

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
```

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

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

```
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
```

```
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
```

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

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

```
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
```

```
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
```

```
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
```

```
seq_along(c(1,3,8,25,100)) # vector of same length 1:length(x) or seq(along=c(1,3,8,25,100))
```

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

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

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

```
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
```

```

# 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

```

```

# 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
yesno <- factor(sample(c("yes","no"), size = 10, replace = TRUE)) # randomly generate factor vector
relevel(yesno, ref = "yes") # change order of levels

```

```

## [1] yes yes no no yes yes yes no no yes
## Levels: yes no

```

```

table(x) # prints counts of each factor

```

```

## x
## female    male
##      3      2

```

```

# 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

```

Data Tables (not Frames)

- Package, faster and more memory efficient

- Inherits from data.frame (all functions), written in C, faster at sub-setting, grouping, and updating
- Much faster reading time and different operations
- <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 -1.3523839 a -1.1719693
## 2  0.5700690 a -1.6128916
## 3  0.3433981 a -0.1209614
```

```
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: -0.5019210      a -1.6371467
## 2: -0.8401438      a -0.4679885
## 3:  0.2900453      a  0.2197382
```

```
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: -0.8401438      a -0.4679885
```

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

```
##           x      y      z
##      <num> <char> <num>
## 1: -0.5019210      a -1.6371467
## 2: -0.8401438      a -0.4679885
## 3:  0.2900453      a  0.2197382
```

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

```
##           x           y           z
##      <num> <char>      <num>
## 1: -0.8401438      a -0.4679885
## 2:  0.2900453      a  0.2197382
```

```
# 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: 0.06663755 -3.588118
```

```
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 evaluates 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
```

```
# 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
```

```
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
```

```
# fast reading in data.table
big_df <- data.frame(x=rnorm(1E5),y=rnorm(1E5))
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
##      0.00    0.00    0.01
```

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

```
##      user  system elapsed
##      0.09    0.00    0.21
```

Date and Time Data Types

```
# 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] 57
```

```
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. daylight savings, timezones, l

## Time difference of 4479.79 days

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

## Time difference of 4479.79 days

rm(list=ls())

```

Basic R Functions

Functions and Operations

Mathematical functions: http://www.biostat.jhsph.edu/~ajaffe/lec_winterR/Lecture%202.pdf <http://statmethods.net/management/functions.html>

```

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

```

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

```
x # auto-prints
```

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

```

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

```

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

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

```

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

```

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

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

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

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

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

```
ceiling(3.275) # round ceiling
```

```
## [1] 4
```

```
floor(3.275) # round floor
```

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

```
round(3.275, digits = 1) # round to the num def digits after decimal
```

```
## [1] 3.3
```

```
signif(3.275, digits = 2) # digits number of sig figs
```

```
## [1] 3.3
```

```
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
```

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

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

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

```
## 25%
```

```
## 4
```

```
cos(1)
```

```
## [1] 0.5403023
```

```
sin(1)
```

```
## [1] 0.841471
```

```
log(5) # nat log
```

```
## [1] 1.609438
```

```
log2(5) # log base 2
```

```
## [1] 2.321928
```

```
log10(5) # log base 10
```

```
## [1] 0.69897
```

```
exp(5) # exponentiate x
```

```
## [1] 148.4132
```

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

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

```
# 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
```

```
# 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
```

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

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

```
## [1,]  40  40
```

```
## [2,]  60  60
```

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

```
## [1] 1.78920600 -3.88737686 1.99518314 -3.20541485 -4.05090128 -3.79883437
## [7] -2.02213195 2.16226157 1.63345371 -0.08532447 -1.25430532 -1.47138793
## [13] 0.03355353 -0.27977959 1.73400401 2.01029366 2.77742379 0.12569706
## [19] 6.45813360 3.31190713
```

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

```
## [1] 0.178920600 -0.388737686 0.199518314 -0.320541485 -0.405090128
## [6] -0.379883437 -0.202213195 0.216226157 0.163345371 -0.008532447
## [11] -0.125430532 -0.147138793 0.003355353 -0.027977959 0.173400401
## [16] 0.201029366 0.277742379 0.012569706 0.645813360 0.331190713
```

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

```
## [1] -0.7148972 -4.2166507 4.0744558 -0.5601991 -2.6841199 -2.8095718
## [7] 5.4103441 7.2562290 -2.7345951 0.9546656
```

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

```
## [1] -0.03574486 -0.21083254 0.20372279 -0.02800996 -0.13420600 -0.14047859
## [7] 0.27051720 0.36281145 -0.13672976 0.04773328
```

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

```
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]
## [1,] 0.12637339 -0.308002882 0.29429933 -0.04419720 0.233493936 -0.61559977
## [2,] 0.79960449 -0.187196467 -1.12538177 -0.52350977 1.523852248 -0.01086159
## [3,] -0.06072802 -0.097523867 -0.39619547 0.09543086 0.073963404 0.15312807
## [4,] 0.48719603 -0.080608608 -0.69093445 -0.16229164 0.864783266 0.57220373
## [5,] 0.13841794 -0.215781772 -0.24470166 -0.38369530 1.080932014 -0.36886757
## [6,] -0.19509399 -0.096189873 0.10081785 0.04854274 -0.169729837 -0.13377226
## [7,] 0.49606493 -0.025917573 -0.46890866 -0.04859081 0.671746677 0.34168032
## [8,] 0.35159448 0.012028100 -0.41749288 0.01654633 -0.008769018 0.42323744
## [9,] -0.29930074 -0.199670934 0.03987822 -0.22093468 -0.381676299 -0.11980347
## [10,] -0.15073889 -0.006812395 0.52517738 0.08918570 -0.424089951 -0.32252362
##           [,7]      [,8]      [,9]      [,10]
## [1,] -0.17587483 -0.128999532 0.11006264 0.2380813
## [2,] -1.00613760 0.313955803 -0.89063012 -0.7062199
## [3,] -0.09934475 0.074066206 -0.28585490 -0.1966579
## [4,] -0.22191176 -0.002097658 -0.49634889 -0.4895038
## [5,] -0.99693376 0.047996684 -0.03923276 0.1581226
## [6,] 0.13683516 0.043410557 0.36446769 0.1034075
## [7,] -0.12545537 0.269876402 -0.47522667 -0.7051514
## [8,] -0.24285996 -0.140684427 -0.16174622 -0.2665632
## [9,] 0.11556737 0.263864991 -0.27932346 -0.2212490
## [10,] 0.26427466 0.019397342 0.27345170 0.5805727
```

```

# 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

5 %in% c(1,4,6,8) # checks if value is in the vector

## [1] FALSE

c(5,6) %in% c(1,4,6,8) # vectorized, each gives logical

## [1] FALSE 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"

```

```
as.numeric(gsub(",", "", c("4,567", "34,234"))) # remove comma from string -> numeric
```

```
## [1] 4567 34234
```

```
# 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
```

```
# Workspace functions
```

```
x <- 1 # assigns object to x variable in ws  
rm("x") # removes variable x from ws  
rm(list=ls()) # removes all variables from ws
```

```
# Misc
```

```
intersect(c(1, 2, 5, 6, 7), c(4, 2, 6, 9, 6))
```

```
## [1] 2 6
```

