Basic Data Wrangling with R

Data wrangling (manipulation, jujitsu, cleaning, etc.) is the part of any data analysis that will take the most time. While it may not necessarily be fun, it is foundational to all the work that follows. For this workshop we are just going to cover the bare essentials of data wrangling in R. We will see how to do this with base R and will practice how to do it with Hadley Wickham's `dplyr` package.

Lesson Outline:

- Indexing vectors
- Indexing lists
- Indexing data frames
- `dplyr`

Lesson Exercises:

- Exercise 3.1
- Exercise 3.2
- Exercise 3.3

Indexing vectors

In base R you can use indexing to select out rows and columns. You will see this quite often in other peoples' code or in other sources for help. So, we should at least cover it so that it isn't foreign when you see it elsewhere.

First lets work with indexing vectors.

```
#Create a vector
x<-c(10:19)
x
```

```
##  [1] 10 11 12 13 14 15 16 17 18 19
```

```
#Positive indexing returns just the value in the ith place
x[7]
```

```
## [1] 16
```

```
#Negative indexing returns all values except the value in the ith place
x[-3]
```

```
## [1] 10 11 13 14 15 16 17 18 19
```

```
#Ranges work too
x[8:10]
```

```
## [1] 17 18 19
```

```
#A vector can be used to index
#Can be numeric
x[c(2,6,10)]
```

```
## [1] 11 15 19
```

```
#Can be boolean - will repeat the pattern
x[c(TRUE,FALSE)]
```

```
## [1] 10 12 14 16 18
```

```
#Can even get fancy
x[x%%2==0]
```

```
## [1] 10 12 14 16 18
```

Indexing lists

Basic indexing of lists isn't too much different than indexing a vector, but remember for our discussion of lists, that you can have any R object stored in a list (e.g., a vector, another list, a data.frame). So to get the item you want can be a bit tricky. A few simple examples.

```
x_list <- list(1:10,letters[1:10])
x_list
```

```
## [[1]]
##  [1]  1  2  3  4  5  6  7  8  9 10
## 
## [[2]]
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j"
```

```
#Get the first item of the list
x_list[[1]]
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
#Or the second item
x_list[[2]]
```

```
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j"
```

```
#And now the letter "g"
x_list[[2]][7]
```

```
## [1] "g"
```

Indexing data frames

You can also index a data frame or select individual columns of a data frame. Since a data frame has two dimensions, you need to specify an index for both the row and the column. You can specify both and get a single value like `data_frame[row,column]`,specify just the row and the get the whole row back like `data_frame[row,]` or get just the column with `data_frame[,column]`. These examples show that.

```
#Let's use one of the stock data frames in R, iris
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```
#And grab a specific value
iris[1,1]
```

```
## [1] 5.1
```

```
#A whole column
petal_len<-iris[,3]
petal_len
```

```
##   [1] 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 1.5 1.6 1.4 1.1 1.2 1.5 1.3
##  [18] 1.4 1.7 1.5 1.7 1.5 1.0 1.7 1.9 1.6 1.6 1.5 1.4 1.6 1.6 1.5 1.5 1.4
##  [35] 1.5 1.2 1.3 1.4 1.3 1.5 1.3 1.3 1.3 1.6 1.9 1.4 1.6 1.4 1.5 1.4 4.7
##  [52] 4.5 4.9 4.0 4.6 4.5 4.7 3.3 4.6 3.9 3.5 4.2 4.0 4.7 3.6 4.4 4.5 4.1
##  [69] 4.5 3.9 4.8 4.0 4.9 4.7 4.3 4.4 4.8 5.0 4.5 3.5 3.8 3.7 3.9 5.1 4.5
##  [86] 4.5 4.7 4.4 4.1 4.0 4.4 4.6 4.0 3.3 4.2 4.2 4.2 4.3 3.0 4.1
##  [ reached getOption("max.print") -- omitted 50 entries ]
```

```
#A row
obs15<-iris[15,]
obs15
```

```
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 15          5.8           4          1.2         0.2  setosa
```

```
#Many rows
obs3to7<-iris[3:7,]
obs3to7
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
## 7          4.6         3.4          1.4         0.3  setosa
```

