Topic 7: Base Plotting

ISOM3390: Business Programming in R

Base Plotting in R

Creating graphs of variables from data and objects created from statistical models is fundamental to gaining actionable knowledge.

The graphics functions that make up the base graphics system are provided in the `graphics` package, which is automatically loaded in a standard installation of R.

```
search()
```

```
##  [1] ".GlobalEnv"        "package:shiny"     "package:stats"
##  [4] "package:graphics"  "package:grDevices" "package:utils"
##  [7] "package:datasets"  "package:methods"   "Autoloads"
## [10] "package:base"
```

Base R has a set of powerful plotting tools. An overview:

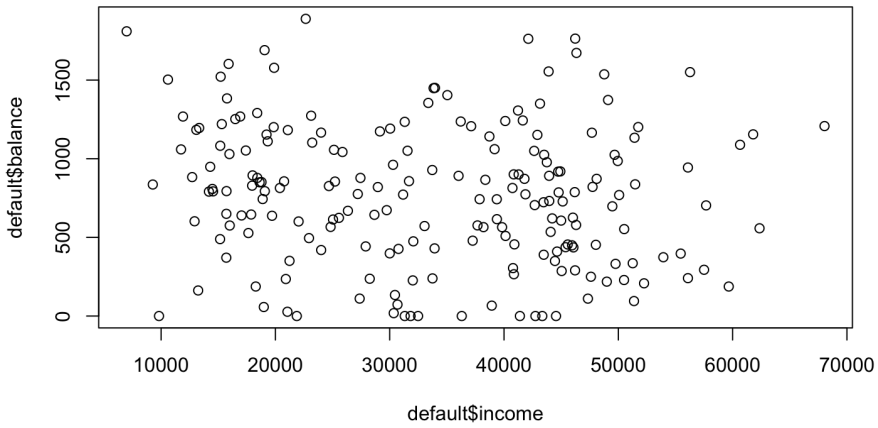
- `plot()`: generic plotting function
- `points()`: add points to an existing plot
- `lines()`, `abline()`: add lines to an existing plot
- `text()`, `legend()`: add text to an existing plot
- `rect()`, `polygon()`: add shapes to an existing plot
- `hist()`: create histograms
- `density()`: estimate density, which can be plotted
- `curve()`: draw a curve, or add to existing plot
- `barplot()`, `boxplot()`: create bar plots and box plots
- ...

`plot()`

Generic, high-level, type of plot depends on class of arguments.

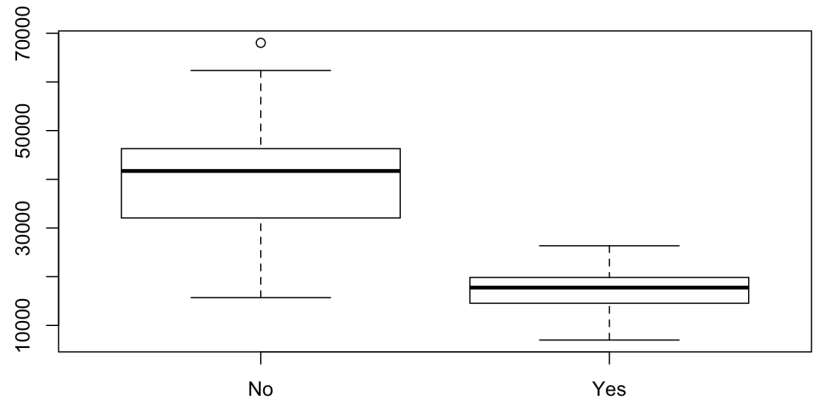
- `plot(x, y)`: a scatterplot if `x` and `y` are two vectors

```
plot(default$income, default$balance)
```



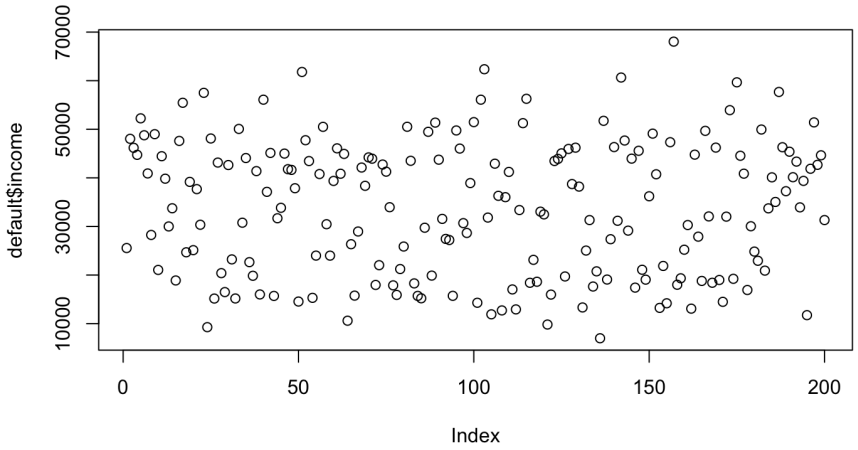
- `plot(f, y)`: a boxplot if `f` is a factor and `y` is a vector

```
plot(default$student, default$income)
```

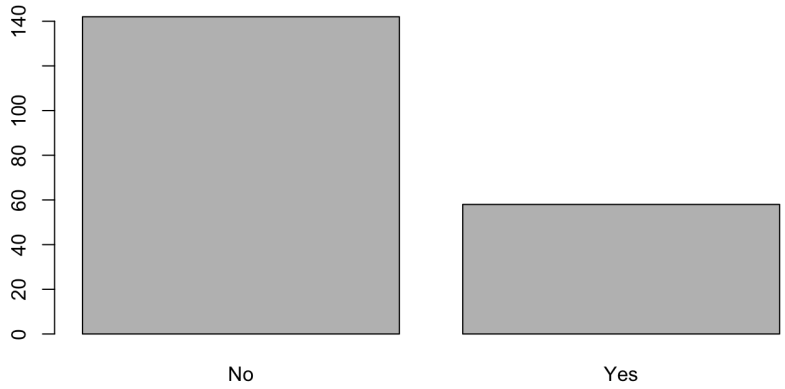


- `plot(x)`: a plot showing the value of `x` at every index if `x` is numeric; Or a barplot of counts for every level if `x` is a factor

```
plot(default$income)
```

