kornel_kovacs_hw_4

Kornel Kovacs

2019 10 11

#1. mutate()

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

```
## -- Attaching packages ---------------------------------------- tidyverse 1.2.1 --

## v ggplot2 3.2.1     v purrr   0.3.2
## v tibble  2.1.3     v dplyr   0.8.3
## v tidyr   0.8.3     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0

## -- Conflicts ------------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

Narrow the tibble to see what mutate() is doing

```
(flights_small <- select(flights,
                         year:day,
                         ends_with("delay"),
                         distance,
                         air_time))
```

```
## # A tibble: 336,776 x 7
##     year month   day dep_delay arr_delay distance air_time
##    <int> <int> <int>     <dbl>     <dbl>    <dbl>    <dbl>
##  1  2013     1     1         2        11     1400      227
##  2  2013     1     1         4        20     1416      227
##  3  2013     1     1         2        33     1089      160
##  4  2013     1     1        -1       -18     1576      183
##  5  2013     1     1        -6       -25      762      116
##  6  2013     1     1        -4        12      719      150
##  7  2013     1     1        -5        19     1065      158
##  8  2013     1     1        -3       -14      229       53
##  9  2013     1     1        -3        -8      944      140
## 10  2013     1     1        -2         8      733      138
## # ... with 336,766 more rows
```

```
mutate(flights_small,
       catchup = dep_delay - arr_delay,
       speed_miles = (distance/air_time) * 60
       )
```

```
## # A tibble: 336,776 x 9
##     year month   day dep_delay arr_delay distance air_time catchup
##    <int> <int> <int>     <dbl>     <dbl>    <dbl>    <dbl>   <dbl>
##  1  2013     1     1         2        11     1400      227      -9
##  2  2013     1     1         4        20     1416      227     -16
##  3  2013     1     1         2        33     1089      160     -31
##  4  2013     1     1        -1       -18     1576      183      17
##  5  2013     1     1        -6       -25      762      116      19
##  6  2013     1     1        -4        12      719      150     -16
##  7  2013     1     1        -5        19     1065      158     -24
##  8  2013     1     1        -3       -14      229       53      11
##  9  2013     1     1        -3        -8      944      140       5
## 10  2013     1     1        -2         8      733      138     -10
## # ... with 336,766 more rows, and 1 more variable: speed_miles <dbl>
```

Magic numbers. Great, every one loves them. They are evil.

```
KM_PER_MILE <- 1.61
mutate(flights_small,
       speed_km = (distance * KM_PER_MILE/air_time) * 60)
```

```
## # A tibble: 336,776 x 8
##     year month   day dep_delay arr_delay distance air_time speed_km
##    <int> <int> <int>     <dbl>     <dbl>    <dbl>    <dbl>    <dbl>
##  1  2013     1     1         2        11     1400      227     596.
##  2  2013     1     1         4        20     1416      227     603.
##  3  2013     1     1         2        33     1089      160     657.
##  4  2013     1     1        -1       -18     1576      183     832.
##  5  2013     1     1        -6       -25      762      116     635.
##  6  2013     1     1        -4        12      719      150     463.
##  7  2013     1     1        -5        19     1065      158     651.
##  8  2013     1     1        -3       -14      229       53     417.
##  9  2013     1     1        -3        -8      944      140     651.
## 10  2013     1     1        -2         8      733      138     513.
## # ... with 336,766 more rows
```

Even nicer is to create intermediate results for clarity

```
mutate(flights_small,
       distance_km = distance * KM_PER_MILE,
       air_time_hours = air_time / 60,
       speed_km = distance_km / air_time_hours
       )
```

```
## # A tibble: 336,776 x 10
##     year month   day dep_delay arr_delay distance air_time distance_km
##    <int> <int> <int>     <dbl>     <dbl>    <dbl>    <dbl>       <dbl>
##  1  2013     1     1         2        11     1400      227       2254
##  2  2013     1     1         4        20     1416      227       2280.
```

```
##  3  2013     1     1        2       33     1089      160       1753.
##  4  2013     1     1       -1      -18     1576      183       2537.
##  5  2013     1     1       -6      -25      762      116       1227.
##  6  2013     1     1       -4       12      719      150       1158.
##  7  2013     1     1       -5       19     1065      158       1715.
##  8  2013     1     1       -3      -14      229       53        369.
##  9  2013     1     1       -3       -8      944      140       1520.
## 10  2013     1     1       -2        8      733      138       1180.
## # ... with 336,766 more rows, and 2 more variables: air_time_hours <dbl>,
## #   speed_km <dbl>
```

transmute only keeps new variables

```
transmute(flights_small,
       distance_km = distance * KM_PER_MILE,
       air_time_hours = air_time / 60,
       speed_km = distance_km / air_time_hours
       )
```

```
## # A tibble: 336,776 x 3
##    distance_km air_time_hours speed_km
##          <dbl>          <dbl>    <dbl>
##  1       2254           3.78      596.
##  2       2280.          3.78      603.
##  3       1753.          2.67      657.
##  4       2537.          3.05      832.
##  5       1227.          1.93      635.
##  6       1158.          2.5       463.
##  7       1715.          2.63      651.
##  8        369.          0.883     417.
##  9       1520.          2.33      651.
## 10       1180.          2.3       513.
## # ... with 336,766 more rows
```

You cannot use all transformations inside mutate. It has to be vectorized: it takes a vector and returns a vector of the same length The reason (I believe) is that the operation is done on the column as a whole, For this the operation needs to make sense for a whole column, not just for one number

SOME VECTORIZED OPERATIONS

```
transmute(flights,
          dep_time,
          dep_hour = dep_time %/% 100,
          dep_minutes = dep_time %% 100
          )
```

```
## # A tibble: 336,776 x 3
##    dep_time dep_hour dep_minutes
##       <int>    <dbl>       <dbl>
```

```
##  1      517        5          17
##  2      533        5          33
##  3      542        5          42
##  4      544        5          44
##  5      554        5          54
##  6      554        5          54
##  7      555        5          55
##  8      557        5          57
##  9      557        5          57
## 10      558        5          58
## # ... with 336,766 more rows
```

How can you test whether something is vectorized?

