MFE Programming Workshop Class 3

Rob Richmond

November 19, 2015

UCLA Anderson

Welcome to Hadley-ville

- Hadley Wickham is practically famous in the R world
- He's developed a ridiculous number of useful packages
- `ggplot2`
- Today we will look at `dplyr` and `tidyr`

`dplyr` intro

- `dplyr` is a package for data manipulation
- `data.table` is another fantastic package of this type
    - I'll post a solution to today's lab using both
- These slides are a cut down version of the dplyr introduction vignette

Data: nycflights13

- To explore the basic data manipulation verbs of dplyr, we'll start with the built in `nycflights13` data frame
- This dataset contains all flights that departed from New York City in 2013

```
library(dplyr)
library(nycflights13)
head(flights)
```

```
## Source: local data frame [6 x 16]
##
##    year month   day dep_time dep_delay arr_time arr_delay
##   (int) (int) (int)    (int)     (dbl)    (int)     (dbl)
## 1  2013     1     1      517         2      830        11
## 2  2013     1     1      533         4      850        20
## 3  2013     1     1      542         2      923        33
## 4  2013     1     1      544        -1     1004       -18
## 5  2013     1     1      554        -6      812       -25
## 6  2013     1     1      554        -4      740        12
```

Single table verbs

Dplyr aims to provide a function for each basic verb of data manipulation:

- `filter()` (and `slice()`)
- `arrange()`
- `select()` (and `rename()`)
- `distinct()`
- `mutate()` (and `transmute()`)
- `summarise()`
- `sample_n()` and `sample_frac()`

Filter rows with `filter()`

- `filter()` allows you to select a subset of rows in a data frame.
- The first argument is the name of the data frame.
- The second and subsequent arguments are the expressions that filter the data frame
- Select all flights on January 1st with:

```
filter(flights, month == 1, day == 1)
```

```
## Source: local data frame [842 x 16]
##
##      year month   day dep_time dep_delay arr_time arr_delay
##     (int) (int) (int)    (int)     (dbl)    (int)     (dbl)
## 1   2013     1     1      517         2      830        11
## 2   2013     1     1      533         4      850        20
## 3   2013     1     1      542         2      923        33
## 4   2013     1     1      544        -1     1004       -18
## 5   2013     1     1      554        -6      812       -25
```

Select rows by position

- To select rows by position, use `slice()`

```
slice(flights, 1:10)
```

```
## Source: local data frame [10 x 16]
##
##     year month   day dep_time dep_delay arr_time arr_delay
##    (int) (int) (int)    (int)     (dbl)    (int)     (dbl)
## 1   2013     1     1      517         2      830        11
## 2   2013     1     1      533         4      850        20
## 3   2013     1     1      542         2      923        33
## 4   2013     1     1      544        -1     1004       -18
## 5   2013     1     1      554        -6      812       -25
## 6   2013     1     1      554        -4      740        12
## 7   2013     1     1      555        -5      913        19
## 8   2013     1     1      557        -3      709       -14
## 9   2013     1     1      557        -3      838        -8
## 10  2013     1     1      558        -2      753         8
## Variables not shown: carrier (chr), tailnum (chr), flight (int), or
##   (chr), dest (chr), air_time (dbl), distance (dbl), hour (dbl), mi
##   (dbl)
```

Arrange rows with `arrange()`

- `arrange()` works similarly to `filter()` except that instead of filtering or selecting rows, it reorders them

```
arrange(flights, year, month, day)
```

```
## Source: local data frame [336,776 x 16]
##
##     year month   day dep_time dep_delay arr_time arr_delay
##    (int) (int) (int)    (int)     (dbl)    (int)     (dbl)
## 1   2013     1     1      517         2      830        11
## 2   2013     1     1      533         4      850        20
## 3   2013     1     1      542         2      923        33
## 4   2013     1     1      544        -1     1004       -18
## 5   2013     1     1      554        -6      812       -25
## 6   2013     1     1      554        -4      740        12
## 7   2013     1     1      555        -5      913        19
## 8   2013     1     1      557        -3      709       -14
## 9   2013     1     1      557        -3      838        -8
## 10  2013     1     1      558        -2      753         8
## ..   ...   ...   ...      ...       ...      ...       ...
```

