Chapter 5

Data Transformation

Icaro Franco Hernandes

2022-09-26

Remember that to view a whole data set we can execute, for example, `view(nycflights13::flights)`. This is a tibble, and tibble $\neq$ table. Tibbles work better for the tidyverse. To check what kind of variable we are working with, we can use the following command:

```
typeof(nycflights13::flights$time_hour)
```

```
## [1] "double"
```

Filtering

Selecting all flights from January first:

```
nycflights13::flights%>%
  dplyr::filter(month==1,day==1)->jan1
#remeber that dplyr does not change the original dataset (always try to be as pure as possible).
```

If we want to also print the new data set, just put between parenthesis:

```
(nycflights13::flights%>%
  dplyr::filter(month==1,day==1)->jan1)
```

```
## # A tibble: 842 x 19
##     year month   day dep_time sched_de~1 dep_d~2 arr_t~3 sched~4 arr_d~5 carrier
##    <int> <int> <int>    <int>      <int>   <dbl>   <int>   <int>   <dbl> <chr>
##  1  2013     1     1      517        515       2     830     819      11 UA
##  2  2013     1     1      533        529       4     850     830      20 UA
##  3  2013     1     1      542        540       2     923     850      33 AA
##  4  2013     1     1      544        545      -1    1004    1022     -18 B6
##  5  2013     1     1      554        600      -6     812     837     -25 DL
##  6  2013     1     1      554        558      -4     740     728      12 UA
##  7  2013     1     1      555        600      -5     913     854      19 B6
##  8  2013     1     1      557        600      -3     709     723     -14 EV
##  9  2013     1     1      557        600      -3     838     846      -8 B6
## 10  2013     1     1      558        600      -2     753     745       8 AA
## # ... with 832 more rows, 9 more variables: flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>, and abbreviated variable names
## #   1: sched_dep_time, 2: dep_delay, 3: arr_time, 4: sched_arr_time,
## #   5: arr_delay
```

Boolean operators are: `&` for **and**, `|` for **or** and `!` for **is not**.

Inclusion operator: `%in%`. For example:

```r
nycflights13::flights%>%
  dplyr::filter(month %in% c(11,12))
```

```
## # A tibble: 55,403 x 19
##     year month   day dep_time sched_de~1 dep_d~2 arr_t~3 sched~4 arr_d~5 carrier
##    <int> <int> <int>    <int>      <int>   <dbl>   <int>   <int>   <dbl> <chr>
##  1  2013    11     1        5       2359       6     352     345       7 B6
##  2  2013    11     1       35       2250     105     123    2356      87 B6
##  3  2013    11     1      455        500      -5     641     651     -10 US
##  4  2013    11     1      539        545      -6     856     827      29 UA
##  5  2013    11     1      542        545      -3     831     855     -24 AA
##  6  2013    11     1      549        600     -11     912     923     -11 UA
##  7  2013    11     1      550        600     -10     705     659       6 US
##  8  2013    11     1      554        600      -6     659     701      -2 US
##  9  2013    11     1      554        600      -6     826     827      -1 DL
## 10  2013    11     1      554        600      -6     749     751      -2 DL
## # ... with 55,393 more rows, 9 more variables: flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>, and abbreviated variable names
## #   1: sched_dep_time, 2: dep_delay, 3: arr_time, 4: sched_arr_time,
## #   5: arr_delay
```

This will filter all flights that happened in november **or** december. The filter already excludes `NA` values.

Exercises 5.2.4

1

Find all flights that

- Had an arrival delay of two or more hours

```r
nycflights13::flights%>%
  dplyr::filter(arr_delay>=120)->f1
```

- Flew to Houston (`IAH` or `HOU`)

```r
nycflights13::flights%>%
  dplyr::filter(dest %in% c("IAH", "HOU"))->f2
```

- Were operated by United, American, or Delta

```r
nycflights13::flights%>%
  dplyr::filter(carrier %in% c("UA", "AA", "DL"))->f3
```

- Departed in Summer (July, August, and September)

```r
nycflights13::flights%>%
  dplyr::filter(month %in% c(7, 8, 9))->fsummer
```

- Arrived more than two hours late, but did not leave late

```r
nycflights13::flights%>%
  dplyr::filter(arr_delay>120,dep_time<=sched_dep_time)->f5
```

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

```r
nycflights13::flights%>%
  dplyr::filter(arr_delay>=60,air_time>30)->f6
```

- Departed between midnight and 6am (inclusive)

```
nycflights13::flights%>%
  dplyr::filter(hour %in% c(seq(0,6)))->f7

or
```

```
## function (e1, e2)  .Primitive("|")
```

```
nycflights13::flights%>%
  dplyr::filter(hour >= 0 & hour<= 6)->f71

# nycflights13::flights%>%
#   dplyr::filter(0 <= hour <= 6) -> this does not work!
```

2

Another useful `dplyr` filter helper is `between()`. What does it do? Can you use it to simplify the code needed to answer the precious questions?

According to the documentation, `between()` let us pick any values between to boundaries, and it is a shortcut for `x>= & x<=`. They would be useful in the cases where we had to filter for the summer months and the flights between midnight and 6 a.m.:

```
nycflights13::flights%>%
  dplyr::filter(between(month, 7, 9))->f8

nycflights13::flights%>%
  dplyr::filter(between(hour, 0, 6))->f9
```

3

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

```
nycflights13::flights%>%
  dplyr::filter(is.na(dep_time))%>%
  dplyr::summarise(n = dplyr::n())->na

na
```

```
## # A tibble: 1 x 1
##       n
##   <int>
## 1  8255
```

Using the count operator from `dplyr` we can see that 8255 flights have missing values for the departure time. This means that theses flights were canceled. If we do not have the departure time, we also cannot check the airtime, the departure delay and the arrival delay. **Remember this count operator (within the summarise function) from dplyr**.

4

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

```
NA^0
```

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

```
NA|TRUE
```

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

```
FALSE&NA
```

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

Since we are working with boolean operators here, the general rule is that R avoids the NA values and does let them contaminate the operation. It is different from the case if we calculate the average of some values with an NA (in that case it does contaminate the average).

```
v1<-c(1,1, NA)
mean(v1)
```

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

```
mean(v1, na.rm = T)
```

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

The command `na.rm=TRUE` discards the `NA` values from the calculation!

