# Assignment 5

*Veronika Palotai*

*2019 10 12*

## Lecture 4 Script

### Let's continue with chapter 5

#### mutate()

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

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_subset <- select(flights,
                         year:day,
                         ends_with('delay'),
                         distance,
                         air_time)

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

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

```
mutate(flights_subset,
       speed_km = (distance*1.61/air_time)*60)
```

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

```
minutes_per_hour <- 60
KM_PER_MILE <- 1.61 # usual way of introducing constants

mutate(flights_subset,
       speed_km = (distance * KM_PER_MILE/air_time)*60)
```

Even nicer is to create intermediate results for clarity

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

#### Transmute only keeps new variables

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

You cannot use any transformation 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, # divide integer-wise
          dep_minutes = dep_time %% 100 # modulo
          )
```

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

#### How can you test whether something is vectorized?

```
(x <- c(0,1,2,3,4,5,6,7,8,9))
(y <- 0:9) #same
(z <- seq(0,9)) #same

(lag(y))
(lag(lag(y))) # usually used for time series data, shifts elements

(lead(y)) # also used for ts data, shifts to the other side
```

#### What do lag and lead do?

lead() and lag() allow us to refer to leading or lagging values. This allows us to compute running differences (e.g. x - lag(x)) or find when values change (x != lag(x)). They are most useful in conjunction with group_by().

#### Some cumulative and aggregate functions

```
cumsum(x)
cumprod(x)
cumprod(lead(x))

?cummin
?cummean
cummean(x)
```

#### Logical operators work

```
x>3
x>y
x==y
(x)
```

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

# for vectorized operations with vectors that are not the
# same length, R repeats the shorter vector
```

### What does the answer to this even mean?

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

They cannot be compared because the longer is not a multiple of the shorter.

### Ranking functions

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

min_rank(y) #10 gets rank 5, 5 gets rank 2, and so on...
?sort
??sort # things that look like sort but are not specifically sort
```

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

It does the most usual type of ranking (e.g. 1st, 2nd, 2nd, 4th). The default gives smallest values the small ranks; use desc(x) to give the largest values the smallest ranks.

So, what is not a vectorized operation? non vectorized = does not return a vector, just a number.

```
kk <- function(x) {x[3]} # kk is now a func that returns the third element of a vector
kk(c(0,1,2,3,4))

mean(x)
```

What happens when we try this on a dataframe?

```
transmute(flights, delay = mean(arr_delay, na.rm = TRUE))
transmute(flights, delay = kk(arr_delay))
# repeat the value over and over again, returns the 3rd row of the column
```

Notice that it does not throw an error. It does something that makes sense, if it is what you want.

### EXERCISES

- Exercise: Try out a few of the other commands in the chapter.

```
x <- c(1:10)
cumsum(x)
```

```
##  [1]  1  3  6 10 15 21 28 36 45 55
```

```
cummean(x)
```

```
##  [1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5
```

```
cummin(x)
```

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

```
cummax(x)
```

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

```
cumprod(x)
```

```
##  [1]       1       2       6      24     120     720    5040   40320
##  [9]  362880 3628800
```

```
y <- c(1,2,2,NA,3,4)
min_rank(y)
```

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

```
row_number(y)
```

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

```
dense_rank(y)
```

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

```
percent_rank(y)
```

```
## [1] 0.00 0.25 0.25   NA 0.75 1.00
```

```
cume_dist(y)
```

```
## [1] 0.2 0.6 0.6  NA 0.8 1.0
```

```
ntile(y,2)
```

```
## [1]  1  1  1 NA  2  2
```

- 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.

```
x <- c(1:9)
y <- c(-1:5)
z <- c(-5:-2)
```

- Exercise: Read ? ":" (the same as help(":"))

The binary operator : has two meanings: for factors a:b is equivalent to interaction(a, b) (but the levels are ordered and labelled differently).

For other arguments from:to is equivalent to seq(from, to), and generates a sequence from from to to in steps of 1 or -1. Value to will be included if it differs from from by an integer up to a numeric fuzz of about 1e-7

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

```
first_10_rows <- slice(flights, 10, .preserve = TRUE)
```

- Do the following exercises from 5.5.2:

Exercise 1

```
mutate(flights,
       hour = dep_time %/% 100,
       minute = dep_time %% 100,
       mins_since_midnight = hour*60 + minute
)
```

```
## # A tibble: 336,776 x 20
##     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 13 more variables: sched_arr_time <int>,
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>, mins_since_midnight <dbl>
```

Exercise 2

What I expect is that 'air_time' is the difference between the 'arr_time' and 'dep_time'. In other words, air_time = arr_time - dep_time.

Since 'air_time' is given in minutes but 'arr_time' and 'dep_time' are not, they need to be converted into minutes.

```
transmute(flights,
          dep_time_in_mins = (dep_time %/% 100 * 60 + dep_time %% 100) %% 1440,
          arr_time_in_mins = (arr_time %/% 100 * 60 + arr_time %% 100) %% 1440,
          air_time,
          air_time_diff = arr_time_in_mins - dep_time_in_mins)
```

```
## # A tibble: 336,776 x 4
##    dep_time_in_mins arr_time_in_mins air_time air_time_diff
##               <dbl>            <dbl>    <dbl>         <dbl>
##  1              317              510      227           193
##  2              333              530      227           197
##  3              342              563      160           221
##  4              344              604      183           260
##  5              354              492      116           138
##  6              354              460      150           106
##  7              355              553      158           198
##  8              357              429       53            72
##  9              357              518      140           161
## 10              358              473      138           115
## # ... with 336,766 more rows
```

Contrary to expectations, they are not equal. One possible reason for this is that 'air_time' does not include time spent on the runway taxiing to and from gates whereas it is included in 'arr_time' and 'dep_time' so the relationship between these values might actually be 'air_time' <= 'arr_time' - 'dep_time'. However, in some cases this is not true which might be the result of 'arr_time' and 'dep_time' not being given in the same time zone.

Exercise 4

```
flights %>%
  mutate(dep_delay_min_rank = min_rank(desc(dep_delay))) %>%
  filter(dep_delay_min_rank <= 10) %>%
  arrange(dep_delay_min_rank)
```

```
## # A tibble: 10 x 20
##     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 13 more variables: sched_arr_time <int>, arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
## #   time_hour <dttm>, dep_delay_min_rank <int>
```

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))
# computes the mean of dep_delay and puts it into a new column
```

How... useful. Might as well do

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

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

```
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
```

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

```
flights$dep_delay # unreadable format
select(flights, dep_delay)
# takes a data frame returns a data frame (one column in this case)
```

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.

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

Looks distinctly the same. But it really isn't!

```
summarise(by_day,
          delay = mean(dep_delay, na.rm = TRUE)
          ) # for each group it will perform the second argument
```

