Getting Started With R

P. Johnson, B. Rogers, and L. Shaw

Wednesday, January 28, 2015

Getting comfortable with R

Today's lab will attempt to introduce you to R, teach you some basics about how R is set up, how to create some data, how to run some basic functions, and how to look for help.

If you do not find what you need through R help commands, search the internet. There are many examples, and if you do not understand one example, keep searching. Someone will write about that function in a way that will make sense to you.

More than just a calculator

- Interacting with R
- Using R as a calculator
- Assigning information to a variable
- Getting help

Working with data

- Using R to create a vector
- Descriptive statistics
- Vector arithmetic
- Referencing specific data
- Plots
- Your workspace
- Packages and **rockchalk**

Interacting with R

- Enter a command in the console, or
- Use one of the editors and send to the console
- Liberally comment your code and save
- Find code that works and modify for your own use

The R console

- Enter code below in a console
- Enter code in editor and send to R console

```
# Comment to describe code
# What summary provides depends on what is in the parentheses
# cars is a built in data set so you get summary statistics
summary(cars)
```

```
##      speed           dist       
##  Min.   : 4.0   Min.   :  2.00  
##  1st Qu.:12.0   1st Qu.: 26.00  
##  Median :15.0   Median : 36.00  
##  Mean   :15.4   Mean   : 42.98  
##  3rd Qu.:19.0   3rd Qu.: 56.00  
##  Max.   :25.0   Max.   :120.00  
```

Basic arithmetic

```
3 + 2  # Addition
```

```
## [1] 5
```

```
3 - 1  # Subtraction
```

```
## [1] 2
```

```
7 * 8  # Multiplication
```

```
## [1] 56
```

```
9 / 3  # Division
```

```
## [1] 3
```

More advanced calculations

```
3 ^ 2  # Power
```

```
## [1] 9
```

```
sqrt(36) # Square root
```

```
## [1] 6
```

```
log(7) # Natural logarithm ln
```

```
## [1] 1.94591
```

Using R as a calculator

```
exp(10) # e^10
```

```
## [1] 22026.47
```

- Use parentheses around functions to force order of operations

```
1 + 3 * log(10) - (exp(7) - 3) / 4
```

```
## [1] -265.5005
```

Assigning information to a variable

- You can store single values, vectors, matrices, data frames, arrays, and other objects
- The matrix object can only hold numbers
- Numbers or strings can be stored in the other object types
- Good form uses **<-** to assign something to a variable
- X and x are two different variables because R is case-sensitive

Assigning numbers to a variable

```
x <- 1
x   # The value can be printed by typing the object name
```

```
## [1] 1
```

```
x <- 2 + 5  # The result is saved in the object 'x'
x
```

```
## [1] 7
```

Assigning text to a variable

```
z <- "Hello!"  # We may save an object as text.
z
```

```
## [1] "Hello!"
```

Assigning numbers to X and x

```
X <- 5
X + x
```

```
## [1] 12
```

```
X <- log(2) # This will overwrite the previous value.
X + x
```

```
## [1] 7.693147
```

Getting help

```
## When R was installed, HTML format help files were copied
## onto your hard drive. To access these files, just type
help.start()
## To request an R document for a specific function, use '?'.
?log
## To request help by keywords, use '??'.
??logarithm
```

Store data in a vector

- Many values can be stored together in vectors (c is combine)

```
midterm <- c(99, 87, 96, 100, 82, 79, 88, 85, 94, 90)
midterm
```

```
##  [1]  99  87  96 100  82  79  88  85  94  90
```

```
f <- 1:20
f
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

```
g <- seq(from = 1, to = 20) # seq(start value, end value, interval)
h <- seq(1, 20, by = 0.2) # specify interval
```

More functions to create vectors

- rep stands for replicate

