

# Introduction to



# What is



Lang  
uage  
Platfo  
rm

- A statistics programming language
- A data visualization tool
- Open source

Com  
munit  
y

- 2.5+M users
- Taught in most universities
- New and recent grad's use it
- Thriving user groups worldwide

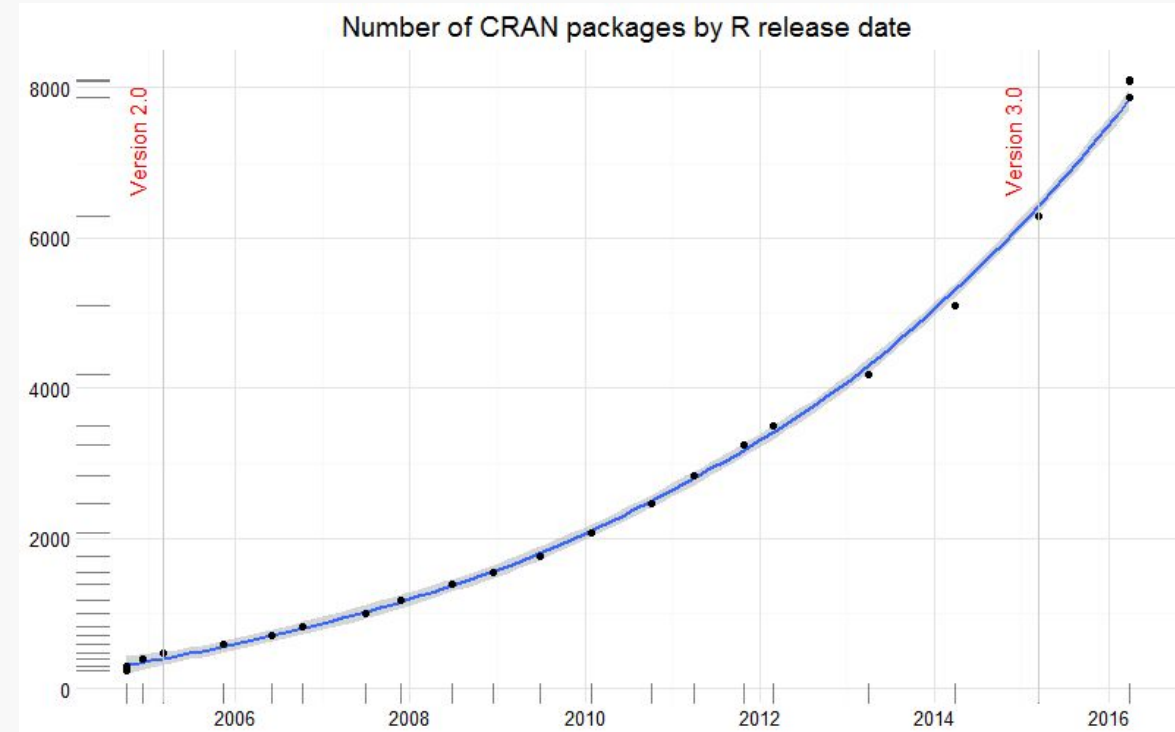
Ecos  
ystem

- 8000+ free algorithms in CRAN
- Scalable to big data
- Rich application & platform integration

# Power of R: R Language + Packages



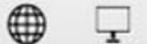





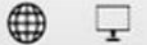

- R is an open source ([GNU](#)) version of the S language developed by John Chambers *et al.* at Bell Labs in 80's [History of R](#)
- R was initially written in early 1990's by [Robert Gentleman](#) and [Ross Ihaka](#) then with the Statistics Department of the University of Auckland
- R is administered and controlled by the [R Foundation](#)
- Microsoft is founding member and Platinum Sponsor of [R Consortium](#)

[R Reference Card from CRAN](#)



3000 packages added in last 2 years

# R Strengths and Adoption

		2016	2015	2014
Language Rank	Types	Spectrum Ranking	Spectrum Ranking	Spectrum Ranking
1. C		100.0	100.0	100.0
2. Java		98.1	99.9	99.3
3. Python		98.0	99.4	95.5
4. C++		95.9	96.5	93.5
5. R		87.9 <b>2016</b>	81.3	92.4
6. C#		86.7	84.8 <b>2015</b>	84.8
7. PHP		82.8	84.5	84.5
8. JavaScript		82.2	83.0	78.9
9. Ruby		74.5	76.2	74.3 <b>2014</b>
10. Go		71.9	72.4	72.8

Source: IEEE Spectrum July 2014, 2015 & 2016

- Comprehensive set of statistical analysis techniques
  - Classical statistical tests
  - Linear and nonlinear modeling
  - Time-series analysis
  - Classification and cluster analysis
  - Spatial statistics
  - Bayesian statistics
- Virtually every statistical technique is either already built into R, or available as a free package
- Completely open-source
  - Users contribute and create new packages
  - Existing R functions can be edited and expanded
  - Free
  - Huge community of scientists using R
- Publication-quality graphics
  - Many default graphics
  - Full control of graphics
  - Make even rudimentary plots vibrant and exciting
- Allows start-to-end reproducibility of your research
  - Read in data
  - Exploration of patterns in complex data
  - Apply statistical tests and fit models
  - Produce summary statistics and tables
  - Create final figures, all in a single script of R
  - If your data change, model has to be redone, or reviewers ask for revisions, it is easy to revise and rerun



# Power of R: R Language + Packages

## CRAN: 8000+ Add-on packages for R

### CRAN Task Views

CRAN Task Views are guides to the packages and functions useful for certain disciplines and methodologies. Many long-term R users I know have no idea they exist. As an effort to make them more widely known I thought I'd jazz up the index page. Images are free to use, and got from [iStock](#) stock photo site. Visual puns are mine. Task View links go to the cran.r-project.org site and not a mirror.



#### Bayesian Inference

Applied researchers interested in Bayesian statistics are increasingly attracted to R because of the ease of which one can code algorithms to sample. [\[more\]](#)



#### Chemometrics and Computational Physics

Chemometrics and computational physics are concerned with the analysis of data arising in chemistry and physics experiments, as well as the simulation of. [\[more\]](#)



#### Clinical Trial Design, Monitoring, and Analysis

This task view gathers information on specific R packages for design, monitoring and analysis of data from clinical trials. It focuses on including. [\[more\]](#)



#### Cluster Analysis & Finite Mixture Models

This CRAN Task View contains a list of packages that can be used for finding groups in data and modelling unobserved cross-sectional heterogeneity. Many... [\[more\]](#)



#### Probability Distributions

For most of the classical distributions, base R provides probability distribution functions (p), density functions (d), quantile functions (q), and. [\[more\]](#)



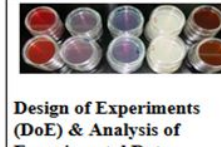
#### Computational Econometrics

Base R ships with a lot of functionality useful for computational econometrics, in particular in the stats package. This functionality is complemented by many... [\[more\]](#)



#### Analysis of Ecological and Environmental Data

This Task View contains information about using R to analyse ecological and environmental data. [\[more\]](#)



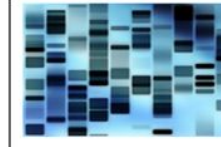
#### Design of Experiments (DoE) & Analysis of Experimental Data

This task view collects information on R packages for experimental design and analysis of data from experiments. Please feel free to suggest enhancements... [\[more\]](#)



#### Empirical Finance

This CRAN Task View contains a list of packages useful for empirical work in Finance, grouped by topic. [\[more\]](#)



#### Statistical Genetics

Great advances have been made in the field of genetic analysis over the last years. The availability of millions of single nucleotide polymorphisms (SNPs). [\[more\]](#)



#### Natural Language Processing

This CRAN task view contains a list of packages useful for natural language processing. [\[more\]](#)



#### Analysis of Pharmacokinetic Data

The primary goal of pharmacokinetic (PK) data analysis is to determine the relationship between the dosing regimen and the body's exposure to the drug as. [\[more\]](#)



#### Official Statistics & Survey Methodology

This CRAN task view contains a list of packages that includes methods typically used in official statistics and survey methodology. Many packages provide... [\[more\]](#)



#### Phylogenetics, Especially Comparative Methods

The history of life unfolds within a phylogenetic context. Comparative phylogenetic methods are statistical approaches for analyzing historical. [\[more\]](#)



