DATA VISUALIZATION

Introduction of R graphics and plots

Basic R Graphics

Introduction

- Why should we plot graphs?
  - To show the data visually.
  - Gives a visual summary of the data.

- R has a lot of very good graphing functions, and most of the time you can produce a clean, high-quality graphic without having to learn very much.

- Unfortunately, there are occasions when you want to do something non-standard. In this case things get complicated.

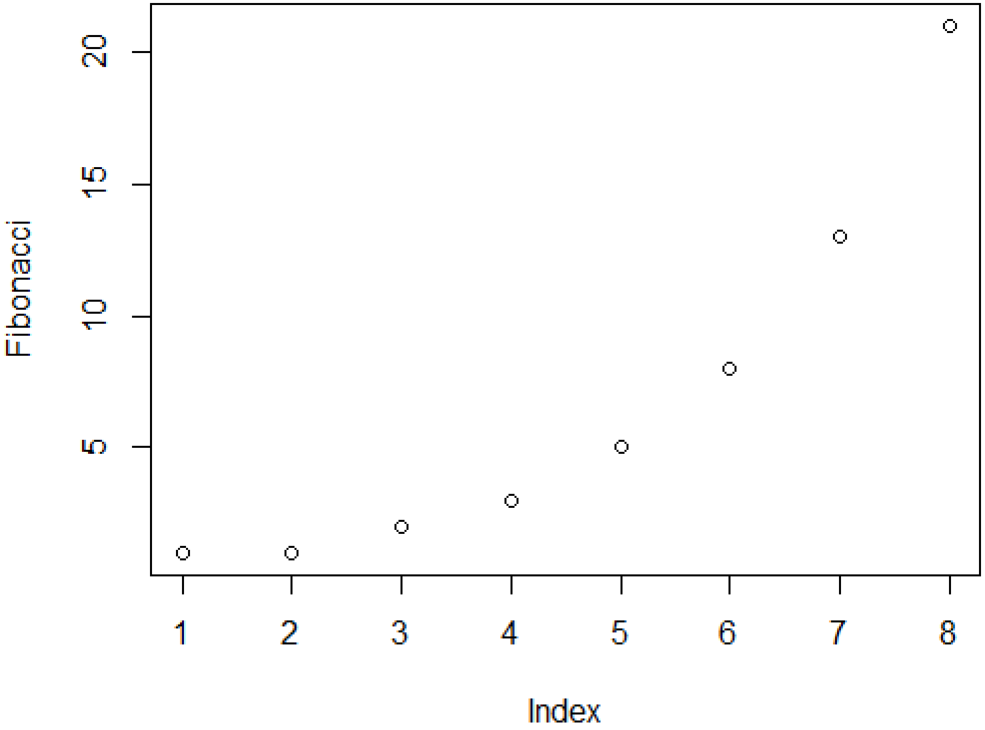
An overview of R graphics

- R graphic as being much like a traditional painting.
  - Start out with an empty canvas.
  - Every time you use a graphics function, it paints some new things onto your canvas.
  - You can also paint more things over the top if you want
  - But just like painting, you can't "undo" your strokes.
  - If you make a mistake, you have to throw away your painting and start over.

An introduction to plotting

- We begin with some simple example:

```
> Fibonacci <- c(1,1,2,3,5,8,13,21)
> plot(Fibonacci)
```



plot() function

- The `plot()` function is a generic function like `print()` or `summary()`. Its behaviour depends on the input.

- If you look at `help(plot)` or `?plot`, you can see that it takes two input, `x` and `y` (optional).

- For now, we are only doing basic plotting which is done by `plot.default()`.

- In `plot.default()` help documentation, you can see more arguments, most which are “graphical parameters”

Graphical parameters

- Basically, there are some characteristics/setting of a plot which are universal:
  - colour to use for the plot
  - font type/size
  - plot title
  - x and y axes
  - etc

- In order to avoid having too many arguments for every single function, R does it by referring these as “graphical parameters”.

- You can check the graphical parameters using the `par()` function and its help documentation.

Graphical parameters

- You can modify the graphical parameters using the `par()` function, but we rarely do this.

- In most cases, we change the parameters for individual plot.

Customizing the title and axes

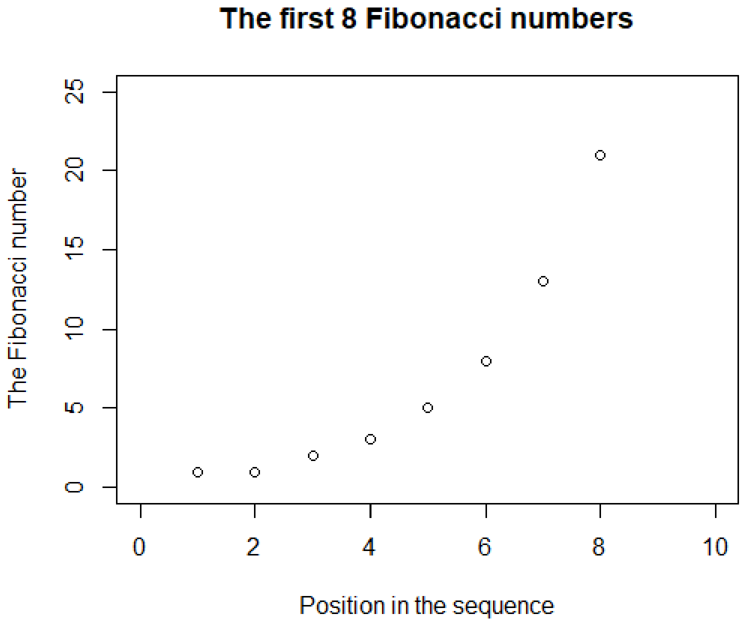
- Here are the commonly used arguments used in the `plot()` function to modify the title and axes:
  - `main`:
    - A character string containing the title.
  - `xlab` and `ylab`:
    - A character string containing the x-axis label and y-axis label, respectively.
  - `xlim`, and `ylim`:
    - A vector to specify the limit for the x-axis and y-axis.

Customizing the title and axes

- More (less commonly used) arguments:
  - `font.main`, `font.axis`, and `font.lab`:
    - Numerical value to change the font style for plot title, axis tick marks, and axis labels.
    - 1 is for plain text, 2 is for boldface, 3 is for italic, and 4 is for bold italic
  - `col.main`, `col.axis`, and `col.lab`:
    - Specify colors for the plot title, axis tick marks, and axis labels.
    - Use `colours()` to see a list of available colour names known in R.
    - Otherwise you can specify using hex code or `rgb()` function.
  - `cex.main`, `cex.axis`, and `cex.lab`:
    - Change the font size for the plot title, axis tick marks, and axis labels.
    - Default to 1.
    - "cex" stands for "character expansion"
  - `axes`:
    - `axes=TRUE` if you want to draw axes (tick marks). Otherwise, `axes=FALSE`.

