Assignment for Lecture 4

R Markdown

mutate()

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

```
## -- Attaching packages -----------------------------------------------------------------------------------

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

```
library(ggpubr)
```

```
## Loading required package: magrittr

##
## Attaching package: 'magrittr'

## The following object is masked from 'package:purrr':
##
##     set_names

## The following object is masked from 'package:tidyr':
##
##     extract
```

```
# select(flights, dep_time, arr_time, air_time)
```

Let's stare at the columns to see what we can choose from View(flights)

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
```

```r
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>
```

No one knows what speed in miles is, let's fix that

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

```r
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
```

```r
# 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>
```

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

Standard arithmetic functions will work: +, *, etc

The time in dep_time is given by HHMM (How do I know this?)

```
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
```

log(), log2(), log10() work

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
```

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

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

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

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

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

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

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

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

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

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

What do lag and lead do?

Some cumulative and aggregate functions

```r
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
# ?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
```

```r
# What does the answer to this even mean?
x == c(2,4)
```

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

```
x > c(2,4,6)
```

```
## Warning in x > c(2, 4, 6): longer object length is not a multiple of
## shorter object length

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

Ranking functions

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

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

Can you figure out from playing around with min_rank() how it works exactly?

So, what is not a vectorized operation?

```
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.

```r
vars <- c(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

row_number(vars)
```

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

```r
dense_rank(vars)
```

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

```r
percent_rank(vars)
```

```
##  [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

```r
cume_dist(vars)
```

```
##  [1] 0.09090909 0.18181818 0.27272727 0.36363636 0.45454545 0.54545455
##  [7] 0.63636364 0.72727273 0.81818182 0.90909091 1.00000000
```

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

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

Ans: Yes we can take negative numbers and descending as well based on the examples below.

```r
vars_asc <- c(0:20)

vars_neg <- c(-5:-1)

vars_desc <- c(7:0)
```

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

Its the same as help.

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

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

```
#Convert into minutes from midnight
min_h <- 60
flights_updated <- flights %>%
  mutate(dep_time = (dep_time %/% 100)*min_h + dep_time %% 100,
sched_dep_time = (sched_dep_time %/% 100)*min_h + sched_dep_time %% 100)
```

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

Ans: Since the values of arr_time and dep_time are not continuous the subraction leads to the wrong result and a value different from the pre-computed air_time value (which is in minutes). Both arr-Time and dep_time need to be converted to minutes_from_midnight continuous values and then the ar_time - dep_time (journey_time in below solution) will be co,puted correctly.

```
flights %>% transmute(air_time,  journey_time = arr_time - dep_time)
```

```
## # A tibble: 336,776 x 2
##    air_time journey_time
##       <dbl>        <int>
##  1      227          313
##  2      227          317
##  3      160          381
##  4      183          460
##  5      116          258
##  6      150          186
##  7      158          358
##  8       53          152
##  9      140          281
## 10      138          195
## # ... with 336,766 more rows
```

```
#joruney_time is arr_time - dep_time
flights_updated <- flights %>%
  transmute(dep_time, arr_time, dep_time = (dep_time %/% 100)*min_h + dep_time %% 100,
arr_time = (arr_time %/% 100)*min_h + arr_time %% 100, journey_time = arr_time - dep_time, air_time)

flights_updated %>% select(dep_time, arr_time, journey_time, air_time)
```

```
## # A tibble: 336,776 x 4
```

```
##    dep_time arr_time journey_time air_time
##       <dbl>    <dbl>        <dbl>    <dbl>
##  1      317      510          193      227
##  2      333      530          197      227
##  3      342      563          221      160
##  4      344      604          260      183
##  5      354      492          138      116
##  6      354      460          106      150
##  7      355      553          198      158
##  8      357      429           72       53
##  9      357      518          161      140
## 10      358      473          115      138
## # ... with 336,766 more rows
```

Exercise 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().

Ans: The 10 most delayed flights can be found using min_rank. Ties are handled by giving the same rank to tied values and min_rank does it the same way.

```
del_flights <- flights %>% filter(min_rank(desc(dep_delay)) <= 10) %>% arrange(desc(dep_delay))
del_flights
```

```
## # 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     9      641            900      1301     1242
##  2  2013     6    15     1432           1935      1137     1607
##  3  2013     1    10     1121           1635      1126     1239
##  4  2013     9    20     1139           1845      1014     1457
##  5  2013     7    22      845           1600      1005     1044
##  6  2013     4    10     1100           1900       960     1342
##  7  2013     3    17     2321            810       911      135
##  8  2013     6    27      959           1900       899     1236
##  9  2013     7    22     2257            759       898      121
## 10  2013    12     5      756           1700       896     1058
## # ... 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>
```

Hint: When you get stuck, try the following two strategies: 1. Take a single row, and work it out by hand 2. Create a variable my_flights which contains only a few rows (4 to 10). Work out a solution for my_flights, where you can check every step.

summarise()

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

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

How… useful. Might as well do

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

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

$ will give you that column. Quick way to choose columns.

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

ERROR: argument is not numeric or logical: returning NA[1] NA

An error I made: I tried this: Huh? What's going on here?

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

I thought select(flights, dep_delay) was the same as flights$dep_delay Aha, we should have guessed, since select returns a *data frame*, but we want a column. A data frame of 1 column is not the same as a single column.

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

```r
by_day <- group_by(flights, year, month, day)
by_day
```

```
## # A tibble: 336,776 x 19
## # Groups:   year, month, day [365]
##     year month   day dep_time sched_dep_time dep_delay arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1  2013     1     1      517            515         2      830
##  2  2013     1     1      533            529         4      850
##  3  2013     1     1      542            540         2      923
##  4  2013     1     1      544            545        -1     1004
##  5  2013     1     1      554            600        -6      812
##  6  2013     1     1      554            558        -4      740
##  7  2013     1     1      555            600        -5      913
##  8  2013     1     1      557            600        -3      709
##  9  2013     1     1      557            600        -3      838
## 10  2013     1     1      558            600        -2      753
## # ... with 336,766 more rows, and 12 more variables: sched_arr_time <int>,
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>
```

Looks distinctly the same

But it really isn't!

```r
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
```

5.6.1 Let's explore link between distance and average delay for every location What that means is that we want to know the average delay for every destination. Then, once we have that, we want to see how the distance to this location is related to the delay to this location.