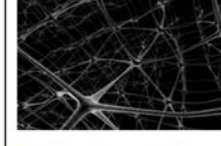
#### Multivariate Statistics

Base R contains most of the functionality for classical multivariate analysis, somewhere. There are a large number of packages on CRAN which extend this... [\[more\]](#)



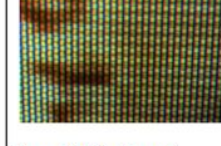
#### Optimization and Mathematical Programming

This CRAN task view contains a list of packages which offer facilities for solving optimization problems. Although every regression model in statistics. [\[more\]](#)



#### Machine Learning & Statistical Learning

Several add-on packages implement ideas and methods developed at the borderline between computer science and statistics - this field of research is usually. [\[more\]](#)



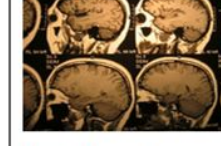
#### Graphic Displays & Dynamic Graphics & Graphic Devices & Visualization

R is rich with facilities for creating and developing interesting graphics. Base R contains functionality for many plot types including coplots, mosaic. [\[more\]](#)



#### High-Performance and Parallel Computing with R

This CRAN task view contains a list of packages, grouped by topic, that are useful for high-performance computing (HPC) with R. In this context, we are. [\[more\]](#)



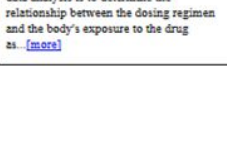
#### Medical Image Analysis

This task view is for input, output, and analysis of medical imaging files... [\[more\]](#)



#### Analysis of Spatial Data

Base R includes many functions that can be used for reading, visualising, and analysing spatial data. The focus in this view is on "geographical" spatial. [\[more\]](#)



#### Survival Analysis

Survival analysis, also called event history analysis in social science, or reliability analysis in engineering, deals with time until occurrence of an. [\[more\]](#)



#### Time Series Analysis

Base R ships with a lot of functionality useful for time series, in particular in the stats package. This is complemented by many packages on CRAN, which are... [\[more\]](#)



#### Robust Statistical Methods

Robust (or "resistant") methods for statistics modelling have been available in S from the start, in R in package stats (e.g., median(), mean(\*, trim = ). [\[more\]](#)



#### Statistics for the Social Sciences

Social scientists use a wide range of statistical methods. To make the burden carried by this task view lighter, I have suppressed detail in some areas that... [\[more\]](#)



#### gRaphical Models in R

Wikipedia defines a graphical model as a graph that represents independencies among random variables by a graph in which each node is a random variable, and. [\[more\]](#)



#### Reproducible Research

The goal of reproducible research is to tie specific instructions to data analysis and experimental data so that scholarship can be recreated, better. [\[more\]](#)



#### Psychometric Models and Methods

Psychometrics is concerned with the design and analysis of research and the measurement of human characteristics. Psychometricians have also worked... [\[more\]](#)

CRAN Task View by Barry Rowlingson: <http://www.maths.lancs.ac.uk/~rowlings/R/TaskViews/>

More packages on [Github](#) and [BioConductor project](#)

# R vs Python

## R Pros

- More mature data science support (20 years +), purpose built
- More established ML support

## Python Pros

- Best all round script language. Data science support improving.
- Better 64 bit support and scalability?

R performance and scalability – don't forget Revolution Analytics

# R Ecosystem – Essential Bits

Download latest R from CRAN – Comprehensive R Archive Network.  
<https://cran.r-project.org/> (Revolution Analytics R not covered)

Get R Studio. <https://www.rstudio.com/>

- **Essential** IDE. But much more, packages, RPubs etc..
- Download R, then R Studio.

RStudio a better environment for test and debug than Azure ML!

Get a Github account. <https://github.com/> and Github shell.

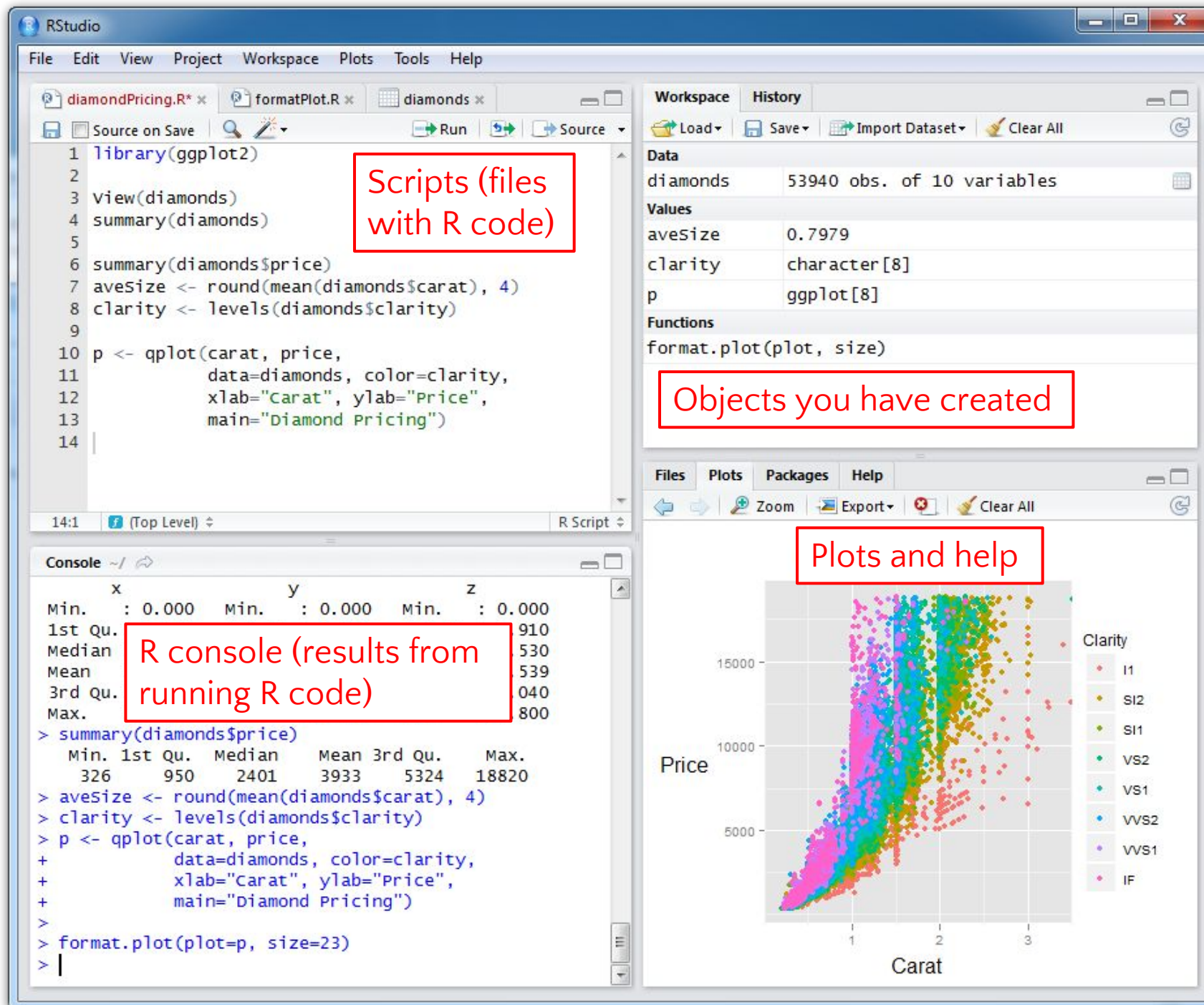
- Distributed source code control system.
- Essential part of R social network.

# RStudio

- It is tedious to write R code in the command line, and your code is not saved nor reproducible
- Much better: use RStudio. Why?
  - Multiple files
  - View variable values, color coding
  - Built-in help
  - Quick running of code
  - Easy file handling
  - Easy package installation
  - Many other reasons

There are other editors, e.g. Vim, Emacs, Sublime Text, TextPad... but Rstudio is easiest.





