

Introduction to R: Basics

Dr Cameron Hurst
cphurst@gmail.com

CEU and DAMASAC, Khon Kaen University

1st September, 2557



What we will cover (Workshop series)

Five sessions:

- ① **Day 1 Session 1: Intro to R basics**
Data I/O, Basic data structures and graphics, intro to modeling in R
- ② **Day 1 Session 2:** Linear regression
- ③ **Day 2 Session 1:** ANOVA and The General Linear Model
- ④ **Day 2 Session 2:** The Generalized Linear Model and Logistic regression
- ⑤ **Day 3 Session 1:** Hands on exercises: Modelling and model selection in R
- ⑥ **Day 3 Session 2:** Survival Analysis

Conventions

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.

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

About R.....

- R is freely available open-source (\Rightarrow 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 R's functionality (more on this later)

What can R do?

R can do pretty much all statistical analysis:

The basic tests

- Pearson's Correlation Coefficient
- Independent and paired t-test
- χ^2 test of independence
- etc etc....

Classical tests vs 'modeling'

Classical tests a little cumbersome. R's real strength is in modeling where unifying theory allows the methods to be easily linked (linear model theory)

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

Longitudinal 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)
- ...and MUCH, MUCH more

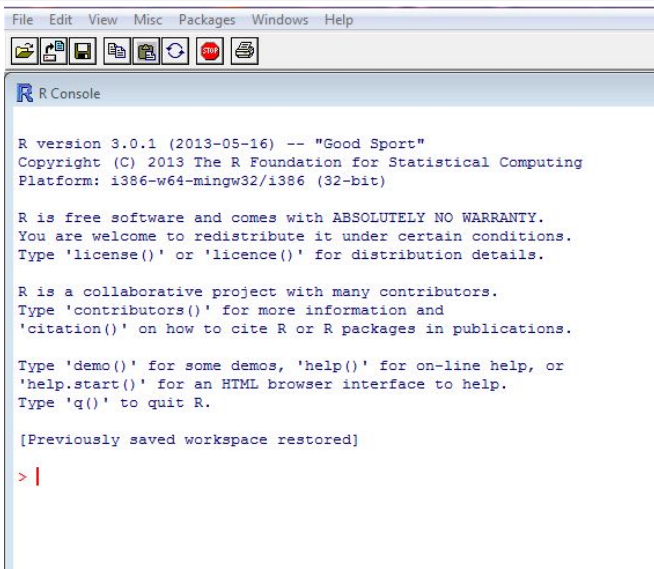
Graphics

- 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

Where can I get R?

- 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



The screenshot shows the R Console window with a menu bar (File, Edit, View, Misc, Packages, Windows, Help) and a toolbar with icons for file operations and execution. The console text is as follows:

```
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 ("R") 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:

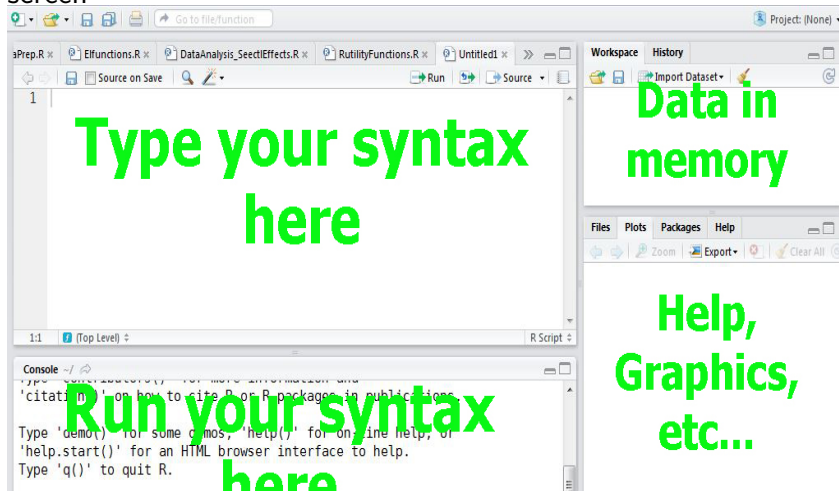
- tinn-R (version 1.17.2.4 later versions are unstable)
- notepad++
- R-studio (**Recommended**)

Hint: Downloading R studio

R studio can be downloaded from www.rstudio.com/ide/. Also cross-platform (Windows, Linux and Mac versions available)

Opening R

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



Objects

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 (advanced: data frames are 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` (for a data frame and matrix, respectively)

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.
- ▶ Note that the operator for "equals to" is `==`, not `=`

Data input and data frames

- 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.

Data input and data frames

Motivating example

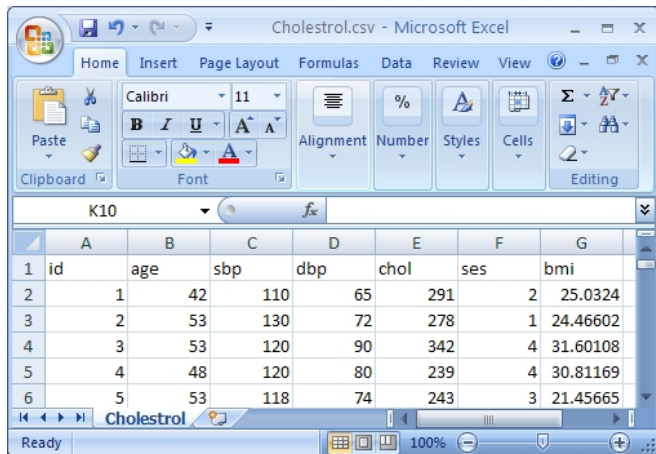
Consider a simple dataset containing 200 patients on which the following variables are measured:

- 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



Cholestrol.csv - Microsoft Excel

Home Insert Page Layout Formulas Data Review View

Paste Clipboard Font Alignment Number Styles Cells Editing

K10 fx

	A	B	C	D	E	F	G
1	id	age	sbp	dbp	chol	ses	bmi
2	1	42	110	65	291	2	25.0324
3	2	53	130	72	278	1	24.46602
4	3	53	120	90	342	4	31.60108
5	4	48	120	80	239	4	30.81169
6	5	53	118	74	243	3	21.45665

Cholestrol

Ready 100%

Cholesterol data in csv file

Motivating example

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

