R for Beginners INTENSIVE
Day 1: Importing data and manipulating objects

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

D-lab Workshops, Spring 2015

Contents

1 Terms and concepts 2
  1.1 Objects .................................................. 2
  1.2 Object types .............................................. 2
  1.3 Commands .................................................. 2

2 How R works 2
  2.1 Creating objects ........................................... 2
  2.2 R Session control ......................................... 3

3 Vectors 3
  3.1 Creating and inspecting vectors .......................... 3
  3.2 Vector classes .............................................. 3
  3.3 Vectorization ................................................. 4
  3.4 Not vectorization ........................................... 4
  3.5 Combining vectors ........................................... 4
  3.6 SUBSETTING VECTORS ........................................ 4

4 Dataframes 5
  4.1 Creating dataframes ....................................... 5
  4.2 Inspecting dataframes ..................................... 5
  4.3 Subsetting and sorting dataframes ....................... 6

5 Subsetting by condition 7
  5.1 Conditions .................................................. 7
  5.2 Subsetting vectors by condition .......................... 7
  5.3 Subsetting dataframes by condition ....................... 8

6 Advanced dataframe manipulations 8
  6.1 Adding columns ............................................. 8
  6.2 Summarizing data patterns ................................ 9
  6.3 Merging dataframes ....................................... 9
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

<table>
<thead>
<tr>
<th>Nums</th>
<th>Things</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>&quot;cat&quot;</td>
</tr>
<tr>
<td>5.2</td>
<td>&quot;dog&quot;</td>
</tr>
<tr>
<td>4</td>
<td>&quot;TRUE&quot;</td>
</tr>
<tr>
<td>0</td>
<td>&quot;35&quot;</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Assignment</th>
<th>x &lt;- 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>12 -&gt; y</td>
</tr>
<tr>
<td></td>
<td>apple = &quot;fuji&quot;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Saving output of commands</th>
<th>total &lt;- x + y</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>applelength &lt;- nchar(apple)</td>
</tr>
</tbody>
</table>

#nchar(): 1 character arg

What happens if you type the following commands?

1. nchar(y)  
2. nchar("y")  
3. y - nchar(apple)  
4. y - nchar("apple")  
5. total = total - applelength  
6. total - x
2.2 R Session control

| Seeing objects that you’ve saved | ls() |
| Setting your working directory | setwd("C:/Users/...") |
| Learning your working directory | getwd() |
| Seeing what else is in the directory | dir() |
| Quitting | quit(), q() |
| Saving your work | save.image("workshop1-13-14.RData") |
| Getting help | help(quit) |

Change your directory to someplace user-friendly. Save your R-session, and then quit it and 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 |
| Repetition | u <- seq( from = 5, to = 10, by = .23) |
| Concatenation | x <- c(1,2,3,4,5,6) |
| Summarizing | summary(y) |
| Finding length | length(huge) |

Create the following vectors:
1. Your name, repeated 4 times.
2. 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") |
| Numeric vectors | x <- c(1,2,3,4,5,6) |
| Factor vectors | q <- as.factor(q) |

# head(10)
Changing vector class:  
\[
y \leftarrow \text{as.character}(y) \\
w \leftarrow \text{as.character}(w) \\
y \leftarrow \text{as.numeric}(y) \\
q \leftarrow \text{as.numeric}(\text{as.character}(q)) 
\]

#as.character(): 1 arg  
#as.numeric(): 1 arg  
#Careful with as.numeric() on factors!

What does summary() do on the following vector classes?
1. character (for example, \(w\))
2. numeric (for example, \(q\))
3. 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 | \(y + 3\) |
| Matching vectors element-by-element | \(\text{nchar}(z)\) |
| | \(\sqrt{x}\) \(\text{sqrt}()\): 1 numeric arg |
| Recycling smaller vectors when lengths are mismatched | \(\text{nchar}(w) + \text{nchar}(z)\) |
| | \(y + y\) |
| | \(y * 2\) |

### 3.4 Not vectorization

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

1. Turn \(y\) into a character vector and sort it. How are digits sorted when they are characters?
2. Turn \(y\) into a numeric vector and sort it from highest to lowest.
3. Sort \(\text{huge}\) 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 | \(\text{intersect}(x, y)\) \(\text{intersect}()\): 2 args |
| Getting all the elements in either vector, once | \(\text{union}(x, y)\) \# \(\text{union}()\): 2 args |

### 3.6 SUBSETTING VECTORS

| Getting each element once | \(\text{unique}(z)\) \#\(\text{unique}()\): 1 arg |

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

\[
\text{OBJECT[ ]}
\]

<table>
<thead>
<tr>
<th>By position (aka index)</th>
<th>huge[ 1 ] #The first element</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>huge[ length(huge) ] #The last element</td>
</tr>
<tr>
<td>Indexes can be vectors</td>
<td>huge[ 1:5 ] #The first five elements</td>
</tr>
<tr>
<td></td>
<td>huge[ c(1,5) ] #The first and fifth elements</td>
</tr>
<tr>
<td></td>
<td>huge[ seq(from = 1, to = length(huge), by = 3) ] #Every third element</td>
</tr>
</tbody>
</table>

Find the following elements of huge:
1. The 15th element
2. The 12th, first, and last element, in that order.

