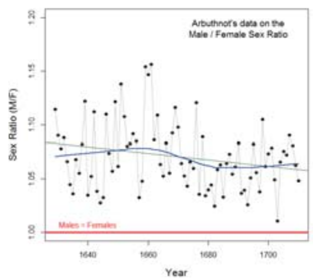
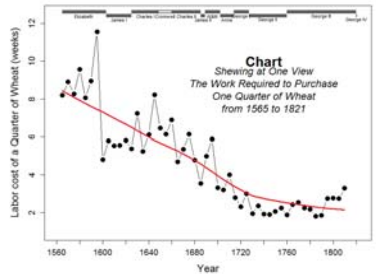




Data Visualization in R

2. Standard graphics in R

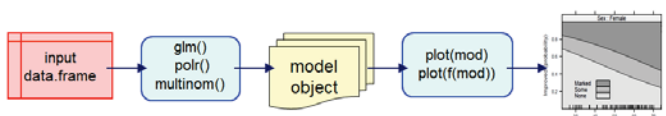
Michael Friendly
SCS Short Course
Sep/Oct, 2018
http://datavis.ca/courses/RGraphics/

Course outline

1. Overview of R graphics
2. Standard graphics in R
3. Grid & lattice graphics
4. ggplot2

Outline: Session 2

- Session 2: Standard graphics in R
  - R object-oriented design



  - Tweaking graphs: control graphic parameters
    - Colors, point symbols, line styles
    - Labels and titles
  - Annotating graphs
    - Add fitted lines, confidence envelopes
    - Add text, legends, point labels

R graphics systems

- Two graphics worlds
  - "graphics"– traditional or base graphics
  - "grid"– new style graphics
- Things work very differently in these
- Infrastructure for both is "grDevices" – the R graphics engine
  - Graphics devices,
  - colors, fonts

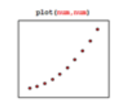
e.g.,
- the Cairo graphics device can create high-quality vector (PDF, PostScript and SVG) and bitmap output (PNG,JPEG,TIFF)
- the tikz device uses the LaTeX tikz package and LaTeX fonts, colors, etc.

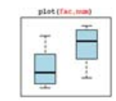
Base graphics functions: high & low

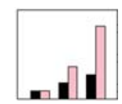
- Graphics functions are mostly one of two types:
  - High-level functions → complete plots
    - plot(), boxplot(), dotplot(), mosaicplot(), …
  - Low-level functions → add to an existing plot
    - lines(), points(), legend(), arrows(), polygon(), text()
  - Some functions can work either way, via an argument add=TRUE/FALSE
    - symbols(x, y, …, add=TRUE)
    - car::dataEllipse(x, y, add=TRUE, …)

The many faces of plot()

- plot() is the most important function in traditional graphics
- It is designed as a generic function, that does different things with numeric data (x, y), factors (FAC), matrices (MAT),…
  - plot(x) - index plot of x[i] vs I
  - plot(x, y) – scatterplot
  - plot(FAC, y) – boxplots
  - plot(x, FAC) – stripchart
  - plot(FAC, FAC) – spineplot, barchart
  - plot(MAT) – scatterplot matrix -> pairs()

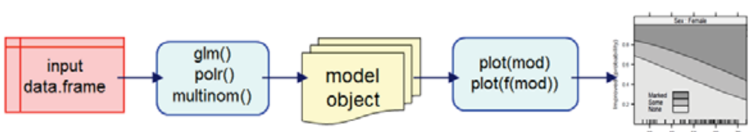






Object-oriented approach in R

- Everything in R is an **object**, and has a **class**
  - data sets: class "data.frame"
  - statistical models: class "lm", "glm", …



- Fit a model: obj <- lm(…) → a "lm" model object
  - print(obj) & summary(obj) → numerical results
  - anova(obj) & Anova(obj) → tests for model terms
  - update(obj), add1(obj), drop1(obj) model selection

Objects & methods

Method dispatch: The S3 object system

- Functions return objects of a given class
  - Anova/regression: lm() → an "lm" object
  - Generalized linear models: glm() → c("glm", "lm") – also inherits from lm()
  - Loglinear models: loglm() → a "loglm" object
- Class-specific methods have names of the form method.class
  - plot.lm(), plot.glm() – model diagnostic plots
- Generic functions– print(), plot(), summary() call the appropriate method for the class
  - plot(Effect(obj)) – calls plot.eff() effect plots
  - plot(influence(obj)) – calls plot.influence() for influence plots
  - plot(prcomp(obj)) – plots a PCA solution for a "prcomp" object

R objects & methods

```
> data(Duncan, package="car")
> class(Duncan)
[1] "data.frame"

> duncan.mod <- lm(prestige ~ income + education, data=Duncan)
> class(duncan.mod)
[1] "lm"

> print(duncan.mod)                                      → print.lm()

Call:
lm(formula = prestige ~ income + education, data = Duncan)

Coefficients:
(Intercept)       income    education
     -6.065        0.599        0.546

> Anova(duncan.mod)
Anova Table (Type II tests)                              → Anova.lm()

Response: prestige
          Sum Sq Df F value  Pr(>F)
income      4474  1    25.0 1.1e-05 ***
education   5516  1    30.9 1.7e-06 ***
Residuals   7507 42
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```


Objects & methods

Some methods for "lm" objects (in the base and car packages):

```
> library(car)
> methods(class="lm")
 [1] add1              alias             anova             Anova
 [5] avPlot            Boot              bootCase          boxCox
 [9] case.names        ceresPlot         coerce            confidenceEllipse
[13] confint           cooks.distance    crPlot            deltaMethod
[17] deviance          dfbeta            dfbetaPlots       dfbetas
[21] dfbetasPlots      drop1             dummy.coef        durbinWatsonTest
[25] effects           extractAIC        family            formula
[29] hatvalues         hccm              infIndexPlot      influence
[33] influencePlot     initialize        inverseResponsePlot kappa
[37] labels            leveneTest        leveragePlot      linearHypothesis
[41] logLik            mcPlot            mmp               model.frame
[45] model.matrix      ncvTest           nextBoot          nobs
[49] outlierTest       plot              powerTransform    predict
[53] print             proj              qqnorm            qqPlot
[57] qr                residualPlot      residualPlots     residuals
[61] rstandard         rstudent          show              sigmaHat
[65] simulate          slotsFromS3       spreadLevelPlot   summary
[69] variable.names    vcov
see '?methods' for accessing help and source code
```


Plot methods

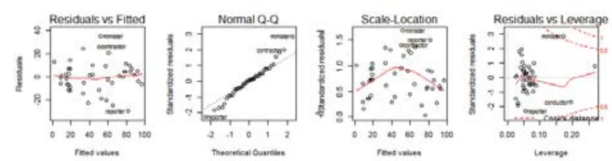
Some available plot() methods