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

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

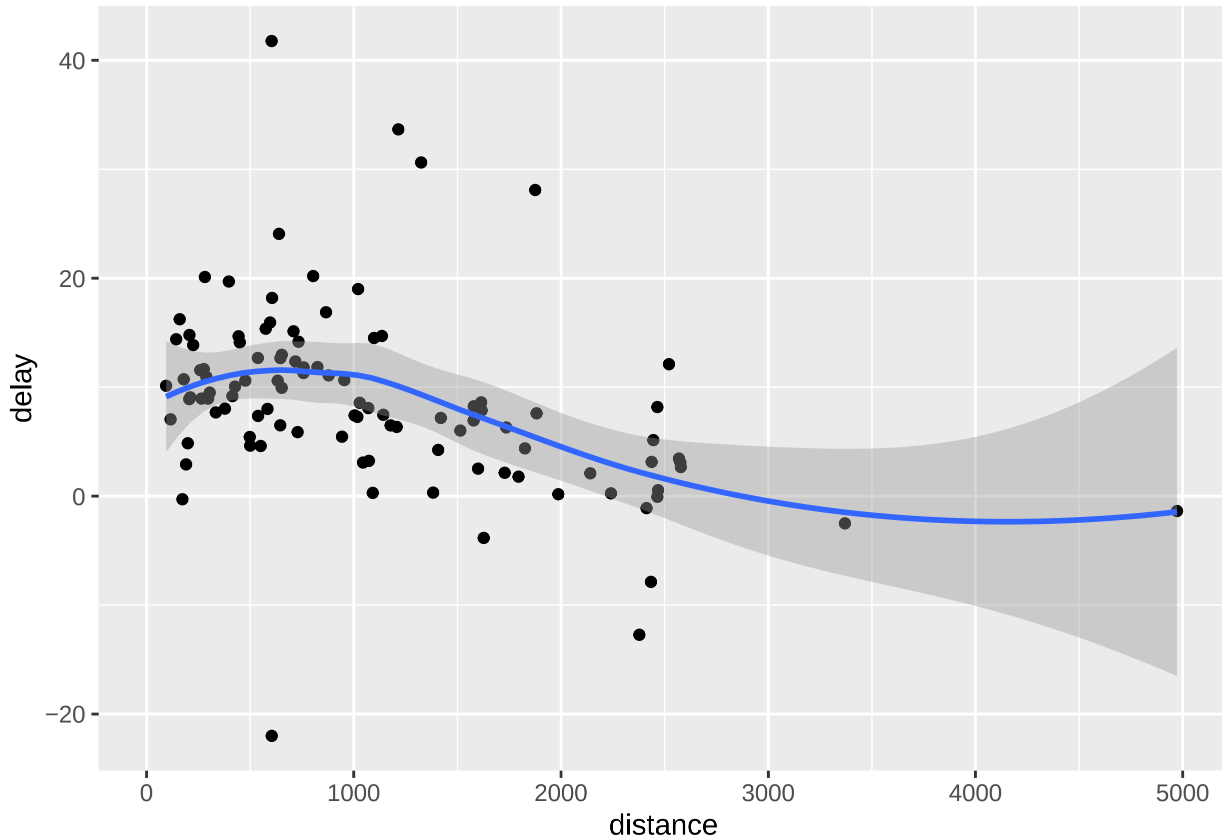
OK, we need the distance too, or else there is not much to plot.

```
(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).
```



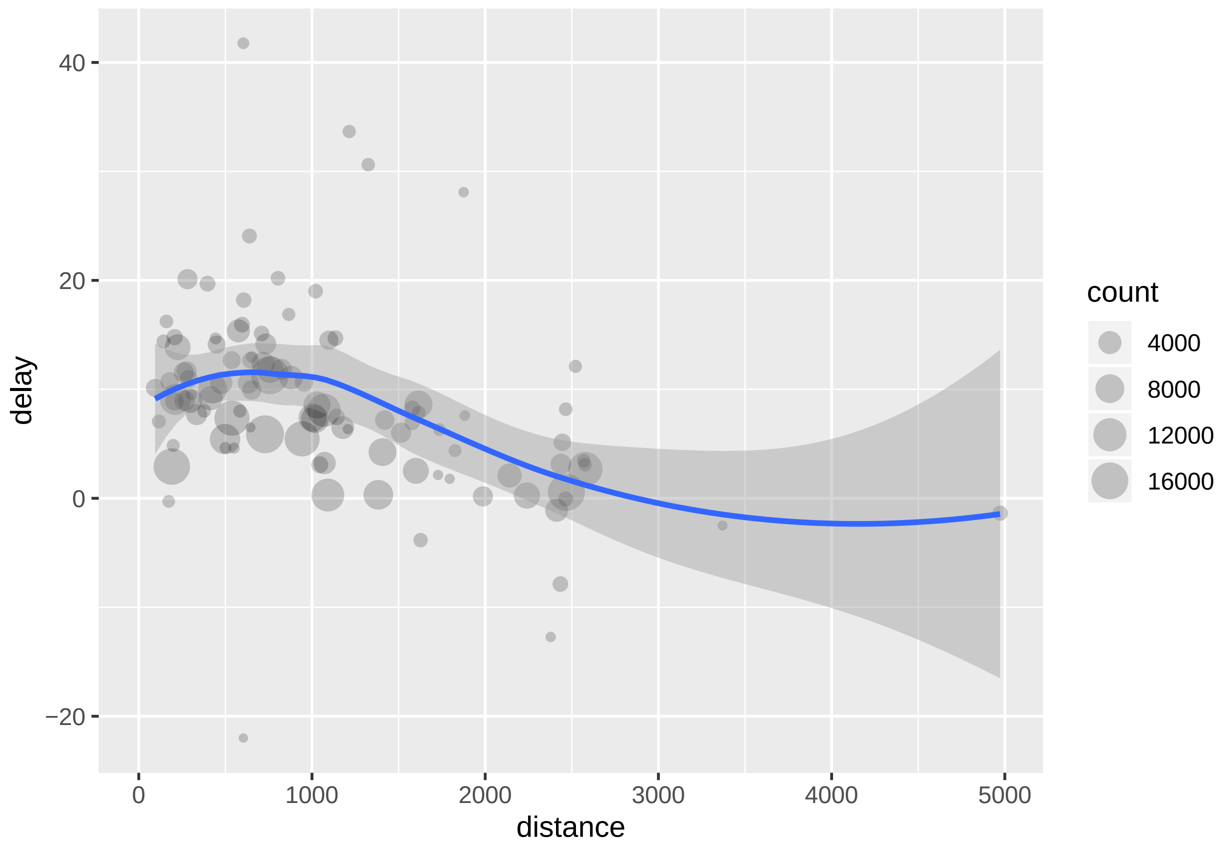
Improving the graph…

```
(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).
```