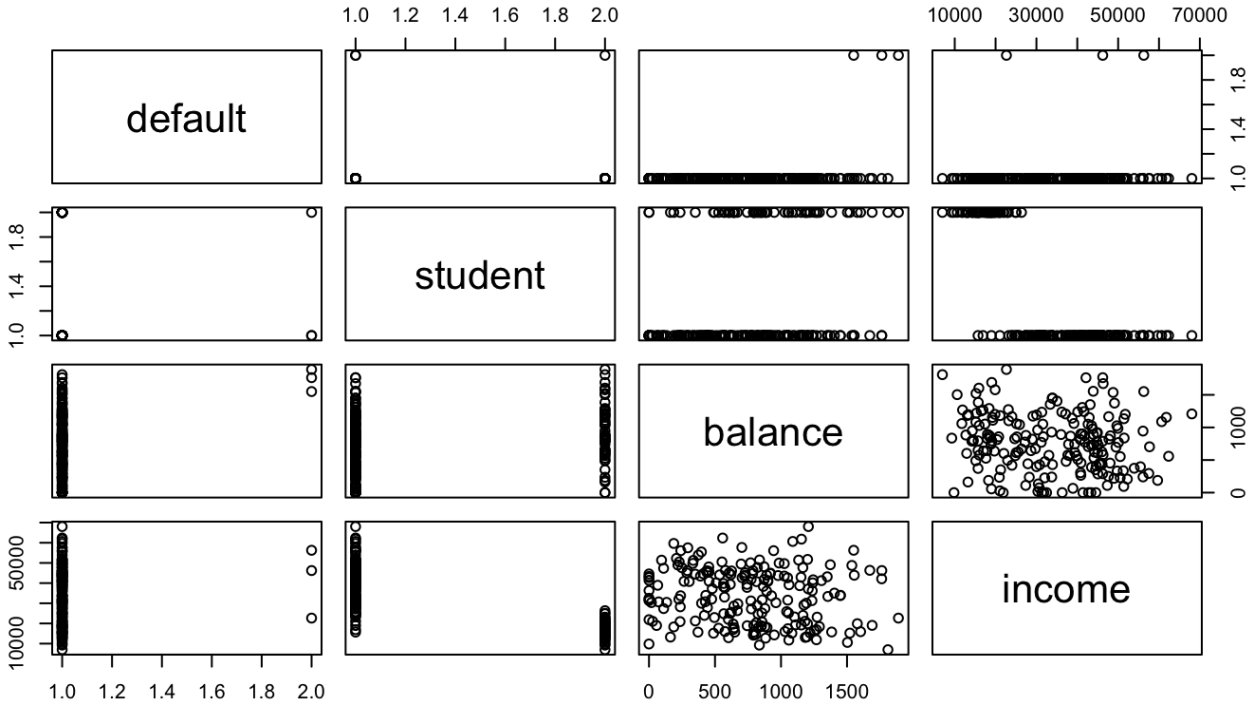



```
plot(default$student)
```



- `plot(data.frame)`: all variables plotted against each other

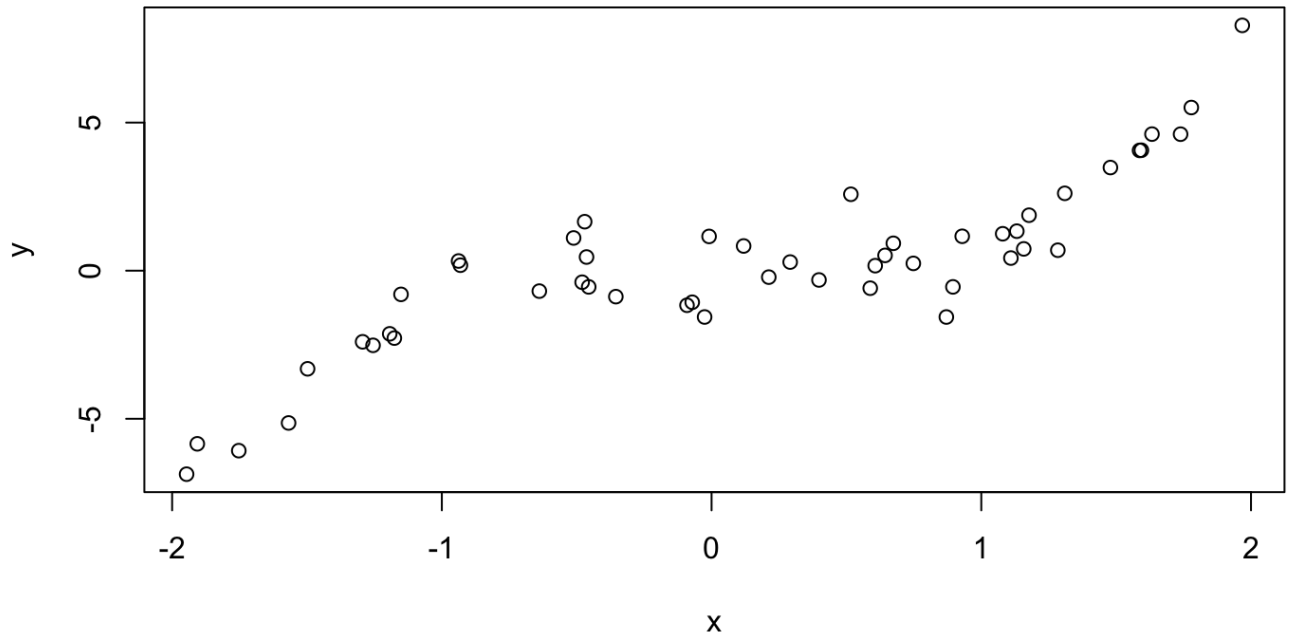
```
plot(default)
```



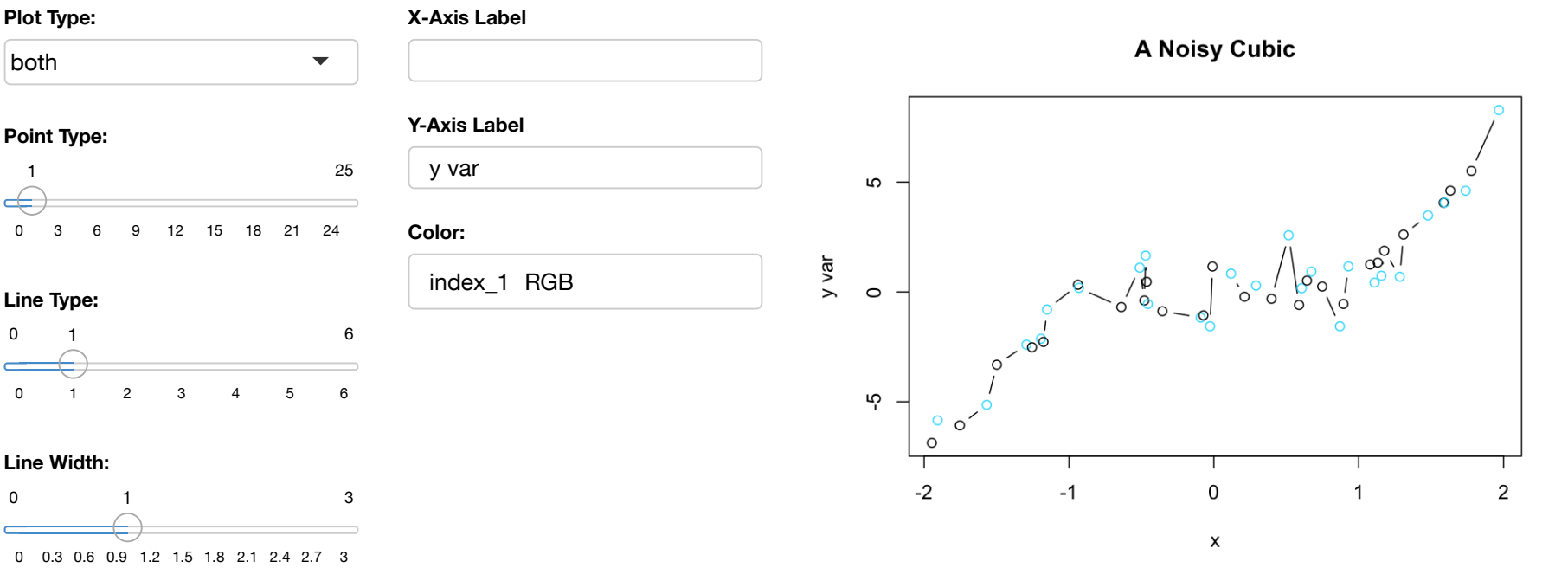
Scatterplots

Calling `plot()` with two vectors of the same length makes a scatter plot of one variable versus another.

```
set.seed(0)  # This makes the result or sampling reproducible
x <- sort(runif(50, min = -2, max = 2))
y <- x^3 + rnorm(50)
plot(x, y)
```



Arguments to `plot()`




```
plot(x, y, type = "b", pch = 1, lty = 1, lwd = 1, col = c("1", "#33DDFF"), main = "A Noisy Cubic", ylab = "y var")
```

Size of Text

Title Size:

0.1 1.6 2

0.1 0.3 0.5 0.7 0.9 1.1 1.3 1.5 1.7 1.9 2

Lab Size:

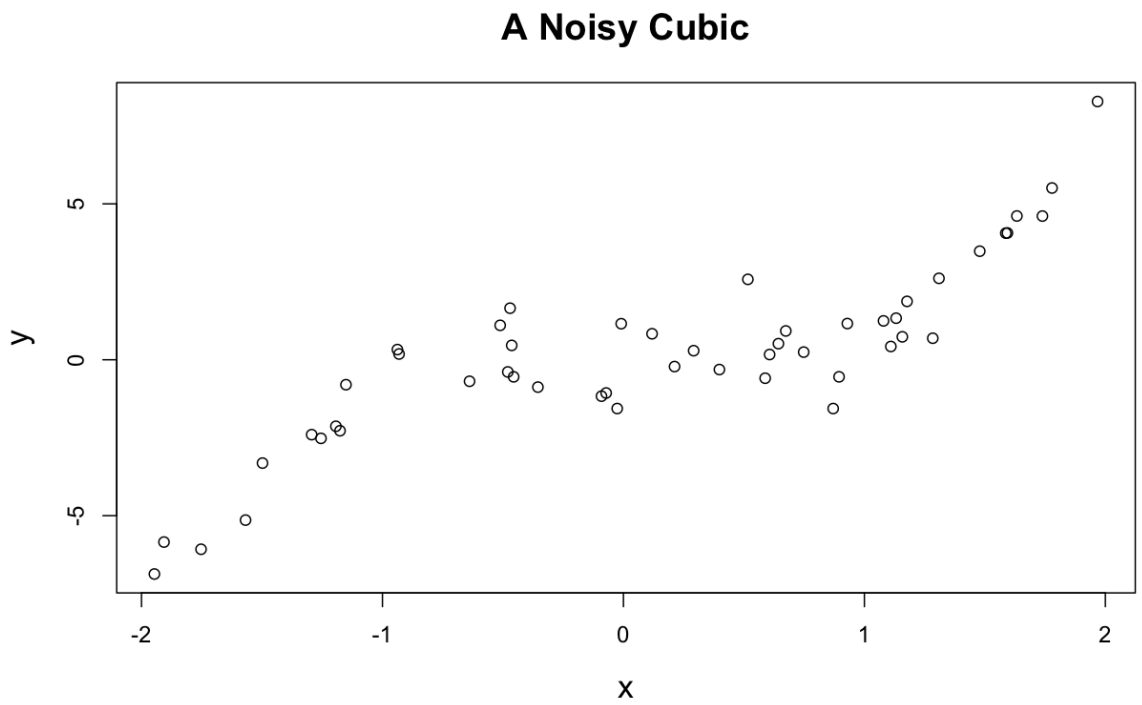
0.1 1.4 2

0.1 0.3 0.5 0.7 0.9 1.1 1.3 1.5 1.7 1.9 2

Axis Size:

0.1 1 2

0.1 0.3 0.5 0.7 0.9 1.1 1.3 1.5 1.7 1.9 2




```
plot(x, y, cex.main = 1.6, cex.lab = 1.4, cex.axis = 1, main = "A Noisy Cubic")
```

Appearance of Axes

Axis Title:

-1 3 4

-1 -0.4 0.2 0.8 1.4 2 2.6 3.2 3.8

Axis Label:

-1 1 3

-1 -0.6 -0.2 0.2 0.6 1 1.4 1.8 2.2 2.6 3

Axis Line:

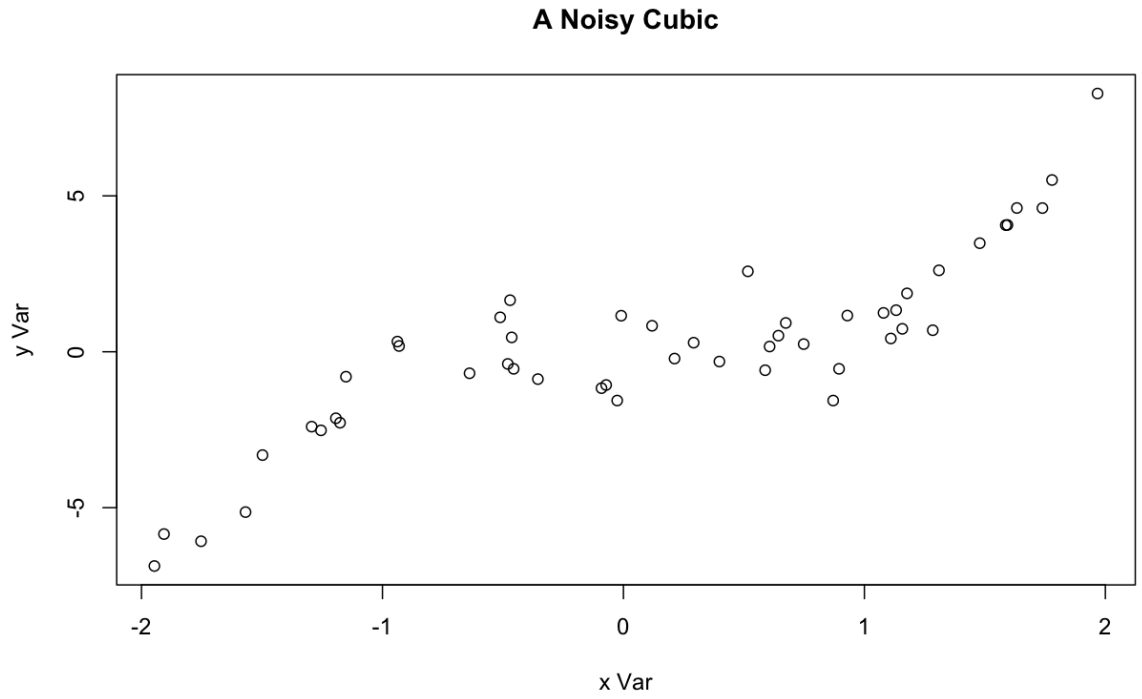
-1 0 2

-1 -0.6 -0.2 0.2 0.6 1 1.4 1.8 2

Tick Mark:

-1 -0.5 1

-1 -0.8 -0.6 -0.4 -0.2 0 0.2 0.4 0.6 0.8 1




```
plot(x, y, main = "A Noisy Cubic", xlab = "x Var", ylab = "y Var", mgp = c(3, 1, 0), tcl = -0.5)
```

Three Distinct Plotting Regions

In R base graphics, the graph area is split into three parts:

- Plot region: area within the axes
- Figure margin region, including axes, axes labels, tick mark labels etc.
- Outer margin region

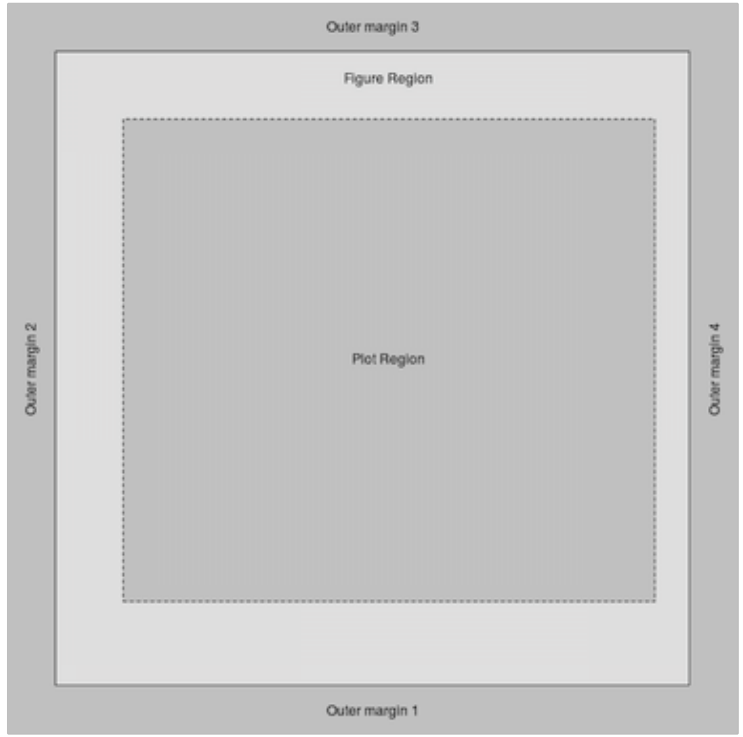


Figure Margins: `par()`'s `mar`

Bottom Margin:

1 5 6

1 1.6 2.2 2.8 3.4 4 4.6 5.2 5.8

Left Margin:

1 4 6

1 1.6 2.2 2.8 3.4 4 4.6 5.2 5.8

Top Margin:

