

Lecture 4: Pipes and variable creation

Managing and Manipulating Data Using R

Introduction

What we will do today

1. Introduction

1.1 Data for lecture

2. Pipes

3. Creating variables using mutate (tidyverse approach)

3.1 Introduce mutate() function

3.2 Using ifelse() function within mutate()

3.3 Using recode() function within mutate()

3.4 Using case_when() function within mutate()

4. Base R approach to creating new variables

Libraries we will use today

“Load” the package we will use today (output omitted)

▶ **you must run this code chunk**

```
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)`

Data for lecture

Lecture 3 data: prospects purchased by Western Washington U.

The “Student list” business

- ▶ Universities identify/target “prospects” by buying “student lists” from College Board/ACT (e.g., \$.40 per prospect)
- ▶ Prospect lists contain contact info (e.g., address, email), academic achievement, socioeconomic, demographic characteristics
- ▶ Universities choose which prospects to purchase by filtering on criteria like zip-code, GPA, test score range, etc.

```
#load prospect list data
```

```
load(url("https://github.com/ozanj/rclass/raw/master/data/prospect_list/wwlist_
```

Object `wwlist`

- ▶ De-identified list of prospective students purchased by Western Washington University from College Board
- ▶ We collected these data using FOIA request
 - ▶ ASIDE: Become an expert on collecting data via FOIA requests and you will become a superstar!

Lecture 3 data: prospects purchased by Western Washington U.

Observations on `wwlist`

- ▶ each observation represents a prospective student

```
typeof(wwlist)
#> [1] "list"
dim(wwlist)
#> [1] 268396    41
```

Variables on `wwlist`

- ▶ some vars provide de-identified data on individual prospects
 - ▶ e.g., `psat_range`, `state`, `sex`, `ethn_code`
- ▶ some vars provide data about zip-code student lives in
 - ▶ e.g., `med_inc`, `pop_total`, `pop_black`
- ▶ some vars provide data about school student enrolled in
 - ▶ e.g., `fr_lunch` is number of students on free/reduced lunch
 - ▶ note: bad merge between prospect-level data and school-level data

```
names(wwlist)
str(wwlist)
glimpse(wwlist) # tidyverse function, similar to str()
```

Lecture 3 data: prospects purchased by Western Washington U.

Variable `firstgen` identifies whether prospect is a first-generation college student

Imagine we want to isolate all the first-generation prospects

1. Investigate variable type/structure.

► A dichotomous var, but stored as character in `wwlist`. So must use quotes (`'` or `"`) to filter/subset based on values of `firstgen`

```
str(wwlist$firstgen)
#> chr [1:268396] NA "N" "N" "N" NA "N" "N" "Y" "Y" "N" "N" "N" "N" "N" ...
```

2. Create frequency table to identify possible values of `firstgen`

```
table(wwlist$firstgen, useNA = "always")
#>
#>      N      Y  <NA>
#> 193333 65046 10017
```

3. Isolate all the first-gen prospects

```
filter(wwlist, firstgen == "Y")
#> # A tibble: 65,046 x 41
#>   receive_date psat_range state zip9 for_country sex hs_eeeb_code
#>   <date>      <chr>      <chr> <chr> <chr>      <chr>      <int>
#> 1 2016-05-31 1170-1520 WA 9812~ <NA>      F      481128
#> 2 2016-05-31 930-1160 WA 9829~ <NA>      M      481335
#> 3 2016-05-31 1030-1160 CO 8012~ <NA>      M      60926
#> 4 2016-05-31 930-1160 WA 9837~ <NA>      F      480442
#> 5 2016-05-31 930-1160 WA 9811~ <NA>      F      181085
```


Pipes

What are “pipes”, %>%

Pipes are a means of performing multiple steps in a single line of code

- ▶ When writing code, the pipe symbol is `%>%`
- ▶ Basic flow of using pipes in code:
 - ▶ `object %>% some_function %>% some_function %>% some_function`
- ▶ Pipes work from left to right:
 - ▶ The object from left of `%>%` pipe symbol is input as the first argument of the function to the right of the `%>%` pipe symbol
 - ▶ In turn, the resulting output becomes the input (the first argument) of the function to the right of the next `%>%` pipe symbol
- ▶ Pipes are part of **tidyverse** suite of packages, not **base R**

Intuitive mnemonic device for understanding pipes

- ▶ whenever you see a pipe `%>%` think of the words “**and then...**”

Example: isolate all the first-generation prospects [output omitted]

- ▶ in words: start with object `wvlist` **and then** filter first generation students
- ```
wvlist %>% filter(firstgen == "Y")
```

## Do task with and without pipes

Task: Using object `wwlist` print data for “first-gen” prospects ( `firstgen == "Y"` )

*# without pipes*

```
filter(wwlist, firstgen == "Y")
```

*# with pipes*

```
wwlist %>% filter(firstgen == "Y")
```

Comparing the two approaches:

- ▶ “without pipes”, object `wwlist` is the first argument `filter()` function
- ▶ In “pipes” approach, you don’t specify object `wwlist` as first argument in `filter()`
  - ▶ Why? Because `%>%` “pipes” the object to the left of the `%>%` operator into the function to the right of the `%>%` operator

Main takeaway:

- ▶ When writing code using pipes, functions to right of `%>%` pipe operator should not explicitly name object that is the input to the function.
- ▶ Rather, object to the left of `%>%` pipe operator is automatically the input.

## More intuition on the pipe operator, `%>%`

The pipe operator “pipes” (verb) an object from left of `%>%` operator into the function to the right of the `%>%` operator

Example, the “structure” function `str()`, with and without pipes

- ▶ Examine syntax for `str()` : `str(object, ...)`

```
?str
```

- ▶ Investigate structure of dataframe `wwlist` without and with pipes

```
str(wwlist) # without pipe
```

```
wwlist %>% str() # with pipe
```

Questions:

- ▶ In the pipes approach, `wwlist %>% str()`, why didn't we need to insert argument values inside `str()`
- ▶ What would happen if we just ran this line of code?

