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.

```
# 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
x <- vector("numeric", length = 10) # create vector of one type, args: class, length
x <- c(1,2,3,4) # creates vector of common denominator class with given values
x <- 1:20 # vector sequence of 20 elements 1-20
x <- pi:10 # will not exceed 10, start from pi, increment by 1
x <- 15:1 # increment -1
x <- seq(1,20) # same as :
```

```r
x <- seq(0,10,by=0.5) # to change increment
x <- seq(5,10,length=30) # to not set increment but number of numbers
x <- seq_along(x) # vector of same length 1:length(x)
x <- rep(10, times = 4) # repeats 10 4 times in vector
x <- rep(c(0, 1, 2), times = 10) # repeats sequence of vector 10 times. Arg each can be used to repeat

# 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
# 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
# vectorized operation
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
```

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

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

```
# 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

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

```
##   [1] "1"   "2"   "3"   "4"   "5"   "6"   "7"   "8"   "9"   "10"  "11"  "12"
##  [13] "13"  "14"  "15"  "16"  "17"  "18"  "19"  "20"  "21"  "22"  "23"  "24"
##  [25] "25"  "26"  "27"  "28"  "29"  "30"  "31"  "32"  "33"  "34"  "35"  "36"
##  [37] "37"  "38"  "39"  "40"  "41"  "42"  "43"  "44"  "45"  "46"  "47"  "48"
##  [49] "49"  "50"  "51"  "52"  "53"  "54"  "55"  "56"  "57"  "58"  "59"  "60"
##  [61] "61"  "62"  "63"  "64"  "65"  "66"  "67"  "68"  "69"  "70"  "71"  "72"
##  [73] "73"  "74"  "75"  "76"  "77"  "78"  "79"  "80"  "81"  "82"  "83"  "84"
##  [85] "85"  "86"  "87"  "88"  "89"  "90"  "91"  "92"  "93"  "94"  "95"  "96"
##  [97] "97"  "98"  "99"  "100" "101" "102" "103" "104" "105" "106" "107" "108"
## [109] "109" "110" "111" "112" "113" "114" "115" "116" "117" "118" "119" "120"
## [121] "121" "122" "123" "124" "125" "126" "127" "128" "129" "130" "131" "132"
## [133] "133" "134" "135" "136" "137" "138" "139" "140" "141" "142" "143" "144"
## [145] "145" "146" "147" "148" "149" "150" "151" "152" "153"
```

```
colnames(x) # get and set row names
```

```
## [1] "Ozone"   "Solar.R" "Wind"    "Temp"    "Month"   "Day"
```

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

```
## [1] 153
```

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

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

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

```
##        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
##   [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 12.0   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
##  [25,]    NA      66 16.6   57     5  25
##  [26,]    NA     266 14.9   58     5  26
##  [27,]    NA      NA  8.0   57     5  27
##  [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
##  [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  8.0   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  8.0   75     6  25
```

```
##  [57,]    NA     127  8.0   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  8.0   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  8.0   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
##  [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  4.0   89     8   7
## [100,]    89     229 10.3   90     8   8
## [101,]   110     207  8.0   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  8.0   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  8.0   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
## [150,]    NA     145 13.2   77     9  27
## [151,]    14     191 14.3   75     9  28
## [152,]    18     131  8.0   76     9  29
## [153,]    20     223 11.5   68     9  30
```

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

```
## NULL
```

```
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
```

```r
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

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

# 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] 35
```

```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 4439.775 days
```

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

```
## Time difference of 4439.775 days
```

Basic R Functions

```r
# managing working directory and work space
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 2024-12-20 23:35:19 2024-12-20 23:35:19
##                         atime exe
## mytest.R 2024-12-20 23:35:19  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.html"         "Course Notes.Rmd"
##  [9] "CourseraDataScience.Rproj" "hw1_data.csv"
## [11] "pollutantmean.R"           "specdata"
## [13] "specdata.zip"              "testdir"
```