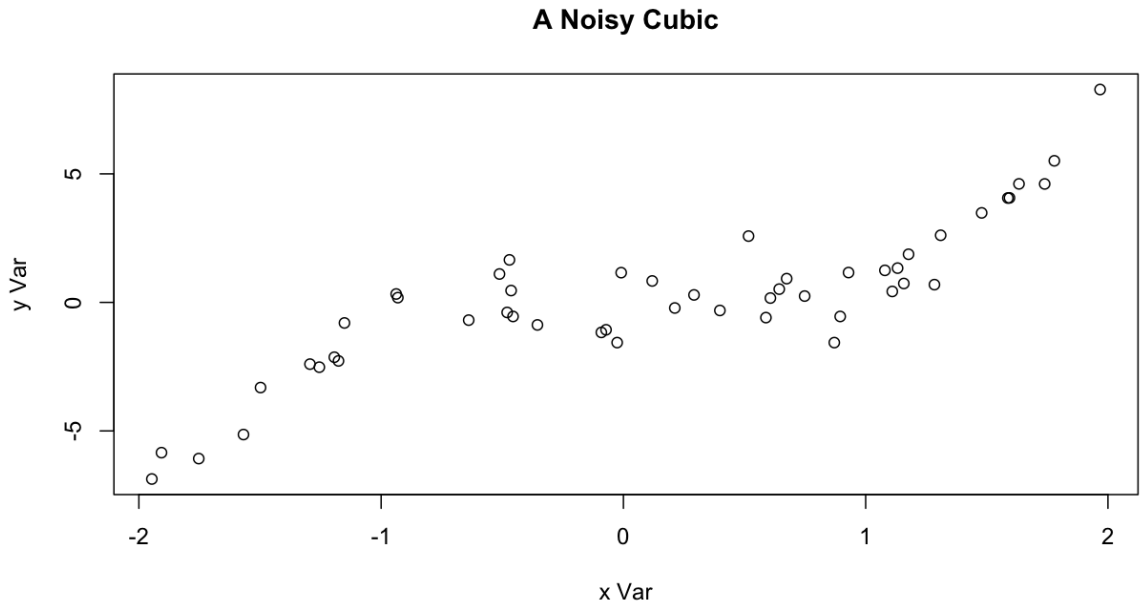
1 4 6

1 1.6 2.2 2.8 3.4 4 4.6 5.2 5.8

Right Margin:

1 2 6

1 1.6 2.2 2.8 3.4 4 4.6 5.2 5.8

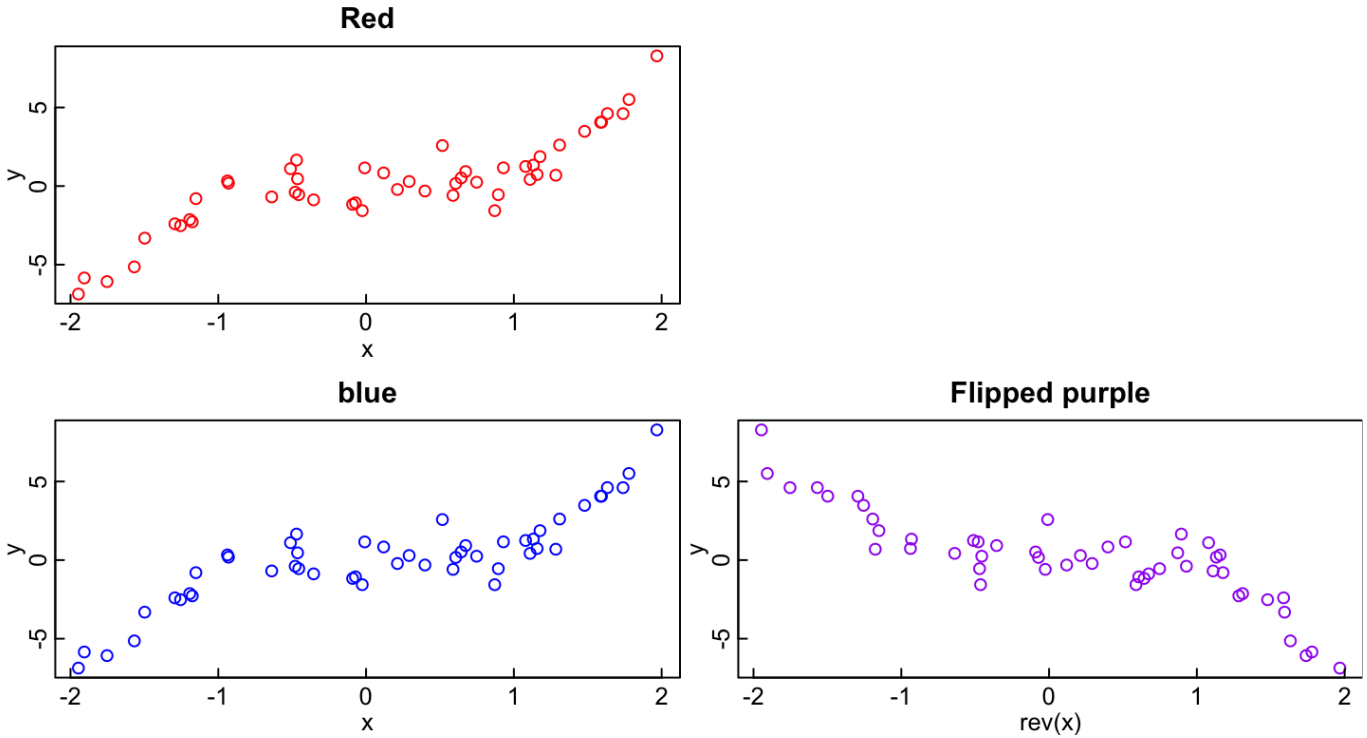



```
par(mar = c(5, 4, 4, 2)) # accepts numerical vectors of the form c(bottom, left, top, right)
# and specifies the sizes of figure margins in terms of lines of text.
```

Multiple Plots: `par()`'s `mfrow`

`mfrow` sets up a plotting grid of arbitrary dimensions.

```
par(mfrow = c(2, 2), mar = c(2, 2, 2, 0)) # Grid elements are filled by row
plot(x, y, main = "Red", col = "red"); plot(1, type = "n", axes = F, xlab = "", ylab = "")
plot(x, y, main = "blue", col = "blue"); plot(rev(x), y, main = "Flipped purple", col = "purple")
```



Outer Margins: `par()`'s `oma`

Bottom Margin:

1 2 4

1 1.4 1.8 2.2 2.6 3 3.4 3.8 4

Left Margin:

1 2 4

1 1.4 1.8 2.2 2.6 3 3.4 3.8 4

Top Margin:

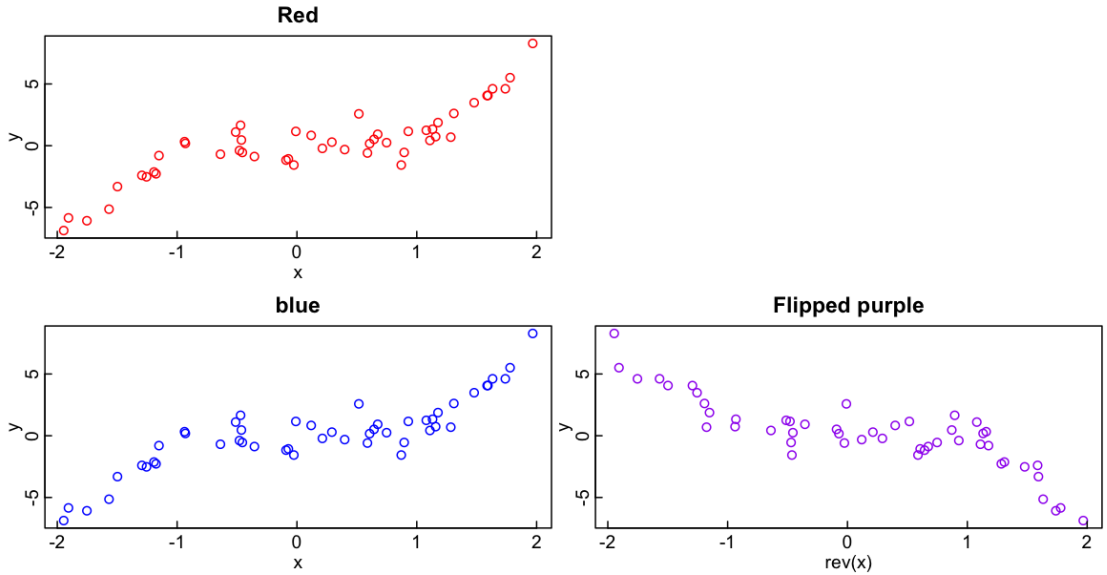
1 2 4

1 1.4 1.8 2.2 2.6 3 3.4 3.8 4

Right Margin:

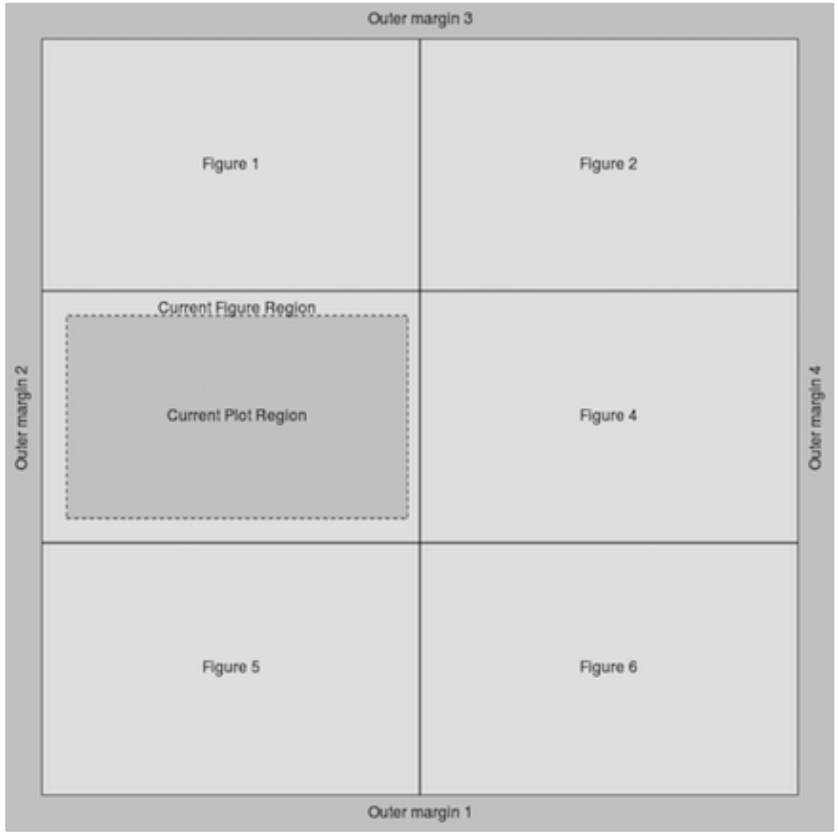
1 2 4

1 1.4 1.8 2.2 2.6 3 3.4 3.8 4




```
par(oma = c(2, 2, 2, 2))
```

How Multiple Panels are Structured in a Plotting Grid?



Precise Appearance Control

- The base graphics system has around 70 graphics parameters, controlling things, such as line style, colors, figure arrangement and text justification among many others, in all 3 regions.

