REVOLUTION ANALYTICS

Fundamentals in R: Data Management

August 12, 2014

Module 3 - Manipulating Data

In this module we'll take off in our data analysis by loading data in, manipulating it and saving our changes.

The objectives are

- get data into the R environment
- execute basic data manipulations
- export data

Getting Data in

Discussion

- What sorts of data files do you work with?
- What do you do with the data? What technologies do you use?
- What is difficult that you would like simplified?
- What would you like to do that you can't do now?

Getting Data in

Let's start with something most of us are familiar with: csv file.

artist.id	artist.mbid	artist.playmeid	artist.7digitalid	arti
AR009211187B989185	9dfe78a6-6d91-454e-9b95-9d7722cbc476	147337	7724	Car
AR009211187B989185	9dfe78a6-6d91-454e-9b95-9d7722cbc476	147337	7724	Car
AR00A6H1187FB5402A	312c14d9-7897-4608-944a-c5b1c76ae682	-1	432034	The
AR00A6H1187FB5402A	312c14d9-7897-4608-944a-c5b1c76ae682	-1	432034	The
AR00A6H1187FB5402A	312c14d9-7897-4608-944a-c5b1c76ae682	-1	432034	The
AR00A6H1187FB5402A	312c14d9-7897-4608-944a-c5b1c76ae682	-1	432034	The
AR00LNI1187FB444A5	7e836d29-fc2d-4a1f-b8da-566d47c49eed	75833	87908	Bru
AR00LNI1187FB444A5	7e836d29-fc2d-4a1f-b8da-566d47c49eed	75833	87908	Bru
AR00MBZ1187B9B5DB1	ff748426-8873-4725-bdc7-c2b18b510d41	17661	52722	Mem
AR00MBZ1187B9B5DB1	ff748426-8873-4725-bdc7-c2b18b510d41	17661	52722	Mem
AR00MBZ1187B9B5DB1	ff748426-8873-4725-bdc7-c2b18b510d41	17661	52722	Mem

Getting Data in

To load a csv in we use a function *read.csv()*. Notice we're creating a new variable "fileName" that contains the string specifying the location and file name of our csv.

```
fileName <- "data/MSS10K.csv"
songData <- read.csv(file = fileName)
```

We use <- to assign something to a variable and store this in memory. When we do not use the assignment operator <- we are simply executing code, the output of which is written to the console & not stored in memory.

Exploring Data

Our data is in memory and accessible to R. Let's see what we have to play with.

To open a tabular view of the data:

```
View(songData)
```

Exploring Data

To return the variable/column names of our data:

```
names(songData)
```

```
##   [1] "artist.id"                 "artist.mbid"
##   [3] "artist.playmeid"           "artist.7digitalid"
##   [5] "artist.name"               "artist.latitude"
##   [7] "artist.longitude"          "artist.location"
##   [9] "artist.hotttnesss"         "artist.familiarity"
##  [11] "song.id"                   "track.id"
...
```

Our data resource provides information on each of these variables http://labrosa.ee.columbia.edu/millionsong/pages/field-list

Exploring Data

To return basic information about the variables/columns of our data:

```
summary(songData)
```

```
##             artist.id                                artist.mbid
##  ARQIHQI122988FEB8E: 13   01d336ae-7f79-496d-94f9-211b57517b17: 12
##  AR12F2S1187FB56EEF: 12   3d2b98e5-556f-4451-a3ff-c50ea18d57cb: 12
##  AR9W3X91187FB3994C: 12   401c3991-b76b-499d-8082-9f2df958ef78: 12
##  AREWQSE1187B9AEC6C: 12   4f91f760-85f6-4603-9b60-15789000e256: 12
##  ARIRD6J1187FB5A98C: 12   5ecc3f72-20a6-47a0-8dc5-fb0b3dadeea0: 12
...
```

How are missing values indicated?

