R Workshop 1: Introduction to R

Gavin Simpson

Environmental Change Research Centre,
Department of Geography
UCL

April 30, 2013

Outline

What is R

- The S statistical language was started at Bell Labs on May 5, 1976
- A system for general data analysis jobs that could replace the *ad hoc* creation of Fortran applications
- The S language was licensed by Insightful Corporation for use in their S-PLUS software
- In 2004 Insightful bought the S language from Lucent (formerly AT&T and before that Bell Labs)
- Robert Gentleman and Ross Ihaka designed a language that was compatible with S but which worked in a different way internally
- They called this language R
- There was a lot of interest in R and eventually it was made Open Source under the Gnu GPL-2
- R has drawn around it a group of dedicated stewards of the R software — R Core
- As well a large, vibrant community has developed around R and which contributes the vast number of R packages available on CRAN

Why R?

- Why use a complicated, command-line driven stats package like R?
- R is an Open Source
- Why is Open Source good? Freedom!
- But why R in particular?
  - Well, it is free!
  - R is the *lingua franca* of statistics — a lot of statisticians implement new methodologies and statistical techniques as R code
  - If something doesn't work the way you like, you can change it
  - As R is a programming language you can add your own functions
  - Also, you can use programming to manipulate data and fit a large number of models automatically
  - You can use R scripts and Sweave documents to perform reproducible research
- R works on Linux, Windows and MacOS plus others

R on the Web

- The R homepage is located at: `http://www.r-project.org`
- The download site is called CRAN — the Comprehensive R Archive Network
- CRAN is a series of mirrored web servers to spread the load of thousands of users downloading R and associated packages
- The CRAN master is at: `http://cran.r-project.org`
- The UK mirror is at: `http://cran.uk.r-project.org`

Starting R and other preliminaries

- You start R in a variety of ways depending on your OS
- R starts in a **working directory** where it looks for files and saves objects
- Best to run R in a new directory for each project or analysis task
- `getwd()` and `setwd()` get and set the working directory
- To exit R, the function `q()` is used
- You will be asked if you want to save your workspace; invariably you should answer `n` to this

```
> getwd()
> setwd("~/work")
> getwd()
```

Getting help

- R comes with a lot of documentation
- To get help on functions or concepts within R, use the `"?"` operator
- For help on the `getwd()` function use: `?getwd`
- Function `help.search("foo")` will search through all packages installed for help pages with `"foo"` in them
- How the help is displayed is system dependent
- To search on-line, use `RSiteSearch()`; this opens results in your web browser and includes searching of the R-Help mailing list

```
> help.search("directory")
> RSiteSearch("directory")
```

Working with R; entering commands

- Type commands at prompt ">" and these are evaluated when you hit RETURN
- If a line is not syntactically complete, the prompt is changed to "+"
- If returned object not assigned, it is printed to console
- Assigning the results of a function call achieved by the assignment operator "<-"
- Whatever is on the right of "<-" is assigned to the object named on the left of "<-"
- Enter the name of an object and hit RETURN to print the contents
- `ls()` returns a list of objects currently in your workspace

```
> 5 * 3

[1] 15

> radius <- 5
> pi * radius^2

[1] 78.53982

> ans <- 5 * 3
> ans

[1] 15

> ans2 <- ans + 20
> ans2

[1] 35

> ls()

[1] "ans"    "ans2"   "radius"
```

Basic R object types

- R has several basic object types
  - **vectors** (character, numeric, factors, Date)
  - **matrices** (numeric or character)
  - **data** frames (matrix-like object with components [columns] of different types)
  - **lists** (arbitrary structures that form basis of many returned objects in R)
- Vectors and matrices contain elements of same basic type
- Data frames are more like Excel spreadsheets; each column can contain a different type of data
- But each column contains only a single type of object
- Data frames must also have components (columns) of the same length

Vectors

```
> vec <- c(1, 2, 2.5, 6.2, 4.8, 3.1)
> vec
[1] 1.0 2.0 2.5 6.2 4.8 3.1
> length(vec)
[1] 6
> chr.vec <- c("one", "two", "three")
> chr.vec
[1] "one"   "two"   "three"
> rnd <- rnorm(20)
> rnd
 [1] -1.20706575  0.27742924  1.08444118 -2.34569770  0.42912469  0.50605589
 [7] -0.57473996 -0.54663186 -0.56445200 -0.89003783 -0.47719270 -0.99838644
[13] -0.77625389  0.06445882  0.95949406 -0.11028549 -0.51100951 -0.91119542
[19] -0.83717168  2.41583518
```

- A vector is a set of 0 or more elements of the same type
- Two main types; character and numeric
- A scalar is a vector of length 1
- When printed, R prepends "`[x]`" to each line - this tells you which element of the vector starts each line
- The `c()` function can be used to create vectors; short for combine or concatenate

Special vectors; factors, Dates

```
> fac <- c("red","blue","green","red","blue","red")
> fac <- factor(fac)
> fac

[1] red   blue  green red   blue  red
Levels: blue green red

> dates <- c("01/11/2007", "10/11/2007", "19/11/2007")
> dates <- as.Date(dates, format = "%d/%m/%Y")
> dates

[1] "2007-11-01" "2007-11-10" "2007-11-19"

> class(dates)

[1] "Date"
```

- Factors are special vectors, used when elements come from a set of possible choices: Male/Female
- R codes these numerically internally, but the labels are easy to read
- Factors can be **ordered**: `ordered()`
- Dates are a special data type and we convert from textual representations into something R understands using `as.Date()`

Matrices

```
> mat <- matrix(1:9, ncol = 3)
> mat
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> chr.mat <- matrix(letters[1:9], nrow = 3)
> chr.mat
     [,1] [,2] [,3]
[1,] "a"  "d"  "g"
[2,] "b"  "e"  "h"
[3,] "c"  "f"  "i"
> matrix(1:9, ncol = 3, byrow = TRUE)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
> dim(mat)
[1] 3 3
```

- Matrices are vectors with dimensions; numeric or character matrices
- All elements of a matrix must be the same type
- By default R fills matrices by column; use argument `"byrow = TRUE"` to change this
- `dim()` returns the dimensions, rows first then cols

Data frames and lists

```
> (df <- data.frame(Var1 = 1:4, Var2 = letters[1:4], Var3 = factor(c("M","M","F","M")), Var5 = rnorm(4)))

  Var1 Var2 Var3       Var5
1    1    a    M  0.1340882
2    2    b    M -0.4906859
3    3    c    F -0.4405479
4    4    d    M  0.4595894

> (lst <- list(A = 1, B = c("Yes","No"), C = matrix(1:4, ncol = 2)))

$A
[1] 1

$B
[1] "Yes" "No"

$C
     [,1] [,2]
[1,]    1    3
[2,]    2    4
```

- Data frames are the main object to handle your own data in R
- Like an Excel spreadsheet; each column can be a different type of data
- You can create data frames yourself using `data.frame()`
- Most likely they result from reading your data into R
- Lists generalize data frames; the components of a list can contain any R object

Subsetting

```
> vec[3]
[1] 2.5
> vec[2:5]
[1] 2.0 2.5 6.2 4.8
> vec[-4]
[1] 1.0 2.0 2.5 4.8 3.1
> mat[2, 3]
[1] 8
> df$Var2 #$
[1] a b c d
Levels: a b c d
> lst[["B"]]
[1] "Yes" "No"
```

