Data Analysis and Visualization with R

Visualizar ’11

Felipe Ortega

jfelize@libresoft.es — @jfelize
GSyC/Libresoft – URJC

June 22, 2011

Universidad Rey Juan Carlos
Introduction
Graphics for Data Analysis
Advanced Graphics in R
References

(cc) 2011 Felipe Ortega.
Some rights reserved. This work is licensed under a Creative Commons Attribution-Share Alike 3.0 License, available at http://creativecommons.org/licenses/by-sa/3.0/
What is GNU R?

- A powerful, easy-to-use statistical software package
- It is *libre software*
- It is *free* and *extensible*
- 2,554 packages available on R CRAN (exp. growth)
Multiplatform support: GNU/Linux, MacOS (and, yes... Windows, as well).

**R CRAN**: Resources for GNU R:
http://lib.stat.cmu.edu/R/CRAN/.

Documentation (on R CRAN).
1. *Packages*: browse the complete list, descriptions, dependencies.
2. Official manuals.
3. Contributed manuals.
4. FAQs
Installation

- Installing R in Debian-like systems is easy:
  1. `fenix@blackstorm:~$ sudo apt-get update`
  2. `fenix@blackstorm:~$ sudo apt-get install r-base r-cran-rmysql`
  3. Other useful packages: r-cran-lattice r-cran-latticeextra r-cran-hmisc...

- Then, to start R just type:
  `fenix@blackstorm:~$ R`

  
  >

  You are now in the R environment, with its own command line
R libraries are packages that provide collections of functions and data sets.

They are the best solution to perform many statistical analyses.

```r
library('libName') #Load library
Example:
> library(MASS)
Install libraries: (execute R with sudo)
> install.packages('ISwR', dep=T)
```
Getting help at R command line:

1. Help about functions (usage, args...).
   ```
   > help(funcName)
   > ?funcName
   ```

2. Search through help doc
   ```
   help.search('word')
   > apropos('word')
   ```

3. What is inside a library? (intro & summary of data sets, functions, etc.)
   ```
   > library(help=MASS)
   ```
Operations and Vectors

- R is a big calculator...

```r
> a=1; b=2; c=3; a+b/c
[1] 1.666667
```

- Vectors: arrangements of objects (numeric, character, etc.)

```r
> v1 = c(1,2,3,4,5)
> v1
[1] 1 2 3 4 5
> seq(1,20,2)
[1]  1  3  5  7  9 11 13 15 17 19
> sqrt(-1+0i)
[1] 0+1i
```
Vectors (cont.)

- Vectorized arithmetic.
  
  ```r
  > v2 = v1^2
  > v2
  [1] 1 4 9 16 25
  ```

- Indexing
  
  ```r
  > v2[2]
  > v2[3]=0
  > v2[c(1,2,3)]
  ```

- Sorting
  
  ```r
  > sort(v1)
  > rev(sort(v1))
  ```
Basic data types (I)

> a = 2 # Plain numbers
> class(a)
[1] "numeric"
> s = "hello world" # Character strings
> class(s)
[1] "character"
> l = list(x=2, y=3, z="peanut") # Lists of variables
> class(l)
[1] "list"
> l
$x
[1] 2
$y
[1] 3
$z
[1] "peanut"
> # Matrices: vector arrangement for 2 dimensions
> m = matrix(data=c(1:9), nrow=3, ncol=3)
> m

   [,1] [,2] [,3]
[1,]  1  4  7
[2,]  2  5  8
[3,]  3  6  9
> class(m)
[1] "matrix"
>
> # Arrays for vectors with more than two dimensions
> ar = array(data=1:12, dim=c(2,2,3))
We will visit other (important) object classes, later on:

- **Factors**: Categorical variables
- **Data frames**: Tabular data (row and column names).
- **Formulas**: Expressions to define mathematical models.

More about classes and internal storage types in R:

- `>` `?class` # Attribute class of R objects
- `>` `?typeof` # Internal storage type
# Control flow (I)

# Conditions
if(cond) expr
if(cond) cons.expr else alt.expr

# Loops
for(var in seq) expr
while(cond) expr
repeat expr
break # Break loop iteration
next # Skip to next iteration
Control flow: examples (II)

```r
> x = 1; y = 2
if (x > y) print(x) else print(y)
# Don't forget curly braces for blocks with +1 line!!
>
> j = 0
> repeat {if (j < 10) print(j) else break; j = j + 2}
>
> while (k < 10) {
+   print(k)
+   k = k + 1
+ }
```
> # For loops
> for (i in 1:10) print(i)
> for (z in seq(from = 0, to = 2*pi, by = pi/2))
+   print(paste("sin(" , z , "") = ", sin(z), sep=""))
> # Beware of round errors!!
>
- Install iterators package to use iter() function (iterators).
- Install foreach package to use foreach() function.
R environment