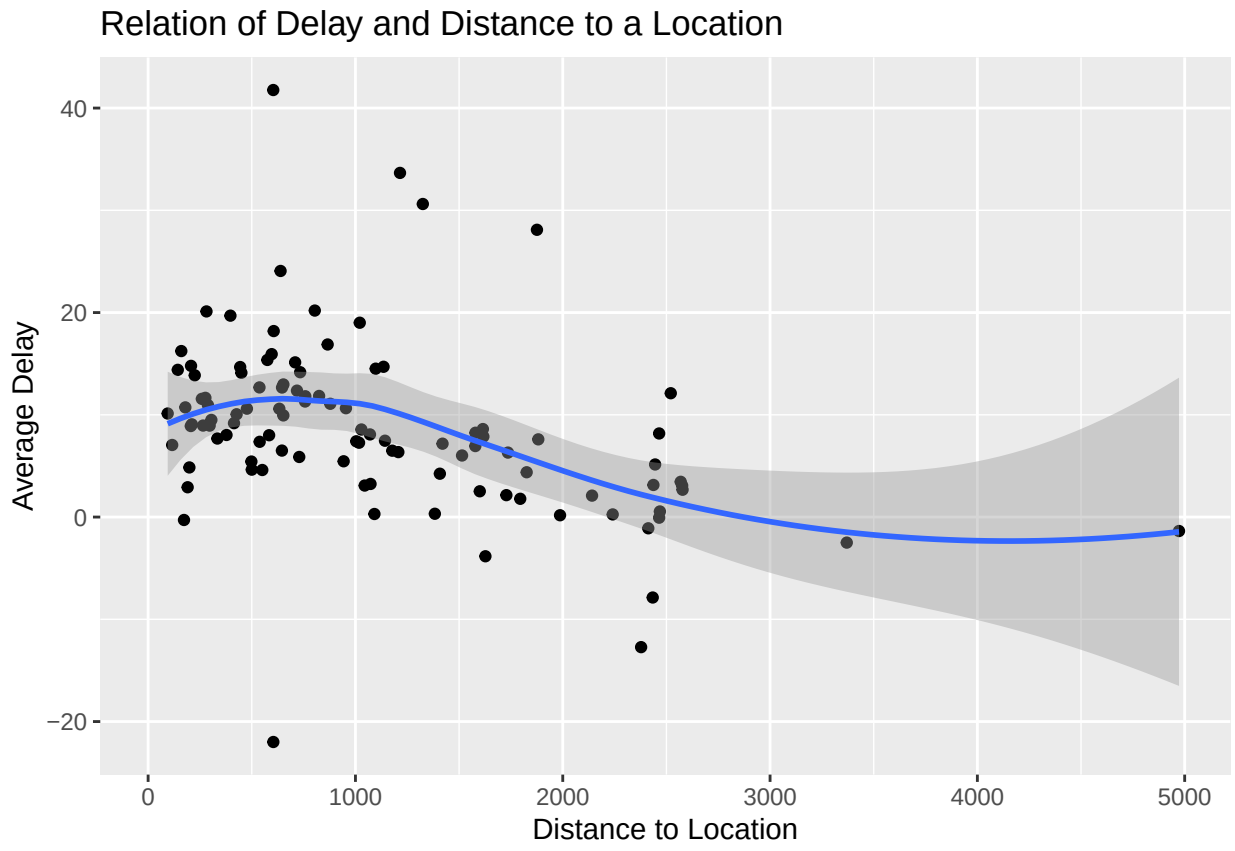
### 5.6.1

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
```

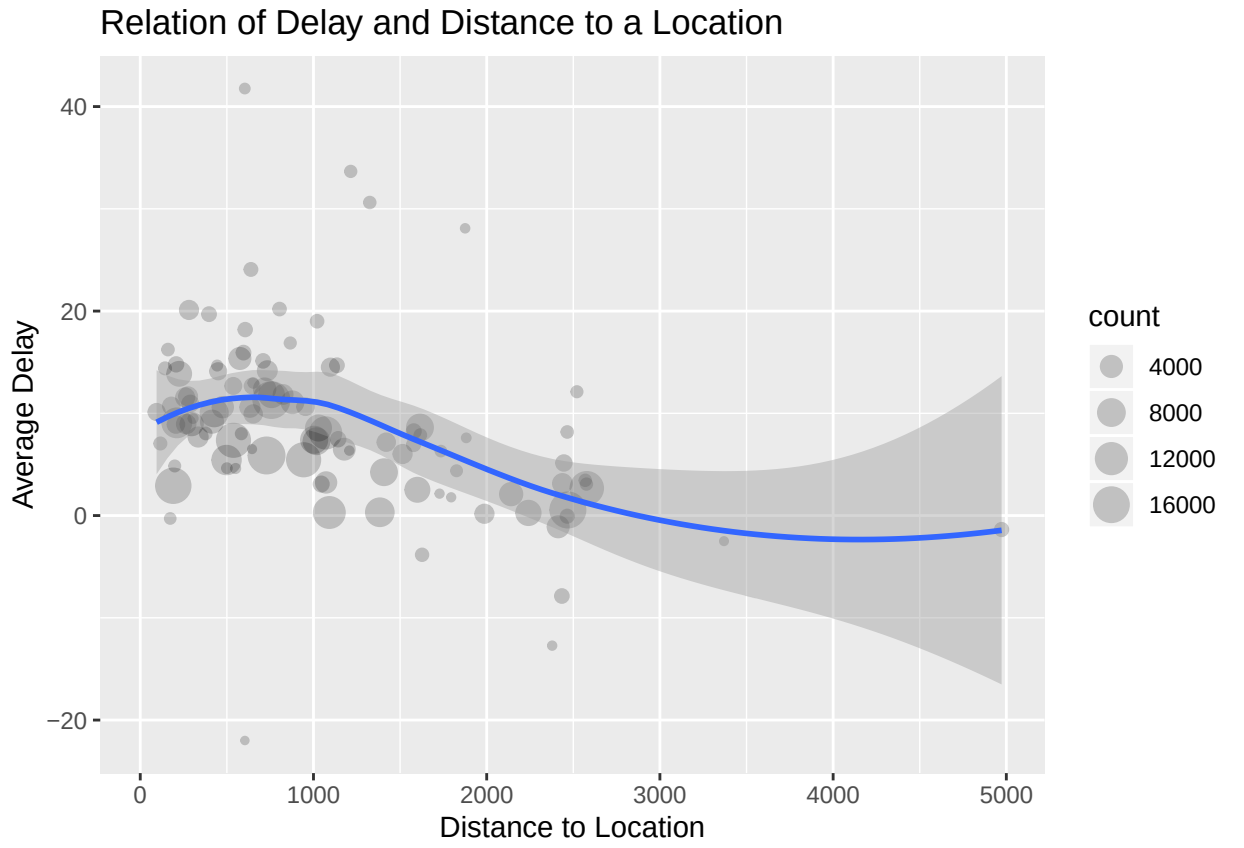
OK, we need the distance too, or else there is not much to plot.

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

p <- ggplot(data = delay,
            mapping = aes(x = distance, y = delay))

p + geom_point() + geom_smooth() +
  labs(x = "Distance to Location",
       y = "Average Delay",
       title = "Relation of Delay and Distance to a Location")
```




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

```
                    distance = mean(distance, na.rm = TRUE)))

# n counts the nr of observations/items in sth
# (in this case it will count the nr of observations per group)

p <- ggplot(data = delay,
            mapping = aes(x = distance, y = delay))

p + geom_point(mapping = aes(size = count), alpha = 0.2) + geom_smooth() +
  labs(x = "Distance to Location",
       y = "Average Delay",
       title = "Relation of Delay and Distance to a Location")
```




```
# smoothing line is misleading in this case
```

n() is a very special function, n() # should only be called in a data context

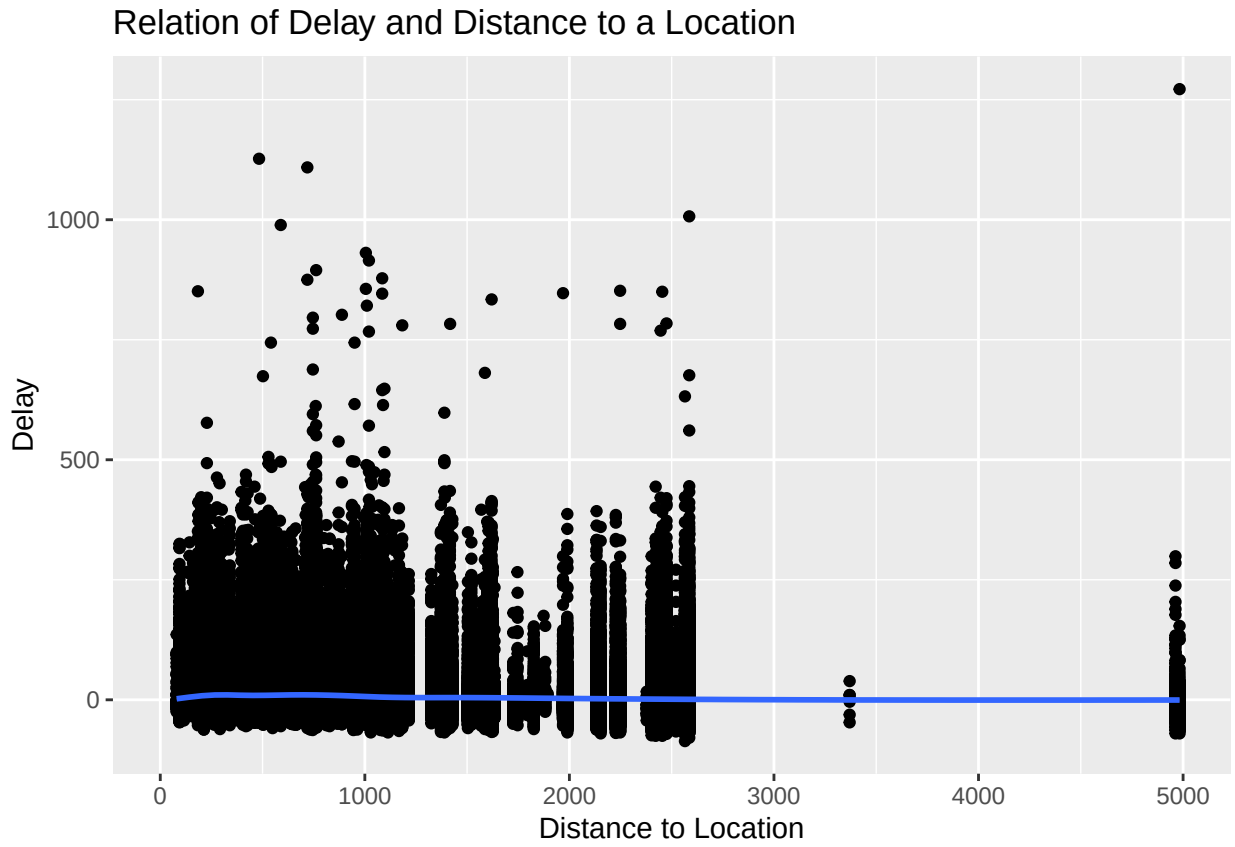
Finally...

*Exercise* as part of assignment 5: 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.

*Answer:* 'n()' counts the number of observations/items in something in this case it will count the number of observations per group (flights per location). This is why the circles representing each location differ in size.

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

```
p <- ggplot(data = flights,
            mapping = aes(x = distance, y = arr_delay))

p + geom_point() + geom_smooth() +
  labs(x = "Distance to Location",
       y = "Delay",
       title = "Relation of Delay and Distance to a Location")
```




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

```
by_destination <- group_by(flights, dest)

by_destination %>%
  summarise(count = n()) %>%
  filter(count>20)
```

```
## # A tibble: 97 x 2
##    dest  count
##    <chr> <int>
##  1 ABQ     254
##  2 ACK     265
##  3 ALB     439
```

```
##  4 ATL   17215
##  5 AUS    2439
##  6 AVL     275
##  7 BDL     443
##  8 BGR     375
##  9 BHM     297
## 10 BNA    6333
## # ... with 87 more rows
```

- to HNL (Honululu), since it's by far the furthest

```
by_destination <- group_by(flights, dest)

by_destination %>%
  filter(dest!='HNL')
```

```
## # A tibble: 336,069 x 19
## # Groups:   dest [104]
##     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 336,059 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>
```

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.

