Intro to dplyr and the tidyverse

Brian Barkley, UNC Biostatistics

September 28, 2016

Hello world

- Lecture slides for an introduction to the `dplyr` package and several other packages in the *tidyverse*
- Slides created for UNC-CH EPID 799B: Introduction to R (for Epidemiologists), Fall 2016
  - Instructor Alan Brookhart
  - TA's Xiaojuan Li and Nat MacNell
  - This lecture: Brian G. Barkley, UNC-CH Biostatistics

What are `dplyr` and the *tidyverse*?

Intro to `dplyr`

- `dplyr` is an R package designed to help you manipulate dataframes.
- A main goal: "Identify the most important data manipulation tools needed for data analysis and make them easy to use from R." (per The README for version 0.5.0)
- Main author is Dr. Hadley Wickham @hadleywickham

Intro to the *tidyverse*

- ~~Dr. Wickham~~ Hadley has also written a handful of other useful software packages (see his website)
- A focus: making data *tidy* and easy to use.
- A goal: make R more user-friendly as well as powerful.
- These packages often work well together and follow similar concepts
- These packages make up so-called *tidyverse*.
- These packages make my life better.

Packages

Getting started

Install the packages

```
install.packages('dplyr') ##for manipulating data frames
install.packages('tidyr') ##for `tidying` data from wide to long format
install.packages('magrittr') ##for making life SO MUCH EASIER
# install.packages('lubridate') ##for working with dates/times
install.packages('ggplot2') ##graphing
install.packages('stringr') ##character variables ('strings')
```

Load the packages

```
library(dplyr) ; library(tidyr) ; library(magrittr)
library(ggplot2); library(stringr); #library(lubridate)
```

A brief note (from Jeff Leek) on trustworthiness of packages: Bioconductor & CRAN $>$ GitHub $>$ other sources

Get data

We will use data from another package

```
install.packages('nycflights13') # get the data
```

Load the library to obtain a dataframe called `flights`.

```
library(nycflights13) #load the data in your environment
```

Brief look at the data

What kind of object is it?

```
class(flights)##What kind of object is it?
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

How big?

```
dim(flights) ##What are its dimensions?
```

```
## [1] 336776     19
```

What are the columns ('features' or 'variables')?

```
str(flights) ##What do we know about its columns/variables
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    336776 obs. of  19 variables:
##  $ year          : int  2013 2013 2013 2013 2013 2013 2013 2013 2013 2013 ...
##  $ month         : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ day           : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ dep_time      : int  517 533 542 544 554 554 555 557 557 558 ...
##  $ sched_dep_time: int  515 529 540 545 600 558 600 600 600 600 ...
##  $ dep_delay     : num  2 4 2 -1 -6 -4 -5 -3 -3 -2 ...
##  $ arr_time      : int  830 850 923 1004 812 740 913 709 838 753 ...
##  $ sched_arr_time: int  819 830 850 1022 837 728 854 723 846 745 ...
##  $ arr_delay     : num  11 20 33 -18 -25 12 19 -14 -8 8 ...
##  $ carrier       : chr  "UA" "UA" "AA" "B6" ...
##  $ flight        : int  1545 1714 1141 725 461 1696 507 5708 79 301 ...
##  $ tailnum       : chr  "N14228" "N24211" "N619AA" "N804JB" ...
##  $ origin        : chr  "EWR" "LGA" "JFK" "JFK" ...
##  $ dest          : chr  "IAH" "IAH" "MIA" "BQN" ...
##  $ air_time      : num  227 227 160 183 116 150 158 53 140 138 ...
##  $ distance      : num  1400 1416 1089 1576 762 ...
```

Show a couple rows

```
head(flights) ##Print a couple
```

```
## Source: local data frame [6 x 19]
##
##    year month   day dep_time sched_dep_time dep_delay arr_time
##   <int> <int> <int>    <int>          <int>     <dbl>    <int>
## 1  2013     1     1      517            515         2      830
## 2  2013     1     1      533            529         4      850
## 3  2013     1     1      542            540         2      923
## 4  2013     1     1      544            545        -1     1004
## 5  2013     1     1      554            600        -6      812
## 6  2013     1     1      554            558        -4      740
## Variables not shown: sched_arr_time <int>, 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>.
```

Printing: very nice!

```
print(flights)
```

```
## Source: local data frame [336,776 x 19]
##
##     year month   day dep_time sched_dep_time dep_delay arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>
## 1   2013     1     1      517            515         2      830
## 2   2013     1     1      533            529         4      850
## 3   2013     1     1      542            540         2      923
## 4   2013     1     1      544            545        -1     1004
## 5   2013     1     1      554            600        -6      812
## 6   2013     1     1      554            558        -4      740
## 7   2013     1     1      555            600        -5      913
## 8   2013     1     1      557            600        -3      709
## 9   2013     1     1      557            600        -3      838
## 10  2013     1     1      558            600        -2      753
## ..   ...   ...   ...      ...            ...       ...      ...
## Variables not shown: sched_arr_time <int>, arr_delay <dbl>, carrier <chr>,
```

Briefly on `data.frames` & `tibbles`

- In `R`, the `data.frame` structure can hold `vectors`/`lists` of the same length - one in each column.
  - So a `data.frame` can hold both numbers and letters in separate columns
  - Objects of `matrix` class are unable to handle both types
- Essentially, a `tibble` or "local" dfm is just a `data.frame`
  - It is a back-end built by Hadley et al
  - it plays a little more nicely than `data.frame()`
- For more on object classes in R, see
  - Advanced R book by Hadley, or
  - (beginner/intermediate) R Programming for Data Science by Roger Peng

Action verbs in `dplyr`

- For looking at the raw data
  - `select()`: focus on a subset of variables
  - `filter()`: focus on a subset of rows
  - `arrange()`: re-order the rows
- For (re)calculating variables
  - `mutate()`: add new columns
- For calculate statistics
  - `summarise()`: reduce each group to a smaller number of summary statistics

(per the README for version 0.5.0)

Helpful/necessary for calculating statistics:
`group_by()`, `un_group()`, `rowwise()`

`select()` to see certain cols/vars

Select certain columns by writing their name or column number

```
select(flights, carrier)
```

`select()` to see certain cols/vars

Select certain columns by writing their name or column number

```
select(flights, carrier) ## compare to: flights[,"carrier"]
```

```
## Source: local data frame [336,776 x 1]
##
##    carrier
##      <chr>
## 1       UA
## 2       UA
## 3       AA
## 4       B6
## 5       DL
## 6       UA
## 7       B6
## 8       EV
## 9       B6
## 10      AA
## ..     ...
```

`select()` to see certain cols/vars

… or by the column number

```
select(flights, 10) ## compare to: flights[,10]
```

`select()` to see certain cols/vars

… or by the column number

```
select(flights, 10) ## compare to: flights[,10]
```

```
## Source: local data frame [336,776 x 1]
##
##    carrier
##      <chr>
## 1       UA
## 2       UA
## 3       AA
## 4       B6
## 5       DL
## 6       UA
## 7       B6
## 8       EV
## 9       B6
## 10      AA
## ..     ...
```

Select multiple columns

NB: pattern is *function(data, action[s])*

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

Select multiple columns

NB: pattern is *function(data, action[s])*

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