Exploring Data

Fill Values

Fill Value	Meaning
NA	missing
NaN	Not A Number (dividing 0 by 0)
Inf, -Inf	Positive & Negative Infinity (dividing const by 0)

Resource for Missing (NA) Values:
http://faculty.nps.edu/sebuttre/home/R/missings.html

Extras

File types you may encounter & How to Load them into R

File Type	Package / Function
txt, csv , etc.	read.table(), read.csv()
SAS, SPSS, Stata	{foreign}
relational databases	{RODBC}
Excel spreadsheets	read.csv(), {XLConnect}
Web API's	{httr}

Resource: http://cran.r-project.org/doc/manuals/r-release/R-data.html

Getting Data in

What did we create in memory when we read in the data?

```
class(songData)
```

```
## [1] "data.frame"
```

What is a data.frame?

Getting Data in

What is a data.frame?

* A type of R object,
* All R functions work on objects,
* All R objects have a class,
* Classes define how objects look
* Function use depends on object classs

Usually, when you bring data in from an external source, it will be brought in as a data frame.

Exploring Data

How do we know for sure what will be returned?

See the description of the function we used to read in the data.

```
?read.csv
```

Exploring Data

Data frames are essentially multiple vectors combined as columns of a 2 dimensional object. Each vector/column can contain only a single kind of value (numeric, character, logical) but a data frame can contain columns of different types.

Other types of R object, some of which we'll touch on later in this course, include:

* atomic vectors
* lists
* matrices and arrays
* factors

Extras

Data types in R

1. Vectors: simplest object, 1-dim, each vector can contain only one type of data (real numbers, strings, logicals, etc). *c(), vector(), as.vector(), is.vector()*
2. Data frame: like a matrix but does not assume that all columns/elements have the same type; like a list but all elements/columns must be of the same length. *data.frame(), as.data.frame(), is.data.frame()*

Extras

3. Factors: like vectors but each element is categorical, efficient storage for many observations staking on a small(er) set of possible values (called levels of the factor variable) *factor(), as.factor(), is.factor()*
4. Matrix: like a vector but 2-dim, all columns must be of the same type & length matrix creates a matrix from the given set of values. *matrix(), as.matrix(), is.matrix()*
5. Arrays: like a matrix but can be of higher dim. *array(), as.array(), is.array()*
6. Lists: unlike the above, elements do not need to be of the same type or length *list(), as.list(), is.list()*

Data Editor for Data Frames

You can manually modify your data frame by running edit(). Below, changes we make to songData will not affect songData but will be saved to songData2.

```
songData2 <- edit(songData)
```

Extracting Columns/ Vectors

We can *extract* a single vector/column by indexing into the data frame

```
artistHotttnesss <- songData$artist.hotttnesss
```

```
class(artistHotttnesss)
```

```
## [1] "numeric"
```

```
head(artistHotttnesss)
```

```
## [1] 0.2974 0.2974 0.3956 0.3956 0.3956 0.3956
```

Extracting Columns/ Vectors

artistHotttnesss is a numeric vector

Vectors:

- the simplest object
- 1-dim
- Each can contain only one type of data
  - numbers
  - strings,
  - logicals

Functions: c(), vector(), as.vector(), is.vector()

Extracting Columns/ Vectors

Instead of using the $ operator & element name, we can use [[]]'s and the element number (OR element name).

```
artistFamiliarity <- songData[[10]]
# artistFamiliarity <- songData[['artist.hotttnesss']]

class(artistFamiliarity)

## [1] "numeric"

head(artistFamiliarity)

## [1] 0.3968 0.3968 0.5141 0.5141 0.5141 0.5141
```

Vectorized Arithmetic

Modifying objects element by element in R is easy, thanks to vectorized arithmetic operations.

```
(h <- head(artistFamiliarity))
```

```
## [1] 0.3968 0.3968 0.5141 0.5141 0.5141 0.5141
```

```
2 * h
```

```
## [1] 0.7936 0.7936 1.0281 1.0281 1.0281 1.0281
```

```
(h - mean(h))/sd(h)
```

```
## [1] -1.2910 -1.2910  0.6455  0.6455  0.6455  0.6455
```

Vectorized Arithmetic

```
head(artistHotttnesss/artistFamiliarity)
```

```
## [1] 0.7495 0.7495 0.7696 0.7696 0.7696 0.7696
```

```
head(songData$artist.hotttnesss/songData$artist.familiarity)
```

```
## [1] 0.7495 0.7495 0.7696 0.7696 0.7696 0.7696
```

Adding Variables/ Columns

To modify a data set by adding a new variable whose values are determine by the values of two other variables in our data set, executing code form the command line can be easier.

