Data Science for Public Policy

Aaron R. Williams and Alena Stern - Georgetown University

PPOL 670 | Assignment 02
tidyverse Basics

Due Date: Tuesday, February 1st at 11:59 PM

Deliverable: A single, well-documented `.R` script submitted to Canvas. The `.R` script should be called `assignment02_<NetID>.R`. Each exercise should be clearly labeled and the grader should be able to run each segment of R code.

Rubric

- Each exercise is clearly labeled with a comment. Use `#` to create a comments in your R script.
- Each answer runs successfully and provides a correct answer.
- Attempted exercises will receive more than zero points. Unattempted exercises will receive zero points.

Points: 5 points

Learning and data science are both collaborative practices. We encourage you to discuss class topics and homework topics with each other. However, the work you submit must be your own. A student should never see another student's code or receive explicit coding instructions for a homework problem. Please attend office hours or contact one of the instructors if you need help or clarification.

Plagiarism on homework or projects will be dealt with to the full extent allowed by Georgetown policy (see http://honorcouncil.georgetown.edu).

Setup

1. Create a new folder on your computer called `assignment02`.
2. Open R Studio.
3. In the top right, click "Project: (None)".
4. Click "New Project".
5. Create a new project in the existing directory `assignment02`.
6. In R Studio, open a `.R` script and save it at `assignment02/assignment02_<NetID>.R`.
7. Add your name and NetID as a comment at the top of the .R script.

The next few exercises will work with a data set that contains counts of PhDs awarded by field from 2008 to 2017. The data come from TidyTuesday–a weekly coding project that tackles a new data set each week in R. The documentation and original source are available here.

Exercise 01 (0.5 point)

1. Add `library(tidyverse)` to the top of your script.
2. Duplicate the following code in your script and run the code to download a .csv from this GitHub repo.

```
phd_link <-
  "https://raw.githubusercontent.com/rfordatascience/tidytuesday/master/data/2019/2019-02-19/phd_by_field.csv"

if (!file.exists("phd_by_field.csv")) {

  download.file(phd_link,
                destfile = "phd_by_field.csv")

}
```

3. Load the .csv into R with `read_csv()` from `library(readr)` and assign it to `phds`. Do not use the point-and-click interface in RStudio.
4. Steps 4 through 6 should be connected by pipes (`%>%`). Create a new `tibble` called `phds2017` that only contains observations from 2017.
5. Drop the year column.
6. Sort the tibble from greatest to least based on `n_phds`.

Exercise 02 (0.5 point)

In the next three exercises, we will replace missing values and calculate summary statistics with four different methods using the `phds` data set.

1. Use `count()` to count the number of observations in each year in `phds`.
2. Use `summarize()` to sum `n_phds` for the entire `phds` data set. Use `na.rm` to drop the missing values.
3. Use `group_by()` and `summarize()` to sum `n_phds` by year in `phds`, again using `na.rm` to drop the missing values. Use the pipe (`%>%`) instead of assignment. Rename the complicated variable name created by `summarize()` to `n_phds`. To do this, simply add `n_phds =` inside of `summarize()`.

Exercise 03 (0.5 point)

1. Drop observations from `phds` with missing `n_phd` with `filter()` and assign the results to a new tibble called `phds_no_na`.
2. Using `phds_no_na`, `group_by()`, `summarize()`, and `%>%` to add up the number of PhDs by `broad_field`. You don't need to assign the results.
3. Using `phds_no_na`, `group_by()`, `summarize()`, and `%>%` to add up the number of PhDs by `major_field`. You don't need to assign the results.
4. Using `phds_no_na`, `group_by()`, `summarize()`, and `%>%` to add up the number of PhDs by `field`. You don't need to assign the results.

Exercise 04 (0.5 point)

Note: don't use assignment for either approach. Use pipes and then let the answer print to the console.

1. Use `mutate()` and `if_else()` to create a new variable called `n_phds_no_na`. To create the variable replace `NA` values in `n_phds` with zeros. You will need to come up with `condition` and `true` in `if_else()`. `false` should be `false = n_phds`. Pipe the result into summarize() and sum the number of PhDs in the entire data set.
2. Use `replace_na()` inside of `mutate()` to replace missing `n_phds` with zeros in the `phds` tibble. Pipe the result into summarize() and sum the number of PhDs in the entire data set.

Note: You've now seen four different ways to `summarize()` with missing values. 1. Use `na.rm`. 2. Filter the missing observations. 3. Replace the missing values with zeros with `if_else()`. 4. Replace missing values with zeros with `replace_na()`. Different situations call for different approaches. For instance, replacing `NA` with `0` results in the same answer as dropping `NA` for summation but would result in different answers for mean.

Exercise 05 (0.5 point)

`pivot_longer()` reshapes tibbles so they are longer. `pivot_wider()` reshapes tibbles so they are wider.

