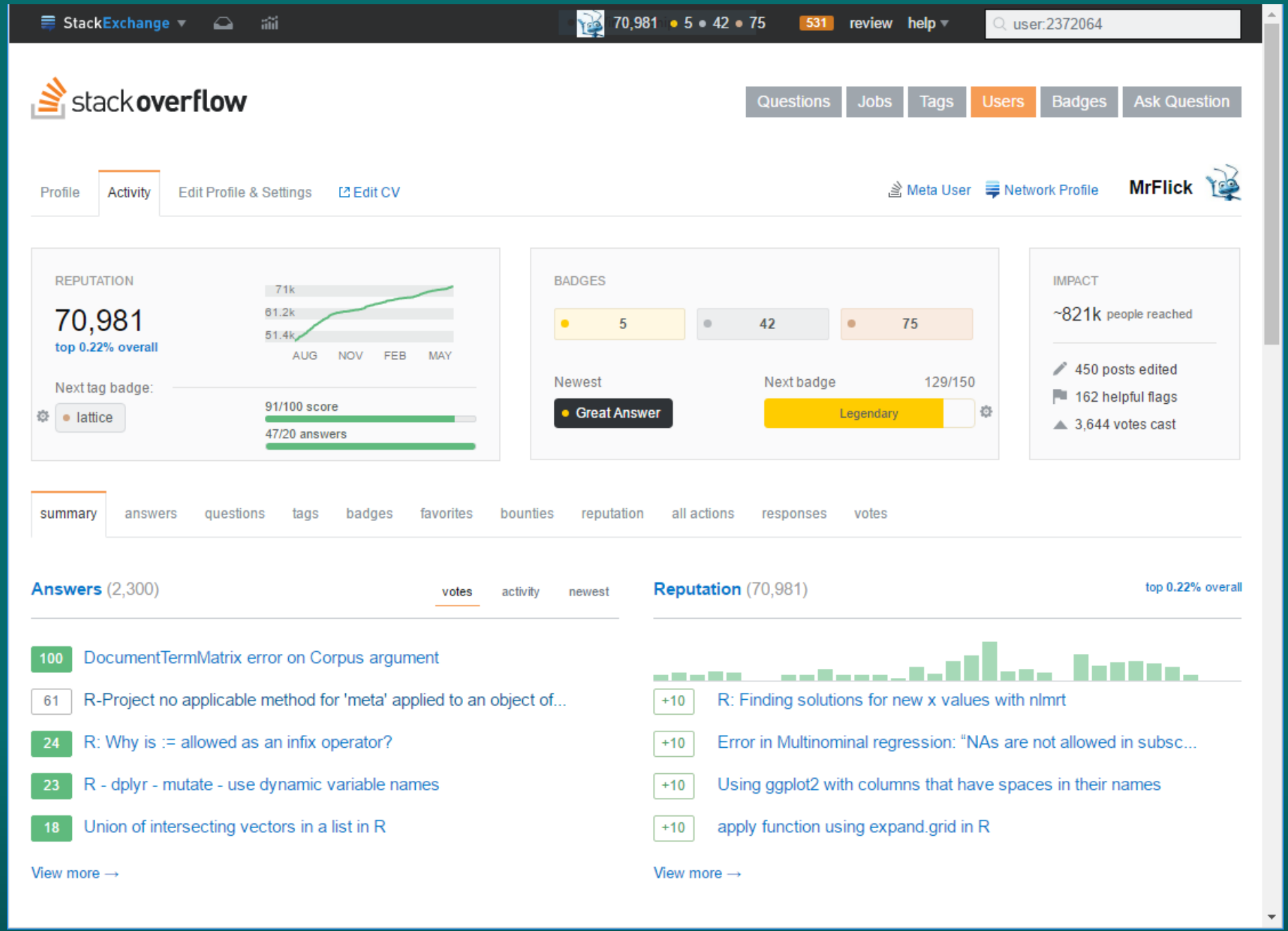
What R You Doing?

Answering Your Own R Questions

Matthew Flickinger, Ph.D.

CSG Tech Talk

May 19, 2016



Learn By Answering Questions

- Stack Overflow [r] tag
- 2,000+ questions answered
- 800,000+ views for my answers

I may not **know** the answer,

But I can **find** the answer -

And so can **you**

Language Overview

```
a <- 1:5
a[3]
b <- (2:6)*2
a + b
c(a,b)
list(a, b, "apple")
```

```
dd <- mtcars
head(dd)
model <- lm(mpg~wt, dd)
summary(model)
coef(model)
plot(mpg~wt, dd)
abline(model)
```

What R you doing?

