MFE R Programming Workshop

Week 3

Brett Dunn

Fall 2016

Data Munging

Hadley Wickham

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

dplyr

`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,4)
```

```
## Source: local data frame [4 x 19]
##
##    year month   day dep_time sched_dep_time dep_delay
##   (int) (int) (int)    (int)          (int)     (dbl)
## 1  2013     1     1      517            515         2
## 2  2013     1     1      533            529         4
## 3  2013     1     1      542            540         2
## 4  2013     1     1      544            545        -1
```

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 19]
##
##      year month   day dep_time sched_dep_time dep_delay
##     (int) (int) (int)    (int)          (int)     (dbl)
## 1   2013     1     1      517            515         2
## 2   2013     1     1      533            529         4
## 3   2013     1     1      542            540         2
## 4   2013     1     1      544            545        -1
## 5   2013     1     1      554            600        -6
## 6   2013     1     1      554            558        -4
```

Select rows by position

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

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

```
## Source: local data frame [10 x 19]
##
##      year month   day dep_time sched_dep_time dep_delay
##     (int) (int) (int)    (int)          (int)     (dbl)
## 1   2013     1     1      517            515         2
## 2   2013     1     1      533            529         4
## 3   2013     1     1      542            540         2
## 4   2013     1     1      544            545        -1
## 5   2013     1     1      554            600        -6
## 6   2013     1     1      554            558        -4
## 7   2013     1     1      555            600        -5
## 8   2013     1     1      557            600        -3
## 9   2013     1     1      557            600        -3
## 10  2013     1     1      558            600        -2
```

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 19]
##
##      year month   day dep_time sched_dep_time dep_delay
##     (int) (int) (int)    (int)          (int)     (dbl)
## 1    2013     1     1      517            515         2
## 2    2013     1     1      533            529         4
## 3    2013     1     1      542            540         2
## 4    2013     1     1      544            545        -1
## 5    2013     1     1      554            600        -6
## 6    2013     1     1      554            558        -4
## 7    2013     1     1      555            600        -5
## 8    2013     1     1      557            600        -3
## 9    2013     1     1      557            600        -3
```

Use desc() to order a column in descending order

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

```
## Source: local data frame [336,776 x 19]
##
##      year month   day dep_time sched_dep_time dep_delay
##     (int) (int) (int)    (int)          (int)     (dbl)
## 1    2013     1     9      641            900      1301
## 2    2013     6    15     1432           1935      1137
## 3    2013     1    10     1121           1635      1126
## 4    2013     9    20     1139           1845      1014
## 5    2013     7    22      845           1600      1005
## 6    2013     4    10     1100           1900       960
## 7    2013     3    17     2321            810       911
## 8    2013     7    22     2257            759       898
## 9    2013    12     5      756           1700       896
## 10   2013     5     3     1133           2055       878
## ..    ...   ...   ...      ...            ...       ...
## Variables not shown: arr_time (int), sched_arr_time (int
```

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
```

You can rename variables with `rename()`

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

```
## Source: local data frame [336,776 x 19]
##
##      year month   day dep_time sched_dep_time dep_delay
##     (int) (int) (int)    (int)          (int)     (dbl)
## 1    2013     1     1      517            515         2
## 2    2013     1     1      533            529         4
## 3    2013     1     1      542            540         2
## 4    2013     1     1      544            545        -1
## 5    2013     1     1      554            600        -6
## 6    2013     1     1      554            558        -4
## 7    2013     1     1      555            600        -5
## 8    2013     1     1      557            600        -3
## 9    2013     1     1      557            600        -3
## 10   2013     1     1      558            600        -2
## ..    ...   ...   ...      ...            ...       ...
## Variables not shown: arr_time (int), sched_arr_time (int
```

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
```

Add new columns with mutate()

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

```
## Source: local data frame [336,776 x 21]
##
##      year month   day dep_time sched_dep_time dep_delay
##     (int) (int) (int)    (int)          (int)     (dbl)
## 1    2013     1     1      517            515         2
## 2    2013     1     1      533            529         4
## 3    2013     1     1      542            540         2
## 4    2013     1     1      544            545        -1
## 5    2013     1     1      554            600        -6
## 6    2013     1     1      554            558        -4
## 7    2013     1     1      555            600        -5
## 8    2013     1     1      557            600        -3
## 9    2013     1     1      557            600        -3
## 10   2013     1     1      558            600        -2
```

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
```

Summarise values with `summarise()`

- The last verb is ‘summarise()}. It collapses a data frame to asingle 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
```

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
##     (int) (int) (int)    (dbl)    (dbl)
```

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

tidyr

Sample data

```
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  1.06263264  0.3846354   0.6182198
## 2  2009-01-02 -0.75972633  4.1446913  -4.4182804
## 3  2009-01-03  0.04539981 -0.5124680   0.7407128
## 4  2009-01-04 -0.10552157 -0.9878174  -2.9890774
## 5  2009-01-05  0.53070515  0.3024455 -10.6035637
## 6  2009-01-06 -2.81381162  3.1397657  -2.5040303
```

Bring columns together with gather()

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

```
##          time stock       price
## 1  2009-01-01     X  1.06263264
## 2  2009-01-02     X -0.75972633
## 3  2009-01-03     X  0.04539981
## 4  2009-01-04     X -0.10552157
## 5  2009-01-05     X  0.53070515
## 6  2009-01-06     X -2.81381162
## 7  2009-01-07     X -1.94841353
## 8  2009-01-08     X -0.98660587
## 9  2009-01-09     X  0.34193470
## 10 2009-01-10     X -2.05957063
## 11 2009-01-01     Y  0.38463544
## 12 2009-01-02     Y  4.14469129
## 13 2009-01-03     Y -0.51246801
## 14 2009-01-04     Y -0.98781738
```

Split a column with spread()

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

```
##          time           X          Y           Z
## 1  2009-01-01  1.06263264  0.3846354   0.6182198
## 2  2009-01-02 -0.75972633  4.1446913  -4.4182804
## 3  2009-01-03  0.04539981 -0.5124680   0.7407128
## 4  2009-01-04 -0.10552157 -0.9878174  -2.9890774
## 5  2009-01-05  0.53070515  0.3024455 -10.6035637
## 6  2009-01-06 -2.81381162  3.1397657  -2.5040303
## 7  2009-01-07 -1.94841353 -1.7766495   9.8588272
## 8  2009-01-08 -0.98660587  0.1021915  -0.2715888
## 9  2009-01-09  0.34193470  3.3832790  -5.5933985
## 10 2009-01-10 -2.05957063  1.9892377  -2.1889344
```

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

```
##   stock 2009-01-01 2009-01-02  2009-01-03 2009-01-04  20
## 1     X  1.0626326 -0.7597263  0.04539981 -0.1055216   (
```

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