Session 4
R Basics

R for Social Data Science

Jeffrey Ziegler, PhD

Assistant Professor in Political Science & Data Science
Trinity College Dublin

Fall 2022

Road map for today

Last time:

- R operators and objects
- Data structures and types

This week:

- Indexing and subsetting
- Attributes

Vector indexing and subsetting

- Indexing in R starts from **1** (as opposed to 0 in Python)
- To subset a vector, use ‘[]‘ to index the elements you would like to select:

```
dbl_vec <- c(300,200,4)
dbl_vec[1]
```

```
[1] 300
```

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

```
[1] 300 4
```

Summary of vector subsetting

Value	Example	Description
Positive integers	‘v[c(3, 1)]‘	Returns elements at specified positions
Negative integers	‘v[-c(3, 1)]‘	Omits elements at specified positions
Logical vectors	‘v[c(FALSE, TRUE)]‘	Returns elements where corresponding logical value is ‘TRUE‘
Character vector	‘v[c(“c”, “a”)]‘	Returns elements with matching names (only for named vectors)
Nothing	‘v[]‘	Returns the original vector
0 (Zero)	‘v[0]‘	Returns a zero-length vector

Generating sequences for subsetting

- You can use ‘:‘ operator to generate vectors of indices for subsetting
- We briefly talked about ‘seq()‘ function, which provides a generalization of ‘:‘ for generating arithmetic progressions

```
2:4
```

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

```
# Similar to Python’s object[start:stop:step] syntax
seq(from=1,to=4,by=2)
```

```
[1] 1 3
```

Vector subsetting examples

```
v
```

```
[1] 8 10 12 14
```

```
v[2:4]
```

```
[1] 10 12 14
```

```
# Argument names can be omitted for matching by position
v[seq(1,4,2)]
```

```
[1] 8 12
```

```
# All but the last element
v[-length(v)]
```

```
[1] 8 10 12
```

```
# Reverse order
v[seq(length(v),1,-1)]
```

```
[1] 14 12 10  8
```

Vector recycling

For operations that require vectors to be of the same length R recycles (reuses) the shorter one

```r
c(0,1)+c(1,2,3,4)
```

```
[1] 1 3 3 5
```

```r
5*c(1,2,3,4)
```

```
[1]  5 10 15 20
```

```r
c(1,2,3,4)[c(TRUE,FALSE)]
```

```
[1] 1 3
```

‘WHICH()‘ FUNCTION

Returns indices of TRUE elements in a vector

```r
char_vec <- c("apple", "banana", "watermelon")
char_vec
```

```
[1] "apple"      "banana"     "watermelon"
```

```r
char_vec=="watermelon"
```

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

```r
which(char_vec=="watermelon")
```

```
[1] 3
```

```r
dbl_vec[char_vec=="watermelon"]
```

```
[1] 4
```

```r
dbl_vec[which(char_vec=="watermelon")]
```

```
[1] 4
```

Lists

- Opposed to vectors, *lists* can contain elements of any type
- List can also have nested lists within it
- Lists are constructed using ‘list()‘ function in R

```r
# We can combine different data types in a list and, optionally, name
combined_l <- list(2:4,"a", B=c(TRUE,FALSE,FALSE), list("x",1L))
```

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

[[2]]
[1] \"a\"

$B
[1]  TRUE FALSE FALSE

[[4]]
[[4]][[1]]
[1] \"x\"

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

R object structure

- ‘str()‘ - one of the most useful functions in R
- It shows the **str**ucture of an arbitrary R object

```
str(l)
```

```
List of 4
$  : int [1:3] 2 3 4
$  : chr \"a\"
$ B: logi [1:3] TRUE FALSE FALSE
$  :List of 2
..$ : chr \"x\"
..$ : int 1
```

List subsetting

- As with vectors you can use ‘[]‘ to subset lists
- This will return a list of length one
- Components of the list can be individually extracted using ‘[[‘ and ‘$‘ operators

```
list[index]
```

```
list[[index]]
```

```
list$name
```

