

Data Transformation

Kylie Ariel Bemis

9/18/2017

Introduction to Data Transformation

In order to visualize and (later) model data, we typically need to perform data transformation.

- ▶ Subset data by rows (observations) or columns (variables)
- ▶ Reorder the rows or columns of the data
- ▶ Calculate new variables from existing variables
- ▶ Calculate summary statistics of variables

One or more of these steps are often necessary to appropriately visualize or model a dataset.

Introduction to Data Transformation (cont'd)

This week, we will discuss data transformations on **tidy** data. That is, data that is already in a form where:

- ▶ Each variable has its own column
- ▶ Each observation has its own row
- ▶ Each value has its own cell

Next week, we will discuss more on tidy data and how to perform the data wrangling necessary to clean up *untidy* data and turn it into tidy data.

dplyr: A grammar of data manipulation

Provides a powerful, flexible “grammar” of data manipulation.

- ▶ Identify the most important data manipulation verbs and make them easy to use from R
- ▶ Provide fast performance for in-memory data with under-the-hood C++ implementation
- ▶ Use the same interface whether the data is stored in-memory or in a database on disk

Part of the “tidyverse” with ggplot2 and others.

Why use dplyr?

The dplyr package (mostly) doesn't implement any functionality that is missing or impossible to perform in base R.

However, code written with dplyr:

- ▶ Can be more expressive, concise, and human-readable
- ▶ Can be more explicit in your intentions for data manipulation
- ▶ Can be faster (sometimes) due to C++ implementation
- ▶ Can also be used with databases on disk
- ▶ Integrates with other tidyverse functions and packages

I will also show you how to perform the same functionality using base R.

Review of subsetting `data.frames` in R

- ▶ Simplifying (returns a vector)
 - ▶ Access individual columns using `df$name`
 - ▶ Access individual columns using `df[["name"]]`
- ▶ Preserving (returns a `data.frame`)
 - ▶ Subset rows using `df[i,]`
 - ▶ Subset rows using `df[c("name1", "name2"),]`
 - ▶ Subset columns using `df[,j]`
 - ▶ Subset columns using `df[,c("name1", "name2")]`
- ▶ You can subset both rows and columns at the same time
- ▶ Subsetting by a single column is always simplifying for `data.frames`
 - ▶ Change this behavior with `drop = FALSE`

```
df <- data.frame(x=c(1L, 2L, 5L, 9L),  
                 y=c('a', 'b', 'c', 'd'),  
                 `z !`=c(1.11, 2.22, 3.33, 4.0),  
                 row.names=c("Jo", "Ha", "Q", "Final"),  
                 check.names=FALSE,  
                 stringsAsFactors=FALSE)  
df
```

```
##      x y  z !  
## Jo   1 a 1.11  
## Ha   2 b 2.22  
## Q    5 c 3.33  
## Final 9 d 4.00
```

```
df$x
```

```
## [1] 1 2 5 9
```

```
df[["y"]]
```

```
## [1] "a" "b" "c" "d"
```



```
df$`z !`
```

```
## [1] 1.11 2.22 3.33 4.00
```

```
df[["z !"]]
```

```
## [1] 1.11 2.22 3.33 4.00
```

```
df[1:3,]
```

```
##      x y  z !  
## Jo  1 a 1.11  
## Ha  2 b 2.22  
## Q   5 c 3.33
```

```
df[c("Jo", "Ha", "Q"),]
```

```
##      x y  z !  
## Jo  1 a 1.11  
## Ha  2 b 2.22  
## Q   5 c 3.33
```

```
df[,2:3]
```

```
##           y  z !  
## Jo      a 1.11  
## Ha      b 2.22  
## Q       c 3.33  
## Final d 4.00
```

```
df[,c("y", "z !")]
```

```
##           y  z !  
## Jo      a 1.11  
## Ha      b 2.22  
## Q       c 3.33  
## Final d 4.00
```

```
df[, "z !"]
```

```
## [1] 1.11 2.22 3.33 4.00
```

```
df[, "z !", drop=FALSE]
```

```
##           z !  
## Jo      1.11  
## Ha      2.22  
## Q       3.33  
## Final  4.00
```

```
df[1:3, c("y", "z !")]
```

```
##      y  z !  
## Jo a 1.11  
## Ha b 2.22  
## Q  c 3.33
```

Verbs in dplyr

Provides most commonly used data manipulation actions.

