r4ds-ch3

2025-05-04

Ch 3 Data transformation

3.1 Introduction

3.1.1 Prerequisites

```
library(nycflights13)
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.2     v tibble    3.2.1
## v lubridate 1.9.4     v tidyr     1.3.1
## v purrr     1.0.4
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

dplyr overwrites some base R functions, so when want to use them have to use their full names e.g. stats::filter() When need to be precise, use packagename::functionname()

3.1.2 nycflights13

nycflights13::flights contains 336,776 flights that departed from NYC in 2013

```
flights
```

```
## # A tibble: 336,776 x 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
## # i 336,766 more rows
## # i 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>
```

flights is a tibble, a special type of data frame used by the tidyverse Most important difference is how they print, tibbles are designed for large datasets so they only show the first few rows and columns that can fit on the screen In RStudio you can use View(flights) for an interactive view of all the data You can also do print(flights, width = Inf) to show all columns

```r
glimpse(flights)  # Or use glimpse()
```

```
## Rows: 336,776
## Columns: 19
## $ year           <int> 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2~
## $ month          <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
## $ day            <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
## $ dep_time       <int> 517, 533, 542, 544, 554, 554, 555, 557, 557, 558, 558, ~
## $ sched_dep_time <int> 515, 529, 540, 545, 600, 558, 600, 600, 600, 600, 600, ~
## $ dep_delay      <dbl> 2, 4, 2, -1, -6, -4, -5, -3, -3, -2, -2, -2, -2, -2, -1~
## $ arr_time       <int> 830, 850, 923, 1004, 812, 740, 913, 709, 838, 753, 849,~
## $ sched_arr_time <int> 819, 830, 850, 1022, 837, 728, 854, 723, 846, 745, 851,~
## $ arr_delay      <dbl> 11, 20, 33, -18, -25, 12, 19, -14, -8, 8, -2, -3, 7, -1~
## $ carrier        <chr> "UA", "UA", "AA", "B6", "DL", "UA", "B6", "EV", "B6", "~
## $ flight         <int> 1545, 1714, 1141, 725, 461, 1696, 507, 5708, 79, 301, 4~
## $ tailnum        <chr> "N14228", "N24211", "N619AA", "N804JB", "N668DN", "N394~
## $ origin         <chr> "EWR", "LGA", "JFK", "JFK", "LGA", "EWR", "EWR", "LGA",~
## $ dest           <chr> "IAH", "IAH", "MIA", "BQN", "ATL", "ORD", "FLL", "IAD",~
## $ air_time       <dbl> 227, 227, 160, 183, 116, 150, 158, 53, 140, 138, 149, 1~
## $ distance       <dbl> 1400, 1416, 1089, 1576, 762, 719, 1065, 229, 944, 733, ~
## $ hour           <dbl> 5, 5, 5, 5, 6, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 5, 6, 6, 6~
## $ minute         <dbl> 15, 29, 40, 45, 0, 58, 0, 0, 0, 0, 0, 0, 0, 0, 0, 59, 0~
## $ time_hour      <dttm> 2013-01-01 05:00:00, 2013-01-01 05:00:00, 2013-01-01 0~
```

Variable names are followed by abbreviations for the type of variable: integers, doubles (real numbers), character (strings), dttm for date-time

3.1.3 dplyr basics

What dplyr verbs (functions) have in common 1. First argument is always a data frame 2. Subsequent arguments describe which columns to operate on using the variable names (without quotes) 3. Output is always a new data frame

Combine multiple verbs with the pipe |> Pipe takes thing on its left and passes it to function on its right x |> f(y) is equivalent to f(x, y) x |> f(y) |> g(z) is equivalent to g(f(x, y), z) Pronounce the pipe as "then"

```r
flights |>
  filter(dest == "IAH") |>
  group_by(year, month, day) |>
```

```
  summarize(
    arr_delay = mean(arr_delay, na.rm = TRUE)
  )
```

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

```
## # A tibble: 365 x 4
## # Groups:   year, month [12]
##     year month   day arr_delay
##    <int> <int> <int>     <dbl>
##  1  2013     1     1     17.8
##  2  2013     1     2      7
##  3  2013     1     3     18.3
##  4  2013     1     4     -3.2
##  5  2013     1     5     20.2
##  6  2013     1     6      9.28
##  7  2013     1     7     -7.74
##  8  2013     1     8      7.79
##  9  2013     1     9     18.1
## 10  2013     1    10      6.68
## # i 355 more rows
```

The code above groups flights whose destination is IAH and displays the mean arrival delay for each day

dplyr verbs are organized into 4 groups based on what they operate on: rows, columns, groups, or tables

3.2 Rows

Most important for rows are 1. filter() which changes which rows are present without changing their order 2. arrange() which changes the order of rows without changing which are present

3.2.1 filter()

Allows you to keep rows based on values of columns First argument is data frame, the next are conditions that must be true for the row to be kept

Find all flights that departed more than 2 hours late

```
flights |>
  filter(dep_delay > 120)
```

```
## # A tibble: 9,723 x 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      848           1835       853     1001           1950
##  2  2013     1     1      957            733       144     1056            853
##  3  2013     1     1     1114            900       134     1447           1222
##  4  2013     1     1     1540           1338       122     2020           1825
##  5  2013     1     1     1815           1325       290     2120           1542
##  6  2013     1     1     1842           1422       260     1958           1535
```

```
## 7 2013 1 1 1856 1645 131 2212 2005
## 8 2013 1 1 1934 1725 129 2126 1855
## 9 2013 1 1 1938 1703 155 2109 1823
## 10 2013 1 1 1942 1705 157 2124 1830
## # i 9,713 more rows
## # i 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>
```

Can combine conditions with & or , to indicate "and" and | to indicate "or"

```r
# Flights that departed on Jan 1
flights |>
  filter(month == 1 & day == 1)
```

```
## # A tibble: 842 x 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
## # i 832 more rows
## # i 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>
```

```r
# Flights that departed in Jan or Feb
flights |>
  filter(month == 1 | month == 2)
```

```
## # A tibble: 51,955 x 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
## # i 51,945 more rows
## # i 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>
```

Useful shortcut to combine | and == which is %in%

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

```
## # A tibble: 51,955 x 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
## # i 51,945 more rows
## # i 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() creates a new data frame then prints it, doesn't modify original dplyr functions never modify their inputs To save the result, do so with the assignment operator

```
jan1 <- flights |>
  filter(month == 1 & day == 1)
```

3.2.2 Common mistakes

Using = instead of == to test for equality, filter() lets you know when this happens

```
#flights |>
  #filter(month = 1)
```

Another mistake is to write "or" statements like you would in English

```
flights |>
  filter(month == 1 | 2)
```

```
## # A tibble: 336,776 x 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
## # i 336,766 more rows
## # i 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>
```

No error, but first checks condition month == 1 and then 2, 2 is always true so this doesn't filter anything

3.2.3 arrange()

Changes order of rows based on value of the columns Takes a data frame and set of column names to order by If provide more than 1 column, each successive column is used to break ties

Sort by departure time, earliest years first, then within a year the earliest months, etc.

```
flights |>
  arrange(year, month, day, dep_time)
```

```
## # A tibble: 336,776 x 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
## # i 336,766 more rows
## # i 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>
```

Can use desc() on a column inside of arrange() to re-order data frame based on that column in descending order

This orders flights from most to least delayed

```
flights |>
  arrange(desc(dep_delay))
```

```
## # A tibble: 336,776 x 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     6    27      959            1900       899      1236           2226
## 9 2013     7    22     2257             759       898       121           1026
## 10 2013   12     5      756            1700       896      1058           2020
## # i 336,766 more rows
## # i 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>
```

Number of rows doesn't change, not filtering, only re-ordering

3.2.4 distinct()

Finds all unique rows in dataset Most of time will want distinct combo of some variables, so can also optionally supply column names

```r
# Remove duplicate rows if there are any
flights |>
  distinct()
```

```
## # A tibble: 336,776 x 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
## # i 336,766 more rows
## # i 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>
```

```r
# Find all unique origin and destination pairs
flights |>
  distinct(origin, dest)
```

```
## # A tibble: 224 x 2
##    origin dest
##    <chr>  <chr>
##  1 EWR    IAH
##  2 LGA    IAH
##  3 JFK    MIA
##  4 JFK    BQN
##  5 LGA    ATL
```

```
##   6 EWR    ORD
##   7 EWR    FLL
##   8 LGA    IAD
##   9 JFK    MCO
##  10 LGA    ORD
## # i 214 more rows
```

If want to keep all other columns when filtering for unique rows, can use .keep_all = TRUE

```r
flights |>
  distinct(origin, dest, .keep_all = TRUE)
```

```
## # A tibble: 224 x 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
## # i 214 more rows
## # i 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>
```

Not coincidence all from Jan 1, distinct() finds first occurrence of row in dataset and discards rest

To find number of occurences, swap distinct() with count() With sort = TRUE, arranges them in descending order of number of occurrences

```r
flights |>
  count(origin, dest, sort = TRUE)
```

```
## # A tibble: 224 x 3
##    origin dest      n
##    <chr>  <chr> <int>
##  1 JFK    LAX   11262
##  2 LGA    ATL   10263
##  3 LGA    ORD    8857
##  4 JFK    SFO    8204
##  5 LGA    CLT    6168
##  6 EWR    ORD    6100
##  7 JFK    BOS    5898
##  8 LGA    MIA    5781
##  9 JFK    MCO    5464
## 10 EWR    BOS    5327
## # i 214 more rows
```