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

*Answer:* In my opinion, the one with piping is easier to read.

```
by_destination <- group_by(flights, dest)

filter(by_destination, dest!='HNL')
```

```
## # A tibble: 336,069 x 19
## # Groups:   dest [104]
##     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 336,059 more rows, and 12 more variables: sched_arr_time <int>,
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>
```

### 5.6.2 Missing values

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

has_dep_delay_only <- flights %>%
  filter(is.na(arr_delay))

has_arr_delay_only <- flights %>%
  filter(is.na(dep_delay))
```

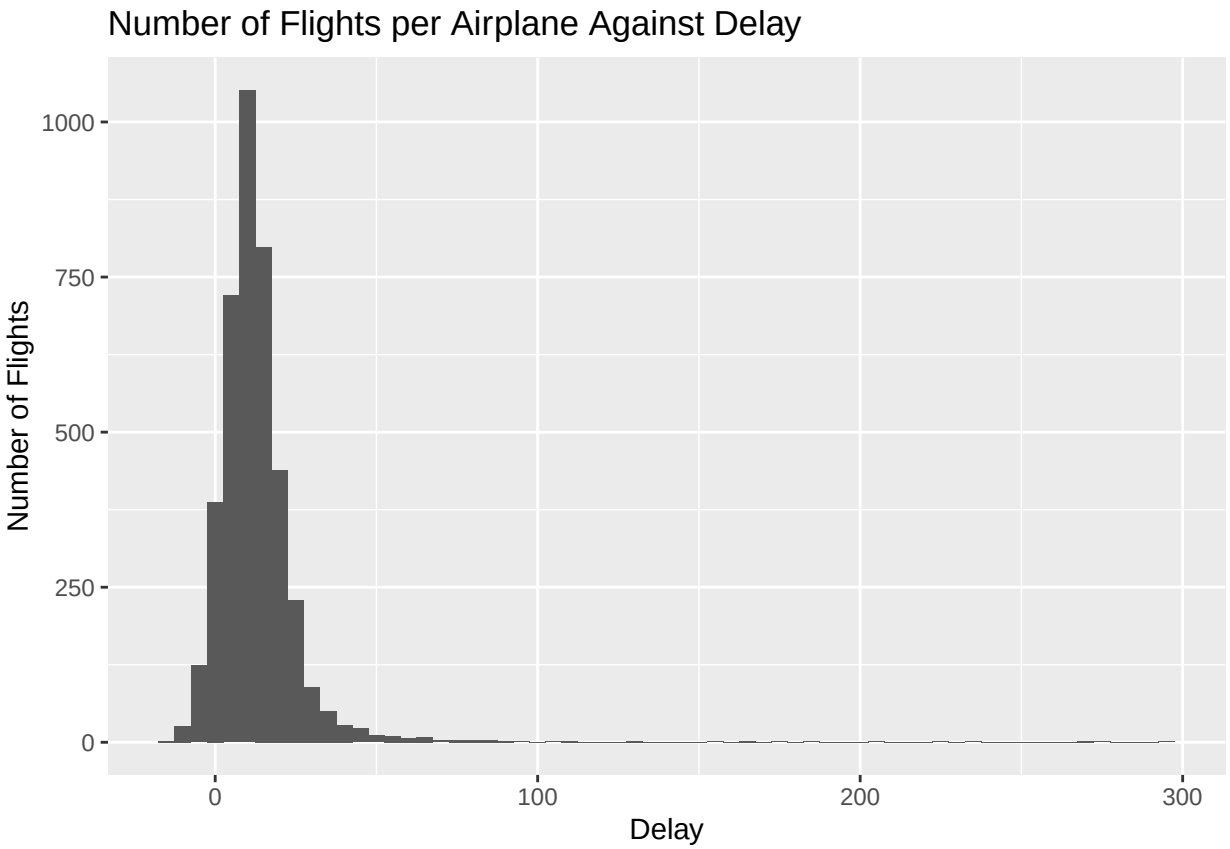
*Exercise:* Does the above command also drop observations that miss only the arr_delay but have a dep_delay? Are there any observations in the dataset for which only dep_delay or arr_delay is missing, but not both?

*Answer:* Comma means AND so it drops those for which one condition is not met (or both). Apparently, there are observtions for which only 'dep_delay' or 'arr_delay' is missing.

### 5.6.3 Counts

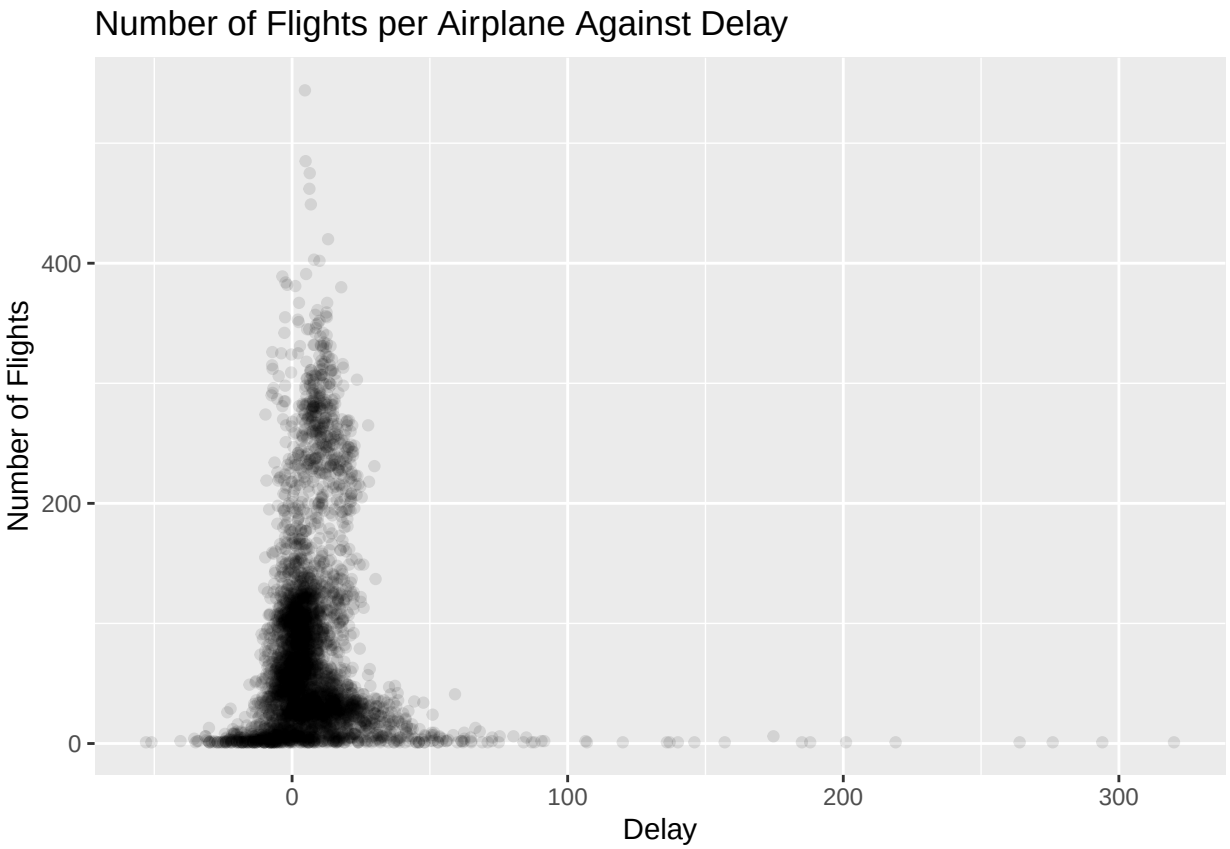
Average delay by airplane (identified by tailnum), plot density. Start with freqpoly, then zoom in on that part of the graph that we are interested.

```
not_missing %>%
  group_by(tailnum) %>%
  summarise(delay = mean(dep_delay)) %>%
  ggplot(mapping = aes(x=delay)) +
  geom_histogram(binwidth = 5) +
  labs(x = "Delay",
       y = "Number of Flights",
       title = "Number of Flights per Airplane Against Delay")
```




**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) +
  labs(x = "Delay",
       y = "Number of Flights",
       title = "Number of Flights per Airplane Against Delay")
```

