Social Sciences Intro to Statistics

Week 2.1 Basics of R

Week 2: Apply basic Dplyr functions in R and produce graphs of continuous and categorical variables.

Introduction

Libraries we will use today

"Load" the package we will use today (output omitted)

- **you must run this code chunk**

```
library(tidyverse)
```

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 **%>%**
- The pipe operator **%>%** is created by the **magrittr** package, which is not part of base R
- However, the magrittr package is automatically loaded when you load the tidyverse package

```
?magrittr::`%>%`
```

What are "pipes", %>%

pipe syntax: `LHS %>% RHS`

- `LHS` (refers to "left hand side" of the pipe) is an object or function
- `RHS` (refers to "right hand side" of the pipe) is a function

How pipes work:

- Object created by `LHS` becomes the first argument of the function (`RHS`) to the right of the `%>%` pipe symbol
- Basic code flow: `object %>% function1 %>% function2 %>% function3`

- Output of `some_function1` becomes the input (the first argument) of the function `some_function2` to the right of the `%>%` pipe symbol

Example of using pipes to calculate mean value of atomic vector

```
1:10 # an atomic vector
#>  [1]  1  2  3  4  5  6  7  8  9 10
mean(1:10) # calculate mean without pipes
#> [1] 5.5
1:10 %>% mean() # calculate mean with pipes
#> [1] 5.5
```

- no pipe: (1) write function; (2) data object `1:10` is 1st argument of `mean()`
- pipe: (1) write data object; (2) "pipe" (verb) object as 1st argument of `mean()`

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

getwd()
#load dataset with one obs per recruiting event
load(url("https://github.com/ozanj/rclass/raw/master/data/recruiting/recruit_event_somevar
#load("../../data/recruiting/recruit_event_somevars.Rdata")

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

```
#load prospect list data
load(url("https://github.com/ozanj/rclass/raw/master/data/prospect_list/wwlist_merged.RDat
```

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

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

What are "pipes", %>%

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 **wwlist** **and then** filter first generation students

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

below code in words:

- start with **wwlist** **and then** select a few vars **and then** filter **and then** sort **and then** investigate structre of object

```
wwlist %>% select(firstgen, state, med_inc_zip) %>%
  filter(firstgen == "Y", state == "WA") %>%
  arrange(desc(med_inc_zip)) %>% str()
#> tibble [32,428 x 3] (S3: tbl_df/tbl/data.frame)
#>  $ firstgen   : chr [1:32428] "Y" "Y" "Y" "Y" ...
#>  $ state      : chr [1:32428] "WA" "WA" "WA" "WA" ...
#>  $ med_inc_zip: num [1:32428] 216720 216720 216720 216720 216720 ...
```

More intuition on the pipe operator, %>%

Example: apply "structure" function **str()** to **wwlist** with and without pipes

```r
str(wwlist) # without pipe
wwlist %>% str() # with pipe
```

I use the `str()` when I add new `%>%`; shows what kind of object being piped in

- task: select a few vars from `wwlist`; isolate first-gen students in WA; sort descending by income (output omitted)

```r
wwlist %>% select(firstgen, state, med_inc_zip) %>% str()

wwlist %>% select(firstgen, state, med_inc_zip) %>%
  filter(firstgen == "Y", state == "WA") %>% str()

wwlist %>% select(firstgen, state, med_inc_zip) %>%
  filter(firstgen == "Y", state == "WA") %>%
  arrange(desc(med_inc_zip)) %>% str()
```

Compare data tasks, with and without pipes

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

```r
# 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

Compare data tasks, with and without pipes

Task: Using object `wwlist`, print data for "first-gen" prospects for selected variables

```r
#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

`str()` helpful to understand object 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: `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()
wwlist %>% count() %>% str()
```

- With vars in `...` argument, counts number of obs per variable value
  - This is the best way to create frequency table, better than `table()`
  - note: by default, `count()` always shows `NAs` [this is good!]

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

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)
```

The power of pipes

You might be thinking, "what's the big deal?"