- ▶ `filter()` subsets data by rows/observations
- ▶ `arrange()` reorders data by rows/observations
- ▶ `select()` subsets data by columns/variables
- ▶ `mutate()` creates new columns/variables
- ▶ `summarise()` calculates summary statistics

Each can be applied over levels of a categorical variable with `group_by()`.

Each takes a `data.frame` as the first argument and outputs a new `data.frame`.

Loading dplyr

```
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 3.4.1
```

```
##
```

```
## 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
```

Name masking and namespaces in R

Several functions are “masked” when `dplyr` is loaded.

They can still be accessed by fully qualifying their names:

- ▶ `stats::filter()`
- ▶ `stats::lag()`

Name masking and namespaces in R (cont'd)

Each package in R creates its own namespace.

R finds functions based on the order packages are loaded.

You can see this search path with `search()`:

```
search()
```

```
## [1] ".GlobalEnv"          "package:dplyr"        "package:stats"
## [4] "package:graphics"    "package:grDevices"    "package:utils"
## [7] "package:datasets"    "package:methods"      "Autoloads"
## [10] "package:base"
```

Name masking and namespaces in R (cont'd)

When name conflicts occur, a warning about masked names is given.

You can always use `package::function()` to find the right one.

Example dataset

Today we will explore the `flights` dataset also used in the homework and the **R4DS** book.

```
library(nycflights13)
flights
```

Note that `flights` is actually a `tibble`, which is simply a special type of `data.frame`.

I will use `data.frame` to refer to both interchangeably. We will discuss the differences in more depth later in the course.

Subsetting rows with filter()

Get only flights from October.

In base R:

```
flights[flights$month == 10,]
```

In dplyr:

```
filter(flights, month == 10)
```

```
filter(flights, month == 10)
```

```
## # A tibble: 28,889 x 19
```

```
##   year month   day dep_time sched_dep_time dep_delay arr_time
##   <int> <int> <int>   <int>         <int>         <dbl>   <int>
## 1  2013    10     1     447           500          -13     614
## 2  2013    10     1     522           517           5     735
## 3  2013    10     1     536           545          -9     809
## 4  2013    10     1     539           545          -6     801
## 5  2013    10     1     539           545          -6     917
## 6  2013    10     1     544           550          -6     912
## 7  2013    10     1     549           600         -11     653
## 8  2013    10     1     550           600         -10     648
## 9  2013    10     1     550           600         -10     649
## 10 2013    10     1     551           600          -9     727
```

```
## # ... with 28,879 more rows, and 12 more variables: sched_arr_time <
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <
## #   minute <dbl>, time_hour <dtm>
```

Why `flights$month` vs `month`?

Like `ggplot2`, `dplyr` uses non-standard evaluation to facilitate ease of interactive programming.

In base R:

```
flights[flights$month == 10,]
```

In `dplyr`:

```
filter(flights, month == 10)
```

This approach is useful for interactive data analysis, but can cause problems when used within user-defined functions.

We may discuss more on non-standard evaluation later in the course if there is interest.

Anatomy of `filter()`

```
filter(data, condition1, condition2, condition3, ...)
```

- ▶ The first argument is the data
- ▶ The following arguments are vectorized logical expressions
- ▶ Additional arguments are joined by `&` (AND)
- ▶ Rows that evaluate to `TRUE` are kept
- ▶ Rows that evaluate to `FALSE` or `NA` are dropped

Review of logical operators in R

- ▶ Standard comparison operators: `==`, `!=`, `>`, `<`, `>=`, `<=`.
 - ▶ Remember to use `==` instead of `=` when doing comparisons
- ▶ `dplyr::near()` for checking floating point equality
- ▶ `dplyr::between()` is a synonym for `a <= x & x <= b`
- ▶ `&`, `|`, and `!` are vectorized AND, OR, and NOT
 - ▶ Non-vectorized versions (for `if` statements) are `&&` and `||`
- ▶ `%in%` checks if an element exists in a set
 - ▶ E.g., `x %in% c(a,b)` is equivalent to `x == a | x == b`
- ▶ `is.na` to check for missing values
 - ▶ Remember that `NA == NA` evaluates to `NA`

Get only flights from Alaska Airlines or Hawaiian Airlines.

