R Overview

Paul E. Johnson[1]

[1]University of Kansas

December 12, 2020

Outline

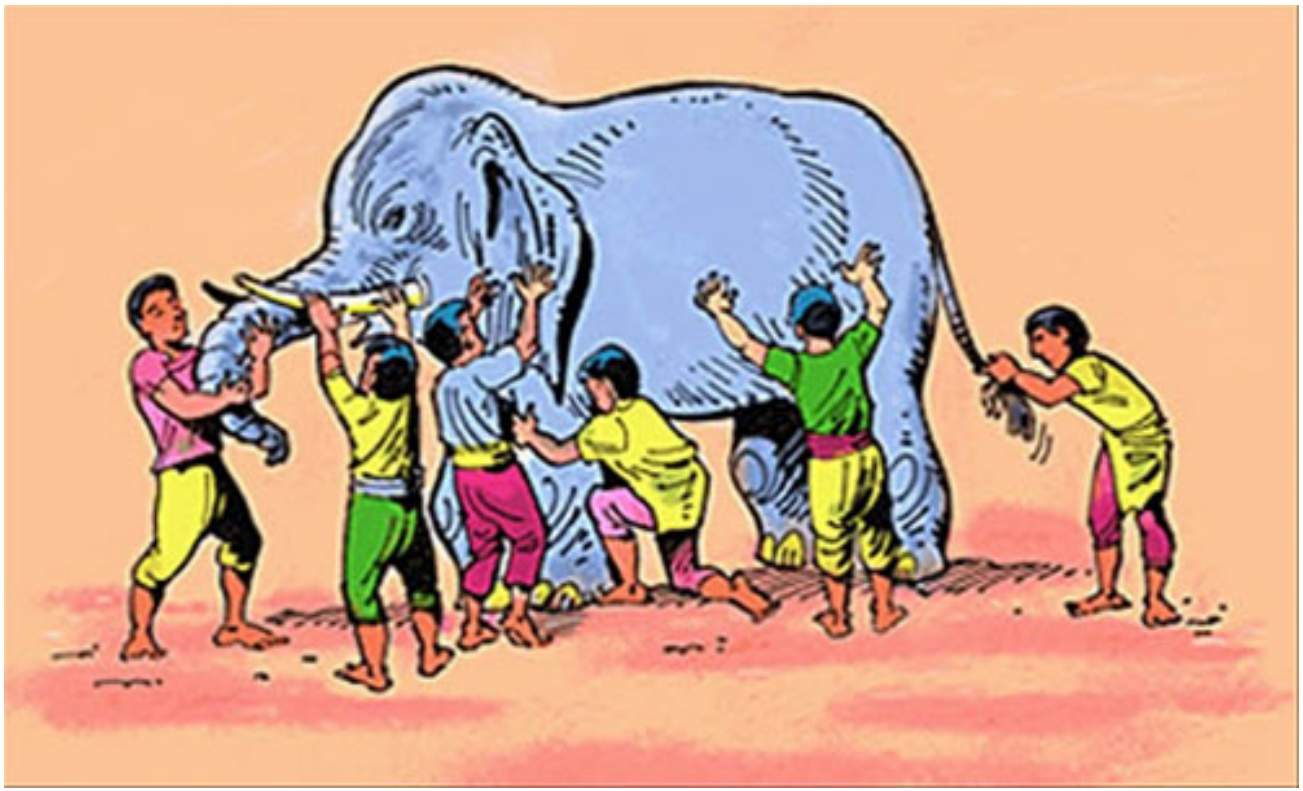
1. Calculator
2. Statistical Package Framework
3. Graphics
4. Stat Toolbench
5. Programming Language
6. If You Want To Get Started
7. Appendix 1: Code for Simulation Examples

KU

What is this Presentation?

- A survey of R(R Core Team, 2018)
- Some "review" of elementary concepts
- Some "preview" of advanced possibilities
- Not a substitute for careful reading of *An Introduction to R* or the *R-FAQ*
- In case you found this and you are not at the KU Summer Stats Camp, consider signing up and coming on over! We have a 1 week-long session on R taught by some well qualified folks :) `http://crmda.ku.edu`

R is a little bit like an elephant



Ouch. That's not my Trunk!

R is a

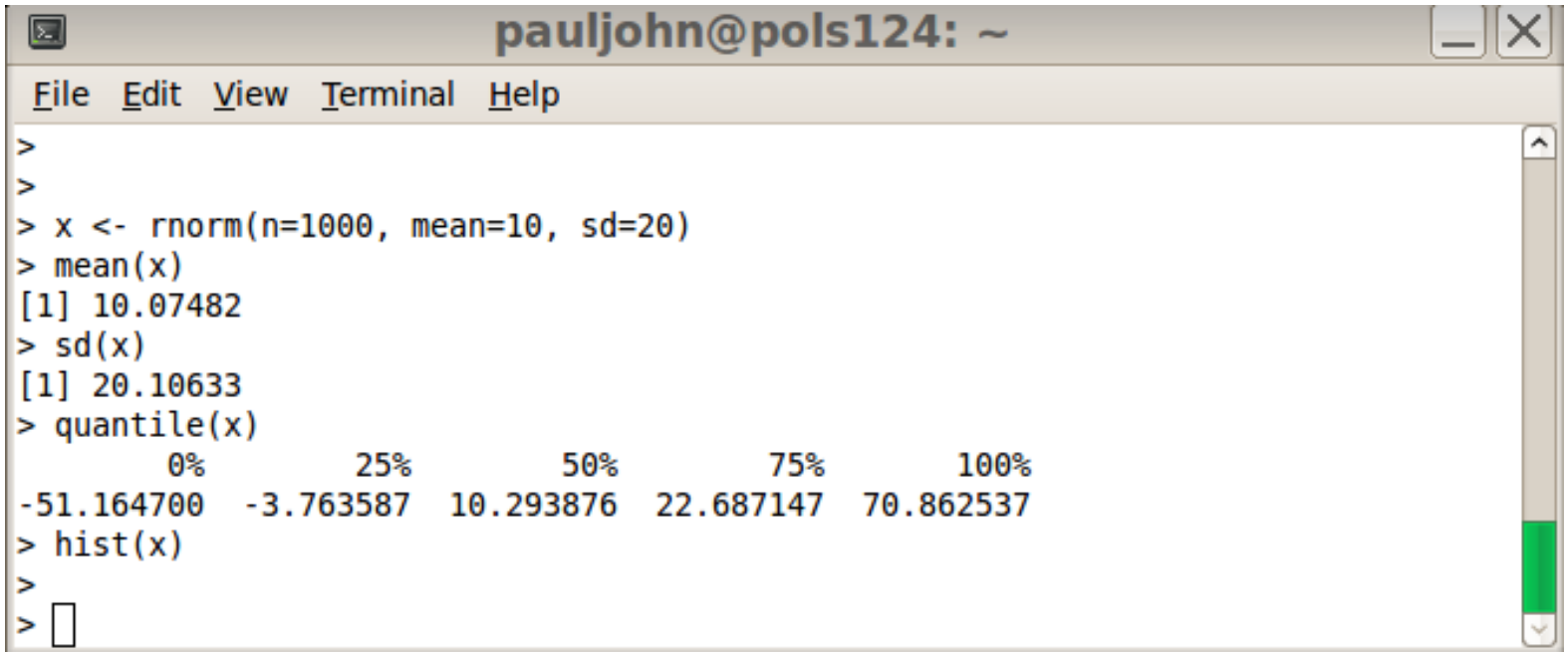
- **calculator**
- **statistical package framework**
- **graphical plotter**
- **statistical toolbench**
- **computing language**

Outline

1. Calculator
2. Statistical Package Framework
3. Graphics
4. Stat Toolbench
5. Programming Language
6. If You Want To Get Started
7. Appendix 1: Code for Simulation Examples

A Free Form Calculator

- Start an R session "interactively" (R or Rterm, for example)
  $>$ is the "prompt". Type stuff there!



A Calculator for Your Math Homework

```
2+3 #addition
```

```
[1] 5
```

```
43 * 67 #multiplication
```

```
[1] 2881
```

```
33/699 #division
```

```
[1] 0.0472103
```

```
5%%3   #modulo (remainer)
```

```
[1] 2
```

```
3^4 #power
```

A Calculator for Your Math Homework ...

```
[1] 81
```

```
log(17.44) #natural log
```

```
[1] 2.858766
```

```
exp(2.33)  #exponentiation
```

```
[1] 10.27794
```

```
sin(2*pi)  #sine
```

```
[1] -2.449294e-16
```

Its a Calculator that Remembers!

