An introduction to R

Scott Powers and Ryan Wang

January 3, 2014

The help function

- R is a very well-documented language
- A crucial function is help, as it allows you to access that documentation
- It can be accessed by help(function) or ?function for short e.g. type ?help to learn more about the help function
- Google is also a great resource and usually links to either stackoverflow or the R mailing list

Outline

Data frames

Let's start by reading in some real-life data

```
>   setwd('~/GitHub/data-mining-intersession/day1')
>   nba <- read.csv('box_2012to2013.csv', stringsAsFactors=
>   class(nba)
```

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

In R terminology, data typically comes in the form of a *data.frame* (try ?data.frame). Data frames are similar to spreadsheets or matrices that can store non-numeric values.

Data frames cont.

This is essentially the same as what you would see if you opened the .csv file in MS Excel

```
>   nba[1:5,1:5]
```

```
    game_id  pid         player pos                  team
1 400277721 4270 Trevor Booker  PF Washington Wizards
2 400277721 2426  Trevor Ariza  SF Washington Wizards
3 400277721 2399  Emeka Okafor   C Washington Wizards
4 400277721 4010    A.J. Price  PG Washington Wizards
5 400277721 6580  Bradley Beal  SG Washington Wizards
```

Columns correspond to *variables* e.g. column 5 indicates the number of minutes played. Rows correspond to *observations* e.g. row 1 corresponds to Trevor Booker's line on the box score.

Data frames cont.

This data frame contains 31,375 rows (observations) and 19 columns (variables)

```
> nrow(nba)
[1] 31375
> ncol(nba)
[1] 19
> dim(nba)
[1] 31375    19
```

Data frames cont.

Specific elements of the data can be accessed with [row, column] coordinates

```
>   nba[1:2,1:2]

    game_id  pid
1 400277721 4270
2 400277721 2426

>   nba[1,1]

[1] 400277721

>   nba[2,1]

[1] 400277721
```

We'll discuss subsetting data frames more in-depth after we have seen how vectors (and specifically sequences) work.

Outline

Vectors

The fundamental unit of computation in R is the vector. Create a vector by combining values with the "c" function (see ?vector and ?c):

```
>   c(1,2,3,4,5)
[1] 1 2 3 4 5
>   c('a','b','c','d','e')
[1] "a" "b" "c" "d" "e"
```

Vectors cont.

Assign a value, which includes vectors, to a name using the "<-" operator (see ?assignOps)

```
>   v1 <- c(1,2,3,4,5)
>   v1

[1] 1 2 3 4 5

>   v1[1]

[1] 1
```

The first element of v1 is 1, and the second element is 2. Note that unlike most programming languages, indexing starts at 1.

Vectors cont.

Most functions involving vectors operate element-wise:

```
>   v1 + 3
[1] 4 5 6 7 8
>   v1 + v1
[1]  2  4  6  8 10
>   sqrt(v1)
[1] 1.000000 1.414214 1.732051 2.000000 2.236068
>   (v1 + v1)[2]
[1] 4
```

Vectors cont.

The ":" operator (see ?colon) creates a sequence, which is also a vector

```
>   x <- 1:5
>   x
[1] 1 2 3 4 5
>   x[1]
[1] 1
```

Many functions return vectors. For example, "runif" returns a random vector of length n (see ?runif).

```
> runif(5)
[1] 0.83960653 0.20799022 0.07380461 0.65508980 0.52938711
```

Vectors cont.

Boolean values are TRUE and FALSE, or T and F for short (see ?logical)

```
> TRUE & FALSE
```

```
[1] FALSE
```

```
> !(!T & F)
```

```
[1] TRUE
```

It is easy to create boolean vectors by writing conditions. This will come in handy soon.

```
> 1:5 > 2
```

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

Outline

Exploring the data

Now that we know about sequences, see that the data frame is subsetted by passing vectors as rows and/or columns

```
>   nba[1:5,1:5] # rows 1 to 5, columns 1 to 5

    game_id  pid        player pos                team
1 400277721 4270 Trevor Booker  PF Washington Wizards
2 400277721 2426  Trevor Ariza  SF Washington Wizards
3 400277721 2399  Emeka Okafor   C Washington Wizards
4 400277721 4010    A.J. Price  PG Washington Wizards
5 400277721 6580  Bradley Beal  SG Washington Wizards

>   nba[1:5,] # would give rows 1 to 5, all columns
```

As you can see, you can access all the entries along a particular dimension by leaving the slot blank. What would happen if you typed *nba[,]*?

Exploring the data cont.

The neat thing about data frames is that columns are named. Access specific columns with the "$" operator (see ?Extract)

```
>   names(nba)[1:5]
[1] "game_id" "pid"     "player"  "pos"     "team"
>   nba$pid[1:5]
[1] 4270 2426 2399 4010 6580
>   nba[1:5, 'pid'] # alternatively...
[1] 4270 2426 2399 4010 6580
```

Exploring the data cont.

Using boolean vectors, we can subset the data to look at specific players' statistics

```
>   lbj <- nba[nba$player == "LeBron James",]
>   basic <- c('player','min','pts','reb')
>   lbj[1:5,basic]
```

```
            player min pts reb
40    LeBron James  29  26  10
464   LeBron James  37  23   7
735   LeBron James  39  20   9
1097  LeBron James  30  23  11
1406  LeBron James  30  20  12
```

Under the hood, the boolean vector is being translated into a vector of indices for the TRUE entries. This can be accomplished explicitly by using the "which" function on a boolean vector (see ?which).

