Managing Data with `dplyr` and `tidyr`

Biostatistics 140.776

dplyr

The data frame is a key data structure in statistics and in R.

- There is one observation per row
- Each column represents a variable or measure or characteristic
- Primary implementation that you will use is likely the default R implementation
- Modified "upgraded" version in the `tibble` package
- Other implementations, particularly relational databases systems

dplyr

- Developed by RStudio
- An optimized and distilled version of `plyr` package (by Hadley Wickham)
- Provides a "grammar" (in particular, verbs) for data manipulation
- Is **very** fast, as many key operations are coded in C++
- Functions (verbs) can be chained together via a "pipe" operator

dplyr Verbs

- `select`: return a subset of the columns of a data frame
- `filter`: extract a subset of rows from a data frame based on logical conditions
- `arrange`: reorder rows of a data frame
- `rename`: rename variables in a data frame
- `mutate`: add new variables/columns or transform existing variables
- `summarize` / `summarise`: generate summary statistics of different variables in the data frame, possibly within strata

dplyr Properties

- The first argument is a data frame.
- The subsequent arguments describe what to do with it, and you can refer to columns in the data frame directly without using the $ operator (just use the names).
- The result is a new data frame
- Data frames must be properly formatted and annotated for this to all be useful
- There are no "inputs" and "results"; it's all just *data*

Load the dplyr package

This step is important!

```
library(dplyr)

Attaching package: 'dplyr'
The following objects are masked from 'package:stats':

    filter, lag
The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

select

```
library(readr)
specdata <- read_csv("SPEC_2014.csv.gz")
dim(specdata)
[1] 1519790      26
```

```
select(specdata, 1:4)
# A tibble: 1,519,790 x 4
  State.Code County.Code Site.Num Parameter.Code
  <chr>      <chr>       <chr>             <int>
1 01         073         0023              88102
2 01         073         0023              88102
3 01         073         0023              88102
4 01         073         0023              88102
5 01         073         0023              88102
# ... with 1.52e+06 more rows
```

select

```
names(specdata)[1:3]
[1] "State.Code"  "County.Code" "Site.Num"
```

```
select(specdata, State.Code:Site.Num)
# A tibble: 1,519,790 x 3
  State.Code County.Code Site.Num
  <chr>      <chr>       <chr>
1 01         073         0023
2 01         073         0023
3 01         073         0023
4 01         073         0023
5 01         073         0023
# ... with 1.52e+06 more rows
```

select

In dplyr you can do

```
select(specdata, -(State.Code:Site.Num))
```

Equivalent base R

```
i <- match("State.Code", names(specdata))
j <- match("Site.Num", names(specdata))
head(specdata[, -(i:j)])
```

filter

```
specdata.f <- filter(specdata,
                     Parameter.Name == "Sulfate PM2.5 LC")
select(specdata.f, 1:3, Parameter.Name)
# A tibble: 36,012 x 4
  State.Code County.Code Site.Num Parameter.Name
  <chr>      <chr>       <chr>    <chr>
1 01         073         0023     Sulfate PM2.5 LC
2 01         073         0023     Sulfate PM2.5 LC
3 01         073         0023     Sulfate PM2.5 LC
4 01         073         0023     Sulfate PM2.5 LC
5 01         073         0023     Sulfate PM2.5 LC
# ... with 3.601e+04 more rows
```

filter

```
specdata.f <- filter(specdata,
                     Parameter.Name == "Sulfate PM2.5 LC"
                     & State.Name == "Texas")
select(specdata.f, 1:2, Parameter.Name, State.Name)
# A tibble: 988 x 4
  State.Code County.Code Parameter.Na~ State.Name
  <chr>      <chr>       <chr>         <chr>
1 48         043         Sulfate PM2.~ Texas
2 48         043         Sulfate PM2.~ Texas
3 48         043         Sulfate PM2.~ Texas
4 48         043         Sulfate PM2.~ Texas
5 48         043         Sulfate PM2.~ Texas
# ... with 983 more rows
```

arrange

Reordering rows of a data frame (while preserving corresponding order of other columns) is normally a pain to do in R.

```
specdata <- arrange(specdata, Date.Local)
select(specdata, Date.Local, Parameter.Name,
       Sample.Value)
# A tibble: 1,519,790 x 3
  Date.Local Parameter.Name                Sample.Value
  <date>     <chr>                                <dbl>
1 2014-01-01 EC CSN PM2.5 LC TOT                  0.866
2 2014-01-01 Total Carbon PM2.5 LC ~              3.66
3 2014-01-01 Optical EC PM2.5 LC TOT              0.457
4 2014-01-01 Sulfate PM2.5 LC                     1.45
5 2014-01-01 OC CSN Unadjusted PM2.~              2.32
# ... with 1.52e+06 more rows
```

arrange

```
tail(select(specdata, Date.Local, Parameter.Name,
            Sample.Value), 3)
# A tibble: 3 x 3
  Date.Local Parameter.Name   Sample.Value
  <date>     <chr>                   <dbl>
1 2014-12-31 EC2 PM2.5 LC            0.002
2 2014-12-31 EC3 PM2.5 LC            0
3 2014-12-31 Sulfate PM2.5 LC        0.783
```

arrange

Rows can be sorted in descending order too.

