Code Library

Ruhika Chatterjee

2024-12-07

Notes taken from Johns Hopkins University Coursera course series Data Science Specialization.

R Data Types

- Basic object is vector of same class except list.
- Atomic classes of objects: character, numeric (real), integer, complex, logical.
- Attributes can include names, dimnames, dimensions, class, length.

Atomic Data

```
# numeric Vector
x <- 5 # numeric vector of 1 element

# integer vector
x <- 5L # integer vector of len 1

x <- Inf # special number infinity, +/-
x <- NaN # special number undefined, usually hijacks operations

# character vector
msg <- "hello" # char vector of len 1

# logical vector
tf <- TRUE # logical vector of value true
# TRUE = 1 = T, FALSE = 0 = F, num > 0 = TRUE

# complex vector
x <- 1+4i # vector of complex num of len 1
```

complex Data Types

```
# vector
vector("numeric", length = 10) # create vector of one type, args: class, length
```

```
##  [1] 0 0 0 0 0 0 0 0 0 0
```

```r
c(1,2,3,4) # creates vector of common denominator class with given values
```

```
## [1] 1 2 3 4
```

```r
1:20 # vector sequence of 20 elements 1-20
```

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

```r
pi:10 # will not exceed 10, start from pi, increment by 1
```

```
## [1] 3.141593 4.141593 5.141593 6.141593 7.141593 8.141593 9.141593
```

```r
15:1 # increment -1
```

```
##  [1] 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1
```

```r
seq(1,20) # same as :
```

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

```r
seq(0,10,by=0.5) # to change increment
```

```
##  [1]  0.0  0.5  1.0  1.5  2.0  2.5  3.0  3.5  4.0  4.5  5.0  5.5  6.0  6.5  7.0
## [16]  7.5  8.0  8.5  9.0  9.5 10.0
```

```r
seq(5,10,length=30) # to not set increment but number of numbers
```

```
##  [1]  5.000000  5.172414  5.344828  5.517241  5.689655  5.862069  6.034483
##  [8]  6.206897  6.379310  6.551724  6.724138  6.896552  7.068966  7.241379
## [15]  7.413793  7.586207  7.758621  7.931034  8.103448  8.275862  8.448276
## [22]  8.620690  8.793103  8.965517  9.137931  9.310345  9.482759  9.655172
## [29]  9.827586 10.000000
```

```r
seq_along(x) # vector of same length 1:length(x)
```

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

```r
rep(10, times = 4) # repeats 10 4 times in vector
```

```
## [1] 10 10 10 10
```

```r
rep(c(0, 1, 2), times = 10) # repeats sequence of vector 10 times. Arg each can be used to repeat first
```

```
##  [1] 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2
```

```r
# lists
# vector capable of carrying different classes
x <- list(1, "a", TRUE, 1+4i) # vector of vectors

# Matrix
# vector of single class with rectangular dimensions (attribute of integer vector len 2)
x <- matrix(nrow=2,ncol=3) # empty matrix of given dimensions
x <- matrix(1:8, nrow = 4, ncol = 2) # creates matrix of given dimensions with values assigned, created
y <- matrix(rep(10,4),2,2) # creates matrix of 4 10s

x <- 1:10
dim(x) <- c(2,5) # creates matrix out of vector with dimension 2 rows x 5 columns

cbind(1:3,10:12) # creates matrix out of values in vector args, adding by column (1st arg = 1st col)
```

```
##      [,1] [,2]
## [1,]    1   10
## [2,]    2   11
## [3,]    3   12
```

```r
rbind(1:3,10:12) # same but using rows
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]   10   11   12
```

```r
# factors
# self-describing type of vector representing categorical data, ordered or unordered (labels)
x <- factor(c("male","female","female","female","male")) # character vector with specific linear modeli
f <- gl(3,10) # factor 3 levels, 10 times each
table(x) # prints counts of each factor
```

```
## x
## female   male
##      3      2
```

Data Frames

```r
# stores tabular/rectangular data, stored as lists of same length where each element is a column, lengt
x <- data.frame(foo=1:4, bar=c(T,T,F,F)) # creates data frame 2 columns foo and bar, 4 rows unnamed. Ca
x <- read.table(file = "hw1_data.csv", header = TRUE, sep = ",") # read in data from file
x <- read.csv("hw1_data.csv") # same
```

Date and Time Data Types

```r
# useful for time-series data (temporal changes) or other temporal info
# lubridate package by Hadley Wickham
```

```r
# Dates and Times
birthday <- as.Date("1970-01-01") # dates are date class defined by converting character string, year-m
today <- Sys.Date()

currentTime <- Sys.time()# time by POSIXct(large integer vector, useful in dataframe) or POSIXlt(list,
timedefined <- as.POSIXct("2012-10-25 06:00:00") # convert char vector, can define timezone
cTConvert <- as.POSIXlt(currentTime) # reclass, works other way
cTConvert$min # to subset list
```

```
## [1] 14
```

```r
datestring <- c("January 10, 2012 10:40", "December 9, 2011 9:10")
x <- strptime(datestring, "%B %d, %Y %H:%M") # Convert character vector to POSIXlt by defining format (
x
```

```
## [1] "2012-01-10 10:40:00 EST" "2011-12-09 09:10:00 EST"
```

```r
weekdays(birthday) # return day of week, date or time classes
```

```
## [1] "Thursday"
```

```r
months(birthday) # return month on date or time
```

```
## [1] "January"
```

```r
quarters(birthday) # return quarter of date or time
```

```
## [1] "Q1"
```

```r
# Operations
# CANNOT MIX CLASSES - convert
# add and subtract dates, compare dates
currentTime - timedefined # time difference, track of discrepancies (i.e. daylightsavings, timezones, l
```

```
## Time difference of 4453.594 days
```

```r
difftime(currentTime, timedefined, units = "days") # to specify unit
```

```
## Time difference of 4453.594 days
```

