R for the Beginner

A Short Introduction to R

For the R Beginner

Mike Montenegro
February 23, 2019

Presentation reflects one R student's perspective (mine).

This is NOT a detailed tutorial;
we will NOT get bogged down with details.

“High-altitude” overview; I will NOT read every slide.

Selected syntax examples used to illustrate key points

Some useful reference sources & links are provided.

The goal is for you to have FUN while learning some
KEY points about R through EXAMPLES

The goal is to give the R beginner a running start

One R student's perspective (mine)

- I am a student of R. Not a nubi, but not advanced.

- I am a dabbler. Not a software developer.

- Elegant code is nice, but I just want it to work
  (it doesn't have to look pretty or run at blinding speed.)

- R is worth learning if it can do what I want; it is the means to an end.
  R helps me learn about and apply machine learning algorithms.
  R helps me access and analyze stock market data (quantmod).

- I like to learn first from examples and read the details later.

- **I have too many slides for the limited time available.**
  For the "high-altitude" overview, I will NOT read every slide.
  I'd like to spend at least half the time on examples.
  You can then use the presentation slides as reference material.

- If you have R installed, fire it up and run the demos with me.

What is R?

R is a free software environment for statistical computing and graphics.

It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS.

I run R in an Ubuntu terminal.

To download R, go to

http://www.r-project.org/

… and please choose your preferred CRAN mirror.

R is a GNU project which is similar to the S language and environment which was developed at Bell Laboratories (formerly AT&T, now Lucent Technologies) by John Chambers and colleagues.

R can be considered as a different implementation of S. There are some important differences, but much code written for S runs unaltered under R.

One interesting tid-bit: John Chambers owns the original CD-ROM (serial number #1) of R 1.0.0, released on February 29 2000, and signed by all the members of R-core.

http://blog.revolutionanalytics.com/2014/01/john-chambers-recounts-the-history-of-s-and-r.html

https://www.linuxjournal.com/content/open-science-open-source-and-r

The R environment

R is an integrated suite of software facilities for
data manipulation, calculation and graphical display.

It includes an effective data handling and storage facility,

- a suite of operators for calculations on arrays, in particular matrices (and data frames),

- a large, coherent, integrated collection of intermediate tools for data analysis, graphical facilities for data analysis and display either on-screen or on hardcopy, and

- a well-developed, simple and effective programming language which includes conditionals, loops, user-defined recursive functions and input and output facilities.

CRAN Mirrors - Where to go to download R and R packages

The **Comprehensive R Archive Network (CRAN)** is a network of ftp and web servers around the world that store identical, up-to-date versions of code and documentation for R.

Please use the CRAN mirror near you to minimize network load.

For example:

http://watson.nci.nih.gov/cran_mirror/

National Cancer Institute, Bethesda, MD

RStudio

RStudio is a powerful user interface (GUI) for R. It's free, open source, and works on Windows, Mac, and Linux.

RStudio, an integrated development environment (IDE), is a software application that provides comprehensive facilities to computer programmers for software development.

See "implementing RStudio and R on Ubuntu" (9 minute video) at http://www.youtube.com/watch?v=P8wx4HY9me0

RStudio Server

RStudio Server enables you to provide a browser based interface to a version of R running on a remote Linux server, bringing the power and productivity of the RStudio IDE to server-based deployments of R.

http://www.rstudio.com/

There's much more to R than initially meets the eye.

- R has its own jargon (some is not initially intuitive)
  e.g., object, class, vector, list, source, packages

- R is about objects, classes and methods

- R comes with its own built-in data sets
  e.g., cars, iris, ChickWeight

- You can install and/or build your own data sets
  e.g., Lahman (baseball), sp500 (stock market)

- R comes with its own built-in 'standard' functions. Examples:
  print, min, max, median, floor, ceiling, sqrt, summary, str, fix, head, tail, class, plot, log, exp, abs, sin, tanh, round, signif, c, cbind, data, require, library, install.packages, lm, rnorm, runif, seq, rep, names, ncol, nrow, rownames, subset, lowess, paste

http://www.statmethods.net/management/functions.html

```
data(ChickWeight)    # Loads the ChickWeight data

head(ChickWeight)   # First six rows

##Grouped Data: weight ~ Time | Chick
##  weight Time Chick Diet
##1     42    0     1    1
##2     51    2     1    1
##3     59    4     1    1
##4     64    6     1    1
##5     76    8     1    1
##6     93   10     1    1

# See what you get

str(ChickWeight)      # str displays the Structure of an R object

# ?str()                # use ? for help

# help(str)             # scroll down to Examples
```

- R has 'data structures'
  e.g., atomic vector, list, matrix, array, data frame

- You can build your own functions

- R has the standard control structures you would expect

- R comes with its own built-in 'standard' packages

- R users can release their own user-built packages
  Some user-built packages expand the scope of R

- You can install user-built packages
  e.g., nnet, quantmod, MASS, ggplot2, Rcpp
  (There are many, many more.)

- You can build (and source) your own R scripts
  e.g., source('~/Dropbox/R_Pres_Demo_01.R', echo=T)

Three things to remember about R:

1 - In R everything is an **object**

The most important concepts for understanding R's structure are objects, classes and methods.

http://www.academia.edu/1972132/Object_Class_and_Method_in_R
http://adv-r.had.co.nz/Data-structures.html for info on data structures

2 - If you have a question about R, start with a "?"

?(cars) is the same as help(cars)

Examples are usually provided at the bottom.

You can also use "??"

3 - You can ask Google for more help (I add "CRAN")

e.g., "R data frame list object"
"R commands on one line"
"R data types structures"
"how to change r prompt in linux"

Conclusion (claim from an article previously)

“The concepts of objects, classes, methods, and generic functions play important roles in the success of the R language. **Many users of R can write their code and get results from it without knowing about objects, classes, methods,** and generic functions, but sometimes they do not understand how the same command can apply to different objects and yield different results. **But after learning about these concepts, every command seems very intuitive to them**. This is the difference that one could cover after reading this article, that will distinguish you from ordinary R users and make you an expert in R.”

Classes

R has an elaborate class system, principally controlled via the class attribute. This attribute is a character vector containing the list of classes that an object inherits from. This forms the basis of the “generic methods” functionality in R.

This attribute can be accessed and manipulated virtually without restriction by users. There is no checking that an object actually contains the components that class methods expect. Thus, altering the class attribute should be done with caution, and when they are available specific creation and coercion functions should be preferred.

Each object in R has a class which can be determined by the class function

```
> class(1)
[1] “numeric”
> class(TRUE)   # same as class(T)
[1] “logical”
> class( rnorm(100) )     # rnorm(n, mean = 0, sd = 1)
[1] “numeric”
> class(NA)
[1] “logical”
> class(“foo”)
[1] “character”
> class(sp500)
[1] "xts" "zoo"
```

xts: Function for creating an extensible time-series object. ‘xts’ is used to create an ‘xts’ object from raw data inputs. ‘zoo’ is the creator for an S3 class of indexed totally ordered observations which includes irregular time series.

