SIOB 296 Introduction to Programming with R

Eric Archer (eric.archer@noaa.gov)

Week 3: April 18, 2017

Reading

The Book of R:
Chapter 1.3
Chapter 8
Chapter 13

The Art of R:
Chapter 8: 189-192, 194-195
Chapter 9.4
Chapter 10.1, 10.2

Directories and files

Working directory

The working directory in R is the default location where files are written to and read from. To see where that currently is, use `getwd()`

```
getwd()
```

```
[1] "/Users/Shared/Work/R.Stuff/SIO Course/Introduction to R/Prep"
```

The working directory can be changed programmatically with `setwd()`, and a character vector of the directory contains viewed with `dir()`:

```
# The contents of this directory
dir()
```

```
 [1] "ctd.csv"
 [2] "Flow Control.Rmd"
 [3] "format ctd.R"
 [4] "Functions.Rmd"
 [5] "multiYearCTD.csv"
 [6] "SIOB 296 - Intro to R - Syllabus.Rmd"
 [7] "Week 1.Rmd"
 [8] "Week 2.Rmd"
 [9] "Week 3.Rmd"
[10] "Week_3.pdf"
[11] "Week_3.Rmd"
```

```
# Move up a directory
setwd("..")
# Show the contents of this directory
dir()
```

```
[1] "Prep"
[2] "README.md"
[3] "SIO 296 - Introduction to R.Rproj"
[4] "SIOB_296_-_Intro_to_R_-_Syllabus.pdf"
[5] "Week 01 - April 04"
[6] "Week 02 - April 11"
[7] "Week 03 - April 18"
```

The `pattern` argument of `dir()` allows you to filter the files that are returned:

```
dir(pattern = "Week")
```

```
[1] "Week 1.Rmd" "Week 2.Rmd" "Week 3.Rmd" "Week_3.pdf" "Week_3.Rmd"
```

Reading and writing data

R workspaces (.Rdata): `save`, `load`

The entire workspace can be saved to disk with `save.image()`. R workspace/object files are binary files that cannot be read by anything but R. They usually end in ".rdata", or ".rda".

```
rm(list = ls())
x <- 1
y <- 2
z <- 3
save.image(file = "test ws.rdata")
```

The file can be read back into the workspace with `load()`:

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

```
character(0)
```

```
load("test ws.rdata")
ls()
```

```
[1] "x" "y" "z"
```

Individual objects can be saved with `save()`:

```
save(x, y, file = "xy.rdata")
rm(list = ls())
load("xy.rdata")
ls()
```

```
[1] "x" "y"
```

Text tables (.csv): `write.table`, `read.table`

Data in tabular format, such as matrices or data frames are saved to and read from disk with `write.table` and `read.table` and their wrappers, most commonly `write.csv` and `read.csv`:

```
x <- data.frame(nums = 51:60, lets = letters[1:10])
write.csv(x, file = "test.csv")
rm(list = ls())
df <- read.csv("test.csv")
df
```

```
    X nums lets
1   1   51    a
2   2   52    b
3   3   53    c
4   4   54    d
5   5   55    e
6   6   56    f
7   7   57    g
8   8   58    h
9   9   59    i
10 10   60    j
```

You'll notice that there is a new column, "X" that has the numbers 1-10 in it. This is because by default, `write.csv` writes a file with the rownames in the first column. To change this behavior, set the argument `row.nams = FALSE` in `write.csv`.

```
x <- data.frame(nums = 51:60, lets = letters[1:10])
write.csv(x, file = "test.csv", row.names = FALSE)
rm(list = ls())
df <- read.csv("test.csv")
df
```

```
   nums lets
1    51    a
2    52    b
3    53    c
4    54    d
5    55    e
6    56    f
7    57    g
8    58    h
9    59    i
10   60    j
```

```
str(df)
```

```
'data.frame':   10 obs. of  2 variables:
 $ nums: int  51 52 53 54 55 56 57 58 59 60
 $ lets: Factor w/ 10 levels "a","b","c","d",..: 1 2 3 4 5 6 7 8 9 10
```

