

An Introduction to R

Chris Hanretty

Introduction

- Using R as a calculator
- Storing things how?

Data in, data out

- Reading data from plain-text files
- Data from other stats packages

Subsetting, indexing

Plotting

Regression

1. Introduction

Using R as a Calculator

- Enter `1+1` and hit return. What happens?
- Standard mathematical operators are supported
- division (`/`), multiplication (`*`), exponentiation (`^`), `sqrt`

Using R as a Calculator (2)

- What if we want to hold on to some value?
- We can use the assignment operator (<- or =)
- `pizza_area<- 2*pi*6^2`
- Anything not assigned is *evaluated* (usually printed to screen)

Using R as a Calculator (3)

- `abs, sign`
- `exp, sqrt, log`
- `floor, ceiling, trunc, round, signif`
- `cos, sin, tan, acos, asin, atan`
- `mean, median, sd, var, sum`

Storing things how?

- R has different ways of storing data
- Enter `class(pizza_area)`. What happens?
- Enter `class("pizza_area")`. What happens?

Storing things how?

The main kinds of object you'll have to deal with are

- `vector`, `list`, `matrix`, `factor`,
`data.frame`

These objects can store data in different *types*

- `logical`, `character`, `numeric`, `[ordered]`

Storing things how?

		data-type	
		same	different
dim.s	one	vector	list
	multiple	matrix	data.frame

Building data with different types

- Suppose we're at the pizzeria san Domenico
- `pizza_lengths <-c(6,7,9,11,12.5)`
- `pizza_types <-c("Crispy", "Crispy", "Pan", "Pan", "Stuffed Crust")`
- `is.vector(pizza_lengths)?`
`is.vector(pizza_types)?`

Grand pizza theory

- Let's link pizza radius and crust type
- We could run the two together:
`pizze<-c(pizza_lengths,pizza_types)`
- What happens to our numbers?
- `mode(pizze)`

Grand pizza theory (2)

- Better to put them in a data frame
- `pizze<- data.frame(pizza_lengths, pizza_types)`
- What happens if you type `pizze`?
- ... and `dim(pizze)`? `summary(pizze)`?

The pizza data-frame

- How do I access my variables?
- `pizze$pizza_lengths`
- Isn't that a bit long-winded?
- `names(pizze)`
- How to change this?

Grand pizza theory (3)

- What kind of variable is pizza crust?
Nominal or ordinal?
- `is.factor(pizze$types)?`
`levels(pizze$types)?`
`ordered(pizze$types)`
- Let's make pizza crust an ordinal variable

Grand pizza theory (4)

- `pizze$types <- factor(pizze$types, levels=c("Crispy", "Pan", "Stuffed Crust"), ordered=TRUE)`
- And if you make a mistake?

Creating new variables

- Creating new variables is easy
- Let's change sizes into centimetres
- `pizze$length2 <-pizze$length*2.54`
- `pizze$type2 <-pizze$type*2.54?`

What do we now know?

- You should now understand the different types of data R stores;
- and how it stores them
- You'll typically be working with `data.frames` storing factors or numbers
- but sooner or later you will slip up with your `factor` levels or character vectors that you thought were `numeric`

2. Data in, data out

The basic case

- The simplest kind of data is plain-text data
- You have values, separated by a delimiter
- That might be a tab, or it might be a comma

Example: comma separated values

"Canton","Population","Protestant","Catholic","Other-Catholic","Orthodox","Jewish","Muslim","None","Others","Yes vote"
"Vaud",640657,256507,215401,491,10560,2062,24757,89405,41474,46.90
"Valais",272399,17186,221146,131,3092,145,7394,10750,12555,58.00
"Genève",413673,72138,163197,610,7166,4356,17762,93634,54810,40.30
"Berne",957197,642297,153357,1064,9153,807,28377,74162,47980,60.70
"Fribourg",241706,36819,170069,162,1961,138,7389,14500,10668,55.90
"Soleure",244341,76292,106263,1876,3561,91,13165,33244,9849,64.00
"Neuchâtel",167949,63974,51257,559,1102,266,5056,36582,9153,49.20
"Jura",68224,8513,51092,55,270,22,1310,4250,2712,51.20
"Bâle-Ville",188079,51103,46802,519,4783,1421,12643,58334,12474,48.40

Example: tab separated values

"Canton"	"Population"	"Protestant"	"Catholic"	
"Vaud"	640657	256507	215401	
"Valais"	272399	17186	221146	
"Geneve"	413673	72138		163197
"Berne"	957197	642297	153357	
"Fribourg"	241706	36819	170069	
"Soleure"	244341	76292	106263	
"Neuchâtel"	167949	63974	51257	
"Jura"	68224	8513	51092	
"Bâle-Ville"	188079	51103	46802	

Reading this kind of data...