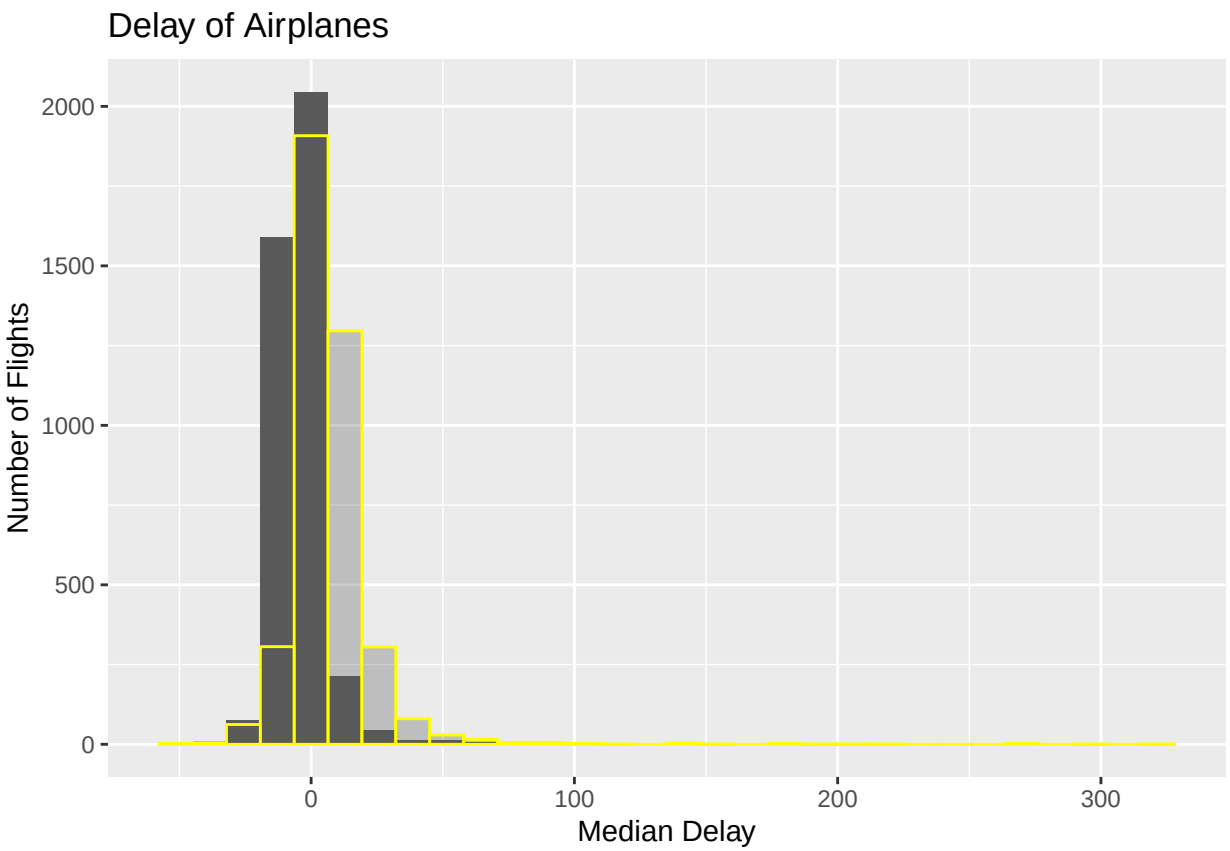



Since I need to filter the same thing, all the time just store in a variable. Delete other stuff.

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

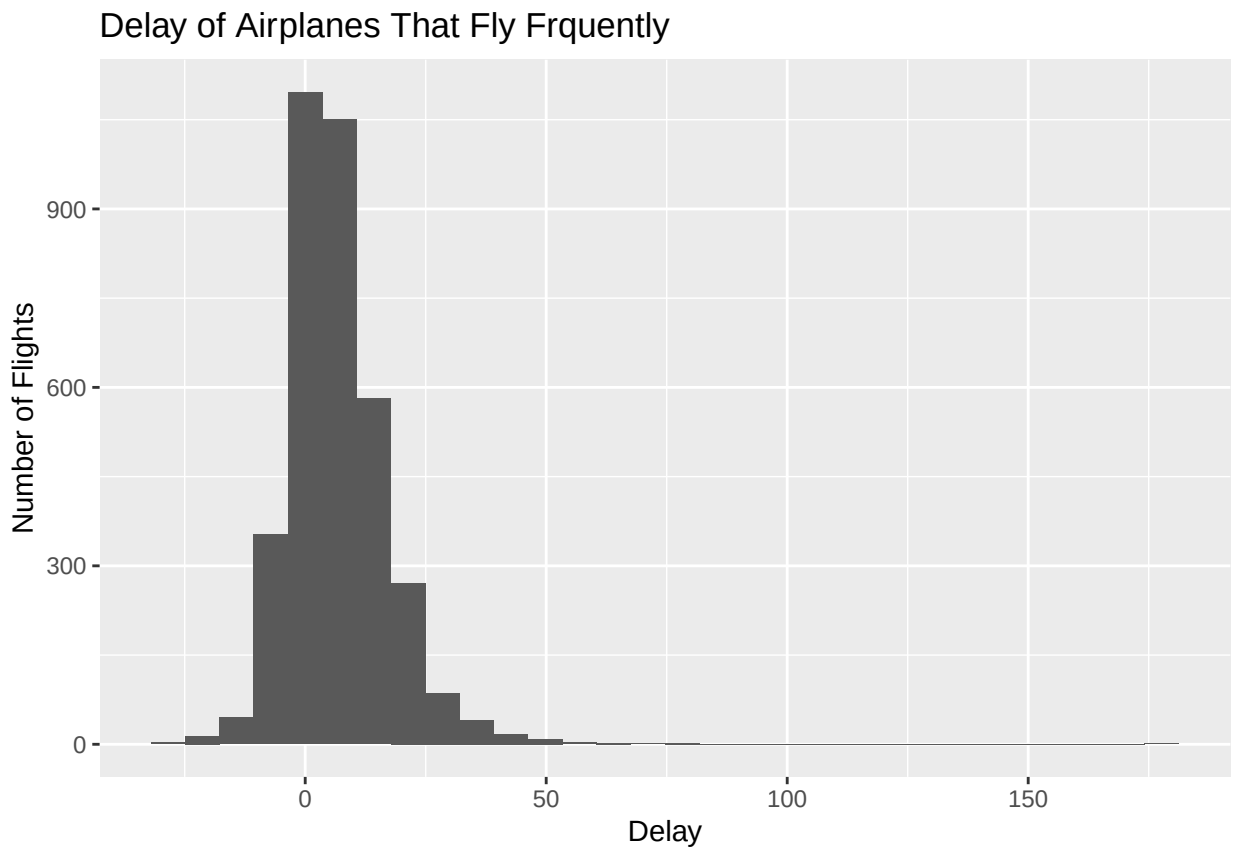
Get the median delay for each ariplane

```
ggplot(data = planes_with_not_missing_values) +
  geom_histogram(mapping = aes(x = delay_median)) +
  geom_histogram(mapping = aes(x = delay), color = 'yellow', alpha = 0.3) +
  labs(x = "Median Delay",
       y = "Number of Flights",
       title = "Delay of Airplanes")
```




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.

```
planes_with_not_missing_values %>%
  filter(count > 5) %>%
  ggplot(mapping = aes(x = delay)) +
  geom_histogram() +
  labs(x = "Delay",
       y = "Number of Flights",
       title = "Delay of Airplanes That Fly Frquently")
```




### 5.6.4 Summary functions

These need to turn a vector of things into a single thing.

What does quantile do?

Not that helpful. Let's use it and find out.

*Exercise:* Find out what these do

```
first(x) # returns first element of x
first(c(3, 4, 2)) # returns the first element from (3,4,2)
last(x) # returns last element of x
nth(x, 2) # returns the element from x with index = 2
```

Counts are important. Count the number of flights to each destination.

```
by_destination <- group_by(flights, dest)

summarise(by_destination, count = n())
```

```
## # A tibble: 105 x 2
##    dest  count
##    <chr> <int>
##  1 ABQ     254
##  2 ACK     265
##  3 ALB     439
##  4 ANC       8
##  5 ATL   17215
```

```
##  6 AUS    2439
##  7 AVL     275
##  8 BDL     443
##  9 BGR     375
## 10 BHM     297
## # ... with 95 more rows
```

Count the number of distinct carriers to each location.

```
(by_destination <- flights %>%
  group_by(dest) %>%
  summarise(length(unique(carrier))))
```

```
## # A tibble: 105 x 2
##    dest  `length(unique(carrier))`
##    <chr>                     <int>
##  1 ABQ                           1
##  2 ACK                           1
##  3 ALB                           1
##  4 ANC                           1
##  5 ATL                           7
##  6 AUS                           6
##  7 AVL                           2
##  8 BDL                           2
##  9 BGR                           2
## 10 BHM                           1
## # ... with 95 more rows
```

You can weight the counting, here by distance. This counts how many airmiles a given airplane did from NYC

```
(by_destination <- flights %>%
  group_by(dest) %>%
  summarise(length(unique(carrier)),sum(distance)))
```

```
## # A tibble: 105 x 3
##    dest  `length(unique(carrier))` `sum(distance)`
##    <chr>                     <int>           <dbl>
##  1 ABQ                           1          463804
##  2 ACK                           1           52735
##  3 ALB                           1           62777
##  4 ANC                           1           26960
##  5 ATL                           7        13033618
##  6 AUS                           6         3693263
##  7 AVL                           2          160485
##  8 BDL                           2           51388
##  9 BGR                           2          141750
## 10 BHM                           1          257201
## # ... with 95 more rows
```

Number of flights each day before 5am.

```
by_day <- flights %>%
  group_by(year, month, day) %>%
  filter(dep_time < 500) %>%
  summarise(count = n())
```

How many flights are delayed each day by more than 1 hour?

```r
by_day <- flights %>%
  group_by(year, month, day) %>%
  filter(dep_delay > 60) %>%
  summarise(count = n())

by_day
```

```
## # A tibble: 365 x 4
## # Groups:   year, month [12]
##     year month   day count
##    <int> <int> <int> <int>
##  1  2013     1     1    51
##  2  2013     1     2    80
##  3  2013     1     3    53
##  4  2013     1     4    43
##  5  2013     1     5    26
##  6  2013     1     6    34
##  7  2013     1     7    41
##  8  2013     1     8    22
##  9  2013     1     9    13
## 10  2013     1    10    21
## # ... with 355 more rows
```

## 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. 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.
3. 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.