Reading and Writing Data

```r
# Read data into R
x <- read.table("hw1_data.csv", header = TRUE, sep = ",") #reading tabular data from text files, return
x <- read.csv("hw1_data.csv") # Same but default separator is ", "  and header = TRUE
write.table(x)
```

```
## "Ozone" "Solar.R" "Wind" "Temp" "Month" "Day"
## "1" 41 190 7.4 67 5 1
## "2" 36 118 8 72 5 2
## "3" 12 149 12.6 74 5 3
## "4" 18 313 11.5 62 5 4
## "5" NA NA 14.3 56 5 5
## "6" 28 NA 14.9 66 5 6
## "7" 23 299 8.6 65 5 7
## "8" 19 99 13.8 59 5 8
## "9" 8 19 20.1 61 5 9
## "10" NA 194 8.6 69 5 10
## "11" 7 NA 6.9 74 5 11
## "12" 16 256 9.7 69 5 12
## "13" 11 290 9.2 66 5 13
## "14" 14 274 10.9 68 5 14
## "15" 18 65 13.2 58 5 15
## "16" 14 334 11.5 64 5 16
## "17" 34 307 12 66 5 17
## "18" 6 78 18.4 57 5 18
## "19" 30 322 11.5 68 5 19
## "20" 11 44 9.7 62 5 20
## "21" 1 8 9.7 59 5 21
## "22" 11 320 16.6 73 5 22
## "23" 4 25 9.7 61 5 23
## "24" 32 92 12 61 5 24
## "25" NA 66 16.6 57 5 25
## "26" NA 266 14.9 58 5 26
## "27" NA NA 8 57 5 27
## "28" 23 13 12 67 5 28
## "29" 45 252 14.9 81 5 29
## "30" 115 223 5.7 79 5 30
## "31" 37 279 7.4 76 5 31
## "32" NA 286 8.6 78 6 1
## "33" NA 287 9.7 74 6 2
## "34" NA 242 16.1 67 6 3
## "35" NA 186 9.2 84 6 4
## "36" NA 220 8.6 85 6 5
## "37" NA 264 14.3 79 6 6
## "38" 29 127 9.7 82 6 7
## "39" NA 273 6.9 87 6 8
## "40" 71 291 13.8 90 6 9
## "41" 39 323 11.5 87 6 10
## "42" NA 259 10.9 93 6 11
## "43" NA 250 9.2 92 6 12
## "44" 23 148 8 82 6 13
## "45" NA 332 13.8 80 6 14
## "46" NA 322 11.5 79 6 15
## "47" 21 191 14.9 77 6 16
## "48" 37 284 20.7 72 6 17
## "49" 20 37 9.2 65 6 18
## "50" 12 120 11.5 73 6 19
## "51" 13 137 10.3 76 6 20
## "52" NA 150 6.3 77 6 21
## "53" NA 59 1.7 76 6 22
```

```
## "54" NA 91 4.6 76 6 23
## "55" NA 250 6.3 76 6 24
## "56" NA 135 8 75 6 25
## "57" NA 127 8 78 6 26
## "58" NA 47 10.3 73 6 27
## "59" NA 98 11.5 80 6 28
## "60" NA 31 14.9 77 6 29
## "61" NA 138 8 83 6 30
## "62" 135 269 4.1 84 7 1
## "63" 49 248 9.2 85 7 2
## "64" 32 236 9.2 81 7 3
## "65" NA 101 10.9 84 7 4
## "66" 64 175 4.6 83 7 5
## "67" 40 314 10.9 83 7 6
## "68" 77 276 5.1 88 7 7
## "69" 97 267 6.3 92 7 8
## "70" 97 272 5.7 92 7 9
## "71" 85 175 7.4 89 7 10
## "72" NA 139 8.6 82 7 11
## "73" 10 264 14.3 73 7 12
## "74" 27 175 14.9 81 7 13
## "75" NA 291 14.9 91 7 14
## "76" 7 48 14.3 80 7 15
## "77" 48 260 6.9 81 7 16
## "78" 35 274 10.3 82 7 17
## "79" 61 285 6.3 84 7 18
## "80" 79 187 5.1 87 7 19
## "81" 63 220 11.5 85 7 20
## "82" 16 7 6.9 74 7 21
## "83" NA 258 9.7 81 7 22
## "84" NA 295 11.5 82 7 23
## "85" 80 294 8.6 86 7 24
## "86" 108 223 8 85 7 25
## "87" 20 81 8.6 82 7 26
## "88" 52 82 12 86 7 27
## "89" 82 213 7.4 88 7 28
## "90" 50 275 7.4 86 7 29
## "91" 64 253 7.4 83 7 30
## "92" 59 254 9.2 81 7 31
## "93" 39 83 6.9 81 8 1
## "94" 9 24 13.8 81 8 2
## "95" 16 77 7.4 82 8 3
## "96" 78 NA 6.9 86 8 4
## "97" 35 NA 7.4 85 8 5
## "98" 66 NA 4.6 87 8 6
## "99" 122 255 4 89 8 7
## "100" 89 229 10.3 90 8 8
## "101" 110 207 8 90 8 9
## "102" NA 222 8.6 92 8 10
## "103" NA 137 11.5 86 8 11
## "104" 44 192 11.5 86 8 12
## "105" 28 273 11.5 82 8 13
## "106" 65 157 9.7 80 8 14
## "107" NA 64 11.5 79 8 15
```

```
## "108" 22 71 10.3 77 8 16
## "109" 59 51 6.3 79 8 17
## "110" 23 115 7.4 76 8 18
## "111" 31 244 10.9 78 8 19
## "112" 44 190 10.3 78 8 20
## "113" 21 259 15.5 77 8 21
## "114" 9 36 14.3 72 8 22
## "115" NA 255 12.6 75 8 23
## "116" 45 212 9.7 79 8 24
## "117" 168 238 3.4 81 8 25
## "118" 73 215 8 86 8 26
## "119" NA 153 5.7 88 8 27
## "120" 76 203 9.7 97 8 28
## "121" 118 225 2.3 94 8 29
## "122" 84 237 6.3 96 8 30
## "123" 85 188 6.3 94 8 31
## "124" 96 167 6.9 91 9 1
## "125" 78 197 5.1 92 9 2
## "126" 73 183 2.8 93 9 3
## "127" 91 189 4.6 93 9 4
## "128" 47 95 7.4 87 9 5
## "129" 32 92 15.5 84 9 6
## "130" 20 252 10.9 80 9 7
## "131" 23 220 10.3 78 9 8
## "132" 21 230 10.9 75 9 9
## "133" 24 259 9.7 73 9 10
## "134" 44 236 14.9 81 9 11
## "135" 21 259 15.5 76 9 12
## "136" 28 238 6.3 77 9 13
## "137" 9 24 10.9 71 9 14
## "138" 13 112 11.5 71 9 15
## "139" 46 237 6.9 78 9 16
## "140" 18 224 13.8 67 9 17
## "141" 13 27 10.3 76 9 18
## "142" 24 238 10.3 68 9 19
## "143" 16 201 8 82 9 20
## "144" 13 238 12.6 64 9 21
## "145" 23 14 9.2 71 9 22
## "146" 36 139 10.3 81 9 23
## "147" 7 49 10.3 69 9 24
## "148" 14 20 16.6 63 9 25
## "149" 30 193 6.9 70 9 26
## "150" NA 145 13.2 77 9 27
## "151" 14 191 14.3 75 9 28
## "152" 18 131 8 76 9 29
## "153" 20 223 11.5 68 9 30
```

```r
# help read.table with colClasses with smaller sample
initial <- read.table("hw1_data.csv", header = TRUE, sep = ",", nrows = 100)
classes <- sapply(initial, class)
tabAll <- read.table("hw1_data.csv", header = TRUE, sep = ",", colClasses = classes)

lines <- readLines("coded.R") # reading lines of text file, return character vector
```

```
## Warning in readLines("coded.R"): incomplete final line found on 'coded.R'
```

```
writeLines("coded.R")
```

```
## coded.R
```

```
# editable textual format retains metadata, helpful for version control, corruption fixable, memory cos
dget("coded.R") # reading R objects deparsed into text files
```

```
## [1] "Hello World"
```

```
dput("coded.R") # takes R object, create R code to reconstruct object saving attributes, names
```

```
## "coded.R"
```

```
source("coded.R") # reading in R code files
```

```
## [1] "Hello World"
```

```
#dump() # multiple R objects

# load() # read in saved workspace read binary objects into R
# save()
# unserialize() # read single R objects in binary form
# serialize()

# Interface to outside world
file(description = "hw1_data.csv") # open connection to standard, uncompressed file. Helps for partial
```

```
## A connection with
## description "hw1_data.csv"
## class       "file"
## mode        "r"
## text        "text"
## opened      "closed"
## can read    "yes"
## can write   "yes"
```

```
# gzfile() # connection to file w compression gzip
# bzfile() # connection to file w compression bzip2
jh <- url("http://www.jhsph.edu", "r") # connection to webpage
close(jh) # to end connection
```

Basic R Functions

Defining Workspace or directory

```r
x <- getwd() # find working directory
dir.create("testdir") # create a directory, args: dir name, for nested recursive = true
```

```
## Warning in dir.create("testdir"): 'testdir' already exists
```

```r
setwd("testdir") # set working dir
file.create("mytest.R") # create file in wd
```

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

```r
file.exists("mytest.R") # check if file exists in wd
```

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

```r
file.info("mytest.R") # file metadata, use $ operator to grab specific items
```

```
##          size isdir mode               mtime               ctime
## mytest.R    0 FALSE  666 2025-01-03 19:14:39 2025-01-03 19:14:39
##                        atime exe
## mytest.R 2025-01-03 19:14:39  no
```

```r
file.rename("mytest.R","mytest2.R") # rename
```

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

```r
file.copy("mytest2.R","mytest3.R") # copy file
```

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

```r
file.remove("mytest2.R") # remove file
```

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

```r
file.path("mytest3.R") # relative path
```

```
## [1] "mytest3.R"
```

```r
setwd(x)

dir() # output files in directory. Also list.files().
```

```
##  [1] "Code-Library.pdf"           "Code-Library.Rmd"
##  [3] "Code Library.Rmd"           "coded.R"
##  [5] "complete.R"                 "corr.R"
##  [7] "Course-Notes.pdf"           "Course Notes.Rmd"
##  [9] "CourseraDataScience.Rproj"  "hw1_data.csv"
## [11] "pollutantmean.R"            "Programming 2.3"
## [13] "Rprof.out"                  "rprog_data_ProgAssignment3"
## [15] "specdata"                   "specdata.zip"
## [17] "testdir"
```

```
files_full <- list.files("specdata", full.names=TRUE) # pull all file names from a directory
ls() # prints the objects in work space
```

```
##  [1] "birthday"    "classes"     "cTConvert"   "currentTime" "datestring"
##  [6] "f"           "files_full"  "initial"     "jh"          "lines"
## [11] "msg"         "tabAll"      "tf"          "timedefined" "today"
## [16] "x"           "y"
```

```
rm(list=ls()) # clear workspace
rm(list=setdiff(ls(), "x")) # clear workspace except x
version #R info version
```

```
##                _
## platform       x86_64-w64-mingw32
## arch           x86_64
## os             mingw32
## crt            ucrt
## system         x86_64, mingw32
## status
## major          4
## minor          4.1
## year           2024
## month          06
## day            14
## svn rev        86737
## language       R
## version.string R version 4.4.1 (2024-06-14 ucrt)
## nickname       Race for Your Life
```

```
sessionInfo() #R info version, packages
```

```
## R version 4.4.1 (2024-06-14 ucrt)
## Platform: x86_64-w64-mingw32/x64
## Running under: Windows 11 x64 (build 22631)
##
## Matrix products: default
##
##
## locale:
## [1] LC_COLLATE=English_United States.utf8
## [2] LC_CTYPE=English_United States.utf8
## [3] LC_MONETARY=English_United States.utf8
## [4] LC_NUMERIC=C
## [5] LC_TIME=English_United States.utf8
##
## time zone: America/New_York
## tzcode source: internal
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
```

```
## loaded via a namespace (and not attached):
##  [1] compiler_4.4.1    fastmap_1.2.0     cli_3.6.3         tools_4.4.1
##  [5] htmltools_0.5.8.1 rstudioapi_0.17.1 yaml_2.3.10       rmarkdown_2.29
##  [9] knitr_1.49        xfun_0.49         digest_0.6.37     rlang_1.1.4
## [13] evaluate_1.0.1
```

```
source("coded.R") # load code into console
```

```
## [1] "Hello World"
```

```
args(ls()) # get arguments for a function
```

```
## NULL
```

```
help(ls) # access documentation on ls() function
```