```
## Source: local data frame [336,776 x 2]
##
##    carrier dep_time
##      <chr>    <int>
## 1       UA      517
## 2       UA      533
## 3       AA      542
## 4       B6      544
## 5       DL      554
## 6       UA      554
## 7       B6      555
## 8       EV      557
## 9       B6      557
## 10      AA      558
## ..     ...      ...
```

Select multiple columns - flexibly

```
select(flights, 10,air_time,11, dep_time:arr_delay, 1:3)
```

Select multiple columns - flexibly

```
select(flights, 10,air_time,11, dep_time:arr_delay, 1:3) #fancy!
```

```
## Source: local data frame [336,776 x 12]
##
##    carrier air_time flight dep_time sched_dep_time dep_delay arr_time
##      <chr>    <dbl>  <int>    <int>          <int>     <dbl>    <int>
## 1       UA      227   1545      517            515         2      830
## 2       UA      227   1714      533            529         4      850
## 3       AA      160   1141      542            540         2      923
## 4       B6      183    725      544            545        -1     1004
## 5       DL      116    461      554            600        -6      812
## 6       UA      150   1696      554            558        -4      740
## 7       B6      158    507      555            600        -5      913
## 8       EV       53   5708      557            600        -3      709
## 9       B6      140     79      557            600        -3      838
## 10      AA      138    301      558            600        -2      753
## ..     ...      ...    ...      ...            ...       ...      ...
## Variables not shown: sched_arr_time <int>, arr_delay <dbl>, year <int>,
##   month <int>, day <int>.
```

Select multiple columns - with helper functions!

```
select(flights, starts_with('dep'))
```

Select multiple columns - with helper functions!

```
select(flights, starts_with('dep')) #only columns "dep*"
```

```
## Source: local data frame [336,776 x 2]
##
##    dep_time dep_delay
##       <int>     <dbl>
## 1       517         2
## 2       533         4
## 3       542         2
## 4       544        -1
## 5       554        -6
## 6       554        -4
## 7       555        -5
## 8       557        -3
## 9       557        -3
## 10      558        -2
## ..      ...       ...
```

more helper functions!

```
select(flights, contains('dep'))
```

more helper functions!

```
select(flights, contains('dep')) # columns with *dep*
```

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

Exclude with the negative sign (think of keep/drop in SAS)

```
select(flights, -contains('dep'))
```

Exclude with the negative sign (think of keep/drop in SAS)

```
select(flights, -contains('dep')) # everything but *dep*
```

```
## Source: local data frame [336,776 x 16]
##
##     year month   day arr_time sched_arr_time arr_delay carrier flight
##    <int> <int> <int>    <int>          <int>     <dbl>   <chr>  <int>
## 1   2013     1     1      830            819        11      UA   1545
## 2   2013     1     1      850            830        20      UA   1714
## 3   2013     1     1      923            850        33      AA   1141
## 4   2013     1     1     1004           1022       -18      B6    725
## 5   2013     1     1      812            837       -25      DL    461
## 6   2013     1     1      740            728        12      UA   1696
## 7   2013     1     1      913            854        19      B6    507
## 8   2013     1     1      709            723       -14      EV   5708
## 9   2013     1     1      838            846        -8      B6     79
## 10  2013     1     1      753            745         8      AA    301
## ..   ...   ...   ...      ...            ...       ...     ...    ...
## Variables not shown: tailnum <chr>, origin <chr>, dest <chr>, air_time
##   <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <time>.
```

reorder with `everything()`

```
select(flights, contains('dep'), everything())
```

reorder with `everything()`

```
select(flights, contains('dep'), everything())
```

```
## Source: local data frame [336,776 x 19]
##
##    dep_time sched_dep_time dep_delay  year month   day arr_time
##       <int>          <int>     <dbl> <int> <int> <int>    <int>
## 1       517            515         2  2013     1     1      830
## 2       533            529         4  2013     1     1      850
## 3       542            540         2  2013     1     1      923
## 4       544            545        -1  2013     1     1     1004
## 5       554            600        -6  2013     1     1      812
## 6       554            558        -4  2013     1     1      740
## 7       555            600        -5  2013     1     1      913
## 8       557            600        -3  2013     1     1      709
## 9       557            600        -3  2013     1     1      838
## 10      558            600        -2  2013     1     1      753
## ..      ...            ...       ...   ...   ...   ...      ...
## Variables not shown: sched_arr_time <int>, arr_delay <dbl>, carrier <chr>,
```

`filter()` to see certain obs/indivs

Use logical operations to reduce number of rows.

```
flights_v2 <- select(flights,carrier,everything())  #re-order columns
filter(flights_v2, carrier=="UA")
```

NB: Pattern is *function(data, action[s])*

`filter()` to see certain obs/indivs

Use logical operations to reduce number of rows.

```
flights_v2 <- select(flights,carrier,everything()) #re-order columns
filter(flights_v2, carrier=="UA") #United Airlines
```

```
## Source: local data frame [58,665 x 19]
##
##    carrier  year month   day dep_time sched_dep_time dep_delay arr_time
##      <chr> <int> <int> <int>    <int>          <int>     <dbl>    <int>
## 1       UA  2013     1     1      517            515         2      830
## 2       UA  2013     1     1      533            529         4      850
## 3       UA  2013     1     1      554            558        -4      740
## 4       UA  2013     1     1      558            600        -2      924
## 5       UA  2013     1     1      558            600        -2      923
## 6       UA  2013     1     1      559            600        -1      854
## 7       UA  2013     1     1      607            607         0      858
## 8       UA  2013     1     1      611            600        11      945
## 9       UA  2013     1     1      623            627        -4      933
## 10      UA  2013     1     1      628            630        -2     1016
```

Logic is flexible

```
filter(flights_v2, carrier %in% c("DL","UA"))
```

Logic is flexible

```
filter(flights_v2, carrier %in% c("DL","UA")) #United or Delta
```

```
## Source: local data frame [106,775 x 19]
##
##    carrier  year month   day dep_time sched_dep_time dep_delay arr_time
##      <chr> <int> <int> <int>    <int>          <int>     <dbl>    <int>
## 1       UA  2013     1     1      517            515         2      830
## 2       UA  2013     1     1      533            529         4      850
## 3       DL  2013     1     1      554            600        -6      812
## 4       UA  2013     1     1      554            558        -4      740
## 5       UA  2013     1     1      558            600        -2      924
## 6       UA  2013     1     1      558            600        -2      923
## 7       UA  2013     1     1      559            600        -1      854
## 8       DL  2013     1     1      602            610        -8      812
## 9       DL  2013     1     1      606            610        -4      837
## 10      UA  2013     1     1      607            607         0      858
## ..     ...   ...   ...   ...      ...            ...       ...      ...
## Variables not shown: sched_arr_time <int>, arr_delay <dbl>, flight <int>,
```

Logic is flexible

