Introduction to Data Mathematics Lab 1: R Basics

Navigating RStudio, Data Types and Indexing

Jared Gridley

Lab Overview

In **Lab 1** you will:

- Learn three of R's most important data types: the R vector, matrix, and data frame
- Experiment with several essential vector and matrix operations
- Learn how to **knit** (render) HTML and PDF files from your RMarkdown notebook.

This notebook includes only **Lab** sections; you were expected to complete the separate **Prelab** work before your Wednesday lab session.

Throughout this notebook we have *highlighted in blue* key tasks you should perform. Questions highlighted as *Exercises* will be graded, so be sure you have answered them in your final submission!

After you have completed this notebook. Knit it and upload it to LMS under Lab 1 assignment. Make sure that your name is at the top.

Review: Fetching a Template Notebook from the Course Directory

1. In the RStudio **Files** tab, make sure you are in your `Home` directory, then use the **New Folder** button to create a new folder called `IDM_work` **NOTE:** DO NOT use spaces in file or folder names.

2. The materials you will need for Lab1 have been stored in the `MATP-4400/Lab1/` directory. Navigate there and click on `Lab1.Rmd`.
3. `Lab1.Rmd` is a template notebook that you will edit with answers during the lab period.
    - Once you've opened the notebook, click on RStudio's **File** menu (top left of RStudio)...
    - Edit author:"your name" to be your actual name.

    - Immediately **save** the notebook into the `IDM_work`.
4. Finally, set your R working directory to `IDM_work` by clicking the **More** tab in the file/plot/packages/help/viewer pane, and then select "Set as Working Directory".

Lab 1: Introduction to R Data Types

The following sections provide a hands-on introduction to working with various R data types critical to IDM. Please work through these exercises and make sure you are comfortable with the operations that they demonstrate.

R Data Types

R has four major data types: numeric, integer, logical, and factors. The class() command will let you see the type of an R object.

Numeric data types are floating point numbers.

```r
# assign kSection the value 32.5
kWeight <- 32.5
# identify the data type usoing
class(kWeight)
```

```
## [1] "numeric"
```

The integers data is similar to id contains whole numbers. Note that R defaults to numeric, so we use the ‘as.integer()’ function to foce the type to be integer.

```r
# assign kSection the value 43
kSection <- as.integer(43)
# identify the data type using
class(kSection)
```

```
## [1] "integer"
```

```r
#You can do the usual numeric operations to numeric and integers
kSection*kWeight
```

```
## [1] 1397.5
```

Logical data types have value `TRUE` or ‘FALSE’. They can be the result of an expression

```r
# assign TRUE to Happy
happy <- TRUE
# check out it's class
class(happy)
```

```
## [1] "logical"
```

```r
happy
```

```
## [1] TRUE
```

```r
# here we create a logical by evaluating an expression
result <- 100 > 33
result
```

```
## [1] TRUE
```

Character data types contain characters or strings of characters.

```r
kProfessor <- "Bennett"
class(kProfessor)
```

```
## [1] "character"
```

Factors are the r-objects which are created using a vector. It stores the vector along with the distinct values of the elements in the vector as levels. The levels are always characters irrespective of whether it is numeric, character or logical etc. in the input vector.

Factors are created using the factor() function. The nlevels functions gives the count of distinct levels.

```r
# Create a vector of different colors
apple_colors <- c('green','yellow','red','green')

# Create a factor object.
factor_apple <- factor(apple_colors)

# Print the factor.
print(factor_apple)
```

```
## [1] green  yellow red    green
## Levels: green red yellow
```

```
print(nlevels(factor_apple))
```

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

TRY IT: In the console, experiment with creating different objects. Create a numeric, logical, and factor object. Use 'class()' to check that you got the object that you desired.

```
#numberic
age <- 19
print(class(age))
```

```
## [1] "numeric"
```

```
year <- as.integer(2021)
class(year)
```

```
## [1] "integer"
```

```
is_Jan <- FALSE
is_equal <- 100 == 13
class(is_Jan)
```

```
## [1] "logical"
```

```
class(is_equal)
```

```
## [1] "logical"
```

```
miles <- c(4, 10, 12.5, 9, 10, 8, 16)
miles_factor <- factor(miles)
class(miles_factor)
```

```
## [1] "factor"
```

