Plotting Complex Figures Using R

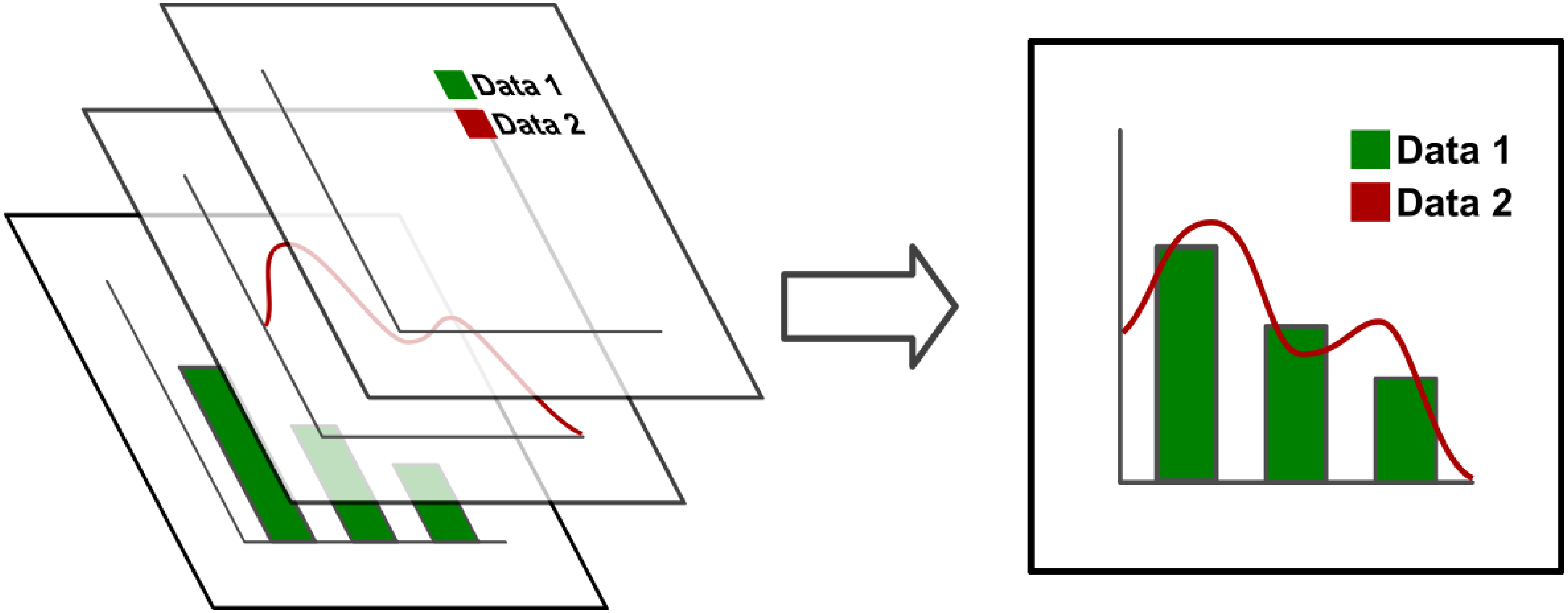
Simon Andrews

simon.andrews@babraham.ac.uk

v2017-11

Babraham Bioinformatics

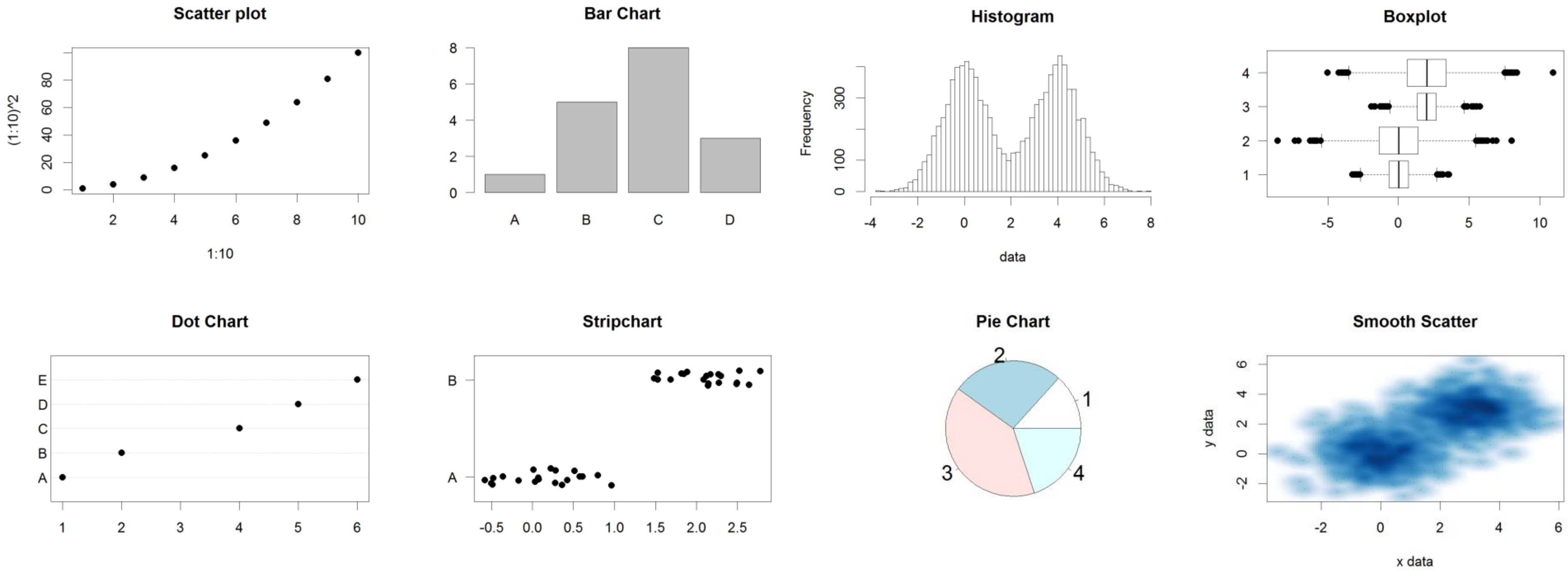
The R Painters Model



- Plot area
- Base plot
- Overlays

Babraham Bioinformatics

Core Graph Types

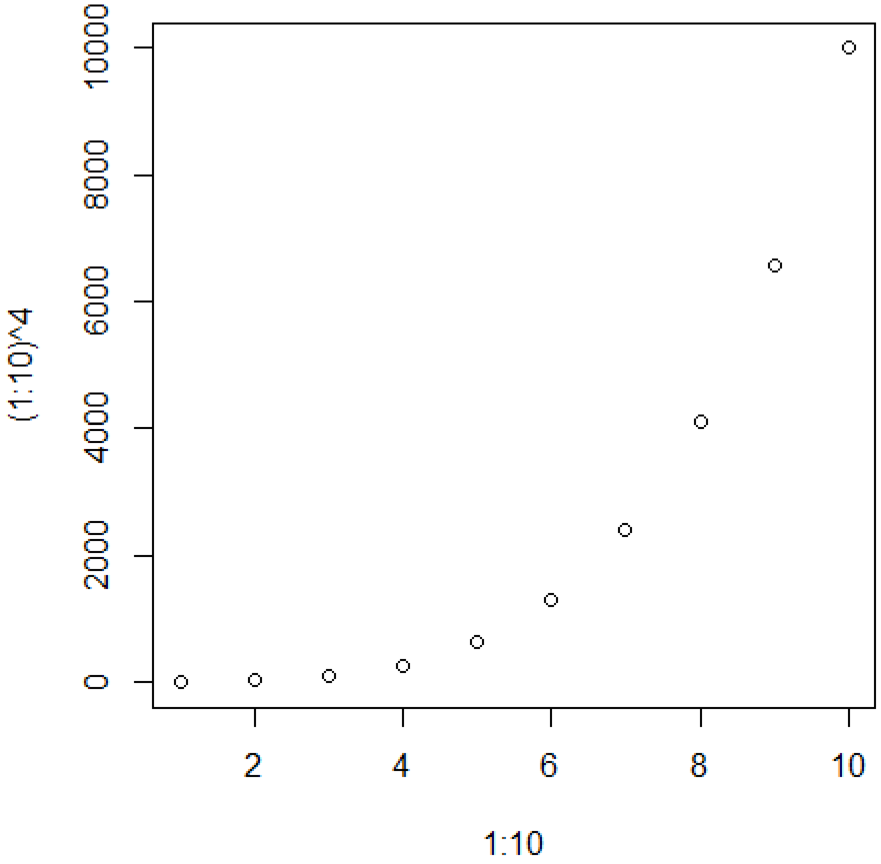


- Local options to change a specific plot
- Global options to affect all graphs

Babraham Bioinformatics

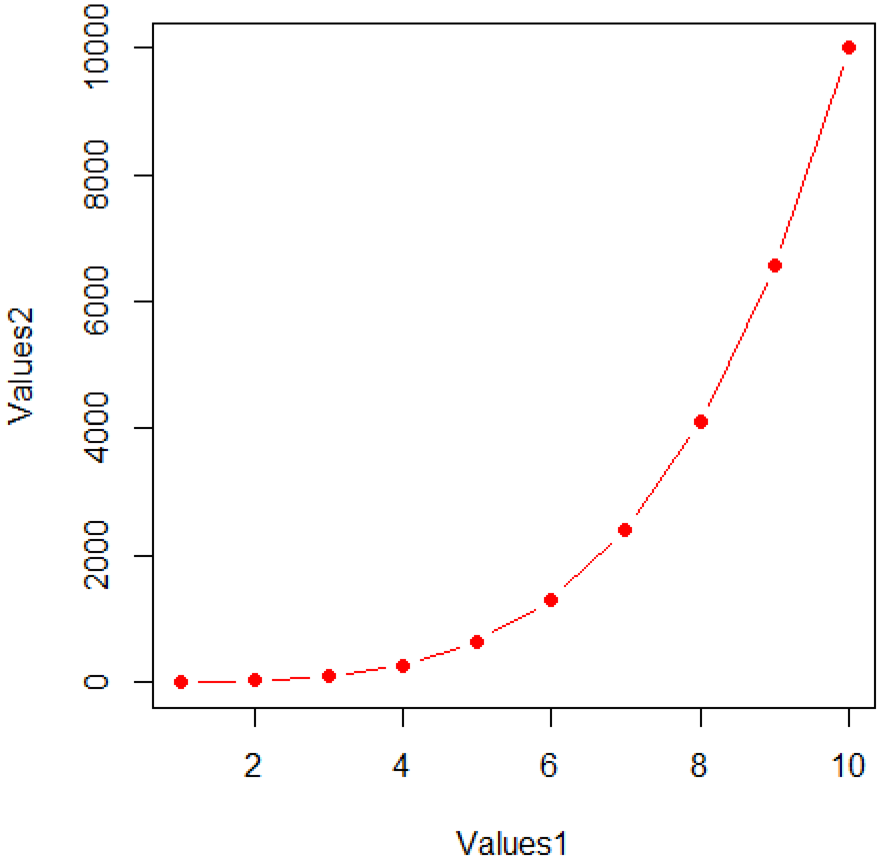
Figures are configured based on the options passed to them

```
plot(
  1:10,(1:10) ^ 4
  )
```