- Subsetting usually done by "`[...]`"
- For vectors, select elements numerically within "`[...]`"
- Negative indices drop those elements
- Matrices and data frames have rows and columns, subsetting becomes "`[r, c]`"
- Components of data frames and list can be selected by use of $
- Lists also subset using "`[[...]]`"

Sequences and patterned vectors

```
> seq(from = 1, to = 10, by = 2)
[1] 1 3 5 7 9
> 1:5
[1] 1 2 3 4 5
> rep(1:3, each = 2)
[1] 1 1 2 2 3 3
> rep(1:3, times = 3:1)
[1] 1 1 1 2 2 3
```

- Sequences and patterned vectors are very useful in some circumstances
- `seq()` is a flexible function to produce sequences of numbers
- `rep()` creates repetitions of its first argument
- The `":"` operator is short hand for `seq(from = x, to = y, by = 1)`

Functions

```
> args(rnorm)

function (n, mean = 0, sd = 1)
NULL

> rnorm(n = 5, mean = 2, sd = 3)

[1] -0.08116074 -2.34461473  3.72426716 -1.07096717  1.95458510

> rnorm(5, 2, 3)

[1] -0.8078458  5.3068926  0.5732208 -0.1283201  0.4962258

> foo <- function(x, y) {(x + y) * 2}
> foo(3, 6)

[1] 18
```

- Functions are R objects that include one or more R function calls
- Encapsulate a set of operations on one or more arguments
- Arguments (options) to functions are entered within `"(...)"`
- Arguments are named and entered in `name = value` pairings
- Don't need to use argument names; but be careful, arguments matched by position

Reading in data from external files

```
> dat <- read.csv("test_file.csv", row.names = 1)
> dat

      Var1 Var2 Var3
Samp1    1    2    3
Samp2    5    6    7
Samp3    3    5    7
Samp4    3    7    9
Samp5    2    3    5

> class(dat)

[1] "data.frame"
```

- R can read from a wide range of file type, connections and databases
- Much of this is beyond the scope of today's workshop
- Easiest to produce spreadsheets in Excel/OpenOffice.Org and save each sheet off as a comma-separated file (*.csv)
- Any labels in row/column 1
- Read data from *.csv file using the `read.csv()` function
- Returns a data frame

Saving objects and writing data out of R

```
> save(dat, file = "test_data_object.rda")
> rm(dat)
> load(file = "test_data_object.rda")
> ls()

 [1] "ans"     "ans2"    "chr.mat" "chr.vec" "dat"     "dates"   "df"
 [8] "fac"     "foo"     "lst"     "mat"     "radius"  "rnd"     "vec"

> write.csv(dat, file = "temp_file2.csv")
> read.csv("temp_file2.csv", row.names = 1)

      Var1 Var2 Var3
Samp1    1    2    3
Samp2    5    6    7
Samp3    3    5    7
Samp4    3    7    9
Samp5    2    3    5
```

- `save()` saves are R object in a compressed, portable format; useful for saving objects that are expensive to produce by don't change regularly
- `load()` used to load saved R objects
- `write.csv()` can be used to write matrix-like objects out as *.csv files

Model formulae in R

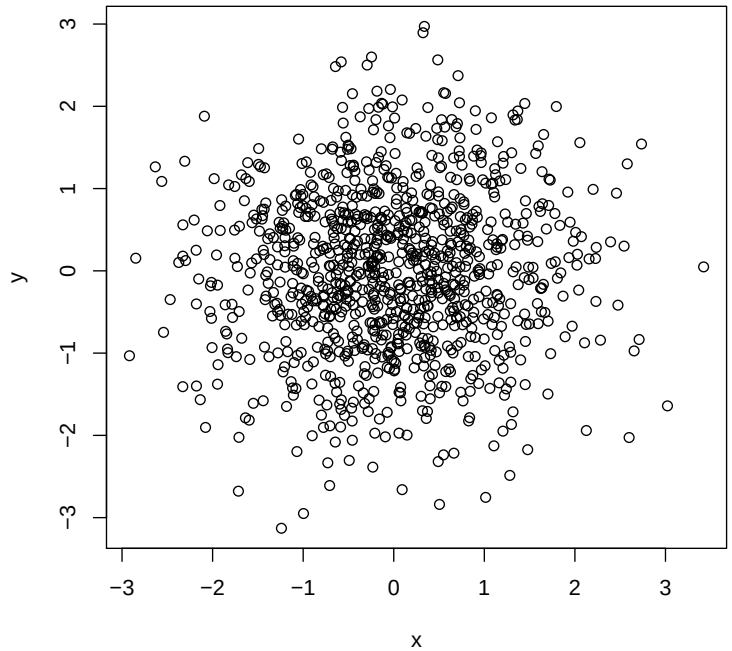
```
> set.seed(123)
> x1 <- runif(100)
> x2 <- runif(100)
> y <- 4 + (2.1 * x1) + (-3.4 * x2) + rnorm(100, 0, 3)
> mod <- lm(y ~ x1 + x2)
> formula(mod)

y ~ x1 + x2
```

- Models and some graphics can be specified using a model formula to symbolically describe the statistical model or relationships between data
- The model above has an implied intercept, which we can drip by adding `- 1` or `+ 0` to the formula
- Interactions between two variables can be added using `:`, e.g. `y ~x1 + x2 + x1:x2`
- There are several shortcuts:
  - The interaction can be simplified to `y ~x1*x2`
  - To refer to all variables in a data frame use `.`, e.g. `y ~.` assuming the variables were in a data frame object

Graphics in R

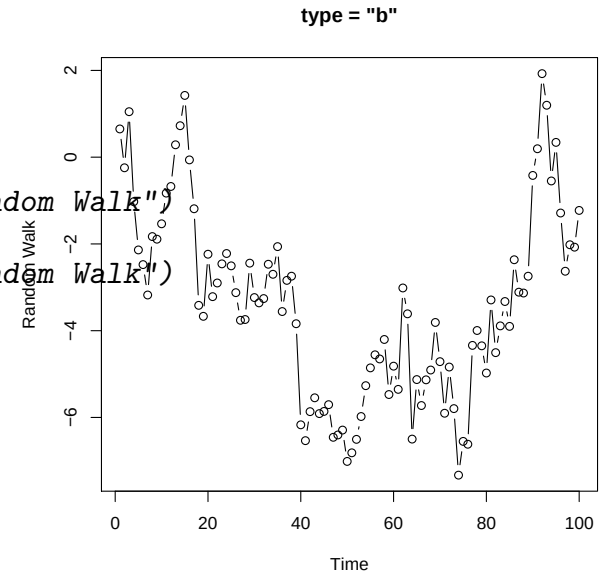
```
> x <- rnorm(1000)
> y <- rnorm(1000)
> plot(x, y)
```

- Standard plotting command is `plot()`
- Takes one or two arguments of coordinates
- By default draws a scatterplot
- R's graphics are like drawing with pen on paper; once you draw anything that sheet of paper is no-longer pristine and you can't erase anything you have drawn
- Vast array of parameters to alter look of plots; see `?par`

