Data Wrangling - dplyr and magrittr

IMC 490: Machine Learning for IMC

4/26/2017

New York Times - For big data scientists, hurdle to insights is janitor work (link)

> "far too much handcrafted work — what data scientists call "data wrangling," "data munging" and "data janitor work" — is still required."
>
> " 'Data wrangling is a huge — and surprisingly so — part of the job,' said Monica Rogati, vice president for data science at Jawbone."

Forbes - Data preparation most time consuming, least enjoyable data science task survey says (link)

> "…data scientist Mike Driscoll popularized the term 'data munging,' describing the 'painful process of cleaning, parsing, and proofing one's data' as one of the three sexy skills of data geeks."
>
> "In 2013, Josh Wills (then director of Data Science at Cloudera, now Director of Data Engineering at Slack) told Technology Review 'I'm a data janitor. That's the sexiest job of the 21st century. It's very flattering, but it's also a little baffling.' "
>
> "Big Data Borat tweeted that 'Data Science is 99% preparation, 1% misinterpretation.' "

Today, we'll be going over three libraries in R for slicing and dicing data:
- dplyr (slice, dice, and summarize)
- magrittr (pipes)

Data wrangling is an important task for every data analyst and data scientist. Much of your time spent analyzing real data will be spent getting the data into a form to be analyzed. `dplyr` is the industry standard R package for cutting up data. In addition, the syntax and skills you learn using dplyr can also be directly applied to big data processing using Apache Spark. Let's get set up.

```r
libs = c("dplyr", "magrittr")
for (lib in libs) {
  install.packages(lib)
  require(lib, character.only = TRUE)
}
```

Remember you'll have to load the packages with `require()` or `library()` the next time you start up R.

```r
require(dplyr)
require(magrittr)
```

A hugely helpful cheat sheet for these libraries:
https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf

Exercises adapted from:
https://github.com/brandon-rhodes/pycon-pandas-tutorial

dplyr

To slice your data:
`select(data, col1, col2)` - select columns
`filter(data, col1 > 3)` - select rows using a logical statement

To modify your data:
`mutate(data, new_col = col1 + col2 / col3)` - create a new column
`arrange(data, col1)` - sort a dataframe by a column

You also have your standard SQL functions.

To group by and summarise:
`group_by(data, col1)`
`summarise(grouped_data, count = n())`

To join:
`left_join(data1, data2, by = "customer_id")`
`right_join()`
`inner_join()`
`outer_join()`

There are many more dplyr functions! These are just my most frequently used ones. Let's practice using them.

```r
# run this code to grab our demo data
data("mtcars")
mtcars$model = row.names(mtcars)
row.names(mtcars) = NULL
mtcars = select(mtcars, model, mpg:gear)
head(mtcars)
```

```
##               model  mpg cyl disp  hp drat    wt  qsec vs am gear
## 1         Mazda RX4 21.0   6  160 110 3.90 2.620 16.46  0  1    4
## 2     Mazda RX4 Wag 21.0   6  160 110 3.90 2.875 17.02  0  1    4
## 3        Datsun 710 22.8   4  108  93 3.85 2.320 18.61  1  1    4
## 4    Hornet 4 Drive 21.4   6  258 110 3.08 3.215 19.44  1  0    3
## 5 Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3
## 6           Valiant 18.1   6  225 105 2.76 3.460 20.22  1  0    3
```

select()

```
d = select(mtcars, model, hp, wt)
head(d, 4)
```

```
##             model  hp    wt
## 1       Mazda RX4 110 2.620
## 2   Mazda RX4 Wag 110 2.875
## 3      Datsun 710  93 2.320
## 4  Hornet 4 Drive 110 3.215
```

filter()

```
d = filter(mtcars, hp > 100 & wt < 3000)
head(d, 4)
```

```
##                model  mpg cyl disp  hp drat    wt  qsec vs am gear
## 1          Mazda RX4 21.0   6  160 110 3.90 2.620 16.46  0  1    4
## 2      Mazda RX4 Wag 21.0   6  160 110 3.90 2.875 17.02  0  1    4
## 3     Hornet 4 Drive 21.4   6  258 110 3.08 3.215 19.44  1  0    3
## 4 Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3
```

mutate()

```
d = mutate(mtcars, kpg = 1.6 * mpg)
d = select(d, model, kpg)
head(d, 4)
```

```
##             model   kpg
## 1       Mazda RX4 33.60
## 2   Mazda RX4 Wag 33.60
## 3      Datsun 710 36.48
## 4  Hornet 4 Drive 34.24
```