```
str()
```

## Do task with and without pipes

**Task:** Using object `wwlist`, print data for “first-gen” prospects for selected variables [output omitted]

*#Without pipes*

```
select(filter(wwlist, firstgen == "Y"), state, hs_city, sex)
```

*#With pipes*

```
wwlist %>% filter(firstgen == "Y") %>% select(state, hs_city, sex)
```

Comparing the two approaches:

- ▶ In the “without pipes” approach, code is written “inside out”
  - ▶ The first step in the task – identifying the object – is the innermost part of code
  - ▶ The last step in task – selecting variables to print – is the outermost part of code
- ▶ In “pipes” approach the left-to-right order of code matches how we think about the task
  - ▶ First, we start with an object **and then** (`%>%`) we use `filter()` to isolate first-gen students **and then** (`%>%`) we select which variables to print

**Important:** `str()` function helpful for understanding what object is piped in from one function to another

*#object that was "piped" into `select()` from `filter()`*

```
wwlist %>% filter(firstgen == "Y") %>% str()
```

*#object that was created after `select()` function*

```
wwlist %>% filter(firstgen == "Y") %>% select(state, hs_city, sex) %>% str()
```

## Aside: the `count()` function [students work on their own]

`count()` function from `dplyr` package counts the number of obs by group

**Syntax** [see help file for full syntax]

▶ `count(x,...)`

**Arguments** [see help file for full arguments]

▶ `x`: an object, often a data frame

▶ `...`: variables to group by

Examples of using `count()`

▶ Without vars in `...` argument, counts number of obs in object

```
count(wwlist)
wwlist %>% count()
```

▶ With vars in `...` argument, counts number of obs per variable value

▶ note: by default, `count()` always shows `NAs` [this is good!]

```
count(wwlist,school_category)
wwlist %>% count(school_category)
```

## Aside: pipe operators and new lines

Often want to insert line breaks to make long line of code more readable

- ▶ When inserting line breaks, **pipe operator %>%** should be the last thing before a line break, not the first thing after a line break

### This works

```
wwlist %>% filter(firstgen == "Y") %>%
 select(state, hs_city, sex) %>%
 count(sex)
```

### This works too

```
wwlist %>% filter(firstgen == "Y",
 state != "WA") %>%
 select(state, hs_city, sex) %>%
 count(sex)
```

### This doesn't work

```
wwlist %>% filter(firstgen == "Y")
 %>% select(state, hs_city, sex)
 %>% count(sex)
```

# Do task with and without pipes

Task:

- ▶ Count the number “first-generation” prospects from the state of Washington

Without pipes

```
count(filter(wwlist, firstgen == "Y", state == "WA"))
#> # A tibble: 1 x 1
#> n
#> <int>
#> 1 32428
```

With pipes

```
wwlist %>% filter(firstgen == "Y", state == "WA") %>% count()
#> # A tibble: 1 x 1
#> n
#> <int>
#> 1 32428
```



## Do task with and without pipes

**Task:** frequency table of `school_type` for non first-gen prospects from WA

### without pipes

```
wwlist_temp <- filter(wwlist, firstgen == "N", state == "WA")
table(wwlist_temp$school_type, useNA = "always")
#>
#> private public <NA>
#> 11 46146 12489
rm(wwlist_temp) # cuz we don't need after creating table
```

### With pipes

```
wwlist %>% filter(firstgen == "N", state == "WA") %>% count(school_type)
#> # A tibble: 3 x 2
#> school_type n
#> <chr> <int>
#> 1 private 11
#> 2 public 46146
#> 3 <NA> 12489
```

### Comparison of two approaches

- ▶ without pipes, task requires multiple lines of code (this is quite common)
  - ▶ first line creates object; second line analyzes object
- ▶ with pipes, task can be completed in one line of code and you aren't left with objects you don't care about

## Student exercises with pipes

1. Using object `wwlist` select the following variables (`state`, `firstgen`, `ethn_code`) and assign `<-` them to object `wwlist_temp`. (ex. `wwlist_temp <- wwlist`)
2. Using the object you just created `wwlist_temp`, create a frequency table of `ethn_code` for first-gen prospects from California.
3. **Bonus:** Try doing question 1 and 2 together. Use original object `wwlist`, but do not assign to a new object.

Once finished you can `rm(wwlist_temp)`

## Solution to exercises with pipes

1. Using object `wwlist` select the following variables (`state`, `firstgen`, `ethn_code`) and assign them to object `wwlist_temp`

```
wwlist_temp <- wwlist %>%
 select(state, firstgen, ethn_code)
```

## Solution to exercises with pipes

- Using the object you just created `wwlist_temp`, create a frequency table of `ethn_code` for first-gen prospects from California.