3.2.5 Exercises

1. In a single pipeline for each condition, find all flights that meet the condition:

a) Arrival delay of two or more hours

```
flights |>
  filter(arr_delay >= 120)
```

```
## # A tibble: 10,200 x 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      811            630       101     1047            830
##  2  2013     1     1      848           1835       853     1001           1950
##  3  2013     1     1      957            733       144     1056            853
##  4  2013     1     1     1114            900       134     1447           1222
##  5  2013     1     1     1505           1310       115     1638           1431
##  6  2013     1     1     1525           1340       105     1831           1626
##  7  2013     1     1     1549           1445        64     1912           1656
##  8  2013     1     1     1558           1359       119     1718           1515
##  9  2013     1     1     1732           1630        62     2028           1825
## 10  2013     1     1     1803           1620       103     2008           1750
## # i 10,190 more rows
## # i 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>
```

b) Flew to Houston (IAH or HOU)

```
flights |>
  filter(dest %in% c("IAH", "HOU"))
```

```
## # A tibble: 9,313 x 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      623            627        -4      933            932
##  4  2013     1     1      728            732        -4     1041           1038
##  5  2013     1     1      739            739         0     1104           1038
##  6  2013     1     1      908            908         0     1228           1219
##  7  2013     1     1     1028           1026         2     1350           1339
##  8  2013     1     1     1044           1045        -1     1352           1351
##  9  2013     1     1     1114            900       134     1447           1222
## 10  2013     1     1     1205           1200         5     1503           1505
## # i 9,303 more rows
## # i 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>
```

c) Were operated by United, American, or Delta

```r
airlines  # Look up airline codes
```

```
## # A tibble: 16 x 2
##    carrier name
##    <chr>   <chr>
##  1 9E      Endeavor Air Inc.
##  2 AA      American Airlines Inc.
##  3 AS      Alaska Airlines Inc.
##  4 B6      JetBlue Airways
##  5 DL      Delta Air Lines Inc.
##  6 EV      ExpressJet Airlines Inc.
##  7 F9      Frontier Airlines Inc.
##  8 FL      AirTran Airways Corporation
##  9 HA      Hawaiian Airlines Inc.
## 10 MQ      Envoy Air
## 11 OO      SkyWest Airlines Inc.
## 12 UA      United Air Lines Inc.
## 13 US      US Airways Inc.
## 14 VX      Virgin America
## 15 WN      Southwest Airlines Co.
## 16 YV      Mesa Airlines Inc.
```

```r
flights |>
  filter(carrier %in% c("UA", "AA", "DL"))
```

```
## # A tibble: 139,504 x 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      554            600        -6      812            837
##  5  2013     1     1      554            558        -4      740            728
##  6  2013     1     1      558            600        -2      753            745
##  7  2013     1     1      558            600        -2      924            917
##  8  2013     1     1      558            600        -2      923            937
##  9  2013     1     1      559            600        -1      941            910
## 10  2013     1     1      559            600        -1      854            902
## # i 139,494 more rows
## # i 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>
```

d) Departed in summer (July, August, and September)

```r
flights |>
  filter(month %in% c(7:9))
```

```
## # A tibble: 86,326 x 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     7     1        1            2029        212       236              2359
## 2  2013     7     1        2            2359          3       344               344
## 3  2013     7     1       29            2245        104       151                 1
## 4  2013     7     1       43            2130        193       322                14
## 5  2013     7     1       44            2150        174       300               100
## 6  2013     7     1       46            2051        235       304              2358
## 7  2013     7     1       48            2001        287       308              2305
## 8  2013     7     1       58            2155        183       335                43
## 9  2013     7     1      100            2146        194       327                30
## 10 2013     7     1      100            2245        135       337               135
## # i 86,316 more rows
## # i 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>
```

e) Arrived more than two hours late but didn't leave late

```
flights |>
  filter(dep_delay <= 0 & arr_delay > 120)
```

```
## # A tibble: 29 x 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    27     1419           1420        -1     1754           1550
## 2  2013    10     7     1350           1350         0     1736           1526
## 3  2013    10     7     1357           1359        -2     1858           1654
## 4  2013    10    16      657            700        -3     1258           1056
## 5  2013    11     1      658            700        -2     1329           1015
## 6  2013     3    18     1844           1847        -3       39           2219
## 7  2013     4    17     1635           1640        -5     2049           1845
## 8  2013     4    18      558            600        -2     1149            850
## 9  2013     4    18      655            700        -5     1213            950
## 10 2013     5    22     1827           1830        -3     2217           2010
## # i 19 more rows
## # i 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>
```

f) Were delayed by at least an hour, but made up over 30 min in flight

```
flights |>
  filter(dep_delay >= 60 & (dep_delay - arr_delay) > 30)
```

```
## # A tibble: 1,844 x 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     2205           1720       285       46           2040
## 2  2013     1     1     2326           2130       116      131             18
## 3  2013     1     3     1503           1221       162     1803           1555
## 4  2013     1     3     1839           1700        99     2056           1950
## 5  2013     1     3     1850           1745        65     2148           2120
## 6  2013     1     3     1941           1759       102     2246           2139
```

```
## 7  2013     1     3     1950           1845        65     2228           2227
## 8  2013     1     3     2015           1915        60     2135           2111
## 9  2013     1     3     2257           2000       177       45           2224
## 10 2013     1     4     1917           1700       137     2135           1950
## # i 1,834 more rows
## # i 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>
```

2. Sort flights to find the flights with the longest departure delays

```r
flights |>
  arrange(desc(dep_delay))
```

```
## # A tibble: 336,776 x 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     6    27      959           1900       899     1236           2226
## 9  2013     7    22     2257            759       898      121           1026
## 10 2013    12     5      756           1700       896     1058           2020
## # i 336,766 more rows
## # i 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>
```

Find the flights that left earliest in the morning

```r
flights |>
  arrange(dep_time)
```

```
## # A tibble: 336,776 x 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    13        1           2249        72      108           2357
## 2  2013     1    31        1           2100       181      124           2225
## 3  2013    11    13        1           2359         2      442            440
## 4  2013    12    16        1           2359         2      447            437
## 5  2013    12    20        1           2359         2      430            440
## 6  2013    12    26        1           2359         2      437            440
## 7  2013    12    30        1           2359         2      441            437
## 8  2013     2    11        1           2100       181      111           2225
## 9  2013     2    24        1           2245        76      121           2354
## 10 2013     3     8        1           2355         6      431            440
## # i 336,766 more rows
## # i 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>
```

3. Sort flights to find the fastest flights

```
# First convert arrival and departure times to minutes
arrival <- function() {
  as.integer(substring(str_pad(flights$arr_time, 4, side = "left", pad = 0), 1, 2))*60 + as.integer(subs
}

departure <- function() {
  as.integer(substring(str_pad(flights$dep_time, 4, side = "left", pad = 0), 1, 2))*60 + as.integer(subs
}

# Subtract departure from arrival to determine complete flight time
flight_time <- function() {
  times <- arrival() - departure()
  # If negative, then add 1400 (means departed at night and arrived morning)
  times[times < 0 & !is.na(times)] <- times[times < 0 & !is.na(times)] + 1400
  times
}

# Arrange by flight time
flights |>
  arrange(flight_time())
```

```
## # A tibble: 336,776 x 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     6    12     2338           2129       129       17           2235
##  2  2013    12    29     2332           2155        97       14           2300
##  3  2013    11     6     2335           2215        80       18           2317
##  4  2013     2    25     2347           2145       122       30           2239
##  5  2013     8    13     2351           2152       119       35           2258
##  6  2013    10    11     2342           2030       192       27           2205
##  7  2013     2    26     2356           2000       236       41           2104
##  8  2013     1    24     2342           2159       103       28           2300
##  9  2013    12    23     2333           2155        98       19           2257
## 10  2013     3    10     2339           2200        99       25           2254
## # i 336,766 more rows
## # i 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>
```

4. Was there a flight every day of 2013?

```
nrow(distinct(flights, month, day)) == 365
```

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

Yes.

5. Which flights traveled the farthest distance?

```r
flights |>
  arrange(desc(distance))
```

```
## # A tibble: 336,776 x 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      857            900        -3     1516           1530
##  2  2013     1     2      909            900         9     1525           1530
##  3  2013     1     3      914            900        14     1504           1530
##  4  2013     1     4      900            900         0     1516           1530
##  5  2013     1     5      858            900        -2     1519           1530
##  6  2013     1     6     1019            900        79     1558           1530
##  7  2013     1     7     1042            900       102     1620           1530
##  8  2013     1     8      901            900         1     1504           1530
##  9  2013     1     9      641            900      1301     1242           1530
## 10  2013     1    10      859            900        -1     1449           1530
## # i 336,766 more rows
## # i 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>
```

Which traveled the least distance?

```r
flights |>
  arrange(distance)
```

```
## # A tibble: 336,776 x 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     7    27       NA            106        NA       NA            245
##  2  2013     1     3     2127           2129        -2     2222           2224
##  3  2013     1     4     1240           1200        40     1333           1306
##  4  2013     1     4     1829           1615       134     1937           1721
##  5  2013     1     4     2128           2129        -1     2218           2224
##  6  2013     1     5     1155           1200        -5     1241           1306
##  7  2013     1     6     2125           2129        -4     2224           2224
##  8  2013     1     7     2124           2129        -5     2212           2224
##  9  2013     1     8     2127           2130        -3     2304           2225
## 10  2013     1     9     2126           2129        -3     2217           2224
## # i 336,766 more rows
## # i 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>
```

6. Does it matter what order you used filter() and arrange() if you are using both? Yes, in terms of speed/how much work the function has to do because using filter() first reduces the number of rows used in arrange() but using arrange() first means every row has to be sorted, and either way you have to filter through every row However, in terms of the result, no For example, let's say I take a vector 1:10 and filter only even numbers, so it is 2, 4, 6, etc., then arrange it descending, I would get it 10, 8, 6, etc. I filtered through 10 numbers but only had to sort 5 numbers If I take the same vector and first arrange it descending it would be 10, 9, 8, etc. and then when I filter out odd numbers I would get 10, 8, 6, etc. However, I filtered through 10 numbers and had to sort 10 numbers While the result is same, filtering before arranging is faster/more efficient

Let's test this

```
df <- data.frame(x = 1:100000000)

filter_first <- function() {
  df |>
    filter(x %% 2 == 0) |>
    arrange(desc(x))
}

arrange_first <- function() {
  df |>
    arrange(desc(x)) |>
    filter(x %% 2 == 0)
}
```

```
system.time(filter_first())
```

```
##    user  system elapsed
##   1.220   1.278   3.295
```

```
system.time(arrange_first())
```

```
##    user  system elapsed
##   1.228   1.268   3.638
```

3.3 Columns

Four important verbs affect columns without changing rows 1. mutate() creates new columns derived from existing ones 2. select() changes which are present 3. rename() changes names 4. relocate() changes positions

3.3.1 mutate()

Can use to compute gain (how much time delayed flight made up in air) and the speed in mph

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

```
## # A tibble: 336,776 x 21
##     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
## # i 336,766 more rows
## # i 13 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>, gain <dbl>, speed <dbl>
```

mutate() adds new rows by default on ride side, can use .before to add to left side instead

```
flights |>
  mutate(
    gain = dep_delay - arr_delay,
    speed = distance / air_time * 60,
    .before = 1
  )
```

```
## # A tibble: 336,776 x 21
##     gain speed  year month   day dep_time sched_dep_time dep_delay arr_time
##    <dbl> <dbl> <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1    -9  370.  2013     1     1      517            515         2      830
##  2   -16  374.  2013     1     1      533            529         4      850
##  3   -31  408.  2013     1     1      542            540         2      923
##  4    17  517.  2013     1     1      544            545        -1     1004
##  5    19  394.  2013     1     1      554            600        -6      812
##  6   -16  288.  2013     1     1      554            558        -4      740
##  7   -24  404.  2013     1     1      555            600        -5      913
##  8    11  259.  2013     1     1      557            600        -3      709
##  9     5  405.  2013     1     1      557            600        -3      838
## 10   -10  319.  2013     1     1      558            600        -2      753
## # i 336,766 more rows
## # i 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>
```

. indicates that .before is an argument to the function, not the name of a third variable we are creating Can also use .after and in both can use variable name instead of position

