Visual Display of Quantitative Data

**Effective Graphical Display of Statistical Data**

Reveals data, does not conceal or distort it.

Clearly communicates complex, multivariate ideas.

Encourages exploring the data at multiple levels.

Efficiently presents many numbers in a small space.

Has purpose, not “chartjunk”, and should focus the viewer on the substance of the data, not distract them.

**Source:** *The Visual Display of Quantitative Information* by Edward R. Tufte
Minard’s Chart of Napoleon’s Russian Campaign

**Source:** Charles Joseph Minard, 1861
Typography

Typography is the art of communicating with letter forms. Decisions have to be made about the typeface to be used, the size (e.g. 12 point size), weight (e.g. light, semi-bold, bold, extra-bold) and style (e.g. italic). In modern use, a font is a typeface. Fonts are typically classified by form (Serif or San-Serif), era (Old Style, Transitional, Modern, etc.) and spacing (Fixed Width or Variable Width).

**Serif (Roman) fonts have decorative lines at the end of a stroke:**
Caslon, Garamond, Goudy, Sabon (Old Style)
Baskerville (Transitional)
Bodoni (Modern)

**San Serif (Gothic or Grotesque) fonts lack serifs:**
Fruitiger, Gill Sans, Myriad Pro (Humanist)
Helvetica (Transitional)
Futura (Geometric)
Fixed Width vs. Variable Width

Each character in a fixed width (monospaced) font takes up the same amount of horizontal space, like a typewriter, allowing multiple sequence alignments to properly align. Variable width fonts can throw off multiple sequence alignments.

**Commonly Used Fixed Width Fonts**

**Mac OS X**  Courier, Courier New, Monaco and Letter Gothic

**Windows**  Courier New and Lucida Sans Typewriter

**Fixed Width Font Alignment**  (Courier)

```
. . . m s h N q f q f i G n L t r D
M A s R G v N K V I L V G n L G q D
M A v R G I N K V I L V G R L G k D
```

**Variable Width Font Alignment**  (Times)

```
. . . m s h N q f q f i G n L t r D
M A s R G v N K V I L V G n L G q D
M A v R G I N K V I L V G R L G k D
```
Elements of Design

Alignment
Nothing should be placed arbitrarily. Every element should have some visual connection with another element. Aligning to guide lines or using an underlying design grid can assist with this.

Proximity
Related items should be grouped in close proximity.

Contrast
To avoid important items from blending into the rest of the elements, make them very different. Bold weights or italic style can help, or a using a san serif font for headers and a serif font for body text (or vice-versa). Avoid displays where everything blends together or lacks contrast.

Consistency
Repeat visual elements of the design throughout, and use them consistently. Use fonts and color carefully and consistently, and avoid overuse or arbitrary use of either.

Source: The Non-Designer’s Design Book by Robin Williams

Lecture 12: Data Visualization, R and Bioconductor
December 8, 2014
Elements of Design

Alignment
Nothing should be placed arbitrarily. Every element should have some visual connection with another element. Aligning to guide lines or using an underlying design grid can assist with this.

Proximity
Related items should be grouped in close proximity.

Contrast
To avoid important items from blending into the rest of the elements, make them very different. Bold weights or italic style can help, or a using a san serif font for headers and a serif font for body text (or vice-versa). Avoid displays where everything blends together or lacks contrast.

Consistency
Repeat visual elements of the design throughout, and use them consistently. Use fonts and color carefully and consistently, and avoid overuse or arbitrary use of either.

Source: The Non-Designer’s Design Book by Robin Williams
Comprehension

**ALL CAPS vs. Mixed or lower case**
Text set in ALL CAPS has a significantly lower level of comprehension than text set in mixed or lower case, even in relatively short text, such as headlines.

**Serif vs. San Serif**
Serif fonts may have an advantage in comprehension when used in blocks of text in print, while san serif fonts may have an advantage in comprehension on displays, but the research in this area is not conclusive.