```
#names(wwlist)
wwlist_temp %>%
 filter(firstgen == "Y", state == "CA") %>% count(ethn_code)
#> # A tibble: 10 x 2
#> ethn_code n
#> <chr> <int>
#> 1 american indian or alaska native 4
#> 2 asian or native hawaiian or other pacific islander 86
#> 3 black or african american 10
#> 4 cuban 1
#> 5 mexican/mexican american 643
#> 6 not reported 113
#> 7 other-2 or more 4197
#> 8 other spanish/hispanic 179
#> 9 puerto rican 8
#> 10 white 2933
```

## Solution to exercises with pipes

3. **Bonus:** Try doing question 1 and 2 together.

```
wwlist %>%
 select(state, firstgen, ethn_code) %>%
 filter(firstgen == "Y", state == "CA") %>%
 count(ethn_code)
#> # A tibble: 10 x 2
#> ethn_code n
#> <chr> <int>
#> 1 american indian or alaska native 4
#> 2 asian or native hawaiian or other pacific islander 86
#> 3 black or african american 10
#> 4 cuban 1
#> 5 mexican/mexican american 643
#> 6 not reported 113
#> 7 other-2 or more 4197
#> 8 other spanish/hispanic 179
#> 9 puerto rican 8
#> 10 white 2933
#rm(wwlist_temp)

rm(wwlist_temp)
```

Creating variables using mutate (tidyverse approach)

# Our plan for learning how to create new variables

Recall that `dplyr` package within `tidyverse` provide a set of functions that can be described as “verbs”: **subsetting**, **sorting**, and **transforming**

| What we've done                      | Where we're going                                                |
|--------------------------------------|------------------------------------------------------------------|
| <b>Subsetting data</b>               | <b>Transforming data</b>                                         |
| - <code>select()</code> variables    | - <code>mutate()</code> creates new variables                    |
| - <code>filter()</code> observations | - <code>summarize()</code> calculates across rows                |
| <b>Sorting data</b>                  | - <code>group_by()</code> to calculate across rows within groups |
| - <code>arrange()</code>             |                                                                  |

## Today

- ▶ we'll use `mutate()` to create new variables based on calculations across columns within a row

## Next week

- ▶ we'll combine `mutate()` with `summarize()` and `group_by()` to create variables based on calculations across rows

## Create new data frame based on `df_school_all`

Data frame `df_school_all` has one obs per US high school and then variables identifying number of visits by particular universities

```
load(url("https://github.com/ozanj/rclass/raw/master/data/recruiting/recruit_sc
names(df_school_all)
```

```
#> [1] "state_code" "school_type" "necessch"
#> [4] "name" "address" "city"
#> [7] "zip_code" "pct_white" "pct_black"
#> [10] "pct_hispanic" "pct_asian" "pct_amerindian"
#> [13] "pct_other" "num_fr_lunch" "total_students"
#> [16] "num_took_math" "num_prof_math" "num_took_rla"
#> [19] "num_prof_rla" "avgmedian_inc_2564" "latitude"
#> [22] "longitude" "visits_by_196097" "visits_by_186380"
#> [25] "visits_by_215293" "visits_by_201885" "visits_by_181464"
#> [28] "visits_by_139959" "visits_by_218663" "visits_by_100751"
#> [31] "visits_by_199193" "visits_by_110635" "visits_by_110653"
#> [34] "visits_by_126614" "visits_by_155317" "visits_by_106397"
#> [37] "visits_by_149222" "visits_by_166629" "total_visits"
#> [40] "inst_196097" "inst_186380" "inst_215293"
#> [43] "inst_201885" "inst_181464" "inst_139959"
#> [46] "inst_218663" "inst_100751" "inst_199193"
#> [49] "inst_110635" "inst_110653" "inst_126614"
#> [52] "inst_155317" "inst_106397" "inst_149222"
#> [55] "inst_166629"
```



## Create new data frame based on `df_school_all`

Let's create new version of this data frame, called `school_v2`, which we'll use to introduce how to create new variables

```
school_v2 <- df_school_all %>%
 select(-contains("inst_")) %>% # remove vars that start with "inst_"
 rename(
 visits_by_berkeley = visits_by_110635,
 visits_by_boulder = visits_by_126614,
 visits_by_bama = visits_by_100751,
 visits_by_stonybrook = visits_by_196097,
 visits_by_rutgers = visits_by_186380,
 visits_by_pitt = visits_by_215293,
 visits_by_cinci = visits_by_201885,
 visits_by_nebraska = visits_by_181464,
 visits_by_georgia = visits_by_139959,
 visits_by_scarolina = visits_by_218663,
 visits_by_ncstate = visits_by_199193,
 visits_by_irvine = visits_by_110653,
 visits_by_kansas = visits_by_155317,
 visits_by_arkansas = visits_by_106397,
 visits_by_sillinois = visits_by_149222,
 visits_by_umass = visits_by_166629,
 num_took_read = num_took_rla,
 num_prof_read = num_prof_rla,
 med_inc = avgmedian_inc_2564)
```

```
names(school_v2)
```

Introduce mutate() function

## Introduce `mutate()` function

`mutate()` is **tidyverse** approach to creating variables (not **Base R** approach)

### Description of `mutate()`

- ▶ creates new columns (variables) that are functions of existing columns
- ▶ After creating a new variable using `mutate()`, every row of data is retained
- ▶ `mutate()` works best with pipes `%>%`

### Task:

- ▶ Using data frame `school_v2` create new variable that measures the pct of students on free/reduced lunch (output omitted)

