tidyr & dplyr

Kam Mix

9/29/2015

Contents

tidyr & dplyr

```
    Manipulating data sets using tidyr and dplyr packages within R Studio to tidy and transform data. I
```

Packages Utilized library(dplyr) library(tidyr)

tidyr: The four fundamental functions of data tidying: 1. gather() takes multiple columns, and gathers them into key-value pairs: it makes "wide" data longer 2. spread() takes two columns (key & value) and spreads in to multiple columns, it makes "long" data wider 3. separate() splits a single column into multiple columns 4. unite() combines multiple columns into a single column

gather( ) function: Objective: Reshaping wide format to long format Description: There are times when our data is considered unstacked and a common attribute of concern is spread out across columns. To reformat the data such that these common attributes are gathered together as a single variable, the gather() function will take multiple columns and collapse them into key-value pairs, duplicating all other columns as needed. Complement to: spread() Function: gather(data, key, value, …, na.rm = FALSE, convert = FALSE) Same as: data %>% gather(key, value, …, na.rm = FALSE, convert = FALSE) Command examples: These all produce the same results DF %>% gather(Quarter, Revenue, Qtr.1:Qtr.4) DF %>% gather(Quarter, Revenue, -Group, -Year) DF %>% gather(Quarter, Revenue, 3:6) DF %>% gather(Quarter, Revenue, Qtr.1, Qtr.2, Qtr.3, Qtr.4)

spread( ) function: Objective: Reshaping long format to wide format Description: There are times when we are required to turn long formatted data into wide formatted data. The spread() function spreads a key-value pair across multiple columns. Complement to: gather() Function: spread(data, key, value, fill = NA, convert = FALSE) Same as: data %>% spread(key, value, fill = NA, convert = FALSE) Command examples: wide_DF <- unite_DF %>% spread(Quarter, Revenue) head(wide_DF, 24)

separate( ) function:

Objective: Splitting a single variable into two

Description: Many times a single column variable will capture multiple variables, or even parts of a variable you just don't care about.

Complement to: unite() Function: separate(data, col, into, sep = " ", remove = TRUE, convert = FALSE) Same as: data %>% separate(col, into, sep =" ", remove = TRUE, convert = FALSE) Command examples: separate_DF <- long_DF %>% separate(Quarter, c("Time_Interval","Interval_ID")) head(separate_DF, 10)

unite( ) function: Objective: Merging two variables into one Description: There may be a time in which we would like to combine the values of two variables. The unite() function is a convenience function to paste together multiple variable values into one. In essence, it combines two variables of a single observation into one variable. Complement to: separate() Function: unite(data, col, …, sep = " ", remove = TRUE) Same as: data %>% unite(col, …, sep =" ", remove = TRUE) Command examples: unite_DF <- separate_DF %>% unite(Quarter, Time_Interval, Interval_ID, sep =".") head(unite_DF, 10)

dplyr: The seven fundamental functions of data transformation: 1. select() selecting variables 2. filter() provides basic filtering capabilities 3. group_by() groups data by categorical levels 4. summarise() summarise data by functions of choice 5. arrange() ordering data 6. join() joining separate dataframes 7. mutate() create new variables

select( ) function: Objective: Reduce dataframe size to only desired variables for current task Description: When working with a sizable dataframe, often we desire to only assess specific variables. The select() function allows you to select and/or rename variables. Function: select(data, ...) Same as: data %>% select(...) Command example: sub.exp <- expenditures %>% select(Division, State, X2007:X2011) head(sub.exp)

filter( ) function: Objective: Reduce rows/observations with matching conditions Description: Filtering data is a common task to identify/select observations in which a particular variable matches a specific value/condition. The filter() function provides this capability. Command example: sub.exp %>% filter(Division == 3)

group_by( ) function: Objective: Group data by categorical variables Description: Often, observations are nested within groups or categories and our goals is to perform statistical analysis both at the observation level and also at the group level. The group_by() function allows us to create these categorical groupings. Function: group_by(data, ...) Same as: data %>% group_by(...) Command example: "The group_by() function is a silent function in which no observable manipulation of the data is performed as a result of applying the function. Rather, the only change you'll notice is, if you print the dataframe you will notice underneath the Source information and prior to the actual dataframe, an indicator of what variable the data is grouped by will be provided. The real magic of the group_by() function comes when we perform summary statistics which we will cover shortly." group.exp <- sub.exp %>% group_by(Division)

summarise( ) function: Objective: Perform summary statistics on variables Description: The summarise() function allows performing the majority of the initial summary statistics when performing exploratory data analysis. Command example: The mean expenditure value across all states in 2011

```
sub.exp %>% summarise(Mean_2011 = mean(X2011))
```

arrange( ) function: Objective: Order variable values Description: View observations in rank order for a particular variable(s). The arrange() function allows ordering data by variables in ascending or descending order. Function: arrange(data, ...) Same as: data %>% arrange(...) Command example: For instance, the summarise example compared the the mean expenditures for each division. Apply the arrange() function at the end to order the divisions from lowest to highest expenditure for 2011. This makes it easier to see the significant differences between Divisions 8,4,1 & 6 as compared to Divisions 5,7,9,3 & 2.

sub.exp %>% group_by(Division)%>% summarise(Mean_2010 = mean(X2010, na.rm=TRUE), Mean_2011 = mean(X2011, na.rm=TRUE)) %>% arrange(Mean_2011)

join( ) functions: Objective: Join two datasets together Description: Often we have separate dataframes that can have common and differing variables for similar observations and we wish to join these dataframes together. The multiple xxx_join() functions provide multiple ways to join dataframes. Function: inner_join(x, y, by = NULL) left_join(x, y, by = NULL) semi_join(x, y, by = NULL) anti_join(x, y, by = NULL) Command example: Our public education expenditure data represents then-year dollars. To make any accurate assessments of longitudinal trends and comparison we need to adjust for inflation. I have the following dataframe which provides inflation adjustment factors for base-year 2012 dollars.

mutate( ) function: Objective: Creates new variables Description: Often we want to create a new variable that is a function of the current variables in our dataframe or even just add a new variable. The mutate() function allows us to add new variables while preserving the existing variables. Function: mutate(data, ...) Same as: data %>% mutate(...) Command example: inflation_adj <- join.exp %>% mutate(Adj_Exp = Expenditure/Inflation) head(inflation_adj)

Works Cited

Boehmke, B. (2015, February 13). Data Processing with dplyr & tidyr. Retrieved September 28, 2015, from RPubs: https://rpubs.com/bradleyboehmke/data_wrangling Grolemund, G. (n.d.). Data Tidying. Retrieved September 26, 2015, from Data Science with R: http://garrettgman.github.io/tidying/