```
(x <- c(0,1,2,3,4,5,6,7,8,9))
```

```
##  [1] 0 1 2 3 4 5 6 7 8 9
```

```
(y <- 0:9)
```

```
##  [1] 0 1 2 3 4 5 6 7 8 9
```

```
(z <- seq(0,9))
```

```
##  [1] 0 1 2 3 4 5 6 7 8 9
```

```
(lag(y))
```

```
##  [1] NA  0  1  2  3  4  5  6  7  8
```

```
(lag(lag(y)))
```

```
##  [1] NA NA  0  1  2  3  4  5  6  7
```

```
(lead(y))
```

```
##  [1]  1  2  3  4  5  6  7  8  9 NA
```

Some cumulative and aggregate functions

```
cumsum(x)
```

```
##  [1]  0  1  3  6 10 15 21 28 36 45
```

```r
cumprod(x)
```

```
##  [1] 0 0 0 0 0 0 0 0 0 0
```

```r
cumprod(lead(x))
```

```
##  [1]      1      2      6     24    120    720   5040  40320 362880     NA
```

```r
?cummin
```

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

```r
?cummax
cummean(x)
```

```
##  [1] 0.0 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5
```

Logical operators work

```r
x > 3
```

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

```r
x > y
```

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

```r
x == y
```

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

Ranking functions

```r
y <- c(10, 5, 6, 3, 7)
min_rank(y)
```

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

So, what is not a vectorized operation?

```r
c(2,4)^2 # This is vectorized
```

```
## [1]  4 16
```

```
kk <- function(x) { x[3]}
kk(1:5) # not vectorized
```

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

```
mean(x)
```

```
## [1] 4.5
```

What happens when we try this on a dataframe

```
transmute(flights, delay = mean(arr_delay, na.rm = TRUE))
```

```
## # A tibble: 336,776 x 1
##    delay
##    <dbl>
##  1  6.90
##  2  6.90
##  3  6.90
##  4  6.90
##  5  6.90
##  6  6.90
##  7  6.90
##  8  6.90
##  9  6.90
## 10  6.90
## # ... with 336,766 more rows
```

```
transmute(flights, delay = kk(arr_delay))
```

```
## # A tibble: 336,776 x 1
##    delay
##    <dbl>
##  1    33
##  2    33
##  3    33
##  4    33
##  5    33
##  6    33
##  7    33
##  8    33
##  9    33
## 10    33
## # ... with 336,766 more rows
```

Exercise: Try out a few of the other commands in the chapter.(KK: Which chapter exactly? I tried some arbitrarily.)

```
transmute(flights, real_delay = sched_arr_time - arr_time)
```

```
## # A tibble: 336,776 x 1
##    real_delay
##         <int>
##  1        -11
##  2        -20
##  3        -73
##  4         18
##  5         25
##  6        -12
##  7        -59
##  8         14
##  9          8
## 10         -8
## # ... with 336,766 more rows
```

```
lead(c(1,2,3,4,5,6))
```

```
## [1]  2  3  4  5  6 NA
```

Exercise: Create several ranges with the n:m notation, i.e. 2:4, 4:8, etc.

```
c(1:13)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13
```

```
c(1:13,2)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13  2
```

```
c(13:13)
```

```
## [1] 13
```

```
c(5:8)
```

```
## [1] 5 6 7 8
```

```
c(pi:6)
```

```
## [1] 3.141593 4.141593 5.141593
```

```
c(0:pi)
```

```
## [1] 0 1 2 3
```

Try to find out whether you can also take negative ranges and descending

```
c(-3:13)
```

```
##  [1] -3 -2 -1  0  1  2  3  4  5  6  7  8  9 10 11 12 13
```

```
c(13:2)
```

```
##  [1] 13 12 11 10  9  8  7  6  5  4  3  2
```

```
c(-9: pi)
```

```
##  [1] -9 -8 -7 -6 -5 -4 -3 -2 -1  0  1  2  3
```

```
c(-pi: 7)
```

```
##  [1] -3.1415927 -2.1415927 -1.1415927 -0.1415927  0.8584073  1.8584073
##  [7]  2.8584073  3.8584073  4.8584073  5.8584073  6.8584073
```

Exercise: Read ¿':" (the same as help(":"))

```
help(":")
```

Exercise: Use slice() to choose the first 10 rows of flights.

```
slice(flights, 1:10)
```

```
## # A tibble: 10 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1  2013     1     1      517            515         2      830
##  2  2013     1     1      533            529         4      850
##  3  2013     1     1      542            540         2      923
##  4  2013     1     1      544            545        -1     1004
##  5  2013     1     1      554            600        -6      812
##  6  2013     1     1      554            558        -4      740
##  7  2013     1     1      555            600        -5      913
##  8  2013     1     1      557            600        -3      709
##  9  2013     1     1      557            600        -3      838
## 10  2013     1     1      558            600        -2      753
## # ... with 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>
```

Do the following exercises from 5.5.2:

Exercise 1

```
transmute(flights,
          dep_time,
          dep_hour = dep_time %/% 100,
          dep_minutes = dep_time %% 100
          )
```

```
## # A tibble: 336,776 x 3
##    dep_time dep_hour dep_minutes
##       <int>    <dbl>       <dbl>
##  1      517        5          17
##  2      533        5          33
##  3      542        5          42
##  4      544        5          44
##  5      554        5          54
##  6      554        5          54
##  7      555        5          55
##  8      557        5          57
##  9      557        5          57
## 10      558        5          58
## # ... with 336,766 more rows
```

Exercise 2

```
transmute(flights,
          air_time = arr_time - dep_time,
          arr_time,
          dep_time
          )
```

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

The formats of `arr_time` and `dep_time` are not suitable for computation in their current form. It would be wise to convert them to a date or time object in order to properly do computations with them.

Exercise 4

```
sort(flights$arr_delay, decreasing = TRUE)[1:10]
```

```
##  [1] 1272 1127 1109 1007  989  931  915  895  878  875
```

```
sort(min_rank(flights$arr_delay), decreasing = TRUE)[1:10]
```

```
##  [1] 327346 327345 327344 327343 327342 327341 327340 327339 327338 327337
```

summarise()

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

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

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

```
## [1] 12.63907
```

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

```
## Warning in mean.default(select(flights, dep_delay), na.rm = TRUE): argument
## is not numeric or logical: returning NA
```

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

Not the same!

