dpolyr

Matthew Flickinger, Ph.D.
University of Michigan
BDSI June 21, 2019
About Me

- Matthew Flickinger
- Ph.D. in Biostatistics, UM
- Application Programmer/Analyst Senior Center for Statistical Genetics
- Answer R questions on Stack Overflow
- RStudio certified "tidyverse" trainer
Quick R Facts
Function Syntax

round(14.752, digits=1)
Function Syntax

name parenthesis "call" the function

```
round(14.752, digits=1)
```
Function Syntax

round(14.752, digits=1)

argument (value passed to function)

comma (separate arguments)

argument name
Creating variables

fruit <- c("apples", "oranges", "bananas")
primes <- c(1, 2, 3, 5, 7, 11)
numbers <- 1:10
age <- 22
age <- age + 1

"<-" assigns a value to a variable
The tidyverse is an opinionated collection of R packages designed for data science. All packages share an underlying design philosophy, grammar, and data structures.
R Packages

Packages add new functions and data into R

install.packages("dplyr")

Run **once** per machine
Downloads package to your computer

library(dplyr)

Run **each time** you start R
Load package into R's "brain"
dplyr
Motivation

- Analysts spend a lot of time manipulating and summarizing data
- Base R provides many functions for this, but
  - the syntax is sometimes verbose or "ugly"
  - the functions can be slow for big data
- dplyr exists to make code easier to read and faster
Install and load dplyr

Install via tidyverse

install.packages("tidyverse")
library(tidyverse)

OR install directly

install.packages("dplyr")
library(dplyr)

This guide assumes you're running dplyr >0.8.1 (released May 14, 2019)
Sample data

We will be using a data set containing all out-bound flights from NYC in 2013

Available as an R package

```r
install.packages("nycflights13")
library(nycflights13)
```
nycflights13 tables
The `flights` table contains data on departure and arrival times of flights for the year 2013. The table includes columns for:

- `year`: The year of the flight, with values ranging from 2013 to 2013.
- `month`: The month of the flight, with values ranging from 1 to 1.
- `day`: The day of the flight, with values ranging from 1 to 1.
- `dep_time`: The scheduled departure time in minutes.
- `sched_dep_time`: The scheduled departure time in minutes.
- `dep_delay`: The departure delay in minutes.
- `arr_time`: The scheduled arrival time in minutes.
- `sched_arr_time`: The scheduled arrival time in minutes.

The variables not shown include:

- `arr_delay`: Arrival delay in minutes.
- `carrier`: The airline carrier.
- `flight`: The flight number.
- `tailnum`: The tail number.
- `origin`: The origin airport.
- `dest`: The destination airport.
- `air_time`: Airline time in minutes.
- `distance`: Distance in miles.
- `hour`: Hour of the flight.
- `minute`: Minute of the flight.
- `time_hour`: Time in hours and minutes.

The table starts with the first row for the year 2013, month 1, day 1, and continues with similar entries for subsequent rows.
Basic single-table verbs
Basic dplyr verbs

- `filter()` - keep rows matching desired properties
- `select()` - choose which columns you want to extract
- `arrange()` - sort rows
- `mutate()` - create new columns
- `summarize()` - collapse rows into summaries
- `group_by()` - operate on subsets of rows at a time
dpolyr verb properties

- Always take a data source as the first argument
- Returns a new data object (never updates/replaces original)
- Specify columns as unquoted strings (symbols)
Filtering Rows

Find all flights to Detroit (DTW) in June (2013)

```r
filter(flights, dest=="DTW" & month==6)
```

Base R

```r
flights[flights$dest=="DTW" & flights$month==6, ]
subset(flights, dest=="DTW" & month==6)
```
How about your home airport?