# R scripts

- A text file (e.g. lab1.r) that contains your R code for one complete analysis
- Scientific method: complete record of your analysis
- Reproducible: rerunning your code is easy for you or someone else
- Easily modified and rerun
- In RStudio, select code and type <ctrl+enter> to run the code in the R console
- **SAVE YOUR SCRIPTS**

# Commenting your code (do it)

- Use “comments” to document the intention of your code
- Anything on a line after # is ignored by R
  - `# Old Faithful geyser, Yellowstone NP`  
RStudio: different color for comments
  - `plot(faithful)`
- Rules of thumb
  - Document the purpose of the code not how it works
  - Use good variable names
  - Document for your future self: you will remember nothing about the code when you look at it next week or year

# R workspaces

- When you close RStudio, **SAVE YOUR .R SCRIPT**
- You can also save data and variables in an R workspace, but this is generally not recommended
  - Exception: working with an enormous dataset
- Better to start with a clean, empty workspace so that past analyses don't interfere with current analyses
- `rm(list = ls())` clears out your workspace
- Should be able to reproduce everything from your R script, so save your R script, don't save your workspace

# Some simple R commands

- `> 2+2`

- `[1] 4` Result of the command

- `> 3^2`

- `[1] 9`

- `> sqrt(25)`

- `[1] 5`

- `> 2*(1+1)`

- `[1] 4` Order of precedence

- `> 2*1+1`

- `[1] 3`

- `> exp(1)`

Optional argument

Incomplete command

# Assigning values

```
answer <- log(2.5)
```

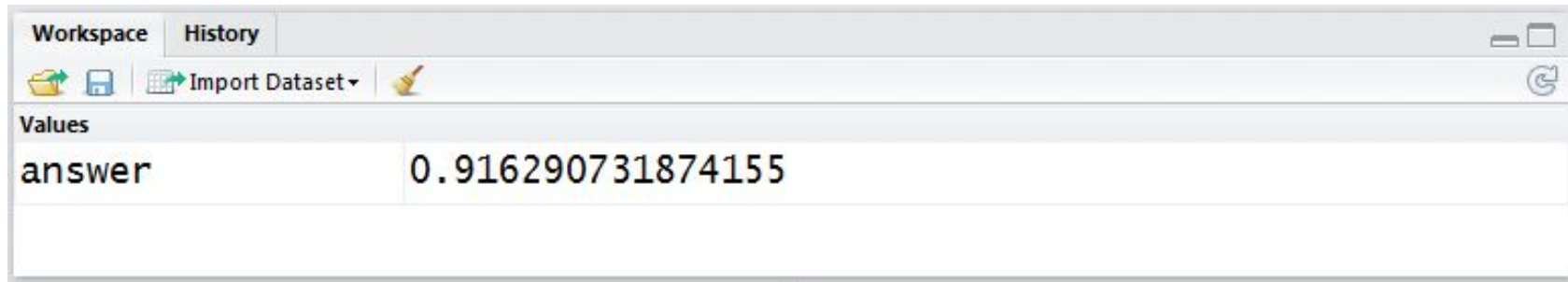
Assign the result of `log(2.5)` to a new object called “answer”

```
answer = log(2.5)
```

= can be used instead of <- but is frowned upon

```
answer <- log(2.5, base=10)
```

optional argument



When you run this command, an object “answer” is created in the workspace that is assigned the value of 0.91629... In RStudio, the top right window lists all the objects in the current workspace



# Assigning values

```
myName <- "Trevor"
```

Characters can also be assigned to objects

```
myName <- 'Trevor'
```

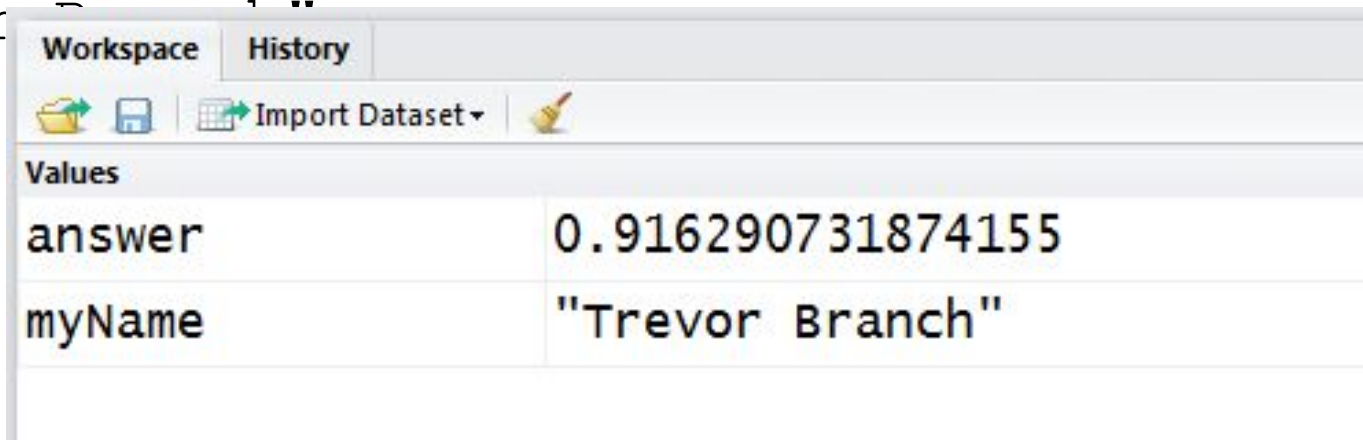
Usually we use double quotation marks but single quotes are treated the same





Surrounding a command in ( ) will display the assigned value in the R console

```
> (myName <- "Trevor Branch")
```

Can include spaces

```
[1] "Trevor Branch"
```



Workspace		History
   Import Dataset ▾ 		
Values		
answer	0.916290731874155	
myName	"Trevor Branch"	

# What is R doing?

```
> 2+4
```

```
[1] 6
```

The **[1]** means the first element of a vector

Even a single number in R is a vector, so "6" is a vector of size 1

```
> x <- 7
```

```
> x + 19
```

```
[1] 26
```

**<-** means "assign" in this case "assign the value 7 to the variable x"

Some use = for this purpose but it is frowned upon

Adding 19 to x gives the expected value of 26

```
> X + 10
```

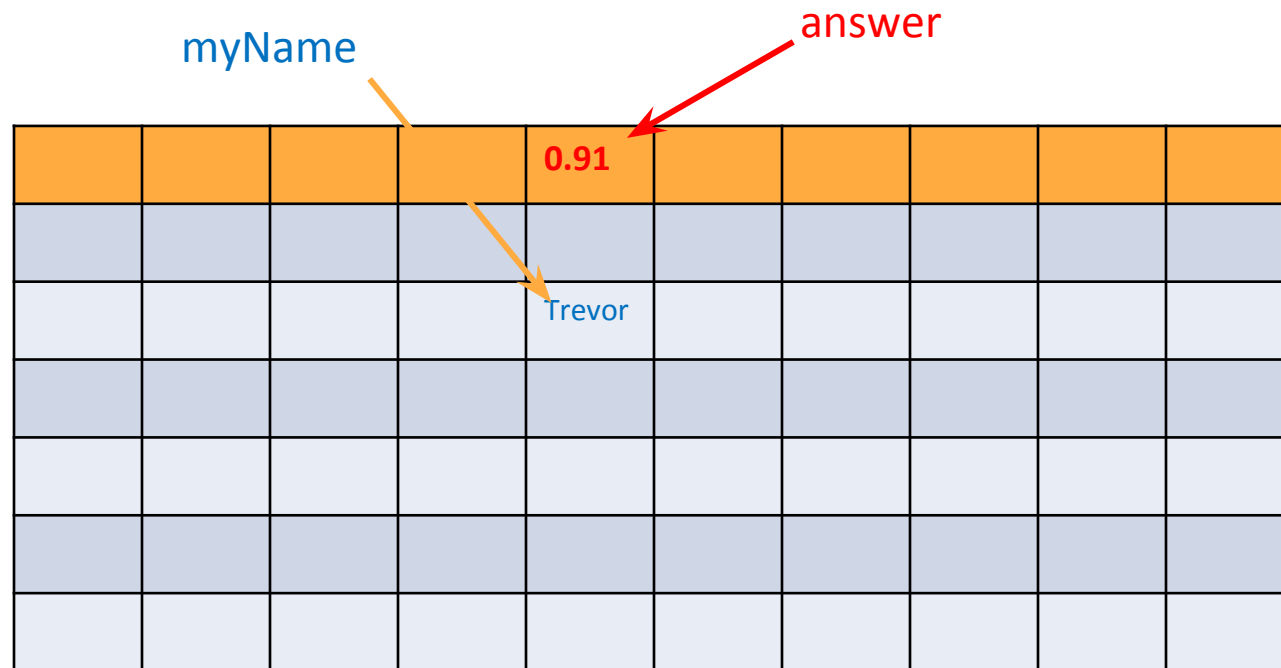
```
Error: object 'X' not found
```

X is not the same as x

R is **case sensitive**: upper case letters are different to lower case letters

# What is an object name?

- Each cell is part of the memory (RAM) of a computer
- Object names are human-convenient “pointers” that “point” at a memory location (e.g. 0x3A28213A) for the computer to look up