Figures are configured based on the options passed to them

```
plot(
  1:10,(1:10) ^ 4,
  pch=19,
  type="b",
  xlab="Values1",
  ylab="Values2",
  col="red"
  )
```

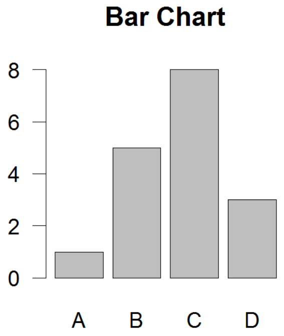


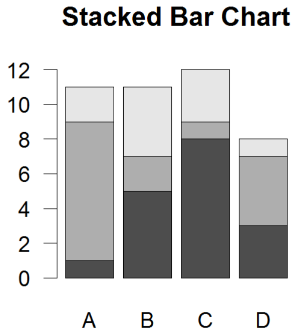
Babraham Bioinformatics

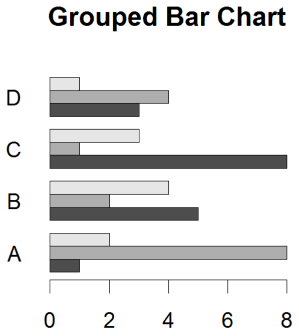
Some options are common to many plot types

- Axis scales
  - `xlim c(min,max)`
  - `ylim c(min,max)`
- Axis labels
  - `xlab(text)`
  - `ylab(text)`
- Plot titles
  - `main(text)`
  - `sub(text)`
- Plot characters
  - `pch(number)`
  - `cex(number)`

Some options are specific to one graph type (eg barplot)