Also, notice that the `lets` column is read in as a factor. This is the default behavior of `read.csv` and can be changed with the `stringsAsFactors` argument:

```
df <- read.csv("test.csv", stringsAsFactors = FALSE)
str(df)
```

```
'data.frame':   10 obs. of  2 variables:
 $ nums: int  51 52 53 54 55 56 57 58 59 60
 $ lets: chr  "a" "b" "c" "d" ...
```

Free text (.txt): `write`, `scan`

Free text can be written and read with `write` and `scan`. With `write`, one line is written per call and the `append` argument is used to add to an existing file:

```
fname <- "free text.txt"
write("Hello, I am the first line", file = fname)
```

```
write("...and I am the second line in the file", file = fname, append = TRUE)
write("I'll be the third line to end it all", file = fname, append = TRUE)
```

`scan` will read a text file into a vector of a type specified by the `what` argument. See the Details section in `?scan` for more info. To read the text file above:

```
rm(list = ls())
x <- scan("free text.txt", what = "character")
x
```

```
 [1] "Hello," "I"      "am"     "the"    "first"  "line"   "...and"
 [8] "I"      "am"     "the"    "second" "line"   "in"     "the"
[15] "file"   "I'll"   "be"     "the"    "third"  "line"   "to"
[22] "end"    "it"     "all"
```

```
# here the delimiter is the end of line character ("\n") so each line is a single element in the return
z <- scan("free text.txt", what = "character", sep = "\n")
z
```

```
[1] "Hello, I am the first line"
[2] "...and I am the second line in the file"
[3] "I'll be the third line to end it all"
```

R scripts (.r): `dump`, `source`

R objects can be written to files as a form of the code used to create them using `dump`. These files usually end in ".R"

```
x <- matrix(1:24, nrow = 4)
dump("x", file = "x.r")
```

These files can be read back in using `source`.

```
rm(list = ls())
source("x.r")
ls()
```

```
[1] "x"
```

`source` is used to execute R commands stored in text files. It is the command you will use to execute saved scripts.

Writing and running scripts

Scripts are text files containing commands and comments written in an order as if they were executed on the command line. They can be executed with `source("filename.r")`, or if loaded into an R editor, run piece by piece or all together. In RStudio, see commands and shortcuts under the Code menu option.
Code style is an important habit to cultivate. Being consistent in your syntax, spacing, and naming will help you create, edit, and understand your code later. There are many good style guides that you can follow. Feel free to mix and match from them choosing what works best for you. Here are a few:

- Google's: https://google.github.io/styleguide/Rguide.xml
- Hadley Wickham's: http://adv-r.had.co.nz/Style.html
- https://csgillespie.wordpress.com/2010/11/23/r-style-guide/
- http://jef.works/R-style-guide/

Object summary

The content of objects can be viewed by simply typing the object name as we've seen before:

```
x <- matrix(1:12, nrow = 3)
x
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

This can also be done with the `print` function:

```
print(x)
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

If the object is big, use `head` to see the first few items and/or `tail` to see the last few items:

```
x <- matrix(1:1000, nrow = 100)
head(x, 10)
```

```
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
 [1,]    1  101  201  301  401  501  601  701  801   901
 [2,]    2  102  202  302  402  502  602  702  802   902
 [3,]    3  103  203  303  403  503  603  703  803   903
 [4,]    4  104  204  304  404  504  604  704  804   904
 [5,]    5  105  205  305  405  505  605  705  805   905
 [6,]    6  106  206  306  406  506  606  706  806   906
 [7,]    7  107  207  307  407  507  607  707  807   907
 [8,]    8  108  208  308  408  508  608  708  808   908
 [9,]    9  109  209  309  409  509  609  709  809   909
[10,]   10  110  210  310  410  510  610  710  810   910
```

```
tail(x, 4)
```

```
       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
 [97,]   97  197  297  397  497  597  697  797  897   997
 [98,]   98  198  298  398  498  598  698  798  898   998
 [99,]   99  199  299  399  499  599  699  799  899   999
[100,]  100  200  300  400  500  600  700  800  900  1000
```

The best summary of an object's structure and contents is the `str` function:

```
str(x)
```

```
 int [1:100, 1:10] 1 2 3 4 5 6 7 8 9 10 ...
```

