

00 - Course Intro

Data Mining

SYS 4582/6018 | Spring 2019

00-intro.pdf

Contents

1	Requirements	2
2	Syllabus	2
2.1	Course Webpage	2
2.2	Course Prereqs	2
3	Introduction to R	2
3.1	Getting Help	2
3.2	RStudio	3
3.3	Using R Packages	3
3.4	Graphics with the <code>ggplot2</code> package	3
3.5	Data Transformation with the <code>dplyr</code> package	4
3.6	Groupwise operations	5
3.7	Relational Data and Joins	6
3.8	Data Importing	7
3.9	Tidy Data with the <code>tidyr</code> package	7
4	RMarkdown	8

1 Requirements

- Working version of R and RStudio
- Install R package: `tidyverse` and `nycflights13`

2 Syllabus

2.1 Course Webpage

- We have a course webpage <https://mdporter.github.io/SYS6018/>
 - lectures
 - R scripts
 - data sets
- We will use the Collab site for homework submission, announcements, Piazza, etc.

2.2 Course Prereqs

- Linear Regression
 - Multiple Linear Regression
 - Logistic Regression
 - Categorical Predictors (dummy coding)
 - Implementation in R (`lm()`, `predict()`, `coef()`, etc.)
 - Estimation / Model Fitting
 - Cross-validation
- Probability and Statistics
 - Bayes Theorem
 - CDF/PDF/PMF
 - Maximum Likelihood Estimation
 - Distributions: normal, binomial, hypergeometric, etc.
 - Expected value, variance, median, quantiles
 - Mean Square Error
 - Confidence Intervals
 - Hypothesis Testing
- Math
 - Calculus
 - Matrix Calculations
 - PCA, SVD
- Computing
 - data types: vector, matrix, array, list, etc.
 - writing simple functions
 - generating random variables
 - RMarkdown

3 Introduction to R

3.1 Getting Help

- A good source of basic data analysis using R is found in the free book [R for Data Science](#).
- Web search, especially stackoverflow.com/ and stats.stackexchange.com
- Troubleshooting/Debugging.
 - Check one line of code at a time.
 - Use scripts

- Make sure it works in plain R before incorporating into Rmd

3.2 RStudio

- Install R and RStudio
- Make use of *Projects* in RStudio

3.3 Using R Packages

It takes two steps to use the functions and data in an R package

1. Install the package
 - i.e. download the package to your computer
 - this only needs to be done one time
 - `install.packages()`
2. Load the package
 - i.e. tell R to look for the package functions and/or data
 - this needs to be done every time R is started (and you want to use the package)
 - `library()`

3.3.1 Note on tidyverse package

- The tidyverse package <https://www.tidyverse.org/packages/> is really just a wrapper to load several related R packages
 - ggplot2 for graphics
 - dplyr for data manipulation
 - tidyr for getting data into tidy form
 - readr for loading in data
 - tibble for improved data frames
 - purrr for functional programming
 - stringr for string manipulation
 - forcats for categorical/factor data
- This provides a nice shortcut to load all of these packages with `library(tidyverse)` instead of each separately:

```
#- the hard way
library(ggplot2)
library(dplyr)
library(tidyr)
library(readr)
library(tibble)
library(purrr)
library(stringr)
library(forcats)
```

```
#- the easy way
library(tidyverse)
```

3.4 Graphics with the ggplot2 package

The [ggplot2](https://ggplot2.tidyverse.org/) package is an approach to creating graphics for data analysis.

- See <https://ggplot2.tidyverse.org/>
- Keep the [ggplot2 cheatsheet](#) handy

3.5 Data Transformation with the `dplyr` package

- See <https://dplyr.tidyverse.org/>
- Keep the [dplyr cheatsheet](#) handy

3.5.1 single table verbs

1. `filter()`: find/keep certain rows
 - alternative to `base::subset()`
 - `slice()` to keep by row number
 - `between()`: numeric values in a range
2. `arrange()`: reorder rows
 - alternative to `base::order()`
 - `desc()` to use descending order
3. `select()`: find/keep certain columns
 - helper functions: `starts_with()`, `ends_with()`, `matches()`, `contains()`, `?select`
4. `mutate()`: add/create new variables
 - alternative to `base::transform()`
 - `transmute()`: only return new variables
5. `summarize()`: produce summary statistics
 - don't confuse with `summary()`
 - most useful when data is *grouped*

3.5.2 Chaining/Pipes

- Multiple operations can be chained together with the *pipe* operator, `%>%`, (pronounced as *then*). Technically, it performs `x %>% f(y) -> f(x, y)`. This lets you focus on the verbs, or actions you are performing.

```
x = c(1:5, NA)
x %>% mean(na.rm=TRUE)
#> [1] 3
mean(x, na.rm=TRUE)
#> [1] 3
```