Use desc( ) to order a column in descending order

```
arrange(flights, desc(arr_delay))
```

```
## Source: local data frame [336,776 x 16]
##
##      year month   day dep_time dep_delay arr_time arr_delay
##     (int) (int) (int)    (int)     (dbl)    (int)     (dbl)
## 1   2013     1     9      641      1301     1242      1272
## 2   2013     6    15     1432      1137     1607      1127
## 3   2013     1    10     1121      1126     1239      1109
## 4   2013     9    20     1139      1014     1457      1007
## 5   2013     7    22      845      1005     1044       989
## 6   2013     4    10     1100       960     1342       931
## 7   2013     3    17     2321       911      135       915
## 8   2013     7    22     2257       898      121       895
## 9   2013    12     5      756       896     1058       878
## 10  2013     5     3     1133       878     1250       875
## ..   ...   ...   ...      ...       ...      ...       ...
## Variables not shown: carrier (chr), tailnum (chr), flight (int), or
##   (chr), dest (chr), air_time (dbl), distance (dbl), hour (dbl), mi
##   (dbl).
```

Select columns with `select()`

- `select()` allows you to rapidly zoom in on a useful subset using operations that usually only work on numeric variable positions:

```
# Select columns by name
select(flights, year, month, day)


## Source: local data frame [336,776 x 3]
##
##     year month   day
##    (int) (int) (int)
## 1   2013     1     1
## 2   2013     1     1
## 3   2013     1     1
## 4   2013     1     1
## 5   2013     1     1
## 6   2013     1     1
## 7   2013     1     1
## 8   2013     1     1
## 9   2013     1     1
```

You can rename variables with `rename()`

```
rename(flights, tail_num = tailnum)
```

```
## Source: local data frame [336,776 x 16]
##
##     year month   day dep_time dep_delay arr_time arr_delay
##    (int) (int) (int)    (int)     (dbl)    (int)     (dbl)
## 1   2013     1     1      517         2      830        11
## 2   2013     1     1      533         4      850        20
## 3   2013     1     1      542         2      923        33
## 4   2013     1     1      544        -1     1004       -18
## 5   2013     1     1      554        -6      812       -25
## 6   2013     1     1      554        -4      740        12
## 7   2013     1     1      555        -5      913        19
## 8   2013     1     1      557        -3      709       -14
## 9   2013     1     1      557        -3      838        -8
## 10  2013     1     1      558        -2      753         8
## ..   ...   ...   ...      ...       ...      ...       ...
## Variables not shown: carrier (chr), tail_num (chr), flight (int), o
##   (chr), dest (chr), air_time (dbl), distance (dbl), hour (dbl), mi
##   (dbl).
```

Extract distinct (unique) rows

- A common use of `select()` is to find the values of a set of variables.
- This is particularly useful in conjunction with the `distinct()` verb

```
distinct(select(flights, tailnum))
```

```
## Source: local data frame [4,044 x 1]
##
##    tailnum
##      (chr)
## 1   N14228
## 2   N24211
## 3   N619AA
## 4   N804JB
## 5   N668DN
## 6   N39463
## 7   N516JB
## 8   N829AS
```

Add new columns with `mutate()`

```
mutate(flights,
  gain = arr_delay - dep_delay,
  speed = distance / air_time * 60)
```

```
## Source: local data frame [336,776 x 18]
##
##      year month   day dep_time dep_delay arr_time arr_delay
##     (int) (int) (int)    (int)     (dbl)    (int)     (dbl)
## 1   2013     1     1      517         2      830        11
## 2   2013     1     1      533         4      850        20
## 3   2013     1     1      542         2      923        33
## 4   2013     1     1      544        -1     1004       -18
## 5   2013     1     1      554        -6      812       -25
## 6   2013     1     1      554        -4      740        12
## 7   2013     1     1      555        -5      913        19
## 8   2013     1     1      557        -3      709       -14
## 9   2013     1     1      557        -3      838        -8
## 10  2013     1     1      558        -2      753         8
## ..   ...   ...   ...      ...       ...      ...       ...
## Variables not shown: carrier (chr), tailnum (chr), flight (int), or
##   (chr), dest (chr), air_time (dbl), distance (dbl), hour (dbl), mi
```