```
flights$dep_delay
select(flights, dep_delay)
```

Still, summarise is way more interesting with its friend, group_by

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

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

Again, not the same structure.

```
by_destination <- group_by(flights, dest)
delay <- summarise(by_destination,
                   delay = mean(arr_delay, na.rm = TRUE))
```

OK, we need the distance too, or else there is not much to plot.

```
(delay <- summarise(by_destination,
                   delay = mean(arr_delay, na.rm = TRUE),
                   distance = mean(distance, na.rm = TRUE)))
```

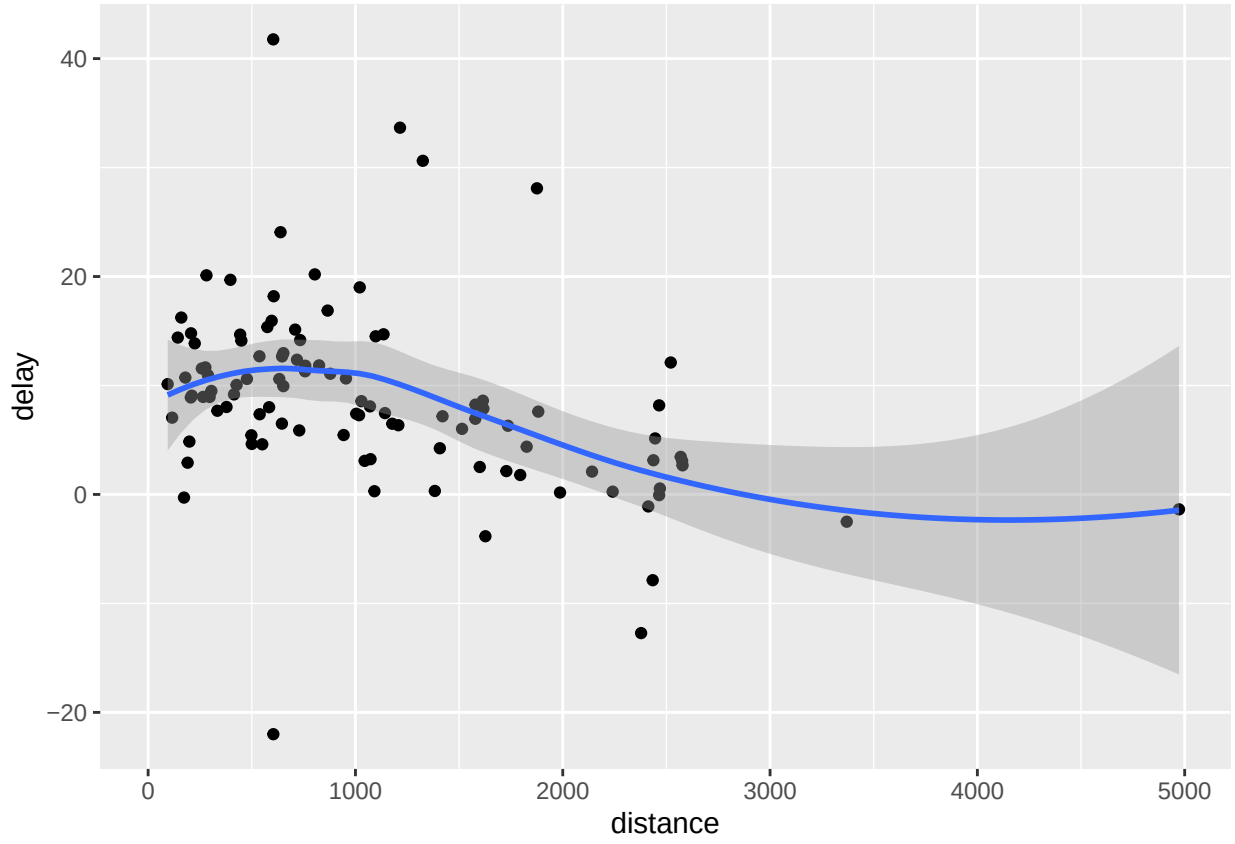
```
## # A tibble: 105 x 3
##    dest  delay distance
##    <chr> <dbl>    <dbl>
##  1 ABQ    4.38    1826
##  2 ACK    4.85     199
##  3 ALB   14.4      143
##  4 ANC   -2.5     3370
##  5 ATL   11.3      757.
##  6 AUS    6.02    1514.
##  7 AVL    8.00     584.
##  8 BDL    7.05     116
##  9 BGR    8.03     378
## 10 BHM   16.9      866.
## # ... with 95 more rows
```

```
p <- ggplot(data = delay,
            mapping = aes(x = distance, y = delay))
p + geom_point() + geom_smooth()
```

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

```
## Warning: Removed 1 rows containing non-finite values (stat_smooth).
```

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




