Notebook for the MOOC Reproducible Research, Module 3, Peer-reviewed Execirce, Subject 4

Latency and capacity estimation for a network connection from asymmetric measurements

Killian Martin

5 mai 2020

Preamble

Following Subject 4 in the MOOC page, I attempt to estimate the performance of a network connection based on data regarding the time T (in seconds) required to send a message as a function of its size S (in bytes). More specifically, I will fit a simple linear model for the relation between T and S is

$$T(S) = L + \frac{S}{C}$$

where:

- L is the connection's *latency*, in seconds
- C is the connection's *capacity*, in bytes/seconds

The task given is to estimate L and C based on data acquired on T as a function of S, using the command `ping`. Each data point is originally a single row of the dataset (see below in the Data Input section for examples). The data for this exercise comes from A. Legrand website. The exact download links are in the Data Input section below, within the first code block of the Data sources sub-section.

Setting up the environment

Packages

I first load all the packages I will use throughout the document:

```
library(tidyverse)
library(magrittr)
library(broom)
```

The tidyverse is used for data manipulation with `dplyr` and `purrr` and plotting with `ggplot2`. `broom`, `ggridges` and `magrittr` are mostly for single-use conveninence, introducing, respectively, concise model overview, ridge plots in `ggplot2`, and the `$%$` operator (which lets me use column names directly in functions, instead of using the `$` notation).

Working directory

No code necessary as the document by default looks into its own directory for files (this is also something I had not realised before, so I'm writing it down).

Details of the compiling environment

This notebook was computed in the following environment:

```
sessionInfo()
```

```
## R version 4.0.0 (2020-04-24)
## Platform: x86_64-apple-darwin17.0 (64-bit)
## Running under: macOS Mojave 10.14.6
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRblas.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] broom_0.5.6     magrittr_1.5    forcats_0.5.0   stringr_1.4.0
##  [5] dplyr_0.8.5     purrr_0.3.4     readr_1.3.1     tidyr_1.0.2
##  [9] tibble_3.0.1    ggplot2_3.3.0   tidyverse_1.3.0
##
## loaded via a namespace (and not attached):
##  [1] tidyselect_1.0.0 xfun_0.13        haven_2.2.0      lattice_0.20-41
##  [5] colorspace_1.4-1 vctrs_0.2.4      generics_0.0.2   htmltools_0.4.0
##  [9] yaml_2.2.1       rlang_0.4.6      pillar_1.4.4     glue_1.4.0
## [13] withr_2.2.0      DBI_1.1.0        dbplyr_1.4.3     modelr_0.1.7
## [17] readxl_1.3.1     lifecycle_0.2.0  munsell_0.5.0    gtable_0.3.0
## [21] cellranger_1.1.0 rvest_0.3.5      evaluate_0.14    knitr_1.28
## [25] fansi_0.4.1      Rcpp_1.0.4.6     scales_1.1.0     backports_1.1.6
## [29] jsonlite_1.6.1   fs_1.4.1         hms_0.5.3        digest_0.6.25
## [33] stringi_1.4.6    grid_4.0.0       cli_2.0.2        tools_4.0.0
## [37] crayon_1.3.4     pkgconfig_2.0.3  ellipsis_0.3.0   xml2_1.3.2
## [41] reprex_0.3.0     lubridate_1.7.8  assertthat_0.2.1 rmarkdown_2.1
## [45] httr_1.4.1       rstudioapi_0.11  R6_2.4.1         nlme_3.1-147
## [49] compiler_4.0.0
```

Data input

Datasets used in the task

Two different datasets are provided for the task, `liglab2` and `stackoverflow`. Each dataset contains the data outlined in the preamble. `liglab2` contains data for a short-range on-campus connection, while `stackoverflow` contains data for a long-range Web-based connection. Both datasets are hosted online at the URL stored below.

NOTE: to get the script to work, I removed the .gz extensions that are present in the main MOOC page. Otherwise I get an error message when trying to read the URL. This does not matter if files are already present locally though.

Loading the data

Import function

For ease of use, the function `download` below handles everything. The user merely needs to provide the URL and the file name to use to save the data to a file using the `url` and `filename` arguments. The remaining arguments are optional and are for convenience: `path` lets the user specify a directory path to use instead of the default path, `overwrite` allows overwriting the data even if a file already exists at the saving location (set to TRUE to overwrite, keep FALSE to avoid overwriting), and `encoding` is used to specify the encoding (given as a character string, e.g. "UTF-8" for UTF-8 encoding). The function reads the data at the provided URL using the base R `read.delim` function, then writes it to the desired location with `write.table`.

```r
download <- function(url, filename, path = NULL, overwrite = F, ...){
  if (!is.null(path)){
    full_name = file.path(path, filename)
  } else {
    full_name = filename
  }

  if (!file.exists(full_name) || overwrite){
    tab <- read.delim(file = url, header = FALSE, ...)
    write.table(x = tab, file = full_name)
  } else {
    print(sprintf("File %s already found at provided location and overwrite = FALSE.", full_name))
    print("If you wish to overwrite the data anyway, set overwrite to TRUE")
  }
}
```

Downloading the data from the hosting webpages

I can then apply `download` to both of the URLs:

```r
# set overwrite to T if updating the logs is desired
download(url = liglab2_url, filename = "liglab2.log", overwrite = F)
```

```
## [1] "File liglab2.log already found at provided location and overwrite = FALSE."
## [1] "If you wish to overwrite the data anyway, set overwrite to TRUE"
```

```r
download(url = stackoverflow_url, filename = "stackoverflow.log", overwrite = F)
```

```
## [1] "File stackoverflow.log already found at provided location and overwrite = FALSE."
## [1] "If you wish to overwrite the data anyway, set overwrite to TRUE"
```

Importing the data

I finally import the datasets from the local files (whether or not I re-downloaded above), using the `read.table` function. To keep character strings as character strings, I specify the `stringsAsFactors` argument to `FALSE`.

```r
liglab2_original <- read.table(file = "liglab2.log", stringsAsFactors = FALSE)
stackoverflow_original <- read.table(file = "stackoverflow.log", stringsAsFactors = FALSE)

head(liglab2_original)
```

```
##                                                                                                  V
## 1  [1421761682.052172] 665 bytes from lig-publig.imag.fr (129.88.11.7): icmp_seq=1 ttl=60 time=22.5 m
## 2 [1421761682.277315] 1373 bytes from lig-publig.imag.fr (129.88.11.7): icmp_seq=1 ttl=60 time=21.2 m
## 3  [1421761682.502054] 262 bytes from lig-publig.imag.fr (129.88.11.7): icmp_seq=1 ttl=60 time=21.2 m
## 4 [1421761682.729257] 1107 bytes from lig-publig.imag.fr (129.88.11.7): icmp_seq=1 ttl=60 time=23.3 m
```

```
## 5 [1421761682.934648] 1128 bytes from lig-publig.imag.fr (129.88.11.7): icmp_seq=1 ttl=60 time=1.41 r
## 6  [1421761683.160397] 489 bytes from lig-publig.imag.fr (129.88.11.7): icmp_seq=1 ttl=60 time=21.9 r
```

```
head(stackoverflow_original)
```

```
##
## 1 [1421771203.082701] 1257 bytes from stackoverflow.com (198.252.206.140): icmp_seq=1 ttl=50 time=120
## 2  [1421771203.408254] 454 bytes from stackoverflow.com (198.252.206.140): icmp_seq=1 ttl=50 time=120
## 3  [1421771203.739730] 775 bytes from stackoverflow.com (198.252.206.140): icmp_seq=1 ttl=50 time=126
## 4 [1421771204.056630] 1334 bytes from stackoverflow.com (198.252.206.140): icmp_seq=1 ttl=50 time=112
## 5   [1421771204.372224] 83 bytes from stackoverflow.com (198.252.206.140): icmp_seq=1 ttl=50 time=111
## 6  [1421771204.688367] 694 bytes from stackoverflow.com (198.252.206.140): icmp_seq=1 ttl=50 time=111
```

Formatting the datasets

We can then parse the data to extract the information of interest, that is - the moment of measurement `M`, between brackets at the beginning of each row, in seconds since January 1st, 1970; - the size in bytes `S` of the message sent ("XXX bytes"); - the time to send the message `T` (time=XXX ms") at the end.

```
reformat <- function(tab){
  transmute(tab,
            M = str_extract(string = V1, pattern = "(?<=\\[).+(?=\\])"),
            # here the pattern is the middle part not between of interest, while the other parts frame
            S = ifelse(str_detect(string = V1, pattern = "bytes"),
                       str_extract(string = V1, pattern = "[0-9]+ bytes"),
                       NA) %>%
              str_remove(pattern = " bytes"), # keep only the number
            T =  ifelse(str_detect(string = V1, pattern = "time="),
                        str_extract(string = V1, pattern = "time=[:alnum:]+"),
                        NA) %>%
              str_remove(pattern = "time=")) %>%   # keep only the time
  type.convert() # convert from character strings to numeric doubles
  # coulv have used type_convert for more general purposes but in this case it removes numbers after th
}
```

```
liglab2_original <- reformat(liglab2_original)
stackoverflow_original <- reformat(stackoverflow_original)

head(liglab2_original)
```

```
##            M    S  T
## 1 1421761682  665 22
## 2 1421761682 1373 21
## 3 1421761683  262 21
## 4 1421761683 1107 23
## 5 1421761683 1128  1
## 6 1421761683  489 21
```

```
head(stackoverflow_original)
```

```
##            M    S   T
## 1 1421771203 1257 120
## 2 1421771203  454 120
## 3 1421771204  775 126
## 4 1421771204 1334 112
## 5 1421771204   83 111
## 6 1421771205  694 111
```

```
print(length(unique(liglab2_original$M)) == nrow(liglab2_original) ) # make sure no unique value is los
```

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

Do note that the display used rounds the number to their closest integer value, but the true values are maintained:

```
liglab2_original$M[1] - round(liglab2_original$M[1])
```

```
## [1] 0.05217195
```

Now that we have the data we are interested in (and in the correct formats, thanks to our function), we are ready to proceed. From now on,I separate the analysis into first the `liglab2` dataset, and then the `stackoverflow` dataset.