```
k <- rep(1, 5)  # rep(value, number of times)
k
```

```
## [1] 1 1 1 1 1
```

```
l <- rep(1:10, 5)  # rep() can accept numbers, text, or NA
l
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10  1  2  3  4  5  6  7  8  9 10  1  2  3
## [24]  4  5  6  7  8  9 10  1  2  3  4  5  6  7  8  9 10  1  2  3  4  5  6
## [47]  7  8  9 10
```

Generating random values

- What is required in the function depends on the type of distribution

```
datn <- rnorm(n = 1000, mean = 24, sd = 3)
# Look at datn to see what R generated

datp <- rpois(1000, 3)     # 1000 random numbers
## from a poisson disribution
## with an expected value of 3.
```

Descriptive statistics

- Re-create midterm variable
- Compute the mean

```
midterm <- c(99, 87, 96, 100, 82, 79, 88, 85, 94, 90)
mean(midterm)
```

```
## [1] 90
```

- Run one of the following functions
- median, var, sd, min, max, range, length, table, sum, summary

What if you are missing data?

```
midterm2 <- c(99, NA, 96, 100, 82, NA, 88, 85, 94, 90)
mean(midterm2)
```

```
## [1] NA
```

- Tell the function to compute even though there is missing data

```
mean(midterm2, na.rm = TRUE)
```

```
## [1] 91.75
```

Simple vector arithmetic

- R can save you time if you understand how whole vectors can be processed

```
c(1, 2, 3, 4) - 4
```

```
## [1] -3 -2 -1  0
```

```
c(1, 2, 3, 4)/4
```

```
## [1] 0.25 0.50 0.75 1.00
```

```
c(1, 2, 3, 4)/c(4, 3, 2, 1)   # 1/4, 2/3, 3/2, 4/1
```

```
## [1] 0.2500000 0.6666667 1.5000000 4.0000000
```

Vector arithmetic with functions

- R can save you time if you understand how whole vectors can be processed

```
log(c(1, 2, 3, 4))  # applies log to each element
```

```
## [1] 0.0000000 0.6931472 1.0986123 1.3862944
```

```
## Can you tell what this is?
sum((midterm - mean(midterm))^2)/(length(midterm) - 1)
```

```
## [1] 50.66667
```

Referencing specific data in vector

- You can use individual values in a vector or other object