R Vectors and Matrices

Use R's `c()` function to create a vector

R vectors can hold any type of object as long all the objects in the R list have the same type. For the time being we will focus on *numeric* vector.

a. Create a `numeric` vector with a length of twelve and whose elements are consecutive integers beginning with 1; call this vector `numbers`.
b. Use the `length()` function to get the length of `numbers`; *print the result*.

```
# a
numbers <- c(1,2,3,4,5,6,7,8,9,10,11,12)
# b
length(numbers)
```

```
## [1] 12
```

TRY IT: Try creating a vector named myvec containing -1 ,4, -4, 6 and 10 in the console. Check its length.

```
myvec <- c(-1,4,-4,6,10)

length(myvec)
```

```
## [1] 5
```

R functions for simple statistics

We easily can obtain summary statistics about vectors in R. Let's find out things about our `numbers` vector

```r
# Number of elements
length(numbers)
```

```
## [1] 12
```

```r
# Minimum value
min(numbers)
```

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

```r
# Maximum
max(numbers)
```

```
## [1] 12
```

```r
# Mean (average)
mean(numbers)
```

```
## [1] 6.5
```

```r
# Median
median(numbers)
```

```
## [1] 6.5
```

```r
# Standard deviatiom
sd(numbers)
```

```
## [1] 3.605551
```

The `summary()` function does all of this at once for us!

```r
summary(numbers)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1.00    3.75    6.50    6.50    9.25   12.00
```

TRY IT: In the console, compute the mean, sd, and summary of myvec.

```r
mean(myvec)
```

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

```r
sd(myvec)
```

```
## [1] 5.567764
```

```r
summary(myvec)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##      -4      -1       4       3       6      10
```

Use R's `matrix()` function to create a matrix

R's `matrix()` function takes as arguments `data`, `nrow`, and `ncol`, where…

- `data` is an R vector
- `nrow` and `ncol` are positive integers specifying the number of rows and columns, respectively.

NOTE: In the R console you may use the `?` operator to display documentation for any function of your choosing. Try it out now by running the command `?matrix`.

Let's create a matrix with two columns (call it `numbermat`) using `numbers` as the argument for `data`; *Print the matrix* to verify it has the correct dimensions. We use the 'str()' function to quickly check we got the desired matrix.

```
# create a matrix with 2 columns
numbermat <- matrix(data=numbers, ncol=2)
# check that is has the right size
str(numbermat)
```

```
##  num [1:6, 1:2] 1 2 3 4 5 6 7 8 9 10 ...
```

All elements in R vectors and matrices must have the same class

a. Create an R vector whose first element is the letter `a` and whose second element is the number `3`.

b. *What do you observe?* Use the `class()` function to answer this question

```
my_R_vec <- c('a', 3)

class(my_R_vec)
```

```
## [1] "character"
```

WARNING. You will see that R made the type that it thinks you wanted. R frequently changes data types in operations. So get used to checking your data types.

Use R's `[]` operator to index a vector or Matrix

- Indexing in R always starts with `1`!!
- The syntax for retrieving the third element of 'numbers`would be`numbers[3]'
- You may also provide sets of indices by including them in the square brackets as numeric R vectors. For instance, the command to access the first and third elements of `numbers` would be `numbers[c(1,3)]`.

```
numbers[3]
```

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

```
numbers[c(1,3)]
```

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

- Usually when indexing into a matrix or data frame, you'll use a *comma* within the square brackets to separate the indices of each dimension that you would like to access.

