R Notebook #1: Introduction to R Notebooks and R Data Types and Structures

Introduction to R Markdown Notebooks

This is an R Markdown Notebook. When you execute code within the notebook, the results appear beneath the code.

Try executing this chunk by clicking the *Run* button within the chunk or by placing your cursor inside it and pressing *Cmd+Shift+Enter*.

```
2+2
```

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

Add a new chunk by clicking the *Insert Chunk* button on the toolbar or by pressing *Cmd+Option+I*.

When you "knit" the notebook, a pdf file containing the code and output will be saved and displayed.

More information can be added on the first line of code chunks. For example, in "`{r testChunk}` we are naming the code chunk `testChunk`. The point is to use a descriptive label.

Options can also be added to the first line. For example "`{r testChunk echo=FALSE}` tells R Studio to not print the code chunk when the notebook is knit. The chunk will be executed, but the code itself won't be displayed in the final document. Only the results or output of the code (like plots or printed values) will be shown.

Also, within a code chunk, text that follows "#" is referred to as a "comment". Other terminology, is that code following a "#" is "commented out." Commented out lines are not run. They are ignored. You can comment out text to add notes within a code chunk. Also, if you want to a line of code to not be run (e.g. for testing an alternative) but you do not want to delete it, you can comment it out. Here's an example.

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

Note that you can also click on the little wheel inside the code chunk. This provides a user-friendly way to specify information about the chunk. Unfortunately, it doesn't show the information you may already entered, but rather provides another way to enter it.

In R Studio, the "Source" tab shows markdown language. Markdown is a lightweight markup language with plain-text formatting syntax that is designed to be converted to HTML, PDF and other formats. It is very popular among R users.

The "Visual" tab feels more like a traditional word processor. The Visual tab can be more user-friendly, but it can also be less flexible.

This R Markdown Cheat Sheet can help you get acquainted with Markdown.

R Objects

In R, the term "object" refers to anything that can be assigned to a variable. This encompasses a wide range of data structures, from simple types like numbers and strings, to more complex structures like vectors, lists, matrices, data frames, and functions, which we will learn about.

In R, everything you manipulate is considered an object. This includes not only data structures like vectors or data frames but also the outputs of functions and the functions themselves. Some key points about objects:

1. **Attributes**: Objects in R can have attributes. These are "metadata" about the object, like its dimensions (for matrices), names (for elements in a vector or columns in a data frame), or class (which defines how the object should behave in different contexts). (Note, in some fields, columns of a data set may be called attributes, so this may be confusing. In this class, I will refer to columns typically as variables or measures.)
2. **Types and Structures**: The type of an object refers to its basic data type (numeric, integer, character, etc.), while the structure refers to how these types are organized (like vectors, lists, matrices). We turn to types and structures below.
3. **Classes and Methods**: In object-oriented programming within R, objects can have specific classes, and there are methods designed to work with objects of these classes. For instance, the `print(x)` function behaves differently for object `x` depending on whether `x` is a numeric vector, a data frame, or a model object.
4. **Environment Objects**: Recall you can see the objects in memory by clicking on the "Environment" tab in the upper right quadrant of R Studio.

R Data Types

1. **Numeric**: This is the default data type for numbers in R. For example, `2` and `6.7` etc. are considered numeric.

```
# x is an object that we assign the numeric value of 2
# note "<-" assigns information to objects
x <- 2

# we can print the object x
print(x)
```

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

```
# we can also print just by typing "x"
x
```

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

```
# we can also add more information to make our output nicer
print(paste("x is equal to",x))
```

```
## [1] "x is equal to 2"
```

```
# we can check the type of the object x as follows
class(x)
```

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

2. **Character**: This type includes strings of text. Any data enclosed in single or double quotes is considered a character type, e.g., `"Blue"`.

```
# city is an object that we assign the character value "Raleigh"
# note that character values are in quotation marks
city <- "Raleigh"

print(city)
```

```
## [1] "Raleigh"
```

```r
# Here we can view the class of the object called city
class(city)
```

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

3. **Logical**: This data type is used for "boolean" values (or "dummy" variables), which can either be **`TRUE`** or **`FALSE`**.

```r
# student is an object that we assign the character value TRUE
# note that logical data is either TRUE of FALSE (case sensitive)
student <- TRUE

print(student)
```

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

```r
# Here we can view the class of the object called city
class(student)
```

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

Notes about object names

In the code chunks above, `x`, `city`, and `student` are all object. We simply assigned values to these objects. As we will see below, we can also assign bigger, more complex information to these objects. But first, let's review some rules and conventions for object names in R:

