Introduction to Social Data Analytics
Class 28

Arushi Kaushik

arkaushi@ucsd.edu

June 4, 2019

Today: `dplyr` package

By the end of today's lecture, you should be able to:

- Download R packages from CRAN (Comprehensive R Archive Network) and load them into your session
- Use the `dplyr` package to accomplish basic data wrangling including:
  - Adding and deleting variables and observations
  - Sorting, merging and reshaping data
  - Collapsing a table to a coarser unit of analysis

Open class28.R if you haven't already.

What is a package?

- A package is a collection of R functions, data, and code that expand the capabilities of base R.
- Anyone can create a package - there's even a package to help with creating packages.

There are more than 14,000 publicly available packages that you can download from the Comprehensive R Archive Network (CRAN).

Using a package requires two steps.

- First, install the package using:

```
install.packages(dplyr)
```

You only need to download a package once. It'll remain on your computer until you uninstall R.

- Second, load the package using:

```
library(dplyr)
```

You must load a package once per session.

Let's create variables with `mutate()`

Create a variable `region$california_ = TRUE` if `region$region == "California"`, `FALSE` otherwise, using:

1. the old fashioned way
2. the dpylr function 'mutate()'

```
# a)
region$california_a <- region$region == "California"
```

Let's create variables with `mutate()`

```
# a)
region <- region %>%
  mutate(california_b = region == "California")

# compare the two:
table(region$california_a, region$california_b)
```

```
##
##           FALSE TRUE
##   FALSE     292    0
##   TRUE        0   28
```

Your turn:

Add the variable `salary$high` =

- 1 if `salary_start_median > 50000`,
- 0 otherwise

using `mutate()`.

Solution:

```
salary <- salary %>%
  mutate(high = as.numeric(salary_start_median > 50000))

salary[6:10, c("salary_start_median", "high")]
```

```
## # A tibble: 5 x 2
##   salary_start_median  high
##                 <dbl> <dbl>
## 1               57200     1
## 2               52600     1
## 3               51100     1
## 4               48600     0
## 5               54800     1
```

Multiple functions at once with piping

One cool aspect of piping is the ability to apply multiple functions at once.

Below we apply two functons to the data frame 'salary':

1. We add a variable salary_thousands = salary_start_median / 1000
2. We rename the variable 'salary' to 'california'

```
salary <- salary %>%
  mutate(salary_thousands = salary_start_median/1000) %>%
  rename(salary_90 = salary_midcareer_90th)
```

It can be helpful to read the %>%s as "and then".

Your turn:

Rename the variable `salary$high` to `salary$salary_high` using rename().

Solution

```
salary <- salary %>% rename(salary_high = high)
```

Keeping certain *observations*

Sometimes we want to subset a data frame that only includes certain observations.

- filter() keeps observations that meet logical criteria
- distinct() removes observations with duplicate values

```
state_schools <- type %>% filter(type == "State")

head(state_schools, 3)
```

```
## # A tibble: 3 x 2
##   name                                type
##   <chr>                               <chr>
## 1 Appalachian State University        State
## 2 Arkansas State University (ASU)     State
## 3 Auburn University                   State
```

Keeping certain *observations*

```
unique_regions <- region %>% distinct(region)

head(unique_regions)
```

```
## # A tibble: 5 x 1
##   region
##   <chr>
## 1 California
## 2 Western
## 3 Midwestern
## 4 Southern
## 5 Northeastern
```

Your turn:

1. Create a table called 'rich.grads' that contains all observations within 'salary' where `salary_90 > 200000`. Use `filter()`.
2. Create a table called 'unique_types' that contains one observation per 'type' in the data frame 'type'. Use `distinct()`.

Solution:

```
rich.grads <- salary %>% filter(salary_90 > 200000)

rich.grads[1:5, c("name", "salary_90")]
```

```
## # A tibble: 5 x 2
##   name                                    salary_90
##   <chr>                                       <dbl>
## 1 Stanford University                        257000
## 2 University of California, Berkeley        201000
## 3 University of Southern California (USC)    201000
## 4 University of California, Davis            202000
## 5 Colorado School of Mines                   201000
```

Solution:

```
unique_types <- type %>% distinct(type)

unique_types
```

```
## # A tibble: 5 x 1
##   type
##   <chr>
## 1 Liberal Arts
## 2 State
## 3 Party
## 4 Ivy League
## 5 Engineering
```

Keeping certain *variables*

- select() keeps only the variables listed
- transmute() keeps only variables listed and allows creation of new variables

```
salary_median <- salary %>%
  select(name, salary_start_median, salary_midcareer_median)

head(salary_median, 3)
```

```
## # A tibble: 3 x 3
##   name                                salary_start_medi~ salar
##   <chr>                                            <dbl>
## 1 Stanford University                              70400
## 2 California Institute of Technol~                 75500
## 3 Harvey Mudd College                              71800
```

Keeping certain variables

```
salary_median_thous <- salary %>%
  transmute(name = name,
      salary_start_thous = salary_start_median/1000,
      salary_midcareer_thous = salary_midcareer_median/1000)

head(salary_median_thous, 3)
```

```
## # A tibble: 3 x 3
##   name                                salary_start_thous sala
##   <chr>                                            <dbl>
## 1 Stanford University                               70.4
## 2 California Institute of Technolo~                 75.5
## 3 Harvey Mudd College                               71.8
```