Displaying and Summarizing Data

```
# Display Data Functions
x <- data.frame(foo = 1:20, rar = 301:320)
print(x) # 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
```

```
print(object.size(rnorm(1e5)), units = "Mb") # can format print based on units
```

```
## 0.8 Mb
```

```
head(x) # 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
```

```
tail(x) # 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
```

```
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 of vector, arg for useNA="if"
```

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

```
table(c(1,1,1,2,2,2,2,2,2,2,2,2,3,3,3,3,4,4,5) %in% c(3)) # get summary of logical count if 1 is in new
```

```
##
## FALSE TRUE
##    15    4
```

```
warpbreaks$rep <- rep(1:9, len = 54)
ftable(xtabs(breaks ~., data = warpbreaks)) # display 3D table as single table
```

```
##           rep  1  2  3  4  5  6  7  8  9
## wool tension
## A      L      26 30 54 25 70 52 51 26 67
##        M      18 21 29 17 12 18 35 30 36
##        H      36 21 24 18 10 43 28 15 26
## B      L      27 14 29 19 29 31 41 20 44
##        M      42 26 19 16 39 28 21 39 29
##        H      20 21 24 17 13 15 15 16 28
```

```
summary(c(3,4,5,6,7,3,3,5,7,2,8,3,5,6)) # result summaries of the results of various model fitting func
```

```
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##      2.000  3.000   5.000   4.786   6.000   8.000
```

```
quantile(c(3,4,5,6,7,3,3,5,7,2,8,3,5,6)) # gives stats quantile calculations, look at numerical spread.
```

```
##      0%  25%  50%  75% 100%
##       2    3    5    6    8
```

```
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 30.7
##  $ min    : int 57
##  $ hour   : int 23
##  $ mday   : int 29
##  $ mon    : int 0
##  $ year   : int 125
```



```
## $ wday : int 3
## $ yday : int 28
## $ 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] 6.16 10.31 -2.52 6.26 2.8 ...
```

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

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

```
str(data.frame(foo = 1:10, rar = 301:310, bee = c("g","v","e","d","d","s","c","t","h","s")))
```

```
## 'data.frame':    10 obs. of  3 variables:
## $ foo: int  1 2 3 4 5 6 7 8 9 10
## $ rar: int 301 302 303 304 305 306 307 308 309 310
## $ bee: chr  "g" "v" "e" "d" ...
```

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
```

```
x <- data.frame(foo=1:4, bar=c(T,T,F,F))
```

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

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

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

```
## [1] "foo" "bar"
```

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

```
## [1] 4
```

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

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

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

```
##      foo bar
## [1,]   1   1
## [2,]   2   1
## [3,]   3   0
## [4,]   4   0
```

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

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

```
x$cow <- 4:7 # add column called cow
x <- cbind(x, 4:7) # as above, without name
rowSums(x) # vector of sum of rows
```

```
## [1] 10 13 15 18
```

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

```
## [1] 2.50 3.25 3.75 4.50
```

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

```
## foo bar cow 4:7
## 10 2 22 22
```

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

```
## foo bar cow 4:7
## 2.5 0.5 5.5 5.5
```

```
# 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, Sorting, and Dealing with NAs

```
# Subsetting Vector
x <- c("a","b","c","c","d","a", NA)
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" NA
```

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

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

```
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" NA
```

```
# Sorting vectors  
sort(x)
```

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

```
sort(x, decreasing = TRUE)
```

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

```
sort(x, na.last = TRUE) # retains NA vals at end
```

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

```
# Subset data frame  
x <- data.frame(foo = 1:6, bar = c("g","h","i","j","k","l"))  
x[,1] # first column
```

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

```
x[, "foo"] # foo column, equivalent to x[,1]
```

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

```
x[1:2, "bar"] # first 2 observations of bar variable
```

```
## [1] "g" "h"
```

```
x[(x$foo >= 2 & x$foo < 4),] # all vars where logical is true of observations
```

```
##   foo bar
## 2    2   h
## 3    3   i
```

```
x[which(x$bar == "h"), "foo"] # get or set foo in the same row as bar of "h", **to deal with NAs**
```

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

```
x[x$bar %in% c("h","j"),] # subset rows that are as given
```

```
##   foo bar
## 2    2   h
## 4    4   j
```

```
# Order data frame
```

```
x <- data.frame(foo = sample(1:6), bar = sample(c("g","h","i","j","k","l")))
x[order(x$bar),] # sort data frame by given variable
```

```
##   foo bar
## 5    1   g
## 6    4   h
## 4    3   i
## 3    2   j
## 2    6   k
## 1    5   l
```

```
x$fact <- factor(c(4,4,3,2,3,2))
```

```
x[order(x$fact,x$foo),] # order by first then second arg
```

```
##   foo bar fact
## 4    3   i    2
## 6    4   h    2
## 5    1   g    3
## 3    2   j    3
## 1    5   l    4
## 2    6   k    4
```

```
library(plyr)
```

```
## Warning: package 'plyr' was built under R version 4.4.2
```

```
arrange(x,bar) # equivalent x[order(x$bar),]
```

```
##   foo bar fact
## 1    1   g    3
## 2    4   h    2
## 3    3   i    2
## 4    2   j    3
## 5    6   k    4
## 6    5   l    4
```

```
arrange(x, desc(bar)) # decreasing order
```

```
##   foo bar fact
## 1   5   l    4
## 2   6   k    4
## 3   2   j    3
## 4   3   i    2
## 5   4   h    2
## 6   1   g    3
```

```
# Cross tabs
```

```
x <- as.data.frame(UCBAdmissions)
```

```
xtabs(Freq ~ Gender + Admit, data = x) # find frequency of admittance based on gender
```

```
##           Admit
## Gender  Admitted Rejected
##   Male      1198      1493
##   Female      557      1278
```

```
warpbreaks$rep <- rep(1:9, len = 54)
```

```
xtabs(breaks ~., data = warpbreaks) # value displayed is breaks broken down by all other variables in d
```

```
## , , rep = 1
##
##   tension
## wool  L  M  H
##   A 26 18 36
##   B 27 42 20
##
## , , rep = 2
##
##   tension
## wool  L  M  H
##   A 30 21 21
##   B 14 26 21
##
## , , rep = 3
##
##   tension
## wool  L  M  H
##   A 54 29 24
##   B 29 19 24
##
## , , rep = 4
##
##   tension
## wool  L  M  H
##   A 25 17 18
##   B 19 16 17
##
## , , rep = 5
##
```

```
##      tension
## wool  L  M  H
##      A 70 12 10
##      B 29 39 13
##
## , , rep = 6
##
##      tension
## wool  L  M  H
##      A 52 18 43
##      B 31 28 15
##
## , , rep = 7
##
##      tension
## wool  L  M  H
##      A 51 35 28
##      B 41 21 15
##
## , , rep = 8
##
##      tension
## wool  L  M  H
##      A 26 30 15
##      B 20 39 16
##
## , , rep = 9
##
##      tension
## wool  L  M  H
##      A 67 36 26
##      B 44 29 28
```

```
ftable(xtabs(breaks ~., data = warpbreaks)) # display as single table
```

```
##           rep  1  2  3  4  5  6  7  8  9
## wool tension
## A      L      26 30 54 25 70 52 51 26 67
##        M      18 21 29 17 12 18 35 30 36
##        H      36 21 24 18 10 43 28 15 26
## B      L      27 14 29 19 29 31 41 20 44
##        M      42 26 19 16 39 28 21 39 29
##        H      20 21 24 17 13 15 15 16 28
```

```
# Subset list
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

# Info about NAs
x <- c(1,2,NA,4,NA,5)
sum(is.na(x)) # Sum of NA values

## [1] 2

