R Club

September 16th 2015

An Introduction to R

Basics

R is a language and a suite of software for programming and data analysis.

- Based on the S language written in the 70's
- ***R*** is used interchangeably to refer to the program and the language
- Typically users use RStudio as an easier interface

R Studio

- Not created by the original writers of R.
- Allows the user to see and explore the variables, including sorting.
- Can manage many codes at once
- Easily search help files

Data Classes

R has 5 main data types, but there are numerous types available These tell R whether 1 means the number 1 (numeric) which we can do math with or the letter representation '1' which has no numeric value

Numeric These are just standard numbers, whether integers or decimals, either one is numeric

```
1
```

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

```
1.1
```

```
## [1] 1.1
```

Integers Integers are also numeric, but integers are whole numbers with no decimal. As we can see, the integer of both 1.1 and 1 is 1

```
as.integer(1.1)
```

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

```
as.integer(1)
```

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

Character These are just letters, they have no numeric value associated with them. R reads these as one long 'string'.

```
str( 'Luke Skywalker')
```

```
##  chr "Luke Skywalker"
```

Infact, R sees this phrase not as 2 words 'luke' and 'skywalker' but the string of characters 'luke','(space)','skywalker'. Which is 14 spaces long

```
nchar('Luke Skywalker')
```

```
## [1] 14
```

Factor Factors are character strings that have some sort of order to them. Like monday, tuesday, and wednesday. In this case the characters are representing objects with some inherit value (an order really 1,2,3 etc...). So say we have the star wars movies

```
star_wars<-factor(c('A New Hope', 'Empire Strikes Back', 'Return of the Jedi', 'The Phantom Menace', 'A

levels(star_wars)
```

```
## [1] "A New Hope"           "Attack of the Clones" "Empire Strikes Back"
## [4] "Return of the Jedi"   "Revenge of the Sith"  "The Phantom Menace"
```

The names do represent something to us, but they don't necessarily have any value. We can, however, give them a value, say we want their order to represent how we personally rank them. We can do this by either giving the object an order vai the levels() function.

```
levels(star_wars)<-c('Empire Strikes Back', 'A New Hope', 'Revenge of Sith', 'Return of Jedi', 'The Pha

star_wars
```

```
## [1] Empire Strikes Back Revenge of Sith     Return of Jedi
## [4] Attack of Clones    A New Hope          The Phantom Menace
## 6 Levels: Empire Strikes Back A New Hope ... Attack of Clones
```

Or by originally telling it what the levels are

```
star_wars<-factor(c('A New Hope', 'Empire Strikes Back', 'Return of the Jedi', 'The Phantom Menace', 'A

levels(star_wars)
```

```
## [1] "Empire Strikes Back" "A New Hope"          "Revenge of Sith"
## [4] "Return of Jedi"      "The Phantom Menace"  "Attack of Clones"
```

Logical Logical is just TRUE or FALSE, there's only 2 possible states, yes or no, on and off, which is represented by True or False. R understands this in 3 different ways. Captial letters TRUE and FALSE

```r
is.logical(TRUE)
```

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

Just the letters T and F

```r
is.logical(F)
```

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

Or by the numbers 1 or 0. Which R can convert to logical.

```r
as.logical(c(1,0,1))
```

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

Data Types

R has 4 main data types These are structural types your data can be stored in. They are defined by if they take more than one class type and the dimensions of the data

Vectors This is a 1 dimensional data set that (only dimension is length) and it can only consist of the same data class. c() creates a vector

```r
c(1,2,4)
```

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

```r
c(1,2,3,4,5,6)
```

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

R attempts to coherse the data into the class it has in common. So if we have two entries 1 and mom, it will coherse them both to characters because 1 can be converted to a character, but mom can not be converted to a number

```r
c(1,  'mom')
```

```
## [1] "1"   "mom"
```

```r
str( c(1,  'mom') )
```

```
##  chr [1:2] "1" "mom"
```