To summarize values the values (distribution, etc.), use the `summary` function:

```
summary(x)
```

```
       V1              V2              V3              V4
 Min.   :  1.00   Min.   :101.0   Min.   :201.0   Min.   :301.0
 1st Qu.: 25.75   1st Qu.:125.8   1st Qu.:225.8   1st Qu.:325.8
 Median : 50.50   Median :150.5   Median :250.5   Median :350.5
 Mean   : 50.50   Mean   :150.5   Mean   :250.5   Mean   :350.5
```

```
 3rd Qu.: 75.25   3rd Qu.:175.2   3rd Qu.:275.2   3rd Qu.:375.2
 Max.   :100.00   Max.   :200.0   Max.   :300.0   Max.   :400.0
       V5              V6              V7              V8
 Min.   :401.0   Min.   :501.0   Min.   :601.0   Min.   :701.0
 1st Qu.:425.8   1st Qu.:525.8   1st Qu.:625.8   1st Qu.:725.8
 Median :450.5   Median :550.5   Median :650.5   Median :750.5
 Mean   :450.5   Mean   :550.5   Mean   :650.5   Mean   :750.5
 3rd Qu.:475.2   3rd Qu.:575.2   3rd Qu.:675.2   3rd Qu.:775.2
 Max.   :500.0   Max.   :600.0   Max.   :700.0   Max.   :800.0
       V9             V10
 Min.   :801.0   Min.   : 901.0
 1st Qu.:825.8   1st Qu.: 925.8
 Median :850.5   Median : 950.5
 Mean   :850.5   Mean   : 950.5
 3rd Qu.:875.2   3rd Qu.: 975.2
 Max.   :900.0   Max.   :1000.0
```

Missing data (NAs)

Missing data is denoted in R with `NA` and has to be explicitly tested for and handled specially. To test if values are equal to NA, you can't use `==`, you have to use `is.na()`

```
x <- c(1, NA, 3, 6, NA)
x == NA
```

```
[1] NA NA NA NA NA
```

```
is.na(x)
```

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

To remove `NA`s from a vector, use `na.omit()`:

```
x2 <- na.omit(x)
x2
```

```
[1] 1 3 6
attr(,"na.action")
[1] 2 5
attr(,"class")
[1] "omit"
```

```
str(x2)
```

```
 atomic [1:3] 1 3 6
 - attr(*, "na.action")=Class 'omit'  int [1:2] 2 5
```

To identify rows in a data frame without `NA`s, use `complete.cases`:

```
mat <- rbind(
  sample(1:5, 8, replace = TRUE),
  sample(1:5, 8, replace = TRUE),
  sample(1:5, 8, replace = TRUE),
  sample(c(NA, 1:5), 8, replace = TRUE),
  sample(c(NA, 1:5), 8, replace = TRUE),
  sample(c(NA, 1:5), 8, replace = TRUE),
  sample(c(NA, 1:5), 8, replace = TRUE),
  sample(c(NA, 1:5), 8, replace = TRUE),
```

```
  sample(c(NA, 1:5), 8, replace = TRUE)
)
mat
```

```
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
 [1,]    3    3    4    5    5    1    2    2
 [2,]    4    4    4    5    2    2    3    5
 [3,]    3    4    4    3    2    3    2    2
 [4,]    3    5    2    1    1    3    2    2
 [5,]    5   NA    3    4    2    3    1    4
 [6,]    1    3   NA    1    3    2   NA    1
 [7,]    2    5    3    5    2    3    3    5
 [8,]    2    5    5    5    4   NA    2    3
 [9,]    2    4    3    1    1    5    2    4
```

```
i <- complete.cases(mat)
i
```

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

```
mat[i, ]
```

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

Math summaries

To get the range of a vector of numerics, use `min`, `max`, and `range`:

```
x <- sample(1:100, 10)
x
```

```
 [1] 16 11 96 67 92 81  2 19 98 77
```

```
min(x)
```

```
[1] 2
```

```
max(x)
```

```
[1] 98
```

```
range(x)
```

```
[1]  2 98
```

Sums and products of vectors can be calculated:

```
sum(x)
```

```
[1] 559
```

```
prod(x)
```

```
[1] 2.418978e+15
```

To calculate the difference between values with a given lag, use `diff`:

```
diff(x)
```

```
[1]  -5  85 -29  25 -11 -79  17  79 -21
```

```
diff(x, lag = 3)
```

```
[1]  51  81 -15 -65 -73  17  75
```