```
filter(flights_v2, dep_time < 600)
```

Logic is flexible

```
filter(flights_v2, dep_time < 600) #early morning flights
```

```
## Source: local data frame [8,730 x 19]
##
##    carrier  year month   day dep_time sched_dep_time dep_delay arr_time
##      <chr> <int> <int> <int>    <int>          <int>     <dbl>    <int>
## 1       UA  2013     1     1      517            515         2      830
## 2       UA  2013     1     1      533            529         4      850
## 3       AA  2013     1     1      542            540         2      923
## 4       B6  2013     1     1      544            545        -1     1004
## 5       DL  2013     1     1      554            600        -6      812
## 6       UA  2013     1     1      554            558        -4      740
## 7       B6  2013     1     1      555            600        -5      913
## 8       EV  2013     1     1      557            600        -3      709
## 9       B6  2013     1     1      557            600        -3      838
## 10      AA  2013     1     1      558            600        -2      753
## ..     ...   ...   ...   ...      ...            ...       ...      ...
## Variables not shown: sched_arr_time <int>, arr_delay <dbl>, flight <int>,
```

Logic is flexible

```
filter(flights_v2, carrier!="UA")
```

Logic is flexible

```
filter(flights_v2, carrier!="UA") #who even likes United??
```

```
## Source: local data frame [278,111 x 19]
##
##    carrier  year month   day dep_time sched_dep_time dep_delay arr_time
##      <chr> <int> <int> <int>    <int>          <int>     <dbl>    <int>
## 1       AA  2013     1     1      542            540         2      923
## 2       B6  2013     1     1      544            545        -1     1004
## 3       DL  2013     1     1      554            600        -6      812
## 4       B6  2013     1     1      555            600        -5      913
## 5       EV  2013     1     1      557            600        -3      709
## 6       B6  2013     1     1      557            600        -3      838
## 7       AA  2013     1     1      558            600        -2      753
## 8       B6  2013     1     1      558            600        -2      849
## 9       B6  2013     1     1      558            600        -2      853
## 10      AA  2013     1     1      559            600        -1      941
## ..     ...   ...   ...   ...      ...            ...       ...      ...
## Variables not shown: sched_arr_time <int>, arr_delay <dbl>, flight <int>,
```

Combine multiple logical statements for sub-subsets

```
filter(flights_v2, dep_time < 600, carrier == "UA")
```

Combine multiple logical statements for sub-subsets

```
filter(flights_v2, dep_time < 600, carrier == "UA") #early United flights
```

```
## Source: local data frame [1,668 x 19]
##
##    carrier  year month   day dep_time sched_dep_time dep_delay arr_time
##      <chr> <int> <int> <int>    <int>          <int>     <dbl>    <int>
## 1       UA  2013     1     1      517            515         2      830
## 2       UA  2013     1     1      533            529         4      850
## 3       UA  2013     1     1      554            558        -4      740
## 4       UA  2013     1     1      558            600        -2      924
## 5       UA  2013     1     1      558            600        -2      923
## 6       UA  2013     1     1      559            600        -1      854
## 7       UA  2013     1     2      512            515        -3      809
## 8       UA  2013     1     2      536            529         7      840
## 9       UA  2013     1     2      558            600        -2      916
## 10      UA  2013     1     2      559            601        -2      809
## ..     ...   ...   ...   ...      ...            ...       ...      ...
## Variables not shown: sched_arr_time <int>, arr_delay <dbl>, flight <int>,
```

Logic can be complex and involve multiple columns

```
flights_v3 <- filter(flights_v2, arr_delay > air_time)
```

Logic can be complex and involve multiple columns

```
flights_v3 <- filter(flights_v2, arr_delay > air_time) #the worst flights
flights_v4 <- select(flights_v3, carrier, arr_delay,air_time, origin,
    dest, dep_delay) #re-ordering columns
flights_v4
```

```
## Source: local data frame [14,009 x 6]
##
##    carrier arr_delay air_time origin  dest dep_delay
##      <chr>     <dbl>    <dbl>  <chr> <chr>     <dbl>
## 1       MQ       137      118    LGA   CLT       101
## 2       MQ       851       41    JFK   BWI       853
## 3       UA       123       37    EWR   BOS       144
## 4       EV       127       63    EWR   RIC       115
## 5       EV       123       53    JFK   IAD       119
## 6       UA        78       35    EWR   BOS        84
## 7       EV        59       53    EWR   IAD        64
## 8       EV        40       36    EWR   ALB        34
## 9       EV       123      119    EWR   DAY        62
```

`arrange()` to sort obs/indivs in order

Sort by one variable

```
arrange(flights_v4, air_time)
```

NB: Pattern is *function(data, action[s])*

`arrange()` to sort obs/indivs in order

Sort by one variable

```
arrange(flights_v4, air_time) #sorted by shortest air time
```

```
## Source: local data frame [14,009 x 6]
##
##    carrier arr_delay air_time origin  dest dep_delay
##      <chr>     <dbl>    <dbl>  <chr> <chr>     <dbl>
## 1       EV        31       20    EWR   BDL        40
## 2       EV        27       21    EWR   BDL        31
## 3       EV        23       21    EWR   PHL        24
## 4       9E        35       21    JFK   PHL        51
## 5       EV        67       21    EWR   BDL        87
## 6       EV       109       21    EWR   BDL       137
## 7       EV       115       21    EWR   BDL       136
## 8       EV       102       21    EWR   BDL       129
## 9       EV       166       22    EWR   BDL       190
## 10      EV        65       22    EWR   BDL        86
## ..     ...       ...      ...    ...   ...       ...
```

What do you think this does?

```
arrange(flights_v4,  desc(air_time))
```

What do you think this does? Use `desc` to reverse-sort

```
arrange(flights_v4,  desc(air_time)) #sorted by LONGEST air time
```

```
## Source: local data frame [14,009 x 6]
##
##    carrier arr_delay air_time origin  dest dep_delay
##      <chr>     <dbl>    <dbl>  <chr> <chr>     <dbl>
## 1       HA      1272      640    JFK   HNL      1301
## 2       UA       422      395    EWR   SFO       374
## 3       AA       380      361    JFK   SFO       364
## 4       UA       399      360    JFK   SFO       325
## 5       UA       373      359    JFK   LAX       364
## 6       VX       360      354    JFK   SFO       304
## 7       AA      1007      354    JFK   SFO      1014
## 8       VX       354      347    JFK   SFO       364
## 9       AA       368      346    JFK   SFO       337
## 10      B6       445      345    JFK   SFO       453
## ..     ...       ...      ...    ...   ...       ...
```

What about strings?

```
arrange(flights_v4, dest)
```

What about strings?

