|  |  |
| --- | --- |
| **Follow the instructions. Please save code and any written answers to questions in the same document and submit to Avenue as a PDF file when finished. Please work on the assignment independently before you collaborate with other students or consult the Internet or other resources (like Stack Overflow or ChatGPT) for direct assistance.**  **Learning objectives:**   1. **Basic data structures in R (constants, vectors, matrices, data frames)** 2. **Understand indices for vectors and matrices** 3. **Basic functions: c(), cbind(), log(), hist(), summary(), matrix(), runif(), order(), aggregate(), length(), rnorm(), names()** 4. **Installing packages and loading libraries** 5. **Importing data and elementary data processing** | |
| Start up R studio by clicking on the R Studio icon on your computer. Remember that at the top left window is where you will write all your R code. Ensure that you save the code regularly (using File -> Save, or Ctrl+S shortcut) |  |
| We’ll start with a few mathematical operations in R. Type the code to the right to see how to add two vectors of one element in length. Write and run the code to the right. |  |
| We can add the elements in two vectors of length greater than 1 as well. Note that the resulting vector, c, has three elements that are the sum of corresponding elements in the a and b vectors. Write and run the code to the right. |  |
| Write and run the code to the right. |  |
| **Q1. Write your own code that will add the elements in the vector d to the elements in vector f and put the result in a new vector g.** | |
| Now we will create two vectors and then a matrix, and do some calculating with the columns of the matrix.Type the code to the right into the coding panel. Run the code. |  |
| To return a column or row from a matrix, we use the row-column reference method. To return the first column write and run the code to the right. |  |
| **Q2. Write your own code that will return the entire row 2 of matrix m.** | |
| m is a matrix made up of the two vectors j and k. We can add the elements in the matrix and create a new column. Type the code to the right into the coding panel. Run the code. |  |
| Write and run the code to the right. Notice the error that pops up in the R Studio console window. |  |
| **Q3. In your own words, describe what caused the error (one sentence)** | |
| Write and run the code to the right. Note how we can subset vectors. What does the vector b contain? |  |
| **Q4. Write code to subset this vector to include the values of 5 and 3 and put this in a new vector called c.** | |
| Write and run the code to the right. Note that once the operation in the brackets is completed, the result becomes the input parameter for the log() function. |  |
| We can create a vector full of random values with the following code. This generates a vector with 100 elements ranging in value from 0 to 1. |  |
| We can now perform some operations on these random data. |  |
| **Q5. If you were to compare results for the different students in the class, you would notice that each student would get different answers. Do some looking into the runif() function to see if you can figure out why. How can you change the code so that everyone gets the same answer? Write your reasoning here (up to three sentences).** | |
| Use the code to the right to create a matrix full of random numbers. There are two functions here, one of them is ‘nested’ in the other. The inner runif() function returns numbers between 0 and 1. The only argument included here specifies how many of these numbers to create. The outer matrix() function creates a matrix with a number of columns and rows as specified. Note that the number of random numbers generated (1000) is equal to the size of the matrix (10 by 100). |  |
| Now let’s sort by the first column of the matrix. To do this we use the order() function. This function is important, but a little unintuitive. Write the code to the right. |  |
| Note the values you see in the console. The order function values the location of the ordered location of items/elements in the vector. The first number is 5. This means that the 5th item in the vector v is the first in order. This is what we expect, since ‘0’ is the smallest number. The 3rd item in vector v is the next smallest number, and so on. |  |
| To use order to sort the matrix, write the code to the right.  What’s happening is that the order function is returning the locations of the items in the first column of m as a vector. Then we return all the data to a new matrix (called m\_ordered) in the order specified in this vector. |  |
| Add a ‘-’ sign in front of the reference to the matrix to sort in descending order |  |
| What we’ve learned here is that not only can we specify what rows and columns to return, but also the order they are returned.  **Q6. In no more than 2 sentences, explain what the code to the right is doing.** |  |
| Working with R matrices is similar to how data matrices can be changed and used in more general-purpose programming languages. They are also (relatively) fast data objects compared to some others available in R. We can subset a matrix into two separate matrices using the code to the right. For any value in column 1 that is less than 0.5, it is put into sub1. For any value in column 1 that is greater than or equal to 0.5, put into sub2. |  |
| We can also re-code the values of any column or row in the matrix. What this code is saying is that if the value of an item in column 1 is greater than 0.5, assign it a value of 1, and if the value of an item in column 1 is less than or equal to 0,5, assign it a value of 0. |  |