List of Errors

- Error: unexpected ')' in: " group_by(year, month, day) %>% filter(dep_delay > 60))". *Reason:* I forgot one opening bracket at the beginning of the code.
- Error: object 'by_day' not found. *Reason:* I did not run the chunk that defined 'by_day' therefore the function where I used it threw an error as it could not find it.
- Error: Column `distance` must be length 1 (a summary value), not 254. *Reason:* I ran into this error when I wanted to use the 'summarise()' function. It collapses a data frame to a single row but I had 254 different values which could not have been collapsed becuase I forgot to use the 'sum()' function that added these values up and resulted in one integer.
- Error: Column `unique(carrier)` must be length 1 (a summary value), not 7. *Reason:* Same as before, I forgot to use the 'length()' function which gives that one particular integer I was interested in.
- Error in unique(carr) : object 'carr' not found. *Reason:* I did not write the name of the column correctly, as it is carrier instead of carr.
- Error in filter(dest != "HNL") : object 'dest' not found. *Reason:* I ran this code: ' by_destination <- filter(dest!='HNL')' where I forgot to include the data frame flights.

- Error: attempt to use zero-length variable name. *Reason:* I accidentally ran the first row of a code chunk where the embedded commands are specified.
- Error: Aesthetics must be valid data columns. Problematic aesthetic(s): size = count. Did you mistype the name of a data column or forget to add stat()? *Reason:* I did not run that part of the chunk where I defined count.
- Error: Column `flights` is of unsupported class data.frame. *Reason:* I ran the code 'top_10_delayed %>% mutate(flights,dep_delay_min_rank = min_rank(desc(dep_delay))) %>% filter(top_10_delayed, dep_delay_min_rank <= 10) %>% arrange(top_10_delayed, dep_delay_min_rank)' where I did the piping incorrectly.
- Error in eval(lhs, parent, parent) : object 'first_10_rows' not found. *Reason:* I ran the code 'first_10_rows %>% slice(flights, 10, .preserve = TRUE)'. Yet again, I did the piping incorrectly, flights should have come before the 'slice()' function.

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.

My choice: Figure 3.1 Histogram of hotel stars, frequency/count plot

CLEAR MEMORY

```
rm(list=ls())
```

Import Libraries

```
# install.packages("scales")
library(ggplot2)
library(tidyverse)
library(scales)
theme_set(theme_bw())
```

Setting the Path

```
dir <-  "D:/Egyetem/CEU/Coding_1/R-Coding/"
```

Location Folders

```
data_in <- paste0(dir,"da_data_repo/hotels-vienna/clean/")
data_out <-  paste0(dir,"da_case_studies/ch03-hotels-vienna-explore/")
output <- paste0(dir,"da_case_studies/ch03-hotels-vienna-explore/output/")
func <- paste0(dir, "da_case_studies/ch00-tech-prep/")
```

Hotels Vienna

Loading Dataset

```
vienna <- read_csv(paste0(data_in,"hotels-vienna.csv"))
```

View Dataset

```
View(vienna)
```

Grouping by stars and counting observations within each group

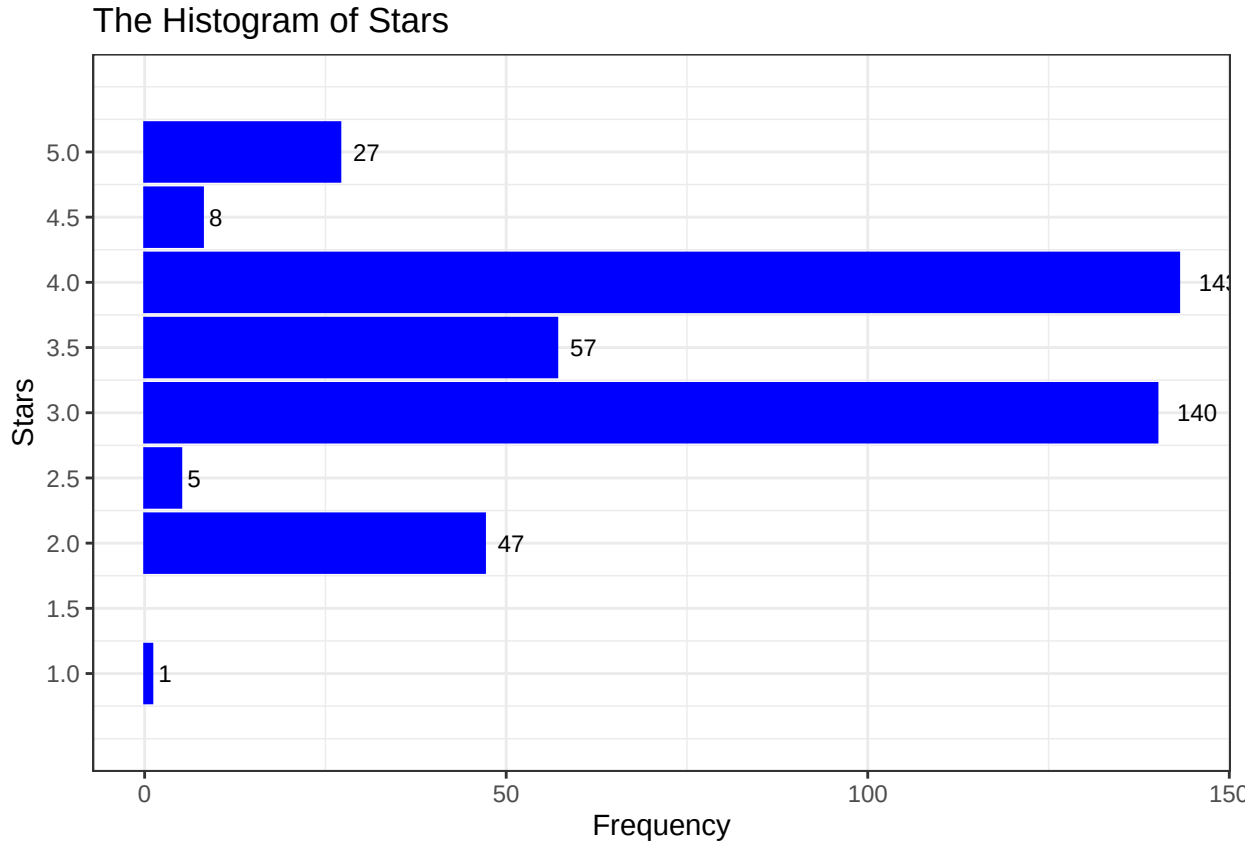
```
by_stars <- vienna %>% group_by(stars)

(stars_count <- by_stars %>% summarise(count = n()))
```

```
## # A tibble: 8 x 2
##   stars count
```

```
##   <dbl> <int>
## 1   1       1
## 2   2      47
## 3   2.5     5
## 4   3     140
## 5   3.5    57
## 6   4     143
## 7   4.5     8
## 8   5      27
```

Creating the histogram plot

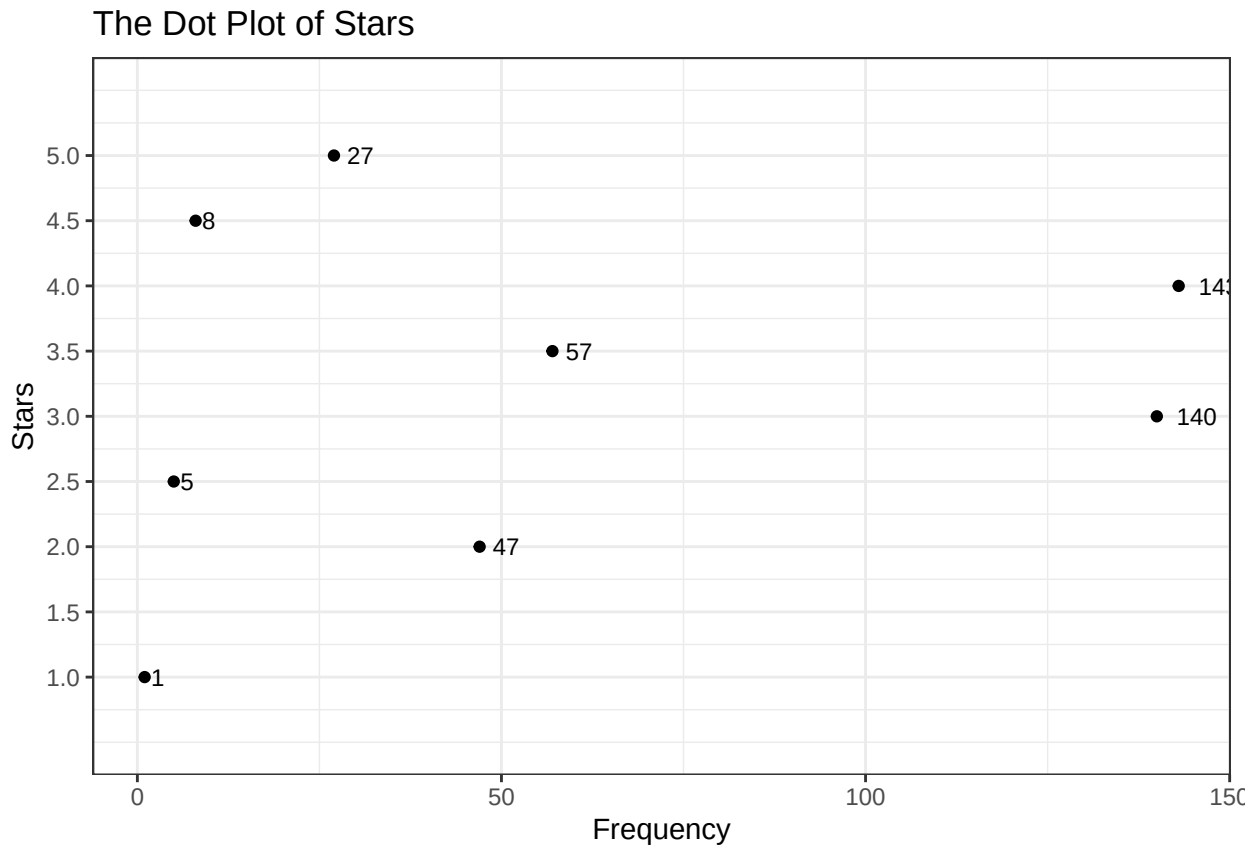
```
(p <- ggplot(data=by_stars, aes(x=stars)) +
  geom_bar(color="blue", fill="blue", stat="count") +
  geom_text(stat='count', aes(label=..count..), hjust=-0.5, size = 3) +
  scale_x_continuous(limits = c(0.5,5.5), breaks = seq(1, 5, 0.5)) +
  coord_flip() +
  labs(x = "Stars",
       y = "Frequency",
       title = "The Histogram of Stars"))
```




When compared to the graph in the book, it can be noticed that the number of hotels in each group are higher in the replaced one. One possible reason for this is that the dataset has been updated and new hotels have been added to it.