1. **Syntax Rules**: In R, valid object names must start with a letter, followed by a combination of letters, numbers, the period (`.`), and the underscore (`_`). They cannot start with a number or an underscore, nor can they contain special characters like `^`, `!`, `@`, `#`, `$`, `%`, `&`, `*`, `(`, `)`, `-`, `+`, `=`, `{`, `}`, `[`, `]`, `|`, `\`, `:`, `;`, `'`, `"`, `<`, `>`, `,`, `?`, `/`, or space.
2. **Reserved Words**: R has a set of reserved words that cannot be used as object names. These include function names, keywords, and other built-in identifiers like `if`, `else`, `repeat`, `while`, `function`, `for`, `in`, `next`, and `break`.
3. **Case Sensitivity**: R is case-sensitive, which means that `Object`, `object`, and `OBJECT` would be considered different objects.
4. **Conventions**: While not enforced by R syntax, there are naming conventions that many R programmers follow for readability and to avoid confusion, such as using `.` or `_` to separate words (e.g., `my.data` or `my_data`), and avoiding the use of periods at the end, as they are conventionally used for other purposes beyond the scope of this class.
5. **Assignment**: You can technically use almost any character in an object name if you use backticks, like `` `1` `` for an object name, but this is not recommended as it can make the code hard to read and maintain.
6. **Global Environment**: Object names in the global environment should not conflict with object names in attached packages. If a name conflict occurs, R will use the object from the most recently attached package or environment.

In practice, it's best to follow the generally accepted coding standards and conventions to ensure that your code is clean, understandable, and doesn't clash with R's syntax or semantics.

R Data Structures

Central to R's flexibility for data analysis are its diverse data structures, which enable users to handle and manipulate data in a variety of ways. Understanding the fundamental data structures in R is crucial for performing data operations efficiently and effectively. Here, we will introduce the primary data structures used in R:

1. **Vectors**: The simplest and most common data structure in R, vectors are one-dimensional arrays that hold elements of the same type. They can be of various types themselves, such as numeric, integer, complex, logical, character, or raw.

```
# The object my.numeric.vector is a numeric vector:
my.vector <- c(1,2,3,4,5)

# The object my.char.vector is a character vector:
my.char.vector <- c("walk","bus","bike","drive")

# The object my.logical.vector is a logical vector
my.logical.vector <- c(TRUE,FALSE,TRUE)
```

2. **Matrices**: A matrix is a two-dimensional data structure with rows and columns, where every element is of the **same data type**. It can be considered as a collection of vectors of equal length arranged in a rectangular layout. We will not use matrices much, if at all. They can be very limiting since all entries need to be the same data type.

```
# my.matrix is a numeric matrix, which I create by combining two vectors.
# Each vector is a column of the matrix.
# I combine the vectors using function cbind().
a <- c(1,2,3)
b <- c(4,5,6)
my.matrix <- cbind(a,b)
print(my.matrix)
```

```
##      a b
## [1,] 1 4
## [2,] 2 5
## [3,] 3 6
```

```
# We can look at the dimensions of the matrix with the function dim(). This will return the number of r
dim(my.matrix)
```

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

3. **Data Frames**: Data frames are the most widely used data structure in R for data analysis. We will frequently use data frames in this class. A data frame is a two-dimensional table-like structure that can hold columns of different types (e.g., numeric, character, and logical). Each column in a data frame can be thought of as a vector, and the length of each vector (number of rows) must be the same across all columns.

```
# Here, first I am loading an R "package" called "datasets." R packages are the building blocks on top
library(datasets)

# One of the datasets is called "mtcars". With class(mtcars), we see it is a data frame
class(mtcars)
```

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

```
# Let's look at the dimensions of the data.frame, just as we did above for a matrix
dim(mtcars)
```

```
## [1] 32 11
```

```
# Let's look at the first 6 rows of the data frame mtcars, by using the function head()
head(mtcars)
```

```
##                    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

Notice that the car types do not have a column name. They techincaly are not a column in the data frame. Instead, the rows have names, which are they car types.

3. **Lists**: Lists are R's answer to the need for a versatile data structure. They are an ordered collection of objects (components), which can be of different types and sizes. Lists can contain vectors, matrices, other lists, and even functions as their elements.

4. **Factors**: Factors are used to handle categorical data. They are stored as integer vectors that have corresponding labels and are especially useful in statistical modeling.

5. **Data Tables and Tibbles**: These are enhanced versions of data frames provided by external packages like `data.table` and `tibble`. They offer advanced functionalities and are optimized for faster manipulation of tabular data.

Each of these data structures has its own set of characteristics and is designed to facilitate certain types of data operations. Choosing the appropriate data structure is key to writing efficient R code, particularly when dealing with complex data manipulation tasks or large datasets. The interplay between these structures, combined with R's comprehensive suite of functions and packages, makes it an exceptionally flexible tool for data analysis and research.