```
specdata <- arrange(specdata, desc(Date.Local))
select(specdata, Date.Local, Parameter.Name,
       Sample.Value)
# A tibble: 1,519,790 x 3
  Date.Local Parameter.Name    Sample.Value
  <date>     <chr>                    <dbl>
1 2014-12-31 Antimony PM2.5 LC        0
2 2014-12-31 Arsenic PM2.5 LC         0.002
3 2014-12-31 Arsenic PM2.5 LC         0.002
4 2014-12-31 Aluminum PM2.5 LC        0.022
5 2014-12-31 Aluminum PM2.5 LC        0.018
# ... with 1.52e+06 more rows
```

rename

Renaming a variable in a data frame in R is surprising hard to do!

```
specdata[, 5:7]
# A tibble: 1,519,790 x 3
    POC Latitude Longitude
  <int>    <dbl>     <dbl>
1     5     33.6     -86.8
2     1     33.6     -86.8
3     5     33.6     -86.8
4     1     33.6     -86.8
5     5     33.6     -86.8
# ... with 1.52e+06 more rows
```

rename

```
specdata <- rename(specdata,
                   lat = Latitude,
                   lon = Longitude)
select(specdata, 5:7)
# A tibble: 1,519,790 x 3
    POC   lat   lon
  <int> <dbl> <dbl>
1     5  33.6 -86.8
2     1  33.6 -86.8
3     5  33.6 -86.8
4     1  33.6 -86.8
5     5  33.6 -86.8
# ... with 1.52e+06 more rows
```

mutate

```
specdata <-
        mutate(specdata,
               city_state = paste(City.Name, State.Name,
                                  sep = ", "),
               sample_mg = Sample.Value / 1000)
```

```
select(specdata, City.Name, State.Name, city_state,
       sample_mg)
# A tibble: 1,519,790 x 4
  City.Name  State.Name city_state          sample_mg
  <chr>      <chr>      <chr>                    <dbl>
1 Birmingham Alabama    Birmingham, Al~ 0
2 Birmingham Alabama    Birmingham, Al~ 0.000002
3 Birmingham Alabama    Birmingham, Al~ 0.000002
4 Birmingham Alabama    Birmingham, Al~ 0.000022
5 Birmingham Alabama    Birmingham, Al~ 0.0000180
# ... with 1.52e+06 more rows
```

group_by

Generating summary statistics by stratum

```
specdata <-
        mutate(specdata,
               region = factor(lon > -100,
                               labels = c("west", "east")))
eastwest <- group_by(specdata, region)
summarize(eastwest,
          pollutant = mean(Sample.Value, na.rm = TRUE),
          obs = mean(Observation.Count, na.rm = TRUE))
# A tibble: 2 x 3
  region pollutant   obs
  <fct>      <dbl> <dbl>
1 west       0.153  1.07
2 east       0.406  1.26
```

group_by

Generating summary statistics by stratum

```
library(lubridate)
specdata <- mutate(specdata, month = month(Date.Local))
months <- group_by(specdata, month)
months <- filter(months,
                 Parameter.Name == "Sulfate PM2.5 LC")
```

group_by

```
summarize(months,
          sulfate = mean(Sample.Value, na.rm = TRUE))
# A tibble: 12 x 2
   month sulfate
   <dbl>   <dbl>
 1     1   0.840
 2     2   1.31
 3     3   1.25
 4     4   1.18
 5     5   1.19
 6     6   1.30
 7     7   1.37
 8     8   1.37
 9     9   1.20
10    10   0.955
11    11   0.895
12    12   1.10
```

%>% (pipe operator)

```
specdata %>%
        mutate(month = month(Date.Local)) %>%
        filter(Parameter.Name=="Sulfate PM2.5 LC") %>%
        group_by(month) %>%
        summarize(sulfate = mean(Sample.Value,
                                 na.rm = TRUE))
```

```
# A tibble: 12 x 2
   month sulfate
   <dbl>   <dbl>
 1     1   0.840
 2     2   1.31
 3     3   1.25
 4     4   1.18
 5     5   1.19
 6     6   1.30
 7     7   1.37
 8     8   1.37
```

%>% (pipe operator)

```
specdata %>%
        mutate(month = month(Date.Local)) %>%
        filter(Parameter.Name=="Sulfate PM2.5 LC") %>%
        group_by(month, region) %>%
        summarize(sulfate = mean(Sample.Value, na.rm = TRU
```

```
# A tibble: 24 x 3
# Groups:   month [?]
  month region sulfate
  <dbl> <fct>    <dbl>
1     1 west     0.468
2     1 east     1.13
3     2 west     0.549
4     2 east     1.92
5     3 west     0.537
# ... with 19 more rows
```

dplyr

Once you learn the dplyr "grammar" there are a few additional benefits

- dplyr can work with other data frame "backends"
- `data.table` for large fast tables
- SQL interface for relational databases via the DBI package

tidyr

The `tidyr` package helps with manipulation of data frames between "wide" and "long" formats, depending on what you're trying to do.

- Sometimes the meaning of a "variable" depends on the application
- Sometimes Sulfate, Aluminum, and Nitrate are all different variables with continuous levels (wide format)
- Sometimes "Pollutant" is the variable with levels "Sulfate", "Aluminum", and "Nitrate" and a *separate* column for the values (long format)s

Long Format

