Lab-Objects_and_Classes

May 22, 2021

R OBJECTS and CLASSES

0.1 Table of Contents

Estimated Time Needed: 20 min

About the Dataset

Imagine you got many movie recomendations from your friends and compiled all of the recomendations in a table, with specific info about each movie.

The table has one row for each movie and several columns

- **name** - The name of the movie
- **year** - The year the movie was released
- **length_min** - The lenght of the movie in minutes
- **genre** - The genre of the movie
- **average_rating** - Average rating on Imdb
- **cost_millions** - The movie's production cost in millions
- **sequences** - The amount of sequences
- **foreign** - Indicative of whether the movie is foreign (1) or domestic (0)
- **age_restriction** - The age restriction for the movie

Part of the table can be seen here

Now, we are going to take a look at how R is storing and outputing the results of its objects. In other words, we'll find out how R can handle different kinds of data.

What is an Object?

Everthing that you manipulate in R, literally every entity in R, is considered an **object**. In real life, we think of an object as something that we can hold and look at. R objects are a lot like that. For example, vector is one of the objects in R.

An object in R has different kinds of properties or attributes. One of the attributes in objects is called the **class** of the object. The **class** of an object is the data type of this object. For instance, the class of vector can be numeric if it's composed of numeric values or character if it's composed of string values. The various classes (data types) of objects in R are important for data science programming.

0.1.1 Class

The most common classes (data types) of objects in R are:

numeric (real numbers)

character

integer

logical (True/False)

complex

If you want to know about the data type of your values, you can use the **"class()"** function and add the variables' name to it. Let's create a variable from the average rating of some movies and then find which data types they belong to:

```
[ ]: movie_rating <- c(8.3, 5.2, 9.3, 8.0) # create a vector from average ratings
     movie_rating # print the variable
```

To check what is the data type, let's use **class()**

```
[ ]: class(movie_rating) # show the variable's data type
```

As you see, the **class()** function shows that the data type of values in the vector is **numeric**.

Tip: A vector can only contain objects of the same class. However, a list can have different data types.

0.1.2 Numeric

Decimal values are called numerics. They are the default computational data type in R. In the example below, If we assign a decimal value to a variable average_rating, then average_rating will be of numeric type.

```
[ ]: average_rating <- 8.3       # assign a decimal value
     average_rating
```

Using **class** to check the data type results in **numeric**

```
[ ]: class(average_rating)
```

0.1.3 Character

A character object is used to represent string values in R, strings are simply text values.

```
[ ]: movies <-c("Toy Story", "Akira", "The Breakfast Club", "The Artist")
     movies
     class(movies)
```

If numbers and texts are combined in a vector, everything is converted to the class **character**. Let's make a vector from combined movie names and their production year, then find the data type for the vector

```
[ ]: combined <- c("Toy Story", 1995, "Akira", 1998)
     combined
     class(combined)
```

When you simply enter numbers into R, they will be saved as class **numeric** by default. For example in the following vector, even though the numbers are integers, they are stored as numeric type in R:

```
[ ]: movie_length <- c(80, 110, 90, 80) # create a vector from movie length
     movie_length # print the variable
     class(movie_length)
```

0.1.4 Integer

An integer is a number that can be written without a fractional component. For example, 21, 4, 0, and −2048 are integers, while 9.75, 5 ½, and √2 are not. In R, when you create a variable from the mentioned numbers, they are not going to be stored as integer data type. In order to get the integer class we need to convert the variable type from numeric to integer using **as.integer()** function. Let's create a vector and check if the data type is numeric.

```
[ ]: age_restriction <- c(12, 10, 18, 18) # create a vector from age restriction
     age_restriction # print the vector

     class(age_restriction)
```

```
[ ]: integer_vector <- as.integer(age_restriction)
     class(integer_vector)
```

0.1.5 Logical

The logical class contains True/False values (Boolean values). Let's create a vector with logical values and check its class:

```
[ ]: logical_vector <- c(T,F,F,T,T) # creating the vector
     class(logical_vector)
```

A logical value is often created via comparison between variables. In the below example, we will compare the length of movies **Toy Story and Akira**.

```
[ ]: length_Akira <- 125
     length_ToyStory <- 81
```

If we assign the result of the compare statement to a variable the variable will have FALSE if the statement was false, and TRUE if the statement is true.

```
[ ]: x <- length_ToyStory > length_Akira      # is ToyStory larger than akira?
     x
```

```
[ ]: x <- length_Akira > length_ToyStory # is akira larger than ToyStory?     ␣
      ↪
     x # print the logical value
```

The resulting variable is of type logical

```
[ ]: class(x)        # print the class name of x
```

0.1.6 Complex

A complex number is a number that can be expressed in the form a + bi, where a and b are real numbers and i is the imaginary unit.

```
[ ]: z = 8 + 6i       # create a complex number
     z
```

```
[ ]: class(z)
```

Converting One Class to Another

We can convert (coerce) one data type to another if we desire. For example, we can convert objects from other data types into character values with the **"as.character()"** function. In the following example, we convert numeric value into character:

```
[ ]: year <- as.character(1995) # convert integer into character data type
     year                       # print the value of year in character data type
```

As we mentioned before, in order to create an integer variable in R, we can use the **"as.integer()"** function. In the following example, even though the number is an integer data type, R saves the number as numeric data type by default. So you need to change the number to integer later if it is necessary.

```
[ ]: Length_ToyStory <- 81
     class(81)
```

```
[ ]: length_ToyStory <- as.integer(81)
     class(length_ToyStory)        # print the class name of length_ToyStory
```

Difference between Class and Mode

For a simple vector, the class and mode of the vector are the same thing: the data type of the values inside the vector (character, numeric, integer, etc). However, in some of other objects such as matrix, array, data frame, and list, class and mode means different things.

