new R markdown – a tutorial and a quick look behind the scenes

Other sites

- Statistics of Israel
- SAS blogs
- Jobs for R-users

dplyr Example #1

April 17, 2014
By dogle

(This article was first published on **fishR » R**, and kindly contributed to R-bloggers)

Hadley Wickam released the **dplyr** package in January 2014. Since then I have been itching to give it a try as it has been suggested to speed up some data manipulations and, more important to me, provide a singular framework for a variety of common data manipulations. Recently, I gave it a try with some simple common manipulations for fisheries work. Below are my examples.

First, the **dplyr** package, **plotrix** (for the plot at the end), and the **FSAdata** package (for the data file) must be loaded.

```
library(dplyr)
library(FSAdata)
library(plotrix)
```

The `RuffeSLRH92` data frame is then loaded.

```
data(RuffeSLRH92)
str(RuffeSLRH92)
```

```
## 'data.frame':    738 obs. of  11 variables:
##  $ fish.id : num  1992 1992 1992 1992 1992 ...
##  $ month   : int  4 4 4 4 4 4 4 4 4 4 ...
##  $ day     : int  23 23 23 23 23 23 23 23 23 23 ...
##  $ year    : int  1992 1992 1992 1992 1992 1992 1992 1992 1992 1992 ...
##  $ species : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ location: int  160170 160170 160170 160170 160170 160170 160170 160170 160170 160170 ...
##  $ length  : int  90 128 112 68 56 58 111 111 115 65 ...
##  $ weight  : num  9.3 32.5 19 4.4 2.1 2.8 16.1 17.9 22.7 3.4 ...
##  $ sex     : Factor w/ 3 levels "female","male",..: 2 1 2 2 3 2 2 2 2 1 ...
##  $ maturity: Factor w/ 11 levels "developing","immature",..: 7 7 7 7 10 7 7 7 7 2 ...
##  $ age     : int  NA NA NA NA NA NA NA NA NA 1 ...
```

Select (columns) Example

Columns can be selected from a data.frame with `select()`, given the original data.frame as the first argument and the variables to select, or include, as further arguments. The following creates a data.frame without the `fish.id`, `species`, `day` and `year` variables (they are not very useful in this context and will make the output further below easier to read).

```
RuffeSLRH92 <- select(RuffeSLRH92,-fish.id,-s
head(RuffeSLRH92)
```

```
##   month location length weight     sex maturity age
## 1     4   160170     90    9.3    male     ripe  NA
## 2     4   160170    128   32.5  female     ripe  NA
## 3     4   160170    112   19.0    male     ripe  NA
## 4     4   160170     68    4.4    male     ripe  NA
## 5     4   160170     56    2.1 unknown  unknown  NA
## 6     4   160170     58    2.8    male     ripe  NA
```

The following creates a data.frame of just the length and weight variables.

```
ruffeLW <- select(RuffeSLRH92,length,weight)
head(ruffeLW)
```

```
##   length weight
## 1     90    9.3
## 2    128   32.5
## 3    112   19.0
## 4     68    4.4
## 5     56    2.1
## 6     58    2.8
```

The **dplyr** package contains a variety of helpers for selecting. As one example, the following will select all variables that contains the letter "l".

```
ruffeL <- select(RuffeSLRH92,contains("l"))
str(ruffeL)
```

```
## 'data.frame':    738 obs. of  2 variables:
##  $ location: int  160170 160170 160170 160170 160170 160170 160170 160170 160170 160170 ...
##  $ length  : int  90 128 112 68 56 58 111 111 115 65 ...
```

Filtering Example

The `filter()` function can be used similarly to `subset()` to select a set of rows from an original data.frame according to some conditioning statement. As with `subset()`, `filter()` returns an object that maintains a list of the original levels whether those levels exist in the new data.frame or not. Use `droplevels()` to restrict the levels to only those that exist in the data.frame. The example below finds just the males from the original data.frame.

```
male <- filter(RuffeSLRH92,sex=="male")
xtabs(~sex,data=male)
```

```
## sex
##  female    male unknown
##       0     201       0
```

```
male <- droplevels(male)
xtabs(~sex,data=male)
```

```
## sex
## male
##  201
```

Multiple conditioning statements can be strung together as additional arguments to `filter()`. The example below finds males that are also ripe.

```
maleripe <- filter(RuffeSLRH92,sex=="male",ma
xtabs(~sex+maturity,data=maleripe)
```

```
##          maturity
## sex       developing immature mature nearly.ripe nearly.spent recovering
##   female           0        0      0           0            0          0
##   male             0        0      0           0            0          0
##   unknown          0        0      0           0            0          0
##          maturity
## sex       ripe running spent unknown yoy
##   female     0       0     0       0   0
##   male      63       0     0       0   0
##   unknown    0       0     0       0   0
```