Bar Chart

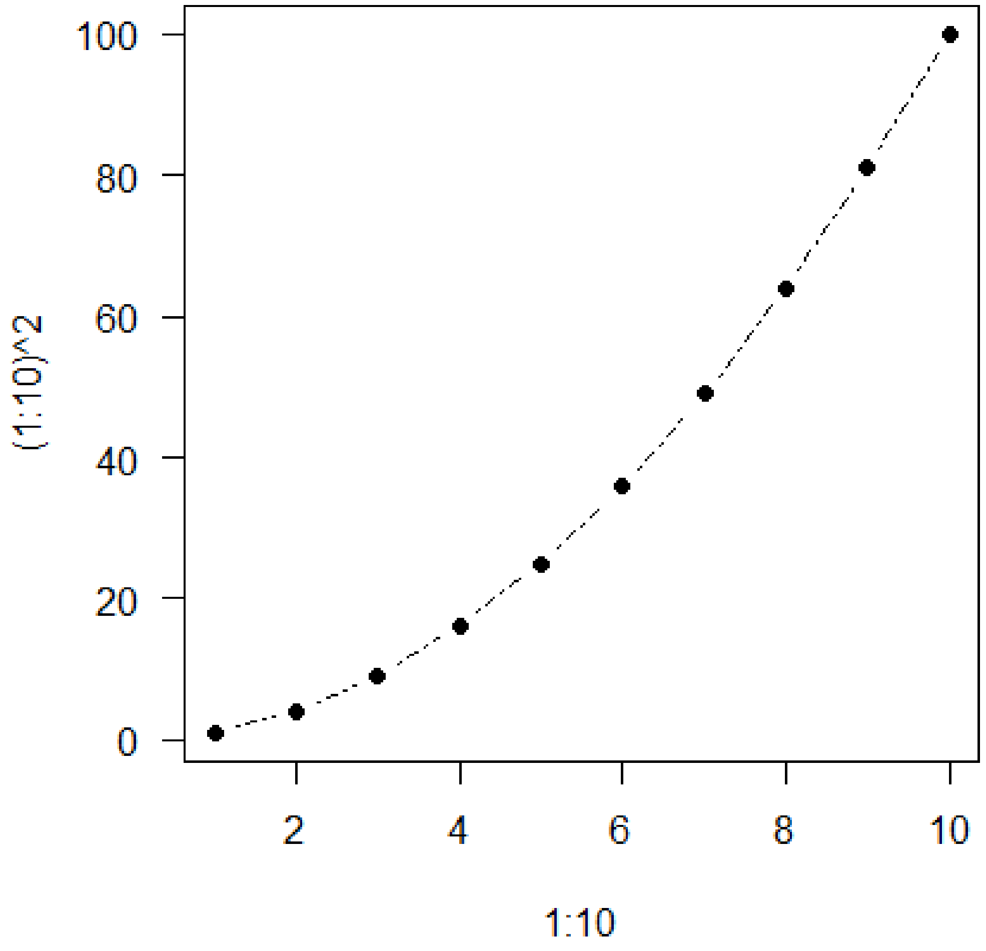

Stacked Bar Chart


Grouped Bar Chart

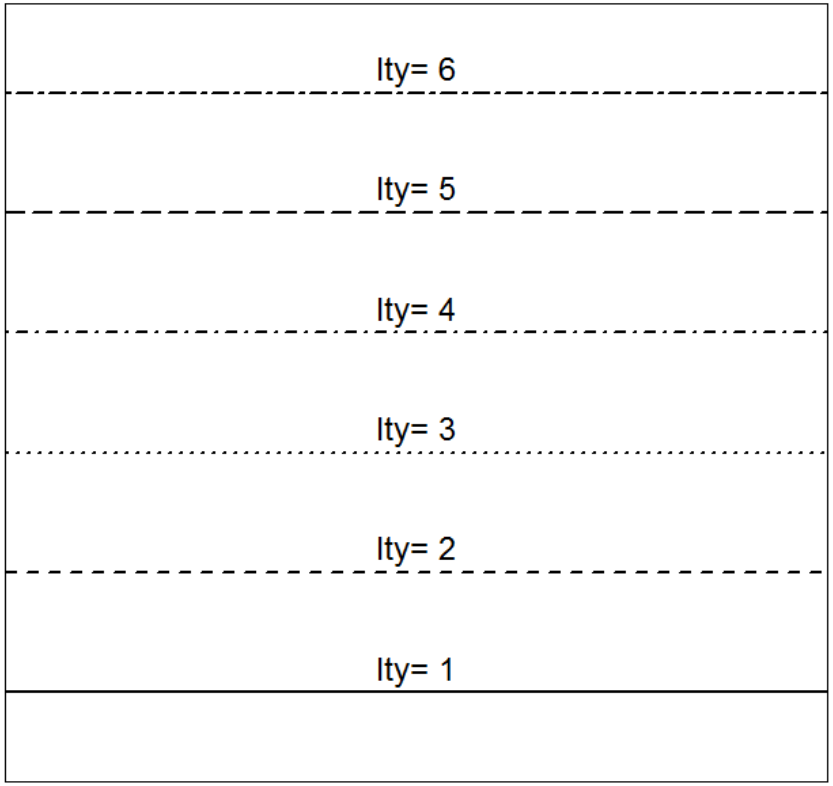
- Options:
  - `names.arg` Bar labels (if not from data)
  - `horiz=TRUE` Plot horizontally
  - `beside=TRUE` Plot multiple series as a group rather than stacked

Some options take 'magic' numbers

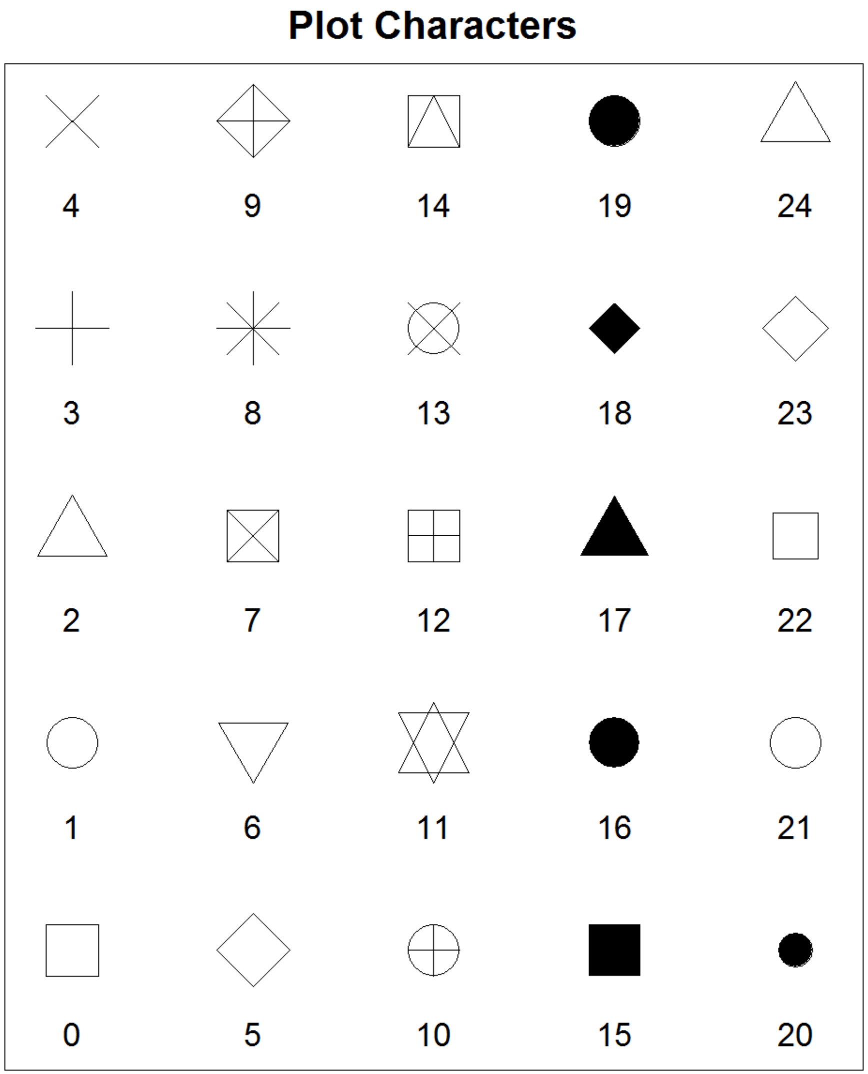
```
plot(
  1:10,
  (1:10)^2,
  type="b",
  lty=2,
  pch=19
)
```



Line types



Plot Characters



Controlling plot area options with `par`

Par

- The `par` function controls global parameters affecting all plots in the current plot area

- Changes affect all subsequent plots

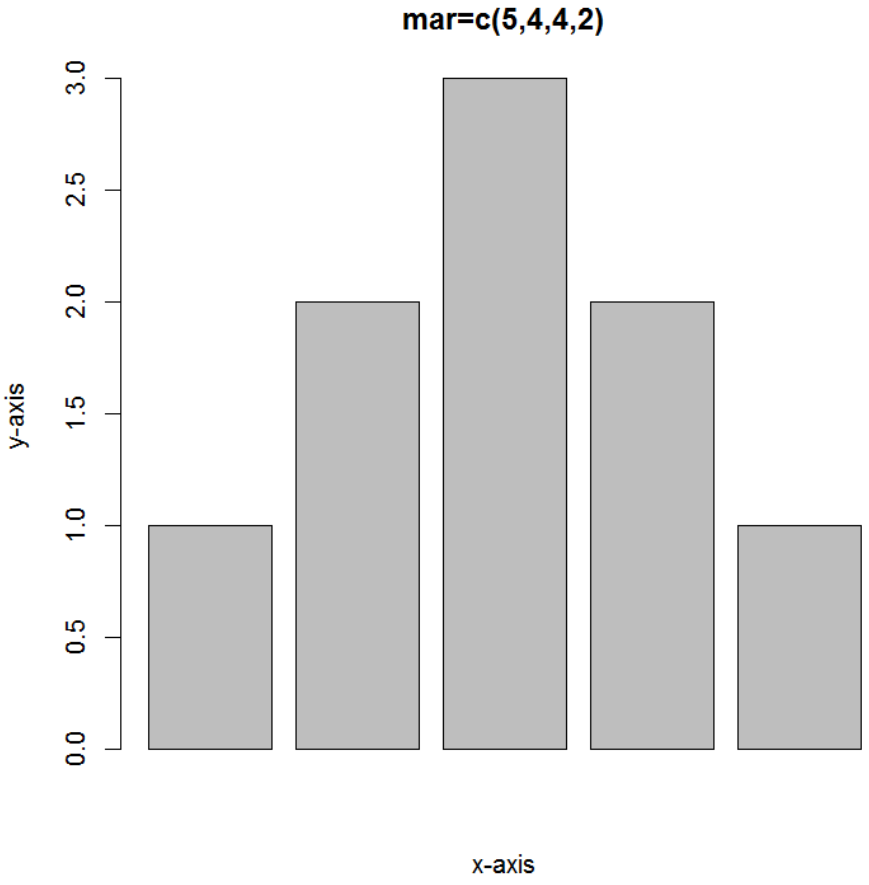
- Many par options can also be passed to individual plots

