Lab-String_Operations

May 22, 2021

Strings in R

This notebook will provide information regarding reading text files, performing various operations on Strings and saving data into various types of files like text files, CSV files, Excel files etc.

0.1 Table of Contents

Estimated Time Needed: 25 min

About the Dataset

In this module, we are going to use **The_Artist.txt** file. This file contains text data which is basically summary of the **The Artist** movie and we are going to perform various operations on this data.

This is how our data look like.

Let's first **download** the data into your account:

[ ]:
```
# Download the data file
download.file("https://ibm.box.com/shared/static/
 ↪l8v8g8e6uzk7yj2j1qc8ypezbhzukphy.txt", destfile="The_Artist.txt")
```

Reading Text Files

To read text files in R, we can use the built-in R function **readLines()**. This function takes **file path** as the argument and read the whole file.

Let's read the **The_Artist.txt** file and see how it looks like.

[ ]:
```
my_data <- readLines("The_Artist.txt")
my_data
```

Tip: If you got an error message here, make sure that you run the code cell above first to download the dataset.

So, we get a character vector which has three elements and these elements can be accessed as we access array.

Let's check the length of **my__data** variable

```
[ ]: length(my_data)
```

Length of **my__data** variable is **5** which means it contains 5 elements.

Similarly, we can check the size of the file by using the **file.size()** method of R and it takes **file path** as argument and returns the number of bytes. By executing code block below, we will get **1065** at the output, which is the size of the file **in bytes**.

```
[ ]: file.size("/resources/data/The_Artist.txt")
```

There is another method **scan()** which can be used to read **.txt** files. The Difference in **readLines()** and **scan()** method is that, **readLines()** is used to read text files line by line whereas **scan()** method read the text files word by word.

scan() method takes two arguments. First is the **file path** and second argument is the string expression according to which we want to separate the words. Like in example below, we pass an empty string as the separator argument.

```
[ ]: my_data1 <- scan("The_Artist.txt", "")
     my_data1
```

And if we will check length of **my__data1** variable then we will get total number of elements at the output.

```
[ ]: length(my_data1)
```

String Operations

There are many string operation methods in R which can be used to manipulate the data. We are going to use some basic string operations on the data that we read before.

nchar()

The first function is **nchar()** which will return the total number of characters in the given string. Let's find out how many characters are there in the first element of **my__data** variable.

```
[ ]: nchar(my_data[1])
```

toupper()

Now, sometimes we need the whole string to be in Upper Case. To do so, there is a function called **toupper()** in R which takes a string as input and provides the whole string in upper case at output.

```
[ ]: toupper(my_data[3])
```

In above code block, we convert the third element of the character vector in upper case.

tolower()

Similarly, **tolower()** method can be used to convert whole string into lower case. Let's convert the same string that we convert in upper case, into lower case.

[ ]:
```
tolower(my_data[3])
```

We can clearly see the difference between the outputs of last two methods.

chartr()

`what if we want to replace any characters in given string?` This operation can also be performed in R using **chartr()** method which takes three arguments. The first argument is the characters which we want to replace in string, second argument is the new characters and the last argument is the string on which operation will be performed.

Let's replace **white spaces** in the string with the **hyphen ("-") sign** in the first element of the **my_data** variable.

[ ]:
```
chartr(" ", "-", my_data[1])
```

strsplit()

Previously, we learned that we can read file word by word using **scan()** function. `But what if we want to split the given string word by word?`

This can be done using **strsplit()** method. Let's split the string according to the white spaces.

[ ]:
```
character_list <- strsplit(my_data[1], " ")
word_list <- unlist(character_list)
word_list
```

In above code block, we separate the string word by word, but **strsplit()** method provides a list at the output which contains all the separated words as single element which is more complex to read. So, to make it more easy to read each word as single element, we used **unlist()** method which converts the list into character vector and now we can easily access each word as a single element.

sort()

Sorting is also possible in R. Let's use **sort()** method to sort elements of the **word_list** character vector in ascending order.

[ ]:
```
sorted_list <- sort(word_list)
sorted_list
```

paste()

Now, we sort all the elements of ** word_list** character vector. Let's use **paste()** function here, which is used to concatenate strings. This method takes two arguments, the strings we want to concatenate and **collapse** argument which defines the separator in the words.

Here, we are going to concatenate all words of **sorted_list** character vector into a single string.

[ ]:
```
paste(sorted_list, collapse = " ")
```

substr()

There is another function **substr()** in R which is used to get a sub section of the string.