Different classes result in different behaviors

Basic Data Types

- Numeric (generally "double" or "integer")
  - `c(1,2,3,4)` or `c(.004, -4.2, 17, 123)` or `1:5`
- Character (string)
  - `c("a", "banana", "cool!")`
- Factors (categorical variables)
  - `factor(c("a", "banana", "cool!"))`
- Date/Time values (POSIXct, POSIXlt)
  - `strptime("2015-06-24", "%Y-%m-%d")`

Other Common Types

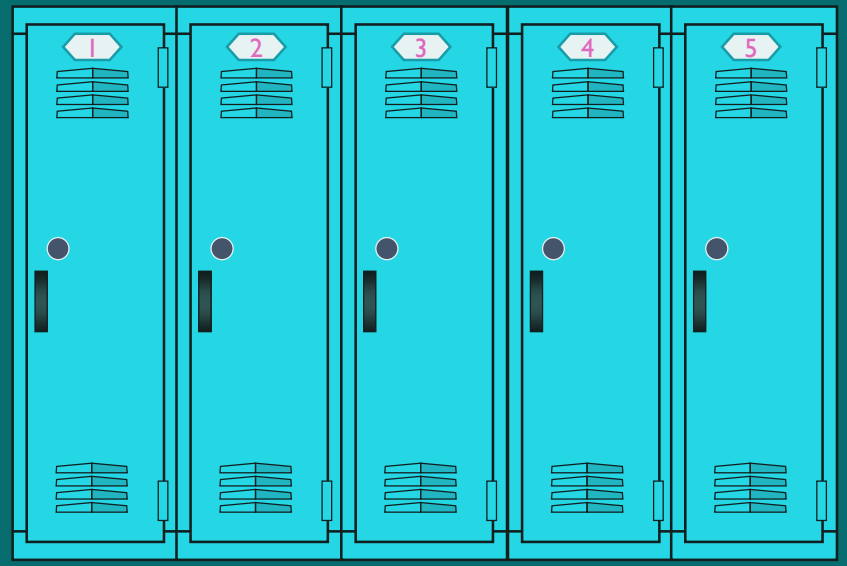
- Arrays and Matrices (numeric or character)
  - `matrix(letters[1:25], ncol=4)`
  - `array(1:(2*4*6), c(2,4,6))`
- Functions
  - `function(x) x+1`
- Formulas (unique to R)
  - `bmi ~ age + gender + calories`
- Environments (less common)
  - `environment()`

Composite Types

- Pretty much everything else is a list
- Data.frame
  - A list of atomic vectors of the same length
  - `data.frame(a=1:3, b=c("f","g","h"))`
- A “raw” list
  - `x <- list(a=1:10, b=function(x) x+5, c=list(x="x", y=a~b))`
- A list is a collection of containers

Subsetting vs Extraction

L=



- Subsetting

L[2]=

Subsetting vs Extraction

L=

- Subsetting

L[2]=

- Extracting

L[[2]]=

What Is This Thing?

- How is this object structured
  - `str(x)`
- What's really in this thing?
  - `dput(x)`
- How does this object behave?
  - `class(x)`

Generic Functions

- Generic functions behave differently based on the class of the first parameter
- See all implementations
  - `methods(print)`
  - `methods("print")`
- See what a class can do
  - `methods(class="lm")`

```
> print
function (x, ...)
UseMethod("print")

> plot
function (x, y, ...)
UseMethod("plot")
```

Model Fitting – Sample Data

```
dd<-read.table(text="
gpa classyear sex
4.0 1 1
3.2 2 2
3.7 3 1
2.9 4 2
3.5 2 2
2.3 2 1", header=TRUE)
```

Class year

- 1 = Freshman
- 2 = Sophomore
- 3 = Junior
- 4 = Senior

Sex

- 1 = Male
- 2 = Female

Model Fitting – Continuous vs Categorical

```
lm(gpa~classyear + sex, dd)
```

```
             Estimate
(Intercept)  3.723809524
classyear   -0.192857143
sex         -0.004761905
```

```
lm(gpa~factor(classyear)
      +factor(sex), dd)
```

```
                  Estimate
(Intercept)           4.00
factor(classyear)2   -1.70
factor(classyear)3   -0.30
factor(classyear)4   -2.15
factor(sex)2          1.05
```

Model Fitting – Variable Recoding

```
dd <- transform(dd,
  classyear = factor(classyear,
    levels=1:4,
    labels=c("Fresh", "Soph",
      "Junior","Senior")),
  sex = factor(sex, levels=1:2,
    labels=c("Male","Female"))
)
```

```
lm(gpa~classyear+sex, dd)
```

	Estimate
(Intercept)	4.00
classyearSoph	-1.70
classyearJunior	-0.30
classyearSenior	-2.15
sexFemale	1.05

What R you showing me?