```
> methods("plot")
 [1] plot.acf*              plot.ACF*              plot.augPred*
 [4] plot.coef.mer*         plot.compareFits*      plot.correspondence*
 [7] plot.data.frame*       plot.decomposed.ts*    plot.default
[10] plot.dendrogram*       plot.density*          plot.ecdf
[13] plot.factor*           plot.formula*          plot.function
[16] plot.gam*              plot.gls*              plot.hclust*
[19] plot.histogram*        plot.HoltWinters*      plot.intervals.lmList*
[22] plot.isoreg*           plot.jam*              plot.lda*
[25] plot.lm*               plot.lme*              plot.lmList*
[28] plot.lmList4*          plot.lmList4.confint*  plot.mca*
[31] plot.medpolish*        plot.merMod*           plot.mlm*
[34] plot.nffGroupedData*   plot.nfnGroupedData*   plot.nls*
[37] plot.nmGroupedData*    plot.PBmodcomp*        plot.pdMat*
[40] plot.powerTransform*   plot.ppr*              plot.prcomp*
[43] plot.princomp*         plot.profile*          plot.profile.nls*
[46] plot.qss1*             plot.qss2*             plot.ranef.lme*
[49] plot.ranef.lmList*     plot.ranef.mer*        plot.raster*
[52] plot.ridgelm*          plot.rq.process*       plot.rqs*
[55] plot.rqss*             plot.shingle*          plot.simulate.lme*
[58] plot.skewpowerTransform* plot.spec*           plot.spline*
[61] plot.stepfun           plot.stl*              plot.summary.crqs*
[64] plot.summary.rqs*      plot.summary.rqss*     plot.table*
[67] plot.table.rq*         plot.trellis*          plot.ts
[70] plot.tskernel*         plot.TukeyHSD*         plot.Variogram*
[73] plot.xyVector*
see '?methods' for accessing help and source code
```

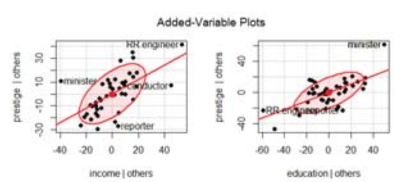

Some plot methods produce multiple plots.

You can control the layout with par() settings

```
op <- par(mfrow=c(1,4))    # change layout parameters
plot(duncan.mod)           # regression diagnostic plots
par(op)                    # restore old parameters
```




```
avPlots(duncan.mod, id.n=2, pch=16,
    ellipse=TRUE,
    ellipse.args=list(levels=0.68, fill=TRUE, fill.alpha=0.1))
```



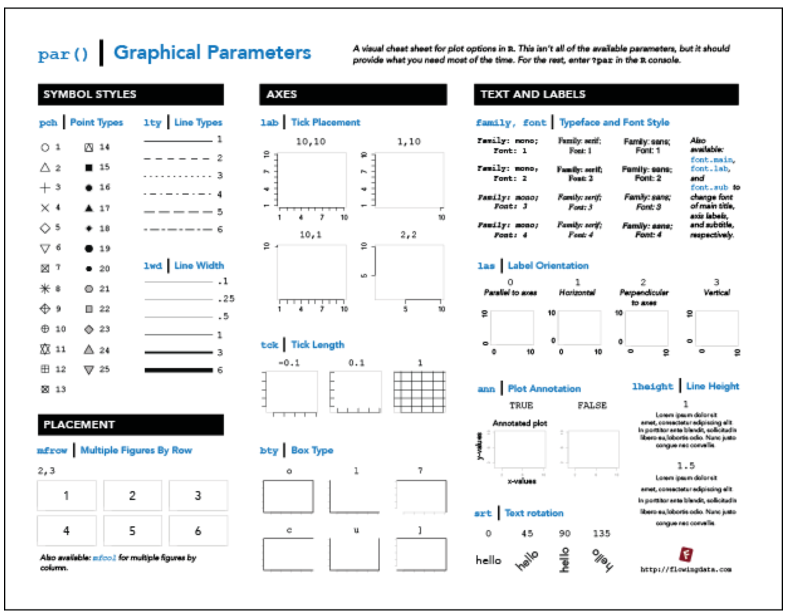
Some plot methods have lots of optional arguments for graphic enhancements.
Use help() for documentation

Graphic parameters

- All graphic functions take arguments that control details
  - colors (col=)
  - point symbols (pch=)
  - line type (lty=); line width (lwd=)
- Often these have default values in the function definition
  - col="black"; col=par("col"); col=palette()[1]
  - lwd=2, lty=1
- Most high-level graphic functions have a "…" argument that allow passing other arguments to the plotting functions

Graphic parameters

- Some graphics parameters can be set globally for all graphs in your session, using the par() function
  - par(mar=c(4,4,1,1)) – plot margins
  - par(cex.lab=1.5) – make axis labels 50% larger
  - par(cex=2) – make text & point symbols 2x larger
  - Graphics functions often use these as defaults
- Most can be set in calls to high-level plotting functions
  - avPlots(duncan.mod, pch=16, cex=2, cex.lab=1.5, …)

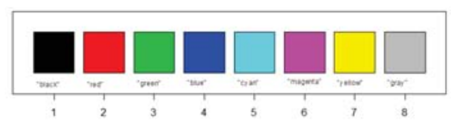


From: http://gastonsanchez.com/r-graphical-parameters-cheatsheet.pdf

Graphic parameters

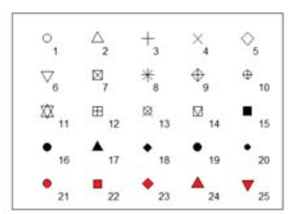
The most commonly used graphic parameters:

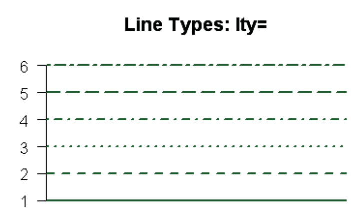
col



These colors are the default, palette()
R packages specifically related to color: colorspace, colorRamps, RColorBrewer, ..

pch

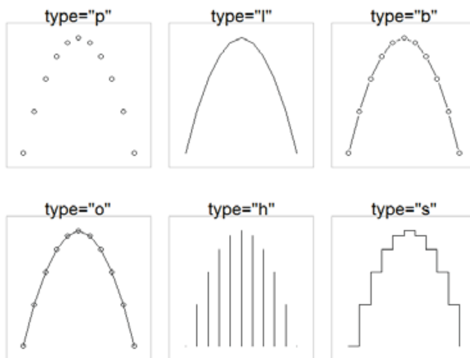




Plot types

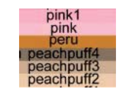
The functions plot(), points() and lines() understand a type= parameter and render the (x, y) values in different ways.

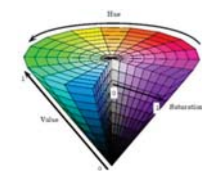
```
x <- -5:5
y <- -x^2 + 25
plot(x, y, type="p")
plot(x, y, type="l")
plot(x, y, type="b")
plot(x, y, type="o")
plot(x, y, type="h")
plot(x, y, type="s")
```



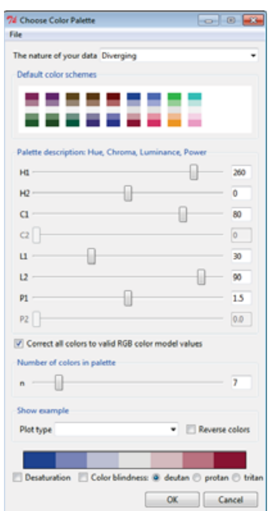
More on color

- Presentation graphs require careful choice of colors
  - Legible if copied in B/W?
  - Visible to those with color deficiency?
  - Mapping categorical or continuous variables to color scale
- R has a variety of ways to specify color
  - color names: see colors()
  - Hex RGB: red = "#FF0000" , blue="#0000FF"
  - with transparency: #rrggbbaa #0000FFA0 #0000FF80
  - hsv(): hue, saturation, value (better as perceptual model)
  - colorRamps: rainbow(n)

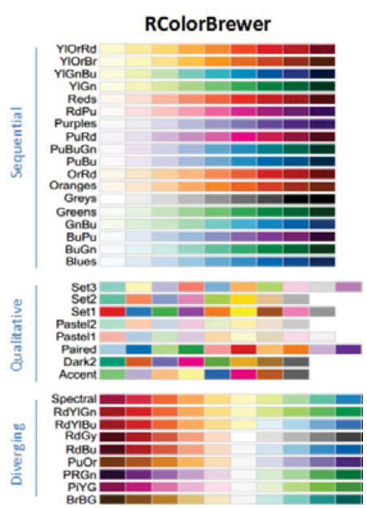




library("colorspace")
pal < - choose_palette()



library("RColorBrewer")
display.brewer.all()



See: https://www.nceas.ucsb.edu/~frazier/RSpatialGuides/colorPaletteCheatsheet.pdf

Traditional R graphics: mental model

- R graphics functions add ink to a canvas – the "painter's model"
  - new graphics elements overlay / obscure what is there before
  - only way to move or remove stuff is to re-draw in white (background color)
  - animated graphs re-do the whole plot in each frame
  - Transparent colors are often useful for filled areas
- Typically, create a graph with a high-level function, then add to it with low-level if desired
- I'll illustrate by re-constructing two historical graphs
  1. Identify the graphical elements: points, lines, text, …
  2. Start with a basic plot, then add to it

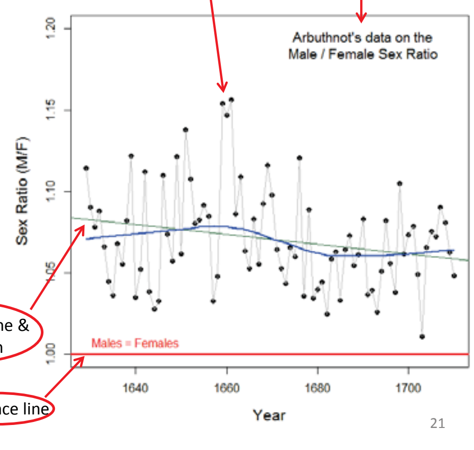
Building a custom graph

Custom graphs can be constructed by adding graphical elements (points, lines, text, arrows, etc.) to a basic plot()

John Arbuthnot: data on male/female sex ratios:

```
> data(Arbuthnot, package="HistData")
> head(Arbuthnot[,c(1:3,6,7)])
  Year Males Females Ratio Total
1 1629  5218    4683 1.114 9.901
2 1630  4858    4457 1.090 9.315
3 1631  4422    4102 1.078 8.524
4 1632  4994    4590 1.088 9.584
5 1633  5158    4839 1.066 9.997
6 1634  5035    4820 1.045 9.855
…  …     …       …     …     …
```

Arbuthnot didn't make a graph. He simply calculated the probability that in 81 years from 1629—1710, the sex ratio would **always** be > 1
The first significance test!



Follow along

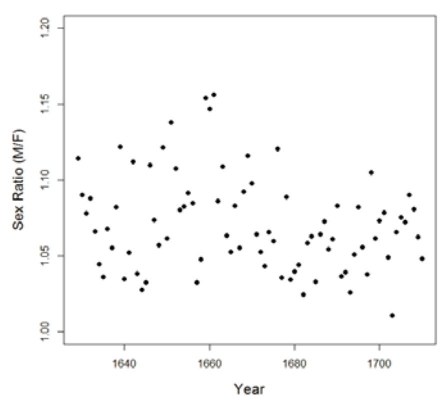
- From the course web page, click on the script arbutnot.R, http://www.datavis.ca/courses/RGraphics/R/arbuthnot.R
- Select all (ctrl+A) and copy (ctrl+C) to the clipboard
- In R Studio, open a new R script file (ctrl+shift+N)
- Paste the contents (ctrl+V)
- Run the lines (ctrl+Enter) to along with me

(You could instead save that file to your lab HOME directory and open it from there.)

Building a custom graph

```
plot(Ratio ~ Year, data=Arbuthnot,
      pch=16,
      ylim=c(1, 1.20),
      cex.lab = 1.3,
      ylab="Sex Ratio (M/F)")
```

1. Start with a basic plot of points

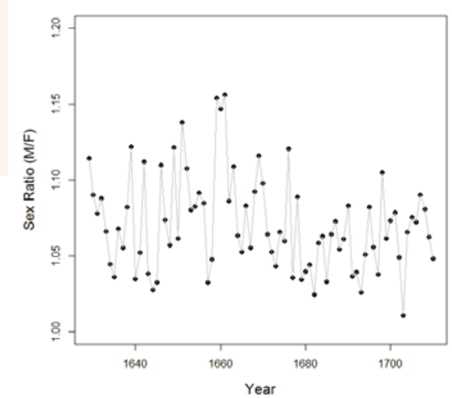


Code details:
pch: I like filled circles (16=●) for points
ylim: allow more vertical space for caption
cex.lab: make axis labels larger

Building a custom graph