Also remember that data frames have column names. We can use those too. Let's try it.

```
#First, there are a couple of ways to use the column names
iris$Petal.Length
```

```
##   [1] 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 1.5 1.6 1.4 1.1 1.2 1.5 1.3
##  [18] 1.4 1.7 1.5 1.7 1.5 1.0 1.7 1.9 1.6 1.6 1.5 1.4 1.6 1.6 1.5 1.5 1.4
##  [35] 1.5 1.2 1.3 1.4 1.3 1.5 1.3 1.3 1.3 1.6 1.9 1.4 1.6 1.4 1.5 1.4 4.7
##  [52] 4.5 4.9 4.0 4.6 4.5 4.7 3.3 4.6 3.9 3.5 4.2 4.0 4.7 3.6 4.4 4.5 4.1
##  [69] 4.5 3.9 4.8 4.0 4.9 4.7 4.3 4.4 4.8 5.0 4.5 3.5 3.8 3.7 3.9 5.1 4.5
##  [86] 4.5 4.7 4.4 4.1 4.0 4.4 4.6 4.0 3.3 4.2 4.2 4.2 4.3 3.0 4.1
##  [ reached getOption("max.print") -- omitted 50 entries ]
```

```
head(iris["Petal.Length"])
```

```
##   Petal.Length
## 1          1.4
## 2          1.4
## 3          1.3
## 4          1.5
## 5          1.4
## 6          1.7
```

```
#Multiple colums
head(iris[c("Petal.Length","Species")])
```

```
##   Petal.Length Species
## 1          1.4  setosa
## 2          1.4  setosa
## 3          1.3  setosa
## 4          1.5  setosa
## 5          1.4  setosa
## 6          1.7  setosa
```

```r
#Now we can combine what we have seen to do some more complex queries
#Lets get all the data for Species with a petal length greater than 6
big_iris<-iris[iris$Petal.Length>=6,]
head(big_iris)
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
## 101          6.3         3.3          6.0         2.5 virginica
## 106          7.6         3.0          6.6         2.1 virginica
## 108          7.3         2.9          6.3         1.8 virginica
## 110          7.2         3.6          6.1         2.5 virginica
## 118          7.7         3.8          6.7         2.2 virginica
## 119          7.7         2.6          6.9         2.3 virginica
```

```r
#Or maybe we want just the sepal widths of the virginica species
virginica_iris<-iris$Sepal.Width[iris$Species=="virginica"]
head(virginica_iris)
```

```
## [1] 3.3 2.7 3.0 2.9 3.0 3.0
```

dplyr

The package `dplyr` is a fairly new (2014) package that tries to provide easy tools for the most common data manipulation tasks. It is built to work directly with data frames. The thinking behind it was largely inspired by the package `plyr` which has been in use for some time but suffered from being slow in some cases. `dplyr` addresses this by porting much of the computation to c++. An additional feature is the ability to work with data stored directly in an external database. The benefits of doing this are that the data can be managed natively in a relational database, queries can be conducted on that database, and only the results of the query returned.

This addresses a common problem with R in that all operations are conducted in memory and thus the amount of data you can work with is limited by available memory. The database connections essentially remove that limitation in that you can have a database of many 100s GB, conduct queries on it directly and pull back just what you need for analysis in R. There is a lot of great info on `dplyr`. If you have an interest, i'd encourage you to look more. The vignettes are particulary good.

- `dplyr` GitHub repo
- CRAN page: vignettes here

Using dplyr

So, base R can do what you need, but it is a bit complicated and the syntax is a bit dense. In `dplyr` this can be done with two functions, `select()` and `filter()`. The code can be a bit more verbose, but it allows you to write code that is much more readable. Before we start we need to make sure we've got everything installed and loaded. If you do not have R Version 3.1.2 or greater you will have some problems (i.e. no `dplyr` for you).

```r
install.packages("dplyr")
library("dplyr")
```

I am going to repeat some of what I showed above on data frames but now with `dplyr`. This is what we will be using in the exercises.

```
#First, select some columns
dplyr_sel<-select(iris,Sepal.Length,Petal.Length,Species)
#That's it.  Select one or many columns
#Now select some, like before
dplyr_big_iris<-filter(iris, Petal.Length>=6)
head(dplyr_big_iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
## 1          6.3         3.3          6.0         2.5 virginica
## 2          7.6         3.0          6.6         2.1 virginica
## 3          7.3         2.9          6.3         1.8 virginica
## 4          7.2         3.6          6.1         2.5 virginica
## 5          7.7         3.8          6.7         2.2 virginica
## 6          7.7         2.6          6.9         2.3 virginica
```

```
#Or maybe we want just the virginica species
virginica_iris<-filter(iris,Species=="virginica")
head(virginica_iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
## 1          6.3         3.3          6.0         2.5 virginica
## 2          5.8         2.7          5.1         1.9 virginica
## 3          7.1         3.0          5.9         2.1 virginica
## 4          6.3         2.9          5.6         1.8 virginica
## 5          6.5         3.0          5.8         2.2 virginica
## 6          7.6         3.0          6.6         2.1 virginica
```

But what if I wanted to select and filter? There are three ways to do this: use intermediate steps, nested functions, or pipes. With the intermediate steps, you essentially create a temporary data frame and use that as input to the next function. You can also nest functions (i.e. one function inside of another). This is handy, but can be difficult to read if too many functions are nested as the process from inside out. The last option, pipes, are a fairly recent addition to R. Pipes in the unix/linux world are not new and allow you to chain commands together where the output of one command is the input to the next. This provides a more natural way to read the commands in that they are executed in the way you conceptualize it and make the interpretation of the code a bit easier. Pipes in R look like `%>%` and are made available via th `magrittr` package, which is installed as part of `dplyr`. We will talk a bit about this, but the best desciption, by far, is the secion on pipes in the R For Data Science book.

