

Quantitative Content Analysis: Lecture 3

Olga Kononykhina

22 Februar 2017

Today's Outline

Intro to R (Part II)

- Vectors, matrices, data frames, lists
- Working with data sets
- Loops and conditions
- Creating and applying functions

Vectors

A vector is a container of objects put together in an order.

```
# Define a vector
```

```
a <- c(1,4,5)
```

```
b <- c(3,6,7)
```

```
# Join multiple vectors
```

```
ab <- c(a,b)
```

```
# Find vector length (number of its elements)
```

```
length(a)
```

```
# Reference vector components
```

```
ab[4] # Index one element in vector
```

```
ab[4:6] # Index several elements in a vector
```

```
ab[ab==6] # Index with condition
```

Operations on vectors

Operation	Meaning
<code>sort(x)</code>	sort a vector
<code>sum(x)</code>	sum of vector elements
<code>mean(x)</code>	arithmetic mean
<code>median(x)</code>	median value
<code>var(x)</code>	variance
<code>sd(x)</code>	standard deviation
<code>factorial(x)</code>	factorial of a number

Task 1

Calculate the mean of the vector 1,99,3,4,5,6,8. What's the mean if you out the 'outlier'?

Task 1 (solution)

Calculate the mean of the vector 1,99,3,4,5,6,8. What's the mean if you out the 'outlier'?

```
# Defining vector using sequence between 3 and 6
```

```
a <- c(1,99,3:6,8)
```

```
mean(a)
```

```
## [1] 18
```

```
# Calculate the mean of a but exclude the highest number
```

```
mean(a[a!=max(a)])
```

```
## [1] 4.5
```

Matrices

A Matrix is a square 2 dimensional container, i.e. vectors combined by row or column

- Must specify number of rows and columns

`matrix(x,nrow,ncol,byrow)`

- `x`: vector of length `nrow*ncol`
- `nrow`: number of rows
- `ncol`: number of columns
- `byrow`: TRUE or FALSE, specifies direction of input

Task 2

Assign a 6×10 matrix with $1, 2, 3, \dots, 60$ as the data.

Task 2 (solution)

Assign a 6×10 matrix with 1,2,3,...,60 as the data.

```
m <- matrix(1:60, nrow=6, ncol=10)
m
```

##		[,1]	[,2]	[,3]	[,4]	[,5]	[,6]	[,7]	[,8]	[,9]	[,10]
##	[1,]	1	7	13	19	25	31	37	43	49	55
##	[2,]	2	8	14	20	26	32	38	44	50	56
##	[3,]	3	9	15	21	27	33	39	45	51	57
##	[4,]	4	10	16	22	28	34	40	46	52	58
##	[5,]	5	11	17	23	29	35	41	47	53	59
##	[6,]	6	12	18	24	30	36	42	48	54	60

Referencing matrices

- Like vectors, you can reference matrices by elements
 - $a[i, j]$ refers to the i th row, j th column element of object a
- Can also reference rows/columns, these are vectors
 - $a[i,]$ is i th row, $a[, j]$ is j th column

Task 3

Extract the 9th column of the matrix from the previous problem. How can you find the 4th element in the 9th column?

Task 3 (solution)

Extract the 9th column of the matrix from the previous problem. How can you find the 4th element in the 9th column?

```
m[,9]
```

```
## [1] 49 50 51 52 53 54
```

```
m[4,9]
```

```
## [1] 52
```

```
m[,9][4]
```

```
## [1] 52
```

Data frames

Data frames are a two-dimensional container of vectors with the same length. Each column (vector) can be of a different class and can be indexed with `[,]` or `$`. You can use functions like `nrow()`, `ncol()`, `dim()`, `colnames()`, or `rownames()` get information about your df.

```
# Combine two vectors into a data frame
```

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

```
name <- c('John', 'Paul', 'George', 'Ringo')
```

```
df <- data.frame(number, name, stringsAsFactors = FALSE)
```

```
df
```

```
##   number   name
```

```
## 1      1   John
```

```
## 2      2   Paul
```

```
## 3      3 George
```

```
## 4      4  Ringo
```

Lists

A list is an object containing other objects that can have different lengths and classes.

```
# Create a list with three objects of different lengths
list1 <- list(beatles = c('John', 'Paul', 'George', 'Ringo'),
             alive = c('Paul', 'Ringo'), albums = 1:13)
list1
```

```
## $beatles
## [1] "John"    "Paul"    "George"  "Ringo"
##
## $alive
## [1] "Paul"    "Ringo"
##
## $albums
## [1] 1 2 3 4 5 6 7 8 9 10 11 12 13
```

R's build-in data sets

There are a number of example data sets available within R.

```
# List internal data sets:  
data()
```

```
# Load swiss data set:  
data(swiss)  
# Find data description:  
?swiss
```

Importing data

You can read data and assign it to an object. The most frequently used functions to read data include:

- `load()`: To open `.RData` files
- `read.csv()`: Reads csv file
- `read.table()`: Is more general and allows to set separators
- `read.dta()`: From foreign library, used to read Stata files

Exporting data

You can export your data in various formats:

- `save()`: Only readable in R, but can store multiple objects
- `write.csv()`: Writes matrix/dataframe to csv
- `write.table`: Writes matrix to a tab delimited text file
- `write.dta()`: Writing in Stata format, requires foreign library