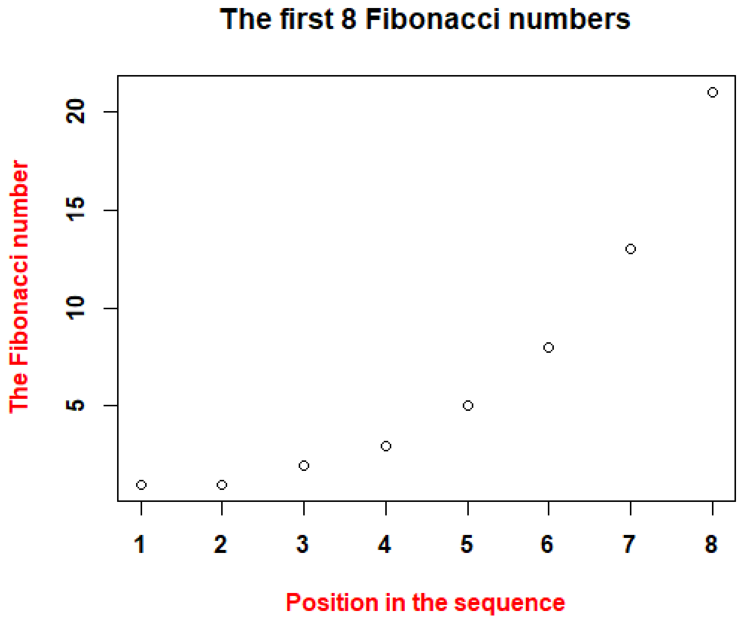
Example

```
plot(Fibonacci,                                # the data to plot
     main = "The first 8 Fibonacci numbers",  # the title
     xlab = "Position in the sequence",       # x-axis label
     ylab = "The Fibonacci number",           # y-axis label
     xlim = c(0,10),                          # x-axis region
     ylim = c(0,25)                           # y-axis region
     )
```



Example

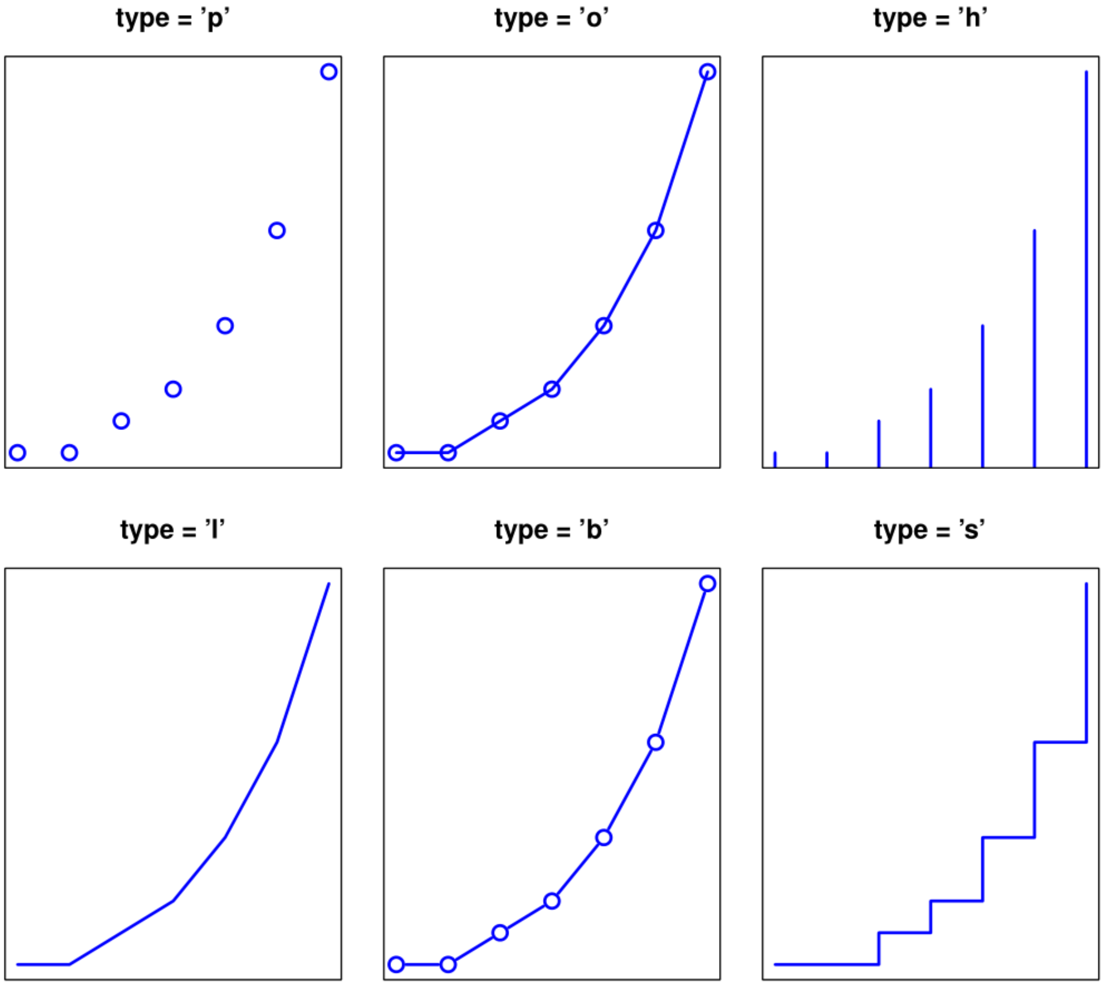
```
plot(Fibonacci,                                   # the data to plot
     main = "The first 8 Fibonacci numbers",      # the title
     xlab = "Position in the sequence",           # x-axis label
     ylab = "The Fibonacci number",               # y-axis label
     font.axis = 2,                               # bold text for numbering
     font.lab = 2,                                # bold text for labels
     col.lab = "red"                              # red colour for labels
     )
```



Changing plot types

- You can also change the type of plots by changing the `type` argument:
  - `type = "p"`: Draw the points only.
  - `type = "l"`: Draw a line through the points.
  - `type = "o"`: Draw the line over the top of the points.
  - `type = "b"`: Draw both points and lines, but don't overplot.
  - `type = "h"`: Draw "histogram-like" vertical bars.
  - `type = "s"`: Draw a staircase, going horizontally then vertically.
  - `type = "S"`: Draw a Staircase, going vertically then horizontally.
  - `type = "c"`: Draw only the connecting lines from the "`b`" version.
  - `type = "n"`: Draw nothing. Useful if you just want an empty box.

