ECO 274 LAB: R Syntax and Data Structure

Learning Objectives

- Construct factor, matrix, data frame and data structures to store data.
- To understand the logical comparison
- Describe and utilize built-in functions in R.
- Modify default behavior of functions using arguments in R.
- Demonstrate how to create user-defined functions in R

Factor

Let's create a factor vector and explore a bit more. We'll start by creating a character vector describing five different levels of utility and temperature label:

```
utility <- c("very satisfied", "not satisfied", "satisfied", " satisfied ", "not satisfied", "unsatisfied", "unsatisfied", "very unsatisfied" )

temp_label<- c("low", "high", "medium", "high", "low", "medium", "high")
```

Now we can convert this character vector into a *factor* using the `factor()` function:

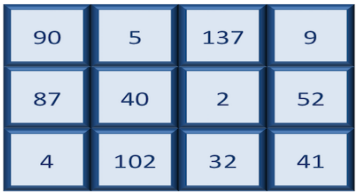
```
utility <- factor(utility)

temp_label<- factor(temp_label)
```

So, what exactly happened when we applied the `factor()` function?

Matrix

A `matrix` in R is a collection of vectors of **same length and identical datatype**. Vectors can be combined as columns in the matrix or by row, to create a 2-dimensional structure.



Matrices are used commonly as part of the mathematical machinery of statistics. They are usually of numeric datatype and used in computational algorithms to serve as a checkpoint.

```
# Generating matrix, A from one vector with all values
v <- c(2,-4,-1,5,7,0)
 A <- matrix(v,nrow=2)
# Generating matrix, A from two vectors corresponding to rows:
row1 <- c(2,-1,7); row2 <- c(-4,5,0)
( A <- rbind(row1, row2) )
# Generating matrix, A from three vectors corresponding to columns:
 col1 <- c(1,6); col2 <- c(2,3); col3 <- c(7,2)
( USTopChart <- cbind(col1, col2, col3) )

# Giving names to rows and columns:
colnames(USTopChart) <- c(“Drake",“ Timberlake",“ Bieber")
rownames(USTopChart) <- c(“weekTop",“totPeak")
US40Chart
```

Data Frame

A `data.frame` is the *de facto* data structure for most tabular data and what we use for statistics and plotting. A `data.frame` is similar to a matrix in that it’s a collection of vectors of the **same length** and each vector represents a column. However, in a dataframe **each vector can be of a different data type** (e.g., characters, integers, factors).

Year	sales	revenue	economy
2010	24	120	"trough"
2012	45	650	"expansion"
2014	67	980	"slowdown"
2016	78	890	"expansion"
2018	89	990	"peak"

A data frame is the most common way of storing data in R, and if used systematically makes data analysis easier.

We can create a dataframe by bringing **vectors** together to **form the columns**. We do this using the `data.frame()` function, and giving the function the different vectors we would like to bind together. *This function will only work for vectors of the same length.*

```
# Define year vector:

year     <- c(2008,2009,2010,2011,2012,2013)

# Define a matrix of product values:

product1<-c(0,3,6,9,7,8)

product2<-c(1,2,3,5,9,6)

product3<-c(2,4,4,2,3,2)

# Create a sales matrix

sales_mat <- cbind(product1,product2,product3)

## Give row names

rownames(sales_mat) <- year

# The matrix looks like this:

sales_mat

# Create a data frame and display it:

sales <- as.data.frame(sales_mat)

sales

# Accessing a single variable:

sales$product2

# Generating a new variable in the data frame:

sales$totalvalue <- sales$product1 + sales$product2 + sales$product3
```

Logical Comparison

Logical operators include greater than (>), less than (<), and equal to (==). A full list of logical operators in R is displayed below:

Operator	Description
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to
==	equal to
!=	not equal to
&	and
|	or

```
3 < 4
3 > 4
#Contrast with 3 = 4; see section about variables below
3 == 4
#!= means "not equal to"
3 != 4
4 >= 5
4 <= 5
2 + 2 == 5
10 - 6 == 4
```

Basic/Built-in functions

What are functions?

