SC-Camp 2016

Introduction to R and Data Analysis

Joseph Emeras

ANEC, Luxembourg

(Formerly University of Luxembourg)

UNIVERSITÉ DU LUXEMBOURG

Summary

UNIVERSITÉ DU LUXEMBOURG

Summary

R?

R (pronounced aRrgh – pirate style) is a programming language and environment for statistical computing and graphics

- oriented towards data handling analysis and storage facility
- R Base
- Packages tools and functions (user contributed)
- R Base and most R packages are available from the Comprehensive R Archive Network (CRAN)
- Use R console or IDE: **Rstudio**, Deducer, vim/emacs...
- Comment is #, help is **?** before a function name

UNIVERSITÉ DU LUXEMBOURG

Using R

Installing/using packages

Install and load the `ggplot2` package (even if already installed)

```
install.packages("ggplot2")
library(ggplot2)
```

Or in one step, install if not available then load:

```
require(ggplot2) || {install.packages("ggplot2");
                     require(ggplot2)}
```

Numbers in R: NaN and NA

- NAN (not a number)
- NA (missing value)

Basic handling of missing values

```
> x
[1]  1  2  3  4  5  6  7  8 NA
> mean(x)
[1] NA
> mean(x,na.rm=TRUE)
[1] 4.5
```

Objects in R

Objects in R obtain values by assignment.
This is achieved by the gets arrow, <-, or the equal sign, =.
Objects can be of different kinds.

Data Structures

- scalar:

  s = 3.14

- vector:

  v = c(1, 2, “ron”)

- list:

  l = list(1:10, ‘a’, pi)

- matrix:

  m = matrix(seq(1:6), 2)

- **dataframe**:

  df = data.frame(“col1” = seq(1:4), “col2” = c(5, 6, “cerveza”, 6*7))

- ...

Functions

Functions are created using the function() directive and are stored as R objects just like anything else. In particular, they are R objects of class "function".

```
f <- function(<arguments>) {
## Do something interesting
}
```

Functions in R are "first class objects", which means that they can be treated much like any other R object.
The return value of a function is the last expression in the function body to be evaluated.

Arguments

R functions arguments can be matched positionally or by name. So the following calls to sd are all equivalent

```
> mydata <- rnorm(100)
> sd(mydata)
> sd(x = mydata)
> sd(x = mydata, na.rm = FALSE)
> sd(na.rm = FALSE, x = mydata)
> sd(na.rm = FALSE, mydata)
```

Even though it's legal, I don't recommend messing around with the order of the arguments too much, since it can lead to some confusion.

Argument Matching

You can mix positional matching with matching by name. When an argument is matched by name, it is "taken out" of the argument list and the remaining unnamed arguments are matched in the order that they are listed in the function definition.
e.g. The following two calls are equivalent.

```
lm(data = mydata, y ~ x, model = FALSE, 1:100)
lm(y ~ x, mydata, 1:100, model = FALSE)
```

UNIVERSITÉ DU LUXEMBOURG

Argument Matching

- Most of the time, named arguments are useful on the command line when you have a long argument list and you want to use the defaults for everything except for an argument near the end of the list
- Named arguments also help if you can remember the name of the argument and not its position on the argument list (plotting is a good example).

Defining a Function

```
f <- function(a, b = 1, c = 2, d = NULL) {
}
```

In addition to not specifying a default value, you can also set an argument value to NULL.

Lazy Evaluation

Arguments to functions are evaluated lazily, so they are evaluated only as needed.

```
f <- function(a, b) {
a^2
}
f(2)
```

This function never actually uses the argument b, so calling f(2) will not produce an error because the 2 gets positionally matched to a.

Lazy Evaluation

Another example

```
f <- function(a, b) {
print(a)
print(b)
}
> f(45)
[1] 45
Error in print(b) : argument "b" is missing,
with no default
>
```

Notice that “45” got printed first before the error was triggered. This is because b did not have to be evaluated until after print(a). Once the function tried to evaluate print(b) it had to throw an error.

Using R

Usefull Functions

- List all objects in memory: `ls()`
- Save an object: `save(obj, file)`
- Load an object: `load(file)`
- Set working directory: `setwd(dir)`

Entering Data

Reading CSV or text files

```
# comma separated values
dat.csv <- read.csv(<file or url>)
# tab separated values
dat.tab <- read.table(<file or url>,
    header=TRUE, sep = "\t")
```

Entering Data

Reading data from other software: Excel, SPSS. . .

