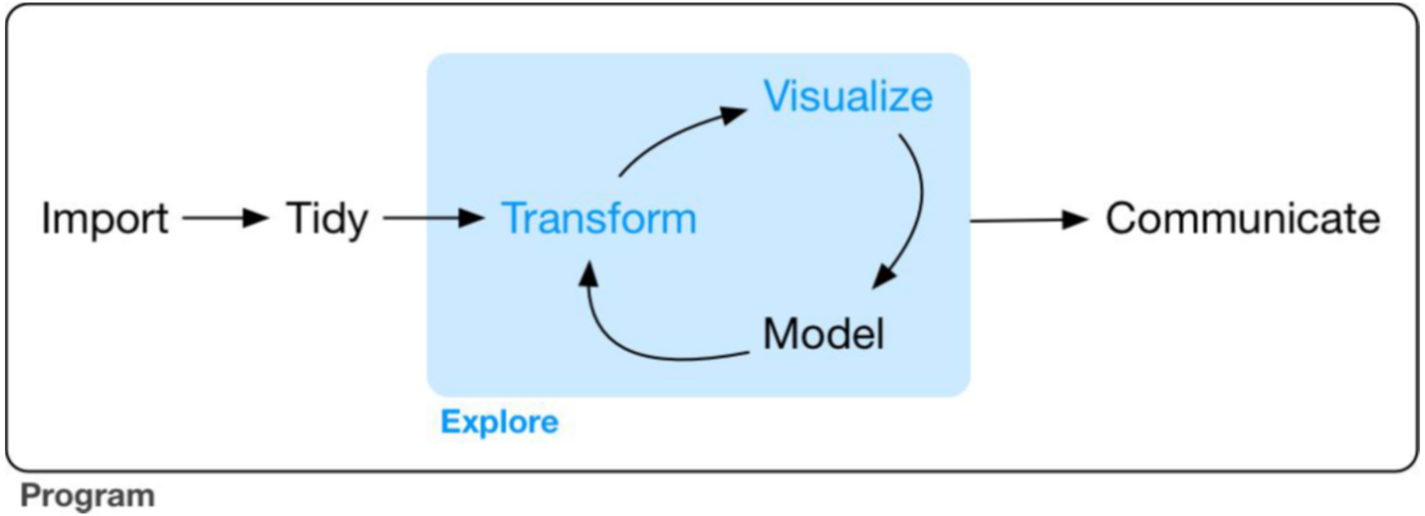
IE6600 Computation and Visualization for Analytics

Data Transformation with dplyr

(updated: 2022-06-13)

Data Transformation

Goal



Wickham, Hadley, and Garrett Grolemund. R For Data Science. OReilly, 2017.

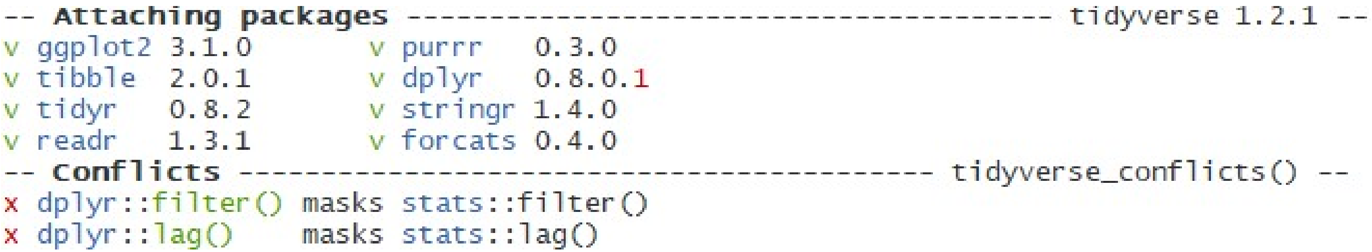
Tips

Try press Alt-Shift-K.

Data Transformation with dplyr

Often you’ll need to create some new variables or summaries, or maybe you just want to rename the variables or reorder the observations in order to make the data a little easier to work with.

```r
library(nycflights13)
library(dplyr)
#or
library(tidyverse)
```



```
-- Attaching packages --------------------------------------- tidyverse 1.2.1 --
v ggplot2 3.1.0       v purrr   0.3.0
v tibble  2.0.1       v dplyr   0.8.0.1
v tidyr   0.8.2       v stringr 1.4.0
v readr   1.3.1       v forcats 0.4.0
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
```

The conflict message tells you some of the other functions have been overwrited by Tidyverse. If you want to use the base version of these functions after loading dplyr, you’ll need to use their full names: stats::filter() and stats::lag(),etc.

search()

```
search()
```

```
##  [1] ".GlobalEnv"          "package:forcats"     "package:stringr"
##  [4] "package:dplyr"       "package:purrr"       "package:readr"
##  [7] "package:tidyr"       "package:tibble"      "package:ggplot2"
## [10] "package:tidyverse"   "package:nycflights13" "package:stats"
## [13] "package:graphics"    "package:grDevices"   "package:utils"
## [16] "package:datasets"    "package:methods"     "Autoloads"
## [19] "package:base"
```

You may use the full syntax package::function_name() to load the specific function, if there are any overwriting issues occured.

nycflights13

To explore the basic data manipulation verbs of dplyr, we'll use nycflights13::flights. This data frame contains all 336,776 flights that departed from New York City in 2013:

```
head(flights)
```

```
## # A tibble: 6 × 19
##    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
## 1  2013     1     1      517            515         2      830            819
## 2  2013     1     1      533            529         4      850            830
## 3  2013     1     1      542            540         2      923            850
## 4  2013     1     1      544            545        -1     1004           1022
## 5  2013     1     1      554            600        -6      812            837
## 6  2013     1     1      554            558        -4      740            728
## # … with 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

Interview data with view()

```
view(flights)
```

nycflights13 (cont'd)

You might also have noticed the row of three- (or four-) letter abbreviations under the column names. These describe the type of each variable:

- `int` stands for integers.
- `dbl` stands for doubles, or real numbers.
- `chr` stands for character vectors, or strings.
- `dttm` stands for date-times (a date + a time).

Flights actually is a tibble, a special type of data.frame. We will talk about it later.

nycflights13 (cont'd)

There are three other common types of variables that aren’t used in this dataset but you’ll encounter later.

- `lgl` stands for logical, vectors that contain only TRUE or FALSE.
- `fctr` stands for factors, which R uses to represent categorical variables with fixed possible values.
- `date` stands for dates.

dplyr Basics

In this slides you are going to learn the five key dplyr functions that allow you to solve the vast majority of your data-manipulation challenges:

- `filter()` pick observations by their values .
- `arrange()` Reorder the rows.
- `select()` Pick variables by their names.
- `mutate()` Create new variables with functions of existing variables.
- `summarize()` Collapse many values down to a single summary.
- `group_by()` Conjunction.

dplyr grammar

All verbs work similarly: `filter(df, argument,...)`

1. The first argument is a data frame.
2. The subsequent arguments describe what to do with the data frame, using the variable names (without quotes).
3. The result is a new data frame.

filter()

Filter rows with filter()

filter() allows you to subset observations based on their values.

Base function in R:

```r
flights[flights$month==1&flights$day==1,]
```

filter() function in dplyr:

```r
filter(flights, month==1,day==1)
```

Filter rows with filter() (cont'd)

We only want to see the Jan.1st flights

```
jan <- filter(flights, month==1, day==1)
jan
```

```
## # A tibble: 842 × 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     1      517            515         2      830            819
##  2  2013     1     1      533            529         4      850            830
##  3  2013     1     1      542            540         2      923            850
##  4  2013     1     1      544            545        -1     1004           1022
##  5  2013     1     1      554            600        -6      812            837
##  6  2013     1     1      554            558        -4      740            728
##  7  2013     1     1      555            600        -5      913            854
##  8  2013     1     1      557            600        -3      709            723
##  9  2013     1     1      557            600        -3      838            846
## 10  2013     1     1      558            600        -2      753            745
## # … with 832 more rows, and 11 more variables: arr_delay <dbl>, carrier <chr>,
## #   flight <int>, tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>,
## #   distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

Filter rows with filter(): examples

```r
dec25 <- filter(flights, month == 12, day == 25)
dec25
```

```
## # A tibble: 719 × 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013    12    25      456            500        -4      649            651
##  2  2013    12    25      524            515         9      805            814
##  3  2013    12    25      542            540         2      832            850
##  4  2013    12    25      546            550        -4     1022           1027
##  5  2013    12    25      556            600        -4      730            745
##  6  2013    12    25      557            600        -3      743            752
##  7  2013    12    25      557            600        -3      818            831
##  8  2013    12    25      559            600        -1      855            856
##  9  2013    12    25      559            600        -1      849            855
## 10  2013    12    25      600            600         0      850            846
## # … with 709 more rows, and 11 more variables: arr_delay <dbl>, carrier <chr>,
## #   flight <int>, tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>,
## #   distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

filter - comparison

To use filtering effectively, you have to know how to select the observations that you want using the comparison operators. R provides the standard suite: >, >=, <, <=, != (not equal), and == (equal).

Flights on Feb~Dec, and before 28 th

```r
filter(flights, month>1&!day>28)
```

Flights on Feb~Dec, and before 28 th

```r
flights28 <- filter(flights, month>1&!day>28)
flights28
```

```
## # A tibble: 285,972 × 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013    10     1      447            500       -13      614            648
##  2  2013    10     1      522            517         5      735            757
##  3  2013    10     1      536            545        -9      809            855
##  4  2013    10     1      539            545        -6      801            827
##  5  2013    10     1      539            545        -6      917            933
##  6  2013    10     1      544            550        -6      912            932
##  7  2013    10     1      549            600       -11      653            716
##  8  2013    10     1      550            600       -10      648            700
##  9  2013    10     1      550            600       -10      649            659
## 10  2013    10     1      551            600        -9      727            730
## # … with 285,962 more rows, and 11 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

tips

When you're starting out with R, the easiest mistake to make is to use = instead of == when testing for equality. When this happens you'll get an informative error:

```r
filter(flights, month = 1)
```

```
## Error in `filter()`:
## ! We detected a named input.
## ℹ This usually means that you've used `=` instead of `==`.
## ℹ Did you mean `month == 1`?
```

tips (cont'd)

There's another common problem you might encounter when using ==: floating-point numbers.

```
sqrt(2)^2==2
```

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

```
1/49*49==1
```

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

tips (cont'd)

Computers use finite precision arithmetic (they obviously can’t store an infinite number of digits!) so remember that every number you see is an approximation. Instead of relying on ==, use near():

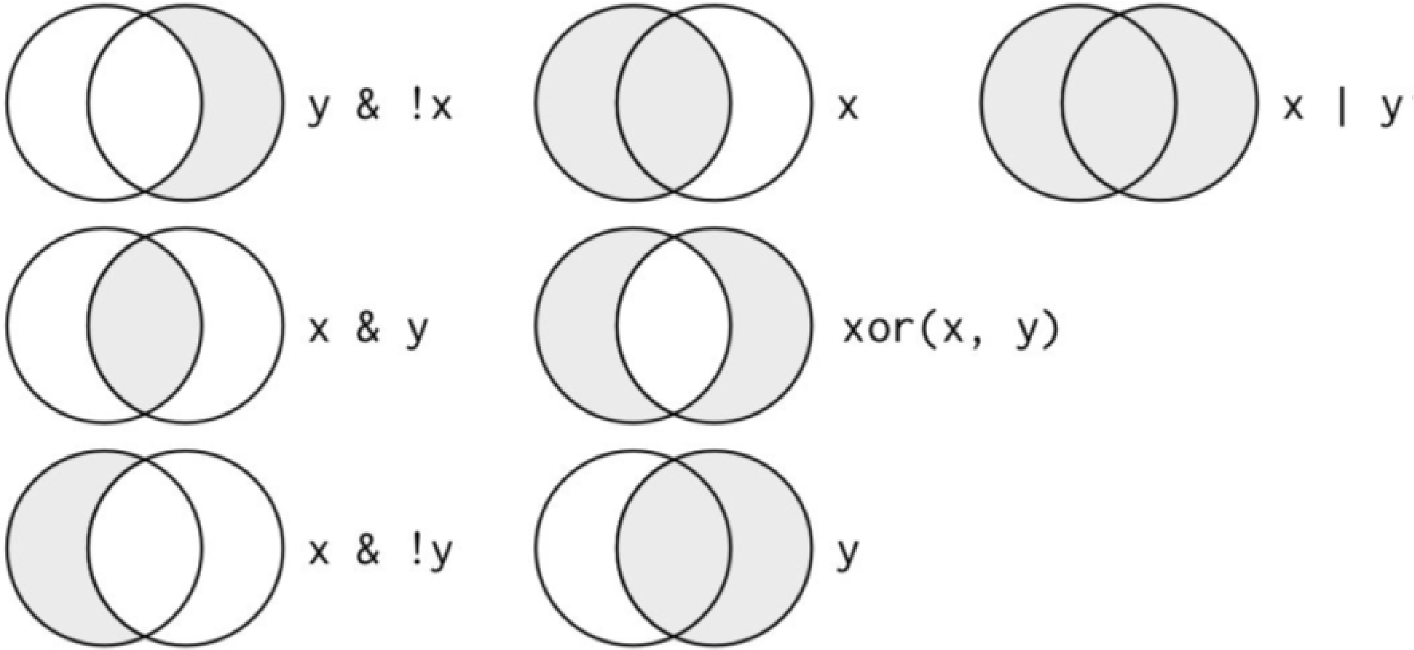
```
near(sqrt(2) ^ 2, 2)
```

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

```
near(1 / 49 * 49, 1)
```

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

Logical Operators



Flights on May or June

```r
mayJune <- filter(flights, month==5|month==6)
mayJune$month %>% unique()
```

```
## [1] 5 6
```

%in%

A useful shorthand for this problem is x %in% y. This will select every row where x is one of the values in y. We could use it to rewrite the preceding code:

Retrieve the flights information on Jan, Feb, and Mar

```
filter(flights, month%in%c(1,2,3)) %>% head()
```