```



```
any(is.na(x)) # logical for if NA are present
```

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

```
# Removing NA values
```

```
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
```

```
# for two vectors, remove all NAs
```

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

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

```
y[good]
```

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

```
# for data frames, remove all NAs
```

```
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
```

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

Random Numbers

```
# Random number generation
# Probability distribution functions have 4 functions associated: d- density, r- random number generation, p- cumulative distribution function, q- quantile function
set.seed(1) # set sequence of random number generation. set.seed(1); rnorm(5) always results in the same sequence of 5 numbers
y <- rnorm(1000) # generate vector of 1000 numbers that are standard normal distribution. Args: n, mean, sd
y <- dnorm(c(0.25,0.5,0.75)) # evaluate Normal probability density, (given mean,sd) at point or vector of points
y <- pnorm(0.5) # evaluate cumulative distribution function for normal distribution. Args: q, mean=0, sd=1
```

```

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(0) + 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
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
# Repeat loop
x0 <- 0.01; tol <- 1e-3
repeat { # infinite loop
  x1 <- rnorm(1)
  if(abs(x1 - x0) < tol) {
```

```

    break # break execution of any loop
  }
  else x0 <- x1
}

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

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

```

Loop Functions

```

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

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

```

```

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

```

```

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

```

```

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

```

```

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

```

```

## $a
## [1] 1 2

```

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

```
# sapply - same as lapply but simplify, i.e. will make list of 1 element vectors a vector, multiple elements
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 single element numeric/logical, so need to use loop

# vapply - pre-specify type of return value, safer and faster. Args: X, FUN, FUN.VALUE (generalized vector)
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 numeric 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 args for every element in list, returns
```

```
##           [,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 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
## 0.2233750 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 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] 0.1165892 -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 functions

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

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

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

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

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

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

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

```
## [1] 7.647769 6.334089 7.593286 6.268142 9.548806 9.191841 8.190586
## [8] 8.226173 8.766742 9.634012 9.245233 5.921075 6.841281 7.993894
## [15] 6.722897 9.420619 8.915033 7.623340 8.032766 8.883314 9.142204
## [22] 8.000106 7.991077 7.685731 6.878269 7.864682 7.372188 8.462985
## [29] 8.260722 6.964953 8.243108 8.238265 7.603194 7.843892 8.568631
## [36] 9.067935 7.573488 9.495201 7.325929 6.319661 7.007791 6.507580
```

```
## [43] 9.482838 9.262762 9.473943 7.560676 7.166530 7.353693 9.219610
## [50] 8.611367 6.644443 8.048528 6.539457 8.539169 7.676676 8.584478
## [57] 8.233269 7.799053 7.504200 7.847320 6.365965 10.249351 9.269278
## [64] 6.715166 8.497680 8.015868 6.417507 7.669606 7.323856 8.684615
## [71] 6.486430 8.711168 8.226389 7.602351 7.960909 5.829723 6.282247
## [78] 7.605290 7.147546 7.433600 7.803719 7.836636 8.618066 8.022089
## [85] 8.608157 8.589619 8.760178 9.216551 6.242489 8.209841 8.268497
## [92] 5.898325 7.066211 7.331588 8.012126 7.676726 9.810349 7.364054
## [99] 6.261003 7.979133
```

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

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

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

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

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

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

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

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

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

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

```
# ways to call functions
# positional matching and naming can be mixed. Partial matching also allowed, if not found uses 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] 1.001767
```

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

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

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

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

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

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

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

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

```
# 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
```

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:plyr"      "package:data.table"
## [4] "package:stats"    "package:graphics"  "package:grDevices"
## [7] "package:utils"    "package:datasets"  "package:methods"
## [10] "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
```

```
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(tidy) 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).

Steps to clean Data

- Look at the data, summarize to find out quirks, missing values, etc.
- Create new data if applicable: i.e. missingness indicators, cutting up quant variables, applying transformations.
- Reshape data to the desired format from raw data.

Creating new variables in your data table


```

fileUrl <- "https://hub.arcgis.com/api/v3/datasets/42f8856d647a41b89561e10fb60bc98a_0/downloads/data?fo
if(!dir.exists("./testdir")) {
  dir.create("./testdir")
}
download.file(fileUrl, destfile = "./testdir/restdata.csv", method = "curl")
dateDownloaded <- date()
restdata <- read.csv("./testdir/restdata.csv")

# Variable to subset data
restdata$nearMe <- restdata$nghbrhd %in% c("Roland Park", "Homeland") # create col for logical in restdata
table(restdata$nearMe) # summarize data

```

```

##
## FALSE TRUE
## 1314 13

```

```

# Create binary variables
restdata$walkingDistance <- ifelse(restdata$zipcode == "21210", TRUE, FALSE) # logical using ifelse command
table(restdata$walkingDistance, restdata$zipcode == "21210")

```

```

##
## FALSE TRUE
## FALSE 1304 0
## TRUE 0 23

```

```

# Create categorical variables
restdata$zipcode <- as.numeric(restdata$zipcode)

```

```

## Warning: NAs introduced by coercion

```

```

restdata$zipgroups <- cut(restdata$zipcode, breaks = quantile(restdata$zipcode, na.rm = TRUE)) # cut using quantiles
table(restdata$zipgroups)

```

```

##
## (21201,21202] (21202,21218] (21218,21226] (21226,21287]
## 201 375 282 332

```

```

library(Hmisc)

```

```

## Warning: package 'Hmisc' was built under R version 4.4.2

```

```

##
## Attaching package: 'Hmisc'

```

```

## The following objects are masked from 'package:plyr':
##
## is.discrete, summarize

```

```
## The following objects are masked from 'package:base':
##
##     format.pval, units
```

```
restdata$zipgroups2 <- cut2(restdata$zipcode, g = 4) # same as above
table(restdata$zipgroups2)
```

```
##
## [21201,21205) [21205,21220) [21220,21227) [21227,21287]
##           337           375           300           314
```

```
# Create factor variables
restdata$zcf <- factor(restdata$zipcode)
restdata$zcf[1:10]
```

```
## [1] 21206 21231 21224 21211 21223 21218 21205 21211 21205 21231
## 31 Levels: 21201 21202 21205 21206 21207 21208 21209 21210 21211 ... 21287
```