**Text and Background Color**
Black text or dark colored text blocks on white backgrounds or lightly tinted (10% to 20%) color backgrounds have the highest level of comprehension. Avoid using lighter color text blocks, or backgrounds with more color, as these can have poor comprehension.
A Study of Text and Background Color

**Black Text on White Background**
Black text on a white background had the highest level of comprehension (70% good).

**Black or Dark Text on Lightly Tinted Backgrounds**
Black or dark colored text on 10% to 20% tinted backgrounds had acceptable comprehension (32% to 68% good).

**Lighter Color Text**
Lighter colored text on white or 10% to 20% tinted backgrounds had poor comprehension (0% to 29% good).

**Colored or Black Backgrounds**
Any text on a 40% or more tinted background had poor comprehension (0% to 22% good). White text on a black background had the worst comprehension (0% good).

**Preference Does Not Mean Comprehension**
Despite the potential for lower comprehension, 81% of the subjects stated they found a colored background more attractive and interesting than black on white text.

*If you must set type on a dark background, lighter translucent backgrounds, heavier font weights (bold or extra-bold) or simpler, more geometric fonts may help.*

**Source:** *Type & Layout* by Colin Wheildon
A Study of Text and Background Color

Black Text on White Background
Black text on a white background had the highest level of comprehension (70% good).

Black or Dark Text on Lightly Tinted Backgrounds
Black or dark colored text on 10% to 20% tinted backgrounds had acceptable comprehension (32% to 68% good).

Lighter Color Text
Lighter colored text on white or 10% to 20% tinted backgrounds had poor comprehension (0% to 29% good).

Colored or Black Backgrounds
Any text on a 40% or more tinted background had poor comprehension (0% to 22% good). White text on a black background had the worst comprehension (0% good).

Preference Does Not Mean Comprehension
Despite the potential for lower comprehension, 81% of the subjects stated they found a colored background more attractive and interesting than black on white text.

If you must set type on a dark background, lighter translucent backgrounds, heavier font weights (bold or extra-bold) or simpler, more geometric fonts may help.

Source: Type & Layout by Colin Wheildon
The R programming language was released in 1993 by Ross Ihaka and Robert Gentleman, statisticians at the University of Auckland in New Zealand. Their original goal was to develop a statistics language suitable for teaching in their Mac computer lab. The language’s name is a reference the S programming language for statistics, which was one of their inspirations, and also refers to the first names of the authors.

The reference implementation of R is primarily written in C, Fortran and R and and is free and open source, released under the GNU General Public License, and supported by a community of open source developers at the Comprehensive R Archive Network, which serves as a repository for R and free third party R software, and currently contains over 6,100 packages.

R is an interpreted language, and primarily supports procedural programming with functions, but has some object oriented functionality. It supports matrix arithmetic, a wide variety of data structures useful in math and statistics, math symbols, and a variety of graphing functions and techniques.

In recent years, R has become one of the most popular programming languages used by statisticians and data miners, and is becoming well established in bioinformatics.

Source: http://www.r-project.org and http://cran.r-project.org
RStudio

RStudio is a free, open source, Integrated Development Environment (IDE) for the R programming language that provides R with a simple graphical user interface and useful development tools. It runs on Mac, Windows and Linux.

The source editor features R specific highlighting, code completion, and smart indentation, and allows you directly run R code from it. Help and documentation are built in, along with the ability to quickly jump to function definitions. Additional support for development is provided by an interactive debugger, support for version control systems (Git and Subversion), and package authoring and documentation tools.

RStudio can simultaneously display multiple panes, typically a source code editor pane, an interactive console pane (like Terminal), a workspace pane, and a plotting pane. Interactive graphics can be created using the `manipulate` package.

Source: http://www.rstudio.com
Working with RStudio

You will typically be working in a project with its own working directory. When not in a project, you can set the working directory under Tools > Global Options... You can customize the appearance, pane order and tabs of RStudio here as well.

The source code editor pane (upper left by default) is useful for larger projects, otherwise the interactive console pane (by default below it) can be used directly. In both the source editor and interactive console, Tab acts as an auto-complete function, suggesting file or function names and Alt and - is a shortcut for the frequently used <- assignment characters.

In the source editor, Command and enter (Mac) or Control and enter (Win) sends the current line of code to the console and runs it. Multiple lines of text can be selected beforehand. The Run button also serves the same function.