!= not equals to

Flights not on Feb

```
filter(flights, month!=2)
```

```
## # A tibble: 311,825 × 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     1      517            515         2      830            819
##  2  2013     1     1      533            529         4      850            830
##  3  2013     1     1      542            540         2      923            850
##  4  2013     1     1      544            545        -1     1004           1022
##  5  2013     1     1      554            600        -6      812            837
##  6  2013     1     1      554            558        -4      740            728
##  7  2013     1     1      555            600        -5      913            854
##  8  2013     1     1      557            600        -3      709            723
##  9  2013     1     1      557            600        -3      838            846
## 10  2013     1     1      558            600        -2      753            745
## # … with 311,815 more rows, and 11 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

|, &, and ","

Compare the following three code chunks

```
filter(flights, !(arr_delay > 120 | dep_delay > 120)) %>%
  select(dep_delay) %>% head(3)
```

```
filter(flights, arr_delay <= 120, dep_delay <= 120) %>%
  select(dep_delay) %>% head(3)
```

```
filter(flights, !arr_delay > 120 & !dep_delay > 120) %>%
  select(dep_delay) %>% head(3)
```

```
## # A tibble: 3 × 1
##   dep_delay
##       <dbl>
## 1         2
## 2         4
## 3         2
```

Missing Values

One important feature of R that can make comparison tricky is missing values, or NAs (“not availables”).

```r
NA>5
```

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

```r
NA==10
```

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

```r
NA+10
```

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

```r
NA/2
```

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

NA==NA

The most confusing result is this one:

```
NA==NA
```

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

But we can understand it easily in one example:

```
# Let ZAge be Zhenyuan's age. We don't know how old he is.
ZAge <- NA
# Let TAge be Trump's age. We don't know how old he is. (don't google it)
TAge <- NA
# Are Zhenyuan and Trump the same age?
ZAge == TAge
```

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

```
# We don't know!
```

NA with filter()

filter() only includes rows where the condition is TRUE; it excludes both FALSE and NA values. If you want to preserve missing values, ask for them explicitly:

```
df <- tibble(x = c(1, NA, 3))
filter(df, x > 1)
```

```
## # A tibble: 1 × 1
##       x
##   <dbl>
## 1     3
```

```
filter(df, is.na(x) | x > 1)
```

```
## # A tibble: 2 × 1
##       x
##   <dbl>
## 1    NA
## 2     3
```

Some other functions

```r
is.na(NA)
```

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

```r
df <- data.frame(A=c(1,NA,2))

na.omit(df)
```

```
##   A
## 1 1
## 3 2
```

```r
sum(df[,1], na.rm=T)
```

```
## [1] 3
```

Exercise 1

Find all flights that:

1. Flew to Houston (IAH or HOU)

2. Were operated by United(UA), American(AA), or Delta(DL)

3. Departed in summer (July, August, and September)

Exercise 2

Use data set `msleep`, and create a new data frame of mammals with feeding type `carnivore` and brain weight less than the average of brain weight over all mammals. Make sure no NA values in column of brain weight.

arrange()

Arrange Rows with arrange()

arrange() works similarly to filter() except that instead of selecting rows, it changes their order

Base function in R:

```
flights[order(flights$year,flights$month, flights$day, decreasing=F),]
```

arrange() function in dplyr:

```
arrange(flights, year, month, day)
```

Arrange Rows with arrange() (cont'd)

```
arrange(flights, year, month, day)
```

```
## # A tibble: 336,776 × 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     1      517            515         2      830            819
##  2  2013     1     1      533            529         4      850            830
##  3  2013     1     1      542            540         2      923            850
##  4  2013     1     1      544            545        -1     1004           1022
##  5  2013     1     1      554            600        -6      812            837
##  6  2013     1     1      554            558        -4      740            728
##  7  2013     1     1      555            600        -5      913            854
##  8  2013     1     1      557            600        -3      709            723
##  9  2013     1     1      557            600        -3      838            846
## 10  2013     1     1      558            600        -2      753            745
## # … with 336,766 more rows, and 11 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

Use desc() to reorder by a column in descending order

```
arrange(flights, desc(arr_delay))
```

```
## # A tibble: 336,776 × 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     9      641            900      1301     1242           1530
##  2  2013     6    15     1432           1935      1137     1607           2120
##  3  2013     1    10     1121           1635      1126     1239           1810
##  4  2013     9    20     1139           1845      1014     1457           2210
##  5  2013     7    22      845           1600      1005     1044           1815
##  6  2013     4    10     1100           1900       960     1342           2211
##  7  2013     3    17     2321            810       911      135           1020
##  8  2013     7    22     2257            759       898      121           1026
##  9  2013    12     5      756           1700       896     1058           2020
## 10  2013     5     3     1133           2055       878     1250           2215
## # … with 336,766 more rows, and 11 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

Missing values are always sorted at the end

```
df <- data.frame(x = c(5, 2, NA))
arrange(df, x) #or arrange(df, desc(x))
```

```
##    x
## 1  2
## 2  5
## 3 NA
```

Exercises

1. Sort flights to find the most delayed flights. Find the flights that left earliest.
2. Sort flights to find the fastest flights.

select()

Select Columns with select()

select() allows you to rapidly zoom in on a useful subset using operations based on the names of the variables.

Base function in R:

```r
# Select columns by name
flights[,c("year","month","day")]
```

select() function in dplyr:

```r
# Select columns by name
select(flights, year, month, day)
```

Select Columns with select() (cont'd)

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

```
## # A tibble: 336,776 × 3
##     year month   day
##    <int> <int> <int>
##  1  2013     1     1
##  2  2013     1     1
##  3  2013     1     1
##  4  2013     1     1
##  5  2013     1     1
##  6  2013     1     1
##  7  2013     1     1
##  8  2013     1     1
##  9  2013     1     1
## 10  2013     1     1
## # … with 336,766 more rows
```

Select all colums between year and day

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

```
## # A tibble: 336,776 × 3
##     year month   day
##    <int> <int> <int>
##  1  2013     1     1
##  2  2013     1     1
##  3  2013     1     1
##  4  2013     1     1
##  5  2013     1     1
##  6  2013     1     1
##  7  2013     1     1
##  8  2013     1     1
##  9  2013     1     1
## 10  2013     1     1
## # … with 336,766 more rows
```

Select all columns except those from year to day

```
select(flights, -(year:day))
```

```
## # A tibble: 336,776 × 16
##    dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay carrier
##       <int>          <int>     <dbl>    <int>          <int>     <dbl> <chr>
##  1      517            515         2      830            819        11 UA
##  2      533            529         4      850            830        20 UA
##  3      542            540         2      923            850        33 AA
##  4      544            545        -1     1004           1022       -18 B6
##  5      554            600        -6      812            837       -25 DL
##  6      554            558        -4      740            728        12 UA
##  7      555            600        -5      913            854        19 B6
##  8      557            600        -3      709            723       -14 EV
##  9      557            600        -3      838            846        -8 B6
## 10      558            600        -2      753            745         8 AA
## # … with 336,766 more rows, and 9 more variables: flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>
```

Other arguments within select()

There are a number of helper functions you can use within select():

- `starts_with("abc")` matches names that begin with “abc”.
- `ends_with("xyz")` matches names that end with “xyz”.
- `contains("ijk")` matches names that contain “ijk”.
- `matches("^a")` selects variables that match a regular expression. (check R4DS "regular expressions")
- `num_range("x", 1:3)` matches x1, x2, and x3.

select(): examples

