Assignment 7

Instructions: Make an R Notebook and in it answer the question or questions below. When you are done, hand in on Quercus the *output* from Previewing (or Knitting) your Notebook, probably an `html` or `pdf` file. An `html` file is easier for the grader to deal with. Do *not* hand in the Notebook itself. You want to show that you can (i) write code that will answer the questions, (ii) run that code and get some sensible output, (iii) write some words that show you know what is going on and that reflect your conclusions about the data. Your goal is to convince the grader that you *understand* what you are doing: not only doing the right thing, but making it clear that you know *why* it's the right thing.

Do *not* expect to get help on this assignment. The purpose of the assignments is for you to see how much *you* have understood. You will find that you also learn something from grappling with the assignments. The time to get help is after you watch the lectures and work through the problems from PASIAS, via tutorial and the discussion board, that is *before* you start work on the assignment. The only reasons to contact the instructor while working on an assignment are to report (i) something missing like a data file that cannot possibly be read, (ii) something *beyond your control* that makes it impossible to finish the assignment in time after you have started it.

There is a time limit on this assignment (you will see Quercus counting down the time remaining).

1. In 2013, there were over 300,000 flights that departed from one of New York City's three airports. The package `nycflights13` contains information about all these flights, as well as details about the airports, airlines and planes referenced in the flights dataset, and also information about the weather in each hour of 2013 (that can be matched up with the flights). Install the package using the usual `install.packages`, and load it. When you load the package, dataframes called `airlines`, `airports`, `flights`, `planes` and `weather` are available to you.

   (a) Using the `airports` dataframe, display the names of New York City's three major airports. They have airport codes EWR, JFK, LGA.

Solution:

Take a look at the `airports` dataframe first:

```
airports
```

```
## # A tibble: 1,458 x 8
##    faa   name                      lat    lon   alt    tz dst   tzone
##    <chr> <chr>                   <dbl>  <dbl> <dbl> <dbl> <chr> <chr>
##  1 04G   Lansdowne Airport        41.1  -80.6  1044    -5 A     America/New_Yo~
##  2 06A   Moton Field Municipal A~ 32.5  -85.7   264    -6 A     America/Chicago
##  3 06C   Schaumburg Regional      42.0  -88.1   801    -6 A     America/Chicago
##  4 06N   Randall Airport          41.4  -74.4   523    -5 A     America/New_Yo~
##  5 09J   Jekyll Island Airport    31.1  -81.4    11    -5 A     America/New_Yo~
##  6 0A9   Elizabethton Municipal ~ 36.4  -82.2  1593    -5 A     America/New_Yo~
##  7 0G6   Williams County Airport  41.5  -84.5   730    -5 A     America/New_Yo~
##  8 0G7   Finger Lakes Regional A~ 42.9  -76.8   492    -5 A     America/New_Yo~
##  9 0P2   Shoestring Aviation Air~ 39.8  -76.6  1000    -5 U     America/New_Yo~
## 10 0S9   Jefferson County Intl    48.1 -123.    108    -8 A     America/Los_An~
```

```
## # ... with 1,448 more rows
```

The airport codes are in the `faa` column, so:

```
airports %>% filter(faa == "EWR" | faa == "LGA" | faa == "JFK")
```

```
## # A tibble: 3 x 8
##   faa   name                  lat   lon   alt    tz dst   tzone
##   <chr> <chr>               <dbl> <dbl> <dbl> <dbl> <chr> <chr>
## 1 EWR   Newark Liberty Intl  40.7 -74.2    18    -5 A     America/New_York
## 2 JFK   John F Kennedy Intl  40.6 -73.8    13    -5 A     America/New_York
## 3 LGA   La Guardia           40.8 -73.9    22    -5 A     America/New_York
```

They are called Newark Liberty (which is actually in New Jersey), John F Kennedy, and La Guardia.

The code uses `|` to represent "or" in a `filter`. Another way to do it is to define a set of abbreviations as a vector, and then search for `faa` values "in" a set:

```
nyc_airports <- c("EWR", "JFK", "LGA")
airports %>% filter(faa %in% nyc_airports)
```

```
## # A tibble: 3 x 8
##   faa   name                  lat   lon   alt    tz dst   tzone
##   <chr> <chr>               <dbl> <dbl> <dbl> <dbl> <chr> <chr>
## 1 EWR   Newark Liberty Intl  40.7 -74.2    18    -5 A     America/New_York
## 2 JFK   John F Kennedy Intl  40.6 -73.8    13    -5 A     America/New_York
## 3 LGA   La Guardia           40.8 -73.9    22    -5 A     America/New_York
```

These are the best way to do it. Inferior ways are finding the airports one at a time, and (even more inferior) scanning the airports dataframe by eye, which doesn't in any case answer the question.

(b) How many flights departed from each of these airports, in the `flights` dataframe?

Solution:

This is actually simple, but you can overthink yourself into something more complicated very easily.

First look at `flights` and see that the column you want is called `origin`. Then count them:

```
flights %>% count(origin)
```

```
## # A tibble: 3 x 2
##   origin      n
##   <chr>   <int>
## 1 EWR    120835
## 2 JFK    111279
## 3 LGA    104662
```

Approximately 120,000 out of Newark, 111,000 out of John F Kennedy, nearly 105,000 out of La Guardia.

Extra: if you want to include the airport *names* on this, you can look them up with `left_join`. This creates a lot of columns, so I would then use `select` to just display the ones I want:

```
flights %>% count(origin) %>%
  left_join(airports, by = c("origin" = "faa")) %>%
  select(name, n)
```

```
## # A tibble: 3 x 2
##   name                     n
##   <chr>                <int>
## 1 Newark Liberty Intl 120835
## 2 John F Kennedy Intl 111279
## 3 La Guardia          104662
```

The reason for the `by` on the `left_join` is that the airports we want to look up the names of are called `origin` in `flights` and `faa` in `airports`. (FAA stands for "Federal Aviation Administration", one of whose responsibilities is maintaining a unique three-character code for each airport and airfield.)

This idea of having short codes in the main database and a separate place where you can look them up is very common; for maintaining or accessing the main database, it is much easier to enter codes compared to full names, and you can always get hold of the full names if you need them. In a business database, you might have codes for customers or transaction types and then separate databases with fuller descriptions and maybe other columns, as here. This is related to what database people call "normal form", limiting duplicated information.

(c) Display the *names* of the five most common destination airports for flights from the New York City airports (taken together), along with the number of flights to each.

