Intro to modeling longitudinal outcomes:
R basics

Dr Cameron Hurst
cphurst@gmail.com

CEU, ACRO and DAMASAC, Khon Kaen University

4th February 2557
Nearly 20 years experience as biostatistician having consulted in thousands of health and medical research projects (Pre-clinical->clinical->population-based)

Hold a joint appointment with Faculties of Public Health (Teaching appointment) and Medicine (Clinical biostatistician), KKU

Main role (PH) is in the supervision of PhD students and (Med) consulting regarding clinical research

Also teaching into courses (like this) in research methods and biostatistics

Come from Queensland in North-eastern Australia (Australia’s version of Isaan)
What we will cover (Workshop series)

Four sessions:

1. **Session 1 AM: Intro to R basics**
   Data I/O, Basic data structures and graphics, intro to modeling in R

2. **Session 2 AM: Continuous longitudinal outcomes**
   Linear Mixed model. Differences between conditional and marginal models

3. **Session 3 PM: Categorical longitudinal outcomes**
   Generalized Estimating equations (marginal model) and Generalized Linear Mixed Model (conditional)

4. **Session 4 PM: Hands on exercises with R**
Before we start, I will just point out a few conventions I will use:

**Note:**
Things to note given in a green box

**Pitfalls:**
Common mistakes and things to watch out for given in a red box

**R SYNTAX:**

**Important** R syntax will be in purple boxes and be in **courier** font. This will help you find it easily when you have to refer back to these notes.
Before we move onto R, I would just like to make one important point:

**You are not expected to learn R syntax!!!!**

You are just expected to remember which session you covered a particular topic, and refer back to those notes (or other resources....e.g. R help files).

**Hint: Learning R**

**USING (not memorizing) syntax is the best way to learn it**
What we cover today (This session)

1. About R
   - What can R do?
   - Downloading R
   - R studio and other R IDEs

2. Data Structures and I/O
   - Basic conventions and data structures
   - Reading in data and data frames
   - Libraries

3. Basic statistics and plots
   - R graphics: Univariate
   - R graphics: Bivariate
   - Introduction to modeling in R

4. Getting help
R is freely available open-source (⇒free) software created under the GNU agreement

It brings together the basic functionality of the creators of R (core packages) and work from others such as you and I (contributed packages)

In this respect R is at the cutting edge in a way that the expensive competitors (i.e. SAS, Stata and SPSS) can’t be

R is also **object-oriented** meaning (after your initial introduction) it is much easier to use and extend (more on this later)
R can do pretty much all statistical analysis:

The basic tests

- Pearson’s Correlation Coefficient
- Independent and paired t-test
- $\chi^2$ test of independence
- etc etc....

Classical tests vs ’modeling’

Classical tests a little cumbersome. R’s real strength is where unifying theory can be used to link methods (i.e. the linear models)
Intermediate and advanced methods
The linear models

- General linear models (ANOVA, linear regression)
- Generalized linear models (Logistic regression, Poisson Regression etc)
- Cox proportional hazard regression (Survival analysis)
- Mixed and marginal models for correlated data (Linear mixed models, Generalized linear mixed models and GEEs)

The linear models are R’s real strength because of the approach R

Multilevel data and the ”Mixed models”:
You will be experts in the last groups of models after this WS series (the Mixed models)
Intermediate and advanced methods
Other methods

- Generalized additive models (good for epidemiological studies of ecological datasets)
- Multivariate methods (Factor analysis, Principal component analysis, Partial least squares, clustering and many more)
- Time series analysis methods
- Specialized analyses (e.g. Genetic/genomic studies)
Graphics

- R also produces excellent publication quality graphics
- It provides a wide range of graphs; and
- (once your R coding is good enough) all statistical method and graphs can be easily extended or enhanced
R can be downloaded from
http://cran.r-project.org/

This web site provides both the base and contributed (written by others) packages.

It is available on Windows, Linux/Unix and Mac platforms
Opening R

R version 3.0.1 (2013-05-16) -- "Good Sport"
Copyright (C) 2013 The R Foundation for Statistical Computing
Platform: i386-w64-mingw32/i386 (32-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY. You are welcome to redistribute it under certain conditions. Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors. Type 'contributors()' for more information and 'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or 'help.start()' for an HTML browser interface to help. Type 'q()' to quit R.

[Previously saved workspace restored]

>
We can use R (interactively) like a calculator

> x <- 1.3
> y <- 2.5
> x+y

[1] 3.8
Interactive or script file?