# Viewing objects

- RStudio: just look at the top-right Workspace tab
- Alternatively (and more generally):

```
> print(answer)
```

```
[1] 0.9162907
```

Very general command, works on all types of objects

```
> answer
```

```
[1] 0.9162907
```

Basic way of asking what is inside

```
> answer * 10
```

```
[1] 9.162907
```

Manipulate the value contained within the object

# Removing objects

- To find a list of all objects in the workspace

```
> ls()
```

```
[1] "answer" "myName"
```

- To remove an object

```
> rm(answer)
```

```
> ls()
```

```
[1] "myName"
```

- To remove **all** objects

```
> rm(list = ls())
```

```
> ls()
```

```
character(0)
```

Or Workspace/Clear All menu option in RStudio

Useful for clearing your workspace at the start of a new session

Nothing left in workspace

# Data types

- Data types describe how objects are stored in computer memory
- In R, you do **not** need to specify the data type
- Common data types (also known as **mode**) include
  - Numeric (integer, floating point numbers or doubles)
  - Logical (Boolean, true or false)
  - Characters (text or string data)
- The object type is not always obvious in R, and knowing what it is can be important



# R Data Types

Atomic data types.

- character
- numeric (real numbers)
- integer
- complex
- logical (True/False)

`typeof` function handy

```
> typeof(0.5)
[1] "double"
> typeof(1)
[1] "double"
> typeof("pizza")
[1] "character"
> typeof(1+4i)
[1] "complex"
> typeof(T)
[1] "logical"
> typeof(1:10)
[1] "integer"
```

# Finding data types

```
> answer <- log(2.718282)
```

```
> answer
```

```
[1] 1
```

```
> mode(answer)
```

Is it numeric or text?

```
[1] "numeric"
```

```
> is.numeric(answer)
```

Part of a family of `is.` functions

```
[1] TRUE
```

```
> typeof(answer)
```

Specifically, what type of object is it?

```
[1] "double"
```

```
> answer <- as.integer(answer)
```

```
> typeof(answer)
```

```
[1] "integer"
```

The `as.` functions **coerce** objects from one type to another

# Wait, what did you do there?

```
> answer <- 3.345452
```

```
> answer <- as.integer(answer)
```

```
> answer
```

Step 1

3.345452

**answer**



Step 2

3

`as.integer(answer)`  
Create a new object that is  
an integer

# Wait, what did you do there?

```
> answer <- 3.345452
```

```
> answer <- as.integer(answer)
```

```
> answer
```

Step 1

3.345452

Step 3

3

**answer**

Point the "answer" label at  
the new object



# Finding data types

- Similar functions can be applied to character variables; character and numeric storage modes will be commonly encountered in this class

```
> is.character(answer)
```

```
[1] FALSE
```

```
> is.character(myName)
```

```
[1] TRUE
```

```
> typeof(myName)
```

```
[1] "character"
```

# Vectors

A vector is a one-dimensional ordered collection of the same type of object

`c()` is a function that concatenates values together

```
> lengths <- c(7.8, 9.0, 7.1, 8.8, 8.8)
```

this is a vector of numbers

```
> lengths
```

```
[1] 7.8 9.0 7.1 8.8 8.8
```

the `:` function is used for consecutive numbers

```
1:10
```

`seq` function allows more flexibility

```
seq(from=1, to=10, by=2)
```

default order of parameters, no labels

```
seq(1,10,2)
```

vector of exactly five  
numbers between `from`  
and `to`

```
seq(from=1, to=10, length.out=5)
```



# Creating vectors using `rep`

```
> rep(3, times=10)
```

repeat 3 10 times

```
[1] 3 3 3 3 3 3 3 3 3 3
```

```
> y <- 1:3
```

y contains 1,2,3

```
> rep(y, times=4)
```

repeat y four times

```
[1] 1 2 3 1 2 3 1 2 3 1 2 3
```

Repeat y until there are 10 elements.

```
> rep(y, length=10)
```

The elements are **recycled**

```
[1] 1 2 3 1 2 3 1 2 3 1
```

```
> rep(y, length=30)
```

```
[1] 1 2 3 1 2 3 1 2 3 1 2 3 1
```

```
[14] 2 3 1 2 3 1 2 3 1 2 3 1 2
```

```
[27] 3 1 2 3
```

numbers [1], [14], [27] are **index** numbers of the first element on each line of the output

# Vector operations work element-wise

```
> (x <- 1:3)
```

```
[1] 1 2 3
```

```
> log(x)
```

```
[1] 0.0000000 0.6931472 1.0986123
```

```
> x+1
```

```
[1] 2 3 4
```

```
> x*2
```

```
[1] 2 4 6
```

# In-class exercise

- Create vectors using `seq()`, `rep()`, and mathematical operators. Only use `c()` when absolutely necessary.
  - Positive integers from 1 to 99
  - Odd integers between 1 and 99
  - The numbers 1,1,1, 2,2,2, 3,3,3
  - The numbers 1,2,3,4,5,4,3,2,1,0
  - The fractions 1, 1/2, 1/3, 1/4, ..., 1/10
  - The cubes 1, 8, 27, 64, 125, 216

# Using functions on vectors

- Many datasets are built into R for testing purposes, for a full list:

```
> library(help="datasets")
```

- For example, the “islands” dataset

```
> islands
```

Africa	Antarctica	Asia	Australia	...
11506	5500	16988	2968	...

# Useful arithmetic functions

```
> min(islands)
```

```
[1] 12
```

```
> max(islands)
```

```
[1] 16988
```

```
> mean(islands)
```

```
[1] 1252.729
```

```
> median(islands)
```

```
[1] 41
```

```
> quantile(islands)
```

```
0% 25% 50% 75% 100%  
var(islands)
```

```
12.00 1264.50 41.00 183.25 16988.00
```

```
> sd(islands)
```

# The length function

This function returns the number of elements in a vector and is very useful for generalizing code

```
> length(islands)
[1] 48
> nislans <- length(islands)
> 1:nislans
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34
35 36 37 38 39 40 41 42 43 44 45 46 47 48
> years <- seq(from=2015, length=nislans)
> years
[1] 2015 2016 2017 2018 2019 2020 2021 2022 2023 2024 2025
2026 2027 2028 2029 2030 2031 2032 2033 2034 2035 2036 2037
2038 2039 2040 2041 2042 2043 2044 2045 2046 2047 2048 2049
2050 2051 2052 2053 2054 2055 2056 2057 2058 2059 2060 2061
2062
```



## R Reference Card 2.0

Public domain, v2.0 2012-12-24.

V 2 by Matt Baggott, matt@baggott.net

V 1 by Tom Short, t.short@ieee.org

Material from *R for Beginners* by permission of Emmanuel Paradis.

