SIOB 296 Introduction to Programming with R

Eric Archer (eric.archer@noaa.gov)

Week 4: April 25, 2017

Reading

The Book of R:
Chapter 4.2: Characters

The Art of R:
Chapter 11.1, 11.2: String Manipulation
Chapter 3.3: Applying Functions to Matrix Rows and Columns
Chapter 6.2: Common Functions Used with Factors

Merging data frames

To merge two data frames based on common columns, use the `merge` function. To demonstrate how it works, we'll load several simple data frames that have data about a related set of specimens:

```
spec.dfs <- load("merge data.rdata")
print(spec.dfs)
```

```
[1] "specimens" "cranial"   "haps"      "sex"       "trials"
```

```
# Skull measurements
str(cranial)
```

```
'data.frame':   20 obs. of  2 variables:
 $ id   : chr  "Specimen-25" "Specimen-37" "Specimen-4" "Specimen-1" ...
 $ skull: num  257 251 261 256 259 ...
```

```
# Haplotypes
str(haps)
```

```
'data.frame':   20 obs. of  2 variables:
 $ id  : chr  "Specimen-7" "Specimen-31" "Specimen-1" "Specimen-32" ...
 $ haps: chr  "E" "C" "D" "E" ...
```

```
# Sex
str(sex)
```

```
'data.frame':   40 obs. of  2 variables:
 $ specimens: chr  "Specimen-1" "Specimen-2" "Specimen-3" "Specimen-4" ...
 $ sex      : chr  "M" "F" "M" "F" ...
```

```
# Results of multiple trials
str(trials)
```

```
'data.frame':   30 obs. of  2 variables:
 $ id   : chr  "Specimen-27" "Specimen-32" "Specimen-13" "Specimen-29" ...
 $ value: num  20.3 23.6 20.3 22.2 38.1 ...
```

`merge` requires two data frames as its first two arguments, `x` and `y`. The two primary sets of arguments that control how the merging happens are `by` (with the related `by.x` and `by.y`) which identifies which column(s) are used to identify equivalent records to be merged, and `all` (with the related `all.x` and `all.y`) which specifies whether the data frame returned contains all records from both data.frames or just all records from one or the other. As an example, we`ll merge the skull measurement and haplotype data frames. They have the same number of rows, but not the exact same specimens. Because they both have a column in common (id), by default, it is used for`by‘:

```
merge(cranial, haps)
```

```
            id    skull haps
1   Specimen-1 255.7784    D
2  Specimen-10 260.8000    B
3  Specimen-15 262.2992    E
4  Specimen-21 255.7237    A
5  Specimen-25 257.3424    D
6  Specimen-29 271.5665    C
7  Specimen-32 263.1267    E
8  Specimen-37 251.4024    E
9   Specimen-4 261.2716    A
10  Specimen-6 264.0681    B
```

Note that there are only 10 rows because by default, the `all` argument is set to `FALSE` which means the function will only return a data frame with ids that occur in both original data frames. If we want to return a data frame with all specimens in both original data frames, we set `all = TRUE`:

```
merge(cranial, haps, all = TRUE)
```

```
            id    skull haps
1   Specimen-1 255.7784    D
2  Specimen-10 260.8000    B
3  Specimen-11       NA    E
4  Specimen-12 261.7318 <NA>
5  Specimen-13       NA    A
6  Specimen-15 262.2992    E
7  Specimen-17 264.1644 <NA>
8  Specimen-18       NA    B
9  Specimen-19 255.9717 <NA>
10 Specimen-21 255.7237    A
11 Specimen-22       NA    B
12 Specimen-23 270.7938 <NA>
13 Specimen-24 264.6283 <NA>
14 Specimen-25 257.3424    D
15 Specimen-26 260.7033 <NA>
16 Specimen-28 264.2351 <NA>
17 Specimen-29 271.5665    C
18 Specimen-31       NA    C
19 Specimen-32 263.1267    E
20 Specimen-33 261.7152 <NA>
21 Specimen-34 253.3402 <NA>
22 Specimen-35       NA    C
23 Specimen-36 258.5202 <NA>
24 Specimen-37 251.4024    E
25 Specimen-38       NA    B
26  Specimen-4 261.2716    A
27  Specimen-5       NA    D
```

```
28  Specimen-6 264.0681    B
29  Specimen-7       NA    E
30  Specimen-9       NA    D
```

