getting staRted in R

Garrick Aden-Buie // Friday, March 25, 2016

INFORMS Code & Data Boot Camp

Today we'll talk about

- The R Universe
- Getting set up
- Working with data
- Base functions
- Where to go from here

Find these slides at
https://github.com/gadenbuie/usf-boot-camp-R

Here's what you need to start

- Install **R**
  - cloud.r-project.org
- Install **R-Studio**
  - rstudio.com
- Download the companion code to this talk
  - `http://bit.ly/1q5Rfpy`

The R Universe

What is R?

- R is an *Open Source* and free programming language for statistical computing and graphics, based on it predecessor S.
- Available for Windows, Mac, and Linux
- Under active development
- R can be easily extended with “packages”:
- code, data and documentation

Why use R?

- Free and open source
- Excellent and robust community
- One of the most popular tools for data analysis
- Growing popularity in science and hacking
  - Article in Fast Company
- Among the highest-paying IT skills on the market
  - 2014 Dice Tech Salary Survey
- So many cool projects and tools that make it easy to collaborate with others and publish your work

Pros of using R

- Available on any platform
- Source code is easy to read
- Lots of work being done in R now, with an excellent and open professional and academic community
- Plays nicely with many other packages (SPSS, SAS)
- Bleeding edge analyses not available in proprietary packages

Some downsides of R

- Older language that can be a little quirky
- User-~~driven~~ supplied features
- It's a programming language, not a point-and-click solution
- Slower than compiled languages
    - To speed up R you vectorize
    - Opposite of other languages

Some R Vocab

Term	Description
console, terminal	The "main" portal to R where you enter commands
scripts	Your "program" or text file containing commands
functions	Repeatable blocks of commands
working directory	Default location of files for input/output
packages	"Apps" for R
vector	The basic unit of data in R
dataframe	Data organized into rows and columns