Using R interactively (typing in individual lines) is generally not the best way to use it. As you get better you will want to start collecting your syntax into files. R comes with a (VERY) basic script file editor, but there are a few much better ones (freely) available. The ones I have used are:

- tinn-R (version 1.17.2.4 later versions are unstable)
- notepad++
- R-studio (*Recommended*)

**Hint: Downloading R studio**

R studio can be downloaded from [www.rstudio.com/ide/](http://www.rstudio.com/ide/).

Also cross-platform (Windows, Linux and Mac versions available)
Opening R

Main advantage of R studio: Keeps everything on the same screen

Type your syntax here
Run your syntax here
Data in memory
Help, Graphics, etc...
R works on objects. These include:

- single number or character
- vector or 1-D array of numbers, characters, logical values, or factors
- matrices (table of values of a single type)
- data frames (table allowed to contain variables of different types) = same as SAS, Stata or SPSS dataset
- lists (data frames as a special case of a list)
- functions and user defined objects

**Hint:**
It is a good idea to include 'type' in object name e.g. `mydata.df` and `mydata.mat`
Naming objects

Object names tend to use letters, numbers and periods.

Values (or functions) are assigned to an object using the `<−` operator. For example: `my.val <- 7` stores the value of 7 into the object `my.val`.

Aside:

The character `=` can also be used to assign values to an object, but `<−` is much more widely used.

Warning/Pitfall:

- Avoid underscores in names. While technically legal, it can cause problems.
- Note that the operator for "equals to" is `==`, not `=`.
The most commonly used data structure you will use in R is the **data frame**

The data frame represents a matrix-like object where the columns represent variables and the rows observations (but unlike matrices, the columns can be of different data types)

Rarely will we manually enter the data directly into R, it is much more common that we will read it in through some data file.

**Important: Data frames**

Unlike matrices, data frames allow columns to be of different data types. E.g. Column 1 could represent **sex** (Categorical: M, F) and column 2 **age** (Continuous: *any value between 15 and 75*) etc.
Consider a simple dataset containing 200 patients on which the following variables are measured:

1. id (Patient ID)
2. age
3. sbp (Systolic blood pressure)
4. dbp (Diastolic blood pressure)
5. chol (LDL-cholesterol)
6. ses (Socio-economic status \(<\) education and income)
7. bmi (Body mass index)

These data are currently stored in a comma delimited text (csv) file. Note that a few other variables (e.g. categorizations of BMI) have been included for your convenience.
Cholesterol data in csv file

Motivating example
To read in the csv file and dump it to a dataframe:

```r
> mychol.df<-read.csv("E:/Introduction to R/Basics/Cholesterol.csv")
```

**Warning/Pitfall:**

Note the use of the forward slash "/" rather than the traditional backslash. Also a double back slash (\\) can be used.

Alternatively we can set up a working directory first:

**R SYNTAX: Reading in a comma delimited text "csv" file**

```r
setwd("D:/IntroR/Basics")
mychol.df<-read.csv("Cholesterol.csv")
```
We can also read data in SPSS (.sav), Stata (.dta) data file (and other stats packages) using the `foreign` R library. E.G. To read in the same data above from a Stata data file:

**R SYNTAX: Reading in a Stata datafile**

```r
library(foreign)
setwd("D:/IntroR/Basics")
mycholdta.df<-read.dta("Cholesterol.dta")
```

**Warning/Pitfall: Data from other (foreign) stats packages**

Avoid using this approach. OK for simple datasets, but can be unstable for other types of data (e.g. data formatted for survival analysis). Also can be problems with SPSS’s value labels. Best to export data from your stats package into a comma delimited (.csv) file, and then read into R
Most of the analysis you will perform when you start with R will use methods available in the libraries that come with the base distribution of R.

For example, most statistical analysis will be from the libraries stats or MASS, and most graphics will employ the graphics library.

These standard libraries automatically load into memory when we start up R.
As you use more specialized and advanced analysis you will have to load a library into memory before you can call its methods. We do this with the `library()` statement (usually at the start of our R file).

Some libraries (e.g. `foreign`) are provided with the base distribution of R, but still need to be loaded into memory (e.g. `library(foreign)`).

Other libraries need to be downloaded from repositories (e.g. CRAN) first.

For example, if you wanted to analyse data arising from a multilevel study, you might use Linear Mixed Models from the library `lme4`. In this case you would have to download this library from CRAN and then start your script file with the command: `library(lme4)`.
This brings us to two distinct steps (for libraries not automatically loaded into memories), **INSTALLING** and **LOADING** libraries...

