dplyr

Brian A. Fannin

August 22, 2017

Overview

Another element of the "tidyverse" developed by Hadley Wickham and many others.

Let's look at some data

```
library(raw)
data("MultiTri")

View(MultiTri)
```

Real quick:

- What's the average IBNR for lag 2 by accident year?

dplyr to the rescue!

Basic verbs

Verb
select/rename
mutate
group
arrange
filter
summarise

select/rename

Comparable to the "select" verb in SQL. This will return only those columns you've requested. A "negative" column name will omit that column.

```
financials <- select(MultiTri, CumulativePaid,
    CumulativeIncurred, IBNR)
no_lag <- select(MultiTri, -Lag)
```

Identifying which columns to select

Loads of helper functions to find columns

function
contains
starts_with
ends_with
matches
num_range

```
years <- select(MultiTri, contains("year"))
```

rename

Closely related to `select`. New name on the left, old name on the right. (Think the way that variables are usually assigned.)

```
new_tri <- rename(MultiTri, DevelopmentLag = Lag)
```

mutate

mutate

Create a new column or alter an existing one.

```
new_tri <- mutate(new_tri, PaidToIncurred = CumulativePaid/CumulativeIncurred,
    Upper = DevelopmentYear <= 1997)
```

Combining operations

```
new_tri <- mutate(MultiTri, Upper = DevelopmentYear <=
    1997)
new_tri <- select(new_tri, -DevelopmentYear)
```

This syntax will get tedious real fast. We need some way to make this more efficient. Fortunately, we have one . . .

The pipe operator

%>%

Key to getting the most out of dplyr is using the `%>%` or "pipe" operator. It takes whatever is to its left and inserts it as the first (unnamed) argument in the function to its left.

Comes from its own package, "magrittr" and may be used without dplyr.

In RStudio, may be inserted with CTRL-SHIFT-M

Pipe example

```
1 %>% exp()
## [1] 2.718282
```

Chain

May be chained as often as you like. Operations happen left to right. The output keeps getting passed as the input of the next function:

```
1 %>% exp() %>% log()
## [1] 1
```

arrange

arrange

Use the desc function to arrange in descending order.

```
MultiTri %>% arrange(AccidentYear)

MultiTri %>% arrange(desc(IBNR))
```

filter/slice

Straightforward

```
upper_tri <- MultiTri %>% filter(DevelopmentYear <=
    1997)
```

Multiple conditions are OK

```
upper_tri <- MultiTri %>% filter(DevelopmentYear <=
    1997, IBNR > 500)
```

slice

`slice` will take specific rows of data.

```
every_fifth <- MultiTri %>% slice(seq(from = 5,
    by = 5, to = nrow(MultiTri)))
```

group_by

group_by

This will group the data. The effect isn't material until another operation is applied.

```
df_grouped <- MultiTri %>% group_by(Company, AccidentYear)
```

summarise

Apply a function across each group

```
dfBigYear <- MultiTri %>% group_by(AccidentYear) %>%
    summarise(BiggestIBNR = max(IBNR))
```

Gotcha

Arrange does not respect grouping! Didn't used to be this way. A rare misstep (IMHO) for Mr. Wickham.

All together now!

Answer a complex question quickly

- For each company which has had paid to incurred ratio less than 40%, which accident year had the highest P2I?

```
dfBigCase <- MultiTri %>% mutate(PaidToIncurred = CumulativePaid/CumulativeIncurred) %>%
    filter(PaidToIncurred < 0.4) %>% group_by(Company) %>%
    arrange(desc(PaidToIncurred)) %>% slice(1) %>%
    select(Company, AccidentYear)
```

A couple more things

What about joining?

```
dfCo <- data.frame(Company = unique(MultiTri$Company),
    stringsAsFactors = FALSE)
dfCo$PolicyHolderSurplus <- rnorm(nrow(dfCo),
    1e+08, 0.3 * 1e+08)
dfCo
##                             Company
## 1            Farm Bureau Of MI Grp
```

```
## 2             West Bend Mut Ins Grp
## 3                Island Ins Cos Grp
## 4  Kentucky Farm Bureau Mut Ins Grp
## 5            Farmers Automobile Grp
## 6                State Farm Mut Grp
## 7            NC Farm Bureau Ins Grp
## 8                  Grinnell Mut Grp
## 9      New Jersey Manufacturers Grp
## 10                  Dorinco Rein Co
##    PolicyHolderSurplus
## 1             97946716
## 2            148279884
## 3             73884341
## 4             73820701
## 5            117292342
## 6             91332377
## 7             91583402
## 8            105599692
## 9            117667883
## 10            45739949
```

