Introduction_to_R

Ravi Mummigatti

7/21/2021

Contents

Variables and Data Types

In this section we will see the rules for naming the variables in R , the basic data types available in R and to see two basic R objects; vectors and lists, in detail.

Naming Variables

The variable name in R has to be alphanumeric characters with an exception of underscore and period, the special characters which can be used in the variable names.

The variable name has to be started always with an alphabet and no other special characters except the underscore and period are allowed in the variable names.

- Allowed characters are Alphanumeric, '_' and '.'

- Always start with alphabets
- No special characters like !,@,#,$,. . . .

```
# assign the value 7 to variable b2
b2 <- 7
b2
```

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

This is a valid variable name because it started with an alphabet and it has only alphanumeric characters

```
# assign the value Scientist to variable Monoj_GDPL
Manoj_GDPL = "Scientist"
Manoj_GDPL
```

```
## [1] "Scientist"
```

This is also a valid variable name because it has a special character, but it is underscore which is allowed special

character for the variable names.

```
# assign the value 7 to variable 2b
2b = 7
```

```
## Error: <text>:2:2: unexpected symbol
## 1: # assign the value 7 to variable 2b
## 2: 2b
##     ^
```

This gives an error because that variable name has started with the numeric character which is not following the rules for the names of the variables in R

Data Types

R has the following basic data types shown in the table below with the values that each data type can take.

Logical data types which take either a value of true or false,

Integer data types which is the set of all integers

Numeric data types which is set of all real numbers.

Complex variables which is a set of all the complex numbers.

Character data type where you have all the alphabets and special characters

Table 1: Basic Data Types

Basic Data Type	Values
Logical	TRUE / FALSE
Integer	Set of Integers (decimals) : 1.2 , 3.4 , 5.8123

Basic Data Type	Values
Numeric	Set of Real numbers (no decimals) : 1, 4 , 8 , 9
Character	Characters / Strings : “a”,“b”,“c”,“@”,“#”,“$”, "“,”*“,”1“,”2",... etc..

There are several task that can be done using data types

Find data type of object : Syntax : typeof(object)

```
# type of a numeric object
typeof(1)
```

```
## [1] "double"
```

```
# type of a character object
typeof("Ravi")
```

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

Verify if an object is of a particular type : Syntax : is.data_type(object)

```
# verify if given object is a character
is.character("Ravi")
```

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

```
# verify if given object is a number
is.numeric("Ravi")
```

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

Coerce or convert data type of object to another : Syntax : as.data_type(object)

Note : Not all coercing is possible and if attempted will return “NA” as output

```
# convert to numeric
as.numeric("123")
```

```
## [1] 123
```

```
# convert to numeric
as.numeric("a")
```

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

Objects

We have several basic objects of R, in this the most important ones are; vectors, lists and data frames

Vectors

A vector is an ordered collection of basic data types of given length. All the elements of a vector must be of same data type. The way you creating vector in R is using the concatenation command c()

```
# Exaample of a numeric vector
X <- c(2.3 , 4.6 , 5.5)
print(X)
```

```
## [1] 2.3 4.6 5.5
```

```
# Example of a character vector
Y <- c("a" , "b" , "c")
print(Y)
```

```
## [1] "a" "b" "c"
```

Lists

List is a generic object consisting of ordered collection of objects.List can be a list of vectors, list of matrices, list of characters and list of functions and so on.

We want to build a list of employees with the details for this we want the attributes such as ID, employee name and number of employees.

We are creating each vector for those attributes and combine all these three different data types into a list containing the details of employees which can be done using a list command

```
# List Example : Employee details
# Vectors of individual elemnts of the list
ID = c(1,2,3,4)
emp_name =c("Man","Rag","Sha","Din")
num_emp = c(4)

# create a list containing the vectors
emp.list = list(ID, emp_name,num_emp)

# print the list
print(emp.list)
```

```
## [[1]]
## [1] 1 2 3 4
##
## [[2]]
## [1] "Man" "Rag" "Sha" "Din"
##
## [[3]]
## [1] 4
```

All the components of a list can be named and you can use that names to access the components of the list.