1. **Installing libraries**: Only has to be done once. Here we will go to the R repository (called CRAN) and download a library we need (Can do this from inside R or R studio)

2. **Loading a library**: Needs to be done at the start of each session. This loads the library into memory, giving R access to all the libraries functions

**Hint: Loading libraries**

You don’t have to, but it is a good idea to start your R script (program) file with all the libraries you will need

**Why do you think that R doesn’t just automatically load ALL the libraries into memory when you start R?**
Viewing data in data frames and matrices

Many ways to view data in R, but the easiest is to just use the `View(mydata.df)` function. For example, to view the cholesterol data:

```r
View(mychol.df)
```

This will open a spreadsheet showing you the data.

**Hint: Viewing the dataset**

Viewing the data is very easy in R studio, just click the data frame in "Workspace" (Top right hand window).
Indexing

One of the best features in R is how object indices can be used to subset the data. The first thing we have to note is that R for data tables (e.g. Matrices or data frame) uses the convention:

```
mydata.df[ROW, COLUMN] format
```

If we leave out the ROW (type nothing) it will give us ALL rows, and if we leave out COL, we get ALL columns. For example:

```
mydata.df[,c(1,3,5)]
```

would give me ALL rows, and only columns 1, 3 and 5

**Hint: The c() function**

The `c()` function in R just means combine. So `c(1,3,4)` will just give us a vector contain 1, 3 and 5

**WHY IS THIS INDEXING CONVENTION SO USEFUL??**
Documenting your code: The importance of comments

If you want to leave yourself "notes" about your syntax use #

R SYNTAX: Reading in a data (with comments)

# I can write anything here blah blah

# Set my working directory
setwd("E:/Introduction to R/Basics")

# Read in comma delimited chol data
mychol.df<-read.csv("Cholesterol.csv")

Hint: Documenting your code

Documenting your code has two MAJOR advantages:

1. You (and other people) can understand what you have done
2. If you do it well, it IS your methods section
• R is very strong regarding plots (with hundreds available from the various libraries)
• Let’s start with a basic histogram using the cholesterol data

R syntax: A (very) basic histogram

```r
# Histogram of cholesterol
hist(mychol.df$chol)
```
A very basic plot

Histogram of mychol.df$chol

Frequency
A bit better histogram

Now let’s put some more features in the graph

R syntax: Histogram

```r
hist(mychol.df$chol, main="My first histogram",
xlab="Cholesterol", c="Red")
```
To generate a simple boxplot..

**R syntax: Boxplots**

```r
# Generating a boxplot
boxplot(mychol.df$chol, main="Box plot of cholesterol levels", ylab="Cholesterol level")
```

![Box plot of cholesterol levels](image)

---

To generate a simple boxplot, you can use the `boxplot` function in R. The code snippet above demonstrates how to create a boxplot for cholesterol levels. The `mychol.df$chol` refers to the cholesterol levels column in the dataset. The `main` parameter sets the title of the plot, and `ylab` sets the y-axis label. The resulting boxplot provides a visual summary of the cholesterol distribution, including quartiles, median, and outliers.
Summarizing categorical data

Univariate

Frequency table for a categorical variable:

R syntax: Frequency tables

table(mychol.df$bmi.class)

Gives:

<table>
<thead>
<tr>
<th>Category</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Underweight</td>
<td>2</td>
</tr>
<tr>
<td>Normal</td>
<td>93</td>
</tr>
<tr>
<td>Overweight/obese</td>
<td>105</td>
</tr>
</tbody>
</table>
For **Bivariate relationships** it is important to note (if relevant) which is the **outcome variable** and which is the **explanatory variable**

**Not really a simple graph for this situation. If we collapse the continuous variables into categories we can use cross-tabulation**
Bivariate plots: Side-by-side boxplots

Here, I collapse BMI into classes (underweight, normal, overweight/obese)

R syntax: Side-by-side boxplots

```r
boxplot(chol~bmi.class, data=mychol.df, main="Cholesterol by BMI class", ylab="Cholesterol level", xlab="BMI class")
```

Note the use of:

`chol~bmi.class`

This is a very important feature in R. It is the formula format and you will see much more of it.
Scatter plot

R syntax: Scatter plots

```r
# Basic scatter plot
plot(x=age, y=chol, data=mychol.df)

# now with more features
plot(x=age, y=chol, data=mychol.df, main="My first scatterplot",
     sub="Cholestrol vs Age", pch=2, col="Blue", xlab="Age",
     ylab="Cholestrol")
```