So far, it seems that an "or" needs to be completed as with `subset()`. For example, the following selects those fish that are male or are ripe.

```
maleripe2 <- filter(RuffeSLRH92,sex=="male" |
xtabs(~sex+maturity,data=maleripe2)
```

```
##          maturity
## sex       developing immature mature nearly.ripe nearly.spent recovering
##   female           0        0      0           0            0          0
##   male            29        5      5          21           14         10
##   unknown          0        0      0           0            0          0
##          maturity
## sex       ripe running spent unknown yoy
##   female   107       0     0       0   0
```

```
##   male      63      17    34       3   0
##   unknown    0       0     0       0   0
```

Arrange Example

The `arrange()` function can be used to order individuals. The first argument is the data.frame and the following arguments are the variables to sort by. The following sorts, in ascending order, the male data.frame created above by length.

```
malea <- arrange(male,length)
head(malea)
```

```
##   month location length weight  sex   maturity age
## 1     5   330040     45    1.4 male    unknown  NA
## 2    10   190170     47    1.3 male   immature  NA
## 3     9   120070     56    2.3 male developing  NA
## 4     4   370010     57    2.5 male       ripe  NA
## 5     4   160170     58    2.8 male       ripe  NA
## 6     5   340050     58    2.9 male     mature  NA
```

```
tail(malea)
```

```
##     month location length weight  sex     maturity age
## 196     6    18007    153   43.4 male nearly.spent  NA
## 197     4    15011    154   48.7 male         ripe  NA
## 198     9    10060    155   41.0 male  nearly.ripe  NA
## 199     4    90060    161   54.0 male       mature  NA
## 200    10    60010    163   51.0 male       mature  NA
## 201     4    15011    183   82.8 male         ripe  NA
```

The following does the same but in descending order.

```
maled <- arrange(male,desc(length))
head(maled)
```

```
##   month location length weight  sex     maturity age
## 1     4    15011    183   82.8 male         ripe  NA
## 2    10    60010    163   51.0 male       mature  NA
## 3     4    90060    161   54.0 male       mature  NA
## 4     9    10060    155   41.0 male  nearly.ripe  NA
## 5     4    15011    154   48.7 male         ripe  NA
## 6     6    18007    153   43.4 male nearly.spent  NA
```

```
tail(maled)
```

```
##     month location length weight  sex   maturity age
## 196     4   160170     58    2.8 male       ripe  NA
## 197     5   340050     58    2.9 male     mature  NA
## 198     4   370010     57    2.5 male       ripe  NA
## 199     9   120070     56    2.3 male developing  NA
## 200    10   190170     47    1.3 male   immature  NA
## 201     5   330040     45    1.4 male    unknown  NA
```

Multiple levels of ordering can be completed by including multiple variables as arguments to `arrange()`. The following sorts the data by ascending length and then ascending weight.

```
ruffe2 <- arrange(RuffeSLRH92,length,weight)
head(ruffe2)
```

```
##   month location length weight     sex maturity age
## 1     7   360020     13    0.1 unknown      yoy  NA
## 2     8   170160     14    0.1 unknown      yoy  NA
## 3     7   160190     15    0.1 unknown      yoy  NA
## 4     7   320060     16    0.1 unknown      yoy  NA
## 5     7   170160     16    0.1 unknown      yoy  NA
## 6     7   300070     17    0.1 unknown      yoy  NA
```

```
tail(ruffe2)
```

```
##     month location length weight    sex     maturity age
## 733     6    50090    178   57.4 female         ripe  NA
## 734     6    60070    180   55.1 female      running  NA
## 735     7    80090    180   63.3 female nearly.spent  NA
## 736     7    50030    180   72.6 female        spent  NA
## 737     4    15011    183   82.8   male         ripe  NA
## 738     7   320060    192   93.1 female nearly.spent  NA
```

Add new variables (i.e., columns) Example

The mutate() function can be used to add new variables to a data.frame. It requires the original data.frame as the first argument and then arguments to create new variables as the remaining arguments. The example below adds the natural log of length and weight to the data.frame created above that contains just the length and weight variables.

```
ruffeLW <- mutate(ruffeLW,logL=log(length),lo
head(ruffeLW)
```

```
##   length weight  logL   logW
## 1     90    9.3 4.500 2.2300
## 2    128   32.5 4.852 3.4812
## 3    112   19.0 4.718 2.9444
## 4     68    4.4 4.220 1.4816
## 5     56    2.1 4.025 0.7419
## 6     58    2.8 4.060 1.0296
```

Aggregation and Summarization Example

The **dplyr** package also provides functions that allow for simple aggregation of results. The `group_by()` function first sets up how you want to group your data. In the code below, the `byMon` data.frame is going to create groups by the `month` variable. The `summarize()` function will then summarize a data.frame by the functions after the first argument. The package also provides `n()` to count the number of individuals. Thus, the example below will count the number of ruffe in the original data.frame by each month.

```
byMon <- group_by(RuffeSLRH92,month)
( sumMon <- summarize(byMon,count=n()) )
```

```
## Source: local data frame [7 x 2]
##
##   month count
## 1     4    62
## 2     5    66
## 3     6   154
## 4     7   182
## 5     8    79
## 6     9   126
## 7    10    69
```

