base_R_I

February 25, 2025

1 Base R Part 1

This is a Jupyter Notebook. Essentially, this is a file where you can both write text (like this) and code (as you will see below). The goal of this lecture is to get you comfortable with basic commands in the R programming language, which we will call R for short.

2 Comments and Basic Data Types

In this first section, we will discuss writing comments and learn about basic data types (numeric, logical, character, boolean).

[1]:
```
# This is a comment, you can use '#' to write notes to yourself in your code
# - Comments can make or break good coders
# - Good comments also create coders who can collaborate with others
# - If you ever think you're writing "too" many comments, you are not
# - The things you think are obvious in your code won't be to others
# - (nor yourself in a year when you get back to a project)
```

[2]:
```
# Comments are written in the same spot as code
# But they are ignored by the computer

# Let's run some code
print("Hello, World!")
```

```
[1] "Hello, World!"
```

Congrats! You just ran your first line of code and joined the thousands of people whose first line of code was "Hello, World!" as well. Welcome to the tradition.

Let's now run our first operation.

[3]:
```
2 + 3
```

5

We can see below our code block that the code block "returned" 5. This means that the computer "executed" the line of code `2 + 3`.

Now, let's create a variable and store a value in it. Creating variables is a core piece of programming.

[4]:
```
# in R, we assign a value to a variable with the following arrow symbol: <-
a <- 2
b <- 3

a + b
```

5

[5]:
```
# The nice thing about a variable is that we can change the value assigned

# Let's change the value of our variable a
a <- 6

a + b
```

9

Let's now discuss some basic data types.

[6]:
```
# Numeric -- integer: no decimal points
myInt <- 1
myInt

# Numeric -- double/float: decimal points
myNum <- 2.4
myNum
```

1

2.4

[7]:
```
# character (string)
myChar_a <- "a"
myChar_a

myChar_b <- 'b'
myChar_b
```

'a'

'b'

[8]:
```
# You can have a variable that stores a character value,
# but the character value is a number

trick_q <- "1"
trick_q
```

'1'

Notice that the `'1'` has quotation marks around it, as the `'a'` and `'b'` did. This indicates to us that it is a character data type. When you look at the output of `myInt`, it also returns `1` but

without the quotation marks because `myInt` is a numeric variable and `trick_q` is a character (or string) variable.

[9]:
```
# logical (Boolean/Indicator variable): a true/false statement.
# Use () to evaluate if something is true or false
myBool_1 <- (3 < 4)
myBool_1

myBool_2 <- (3 > 4)
myBool_2
```

TRUE

FALSE

2.1 Ways to store data types

[10]:
```
# Vector: can only be a vector of one data type (numeric, logical, string)

# numeric vector
myVec_n <- c(1, 2, 3, 4, 5)
myVec_n

# string vector
myVec_s <- c(myChar_a, "b", "c")
myVec_s

# tricky! But a string
myVec_string <- c(1, "b", "c")
myVec_string # notice the 1 has been made a character because of the "
```

1. 1 2. 2 3. 3 4. 4 5. 5

1. 'a' 2. 'b' 3. 'c'

1. '1' 2. 'b' 3. 'c'

Vectors are a useful way to store multiple observations. However, they only have a single column of information. In EDS, we are interested in relating different variables to one another (*i.e.,* How does temperature affect a lion's hunting success rate?)

To relate multiple variables together, we need to make collections of data. The most basic way to do this is a **matrix**.

[11]:
```
# We already have a vector of numeric data
myVec_n

# Matrix: should only be a matrix of one data type
myMat_n <- matrix(c(myVec_n, # first elements will come from our pre-existing␣
 ↪vector
                    6, 7, 8, 9, 10),
```

```
              nrow = 2,
              ncol = 5)

myMat_n
```

1. 1 2. 2 3. 3 4. 4 5. 5

A matrix: 2 x 5 of type dbl

1	3	5	7	9
2	4	6	8	10

We can also collect objects in a "list". Lists are very powerful, but more advanced. For now, just know that they also exist.