A variation of this graph: a dot plot, as we are only interested in the ends of the bars.

```
(p<-ggplot(by_stars) +
   geom_point(stat="count", aes(x = stars)) +
   geom_text(stat="count", aes(x=stars, label=..count..), hjust=-0.5, size = 3) +
   scale_x_continuous(limits = c(0.5,5.5), breaks = seq(1, 5, 0.5)) +
   coord_flip() +
   labs(x = "Stars",
        y = "Frequency",
        title = "The Dot Plot of Stars"))
```




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).

Chosen city: Dublin, Ireland

Import libraries

```
library(rlang)
```

Set the path

```
dir <-  "D:/Egyetem/CEU/Coding_1/R-Coding/"
```

Location Folders

```
data_in <- paste0(dir,"da_data_repo/hotels-europe/clean/")
data_out <-  paste0(dir,"da_case_studies/ch03-hotels-europe-compare/")
output <- paste0(dir,"da_case_studies/ch03-hotels-europe-compare/output/")
```

```
func <- paste0(dir, "da_case_studies/ch00-tech-prep/")
```

Load in clean and tidy data and create workfile

```
hotels_europe_price <- read_csv(paste0(data_in,"hotels-europe_price.csv"))
hotels_europe_features <- read_csv(paste0(data_in,"hotels-europe_features.csv"))

hotels_europe <- left_join(hotels_europe_price, hotels_europe_features, by = "hotel_id")
rm(hotels_europe_price)
rm(hotels_europe_features)
```

View Dataset

```
View(hotels_europe)
```

Histogram of stars in both cities

Filtering the data based on city

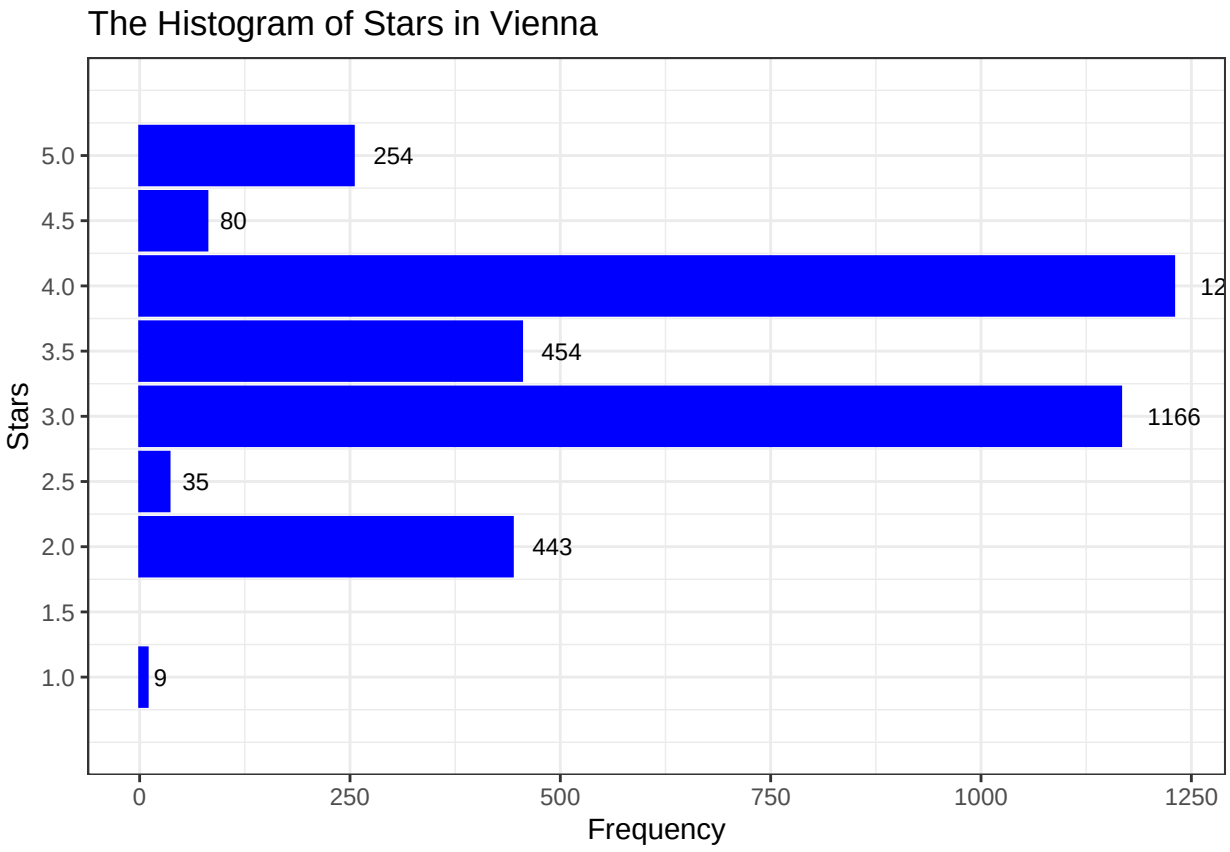
```
hotels_vienna <- hotels_europe %>%
  filter(city=='Vienna')

hotels_dublin <- hotels_europe %>%
  filter(city=='Dublin')
```

**Vienna**

```
by_stars_vienna <- hotels_vienna %>% group_by(stars)

(stars_count_vienna <- by_stars_vienna %>% summarise(count = n()))
```

```
## # A tibble: 8 x 2
##   stars count
##   <dbl> <int>
## 1   1       9
## 2   2     443
## 3   2.5    35
## 4   3    1166
## 5   3.5   454
## 6   4    1229
## 7   4.5    80
## 8   5     254
```

```
(p <- ggplot(data=by_stars_vienna, aes(x=stars)) +
  geom_bar(color="blue", fill="blue", stat="count") +
  geom_text(stat='count', aes(label=..count..), hjust=-0.5, size = 3) +
  scale_x_continuous(limits = c(0.5,5.5), breaks = seq(1, 5, 0.5)) +
  coord_flip() +
  labs(x = "Stars",
       y = "Frequency",
       title = "The Histogram of Stars in Vienna"))
```