Arranging

Exercises 5.3.1

1

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

```
v<-tibble::tibble(
  x=(c(3,7,1,NA))
)

x=c(3,7,1,NA)

sort(x, decreasing = FALSE, na.last=FALSE)
```

```
## [1] NA  1  3  7
```

I used the base R command for sorting.

2

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

For the most delayed flights we just need to arrange in descending format:

```
nycflights13::flights%>%
  dplyr::arrange(desc(dep_delay))
```

```
## # A tibble: 336,776 x 19
##     year month   day dep_time sched_de~1 dep_d~2 arr_t~3 sched~4 arr_d~5 carrier
##    <int> <int> <int>    <int>      <int>   <dbl>   <int>   <int>   <dbl> <chr>
##  1  2013     1     9      641        900    1301    1242    1530    1272 HA
##  2  2013     6    15     1432       1935    1137    1607    2120    1127 MQ
##  3  2013     1    10     1121       1635    1126    1239    1810    1109 MQ
##  4  2013     9    20     1139       1845    1014    1457    2210    1007 AA
##  5  2013     7    22      845       1600    1005    1044    1815     989 MQ
```

```
## 6  2013     4    10     1100       1900     960    1342    2211     931 DL
## 7  2013     3    17     2321        810     911     135    1020     915 DL
## 8  2013     6    27      959       1900     899    1236    2226     850 DL
## 9  2013     7    22     2257        759     898     121    1026     895 DL
## 10  2013    12     5      756       1700     896    1058    2020     878 AA
## # ... with 336,766 more rows, 9 more variables: flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>, and abbreviated variable names
## #   1: sched_dep_time, 2: dep_delay, 3: arr_time, 4: sched_arr_time,
## #   5: arr_delay
```

And for the earliest departures we just need to arrange them:

```
nycflights13::flights%>%
  dplyr::arrange(dep_delay)
```

```
## # A tibble: 336,776 x 19
##     year month   day dep_time sched_de~1 dep_d~2 arr_t~3 sched~4 arr_d~5 carrier
##    <int> <int> <int>    <int>      <int>   <dbl>   <int>   <int>   <dbl> <chr>
##  1  2013    12     7     2040       2123     -43      40    2352      48 B6
##  2  2013     2     3     2022       2055     -33    2240    2338     -58 DL
##  3  2013    11    10     1408       1440     -32    1549    1559     -10 EV
##  4  2013     1    11     1900       1930     -30    2233    2243     -10 DL
##  5  2013     1    29     1703       1730     -27    1947    1957     -10 F9
##  6  2013     8     9      729        755     -26    1002     955       7 MQ
##  7  2013    10    23     1907       1932     -25    2143    2143       0 EV
##  8  2013     3    30     2030       2055     -25    2213    2250     -37 MQ
##  9  2013     3     2     1431       1455     -24    1601    1631     -30 9E
## 10  2013     5     5      934        958     -24    1225    1309     -44 B6
## # ... with 336,766 more rows, 9 more variables: flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>, and abbreviated variable names
## #   1: sched_dep_time, 2: dep_delay, 3: arr_time, 4: sched_arr_time,
## #   5: arr_delay
```

3

Sort `flights` to find the fastest (highest speed) flights

```
nycflights13::flights%>%
  dplyr::select(air_time,
              dplyr::everything())%>%
  dplyr::arrange(air_time)
```

```
## # A tibble: 336,776 x 19
##    air_time  year month   day dep_time sched_d~1 dep_d~2 arr_t~3 sched~4 arr_d~5
##       <dbl> <int> <int> <int>    <int>     <int>   <dbl>   <int>   <int>   <dbl>
##  1       20  2013     1    16     1355      1315      40    1442    1411      31
##  2       20  2013     4    13      537       527      10     622     628      -6
##  3       21  2013    12     6      922       851      31    1021     954      27
##  4       21  2013     2     3     2153      2129      24    2247    2224      23
##  5       21  2013     2     5     1303      1315     -12    1342    1411     -29
##  6       21  2013     2    12     2123      2130      -7    2211    2225     -14
##  7       21  2013     3     2     1450      1500     -10    1547    1608     -21
##  8       21  2013     3     8     2026      1935      51    2131    2056      35
##  9       21  2013     3    18     1456      1329      87    1533    1426      67
```

```
## 10       21  2013     3    19     2226      2145      41    2305    2246      19
## # ... with 336,766 more rows, 9 more variables: carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>, and abbreviated variable names
## #   1: sched_dep_time, 2: dep_delay, 3: arr_time, 4: sched_arr_time,
## #   5: arr_delay
```

4

Which flights traveled the farthest? Which traveled the shortest?

The ones traveled the farthest:

```
nycflights13::flights%>%
  dplyr::select(distance,
                dplyr::everything())%>%
  dplyr::arrange(desc(distance))
```

```
## # A tibble: 336,776 x 19
##    distance  year month   day dep_time sched_d~1 dep_d~2 arr_t~3 sched~4 arr_d~5
##       <dbl> <int> <int> <int>    <int>     <int>   <dbl>   <int>   <int>   <dbl>
##  1     4983  2013     1     1      857       900      -3    1516    1530     -14
##  2     4983  2013     1     2      909       900       9    1525    1530      -5
##  3     4983  2013     1     3      914       900      14    1504    1530     -26
##  4     4983  2013     1     4      900       900       0    1516    1530     -14
##  5     4983  2013     1     5      858       900      -2    1519    1530     -11
##  6     4983  2013     1     6     1019       900      79    1558    1530      28
##  7     4983  2013     1     7     1042       900     102    1620    1530      50
##  8     4983  2013     1     8      901       900       1    1504    1530     -26
##  9     4983  2013     1     9      641       900    1301    1242    1530    1272
## 10     4983  2013     1    10      859       900      -1    1449    1530     -41
## # ... with 336,766 more rows, 9 more variables: carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>, and abbreviated variable names
## #   1: sched_dep_time, 2: dep_delay, 3: arr_time, 4: sched_arr_time,
## #   5: arr_delay
```