How many flights went to your “home” airport?

filter(flights, dest=="   ")
    # ^ put airport code here

Or try these
- "HNL" (Honolulu, Hawaii)
- "SLC" (Salt Lake City, Utah)
Comparisons

==  Equal  month == 6
!=  Not Equal  month != 6
<, <=  Less than (or equal)  month > 2 & month < 7
>, >=  Greater than (or equal)  month < 3 | month > 6
&  And
|  Or
!  Not
%in%  Matches one of  month %in% c(1, 12)
Selecting columns

Select specific columns

```python
select(flights, dep_time, arr_time, carrier)
```

Exclude columns

```python
select(flights, -year, -tailnum)
```

Select column range

```python
select(flights, month:dep_delay)
```
Selecting columns ... part 2

- `select(flights, starts_with("d"))`
- `select(flights, ends_with("time"))`
- `select(flights, contains("arr"))`
- `select(flights, -starts_with("d"))`
- `select(flights, flight, everything())`

See "?select" for complete list and examples
Can we select AND filter?

# Temporary variables
filtered <- filter(flights, dest=="DTW")
select(filtered, carrier)

# Nested calls
select(filter(flights, dest=="DTW"), carrier)

# But these can get messy
Verb composition with pipes

```r
flights %>%
  filter(dest == "DTW") %>%
  select(carrier)
```

%>% is the "pipe" operator (often read as "then")

RStudio: ctrl-shift-M (win)    cmd-shift-M (mac)

#pipe
Verb composition with pipes

The %>% operator passes the result from the left side to the first argument of the right side

\[
a \ %>% \ x() \ %>% \ y() \ %>% \ z() \ \ <-> \ \ z(y(x(a)))
\]

\[
a \ %>% \\
x() \ %>% \\
y() \ %>% \\
z()
\]

\[
x(a) \ %>% \\
y() \ %>% \\
z()
\]

\[
y(x(a)) \ %>% \\
z()
\]

\[
z(y(x(a)))
\]
Try using chains

Rewrite the following as a chain

\[
\text{round(exp(sin(.5)),2)}
\]

# [1] 1.62
Sort data

Use `arrange()` to sort your rows
```r
flights %>% arrange(sched_dep_time)
```

Use `desc()` to reverse the sort order of a column
```r
flights %>% arrange(month, desc(day))
```

You can sort on functions of variables
```r
flights %>%
  arrange(desc(dep_time - sched_dep_time))
```
What is the last day that flight 4401 arrived in Detroit?

```r
flights %>%
  filter(      ) %>%
  arrange(     ) %>%
  select(      )
```
Create new variables

Mutate allows you to create columns using existing values

```
flights %>%
  mutate(speed = distance/(air_time/60)) %>%
  arrange(desc(speed)) %>%
  select(flight, speed)
```

Remember, changes are not saved to "flights", be sure to save the result if you want to use it later

```
new_flights <- flights %>% mutate(…)
```
Use new variables right away

The arguments to mutate are run in the order they appear

```r
flights %>%
  mutate(
    dist_km = distance * 1.61,
    hours = air_time / 60,
    kph = dist_km/hours ) %>%
  select(flight, kph)
```

Be careful! You can overwrite existing variables
Summarize data

You generally use `summarize()` to reduce the number of rows in your data by specifying summary functions for each of the columns

```r
flights %>%
  filter(!is.na(arr_delay)) %>%
  summarize(avg_arr_delay = mean(arr_delay))
```

Useful summary functions: `mean()`, `median()`, `var()`, `sd()`, `min()`, `max()`, `first()`, `last()`, `n()`, `n_distinct()`
Create one statement to calculate the minimum ($\text{min}$) and maximum ($\text{max}$) departure delay ($\text{dep\_delay}$). Be sure to remove rows that have missing values.

```r
flights %>%
  filter( ) %>%
  mutate( , )
```
Grouping data

- Often you want to perform summaries for groups of rows at a time
- The `group_by()` function allows you to specify columns that define groups
- Functions like `mutate()` and `summarize()` are performed for each group
group_by() + summarize() example

flights %>%
  filter(!is.na(arr_delay)) %>%
  group_by(carrier) %>%
  summarize(avg_arr_delay = mean(arr_delay))

# A tibble: 16 x 2
  carrier avg_arr_delay
     <chr>        <dbl>
1      9E        7.3796692
2      AA       0.3642909
3      AS       -9.9308886
4      B6        9.4579733
...
**group_by() + mutate() example**

```r
flights %>%
  filter(!is.na(arr_delay)) %>%
  group_by(carrier) %>%
  mutate(avg_arr_delay = mean(arr_delay)) %>%
  select(carrier, arr_delay, avg_arr_delay)
```