Other numeric summaries such as the median, mean, variance, and standard deviation are available:

```
median(x)
```

```
[1] 72
```

```
mean(x)
```

```
[1] 55.9
```

```
var(x)
```

```
[1] 1528.544
```

```
sd(x)
```

```
[1] 39.0966
```

Any set of quantiles can be calculated with the `quantiles` function:

```
x <- sample(1:1000, 100)
quantile(x, probs = c(0.025, 0.05, 1/3, 0.5, 0.99))
```

```
     2.5%        5% 33.33333%       50%       99% 
    18.90     29.75    295.00    430.00    950.32 
```

Discrete values

The function `unique()` will list the unique values in a vector in the order it finds them:

```
x <- sample(letters, 10, replace = TRUE)
x
```

```
 [1] "a" "h" "m" "g" "m" "h" "s" "v" "z" "e"
```

```
unique(x)
```

```
[1] "a" "h" "m" "g" "s" "v" "z" "e"
```

The function `duplicated()` will identify those elements in a vector that occur at an earlier position:

```
duplicated(x)
```

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

```
# the negation of duplicated is the same as unique
x[!duplicated(x)]
```

```
[1] "a" "h" "m" "g" "s" "v" "z" "e"
```

```
unique(x)
```

```
[1] "a" "h" "m" "g" "s" "v" "z" "e"
```

To calculate the frequency of values in a vector (the number of occurences), use `table()`:

```
x <- sample(letters, 20, replace = TRUE)
table(x)
```

```
x
a b c e f h j l n o p q t u v y
1 1 2 1 1 1 2 1 1 1 1 1 2 2 1 1
```

`table` can be used for cross-tabulation as well - counting frequency of occurence of a combination of categories

```
months <- sample(month.abb, 20, replace = TRUE)
sex <- sample(c("m", "f"), 20, replace = TRUE)
freq <- table(sex, months)
freq
```

```
   months
sex Apr Dec Feb Jan Jul May Nov Oct
  f   0   2   1   1   1   2   1   1
  m   4   0   1   1   1   3   1   0
```

The values in a table can be accessed like a vector or matrix

```
freq["m", ]
```

```
Apr Dec Feb Jan Jul May Nov Oct
  4   0   1   1   1   3   1   0
```

```
freq["f", c("Jan", "Feb", "Mar")]
```

```
Error in `[.default`(freq, "f", c("Jan", "Feb", "Mar")): subscript out of bounds
```

Data selection and manipulation

To identify values of one vector that are within another one, use `%in%`:

```
letters %in% c("a", "f", "G", "b")
```

```
 [1]  TRUE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE
[12] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[23] FALSE FALSE FALSE FALSE
```

To identify values of a logical vector that are `TRUE`, use `which`:

```
x <- sample(1:100, 20)
x
```

```
 [1] 71 57 16 79 90 40 65 98  9 93  3 47 58 17 13 18 69 33 83 97
```

```
which(x < 50)
```

```
[1]  3  6  9 11 12 14 15 16 18
```

To identify the minimum and maximum values, use `which.min` and `which.max`:

```
which.min(x)
```

```
[1] 11
```

```
which.max(x)
```

```
[1] 8
```

To see if any values meet a condition, use `any`:

```
any(x < 50)
```

```
[1] TRUE
```

```
any(x > 200)
```

```
[1] FALSE
```

To see if all values meet a condition, use `all`:

```
all(x < 50)
```

```
[1] FALSE
```

```
all(x < 200)
```

```
[1] TRUE
```

Vectors can be reversed with `rev`:

```
x <- sample(1:5, 10, replace = T)
x
```

```
 [1] 5 5 2 2 5 5 3 5 5 2
```

```
rev(x)
```

```
 [1] 2 5 5 3 5 5 2 2 5 5
```

and sorted with `sort`:

```
sort(x)
```

```
 [1] 2 2 2 3 5 5 5 5 5 5
```

```
# in decreasing order
sort(x, decreasing = TRUE)
```

```
 [1] 5 5 5 5 5 5 3 2 2 2
```