```
as.numeric(restdata$zcf) # helpful in some models
```

```
## [1] 4 26 20 9 19 16 3 9 3 26 22 13 20 20 2 1 3 20 26 25 18 1 19 2
## [25] 26 26 13 19 10 20 25 20 1 1 4 25 8 4 11 26 16 20 2 12 2 20 19 18
## [49] 25 1 20 26 13 9 21 26 20 4 25 8 16 25 19 15 22 25 20 27 9 2 20 20
## [73] 1 13 2 25 25 25 25 25 25 25 25 25 1 2 15 26 26 26 9 10 2 20 20 2
## [97] 20 15 19 19 19 20 2 7 2 1 1 8 20 16 11 2 2 2 2 2 1 19 26 25
## [121] 1 25 2 26 2 25 25 1 20 26 4 20 2 1 26 20 20 20 25 20 2 26 1 8
## [145] 26 8 9 25 20 26 20 4 21 21 21 19 26 20 27 16 27 25 26 20 2 15 9 2
## [169] 13 25 13 2 2 20 26 26 20 16 1 2 1 2 26 25 1 2 10 2 1 13 2 20
## [193] 20 20 20 25 1 1 26 20 25 8 26 26 19 25 15 26 25 20 23 27 1 13 2 4
## [217] 1 2 1 10 2 1 16 1 16 2 25 25 20 20 26 26 1 16 20 2 2 20 13 20
## [241] 20 1 2 29 1 9 2 16 2 7 16 15 8 25 26 20 29 2 26 2 2 25 20 1
## [265] 1 11 9 1 15 1 1 1 19 26 2 11 19 19 3 21 2 2 19 2 1 2 14 19
## [289] 11 4 11 25 1 1 21 20 25 20 9 22 2 1 25 9 6 2 20 12 15 2 2 2
## [313] 25 20 7 20 7 20 22 22 3 20 3 2 25 13 25 16 13 20 1 9 1 26 27 22
## [337] 25 20 2 20 7 15 2 3 20 8 9 26 2 9 20 25 1 18 1 16 8 19 1 25
## [361] 1 11 21 25 13 20 26 26 1 9 20 7 10 13 12 20 1 2 20 22 2 20 11 1
## [385] 26 12 2 2 1 3 2 20 20 20 9 26 26 1 20 20 2 20 20 25 20 2 13 16
## [409] 2 22 16 22 20 25 26 25 10 14 25 14 2 20 1 2 26 2 3 2 2 26 9 2
## [433] 2 25 25 5 2 5 4 9 9 22 24 16 26 26 15 15 20 20 20 1 1 3 25 20
## [457] 2 26 3 18 13 2 26 19 20 19 2 12 2 2 9 15 1 2 10 13 20 19 24 1
## [481] 3 2 15 9 9 3 20 21 25 20 20 16 2 16 1 3 26 20 15 14 2 26 25 22
## [505] 26 20 13 1 20 25 13 15 1 20 25 25 3 26 26 2 2 8 20 20 25 16 1 23
## [529] 25 25 1 20 2 2 26 26 26 8 8 25 4 19 25 2 11 15 2 22 20 18 25 15
## [553] 20 15 26 20 1 4 2 20 25 14 20 20 20 4 9 1 23 2 2 27 12 12 25 2
## [577] 20 9 1 1 1 8 25 2 3 30 1 2 25 1 1 2 19 20 2 20 15 1 7 25
## [601] 1 25 25 10 26 20 2 20 2 1 16 16 25 20 3 10 9 1 25 11 24 13 16 19
## [625] 16 11 15 16 24 26 25 25 2 20 25 25 13 20 20 26 21 20 20 2 1 25 12 2
## [649] 21 19 25 2 9 2 9 26 2 8 1 16 20 26 2 2 20 25 25 2 2 2 2 2
## [673] 16 16 20 2 2 10 20 19 2 25 2 19 2 16 1 13 2 2 13 20 1 26 25 2
## [697] 14 19 20 20 19 25 2 4 20 25 2 20 26 26 10 12 26 19 4 1 2 16 24 20
## [721] 13 4 13 4 25 13 14 25 25 1 9 1 20 26 20 16 4 20 25 12 9 26 13 16
## [745] 1 25 14 1 24 25 25 26 26 20 25 26 2 20 2 16 20 16 11 2 26 26 2 26
```

```
## [769] 20 20 26 20 2 19 2 15 11 2 11 25 19 2 25 19 15 25 1 15 1 20 25 20
## [793] 20 8 25 2 13 9 19 20 26 26 3 20 13 25 26 1 1 2 1 16 13 2 15 25
## [817] 1 20 4 20 13 13 9 19 25 1 9 2 25 26 27 25 20 26 21 2 1 1 9 2
## [841] 25 2 2 2 11 1 16 2 24 9 13 10 4 4 20 13 25 19 24 1 13 2 25 9
## [865] 26 26 25 2 20 26 1 2 10 26 16 25 19 25 1 20 16 25 25 20 2 20 13 16
## [889] 2 21 20 20 15 4 11 2 26 16 15 14 8 26 18 16 20 16 2 20 4 15 13 16
## [913] 22 19 19 20 13 2 10 12 16 8 9 13 25 2 23 26 20 1 20 19 20 16 25 26
## [937] 25 1 26 10 21 2 1 25 22 21 2 13 10 2 2 16 26 8 16 2 11 2 2 1
## [961] 25 2 13 20 2 2 26 20 25 20 20 26 20 24 4 10 4 25 13 13 20 20 25 13
## [985] 2 2 2 2 16 26 26 22 13 11 1 2 21 9 26 20 19 25 19 26 2 17 2 25
## [1009] 29 16 3 19 25 19 2 1 11 10 25 20 2 10 19 8 1 26 1 20 19 25 20 19
## [1033] 20 26 2 20 25 2 3 8 25 25 1 20 26 4 1 1 1 1 20 26 11 20 26 1
## [1057] 2 25 20 26 16 20 16 26 2 9 13 25 25 25 26 25 4 25 26 19 16 14 25 21
## [1081] 25 2 1 2 2 1 1 19 1 26 25 20 20 20 1 19 26 19 25 20 13 5 20 19
## [1105] 16 16 2 25 3 2 7 20 16 1 1 15 25 1 20 11 21 11 30 2 25 2 31 24
## [1129] 26 1 15 16 25 26 8 21 20 2 9 2 16 25 24 24 25 20 25 2 26 26 2 16
## [1153] 10 1 10 22 NA 22 11 25 15 1 25 16 25 26 25 2 20 25 26 1 1 2 26 2
## [1177] 5 26 12 2 1 19 16 1 1 11 11 1 8 20 4 16 3 20 20 1 26 10 26 4
## [1201] 12 25 20 26 2 24 26 2 11 20 20 16 16 12 2 19 1 13 2 20 26 25 12 1
## [1225] 26 1 20 25 16 4 25 11 25 22 20 26 25 26 2 2 26 3 20 20 16 11 3 20
## [1249] 20 2 2 2 20 13 1 13 8 25 11 2 1 25 3 1 16 1 26 26 2 25 20 20
## [1273] 3 1 19 21 2 12 2 20 16 15 2 20 20 20 10 13 28 20 16 18 19 13 1 20
## [1297] 16 10 13 2 19 15 26 11 4 12 20 20 3 9 16 10 13 2 1 1 13 20 10 10
## [1321] 26 25 10 26 11 9 20
```

```
# Mutate function
```

```
library(Hmisc); library(plyr)
```

```
restdata2 <- mutate(restdata, zipgroups3 = cut2(zipcode, g = 4)) # create a new data frame with new var
```

```
table(restdata2$zipgroups3)
```

```
##
```

```
## [21201,21205) [21205,21220) [21220,21227) [21227,21287]
```

```
## 337 375 300 314
```

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

Reshaping Data

- <http://vita.had.co.nz/papers/tidy-data.pdf>
- <http://www.slideshare.net/jeffreymbreen/reshaping-data-in-r>
- <http://www.r-bloggers.com/a-quick-primer-on-split-apply-combine-problems/>
- Useful functions: `acast` (multi-dim arrays like `dcast`), `arrange` (faster reordering), `mutate` (add new variables).