In base R:

```
flights[flights$carrier %in% c("AS", "HA"),]
```

In dplyr:

```
filter(flights, carrier %in% c("AS", "HA"))
```

```
filter(flights, carrier %in% c("AS", "HA"))
```

```
## # A tibble: 1,056 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     724           725         -1    1020
## 2  2013     1     1     857           900         -3    1516
## 3  2013     1     1    1808          1815         -7    2111
## 4  2013     1     2     722           725         -3     949
## 5  2013     1     2     909           900          9    1525
## 6  2013     1     2    1818          1815          3    2131
## 7  2013     1     3     724           725         -1    1012
## 8  2013     1     3     914           900         14    1504
## 9  2013     1     3    1817          1815          2    2121
## 10 2013     1     4     725           725          0    1031
```

```
## # ... with 1,046 more rows, and 12 more variables: sched_arr_time <int>,
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <int>,
## #   minute <dbl>, time_hour <dtm>
```

Get only flights from Alaska Airlines or Hawaiian Airlines.

In base R:

```
flights[flights$carrier == "AS" | flights$carrier == "HA",]
```

In dplyr:

```
filter(flights, carrier == "AS" | carrier == "HA")
```

```
filter(flights, carrier == "AS" | carrier == "HA")
```

```
## # A tibble: 1,056 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     724           725          -1    1020
## 2  2013     1     1     857           900          -3    1516
## 3  2013     1     1    1808          1815          -7    2111
## 4  2013     1     2     722           725          -3     949
## 5  2013     1     2     909           900           9    1525
## 6  2013     1     2    1818          1815           3    2131
## 7  2013     1     3     724           725          -1    1012
## 8  2013     1     3     914           900          14    1504
## 9  2013     1     3    1817          1815           2    2121
## 10 2013     1     4     725           725           0    1031
```

```
## # ... with 1,046 more rows, and 12 more variables: sched_arr_time <int>,
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <int>,
## #   minute <dbl>, time_hour <dtm>
```

Get only flights between Honolulu and JFK.

In base R:

```
flights[flights$origin == "JFK" & flights$dest == "HNL",]
```

In dplyr:

```
filter(flights, origin == "JFK" & dest == "HNL")
```

```
filter(flights, origin == "JFK" & dest == "HNL")
```

```
## # A tibble: 342 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     857           900          -3    1516
## 2  2013     1     2     909           900           9    1525
## 3  2013     1     3     914           900          14    1504
## 4  2013     1     4     900           900           0    1516
## 5  2013     1     5     858           900          -2    1519
## 6  2013     1     6    1019           900          79    1558
## 7  2013     1     7    1042           900         102    1620
## 8  2013     1     8     901           900           1    1504
## 9  2013     1     9     641           900        1301    1242
## 10 2013     1    10     859           900          -1    1449
```

```
## # ... with 332 more rows, and 12 more variables: sched_arr_time <int>
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <
## #   minute <dbl>, time_hour <dtm>
```

Get only flights between Honolulu and JFK.

In base R:

```
flights[flights$origin == "JFK" & flights$dest == "HNL",]
```

In dplyr:

```
filter(flights, origin == "JFK", dest == "HNL")
```

```
filter(flights, origin == "JFK", dest == "HNL")
```

```
## # A tibble: 342 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     857           900          -3    1516
## 2  2013     1     2     909           900           9    1525
## 3  2013     1     3     914           900          14    1504
## 4  2013     1     4     900           900           0    1516
## 5  2013     1     5     858           900          -2    1519
## 6  2013     1     6    1019           900          79    1558
## 7  2013     1     7    1042           900         102    1620
## 8  2013     1     8     901           900           1    1504
## 9  2013     1     9     641           900        1301    1242
## 10 2013     1    10     859           900          -1    1449
```

```
## # ... with 332 more rows, and 12 more variables: sched_arr_time <int>
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <
## #   minute <dbl>, time_hour <dtm>
```