R's base data structures are summarized in the table below, organized by their dimensionality and whether they're homogeneous (all contents must be of the same type) or heterogeneous (the contents can be of different types):

	Homogeneous	Heterogeneous
1d	Atomic vector	List
2d	Matrix	Data frame
Nd	Array	

Note that R has no scalar, or 0-dimensional, types.
All scalars (single numbers or strings) are length-one vectors.

Almost all other objects in R are built upon these foundations, and in the OO field guide you'll see how R's object oriented tools build on top of these basics.

When trying to understand the structure of an arbitrary object in R your most important tool is str(), short for structure: it gives a compact human readable description of any R data structure.

See http://adv-r.had.co.nz/Data-structures.html for more details.

Basic Operations

R can be used as an ordinary calculator. A few examples:

```
2 + 3 * 5      # 17 -- Note the order of operations
log (10)       # 2.302585 -- Natural logarithm with base e=2.718282
log10 (2)      # 0.30103 -- Logarithm with base 10
4^2            # 16 -- 4 raised to the second power

3/2            # 1.5 -- Division
sqrt (16)      # 4 -- Square root
abs (3-7)      # 4 -- Absolute value of 3 minus 7

pi             # 3.141593 -- The mysterious number
exp(2)         # 7.389056 -- exponential function
15 %/% 4       # 3 -- This is the integer divide operation

# This is a comment line
```

The assignment operator (<-) stores the value on the right side of (<-) expression, on the left side. Once assigned, the object can be used just as an ordinary component of the computation. To find out what the object looks like, simply type its name. Note that R is case sensitive. Object names abc, ABC, Abc are all different.

```
 x <- log(2.843432) * pi
 x
##[1] 3.283001

 sqrt(x)
##[1] 1.811905

 floor(x)        # largest integer less than or equal to x
##[1] 3

 ceiling(x)      # smallest integer greater than or equal to x
##[1] 4
```

R can also handle complex numbers.

```
 x <- 3+2i
 Re(x)          # Real part of the complex number x
##[1] 3

 Im(x)          # Imaginary part of x (use capital I)
##[1] 2

 y <- -1+1i
 x+y
##[1] 2+3i

 x*y
##[1] -5+1i
```

Vectors and assignment

R operates on named data structures.
The simplest such structure is the numeric vector,
which is a single entity consisting of an ordered collection of numbers.

To set up a vector named x, consisting of five numbers,
namely 10.4, 5.6, 3.1, 6.4 and 21.7, use the R command

```
> x <- c(10.4, 5.6, 3.1, 6.4, 21.7)
```

This is an assignment statement using the function c() which in this context can take an arbitrary number of vector arguments and whose value is a vector got by concatenating its arguments end to end.

A number occurring by itself in an expression is taken as a vector of length one.

Notice that the assignment operator (‘<-’), which consists of the two characters ‘<’ (“less than”) and ‘-’ (“minus”) occurring strictly side-by-side and it ‘points’ to the object receiving the value of the expression.

In most contexts the ‘=’ operator can be used as an alternative.

R handles vector objects quite easily and intuitively.

```
> x <- c(1,3,2,10,5)    # create a vector x with 5 components
> x
[1]  1  3  2 10  5
> y <- 1:5              # create a vector of consecutive integers
> y
[1] 1 2 3 4 5
> y+2                   # scalar addition
[1] 3 4 5 6 7
> 2*y                   # scalar multiplication
[1]  2  4  6  8 10
> y^2                   # raise each component to the second power
[1]  1  4  9 16 25
> 2^y                   # raise 2 to the first through fifth power
[1]  2  4  8 16 32
> y                     # y itself has not been unchanged
[1] 1 2 3 4 5
> y <- y*2
> y                     # y itself has now changed
[1]  2  4  6  8 10
```

Lists

A list is a “generic vector” containing other objects that are usually named and can be anything: numbers, character strings, matrices or even lists. Unlike a vector, whose elements must all be of the same type (all numeric or all character), the elements of a list may have different types. Here's a list with two components created using the function list:

```
> person <- list(name="Jane", age=24)
```

Typing the name of the list prints all elements. You can extract a component of a list using the extract operator $. For example we can list just the name or age of this person:

```
> person$name
[1] "Jane"
> person$age
[1] 24
```

Individual elements of a list can also be accessed using their indices or their names as subscripts. For example we can get the name using person[1] or person["name"]. See http://www.r-tutor.com/r-introduction/list for more info.

Excerpt from: 4 data wrangling tasks in R for advanced beginners
http://www.computerworld.com/s/article/9243391/4_data_wrangling_tasks_in_R_for_advanced_beginners

Here's a sample data set with three years of revenue and profit data
from Apple, Google and Microsoft. (The source of the data was the
companies themselves; fy means fiscal year.) If you'd like to follow
along, you can type (or cut and paste) this into your R terminal window:

```
fy <- c(2010,2011,2012,2010,2011,2012,2010,2011,2012)

company <- c("Apple","Apple","Apple","Google","Google","Google","Microsoft",
"Microsoft","Microsoft")

revenue <- c(65225,108249,156508,29321,37905,50175,62484,69943,73723)

profit <- c(14013,25922,41733,8505,9737,10737,18760,23150,16978)

companiesData <- data.frame(fy, company, revenue, profit)

print(companiesData)
```

The code above creates and prints a data frame,
stored in a variable named "companiesData"

```
str(companiesData)    # Compactly displays the internal *str*ucture of an R object

##'data.frame':     9 obs. of  4 variables:
## $ fy     : num  2010 2011 2012 2010 2011 ...
## $ company: Factor w/ 3 levels "Apple","Google",..: 1 1 1 2 2 2 3 3 3
## $ revenue: num  65225 108249 156508 29321 37905 ...
## $ profit : num  14013 25922 41733 8505 9737 ...
```

How to Source a Script in R

When you want to tell R to perform several commands one after the other without waiting for additional instructions, you use the source() function.

R users refer to this process as sourcing a script.

To prepare your script to be sourced, you first write the entire script in a text editor or in Rstudio, and then you save the script file (using an R extension).

In Rstudio, the editor window is in the top-left corner of the screen. When you press Enter in the editor window, the cursor moves to the next line, as in any text editor.