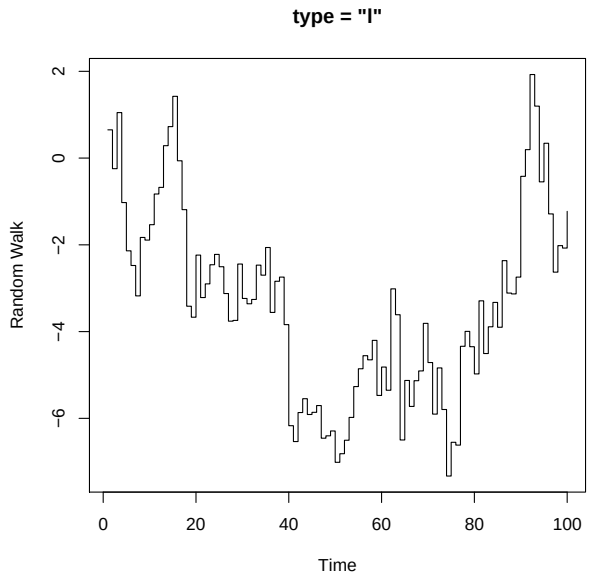


Graphics in R

```
> x <- 1:100
> y <- cumsum(rnorm(100))

> plot(x, y, type = "b", main = "type = \"b\"", xlab = "Time", ylab = "Random Walk")

> plot(x, y, type = "s", main = "type = \"l\"", xlab = "Time", ylab = "Random Walk")
```

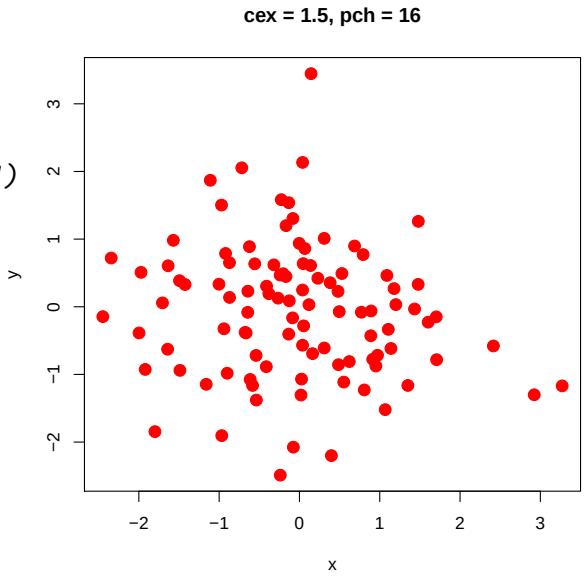


- The `"type"` argument changes the type of plotting done
  - `type = "p"` draws points
  - `type = "l"` draws lines
  - `type = "o"` draws lines and points over-plotted
  - `type = "b"` draws lines and points
  - `type = "h"` draws histogram-like bars
  - `type = "s"` draws stepped lines
- `"main"` control the title of the plot
- `"xlab"` & `"ylab"` control axis labels

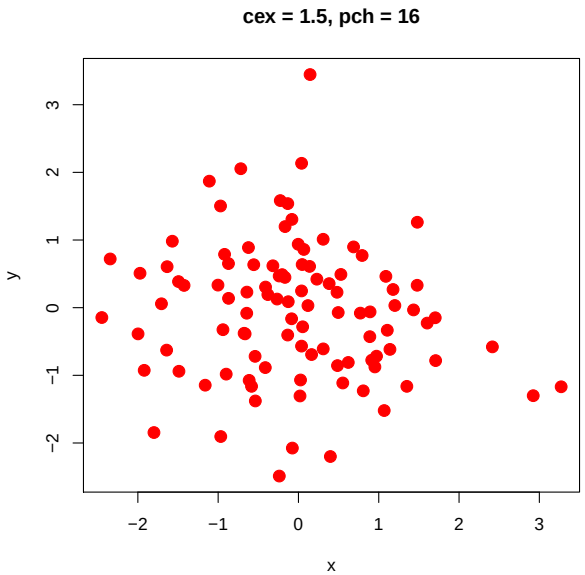


Graphics in R

```
> x <- rnorm(100)
> y <- rnorm(100)

> plot(x, y, main = "cex = 1.5, pch = 16", cex = 1.5, pch = 19, col = "red")

> plot(x, y, cex = 1.5, pch = 19, col = "red", axes = FALSE, ann = FALSE)
> axis(side = 1)
> axis(side = 2)
> title(main = "cex = 1.5, pch = 16", xlab = "x", ylab = "y")
> box()
```

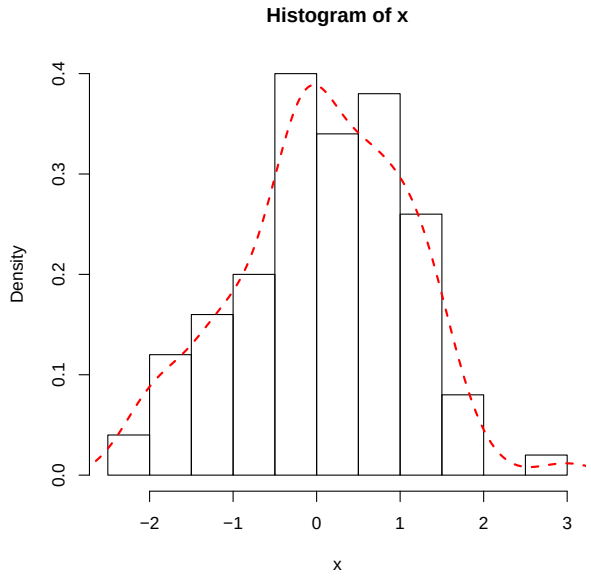


- "pch" controls the plotting character
- "cex" controls the size of the character
- "col" controls colour
- "axes" logical; should axes be drawn
- "ann" logical; should the plot be annotated
- axis(), title(), box() used to build up plotting
- Allows finer control

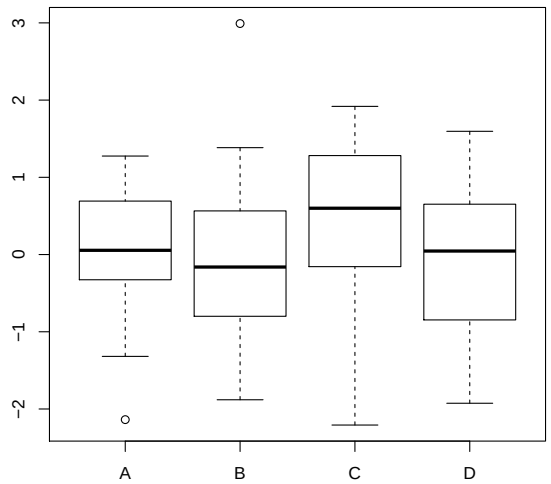


Graphics in R

```
> x <- rnorm(100)
> grps <- factor(sample(LETTERS[1:4], 100, replace = TRUE))

> dens <-density(x)
> hist(x, freq = FALSE)
> lines(dens, col = "red", lwd = 2, lty = "dashed")

> boxplot(x ~ grps)
```