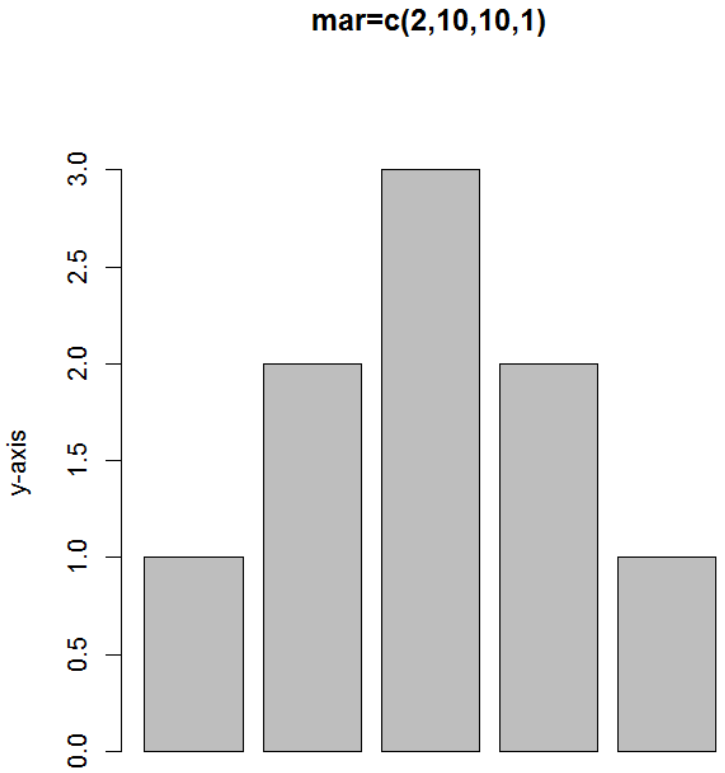
Par examples

- Reading current value
  - `par()$cex`
- Setting a value
  - `par(cex=1.5) -> old.par`
- Restoring a value
  - `par(old.par)`
  - `dev.off()`

Par options

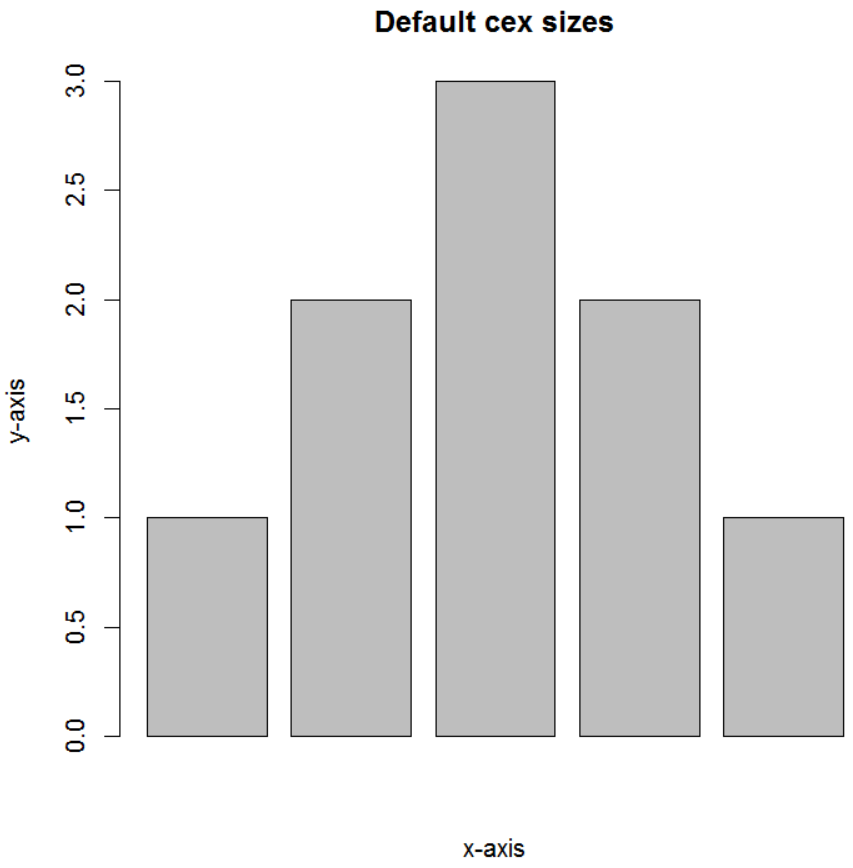
- Margins
  - `mai` (set margins in inches)
  - `mar` (set margins in number of lines)
  - `mex` (set lines per inch)
  - 4 element vector (bottom, left, top, right)
- Warning
  - `Error in plot.new() : figure margins too large`

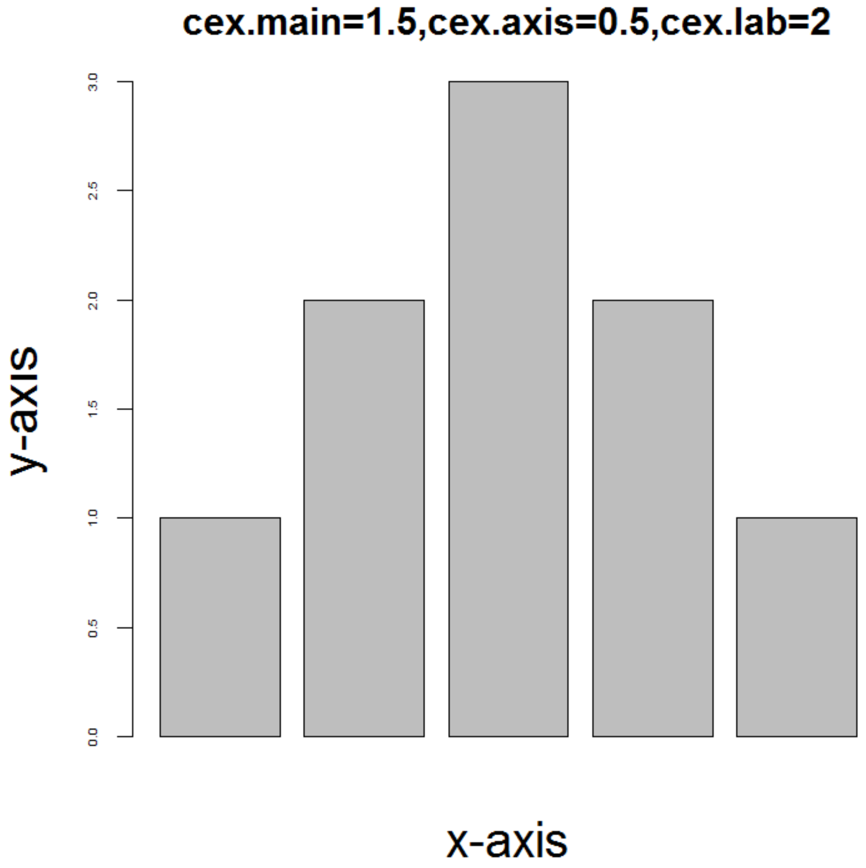

mar=c(5,4,4,2)
y-axis
0.0
0.5
1.0
1.5
2.0
2.5
3.0
x-axis


mar=c(2,10,10,1)
y-axis
0.0
0.5
1.0
1.5
2.0
2.5
3.0

Par options

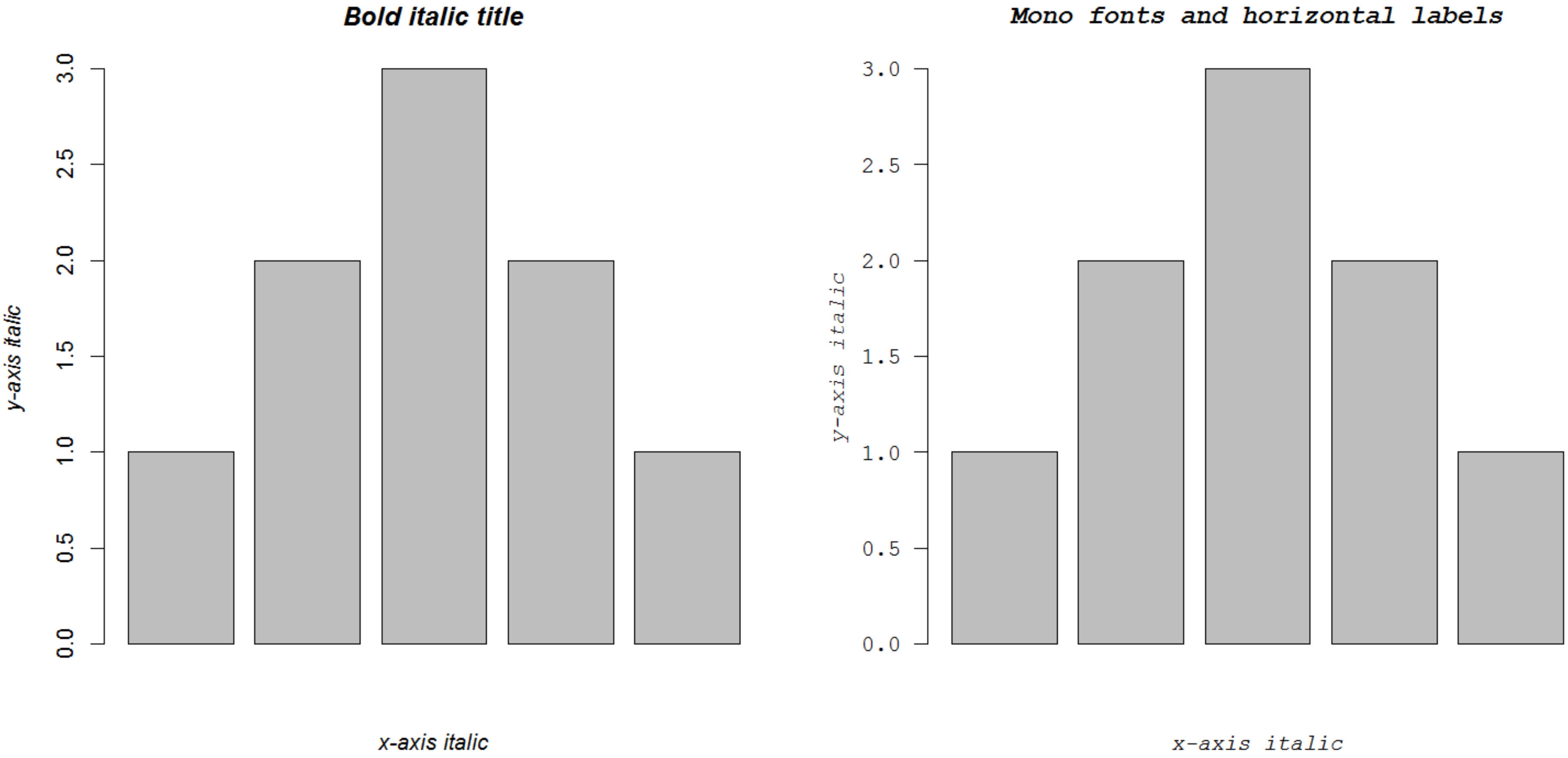
- Fonts and labels
  - `cex` – global char expansion
    - `cex.axis`
    - `cex.lab`
    - `cex.main`
    - `cex.sub`