However, sort can't be applied to a matrix or data.frame to sort the rows. For that, you need `order`. `order` returns a vector of indices in the order they should be as if they were sorted:

```
x <- data.frame(
  v1 = sample(letters, 20, replace = TRUE),
  v2 = sample(letters, 20, replace = TRUE),
  v3 = sample(letters, 20, replace = TRUE)
)
x
```

```
   v1 v2 v3
1   e  b  d
2   x  b  f
3   y  t  j
4   d  r  n
5   d  x  g
6   n  x  f
7   r  t  t
8   g  n  f
9   e  n  o
10  y  o  n
```

```
11  t  g  b
12  l  v  i
13  f  c  h
14  n  s  p
15  f  x  o
16  w  l  j
17  o  x  q
18  y  f  o
19  f  w  f
20  i  w  k
```

```
x.ord <- order(x$v1)
x[x.ord, ]
```

```
   v1 v2 v3
4   d  r  n
5   d  x  g
1   e  b  d
9   e  n  o
13  f  c  h
15  f  x  o
19  f  w  f
8   g  n  f
20  i  w  k
12  l  v  i
6   n  x  f
14  n  s  p
17  o  x  q
7   r  t  t
11  t  g  b
16  w  l  j
2   x  b  f
3   y  t  j
10  y  o  n
18  y  f  o
```

```
# also in decreasing order
x[order(x$v1, decreasing = TRUE), ]
```

```
   v1 v2 v3
3   y  t  j
10  y  o  n
18  y  f  o
2   x  b  f
16  w  l  j
11  t  g  b
7   r  t  t
17  o  x  q
6   n  x  f
14  n  s  p
12  l  v  i
20  i  w  k
8   g  n  f
13  f  c  h
15  f  x  o
```

```
19  f  w  f
1   e  b  d
9   e  n  o
4   d  r  n
5   d  x  g
```

`order` can take several vectors to do hierarchical sorting.

```
i <- order(x$v2, x$v1, x$v3)
i
```

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

```
x[i, ]
```

```
   v1 v2 v3
1   e  b  d
2   x  b  f
13  f  c  h
18  y  f  o
11  t  g  b
16  w  l  j
9   e  n  o
8   g  n  f
10  y  o  n
4   d  r  n
14  n  s  p
7   r  t  t
3   y  t  j
12  l  v  i
19  f  w  f
20  i  w  k
5   d  x  g
15  f  x  o
6   n  x  f
17  o  x  q
```

Binning values

To create bins of a continuous variable, the `cut` function is very handy. It has several arguments that regulate how the binning is to be done that are worth examining:

```
y <- c(4, 5, 6, 10, 11, 30, 49, 50, 51)

# We want the following bins : 5 > y <= 10, 10 > y <= 30, 30 > y <= 50
y.cut <- cut(y, breaks = c(5, 10, 30, 50))
y.cut
```

```
[1] <NA>    <NA>    (5,10]  (5,10]  (10,30] (10,30] (30,50] (30,50] <NA>
Levels: (5,10] (10,30] (30,50]
```

```
str(y.cut)
```

```
 Factor w/ 3 levels "(5,10]","(10,30]",..: NA NA 1 1 2 2 3 3 NA
```

A factor is created that replaces the values with the selected bins. The bins labels use the parentheses ("`(`" and "`)`") to denote that the value is not included in the bin, while the brackets ("`[`" and "`]`") denote

that the value is included. Let's change the binning, so that 5 (the lowest bin value) is included, using `include.lowest = TRUE`:

```
# Bins : 5 >= y <= 10, 10 > y <= 30, 30 > y <= 50
cut(y, breaks = c(5, 10, 30, 50), include.lowest = TRUE)
```

```
[1] <NA>    [5,10]  [5,10]  [5,10]  (10,30] (10,30] (30,50] (30,50] <NA>
Levels: [5,10] (10,30] (30,50]
```

By including the argument `right = FALSE`, the default binning is flipped so that the lowest value is included, but the highest is not:

```
# Bins : 5 >= y < 10, 10 >= y < 30, 30 >= y < 50
cut(y, breaks = c(5, 10, 30, 50), right = FALSE)
```

```
[1] <NA>    [5,10)  [5,10)  [10,30) [10,30) [30,50) [30,50) <NA>    <NA>
Levels: [5,10) [10,30) [30,50)
```