```r
ls() # prints the objects in work space
```

```
##  [1] "birthday"    "cTConvert"   "currentTime" "datestring"  "f"
##  [6] "m"           "msg"         "tf"          "timedefined" "today"
## [11] "vect"        "x"           "y"
```

```r
rm(list=ls()) # clear workspace
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
```

```r
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
```

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

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

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

```
## NULL
```

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

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

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

# Basic Functions
# add numbers, vectors (element-by-element or recycling)
x <- 5 + 7 # basic arithmetic operations all work +, -, *, /, ^, %% (modulus). NA in expression returns
sqrt(4) # square root
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```r
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
str(unclass(as.POSIXlt(Sys.time()))) # prints list clearly
```

```
## List of 11
##  $ sec   : num 20.2
##  $ min   : int 35
##  $ hour  : int 23
##  $ mday  : int 20
##  $ mon   : int 11
##  $ year  : int 124
##  $ wday  : int 5
##  $ yday  : int 354
##  $ isdst : int 0
##  $ zone  : chr "EST"
##  $ gmtoff: int -18000
##  - attr(*, "tzone")= chr [1:3] "" "EST" "EDT"
##  - attr(*, "balanced")= logi TRUE
```

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

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

```r
x # auto-prints
```

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

```r
# in console, press Tab for auto-completion

# Random number generation
y <- rnorm(1000) # generate vector of 1000 numbers that are standard normal distribution.
y <- sample(1:6,3) # random selection of 3 elements from array
ints <- sample(10) # random sample of integers from 1 to 10 without replacement
coin <- rbinom(1,1,0.5) # simulating coin flip

# Functions on Objects
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] 1
```

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

```
## NULL
```

```r
identical(x,y) # check if objects are identical
```

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

```r
c(0.5,0.8,10) # creates a vector of a certain class otherwise coercion
```

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

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

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

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

```
##  [1] -2.404484052 -2.448906848 -0.581911586  1.439227068  0.137679401
##  [6] -3.045262094  1.822774773 -5.631036743 -1.904609765  1.964216542
## [11]  3.130819738 -0.005639412  6.103161865 -4.205879724 10.478585372
## [16]  0.782145248  2.472571600  2.565095411 -4.256154707 -3.617169248
```

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

```
##  [1] -0.2404484052 -0.2448906848 -0.0581911586  0.1439227068  0.0137679401
##  [6] -0.3045262094  0.1822774773 -0.5631036743 -0.1904609765  0.1964216542
## [11]  0.3130819738 -0.0005639412  0.6103161865 -0.4205879724  1.0478585372
## [16]  0.0782145248  0.2472571600  0.2565095411 -0.4256154707 -0.3617169248
```

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

```
##  [1]   7.78389189  -4.08949281 -10.58736818  -0.29459846   0.56393883
##  [6]   1.71990831   5.04512323   4.11250230   0.01698894  -1.47567121
```

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

```
##  [1]  0.389194594 -0.204474641 -0.529368409 -0.014729923  0.028196941
##  [6]  0.085995416  0.252256161  0.205625115  0.000849447 -0.073783560
```

```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
# 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
```

```
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
```

```
# 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"
```

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

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

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

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

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

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

```r
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
```

```r
# 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] 4
## [1] 5
## [1] 4
## [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 - 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] 0.2930604
```

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

```
## [[1]]
## [1] 8.665202
##
## [[2]]
## [1] 4.287950 9.624047
##
## [[3]]
## [1] 1.652641 7.014087 7.330731
##
## [[4]]
## [1] 0.7127642 2.0038417 7.5140810 8.9783997
```

```
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] -0.2431055
```

```
# 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 -0.2431055
```

```
# 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]  0.0462279883 -0.0160286154 -0.2061569007 -0.2133892283 -0.1542309244
##  [6]  0.0005256159  0.1604264196 -0.0953182149 -0.2551506170  0.3393495895
```