```
library(reshape2); library(plyr)
```

```
## Warning: package 'reshape2' was built under R version 4.4.2
```

```
##
## Attaching package: 'reshape2'

## The following objects are masked from 'package:data.table':
##
## dcast, melt
```

```
head(mtcars)
```

```
##           mpg  cyl  disp  hp  drat    wt  qsec vs  am  gear  carb
## Mazda RX4      21.0   6  160 110 3.90 2.620 16.46 0   1    4    4
## Mazda RX4 Wag  21.0   6  160 110 3.90 2.875 17.02 0   1    4    4
## Datsun 710      22.8   4  108  93 3.85 2.320 18.61 1   1    4    1
## Hornet 4 Drive  21.4   6  258 110 3.08 3.215 19.44 1   0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02 0   0    3    2
## Valiant        18.1   6  225 105 2.76 3.460 20.22 1   0    3    1
```

```
# Melt data frames
```

```
mtcars$carname <- rownames(mtcars)
carmelt <- melt(mtcars, id = c("carname", "gear", "cyl"), measure.vars = c("mpg", "hp")) # assign id variables
head(carmelt, n=3)
```

```
##           carname gear  cyl variable value
## 1      Mazda RX4    4    6      mpg    21.0
## 2 Mazda RX4 Wag    4    6      mpg    21.0
## 3      Datsun 710    4    4      mpg    22.8
```

```
tail(carmelt, n=3)
```

```
##           carname gear  cyl variable value
## 62  Ferrari Dino    5    6      hp    175
## 63 Maserati Bora    5    8      hp    335
## 64   Volvo 142E     4    4      hp    109
```

```
# Casting data frames
```

```
dcast(carmelt, cyl ~ variable) # reshape data as cylinders (rows) broken down by the variables (cols) separately
```

```
## Aggregation function missing: defaulting to length
```

```
##   cyl mpg hp
## 1   4  11 11
## 2   6   7  7
## 3   8  14 14
```

```
dcast(carmelt, cyl ~ variable, mean) # as above, using mean to summarize
```

```
##   cyl      mpg      hp
## 1   4 26.66364 82.63636
## 2   6 19.74286 122.28571
## 3   8 15.10000 209.21429
```

```
# Summing (apply function to) values
head(InsectSprays)
```

```
##      count spray
## 1      10     A
## 2       7     A
## 3      20     A
## 4      14     A
## 5      14     A
## 6      12     A
```

```
tapply(InsectSprays$count, InsectSprays$spray, sum) # take sum of values within same spray value (i.e. .
```

```
##      A      B      C      D      E      F
## 174 184   25   59   42 200
```

```
spIns <- split(InsectSprays$count, InsectSprays$spray) # split insect spray into different vectors in a
sapply(spIns, sum) # as tapply above, sum each vector into names vector format
```

```
##      A      B      C      D      E      F
## 174 184   25   59   42 200
```

```
ddply(InsectSprays,.(spray),sum=sum(count)) # get sum using plyr package, list variables to summarize,
```

```
##      count spray
## 1      10     A
## 2       7     A
## 3      20     A
## 4      14     A
## 5      14     A
## 6      12     A
## 7      10     A
## 8      23     A
## 9      17     A
## 10     20     A
## 11     14     A
## 12     13     A
## 13     11     B
## 14     17     B
## 15     21     B
## 16     11     B
## 17     16     B
## 18     14     B
## 19     17     B
## 20     17     B
## 21     19     B
## 22     21     B
## 23       7     B
## 24     13     B
## 25       0     C
## 26       1     C
```

```
## 27      7      C
## 28      2      C
## 29      3      C
## 30      1      C
## 31      2      C
## 32      1      C
## 33      3      C
## 34      0      C
## 35      1      C
## 36      4      C
## 37      3      D
## 38      5      D
## 39     12      D
## 40      6      D
## 41      4      D
## 42      3      D
## 43      5      D
## 44      5      D
## 45      5      D
## 46      5      D
## 47      2      D
## 48      4      D
## 49      3      E
## 50      5      E
## 51      3      E
## 52      5      E
## 53      3      E
## 54      6      E
## 55      1      E
## 56      1      E
## 57      3      E
## 58      2      E
## 59      6      E
## 60      4      E
## 61     11      F
## 62      9      F
## 63     15      F
## 64     22      F
## 65     15      F
## 66     16      F
## 67     13      F
## 68     10      F
## 69     26      F
## 70     26      F
## 71     24      F
## 72     13      F
```

```
ddply(InsectSprays,.(spray),sum=ave(count, FUN = sum)) # apply sums then create data frame with sum of
```

```
##      count spray
## 1      10      A
## 2       7      A
## 3     20      A
## 4     14      A
```

## 5	14	A
## 6	12	A
## 7	10	A
## 8	23	A
## 9	17	A
## 10	20	A
## 11	14	A
## 12	13	A
## 13	11	B
## 14	17	B
## 15	21	B
## 16	11	B
## 17	16	B
## 18	14	B
## 19	17	B
## 20	17	B
## 21	19	B
## 22	21	B
## 23	7	B
## 24	13	B
## 25	0	C
## 26	1	C
## 27	7	C
## 28	2	C
## 29	3	C
## 30	1	C
## 31	2	C
## 32	1	C
## 33	3	C
## 34	0	C
## 35	1	C
## 36	4	C
## 37	3	D
## 38	5	D
## 39	12	D
## 40	6	D
## 41	4	D
## 42	3	D
## 43	5	D
## 44	5	D
## 45	5	D
## 46	5	D
## 47	2	D
## 48	4	D
## 49	3	E
## 50	5	E
## 51	3	E
## 52	5	E
## 53	3	E
## 54	6	E
## 55	1	E
## 56	1	E
## 57	3	E
## 58	2	E