http://adv-r.had.co.nz/Vocabulary.html

The R Console

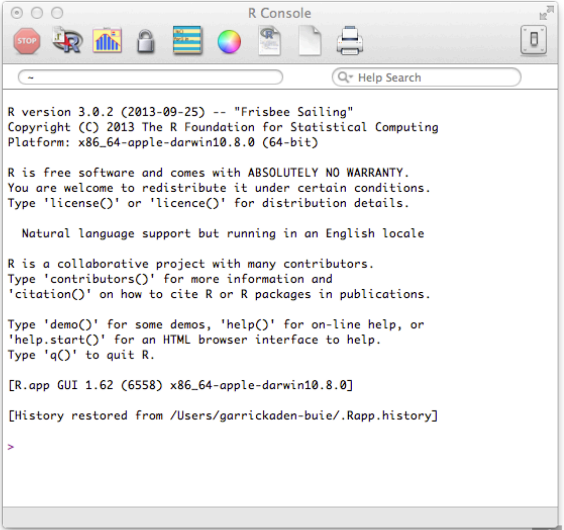


Figure 1:Standard R Console

R Studio: Standard View

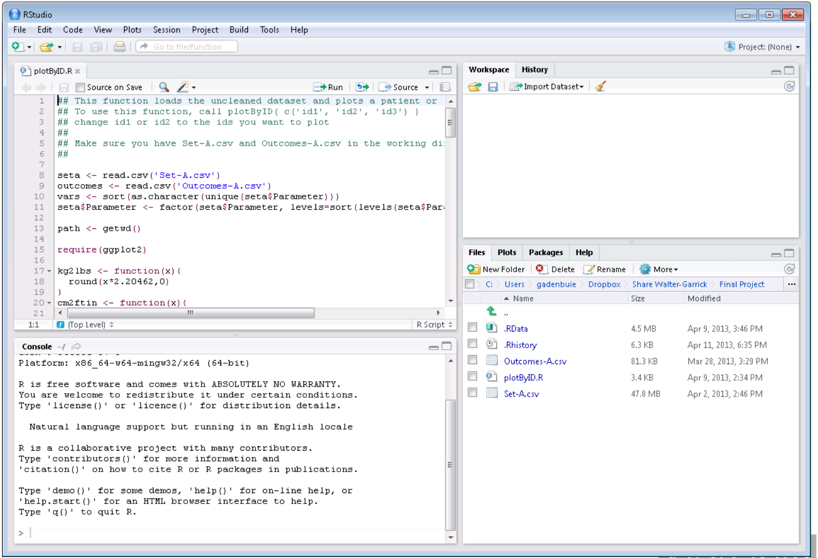


Figure 2

R Studio: My personalized view

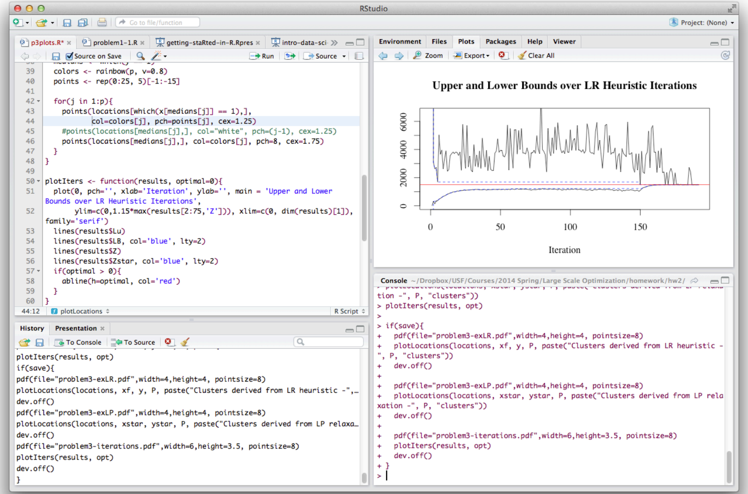


Figure 3

Take it for a quick spin

```r
3+3
## [1] 6
sqrt(4^4)
## [1] 16
2==2
## [1] TRUE
```

Setting up RStudio

- Under settings, move panes to where you want them to be
- Change font colors, etc
- Browse to downloaded companion script in **Files** pane
- Open script and set working directory

Where to get help

- Every R packages comes with documentation and examples
  - Try `?summary` and `??regression`
  - RStudio + tab completion = FTW!
- Get help online
  - StackExchange
  - Google (add `in R` or `R stats` to your query)
  - RSeek
- For really odd messages, copy and paste error message into Google

Working directory

Set working directory with

```
setwd("path/to/directory/")
```

Check to see where you are with

```
getwd()
```

Packages

Install packages[1]

```
install.packages('ggplot2')
```

Load packages

```
library(ggplot2)
```

Find packages on CRAN or Rdocumentation. Or

```
?ggplot
```

[1] Windows 7+ users need to run RStudio with System Administrator privileges.

Basics of the language

Basic Operators

```r
2 + 2
2/2
2*2
2^2
2 == 2
42 >= 2
2 <= 42
2 != 42
23 %/% 2    # Integer division -> 11
23 %% 2     # Remainder -> 1
```

Key Symbols

```
x <- 10           # Assigment operator
y <- 1:x          # Sequence
y[2]              # Element selection
## [1] 2
"str" == 'str'    # Strings
## [1] TRUE
```

Functions

Functions have the form `functionName(arg1, arg2, ...)` and arguments always go inside the parenthesis.

Define a function:

```
fun <- function(x=0){
  # Adds 42 to the input number
  return(x+42)
}
fun(8)
## [1] 50
```

Data types

```
1L          # integer
1.0         # numeric
'1'         # character
TRUE == 1   # logical
FALSE == 0  # logical
NA          # NA
factor()    # factor
```

You can check to see what type a variable is with `class(x)` or `is.numeric()`.

