Intro to R

Clara Cohen
Department of Linguistics
cpccohen@berkeley.edu
`http://linguistics.berkeley.edu/~cpccohen/`

D-Lab Workshops, Spring 2015

Contents

1 Terms and concepts

1.1 Objects

- `x <- 3`
- `3 -> pineapple`
- `MyVeryLongVariableName = 3`

1.2 Object types

Data types

- Atomic vectors
  - `3`
  - `"cat"`
- Vectors
  - `3, 5.2, 4, 0`
  - `"cat","dog","TRUE","35"`
- Dataframes

```
Nums  Things
3      "cat"
5.2    "dog"
4     "TRUE"
0       "35"
```

- (. . .)

Data classes

- Character
  - `"a","cat","big","32"` etc.
- Numeric
  - `23.1, 0, 54, 1, 5, 3` etc.
- Factor (aka categorical variable)
  - `"apple","orange","apple","orange"` etc.
- (. . .)

1.3 Commands

- Assignment: `->, <-, =`
- Functions: `<Function>( <argument>, <argument> . . . )`
- Operators: `+, -, ^, *`, etc.
- Conditions: `8 > 5, 3+5 == 8`

2 How R works

2.1 Creating objects

Assignment	`x <- 3`	
	`12 -> y`	
	`apple = "fuji"`	
Saving output of commands	`total <- x + y`	
	`applelength <- nchar(apple)`	#nchar(): 1 character arg

What happens if you type the following commands?

- `nchar(y)`
- `nchar("y")`
- `y - nchar(apple)`
- `y - nchar("apple")`
- `total = total - applelength`
- `total - x`

2.2 R Session control

Seeing objects that you've saved	`ls()`	#ls(): 0 args
Setting your working directory	`setwd("C:/Users/...")`	#setwd() : 1 arg
Learning your working directory	`getwd()`	#getwd(): 0 arg
Seeing what else is in the directory	`dir()`	# dir(): 0 arg

Quitting	`quit()`	
	`q()`	#quit() and q() are identical
Getting help	`?quit`	
	`help(quit)`	#?quit and help(quit) are identical

Change your directory to someplace user-friendly. Quit your R-session, and then re-open it. See what objects have been saved, and what their values are.

3 Vectors

3.1 Creating and inspecting vectors

Sequences	`y<-1:10`	
	`u <- seq( from = 5, to = 10, by = .23)`	#seq(): 3 args
Repetition	`w<-rep("fishsticks",3)`	#rep(): 2 args
	`q <- rep(y, 3)`	
Concatenation	`x <- c(1,2,3,4,5,6)`	#: c(): as many args as you like
	`z<-c("blue","rhinoceros","triangle")`	
	`huge <- c(675:659, z, rep("Spock",3))`	
Summarizing	`summary(y)`	#summary(): 1 arg
	`summary(z)`	
Finding length	`length(huge)`	#length(): 1 arg

Create the following vectors:

- Your name, repeated 4 times.
- The sequence of numbers from 5 to 90, in increments of 14.1. How long is it?

3.2 Vector classes

Character vectors	`z <- c("blue","rhinoceros","triangle", "triangle")`	
	`w <-rep("fishsticks",4)`	
Numeric vectors	`x <- c(1,2,3,4,5,6)`	
	`y <- 3:13`	
	`q <- rep(y, 3)`	
	`u <- seq( from = 5, to = 10, by = .23)`	
Factor vectors	`q <- as.factor(q)`	#as.factor(): 1 arg
	`w <- as.factor(w)`	
Changing vector class:	`y<- as.character(y)`	#as.character(): 1 arg
	`w <- as.character(w)`	
	`y <- as.numeric(y)`	#as.numeric(): 1 arg
	`q <- as.numeric(as.character(q))`	#Careful with as.numeric() on factors!

What does summary() do on the following vector classes?

- character (for example, w)
- numeric (for example, q)

- Factor (for example, z. You may need to turn it into a factor first.)