n() is a very special function #n()

Finally…

Optional exercise as part of assignment 5 (somewhat harder): The above does not take into account

the number of flights per location. A location with 1 flight matters as much

for smoothing as a location with 300.

That is rarely what we want when smoothing globally. Read the following code,

to see if you understand how it works. Explain in your words in the .Rmd file.

Let's plot the original data, without first taking means by group

Woah, that looks different! (And ugly.)

So, not too misleading, but still…

 # END OF EXERCISE

Doing this with a pipe, and filtering out destinations with - less than 20 flights - to HNL (Honululu), since it's by far the furthest Note: I am not a big fan of dropping things that 'look too different'. You should do such robustness checks, but you shouldn't start there.

```
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")
```

Exercise: Rewrite the above command without the pipe. Which one do you find easier to read?

Ans: Piping makes it much easier to write and much easier to read as well for me because I look at different steps without getting confused by extra information.

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

```
## # A tibble: 96 x 4
##    dest  delay count distance
##    <chr> <dbl> <int>    <dbl>
##  1 ABQ    4.38   254     1826
```

```
##   2 ACK    4.85   265      199
##   3 ALB   14.4    439      143
##   4 ATL   11.3  17215      757.
##   5 AUS    6.02  2439     1514.
##   6 AVL    8.00   275      584.
##   7 BDL    7.05   443      116
##   8 BGR    8.03   375      378
##   9 BHM   16.9    297      866.
##  10 BNA   11.8   6333      758.
## # ... with 86 more rows
```

5.6.2 Missing values

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

Exercise: Does the above command also drop observations that miss only the arr_delay but have a dep_delay?

Ans: YES

Are there any observations in the dataset for which only dep_delay or arr_delay is missing, but not both?

Ans: No, because both of the commands below return 0 result set.

```
flights %>%
  filter(!is.na(dep_delay) & is.na(dep_delay)) %>%  select(arr_delay, dep_delay)
```

```
## # A tibble: 0 x 2
## # ... with 2 variables: arr_delay <dbl>, dep_delay <dbl>
```

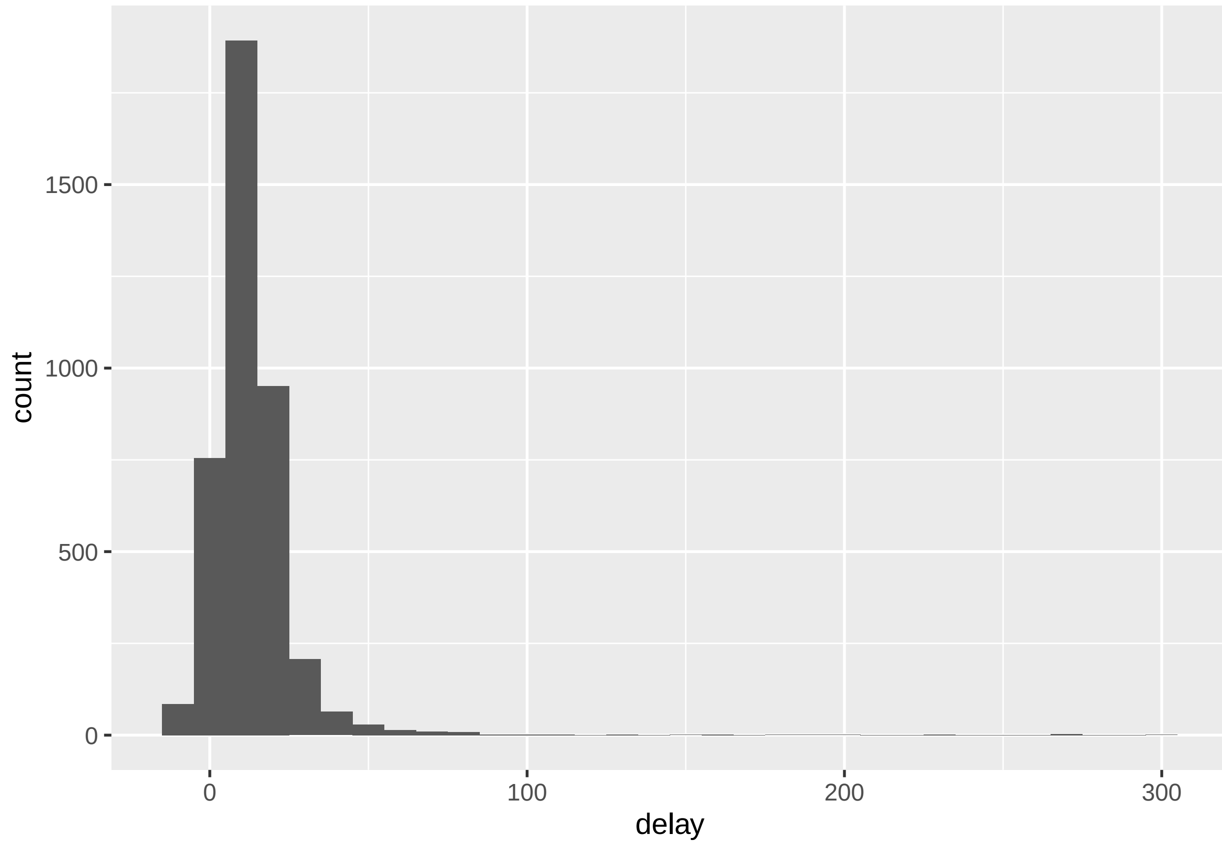
```
flights %>%
  filter(is.na(dep_delay) & !is.na(dep_delay)) %>%  select(arr_delay, dep_delay)
```

```
## # A tibble: 0 x 2
## # ... with 2 variables: arr_delay <dbl>, dep_delay <dbl>
```