If you only want to keep the new variables, use `transmute()`

```
transmute(flights,
  gain = arr_delay - dep_delay,
  gain_per_hour = gain / (air_time / 60)
)
```

```
## Source: local data frame [336,776 x 2]
##
##     gain gain_per_hour
##    (dbl)         (dbl)
## 1      9      2.378855
## 2     16      4.229075
## 3     31     11.625000
## 4    -17     -5.573770
## 5    -19     -9.827586
## 6     16      6.400000
## 7     24      9.113924
## 8    -11    -12.452830
## 9     -5     -2.142857
## 10    10      4.347826
## ..   ...           ...
```

Summarise values with `summarise()`

- The last verb is **`summarise()`**. It collapses a data frame to a single row:

```
summarise(flights,
  delay = mean(dep_delay, na.rm = TRUE))


## Source: local data frame [1 x 1]
##
##       delay
##       (dbl)
## 1 12.63907
```

Commonalities

- The syntax and function of all these verbs are very similar:
    - The first argument is a data frame.
    - The subsequent arguments describe what to do with the data frame.
    - The result is a new data frame
- Together these properties make it easy to chain together multiple simple steps to achieve a complex result.

Grouped operations

- These verbs are useful on their own, but they become really powerful when you apply them to groups of observations
- In dplyr, you do this by with the `group_by()` function
- It breaks down a dataset into specified groups of rows

Grouped operations (cont.)

Grouping affects the verbs as follows:

- grouped `select()` is the same as ungrouped `select()`, except that grouping variables are always retained.
- grouped `arrange()` orders first by the grouping variables
- `mutate()` and `filter()` are most useful in conjunction with window functions (like `rank()`, or `min(x) == x`). They are described in detail in `vignette("window-functions")`.
- `sample_n()` and `sample_frac()` sample the specified number/fraction of rows in each group.
- `slice()` extracts rows within each group.
- `summarise()` is powerful and easy to understand, as described in more detail below.

group_by Example

For example, we could use these to find the number of planes and the number of flights that go to each possible destination:

```
destinations <- group_by(flights, dest)
summarise(destinations,
  planes = n_distinct(tailnum),
  flights = n()
)
```

```
## Source: local data frame [105 x 3]
##
##     dest planes flights
##    (chr)  (int)   (int)
## 1    ABQ    108     254
## 2    ACK     58     265
## 3    ALB    172     439
## 4    ANC      6       8
## 5    ATL   1180   17215
## 6    AUS    993    2439
## 7    AVL    159     275
## 8    BDL    186     443
```

Chaining

- The dplyr API is functional — function calls don't have side-effects.
- You must always save their results. UGLY
- To get around this problem, dplyr provides the %>% operator
- `x %>% f(y)` turns into `f(x, y)`

```
flights %>%
    group_by(year, month, day) %>%
    select(arr_delay, dep_delay) %>%
    summarise(arr = mean(arr_delay, na.rm = TRUE),
              dep = mean(dep_delay, na.rm = TRUE)) %>%
    filter(arr > 30 | dep > 30)


## Source: local data frame [49 x 5]
## Groups: year, month [11]
##
##     year month   day      arr      dep
```

Multiple table verbs

dplyr implements the four most useful SQL joins:

- `inner_join(x, y)`: matching x + y
- `left_join(x, y)`: all x + matching y
- `semi_join(x, y)`: all x with match in y
- `anti_join(x, y)`: all x without match in y