```
> 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

```
setwd("D:/IntroR/Basics")  
mychol.df<-read.csv("Cholesterol.csv")
```

Reading in data from other stats packages

We can also read data from SPSS (.sav), Stata (.dta) and other stats packages data files 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

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

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

Libraries

- 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

Libraries

- 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 longitudinal 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:

Installing vs Loading libraries

This brings us to the 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

It's a good idea to load libraries at the start of your R script file

Why doesn't R just automatically load ALL the libraries when starting?

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:

```
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. You should note that R for data tables (e.g. Matrices or data frames) uses the convention:

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

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

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

would give me ALL rows (i.e. no filtering), 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

The importance of documenting your code

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 it (later)
- 2 If you do it well, it IS most of your methods section

Generating plots

- 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

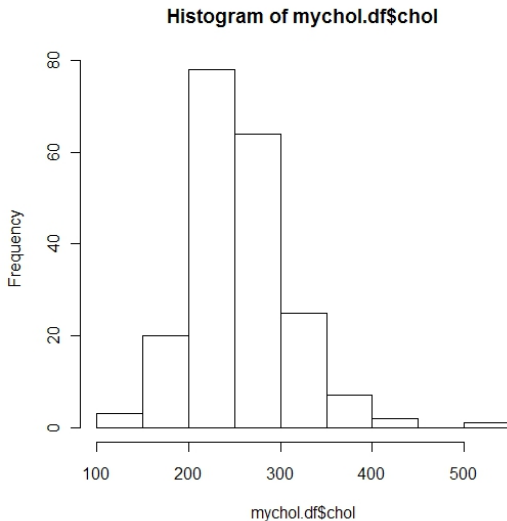
```
# Histogram of cholesterol  
hist(mychol.df$chol)
```

Hint: The \$ operator

Think of the \$ as a 's in English. For example:

Cameron\$car would be "Cameron's car"; or
mychol.df\$chol would be the data frame mychol.df's
variable called chol (LDL cholesterol)

A very basic plot

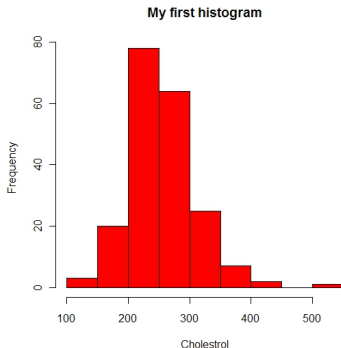


A bit better histogram

Now let's put some more features in the graph

R syntax: Histogram

```
hist(mychol.dfschol, main="My first histogram",  
     xlab="Cholesterol", c="Red")
```

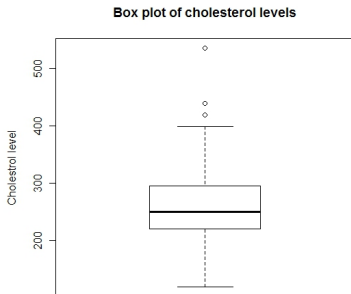


A simple boxplot

To generate a simple boxplot..

R syntax: Boxplots

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



Summarizing categorical data

Univariate

Frequency table for a categorical variable:

R syntax: Frequency tables

```
table(mychol.df$bmi.class)
```

Gives:

Underweight	Normal	Overweight/obese
2	93	105

Data type vs. Data representation

Bivariate relationships

For **Bivariate relationships** it is important to note (if relevant) which is the **outcome variable** and which is the **explanatory variable**

		Explanatory	
		Continuous	Categorical
Outcome	Continuous	Scatter plot	Side-by-side boxplot
	Categorical	**	Cross-tabulation

*** 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

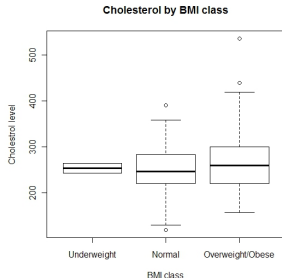
R syntax: Side-by-side boxplots

```
boxplot(chol~bmi.class, data=mychol.df, main="Cholesterol by  
BMI class", ylab="Cholestrol 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.



