R

Presentation by: Joshua D. Habiger
Oklahoma State University
Department of Statistics

November, 2010

What is R?

R is a *free*, *object oriented* programming language and software environment for statistical computing and graphics

Some History

1978: S project begins at Bell Labs
- John Chambers et. al

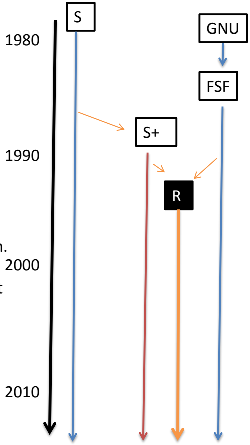
1985: Chambers' transition to functions fails

1988: S+ Sofware produced by Statistical Sciences Inc.
- Function oriented
- Douglas Martin, U. Wash.

1991: Ross Ihaka and Robert Genteman develop R
- It mimics S+

2000: R version 1.0 Released as a GNU project

2010: R version 10.0 Released as a GNU project



1983: GNU Manefesto
- Richard Stallman
- All programs should be free

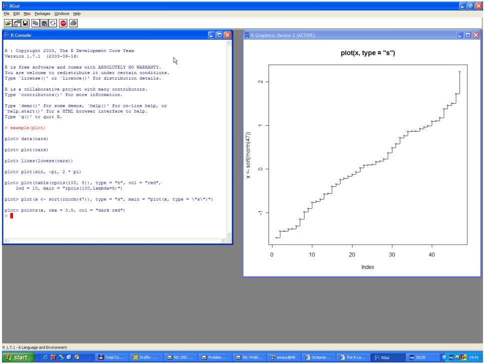
1985: Free Software Foundation

1996: Ihaka and Gentleman get GNU General Public License for R
- "Copyleft"

- **Operating Characteristics**
- Functions
- Graphics
- Creating Functions
- Conclusion

Starting R

- To run R from the terminal type R



R Workspace

The **R workspace** is your current working environment.

- Contains all user defined **objects** and some default objects
- You can load a workspace, define some objects, then save the workspace

```
> load("C:\\Users\\Habiger\\Documents\\myworkspace.RData")
> define some objects
Error: unexpected symbol in "define some"
> save.image("C:\\Users\\Habiger\\Documents\\myworkspace.RData")
```

Defining Objects

- An **object** is defined with an "<-" or an "="
- x,y,z,v exist as objects in the workspace

```
> x<-c(1,2,3)
> y = "y is a character and x is a vector"
> z<-list(x,y,element3="A list can contain characters")
> v<-matrix(1:10, nrow=5)
```

Getting Output

- For most objects, just type the name of the object
- Sometimes you will need summary(object)... more later

```
> x
[1] 1 2 3
> z
[[1]]
[1] 1 2 3
[[2]]
[1] "y is a character and x is a vector"
$element3
[1] "A list can contain characters"
```

Getting elements of an Object

- Two ways to refer to a “named” element in an object

```
>z[[3]]
$element3
[1] "A list can contain characters"
>
> z$element3
[1] "A list can contain characters"
```

- For the matrix *v* you could use v[1,2], v[,1], v[2,].
- Syntax for referring to elements of an object depends on the objects **class**.

Classes

An **object** belongs to a **class**.

- Ex: Numeric, Logical, Character, Vector ...

```
> class(x)
[1] "numeric"
>is.vector(x)
[1] TRUE
>class(z)
[1] "list"
>class(z[[3]])
[1] "character"
```

More on Classes

- **Objects** can be coerced into a different **class**

```
> w<-T
> w
[1] TRUE
>is.logical(w)
[1] TRUE
> as.numeric(w)
[1] 1
> class(w)<-"numeric"
> is.logical(w)
[1] FALSE
```

Coercing Classes

- R will attempt to coerce objects into a class for you if necessary

```
>x=T
> x
[1] TRUE
> x+pi
[1] 4.141593
```