Example: source("~/Dropbox/Create_Data_Frame_03.R")

For more info use ?source and the link below.

http://www.dummies.com/how-to/content/how-to-source-a-script-in-r.html

Built-in data sets

How can I see what data sets are available when I start R?
http://www.ats.ucla.edu/stat/r/faq/data_sets_avaiable_R.htm

```
library(MASS)  # Loading the MASS package into R
data()
```

Data sets in package ‘datasets’:

AirPassengers	Monthly Airline Passenger Numbers 1949-1960
BJsales	Sales Data with Leading Indicator
BJsales.lead (BJsales)	Sales Data with Leading Indicator
BOD	Biochemical Oxygen Demand
CO2	Carbon Dioxide Uptake in Grass Plants
ChickWeight	Weight versus age of chicks on different diets

. . . and many more

```
# cars_lm_demo.R

# cars is one of the built-in data sets

 lin.mod <- lm(dist~speed,data=cars) # lm is linear model for linear regression
 lin.mod

## Call:
## lm(formula = dist ~ speed, data = cars)

## Coefficients:
## (Intercept)        speed
##     -17.579        3.932

plot(cars$speed, cars$dist) ; grid() ; Sys.sleep(4)
abline(lin.mod, col = "red")

# To learn more:
#     ?lm
#     ?plot
#     ?abline
```

```
class(lin.mod)
##[1] "lm"

summary(lin.mod)

##Call:
##lm(formula = dist ~ speed, data = cars)

##Residuals:
##     Min      1Q  Median      3Q     Max
##-29.069  -9.525  -2.272   9.215  43.201

##Coefficients:
##             Estimate Std. Error  t value  Pr(>|t|)
##(Intercept) -17.5791     6.7584  -2.601   0.0123 *
##speed         3.9324     0.4155   9.464 1.49e-12 ***
##---
##Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

##Residual standard error: 15.38 on 48 degrees of freedom
##Multiple R-squared:  0.6511,    Adjusted R-squared:  0.6438
##F-statistic: 89.57 on 1 and 48 DF,  p-value: 1.49e-12
```

```
#  Another linear model fit and plot example

set.seed(2014)   # set seed value for reproducibility
x <- rnorm(100)  # same as rnorm(100, 0, 1)
y <- x + rnorm(100, 0, 0.1) # (N, center, sigma)
fit <- lm(y ~ x)     # Ord Least Sqs (OLS) linear regression model
fit

##Call:
##lm(formula = y ~ x)

##Coefficients:
##  (Intercept)          x
##    0.001793     1.009070

class(fit)
##[1] "lm"

plot(x, y) ; grid() ; Sys.sleep(3)
abline(fit, col = "red")
```

```
# Another example

set.seed(2014)   # set seed value for reproducibility
x <- rnorm(100)  # same as rnorm(100, 0, 1)
y <- x^2 + rnorm(100, 0, 0.1) # (N, center, sigma)
fit <- lm(y ~ x)     # Ord Least Sqs (OLS) linear regression model
fit

##Call:
##lm(formula = y ~ x)

##Coefficients:
##  (Intercept)          x
##    0.001793     1.009070

class(fit)
##[1] "lm"

plot(x, y) ;  grid(col="blue") ; Sys.sleep(4)
abline(fit, col = "red")
```

```
# Another example

set.seed(2014)   # set seed value for reproducibility
x <- abs( rnorm(100) )  # absolute value of rnorm(100, 0, 1)
y <- x^2 + rnorm(100, 0, 0.1) # (N, center, sigma)
fit <- lm(y ~ x)     # Ord Least Sqs (OLS) linear regression model
fit

##Call:
##lm(formula = y ~ x)

##Coefficients:
##(Intercept)            x
##    -0.8464       2.4386

class(fit)
##[1] "lm"

plot(x, y) ;  grid(col="green") ; Sys.sleep(4)
abline(fit, col = "red")
```

Import or build your own data sets

Importing Data Into R from Different Sources
http://www.r-bloggers.com/importing-data-into-r-from-different-sources/

Example: Import a file and create a data frame

```
traindata <- read.table("~/Dropbox/data/sptrain1.txt", header=FALSE)

class(traindata)
##[1] "data.frame"

str(traindata)

summary(traindata)
```

read.table() reads a file in table format and creates a data frame from it, with cases (observations) corresponding to lines and variables (column names) to fields in the file.

?read.table for more details

Another example:

Let’s create a small and orderly matrix.

```
 Z <- matrix(1:100, ncol = 10)
 Z
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,]    1   11   21   31   41   51   61   71   81    91
## [2,]    2   12   22   32   42   52   62   72   82    92
## [3,]    3   13   23   33   43   53   63   73   83    93
## [4,]    4   14   24   34   44   54   64   74   84    94
## [5,]    5   15   25   35   45   55   65   75   85    95
## [6,]    6   16   26   36   46   56   66   76   86    96
## [7,]    7   17   27   37   47   57   67   77   87    97
## [8,]    8   18   28   38   48   58   68   78   88    98
## [9,]    9   19   29   39   49   59   69   79   89    99
##[10,]   10   20   30   40   50   60   70   80   90   100
```

```
class(Z)
##[1] "matrix"
```

```
# Example: Create an extensible time-series (xts) object

# install.packages('xts')  # if necessary
library(xts)

xtx <- xts(cbind(a=1:9, b=11:19, c=21:29, d=31:39, e=41:49),
order=Sys.Date() + 1:9) # Sys.Date()

xtx

##            a  b  c  d  e
##2014-02-08 1 11 21 31 41
##2014-02-09 2 12 22 32 42
##2014-02-10 3 13 23 33 43
##2014-02-11 4 14 24 34 44
##2014-02-12 5 15 25 35 45
##2014-02-13 6 16 26 36 46
##2014-02-14 7 17 27 37 47
##2014-02-15 8 18 28 38 48
##2014-02-16 9 19 29 39 49  # Sys.Date() is the current date
```

```
xtx.df <- as.data.frame(xtx)    # xtx.df is a data frame, not an xts object

# Another xts example: sp500

> tail(sp500)
                SP500    SPvolume
2013-11-20 1781.37 3109140000
2013-11-21 1795.85 3256630000
2013-11-22 1804.76 3055140000
2013-11-25 1802.48 2998540000
2013-11-26 1802.75 3427120000
2013-11-27 1807.23 2613590000
```

What is the difference between require() and library()?
http://stackoverflow.com/questions/5595512/what-is-the-difference-between-require-and-library

Detecting seasonality (not for beginners)

How to detect whether seasonality is present in a data set.
Sometimes the period of the potential seasonality is known,
but in other cases it is not.

To test if a series is seasonal when the potential period
is known (e.g., with quarterly, monthly, daily or hourly data).

See http://www.r-bloggers.com/detecting-seasonality/ for more details

One simple approach is to fit a model which allows for seasonality
if it is present. For example, you can fit an ETS model using ets()
in R, and if the chosen model has a seasonal component, then
the data is seasonal. For higher frequency data, or where the
seasonal period is non-integer, a TBATS model will do much the
same thing via the tbats() function.

```
# The pigs data (monthly number of pigs slaughtered in Victoria)
# does not look very seasonal when plotted, but the ets function
# selects an ETS(A,N,A) model. That is, it detects an additive
# seasonal component. We can formally test the significance
# of the seasonal component as follows:

library(fma)
fit1 <- ets(pigs)
fit2 <- ets(pigs,model="ANN")

deviance <- 2*c(logLik(fit1) - logLik(fit2))
df <- attributes(logLik(fit1))$df - attributes(logLik(fit2))$df
#P value
1-pchisq(deviance,df)

# The resulting p-value is 5.225962e-07,
# so the additional seasonal component is significant.
```

How to Create an Array in R (from R For Dummies) (not for beginners)

You have two different options for constructing matrices or arrays. Either you use the creator functions matrix() and array(), or you simply change the dimensions using the dim() function.