```
(delay <- summarise(by_destination,
                    count = n(),
                   delay = mean(arr_delay, na.rm = TRUE),
                   distance = mean(distance, na.rm = TRUE)))
```

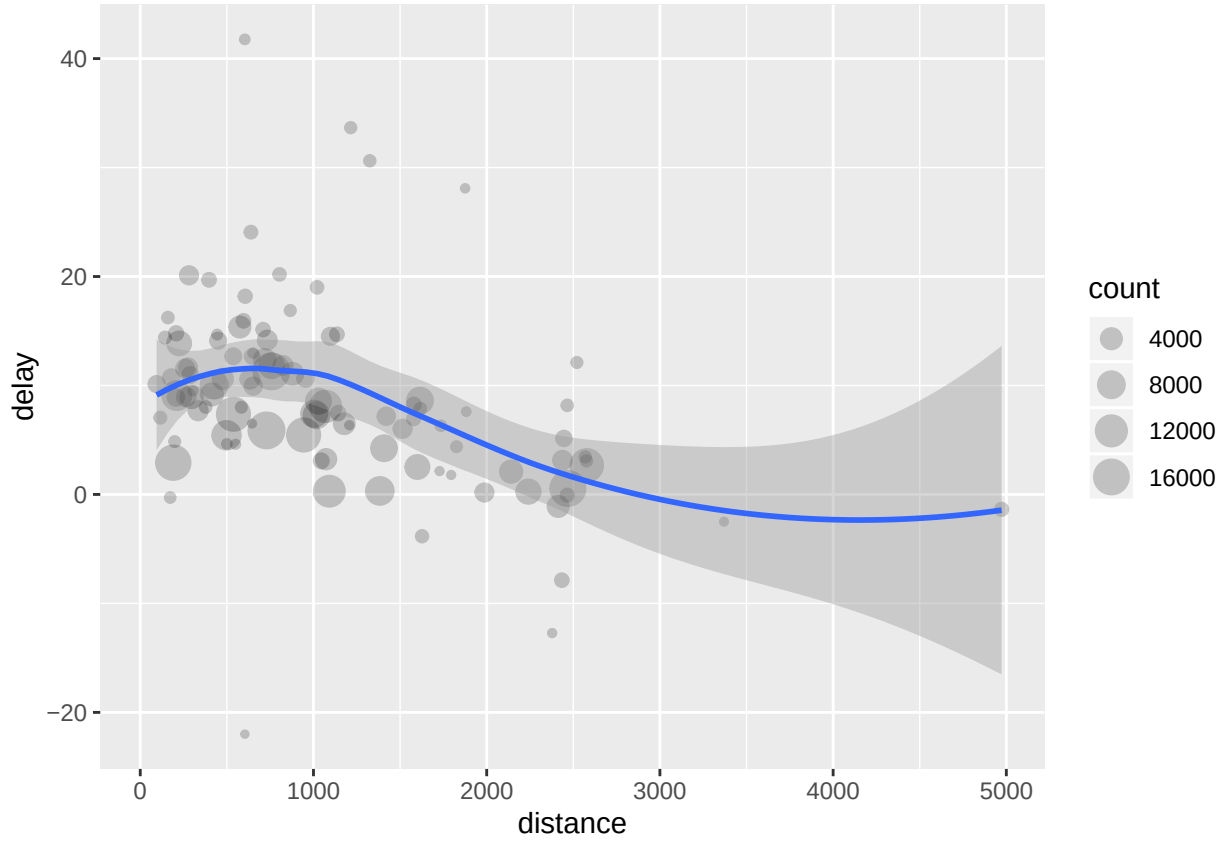
```
## # A tibble: 105 x 4
##    dest  count delay distance
##    <chr> <int> <dbl>    <dbl>
##  1 ABQ     254  4.38    1826
##  2 ACK     265  4.85     199
##  3 ALB     439 14.4      143
##  4 ANC       8 -2.5     3370
##  5 ATL   17215 11.3      757.
##  6 AUS    2439  6.02    1514.
##  7 AVL     275  8.00     584.
##  8 BDL     443  7.05     116
##  9 BGR     375  8.03     378
## 10 BHM     297 16.9      866.
## # ... with 95 more rows
```

```
p <- ggplot(data = delay,
            mapping = aes(x = distance, y = delay))
p + geom_point(mapping = aes(size = count), alpha = 0.2) +
  geom_smooth()
```

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