5.6.3 Counts

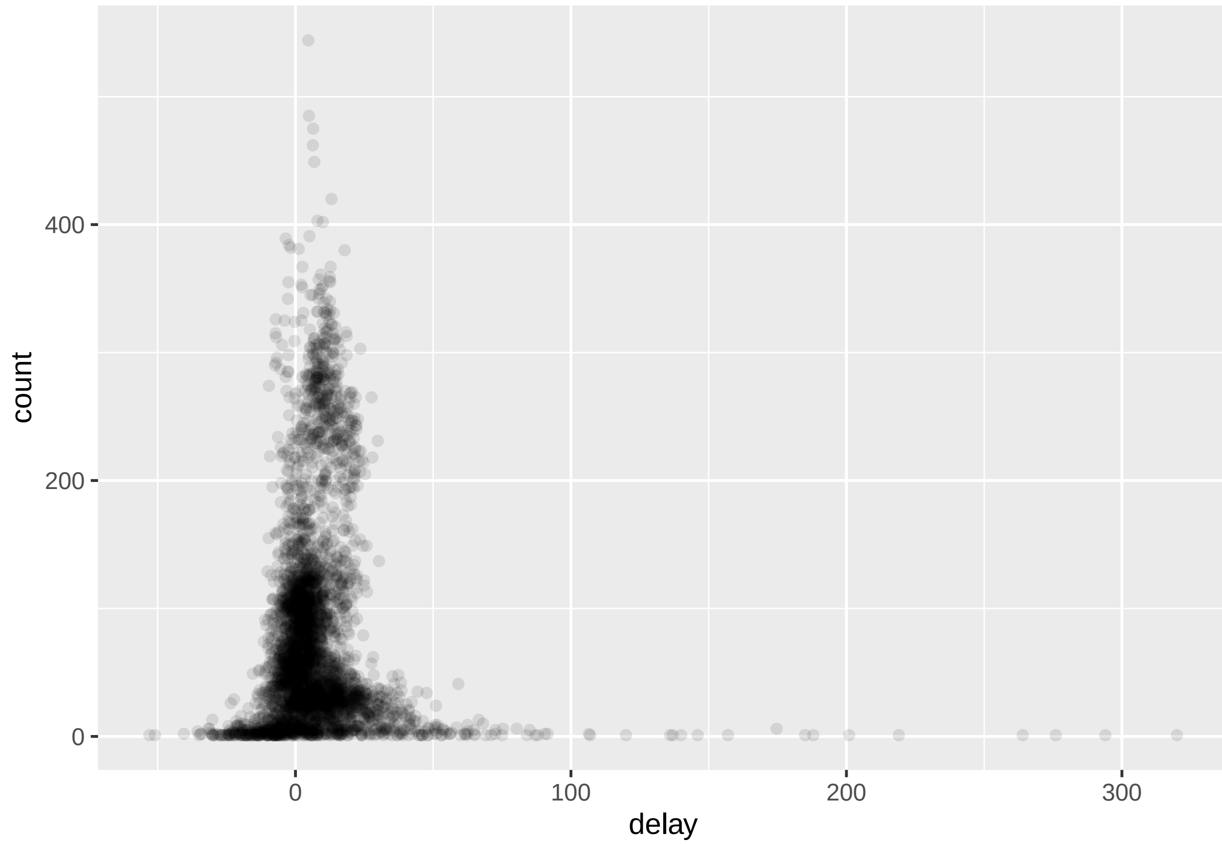
Average delay by airplane (identified by tailnum), plot density Start with freqpoly, then zoom in on that part of the graph that we are interested in..

```
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)
```