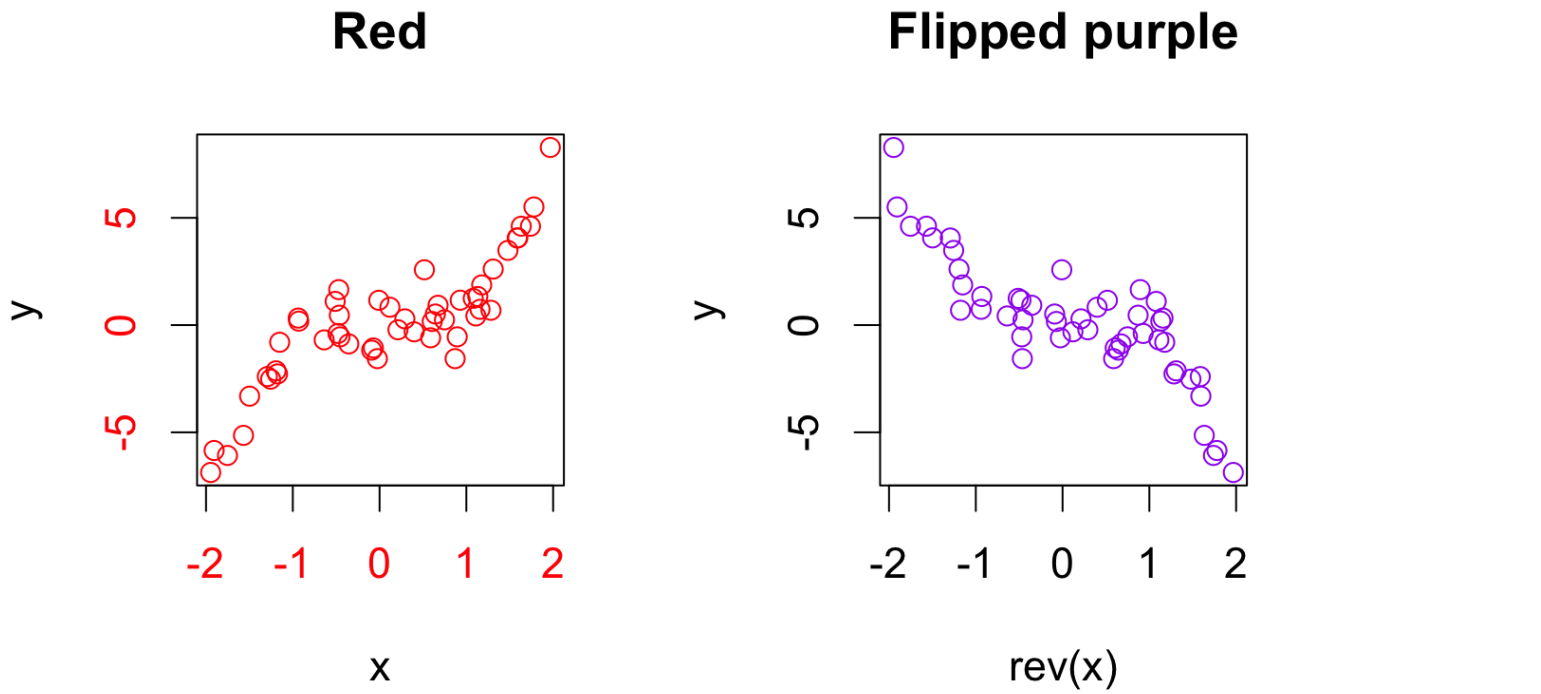
```
names(par())
```

```
##  [1] "xlog"      "ylog"      "adj"       "ann"       "ask"
##  [6] "bg"        "bty"       "cex"       "cex.axis"  "cex.lab"
## [11] "cex.main"  "cex.sub"   "cin"       "col"       "col.axis"
## [16] "col.lab"   "col.main"  "col.sub"   "cra"       "crt"
## [21] "csi"       "cxy"       "din"       "err"       "family"
## [26] "fg"        "fig"       "fin"       "font"      "font.axis"
## [31] "font.lab"  "font.main" "font.sub"  "lab"       "las"
## [36] "lend"      "lheight"   "ljoin"     "lmitre"    "lty"
## [41] "lwd"       "mai"       "mar"       "mex"       "mfcol"
## [46] "mfg"       "mfrow"     "mgp"       "mkh"       "new"
## [51] "oma"       "omd"       "omi"       "page"      "pch"
## [56] "pin"       "plt"       "ps"        "pty"       "smo"
## [61] "srt"       "tck"       "tcl"       "usr"       "xaxp"
## [66] "xaxs"      "xaxt"      "xpd"       "yaxp"      "yaxs"
## [71] "yaxt"      "ylbias"
```

- Most of these parameters, except a few read-only ones, can be set globally by the `par()` function to affect all plots in an R session.

- A majority of them can be overridden by arguments to specific plotting functions (e.g., `plot`, `lines`, `abline`, `axis`, `title`, `text`, etc.) so as to change the appearance of parts of a plot.

```
par(mfrow = c(1, 2), col.axis = "red", cex = 1.4)
plot(x, y, main = "Red", col = "red"); plot(rev(x), y, main = "Flipped purple", col = "purple", col.axis = 1)
```



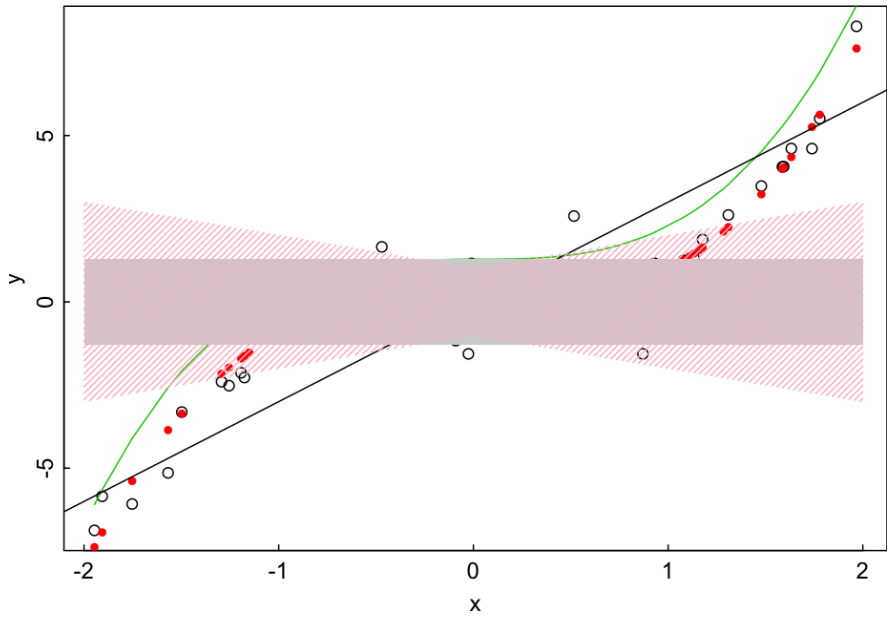
A Painters Model

R follows a **painters model**, starting with a "blank canvas" and adding items like we would add ink/paint to the canvas, with later items drawn on top of any previous items.

- High-level functions, such as `plot()`, `barplot()` and `hist()`, create complete plots, charts, etc. (if there is an existing plot, wipe the "canvas" clean and then paint a new plot)
- low-level functions, such as `points()`, `lines()`, and `rect()`, add graphics elements (including points, lines, arrows, rectangles, etc.) to existing plots and obscure what are below them.

Adding Items

- [x] Add Points
- [ ] Add Line 1
- [x] Add Line 2
- [ ] Add Line 3
- [x] Add Line 4
- [ ] Add Line 5
- [ ] Add Text
- [x] Add A Rectagle
- [x] Add A Polygon

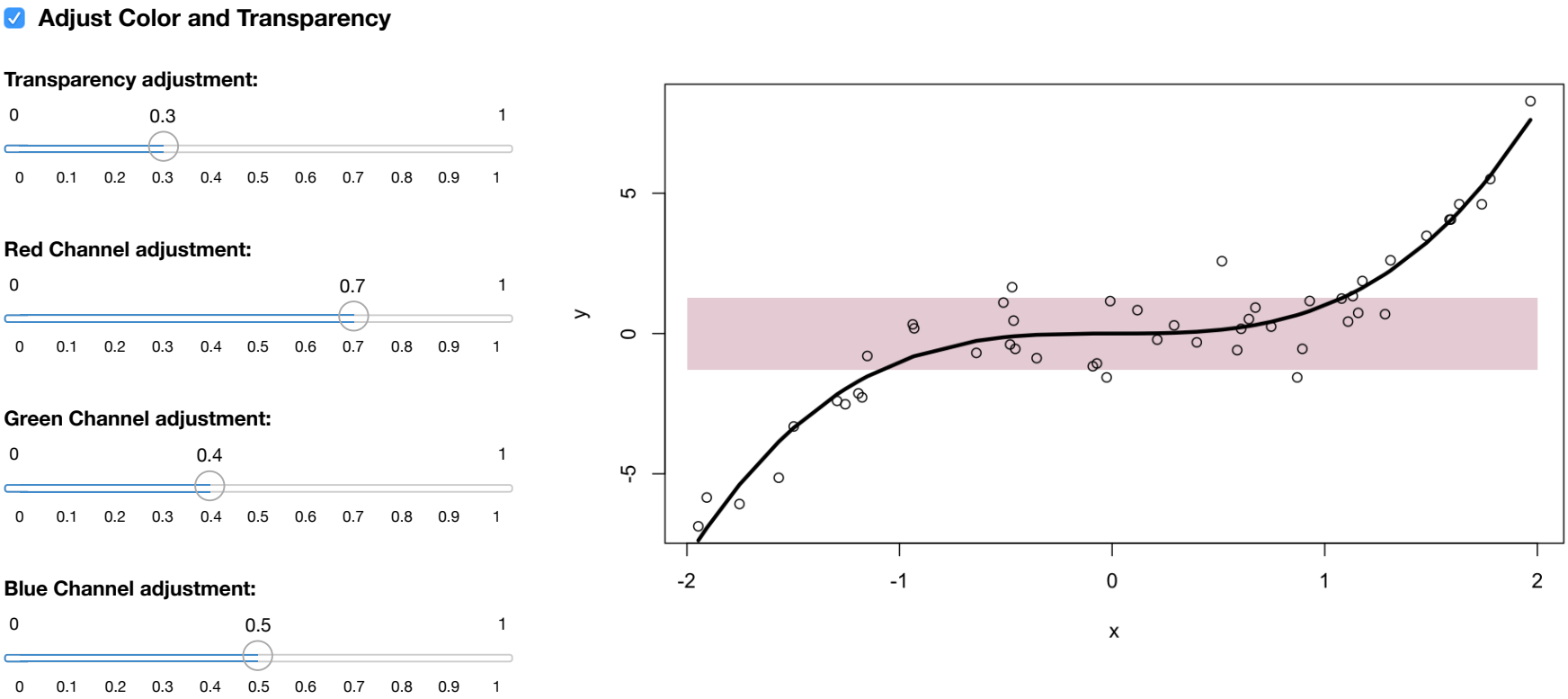



```
plot(x, y)
points(x, x^3, pch = 20, col= "red")
lines(x, x^3 + qnorm(0.90), col = 3
abline(a = 0, b = 3)

rect(-2, qnorm(0.10), 2, qnorm(0.90),
# xleft, ybottom, xright, ytop
    col = "lightgrey", border = NA)

polygon(c(-2, 0, 2, 2, 0, -2), c(-3, -1, -3, 3, 1, 3),
# x, y (vertex coords)
    col = "pink", density = 30, border = NA)
```

Use `plot()` with `type = "n"`, then `rect()` and `points()` and custom transparent colors with `rgb()`:

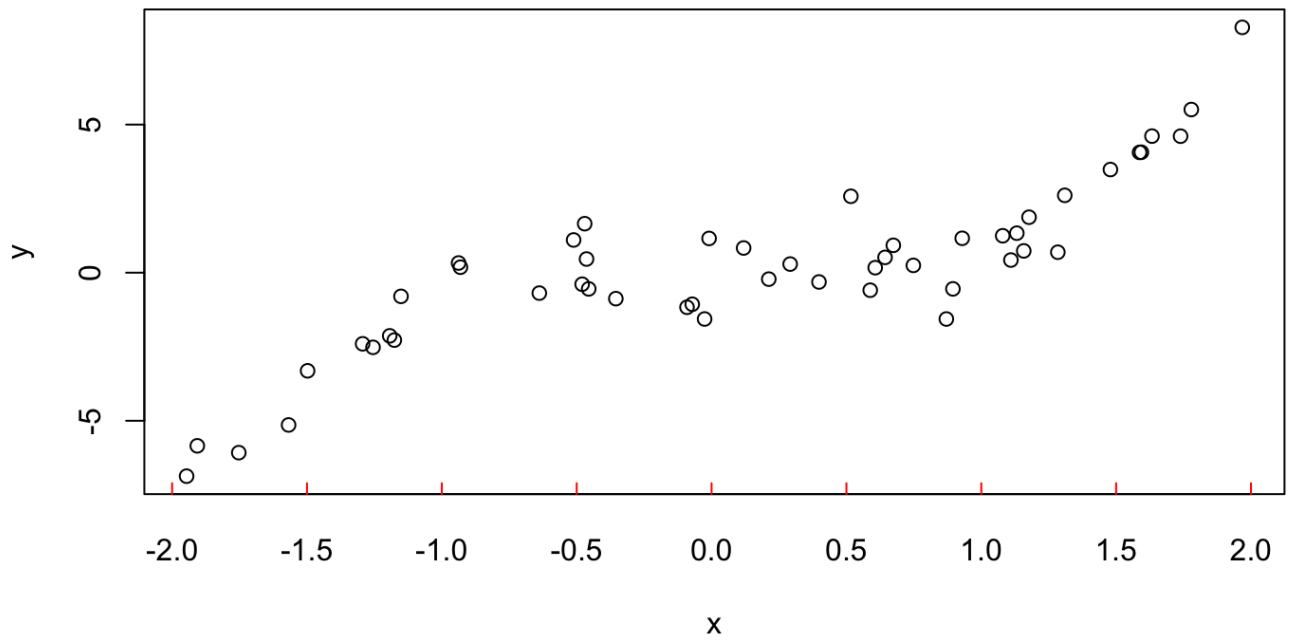



```
plot(x, y, type = "n")
rect(-2, qnorm(0.10), 2, qnorm(0.90), col = rgb(0.7, 0.4, 0.5, 0.3), border = NA)
points(x, y)
```

Adding Axes

Set `xaxt`/`yaxt` options in `plot()` to `"n"` or `axes = FALSE` or to suppress the plotting of one or both axes, and use `axis()` to add an axis to the existing plot with fine-tuning parameters.