- Create a new variable with the symbol "<-" (read: is assigned as).
  > x <- 5
- x is now a collection with just one number, 5
- R has many functions that we "call" with x as an "argument".
  - The square root of x is found by
    > sqrt(x)

Its "Vectorized"

```
myvector <- c(1,2,3,4,5,6,7)
sqrt(myvector)
```

```
[1] 1.000000 1.414214 1.732051 2.000000 2.236068 2.449490 2.645751
```

Its "Matricized"

```
x <- c(1,2,3,4,5,6,7,8,9)
xmat <- matrix(x, ncol=3)
xmat
```

```
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

```
xmat[1,3]
```

```
[1] 7
```

```
xmat[ ,2]
```

```
[1] 4 5 6
```

```
xmat[2, ]
```

Its "Matricized" ...

```
[1] 2 5 8
```

```
apply(xmat, 2, sum)
```

```
[1]  6 15 24
```

Outline

1 Calculator

2 Statistical Package Framework

3 Graphics

4 Stat Toolbench

5 Programming Language

6 If You Want To Get Started

7 Appendix 1: Code for Simulation Examples

Its Like a "Statistical Package" (sorta)

- Programs like SPSS or SAS are called "stat packs."
- These are "pre-packaged" routines
  - No chance to review internal code
  - Difficult to modify/extend calculations
- User has a "plug and play" list. "If data is like X, then run procedure X"
- Many people use R that way, although they are missing part of the point.

More Correct: R is a Package Framework

- R has plenty of pre-packaged routines
- Inspect Your Computer: What packages are currently installed?

```
> library()
```

- Want version numbers, install locations, etc?

```
> installed.packages()
```

- Want package names only?

```
> row.names(installed.packages())
```

Large Collection of Regression Routines

- Linear Regression is in the base stats package

```
mymodel <- lm (depvar ~ indepvar1 + indepvar2,
   data = mydfname)
```

- "nls" nonlinear least squares
- "glm" Generalized Linear model
- Countless packages for other regression models
  - nlme - nonlinear mixed effects
  - lme4 - linear mixed effect (next generation of nlme)
  - MASS - negative binomial regression, robust and smooth regressions
  - mgcv - generalized additive models
  - "mars" - Multivariate Adaptive Regression Splines
  - "betareg" - regression with a "Beta distributed" outcome variable

A Little Introspection, Please

- After a fresh install, one has only the packages written by the R core team and a very selective set of packages that they recommend.
- Thousands of other packages available
- Tip: Where does R search for packages in your system

```
> .libPaths()
```

- Note some paths can only be written into by an "administrator", but some may be written in by an "ordinary user".
- If somebody emails you a package ("whatever-2.1.tar.gz") it can be manually installed. In a Linux shell:

```
$ R CMD INSTALL whatever-2.1.tar.gz
```

KU

CRAN: a service from the R Core Team

- CRAN is the largest indexed set of packages (but others exist)
- R Package Writers follow a set of guidelines, but nobody "certifies" them "officially"
- Available after passing build checks & sanity tests
- Package server allows "automagical" installation
- For convenience, R users can download & install from within R.

```
> install.packages(c("lmtest","car"), dep =
   TRUE)
```

- Install path depends on user's admin authority (In Windows, run R "as administrator" to do package installs).

Prodigious Profusion of Packages

- Wonder what you are missing out on?

```
> rownames(available.packages())
```

  On 2011-01-31, that command returned a list of 2769 packages.
  On 2013-05-10, that returned 4467 packages!

- I want it ALL!
  I wrote a script that installed them all on a Windows system. Download and Install took
  - 3 hours
  - 2.7 Gigabytes of storage
  - Scripts: http://pj.freefaculty.org/R/SystemAdmin
- Periodic Maintenance: Check for updates periodically

```
> update.packages(ask=F, checkBuilt = TRUE)
```

Every Time I Load a Package, I . . .

- Load a package that is already installed, e.g. "lme4"
  $>$ library(lme4)
- Review the list of functions in that package

```
> library(help = lme4)
```

- Read the vignettes listed.
- Read the help on the important functions

```
> ?lmer
```

- Run the examples on the important functions

```
> example(lmer)
```

A Vignette on Sudoku

- I recently learned there is an R package for making and playing SudoKu puzzles.
- I installed it

```
> install.packages("sudoku")
```

- I loaded it

```
> library(sudoku)
```

What is that Sudoku thing?

Always do this:

```
> library(help = sudoku)
```

Documentation Included! No Extra Charge!

```
Information on package 'sudoku'
Description:
Package: sudoku
Version: 2.2
Date: 2009-02-02
Title: Sudoku Puzzle Generator and Solver
Author: David Brahm <brahm@alum.mit.edu> and Greg Snow
    <Greg.Snow@intermountainmail.org>, with contributions from
    Curt Seeliger <Seeliger.Curt@epamail.epa.gov> and Henrik
    Bengtsson <hb@maths.lth.se>.
Maintainer: David Brahm <brahm@alum.mit.edu>
Suggests: tkrplot
Description: Generates, plays, and solves Sudoku puzzles. The
    GUI playSudoku() needs package "tkrplot" if you are not on
    Windows.
License: GPL
Packaged: Mon Feb 2 16:28:15 2009; a215020
Built: R 2.10.1; ; 2010-03-19 06:50:35 UTC; unix
```

```
Index:

fetchSudokuUK          Fetch the daily sudoku puzzle from
                       http://www.sudoku.org.uk/
generateSudoku         Randomly Generate a Sudoku Puzzle Grid
hintSudoku             Give a Hint for a Sudoku Cell
playSudoku             Interactively play a game of Sudoku
printSudoku            Print a Sudoku Grid to the Terminal.
readSudoku             Read a File Containing a Sudoku Grid
solveSudoku            Solve a Sudoku Puzzle
writeSudoku            Write a Sudoku Grid to a File
```

Documentation Included! No Extra Charge!

- Then I use the help feature to find out more on the interesting-looking ones:

```
> ?generateSudoku
```

- That's the same as:

```
> help(generateSudoku)
```

- Perhaps I run the example that is displayed on the help page:

```
> example(generateSudoku)
```

When you run a function, the parentheses are required, even if you don't add any specific arguments. This tells generateSudoku to use the default settings.

```
generateSudoku()
```

```
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
 [1,]    2    0    0    0    0    0    4    0    0
 [2,]    0    9    0    0    0    2    7    8    0
 [3,]    0    0    5    0    4    3    0    0    0
 [4,]    9    0    4    2    6    0    5    0    0
 [5,]    0    0    0    8    0    0    9    0    0
 [6,]    7    0    8    0    0    0    1    2    0
 [7,]    3    0    0    0    0    0    0    5    7
 [8,]    5    0    7    0    0    9    0    1    0
 [9,]    6    0    0    7    0    5    3    0    0
```

```
> generateSudoku()
```

```
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
 [1,]    1    0    0    0    0    0    0    0    0
 [2,]    7    0    0    0    1    3    5    8    2
 [3,]    8    2    0    0    6    0    0    0    0
 [4,]    4    0    1    0    2    8    6    0    0
 [5,]    0    5    8    0    0    0    4    0    1
 [6,]    0    0    0    3    4    0    0    0    0
 [7,]    5    0    2    0    7    9    3    1    4
 [8,]    0    0    0    0    0    2    0    0    0
 [9,]    0    7    0    0    0    0    0    5    0
```

A Nicer Looking Sudoku Puzzle

```
myPuzzle <- generateSudoku(Nblank = 20, print.it
    = FALSE)
printSudoku(myPuzzle)
```

