Part 5: Data frames

Introduction to R

This assignment consists of six parts:

- Intro to Basics
- Vectors
- Matrices
- Factors
- Data frames (this document)
- Lists

Create a script called `hw02-5.R` and save it in your `hw02` folder.

After you complete each exercise, commit and push your R script to your remote repo. See Part 0 for instructions. Do *not* push this document.

5.1 Data frames

Recall that amatrix in R is a collection of elements of *one data type* arranged in rows and columns. **Data frames** also have the elements arranged in rows and columns but with one important difference: the elements can be *different* data types.

If you have used a spreadsheet before, then you are familiar with the concept of a data frame. A row has data related to a single instance or observation. A column is a variable with one type of data shared among the observations. Consider these hypothetical plant data:

Genus	Species	Number	Flowering
Asclepias	tuberosa	4	FALSE
Asclepias	viridis	1	TRUE
Baptisia	australis	7	TRUE
Rudbeckia	hirta	2	FALSE

Each row represents the data for a single observation, in this case plant species. Each column represents one variable shared among observations, such as the number of individual plants observed and whether the plants were flowering.

Each row (observation) can have multiple data types, such as text, numeric, *and* logical in this case. Each column (variable) has just one data type, such as text, numeric, *or* logical.

Data frames are the objects that you will use most often (probably) as a scientist. You will most often use data from spreadsheets, which import easily into R as data frames. Data frames are required for most types of analyses and visualization. You will therefore work extensively with data frames throughout this course so *be sure to learn this information well!*

5.1 Instructions

R comes installed with many data sets. To see the list of available data sets, enter `data()` into the console. Scroll through the list. The vast majority of those data sets are data frames. Look for `chickwts`. You may recall that you used data from `chickwts` in the vectors exercise.

Enter `class(chickwts)` in the console. It is `data.frame` class. Do the same for 2-3 other data sets in the list that look interesting to you.

You do not need to include this code in your script.

5.2 Viewing data frames

To view a built-in data set like chickwts, you use the `data()` function to load the data set into memory. You then type the name of the data set to view it. Study this example and run this code in your console. (In most but not all cases, you do not have to use the `data()` function to load the data set but it is a best practice to do so.)

```
data(chickwts)

chickwts
```

The data probably scrolled by quickly in your console. Scroll back up to see the column headers, "weight" and "feed." The unlabeled first column has row numbers automatically added by R when viewed.

Most data sets will be large, often hundreds or thousands of rows (millions of rows in so-called "big data".) You do not want to scroll up hundreds of rows to see the column headers. R provides several functions that allow you to inspect the contents and structure of your data frame.

- `head()` displays the column headers and first five (default) rows of a data frame. You can change the number of rows to display, as you'll see in the example code below.
- `tail()` is similar to `head()` but displays the last rows of the data frame.
- `nrow()` and `ncol()` returns the number of rows or columns of a data frame, respectively. `dim()` displays the number of rows *and* columns (dimensions) of the data frame. Each has its use.
- `str()` displays the structure of your data frame. `str()` displays the class type, the dimensions, the column headers, and the data type of each column. `str()` *is a very handy function.* You will probably use it often.

Study this example code to see the differences. Feel free to run the code in your console (not script).

```
# Use `head()` to show the head of the data frame. Default is first six rows.
head(chickwts)
```

```
##   weight      feed
## 1    179 horsebean
## 2    160 horsebean
## 3    136 horsebean
## 4    227 horsebean
## 5    217 horsebean
## 6    168 horsebean
```

```
# Use head() with a numeric argument to change the number of rows displayed
head(chickwts, n = 4)
```

```
##   weight      feed
## 1    179 horsebean
## 2    160 horsebean
## 3    136 horsebean
## 4    227 horsebean
```

```r
# `tail()` with argument to view last 3 rows
tail(chickwts, n = 3)
```

```
##    weight   feed
## 69    222 casein
## 70    283 casein
## 71    332 casein
```

```r
# `nrow()` displays the number of rows
nrow(chickwts)
```

```
## [1] 71
```

```r
# `ncol()` displays the number of columns
ncol(chickwts)
```

```
## [1] 2
```

```r
length(chickwts) # Another way to get the number of columns
```

```
## [1] 2
```

```r
# dim() returns the number of rows and columns
dim(chickwts)  # Notice result is rows then columns
```

```
## [1] 71  2
```

```r
# `str()` is very useful. You get most of the above info (not tail) in a single, small function.
str(chickwts)
```

```
## 'data.frame':    71 obs. of  2 variables:
##  $ weight: num  179 160 136 227 217 168 108 124 143 140 ...
##  $ feed  : Factor w/ 6 levels "casein","horsebean",..: 2 2 2 2 2 2 2 2 2 2 ...
```