Default cex sizes
y-axis
x-axis
0.0
0.5
1.0
1.5
2.0
2.5
3.0


cex.main=1.5,cex.axis=0.5,cex.lab=2
y-axis
x-axis
0.0
0.5
1.0
1.5
2.0
2.5
3.0

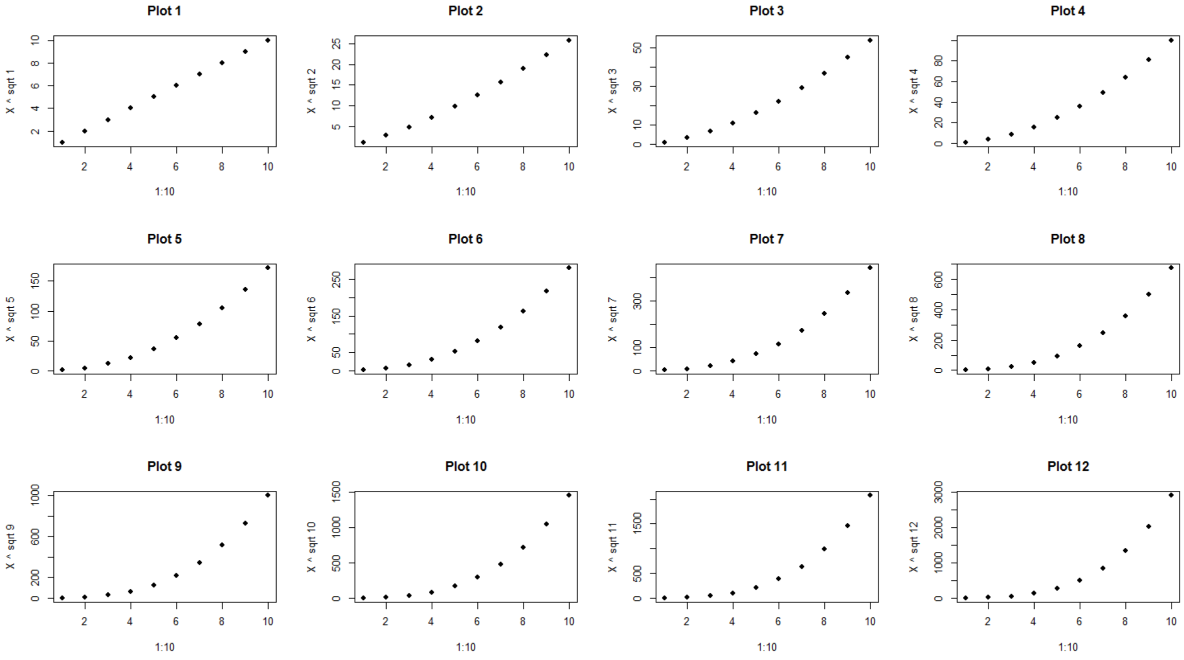
Par options

- Font style
  - `font` (`font.axis`,`font.main`,`font.sub`,`font.lab`)
    - 1 = Plain text
    - 2 = Bold text
    - 3 = Italic text
    - 4 = Bold italic text
  - `las` (label orientation)
    - 0 = Parallel to axis
    - 1 = Horizontal
    - 2 = Perpendicular
    - 3 = Vertical


Bold italic title
3.0
2.5
2.0
1.5
1.0
0.5
0.0
y-axis italic
x-axis italic
Mono fonts and horizontal labels
3.0
2.5
2.0
1.5
1.0
0.5
0.0
y-axis italic
x-axis italic