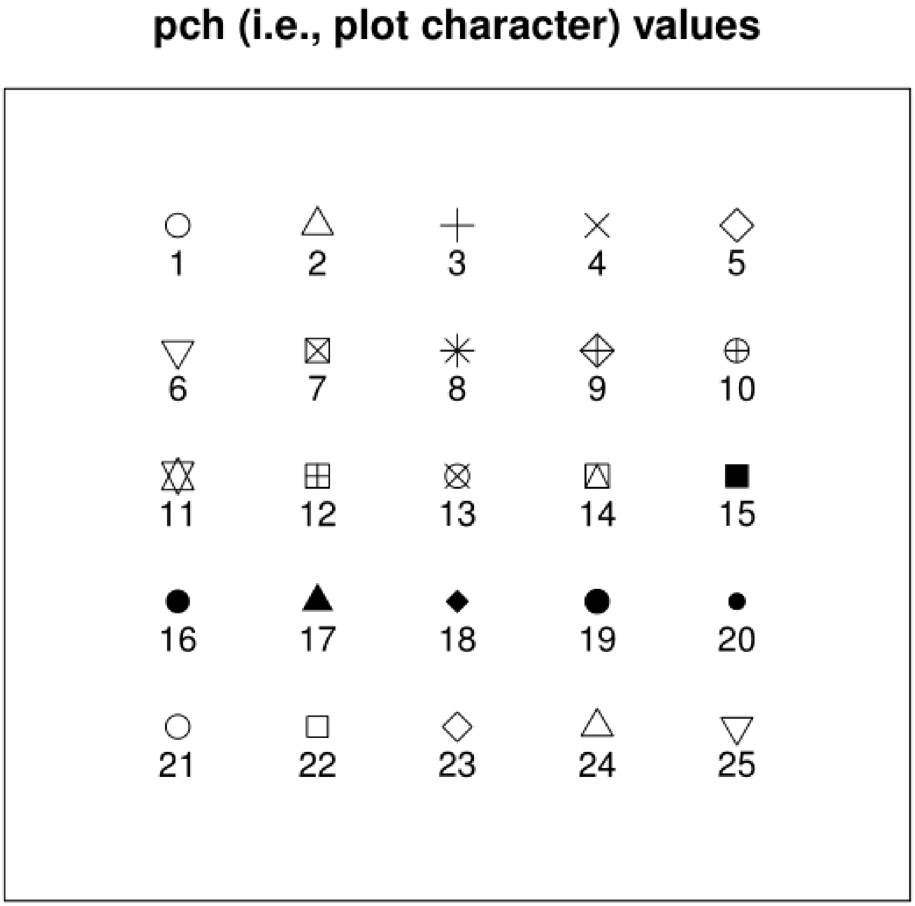
Plot types

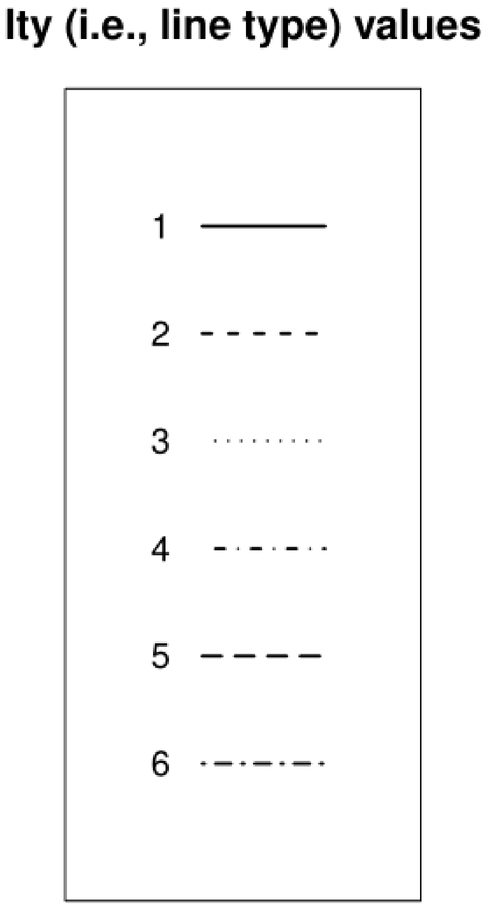


Changing other features of the plot

- `col`:
  - Specify the colour of the points/lines.
  - Similar to the previous colour argument, you can use on one of `colours()` or specify using hex code or `rgb()`.
- `pch`:
  - Specify the plot character (character used for plot points).
  - More info on next page.
- `cex`:
  - Specify the size of the plot points. Default is value 1.
- `lty`:
  - Specify the line type.
  - More info on next page.
- `lwd`:
  - Specify the line width. Default value is 1.

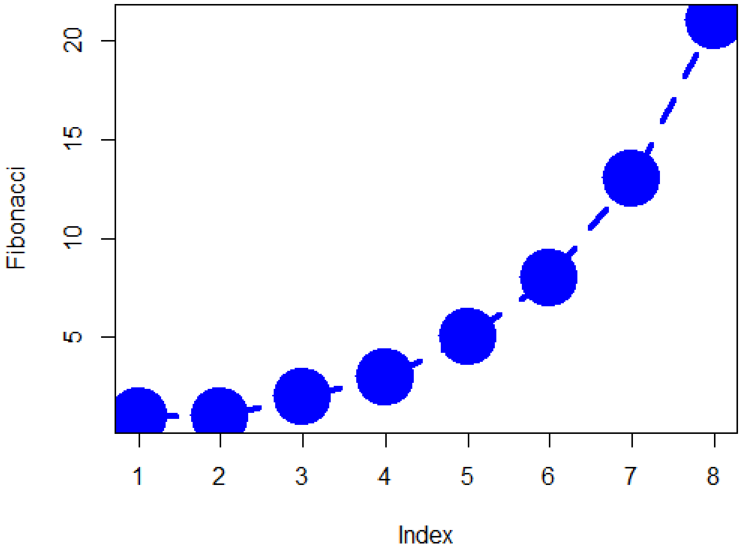
Plot characters and line types values





Example

```
plot(Fibonacci,           # the data set
     type = "b",          # plot both points and lines
     col = "blue",        # change the plot colour to blue
     pch = 19,            # plotting character is a solid circle
     cex = 5,             # plot it at 5x the usual size
     lty = 2,             # change line type to dashed
     lwd = 4              # change line width to 4x the usual
     )
```



Adding more points/lines on an already draw plot

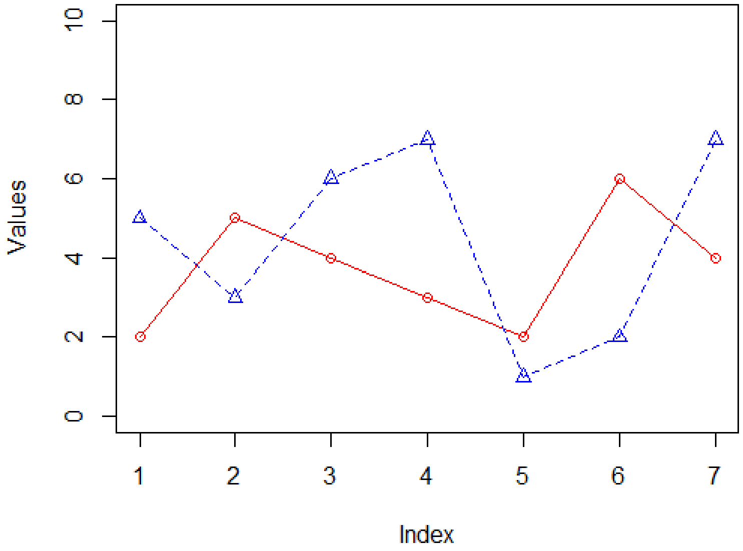
- □ If you want to add more points/lines on a plot you have drawn, you can use either `points()` or `lines()` function.

- □ The two functions are essentially the same but with different default value for `type`.

- □ You can modify the features of plotted points/lines similar to `plot()`.

Example

```
x <- c(2,5,4,3,2,6,4)
y <- c(5,3,6,7,1,2,7)
plot(x, col="red", type="o", ylim=c(0,10), ylab="Values")
points(y, col="blue", type="o", lty=2, pch=2)
```



Adding legends

- Use `legend()` function to add legends after drawing the plot.

