R4ds Chapter 5 Data Transformation Yuqing Xue Personal Note

Yuqing Xue

October 22, 2017

The following personal code and notes are based on the materials from Hadley Wickham's 'R for Data Science', Chapter 5. http://r4ds.had.co.nz/transform.html#introduction-2

5.1 Introduction

nycflights13

```
########################################
####### Dataset: nycflights13::flights
########################################
library(nycflights13)
library(tidyverse)
nycflights13::flights
```

```
## # 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 <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 <dttm>
```

5.2 Filter

Explore, try %in%

```
jan1 <- filter(flights, month == 1, day == 1)

nov_and_dec = filter(flights, month == 11 | month == 12)
## A useful short-hand for this problem is x %in% y. This will select every row where x is one of the va
nov_dec <- filter(flights, month %in% c(11, 12))
```

```
NYCtoSF <-
  flights %>%
    filter(dest %in% c("SFO", "OAK", "SJC")) %>%
      select(year, month, day, carrier, origin, dest, air_time)
NYCtoSF
```

```
## # A tibble: 13,972 x 7
##     year month   day carrier origin  dest air_time
##    <int> <int> <int>   <chr>  <chr> <chr>    <dbl>
## 1   2013     1     1      UA    EWR   SFO      361
## 2   2013     1     1      UA    JFK   SFO      366
## 3   2013     1     1      DL    JFK   SFO      362
## 4   2013     1     1      VX    JFK   SFO      356
## 5   2013     1     1      B6    JFK   SFO      350
## 6   2013     1     1      AA    JFK   SFO      378
## 7   2013     1     1      UA    EWR   SFO      373
## 8   2013     1     1      UA    JFK   SFO      369
## 9   2013     1     1      UA    EWR   SFO      357
## 10  2013     1     1      AA    JFK   SFO      389
## # ... with 13,962 more rows
```

```
NYCtoSF2 <-
  flights %>%
    filter(dest == "SFO" | dest == "OAK" |  dest == "SJC") %>%
      select(year, month, day, carrier, origin, dest, air_time)
NYCtoSF2
```

```
## # A tibble: 13,972 x 7
##     year month   day carrier origin  dest air_time
##    <int> <int> <int>   <chr>  <chr> <chr>    <dbl>
## 1   2013     1     1      UA    EWR   SFO      361
```

```
## 2   2013     1     1      UA    JFK   SFO      366
## 3   2013     1     1      DL    JFK   SFO      362
## 4   2013     1     1      VX    JFK   SFO      356
## 5   2013     1     1      B6    JFK   SFO      350
## 6   2013     1     1      AA    JFK   SFO      378
## 7   2013     1     1      UA    EWR   SFO      373
## 8   2013     1     1      UA    JFK   SFO      369
## 9   2013     1     1      UA    EWR   SFO      357
## 10  2013     1     1      AA    JFK   SFO      389
## # ... with 13,962 more rows
```

```
# H.W. As well as & and |, R also has && and ||. Don't use them here! You'll learn when you should use
```

5.2.4 Exercises

1. Find all flights that

1.1 Had an arrival delay of two or more hours

1.2. Flew to Houston (IAH or HOU)

1.3. Were operated by United, American, or Delta

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

1.5. Arrived more than two hours late, but didn't leave late

1.6. Were delayed by at least an hour, but made up over 30 minutes in flight

1.7. Departed between midnight and 6am (inclusive)

```
Ex1.1 <-
  flights %>%
    filter(arr_delay >= 120) %>%
      arrange(desc(arr_delay)) %>%
        select(month, day, carrier, origin, dest, dep_delay, arr_delay)
Ex1.1
```

```
## # A tibble: 10,200 x 7
##    month   day carrier origin  dest dep_delay arr_delay
##    <int> <int>   <chr>  <chr> <chr>     <dbl>     <dbl>
## 1      1     9      HA    JFK   HNL      1301      1272
## 2      6    15      MQ    JFK   CMH      1137      1127
## 3      1    10      MQ    EWR   ORD      1126      1109
## 4      9    20      AA    JFK   SFO      1014      1007
## 5      7    22      MQ    JFK   CVG      1005       989
## 6      4    10      DL    JFK   TPA       960       931
## 7      3    17      DL    LGA   MSP       911       915
## 8      7    22      DL    LGA   ATL       898       895
## 9     12     5      AA    EWR   MIA       896       878
```

```
## 10     5     3      MQ    EWR   ORD       878       875
## # ... with 10,190 more rows
```

```
Ex1.2 = filter(flights, dest %in% c("IAH","HOU"))
Ex1.3 = filter(flights, carrier %in% c("UA","AA","DL"))
Ex1.4 = filter(flights, month %in% c(7,8,9))
```

```
Ex1.5 = filter(flights, arr_delay >= 120 & dep_delay <= 0 )
Ex1.6 = filter(flights, dep_delay >= 60 & dep_delay - arr_delay >= 30)
Ex1.7 = filter(flights, dep_time <= 600)
```

2. Another useful dplyr filtering helper is `between()`. What does it do? Can you use it to simplify the code needed to answer the previous challenges?

```
?between
Ex1.4b = filter(flights, between(month, 7, 9))
```

3. How many flights have a missing dep_time? What other variables are missing? What might these rows represent?

```
sum(is.na(flights$dep_time)) # 8255
```

```
## [1] 8255
```

```
flights_missing_dep_time <- flights %>% filter(is.na(dep_time))
flights_missing_dep_time
```

```
## # A tibble: 8,255 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       NA           1630        NA       NA
## 2   2013     1     1       NA           1935        NA       NA
## 3   2013     1     1       NA           1500        NA       NA
## 4   2013     1     1       NA            600        NA       NA
## 5   2013     1     2       NA           1540        NA       NA
## 6   2013     1     2       NA           1620        NA       NA
## 7   2013     1     2       NA           1355        NA       NA
## 8   2013     1     2       NA           1420        NA       NA
## 9   2013     1     2       NA           1321        NA       NA
## 10  2013     1     2       NA           1545        NA       NA
## # ... with 8,245 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 <dbl>,
## #   minute <dbl>, time_hour <dttm>
```

```
flights_cancelled <- flights %>%
  filter(is.na(dep_time) & is.na(dep_delay) & is.na(arr_time) & is.na(arr_delay) & is.na(arr_time))
# same 8255 flights
```

Beside `dep_time`, `dep_delay`, `arr_time`, `arr_delay` and `air_time` are also missing, for all 8255 flights. One possible explanation for it is that those flights were cancelled.

4. Why is `NA ^ 0` not missing? Why is `NA | TRUE` not missing? Why is `FALSE & NA` not missing? Can you figure out the general rule? (`NA * 0` is a tricky counterexample!)

`NA ^ 0` returns 1. Any number to the power of 0 is 1, whether the number is missing or not does not matter.

`NA | TRUE` returns TRUE, since the `|` operator returns TRUE if either of the terms are TRUE. In this case, the right half is TRUE, so the whole expression will always return TRUE.

`FALSE & NA` returns FALSE, because operator `&` returns TRUE when both terms are true. The left half is FALSE, so the whole expression returns FALSE despite the NA on the right half.

```
NA * 0
```

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

`NA * 0` returns 0, whicn may rendered the general rule we discovered from the previous not definite: if NA represent a value that is `Inf`, and we know `Inf` * 0 should not be a number, i.e., `NaN`.

5.3 Arrange

`arrange()` works similarly to `filter()` except that instead of selecting rows, it changes their order. It takes a data frame and a set of column names (or more complicated expressions) to order by. **If you provide more than one column name, each additional column will be used to break ties in the values of preceding columns**:

```
arrange(flights, year, month, 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 <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 <dttm>
```

```
arrange(flights, desc(arr_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     7    22     2257            759       898      121
## 9   2013    12     5      756           1700       896     1058
```

```
## 10  2013     5     3     1133           2055       878     1250
## # ... with 336,766 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 <dbl>,
## #   minute <dbl>, time_hour <dttm>
```

```
df <- tibble(x = c(5, 2, NA))
arrange(df, x)
```

```
## # A tibble: 3 x 1
##       x
##   <dbl>
## 1     2
## 2     5
## 3    NA
```

In this way, `x` then becomes the column name of the tibble `df`.

5.3.1 Exercise

1. How could you use `arrange()` to sort all missing values to the start? (Hint: use `is.na()`).

```
arrange(df, !is.na(x))
```

```
## # A tibble: 3 x 1
##       x
##   <dbl>
## 1    NA
## 2     5
## 3     2
```

It seems that R recognized `FALSE` as 0 and `TRUE` as 1, which makes `FALSE` is less than `TRUE`, hence can be sorted to the start use `arrange`.