The following counts the number of ruffe by each month and sex.

```
byMonSex <- group_by(RuffeSLRH92,month,sex)
( sumMonSex <- summarize(byMonSex,count=n()))
```

```
## Source: local data frame [20 x 3]
## Groups: month
##
##    month     sex count
## 1      4  female    30
## 2      4    male    30
## 3      4 unknown     2
## 4      5  female    35
## 5      5    male    31
## 6      6  female   114
## 7      6    male    38
## 8      6 unknown     2
## 9      7  female    96
## 10     7    male    20
## 11     7 unknown    66
## 12     8  female    28
## 13     8    male    18
## 14     8 unknown    33
## 15     9  female    68
## 16     9    male    39
## 17     9 unknown    19
## 18    10  female    36
## 19    10    male    25
## 20    10 unknown     8
```

Multiple functions can be used to create multiple summaries at once. The following summarizes the number of fish and the mean and standard

deviation of length by month.

```
( LenSumMon <- summarize(byMon,n=n(),mn=mean(
```

```
## Source: local data frame [7 x 4]
##
##   month   n     mn    sd
## 1     4  62 111.37 35.42
## 2     5  66 112.41 35.60
## 3     6 154 113.62 31.61
## 4     7 182  93.55 50.55
## 5     8  79  82.22 40.80
## 6     9 126  98.76 39.15
## 7    10  69  98.72 39.16
```

Putting It All Together

Finally, all of the functions described above can be strung together with the `%.%` operator. This is best shown with an example. The following set of code will compute the proportion of all captured fish captured in each month by (1) grouping the data by month, (2) summarizing the number per month, (3) adding a new variable that is the proportion of the total catch in each month, and then (4) sorting the results such that the month with the largest catch is listed first.

```
fnl1 <- RuffeSLRH92 %.%
  group_by(month) %.%
  summarize(n=n()) %.%
  mutate(prop.catch=n/sum(n)) %.%
  arrange(desc(prop.catch))
fnl1
```

```
## Source: local data frame [7 x 3]
##
##   month   n prop.catch
## 1     7 182    0.24661
## 2     6 154    0.20867
## 3     9 126    0.17073
## 4     8  79    0.10705
## 5    10  69    0.09350
## 6     5  66    0.08943
## 7     4  62    0.08401
```

The following example constructs a data.frame (that is then used to construct a plot) that contains the mean, sd, se, and approximate 95% confidence interval for length by month.

```
fnl2 <- RuffeSLRH92 %.%
  group_by(month) %.%
  summarize(n=n(),mn=mean(length),sd=sd(lengt
  mutate(se=sd/sqrt(n),LCI=mn+qnorm(0.025)*se
fnl2
```

```
## Source: local data frame [7 x 7]
##
##   month   n     mn    sd    se    LCI    UCI
## 1     4  62 111.37 35.42 4.498 102.55 120.19
## 2     5  66 112.41 35.60 4.382 103.82 121.00
## 3     6 154 113.62 31.61 2.547 108.63 118.62
## 4     7 182  93.55 50.55 3.747  86.21 100.90
## 5     8  79  82.22 40.80 4.590  73.22  91.21
## 6     9 126  98.76 39.15 3.488  91.93 105.60
## 7    10  69  98.72 39.16 4.714  89.48 107.96
```

```
with(fnl2,plotCI(month,mn,ui=UCI,li=LCI,pch=1
```

Final Thoughts

The **dplyr** package appears to have many more useful functions. This first foray suggests to me that it is going to be a useful "grammar" for simplifying data manipulations. I will post more as I explore it more. In the meantime, let me know what your experience with **dplyr** is.

Filed under: Fisheries Science, R Tagged: Data, Manipulation, R

Comments: 7

Tweet

To **leave a comment** for the author, please follow the link and comment on his blog: **fishR » R**.

R-bloggers.com offers **daily e-mail updates** about R news and tutorials on topics such as: visualization (ggplot2, Boxplots, maps, animation), programming (RStudio, Sweave, LaTeX, SQL, Eclipse, git, hadoop, Web Scraping) statistics (regression, PCA, time series, trading) and more...

If you got this far, why not **subscribe for updates** from the site? Choose your flavor: e-mail, twitter, RSS, or facebook...

Tweet

Comments are closed.

Top 3 Posts from the past 2 days

- Variable Selection in Market Segmentation: Clustering or Biclustering?
- Building Interactive Graphs with ggplot2 and Shiny
- Using apply, sapply, lapply in R

Search & Hit Enter

Top 9 articles of the week

1. Using apply, sapply, lapply in R
2. Installing R packages
3. library() vs require() in R
4. Things to try after useR! - Part 1: Deep Learning with H2O
5. Basics of Histograms
6. Box-plot with R – Tutorial
7. Fast-track publishing using the new R markdown – a tutorial and a quick look behind the scenes
8. Interactive visualization of non-linear logistic regression decision boundaries with Shiny
9. Analyzing package dependencies and download logs from Rstudio, and a start towards building an R recommendation engine

Sponsors