Instead of directly creating a list you can also give the names for this attributes as ID, names of employees and the total staff as shown in the code here.

```r
# List Example : Employee details
# Vectors of individual elemnts of the list
ID = c(1,2,3,4)
emp_name =c("Man","Rag","Sha","Din")
num_emp = c(4)

# create a list containing the vectors with names for each vector
emp_list = list("Id" = ID,
                "Names" = emp_name,
                "Total_staff"=num_emp)

# print the list of employees
print(emp_list)
```

```
## $Id
## [1] 1 2 3 4
##
## $Names
## [1] "Man" "Rag" "Sha" "Din"
##
## $Total_staff
## [1] 4
```

We can access individual components of the list by using the "names" of the vectors in the list **list$vec_name**

```r
# print the employee names
print(emp_list$Names)
```

```
## [1] "Man" "Rag" "Sha" "Din"
```

```r
# print the employee IDs
print(emp_list$Id)
```

```
## [1] 1 2 3 4
```

We can access sub-elements from a list.To access top level components, we use double slicing operator " [[ ]]" and for lower/inner level components use "[ ]" along with "[[ ]]"

e.g. Get the 1st element of the 1st list of emp_list

```r
# print the 1st list of emp_list
print(emp_list[1])
```

```
## $Id
## [1] 1 2 3 4
```

```r
# print the 1st element of the 1st list
print(emp_list[[1]][1])
```

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

```r
# print the 2nd list from emp_list
print(emp_list[2])
```

```
## $Names
## [1] "Man" "Rag" "Sha" "Din"
```

```r
# print the 2nd element of the 2nd list
print(emp_list[[2]][2])
```

```
## [1] "Rag"
```

Concatenation of lists : Two lists can be concatenated using the concatenation function, c(list1, list2)

```r
# create a list of employee ages
emp_ages = list("ages" = c(23,45,30,32))
```

```r
# combine emp_list with emp_ages
emp_list = c(emp_list , emp_ages)
```

```r
# print the new list
print(emp_list)
```

```
## $Id
## [1] 1 2 3 4
##
## $Names
## [1] "Man" "Rag" "Sha" "Din"
##
## $Total_staff
## [1] 4
##
## $ages
## [1] 23 45 30 32
```

DataFrames

Data frame are generic data objects of R which you are used to store the tabular data. Data frames are the most popular data objects in R programming because we are comfortable in seeing the data in the tabular form.

Data frames can also be thought as matrices where each column of a matrix can be of different data type.

Let us see how to create a data frame in R.

Create Data Frames from Vectors The way you create the data frame is use the data.frame() command and then pass each of the vector elements you have created as arguments to the function data.frame()

```r
# Create the individual Vectors
vec1 = c(1,2,3)
vec2 = c("R","Scilab","Java")
vec3 = c("For prototyping","For prototyping","For Scaleup")
```

```r
# Create a dataFrame df
df = data.frame(vec1 , vec2 , vec3)

# View the data frame
df
```

```
##   vec1   vec2           vec3
## 1    1      R For prototyping
## 2    2 Scilab For prototyping
## 3    3   Java     For Scaleup
```

Create a Data Frame from a File A Dataframe can be created by reading data from a file e.g. ".csv file" using the read.table() method

We need to specify the "separator" and "header = TRUE"

```r
new_df = read.table("artists.csv" , sep = "," , header = TRUE)
```

A Dataframe can also be created using the "dplyr's" `read_csv()` function

```r
# load the tidyverse library
library(tidyverse)
```

```r
# create a dataframe
artists_df = read_csv("artists.csv")
```

Inspecting a Data-Frame The `head()` function returns the first 6 rows of a data frame. If you want to see more rows, you can pass an additional argument `n` to `head()`. For example, `head(df,8)` will show the first 8 rows.

The glimpse() function returns the structure of the dataframe along with sample observations

The function `summary()` will return summary statistics such as mean, median, minimum and maximum for each numeric column while providing class and length information for non-numeric columns.

```r
# inspect top 6 rows
head(artists_df)
```

```
## # A tibble: 6 x 7
##   group   country   genre spotify_monthly_~ youtube_subscri~ year_founded albums
##   <chr>   <chr>     <chr>             <dbl>            <dbl>        <dbl>  <dbl>
## 1 Imagin~ United S~ Rock           37830079         16710940         2008      4
## 2 BTS     South Ko~ K-Pop           8409314         15625947         2013      6
## 3 Maroon~ United S~ Rock           35215180         24071114         1994      6
## 4 Migos   United S~ Hip ~          20929342          8015917         2008      3
## 5 Coldpl~ United K~ Rock           27810924         13891749         1996      7
## 6 U2      Ireland   Rock           11490382          1423636         1997     14
```

```r
# inspect the structure of the dataframe
glimpse(artists_df)
```

```
## Rows: 7
## Columns: 7
## $ group                      <chr> "Imagine Dragons", "BTS", "Maroon 5", "Migos~
## $ country                    <chr> "United States", "South Korea", "United Stat~
## $ genre                      <chr> "Rock", "K-Pop", "Rock", "Hip Hop", "Rock", ~
## $ spotify_monthly_listeners <dbl> 37830079, 8409314, 35215180, 20929342, 27810~
## $ youtube_subscribers        <dbl> 16710940, 15625947, 24071114, 8015917, 13891~
## $ year_founded               <dbl> 2008, 2013, 1994, 2008, 1996, 1997, 1962
## $ albums                     <dbl> 4, 6, 6, 3, 7, 14, 25
```

```
# summary statistics
summary(artists_df)
```

```
##     group             country             genre
##  Length:7           Length:7           Length:7
##  Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character
##
##
##
##  spotify_monthly_listeners youtube_subscribers  year_founded      albums
##  Min.   : 8409314          Min.   : 1423636    Min.   :1962   Min.   : 3.00
##  1st Qu.:12187214          1st Qu.: 4777775    1st Qu.:1995   1st Qu.: 5.00
##  Median :20929342          Median :13891749    Median :1997   Median : 6.00
##  Mean   :22081324          Mean   :11611277    Mean   :1997   Mean   : 9.29
##  3rd Qu.:31513052          3rd Qu.:16168444    3rd Qu.:2008   3rd Qu.:10.50
##  Max.   :37830079          Max.   :24071114    Max.   :2013   Max.   :25.00
```