```
school_sml <- school_v2 %>% # create new dataset with fewer vars; not necessary
 select(ncesssch, school_type, num_fr_lunch, total_students)
```

```
school_sml %>%
 mutate(pct_fr_lunch = num_fr_lunch/total_students) # create new var
rm(school_sml)
```

# Syntax for `mutate()`

Let's spend a couple minutes looking at help file for `mutate()`

## Usage (i.e., syntax)

- ▶ `mutate(.data, ...)`

## Arguments

- ▶ `.data` : a data frame
  - ▶ if using `mutate()` after pipe operator `%>%`, then this argument can be omitted
    - ▶ Why? Because data frame object to left of `%>%` "piped in" to first argument of `mutate()`
- ▶ `...` : expressions used to create new variables
  - ▶ Can create multiple variables at once

## Value

- ▶ returns an object that contains the original input data frame and new variables that were created by `mutate()`

## Useful functions (i.e., "helper functions")

- ▶ These are standalone functions can be called *within* `mutate()`
  - ▶ e.g., `if_else()`, `recode()`, `case_when()`
- ▶ will show examples of this in subsequent slides

## Introduce `mutate()` function

New variable not retained unless we **assign** `<-` it to an object (existing or new)

`mutate()` **without assignment**

```
school_v2 %>% mutate(pct_fr_lunch = num_fr_lunch/total_students)

names(school_v2)
```

`mutate()` **with assignment**

```
school_v2_temp <- school_v2 %>%
 mutate(pct_fr_lunch = num_fr_lunch/total_students)

names(school_v2_temp)
rm(school_v2_temp)
```

`mutate()` can create multiple variables at once

`mutate()` can create multiple variables at once

```
school_v2 %>%
 mutate(pct_fr_lunch = num_fr_lunch/total_students,
 pct_prof_math= num_prof_math/num_took_math) %>%
 select(num_fr_lunch, total_students, pct_fr_lunch,
 num_prof_math, num_took_math, pct_prof_math)
```

Or we could write code this way:

```
school_v2 %>%
 select(num_fr_lunch, total_students, num_prof_math, num_took_math) %>%
 mutate(pct_fr_lunch = num_fr_lunch/total_students,
 pct_prof_math= num_prof_math/num_took_math)
```

## Student exercise using mutate()

1. Using the object `school_v2`, select the following variables ( `num_prof_math`, `num_took_math`, `num_prof_read`, `num_took_read` ) and create a measure of percent proficient in math `pct_prof_math` and percent proficient in reading `pct_prof_read`.
2. Now using the code for question 1, filter schools where at least 50% of students are proficient in math & reading.
3. If you have time, count the number of schools from question 2.

## Solutions for exercise using mutate()

1. Using the object `school_v2`, select the following variables ( `num_prof_math`, `num_took_math`, `num_prof_read`, `num_took_read` ) and create a measure of percent proficient in math `pct_prof_math` and percent proficient in reading `pct_prof_read`.

```
school_v2 %>%
 select(num_prof_math, num_took_math, num_prof_read, num_took_read) %>%
 mutate(pct_prof_math = num_prof_math/num_took_math,
 pct_prof_read = num_prof_read/num_took_read)

#> # A tibble: 21,301 x 6
#> num_prof_math num_took_math num_prof_read num_took_read pct_prof_math
#> <dbl> <dbl> <dbl> <dbl> <dbl>
#> 1 24.8 146 25.0 147 0.17
#> 2 1.7 17 1.7 17 0.10
#> 3 3.5 14 3.5 14 0.25
#> 4 3 30 3 30 0.1
#> 5 2.8 28 2.8 28 0.10
#> 6 2.5 25 2.4 24 0.1
#> 7 1.55 62 1.55 62 0.025
#> 8 2.1 21 2.2 22 0.1
#> 9 2.3 23 2.3 23 0.10
#> 10 1.9 19 1.9 19 0.10
#> # ... with 21,291 more rows, and 1 more variable: pct_prof_read <dbl>
```



## Solutions for exercise using mutate()

2. Now using the code for question 1, filter schools where at least 50% of students are proficient in math & reading.

```
school_v2 %>%
 select(num_prof_math, num_took_math, num_prof_read, num_took_read) %>%
 mutate(pct_prof_math = num_prof_math/num_took_math,
 pct_prof_read = num_prof_read/num_took_read) %>%
 filter(pct_prof_math >= 0.5 & pct_prof_read >= 0.5)
#> # A tibble: 7,760 x 6
#> num_prof_math num_took_math num_prof_read num_took_read pct_prof_math
#> <dbl> <dbl> <dbl> <dbl> <dbl>
#> 1 135. 260 149. 261 0.520
#> 2 299. 475 418 475 0.63
#> 3 213. 410 332. 410 0.52
#> 4 54.6 105 96.6 105 0.52
#> 5 111. 121 118. 121 0.92
#> 6 1057. 1994 1477. 2204 0.530
#> 7 100. 103 125. 128 0.975
#> 8 56.4 99 84.4 148 0.570
#> 9 445. 586 392. 594 0.76
#> 10 56.0 59 53.1 61 0.95
#> # ... with 7,750 more rows, and 1 more variable: pct_prof_read <dbl>
```

## Solutions for exercise using mutate()

3. If you have time, count the number of schools from question 2.