Let's create an R vector whose elements are the top left and bottom left elements of `numbermat`; *print the result.*

```
# Select the elements
e1 <- numbermat[1,1]
e2 <- numbermat[6,1]

# copy these into a new vector and view it
e1e2 <- c(e1,e2)
e1e2
```

```
## [1] 1 6
```

Now let's create a new R matrix based on `numbers` and using index ranges to select the upper-left 2x2 sub-matrix:

```
# Create a 4x3 matrix
numbermat2 <- matrix(data=numbers, ncol=3)
```

```
# Select a subset
numbermat3 <- numbermat2[1:2,1:2]
numbermat3
```

```
##      [,1] [,2]
## [1,]    1    5
## [2,]    2    6
```

```
dim(numbermat3)
```

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

TRY IT: In the console, type numbermat[,2] and numbermat[1,]. What happens when you do not specify the row or column when indexing a matrix?

```
numbermat[,2]
```

```
## [1]  7  8  9 10 11 12
```

```
#gives all the numbers in the second row
numbermat[1,]
```

```
## [1] 1 7
```

```
#gives all the numbers in the first column
```

Tryit : Create an R vector called myvec containing the consecutive integers from 1 to 18. Then use this vector to create a matrix with 3 rows and 6 columns. *Print the result.*

```
myvec <- c(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18)
mymat <- matrix(data = myvec, ncol = 6)
print(mymat)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    1    4    7   10   13   16
## [2,]    2    5    8   11   14   17
## [3,]    3    6    9   12   15   18
```

Getting summary information about matrix rows and columns

We can use indexing to obtain summary information about rows and columns of matrices. Let's find out some things about `numbermat2`

```
# The mean of the second column
c2<-numbermat2[,2]       # View column 2
str(c2)
```

```
##  num [1:4] 5 6 7 8
```

```
mean(c2) # view the mean of column 2
```

```
## [1] 6.5
```

```
# The median of row 3
r3<-numbermat2[3,]
median(numbermat2[3,])
```

```
## [1] 7
```

** WARNING R thinks`numbermat2[,2]` is now a vector instead of a matrix.** Here we used the 'str()' to see that 'c2' is a vector. Alternatively you can see the object 'c2' listed in the environment pane.

Row and column vectors extracted from matrices are converted by R to vectors by default.

Introduction to Data Frames

A data file will often have columns with different data types, including numbers, characters, logicals, and "factors." This is very common when we import real-life data into R, whether from research projects or downloaded from the Web.

For example, we might have a data set with one column called "Country," composed of character strings or "factors," and another column called "Average Annual Income," populated with numerical values. To handle heterogeneous data of this type R provides the *data frame* (generated with the `data.frame()` function) which can be crudely be thought of as a matrix whose columns can hold data of different classes. The only required arguments in `data.frame()` are valid R vectors which all have the same length. Each of these vectors will constitute a column of the resulting data frame.

Creating data frames from vectors

Let's create a `data.frame` with two columns; the first one being `myletters` and the second one being the first l elements of `numbers`, where l is the length of `myletters`. Call it `df1` and *print the result.*

```
# First create a character vector
myletters <- c("a","b","c","c","d","d","e","d","a","d","k")

l <- length(myletters)    # Determine number of elements in myletters
numbers6 <- numbers[1:l]  # A subset of numbers of length l

# Create a dataframe with these two same-length vectors
df1 <- data.frame(myletters,numbers6)
df1
```

```
##    myletters numbers6
## 1          a        1
## 2          b        2
## 3          c        3
## 4          c        4
## 5          d        5
## 6          d        6
## 7          e        7
## 8          d        8
## 9          a        9
## 10         d       10
## 11         k       11
```

Notice that data frames have column names, which matrices do not automatically have. We can use `colnames()` to view and set these names.

```
# View the existing column names
colnames(df1)
```

```
## [1] "myletters" "numbers6"
```

```
# Change the column names
colnames(df1) <- c("Letters", "Numbers")
df1
```

```
##    Letters Numbers
## 1        a       1
## 2        b       2
## 3        c       3
## 4        c       4
## 5        d       5
## 6        d       6
## 7        e       7
## 8        d       8
## 9        a       9
## 10       d      10
## 11       k      11
```

Obtaining summary information about data frames

To obtain the dimensions of a matrix or data frame, use `dim()`, `nrow()` and `ncol()`

a. Get the dimensions of `df1` and store it in a variable named `dm`.
b. Use `ncol()` and `nrow()` to get the number of columns and number of rows of a data frame or matrix. Compare this with your previous result.

```
dm <- dim(df1)
dm_col <- ncol(df1)
dm_row <- nrow(df1)

print(dm_col)
```