String Handling

We will learn to work with strings. For this we will analyse one of my favorite books: George Orwell's 1984.

Objectives: Learn string handling, e.g. functions _grep()_, _gsub()_, _nchar()_, _strsplit()_, and many more

Packages

Let us load the packages required for analyzing strings

```
library(tidyverse)
```

Import

Let us import the book into an object. This book is available for download. The link is stored in variable "url" and the text is downloaded with _readLines()_ and save in "text_1984".

```
url <- "http://gutenberg.net.au/ebooks01/0100021.txt"
text_1984 <- readLines(url)
```

Filtering

The book has some overhead: introductory text at the beginning and some appendix at the end. We want to analyse the pure book, so we filter the text to its core

We will extract the text from line#47 till the end of the text and store this in a new object text_1984_filt. Note the index of the last element of the text = length of the text

```
# filter the desired text
text_1984_filt <- text_1984[46: length(text_1984)]
text_1984_filt <- text_1984_filt[1:9859]
```

```
# look at the top 6 rows
head(text_1984_filt)
```

```
## [1] "It was a bright cold day in April, and the clocks were striking thirteen."
## [2] "Winston Smith, his chin nuzzled into his breast in an effort to escape the"
## [3] "vile wind, slipped quickly through the glass doors of Victory Mansions,"
## [4] "though not quickly enough to prevent a swirl of gritty dust from entering"
## [5] "along with him."
## [6] ""
```

```
# look at the bottom 6 rows
tail(text_1984_filt)
```

```
## [1] ""        ""        "THE END" ""        ""        ""
```

What is the structure of the object?

```
str(text_1984_filt)
```

```
##  chr [1:9859] "It was a bright cold day in April, and the clocks were striking thirteen." ...
```

It is a character vector with 9865 elements. There is just one problem. Some elements contain several words, some don't contain a single word. Our aim is to have a vector with a single word as each element.

We will now convert this into one single string i.e. concatenate all the elements together

Concatenate with paste()

First, we collapse this vector to one single string. This can be done with _paste()_ function. The separators will be blank signs between the words. In a second step we modify all letters to lower letters with _str_to_lower()_

```
# collapse to a single string
text_1984_one_single_string <- paste(text_1984_filt, collapse = " ")
text_1984_one_single_string <- str_to_lower(text_1984_one_single_string)
```

You can see in the "environment" that the size is now reduced to 560KB from 1.3MB...

Separate each word str_split()

We will now separate each sentence into a word considering the fact that words are separated by "blank spaces" and get the 1st list element

```
# separate each word
text_1984_separate_words <- str_split(string = text_1984_one_single_string, pattern = " ")[[1]]
```

Finding Patterns : grep()

we are interested in analyzing words so we will exclude all numbers. We will find out with _grep()_ or _str_subset()_. These commands search for matches within the text.

But how can we define all numbers? The easy way is to run _grep()_ with parameter "0", "1", "2", ... But this takes quite some effort and contradicts DRY (don't repeat yourself principle).

There is a better way. You can use "[0-9]" for all numbers from 0 to 9. This is a regular expression.

Let us look at what are the numbers in the text. Here we use the str_subset() function with pattern = [0-9].Here [0-9] is the regular expression "regex" pattern for numbers

```
# head(grep("[0-9]", text_1984_separate_words, value = T))
head(str_subset(string = text_1984_separate_words, pattern = "[0-9]"))
```

```
## [1] "300"   "4th,"  "1984." "1984." "1944"  "1945;"
```

We can use this regular expression to remove numbers and hyphens.