Data Structures

Vectors

Basic data type is a vector, built with `c()` for **concatenate**.

```
x <- c(1, 2, 3, 4, 5); x
## [1] 1 2 3 4 5
y <- c(6:10); y
## [1]  6  7  8  9 10
```

Working with vectors

```r
a <- sample(1:5, 10, replace=TRUE)
length(a)
## [1] 10
unique(a)
## [1] 4 5 3 1 2
length(unique(a))
## [1] 5
a * 2
##  [1]  8 10 10  6 10  2  2  4  2  2
```

Strings

Strings use either the ' ' or the " " characters.

```
mystr <- 'Glad you\'re here'
print(mystr)
## [1] "Glad you're here"
```

Use `paste()` to concatenate strings, not `c()`.

```
paste(mystr, '!', sep='')
## [1] "Glad you're here!"
c(mystr, '!')
## [1] "Glad you're here" "!"
```

Matrices: binding vectors

Matrices can be built by row binding or column binding vectors:

```
cbind(x,y)   # 5 x 2 matrix
##       x  y
## [1,] 1  6
## [2,] 2  7
## [3,] 3  8
## [4,] 4  9
## [5,] 5 10
rbind(x,y)   # 2 x 5 matrix
##   [,1] [,2] [,3] [,4] [,5]
## x    1    2    3    4    5
## y    6    7    8    9   10
```

Matrices: matrix function

Or you can build a matrix using the `matrix()` function:

```
matrix(1:10, nrow=2, ncol=5, byrow=TRUE)
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    2    3    4    5
## [2,]    6    7    8    9   10
```

Coercion

Vectors and matrices need to have elements of the same type, so R pushes mismatched elements to the best common type.

```
c('a', 2)
## [1] "a" "2"
c(1L, 1.0)
## [1] 1 1
c(1L, 1.1)
## [1] 1.0 1.1
```

Recycling

Recycling occurs when a vector has mismatched dimensions. R will fill in dimensions by *repeating* a vector from the beginning.

```
matrix(1:5, nrow=2, ncol=5, byrow=FALSE)
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    3    5    2    4
## [2,]    2    4    1    3    5
```

Factors

Factors are a special (at times frustrating) data type in R.

```
x <- rep(1:3, 2)
x
## [1] 1 2 3 1 2 3
x <- factor(x, levels=c(1, 2, 3),
            labels=c('Bad', 'Good', 'Best'))
x
## [1] Bad  Good Best Bad  Good Best
## Levels: Bad Good Best
```

Ordering factors

Order of factors is important for things like plot type, output, etc. Also factors are really two things tied together: the data itself and the labels.

```
x[order(x)]
## [1] Bad  Bad  Good Good Best Best
## Levels: Bad Good Best
x[order(x, decreasing=T)]
## [1] Best Best Good Good Bad  Bad
## Levels: Bad Good Best
```

Ordering factor labels

That reordered the elements of `x`, but not the factor levels.

Compare:

```
factor(x, levels=c('Best', 'Good', 'Bad'))
## [1] Bad  Good Best Bad  Good Best
## Levels: Best Good Bad
factor(x, labels=c('Best', 'Good', 'Bad'))
## [1] Best Good Bad  Best Good Bad
## Levels: Best Good Bad
```

Squashing factors

What if you want your drop the "factor" and keep the data?

Keep the numbers[2]

```
as.numeric(x)
## [1] 1 2 3 1 2 3
```

Keep the labels

```
as.character(x)
## [1] "Bad"  "Good" "Best" "Bad"  "Good" "Best"
```

[2] Risky, order matters!

Lists

Lists are arbitrary collections of objects. They don't have to be the same type or element or have the same dimensions.

```
mylist <-  list(vec = 1:5, str = "Strings!")
mylist
## $vec
## [1] 1 2 3 4 5
##
## $str
## [1] "Strings!"
```

Finding list elements

Use double brackets to return the list item or the $ operator.

```
mylist[[1]]
## [1] 1 2 3 4 5
mylist$str
## [1] "Strings!"
mylist$vec[2]
## [1] 2
```

Data frames

Data frames are like matrices, but better. Column vectors are *not* required to be the same type, so they can handle diverse data.

```
require(ggplot2)
data(diamonds, package='ggplot2')
head(diamonds)
```

carat	cut	color	clarity	depth	table	price	x	y	z
0.23	Ideal	E	SI2	61.5	55	326	3.95	3.98	2.43
0.21	Premium	E	SI1	59.8	61	326	3.89	3.84	2.31
0.23	Good	E	VS1	56.9	65	327	4.05	4.07	2.31
0.29	Premium	I	VS2	62.4	58	334	4.20	4.23	2.63
0.31	Good	J	SI2	63.3	58	335	4.34	4.35	2.75
0.24	Very Good	J	VVS2	62.8	57	336	3.94	3.96	2.48