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

```
print(dm_row)
```

```
## [1] 11
```

As with matrices, we can use `summary()` to obtain summary information about data frames, based on the class of each column. **TRY IT:** In the console, type summary(df1) What is the summary of the `Letters` column showing? View the contents of `df1` if you are confused…

```
summary(df1)
```

```
##  Letters    Numbers
##  a:2     Min.   : 1.0
##  b:1     1st Qu.: 3.5
##  c:2     Median : 6.0
##  d:4     Mean   : 6.0
##  e:1     3rd Qu.: 8.5
##  k:1     Max.   :11.0
```

```
#The letters column is showing the number of those letters present in the data frame
```

Get this thing? Extracting subsets of data using ==

We can also get parts of dataframes by using the `<data frame name>$<the column name>`. The `==` command lets us do comparisons to filter on elements of a list that we are interested in. This can let us compute all sorts of fun stuff. Here is code to compute to find the mean of the number in df and to compute the the mean of the numbers corresponding to the letter 'c'.

```
# calculate the mean of Numbers
mean(df1$Numbers)
```

```
## [1] 6
```

```r
# Make a data frame with just letters "c"
justc.df<-df1[df1$Letters=="c",]
# Calculate the mean
mean(justc.df$Numbers)
```

```
## [1] 3.5
```

TRY IT: In the console, write a line of code that determines how many entries correspond to letter "d"?

```r
print(nrow(df1[df1$Letters=="d",]))
```

```
## [1] 4
```

What is this thing? Determining the class of R objects

Use the `class()` function to obtain the class of an object.

a. Get the class of `df1`, `numbermat`, `myletters`, and `numbers`.

```r
class(df1)
```

```
## [1] "data.frame"
```

```r
class(numbermat)
```

```
## [1] "matrix"
```

```r
class(myletters)
```

```
## [1] "character"
```

```r
class(numbers)
```

```
## [1] "numeric"
```

Note that myletters and numbers are both vectors, but they have different types, "character" and "numeric" respectively.

Exercise 1 :

a. (2 pts) Create a `data.frame` with two columns: the first one containing the letters of your first and last name and the second containing a vector of l consecutive integers, starting from 1, where l is the length of your first name and last name together.

```r
name_vec <- c('j','a','r','e','d','g','r','i','d','l','e','y')
namedf <- data.frame(name_vec, c(1:length(name_vec)))
colnames(namedf) <- c("Letters", "Number")
print(namedf)
```

```
##    Letters Number
## 1        j      1
## 2        a      2
## 3        r      3
## 4        e      4
## 5        d      5
## 6        g      6
## 7        r      7
## 8        i      8
## 9        d      9
## 10       l     10
## 11       e     11
## 12       y     12
```

b. (2 pts) Convert this data frame into a matrix and print it.

```r
name_matrix <- as.matrix(namedf)
print(name_matrix)
```

```
##       Letters Number
##  [1,] "j"     " 1"
##  [2,] "a"     " 2"
##  [3,] "r"     " 3"
##  [4,] "e"     " 4"
##  [5,] "d"     " 5"
##  [6,] "g"     " 6"
##  [7,] "r"     " 7"
##  [8,] "i"     " 8"
##  [9,] "d"     " 9"
## [10,] "l"     "10"
## [11,] "e"     "11"
## [12,] "y"     "12"
```

c. (2 pts) What are the classes of each of its (the matrix) columns?

```r
class(name_matrix[1])
```

```
## [1] "character"
```

```r
class(name_matrix[2])
```

```
## [1] "character"
```

```r
#They are both character classes
```

d. (2 pts) Determine the dimensions of the `data.frame` you created in two different ways.

```r
#Way 1
dim(namedf)
```

```
## [1] 12  2
```

```r
#way 2
dimensions <- c(nrow(namedf), ncol(namedf))
print(dimensions)
```

```
## [1] 12  2
```

e. (2 pts) Print the top right corner of the matrix you created.

```r
print(name_matrix[1,2])
```

```
## Number
##   " 1"
```

TRY IT: In the console, type numbermat[,2] and numbermat[1,]. What happens when you do not specificy the row or column when indexing a matrix?