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

```
?ls # same. for operator use ?`:`
```

Functions and Operations

```
# Input and Evaluation
x <- 1 # assignment operator, evaluates and returns
print(x) # print value as vector
```

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

```
x # auto-prints
```

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

```
# in console, press Tab for auto-completion
LETTERS # predefined character vector of capital letters
```

```
##  [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R" "S"
## [20] "T" "U" "V" "W" "X" "Y" "Z"
```

```
# <<- operator can be used to assign a value to an object in an environment that is different from the

# Mathematical and Statistical Functions
5 + 7 # basic arithmetic operations all work +, -, *, /, ^, %% (modulus). NA affects operation.
```

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

```r
sqrt(4) # square root
```

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

```r
abs(-1:2) # absolute value
```

```
## [1] 1 0 1 2
```

```r
mean(c(3,4,5,6,7)) # return mean of numeric vector
```

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

```r
sd(c(3,4,5,6,7)) # returns standard deviation of numeric vector
```

```
## [1] 1.581139
```

```r
cor(c(3,4,5,6,7), c(61,47,18,18,5)) # correlation of x and y vectors make sure to set arg use for NAs
```

```
## [1] -0.9587623
```

```r
range(c(3,4,5,6,7)) # returns min and max as numeric vector of 2
```

```
## [1] 3 7
```

```r
quantile(c(3,4,5,6,7), probs = 0.25) # returns 25th percentile
```

```
## 25%
##   4
```

```r
-c(0.5,0.8,10) # distributes the negative to all elements of vector
```

```
## [1]  -0.5  -0.8 -10.0
```

```r
# vectorized operations
x <- 1:4; y <- 6:9 # different length vectors
x + y # add the elements of the vectors, all operators work
```

```
## [1]  7  9 11 13
```

```r
x > 2 # returns logical vector, >= or == or any of the logical expressions work
```

```
## [1] FALSE FALSE  TRUE  TRUE
```

```r
# Matrix Operations
x <- matrix(1:4,2,2); y <- matrix(rep(10,4),2,2)
x * y # element wise multiplication, for all operators
```

```
##      [,1] [,2]
## [1,]   10   30
## [2,]   20   40
```

```r
x %*% y # matrix multiplication
```

```
##      [,1] [,2]
## [1,]   40   40
## [2,]   60   60
```

```r
x <- matrix(rnorm(200), 20, 10)
rowSums(x) # vector of sum of rows
```

```
##  [1]  0.05276823 -0.06425126  0.67473812 -0.62814430 -3.19689529  4.03895879
##  [7] -1.91485726  5.96478725  0.64521862  3.37412041 -0.82036547  3.26554188
## [13]  5.96996836  1.93870863 -6.19590049 -2.97490325  0.41344167  0.26612746
## [19] -0.60159161 -0.07419933
```

```r
rowMeans(x) # vector of mean of rows
```

```
##  [1]  0.005276823 -0.006425126  0.067473812 -0.062814430 -0.319689529
##  [6]  0.403895879 -0.191485726  0.596478725  0.064521862  0.337412041
## [11] -0.082036547  0.326554188  0.596996836  0.193870863 -0.619590049
## [16] -0.297490325  0.041344167  0.026612746 -0.060159161 -0.007419933
```

```r
colSums(x) # vector of sum of cols
```

```
##  [1]  6.0257191 -4.3757263 -2.5231993  0.9173667  4.9444562 -0.2507817
##  [7]  1.4766373 -0.2367037  9.0377175 -4.8822146
```

```r
colMeans(x) # vector of mean of cols
```

```
##  [1]  0.30128596 -0.21878632 -0.12615997  0.04586833  0.24722281 -0.01253909
##  [7]  0.07383186 -0.01183518  0.45188588 -0.24411073
```

```r
x <- matrix(rnorm(100), 10, 10)
solve(x) # returns inverse of matrix if invertible
```

```
##             [,1]        [,2]        [,3]        [,4]        [,5]        [,6]
##  [1,]  0.19083019 -0.12849444 -0.17402767  0.10209729  0.03195562 -0.08950636
##  [2,] -0.12957757 -0.10676707 -0.10466807  0.15574729  0.06574375 -0.22646337
##  [3,] -0.09789426  0.32304543 -0.03300032 -0.22803038 -0.20667653  0.03239330
##  [4,]  0.21419275 -0.43591631 -0.21466087  0.50949620 -0.07634813 -0.21113883
##  [5,] -0.03505844  0.21863664  0.28737194 -0.30186055 -0.19010520 -0.13443087
```

```
##  [6,]  0.02041334  0.03577045  0.09430327 -0.31052335 -0.07618801 -0.12804903
##  [7,] -0.06832517  0.02791415  0.04232171 -0.04311783 -0.08414035 -0.02122742
##  [8,]  0.11274286  0.11173587  0.17993679  0.25976864  0.31349826  0.03407510
##  [9,]  0.36300608 -0.28345539 -0.19160333  0.45506934 -0.15380183  0.33410930
## [10,] -0.10447283 -0.08224679  0.08749902 -0.10553960 -0.03806741  0.01992169
##              [,7]         [,8]         [,9]       [,10]
##  [1,] -0.08977329  0.152005853  0.004054082  0.04337136
##  [2,]  0.12161870 -0.106330701  0.082067957  0.22319731
##  [3,]  0.06961687 -0.004216302 -0.185172350 -0.34975762
##  [4,] -0.11920607 -0.279017049  0.286207220  0.10811954
##  [5,]  0.50904289  0.044502805  0.017362982 -0.13310455
##  [6,]  0.17630201 -0.313933834 -0.332071015  0.04255969
##  [7,] -0.24645762  0.010866183 -0.136637582  0.10327416
##  [8,] -0.08915845  0.020583481  0.138135229  0.11404667
##  [9,] -0.10249175  0.214760423 -0.065244234  0.17114748
## [10,]  0.17172469  0.052509908 -0.340499813 -0.25472518
```

```
# Logical operators
5 >= 2 # returns logical. <, >, <=, >=, ==, !=. NA in expression returns NA. Can also use to compare lo
```

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

```
TRUE | FALSE # OR A|B union, AND A&B intersection, NOT !A negation. & operates across vector, && evalua
```

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

```
isTRUE(6 > 4) # also evaluates logical expression
```

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

```
xor(5 == 6, !FALSE) # only returns TRUE if one is TRUE, one is FALSE
```

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

```
which(c(1,2,3,4,5,6) < 2) # returns indices of logical vector where element is TRUE
```

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

```
any(c(1,2,3,4,5,6) < 2) # returns TRUE if any of the logical index values are TRUE
```

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

```
all(c(1,2,3,4,5,6) < 2) # returns TRUE only if all the elements of vector are TRUE
```

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

```r
# Character functions
paste(c("My","name","is"),collapse = " ") # join elements into one element, can join multiple vectors w
```

```
## [1] "My name is"
```

```r
c (c("My","name","is"), "Bob") # add to the vector
```

```
## [1] "My"   "name" "is"   "Bob"
```

```r
# Factors functions
x <- factor(c("male","female","female","female","male")) # can include levels argument to set order (ba
x # prints values in vector and levels
```

```
## [1] male   female female female male
## Levels: female male
```

```r
table(x) # prints labels and counts present
```

```
## x
## female   male
##      3      2
```

```r
unclass(x) # strips class to integer with levels of labels
```

```
## [1] 2 1 1 1 2
## attr(,"levels")
## [1] "female" "male"
```

```r
# Display Data Functions
print(data.frame(foo = 1:20, rar = 301:320)) # print whole object
```

```
##    foo rar
## 1    1 301
## 2    2 302
## 3    3 303
## 4    4 304
## 5    5 305
## 6    6 306
## 7    7 307
## 8    8 308
## 9    9 309
## 10  10 310
## 11  11 311
## 12  12 312
## 13  13 313
## 14  14 314
## 15  15 315
## 16  16 316
## 17  17 317
## 18  18 318
## 19  19 319
## 20  20 320
```

```r
head(data.frame(foo = 1:20, rar = 301:320)) # prints preview of first 6 lines
```

```
##   foo rar
## 1   1 301
## 2   2 302
## 3   3 303
## 4   4 304
## 5   5 305
## 6   6 306
```

```r
tail(data.frame(foo = 1:20, rar = 301:320)) # prints preview of last 6 lines
```

```
##    foo rar
## 15  15 315
## 16  16 316
## 17  17 317
## 18  18 318
## 19  19 319
## 20  20 320
```

```r
table(c(1,1,1,2,2,2,2,2,2,2,2,2,3,3,3,3,4,4,5)) # returns table of counts
```

```
##
## 1 2 3 4 5
## 3 9 4 2 1
```

```r
summary(c(3,4,5,6,7)) # result summaries of the results of various model fitting functions based on cla
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##       3       4       5       5       6       7
```

```r
unique(c(3,4,5,6,7,3,3,5,7,2,8,3,5,6)) # returns only unique elements, duplicates removed
```

```
## [1] 3 4 5 6 7 2 8
```

```r
# str function - compactly display internal structure of R object (esp large lists). Diagnostic, altern
str(unclass(as.POSIXlt(Sys.time()))) # prints list clearly
```

```
## List of 11
##  $ sec   : num 40.4
##  $ min   : int 14
##  $ hour  : int 19
##  $ mday  : int 3
##  $ mon   : int 0
##  $ year  : int 125
##  $ wday  : int 5
##  $ yday  : int 2
##  $ isdst : int 0
##  $ zone  : chr "EST"
##  $ gmtoff: int -18000
##  - attr(*, "tzone")= chr [1:3] "" "EST" "EDT"
##  - attr(*, "balanced")= logi TRUE
```

```r
str(lm) # list of function arguments
```

```
## function (formula, data, subset, weights, na.action, method = "qr", model = TRUE,
##     x = FALSE, y = FALSE, qr = TRUE, singular.ok = TRUE, contrasts = NULL,
##     offset, ...)
```

```r
str(rnorm(100,2,4)) # type of vector, length, first 5 elements
```

```
##  num [1:100] 6.83 1.951 1.259 -0.724 3.283 ...
```

```r
str(gl(40,10)) # for factors
```

```
##  Factor w/ 40 levels "1","2","3","4",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```r
# Missing Values
# represented as NA (missing, with specified class) or NaN (missing or undefined)
# NaN is NA but NA not always NaN
is.na(c (1,2,NA,5,6,NA, NA,3, NaN)) # output logical vector of length of input
```

```
## [1] FALSE FALSE  TRUE FALSE FALSE  TRUE  TRUE FALSE  TRUE
```

```r
is.nan(c (1,2,NaN,5,6,NA, NaN,3)) # output logical vector of length of input
```

```
## [1] FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE
```