Reordering rows with `arrange()`

Sort by flights that departed with least delay (most ahead of schedule):

In base R:

```
flights[order(flights$dep_delay),]
```

In dplyr:

```
arrange(flights, dep_delay)
```

```
arrange(flights, dep_delay)
```

```
## # A tibble: 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    12     7    2040           2123         -43     40
## 2  2013     2     3    2022           2055         -33    2240
## 3  2013    11    10    1408           1440         -32    1549
## 4  2013     1    11    1900           1930         -30    2233
## 5  2013     1    29    1703           1730         -27    1947
## 6  2013     8     9     729            755         -26    1002
## 7  2013    10    23    1907           1932         -25    2143
## 8  2013     3    30    2030           2055         -25    2213
## 9  2013     3     2    1431           1455         -24    1601
## 10 2013     5     5     934            958         -24    1225
```

```
## # ... with 336,766 more rows, and 12 more variables: sched_arr_time
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <
## #   minute <dbl>, time_hour <dtm>
```

Use desc() to sort by a variable in descending order

Sort by flights that departed with most delay:

In base R:

```
flights[order(flights$dep_delay, decreasing=TRUE),]
```

In dplyr:

```
arrange(flights, desc(dep_delay))
```

```
arrange(flights, desc(dep_delay))
```

```
## # A tibble: 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     9     641           900         1301    1242
## 2  2013     6    15    1432          1935         1137    1607
## 3  2013     1    10    1121          1635         1126    1239
## 4  2013     9    20    1139          1845         1014    1457
## 5  2013     7    22     845          1600         1005    1044
## 6  2013     4    10    1100          1900          960    1342
## 7  2013     3    17    2321           810          911     135
## 8  2013     6    27     959          1900          899    1236
## 9  2013     7    22    2257           759          898     121
## 10 2013    12     5     756          1700          896    1058
```

```
## # ... with 336,766 more rows, and 12 more variables: sched_arr_time
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <
## #   minute <dbl>, time_hour <dtm>
```

Subsetting columns with `select()`

Keep only date and delay columns.

In base R:

```
flights[,c("year", "month", "day", "dep_delay", "arr_delay")]
```

In dplyr:

```
select(flights, year, month, day, dep_delay, arr_delay)
```

```
select(flights, year, month, day, dep_delay, arr_delay)
```

```
## # A tibble: 336,776 x 5
##   year month   day dep_delay arr_delay
##   <int> <int> <int>     <dbl>     <dbl>
## 1  2013     1     1         2         11
## 2  2013     1     1         4         20
## 3  2013     1     1         2         33
## 4  2013     1     1        -1        -18
## 5  2013     1     1        -6        -25
## 6  2013     1     1        -4         12
## 7  2013     1     1        -5         19
## 8  2013     1     1        -3        -14
## 9  2013     1     1        -3        -8
## 10 2013     1     1        -2         8
## # ... with 336,766 more rows
```

Keep first 9 columns (year through arr_delay).

In base R:

```
flights[,1:9]
```

In dplyr:

```
select(flights, 1:9)
```

Keep first 9 columns (year through arr_delay) by name.

In base R:

```
flights[,which(names(flights)=="year"):  
          which(names(flights)=="arr_delay")]
```

In dplyr:

```
select(flights, year:arr_delay)
```



```
select(flights, year:arr_delay)
```

```
## # A tibble: 336,776 x 9
```

```
##   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
```

```
## # ... with 336,766 more rows, and 2 more variables: sched_arr_time <
```

```
## #   arr_delay <dbl>
```

Keep all columns except tail number and flight number.

In base R:

```
flights[, -c(which(names(flights)=="tailnum"),  
             which(names(flights)=="flight"))]
```

In dplyr:

```
select(flights, -tailnum, -flight)
```

```
select(flights, -tailnum, -flight)
```

```
## # A tibble: 336,776 x 17
```

```
##   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
```

```
## # ... with 336,766 more rows, and 10 more variables: sched_arr_time
## #   arr_delay <dbl>, carrier <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
## #   time_hour <dtm>
```

Useful functions for selecting column/variable names

- ▶ `dplyr::starts_with("arr")`
 - ▶ matches column names that begin with "arr"
- ▶ `dplyr::ends_with("time")`
 - ▶ matches column names that end with "time"
- ▶ `dplyr::contains("dep")`
 - ▶ matches column names that contain "dep"
- ▶ `dplyr::num_range("x", 1:3)`
 - ▶ matches "x1", "x2", and "x3"

Keep only columns starting with “arr”.