### Getting help and info

**help(topic)** documentation on topic

**?topic** same as above; special chars need quotes: for example `?"&&"`

**help.search("topic")** search the help system; same as `??topic`

**apropos("topic")** the names of all objects in the search list matching the regular expression "topic"

**help.start()** start the HTML version of help

**summary(x)** generic function to give a "summary" of x, often a statistical one

**str(x)** display the internal structure of an R object

**ls()** show objects in the search path; specify `pat="pat"` to search on a pattern

**ls.str()** str for each variable in the search path

**dir()** show files in the current directory

**methods(x)** shows S3 methods of x

**methods(class=class(x))** lists all the methods to handle objects of class x

**findFn()** searches a database of help packages for functions and returns a data.frame (*sos*)

### Other R References

**CRAN task views** are summaries of R resources for task domains at: [cran.r-project.org/web/views](http://cran.r-project.org/web/views)

Can be accessed via *ctv* package

**R FAQ:** [cran.r-project.org/doc/FAQ/R-FAQ.html](http://cran.r-project.org/doc/FAQ/R-FAQ.html)

**R Functions for Regression Analysis**, by Vito Ricci: [cran.r-project.org/doc/contrib/Ricci-refcard-regression.pdf](http://cran.r-project.org/doc/contrib/Ricci-refcard-regression.pdf)

**R Functions for Time Series Analysis**, by Vito Ricci: [cran.r-project.org/doc/contrib/Ricci-refcard-ts.pdf](http://cran.r-project.org/doc/contrib/Ricci-refcard-ts.pdf)

**R Reference Card for Data Mining**, by Yanchang Zhao: [www.rdatamining.com/docs/R-refcard-data-mining.pdf](http://www.rdatamining.com/docs/R-refcard-data-mining.pdf)

**R Reference Card**, by Jonathan Baron: [cran.r-project.org/doc/contrib/refcard.pdf](http://cran.r-project.org/doc/contrib/refcard.pdf)

## Operators

<-	Left assignment, binary
->	Right assignment, binary
=	Left assignment, but not recommended
<<-	Left assignment in outer lexical scope; not for beginners
\$	List subset, binary
-	Minus, can be unary or binary
+	Plus, can be unary or binary
~	Tilde, used for model formulae
:	Sequence, binary (in model formulae: interaction)
::	Refer to function in a package, i.e., <code>pkg::function</code> ; usually not needed
*	Multiplication, binary
/	Division, binary
^	Exponentiation, binary
%x%	Special binary operators, x can be replaced by any valid name
%%	Modulus, binary
%/%	Integer divide, binary
%*%	Matrix product, binary
%o%	Outer product, binary
%x%	Kronecker product, binary
%in%	Matching operator, binary (in model formulae: nesting)
! x	logical negation, NOT x
x & y	elementwise logical AND
x && y	vector logical AND
x   y	elementwise logical OR
x    y	vector logical OR
xor(x, y)	elementwise exclusive OR
<	Less than, binary
>	Greater than, binary
==	Equal to, binary
>=	Greater than or equal to, binary
<=	Less than or equal to, binary

## Packages

**install.packages("pkgs", lib)** download and install pkgs from repository (lib) or other external source

**update.packages** checks for new versions and offers to install

**library(pkg)** loads pkg, if pkg is omitted it lists packages

**detach("package:pkg")** removes pkg from memory

## Indexing vectors

x[n]	nth element
x[-n]	all but the nth element
x[1:n]	first n elements
x[-(1:n)]	elements from n+1 to end
x[c(1,4,2)]	specific elements
x["name"]	element named "name"
x[x > 3]	all elements greater than 3
x[x > 3 & x < 5]	all elements between 3 and 5
x[x %in% c("a","if")]	elements in the given set

## Indexing lists

x[n]	list with elements n
x[[n]]	nth element of the list
x[["name"]]	element named "name"
x\$name	as above (w. partial matching)

## Indexing matrices

x[i,j]	element at row i, column j
x[i,]	row i
x[,j]	column j
x[,c(1,3)]	columns 1 and 3
x["name",]	row named "name"

## Indexing matrices data frames (same as matrices plus the following)

X[["name"]]	column named "name"
x\$name	as above (w. partial matching)

## Input and output (I/O)

### R data object I/O

**data(x)** loads specified data set; if no arg is given it lists all available data sets

**save(file,...)** saves the specified objects (...) in XDR platform-independent binary format

**save.image(file)** saves all objects

**load(file)** load datasets written with save

### Database I/O

Useful packages: *DBI* interface between R and relational DBMS; *RJDBC* access to databases through the JDBC interface; *RMySQL* interface to MySQL database; *RODBC* ODBC database access; *ROracle* Oracle database interface driver; *RpgSQL* interface to PostgreSQL database; *RSQLite* SQLite interface for R

# Help from within R

- Getting help for a function

```
→ help("log")
```

```
→ ?log
```

- Searching across packages

```
→ help.search("logarithm")
```

- Finding all functions of a particular type

```
> apropos("log")
```

```
[7] "SSlogis" "as.data.frame.logical" "as.logical" "as.logical.factor"  
     "dlogis" "is.logical"
```

```
[13] "log" "log10" "log1p" "log2" "logLik" "logb"
```

```
[19] "logical" "loglin" "plogis" "print.logLik" "qlogis" "rlogis"
```

# Reading Data and Missing Values

A number of functions to read data files (usually `read.table`).

- Generally into data frames.

How are values not entered handled?

- R default is NA
- This can be overwritten

```
trainingBaseline = read.csv("pml-training.csv",  
                             na.strings=c("", "NA", "NULL", "#DIV/0!") )
```

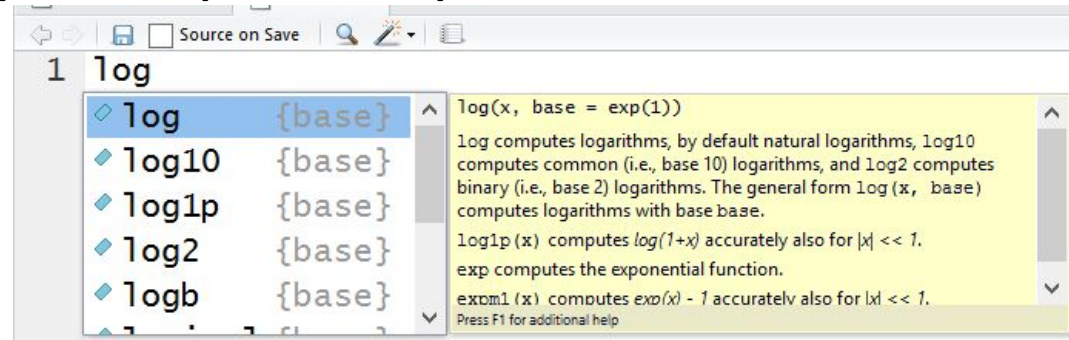
# Looking at the data

A number of handy functions. (Factor – discrete values)

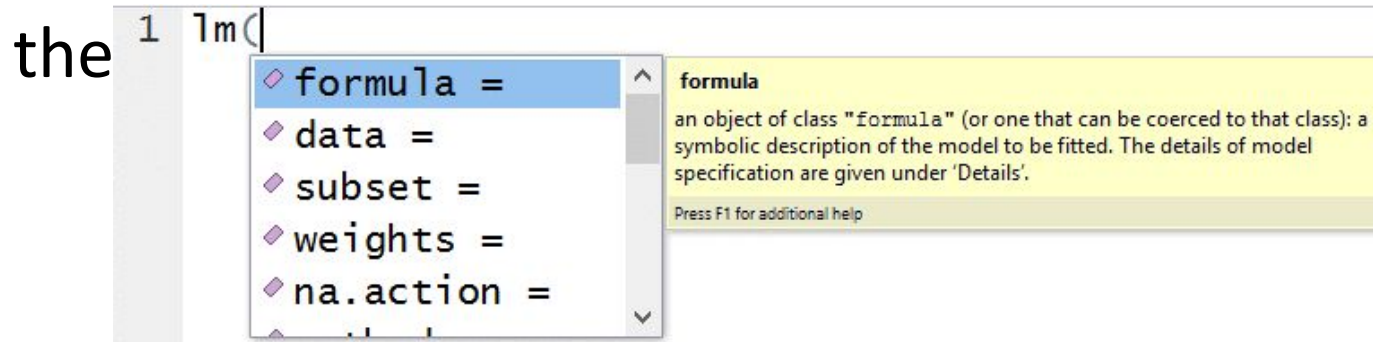
```
> data("iris")
> dim(iris)
[1] 150    5
> str(iris)
'data.frame':   150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species      : Factor w/ 3 levels "setosa","versicolor",...: 1 1 1 1 1 1 1 1 1 1 ...
> head(iris)
  Sepal.Length Sepal.width Petal.Length Petal.width Species
1          5.1          3.5          1.4          0.2  setosa
2          4.9          3.0          1.4          0.2  setosa
3          4.7          3.2          1.3          0.2  setosa
4          4.6          3.1          1.5          0.2  setosa
5          5.0          3.6          1.4          0.2  setosa
6          5.4          3.9          1.7          0.4  setosa
> |
```