The ones that traveled the shortest:

```
nycflights13::flights%>%
  dplyr::select(distance,
                dplyr::everything())%>%
  dplyr::arrange(distance)
```

```
## # A tibble: 336,776 x 19
##    distance  year month   day dep_time sched_d~1 dep_d~2 arr_t~3 sched~4 arr_d~5
##       <dbl> <int> <int> <int>    <int>     <int>   <dbl>   <int>   <int>   <dbl>
##  1       17  2013     7    27       NA       106      NA      NA     245      NA
##  2       80  2013     1     3     2127      2129      -2    2222    2224      -2
##  3       80  2013     1     4     1240      1200      40    1333    1306      27
##  4       80  2013     1     4     1829      1615     134    1937    1721     136
##  5       80  2013     1     4     2128      2129      -1    2218    2224      -6
##  6       80  2013     1     5     1155      1200      -5    1241    1306     -25
##  7       80  2013     1     6     2125      2129      -4    2224    2224       0
##  8       80  2013     1     7     2124      2129      -5    2212    2224     -12
##  9       80  2013     1     8     2127      2130      -3    2304    2225      39
## 10       80  2013     1     9     2126      2129      -3    2217    2224      -7
```

```
## # ... with 336,766 more rows, 9 more variables: carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>, and abbreviated variable names
## #   1: sched_dep_time, 2: dep_delay, 3: arr_time, 4: sched_arr_time,
## #   5: arr_delay
```

Selecting Columns

Selecting by the specific name of each column:

```
nycflights13::flights%>%
  dplyr::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
## # ... with 336,766 more rows
```

Selecting an interval of columns:

```
nycflights13::flights%>%
  dplyr::select(
    month:arr_time
  )
```

```
## # A tibble: 336,776 x 6
##    month   day dep_time sched_dep_time dep_delay arr_time
##    <int> <int>    <int>          <int>     <dbl>    <int>
##  1     1     1      517            515         2      830
##  2     1     1      533            529         4      850
##  3     1     1      542            540         2      923
##  4     1     1      544            545        -1     1004
##  5     1     1      554            600        -6      812
##  6     1     1      554            558        -4      740
##  7     1     1      555            600        -5      913
##  8     1     1      557            600        -3      709
##  9     1     1      557            600        -3      838
## 10     1     1      558            600        -2      753
## # ... with 336,766 more rows
```

Negative selection:

```
nycflights13::flights%>%
  dplyr::select(
    -(month:arr_time))
```

```
## # A tibble: 336,776 x 13
##     year sched_arr~1 arr_d~2 carrier flight tailnum origin dest  air_t~3 dista~4
##    <int>       <int>   <dbl> <chr>    <int> <chr>   <chr>  <chr>   <dbl>   <dbl>
##  1  2013         819      11 UA        1545 N14228  EWR    IAH       227    1400
##  2  2013         830      20 UA        1714 N24211  LGA    IAH       227    1416
##  3  2013         850      33 AA        1141 N619AA  JFK    MIA       160    1089
##  4  2013        1022     -18 B6         725 N804JB  JFK    BQN       183    1576
##  5  2013         837     -25 DL         461 N668DN  LGA    ATL       116     762
##  6  2013         728      12 UA        1696 N39463  EWR    ORD       150     719
##  7  2013         854      19 B6         507 N516JB  EWR    FLL       158    1065
##  8  2013         723     -14 EV        5708 N829AS  LGA    IAD        53     229
##  9  2013         846      -8 B6          79 N593JB  JFK    MCO       140     944
## 10  2013         745       8 AA         301 N3ALAA  LGA    ORD       138     733
## # ... with 336,766 more rows, 3 more variables: hour <dbl>, minute <dbl>,
## #   time_hour <dttm>, and abbreviated variable names 1: sched_arr_time,
## #   2: arr_delay, 3: air_time, 4: distance
```

The helpers are from `tidyselect`:

- `tidyselect::starts_with("")`
- `tidyselect::ends_with("")`
- `tidyselect::contains("")`
- `tidyselect::matches("")`
- `tidyselect::num_range("")`

We can select a few columns and also the rest of them:

```
nycflights13::flights%>%
  dplyr::select(
    minute,
    distance,
    dplyr::everything())
```

```
## # A tibble: 336,776 x 19
##    minute distance  year month   day dep_time sched_de~1 dep_d~2 arr_t~3 sched~4
##     <dbl>    <dbl> <int> <int> <int>    <int>      <int>   <dbl>   <int>   <int>
##  1     15     1400  2013     1     1      517        515       2     830     819
##  2     29     1416  2013     1     1      533        529       4     850     830
##  3     40     1089  2013     1     1      542        540       2     923     850
##  4     45     1576  2013     1     1      544        545      -1    1004    1022
##  5      0      762  2013     1     1      554        600      -6     812     837
##  6     58      719  2013     1     1      554        558      -4     740     728
##  7      0     1065  2013     1     1      555        600      -5     913     854
##  8      0      229  2013     1     1      557        600      -3     709     723
##  9      0      944  2013     1     1      557        600      -3     838     846
## 10      0      733  2013     1     1      558        600      -2     753     745
## # ... with 336,766 more rows, 9 more variables: arr_delay <dbl>, carrier <chr>,
## #   flight <int>, tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>,
## #   hour <dbl>, time_hour <dttm>, and abbreviated variable names
## #   1: sched_dep_time, 2: dep_delay, 3: arr_time, 4: sched_arr_time
```

Exercises 5.4.1

1

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

2

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

```
nycflights13::flights%>%
  dplyr::select(
    year,
    minute,
    year) #crime ocorre e nada acontece nesse caso
```

```
## # A tibble: 336,776 x 2
##     year minute
##    <int>  <dbl>
##  1  2013     15
##  2  2013     29
##  3  2013     40
##  4  2013     45
##  5  2013      0
##  6  2013     58
##  7  2013      0
##  8  2013      0
##  9  2013      0
## 10  2013      0
## # ... with 336,766 more rows
```

```
nycflights13::flights%>%
  dplyr::select(
    year,
    -(year)) #empty vector
```

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

```
nycflights13::flights%>%
  dplyr::select(
    (year:day),
    year)
```

```
## # 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
```

```
## # ... with 336,766 more rows
```

If we include the same column more than one time, the selection takes into account just the first operation. Unless it is a contradictory selection, which is this case we probably get an empty vector.

3

What does the `any_of()` function do? Why might if be helpful in conjunction with this vector?