```
arrange(flights_v4, dest) #sorted by destination ABC'ly
```

```
## Source: local data frame [14,009 x 6]
##
##    carrier arr_delay air_time origin  dest dep_delay
##      <chr>     <dbl>    <dbl>  <chr> <chr>     <dbl>
## 1       B6        49       42    JFK   ACK        60
## 2       B6        55       45    JFK   ACK        41
## 3       B6        92       37    JFK   ACK       101
## 4       B6       150       99    JFK   ACK        71
## 5       B6        54       38    JFK   ACK        55
## 6       B6       122       48    JFK   ACK       100
## 7       B6        79       45    JFK   ACK        85
## 8       B6        81       52    JFK   ACK        73
## 9       B6        75       42    JFK   ACK        53
## 10      B6       130       41    JFK   ACK       138
## ..     ...       ...      ...    ...   ...       ...
```

You can sort by multiple variables

```
#
arrange(flights_v4, air_time, dest,arr_delay,carrier)
```

You can sort by multiple variables

```
#Sorted first by duration, then destination, then tardiness, then airline
arrange(flights_v4, air_time, dest,arr_delay,carrier)
```

```
## Source: local data frame [14,009 x 6]
##
##    carrier arr_delay air_time origin  dest dep_delay
##      <chr>     <dbl>    <dbl>  <chr> <chr>     <dbl>
## 1       EV        31       20    EWR   BDL        40
## 2       EV        27       21    EWR   BDL        31
## 3       EV        67       21    EWR   BDL        87
## 4       EV       102       21    EWR   BDL       129
## 5       EV       109       21    EWR   BDL       137
## 6       EV       115       21    EWR   BDL       136
## 7       EV        23       21    EWR   PHL        24
## 8       9E        35       21    JFK   PHL        51
## 9       EV        50       22    EWR   BDL        42
## 10      EV        65       22    EWR   BDL        86
## ..     ...       ...      ...    ...   ...       ...
```

Now you know how to look at data

`select()`, `filter()` and `arrange()` allow you to view the raw data in ways that make sense for you.

Now, how to manipulate the data to get the variables you need.

enter `mutate()`

`mutate()` to make calculations

You can make a new column/variable easily.

```
mutate(flights_v4, my_factor = arr_delay/air_time)
```

NB: Pattern is *function(data, action[s])*

`mutate()` to make calculations

You can make a new column/variable easily.

```
mutate(flights_v4, my_factor = arr_delay/air_time)
```

```
## Source: local data frame [14,009 x 7]
##
##    carrier arr_delay air_time origin  dest dep_delay my_factor
##      <chr>     <dbl>    <dbl>  <chr> <chr>     <dbl>     <dbl>
## 1       MQ       137      118    LGA   CLT       101  1.161017
## 2       MQ       851       41    JFK   BWI       853 20.756098
## 3       UA       123       37    EWR   BOS       144  3.324324
## 4       EV       127       63    EWR   RIC       115  2.015873
## 5       EV       123       53    JFK   IAD       119  2.320755
## 6       UA        78       35    EWR   BOS        84  2.228571
## 7       EV        59       53    EWR   IAD        64  1.113208
## 8       EV        40       36    EWR   ALB        34  1.111111
## 9       EV       123      119    EWR   DAY        62  1.033613
## 10      EV        51       45    EWR   DCA        54  1.133333
## ..     ...       ...      ...    ...   ...       ...       ...
```

You can create multiple variables in one step

```
mutate(flights_v4,
         my_factor = arr_delay/air_time,
         how_bad = ifelse(my_factor<3, "bad", "AWFUL") )
```

You can create multiple variables in one step

```
mutate(flights_v4,
         my_factor = arr_delay/air_time,
         how_bad = ifelse(my_factor<3, "bad", "AWFUL") )
```

```
## Source: local data frame [14,009 x 8]
##
##    carrier arr_delay air_time origin  dest dep_delay my_factor how_bad
##      <chr>     <dbl>    <dbl>  <chr> <chr>     <dbl>     <dbl>   <chr>
## 1       MQ       137      118    LGA   CLT       101  1.161017     bad
## 2       MQ       851       41    JFK   BWI       853 20.756098   AWFUL
## 3       UA       123       37    EWR   BOS       144  3.324324   AWFUL
## 4       EV       127       63    EWR   RIC       115  2.015873     bad
## 5       EV       123       53    JFK   IAD       119  2.320755     bad
## 6       UA        78       35    EWR   BOS        84  2.228571     bad
## 7       EV        59       53    EWR   IAD        64  1.113208     bad
## 8       EV        40       36    EWR   ALB        34  1.111111     bad
## 9       EV       123      119    EWR   DAY        62  1.033613     bad
## 10      EV        51       45    EWR   DCA        54  1.133333     bad
## ..     ...       ...      ...    ...   ...       ...       ...     ...
```

What if your name your variable after a column that already exists?

```
mutate(flights_v4,
        my_factor = arr_delay/air_time,
        dep_delay = "too long", ##already exists
        dest = ifelse(dest=="ALB", "Not Albany!!",dest) ) #already exists
```

be careful…

What if your name your variable after a column that already exists? It will **overwrite the original variable**!

```
mutate(flights_v4,
        my_factor = arr_delay/air_time,
        dep_delay = "too long", ##OVERWRITTEN
        dest = ifelse(dest=="ALB", "Not Albany!!",dest) ) #edited
```

```
## Source: local data frame [14,009 x 7]
##
##    carrier arr_delay air_time origin         dest dep_delay my_factor
##      <chr>     <dbl>    <dbl>  <chr>        <chr>     <chr>     <dbl>
## 1       MQ       137      118    LGA          CLT  too long  1.161017
## 2       MQ       851       41    JFK          BWI  too long 20.756098
## 3       UA       123       37    EWR          BOS  too long  3.324324
## 4       EV       127       63    EWR          RIC  too long  2.015873
## 5       EV       123       53    JFK          IAD  too long  2.320755
## 6       UA        78       35    EWR          BOS  too long  2.228571
## 7       EV        59       53    EWR          IAD  too long  1.113208
## 8       EV        40       36    EWR Not Albany!!  too long  1.111111
## 9       EV       123      119    EWR          DAY  too long  1.033613
```

`summarise()` statistics!

Tell me about the air times

```
summarize(flights_v4, avg_at = mean(air_time))
```

NB: pattern is *function(data, action[s])*

`summarise()` statistics!

Tell me about the air times

```
summarize(flights_v4, avg_at = mean(air_time))
```

```
## Source: local data frame [1 x 1]
##
##     avg_at
##      <dbl>
## 1 78.36312
```

NB: pattern is *function(data, action[s])*

Multiple summaries at the same time?

(and do you *need* to name them?)

```
summarize(flights_v4, avg_at = mean(air_time), median(air_time))
```

Multiple summaries at the same time!

(and do you *need* to name them?) (Nope!)

```
summarize(flights_v4, avg_at = mean(air_time), median(air_time))
```

```
## Source: local data frame [1 x 2]
##
##     avg_at median(air_time)
##      <dbl>            <dbl>
## 1 78.36312               66
```

Can we make table 1?

```
flights_v5 <- mutate(flights_v4, short_flight = ifelse(air_time<60,1,0))
summarize(flights_v5, short_flight_pct =mean(short_flight))
```

```
## Source: local data frame [1 x 1]
##
##   short_flight_pct
##              <dbl>
## 1        0.4542794
```

We still need one more tool…

group_by for better statistics!

What does this do?

