

Stat 260, Lecture 4, Data Transformation

Brad McNeney

Load packages

```
library(gapminder)
library(dplyr)
library(nycflights13)
head(flights)
```

```
## # A tibble: 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
## # ... with 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 <dbl>, minute <dbl>,
## #   time_hour <dtm>
```

Reading

- ▶ Workflow basics: Chapter 2 of printed text, Chapter 4 of online text
- ▶ Data transformation: Chapter 3 of printed text, Chapter 5 of online text
- ▶ dplyr cheatsheet at
[<https://github.com/rstudio/cheatsheets/raw/master/data-transformation.pdf>]

Data manipulation with dplyr

- ▶ We have seen the need to manipulate datasets when studying visualization:

```
gapminder <- mutate(gapminder, log10GdpPercap = log10(gdpPercap))  
gm07 <- filter(gapminder, year==2007)
```

- ▶ In this lecture we will cover data manipulation more systematically

dplyr

- ▶ The tidyverse package dplyr contains intuitive tools for manipulating data sets.
- ▶ These tools are named after verbs.
- ▶ The five key verbs (functions) are
 1. `filter()` to select subsets of observations (rows).
 2. `arrange()` to reorder rows
 3. `select()` to select variables (columns)
 4. `mutate()` to create new variables from existing ones, and
 5. `summarize()` to calculate summary statistics

Filter rows with `filter()`

- ▶ We previously used `filter()` to extract a subset of the `gapminder` dataset based on a logical condition.

```
gm07 <- filter(gapminder, year==2007)
```

- ▶ `gm07` now contains the rows of `gapminder` in which the variable `year` is equal to 2007.
- ▶ The `==` is a comparison.
- ▶ We specify the rows to extract with comparison and logical operators.

Relational (comparison) operators

- ▶ The basic relational operators are described in ?Comparison: == is equal, != is not equal, > is greater than, < is less than, >= is greater than or equal, <= is less than or equal.
- ▶ Watch out for finite-precision arithmetic.

```
2>3; 2/2 == 1; sqrt(2)^2 == 2; near(sqrt(2)^2,2)
```

```
## [1] FALSE
```

```
## [1] TRUE
```

```
## [1] FALSE
```

```
## [1] TRUE
```


Logical operators

- ▶ The basic logical operators are described in ?Logic: ! is NOT, & is AND, | is OR.

```
!TRUE ; TRUE | FALSE & FALSE; (TRUE | FALSE) & FALSE # eval parentheses first
```

```
## [1] FALSE
```

```
## [1] TRUE
```

```
## [1] FALSE
```

- ▶ Relational comparisons can be combined with logicals

```
(2==2) | (2==3)
```

```
## [1] TRUE
```

- ▶ Logical operations between vectors are element-wise.

```
x <- c(TRUE,TRUE,FALSE); y <- c(FALSE,TRUE,TRUE)  
!x ; x&y ; x|y
```

```
## [1] FALSE FALSE TRUE
```

```
## [1] FALSE TRUE FALSE
```

```
## [1] TRUE TRUE TRUE
```

filter() example

- ▶ Extract all flights from January, with departure delay of 1 or more:

```
jan13 <- filter(flights, month==1 & dep_delay > 1)
```

- ▶ **Exercises** Extract all flights from January or February. Extract all flights from January or February that have a departure delay of 1 or more.

Many logicals and %in%

- ▶ In lab 2 we saw the %in% operator:

```
hiv <- read.csv("../Labs/HIVprev.csv", stringsAsFactors = FALSE)
cc <- c("Botswana", "Central African Republic", "Congo", "Kenya", "Lesotho", "Malawi")
hihiv <- filter(hiv, Country %in% cc)
```

- ▶ This is a convenient alternative to

```
filter(hiv, Country=="Botswana" | Country=="Central African Republic", etc. )
```

- ▶ **Exercise** The nycflights13 package includes a table `airlines` that translates the two-letter airline codes in `flights` into the full names of the airlines. Extract all flights operated by United, American or Delta.

Missing data: NA

- ▶ In R, missing data (not available) is denoted by NA.
- ▶ NA takes precedence in all comparison and arithmetic operations, and almost all logical operations.

```
NA > 3; NA+10; NA & TRUE; NA | TRUE
```

```
## [1] NA
```

```
## [1] NA
```

```
## [1] NA
```

```
## [1] TRUE
```

is.na()

- ▶ Test for NA with is.na():

```
vv <- c(10, NA, 1)
is.na(vv)
```

```
## [1] FALSE TRUE FALSE
```

```
vv>1
```

```
## [1] TRUE NA FALSE
```

```
is.na(vv) | vv> 1
```

```
## [1] TRUE TRUE FALSE
```

- ▶ **Exercise** Extract all flights from January with missing departure delay.