Let's take an example to understand it more. In example below, we use the **substr()** method and provide it three arguments. First argument is the data string from which we want the sub string. Second argument is the starting point from where function will start reading the string and the third argument is the stopping point till where we want the function to read string.

[ ]:
```
sub_string <- substr(my_data[1], start = 4, stop = 50)
sub_string
```

So, from the character vector, we start reading the first element from 4th position and read the string till 50th position and at the output, we get the resulted string which we stored in **sub_string** variable.

trimws()

As the sub string that we get in code block above, have some white spaces at the initial and end points. So, to quickly remove them, we can use **trimws()** method of R like shown below.

[ ]:
```
trimws(sub_string)
```

So, at the output, we get the string which does not contain any white spaces at the both ends.

str_sub()

To read string from last, we are using **stringr** library. This library contains **str_sub()** method, which takes same arguments as **sub_stirng** method but read string from last.

Like in the example below, we provide a data string and both starting and end points with negative values which indicates that we are reading string from last.

[ ]:
```
library(stringr)
str_sub(my_data[1], -8, -1)
```

So, we read string from -1 till -8 and it gives **talkies.** with full stop mark at the output.

Writing and Saving to Files

After reading files, we can also write data into files and save them in different file formats like **.txt, .csv, .xls (for excel files) etc**. Let's take a look at some examples.

Exporting as Text File

Suppose we want to export a matrices or String in **.txt** file. To do so, we can use **write()** method which writes into file and save that on to disk.

Let's create a matrix and try to save it into file.

[ ]:
```
m <- matrix(c(1, 2, 3, 4, 5, 6), nrow = 2, ncol = 3)
m
```

[ ]:
```
write(m, file = "my_text_file.txt", ncolumns = 3, sep = " ")
```

In above code block, we provide the input data, file name in which we want to store data along with its path and as we are using matrices to output in file, we provide **ncolumns** argument value and **sep** argument.

Let's try to write a string from our **my__data** variable into file named as **my__text__file2.txt**

[ ]:
```
write(my_data[1], file = "my_text_file2.txt", ncolumns = 1, sep = " ")
```

So, we get the first element from **my__data** variable and provide it as input to write function and this time we assign **ncolumn** argument with value 1 because we want a single column for string.

Exporting as CSV File

As we export data in text files, we can export data into **CSV** files also. To do so, we need a data frame which have data. For this, we can use built-in datasets.

Let's use **CO2** dataset of R which contains data about Carbon Dioxide Uptake in Grass Plants. Let's see how data look like in **CO2**dataset.

[ ]:
```
head(CO2)
```

Now, let's export this data into **CSV** file. We will use **write.csv()** method which takes data frame as input and a **file** argument to specify output filename.

[ ]:
```
write.csv(CO2, file = "my_csv.csv")
```

Now, when we will execute above code block, all data will be exported in CSV file. Now, the first column of CSV file contains row numbers which we do not want in our CSV file. So, we have to define **row.names** to **FALSE** in **write.csv()** method.

[ ]:
```
write.csv(CO2, file = "my_csv.csv", row.names = FALSE)
```

Similarly, to remove column names just make **col.names** to **FALSE**.

Exporting as Excel File

To save data into excel files, we have to install an external library called **xlsx**, which will provide us easy methods to export data into **.xlsx** files.

Let's install this library. (This may take a minute or two)

[ ]:
```
install.packages("xlsx")
```

[ ]:
```
library(xlsx)
write.xlsx(CO2, file = "my_excel.xlsx", row.names = FALSE)
```

So, exporting data in **.xlsx** files is similary to **.csv** files just function name is different plus we had to install external library to do this operation.

Exporting as .RData File

In R, we can also save files in **.RData** format. **.RData** format provides a way to save and load our R objects.

Let's create simple variable objects and save that into file with extension**.RData**.

[ ]:
```
var1 <- "var1"
var2 <- "var2"
var3 <- "var3"
```

Now, to write in **.RData** file, we will use **save()** method of R. It has a **list** argument which is the list containing the variable names of all the objects we want to save (which in this case are three vaiables), **file** argument which contains file name in which we are going to write/save data and **safe** argument is to specify whether you want the saving to be performed atomically.

[ ]:
```
save(list = c("var1", "var2", "var3"), file = "variables.RData", safe = T)
```

The file with name **variables.RData** is generated on the provided location.

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.

0.1.1 About the Author:

Hi! It's Iqbal Singh, 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.