In base R:

```
flights[,substr(names(flights), 1, 3) == "arr"]
```

In dplyr:

```
select(flights, starts_with("arr"))
```

```
select(flights, starts_with("arr"))
```

```
## # A tibble: 336,776 x 2
##   arr_time arr_delay
##   <int>     <dbl>
## 1      830         11
## 2      850         20
## 3      923         33
## 4     1004        -18
## 5      812        -25
## 6      740         12
## 7      913         19
## 8      709        -14
## 9      838         -8
## 10     753          8
## # ... with 336,766 more rows
```

Rename columns with `rename()`

`rename()` is a variant of `select()` that keeps all variables/columns while renaming the specified ones.

```
flights2 <- flights
names(flights2)[names(flights2)=="year" |
                 names(flights2)=="month" |
                 names(flights2)=="day"] <- c("YEAR", "MONTH", "DAY")
```

```
rename(flights, YEAR=year, MONTH=month, DAY=day)
```

```
rename(flights, YEAR=year, MONTH=month, DAY=day)
```

```
## # A tibble: 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
```

```
## # ... with 336,766 more rows, and 12 more variables: sched_arr_time
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <
## #   minute <dbl>, time_hour <dtm>
```


New variables with mutate()

Create a new variable giving the average air speed (in mph) of each flight.

In base R:

```
flights2 <- flights  
flights2$speed <- flights2$distance / flights2$air_time * 60
```

In dplyr:

```
mutate(flights, speed = distance / air_time * 60)
```

```
mutate(flights, speed = distance / air_time * 60)
```

```
## # A tibble: 336,776 x 20
```

```
##   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
```

```
## # ... with 336,766 more rows, and 13 more variables: sched_arr_time
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <
## #   minute <dbl>, time_hour <dtm>, speed <dbl>
```

Create two new variables giving (1) the average air speed (in mph) of each flight and (2) the amount of time gained in the air.

In base R:

```
flights2 <- flights
flights2$speed <- flights2$distance / flights2$air_time * 60
flights2$gain <- flights2$arr_delay - flights2$dep_delay
```

In dplyr:

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

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

```
## # A tibble: 336,776 x 21
```

```
##   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
```

```
## # ... with 336,766 more rows, and 14 more variables: sched_arr_time
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <
## #   minute <dbl>, time_hour <dtm>, speed <dbl>, gain <dbl>
```

Anatomy of mutate()

```
mutate(data, var1 = expr1, var2 = expr2, ...)
```

- ▶ The first argument is the data
- ▶ The following arguments are named vectorized expressions that output a vector of the same length
- ▶ When used with `data.frames`, you can use variables created in the same `mutate()` call in the subsequent expressions

Useful functions for creating variables

- ▶ Arithmetic such as `+`, `-`, `*`, `/`, `^`, etc.
 - ▶ These are vectorized and will recycle shorter variables
- ▶ Modular arithmetic such as `%%` and `%/%`
 - ▶ Useful for breaking apart integers (e.g., time into hours + minutes)
- ▶ Logs such as `log()`, `log2()` and `log10`
 - ▶ Useful for data with multiplicative variance
- ▶ Offsets such as `dplyr::lead()` and `dplyr::lag()`
 - ▶ Useful for running differences and data over time points
- ▶ Cumulative summaries such as `cumsum()`, `cumprod()`, `cummax()`, and `cummin()`
 - ▶ Also `dplyr::cummean()` for running means
- ▶ Logical operators such as `'=='`, `!='`, `>`, `<`, `>=`, `<=`, etc.
 - ▶ Useful for turning continuous variables into categorical
- ▶ `dplyr::n()` gives the number of observations
 - ▶ `n()` can only be used inside `mutate()`, `filter()` and `summarise()`

New variables with transmute()

`transmute()` is a variant of `mutate()` that keeps only the new variables and drops the rest.