```
midterm
```

```
##  [1]  99  87  96 100  82  79  88  85  94  90
```

```
midterm[2]
```

```
## [1] 87
```

```
midterm[length(midterm)] # What is happening here?
```

```
## [1] 90
```

Reference more than one thing

```
midterm[1:4] # take more than one element at a time.
```

```
## [1]  99  87  96 100
```

```
midterm[c(1:4, 8, 10)]
```

```
## [1]  99  87  96 100  85  90
```

```
midterm[1, 2, 3, 4, 8, 10]  # This is NOT THE SAME
## R could only evaluate this if midterm was an array with
## 6 dimensions.
```

Histogram help

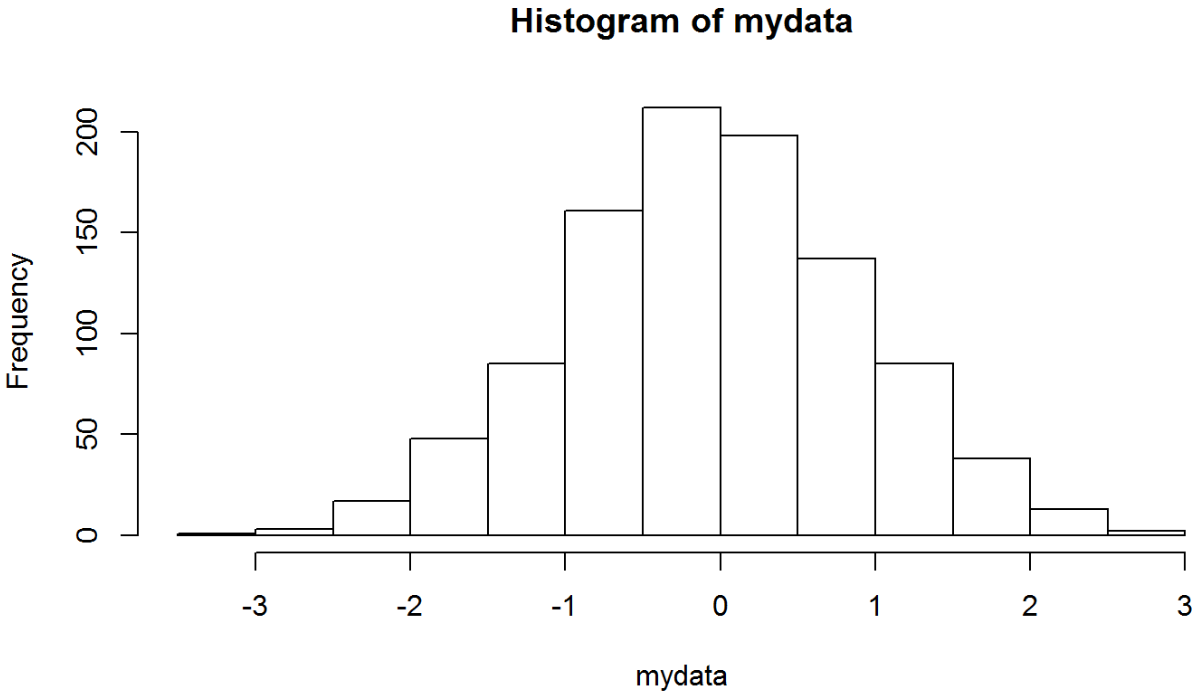
```
mydata <- rnorm(1000, 0, 1)
```