You can create an array easily with the array() function, where you give the data as the first argument and a vector with the sizes of the dimensions as the second argument. The number of dimension sizes in that argument gives you the number of dimensions. For example, you make an array with four columns, three rows, and two "tables" like this:

```
> my.array <- array(1:24, dim=c(3,4,2))
> my.array
, , 1
   [,1] [,2] [,3] [,4]
[1,]  1  4  7  10
[2,]  2  5  8  11
[3,]  3  6  9  12
, , 2
   [,1] [,2] [,3] [,4]
[1,]  13  16  19  22
[2,]  14  17  20  23
[3,]  15  18  21  24
```

This array has three dimensions. Notice that, although the rows are given as the first dimension, the tables are filled column-wise. So, for arrays, R fills the columns, then the rows, and then the rest.

Change the dimensions of a vector in R (not for beginners)

Alternatively, you could just add the dimensions using the dim() function. This is a little hack that goes a bit faster than using the array() function; it's especially useful if you have your data already in a vector. (This little trick also works for creating matrices, by the way, because a matrix is nothing more than an array with only two dimensions.)

Say you already have a vector with the numbers 1 through 24, like this:

```
> my.vector <- 1:24
```

You can easily convert that vector to an array exactly like my.array simply by assigning the dimensions, like this:

```
> dim(my.vector) <- c(3,4,2)
```

If you check how my.vector looks like now, you see there is no difference from the array my.array that you created before.

See https://stat.ethz.ch/pipermail/r-help/2007-December/147607.html

Built-in Functions

Almost everything in R is done through functions.

Here I'm only referring to numeric and character functions
that are commonly used in creating or recording variables.

See more info at:

http://www.statmethods.net/management/functions.html

Numeric Function examples:

Function	Description
abs(x)	absolute value
sqrt(x)	square root
ceiling(x)	ceiling(3.475) is 4
floor(x)	floor(3.475) is 3
trunc(x)	trunc(5.99) is 5

round(x, digits=n)	round(3.475, digits=2) is 3.48
signif(x, digits=n)	signif(3.475, digits=2) is 3.5

cos(x), sin(x), tan(x) also acos(x), cosh(x), acosh(x), etc.

log(x)	natural logarithm
log10(x)	common logarithm
exp(x)	e^x

Character Functions

Statistical Probability Functions

Other Statistical Functions

See more info at:

http://www.statmethods.net/management/functions.html

Other Useful Functions

Function	Description
seq(from, to, by)	generate a sequence
indices <- seq(1,10,2) # indices is c(1, 3, 5, 7, 9)	
rep(x, ntimes)	repeat x n times
y <- rep(1:3, 2)	# y is c(1, 2, 3, 1, 2, 3)
cut(x, n)	divide continuous variable in factor with n levels
y <- cut(x, 5)	

install.packages("Lahman") # installs R baseball package "Lahman"

library('Lahman') # loads R baseball package "Lahman"

Use ?cut for more info & more examples

An example: Using some built-in functions

```
x  <- seq(-2, 2, 0.05)  # Creates a vector

length(x)  #  x vector has a length of 81

y1 <- pnorm(x)  # pnorm(x, mean = 0, sd = 1)
                # same as pnorm(x,0,1)

y2 <- pnorm(x, 1, 1)

plot(x, y1, type="l", col="red"); grid(col="blue")
Sys.sleep(4)
lines(x, y2, col="green")
```

Try the code.
?seq , ?pnorm , ?plot & ?lines

Modified code to use two different functions

```
x  <- seq(-2, 2, 0.05)

length(x)

y1 <- tanh(x)

y2 <- sin(x)

plot(x,y1,type="l",col="red");grid(col="blue")
Sys.sleep(4)
lines(x,y2,col="green")
```

```
# Same code with modified range for object x

 x  <- seq(-2*pi, 2*pi, 0.05)

 length(x)

y1 <- tanh(x)

y2 <- sin(x)

 plot(x,y1,type="l",col="red");grid(col="blue")
  Sys.sleep(4)
lines(x,y2,col="green")
```

Build your own functions

One of the great strengths of R is the user's ability to add functions. In fact, many of the functions in R are actually functions of functions.

The structure of a function is given below.

```
myfunction <- function(arg1, arg2, ... ){
statements
return(object)
}
```

See:

Introduction to Functions in R
http://www.youtube.com/watch?v=gl9opYcRxO8 (6 minute video)

http://www.statmethods.net/management/userfunctions.html

```
# Example of building your own function

waitafew <- function(x=5)
 {
    p1 <- proc.time()
    Sys.sleep(x)
    proc.time() - p1 # The cpu usage should be negligible
 }

# you can now copy & paste:

# waitafew function will pause for number of seconds specified
waitafew <- function(x=5)  # default is 5 seconds
 {
    p1 <- proc.time()
    Sys.sleep(x)
    proc.time() - p1 # The cpu usage should be negligible
 }

   waitafew(); print(tanh(1)) # pauses 5 secs then prints tanh(1)
```

```
# Another example of building your own function. The Fibonacci series is a
# series of numbers in which each number (Fibonacci number) is the sum of
# the two preceding numbers. The simplest is the series 1, 1, 2, 3, 5, 8, etc.

fib <- function(n=20){
  if(n<3){
  return(c(1,1))
  } else{
    fib <- rep(0, n)
    for(i in 1:n){
      if(i <= 2){
        fib[i] <- 1
      } else{
        fib[i] <- fib[i-1] + fib[i-2]
      }
    }
  }
  return(fib)
}

fib()
[1]   1   1   2   3   5   8  13  21  34  55  89 144 233 377 610 987 1597 2584 4181 6765

fib(5)
[1] 1 1 2 3 5
```

Control Structures

R has the standard control structures you would expect. expr can be multiple (compound) statements by enclosing them in braces { }. It is more efficient to use built-in functions rather than control structures whenever possible.

if-else

if (cond) expr

if (cond) expr1 else expr2

for

for (var in seq) expr

while

while (cond) expr

switch

switch(expr, ...)

ifelse

ifelse(test,yes,no)

Example

```
# transpose of a matrix
# a poor alternative to built-in t() function

mytrans <- function(x) {
  if (!is.matrix(x)) {
    warning("argument is not a matrix: returning NA")
    return(NA_real_)
  }
```

```
y <- matrix(1, nrow=ncol(x), ncol=nrow(x))
  for (i in 1:nrow(x)) {
    for (j in 1:ncol(x)) {
      y[j,i] <- x[i,j]
    }
  }
return(y)
}

# try it
z <- matrix(1:10, nrow=5, ncol=2)
tz <- mytrans(z)
```

Packages

Packages are collections of R functions, data, and compiled code in a well-defined format. The directory where packages are stored is called the library. R comes with a standard set of packages. Others are available for download and installation.
Once installed, they have to be loaded into the session to be used.

```
.libPaths() # get  library location
library()     # see all packages installed
search()    # see packages currently loaded
```

http://www.statmethods.net/interface/packages.html

Install.packages('packageName') # Install package

require(packageName) # package is available for current session

What is the difference between require() and library()?
http://stackoverflow.com/questions/5595512/what-is-the-difference-between-require-and-library

10 Algorithms for Machine Learning Newbies

Machine learning algorithms are described as learning a target function (f)
that best maps input variables (X) to an output variable (Y): Y = f(X)