```
 +-------+-------+-------+
 | 2   5 | 7   4 | 8 1   |
 | 4 7   | 1 9 8 | 5 2   |
 | 9 8 1 |   2 3 | 7 6 4 |
 +-------+-------+-------+
 | 5 2 3 | 4 7 6 | 9 8 1 |
 |     8 |     2 | 4 7   |
 | 6   7 | 8 1 9 | 3 5   |
 +-------+-------+-------+
 | 7 6 4 | 9 8 1 |   3 5 |
 |     9 |     5 | 6 4 7 |
 |   5 2 | 6 4 7 |   9 8 |
 +-------+-------+-------+
```

Torture Yourself with British Sudoku

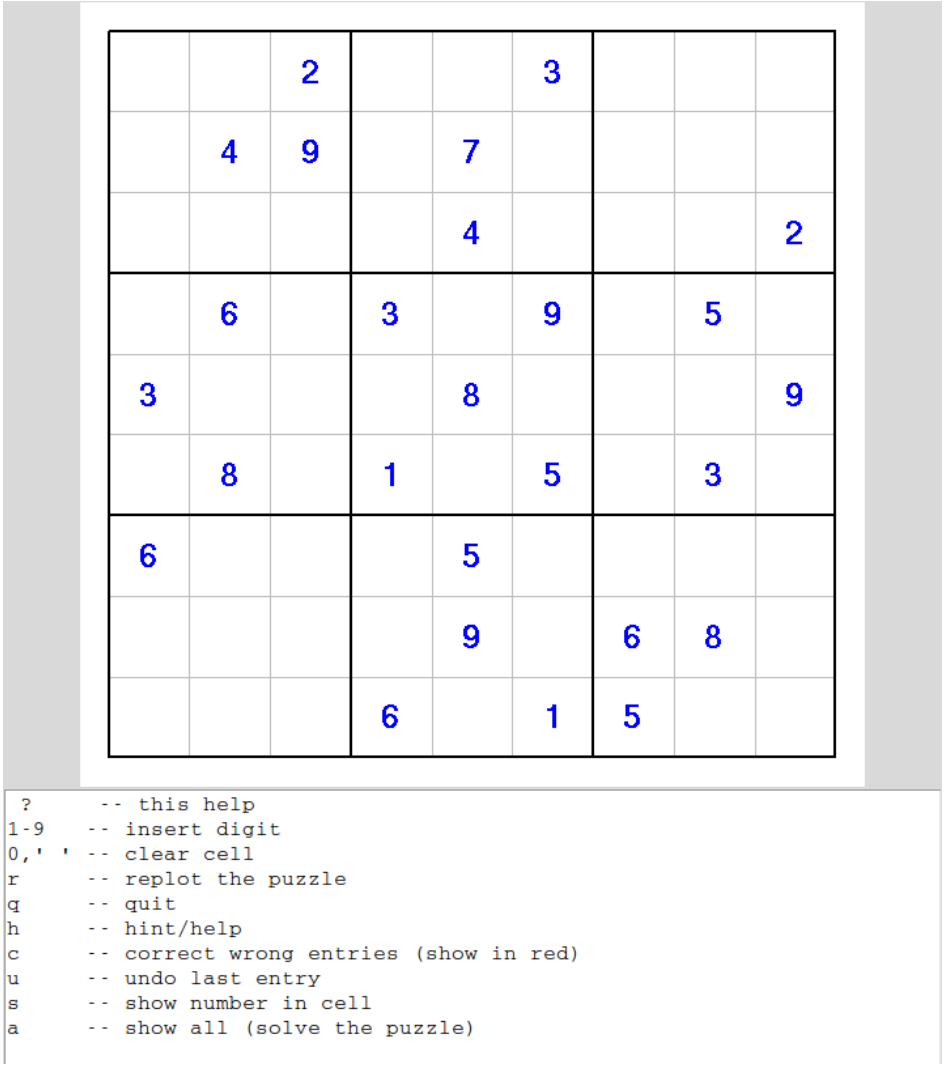
```
printSudoku(fetchSudokuUK())
```