# A tibble: 327,346 x 3
# Groups:   carrier [16]

<table>
<thead>
<tr>
<th>carrier</th>
<th>arr_delay</th>
<th>avg_arr_delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>UA</td>
<td>11</td>
<td>3.5580111</td>
</tr>
<tr>
<td>UA</td>
<td>20</td>
<td>3.5580111</td>
</tr>
<tr>
<td>AA</td>
<td>33</td>
<td>0.3642909</td>
</tr>
<tr>
<td>B6-</td>
<td>-18</td>
<td>9.4579733</td>
</tr>
</tbody>
</table>

...
Combining `group_by()` with transformations

**mutate()**
- Will not change the number of rows
- Collapsing functions like `max()` will return the max for each group
- Keep all existing columns and adds new ones

**summarize()**
- Returns one row per group
- Only returns columns that are used as groups and those new values created from collapsing functions
Count summaries

The `count()` function is a special combination of `summarize()` + `group_by()` to see how often certain values are repeated

```r
flights %>%
count(carrier)
```

```
# A tibble: 16 x 2
  carrier n
  <chr> <int>
1 9E    18460
2 AA   32729
3 AS   714
4 B6   54635
5 DL   48110
6 EV   54173
...  
```
Special Shortcuts

**summarize_all()**/**mutate_all()**
Apply function to all non-grouped columns

**summarize_at()**/**mutate_at()**
Apply function to chosen columns

```r
flights %>%
  summarize_at(vars(ends_with("time")),
               mean, na.rm=T)
```

#summarize_at
Summarization exercises

Calculate was the longest arrival delay for each carrier. Which carrier’s longest delay was the shortest compared to all the other carriers?

```r
flights %>%
  filter() %>%
  group_by() %>%
  summarize() %>%
  arrange()
```
Merging data
What are these carrier codes?

flights %>%
  filter(!is.na(arr_delay)) %>%
  group_by(carrier) %>%
  summarize(avg_arr_delay = mean(arr_delay))

# A tibble: 16 x 2
  carrier avg_arr_delay
    <chr>        <dbl>
1     9E        7.3796692
2     AA        0.3642909
3     AS       -9.9308886
4     B6        9.4579733
5...

#group_by-1
### "airlines" table

```r
> airlines
Source: local data frame [16 x 2]

carrier name
(chr) (chr)
1 9E Endeavor Air Inc.
2 AA American Airlines Inc.
3 AS Alaska Airlines Inc.
4 B6 JetBlue Airways
5 DL Delta Air Lines Inc.
6 EV ExpressJet Airlines Inc.
...
```
Join flights to airlines

```r
> flights %>%
  filter(!is.na(arr_delay)) %>%
  group_by(carrier) %>%
  summarize(avg_arr_delay = mean(arr_delay)) %>%
  left_join(airlines)
Joining by: "carrier"

Source: local data frame [16 x 3]

<table>
<thead>
<tr>
<th>carrier</th>
<th>avg_arr_delay</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>9E</td>
<td>7.3796692</td>
<td>Endeavor Air Inc.</td>
</tr>
<tr>
<td>AA</td>
<td>0.3642909</td>
<td>American Airlines Inc.</td>
</tr>
<tr>
<td>AS</td>
<td>-9.9308886</td>
<td>Alaska Airlines Inc.</td>
</tr>
<tr>
<td>B6</td>
<td>9.4579733</td>
<td>JetBlue Airways</td>
</tr>
<tr>
<td>DL</td>
<td>1.6443409</td>
<td>Delta Air Lines Inc.</td>
</tr>
</tbody>
</table>
```

#left_join
Types of joins (merges)

inner_join(x, y)
full_join(x, y)
left_join(x, y)
right_join(x, y)

Table join images courtesy http://r4ds.had.co.nz/relational-data.html
Types of joins (merges)

- inner_join(x, y)
- full_join(x, y)
- left_join(x, y)
- right_join(x, y)

Diagram illustrating join operations with two sets of data:

```
<table>
<thead>
<tr>
<th>key</th>
<th>val_x</th>
<th>val_y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>x1</td>
<td>y1</td>
</tr>
<tr>
<td>2</td>
<td>x2</td>
<td>y2</td>
</tr>
</tbody>
</table>
```
Types of joins (merges)

- inner_join(x, y)
- full_join(x, y)
- left_join(x, y)
- right_join(x, y)
Types of joins (merges)

- inner\_join(x, y)
- left\_join(x, y)
- full\_join(x, y)
- right\_join(x, y)
What type of join is this?