```
plot(Ratio ~ Year, data=Arbuthnot,
      pch=16,
      ylim=c(1, 1.20),
      cex.lab = 1.3,
      ylab="Sex Ratio (M/F)")
# connect points by lines
lines(Ratio ~ Year, data=Arbuthnot, col="gray")
```

2. Add gray lines

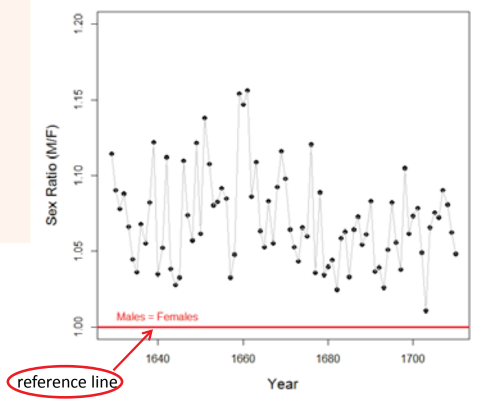


Code details:
I could have used type="b" or type="o"
But I wanted the lines in gray, not black

Building a custom graph

```
plot(Ratio ~ Year, data=Arbuthnot,
        pch=16,
        ylim=c(1, 1.20),
        cex.lab = 1.3,
        ylab="Sex Ratio (M/F)")
# connect points by lines
lines(Ratio ~ Year, data=Arbuthnot, col="gray")
# add reference line
abline(h=1, col="red", lwd=3)
text(1640, 1, "Males = Females", col="red")
```

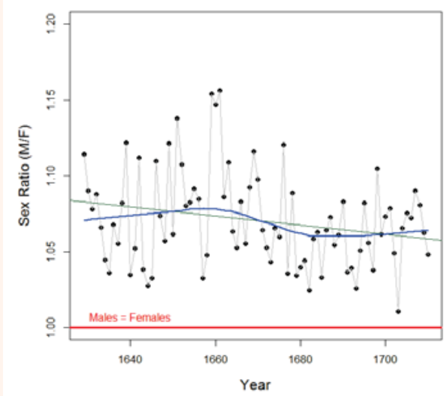
3. Add horizontal reference line & label



Building a custom graph

```
plot(Ratio ~ Year, data=Arbuthnot,
        pch=16,
        ylim=c(1, 1.20),
        cex.lab = 1.3,
        ylab="Sex Ratio (M/F)")
# connect points by lines
lines(Ratio ~ Year, data=Arbuthnot, col="gray")
# add reference line
abline(h=1, col="red", lwd=3)
text(1640, 1, "Males = Females", col="red")
# add linear regression line
abline(lm(Ratio ~ Year, data=Arbuthnot),
         col="darkgreen")
#  add loess smooth
Arb.smooth <- with(Arbuthnot,
                        loess.smooth(Year,  Ratio))
lines(Arb.smooth$x, Arb.smooth$y,
        col="blue", lwd=2)
```

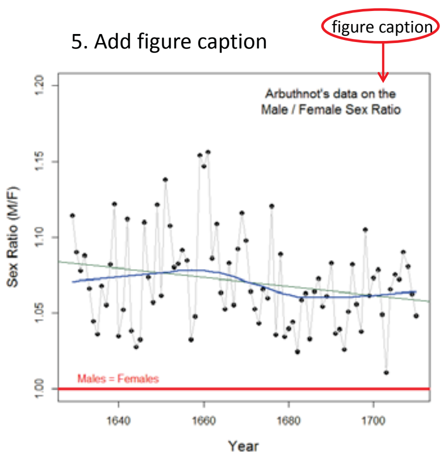
4. Add regression & smoothed lines



Building a custom graph

```
plot(Ratio ~ Year, data=Arbuthnot,
        pch=16,
        ylim=c(1, 1.20),
        cex.lab = 1.3,
        ylab="Sex Ratio (M/F)")
# connect points by lines
lines(Ratio ~ Year, data=Arbuthnot, col="gray")
# add reference line
abline(h=1, col="red", lwd=3)
text(1640, 1, "Males = Females", col="red")
# add linear regression line
abline(lm(Ratio ~ Year, data=Arbuthnot),
         col="darkgreen")
#  add loess smooth
Arb.smooth <- with(Arbuthnot,
                        loess.smooth(Year,  Ratio))
lines(Arb.smooth$x, Arb.smooth$y,
        col="blue", lwd=2)
# add internal figure caption
text(1690, 1.19, "Arbuthnot's data on the\nMale /
Female Sex Ratio", cex=1.2)
```

5. Add figure caption

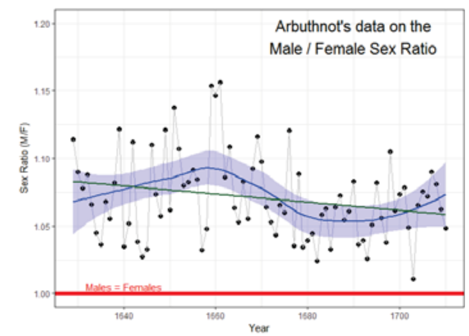


The same graph, using ggplot2

ggplot2 has a totally different idea about constructing graphs

The syntax adds elements and layers to a graph with functions connected with "+" signs.

Details in a following lecture



ggplot code:

```
library(ggplot2)
ggplot(Arbuthnot, aes(x=Year, y=Ratio)) +
    ylim(1, 1.20) +
    ylab("Sex Ratio (M/F)") +
    geom_point(pch=16, size=2) +
    geom_line(color="gray") +
    geom_smooth(method="loess", color="blue", fill="blue", alpha=0.2) +
    geom_smooth(method="lm", color="darkgreen", se=FALSE) +
    geom_hline(yintercept=1, color="red", size=2) +
    annotate("text", x=1640, y=1.005, label="Males = Females", color="red", size=4) +
    annotate("text", x=1690, y=1.19,
             label="Arbuthnot's data on the\nMale / Female Sex Ratio", size=6) +
    theme_bw()
```

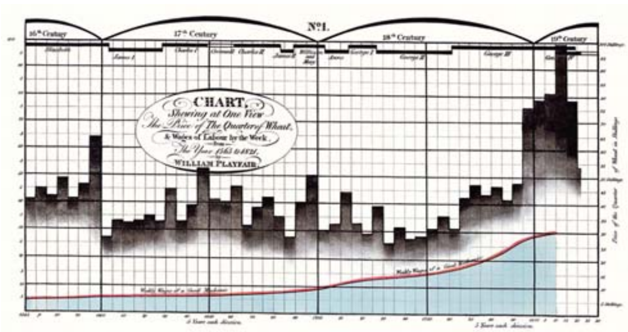
Playfair's wheat

William Playfair (1759—1836) invented most of the forms of modern data graphics: the bar chart, line graph and pie chart.

- This multivariate chart shows the price of wheat (bars), wages of a good mechanic (line graph), and the reigns of British monarchs over 250 years, 1565—1830
- Playfair's goal: Show that workers were better off now than at any time in the past.

Did Playfair succeed?

What can you read from this chart re: wages vs. price of wheat?



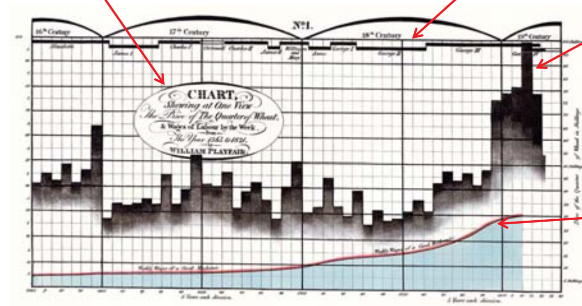
Reproducing Playfair's chart

To try to reproduce this chart:

- Identify the graphical elements: 3 time series, cartouche caption, grid lines, …
- Make a basic plot setting up (x,y) range, axis labels, …
- Use low-level functions to add graphical elements

Caption: plot using multiline text()

Monarchs: draw using segments(), label with text()



Time series of wheat: plot as step function: type="s"

Wages: draw using lines()

Reproducing Playfair's chart

Playfair's data was digitized from his chart. The HistData package records this as two data frames.