- Uses: Can perform operations on sets
  - $P(-1.96 < Z < 1.96) = \int_{-\infty}^{\infty} I(-1.96 < z < 1.96) f(z) dz$

```
> integrate(function(z){(-1.96<z&z<1.96)*dnorm(z)}, lower=-Inf, upper=Inf)
[1] 0.9499932 with absolute error < 8.2e-05
```

Matrix

- Operations are performed element-wise unless otherwise specified

```
> x<-matrix(c(1,2),ncol=1)
> x
     [,1]
[1,]    1
[2,]    2
> y
     [,1] [,2]
[1,]    1    1
> y*x
Error in y * x : non-conformable arrays
> y%*%x
     [,1]
[1,]    3
```

Lists and Arrays

- Lists vs. Arrays

```
> x<-list(T)
> y<-array(T)
> x+1
Error in y + 1 : non-numeric argument to binary operator
> y+1
[1] 2
```

Data Frames

- Data frames useful for statistical modeling

```
> data.frame(matrix(1:10,5), X3=c(T,T,F,F,F))
  X1 X2    X3
1  1  6  TRUE
2  2  7  TRUE
3  3  8 FALSE
4  4  9 FALSE
5  5 10 FALSE
```

- Operating Characteristics
- **Functions**
- Graphics
- Creating Functions
- Conclusion

What are functions

Functions are **objects** belonging to the class “function”.

```
>function(input)
output
```

- Input and output for functions can be any object

Help

- To **find** a function: help.search("description")
- To learn how to **use** a function: help(function name)

```
help.search("linear model")
help(glm)
```

help(glm)

glm {stats} R Documentation

Fitting Generalized Linear Models

Description

`glm` is used to fit generalized linear models, specified by giving a symbolic description of the linear predictor and a description of the error distribution.

Usage

```
glm(formula, family = gaussian, data, weights, subset,
    na.action, start = NULL, etastart, mustart, offset,
    control = list(...), model = TRUE, method = "glm.fit",
    x = FALSE, y = TRUE, contrasts = NULL, ...)
....
```

Arguments

`formula` an object of class "formula" (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under 'Details'.

`family` a description of the error distribution and link function to be used in the model. This can be a character string naming a family function, a family function or the result of a call to a family function. (See family for details of family functions.)

`data` an optional data frame, list or environment (or object coercible by as.data.frame to a data frame) containing the variables in the model. If not found in `data`, the variables are taken from `environment(formula)`, typically the environment from which `glm` is called.

`weights` an optional vector of 'prior weights' to be used in the fitting process. Should be `NULL` or a numeric vector.

.....

Example: glm

```
> model=glm(log(y)~x*z)
>model

Call:  glm(formula = log(y) ~ x * z)

Coefficients:
(Intercept)            x            z          x:z
    -0.4826      -0.1274       0.1954       0.1329

Degrees of Freedom: 10 Total (i.e. Null);  7 Residual
Null Deviance:       101.6
Residual Deviance: 72.47        AIC: 61.95
```

More glm

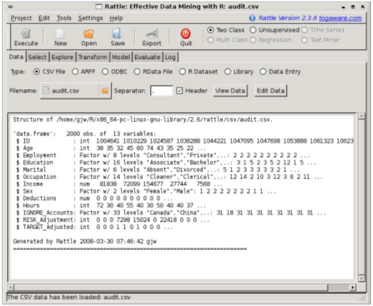
```
> summary(model)
Call:
glm(formula = log(y) ~ x * z)

Deviance Residuals:
    Min        1Q    Median       3Q       Max
-7.5335   0.1449   0.8711   1.5241   1.7335

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -0.48256    1.38929  -0.347    0.739
x           -0.12741    1.16377  -0.109    0.916
z            0.19544    1.03613   0.189    0.856
x:z          0.13293    0.08494   1.565    0.162

(Dispersion parameter for gaussian family taken to be 10.35218)
    Null deviance: 101.611  on 10  degrees of freedom
Residual deviance:  72.465  on  7  degrees of freedom
AIC: 61.954
Number of Fisher Scoring iterations: 2
```