1-Linear Regression is perhaps one of the most well-known and well-understood algorithms
in statistics and machine learning. lm()

2-Logistic Regression is a technique borrowed by machine learning from the field of statistics.
It is the go-to method for binary classification problems (problems with two class values). glm()

3-Linear Discriminant Analysis - Logistic Regression is a classification algorithm traditionally
limited to only two-class classification problems. If you have more than two classes then the
Linear Discriminant Analysis algorithm is the preferred linear classification technique.
install.packages('devtools')
library(devtools)
install_github("Displayr/flipMultivariates")
library(flipMultivariates)

4-Classification and Regression Trees - Decision Trees are an important type of algorithm
for predictive modeling machine learning. library(evtree), library(rpart)

5-Naive Bayes is a simple but surprisingly powerful algorithm for predictive modeling. The model
is comprised of two types of probabilities that can be calculated directly from your training data:
The probability of each class and the conditional probability for each class given each x value.
devtools::install_github("majkamichal/naivebayes")
library(naivebayes)

6-K-Nearest Neighbors - The KNN cluster algorithm is very simple and very effective. The model representation for KNN is the entire training dataset. Simple right? knn()

7-Learning Vector Quantization - A downside of K-Nearest Neighbors is that you need to hang on to your entire training dataset. The Learning Vector Quantization algorithm (or LVQ for short) is an artificial neural network algorithm that allows you to choose how many training instances to hang onto and learns exactly what those instances should look like.

8-Support Vector Machines are perhaps one of the most popular and talked about machine learning algorithms. A hyperplane is a line that splits the input variable space. In SVM, a hyperplane is selected to best separate the points in the input variable space by their class, either class 0 or class 1. In two-dimensions, you can visualize this as a line, and let's assume that all of our input points can be completely separated by this line. install.packages('e1071'), library('e1071')

9-Bagging and Random Forest - Random Forest is one of the most popular and most powerful machine learning algorithms. It is a type of ensemble machine learning algorithm called Bootstrap Aggregation or bagging. If you get good results with an algorithm with high variance (like decision trees), you can often get better results by bagging that algorithm. install.packages('randomForest'), library(randomForest)

10-Boosting and AdaBoost - Boosting is an ensemble technique that attempts to create a strong classifier from a number of weak classifiers. This is done by building a model from the training data, then creating a second model that attempts to correct the errors from the first model. Models are added until the training set is predicted perfectly or a maximum number of models are added. Adaboost(), Xgboost is short for eXtreme Gradient Boosting package.

https://towardsdatascience.com/a-tour-of-the-top-10-algorithms-for-machine-learning-newbies-dde4edffae11

Bayesian data analysis using Rethinking

This Rethinking R package accompanies a course and book on Bayesian data analysis (McElreath 2016. Statistical Rethinking. CRC Press.). It contains tools for conducting both quick quadratic approximation of the posterior distribution as well as Hamiltonian Monte Carlo (through RStan - mc-stan.org). Many packages do this.

The signature difference of this package is that it forces the user to specify the model as a list of explicit distributional assumptions. This is more tedious than typical formula-based tools, but it is also much more flexible and powerful and --- most important --- useful for teaching and learning. When students have to write out every detail of the model, they actually learn the model.

For example, a simple Gaussian model could be specified with this list of formulas:

```
f <- alist(
    y ~ dnorm( mu , sigma ),
    mu ~ dnorm( 0 , 10 ),
    sigma ~ dexp( 1 )
)
```

The first formula in the list is the probability of the outcome (likelihood); the second is the prior for mu; the third is the prior for sigma.

Quick Installation

You can find a manual with expanded installation and usage instructions here: http://xcelab.net/rm/software/

Here's the brief version.

You'll need to install rstan first. Go to http://mc-stan.org and follow the instructions for your platform. The biggest challenge is getting a C++ compiler configured to work with your installation of R.
The instructions at

https://github.com/stan-dev/rstan/wiki/RStan-Getting-Started

are quite thorough. Obey them, and you'll likely succeed. Then you can install rethinking from within R using:

```
install.packages(c("devtools","mvtnorm","loo","coda"),
                 dependencies=TRUE)
library(devtools)
install_github("rmcelreath/rethinking",ref="Experimental")
```

The code is all on github:

https://github.com/rmcelreath/rethinking/tree/ExperimentalB

Example of how search() is used to see packages currently loaded

```
> search()
 [1] ".GlobalEnv"        "package:tcltk"     "package:quantmod"
 [4] "package:TTR"       "package:Defaults"  "package:xts"
 [7] "package:zoo"       "package:stats"     "package:graphics"
[10] "package:grDevices" "package:utils"     "package:datasets"
[13] "package:methods"   "Autoloads"         "package:base"
```

Example of how library() is used to see all packages installed

```
> library()
Packages in library ‘/home/mike/R/i686-pc-linux-gnu-library/3.0’:

animation               A gallery of animations in statistics and
                         utilities to create animations
assertthat              Easy pre and post assertions.
audio                   Audio Interface for R
betareg                  Beta Regression
BH                      Boost C++ header files
```

. . . and many others

Installing a package – a practice exercise

```
install.packages('tcltk')     # if necessary

library(tcltk)     # Loading the tcltk package into R
demo(tkcanvas)

# -------

library(tcltk)      # Loading the tcltk package into R
demo(tkdensity)
```

```
# For info about adding packages see:
http://www.statmethods.net/interface/packages.html
# A complete list of contributed packages is available from CRAN.

# quantmod package example (demo)

install.packages('quantmod')  # A package to access financial data
library(quantmod)

# Get S&P closing prices and daily volume since 1/1/1950
sp500 <- getSymbols("^GSPC", auto.assign = FALSE, from="1950-01-01")[, (4:5)]

## Change column names
colnames(sp500) <- c("SP500", "SPvolume")

head(sp500)       # to see oldest data (first 6 obs/rows/days)
tail(sp500, 10)    # to see most recent data (last 10 days)

# Calculate the S&P percentage below the all time high
SPdnPCT <-  100 * (1- (tail(sp500$SP500, 1)) / max(sp500$SP500))

print(round(SPdnPCT, digits=2)) # S&P % below the all time high
```

quantmod - Quantitative Financial Modelling & Trading Framework for R
http://www.quantmod.com/

Examples: http://www.quantmod.com/examples/
http://www.quantmod.com/examples/intro/

It is possible with one quantmod function to load data from a variety of sources, including...

Yahoo! Finance (OHLC data)
Federal Reserve Bank of St. Louis FRED® (11,000 economic series)
Google Finance (OHLC data)
Oanda, The Currency Site (FX and Metals)
MySQL databases (Your local data).
R binary formats (.RData and .rda)
Comma Separated Value files (.csv)

More to come including (RODBC,economagic,Rbloomberg,...)

Financial Data Accessible from R

http://www.thertrader.com/2013/10/30/financial-data-accessible-from-r-part-ii/

"I updated my initial post with two new sources of data and the associated R packages: Datastream and PWT. I also added the fImport package from Rmetrics. Following a reader suggestion,
I made the initial table more interactive, moved the data description and package detail below the main table and updated them."