List subsetting examples

```r
l[3]
```

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

```r
str(l[3])
```

```
[1] List of 1
$ B: logi [1:3] TRUE FALSE FALSE
```

```r
l[[3]][1]
```

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

```r
# Only works with named elements
l$B
```

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

ATTRIBUTES

- All R objects can have attributes that contain metadata about them
- Attributes can be thought of as named lists
- Names, dimensions and class are common examples of attributes
- They (and some other) have special functions for getting and setting them
- More generally, attributes can be accessed and modified individually with ‘attr()‘ function

Attributes examples

```r
v
```

```
[1] 8 10 12 14
```

```r
attr(v, "example_attribute") <- "This is a vector"
attr(v, "example_attribute")
```

```
[1] "This is a vector"
```

```r
# To set names for vector elements we can use names() function
names(v) <- c("a","b","c","d")
v
```

```
[1] a  b  c  d
8 10 12 14
attr(,"example_attribute")
[1] "This is a vector"
```

```r
# Names of vector elements can be used for subsetting
v["b"]
```

Factors

- Factors form the basis of categorical data analysis in R
- Values of nominal (categorical) variables represent categories rather than numeric data
- Examples are abundant in social sciences (gender, party, region, etc.)
- Internally, in R factor variables are represented by integer vectors
- With 2 additional attributes:
  - ‘class()‘ attribute which is set to ‘factor‘
  - ‘levels()‘ attribute which defines allowed values"

Factors example

```r
cities <- c("Dublin","Cork","Cork","Limerick","Galway")
cities
```

```
[1] "Dublin" "Cork" "Cork" "Limerick" "Galway"
```

```r
typeof(cities)
```

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

```r
# We use factor() function to convert character vector into factor
# Only unique elements of character vector are considered as a level
cities <- factor(cities)
cities
```

```
[1] Dublin Cork Cork Limerick Galway
Levels: Cork Dublin Galway Limerick
```

Factors example continued

```r
class(cities)
```

```
[1] "factor"
```

```r
# Data type of this vector is integer (and not character)
```

```
[1] "integer"
```

Factors example continued

```r
# Note that R automatically sorted the categories alphabetically
levels(cities)
```

```
[1] "Cork"   "Dublin" "Galway" "Limerick"
```

```r
# You can change the reference category using relevel()
cities <- relevel(cities,ref="Dublin")
levels(cities)
```

```
[1] "Dublin" "Cork" "Galway" "Limerick"
```

Factors example continued

```r
# Or define an arbitrary ordering of levels using levels argument
cities <- factor(cities,levels=c("Limerick","Galway","Dublin","Cork"))
```

```
[1] "Limerick" "Galway" "Dublin" "Cork"
```

```r
# Under the hood factors continue to be integer vectors
as.integer(cities)
```

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

Tabulation

- ‘table()‘ function is very useful for describing discrete data
- It can be used for:
  - Tabulating a single variable
  - creating contingency tables (crosstabs)
- Implicitly, R treats tabulated variables as factors

```r
var_1 <- sample(c("a","b","c"), size=50, replace=TRUE)
var_2 <- sample(c(1,2,3), size=50, replace=TRUE)
table(var_1,var_2)
```

```
[1]     var_2
var_1    1 2 3
a 7 4 5
b 6 6 5
c 7 2 8
```

Factors in crosstabs

```r
var_2 <- factor(var_2,levels=c(3,1,2))
table(var_2)
```

```
[1] var_2
3  1  2
18 20 12
```

```r
var_2 <- factor(var_2,levels=c(3,1,2),labels=c("Three","One","Two"))
table(var_1,var_2)
```

```
[1] var_2
var_1   Three One Two
a       5   7   4
b       5   6   6
c       8   7   2
```