| **Q7. Modify the code above to recode all items in all columns so that any value greater than 0.3 is coded as 0 and any other value is coded as 1** | |
| People from around the world develop R ‘packages’, which are modules that provide special functionality within R. Installing packages is pretty easy, and can be done many different ways.  Type the code to the right into the coding panel. Run the code.  This should install a package into your version of R. Once a package is installed, you won’t need to reinstall it again. |  |
| Next you have to put the package into your library of activated packages. Type the code to the right into the coding panel. Run the code.  You can now use the package. You will need to add this package to your library every time you start a new R session in which you need to use it. |  |
| The above ‘install.packages()’ function could now be removed from the code, but even better, can be ‘commented out’. Commenting is a way of silencing a line or lines of code so they are not read by the R interpreter, but so you can still see them in your code for reference.  Put a ‘#’ at the start of line ‘1’.  In RStudio, this comment code will make the line green. Now if you run this line, the R interpreter reading your code will ignore it. |  |
| Type the code to the right into the coding panel, but do **not** run it yet.  *Note that line 3 is commented; the R interpreter will ignore this line of code. But the comment helps you remember what the code is for. Use comments to help organize and make sense of the code you write.* |  |
| Copy and paste (or type) the following code directly in between the double quotes (“”) in the code in line 4 above:  <http://www.healthgeomatics.com/wp-content/uploads/2016/05/Canadian-populated-places.csv>  Run the code. It will automatically download the file from the internet and load it into your R session! This works because the read.csv function can read data in \*.csv format (‘comma separated values’). Curl streams data from the Internet onto the computer. The functions are nested so that you first stream the data and then you read it with the read.csv function. If you nested the functions the other way around (curl(read.csv)) it wouldn’t work. | |
| If you look to the right, you’ll notice that a new data object has been created. |  |
| Type the code to the right into the coding panel. Run the code.  This code tests whether the data object we imported is a *data frame*. Data frames are handy data objects in R. **Press Ctrl + S** to save your work. |  |
| If you look in the console, you should see the [1] TRUE after you run the code on line 5. This means that data\_csv is a data frame. |  |
| To see what the data look like, we can click on the little blue circle next to the name of the data. |  |
| This shows us the names of the variables in the data frame, as well as the data types, and examples of the data.  For example, in line 1 is the variable ‘geonameid’, which is an *Integer*. In line 2 is the variable ‘name’ which is a *Factor*. A factor variable is a categorical variable that can be comprised of letters or numbers. |  |
| For factor variables it is sometimes useful to identify the unique names or ‘levels’ they contain. Once you type the code to the right, you create a new character vector containing all the unique place names. |  |
| Type and run the code to the right to count up the number of unique place names. |  |
| **Q8. Write your own code to put the output from the length function above into a new variable called number\_unique**  *Note: it is possible that when the data are imported, the ‘name’ variable is of a type that will give you a null result when using the levels() function. If unique\_names is empty (and length(unique\_names returns 0) then include the code:*  **data\_csv$name <- as.factor(data\_csv$name)**  *before you create the unique\_names variable above.* | |
| We are going to change the “name” variable from one type (Factor) into another (Character). Factor is a better variable type when the variable is categorical. Place names are not categories, but names, so this change is useful. |  |
| Pick the name of any real town or city in Alberta (you can search the Internet to find one). Then modify the code to the right to find out if that town is in the dataframe. Important: replace the word “Abbeydale” with the name of the place you have chosen.  *Note: This code selects all records with the name you chose equal to the variable* name*.* Some place names are in several provinces, so it’s possible that you will get *more than one record. If you get no records in the resulting file, pick another town.* |  |
| Type the code to the right into the coding panel. Run the code. |  |
| This has created a new variable called place\_names |  |
| From this data object we are going to get the length of place names in Canada.  Type the code to the right into the coding panel. Run the code.  *Note: This code does several things. On line 9 the place\_name variable is being changed into a character variable, and then the number of characters in that variable is being counted. The output is a new variable that has the length (in characters) of each Canadian place name. In the second line, we are combining the place names and place name lengths into a new data frame called ‘places’.* |  |
| Now we are going to sort these places from longest to shortest place names.  Type the code to the right into the coding panel. Run the code. |  |
