Frames with (1) [] and $; (2) subset() and filter():

1. Show how many public high schools in California with at least 50% Latinx (hispanic in data) student enrollment from df_school.
2. Show how many out-state events at public high schools with more than $30K median from df_event (do not forget to exclude missing values).

Solution to Student Exercises

Solution to 1

base R using [] and $

```
df_school_br1<- df_school[df_school$school_type == "public"
                 & df_school$pct_hispanic >= 50
                 & df_school$state_code == "CA", ]
nrow(df_school_br1)
#> [1] 713
```

base R using `subset()` function

```
df_school_br2 <- subset(df_school, school_type == "public"
                 & pct_hispanic >= 50
                 & state_code == "CA" )
nrow(df_school_br2)
#> [1] 713
```

tidyverse using `filter()` function

```
df_school_tv <- df_school %>% filter(school_type == "public"
                 & pct_hispanic >= 50
                 & state_code == "CA" )
nrow(df_school_tv)
#> [1] 713
```

Solution to Student Exercises

Solution to 2:

base R using [] and $ (NA included)

```
# use is.na to exclude NA
nrow(df_event[df_event$event_type == "public hs" & df_event$event_inst =="Out-St
              & df_event$med_inc > 30000 & is.na(df_event$med_inc) ==0, ])
#> [1] 7784

# use which to exclude NA
nrow(df_event[which(df_event$event_type == "public hs" & df_event$event_inst =="
              & df_event$med_inc > 30000 ), ])
#> [1] 7784
```

base R using `subset()` function (NA excluded)

```
nrow(subset(df_event, event_type == "public hs"
                        & event_inst =="Out-State"& df_event$med_inc > 30000 ))
#> [1] 7784
```

tidyverse using `filter()` function (NA excluded)

```
count(filter(df_event, event_type == "public hs"
                        & event_inst =="Out-State" & df_event$med_inc > 30000 ))
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1  7784
```

4 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), ]
```

Compare tidyverse to base r, sorting

-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), ]
```