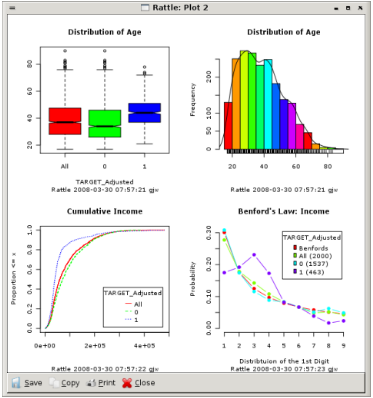
- Try also plot(model), residuals(model), coefficients(model), anova(model)...

Non Standard Functions

- Functions for relatively new methodology may not be in the base **package**.
- They may exist in another **package**.
  - Thousands of packages
  - Can be installed from the file menu for Windows GUI.
- Some packages come with their own GUI's
  - Rattle: Gnome Cross Platform GUI for Data Mining using R

Rattle

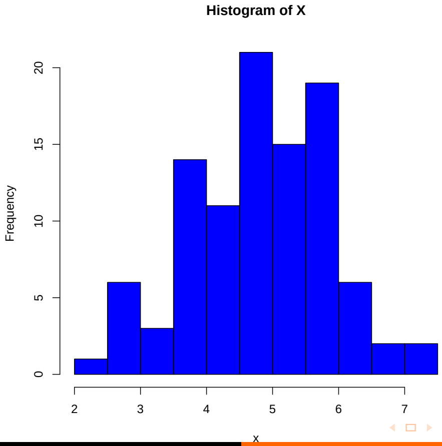

Rattle: Effective Data Mining with R: audit.csv
Project Edit Tools Settings Help
Rattle Version 2.3.6 togaware.com
Execute New Open Save Export Quit
Two Class Unsupervised Time Series
Multi Class Regression Text Miner
Data Select Explore Transform Model Evaluate Log
Type: CSV File ARFF ODBC RData File R Dataset Library Data Entry
Filename: audit.csv Separator: , Header View Data Edit Data
Structure of /home/gjw/R/x86_64-pc-linux-gnu-library/2.6/rattle/csv/audit.csv.
'data.frame': 2000 obs. of 13 variables:
$ ID : int 1004641 1010229 1024587 1038288 1044221 1047095 1047698 1053888 1061323 10623
$ Age : int 38 35 32 45 60 74 43 35 25 22 ...
$ Employment : Factor w/ 8 levels "Consultant","Private",..: 2 2 2 2 2 2 2 2 2 2 ...
$ Education : Factor w/ 16 levels "Associate","Bachelor",..: 3 1 5 2 3 5 2 12 1 5 ...
$ Marital : Factor w/ 6 levels "Absent","Divorced",..: 5 1 2 3 3 3 3 3 2 1 ...
$ Occupation : Factor w/ 14 levels "Cleaner","Clerical",..: 12 14 2 10 3 12 3 6 2 11 ...
$ Income : num 81838 72099 154677 27744 7568 ...
$ Sex : Factor w/ 2 levels "Female","Male": 1 2 2 2 2 2 2 2 1 1 ...
$ Deductions : num 0 0 0 0 0 0 0 0 0 0 ...
$ Hours : int 72 30 40 55 40 30 50 40 40 37 ...
$ IGNORE_Accounts: Factor w/ 33 levels "Canada","China",..: 31 18 31 31 31 31 31 31 31 31 ...
$ RISK_Adjustment: int 0 0 0 7298 15024 0 22418 0 0 0 ...
$ TARGET_Adjusted: int 0 0 0 1 1 0 1 0 0 0 ...
Generated by Rattle 2008-03-30 07:46:42 gjw
The CSV data has been loaded: audit.csv