Note that here, `NA`s are inserted where there is no data in the other data frame. We can also specify that we only want all records in one data frame:

```r
merge(cranial, haps, all.x = TRUE)
```

```
            id    skull haps
1   Specimen-1 255.7784    D
2  Specimen-10 260.8000    B
3  Specimen-12 261.7318 <NA>
4  Specimen-15 262.2992    E
5  Specimen-17 264.1644 <NA>
6  Specimen-19 255.9717 <NA>
7  Specimen-21 255.7237    A
8  Specimen-23 270.7938 <NA>
9  Specimen-24 264.6283 <NA>
10 Specimen-25 257.3424    D
11 Specimen-26 260.7033 <NA>
12 Specimen-28 264.2351 <NA>
13 Specimen-29 271.5665    C
14 Specimen-32 263.1267    E
15 Specimen-33 261.7152 <NA>
16 Specimen-34 253.3402 <NA>
17 Specimen-36 258.5202 <NA>
18 Specimen-37 251.4024    E
19  Specimen-4 261.2716    A
20  Specimen-6 264.0681    B
```

If the common column is not the same in both data frames, you have to specify it with `by.x` and `by.y`:

```r
merged.df <- merge(sex, trials, by.x = "specimens", by.y = "id")
head(merged.df)
```

```
    specimens sex    value
1 Specimen-11   M 28.29490
2 Specimen-12   M 36.97172
3 Specimen-12   M 25.35317
4 Specimen-13   M 26.55486
5 Specimen-13   M 20.30126
6 Specimen-14   F 31.92108
```

Here, the identifier column name of the `x` data.frame is retained.

Character and string manipulation

nchar

A character vector is a vector where every element is a character string of any length. The `length()` of a character vector is the number of elements in it:

```r
x <- c("This is a sentence", "Hello World!", "This is the third element")
length(x)
```

```
[1] 3
```

To get the number of characters in each element, use `nchar()`:

```
nchar(x)
```

```
[1] 18 12 25
```

substr

Strings can be extracted from elements using `substr()`. You specify the first and last characters to be extracted from each string:

```
# get the first three characters from every string
substr(x, 1, 3)
```

```
[1] "Thi" "Hel" "Thi"
```

```
# get the 3rd character from every string
substr(x, 3, 3)
```

```
[1] "i" "l" "i"
```

`substr` can also be used to replace values within strings by assigning:

```
substr(x, 1, 4) <- "That"
x
```

```
[1] "That is a sentence"        "Thato World!"
[3] "That is the third element"
```

strsplit

Strings can be split based on some common delimiter using `strsplit()`:

```
# split based on spaces
x.split <- strsplit(x, " ")
x.split
```

```
[[1]]
[1] "That"     "is"       "a"        "sentence"

[[2]]
[1] "Thato"  "World!"

[[3]]
[1] "That"    "is"      "the"     "third"   "element"
```

```
str(x.split)
```

```
List of 3
 $ : chr [1:4] "That" "is" "a" "sentence"
 $ : chr [1:2] "Thato" "World!"
 $ : chr [1:5] "That" "is" "the" "third" ...
```