- The important arguments are:
  - `x`, `y`:
    - The x and y position of the top left of the legend box.
    - You can also set `x="topright"`, `"topleft"`, `"bottomright"`, or `"bottomleft"`.
  - `legend`:
    - Names to display.
  - `horiz`:
    - Set legend in column or in row. If in row, set `horiz=TRUE`.
  - `text.col`:
    - Text color
  - `inset`:
    - To draw the legend away from x and y axis. Must be between 0 and 1

Adding legends

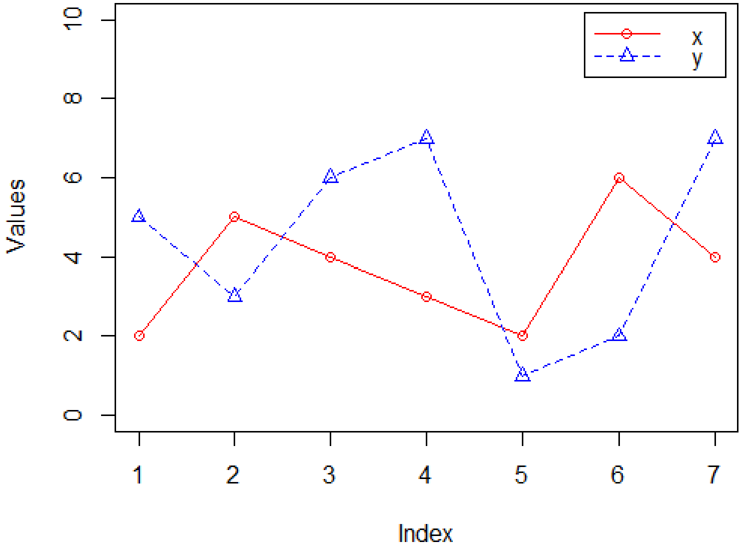
- You must specify the line type, the plot point character, line width, colours, and other characteristics the lines/points for the legend.

- You can also modify the box line type, color, etc.

- If there are multiple legends, use vector form to specify the legend for each lines/points.

Example

```
plot(x, col="red", type="o", ylim=c(0,10), ylab="Values")
points(y, col="blue", type="o", lty=2, pch=2)
legend(x = "topright",             # put the legends at top right
       inset = 0.02,               # use 2% inset
       legend = c("x","y"),        # legends for each lines/points
       col = c("red","blue"),      # colours for each lines/points
       lty = c(1,2),               # line type for each lines/pts
       pch = c(1,2),               # point characters
       )
```



Other things you can add

Universiti Kebangsaan Malaysia
The National University of Malaysia

- Texts – `text()`.
- Straight lines – `abline()`.
- Arrows – `arrows()`.
- Rectangles – `rect()`.
- Circle – `draw.circle()` (using `plotrix` package).

Commonly drawn figures

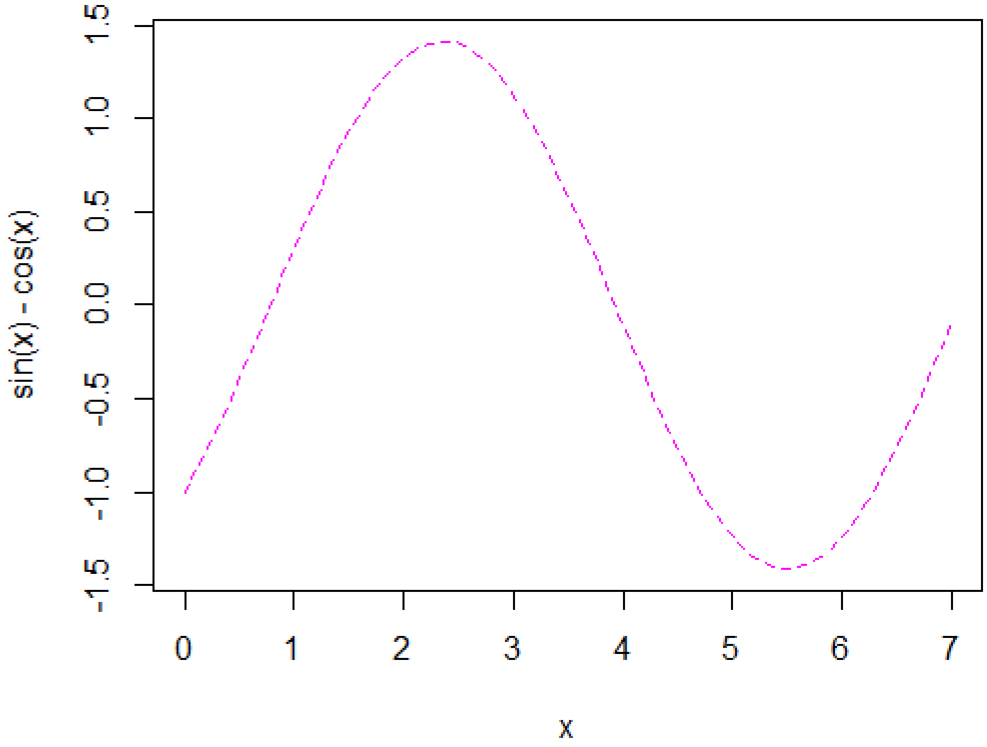
Plotting a function

- To draw a function $y = f(x)$, we can use the `curve()` function in R.

- Syntax: `curve(expression, from, to, n, add, ...)`
  - `expression`: the expression of the function of x.
  - `from`, `to`: the range over which the function will be plotted.
  - `n`: number of x values to be evaluated. The default value is 101.
  - `add`: if `TRUE`, it will draw the function on existing plot. The default is `FALSE`.

Example (plotting a function)

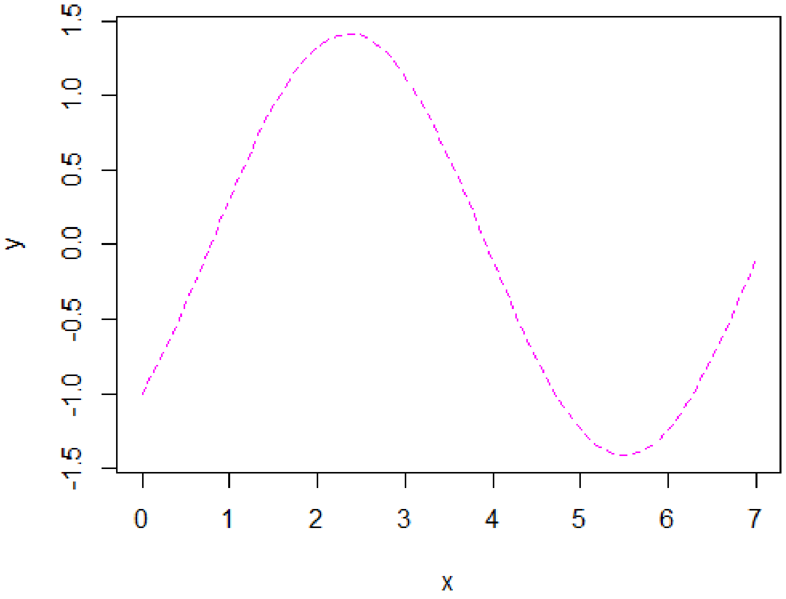
```
curve(sin(x)-cos(x), from=0, to=7, col="magenta",
      lty=2)
```



