Module 4: Pipes and variable creation

Managing and Manipulating Data Using R

Introduction

Download Module 4 Rmd and knit!

- From the class website, download the module4.Rmd and module4.R files; move the files from the downloads folder to you `HED696C_Rclass/modules/module4` subfolder
- Open R Studio via `HED696C_Rclass.rproj`
- Once in R Studio, go to File » Open File…» Navigate to and click on `module4.Rmd`
- Try to knit `module4.Rmd` to pdf

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 appraoch to creating new variables [Not Required/Optional Reading]

Libraries we will use today

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

- **you must run this code chunk**

```
library(tidyverse)
```

Data for lecture

Data: prospective student lists purchased by Western Washington Univ

The “Student list” business

- Universities identify/target “prospects” (prospective students) by buying “student lists” from College Board/ACT (e.g., $0.51 for each prospective student on list)
- Student lists contain contact info (e.g., address, email, phone number), academic achievement (e.g., PSAT and SAT scores), demographic characteristics
- Student lists used for university recruiting and marketing campaigns
- Universities choose which prospect students to include in lists they purchase by filtering on criteria like zip-code, GPA, test score range, etc.
- This common recruitment tool reinforces bias in admissions practices
  - Acquired these data via Freedom of Information Act (FOIA); our team’s data management skills were key to the success of this project!
  - Washington Post
  - Washington Monthly

```r
#load prospect list data
load(url("https://github.com/ksalazar3/HED696C_RClass/raw/master/data/prospect_
```

Object `wwlist`

- De-identified prospective student list purchased by Western Washington University from College Board

Module 4 data: student lists 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 prospective students
    - e.g., `psat_range`, `state`, `sex`, `ethn_code`
- some vars provide data about the zip-code student lives in
    - e.g., `med_inc`, `pop_total`, `pop_black`
- some vars provide data about the high school the student is enrolled in
    - e.g., `fr_lunch` is number of students on free/reduced lunch
    - note: this is actually terrible data management structure (data at different levels)

```
names(wwlist)
str(wwlist)
```

Pipes

What are "pipes", %>%

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

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

Intuitive mnemonic device for understanding pipes

- whenever you see a pipe `%>%` think of the words "**and then...**"
- Example: `wwlist %>% filter(firstgen == "Y")`
  - in words: start with object `wwlist` **and then** filter for prospective students that identify as first generation college students

Do task with and without pipes

Task:

- Using object `wwlist` print data for "first-generation" prospects (`firstgen == "Y"`)

```r
filter(wwlist, firstgen == "Y") # without pipes
wwlist %>% filter(firstgen == "Y") # with pipes
```

Comparing the two approaches:

- In the "without pipes" approach, the object is the first argument `filter()` function
- In the "pipes" approach, you don't specify the object as the first argument of `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:

```
str(wwlist) # without pipe

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

Do task with and without pipes

Task: Using object `wwlist`, print data for "first-gen" prospects (`firstgen`) for the following selected variables: `state`, `hs_city`, `sex` [output omitted, run each line to see output]

```r
#investigate the "first-gen" var so we know what to filter for...
str(wwlist$firstgen)
typeof(wwlist$firstgen)
table(wwlist$firstgen, useNA = "always")
is.na(wwlist$firstgen)



#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/word 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

Aside: the `count()` function

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

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

Investigate the `state` var so we know what to filter for...

```
#investigate the "first-gen" var so we know what to filter for...
str(wwlist$state)
typeof(wwlist$state)
table(wwlist$state, useNA = "always")
```

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: create a frequency table (use `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`.
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

2. 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 spanish/hispanic                                 179
#>  8 other-2 or more                                       4197
#>  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 spanish/hispanic                               179
#>  8 other-2 or more                                     4197
#>  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
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()`	

Today

- we’ll use `mutate()` to create new variables based on calculations across columns

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/ksalazar3/HED696C_RClass/raw/master/data/recruitin
names(df_school_all)
#>  [1] "state_code"         "school_type"        "ncessch"
#>  [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; rename institution vars for universities

```
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)
  - % of students on FRL = (number of students on FRL/total number of students)*100
  - Sidenote: YOU need to tell R order of operations by using parantheses
- In order to "save" or "keep" this new variable to use at some later point, you need to use the assignment operator
  - You can save/keep the variable by **adding it** to the original object; overwriting the original object *when you only use the mutate() function* will simply add the variable to the original df with all other variables