Your turn:

Overwrite the table 'rich.grads' to only contain the variables:

- `name`
- `salary_90_thous = salary_/1000.`

Solution:

```
rich.grads <- rich.grads %>%
  transmute(name = name, salary_90_thous = salary_90/1000)

head(rich.grads)
```

```
## # A tibble: 6 x 2
##   name                                  salary_90_thous
##   <chr>                                           <dbl>
## 1 Stanford University                               257
## 2 University of California, Berkeley                201
## 3 University of Southern California (USC)           201
## 4 University of California, Davis                   202
## 5 Colorado School of Mines                          201
## 6 University of Notre Dame                          235
```

Sorting data in ascending order

We can sort data using `arrange()`.

```
salary <- salary %>% arrange(salary_start_median)

head(salary, 3) # lowest to highest
```

```
## # A tibble: 3 x 9
##   name  salary_start_me~ salary_midcaree~ salary_midcaree~
##   <chr>            <dbl>            <dbl>            <dbl>
## 1 Lee ~            34500            53900               NA
## 2 Virg~            34600            54900               NA
## 3 More~            34800            60600            34300
## # ... with 4 more variables: salary_midcareer_75th <dbl>, s
## #   salary_high <dbl>, salary_thousands <dbl>
```

Sorting data in descending order

```
salary <- salary %>% arrange(desc(salary_start_median))

head(salary, 3) # highest to lowest
```

```
## # A tibble: 3 x 9
##   name  salary_start_me~ salary_midcaree~ salary_midcaree~
##   <chr>            <dbl>            <dbl>            <dbl>
## 1 Cali~            75500           123000               NA
## 2 Mass~            72200           126000            76800
## 3 Harv~            71800           122000               NA
## # ... with 4 more variables: salary_midcareer_75th <dbl>, s
## #   salary_high <dbl>, salary_thousands <dbl>
```

Your turn:

Sort `rich.grads` to go from highest 90th percentile salary to lowest.

Solution:

```
rich.grads <- rich.grads %>%
  arrange(desc(salary_90_thous))

head(rich.grads) # highest to lowest
```

```
## # A tibble: 6 x 2
##   name                          salary_90_thous
##   <chr>                                   <dbl>
## 1 Yale University                           326
## 2 Dartmouth College                         321
## 3 Harvard University                        288
## 4 University of Pennsylvania                282
## 5 Colgate University                        265
## 6 Princeton University                      261
```

Merging data

We can join data tables using a common identifier, in our case 'name' (note the *left* join).

```
df <- salary %>% left_join(region, by = "name")
```

After that, we will left join 'type':

```
df <- df %>% left_join(type, by = "name")
names(df)
```

```
## [1] "name"                          "salary_start_median"
## [3] "salary_midcareer_median" "salary_midcareer_10th"
## [5] "salary_midcareer_25th"   "salary_midcareer_75th"
## [7] "salary_midcareer_90th"   "region"
## [9] "type"
```

Two merges in one line of code:

```
df <- salary %>%
  left_join(region, by = "name") %>%
  left_join(type, by = "name")

names(df)
```

```
## [1] "name"                      "salary_start_median"
## [3] "salary_midcareer_median" "salary_midcareer_10th"
## [5] "salary_midcareer_25th"   "salary_midcareer_75th"
## [7] "salary_midcareer_90th"   "region"
## [9] "type"
```

Group-wise operations

We can apply functions after using the group_by() function. What's the new unit of analysis in this example?

```
df_region <- df %>%
  group_by(region) %>%
  summarise(mean_start = mean(salary_start_median),
            mean_mid = mean(salary_midcareer_median))

head(df_region, 3)
```

```
## # A tibble: 3 x 3
##   region        mean_start mean_mid
##   <chr>              <dbl>    <dbl>
## 1 California        51032.   93132.
## 2 Midwestern        44225.   78180.
## 3 Northeastern      48496    91352
```

Your turn:

Create `df_type` that is a data frame at the type-level and contains the average starting and mid career salaries by univeristy type.

Solution:

```
df_type <- df %>%
  group_by(type) %>%
  summarise(mean_start = mean(salary_start_median),
            mean_mid = mean(salary_midcareer_median))

head(df_type, 3)
```

```
## # A tibble: 3 x 3
##   type        mean_start mean_mid
##   <chr>            <dbl>    <dbl>
## 1 <NA>            46885.   84106.
## 2 Engineering     59411.  105128.
## 3 Ivy League      60475   120125
```

Summary

class28.R contains an example of reshaping in `R` using another package, `tidyr`. Also check out the data-wrangling cheat sheet on TritonEd.

Here are the commands/operators we covered today:

- `install.packages`
- `library`
- `mutate`
- `rename`
- `filter`
- `distinct`
- `select`
- `transmute`
- `arrange`
- `desc`
- `left_join`
- `group_by`
- `summarise`
- `everything`
- `gather`
- `spread`