1. Explore the documentation for both functions using the `?` operator.
2. Read the pivoting section in R4DS.
3. Examine and duplicate the following code into your script.

```
# create an example tibble
ex5 <- tibble::tribble(
  ~student, ~math_score, ~reading_score,
  "Aaron", 98, 97,
  "Alex", 100, 100
)

# transform from wide to long
ex5_longer <- pivot_longer(
  data = ex5,
  cols = c(math_score, reading_score),
  names_to = "subject",
  values_to = "score"
)

# print the result
ex5_longer
```

```
## # A tibble: 4 x 3
##   student subject       score
##   <chr>   <chr>         <dbl>
## 1 Aaron   math_score       98
## 2 Aaron   reading_score    97
## 3 Alex    math_score      100
## 4 Alex    reading_score   100
```

4. Use `pivot_wider()` to make `ex5_longer` wider. It should be identical to the original `ex5`.

Exercise 06 (1 point)

The next set of exercises will use data about trains in France from TidyTuesday.

"The SNCF (National Society of French Railways) is France's national state-owned railway company. Founded in 1938, it operates the country's national rail traffic along with Monaco, including the TGV, France's high-speed rail network. This dataset covers 2015-2018 with a lot of different train stations. The dataset primarily covers aggregate trip times, delay times, cause for delay, etc for each station - lots of different ways to approach the full_trains.csv dataset with it's 27 columns!"

1. `read_csv()` can read data from a URL if the URL contains a raw csv. Accordingly, the step of downloading the data in exercise 1 was unnecessary. Use `read_csv()` to directly read in this data and assign it to `trains`.
2. The unit of observation for this tibble is year-month-train line. We want to make the data longer so the unit of observation is year-month-train line-delay type. Essentially, we want to turn the column names that begin with "delay" into a character vector called `delay_cause` and the values in those columns into a column titled `delayed_number`. Tips: include `cols = starts_with("delay")` to pick the columns that should pivoted to longer. Assign the result to `trains_long`. It should have 32,772 observations and 23 variables.
3. Pipe the result from subsection 2. to sort `trains_long` by `year`, `month`, and `delay_cause`.
4. Load this data with `read_csv()` and assign it to `trains_small`. Sort the data frame by `year`, `month`, and `delay_cause`.
5. The `trains_long` you created should match `trains_small`. Add `mean(pull(trains_long, delay_cause) == pull(trains_small, delay_cause))` to your script – it should return the value `1`. Perform the same check for `delayed_number` – it should also return the value `1`, but you will need to drop missing values with `na.rm = TRUE`.

Exercise 07 (1.5 points)

We want to recode the values of `month` in the trains tibble from integers to month names (e.g. "January", "February", "March"). We will do this three different ways and then sum the number `total_num_trips` by month.

First approach (Optional)

1. Add the following tibble to your script.

```
months <- tibble::tribble(
  ~month, ~month_name,
  1, "January",
  2, "February",
  3, "March",
  4, "April",
  5, "May",
  6, "June",
  7, "July",
  8, "August",
  9, "September",
  10, "October",
  11, "November",
  12, "December"
)
```

2. `left_join()` the months `months` tibble to `trains`.

3. Pipe (`%>%`), `group_by()` `month_name`, and sum `total_num_trips` with `summarize()`.

Second approach

`case_when()` can be used inside `mutate()` to conditionally code the levels of a variables. This can be used to create new variables or edit existing variables. For example:

```
tribble(
  ~student_id, ~score,
  "101", 83,
  "102", 99,
  "103", 37,
  "104", 78,
  "105", 92
  ) %>%
  mutate(
    grade = case_when(
      score >= 90 ~ "A",
      score >= 80 ~ "B",
      score >= 70 ~ "C",
      score >= 60 ~ "D",
      TRUE ~ "F"
    )
  )
```

```
## # A tibble: 5 x 3
##   student_id score grade
##   <chr>      <dbl> <chr>
## 1 101           83 B
## 2 102           99 A
## 3 103           37 F
## 4 104           78 C
## 5 105           92 A
```

The logical condition goes to the left of `~` and the new value goes to the right of `~`. Conditions are evaluated from top to bottom (i.e. the second condition is really greater than or equal to 80 and less than 90), and `TRUE` can be used at the end as an "otherwise" at the end.

1. Use `case_when()` and `mutate()` to replace integer `month` in `trains` with month names in a new variable called `month_name`. You will need to use `==` for the logical condition.
2. Pipe (`%>%`), `group_by()` `month_name`, and sum `total_num_trips` with `summarize()`.

Third approach

`recode()` can be used to replace values in a vector with a new vector. Consider the following example from the function documentation:

```
char_vec <- sample(c("a", "b", "c"), 10, replace = TRUE)
recode(char_vec, a = "Apple")
recode(char_vec, a = "Apple", b = "Banana")
```

Note: `recode()` operates on a vector and will need to be used inside `mutate()`.

1. Use `recode()` to replace the integers with month names in a variable called `month_names`. Unlike the above example, you will need to wrap the integers in ticks (the key under escape) inside of the

`recode()` function.

2. Pipe (`%>%`), `group_by()` `month_name`, and sum `total_num_trips` with `summarize()`.

The three different approaches have different advantages. `case_when()` is useful because the conditions to the left of `~` can be arbitrarily complex. `left_join()` is useful when the updated variable is stored in a separate tibble. `recode()` is useful when there are a few one-to-one recodes.