NOVA IMS Information Management School

Group Project Seminar 2014/2015

4th lecture, 6th Feb 2015

Júlio Caineta

jcaineta@novaims.unl.pt

Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

By the end of this class you will know how to

- Load, save and edit data in R
  - R Objects
  - Different file formats
- Prepare data
- File handling

Entering data

- Do you have a small number of observations?
  - Enter directly in R, e.g., creating vectors

```
> salary = c(18700000, 14626720, 14137500, 13980000, 12916666)
> position = c("QB", "QB", "DE", "QB", "QB")
> team = c("Colts", "Patriots", "Panthers", "Bengals", "Giants")
> name.last = c("Manning", "Brady", "Pepper", "Palmer", "Manning")
> name.first = c("Peyton", "Tom", "Julius", "Carson", "Eli")
```

It is convenient to have all those related data sets in a single data structure? Then create a data frame

```
> top.5.salaries = data.frame(name.last, name.first, team, position, salary)
```

Saving and Loading R objects

- The simplest way is with the **save** function
- Always specify paths with forward slashes /
- The **file** argument must be explicitly named
- Cross platform
- Save the whole workspace with the **save.image** function
- ASCII or binary
- Load with the load function

```
> save(top.5.salaries,file="~/top.5.salaries.Rdata")
> save(top.5.salaries,
+
file="C:/Documents and Settings/me/My
Documents/top.5.salaries.Rdata")
> load("~/top.5.salaries.RData")
```

Saving and Loading R objects

- Let us try now saving and loading some existing variable on our workspace
  - List existing variables with the **ls** function
  - **save** and **load**
  - When saving, try different arguments values and check what happens to the generated file

Load data from other sources

- You can load (and save) data from different types of files
  - Text files (csv, tsv, etc.)
  - Excel (xls) – it is not easy nor advisable
  - SAS
  - Open/Libre Office (ods)
  - Shapefile
  - Raster
  - And many others
- R can load data from files on your computer or on the net

Text files

- Most of them have a similar structure
  - Each line represents an observation (or record)
  - Each line contains a set of different variables associated with that observation
  - Different variables may be separated by a special character called the delimiter (delimited format)
  - Other times, variables are differentiated by their location on each line (fixed format)

Delimited text files

- R provides a family of functions for reading this kind of file, based on the **read.table** function
- It returns a **data.frame** object
  - Rows → Observations
  - Columns → Variables
  - Values separated by a delimiter
- This functions is very flexible, it can load files with different properties
- Check its help entry (**?read.table**)

Let's try to load a CSV file

- Here is an example for us to try
- Open this file with a text editor: http://goo.gl/sa7cOz
- Analyse how it is organised
- Load it into R

Let's try to load a CSV file

- We've learned that this file
  - The first row contained column names (**header=TRUE**)
  - The delimiter was a comma (**sep=","**)
  - Text is encapsulated with quotes (**quote="\""**)

```
> top.5.salaries <- read.table("top.5.salaries.csv",
+ header=TRUE, sep=",", quote="\"")
```

Delimited text files

- The read.table function has a family
  - read.csv and read.csv2
  - read.delim and read.delim2

Function	header	sep	quote	dec	fill	comment.char
read.table	FALSE		\" or \'	.	! blank.lines.skip	#
read.csv	TRUE	,	\"	.	TRUE	
read.csv2	TRUE	;	\"	,	TRUE	
read.delim	TRUE	\t	\"	.	TRUE	
read.delim2	TRUE	\t	\"	,	TRUE	

Source: R in a Nutshell

More on loading text files

- Fixed format files can be read with the **read.fwf** function
- **readLines** reads data one line at a time
- You can read text files with a more complex format with the **scan** function, where you can define the data structure

Writing text files

- R can also export data objects (usually data frames and matrices) as text files
- Use the **write.table** function to save your data as text files
- It has an family of functions analogous to **read.table**