Attributes of objects

```r
x <- c(0.5,105,10,0.1,2)
class(x) # determine class of object
```

```
## [1] "numeric"
```

```r
attributes(x) # function to return or modify attributes of object
```

```
## NULL
```

```r
identical(x,x) # returns logical for if two objects are identical
```

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

```r
length(x) # to specifically get the length of vector
```

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

```r
dim(x) # to get dimensions of matrix, data frame (row, column)
```

```
## NULL
```

```r
object.size(x) # return memory occupied in bytes
```

```
## 96 bytes
```

```r
as.numeric(0:6) # explicit coercion, works on all atomic classes, if not possible converts to NA and wa
```

```
## [1] 0 1 2 3 4 5 6
```

```r
# data frames
row.names(x) # get and set row names (attributes). Can also use rownames(x)
colnames(x) # get and set row names
```

```
## NULL
```

```r
nrow(x) # number of rows
```

```
## NULL
```

```r
ncol(x) # number of columns
```

```
## NULL
```

```r
data.matrix(x) # converts data frame to matrix, coercion
```

```
##       [,1]
## [1,]   0.5
## [2,] 105.0
## [3,]  10.0
## [4,]   0.1
## [5,]   2.0
```

```r
dim(x) # (row, column) dimensions of data frame
```

```
## NULL
```

```r
# names attribute
x <- 1:3
names(x) # is null
```

```
## NULL
```

```r
names(x) <- c("foo","bar","norf") #now not numbered vector but named, print x and names(x) with names
vect <- c(foo = 11, bar = 2, norf = NA) # adds elements with names to vector directly
# also for lists, names vectors not items
m <- matrix(1:4,nrow = 2, ncol = 2)
dimnames(m) <- list(c("a","b"),c("c","d")) # each dimension has a name for matrices, rows names then co
```

Indexing, Subsetting, and Dealing with NAs

```r
# Subsetting R Objects
x <- c("a","b","c","c","d","a")
x[1] # more than one element extracted, returns same class as the original, numeric/logical index
```

```
## [1] "a"
```

```r
x[1:4] # sequence of num index
```

```
## [1] "a" "b" "c" "c"
```

```r
x[x>"a"] # logical indexing, returns vector where logical is true
```

```
## [1] "b" "c" "c" "d"
```

```r
u <- x > "a" # create logical vector
x[u] # same as x[x>"a"]
```

```
## [1] "b" "c" "c" "d"
```

```r
x[!is.na(x) & x > 0] # returns only positive, non NA values
```

```
## [1] "a" "b" "c" "c" "d" "a"
```

```r
x[c(-2, -10)] # returns vector with 2nd and 10th elements removed
```

```
## [1] "a" "c" "c" "d" "a"
```

```r
x <- data.frame(foo = 1:6, bar = c("g","h","i","j","k","l"))
x[[which(x$bar == "h"), "foo"]] # get or set foo in the same row as bar of "h"
```

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

```r
x <- list(foo = 1:4, bar = 0.6, baz = "hello")
x[1] # list containing first element
```

```
## $foo
## [1] 1 2 3 4
```

```r
x[[1]] # extract from list/data frame, single element, class can change. Ex, numerical vector returned
```

```
## [1] 1 2 3 4
```

```r
x$bar # like [[]] but by name. Ex, return num vector 0.6. Equivalent to x[["bar"]]. Expression x["bar"]
```

```
## [1] 0.6
```

```r
x[c(1,3)] # multiple object extraction from list, returns list
```

```
## $foo
## [1] 1 2 3 4
##
## $baz
## [1] "hello"
```

```r
name = "foo"
x[[name]] # must be used if using computed index
```

```
## [1] 1 2 3 4
```

```r
x[1][3] # return element in element in object
```

```
## $<NA>
## NULL
```

```r
x[[c(1,3)]]
```

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

```r
# Subsetting Matrix
x <- matrix(1:6, 2, 3)
x [1,2] # returns vector len 1, different that x[2,1]. Get matrix using arg drop = FALSE.
```

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

```r
x[1,] # get num vector of first row, can also get col x[,2]. drop = FALSE also works
```

```
## [1] 1 3 5
```

```r
# Removing NA values
x <- c(1,2,NA,4,NA,5)
bad <- is.na(x) # logical vector indicating presence of NA
x[!bad] # removes NA values
```

```
## [1] 1 2 4 5
```

```r
x[!is.na(x)] # simplified returns vector removing NA values
```

```
## [1] 1 2 4 5
```

```r
x <- c(1,2,NA,4,NA,5) # for two vectors
y <- c("a","b",NA,"d",NA,"f")
good <- complete.cases(x,y) # logical vectors where there is no NA in either list
x[good]
```

```
## [1] 1 2 4 5
```

```r
y[good]
```

```
## [1] "a" "b" "d" "f"
```

```r
# Sum of NA values
my_na <- is.na(x)
sum(my_na)
```

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

```r
x <- read.csv("hw1_data.csv") # for data frames
goodVals <- complete.cases(x) # complete rows in the data frame
x[goodVals,]
```

```
##     Ozone Solar.R Wind Temp Month Day
## 1      41     190  7.4   67     5   1
## 2      36     118  8.0   72     5   2
## 3      12     149 12.6   74     5   3
## 4      18     313 11.5   62     5   4
## 7      23     299  8.6   65     5   7
## 8      19      99 13.8   59     5   8
## 9       8      19 20.1   61     5   9
## 12     16     256  9.7   69     5  12
## 13     11     290  9.2   66     5  13
## 14     14     274 10.9   68     5  14
## 15     18      65 13.2   58     5  15
## 16     14     334 11.5   64     5  16
## 17     34     307 12.0   66     5  17
## 18      6      78 18.4   57     5  18
## 19     30     322 11.5   68     5  19
## 20     11      44  9.7   62     5  20
## 21      1       8  9.7   59     5  21
## 22     11     320 16.6   73     5  22
## 23      4      25  9.7   61     5  23
## 24     32      92 12.0   61     5  24
## 28     23      13 12.0   67     5  28
## 29     45     252 14.9   81     5  29
## 30    115     223  5.7   79     5  30
## 31     37     279  7.4   76     5  31
```

```
## 38     29     127  9.7   82     6   7
## 40     71     291 13.8   90     6   9
## 41     39     323 11.5   87     6  10
## 44     23     148  8.0   82     6  13
## 47     21     191 14.9   77     6  16
## 48     37     284 20.7   72     6  17
## 49     20      37  9.2   65     6  18
## 50     12     120 11.5   73     6  19
## 51     13     137 10.3   76     6  20
## 62    135     269  4.1   84     7   1
## 63     49     248  9.2   85     7   2
## 64     32     236  9.2   81     7   3
## 66     64     175  4.6   83     7   5
## 67     40     314 10.9   83     7   6
## 68     77     276  5.1   88     7   7
## 69     97     267  6.3   92     7   8
## 70     97     272  5.7   92     7   9
## 71     85     175  7.4   89     7  10
## 73     10     264 14.3   73     7  12
## 74     27     175 14.9   81     7  13
## 76      7      48 14.3   80     7  15
## 77     48     260  6.9   81     7  16
## 78     35     274 10.3   82     7  17
## 79     61     285  6.3   84     7  18
## 80     79     187  5.1   87     7  19
## 81     63     220 11.5   85     7  20
## 82     16       7  6.9   74     7  21
## 85     80     294  8.6   86     7  24
## 86    108     223  8.0   85     7  25
## 87     20      81  8.6   82     7  26
## 88     52      82 12.0   86     7  27
## 89     82     213  7.4   88     7  28
## 90     50     275  7.4   86     7  29
## 91     64     253  7.4   83     7  30
## 92     59     254  9.2   81     7  31
## 93     39      83  6.9   81     8   1
## 94      9      24 13.8   81     8   2
## 95     16      77  7.4   82     8   3
## 99    122     255  4.0   89     8   7
## 100    89     229 10.3   90     8   8
## 101   110     207  8.0   90     8   9
## 104    44     192 11.5   86     8  12
## 105    28     273 11.5   82     8  13
## 106    65     157  9.7   80     8  14
## 108    22      71 10.3   77     8  16
## 109    59      51  6.3   79     8  17
## 110    23     115  7.4   76     8  18
## 111    31     244 10.9   78     8  19
## 112    44     190 10.3   78     8  20
## 113    21     259 15.5   77     8  21
## 114     9      36 14.3   72     8  22
## 116    45     212  9.7   79     8  24
## 117   168     238  3.4   81     8  25
## 118    73     215  8.0   86     8  26
```

```
## 120    76     203  9.7   97     8  28
## 121   118     225  2.3   94     8  29
## 122    84     237  6.3   96     8  30
## 123    85     188  6.3   94     8  31
## 124    96     167  6.9   91     9   1
## 125    78     197  5.1   92     9   2
## 126    73     183  2.8   93     9   3
## 127    91     189  4.6   93     9   4
## 128    47      95  7.4   87     9   5
## 129    32      92 15.5   84     9   6
## 130    20     252 10.9   80     9   7
## 131    23     220 10.3   78     9   8
## 132    21     230 10.9   75     9   9
## 133    24     259  9.7   73     9  10
## 134    44     236 14.9   81     9  11
## 135    21     259 15.5   76     9  12
## 136    28     238  6.3   77     9  13
## 137     9      24 10.9   71     9  14
## 138    13     112 11.5   71     9  15
## 139    46     237  6.9   78     9  16
## 140    18     224 13.8   67     9  17
## 141    13      27 10.3   76     9  18
## 142    24     238 10.3   68     9  19
## 143    16     201  8.0   82     9  20
## 144    13     238 12.6   64     9  21
## 145    23      14  9.2   71     9  22
## 146    36     139 10.3   81     9  23
## 147     7      49 10.3   69     9  24
## 148    14      20 16.6   63     9  25
## 149    30     193  6.9   70     9  26
## 151    14     191 14.3   75     9  28
## 152    18     131  8.0   76     9  29
## 153    20     223 11.5   68     9  30
```