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","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" ) #Use ?read.table for full set of arguments |
|                         | crime <- read.table( "crime.csv", sep = ",", header = TRUE ) |

Create your own dataframe, with the following columns:
- The names of your immediate family members
- Their ages
- Their relation to you

Example:

<table>
<thead>
<tr>
<th>name</th>
<th>age</th>
<th>relation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sophie</td>
<td>61</td>
<td>mother</td>
</tr>
<tr>
<td>Doug</td>
<td>61</td>
<td>father</td>
</tr>
<tr>
<td>Clara</td>
<td>29</td>
<td>me</td>
</tr>
<tr>
<td>Pheebe</td>
<td>32</td>
<td>sister</td>
</tr>
<tr>
<td>Roy</td>
<td>2</td>
<td>nephew</td>
</tr>
<tr>
<td>Daniel</td>
<td>31</td>
<td>husband</td>
</tr>
</tbody>
</table>

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()`  # `colnames()`: 1 arg

### Changing column names

- `colnames(df) <- c("AwesomeLetters", "integers")`
- `colnames(df)[1] <- "letters"`

---

Figure out the following information:

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

### 4.3 Subsetting and sorting dataframes

#### By position

- `d[3,5]`  # TWO dimensions: [row] [column]
- `d[,1]`  
- `d[5,]`  
  
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,"letters"]`  # Third row, first column
- `d[3,1]`  # Third row, letters column
- `d$letters[3]`  # Third element in letters column

Sorting uses subsetting

- `order(d$letters)`  # Gives indices of letters, sorted
- `d <- d[order(d$letters),]`  # Sort d by letters column

---

Using `ratings` and `crime`, figure out the following information:

1. What is the `meanFamiliarity` value in the first row of `ratings`? Find it out in at least two ways.
2. Pull out the first, eighth, and seventy-fifth word (i.e., the thing in the `Word` column), Do it in at least two ways.
3. Pull out the values in the `Frequency`, `FamilySize`, and `Class` columns for the first row in `ratings`
4. Pull out the `murder` and `assault` rates for the first three rows in `crime`
5. What is the twelfth-highest `murder` rate? Make sure not to double-count duplicates!
5 Subsetting by condition

5.1 Conditions

<table>
<thead>
<tr>
<th>Testing equality</th>
<th>(5 == 5)</th>
<th># NOTE THE DOUBLE ==!!</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>&quot;cat&quot; == &quot;cat&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;cat&quot; == &quot;dog&quot;</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Testing inequality</th>
<th>(10 &lt; 10)</th>
<th># “less than”</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(10 \leq 11)</td>
<td># “less than or equal to”</td>
</tr>
<tr>
<td></td>
<td>(10 \geq 12)</td>
<td># “greater than or equal to”</td>
</tr>
<tr>
<td></td>
<td>(10 != 10)</td>
<td># “not equal to”</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Testing containment</th>
<th>(10 %in% c(10, 11, 12))</th>
<th># %in%: in the following vector</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>&quot;cat&quot; %in% c(&quot;dog&quot;, 10, &quot;rat&quot;,&quot;McCoy&quot;)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Vectorization and conditions</th>
<th>(y &gt; 5)</th>
<th>#Test each element in (y) for this condition</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>huge == &quot;Spock&quot;</td>
<td></td>
</tr>
</tbody>
</table>

| 5.2 Subsetting vectors by condition |

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:

1. \(\text{logic} \leftarrow \text{c(TRUE, FALSE,TRUE, FALSE,TRUE, FALSE,TRUE, FALSE,TRUE)}\)  
   \#Note the capitals, which signal logical values

2. \(y[\text{logic}]\)  
   \#Get every other value in \(y\), because every other value in \(\text{logic}\) 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:

1. \(y[y > 5]\)  
   \#Returns only the values of \(y\) greater than 5

2. \(\text{huge}[\text{huge} == "triangle"]\)  
   \#Returns only the values of \(\text{huge}\) that are \(\text{triangle}\)

3. \(\text{huge}[\text{huge} \%in\% c("Spock", "rhinoceros")]\)  
   \#Returns only the values of \(\text{huge}\) that are "Spock" or "rhinoceros"

Practice:
1. R has a vector built in, called \texttt{letters}. Pull out only the vowels. (Hint: you can think of vowels as a vector containing "a", "e", "i", "o", and "u".)
2. Pull out the elements of \(q\) that are greater than 8