Subsetting Data with R

See http://www.ats.ucla.edu/stat/r/modules/subsetting.htm

```
hsb2.small <- read.csv("http://www.ats.ucla.edu/stat/data/hsb2_small.csv")
# using the names function to see names of the variables
# and which column of data to which they correspond
names(hsb2.small)
##  [1] "id"      "female"  "race"    "ses"     "schtyp"  "prog"    "read"
##  [8] "write"   "math"    "science" "socst"
(hsb3 <- hsb2.small[, c(1, 7, 8)])    # Print hsb3
(hsb4 <- hsb2.small[, 1:4])
(hsb5 <- hsb2.small[1:10, ])
(hsb6 <- hsb2.small[hsb2.small$ses == 1, ])
(hsb7 <- hsb2.small[hsb2.small$id %in% c(12, 48, 86, 11, 20, 195), ])
(hsb8 <- hsb2.small[with(hsb2.small, ses == 3 & female == 0), ])
```

Examples of R Scripts

source('~/Dropbox/R_Pres_Demo_02.R', echo=TRUE)

source('~/Dropbox/SP500_demo_01.R', echo=T)

source('~/Dropbox/Sochi_Medals.R', echo=T)

source("~/Dropbox/Create_Data_Frame_03.R", echo=T)

source('~/Dropbox/SellSignal07.R', echo=T)

source('~/Dropbox/SnP_Signals_02.R', echo=T)

source("~/Dropbox/Neural_Network_Model_Fitting_02.R", echo=T)

source("~/Dropbox/NYSE_nnet_08.R", echo=T)

source("~/Dropbox/Neural_Network_for_Regression_01.R", echo=T)

source("~/Dropbox/Visualizing_linear_systems_01.R", echo=T)

source('~/Dropbox/plot_sine_demo_01.R', echo=T)

source('~/Dropbox/plot_tanh_demo_01.R', echo=T)

Another simple plot example:

```
# Get stats package.  See ?stats for more info.
require(stats)

# Plot data from the cars data set
plot(cars)

# Add lines using the lowess function.
# See ?lowess and ?lines
lines(lowess(cars))
```

```
# Happy belated Valentine’s Day
# https://www.r-bloggers.com/happy-valentines-day-by-nerds/

 dat <- data.frame( t=seq(0, 2*pi, by=0.01) )
 xhrt <- function(t) 16*sin(t)^3
 yhrt <- function(t) 13*cos(t) - 5*cos(2*t) - 2*cos(3*t) - cos(4*t)
 dat$y = yhrt(dat$t)
 dat$x = xhrt(dat$t)

with(dat, plot(x,y, type="l", axes=FALSE, frame.plot=FALSE,
       labels = FALSE, xlab = '', ylab = ''))
with(dat, polygon(x,y, col="#FF7575"))



# Inspired by: http://mathworld.wolfram.com/HeartCurve.html
```

```
# A function to clear the console in R
clc <- function(x=50) cat(rep("\n", x))
clc() # Default is 50 lines

Sys.sleep(4)

clc(4) # 4 lines
```

```
# A quantmod example:

# Don't forget to install the necessary packages
install.packages('quantmod')     # If required

# Load the quantmod package
library(quantmod)

# Get S&P data since 1/1/1950
sp500 <- getSymbols("^GSPC", auto.assign = FALSE, from="1950-01-01")

tail(sp500)

##            GSPC.Open GSPC.High GSPC.Low GSPC.Close GSPC.Volume GSPC.Adjusted
##2013-11-20   1789.59   1795.73  1777.23    1781.37  3109140000      1781.37
##2013-11-21   1783.52   1797.16  1783.52    1795.85  3256630000      1795.85
##2013-11-22   1797.21   1804.84  1794.70    1804.76  3055140000      1804.76
##2013-11-25   1806.33   1808.10  1800.58    1802.48  2998540000      1802.48
##2013-11-26   1802.87   1808.42  1800.77    1802.75  3427120000      1802.75
##2013-11-27   1803.48   1808.27  1802.77    1807.23  2613590000      1807.23

# Not interested in all columns, just two
# Get S&P closing prices and corresponding daily volume since 1/1/1950
sp500 <- getSymbols("^GSPC", auto.assign = FALSE, from="1950-01-01")[, (4:5)]
```

```
tail(sp500)


# Change column names to something more understandable

colnames(sp500) <- c("SP500", "SPvolume")

tail(sp500)

##                SP500   SPvolume
##2013-11-20 1781.37 3109140000
##2013-11-21 1795.85 3256630000
##2013-11-22 1804.76 3055140000
##2013-11-25 1802.48 2998540000
##2013-11-26 1802.75 3427120000
##2013-11-27 1807.23 2613590000
```

My nnet example:

The nnet package is used to fit Feed-forward Neural Networks
with a single hidden layer, possibly with skip-layer connections,
and for Multinomial Log-Linear Models

Some code from an R script that uses nnet:

```
# ?read.table for more details
traindata <- read.table("~/Dropbox/data/sptrain1.txt", header=FALSE)

## ***** Using nnet() in R, train the neural network *****

# library(nnet)
# set.seed(2013) for reproducibility
# Scale the output values (divide by max output)  ==> max(traindata$V15)
# Set size (# of units in the hidden layer) to 20.
# Warning:
#  There's a trade-off between a low error for training vs over-training
#  Over-training will result in poor and ineffective predictions
# skip: switch to add skip-layer connections from input to output.
# linout: switch for linear output units. Default logistic output units.
# decay: parameter for weight decay.  Default 0. Set decay=0.0001
# maxit: maximum number of iterations. Default 100. Set to 19000.
```

```
library(nnet)    # Loading the nnet package into R

set.seed(2013)  ## For reproducibility

# A simple transformation can be produced by xnew = xold * (vmaxn-vminn)/(vmaxo-vmino).
# K <- ( max(traindata$V15) - min(traindata$V15) ) / (+0.95-(-0.95) )
K <- ( max(traindata$V15) - min(traindata$V15) ) / 1.9
K <- ceiling(K)
K

net.stocks <- nnet(V15/K
~V1+V2+V3+V4+V5+V6+V7+V8+V9+V10+V11+V12+V13+V14, data=traindata,
size=25, maxit=100000, skip=TRUE, linout=TRUE, decay=0.00001)

# multiply by K to restore original scale
net.predict <- predict(net.stocks)*K

tail(net.predict, 35)   # To see last 35 predicted values

plot(traindata$V15, net.predict,
    main="Neural network predictions vs actual",
    xlab="Actual")

grid()
```

```
## Run linear regression model
lin.mod <- lm(net.predict ~ traindata$V15)

summary(lin.mod)

#Call:
#lm(formula = net.predict ~ traindata$V15)

#Residuals:
#     Min       1Q   Median       3Q      Max
#-15.2631  -3.2282  -0.3094   3.3986  15.0411

#Coefficients:
#             Estimate Std. Error t value Pr(>|t|)
#(Intercept)    2.52633    0.20979   12.04   <2e-16 ***
#traindata$V15  0.47699    0.01851   25.76   <2e-16 ***
#---
#Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

#Residual standard error: 5.128 on 728 degrees of freedom
#Multiple R-squared:  0.4769,      Adjusted R-squared:  0.4762
#F-statistic: 663.7 on 1 and 728 DF,  p-value: < 2.2e-16
```

```
## ------------------------------------

## Mike added the pause feature
## Pause (sleep) for specified number of seconds
testit <- function(x)
{
    p1 <- proc.time()
    Sys.sleep(x)
    proc.time() - p1 # The cpu usage should be negligible
}
testit(15)  ## Pause will last this number of seconds

## ------------------------------------

plot(traindata$V15, net.predict,
    main="Neural network predictions vs actual",
    xlab="Actual")

## NYSE_nnet_02.R ==> source("~/Dropbox/NYSE_nnet_02.R", echo=T)

## NYSE_nnet_03.R ==> source("~/Dropbox/NYSE_nnet_03.R", echo=T)
## A good example of NOT over-training?
```