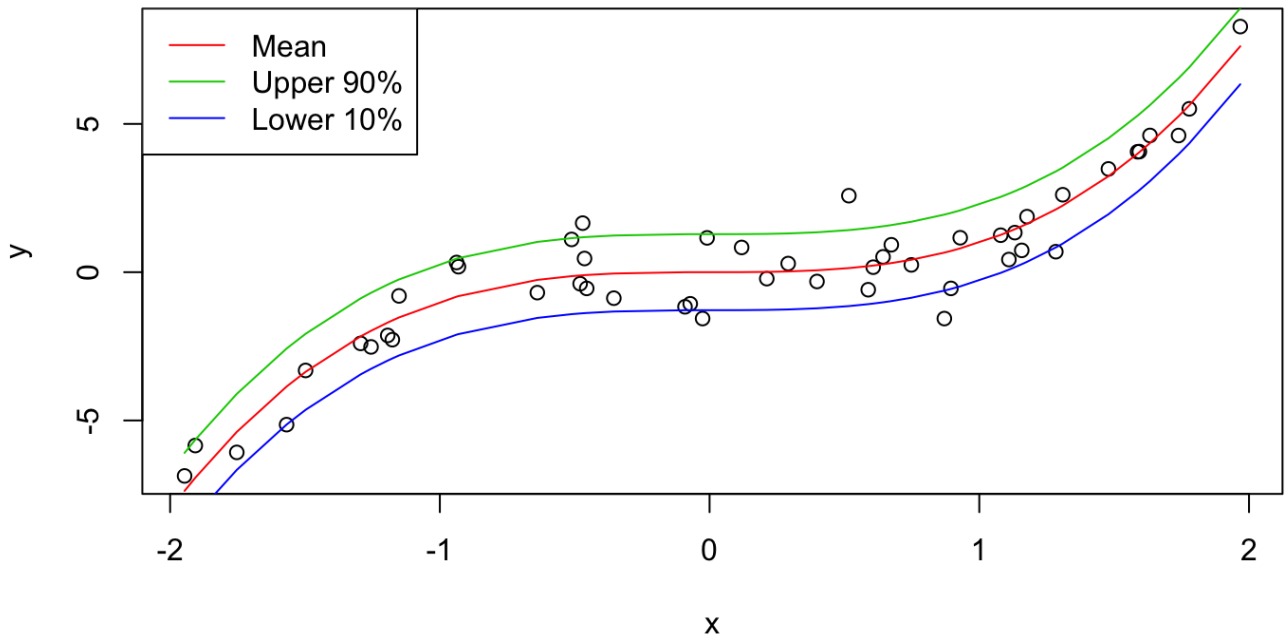
```
plot(x, y, xaxt = "n")
axis(side = 1, at = seq(-2, 2, by = 0.5), col.ticks = "red", tcl = 0.3)  # side: 1 to 4, counting clockwise from the bottom
```



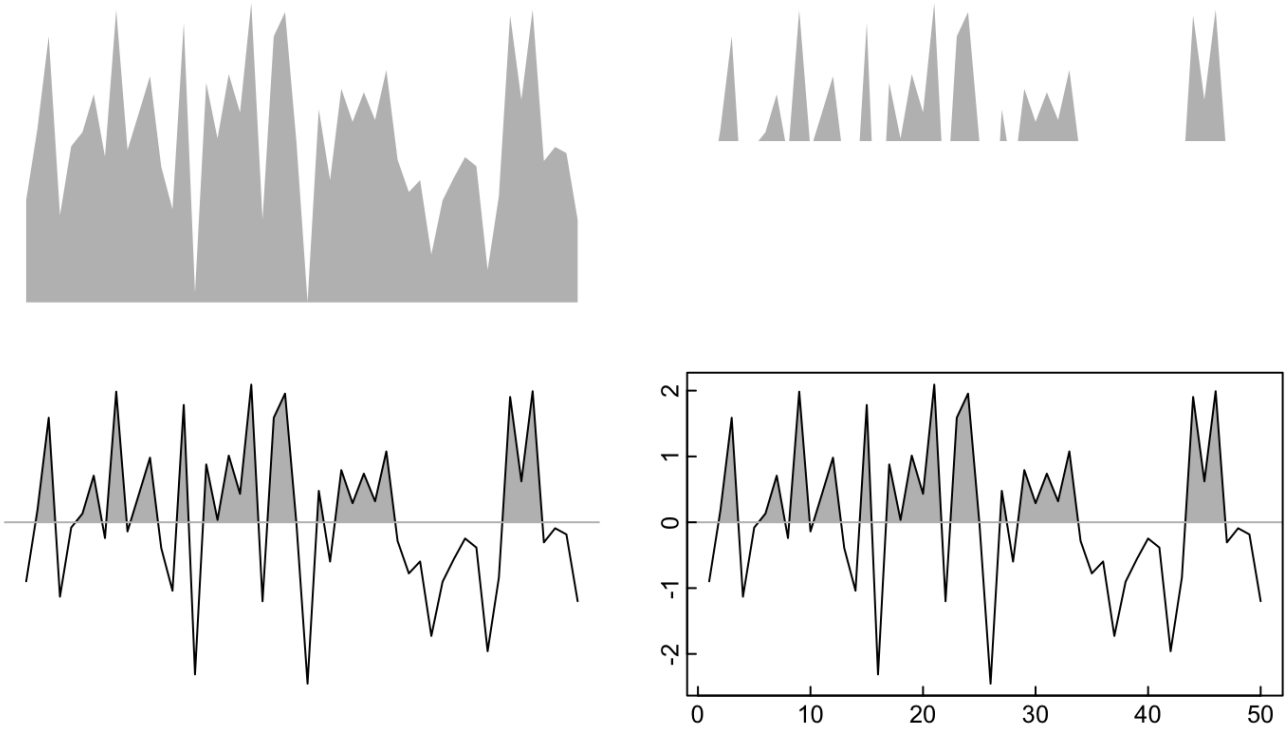
Adding a Legend

Use `legend()` to add a legend to an existing plot. The legend position can be represented by `x` and `y` coordinates or one of `"topleft"`, `"topright"`, `"bottomleft"`, `"bottomright"`:

```
plot(x, y)
lines(x, x^3, col = 2); lines(x, x^3 + qnorm(0.90), col = 3); lines(x, x^3 + qnorm(0.10), col = 4)
legend("topleft", legend = c("Mean", "Upper 90%", "Lower 10%"), lty = 1, col = 2:4)
```



```
set.seed(2); xx <- 1:50; yy <- rnorm(50)
plot (yy ~ xx, type="n", axes = FALSE, ann = FALSE)
polygon(c(xx[1], xx, xx[50]), c(min(yy), yy, min(yy)), col = "gray", border = NA)
usr <- par("usr")
rect(usr[1], usr[3], usr[2], 0, col = "white", border = NA)
lines(xx, yy)
abline (h = 0,col = "gray")
box(); axis(1); axis(2)
```



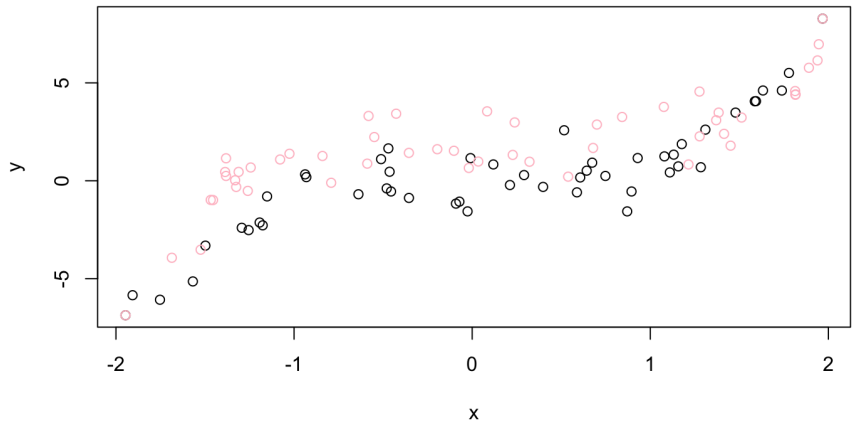
Another Way to Plot Multiple Data Series

Setting `new` to `"TRUE"` with `par()` allows R to make the second plot without cleaning the first.

Additional things to consider:

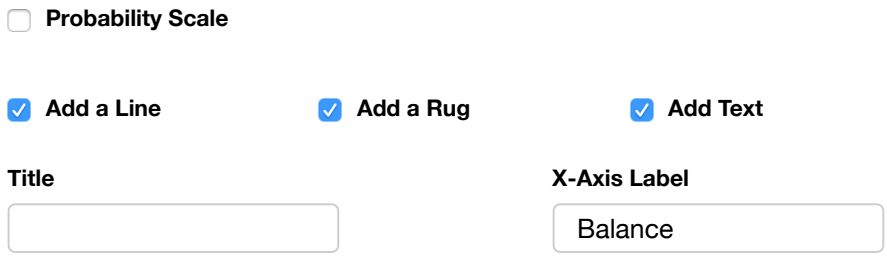
- Set the `axes` of the second plot to FALSE and `xlab` and `ylab` to empty strings.
- Make sure the axes ranges are equal across these plots.