Matrices This is a 2 dimensional container/array that are all of the same data type

- essentially a 2 dimensional vector (dimensions are rows and columns)

```
matrix_1<-matrix(1:100, nrow=2, ncol=50)
matrix_1
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
## [1,]    1    3    5    7    9   11   13   15   17    19    21    23    25
## [2,]    2    4    6    8   10   12   14   16   18    20    22    24    26
##      [,14] [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24]
## [1,]    27    29    31    33    35    37    39    41    43    45    47
## [2,]    28    30    32    34    36    38    40    42    44    46    48
##      [,25] [,26] [,27] [,28] [,29] [,30] [,31] [,32] [,33] [,34] [,35]
## [1,]    49    51    53    55    57    59    61    63    65    67    69
## [2,]    50    52    54    56    58    60    62    64    66    68    70
##      [,36] [,37] [,38] [,39] [,40] [,41] [,42] [,43] [,44] [,45] [,46]
## [1,]    71    73    75    77    79    81    83    85    87    89    91
## [2,]    72    74    76    78    80    82    84    86    88    90    92
##      [,47] [,48] [,49] [,50]
## [1,]    93    95    97    99
## [2,]    94    96    98   100
```

We can refer to each of these by their position using brackets [,]. The left position is the row and the right is the column: data[row, column].
They can only be of the same data class

```
matrix_1[2, ]
```

```
##  [1]   2   4   6   8  10  12  14  16  18  20  22  24  26  28  30  32  34
## [18]  36  38  40  42  44  46  48  50  52  54  56  58  60  62  64  66  68
## [35]  70  72  74  76  78  80  82  84  86  88  90  92  94  96  98 100
```

Data.frames Data.frames are 2 dimensional array, where each column is the same data class. So we can have multiple data classes in one container. The right side of the equals sign will automatically give the name to the column

```
dd<-data.frame(id = 1:5, value = c('a','b','c','d','e'))
dd
```

```
##   id value
## 1  1     a
## 2  2     b
## 3  3     c
## 4  4     d
## 5  5     e
```

We can also manually define the column names

```
colnames(dd)<-c('monkeys','gorillas')
dd
```

```
##   monkeys gorillas
## 1       1        a
```

```
## 2       2        b
## 3       3        c
## 4       4        d
## 5       5        e
```

You can refer to column names either by their position or by the column name:

```
dd$monkeys
```

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

```
dd[,1]
```

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

Lists Lists are containers for the other data types. Can contain any data.type and can be any dimension. You can name lists in the same way as column names in data.frames. They are the storage containers for your data. Think of them like a tackle box / or jewlery box that contain your data with varying sizes and types of items in each bin. At a basic level they're useful for storing data of different types, at a more advanced level you can do all of your data analysis in them using the lapply function.

```
list_1<-list(id= 1:4, names=c('1','2'), data_1 = data.frame(id=1:10, rep(c('a','b'),5)))
list_1
```

```
## $id
## [1] 1 2 3 4
##
## $names
## [1] "1" "2"
##
## $data_1
##    id rep.c..a....b....5.
## 1   1                   a
## 2   2                   b
## 3   3                   a
## 4   4                   b
## 5   5                   a
## 6   6                   b
## 7   7                   a
## 8   8                   b
## 9   9                   a
## 10 10                   b
```

```
list_1$data_1$id   ###you can refer to items and columns in the same way you would a dataframe, but hie
```

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

```
names(list_1)
```

```
## [1] "id"     "names"  "data_1"
```

```
str(list_1)
```

```
## List of 3
##  $ id    : int [1:4] 1 2 3 4
##  $ names : chr [1:2] "1" "2"
##  $ data_1:'data.frame':  10 obs. of  2 variables:
##   ..$ id                 : int [1:10] 1 2 3 4 5 6 7 8 9 10
##   ..$ rep.c..a....b....5.: Factor w/ 2 levels "a","b": 1 2 1 2 1 2 1 2 1 2
```

Arrays Arrays are multi-dimensional matrices, all data has to be the same data.class though.