Rattle: Plot 2
Distribution of Age
All
0
1
TARGET_Adjusted
Rattle 2008-03-30 07:57:21 gjw
Distribution of Age
Frequency
Rattle 2008-03-30 07:57:21 gjw
Cumulative Income
Proportion <= x
TARGET_Adjusted
All
0
1
Rattle 2008-03-30 07:57:22 gjw
Benford's Law: Income
Probability
TARGET_Adjusted
Benfords
All (2000)
0 (1537)
1 (463)
Distribtuion of the 1st Digit
Rattle 2008-03-30 07:57:23 gjw
Save Copy Print Close

- Operating Characteristics
- Functions
- **Graphics**
- Creating Functions
- Conclusion

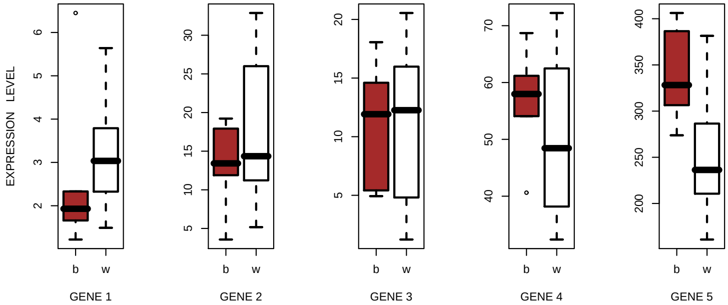
Histograms

```
>hist(rnorm(n=100,mean=5,sd=1),main="Histogram of X",xlab="x", color="blue")
```



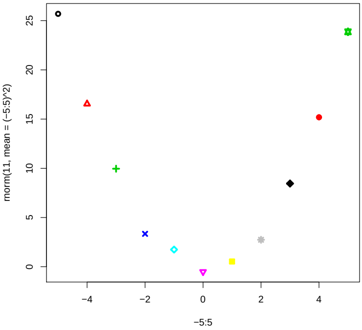
Boxplots

```
>boxplot(...)
```



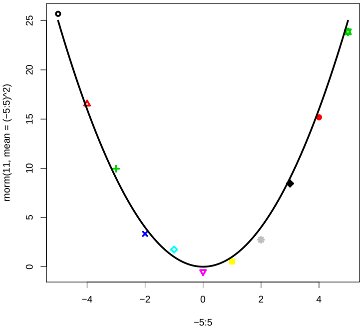
Plot Function

```
>plot(-5:5, rnorm(11,mean=(-5:5)^2), pch=1:11,col=1:11,lwd=3)
```



Plot Function cont.

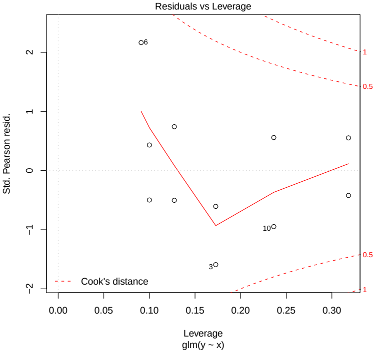
```
> curve(x^2,xlab="",ylab="",add=T,lwd=3)
```



Plot Function cont.

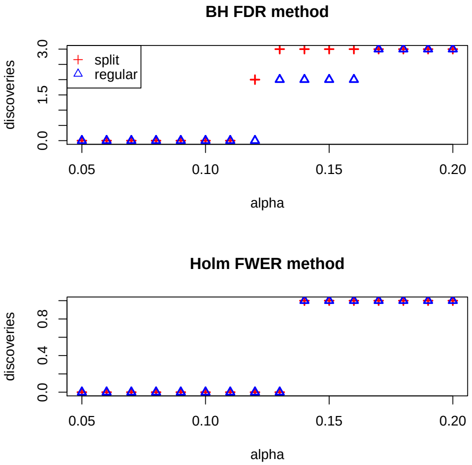
Hit return to get a new plot

```
>plot(model)
Waiting to confirm page change
```



multiple plots

```
par(mfrow=c(2,1))
matplot(...)
legend(...)
matplot(...)
```



Syntax for 3d Plotting

- Three main arguments for many 3d plot functions (the exception is scatter plot-type functions)
  - $X0$ is vector of length m
  - $Y0$ is a vector of length n
  - $Z0 = (z_{ij})$ is an $m \times n$ matrix