```
## Warning: Removed 1 rows containing non-finite values (stat_smooth).
```

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



Dropping some points

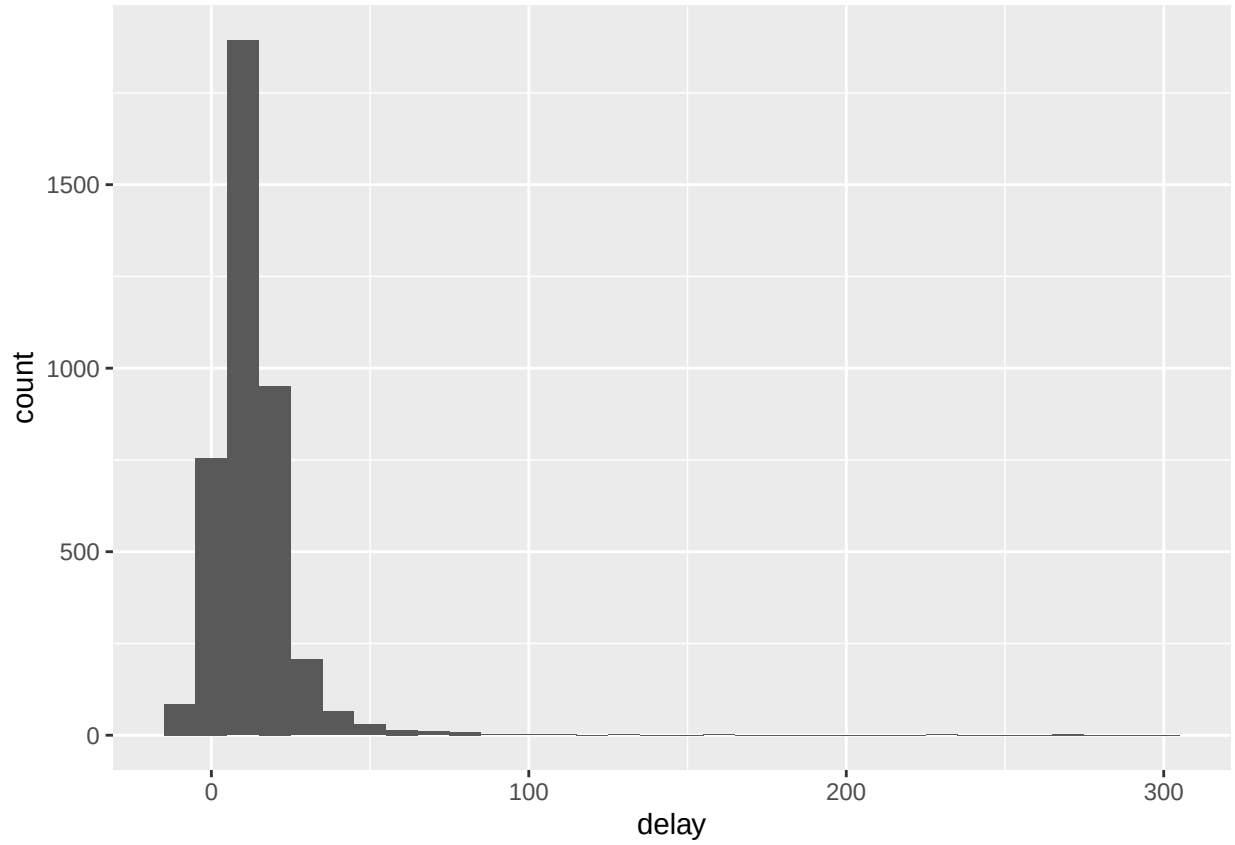
```
delays <- flights %>%
  group_by(dest) %>%
  summarise(
    delay = mean(arr_delay, na.rm = TRUE),
    count = n(),
    distance = mean(distance, na.rm = TRUE)
    ) %>%
  filter( count > 20, dest != "HNL")
```

Getting rid of missing values

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

Average delay by airplane (identified by tailnum), plot density

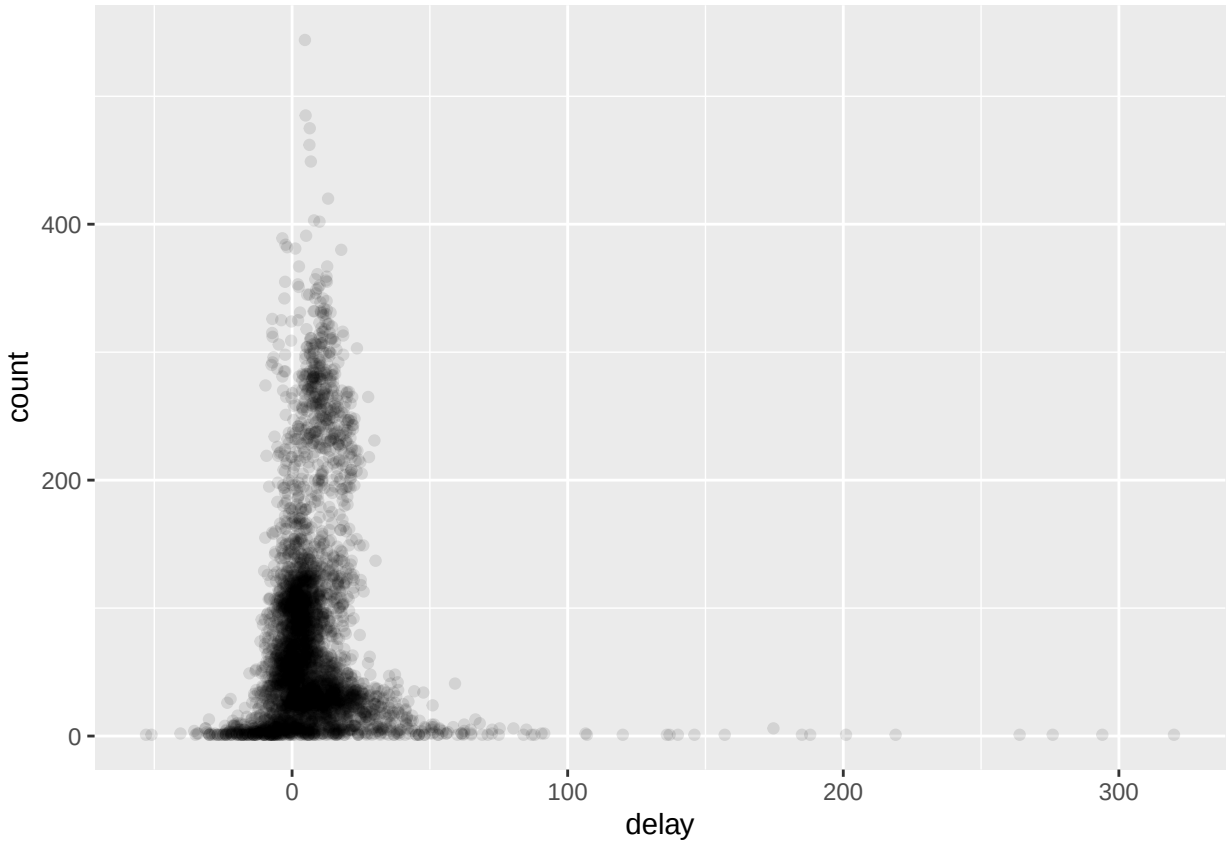
```
not_missing %>%
  group_by(tailnum) %>%
  summarise(delay = mean(dep_delay)) %>%
  ggplot(mapping = aes(x = delay)) +
  geom_histogram(binwidth = 10)
```



Plot number of flights per airplane against delay

```
not_missing %>%
  group_by(tailnum) %>%
  summarise(
    count = n(),
    delay = mean(arr_delay)
```