Analysis on the `liglab2` dataset

Missing data

First I check the proportion of NAs in the liglab2_formatted dataset:

```
NAs_liglab2 <- liglab2_original %>%
  filter(is.na(S) | is.na(T))

print(sprintf("Total number of rows with NA: %i", nrow(NAs_liglab2)))
```

```
## [1] "Total number of rows with NA: 377"
```

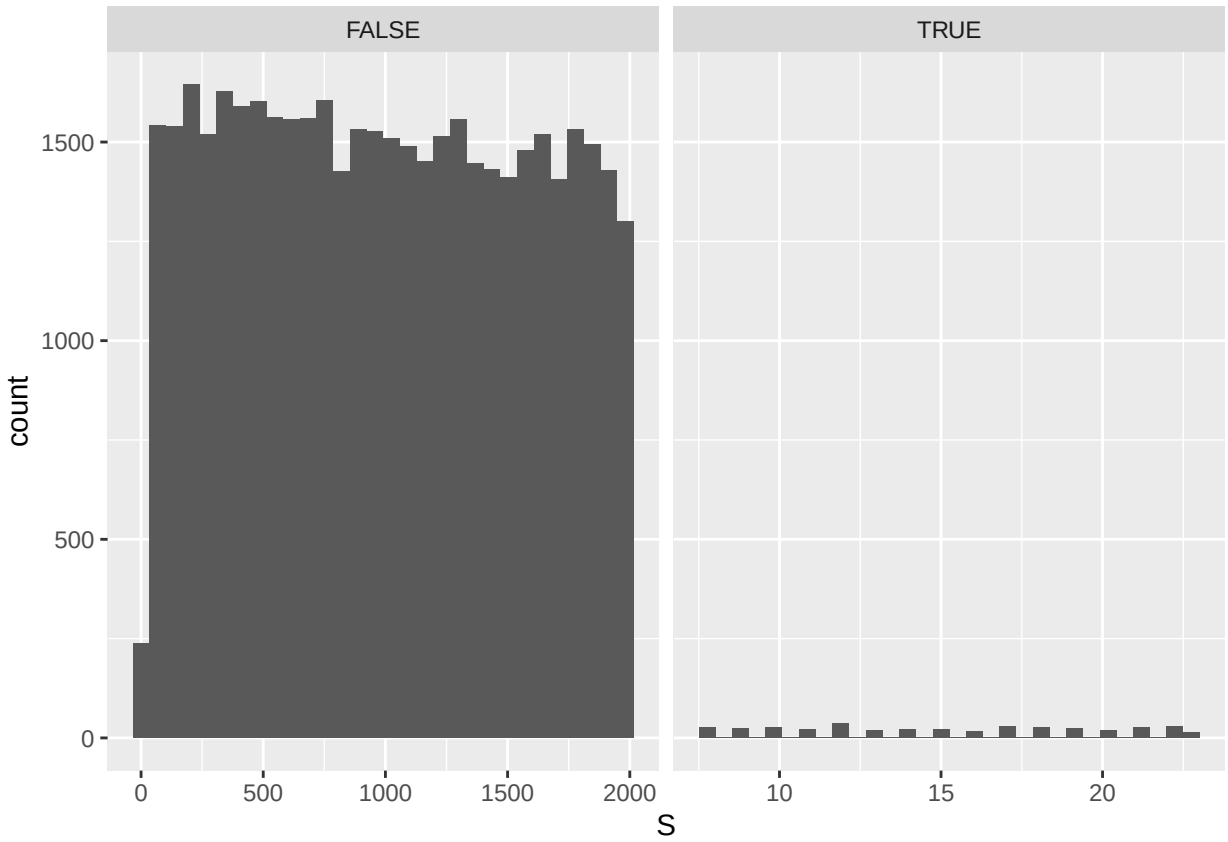
```
print(sprintf("Proportion of rows with NA: %f", nrow(NAs_liglab2)/nrow(liglab2_original) * 100))
```

```
## [1] "Proportion of rows with NA: 0.848851"
```

Given the proportion of less than 0.1% of rows containing NAs, I would feel quite comfortable simply removing the rows with NAs, however it could introduce bias if the missing values are not Missing At Random, which I can check graphically:

```
liglab2_original %>%
  ggplot(aes(x = S)) +
  geom_histogram() +
  facet_wrap(~is.na(T), scales = "free_x")
```

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



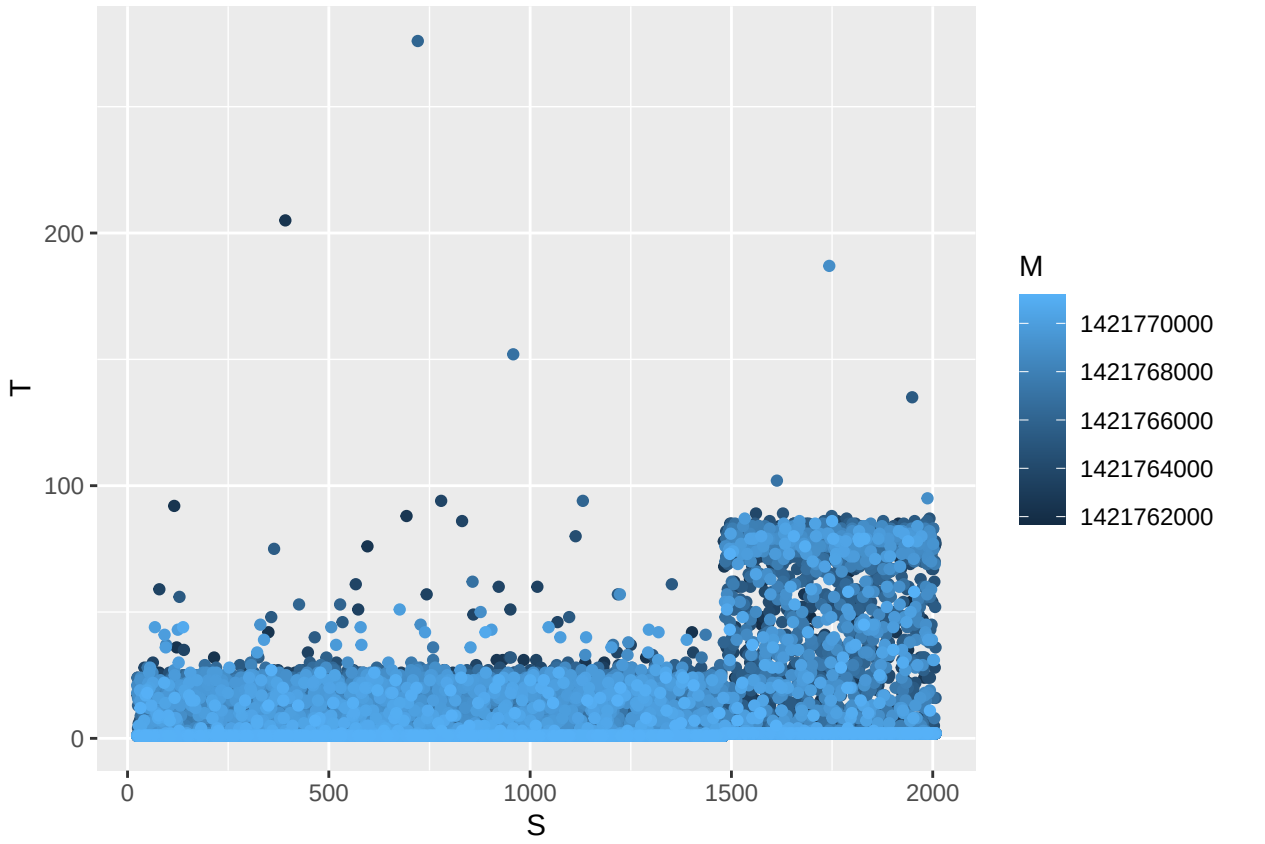
We can see that indeed, T seems to be missing only for very low values of S. We will therefore have to be a bit careful interpreting our results, but there is not much else we can actually do. However, given we can probably expect messages sent in a real use case to exceed 30 bytes most of the time, it should not be too much of a problem.

```
liglab2 <- liglab2_original %>%
  filter(!is.na(T)) # remove all missing values
```

Graphical exploration

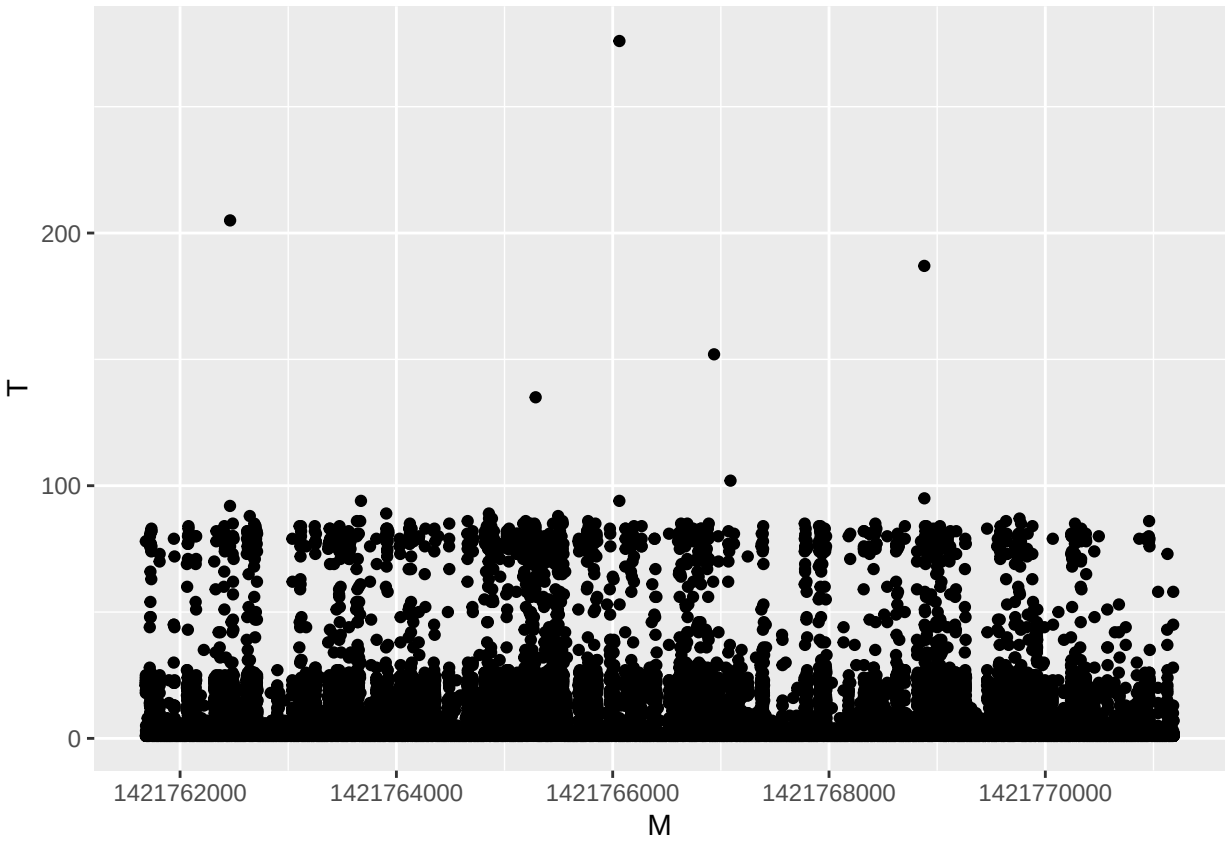
To get a better idea of our data, let's draw a scatterplot of the relation $T = f(S)$, using `ggplot2`. To get an idea of possible effects of time, I add M as the colour parameter for the points.

```
ggplot(liglab2, aes(x = S, y = T, colour = M)) +
  geom_point()
```



