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)
```

Summary

References