```
spec <- select(specdata, city_state, Date.Local,
                  Parameter.Name, Sample.Value)
```

Here are the specdata pollution data in long format

```
spec
# A tibble: 1,519,790 x 4
  city_state Date.Local Parameter.Name
  <chr>      <date>     <chr>
1 Birmingha~ 2014-12-31 Antimony PM2.~
2 Birmingha~ 2014-12-31 Arsenic PM2.5~
3 Birmingha~ 2014-12-31 Arsenic PM2.5~
4 Birmingha~ 2014-12-31 Aluminum PM2.~
5 Birmingha~ 2014-12-31 Aluminum PM2.~
# ... with 1.52e+06 more rows, and 1 more
#   variable: Sample.Value <dbl>
```

Long Format

```
stats <- group_by(spec, Parameter.Name) %>%
        summarize(mean = mean(Sample.Value, na.rm = TRUE),
                  median = median(Sample.Value, na.rm = TRU
                  max = max(Sample.Value, na.rm = TRUE))
stats
# A tibble: 86 x 4
  Parameter.Name                mean median    max
  <chr>                        <dbl>  <dbl>  <dbl>
1 Aluminum PM2.5 LC          0.0530    0.021  2.78
2 Ammonium Ion PM2.5 LC      0.700     0.447 15.2
3 Antimony PM2.5 LC          0.00517   0       0.101
4 Arsenic PM2.5 LC           0.000333  0       0.092
5 Barium PM2.5 LC            0.00242   0       0.602
# ... with 81 more rows
```

gather

An alternate representation could have three variables: pollutant, statistic, and value

- `gather` is a function that "gathers" multiple columns and essential sticks them into one column
- The names of multiple columns become levels of a single variable
- In this case mean, median, max –> levels of a "statistic" variable

gather

```
library(tidyr)
gather(stats, statistic, value, -Parameter.Name)
# A tibble: 258 x 3
  Parameter.Name          statistic    value
  <chr>                   <chr>        <dbl>
1 Aluminum PM2.5 LC       mean       0.0530
2 Ammonium Ion PM2.5 LC   mean       0.700
3 Antimony PM2.5 LC       mean       0.00517
4 Arsenic PM2.5 LC        mean       0.000333
5 Barium PM2.5 LC         mean       0.00242
# ... with 253 more rows
```

`spread`

The `spread` function does the inverse of the `gather` function

- `spread` takes a single variable (with multiple levels) and *spreads* them across multiple columns
- Sometimes more intuitive if you want to compute a statistic across multiple levels/variables
- e.g. Compute the maximum of three different pollutants on each day and create a new variable

spread

```
wide <- filter(spec, city_state=="Essex, Maryland") %>%
        spread(Parameter.Name, Sample.Value)
wide
# A tibble: 112 x 53
  city_state Date.Local `Aluminum PM2.5~
  <chr>      <date>                <dbl>
1 Essex, Ma~ 2014-01-05            0
2 Essex, Ma~ 2014-01-08            0.016
3 Essex, Ma~ 2014-01-11            0
4 Essex, Ma~ 2014-01-14            0.013
5 Essex, Ma~ 2014-01-17            0.004
# ... with 107 more rows, and 50 more variables:
#   `Ammonium Ion PM2.5 LC` <dbl>, `Antimony
#   PM2.5 LC` <dbl>, `Arsenic PM2.5 LC` <dbl>,
#   `Barium PM2.5 LC` <dbl>, `Bromine PM2.5
#   LC` <dbl>, `Cadmium PM2.5 LC` <dbl>, `Calcium
#   PM2.5 LC` <dbl>, `Cerium PM2.5 LC` <dbl>,
#   `Cesium PM2.5 LC` <dbl>, `Chlorine PM2.5
```

separate

Sometimes you need to split one column into two separate columns.

```
m
             mood    day
1  happy-working Sunday
2 happy-partying Sunday
3  happy-working Sunday
4   sad-partying Sunday
5    sad-working Monday
6   sad-partying Monday
```

separate

```
m <- separate(m, mood, c("state", "activity"),
              sep = "-")
m
  state activity    day
1 happy  working Sunday
2 happy partying Sunday
3 happy  working Sunday
4   sad partying Sunday
5   sad  working Monday
6   sad partying Monday
```

unite

The inverse of `separate`

```
unite(m, mood, state, activity, sep = "-")
             mood    day
1  happy-working Sunday
2 happy-partying Sunday
3  happy-working Sunday
4   sad-partying Sunday
5    sad-working Monday
6   sad-partying Monday
```

Summary

`dplyr`

- Verbs/functions for manipulating data frames in tidy format
- select, filter, arrange, group_by, summarize, rename, mutate

`tidyr`

- Transform data frames from wide to long formats
- spread, gather, separate, unite