Code_Lib_Manipulate

Ruhika Chatterjee

2024-12-07

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

R Data Types

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

Atomic Data

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

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

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

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

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

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

complex Data Types

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Data Frames

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

Date and Time Data Types

```r
# useful for time-series data (temporal changes) or other temporal info
# lubridate package by Hadley Wickham
# 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] 48
```

```
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. daylightsavings, timezones, l
```

```
## Time difference of 4476.284 days
```

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

```
## Time difference of 4476.284 days
```

```
rm(list=ls())
```

Basic R Functions

Functions and Operations

```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
LETTERS # predefined character vector of capital letters
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
##  [1]  4.71446853 -3.98952921 -3.58777702 -0.36696075  3.54259274 -2.66588110
##  [7]  1.11244809 -0.25469885 -0.06099387  2.15840209  1.30552140 -5.20018338
## [13]  0.69239529 -3.21521989  2.19741164  3.17234468 -0.24893580  1.74442848
## [19]  1.63127556 -0.01030102
```

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

```
##  [1]  0.471446853 -0.398952921 -0.358777702 -0.036696075  0.354259274
##  [6] -0.266588110  0.111244809 -0.025469885 -0.006099387  0.215840209
## [11]  0.130552140 -0.520018338  0.069239529 -0.321521989  0.219741164
## [16]  0.317234468 -0.024893580  0.174442848  0.163127556 -0.001030102
```

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

```
##  [1] -3.1421371 -5.0730665  5.4745981  1.5773425 -1.0779831 -2.5687226
##  [7]  7.7502159 -0.5971014  0.7494149 -0.4217530
```

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

```
##  [1] -0.15710686 -0.25365333  0.27372991  0.07886712 -0.05389916 -0.12843613
##  [7]  0.38751080 -0.02985507  0.03747075 -0.02108765
```

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

```
##             [,1]        [,2]        [,3]         [,4]        [,5]        [,6]
##  [1,]  0.5373329  0.11664366  0.12508470 -0.263920679 -0.25459203  0.29756175
##  [2,] -0.1752860 -0.08511148 -0.06851201  0.221004266  0.06535676  0.25639183
##  [3,]  0.2451309  0.84761061 -0.12666496  0.007679055  0.13757643  0.63587759
##  [4,] -0.3005111 -0.40435822 -0.31036865 -0.976470459  1.03083439  0.04185183
##  [5,] -0.6136179  0.17903153 -0.14907043  0.336497969  0.35656630  0.31417533
##  [6,] -0.1114019 -0.03877566  0.38939018  0.475376545 -0.31883791 -0.51838537
##  [7,] -0.5790278 -0.24259157  0.12232110 -0.144181484  0.86391963 -0.21697547
##  [8,] -0.8208051 -0.57589325  0.23780404  0.422867599  0.61594031 -0.42248004
##  [9,] -0.7709562 -0.70086413  0.15543899 -0.267546209  1.38309170 -0.47314019
## [10,] -0.4767222 -0.29958601 -0.15907598  0.210751320  0.85989339 -0.09818059
##              [,7]        [,8]       [,9]       [,10]
##  [1,] -0.20557681 -0.36217545  0.1891460 -0.30566921
##  [2,]  0.29700030 -0.03792023  0.0866690  0.12577389
##  [3,] -0.37962659  0.01106827  0.1692910  0.26355143
##  [4,]  0.02763445  0.28747068  0.9508545  0.21114869
##  [5,] -0.03030338  0.05094483 -0.2634879  0.67173273
##  [6,]  0.20883818 -0.14372369 -0.2878725 -0.03887496
##  [7,]  0.37425146  0.18654161 -0.2119517  0.12320275
##  [8,]  0.42065129  0.28260475 -0.2079188  0.18860657
##  [9,]  0.53297159  0.55635005  0.3335213  0.59956711
## [10,]  0.25276901 -0.02911616  0.2649991  0.52357042
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
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
```

```
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
```

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

```
## List of 11
##  $ sec   : num 56.4
##  $ min   : int 48
##  $ hour  : int 11
##  $ mday  : int 26
##  $ mon   : int 0
##  $ year  : int 125
##  $ wday  : int 0
##  $ yday  : int 25
##  $ isdst : int 0
##  $ zone  : chr "EST"
##  $ gmtoff: int -18000
##  - attr(*, "tzone")= chr [1:3] "" "EST" "EDT"
##  - attr(*, "balanced")= logi TRUE
```

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

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

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

```
##  num [1:100] 3.723 -3.221 0.437 4.194 -4.932 ...
```

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

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

```
# 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
```

Attributes of objects

```
x <- c(0.5,105,10,0.1,2)
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] 5
```

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

```
## NULL
```

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

```
## 96 bytes
```

```
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
```