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

```
colnames(flights)
```

```
##  [1] "year"           "month"          "day"            "dep_time"
##  [5] "sched_dep_time" "dep_delay"      "arr_time"       "sched_arr_time"
##  [9] "arr_delay"      "carrier"        "flight"         "tailnum"
## [13] "origin"         "dest"           "air_time"       "distance"
## [17] "hour"           "minute"         "time_hour"
```

```
flights %>%
  arrange(desc(arr_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     7    22     2257           759       898      121
## 9   2013    12     5      756          1700       896     1058
## 10  2013     5     3     1133          2055       878     1250
## # ... with 336,766 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 <dbl>,
## #   minute <dbl>, time_hour <dttm>
```

```
flights %>%
  arrange(dep_time)
```

```
## # 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    13        1           2249        72      108
## 2   2013     1    31        1           2100       181      124
## 3   2013    11    13        1           2359         2      442
## 4   2013    12    16        1           2359         2      447
## 5   2013    12    20        1           2359         2      430
## 6   2013    12    26        1           2359         2      437
## 7   2013    12    30        1           2359         2      441
## 8   2013     2    11        1           2100       181      111
## 9   2013     2    24        1           2245        76      121
## 10  2013     3     8        1           2355         6      431
## # ... with 336,766 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 <dbl>,
## #   minute <dbl>, time_hour <dttm>
```

which may lead to an idea to create a function:

idea: do all the common topRanking (smallest/largest, variable) and put it on Shiny

3. Sort `flights` to find the fastest flights.

time form needs to be clearer, answer not reliable

Idea: create a variable `travelTime` to measure the flight time (`arr_time - dep_time`. Noted that `dep_time` and `arr_time` are coded in a form that is intuitive to read and comprehend, but a little harder to make further calculation with, because they're not continuous. A more convenient representation of number of minutes since midnight are yet to be computed. So `travelTime` are neither real minutes nor hours and minutes like `dep_time` and `arr_time`. More details and discussion can be found in exersice 5.5.2 problem 2. http://r4ds.had.co.nz/transform.html#exercises-10).

Tricky: The way how `dep_time` and `arr_time` is coded needs a slightly modification.

```
flights %>%
  mutate(travelTime = ifelse(arr_time - dep_time < 0, arr_time + 2400 - dep_time, arr_time - dep_time))
    select(origin, dest, dep_time, sched_dep_time, arr_time, sched_arr_time, travelTime) %>%
      arrange(travelTime)
```

```
## # A tibble: 336,776 x 7
##    origin  dest dep_time sched_dep_time arr_time sched_arr_time travelTime
##     <chr> <chr>    <int>          <int>    <int>          <int>      <dbl>
## 1     EWR   BDL     1323           1325     1358           1421         35
## 2     EWR   BDL     1312           1316     1347           1413         35
```

```
## 3     EWR   BDL     1203           1153     1238           1250         35
## 4     EWR   BDL      722            730      758            830         36
## 5     EWR   BDL      722            729      758            830         36
## 6     EWR   BDL      718            725      754            822         36
## 7     EWR   BDL     1418           1329     1455           1426         37
## 8     EWR   BDL       16           2159       53           2304         37
## 9     EWR   BDL      717            725      754            822         37
## 10    EWR   BDL     1315           1320     1353           1419         38
## # ... with 336,766 more rows
```

```
# and the slowest flights
flights %>%
  mutate(travelTime = ifelse(arr_time - dep_time < 0, arr_time + 2400 - dep_time, arr_time - dep_time)) %>%
    select(origin, dest, dep_time, sched_dep_time, arr_time, sched_arr_time, travelTime) %>%
      arrange(desc(travelTime))
```

```
## # A tibble: 336,776 x 7
##    origin  dest dep_time sched_dep_time arr_time sched_arr_time travelTime
##     <chr> <chr>    <int>          <int>    <int>          <int>      <dbl>
## 1     EWR   ORD     1623           1545     1148           1710       1925
## 2     EWR   DFW      959            920     2129           1240       1170
## 3     LGA   ATL     1500           1459      245           1737       1145
## 4     EWR   LAX     1259           1300     2358           1555       1099
## 5     JFK   BOS     2313           2050     1003           2203       1090
## 6     EWR   GSP      653            659     1704            857       1051
## 7     JFK   LAX      625            630     1635            922       1010
## 8     LGA   ATL     1415           1355       18           1617       1003
## 9     LGA   DEN      755            800     1737           1025        982
## 10    JFK   SLC      644            655     1624           1030        980
## # ... with 336,766 more rows
```

```
flights_Testimate <- flights %>%
  mutate(travelTime = ifelse(arr_time - dep_time < 0, arr_time + 2400 - dep_time, arr_time - dep_time))
```

The most unexpected flights, like the one from EWR to ORD that took 19 hours 25 minutes, can be spotted and further investigated.

4. Which flights travelled the longest? Which travelled the shortest?

Idea: sort variable `distance`

```
# The longest flights
flights_Testimate %>%
  select(carrier:distance, travelTime) %>%
    arrange(desc(distance))
```

```
## # A tibble: 336,776 x 8
##    carrier flight tailnum origin  dest air_time distance travelTime
##      <chr>  <int>   <chr>  <chr> <chr>    <dbl>    <dbl>      <dbl>
## 1       HA     51  N380HA    JFK   HNL      659     4983        659
## 2       HA     51  N380HA    JFK   HNL      638     4983        616
## 3       HA     51  N380HA    JFK   HNL      616     4983        590
## 4       HA     51  N384HA    JFK   HNL      639     4983        616
## 5       HA     51  N381HA    JFK   HNL      635     4983        661
## 6       HA     51  N385HA    JFK   HNL      611     4983        539
## 7       HA     51  N385HA    JFK   HNL      612     4983        578
```

```
## 8       HA     51  N389HA    JFK   HNL       645     4983        603
## 9       HA     51  N384HA    JFK   HNL       640     4983        601
## 10      HA     51  N388HA    JFK   HNL       633     4983        590
## # ... with 336,766 more rows
```

```
# The longest flight where destination is not HNL
flights_Testimate %>%
  select(carrier:distance, travelTime) %>%
    filter(dest != 'HNL') %>%
      arrange(desc(distance))
```

```
## # A tibble: 336,069 x 8
##    carrier flight tailnum origin  dest air_time distance travelTime
##      <chr>  <int>   <chr>  <chr> <chr>    <dbl>    <dbl>      <dbl>
## 1       UA    887  N587UA    EWR   ANC      418     3370        325
## 2       UA    887  N572UA    EWR   ANC      404     3370        337
## 3       UA    887  N567UA    EWR   ANC      418     3370        385
## 4       UA    887  N559UA    EWR   ANC      388     3370        289
## 5       UA    887  N572UA    EWR   ANC      434     3370        388
## 6       UA    887  N559UA    EWR   ANC      411     3370        309
## 7       UA    887  N528UA    EWR   ANC      404     3370        302
## 8       UA    887  N534UA    EWR   ANC      428     3370        326
## 9       UA    303  N532UA    JFK   SFO      366     2586        334
## 10      DL   1865  N705TW    JFK   SFO      362     2586        382
## # ... with 336,059 more rows
```

```
# Besides HNL and ANC
flights_Testimate %>%
  select(carrier:distance, travelTime) %>%
    filter(dest != 'HNL' & dest != "ANC") %>%
      arrange(desc(distance))
```

```
## # A tibble: 336,061 x 8
##    carrier flight tailnum origin  dest air_time distance travelTime
##      <chr>  <int>   <chr>  <chr> <chr>    <dbl>    <dbl>      <dbl>
## 1       UA    303  N532UA    JFK   SFO      366     2586        334
## 2       DL   1865  N705TW    JFK   SFO      362     2586        382
## 3       VX     11  N635VA    JFK   SFO      356     2586        320
## 4       B6    643  N625JB    JFK   SFO      350     2586        313
## 5       AA     59  N336AA    JFK   SFO      378     2586        390
## 6       UA    223  N510UA    JFK   SFO      369     2586        329
## 7       AA    179  N325AA    JFK   SFO      389     2586        398
## 8       VX     23  N625VA    JFK   SFO      363     2586        322
## 9       UA    285  N517UA    JFK   SFO      364     2586        328
## 10      B6    641  N590JB    JFK   SFO      349     2586        311
## # ... with 336,051 more rows
```

```
# Besides HNL, ANC and JFK to SFO
flights_Testimate %>%
  select(carrier:distance, travelTime) %>%
    filter(dest != 'HNL' & dest != "ANC" & dest != "SFO") %>%
      arrange(desc(distance))
```

```
## # A tibble: 322,730 x 8
##    carrier flight tailnum origin  dest air_time distance travelTime
##      <chr>  <int>   <chr>  <chr> <chr>    <dbl>    <dbl>      <dbl>
```

```
## 1       B6     91  N523JB    JFK   OAK      330     2576        312
## 2       B6     91  N646JB    JFK   OAK      328     2576        317
## 3       B6     91  N636JB    JFK   OAK      319     2576        278
## 4       B6     91  N524JB    JFK   OAK      351     2576        313
## 5       B6     91  N659JB    JFK   OAK      351     2576        320
## 6       B6     91  N784JB    JFK   OAK      341     2576        361
## 7       B6     91  N517JB    JFK   OAK      345     2576        358
## 8       B6     91  N656JB    JFK   OAK      353     2576        313
## 9       B6     91  N563JB    JFK   OAK      364     2576        333
## 10      B6     91  N705JB    JFK   OAK      346     2576        365
## # ... with 322,720 more rows
```