```
group_by(flights, carrier)
```

NB: pattern is *function(data, action[s])*

group_by for better statistics!

What does this do?

```
group_by(flights, carrier)
```

```
## Source: local data frame [336,776 x 19]
## Groups: carrier [16]
##
##     year month   day dep_time sched_dep_time dep_delay arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>
## 1   2013     1     1      517            515         2      830
## 2   2013     1     1      533            529         4      850
## 3   2013     1     1      542            540         2      923
## 4   2013     1     1      544            545        -1     1004
## 5   2013     1     1      554            600        -6      812
## 6   2013     1     1      554            558        -4      740
## 7   2013     1     1      555            600        -5      913
## 8   2013     1     1      557            600        -3      709
## 9   2013     1     1      557            600        -3      838
## 10  2013     1     1      558            600        -2      753
```

group_by for better statistics!

I'm tired of marginal stats. What does this do?

```
grouped_flights <- group_by(flights, carrier) #grouping
summarize(grouped_flights, avg_arr_delay = mean(na.omit(arr_delay)))
```

NB: pattern is *function(data, action[s])*

group_by for better statistics!

This calculates stats for each group variable!

```
grouped_flights <- group_by(flights, carrier) #grouping
summarize(grouped_flights, avg_arr_delay = mean(na.omit(arr_delay)))
```

```
## Source: local data frame [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     1.6443409
## 6       EV    15.7964311
## 7       F9    21.9207048
## 8       FL    20.1159055
## 9       HA    -6.9152047
## 10      MQ    10.7747334
```

Table 1

```
grouped_flights <- group_by(flights, origin) #grouping
table1 <- summarize(grouped_flights, avg_arr_delay = mean(na.omit(arr_delay)),
          num_departs = n())
table1
```

```
## Source: local data frame [3 x 3]
##
##   origin avg_arr_delay num_departs
##    <chr>         <dbl>       <int>
## 1    EWR      9.107055      120835
## 2    JFK      5.551481      111279
## 3    LGA      5.783488      104662
```

Table 1, improved

```
mutate(table1,tot_departs = sum(num_departs),
       pct_departs = num_departs/tot_departs)
```

```
## Source: local data frame [3 x 5]
##
##   origin avg_arr_delay num_departs tot_departs pct_departs
##    <chr>         <dbl>       <int>       <int>       <dbl>
## 1    EWR      9.107055      120835      336776   0.3587993
## 2    JFK      5.551481      111279      336776   0.3304244
## 3    LGA      5.783488      104662      336776   0.3107763
```

What days are the worst to fly? (longest arrival delay)

What days are the worst to fly? (longest arrival delay)

```
grouped_flights2 <- group_by(flights, month, day) #group by unique day/month
summ2 <- summarize(grouped_flights2,  #Get stats for each day/month combo
                   avg_arr_delay = mean(na.omit(arr_delay)))
arrange(summ2, desc(avg_arr_delay)) ##show the worst days
```

```
## Source: local data frame [365 x 3]
## Groups: month [12]
##
##    month   day avg_arr_delay
##    <int> <int>         <dbl>
## 1      3     8      85.86216
## 2      6    13      63.75369
## 3      7    22      62.76340
## 4      5    23      61.97090
## 5      7    10      59.62648
## 6      9    12      58.91242
## 7      7     1      58.28050
## 8     12    17      55.87186
## 9      8     8      55.48116
```

Two-table verbs (*joining*)

```
my_data <- dplyr::data_frame( #creating data frame
  person = c("Brian", "Xiaojuan", "Nat", "Alan"),
  origin = "EWR", #this will be smartly copied 4 times
  dest = c("MKE", "DCA", "ORD", "LAX"),
  year = 2013, month = 4, day =5, #each is smartly copied 4 times
  leaving_at = c(2046,  1725, 1150, 638)
)
my_data
```

```
## Source: local data frame [4 x 7]
##
##     person origin  dest  year month   day leaving_at
##      <chr>  <chr> <chr> <dbl> <dbl> <dbl>      <dbl>
## 1    Brian    EWR   MKE  2013     4     5       2046
## 2 Xiaojuan    EWR   DCA  2013     4     5       1725
## 3      Nat    EWR   ORD  2013     4     5       1150
## 4     Alan    EWR   LAX  2013     4     5        638
```

Two-table verbs (*joining*)

Let's say you want to link two data sources. There's `left_join()`, `right_join()`, `full_join()`, `inner_join()`, etc. Look up which one suits your purpose. See Jenny Bryan's cheatsheet

Each row in `my_data` links to exactly one row in `flights`. Let's use `left_join()` to see what our travels were like…

NB: pattern is *function(data1, data2, action[s])*

```
joined_data <- dplyr::left_join( #info FROM 'y' ONTO rows of 'x'
  x = my_data, ##'x' is left side dfm- we want to keep these rows
  y = flights, ##bringing info from flights (right) to my_data (left)
  by = c("origin", "dest", "year", "month", "day",
         #syntax for columns with the same name in x and y
         "leaving_at"="sched_dep_time"))
         #use this syntax if columns have different names
select(joined_data,person, arr_delay, dest, leaving_at)
```

```
## Source: local data frame [4 x 4]
##
##     person arr_delay  dest leaving_at
##      <chr>     <dbl> <chr>      <dbl>
## 1    Brian         5   MKE       2046
## 2 Xiaojuan        -1   DCA       1725
## 3      Nat        -2   ORD       1150
## 4     Alan       -16   LAX        638
```

Now: `person,leaving_at` from `x`, & `arr_delay,dest` from `y`!

Chaining workflow

Analysis question: What is the average arrival delay for each airline on the flights from Newark to O'Hare?

Analysis question: What is the average arrival delay for each airline on the flights from Newark to O'Hare?

Compare the pseudo-code to the following workflows:

1. Filter down to only flights from Newark to O'Hare, then
2. Group by the carrier, then
3. Summarize by average arrival delay

Analysis question: What is the average arrival delay for each airline on the flights from Newark to O'Hare?

Compare the pseudo-code to the following workflows:

1. Filter down to only flights from Newark to O'Hare, then
2. Group by the carrier, then
3. Summarize by average arrival delay

Attempt 1

```
filtering_flights <- filter(flights, origin=="EWR", dest == "ORD")
grouping_flights <- group_by(filtering_flights, carrier)
summarize(grouping_flights, avg_arr_delay = mean(na.omit(arr_delay)))
```

So many intermediate objects! What if you have to change one? What about processing speed? Potential for many issues here.

Analysis question: What is the average arrival delay for each airline on the flights from Newark to O'Hare?

Compare the pseudo-code to the following workflows:

1. Filter down to only flights from Newark to O'Hare, then
2. Group by the carrier, then
3. Summarize by average arrival delay

Attempt 2