For example can add new variables after day column

```
flights |>
  mutate(
    gain = dep_delay - arr_delay,
    speed = distance / air_time * 60,
    .after = day
  )
```

```
## # A tibble: 336,776 x 21
##     year month   day  gain speed dep_time sched_dep_time dep_delay arr_time
##    <int> <int> <int> <dbl> <dbl>    <int>          <int>     <dbl>    <int>
##  1  2013     1     1    -9  370.      517            515         2      830
##  2  2013     1     1   -16  374.      533            529         4      850
```

```
##  3  2013     1     1   -31  408.      542            540          2       923
##  4  2013     1     1    17  517.      544            545         -1      1004
##  5  2013     1     1    19  394.      554            600         -6       812
##  6  2013     1     1   -16  288.      554            558         -4       740
##  7  2013     1     1   -24  404.      555            600         -5       913
##  8  2013     1     1    11  259.      557            600         -3       709
##  9  2013     1     1     5  405.      557            600         -3       838
## 10  2013     1     1   -10  319.      558            600         -2       753
## # i 336,766 more rows
## # i 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>
```

Alternatively can control which variables are kept with .keep, can use "used" argument which specifies to only keep columns involved or created in mutate()

```r
flights |>
  mutate(
    gain = dep_delay - arr_delay,
    hours = air_time / 60,
    gain_per_hour = gain / hours,
    .keep = "used"
  )
```

```
## # A tibble: 336,776 x 6
##    dep_delay arr_delay air_time  gain hours gain_per_hour
##        <dbl>     <dbl>    <dbl> <dbl> <dbl>         <dbl>
##  1         2        11      227    -9 3.78          -2.38
##  2         4        20      227   -16 3.78          -4.23
##  3         2        33      160   -31 2.67         -11.6
##  4        -1       -18      183    17 3.05           5.57
##  5        -6       -25      116    19 1.93           9.83
##  6        -4        12      150   -16 2.5           -6.4
##  7        -5        19      158   -24 2.63          -9.11
##  8        -3       -14       53    11 0.883         12.5
##  9        -3        -8      140     5 2.33           2.14
## 10        -2         8      138   -10 2.3           -4.35
## # i 336,766 more rows
```

3.2.2 select()

Allows you to zoom in on useful subset of variables based on their names

Select by names

```r
flights |>
  select(year, month, day)
```

```
## # A tibble: 336,776 x 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
## # i 336,766 more rows
```

Select all columns between year and day (inclusive)

```r
flights |>
  select(year:day)
```

```
## # A tibble: 336,776 x 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
## # i 336,766 more rows
```

Select all columns except those from year to day (inclusive)

```r
flights |>
  select(!year:day)
```

```
## # A tibble: 336,776 x 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
## # i 336,766 more rows
## # i 9 more variables: flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

Recommend use ! instead of -

Select all columns that are characters

```
flights |>
  select(where(is.character))
```

```
## # A tibble: 336,776 x 4
##    carrier tailnum origin dest
##    <chr>   <chr>   <chr>  <chr>
##  1 UA      N14228  EWR    IAH
##  2 UA      N24211  LGA    IAH
##  3 AA      N619AA  JFK    MIA
##  4 B6      N804JB  JFK    BQN
##  5 DL      N668DN  LGA    ATL
##  6 UA      N39463  EWR    ORD
##  7 B6      N516JB  EWR    FLL
##  8 EV      N829AS  LGA    IAD
##  9 B6      N593JB  JFK    MCO
## 10 AA      N3ALAA  LGA    ORD
## # i 336,766 more rows
```

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" - num_range("x", 1:3) matches x1, x2, and x3

For more details

```
?select
```

Can use matches() with regex

Can rename variables as you select() with =, new name on LHS, old on RHS

```
flights |>
  select(tail_num = tailnum)
```

```
## # A tibble: 336,776 x 1
##    tail_num
##    <chr>
##  1 N14228
##  2 N24211
##  3 N619AA
##  4 N804JB
##  5 N668DN
##  6 N39463
##  7 N516JB
##  8 N829AS
##  9 N593JB
## 10 N3ALAA
## # i 336,766 more rows
```

3.3.3 rename()

If want to keep all existing variables and rename a few, use rename() instead of select()

```r
flights |>
  rename(tail_num = tailnum)
```

```
## # A tibble: 336,776 x 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
## # i 336,766 more rows
## # i 11 more variables: 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>
```

For automated column name cleaning check out janitor::clean_names()

3.3.4 relocate()

By default moves variables to the front

```r
flights |>
  relocate(time_hour, air_time)
```

```
## # A tibble: 336,776 x 19
##    time_hour           air_time  year month   day dep_time sched_dep_time
##    <dttm>                 <dbl> <int> <int> <int>    <int>          <int>
##  1 2013-01-01 05:00:00      227  2013     1     1      517            515
##  2 2013-01-01 05:00:00      227  2013     1     1      533            529
##  3 2013-01-01 05:00:00      160  2013     1     1      542            540
##  4 2013-01-01 05:00:00      183  2013     1     1      544            545
##  5 2013-01-01 06:00:00      116  2013     1     1      554            600
##  6 2013-01-01 05:00:00      150  2013     1     1      554            558
##  7 2013-01-01 06:00:00      158  2013     1     1      555            600
##  8 2013-01-01 06:00:00       53  2013     1     1      557            600
##  9 2013-01-01 06:00:00      140  2013     1     1      557            600
## 10 2013-01-01 06:00:00      138  2013     1     1      558            600
## # i 336,766 more rows
## # i 12 more variables: dep_delay <dbl>, arr_time <int>, sched_arr_time <int>,
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>, origin <chr>,
## #   dest <chr>, distance <dbl>, hour <dbl>, minute <dbl>
```

Can also specify where to put them using .before and .after like in mutate()

```
flights |>
  relocate(year:dep_time, .after = time_hour)
```

```
## # A tibble: 336,776 x 19
##    sched_dep_time dep_delay arr_time sched_arr_time arr_delay carrier flight
##             <int>     <dbl>    <int>          <int>     <dbl> <chr>    <int>
##  1            515         2      830            819        11 UA        1545
##  2            529         4      850            830        20 UA        1714
##  3            540         2      923            850        33 AA        1141
##  4            545        -1     1004           1022       -18 B6         725
##  5            600        -6      812            837       -25 DL         461
##  6            558        -4      740            728        12 UA        1696
##  7            600        -5      913            854        19 B6         507
##  8            600        -3      709            723       -14 EV        5708
##  9            600        -3      838            846        -8 B6          79
## 10            600        -2      753            745         8 AA         301
## # i 336,766 more rows
## # i 12 more variables: tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>,
## #   distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>, year <int>,
## #   month <int>, day <int>, dep_time <int>
```

```
flights|>
  relocate(starts_with("arr"), .before = dep_time)
```

```
## # A tibble: 336,776 x 19
##     year month   day arr_time arr_delay dep_time sched_dep_time dep_delay
##    <int> <int> <int>    <int>     <dbl>    <int>          <int>     <dbl>
##  1  2013     1     1      830        11      517            515         2
##  2  2013     1     1      850        20      533            529         4
##  3  2013     1     1      923        33      542            540         2
##  4  2013     1     1     1004       -18      544            545        -1
##  5  2013     1     1      812       -25      554            600        -6
##  6  2013     1     1      740        12      554            558        -4
##  7  2013     1     1      913        19      555            600        -5
##  8  2013     1     1      709       -14      557            600        -3
##  9  2013     1     1      838        -8      557            600        -3
## 10  2013     1     1      753         8      558            600        -2
## # i 336,766 more rows
## # i 11 more variables: sched_arr_time <int>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

3.3.5 Exercises

1. Compare dep_time, sched_dep_time, and dep_delay, how would you expect those 3 numbers to be related? They are all related to the departure time, specifically: dep_delay = dep_time - sched_dep_time

```r
flights |>
  select(contains("dep"))
```

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

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

```r
flights |>  # Prefix
  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
## # i 336,766 more rows
```

```r
flights |>  # Regex method 1
  select(matches("\\b[a,d,e,p,r]{3}_[a-z]{4,5}\\b"))
```

```
## # 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
## # i 336,766 more rows
```

```
flights |>  # Regex method 2
  select(matches("\\b(arr|dep)_(time|delay)\\b"))
```

```
## # 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
## # i 336,766 more rows
```

3. What happens if you specify the name of the same variable multiple times in a select() call? It only selects that variable once

```
flights |>
  select(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
## # i 336,766 more rows
```

4. What does any_of() function do? Why might it be helpful in conjunction with this vector?

```
variables <- c("year", "month", "day", "dep_delay", "arr_delay")
```

It doesn't check for missing variables, so you can throw in a column that doesn't exist and you won't get an error unlike all_of()

```r
flights |>
  select(any_of(variables))
```

```
## # 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
## # i 336,766 more rows
```

```r
flights |>
  select(all_of(variables))
```

```
## # 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
## # i 336,766 more rows
```

```r
# Add a variable that doesn't exist
variables <- c("year", "month", "day", "dep_delay", "arr_delay", "seconds")

flights |>
  # select(all_of(variables))  Error, can't subset elements that don't exist
  select(any_of(variables))
```

```
## # 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
## # i 336,766 more rows
```

5. Does result of running following code surprise you? How do select helpers deal with upper and lower case by default and how can you change that?

```
flights |> select(contains("TIME"))
```

```
## # A tibble: 336,776 x 6
##    dep_time sched_dep_time arr_time sched_arr_time air_time time_hour
##       <int>          <int>    <int>          <int>    <dbl> <dttm>
##  1      517            515      830            819      227 2013-01-01 05:00:00
##  2      533            529      850            830      227 2013-01-01 05:00:00
##  3      542            540      923            850      160 2013-01-01 05:00:00
##  4      544            545     1004           1022      183 2013-01-01 05:00:00
##  5      554            600      812            837      116 2013-01-01 06:00:00
##  6      554            558      740            728      150 2013-01-01 05:00:00
##  7      555            600      913            854      158 2013-01-01 06:00:00
##  8      557            600      709            723       53 2013-01-01 06:00:00
##  9      557            600      838            846      140 2013-01-01 06:00:00
## 10      558            600      753            745      138 2013-01-01 06:00:00
## # i 336,766 more rows
```

Yes as it selects rows that contain "time" This is because select helpers default for ignore.case = TRUE You can change that by running the following code

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

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

Now it selects nothing!