The second longest flight is from EWR to ANC. The third one is JFK to SFO, and the forth longest flight within United States is from JFK to OAK.

```
# The shortest flights
flights_Testimate %>%
  select(carrier:distance, travelTime) %>%
    arrange(distance)
```

```
## # A tibble: 336,776 x 8
##    carrier flight tailnum origin  dest air_time distance travelTime
##      <chr>  <int>   <chr>  <chr> <chr>    <dbl>    <dbl>      <dbl>
## 1       US   1632    <NA>    EWR   LGA       NA       17         NA
## 2       EV   3833  N13989    EWR   PHL       30       80         95
## 3       EV   4193  N14972    EWR   PHL       30       80         93
## 4       EV   4502  N15983    EWR   PHL       28       80        108
## 5       EV   4645  N27962    EWR   PHL       32       80         90
## 6       EV   4193  N14902    EWR   PHL       29       80         86
## 7       EV   4619  N22909    EWR   PHL       22       80         99
## 8       EV   4619  N33182    EWR   PHL       25       80         88
## 9       EV   4619  N11194    EWR   PHL       30       80        177
## 10      EV   4619  N17560    EWR   PHL       27       80         91
## # ... with 336,766 more rows
```

```
flights_Testimate %>%
  filter(distance == 17)
```

```
## # A tibble: 1 x 20
##    year month   day dep_time sched_dep_time dep_delay arr_time
##   <int> <int> <int>    <int>          <int>     <dbl>    <int>
## 1  2013     7    27       NA            106        NA       NA
## # ... with 13 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 <dttm>, travelTime <dbl>
```

5.4 Select

`select()` can be used to rename variables, but it's rarely useful because it drops all of the variables not explicitly mentioned. Instead, use `rename()`, which is a variant of `select()` that keeps all the variables that aren't explicitly mentioned

```
# change tailnum to tail_num
```

```
flights %>%
  rename(tail_num = tailnum)
```

```
## # 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 <int>,
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tail_num <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>
```

Nothing changes the original dataset `flights_Testimate`, which still has that variable named `tailnum`.

Another option is to use `select()` in conjunction with the `everything()` helper. This is useful if you have a handful of variables you'd like to move to the start of the data frame.

Move some main information to the start of the dataframe and keep the rest:

```
flights %>%
  select(carrier:dest, everything())
```

```
## # A tibble: 336,776 x 19
##    carrier flight tailnum origin  dest  year month   day dep_time
##      <chr>  <int>   <chr>  <chr> <chr> <int> <int> <int>    <int>
## 1       UA   1545  N14228    EWR   IAH  2013     1     1      517
## 2       UA   1714  N24211    LGA   IAH  2013     1     1      533
## 3       AA   1141  N619AA    JFK   MIA  2013     1     1      542
## 4       B6    725  N804JB    JFK   BQN  2013     1     1      544
## 5       DL    461  N668DN    LGA   ATL  2013     1     1      554
## 6       UA   1696  N39463    EWR   ORD  2013     1     1      554
## 7       B6    507  N516JB    EWR   FLL  2013     1     1      555
## 8       EV   5708  N829AS    LGA   IAD  2013     1     1      557
## 9       B6     79  N593JB    JFK   MCO  2013     1     1      557
## 10      AA    301  N3ALAA    LGA   ORD  2013     1     1      558
## # ... with 336,766 more rows, and 10 more variables: sched_dep_time <int>,
## #   dep_delay <dbl>, arr_time <int>, sched_arr_time <int>,
## #   arr_delay <dbl>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>
```

5.4.1 Exercise

1.Brainstorm as many ways as possible to select `dep_time`, `dep_delay`, `arr_time`, and `arr_delay` from `flights`.

1.0. Just list all the variables to select

```
flights %>%
  select(dep_time, dep_delay, arr_time, arr_delay)
```

```
## # A tibble: 336,776 x 4
##    dep_time dep_delay arr_time arr_delay
##       <int>     <dbl>    <int>     <dbl>
## 1       517         2      830        11
## 2       533         4      850        20
## 3       542         2      923        33
## 4       544        -1     1004       -18
## 5       554        -6      812       -25
## 6       554        -4      740        12
## 7       555        -5      913        19
## 8       557        -3      709       -14
## 9       557        -3      838        -8
## 10      558        -2      753         8
## # ... with 336,766 more rows
```

1.1. starts_with and ends_with

```
flights %>%
  select(starts_with("dep"), starts_with("arr"))
```

```
## # A tibble: 336,776 x 4
##    dep_time dep_delay arr_time arr_delay
##       <int>     <dbl>    <int>     <dbl>
## 1       517         2      830        11
## 2       533         4      850        20
## 3       542         2      923        33
## 4       544        -1     1004       -18
## 5       554        -6      812       -25
## 6       554        -4      740        12
## 7       555        -5      913        19
## 8       557        -3      709       -14
## 9       557        -3      838        -8
## 10      558        -2      753         8
## # ... with 336,766 more rows
```

```
flights %>%
  select(ends_with("time"), ends_with("delay")) %>%
    select(-c(starts_with("sched"), starts_with("air")))
```

```
## # A tibble: 336,776 x 4
##    dep_time arr_time dep_delay arr_delay
##       <int>    <int>     <dbl>     <dbl>
## 1       517      830         2        11
## 2       533      850         4        20
## 3       542      923         2        33
## 4       544     1004        -1       -18
## 5       554      812        -6       -25
## 6       554      740        -4        12
## 7       555      913        -5        19
## 8       557      709        -3       -14
## 9       557      838        -3        -8
## 10      558      753        -2         8
## # ... with 336,766 more rows
```

1.2. Fancier: `contains`

```
flights %>% select(dep_time:arr_delay, -c(contains("sched")))
```

```
## # A tibble: 336,776 x 4
##    dep_time dep_delay arr_time arr_delay
##       <int>     <dbl>    <int>     <dbl>
## 1       517         2      830        11
## 2       533         4      850        20
## 3       542         2      923        33
## 4       544        -1     1004       -18
## 5       554        -6      812       -25
## 6       554        -4      740        12
## 7       555        -5      913        19
## 8       557        -3      709       -14
## 9       557        -3      838        -8
## 10      558        -2      753         8
## # ... with 336,766 more rows
```

1.3. Start learning `matches` syntax

```
flights %>% select(matches("^dep|^arr"))
```

```
## # A tibble: 336,776 x 4
##    dep_time dep_delay arr_time arr_delay
##       <int>     <dbl>    <int>     <dbl>
## 1       517         2      830        11
## 2       533         4      850        20
## 3       542         2      923        33
## 4       544        -1     1004       -18
## 5       554        -6      812       -25
## 6       554        -4      740        12
## 7       555        -5      913        19
## 8       557        -3      709       -14
## 9       557        -3      838        -8
## 10      558        -2      753         8
## # ... with 336,766 more rows
```

2. What happens if you include the name of a variable multiple times in a `select()` call?

```
flights %>% select(dep_time, dep_time, dep_time)
```

```
## # A tibble: 336,776 x 1
##    dep_time
##       <int>
## 1       517
## 2       533
## 3       542
## 4       544
## 5       554
## 6       554
## 7       555
## 8       557
## 9       557
## 10      558
## # ... with 336,766 more rows
```

Just select the variable once despite multiple select call.

3. What does the one_of() function do? Why might it be helpful in conjunction with this vector? `vars <- c("year", "month", "day", "dep_delay", "arr_delay")`

```
vars <- c("year", "month", "day", "dep_delay", "arr_delay")
flights %>% select(one_of(vars))
```

```
## # 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
```

4. Does the result of running the following code surprise you? How do the select helpers deal with case by default? How can you change that default?