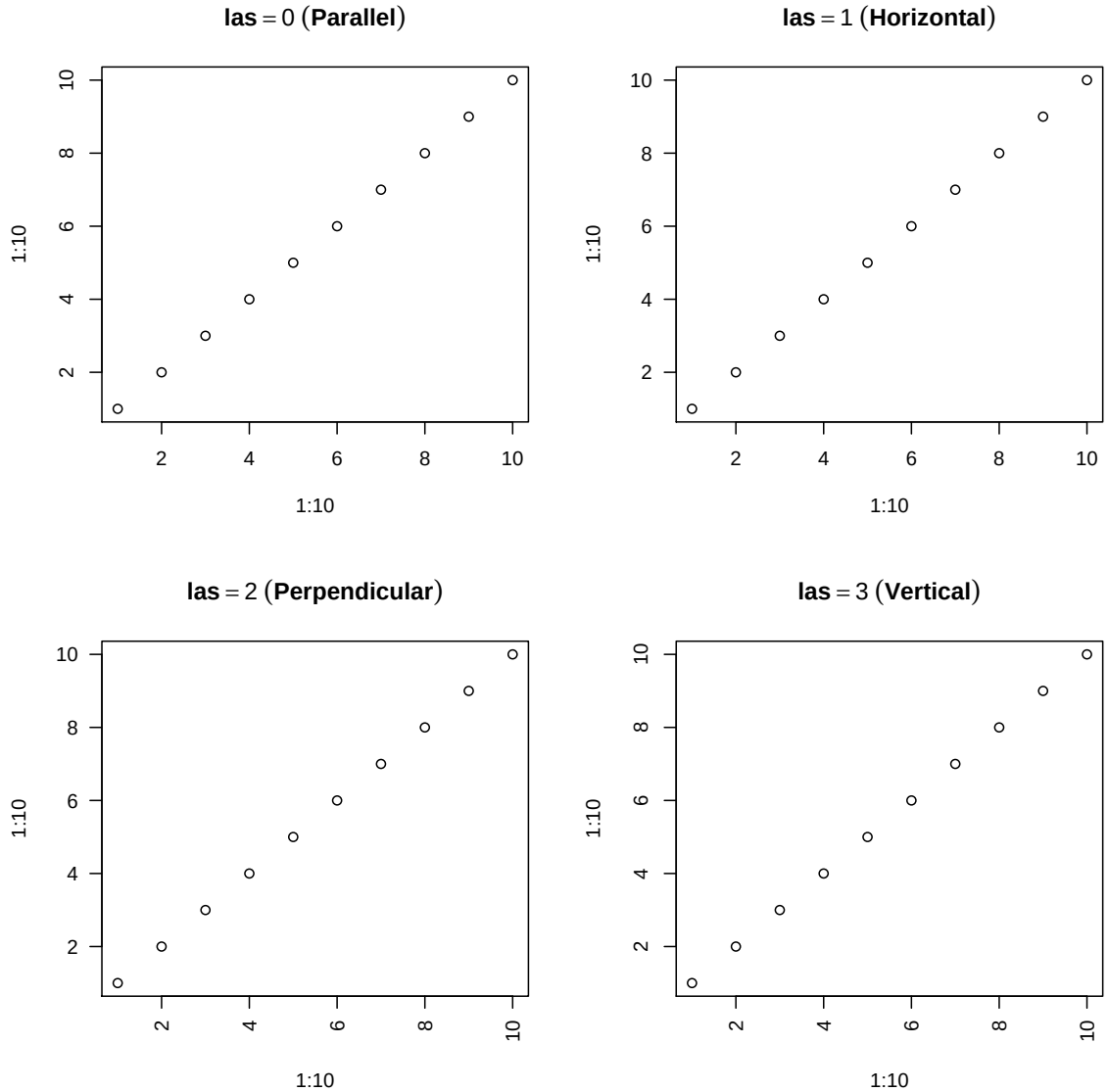


- `hist()` draws histograms
- `boxplot()` draws boxplots
- `"lwd"` controls the line width
- `"lty"` controls the line type
- `lines()` used to add lines to an existing plot
- Also `points()`

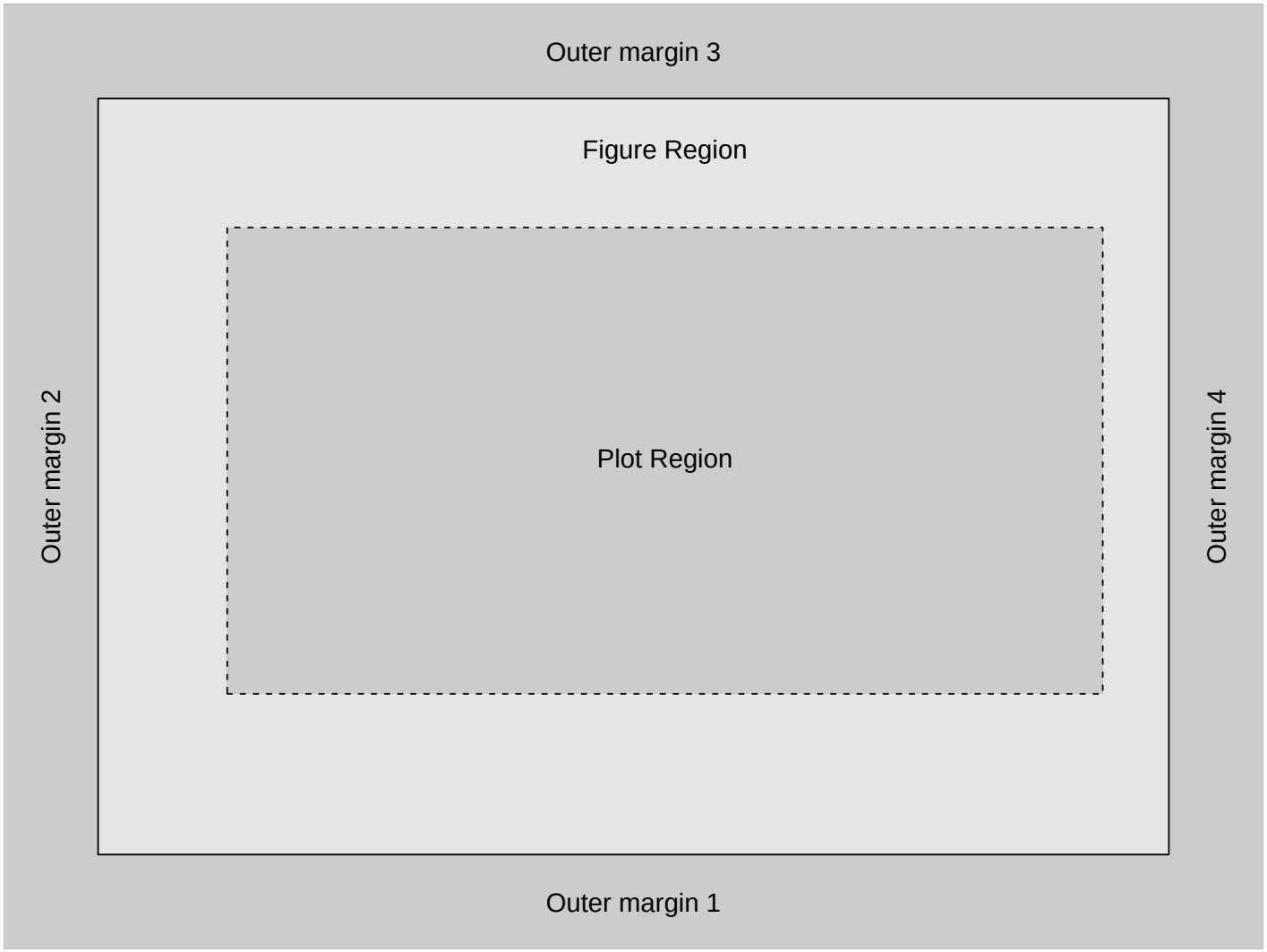


Controlling tick labels

Gross rotation of tick-labels is controlled by parameter `las`



Plotting device regions and margins



Plotting device regions and margins

- Control the size of margins using several parameters
  - `mar` — set margins in terms of number of lines of text
  - `mai` — set margins in terms of number of inches
- Specify as a vector of length 4 — `mar = c(5,4,4,2) + 0.1)`
- The ordering is Bottom, Left, Top, Right

```
> x <- runif(100)
> y <- 4 + (2.1 * x) + rnorm(100, 0, 3)
> op <- par(mar = c(4,4,4,4) + 0.1)
> plot(y ~ x)
> op <- par(op)
```