Par options

- Multi-panel
  - `mfrow(rows,cols)`
  - Not supported by some packages



Exercise 1

Using Colour

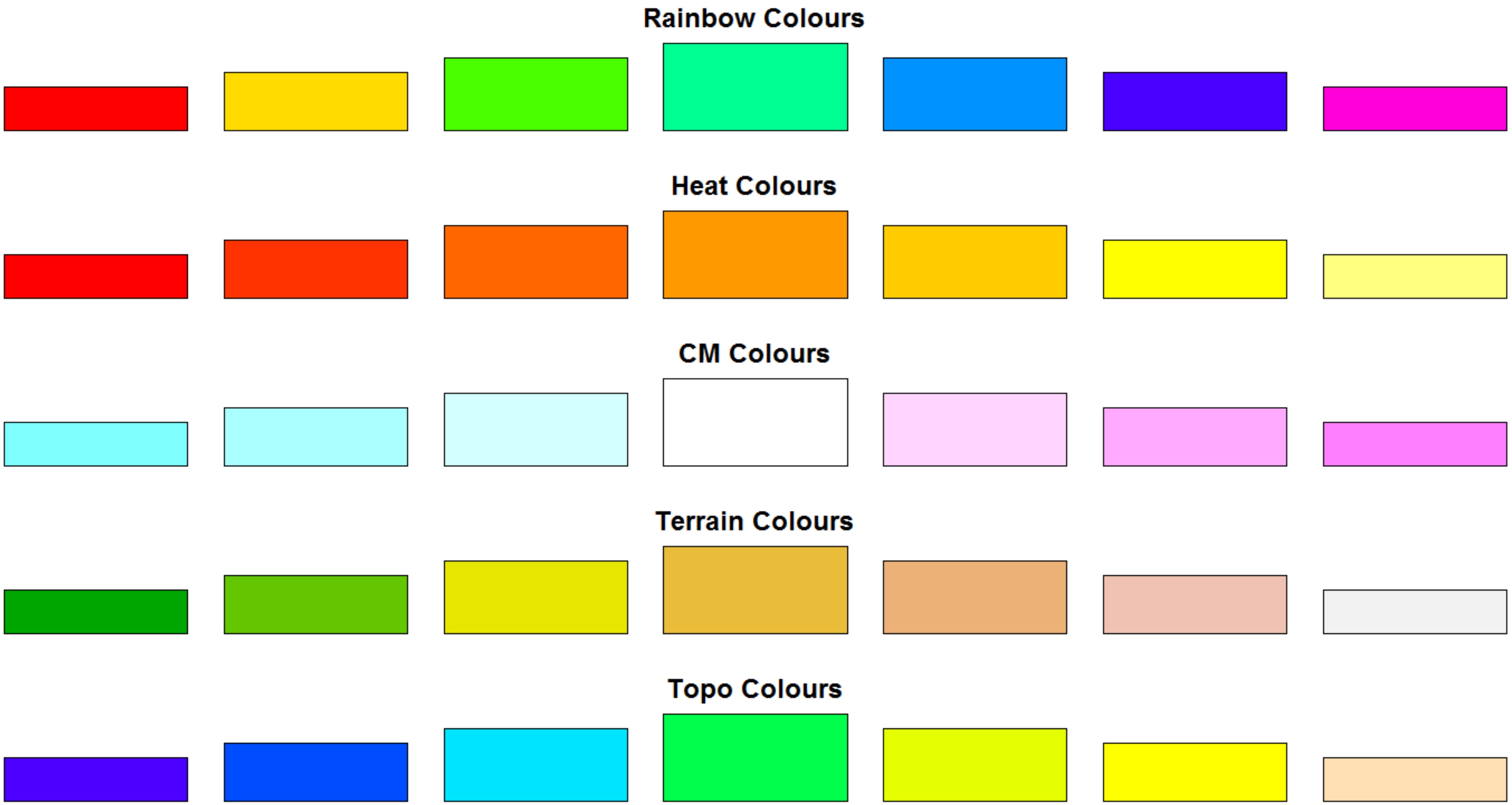
Babraham Bioinformatics

Specifying colours

- Hexadecimal strings
  - `#FF0000` (red)
  - `#0000FF` (blue)
  - `#CC00CC` (purple)
- Controlled names
  - `“red” “green”` etc.
  - `colors()`

Built in colour schemes

- Functions to generate colours
- Pass in number of colours to make
- Functions:
  - `rainbow`
  - `heat.colors`
  - `cm.colors`
  - `terrain.colors`
  - `topo.colors`



Rainbow Colours

Heat Colours

CM Colours

Terrain Colours

Topo Colours

Colour Packages

- Color Brewer
  - Set of pre-defined, optimised palettes
  - `library(RColorBrewer)`
  - `brewer.pal(no colours, palette)`

- ColorRamps
  - Create smooth palettes for ramped colour
  - Generates a function to make actual colour vectors
  - `colorRampPalette(c(“red”,”white”,”blue”))`
  - `colorRampPalette(c(“red”,”white”,”blue”))(5)`

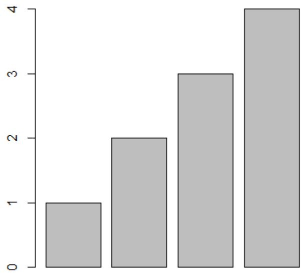
Applying Colour to Plots

- Vector of colours passed to the `col` parameter

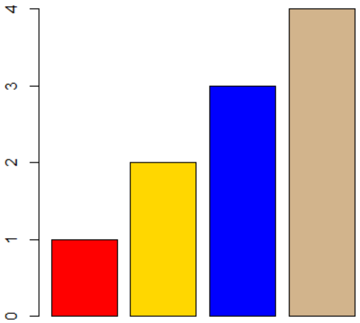
- Vector of factors used to divide the data
  - Colours taken from palette
  - Can read or set using palette function
    - `palette()`
    - `palette(brewer.pal(9,”Set1”)`
    - Ordered by levels of factor vector

Applying Colour to Plots

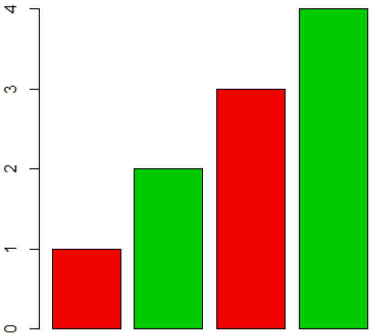
```
barplot(1:4)
```



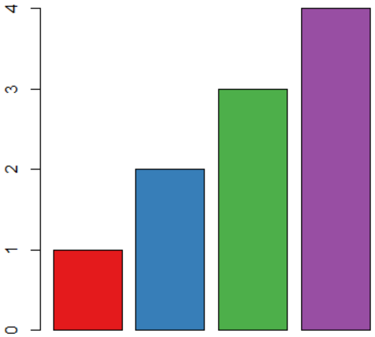
```
barplot(
  1:4,
  col=c("red","gold","blue","tan")
)
```



```
barplot(
  1:4,
  col=c("red2","green3")
)
```



```
library(RColorBrewer)
barplot(
  1:4,
  col=brewer.pal(4,"Set1")
)
```



Applying Colour to Plots

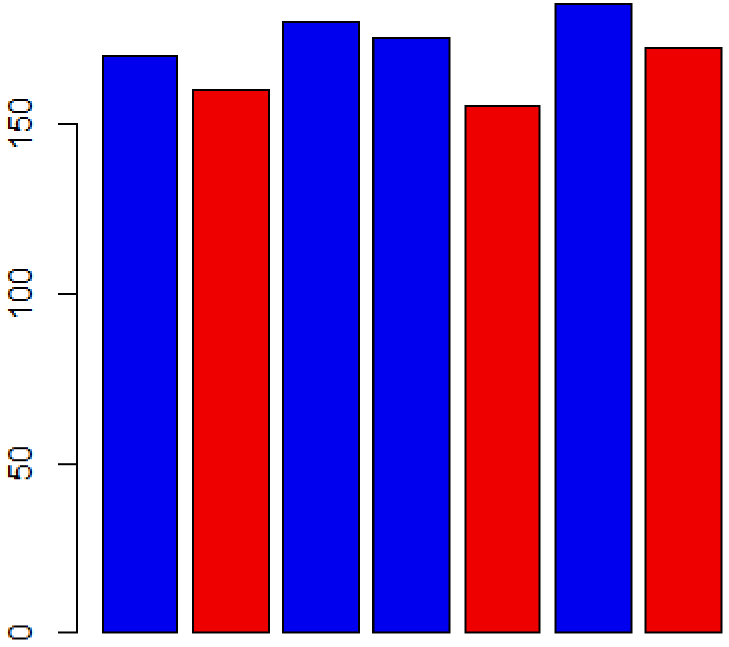
```
> height.data
  height sex
1    170   M
2    160   F
3    180   M
4    175   M
5    155   F
6    185   M
7    172   F
```

```
barplot(height.data$height,col=height.data$sex)
```

```
> palette()
[1] "black" "red"     "green3"  "blue"
[5] "cyan"  "magenta" "yellow"  "gray"
```

```
> levels(height.data$sex)
[1] "F" "M"
```

```
> palette(c("red2","blue2"))
```



Babraham Bioinformatics

Dynamic use of colour

- Colouring by density
  - Pass data and palette to `densCols`
  - Vector of colours returned
- Colouring by value
  - Need function to map values to colours

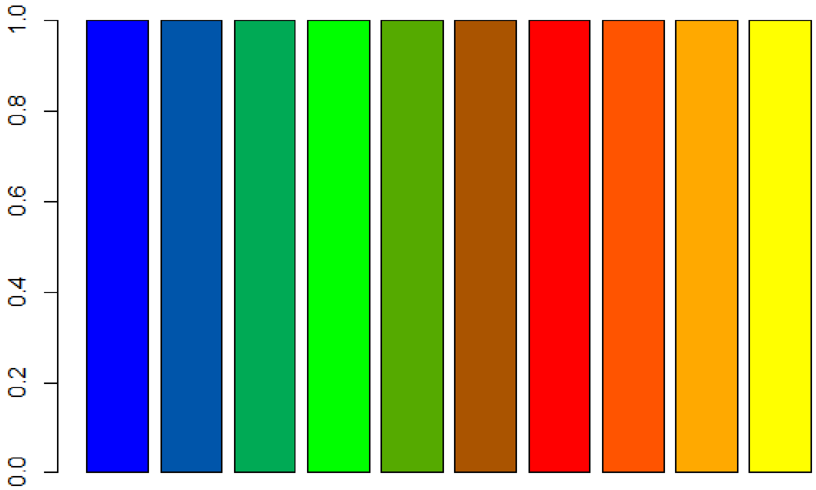
Making colour ramps

```
> colorRampPalette(c("blue","green","red","yellow"))
function (n)
{
    x <- ramp(seq.int(0, 1, length.out = n))
    if (ncol(x) == 4L)
        rgb(x[, 1L], x[, 2L], x[, 3L], x[, 4L], maxColorValue = 255)
    else rgb(x[, 1L], x[, 2L], x[, 3L], maxColorValue = 255)
}
```

```
> colorRampPalette(c("blue","green","red","yellow"))(10)
```

