Basic Data Aggregation with R

Admittedly, data aggregation and manipulation are somewhat artificial distinctions but for the purposes of these lessons I am using them to break apart what we do to data in its existing format (data manipulation, the last lesson) and what we do to more radically change the format of our data through aggregating and summarizing data. We will first introduce the `tidyr` package and then jump back into summarizing data with `dplyr`. To goal of these two lessons are to provide you with enough tools to accomplish the lion's share of data manipulation and aggregation tasks.

Lesson Outline:

- Tidyverse
- `tidyr`
- Using groups to summarize data

Lesson Exercises:

- Exercise 4.1
- Exercise 4.2

Tidyverse

I probably should have brought this up already, but most of what we are learning in this workshop is centered around what is now known as the "Tidyverse". The "tidyverse" is build on a mindset for data analysis that assumes our data is in a data frame and that the rows off that data frame represent observations and the columns represent variables and the cells are values for those observations and variables. In addition, the "tidyverse" is also a mindset for package development that implements this. All of Hadley Wickham's packages that we are using in here (`dplyr`,`tidyr`, `ggplot2`) are part of the "tidyverse." Beyond this, I'm not going to go into too much detail (other than by example), but I did want to provide some links so that you can read up on this if you want.

- Tidy Data Paper: The original paper that laid out the concept. The packages described in the paper are now mostly replaced (e.g. with `dplyr` and `tidyr`), but the concepts remain valid.
- Tidyverse: Website for all things tidy.
- R for Data Science: New book that does a better job of showing how to do data analysis using the "tidyverse" than any other resource. The website is free, but you should buy the book!

note: Just to be clear, I am not getting kick-backs. It's just that the tidyverse has made analysis in R MUCH easier.

tidyr

The `tidyr` package makes it easier to get datasets into a tidy format and largely replaces the `reshape` packages. If you are a spreadsheet person, this will feel a lot like pivot tables. There are several functions in the package, but the only two we will look at are the main ones, `gather()` and `spread()`. Let's try some examples with a simple dataset that comes with `tidyr`, the `smiths` data. You know data about a few people named "Smith."

First, let's add the package and take a quick look at the `smiths` data.

```
library(tidyr)
smiths
```

```
## # A tibble: 2 × 5
##      subject  time   age weight height
##        <chr> <dbl> <dbl>  <dbl>  <dbl>
## 1 John Smith     1    33     90   1.87
## 2 Mary Smith     1    NA     NA   1.54
```

This data is in what I would refer to as a"wide" format becuase it has a column for each variable. It is a tidy dataset but may not be best for the analysis or visualization you want to do Depending on the analysis or the visualization, another format, the "long" format, might be preferrable. As you work more with the various packages in the "tidyverse" finding the proper format becomes easier but will still require some degree of iteration to find what works best. The key with the "tidyverse" packages are that they make it relatively painless to switch between different formats.

Let's take this "wide" format and make it "long" using the `gather` function. With this function we will take the time, age, weight, and height columns and "gather" them into two new columns. These new columns will hold the name of the variable, we will call it "variable" (creative, eh?) and the value of the variable for a given observation. Any guesses what we might call that... I was thinking "value". Here it is in action:

```
smiths_long <- smiths %>%
  gather("variable","value",2:5) %>%
  arrange(subject)
smiths_long
```

```
## # A tibble: 8 × 3
##      subject variable value
##        <chr>    <chr> <dbl>
## 1 John Smith     time  1.00
## 2 John Smith      age 33.00
## 3 John Smith   weight 90.00
## 4 John Smith   height  1.87
## 5 Mary Smith     time  1.00
## 6 Mary Smith      age    NA
## 7 Mary Smith   weight    NA
## 8 Mary Smith   height  1.54
```

This is the `tidyr` function I tend to use the most. I find it quite useful when paired with `ggplot2` becuase the categorical variables are very useful for plotting multiple groups on figures.

The opposite of `gather()` is `spread()`. It takes something in the "long" format and makes it "wide." In other words, `gather()` and `spread()` are complements.

```
smiths_wide <- smiths_long %>%
  spread(variable,value)
smiths_wide
```

```
## # A tibble: 2 × 5
##      subject   age height  time weight
## *      <chr> <dbl>  <dbl> <dbl>  <dbl>
## 1 John Smith    33   1.87     1     90
## 2 Mary Smith    NA   1.54     1     NA
```

Exercise 4.1

Let's practice both of these using the `nla_wq_subset`.

1. Using `spread()` create a new data frame that has the SITE_ID, a column for each EPA_REG, and the CHLA concentration (hint: use both `select()` and `spread()`)

2. Now create another data frame, but this time gather NTL, PTL, and CHLA into "variables" and "values" columns.

Using groups to summarize data

One area where `dplyr` really shines is in aggregating data using groups. We can do this with `group_by()` and `summarize()`.

First, we'll look at an example of grouping a data frame and summarizing the data within those groups.

```
#Chained with Pipes
iris %>%
  group_by(Species)%>%
  summarize(mean_sepal_length = mean(Sepal.Length),
            mean_sepal_width = mean(Sepal.Width),
            mean_petal_length = mean(Petal.Length),
            mean_petal_width = mean(Petal.Width))
```

```
## # A tibble: 3 × 5
##      Species mean_sepal_length mean_sepal_width mean_petal_length
##       <fctr>             <dbl>            <dbl>             <dbl>
## 1     setosa             5.006            3.428             1.462
## 2 versicolor             5.936            2.770             4.260
## 3  virginica             6.588            2.974             5.552
## # ... with 1 more variables: mean_petal_width <dbl>
```

Any function that accepts a vector as input and outputs a single value can be used this way (e.g. `mean`, `median`, `max`, `sd` etc.). One `dplyr` function that is very useful with these grouped summaries is `n()` which returns the number of rows (i.e. samples) in each group. Let's look at `mtcars` for this examples

```
mtcars %>%
  group_by(cyl) %>%
  summarize(mean_mpg = mean(mpg),
            sd_mpg = sd(mpg),
            samp_size = n())
```

```
## # A tibble: 3 × 4
##     cyl mean_mpg   sd_mpg samp_size
##   <dbl>    <dbl>    <dbl>     <int>
## 1     4 26.66364 4.509828        11
## 2     6 19.74286 1.453567         7
## 3     8 15.10000 2.560048        14
```

Let's practice some of these summarizing functions with our NLA data.

Exercise 4.2

Add a new section to our script to calculate the nla water quality means.

1. Use `nla_wq_subset` that we created in the previous lesson.
2. Add some lines to your script to calculate the mean by WSA_ECO9 ecoregions, for TURB, NTL, PTL, and CHLA. Save this to a data frame named `nla_wq_means_eco`.
3. It might be interesting to compare the grouped means to the means of each value for the entire dataset. Using `summarize()`, calculate the mean wq for all lakes (hint: no groups!). Save this as `nla_wq_means`.