```
## 59      6      E
## 60      4      E
## 61     11      F
## 62      9      F
## 63     15      F
## 64     22      F
## 65     15      F
## 66     16      F
## 67     13      F
## 68     10      F
## 69     26      F
## 70     26      F
## 71     24      F
## 72     13      F
```

Managing data frames with dplyr

- Package designed to work with data frame (assumes tidy data, properly formatted and annotated), can use with R implementation or data.table, SQL(DBI package), or other implementations.
- Developed by Hadley Wickham, optimized and distilled plyr (faster, coded low level in C++). Consistent and concise grammar.
- Format: first arg is a data frame, subsequent args explain what to do with it. Refer to cols just by name without \$ operator. Results in new data frame.

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:Hmisc':
##
##      src, summarize

## The following objects are masked from 'package:plyr':
##
##      arrange, count, desc, failwith, id, mutate, rename, summarise,
##      summarize

## The following objects are masked from 'package:data.table':
##
##      between, first, last

## The following objects are masked from 'package:stats':
##
##      filter, lag

## The following objects are masked from 'package:base':
##
##      intersect, setdiff, setequal, union
```



```
chicago <- readRDS("chicago.rds")
str(chicago)
```

```
## 'data.frame': 6940 obs. of 8 variables:
## $ city : chr "chic" "chic" "chic" "chic" ...
## $ tmpd : num 31.5 33 33 29 32 40 34.5 29 26.5 32.5 ...
## $ dptp : num 31.5 29.9 27.4 28.6 28.9 ...
## $ date : Date, format: "1987-01-01" "1987-01-02" ...
## $ pm25tmean2: num NA NA NA NA NA NA NA NA NA ...
## $ pm10tmean2: num 34 NA 34.2 47 NA ...
## $ o3tmean2 : num 4.25 3.3 3.33 4.38 4.75 ...
## $ no2tmean2 : num 20 23.2 23.8 30.4 30.3 ...
```

```
# select: return subset of cols of a data frame
head(select(chicago, city:dptp)) # use dplyr to select subset of cols by names
```

```
## city tmpd dptp
## 1 chic 31.5 31.500
## 2 chic 33.0 29.875
## 3 chic 33.0 27.375
## 4 chic 29.0 28.625
## 5 chic 32.0 28.875
## 6 chic 40.0 35.125
```

```
head(select(chicago, -(city:dptp))) # exclude in selection, equivalent to head(chicago[, -(match("city"
```

```
## date pm25tmean2 pm10tmean2 o3tmean2 no2tmean2
## 1 1987-01-01 NA 34.00000 4.250000 19.98810
## 2 1987-01-02 NA NA 3.304348 23.19099
## 3 1987-01-03 NA 34.16667 3.333333 23.81548
## 4 1987-01-04 NA 47.00000 4.375000 30.43452
## 5 1987-01-05 NA NA 4.750000 30.33333
## 6 1987-01-06 NA 48.00000 5.833333 25.77233
```

```
head(select(chicago, city, dptp)) # gives only two named cols as specified
```

```
## city dptp
## 1 chic 31.500
## 2 chic 29.875
## 3 chic 27.375
## 4 chic 28.625
## 5 chic 28.875
## 6 chic 35.125
```

```
# filter: extract subset of rows from data frame based on logical condition
chic.f <- filter(chicago, pm25tmean2 > 30) # logical as second arg, creates logical sequence to subset
head(chic.f, 10)
```

```
## city tmpd dptp date pm25tmean2 pm10tmean2 o3tmean2 no2tmean2
## 1 chic 23 21.9 1998-01-17 38.10 32.46154 3.180556 25.30000
```

```
## 2  chic  28 25.8 1998-01-23      33.95   38.69231  1.750000  29.37630
## 3  chic  55 51.3 1998-04-30      39.40   34.00000 10.786232  25.31310
## 4  chic  59 53.7 1998-05-01      35.40   28.50000 14.295125  31.42905
## 5  chic  57 52.0 1998-05-02      33.30   35.00000 20.662879  26.79861
## 6  chic  57 56.0 1998-05-07      32.10   34.50000 24.270422  33.99167
## 7  chic  75 65.8 1998-05-15      56.50   91.00000 38.573007  29.03261
## 8  chic  61 59.0 1998-06-09      33.80   26.00000 17.890810  25.49668
## 9  chic  73 60.3 1998-07-13      30.30   64.50000 37.018865  37.93056
## 10 chic  78 67.1 1998-07-14      41.40   75.00000 40.080902  32.59054
```

```
head(filter(chicago, pm25tmean2 > 30, tmpd > 80)) # multiple logical, can sep using commas for and
```

```
##   city tmpd dtp      date pm25tmean2 pm10tmean2 o3tmean2 no2tmean2
## 1 chic  81 71.2 1998-08-23   39.6000      59.0 45.86364  14.32639
## 2 chic  81 70.4 1998-09-06   31.5000      50.5 50.66250  20.31250
## 3 chic  82 72.2 2001-07-20   32.3000      58.5 33.00380  33.67500
## 4 chic  84 72.9 2001-08-01   43.7000      81.5 45.17736  27.44239
## 5 chic  85 72.6 2001-08-08   38.8375      70.0 37.98047  27.62743
## 6 chic  84 72.6 2001-08-09   38.2000      66.0 36.73245  26.46742
```

```
# arrange: reorder rows of a data frame while preserving order of other cols
```

```
head(arrange(chicago, date)) # arrange dt based on one variable, ascending, can sort by multiple
```

```
##   city tmpd  dtp      date pm25tmean2 pm10tmean2 o3tmean2 no2tmean2
## 1 chic 31.5 31.500 1987-01-01         NA    34.00000 4.250000  19.98810
## 2 chic 33.0 29.875 1987-01-02         NA         NA 3.304348  23.19099
## 3 chic 33.0 27.375 1987-01-03         NA    34.16667 3.333333  23.81548
## 4 chic 29.0 28.625 1987-01-04         NA    47.00000 4.375000  30.43452
## 5 chic 32.0 28.875 1987-01-05         NA         NA 4.750000  30.33333
## 6 chic 40.0 35.125 1987-01-06         NA    48.00000 5.833333  25.77233
```

```
chicago <- arrange(chicago, desc(date)) # descending
```

```
# rename: rename variables in a data frame
```

```
chicago <- rename(chicago, pm25 = pm25tmean2, dewpoint = dtp) # renames multiple variables
str(chicago)
```

```
## 'data.frame':   6940 obs. of  8 variables:
## $ city      : chr  "chic" "chic" "chic" "chic" ...
## $ tmpd      : num  35 36 35 37 40 35 35 37 41 22 ...
## $ dewpoint   : num  30.1 31 29.4 34.5 33.6 29.6 32.1 35.2 32.6 23.3 ...
## $ date       : Date, format: "2005-12-31" "2005-12-30" ...
## $ pm25       : num  15 15.06 7.45 17.75 23.56 ...
## $ pm10tmean2 : num  23.5 19.2 23.5 27.5 27 8.5 8 25.2 34.5 42.5 ...
## $ o3tmean2   : num  2.53 3.03 6.79 3.26 4.47 ...
## $ no2tmean2  : num  13.2 22.8 20 19.3 23.5 ...
```

```
# mutate: add new variables/columns or transform existing variables
```

```
chicago <- mutate(chicago, pm25detrend = pm25 - mean(pm25, na.rm = TRUE)) # add new col based on other cols
head(select(chicago, pm25, pm25detrend))
```

```
##      pm25 pm25detrend
## 1 15.00000 -1.230958
## 2 15.05714 -1.173815
## 3  7.45000 -8.780958
## 4 17.75000  1.519042
## 5 23.56000  7.329042
## 6  8.40000 -7.830958
```

```
# group_by: split data frame based on categorical variables
```

```
chicago <- mutate(chicago, tempcat = factor(1 * (tmpd > 80), labels = c("cold","hot"))) # create factor
hotcold <- group_by(chicago,tempcat) # group by factor variable
head(hotcold)
```

```
## # A tibble: 6 x 10
## # Groups:   tempcat [1]
##   city   tmpd dewpoint date      pm25 pm10tmean2 o3tmean2 no2tmean2
##   <chr> <dbl>   <dbl> <date>   <dbl>      <dbl>      <dbl>      <dbl>
## 1 chic    35    30.1 2005-12-31 15        23.5        2.53       13.2
## 2 chic    36    31   2005-12-30 15.1       19.2        3.03       22.8
## 3 chic    35    29.4 2005-12-29  7.45      23.5        6.79       20.0
## 4 chic    37    34.5 2005-12-28 17.8       27.5        3.26       19.3
## 5 chic    40    33.6 2005-12-27 23.6       27         4.47       23.5
## 6 chic    35    29.6 2005-12-26  8.4       8.5       14.0       16.8
## # i 2 more variables: pm25detrend <dbl>, tempcat <fct>
```

```
summarize(hotcold, pm25 = mean(pm25, na.rm = TRUE), o3 = max(o3tmean2), no2 = median(no2tmean2)) # gene
```

```
## # A tibble: 3 x 4
##   tempcat pm25   o3   no2
##   <fct>   <dbl> <dbl> <dbl>
## 1 cold    16.0 66.6  24.5
## 2 hot     26.5 63.0  24.9
## 3 <NA>    47.7  9.42 37.4
```

```
chicago <- mutate(chicago, year = as.POSIXlt(date)$year + 1900)
```

```
years <- group_by(chicago, year)
```

```
summarize(years, pm25 = mean(pm25, na.rm = TRUE), o3 = max(o3tmean2), no2 = median(no2tmean2)) # count
```

```
## # A tibble: 19 x 4
##   year pm25   o3   no2
##   <dbl> <dbl> <dbl> <dbl>
## 1 1987 NaN    63.0 23.5
## 2 1988 NaN    61.7 24.5
## 3 1989 NaN    59.7 26.1
## 4 1990 NaN    52.2 22.6
## 5 1991 NaN    63.1 21.4
## 6 1992 NaN    50.8 24.8
## 7 1993 NaN    44.3 25.8
## 8 1994 NaN    52.2 28.5
## 9 1995 NaN    66.6 27.3
## 10 1996 NaN    58.4 26.4
## 11 1997 NaN    56.5 25.5
```

```
## 12 1998 18.3 50.7 24.6
## 13 1999 18.5 57.5 24.7
## 14 2000 16.9 55.8 23.5
## 15 2001 16.9 51.8 25.1
## 16 2002 15.3 54.9 22.7
## 17 2003 15.2 56.2 24.6
## 18 2004 14.6 44.5 23.4
## 19 2005 16.2 58.8 22.6
```