3.3 Vectorization

Doing the same thing to every element in a vector	`y + 3`	
	`nchar(z)`	
	`sqrt(x)`	#sqrt(): 1 numeric arg
Matching vectors element-by-element	`nchar(w) + nchar(z)`	
	`y + y`	
	`y * 2`	
Recycling smaller vectors when lengths are mismatched	`y + x`	

3.4 Not vectorization

Combining all elements in a vector in some way	`sum(y)`	#sum(): 1 numeric arg
	`mean(y)`	#mean(): 1 numeric arg
	`sd(y)`	#sd(): 1 numeric arg
	`min(y)`	#min(): 1 numeric arg
	`max(y)`	#max(y): 1 numeric arg
Sorting the vector	`sort(q)`	#sort(): 1 argument (1 optional)
	`sort(q, decreasing = TRUE)`	

- Turn y into a character vector and sort it. How are digits sorted when they are characters?
- Turn y into a numeric vector and sort it from highest to lowest.
- Sort huge in reverse alphabetical order

3.5 Combining vectors

Pasting one vector on the end of another	`c(x, y, z, w, q)`	
Getting only the elements in common, once	`intersect(x, y)`	#intersect(): 2 args
Getting all the elements in either vector, once	`union(x,y)`	# union(): 2 args

3.6 SUBSETTING VECTORS

Getting each element once	`unique(z)`	#unique(): 1 arg

All other subsets in R (vectors, dataframes, etc.) can be understood as a variation on the following syntax. Learn to love square brackets!

```
OBJECT[    ]
```

By position (aka index)	`huge[ 1 ]`	#The first element
	`huge[ length(huge) ]`	#The last element

Indexes can be vectors	`huge[ 1:5 ]`	#The first five elements
	`huge[ c(1,5) ]`	#The first and fifth elements
	`huge[ seq( from = 1, to = length(huge), by = 3) ]`	#Every third element

Find the following elements of huge:

- The 15th element
- The 12th, first, and last element, in that order.

3.6.1 Conditions

Testing equality	`5 == 5`	# NOTE THE DOUBLE == !!
	`"cat" == "cat"`	
	`"cat" == "dog"`	
Testing inequality	`10 < 10`	# “less than”
	`10 <= 11`	# “less than or equal to”
	`10 >= 12`	# “greater than or equal to”
	`10 != 10`	# “not equal to”
Testing containment	`10 %in% c(10, 11, 12)`	# %in%: in the following vector
	`"cat" %in% c("dog", 10, "rat","McCoy")`	
Vectorization and conditions	`y > 5`	#“Test each element in y for this condition"
	`huge == "Spock"`	

Logical vectors are strings of TRUE and FALSE. When you use a logical vector to subset another vector *of the same length*, you get back only those elements for which their counterparts in the logical vector have the value TRUE. Convince yourself of this:

- `logic <- c(TRUE, FALSE,TRUE, FALSE,TRUE, FALSE,TRUE, FALSE,TRUE)` #Note the capitals, which signal logical values
- `y[ logic ]` #Get every other value in y, because every other value in logic was TRUE

When you test a vector for a condition, in fact you are making use of vectorization: each element of the vector is tested for that condition. This operation returns a vector of TRUE and FALSE. Therefore, the fastest way to get the values of a vector that meet a condition, is simply to put the condition inside square brackets. Convince yourself of this:

- `y[ y > 5 ]` #Returns only the values of y greater than 5
- `huge[ huge == "triangle" ]` #Returns only the values of huge that are "triangle"
- `huge[ huge %in% c("Spock", "rhinoceros") ]` #Returns only the values of huge that are "Spock" or "rhinoceros"

Practice:

- R has a vector built in, called ‘letters.’ Pull out only the vowels. (Hint: you can think of vowels as a vector containing "a", "e", "i", "o", and "u".)
- Pull out the elements of q that are greater than 8

Combining conditions	`"cat" %in% c( "cat" , "dog") & 5 > 2`	# &: “and”
	`"cat" %in% c( "cat" , "dog") & 5 < 2`	
	`"cat" %in% c( "cat" , "dog") | 5 < 2`	# | : “or”
	`10 = 11 | 5 < 2`	

Practice:

- Pull out the elements of q that are less than 12 and also have two characters
- Pull out the elements of q that meet either of the following two conditions: they are less than 4, OR (hint hint) their square is greater than 100

4 Dataframes

Dataframes are sets of vectors that have been glued together in rows and columns. Each row is a vector, and each column is a vector.