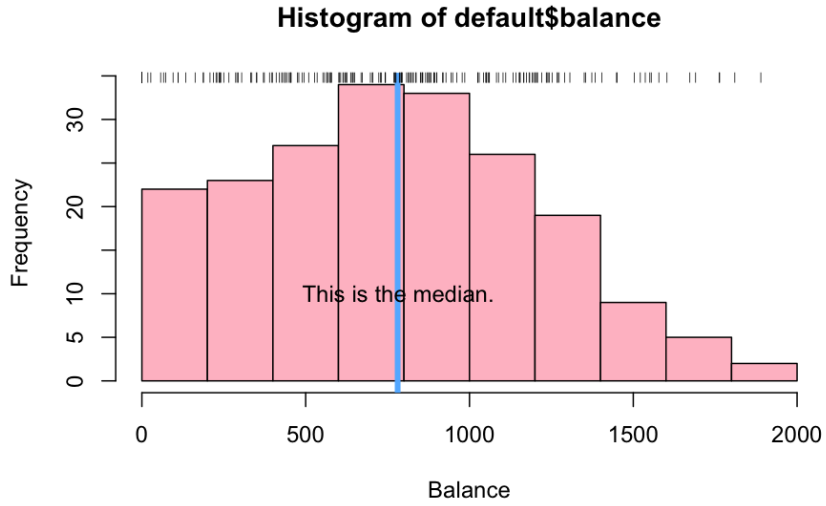
```
set.seed(2)
x1 <- sort(runif(50, min = -2, max = 2))
y1 <- x^3 + rnorm(50)
plot(x, y)
par(new = TRUE)
plot(x1, y1, axes = FALSE, xlab = "", ylab = "", col = "pink")
```



Plotting a Histogram

Use `hist()` to plot a histogram of a numeric vector,






```
hist(default$balance, freq = TRUE, xlab = "Balance")
abline(v = median(default$balance))
rug(default$balance, side = 3)
text(median(default$balance) + 2, 10, labels = "This is the median.")
```

Adding a Density Curve to a Histogram

To estimate a density from a numeric vector, use `density()`. This returns a list; it has components `x` and `y`.

```
density.est <- density(default$balance, adjust = 2)  # Twice the default smoothing bandwidth
class(density.est)
```

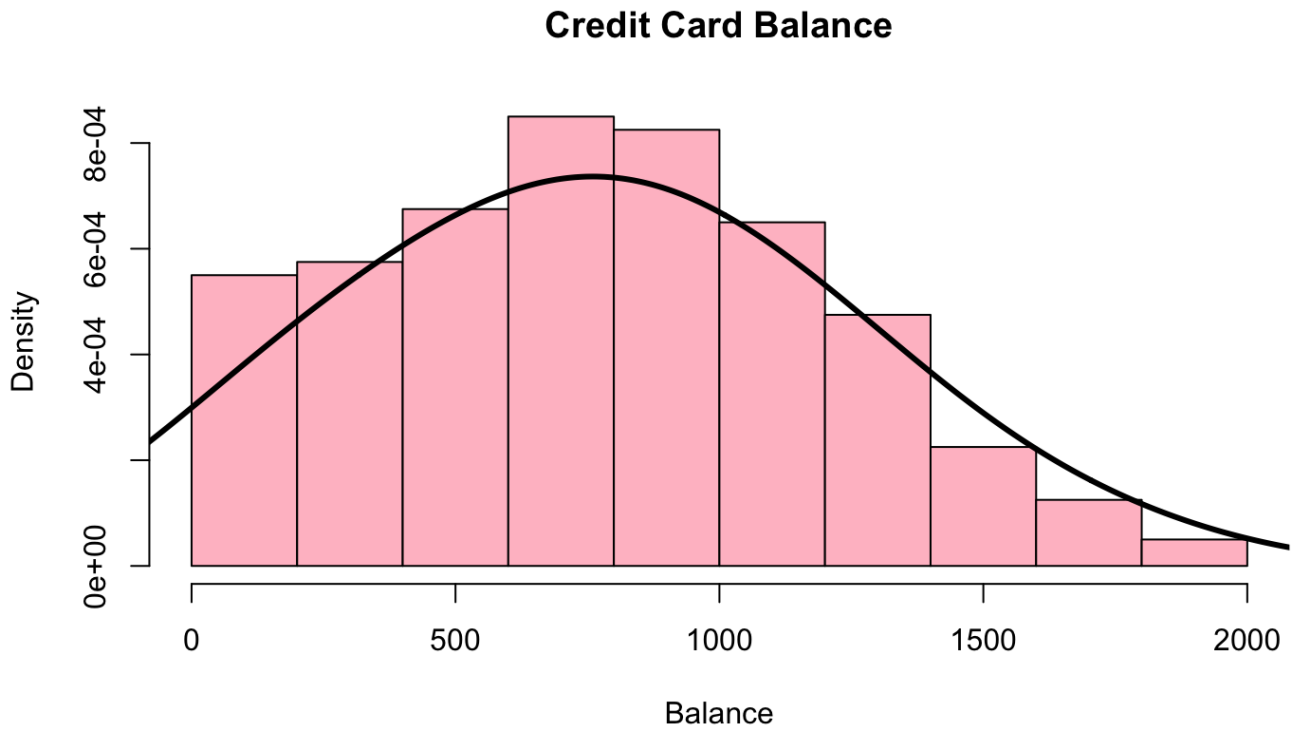
```
## [1] "density"
```

```
names(density.est)
```

```
## [1] "x"         "y"         "bw"        "n"         "call"      "data.name"
## [7] "has.na"
```

We can call `lines()` directly on the object returned by `density()`.

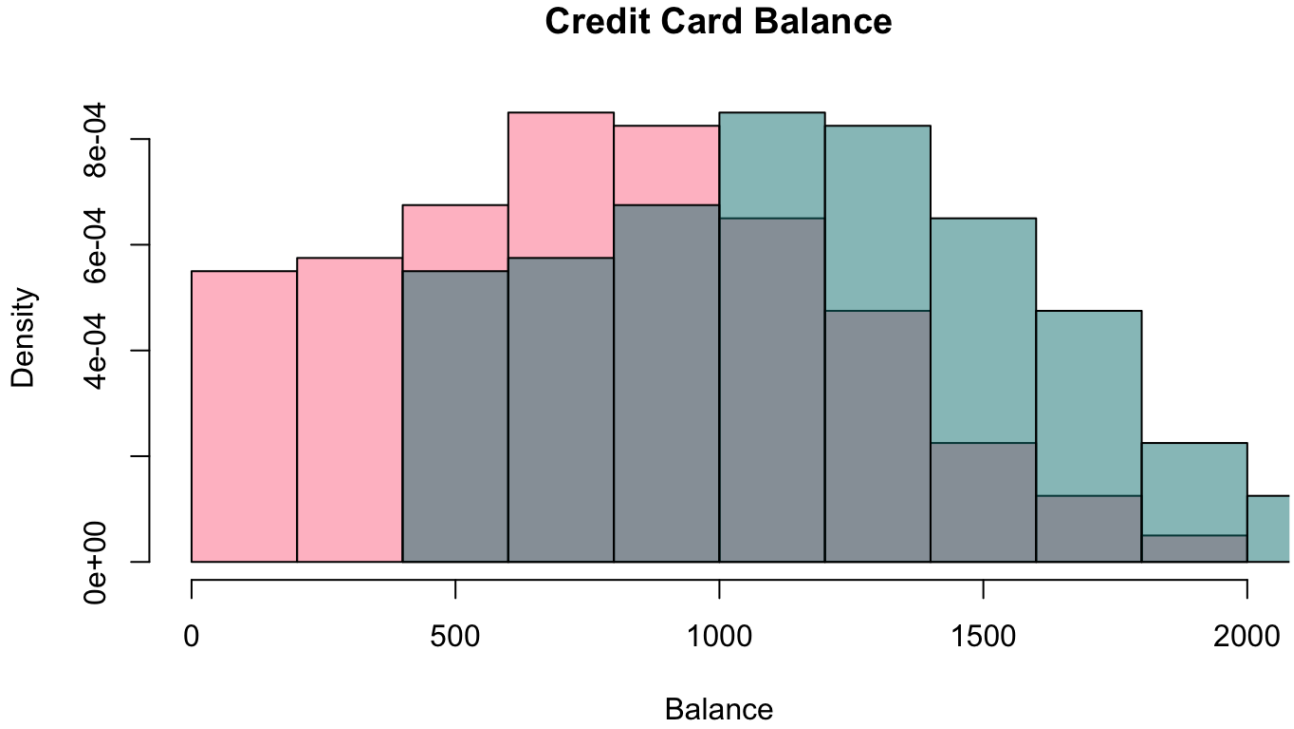
```
hist(default$balance, col = "pink", freq = FALSE, xlab = "Balance", main = "Credit Card Balance")
lines(density.est, lwd = 3)
```



Adding a Histogram to an Existing Plot

To add a histogram to an existing plot (say, another histogram), use `hist()` with `add = TRUE`

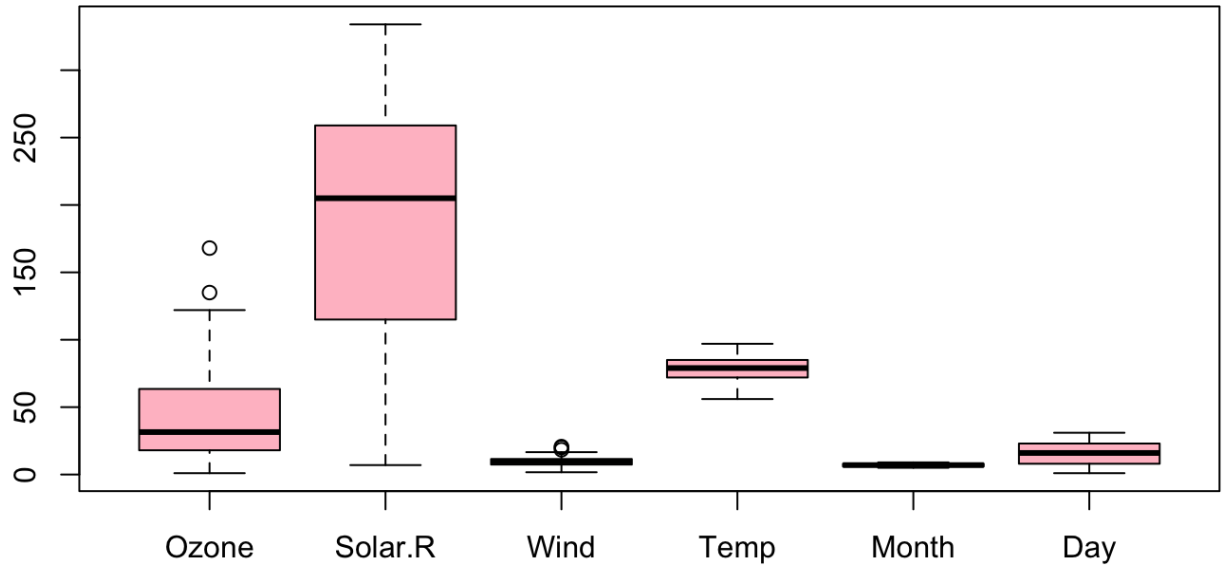
```
hist(default$balance, col = "pink", freq = FALSE, xlab = "Balance", main = "Credit Card Balance")
hist(default$balance + 400, col = rgb(0,0.5,0.5,0.5), # Note the use of transparency
     freq = FALSE, add = TRUE)
```



Plotting a Boxplot

A boxplot show the distribution of a vector. It is very useful to example the distribution of different variables.