TasK:

- in one line of code, modify `wwlist` and create bar chart that counts number of prospects purchased by race/ethnicity, separately for in-state vs. out-of-state

```
wwlist %>%  filter(is.na(state)==0) %>% # drop obs where variable state missing
  mutate( # create out-of-state indicator; create recoded ethnicity var
    out_state = as_factor(if_else(state != "WA", "out-of-state", "in-state")),
    ethn_race = recode(ethn_code,
      "american indian or alaska native" = "nativeam",
      "asian or native hawaiian or other pacific islander" = "api",
      "black or african american" = "black",
      "cuban" = "latinx",
      "mexican/mexican american" = "latinx",
      "not reported" = "not_reported",
      "other-2 or more" = "multirace",
      "other spanish/hispanic" = "latinx",
      "puerto rican" = "latinx",
      "white" = "white")) %>%
    group_by(out_state) %>% # group_by "in-state" vs. "out-of-state"
    count(ethn_race) %>%  # count of number of prospects purchased by race
    ggplot(aes(x=ethn_race, y=n)) +  # plot
    ylab("number of prospects") + xlab("race/ethnicity") +
    geom_col() +  coord_flip() + facet_wrap(~ out_state)
```

The power of pipes

TasK:

- in one line of code, modify `wwlist` and create bar chart of median income (in zip-code) of prospects purchased by race/ethnicity, separately for in-state vs. out-of-state

```
wwlist %>%  filter(is.na(state)==0) %>% # drop obs where variable state missing
  mutate( # create out-of-state indicator; create recoded ethnicity var
    out_state = as_factor(if_else(state != "WA", "out-of-state", "in-state")),
    ethn_race = recode(ethn_code,
      "american indian or alaska native" = "nativeam",
      "asian or native hawaiian or other pacific islander" = "api",
      "black or african american" = "black",
      "cuban" = "latinx",
      "mexican/mexican american" = "latinx",
      "not reported" = "not_reported",
      "other-2 or more" = "multirace",
```

```
    "other spanish/hispanic" = "latinx",
    "puerto rican" = "latinx",
    "white" = "white")) %>%
  group_by(out_state, ethn_race) %>% # group_by "out-state" and ethnicity
  summarize(avg_inc_zip = mean(med_inc_zip, na.rm = TRUE)) %>% # calculate avg. inc
  ggplot(aes(x=out_state, y=avg_inc_zip)) +
  ylab("avg. income in zip code") + xlab("") +
  geom_col() +  coord_flip() + facet_wrap(~ ethn_race) # plot
```

The power of pipes

Example R script from Ben Skinner, which creates analysis data for Skinner (2018)

- Link to R script

Other relevant links

- Link to Github repository for Skinner (2018)
- Link to published paper
- Link to Skinner's Github page
  - A lot of cool stuff here
- Link to Skinner's personal website
  - A lot of cool stuff here

Which objects and functions are pipeable

Which objects and functions are "pipeable" (i.e., work with pipes)

- function is pipeable if it takes a data object as first argument and returns an object of same type
- In general, doesn't seem to be any limit on which kinds of objects are pipeable (could be atomic vector, list, data frame)

```
# applying pipes to atomic vectors
1:10 %>% mean
#> [1] 5.5
1:10 %>% mean %>% str()
#>  num 5.5
```

But some pipeable functions restrict which kinds of data objects they accept

- In particular, the `dplyr` functions (e.g., `filter`, `arrange`, etc.) expect the first argument to be a data frame.
- `dpylr` functions won't even accept a list as first argument, even though data frames are a particular class of list

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

as.data.frame(wwlist)  %>% str()
as.data.frame(wwlist) %>% filter(firstgen == "Y") %>% str()

as.list(wwlist)  %>% str()
# as.list(wwlist) %>% filter(firstgen == "Y") %>% str() # error
```

Do task with and without pipes [STUDENTS WORK ON THEIR OWN]

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 [STUDENTS WORK ON THEIR OWN]

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

Without pipes

```r
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

```r
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`

```r
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.

```r
#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.

```r
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)
```