```
abc.df <- data.frame(apple=c("b", "c"), an.orange=1:2, orange1=2:3)
abc.df
```

```
##   apple an.orange orange1
## 1     b         1       2
## 2     c         2       3
```

```
select(abc.df, starts_with("app"))
```

```
##   apple
## 1     b
## 2     c
```

```r
select(abc.df, ends_with("ge"))
```

```
##   an.orange
## 1         1
## 2         2
```

```r
select(abc.df, contains("pp"))
```

```
##   apple
## 1     b
## 2     c
```

```r
select(abc.df, matches("^a"))
```

```
##   apple an.orange
## 1     b         1
## 2     c         2
```

```r
select(abc.df, num_range("orange", 1))
```

```
##   orange1
## 1       2
## 2       3
```

mutate()

Add New Variables with mutate()

Besides selecting sets of existing columns, it’s often useful to add new columns that are functions of existing columns. That’s the job of `mutate()`.

Create a new data frame

```r
temp <- data.frame(A=c(1:3),B=c(2:4))
```

Base function in R:

```r
temp$C <- temp$A-temp$B
```

mutate() function in dplyr:

```r
mutate(temp, C=A-B)
```

Add New Variables with mutate() (cont'd)

```r
temp <- data.frame(A=c(1:3),B=c(2:4))
mutate(temp, C=A-B)
```

```
##   A B  C
## 1 1 2 -1
## 2 2 3 -1
## 3 3 4 -1
```

transmute() for only keeping the new variables

If you only want to keep the new variables, use transmute():

```
transmute(temp, C=A-B)
```

```
##    C
## 1 -1
## 2 -1
## 3 -1
```

Example I

```r
flights_delay <- select(flights,
                        year:day,
                        ends_with("delay"),
                        distance,
                        air_time)
mutate(flights_delay,
       gain = arr_delay - dep_delay,
       speed = distance / air_time * 60)
```

```
## # A tibble: 336,776 × 9
##     year month   day dep_delay arr_delay distance air_time  gain speed
##    <int> <int> <int>     <dbl>     <dbl>    <dbl>    <dbl> <dbl> <dbl>
##  1  2013     1     1         2        11     1400      227     9  370.
##  2  2013     1     1         4        20     1416      227    16  374.
##  3  2013     1     1         2        33     1089      160    31  408.
##  4  2013     1     1        -1       -18     1576      183   -17  517.
##  5  2013     1     1        -6       -25      762      116   -19  394.
##  6  2013     1     1        -4        12      719      150    16  288.
##  7  2013     1     1        -5        19     1065      158    24  404.
##  8  2013     1     1        -3       -14      229       53   -11  259.
##  9  2013     1     1        -3        -8      944      140    -5  405.
## 10  2013     1     1        -2         8      733      138    10  319.
## # … with 336,766 more rows
```

Example II

Note that you can refer to columns that you've just created:

```
mutate(
  flights_delay,
  gain = arr_delay - dep_delay, # New column
  hours = air_time / 60, # New column
  gain_per_hour = gain / hours
)
```

```
## # A tibble: 336,776 × 10
##     year month   day dep_delay arr_delay distance air_time  gain hours
##    <int> <int> <int>     <dbl>     <dbl>    <dbl>    <dbl> <dbl> <dbl>
##  1  2013     1     1         2        11     1400      227     9 3.78
##  2  2013     1     1         4        20     1416      227    16 3.78
##  3  2013     1     1         2        33     1089      160    31 2.67
##  4  2013     1     1        -1       -18     1576      183   -17 3.05
##  5  2013     1     1        -6       -25      762      116   -19 1.93
##  6  2013     1     1        -4        12      719      150    16 2.5
##  7  2013     1     1        -5        19     1065      158    24 2.63
##  8  2013     1     1        -3       -14      229       53   -11 0.883
##  9  2013     1     1        -3        -8      944      140    -5 2.33
## 10  2013     1     1        -2         8      733      138    10 2.3
```

Example III with transmute() for only keep the new variables

```
transmute(
  flights,
  gain = arr_delay - dep_delay,
  hours = air_time / 60,
  gain_per_hour = gain / hours
)
```

```
## # A tibble: 336,776 × 3
##     gain hours gain_per_hour
##    <dbl> <dbl>         <dbl>
##  1     9 3.78           2.38
##  2    16 3.78           4.23
##  3    31 2.67          11.6 
##  4   -17 3.05          -5.57
##  5   -19 1.93          -9.83
##  6    16 2.5            6.4 
##  7    24 2.63           9.11
##  8   -11 0.883        -12.5 
##  9    -5 2.33          -2.14
## 10    10 2.3            4.35
```

mutate() with ifelse()

```r
temp <- data.frame(A=c(1:3),B=c(2:4))
temp
```

```
##   A B
## 1 1 2
## 2 2 3
## 3 3 4
```

mutate() with ifelse() (cont'd)

In column B, We would like to replace all the values with the following pattern: if observation >3, then 1; if observation <=3, then 0

```
mutate(temp, B=ifelse(B>3, 1, 0))
```

```
##   A B
## 1 1 0
## 2 2 0
## 3 3 1
```

rename(), recode()

rename()

Rename variables by name

```r
temp <- data.frame(A=c(1:3),B=c(2:4))
temp
```

```
##   A B
## 1 1 2
## 2 2 3
## 3 3 4
```

rename() vs names()

Rename column A as A1, column B as B1

```
names(temp) <- c("A1", "B1")
```

```
rename(temp, A1=A,B1=B)
```

```
##   A1 B1
## 1  1  2
## 2  2  3
## 3  3  4
```

recode()

Recode values

```
temp1 <- data.frame(A=letters[1:3], B=c(1:3))
temp1
```

```
##   A B
## 1 a 1
## 2 b 2
## 3 c 3
```

```
recode(temp1$A, "b"="a")
```

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

```
mutate(temp1, A=recode(temp1$A, "b"="2"))
```

```
##   A B
## 1 a 1
## 2 2 2
## 3 c 3
```

```
# temp1 %>% mutate(A=recode(A, "b"="2"))
```

```
mutate(temp1, B=recode(temp1$B, "2"="b"))
```

```
##   A    B
## 1 a <NA>
## 2 b    b
## 3 c <NA>
```

According to the coercion rule, all the elements should be in the same data type within one variable/vector. since "b" is char data type, when 2 (integer) has been replaced with b, the other two integer 1 and 3 will be overwrited as NA.

Exercise

Let's create one data frame as follows:

```
##   A B  C
## 1 a 1  2
## 2 b 2 NA
## 3 c 3 13
## 4 d 4 56
## 5 e 5 NA
```

Then we would like to replace all the NAs in column C. If the value is NA, we replace it with the mean of column C, otherwise just keep the value.