Including both `include.lowest` and `right` causes all bin values to be included:

```
# Bins : 5 >= y < 10, 10 >= y < 30, 30 >= y <= 50
cut(y, breaks = c(5, 10, 30, 50), include.lowest = TRUE, right = FALSE)
```

```
[1] <NA>    [5,10)  [5,10)  [10,30) [10,30) [30,50] [30,50] [30,50] <NA>
Levels: [5,10) [10,30) [30,50]
```

Merging data frames

Two data frames can be merged (or "joined" if you're used to SQL database syntax) using the `merge` function. With `merge`, you have to specify the columns in common between the data frames used to identify equivalent rows (the `by` argument(s)), as well as whether or not to include all of one data frame or both (the `all` argument(s)).

```
# set up two data.frames using the built-in states data (see ?state for more info)
st.data <- as.data.frame(state.x77)
st.data$name <- rownames(st.data)
str(st.data)
```

```
'data.frame':   50 obs. of  9 variables:
 $ Population: num  3615 365 2212 2110 21198 ...
 $ Income    : num  3624 6315 4530 3378 5114 ...
 $ Illiteracy: num  2.1 1.5 1.8 1.9 1.1 0.7 1.1 0.9 1.3 2 ...
 $ Life Exp  : num  69 69.3 70.5 70.7 71.7 ...
 $ Murder    : num  15.1 11.3 7.8 10.1 10.3 6.8 3.1 6.2 10.7 13.9 ...
 $ HS Grad   : num  41.3 66.7 58.1 39.9 62.6 63.9 56 54.6 52.6 40.6 ...
 $ Frost     : num  20 152 15 65 20 166 139 103 11 60 ...
 $ Area      : num  50708 566432 113417 51945 156361 ...
 $ name      : chr  "Alabama" "Alaska" "Arizona" "Arkansas" ...
```

```
st.grp <- data.frame(name = state.name, region = state.region, division = state.division)
str(st.grp)
```

```
'data.frame':   50 obs. of  3 variables:
 $ name    : Factor w/ 50 levels "Alabama","Alaska",..: 1 2 3 4 5 6 7 8 9 10 ...
 $ region  : Factor w/ 4 levels "Northeast","South",..: 2 4 4 2 4 4 1 2 2 2 ...
 $ division: Factor w/ 9 levels "New England",..: 4 9 8 5 9 8 1 3 3 3 ...
```

```r
# extract just the Pacific states from the st.grp data frame and merge with
#   the st.data, keeping only the Pacific state rows
pac <- st.grp[st.grp$division == "Pacific", ]
pac.data <- merge(pac, st.data, by = "name", all.x = TRUE)
pac.data
```

```
        name region division Population Income Illiteracy Life Exp Murder
1     Alaska   West  Pacific        365   6315        1.5    69.31   11.3
2 California   West  Pacific      21198   5114        1.1    71.71   10.3
3     Hawaii   West  Pacific        868   4963        1.9    73.60    6.2
4     Oregon   West  Pacific       2284   4660        0.6    72.13    4.2
5 Washington   West  Pacific       3559   4864        0.6    71.72    4.3
  HS Grad Frost   Area
1    66.7   152 566432
2    62.6    20 156361
3    61.9     0   6425
4    60.0    44  96184
5    63.5    32  66570
```

```r
# merge using columns of different names. keep all rows - NAs where no match
st.data$state.name <- st.data$name
st.data$name <- NULL
pac.all <- merge(pac, st.data, by.x = "name", by.y = "state.name", all.y = TRUE)
head(pac.all)
```

```
        name region division Population Income Illiteracy Life Exp Murder
1    Alabama   <NA>     <NA>       3615   3624        2.1    69.05   15.1
2     Alaska   West  Pacific        365   6315        1.5    69.31   11.3
3    Arizona   <NA>     <NA>       2212   4530        1.8    70.55    7.8
4   Arkansas   <NA>     <NA>       2110   3378        1.9    70.66   10.1
5 California   West  Pacific      21198   5114        1.1    71.71   10.3
6   Colorado   <NA>     <NA>       2541   4884        0.7    72.06    6.8
  HS Grad Frost   Area
1    41.3    20  50708
2    66.7   152 566432
3    58.1    15 113417
4    39.9    65  51945
5    62.6    20 156361
6    63.9   166 103766
```