```
    ) %>%
  ggplot(mapping = aes(x = delay, y = count)) +
  geom_point(alpha = 0.1)
```



Since I need to filter the same thing, all the time just store in a variable.

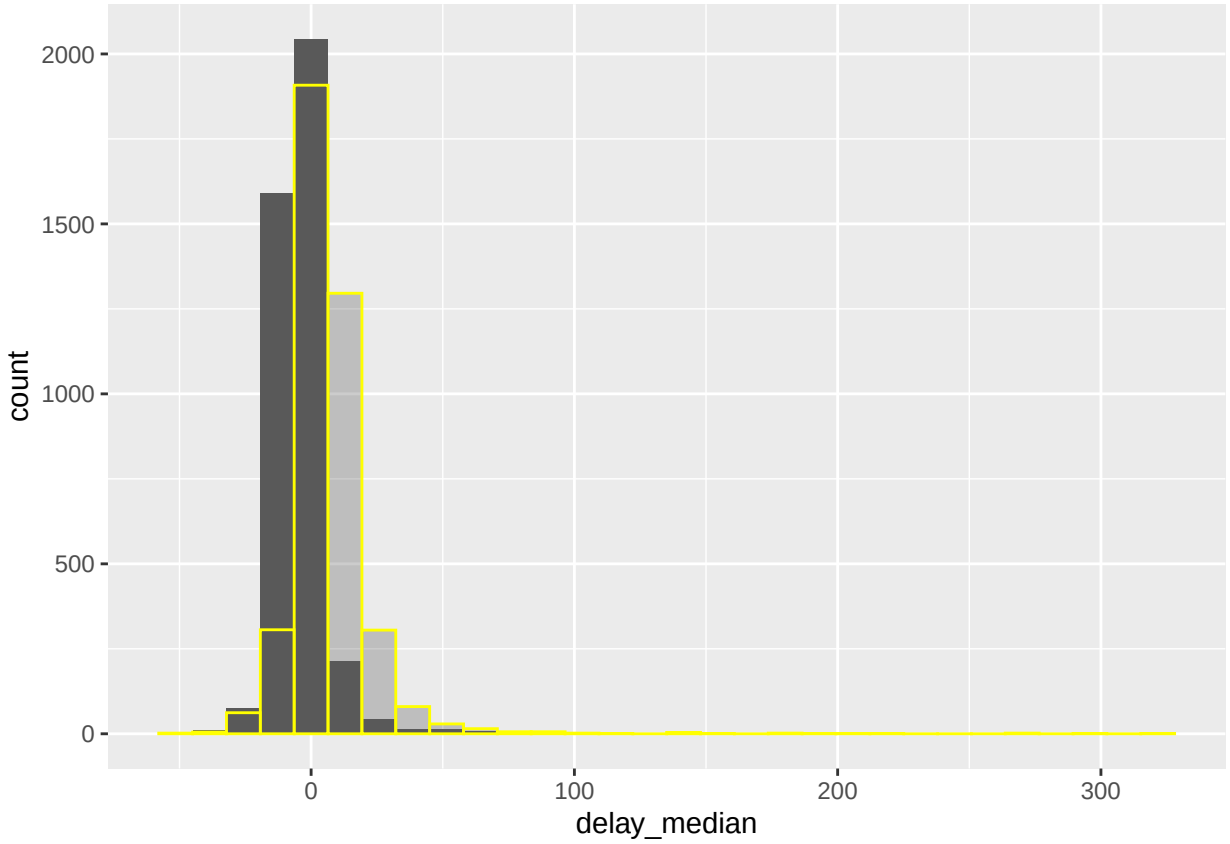
```
not_missing_planes <- not_missing %>%
  group_by(tailnum) %>%
  summarise(
    count = n(),
    delay = mean(arr_delay),
    delay_median = median(arr_delay)
    )
```

Get the median delay for each ariplane