Solution:

This begins like the previous part, noting that the column you want is called `dest`:

```
flights %>% count(dest)
```

```
## # A tibble: 105 x 2
##    dest      n
##    <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
```

Then arrange in descending order by `n`, and slice off the top 5:

```
flights %>% count(dest) %>%
  arrange(desc(n)) %>%
  slice(1:5)
```

```
## # A tibble: 5 x 2
```

```
##   dest      n
##   <chr> <int>
## 1 ORD   17283
## 2 ATL   17215
## 3 LAX   16174
## 4 BOS   15508
## 5 MCO   14082
```

Another way is this:

```
flights %>% count(dest) %>%
  slice_max(n, n=5)
```

```
## # A tibble: 5 x 2
##   dest      n
##   <chr> <int>
## 1 ORD   17283
## 2 ATL   17215
## 3 LAX   16174
## 4 BOS   15508
## 5 MCO   14082
```

I can't remember (as I write this) whether I've shown you this last one anywhere or not. It's probably a good idea to cite your source if you do it this way. The first way is definitely something you've seen in lecture.

Now you need to think about your reader. I happen to know four of these airports (Chicago[1] Atlanta, Los Angeles and Boston are the first four), but it's asking a lot to expect your reader to know that. So you need to translate these to actual airport names. You could do this by looking in the `airports` dataframe for these abbreviations by hand and then adding the names to your answer, but I think you should be suspecting that there is a better, code, way to do this, and it's the same idea as the Extra to the previous part: look up the abbreviations in the `airports` dataframe using `left_join`, and then grab only the columns you want to display, like this:

```
flights %>% count(dest) %>%
  arrange(desc(n)) %>%
  slice(1:5) %>%
  left_join(airports, by = c("dest" = "faa")) %>%
  select(name, dest, n)
```

```
## # A tibble: 5 x 3
##   name                                dest      n
##   <chr>                               <chr> <int>
## 1 Chicago Ohare Intl                  ORD   17283
## 2 Hartsfield Jackson Atlanta Intl     ATL   17215
## 3 Los Angeles Intl                    LAX   16174
## 4 General Edward Lawrence Logan Intl  BOS   15508
## 5 Orlando Intl                        MCO   14082
```

This is the best way to do it. The airports are O'Hare in Chicago, Atlanta, Los Angeles, Logan in Boston, and Orlando.

Extra: Some of these are big cities, and some of them not so much. I think Atlanta and Orlando are "hub cities"; airlines tend to have a "base" with a lot of flights into it, and if you want to

go somewhere else, you change planes in the hub city.

Well, we have airline codes in the `flights` dataframe, so let's investigate this. I'm pretty sure that Atlanta is a hub for Delta Airlines, so let's look at Atlanta first. Step one is to look only at the flights with destination Atlanta:

```
flights %>%
  filter(dest == "ATL")
```

```
## # A tibble: 17,215 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     1      554            600        -6      812            837
##  2  2013     1     1      600            600         0      837            825
##  3  2013     1     1      606            610        -4      837            845
##  4  2013     1     1      615            615         0      833            842
##  5  2013     1     1      658            700        -2      944            939
##  6  2013     1     1      754            759        -5     1039           1041
##  7  2013     1     1      807            810        -3     1043           1043
##  8  2013     1     1      814            810         4     1047           1030
##  9  2013     1     1      830            835        -5     1052           1105
## 10  2013     1     1      855            859        -4     1143           1145
## # ... with 17,205 more rows, and 11 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

A mere 17,000 flights. The airline info is in the column called `carrier`:

```
flights %>%
  filter(dest == "ATL") %>%
  count(carrier) %>%
  arrange(desc(n))
```

```
## # A tibble: 7 x 2
##   carrier     n
##   <chr>   <int>
## 1 DL      10571
## 2 FL       2337
## 3 MQ       2322
## 4 EV       1764
## 5 UA        103
## 6 9E         59
## 7 WN         59
```

I have even less clue about airline abbreviations than airport ones, so we definitely need to look these up. Let's examine the `airlines` dataframe:

```
airlines
```

```
## # A tibble: 16 x 2
##    carrier name
##    <chr>   <chr>
##  1 9E      Endeavor Air Inc.
##  2 AA      American Airlines Inc.
##  3 AS      Alaska Airlines Inc.
```

```
##  4 B6      JetBlue Airways
##  5 DL      Delta Air Lines Inc.
##  6 EV      ExpressJet Airlines Inc.
##  7 F9      Frontier Airlines Inc.
##  8 FL      AirTran Airways Corporation
##  9 HA      Hawaiian Airlines Inc.
## 10 MQ      Envoy Air
## 11 OO      SkyWest Airlines Inc.
## 12 UA      United Air Lines Inc.
## 13 US      US Airways Inc.
## 14 VX      Virgin America
## 15 WN      Southwest Airlines Co.
## 16 YV      Mesa Airlines Inc.
```

This is going to be easier than the other ones, because the airline abbreviations have the same name `carrier` in both dataframes, and there are no other columns that have the same name in either dataframe:

```
flights %>%
  filter(dest == "ATL") %>%
  count(carrier) %>%
  arrange(desc(n)) %>%
  left_join(airlines)
```

```
## Joining, by = "carrier"
```

```
## # A tibble: 7 x 3
##   carrier     n name
##   <chr>   <int> <chr>
## 1 DL      10571 Delta Air Lines Inc.
## 2 FL       2337 AirTran Airways Corporation
## 3 MQ       2322 Envoy Air
## 4 EV       1764 ExpressJet Airlines Inc.
## 5 UA        103 United Air Lines Inc.
## 6 9E         59 Endeavor Air Inc.
## 7 WN         59 Southwest Airlines Co.
```

Looks like my guess of Delta was on the money. What about Orlando? Wikipedia says that this is a hub for JetBlue and Southwest, among others. The same code idea will work again:

```
flights %>%
  filter(dest == "MCO") %>%
  count(carrier) %>%
  arrange(desc(n)) %>%
  left_join(airlines)
```

```
## Joining, by = "carrier"
```

```
## # A tibble: 4 x 3
##   carrier     n name
##   <chr>   <int> <chr>
## 1 B6       6472 JetBlue Airways
## 2 DL       3663 Delta Air Lines Inc.
## 3 UA       3217 United Air Lines Inc.
## 4 AA        730 American Airlines Inc.
```

JetBlue for sure, but what happened to Southwest? Do they fly out of New York at all, and if so, where to? We need their abbreviation first:

```
airlines %>%
  filter(str_detect(name, "Southwest"))
```

```
## # A tibble: 1 x 2
##   carrier name
##   <chr>   <chr>
## 1 WN      Southwest Airlines Co.
```

I did something a little different here: I didn't know what the exact `name` would be, so I asked for the rows where the text `Southwest` was found in the name somewhere.[2]

All right, let's find the flights with this carrier, count their destinations, and look them up:

```
flights %>%
  filter(carrier == "WN") %>%
  count(dest) %>%
  arrange(desc(n)) %>%
  left_join(airports, by = c("dest"="faa")) %>%
  select(name, dest, n)
```