6. Rename air_time to air_time_main to indicate units of measurement and move it to the beginning of the data frame

```
flights |>
  rename(air_time_main = air_time) |>
  relocate(air_time_main, .before = 1)
```

```
## # A tibble: 336,776 x 19
##    air_time_main  year month   day dep_time sched_dep_time dep_delay arr_time
##            <dbl> <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1           227  2013     1     1      517            515         2      830
##  2           227  2013     1     1      533            529         4      850
##  3           160  2013     1     1      542            540         2      923
##  4           183  2013     1     1      544            545        -1     1004
##  5           116  2013     1     1      554            600        -6      812
##  6           150  2013     1     1      554            558        -4      740
```

```
##  7          158  2013     1     1      555            600        -5      913
##  8           53  2013     1     1      557            600        -3      709
##  9          140  2013     1     1      557            600        -3      838
## 10          138  2013     1     1      558            600        -2      753
## # i 336,766 more rows
## # i 11 more variables: sched_arr_time <int>, arr_delay <dbl>, carrier <chr>,
## #   flight <int>, tailnum <chr>, origin <chr>, dest <chr>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

7. Why doesn't the following work, and what does the error mean?

```
#flights |>
  #select(tailnum) |>
  #arrange(arr_delay)
```

Error in arrange(select(flights, tailnum), arr_delay) : Caused by error: ! object 'arr_delay' not found When you only select tailnum, select outputs a copy of the flights data frame with just the tailnum column, which is the input to the arrange function, which now can't find the arr_delay column to sort by

3.4 The pipe

Real power comes from combining multiple verbs

Find fastest flights from Houston's IAH airport

```
flights |>
  filter(dest == "IAH") |>
  mutate(speed = distance / air_time * 60) |>
  select(year:day, dep_time, carrier, flight, speed) |>
  arrange(desc(speed))
```

```
## # A tibble: 7,198 x 7
##     year month   day dep_time carrier flight speed
##    <int> <int> <int>    <int> <chr>    <int> <dbl>
##  1  2013     7     9      707 UA         226  522.
##  2  2013     8    27     1850 UA        1128  521.
##  3  2013     8    28      902 UA        1711  519.
##  4  2013     8    28     2122 UA        1022  519.
##  5  2013     6    11     1628 UA        1178  515.
##  6  2013     8    27     1017 UA         333  515.
##  7  2013     8    27     1205 UA        1421  515.
##  8  2013     8    27     1758 UA         302  515.
##  9  2013     9    27      521 UA         252  515.
## 10  2013     8    28      625 UA         559  515.
## # i 7,188 more rows
```

Pipe makes this code very easy to read

If we didn't have the pipe… could nest function calls

```
arrange(
  select(
    mutate(
      filter(
        flights,
        dest == "IAH"
      ),
      speed = distance / air_time * 60
    ),
    year:day, dep_time, carrier, flight, speed
  ),
  desc(speed)
)
```

```
## # A tibble: 7,198 x 7
##     year month   day dep_time carrier flight speed
##    <int> <int> <int>    <int> <chr>    <int> <dbl>
##  1  2013     7     9      707 UA         226  522.
##  2  2013     8    27     1850 UA        1128  521.
##  3  2013     8    28      902 UA        1711  519.
##  4  2013     8    28     2122 UA        1022  519.
##  5  2013     6    11     1628 UA        1178  515.
##  6  2013     8    27     1017 UA         333  515.
##  7  2013     8    27     1205 UA        1421  515.
##  8  2013     8    27     1758 UA         302  515.
##  9  2013     9    27      521 UA         252  515.
## 10  2013     8    28      625 UA         559  515.
## # i 7,188 more rows
```

Or use bunch of intermediate objects (the pandas way)

```
flights1 <- filter(flights, dest == "IAH")
flights2 <- mutate(flights1, speed = distance / air_time * 60)
flights3 <- select(flights2, year:day, dep_time, carrier, flight, speed)
arrange(flights3, desc(speed))
```

```
## # A tibble: 7,198 x 7
##     year month   day dep_time carrier flight speed
##    <int> <int> <int>    <int> <chr>    <int> <dbl>
##  1  2013     7     9      707 UA         226  522.
##  2  2013     8    27     1850 UA        1128  521.
##  3  2013     8    28      902 UA        1711  519.
##  4  2013     8    28     2122 UA        1022  519.
##  5  2013     6    11     1628 UA        1178  515.
##  6  2013     8    27     1017 UA         333  515.
##  7  2013     8    27     1205 UA        1421  515.
##  8  2013     8    27     1758 UA         302  515.
##  9  2013     9    27      521 UA         252  515.
## 10  2013     8    28      625 UA         559  515.
## # i 7,188 more rows
```

Pipe code is easier to write/read Shortcut is Cmd + Shift + M

Default is magritrr %>%, part of tidyverse

```r
mtcars %>%
  group_by(cyl) %>%
  summarize(n = n())
```

```
## # A tibble: 3 x 2
##     cyl     n
##   <dbl> <int>
## 1     4    11
## 2     6     7
## 3     8    14
```

For simple cases, behaves identical to base pipe |>, but base pipe is part of base R and is simpler

3.5 Groups

So far worked with rows and columns, dplyr even more powerful when work with groups

3.5.1 group_by()

Use to divide dataset into groups meaningful for analysis

```r
flights |>
  group_by(month)
```

```
## # A tibble: 336,776 x 19
## # Groups:   month [12]
##     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
## # i 336,766 more rows
## # i 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>
```

Doesn't change data but output shows Groups: month [12], which means subsequent operations will work "by month" group_by() adds this class to data frame, which changes behavior of subsequent verbs

3.5.2 summarize()

Most important grouped operation is a summary, which if used to calculate one summary statistic reduces each group to one row

Compute average departure delay by month

```
flights |>
  group_by(month) |>
  summarize(
    avg_delay = mean(dep_delay)
  )
```

```
## # A tibble: 12 x 2
##    month avg_delay
##    <int>     <dbl>
##  1     1        NA
##  2     2        NA
##  3     3        NA
##  4     4        NA
##  5     5        NA
##  6     6        NA
##  7     7        NA
##  8     8        NA
##  9     9        NA
## 10    10        NA
## 11    11        NA
## 12    12        NA
```

Whoops, everything is NA! This is because each month contained missing data for some flights, for now, tell mean() to ignore missing values with na.rm = TRUE

```
flights |>
  group_by(month) |>
  summarize(
    avg_delay = mean(dep_delay, na.rm = TRUE)
  )
```

```
## # A tibble: 12 x 2
##    month avg_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
```

Can create as many summaries as want in a single call, one very useful is n() which returns number of rows in each group

```r
flights |>
  group_by(month) |>
  summarize(
    avg_delay = mean(dep_delay, na.rm = TRUE),
    n = n()
  )
```

```
## # A tibble: 12 x 3
##    month avg_delay     n
##    <int>     <dbl> <int>
##  1     1     10.0  27004
##  2     2     10.8  24951
##  3     3     13.2  28834
##  4     4     13.9  28330
##  5     5     13.0  28796
##  6     6     20.8  28243
##  7     7     21.7  29425
##  8     8     12.6  29327
##  9     9      6.72 27574
## 10    10      6.24 28889
## 11    11      5.44 27268
## 12    12     16.6  28135
```

3.5.3 The slice_ functions

Five functions allow for extracting specific rows within each group 1. df |> slice_head(n = 1) takes first row from each group 2. df |> slice_tail(n = 1) takes last row from each group 3. df |> slice_min(x, n = 1) takes row with smallest value of column x 4. df |> slice_max(x, n = 1) takes row with largest value of column x 5. df |> slice_sample(x, n = 1) takes one random row Can vary n for more rows, or use prop = 0.1 to select 10% of rows in each group

Find flights most delayed upon arrival at each destination

```r
flights |>
  group_by(dest) |>
  slice_max(arr_delay, n = 1) |>
  relocate(dest)
```

```
## # A tibble: 108 x 19
## # Groups:   dest [105]
##    dest   year month   day dep_time sched_dep_time dep_delay arr_time
##    <chr> <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1 ABQ    2013     7    22     2145           2007        98      132
##  2 ACK    2013     7    23     1139            800       219     1250
##  3 ALB    2013     1    25      123           2000       323      229
##  4 ANC    2013     8    17     1740           1625        75     2042
##  5 ATL    2013     7    22     2257            759       898      121
##  6 AUS    2013     7    10     2056           1505       351     2347
##  7 AVL    2013     8    13     1156            832       204     1417
##  8 BDL    2013     2    21     1728           1316       252     1839
##  9 BGR    2013    12     1     1504           1056       248     1628
## 10 BHM    2013     4    10       25           1900       325      136
## # i 98 more rows
```

```
## # i 11 more variables: sched_arr_time <int>, arr_delay <dbl>, carrier <chr>,
## #   flight <int>, tailnum <chr>, origin <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

There are 105 destinations but get 108 rows... Why? slice_min() and slice_max() keep tied values, if want exactly one row per group use with_ties = FALSE

This is similar to computing max delay with summarize() but get whole corresponding row instead of single summary statistic

3.5.4 Grouping by multiple variables

Can make a group for each date

```
daily <- flights |>
  group_by(year, month, day)
daily
```

```
## # A tibble: 336,776 x 19
## # Groups:   year, month, day [365]
##     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
## # i 336,766 more rows
## # i 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>
```

When summarize tibbled group each summary peels off last group

```
daily_flights <- daily |>
  summarize(n = n())
```

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

If happy with this behavior can explicitly request it to suppress message

```
daily_flights <- daily |>
  summarize(
    n = n(),
    .groups = "drop_last"
  )
```

Alternatively can use “drop” to drop all grouping or “keep” to preserve groups

3.5.5 Ungrouping

Use ungroup() to remove grouping without using summarize()

```
daily |>
  ungroup()
```

```
## # A tibble: 336,776 x 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
## # i 336,766 more rows
## # i 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>
```

Now try summarizing an ungrouped data frame

```
daily |>
  ungroup() |>
  summarize(
    avg_delay = mean(dep_delay, na.rm = TRUE),
    flights = n()
  )
```

```
## # A tibble: 1 x 2
##   avg_delay flights
##       <dbl>   <int>
## 1      12.6  336776
```

Get one row because dplyr treats all rows in ungrouped data frame as one group

3.5.6 .by

Can now also use .by argument to group within a single function

```
flights |>
  summarize(
    delay = mean(dep_delay, na.rm = TRUE),
    n = n(),
    .by = month
  )
```

```
## # A tibble: 12 x 3
##    month delay     n
##    <int> <dbl> <int>
##  1     1 10.0  27004
##  2    10  6.24 28889
##  3    11  5.44 27268
##  4    12 16.6  28135
##  5     2 10.8  24951
##  6     3 13.2  28834
##  7     4 13.9  28330
##  8     5 13.0  28796
##  9     6 20.8  28243
## 10     7 21.7  29425
## 11     8 12.6  29327
## 12     9  6.72 27574
```

