Applied Economic Forecasting

Tutorial on R Studio III

Spring 2020

VIRGINIA TECH

Data types in R

Here we discuss various types of data in R. R works with numerous data types. Some of the most basic types to get started are:

1. **Decimal values** like 4.5 are called numerics.
2. **Natural numbers** like 4 are called integers. Integers are also numerics.
3. **Boolean values** (TRUE or FALSE) are also called logical.
4. **Text** (or string) values are called characters.

You can check the type of data by using `class()`.

```
x <- "Melody to Old Town Road!"
class(x)
```

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

VIRGINIA TECH

```r
x <- c("TRUE", "FALSE")
x <- as.logical(x) #Declare the data type
class(x)
```

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

```r
x <- 1:20
x %% 4 #x mod 4
```

```
##  [1] 1 2 3 0 1 2 3 0 1 2 3 0 1 2 3 0 1 2 3 0
```

```r
x %% 4 == 0
```

```
##  [1] FALSE FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE
## [13] FALSE FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE
```

```r
class(x %% 4 == 0)
```

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

A collection of data points

Vectors

A vector is the most common and basic data type in R, and is pretty much the workhorse of R. A vector is composed by a series of values, which can be either numbers or characters. We can assign a series of values to a vector using the c() function.

Here `c()` stands for **concatenate** or **combine**.

VIRGINIA TECH

Some basic examples

```
(v <- c(1, 2, 3, 4))
```

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

```
(v <- 1:4)
```

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

```
(v <- seq(from = 0, to = 0.5, by = 0.1))
```

```
## [1] 0.0 0.1 0.2 0.3 0.4 0.5
```

```
#A vector can also contain characters:
(v_colors <- c("blue", "yellow", "light green") )
```

```
## [1] "blue"        "yellow"      "light green"
```

VIRGINIA TECH

Subsetting vectors (Indexing/reassigning elements)

```
v_colors[2]
```

```
## [1] "yellow"
```

```
v_colors[c(1, 3)]
```

```
## [1] "blue"        "light green"
```

```
(v_colors[2:3]  <- c("red", "purple")   )
```

```
## [1] "red"    "purple"
```

```
j <- c(-1,-2,-3)
x[j]
```

```
##  [1]  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

Conditional subsetting

Another common way of subsetting is by using a logical vector. `TRUE` will select the element with the same index, while `FALSE` will not. Typically, these logical vectors are not typed by hand, but are the output of other functions or logical tests such as:

```
x <- 100:110
# returns TRUE or FALSE depending on which elements
# that meet the condition
x > 105
```

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

```
select <- x > 105
x[x > 105]
```

```
## [1] 106 107 108 109 110
```

VIRGINIA TECH

You can combine multiple tests using `&` (both conditions are true, `AND`) or `|` (at **least one** of the conditions is true, `OR`):

x is between 103 and 106

```
x[x >= 103 & x <= 106]
```

```
## [1] 103 104 105 106
```

x is greater than 103 but less than or equal to 106

```
x[x <= 106 & x > 103] # order of subsetting does not matter he
```

```
## [1] 104 105 106
```

x is less than 103 `or` greater than 106

```
x[x >= 106 | x < 103]
```

```
## [1] 100 101 102 106 107 108 109 110
```

VIRGINIA TECH

Sometimes we will need to search for certain strings in a vector. With mutlpile conditions, it becomes difficult to use the "OR" operator `|`. The function `%in%` allows you to test if any of the elements of a search vector are found:

```
animals <- c("mouse", "rat", "dog", "cat")
animals[animals == "cat" | animals == "rat"] # returns both ra
```

```
## [1] "rat" "cat"
```

```
animals %in% c("rat", "cat", "dog", "duck", "goat")
```

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

```
animals[animals %in% c("rat", "cat", "dog", "duck", "goat")]
```

```
## [1] "rat" "dog" "cat"
```

Additional Resources

If you would like to get a better grasp on subsetting vectors, please feel free to try out R Intro on Datacamp. I would highly reccommend this!

Names of a vector

Let's say that we want to know which color robe each of 3 patients is wearing, we can assign names to the vector of colors.

```
v_colors
```

```
## [1] "blue"   "red"    "purple"
```

```
names(v_colors) <- c("Thomas", "Liz", "Tucker")
v_colors
```

```
##   Thomas      Liz   Tucker
##   "blue"    "red" "purple"
```

VIRGINIA TECH

Algebraic Operations of Vectors

```
x <- c(1,2,3)
y <- c(4,5,6)
# component-wise addition
x+y
```

```
## [1] 5 7 9
```

```
# component-wise multiplication
x*y
```

```
## [1]  4 10 18
```

```
# What happens to the following
y^x # or y**x
```

```
## [1]   4  25 216
```

VIRGINIA TECH

Repeating Vector in R

```
# Would this work?
c(1,2,3,4) + c(1,2)
```

```
## [1] 2 4 4 6
```

```
# Would this work?
c(1,2,3) + c(1,2)
```

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

Why the weird results?

- When you are adding vectors of unequal size, if the long one is a multiple of the short one, R automatically repeats the short one to fill in the operation.