```
## # A tibble: 11 x 3
##    name                              dest      n
##    <chr>                             <chr> <int>
##  1 Chicago Midway Intl               MDW    4113
##  2 Lambert St Louis Intl             STL    1469
##  3 Denver Intl                       DEN    1404
##  4 William P Hobby                   HOU    1401
##  5 General Mitchell Intl             MKE    1310
##  6 Nashville Intl                    BNA    1284
##  7 Phoenix Sky Harbor Intl           PHX     431
##  8 Austin Bergstrom Intl             AUS     298
##  9 Louis Armstrong New Orleans Intl  MSY     298
## 10 Baltimore Washington Intl         BWI     208
## 11 Hartsfield Jackson Atlanta Intl   ATL      59
```

If you wanted to fly Southwest from New York to Orlando, you couldn't do it direct, but I imagine you could fly to one of those airports first and from there to Orlando.

(d) One of the main hazards to air travel is bad visibility, such as fog. The `flights` dataframe contains a column `dep_delay` that is the number of minutes late each flight was in taking off from its New York airport, and also a column `time_hour` which is the scheduled departure day and hour of that flight. There is also a dataframe `weather` that contains weather information about each airport for each hour of 2013, including a column `visib` that shows the number of miles of visibility from that airport at that time on that date.

Obtain and save a dataframe that contains, for each flight, its departure delay, and the number of miles of visibility at its scheduled departure time at the airport that it departed from.

Solution:

This is rather a lot to think about, but the weather will have to be looked up for each flight.

The key[3] column is the one called `time_hour` in `flights`. What does `weather` contain?

```
weather
```

```
## # A tibble: 26,115 x 15
##    origin  year month   day  hour  temp  dewp humid wind_dir wind_speed
##    <chr>  <int> <int> <int> <int> <dbl> <dbl> <dbl>    <dbl>      <dbl>
##  1 EWR     2013     1     1     1  39.0  26.1  59.4      270      10.4
##  2 EWR     2013     1     1     2  39.0  27.0  61.6      250       8.06
##  3 EWR     2013     1     1     3  39.0  28.0  64.4      240      11.5
##  4 EWR     2013     1     1     4  39.9  28.0  62.2      250      12.7
##  5 EWR     2013     1     1     5  39.0  28.0  64.4      260      12.7
##  6 EWR     2013     1     1     6  37.9  28.0  67.2      240      11.5
##  7 EWR     2013     1     1     7  39.0  28.0  64.4      240      15.0
##  8 EWR     2013     1     1     8  39.9  28.0  62.2      250      10.4
##  9 EWR     2013     1     1     9  39.9  28.0  62.2      260      15.0
## 10 EWR     2013     1     1    10  41    28.0  59.6      260      13.8
## # ... with 26,105 more rows, and 5 more variables: wind_gust <dbl>,
## #   precip <dbl>, pressure <dbl>, visib <dbl>, time_hour <dttm>
```

A column of the same name right at the end. That makes our job a bit easier. We want the weather at the same airport and same time as the flight departed from, so the `by` has to contain both (they have the same names in the two dataframes). If you get over a million rows, it's probably because you forgot to match the airport (and thus you'd have three rows in the output, one for each airport):

```
flights %>%
  left_join(weather, by = c("origin", "time_hour")) %>%
  select(dep_delay, visib) %>%
  drop_na() -> d
d
```

```
## # A tibble: 326,993 x 2
##    dep_delay visib
##        <dbl> <dbl>
##  1         2    10
##  2         4    10
##  3         2    10
##  4        -1    10
##  5        -6    10
##  6        -4    10
##  7        -5    10
##  8        -3    10
##  9        -3    10
## 10        -2    10
## # ... with 326,983 more rows
```

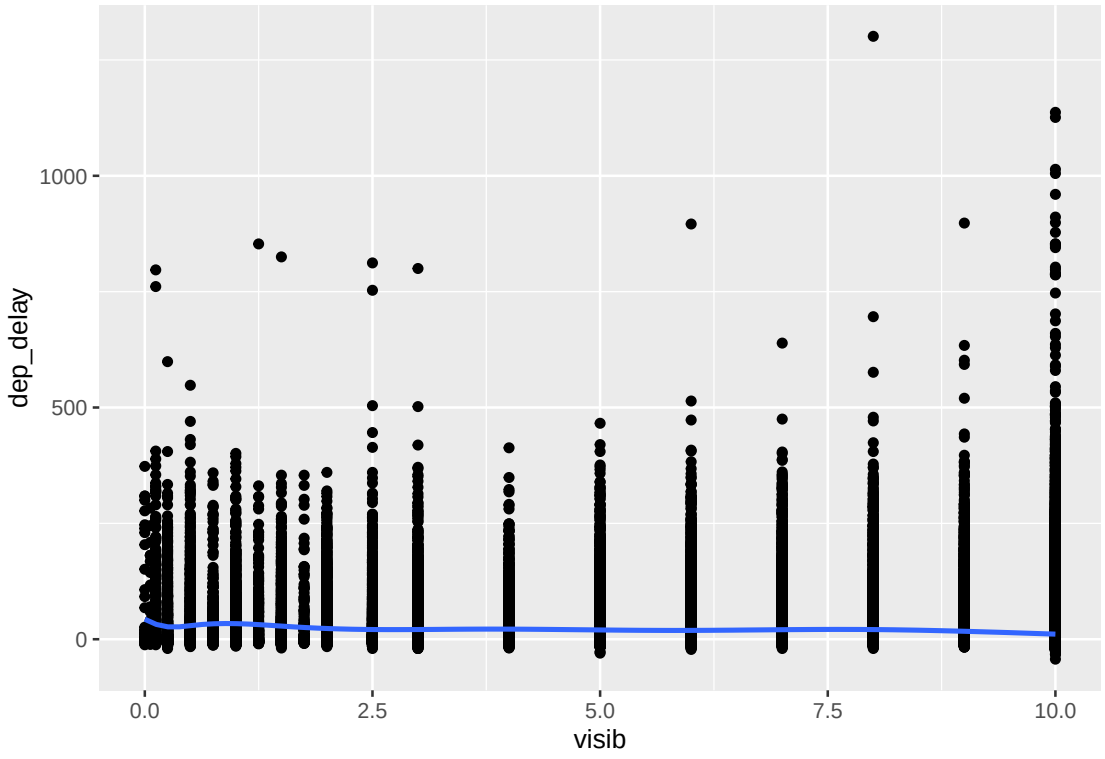
(e) Using your dataframe from the previous part, make a suitable plot of the number of minutes of delay on departure against visibility. Is there any indication that delays are typically worse when the visibility is worse? You may need to use some imagination to come up with a plot that shows what you want to see.

Solution:

Two quantitative variables, so a scatterplot. I would put a smooth trend on it, to make it easier to see whether there is a relationship. There are a *lot* of observations, so the plot takes an appreciable time to draw:

```
ggplot(d, aes(x = visib, y = dep_delay)) + geom_point() + geom_smooth(se = FALSE)
```

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

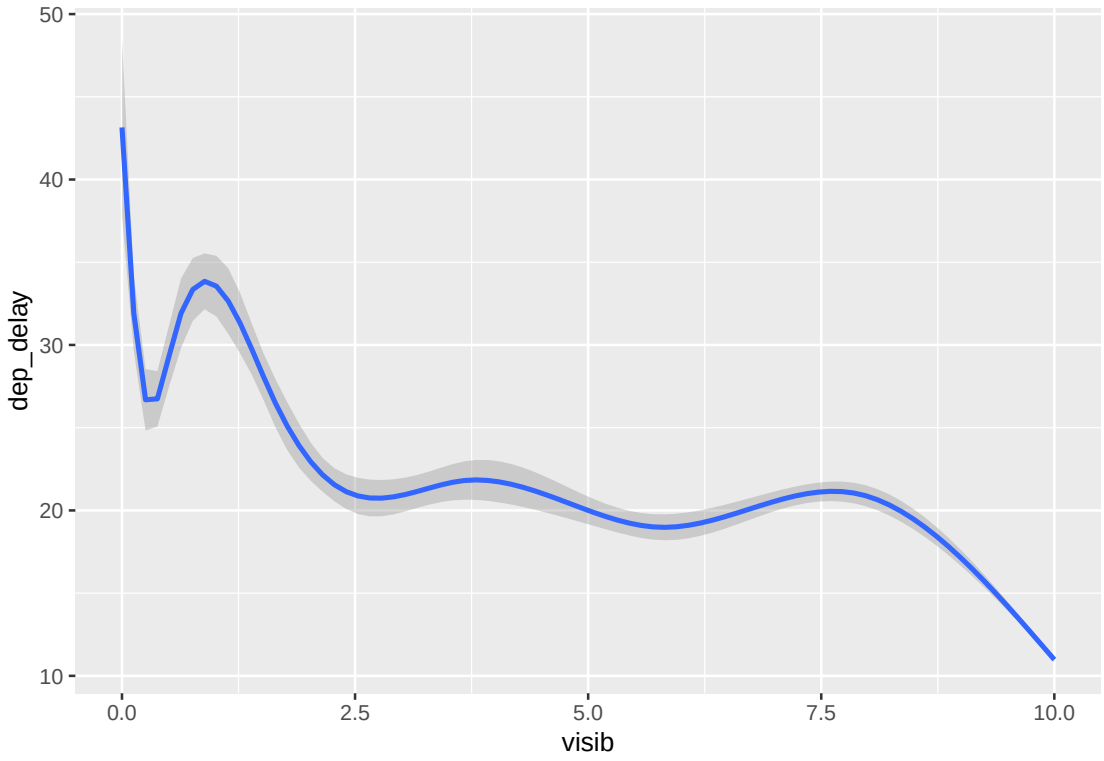


The big takeaway here is that the delay can be very large for any visibility, and this makes it hard to see whether the average delay has any trend with visibility. I think it's a tiny bit higher for low (that is, bad) visibility, but it really isn't clear. 500 minutes of delay is over 8 hours, which is a long time for a flight to be late leaving. This is therefore not the best plot, because it is dominated by the (huge number of) outliers.

So let's try removing the points and just leaving the smooth trend, which is down at the bottom of this plot. That might be clearer:

```
ggplot(d, aes(x = visib, y = dep_delay)) + geom_smooth()
```

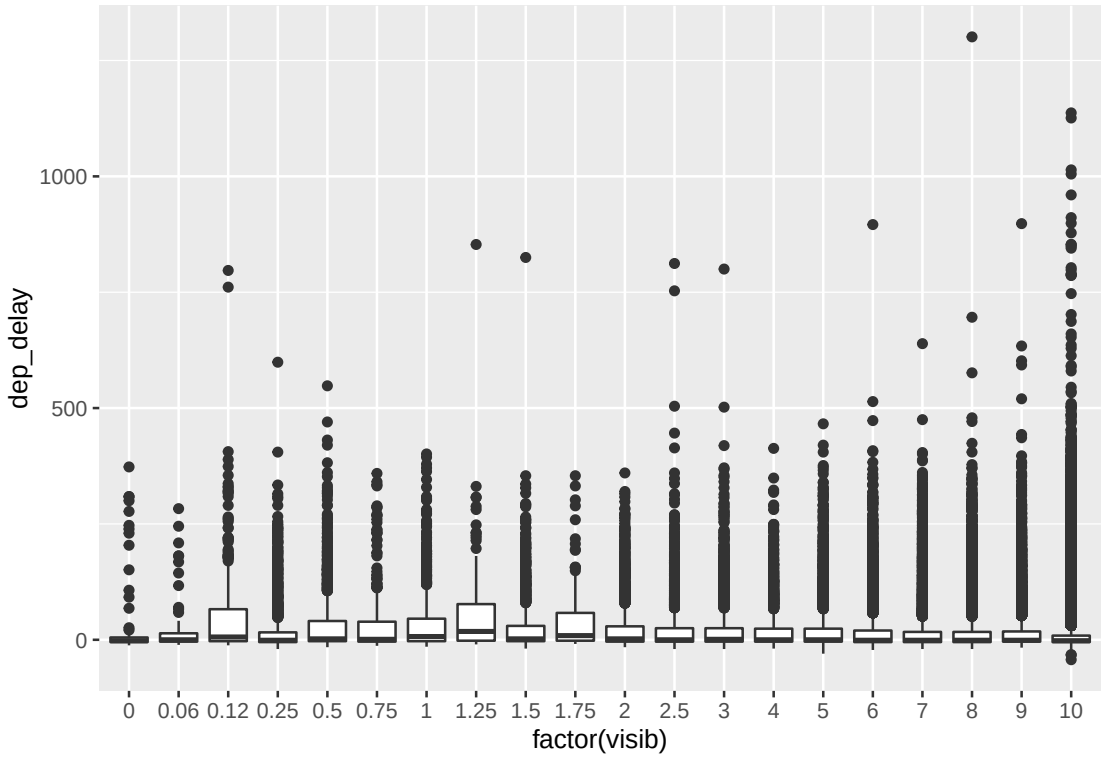
```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



This is a lot clearer: once the visibility gets down below about 2.5 miles, the average delay increases. I'm not sure about that wiggle around 0.5 miles, but the grey envelope suggests that the smooth trend is an accurate picture, based as it is on so many observations. You might also note that the average delay is smaller when the visibility is at its maximum of 10 miles.