Example (plotting a function)

- Note: the previous code is equivalent to

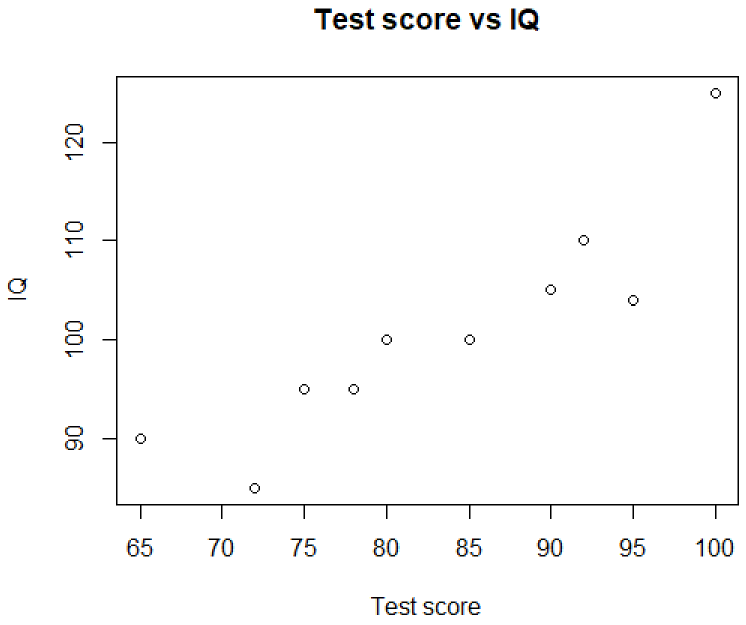
```
x <- seq(from=0, to=7, length.out=101)
y <- sin(x)-cos(x)
plot(x, y, type='l', lty=2, col="magenta")
```



Scatterplot

- To draw a scatterplot, simply use `plot()` function with x and y arguments.

```
TestScore <- read.csv(...)
plot(x=TestScore$Score, y=TestScore$IQ,
     main="Test score vs IQ",
     xlab="Test score", ylab="IQ")
```



Scatterplot

- An alternative way to write the code is by using `formula` class object for the argument.

- Formula object is a variable that specifies a relationship between other variables.

- Example:

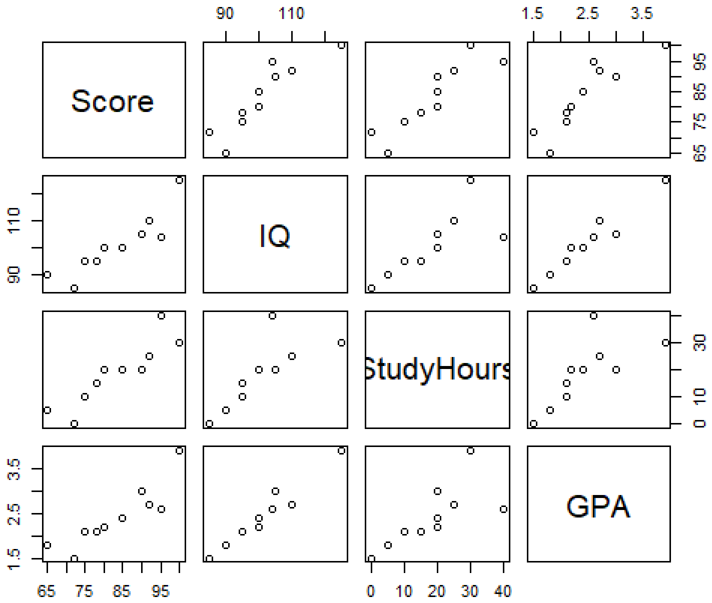
```
plot(IQ~Score, data=TestScore,
     main="Test score vs IQ",
     xlab="Test score", ylab="IQ")
```

Scatterplot matrix

- To draw a scatterplot matrix, use the `pairs()` function.

- Example:

```
pairs(~Score+IQ+StudyHours+GPA, data=TestScore)
```



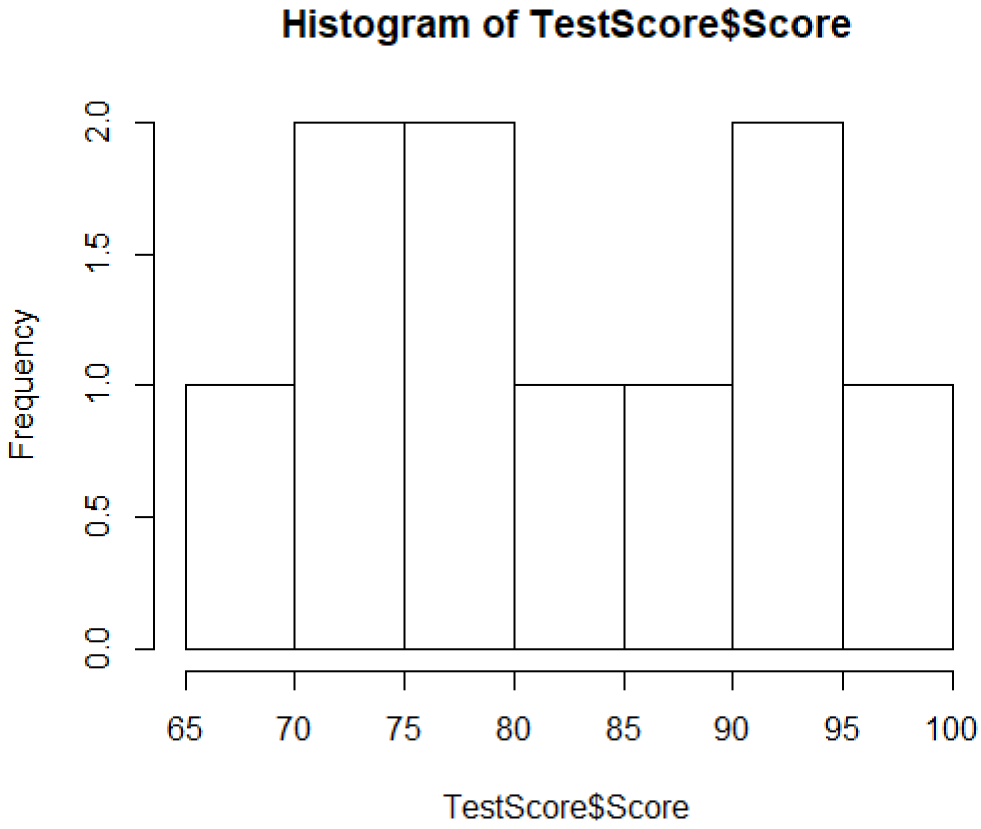
Histogram

- To draw a histogram, use `hist()` function.

- Syntax: `hist(x, breaks, freq, ...)`
  - `x`: a vector of values for which the histogram is desired.
  - `breaks`: how many "breaks" you want the histogram to bin the data. There are few ways to specify this argument:
    - a single number giving the number of bins for the histogram
    - a vector giving the breakpoints between bins
    - a character string naming an algorithm to compute the number of bins
  - `freq`: `freq=TRUE` if you want the y-axis to be the frequency for each bin. `freq=FALSE` if you want the y-axis to be the relative frequency or probability for each bin.