5.2 Instructions

The `iris` data set is another built-in data set. It's much larger than `chickwts`. Add code to your script to do the following

- Use `data()` to load the `iris` data frame.
- Enter `iris` on a line by itself to see the entire data set scroll by. Scroll back to the top to see the column headers. Annoying, isn't it. We'll fix that later.
- Display the first 10 rows of the data frame.
- Display the last rows of the data frame, using the default settings.
- Display the dimensions of the data frame. Use the least amount of code possible.
- Display the struture of the data frame.

```r
# Use `data()` to load the `iris` data frame.


# Enter `iris` on a line by itself to display the full data frame.


# Display the first 10 rows of the data frame.


# Display the last rows of the data frame,
```

```
# Display the dimensions using the least amount of code (9 characters).


# Display the structure of the data frame.


#
```

The `str()` function will often by the first thing you do when you first load or import a data set. *Remember this function!*

5.3 Extracting elements from a data frame

How do you think you would select elements from a data frame? If you thought about square brackets `[]`, then you would be correct. Recall from the matrix exercise the `[r,c]` format to extract data from a specific row and column. Specify the row first, then the column, separated by a column. If you specify only a row or column and leave the other number blank (still with the comma), you will get the contents of the row or column specified.

```
# Extract the 10th row from the first column
chickwts[10,1]


# Extract rows 9-12 of the first two columns.
chickwts[9:12,1:2]

# Dipslay all columns for rows 21-24. Notice the comma is present after the row numbers.
chickwts[21:24,]

# Display all rows for the second column. Notice the comma is present before the column number.
chickwts[,2]
```

You can also use the names of the column headers to extract a column. This is better than using column numbers because you don't have to count the columns to find out what is the number of the column you want to display. Just use it's name.

```
# Display the contents of all rows for the "weight" column.
chickwts[,"weight"]

# Display rows 7-14 for the "feed" column.
chickwts[7:14,"feed"]
```

You will soon see that you will often need to extract one or two columns from a data frame for analysis or visualization. R uses the dollar sign `$` as a short cut. Take a look again at the structure of the `chickwts` data set. Notice the `$` is shown before each variable (column).

```
str(chickwts)
```

```
## 'data.frame':    71 obs. of  2 variables:
##  $ weight: num  179 160 136 227 217 168 108 124 143 140 ...
##  $ feed  : Factor w/ 6 levels "casein","horsebean",..: 2 2 2 2 2 2 2 2 2 2 ...
```

Instead of using `chickwts[,1]` or `chickwts[,"weight"]`, you can use

```
chickwts$weight
```

Type `chickwts$` in your console. As soon as you type `$`, RStudio pops up a list of columns (variables) from the data frame. Just choose the variable you want to display and press enter. **Note: the columns must have names for this to work.** Otherwise, you must use column numbers.

5.3 Instructions

Add code to your script to extract the following information from the `iris` data set.