Random Numbers

```
# Random number generation
# Probability distribution functions have 4 functions associated: d- density, r- random number generati
set.seed(1) # set sequence of random number generation. set.seed(1); rnorm(5) always results in the sam
y <- rnorm(1000) # generate vector of 1000 numbers that are standard normal distribution. Agrs: n, mean
y <- dnorm(c(0.25,0.5,0.75)) # evaluate Normal probability density, (given mean,sd) at point or vector
y <- pnorm(0.5) # evaluate cumulative distribution function for normal distribution. Args: q, mean=0, s
y <- qnorm(0.5) # evaluates quantiles for normal distribution. Args: p, mean=0, sd=1, lower.tail=TRUE,
y <- sample(1:6,3) # random selection of 3 elements from array
ints <- sample(10) # random sample all integers from 1 to 10 without replacement. Permutation
nums <- sample(1:10, replace = TRUE) # with replacement
let <- sample(LETTERS) # sample all letters without replacement
flips <- sample(c(0,1), 100, replace = TRUE, prob = c(0.3,0.7)) # unfair coin
coin <- rbinom(1,1,0.5) # simulating coin flip
unfairflip <- rbinom(1, size = 100, prob = 0.7) # sum of flips above
flips2 <- rbinom(100,1,0.7) # flips above
```

```r
y <- rpois(10, 1) # generate random poisson variates with given rate. Args: n (count), rate (mean)
pois_mat <- replicate(100, rpois(5, 10))

# Simulate Linear Model Ex
# y = B(o) + B(1) * x + e
# e ~ N(0,2^2) assume x ~ N(0,1^2), B(0) = 0.5, B(1) = 2.
set.seed(20)
x <- rnorm(100)
e <- rnorm(100,0,2)
y <- 0.5 + 2 * x + e
# can combine different distributions
# Poisson: Y ~ Poisson(mu)
# log(mu) = B(0) + B(1)x
# B(0) = 0.5 and B(1) = 0.3
set.seed(1)
x <- rnorm(100)
log.mu <- 0.5 + 0.3 * x
y <- rpois(100, exp(log.mu))
```

Control Functions and Loop Functions

Control Functions

```r
# control execution of program

x = 2
# if,else loops
y <- if(x > 3){ # testing condition
  10
} else if(x > 0 & x <= 3) { # can not have or multiple
  5
} else{ # can not have, at end
  0
}

if(x-5 == 0){
  y <- 0
} else{
  y <- 2
}

# for loops
for(i in 1:10) {# execute loop fixed number of times. Args iterator variable and vector(inc seq) or lis
  print(i)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
```

```
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
```

```
x <- c("a","b","c","d")
for(i in 1:4){
  print(x[i])
}
```

```
## [1] "a"
## [1] "b"
## [1] "c"
## [1] "d"
```

```
for(i in seq_along(x)){
  print(x[i])
}
```

```
## [1] "a"
## [1] "b"
## [1] "c"
## [1] "d"
```

```
for(letter in x){
  print(letter)
}
```

```
## [1] "a"
## [1] "b"
## [1] "c"
## [1] "d"
```

```
for(i in 1:4) print(x[i])
```

```
## [1] "a"
## [1] "b"
## [1] "c"
## [1] "d"
```

```
x <- matrix(1:6,2,3)
for(i in seq_len(nrow(x))) { # nested, don't use more than 2-3 for readability
  for(j in seq_len(ncol(x))) {
    print(x[i,j])
  }
}
```

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

```
## [1] 3
## [1] 5
## [1] 2
## [1] 4
## [1] 6
```

```
# while loops
count <- 0
while(count < 10){ # loop while condition is true
  print(count)
  count <- count + 1
} # be wary of infinite loops!! when condition cannot be true
```

```
## [1] 0
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
```

```
z <- 5
while(z >= 3 & z <= 10){
  print(z)
  coin <- rbinom(1,1,0.5)

  if (coin == 1) z <- z+1
  else z <- z-1
}
```

```
## [1] 5
## [1] 6
## [1] 5
## [1] 4
## [1] 3
```

```
# Repeat loop
x0 <- 0.01; tol <- 1e-3
repeat { # infinite loop
  x1 <- rnorm(1)
  if(abs(x1 - x0) < tol) {
    break # break execution of any loop
  }
  else x0 <- x1
}

# control a loop
for(i in 1:100) {
  if(i <= 20) next # skip next iteration of loop
```

```
  else {
    if (i > 50) break # exit for loop
  }
}

# return to exit a function, will end control structure inside function
```