```
?hist
```

```
## starting httpd help server ... done
```

Histogram with frequencies

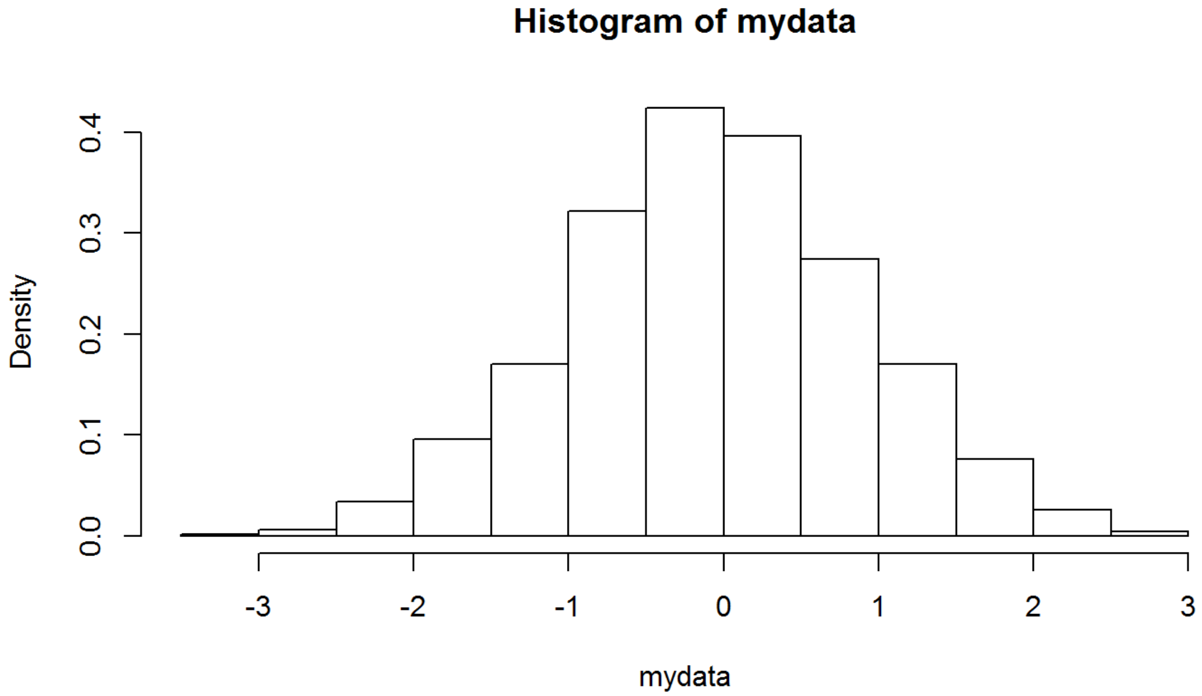
```
hist(mydata)
```



Histogram with proportions

- Values on y-axis are sometimes referred to as the following: proportion, probability, and density

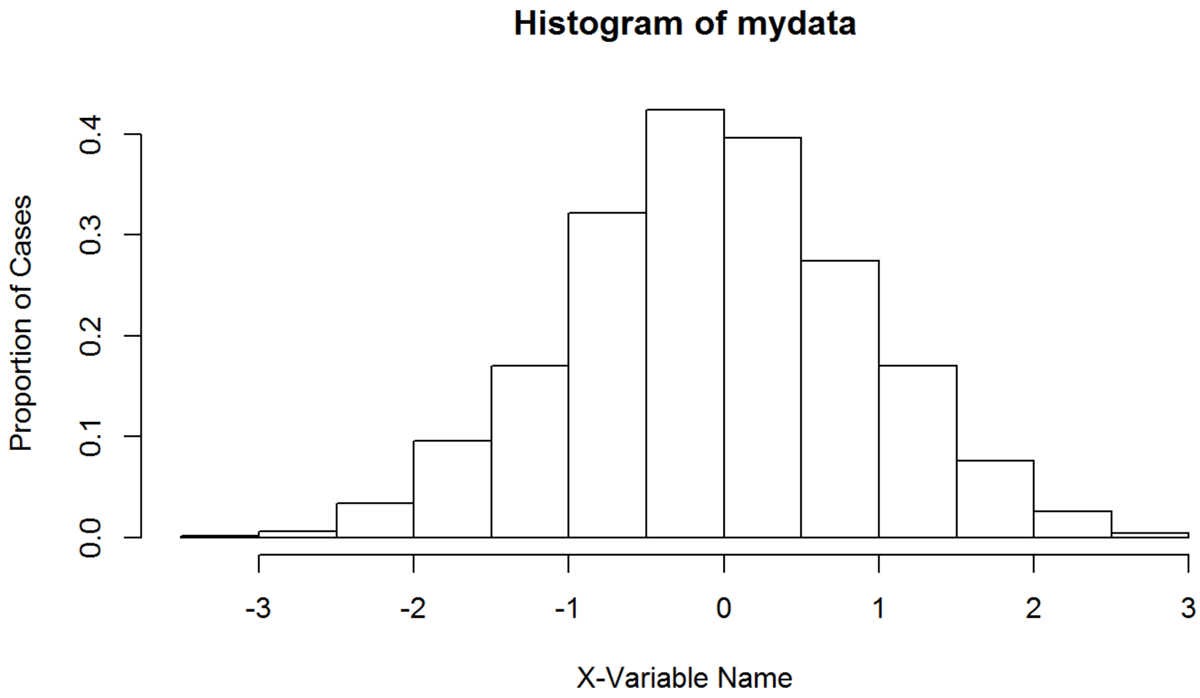
```
hist(mydata, prob = TRUE)
```



26/37

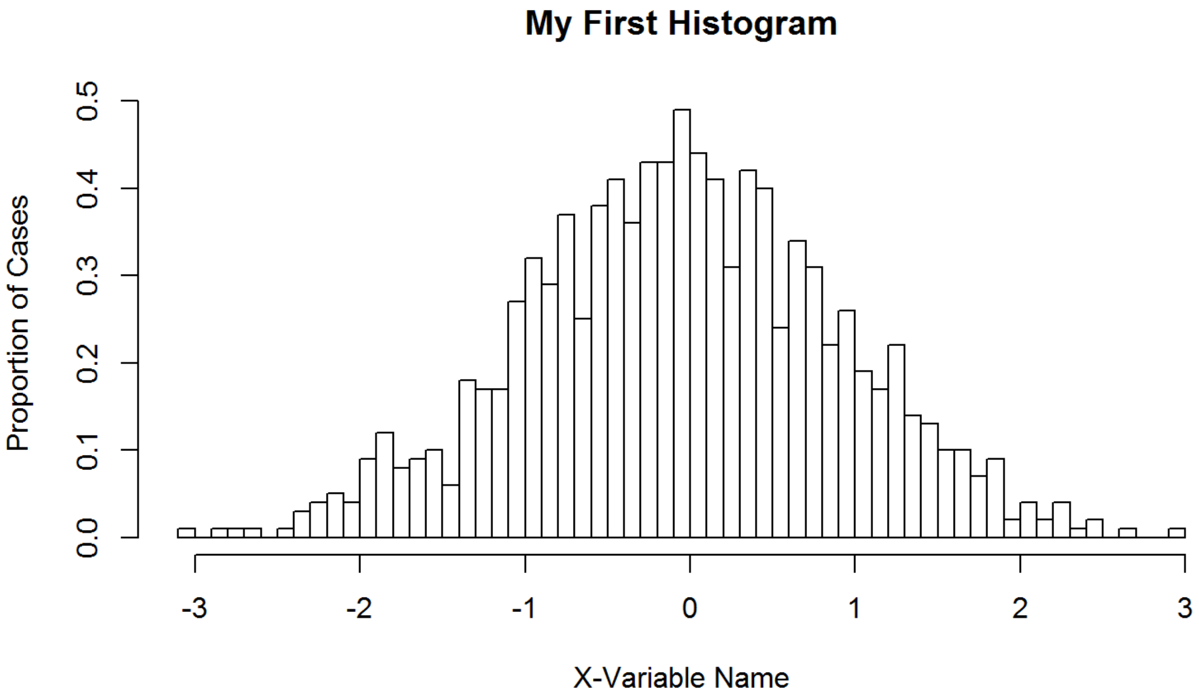
Histogram with labels