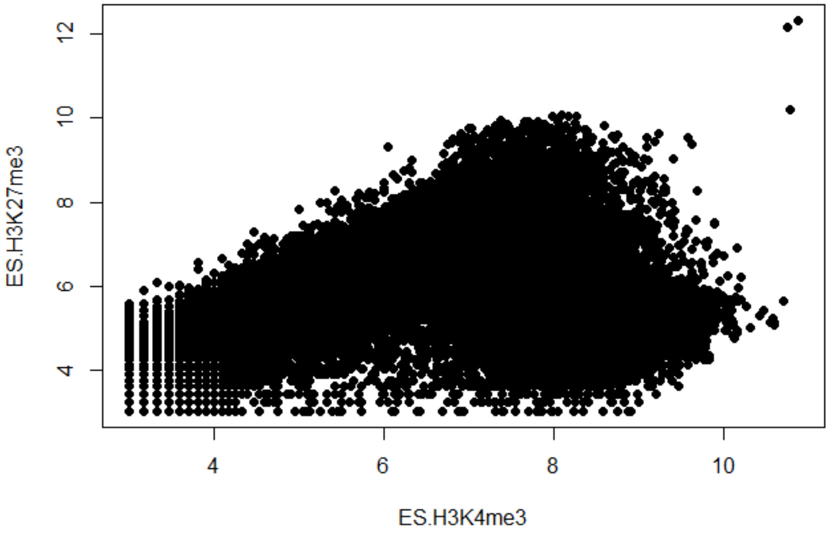
```
 [1] "#0000FF" "#0055AA" "#00AA55" "#00FF00" "#55AA00" "#AA5400" "#FF0000" "#FF5400" "#FFA900" "#FFFF00"
```

```
> barplot(
  rep(1,10),
  col=colorRampPalette(
        c("blue","green","red","yellow")
      )(10)
  )
```

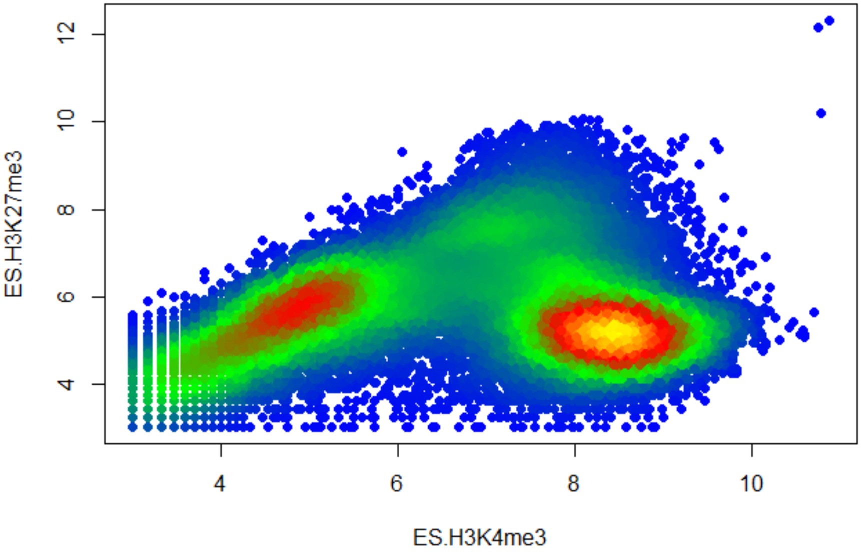


Using colour to plot density

```
plot(lots.of.data, pch=19)
```



```
plot(
  lots.of.data,
  pch=19,
  col=densCols(
    lots.of.data,
    colramp=colorRampPalette(c(
      "blue","green","red","yellow")
      )
    )
)
```



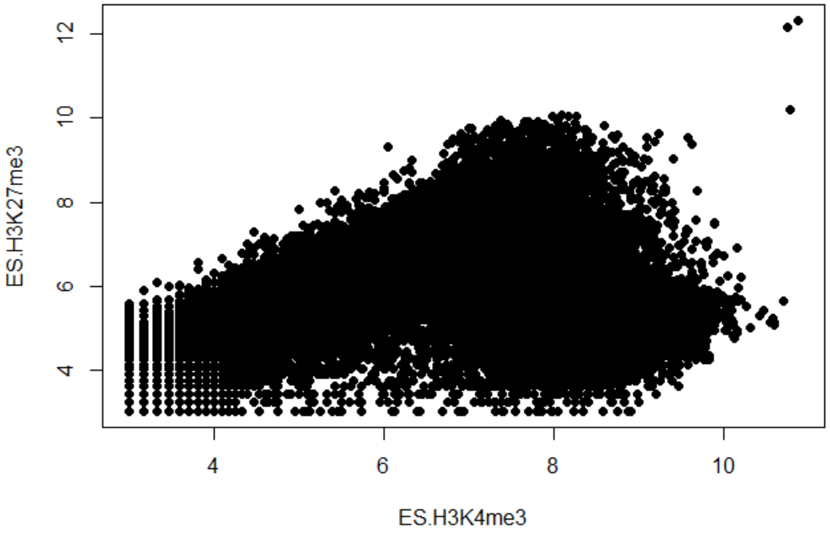
Colour Mapping Function

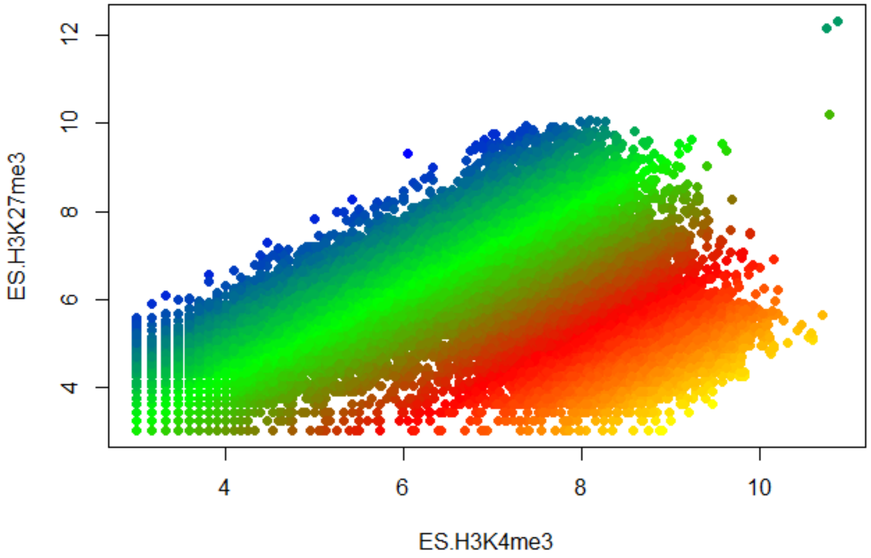
```
map.colours <- function(values,palette) {

  range <- range(values)

  proportion <- (values-range[1])/(range[2]-range[1])
  index <- round((length(palette)-1)*proportion)+1

  return(palette[index])
}
```

Plotting Quantitative Colour

```
plot(lots.of.data, pch=19)
```

```
plot(
  lots.of.data,
  pch=19,
  col=map.colours(
    lots.of.data$K4 - lots.of.data$K27,
    colorRampPalette(c(
      "blue","green","red","yellow")
    )(100)
  )
)
```