```
ggplot(data = not_missing_planes) +
  geom_histogram(mapping = aes(x = delay_median)) +
  geom_histogram(mapping = aes(x = delay), color = 'yellow', alpha = 0.3)
```

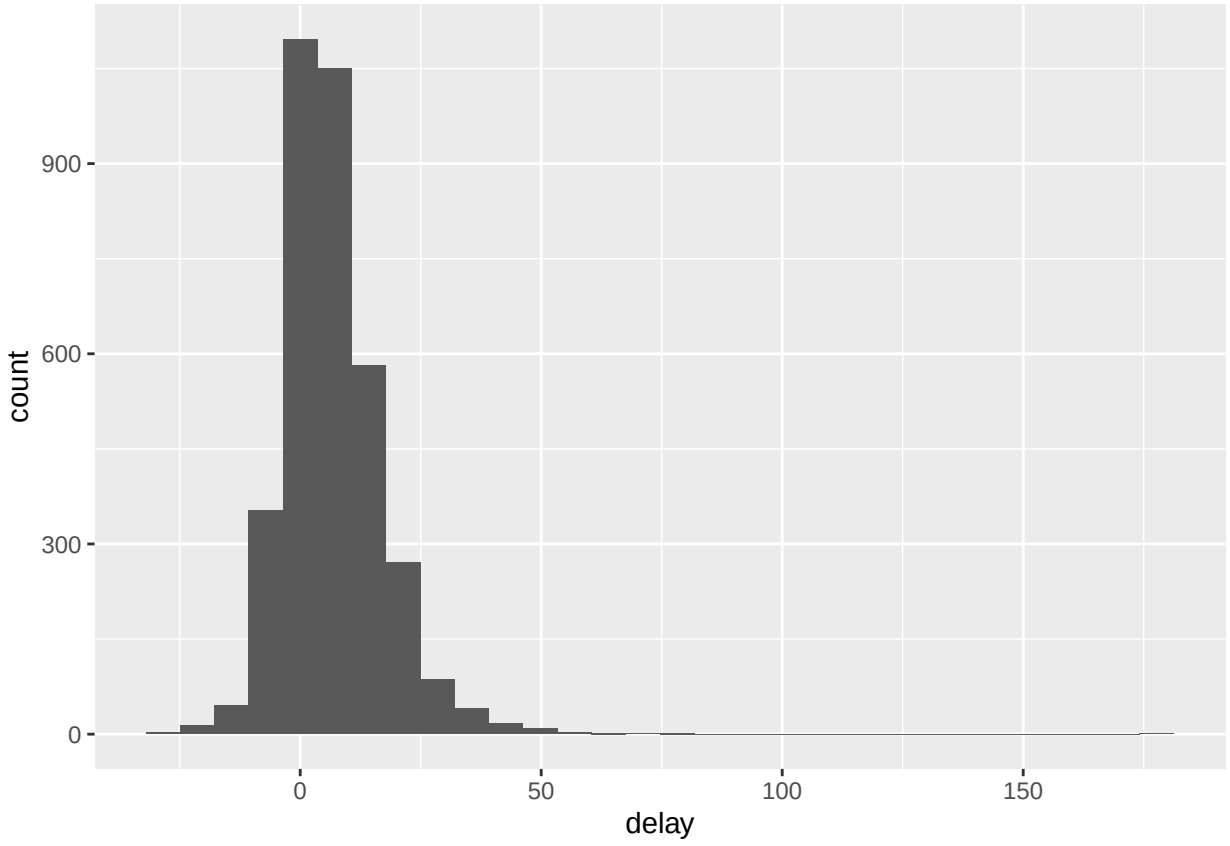
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
not_missing_planes %>%
  filter(count > 5) %>%
  ggplot(mapping = aes(x = delay)) +
  geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Assignment 5:

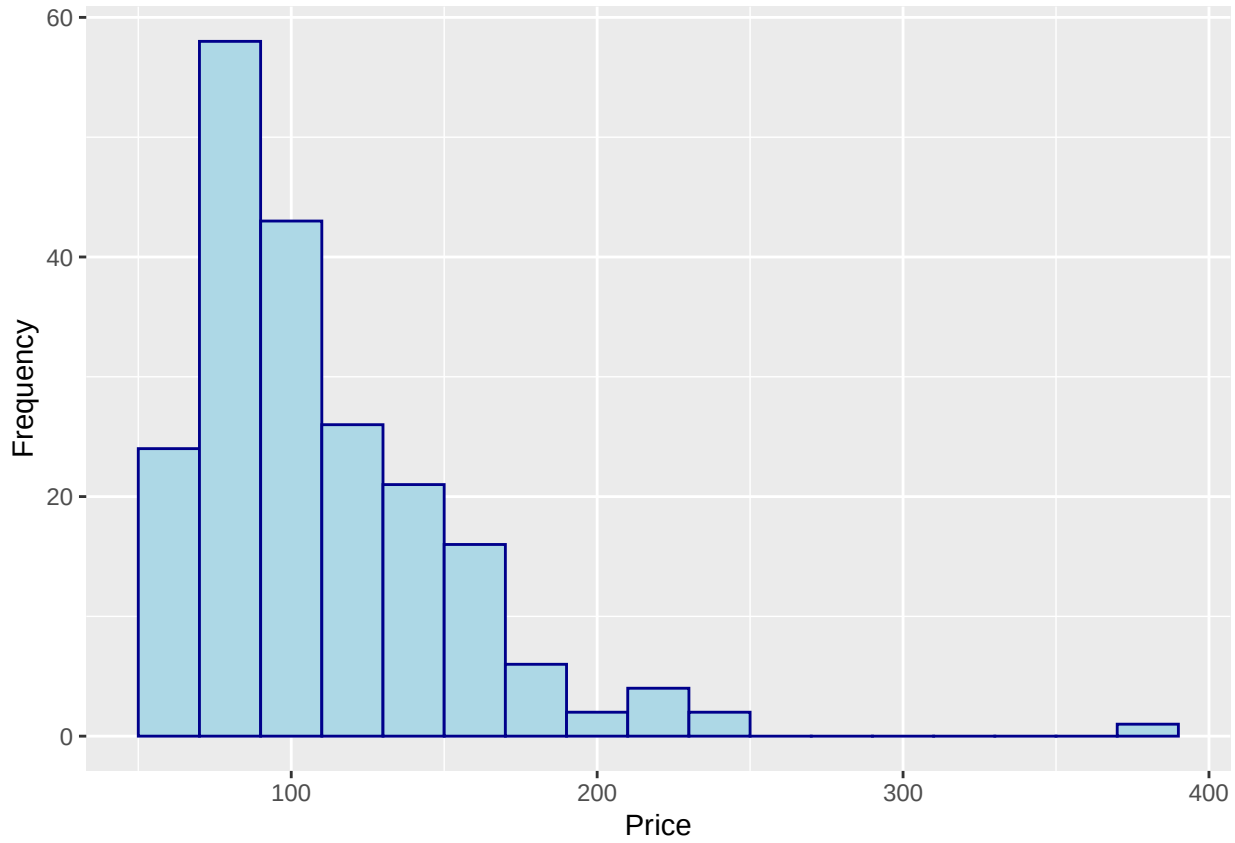
1. Do the exercises in this script file and work through the examples we didn't cover in class. As usual, turn the script into an .Rmd file, knit it, upload the .html and .pdf. - DONE
2. Read/skim the chapter 5 from 'R for Data Science' to see what is available. Don't try to remember everything, but you should be able to remember what is possible so that you can find the commands again should you need them in the future. - DONE
3. Grade Assignment 4 of your peers. - DONE
4. Document at least 10 errors and warnings you actually hit during the week. If you do *not* hit that many errors or receive such warnings, congratulations. - I did not really have any errors.
5. Pick one of the hotels graphs in Chapter 3, section 6, A1. Case study, finding a good deal among hotels. Replicate it – try it yourself for 10 minutes before you go looking at the code – and then make a variation of it.

```
hotels <- read.csv(file = "..//da_data_repo/hotels-vienna/clean/hotels-vienna.csv")
head(hotels)
```

```
##   country city_actual rating_count center1label center2label neighbourhood
## 1 Austria      Vienna           36  City centre    Donauturm   17. Hernals
## 2 Austria      Vienna          189  City centre    Donauturm   17. Hernals
## 3 Austria      Vienna           53  City centre    Donauturm    Alsergrund
```

```
## 4 Austria      Vienna           55  City centre    Donauturm    Alsergrund
## 5 Austria      Vienna           33  City centre    Donauturm    Alsergrund
## 6 Austria      Vienna           25  City centre    Donauturm    Alsergrund
##   price   city stars ratingta ratingta_count scarce_room hotel_id offer
## 1    81 Vienna     4      4.5           216           1    21894     1
## 2    81 Vienna     4      3.5           708           0    21897     1
## 3    85 Vienna     4      3.5           629           0    21901     1
## 4    83 Vienna     3      4.0            52           0    21902     1
## 5    82 Vienna     4      3.5           219           1    21903     1
## 6   229 Vienna     5      4.5            27           1    21904     1
##      offer_cat year month weekend holiday distance distance_alter
## 1 15-50% offer 2017    11       0       0      2.7            4.4
## 2  1-15% offer 2017    11       0       0      1.7            3.8
## 3 15-50% offer 2017    11       0       0      1.4            2.5
## 4 15-50% offer 2017    11       0       0      1.7            2.5
## 5 15-50% offer 2017    11       0       0      1.2            2.8
## 6  1-15% offer 2017    11       0       0      0.9            3.0
##   accommodation_type nnights rating
## 1          Apartment       1    4.4
## 2              Hotel       1    3.9
## 3              Hotel       1    3.7
## 4              Hotel       1    4.0
## 5              Hotel       1    3.9
## 6          Apartment       1    4.8
```

```
hotels_3_4_star <- filter(hotels, stars == 3 | stars == 4, city == 'Vienna', price < 1000, accommodatio
ggplot(hotels_3_4_star, aes(x = price)) +
  geom_histogram(binwidth = 20, color="darkblue", fill = "lightblue") +
  labs(x = "Price", y = "Frequency")
```



I could not really produce the very same graph, but it is close, I think. Furthemore, I experimented a lot.

6. Instead of using the Vienna data, use the data for another city (pick London if you don't want to choose). Do a basic data exploration, comparing the city to Vienna in terms of any variables you find interesting. Three plots maximum, don't spend more than 30 minutes on the analysis, before writing it down (if you are not doing this in parallel).