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

```
## # A tibble: 336,776 x 6
##    dep_time sched_dep_time arr_time sched_arr_time air_time
##       <int>          <int>    <int>          <int>    <dbl>
## 1       517            515      830            819      227
## 2       533            529      850            830      227
## 3       542            540      923            850      160
## 4       544            545     1004           1022      183
## 5       554            600      812            837      116
## 6       554            558      740            728      150
## 7       555            600      913            854      158
## 8       557            600      709            723       53
## 9       557            600      838            846      140
## 10      558            600      753            745      138
## # ... with 336,766 more rows, and 1 more variables: time_hour <dttm>
```

```
select(flights, contains("TIME", ignore.case = FALSE))
```

```
## # A tibble: 336,776 x 0
```

`contains` does not treat argument with case sensitive by default. Use `ignore.case = FALSE` to cange that setting.

5.5 Add new variables with `mutate()`

```
flights_sml <- select(flights,
  year:day,
```

```
  ends_with("delay"),
  distance,
  air_time
)
mutate(flights_sml,
  gain = arr_delay - dep_delay,
  speed = distance / air_time * 60
)
```

```
## # A tibble: 336,776 x 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     9 370.0441
## 2   2013     1     1         4        20     1416      227    16 374.2731
## 3   2013     1     1         2        33     1089      160    31 408.3750
## 4   2013     1     1        -1       -18     1576      183   -17 516.7213
## 5   2013     1     1        -6       -25      762      116   -19 394.1379
## 6   2013     1     1        -4        12      719      150    16 287.6000
## 7   2013     1     1        -5        19     1065      158    24 404.4304
## 8   2013     1     1        -3       -14      229       53   -11 259.2453
## 9   2013     1     1        -3        -8      944      140    -5 404.5714
## 10  2013     1     1        -2         8      733      138    10 318.6957
## # ... with 336,766 more rows
```

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

```
transmute(flights,
  gain = arr_delay - dep_delay,
  hours = air_time / 60,
  gain_per_hour = gain / hours  # minutes gain per airtime hour
)
```

```
## # A tibble: 336,776 x 3
##     gain     hours gain_per_hour
##    <dbl>     <dbl>         <dbl>
## 1      9 3.7833333      2.378855
## 2     16 3.7833333      4.229075
## 3     31 2.6666667     11.625000
## 4    -17 3.0500000     -5.573770
## 5    -19 1.9333333     -9.827586
## 6     16 2.5000000      6.400000
## 7     24 2.6333333      9.113924
## 8    -11 0.8833333    -12.452830
## 9     -5 2.3333333     -2.142857
## 10    10 2.3000000      4.347826
## # ... with 336,766 more rows
```

5.5.1 Useful creation functions

The key property is that the function `mutate` must be vectorised: it must take a vector of values as input, return a vector with the same number of values as output.

If one parameter is shorter than the other, it will be automatically extended to be the same length. This is most useful when one of the arguments is a single number: air_time / 60, hours * 60 + minute, etc.

Ranking: The default gives smallest values the small ranks; use `desc(x)` to give the largest values the smallest ranks.

```
y <- c(70, 10, 10, NA, 30, 40)
min_rank(y)
```

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

```
min_rank(desc(y))  # order by value, more common
```

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

5.5.2 Exercises

1. Currently `dep_time` and `sched_dep_time` are convenient to look at, but hard to compute with because they're not really continuous numbers. Convert them to a more convenient representation of number of minutes since midnight.

Modular arithmetic: `%/%` (integer division) and `%%` (remainder), where `x == y * (x %/% y) + (x %% y)`

```
flights_5.5.2.1 = flights_Testimate %>%
    mutate(dep_time_totalMinutes = dep_time %/% 100 * 60 + dep_time %% 100, sched_dep_time_totalMinutes
    select(year:dep_delay,dep_time_totalMinutes,sched_dep_time_totalMinutes,arr_time:time_hour )
```

Use `select` to rearrange column orders.

2. Compare `air_time` with `arr_time - dep_time`. What do you expect to see? What do you see? What do you need to do to fix it?

```
flights_Testimate %>%
  mutate(arr_dep = arr_time - dep_time) %>%
    select(carrier:air_time, arr_dep)
```

```
## # A tibble: 336,776 x 7
##    carrier flight tailnum origin  dest air_time arr_dep
##      <chr>  <int>   <chr>  <chr> <chr>    <dbl>   <int>
## 1       UA   1545  N14228    EWR   IAH      227     313
## 2       UA   1714  N24211    LGA   IAH      227     317
## 3       AA   1141  N619AA    JFK   MIA      160     381
## 4       B6    725  N804JB    JFK   BQN      183     460
## 5       DL    461  N668DN    LGA   ATL      116     258
## 6       UA   1696  N39463    EWR   ORD      150     186
## 7       B6    507  N516JB    EWR   FLL      158     358
## 8       EV   5708  N829AS    LGA   IAD       53     152
## 9       B6     79  N593JB    JFK   MCO      140     281
## 10      AA    301  N3ALAA    LGA   ORD      138     195
## # ... with 336,766 more rows
```

We expect to see `air_time` equals `arr_time - dep_time`, however they are not remotely equal, and they shouldn't be, because as mentioned in problem 1, `dep_time` and `arr_time` (and their corresponding scheduked version) are not recorded in a manner that easy to add or substract. So a more convenient version of representation is preferred.

```
flights_Testimate = flights_Testimate %>%
  mutate(dep_time_totalMinutes = dep_time %/% 100 * 60 + dep_time %% 100,
         sched_dep_time_totalMinutes = sched_dep_time %/% 100 * 60 + sched_dep_time %% 100,
         arr_time_totalMinutes = arr_time %/% 100 * 60 + arr_time %% 100,
         sched_arr_time_totalMinutes = sched_arr_time %/% 100 * 60 + sched_arr_time %% 100) %>%
```

```
   select(year:arr_delay,dep_time_totalMinutes, sched_dep_time_totalMinutes, arr_time_totalMinutes,
          sched_arr_time_totalMinutes, air_time, carrier:travelTime)
```

```
flights_Testimate %>%
  mutate(air_time_minutes = arr_time_totalMinutes - dep_time_totalMinutes) %>%
    select(carrier:distance, sched_dep_time_totalMinutes, sched_arr_time_totalMinutes, dep_time_totalMi
```

```
## # A tibble: 336,776 x 12
##    carrier flight tailnum origin  dest distance
##      <chr>  <int>   <chr>  <chr> <chr>    <dbl>
## 1       UA   1545  N14228    EWR   IAH     1400
## 2       UA   1714  N24211    LGA   IAH     1416
## 3       AA   1141  N619AA    JFK   MIA     1089
## 4       B6    725  N804JB    JFK   BQN     1576
## 5       DL    461  N668DN    LGA   ATL      762
## 6       UA   1696  N39463    EWR   ORD      719
## 7       B6    507  N516JB    EWR   FLL     1065
## 8       EV   5708  N829AS    LGA   IAD      229
## 9       B6     79  N593JB    JFK   MCO      944
## 10      AA    301  N3ALAA    LGA   ORD      733
## # ... with 336,766 more rows, and 6 more variables:
## #   sched_dep_time_totalMinutes <dbl>, sched_arr_time_totalMinutes <dbl>,
## #   dep_time_totalMinutes <dbl>, arr_time_totalMinutes <dbl>,
## #   air_time_minutes <dbl>, air_time <dbl>
```

3. Compare `dep_time`, `sched_dep_time`, and `dep_delay`. How would you expect those three numbers to be related?

```
flights_Testimate %>%
  mutate(dep_delay_copy = dep_time - sched_dep_time) %>%
    select(dep_time, sched_dep_time, dep_delay_copy, dep_delay)
```

```
## # A tibble: 336,776 x 4
##    dep_time sched_dep_time dep_delay_copy dep_delay
##       <int>          <int>          <int>     <dbl>
## 1       517            515              2         2
## 2       533            529              4         4
## 3       542            540              2         2
## 4       544            545             -1        -1
## 5       554            600            -46        -6
## 6       554            558             -4        -4
## 7       555            600            -45        -5
## 8       557            600            -43        -3
## 9       557            600            -43        -3
## 10      558            600            -42        -2
## # ... with 336,766 more rows
```

```
flights_Testimate %>%
  mutate(dep_delay_copy = dep_time_totalMinutes - sched_dep_time_totalMinutes) %>%
    select(dep_time, sched_dep_time, dep_delay_copy, dep_delay)
```

```
## # A tibble: 336,776 x 4
##    dep_time sched_dep_time dep_delay_copy dep_delay
##       <int>          <int>          <dbl>     <dbl>
## 1       517            515              2         2
## 2       533            529              4         4
```

```
## 3       542            540             2         2
## 4       544            545            -1        -1
## 5       554            600            -6        -6
## 6       554            558            -4        -4
## 7       555            600            -5        -5
## 8       557            600            -3        -3
## 9       557            600            -3        -3
## 10      558            600            -2        -2
## # ... with 336,766 more rows
```

4. Find the 10 most delayed flights using a ranking function. How do you want to handle ties? Carefully read the documentation for `min_rank()`.