Or if want to group by multiple variables

```r
flights |>
  summarize(
    delay = mean(dep_delay, na.rm = TRUE),
    n = n(),
    .by = c(origin, dest)
  )
```

```
## # A tibble: 224 x 4
##    origin dest  delay     n
##    <chr>  <chr> <dbl> <int>
##  1 EWR    IAH   11.8   3973
##  2 LGA    IAH    9.06  2951
##  3 JFK    MIA    9.34  3314
##  4 JFK    BQN    6.67   599
##  5 LGA    ATL   11.4  10263
##  6 EWR    ORD   14.6   6100
##  7 EWR    FLL   13.5   3793
##  8 LGA    IAD   16.7   1803
##  9 JFK    MCO   10.6   5464
## 10 LGA    ORD   10.7   8857
## # i 214 more rows
```

.by works with all verbs, has advantage of not needing to use .groups to suppress grouping message or ungroup() when you are done

3.5.7 Exercises

1. Which carrier has the worst average delays?

```r
flights |>
  summarize(
    delay = mean(dep_delay, na.rm = TRUE),
    .by = carrier
    ) |>
  arrange(desc(delay))
```

```
## # A tibble: 16 x 2
##    carrier delay
##    <chr>   <dbl>
##  1 F9      20.2
##  2 EV      20.0
##  3 YV      19.0
##  4 FL      18.7
##  5 WN      17.7
##  6 9E      16.7
##  7 B6      13.0
##  8 VX      12.9
##  9 OO      12.6
## 10 UA      12.1
## 11 MQ      10.6
## 12 DL       9.26
## 13 AA       8.59
## 14 AS       5.80
## 15 HA       4.90
## 16 US       3.78
```

Challenge: can you disentangle effects of bad airports vs bad carriers?

```
# Hint given by problem
flights |>
  summarize(n(), .by = c(carrier, dest))
```

```
## # A tibble: 314 x 3
##    carrier dest  `n()`
##    <chr>   <chr> <int>
##  1 UA      IAH    6924
##  2 AA      MIA    7234
##  3 B6      BQN     599
##  4 DL      ATL   10571
##  5 UA      ORD    6984
##  6 B6      FLL    6563
##  7 EV      IAD    4048
##  8 B6      MCO    6472
##  9 AA      ORD    6059
## 10 B6      PBI    3161
## # i 304 more rows
```

```
# Get average delays for carrier/airport combo
by_carrier <- flights |>
  summarize(
    carrier_delay = mean(dep_delay, na.rm = TRUE),
    n = n(),
    .by = c(carrier, dest)
    )

# Get average delays for each airport
by_airport <- flights |>
  summarize(
    airport_delay = mean(dep_delay, na.rm = TRUE),
```

```r
    .by = dest
  )

# Merge and compare difference between carrier and airport
left_join(by_carrier, by_airport, by = join_by(dest)) |>
  mutate(diff = carrier_delay - airport_delay) |>
  relocate(diff, .after = dest) |>
  arrange(desc(diff))
```

```
## # A tibble: 314 x 6
##    carrier dest   diff carrier_delay     n airport_delay
##    <chr>   <chr> <dbl>         <dbl> <int>         <dbl>
##  1 UA      STL    61.5          77.5     2         16.0
##  2 OO      ORD    53.4          67       1         13.6
##  3 OO      DTW    49.2          61       2         11.8
##  4 UA      RDU    47.6          60       1         12.4
##  5 EV      PBI    35.7          48.7     6         13.0
##  6 WN      MSY    19.1          33.4   298         14.2
##  7 9E      BGR    14.5          34       1         19.5
##  8 9E      CLT    14.0          23.2   291          9.22
##  9 EV      CLT    13.9          23.1  2508          9.22
## 10 EV      TYS    13.3          41.8   323         28.5
## # i 304 more rows
```

Using a join we can compare a carrier's performance at each airport to the average delay at an airport, however, as you can see, some of these carriers have relatively few flights to a certain airport, so we can't disentangle in all cases

Let's drop cases with relatively few flights

```r
left_join(by_carrier, by_airport, by = join_by(dest)) |>
  mutate(diff = carrier_delay - airport_delay) |>
  relocate(diff, .after = dest) |>
  arrange(desc(diff)) |>
  filter(n >= 10)
```

```
## # A tibble: 270 x 6
##    carrier dest   diff carrier_delay     n airport_delay
##    <chr>   <chr> <dbl>         <dbl> <int>         <dbl>
##  1 WN      MSY   19.1          33.4   298         14.2
##  2 9E      CLT   14.0          23.2   291          9.22
##  3 EV      CLT   13.9          23.1  2508          9.22
##  4 EV      TYS   13.3          41.8   323         28.5
##  5 9E      CLE   12.2          25.6   349         13.4
##  6 UA      BQN   11.5          23.9   297         12.4
##  7 9E      DFW   11.0          19.7   379          8.68
##  8 EV      DCA   11.0          21.3  1717         10.3
##  9 9E      ORD    9.97         23.5  1056         13.6
## 10 B6      SLC    9.93         19.0   365          9.03
## # i 260 more rows
```

Here we can confirm WN, EV, and 9E are among some of the worst performing carriers in terms of delays

2. Find the flights that are most delayed upon departure from each destination

```
flights |>
  summarise(delay = mean(dep_delay, na.rm = TRUE), .by = dest) |>
  arrange(desc(delay))
```

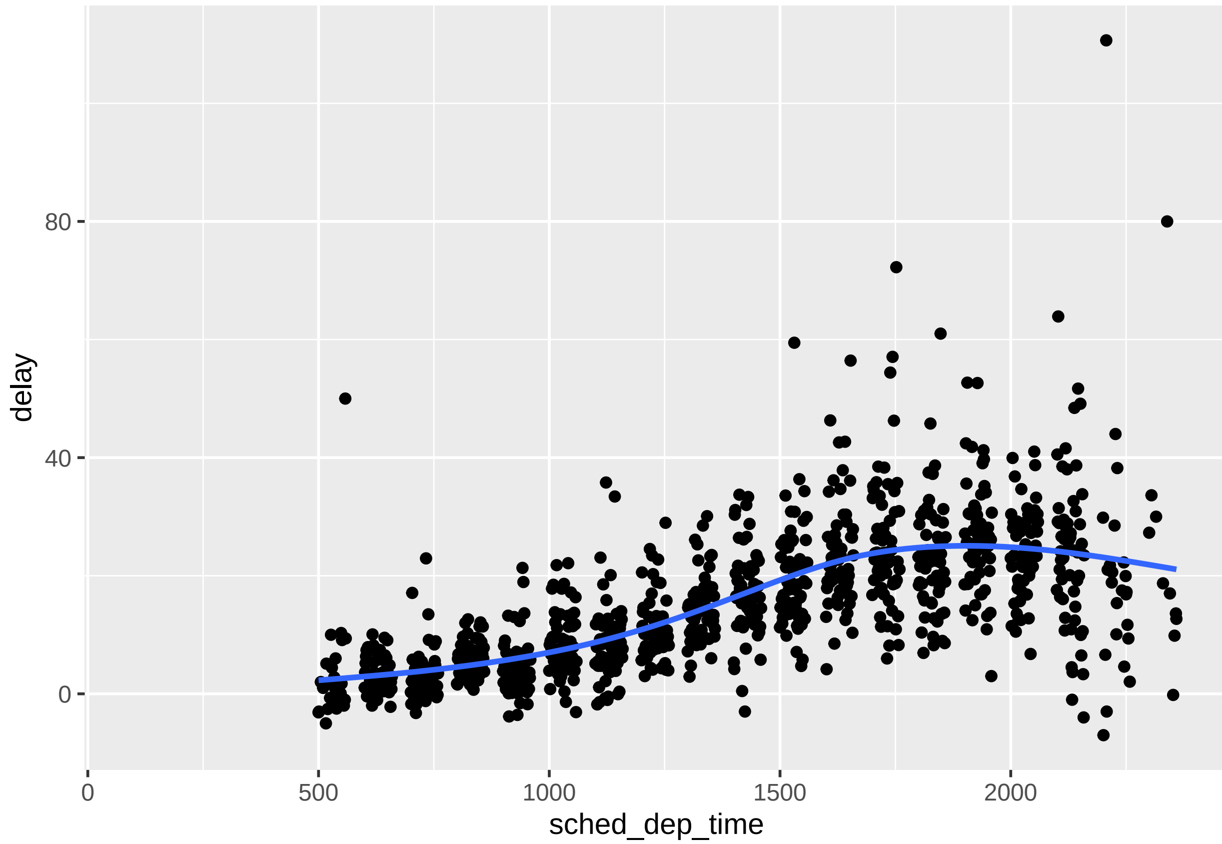
```
## # A tibble: 105 x 2
##    dest  delay
##    <chr> <dbl>
##  1 CAE    35.6
##  2 TUL    34.9
##  3 OKC    30.6
##  4 BHM    29.7
##  5 TYS    28.5
##  6 JAC    26.5
##  7 DSM    26.2
##  8 RIC    23.6
##  9 ALB    23.6
## 10 MSN    23.6
## # i 95 more rows
```

3. How do delays vary over the course of the day? Illustrate with a plot

```
flights |>
  summarise(
    delay = mean(dep_delay, na.rm = TRUE),
    .by = sched_dep_time
    ) |>
  ggplot(
    mapping = aes(
      x = sched_dep_time,
      y = delay
    )
) +
  geom_point(na.rm = TRUE) +
  geom_smooth(se = FALSE)
```

```
## `geom_smooth()` using method = 'gam' and formula = 'y ~ s(x, bs = "cs")'
```

```
## Warning: Removed 1 row containing non-finite outside the scale range
## (`stat_smooth()`).
```



4. What happens if you supply a negative n to slice_min() and friends? It will grab everything except n rows So slice_min(-1) will grab all rows except the smallest row slice_max(-1) will grab all rows except the largest row

```
flights |>
  group_by(dest) |>
  relocate(dest) |>
  slice_min(dep_delay, n = -1)
```

```
## # A tibble: 336,760 x 19
## # Groups:   dest [103]
##    dest   year month   day dep_time sched_dep_time dep_delay arr_time
##    <chr> <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1 ABQ    2013    11    20     1948           2000       -12     2236
##  2 ABQ    2013     9    10     1949           2001       -12     2225
##  3 ABQ    2013    11     1     1950           2000       -10     2226
##  4 ABQ    2013    11     5     1950           2000       -10     2310
##  5 ABQ    2013    11    15     1950           2000       -10     2304
##  6 ABQ    2013    12    28     1951           2001       -10     2243
##  7 ABQ    2013     9    18     1951           2001       -10     2210
##  8 ABQ    2013    10    19     1950           1959        -9     2236
##  9 ABQ    2013    10    26     1950           1959        -9     2217
## 10 ABQ    2013    11    10     1951           2000        -9     2258
## # i 336,750 more rows
## # i 11 more variables: sched_arr_time <int>, arr_delay <dbl>, carrier <chr>,
```

```
## #   flight <int>, tailnum <chr>, origin <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

```
flights |>
  group_by(dest) |>
  relocate(dest) |>
  slice_max(dep_delay, n = -1)
```