Another way to try is to treat `visib` as categorical, which it really isn't, but there is a fairly small set of discrete values. If you do that, you have a quantitative delay and a categorical visibility, which you have to make such, and thus a boxplot:

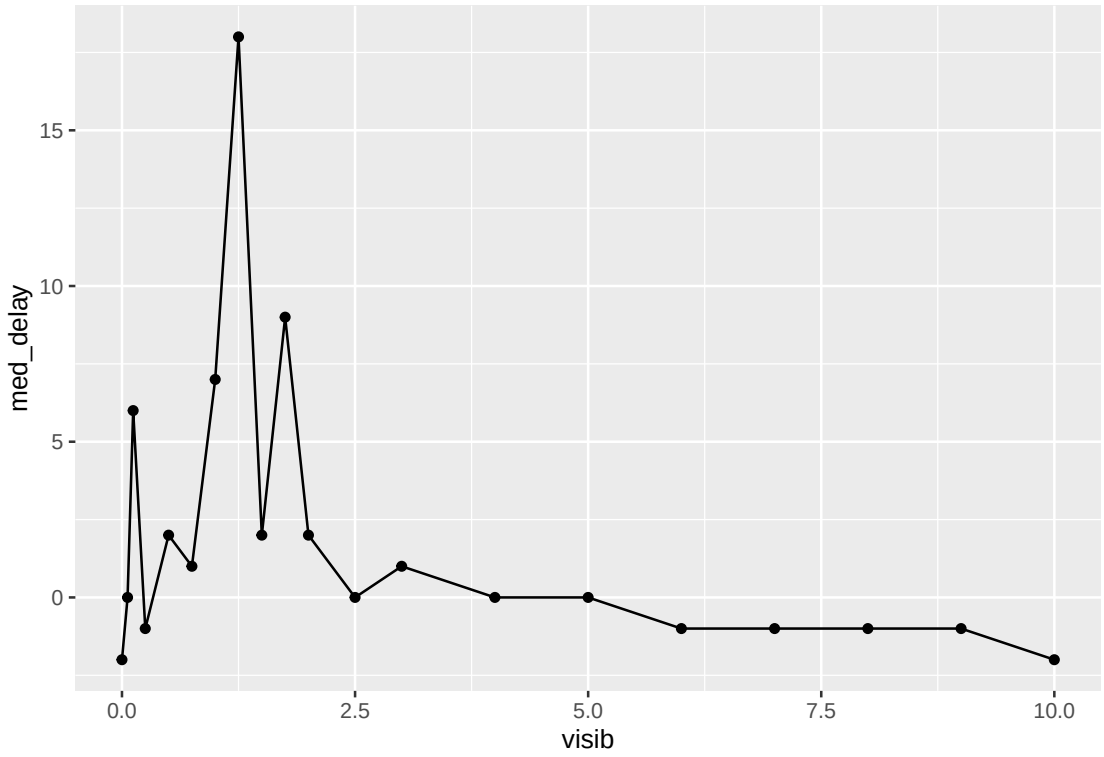
```
ggplot(d, aes(x = factor(visib), y = dep_delay)) + geom_boxplot()
```



This is also not the best, because it is dominated by the outliers.

You could just plotting the median for each `visib` value, thus:

```
d %>%
  group_by(visib) %>%
  summarize(med_delay = median(dep_delay)) %>%
  ggplot(aes(x = visib, y = med_delay)) +
    geom_point() + geom_line()
```



Not all the median delays are large when visibility is under 2.5 miles, but at least some of them are, and the implication is that you could be delayed a long time if the visibility is bad. This also shows that the median delay drops when the visibility is best. The median delay is actually *negative* when the visibility is good, which means that the flights on average left *early*, which they can do if all the booked passengers, their baggage and the crew are on board the plane.

What about if we plot the median *and* quartiles for each visibility value? I don't know how to do that yet, but I know how to *calculate* them, so let's do that first:[4]

```
d %>%
  group_by(visib) %>%
  summarize(q1 = quantile(dep_delay, 0.25),
            med = median(dep_delay),
            q3 = quantile(dep_delay, 0.75)) -> d_summ
d_summ
```

```
## # A tibble: 20 x 4
##    visib    q1   med    q3
##    <dbl> <dbl> <dbl> <dbl>
##  1  0       -5    -2   5
##  2  0.06    -4     0  14
##  3  0.12    -3     6  66
##  4  0.25    -5    -1  16
##  5  0.5     -3     2  40.5
##  6  0.75    -4     1  39
##  7  1       -3     7  45.5
##  8  1.25    -2    18  77
##  9  1.5     -3     2  30
```

```
## 10  1.75    -2     9  58
## 11  2       -4     2  29
## 12  2.5     -4     0  25
## 13  3       -4     1  25
## 14  4       -4     0  24
## 15  5       -4     0  24
## 16  6       -5    -1  20
## 17  7       -5    -1  17
## 18  8       -5    -1  17
## 19  9       -4    -1  18
## 20 10       -5    -2   9
```

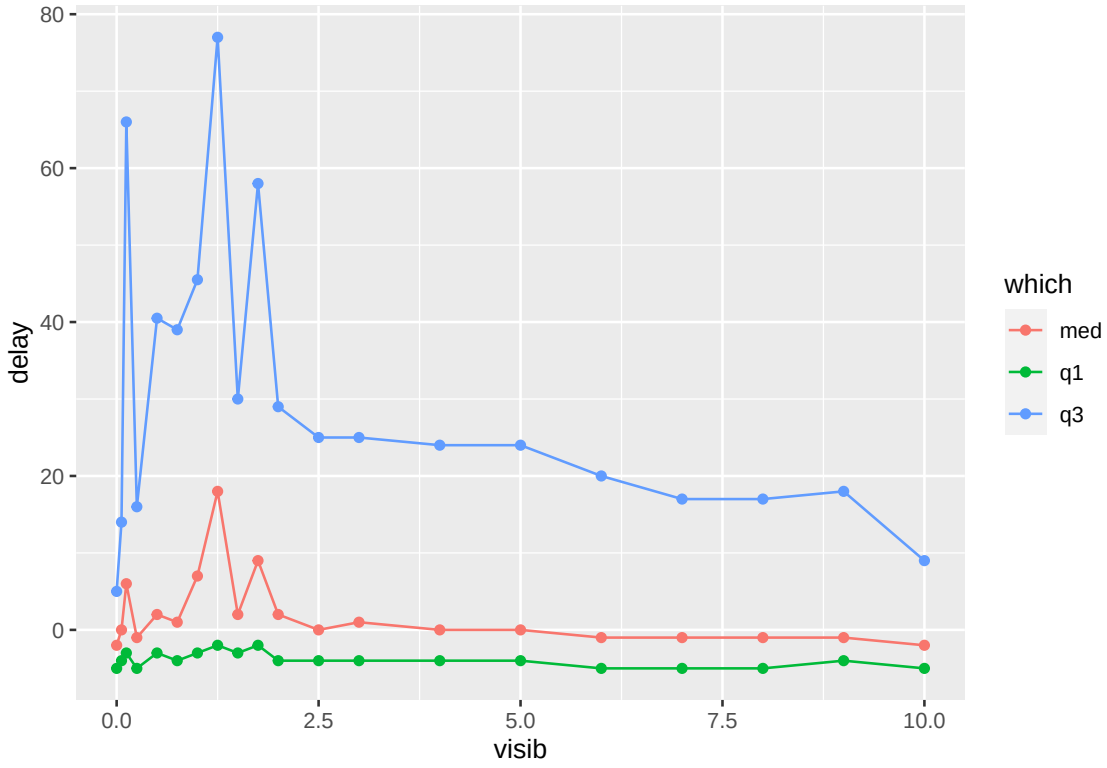
Well, this I can do now.[5] Let's make this longer, since we want *one* column of delay summaries to plot:

```
d_summ %>%
  pivot_longer(q1:q3, names_to = "which", values_to = "delay")
```

```
## # A tibble: 60 x 3
##    visib which delay
##    <dbl> <chr> <dbl>
##  1  0    q1       -5
##  2  0    med      -2
##  3  0    q3        5
##  4  0.06 q1       -4
##  5  0.06 med       0
##  6  0.06 q3       14
##  7  0.12 q1       -3
##  8  0.12 med       6
##  9  0.12 q3       66
## 10  0.25 q1       -5
## # ... with 50 more rows
```

then plot these as points, joining up the ones that are the same quantile:

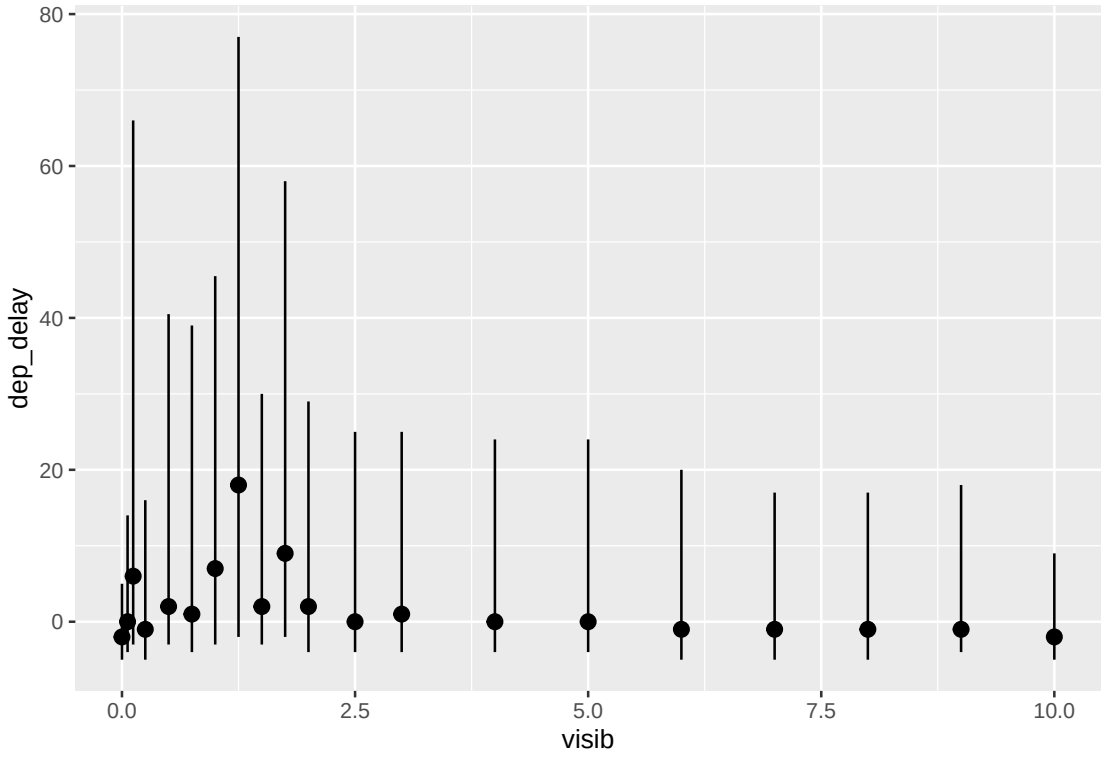
```
d_summ %>%
  pivot_longer(q1:q3, names_to = "which", values_to = "delay") %>%
  ggplot(aes(x = visib, y = delay, colour = which)) + geom_point() + geom_line()
```



My take from this is that delays can be bigger when the visibility is less than about 2.5 miles. It's a bit harder to interpret because it is jumpy, even with so much data.

Extra 1: another way of making a plot like these is to have `ggplot` do the calculations for you. We haven't talked about writing functions yet, so some of this won't make much sense to you, but (this way)[https://stackoverflow.com/questions/41077199/how-to-use-r-ggplot-stat-summary-to-plot-median-and-quartiles/41077282] also works, though it too takes noticeable time:

```
ggplot(d, aes(x = visib, y = dep_delay)) + stat_summary(
  fun.min = function(z) { quantile(z, 0.25) },
  fun.max = function(z) { quantile(z, 0.75) },
  fun = median)
```



The dots are the medians, and the lines extend from the median to the quartiles for each visibility value. Once again, the impression is that longer delays are possible when the visibility is worse, and it is unusual to have a long delay once the visibility gets beyond about 2.5 miles.

The code idea is that `stat_summary` contains instructions for calculating the summaries you are going to plot. The input `fun` is a function that calculates where the points are going to go. In this case, that is the median. `fun.min` and `fun.max` are the bottom and top of the vertical lines, which here need to be the quartiles. The `quantile` function needs an extra input, saying which quantile (percentile) is wanted, and one way around this is to write a little (nameless) function that is input to `fun.min` or `fun.max`, taking input `z` (the name doesn't matter as long as you use the chosen name in the function) and calculating what is needed from that `z`.

Extra 2: if you calculate the three percentiles in one go, the process to the plot is different, thus:

```
qs <- c(0.25, 0.5, 0.75)
d %>%
  group_by(visib) %>%
  summarize(q_delay =  quantile(dep_delay, qs))
```

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

```
## # A tibble: 60 x 2
## # Groups:   visib [20]
##    visib q_delay
##    <dbl>   <dbl>
##  1  0         -5
##  2  0         -2
##  3  0          5
```

```
##  4  0.06      -4
##  5  0.06       0
##  6  0.06      14
##  7  0.12      -3
##  8  0.12       6
##  9  0.12      66
## 10  0.25      -5
## # ... with 50 more rows
```

This has a row for each summary, but unfortunately there is no indication of which is which, so we need to put them in ourselves:[6]

```
qs <- c(0.25, 0.5, 0.75)
d %>%
  group_by(visib) %>%
  summarize(q_delay =  quantile(dep_delay, qs)) %>%
  mutate(which = rep(qs))
```

