Rebecca Leu

Using the Dplyr and Tidyr Packages

With the gapminder dataset

**Introduction:**

The gapminder dataset consist of 142 observations and 38 variables. There are 142 countries and each list their gdp per capita, life expectancy and population from 1952-2007 in increments of five years.

**Part 1: dplyr: A Grammar of Data Manipulation**

Using the dplyr package we are able to sort, filter, select, reorder and manipulate the data.

**filter()**

Using the filter function we are able to select certain attributes of the data to view. For example, using the code filter(gapminder, lifeExp < 29) we can see if there were ever countries in time that had a life expectancy less than 29. Shockingly there were two, in 1952 Afghanistan’s life expectancy was 28.8, and even more startling, in 1992 Rwanda had a life expectancy of just 23.6. Another example is to look at just Mexico’s data over the years using filter(gapminder, country == "Mexico"). With this we get a tidy dataset with the 12 different observations from 1952-2007. Another way to filter is to use the %in% inside the filter command like so filter(gapminder, country %in% c("Mexico", "Afghanistan")) which will produce a subset of data showing just Mexico and Afghanistan’s data.

**Pipe Operator %>%**

Alternatives to using filter() are indexing gapminder[gapminder$lifeExp < 29, ] and using the subset() verb subset(gapminder, country == "Mexico"). These give the same information as the filter() we used above. Even better we can try the pipe operator %>%. So instead of writing this head(gapminder), we can do this gapminder %>% head using the pipe operator. If we wanted to select a list of just the years and life expectancy’s we could use specify the gapminder dataset, use the pipe operator and then use the select function. gapminder %>% select(year, lifeExp).

This pipe operator can also be used in a chain of commands. If we want to select just the year and life expectancy of the first 4 rows of the gapminder dataset we could chain together these commands like this: gapminder %>% select(year, lifeExp) %>% head(4).

**mutate()**

Using the mutate fuction we can create new columns of data using variables from the other columns. Here we are first filter to Cambodia, then withdraw the continent and life expectancy observations. Then we add in a new column using mutate. For this we have added in the column gdp which is jus the population times the gdp per capita.

gapminder %>%

filter(country == "Cambodia") %>%

select(-continent, -lifeExp) %>%

mutate(gdp = pop \* gdpPercap)

We can also use this same idea to calculate a mean of the gdp of Cambodia only.

gapminder %>%

filter(country == "Cambodia") %>%

select(-continent, -lifeExp) %>%

mutate(gdp = pop \* gdpPercap) %>%

group\_by(country) %>%

summarize(mean\_gdp = mean(gdp)) %>%

ungroup()

In the same fashion we can look at the mean gpd for all countries by leaving out the Cambodia filter. filter(country == "Cambodia") %>%

**Part 2: tidyr: Tidy Messy Data**

**gather()**

Using the gather function in the tidyr package we can reshape the data using key value pairs to form a longer dataset. For instance, we want to create a column of the observed years for the population, life expectancy and gdp per capita and then another column for the values of those year observations. We specify the column names using key = and value =. We only want to see these values, so we take away the continent and country columns. We call this reshaped data frame, gap\_long.

gap\_long <- gapminder\_wide %>%

gather(key = obstype\_year,

value = obs\_values,

-continent, -country)

Using the long format, we can do some calculations with the new data frame.

gap\_long %>%

group\_by(continent, obstype\_year) %>%

summarize(means = mean(obs\_values))

This will give us a mean of the observation type with the values grouped by continent.

**spread()**

Just as we gathered the data into a longer dataset with gather(), we can preform the opposite function with spread(). Here we are taking the dataset we just created named gap\_long and form it back into our original dataset, which we will now name gap\_normal. Using the spread() verb we specify we are spreading the obs\_type and obs\_values colums. We add back in the names of the columns from gapminder.

gap\_normal <- gap\_long %>%

spread(obs\_type, obs\_values) %>%

gap\_normal <- gap\_normal[,names(gapminder)]

Using the function all equal we can confirm that the dataset is now back to normal.

all.equal(gap\_normal,gapminder)

We can also reshape the long gap data into a different type of wide data with the year and observed values of the dataset as the columns.

gap\_wide\_new <- gap\_long %>%

unite(col = var\_names, obs\_type, year, sep = "\_") %>%

spread(key = var\_names, value = obs\_values)

str(gap\_wide\_new)