Example (histogram)

```
hist(TestScore$Score)
```

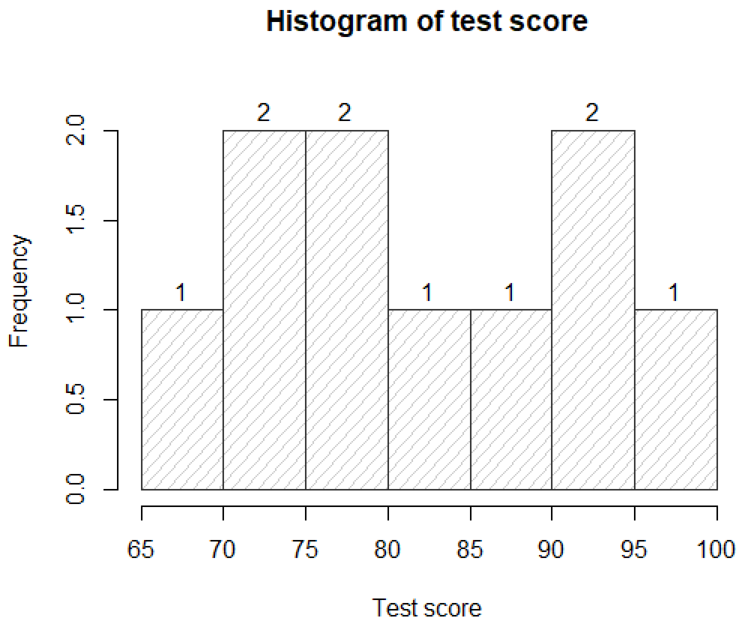


Some visual options for histogram

- Shading lines: `density`, `angle`.
  - Add diagonal lines to shade the bars.
  - The `density` value is a number indicating how many lines per inch R should draw (the default value of `NULL` means no lines).
  - The `angle` is a number indicating how many degrees from horizontal the lines should be drawn at (default is `angle=45` degrees).
- Specifics regarding colours: `col`, `border`.
  - The `col` parameter sets the colour of the shading.
  - The `border` argument sets the colour of the edges of the bars.
- Labelling the bars: `labels`.
  - Attach labels to each of the bars.
  - If `labels=TRUE`, R will add a number of observations above each bar.
  - Alternatively, you can choose the labels yourself, by inputting a vector of strings, e.g., `labels=c("label 1","label 2","etc")`

Example (histogram)

```
hist(TestScore$Score,
     main = "Histogram of test score",  # title of the plot
     xlab = "Test score",               # set the x-axis label
     density = 10,                      # draw shading lines: 10 per inch
     angle = 40,                        # set the angle of the shading lines
     border = "gray20",                 # set the colour of the borders of the bars
     col = "gray80",                    # set the colour of the shading lines
     labels = TRUE,                     # add frequency labels to each bar
     ylim = c(0,2.2)                    # change the scale of the y-axis
     )
```

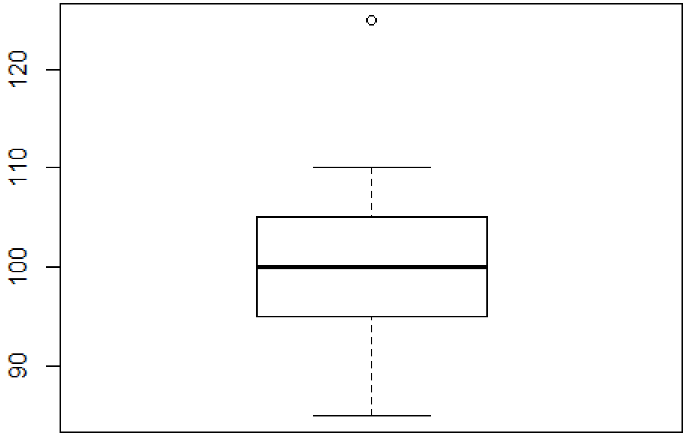


Boxplot

- □ Use `boxplot()` to draw the boxplot.

- □ Example::

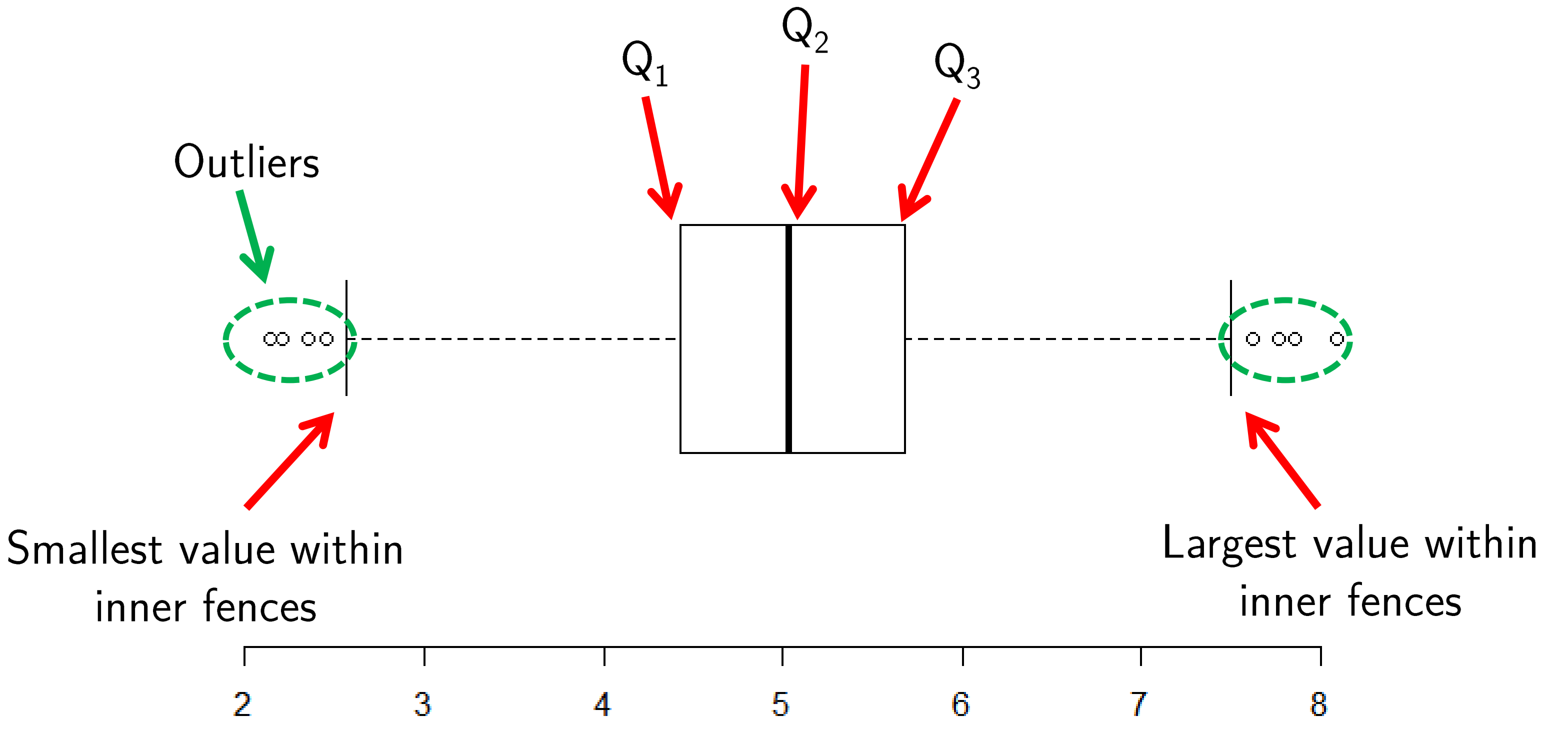
```
boxplot(TestScore$IQ)
```



- □ Note: You can use the argument `horizontal=TRUE` if you want the boxplot to be horizontal.