```
summarize(group_by(filter(flights, origin=="EWR", dest=="ORD"),
  carrier), avg_arr_delay = mean(na.omit(arr_delay)))
```

This is just ugly! Easy to get lost.

Analysis question: What is the average arrival delay for each airline on the flights from Newark to O'Hare?

Compare the pseudo-code to the following workflows:

1. Filter down to only flights from Newark to O'Hare, then
2. Group by the carrier, then
3. Summarize by average arrival delay

Wouldn't it be great if the code looked like the pseudo-code?

Workflow chaining with `magrittr`

basically the best thing ever

The `magrittr` package offers a function called the pipe, `%>%`

```
flights %>%                            #Using the data.frame "flights", THEN
  filter(origin=="EWR", dest == "ORD") %>%     #1. only for EWR-ORD, THEN
  group_by(carrier) %>%                       #2. group by carrier, THEN
  summarize(avg_arr_delay = mean(na.omit(arr_delay)))  #3. summarize
```

```
## Source: local data frame [3 x 2]
##
##   carrier avg_arr_delay
##     <chr>         <dbl>
## 1      EV     17.500000
## 2      MQ     16.307105
## 3      UA      4.882006
```

Workflow chaining with `magrittr`

basically the best thing ever

The `magrittr` package offers a function called the pipe, `%>%`

```
flights %>%                            #Using the data.frame "flights", THEN
  filter(origin=="EWR", dest == "ORD") %>%     #1. only for EWR-ORD, THEN
  group_by(carrier) %>%                       #2. group by carrier, THEN
  summarize(avg_arr_delay = mean(na.omit(arr_delay)))  #3. summarize
```

NB: Each function acts on the data that's piped into it

1. input to `filter()` is the entire dataframe
2. input to `group_by()` is the filtered dataframe
3. input to `summarize()` it the filtered, grouped dataframe

The pipe is smart

NB: Each function acts on the data that's piped into it

Usually, pattern is `function(data, action[s])`

When chaining/piping with `magrittr`, pattern is usually `function(D, action[s])` where `D` is the output from the previous line that is piped (`%>%`) to the function.

When pattern is `function(x, y, action[s])`, then `x` will be piped in, *unless otherwise specified.*

The pipe is often very smart

Here we specify that `x = my_data` so the next argument (`y`) will take the piped object (`flights`) as input.

```
flights %>% ##We want this to be data2 (second argument is y)
  left_join( #joins info FROM 2nd dataframe ONTO the rows of 1st
  x = my_data, ##this is data1 (first argument is x)
  #y = flights, ##This is piped in
  by = c("origin", "dest", "year", "month", "day",
         "leaving_at"="sched_dep_time")) %>%
  #output from the left_join() function is piped to select() function
select(person, arr_delay, dest, leaving_at)
```

```
## Source: local data frame [4 x 4]
##
##     person arr_delay  dest leaving_at
##      <chr>     <dbl> <chr>      <dbl>
## 1    Brian         5   MKE       2046
## 2 Xiaojuan        -1   DCA       1725
```

Incremental changes -> big progress

My opinion is that when I break down each step of your data management into small, easy-to-understand chunks, it is easier for me to understand what I'm doing and what I still need to do.

```
my_stats <- flights %>% filter(origin=="EWR", dest == "ORD") %>%
  group_by(carrier) %>% summarize(avg_arr_delay = mean(na.omit(arr_delay)))
#Saving this data to my_stats, but now I want to do further operations
my_stats %>% arrange(avg_arr_delay) %>% filter(carrier !="UA")
```

```
## Source: local data frame [2 x 2]
##
##   carrier avg_arr_delay
##     <chr>         <dbl>
## 1      MQ      16.30711
## 2      EV      17.50000
```

Helpful trick: use the `knitr` package and the function `kable()` to make nicely formatted tables.

```
my_stats %>% arrange(avg_arr_delay) %>% filter(carrier !="UA") %>%
  knitr::kable(digits = 3,
               caption = "delays for most carriers from EWR to ORD")
```

delays for most carriers from EWR to ORD

carrier	avg_arr_delay
MQ	16.307
EV	17.500

NB: The first argument of `kable()` is the dataframe, which is piped in from our previous work. You can pipe objects into many functions, even those not written by Hadley!

Remember `flights_v4`?

```
flights_v4_tidy <- flights %>% #going to follow the steps we took earlier
  select(carrier, everything()) %>% #v2: reorder columns
  filter(arr_delay>air_time) %>% #v3: filter to fewer observations
  select(carrier, arr_delay, air_time, origin, dest, dep_delay) ##cut cols
```

What happens if you had forgotten a step?

```
flights_v4_tidy <- flights %>%
  select(carrier, everything()) %>%
  filter(arr_delay>air_time) %>%
# I want to calculate a new variable
# And I want to filter by it
# I might even want to calculate a second variable
  select(carrier, arr_delay, air_time, origin, dest,
# And I want to keep these variables in the end result
         dep_delay)
```

Instead of stuffing `flights_v3 -> flights_v3.1 -> flights_v4`, you simply add/remove steps in your workflow.

What happens if you had forgotten a step?

```
flights_v4_tidy <- flights %>%
  select(carrier, everything()) %>%
  filter(arr_delay>air_time) %>%
  mutate(delay_change = arr_delay - dep_delay) %>% #adding calc
  filter(delay_change <= 5) %>% #adding filter
  # mutate(did_makeup = delay_change>0) %>% #commented out
  select(carrier, arr_delay, air_time, origin, dest, dep_delay, #keep var
         # did_makeup, #commented out
         delay_change)
flights_v4_tidy
```

```
## Source: local data frame [8,890 x 7]
##
##    carrier arr_delay air_time origin  dest dep_delay delay_change
##      <chr>     <dbl>    <dbl>  <chr> <chr>     <dbl>        <dbl>
## 1       MQ       851       41    JFK   BWI       853           -2
## 2       UA       123       37    EWR   BOS       144          -21
## 3       EV       123       53    JFK   IAD       119            4
## 4       UA        78       35    EWR   BOS        84           -6
```

Tidying data with `tidyr`

Messy data!

```
health_wide
```

```
## Source: local data frame [5 x 7]
##
##   person      age   sex visitJan visitedFeb visitApr visitMarch
##    <int>    <dbl> <chr>    <dbl>      <dbl>    <dbl>      <dbl>
## 1      1 19.57633     F 7.601109   8.618436       NA   9.183470
## 2      2 17.91682     F 7.765577   8.547005 10.94203   8.798914
## 3      3 17.69338     M 8.021973   8.569269       NA         NA
## 4      4 20.64306     F 8.091600   8.970724 11.66738   9.332553
## 5      5 16.99974     M 7.394355   8.137114 11.07792   9.283588
```

- Columns in wrong order
- Columns don't follow naming convention
- nobody likes `NA's`
- Some software prefers "long" format.

From wide to long with **tidyr**

```
health_tidy <- health_wide %>%
  tidyr::gather( #gathers multiple columns into one
    key = visit_id, #columns' names become entries under column key
    value = measurement, #columns' entries all go into new column value
    starts_with('visit') #Gather these columns. Note use of helper function!!
  ) %>% arrange((person))
health_tidy
```