```
data.frame(A=letters[1:5], B=c(1:5), C=c(2, NA, 13, 56, NA)) %>%
  mutate(C=ifelse(is.na(C), mean(C,na.rm = T),C))
```

summarize()

Summaries with summarize()

The last key verb is summarize(). It collapses a data frame to a single row:

```
summarize(flights, delay = mean(dep_delay, na.rm = TRUE))
```

```
## # A tibble: 1 × 1
##   delay
##   <dbl>
## 1  12.6
```

summarize() with group_by()

For example, if we applied exactly the same code to a data frame grouped by date, we get the average delay per month:

```
flights %>%
  group_by(year, month) %>%
  summarize(delay=mean(dep_delay, na.rm = T))
```

```r
flights %>%
  group_by(year, month) %>%
  summarize(delay=mean(dep_delay, na.rm = T))
```

```
## `summarise()` has grouped output by 'year'. You can override using the
## `.groups` argument.
```

```
## # A tibble: 12 × 3
## # Groups:   year [1]
##     year month delay
##    <int> <int> <dbl>
##  1  2013     1 10.0
##  2  2013     2 10.8
##  3  2013     3 13.2
##  4  2013     4 13.9
##  5  2013     5 13.0
##  6  2013     6 20.8
##  7  2013     7 21.7
##  8  2013     8 12.6
##  9  2013     9  6.72
## 10  2013    10  6.24
## 11  2013    11  5.44
## 12  2013    12 16.6
```

Missing values with summarize()

You may have wondered about the na.rm argument we used earlier. What happens if we don’t set it?

```
flights %>%
  group_by(year, month, day) %>%
  summarize(mean = mean(dep_delay))
```

```
## `summarise()` has grouped output by 'year', 'month'. You can override using the
## `.groups` argument.

## # A tibble: 365 × 4
## # Groups:   year, month [12]
##     year month   day  mean
##    <int> <int> <int> <dbl>
##  1  2013     1     1    NA
##  2  2013     1     2    NA
##  3  2013     1     3    NA
##  4  2013     1     4    NA
##  5  2013     1     5    NA
##  6  2013     1     6    NA
##  7  2013     1     7    NA
##  8  2013     1     8    NA
```

Removing missing values with na.rm in summarize()

```
flights %>%
  group_by(year, month, day) %>%
  summarize(mean = mean(dep_delay, na.rm = TRUE))
```

```
## `summarise()` has grouped output by 'year', 'month'. You can override using the
## `.groups` argument.
```

```
## # A tibble: 365 × 4
## # Groups:   year, month [12]
##     year month   day  mean
##    <int> <int> <int> <dbl>
##  1  2013     1     1 11.5 
##  2  2013     1     2 13.9 
##  3  2013     1     3 11.0 
##  4  2013     1     4  8.95
##  5  2013     1     5  5.73
##  6  2013     1     6  7.15
##  7  2013     1     7  5.42
##  8  2013     1     8  2.55
##  9  2013     1     9  2.28
```

When you use the dplyr verbs on a grouped data frame they'll be automatically applied "by group." For example, if we applied exactly the same code to a data frame grouped by date, we get the average delay per date: By year, month, and date

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

```
## `summarise()` has grouped output by 'year', 'month'. You can override using the
## `.groups` argument.

## # A tibble: 365 × 4
## # Groups:   year, month [12]
##     year month   day delay
##    <int> <int> <int> <dbl>
##  1  2013     1     1 11.5 
##  2  2013     1     2 13.9 
##  3  2013     1     3 11.0 
##  4  2013     1     4  8.95
##  5  2013     1     5  5.73
##  6  2013     1     6  7.15
##  7  2013     1     7  5.42
##  8  2013     1     8  2.55
##  9  2013     1     9  2.28
## 10  2013     1    10  2.84
## # … with 355 more rows
```

By month

```r
by_month <- group_by(flights, month)
summarize(by_month, delay = mean(dep_delay, na.rm = TRUE))
```

```
## # A tibble: 12 × 2
##    month delay
##    <int> <dbl>
##  1     1 10.0
##  2     2 10.8
##  3     3 13.2
##  4     4 13.9
##  5     5 13.0
##  6     6 20.8
##  7     7 21.7
##  8     8 12.6
##  9     9  6.72
## 10    10  6.24
## 11    11  5.44
## 12    12 16.6
```

By year

```r
by_year <- group_by(flights, year)
summarize(by_year, delay = mean(dep_delay, na.rm = TRUE))
```

```
## # A tibble: 1 × 2
##    year delay
##   <int> <dbl>
## 1  2013  12.6
```

Counts

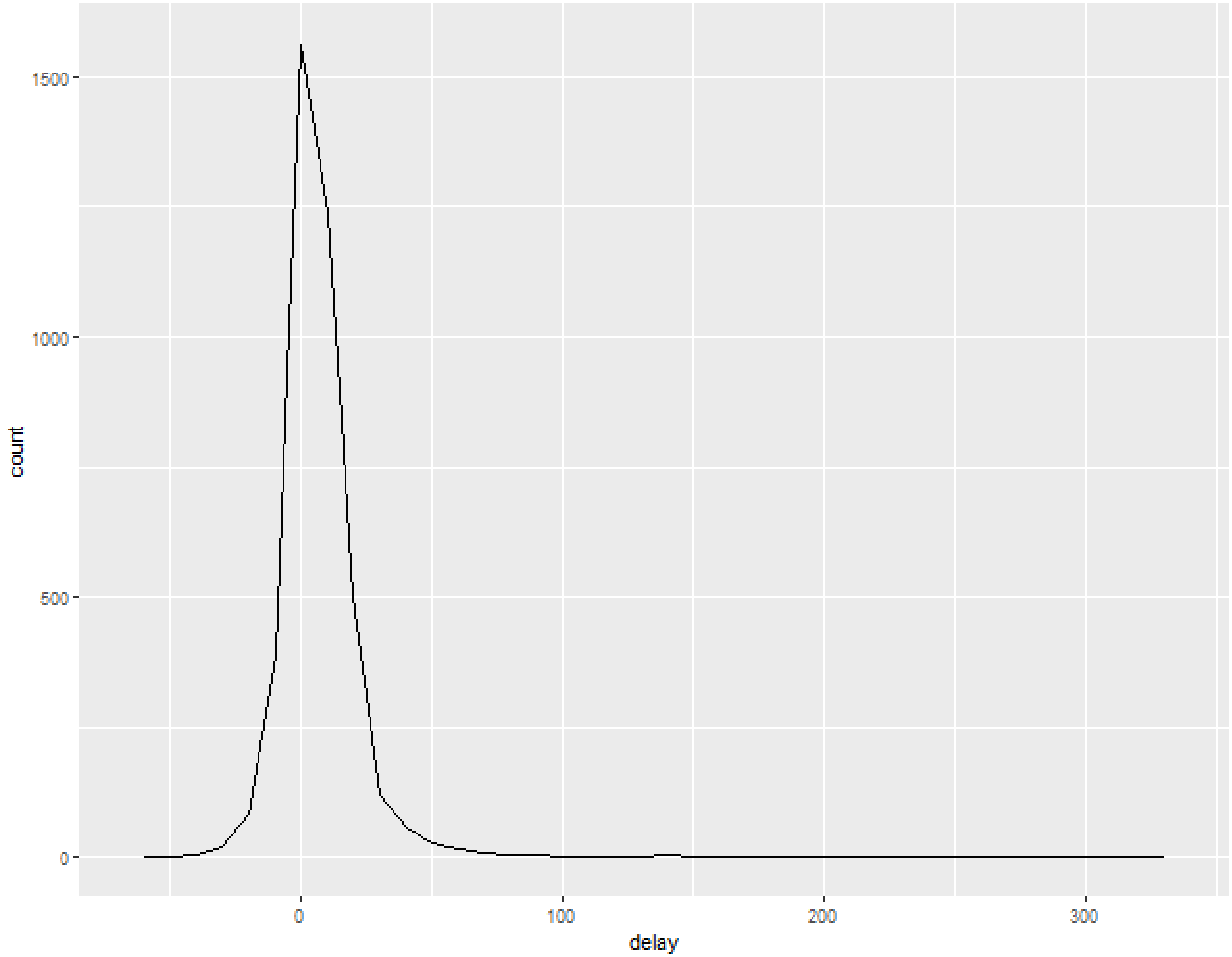
Whenever you do any aggregation, it's always a good idea to include either a count `n()`, or a count of nonmissing values `sum(!is.na(x))`.