```
## # A tibble: 336,762 x 19
## # Groups:   dest [103]
##    dest   year month   day dep_time sched_dep_time dep_delay arr_time
##    <chr> <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1 ABQ    2013    12    14     2223           2001       142      133
##  2 ABQ    2013    12    17     2220           2001       139      120
##  3 ABQ    2013     7    30     2212           2007       125       57
##  4 ABQ    2013     9     2     2212           2007       125       48
##  5 ABQ    2013     7    23     2206           2007       119      116
##  6 ABQ    2013     5    10     2158           2001       117       53
##  7 ABQ    2013     8     9     2159           2007       112       37
##  8 ABQ    2013     6     8     2148           2001       107        9
##  9 ABQ    2013     9    12     2147           2001       106       25
## 10 ABQ    2013    10    15     2146           2001       105      106
## # i 336,752 more rows
## # i 11 more variables: sched_arr_time <int>, arr_delay <dbl>, carrier <chr>,
## #   flight <int>, tailnum <chr>, origin <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

5. Explain what count() does in terms of the dplyr verbs, what does sort argument to count() do?

```
?count
```

It groups by the argument you feed it, and then returns n(), the number of rows If sort is TRUE, the largest groups will be shown on top

6. Suppose we have the following tiny data frame:

```
df <- tibble(
  x = 1:5,
  y = c("a", "b", "a", "a", "b"),
  z = c("K", "K", "L", "L", "K")
)
```

a) Write down what you think the output looks like, then check if you are correct and describe what group_by() does

```
df |>
  group_by(y)
```

```
## # A tibble: 5 x 3
## # Groups:   y [2]
##       x y     z
```

```
##   <int> <chr> <chr>
## 1     1 a     K
## 2     2 b     K
## 3     3 a     L
## 4     4 a     L
## 5     5 b     K
```

tibble will show Groups: y [2] because there are two groups, "a" and "b" group_by() creates groups for each distinct value in y, but it only creates those groups, so the rest of the tibble looks the same

b) Predict the output, then check if correct, describe what arrange() does, comment how its different from group_by() in a)

```r
df |>
  arrange(y)
```

```
## # A tibble: 5 x 3
##       x y     z
##   <int> <chr> <chr>
## 1     1 a     K
## 2     3 a     L
## 3     4 a     L
## 4     2 b     K
## 5     5 b     K
```

tibble will have re-ordered rows, with rows where x is 1, 3, and 4 appearing first because the tibble is sorted by column y Unlike part (a, the rows are manipulated and no groups are created

c) Predict output, describe what pipeline does

```r
df |>
  group_by(y) |>
  summarize(mean_x = mean(x))
```

```
## # A tibble: 2 x 2
##   y     mean_x
##   <chr>  <dbl>
## 1 a       2.67
## 2 b       3.5
```

y mean_x a 8/3 b 7/2 Takes the mean of the x values for each group in y, which means taking the mean of all the x values for rows where y is "a" and then doing same for "b"

d) Do the same then comment on what the message says

```r
df |>
  group_by(y, z) |>
  summarize(mean_x = mean(x))
```

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

```
## # A tibble: 3 x 3
## # Groups:   y [2]
##   y     z     mean_x
##   <chr> <chr>  <dbl>
## 1 a     K        1
## 2 a     L        3.5
## 3 b     K        3.5
```

There will be three groups: 1) y = a, z = K 2) y = a, Z = L 3) y = b, z = K Then within each group the mean value of x will be computed y z mean_x a K 2 a L 4 b K 3.5 After computing the mean, summarise() will return the dataset with the last group (in this case z) dropped, so to override this and keep all groups use .groups = "keep"

e) Do the same, how is output different from d)

```
df |>
  group_by(y, z) |>
  summarize(mean_x = mean(x), .groups = "drop")
```

```
## # A tibble: 3 x 3
##   y     z     mean_x
##   <chr> <chr>  <dbl>
## 1 a     K        1
## 2 a     L        3.5
## 3 b     K        3.5
```

The output will be exactly the same as d), except the resulting tibble will have no groups, so if you were to chain another summarize with a pipe, it will calculate the mean for all of the data

```
df |>
  group_by(y, z) |>
  summarize(mean_x = mean(x), .groups = "drop") |>
  summarize(mean_x = mean(mean_x))
```

```
## # A tibble: 1 x 1
##   mean_x
##    <dbl>
## 1   2.67
```

f) Do the same, how are the outputs of the two pipelines different

```
df |>
  group_by(y, z) |>
  summarize(mean_x = mean(x))
```

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

```
## # A tibble: 3 x 3
## # Groups:   y [2]
##   y     z     mean_x
```

```
##   <chr> <chr>  <dbl>
## 1 a     K        1
## 2 a     L        3.5
## 3 b     K        3.5
```

```
df |>
  group_by(y, z) |>
  mutate(mean_x = mean(x))
```

```
## # A tibble: 5 x 4
## # Groups:   y, z [3]
##       x y     z     mean_x
##   <int> <chr> <chr>  <dbl>
## 1     1 a     K        1
## 2     2 b     K        3.5
## 3     3 a     L        3.5
## 4     4 a     L        3.5
## 5     5 b     K        3.5
```

The first pipeline is the same as d) and it will not have column x in the output, its dimensions will be 3x3 The second pipeline creates a new column with the mean x value for each group, so the resulting dimension will be 5x4 since each original row will be retained except it will also have the mean value for its group at the end

3.6 Case study: aggregates and sample size

Whenever do any aggregation, good idea to include a count, n(), so you make sure you aren't drawing conclusions from small amounts of data

Demonstrate this with baseball data from Lahman package, specifically will compare batting average (hit / at bat)

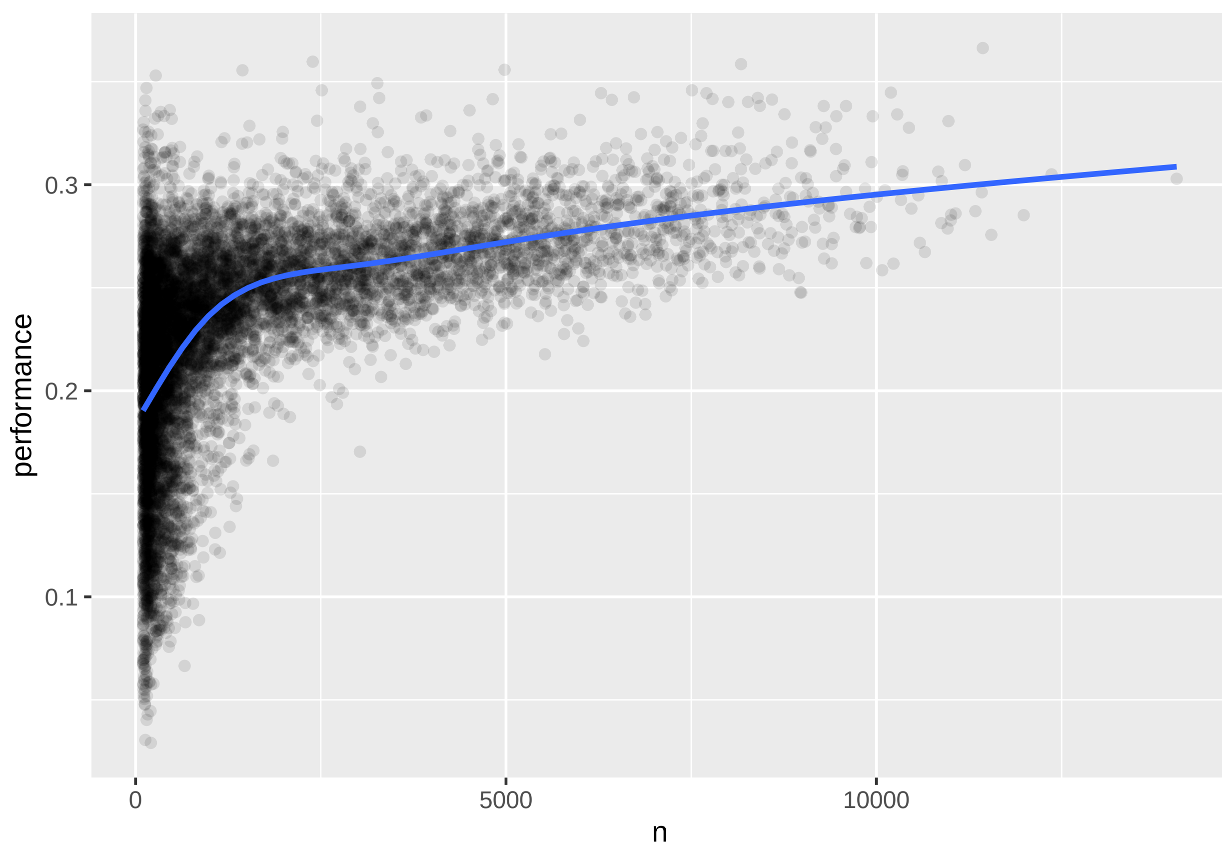
```
batters <- Lahman::Batting |>
  group_by(playerID) |>
  summarize(
    performance = sum(H, na.rm = TRUE) / sum(AB, na.rm = TRUE),
    n = sum(AB, na.rm = TRUE)
  )
batters
```

```
## # A tibble: 20,730 x 3
##    playerID  performance     n
##    <chr>           <dbl> <int>
##  1 aardsda01      0          4
##  2 aaronha01      0.305  12364
##  3 aaronto01      0.229    944
##  4 aasedo01       0          5
##  5 abadan01       0.0952    21
##  6 abadfe01       0.111      9
##  7 abadijo01      0.224     49
##  8 abbated01      0.254   3044
##  9 abbeybe01      0.169    225
## 10 abbeych01      0.281   1756
## # i 20,720 more rows
```

When plot batting average (performance) against number of opportunities to hit the ball (n), see two patterns 1. Variation in performance is larger among players with fewer at-bats, shape is very characteristic, whenever you plot a mean (or other summary statistics) vs group size, will see variation decreases as sample size increases (law of large numbers) 2. Positive correlation between skill (performance) and at-bats (n) because teams want to give best batters most opportunities to hit

```
batters |>
  filter(n > 100) |>
  ggplot(aes(x = n, y = performance)) +
  geom_point(alpha = 1 / 10) +
  geom_smooth(se = FALSE)
```

```
## `geom_smooth()` using method = 'gam' and formula = 'y ~ s(x, bs = "cs")'
```



Note handy integration between dply and ggplot2, just have to remember to switch from |> to +

This concept also has implications for ranking, if naively sort on desc(performance), will find many players with few at-bats, not necessarily the most skilled players

```
batters |>
  arrange(desc(performance))
```

```
## # A tibble: 20,730 x 3
##    playerID  performance     n
##    <chr>           <dbl> <int>
##  1 abramge01           1     1
```

```
##   2 alberan01           1      1
##   3 banisje01           1      1
##   4 bartocl01           1      1
##   5 bassdo01            1      1
##   6 birasst01           1      2
##   7 bruneju01           1      1
##   8 burnscb01           1      1
##   9 cammaer01           1      1
##  10 campsh01            1      1
## # i 20,720 more rows
```