```
> printSudoku(fetchSudokuUK())
```

```
+-------+-------+-------+
|     2 |     3 |       |
|   4 9 |   7   |       |
|       |   4   |     2 |
+-------+-------+-------+
|   6   | 3   9 |   5   |
| 3     |   8   |     9 |
|   8   | 1   5 |   3   |
+-------+-------+-------+
| 6     |   5   |       |
|       |   9   | 6 8   |
|       | 6   1 | 5     |
+-------+-------+-------+
```

Play Sudoku interactively against R

There is even an interactive on-screen game to be played (with hints for cheaters)

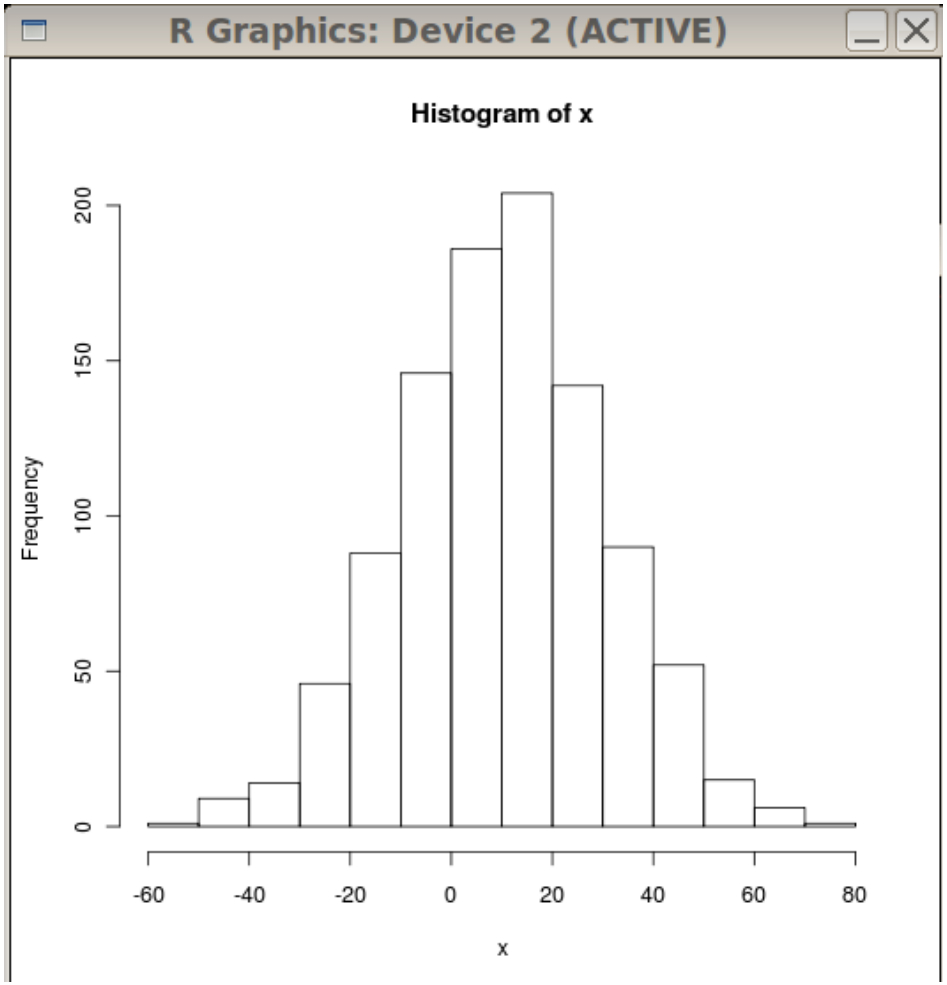


Outline

1. Calculator
2. Statistical Package Framework
3. Graphics
4. Stat Toolbench
5. Programming Language
6. If You Want To Get Started
7. Appendix 1: Code for Simulation Examples

Consider an Ugly Basic Graph

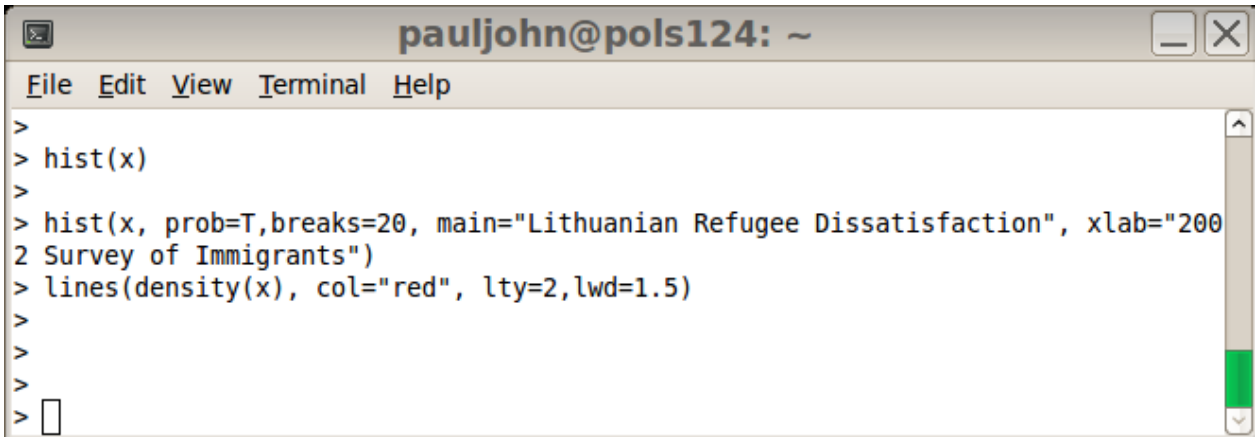
- A graph pop ups when you type "hist(x)"



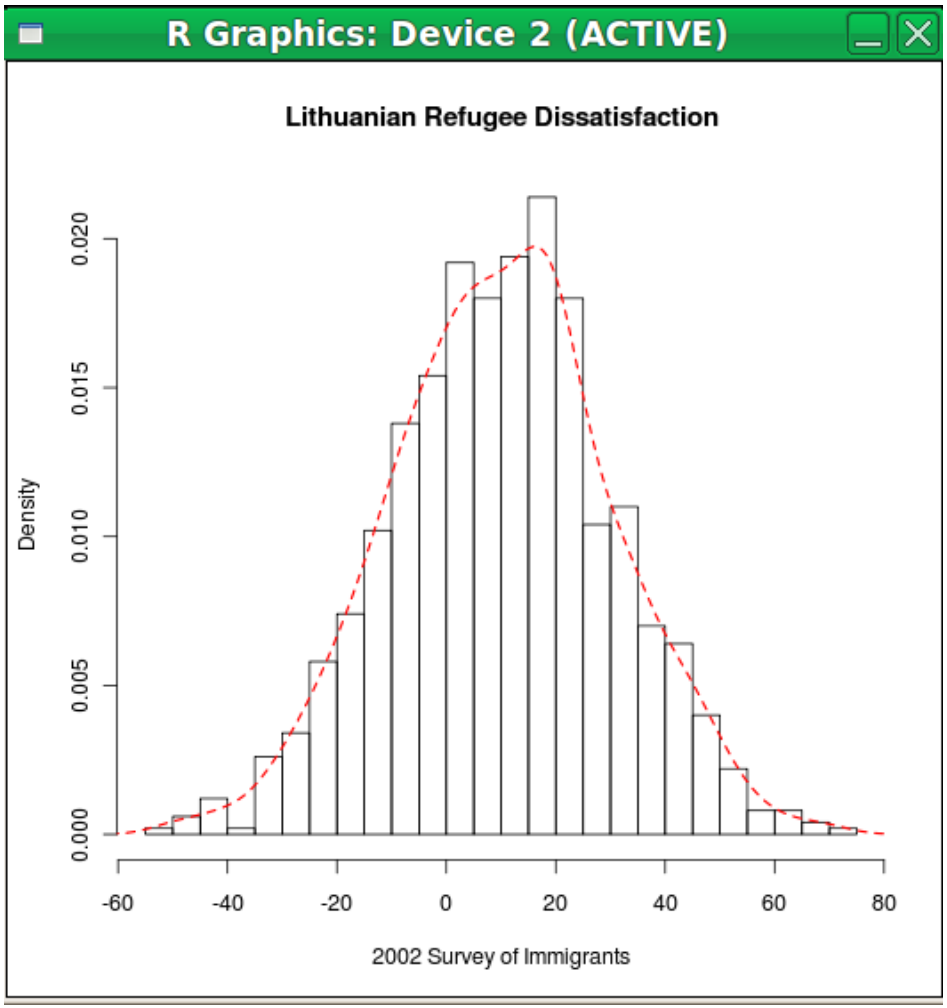
- But clicking on the graph doesn't do anything.

But you do interact with R

- Re-run hist() with more details to beautify the graph.
- Then decorate with "lines()" "text()" etc.



And a nicer looking histogram pops up



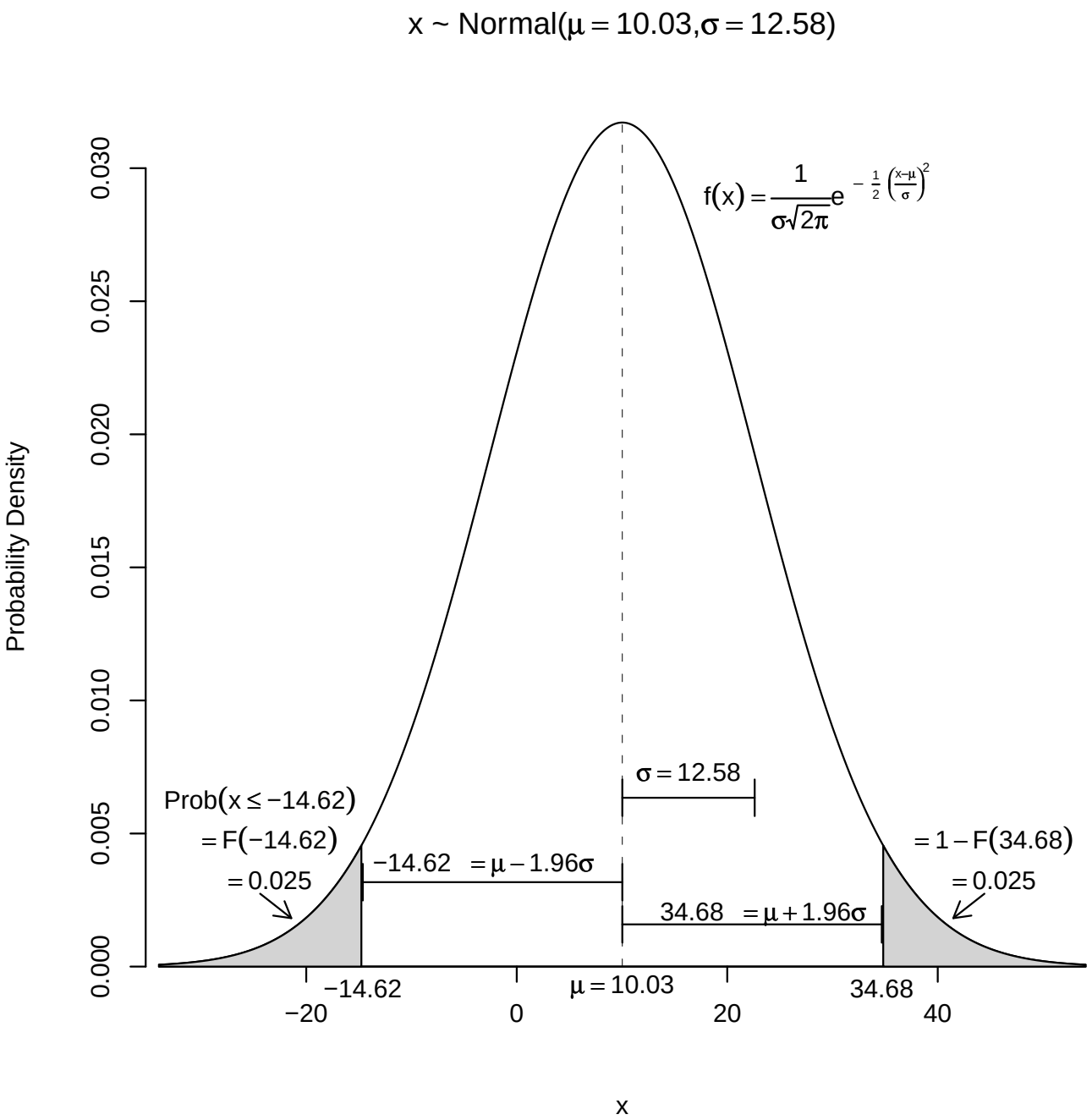
I Use R to Make Line Art

- R can create a "blank canvas"
- Which can then be decorated with subsidiary plotting commands like
  - lines
  - points
  - text
  - polygon

Hold your Seats! Prepare for the Graphic of the Century

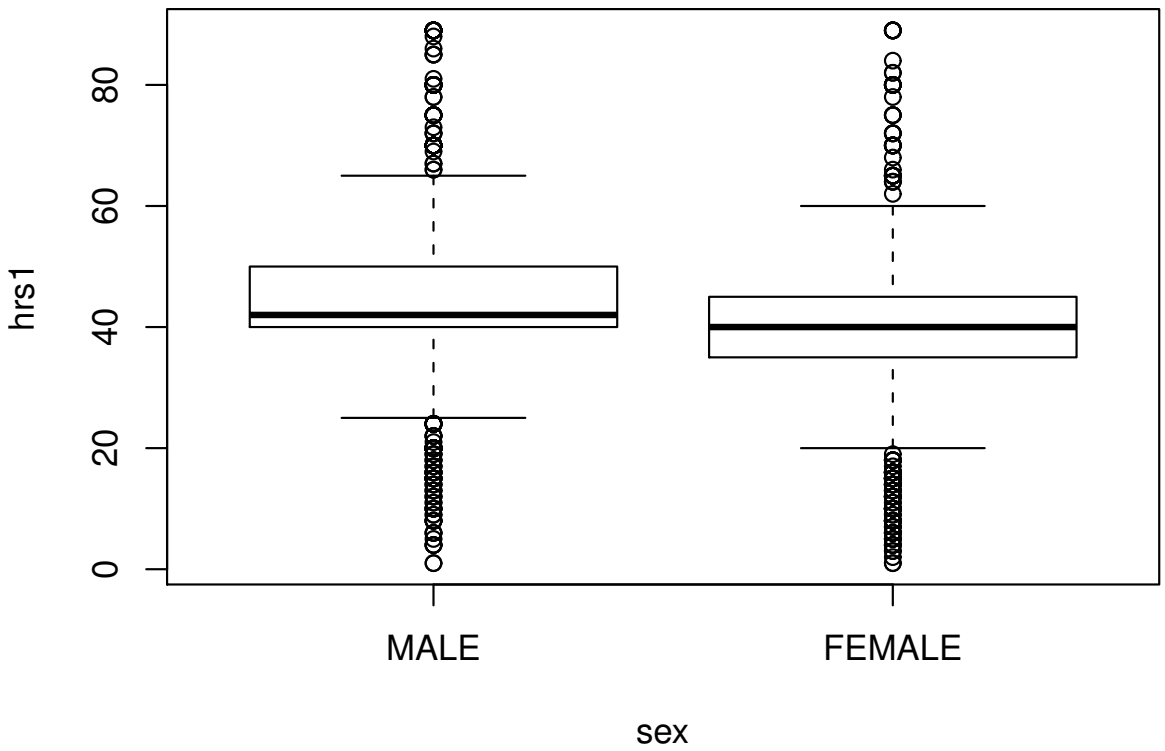
I'm serious. I won't be responsible for injuries to people who faint from a standing position. This sight may be overwhelming to the elderly and infirm.

Be Careful. Sit down.


x ~ Normal(μ = 10.03,σ = 12.58)
f(x) = 1/(σ√2π) e^{−½((x−μ)/σ)²}
Probability Density
0.000
0.005
0.010
0.015
0.020
0.025
0.030
Prob(x ≤ −14.62)
= F(−14.62)
= 0.025
−14.62 = μ − 1.96σ
σ = 12.58
34.68 = μ + 1.96σ
= 1 − F(34.68)
= 0.025
−14.62
μ = 10.03
34.68
−20
0
20
40
x

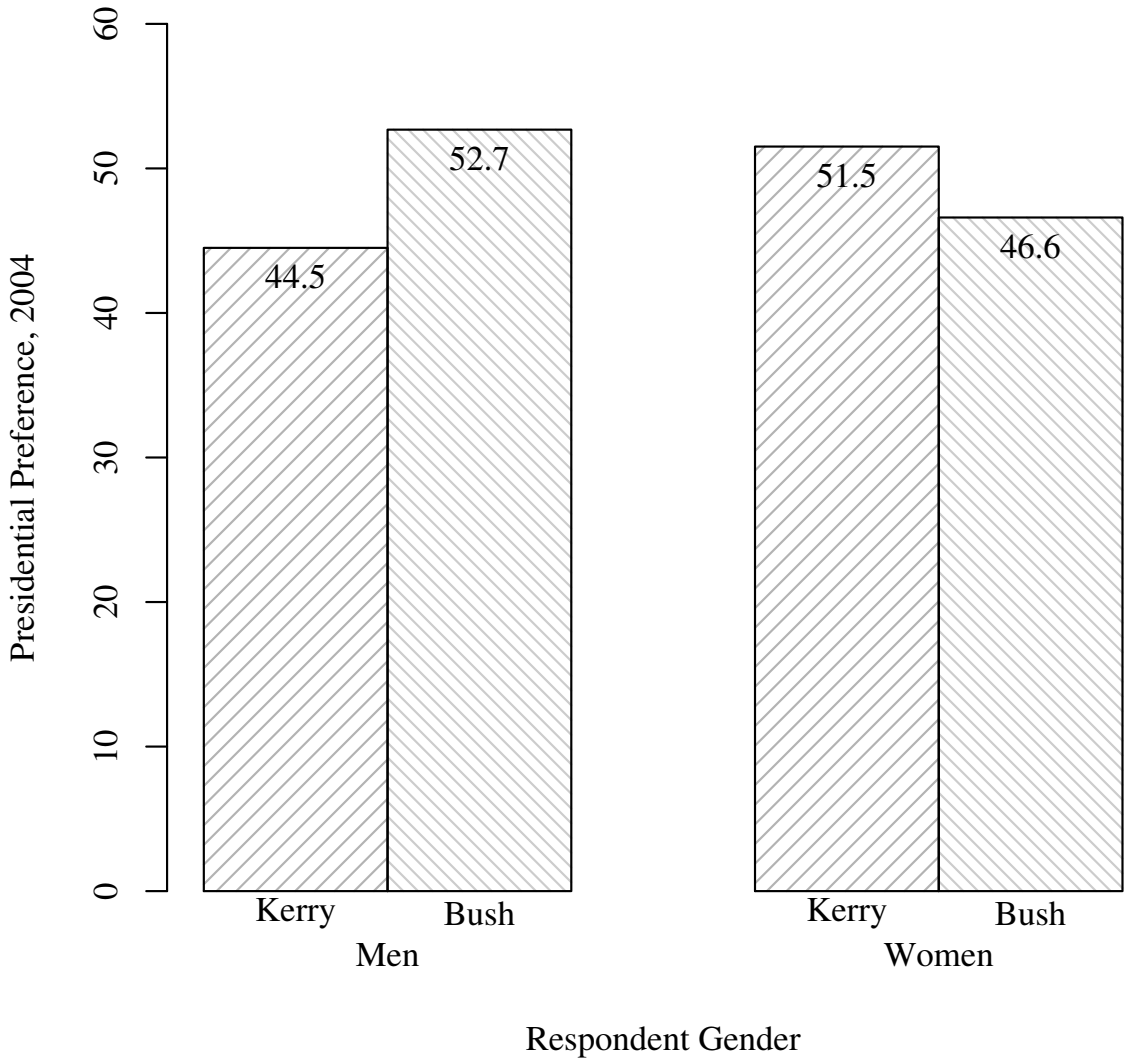
Data Displays are Nice too

"Box and whisker" plot



Barplots are nice too

The Gender Gap in 2004



Outline

1. Calculator
2. Statistical Package Framework
3. Graphics
4. Stat Toolbench
5. Programming Language
6. If You Want To Get Started
7. Appendix 1: Code for Simulation Examples

R has random variables

- Create "random data"
- Want some numbers between 0 and 1?