Hint:
- R doesn’t care what order you specify function parameters
- if you don’t specify a parameter, R will use the default
- `pch` stands for plot character
- By using `data=mychol.df` I don’t have to prefix variable names with `chol.df$`
Scatter plot

The previous R code chunk gives .....
Summarizing categorical data

Bivariate

Now to generate a cross-tabulation:

R syntax: Cross tabulation

```r
# generate cross-tabulation
table(mychol.df$bmi.class, mychol.df$ses.class)
```

Gives:

<table>
<thead>
<tr>
<th>BMI class</th>
<th>low</th>
<th>low-mid</th>
<th>mid</th>
<th>high-mid</th>
<th>high</th>
</tr>
</thead>
<tbody>
<tr>
<td>Underweight</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Normal</td>
<td>10</td>
<td>15</td>
<td>55</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>Overweight/obese</td>
<td>12</td>
<td>19</td>
<td>47</td>
<td>13</td>
<td>14</td>
</tr>
</tbody>
</table>
Modeling in R

I won’t bother going through classical bivariate tests (t-tests, $\chi^2$ test of independant etc) mainly because:

- As non-model based methods there is not the consistent ‘thread’ running through these tests
- Also, many similar analyses can be conducted using bivariate models anyway
- In contrast to the classical tests, modeling in R is very intuative. Once you understand it for one method (e.g. Linear regression), it is very easy to extend to other models

I will (very) briefly cover two models:

1. Linear Regression (and the General Linear Model)
2. Generalized Linear Models (focusing on binary logistic regression)
Review: Linear regression and the general model

Let’s start by reviewing the linear regression model:

\[ Y_i = \beta_0 + \beta_1 X_{i,1} + \beta_2 X_{i,2} + \ldots + \beta_{k-1} X_{i,k-1} + \epsilon_i \]

Alternatively, we can use matrix notation to represent this model:

\[ \mathbf{y} = \mathbf{X}\beta + \epsilon \]

Where
- \( \mathbf{Y} \) is vector of observation of our outcome variable;
- \( \mathbf{X} \) is a matrix containing the explanatory variable(s); and
- \( \beta \) is a vector of parameters relating \( \mathbf{X} \) to \( \mathbf{Y} \)

The General linear model

The Linear regression model is often called a General (or Normal) Linear Model when at least one of the explanatory variables is a dummy variable (or set thereof) representing a categorical predictor.
To fit a general linear model (or linear regression) in R:

```R
# Fit a general linear model
my.lm<-lm(y ~ x, data=mydata.df)
```

To get the individual coefficients, overall model significance (and $R^2$), or to test the significance of multicategory terms:

```R
# Coefficients and overall model significance
summary(my.lm)

# Global tests for multiclass Xs
anova(my.lm)
```
DEFINITION: A **Generalized Linear Model** is a model that can be represented:

\[ g(y) = \mathbf{X}\beta + \epsilon \]

Or equivalently,

\[ y = g^{-1}(\mathbf{X}\beta + \epsilon) \]

where \( g() \) is some link function that is both **monotonic** and **differentiable**
What are the main differences between the General Linear Model (Linear regression) and the Generalized Linear Model?

- The Generalized linear model can be used to model outcomes from distributions other than the normal distribution (Binomial, Multinomial, Poission etc)
- These distributions are commonly observed in health studies
- GLMs do this via a **link function**: A function that 'linearizes' the relationship between Y (LHS) and the Xs (RHS)
- $\beta$s in GLMs are estimated via Maximum Likelihood Estimation (MLE) as opposed to Least Squares (like Linear regression)
General Linear Models: Binary Logistic Regression

- Classical model was developed for binary outcomes
- Involves modelling the OR (odds ratio)
- Uses the logit link, which is simply the log of the odds ratio. That is:

\[ \ln(OR) = X\beta + \epsilon \]

Or,

\[ \ln \left( \frac{Odds_{D+}}{Odds_{D-}} \right) = X\beta + \epsilon \]
(As promised) running Logistic regression (and any GLM) in R is a simple extension to the General Linear Model we saw last week.