Note that the return value from `strsplit` is a list. Each element in the list corresponds to a vector resulting from splitting every element in the orginal vector

```
x.split[[1]]
```

```
[1] "That"     "is"       "a"        "sentence"
```

paste

To create strings from combinations of strings (or numbers) we use `paste()`. This function takes a set of vectors, and pastes the elements together using recycling:

```
# vectors are equal length
paste(letters[1:6], 1:6)
```

```
[1] "a 1" "b 2" "c 3" "d 4" "e 5" "f 6"
```

```
# one vector is a multiple of the other
paste(letters[1:6], 1:2)
```

```
[1] "a 1" "b 2" "c 1" "d 2" "e 1" "f 2"
```

```
# one vector is not a multiple of the other
paste(letters[1:6], 1:4)
```

```
[1] "a 1" "b 2" "c 3" "d 4" "e 1" "f 2"
```

The argument `sep` determines what character is used as a separator between the characters:

```
paste(letters[1:6], 1:2, sep = "-")
```

```
[1] "a-1" "b-2" "c-1" "d-2" "e-1" "f-2"
```

If you do not want a separator character, either set `sep = ""` or use `paste0()`:

```
paste0(letters[1:6], 1:2)
```

```
[1] "a1" "b2" "c1" "d2" "e1" "f2"
```

If you want to paste all of the arguments to create a single element vector, set the `collapse` argument:

```
paste(letters[1:6], 1:2, sep = "-", collapse = "#")
```

```
[1] "a-1#b-2#c-1#d-2#e-1#f-2"
```

tolower, toupper

Character case can be changed with `tolower` and `toupper`:

```
tolower(x)
```

```
[1] "that is a sentence"        "thato world!"
[3] "that is the third element"
```

```
toupper(x)
```

```
[1] "THAT IS A SENTENCE"        "THATO WORLD!"
[3] "THAT IS THE THIRD ELEMENT"
```

Regular Expressions

For finer control on searching and replacing text within strings, you will have to turn to "regular expressions", which is a kind of syntax of its own and is common across several platforms. The help page for regular expressions in R is `?regex`. The functions that are most commonly used with regular expressions are given in `grep`. The most commonly used on this page are:

`grep` and `grepl`: Identify elements that have the sought after pattern `sub` and `gsub`: Replace a desired pattern with other text

```r
x <- c("Here is some text", "This is more text", "I have the number 1", "22 is the number I have")
# which elements have the word "text"?
grep("text", x)
```

```
[1] 1 2
```

```r
# which elements have numbers?
grep("[[:digit:]]", x)
```

```
[1] 3 4
```

```r
# replace the word "This" with "That"
gsub("This", "That", x)
```

```
[1] "Here is some text"       "That is more text"
[3] "I have the number 1"     "22 is the number I have"
```

apply Functions

Many times, we want to execute the same function on sequential elements of some object. This could be things like the elements of a vector or list, the rows of a matrix, or the columns of a data frame. For these, R provides a family of functions that usually end in `-apply` or are based on them.

lapply

The most basic of these functions is `lapply`. The "l" refers to the fact that `lapply` will always return a list. There are two main arguments to `lapply`: the first is the object to be iterated over, and the second is a function that takes sequential elements of that object. As an example, let's use the `sample` function. Recall that if you execute `sample` with a single integer(`n`), it will return a permutation of the vector `1:n`:

```r
sample(5)
```

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

```r
sample(10)
```

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

Here is a list resulting from calls to sample with the elements of the vector `1:5`:

```r
x <- lapply(1:5, sample)
str(x)
```

```
List of 5
 $ : int 1
 $ : int [1:2] 2 1
 $ : int [1:3] 2 1 3
 $ : int [1:4] 3 1 4 2
 $ : int [1:5] 5 3 2 4 1
```

```r
x
```

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

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

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

[[4]]
[1] 3 1 4 2

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

Note that the result is a list, the elements of which are the result of calls to `sample(1)`, `sample(2)`, `sample(3)`, etc. The elements of the return value are in the same order as the original object being iterated over:

```r
lapply(c(5, 3, 1, 8), sample)
```