Lets read the documentation:

```
?any_of()
```

```
## starting httpd help server ... done
```

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

nycflights13::flights%>%
  dplyr::select(
    any_of(vars))
```

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

```
nycflights13::flights%>%
  dplyr::select(
    -any_of(vars))
```

```
## # A tibble: 336,776 x 14
##    dep_time sched_~1 arr_t~2 sched~3 carrier flight tailnum origin dest  air_t~4
##       <int>    <int>   <int>   <int> <chr>    <int> <chr>   <chr>  <chr>   <dbl>
##  1      517      515     830     819 UA        1545 N14228  EWR    IAH       227
##  2      533      529     850     830 UA        1714 N24211  LGA    IAH       227
##  3      542      540     923     850 AA        1141 N619AA  JFK    MIA       160
##  4      544      545    1004    1022 B6         725 N804JB  JFK    BQN       183
##  5      554      600     812     837 DL         461 N668DN  LGA    ATL       116
##  6      554      558     740     728 UA        1696 N39463  EWR    ORD       150
##  7      555      600     913     854 B6         507 N516JB  EWR    FLL       158
##  8      557      600     709     723 EV        5708 N829AS  LGA    IAD        53
##  9      557      600     838     846 B6          79 N593JB  JFK    MCO       140
## 10      558      600     753     745 AA         301 N3ALAA  LGA    ORD       138
## # ... with 336,766 more rows, 4 more variables: distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>, and abbreviated variable names
## #   1: sched_dep_time, 2: arr_time, 3: sched_arr_time, 4: air_time
```

The `any_of()` command takes a vector of variables and selects columns according to the variables inside the vector. The documentation states that it is useful for negative selections and also that it does not check for errors. It is not clear yet to me what are the advantages.

4

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

```
nycflights13::flights%>%
  dplyr::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
## # ... with 336,766 more rows
```

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

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

```
select(flights,
       tidyselect::contains("time", ignore.case=FALSE)) #this one should return the same results as the
```

```
## # 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
## # ... with 336,766 more rows
```

The result does surprise. At a first glance we would thought that this selection would return empty vectors, however the default setting of `contains()` ignore the differences between upper and lower case. If we want to make a more precise selection (and make a distinction of upper and lower case) we can add `ignore.case=FALSE`.

Mutate

```
nycflights13::flights%>%
  dplyr::select(
    year:day,
    tidyselect::ends_with("delay"),
    distance,
    air_time)%>%
  dplyr::mutate(
    gain=dep_delay - arr_delay,
    speed=(distance/air_time)*60)->flights_sml
```

If we want just to keep the new variables we can use `dplyr::transmute` instead.

We can use `dplyr::mutate` with any vectorised operation. If one vector is shorter than the other, the operation will be recycled (will repeat until the the end of the bigger vector).

- Integer division: `%/%`;
- Remainder: `%%`;
- Lead and lag: `lead()`, `lag()`;
- Cumulative mean: `cummean()`;

Exercises 5.5.2

1

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

```
nycflights13::flights%>%
  dplyr::select(
    dep_time,
    sched_dep_time)%>%
  dplyr::mutate(
    dep_time_min=(dep_time%/%100)*60+(dep_time)%%100,
    sched_dep_time_min=(sched_dep_time%/%100)*60+(sched_dep_time)%%100)
```

```
## # A tibble: 336,776 x 4
##    dep_time sched_dep_time dep_time_min sched_dep_time_min
##       <int>          <int>        <dbl>              <dbl>
##  1      517            515          317                315
##  2      533            529          333                329
##  3      542            540          342                340
##  4      544            545          344                345
##  5      554            600          354                360
##  6      554            558          354                358
##  7      555            600          355                360
##  8      557            600          357                360
##  9      557            600          357                360
## 10      558            600          358                360
## # ... with 336,766 more rows
```

2

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

```
nycflights13::flights%>%
  dplyr::select(
    air_time,
    arr_time,
    dep_time)%>%
  dplyr::mutate(
    diff=arr_time-dep_time)
```

```
## # A tibble: 336,776 x 4
##    air_time arr_time dep_time  diff
##       <dbl>    <int>    <int> <int>
##  1      227      830      517   313
##  2      227      850      533   317
##  3      160      923      542   381
##  4      183     1004      544   460
##  5      116      812      554   258
##  6      150      740      554   186
##  7      158      913      555   358
##  8       53      709      557   152
##  9      140      838      557   281
## 10      138      753      558   195
## # ... with 336,766 more rows
```

The variable air time is the amount of time spent in the air, in minutes. However the arrival and departure time are in the format of HoursMinutes. So, to fix it, we must compute the difference in minutes (or convert the arrival and departure time into minutes before subtracting).

```
nycflights13::flights%>%
  dplyr::select(
    air_time,
    arr_time,
    dep_time)%>%
  dplyr::mutate(
    arr_time_min = (arr_time%/%100)*60+(arr_time%%100),
    dep_time_min = (dep_time%/%100)*60+(dep_time%%100),
    air_time2 = arr_time_min-dep_time_min)
```

```
## # A tibble: 336,776 x 6
##    air_time arr_time dep_time arr_time_min dep_time_min air_time2
##       <dbl>    <int>    <int>        <dbl>        <dbl>     <dbl>
##  1      227      830      517          510          317       193
##  2      227      850      533          530          333       197
##  3      160      923      542          563          342       221
##  4      183     1004      544          604          344       260
##  5      116      812      554          492          354       138
##  6      150      740      554          460          354       106
##  7      158      913      555          553          355       198
##  8       53      709      557          429          357        72
##  9      140      838      557          518          357       161
## 10      138      753      558          473          358       115
## # ... with 336,766 more rows
```

3

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

```
nycflights13::flights%>%
  dplyr::select(
    dep_time,
    sched_dep_time,
    dep_delay)
```

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

They are related in the following form:

```
nycflights13::flights%>%
  dplyr::select(
    dep_time,
    sched_dep_time,
    dep_delay)%>%
  dplyr::mutate(
    dep_time_min = (dep_time%/%100)*60+(dep_time%%100),
    sched_dep_time_min = (sched_dep_time%/%100)*60+(sched_dep_time%%100),
    calc=dep_time_min-sched_dep_time_min)
```

```
## # A tibble: 336,776 x 6
##    dep_time sched_dep_time dep_delay dep_time_min sched_dep_time_min  calc
##       <int>          <int>     <dbl>        <dbl>              <dbl> <dbl>
##  1      517            515         2          317                315     2
##  2      533            529         4          333                329     4
##  3      542            540         2          342                340     2
##  4      544            545        -1          344                345    -1
##  5      554            600        -6          354                360    -6
##  6      554            558        -4          354                358    -4
##  7      555            600        -5          355                360    -5
##  8      557            600        -3          357                360    -3
##  9      557            600        -3          357                360    -3
## 10      558            600        -2          358                360    -2
## # ... with 336,766 more rows
```

4

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