In base R:

```
data.frame(distance = flights$distance,  
            speed = flights$distance / flights$air_time * 60,  
            gain = flights$arr_delay - flights$dep_delay,  
            gain_per_mile = (flights$arr_delay - flights$dep_delay) /  
                             flights$distance)
```

In dplyr:

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

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

```
## # A tibble: 336,776 x 4  
##   distance    speed  gain gain_per_mile  
##   <dbl>    <dbl> <dbl>         <dbl>  
## 1    1400  370.0441     9    0.006428571  
## 2    1416  374.2731    16    0.011299435  
## 3    1089  408.3750    31    0.028466483  
## 4    1576  516.7213   -17   -0.010786802  
## 5     762  394.1379   -19   -0.024934383  
## 6     719  287.6000    16    0.022253129  
## 7    1065  404.4304    24    0.022535211  
## 8     229  259.2453   -11   -0.048034934  
## 9     944  404.5714    -5   -0.005296610  
## 10     733  318.6957    10    0.013642565  
## # ... with 336,766 more rows
```


Summary statistics with summarise()

Get the mean departure and arrival delay.

In base R:

```
data.frame(mean_dep_delay = mean(flights$dep_delay,  
                                na.rm=TRUE),  
           mean_arr_delay = mean(flights$arr_delay,  
                                na.rm=TRUE))
```

In dplyr:

```
summarise(flights,  
          mean_dep_delay = mean(dep_delay, na.rm=TRUE),  
          mean_arr_delay = mean(arr_delay, na.rm=TRUE))
```

```
summarise(flights,  
          mean_dep_delay = mean(dep_delay, na.rm=TRUE),  
          mean_arr_delay = mean(arr_delay, na.rm=TRUE))
```

```
## # A tibble: 1 x 2  
##   mean_dep_delay mean_arr_delay  
##           <dbl>           <dbl>  
## 1      12.63907         6.895377
```

Anatomy of summarise()

```
summarise(data, summary1 = expr1, summary2 = expr2, ...)
```

- ▶ The first argument is the data
- ▶ The following arguments are expressions that output a single value from a vector of values
- ▶ It is particularly important to consider missing values when summarizing data
- ▶ Also available as `summarize()`

Useful functions for calculating summary statistics

- ▶ Measures of location such as `mean()` and `median()`
- ▶ Measures of spread such as `sd()`, `var()`, `IQR()`, and `mad()`
- ▶ Measures of rank such as `min()`, `max()`, and `quantile()`
- ▶ Counts such as:
 - ▶ `dplyr::n()` gives the number of observations
 - ▶ `sum(!is.na(x))` gives the number of non-missing values
 - ▶ `dplyr::n_distinct()` gives the number of unique values
- ▶ Remember that `sum(x == 10)` gives the count of `x == 10`
 - ▶ What does `mean(x == 10)` calculate?

Calculate the proportion of flights delayed more than 2 hours on arrival.

```
summarise(flights, mean(arr_delay > 120, na.rm=TRUE))
```

Calculate the number of unique airline carriers.

```
summarise(flights, n_distinct(carrier))
```

Calculate the proportion of flights with missing air times.

```
summarise(flights, sum(is.na(air_time)) / n())
```

```
summarise(flights, mean(arr_delay > 120, na.rm=TRUE))
```

```
## # A tibble: 1 x 1
##   `mean(arr_delay > 120, na.rm = TRUE)`
##                                     <dbl>
## 1                                0.03065258
```

```
summarise(flights, n_distinct(carrier))
```

```
## # A tibble: 1 x 1
##   `n_distinct(carrier)`
##                 <int>
## 1                    16
```

```
summarise(flights, sum(is.na(air_time)) / n())
```

```
## # A tibble: 1 x 1
##   `sum(is.na(air_time))/n()`
##                 <dbl>
## 1                0.02800081
```

Grouped transformations with `group_by()`

`summarise()` and the other data manipulation verbs in `dplyr` become much more powerful when paired with `group_by()`.

Count the number of flights from each carrier.

```
summarise(group_by(flights, carrier), n())
```

Calculate the average arrival delay for each carrier.

```
summarise(group_by(flights, carrier),  
           mean(arr_delay, na.rm=TRUE))
```

You can group by multiple variables.