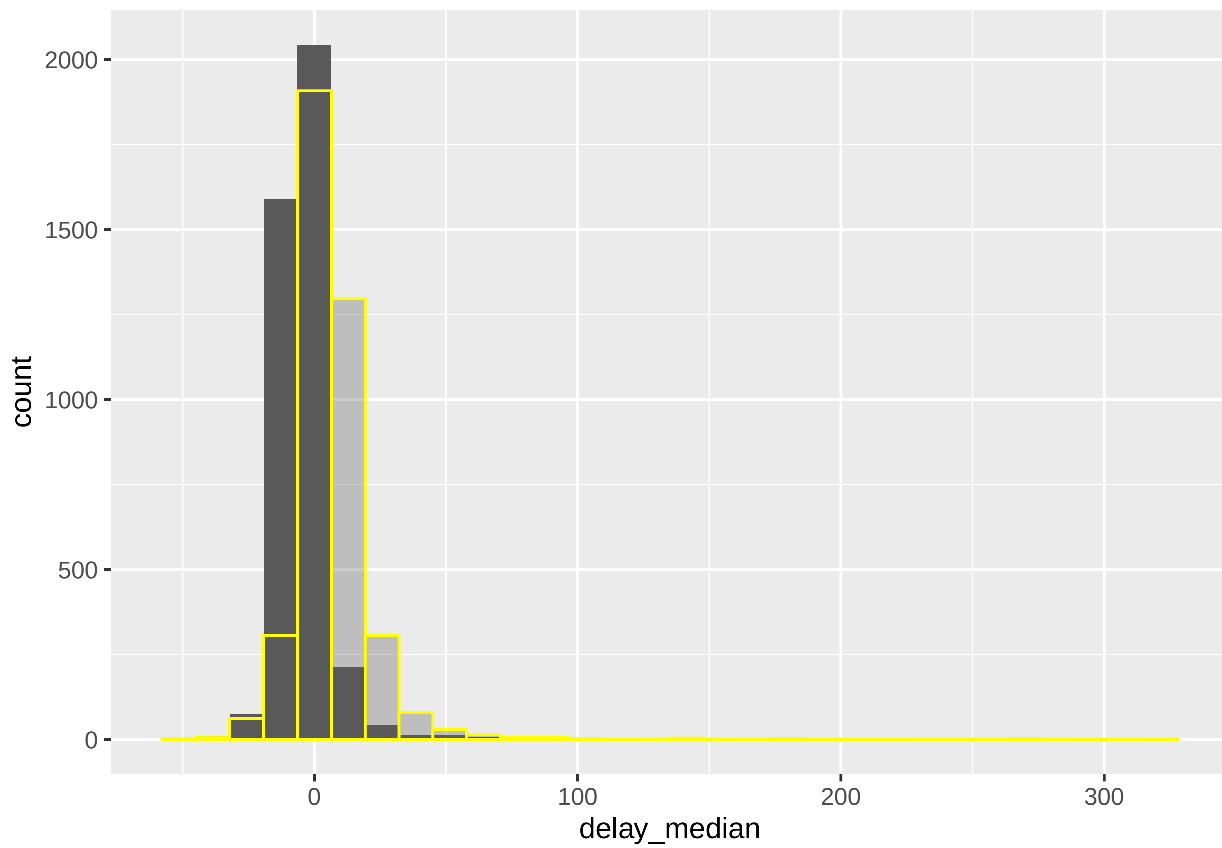
Since I need to filter the same thing, all the time just store in a variable. Delete other stuff.

```
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

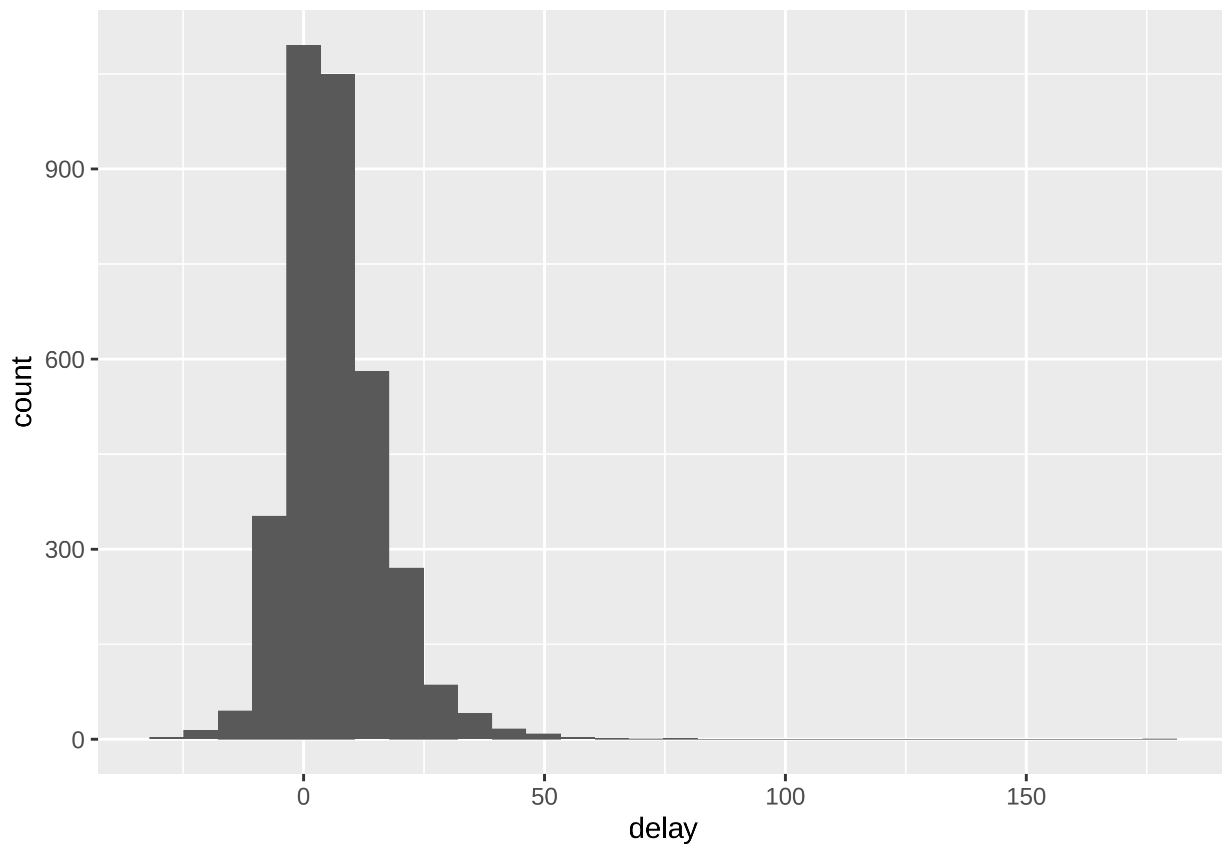
```
ggplot(data = not_missing_planes) +
  geom_histogram(mapping = aes(x = delay_median)) +
  geom_histogram(mapping = aes(x = delay), color = 'yellow', alpha = 0.3)
```

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




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

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



Filter the airplanes that fly rarely and pipe them into ggplot which gets plussed into geoms.

Try a few values for how many flights one should have done

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.
2. Grade Assignment 4 of your peers.

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.

1. Row_number() ERROR: is.na() applied to non-(list or vector) of type 'closure'Error in x[!nas] : object of type 'closure' is not subsettable

Was passing a dataframe to is.na() instead of vector

2.dense_rank() ERROR: Error in unique.default(x) : unique() applies only to vectors

I was using an array of strings instead of integers

3. del_flights <- filter(flights, min_rank(dep_delay)) Error: Argument 2 filter condition does not evaluate to a logical vector - learned how to use min_rank and filter together

Was using a dataframe with one column instead of a vector

4. min and filter Error in min_rank(., desc(dep_delay)) : unused argument (desc(dep_delay)) -