Boxplot

- Inner fences:
  - Lower inner fence = $Q_1 - 1.5 \times IQR$
  - Upper inner fence = $Q_3 + 1.5 \times IQR$

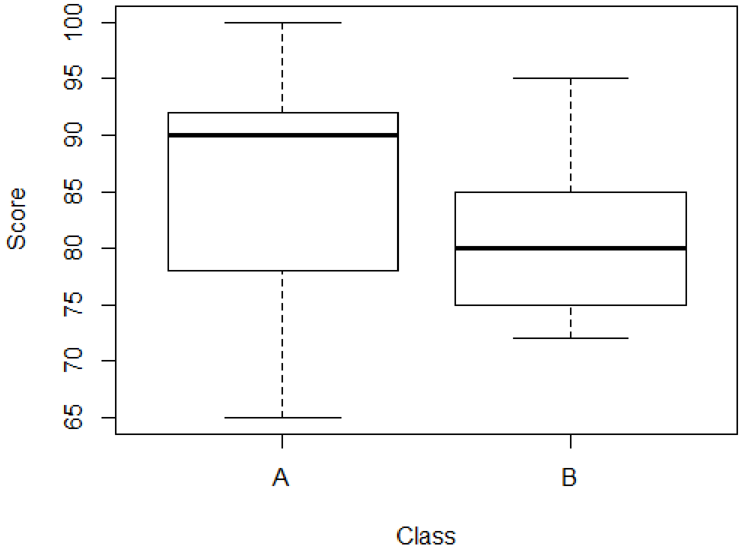

Q_1
Q_2
Q_3
Outliers
Smallest value within inner fences
Largest value within inner fences
2
3
4
5
6
7
8

Multiple boxplots

- You can draw multiple boxplots in the same figure using the `formula` class object.

- Example:

```
boxplot(Score~Class, data=TestScore)
```

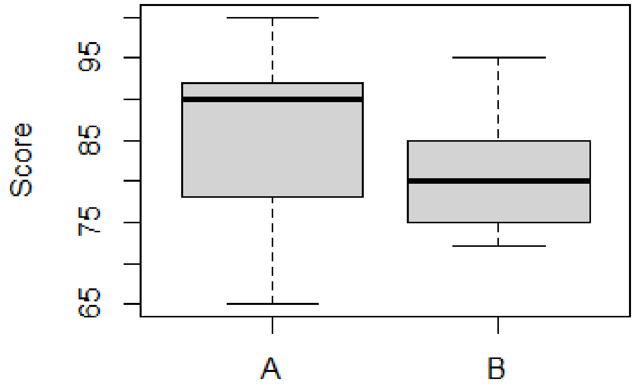


Multiple boxplots

- Alternatively, specifying more than one vector will draw a side-by-side boxplot.

- Example:

```
ScoreA = TestScore$Score[TestScore$Class=="A"]
ScoreB = TestScore$Score[TestScore$Class=="B"]
boxplot(ScoreA, ScoreB, names = c("A","B"),
        ylab="Score")
```



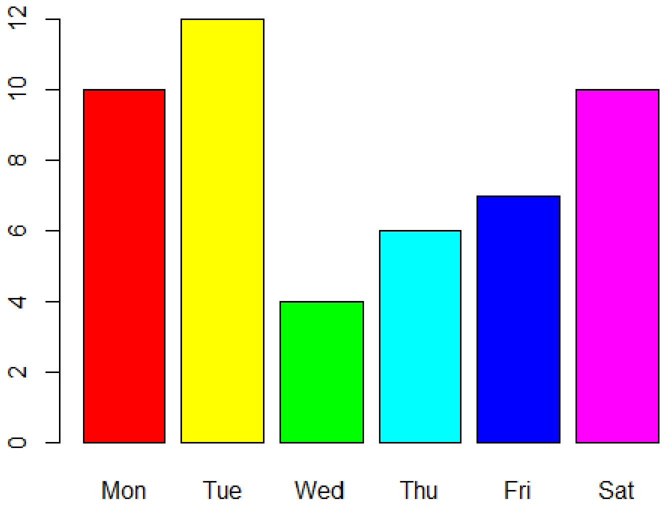
Bar graph

- Use `barplot()` function.

- Syntax: `barplot(height, names.arg, ...)`
  - `height`: the height for each bar.
  - `names.arg`: a vector of names for each bar

Example (bar graph)

```
x <- c(10,12,4,6,7,10)
barplot(height=x,
        names.arg=c("Mon","Tue","Wed","Thu","Fri","Sat"),
        col=rainbow(6)
        )
```



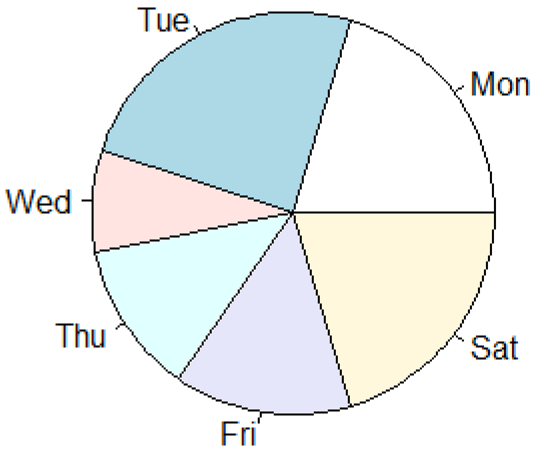
Pie chart

- Use `pie()` function.

- Syntax: `pie(x, labels, ...)`
  - `x`: a vector of non-negative numerical quantities. The values in x are displayed as the areas of pie slices.
  - `labels`: the labels for each pie slice.

Example (pie chart)

```
x <- c(10,12,4,6,7,10)
pie(x=x, labels=c("Mon","Tue","Wed","Thu","Fri","Sat"))
```



A few notes on drawing images

Saving image

- □ Saving images using RStudio is easy. Just click the export button and select the desired output file.
- □ But saving images using default R application (i.e. not RStudio) is kind of complicated.
- □ The function `dev.print()` can be used if you want to save the images using the console.

Example (saving image)

```
boxplot(Score~Class, data=TestScore)
dev.print(device = png,                # what are we printing to?
          filename = "boxplot.png",    # name of the image file
          width = 4,                   # how wide should it be
          height = 3,                  # how high should it be
          units = "in",                # what the units are
          res = 300                    # the resolution or dpi
          )
dev.off()                              # shuts down the device
```

Specifying diagram dimension

- You can create a new window for the diagram by using `dev.new()` function.

- This can be useful for functions that depends on the dimension of the diagram window for example `legend()`.