ARRAYS AND MATRICES

- Arrays are vectors with an added class and dimensionality attribute
- These attributes can be accessed using ‘class()‘ and ‘dim()‘ functions
- Arrays can have an arbitrary number of dimensions
- Matrices are special cases of arrays that have just two dimensions
- Arrays and matrices can be created using ‘array()‘ and ‘matrix()‘ functions
- Or by adding dimension attribute with ‘dim()‘ function"

Array example

```r
# : operator can be used generate vectors of sequential numbers
a <- 1:12
a
```

```
[1]  1  2  3  4  5  6  7  8  9 10 11 12
```

```r
class(a)
```

```
[1] "integer"
```

Array example

```r
dim(a) <- c(3,2,2)
a
```

```
[1] 1
[,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6

, , 2

[,1] [,2]
[1,]    7   10
[2,]    8   11
[3,]    9   12
```

```r
class(a)
```

```
[1] "array"
```

Matrix example

```r
m <- 1:12
dim(m) <- c(3,4)
```

```
[1] [,1] [,2] [,3] [,4]
[1,] 1    4    7    10
[2,] 2    5    8    11
[3,] 3    6    9    12
```

```r
# Alternatively, we could use matrix() function
m <- matrix(1:12,nrow=3,ncol=4)
```

```
[1] [,1] [,2] [,3] [,4]
[1,] 1    4    7    10
[2,] 2    5    8    11
[3,] 3    6    9    12
```

```r
# Note that length() function displays the length of underlying vector
length(m)
```

```
[1] 12
```

ARRAY AND MATRIX SUBSETTING

- Subsetting higher-dimensional (> 1) structures is a generalisation of vector subsetting
- But, since they are built upon vectors there is a nuance (albeit uncommon)
- They are usually subset in 2 ways:
  - With multiple vectors, where each vector is a sequence of elements in that dimension
  - With 1 vector, in which case subsetting happens from the underlying vector

```
array[vector_1, vector_2, ..., vector_n]
```

```
array[vector]
```

Array subsetting example

```r
a
```

```
, , 1

     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6

, , 2

     [,1] [,2]
[1,]    7   10
[2,]    8   11
[3,]    9   12
```

```r
# Most common way
a[1,2,2]
```

```
[1] 10
```

Array subsetting example

```r
# Specifying drop = FALSE after indices retains original dimensions
a[1,2,2,drop=FALSE]
```

```
[1] , , 1

[,1]
[1,]   10
```

```r
# Here elements are subset from underlying vector (with repetition)
a[c(1,2,2)]
```

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

Matrix subsetting example

```r
m
```

```
[1]  [,1] [,2] [,3] [,4]
[1,] 1    4    7    10
[2,] 2    5    8    11
[3,] 3    6    9    12
```

```r
# Drop = FALSE prevents from this object being collapsed
m[,1,drop=FALSE]
```

```
[1] [,1]
[1,] 1
[2,] 2
[3,] 3
```

```r
# Subset all rows, first two columns
m[1:nrow(m),1:2]
```

```
[1] [,1] [,2]
[1,] 1    4
[2,] 2    5
[3,] 3    6
```

R PACKAGES

- R’s flexibility comes from its rich package ecosystem
- Comprehensive R Archive Network (CRAN) is the official repository of R packages
- At the moment it contains > 18K external packages
- Use ‘install.packages(<package_name>)‘ function to install packages that were released on CRAN
- Check ‘devtools‘ package if you need to install a package from other sources (e.g. GitHub, Bitbucket, etc.)
- Type ‘library(<package_name>)‘ to load installed packages"

Help!

R has an inbuilt help facility which provides more information about any function:

- The quality of documentation varies a lot across packages
- Stackoverflow is a good resource for many standard tasks
- For custom packages it is often helpful to check the issues page on the GitHub
- E.g. for ‘ggplot2‘

Class business

Today, we talked about...

- Indexing and subsetting
- Attributes

Next week...

- Control flow in R
  - Algorithms
  - Conditional statements
  - Loops and Iteration