

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 :
```

```

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
```

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

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

```

# 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

```

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

```

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

```

```

# 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 model
```

```
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, length
```

```
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
```

```

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] 29

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"

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

## [1] "Thursday"

months(birthday) # return month on date or time

## [1] "January"

quarters(birthday) # return quarter of date or time

## [1] "Q1"

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

## Time difference of 4438.812 days

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

## Time difference of 4438.812 days

```

Basic R Functions

```
# 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
```

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

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

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

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

```
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 00:29:19 2024-12-20 00:29:19
##                                     atime exe
## mytest.R 2024-12-20 00:29:19 no
```

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

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

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

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

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

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

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

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



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

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

```
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
```

```
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 ?`:`
```

```
# 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
```

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

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

```
# 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
```

```
# Character functions
```

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

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

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

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

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

```
## List of 11
## $ sec   : num 20
## $ min   : int 29
## $ hour  : int 0
## $ 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
```

```
# 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
```

```
# 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"
```

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

```
## NULL
```

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

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

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

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

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

```
## NULL
```

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

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

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

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

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

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

```
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
```

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

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

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

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

```
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
```

```
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
```

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

```
## 25%
```

```
## 4
```

```
# Matrix function
```

```
x <- matrix(rnorm(200), 20, 10)
```

```
rowSums(x) # vector of sum of rows
```

```
## [1] -0.8819888 1.7747369 -2.3887198 3.8342416 -1.1630186 3.6243541
```

```
## [7] -1.0879333 3.9135224 6.1126977 0.2609308 -0.2177518 -3.3266591
```

```
## [13] 3.6248183 1.4678541 -6.3949753 -1.9793532 1.3027306 1.2188769
```

```
## [19] -2.3242100 -2.1757114
```

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

```
## [1] -0.08819888 0.17747369 -0.23887198 0.38342416 -0.11630186 0.36243541
```

```
## [7] -0.10879333 0.39135224 0.61126977 0.02609308 -0.02177518 -0.33266591
```

```
## [13] 0.36248183 0.14678541 -0.63949753 -0.19793532 0.13027306 0.12188769
```

```
## [19] -0.23242100 -0.21757114
```

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

```
## [1] 7.1657070 1.6431573 2.9261624 6.4542778 -6.8058463 -3.9302532
```

```
## [7] -1.8062928 2.7309542 -2.6166728 -0.5667515
```

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

```
## [1] 0.35828535 0.08215787 0.14630812 0.32271389 -0.34029231 -0.19651266
```

```
## [7] -0.09031464 0.13654771 -0.13083364 -0.02833758
```

```
# 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
```

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

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

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

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

```
# 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"
```

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



```
## [1] 6
## [1] 5
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [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 - useful for looping in the command line

# 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] -0.3697327
```

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

```
## [[1]]
## [1] 1.672294
##
## [[2]]
## [1] 2.782296 3.536589
##
```

```
## [[3]]
## [1] 5.244323 8.578589 7.664268
##
## [[4]]
## [1] 2.9436306 0.6509118 0.5810593 5.2774496
```

```
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] 0.08419348
```

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

# 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] -0.33128198 0.03131888 -0.23666358 -0.02970411 -0.33372594 -0.09246837
## [7] -0.36819664 0.02449333 -0.07785099 -0.18101808
```

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

```
## [1] -1.4077387 1.5533325 -3.8848452 -7.2182423 5.4590458 -4.5560558
## [7] -3.1294259 -2.3127829 -0.6556856 -1.5432140 -2.7350947 -1.6431589
## [13] 1.5307028 -6.3627220 -0.3124300 7.5730751 -8.0894809 0.7173517
## [19] 0.6571952 -5.5417756
```

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

```
## [1] -0.14077387 0.15533325 -0.38848452 -0.72182423 0.54590458 -0.45560558
## [7] -0.31294259 -0.23127829 -0.06556856 -0.15432140 -0.27350947 -0.16431589
## [13] 0.15307028 -0.63627220 -0.03124300 0.75730751 -0.80894809 0.07173517
## [19] 0.06571952 -0.55417756
```

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

```
## [1] -6.6256396 0.6263775 -4.7332715 -0.5940823 -6.6745187 -1.8493673
## [7] -7.3639329 0.4898666 -1.5570198 -3.6203615
```

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

```
## [1] -0.33128198 0.03131888 -0.23666358 -0.02970411 -0.33372594 -0.09246837
## [7] -0.36819664 0.02449333 -0.07785099 -0.18101808
```

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

```
##           [,1]           [,2]           [,3]           [,4]           [,5]           [,6]
## 25% -0.6449820 -0.7279216 -0.825090205 -1.3105926 0.1890075 -0.45142341
## 75% 0.4921431 0.9582862 0.004136585 -0.2933289 0.8076560 0.08699773
##           [,7]           [,8]           [,9]           [,10]           [,11]           [,12]           [,13]
## 25% -0.7582134 -0.4166352 -1.1830048 -0.7904071 -1.208965 -0.9647514 -0.6143917
## 75% 0.1481705 0.1131617 0.9685584 0.4470652 0.342960 0.2104971 1.1603363
##           [,14]           [,15]           [,16]           [,17]           [,18]           [,19]
## 25% -1.1524362 -0.7205686 -0.1460886 -1.4194700 -0.5734712 -0.3928982
## 75% -0.3025489 0.7550855 1.9039727 -0.1915858 0.8406532 0.4951011
##           [,20]
## 25% -1.1583882
## 75% 0.4044113
```

```
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 = 3)
```

```
##           [,1]           [,2]
## [1,] 0.3631106 -0.007265992
## [2,] 0.2290802 -0.122994593
```

```
# tapply - apply function over subset of a vector. args: X is vector, INDEX is factor/list factors vector
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
## 0.1083562 0.4893649 1.0286267
```

```
# 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
```

```
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] 1.9776230 0.5522527 1.0400631 3.8310498 5.3996209
```