General Linear Model

```r
my.linreg <- lm(my.y ~ x1 + x2, data = mydata.df)
```

To run a logistic regression model:

Logistic regression (and any other Generalized Linear Model)

```r
my.logreg <- glm(my.y ~ x1 + x2, data = mydata.df, family = binomial())
```

Note: For the GENERAL linear model, `my.y` is assumed to be continuous; and for Logistic regression, `my.y` must be binary (0,1)
Logistic regression in R: Summarizing the model

**EXACTLY** like linear regression, we can get model details...

### R syntax: Model details

```r
# Coefficients and overall model significance
summary(my.logreg)

# Global tests for multiclass Xs
anova(my.logreg)
```

Often we want the Odds ratio, not $\beta$s, from a Logistic regression ($\text{rem } OR = e^{\beta}$), to get the ORs and their 95% CIs:

### R syntax: ORs and their 95% CIs

```r
# ORs and CIs
print.ORCIs(my.logreg)
```

**Note:** `print.ORCIs()` is not a standard in R (I wrote it). I provide this function (with others) in your R material.
Saving an R session (workspace)

- After reading in data, generating graphs and analyses, we can save all of this "R session" into a 'workspace'.
- Allows us to start of where we left off next time we start R.
- This information is stored in an `.RData` file, and generally you would have one for each project you are doing. E.G.
  1. Cholesterol.RData for my cholesterol analysis
  2. T2DM.RData for my Type 2 diabetes study
  3. Sleep.RData for my analysis of sleep data

R workspaces (.RData files)

Workspaces (.RData files) allow us to save data, graphs, analyses etc; a major advantages of R’s object oriented approach
Saving and loading an R workspace (.RData files)

To save an R workspace:

**Saving a session**

```r
# Make sure you have set your working directory
save.image("Blah.RData")
```

To open a session that you have previously saved....

**Loading a (previously saved) session**

```r
load("Blah.RData")
```

Alternatively, you can just save and load the sessions in the 'workspace' pane in RStudio (top right-hand window)
R can be quite tricky when you first begin

One of the ways to make this much easier is to understand and use the help

I will talk about three main aspects of the R help system:

1. **free introductory text**
2. **searching help**
3. **HTML package help**

I will also discuss resources on the internet that make it MUCH easier
Free introductory text

R is free software and comes with no warranty. You are welcome to redistribute it, Type 'license()' or 'licence()' for full details.

R is a collaborative project with many contributors. Type 'contributors()' for more information and 'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or 'help.start()' for an HTML browser interface to help.

Type 'q()' to quit R.

[Previously saved workspace restored]

>
Searching help

Searching the help will identify methods associated with a key word. Three ways of using:

1. \textit{Help} -> \textit{Search help}....->\textit{type in keyword}

2. \textit{type in help.search("t test")} This will identify libraries with this (or similarly named) functions
Finally, if we know the function name (e.g. `t.test`) and want help in its properties use:

- `help("myfunction")` or `?("myfunction")`
  
  *For example:* `help("t.test")`

- If we know what package the function is, we can go to: `Help -> HTML help -> packages`, and then we would choose the appropriate package.
The help I most commonly use is the HTML help pages
Go to Help → html help, and then hit the 'packages’ hyperlink
This lists all of the packages (and their functions and datasets)

In R studio, it is even easier to get to these pages. Go to the help window (bottom right window) and hit the "help" pane
R resources on the internet

- Being open source, many people have put a lot of resources on the internet
- Also a lot of wikibooks and free textbooks available on the web (e.g. "The R book" is about 1300 pages)
- One of the best websites for beginners is the Quick R site: [www.statmethods.net](http://www.statmethods.net)
- There are also HEAPS of youtube walk-thrus. There is a really good set listed on "R function of the day" site [rfunction.com](http://rfunction.com). This provides about 20 5-minute you tube walk-thrus which will get you well on the way to on the way
- Alternatively, just type "R for beginners" or "Introduction to R" in youtube’s search engine.
- Help pages (more advanced users). I have rarely come across a problem in R someone else hasn’t come across (and solved). I usually just google my problem.
R is a comprehensive and comparatively ‘low-level’ language
For those of you not used to coding/programming, mastering R will take a little while
However, it is free (open source), extendible, and is one of the strongest and most versatile statistical packages available
For this reason it is the package of choice among many statisticians (few people look back after converting over to R)
Once you have put this work in, you won’t ever look back either

Number one hint for learning R
USE IT!!!!!!!
One tricky aspect of learning R (especially the basics), is that there often many ways of doing the same thing.

A good example of this is reading in data. If you look at 10 textbooks, you will get 11 ways of reading in a datafile.

I use the `read.csv()` method because it works well for me (and I don’t have to remember lots of parameters), but you might find something different works better for you.

My hint: Stick with methods that work for you and (to start with) go back to your old syntax files and copy and paste.
Thank-you!!!!!!

QUESTIONS???