```
school_v2 %>%
 select(num_prof_math, num_took_math, num_prof_read, num_took_read) %>%
 mutate(pct_prof_math = num_prof_math/num_took_math,
 pct_prof_read = num_prof_read/num_took_read) %>%
 filter(pct_prof_math >= 0.5 & pct_prof_read >= 0.5) %>%
 count()
#> # A tibble: 1 x 1
#> n
#> <int>
#> 1 7760
```

Using ifelse() function within mutate()

## Using `ifelse()` function within `mutate()`

?if\_else

### Description

- ▶ if `condition` `TRUE`, assign a value; if `condition` `FALSE` assign a value

### Usage (i.e., syntax)

- ▶ `if_else(logical condition, true, false, missing = NULL)`

### Arguments

- ▶ `logical condition`: a condition that evaluates to `TRUE` or `FALSE`
- ▶ `true`: value to assign if condition `TRUE`
- ▶ `false`: value to assign if condition `FALSE`

### Value

- ▶ “Where condition is `TRUE`, the matching value from `true`, where it’s `FALSE`, the matching value from `false`, otherwise `NA`.”
- ▶ missing values from “input” var are assigned missing values in “output var”, unless you specify otherwise

**Example:** Create 0/1 indicator of whether got at least one visit from Berkeley

```
school_v2 %>%
 mutate(got_visit_berkeley = ifelse(visits_by_berkeley>0,1,0)) %>%
 count(got_visit_berkeley)
```

`ifelse()` within `mutate()` to create 0/1 indicator variables

We often create dichotomous (0/1) indicator variables of whether something happened (or whether something is TRUE)

- ▶ Variables that are of substantive interest to project
  - ▶ e.g., did student graduate from college
- ▶ Variables that help you investigate data, check quality
  - ▶ e.g., indicator of whether an observation is missing/non-missing for a particular variable

## Using `ifelse()` within `mutate()`

### Task

- ▶ Create 0/1 indicator if school has median income greater than \$100,000

Usually a good idea to investigate “input” variables **before** creating analysis vars

```
str(school_v2$med_inc) # investigate variable type
school_v2 %>% count(med_inc) # frequency count, but this isn't very helpful

school_v2 %>% filter(is.na(med_inc)) %>% count(med_inc)
shows number of obs w/ missing med_inc
```

### Create variable

```
school_v2 %>% select(med_inc) %>%
 mutate(inc_gt_100k= ifelse(med_inc>100000,1,0)) %>%
 count(inc_gt_100k) # note how NA values of med_inc treated

#> # A tibble: 3 x 2
#> inc_gt_100k n
#> <dbl> <int>
#> 1 0 18632
#> 2 1 2045
#> 3 NA 624
```

## Using `ifelse()` function within `mutate()`

### Task

- ▶ Create 0/1 indicator variable `nonmiss_math` which indicates whether school has non-missing values for the variable `num_took_math`

- ▶ note: `num_took_math` refers to number of students at school that took state math proficiency test

Usually a good to investigate “input” variables before creating analysis vars

```
school_v2 %>% count(num_took_math) # this isn't very helpful
school_v2 %>% filter(is.na(num_took_math)) %>% count(num_took_math) # shows num
```

Create variable

```
school_v2 %>% select(num_took_math) %>%
 mutate(nonmiss_math= ifelse(!is.na(num_took_math),1,0)) %>%
 count(nonmiss_math) # note how NA values treated
#> # A tibble: 2 x 2
#> nonmiss_math n
#> <dbl> <int>
#> 1 0 4103
#> 2 1 17198
```

## Student exercises `ifelse()`

1. Using the object `school_v2`, create 0/1 indicator variable `in_state_berkeley` that equals `1` if the high school is in the same state as UC Berkeley (i.e., `state_code=="CA"` ).
2. Create 0/1 indicator `berkeley_and_irvine` of whether a school got at least one visit from UC Berkeley **AND** from UC Irvine.
3. Create 0/1 indicator `berkeley_or_irvine` of whether a school got at least one visit from UC Berkeley **OR** from UC Irvine.



## Exercise ifelse() solutions

1. Using the object `school_v2`, create 0/1 indicator variable `in_state_berkeley` that equals `1` if the high school is in the same state as UC Berkeley (i.e., `state_code=="CA"` ).

```
str(school_v2$state_code) # investigate input variable
school_v2 %>% filter(is.na(state_code)) %>% count() # investigate input var

#Create var
school_v2 %>% mutate(in_state_berkeley=ifelse(state_code=="CA",1,0)) %>%
 count(in_state_berkeley)
```

## Exercise ifelse() solutions

2. Create 0/1 indicator `berkeley_and_irvine` of whether a school got at least one visit from UC Berkeley **AND** from UC Irvine.

```
#investigate input vars
```

```
school_v2 %>% select(visits_by_berkeley, visits_by_irvine) %>% str()
```

```
school_v2 %>% filter(is.na(visits_by_berkeley)) %>% count()
```

```
school_v2 %>% filter(is.na(visits_by_irvine)) %>% count()
```

```
#create variable
```

```
school_v2 %>%
```

```
 mutate(berkeley_and_irvine=ifelse(visits_by_berkeley>0 & visits_by_irvine>0,1
 count(berkeley_and_irvine)
```

## Exercise `ifelse()` solutions

3. Create 0/1 indicator `berkeley_or_irvine` of whether a school got at least one visit from UC Berkeley **OR** from UC Irvine.