Combining conditions

\(\text{"cat" \%in\% c( "cat" , "dog") \& 5 > 2}\)  
\# \&: “and”

\(\text{"cat" \%in\% c( "cat" , "dog") \& 5 < 2}\)

\(\text{"cat" \%in\% c( "cat" , "dog") | 5 < 2}\)  
\# | : “or”

10 = 11 | 5 < 2

Practice:
1. Pull out the elements of \(q\) that are less than 12 and also have two characters
2. 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

### 5.3 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:"
  \[
  d[ d$integers > 57 , "letters" ]
  \]

- “Give me the integers for which the value in the letters column is "a":"
  \[
  d[ d$letters == "a", "integers" ]
  \]

- “Give me the letters for which the integer is less than 54 OR greater than 60:"
  \[
  d[ d$integer < 54 | d$integer > 60 , "letters" ]
  \]

Using \( crime \), figure out the following information:

1. The murder rate for California
2. Which states have a murder rate higher than 11.25
3. Which states have an assault rate less than 170, but a murder rate greater than 7.7
4. Which states have an urban population percent rate that is exactly the median urban percent rate
5. 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:

1. Which words are plants (Class column)
2. Which words are complex (Complex column)
3. Which words are both animals (Class column) AND complex
4. Create a dataframe called "animals," which contains only the animal rows of \( ratings \)

### 6 Advanced dataframe manipulations

#### 6.1 Adding columns

<table>
<thead>
<tr>
<th>Method</th>
<th>Instructions</th>
</tr>
</thead>
</table>
| By flat              | 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 \( ratings \):

1. The ratio of a word’s meanSizeRating to its meanWeightRating
2. The difference between a word’s length and the mean length of all the words
3. The standard deviation of the word-lengths in this dataframe (this will be the same value for all rows).
4. 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)

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

\[
\text{aggregate( \langle Outcome column\rangle, list( \langle Factor 1\rangle, \langle Factor 2\rangle, ...), \langle function\rangle) }
\]

- “Dear R: Please find the mean frequency for all words that are animals, and all words that are plants”:
  \[
  \text{aggregate( \langle ratings$Frequency\rangle, list(\langle ratings$Class\rangle), \langle mean\rangle) }
  \]
- “Find the median length for all words that are complex, and all words that are simplex”:
  \[
  \text{aggregate( \langle ratings$Length\rangle, list(\langle ratings$Complex\rangle), \langle median\rangle) }
  \]
- “Find the standard deviation of frequency for all combinations of word class and word complexity”:
  \[
  \text{aggregate( \langle ratings$Frequency\rangle, list(\langle ratings$Class\rangle, \langle ratings$Complex\rangle), \langle sd\rangle) }
  \]

Counting up the number of observations:

\[
\text{xtabs( } \sim \langle Factor 1\rangle + \langle Factor 2\rangle \text{ ...)}
\]

- “Dear R: How many words are plants, and how many are animals?”
  \[
  \text{xtabs( } \sim \langle Class\rangle, \text{ data = \langle ratings\rangle) }
  \]
- “What is the breakdown of observations for all combinations of class and complexity?”
  \[
  \text{xtabs( } \sim \langle Class\rangle + \langle Complex\rangle, \text{ data = \langle ratings\rangle) }
  \]
- “How many states have more than the mean value of murders?”
  \[
  \text{xtabs( } \sim \langle assault\rangle > \langle mean( assault)\rangle, \text{ data = \langle crime\rangle) }
  \]
  \# Returns TRUE/FALSE counts

Practice:
1. What is the mean Length of animal words and of plant words?
2. How many states have less than half of their population living in cities?

6.3 Merging dataframes
How do you unite this information into one object?

states1

<table>
<thead>
<tr>
<th>state.name</th>
<th>state.abb</th>
<th>state.division</th>
<th>state.region</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alabama</td>
<td>AL</td>
<td>East South Central</td>
<td>South</td>
</tr>
<tr>
<td>Alaska</td>
<td>AK</td>
<td>Pacific</td>
<td>West</td>
</tr>
<tr>
<td>Arizona</td>
<td>AZ</td>
<td>Mountain</td>
<td>West</td>
</tr>
<tr>
<td>Arkansas</td>
<td>AR</td>
<td>West South Central</td>
<td>South</td>
</tr>
<tr>
<td>California</td>
<td>CA</td>
<td>Pacific</td>
<td>West</td>
</tr>
<tr>
<td>Colorado</td>
<td>CO</td>
<td>Mountain</td>
<td>West</td>
</tr>
</tbody>
</table>
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[,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") # merge(): magic.
  states <- merge(crime, states, by.x="state", by.y="state.name")

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() and merge().)

very important! before you go!

write.csv(states, "states.csv", row.names=FALSE)