arrange()

```
d = arrange(mtcars, mpg)
head(d, 4)
```

```
##                  model  mpg cyl disp  hp drat    wt  qsec vs am gear
## 1  Cadillac Fleetwood 10.4   8  472 205 2.93 5.250 17.98  0  0    3
## 2 Lincoln Continental 10.4   8  460 215 3.00 5.424 17.82  0  0    3
## 3          Camaro Z28 13.3   8  350 245 3.73 3.840 15.41  0  0    3
## 4          Duster 360 14.3   8  360 245 3.21 3.570 15.84  0  0    3
```

```
d = arrange(mtcars, desc(wt))
head(d, 4)
```

```
##                  model  mpg cyl  disp  hp drat    wt  qsec vs am gear
## 1 Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3
## 2   Chrysler Imperial 14.7   8 440.0 230 3.23 5.345 17.42  0  0    3
## 3  Cadillac Fleetwood 10.4   8 472.0 205 2.93 5.250 17.98  0  0    3
## 4          Merc 450SE 16.4   8 275.8 180 3.07 4.070 17.40  0  0    3
```

group_by() and summarise()

```
d = group_by(mtcars, cyl)
d = summarise(d, avg_weight = mean(wt), number_cars = n())
d
```

```
## # A tibble: 3 × 3
##     cyl avg_weight number_cars
##   <dbl>      <dbl>       <int>
## 1     4   2.285727          11
## 2     6   3.117143           7
## 3     8   3.999214          14
```

magrittr %>%

dplyr by itself provides a very intuitive grammar of data manipulation. Combined with magrittr, you can write clean data transformations in R that read like english. magrittr brings pipes into R. The pipe symbol in R looks like:

%>%

A pipe is a special symbol that transfers, or "pipes", the output of one command into the next. Using pipes, you can chain commands to build complex queries like these.

What is the average mpg of with over 100 horsepower that weigh between 2500 and 3500 lbs, split by the number of cylinders they have?

```
mtcars %>%
  mutate(wt = wt*1000) %>%
  filter(hp > 100, wt < 3500, wt > 2500) %>%
  group_by(cyl) %>%
  summarise(avg_mpg = mean(mpg))
```

```
## # A tibble: 3 × 2
##     cyl  avg_mpg
##   <dbl>    <dbl>
## 1     4 21.40000
## 2     6 19.74286
## 3     8 16.56667
```

```
mtcars %>%
  mutate(wt = wt*1000) %>%   # convert weight from tons to pounds
  filter(hp > 100, wt < 3500, wt > 2500) %>% # filter by the horsepower and weight
  group_by(cyl) %>%  # group by cylinder
  summarise(avg_mpg = mean(mpg))  # create new field
```

Tip: the pipe shortcut in RStudio is COMMAND+SHIFT+M (Mac) and CTRL+SHIFT+M (PC).

Exercises

Download and read the `cast` and `titles` dataframe from Canvas. This is data scraped from IMDB (The Internet Movie Database) on the titles and casts of movies.

Please note that the cast dataframe is intentionally big (75MB, 1M rows) - this is to demonstrate how fast dplyr can process large dataframes. It may take a second to read into R. I would suggest adding the parameter `stringsAsFactors = FALSE` to `read.csv()` as it will speed things up, but you'll ned to convert some columns like `cast$type` to factors manually.

```r
titles = read.csv("titles.csv", stringsAsFactors = FALSE)
cast = read.csv("cast.csv", , stringsAsFactors = FALSE)
```

0. How many movies are listed in the titles dataframe?
1. What are the earliest two films listed in the titles dataframe?
2. How many movies have the title "Hamlet"?
3. When was the first movie titled "Hamlet" made?
4. List all of the "Treasure Island" movies from earliest to most recent.
5. In what years has a movie titled "Batman" been released?
6. How many roles were there in the movie "Inception"?
7. What were the 5 most important roles in "Inception"? (n = 1 is the most important)
8. How many people have played an "Ophelia"?
9. How many people have played a role called "The Dude"?
10. How many roles were available for actors in the 1950s?
11. How many roles were avilable for actresses in the 1950s?
12. What are the 11 most common character names in movie history?
13. Who are the 10 people most often credited as "Herself" in film history?
14. Using group_by(), find the number of films that have been released each year. Plot it.
15. Find the number of actor roles and actress roles available each year.