- The outer margin is controlled via parameter `oma` and `omi`, just like `mar`
- By default, there is no outer margin — `oma = c(0,0,0,0)`

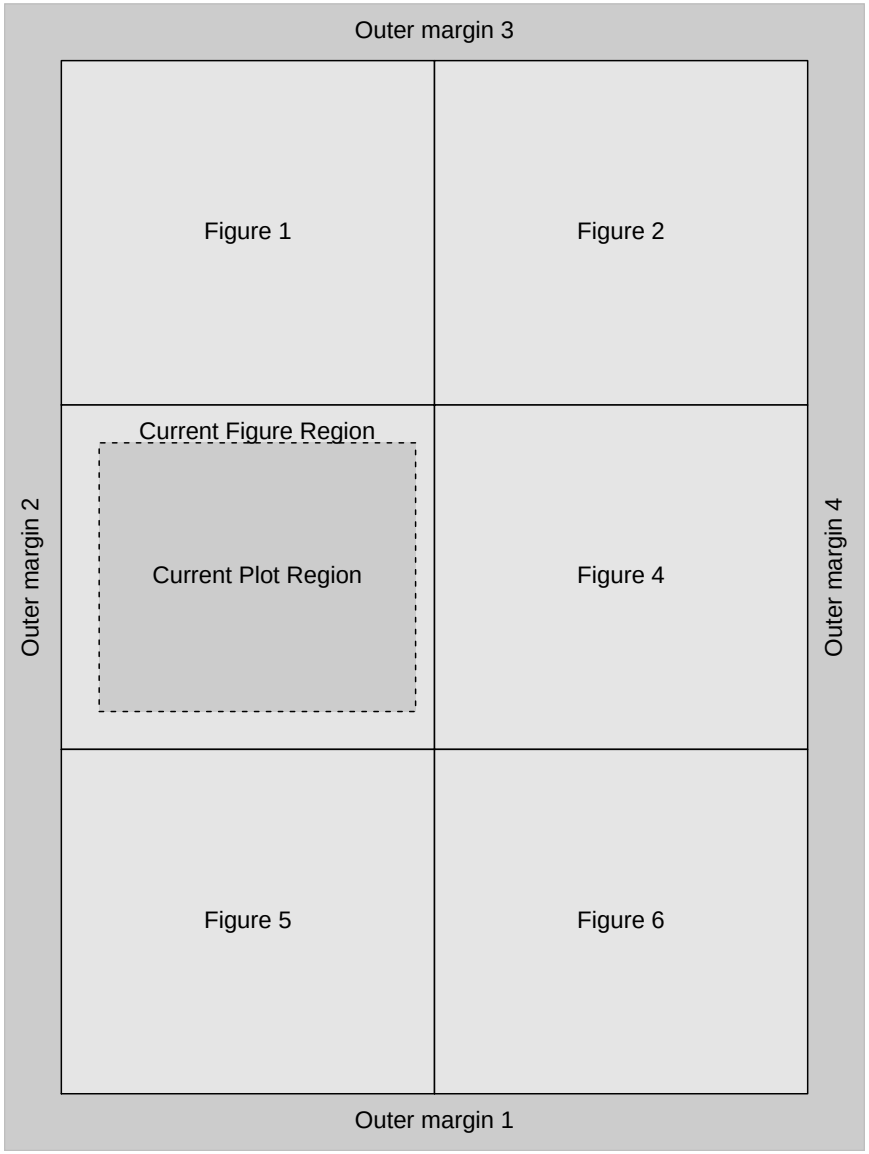
```
> x <- runif(100)
> y <- 4 + (2.1 * x) + rnorm(100, 0, 3)
> op <- par(mar = c(4,4,4,4) + 0.1, oma = rep(2,4))
> plot(y ~ x)
> op <- par(op)
```

Setting graphical parameters

- Base graphic are controlled by a large number of plotting graphical parameters
- These are detailed in the help page `?par`
- Graphical parameters are changed using the `par()` function and some may be changed within plotting calls
- To avoid getting into a muddle, when changing par you should
  - Store the defaults
  - Change your parameters as required
  - When finished the current plot, reset the parameters
- The first two can be done with a single R call

```
> ## Store defaults in 'op' and change current parameters
> op <- par(las = 2, mar = rep(4,4), oma = c(1,3,4,2), cex.main = 2)
> plot(1:10) ## plot something
> par(op) ## reset
```

Plotting on multiple device regions



Plotting on multiple device regions

- Several ways to split a region into multiple plotting regions
  - Graphical parameters `mfrow` & `mfcol`
  - The `layout()` function
  - The `split.screen()` function
- Upper plot produced with

```
> op <- par(mfrow = c(2,2))
> plot(1:10)
> plot(1:10)
> plot(1:10)
> plot(1:10)
> par(op)
```

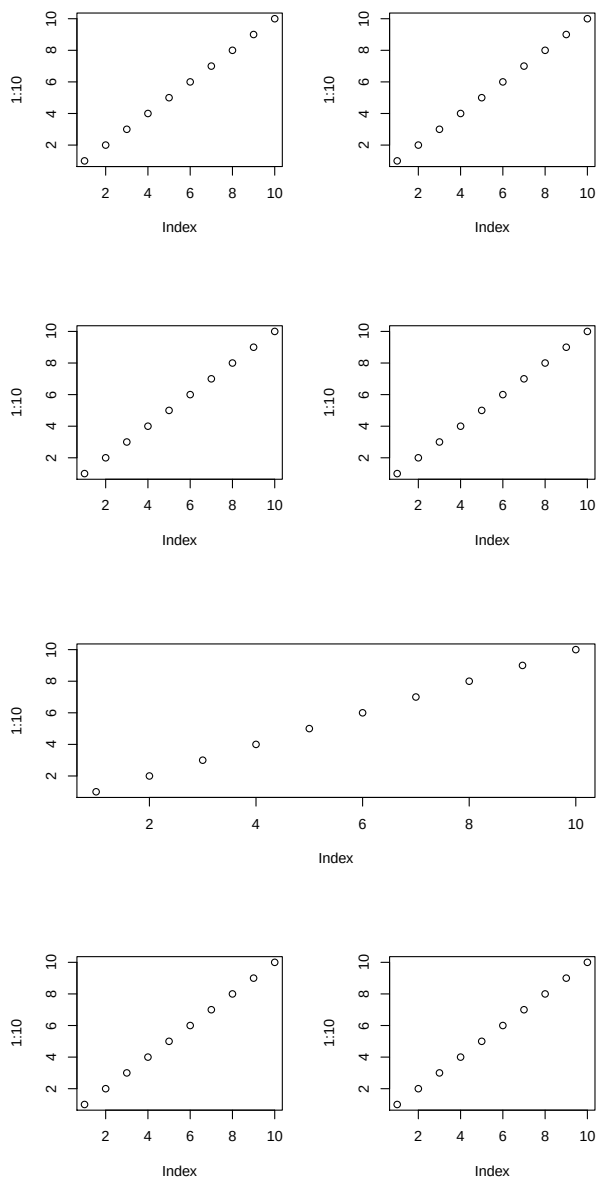
- Lower plot produced with

```
> layout(matrix(c(1,1,2,3), ncol = 2, byrow = TRUE))
> plot(1:10)
> plot(1:10)
> plot(1:10)
> layout(1)
```