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

```
##  [1] -0.35746083 -2.45663755  1.28506726  2.78341088 -5.33387330 -4.29321725
##  [7] -2.94856573 -7.50083548 -3.51860790 -2.93731900  0.03583825  0.70646352
## [13] 11.26088068  5.46759470  2.46717396 -0.25932869  3.80730577 -0.38867388
## [19] -1.44008330 -4.25402987
```

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

```
##  [1] -0.035746083 -0.245663755  0.128506726  0.278341088 -0.533387330
##  [6] -0.429321725 -0.294856573 -0.750083548 -0.351860790 -0.293731900
## [11]  0.003583825  0.070646352  1.126088068  0.546759470  0.246717396
## [16] -0.025932869  0.380730577 -0.038867388 -0.144008330 -0.425402987
```

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

```
##  [1]  0.92455977 -0.32057231 -4.12313801 -4.26778457 -3.08461849  0.01051232
##  [7]  3.20852839 -1.90636430 -5.10301234  6.78699179
```

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

```
##  [1]  0.0462279883 -0.0160286154 -0.2061569007 -0.2133892283 -0.1542309244
##  [6]  0.0005256159  0.1604264196 -0.0953182149 -0.2551506170  0.3393495895
```

```
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]
## 25% -0.6304134 -0.8615475 -0.3091682 0.03938052 -1.1483544 -0.91337318
## 75%  0.5228807  0.2243711  1.0030134 0.90362867  0.1762393  0.05991247
##           [,7]        [,8]         [,9]      [,10]      [,11]     [,12]
## 25% -0.9628354 -1.72697274 -1.004961718 -0.8421042 -0.5247483 -0.748542
## 75%  0.8205191  0.07586177  0.004197088  0.3251059  1.0889345  1.159389
##         [,13]     [,14]      [,15]      [,16]     [,17]     [,18]      [,19]
## 25% 0.4577822 0.2401517 -0.2447379 -0.6862357 -0.192493 -1.077320 -0.4446361
## 75% 1.5269876 0.9608696  0.7920313  0.5257245  0.838937  1.244578  0.4539559
##           [,20]
## 25% -0.80607698
## 75%  0.02785533
```

```
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,]  0.4173641 -0.2175200
## [2,] -0.1189923  0.2792663
```

```r
# 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
## -0.5007504  0.3932776  0.8316159
```

```r
# 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
```

```r
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] 5.022312 2.040476 4.190005 5.305180 5.670255
```

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

```
## [[1]]
## [1] 2.517758
##
## [[2]]
## [1] 3.671419 2.193412
##
```

```
## [[3]]
## [1] 6.177710 4.516818 1.249950
##
## [[4]]
## [1] 3.6522265 2.6659998 8.2017960 0.2723429
##
## [[5]]
## [1] 5.034820 5.310769 4.096047 1.838986 3.516144
```

```
# 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]  0.2649714 -0.2445456  0.9436649  0.8618395 -0.5005153  1.3826544
##  [7] -1.0584796  1.1743467  0.7417109  0.7710143
##
## $`2`
##  [1] 0.239425116 0.251794757 0.631922012 0.335526289 0.409599698 0.006501552
##  [7] 0.478866204 0.029563925 0.202711037 0.715982011
##
## $`3`
##  [1] -0.73281532  1.37326421  2.20456592  0.01895387  0.45908250  0.28796139
##  [7]  2.30299399 -0.26814742  2.10482627  2.34780831
```

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

```
## $`1`
## [1] 0.4336662
##
## $`2`
## [1] 0.3301893
##
## $`3`
## [1] 1.009849
```

```
# 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
```

```
# 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
```