```
nycflights13::flights%>%
  select(
    flight,
    dep_delay)->delays
```

I don't know what exactly is happening here with the `min_rank` function.

5

What does `1:3+1:10` return? Why?

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

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

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

Since the vectors are of different length the smallest vector is repeated in the operation.

6

What trigonometric functions does R provide?

The basic three functions are: * `cos(x)`
* `sin(x)` * `tan(x)`

These three following functions calculate the arc and the two-argument arc-tan:

- `acos(x)`
- `asin(x)`
- `atan(x)`
- `atan2(x,y)`

The last three functions calculate the `function(pi*x)`.

- `cospi(x)`
- `sinpi(x)`
- `tanpi(x)`

Grouped summaries with `summarise()`

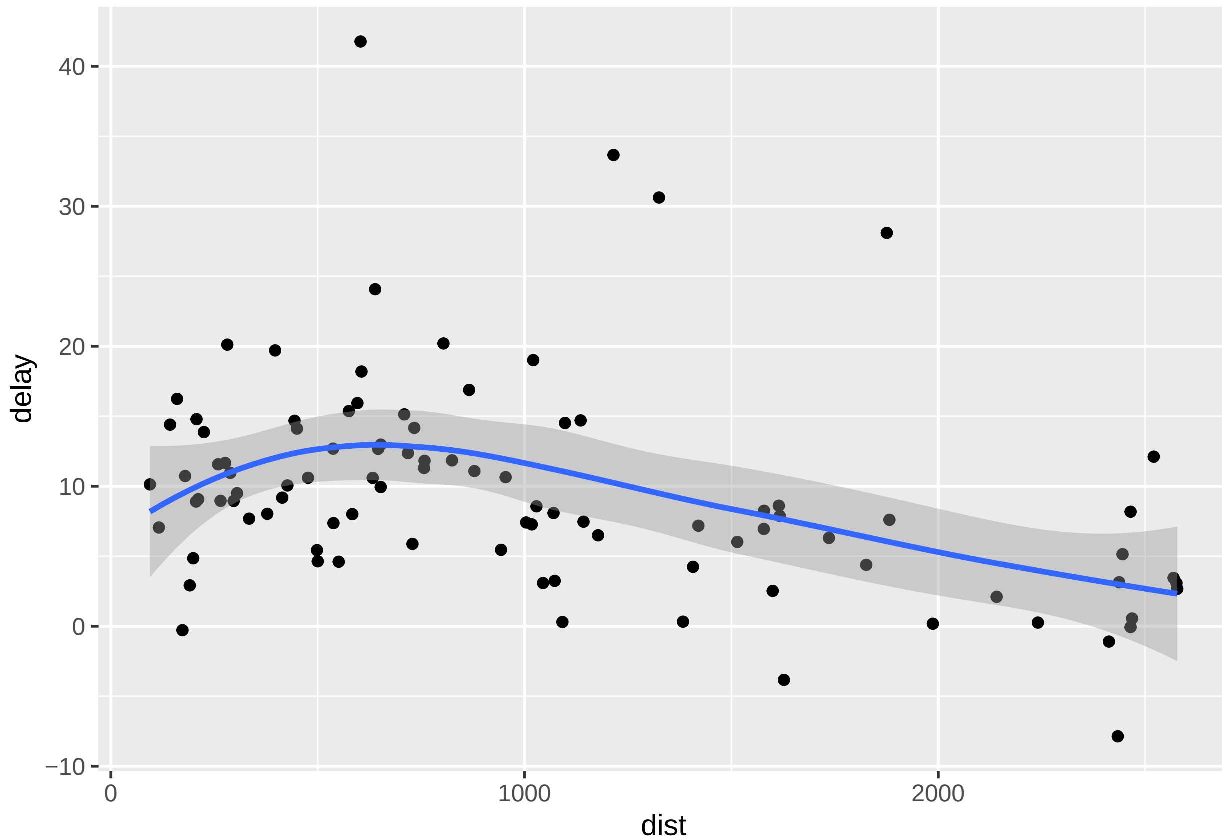
```
nycflights13::flights%>%
  dplyr::group_by(year, month, day)%>%
  dplyr::summarise(
    delay = mean(dep_delay, na.rm=TRUE),
    .groups = "drop")->gp
```

```
nycflights13::flights%>%
  dplyr::group_by(dest)%>%
  dplyr::summarise(
    count = n(),
    dist = mean(distance, na.rm=TRUE),
    delay = mean(arr_delay, na.rm=TRUE)
  )->dest  #just to visualize the dataframe