4.1 Creating dataframes

By hand	`lets <- c("a","q","r","l","s","t","r","v", "a","a")`	
	`nums <- 53:62`	
	`df <- data.frame( letters = lets, numbers = nums )`	#data.frame(): as many args as columns
Importing	`ratings <- read.csv( "ratings.csv", header = TRUE )`	
	`crime <- read.table( "crime.csv", sep = "," )`	# See help(read.table) for full set of arguments

Create your own dataframe, with the following columns:

- The names of your immediate family members
- Their ages
- Their relation to you

Example:

```
  name  age  relation
Sophie   62    mother
  Doug   62    father
 Clara   30        me
Phoebe   33    sister
   Roy    3    nephew
Daniel   33   husband
```

4.2 Inspecting dataframes

Summarizing	`summary( df )`	
Getting size	`dim( df )`	# dim(): 1 arg
	`nrow( df )`	# nrow(): 1 dataframe arg
Seeing top	`head( df )`	
	`head( df , 3 )`	# head(): 1 obligatory, 1 optional arg

Seeing bottom	`tail( df , 3 )`	# tail(): exactly like head()
Seeing column names	`colnames( df )`	#colnames(): 1 arg
Changing column names	`colnames( df ) <- c("AwsomeLetters", "integers" )`	
	`colnames( df )[1] <- "letters"`	

Figure out the following information:

- How many rows are in ratings?
- What are the column names of ratings?
- What are the last 4 rows of crime?
- Change one of the column names in ratings.
- Using summary(), determine which columns in crime are numeric.

4.3 Subsetting dataframes

By position	`d[ 3 , 5 ]`	# TWO dimensions: [<row> , <column>]
	`d[ , 1 ]`	# [, <column>]: Give ALL rows
	`d[ 5 , ]`	# [<row> ,]: Give ALL columns
Indices can be vectors	`d[ c(1,3,5) , 2 ]`	# Give first, third, fifth row, and second column
By column name	`d$letters`	# <dataframe> $ <columnname>
	`d$integers`	
Column name in brackets	`d[ 3 , "letters" ]`	#Element in third row, “letters” column
Multiple column names at once	`d[ 1 , c( "letters" , "integers" ) ]`	#Elements in first row, in both “letters” and “integers” columns
Columns are vectors	`d$letters[ 1:3 ]`	# The first three items in the “letters” column
	`d$integers[ nrow(d) ]`	# The last item in the “integers” column
Three ways to pull out the same element	`d[ 3 , 1 ]`	#Third row, first column
	`d[ 3, "letters" ]`	# Third row, "letters" column
	`d$letters[ 3 ]`	#Third element in “letters” column

Using ratings and crime, figure out the following information:

- What is the meanFamiliarity value in the first row of ratings? Find it out in at least two ways.
- Pull out the first, eighth, and seventy-fifth word (i.e., the thing in the "Word" column), Do it in at least two ways.
- Pull out the values in the Frequency, FamilySize, and Class columns for the first row in ratings
- Pull out the murder and assault rates for the first three rows in crime

4.3.1 Subsetting dataframes by condition

You can specify which rows of a dataframe you want by giving a vector of desired rows. This vector can be a set of TRUE and FALSE values, which are specified by a condition.

- “Give me only the rows for which the “integers” column is greater than 57:”
  `d[ d$integers > 57 , ]`
- “Give me the *letters* for which the value in the “integers” column is greater than 57:”
  `d[ d$integers > 57 , "letters" ]` #COMMA!
  `d$letters[ d$integers > 57 ]` #No comma
- “Give me the *integers* for which the value in the “letters” column is "a":”
  `d[ d$letters == "a", "integers" ]` #COMMA!