We create a new vector which is an element-by-element ratio of two existing vectors

```
hfRatio <- artistHotttnesss/artistFamiliarity
```

Adding Variables/ Columns

We append this vector to our existing data, creating a new variable called hfRatio. Note, number of rows in data frame (songData) must be equal to the number of elements in vector (hfRatio)

```
songData$hfRatio <- hfRatio
```

Or, in one line

```
songData$hfRatio <- songData$artist.hotttnesss/songData$artist.familiari
```

Extras

Operators in R

Operator	Function
^	exponentiation (right to left)
- +	subtract, add
* /	multiply, divide
< > <= >= == !=	ordering and comparison
!	negation
& &&	and
| ||	or
-> ->> <- <<- =	assignment

Extras

Operators in R help file

```
help(Syntax)
```

Extras

Built in functions that operate on vectors

```
prod(x)
sum(x)
length(x)
mean(x)
var(x)
max(x)
min(x)
range(x)
sd(x)
sort(x)
order(x)
```

Extras

The recycling rule

What happens if we add two vectors of different lengths?

```
(v1 <- c(1:10))
```

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

```
(v2 <- c(1:2))
```

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

```
v1 + v2
```

```
##  [1]  2  4  4  6  6  8  8 10 10 12
```

Extras

The following example illustractes vectorized arithmetic & the recycling rule

```
v1 <- c(1:10)
v2 <- 2

v1 * v2
```

```
##  [1]  2  4  6  8 10 12 14 16 18 20
```

Objects in your Workspace

Every time we've used the assignment operator "<-", we've created/modified objects in our workspace (global environment).

To return a character vector containing the names of all objects in your workspace:

```
ls()
```

To remove an object:

```
rm(h)
```

… or

```
rm("h")
```

Objects in your Workspace

To remove multiple objects:

```
rm(list=c("v1", "v2"))
```

To remove all objects: Note: You will get no warning, so don't do this unless you are really sure.

```
# rm(list=ls())
```

Exporting Data

To save an object you have at least two options:

- save it in a file format that you can later load using R and preserve the object type & structure OR
- export as, say, a csv

Exporting Data

Saving .RData files:

Save specific objects

```
save(songData, file = "data/Module3_songData.RData")
```

We can remove the objects just saved form memory

```
rm(list = c("songData", "songSubset4"))
```

Exporting Data

Load the saved and subsequently removed objects

```
load("songObjects.RData")
```

Delete the saved file

```
file.remove("songObjects.RData")
```

Exporting Data

Save all objects in memory

```
save(list = ls(), file = ".RData")
```

… or equivalently but more concisely

```
save.image()
```

Exporting Data

If you want to save your data as a csv file, we can use a function like write.csv()

```
write.csv(songData, "data/songData.csv")
```

Working Directory

Where are all of these files written? Since we did not specify a file path and only specified a file name, these files were written to our working directory.

Identify your working directory

```
getwd()
```

```
## [1] "C:/Users/Jamie/Revolution/AcademyR/Fundamentals/doc"
```

Working Directory

Change your working directory

```
setwd("C:/New/Directory/Path/Yourdata")
```

Write to/ read from outside your working directory

```
write.csv(songData, "C:/Apath/Outside/YourWD/songData.csv")
```

Exercise

- Load revGeocodeDF.RData into memory. What type of object is this?
- Explore this data set.
- Export the data to a csv in your working directory.
- Locate the csv and open it using Notepad or Excel.

Hint: load(), class(), names(), View(), write.csv()

Solution

```
load("data/revGeocodeDF.RData")

class(d)

names(d)

View(d)

write.csv(d, "data/revGeocodeDF.csv")
```

REVOLUTION ANALYTICS

Thank you

Revolution Analytics is the leading commercial provider of software and support for the popular open source R statistics language.

www.revolutionanalytics.com, 1.855.GET.REVO, Twitter: @RevolutionR