```r
# Look at the planes (identified by their tail number) that have the
# highest average delays:
not_cancelled <- flights %>%
  filter(!is.na(dep_delay), !is.na(arr_delay))

delays <- not_cancelled %>%
  group_by(tailnum) %>%
  summarize(delay = mean(arr_delay))
delays
```

```
## # A tibble: 4,037 × 2
##    tailnum  delay
##    <chr>    <dbl>
##  1 D942DN  31.5
##  2 N0EGMQ   9.98
##  3 N10156  12.7
##  4 N102UW   2.94
##  5 N103US  -6.93
##  6 N104UW   1.80
```

Counts (cont'd)

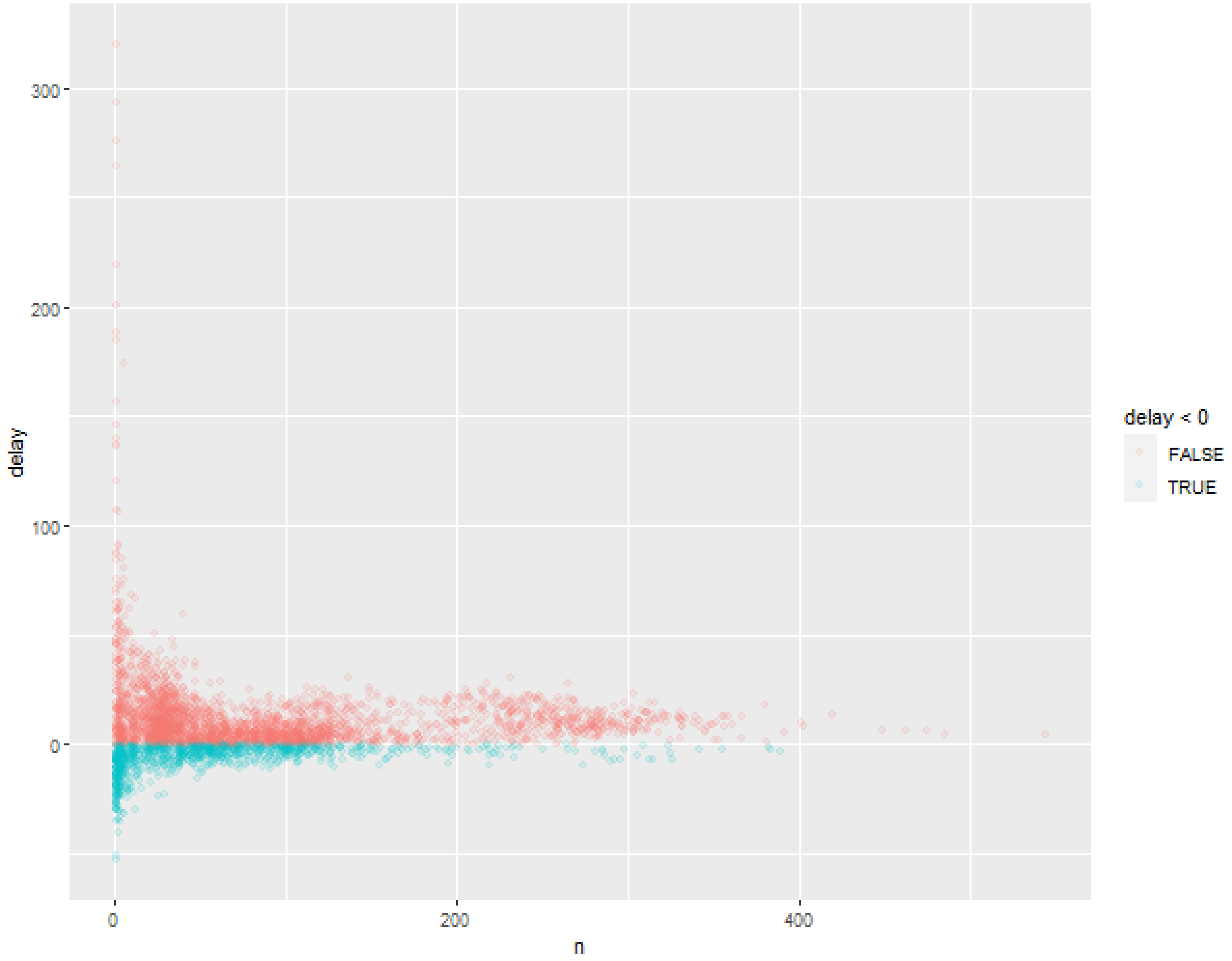


Counts (cont'd)

We can get more insight if we draw a scatterplot of number of flights versus average delay:

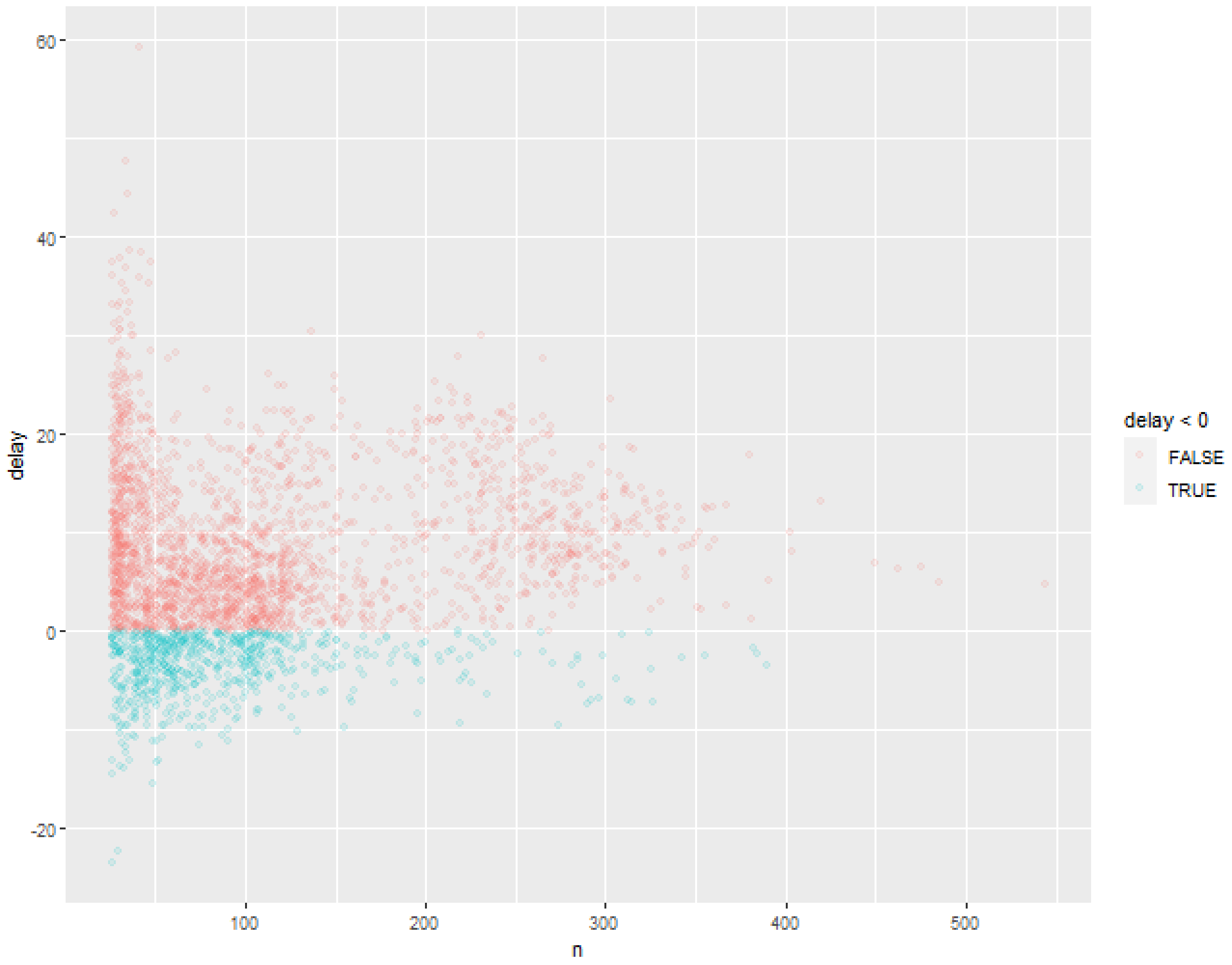
```
not_cancelled %>%
  group_by(tailnum) %>%
  summarize(delay = mean(arr_delay, na.rm = TRUE),
            n = n()) %>%
  ggplot(mapping = aes(x = n, y = delay, color = delay < 0)) +
  geom_point(alpha = 1 / 10)
```

Counts (cont'd)



Filter out all delay times less than and equals to 25 to see more details.