Let's try all three with the same analysis: selecting out a subset of columns but for only a single species.

```
#Intermediate data frames
#Select First: note the order of the output, neat too!
dplyr_big_iris_tmp1<-select(iris,Species,Sepal.Length,Petal.Length)
dplyr_big_iris_tmp<-filter(dplyr_big_iris_tmp1,Petal.Length>=6)
head(dplyr_big_iris_tmp)
```

```
##     Species Sepal.Length Petal.Length
## 1 virginica          6.3          6.0
## 2 virginica          7.6          6.6
## 3 virginica          7.3          6.3
## 4 virginica          7.2          6.1
## 5 virginica          7.7          6.7
## 6 virginica          7.7          6.9
```

```r
#Nested function
dplyr_big_iris_nest<-filter(select(iris,Species,Sepal.Length,Petal.Length),Species=="virginica")
head(dplyr_big_iris_nest)
```

```
##     Species Sepal.Length Petal.Length
## 1 virginica          6.3          6.0
## 2 virginica          5.8          5.1
## 3 virginica          7.1          5.9
## 4 virginica          6.3          5.6
## 5 virginica          6.5          5.8
## 6 virginica          7.6          6.6
```

```r
#Pipes
dplyr_big_iris_pipe<-select(iris,Species,Sepal.Length,Petal.Length) %>%
  filter(Species=="virginica")
head(dplyr_big_iris_pipe)
```

```
##     Species Sepal.Length Petal.Length
## 1 virginica          6.3          6.0
## 2 virginica          5.8          5.1
## 3 virginica          7.1          5.9
## 4 virginica          6.3          5.6
## 5 virginica          6.5          5.8
## 6 virginica          7.6          6.6
```

Exercise 3.1

This exercise is going to focus on using what we just covered on `dplyr` to start to clean up the National Lakes Assessment data files. Remember to use the stickies: green when you're done, red if you have a problem.

1. If it isn't already open, make sure you have the script we created, "nla_analysis.R" opened up.
2. Start a new section of code in this script by simply putting in a line or two of comments indicating what it is this set of code does.
3. Our goal for this is to create a new data frame that represents a subset of the observations as well as a subset of the columns.
4. We want a selction of columns from the water quality data, `nla_wq`, stored in a new data frame calles `nla_wq_subset`. The columns we want for this are: SITE_ID, VISIT_NO, SITE_TYPE, ST, EPA_REG, LAKE_ORIGIN, WSA_ECO9, TURB, NTL, PTL, and CHLA.
5. Last thing we are going to need to do is get a subset of the observations. We need only the lakes with VISIT_NO equal to 1 and SITE_TYPE equal to "PROB_Lake". Keep the same name, `nla_wq_subset`, for this data frame.

Arrange, select rows, and add columns with dplyr

There are many functions for modifying data in `dplyr` that are useful. Much of what they do, can certainly be accomplished with base R, but not quite as intuitively. Let's run through some examples with `arrange()`, `slice()`, and `mutate()`.