1. List all objects in *working space*.
   
   ```
   > ls()
   ```

2. Save individual variables to file.
   
   ```
   > save(‘filename’)
   ```

3. And restore them later.
   
   ```
   > load(‘filename’).
   ```

4. Save the entire workspace:
   
   ```
   > save.image(‘filename’)
   ```

5. Delete objects
   
   ```
   > rm(obj1, obj2)
   ```

6. Redirect R output to a file
   
   ```
   > sink(‘myfile’)
   > sink() #To recover output in R again
   ```
Automate execution of code, speed up analysis, include comments, avoid input errors.

- Just type your commands on a common text file, save it with .r or .R extension.
- Execute your script, either interactively:
  ```
  > source('myScript.R')
  ```
- or else, from the command line (batch mode):
  ```
  phoenix@blackstorm:~$ R --vanilla < myScript.R
  ```
Loading data: files

- [Link to R data documentation](http://cran.r-project.org/doc/manuals/R-data.html)
- Read CSV or customized data files.
- Just need to specify separator and quote characters, header present or not, etc.

```r
> # function to read tabulated data from files
> ?read.table
> fortune1000 = read.table("fortune1000.csv",
  + header=T, sep="","", quote="", row.names=2)
> # Now, you can operate with these values
> # Order by rank, take top 10 companies
> fortune1000[order(fortune1000$rank)[1:10],]
```
Loading data: library data sets

- Load data from existing libraries.

```r
library(HSAUR2)
data(Forbes2000)
head(Forbes2000)
summary(Forbes2000)
```
Categorical data in R: class factor

- Function: `factor()` #create data from character vars
- `levels()`: returns names of categories.
- `nlevels()`: returns num. of diff. categories

```r
> Forbes2000$country = factor(Forbes2000$country)
> table(Forbes2000$country)[order(table(Forbes2000$country), + decreasing=T)]
```
Loading special values: date and time

- `as.Date('value', format='format_string')`
- Example: 'October 27, 2010', format='%B %d, %Y'
- Example: '2011-06-19', format='%Y-%m-%d'
- Advanced support: `as.POSIXlt()` for full format and `as.POSIXct` for epoch style.
- Extract info with weekdays, months or quarters.
- `strptime` to format input and `strftime` to format output.

```r
> tw_dates = read.table("twitter_egypt/tweet_created_at.csv", + header=T, sep="",")
> tw_dates$created_at = as.POSIXlt(tw_dates$created_at)
> # distribution of tweets about Egypt over week days.
> table(weekdays(tw_dates$created_at))
```
Simple stats

```r
> length(v1)
[1] 5
> mean(v1)
[1] 3
> sum(v1)
[1] 15
> median(v1)
[1] 3
> summary(v1)

Min. 1st Qu.  Median    Mean  3rd Qu.    Max.  
 1      2      3      3      4      5

Special attention to the cool summary(...) function
```
Data Frames (I)

- Data frames let you arrange values of different variables for each sample.
- Rows correspond to samples and columns to variables values for each sample.
- Examples with Animals data frame from library 'MASS'.
  ```r
  > head(Animals) # From library MASS, loaded by default
  ```
- You can retrieve the values for each column either using its index or its name (the latter is recommended).
  ```r
  > Animals$body
  > Animals[,1]
  ```
Data Frames (II)

- Summarizing data frames.

  ```r
  > summary(Animals)
  body           brain
  Min.    : 0.023  Min.    : 0.40
  1st Qu.:  3.100  1st Qu.: 22.23
  Median : 53.830  Median : 137.00
  Mean   :4278.439  Mean   : 574.52
  3rd Qu.: 479.000  3rd Qu.: 420.00
  Max.   :87000.000  Max.   :5712.00
  ```

- Access a column without using $ syntax.

  ```r
  > with(Animals, body)
  ```
Data Frames (III)

- Automatically select rows that satisfy a certain condition.

```r
> # All columns
> heavy = Animals[, Animals$body > 7000,]
> heavy

body   brain
Dipliodocus 11700  50.0
Triceratops  9400  70.0
Brachiosaurus 87000 154.5

> # Certain columns
> Animals[, Animals$body > 7000, 1]
```
Loading data from DBs: libre software projects

- 3rd data set: summary of metrics about libre software projects.
- File summaryFM3.sql

1. `fenix@blackstorm:~$ mysql -u user -ppassword < summaryFM3.sql`
2. `fenix@blackstorm:~$ mysql -u user -p`
3. `mysql> use fm3_classify`
4. `mysql> describe summary`
You can access MySQL DBs directly from GNU R.