A key feature of R is functions. Functions are “self contained” modules of code that accomplish a specific task. Functions usually take in some sort of data structure (value, vector, dataframe etc.), process it, and return a result. The general usage for a function is the name of the function followed by parentheses:

function_name(input)

The input(s) are called arguments, which can include:

```
sqrt(81)
round(3.14159)
round(3.14159, 2)
```

The basic (built-in) functions are available as part of R's built in capabilities, and we will explore a few more of these base functions below.

```
my_vec_<- c(88, 95, 92, 97, 96, 97, 94, 86, 91, 95, 97, 88, 85, 76, 68)
mean(my_vec)
median(my_vec)
sd(my_vec)
hist(my_vec)
range(my_vec)
sum(my_vec)
summary(my_vec)

p_value <- .034/15476587634566
p_value
```

Built-in Functions in R

There are plenty of helpful built-in functions in R used for various purposes. Some of the most popular ones are:

- min(), max(), mean(), median() – return the minimum / maximum / mean / median value of a numeric vector, correspondingly
- sum() – returns the sum of a numeric vector
- range() – returns the minimum and maximum values of a numeric vector
- abs() – returns the absulute value of a number
- str() – shows the structure of an R object
- print() – displays an R object on the console
- ncol() – returns the number of columns of a matrix or a dataframe
- length() – returns the number of items in an R object (a vector, a list, etc.)
- nchar() – returns the number of characters in a character object
- sort() – sorts a vector in ascending or descending (decreasing=TRUE) order
- exists() – returns TRUE or FALSE depending on whether or not a variable is defined in the R environment

User-defined Functions

One of the best ways to improve your reach as a data scientist is to write functions. Functions allow you to automate common tasks in a more powerful and general way than copy-and-pasting. Writing a function has three big advantages over using copy-and-paste:

- You can give a function an evocative name that makes your code easier to understand.
- As requirements change, you only need to update code in one place, instead of many.
- You eliminate the chance of making incidental mistakes when you copy and paste (i.e. updating a variable name in one place, but not in another).

Writing good functions is a lifetime journey. Even after using R for many years, you still learn new techniques and better ways of approaching old problems.

Function Components

The different parts of a function are −

Function Name − This is the actual name of the function. It is stored in R environment as an object with this name.

Arguments − An argument is a placeholder. When a function is invoked, you pass a value to the argument. Arguments are optional; that is, a function may contain no arguments. Also arguments can have default values.

Function Body − The function body contains a collection of statements that defines what the function does.

Return Value − The return value of a function is the last expression in the function body to be evaluated.

The structure of a function is given below:

```
name_of_function <- function(argument1, argument2) {
    statements or code that does something
    return(something)
}
```

When defining the function you will want to provide the list of arguments required (inputs and/or options to modify behavior of the function), and wrapped between curly brackets place the tasks that are being executed on/using those arguments. The argument(s) can be any type of object (like a scalar, a matrix, a dataframe, a vector, a logical, etc), and it's not necessary to define what it is in any way.

Finally, you can "return" the value of the object from the function, meaning pass the value of it into the global environment. The important idea behind functions is that objects that are created within the function are local to the environment of the function – they don't exist outside of the function.

Let's try creating a simple example function. This function will take in a numeric value as input, and return the squared value.

```
square_it <- function(x) {
    square = x * x
    return(square)
}

square_it(5)
```

```
my_love <- function(myLove = "Disney") {
  paste("I like to visit ", myLove)
}

my_love("Niagara Falls")
my_love("Statue of Liberty")
my_love() # will get the default value, which is Disney
my_love("The Grand Canyon")
my_love("Chicago")
```

Let's try more:

```
fahr_to_cel <- function(temp_F) {
  temp_C <- (temp_F - 32) * 5 / 9
  return(temp_C)
}

fahr_to_cel(32)
fahr_to_cel(78)
```

```
new.function_1 <- function(a) {
  for(i in 1:a) {
    b <- i^2
    print(b)
  }
}

# Call the function new.function supplying 6 as an argument.
new.function_1(6)
```

Also,

```
# Create a function with arguments.
new.function_2 <- function(a,b,c) {
  result <- a * b + c
  print(result)
}

# Call the function by position of arguments.
new.function_2(5,3,11)

# Call the function by names of the arguments.
new.function_2(a = 11, b = 5, c = 3)
```

```
mean_median <- function(vector){
    mean <- mean(vector)
    median <- median(vector)
    return(c(mean, median))
}
print(mean_median(c(1, 1, 1, 2, 3)))
```

```
add_or_subtract <- function(first_num, second_num, type = "add") {

  if (type == "add") {
    first_num + second_num
  } else if (type == "subtract") {
    first_num - second_num
  } else {
    stop("Please choose `add` or `subtract` as the type.")
  }

}


add_or_subtract(10, 6, type = "add")

add_or_subtract(5, 6, type = "subtract")

add_or_subtract(5, 6, type = "multiply")
```

Reference and acknowledgement:

1. The materials used in this lesson are adapted from work that is Copyright © Data Carpentry (http://datacarpentry.org/), Harvard Chan Bioinformatics Core (HBC) under the open access terms of the Creative Commons Attribution license (CC BY 4.0), which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

2. The Book of R: A First Course in Programming and Statistics by Tilman M. Davies