A well constructed graph is worth a thousand words.

Many people use R mainly for obtaining effective graphs.

I usually use functions to build up the graph I want.

That way I can reuse them for other but similar situations, say for different data sets.

I have used it to make technical drawings for illustrations.

You can annotate graphs in many ways.

You can even use mathematical expressions in annotations.

There are many generic plot commands.

Many further commands add graphics elements to plots.

We illustrate by example, use documentation to explore more.

See also: R Graphics by Paul Murrell, Chapman & Hall/CRC.
Scatter Plot: `plot(faithful)`

RStudio saves plots in various formats: ⇒ Plots ⇒ Export
faithful is a data frame with 2 columns: eruptions and waiting

From the data frame nature of 2 columns the plot command knows to plot one column against the other.

Normal usage is plot(x, y) with x and y numerical vectors of equal length.

Note the resulting difference in the following commands

plot(faithful[,1], faithful[,2])

plot(faithful[,1], faithful[,2],
   xlab="eruption length (min)",
   ylab="waiting time to next eruption (min)"
plot (faithful[,1], faithful[,2])
plot(faithful[,1], faithful[,2]),
xlab = "eruption length (min)", ylab = "waiting time to next eruption (min)"
fit <- lm(waiting ~ eruptions, data=faithful)
plot(faithful[,1],faithful[,2],
     xlab="eruption length (min)",
ylab="waiting time to next eruption (min)"
) 
lines(faithful$eruptions,fitted(fit),
      col="red",lwd=2,lty=4) 
abline(fit)

- **lm** fits waiting as linear function of eruptions
- **data=faithful** indicates reference data set, avoids using faithful$waiting, etc.
- **fitted(fit)** gives the fitted values (y-coordinates).
- **abline(fit)** draws a straight line with a and b from fit.
Faithful Eruptions and Fitted Line: The Plot

![Graph showing eruption length (min) on the x-axis and waiting time to next eruption (min) on the y-axis. The data points are scattered and there is a fitted line indicating a positive correlation between the two variables.](image-url)
plot(0, 0, xlim = c(-10, 10), ylim = c(-10, 10),
    type = "n", xlab = "", ylab = "")
for(i in 1:20) abline(i - 10, 1, lty = i)
lty = 0 = blank, 1 = solid (default), 2 = dashed,
     3 = dotted, 4 = dotdash, 5 = longdash, 6 = twodash
Using Previously Importe ReactionTime Data Frame

head(ReactionTime)

<table>
<thead>
<tr>
<th>Reaction</th>
<th>Station</th>
<th>Shift</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>86</td>
<td>ST64  B</td>
</tr>
<tr>
<td>2</td>
<td>182</td>
<td>ST64  B</td>
</tr>
<tr>
<td>3</td>
<td>132</td>
<td>ST64  B</td>
</tr>
<tr>
<td>4</td>
<td>196</td>
<td>ST64  B</td>
</tr>
<tr>
<td>5</td>
<td>160</td>
<td>ST64  A</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>ST65  A</td>
</tr>
</tbody>
</table>

hist(ReactionTime$Reaction,breaks=seq(0,400,5))
# histogram bins cover [0,400] in increments of 5.

hist(ReactionTime$Reaction,breaks=seq(0,400,5),
main="EMS Turnout Time", xlab="time (seconds)",
col=c("blue","orange"))
```{r}
hist(ReactionTime$Reaction, breaks=seq(0,400,5))
```
hist(ReactionTime$Reaction, breaks=seq(0,400,5), main="EMS Turnout Time", xlab="time (seconds)", col=c("blue","orange"))
Plotting Symbols and Colors: `pcol <- c("green", "yellow", "purple", "red", "blue", "gold", "black", "cyan", "orange")`

```
plot(1:25, 1:25, pch=1:25, col=pcol)
```

See all 657 color names via `colors()`.
Character Plotting Symbols: `plot(1:25, 1:25, pch=letters[1:25], col=pcol)`
`points(1:25, 25:1, pch=rep(c("." , "*" ), length.out=25), col=pcol)`
x <- rnorm(100)
# x is a standard normal random sample, n=100

qqnorm(x, pch=16, cex=.5)
# makes QQ-plot of sample

> qqline(x)
# adds a fitted line to the previous plot.
# line is fitted through 1st and 3rd quartiles
```r
qqnorm(x, pch=16, cex=.5)
qqline(x)
```
Normal QQ-Plot \( n = 30 \)

\[
\text{par(mfrow=c(2,3))}
\]
\[
x <- \text{rnorm(30)}; \text{qqnorm(x)}; \text{qqline(x)}
\]
\[
x <- \text{rnorm(30)}; \text{qqnorm(x)}; \text{qqline(x)}
\]
\[
x <- \text{rnorm(30)}; \text{qqnorm(x)}; \text{qqline(x)}
\]
\[
x <- \text{rnorm(30)}; \text{qqnorm(x)}; \text{qqline(x)}
\]
\[
x <- \text{rnorm(30)}; \text{qqnorm(x)}; \text{qqline(x)}
\]
\[
x <- \text{rnorm(30)}; \text{qqnorm(x)}; \text{qqline(x)}
\]