- Display the 101st row of the `Petal.Length` column, using column numbers.
- Display the first six rows of all columns (mimic the `head()` function without using `head()`).
- Display rows 48-52 of the second column, using the column header name in square brackets.
- Display the entire contents of the `Sepal.Width` column using the `$`.
- **Optional challenge:** Use a combination of dollar sign and square brackets in one line of code to show rows 50 and 51 from the `Species` column. *Hint:* The `$` returns a vector. How would you get the 50th and 51st elements from a vector?

```
# Display the 101st row of the `Petal.Length` column, using column numbers.


# Display the first six rows of all columns (mimic head ())


# Display rows 48-52 of the fourth column, using the column header name in square brackets.


# Display the contents of the `Sepal.Width` column using the `$`


# Optional challenge



#
```

By now, you should be comfortable extracting elements from vectors, matrices, and data frames. If not, you should practice until you do get comfortable.

5.4 Extracting elements with boolean vectors

You can extract data from a data frame using boolean vectors, just like you did for vectors and matrices. Study these examples.

```
# Extract data for chicks raised on meatmeal feed.
meatmeal_chicks <- chickwts$feed == "meatmeal"
chickwts[meatmeal_chicks,]  # Note the comma to get rows
```

```
##    weight     feed
## 49    325 meatmeal
## 50    257 meatmeal
## 51    303 meatmeal
## 52    315 meatmeal
## 53    380 meatmeal
## 54    153 meatmeal
```

```
## 55    263 meatmeal
## 56    242 meatmeal
## 57    206 meatmeal
## 58    344 meatmeal
## 59    258 meatmeal
```

```
# Extract data for all chicks that weigh 325g or more.
# Seems like sunflower produces heavier chicks at 6 weeks
heavy_chicks <- chickwts$weight >= 325
chickwts[heavy_chicks,]
```

```
##    weight      feed
## 26    327   soybean
## 27    329   soybean
## 37    423 sunflower
## 38    340 sunflower
## 39    392 sunflower
## 40    339 sunflower
## 41    341 sunflower
## 45    334 sunflower
## 49    325  meatmeal
## 53    380  meatmeal
## 58    344  meatmeal
## 60    368    casein
## 61    390    casein
## 62    379    casein
## 64    404    casein
## 66    352    casein
## 67    359    casein
## 71    332    casein
```

5.4 Instructions

Add code to your script to extract the following data from the `iris` data set. The instructions for the first two problems especially are somewhat vague because you have to 1) problem-solve and 2) there are at least two possible approaches I can think of to the problem.

- Extract sepal lengths less than or equal to 5.5. Save a vector of the results to a suitably-named variable.
- Apply the `min()` and `max()` functions to the variable you just created to find the minimum and maximum sepal lengths in your extracted data.
- Display rows where sepal width is less than 3.2 *and* species is setosa. Remember `ampersand` is the logical `AND`. You do not have to save results.
- Displays rows where sepal width is less than 2.5 *or* petal width is greater than 2.0. Use the vertical pipe `|` (usually near the right side of a keyboard, perhaps with the backslash key) for logical `OR`. It works like `&` except it finds records that matching one condition *OR* the other condition *OR* both.

```
# Extract rows where sepal length less than or equal to 5.5. Save the result.


# Apply the `min()` and `max()` functions to your result from above.


# Display rows where sepal width is less than 3.2 AND species is setosa.
```

```
# Display rows where sepal width is less than 2.5 OR petal width is greater than 2.0.


#
```