Now let's try: add a column to the previously loaded file and write it back again

Shapefile (vector data)

- There are a few different ways, we are going to use the package **maptools** (and **rgeos**)
  - Let us try with an example from the IGeoE ("Army Geographical Institute"): http://www.igeoe.pt/index.php?id=86
  - Load the file *ShapeFile/ALTIMETRIA/576CurvasDeN°vel.shp* (it is a map with contour lines) with **readShapeSpatial** or **readShapeLines**
  - You can see a simple but nice graphic with the **plot** function

Raster (gridded data)

- We are going to use the packages **rgdal** and **raster**
- Convert as raster image and plot using default colours

```
> raster_data = raster(raster_file)
> plot(raster_data, main="Title", xlab="x axis", ylab="y
axis")
```

Try with any example from the IGeoE website

Preparing data

- It may happen that most of the time you spend on data analysis projects is spent on preparing data for analysis
- But if we get data from a...
  - web server
  - financial database
  - or some kind of electronic records

  doesn't it all come from computers? What is the big deal?
- In practice,
  - data is almost never stored in the right form for analysis
  - there are often surprises in the data
  - it takes a lot of work to pull together a usable data set

Combining data sets

- It is common to work with data that is stored in different places
- R provides several functions to turn multiple data structures into a single one
  - **paste**
  - **rbind** (rows, combine vertically)
  - **cbind** (column, combine horizontally)

```
paste (..., sep = " ", collapse = NULL)
```

Merging by common fields

- **merge** joins two data frames by common columns or row names
- You can use **intersect** to check common columns in two data frames

```
> merge(df1, df2)
> intersect(df1, df2)
```

Transformations

- It may happen that some variables aren't quite right and they need some kind of transformation:
  - Reassigning variables (**as** family of functions)
  - Transform existing or new variables

```
> transform (df, col1 = expression, new_col = expression)
```

  - Apply a function (you already know this one)
    - The **plyr** package comes to the rescue!

The **apply** family!

Input	Array Output	Data Frame Output	List Output	Discard Output
Array	aaply	adply	alply	a_ply
Data Frame	daply	ddply	dlply	d_ply
List	laply	ldply	llply	l_ply

Source: R in a Nutshell

Binning

- Group a set of observations into bins based on the value of a specific variable (binning)
  - **shingle:** overlapping intervals
  - **cut:** continuous to discrete
  - **make.groups:** combine similar objects with a column labelling the source (in the **lattice** package)

Subsets

- With subsets you can analyse just a part of your data set
  - Bracket notation (indexes, ranges or logical)
  - **subset** function (more readable)
  - **sample**: random sampling (more complicated methods in the **sampling** package)

Summarising

- You may have too fined data for your analisys
- Group together records to build a smaller data set
  - **tapply** (here are they again!)
  - **by**
  - **aggregate**
  - **rowsum**
  - Count values (**tabulate**, **table**, **xtabs**)

Reshaping data

- Data may have the wrong “shape”
  - **t** is for transpose
  - **stack** and **unstack**
  - **reshape** (powerful, but too confusing... pretty much like the others)
  - Thanks Hadley for **reshape-**ing R!
    - **melt**
    - **cast**

Data cleaning

- Even when data is in the right form, there are often surprises in the data
  - Doesn’t mean changing the meaning of data
  - But identifying problems caused by data collection, processing, and storage processes
  - Modifying the data so that these problems don’t interfere with analysis

Find and removing duplicates

- Depending on how you plan to use the data and on the type of data, duplicates might be an issue
  - **duplicated**
  - **unique**

Sorting and ranking

- Two other operations that you might find useful for analysis
  - Sorting (**sort**)
  - Ranking (**order**, for data frames use **do.call(order, df)**)

File handling

- OK, dealing with one file is kind of easy. What if you have a million of files organised in several different folders?
- R saves your day
  - **list.files**
  - **file.info**
  - **file.choose**
  - And much more, see **?file**