Building a data frame

Data frames require vectors of the same dimension, but not the same type.

```
mydf <- data.frame(My.Numbers = sample(1:10, 6),
                   My.Factors = x)
mydf
##   My.Numbers My.Factors
## 1          3        Bad
## 2         10       Good
## 3          2       Best
## 4          6        Bad
## 5          9       Good
## 6          1       Best
```

Naming columns and rows

Data frames and matrices can have named rows and columns.

```
names(mydf)
## [1] "My.Numbers" "My.Factors"
```

```
colnames(mydf) <- c('Num', 'Fak')  # Set column names
rownames(mydf)                     # Same for rows
```

To find the dimensions of a matrix or data frame (*rows*, *cols*):

```
dim(mydf)
## [1] 6 2
```

Reading and writing data in data frames

R works well with Excel and CSV files, among many others. I usually work with CSV, but that's mostly personal preference.

Reading data

```
mydata <- read.csv('filename.csv', header=T)
```

Writing data

```
write.csv(mydata, 'filename.csv')
```

Control structures

if, else if, else

```
a <- 10
if(a > 11){
  print('Bigger!')
} else if(a < 9){
  print('Smaller!')
} else {
  print('On the money!')
}
## [1] "On the money!"
```

for loops

```r
z <- c()
for(i in 1:10){
  z <- c(z, i^2)
}
z
##  [1]   1   4   9  16  25  36  49  64  81 100
```

while loops

```r
z <- c()
i <- 1

while(i <= 5){
  z <- c(z, i^3)
  i <- i+1
}

z
## [1]   1   8  27  64 125
```

Manipulating data

`mtcars` data frame

R includes a number of datasets in the package `datasets` including `mtcars`. Try `?mtcars` to learn more. The data was extracted from the 1974 issue of *Motor Trend*.

If entering `mtcars` doesn't work, run `data(mtcars)` first.

```
head(mtcars)
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160	110	3.90	2.62	16.5	0	1	4	4
Mazda RX4 Wag	21.0	6	160	110	3.90	2.88	17.0	0	1	4	4
Datsun 710	22.8	4	108	93	3.85	2.32	18.6	1	1	4	1
Hornet 4 Drive	21.4	6	258	110	3.08	3.21	19.4	1	0	3	1
Hornet Sportabout	18.7	8	360	175	3.15	3.44	17.0	0	0	3	2
Valiant	18.1	6	225	105	2.76	3.46	20.2	1	0	3	1

Selecting rows and columns

Rows and columns are selected using brackets:

```
dataframe[<row conditions>, <column conditions>]
```

For example, `mtcars[1,2]` returns row 1, column 2:

```
mtcars[1,2]
## [1] 6
```

Select a whole row by leaving the column blank

```
mtcars[1,]
##           mpg cyl disp  hp drat   wt qsec vs am gear carb
## Mazda RX4  21   6  160 110  3.9 2.62 16.5  0  1    4    4
```

or similarly select a column by leaving the row condition blank

```
mtcars[,'qsec'][1:10]
##  [1] 16.5 17.0 18.6 19.4 17.0 20.2 15.8 20.0 22.9 18.3
```

More ways to select rows and columns

```
mtcars[-1,]                # Drop first row
mtcars[, -2:-4]            # Drop columns 2-4
mtcars[, c('mpg', 'cyl')] # Only mpg and cyl columns
mtcars[c(1,5,8,10),'am']
mtcars['Valiant',]         # Works when rows have names
mtcars$mpg                 # Select 'mpg' col
mtcars[[1]]                # Same
mtcars[['mpg']]            # Also the same
mtcars$mpg[1:5]            # == mtcars[1:5, 'mpg']
```

Subsetting

What if you want to look at the gas guzzlers only?