And provides methods for:

- `intersect(x, y)`: all rows in both x and y
- `union(x, y)`: rows in either x or y
- `setdiff(x, y)`: rows in x, but not y

Sample data

```r
library(tidyr)

stocks <- data.frame(
    time = as.Date('2009-01-01') + 0:9,
    X = rnorm(10, 0, 1),
    Y = rnorm(10, 0, 2),
    Z = rnorm(10, 0, 4)
)

stocks
```

```
##          time           X          Y          Z
## 1  2009-01-01 -1.4303951  2.0325580 -3.9734501
## 2  2009-01-02 -1.7013768 -1.7610001  0.1879096
## 3  2009-01-03 -1.0428679 -2.0256532  5.2445267
## 4  2009-01-04 -0.4309410  1.9164783 -4.5941203
## 5  2009-01-05  0.8220322  0.2103667 -2.7040649
## 6  2009-01-06  1.6747590  0.1976161  1.2507864
## 7  2009-01-07  0.6309603 -2.1840579 -1.4701481
## 8  2009-01-08  1.3935541  2.9236443 -0.5024605
## 9  2009-01-09  1.7385731  4.9180285  7.9599304
```

Bring columns together with `gather()`

```
stocksm <- stocks %>% gather(stock, price, -time)
stocksm
```

```
##          time stock      price
## 1  2009-01-01     X -1.4303951
## 2  2009-01-02     X -1.7013768
## 3  2009-01-03     X -1.0428679
## 4  2009-01-04     X -0.4309410
## 5  2009-01-05     X  0.8220322
## 6  2009-01-06     X  1.6747590
## 7  2009-01-07     X  0.6309603
## 8  2009-01-08     X  1.3935541
## 9  2009-01-09     X  1.7385731
## 10 2009-01-10     X  1.8460525
## 11 2009-01-01     Y  2.0325580
## 12 2009-01-02     Y -1.7610001
## 13 2009-01-03     Y -2.0256532
## 14 2009-01-04     Y  1.9164783
## 15 2009-01-05     Y  0.2103667
## 16 2009-01-06     Y  0.1976161
## 17 2009-01-07     Y -2.1840579
```

Split a column with spread( )

```
stocksm %>% spread(stock, price)
```

```
##          time          X          Y          Z
## 1  2009-01-01 -1.4303951  2.0325580 -3.9734501
## 2  2009-01-02 -1.7013768 -1.7610001  0.1879096
## 3  2009-01-03 -1.0428679 -2.0256532  5.2445267
## 4  2009-01-04 -0.4309410  1.9164783 -4.5941203
## 5  2009-01-05  0.8220322  0.2103667 -2.7040649
## 6  2009-01-06  1.6747590  0.1976161  1.2507864
## 7  2009-01-07  0.6309603 -2.1840579 -1.4701481
## 8  2009-01-08  1.3935541  2.9236443 -0.5024605
## 9  2009-01-09  1.7385731  4.9180285  7.9599304
## 10 2009-01-10  1.8460525 -2.2721882  9.0867334
```

```
stocksm %>% spread(time, price)
```

```
##   stock 2009-01-01 2009-01-02 2009-01-03 2009-01-04 2009-01-05 2009
## 1     X  -1.430395 -1.7013768  -1.042868  -0.430941  0.8220322  1.6
## 2     Y   2.032558 -1.7610001  -2.025653   1.916478  0.2103667  0.1
## 3     Z  -3.973450  0.1879096   5.244527  -4.594120 -2.7040649  1.2
```

spread() and gather() are complements

```
df <- data.frame(x = c("a", "b"), y = c(3, 4), z = c(5, 6))
df
```

```
##   x y z
## 1 a 3 5
## 2 b 4 6
```

```
df %>% spread(x, y) %>% gather(x, y, a:b, na.rm = TRUE)
```

```
##   z x y
## 1 5 a 3
## 4 6 b 4
```

There’s much more

- As usual, read the vignette on the CRAN page