```
## Source: local data frame [20 x 5]
##
##    person      age   sex   visit_id measurement
##     <int>    <dbl> <chr>      <chr>       <dbl>
## 1       1 19.57633     F   visitJan    7.601109
## 2       1 19.57633     F visitedFeb    8.618436
## 3       1 19.57633     F   visitApr          NA
## 4       1 19.57633     F visitMarch    9.183470
## 5       2 17.91682     F   visitJan    7.765577
## 6       2 17.91682     F visitedFeb    8.547005
## 7       2 17.91682     F   visitApr   10.942033
```

Enjoy the *tidyverse*!

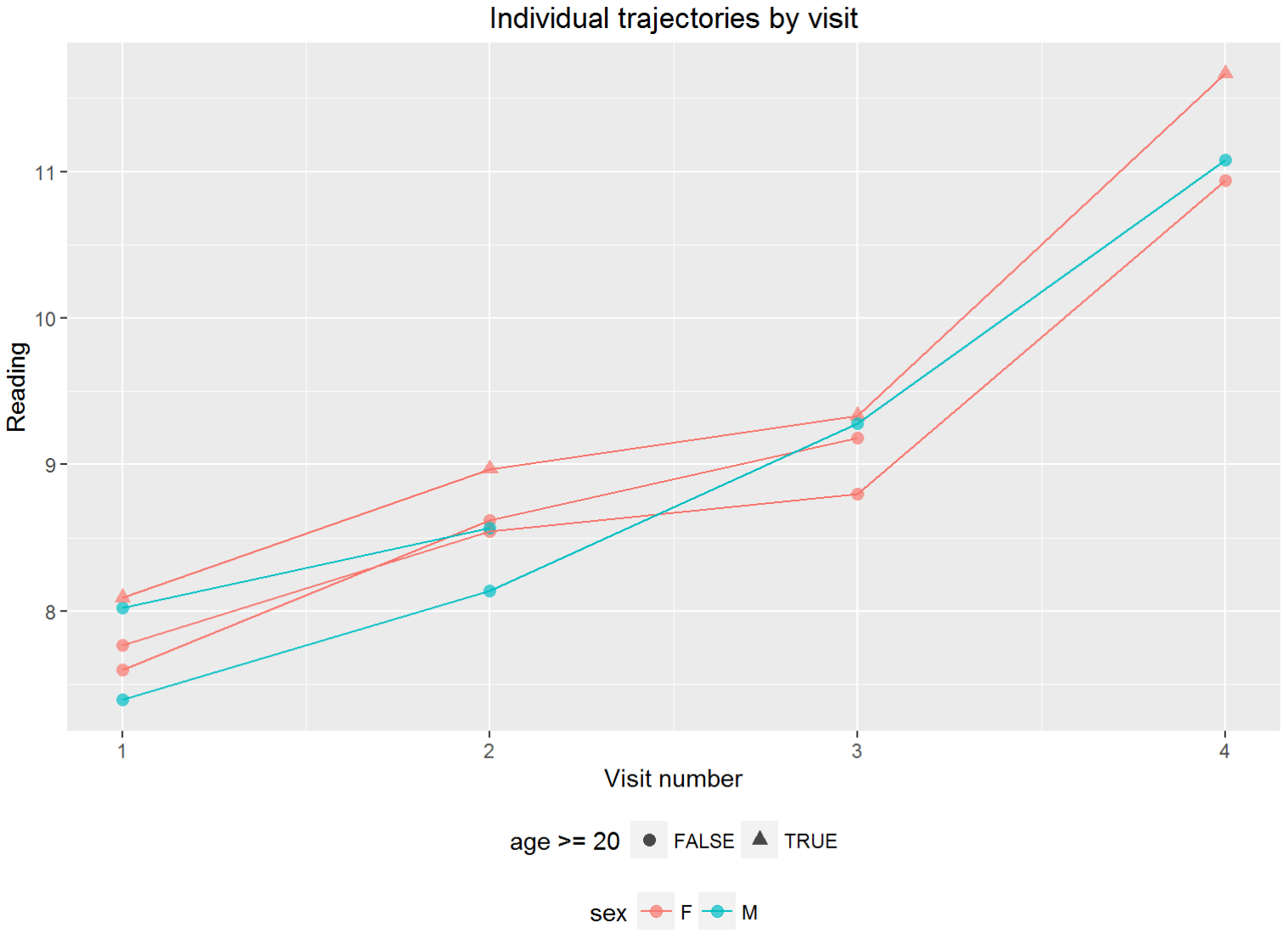
```
library(stringr) ##make character variables place nicely


month_info <- data_frame( month_num = 1:5,
  month_nickname = c("JAN", "FEB", "MAR", "APR", "MAY"))
health_tidier <- health_tidy %>% # Use stringr to modify visit_id
  mutate(visit_id = stringr::str_replace(visit_id, "visited","visit")) %>%
  mutate(visit_mon = stringr::str_replace(visit_id, "visit",""),
         visit_mon = stringr::str_sub(visit_mon, start=1,end=3),
         visit_mon = stringr::str_to_upper(visit_mon)) %>%
  left_join(month_info, by = c("visit_mon"="month_nickname"))
health_tidier


## Source: local data frame [20 x 7]
##
##    person      age   sex   visit_id measurement visit_mon month_num
##     <int>    <dbl> <chr>      <chr>       <dbl>     <chr>     <int>
## 1       1 19.57633     F   visitJan    7.601109       JAN         1
## 2       1 19.57633     F   visitFeb    8.618436       FEB         2
## 3       1 19.57633     F   visitApr          NA       APR         4
```

```r
library(ggplot2) ##Clean graphs from clear workflow

health_tidier %>%
  # Plot the trajectories
  ggplot(aes(x = month_num, y = measurement, group = person,
             color = sex, shape = age>=20)) +
  # I want a scatterplot (w/ transparency)
  geom_point(alpha = 0.4, size = 1.9) +
  # I want to connect the dots
  geom_line() +
  # I want custom axis labels
  xlab("Visit number")+ ylab("Reading") +
  # Always title your graphs
  ggtitle("Individual trajectories by visit") +
  # and I don't want the legend on the right side
  theme(legend.position = "bottom")
```


Individual trajectories by visit
Reading
11
10
9
8
1
2
3
4
Visit number
age >= 20
FALSE
TRUE
sex
F
M

Powerful additions

Save yourself keystrokes

`dplyr` can help you to use fewer keystrokes

```
summarize_each(flights_v4, ##data
               funs(mean,sd, median, max), ##actions we want
               arr_delay, air_time, dep_delay) ##vars we action on
```

```
## Source: local data frame [1 x 12]
##
##   arr_delay_mean air_time_mean dep_delay_mean arr_delay_sd air_time_sd
##            <dbl>         <dbl>          <dbl>        <dbl>       <dbl>
## 1       139.9143      78.36312       134.9555     83.15407    42.68988
## Variables not shown: dep_delay_sd <dbl>, arr_delay_median <dbl>,
##   air_time_median <dbl>, dep_delay_median <dbl>, arr_delay_max <dbl>,
##   air_time_max <dbl>, dep_delay_max <dbl>.
```

`summarize_each()` and `mutate_each()` will compute lots of quantities and give them logical names

Let the computer do the work

`dplyr` lets you work programatically (but syntax is trickier)