```
gas_guzzlers <- mtcars[mtcars$mpg < 20,]
head(gas_guzzlers)
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Hornet Sportabout	18.7	8	360	175	3.15	3.44	17.0	0	0	3	2
Valiant	18.1	6	225	105	2.76	3.46	20.2	1	0	3	1
Duster 360	14.3	8	360	245	3.21	3.57	15.8	0	0	3	4
Merc 280	19.2	6	168	123	3.92	3.44	18.3	1	0	4	4
Merc 280C	17.8	6	168	123	3.92	3.44	18.9	1	0	4	4
Merc 450SE	16.4	8	276	180	3.07	4.07	17.4	0	0	3	3

Subsetting

Or 6-cylinder gas guzzlers only...

```
gas_guzzlers <- mtcars[mtcars$mpg < 20 & mtcars$cyl == 6,]
head(gas_guzzlers)
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Valiant	18.1	6	225	105	2.76	3.46	20.2	1	0	3	1
Merc 280	19.2	6	168	123	3.92	3.44	18.3	1	0	4	4
Merc 280C	17.8	6	168	123	3.92	3.44	18.9	1	0	4	4
Ferrari Dino	19.7	6	145	175	3.62	2.77	15.5	0	1	5	6

Setting values based on subsets

Create a new column for speed class based on quarter mile time.

```
mtcars[mtcars$qsec < 17, 'Class'] <- 'Slow'
mtcars[mtcars$qsec > 17, 'Class'] <- 'Medium'
mtcars[mtcars$qsec > 20, 'Class'] <- 'Fast'
table(mtcars$Class)
##
##    Fast Medium   Slow
##       3     20      9
```

Any expression that evaluates to `TRUE` or `FALSE` can be used as a column or row condition.

```
mtcars$qsec[1:10] > 17
##  [1] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE
```

Dealing with missing values

Missing values show up as NAs, which is actually a data type.

```
foo <- c(1.2, NA, 2.4, 6.2, 8.3)
bar <- c(9.1, 7.6, NA, 1.1, 4.7)
fb <- cbind(foo, bar)
fb[complete.cases(fb),]
##      foo bar
## [1,] 1.2 9.1
## [2,] 6.2 1.1
## [3,] 8.3 4.7
foo[!is.na(foo)]
## [1] 1.2 2.4 6.2 8.3
```

Base functions

Summarize just about anything

```
summary(mtcars[,1:3])
##       mpg             cyl             disp      
##  Min.   :10.4   Min.   :4.00   Min.   : 71  
##  1st Qu.:15.4   1st Qu.:4.00   1st Qu.:121  
##  Median :19.2   Median :6.00   Median :196  
##  Mean   :20.1   Mean   :6.19   Mean   :231  
##  3rd Qu.:22.8   3rd Qu.:8.00   3rd Qu.:326  
##  Max.   :33.9   Max.   :8.00   Max.   :472  
```

All around great functions: str

"Quick look" function

```
str(mtcars)
## 'data.frame':    32 obs. of  12 variables:
##  $ mpg  : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
##  $ cyl  : num  6 6 4 6 8 6 8 4 4 6 ...
##  $ disp : num  160 160 108 258 360 ...
##  $ hp   : num  110 110 93 110 175 105 245 62 95 123 ...
##  $ drat : num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
##  $ wt   : num  2.62 2.88 2.32 3.21 3.44 ...
##  $ qsec : num  16.5 17 18.6 19.4 17 ...
##  $ vs   : num  0 0 1 1 0 1 0 1 1 1 ...
##  $ am   : num  1 1 1 0 0 0 0 0 0 0 ...
##  $ gear : num  4 4 4 3 3 3 3 4 4 4 ...
##  $ carb : num  4 4 1 1 2 1 4 2 2 4 ...
##  $ Class: chr  "Slow" "Medium" "Medium" "Medium" ...
```

All around great functions: attributes

Learn more about the object