Sorting with arrange()

- ▶ `arrange()` changes the order of rows, putting NAs last

```
vv <- tibble(x=c(NA,10,10,1),y=c("one","two","three","four"))  
arrange(vv,x)
```

```
## # A tibble: 4 x 2  
##       x y  
##   <dbl> <chr>  
## 1     1 four  
## 2    10 two  
## 3    10 three  
## 4    NA one
```

```
arrange(vv,x,y)
```

```
## # A tibble: 4 x 2  
##       x y  
##   <dbl> <chr>  
## 1     1 four  
## 2    10 three  
## 3    10 two  
## 4    NA one
```

Descending order

- ▶ The default in `arrange()` is increasing order.
- ▶ Use `desc()` for descending order.

```
arrange(vv, desc(x))
```

```
## # A tibble: 4 x 2
##       x y
##   <dbl> <chr>
## 1    10 two
## 2    10 three
## 3     1 four
## 4    NA one
```

- ▶ **Exercise** Arrange the mpg data set by *decreasing order* in the number of cylinders (variable `cyl`) and *increasing order* by engine displacement (variable `displ`) within cylinders.

Selecting columns with select()

- ▶ Where as filter() subsets by row, select() subsets by column.
 - ▶ We specify columns by their name, possibly using helper functions.
- ▶ Select month, day, hour and minute from flights:

```
select(flights,month,day,hour,minute)
```

```
## # A tibble: 336,776 x 4
##   month   day hour minute
##   <int> <int> <dbl>  <dbl>
## 1     1     1     5     15
## 2     1     1     5     29
## 3     1     1     5     40
## 4     1     1     5     45
## 5     1     1     6      0
## 6     1     1     5     58
## 7     1     1     6      0
## 8     1     1     6      0
## 9     1     1     6      0
## 10    1     1     6      0
## # ... with 336,766 more rows
```


Ranges

- Select or de-select a range of columns with :

```
select(flights, month:minute)
```

```
## # A tibble: 336,776 x 17
##   month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int>   <int>         <int>         <dbl>   <int>         <int>
## 1     1     1     517             515           2       830           819
## 2     1     1     533             529           4       850           830
## 3     1     1     542             540           2       923           850
## 4     1     1     544             545          -1      1004          1022
## 5     1     1     554             600          -6       812           837
## 6     1     1     554             558          -4       740           728
## 7     1     1     555             600          -5       913           854
## 8     1     1     557             600          -3       709           723
## 9     1     1     557             600          -3       838           846
## 10    1     1     558             600          -2       753           745
## # ... with 336,766 more rows, and 10 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>
select(flights, -(month:minute))
```

```
## # A tibble: 336,776 x 2
##   year time_hour
##   <int> <dtm>
## 1  2013 2013-01-01 05:00:00
## 2  2013 2013-01-01 05:00:00
## 3  2013 2013-01-01 05:00:00
## 4  2013 2013-01-01 05:00:00
## 5  2013 2013-01-01 06:00:00
## 6  2013 2013-01-01 05:00:00
## 7  2013 2013-01-01 06:00:00
## 8  2013 2013-01-01 06:00:00
## 9  2013 2013-01-01 06:00:00
```

Renaming variables

- ▶ You can rename variables as they are selected.
- ▶ To rename a variable without selecting it use `rename()`.

```
flights <- rename(flights, tail_num = tailnum)
```

Helper functions

- ▶ Some useful helper functions are `starts_with()`, `ends_with()` and `contains()`.
 - ▶ See `?select` for a complete list.

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

```
## # A tibble: 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
## # ... with 336,766 more rows
```

- ▶ **Exercise** Select all variables with “dep” or “arr” in the name.

Add new variables with mutate()

- ▶ We've used mutate() to add log-GDP to the gapminder data set.

```
gapminder <- mutate(gapminder, log10GdpPercap = log10(gdpPercap))
```

- ▶ You can use variables as they are created by mutate():

```
flights <- mutate(flights, gain = arr_delay - dep_delay, gainh = gain/60)
```

Summaries and grouping

- ▶ A common task is “split-apply-combine”: We want to split a data set into groups defined by one variable, apply a statistical summary to each group, and then combine the results.
- ▶ With dplyr we use `group_by()` and `summarize()`.

```
by_day <- group_by(flights, month, day)
summarize(by_day, count = n(), delay = mean(dep_delay, na.rm=TRUE))
```

```
## # A tibble: 365 x 4
## # Groups:   month [12]
##   month    day count delay
##   <int> <int> <int> <dbl>
## 1     1     1   842  11.5
## 2     1     2   943  13.9
## 3     1     3   914  11.0
## 4     1     4   915   8.95
## 5     1     5   720   5.73
## 6     1     6   832   7.15
## 7     1     7   933   5.42
## 8     1     8   899   2.55
## 9     1     9   902   2.28
## 10    1    10   932   2.84
## # ... with 355 more rows
```

- ▶ Note: `mean()` returns NA whenever there are missing values.

Combining operations with the pipe

- ▶ The pipe `%>%` can be used to chain together operations, without the need to store intermediate results

```
select(flights, month, day, dep_delay) %>%  
  group_by(month, day) %>%  
  summarize(count = n(), delay = mean(dep_delay, na.rm = TRUE))
```

```
## # A tibble: 365 x 4  
## # Groups:   month [12]  
##   month    day count delay  
##   <int> <int> <int> <dbl>  
## 1     1     1    842  11.5  
## 2     1     2    943  13.9  
## 3     1     3    914  11.0  
## 4     1     4    915   8.95  
## 5     1     5    720   5.73  
## 6     1     6    832   7.15  
## 7     1     7    933   5.42  
## 8     1     8    899   2.55  
## 9     1     9    902   2.28  
## 10    1    10    932   2.84  
## # ... with 355 more rows
```

- ▶ Notice that we omit the data set name when a function receives data from the pipe.