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

```
## $`1.1`
## [1] -0.143666 -0.371580
##
## $`2.1`
## numeric(0)
##
## $`1.2`
## [1] -0.2004512 -1.3311664
##
## $`2.2`
## numeric(0)
##
## $`1.3`
## [1] 0.7043021
##
## $`2.3`
## [1] 0.2645423
##
## $`1.4`
## numeric(0)
##
## $`2.4`
## [1] -0.1145079 -1.4622955
##
## $`1.5`
## numeric(0)
##
## $`2.5`
## [1] -0.2972190  0.3841342
```

```
# 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] -0.0007392166
```

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

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

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

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

```r
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] 7.330299 7.268177 9.317018 8.278727 6.756955 5.793709 9.949637 7.891419
##   [9] 8.339102 9.397801 7.269948 6.970963 6.039676 7.236905 8.025396 8.282245
##  [17] 8.858824 7.606713 7.111068 7.889009 8.231957 8.324289 8.528621 7.359797
##  [25] 9.513360 7.695836 6.684896 9.314731 8.462481 7.107971 6.514310 6.128432
##  [33] 9.120731 7.776357 7.124967 7.701372 6.753800 8.456006 7.426689 9.967467
##  [41] 9.252508 7.272151 8.097153 8.565700 7.913534 8.368916 7.307506 8.008220
##  [49] 8.087189 9.794704 5.449239 9.137585 6.921721 8.287041 9.072561 9.527693
##  [57] 7.203231 7.195561 7.223752 7.696879 8.178516 7.804960 9.360901 7.599586
##  [65] 8.219411 7.322678 8.884354 7.890126 7.642541 7.541610 8.319940 9.880537
##  [73] 7.755222 9.390619 7.870653 7.212679 8.327195 6.696257 8.737004 8.179578
##  [81] 8.448653 8.009305 7.488703 7.895802 7.448258 7.338577 8.470568 8.363580
##  [89] 9.037386 5.070300 8.483929 7.966914 7.965463 5.565573 7.909624 7.234787
##  [97] 7.549609 8.008979 8.616353 7.071043
```

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

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

```
##   [1]  5.993742  7.665344  6.999558  8.122107 10.629535 11.875359 11.805375
##   [8]  5.361472  6.103784  8.255720  9.553411 10.975108  8.887274 11.590359
##  [15]  7.048433  6.268641  6.639802  9.868776 11.795788  9.939516 11.980096
##  [22]  4.852225  8.056356  8.529368  8.775778  8.836328 10.740529 11.708235
##  [29]  5.594125  6.022559  8.087104  9.600488  9.134798  9.470025 13.163140
##  [36]  5.490199  5.859300  5.986829  8.072877 10.284797 10.634407 12.689654
##  [43]  5.839335  7.764957  7.172407  8.360172  8.346045 10.101801 11.791059
##  [50]  5.644561  7.075899  7.612891  9.573265 10.309801 12.580386 12.104463
##  [57]  6.597380  6.462992  8.195707  9.570558  9.870160 11.573129 12.183641
##  [64]  5.697176  6.308225  7.366879  8.403315  9.878625 10.376407 12.696935
##  [71]  4.521981  8.399881  7.003723 10.801219 11.463496 11.965541 11.372212
##  [78]  6.273711  7.820579  8.984447  7.040694 10.661246 11.573259 12.656363
##  [85]  6.853243  6.846838  6.472861  8.407311  8.661118 10.444911 13.758408
##  [92]  6.556139  6.600362  7.479234  9.645503  9.322073 12.583283 11.671188
##  [99]  8.275595  5.979823
```

```r
# 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] 1.029819
```

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

```
## [1] 1.029819
```

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

```
## [1] 1.029819
```

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

```
## [1] 1.029819
```

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

```
## [1] 1.029819
```

```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
```

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

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

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

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

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

```r
square(3)
```

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

```r
# 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"
```

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

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

```r
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.

```r
# 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
```

Data in R

```
# Pull all file names from a directory into a character vector
files_full <- list.files("specdata", full.names=TRUE)

# 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
```

```
# 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"
```

```r
#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"
```

```r
# 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

# 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
```