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

[[2]]
[1] 3 1 2

[[3]]
[1] 1

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

The first argument can be a list too:

```r
lapply(x, sum)
```

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

[[2]]
[1] 3

[[3]]
[1] 6

[[4]]
[1] 10

[[5]]
[1] 15
```

sapply

If the return value from every iteration was the same length, you may want to simplify the result. This is what `sapply` is for. If every call to the function returns a scalar, then `sapply` will return a vector. If every call to the function returns a vector of equal length, then `sapply` will return a matrix. If every call to the function returns a value of different lengths, then `sapply` defaults to returning a list:

```r
# every return value from sum is a scalar - sapply returns a vector
sapply(x, sum)
```

```
[1]  1  3  6 10 15
```

```r
# every return value from sample is a 5 element vector - sapply returns a matrix
sapply(rep(5, 8), sample)
```

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

```r
# this is the same as our lapply example - sapply returns a list
sapply(c(5, 3, 1, 8), sample)
```

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

[[2]]
[1] 1 2 3

[[3]]
[1] 1

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

Arguments to the function can be specified in the `lapply` or `sapply` call:

```r
sapply(c(5, 3, 1, 8), sample, size = 5, replace = TRUE)
```

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

apply

If you are dealing with a multi-dimensional object (matrix, array, or data frame) and you want to apply a function to a given dimension (i.e, each row or each column), use `apply`. You have to specify the dimension that you will be iterating over as the second argument (1 = rows, 2 = columns, etc). `apply` will try to simplify the results like `sapply`:

```r
x <- matrix(sample(1:100, 24, replace = TRUE), nrow = 4)
x
```

```
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]   61   91    6   31   18   38
[2,]   46   81   56   50   45   83
[3,]    3   75   40   51   79    8
[4,]   76   40   94   79   22   47
```

```r
# median of each row
apply(x, 1, median)
```

```
[1] 34.5 53.0 45.5 61.5
```

```r
# difference of each column
apply(x, 2, diff)
```

```
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]  -15  -10   50   19   27   45
[2,]  -43   -6  -16    1   34  -75
[3,]   73  -35   54   28  -57   39
```

tapply

If you want to execute a function on groups of values, `tapply` can often be a good choice. The arguments are a vector that will be summarized, another vector or set of vectors that represent identify elements to groups, and the function that will get the sequential subsets of the original vector. As a simple example, we calculate the means of subsets of a random vector of numbers

```r
x <- sample(1:100, 100, replace = TRUE)
grp <- sample(letters[1:5], 100, replace = TRUE)
tapply(x, grp, mean)
```

```
       a        b        c        d        e 
44.80000 44.50000 54.61905 66.22222 48.15000 
```

As a more practical example, we can calculate the average temperature at each station in our ctd dataset:

```r
ctd <- read.csv("ctd.csv")
tapply(ctd$temp, ctd$station, mean)
```

```
 Station.1 Station.10 Station.11 Station.12 Station.13 Station.14 
  13.56772   14.57675   14.93466   14.40093   13.84903   14.42424 
Station.15 Station.16 Station.17 Station.18 Station.19  Station.2 
  14.10233   14.42280   14.27389   14.57620   16.32791   14.24618 
Station.20 Station.21 Station.22 Station.23 Station.24 Station.25 
  13.34375   13.77648   14.18930   14.60143   16.35819   15.82101 
Station.26 Station.27 Station.28 Station.29  Station.3 Station.30 
  15.84702   14.10141   13.21872   13.76510   14.23830   14.19600 
Station.31 Station.32 Station.33 Station.34 Station.35 Station.36 
  14.85625   16.73097   14.07543   14.62647   14.74482   15.53890 
Station.37 Station.38 Station.39  Station.4 Station.40  Station.5 
  15.12451   15.57174   15.14841   14.51093   16.37120   14.65009 
 Station.6  Station.7  Station.8  Station.9 
  14.23170   13.63123   13.94914   14.18727 
```

We can use two grouping variables to return a matrix. However, when we do this, the second argument must be specified as a list.