**x**

```r
# A tibble: 4 x 2
  id  count
  <dbl> <dbl>
1     1     5
2     2    12
3     4    14
4     5    16
```

**y**

```r
# A tibble: 4 x 2
  id  color
  <dbl> <chr>
1     1 red
2     3 blue
3     4 green
4     5 orange
```

**??**

```r
# A tibble: 4 x 3
  id  count color
  <dbl> <dbl> <chr>
1     1     5 red
2     3  NA  blue
3     4    14 green
4     5    16 orange
```

A) `left_join(x, y)`  
B) `right_join(x, y)`  
C) `full_join(x, y)`  
D) `inner_join(x, y)`
Non-merging joins

These "joins" do not add any new columns to your data but are subsetting with multi-column matches (like a multi-column `%in%`)

`semi_join()`
  Only keep rows in left table with matches in right

`anti_join()`
  Drop rows in left table with matches in right
Join by

By default the join commands will join two tables based on all matching column names

flights %>% inner_join(planes)

# returns many fewer rows!

You can control the joining by specifying the column names

flights %>% inner_join(planes, by = "tailnum")
Other dplyr functions
Subsetting observations

top_n()
   Return top or bottom entries
   flights %>% top_n(3, air_time)

sample_n()
   Randomly choose n rows
   flights %>% sample_n(3)

distinct()
   Returns unique combinations of values (like `count()`) without the count
   flights %>% distinct(year, month)
Counting functions

n() and n_distinct()

Number of (distinct) values is a vector
Can only be used within summarize(), mutate(), filter()

flights %>%
  group_by(tailnum) %>%
  summarize(
    routes = n_distinct(flight),
    flights = n()
  )
What values made this plot?

flights %>%
  filter(!is.na(tailnum)) %>%
  group_by(tailnum) %>%
  summarize(x=??, y=??)

A) n_distinct(flight) n()
B) n()
C) n()
D) n(tailnum)

x=  
y=
   n()
   n_distinct(tailnum)
   n_distinct(flight)
   n(flight)
Lead/lag functions

"Shift" values so you can compare current value to a previous or following value

growth <- tibble(
  age = 2:9,
  height = c(33.7, 37.0, 39.4, 42.2, 45.5, 47.7, 50.6, 52.7))

growth %>%
  mutate(
    prevh = lag(height),
    nexth = lead(height),
    growth = height - prevh)

<table>
<thead>
<tr>
<th>age</th>
<th>height</th>
<th>prevh</th>
<th>nexth</th>
<th>growth</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>33.7</td>
<td>NA</td>
<td>37</td>
<td>NA</td>
</tr>
<tr>
<td>2</td>
<td>37</td>
<td>33.7</td>
<td>39.4</td>
<td>3.30</td>
</tr>
<tr>
<td>3</td>
<td>39.4</td>
<td>37</td>
<td>42.2</td>
<td>2.40</td>
</tr>
<tr>
<td>4</td>
<td>42.2</td>
<td>39.4</td>
<td>45.5</td>
<td>2.8</td>
</tr>
<tr>
<td>5</td>
<td>45.5</td>
<td>42.2</td>
<td>47.7</td>
<td>3.30</td>
</tr>
<tr>
<td>6</td>
<td>47.7</td>
<td>45.5</td>
<td>50.6</td>
<td>2.2</td>
</tr>
<tr>
<td>7</td>
<td>50.6</td>
<td>47.7</td>
<td>52.7</td>
<td>2.90</td>
</tr>
<tr>
<td>8</td>
<td>52.7</td>
<td>50.6</td>
<td>NA</td>
<td>2.1</td>
</tr>
</tbody>
</table>
Conditional replacement