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

```
## [[1]]
## [1] 3.127712
##
## [[2]]
## [1] 3.746763 3.248756
##
## [[3]]
## [1] -1.545892 3.557573 3.864755
##
## [[4]]
## [1] 2.350157 6.675283 3.415370 4.751896
##
## [[5]]
## [1] 6.557293 6.867938 5.336184 6.005077 2.617570
```

```
# 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] 0.29817623 0.05723172 -0.18389557 -0.46081940 -1.09332563 -0.93862384
## [7] -0.23975396 1.58129013 0.42444590 -0.18894022
##
```

```
## $'2'
## [1] 0.03150495 0.69569963 0.33804092 0.25334736 0.88631689 0.02884114
## [7] 0.45265420 0.66054440 0.74759966 0.45725463
##
## $'3'
## [1] 1.7205866 1.9374183 1.4477518 0.6930081 2.6167641 0.1938291 2.2138445
## [8] 1.9644152 1.7364775 0.1827520
```

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

```
## $'1'
## [1] -0.07442146
##
## $'2'
## [1] 0.4551804
##
## $'3'
## [1] 1.470685
```

```
# 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 into
```

```
##           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] -0.4003284 1.6925184
##
## $'2.1'
## numeric(0)
##
## $'1.2'
## [1] 1.6059892 -0.1397834
##
## $'2.2'
## numeric(0)
##
```

```
## $'1.3'
## [1] -0.04882373
##
## $'2.3'
## [1] -0.5737636
##
## $'1.4'
## numeric(0)
##
## $'2.4'
## [1] -0.4921061 -0.4497006
##
## $'1.5'
## numeric(0)
##
## $'2.5'
## [1] -0.20972823  0.04782447
```

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

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

```
## [1] -0.08182836
```

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] 7.704082 8.232904 6.850951 8.702282 7.401276 7.940254 9.859489
## [8] 7.756051 9.037192 8.111815 6.770795 7.323168 8.512015 9.771271
## [15] 9.313652 7.358123 7.156966 7.977527 8.209639 6.234496 8.694695
## [22] 7.609760 6.832614 9.375318 9.156337 8.460443 6.726447 8.082126
## [29] 5.126210 8.342042 8.200342 8.652139 9.367762 7.251905 8.377801
```

```
## [36] 4.964745 8.043805 8.013741 9.430884 5.651612 9.257331 8.084639
## [43] 9.041420 7.448916 9.228576 7.514977 7.366532 6.670365 8.466280
## [50] 9.288944 9.135785 7.118835 8.143962 6.115324 10.422906 6.884116
## [57] 6.684035 7.961339 8.161960 7.455634 5.182882 8.650757 8.934914
## [64] 8.607348 7.613738 7.103890 7.977454 7.193158 10.075298 7.299324
## [71] 7.790251 6.285707 7.394998 8.278660 8.303436 8.706785 7.157620
## [78] 7.095322 7.224526 8.504852 8.173011 7.637584 6.998574 8.015838
## [85] 9.580592 7.384305 7.052117 7.615735 7.500367 7.891597 7.939247
## [92] 8.511593 7.912969 7.259374 8.939227 8.257441 8.136576 7.231978
## [99] 9.479334 6.826572
```

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

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

```
## [1] 5.739309 4.544662 8.832782 8.866066 9.838152 10.109572 12.433109
## [8] 4.208221 7.799510 8.182841 9.359853 9.595634 9.751589 11.611704
## [15] 5.119842 5.833274 8.567129 9.939996 10.910448 10.853816 12.432649
## [22] 3.612077 7.247725 8.552485 10.564164 7.836834 10.689614 12.088645
## [29] 6.452994 6.116485 7.851763 8.996446 9.979539 11.256389 10.160798
## [36] 4.407228 7.717416 7.226549 8.749156 10.277437 10.241883 12.269562
## [43] 6.757464 8.853328 8.152577 9.955932 9.082148 11.545607 10.655808
## [50] 6.282655 4.574859 8.257775 9.089383 9.575461 11.168200 12.557171
## [57] 6.869095 7.557900 8.157421 10.509107 9.958380 11.716737 10.910152
## [64] 7.875952 8.186061 9.282413 8.358274 10.746425 11.661816 11.357011
## [71] 6.786386 6.671684 8.405859 8.631223 9.749923 11.998144 12.146939
## [78] 6.032867 6.317683 7.976086 7.959009 10.363291 10.427227 11.280545
## [85] 6.398805 5.669763 8.002947 9.638303 8.272610 10.571218 13.355766
## [92] 5.034378 7.021786 7.874661 10.207478 9.939140 9.464969 11.402681
## [99] 6.134252 7.415739
```

```
# 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)
  means
}
```

```
# 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 position
# 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] 0.939509
```

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

```
## [1] 0.939509
```

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

```
## [1] 0.939509
```

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

```
## [1] 0.939509
```

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

```
## [1] 0.939509
```

```
# 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
```

```
# 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
```

```
# 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
```

```
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
```

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 = ",", nrow = 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

# 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"

x[1:4] # sequence of num index

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

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

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

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

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

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

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

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

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

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

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

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

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

## [1] 1 2 3 4

```

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

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

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

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

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

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

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

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

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

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

```
# 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
```

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

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

```
# 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
```

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

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



```
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
```

```
y[good]
```

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

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

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

```
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