5.5 Use subset to extract data from a data frame

Extracting data from data frames a very common task. You can extract data more quickly using the `subset()` function. Let's use `subset()` to extract the same data from `chickwts` we did earlier. I've shown the code from above, followed by the equivalent result with `subset()`.

```
## Extract data for chicks raised on meatmeal feed.
# meatmeal_chicks <- chickwts$feed == "meatmeal"
# chickwts[meatmeal_chicks,]  # Note the comma to get rows

subset(chickwts, feed == "meatmeal")
```

```
##    weight     feed
## 49    325 meatmeal
## 50    257 meatmeal
## 51    303 meatmeal
## 52    315 meatmeal
## 53    380 meatmeal
## 54    153 meatmeal
## 55    263 meatmeal
## 56    242 meatmeal
## 57    206 meatmeal
## 58    344 meatmeal
## 59    258 meatmeal
```

```
## Extract data for all chicks that weigh 325g or more.
# heavy_chicks <- chickwts$weight >= 325
# chickwts[heavy_chicks,]

subset(chickwts, weight >= 325)
```

```
##    weight      feed
## 26    327   soybean
## 27    329   soybean
## 37    423 sunflower
## 38    340 sunflower
## 39    392 sunflower
## 40    339 sunflower
## 41    341 sunflower
## 45    334 sunflower
## 49    325  meatmeal
## 53    380  meatmeal
## 58    344  meatmeal
## 60    368    casein
## 61    390    casein
## 62    379    casein
## 64    404    casein
## 66    352    casein
## 67    359    casein
```

```
## 71    332    casein
```

5.5 Instructions

Add code to your script to use `subset()` on the `iris` data to display the following. You do not have to save your results to a variable.

- Display rows for petal length between and including 4.0 and 5.0.
- Display rows for sepal length < 5.2 and species is versicolor.

```
# Display rows for petal length between and including 4.0 and 5.0.


# Display rows for sepal length < 5.2 and species is versicolor.


#
```

5.6 Sort

Functions like `min()` and `max()` show you the minimum and maximum values in a data frame column but sorting a column from low to high (or reverse) is often useful. This is accomplished with the `order()` function.

The `order()` does not actually sort the data frame. Instead, `order()` function returns a ranked position for each element in a vector. Study this example carefully. The first element of `some_vector` is `30`, the second element is `10` and the third element is `20`.

```
# Create `some_vector` with three elements
some_vector <- c(30, 10, 20)

# Display the order of the elements in `some_vector`
order(some_vector)
```

```
## [1] 2 3 1
```

Order determines that the second element of `some_vector` (`10`) will be first when sorted because it is lower than the other two elements. Likewise, the third element 3 (`20`) should be second, and the first element (`30`) will be third. However, at this point, `some_vector` is not yet sorted. Here the steps needed to display the sorted vector. Notice the use of the square brackets in the final step.

```
# Create `some_vector` with three elements
some_vector <- c(30, 10, 20)

# Sve the order of the elements in `sort_order` variable
sort_order <- order(some_vector)

# Display the sorted result
some_vector[sort_order]
```

```
## [1] 10 20 30
```

```
# THe original vector is unchanged
some_vector
```

```
## [1] 30 10 20
```

```r
# Permanently sort the vector
some_vector <- some_vector[sort_order]

# Show that `some_vector` has been updated
some_vector
```

```
## [1] 10 20 30
```

You sort data frames the same way, recalling that a variable (column) is just a vector. Enter the following code into your console (not script) to see the results. (I did not want to include the entire output of ordered `chickwts` in this exercise).

```r
# Sort chick weights from lightest to heaviest
sort_order <- order(chickwts$weight)

# Display the order. Show just the "weight" column. How would you show all columns?
chickwts[sort_order,"weight"]

# Use the argument `decreasing = TRUE` to sort from high to low. Show all columns.
sort_order <- order(chickwts$weight, decreasing = TRUE)
chickwts[sort_order,]
```

5.6 Instructions

Add code to your script to do the following with the `iris` data set.

- Order the data frame from shortest to longest sepal length. Display the entire data frame sorted.
- Order the data frame from widest to narrowest petal width (decreasing order). Display *only* the species and petal width columns. Display the species column *first*, before the petal width column. Think through the problem; you have the skills to do this. It is better to use column names rather than numbers.

```r
# Order the data frame from shortest to longest sepal length.


# Display the species and petal width columns in decreasing order of petal width.


#
```

Save to GitHub

Be sure you have used comments throughout your code to identify sections and to describe what you are doing, then commit and push your `hw02-5.R` script to GitHub.

The skills you learned in this exercise are fundamental to working with data frames so be sure you really understand them. However, you will later learn more efficient techniques to extract and sort data frames.