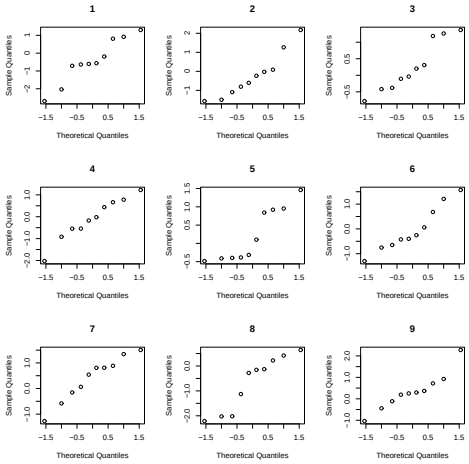
Multiple Graphs on a Page

- `par(mfrow=c(`$nrow$`,`$ncol$`))`
  `par(mfcol=c(`$nrow$`,`$ncol$`))`

  Lay out next $nrow \times ncol$ plots in an array. With `mfrow`, fill in row-by-row; with `mfcol`, fill in col-by-col.

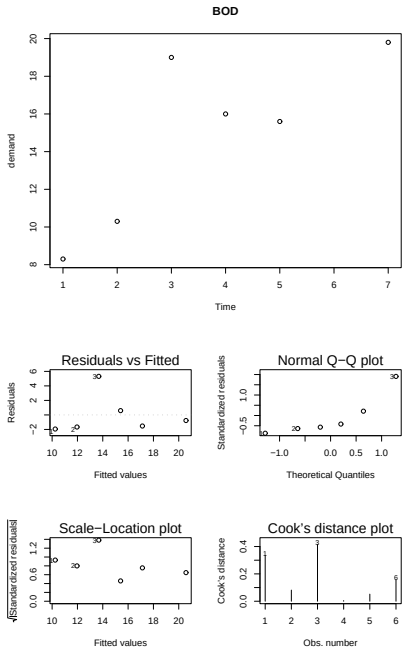
```
> par(mfrow=c(3,3))
> for (i in 1:9) qqnorm(rnorm(10), main=i)
> par(mfrow=c(1,1))  # go back to normal
```



Multiple Graphs on a Page

- `layout(...)`

```
> layout(rbind(c(1,1),c(2,3),c(4,5)),heights=c(2,1,1))
> with(BOD, plot(Time, demand, main="BOD"))
> plot(lm(demand ~ Time, data=BOD))
```



Graphical Parameters: `par(...)`

See `help(par)` for detailed explanations of graphical parameters including `col`, `lty`, etc. Also, use `par(...)` to get and globally set graphical parameters:

```
> par()
$adj
[1] 0.5
[ . .  . and 72 more . . .]
> par("pch","lwd")
$pch
[1] 1

$lwd
[1] 1
> par(pch=0, lwd=3)
>
```

From now on (until the graphics device is reset), default plot character is a square and all lines are triple-thick. To make a semi-permanent change:

```
> opar <- par(mfrow=c(2,2), cex=2, lty="dashed")
> opar
$mfrow
[1] 1 1

$cex
[1] 0.83

$lty
[1] "solid"
> [ . . . do some plots . . . ]
> par(opar)  # back to normal
>
```

Note: an important one is `par(ask=T)`.

Graphics Devices

R has a list of graphics devices:

```
> dev.list()       # just after starting R
NULL
> plot(rnorm(100)) # will auto-open a new window
> dev.list()
X11
  2
> X11()            # create another X11 window
> dev.list()
X11 X11
  2   3
> hist(rgamma(100,1,2))
> dev.off()        # close hist window
> dev.off()        # close original window
```

or

```
> graphics.off()   # close all graphics devices
```

Devices besides `X11()` are available:

```
> qqnorm(rnorm(10), main="Appears on screen")
> postscript("myplots.ps")
> qqnorm(rnorm(10), main="First page of PostScript file")
> qqnorm(rt(10,5), main="Second page of PostScript file")
> dev.off()
X11
  2
> qqnorm(rnorm(100), main="Back to the screen")
>
```

See `help(postscript)` and `help(pdf)` for parameters.

Functions

- Displaying functions:

```
> qqplot
function (x, y, plot.it = TRUE, xlab = deparse(substitute(x)),
    ylab = deparse(substitute(y)), ...)
{
    sx <- sort(x)
    sy <- sort(y)
    lenx <- length(sx)
    leny <- length(sy)
    if (leny < lenx)
        sx <- approx(1:lenx, sx, n = leny)$y
    if (leny > lenx)
        sy <- approx(1:leny, sy, n = lenx)$y
    if (plot.it)
        plot(sx, sy, xlab = xlab, ylab = ylab, ...)
    invisible(list(x = sx, y = sy))
}
<environment: namespace:stats>
>
```

- Defining functions:

```
> f <- function() { print("Hi, I'm a function!") }
> class(f)
[1] "function"
> f
function() { print("Hi, I'm a function!") }
> f()
[1] "Hi, I'm a function!"
>
```

- Editting functions:

```
> options(editor="pico")
> fix(f)
```

  If you get "stuck" in `vi`, the default editor, use `:q!`⟨Enter⟩ to escape.

Defining Functions

An example:

```
f <- function(x, y, diag=FALSE)
{
   l <- lm(y ~ x)
   opar <- par(ask=T)
   plot(x, y)
   abline(l)
   if(diag)
     plot(l)
   par(opar)
   l   # last expression is return value of f
}
```

Things to note:

- zero or more named arguments, optionally with default values;
- one or more expressions in the function "body" (that may reference the named arguments);
- the last function call or expression, to be used as the value of the function call (except if the function is terminated early using `return(value)`).

```
> l1 <- f(forbes$bp, forbes$pres, diag=T)
[ . . . shows plot of fit and diag plots . . . ]
> coef(l1)
(Intercept)           x
-81.0637271   0.5228924
>
```


Arguments

Given the function from the last slide:

```
f <- function(x, y, diag=FALSE)
{
  . . .
}
```

consider:

```
> f()
Error in eval(expr, envir, enclos) :
Argument "y" is missing, with no default
> f(forbes$bp, forbes$pres)              # uses default diag=FALSE
> f(forbes$bp, forbes$pres, F)           # same effect
> f(x=forbes$pres, diag=F, y=forbes$bp)  # any order okay if explicit
> f(forbes$pres, x=forbes$bp, diag=F)    # more complicated example
>
```

The actual rules for argument assignment are a little complicated. You don't have to worry about them if you call functions like this:

```
> text(c(1,2,3),rep(0,3),c("a","b","c"), pos=1, offset=0.75,cex=0.7)
```

with initial unnamed arguments assigned in order and extra, named arguments (if any) following the unnamed arguments.