The relation looks highly non-linear, with a threshold of S (hereafter $S_{threshold}$) a bit under 1500 bytes. This threshold separates two regimes of T: one with low variance, where $S < S_{threshold}$, and one with high variance, where $S > S_{threshold}$. Moreover, adding the time of measurement M seems to have no effect on T. For a more direct visualisation of this, let's plot T directly as a function of M:

```
ggplot(liglab2, aes(x = M, y = T)) +
  geom_point()
```

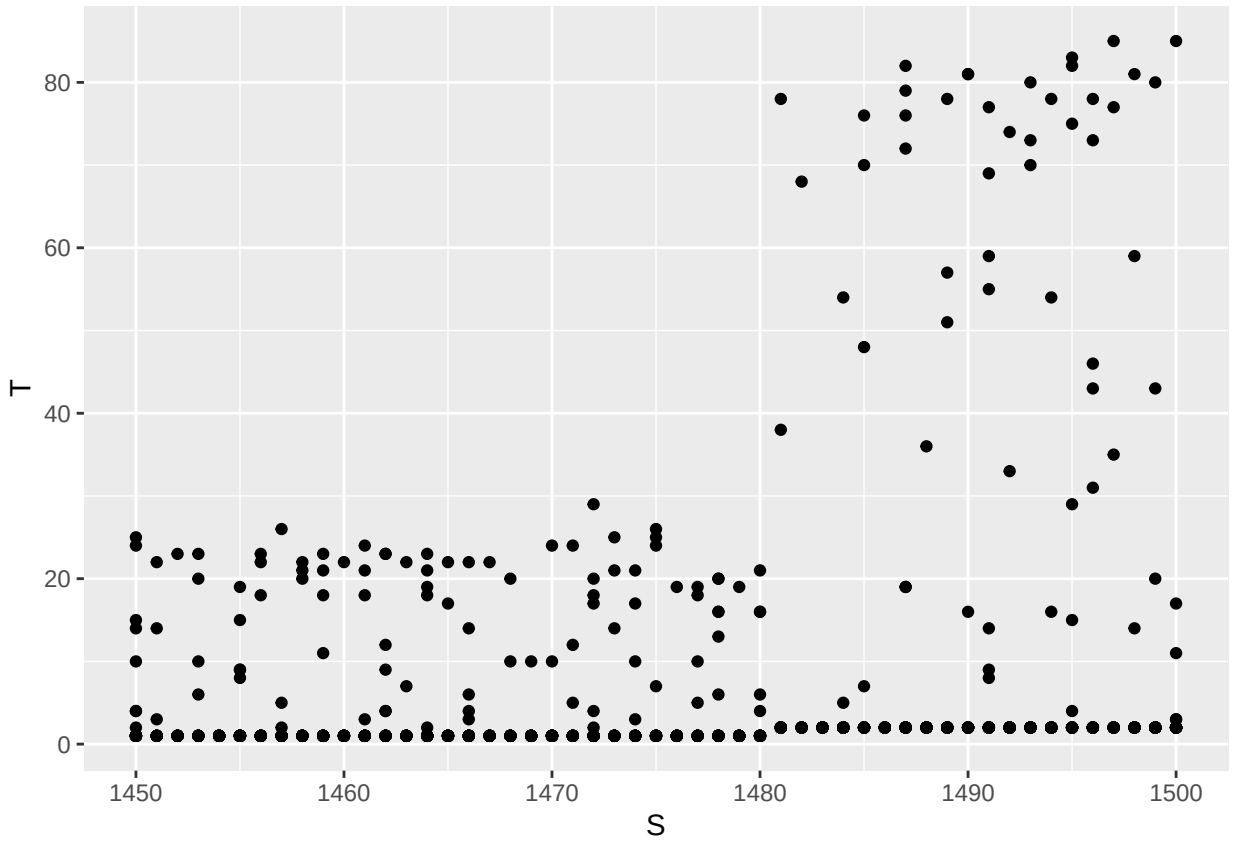


Indeed, there is no evident relation between T and M Therefore, we can focus on estimating the value of $S_{threshold}$

Graphical estimation of $S_{threshold}$

First I take a zoomed-in version of the graph above, to see if we can separate the two regimes by eye only:

```
liglab2 %>%
  filter(between(S, 1450, 1500)) %>%  # take only values of S between 1450 and 1500
  ggplot(aes(x = S, y = T)) +
    geom_point()
```



We can see here that the separation between regimes of S is between values of 1480 and 1481 bytes. We then set $S_{threshold} = 1480.5$ for simplicity (given only integers are possible values of S, as a half-byte object sounds difficult to encode, by definition), and store it in a variable for later:

```
S_threshold <- 1480.5
```

Estimation of L and C by regime of S via linear regression (`lm`)

As a reminder, we have the following linear relationship between T and S:

$$T(S) = L + \frac{S}{C}$$

We can separate the dataset by separating the values of S below and above 1480.5, which can do with `dplyr`:

```
liglab2 <- liglab2 %>%
  mutate(`S>S_threshold` = S > 1480.5) %>% # check if a given row has S above or below S_threshold
  group_by(`S>S_threshold`) %>%  # make two groups, corresponding to S being below or above S_threshold
  nest()
head(liglab2)
```

```
## # A tibble: 2 x 2
## # Groups:   S>S_threshold [2]
##   `S>S_threshold` data
##   <lgl>           <list>
## 1 FALSE           <tibble [32,667 x 3]>
## 2 TRUE            <tibble [11,369 x 3]>
```

Here the FALSE row corresponds to the sub-dataset including values of S below $S_{threshold}$, and the TRUE row to the one with values of S above $S_{threshold}$. Then we can fit two linear models (however, defined by the same equation) on those sub-datasets, again using `dplyr`:

```
liglab2 <- liglab2 %>%
  mutate(models = map(.x = data, .f = ~ lm(data = .x, T~S)))
liglab2
```

```
## # A tibble: 2 x 3
## # Groups:   S>S_threshold [2]
##   `S>S_threshold` data                  models
##   <lgl>           <list>                <list>
## 1 FALSE           <tibble [32,667 x 3]> <lm>
## 2 TRUE            <tibble [11,369 x 3]> <lm>
```

Finally, we can extract the model coefficients as well as some statistics on them using the `broom` package:

```
liglab2 %<>%
  mutate(model_stats = map(.x = models, .f = glance), # glance: gives a few model-wide statistics like
         model_coefs = map(.x = models, .f = tidy)) # tidy: gives the model coefficients, their standar
```

We can check these statistics:

```
liglab2 %>%
  select(c("S>S_threshold", "model_stats")) %>%
  unnest(c(model_stats)) # extract the model_stats column for inspection
```

```
## # A tibble: 2 x 12
## # Groups:   S>S_threshold [2]
##   `S>S_threshold` r.squared adj.r.squared sigma statistic p.value    df  logLik
##   <lgl>               <dbl>         <dbl> <dbl>     <dbl>   <dbl> <int>   <dbl>
## 1 FALSE           0.0000631     0.0000325  6.40      2.06  0.151      2 -1.07e5
## 2 TRUE            0.000303      0.000215  20.7       3.45  0.0634     2 -5.06e4
## # ... with 4 more variables: AIC <dbl>, BIC <dbl>, deviance <dbl>,
## #   df.residual <int>
```

We can see that the R^2 values of the two models are extremely low in both cases, which suggests that our linear model is a poor fit to the data, and in fact linear modelling at all is probably not the best method to apply to our question.

Indeed, as we saw on the graph above, the variability of T is quite high for a given S, which is supported by the coefficients of our models:

```
liglab2_coefs <- liglab2 %>%
  select(c(1, 5)) %>%
  unnest(c(model_coefs))
liglab2_coefs
```

```
## # A tibble: 4 x 6
## # Groups:   S>S_threshold [2]
##   `S>S_threshold` term        estimate std.error statistic p.value
##   <lgl>           <chr>          <dbl>     <dbl>     <dbl>   <dbl>
## 1 FALSE           (Intercept) 3.08     0.0719        42.8   0
## 2 FALSE           S           0.000121 0.0000845      1.44  0.151
## 3 TRUE            (Intercept) 5.31     2.24           2.37  0.0178
## 4 TRUE            S           0.00238  0.00128        1.86  0.0634
```

Here, we don't quite have our coefficients here, as while the $(Intercept)$ term is exactly L, the S term is actually $\frac{1}{C}$. However, we can already see that while L is significantly different from 0 in both regimes of S, C is not, especially in the regime below $S_{threshold}$, whereas in the regime above $S_{threshold}$ it comes closer to a trend.

We could interpret this as the fact that there is always a flat duration necessary for sending a message at all, but *on average*, this duration hardly increases as the message becomes larger. However, given the R^2, this interpretation should obviously be taken with a pinch of salt.