%>%: chain operations

```
chicago %>% mutate(month = as.POSIXlt(date)$mon + 1) %>% group_by(month) %>% summarize(pm25 = mean(pm25
```

```
## # A tibble: 12 x 4
##   month pm25    o3    no2
##   <dbl> <dbl> <dbl> <dbl>
## 1     1  17.8  28.2  25.4
## 2     2  20.4  37.4  26.8
## 3     3  17.4  39.0  26.8
## 4     4  13.9  47.9  25.0
## 5     5  14.1  52.8  24.2
## 6     6  15.9  66.6  25.0
## 7     7  16.6  59.5  22.4
## 8     8  16.9  54.0  23.0
## 9     9  15.9  57.5  24.5
## 10    10  14.2  47.1  24.2
## 11    11  15.2  29.5  23.6
## 12    12  17.5  27.7  24.5
```

summarize: generate summary statistics of different variables in the data frame, possibly within strata
summarize(chicago, pm25 = mean(pm25, na.rm = TRUE), o3 = max(o3tmean2), no2 = median(no2tmean2)) # can

```
##           pm25           o3           no2
## 1 16.23096 66.5875 24.55556
```

print: prevents printing lots of data to console

```
tabular <- as_tibble(chicago) # converts to a tbl_df object, can be done before all dplyr operations
tabular # printing useful in tibble, prints neatly and concisely
```

```
## # A tibble: 6,940 x 11
##   city tmpd dewpoint date pm25 pm10tmean2 o3tmean2 no2tmean2
##   <chr> <dbl>    <dbl> <date>    <dbl>    <dbl>    <dbl>    <dbl>
## 1 chic  35    30.1 2005-12-31 15      23.5     2.53    13.2
## 2 chic  36    31   2005-12-30 15.1     19.2     3.03    22.8
## 3 chic  35    29.4 2005-12-29 7.45     23.5     6.79    20.0
## 4 chic  37    34.5 2005-12-28 17.8     27.5     3.26    19.3
## 5 chic  40    33.6 2005-12-27 23.6     27      4.47    23.5
## 6 chic  35    29.6 2005-12-26 8.4      8.5     14.0    16.8
## 7 chic  35    32.1 2005-12-25 6.7      8      14.4    13.8
## 8 chic  37    35.2 2005-12-24 30.8     25.2     1.77    32.0
## 9 chic  41    32.6 2005-12-23 32.9     34.5     6.91    29.1
## 10 chic  22    23.3 2005-12-22 36.6     42.5     5.39    33.7
## # i 6,930 more rows
## # i 3 more variables: pm25detrend <dbl>, tempcat <fct>, year <dbl>
```

```
# join: merge two data sets using ID of common name, join_all for multiple data frames
df1 <- data.frame(id = sample(1:10), x = rnorm(10))
df2 <- data.frame(id = sample(1:10), y = rnorm(10))
arrange(join(df1,df2),id)
```

```
## Joining by: id
```

```
##      id      x      y
## 1     1 -1.6255466 -1.03720971
## 2     2  0.8789114 -0.08514748
## 3     3 -0.6765178 -0.09053349
## 4     4  1.2359234 -0.44137715
## 5     5  0.7635296 -0.34346738
## 6     6 -1.0199867 -0.12145522
## 7     7 -2.5337551 -0.68485203
## 8     8  1.6377076 -0.43148133
## 9     9 -0.1192970  0.20709858
## 10    10  0.8568079 -0.29898524
```

```
df3 <- data.frame(id = sample(1:10), z = rnorm(10))
join_all(list(df1,df2,df3)) # based on common name
```

```
## Joining by: id
```

```
## Joining by: id
```

```
##      id      x      y      z
## 1     7 -2.5337551 -0.68485203  1.7862947
## 2     6 -1.0199867 -0.12145522 -0.7796692
## 3     1 -1.6255466 -1.03720971 -0.1438986
## 4    10  0.8568079 -0.29898524 -0.3877775
## 5     4  1.2359234 -0.44137715 -0.7889068
## 6     8  1.6377076 -0.43148133 -0.6195299
## 7     3 -0.6765178 -0.09053349  0.1134572
## 8     9 -0.1192970  0.20709858  0.5250819
## 9     5  0.7635296 -0.34346738  0.1058529
## 10    2  0.8789114 -0.08514748 -0.5787796
```

Merge data

- Usually done by matching data sets using IDs, similar to SQL.
- <http://www.statmethods.net/management/merging.html>
- [http://en.wikipedia.org/wiki/Join_\(SQL\)](http://en.wikipedia.org/wiki/Join_(SQL))

```
# Get data
if(!dir.exists("./testdir")) dir.create("./testdir")
fileURL1 <-
  "https://raw.githubusercontent.com/DataScienceSpecialization/courses/refs/heads/master/03_GettingData"
```

```

fileURL2 <- "https://raw.githubusercontent.com/DataScienceSpecialization/courses/refs/heads/master/03_G
download.file(fileURL1,destfile="./testdir/reviews.csv",method="curl")
download.file(fileURL2,destfile="./testdir/solutions.csv",method="curl")
dateDownloaded <- date()
reviews <- read.csv("./testdir/reviews.csv"); solutions <- read.csv("./testdir/solutions.csv")
head(reviews,2)

```

```

##   id solution_id reviewer_id      start      stop time_left accept
## 1  1           3          27 1304095698 1304095758      1754      1
## 2  2           4          22 1304095188 1304095206      2306      1