- The whole first row used for region 1

```
> matrix(c(1,1,2,3), ncol = 2, byrow = TRUE)
```

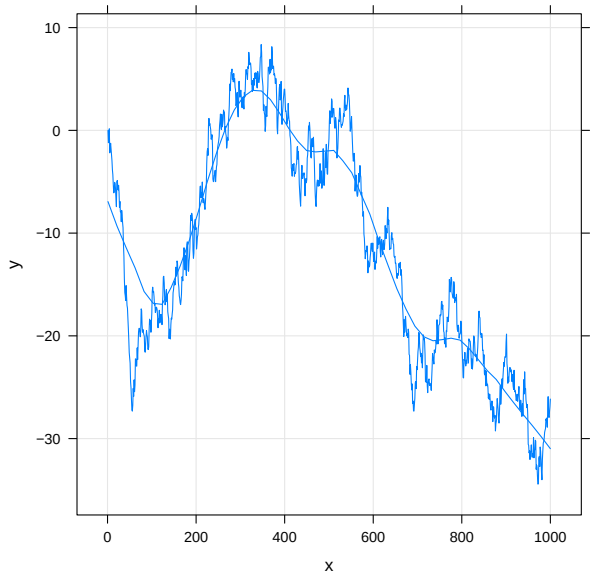
```
     [,1] [,2]
[1,]    1    1
[2,]    2    3
```



Lattice Plots

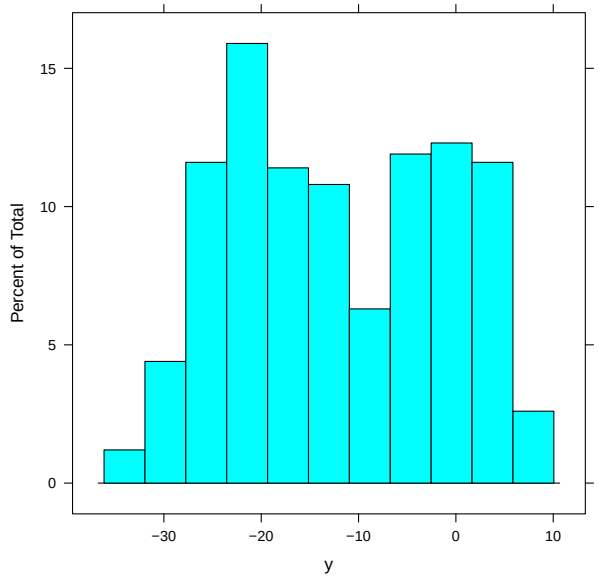
- Lattice graphics is a high level plotting package based on the Trellis graphics of Bill Cleveland
- Can be used to produce versions of the standard plots but comes into it's own when we condition on other variables to produce multiple plots, one per group

```
> set.seed(1234)
> dat <- data.frame(x = 1:1000, y = cumsum(c(0, rnorm(999))))

> xyplot(y ~ x, data = dat, type = c("l","smooth","g"), span = 0.2)
```



- Other basic plot types
  - `histogram()`
  - `bwplot()`
  - `densityplot()`

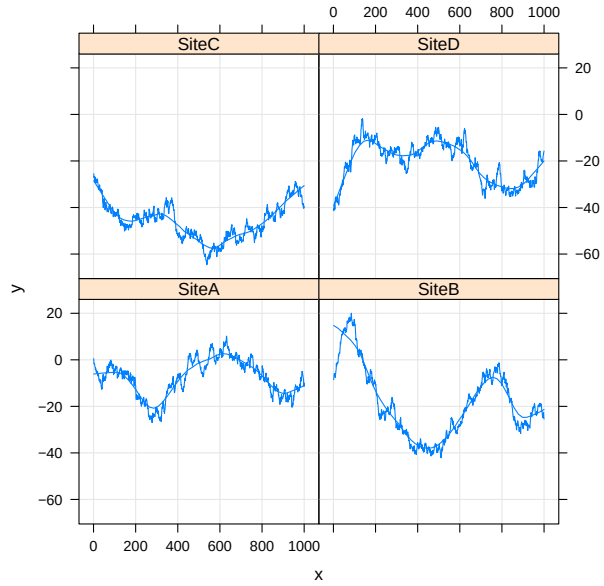
```
> histogram(~ y, data = dat)
```

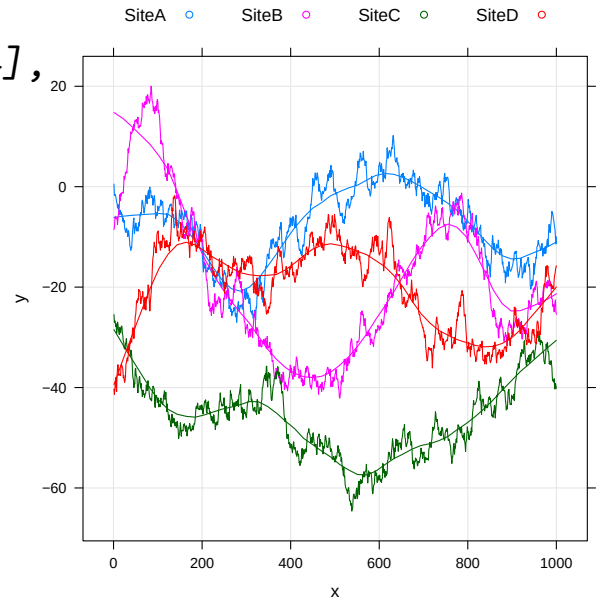


Trellising

- Multiple sets of data of the same thing recorded on several groups
- Plot all groups in separate panels or in same panel with different coding

```
> set.seed(789)
> dat2 <- data.frame(x = rep(1:1000, 4),
+                    y = cumsum(rnorm(1000*4)),
+                    Site = factor(rep(paste("Site",
+                                            LETTERS[1:4],
+                                            sep = ""),
+                                      each = 1000)))

> xyplot(y ~ x | Site, data = dat2,
+        type = c("l","smooth","g"), span = 0.2)

> xyplot(y ~ x, data = dat2, group = Site,
+        type = c("l","smooth","g"), span = 0.2,
+        auto.key = list(space = "top", columns = 4))
```

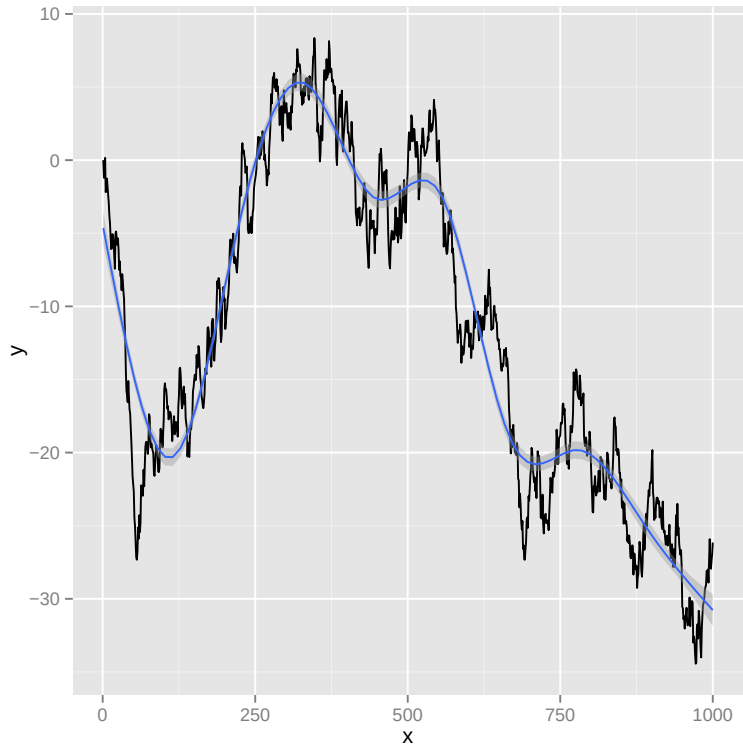




Plotting with ggplot2

- ggplot2 is what all the cool, young kids are using
- High-level plotting package like Lattice, but designed for ease of use
- Based on the *Grammar of Graphics*
- `qplot()` is the simple function for **q***uick plot*
- Build plots up in layers using `geoms`

```
> qplot(x, y, data = dat,
+       geom = c("line","smooth"))

> qplot(x, y, data = dat, geom = "line") +
+     geom_smooth()
```