Nevertheless, in both cases the value of $\frac{1}{C}$ is extremely low, which makes an estimation C difficult at best (as small variations of the latter lead to high variations of the latter by taking the inverse). However, we can simply apply a transformation on these rows (and also relabel the terms so we have the correct notation):

```
liglab2_coefs %>%
  mutate(estimate = ifelse(term == "S", 1/estimate, estimate),
         term = ifelse(term == "S", "C", "L"))
```

```
## # A tibble: 4 x 6
## # Groups:   S>S_threshold [2]
##   `S>S_threshold` term  estimate std.error statistic p.value
##   <lgl>           <chr>    <dbl>     <dbl>     <dbl>   <dbl>
## 1 FALSE           L         3.08 0.0719        42.8   0
## 2 FALSE           C      8244.   0.0000845      1.44  0.151
## 3 TRUE            L         5.31 2.24           2.37  0.0178
## 4 TRUE            C       421.   0.00128        1.86  0.0634
```

Indeed, the values of C are extremely high, but they are also probably unreliable for the reasons cited above. However, this illustrate the fact that the relation between T and S is *not* as simple as we assumed above, and in particular that both L and C are dependent on S, such that a better relation would be:

$$T = L(S) + \frac{S}{C(S)}$$

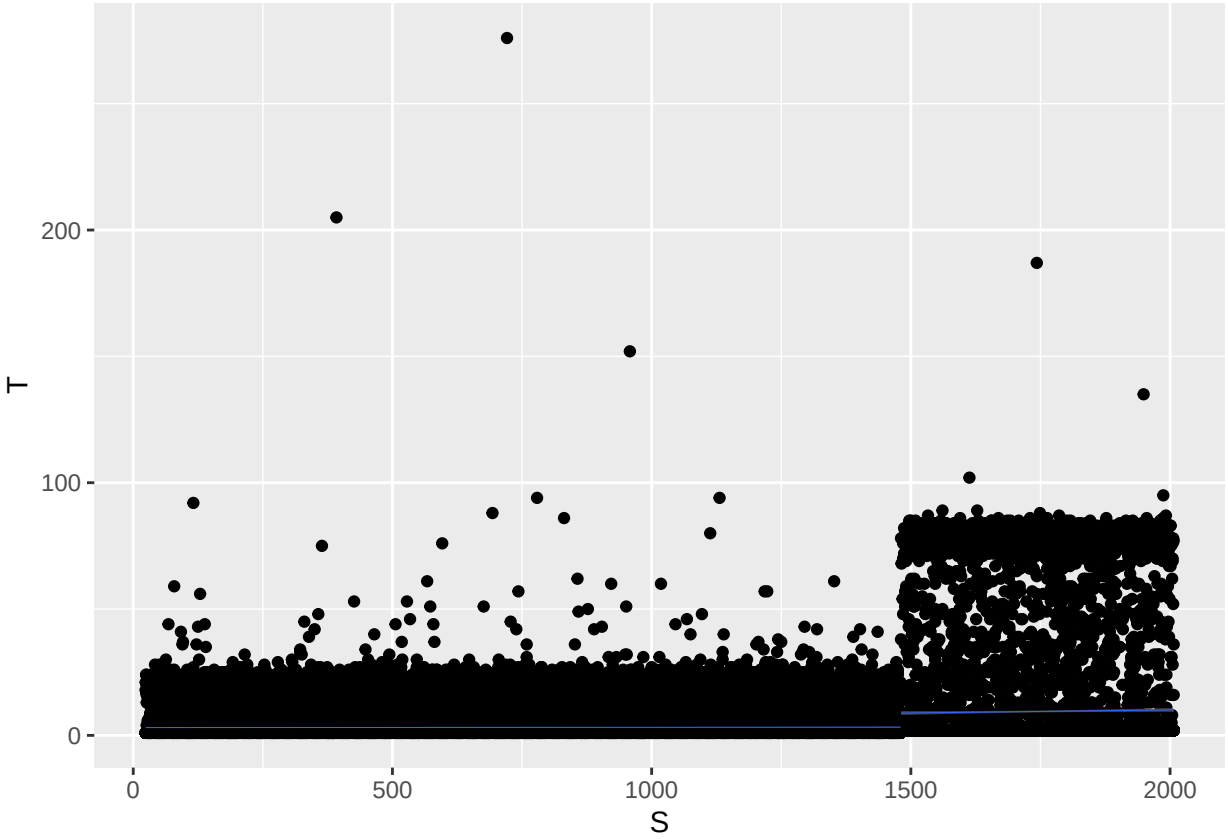
Where the estimation of L and C would be more complex than what we have done. Finally, note that the std.error column is still on the model scale, largely because given how poor a fit the model is, I would not pursue this analysis further in a real case. One could however try to use to obtain confidence intervals.

Plotting the relationships

Despite our models not being very interesting to explain the relation between T and S, we can still try to plot the relationships between the two, separated by regime of S by taking the original plot above, and adding two `geom_smooth` commands, with the two datasets defined above and below $S_{threshold}$, using `method = "lm"` to specify the plotting method and get the 95% confidence intervals:

```
# unnest to be able to access the values of S and T, as otherwise they are stored in a list and not in
unnest(liglab2, cols = c(data)) %>%
  ggplot(aes(x = S, y = T)) +
  geom_point() +
  geom_smooth(data = liglab2$data[[1]], method = "lm", size = .1) +
  geom_smooth(data = liglab2$data[[2]], method = "lm", size = .1)
```

```
## `geom_smooth()` using formula 'y ~ x'
## `geom_smooth()` using formula 'y ~ x'
```



The confidence intervals are there (which can be seen by playing with the `size` argument in the `geom_smooth` calls), but they are tiny. This could be because we have many points in our data.

Same analysis with the `stackoverflow` dataset

Since I will follow largly the same beginning workflow as above, I will merely repeat the titles and code blocks, and not the reasoning for them. ### Missing data

```
NAs_so <- stackoverflow_original %>%
  filter(is.na(T))

print(sprintf("Total number of rows with NA: %i", nrow(NAs_so)))
```

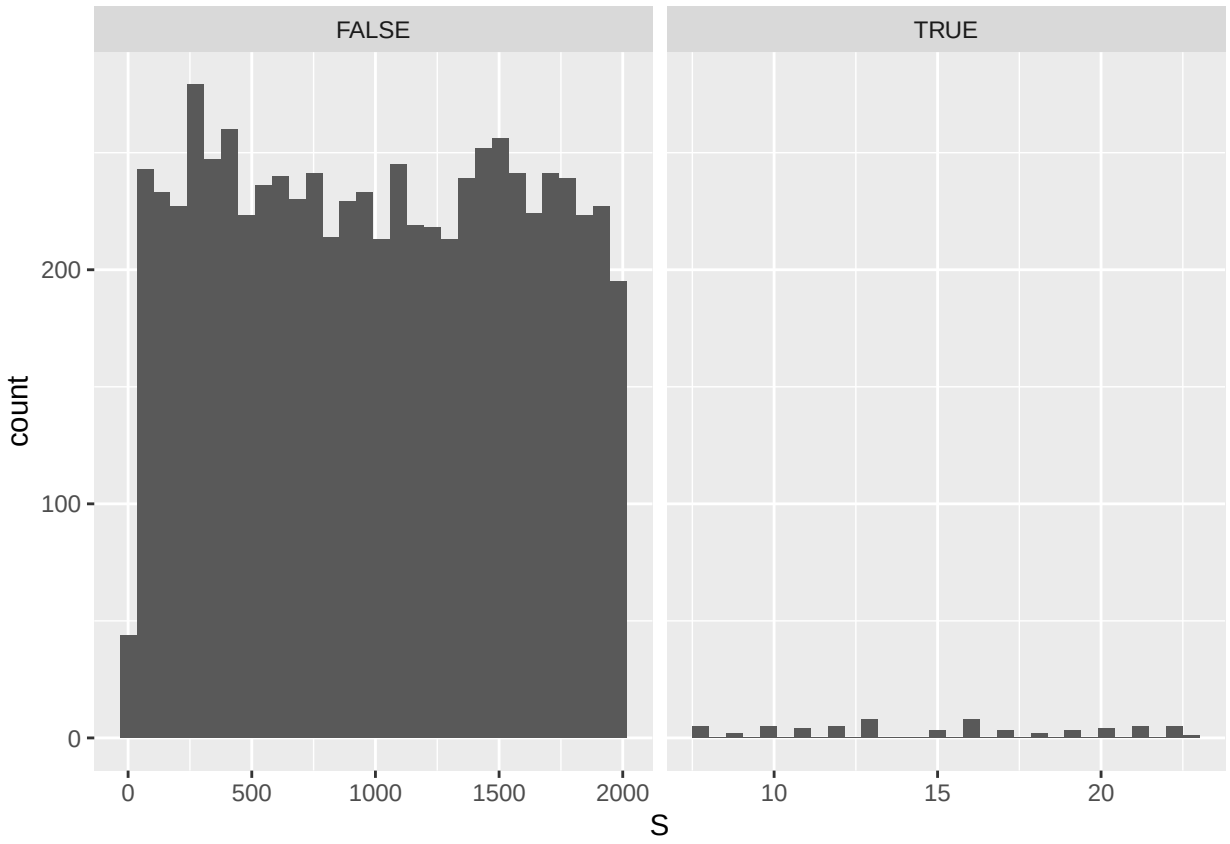
```
## [1] "Total number of rows with NA: 63"
```

```
print(sprintf("Proportion of rows with NA: %f", nrow(NAs_so)/nrow(stackoverflow_original) * 100))
```

```
## [1] "Proportion of rows with NA: 0.914767"
```

```
stackoverflow_original %>%
  ggplot(aes(x = S)) +
  geom_histogram() +
  facet_wrap(~is.na(T), scales = "free_x")
```