Functions useful for viewing aspects of a data frame

```
head(mydata)          # prints the first few rows
tail(mydata)          # prints the last few rows
names(mydata)         # see the variable names
str(mydata)           # check the variable types
summary(mydata)       # detailed info about mydata
ncol(mydata)          # number of columns in data frame
nrow(mydata)          # number of rows
rownames(mydata)      # view row names

fix(mydata)           # To view the entire data frame in a window

subset(x.df, y > 2)   # subsetting rows using the subset function

# How can I learn more about subsetting a data set?
# See http://www.ats.ucla.edu/stat/r/faq/subset_R.htm
```

Useful data frame functions and operations

```
str(mydata)                          # summary of variables included
is.data.frame(mydata)                # TRUE or FALSE

ncol(mydata)                         # number of columns in data frame
nrow(mydata)                         # number of rows

names(mydata)                        # variable names
names(mydata)[1] <- c("quad")        # change 1st variable name to quad
rownames(mydata)                     # optional row names
```

Be careful if you use attach()

It's convenient but risky

If you use attach don't forget to detach when you're done

http://www.r-bloggers.com/to-attach-or-not-attach-that-is-the-question/

So what options exist for those who decide to avoid attach?

- Reference variables directly

    e.g. lm(ds$y ~ ds$x)

- Specify the dataframe for commands which support this

    e.g. lm(y ~ x, data=ds)

- Use the with() function, which returns the value of whatever expression is evaluated

    e.g. with(ds, lm(y ~ x))

http://www.r-bloggers.com/to-attach-or-not-attach-that-is-the-question/

```
# R scripts and associated comments

# SellSignal07.R
# To run as a script:
# source('~/Dropbox/SellSignal07.R', echo=T)
# or source("/home/mike/Dropbox/SellSignal07.R", echo = T)
# The code below is designed to find stock market
# Sell Signals. It is reviewed and improved on a regular basis
# to make sure it works, although maybe not elegantly.

# SnP_Signals_01.R ==> source('~/Dropbox/SnP_Signals_01.R', echo=T)
# Initially based on Things2do_01.R
# Still needs work

# NYSE_nnet_02.R ==> source("~/Dropbox/NYSE_nnet_02.R", echo=T)
# This takes 40 seconds to complete on the netbook.

# NYSE_nnet_03.R ==> source("~/Dropbox/NYSE_nnet_03.R", echo=T)
# A good example of NOT over-training?
# This takes 402 seconds to complete on the netbook. About 6.7 minutes.

# Be kind to your future self
```

```
# These comments are an example of “over-using” comments.
# Using the Batting dataset from the Lahman package which makes
# the complete Lahman baseball database easily accessible from R.

# Say we want to find the ten baseball players who have
# batted in the most games in all of baseball history.

 install.packages('Lahman')   # Installing the Lahman package into R
 library(Lahman)              # Loading the Lahman package into R
 install.packages('plyr')     # Installing the plyr package into R
 library(plyr)                # Loading the plyr package into R

# The next line takes a while. Takes about 68 secs on Mike's netbook.

games <- ddply(Batting, "playerID", summarise, total = sum(G))

head(arrange(games, desc(total)), 10)

# See results on next page
```

```
head(arrange(games, desc(total)), 10)   # See results below
##   playerID    total
##1   rosepe01  3562
##2  yastrca01  3308
##3  aaronha01  3298
##4  henderi01  3081
##5   cobbty01  3035
##6  murraed02  3026
##7  musiast01  3026
##8  ripkeca01  3001
##9   mayswi01  2992
##10 bondsba01  2986
# Look who's in first place with a good lead. Will Pete Rose ever get into the Hall of Fame?
# Look who's tied for sixth place.
# Cal played in more games than Willie.
# Only 8 players played in more than 3000 games,
# Looking further into the database, I noticed that there are only 55 players, out of 17,908,
                                    who played in at least 2500 games.
```

More examples available at:

http://www.cyclismo.org/tutorial/R/

http://math.illinoisstate.edu/dhkim/rstuff/rtutor.html

```
# Let's look at the Old Faithful geyser data, a built-in R data set.

data(faithful)
attach(faithful)  # Be careful
names(faithful)
##[1] "eruptions" "waiting"
hist(eruptions, seq(1.6, 5.2, 0.2), prob=T) # See ?hist
lines(density(eruptions, bw=0.1))
rug(eruptions, side=1) # See ?rug

median(faithful$eruptions)
summary(faithful)
str(faithful)

detach(faithful)
```

