dplyr
(and the tidyverse)
Matthew Flickinger, Ph.D.
CSG Tech Talk
University of Michigan
July 12, 2017
The Tidyverse
Tidyverse

- Very popular, widely used
- Prioritize data analysis rather than computer science
- Enable learners to become productive more quickly
- Encourages readable code

http://varianceexplained.org/r/teach-tidyverse/
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 0.7.1 (released June 22, 2017)
Sample data

- Examples use a data set containing all out-bound flights from NYC in 2013
- Available as an R package
  - `install.packages("nycflights13")`
  - `library(nycflights13)`
"flights" table

```r
> flights
Source: local data frame [336,776 x 19]

<table>
<thead>
<tr>
<th>year</th>
<th>month</th>
<th>day</th>
<th>dep_time</th>
<th>sched_dep_time</th>
<th>dep_delay</th>
<th>arr_time</th>
<th>sched_arr_time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2013</td>
<td>1</td>
<td>1</td>
<td>517</td>
<td>515</td>
<td>2</td>
<td>830</td>
<td>819</td>
</tr>
<tr>
<td>2013</td>
<td>1</td>
<td>1</td>
<td>533</td>
<td>529</td>
<td>4</td>
<td>850</td>
<td>830</td>
</tr>
<tr>
<td>2013</td>
<td>1</td>
<td>1</td>
<td>542</td>
<td>540</td>
<td>2</td>
<td>923</td>
<td>850</td>
</tr>
<tr>
<td>2013</td>
<td>1</td>
<td>1</td>
<td>544</td>
<td>545</td>
<td>-1</td>
<td>1004</td>
<td>1022</td>
</tr>
<tr>
<td>2013</td>
<td>1</td>
<td>1</td>
<td>554</td>
<td>600</td>
<td>-6</td>
<td>812</td>
<td>837</td>
</tr>
<tr>
<td>2013</td>
<td>1</td>
<td>1</td>
<td>554</td>
<td>558</td>
<td>-4</td>
<td>740</td>
<td>728</td>
</tr>
<tr>
<td>2013</td>
<td>1</td>
<td>1</td>
<td>555</td>
<td>600</td>
<td>-5</td>
<td>913</td>
<td>854</td>
</tr>
<tr>
<td>2013</td>
<td>1</td>
<td>1</td>
<td>557</td>
<td>600</td>
<td>-3</td>
<td>709</td>
<td>723</td>
</tr>
<tr>
<td>2013</td>
<td>1</td>
<td>1</td>
<td>557</td>
<td>600</td>
<td>-3</td>
<td>838</td>
<td>846</td>
</tr>
<tr>
<td>2013</td>
<td>1</td>
<td>1</td>
<td>558</td>
<td>600</td>
<td>-2</td>
<td>753</td>
<td>745</td>
</tr>
</tbody>
</table>

Variables not shown: arr_delay (dbl), carrier (chr), flight (int), tailnum (chr), origin (chr), dest (chr), air_time (dbl), distance (dbl), hour (dbl), minute (dbl), time_hour (time)
Basic single-table dplyr 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 parameter
- Returns a new data object, never updates/replace original
- Specify columns as unquoted strings (symbols)
Filtering Rows

- Find all flights to Detroit (DTW) in June (2013)
- dplyr
  - `filter(flights, dest=="DTW" & month==6)`
- Base R
  - `flights[flights$dest=="DTW" & flights$month==6, ]`
  - `subset(flights, dest=="DTW" & month==6)`
Selecting columns

- Select specific columns
  - `select(flights, dep_time, arr_time, carrier)`

- Exclude columns
  - `select(flights, -year, -tailnum)`

- Select column range
  - `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
Verb composition via magrittr

- What if we want to both filter and select?
- We could create variables for each intermediate step
  - `filtered <- filter(flights, dest="DTW")`
  - `select(filtered, carrier)`
- Or we could nest the calls
  - `select(filter(flights, dest="DTW"), carrier)`
- But these can get messy
Verb composition via magrittr

- The magrittr package introduces the "%>%" operator to "pipe" data into function
- Similar to the unix pipe operator
  - `grep apple fruit.txt | head -50 | cut -f3 | sort | uniq -c`
- The %>% operator passes the result from the left side to the first argument of the right side
  - `a(b(c(x))) ⇔ x %>% c() %>% b() %>% a()`
- `flights %>% filter(dest=="DTW") %>% select(carrier)`
Sort data

- Use `arrange()` to sort your rows
  - `flights %>% arrange(sched_dep_time)`
- Use `desc()` to reverse the sort order of a column
  - `flights %>% arrange(month, desc(day))`
- You can sort on functions of variables
  - `flights %>% ` arrange(desc(dep_time-sched_dep_time ))`
Create new variables

- Mutate allows you to create columns using existing values
  ```r
  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
  ```r
  new_flights <- flights %>% mutate(....)
  ```
Use new variables right away

- The parameters to mutate are processed in the order they appear
- `flights %>% mutate(`
  ```r
  dist_km = distance * 1.61,
  hours = air_time / 60,
  kph = dist_km/hours
  ```
- `select(flight, kph)`
- Be careful! You can overwrite existing variables
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()`
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
...
Combining group_by() with transformations

- **mutate()**
  - Will not change the number of rows
  - Functions like `max()` will return the max for each group