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



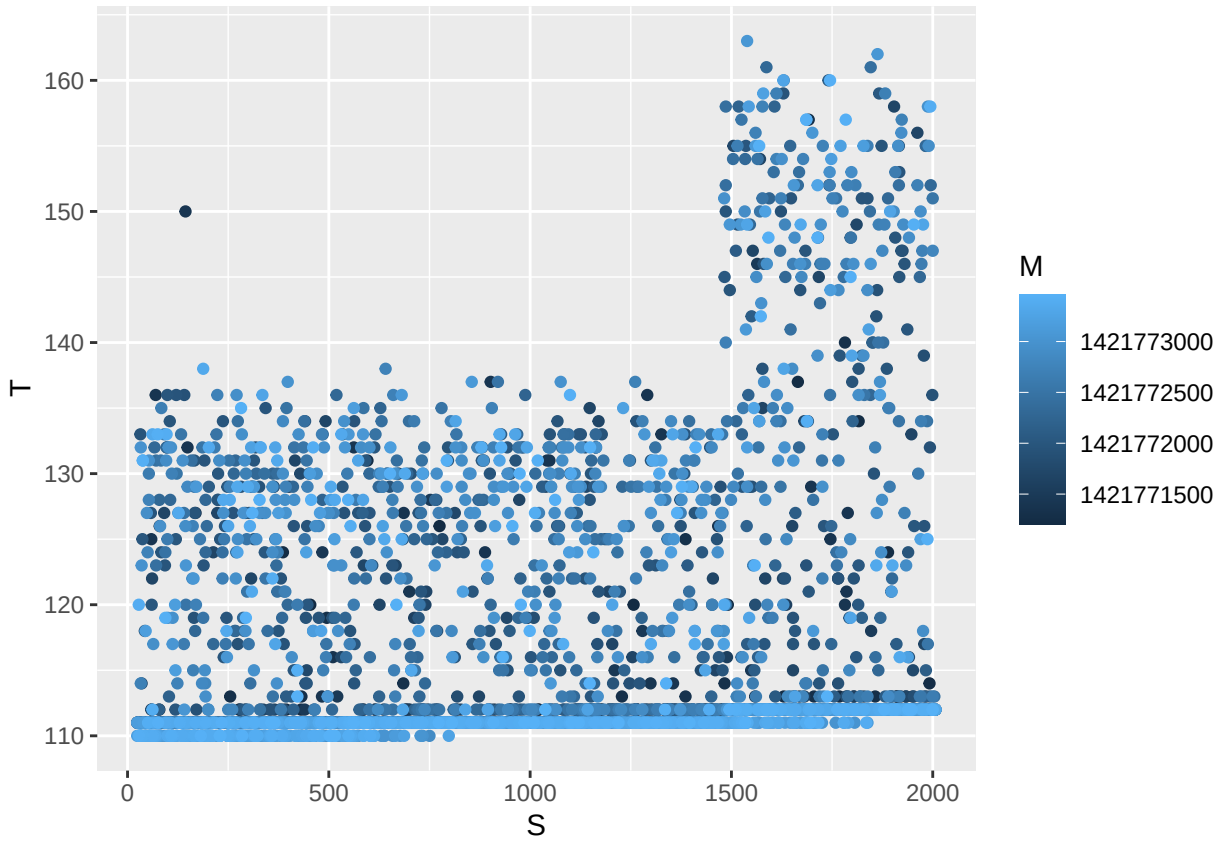
Like with the `liglab2` dataset, the NAs are not Missing at Random, but given with have roughly 1% of them in all the data, I drop them for the analysis, for the same reasons as above (namely, that messages are more likely to be larger than the sizes for who T is missing in the dataset).

```
stackoverflow <- stackoverflow_original %>%
  filter(!(is.na(S) | is.na(T)))
```

Graphical exploration

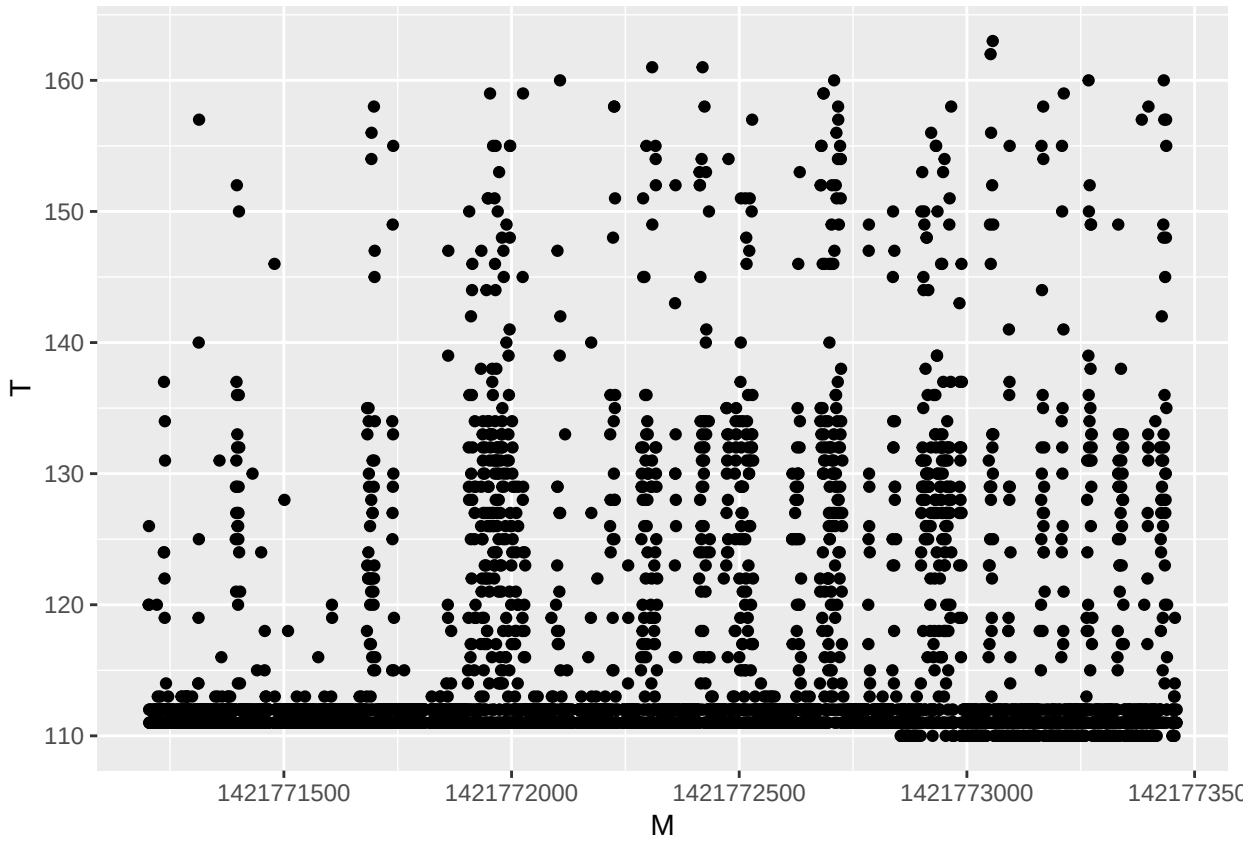
Exploration of the data via plotting

```
ggplot(stackoverflow, aes(x = S, y = T, colour = M)) +
  geom_point()
```



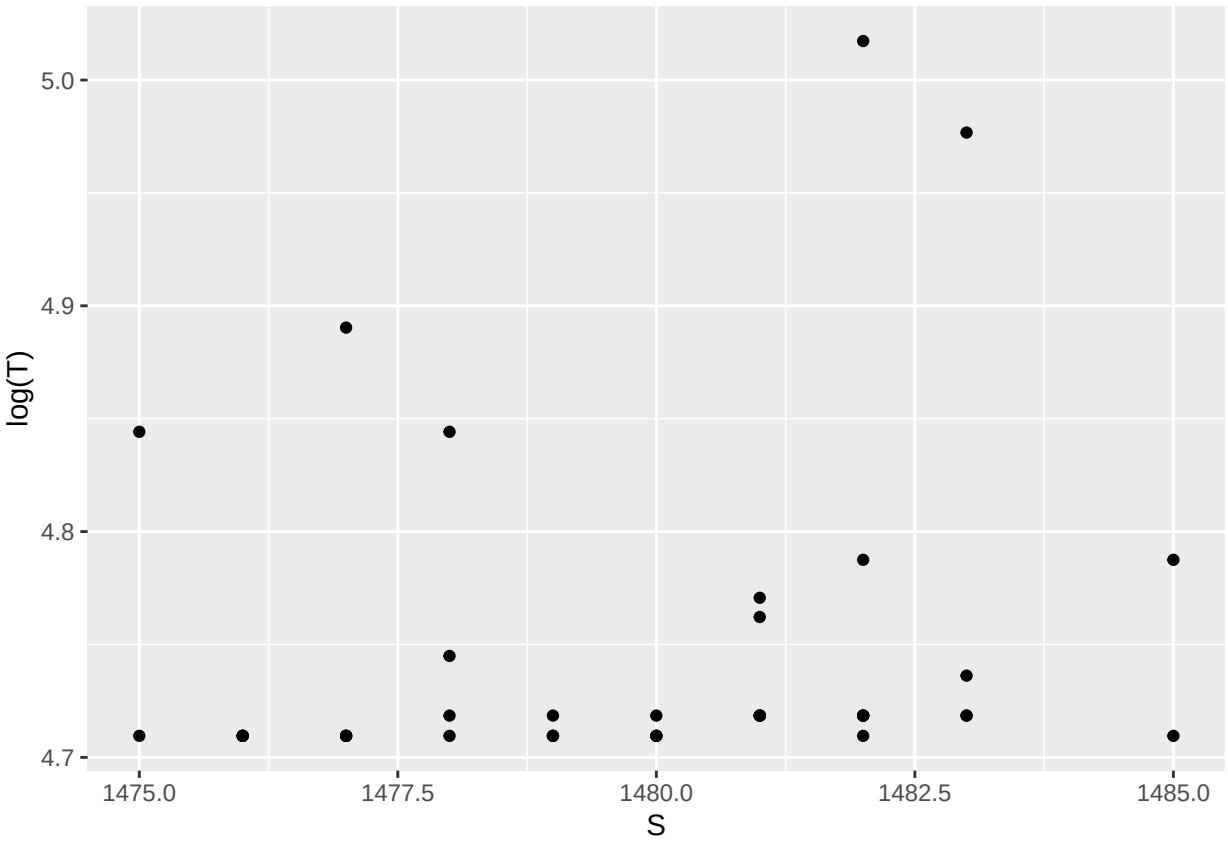
Again, M seems to have little effect on T (i.e. we have temporal stability of T):

```
ggplot(stackoverflow, aes(x = M, y = T)) +
  geom_point()
```



Moreove, it looks like we have the same threshold of S as for the `liglab2` dataset. Let's check it directly with the same plot as above, around S = 1480:

```
stackoverflow %>%
  filter(between(S, 1475, 1485)) %>%
  ggplot(aes(x = S, y = log(T))) +
    geom_point()
```



Here the separation is a little less clear than for the previous dataset, but still appears likely to have the same threshold value. Therefore, we'll keep the same value of $S_{threshold}$. For comparison, we can try repeating the analysis with $S_{threshold} = 1481.5$, but it's unlikely to change much.

