3: Matrices

Introduction to R

This assignment consists of six parts:

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

Create a script called `hw02-3.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.

3.1 What's a matrix?

In R, a matrix is a collection of elements of the same data type (numeric, character, or logical) arranged into a fixed number of rows and columns. Since you are only working with rows and columns, a matrix is called two-dimensional.

You can construct a matrix in R with the `matrix()` function. Consider the following example that creates a matrix called `first_matrix`.

```
first_matrix <- matrix(1:9, byrow = TRUE, nrow = 3)

first_matrix
```

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

- Type `?matrix` in the console and press the enter key. Notice that the help tab in the lower left panel of RStudio describes the `matrix()` function and how to use it. The `?` followed by the function name is a short cut to get help for a function. You can also get help by putting your cursor inside the function name in a script and press the `F1` key (if your keyboard has one).

The information you provide in the parentheses of `matrix()` function are called arguments. For `matrix()`,

- The first argument is a vector of elements that `matrix()` will arrange into the rows and columns. Recall that `1:9` is a shortcut for `c(1, 2, 3, 4, 5, 6, 7, 8, 9)`.
- The argument `byrow = TRUE` tells the `matrix()` function to fill the matrix by rows. Change the argument to `byrow = FALSE` to fill the matrix by columns.
- The `nrow = 3` argument tells `matrix()` that the matrix should have three rows. You can use `nrow` or `ncol`.

You must supply some arguments to a function but other arguments usually have a default. If you look at the help for `matrix()`, the first argument is `data = NA`. That means you must provide the data. Argument `nrow = 1` defaults to creating a matrix with one row. By giving a different value to the argument, such as `nrow = 3`, you override the default.

Instructions

Add code to your script to

- Construct a matrix with 4 rows containing the numbers 11 to 30, filled row-wise.
- Construct a matrix with 5 columns containing the numbers 11 to 30, filled column-wise.

```
# Construct a matrix with 4 rows containing the numbers 11 to 30, filled row-wise.


# Construct a matrix with 5 cols containing the numbers 11 to 30, filled column-wise.



#
```

If your result looks like this, you wrote your code correctly. If it doesn't look like this, edit your code until you get the correct result. Notice the numbers increase across each row when `byrow = TRUE` and down each column when `byrow = FALSE`.

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]   11   12   13   14   15
## [2,]   16   17   18   19   20
## [3,]   21   22   23   24   25
## [4,]   26   27   28   29   30
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]   11   15   19   23   27
## [2,]   12   16   20   24   28
## [3,]   13   17   21   25   29
## [4,]   14   18   22   26   30
```

3.2 Construct a matrix

R contains many built-in data sets. One of them is called `chickwts` that contains the weights of six-week old chickens that were raised on different diets. Enter `chickwts` in the console to see the entire data set. You'll use the first 10 observations from the first three diet types (horsebean, linseed, or soybean) for this set of exercises.

Instructions

Add code to your script to do the following.

- Enter `chick_weights <- chickwts$weight[c(1:20, 23:32)]` to create a vector with the weight data from the data set. You may copy and paste this line to be sure you get the correct data.
- Create a matrix column-wise with three columns and ten rows from the `chick_weights` vector. Save the matrix in a variable called `three_diets`. Creating the matrix column-wise will put chicks raised on the same diet in the same column.
- Display the results stored in the `three_diets` matrix.

```
# Create a column-wise matrix called `three_diets` with three columns and ten rows from the `chick_weig

# Display the values contained in the `three_diets` matrix.

#
```

Your results should look like this.

```
##       [,1] [,2] [,3]
##  [1,]  179  309  243
##  [2,]  160  229  230
##  [3,]  136  181  248
##  [4,]  227  141  327
##  [5,]  217  260  329
##  [6,]  168  203  250
##  [7,]  108  148  193
##  [8,]  124  169  271
##  [9,]  143  213  316
## [10,]  140  257  267
```

3.3 Name the columns and rows

You can name the rows and columns of your matrix just like you named the elements of a vector. Like vectors, accessing individual columns, rows, or elements is easier with names. Study this example that names the rows and columns from the `first_matrix` example above.