In the interactive console, Command and up arrow (Mac) or Control and up arrow (Win) brings up a historical list of previous commands.

The workspace pane (upper right by default) has an Environment tab which displays anything created during an R session, including values, objects or functions and a History tab that stores all previous commands run.

The view pane (by default below the workspace pane) has a Files tab for displaying the directory structure, a Plots tab for displaying graphs (the arrows move between multiple graphs), a Packages tab that shows loaded packages, a Help tab for displaying documentation and a Viewer tab that displays local HTML and web content.
R Functions and Statements

Functions in R are written as the name of the function, directly followed by parentheses, e.g. `edit()`. No special characters are required following a function, simply a carriage return as the end of the line.

To use a function, provide the appropriate arguments in the parentheses, e.g. `edit(data)`

Statements in R are normally written one to a line, with curly brackets used to group statements.

You can define your own unique functions in R using the syntax:

```
uniquefunctionname <- function(arg1, arg2, ... ){
   statements
   return(object)
}
```

**Help**

Typing a question mark directly in front of a function name displays its documentation, e.g. `?edit`

Typing the name of the function without parentheses returns its code in the console, e.g. `edit`

`example(functionname)` displays examples of how the function is used, e.g. `example(edit)`

`args(functionname)` displays a list of the function's arguments, e.g. `args(edit)`

`??("searchterm")` searches all of R's help documentation for the search term.
R Syntax

R supports a number of data types, including scalar variables, vector variables (numeric, character, logical), lists, matrices, and data frames. Variable names may not start with a digit, and can contain underscores, but not many other special characters or spaces, and R is case sensitive. Anything following a # character is considered a comment.

Assigning Values
In R, <- is usually used to assign a value to a variable, not the = symbol e.g. x_scalar <- 1. Most other mathematical operators function as expected, and R supports many higher mathematical and statistical functions.

The combine function, c() is used to assign multiple same type values to a variable, e.g. x_vec <- c(1, 2, 3)

The list function, list() is used to assign multiple different types of values to a variable, e.g. x_list <- list(1, "two", 3.0). Note that the first value in an R list is at position 1, not 0, and R assumes that 1 and 3.0 are both numeric data types, not an integer and a float.

Matrices and Data Frames
A matrix consists of rows and columns of the same data type, e.g. x_matrix <- matrix(c(1, 2, 3, 4), nrow=2, ncol=2). By default, a matrix fills by column. The dim() function returns the dimensions of a matrix.

A data frame has rows and columns, but can store different data types in each column, and each column must have a name, preferably unique, e.g. x_dataframe <- data.frame(a = 1:5, b = 6:10). Note that the colon operator is used to generate a range of numbers and the equal sign is properly used here as an equality operator.

Operations and Loops
Operations, including loops, are generally applied in R with the apply() function, although if and if...else statements are also supported. The third-party plyr package lets you easily apply operations to part of a data set.
R Standard Library

Viewing Data
attach() attaches a list or data frame to the R search path for easy searching.
edit() displays an editable version of an object.
print() displays the values of an object.

Graphing Data
barplot() creates a bar plot.
boxplot() creates a box plot.
heatmap() creates a heat map.
hist() creates a histogram.
plot() creates a scatter plot.

Statistics
mean() calculates the arithmetic mean.
sd() calculates the standard deviation.
summary() provides summary statistics if used with a data frame.
table() creates a cross-tabulation and table creation.
t.test() one and two sample t-tests on vectors of data (Student’s t-Test).
var, cov and cor compute the variance of x and the covariance or correlation of x and y.
R Input and Output

Input

\[
test\textunderscore data \leftarrow \text{read.csv("filename.txt", header=FALSE)}\]

will read the data in the named CSV file into a data frame. Note that a header is normally expected, and in some cases you may need to add \texttt{stringsAsFactor=FALSE} to prevent strings from being interpreted as statistical values by R. You can also provide a URL instead of a file name (assuming the data at the URL is in standard CSV format). \texttt{read.csv()} is identical to \texttt{read2.csv()} and \texttt{read.table()} except with a different set of defaults.

In RStudio, you can simply use \texttt{Environment tab > Import Dataset…}

Many third party packages exist for reading data in particular file formats, databases or online data sources.

