WK5 925

Zichun Liu

9/25/2017

0

```
knitr::opts_chunk$set(echo = TRUE)
library(nycflights13)
library(microbenchmark)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

1 Why dplyr

1. Easier to read and write
2. More efficient

```
microbenchmark(
  filter(flights, month == 1, day == 1),
  subset(flights, subset = month == 1 & day ==1)
)
```

```
## Unit: milliseconds
##                                           expr       min        lq
##            filter(flights, month == 1, day == 1)  5.524863  7.047614
##  subset(flights, subset = month == 1 & day == 1) 11.365593 16.387837
##      mean    median        uq       max neval cld
##  14.42808  8.322494  9.439189 545.10412   100   a
##  19.30123 19.351335 21.443072  37.14389   100   a
```

2 %>% Operator

Although not required, the dplyr packages make use of the pipe operator `%>%` developed by Stefan Milton Bache in the R package `magrittr`. Although all the functions in dplyr can be used without the pipe operator, one of the great conveniences these packages provide is the ability to string multiple functions together by incorporating `%>%`.

This operator will forward a value, or the result of an expression, into the next function call/expression. For instance a function to filter data can be written as:

```
filter(data, variable == numeric_value)

data %>% filter(variable == numeric_value)
```

Both functions complete the same task and the benefit of using %>% is not evident; however, when you desire to perform multiple functions its advantage becomes obvious. For instance, if we want to filter some data, summarize it, and then order the summarized results we would write it out as:

```
#Nested Option:
arrange(summarize(filter(data, variable == numeric_value),Total = sum(variable)),desc(Total))

#or
#Multiple Object Option:
a = filter(data, variable == numeric_value)
b = summarise(a, Total = sum(variable))
c = arrange(b, desc(Total))

#or
#%>% Option:
data %>%
  filter(variable == "value") %>%
  summarise(Total = sum(variable)) %>%
  arrange(desc(Total))
```

As your function tasks get longer the %>% operator becomes more efficient and makes your code more legible. In addition, although not covered in this tutorial, the %>% operator allows you to flow from data manipulation tasks straight into vizualization functions (via ggplot and ggvis) and also into many analytic functions.

```
summarise(mtcars, mean(disp))
```

```
##   mean(disp)
## 1   230.7219
```

```
summarise(group_by(mtcars, cyl), mean(disp))
```

```
## # A tibble: 3 × 2
##     cyl `mean(disp)`
##   <dbl>        <dbl>
## 1     4     105.1364
## 2     6     183.3143
## 3     8     353.1000
```

```
summarise(group_by(mtcars, cyl), m = mean(disp), sd = sd(disp))
```

```
## # A tibble: 3 × 3
##     cyl        m       sd
##   <dbl>    <dbl>    <dbl>
## 1     4 105.1364 26.87159
## 2     6 183.3143 41.56246
## 3     8 353.1000 67.77132
```