| To look at the data frame, click on the little table icon in the data object panel. Each data frame can be opened by clicking these little tables. |  |
| You should see that the longest place names are at the top of the data file. |  |
| To sort the data from shortest to longest length, type the code to the right. This sorts data in ascending order (by default) by the variable in the order() function. The order function is ‘inside’ the row placeholder of the places dataframe. Note the comma after the order function. This needs to be there in order to know what to do with the columns in the places dataframe; in this case, leaving nothing after the column ensures that all data columns are included in the newly sorted places dataframe. |  |
| Next, we want to find out the most frequently used place name in Canada. One way to do this is to aggregate on place name using a count function, and then sort the resulting file in descending order.  Type the code to the right into the coding panel. Run the code. |  |
| Note If you want to avoid using the data.table package above (or have trouble installing it), here is some alternative code that will also work:    *Note: You may need to rename the variables in the data so they are consistent with the ones generated by the table() function above. Use the code:*  names(aplaces) <- c(“Var1”,”Freq”) *to change variable names* | |
| *Note: The code above installs a new library, activates the library, and then makes a table by counting the unique instance of each place name, then puts the table into a new data frame. If you open the data frame, you’ll see place names and a column of numbers to the right of it.* |  |
| To sort in descending order, type the code to the right into the coding panel. Run the code. Note that the ‘-’ sign in front of the column name sorts the data in descending order. |  |
| **Q9. Write your own code to create a new data frame called most\_common that contains only the first row of the aplaces data frame.** | |
| Now we’re going to generate some random data. Type the code to the right into the coding panel. Run the code.  *Note: The first parameter in this function call is the number of random values generated. The second is the mean of the random numbers generated, and 1 is the standard deviation of the random numbers generated.* |  |
| **Q10. Write your own code to generate another vector of random numbers called rand2 with the same parameters as above.** | |
| We can combine these two vectors of random numbers using the cbind() function. This function combines the vectors as columns into a single matrix. Type the code to the right into the coding panel. Run the code.  *Note: the inner part of the code uses cbind to merge two vectors with the same number of rows. The outer part of the code takes the result of the inner part of the code and turns it into a data frame.* |  |
| Now we’ll create a new variable based on values in one of the columns in the\_data.  Type the code to the right into the coding panel. Run the code. |  |
| Now we’ll try to aggregate the numeric variable rand2 based on values of the reclassified variable r1class. Type the code below into the coding panel. Run the code. | |
| *Note: This takes the mean of the rand2 data for each level (“Negative” and “Positive”) of the new variable (r1class) and puts it into a new data frame.* | |
| Let’s rename the variables in the new data frame. Type the code to the right into the coding panel. Run the code. |  |
| The last thing you’ll do in this assignment is use the apply() function. The apply() function in R is used to apply a function to the margins (rows and columns) of an array or matrix. It's a powerful tool in the R programming language that helps you perform repetitive tasks efficiently, making code more concise and readable. Create a matrix, m, using the code below: | |
| One of the simplest and most obvious uses of the apply function is to use it to sum up column or row totals. The value of ‘1’ indicates row, a value of ‘2’ indicates column. |  |
| You can also sum up column totals. Note that in both cases, the result is a vector the length of the number of rows (when the parameter = ‘1’) and the number of columns (when the parameter equals ‘2’. |  |
| The apply function can also be used with custom functions. Custom functions are useful when R’s existing functions are not helpful. The code to the right is a custom function that returns the the difference between the largest and smallest number in a vector. |  |
| We can use apply() to call this function. |  |
| **Q11. Modify the difference\_max\_min function so that it calculates the difference between the mean of a vector and the minimum value. Use the apply to call the function on the rows of matrix m.** | |
| **Now it’s time to do a final bit of R coding on your own. Write your own code that will do the following:**  **Q12. Import data from:** [**http://www.healthgeomatics.com/wp-content/uploads/2016/06/HockeySeasonData.csv**](http://www.healthgeomatics.com/wp-content/uploads/2016/06/HockeySeasonData.csv)  **Q13. Keep only records for years after 2005**  **Q14. Aggregate the points (PTS) by team (Team), using a sum function, and put result into a data frame called totalpoints**  **Save all the code and written answers (as appropriate) for the questions above into a document. Do not include output from the code, and ensure that the document is nice, clean and readable. Save the document in a pdf file.**  **This assignment is due *before* next class. Please submit on avenue before next week’s class.** | |
|
|
|