```
school_v2 %>%
 mutate(berkeley_or_irvine=ifelse(visits_by_berkeley>0 | visits_by_irvine>0,1,
 count(berkeley_or_irvine)
```

Using `recode()` function within `mutate()`

## Using `recode()` function within `mutate()`

**Description:** Recode values of a variable

**Usage (i.e., syntax)**

▶ `recode(.x, ..., .default = NULL, .missing = NULL)`

**Arguments** [see help file for further details]

- ▶ `.x` A vector (e.g., variable) to modify
- ▶ `...` Specifications for recode, of form `current_value = new_recoded_value`
- ▶ `.default` : If supplied, all values not otherwise matched given this value.
- ▶ `.missing` : If supplied, any missing values in `.x` replaced by this value.

**Example:** Using data frame `wwlist`, create new 0/1 indicator `public_school` from variable `school_type`

```
str(wwlist$school_type)
wwlist %>% count(school_type)

wwlist_temp <- wwlist %>% select(school_type) %>%
 mutate(public_school = recode(school_type, "public" = 1, "private" = 0))

wwlist_temp %>% head(n=10)
str(wwlist_temp$public_school)
wwlist_temp %>% count(public_school)
rm(wwlist_temp)
```

## Using `recode()` function within `mutate()`

Recoding `school_type` could have been accomplished using `if_else()`

- Use `recode()` when new variable has more than two categories

**Task:** Create `school_catv2` based on `school_category` with these categories:

- "regular"; "alternative"; "special"; "vocational"

Investigate input var

```
str(wwlist$school_category)
wwlist %>% count(school_category)
```

Recode

```
wwlist_temp <- wwlist %>% select(school_category) %>%
 mutate(school_catv2 = recode(school_category,
 "Alternative Education School" = "alternative",
 "Alternative/other" = "alternative",
 "Regular elementary or secondary" = "regular",
 "Regular School" = "regular",
 "Special Education School" = "special",
 "Special program emphasis" = "special",
 "Vocational Education School" = "vocational")
)
str(wwlist_temp$school_catv2)
wwlist_temp %>% count(school_catv2)
wwlist %>% count(school_category)
rm(wwlist_temp)
```

## Using `recode()` within `mutate()` [do in pairs/groups]

**Task:** Create `school_catv2` based on `school_category` with these categories:

- ▶ “regular”; “alternative”; “special”; “vocational”
- ▶ This time use the `.missing` argument to recode `NA`s to “unknown”

```
wwlist_temp <- wwlist %>% select(school_category) %>%
 mutate(school_catv2 = recode(school_category,
 "Alternative Education School" = "alternative",
 "Alternative/other" = "alternative",
 "Regular elementary or secondary" = "regular",
 "Regular School" = "regular",
 "Special Education School" = "special",
 "Special program emphasis" = "special",
 "Vocational Education School" = "vocational",
 .missing = "unknown")
)
str(wwlist_temp$school_catv2)
wwlist_temp %>% count(school_catv2)
wwlist %>% count(school_category)
rm(wwlist_temp)
```

## Using `recode()` within `mutate()`

**Task:** Create `school_catv2` based on `school_category` with these categories:

- ▶ “regular”; “alternative”; “special”; “vocational”
- ▶ This time use the `.default` argument to assign the value “regular”

```
wwlist_temp <- wwlist %>% select(school_category) %>%
 mutate(school_catv2 = recode(school_category,
 "Alternative Education School" = "alternative",
 "Alternative/other" = "alternative",
 "Special Education School" = "special",
 "Special program emphasis" = "special",
 "Vocational Education School" = "vocational",
 .default = "regular")
)
str(wwlist_temp$school_catv2)
wwlist_temp %>% count(school_catv2)
wwlist %>% count(school_category)
rm(wwlist_temp)
```



## Using `recode()` within `mutate()`

**Task:** Create `school_catv2` based on `school_category` with these categories:

► This time create a numeric variable rather than character:

► 1 for “regular”; 2 for “alternative”; 3 for “special”; 4 for “vocational”

```
wwlist_temp <- wwlist %>% select(school_category) %>%
 mutate(school_catv2 = recode(school_category,
 "Alternative Education School" = 2,
 "Alternative/other" = 2,
 "Regular elementary or secondary" = 1,
 "Regular School" = 1,
 "Special Education School" = 3,
 "Special program emphasis" = 3,
 "Vocational Education School" = 4)
)
str(wwlist_temp$school_catv2)
wwlist_temp %>% count(school_catv2)
wwlist %>% count(school_category)
rm(wwlist_temp)
```

## Student exercise using `recode()` within `mutate()`

```
load(url("https://github.com/ozanj/rclass/raw/master/data/recruiting/recruit_event_names(df_event)
```

1. Using object `df_event`, assign new object `df_event_temp` and create `event_typev2` based on `event_type` with these categories:
  - ▶ 1 for "2yr college"; 2 for "4yr college"; 3 for "other"; 4 for "private hs"; 5 for "public hs"
2. This time use the `.default` argument to assign the value 5 for "public hs"

## Exercise using `recode()` within `mutate()` solutions

Check input variable