```
x <- runif(10)
x
```

```
 [1] 0.71694462 0.04077934 0.02560045 0.74061643 0.77014907 0.53867819
 [7] 0.38813644 0.46941770 0.12248648 0.83938921
```

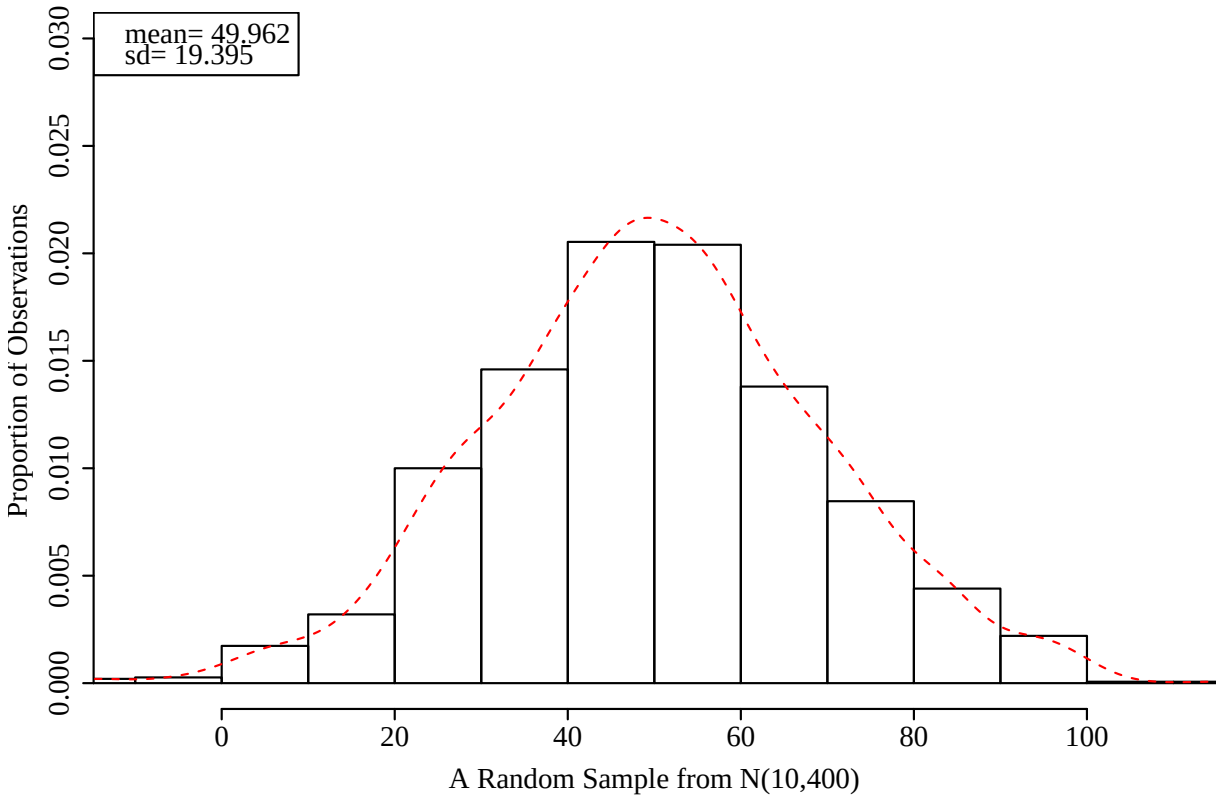
```
mean(x)
```