Your Turn #1

1. Load the `nycflights13` package, which contains airline on-time data for all flights departing NYC in 2013. Also includes useful 'metadata' on airlines, airports, weather, and planes.
2. Using the `flights` data,
 - find all flights that were less than 1000 miles (`distance`)
 - Keep only the columns: `dep_delay`, `arr_delay`, `origin`, `dest`, `air_time`, and `distance`
 - Add the Z-score for departure delays
 - Convert the departure and arrival delays into hours
 - Calculate the average flight speed (in mph)
 - order by average flight speed (fastest to slowest)
 - return the first 12 rows

3.5.3 Other useful `dplyr` functions

- `distinct()`: retain unique/distinct rows
- `sample_n()` and `sample_frac()`: randomly sample rows

- `top_n()`: selects and orders the top `n` rows according to `wt`
- `add_column()` add new column in particular position
- `add_row()` adds new row(s) to the table
- `na_if(x, y)` converts the `y` valued elements in `x` to NA

```
x = c(1, 2, -99, 5, 5, -99)
na_if(x, -99)      # replace -99 with NA
#> [1] 1 2 NA 5 5 NA
```

- `coalesce(x, y)` replaces the NA in `x` with `y`

```
x = c(1, 2, NA, 5, 5, NA)
coalesce(x, 0)     # replace NA with 0
#> [1] 1 2 0 5 5 0
```

3.6 Groupwise operations

3.6.1 Split - Apply - Combine

The dplyr operations are more powerful when they can be used with grouping variables. Split - Apply - Combine.

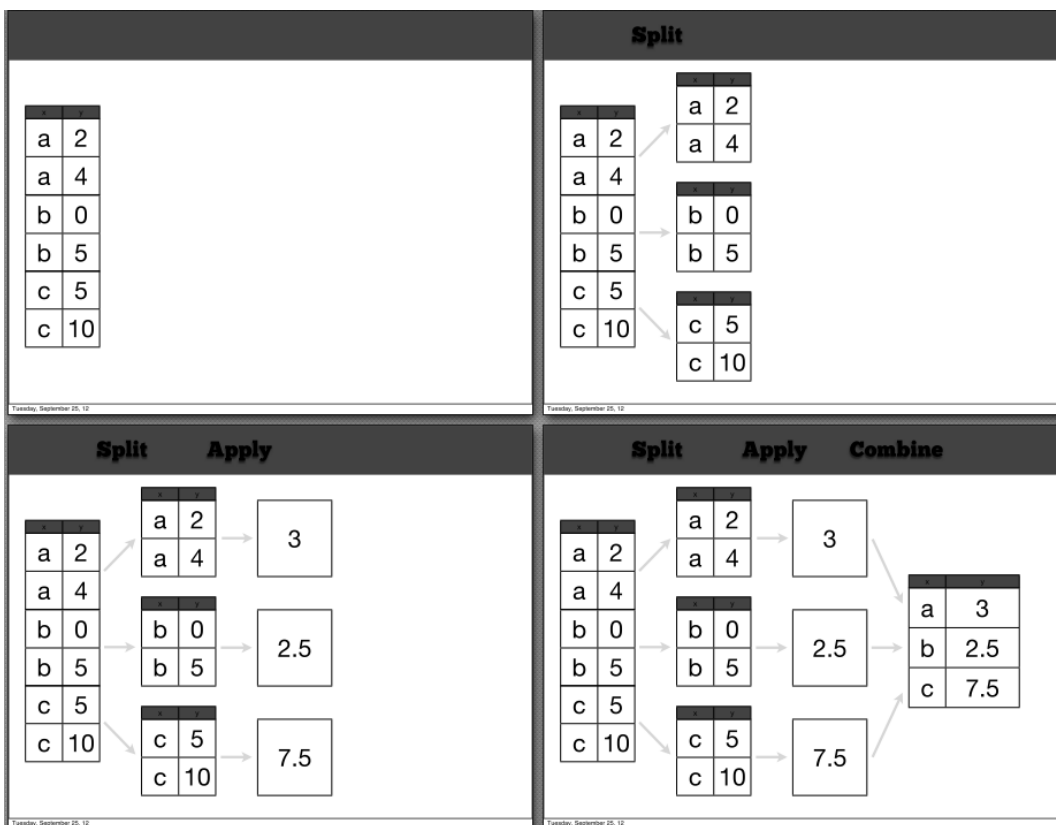


Image from Hadley Wickham UseR tutorial June 2014 <http://www.dropbox.com/sh/i8qnlwmuieicxc/AAAg9tIKoIm7WZKIyK25lh6a>