```
row_names_vector <- c("Row 1", "Row 2", "Row 3")
col_names_vector <- c("Col 1", "Col 2", "Col 3")

rownames(first_matrix) <- row_names_vector
colnames(first_matrix) <- col_names_vector

first_matrix
```

```
##       Col 1 Col 2 Col 3
## Row 1     1     2     3
## Row 2     4     5     6
## Row 3     7     8     9
```

Your data has 10 chicks ("Replicates", in rows) raised on one of three diets (columns).

Instructions

- Use `colnames()` to name the columns `horsebean`, `linseed`, and `soybean` (the three diets). You can first create a vector with the column names or use them directly with the `c()` function. If necessary, review your options from the Vector exercise.
- Use `rownames()` to name the rows with the pattern `Replicate #` where `#` is the replicate number 1 through 10.

Tip: You could type "Replicate 1", "Replicate 2" and so on, but this gets tedious quickly. Use the following code to take advantage of R's ability to work efficiently with vectors. `paste("Replicate", 1:10)`. This code creates 10 copies of the word "Replicate" followed by the numbers 1 through 10. I'll leave it to you to

figure out how to use the code properly but you have the knowledge. Use `?paste()` to get help on using the `paste()` function.

```
# Use `colnames()` to name the columns `horsebean`, `linseed`, and `soybean`.


# Use `rownames()` to name the rows with the pattern `Replicate #` Use the `paste()` function for effic


#
```

3.4 Calculations on matrices

Like vectors, you can perform some calculations on rows and columns.

- The functions `rowSums()` and `colSums()` returns a vector of sums for each row or column, respectively, in a matrix. The number of elements in the vector depends on the number of rows or columns.
- The functions `rowMeans()` and `colMeans()` returns a vector of averages for each row or column, respectively, in a matrix. The number of elements in the vector depends on the number of rows or columns.

```
rowSums(first_matrix)
```

```
## Row 1 Row 2 Row 3
##     6    15    24
```

```
colMeans(first_matrix)
```

```
## Col 1 Col 2 Col 3
##     4     5     6
```

Instructions

- Calculate the average chick weight for each diet type (columns) in the `three_diets` matrix. Store the result in a vector called `mean_weights`.
- Print the values stored in `mean_weights`.
- Calculate the mean weight of all 30 chicks in the matrix. You can do this at least one of two ways. Do at least one but try to find at least two. You do not have to save this calculation in a variable.

```
# Store the the average chick weight for each diet type in `mean_weights`


# Print the values in mean_weights


# Calculate the mean weight of all 30 chicks in the matrix.

#
```

3.5 Add a column with another diet type.

Sometimes you have to add a new row or column to an existing matrix. R provides the functions `cbind()` and `rbind()` to add (bind) columns and rows to an existing matrix. The first example in this assignment created the `first_matrix` matrix with the values 1 to 9 stored in three rows.

This example uses `rbind()` to add a new row (a vector) with three values.

```
# Review the contents of first_matrix
first_matrix
```

```
##       Col 1 Col 2 Col 3
## Row 1     1     2     3
## Row 2     4     5     6
## Row 3     7     8     9
```

```
# Create a vector with the new data to add
new_row <- c(10, 11, 12)

# Bind the new row
second_matrix <- rbind(first_matrix, new_row)

# Inspect the contents of second_matrix
second_matrix
```

```
##         Col 1 Col 2 Col 3
## Row 1       1     2     3
## Row 2       4     5     6
## Row 3       7     8     9
## new_row    10    11    12
```

Notice that R automatically created a name for the row from the vector name but it doesn't match the other row names. You could name all of the rows again with `rownames(second_matrix) <- c("Row 1", "Row 2", ..., "Row 4")` but this again grows tiresome. You could use the `paste()` trick, too.

Instead, this code allows you to rename a specific row, in this case the fourth row. The row names are effectively a vector of names for each row, so the `[4]` accesses the fourth element and gives it the name.