```
outside_factor <- 2.5 #we can incorporate objects that lie outside the data frame
test <- select(flights_v4, arr_delay, air_time, carrier)
test <- mutate(test, delay_factor = arr_delay/air_time)
mutate_(test, painfulness = ~ delay_factor+arr_delay*outside_factor) ##fancy!!
```

```
## Source: local data frame [14,009 x 5]
##
##    arr_delay air_time carrier delay_factor painfulness
##        <dbl>    <dbl>   <chr>        <dbl>       <dbl>
## 1        137      118      MQ     1.161017    343.6610
## 2        851       41      MQ    20.756098   2148.2561
## 3        123       37      UA     3.324324    310.8243
## 4        127       63      EV     2.015873    319.5159
## 5        123       53      EV     2.320755    309.8208
## 6         78       35      UA     2.228571    197.2286
## 7         59       53      EV     1.113208    148.6132
## 8         40       36      EV     1.111111    101.1111
```

Make your work reproducible

`dplyr` lets you work programatically (but syntax is trickier)

```
outside_factor <- 2.5; outside_limit <- 300
test <- select(flights_v4, arr_delay, air_time, carrier)
test <- mutate(test, delay_factor = arr_delay/air_time)
test <- mutate_(test, painfulness = ~ delay_factor+arr_delay*outside_factor)
## Program can be changed with minimal user input
my_logical_filter <- lazyeval::interp( ~ painfulness < outside_limit)
filter_(test,my_logical_filter) #super fancy!
```

```
## Source: local data frame [6,683 x 5]
##
##    arr_delay air_time carrier delay_factor painfulness
##        <dbl>    <dbl>   <chr>        <dbl>       <dbl>
## 1         78       35      UA     2.228571    197.2286
## 2         59       53      EV     1.113208    148.6132
## 3         40       36      EV     1.111111    101.1111
## 4         51       45      EV     1.133333    128.6333
## 5         44       31      EV     1.419355    111.4194
```

An example of how I may use the *tidyverse*

```
get_IPW <- function(treatment, covariates){things happen}

IPW_estimates <- the_data %>%
  group_by(cluster) %>%
  summarize_(IPW = ~ get_IPW(treatment=treat, covariates = X))

Estimates <- the_data %>% left_join(
  IPW_estimates, by = "cluster") %>%
  mutate_(weighted_outcome= ~ outcome*IPW) %>%
  group_by(cluster) %>%
  summarize(mwo = mean(weighted_outcome)) %>%
  ungroup() %>%
  summarize(target_estimate = mean(mwo))
```

Practice

Questions

1. What are median and average arrival and departure delays for flights from JFK to DCA, PIT, and ORD for each carrier?

2. Which of those carriers have highest median departure delay? Lowest mean arrival delay?

3. What are the mean and standard deviation of the flight distance and the flight duration of all flights departing EWK in February and March of 2013 for each carrier?

4. What are the mean and standard deviation of the flight distance and the flight duration of all flights departing EWK in February and March of 2013 for each carrier? What is the average miles per hour for each?

5. Which has lower average arrival delay on all flights on New Year's Eve 2013: (A) United and Delta, or (B) every other airline?

Answer to 1.

- What are the median & avg arr. and dept. delays for flights from JFK to DCA, PIT, and ORD for each carrier?

```
## Source: local data frame [5 x 2]
##
##   carrier med_DD
##     <chr>  <dbl>
## 1      AA    1.0
## 2      9E   -1.0
## 3      B6   -2.0
## 4      MQ   -3.0
## 5      DL   -3.5
```

Answer to 2.

- Which of those carriers have highest median departure delay? Lowest mean arrival delay?

```
## Source: local data frame [5 x 2]
##
##   carrier    avg_AD
##     <chr>     <dbl>
## 1      DL -9.000000
## 2      9E  6.214998
## 3      AA  8.569014
## 4      MQ 10.930963
## 5      B6 15.181448
```

Answer to 3.

- What are the mean and standard deviation of the flight distance and the flight duration of all flights departing EWK in February and March of 2013 for each carrier?

```
## Source: local data frame [10 x 5]
##
##    carrier distance_mean air_time_mean distance_sd air_time_sd
##      <chr>         <dbl>         <dbl>       <dbl>       <dbl>
## 1       9E      583.8026      99.90132    110.3566   15.191388
## 2       AA     1392.2237     204.14787    383.9105   52.841695
## 3       AS     2402.0000     327.19828      0.0000   13.099764
## 4       B6      852.8807     126.33004    384.2624   49.681161
## 5       DL      859.9838     129.45946    379.3161   49.829707
## 6       EV      549.1439      90.84714    293.4601   42.151848
## 7       MQ      719.0000     113.46615      0.0000    8.834356
## 8       UA     1383.1719     197.33808    759.9022   97.265371
## 9       US      920.5089     137.33284    689.4957   89.023826
## 10      WN     1049.0904     158.92100    542.1092   75.062915
```

Answer to 4.

- What are the mean and standard deviation of the flight distance and the flight duration of all flights departing EWK in February and March of 2013 for each carrier? What is the average miles per hour for each?

```
## Source: local data frame [10 x 2]
##
##    carrier mean(MPH)
##      <chr>     <dbl>
## 1       9E  5.844739
## 2       AA  6.836150
## 3       AS  7.352748
## 4       B6  6.463105
## 5       DL  6.588888
## 6       EV  5.830959
## 7       MQ  6.373011
## 8       UA  6.780038
## 9       US  6.414446
## 10      WN  6.387701
```

Answer to 5.

- Which has lower average arrival delay on all flights on New Year's Eve 2013: (A) United and Delta, or (B) every other airline?

```
## Source: local data frame [2 x 2]
##
##   my_var avg_arr_delay
##    <chr>         <dbl>
## 1  UA/DL      1.470817
## 2  other      8.639442
```

Discussion

`dplyr` versus `data.table`

- "Which one is better?"
  - Use whichever one you prefer
  - Either one is fast enough for you for now
  - You can/should use base `R` when it is easier

Where to go for help

Prevention

- Roger Peng's R Programming for Data Science - highly recommended. Easy read. Pay what you want.
- Use style guides
- Advanced R is challenging but great!
- There are LOTS of other books and resources. Some may not be what you want. (i.e. written with a different focus)
- r-bloggers.com (for intros and tutorials)
- twitter.com (for learning about how others solve problems)

Cure

- StackOverflow.com (for issues coming from your code)
- github.com (for issues coming from a package)

Get a stackoverflow account, follow smart people on twitter, and join the community!

- You are *NOT* the only one who has had this issue before. Someone may have even solved it by now. These communities can help!
- You may also feel free to email me with questions or to set up a meeting; Brian dot Barkley attt UNC point edu
- Go Heels!