```
hist(mydata, prob=TRUE, xlab="X-Variable Name",
     ylab="Proportion of Cases") # Label x and y axes
```



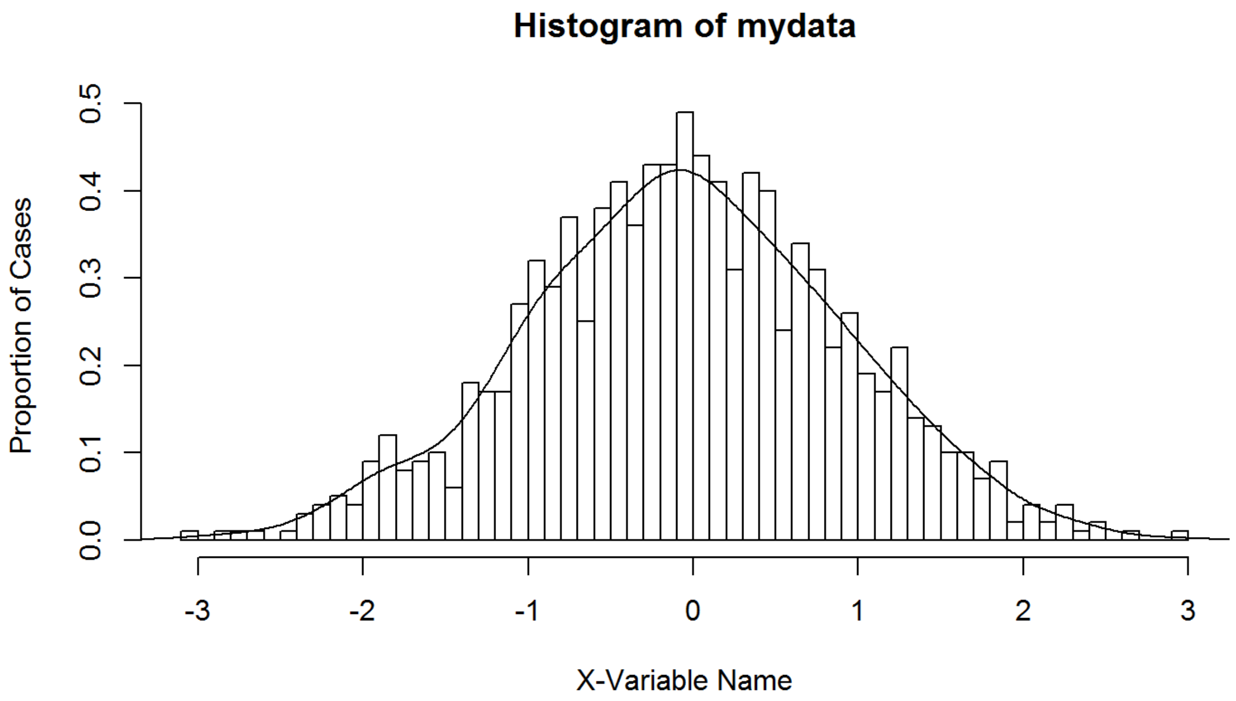
Add a title and number of bins