```
[1] 0.4652198
```

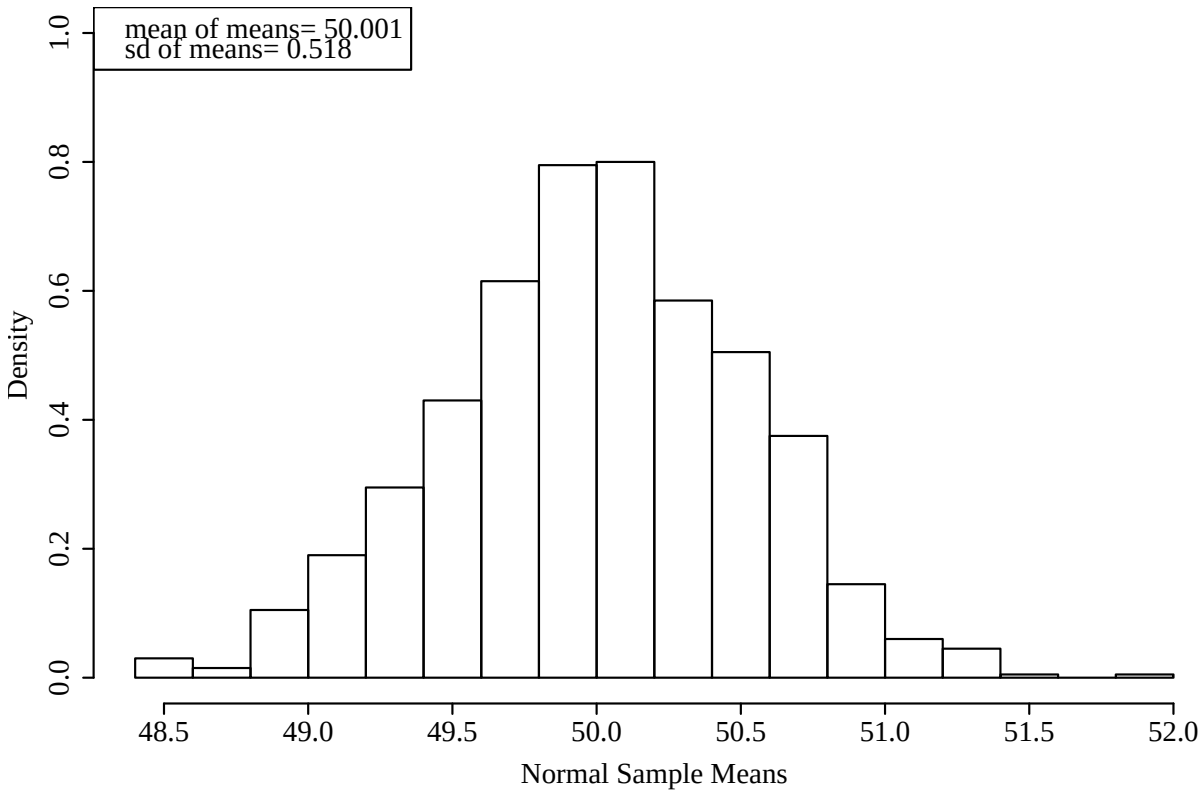
Conduct "Monte Carlo" Experiments

- Draw 1000 samples
- Repeat a calculation with each one
- Consider the 1000 results
- In R this is easy, whereas it is tedious with SAS and impossible with SPSS

One Normal Sample, μ=50, σ=20, 1500 Observations

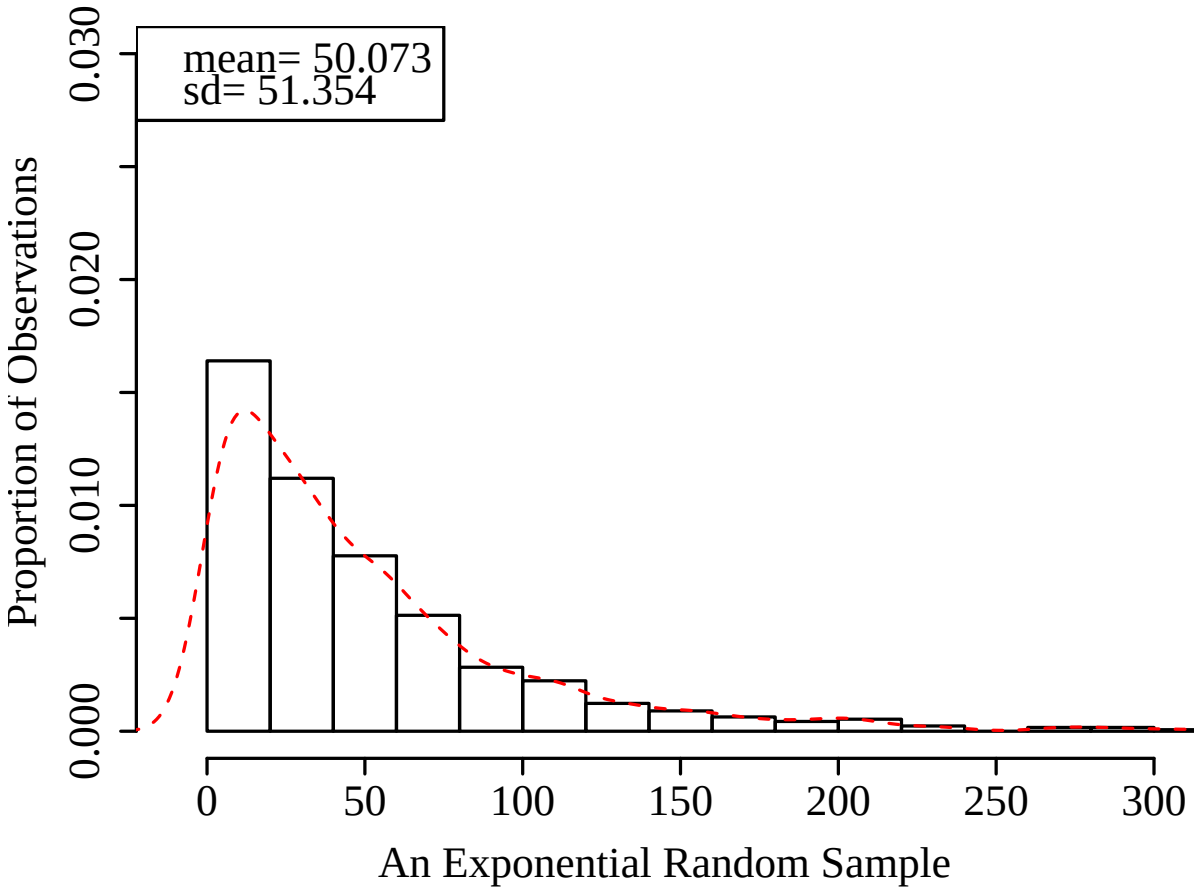


1000 Sample Means

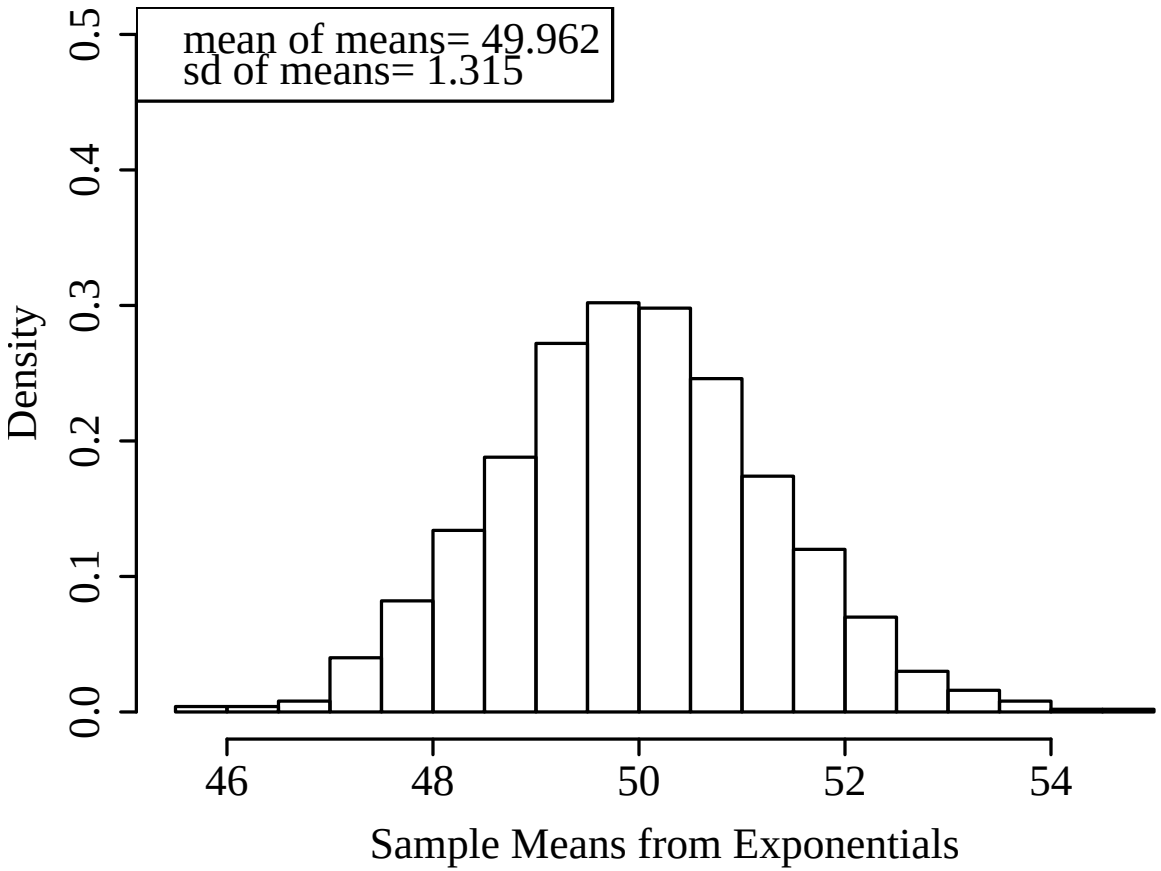


Consistent with theory, means should be Normal(μ=50, σ =20/$\sqrt{1}$500

Sample from an Exponential Distribution



Surprise. Look how Unimodal and "normal" the Exponential Means Look



As we shall see, this is a general phenomenon called "the Central Limit Theorem". Even "funny shaped" distributions have means that are "pleasant"!

Outline

1. Calculator
2. Statistical Package Framework
3. Graphics
4. Stat Toolbench
5. Programming Language
6. If You Want To Get Started
7. Appendix 1: Code for Simulation Examples

It is a Functional Language

One can create functions "on the fly" and then put them to use