Exploring the data cont.

It's not enough to know only the dimensions of the data frame, which tell us the number of box scores. We might need to know things such as: how many players were active?

```
>   v1 <- c('a','a','b','d')
>   unique(v1)

[1] "a" "b" "d"

>   length(unique(v1))

[1] 3

>   length(unique(nba$pid)) # number of players active

[1] 476
```

The "unique" function returns the unique values that appear in the inputted vector. The "length" function returns its length.

Exercises!

1. Create a vector containing the even numbers between 1 and 50
2. Create a subset of the data that contains entries for all point guards (PG)
3. What are the dimensions of the data.frame from 2)?
4. How many total games were played in the '12-'13 season? (note: some questions don't require data)

Outline

Time for some fun!

We now have some basic tools for exploring and manipulating data. We can use them to answer question such as: how many points did LeBron James score in the '12-'13 season?

```
>   sum(lbj$pts) # see ?sum (hopefully you know this...)
```

```
[1] NA
```

He scored 'NA' points??!?

Time for some fun! cont.

One more thing: our data contains missing values (see ?NA and ?is.na)

```
>   lbj[lbj$game_id=='400278818',basic] # did not play
```

```
             player min pts reb
28015 LeBron James  NA  NA  NA
```

Sorry for the pump fake... How to deal with missing data is an important consideration (which has luckily been taken care of for us). I once saw a company using 0, 10, and 100 to indicate invalid values in a dataset that tracked dollar amounts!?

Time for some fun! cont.

To get around the missing values, we need to specify the "na.rm" option in the "sum" function

```
>   sum(lbj$pts, na.rm=T)
```

```
[1] 2055
```

Check this against ESPN's records `http://espn.go.com/nba/player/stats/_/id/1966/lebron-james`. We're off...

Time for some fun! cont.

Our data says 2,055 while ESPN says 2,036. What gives?

```
>   nba[nba$game_id=='400436572',][1:5,basic]
```

	player	min	pts	reb
20088	Kevin Garnett	6	0	3
20089	LeBron James	30	19	3
20090	Carmelo Anthony	31	26	12
20091	Chris Bosh	23	6	2
20092	Dwyane Wade	29	21	3

Time for some fun! cont.

... because it's the All-Star game! Let's try removing it.

```
>   nba <- subset(nba, nba$game_id!='400436572')
>   lbj <- nba[nba$player == "LeBron James",]
>   sum(lbj$pts, na.rm=T) # much better
```

```
[1] 2036
```

"Real-world" data is rarely clean. You have to constantly check it against your intuition and what you know to be true e.g. other sources. Some people think this "data munging" is unsexy but necessary (link). Others come up with cooler names and name their blogs after them (link).

Most points per game (PPG)

Isn't this fun? Let's try answering more substantive questions such as: What was the highest PPG in '12-'13?

```
>   nba$gp <- as.integer(!is.na(nba$min)) # boolean values
>   ppg <- aggregate(nba[,c('pts','gp')], by=list(pid=nba$p
>   ppg$ppg <- ppg$pts / ppg$gp
>   max(ppg$ppg, na.rm=T)
```

```
[1] 28.65672
```

The "aggregate" function subsets a dataset by it's "by" argument and applies the "FUN" argument (which in this case is a function) to each subset. Here, we are calculating the sum of the 'pts' vector and the sum of the 'gp' vector for each 'pid'. This idea is known as split-apply-combine (link).

Most points per game (PPG) cont.

Let's determine the identity of this mysterious high-scorer.

```
>   idx <- which.max(ppg$ppg)
>   ppg$pid[idx] # It's player 1975!
```

```
[1] 1975
```

How about with something a bit more personal?

```
>   plyrs <- unique(nba[,c('pid','player')])
>   ppg <- merge(ppg, plyrs, by='pid')
>   ppg$player[idx]
```

```
[1] "Carmelo Anthony"
```

We use "unique" to get a mapping from pid to the player's name. Then we use the "merge" function to add player names to the ppg data frame. Merging is also known as joining in other languages (e.g. SQL).

Most points per game (PPG) cont.

What if we want to know the top 5 scorers? The top N scorers (That would be more like it!)?

```
>   N <- 5
>   ord_ppg <- order(ppg$ppg, decreasing=T)
>   ord_ppg[1:N] # returns a sorted vector of indices
```

```
[1]  80 202   6  79 280
```

```
>   ppg_sorted <- ppg[ord_ppg,]
>   ppg_sorted[1:N,]
```

```
     pid  pts gp      ppg           player
80  1975 1920 67 28.65672 Carmelo Anthony
202 3202 2280 81 28.14815    Kevin Durant
6    110 2133 78 27.34615     Kobe Bryant
79  1966 2036 76 26.78947    LeBron James
280 3992 2023 78 25.93590    James Harden
```

More exercises!

1. Calculate per-game stats for your favorite player (e.g. reb, ast, blk) and check that match they ESPN's
2. Which player scored the most total points?
3. Which player gathered the most rebounds per game?
4. Which team scored the most points per game?