# RStudio quick help

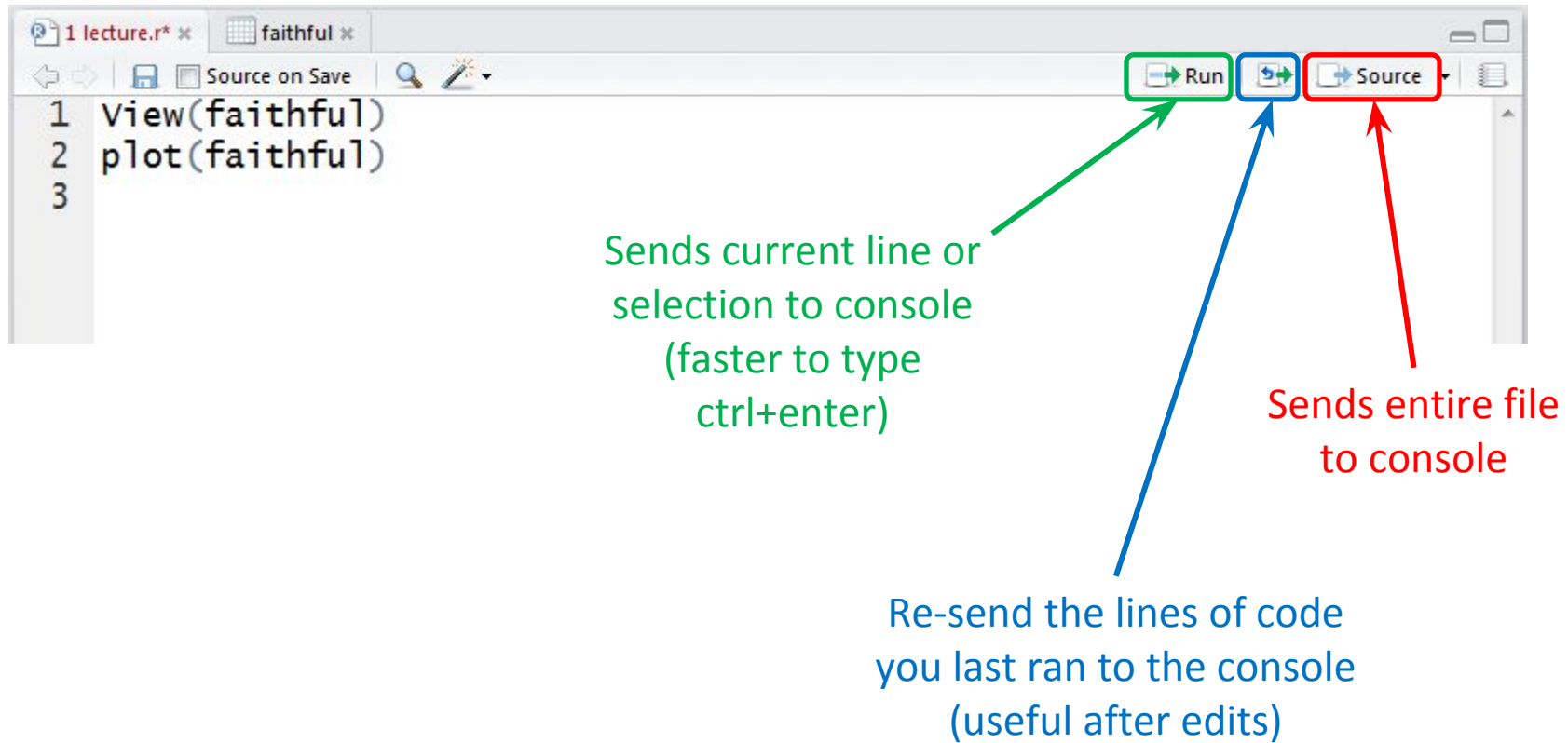
- Start typing `log` in the Scripts window (top-left) and a list of available functions starting with those letters appears, plus help



- Try typing `lm(` and then `<Tab>` for the arguments of the



# RStudio tips



# R Visualisation Packages

## plot

- Standard package. Easy to use but presentation ordinary.

## lattice

- Enhanced package. Not very widely adopted.

## ggplot (by Hadley Wickham) – Grammar of Graphics

- Best quality presentations yet easy to use
- Layers approach: ggplot
- Quickie version: qplot



# R Package Install/ Reference

To use an installed package.

At the command line.

```
library("ggplot2")
```

R Studio code hint.

Can install libraries from Github (user/repository)

```
library("devtools")
```

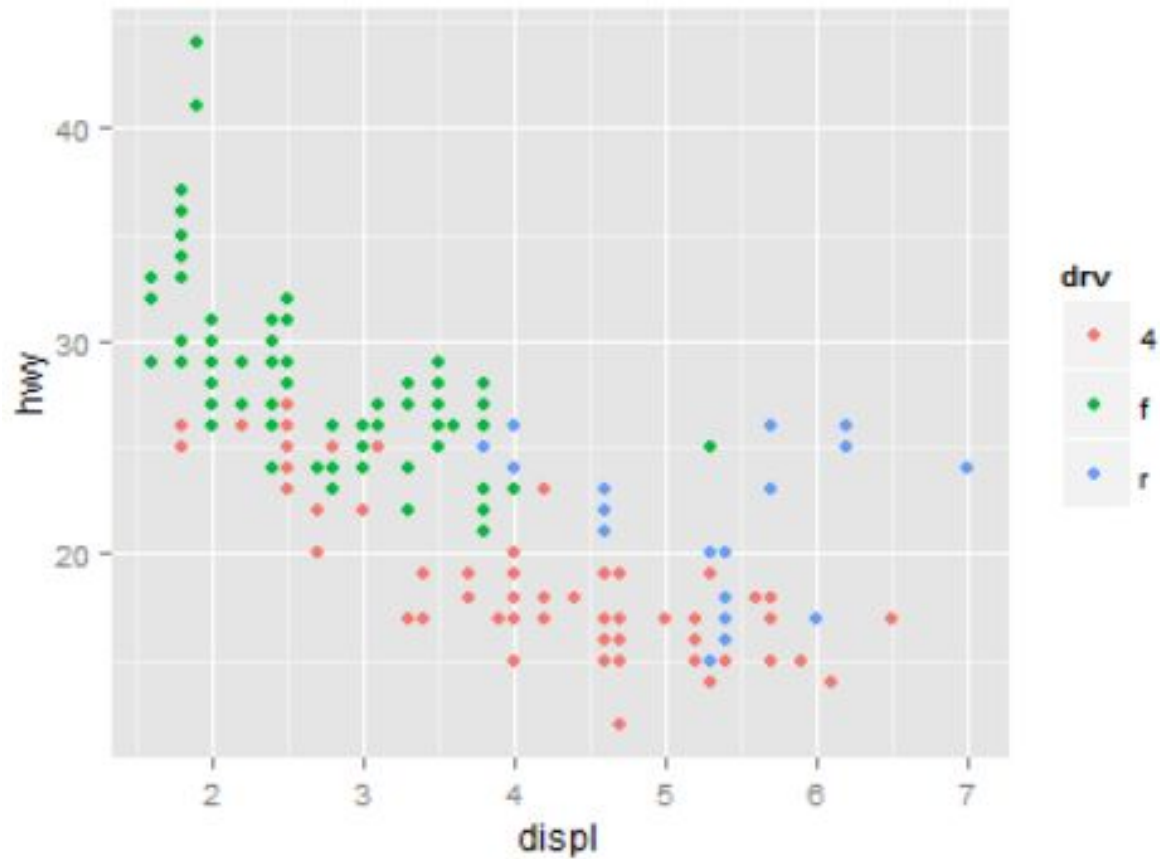
```
install_github( 'ramnathv/rCharts' )
```