```r
# What is the average temperature at each station and depth?
mean.temp <- tapply(ctd$temp, list(station = ctd$station, depth = ctd$depth), mean)
head(mean.temp[, 1:5])
```

```
            depth
station             1        2        3        4        5
  Station.1  17.22627 17.18102 17.07373 16.92864 16.75797
  Station.10 16.67695 16.55271 16.24712 15.95271 15.66983
  Station.11 16.39310 16.17458 15.89458 15.59407 15.34814
  Station.12 16.86448 16.74119 16.52642 16.35448 16.15030
  Station.13 17.05638 16.86203 16.68034 16.49610 16.24271
  Station.14 16.98061 16.81493 16.59866 16.41940 16.16791
```

aggregate

If we want to apply the same grouped summary to every column in a data frame, we can use `aggregate`:

```r
# what is the median of each measurement at each station?
st.medians <- aggregate(ctd[, 3:8], list(station = ctd$station), median, na.rm = TRUE)
head(st.medians)
```

```
     station   temp salinity  dox   ph pct_light density
1  Station.1 13.070  33.4570 7.05 8.05    88.330 25.1380
2 Station.10 14.445  33.4695 7.90 8.15    81.530 24.8515
3 Station.11 14.940  33.4625 7.88 8.15    76.270 24.7710
4 Station.12 14.095  33.4530 7.66 8.12    85.035 24.8905
5 Station.13 13.500  33.4635 7.42 8.10    86.640 25.0325
6 Station.14 14.170  33.4640 7.67 8.13    84.700 24.8860
```

Be careful if the function returns more than one thing though.

```r
st.range <- aggregate(ctd[, 3:8], list(station = ctd$station), range, na.rm = TRUE)
head(st.range)
```

```
     station temp.1 temp.2 salinity.1 salinity.2 dox.1 dox.2 ph.1 ph.2
1  Station.1   9.92  22.74     33.130     34.033  2.06 10.61 7.66 8.62
2 Station.10  10.36  22.65     33.162     33.864  2.14 13.03 7.66 8.55
3 Station.11  10.58  23.06     33.209     33.817  2.52 11.77 7.69 8.50
4 Station.12  10.24  23.00     32.561     34.311  2.28 11.38 7.67 8.63
5 Station.13  10.00  22.99     33.090     33.879  2.51 10.88 7.69 8.59
6 Station.14  10.20  22.74     33.069     33.891  2.25 11.35 7.65 8.61
  pct_light.1 pct_light.2 density.1 density.2
1       69.45       92.25    22.923    26.196
2       30.53       89.64    22.945    25.995
3        5.34       89.20    22.822    25.895
4       47.59       90.87    22.841    26.041
5       55.29       91.79    22.841    26.076
6       41.77       90.71    22.909    26.052
```

```r
str(st.range)
```

```
'data.frame':   40 obs. of  7 variables:
 $ station  : Factor w/ 40 levels "Station.1","Station.10",..: 1 2 3 4 5 6 7 8 9 10 ...
 $ temp     : num [1:40, 1:2] 9.92 10.36 10.58 10.24 10 ...
 $ salinity : num [1:40, 1:2] 33.1 33.2 33.2 32.6 33.1 ...
 $ dox      : num [1:40, 1:2] 2.06 2.14 2.52 2.28 2.51 2.25 2.26 2.24 2.33 2.33 ...
 $ ph       : num [1:40, 1:2] 7.66 7.66 7.69 7.67 7.69 7.65 7.67 7.66 7.65 7.67 ...
 $ pct_light: num [1:40, 1:2] 69.45 30.53 5.34 47.59 55.29 ...
 $ density  : num [1:40, 1:2] 22.9 22.9 22.8 22.8 22.8 ...
```

Note that the column names seem to have `.1` and `.2` when you print the object, but they aren't in the structure. In this case, every measurement column is itself a two column matrix:

```r
dim(st.range$temp)
```

```
[1] 40  2
```

```r
head(st.range$temp)
```

```
      [,1]  [,2]
[1,]  9.92 22.74
[2,] 10.36 22.65
[3,] 10.58 23.06
[4,] 10.24 23.00
[5,] 10.00 22.99
[6,] 10.20 22.74
```

by

To apply a function to an entire data frame, use `by()`, which is works much like `tapply`:

```
# How many records per station?
st.rows <- by(ctd, ctd$station, nrow)
head(st.rows)
```

```
ctd$station
 Station.1 Station.10 Station.11 Station.12 Station.13 Station.14
      3535       1120        762       1876       2229       1865
```

```
str(st.rows)
```

```
 'by' int [1:40(1d)] 3535 1120 762 1876 2229 1865 1826 1865 1474 1120 ...
 - attr(*, "dimnames")=List of 1
  ..$ ctd$station: chr [1:40] "Station.1" "Station.10" "Station.11" "Station.12" ...
 - attr(*, "call")= language by.data.frame(data = ctd, INDICES = ctd$station, FUN = nrow)
```

```
st.rows["Station.5"]
```

```
Station.5
      809
```

You can also summarize with multiple groups, which have to be included as a list:

```
# How many records per station?
st.depth.rows <- by(ctd, list(station = ctd$station, depth = ctd$depth), nrow)
str(st.depth.rows)
```

```
 'by' int [1:40, 1:60] 59 59 58 67 58 66 59 67 59 59 ...
 - attr(*, "dimnames")=List of 2
  ..$ station: chr [1:40] "Station.1" "Station.10" "Station.11" "Station.12" ...
  ..$ depth  : chr [1:60] "1" "2" "3" "4" ...
 - attr(*, "call")= language by.data.frame(data = ctd, INDICES = list(station = ctd$station, depth = ctd
```

```
# The object can be indexed like a matrix
st.depth.rows["Station.1", "12"]
```

```
[1] 59
```

mapply

To apply a function to sequential elements of multiple vectors, use `mapply`. The first argument is a function, and every argument afterwards is an argument to that function composed of vectors being iterated over. For example, the following creates a list of random numbers of alternating length with increasing range:

```
mapply(sample, x = 5:10, size = c(20, 4), replace = TRUE)
```

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

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

[[3]]
 [1] 3 7 3 6 5 3 2 2 1 7 2 1 5 6 1 3 6 5 2 6

[[4]]
[1] 8 4 6 4

[[5]]
 [1] 9 6 3 2 1 1 7 3 5 4 9 4 8 7 8 7 9 6 8 8

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

split

A handy function for creating lists based on a grouping variable is `split`. It will split a vector, matrix, or data frame. For instance, here is a list where every element is a data frame containing only one station's data:

```r
st.list <- split(ctd, ctd$station)
head(st.list[[1]])
```

```
    station sample_date  temp salinity  dox   ph pct_light density depth
1 Station.1  2012-11-08 16.81   33.420 8.07 8.20     90.32  24.346    16
2 Station.1  2012-04-19 10.52   33.805 3.16 7.73     88.14  25.930    18
3 Station.1  2010-01-06 15.11   33.415 7.22 8.13     88.97  24.725    32
4 Station.1  2014-02-06 14.00   33.430 7.31   NA     88.01  24.974    41
5 Station.1  2011-01-05 14.20   33.286 7.91 8.16     86.17  24.822     3
6 Station.1  2015-02-03 13.92   33.382 6.45 8.05     87.68  24.953    51
```

```r
head(st.list[[2]])
```

```
        station sample_date  temp salinity  dox   ph pct_light density
3536 Station.10  2010-05-10 14.99   33.479 9.62 8.35     70.32  24.799
3537 Station.10  2011-02-02 13.10   33.337 7.24 8.06     65.39  25.085
3538 Station.10  2010-03-17 13.45   33.406 8.62 8.17     73.64  25.069
3539 Station.10  2016-08-02 19.91   33.465 8.98 8.28     82.14  23.616
3540 Station.10  2016-11-02 14.00   33.279 6.68 8.02     79.46  24.858
3541 Station.10  2010-03-17 13.53   33.404 8.62 8.20     72.08  25.050
     depth
3536     4
3537     6
3538     6
3539    12
3540    19
3541     4
```

Here's the same creating an elment for each cast (station x date):