```
ncol(school_v2)

school_v2_temp <- school_v2 %>%
  mutate(pct_fr_lunch = (num_fr_lunch/total_students)*100)

ncol(school_v2_temp)
```

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)*100)

names(school_v2)
```

`mutate()` **with assignment**

```
school_v2 <- school_v2 %>%
  mutate(pct_fr_lunch = (num_fr_lunch/total_students)*100)

names(school_v2)
```

`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)*100,
         pct_prof_math= (num_prof_math/num_took_math)*100) %>%
  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)*100,
         pct_prof_math= (num_prof_math/num_took_math)*100)
```

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`.

   - `num_took_math` and `num_took_read` are variables with the number of students that took math and reading tests, respectively
   - `num_prof_math` and `num_prof_read` are variables with the number of students that took tests and scored at proficient levels for math and reading, respectively

2. Now add to the same code for question 1 by filtering schools where at least 50% of students are proficient in math **&** reading.

3. Finish the code in question 2 by counting 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)*100,
         pct_prof_read = (num_prof_read/num_took_read)*100)
#> # 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            17
#>  2           1.7            17           1.7            17            10
#>  3           3.5            14           3.5            14            25
#>  4           3              30           3              30            10
#>  5           2.8            28           2.8            28            10
#>  6           2.5            25           2.4            24            10
#>  7           1.55           62           1.55           62             2.5
#>  8           2.1            21           2.2            22            10
#>  9           2.3            23           2.3            23            10
#> 10           1.9            19           1.9            19            10
#> # i 21,291 more rows
#> # i 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)*100,
         pct_prof_read = (num_prof_read/num_took_read)*100) %>%
  filter(pct_prof_math >= 50 & pct_prof_read >= 50)
#> # 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          52
#>  2         299.            475         418             475          63
#>  3         213.            410         332.            410          52
#>  4          54.6           105          96.6           105          52
#>  5         111.            121         118.            121          92
#>  6        1057.           1994        1477.           2204          53
#>  7         100.            103         125.            128          97.5
#>  8          56.4             99          84.4           148          57
#>  9         445.            586         392.            594          76
#> 10          56.0             59          53.1             61          95
#> # i 7,750 more rows
#> # i 1 more variable: pct_prof_read <dbl>
```

Solutions for exercise using mutate()

3. 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)*100,
         pct_prof_read = (num_prof_read/num_took_read)*100) %>%
  filter(pct_prof_math >= 50 & pct_prof_read >= 50) %>%
  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

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

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

```
school_v2 %>% count(visits_by_berkeley)
#> # A tibble: 4 x 2
#>   visits_by_berkeley     n
#>                <int> <int>
#> 1                  0 20732
#> 2                  1   528
#> 3                  2    36
#> 4                  3     5

#option1: create and save variable; check new variable
school_v2 <- school_v2 %>%
  mutate(got_visit_berkeley = ifelse(visits_by_berkeley>0,1,0))

school_v2 %>%
  count(got_visit_berkeley)
#> # A tibble: 2 x 2
#>   got_visit_berkeley     n
#>                <dbl> <int>
#> 1                  0 20732
#> 2                  1   569

#option2: create variable and check new variable [one step, don't overwrite orig
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
                              # because variable is continuous

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

Create variable and check variable all in one step [but never overwrite original df when using `count()`]

```
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 and save variable
school_v2 <- school_v2 %>%
  mutate(in_state_berkeley=ifelse(state_code=="CA",1,0))

#check new variable
school_v2 %>%
  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 and save variable
school_v2 <- school_v2 %>%
  mutate(berkeley_and_irvine =
           ifelse(visits_by_berkeley>0 & visits_by_irvine>0,1,0))

#check variable
school_v2 %>%
  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.

```r
#create and save new variable
school_v2<- school_v2 %>%
  mutate(berkeley_or_irvine=
           ifelse(visits_by_berkeley>0 | visits_by_irvine>0,1,0))

#check new variable
school_v2 %>%
  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`

```r
str(wwlist$school_type) #investigate input var
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()`

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

- "regular"; "alternative"; "special"; "vocational"
- This time use the `.missing` argument to recode `NAs` to "unknown"

```r
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/ksalazar3/HED696C_RClass/raw/master/data/recruitin
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

```r
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”

```r
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”

```r
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 appraoch to creating new variables [Not Required/Optional Reading]

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
```