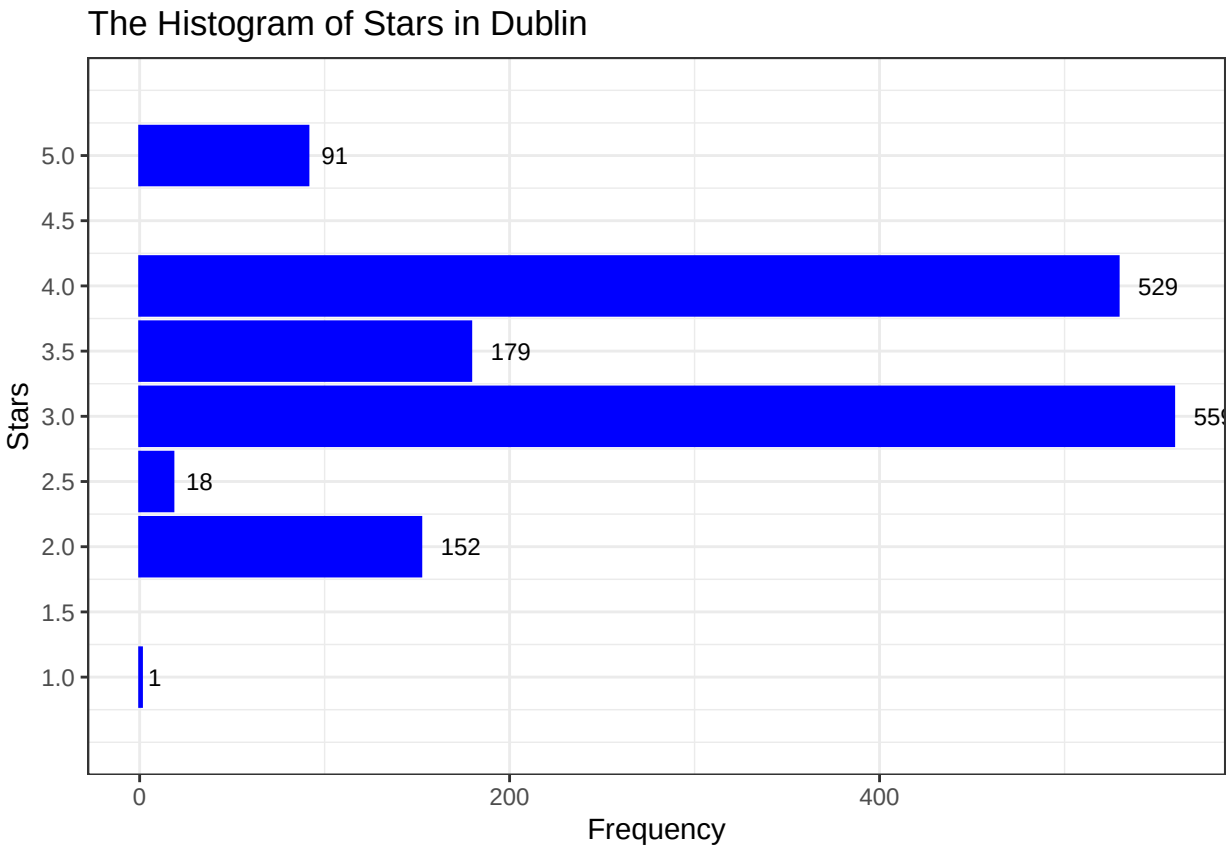



**Dublin**

```
by_stars_dublin <- hotels_dublin %>% group_by(stars)

(stars_count_dublin <- by_stars_dublin %>% summarise(count = n()))
```

```
## # A tibble: 7 x 2
##   stars count
##   <dbl> <int>
## 1   1       1
## 2   2     152
## 3   2.5    18
## 4   3     559
## 5   3.5   179
## 6   4     529
## 7   5      91
```

```
(p <- ggplot(data=by_stars_dublin, aes(x=stars)) +
  geom_bar(color="blue", fill="blue", stat="count") +
  geom_text(stat='count', aes(label=..count..), hjust=-0.5, size = 3) +
  scale_x_continuous(limits = c(0.5,5.5), breaks = seq(1, 5, 0.5)) +
  coord_flip() +
  labs(x = "Stars",
       y = "Frequency",
       title = "The Histogram of Stars in Dublin"))
```

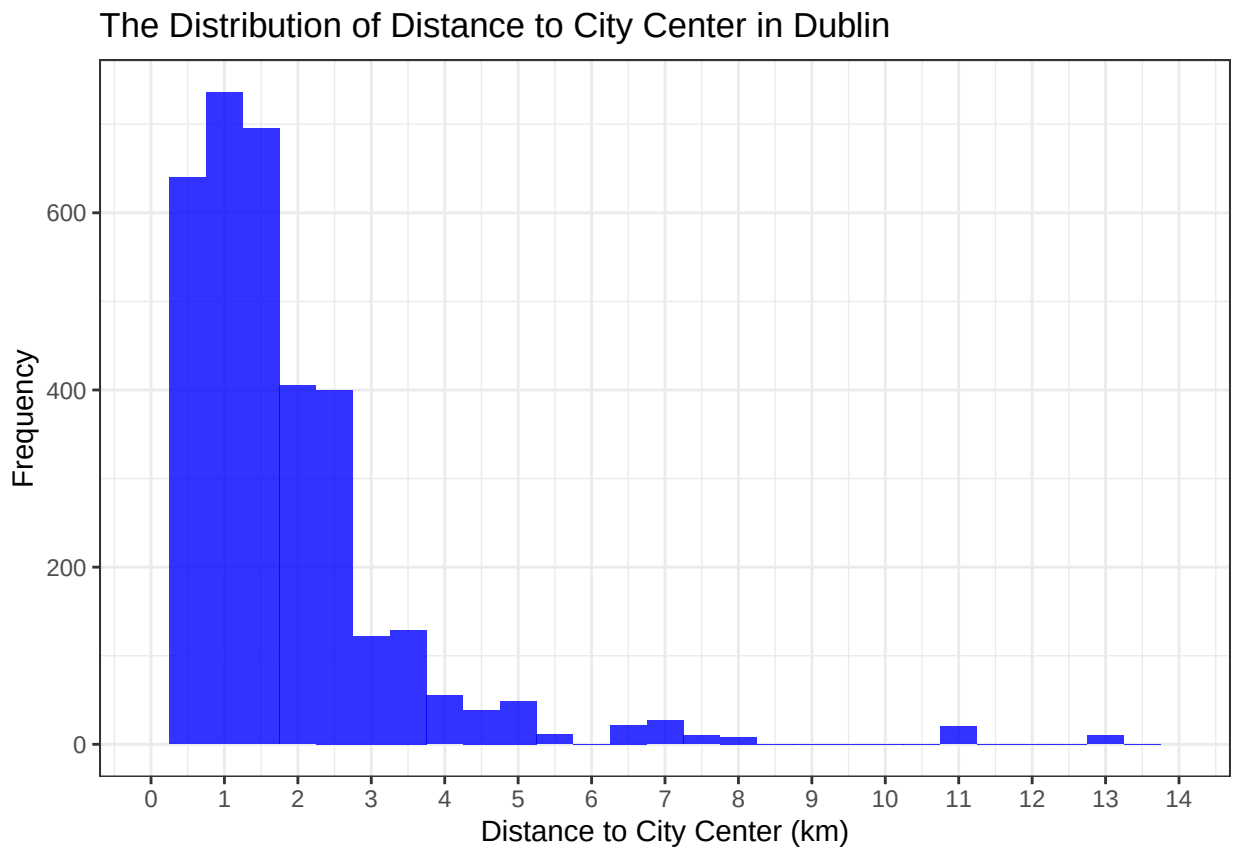



The frequency distribution of hotels with certain stars is quite similar in the two city although their modes differ. For Vienna this is at 4.0 stars with a count of 1229 hotels, for Dublin the mode is at 3.0 stars with a count of 559. There are fewer hotels with extra high or extra low ratings in both cases and '.5' ratings also don't tend to be common in either case.

First, let's look at the hotels' distances from the city centre in each city.

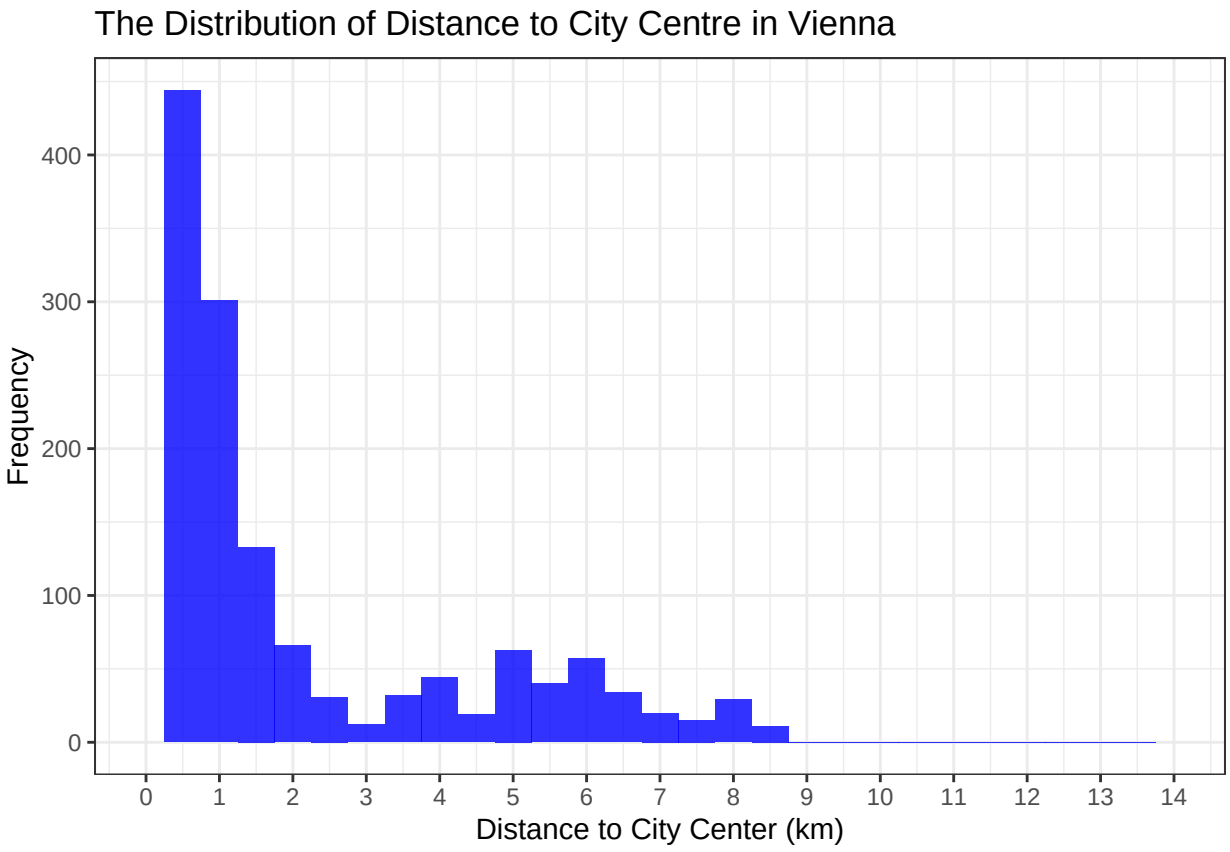
**Vienna**

```
ggplot(data = hotels_vienna, aes (x = distance, y = (..count..))) +
  geom_histogram(binwidth = 0.5, fill = "blue", size = 0.25, alpha = 0.8,  show.legend=F, na.rm =TRUE) +
  labs(x = "Distance to City Center (km)", y = "Frequency", title="The Distribution of Distance to City
  scale_x_continuous(limits = c(0, 14), breaks = seq(0, 14, by = 1))
```




**Dublin**

```
ggplot(data = hotels_dublin, aes (x = distance, y = (..count..))) +
  geom_histogram(binwidth = 0.5, fill = "blue", size = 0.25, alpha = 0.8,  show.legend=F, na.rm =TRUE) +
  labs(x = "Distance to City Center (km)", y = "Frequency", title="The Distribution of Distance to City
  scale_x_continuous(limits = c(0, 14), breaks = seq(0, 14, by = 1))
```




Both in Vienna and in Dublin most of the hotels are within 2 kms from the city center. In case of Vienna, the mode is around 1 km while for Dublin the mode is around 0.5 km.