Exercise 2 :

Now we will look at a dataset built into R.
The ToothGrowth data set contains the result from an experiment studying the effect of vitamin C on tooth growth in 60 Guinea pigs. Each animal received one of three dose levels of vitamin C (0.5, 1, and 2 mg/day) by one of two delivery methods, (orange juice or ascorbic acid (a form of vitamin C and coded as VC). This

code chunk puts the data into data.df. It uses the command 'str' to quickly see what's in the data frame. The second column names "supp".

```r
# Get the data frame and assign to  toothgrowth.df
toothgrowth.df <-ToothGrowth
# Check out its stucture
str(toothgrowth.df)
```

```
## 'data.frame':    60 obs. of  3 variables:
##  $ len : num  4.2 11.5 7.3 5.8 6.4 10 11.2 11.2 5.2 7 ...
##  $ supp: Factor w/ 2 levels "OJ","VC": 2 2 2 2 2 2 2 2 2 2 ...
##  $ dose: num  0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 ...
```

Answer the following questions. To obtain full credit the calculations for each section must be given in the code chunk above.

a. (1 pt) How many guinea pigs are in the dataset?

```r
sample_size <- nrow(toothgrowth.df)
```

b. (1 pt) How many guinea pigs received Orange Juice?

```r
OJ_pigs <- toothgrowth.df[toothgrowth.df$supp=="OJ",]
num_OJ <- nrow(OJ_pigs)
print(num_OJ)
```

```
## [1] 30
```

c. (1 pt) What type of delivery method was used on the third guinea pig?

```r
print(toothgrowth.df[3,"supp"])
```

```
## [1] VC
## Levels: OJ VC
```

```r
#Can also do 2 instead of supp
```

d. (1 pt) How long was the 35th guinea pigs tooth?

```r
print(toothgrowth.df[35,"len"])
```

```
## [1] 14.5
```

e. (1 pt) Calculate the mean toothlength of guinea pigs for each dose of Vitamin C (0.5, 1 and 2 mg/day). 3 points

```r
VC_pigs <- toothgrowth.df[toothgrowth.df$supp=="VC",]
print(mean(VC_pigs[,"len"]))
```

```
## [1] 16.96333
```

f. (3 pts) Use the data to investigate whether vitamin C contributes to tooth growth in guinea pigs? Calculate your results. Give your conclusion and indicate your reasoning.

```r
OJ_mean <- mean(OJ_pigs[,"len"])
VC_mean <- mean(VC_pigs[,"len"])
print(OJ_mean)
```

```
## [1] 20.66333
```

```r
print(VC_mean)
```

```
## [1] 16.96333
```

The mean length of the guinea pigs given Orange Juice is higher than the pigs given Vitamin C. The mean length from OJ pigs was 20.663 and the mean length of the VC pigs was 16.963. This implies that Orange Juice was more effective at stimulating tooth growth than Vitamin C. However, we are missing a control group from the dataset, a group of guinea pigs that did not recieve any additional sources of vitamin C (because Orange Juice does contain a significant amount of Vitamin C). Therefore, Vitamin C by itself does not appear to stimulate tooth growth, however it help when combined with other vitamins or supplements.

g. (3 pts) Do a literature search and see if you can find an article that shows vitamin C is important for tooth or bone growth in some species besides guinea pigs. Summarize your findings and give a reference here in this file.

I found and article from runner's world that looked at 66 studies between 2000 and 2020. It found that ascorbic acid, by itself, cannot replace the bone growing benefits of calcium (which are well-documented), but that it did seem to have a complementary effect. It found that some research suggested that Vitamin C did help bone growth by stimulating other processes that are necessary for healthy bone growth, but that more research was necessary to prove a direct relationship. (https://www.runnersworld.com/news/a33633383/vitamin-c-for-bone-health-study/)

h. (2 pts) Then introduce yourself and post your findings in the discussion in LMS. Keep one of the statements below.

Yes, I introduced myself and posted my findings.