```
rownames(second_matrix)[4] <- "Row 4"

second_matrix
```

```
##       Col 1 Col 2 Col 3
## Row 1     1     2     3
## Row 2     4     5     6
## Row 3     7     8     9
## Row 4    10    11    12
```

Instructions

If you inspected the built-in `chickwts` data set back in section 3.2, you know that six diet types were tested. Let's add the first 10 chicks from the casein diet and recalculate the mean weights.

- Use `chickwts$weight[60:69]` to access the first 10 chicks raised on casein diet. Save it to a variable of a suitable name.
- Use `cbind()` to add this new column of data to your `three_diets` matrix. Save this new matrix in a variable called `four_diets`. **Hint:** Choosing the variable name wisely in the previous step will save you some coding in the next step. **Work smart, not hard!**
- Be sure the four columns are named "horsebean," "linseed," "soybean," and "casein." If not, write the code to rename the new column (or, edit your code from the first step).
- Calculate the mean weights of each of the four diet types. You do not have to save these results to a variable.

```
# Use `chickwts$weight[60:69]` to access the first 10 chicks raised on casein diet.

# Use `cbind()` to add this new column of data to your `weights` matrix. Save as `four_diets`

# Rename the columns if necessary

# Calculate the mean weights of chicks for each diet type.

#
```

The same logic applies for adding rows to a matrix, which I will leave for you to try on your own.

3.6 Selecting matrix elements

How might you select rows, columns, or individual elements from a matrix? If you thought about using square brackets `[]` like you did with vectors, you are correct. Whereas vectors have one dimension, matrices have two dimensions. You should therefore use a comma to separate the rows you want to select from the columns.

Tip: The key to extract elements from a matrix is to remember the `[r,c]` pattern, where `r` is the row number and `c` is the column number. For example:

- `first_matrix[1,2]` selects the element at the first row and second column.
- `second_matrix[1:3,2:4]` results in a matrix with the data on the rows 1, 2, 3 and columns 2, 3, 4.

If you want to select all elements of a row or a column, no number is needed before or after the comma, respectively:

- `first_matrix[,1]` selects all elements of the first column.
- `second_matrix[1,]` selects all elements of the first row.

You can also use row and column names to select rows, columns, and individual elements.

- `first_matrix[,"Col 2"]` selects all elements of the second column.
- `second_matrix["Row 4",]` selects all elements of the fourth row.
- `second_matrix["Row 3", "Col 1"]` selects the element from the third row, first column.
- You can combine names and numbers such as `first_matrix[2, "Col 2"]` although it is best to be consistent. Remember that using names instead of positions makes your code easier to read.

Instructions

Add code to your script to obtain the following from the `four_diets` matrix.

- Select the entire linseed column by column number.
- Select the entire soybean column by name.
- Select the entire nineth row by row number.
- Select the entire third row by row name.

- Select the fifth replicate from the horsebean column. Use whatever method you which to extract that specific element.

```
# Select the entire linseed column by column number.


# Select the entire soybean column by name.


# Select the entire ninth row by row number.


# Select the entire third row by row name.


# Select the fifth replicate from the horsebean column with any method.


#
```

3.7 A little arithmetic with matrices

Once again, like vectors, you can do math and apply functions to matrices. The basic math operators like `+`, `-`, `*`, `/`, and so work on every element in a matrix.

For example, `3 + first_matrix` adds 3 to each element of `first_matrix`. `second_matrix / 3` divides every element of `second_matrix` by 3.

You can also apply functions, such as `sum` and `mean` to entire matrices.

Instructions

Add code to your script to do the following. Use the weights in `four_diets`. You do not have to save the results, just display them.

- The chick weights are measured in grams. You have a terrible supervisor who wants the results in ounces (gack!). Convert the grams to ounces. Use 0.035 ounces per gram or 28.35 grams per ounce as needed to make the conversion. The conversion values are approximate so one method will not give you exactly the same values as the other method. **You only have to apply one method.**
- Use the `log()` function to calculate the logarithm of the each weight.
- Use the `dim()` (dimension) function to count the total number of rows and columns in your matrix.
- Use the `length()` function to count the total number of chicks weighed.

```
# Convert grams to ounces


# Apply the log() function.


# Apply the dim() function.


# Apply the length() function.
```

```
#
```

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 `hw02-3.R` script to GitHub.