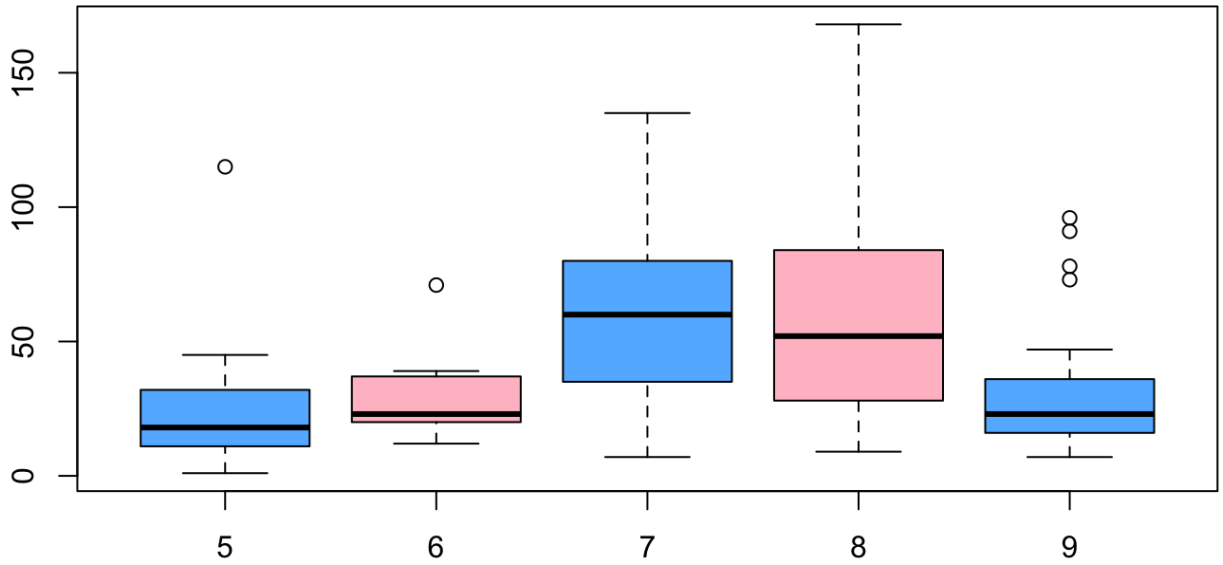
```
boxplot(airquality, col = "pink")
```



Grouped Boxplots

`boxplot()` can also accept a formula such as `y ~ x` as an argument, where y is a vector of numeric values to be split into groups according to the grouping variable `x` (usually a **factor**).

```
# equivalent to plot(factor(airquality$Month), airquality$Ozone)
boxplot(Ozone ~ Month, data = airquality, col = c("steelblue1", "pink"))
```

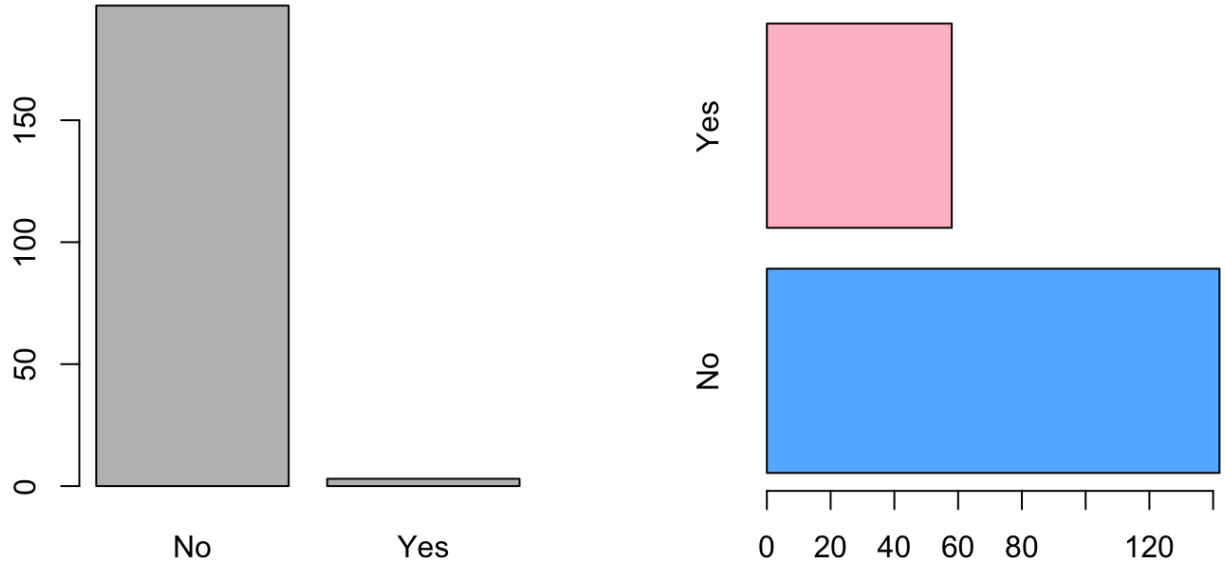


Plotting a Barplot

A barplot is a frequently used type of display that compares counts, frequencies, totals or other summary measures for a series of categories.

`barplot()` creates a barplot with vertical or horizontal bars.

```
par(mfrow = c(1, 2))
barplot(table(default$default))
barplot(table(default$student), horiz = TRUE, col = c("steelblue1", "pink"))
```



Working with a Matrix

In addtion to a vector, `barplot()` can take in a matrix. If the input is a matrix, a stacked bar is plotted, and each column is represented by a stacked bar.

```
VADeaths
```

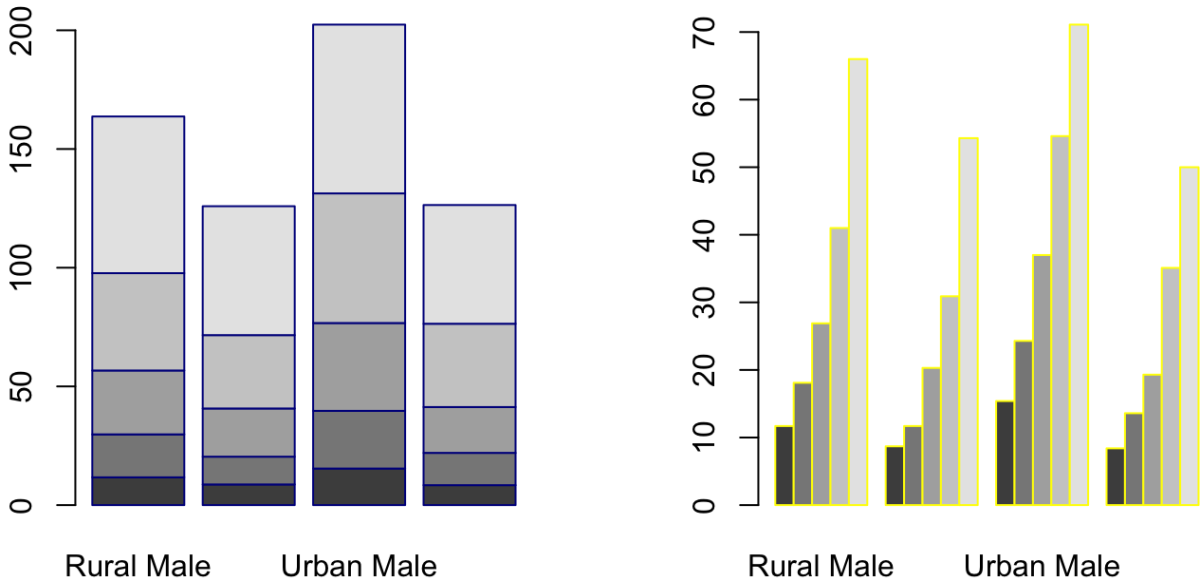
```
##       Rural Male Rural Female Urban Male Urban Female
## 50-54       11.7          8.7       15.4          8.4
## 55-59       18.1         11.7       24.3         13.6
## 60-64       26.9         20.3       37.0         19.3
## 65-69       41.0         30.9       54.6         35.1
## 70-74       66.0         54.3       71.1         50.0
```

```
class(VADeaths)
```

```
## [1] "matrix"
```

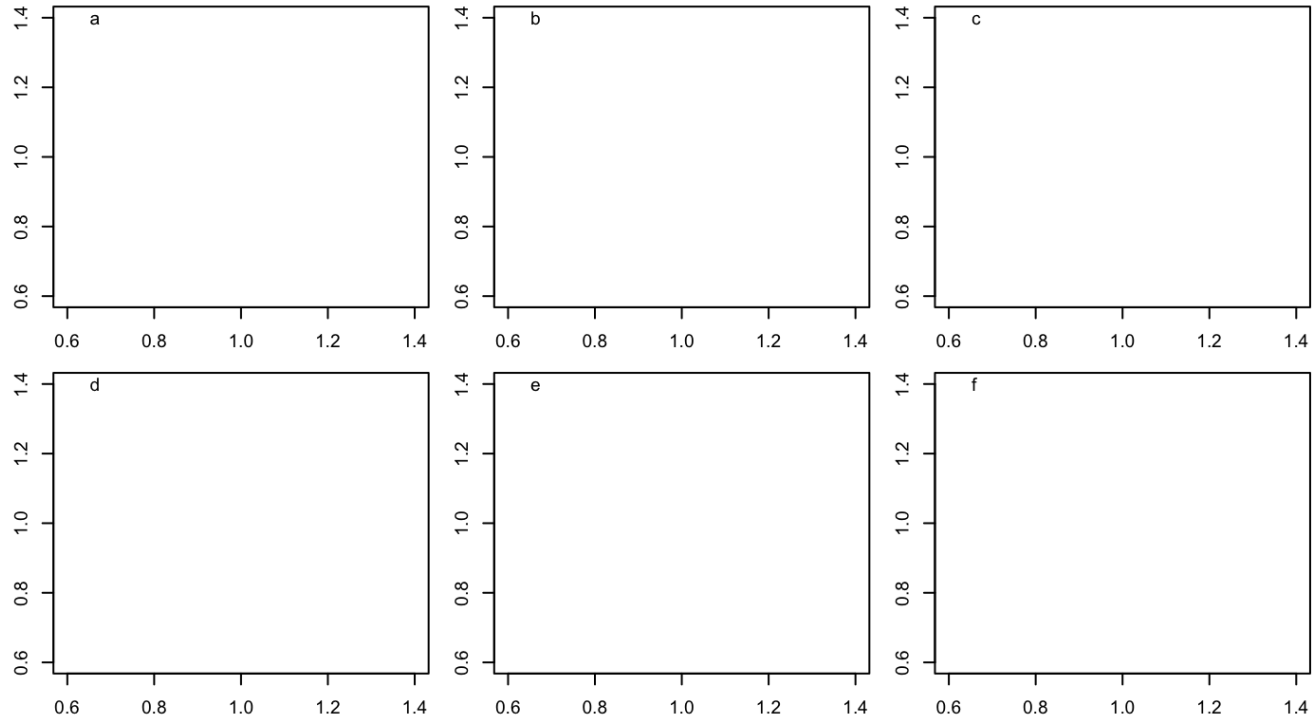
Instead of a stacked bar, we can have different bars for each element in a column juxtaposed to each other by specifying the parameter `beside = TRUE`.

```
par(mfrow = c(1, 2))
barplot(VADeaths, border = "dark blue")
barplot(VADeaths, border = "yellow", beside = TRUE)
```



Basic Multipanel Layouts with `par()`

```
par(mfrow = c(2, 3), cex = 0.6, mar = c(3, 3, 0, 0), oma = c(1, 1, 3, 1))
for (i in 1:6) {
    plot(1, 1, type = "n")
    mtext(letters[i], side = 3, line = -1, adj = 0.1, cex = 0.6)
}
```



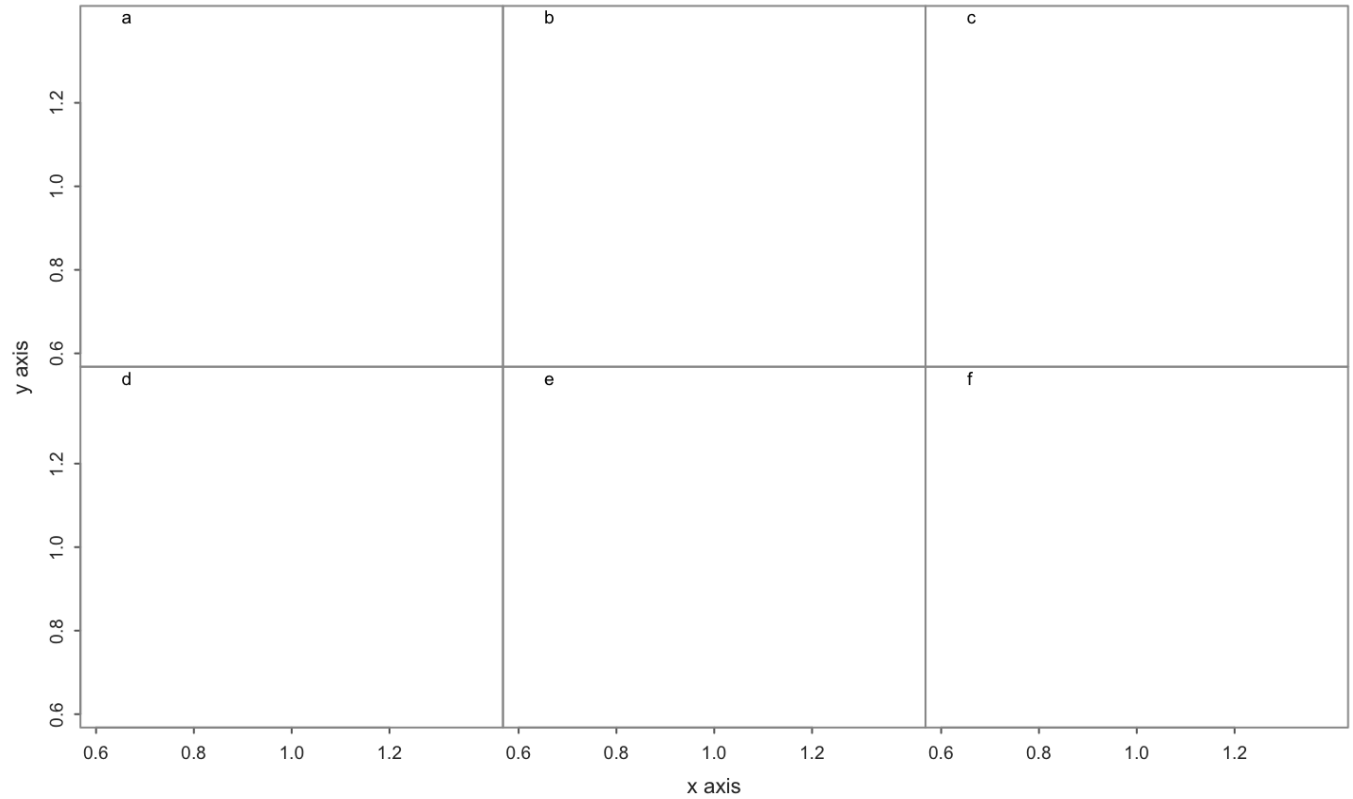
Some of the Common "Tricks"