Use `ungroup()` to ungroup a grouped dataset.

```
summarise(group_by(flights, carrier),  
           mean(arr_delay, na.rm=TRUE))
```

```
## # A tibble: 16 x 2  
##   carrier `mean(arr_delay, na.rm = TRUE)`  
##   <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  
## 11     OO          11.9310345  
## 12     UA           3.5580111  
## 13     US           2.1295951  
## 14     VX           1.7644644  
## 15     WN           9.6491199  
## 16     YV          15.5569853
```


Piping with the pipe operator

Combining multiple `dplyr` verbs becomes much more expressive when used with the pipe operator `%>%`.

The pipe operator takes the return value of the expression on the LHS and turns it into the first argument of the function on the RHS.

```
foo(bar(baz(x)))
```

is the same as

```
baz(x) %>% bar() %>% foo()
```

is the same as

```
x %>% baz() %>% bar() %>% foo()
```

Piping with the pipe operator (cont'd)

```
summarise(group_by(flights, carrier),  
           mean(arr_delay, na.rm=TRUE))
```

becomes

```
group_by(flights, carrier) %>%  
  summarise(mean(arr_delay, na.rm=TRUE))
```

or

```
flights %>%  
  group_by(carrier) %>%  
  summarise(mean(arr_delay, na.rm=TRUE))
```

```
flights %>%  
  group_by(carrier) %>%  
  summarise(mean(arr_delay, na.rm=TRUE))
```

```
## # A tibble: 16 x 2  
##   carrier `mean(arr_delay, na.rm = TRUE)`  
##   <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  
## 11     OO          11.9310345  
## 12     UA           3.5580111  
## 13     US           2.1295951  
## 14     VX           1.7644644  
## 15     WN           9.6491199  
## 16     YV          15.5569853
```

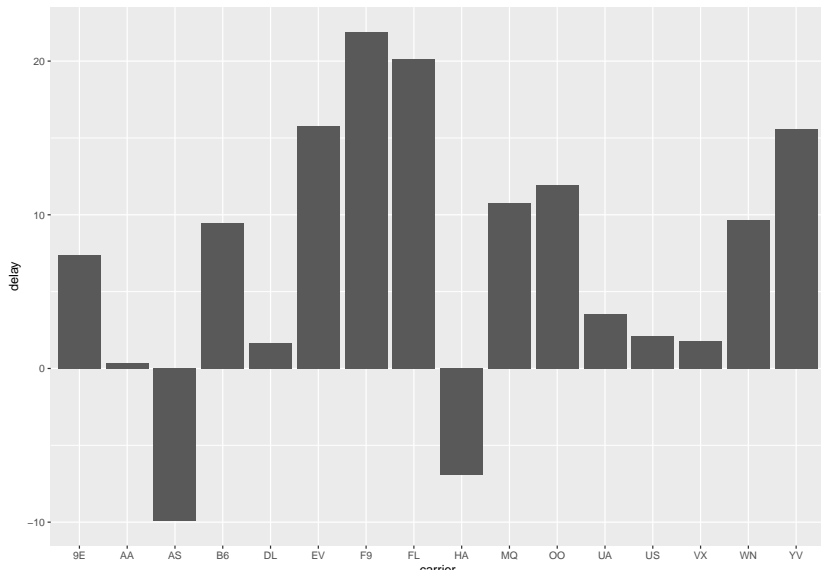
Data transformation + visualization

You can chain together dplyr verbs with ggplot2 too.