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

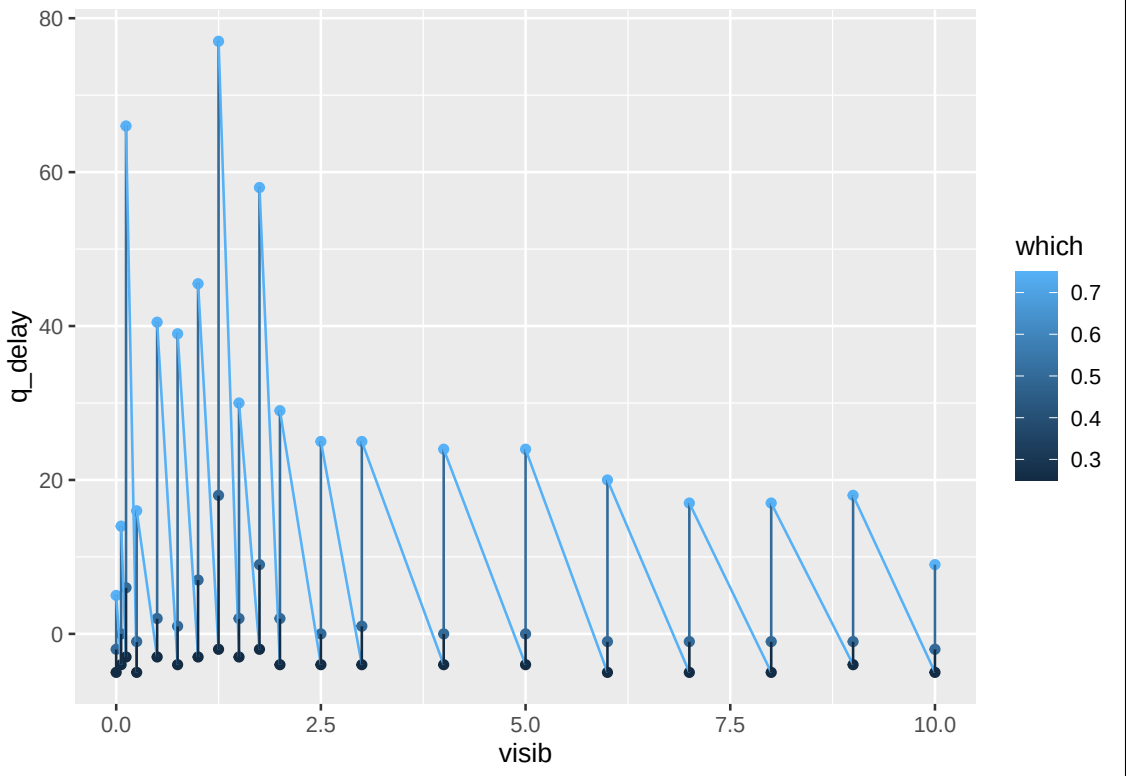
```
## # A tibble: 60 x 3
## # Groups:   visib [20]
##    visib q_delay which
##    <dbl>   <dbl> <dbl>
##  1  0         -5  0.25
##  2  0         -2  0.5
##  3  0          5  0.75
##  4  0.06      -4  0.25
##  5  0.06       0  0.5
##  6  0.06      14  0.75
##  7  0.12      -3  0.25
##  8  0.12       6  0.5
##  9  0.12      66  0.75
## 10  0.25      -5  0.25
## # ... with 50 more rows
```

`rep` repeats its input a number of times that you normally have to specify, but here it works out that the new column `which` has to be the same length as the other columns (because we're working with a dataframe). So we are good.

Now everything is ready to go:

```
qs <- c(0.25, 0.5, 0.75)
d %>%
  group_by(visib) %>%
  summarize(q_delay =  quantile(dep_delay, qs)) %>%
  mutate(which = rep(qs)) %>%
  ggplot(aes(x = visib, y = q_delay, colour = which)) +
    geom_point() + geom_line()
```

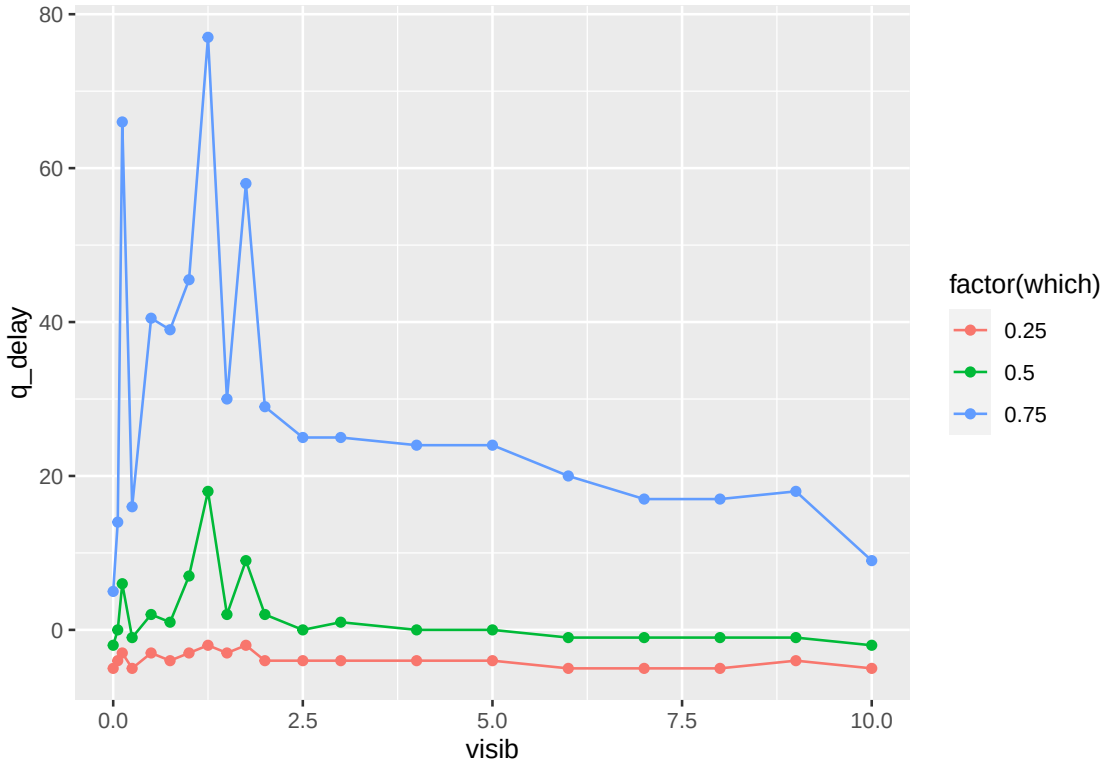
```
## `summarise()` has grouped output by 'visib'. You can override using the `.groups` argument.
```



Oops. The column `which` is quantitative, but I want it treated as categorical, so I should wrap it in `factor`:

```
qs <- c(0.25, 0.5, 0.75)
d %>%
  group_by(visib) %>%
  summarize(q_delay =  quantile(dep_delay, qs)) %>%
  mutate(which = rep(qs)) %>%
  ggplot(aes(x = visib, y = q_delay, colour = factor(which))) +
    geom_point() + geom_line()
```

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



as we had before.