Using min_rank inside filter - min_rank was not evaluating to a boolean/logical vector because I was missing out a comparison statement

5. Error in plotting Vienna Data against Hotels Data ggplot(mapping = aes(x = vienna$price, y = hotels_{d}ata$price))

Both x and y need to be equal in length to make a scatter plot

6. Error in plotting side by side bar Could not figure out how to plot a side by side bar in ggplot - hit a number of errors and then gave up.

7. Had problems in leading ggarrange to combine multiple plots in one figure Error in ggarrange() : could not find function "ggarrange" Error in library(ggpubr) : there is no package called 'ggpubr'

Fixed by installing the ggpubr package

8. Error in using hjust parameter to adjust the horizontal allignment of the labels of the combined plots Error: unexpected symbol in: "ggarrange(vienna_plt, ams_plt, labels = c("Vienna“,"Amsterdam“), hjust = c(-1, -1) ggarrange"

Turns out we cannot adjust the horizontal allignement of both lables seperately but it has to be a common value for both - need to pass an int instead of a list of ints

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.

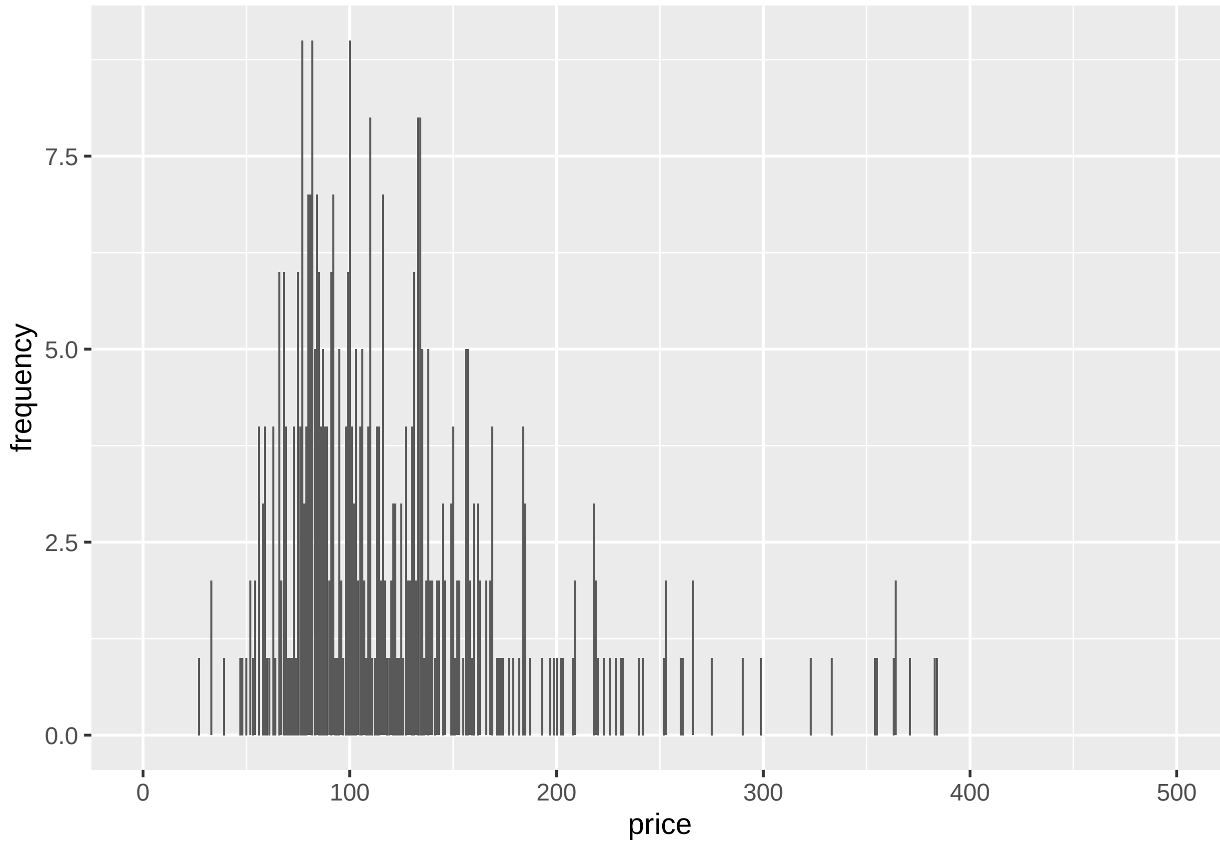
```
vienna <- read_csv('hotels-vienna.csv')
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   country = col_character(),
##   city_actual = col_character(),
##   center1label = col_character(),
##   center2label = col_character(),
##   neighbourhood = col_character(),
##   city = col_character(),
##   offer_cat = col_character(),
##   accommodation_type = col_character()
## )