What you see isn't what you've got

R Console

- The R console behaves as a REPL
  - Read - accept user commands
  - Evaluate - execute those commands
  - Print - display the results
  - Loop - start over with next input
- When running interactively, the results of all statements are print()-ed

Pretty Printing

- R tries to make output look pretty on screen
- Wraps values depending on value of `options("width")`
- Some classes have elaborate printing methods
- See `?print` for more options

```
sqrt(2)
[1] 1.414214
c(sqrt(2), 1e6)
[1] 1.414214e+00 1.000000e+06
c(sqrt(2), 2)
[1] 1.414214 2.000000
print(c(sqrt(2), 2), digits=2)
[1] 1.4 2.0
print(sqrt(2), digits=12)
[1] 1.41421356237
```

String Escaping

- Special characters are replaced with escape sequences
  - `"\t"` = tab
  - `"\n"` = new line
  - `"\\"` = slash
- `print()` will always display the escaped version
- `cat()` will render the "true" value

```
x <- "Hi\t1\t2\\3\nDone!\n"
x
[1] "Hi\t1\t2\\3\nDone!\n"
cat(x)
Hello   1       2\3
Done!
```

Floating Point Arithmetic

- Computers aren't great with certain fractions
- If comparing numeric values, use `all.equal()`
- Common programming problem (not unique to R)

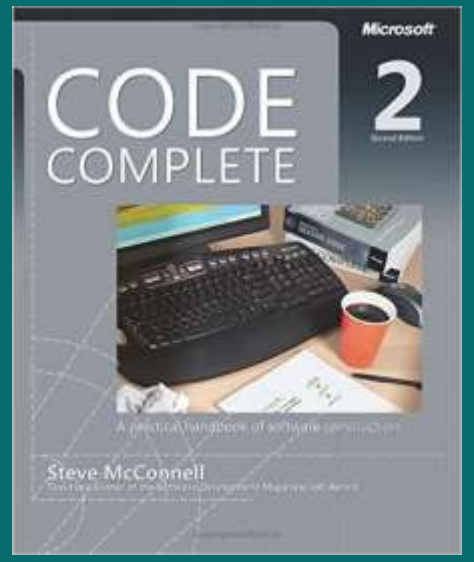
```
a <- 0.1
a <- a + .05
b <- 0.15
a==b
[1] FALSE
all.equal(a, b)
[1] TRUE
sprintf("%a",a)
sprintf("%.20f",a)
```

How R you solving the problem?

Good debugging uses the scientific method

Scientific Debugging

- Research
- Hypothesize
  - Is the problem with the data?
  - Am I passing the right arguments to the function?
  - Are the functions return what I expect?
  - What has changed?
- Test
  - Change one thing at a time
  - Revert changes that don't make a difference
- What have you learned?



Steve McConnell

Find the error

- See where error same from
  - `traceback()`
- Stop running code on error
  - `options(error=browser), options(error=recover)`
  - Turn off with `options(error=NULL)`
- Turn warnings into errors
  - `options(warn=2)`
  - Turn off with `options(warn=0)`
- Test if a belief is true
  - `stopifnot()`

Think Like R

- An exciting opportunity to learn more about R!
- Build a mental model of the program (and R)
- Did I give this function all the information it needs to do what I ask?
- Does each function return what I expect?
- Be paranoid – trust no one!
- What are the consequences of my hypothesis?

What R you really doing?

The source will set you free

Read The Documentation

- Even if it's a function you've used before, there maybe parameters you've never needed before.
- The "Examples" section shows the function in action
- The "Value" section describes what the function returns
- Finding help pages
  - Within R: `?"functionname"`
  - On the web: http://www.rdocumentation.org/

Read The R Code

- Type a function name without parenthesis to see the code
  - `ggplot`
- Use a namespace with ":::" to access "unexported" functions
  - `ggplot:::ggplot.data.frame`
- Find function in any package
  - `getAnywhere("ggplot.data.frame")`

Read The C Code – Last Resort

- Primitive() and Internal() functions are implemented in C
- View R source code online (mirror of primary SVN repo)
  - https://github.com/wch/r-source
  - Google "R github source"
- Find C function name
  - Look in src/main/names.c
  - Usually R function "fun" maps to C function "do_fun"
- Find function
  - Search repo: `"do_fun extension:c"`

Step Through The R Code

- Debug a function
  - `debug() / undebug()`
  - `debugonce()`
  - `browser()` – inside the function
- Using the debugger (`?browser`)
  - n – continue to next line
  - s – step into function calls on current line
  - f – finish current loop or function
  - Q – exit the browser

What R you trying?