Output

\[
\text{save.image()}\]

will save the entire workspace to a file named \texttt{.RData}

To save data in a compact binary file, use \texttt{save(data, file="filename.rda")} and to load the object, use \texttt{load("filename.rda")}.

To save data in text format, use \texttt{dump(data, "filename.Rdmpd")} and to load it, use \texttt{source("filename.Rdmpd")}.

\texttt{write.csv(test\textunderscore data, "test.csv")} will save data to a CSV text file. To omit headers, add the argument \texttt{row.names=F}. Use \texttt{write(test\textunderscore data, "test.txt")} to save data to a text file.
Statistics Examples with R

```r
x <- runif(10)  #Also try 10000
summary(x)

y <- sample(1:100, 10, replace=T)
summary(y)
print(y)

y <- sample(1:100, 100, replace=T)
summary(y)
mean(y)
sd(y)
unique(y)
length(unique(y))

y <- sample(1:100, 10000, replace=T)
summary(y)
sd(y)

z <- rnorm(10)  #Also try 10000
summary(z)
sd(z)  #Expected value is 1.0
```
Probability with R

Coin Tosses

```r
rbinom(10,1,0.5)
rbinom(10,100,0.5)
rbinom(10,10000,0.5)
rbinom(10,1000000,0.5)
```

Random DNA

```r
x <- sample(c("A","C","G","T"),10,replace=T)
x

x <- sample(c("A","C","G","T"),100,replace=T)
x
write(x, file="100test.txt")  #To avoid one character per line
y = paste(x, collapse="")    #use an empty collapse argument
y
write(y, file="100randomdna.txt")
```
a <- sample(1:100, 100, replace=T)
plot(a)
b <- rnorm(100, mean=50, sd=9)
plot(b)  # looks similar, but check the y axis!
c <- rnorm(100, mean=50, sd=3)
plot(c)  # looks similar, but check the y axis!

How to display this on a single plot? The lines() function will add lines to an existing plot.

a <- sample(1:100, 10000, replace=T)  # Increased n for visibility
b <- rnorm(10000, mean=50, sd=9)
c <- rnorm(10000, mean=50, sd=3)

plot(a, type="l", col="red")
lines(b, col="green")
lines(c, col="blue")

Histogram of normal distribution?

x <- rnorm(100)
hist(x, col="lightblue", freq=T)
summary(x)

x <- rnorm(100)  # can vary n
hist(x, col="lightblue", freq=F)  # density instead of frequency
curve(dnorm(x, mean=0, sd=1), add=T, col="darkblue")  # adds a density curve
Heat Maps with R

The following R source code generates a random matrix of 10 columns and 20 rows containing 200 random integers between 1 and 100, then views the randomly generated data. It then creates a heat map using the default cyan to purple heatmap colors. Note that there is no line break in the third line.

\[
\text{hm} \leftarrow \text{matrix(sample(1:100, 200, replace=T), ncol=10)}
\]

\[
\text{hm}
\]

\[
\text{hm\_heatmap} \leftarrow \text{heatmap(hm, Rowv=NA, Colv=NA, col = cm.colors(256), scale="column", margins=c(10,20))}
\]

Use \text{col = heat.colors(256)} for more temperature-like colors. Third-party packages such as \text{ggplot2} or \text{RColorBrewer} offer far more options.
Graphing Error Bars with R

```r
error.bar <- function(x, y, upper, lower=upper, length=0.1,...)
  if(length(x) != length(y) | length(y) != length(lower) | length(lower) != length(upper))
    stop("vectors must be same length")
  arrows(x,y+upper, x, y-lower, angle=90, code=3, length=length, ...)

y <- rnorm(50000, mean=1)
y <- matrix(y,10000,5)
y.means <- apply(y,2,mean)
y.sd <- apply(y,2,sd)
y1 <- rnorm(50000, mean=1.1)
y1 <- matrix(y1,10000,5)
y1.means <- apply(y1,2,mean)
y1.sd <- apply(y1,2,sd)

yy <- matrix(c(y.means,y1.means),2,5,byrow=TRUE)
ee <- matrix(c(y.sd,y1.sd),2,5,byrow=TRUE)*1.96/sqrt(10000)

barx <- barplot(yy, beside=TRUE,col=c("blue","magenta"), ylim=c(0,1.5),
  names.arg=1:5, axis.lty=1, xlab="Replicates", ylab="Value (arbitrary units)"
)
error.bar(barx,yy,ee)
```