```
> str(Wheat)
'data.frame':   53 obs. of  3 variables:
 $ Year : int  1565 1570 1575 1580 1585 1590 1595 1600 1605 1610 ...
 $ Wheat: num  41 45 42 49 41.5 47 64 27 33 32 ...
 $ Wages: num  5 5.05 5.08 5.12 5.15 5.25 5.54 5.61 5.69 5.78 ...

> str(Wheat.monarchs)
'data.frame':   12 obs. of  4 variables:
 $ name        : Factor w/ 12 levels "Anne","Charles I",..: 5 10 2 4 3 11 12 1 6 7 ...
 $ start       : int  1565 1603 1625 1649 1660 1685 1689 1702 1714 1727 ...
 $ end         : int  1603 1625 1649 1660 1685 1689 1702 1714 1727 1760 ...
 $ commonwealth: int  0 0 0 1 0 0 0 0 0 0 ...
```

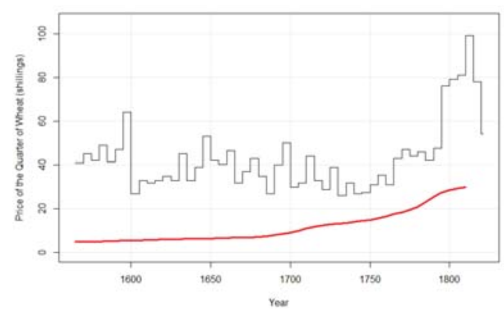
Code for this example: http://datavis.ca/courses/RGraphics/R/playfair-wheat.R

Reproducing Playfair's chart

```
with(Wheat, {
  plot(Year, Wheat, type="s", ylim=c(0,105),
     ylab="Price of the Quarter of Wheat (shillings)",
     panel.first=grid(col=gray(.9), lty=1))
  lines(Year, Wages, lwd=3, col="red")
  })
```

with(Wheat, { expressions })
makes the variables in Wheat available in evaluating the {expressions}

I could have used:
plot(Wheat ~ Year, data=Wheat, …)



The basic plot is a step-curve for wheat (type="s")

Add lines for Wages

The area beneath the curve could be filled, using polygon()

Reproducing Playfair's chart

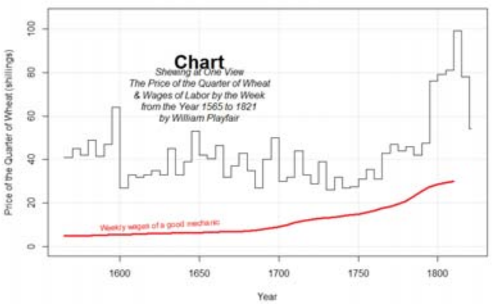
```
# label the curve of Wages
text(1625,10, "Weekly wages of a good mechanic", cex=0.8, srt=3, col="red")

# cartouche
text(1650, 85, "Chart", cex=2, font=2)
text(1650, 70,
   paste("Shewing at One View",
      "The Price of the Quarter of Wheat",
      "& Wages of Labor by the Week",
      "from the Year 1565 to 1821",
      "by William Playfair",  sep="\n"), font=3)
```

text label: srt=3 rotates the text 3°

paste(s1, s2, …, sep="\n") makes separate lines

font=3: italic



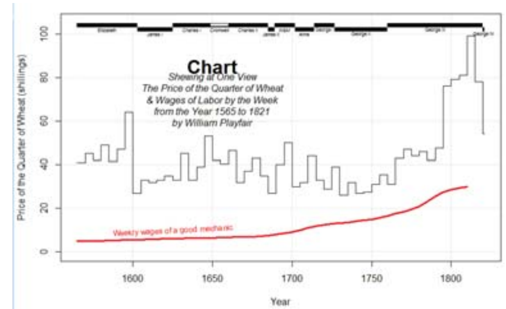
The decorative cartouche is drawn with text(), using a vector of strings

Reproducing Playfair's chart

```
with(Wheat.monarchs, {
   y <- ifelse( !commonwealth & (!seq_along(start) %% 2), 102, 104)
   segments(start, y, end, y, col="black", lwd=7, lend=1)
   segments(start, y, end, y, col=ifelse(commonwealth, "white", NA), lwd=4, lend=1)
   text((start+end)/2, y-2, name, cex=0.5)
   })
```

alternate y values as (102, 104)

re-draw the white one



The timeline for monarchs is drawn using segments()

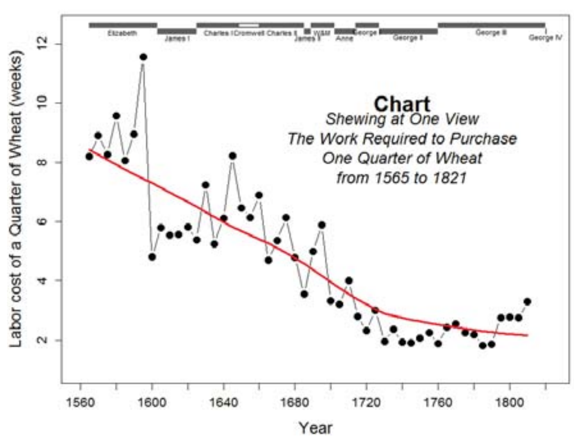
This part is tricky because I need to calculate the y value for each segment and position the labels accordingly.

Consulting for Playfair

WP: Can you help me make a better graph?
SCS: Yes, plot the ratio of Wheat / Wages : the labor cost to buy a quarter of wheat

This clearly shows that wheat was becoming cheaper in terms of the amount of labor required

Plotting data was so new that Playfair did not think of plotting a derived value.



Consulting for Playfair

Code for this plot:

calculate ratio

```
Wheat1 <- within(na.omit(Wheat), {Labor=Wheat/Wages})