Joining

```
dfJoined <- dplyr::inner_join(MultiTri, dfCo)
dfJoined %>% select(Company, PolicyHolderSurplus,
    DevelopmentYear) %>% head(3)
## # A tibble: 3 x 3
##                  Company PolicyHolderSurplus
##                    <chr>               <dbl>
## 1 Farm Bureau Of MI Grp            97946716
## 2 Farm Bureau Of MI Grp            97946716
## 3 Farm Bureau Of MI Grp            97946716
## # ... with 1 more variables:
## #   DevelopmentYear <int>
```

What's tidyr?

Does a number of things, but I'm only going to talk about 2: spreading and gathering

```
one_co <- new_tri %>% filter(Company == unique(MultiTri$Company)[1],
    Line == "Workers Comp")
library(tidyr)
```

tidyr::spread

```
wide_tri <- one_co %>% select(AccidentYear, Lag,
    NetEP, CumulativePaid) %>% spread(Lag, CumulativePaid)
wide_tri
## # A tibble: 10 x 12
##    AccidentYear NetEP   `1`   `2`   `3`
##  *        <int> <dbl> <dbl> <dbl> <dbl>
##  1         1988  7122  1346  3389  4666
##  2         1989  7588  1411  3641  4729
##  3         1990 10232  1424  4460  5791
##  4         1991 12731  2355  6208  8191
##  5         1992 16847  2544  7554  8008
##  6         1993 21327  3512  6745  9173
##  7         1994 21686  2708  7360 10084
##  8         1995 24955  2609  6240  8280
##  9         1996 22316  2652  5332  6822
## 10         1997 20975  2192  5320  7109
## # ... with 7 more variables: `4` <dbl>,
## #   `5` <dbl>, `6` <dbl>, `7` <dbl>,
## #   `8` <dbl>, `9` <dbl>, `10` <dbl>
```

What about missing values?

```
wide_tri <- one_co %>% filter(Upper) %>% select(AccidentYear,
    Lag, NetEP, CumulativePaid) %>% spread(Lag,
    CumulativePaid)
wide_tri
## # A tibble: 10 x 12
##    AccidentYear NetEP   `1`   `2`   `3`
##  *        <int> <dbl> <dbl> <dbl> <dbl>
##  1         1988  7122  1346  3389  4666
##  2         1989  7588  1411  3641  4729
##  3         1990 10232  1424  4460  5791
##  4         1991 12731  2355  6208  8191
##  5         1992 16847  2544  7554  8008
##  6         1993 21327  3512  6745  9173
##  7         1994 21686  2708  7360 10084
##  8         1995 24955  2609  6240  8280
##  9         1996 22316  2652  5332    NA
## 10         1997 20975  2192    NA    NA
## # ... with 7 more variables: `4` <dbl>,
## #   `5` <dbl>, `6` <dbl>, `7` <dbl>,
## #   `8` <dbl>, `9` <dbl>, `10` <dbl>
```

tidyr::gather

Note that we're *excluding* AccidentYear and Net EP from the gathering.

```
long_tri <- wide_tri %>% gather(Lag, CumulativePaid,
    -AccidentYear, -NetEP)
long_tri
## # A tibble: 100 x 4
##    AccidentYear NetEP   Lag CumulativePaid
##           <int> <dbl> <chr>          <dbl>
##  1         1988  7122     1           1346
##  2         1989  7588     1           1411
##  3         1990 10232     1           1424
##  4         1991 12731     1           2355
##  5         1992 16847     1           2544
##  6         1993 21327     1           3512
##  7         1994 21686     1           2708
##  8         1995 24955     1           2609
##  9         1996 22316     1           2652
## 10         1997 20975     1           2192
## # ... with 90 more rows
```

What's a tibble?

More or less a data frame.

- The print command won't try to print every row in the tibble.
- They don't automatically convert strings to factors
- add_row and add_column methods are kinda cool, especially add_row
- Some other stuff (no partial matching on column names)

References

- `http://www.tidyverse.org/`
- `http://dplyr.tidyverse.org/`
-