```r
not_cancelled %>%
  group_by(tailnum) %>%
  summarize(delay = mean(arr_delay, na.rm = TRUE),
            n = n()) %>%
  filter(n > 25) %>%
  ggplot(mapping = aes(x = n, y = delay,  color = delay < 0)) +
  geom_point(alpha = 1 / 10)
```


60
40
20
0
-20
delay
100
200
300
400
500
n
delay < 0
FALSE
TRUE

Useful Summary Functions

Measure of location: `mean()` and `median()`

```r
not_cancelled %>%
  group_by(year, month, day) %>%
  summarize(# average delay:
    avg_delay1 = mean(arr_delay),
    # average positive delay:
    avg_delay2 = mean(arr_delay[arr_delay > 0]))
```

```
## `summarise()` has grouped output by 'year', 'month'. You can override using the
## `.groups` argument.

## # A tibble: 365 × 5
## # Groups:   year, month [12]
##     year month   day avg_delay1 avg_delay2
##    <int> <int> <int>      <dbl>      <dbl>
##  1  2013     1     1     12.7         32.5
##  2  2013     1     2     12.7         32.0
##  3  2013     1     3      5.73        27.7
##  4  2013     1     4     -1.93        28.3
##  5  2013     1     5     -1.53        22.6
##  6  2013     1     6      4.24        24.4
```

Useful Summary Functions (cont'd)

Measure of spread `sd()`

Some destinations more variable than to others

```
not_cancelled %>%
  group_by(dest) %>%
  summarize(distance_sd = sd(distance)) %>%
  arrange(desc(distance_sd))
```

```
## # A tibble: 104 × 2
##    dest  distance_sd
##    <chr>       <dbl>
##  1 EGE         10.5
##  2 SAN         10.4
##  3 SFO         10.2
##  4 HNL         10.0
##  5 SEA          9.98
##  6 LAS          9.91
##  7 PDX          9.87
##  8 PHX          9.86
##  9 LAX          9.66
## 10 IND          9.46
## # … with 94 more rows
```

Useful Summary Functions (cont'd)

Measures of rank `min(x)`, `max(x)`

```
# When do the first and last flights leave each day?
not_cancelled %>%
  group_by(year, month, day) %>%
  summarize(first = min(dep_time),
            last = max(dep_time))
```

```
## `summarise()` has grouped output by 'year', 'month'. You can override using the
## `.groups` argument.

## # A tibble: 365 × 5
## # Groups:   year, month [12]
##     year month   day first  last
##    <int> <int> <int> <int> <int>
##  1  2013     1     1   517  2356
##  2  2013     1     2    42  2354
##  3  2013     1     3    32  2349
##  4  2013     1     4    25  2358
##  5  2013     1     5    14  2357
##  6  2013     1     6    16  2355
##  7  2013     1     7    49  2359
```

Useful Summary Functions (cont'd)

Measures of position `first(x)`, `nth(x, 2)`, `last(x)`

```
not_cancelled %>%
  group_by(year, month, day) %>%
  summarize(first_dep = first(dep_time),
            second_dep = nth(dep_time, 2))
```

```
## `summarise()` has grouped output by 'year', 'month'. You can override using the
## `.groups` argument.