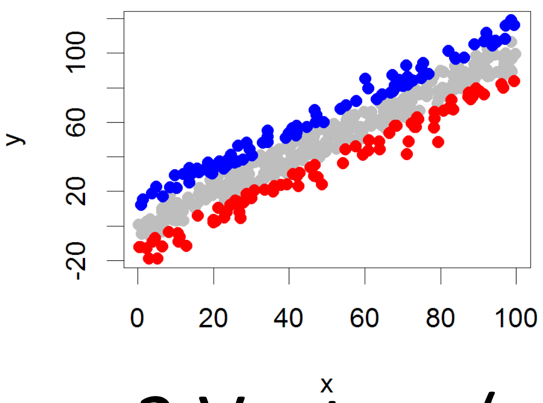


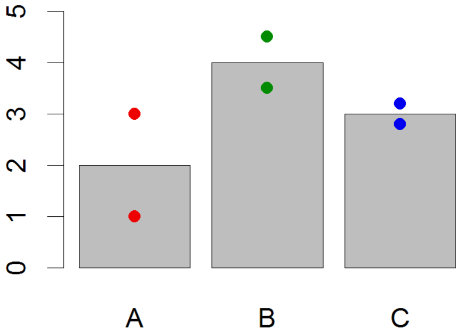
Exercise 2

Babraham Bioinformatics

Plot Overlays

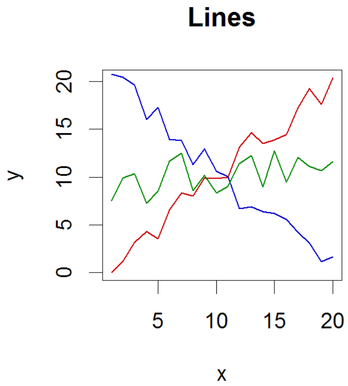
Points

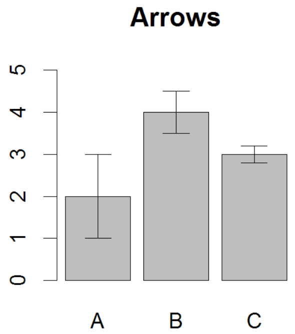


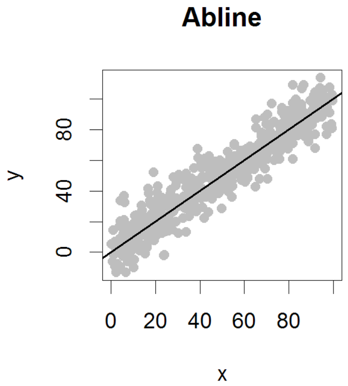


- Input: 2 Vectors (x and y positions)
- Options:
  - `pch`
  - `cex`

Lines / Arrows / Abline



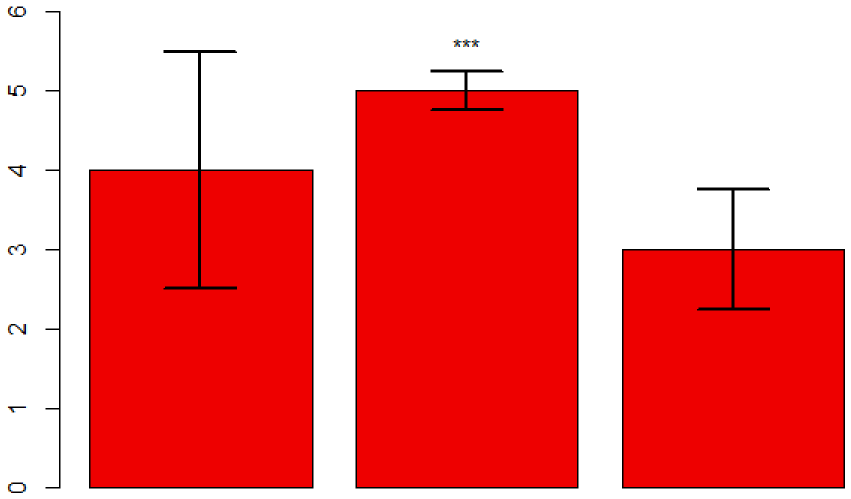




- Input:
  - Lines 2 vectors (x and y)
  - Arrows 4 vectors (x0,y0,x1,y1)
  - Abline Intercept and slope (or correlation object)
- Options:
  - `lwd`
  - `angle` (arrows)

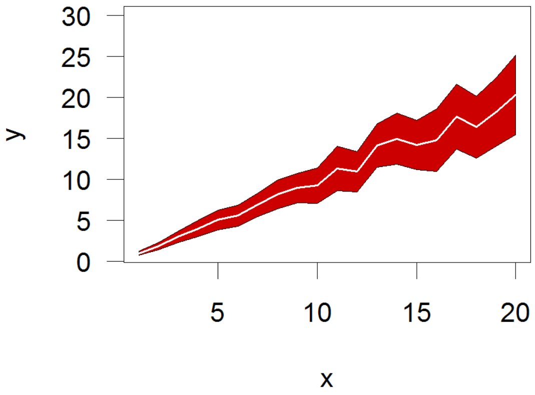
Example multi-layer plot

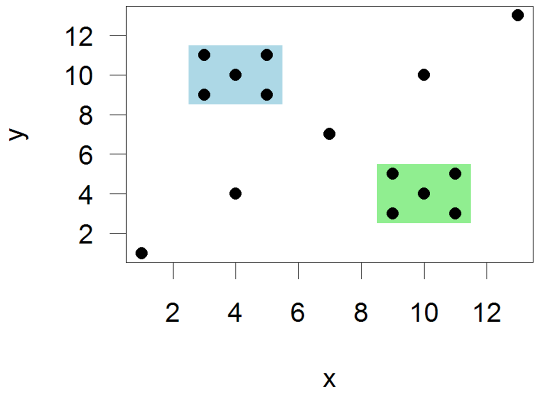
```
barplot(
  error.data$values,
  col="red2",
  ylim=(c(0,6))
) -> bar.centres

arrows(
  x0=bar.centres,
  y0=error.data$values - error.data$sem,
  x1=bar.centres,
  y1=error.data$values + error.data$sem,
  angle=90,
  code = 3,
  lwd=2
)

text(
  bar.centres[2],
  y = error.data$values[2] + error.data$sem[2],
  labels = "***",
  pos=3
)
```



```
> error.data
  values   sem
1      4  1.50
2      5  0.25
3      3  0.75
```

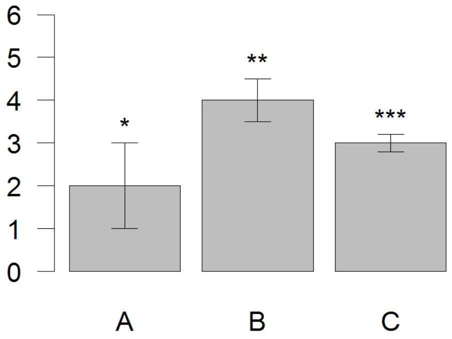
Polygon (shaded areas)

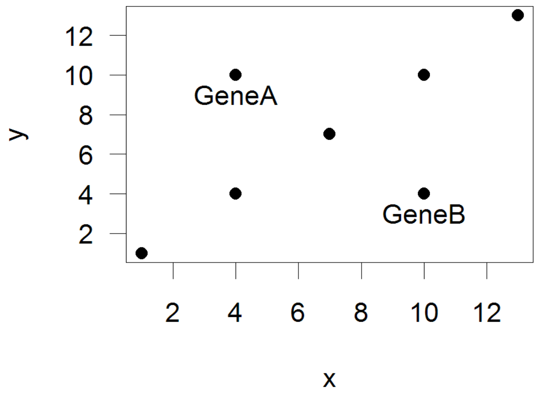




- Input:
  - 2 vectors (x and y) for bounding region
- Options:
  - `col`

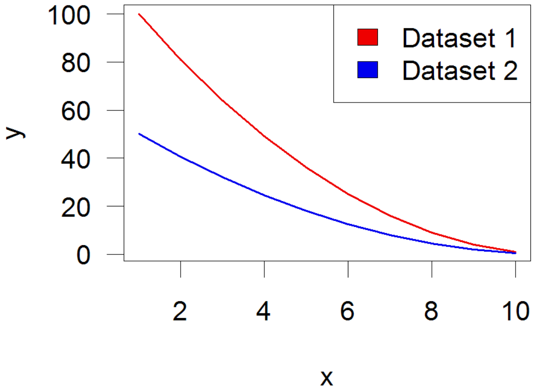
Text (in plot text)

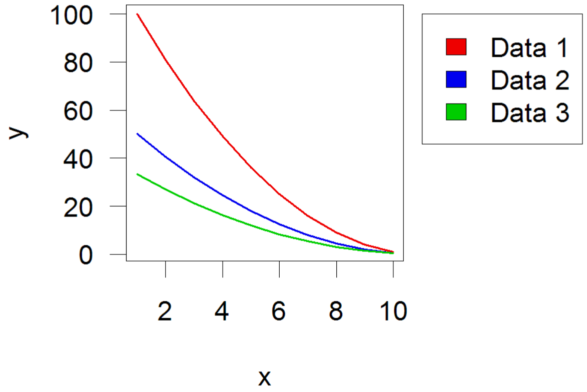




- Input:
  - Text, x, y
- Options:
  - `adj` (x and y offsets)
  - `pos` (auto offset 1=below,2=left,3=above, 4=right)

Legend





- Input:
  - Position (x,y or “topright”,”bottomleft” etc)
  - Text labels
- Options:
  - `fill` (colours for shaded boxes)
  - `xpd=NA` (draw outside plot area)

Exercise 3