Estimating L and C

The pipeline is a bit more involved here, but only because I'm condensing all the steps I did for `liglab2` in one code block.

```
stackoverflow <- stackoverflow %>%
  mutate(`S > S_threshold` = S > S_threshold) %>%
  group_by(`S > S_threshold`) %>%
  nest() %>%
  mutate(models = map(.x = data, .f = ~ lm(data = .x, T~S)),
         coefs = map(.x = models, .f = broom::tidy),
         stats = map(.x = models, .f = broom::glance))
```

Let's check the model statistics:

```
stackoverflow %>%
  select(c("S > S_threshold", "stats")) %>%
  unnest(c(stats))
```

```
## # A tibble: 2 x 12
## # Groups:   S > S_threshold [2]
##   `S > S_threshol~ r.squared adj.r.squared sigma statistic p.value    df  logLik
##   <lgl>               <dbl>         <dbl> <dbl>     <dbl>   <dbl> <int>   <dbl>
## 1 FALSE            0.0000109    -0.000189   5.82    0.0544   0.816     2 -15946.
## 2 TRUE             0.000539     -0.0000144 11.9     0.974    0.324     2  -7043.
## # ... with 4 more variables: AIC <dbl>, BIC <dbl>, deviance <dbl>,
```

```
## #   df.residual <int>
```

Largely the same observation here, that the R^2 values are extremely low, so the models are poor fits to the data. Then for the coefficients:

```
stackoverflow %>%
  select(c("S > S_threshold", "coefs")) %>%
  unnest(c(coefs)) %>%
  mutate(estimate = ifelse(term == "S", 1/estimate, estimate),
         term = ifelse(term == "S", "C", "L"))
```

```
## # A tibble: 4 x 6
## # Groups:   S > S_threshold [2]
##   `S > S_threshold` term  estimate std.error statistic   p.value
##   <lgl>             <chr>    <dbl>     <dbl>     <dbl>     <dbl>
## 1 FALSE             L         113.  0.165      684.    0.
## 2 FALSE             C       22119.  0.000194     0.233 8.16e-  1
## 3 TRUE              L         120.  3.19        37.6   1.72e-229
## 4 TRUE              C        -555.  0.00183     -0.987 3.24e-  1
```

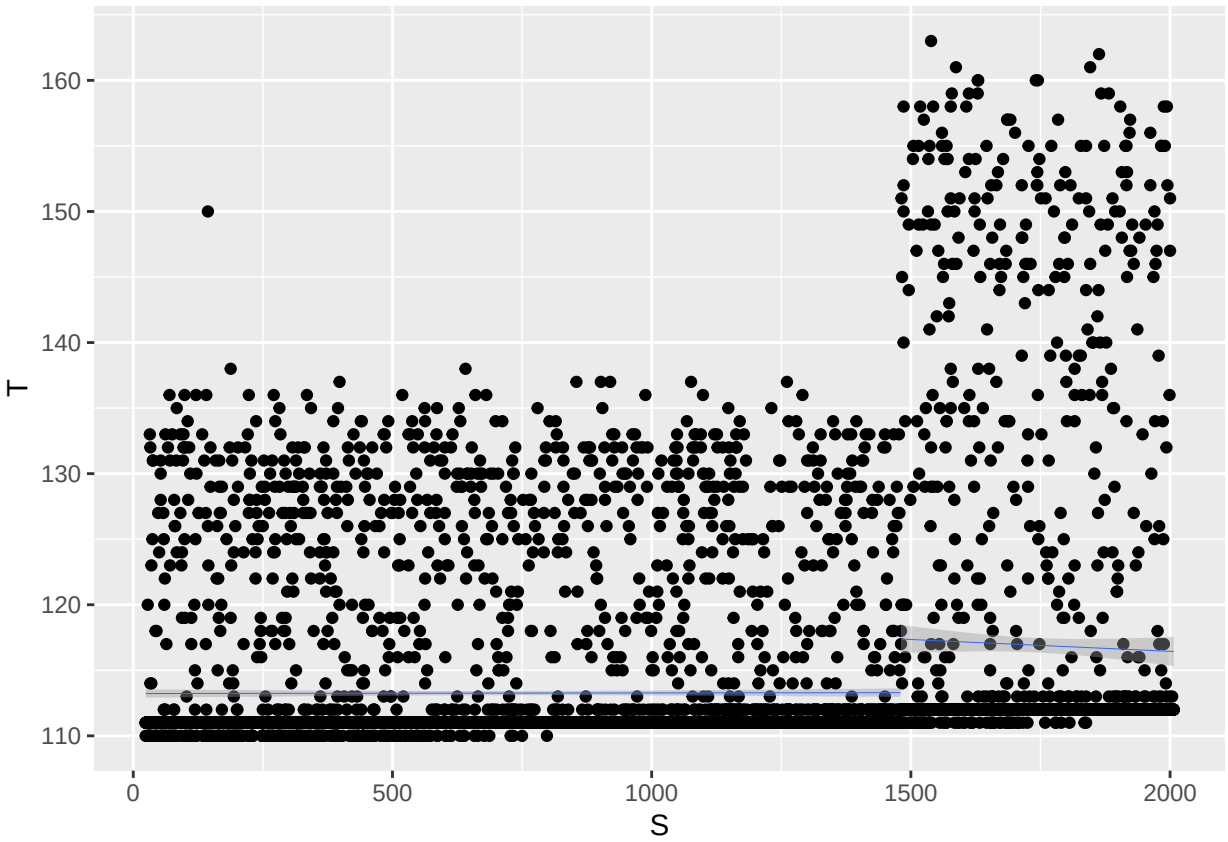
The situation is the same, but even stronger than for `liglab2`: there is no effect whatsoever of C on the value of T, as only L differs significantly from the default.

Interestingly, we find a *negative* value of C for large messages, implying (statistical interpretation notwithstanding) that large messages are sent faster the larger they are, which would be quite a good thing if it was real... Unfortunately it is most probably simply an artefact due to the instability of the estimatation of C.

Plotting the relationships

```
ggplot(unnest(stackoverflow, cols = data), aes(x = S, y = T)) +
  geom_point() +
  geom_smooth(data = stackoverflow$data[[1]], method = "lm", size = .1) +
  geom_smooth(data = stackoverflow$data[[2]], method = "lm", size = .1)
```

```
## `geom_smooth()` using formula 'y ~ x'
## `geom_smooth()` using formula 'y ~ x'
```



Once again, the confidence intervals are shown here but are very narrow. Although they also are likely to make little sense, given the previous observations on the models...

Going further: Quantile regression

Estimating the minimal values of T as a function of S

As we saw, estimating the average of T as a function of L made little statistical sense, in large part due to the variability of T. However, the lower quantiles of T are a little less variable (such as can be seen in the graph just above). We might then wonder if, instead of the *average* value of T like in `lm` we might not have better luck trying to predict the *minimum* value of T. Moreover, the lower value of T does not seem to follow different regimes of S.

One way to accomplish this, suggested on the MOOC page, is to filter the smallest values of T for each message size S.

First, I want to get access to the original data. We can go back to the original formatted data and remove the missing values:

```
liglab2 <- liglab2_original %>%
  filter(!is.na(T))
stackoverflow <- stackoverflow_original %>%
  filter(!is.na(T))
```

Given that we're going to do the exact same things on both datasets, I then combine them into a single data frame, adding a column to record the origin of each data point:

```
dataset_full <- bind_rows(liglab2 = liglab2,  # passing a named list to bind_rows allows using the name
                          stackoverflow = stackoverflow,
                          .id = "dataset") # this only gives a name to the .id column
head(dataset_full)
```

```
##   dataset          M    S  T
## 1 liglab2 1421761682  665 22
## 2 liglab2 1421761682 1373 21
## 3 liglab2 1421761683  262 21
## 4 liglab2 1421761683 1107 23
## 5 liglab2 1421761683 1128  1
## 6 liglab2 1421761683  489 21
```

We can go ahead and add the `S > S_threshold` column as well, so we can separate the two regimes of S later:

```
dataset_full <- dataset_full %>%
  mutate(`S > S_threshold` = S > S_threshold)
```

Next, we'll use `group_by` and `summarise` to take the minimum values of T for each value of S:

```
min_T_per_S <- function(df){
  df %>%
    group_by(dataset, S) %>%
    filter(T == min(T))
}
dataset_min <- min_T_per_S(dataset_full)
```

Interestingly, we can see that this actually keeps a large part of the original data anyway:

```
nrow(dataset_min[dataset_min$dataset == "liglab2", ])/nrow(dataset_full[dataset_full$dataset == "liglab
```

```
## [1] 0.8448996
```