**Source:** James Holland Jones, http://monkeysuncle.stanford.edu/?p=485
Installing R Packages

Additional third-party packages for R can be browsed for at the Comprehensive R Archive Network (CRAN) at http://cran.r-project.org or searched and browsed for at http://www.rdocumentation.org (which also includes package documentation and download statistics).

Installation of third-party packages is handled by the built-in `install.packages()` function, e.g. `install.packages("ggplot2")`. Run `update.packages()` beforehand to make sure your other packages are up to do. In RStudio, you can also use Tools > Install Packages...

R packages are installed into libraries, which are directories containing a subdirectory for each package installed there. To load a package, use the `library()` function, e.g. `library("ggplot2")`

Some notable third-party R packages include:

- **ggplot2**, a plotting system featuring a variety of plots, statistical transformations and display options.
- **plotly**, an interface to interactive online plot.ly graphs.
- **plyr**, tools for splitting, applying and combining data.
- **shiny**, an interactive web application framework for Shiny servers or cloud hosting.
- **stringr**, makes R string functions more consistent, simpler and easier to use.
Bioconductor

Bioconductor is a set of free, open source, primarily R based tools for bioinformatics and computational biology focused on the analysis and comprehension of high-throughput genomic data. It was first released in 2001, the current version, 3.0 was released in October 2014, and consists of 934 software packages, 219 experiment data packages, and 870 annotation packages.

Bioconductor provides statistical and graphical methods for the analysis of genomics and proteomics data, including numerous packages for parsing and analyzing biological data from various sources: ChIP-seq, DNA sequencers (including Illumina, IonTorrent and Roche 454), flow cytometry, microarray (including Affymetrix and Illumina), qPCR, RT-qPCR, RNA-Seq and SNP.

Bioconductor includes annotation packages for assembling and processing genomic annotation data from a variety of databases (including GenBank, GO, Entrez, UniGene and the UCSC Human Genome Project) or associating microarray and other genomic data in real time with biological metadata from web databases (including GenBank, Entrez genes and PubMed).

The basic Bioconductor installation only installs a set of core packages and functions (Biobase, BiocGenerics, BiocInstaller and GenomeInfoDb). To install additional Bioconductor packages, use the bioCLite() function.

Source: http://www.bioconductor.org
Installing Bioconductor Packages

Browse or search for additional Bioconductor packages at:
http://www.bioconductor.org/packages/release/ or http://www.rdocumentation.org

To install the core Bioconductor packages in a new R installation, use:
source("http://bioconductor.org/biocLite.R")
biocLite()

To install additional Bioconductor packages, use biocLite("packagename"), e.g:
source("http://bioconductor.org/biocLite.R")
biocLite("Biostrings")

Notable Bioconductor packages include:

annotate, functions for extracting data from meta-data libraries (including NLM and NCBI), geneplotter support, HTML output.

Biostrings, string objects and algorithms for working with biological sequence data.

CummeRbund, exploration, analysis and visualization of Cufflinks high-throughput RNA-Seq data.

DESeq2, differential gene expression analysis based on the negative binomial distribution.

edgeR, differential expression analysis of RNA-seq and digital gene expression profiles with biological replication.

geneplotter, graphic related functions for plotting genomic data.

phyloseq, handling and analysis of high-throughput microbiome census data.
References

An Introduction to R free at:

A Little Book of R for Bioinformatics free at:

Bioconductor Short Courses free at:
http://www.bioconductor.org/help/course-materials/

swirl, a free interactive console tutorial package for R:
install.packages("swirl")
http://swirlstats.com

The Visual Display of Quantitative Information by Edward R. Tufte

The Non-Designer’s Design Book, 4th edition by Robin Williams

Type & Layout by Colin Wheildon

Introductory Statistics with R by Peter Dalgaard