Visualize the average delay for each carrier.

```
library(ggplot2)
flights %>%
  group_by(carrier) %>%
  summarise(delay=mean(arr_delay, na.rm=TRUE)) %>%
  ggplot(aes(x=carrier, y=delay)) + geom_col()
```

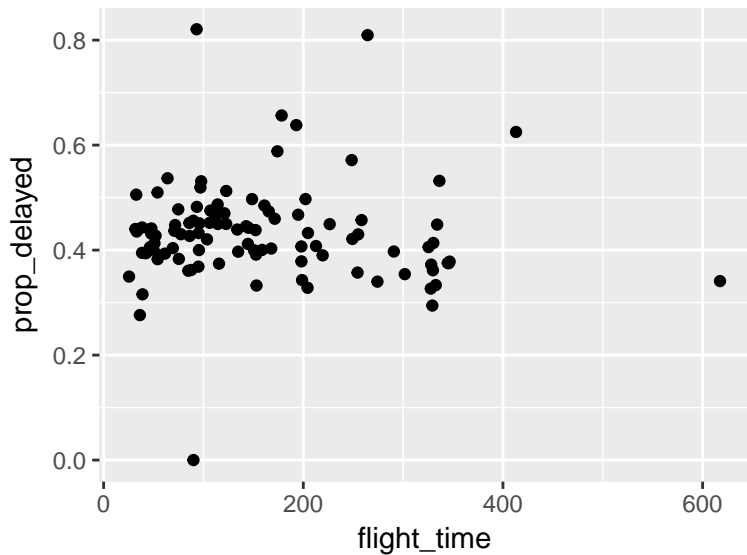
```
flights %>%  
  group_by(carrier) %>%  
  summarise(delay=mean(arr_delay, na.rm=TRUE)) %>%  
  ggplot(aes(x=carrier, y=delay)) + geom_col()
```



For each destination, visualize the proportion of delayed arriving flights versus the average flight time in the air.

```
flights %>%  
  group_by(dest) %>%  
  summarise(prop_delayed = mean(arr_delay > 0, na.rm=TRUE),  
            flight_time = mean(air_time, na.rm=TRUE),  
            count = n()) %>%  
  ggplot(aes(x=flight_time, y=prop_delayed)) + geom_point()
```

```
## Warning: Removed 1 rows containing missing values (geom_point)
```



Map the number of flights from each destination to an aesthetic.

Add a smooth fitted line.

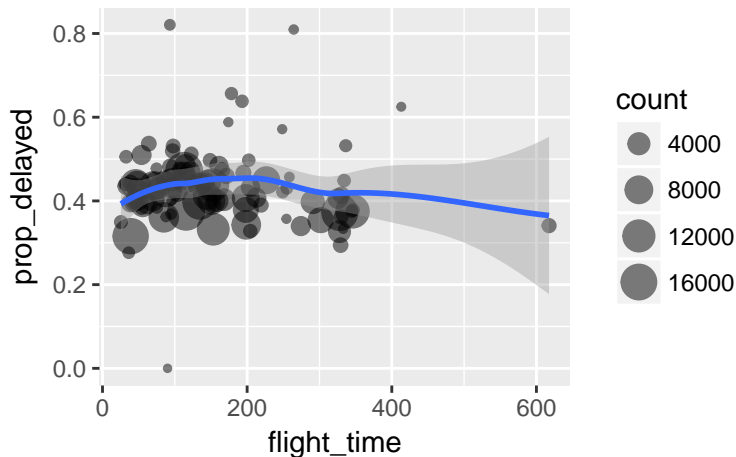
```
flights %>%  
  group_by(dest) %>%  
  summarise(prop_delayed = mean(arr_delay > 0, na.rm=TRUE),  
            flight_time = mean(air_time, na.rm=TRUE),  
            count = n()) %>%  
  ggplot(aes(x=flight_time,  
             y=prop_delayed)) +  
  geom_point(aes(size=count), alpha=1/2) +  
  geom_smooth()
```



```
## `geom_smooth()` using method = 'loess'
```

```
## Warning: Removed 1 rows containing non-finite values (stat_smooth)
```

```
## Warning: Removed 1 rows containing missing values (geom_point)
```



Exercises

- ▶ For carriers that flew more than 1000 flights in 2013, find the number of flights that weren't delayed on arrival.
- ▶ Find the average distance flown by each carrier in each month of 2013.
- ▶ Plot total miles flown each month in 2013.
- ▶ Plot the proportion of flights delayed by 10 minutes or more for each hour of the day.
- ▶ For each plane, plot the total distance flown versus the total time in arrival delays.
- ▶ Plot the relationship between the time non-scheduled time in the air and the distance for each flight.
- ▶ For each destination, plot the average speed flown versus the average distance flown.
- ▶ Find the fastest plane.