We can eliminate the redundant axes, remove margin space, and reduce the emphasis on the structural (non-data) elements of the figure.

```
par(mfrow = c(2, 3), cex = 0.6, mar = c(0, 0, 0, 0), oma = c(4, 4, 0.5, 0.5),
    tcl = -0.25, mgp = c(2, 0.6, 0))
for (i in 1:6) {
    plot(1, axes = FALSE, type = "n")
    mtext(letters[i], side = 3, line = -1, adj = 0.1, cex = 0.6)
    if (i %in% c(4, 5, 6))
        axis(1, col = "grey40", col.axis = "grey20", at = seq(0.6, 1.2, 0.2))
    if (i %in% c(1, 4))
        axis(2, col = "grey40", col.axis = "grey20", at = seq(0.6, 1.2, 0.2))
    box(col = "grey60")
}

mtext("x axis", side = 1, outer = TRUE, cex = 0.7, line = 2.2, col = "grey20")
mtext("y axis", side = 2, outer = TRUE, cex = 0.7, line = 2.2, col = "grey20")
```

Some of the Common "Tricks", Continued



Fancy Multipanel Layouts with `layout()`

`layout()` takes a matrix and turns it into a layout. The numbers in the matrix correspond to the order that the panels will be plotted in. Cells with the same number represent a single panel.

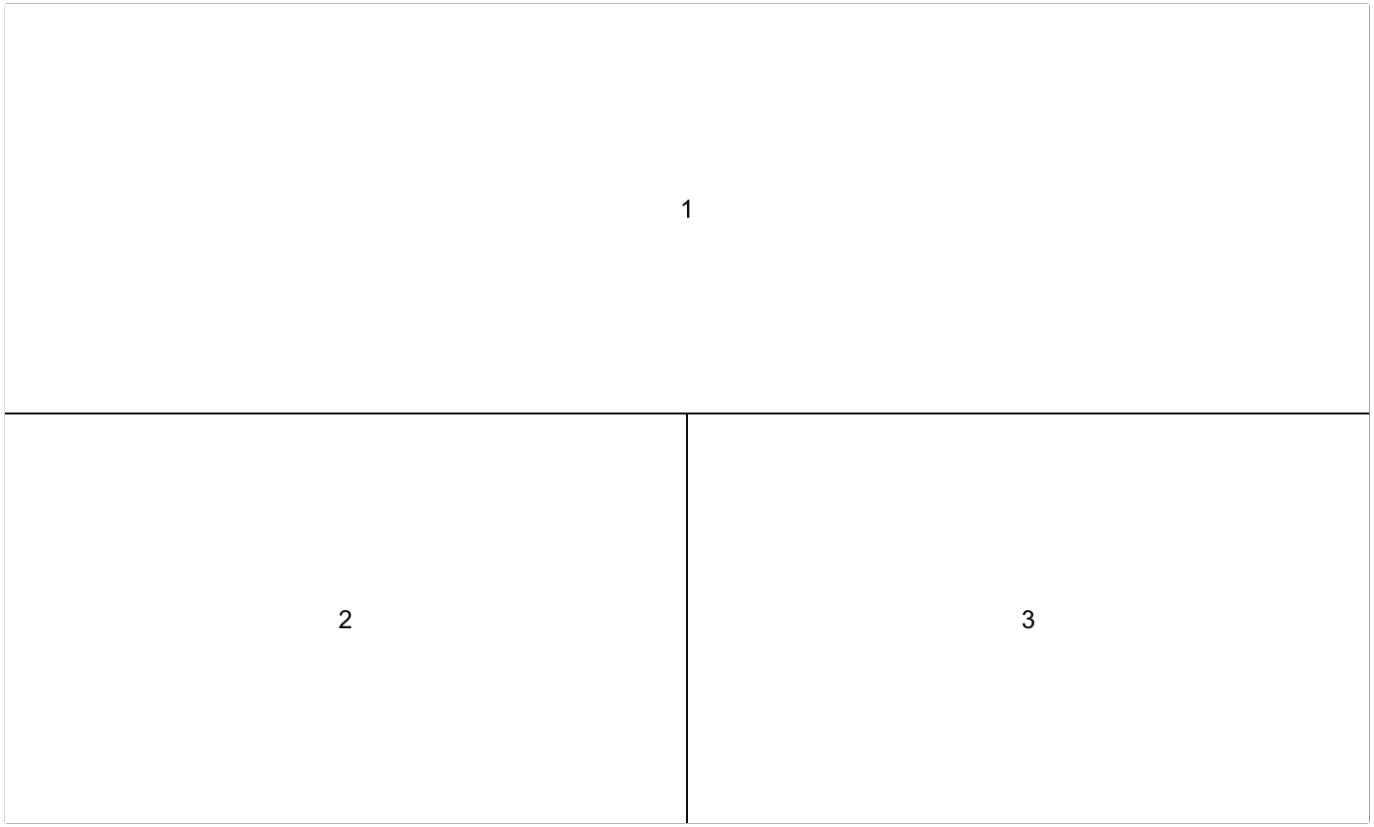
```
(m <- rbind(c(1, 1), c(2, 3)))
```

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

```
layout(m)
```

```
layout.show(3)  # The number of the panels to show
```

Fancy Multipanel Layouts with `layout()`, Continued

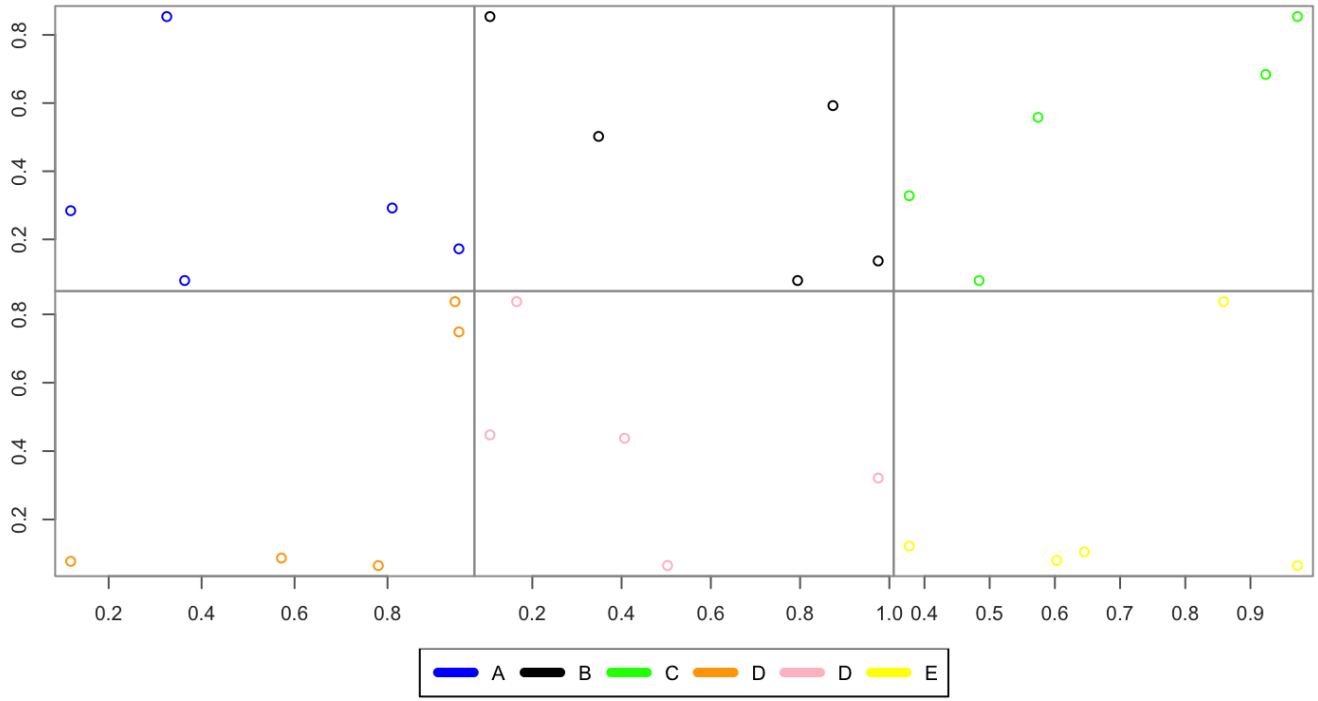


Fancy Multipanel Layouts with `layout()`, Continued

We can create some fairly complex layouts with `layout()`. For example, some panels empty can be left empty as margin space.

```
m <- matrix(c(1, 2, 3, 4, 5, 6, 7, 7, 7), nrow = 3, ncol = 3, byrow = TRUE)
layout(mat = m, heights = c(0.4, 0.4, 0.2))
par(oma = c(4, 4, 0.5, 0.5))
plot_colors <- c("blue", "black", "green", "orange", "pink", "yellow")
for (i in 1:6) {
    par(mar = c(0, 0, 0, 0))
    plot(runif(5), runif(5), axes = FALSE, col = plot_colors[i])
    if (i %in% c(4, 5, 6))
        axis(1, col = "grey40", col.axis = "grey20")
    if (i %in% c(1, 4))
        axis(2, col = "grey40", col.axis = "grey20")
    box(col = "grey60")
}
par(mar = c(0, 0, 3, 0))
plot(1, type = "n", axes = FALSE, xlab = "", ylab = "")
legend(x = "top", inset = 0, legend = c("A", "B", "C", "D", "D", "E"), col = plot_colors,
    lwd = 5, horiz = TRUE)
```

Fancy Multipanel Layouts with `layout()`, Continued



Saving Plots

Use the `pdf()` function to save a pdf file of your plot, in your R working directory. Use `getwd()` to get the working directory, and `setwd()` to set it.

```
getwd()  # This is where the pdf will be saved
```

```
## [1] "/Users/jiajia/Dropbox/Courses/ISOM3390/Lecture Notes/BasePlotting"
```

```
pdf(file = "noisy_cubics.pdf", height = 7, width = 7)  # Height, width are in inches
par(mfrow = c(2, 2), mar = c(4, 4, 2, 0.5))
plot(x, y, main = "Red cubic", pch = 20, col = "red")
plot(x, y, main = "Blue cubic", pch = 20, col = "blue")
plot(rev(x), y, main = "Flipped green", pch = 20, col = "green")
plot(rev(x), y, main = "Flipped purple", pch = 20, col = "purple")
graphics.off()
```

Also, use the `jpg()` and `png()` functions to save jpg and png files

Base Plotting System Summary

A plot is just a series of R commands.

Pros:

- Convenient, mirrors how we think of building plots and analyzing data.

Cons:

- Can't go back once plot has started (i.e. to adjust margins); need to plan in advance.
- Difficult to "translate" to others once a plot has been created (no graphical "language").