Basic ggplot2 usage

- Basic form of `qplot()` call is `qplot(xVar, yVar, dataObject)`

```
> data(diamonds)
> set.seed(1410)
> dsmall <- diamonds[sample(nrow(diamonds),
+                           100), ]
> qplot(carat, price, data = diamonds)
```

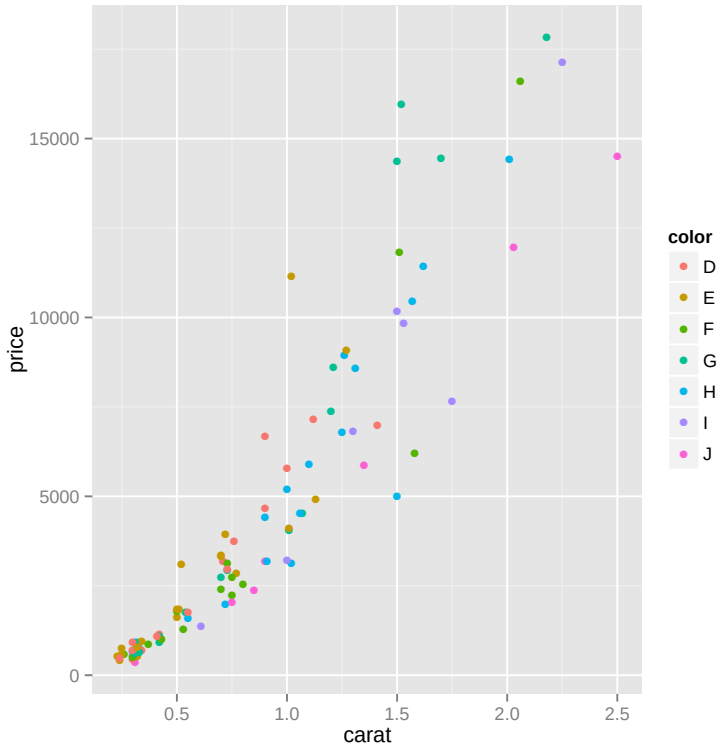
- We can use functions of variables within the call

```
> qplot(log(carat), log(price),
+       data = diamonds)
```

- To condition on a third variable we can vary the colour or the shape of the plotting characters

```
> qplot(carat, price, data = dsmall,
+       colour = color)
```

```
> qplot(carat, price, data = dsmall,
+       shape = cut)
```



Geoms — geometric objects

- Geometric objects control the way the data are represented on the plot
- `geom = "point"` — scatterplot
- `geom = "smooth"` — fits a smooth to the data and draws the smooth and its standard error
- `geom = "boxplot"` — box plots
- `geom = "line"` and `geom = "path"` produce line plots. `"line"` produces lines from left to right, whilst `"path"` can go in any direction
- `geom = "histogram"` — histograms
- `geom = "freqpoly"` — frequency polygons
- `geom = "density"` — density plots
- `geom = "bar"` — bar plots

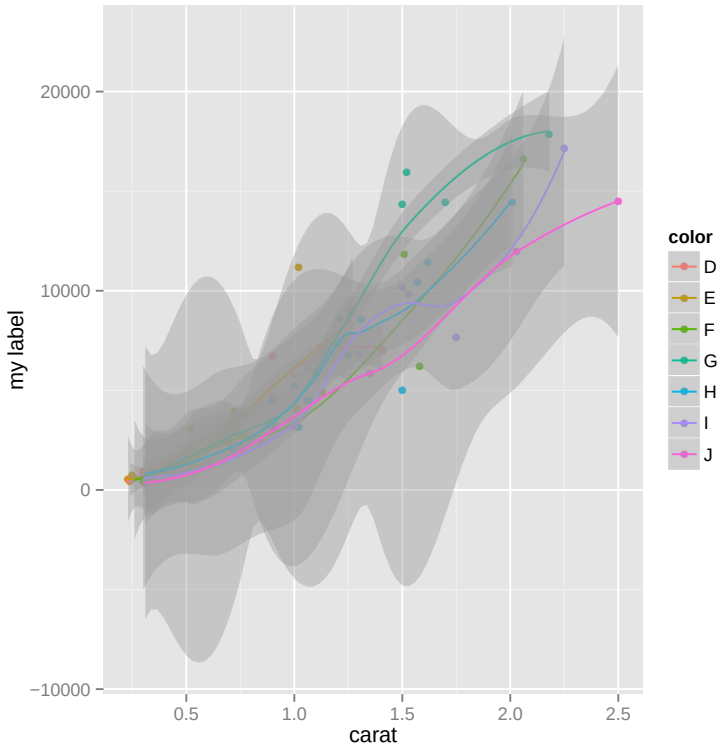
Experimenting with types of geoms

```
> qplot(color, price/carat, data = diamonds, geom = "boxplot")
> qplot(cut, price/carat, data = diamonds, geom = "boxplot")
> qplot(carat, data = diamonds, geom = "density", colour = color)
> qplot(carat, data = diamonds, geom = "histogram", fill = color)
```

Going further with ggplot2

- `qplot()` allows us to quickly produce plots
- The real power comes from working with the `ggplot()`
- Now we need to specify an aesthetic which specifies the data and appearance

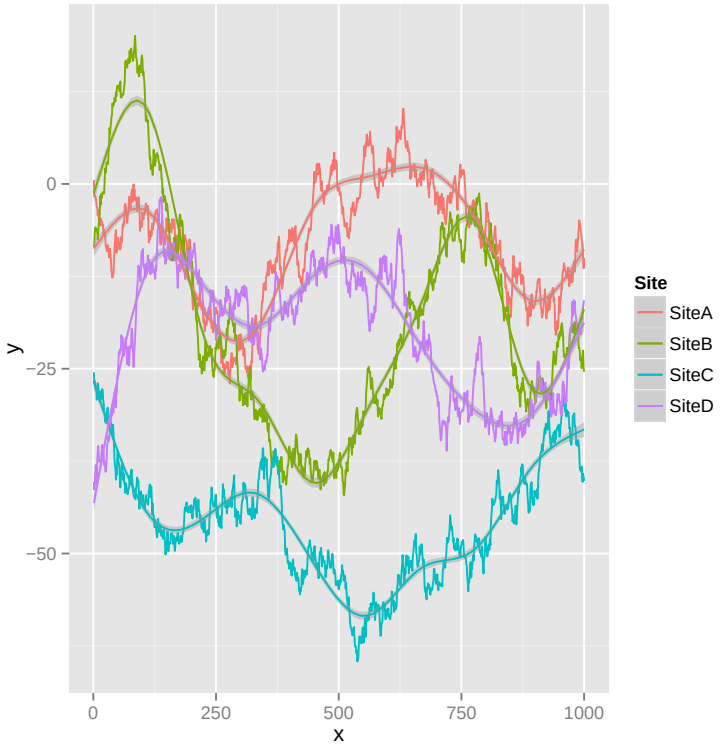
```
> p <- ggplot(dsmall, aes(x = carat, y = price, colour = color))
> p + geom_point() + geom_smooth() + ylab("my label")
```