3.6.1 Extension

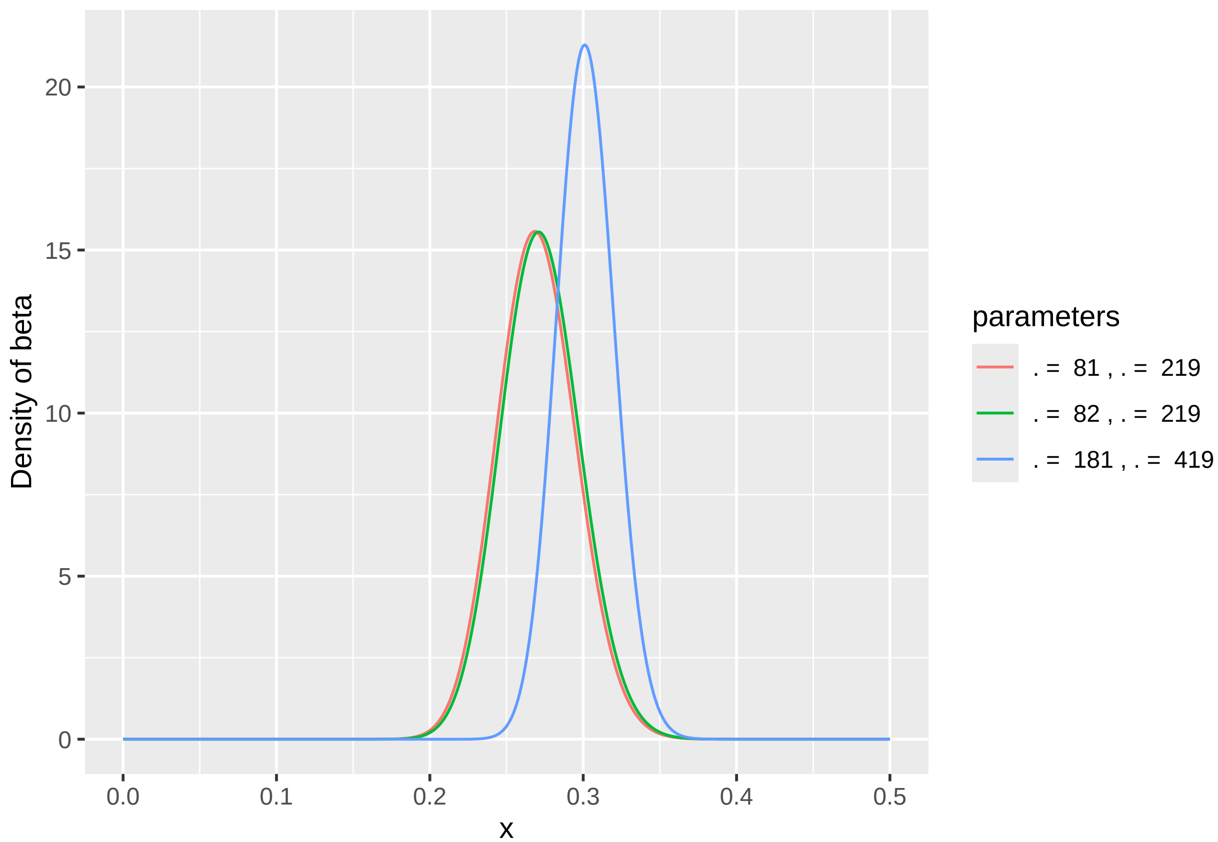
Understanding empirical Bayesian estimation (using baseball statistics)

You can't always throw out data that doesn't meet some minimum, losing information

One approach is beta distribution (probability distribution of probabilities) - If a player's expected pre-season batting average is 0.27 (which will be represented by beta distribution) then bats 100 out of 300, will update beta distribution to be 0.303 less than naive estimate of 0.333 but higher than season-start - Can also see that hitting on first bat (2nd curve) is barely noticable because one hit doesn't tell us much - Helps represent prior expectations and updating based on new evidence

```r
# Will plot three curves:
# 1) alpha = 81, beta = 219
# 2) alpha = 82, beta = 219
# 3) alpha = 181, beta = 419
tibble(a = c(81, 82, 81 + 100), b = c(219, 219, 219 + 200)) |>
  rowwise() |>  # Go row by row
  mutate(data = list(tibble(  # Creates column data with tibbles as value
    # List serves as container to hold nested tibbles in parent table
    # Each parent row values gets broadcasted to the nested rows in its tibble
    x = seq(0, 1, 0.001),  # Points at increments 0.001
    y = dbeta(x, a, b),  # Apply to the beta distribution function
    parameters = paste("\u03B1 = ", a, ", \u03B2 = ", b)  # For label
  ))) |>
  unnest(data) |>  # Expands tibbles to data frame
  # Turning parameter into factor will preserve legend order
  mutate(parameters = factor(parameters, levels = unique(parameters))) |>

  # Plot distributions
  ggplot(aes(x, y, color = parameters)) +
  geom_line(na.rm = TRUE) +
  xlim(0, 0.5) +
  ylab("Density of beta")
```



Related method is empirical Bayesian estimation, where beta distribution is used to improve large set of estimates, and as long as you have a lot of examples, you don't need prior expectations

Prepare and clean data first

```
career <- Lahman::Batting |>
  filter(AB > 0) |>
  # Get rid of pitchers (weak batters)
  anti_join(Lahman::Pitching, by = "playerID") |>
  summarize(H = sum(H), AB = sum(AB), .by = playerID) |>
  mutate(average = H / AB)

# Use names as identifier instead
career <- Lahman::People |>
  as_tibble() |>
  select(playerID, nameFirst, nameLast) |>
  unite(name, nameFirst, nameLast, sep = " ") |>  # Pastes columns into one
  inner_join(career, by = "playerID") |>
  select(-playerID)

career
```

```
## # A tibble: 10,056 x 4
##    name                   H    AB average
##    <chr>              <int> <int>   <dbl>
##  1 Hank Aaron          3771 12364   0.305
##  2 Tommie Aaron         216   944   0.229
```

```
##   3 Andy Abad             2    21  0.0952
##   4 John Abadie          11    49  0.224
##   5 Ed Abbaticchio      772  3044  0.254
##   6 Fred Abbott         107   513  0.209
##   7 Jeff Abbott         157   596  0.263
##   8 Kurt Abbott         523  2044  0.256
##   9 Ody Abbott           13    70  0.186
##  10 Frank Abercrombie     0     4  0
## # i 10,046 more rows
```

Who are best players in history? Let's check players with highest batting average…

```
career |>
  arrange(desc(average)) |>
  head(5)
```

```
## # A tibble: 5 x 4
##   name                  H    AB average
##   <chr>             <int> <int>   <dbl>
## 1 Jeff Banister         1     1       1
## 2 Doc Bass              1     1       1
## 3 Steve Biras           2     2       1
## 4 C. B. Burns           1     1       1
## 5 Jackie Gallagher      1     1       1
```

Probably just got lucky…

What about the worst?

```
career |>
  arrange(average) |>
  head(5)
```

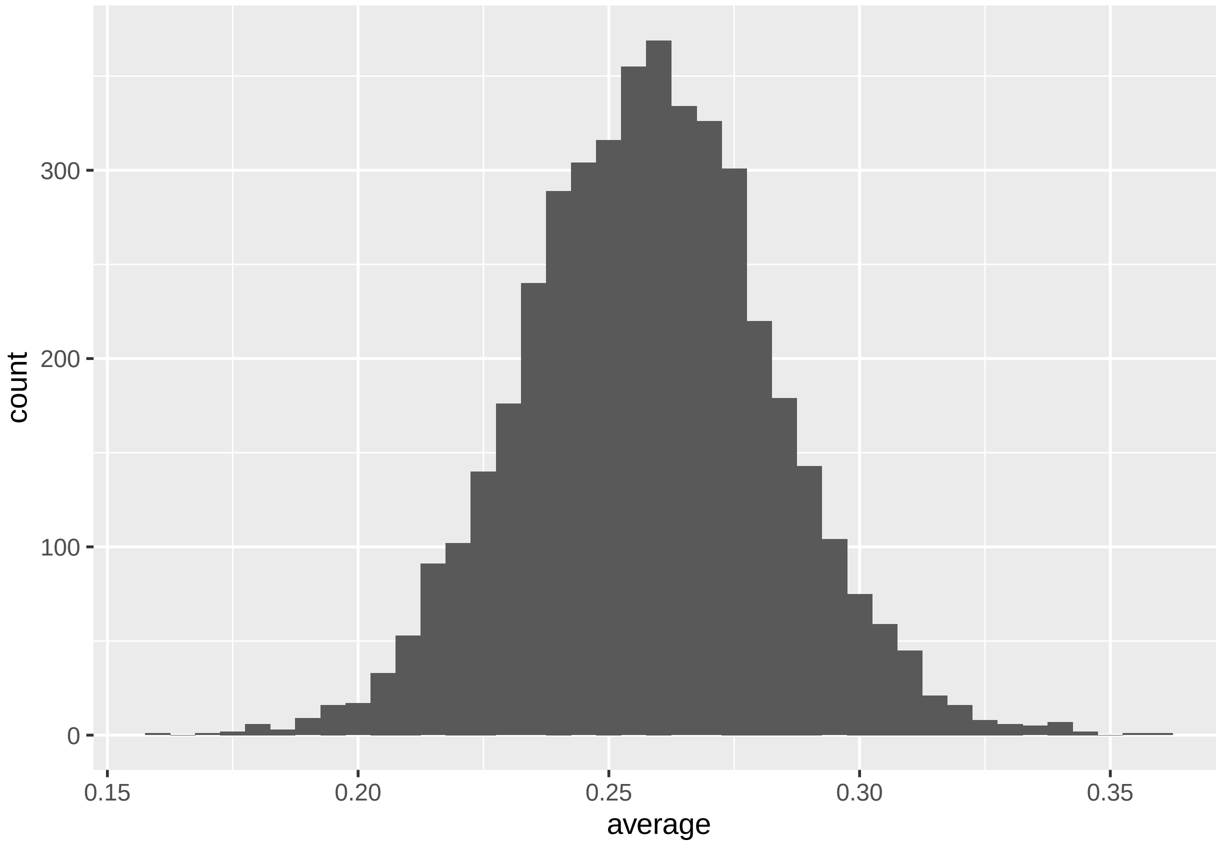
```
## # A tibble: 5 x 4
##   name                    H    AB average
##   <chr>               <int> <int>   <dbl>
## 1 Frank Abercrombie       0     4       0
## 2 Horace Allen            0     7       0
## 3 Pete Allen              0     4       0
## 4 Walter Alston           0     1       0
## 5 Trey Amburgey           0     4       0
```