- ... is fairly simple
- `read.table`, `read.csv`
- default behaviour -- first line contains variable names, convert character variables to factors,

```
ref<-read.csv("http://www.chrishanretty.co.uk/swiss.csv",  
             header=T)  
summary(ref)  
head(ref)
```

Other stats packages

- What if you have your data in a proprietary format?
- You'll need to load the `foreign` package and do it that way

```
library(foreign)
data<-read.spss(file="mydata.sav")
data<-read.dta(file="mydata.dta")
```

Inbuilt data sets

- Finally, you might want a data-set from a particular package
- Suppose we're interested in the Duncan data-set from the `car` package

```
library(car)  
data(Duncan)  
data<-Duncan
```

or

```
data("Duncan", package="car")
```

Save data

- STOP! Why do you want to save data?
- If you want to share with others, then try `write.csv(pizze, file="mypizze.csv")`
- If you want to save a specific R object, `save(pizze, file="mypizze.RData")`
- If you want to save everything, `save.image(file="mydata.RData")`

3. Subsetting, indexing

The first few rows

- What do we do after loading data (say, called `ref`)?
- we can try `ref`
- but not if data is very large
- we can try `summary(ref)`
- or we can just look at the first few lines
- `head(ref)`

The first few rows (2)

- `head(...)` gives us the first ten rows
- but we can do this another way:
- `ref[1:10,]`
- What about `ref[1:4,]`?
- and `ref[,1:4]`?
- and negative numbers?

Indexing (1)

- Select any cell in a data frame or matrix with

`data[R,C]`

- Remember, matrices are *Roman Catholic!*

Indexing (2)

- We can select arbitrary row- and column-combinations
- Suppose we want the fifth and sixth columns of the second and third rows

```
ref [c(2,3) , c(5,6)]
```

Indexing (3)

- Even better, we can select rows based on the values of variables!
- Suppose we're interested in majority Protestant cantons, i.e., where `ref$Protestant > 0.5`
- What happens when you type that in?
- Now what happens when you put that into `ref[...]`
- And how would you select just the variable "Yes.vote"?

Indexing (4)

- This can quickly get complicated pretty

```
ref$Yes.vote[(ref$Protestant < 0.5  
& !(ref$Catholic > 0.5))]
```

- (If you're really only interested in part of the data-set, consider subset)

4. Plotting

Plotting

- What's the second thing we do after loading a data-set?
- we can either do a scatterplot of variables of interest:
- `plot(ref$Muslim, ref$Yes.vote)`
- or we can be fancy...
- `pairs(ref)`

Over-plotting

- Once we have the basic plot, we can add elements
- Say we want to draw a local regression line
- `lines(lowess(ref$Muslim, ref$Yes.vote), col="red")`
- See `?plot.default` for more details

5. Regression

Regression

- What's the third thing we do with our data after inspecting it and visualizing it?
- We model it!
- Since we're ultimately interested in causality rather than data reduction, let's start with OLS regression

Let's get some suitable data

- Load the car package (`library(car)`)
- We're going to load a pre-stored dataset on occupational prestige
- `data(Duncan)`
- Let's check out our data (`head`, `summary`)

The R formula interface

Suppose you have a linear regression of the form:

$$y = \alpha + \beta x + \epsilon$$

In R this becomes

$$y \sim x$$

You can make the intercept explicit:

$$y \sim 1 + x$$

or disable it:

$$y \sim 0 + x$$

The `lm` command

- This formula interface is accepted by a number of commands
- The workhorse command is `lm`
- Suppose prestige is a linear function of income.
- `my.mod1<-lm(prestige~income, data=Duncan)`

Adding terms

- Or maybe it's a function of income and req'd education.
- `my.mod2<-lm(prestige~income+education, data=Duncan)`
- So, add terms by... umm, '+'-ing them.

Interaction terms

- Or are there interactions between education and income?
- `my.mod3<-lm

```
(prestige~income*
education, data=Duncan)
````
- What's the difference between '*' and ':'?

LM objects

- Our models are now stored in a variety of `lm` objects
- We can now apply various functions to these objects
- `coef`, `residuals`, `fitted`,
- `logLik`, `AIC`
- `predict`, `confint`

Plotting regression diagnostics

- We can also plot regressions
- just try `plot(my.mod2)`
- Single best R feature?

How much prestige will we have?

- Suppose we're the average Canadian
- Let's predict our prestige
- `predict(my.mod2, newdata = data.frame(income = mean(Duncan$income), education = mean(Duncan$education)))`

Want some extra quantities with that?

- Add confidence intervals
(...interval="confidence")
- Provide a range of values
(income=quantile(Duncan\$income))

Beyond OLS

- Other R commands exist for other types of regression
- For dichotomous outcomes, use `glm` with `family="binomial"`
- For count data, use `glm` with `family="poisson"`
- For discrete ordered outcomes, use `polr` in the MASS package
- For duration models, check out the `survival` package

Multi-level stuff

- You can also do multi-level models.
- This requires the `lme4` package
- Formula interface extended:

$y \sim x + (1 | \text{group})$ for fixed effects,

$y \sim x + (z | \text{group})$ for random effects of z
according to group.

Also, the `plm` package for TSCS.