Answers begin with a minimal, complete, reproducible example

```r
library(lattice)
qqunif.plot<-function(pvalues,
        should.thin=T, thin.obs.places=2, thin.exp.places=2,
        xlab=expression(paste("Expected (",-log[10], " p-value)")),
        ylab=expression(paste("Observed (",-log[10], " p-value)")),
        draw.conf=TRUE, conf.points=1000, conf.col="lightgray", conf.alpha=.05,
        already.transformed=FALSE, pch=20, aspect="iso", prepanel=prepanel.qqunif,
        par.settings=list(superpose.symbol=list(pch=pch)), ...) {


        #error checking
        if (length(pvalues)==0) stop("pvalue vector is empty, can't draw plot")
        if(!(class(pvalues)=="numeric" ||
                (class(pvalues)=="list" && all(sapply(pvalues, class)=="numeric"))))
                stop("pvalue vector is not numeric, can't draw plot")
        if (any(is.na(unlist(pvalues)))) stop("pvalue vector contains NA values, can't draw plot")
        if (already.transformed==FALSE) {
                if (any(unlist(pvalues)==0)) stop("pvalue vector contains zeros, can't draw plot")
        } else {
                if (any(unlist(pvalues)<0)) stop("-log10 pvalue vector contains negative values, can't draw plot")
        }


        grp<-NULL
        n<-1
        exp.x<-c()
        if(is.list(pvalues)) {
                nn<-sapply(pvalues, length)
                rs<-cumsum(nn)
                re<-rs-nn+1
                n<-min(nn)
                if (!is.null(names(pvalues))) {
                        grp=factor(rep(names(pvalues), nn), levels=names(pvalues))
                        names(pvalues)<-NULL
                } else {
                        grp=factor(rep(1:length(pvalues), nn))
                }
                pvo<-pvalues
                pvalues<-numeric(sum(nn))
                exp.x<-numeric(sum(nn))
                for(i in 1:length(pvo)) {
                        if (!already.transformed) {
                                pvalues[rs[i]:re[i]] <- -log10(pvo[[i]])
                                exp.x[rs[i]:re[i]] <- -log10((rank(pvo[[i]], ties.method="first")-.5)/nn[i])
                        } else {
                                pvalues[rs[i]:re[i]] <- pvo[[i]]
                                exp.x[rs[i]:re[i]] <- -log10((nn[i]+1-rank(pvo[[i]], ties.method="first")-.5)/(nn[i]+1))
                        }
                }
        } else {
                n <- length(pvalues)+1
                if (!already.transformed) {
                        exp.x <- -log10((rank(pvalues, ties.method="first")-.5)/n)
                        pvalues <- -log10(pvalues)
                } else {
                        exp.x <- -log10((n-rank(pvalues, ties.method="first")-.5)/n)
                }
        }


        #this is a helper function to draw the confidence interval
        panel.qqconf<-function(n, conf.points=1000, conf.col="gray", conf.alpha=.05, ...) {
                require(grid)
                conf.points = min(conf.points, n-1);
                mpts<-matrix(nrow=conf.points*2, ncol=2)
                for(i in seq(from=1, to=conf.points)) {
                        mpts[i,1]<- -log10((i-.5)/n)
                        mpts[i,2]<- -log10(qbeta(1-conf.alpha/2, i, n-i))
                        mpts[conf.points*2+1-i,1]<- -log10((i-.5)/n)
                        mpts[conf.points*2+1-i,2]<- -log10(qbeta(conf.alpha/2, i, n-i))
                }
```

Why doesn't this code work?

Asking For Help

- Error message != error
- What is the expected behavior?
- What are the inputs to your code?
- Take away anything not directly related to the error
- Start a new script to isolate the problem
  - Start with code from examples on help pages
  - Start with other working examples

Stack Overflow

- Specific programming questions
  - Less about design or architecture
  - "How do I solve this particular problem"
  - **Provide a minimal, complete, reproducible example!**
  - Should be able to test and verify solutions
- Avoid questions about opinions
  - Bad: what's the "best" way to do something – define "best"
- Other sites exist for other questions
  - Statistics: http://stats.stackexchange.com/
  - Conceptual programming: http://programmers.stackexchange.com/
  - Code review: http://codereview.stackexchange.com/

Stack Overflow

- If you post a question, be prepared to respond to feedback quickly
- Write questions to help people with same problem in the future
  - Helping you personally is only a side effect, not the goal
- Good tags help to get to the right people (specific packages)
- Give explicit version numbers when you have questions about particular packages – `sessionInfo()`
- Pictures can help with plotting questions
- Users earn no points for niceness

[R]ecap

- Different classes result in different behaviors
- What you see isn't what you've got
- Good debugging uses the scientific method
- The source will set you free
- Answers begin with a minimal, complete, reproducible example

q()