Older versions of install\_github have user and repository as separate arguments



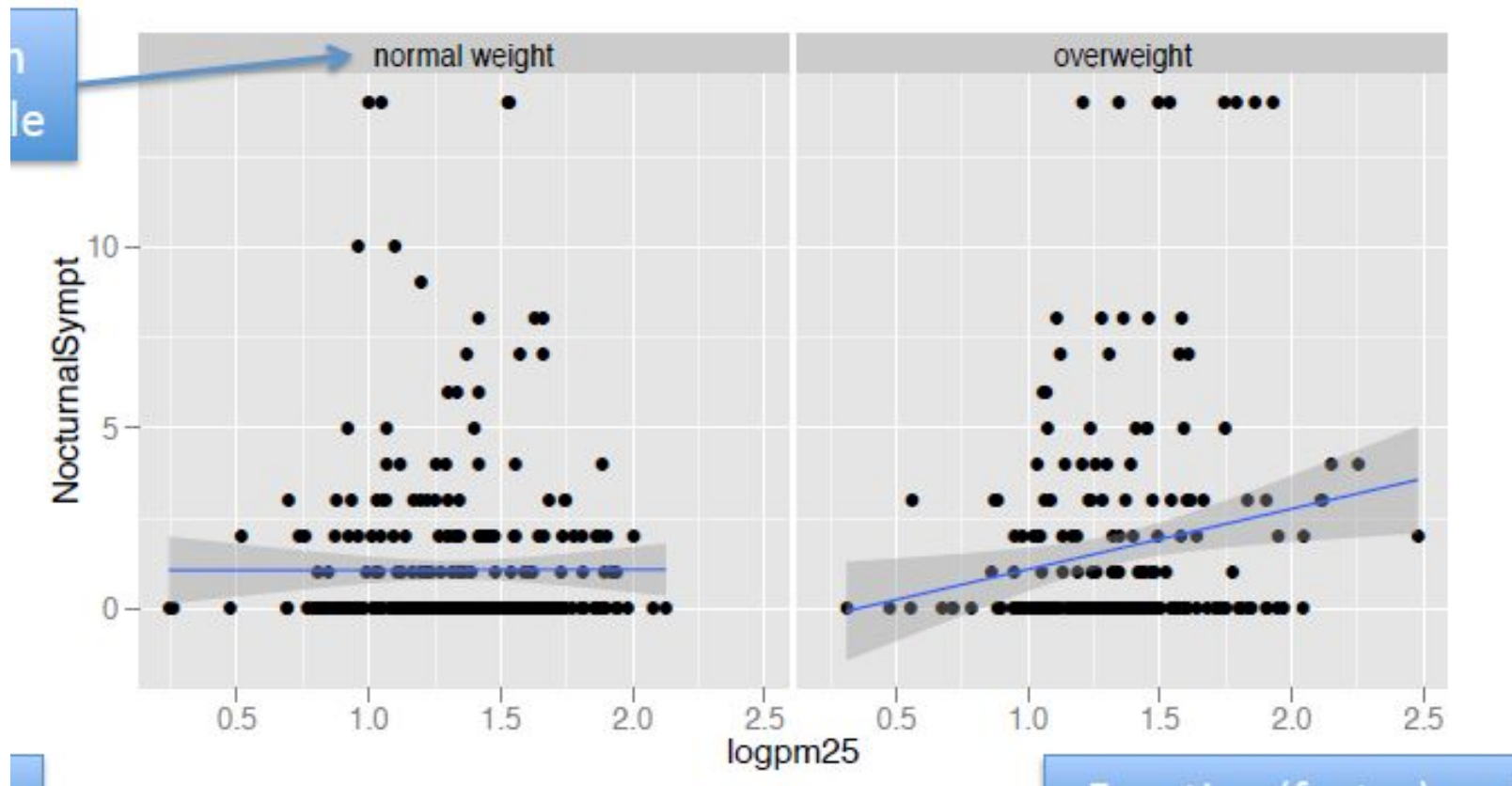
# qplot simple example

```
> library(ggplot2)  
> qplot(displ, hwy, data = mpg, color = drv)
```

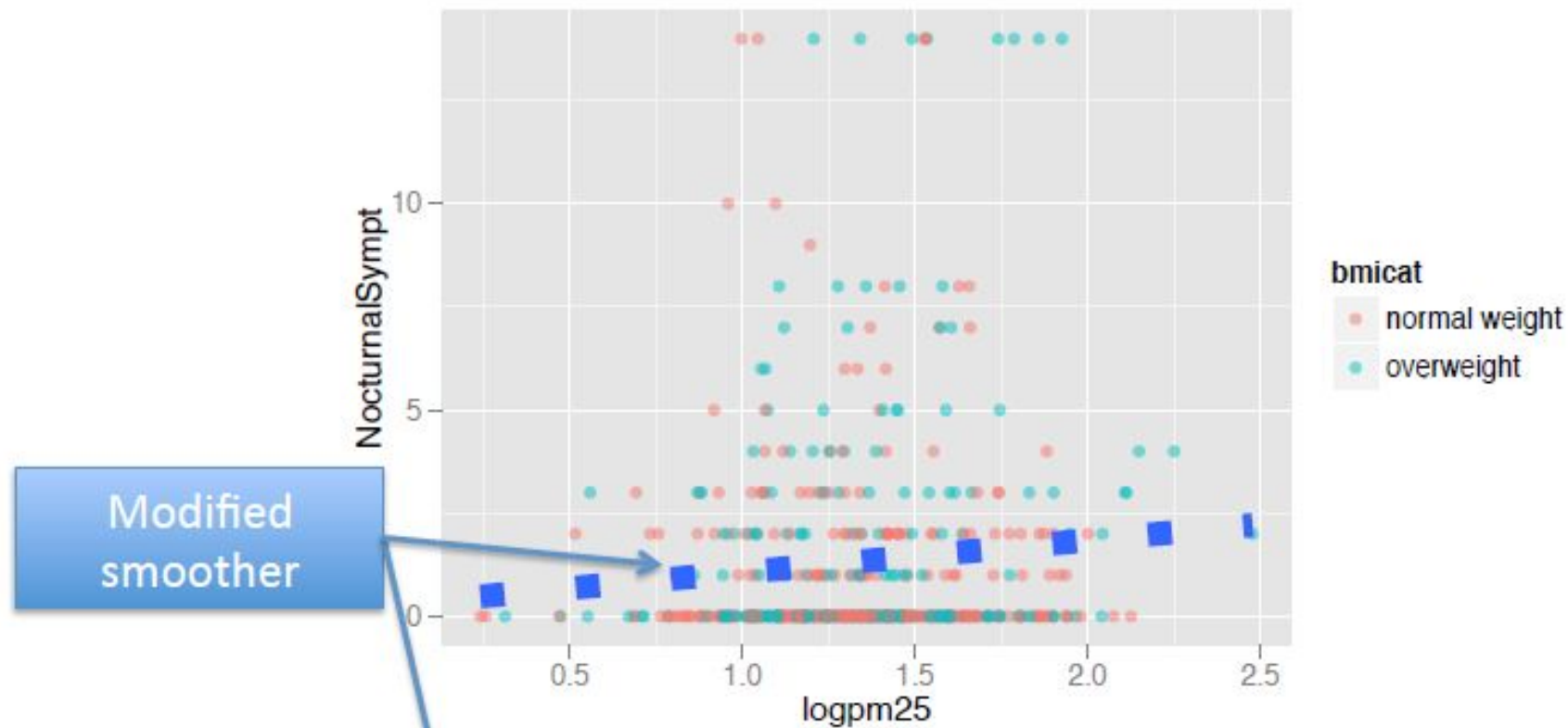


# ggplot2 example inc Linear Model

```
g <- ggplot(maacs, aes(logpm25, NocturnalSympt))  
g + geom_point() + facet_grid(. ~ bmicat) + geom_smooth(method = "lm")
```