My first answer: arr_delay

```
flights_Testimate %>%
  mutate(delay_rank = min_rank(desc(arr_delay))) %>%
    select(carrier:dest, arr_delay, delay_rank) %>%
      arrange(delay_rank)
```

```
## # A tibble: 336,776 x 7
##    carrier flight tailnum origin  dest arr_delay delay_rank
##      <chr>  <int>   <chr>  <chr> <chr>     <dbl>      <int>
## 1       HA     51  N384HA    JFK   HNL      1272          1
## 2       MQ   3535  N504MQ    JFK   CMH      1127          2
## 3       MQ   3695  N517MQ    EWR   ORD      1109          3
## 4       AA    177  N338AA    JFK   SFO      1007          4
## 5       MQ   3075  N665MQ    JFK   CVG       989          5
## 6       DL   2391  N959DL    JFK   TPA       931          6
## 7       DL   2119  N927DA    LGA   MSP       915          7
## 8       DL   2047  N6716C    LGA   ATL       895          8
## 9       AA    172  N5DMAA    EWR   MIA       878          9
## 10      MQ   3744  N523MQ    EWR   ORD       875         10
## # ... with 336,766 more rows
```

Improved version (without having to create a new variable)

```
flights_Testimate %>%
  filter(min_rank(desc(arr_delay)) <= 10) %>%
    select(carrier:dest, arr_delay) %>%
      arrange(desc(arr_delay))
```

```
## # A tibble: 10 x 6
##    carrier flight tailnum origin  dest arr_delay
##      <chr>  <int>   <chr>  <chr> <chr>     <dbl>
## 1       HA     51  N384HA    JFK   HNL      1272
## 2       MQ   3535  N504MQ    JFK   CMH      1127
## 3       MQ   3695  N517MQ    EWR   ORD      1109
## 4       AA    177  N338AA    JFK   SFO      1007
## 5       MQ   3075  N665MQ    JFK   CVG       989
## 6       DL   2391  N959DL    JFK   TPA       931
## 7       DL   2119  N927DA    LGA   MSP       915
## 8       DL   2047  N6716C    LGA   ATL       895
## 9       AA    172  N5DMAA    EWR   MIA       878
## 10      MQ   3744  N523MQ    EWR   ORD       875
```

5. What does 1:3 + 1:10 return? Why?

```
1:3 + 1:10
```

```
## Warning in 1:3 + 1:10: longer object length is not a multiple of shorter
## object length
```

```
##  [1]  2  4  6  5  7  9  8 10 12 11
```

It returns a 10 dimensional vector along with a warning message. Looks like the shorter vector 1:3 is repeated out to the length of the longer 1:10 one.

5.6 Grouped summaries with `summarise`

It collapses a data frame to a single row.

`summarise()` is not terribly useful unless we pair it with `group_by()`

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

```
## Source: local data frame [365 x 4]
## Groups: year, month [?]
##
##     year month   day     delay
##    <int> <int> <int>     <dbl>
## 1   2013     1     1 11.548926
## 2   2013     1     2 13.858824
## 3   2013     1     3 10.987832
## 4   2013     1     4  8.951595
## 5   2013     1     5  5.732218
## 6   2013     1     6  7.148014
## 7   2013     1     7  5.417204
## 8   2013     1     8  2.553073
## 9   2013     1     9  2.276477
## 10  2013     1    10  2.844995
## # ... with 355 more rows
```

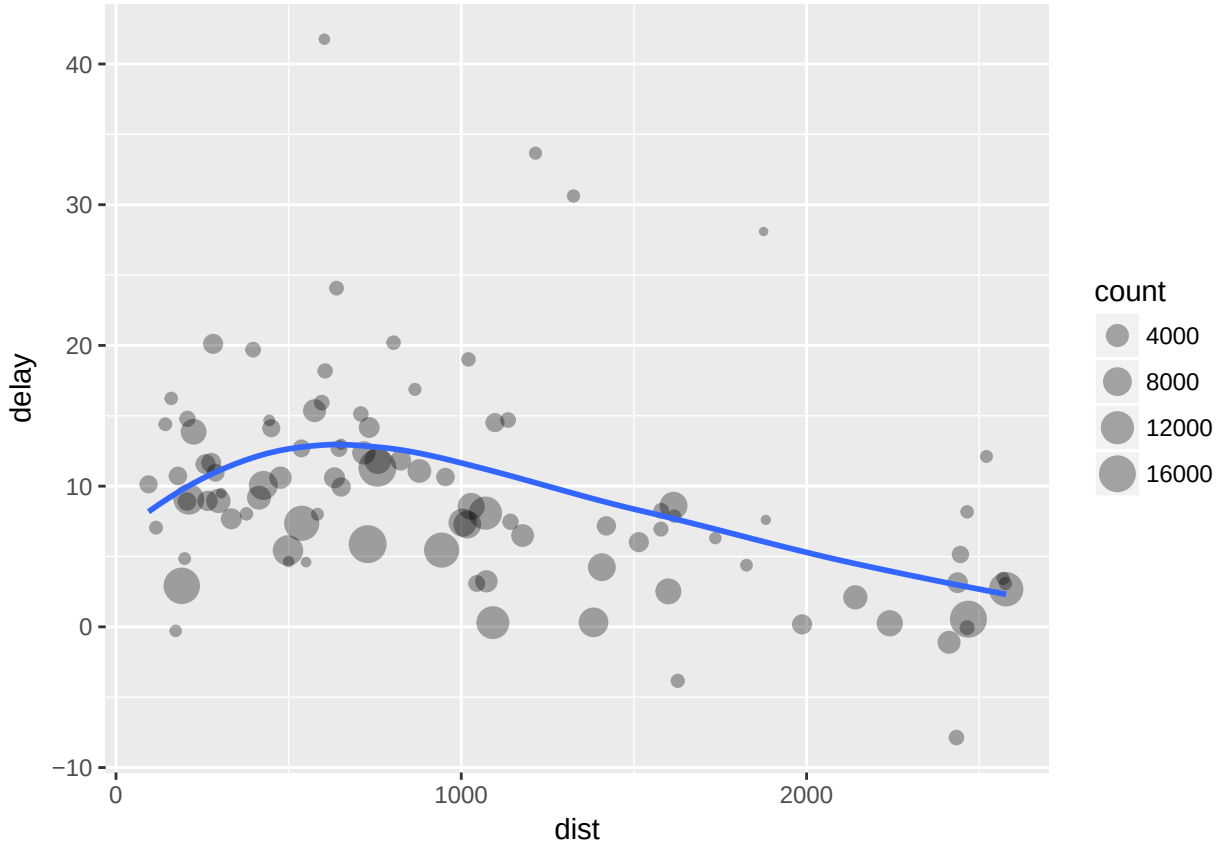
5.6.1 Combining multiple operations with the pipe

Imagine that we want to explore the relationship between the distance and average delay for each location. Using what you know about dplyr, you might write code like this:

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

# It looks like delays increase with distance up to ~750 miles and then decrease. Maybe as flights get
ggplot(data = delays, mapping = aes(x = dist, y = delay)) +
```

```
  geom_point(aes(size = count), alpha = 1/3) +
  geom_smooth(se = FALSE)
```

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




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

There are three steps to prepare this data:

Group flights by destination.

Summarise 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.

This code is a little frustrating to write because we have to give each intermediate data frame a name, even though we don't care about it. Naming things is hard, so this slows down our analysis.

5.6.2 Missing Values

Naive failure try:

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

```
## Source: local data frame [365 x 4]
## Groups: year, month [?]
##
```

```
##     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
## 9   2013     1     9    NA
## 10  2013     1    10    NA
## # ... with 355 more rows
```

All aggregation functions have an `na.rm` argument which removes the missing values prior to computation:

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

```
## Source: local data frame [365 x 4]
## Groups: year, month [?]
##
##     year month   day      mean
##    <int> <int> <int>     <dbl>
## 1   2013     1     1 11.548926
## 2   2013     1     2 13.858824
## 3   2013     1     3 10.987832
## 4   2013     1     4  8.951595
## 5   2013     1     5  5.732218
## 6   2013     1     6  7.148014
## 7   2013     1     7  5.417204
## 8   2013     1     8  2.553073
## 9   2013     1     9  2.276477
## 10  2013     1    10  2.844995
## # ... with 355 more rows
```

Departure delay by day:

```
not_cancelled <- flights %>%
  filter(!is.na(dep_delay), !is.na(arr_delay))

not_cancelled %>%
  group_by(year, month, day) %>%
  summarise(mean = mean(dep_delay))
```