Average here is not a great estimate

Step 1: estimate a prior from all your data

Let's filter out noise (< 500 at bats) and look at the distribution of batting averages across players

```
career |>
  filter(AB >= 500) |>
  ggplot(aes(average)) +
  geom_histogram(binwidth = 0.005)
```



Step 1: estimate a beta prior using this data; estimating from data currently analyzing not typical, usually decide ahead of time Empirical Bayes is an approximation of more exact Bayesian methods, with the amount of data we have (a lot), it's very good

So far data looks good, if it had two or more peaks then might need mixture of betas/more complicated model

Need to pick alpha0 and beta0, the "hyper-parameters" of the model

Use fitdistr function from MASS to fit probability distribution to data

```
# Filter players we have good estimate of (just like graph)
career_filtered <- career |>
  filter(AB >= 500)

m <- MASS::fitdistr(
  career_filtered$average,  # Numeric vector with data
  dbeta,  # Function returning a density
  start = list(shape1 = 1, shape2 = 2)
  )  # List with parameters to be optimized
```

```
## Warning in densfun(x, parm[1], parm[2], ...): NaNs produced
```

```
alpha0 <- m$estimate[1]
beta0 <- m$estimate[2]

alpha0
```

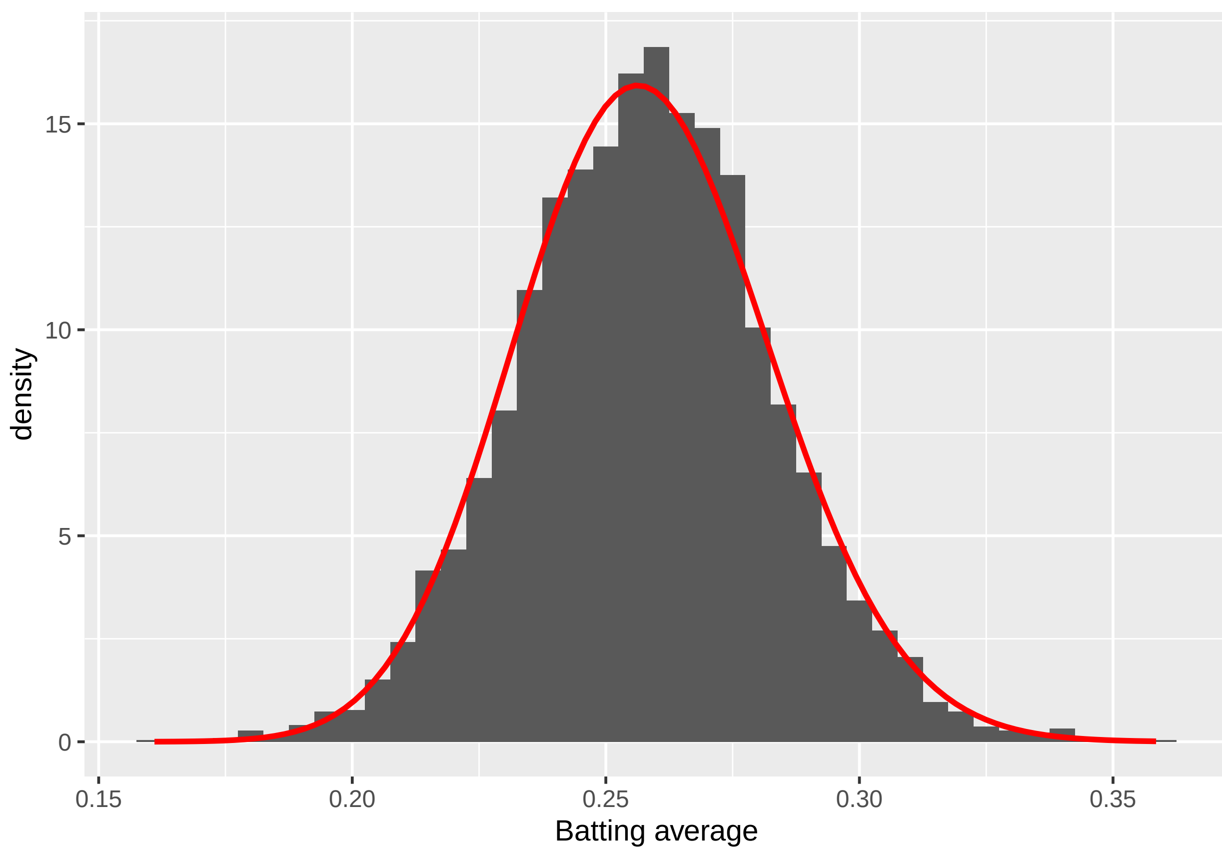
```
##   shape1
## 78.59667
```

```
beta0
```

```
##   shape2
## 226.1636
```

Lets fit to data

```
ggplot(data = career_filtered) +
  geom_histogram(
    mapping = aes(average, y = after_stat(density)),
    binwidth = 0.005
  ) +
  stat_function(
    fun = function(x) dbeta(x, alpha0, beta0),
    color = "red",
    linewidth = 1
  ) +
  xlab("Batting average")
```



Step 2: use that distribution as prior for each individual estimate

Start with overall prior and update based on individual evidence

This would yield a higher estimate for the batter who has 300 hits in 1000 AB's than the batter who has 4 hits in 10 AB's, whereas without this method the 4/10 would be considered higher

Perform calculation for all batters

```
career_eb <- career |>
  mutate(eb_estimate = (H + alpha0) / (AB + alpha0 + beta0))
```

Results:

Now we can ask who are the best batters?

```
career_eb |>
  arrange(desc(eb_estimate)) |>
  head(5)
```

```
## # A tibble: 5 x 5
##   name                    H    AB average eb_estimate
##   <chr>               <int> <int>   <dbl>       <dbl>
## 1 Rogers Hornsby       2930  8173   0.358       0.355
## 2 Shoeless Joe Jackson 1772  4981   0.356       0.350
## 3 Ed Delahanty         2597  7510   0.346       0.342
## 4 Billy Hamilton       2164  6283   0.344       0.340
## 5 Harry Heilmann       2660  7787   0.342       0.338
```

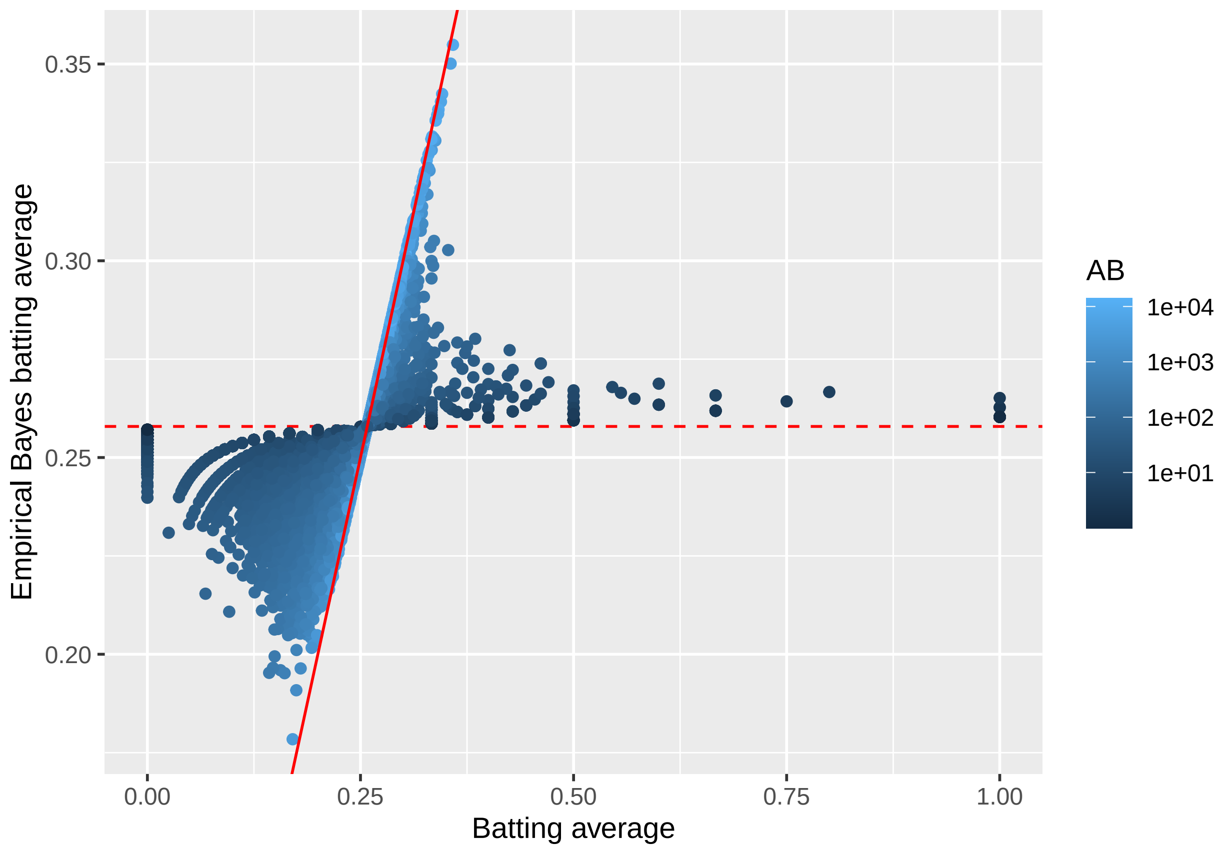
Who are the worst batters?

```
career_eb |>
  arrange(eb_estimate) |>
  head(5)
```

```
## # A tibble: 5 x 5
##   name                H    AB average eb_estimate
##   <chr>           <int> <int>   <dbl>       <dbl>
## 1 Bill Bergen       516  3028   0.170       0.178
## 2 Ray Oyler         221  1265   0.175       0.191
## 3 John Vukovich      90   559   0.161       0.195
## 4 John Humphries     52   364   0.143       0.195
## 5 George Baker       74   474   0.156       0.196
```

Empirical Bayes did not simply pick batters with one or two at-bats, instead finding players who bat well/poorly over a long career

```
ggplot(career_eb, aes(x = average, y = eb_estimate, color = AB)) +
  geom_hline(yintercept = alpha0 / (alpha0 + beta0), color = "red", lty = 2) +
  geom_point() +
  geom_abline(color = "red") +
  scale_color_gradient(transform = "log", breaks = 10 ^ (1:5)) +
  xlab("Batting average") +
  ylab("Empirical Bayes batting average")
```



Horizontal dashed line marks y = alpha0 / alpha0 + beta0, what we would guess would be someone's batting average with no evidence at all Points above it move down towards it, points below move up towards it Diagonal line is x = y, points near it didn't get shrunk by empirical Bayes, these are also the ones with the highest at-bats, light blue; there was enough evidence to believe the naive estimate This process is sometimes called shrinkage because moved all estimates towards the average, the less evidence the more movement Extraordinary outliers require extraordinary evidence!