nycflights13::flights%>%
  dplyr::group_by(dest)%>%
```

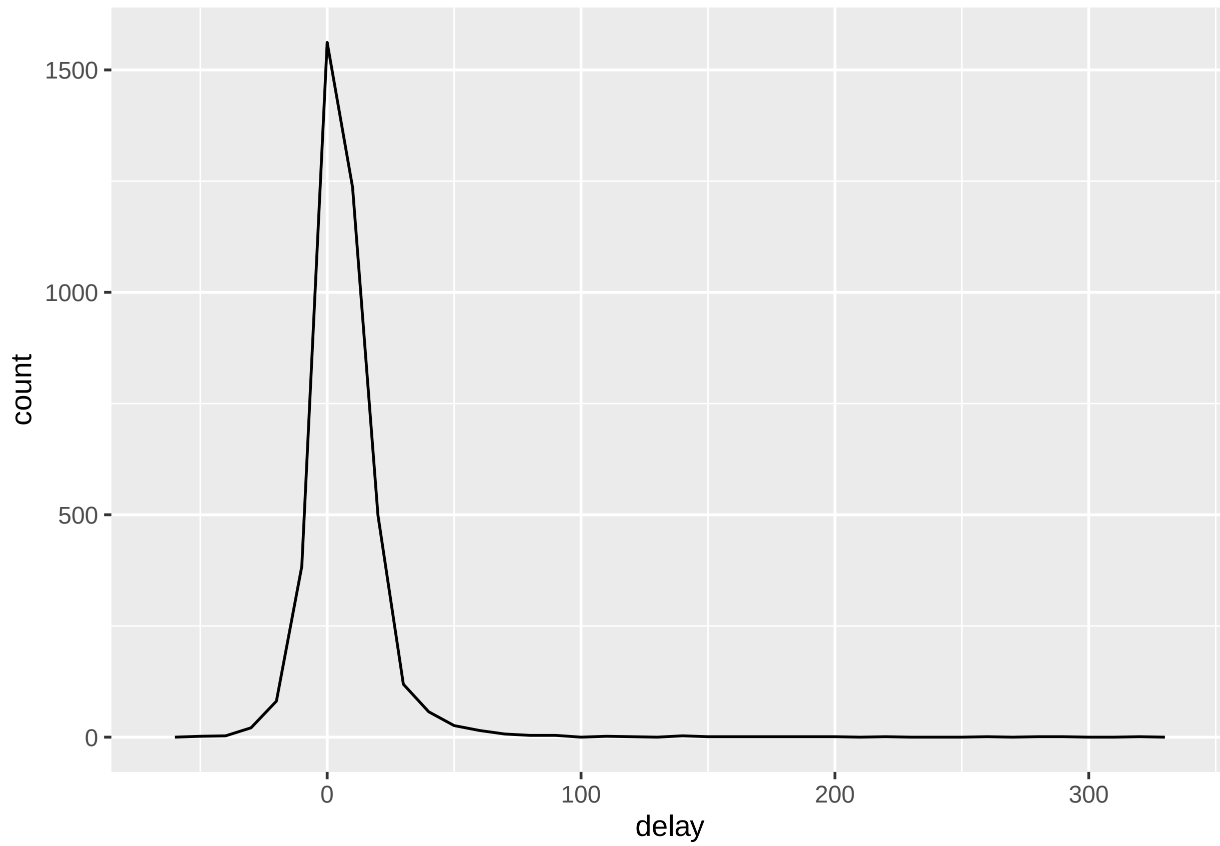
```
  dplyr::summarise(
    count = n(),
    dist = mean(distance, na.rm=TRUE),
    delay = mean(arr_delay, na.rm=TRUE))%>%
  dplyr::filter(count > 20, dest != "HNL")%>%
  ggplot2::ggplot(aes(x=dist, y=delay))+
  geom_point()+
  geom_smooth()
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```




```
nycflights13::flights%>%
  dplyr::group_by(tailnum)%>%
  summarise(
    delay = mean(arr_delay, na.rm=TRUE)
  )%>%
  ggplot2::ggplot(aes(x=delay))+
  geom_freqpoly(binwidth=10)
```

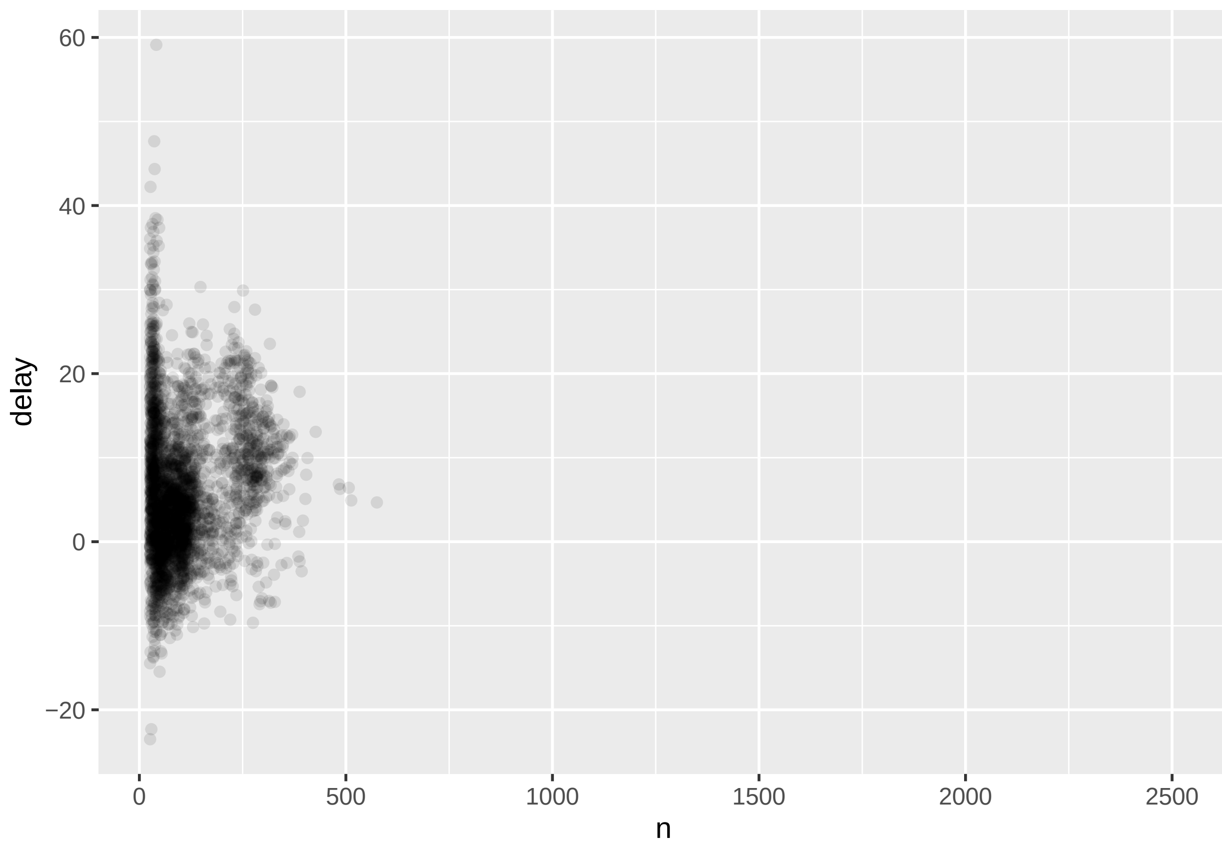
```
## Warning: Removed 7 rows containing non-finite values (stat_bin).
```



Number of flights per average delay:

```
nycflights13::flights%>%
  dplyr::group_by(tailnum)%>%
  summarise(
    delay = mean(arr_delay, na.rm=TRUE),
    n=n())%>%
  dplyr::filter(n > 25)%>%
  ggplot2::ggplot(aes(x=n, y=delay))+
  geom_point(alpha = 1/10)
```

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



Counts and proportions of logical values: `sum(x > 10)`, `mean(y == 0)`. When used with numeric functions, `TRUE` is converted to 1 and `FALSE` to 0. This makes `sum()` and `mean()` very useful: `sum(x)` gives the number of `TRUE`s in x, and `mean(x)` gives the proportion.