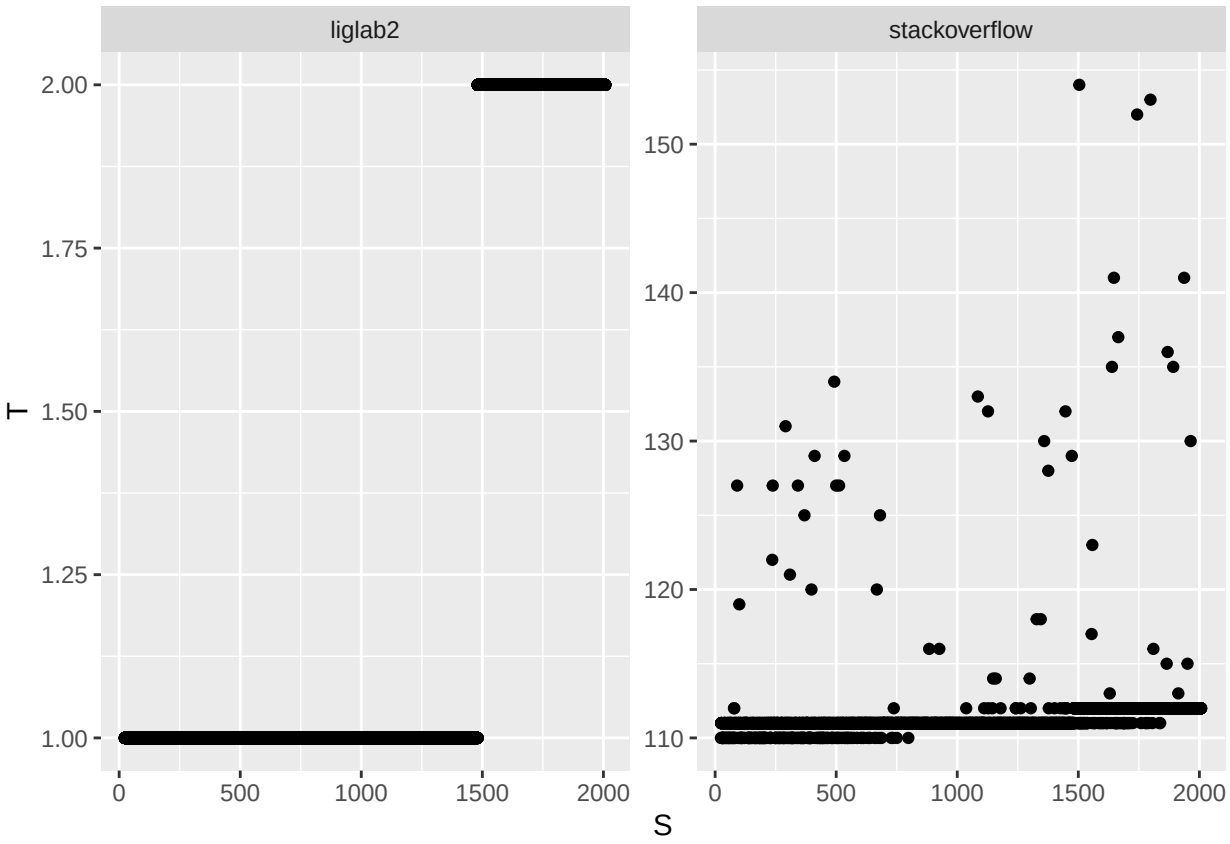
```
nrow(dataset_min[dataset_min$dataset == "stackoverflow", ])/nrow(dataset_full[dataset_full$dataset == "
```

```
## [1] 0.6976846
```

Working only with the minima of T over S

We can plot again the data here to get an idea of what we have to work with:

```
ggplot(dataset_min, aes(x = S, y = T)) +
  geom_point() +
  facet_wrap(~dataset, scales = "free_y")
```



Interestingly, while this suppressed the variability of T completely for `liglab2` (aside from the regimes of S, which are even more blatant here), some variability remains for the `stackoverflow` dataset.

However, in this case, the `liglab2` dataset, the model could be summed up as a simple enough equation:

$$T = \begin{cases} 1 & if\ S < S_{threshold} \\ \\ 2 & if\ S > S_{threshold} \end{cases}$$

Which makes it of little interest to actually make a model of the relation, meaning we lost quite a bit of information by restricting the data to the minimum values of T for each value of S. We could of course do it, but it would make little theoretical sense. Therefore, let's move on.

However, for `stackoverflow`, there is still some variability left, some we could try making a model to estimate L and C like we did before:

```
so_min <- dataset_min %>%
  filter(dataset == "stackoverflow") %>%  # remove the liglab2 data
  group_by(`S > S_threshold`) %>%
  nest() %>%
  mutate(models = map(.x = data, .f = ~ lm(T ~ S, data = .x)),
         coefs = map(.x = models, .f = tidy),
         stats = map(.x = models, .f = glance))
```

Let's inspect the stats:

```
so_min %>%
  select(c("S > S_threshold", "stats")) %>%
  unnest(c(stats))
```

```
## # A tibble: 2 x 12
```

```
## # Groups:   S > S_threshold [2]
##   `S > S_threshol~ r.squared adj.r.squared sigma statistic p.value    df logLik
##   <lgl>               <dbl>         <dbl> <dbl>     <dbl>   <dbl> <int>  <dbl>
## 1 FALSE             0.00310       0.00282   1.36    10.9   9.78e-4     2 -6039.
## 2 TRUE              0.000128     -0.000668  2.75     0.160 6.89e-1     2 -3057.
## # ... with 4 more variables: AIC <dbl>, BIC <dbl>, deviance <dbl>,
## #   df.residual <int>
```

Looks like we're having the same issue as before with the R^2, so taking the minimal values of T probably didn't help too much to get a good model. Let's see the coefficients of the models:

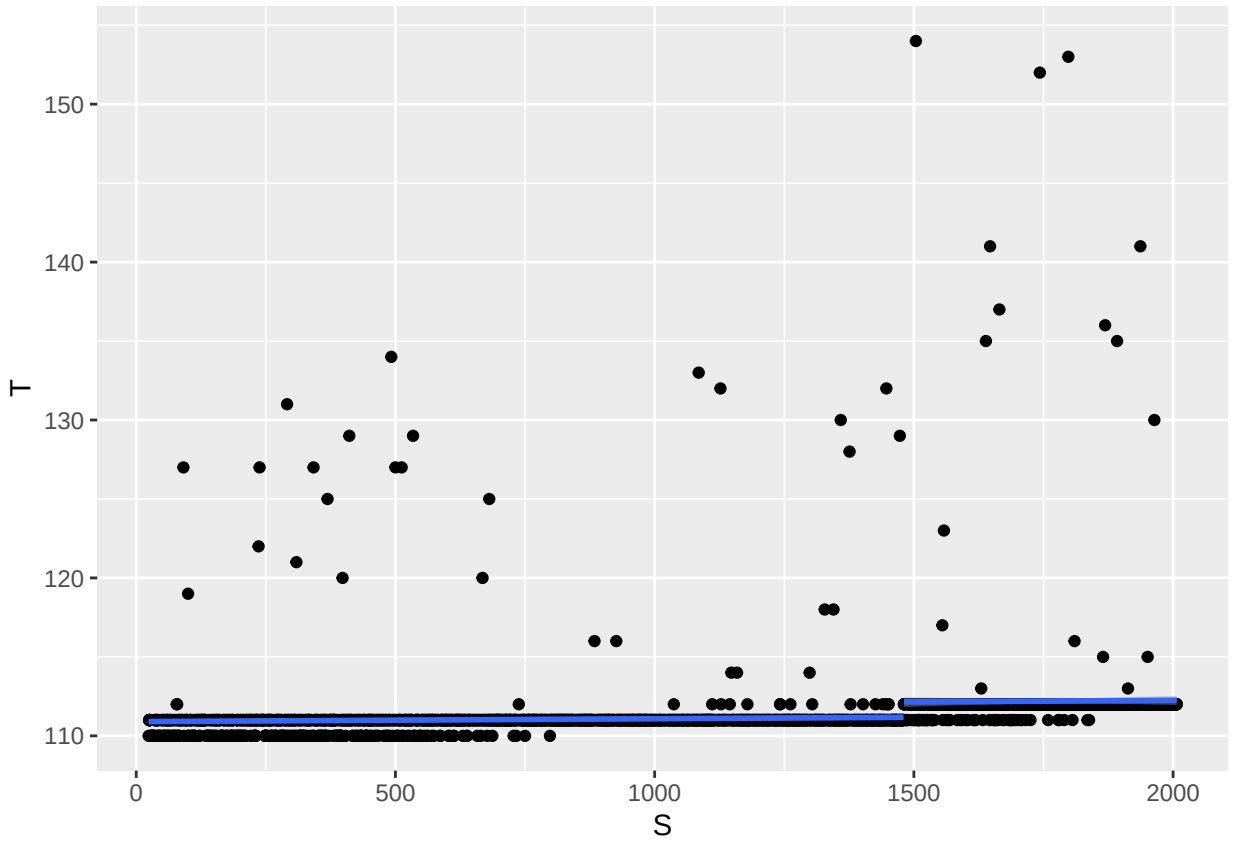
```
so_min %>%
  select(c("S > S_threshold", "coefs")) %>%
  unnest(c(coefs)) %>%
  mutate(estimate = ifelse(term == "S", 1/estimate, estimate),
         term = ifelse(term == "S", "C", "L"))
```

```
## # A tibble: 4 x 6
## # Groups:   S > S_threshold [2]
##   `S > S_threshold` term  estimate std.error statistic  p.value
##   <lgl>             <chr>    <dbl>     <dbl>     <dbl>    <dbl>
## 1 FALSE             L         111. 0.0491     2257.    0
## 2 FALSE             C        5457. 0.0000555     3.30  0.000978
## 3 TRUE              L         112. 0.903       124.    0
## 4 TRUE              C        4878. 0.000512      0.400 0.689
```

Ah, unlike earlier, we do have a significant effect of C for S under $S_threshold$, but not over it. Let's see what this looks like in a plot:

```
ggplot(unnest(so_min, cols = data), aes(x = S, y = T)) +
  geom_point() +
  geom_smooth(data = so_min$data[[1]], method = "lm", size = 1) +
  geom_smooth(data = so_min$data[[2]], method = "lm", size = 1)
```

```
## `geom_smooth()` using formula 'y ~ x'
## `geom_smooth()` using formula 'y ~ x'
```



Not really the most explicit of graphs either... The regressions are barely more than lines, like above. Looks like just taking the minimal values of T doesn't solve our problem, so let's try something else, a little more general: quantile regression.