```
## Source: local data frame [365 x 4]
## Groups: year, month [?]
##
##     year month   day      mean
##    <int> <int> <int>     <dbl>
## 1   2013     1     1 11.435620
## 2   2013     1     2 13.677802
## 3   2013     1     3 10.907778
## 4   2013     1     4  8.965859
## 5   2013     1     5  5.732218
```

```
## 6   2013     1     6  7.145959
## 7   2013     1     7  5.417204
## 8   2013     1     8  2.558296
## 9   2013     1     9  2.301232
## 10  2013     1    10  2.844995
## # ... with 355 more rows
```

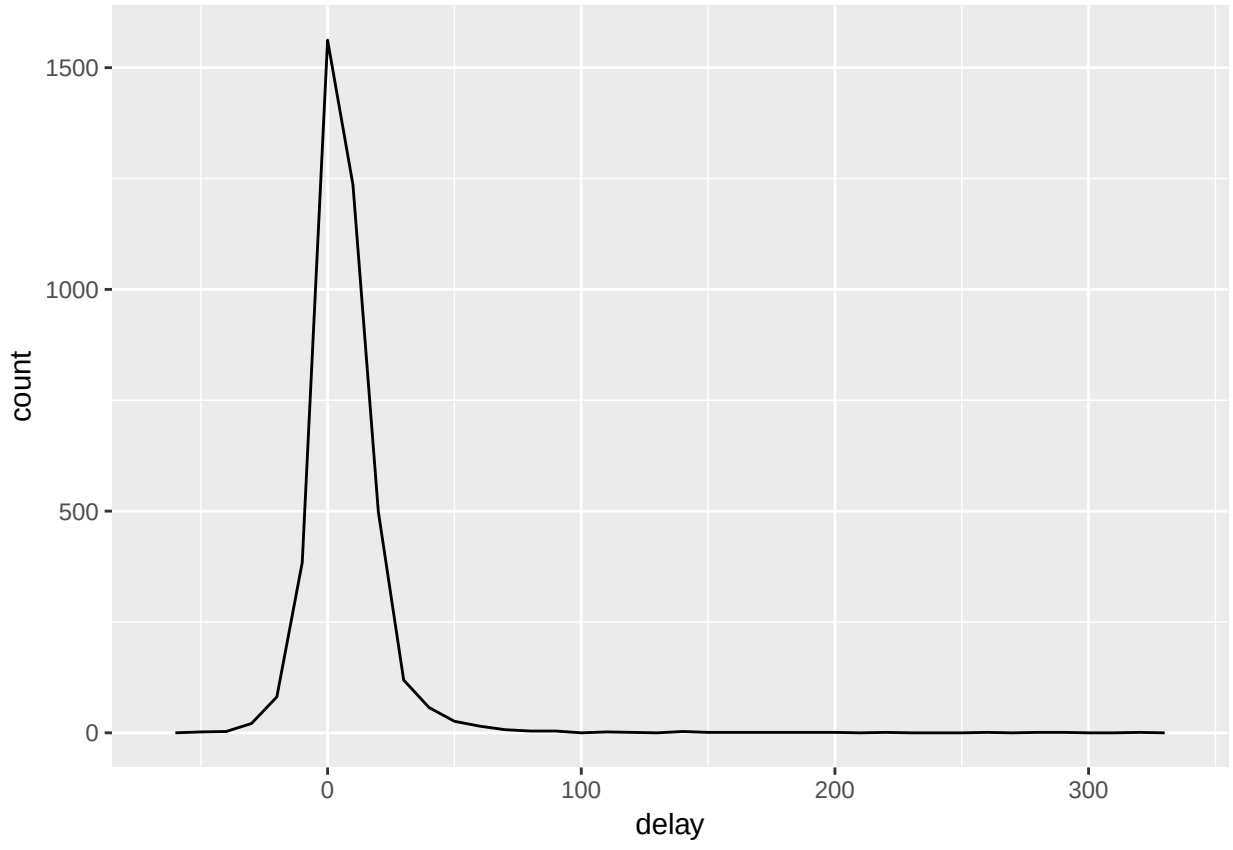
Useful tool alert!

5.6.3 Counts

Whenever you do any aggregation, it's always a good idea to include either a count `n()`, or a count of non-missing values `(sum(!is.na(x)))`. That way you can check that you're not drawing conclusions based on very small amounts of data. For example, let's look at the planes (identified by their tail number) that have the highest average delays:

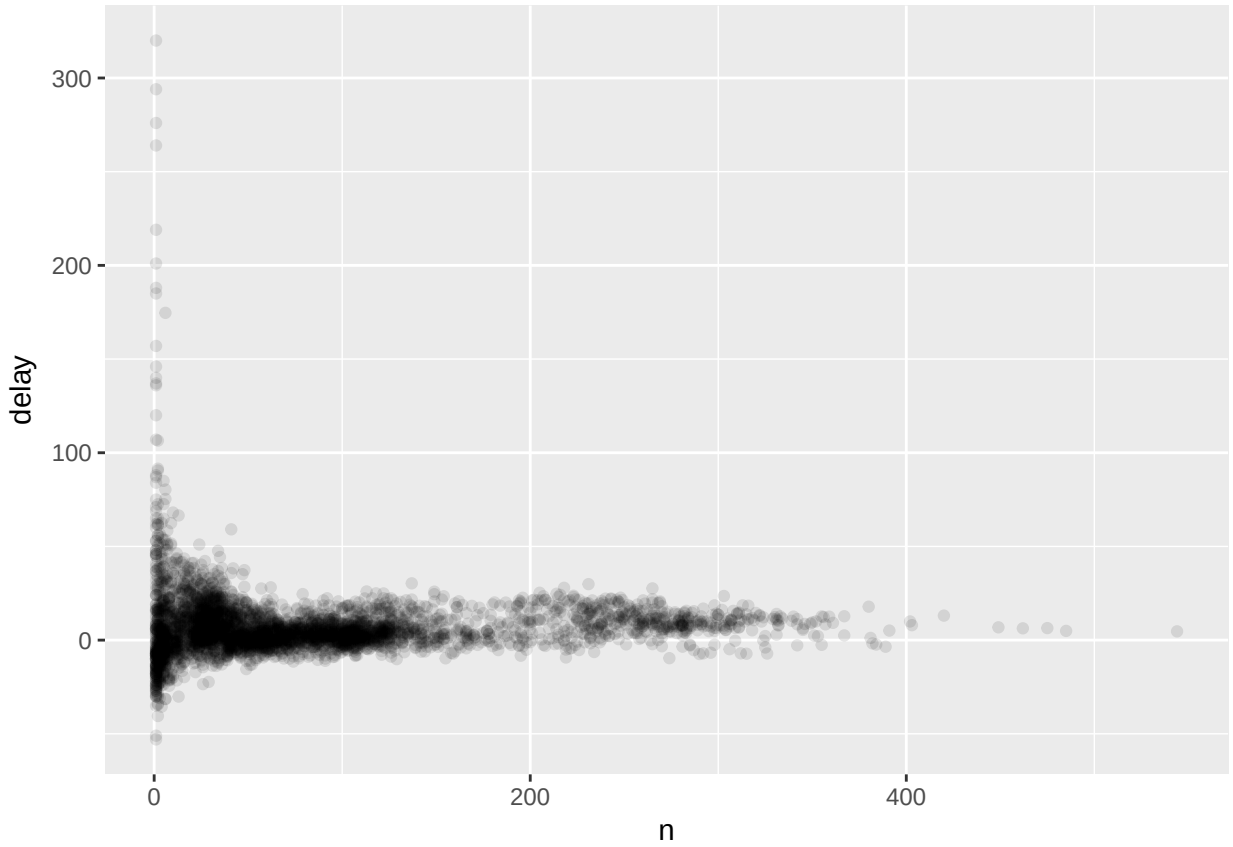
planes identified by their tail number

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

ggplot(data = delays, mapping = aes(x = delay)) +
  geom_freqpoly(binwidth = 10)
```



There are some planes that have an average delay of 5 hours (300 minutes), but the story is actually a little more nuanced. We can get more insight if we draw a scatterplot of number of flights vs. average delay:

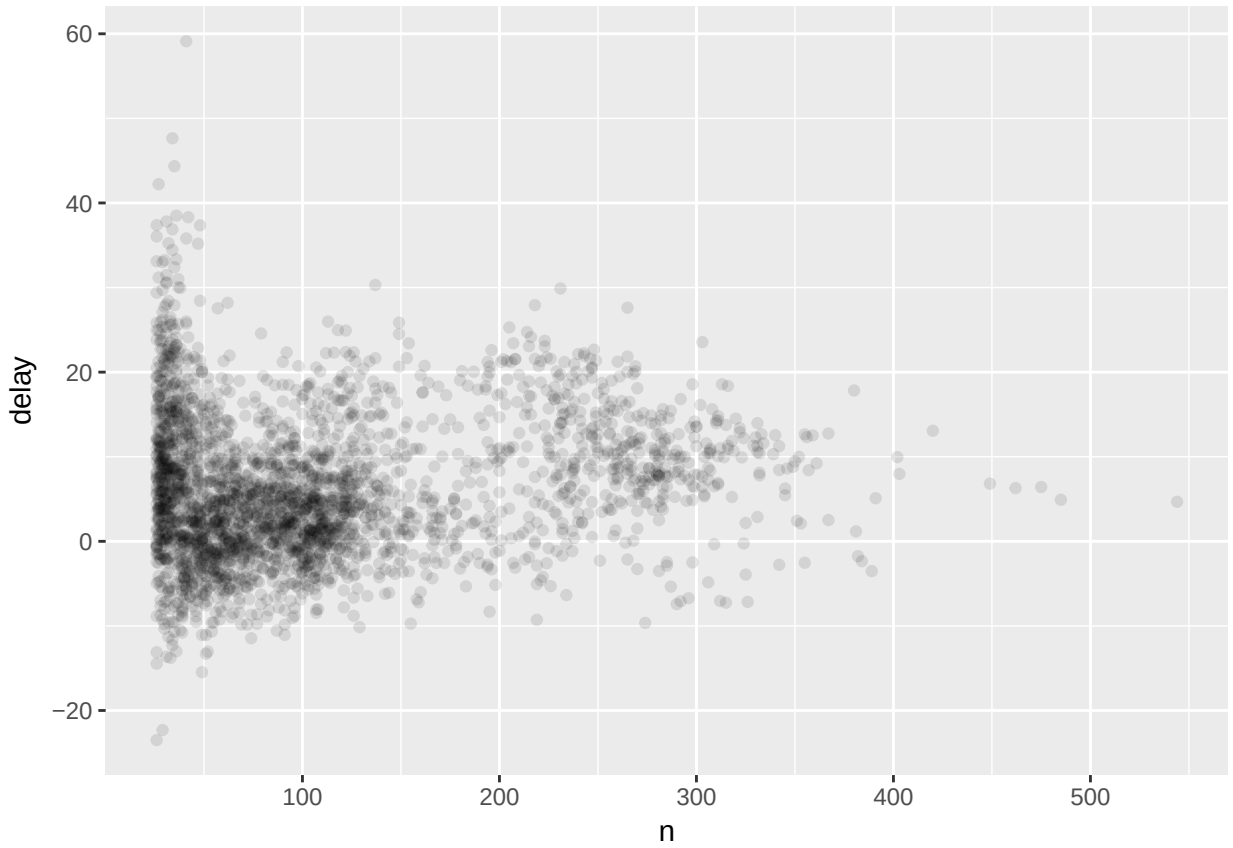
```
delays <- not_cancelled %>%
  group_by(tailnum) %>%
  summarise(
    delay = mean(arr_delay, na.rm = TRUE),
    n = n()  # preparation for 2 uses: 1. as x axis when calling in ggplot
  )

ggplot(data = delays, mapping = aes(x = n, y = delay)) +
  geom_point(alpha = 1/10)
```



Not surprisingly, there is much greater variation in the average delay when there are few flights. The shape of this plot is very characteristic: whenever you plot a mean (or other summary) vs. group size, you'll see that the variation decreases as the sample size increases.

When looking at this sort of plot, it's often useful to filter out the groups with the smallest numbers of observations, so you can see more of the pattern and less of the extreme variation in the smallest groups.

```
delays %>%
  filter(n > 25) %>%  # 2. n = n()  can be used later if data needs to be filtered
  ggplot(mapping = aes(x = n, y = delay)) +
    geom_point(alpha = 1/10)
```



RStudio tip: a useful keyboard shortcut is Cmd/Ctrl + Shift + P. This resends the previously sent chunk from the editor to the console. This is very convenient when you're (e.g.) exploring the value of n in the example above. You send the whole block once with Cmd/Ctrl + Enter, then you modify the value of n and press Cmd/Ctrl + Shift + P to resend the complete block.

Baseball performance

Here I use data from the Lahman package to compute the batting average (number of hits / number of attempts) of every major league baseball player.

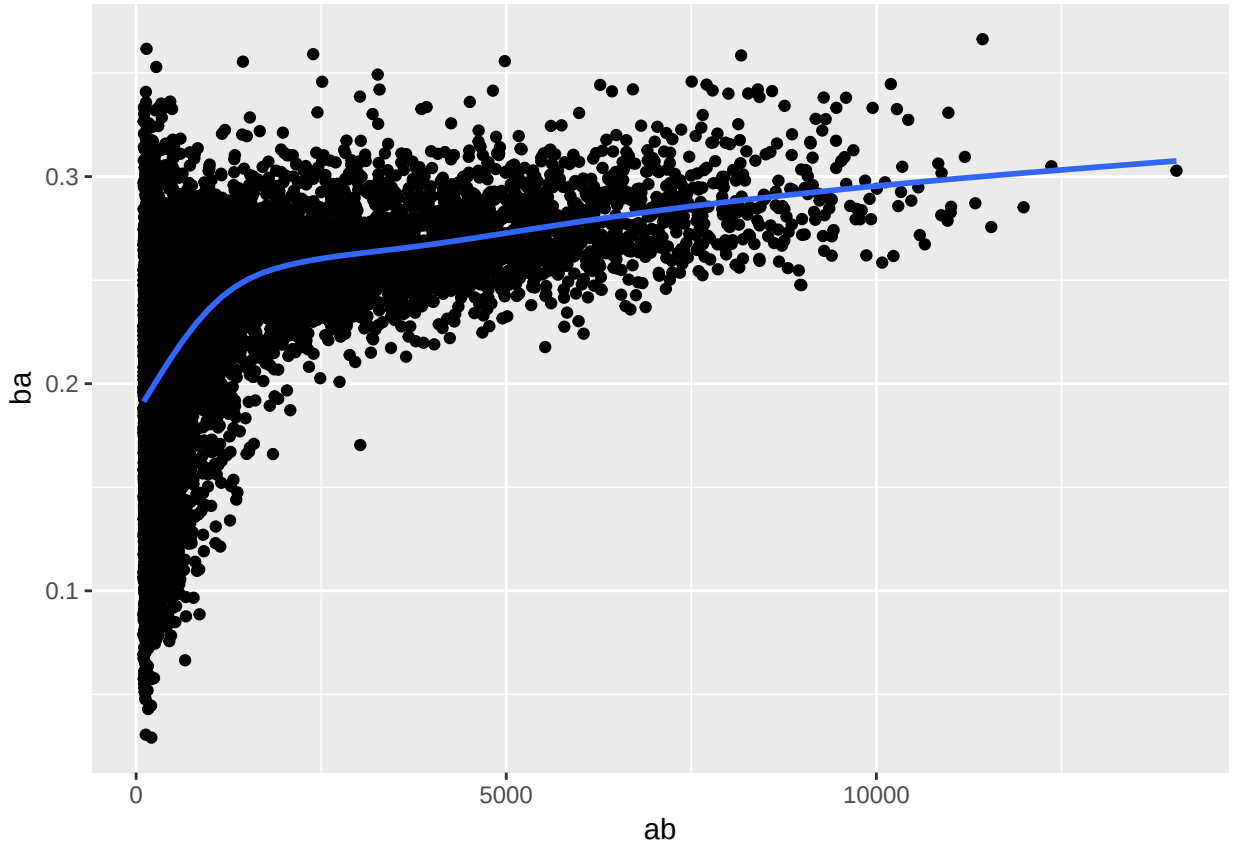
```
# Convert to a tibble so it prints nicely
library(Lahman)
```

```
## Warning: package 'Lahman' was built under R version 3.3.2
```

```
batting <- as_tibble(Lahman::Batting)

batters <- batting %>%
  group_by(playerID) %>%
  summarise(
    ba = sum(H, na.rm = TRUE) / sum(AB, na.rm = TRUE),
    ab = sum(AB, na.rm = TRUE)
  )

batters %>%
  filter(ab > 100) %>%
  ggplot(mapping = aes(x = ab, y = ba)) +
```

```
    geom_point() +
    geom_smooth(se = FALSE)
```

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




```
#> `geom_smooth()` using method = 'gam'
```

you see two patterns:

1. As above, the variation in our aggregate decreases as we get more data points.
2. There's a positive correlation between skill (`ba`) and opportunities to hit the ball (`ab`). This is because teams control who gets to play, and obviously they'll pick their best players.

5.6.4 Useful Summary Functions

It's sometimes useful to combine aggregation with logical subsetting.