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

```
## NULL
```

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

```
## NULL
```

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

```
## NULL
```

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

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

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

```
## NULL
```

```
# 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
```

Indexing, Subsetting, and Dealing with NAs

```
# 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
```

```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
```

Data Tables (not Frames)

- Package, faster and more memory efficient
- Inherets from data.frame (all functions), written in C, faster at sub-setting, grouping, and updating
- http://stackoverflow.com/questions/13618488/what-you-can-do-with-data-frame-that-you-cant-in-data-table
- https://github.com/Rdatatable/data.table

```
library(data.table)
```

```
## Warning: package 'data.table' was built under R version 4.4.2
```

```
DF = data.frame(x=rnorm(9),y=rep(c("a","b","c"), each=3),z=rnorm(9))
head(DF,3)
```

```
##            x y          z
## 1 -0.7068205 a  0.7140727
## 2 -1.2413159 a  0.1378134
## 3  0.1707718 a -1.2797602
```

```
DT = data.table(x=rnorm(9),y=rep(c("a","b","c"), each=3),z=rnorm(9))
head(DT,3)
```

```
##            x      y         z
##        <num> <char>     <num>
## 1: 1.4388242      a 0.3164175
## 2: 1.7242402      a 0.7911562
## 3: 0.4852805      a 0.9485147
```

```
tables() # get all data tables in memory
```

```
##    NAME NROW NCOL MB  COLS    KEY
## 1:   DT    9    3  0 x,y,z [NULL]
## Total: 0MB using type_size
```

```
# subsetting
DT[2,] # subset rows
```

```
##          x      y         z
##      <num> <char>     <num>
## 1: 1.72424      a 0.7911562
```

```
DT[DT$y=="a",] # subset where y is "a"
```

```
##            x      y         z
##        <num> <char>     <num>
## 1: 1.4388242      a 0.3164175
## 2: 1.7242402      a 0.7911562
## 3: 0.4852805      a 0.9485147
```

```
DT[c(2,3)] # subset rows 2 & 3, one variable is assigned to rows
```

```
##            x      y         z
##        <num> <char>     <num>
## 1: 1.7242402      a 0.7911562
## 2: 0.4852805      a 0.9485147
```

```
# subset cols, DT[,c(2.3)] does not work bc uses expressions
DT[,list(mean(x),sum(z))] # pass list of functions applied by names of columns
```

```
##          V1       V2
##       <num>    <num>
## 1: 0.891054 2.907013
```

```
DT[,table(y)] # get table of y values
```

```
## y
## a b c
## 3 3 3
```

```
DT[, w := z^2] # adds columns quickly
DT2 <- DT # does not make a copy in memory, change one changes all, pointing to same memory. Use copy f
DT[,m:= {tmp <- (x+z); log2(tmp+5)}] # multiple step function, returns last statement in evaluation
DT[,a:=x>0] # expression exaluates boolean for new variable
DT[,b:= mean(x+w),by=a] # grouping by boolean a into factors to evaluate expression
# special variable .N integer len 1 num times group appears
set.seed(123)
DT <- data.table(x=sample(letters[1:3], 1E5, TRUE))
DT[, .N, by=x] # count number of times grouped by x variable
```

```
##         x     N
##    <char> <int>
## 1:      c 33294
## 2:      b 33305
## 3:      a 33401
```

```r
# data.table contains keys
DT <- data.table(x=rep(c("a","b","c"),each=100), y=rnorm(300))
setkey(DT, x)
DT["a"] # subset based on key x, faster
```

```
## Key: <x>
##          x           y
##       <char>       <num>
##    1:      a  0.88631257
##    2:      a  2.82858132
##    3:      a  2.03145429
##    4:      a  1.90675413
##    5:      a  0.21490826
##    6:      a -0.86273413
##    7:      a -2.20493863
##    8:      a  0.24105923
##    9:      a  1.83832419
##   10:      a  0.79205468
##   11:      a  0.65053469
##   12:      a -1.53912061
##   13:      a -0.60830053
##   14:      a  0.38195644
##   15:      a -1.07500044
##   16:      a  0.21994264
##   17:      a -0.78288781
##   18:      a -1.11003346
##   19:      a -1.65871456
##   20:      a -0.50147343
##   21:      a  1.91636375
##   22:      a  1.41236645
##   23:      a  0.92260986
##   24:      a  1.01106201
##   25:      a  0.57213026
##   26:      a -0.62843126
##   27:      a -0.36316140
##   28:      a -1.05858811
##   29:      a -0.42935803
##   30:      a  0.86941467
##   31:      a -0.54001647
##   32:      a -1.14647747
##   33:      a -0.17151840
##   34:      a -0.56368340
##   35:      a -0.42994346
##   36:      a -1.23723779
##   37:      a  0.15901329
##   38:      a -1.16711067
##   39:      a -0.08111944
##   40:      a -0.51667953
##   41:      a  0.99540703
##   42:      a  0.79752142
##   43:      a  0.53895224
##   44:      a -1.40405605
##   45:      a  0.40144065
##  46:      a -0.52432237
##  47:      a -0.83952146
##  48:      a  0.47556591
##  49:      a -0.01194696
##  50:      a  0.10319780
##  51:      a -0.38575415
##  52:      a  1.11726438
##  53:      a -0.49961390
##  54:      a -0.44735091
##  55:      a -0.23784512
##  56:      a -0.86939374
##  57:      a  1.14887678
##  58:      a  0.53864996
##  59:      a -0.10680992
##  60:      a  0.60053649
##  61:      a -1.47499445
##  62:      a  0.98126964
##  63:      a -0.61118738
##  64:      a  0.08938648
##  65:      a -0.01327227
##  66:      a -0.97219341
##  67:      a -0.57946225
##  68:      a  0.14963144
##  69:      a  0.47640689
##  70:      a  0.44729682
##  71:      a -0.19180956
##  72:      a  0.51712710
##  73:      a  0.40338273
##  74:      a  1.78411385
##  75:      a  0.27775645
##  76:      a  0.77394978
##  77:      a -2.08081928
##  78:      a -0.35920889
##  79:      a -0.45932217
##  80:      a  0.20181947
##  81:      a  0.62401138
##  82:      a -0.25722981
##  83:      a  0.94414021
##  84:      a  0.25074808
##  85:      a -0.72784257
##  86:      a  0.36881323
##  87:      a  0.44415068
##  88:      a -1.00535422
##  89:      a -0.33152471
##  90:      a -0.37039325
##  91:      a -0.79701529
##  92:      a  0.28148559
##  93:      a  0.33307250
##  94:      a  0.52690325
##  95:      a -0.78168949
##  96:      a -0.02793948
##  97:      a -1.74492339
##  98:      a  0.65284209
##  99:      a -0.93830821
## 100:      a  0.62753159
##           x           y
```

```r
DT1 <- data.table(x=c("a","a","b","dt1"), y=1:4)
DT2 <- data.table(x=c("a","b","dt2"), z=5:7)
setkey(DT1,x); setkey(DT2,x)
merge(DT1,DT2) # uses keys to merge
```

```
## Key: <x>
##         x     y     z
##    <char> <int> <int>
## 1:      a     1     5
## 2:      a     2     5
## 3:      b     3     6
```

```r
# fast reading in data.table
big_df <- data.frame(x=rnorm(1E6),y=rnorm(1E6))
file <- tempfile()
write.table(big_df, file=file, row.names=FALSE, col.names=TRUE, sep="\t", quote=FALSE)
system.time(fread(file)) # basically read.table for csv
```

```
##    user  system elapsed
##    0.08    0.00    0.10
```

```r
system.time(read.table(file,header=TRUE,sep="\t"))
```

```
##    user  system elapsed
##    3.64    0.13    4.12
```

```r
rm(list=ls())
```