[12]:
```
# Lists: Very powerful, but somewhat confusing. For now, just know they exist
myList <- list(2, "c", myMat_n)
myList[[1]] # returns numeric
myList[[2]] # returns string
myList[[3]] # returns matrix
```

2

'c'

A matrix: 2 x 5 of type dbl

1	3	5	7	9
2	4	6	8	10

2.2 Data Frames

- Like matrices
- Can have different data types in each column
- Reference specific columns using the "$" operator, followed by the name of the column
- For the most part, you'll be loading new data by reading a CSV
- You might have to create one at some point
- By looking at how they're created we can get a better sense of what goes into them

Let's convert our matrix into a data frame.

[13]:
```
# Data frame: can have multiple data types
myDF <- as.data.frame(myMat_n)
colnames(myDF) # print the column names
```

1. 'V1' 2. 'V2' 3. 'V3' 4. 'V4' 5. 'V5'

These column names don't mean anything to me. Let's assign some column names. Each column is a variable that you may have collected data.

[14]:
```
colnames(myDF) <- c("num_hunts", "temp", "num_adults", "num_cubs",␣
 ↪"distance_from_road")
colnames(myDF)
```

1. 'num_hunts' 2. 'temp' 3. 'num_adults' 4. 'num_cubs' 5. 'distance_from_road'

We can now look at a single column using the '$' operator.

[15]:
```
# Investigate one column
myDF$num_adults
```

1. 5 2. 6

Let's create a new column.

[16]:
```
# Create a new column
myDF$total_lions <- myDF$num_adults + myDF$num_cubs
myDF
```

A data.frame: 2 x 6

num_hunts <dbl>	temp <dbl>	num_adults <dbl>	num_cubs <dbl>	distance_from_road <dbl>	total_lions <dbl>
1	3	5	7	9	12
2	4	6	8	10	14

Let's build a dataframe with multiple datatypes. (Our lion dataframe is only numeric columns).

[17]:
```
# Create the data frame
myPpl <- data.frame(
   name = c("Andie", "Sam", "Bill"),
   gender = c("Female", "non-binary", "Male"),
   male = c(FALSE, FALSE, TRUE),
   income_cat = c("middle", "poor", "rich"),
   park_dist_mi = c(1, 0.5, 0.1)
)
myPpl
```

A data.frame: 3 x 5

name <chr>	gender <chr>	male <lgl>	income_cat <chr>	park_dist_mi <dbl>
Andie	Female	FALSE	middle	1.0
Sam	non-binary	FALSE	poor	0.5
Bill	Male	TRUE	rich	0.1

Now that we have our dataframe built, we can reference columns in the data frame.

First, there are multiple ways to reference a column. The first is to reference by the column name.

[18]:
```
# Try referencing one column
myPpl$name # version 1
```

1. 'Andie' 2. 'Sam' 3. 'Bill'

The second is the column number in the data frame using square bracket notation `[row, column]`

[19]:
```
myPpl[, 1] # version 2
```

1. 'Andie' 2. 'Sam' 3. 'Bill'

We can also reference a single row, instead of a column.

[20]:
```
# Try referencing one row
myPpl[1, ]
```

A data.frame: 1 x 5

	name	gender	male	income_cat	park_dist_mi
	<chr>	<chr>	<lgl>	<chr>	<dbl>
1	Andie	Female	FALSE	middle	1

Finally, let's try to reference a single cell. Like referencing columns, there are two ways to do this.

[21]:
```
# Try referencing one cell
myPpl$name[2] # version 1: combination of $ and []
```

'Sam'

[22]:
```
myPpl[2, 1] # version 2
```

'Sam'

3 A word of caution

- Make sure you don't overwrite your variables by accident.

[23]:
```
# Assigning new value to same variable (something to do carefully)
a <- 5
a <- a + 1 # If you run this line more than once, you will NOT get six
a

# Assigning new value to new variable
a <- 5
a_new <- a + 1 # If you run this line more than once, you WILL get six
a_new
```

6

6