```
celToFaren <- function(input=0){ 9/5 * input + 32
    }
celToFaren(100)
```

```
[1] 212
```

```
celToFaren(25)
```

```
[1] 77
```

```
celToFaren(0)
```

```
[1] 32
```

Note we get Free Vectorization

```
mytemps <- 50 * runif(10)
mytemps
```

```
[1] 13.368527  9.755055 12.565008 19.456645 42.179980 49.029872
[7] 15.315985 13.520839 31.146880 31.192590
```

```
celToFaren(mytemps)
```

```
[1]  56.06335  49.55910  54.61701  67.02196 107.92396 120.25377
[7]  59.56877  56.33751  88.06438  88.14666
```

There's a Lot of Computer Science in There

This is not the time to go in to detail, but here's the big idea.

- A function can create an "object" and mark it with a "class" indicator
- Other functions can receive that object, inspect its class, and then "do the right thing."
- In R packages, policy says use the "period" as a joining character for functions that are applied to certain types of things, such as
  - "plot.lm" to plot lm objects
  - "summary.lm" to summarize an lm object
  - "vcov.lm" to extract the variance matrix from an lm object
- Hence, a commonly used idioms like

```
mod1 <- lm ( y ~ x, data=mine)
summary(mod1)
plot(mod1)
```

R Reinterprets and Re-arranges Input

- User can freely rearrange arguments, optional to name them if context is clear

```
plot( myinput , myoutput)
```

  - Same as

```
plot(x = myinput , y = myoutput)
```

  - Same effect as

```
plot(y = myoutput , x = myinput)
```

- Can abbreviate argument names if unique.

```
plot(x1 , y1 , main = "my name")
```

  - Same as

```
plot(x1 , y1 , m = "my name")
```

Verbose code may be Clear, But its also Verbose

- The R experts prefer brevity
- I tend to like fully named function arguments, probably because I'm a teacher

Sudoku, for example

R interprets all of these commands in the same way:

```
> generateSudoku(Nblank=20, print.it = TRUE)
> generateSudoku(20,T)
> generateSudoku(N=20, p=T)
> generateSudoku(p=T, N=20)
```

R will try to match up the options with your arguments, but I try to avoid gambling by explicitly naming options.

This does not give what you want because the arguments are out of order and unnamed

```
> generateSudoku(T, 20)
```

I Like "Camel Case" names

- I don't mind smashing together words like "myX" or "smallSampleData".
- Historically, "_" was the assignment operator in S, so I don't use that in R names.
- Period "." is a joining character in R functions that are part of the "class" structure

Be Careful about the Names You Choose

- Don't steal names of R "built in" functions and variables.
- Naming variables by special names like "mymod1" or such offers some protection.
- More formally, the function "exists()" will ask R if a symbol is currently used.

```
exists("sqrt")
```

```
[1] TRUE
```

```
exists("c")
```

```
[1] TRUE
```

About Those Parentheses

- Parentheses are required to let R know you are trying to call one of its functions
  - To quit R, run the quit() function, for which q is an abbreviation:

```
q()
```

- Without parentheses, it thinks you want it to print the contents of "q" function.

Many Functions Let You Read Them

- q is not interesting, but it is there.
- And many other functions are there. Please run:■eval=F■= lm lm.fit predict.glm
- That doesn't show the "actual R source code", but rather one stylized, tidied up presentation of the logical structure of the function after R has read the source code and gobbled it into the runtime engine (See the rockchalk vignette "Rstyle" for an explanation).

Outline

1 Calculator

2 Statistical Package Framework

3 Graphics

4 Stat Toolbench

5 Programming Language

6 If You Want To Get Started

7 Appendix 1: Code for Simulation Examples

R usage for Dummies

My new policy. Students should follow my "Workspace Advice" for R.[1]. Keep related files IN A FOLDER! In essence,

1. Create a "folder"
2. Copy a template R file into that folder
3. Open that R file with the programmer's file text editor (for me, Emacs)
4. Launch an R session inside the editor's awareness, so code can be "sent" to R for evaluation.
5. Develop the R code by going back-and-forth between the "program buffer" and the "R buffer"

[1] I put it in the Emacs wiki, it must be right!
http://www.emacswiki.org/emacs/CategoryESS

Commands on left, R session on Right

distrodemo.R

File Edit Options Buffers Tools Imenu-S ESS Help

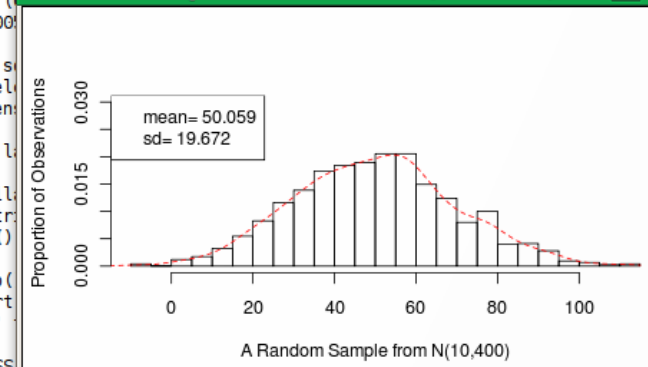
```
##################################################
### chunk number 2: Roptions
##################################################
options(width=60, continue="  ")
###Leave less white space at top
options(SweaveHooks=list(fig=function() par(mar=c(5.1, 4.1, 0.5, 2.1))))
###Sweave appears to ignore following settings 2010-03-20
ps.options(horizontal=F, onefile=F, family="Times",    paper="special", height=4, width=6 )
pdf.options(onefile=F,family="Times",paper="special",height=4,width=6)
options(papersize="special")