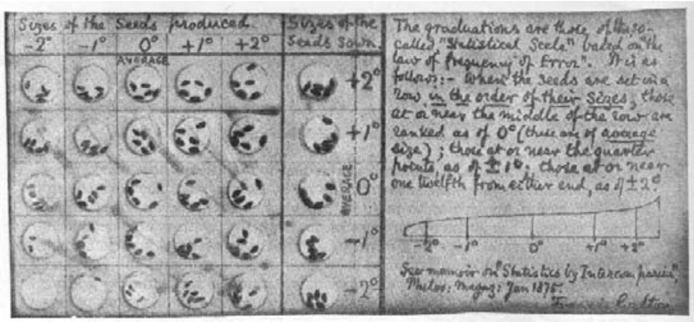
with(Wheat1, {
      plot(Year, Labor, type='b', pch=16, cex=1.5, lwd=1.5,
         ylab="Labor cost of a Quarter of Wheat (weeks)",
         ylim=c(1,12.5), xlim=c(1560,1823),
         cex.axis=1.2, cex.lab=1.5,
         lab=c(12,5,7)
         );
      lines(lowess(Year, Labor), col="red", lwd=3)
      })
```

The remainder of the code is similar to that for the original plot

Galton's peas

In 1875 Francis Galton studied heredity of physical traits. In one experiment, he sent packets of sweet peas of 7 different sizes to friends, and measured the sizes of their offspring.

His first attempt was a semi-graphic table, tabulating the number of parent-child seeds in each combination of values. He noted that both distributions followed the "law of frequency of error" (Normal distribution)



Galton's peas: The first regression line

Galton's (1877) presentation graph:

- Plotted mean diameter of child seeds vs. mean of parents
- Noticed these were nearly in a line– An "Ah ha" moment!
- The slope of the line said something about heredity

But, the slope of the line < 1

→ "reversion" toward mean

→ children of large/small parents less extreme than their parents

Later used the term "regression" for this phenomenon, and statistical explanation

Image: From K. Pearson, The Life, Letters and Labours of Francis Galton, Volume 3A, Chapter 14, Fig. 1

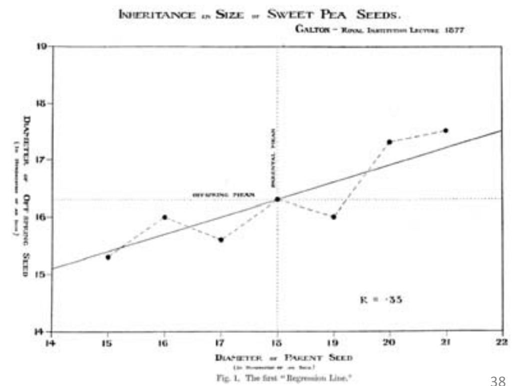


Fig. 1. The first "Regression Line."

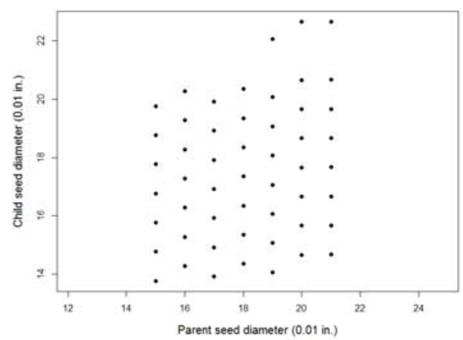
Galton's peas: Plotting discrete data

How Galton got there – the untold story
His friend, JFW Herschel said, "Why don't you make a scatterplot?"
He fired up R on his Babbage machine …
… but was initially disappointed in the result: too much overplotting

Another great R historical moment

```
data(peas, package="psych")

plot(child ~ parent, data=peas,
   pch=16, cex.lab=1.25,
   asp=1, xlim=c(14, 23),
   xlab="Parent seed diameter (0.01 in.)",
   ylab="Child seed diameter (0.01 in.)")
```



NB: Galton was careful to

- Set aspect ratio = 1
- Use explicit axis labels

Code: http://datavis.ca/courses/RGraphics/R/galton-peas.R

Galton's peas: Data wrangling

Galton thoughtfully met with an SCS consultant, who said: "Show me your data!!!"

```
> str(peas)
'data.frame':   700 obs. of  2 variables:
 $ parent: num  21 21 21 21 21 21 21 21 21 21 ...
 $ child : num  14.7 14.7 14.7 14.7 14.7 ...
```

SCS: Ah! your data are discrete.

SCS: Summarize them with dplyr

```
library(dplyr)
peas.freq <- peas %>%
    group_by(parent, child) %>%
    summarise( count=n() )
```

```
> peas.freq
Source: local data frame [52 x 3]
Groups: parent [?]

   parent child count
    <dbl> <dbl> <int>
1     15 13.77    46
2     15 14.77    14
3     15 15.77     9
4     15 16.77    11
5     15 17.77    14
6     15 18.77     4
7     15 19.77     2
8     16 14.28    34
9     16 15.28    15
10    16 16.28    18
# ... with 42 more rows
```

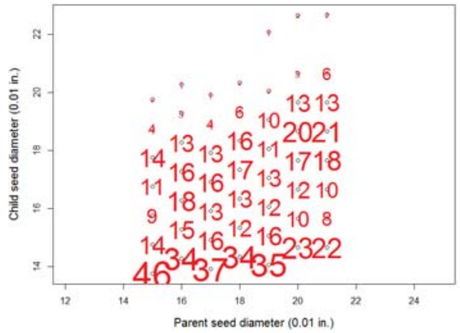
Galton's peas: a text-table plot

Galton: Ah! Maybe I'll just go back to my original table
SCS consultant: Good, but make it into a plot also: use text()
Here's a good graphic trick: make font size ~ f(n)

```
plot(child ~ parent, data=peas, ...)
with( peas.freq,
  text( parent, child,
    count, col="red", cex=log(count)))
```

text label

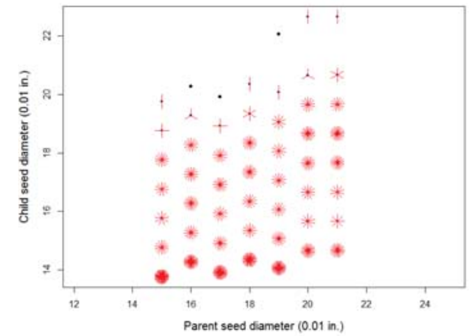
size ~ log(n)



Galton's peas: Sunflower plots

Perhaps better: use point symbols that show explicitly the number of observations at each (x, y) location
A sunflower plot uses symbols with the number of rays = # of obs at each (x, y)
Now, he could see the upward trend – sort of