First `arrange()` will re-order a data frame based on the values of a columns. It will take multiple columns and can be in descending or ascending order. I think `iris` is getting a bit tired, let's try a different stock data frame this time: `mtcars`. If you are interested you can try `data()` to see what is available.

```r
#dplyr provides its own object type, tbl that has lots of nice properties
mtcars_tbl <- as.tbl(mtcars)
```

```r
#ascending order is default
arrange(mtcars_tbl,mpg)
```

```
## # A tibble: 32 × 11
##      mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
##    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1   10.4     8 472.0   205  2.93 5.250 17.98     0     0     3     4
## 2   10.4     8 460.0   215  3.00 5.424 17.82     0     0     3     4
## 3   13.3     8 350.0   245  3.73 3.840 15.41     0     0     3     4
## 4   14.3     8 360.0   245  3.21 3.570 15.84     0     0     3     4
## 5   14.7     8 440.0   230  3.23 5.345 17.42     0     0     3     4
## 6   15.0     8 301.0   335  3.54 3.570 14.60     0     1     5     8
## 7   15.2     8 275.8   180  3.07 3.780 18.00     0     0     3     3
## 8   15.2     8 304.0   150  3.15 3.435 17.30     0     0     3     2
##  [ reached getOption("max.print") -- omitted 2 rows ]
## # ... with 22 more rows
```

```r
#descending
arrange(mtcars_tbl,desc(mpg))
```

```
## # A tibble: 32 × 11
##      mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
##    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1   33.9     4  71.1    65  4.22 1.835 19.90     1     1     4     1
## 2   32.4     4  78.7    66  4.08 2.200 19.47     1     1     4     1
## 3   30.4     4  75.7    52  4.93 1.615 18.52     1     1     4     2
## 4   30.4     4  95.1   113  3.77 1.513 16.90     1     1     5     2
## 5   27.3     4  79.0    66  4.08 1.935 18.90     1     1     4     1
## 6   26.0     4 120.3    91  4.43 2.140 16.70     0     1     5     2
## 7   24.4     4 146.7    62  3.69 3.190 20.00     1     0     4     2
## 8   22.8     4 108.0    93  3.85 2.320 18.61     1     1     4     1
##  [ reached getOption("max.print") -- omitted 2 rows ]
## # ... with 22 more rows
```

```r
#multiple columns: most cyl with best mpg at top
arrange(mtcars_tbl,desc(cyl),desc(mpg))
```

```
## # A tibble: 32 × 11
##      mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
##    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1   19.2     8 400.0   175  3.08 3.845 17.05     0     0     3     2
## 2   18.7     8 360.0   175  3.15 3.440 17.02     0     0     3     2
## 3   17.3     8 275.8   180  3.07 3.730 17.60     0     0     3     3
## 4   16.4     8 275.8   180  3.07 4.070 17.40     0     0     3     3
## 5   15.8     8 351.0   264  4.22 3.170 14.50     0     1     5     4
## 6   15.5     8 318.0   150  2.76 3.520 16.87     0     0     3     2
## 7   15.2     8 275.8   180  3.07 3.780 18.00     0     0     3     3
## 8   15.2     8 304.0   150  3.15 3.435 17.30     0     0     3     2
##  [ reached getOption("max.print") -- omitted 2 rows ]
## # ... with 22 more rows
```

Now `slice()` which accomplishes what we did with the numeric indices before. Remembering back to that, we'd could grab rows of the data frame with something like `x[1:3,]`.

```r
#grab rows 3 through 10
slice(mtcars_tbl,3:10)
```

```
## # A tibble: 8 × 11
##     mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
##   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1  22.8     4 108.0    93  3.85 2.320 18.61     1     1     4     1
## 2  21.4     6 258.0   110  3.08 3.215 19.44     1     0     3     1
## 3  18.7     8 360.0   175  3.15 3.440 17.02     0     0     3     2
## 4  18.1     6 225.0   105  2.76 3.460 20.22     1     0     3     1
## 5  14.3     8 360.0   245  3.21 3.570 15.84     0     0     3     4
## 6  24.4     4 146.7    62  3.69 3.190 20.00     1     0     4     2
## 7  22.8     4 140.8    95  3.92 3.150 22.90     1     0     4     2
## 8  19.2     6 167.6   123  3.92 3.440 18.30     1     0     4     4
```

`mutate()` allows us to add new columns based on expressions applied to existing columns

```
mutate(mtcars_tbl,kml=mpg*0.425)
```

```
## # A tibble: 32 × 12
##      mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
##    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1   21.0     6 160.0   110  3.90 2.620 16.46     0     1     4     4
## 2   21.0     6 160.0   110  3.90 2.875 17.02     0     1     4     4
## 3   22.8     4 108.0    93  3.85 2.320 18.61     1     1     4     1
## 4   21.4     6 258.0   110  3.08 3.215 19.44     1     0     3     1
## 5   18.7     8 360.0   175  3.15 3.440 17.02     0     0     3     2
## 6   18.1     6 225.0   105  2.76 3.460 20.22     1     0     3     1
## 7   14.3     8 360.0   245  3.21 3.570 15.84     0     0     3     4
## 8   24.4     4 146.7    62  3.69 3.190 20.00     1     0     4     2
##  [ reached getOption("max.print") -- omitted 2 rows ]
## # ... with 22 more rows, and 1 more variables: kml <dbl>
```

We now have quite a few tools that we can use to clean and manipulate data in R. We have barely touched what both base R and `dplyr` are capable of accomplishing, but hopefully you now have some basics to build on.

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

Exercise 3.2

In this exercise we want to select out the lakes that had the highest and lowest N:P ratio. We can use `mutate`, `arrange`, and `slice` to accomplish this.

1. Add a new section to your script to hold a single, piped workflow to accomplish this
2. Use `mutate` to add a column to your data.frame that contains the N:P ratio.
3. Use `arrange` to order that data.frame
4. Use `slice` to select out the first and last rows (hint: you'll need to figure out how many rows are in your data.frame).