- The `par` function controls many plotting parameters. 
  \[\Rightarrow \text{?par.}\]
- Some plotting parameters work within the plotting function, others only within a prior `par(...)` call.
- The `;` separation allows several commands on one line.
Judging Normality Takes Lots of Practice
Normal QQ-Plot $n = 100$

```r
par(mfrow=c(2,3))
x <- rnorm(100); qqnorm(x, pch=16, cex=.5); qqline(x)
x <- rnorm(100); qqnorm(x, pch=16, cex=.5); qqline(x)
x <- rnorm(100); qqnorm(x, pch=16, cex=.5); qqline(x)
x <- rnorm(100); qqnorm(x, pch=16, cex=.5); qqline(x)
x <- rnorm(100); qqnorm(x, pch=16, cex=.5); qqline(x)
x <- rnorm(100); qqnorm(x, pch=16, cex=.5); qqline(x)
```
Increasing $n$ to 100 Helps
boxplot(Reaction~Station+Shift, 
data=ReactionTime, cex.axis=.5)
The horizontal box lines ≡ 3 quartiles $Q(.25), Q(.5), Q(.75)$.
The dashed vertical lines extend to the adjacent values.
- Compute the interquartile range $IQR = Q(.75) - Q(.25)$.
- The upper adjacent value is the largest observation
  $\leq Q(.75) + 1.5 \times IQR$
- The lower adjacent value is the largest observation
  $\geq Q(.75) - 1.5 \times IQR$

Points beyond adjacent values shown individually (outliers?)
For $\mathcal{N}(\mu, \sigma^2) \approx .35\%$ are beyond each adjacent value.

`data=ReactionTime` ⇒ simpler reference to variables.

`Reaction ~ Station + Shift` implies boxplots for all factor combinations of `Station` and `Shift`. 
Many graphics functions allow fine tuning control as follows.

- Plot dimensions are controlled by `xlim=c(a,b)` and `ylim=c(c,d)`, using your `a,b,c,d`.

- Axis labels are controlled by `xlab="your x-label"` and `ylab="your y-label"`.

- Set the main plot title by `main="Your Main Title"`.

- Set the plot sub title by `sub="Your Sub Title"`.

- See `par` for many graphics control options, like
  - `cex`, `cex.axis`, `cex.main`, `cex.sub` character expansion factors.
  - `col`, `col.axis`, `col.lab`, `col.main`, `col.sub` specifying colors.
  - `font`, `font.axis`, `font.lab`, `font.main`, `font.sub` font choices, `1 = plain text (the default), 2 = bold face, 3 = italic and 4 = bold italic.`
Augmentation to Plots

- Some commands only work after a plot has been initiated.
- `abline(a,b)` draws line with intercept $a$ and slope $b$.
- `segments(...)` draws line segment(s) from $P_1$ to $P_2$.
- `arrows(...)` draws arrow(s) from $P_1$ to $P_2$.
- `lines(...)` draws curves through points by line segments.
- `points(...)` plots symbols (pch) at specified locations.
- `polygon(...), rect(...)` draw polygons and rectangles.
- `text(...)` puts specified text at selected positions.
- `legend(...)` adds legends to plots.
- `mtext(...)` adds text to plot margins.
- and lots more ⇒ `help.start()` ⇒ Graphics ⇒ `aplot`.
par(mfrow=c(1,3))

plot(LakeHuron,type="l",main='type="l"')
# points connected by lines

plot(LakeHuron,type="p",main='type="p"')
# only points are plotted

plot(LakeHuron,type="b",main='type="b"')
# both points and lines are plotted

# see ?plot for more on the type argument
Fisher’s or Anderson’s Iris Data

```r
str(iris)
'data.frame': 150 obs. of 5 variables:
$ Sepal.Length: num 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...  
$ Sepal.Width : num 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...  
$ Petal.Length: num 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1 ...  
$ Petal.Width : num 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...  
$ Species     : Factor w/ 3 levels "setosa","versi...  

unique(iris$Species)
[1] setosa versicolor virginica
Levels: setosa versicolor virginica

pairs(iris,col=rep(c("red","blue","orange"),each=50))
```
An Attempt to Visualize Higher Dimensional Data

Sepal.Length

Sepal.Width

Petal.Length

Petal.Width

Species
⇒ ?plotmath gives documentation on it.

> demo(plotmath) gives examples by commands and results.

"An approach to providing mathematical annotation in plots."
*Journal of Computational and Graphical Statistics*, 9, 582-599.
normalhist <- function(n=1000){
  x <- rnorm(n)
  xx <- seq(-4,4,.1)
  hist(x,breaks=xx,probability=T,
       main="normal histogram")
  yy <- dnorm(xx)
  lines(xx,yy,col="blue")
  text(-4,.3,expression(varphi(x)==
                        over(1,sqrt(2*pi))*phantom(0)*
                        e^{-x^2/2}),adj=0,col="blue")
}

\[ \phi(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2} \]
Saving Plots

- We indicated the interactive way within the RStudio interface.
- There are also various other ways by direct commands.
- `pdf(file="myplot.pdf", width=8, height=6)` opens pdf-file "myplot.pdf". width, height are in inches.
- Any subsequent graphics commands produce output to that file, until `dev.off()` is issued, or the R session terminates.
- Similar commands exist for other graphics formats
  ⇒ ?Devices
  for tiff, jpeg, bmp, png, postscript, quartz (Mac).
Add-on packages provide more graphics capabilities. We mention just three.

These are too complex to delve into here. Good as projects.

The *lattice* package.


The *ggplot2* package, not covered here, but see *R Graphics Cookbook* by Winston Chang, O’Reilly, 2013.