The `if_else()` function helps to only change values in certain cases.

```r
if_else(<condition>, <if true>, <if false>)
```

```r
flights %>%
  mutate(
    real_delay = if_else(arr_delay<0, 0, arr_delay)
  )
```

If you have more cases than TRUE/FALSE, check out the `case_when()` function
Programming with dplyr
# Writing functions with dplyr

# WORKS
flights %>%
  group_by(carrier) %>%
  summarize(delay=mean(arr_delay, na.rm=T))

# DOESN’T WORK
f <- function(x) {
  flights %>%
    group_by(x) %>%
    summarize(delay=mean(arr_delay, na.rm=T))
}
f(carrier)  # ERROR: Column `x` is unknown
f <- function(x) {
  flights %>% group_by(!!x) %>%
    summarize(delay = mean(arr_delay, na.rm=T))
}

f(quo(carrier))

g <- function(x) {
  x <- enquo(x)
  flights %>% group_by(!!x) %>%
    summarize(delay = mean(arr_delay, na.rm=T))
}
g(carrier)

# A tibble: 16 x 2
  carrier delay
  <chr>   <dbl>
1  9E      7.3796692
2  AA      0.3642909
...
Rename output (:=)

```r
h <- function(x) {
  x <- enquo(x)
  outname <- paste(quo_name(x), "delay", sep="_")
  flights %>% group_by(!!x) %>%
  summarize(!!outname := mean(arr_delay, na.rm=T))
}

h(carrier)
```

# A tibble: 16 x 2
  carrier   carrier_delay
     <chr>       <dbl>
 1      9E         7.379669
 2      AA         0.364291
 3      9E       7.3796692
 4      AA       0.3642909
 5      9E       7.3796692
 6      AA       0.3642909
 7      9E       7.3796692
 8      AA       0.3642909
 9      9E       7.3796692
10      AA       0.3642909
11      9E       7.3796692
12      AA       0.3642909
13      9E       7.3796692
14      AA       0.3642909
15      9E       7.3796692
16      AA       0.3642909

#quo_name
Loading your own data
RStudio will write your import code
Data Transformation with dplyr: CHEAT SHEET

### Manipulate Cases

#### Extract Cases

Row functions return a subset of rows as a new table, use a vector that ends in _r_ for non-standard evaluation-friendly code.

- `filter(x, ...)`
- `distinct(x, ...)`
- `sample(x, ...)`
- `sample_n(x, ...)`
- `sample_frac(x, ...)`
- `dplyr::..n()`
- `top_n(x, y)`
- `nrow(x)`

Column functions return a subset of columns as a new table, use a vector that ends in _v_ for non-standard evaluation-friendly code.

- `select(x, ...)`
- `filter(x, ...)`
- `distinct(x, ...)`
- `sample(x, ...)`
- `sample_n(x, ...)`
- `sample_frac(x, ...)`
- `dplyr::..v()`
- `top_v(x, y)`
- `ncol(x)`

#### Arrange Cases

- `arrange(x, ...)`
- `arrange_H(x, ...)`
- `arrange_9(x, ...)`
- `arrange(Term)`

#### Add Cases

- `add_column(x)`
- `add_rows(x)`
- `add_which(x)`
- `add_which(Term)`

#### Logical and Boolean operators to use with `filter()`

- `<`
- `<=`
- `>`
- `>=`
- `!=`
- `==`

See `RStudio::Logic` and `RStudio::Comparison` for help.

#### Group Cases

Use `group_by()` to create a ‘grouped’ copy of a table. dplyr functions will manipulate each ‘group’ separately and then combine the results.

- `group_by()`
- `summarize()`
- `mutate()`
- `select()`
- `arrange()`

### Summary

- **summarize()**
- **select()**
- **group_by()**

### Variables

These apply `summarize()` functions to columns. Vectorized functions take vectors in `input` and return vectors of the same length as output.

See `RStudio::Reduce`
you %>%
  select(interesting_dataset) %>%
  summarize(features) %>%
  test(hypothesis) %>%
  profit() %>%
  the_end()
nycflights13 tables