```
attributes(mtcars[1:10,])
## $names
##  [1] "mpg"   "cyl"   "disp"  "hp"    "drat"  "wt"    "qsec"  "vs"    "am"
## [10] "gear"  "carb"  "Class"
##
## $row.names
##  [1] "Mazda RX4"         "Mazda RX4 Wag"     "Datsun 710"
##  [4] "Hornet 4 Drive"    "Hornet Sportabout" "Valiant"
##  [7] "Duster 360"        "Merc 240D"         "Merc 230"
## [10] "Merc 280"
##
## $class
## [1] "data.frame"
```

All around great functions: table

Quick and dirty tables

```
table(mtcars$cyl, mtcars$gear)
##
##      3  4  5
##   4  1  8  2
##   6  2  4  1
##   8 12  0  2
```

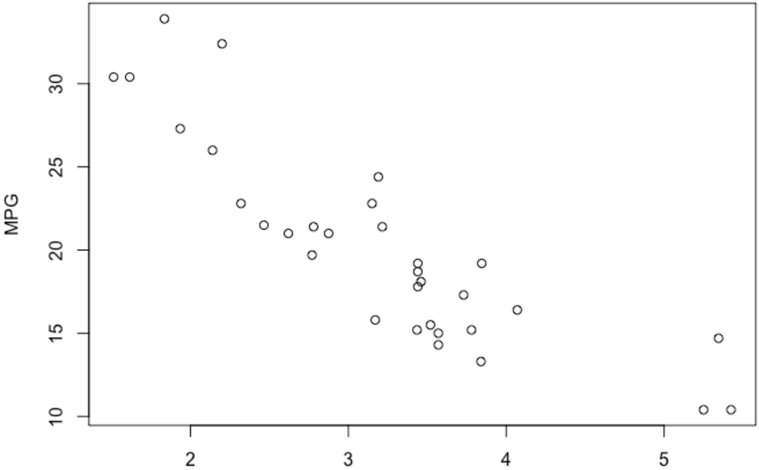
Basic functions for vectors

```
sum()
mean()
sd()     # standard deviation
max()
min()
median()
range()
rev()    # reverse
unique() # unique elements
length()
```

Visualizing data

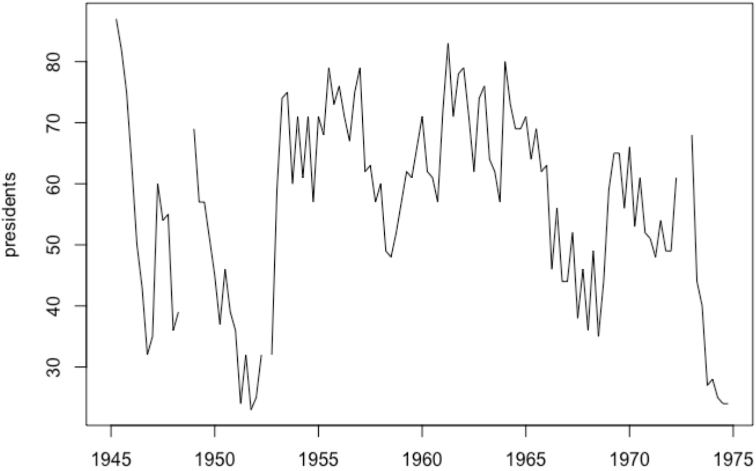
Plotting points

```
plot(mtcars$wt, mtcars$mpg,
     xlab='Weight', ylab='MPG')
```



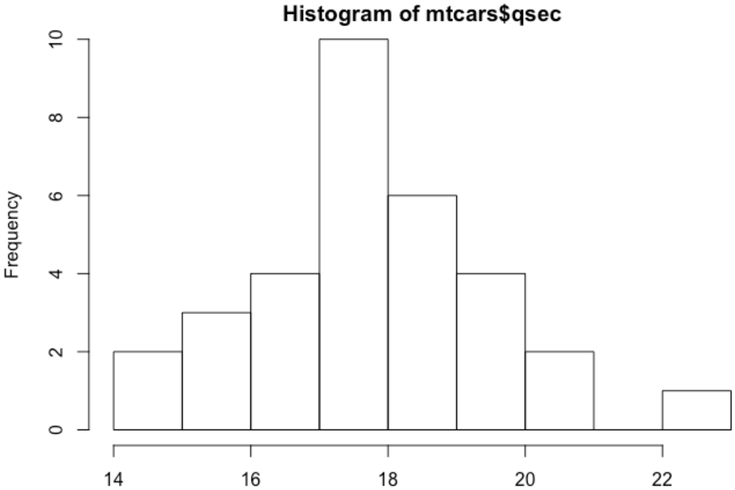
Plotting lines