```
sunflowerplot(child ~ parent, data=peas,
  pch=16, cex.lab=1.25,
  asp=1, xlim=c(14, 23),
  xlab="Parent seed diameter (0.01 in.)",
  ylab="Child seed diameter (0.01 in.)")
```

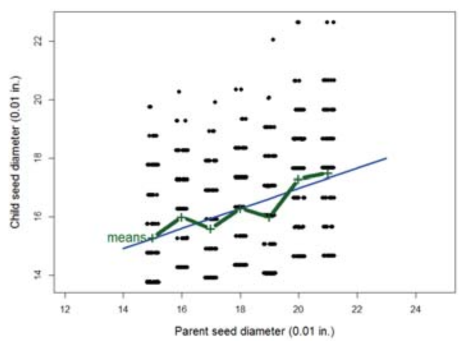


Galton's peas: jittering

Another possibility is to jitter() the plotted points by adding little random #s
But, he also needed to calculate and plot the line of means and the trend line

```
plot(jitter(child) ~ jitter(parent), data=peas,
  pch=16, cex.lab=1.25,
  asp=1, xlim=c(14, 23),
  xlab="Parent seed diameter (0.01 in.)",
  ylab="Child seed diameter (0.01 in.)")
```

```
# add line of means
means <- aggregate(child ~ parent, data=peas,
            FUN=mean)
lines (child ~ parent, data=means, type="b",
     pch="+", cex=2, lwd=7, col="darkgreen")
text(15, 15.3, "means", col="darkgreen", cex=1.4,
     pos=2)
# calculate & draw the regression line
peas.mod <- lm(child ~ parent, data=peas)
abline(peas.mod, lwd=2, col="blue")
```

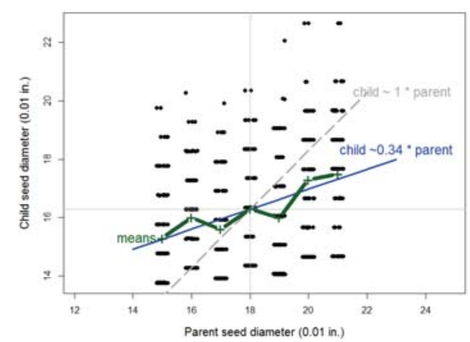


Plotting discrete data: Galton's peas

Making Galton's argument visually clearer:

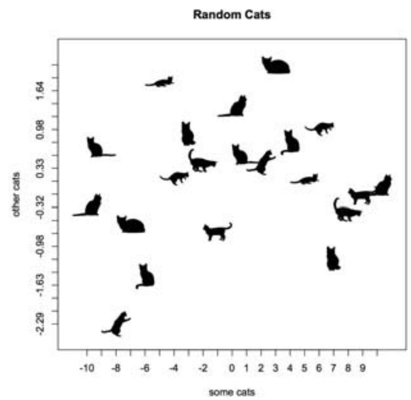
- Label the regression line with its slope
- Show the comparison line (slope=1) if there was no regression toward the mean

```
text(23, y=18.3, "child ~0.34 * parent",
     cex=1.4, col="blue")
# line of unit slope
mx <- mean(peas$parent)
my <- mean(peas$child)
xp <- 14:22
yp <- my + 1* (xp - mx)
lines(xp, yp, col="darkgray", lwd=3,
      lty="longdash")
text(23.2, yp[9], "child ~ 1 * parent",
     cex=1.4, col="darkgray")
```



Just for fun: CatterPlots

```
library(devtools)
install_github("Gibbsdavidl/CatterPlots")
# plot random cats
multicat(xs=-10:10, ys=rnorm(21),
    cat=c(1,2,3,4,5,6,7,8,9,10),
    catcolor=list(c(0,0,0,1)),
    canvas=c(-0.1,1.1, -0.1, 1.1),
    xlab="some cats",
    ylab="other cats",
    main="Random Cats")
```



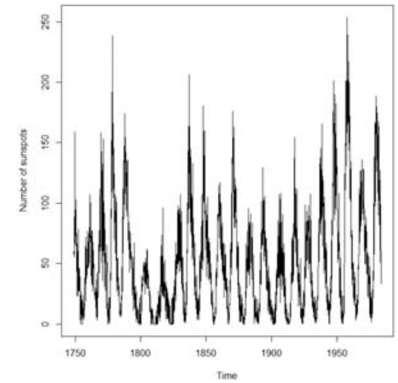
How this works:

- 11 cat shape images saved as PNG
- Calls plot(x, y, …) – set up plot frame
- rasterImage(catImg, …) – plot each cat

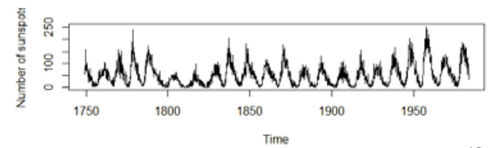
Time series plots

R has special methods for dealing with time series data
The sunspots data set records monthly mean relative sunspot numbers from 1749 to 1983

```
> data(sunspots)
> str(sunspots)
 Time-Series [1:2820] from 1749 to 1984: 58 62.6 70 55.7 85 83.5 94.8 75.5 ...
plot(sunspots, cex.lab=1.5, ylab="Number of sunspots")
```



But the aspect ratio (V/H) of the plot is often important.
A systematic pattern is revealed when the average local trend is ~ 45º

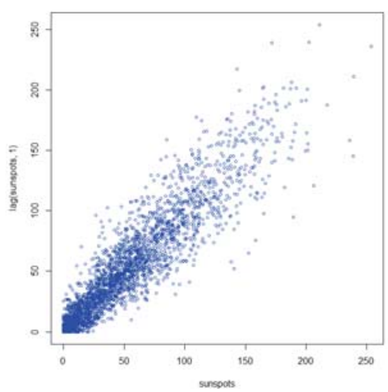


Time series: lag plots

Lag plots show a time series against lagged versions of themselves. This helps visualizing 'auto-dependence'.