Random Numbers

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

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

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

rm(list=ls())
```

Control Functions and Loop Functions

Control Functions

```
# 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
```

```r
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
```

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

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

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

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

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

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

# control a loop
for(i in 1:100) {
  if(i <= 20) next # skip next iteration of loop
  else {
    if (i > 50) break # exit for loop
  }
}

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

Loop Functions

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
rm(list=ls())
```

Defining Functions

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

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

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

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

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

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

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

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

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

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

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

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

```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
```

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

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

```r
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
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Lexical Scoping

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

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

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

```
square(3)
```

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

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

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

```
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
rm(list=ls())
```

Cleaning Data

- End of process generate: raw data, tidy data set, code book (metadata) describing each variable and its values in the tidy data set, explicit and exact recipe used to convert raw data to tidy data set and code book.

- Raw Data: original source of data i.e. no processing, editing, summarizing. Must process for data analysis (merging, sub-setting, transforming, etc.), be mindful of processing standards. Colloquially may be later step i.e. genome seq but must use rawest.

- Processed Data: ready for analysis, steps to reach stage must be recorded. Must: one variable per column, each observation in a separate row, different tables for different types of variables, if multiple tables allow for linking. Useful: top row of variable names which are human readable, save one file per table.

- Code book: info about variables not contained in data incl. units called *Code book*, info about summary choices, info about experimental study design called *Study design*. Often word/text file.

- Instruction list: in a computer script where input is raw data and output is tidy data with no parameters to the script. If not possible, provide instructions in steps (incl. parameters, software versions, how to use software).