```
names(df_event)
str(df_event$event_type)
df_event %>% count(event_type)
```

## Exercise using `recode()` within `mutate()` solutions

1. Using object `df_event`, assign new object `df_event_temp` and create `event_typev2` based on `event_type` with these categories:

► 1 for "2yr college"; 2 for "4yr college"; 3 for "other"; 4 for "private hs"; 5 for "public hs"

```
df_event_temp <- df_event %>%
 select(event_type) %>%
 mutate(event_typev2 = recode(event_type,
 "2yr college" = 1,
 "4yr college" = 2,
 "other" = 3,
 "private hs" = 4,
 "public hs" = 5)
)
str(df_event_temp$event_typev2)
df_event_temp %>% count(event_typev2)
df_event %>% count(event_type)
```

## Exercise using `recode()` within `mutate()` solutions

2. This time use the `.default` argument to assign the value 5 for “public hs”

```
df_event %>% select(event_type) %>%
 mutate(event_typev2 = recode(event_type,
 "2yr college" = 1,
 "4yr college" = 2,
 "other" = 3,
 "private hs" = 4,
 .default = 5)
)
str(df_event_temp$event_typev2)
df_event_temp %>% count(event_typev2)
df_event %>% count(event_type)
```

Using `case_when()` function within `mutate()`

## Using `case_when()` function within `mutate()`

**Description** Useful when the variable you want to create is more complicated than variables that can be created using `ifelse()` or `recode()`

- ▶ Useful when new variable is a function of multiple “input” variables

**Usage (i.e., syntax):** `case_when(...)`

**Arguments** [from help file; see help file for more details]

- ▶ `...`: A sequence of two-sided formulas.
  - ▶ The left hand side (LHS) determines which values match this case.
    - ▶ LHS must evaluate to a logical vector.
  - ▶ The right hand side (RHS) provides the replacement value.

**Example task:** Using data frame `wwlist` and input vars `state` and `firstgen`, create a 4-category var with following categories:

- ▶ “instate\_firstgen”; “instate\_nonfirstgen”; “outstate\_firstgen”; “outstate\_nonfirstgen”

```
wwlist_temp <- wwlist %>% select(state,firstgen) %>%
 mutate(state_gen = case_when(
 state == "WA" & firstgen == "Y" ~ "instate_firstgen",
 state == "WA" & firstgen == "N" ~ "instate_nonfirstgen",
 state != "WA" & firstgen == "Y" ~ "outstate_firstgen",
 state != "WA" & firstgen == "N" ~ "outstate_nonfirstgen")
)
str(wwlist_temp$state_gen)
wwlist_temp %>% count(state_gen)
```

## Using `case_when()` function within `mutate()`

**Task:** Using data frame `wwlist` and input vars `state` and `firstgen`, create a 4-category var with following categories:

- ▶ "instate\_firstgen"; "instate\_nonfirstgen"; "outstate\_firstgen";  
"outstate\_nonfirstgen"

Let's take a closer look at how values of inputs are coded into values of outputs

```
wwlist %>% select(state,firstgen) %>% str()
count(wwlist,state)
count(wwlist,firstgen)
```

```
wwlist_temp <- wwlist %>% select(state,firstgen) %>%
 mutate(state_gen = case_when(
 state == "WA" & firstgen == "Y" ~ "instate_firstgen",
 state == "WA" & firstgen == "N" ~ "instate_nonfirstgen",
 state != "WA" & firstgen == "Y" ~ "outstate_firstgen",
 state != "WA" & firstgen == "N" ~ "outstate_nonfirstgen")
)
```

```
wwlist_temp %>% count(state_gen)
wwlist_temp %>% filter(is.na(state)) %>% count(state_gen)
wwlist_temp %>% filter(is.na(firstgen)) %>% count(state_gen)
```

**Take-away:** by default var created by `case_when()` equals `NA` for obs where one of the inputs equals `NA`



## Student exercise using `case_when()` within `mutate()`

1. Using the object `school_v2` and input vars `school_type` , and `state_code` , create a 4-category var `state_type` with following categories:
  - ▶ “instate\_public”; “instate\_private”; “outstate\_public”; “outstate\_private”
  - ▶ Note: We are referring to CA as in-state for this example

## Exercise using `case_when()` within `mutate()` solution

### Investigate

```
school_v2 %>% select(state_code,school_type) %>% str()
count(school_v2,state_code)
school_v2 %>% filter(is.na(state_code)) %>% count()

count(school_v2,school_type)
school_v2 %>% filter(is.na(school_type)) %>% count()
```

## Exercise using `case_when()` within `mutate()` solution

1. Using the object `school_v2` and input vars `school_type` , and `state_code` , create a 4-category var `state_type` with following categories:

► "instate\_public"; "instate\_private"; "outstate\_public"; "outstate\_private"

```
school_v2_temp <- school_v2 %>% select(state_code,school_type) %>%
 mutate(state_type = case_when(
 state_code == "CA" & school_type == "public" ~ "instate_public",
 state_code == "CA" & school_type == "private" ~ "instate_private",
 state_code != "CA" & school_type == "public" ~ "outstate_public",
 state_code != "CA" & school_type == "private" ~ "outstate_private")
)
```