```
hotels_ams <- filter(read.csv(file = "..//da_data_repo//hotelbookingdata.csv"), city_actual == "Amsterda

hotels <- filter(read.csv(file = "..//da_data_repo//hotelbookingdata.csv"), city_actual == "Vienna")

mean(hotels_ams$price)
```

```
## [1] 339.6927
```

```
mean(hotels$price)
```

```
## [1] 213.8468
```

```
vienna_desc <-
  hotels %>%
  summarise(n = length(price),
            mean=mean(price),
            median=median(price),
```

```
            min = min(price),
            max = max(price),
            sd = sd(price),
            skew= ((mean(price)-median(price))/sd(price)))

ams_desc <-
  hotels_ams %>%
  summarise(n = length(price),
            mean=mean(price),
            median=median(price),
            min = min(price),
            max = max(price),
            sd = sd(price),
            skew= ((mean(price)-median(price))/sd(price)))
ams_desc
```

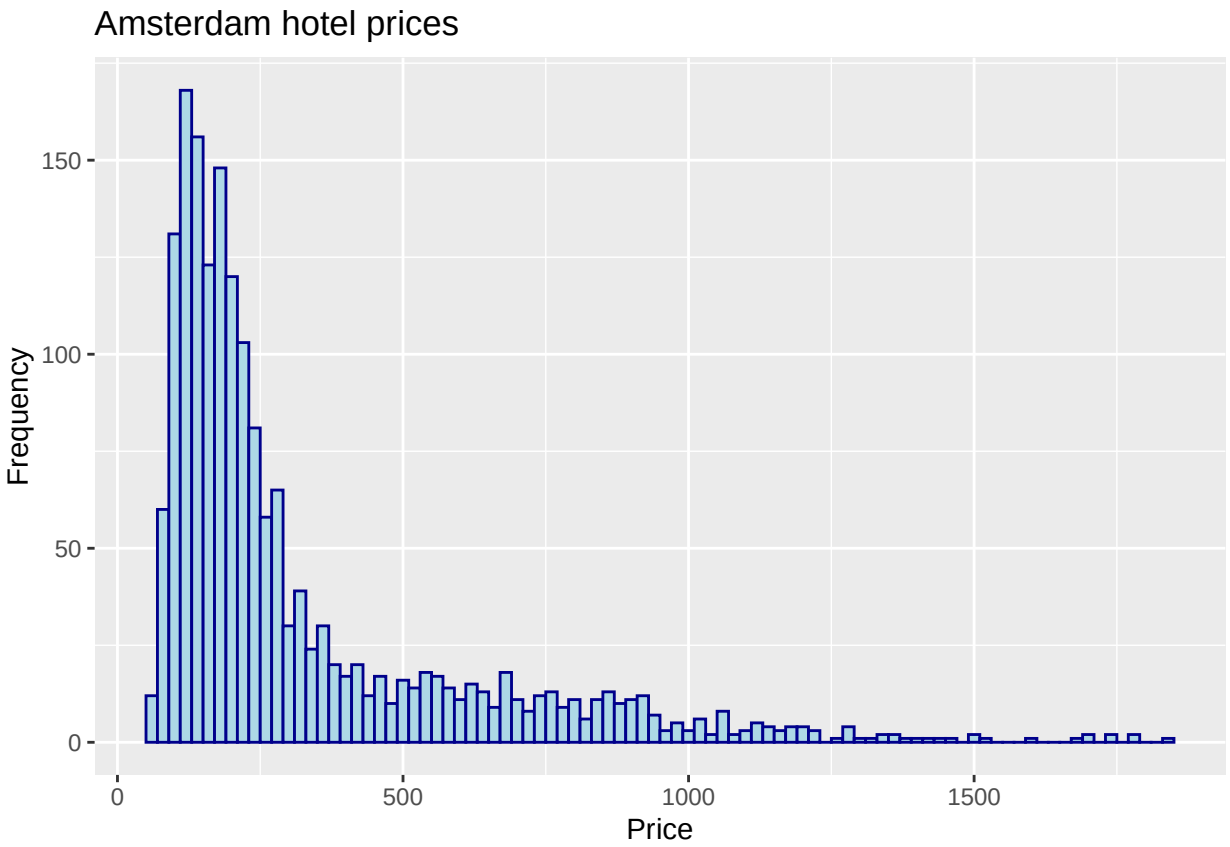
```
##      n     mean median min  max       sd      skew
## 1 1819 339.6927    207  50 3620 348.5161 0.3807362
```

```
vienna_desc
```

```
##      n     mean median min  max       sd      skew
## 1 3596 213.8468    138  27 6510 270.4845 0.2804108
```

```
ggplot(filter(hotels, price < 2000), aes(x = price)) +
  geom_histogram(binwidth = 20, color="darkblue", fill = "lightblue") +
  labs(x = "Price", y = "Frequency",  title = "Vienna hotel prices")
```


```
ggplot(filter(hotels_ams, price < 2000), aes(x = price)) +
  geom_histogram(binwidth = 20, color="darkblue", fill = "lightblue") +
  labs(x = "Price", y = "Frequency", title = "Amsterdam hotel prices")
```


Amsterdam hotel prices
Frequency
150
100
50
0
0
500
1000
1500
Price