  `d$integers[ d$letters == "a" ]` #No comma

- “Give me the letters for which the integer is less than 54 OR greater than 60:”
  `d[ d$integer < 54 | d$integer > 60, "letters" ]` #COMMA!
  `d$letters[ d$integer < 54 | d$integer > 60 ]` #No comma

Using crime, figure out the following information:

- The murder rate for California
- Which states have a murder rate higher than 11.25
- Which states have an assault rate less than 170, but a murder rate greater than 7.7
- Which states have an urban population percent rate that is exactly the median urban percent rate
- Which states have a rape rate that is less than the median value, but an assault rate that is higher than the median rate for assault

Using ratings, figure out the following information:

- Which words are plants (Class column)
- Which words are complex (Complex column)
- Which words are both animals (Class column) AND complex
- Create a dataframe called "animals," which contains only the animal rows of ratings

4.4 Adding columns

By fiat	`crime$greeting <- "hi"`
	`crime$numbers <- 1:nrow(crime)`
By vectorization	`crime$urban <- crime$urbanPop / 100`
	`crime$lowAssault <- crime$assault - 20`
	`crime$noAssault <- crime$assault - crime$assault]`
Referring to other columns	`crime$assaultDif <- crime$assault - mean(crime$assault)`
	`crime$murderRatio <- crime$murder / crime$assault`

Add the following columns to ratings:

- The ratio of a word’s meanSizeRating to its meanWeightRating
- The difference between a word’s length and the mean length of all the words
- The standard deviation of the word-lengths in this dataframe (this will be the same value for all rows).
- The z-score of a word’s length (i.e., the distance between its length and the mean, divided by the standard deviation of all word-lengths)

4.5 Merging dataframes

How do you unite this information into one object?

```
states1
  state.name state.abb     state.division state.region
1    Alabama        AL East South Central        South
2     Alaska        AK            Pacific         West
3    Arizona        AZ           Mountain         West
4   Arkansas        AR West South Central        South
5 California        CA            Pacific         West
```

```
6   Colorado        CO          Mountain        West
. . .

states2
  state.abb state.area center.longitude center.latitude
1        RI       1214         -71.1244         41.5928
2        DE       2057         -74.9841         38.6777
3        CT       5009         -72.3573         41.5928
4        HI       6450        -126.2500         31.7500
5        NJ       7836         -74.2336         39.9637
6        MA       8257         -71.5800         42.3645
. . .
```

If the row orders
match
	`cbind(crime, states1, states3)`
	`cbind(crime, states1[,2:4], states3[,2:9])`
		`#cbind()`: any vector or
		dataframe args.

If the row orders
don't match
	`states <- merge(states1, states2, by="state.abb")`
	`states <- merge(crime, states, by.x="state", by.y="state.name")`
		`# merge()`: magic.

Practice:

1. Merge `states` and `states3`. Save this new dataframe as `states` (yes, overwriting old `states`).
2. **Advanced**: Add a column to the `states` dataframe, which gives the difference between that state's area and the average area for that geographical region (`state.region`). (*Hint: you will need to use both* `aggregate()` (next section) *and* `merge()`.)

4.6 Summarizing data patterns

Finding mean (median, standard deviation . . .) of all the values of some factor:

```
aggregate( <Outcome column>, list( <Factor 1> , <Factor 2 >, ...), <function> )
```

- "Dear R: Please find the mean frequency for all words that are animals, and all words that are plants":
  `aggregate( ratings$Frequency, list(ratings$Class), mean )`
- "Find the median length for all words that are complex, and all words that are simplex":
  `aggregate( ratings$Length, list(ratings$Complex), median )`
- "Find the standard deviation of Frequency for all combinations of word Class and word Complexity":
  `aggregate( ratings$Frequency, list(ratings$Class, ratings$Complex), sd )`

Practice:

- What is the mean Length of animal words and of plant words?

Counting up the number of observations:

```
xtabs( ~ <Factor 1> + <Factor 2> ...)
```

- "Dear R: How many words are plants, and how many are animals?"
  ```
  xtabs( ~Class, data = ratings )
  ```
- "What is the breakdown of observations for all combinations of Class and by Complexity?"
  ```
  xtabs( ~ Class + Complex, data = ratings )
  ```
- "How many states have more than the mean value of murders?"
  ```
  xtabs( ~ assault > mean( assault ), data = crime )       #Returns TRUE/FALSE
  counts
  ```

5 (Simple) plots

Scatterplots	`plot( murder ~ assault, data = crime )`
	`plot( meanFamiliarity ~ Frequency, data = ratings )`
Box and whisker plots	`plot( meanSizeRating ~ Class, data = ratings)`