In those mentioned objects, the **class()** function shows the type of the data structure. What does that mean? The class of matrix will be **matrix** regardless of what data types are **inside** the matrix. The same applies to list, array and data frame.

Mode on the other hand, determines what types of data can be found within the object and how that values are stored. So, you need to use the **mode()** function to find the data type of values inside a matrix (character, numeric, integer, etc).

So, in addition to the classes such as numeric, character, integer, logical, and complex, we have other classes such as matix, array, dataframe, and list

0.1.7 Matrix

Let's create a matrix storing the genre for each movie. Then, we will find the class and mode of the created matrix to see which information we will get from them.

First, let's check the effect of class and mode on a **vector**.

[ ]:
```
movies <- c("Toy Story", "Akira", "The Breakfast Club", "The Artist") #␣
↪creating two vectors
genre <- c("Animation/Adventure/Comedy", "Animation/Adventure/Comedy", "Comedy/
↪Drama", "Comedy/Drama")

class(movies)
mode(movies)
```

As you see in the above, for the vector the class and mode shows the data type of values. Now lets create a matrix from these two vectors.

[ ]:
```
movies_genre <- cbind(movies, genre)
movies_genre
```

Now **class()** shows that the data type is **matrix**.

[ ]:
```
class(movies_genre)
```

And **mode** shows the data type of the elements of the matrix

[ ]:
```
mode(movies_genre)
```

For the matrix, the **class()** shows how the values are stored and shown in R, in this case, in a matrix. However, **mode()** shows the data type of values in the matrix. In the above example we have made a matrix filled with **character** values.

0.1.8 Array

A slightly more complicated version of the **matrix** data type is the **array** data type. The **array** data type can still only have one data type inside of it, but the set of data types it can store is larger. In addition to the data types an array can store matrices as its elements. In the following, we are going to create the array from integer number (1 to 12) and then compare the class and mode in an array:

```
[ ]: sample_array <- array(1:12, dim = c(3, 2, 2)) # create an array with dimensions␣
     ↪3 x 2 x 2
     sample_array
     class(sample_array)
     mode(sample_array)
```

So, the array's class is **array** and its mode is **numeric**.

0.1.9 Data Frame

Data frames are similar to arrays but they have certain advantages over arrays. Data frames allow you to associate each row and each column with a name of your choosing and allow **each column** of the data frame to have a **different data type** if you like. Let's create a data frame from the movie names, year and their length:

```
[ ]: Name <- c("Toy Story", "Akira", "The Breakfast Club", "The Artist")
     Year <- c(1995, 1998, 1985, 2011)
     Length <- c(81, 125, 97, 100)
     RowNames = c("Movie 1", "Movie 2", "Movie 3", "Movie 4")

     sample_DataFrame <- data.frame(Name, Year, Length, row.names=RowNames)
     sample_DataFrame

     class(sample_DataFrame)
```

So, the class of the above table is "data.frame".

0.1.10 List

The final data type that we are going to go over is **list**. Lists are similar to vectors, but they can contain multiple data types. For example:

```
[ ]: sample_List = list("Star Wars", 8.7, TRUE)
     sample_List

     class(sample_List)

     mode(sample_List)

     mode(sample_List[[3]])
```

As you see, we have character, numeric, and logical data types in the list. The data type of third element in the list is logical as the "mode()" function shows us. The **mode** for the entire list is **list**, it could show the type of all the elements, since they don't all have the same data type.

Attributes

Objects have one or more **attributes** that can modify how R thinks about the object. Imagine you have a bowl of pasta and cheese. If you add spice, you change it to something with new taste. Different spices makes different dishes.

Attributes are like spice. You can change any individual attribute of object with the **attr()** function. You also can use the **attribute()** function to return a list of all of the attributes currently defined for that object.

For example in the following code, we will create a vector from the average ratings (8.3, 8.1, 7.9, 8) and costs of four movies (30, 10.4, 1, 15), and then we change the **dim** attribute of the vector. R will now treat z as it were a 4-by-2 matrix.

```
[ ]: z <- c(8.3, 8.1, 7.9, 8, 30, 10.4, 1, 15)
     z
     attr(z, "dim") <- c(4,2)
     z
```

Now, we can find the class and mode of the above matrix.

```
[ ]: class(z)
     mode(z)
```

Scaling R with big data As you learn more about R, if you are interested in exploring platforms that can help you run analyses at scale, you might want to sign up for a free account on IBM Watson Studio, which allows you to run analyses in R with two Spark executors for free.

0.1.11 About the Author:

Hi! It's Rishabh Jain, the author of this notebook. I hope you found R easy to learn! There's lots more to learn about R but you're well on your way. Feel free to connect with me if you have any questions.