```
plot(sunspots, lag(sunspots, 1),
   cex=0.7, col="blue")
```

There is a strong dependence between this year's sunspots and last.

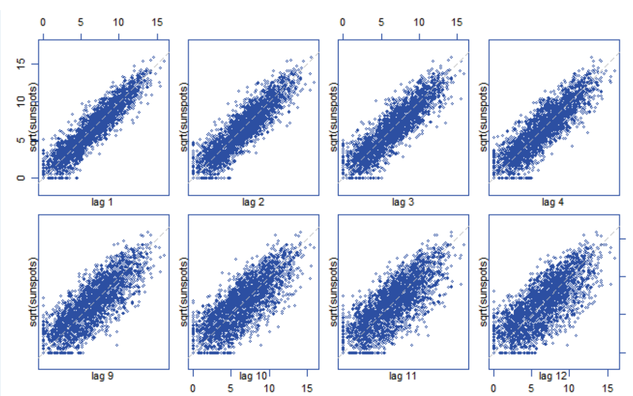


Time series: lag plots

Often, we want to see dependence across a range of lag values. lag.plot(series) does this quite flexibly

```
lag.plot(sqrt(sunspots), set = c(1:4, 9:12), layout=c(2,4), col="blue", cex=0.7 )
```

dependence is persistent, but weakens over lags

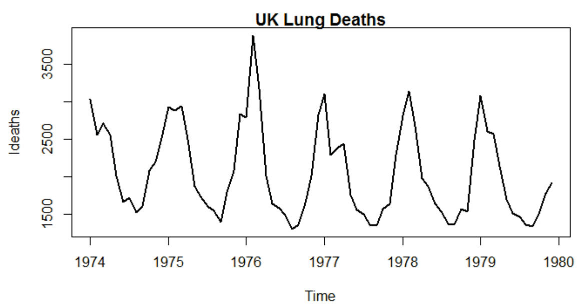


Time series: Seasonal patterns

Data UKLungDeaths: monthly deaths from bronchitis, emphysema and asthma in the UK, 1974–1979

```
data(UKLungDeaths)
plot(ldeaths, lwd=2, main="UK Lung Deaths")
```

Looks like cycles, peaking in winter



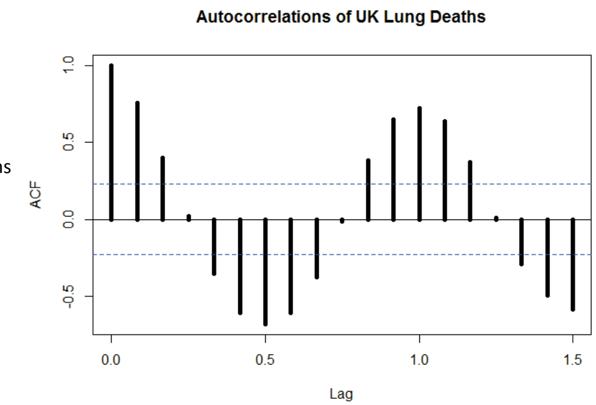
Time series: Seasonal patterns

The acf() function calculates and plots autocorrelations of a time series across various lags

```
acf(ldeaths, lwd=5, main="Autocorrelations of UK Lung Deaths")
```

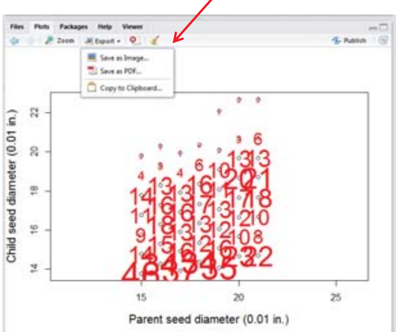
This gives a compact view of the seasonal pattern

Other time-series graphs can show other details

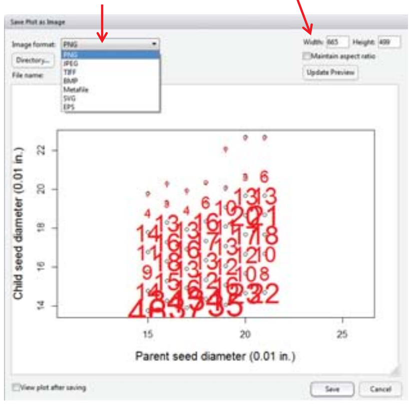


Saving image files: R Studio

From the R Studio Plots tab, you can save any image in a variety of types



Some options are available in the menu to control the details of size, shape & image format

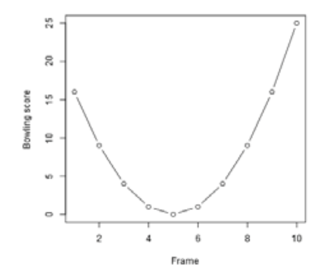


For publication purposes, you will often want more control: plot margins, font sizes, figure shape, etc.

Saving image files: R scripts

- The default graphics device in R is your computer screen.
- In an R script, there are 3 steps:
  1. Open a graphics device, with desired parameters
     - Call png(), jpg(), pdf(), …
  2. Create the plot
  3. Close the graphics device: dev.off()

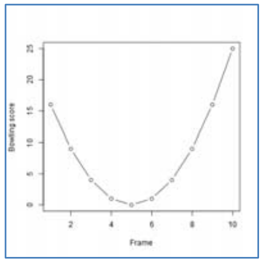
```
1  png(file="bowling.png", width=400, height=400)
     x <- 1:10
     y <- (x - 5)^2
2    plot(y ~ x, type="b",
        xlab="Frame",
        ylab="Bowling score")
3  dev.off()
```



Much easier with ggplot2: ggsave()

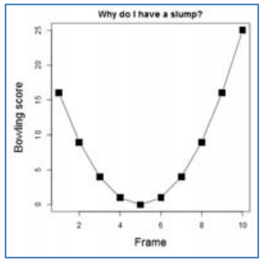
Saving image files: Margins, fonts

```
png(file="bowling0.png", width=400, height=400)
   x <- 1:10
   y <- (x - 5)^2
   plot(y ~ x, type="b",
       xlab="Frame",
       ylab="Bowling score")
dev.off()
```



Set plot margins, font size for points & labels

```
png(file="bowling.png", width=400, height=400)
op <- par(mar=c(5, 5, 2, 1))
   x <- 1:10
   y <- (x - 5)^2
   plot(y ~ x, type="b",
       pch=15, cex=2, cex.lab=1.5,
       xlab="Frame",
       ylab="Bowling score",
       main="Why do I have a slump?")
par(op)
dev.off()
```



Saving image files: R markdown

- In R markdown files, use chunk options to control figures & other output
  - global options -- control all chunks: knitr::opts_chunk$set()
  - individual chunk options

Set global options:

```
```{r setup, include=FALSE, message=FALSE}
opts_chunk$set(fig.path="figs/",
   dev=c("png","pdf"),          # devices for figs
   fig.width=6, fig.height=5,   # fig size
   fig.align="center",
   dpi=300,                     # make high res.
   digits=4, ... )              # printed output
```
```

Change size for this figure:

```
```{r wheat1, fig.width=9, fig.height=4}
data(Wheat)
plot(Wheat ~ Year, data=Wheat, type="s")
 ...
```
```

Details: see https://yihui.name/knitr/options/

Summary

- Standard R graphics
  - High-level (plot()) vs. low level (lines()) functions
  - Understand object-oriented methods
- Graphics parameters
  - Understand the basic ones: col, pch, lty, lwd
  - Use help(par) or cheat sheet to find others
  - For a high-level function, use help(fun)
- Building graphs
  - Think about graphic elements: points, lines, areas, …
  - How these should be rendered: graphical attributes