## See spec(...) for full column specifications.
```

```
vienna %>% group_by(price) %>%
  summarise(frequency = n()) %>%
  ggplot(mapping = aes(x = price, y = frequency)) + geom_col() + xlim(0, 500)
```

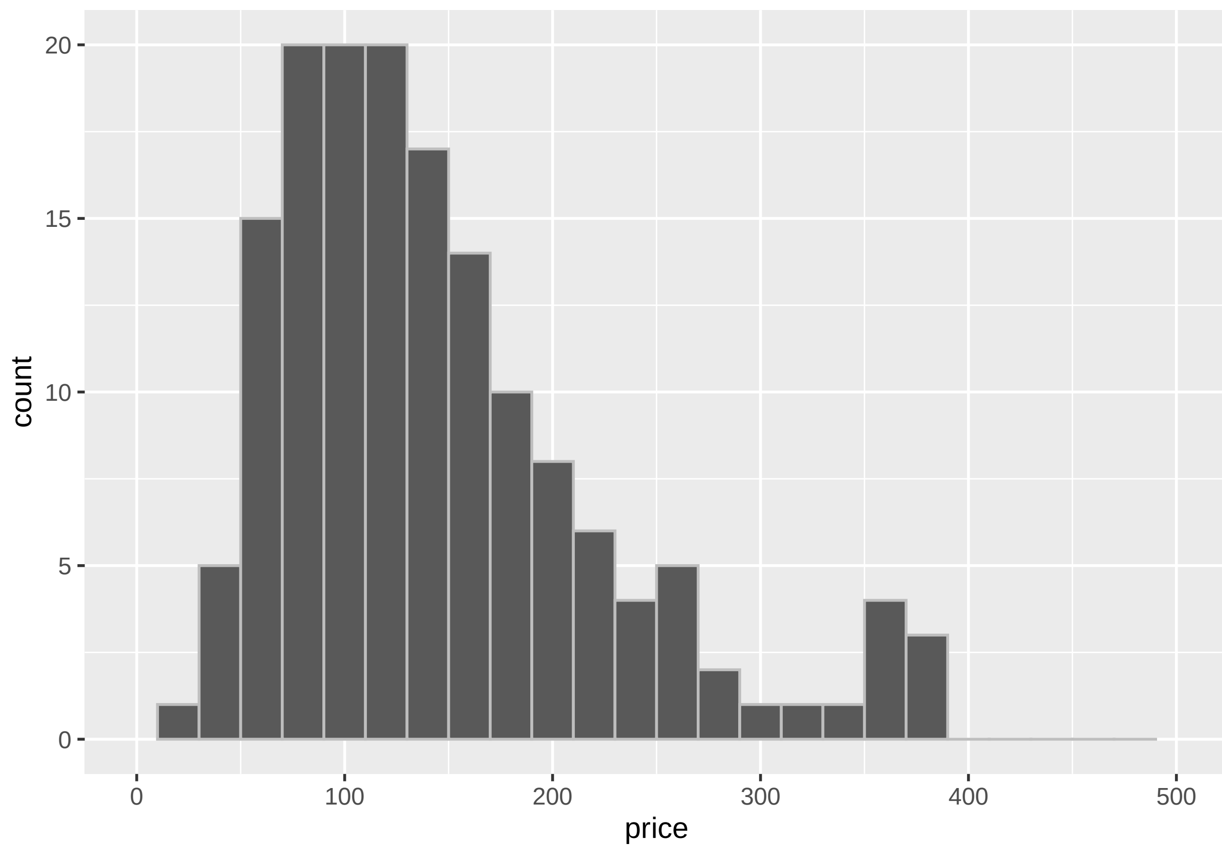
```
## Warning: Removed 7 rows containing missing values (position_stack).
```




```
vienna %>% group_by(price) %>%
  summarise(frequency = n()) %>%
  ggplot(mapping = aes(x = price)) + geom_histogram(binwidth = 20, color = 'Grey') + xlim(0, 500)
```

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

```
## Warning: Removed 2 rows containing missing values (geom_bar).
```



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).

```r
#two different sheets with price and distance (in _features sheet)
features <- read_csv('hotels-europe_features.csv')
```

```
## Parsed with column specification:
## cols(
##   hotel_id = col_double(),
##   city = col_character(),
##   distance = col_double(),
##   stars = col_double(),
##   rating = col_double(),
##   country = col_character(),
##   city_actual = col_character(),
##   rating_reviewcount = col_double(),
##   center1label = col_character(),
##   center2label = col_character(),
##   neighbourhood = col_character(),
##   ratingta = col_double(),
##   ratingta_count = col_double(),
##   distance_alter = col_double(),
##   accommodation_type = col_character()
## )
```

```r
price <- read_csv('hotels-europe_price.csv')
```

```
## Parsed with column specification:
## cols(
##   hotel_id = col_double(),
##   price = col_double(),
##   offer = col_double(),
##   offer_cat = col_character(),
##   year = col_double(),
##   month = col_double(),
##   weekend = col_double(),
##   holiday = col_double(),
##   nnights = col_double(),
##   scarce_room = col_double()
## )
```

```r
#merge the two tables
hotels_data <- merge(features, price, by = 'hotel_id')

ams_data <- filter(hotels_data, city == "Amsterdam" & stars == 3.0) #filtered by amsterdam

vienna_data <- vienna %>% filter(stars == 3.0)

v_stat <- vienna_data %>% group_by(city)

v_stat <- v_stat %>% summarise(mean = mean(price), median = median(price))
v_stat
```

```
## # A tibble: 1 x 3
##   city    mean median
##   <chr>  <dbl>  <dbl>
## 1 Vienna  106.     89
```

```r
ams_stat <- ams_data %>% group_by(city)

ams_stat <- ams_stat %>% summarise(mean = mean(price), median = median(price))
ams_stat
```