- A 2 dimensional array is a matrix In this following example we are going to make a 2x2x5 dimensiaonl array. Think of this 3d example as an apartment complex with 4 rooms on each level and 5 levels total. We can make arrays as many dimensions as possible.

```
data<-array(1:100, dim=c(2,2,5))
data
```

```
## , , 1
##
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
##
## , , 2
##
##      [,1] [,2]
## [1,]    5    7
## [2,]    6    8
##
## , , 3
##
##      [,1] [,2]
## [1,]    9   11
## [2,]   10   12
##
## , , 4
##
##      [,1] [,2]
## [1,]   13   15
## [2,]   14   16
##
## , , 5
##
##      [,1] [,2]
## [1,]   17   19
## [2,]   18   20
```

Function interface

One of the goals of the writers of the S language was to make it easy to go from idea, to implementation of analysis or creation of software

- basically all operations in R are written as functions

```
head(mtcars)
```

```
##                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

```
###Subset is really just a function thats essentially subsetting like you would
subset(mtcars, mpg>30)
```

```
##                mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Fiat 128       32.4   4 78.7  66 4.08 2.200 19.47  1  1    4    1
## Honda Civic    30.4   4 75.7  52 4.93 1.615 18.52  1  1    4    2
## Toyota Corolla 33.9   4 71.1  65 4.22 1.835 19.90  1  1    4    1
## Lotus Europa   30.4   4 95.1 113 3.77 1.513 16.90  1  1    5    2
```

```
mtcars[mtcars$mpg>30,]
```

```
##                mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Fiat 128       32.4   4 78.7  66 4.08 2.200 19.47  1  1    4    1
## Honda Civic    30.4   4 75.7  52 4.93 1.615 18.52  1  1    4    2
## Toyota Corolla 33.9   4 71.1  65 4.22 1.835 19.90  1  1    4    1
## Lotus Europa   30.4   4 95.1 113 3.77 1.513 16.90  1  1    5    2
```

```
#These are equivalent


###in this example we can see calling a function shows us that R is hiding all the code behind the scen
library(RAtmosphere)
suncalc
```

```
## function (d, Lat = 0, Long = 0, UTC = TRUE)
## {
##     rad <- function(x) pi * x/180
##     R = 6378
##     epsilon = rad(23.45)
##     L = rad(Lat)
##     if (UTC) {
##         timezone = 0
##     }
##     else {
```

```
##         timezone = -4 * (abs(Long)%%15) * sign(Long)
##     }
##     r = 149598000
##     theta = 2 * pi/365.25 * (d - 80)
##     z.s = r * sin(theta) * sin(epsilon)
##     r.p = sqrt(r^2 - z.s^2)
##     t0 = 1440/(2 * pi) * acos((R - z.s * sin(L))/(r.p * cos(L)))
##     that = t0 + 5
##     n = 720 - 10 * sin(4 * pi * (d - 80)/365.25) + 8 * sin(2 *
##         pi * d/365.25)
##     if (UTC) {
##         sunrise = (n - that)/60
##         sunset = (n + that)/60
##     }
##     else {
##         sunrise = (n - that + timezone)/60
##         sunset = (n + that + timezone)/60
##     }
##     return(list(sunrise = sunrise, sunset = sunset))
## }
## <environment: namespace:RAtmosphere>
```

But we can actually do this as well!

Say we want to write a function to calculate Standard Error Tip: Remember the object names inside 'function()' is the **variable** r stores your input as. So whatever vector we give to our new function, R will store that vector as 'data'. We then proceed under the assumption that our vector is now called 'data'

```
#SD/sqrt(n)

SEM<-function(data){

  sd(data)/length(data)