##################################################
### chunk number 3: fig1
##################################################
var1 <- rnorm(n=1500, mean=50, sd=20)
hist(x=var1, prob=T, breaks= 20, xlim=c(-10,110), ylim=c(0,0.03), xlab="A Random Sample from N(10,400)", ylab="Proportion of Observations", main="")
den1 <- density(var1)
lines(den1, lty=2, col="red")
legend("topleft",legend=c(paste("mean=",round(mean(var1),3)),paste("sd=",round(sd(var1),3))))


##################################################
### chunk number 4: fig2
##################################################
plot(den1, xlim=c(-10,110), ylim=c(0,0.03), xlab="Possible Values", type="l", lty=2, col="red",main="")
possValues <- seq(-10,110)
trueProbs <- dnorm(possValues, mean=50, sd=20)
lines(possValues, trueProbs, lty=1, col="black")
legend("topright", legend=c("true under N(50,400)","observed in sample"),lty=c(1,2),col=c("black","red"))


##################################################
### chunk number 5: fig3
##################################################
samp <- replicate(1000, mean(rnorm(n=1500, mean=50, sd=20)))
hist(samp, prob=T, breaks=20, ylim=c(0,1), xlab="Normal Sample Means", main="")
```

--:--- **distrodemo.R** Top L28 (ESS[S] [R] Rox)----------

R

File Edit Options Buffers Tools iESS Complete In/Out Signals Help

R Graphics: Device 2 (ACTIVE)




```
R version 2
Copyright (
ISBN 3-9000

R is free s
You are wel
Type 'licen

  Natural l

R is a coll
Type 'contr
'citation()

Type 'demo(
'help.start
Type 'q()'

> .help.ESS
> options(S
> options(width=60, continue="  ")
> options(SweaveHooks=list(fig=function() par(mar=c(5.1, 4.1, 0.5, 2.1))))
> ps.options(horizontal=F, onefile=F, family="Times",    paper="special", height=4, width=6 )
> pdf.options(onefile=F,family="Times",paper="special",height=4,width=6)
> options(papersize="special")
> var1 <- rnorm(n=1500, mean=50, sd=20)
> hist(x=var1, prob=T, breaks= 20, xlim=c(-10,110), ylim=c(0,0.03), xlab="A Random Sample from N(10,400)", ylab="Proportion of Observations", main="")
>
> den1 <- density(var1)
> lines(den1, lty=2, col="red")
> legend("topleft",legend=c(paste("mean=",round(mean(var1),3)),paste("sd=",round(sd(var1),3))))
>
```

-U:**- *R* All L33 (iESS [R]: run)----------

Emacs is like Democracy. Its the worst, except for all of the others that have been tried...

- Emacs
  - Free
  - Available on all platforms (Mac (AquaMacs)), Windows
  - Highly configurable
  - Useful for many other kinds of projects.

I'm not a Mac User, but...

I observe

- R for Macintosh is provided with a MUCH better editor than the one which is provided with R for Windows. It has
  - indentation
  - paren matching
  - highlighting
- So, if you are a Mac user, it is not bad to use the base Macintosh editor
  - and then ignore the Windows and Linux users who fight about which editor is best
- But you might also be aware of Emacs for Macintosh, AquaMacs. If you learn to use that, then you can be comfortable if you go onto other operating systems.

KU

Other Editors: Multi-Platform

RStudio , a somewhat limited but more idiot-proof R "integrated development environment" (IDE). This is not a general purpose programming editor, but rather it is intended for convenience of R elementary users.

- I recommend this for R novices who don't have much experience at installing software. Almost always, it finds R and interacts with it.
- Disadvantages:
  - horrible interaction with plot devices
  - frustrating Rstudio-specific package management framework

Eclipse An expansive, general purpose programming editor and IDE with a special plugin for R. Has many eager proponents. In 2009, I thought Eclipse would take over the world.

vim The updated version of 'vi' (pronounced "vee-eye"). Like Emacs, was developed in the time before mice. Many of the most disciplined programmers I know cling to vi like a flotation device.

Rcmdr An R packages that provides "pull down menu" system provided by Prof. John Fox in support of his excellent stats textbooks.

KU

Other Editors: Multi-Platform ...

- Disadvantage
  - requires the tcltk programming library (which is becoming more tenuous)
  - makes it very easy to run some commands, but others completely omitted

JGR An R package that launches a program editor in Java. This still works, but it appears most of the people who would use it are now adopting RStudio.

Other Editors: Windows Only

- Notepad++, including the "addon" NPPTOR. A better program editor that RStudio, and NPPTO R allows a function key (usually F8) to send lines to an R session. This is the most popular option among the Windows-using R programmers that I know.
  - I don't use it because it is Windows only (why hobble oneself by marrying an OS?).
- WinEdt: a commercial product that was quite popular before Notepad++ was introduced.

Outline

1. Calculator
2. Statistical Package Framework
3. Graphics
4. Stat Toolbench
5. Programming Language
6. If You Want To Get Started
7. Appendix 1: Code for Simulation Examples

Draw a Sample from the Normal, Create a Histogram

```
var1 <- rnorm(n = 1500, mean = 50, sd = 20)
hist(x = var1, prob = T, breaks = 20, xlim =
    c(-10,
     110), ylim = c(0, 0.03), xlab = "A Random
        Sample from N(10,400)",
     ylab = "Proportion of Observations", main =
        "")
den1 <- density(var1)
lines(den1, lty = 2, col = "red")
legend("topleft", legend = c(paste("mean=",
    round(mean(var1), 3)), paste("sd=",
        round(sd(var1),
    3))))
```

Compare Theoretical Probabilities and Observed Sample

```
plot(den1, xlim = c(-10, 110), ylim = c(0,
    0.03), xlab = "Possible Values", type = "l",
    lty = 2, col = "red", main = "")
possValues <- seq(-10, 110)
trueProbs <- dnorm(possValues, mean = 50,
    sd = 20)
lines(possValues, trueProbs, lty = 1, col =
   "black")
legend("topright", legend = c("true under
   N(50,400)",
    "observed in sample"), lty = c(1, 2),
    col = c("black", "red"))
```

Draw Lots of Samples, Calculate their Means, and Plot

```
samp <- replicate(1000, mean(rnorm(n = 1500,
    mean = 50, sd = 20)))
hist(samp, prob = T, breaks = 20, ylim = c(0,
    1), xlab = "Normal Sample Means", main = "")
legend("topleft", legend = c(paste("mean of
   means=",
    round(mean(samp), 3)), paste("sd of means=",
    round(sd(samp), 3))))
```

Re-scale the Previous Histogram

```
hist(samp, prob = T, breaks = 20, xlab = "Normal
    Sample Means",
     xlim = c(-10, 110), ylim = c(0, 1), main = "")
legend("topleft", legend = c(paste("mean of
    means=",
     round(mean(samp), 3)), paste("sd of means=",
     round(sd(samp), 3))))
```

Create and Plot an Exponential Variate

```
var1 <- rexp(n = 1500, rate = 1/50)
hist(x = var1, prob = T, breaks = 20, xlim =
    c(-10,
     300), ylim = c(0, 0.03), xlab = "An
        Exponential Random Sample",
     ylab = "Proportion of Observations", main =
        "")
den1 <- density(var1)
lines(den1, lty = 2, col = "red")
legend("topleft", legend = c(paste("mean=",
    round(mean(var1), 3)), paste("sd=",
        round(sd(var1),
    3))))
```

The Central Limit Theorem is Correct

```
samp <- replicate(1000, mean(rexp(n = 1500,
    rate = 1/50)))
hist(samp, prob = T, breaks = 20, ylim = c(0,
    0.5), xlab = "Sample Means from Exponentials",
    main = "")
legend("topleft", legend = c(paste("mean of
   means=",
    round(mean(samp), 3)), paste("sd of means=",
    round(sd(samp), 3))))
```

References

R Core Team (2018). *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria: R Foundation for Statistical Computing.

Session

```
sessionInfo()
```

```
R version 4.0.2 (2020-06-22)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 20.10

Matrix products: default
BLAS:   /usr/lib/x86_64-linux-gnu/blas/libblas.so.3.9.0
LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.9.0

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
 [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
 [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
 [9] LC_ADDRESS=C               LC_TELEPHONE=C
[11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods
[7] base

other attached packages:
[1] sudoku_2.6          stationery_0.98.30
```

Session ...

```
loaded via a namespace (and not attached):
 [1] Rcpp_1.0.5       digest_0.6.27    plyr_1.8.6       xtable_1.8-4
 [5] evaluate_0.14    zip_2.1.1        rlang_0.4.8      stringi_1.5.3
 [9] openxlsx_4.2.3   rmarkdown_2.5    tools_4.0.2      foreign_0.8-80
[13] kutils_1.70      xfun_0.19        compiler_4.0.2   htmltools_0.5.0
[17] knitr_1.30
```