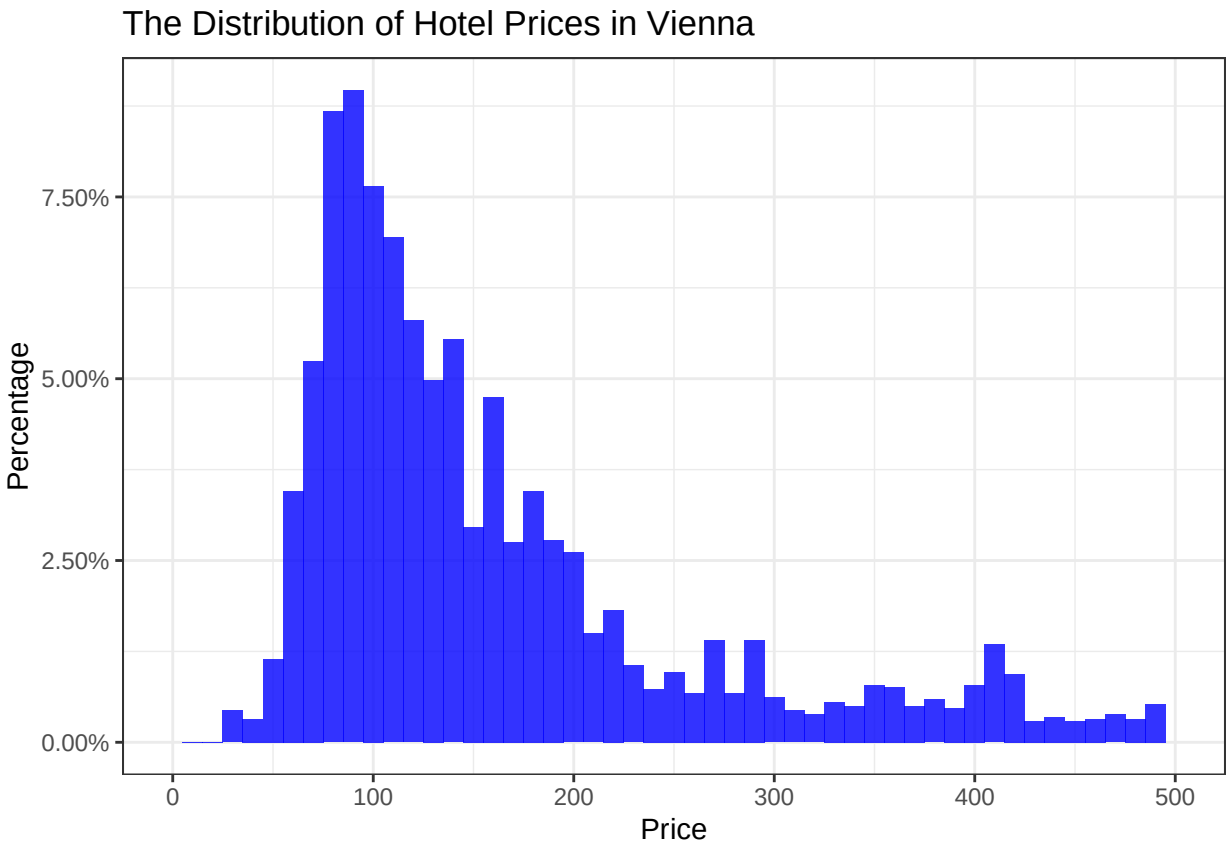
On the Vienna plot there are a few extreme values which is the result of taking those hotels also into consideration which are closer to the airport rather than the city.

For Dublin no such values can be observed, however, after the initial decrease with the distance, there is a slight increase in the number of hotels that are 5 to 6 kms from the center.

Next, let's look at the relative price distribution of hotels in each city.

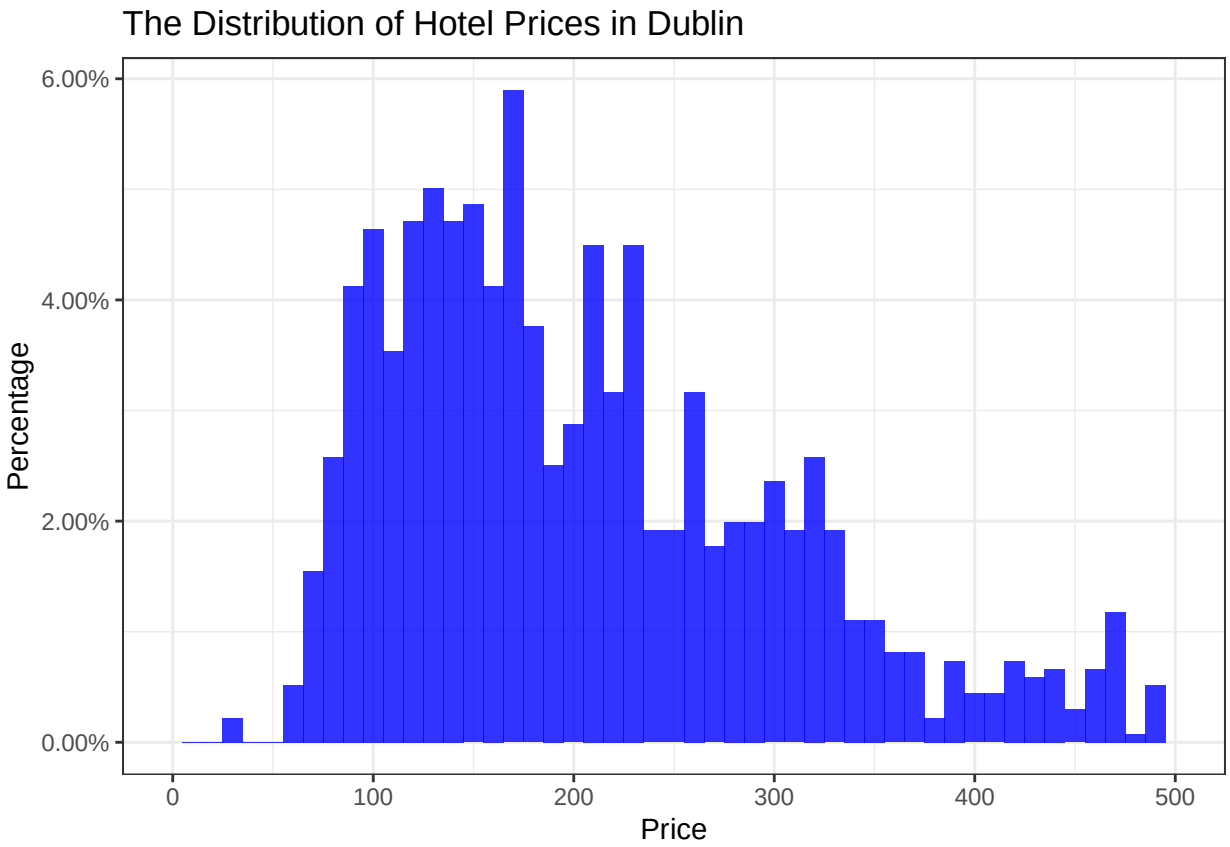
**Vienna**

```
ggplot(data = hotels_vienna, aes (x = price, y = (..count..)/sum(..count..))) +
  geom_histogram(binwidth = 10, fill = "blue", size = 0.25, alpha = 0.8, show.legend=F, na.rm =TRUE) +
  labs(x = "Price", y = "Percentage", title="The Distribution of Hotel Prices in Vienna") +
  scale_x_continuous(limits = c(0, 500), breaks = seq(0, 500, by = 100)) +
  scale_y_continuous(labels = scales::percent_format())
```




**Dublin**

```
ggplot(data = hotels_dublin, aes (x = price, y = (..count..)/sum(..count..))) +
  geom_histogram(binwidth = 10, fill = "blue", size = 0.25, alpha = 0.8,  show.legend=F, na.rm =TRUE) +
  labs(x = "Price", y = "Percentage", title="The Distribution of Hotel Prices in Dublin") +
  scale_x_continuous(limits = c(0, 500), breaks = seq(0, 500, by = 100)) +
  scale_y_continuous(labels = scales::percent_format())
```




One of the tendencies that can be observed is that prices in Dublin tend to be higher than in Vienna which is not surprising as Dublin is usually listed as quite an expensive city in Europe. On the Vienna related plot we can observe a long right tail which means that we have a few observations with large values with most observations having smaller values. This kind of skewness is quite common when we are looking at distributions of prices, incomes or population.

The mode of the histogram is at 90 euros for Vienna and 170 euros for Dublin.

This time there are no extreme values in either case. If there were, they could be the results of certain events (e.g. a conference) in that particular hotel.