```
hist(mydata, prob=TRUE, xlab="X-Variable Name",
     ylab="Proportion of Cases", main="My First Histogram", breaks=50)
```



Add a density line

```
hist(mydata, prob=TRUE, xlab="X-Variable Name",
     ylab="Proportion of Cases", breaks=50)
lines(density(mydata))
```



Your working directory

- When reading data or saving objects to your computer, you need to know where R is looking

```
getwd()
```

```
## [1] "D:/Users/l076s857/Dropbox/GTA706"
```

- To change where R is looking, specify relative or full path

```
setwd("Labs")
setwd("D:\\Users\\l076s857\\Desktop") # For Windows
setwd("D:/Users/l076s857/Desktop") # For Windows, Mac, Linux
```

Your workspace

- List all items

```
ls()
```

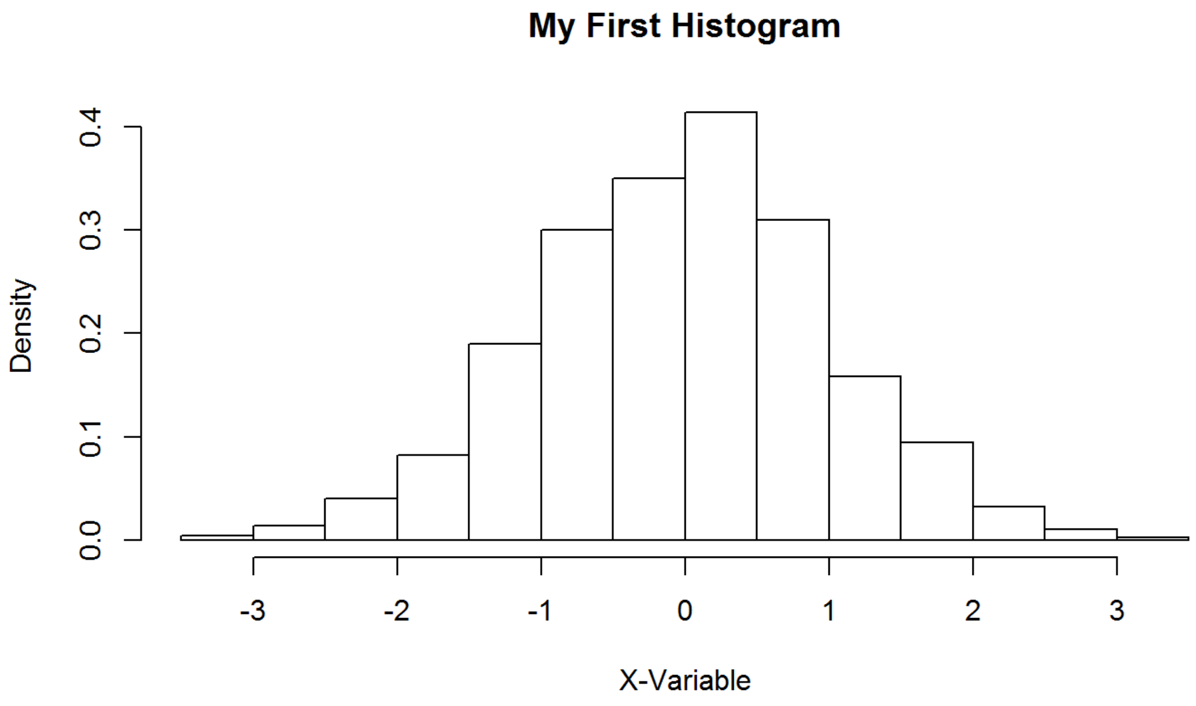
```
##  [1] "datn"     "datp"     "f"        "g"        "h"        "h1"
##  [7] "k"        "l"        "midterm"  "midterm2" "mydata"   "x"
## [13] "X"        "z"
```

- Remove one item, then all