Loop Functions

```
# Loop functions - useful for looping in the command line
# Hadley Wickham's Journal of Statistical Software paper titled 'The Split-Apply-Combine Strategy for D

# lapply - loop over a list and evaluate on each element. args: X (list or coercion), FUN (function or
x <- list(a = 1:5, b = rnorm(10))
lapply(x, mean) # returns list of 2 numerics
```

```
## $a
## [1] 3
##
## $b
## [1] 0.3985388
```

```
x <- 1:4
lapply(x, runif, min = 0, max = 10) # passes subsequent args to function
```

```
## [[1]]
## [1] 4.180447
##
## [[2]]
## [1] 5.3804163 0.7510495
##
## [[3]]
## [1] 3.049216 2.719333 8.182229
##
## [[4]]
## [1] 0.8832537 3.4918707 8.5187127 9.8035107
```

```
x <- list(a = matrix(1:4, 2, 2), b = matrix(1:6, 3, 2))
lapply(x, function(elt) elt[,1]) # define an anonymous function inside lapply
```

```
## $a
## [1] 1 2
##
## $b
## [1] 1 2 3
```

```
# sapply - same as lapply but simplify, i.e. will make list of 1 element vectors a vector, multiple ele
x <- list(a = 1:5, b = rnorm(10))
lapply(x, mean) # now returns vector length 2
```

```
## $a
## [1] 3
##
## $b
## [1] 0.3902621
```

```
# mean only operates on signle element numeric/logical, so need to use loop

# vapply - pre-specify type of return value, safer and faster. Args: X, FUN, FUN.VALUE (generalized vec
vapply(x, mean, numeric(1)) # same as sapply(x, mean)
```

```
##         a         b
## 3.0000000 0.3902621
```

```
# apply - apply function over margins of array (good for summary of matrices or higher level array). No
x <- matrix(rnorm(200), 20, 10)
apply(x, 2, mean) # mean of each column by collapsing 1st dimension, returns num vector length of ncol.
```

```
##  [1]  0.39576926  0.39693829 -0.29548099 -0.30587580  0.31690617 -0.24744022
##  [7]  0.26027330  0.07700510 -0.04652335 -0.23800285
```

```
rowSums(x) # equivalent to apply(x, 1, sum)
```

```
##  [1]  0.7609383 -4.1967248  5.0584592  0.5808195 -3.3859346  8.2206313
##  [7]  1.3595547 -0.1567391  2.1183256  2.2432819  3.0644650  1.3968116
## [13]  5.4715183 -2.0890661 -0.7462932  0.6588537 -2.3037316 -3.9577417
## [19] -4.5847842 -3.2412655
```

```
rowMeans(x) # equivalent to apply(x, 1, mean)
```

```
##  [1]  0.07609383 -0.41967248  0.50584592  0.05808195 -0.33859346  0.82206313
##  [7]  0.13595547 -0.01567391  0.21183256  0.22432819  0.30644650  0.13968116
## [13]  0.54715183 -0.20890661 -0.07462932  0.06588537 -0.23037316 -0.39577417
## [19] -0.45847842 -0.32412655
```

```
colSums(x) # apply(x, 2, sum)
```

```
##  [1]  7.9153853  7.9387658 -5.9096198 -6.1175160  6.3381234 -4.9488043
##  [7]  5.2054659  1.5401019 -0.9304669 -4.7600570
```

```
colMeans(x) #apply(x, 2, mean)
```

```
##  [1]  0.39576926  0.39693829 -0.29548099 -0.30587580  0.31690617 -0.24744022
##  [7]  0.26027330  0.07700510 -0.04652335 -0.23800285
```

```
apply(x, 1, quantile, probs = c(0.25, 0.75)) # runs quantile with 2 agrs for every element in list, ret
```

```
##           [,1]       [,2]       [,3]       [,4]       [,5]      [,6]      [,7]
## 25% -0.7571352 -0.7008243 -0.2473744 -0.6418233 -0.5553241 0.4407991 0.1330451
## 75%  1.0018439 -0.1354330  1.4629405  0.7853807  0.1146551 1.2413891 1.0496289
##           [,8]       [,9]      [,10]      [,11]      [,12]      [,13]
## 25% -0.7121101 -0.2189623 -0.1192721 -0.1532343 -0.1754369 -0.2377421
## 75%  0.2584542  0.7632128  1.0476237  0.7373560  0.4873948  1.1822437
##          [,14]      [,15]      [,16]      [,17]      [,18]     [,19]      [,20]
## 25% -0.7463529 -0.3586825 -0.7882050 -0.7254670 -0.8275423 -0.799203 -0.8441518
## 75%  0.2985215  0.1503075  0.7323559  0.3109353  0.7182965  0.290152 -0.3795045
```

```
a <- array(rnorm(2 * 2 * 10), c(2, 2, 10)) # array in 3D
apply(a, c(1,2), mean) # collapses only 3rd dimension, returns 2x2 matrix. Equivalent rowMeans(a, dims
```

```
##            [,1]       [,2]
## [1,] -0.3294688  0.1786066
## [2,] -0.1456898 -0.2877835
```

```
# tapply - apply function over subset of a vector. args: X is vector, INDEX is factor/list factors vect
x <- c(rnorm(10), runif(10), rnorm(10,1))
f <- gl(3,10) # factor 3 levels, 10 times each
tapply(x,f,mean)
```

```
##         1         2         3
## 0.2233750 0.3445618 0.4956007
```

```
# mapply - multivariate version of lapply. args: FUN as above, ... (arguments to apply over), MoreArgs
list(rep(1,4), rep(2,3), rep(3,2), rep(4,1))
```

```
## [[1]]
## [1] 1 1 1 1
##
## [[2]]
## [1] 2 2 2
##
## [[3]]
## [1] 3 3
##
## [[4]]
## [1] 4
```

```
mapply(rep, 1:4, 4:1) # equivalent
```

```
## [[1]]
## [1] 1 1 1 1
##
## [[2]]
## [1] 2 2 2
##
## [[3]]
## [1] 3 3
##
## [[4]]
## [1] 4
```

```r
noise <- function(n,mean,sd){rnorm(n,mean,sd)}
noise(1:5,1:5,2) # gives vector of 5, same as single num args
```

```
## [1] 0.5569244 1.6755360 3.6736906 6.1633545 7.0603864
```

```r
mapply(noise,1:5,1:5,2) # applies function for each pair, list of 5 of length i
```

```
## [[1]]
## [1] 0.2931417
##
## [[2]]
## [1] 4.150763 1.569355
##
## [[3]]
## [1] 3.367118 4.901136 5.167102
##
## [[4]]
## [1] 1.010141 2.712134 6.857595 3.225181
##
## [[5]]
## [1] 4.858346 4.861672 4.799808 5.668715 2.350646
```

```r
# split - in conjunction with lapply to split objects into subpieces. Args: x (any object), f (factor),
x <- c(rnorm(10), runif(10), rnorm(10,1))
f <- gl(3,10) # factor 3 levels, 10 times each
split(x,f) # tapply without function, sorts into list based on levels, can then use lapply or sapply.
```

```
## $`1`
##  [1]  0.78002347 -0.78709697 -0.58691682 -0.54546587  0.76247880  0.06403316
##  [7]  0.12819144  0.60560030  0.39492984 -0.53621606
##
## $`2`
##  [1] 0.61128364 0.50431157 0.49886556 0.15303652 0.58167801 0.05305581
##  [7] 0.08354486 0.19449867 0.50655472 0.80669924
##
## $`3`
##  [1]  1.1065169  1.2236401  0.7009779  1.8481351  2.4935228  0.3468278
##  [7] -0.3368842  1.8210809  0.5114539  1.2572268
```

```r
lapply(split(x,f), mean) # in this case can use tapply
```

```
## $`1`
## [1] 0.02795613
##
## $`2`
## [1] 0.3993529
##
## $`3`
## [1] 1.09725
```

```r
# can do data frames
data <- read.csv("hw1_data.csv")
s <- split(data, data$Month)
sapply(s, function(x) colMeans(x[,c("Ozone","Solar.R","Wind")], na.rm = TRUE)) # data$Month coerced int
```

```
##                 5         6          7          8         9
## Ozone    23.61538  29.44444  59.115385  59.961538  31.44828
## Solar.R 181.29630 190.16667 216.483871 171.857143 167.43333
## Wind     11.62258  10.26667   8.941935   8.793548  10.18000
```

```r
# Multi-level split
x <- rnorm(10)
f1 <- gl(2,5); f2 <- gl(5,2) # ex. race and gender 2 factors
interaction(f1,f1) # combine each pair, 10 factors
```

```
##  [1] 1.1 1.1 1.1 1.1 1.1 2.2 2.2 2.2 2.2 2.2
## Levels: 1.1 2.1 1.2 2.2
```

```r
split(x, list(f1,f2)) # interaction called, list returned for combination sort, drop = TRUE to remove u
```

```
## $`1.1`
## [1]  0.1165892 -0.1194990
##
## $`2.1`
## numeric(0)
##
## $`1.2`
## [1]  0.4679266 -1.4368877
##
## $`2.2`
## numeric(0)
##
## $`1.3`
## [1] 0.5310122
##
## $`2.3`
## [1] -0.8627139
##
## $`1.4`
## numeric(0)
##
## $`2.4`
## [1] -1.2451944  0.6457308
##
## $`1.5`
## numeric(0)
##
## $`2.5`
## [1] -0.3394378 -0.2064004
```

Defining Functions

```
# stored in txt or R script, functions are R objects. Can pass functions as arguments for other functio

myfunction <- function(){ #create a function
  x <- rnorm(100)
  mean(x)
}
myfunction() #call created function
```

```
## [1] -0.1028367
```

```
myfunction # prints source code for function
```

```
## function ()
## {
##     x <- rnorm(100)
##     mean(x)
## }
```

```
args(myfunction) # returns arguments for passed function
```

```
## function ()
## NULL
```

```
myaddedfunction <- function(x,y){ #create a function with formal arguments x and y
  x + y + rnorm(100) # implicit return last expression
}
myaddedfunction(5,3)
```

```
##   [1]  7.647769  6.334089  7.593286  6.268142  9.548806  9.191841  8.190586
##   [8]  8.226173  8.766742  9.634012  9.245233  5.921075  6.841281  7.993894
##  [15]  6.722897  9.420619  8.915033  7.623340  8.032766  8.883314  9.142204
##  [22]  8.000106  7.991077  7.685731  6.878269  7.864682  7.372188  8.462985
##  [29]  8.260722  6.964953  8.243108  8.238265  7.603194  7.843892  8.568631
##  [36]  9.067935  7.573488  9.495201  7.325929  6.319661  7.007791  6.507580
##  [43]  9.482838  9.262762  9.473943  7.560676  7.166530  7.353693  9.219610
##  [50]  8.611367  6.644443  8.048528  6.539457  8.539169  7.676676  8.584478
##  [57]  8.233269  7.799053  7.504200  7.847320  6.365965 10.249351  9.269278
##  [64]  6.715166  8.497680  8.015868  6.417507  7.669606  7.323856  8.684615
##  [71]  6.486430  8.711168  8.226389  7.602351  7.960909  5.829723  6.282247
##  [78]  7.605290  7.147546  7.433600  7.803719  7.836636  8.618066  8.022089
##  [85]  8.608157  8.589619  8.760178  9.216551  6.242489  8.209841  8.268497
##  [92]  5.898325  7.066211  7.331588  8.012126  7.676726  9.810349  7.364054
##  [99]  6.261003  7.979133
```

```
myaddedfunction(4:10,2)
```

```
## Warning in x + y + rnorm(100): longer object length is not a multiple of
## shorter object length
```

```
##   [1]  6.409714  6.341150  8.912148  8.792755 11.154946 12.277935 12.926515
##   [8]  5.939344  4.986605  9.444333  8.014862 10.204588 10.538864 13.736550
##  [15]  4.563040  8.080383  8.684333  8.388580 11.043038 10.504557 12.941534
##  [22]  6.405706  5.874937  8.759100  8.274475  9.002475 10.189102 12.813112
##  [29]  8.025275  9.588654  7.999254  7.269448 10.570705 13.922480 13.380562
##  [36]  6.301785  7.219056  8.360819  9.090101 10.532460 10.865300 11.339198
##  [43]  5.243282  6.518864  7.321305  8.387538 10.510045 12.148739 11.287710
##  [50]  5.292170  5.165806  7.699683  8.664981  9.362315 10.495106 14.330201
##  [57]  6.509956  6.178560  7.865012  9.993201  9.207881  9.523909 12.187720
##  [64]  5.027184  7.545402  9.329583  9.364413  9.260397 10.757635 11.208893
##  [71]  6.239572  6.448273  9.943164  8.470355  8.911138 12.665069 11.722099
##  [78]  4.015922  5.826076  7.836202 10.036603  9.776626 11.550650 11.751905
##  [85]  7.533689  5.243682  7.060479  8.798539 10.764209 10.285979  9.854701
##  [92]  5.259725  8.486536  7.838495  9.715276  8.769817 11.769759 11.175956
##  [99]  7.670140  7.005200
```

```
# function with default argument if left unspecified, for common cases
above <- function(x, n = 10){
  use <- x > n
  x[use]
}
above(1:20) # n is default set to 10
```

```
##  [1] 11 12 13 14 15 16 17 18 19 20
```

```
above(1:20, 12) # n set at 12
```

```
## [1] 13 14 15 16 17 18 19 20
```

```
columnmean <- function(y, removeNA = TRUE) {
  nc <- ncol(y)
  means <- numeric(nc)
  for(i in 1:nc) means[i] <- mean(y[,i], na.rm = removeNA)
  invisible(means) # auto-return blocks auto-print
}

# Lazy Evaluation: R evaluated statements and arguments as they come
f <- function (a,b,c){
  print(a)
  #print(b) # error
}
f(3) # prints a, error for b, no rxn to not having c
```