3.6.2 `group_by()`

First use the `group_by()` function to group the data (determines how to split), then apply function(s) to each group using the `summarise()` function. Note: grouping should be applied on discrete variables (categorical, factor, or maybe integer valued columns).

```

flights %>%
  group_by(origin, dest) %>% # group by both origin and dest
  summarize(max.delay = max(arr_delay, na.rm=TRUE),
            avg.delay = mean(arr_delay, na.rm=TRUE),
            min.delay = min(arr_delay, na.rm=TRUE),
            count = n() )    # n() gives the group count
#> # A tibble: 224 x 6
#> # Groups:   origin [?]
#>   origin dest max.delay avg.delay min.delay count
#>   <chr>   <chr>     <dbl>    <dbl>    <dbl> <int>
#> 1 EWR    ALB         328     14.4      -34    439
#> 2 EWR    ANC          39     -2.5      -47     8
#> 3 EWR    ATL        796     13.2      -39   5022
#> 4 EWR    AUS        349    -0.474     -59   968
#> 5 EWR    AVL        228     8.80     -26   265
#> 6 EWR    BDL        266     7.05     -43   443
#> 7 EWR    BNA        364    12.7      -41  2336
#> 8 EWR    BOS        422     4.78     -47  5327
#> 9 EWR    BQN        208    10.9      -43   297
#> 10 EWR   BTV        306    12.2      -41   931
#> # ... with 214 more rows

```

- `count(...)` is a shortcut for `group_by(...) %>% summarize(n=n())`
- `ungroup()` removes the grouping

3.6.3 Grouped Mutate and Filter

- When data is *grouped*, `mutate()` and `filter()` operate on each group independently

```

#- proportion of carrier at each dest
flights %>%
  count(dest, carrier) %>%
  group_by(dest) %>% # group by dest
  mutate(total=sum(n), p=n/sum(n)) %>% # grouped mutate sum(n) is by group
  arrange(desc(total), -p) # arrange by most freq dest and prop
#> # A tibble: 314 x 5
#> # Groups:   dest [105]
#>   dest carrier    n total      p
#>   <chr>   <chr> <int> <int>    <dbl>
#> 1 ORD    UA    6984 17283 0.404
#> 2 ORD    AA    6059 17283 0.351
#> 3 ORD    MQ    2276 17283 0.132
#> 4 ORD    9E    1056 17283 0.0611
#> 5 ORD    B6     905 17283 0.0524
#> 6 ORD    EV       2 17283 0.000116
#> 7 ORD    OO       1 17283 0.0000579
#> 8 ATL    DL   10571 17215 0.614
#> 9 ATL    FL    2337 17215 0.136
#> 10 ATL   MQ    2322 17215 0.135
#> # ... with 304 more rows

```

3.7 Relational Data and Joins

Joins are used to combine or merge two datasets. This is a major aspect of SQL.

3.7.1 Mutating Joins

See 13.4 of R4DS

- `inner_join(x, y)` only includes observations that having matching x and y key values. Rows of x can be dropped/filtered.
- `left_join(x, y)` includes all observations in x, regardless of whether they match or not. This is the most commonly used join because it ensures that you don't lose observations from your primary table.
- `right_join(x, y)` includes all observations in y. It's equivalent to `left_join(y, x)`, but the columns will be ordered differently.
- `full_join()` includes all observations from x and y.
- The left, right and full joins are collectively known as **outer joins**. When a row doesn't match in an outer join, the new variables are filled in with missing values.
 - **outer joins** will fill any missing values with NA
- If there are duplicate keys, all combinations are returned.
- Missing values are given NA.

3.8 Data Importing

3.8.1 readr package

- See <https://readr.tidyverse.org/>
- Keep the [data import cheatsheet](#) handy

3.8.2 readxl package

- See <https://readxl.tidyverse.org/> for importing excel files

3.9 Tidy Data with the tidyr package

- <https://tidyr.tidyverse.org/>
- Keep the [data import cheatsheet](#) handy. Page two describes the `tidyr` functionality

3.9.1 Why Tidy Data?

- Tidy data (in form of a data frame) is usually the best form for analysis
 - some exceptions are for modeling (e.g., matrix manipulations and algorithms)
- For presentation of data (e.g., in tables), non-tidy form can often do better
- the functions in `tidyr` usually allow us to convert from non-tidy to tidy for analysis and also from tidy to non-tidy for presentation

3.9.2 Main tidyr functions

function	description
<code>spread()</code>	Spreads a pair of key:value columns into a set of tidy columns
<code>gather()</code>	Gather takes multiple columns and collapses into key-value pairs, duplicating all other columns as needed. You use <code>gather()</code> when you notice that you have columns that are not variables
<code>separate()</code>	turns a single character column into multiple columns
<code>unite()</code>	paste together multiple columns into one (reverse of <code>separate()</code>)

4 RMarkdown

- Homework will be submitted in Rmd and (pdf/html) format
- When you *knit* a Rmd, it:
 1. starts a new instance of R (clean environment)
 2. in the current directory
- Any data or code must first be put into the Rmd file
 - The Rmd won't know about anything in another script or in your R environment
 - Any `source()` or data paths are relative to the current directory of the Rmd
- A homework template will be provided for each homework