}

SEM(data=1:10000)
```

```
## [1] 0.2886896
```

Now no matter what numerical vector we put into SEM(), it will give us the standard error. It does this by finding the standard error, and then infering the 'n' by calculating the length of the vector (which are essentially the same thing!)

Good places to go from here

Baby steps For some who is just starting out and who is new to statistical software in general SWIRL!

Serious steps If you want to tackle learning R in a more ambitious or complete manner, I suggest taking the coursera course, which is free and runs essentially every month. Coursera

And then move onto: Advanced R

This advanced course, is really just an html version of Hadley Wickhams advanced R book. It's written using Markdown, and by the end of it should give you a fluent grasp on R

News and Notes

Git hub site/repository! OUR GITHUB IS LIVE (Thanks to Mike)
GO HERE FOR ALL OF OUR LINKS, INFO, and REFERNECES!
https://github.com/MTHallworth/ENWC_R

Interesting Packages Rwars gives you access to all star wars knowledge you ever needed

```
###These are required if loading for the first time
#library(devtools)
#install_github("ironholds/rwars", ref = "0.5.0")
library(rwars)



#get_film(2)

get_person(1)
```

```
## $name
## [1] "Luke Skywalker"
##
## $height
## [1] "172"
##
## $mass
## [1] "77"
##
## $hair_color
## [1] "blond"
##
## $skin_color
## [1] "fair"
##
## $eye_color
## [1] "blue"
##
## $birth_year
## [1] "19BBY"
##
## $gender
## [1] "male"
##
```

```
## $homeworld
## [1] "http://swapi.co/api/planets/1/"
##
## $films
## $films[[1]]
## [1] "http://swapi.co/api/films/7/"
##
## $films[[2]]
## [1] "http://swapi.co/api/films/6/"
##
## $films[[3]]
## [1] "http://swapi.co/api/films/3/"
##
## $films[[4]]
## [1] "http://swapi.co/api/films/2/"
##
## $films[[5]]
## [1] "http://swapi.co/api/films/1/"
##
##
## $species
## $species[[1]]
## [1] "http://swapi.co/api/species/1/"
##
##
## $vehicles
## $vehicles[[1]]
## [1] "http://swapi.co/api/vehicles/14/"
##
## $vehicles[[2]]
## [1] "http://swapi.co/api/vehicles/30/"
##
##
## $starships
## $starships[[1]]
## [1] "http://swapi.co/api/starships/12/"
##
## $starships[[2]]
## [1] "http://swapi.co/api/starships/22/"
##
##
## $created
## [1] "2014-12-09T13:50:51.644000Z"
##
## $edited
## [1] "2014-12-20T21:17:56.891000Z"
##
## $url
## [1] "http://swapi.co/api/people/1/"
```

```
get_planet(4)
```

```
## $name
## [1] "Hoth"
```

```
##
## $rotation_period
## [1] "23"
##
## $orbital_period
## [1] "549"
##
## $diameter
## [1] "7200"
##
## $climate
## [1] "frozen"
##
## $gravity
## [1] "1.1 standard"
##
## $terrain
## [1] "tundra, ice caves, mountain ranges"
##
## $surface_water
## [1] "100"
##
## $population
## [1] "unknown"
##
## $residents
## list()
##
## $films
## $films[[1]]
## [1] "http://swapi.co/api/films/2/"
##
##
## $created
## [1] "2014-12-10T11:39:13.934000Z"
##
## $edited
## [1] "2014-12-20T20:58:18.423000Z"
##
## $url
## [1] "http://swapi.co/api/planets/4/"
```

```
movies<-lapply(1:6, function(x) get_film(x)$title)
data.frame(id=1:6, title=unlist(movies))
```

```
##   id                   title
## 1  1               A New Hope
## 2  2 The Empire Strikes Back
## 3  3      Return of the Jedi
## 4  4      The Phantom Menace
## 5  5    Attack of the Clones
## 6  6     Revenge of the Sith
```

And finally

Possibly the worst package ever:

https://github.com/Ironholds/jammr