Quantile regression with `quantreg`

Instead of estimating the average of T as a function of S, or artificially filtering the values of T over S like we have done above, we can try to estimate *quantiles* of T as a function of S, using quantile regression. In R, this is handled by the `quantereg` package:

```
library(quantreg)
```

```
## Loading required package: SparseM
```

```
##
## Attaching package: 'SparseM'
```

```
## The following object is masked from 'package:base':
##
##     backsolve
```

We then apply the `rq` function to do quantile regression on both datasets. To spice things up, instead of a single model, let's treat `tau`, the parameter of `rq` the determines the quantile of T to predict, as a hyperparameter, and try to see how it changes the resulting models. The models are extremely unlikely to change under around $tau = 0.65$ (given that 84 and 65 percents of the data in the `liglab2` and `stackoverflow` datasets, respectively, are equal to the minimum value of T), but we'll check it anyway, because it will change the distribution of S ever so slightly. We can do the estimation by taking values of `tau` every .05 from 0.05 to 0.95 (so from the bottom 5% of T to the bottom 95%): **NB: this will take some time (around 30 s on my MacBook 2015)**

```
models <- dataset_full %>%
  group_by(dataset, `S > S_threshold`) %>%
  nest() %>%
  mutate(model = map(.x = data,
                     .f = ~ rq(T~S, tau = seq(0.05, .95, by = .05), data=.x)),
         coefs = map(.x = model, .f = tidy, conf.level = 0.95)) # computes the 95% confidence interval
        # note: .95 is the default value anyway, but it is not specified in the outputs, so I put it he
```

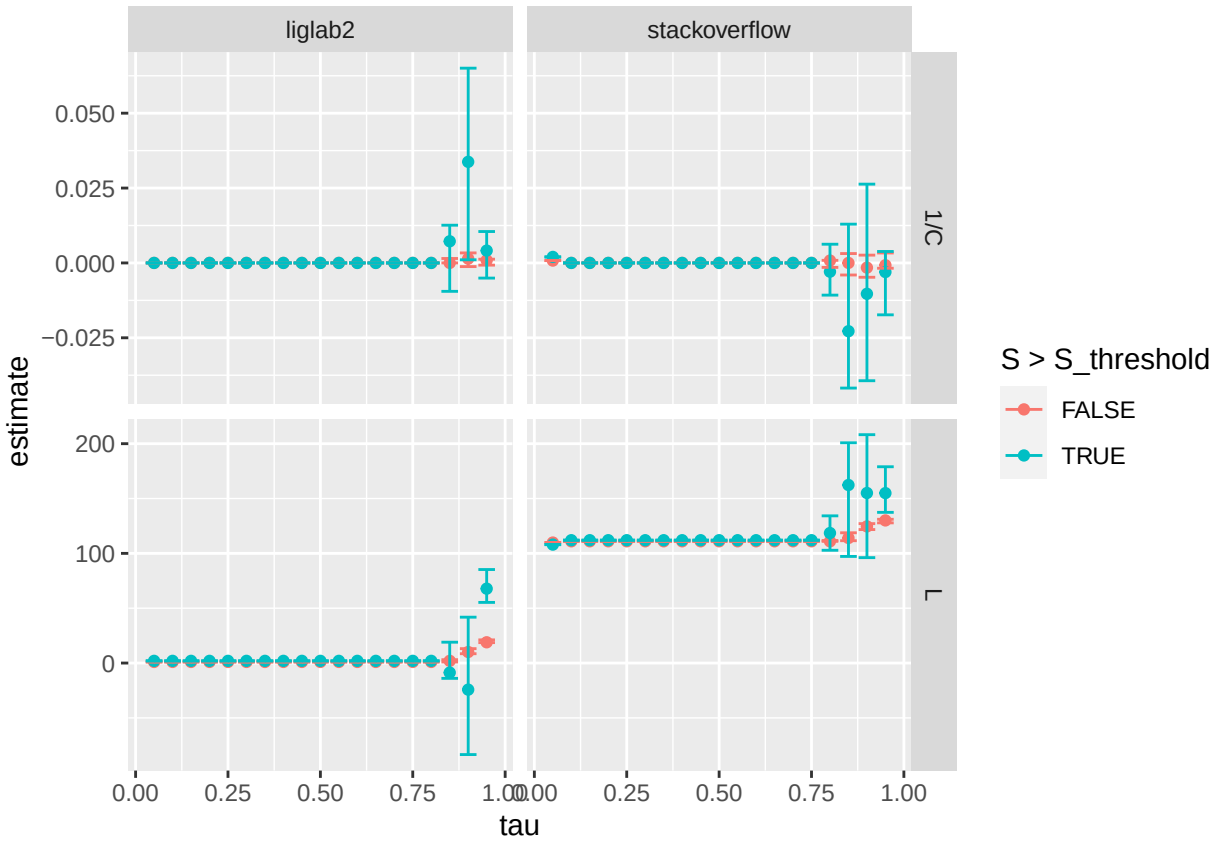
Then we can unpack the resulting coefficients:

```
coefs <- models %>%
  select(c("dataset", "S > S_threshold", "coefs")) %>%
  unnest(c(coefs)) %>%
  mutate(term = ifelse(term == "S", "1/C", "L"))
coefs
```

```
## # A tibble: 152 x 7
## # Groups:   dataset, S > S_threshold [4]
##    dataset `S > S_threshold` term  estimate conf.low conf.high   tau
##    <chr>   <lgl>             <chr>    <dbl>    <dbl>     <dbl> <dbl>
##  1 liglab2 FALSE             L            1        1         1  0.05
##  2 liglab2 FALSE             1/C          0        0         0  0.05
##  3 liglab2 FALSE             L            1        1         1  0.1
##  4 liglab2 FALSE             1/C          0        0         0  0.1
##  5 liglab2 FALSE             L            1        1         1  0.15
##  6 liglab2 FALSE             1/C          0        0         0  0.15
##  7 liglab2 FALSE             L            1        1         1  0.2
##  8 liglab2 FALSE             1/C          0        0         0  0.2
##  9 liglab2 FALSE             L            1        1         1  0.25
## 10 liglab2 FALSE             1/C          0        0         0  0.25
## # ... with 142 more rows
```

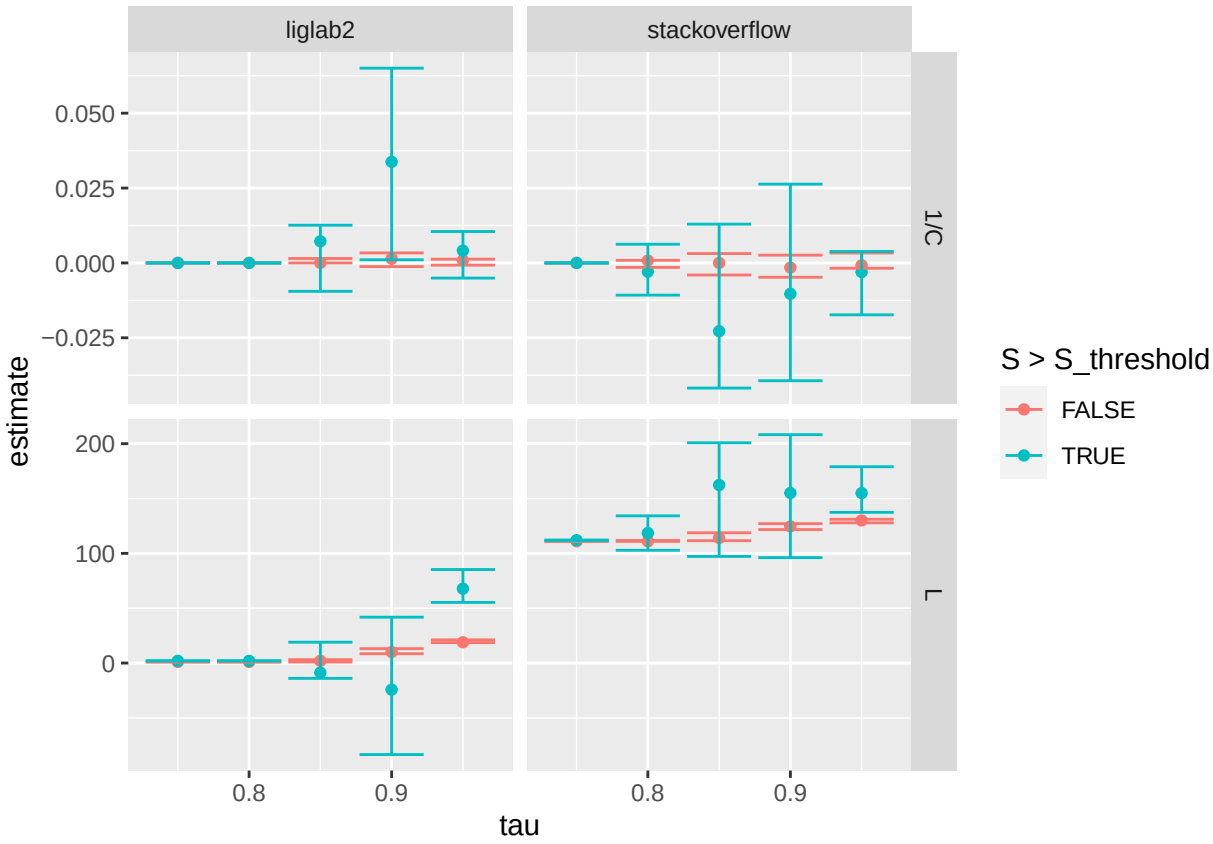
Immediately (and somewhat coincidentally), we can tell that we'll have a bit of a problem with the lower quantiles for `liglab2`, which makes sense as seen above for the lower quantiles of T, there's no variation in S whatsoever, so of course the value of $\frac{1}{C}$ will be 0. To get a better idea of the coefficients, let's plot their value as a function of *tau*. For clarity, we'll separate the terms of the models, as well as the datasets, and use different colours of the two different regimes of S.

```
ggplot(coefs, aes(x = tau, y = estimate, colour = `S > S_threshold`)) +
  geom_point() +
  geom_errorbar(aes(ymin = conf.low, ymax = conf.high)) +
  facet_grid(term~dataset, scales = "free_y")
```



Predictably, only the higher quartiles start to differ, so let's just zoom in for a better look, say above .75:

```
ggplot(coefs %>% filter(tau >= 0.75), # take the higher quartile models only
       aes(x = tau, y = estimate, colour = `S > S_threshold`)) +
  geom_point() +
  geom_errorbar(aes(ymin = conf.low, ymax = conf.high)) +
  facet_grid(term~dataset, scales = "free_y")
```



Conclusions We can see here than, much like we found precedently, C has relatively little influence on T, although this increases at the higher quantiles of T.

Interestingly, what little trends are there are contrary between the two datasets, with $1/C$ increasing at higher values of `tau` (and so C would decrease) for the `liglab2` dataset, and vice-versa for the `stackoverflow` dataset. However, this trend only exists for values of S above $S_{threshold}$ and not under. However, the capacity of the connection appears to vary little when any other parameter varies, which makes sense.

Moreover, L has a much higher value for the `stackoverflow` dataset than for the `liglab2` dataset, regardless of the value of `tau`. This can likely be explained by the distance of the two connections: `liglab2` comes from a local, on-campus connection, while `stackoverflow` comes from a long-distance connection, which automatically means a higher latence to transfer information.

We have compared the parameters of two different connections using linear and quantile regression. We have seen that linear regression was a poor choice of method for the data, as was a simple restriction of the values, thus orienting our choice to a different technique, that bypasses the problems of linear regression (namely, that it estimates the *average* value of the response variable), by estimating *quantiles* of the response variable T.