Problem: Calculate the average postice delay

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

```
## Source: local data frame [365 x 5]
## Groups: year, month [?]
##
```

```
##     year month   day avg_delay1 avg_delay2
##    <int> <int> <int>      <dbl>      <dbl>
## 1   2013     1     1 12.6510229   32.48156
## 2   2013     1     2 12.6928879   32.02991
## 3   2013     1     3  5.7333333   27.66087
## 4   2013     1     4 -1.9328194   28.30976
## 5   2013     1     5 -1.5258020   22.55882
## 6   2013     1     6  4.2364294   24.37270
## 7   2013     1     7 -4.9473118   27.76132
## 8   2013     1     8 -3.2275785   20.78909
## 9   2013     1     9 -0.2642777   25.63415
## 10  2013     1    10 -5.8988159   27.34545
## # ... with 355 more rows
```

Measures of Position

`first(x)`, `nth(x, 2)`, `last(x)`. These work similarly to `x[1]`, `x[2]`, and `x[length(x)]` but let you set a default value if that position does not exist (i.e. you're trying to get the 3rd element from a group that only has two elements). For example, we can find the first and last departure for each day:

```
not_cancelled %>%
  group_by(year, month, day) %>%
  summarise(
    first_dep = first(dep_time),
    last_dep = last(dep_time)
  )
```

```
## Source: local data frame [365 x 5]
## Groups: year, month [?]
##
##     year month   day first_dep last_dep
##    <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
## 8   2013     1     8       454     2351
## 9   2013     1     9         2     2252
## 10  2013     1    10         3     2320
## # ... with 355 more rows
```

These functions are complementary to filtering on ranks. Filtering gives you all variables, with each observation in a separate row

Testimate Smart: first and last - filter(r %in% range(r))

`min_rank`

```
not_cancelled %>%
  group_by(year, month, day) %>%
  mutate(r = min_rank(desc(dep_time))) %>%
  filter(r %in% range(r))
```

```
## Source: local data frame [770 x 20]
## Groups: year, month, day [365]
##
##     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     2356           2359        -3      425
## 3   2013     1     2       42           2359        43      518
## 4   2013     1     2     2354           2359        -5      413
## 5   2013     1     3       32           2359        33      504
## 6   2013     1     3     2349           2359       -10      434
## 7   2013     1     4       25           2359        26      505
## 8   2013     1     4     2358           2359        -1      429
## 9   2013     1     4     2358           2359        -1      436
## 10  2013     1     5       14           2359        15      503
## # ... with 760 more rows, and 13 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 <dttm>, r <int>
```

Counts: To count the number of non-missing values, use `sum(!is.na(x))`. To count the number of distinct (unique) values, use `n_distinct(x)`. For example:

Which destinations have the most carriers?

```
not_cancelled %>%
  group_by(dest) %>%
  summarise(carriers = n_distinct(carrier)) %>%
  arrange(desc(carriers))
```

```
## # A tibble: 104 x 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
```

```
not_cancelled %>%
  count(dest)
```

```
## # A tibble: 104 x 2
##     dest     n
##    <chr> <int>
## 1    ABQ   254
## 2    ACK   264
## 3    ALB   418
## 4    ANC     8
## 5    ATL 16837
```

```
## 6    AUS  2411
## 7    AVL   261
## 8    BDL   412
## 9    BGR   358
## 10   BHM   269
## # ... with 94 more rows
```

Testimate: similar to `table()`, with even straightforward command.

You can optionally provide a weight variable. For example, you could use this to "count" (sum) the total number of miles a plane flew:

```
not_cancelled %>%
  count(tailnum, wt = distance)
```

```
## # A tibble: 4,037 x 2
##    tailnum      n
##      <chr>  <dbl>
## 1   D942DN   3418
## 2   N0EGMQ 239143
## 3   N10156 109664
## 4   N102UW  25722
## 5   N103US  24619
## 6   N104UW  24616
## 7   N10575 139903
## 8   N105UW  23618
## 9   N107US  21677
## 10  N108UW  32070
## # ... with 4,027 more rows
```

To verify, we may simply calculate / observe like this:

```
not_cancelled %>%
  filter(tailnum == "D942DN") %>%
    select(tailnum, distance)
```

```
## # A tibble: 4 x 2
##   tailnum distance
##     <chr>    <dbl>
## 1  D942DN      762
## 2  D942DN      950
## 3  D942DN      944
## 4  D942DN      762
```

Counts and proportions of logical values: `sum(x)` gives the number of TRUEs in x, and `mean(x)` gives the proportion.

```
# How many flights left before 5am? (these usually indicate delayed
# flights from the previous day)

not_cancelled %>%
  group_by(year, month, day) %>%
  summarize(n_early = sum(dep_time < 500))
```

```
## Source: local data frame [365 x 4]
## Groups: year, month [?]
##
##     year month   day n_early
```

```
##    <int> <int> <int>   <int>
## 1   2013     1     1       0
## 2   2013     1     2       3
## 3   2013     1     3       4
## 4   2013     1     4       3
## 5   2013     1     5       3
## 6   2013     1     6       2
## 7   2013     1     7       2
## 8   2013     1     8       1
## 9   2013     1     9       3
## 10  2013     1    10       3
## # ... with 355 more rows
```

```
# What proportion of flights are delayed by more than an hour?
not_cancelled %>%
  group_by(year, month, day) %>%
  summarise(hour_perc = round(mean(arr_delay > 60), digit = 3))
```

```
## Source: local data frame [365 x 4]
## Groups: year, month [?]
##
##     year month   day hour_perc
##    <int> <int> <int>     <dbl>
## 1   2013     1     1     0.072
## 2   2013     1     2     0.085
## 3   2013     1     3     0.057
## 4   2013     1     4     0.040
## 5   2013     1     5     0.035
## 6   2013     1     6     0.047
## 7   2013     1     7     0.033
## 8   2013     1     8     0.021
## 9   2013     1     9     0.020
## 10  2013     1    10     0.018
## # ... with 355 more rows
```

5.6.5 Grouping by multiple variables

When you group by multiple variables, each summary peels off one level of the grouping. That makes it easy to progressively roll up a dataset:

```
daily <- group_by(flights, year, month, day)
(per_day  <- summarise(daily, flights = n()))
```

```
## Source: local data frame [365 x 4]
## Groups: year, month [?]
##
##     year month   day flights
##    <int> <int> <int>   <int>
## 1   2013     1     1     842
## 2   2013     1     2     943
## 3   2013     1     3     914
## 4   2013     1     4     915
## 5   2013     1     5     720
## 6   2013     1     6     832
## 7   2013     1     7     933
```

```
## 8   2013     1     8     899
## 9   2013     1     9     902
## 10  2013     1    10     932
## # ... with 355 more rows
```

```
(per_month <- summarise(per_day, flights = sum(flights)))
```

```
## Source: local data frame [12 x 3]
## Groups: year [?]
##
##     year month flights
##    <int> <int>   <int>
## 1   2013     1   27004
## 2   2013     2   24951
## 3   2013     3   28834
## 4   2013     4   28330
## 5   2013     5   28796
## 6   2013     6   28243
## 7   2013     7   29425
## 8   2013     8   29327
## 9   2013     9   27574
## 10  2013    10   28889
## 11  2013    11   27268
## 12  2013    12   28135
```

```
(per_year  <- summarise(per_month, flights = sum(flights)))
```

```
## # A tibble: 1 x 2
##    year flights
##   <int>   <int>
## 1  2013  336776
```

Be careful when progressively rolling up summaries: it's OK for sums and counts, but you need to think about weighting means and variances, and it's not possible to do it exactly for rank-based statistics like the median. In other words, the sum of groupwise sums is the overall sum, but the median of groupwise medians is not the overall median.

5.6.6 Ungrouping

```
daily %>%
  ungroup() %>%             # no longer grouped by date
  summarise(flights = n())
```

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

15.1 Factor

Historically, factors were much easier to work with than characters. As a result, many of the functions in base R automatically convert characters to factors. This means that factors often crop up in places where they're not actually helpful. Fortunately, you don't need to worry about that in the tidyverse, and can focus on situations where factors are genuinely useful

15.2 Creating factors

```
x1 <- c("Dec", "Apr", "Jan", "Mar")

sort(x1) # doesn't sort in a useful way
```

```
## [1] "Apr" "Dec" "Jan" "Mar"
```

You can fix both of these problems with a factor. To create a factor you must start by creating a list of the valid levels:

```
month_levels <- c(
  "Jan", "Feb", "Mar", "Apr", "May", "Jun",
  "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"
)

y1 <- factor(x1, levels = month_levels)
y1
```

```
## [1] Dec Apr Jan Mar
## Levels: Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
```

```
sort(y1)
```

```
## [1] Jan Mar Apr Dec
## Levels: Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
```