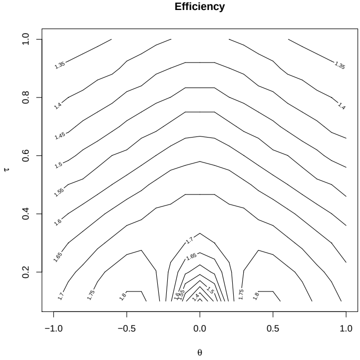
```
>x0<-1:5
>y0<-1:3
>z0<-x%o%y
>z0
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    2    4    6
[3,]    3    6    9
[4,]    4    8   12
[5,]    5   10   15
```

- The pseudo syntax

```
>generic3dplot(x=x0,y=y0,z=z0, option1, option2, ...)
```

Contour Plot

```
> contour(z=myZ0, x=myX0,y=myY0,xlab=expression(theta),...)
```



Another Contour Plot

```
> filled.contour(...)
```

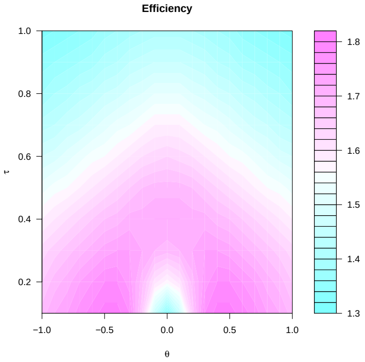
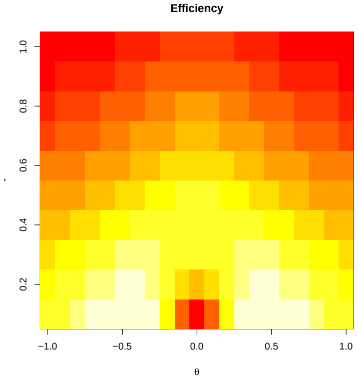


Image Plot

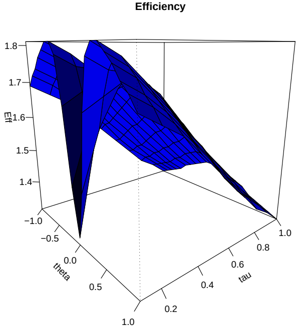
```
> image(...)
```



Perspective Plot

```
> persp(...)
```

Try also persp3d() in the rgl package.



- Operating Characteristics
- Functions
- Graphics
- **Creating Functions**
- Conclusion

Defining a Function

The pseudo syntax is

```
>myfunction<-function(x=defaultx, y= defaulty,...)
  {
  output<-operations on x,y
  return(output)
  }
>mfunction(x,y)
 output
```

- You can use any existing objects in a function
- Objects created within a function will not remain in the workspace