For() loops

For() loops are used to loop around a vector/matrix to do something.

```
m <- matrix(1:5, nrow=1, ncol=5)
m
```

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

```
for (j in 1:3){
  m[,j]=0
}
m
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    0    0    0    4    5
```

For() loops (II)

You can also 'nest' a for() loop in another for() loop

```
m <- matrix(1:15, nrow=3, ncol=5)
for (i in 1:2){
  for (j in 1:4){
    m[i,j]=0
  }
}
m
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    0    0    0    0   13
## [2,]    0    0    0    0   14
## [3,]    3    6    9   12   15
```

Task 4

Create a matrix with `matrix(1:20,nrow=4,ncol=5)` and another with `matrix(NA,nrow=4,ncol=5)`. Adapt the second to the first matrix using `for()`

- hint: define a 'counter' variable that increments by 1 each time you move to the next cell

Task 4 (solution)

Create a matrix with `matrix(1:20,nrow=4,ncol=5)` and another with `matrix(NA,nrow=4,ncol=5)`. Adapt the second to the first matrix using `for()`

```
counter=1
m1 <- matrix(1:20,nrow=4,ncol=5)
m2 <- matrix(NA,nrow=4,ncol=5)
for (j in 1:5){
  for (i in 1:4){
    m2[i,j]=counter
    counter=counter+1
  }
}
```

Task 4 (alternative solution)

Create a matrix with `matrix(1:20,nrow=4,ncol=5)` and another with `matrix(NA,nrow=4,ncol=5)`. Adapt the second to the first matrix using `for()`

```
m1 <- matrix(1:20,nrow=4,ncol=5)
m2 <- matrix(NA,nrow=4,ncol=5)
for (j in 1:5){
  for (i in 1:4){
    m2[i,j]=m1[i,j]
  }
}
```

If() statements

If() statements are used to make conditions on executing some code. If condition is true, action is done.

```
a <- 3
b <- 4
number <- 0
if(a<b){
  number=number+1
}
number
```

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

Tests for conditions: ==; >; <; >; <; !=

Task 5

Create the two objects `a <- sample(c(4,0),20,replace=TRUE)` and `m <- matrix(a,nrow=4,ncol=5)`. Now recode all the 4s into 1s using `if()` and `for()` statements.

Task 5 (solution)

Create the objects `a <- sample(c(4,0),20,replace=TRUE)` and `b <- matrix(a,nrow=4,ncol=5)`. Now recode all the 4s into 1s in `b` using `if()` and `for()` statements.

```
a <- sample(c(4,0),20,replace=TRUE)
b <- matrix(a,nrow=4,ncol=5)

for (j in 1:5){
  for (i in 1:4){
    if (b[i,j]==4){
      b[i,j]=1
    }
  }
}
```

Creating Functions

If you want to repeat an operation it is often useful to create your own Function. Functions have names, inputs and outputs and simplify your code.

```
# Find the sample mean of a vector
fun_mean <- function(x){
  sum(x) / length(x)
}

## Find the mean
data(swiss)
fun_mean(x = swiss$Infant.Mortality)

## [1] 19.94255
```

Task 6

Write a function that takes a number and doubles it.

Task 6 (solution)

Write a function that takes a number and doubles it.

```
double <- function(x){  
  output <- x * 2  
  output  
}  
double(8)
```

```
## [1] 16
```

Apply function

Apply allows you to apply a function to every row or every column. This can be done with a `for()` loop, but `apply()` is usually much faster and simpler. `apply()` takes the following form: `apply(X, MARGIN, FUN, ...)`.

```
m <- matrix(c(1:10, 11:20), nrow = 10, ncol = 2)
# mean of the rows
apply(m, 1, mean)
```

```
## [1] 6 7 8 9 10 11 12 13 14 15
```

```
# mean of the columns
apply(m, 2, mean)
```

```
## [1] 5.5 15.5
```

Task 7

Load up the build-in R dataset 'iris' and use `apply()` to get the mean of the first 4 variables.

Task 7 (solution)

Load up the build-in R dataset 'iris' and use `apply()` to get the mean of the first 4 variables.

```
attach(iris)
apply(iris[,1:4], 2, mean)
```

```
## Sepal.Length Sepal.Width Petal.Length Petal.Width
##      5.843333      3.057333      3.758000      1.199333
```

Packages

You can greatly expand the number of functions by installing and loading user-created packages.

```
# Install dplyr package  
install.packages('dplyr')
```

```
# Load dplyr package  
library(dplyr)
```

You can also call a function directly from a specific package with the double colon operator (`::`).

```
Grouped <- dplyr::group_by(combined_df, character_vector)
```


Piping

Piping allows to pass a value forward to a function call and produces faster compilation and enhanced code readability. In R use %>% from the dplyr package.

Not piped:

```
values <- rnorm(1000, mean = 10)
value_mean <- mean(values)
round(value_mean, digits = 2)
```

```
## [1] 10.05
```

Piped:

```
library(dplyr)
rnorm(1000, mean = 10) %>% mean() %>% round(digits = 2)
```

```
## [1] 9.95
```

- Type `help()` (or `?`) to see documentation
- Read Wickham & Grolemund's *R For Data Science Handbook*
- Check out the help function for a couple of functions used in today's course to see 'what is what' in the documentation

Next week

- Tidy data
- Data manipulation with `dplyr`
- Visualizations with `ggplot`