Going further with ggplot2

- Return to our multiple time series

```
> p2 <- ggplot(dat2, aes(x = x, y = y, colour = Site))
> p2 <- p2 + geom_line() + geom_smooth()
```



Facetting with ggplot2 — facet_wrap()

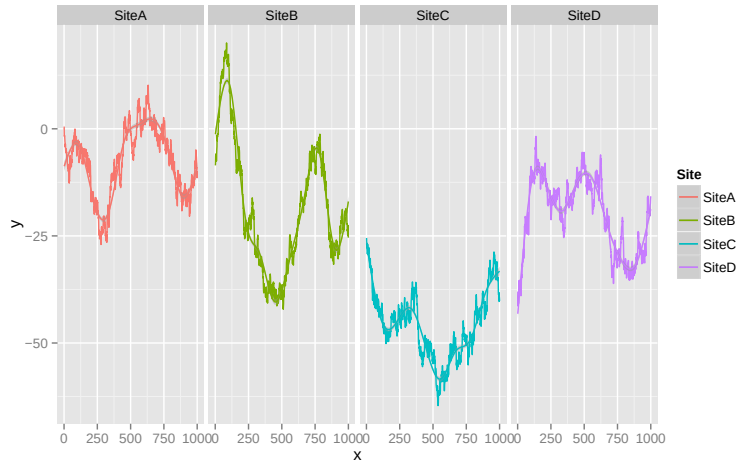
- Facetting is the name used in ggplot2 for the Trellis plots of Lattice
- Two types of faceting:
  - `facet_wrap()` — wraps facets into a tabular arrangement
  - `facet_grid()` — arranges facets by 2 categorical variables
- `facet_wrap()` takes a one-sided formula
- `facet_grid()` takes a two-sided formula

```
> p2 <- ggplot(dat2, aes(x = x, y = y, colou
> p2 <- p2 + geom_line() + geom_smooth()
> p3 <- p2 + facet_wrap( ~ Site, ncol = 2)
> p3
```

Facetting with ggplot2 — facet_grid()

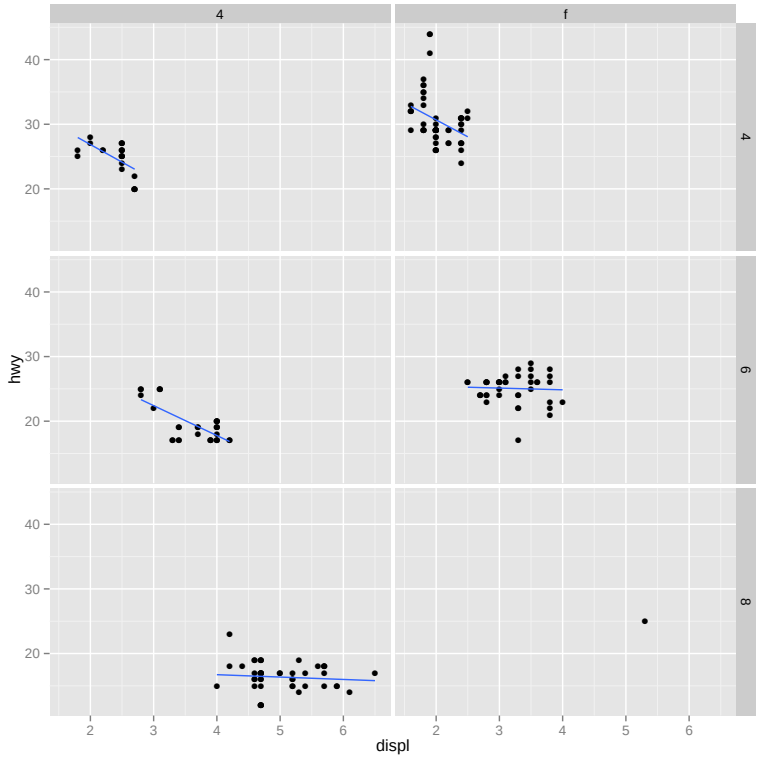
- Facetting is the name used in ggplot2 for the Trellis plots of Lattice
- Two types of faceting:
  - `facet_wrap()` — wraps facets into a tabular arrangement
  - `facet_grid()` — arranges facets by 2 categorical variables
- `facet_wrap()` takes a one-sided formula
- `facet_grid()` takes a two-sided formula

```
> p4 <- ggplot(dat2, aes(x = x, y = y,
+                        colour = Site))
> p4 <- p4 + geom_line() + geom_smooth()
> p4 <- p4 + facet_grid(. ~ Site)
> p4
```



Facetting with ggplot2 — facet_grid()

- Facetting is the name used in ggplot2 for the Trellis plots of Lattice
- Two types of faceting:
  - `facet_wrap()` — wraps facets into a tabular arrangement
  - `facet_grid()` — arranges facets by 2 categorical variables
- `facet_wrap()` takes a one-sided formula
- `facet_grid()` takes a two-sided formula




```
> data(mpg)
> mpg2 <- subset(mpg, cyl != 5 &
+                    drv %in% c("4", "f"))
> p5 <- ggplot(mpg2, aes(displ, hwy)) + geom_point()
> p5 <- p5 + geom_smooth(method = "lm", se = FALSE)
> p5 <- p5 + facet_grid(cyl ~ drv)
```

Something extra — play with the mpg2 data set

```
> head(mpg2)
```

	manufacturer	model	displ	year	cyl	trans	drv	cty	hwy	fl	class
1	audi	a4	1.8	1999	4	auto(l5)	f	18	29	p	compact
2	audi	a4	1.8	1999	4	manual(m5)	f	21	29	p	compact
3	audi	a4	2.0	2008	4	manual(m6)	f	20	31	p	compact
4	audi	a4	2.0	2008	4	auto(av)	f	21	30	p	compact
5	audi	a4	2.8	1999	6	auto(l5)	f	16	26	p	compact
6	audi	a4	2.8	1999	6	manual(m5)	f	18	26	p	compact

R Packages

```
> install.packages("vegan")
> update.packages()
> library("vegan")
> require("vegan")
```

- CRAN contains hundreds of packages of user-contributed code that you can install from an R session
- Package installation via function `install.packages()`
- Packages can be updated via function `updates.packages()`
- When installing or updating for the first time in a session, R will prompt you to choose a mirror to download from
- Once a package is installed you need to load it ready for use
- Load a package from your library using `library()` or `require()`
- Windows and MacOS have menu items to assist with these operations

R Package Management

- It is useful to create your own library for downloaded packages
- This library will not be overwritten when you install a new version of R
- To set a directory you have write permissions on as your user library, create a file named `".Renviron"` in your home directory
  - On Windows this is usually
    `C:\Documents and Settings\username\My Documents`
  - On Linux it is `/home/user/`
- To set your user library to stated directory, add following to your `".Renviron"`
  - On Windows if installed R to `C:\R` add: `R_LIBS=C:/R/myRlib`
  - On Linux, create directory `/home/user/R/libs` say and then add:
    `R_LIBS=/home/user/R/libs`