Excel Spreadsheets – need `xlsx` package

```
read.xlsx()
```

SPSS and Stata both need the `foreign` package

```
dat.spss <- read.spss(<file or url>,
                      to.data.frame=TRUE)

dat.dta <- read.dta(<file or url>)
```

UNIVERSITÉ DU LUXEMBOURG

Data Frames

Most easy structure to use, have a matrix structure.

- **Observations** are arranged as **rows** and **variables**, either numerical or categorical, are arranged as **columns**.
- Individual rows, columns, and cells in a data frame can be accessed through many methods of indexing.
- We most commonly use **object[row,column]** notation.

UNIVERSITÉ DU LUXEMBOURG

Accessing Items in a `data.frame`

Aside with R are provided example datasets, i.e. `mtcars` that can be used

```
data(mtcars)
head(mtcars)
colnames(mtcars)

# single cell value
mtcars[2,3]
# omitting row value implies all rows
mtcars[,3]
# omitting column values implies all columns
mtcars[2,]
```

Accessing Items in a `data.frame`

We can also access variables directly by using their names, either with **object[,"variable"]** notation or **object$variable** notation.

```
# get first 10 rows of variable ‘mpg‘ using two methods:
mtcars[1:10, "mpg"]
mtcars$mpg[1:10]
```

Exploring Data

Description Of Dataset

- Using **dim**, we get the number of observations(rows) and variables(columns) in the dataset.
- Using **str**, we get the structure of the dataset, including the class(type) of all variables.

  dim(mtcars) str(mtcars)
- **summary** when used on a dataset, returns distributional summaries of variables in the dataset.

  summary(mtcars)
- **quantile** function enables to get statistical metrics on the selected data

  quantile(mtcars$mpg)

Exploring Data

Conditional Exploration

- **subset** enables to explore data conditionally

  subset(mtcars, cyl <= 5)

- **by** enables to call a particular function to sub-groups of data

  by(mtcars, mtcars$cyl, summary)

Summary

Install and Run R

1. On your local machine:
   - Find a release that fits your distribution at CRAN Archive http://cran.r-project.org/
   - Install and launch R-Studio https://www.rstudio.com/

2. On the cluster
   First connect to the cluster, then submit a job to run R.

```
(localhost)$> ssh cluster
(frontend)$> oarsub -I -l core=1,walltime="00:30:00"
(node)$> R
```

3. Install and Load a Package

```
(R-shell)$> install.packages("ggplot2")
(R-shell)$> library(ggplot2)
```

UNIVERSITÉ DU LUXEMBOURG

Objectives of this Practical Session

- Being able to plot data
  - histogram for data distribution
  - plot in different colors from different data sources
- Know some tips to organize your data
  - aggregate a dataset by column and apply an aggregation function
  - data.table package for binary search in datasets
  - performance in R operations
- R in parallel
  - on one machine
  - on a cluster with socket communications
  - MPI communications

Exercises

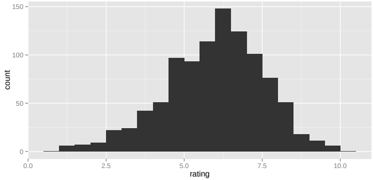
- Start the tutorial https://github.com/sc-camp/camp2016
  - Plot 2 graphs in section Simple Plotting
  - Answer 2 questions at the end of section Organizing your Data
  - Compare performance of aggregation operations w/wo parallelization
- Plot a speedup graph
  - with different number of cores and/or machines
  - needs: ggplot, parallel R

After the Practical Session: Problems

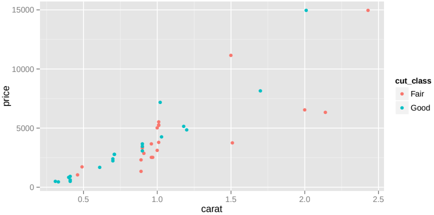
- regression models: example and exercise
- parallelization of K-means Clustering: example and exercise

Simple Plotting

Movies Histogram:



Diamonds Plot with 2 colours:



PS Questions

Question: use ddply instead of tapply in the first example

```
ddply(DT, .(x), summarize, sum(v))
```

Question: return the min and max instead of the sum.

```
min_max = function(data){
              c(min(data), max(data))
        }
DT[,min_max(v),by=x]

## or

DT[,c(min(v), max(v)),by=x]
```

Questions?

Usefull links

- CRAN Archive http://cran.r-project.org/
- ggplot2 Documentation http://docs.ggplot2.org/current/
- CRAN HPC Packages http://cran.r-project.org/web/views/HighPerformanceComputing.html
- Advanced R programming by Hadley Wickham http://adv-r.had.co.nz/

UNIVERSITÉ DU LUXEMBOURG