```

```
head(solutions,2)
```

```

##   id problem_id subject_id      start      stop time_left answer
## 1  1          156         29 1304095119 1304095169      2343      B
## 2  2          269         25 1304095119 1304095183      2329      C

```

```

# merge: args x,y (dataframes),by,by.x,by.y (cols to merge by), all(include all values even if missing)
testingData <- merge(reviews, solutions, by.x = "solution_id", by.y = "id", all = TRUE)
head(testingData)

```

```

##   solution_id id reviewer_id      start.x      stop.x time_left.x accept
## 1           1  4          26 1304095267 1304095423      2089      1
## 2           2  6          29 1304095471 1304095513      1999      1
## 3           3  1          27 1304095698 1304095758      1754      1
## 4           4  2          22 1304095188 1304095206      2306      1
## 5           5  3          28 1304095276 1304095320      2192      1
## 6           6 16          22 1304095303 1304095471      2041      1
##   problem_id subject_id      start.y      stop.y time_left.y answer
## 1          156         29 1304095119 1304095169      2343      B
## 2          269         25 1304095119 1304095183      2329      C
## 3           34         22 1304095127 1304095146      2366      C
## 4           19         23 1304095127 1304095150      2362      D
## 5          605         26 1304095127 1304095167      2345      A
## 6          384         27 1304095131 1304095270      2242      C

```

```

mergedData <- merge(reviews, solutions, all = TRUE) # merges with all intersecting data, "id", "start",
head(mergedData)

```

```

##   id      start      stop time_left solution_id reviewer_id accept problem_id
## 1  1 1304095119 1304095169      2343          NA          NA      NA      156
## 2  1 1304095698 1304095758      1754           3          27      1        NA
## 3  2 1304095119 1304095183      2329          NA          NA      NA      269
## 4  2 1304095188 1304095206      2306           4          22      1        NA
## 5  3 1304095127 1304095146      2366          NA          NA      NA      34
## 6  3 1304095276 1304095320      2192           5          28      1        NA
##   subject_id answer
## 1          29      B
## 2          NA    <NA>
## 3          25      C
## 4          NA    <NA>
## 5          22      C
## 6          NA    <NA>

```

```
# join in dplyr, faster but less featured. Works best for multiple data sets
```

Tidy data with tidyr

- By Hadley Wickham. Tidy data is formatted in a standard way that facilitates exploration and analysis and works seamlessly with other tidy data tools. Messy data symptoms: Column headers are values, not variable names; Variables are stored in both rows and columns; A single observational unit is stored in multiple tables; Multiple types of observational units are stored in the same table; Multiple variables are stored in one column.

- <http://vita.had.co.nz/papers/tidy-data.pdf>

```
library(tidyr); library(dplyr); library(readr)
```

```
## Warning: package 'tidyr' was built under R version 4.4.2
```

```
##
```

```
## Attaching package: 'tidyr'
```

```
## The following object is masked from 'package:reshape2':
```

```
##
```

```
## smiths
```

```
## Warning: package 'readr' was built under R version 4.4.2
```

```
# gather: column headers that are values, not variable names
```

```
students <- data.frame(grade = c("A","B","C","D","E"), male = c(1,5,5,5,7), female = c(5,0,2,5,4))
```

```
# note the actual variables are grade, sex, and count
```

```
gather(students, sex, count, -grade) # now is tidy data, each row is separate observation (grade, sex c
```

```
##   grade    sex count
## 1     A  male     1
## 2     B  male     5
## 3     C  male     5
## 4     D  male     5
## 5     E  male     7
## 6     A female     5
## 7     B female     0
## 8     C female     2
## 9     D female     5
## 10    E female     4
```

```
# separate: multiple variables are stored in one column
```

```
students2 <- data.frame(grade = c("A","B","C","D","E"), male_1 = c(3,6,7,4,1), female_1 = c(4,4,4,0,1),
```

```
res <- gather(students2, sex_class, count, -grade) # split variables of count
```

```
separate(res, col = sex_class, into = c("sex","class")) # separate sex_count in same col, splits on non
```

```
##      grade    sex class count
## 1      A    male     1     3
## 2      B    male     1     6
## 3      C    male     1     7
## 4      D    male     1     4
## 5      E    male     1     1
## 6      A female     1     4
## 7      B female     1     4
## 8      C female     1     4
## 9      D female     1     0
## 10     E female     1     1
## 11     A    male     2     3
## 12     B    male     2     3
## 13     C    male     2     3
## 14     D    male     2     8
## 15     E    male     2     2
## 16     A female     2     4
## 17     B female     2     5
## 18     C female     2     8
## 19     D female     2     1
## 20     E female     2     7
```

spread: variables are stored in both rows and columns

```
students3 <- data.frame(name = c("Sally", "Sally", "Jeff", "Jeff", "Roger", "Roger", "Bree", "Bree", "Brian", "B"),
  students3 %>%
    gather(key = class, value = grade, class1:class5 , na.rm = TRUE) %>%
    spread(key = test , value = grade) %>%
    mutate(class = parse_number(class)) %>%
    print
```

```
##      name class final midterm
## 1    Bree     3      C        C
## 2    Bree     4      A        A
## 3   Brian     1      B        B
## 4   Brian     5      C        A
## 5    Jeff     2      E        D
## 6    Jeff     4      C        A
## 7   Roger     2      A        C
## 8   Roger     5      A        B
## 9   Sally     1      C        A
## 10  Sally     3      C        B
```

Multiple observational units stored in same table

```
students4 <- data.frame(id = c(168,168,588,588,710,710,731,731,908,908), name = c("Sally", "Sally", "Jeff", "Jeff", "Roger", "Roger", "Sally", "Sally"),
  student_info <- students4 %>%
    select(id, name, sex) %>%
    unique %>%
    print
```

```
##      id name sex
## 1 168 Sally  F
## 3 588  Jeff  M
## 5 710 Roger  M
```



```
## 7 731 Bree F
## 9 908 Brian M
```

```
gradebook <- students4 %>%
  select(id, class, midterm, final) %>%
  print
```

```
##      id class midterm final
## 1  168     1       B      B
## 2  168     5       A      C
## 3  588     1       A      C
## 4  588     3       B      C
## 5  710     2       D      E
## 6  710     4       A      C
## 7  731     2       C      A
## 8  731     5       B      A
## 9  908     3       C      C
## 10 908     4       A      A
```

```
# single observational unit is stored in multiple tables
passed <- data.frame(name = c("Brian","Roger","Roger","Karen"), class = c(1,2,5,4), final = c("B","A","A","A"),
failed <- data.frame(name = c("Brian","Sally","Sally","Jeff","Jeff","Karen"), class = c(5,1,3,2,4,3), final = c("C","C","C","E","C","C"),
failed <- mutate(failed, status = "failed")
passed <- mutate(passed, status = "passed")
bind_rows(passed, failed)
```

```
##      name class final status
## 1  Brian     1      B passed
## 2  Roger     2      A passed
## 3  Roger     5      A passed
## 4  Karen     4      A passed
## 5  Brian     5      C failed
## 6  Sally     1      C failed
## 7  Sally     3      C failed
## 8   Jeff     2      E failed
## 9   Jeff     4      C failed
## 10 Karen     3      C failed
```

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