After you have completed these exercises and put your answers into this document, knit the pdf, download the knitted pdf, and submit it to LMS. If you need help knitting, see the appendix.

Appendix: Lab 1

The appendix contains a summary of the commands in the lab and a bunch of potentially useful but optional additional documentation.

Reference: Knitting and Uploading to LMS:

1. Always put your name in the `author` field at the very top of any `Rmd` (R Markdown) file you create!
2. Generate a PDF of your notebook for submission on LMS. To `knit` (render) your notebook into a PDF document, click the arrow on the right side of the "knit" button at the top of your R Studio window, and then click "knit to PDF".

3. Assuming there are no bugs in your code, a new window containing the PDF will automatically open (it won't happen instantly as it needs some time to compile your code, so have patience :D). A PDF file will also appear in your current working directory; if you've followed these instructions closely, this will be `IDM_work`.
4. Find the PDF you just created in using the **Files** pane and check the box on its left hand side. Then click the **More** tab and select **Export...** . Click the **Download** button and the file will download to your personal computer.
5. From there, follow the normal procedure for uploading to LMS.

Reference: Essential R Commands for Lab 1

The following are the most common R commands and operators you will use in your R scripts and from the R console, with a brief description of their use:

- `<-` - Assign a value to a variable.
  - Example: `var1 <- 3`
- `c()` - Create an R vector.
  - Example: `c(1,2,3,4,5)`
- `length()` - Get the length of an R vector.

    - Example: `length(some_vector)`
- `matrix()` - Used to create a matrix.
    - Example: `matrix(data=c(1,2,3,4),nrow=2,ncol=2))`
- `?` - Pull up the documentation for an R function.
    - Example: `?some_function`
- `class()` - Get the class of an object.
    - Example: `class(some_object)`
- `[` - Access elements of an R data structure.
    - Example: `some_data_structure[c(3,7)]` (returns third and seventh elements of `some_data_structure` in an R vector)
- `data.frame()` - 2-dimensional data structure composed of columns with different data types.
    - Example: `data.frame(col1=c(1,2,3), col2=c('a','b','c'))`
- `as.data.frame()/as.matrix()` - Convert an object to `data.frame/matrix`
    - Example: `as.data.frame(some_matrix)/as.matrix(some_data.frame)`
- `dim()` - Get the dimension of a data frame or matrix.
    - Example: `dim(some_data_frame)`
- `ncol()/nrow()` - Get number of columns/rows of a matrix or data frame
    - Example: `ncol(some_matrix)/nrow(some_matrix)`
- `==` - Compare each element of a to the value in b and return true or false
    - Example: `a=="S"`
- `$` - Extract column from a data frame by column name
    - Example: `a$Length`

Reference: RStudio Cheatsheets

RStudio provides a number of convenient and useful "cheatsheets" through their web site https://www.rstudio.com/resources/cheatsheets/ and via the RStudio **Help** menu. Some cheatsheets you may want to have available this term include:

1. RStudio IDE download
2. RMarkdown download
3. Data Import download
4. Data Visualization download

Reference: Knitting and Uploading to LMS:

1. Always put your name in the `author` field at the very top of any `Rmd` (R Markdown) file you create!
2. Generate a PDF of your notebook for submission on LMS. To `knit` (render) your notebook into a PDF document, click the arrow on the right side of the "knit" button at the top of your R Studio window, and then click "knit to PDF".

3. Assuming there are no bugs in your code, a new window containing the PDF will automatically open (it won't happen instantly as it needs some time to compile your code, so have patience :D). A PDF file will also appear in your current working directory; if you've followed these instructions closely, this will be `IDM_work`.
4. Find the PDF you just created in using the **Files** pane and check the box on its left hand side. Then click the **More** tab and select **Export...**. Click the **Download** button and the file will download to your personal computer.
5. From there, follow the normal procedure for uploading to LMS.

Reference: VPN Remote Access

You can access RStudio Server on the IDEA Cluster as long as you are on the RPI network. If you are off campus, you can VPN in by following the instructions at http://dotcio.rpi.edu/services/network-remote-access/vpn-connection-and-installation