Example with Loop

$$a_n = a_{n-1} + a_{n-2}$$

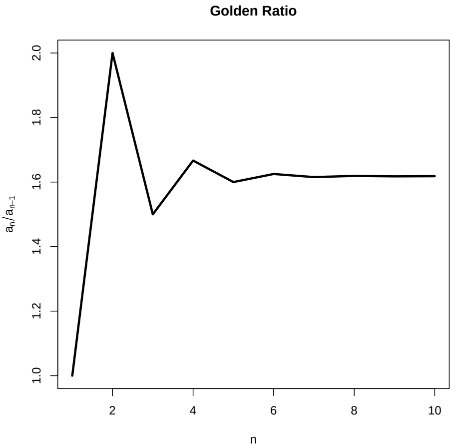
1 + 1 + 2 + 3 + 5 + 8 + 13 + 21 + 34 + 55 +...

```
> fibonacci<-function(n=100)
{
 y<-c(1,1)
 for(i in 1:n)
 {
   y<-c(y,sum(y))
   y<-y[-1]
 }
 return(y[2])
}
>fibonacci()
> fibonacci()
[1] 9.273727e+20

> fibonacci(1001)/fibonacci(1000)
[1] 1.618034
> y
Error: object 'y' not found
```

Another Fibonacci

```
> fibonacci2<-function(n=10)
{
 y<-rep(1,n)
 for(i in 3:n)
 {
   y[i]<-y[i-1] + y[i-2]
 }
 return(y)
}
>fibonacci2(11)[-1]/fibonacci(10)
[1] 1.000000 2.000000 1.500000 1.666667 1.600000 1.625000 1.615385 1.619048
[9] 1.617647 1.618182
```

Plot of Golden Ratio



Example with Roots & Integration

My goal: Find value α s.t.

$$h(\alpha) = \int_0^1 g(p, \alpha) dp = .95$$

where

$$g(p, \alpha) = \sum_{x=0}^{n} I\left(L(x, \alpha) \leq p \leq U(x, \alpha)\right) \Pr(X = x|p)$$

Code for Example

Code closely mimics what I want to do!

```
>U<-function(x,alpha){...return(output)}
>L<-function(x,alpha){...return(output)}
>g<-function(p,alpha){return(sum(I(L(0:n,alpha)<=p&&p<=U(0:n,alpha))dbinom(0:n,p,n)))}
>h(alpha)<-function(p,alpha){return(integrate(g(p,alpha), lower=0, upper=1))[[1]]}
>uniroot(h(alpha)-.95, interval=c(0,1))
[1] .034
```

Optimization

- optim(), constrOptim(), nlm(): most have several methods to choose from

```
> y<-rnorm(50,sd=5,mean=10)
> l<-function(parms){-log(prod(dnorm(y,sd=parms[1],mean=parms[2])))}
> optim(c(20,-5),l)
$par
[1] 5.292001 9.462908

$value
[1] 154.2492

$counts
function gradient
      61       NA

$convergence
[1] 0

$message
NULL

Warning message:
In dnorm(x, mean, sd, log) : NaNs produced
```

Snags with Functions

- R is not efficient with loops
  - apply() function helps
  - .C(), .Fortran()
- Functions like optim(), integrate(), uniroot() require that the function to be integrated/optimized/solved allow for vector inputs and ouputs
  - First just try inputting a vector.
  - Otherwise

```
myfunction<-Vectorize(myfunction)
```

- Operating Characteristics
- Existing Functions
- Graphics
- Creating Functions
- **Conclusion**

Scripting

Useful functions

- getwd(), setwd(), read.table(), write.table(), read.csv(), write.csv()
- paste(), strsplit(), cat(), parse()
- .Python()
- call()

Other Useful Functions/Software

- pdf(), postscript(), dev.off()
- debug()
- cbind(), rbind()
- names(), dim(), colnames(), rownames()
- solve()
- Bioconductor (www.bioconductor.org)

Remarks

- SAS can do anything R can do
- R can do anything SAS can do
- Before you start coding...
    - See if the wheel has been invented

References

- New R GUI: is this the wave of the future? *Statistical Modeling, Causal Inference, and Social Science*, 2009. Andrew Gelman.
- http://cran.r-project.org
  - http://cran.r-project.org/doc/manuals/R-lang.html
  - http://www.r-project.org/doc/bib/R-books.html
- *Software for Data Analysis: Programming with R*, John Chambers

- These slides available at: http://casa.okstate.edu/cas2/Habiger

THANK YOU