```
rm(x)
rm(list = ls(all = TRUE))
```

Information stored in objects

```
mydata <- rnorm(1000, 0, 1)
h1 <- hist(mydata, prob=TRUE, xlab="X-Variable", ylab="Density",
           main="My First Histogram", breaks=10)
```



Information stored in histograms

```
h1
```

```
## $breaks
##  [1] -3.5 -3.0 -2.5 -2.0 -1.5 -1.0 -0.5  0.0  0.5  1.0  1.5  2.0  2.5  3.0
## [15]  3.5
##
## $counts
##  [1]   2   7  20  41  95 150 175 207 155  79  47  16   5   1
##
## $density
##  [1] 0.004 0.014 0.040 0.082 0.190 0.300 0.350 0.414 0.310 0.158 0.094
## [12] 0.032 0.010 0.002
##
## $mids
##  [1] -3.25 -2.75 -2.25 -1.75 -1.25 -0.75 -0.25  0.25  0.75  1.25  1.75
## [12]  2.25  2.75  3.25
##
## $xname
## [1] "mydata"
##
## $equidist
## [1] TRUE
##
```

Information stored in objects

```
attributes(h1)
```

```
## $names
## [1] "breaks"   "counts"   "density"  "mids"     "xname"    "equidist"
## 
## $class
## [1] "histogram"
```

```
names(h1)
```

```
## [1] "breaks"   "counts"   "density"  "mids"     "xname"    "equidist"
```

```
h1$mids   # The $ allows you to access a specific attribute of the object
```

```
##  [1] -3.25 -2.75 -2.25 -1.75 -1.25 -0.75 -0.25  0.25  0.75  1.25  1.75
## [12]  2.25  2.75  3.25
```

Working with packages

- Some functions are built into R, like all of the functions that we have used so far.
- There are many functions in other packages and a list of packages can be found at http://cran.r-project.org/web/packages/available_packages_by_name.html
- The first time you want to use a function in a new package, you need to install it.
- Once the package is installed, you will need to reference the package in your R code before trying to use that function.

Install and use package

- If you do not have admin rights, a local directory will be created by R for the packages you install.

```
install.packages("rockchalk", dep = TRUE)
```

```
library(rockchalk)
```

```
## Warning: package 'rockchalk' was built under R version 3.0.3
```

Use function from rockchalk

```r
mydata <- rnorm(1000, 0, 1)
summarize(mydata)
```

```
## $numerics
##         mydata
## 0%     -3.1151
## 25%    -0.6887
## 50%     0.0276
## 75%     0.6963
## 100%    3.8275
## mean    0.0102
## sd      1.0007
## var     1.0014
## NA's    0.0000
## N    1000.0000
##
## $factors
## NULL
```