We will find the patterns [0-9] and "-" and replace these with "blank spaces"

```
# delete numbers
text_1984_separate_words <- str_replace_all(pattern = "[0-9]",
                                            replacement = "",
                                            string = text_1984_separate_words)

# delete hyphens
text_1984_separate_words <- str_replace_all(pattern = "-",
                                            replacement = " ",
                                            string = text_1984_separate_words)
head(text_1984_separate_words)
```

```
## [1] "it"     "was"    "a"      "bright" "cold"   "day"
```

Delete Empty Words : str_length()

There are still empty elements, which we will delete in the next step. Empty element have zero characters, which we can find out with _str_length()_. so we filter for _str_length()_ > 0.

```
# delete empty words
text_1984_separate_words <- text_1984_separate_words[str_length(text_1984_separate_words) > 0]
```

Character Occurrences

The main characters are “Winston”, “Julia”, “O’Brien” and of in a way “big brother”. with _table()_ the number of occurrences are shown. We concentrate on the main characters and filter for them with“[ ]”.

```
table(text_1984_separate_words)[c("winston", "julia", "o\'brien", "brother")]
```

```
## text_1984_separate_words
## winston   julia o’brien brother
##     315      44     120      40
```

Not surprisingly “Winston” as the main character has the most appearances.This is not sorted. We can order this table with _sort()_. Default is ascending order, but with parameter “decreasing = T” it is changed to decreasing.

```
sort(table(text_1984_separate_words)[c("winston", "julia", "o\'brien", "brother")], decreasing = T)
```

```
## text_1984_separate_words
## winston o’brien   julia brother
##     315     120      44      40
```

Shortest and Longest Word : which.max/min combined with str_length

I am curious about finding out, what the shortest word is. We already know the length of a word can be found with _str_length()_. Now, we need to find the position of maximum and use _which.max()_.

```
pos_min <- which.min(str_length(text_1984_separate_words))
text_1984_separate_words[pos_min]
```

```
## [1] "a"
```

Surprise, surprise. The shortest word is “a”. The opposite is _which.max()_. Find out for yourself what the longest word is.

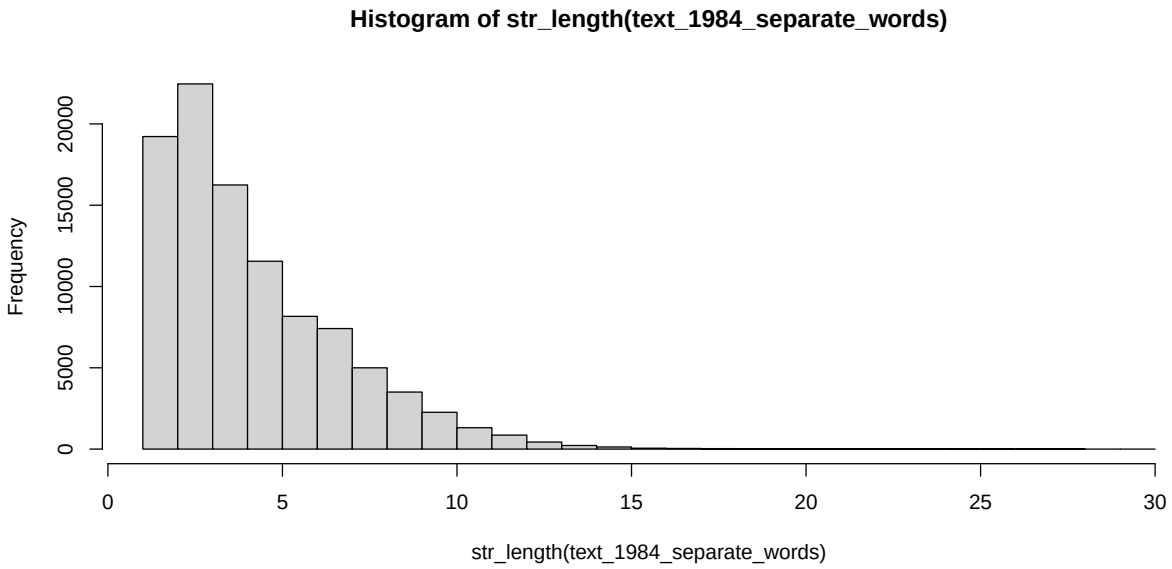
```
pos_max <- which.max(str_length(text_1984_separate_words))
text_1984_separate_words[pos_max]
```

```
## [1] "dirty mindedness  everything"
```

Word Lengths : Distribution of Words

What are the distribution of word lengths. Let’s see with hist()_ and plot a histogram.

```
hist(str_length(text_1984_separate_words), breaks = seq(1, 30, 1))
```


Histogram of str_length(text_1984_separate_words)
Frequency
0
5000
10000
15000
20000
0
5
10
15
20
25
30
str_length(text_1984_separate_words)