Exercises 5.6.7

1

Brainstorm at least 5 different ways to assess the typical delay characteristics of a group of flights:

- A flight is 15 minutes early 50% of the time, and 15 minutes late 50% of the time.
- A flight is always 10 minutes late.
- A lfight is 30 minutes early 50% of the time, and 30 minutes late 50% of the time.
- 99% of the time a flight is on time. 1% of the time it's 2 hours late.

Which is more important: arrival delay or departure delay?

2

Come up with another approach that will give you the same output as `not_cancelled%>%count(dest)` and `not_cancelled%>%count(tailnum, wt=distance)` (without using `count()`).

Lets check the output from the book:

```
nycflights13::flights%>%
  dplyr::filter(!is.na(dep_delay), !is.na(arr_delay))->not_cancelled
```

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

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

The other way to achieve the same results is grouping by and then summarizing:

```
not_cancelled%>%
  dplyr::group_by(dest)%>%
  dplyr::summarise(
    n=n()
  )->my_nc

not_cancelled%>%
  dplyr::group_by(tailnum)%>%
  summarise(
    totalmiles = sum(distance)
  )->my_wtd
```

3

Our definition of cancelled flights (`is.na(dep_delay)|is.na(arr_dealy)`) is slightly sub optimal. Why? Which is the most important column?

This is suboptimal because the command is assessing two vectors. Instead I propose using just the variable Air Time. We can check the following code:

```
nycflights13::flights%>%
  dplyr::select(
    dep_delay,
    arr_delay,
    air_time,
    dplyr::everything())%>%
  dplyr::filter(is.na(air_time))->arrg
```

Filtering for just the non-available numbers in the air time variable, we can see that is basically the same result as getting the n.a. values from departure delay **or** n.a. values from arrival delay (I believe: `is.na(air_time)==is.na(dep_delay)|is.na(arr_dealy)`)

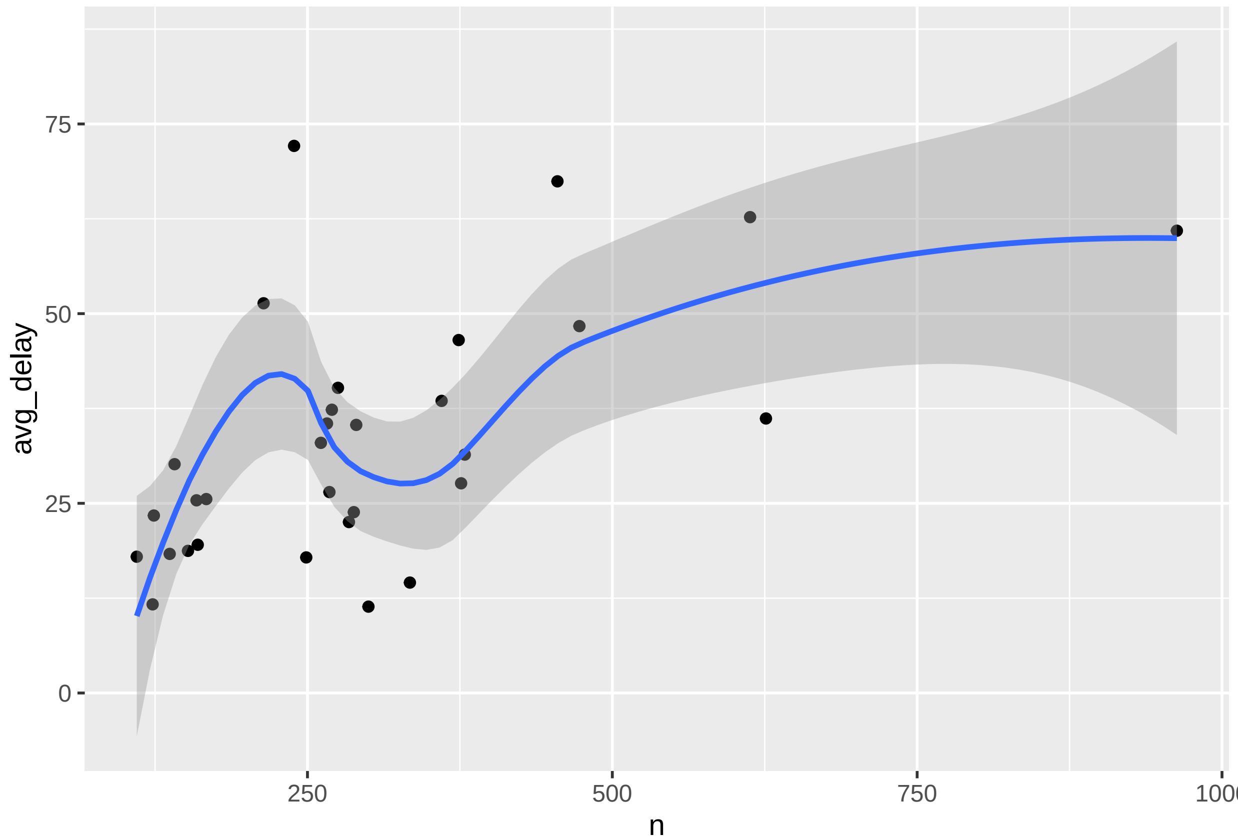
4

Look at the number of cancelled flights per day. Is there a pattern? Is the proportion of cancelled flights related to the average delay?