```r
st.dt.list <- split(ctd, list(station = ctd$station, date = ctd$sample_date))
st.dt.list[[1]]
```

```
[1] station     sample_date temp        salinity    dox         ph
[7] pct_light   density     depth
<0 rows> (or 0-length row.names)
```

Because it does all combinations of the grouping factors, a lot will be empty. Let's find them:

```
num.rows <- sapply(st.dt.list, nrow)
zero.rows <- which(num.rows == 0)
st.dt.list <- st.dt.list[-zero.rows]
st.dt.list[[1]]
```

```
        station sample_date  temp salinity  dox   ph pct_light density
5929 Station.12  2010-01-05 14.72   33.374 7.61 8.18     78.87  24.779
6294 Station.12  2010-01-05 14.72   33.374 7.61 8.18     79.14  24.778
6295 Station.12  2010-01-05 14.72   33.373 7.59 8.18     79.32  24.778
6750 Station.12  2010-01-05 14.72   33.373 7.59 8.18     79.11  24.778
6775 Station.12  2010-01-05 14.72   33.374 7.60 8.18     78.97  24.779
6778 Station.12  2010-01-05 14.64   33.375 7.50 8.18     77.96  24.796
6794 Station.12  2010-01-05 14.72   33.373 7.59 8.18     79.13  24.778
6856 Station.12  2010-01-05 14.71   33.374 7.56 8.18     79.24  24.781
6957 Station.12  2010-01-05 14.59   33.373 7.43 8.16     68.05  24.807
6973 Station.12  2010-01-05 14.59   33.373 7.44 8.16     69.51  24.806
6992 Station.12  2010-01-05 14.72   33.365 7.59 8.18     78.82  24.773
7061 Station.12  2010-01-05 14.71   33.363 7.62 8.19     78.36  24.772
7067 Station.12  2010-01-05 14.75   33.245 7.52 8.17     74.31  24.674
7087 Station.12  2010-01-05 14.72   33.368 7.58 8.18     79.09  24.775
7094 Station.12  2010-01-05 14.59   33.373 7.45 8.17     70.50  24.805
7103 Station.12  2010-01-05 14.60   33.374 7.45 8.17     72.98  24.804
7108 Station.12  2010-01-05 14.75   33.243 7.52 8.17     74.02  24.671
7131 Station.12  2010-01-05 14.73   33.328 7.61 8.18     76.89  24.742
7161 Station.12  2010-01-05 14.71   33.364 7.61 8.18     79.00  24.773
7172 Station.12  2010-01-05 14.71   33.360 7.64 8.18     78.11  24.770
7199 Station.12  2010-01-05 14.72   33.366 7.58 8.18     78.80  24.773
7218 Station.12  2010-01-05 14.72   33.347 7.63 8.18     77.19  24.759
7260 Station.12  2010-01-05 14.73   33.310 7.58 8.18     74.98  24.727
7266 Station.12  2010-01-05 14.72   33.372 7.59 8.18     78.90  24.777
7271 Station.12  2010-01-05 14.72   33.368 7.60 8.18     78.87  24.775
7275 Station.12  2010-01-05 14.74   33.268 7.55 8.17     73.94  24.693
7284 Station.12  2010-01-05 14.73   33.324 7.60 8.18     76.35  24.739
7285 Station.12  2010-01-05 14.72   33.371 7.58 8.18     79.13  24.777
     depth
5929    21
6294    20
6295    19
6750    18
6775    22
6778    24
6794    17
6856    23
6957    28
6973    27
6992    11
7061     9
7067     2
7087    14
```

```
7094    26
7103    25
7108     1
7131     6
7161    10
7172     8
7199    12
7218     7
7260     4
7266    16
7271    13
7275     3
7284     5
7285    15
```