# ggplot2 ... if you really want to get funky...

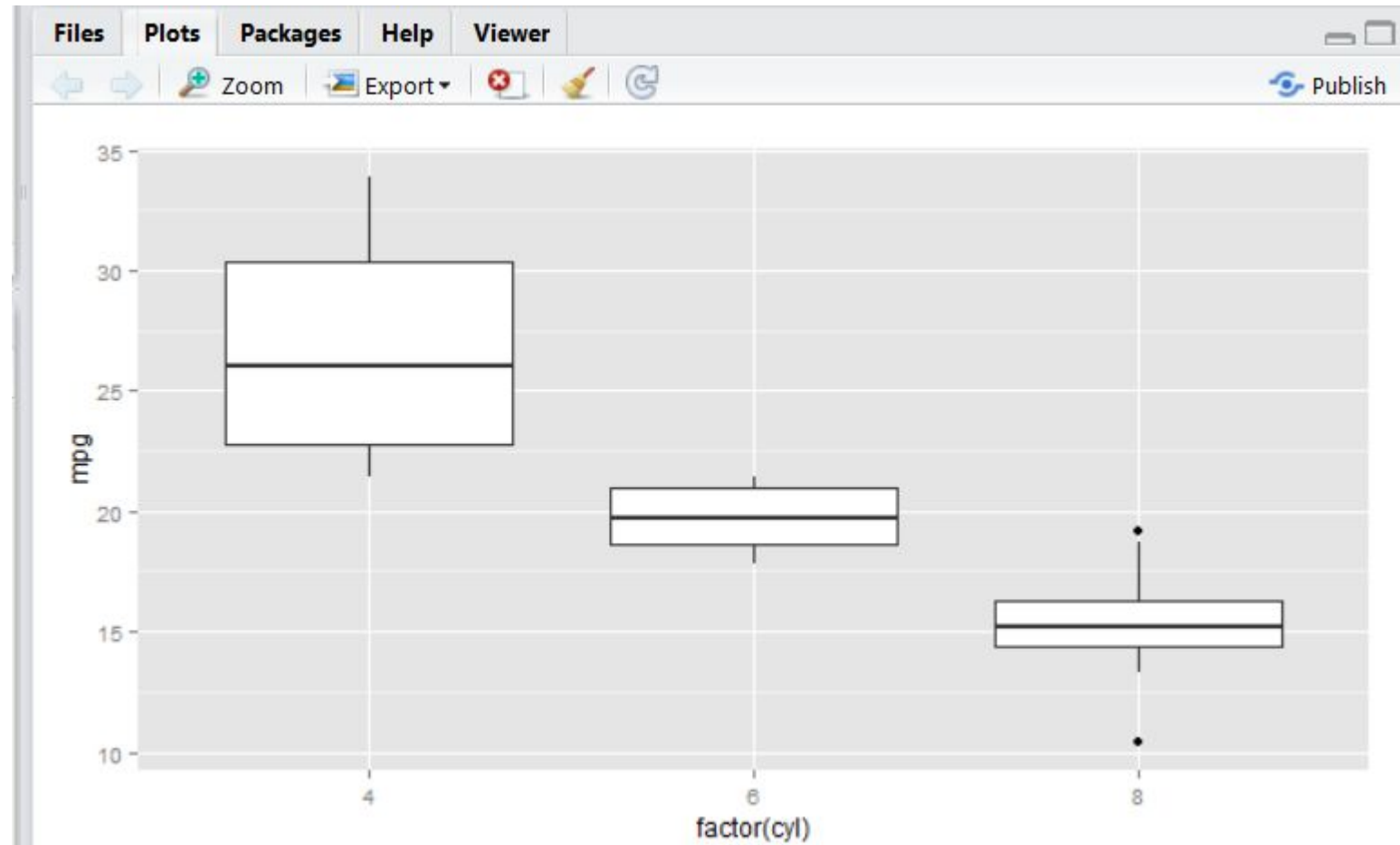


```
g + geom_point(aes(color = bmicat), size = 2, alpha = 1/2) +  
  geom_smooth(size = 4, linetype = 3, method = "lm", se = FALSE)
```

# ggplot2 and the Boxplot

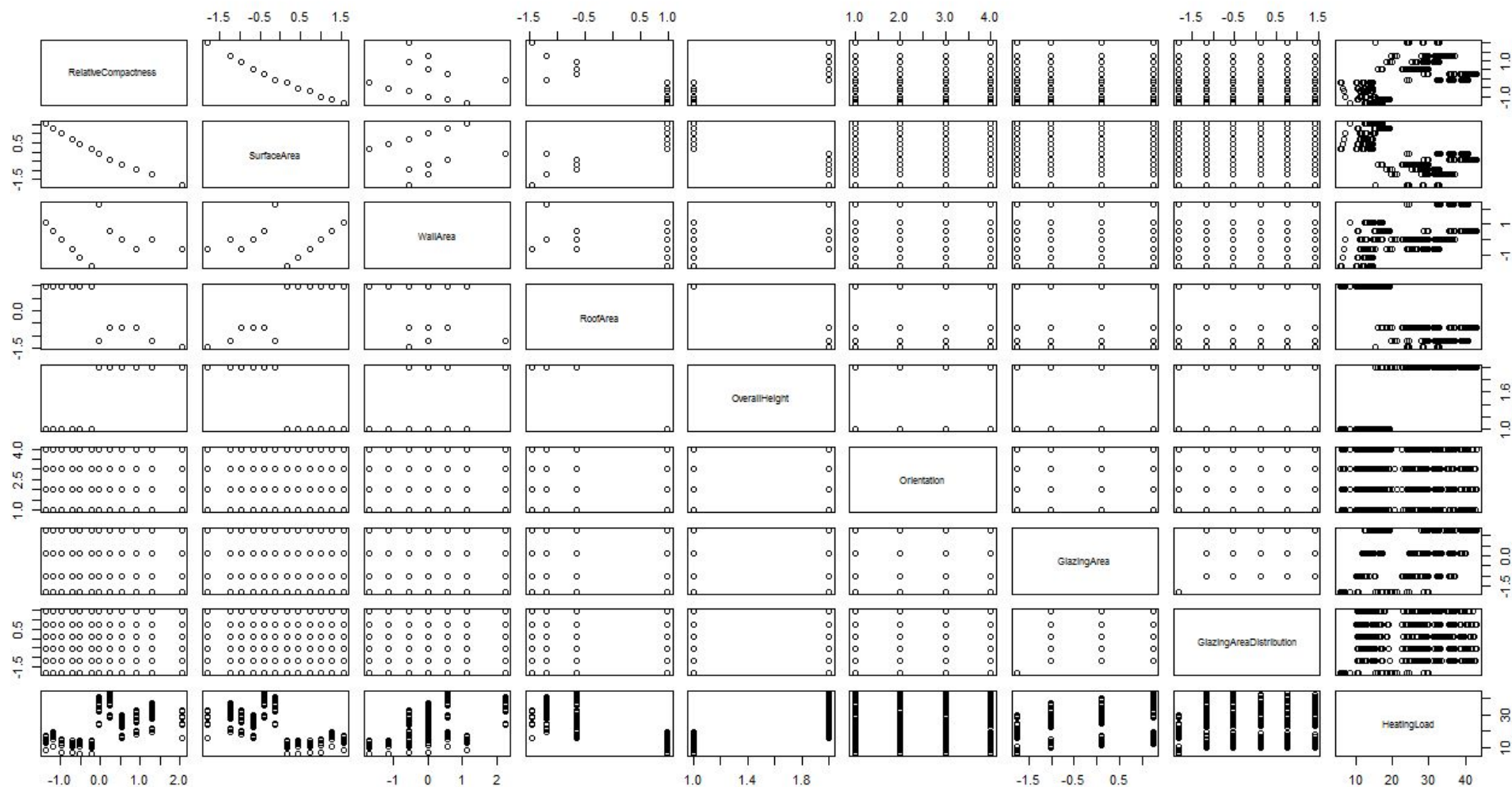
Concise way to show median, 1<sup>st</sup>/ 3<sup>rd</sup> quartiles, 1.5 \* IQR and outliers.

```
> library(ggplot2)
> p <- ggplot(mtcars, aes(factor(cyl), mpg))
> p + geom_boxplot()
> |
```



# Scatter plot matrix and R pairs function

Concise way to relationships between all features.



# R Data Wrangling Packages

## dplyr

- Extensive function set for select/ sort/ filter/ derived columns/ group by/ top n.
- Note %>% directive to chain dplyr functions – pipeline like

## tidyr (Hadley Wickham)

- Statisticians called cleansed data **tidy** data.
- Normalise/ denormalise.

## sqldf

- Surprisingly good SQL syntax fidelity

# R Dynamic Report Packages

## knitr

- R Markdown + embedded R code => reports. HTML/ PDF/ Latex.
- Ideal platform for Reproducible Research.
- Demo. Properly cool.

## shiny

- Interactive publishing of R driven web pages. Client and server bits.

## slidify

- Generation of slide decks from R Markdown/ YAML/ R.

# Some General Notes

## Algorithms vs Data

- Lots of data tends to be more influential than choice of algorithm
- Data collection methodology is critical

## Correlation implies Causation?

- No!

## Outliers

- Extreme values well outside the norm. Eg Australia's billionaires
- How are they handled? Depends.

## Variable Types (affects Algorithm choice)

- Continuous, eg apartment price
- Discrete, eg species of Iris. Don't forget R function `stringsAsFactors`



# Github Lifecycle Cheat Sheet

From [github.com](https://github.com)

- Create repository (or fork someone else's).

From local Github shell.

```
git clone <URL_of_repository>
```

```
cd <repository>
```

```
git add <files>
```

```
git commit -a -m "some_message"
```

```
git push
```