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

```
# ways to call functions
# positional matching and naming can be mixed. Partial matching also allowed, if not found uses positio
# named helps for long arg list where most defaults are maintained or if order is hard to remember.
mydata <- rnorm(100)
sd(mydata) # default to first argument
```

```
## [1] 1.001767
```

```r
sd(x = mydata)
```

```
## [1] 1.001767
```

```r
sd(x = mydata, na.rm = FALSE)
```

```
## [1] 1.001767
```

```r
sd(na.rm = FALSE, x = mydata)
```

```
## [1] 1.001767
```

```r
sd(na.rm = FALSE, mydata) # remove argument from list, default works on first unspecified arg
```

```
## [1] 1.001767
```

```r
# Variable Arguments
# to extend another function without copying arg list of OG function
simon_says <- function(...){
  paste("Simon says:", ...)
}
# or for generic functions passed to methods
# unpacking an ellipses
mad_libs <- function(...){
  args <- list(...)
  place <- args$place
  adjective <- args$adjective
  noun <- args$noun
  paste("News from", place, "today where", adjective, "students took to the streets in protest of the n
}
# or when number of args unknown in advance (if at beginning, no positional or partial matching)
args(paste) # operates on unknown sets of character vectors
```

```
## function (..., sep = " ", collapse = NULL, recycle0 = FALSE)
## NULL
```

```r
# function as an argument
some_function <- function(func){
  func(2, 4) # returns result of function with 2,4 arguments
}
some_function(mean) # returns mean of 2,4
```

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

```r
# Anonymous function (chaos)
evaluate  <- function(func, dat){
  func(dat)
}
evaluate(function(x){x+1}, 6) # creates a function when calling evaluate to add 1
```

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

```
# create a binary operation
"%mult_add_one%" <- function(left, right){
  left * right + 1
}
4 %mult_add_one% 5
```

```
## [1] 21
```

Lexical Scoping

```
make.power <- function(n) {
  pow <- function(x) {
    x^n
  }
  pow
}

cube <- make.power(3)
square <- make.power(2)
cube(3)
```

```
## [1] 27
```

```
square(3)
```

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

```
# Scoping - environments
search()# provides list of environments
```

```
## [1] ".GlobalEnv"        "package:stats"     "package:graphics"
## [4] "package:grDevices" "package:utils"     "package:datasets"
## [7] "package:methods"   "Autoloads"         "package:base"
```

```
ls(environment(cube)) # object names in function environment, same for square
```

```
## [1] "n"   "pow"
```

```
get("n",environment(cube)) # values in function environment, changes for square
```

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

R Packages

- Repositories: CRAN, BioConductor (bioinformatics), GitHub

- Search: https://www.rdocumentation.org/
- Base packages: utils, stats, datasets, graphics, grDevices, grid, methods, tools, parallel, compiler, splines, tcltk, stats4.

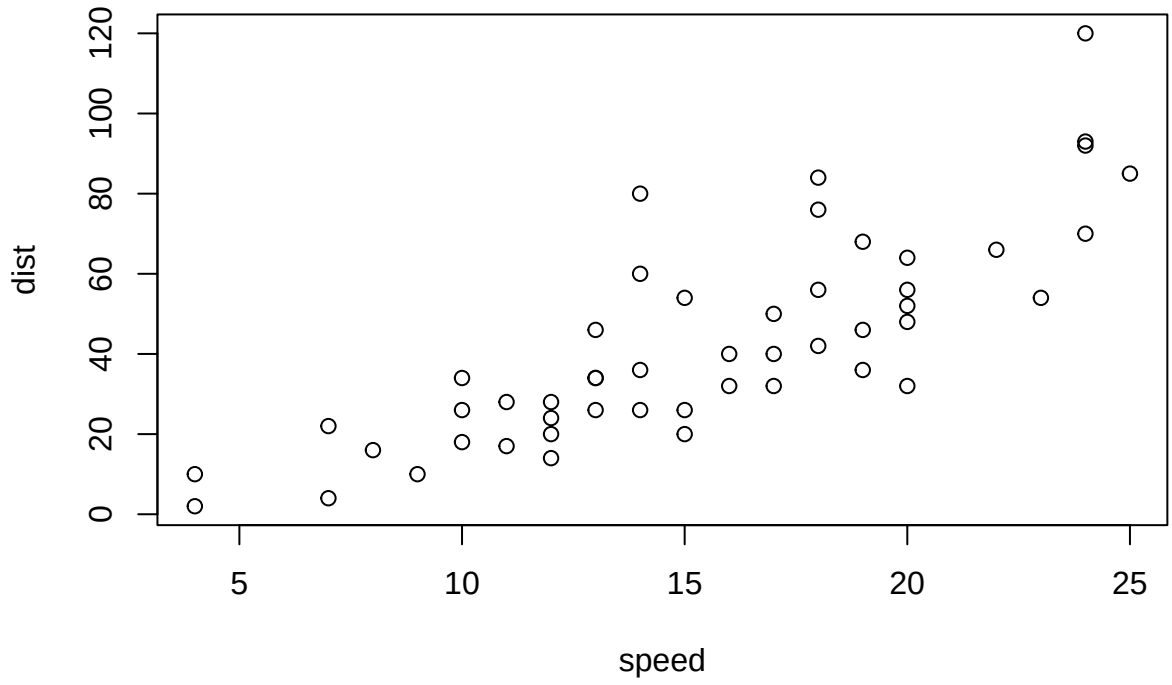
- Recommended packages: boot, class, cluster, codetools, foreign, KernSmooth, lattice, mgcv, nime, rpart, survival, MASS, spatial, nnet, Matrix.

```
# Install from CRAN:
#   install.packages("ggplot2", repos = "http://cran.us.r-project.org") #install
#   install.packages(c("labeling","tibble"), repos = "http://cran.us.r-project.org") #multiple

# Install from Bioconductor
#   install.packages("BiocManager", repos = "https://bioconductor.org/biocLite.R")
#   BiocManager::install(c("GenomicFeatures", "AnnotationDbi")) #install package

# Install from GitHub (need package, author name)
#   install.packages("devtools", repos = "http://cran.us.r-project.org") #only once
#   library(devtools)
#   install_github("author/package") #installs package

# library(ggplot2)# Load package, careful of dependencies
# installed.packages() #check installed packages
# library() #alternate
# old.packages(repos = "http://cran.us.r-project.org") #check packages to update
# update.packages(repos = "http://cran.us.r-project.org") #update all packages
# install.packages("ggplot2") #to update single package
# detach("package:ggplot2", unload=TRUE) #unload function
# remove.packages("ggtree") #remove package
# help(package = "ggplot2") #package info
# browseVignettes("ggplot2") #extended help files
```