Use next demo to move the dots

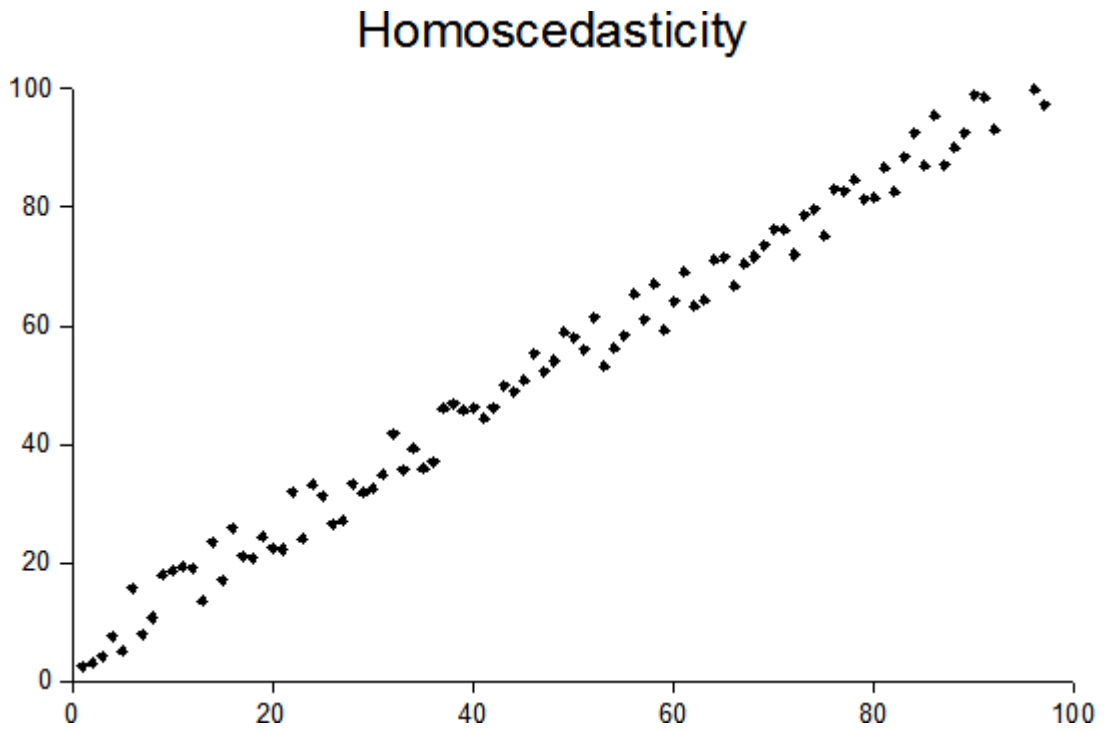
```
library(tcltk)
demo(tkcanvas)
```

Use next demo to move the bw bar

```
library(tcltk)
demo(tkdensity)
```

Homoscedasticity

In statistics, a sequence or a vector is homoscedastic if all random variables in the sequence or vector have the same finite variance. This is also known as homogeneity of variance. The complementary notion is called heteroscedasticity



How to loop in R

Use the for loop if you want to do the same task a specific number of times. It looks like this.

for (counter in vector) {commands}

I’m going to set up a for-loop to square every element of my dataset, foo, which contains the odd integers from 1 to 100.

```
 foo <- seq(1, 100, by=2)

 foo.squared <- NULL  # object foo.squared exists but has no content

for (i in 1:50 ) {
  foo.squared[i] = foo[i]^2
}
```

See http://paleocave.sciencesortof.com/2013/03/writing-a-for-loop-in-r/

If the creation of a new vector is the goal, first you have to set up a vector to store things in prior to running the loop. This is the foo.squared <- NULL part. This was a hard lesson for me to learn. R doesn't like being told to operate on a vector that doesn't exist yet. So, we set up an empty vector to add stuff to later (note that this isn't the most speed efficient way to do this, but it's fairly fool-proof). Next, the real for-loop begins. This code says we'll loop 50 times(1:50). The counter we set up is 'i' (but you can put whatever variable name you want there). For our new vector foo.squared, the ith element will equal the number of loops that we are on (for the first loop, i=1; second loop, i=2).

If you are new to programming it is sometimes difficult to keep straight the difference in the number of loops you are on versus the value of the element of vector being operated on. For example when we've looped through the instructions 4 times, the next loop will be loop number 5 (so i=5). However the 5th element of foo will be foo[5], which is equal to 9. Therefore, foo.squared[5] should equal 81.

http://paleocave.sciencesortof.com/2013/03/writing-a-for-loop-in-r/

R has many diverse applications, e.g.,

Basics of Histograms
http://rforpublichealth.blogspot.com/2012/12/basics-of-histograms.html

Visualizing neural networks in R – update
http://beckmw.wordpress.com/2013/11/14/visualizing-neural-networks-in-r-update/

Introducing CrimeMap - A Web App Powered by ShinyApps!
http://www.r-bloggers.com/introducing-crimemap-a-web-app-powered-by-shinyapps/

Analyzing baseball data with R (another R book review for today)
http://www.r-bloggers.com/analyzing-baseball-data-with-r/

Fantasy Football Modeling with R
http://blog.revolutionanalytics.com/2013/10/fantasy-football-modeling-with-r.html

Ordinary Least Squares is dead to me
http://www.r-bloggers.com/ordinary-least-squares-is-dead-to-me/

You can search for many others at http://www.r-bloggers.com/

More R information is available at:

Top 3 R resources for beginners
http://www.r-bloggers.com/top-3-r-resources-for-beginners/

An Introduction to R - Table of Contents
http://cran.r-project.org/doc/manuals/R-intro.html

Quick-R - accessing the power of R
http://www.statmethods.net/index.html

http://www.cyclismo.org/tutorial/R/

http://math.illinoisstate.edu/dhkim/rstuff/rtutor.html

http://www.r-bloggers.com/

http://ryouready.wordpress.com/

http://www.r-statistics.com/

Additional R information sources:

Statistics with R: Regression, Lesson 9 by Courtney Brown 10/20/13
http://www.youtube.com/watch?v=mNluvMrY9Mc

Statistics with R: Multiple Regression, Lesson 10 by Courtney Brown 10/25/13
http://www.youtube.com/watch?v=g1Ozie3v-Yg
reagan.model <- lm(REAFEEL3 ~ INC + AGE + PARTID, data=mydata)

Manipulating Data Frames in R Day 1 Part 1 of 5 (see all 5 videos)
http://www.youtube.com/watch?v=jFj7tDJxu9Y

Time series forecasting using R
https://www.otexts.org/fpp/resources

R for Dummies (the book)

R programming tips
http://www.avrahamadler.com/r-tips/

“R Instructor” app for tablets & smart phones (worth the $5)

R Script: source('~/Dropbox/R_Pres_Demo_02.R', echo=T)

Additional R information links:

Installation of R 3.5 on Ubuntu 18.04 LTS and tips for spatial packages
https://rtask.thinkr.fr/blog/installation-of-r-3-5-on-ubuntu-18-04-lts-and-tips-for-spatial-packages/

Beyond Basic R - Introduction and Best Practices
https://owi.usgs.gov/blog/intro-best-practices/

http://faculty.chicagobooth.edu/richard.hahn/teaching/FormulaNotation.pdf

http://r4ds.had.co.nz
R for Data Science by Hadley Wickham and Garrett Grolemund

How to Learn R by Tal Galili
http://www.r-bloggers.com/how-to-learn-r-2/

R Presentation Summary

- Download R at http://www.r-project.org/

- Use CRAN - Comprehensive R Archive Network

- Consider using RStudio http://www.rstudio.com/
  http://www.youtube.com/watch?v=jPk6-3prknk
  http://www.youtube.com/watch?v=enVcKR9RMzQ

- R has its own jargon (some of it is not initially intuitive)
  e.g., object, class, vector, list, source, packages

- R is about objects, classes and methods

- Remember to use the assignment operator (‘<-’)

- R comes with its own built-in data sets

- You can install and/or build your own data sets

- R comes with its own built-in functions
  See http://www.statmethods.net/management/functions.html

- You can build your own functions

- R comes with its own built-in packages

- R users can release their own user-built packages
  Some user-built packages expand the scope of R

- You can install user-built packages to address your specific interests e.g., nnet, quantmod, MASS, ggplot2, Rcpp
  e.g., install.packages('quantmod')
  e.g., library(quantmod)

- R has data structures

	Homogeneous	Heterogeneous
1d	Atomic vector	List
2d	Matrix	Data frame
Nd	Array	

See http://adv-r.had.co.nz/Data-structures.html for more details.

- You can build and source your own R scripts
  e.g., source('~/Dropbox/R_Pres_Demo_02.R', echo=T)

Thank you for your attention.