All you need:

```r
> library(RMySQL)
> con = dbConnect(MySQL(), user="phoenix", pass="secret", db="fm3_classify") #Open DB connection
> dbListTables(con)
> dbListFields(con, "summary")
> #Execute query, store results in a data frame
> dbGetQuery(con, "select * months, ncommitters from summary")
> dbDisconnect(con) #Close DB connection
> ?RMySQL # more examples
```
Scatterplots

- The `plot(...)` function is your friend.
  > `plot(Animals$body, Animals$brain)`

- The short way.
  > `with(Animals, plot(body, brain))`

- Let’s try to improve it
  > `with(Animals, plot(body, brain, log="xy"))`
  > `with(Animals, plot(body, brain, log="xy", xlab="body", ylab="brain", main="Plot of brain~body in 28 species"))`

- Other useful functions.
  - `lines()`
  - `points()`
  - `abline()` # Draw fits
> with(Animals, plot(body, brain, log="xy", col="navy", xlab="body", ylab="brain", main="brain vs. body"))
> text(1e+03,5, paste("By the way, v1[1] = ", v1[1]))

By the way, v1[1] = 1
Exercise

> con = dbConnect(MySQL(), user="phoenix", pass="secret", + db="fm3_classify")
> floss_proj = dbGetQuery(con, "select project, months, + ncommittees + from summary + where months > -1 and ncommittees > -1")
> dbDisconnect(con)

- Explore the distribution of values using the tools from previous slide.
Histogram and KDE

```r
> with(floss_proj, hist(log10(ncommitters), freq=F))
> with(floss_proj, lines(density(log10(ncommitters)), col="red", + lwd=2, lty=2))
```
> boxplot(floss_proj, log="y")
Correlations (exercise)

```r
> con = dbConnect(MySQL(), user="phoenix", pass="secret", +
  + db="fm3_classify")
> floss_bugs = dbGetQuery(con, "select ncommits, ncommitters,
  + nreporters, nreports
  + from summary
  + where ncommits > 0 and ncommitters > 0
  + and nreporters > 0 and nreports > 0")
> dbDisconnect(con)
> pairs(log10(floss_bugs), panel=panel.smooth)
> example(pairs) # Customization examples
```

- Use `cor(floss_bugs)` to get Pearson correlation coeff.
- What can we say about the relationship between these variables?
3D plots

```r
> grid = matrix(rnorm(100), nrow=10, ncol=10)
> persp(grid, col="orange", border=NA, expand=.2, theta=45, phi=25,
> + ltheta=45, lphi=20, shade=.5)
```

Perspective normal surface
Working with several plots

- You can create many plots at the same time.
- Use dev functions to operate with plots.

```
> dev.new()  # Open new plot with next sequential ID
> dev.set(numPlot)  # Set numPlot as the active plot
> dev.prev()  # What's the previous plot ID?
> dev.next()  # What's next plot ID?
> dev.off(numPlot)  # Close plot identified by numPlot
> dev.off()  # Close current plot
```
Save your plots

- You can save your plots to files with special devices. Just call the device before the plotting commands, plot your graph and then close the device.

- Example

  ```r
  > postscript("myFig.eps")
  > plot(....)
  > # Other plotting commands
  > dev.off()
  ```

- Possible devices

  ```r
  > ?Devices #png, jpeg, bmp, tiff, pdf, etc.
  ```
The goal is to fit a model characterizing the behavior of an output (response or dependant) variable, according to one or several input (predictor, independent or explanatory) variables.
Preparing your linear model

- Identify your response variable
- Identify the explanatory variable(s).
- Are the explanatory variables continuous, factors (categorical) or mixed?
- What’s the type of your response variable (continuous, factor, count, proportion, binary, time at death, category)
Available linear models

Explanatory variables

- All continuous: Regression.
- All categorical: Analysis of Variance (ANOVA).
- Mixed: Analysis of Covariance (ANCOVA).
Available linear models

Output variable

- Continuous: Standard (simple/multivariate) regression, ANOVA, ANCOVA.
- Proportion: Logistic regression.
- Count: Log-linear models (Poisson regression).
- Binary: Binary logistic analysis.
- Time at death: Survival analysis.
lm() function in R