```
## # A tibble: 1 x 3
##   city       mean median
##   <chr>     <dbl>  <dbl>
## 1 Amsterdam  275.    171
```

Compare the prices of 3 star hotels in Vienna and Amsterdam

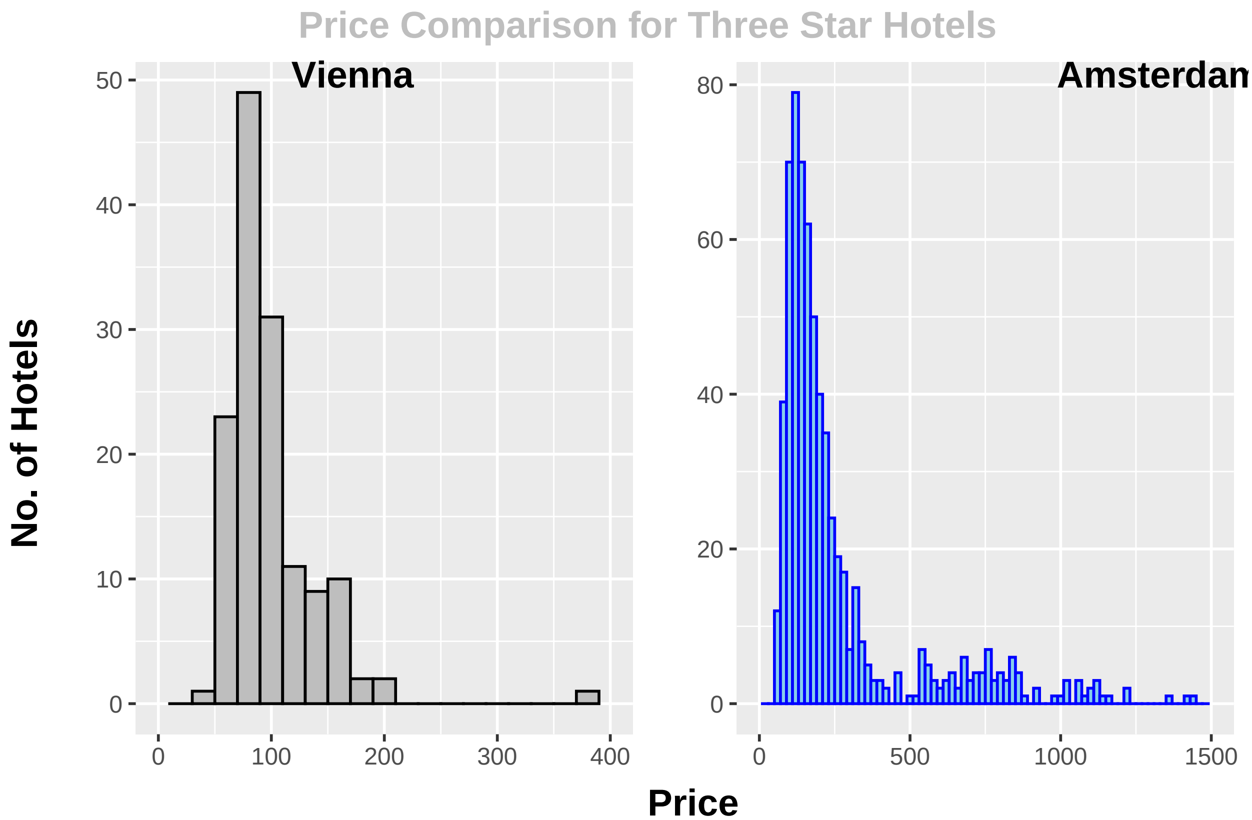
```r
vienna_plt <- vienna_data %>% ggplot(mapping = aes(x = price)) + geom_histogram(binwidth = 20, color =

ams_plt <- ams_data %>% filter(stars == 3.0) %>% ggplot(mapping = aes(x = price)) + geom_histogram(binw

combined_figure <- ggarrange(vienna_plt, ams_plt, labels = c("Vienna", "Amsterdam"), hjust = -2)

annotate_figure(combined_figure,
                top = text_grob("Price Comparison for Three Star Hotels", color = "Grey", face = "bold"
                bottom = text_grob("Price", vjust = -1,
                                   hjust = 0, x = 0.5, face = "bold", size = 14),
                left = text_grob("No. of Hotels", rot = 90, size = 14, face = "bold"),
```

```
)
```




```
ams_data <- filter(hotels_data, city == "Amsterdam") #filtered by amsterdam

vienna_data <- vienna

v_stat <- vienna_data %>% group_by(city)

v_stat <- v_stat %>% summarise(mean = mean(price), median = median(price))
v_stat
```

```
## # A tibble: 1 x 3
##   city    mean median
##   <chr>  <dbl>  <dbl>
## 1 Vienna  131.   110.
```

```
ams_stat <- ams_data %>% group_by(city)

ams_stat <- ams_stat %>% summarise(mean = mean(price), median = median(price))
ams_stat
```

```
## # A tibble: 1 x 3
##   city       mean median
##   <chr>     <dbl>  <dbl>
## 1 Amsterdam  318.    196
```

Compare the overall spread of prices of hotels in Vienna and Amsterdam

```
vienna_plt <- vienna_data %>% ggplot(mapping = aes(x = price)) + geom_histogram(binwidth = 20, color =

ams_plt <- ams_data %>% filter(stars == 3.0) %>% ggplot(mapping = aes(x = price)) + geom_histogram(binw
  labs(x = "", y = "")

combined_figure <- ggarrange(vienna_plt, ams_plt, labels = c("Vienna", "Amsterdam"), hjust = -2)

annotate_figure(combined_figure,
                top = text_grob("Price Comparison for all hotels in Amsterdam and Vienna",
                                color = "Grey", face = "bold", size = 14),
                bottom = text_grob("Price", vjust = -1,
                                   hjust = 0, x = 0.5, face = "bold", size = 14),
                left = text_grob("No. of Hotels", rot = 90, size = 14, face = "bold"),
)
```