## # A tibble: 365 × 5
## # Groups:   year, month [12]
##     year month   day first_dep second_dep
##    <int> <int> <int>     <int>      <int>
##  1  2013     1     1       517        533
##  2  2013     1     2        42        126
##  3  2013     1     3        32         50
##  4  2013     1     4        25        106
##  5  2013     1     5        14         37
##  6  2013     1     6        16        458
##  7  2013     1     7        49        454
##  8  2013     1     8       454        524
```

Useful Summary Functions (cont'd)

Distinct values, n_distinct(x)

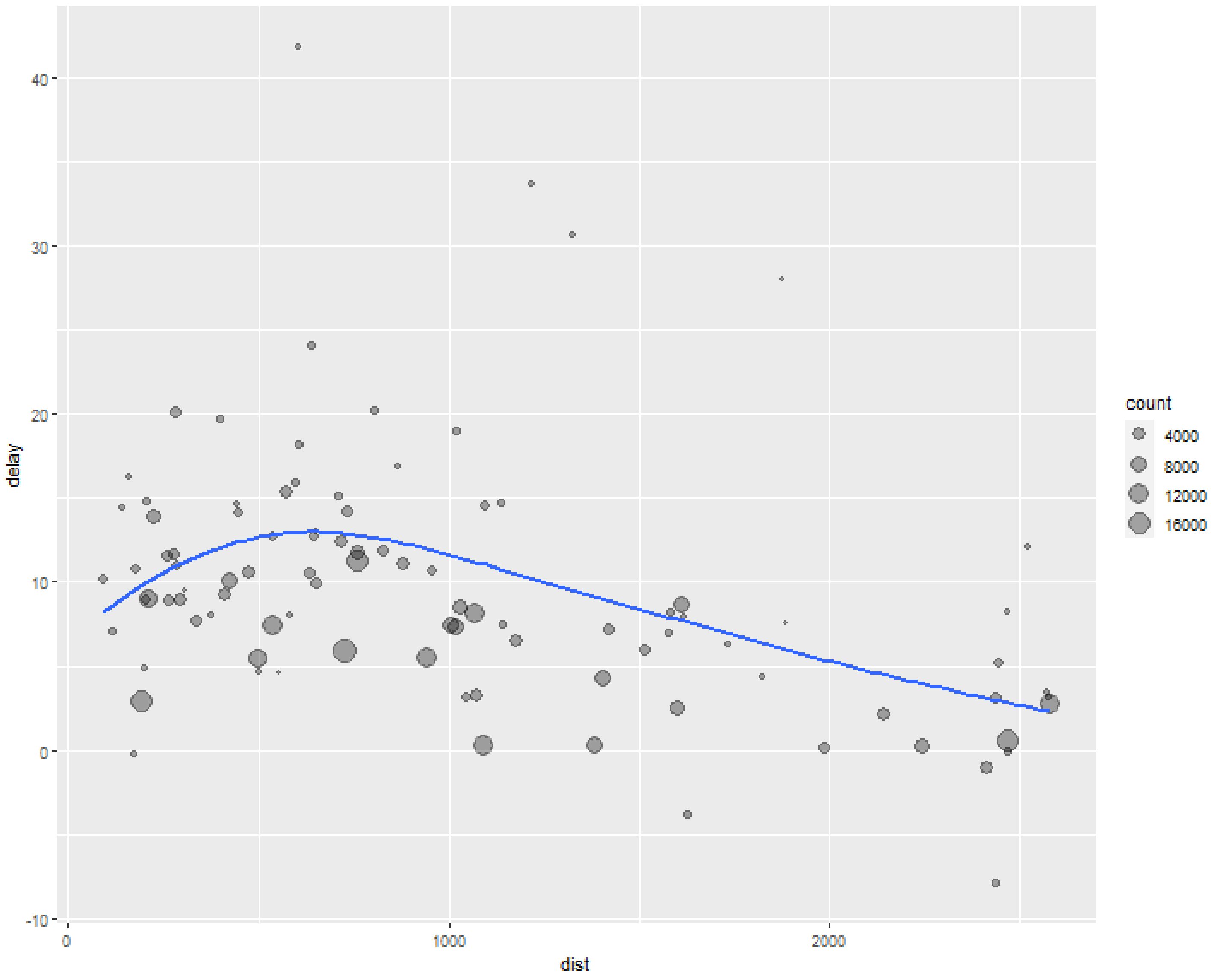
```
# Which destinations have the most carriers?
not_cancelled %>%
  group_by(dest) %>%
  summarize(carriers = n_distinct(carrier)) %>%
  arrange(desc(carriers))
```

```
## # A tibble: 104 × 2
##    dest  carriers
##    <chr>    <int>
##  1 ATL          7
##  2 BOS          7
##  3 CLT          7
##  4 ORD          7
##  5 TPA          7
##  6 AUS          6
##  7 DCA          6
##  8 DTW          6
##  9 IAD          6
## 10 MSP          6
## # … with 94 more rows
```

Combining Multiple Operations with the Pipe %>%

Imagine that we want to explore the relationship between the distance and average delay for each location.

```r
by_dest <- group_by(flights, dest)
delay <- summarize(
  by_dest,
  count = n(),
  dist = mean(distance, na.rm = TRUE),
  delay = mean(arr_delay, na.rm = TRUE)
)
delay <- filter(delay, count > 20, dest != "HNL")
```


40
30
20
10
0
-10
delay
0
1000
2000
dist
count
4000
8000
12000
16000

There are three steps to prepare this data:

- Group flights by destination.
- Summarize to compute distance, average delay, and number of flights.
- Filter to remove noisy points and Honolulu airport, which is almost twice as far away as the next closest airport.

Combining Multiple Operations with the Pipe %>%

There's another way to tackle the same problem with the pipe, %>%:

```r
flights %>%
  group_by(dest) %>%
  summarize(
    count = n(),
    dist = mean(distance, na.rm = TRUE),
    delay = mean(arr_delay, na.rm = TRUE)
  ) %>%
  filter(count > 20, dest != "HNL") %>%
  ggplot(mapping = aes(x = dist, y = delay)) +
  geom_point(aes(size = count), alpha = 1 / 3) +
  geom_smooth(se = FALSE)
```

Combining Multiple Operations with the Pipe %>%

There's another way to tackle the same problem with the pipe, %>%:

small data set: transmute() vs summarize()

```
a <-
  data.frame(
    c = c("111", "112", "111", "113"),
    a = c("a", "b", "b", "c"),
    b = 1:4
  )
a
```

```
##     c a b
## 1 111 a 1
## 2 112 b 2
## 3 111 b 3
## 4 113 c 4
```

transmute()

```
a %>%
  group_by(c) %>% transmute(a1 = n_distinct(a))
```

```
## # A tibble: 4 × 2
## # Groups:   c [3]
##   c        a1
##   <chr> <int>
## 1 111       2
## 2 112       1
## 3 111       2
## 4 113       1
```

summarize()

```
a %>%
  group_by(c) %>% summarize(a1 = n_distinct(a))
```

```
## # A tibble: 3 × 2
##   c        a1
##   <chr> <int>
## 1 111       2
## 2 112       1
## 3 113       1
```

Tips

You can use () for directly retrive the variabl For example:

```r
x <- 1:5
x
```

```
## [1] 1 2 3 4 5
```

```r
(x <- 1:5)
```

```
## [1] 1 2 3 4 5
```

Ungrouping

If you need to remove grouping, and return to operations on ungrouped data, use ungroup(): Notice: it's a common issue without removing grouping before another new grouping.

Without ungrouping

```
daily <- group_by(flights, year, month, day)
daily %>%
  group_by(month) %>%
  summarize(flights=n())
```

```
## # A tibble: 12 × 2
##    month flights
##    <int>   <int>
##  1     1   27004
##  2     2   24951
##  3     3   28834
##  4     4   28330
##  5     5   28796
##  6     6   28243
##  7     7   29425
##  8     8   29327
##  9     9   27574
## 10    10   28889
## 11    11   27268
## 12    12   28135
```

With ungrouping

```
daily %>%
  group_by(month) %>%
  ungroup() %>%
  summarize(flights=n())
```

```
## # A tibble: 1 × 1
##   flights
##     <int>
## 1  336776
```