- Example:

```
dev.new(width=6, height=5, unit="in", noRStudioGD=TRUE)
```

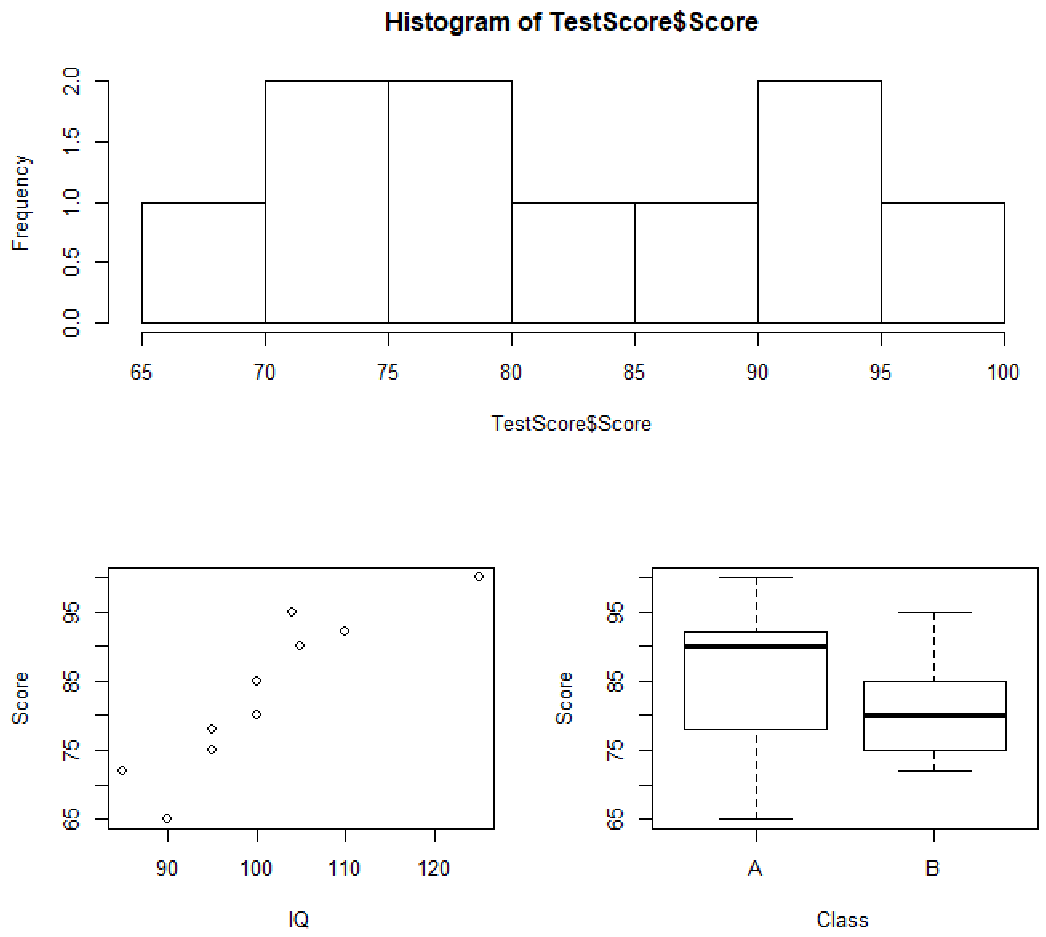
Multiple plots in the same figure

- You can change the number of plots in the figure by using the `layout()` function.

- Example:

```
layout(matrix(c(1,1,2,3),nrow=2,byrow=TRUE)) # set the layout
hist(TestScore$Score)                        # first plot
plot(Score~IQ, data=TestScore)               # second plot
boxplot(Score~Class, data=TestScore)         # third plot
layout(1)                                    # reset layout
```

Multiple plots in the same figure



Multiple plots in the same figure

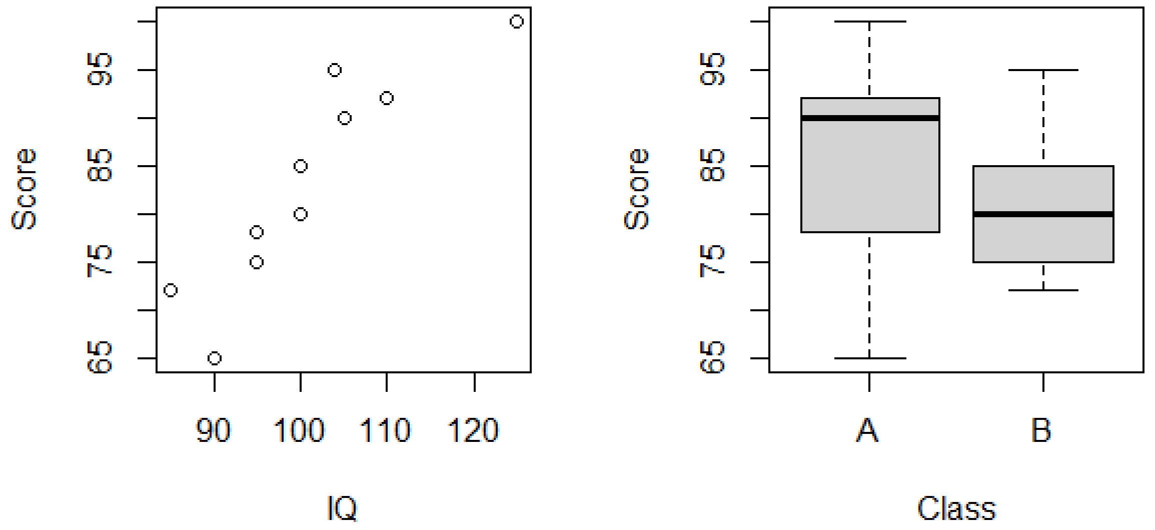
- Alternatively, you can change the `mfrow` or `mfcol` graphical parameter.

- Running `par(mfrow = c(a,b))` will give a a-by-b matrix-like figure.

- Example:

```
par(mfrow = c(1,2))                       # set the layout
plot(Score~IQ, data=TestScore)            # first plot
boxplot(Score~Class, data=TestScore)      # second plot
par(mfrow = c(1,1))                       # reset layout
```

Multiple plots in the same figure



Functions & descriptions

Function	Description
`plot()`	The basic plot function. Can be used to draw points or lines.
`points()`, `lines()`	Add points or lines to an existing figure.
`legend()`	Add legends to labels each point/line
`curve()`	Can be used to draw a function of x.
`pairs()`	Used to draw scatterplot matrix.
`hist()`	Used to draw histogram.
`boxplot()`	Used to draw boxplot.
`barplot()`	Used to draw bar graph.
`pie()`	Used to draw pie chart.
`layout()`	Used to set the layout of the figure for multiple plots in one figure.

Plot arguments

UNIVERSITI KEBANGSAAN MALAYSIA
The National University of Malaysia

Function	Description
`main`	Title of the diagram.
`xlab`	Label for x-axis.
`ylab`	Label for y-axis.
`xlim`	Vector specifying the limit for the x-axis.
`ylim`	Vector specifying the limit for the y-axis.
`type`	Type of plot. "`l`" for line, "`p`" for points, etc.
`col`	Colour of the line/points.
`lty`	Line type (solid/dashed/dotted/etc).
`lwd`	Line width.
`pch`	Plot point character.
`cex`	Size of plot points.