Extra 3: an attempt to be clever.[7]

`quantile` does actually keep track of which quantile is which (using the same `qs` as above):

```
quantile(1:10, qs)
```

```
##  25%  50%  75%
## 3.25 5.50 7.75
```

except that the percentages are what is called the "names attribute" of the output, and the `tidyverse` tends to lose things like names attributes, *unless* you explicitly keep hold of them, like this:

```
enframe(quantile(1:10, qs), name = "percent", value = "percentile")
```

```
## # A tibble: 3 x 2
##   percent percentile
##   <chr>        <dbl>
## 1 25%           3.25
## 2 50%           5.5
## 3 75%           7.75
```

So now, what if we add the `enframe` to the calculation we had above?

```
d %>%
  group_by(visib) %>%
  summarize(q_delay =  enframe(quantile(dep_delay, qs), name = "percent", value = "delay"))
```

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

```
## # A tibble: 60 x 2
## # Groups:   visib [20]
##    visib q_delay$percent $delay
##    <dbl> <chr>            <dbl>
##  1  0    25%                 -5
##  2  0    50%                 -2
##  3  0    75%                  5
##  4  0.06 25%                 -4
##  5  0.06 50%                  0
##  6  0.06 75%                 14
##  7  0.12 25%                 -3
##  8  0.12 50%                  6
##  9  0.12 75%                 66
## 10  0.25 25%                 -5
## # ... with 50 more rows
```

We now have a list-column, which is OK, but each visibility value is repeated three times, which is not, so I don't like this. I think I need to explicitly add a `list` in the right place:

```
d %>%
  group_by(visib) %>%
  summarize(q_delay =  list(enframe(quantile(dep_delay, qs), name = "percent", value = "delay"
```

```
## # A tibble: 20 x 2
##    visib q_delay
##    <dbl> <list>
##  1  0    <tibble [3 x 2]>
##  2  0.06 <tibble [3 x 2]>
##  3  0.12 <tibble [3 x 2]>
##  4  0.25 <tibble [3 x 2]>
##  5  0.5  <tibble [3 x 2]>
##  6  0.75 <tibble [3 x 2]>
##  7  1    <tibble [3 x 2]>
##  8  1.25 <tibble [3 x 2]>
##  9  1.5  <tibble [3 x 2]>
## 10  1.75 <tibble [3 x 2]>
## 11  2    <tibble [3 x 2]>
## 12  2.5  <tibble [3 x 2]>
## 13  3    <tibble [3 x 2]>
## 14  4    <tibble [3 x 2]>
## 15  5    <tibble [3 x 2]>
## 16  6    <tibble [3 x 2]>
## 17  7    <tibble [3 x 2]>
## 18  8    <tibble [3 x 2]>
## 19  9    <tibble [3 x 2]>
## 20 10    <tibble [3 x 2]>
```

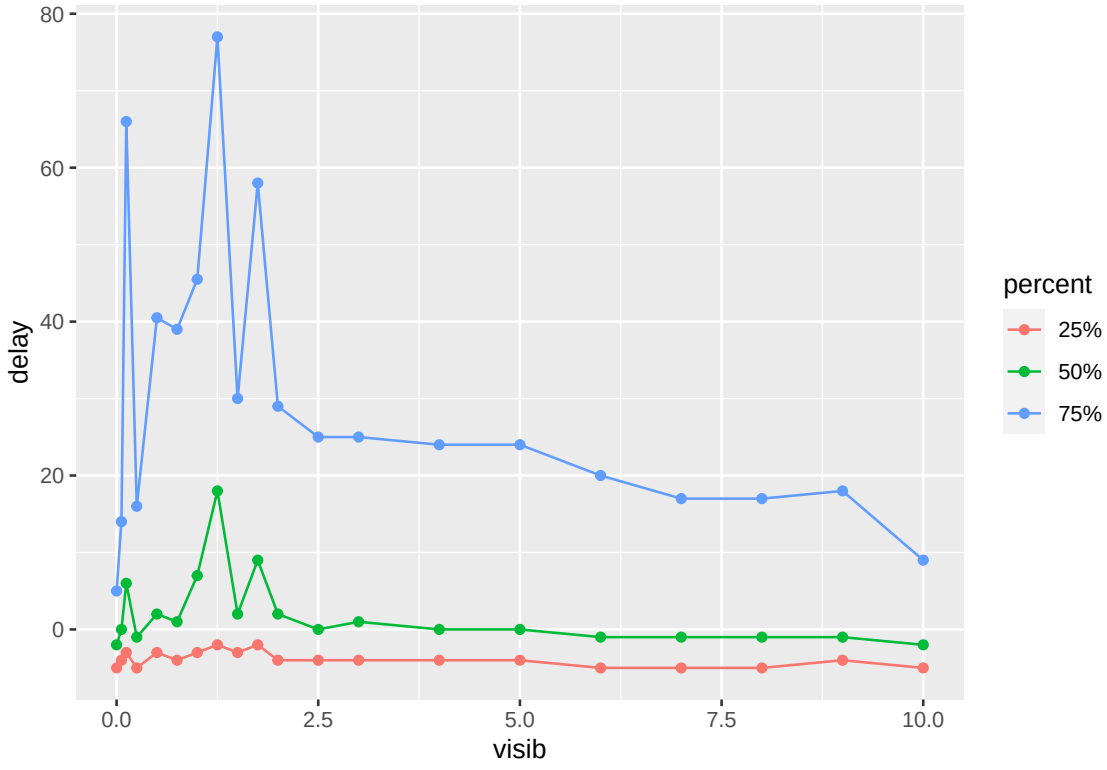
Now each `visib` value appears once, and in the list-column `q_delay` there is some kind of dataframe for each `visib`, which we hope is the same kind of thing as `enframe` gave us before. Let's `unnest` and take a look:

```
d %>%
  group_by(visib) %>%
  summarize(q_delay =  list(enframe(quantile(dep_delay, qs), name = "percent", value = "delay"
  unnest(q_delay)
```

```
## # A tibble: 60 x 3
##    visib percent delay
##    <dbl> <chr>   <dbl>
##  1  0    25%        -5
##  2  0    50%        -2
##  3  0    75%         5
##  4  0.06 25%        -4
##  5  0.06 50%         0
##  6  0.06 75%        14
##  7  0.12 25%        -3
##  8  0.12 50%         6
##  9  0.12 75%        66
## 10  0.25 25%        -5
## # ... with 50 more rows
```

Genius! Or, as my daughter says, "pog":

```
d %>%
  group_by(visib) %>%
  summarize(q_delay =  list(enframe(quantile(dep_delay, qs), name = "percent", value = "delay"
  unnest(q_delay) %>%
  ggplot(aes(x = visib, y = delay, colour = percent)) + geom_point() + geom_line()
```



Notes

1. The abbreviation comes from “O’Hare Field”, which is the original name of Chicago’s airport,

2. This is actually a regular expression match.

3. A joke. The most important column, and also the one the database lookup is going to be based on.

4. The *factor* thing is not needed here, because grouping will use each of the distinct values even if the variable is really quantitative.

5. If I remember to put quotes around the names of my new columns, that is.

6. Or be cleverer than I am capable of.

7. And thus most likely doomed to failure.