```
school_v2_temp %>% count(state_type)
```

```
#> # A tibble: 4 x 2
```

```
#> state_type n
```

```
#> <chr> <int>
```

```
#> 1 instate_private 366
```

```
#> 2 instate_public 1404
```

```
#> 3 outstate_private 3456
```

```
#> 4 outstate_public 16075
```

```
#school_v2_temp %>% filter(is.na(state_code)) %>% count(state_type) #no missing
```

```
#school_v2_temp %>% filter(is.na(school_type)) %>% count(state_type) #no missing
```

## Base R approach to creating new variables

# Base R approach to creating new variables

Subsetting operators `[]` and `$` are used to create new variables and set conditions of the input variables

If creating new variable based on calculation of input variables, basically the tidyverse equivalent of `mutate()` **without** `ifelse()` or `recode()`

▶ Sudo syntax: `df$newvar <- ...`

▶ where ... argument is expression(s)/calculation(s) used to create new variables

**Task:** Create measure of percent of students on free-reduced lunch

## base R approach

```
school_v2_temp<- school_v2 #create copy of dataset; not necessary
school_v2_temp$pct_fr_lunch <-
 school_v2_temp$num_fr_lunch/school_v2_temp$total_students
```

## tidyverse approach (with pipes)

```
school_v2_temp <- school_v2 %>%
 mutate(pct_fr_lunch = num_fr_lunch/total_students)
```

## Base R approach to creating new variables

If creating new variable based on the condition/values of input variables, basically the tidyverse equivalent of `mutate()` **with** `ifelse()` or `recode()`

► Sudo syntax: `df$newvar[logical condition]<- new value`

► `logical condition`: a condition that evaluates to `TRUE` or `FALSE`

## Base R approach to creating new variables

**Task:** Create 0/1 indicator if school has median income greater than \$100k

**tidyverse approach (using pipes)**

```
school_v2_temp %>% select med_inc) %>%
 mutate(inc_gt_100k= ifelse(med_inc>100000,1,0)) %>%
 count(inc_gt_100k) # note how NA values of med_inc treated
#> # A tibble: 3 x 2
#> inc_gt_100k n
#> <dbl> <int>
#> 1 0 18632
#> 2 1 2045
#> 3 NA 624
```

**Base R approach**

```
school_v2_temp$inc_gt_100k<-NA #initialize an empty column with NAs
otherwise you'll get warning
school_v2_temp$inc_gt_100k[school_v2_temp$med_inc>100000] <- 1
school_v2_temp$inc_gt_100k[school_v2_temp$med_inc<=100000] <- 0
count(school_v2_temp, inc_gt_100k)
#> # A tibble: 3 x 2
#> inc_gt_100k n
#> <dbl> <int>
#> 1 0 18632
#> 2 1 2045
#> 3 NA 624
```

## Base R approach to creating new variables

**Task:** Using data frame `wwlist` and input vars `state` and `firstgen`, create a 4-category var with following categories:

- ▶ "instate\_firstgen"; "instate\_nonfirstgen"; "outstate\_firstgen"; "outstate\_nonfirstgen"

### tidyverse approach (using pipes)

```
wwlist_temp <- wwlist %>%
 mutate(state_gen = case_when(
 state == "WA" & firstgen == "Y" ~ "instate_firstgen",
 state == "WA" & firstgen == "N" ~ "instate_nonfirstgen",
 state != "WA" & firstgen == "Y" ~ "outstate_firstgen",
 state != "WA" & firstgen == "N" ~ "outstate_nonfirstgen")
)
str(wwlist_temp$state_gen)
#> chr [1:268396] NA "instate_nonfirstgen" "instate_nonfirstgen" ...
wwlist_temp %>% count(state_gen)
#> # A tibble: 5 x 2
#> state_gen n
#> <chr> <int>
#> 1 instate_firstgen 32428
#> 2 instate_nonfirstgen 58646
#> 3 outstate_firstgen 32606
#> 4 outstate_nonfirstgen 134616
#> 5 <NA> 10100
```



## Base R approach to creating new variables

**Task:** Using data frame `wwlist` and input vars `state` and `firstgen`, create a 4-category var with following categories:

- ▶ “instate\_firstgen”; “instate\_nonfirstgen”; “outstate\_firstgen”;  
“outstate\_nonfirstgen”

### base R approach

```
wwlist_temp <- wwlist
```

```
wwlist_temp$state_gen <- NA
```

```
wwlist_temp$state_gen[wwlist_temp$state == "WA" & wwlist_temp$firstgen == "Y"] <
```

```
wwlist_temp$state_gen[wwlist_temp$state == "WA" & wwlist_temp$firstgen == "N"] <
```

```
wwlist_temp$state_gen[wwlist_temp$state != "WA" & wwlist_temp$firstgen == "Y"] <
```

```
wwlist_temp$state_gen[wwlist_temp$state != "WA" & wwlist_temp$firstgen == "N"] <
```

```
str(wwlist_temp$state_gen)
```

```
#> chr [1:268396] NA "instate_nonfirstgen" "instate_nonfirstgen" ...
```

```
count(wwlist_temp, state_gen)
```

```
#> # A tibble: 5 x 2
```

```
#> state_gen n
```

```
#> <chr> <int>
```

```
#> 1 instate_firstgen 32428
```

```
#> 2 instate_nonfirstgen 58646
```

```
#> 3 outstate_firstgen 32606
```

```
#> 4 outstate_nonfirstgen 134616
```

```
#> 5 <NA> 10100
```