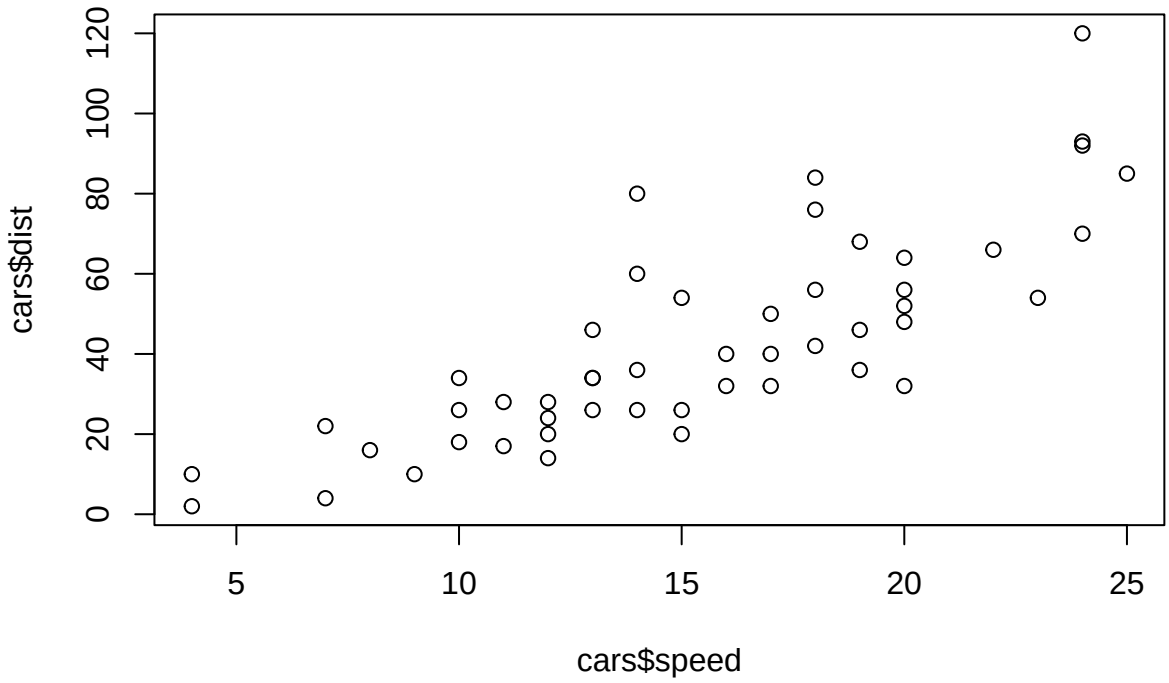
Graphics

http://www.ling.upenn.edu/~joseff/rstudy/week4.html

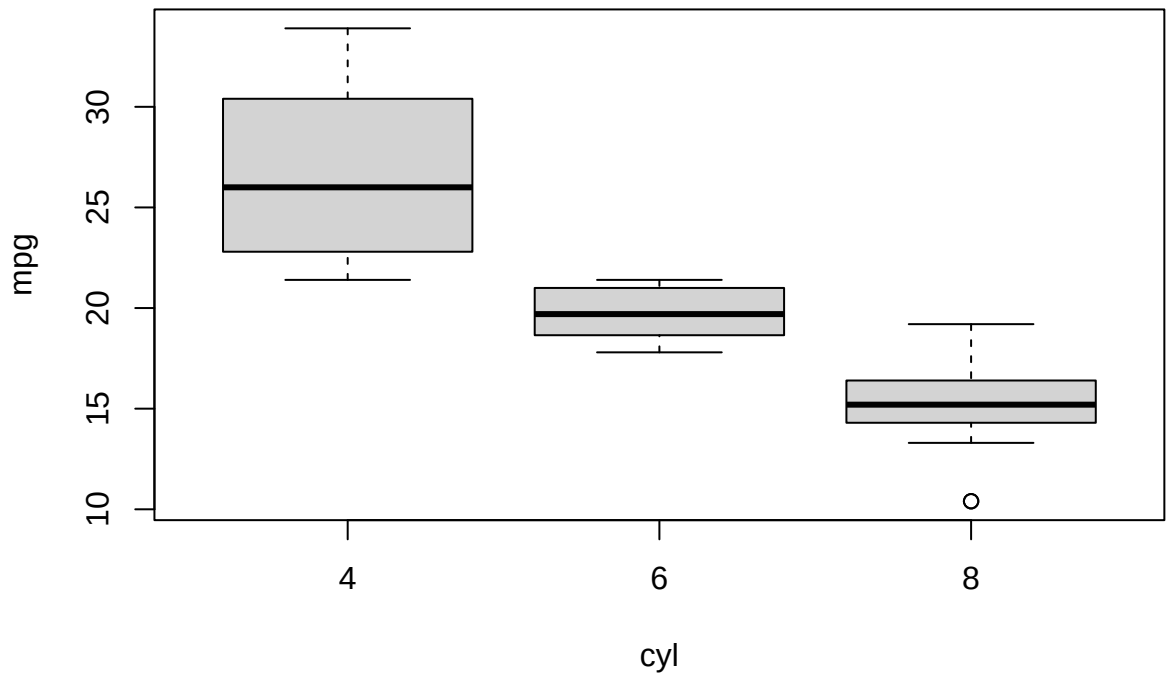
```
# Start by getting sense of the data: dim(), names(), head(), tail() and summary().

# Scatterplot
data(cars)
plot(cars) # generates scatterplot with two columns against each other
```



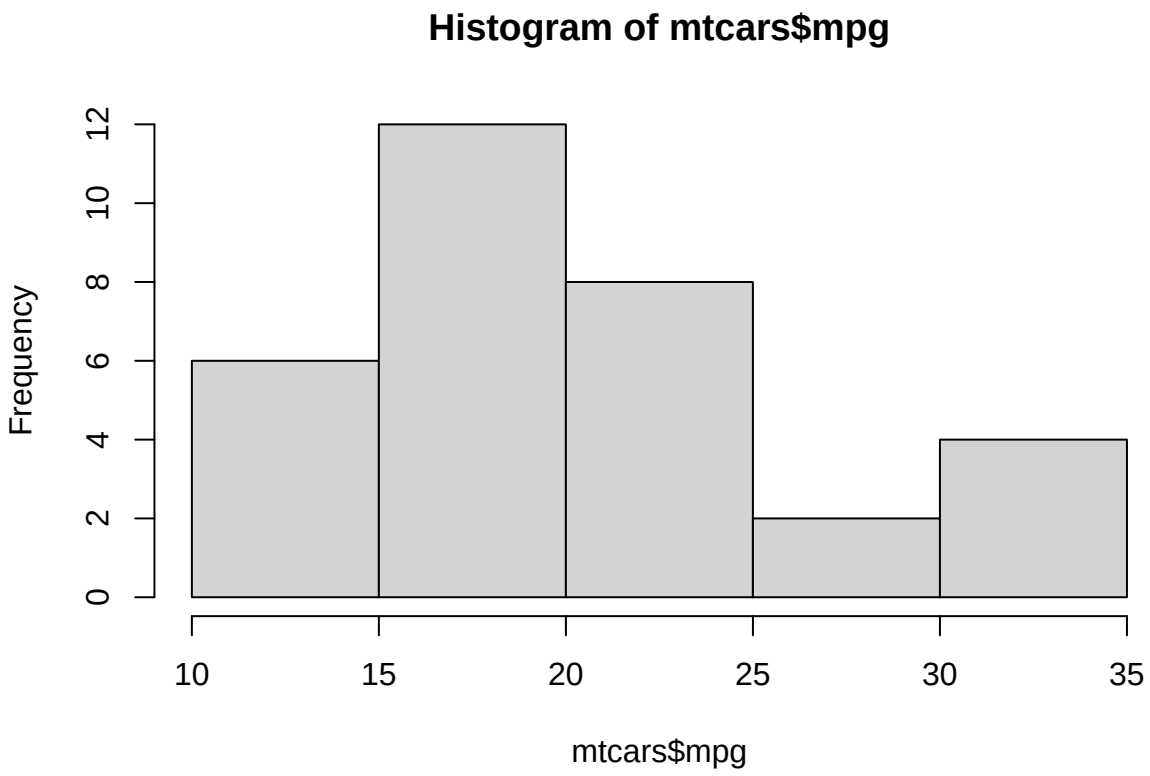

```
plot(x = cars$speed, y = cars$dist) # explicit, can also pass plot(dist ~ speed, cars). Args: x, y = NU
```




```r
# Boxplot
data(mtcars)
boxplot(mpg ~ cyl, data = mtcars) # generate relationship between mpg(x) and cyl(y) from mtcars
```




```r
# Histogram
hist(mtcars$mpg) # generate a histogram of vector
```



R Profiler and Optimization

- Systematic way to examine time spent in various part of the program. Useful to optimize the code.
- DON'T PREMATURELY OPTIMIZE
- Measure, not guess, data on what needs to be optimized.
- User time: computer experienced, may be greater if multiple cores/processors (accessible in multi-threaded BLAS libraries). Elapsed time: wall-clock time, may be greater if other computing tasks.

```
system.time(read.csv("hw1_data.csv")) # returns seconds to execute, if error then seconds to error. Wra
```

```
##    user  system elapsed
##       0       0       0
```

```
data(mtcars)
Rprof() # track function call stack at intervals (def = 0.02 sec), time spent in functions.

fit <- lm(mtcars$mpg ~ mtcars$cyl)

Rprof(NULL)

summaryRprof() # makes Rprof readable, tabluates, time in each function
```

```
## $by.self
## [1] self.time  self.pct   total.time total.pct
## <0 rows> (or 0-length row.names)
##
## $by.total
## [1] total.time total.pct  self.time  self.pct
## <0 rows> (or 0-length row.names)
##
## $sample.interval
## [1] 0.02
##
## $sampling.time
## [1] 0
```

```
# $by.total - divides time spent per function by total run time
# $by.self - same as by.total but first subtracts time spent in function above in call stack. Helps tar
```