```
plot(presidents, type='l',
     xlab = 'Approval Rating')
```



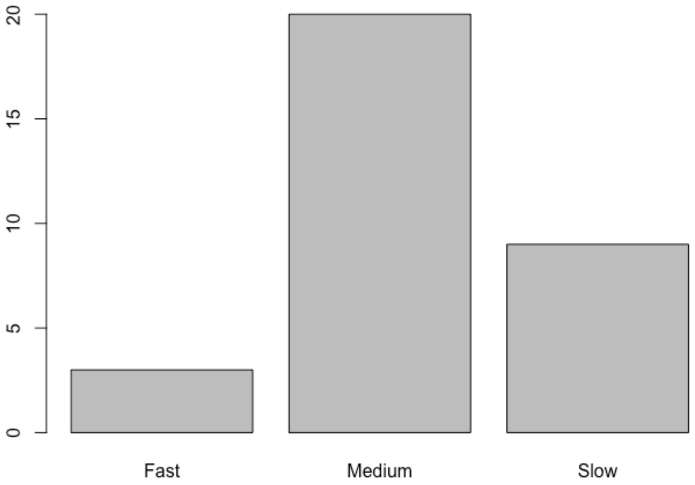
Histograms

```
par(mar=c(5,4,1,1), bg='white')
hist(mtcars$qsec, xlab='Quarter Mile Time')
```



Bar plots

```
barplot(table(mtcars$Class))
```



Base stats information

r*, p*, q*, d* functions

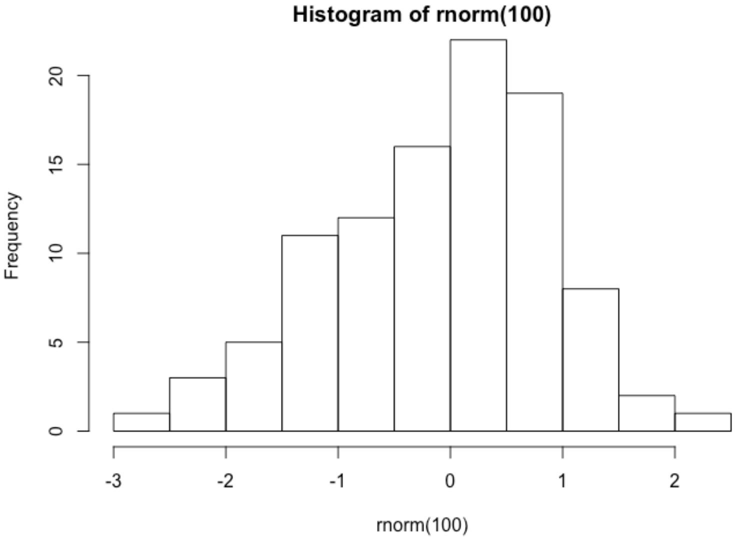
For all of the statistical distributions, R uses the following naming conventions (incredible how useful this is!):

- `d*` = density/mass function
- `p*` = cumulative distribution function
- `q*` = quantile function
- `r*` = random variate generation

There are quite a few distributions available in base R packages. Just run `?Distributions` to see a full list.

rnorm() example

```
hist(rnorm(100))
```



Better than base packages

- Manipulating data
  - ddply and plyr and now dplyr
- Visualizing data
  - ggplot2
- Reporting data
  - knitr
- Interactive online R sessions
  - shiny

Go ExploR

Resources for learning more

- Advanced R Programming
  - By one of the best and most important R developers.
- TwoTorials
  - Quick two minute videos on doing things in R.
- An R Meta Book
  - A collection of online books.
- R Bloggers
  - A mailing list and central hub of all things online regarding R.

Thanks!