- `lm(Formula, data = dataset)` let us fit linear models to data.
- We need to write the adequate formula to build the model.
- `lm(y ~ x0 + x1 + x2 + x3, data = myDataFrame)`
- We can express more complicated interactions (still linear models):
  - `lm(y ~ x0 + x1 * x2 + log(x3), data = myDataFrame)`
Plotting regression lines

- \( fit = \text{lm}(y \sim x, \text{data} = \text{dataset}) \)
- Now we can display a scatterplot and the regression line:
  - \( \text{plot}(x, y, \text{data} = \text{dataset}) \)
  - \( \text{abline(coef(fit), lty = 5, col = "green")} \)
- And we can also inspect the fit, and the correlations:
  - \( \text{summary(fit)} \)
  - \( \text{cor}(x, y) \) (also works \( \text{cor(myDataFrame)} \)).
Exercise

- Fit a simple linear regression line to describe the relationship between number of bug reports and number of bug reporters in FLOSS projects.
Lattice: Multivariate data visualization in R

- Package created exclusively for multivariate data visualization.
- Display variable or relationship between variables conditioned to value of one or more other variables (Trellis graphs).
- Deals with aspect ratio (banking to 45°).
Lattice: Density plots

```r
> con = dbConnect(MySQL(), user="phoenix", pass="secret", +
+ db="wx_eswiki")
> page_len = dbGetQuery(con, "select page_id, page_namespace, +
+ page_len from sample_page +
+ where page_namespace not in (2,3,5,104)"
)>
> dbDisconnect(con)
> page_len$page_namespace = factor(page_len$page_namespace,
+ levels=c(0,1,4), labels=c("Main","Talk","Wikipedia"))
> densityplot(~ page_len | page_namespace, data=page_len, ref=TRUE)
```
Lattice: Density plots

![Density plots for Main, Talk, and Wikipedia categories](chart.png)

- **Main** category shows a broader distribution with a peak around 1e-05.
- **Talk** category has a more concentrated distribution with a peak around 2e-05.
- **Wikipedia** category displays a sharp peak around 4e-05.

The x-axis represents `page_len` ranging from 0 to 4e+05, while the y-axis shows the density on a log scale from 0.0e+00 to 2.5e-05.

**References**

- **Lattice**: Density plots
- **Felipe Ortega**: Data Analysis and Visualization with R
Lattice: Density plots (groups)

```r
> densityplot(~ page_len, groups = page_namespace,
+ data=page_len, ref=TRUE, auto.key = TRUE)
```

![Density plot with groups](image)
Lattice: Ecdf

\[
\text{> ecdfplot(~ page_len | page_namespace, data=page_len, ref=TRUE)}
\]
Lattice: Barcharts

```r
> tab_tw_dow = table(strftime(tw_dates$created_at, format = "%w"))
> barchart(tab_tw_dow)
```

![Barchart example](image)
Lattice: Barcharts

```r
> revs_dow_month = dbGetQuery(con,
+  "select WEEKDAY(rev_timestamp) AS dow, 
+  MONTH(rev_timestamp) AS month from sample_rev")
> barchart(tab_revs_dow_month, stack=FALSE,
+  auto.key=list(space = "right"))
```
Lattice: Multivariate linear models

Figure 5.10

http://lmdvr.r-forge.r-project.org/figures/figures.html
Ggplot2: Elegant graphics

- New grammar for (appealing) graphics.
- Smooth lines.
  - http://had.co.nz/ggplot2/stat_smooth.html
- Coordinate maps.
  - http://had.co.nz/ggplot2/coord_map.html
- Smooth gradients.
  - http://had.co.nz/ggplot2/scale_gradientn.html
- Cartesian coordinates.
  - http://had.co.nz/ggplot2/coord_cartesian.html
  - http://had.co.nz/ggplot2/coord_trans.html
- Polar coordinates
  - http://had.co.nz/ggplot2/coord_polar.html
Network analysis: Statnet

- [http://statnetproject.org/](http://statnetproject.org/)
- Support for SNA, dynamic graphs, directed graphs, and much more.
- Check project website for documentation
- Tutorial: [http://www.jstatsoft.org/v24/i09/paper](http://www.jstatsoft.org/v24/i09/paper)
- Video tutorials:
- Also check other popular package: igraph.
Good references for GNU R.

Journals and web resources

1. **Journal of Statistical Software.**
   - [http://www.jstatsoft.org/](http://www.jstatsoft.org/)

2. **The R Journal.**
   - Former R News, offers informative articles on libraries and new releases.
   - [http://journal.r-project.org/](http://journal.r-project.org/)

3. **Websites and blogs.**
   - R-Forge (828 projects): [https://r-forge.r-project.org/](https://r-forge.r-project.org/)
References on statistics

Linear models with R