Q: Why didn't I have to use `mychol.df$`???

Scatter plot

R syntax: Scatter plots

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

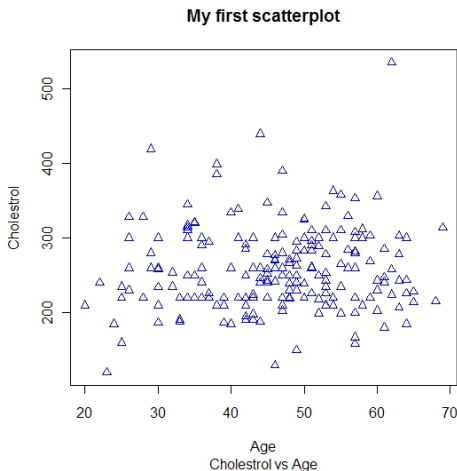
# now with more features
plot(x=age, y=chol, data=mychol.df, main="My first scatterplot",
     sub="Cholestrol vs Age", pch=2, col="Blue", xlab="Age",
     ylab="Cholestrol")
```

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

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

Gives:

BMI class	low	low-mid	mid	high-mid	high
Underweight	0	1	1	0	
Normal	10	15	55	5	
Overweight/obese	12	19	47	13	

Modeling in R

I won't bother going through classical bivariate tests (t-tests, χ^2 test of independant etc) because:

- For non-model based methods there is notheoretical 'thread' running through these tests
- Also, similar (or even identical) analyses can be conducted using bivariate models anyway
- In contrast to the classical tests, modeling in R is very intuitive. 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:

$$Y_i = \beta_0 + \beta_1 X_{i,1} + \beta_2 X_{i,2} + \dots + \beta_{k-1} X_{i,k-1} + \epsilon_i$$

Alternatively, we can use matrix notation to represent this:

$$\mathbf{y} = \mathbf{X}\beta + \epsilon$$

Where

Y is vector of observation of our outcome variable;

X is a matrix containing the explanatory variable(s); and

β is a vector of parameters relating X to 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

Modeling in R: The general linear model

To fit a general linear model (or linear regression) in R:

R syntax: The general linear model

```
# 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 syntax: Model details

```
# Coefficients and overall model  
significance  
summary(my.lm)  
  
# Global tests for multiclass Xs  
anova(my.lm)
```


Review: The Generalized Linear Model

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**

Review: The Generalized Linear Model

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, Poisson etc)
- These distributions are commonly observed in health studies
- GLMs do this is via a **link function**: A function that 'linearizes' the relationship between Y (LHS) and the Xs (RHS)
- β 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:

$$\ln(OR) = \mathbf{X}\beta + \epsilon$$

Or,

$$\ln \left(\frac{Odds_{D+}}{Odds_{D-}} \right) = \mathbf{X}\beta + \epsilon$$

Logistic Regression in R

(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

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

To run a logistic regression model:

Logistic regression (and any other Generalized Linear Model)

```
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: Sumamrizing the model

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

R syntax: Model details

```
# Coefficients and overall model  
significance  
summary(my.logreg)  
  
# Global tests for multiclass Xs  
anova(my.logreg)
```

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

R syntax: ORs and their 95%CIs

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

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.

Saving and loading an R workspace (.RData files)

To save an R workspace:

Saving a session

```
# 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

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

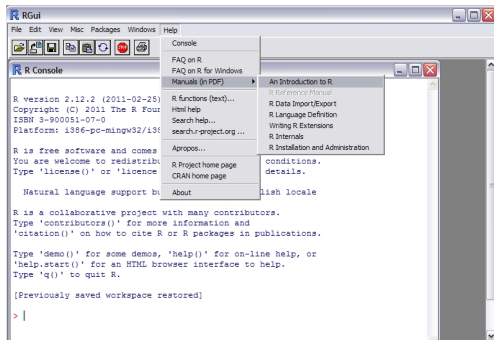
Alternatively, you can just save and load the sessions in the 'workspace' pane in RStudio (top right-hand window)

R for beginners

- 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



Searching help

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

- 1 *Help -> Search help....->type in keyword*
- 2 type in `help.search("t test")` This will identify libraries with this (or similarly named) functions

Searching help

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

HTML help

- 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. We will practice this in the Hands-on session.

R resources on the internet

- Being open source, many 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`
- There are HEAPS of youtube walk-thrus. There is a really good set listed on "R function of the day" site `rfunction.com`. This provides about 20 5-minute youtube walk-thrus which will get you well 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.

A final word...

- 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 **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!!!!!!!

What I haven't covered

- 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

Any questions??????

Thank-you!!!!!!
QUESTIONS???