```
2*c(1,2,3)
```

```
## [1] 2 4 6
```

VIRGINIA TECH

Matrix

Create a new matrix

```
(matrix<-matrix(1:16, nrow = 4, byrow = TRUE))
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]    5    6    7    8
## [3,]    9   10   11   12
## [4,]   13   14   15   16
```

Note that `:` means every number from 1 to 4. In the `matrix()` function:

1. The first argument is the collection of elements that `R` will arrange into the rows and columns of the matrix. Here, we use 1:16 which is a shortcut for c(1, 2, 3, 4, ... 16).
2. The argument `byrow` indicates that the matrix is filled by the rows. If we want the matrix to be filled by the columns, we use `byrow = FALSE`.
3. The argument `nrow` indicates that the matrix should have 4 rows.

Selection of Matrix Elements

```
# Extract element in the first row and second column
matrix[1, 2]
```

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

```
# Extract the entire first row and second columns
matrix[, 1:2]
```

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    5    6
## [3,]    9   10
## [4,]   13   14
```

VIRGINIA TECH

Assign dimension names to Matrix

```
rownames(matrix) <- c("Yes", "No", "Perhaps", "Maybe")
colnames(matrix) <- c("Apple", "Pear", "Banana", "Grapes")
matrix
```

```
##         Apple Pear Banana Grapes
## Yes         1    2      3      4
## No          5    6      7      8
## Perhaps     9   10     11     12
## Maybe      13   14     15     16
```

VIRGINIA TECH

Dimension of a matrix vs vector

```
x <- c(1,2,3)
matrix<-matrix(1:4, byrow = TRUE, nrow = 2)
length(x)
```

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

```
length(matrix)
```

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

```
dim(matrix)
```

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

```
dim(x)
```

```
## NULL
```

VIRGINIA TECH

Lists

`R` doesn't like vectors to have different types: `c(TRUE, 1, "Frank")` becomes `c("TRUE", "1", "Frank")`. But storing objects with different types is absolutely fundamental to data analysis. `R` has a different type of object besides a vector used to store data of different types side-by-side: a list:

```
c(TRUE, 1, "Frank")
```

```
## [1] "TRUE"  "1"     "Frank"
```

```
x <- list(TRUE, 1, "Frank")
```

Many different things not necessarily of same length can be put together.

```
x <- list(c(1:5), c("a", "b","c"), c(TRUE, FALSE), c(5L, 6L))
```

VIRGINIA TECH

Dataframes

- Data frames are like spreadsheet data, rectangular with rows and columns.
- Ideally each row represents data on a single observation and each column contains data on a single variable, or characteristic, of the observation.
- It represents the data in a tabular format where the columns are vectors that all have the same length. Because columns are vectors, each column must contain a single type of data (e.g., characters, integers, factors).
- We can open a data viewer window to see the contents of `R`'s `iris` data frame by typing

```
View(iris)
```

VIRGINIA TECH

Create a data frame

Data frame with Harry Potter characters

```
name <- c("Harry", "Ron", "Hermione", "Hagrid", "Voldemort")
height <- c(176, 175, 167, 230, 180)
gpa <- c(3.4, 2.8, 4, 2.2, 3.4)
df_students <- data.frame(name, height, gpa)
df_students
```

```
##         name height gpa
## 1      Harry    176 3.4
## 2        Ron    175 2.8
## 3   Hermione    167 4.0
## 4     Hagrid    230 2.2
## 5 Voldemort    180 3.4
```

VIRGINIA TECH

Alternative way of creating DF

```
df_students <- data.frame(name = c("Harry", "Ron",
    "Hermione", "Hagrid", "Voldemort"), height = c(176,
    175, 167, 230, 180), gpa = c(3.4, 2.8, 4, 2.2,
    3.4))
df_students
```

```
##         name height gpa
## 1      Harry    176 3.4
## 2        Ron    175 2.8
## 3   Hermione    167 4.0
## 4     Hagrid    230 2.2
## 5 Voldemort    180 3.4
```

VIRGINIA TECH

Adding variable

```
df_students$good <- c(1, 1, 1, 1, 0)
df_students
```

```
##         name height gpa good
## 1      Harry    176 3.4    1
## 2        Ron    175 2.8    1
## 3   Hermione    167 4.0    1
## 4     Hagrid    230 2.2    1
## 5 Voldemort    180 3.4    0
```

VIRGINIA TECH

Features of the DF

```
dim(df_students)
df_students[2, 3]  #Ron's GPA
df_students$gpa[2]  #Ron's GPA
df_students[5, ]  #get row 5
df_students[3:5, ]  #get rows 3-5
df_students[, 2]  #get column 2 (height)
df_students$height  #get column 2 (height)
df_students[, 1:3]  #get columns 1-3
df_students[4, 2] <- 255  #reassign Hagrid's height
df_students$height[4] <- 255  #same thing as above
df_students
```

VIRGINIA TECH