- **summarize()**
  - Returns one row per group
  - You must tell summarize how to collapse all non-grouped columns that you want
group_by() + mutate() example

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]

#   carrier arr_delay avg_arr_delay
#   <chr>    <dbl>       <dbl>
# 1   UA      11       3.558011
# 2   UA      20       3.558011
# 3   AA      33       0.3642909
# 4   B6     -18       9.4579733
# ...
Merging data
What are these carrier codes?

```r
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   DL         16.4634715
##  6   EV         23.8142303
##  7   G4          5.1399783
##  8   HA          4.5087446
##  9   HP          6.5875333
## 10   JI          6.3238431
## 11   JQ          3.7212222
## 12   MS         11.7215659
## 13   NW         14.9011742
## 14   P0          7.1258020
## 15   U1          1.1159569
## 16   US          8.9080787
...
"airlines" table

```r
> airlines
# A tibble: 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.          
...```

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"

# A tibble: 16 x 3

  carrier avg_arr_delay               name
  <chr>       <dbl>                       <chr>
1  9E         7.3796692        Endeavor Air Inc.
2  AA        0.3642909      American Airlines Inc.
3  AS       -9.9308886        Alaska Airlines Inc.
4  B6       9.4579733              JetBlue Airways
```
Types of joins (merges)

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

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>x1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>x2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>x3</td>
<td>4</td>
</tr>
</tbody>
</table>
Types of joins (merges)

inner_join(x, y)

left_join(x, y)

full_join(x, y)

right_join(x, y)

Table:

<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)

```
<table>
<thead>
<tr>
<th></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>
<tr>
<td>3</td>
<td>x3</td>
<td>NA</td>
</tr>
<tr>
<td>4</td>
<td>NA</td>
<td>y3</td>
</tr>
</tbody>
</table>
```
Non-merging joins

- These "joins" do not add any new columns to your data
- They useful for subsetting with multi-column matches
- semi_join()
  - Only keep rows in left table with matches in right
- anti_join()
  - Drop rows in left table with matches in right
Join on

- By default the join commands will join two tables based on all matching column names
- You can control the joining by specifying the column names
- `flights %>% inner_join(planes) %>% nrow`
- `flights %>% inner_join(planes, "tailnum") %>% nrow`
Other dplyr functions
Subsetting observations

- `distinct()`
  - Return unique rows
  - `flights %>% distinct(tailnum, carrier)`

- `count()`
  - Count rows with unique values of selected columns
  - `flights %>% count(carrier)`

- `sample_n()`
  - Randomly choose n rows
  - `flights %>% sample_n(3)`
Transformation functions

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

- `summarize_at()`/`mutate_at()`
  - Apply function to chosen columns
    - `flights %>% summarize_at(vars(ends_with("time")), mean, na.rm=T)`

- `summarize_if()`/`mutate_if()`
  - Apply function to matching columns
    - `flights %>% summarize_if(is.numeric, funs(mean, var), na.rm=T)`

- See `vars()` and `funs()` for passing multiple columns or functions
Other functions

- **lead()*/lag()**
  - Get value just after/before current value
  - x<-1:5; x; lead(x); lag(x)

- **coalesce()**
  - Returns first non-NA value from vectors
  - coalesce(c(NA,2,NA), c(1, NA, NA), 3)

- **n()*/n_distinct()**
  - Number of (distinct) values is a vector
  - Can only be used within summarize(), mutate(), filter()
  - flights %>% group_by(tailnum) %>%
    summarize(flights=n(), routes=n_distinct(flight))
Other functions

- recode()
  - Replace values in vector with different values
  ```r
  recode(letters[1:5], b="boo")
  ```

- case_when()
  - Alternative to nested ifelse() calls
    ```r
    x <- 1:50
    case_when(
      x %% 35 == 0 ~ "fizz buzz",
      x %% 5 == 0 ~ "fizz",
      x %% 7 == 0 ~ "buzz",
      TRUE ~ as.character(x)
    )
    ```
Combining data frames

- **bind_rows()**
  - Stack two data frames on top of each other (should have the same number of columns)

- **bind_columns()**
  - Place two data frames next to each other (should have the same number of rows) - no merge-able columns

- **intersect(), union(), setdiff()**
  - For rows shared or exclusive to data frames
Other benefits of dplyr
In all these examples we've been working with basic R data.frames.

However, dplyr can act as a front end to other data types.

This include databases (via SQL) and data.tables.
> class(flights)
[1] "tbl_df"     "tbl"        "data.frame"

- dplyr works with objects of class "tbl" (pronounced tibble)
- Here we have a wrapped data.frame
Data tables

- The "data.table" package exist to make data frame like structures that are faster and more efficient to work with.
- The "data.table" package overload the subset operator "[" to allow for grouping and subsetting in a non-standard way.
- If you load the "dplyr" package, you can use the nicer dplyr functions to work with data tables as well.
- (Prior to dplyr 0.5.0, the data tables functions were in the same package and didn't need to be loaded separately.)
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
Quosures

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)
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          7.3796692
2    AA          0.3642909
...
"Data Wrangling Cheat Sheet"
you %>%
  select(interesting_dataset) %>%
  summarize(features) %>%
  test(hypothesis) %>%
  profit() %>%
  the_end()
nycflights13 tables