Even if is suboptimal, I will use the book's definition of cancelled flights:

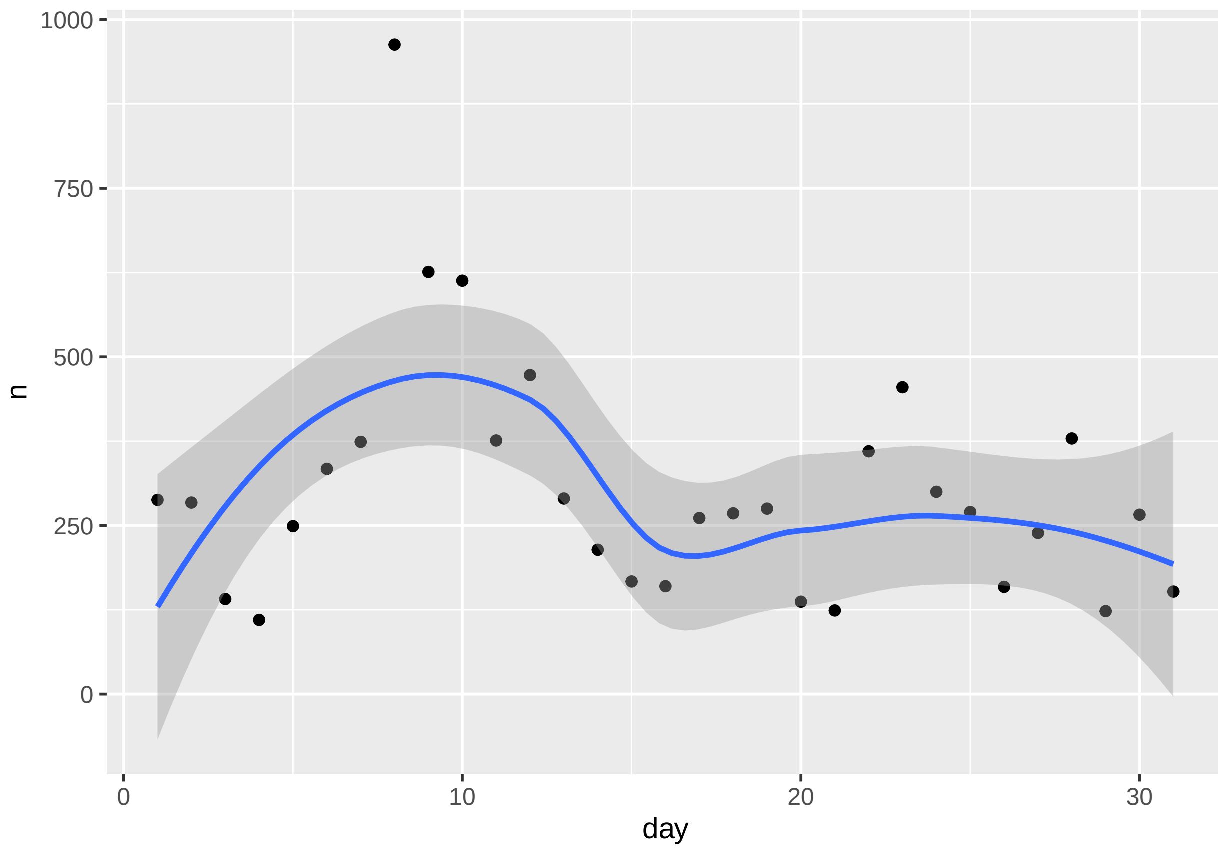
```
nycflights13::flights%>%
  dplyr::filter(is.na(dep_delay)|is.na(arr_delay))%>%
  dplyr::group_by(day)%>%
  dplyr::summarise(
    avg_delay=mean(dep_delay, na.rm=TRUE),
    n=n())%>%
  ggplot2::ggplot(aes(x=n, y=avg_delay))+
  geom_point()+
  geom_smooth()
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```




```
nycflights13::flights%>%
  dplyr::filter(is.na(dep_delay)|is.na(arr_delay))%>%
  dplyr::group_by(day)%>%
  dplyr::summarise(
    n=n())%>%
  ggplot2::ggplot(aes(x=day,y=n))+
  geom_point()+
  geom_smooth()
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



It does not seem to be a clear pattern.

5

Which carrier has the worst delays? Challenge: can you disentangle the effects of bad airport vs. bad carriers? Why/why not? (Hint: think about `flights%>%group_by(carrier,dest)%>%summarise(n())`)

```
nycflights13::flights%>%
  dplyr::group_by(carrier)%>%
  summarise(
    avg_dep_delay=mean(dep_delay, na.rm=TRUE))%>%
  dplyr::arrange(desc(avg_dep_delay))
```

```
## # A tibble: 16 x 2
##    carrier avg_dep_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
```

```
nycflights13::flights%>%
  dplyr::group_by(carrier)%>%
  summarise(
    avg_arr_delay=mean(arr_delay, na.rm=TRUE))%>%
  dplyr::arrange(desc(avg_arr_delay))
```

```
## # A tibble: 16 x 2
##    carrier avg_arr_delay
##    <chr>           <dbl>
##  1 F9             21.9
##  2 FL             20.1
##  3 EV             15.8
##  4 YV             15.6
##  5 OO             11.9
##  6 MQ             10.8
##  7 WN              9.65
##  8 B6              9.46
##  9 9E              7.38
## 10 UA              3.56
## 11 US              2.13
## 12 VX              1.76
## 13 DL              1.64
## 14 AA              0.364
## 15 HA             -6.92
## 16 AS             -9.93
```

F9 (frontier) is the worst carrier for both arrival and departure delays.

Now, taking the hint from the book:

```
nycflights13::flights%>%
  dplyr::group_by(carrier, dest)%>%
  summarise(n=n())->destcar
```

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

If we filter the flights made by frontier, we can see that they operate only between La Guardia (LGA) and Denver (DEN). Denver is one of the busiest airports in the US, it is possible that all flights related to Denver have big delays.

6

What does the `sort` argument do to `count()`. When might you use it?

When `sort=TRUE`, according to the documentation, the largest groups will appear at the top. It is not clear for me when this would be most useful.

```
People<-tibble::tibble(
  number=c(1,2,3,4,5,6),
  group=c(1,2,1,2,1,2))
```

Grouped Mutates (and filters)

Exercises 5.7.1

1

Refer back to the lists of useful mutate and filtering functions. Describe how each operation changes when you combine it with grouping.

2

Which plane (`tailnum`) has the worst on-time record?

3

What time of day should you fly if you want to avoid delays as much as possible?

4

For each destination, compute the total minutes of delay. For each flight, compute the proportion of the total delay for its destination.

5

Delays are typically temporally correlated: even once the problem that caused the initial delay has been resolved, later flights are delayed to allow earlier flights to leave. Using `lag()`, explore how the delay of a flight is related to the delay of the immediately preceding flight.

6

Look at each destination. Can you find flights that are suspiciously fast? (i.e. flights that represent a potential data entry error). Compute the air time of a flight relative to the shortest flight to that destination. Which flights were most delayed in the air?

7

Find all destinations that are flown by at least two carriers. Use that information to rank the carriers.

8

For each plane, count the number of flights before the first delay of greater than 1 hour.