EDA and data visualization

Monica Alexander

01/02/23

Table of contents

1 Overview

This week we will be going through some exploratory data analysis (EDA) and data visualization steps in R. The aim is to get you used to working with real data (that has issues) to understand the main characteristics and potential issues.

We will be using the `opendatatoronto` R package, which interfaces with the City of Toronto Open Data Portal.

A good resource is part 1 (especially chapters 3 and 7) of 'R for Data Science' by Hadley Wickham, available for free here: https://r4ds.had.co.nz/.

1.1 What to hand in via GitHub

There are exercises at the end of this lab. Please make a new .Rmd file with your answers, call it something sensible (e.g. `week_2_lab.Rmd`), commit to your git repo from last week, and push to GitHub. Due on Monday by 9am.

1.2 A note on packages

You may need to install various packages used (using the `install.packages` function). Load in all the packages we need:

```
library(opendatatoronto)
```

```
Warning: package 'opendatatoronto' was built under R version 4.2.2
```

```
library(tidyverse)
```

```
Warning: package 'tidyverse' was built under R version 4.2.2
```

```
library(stringr)
library(skimr) # EDA
```

```
Warning: package 'skimr' was built under R version 4.2.2
```

```
library(visdat) # EDA
```

```
Warning: package 'visdat' was built under R version 4.2.2
```

```
library(janitor)
```

```
Warning: package 'janitor' was built under R version 4.2.2
```

```
library(lubridate)
```

```
Warning: package 'lubridate' was built under R version 4.2.2

Warning: package 'timechange' was built under R version 4.2.2
```

```r
library(ggrepel)
```

```
Warning: package 'ggrepel' was built under R version 4.2.2
```

2 TTC subway delays

This package provides an interface to all data available on the Open Data Portal provided by the City of Toronto.

Use the `list_packages` function to look whats available look at what's available

```r
all_data <- list_packages(limit = 500)
head(all_data)
```

```
# A tibble: 6 x 11
  title     id    topics civic~1 publi~2 excerpt datas~3 num_r~4 formats refre~5
  <chr>     <chr> <chr>  <chr>   <chr>   <chr>   <chr>     <int> <chr>   <chr>
1 Developm~ 0aa7~ <NA>   <NA>    City P~ "This ~ Table         4 CSV,XM~ Monthly
2 Polls co~ 7bce~ City ~ <NA>    City C~ "Polls~ Table         5 XML,JS~ Daily
3 COVID-19~ d3f2~ Health <NA>    Toront~ "This ~ Map          13 GEOJSO~ Daily
4 Toronto'~ c6d6~ <NA>   <NA>    City M~ "This ~ Table         4 XML,JS~ Daily
5 Committe~ 260e~ City ~ Afford~ City P~ "This ~ Table        96 CSV,XM~ Weekly
6 Apartmen~ 4ef8~ Locat~ Afford~ Munici~ "This ~ Table         4 XML,JS~ Daily
# ... with 1 more variable: last_refreshed <date>, and abbreviated variable
#   names 1: civic_issues, 2: publisher, 3: dataset_category, 4: num_resources,
#   5: refresh_rate
```

Let's download the data on TTC subway delays in 2022.

```r
res <- list_package_resources("996cfe8d-fb35-40ce-b569-698d51fc683b") # obtained code from
res <- res |> mutate(year = str_extract(name, "202.?"))
delay_2022_ids <- res |> filter(year==2022) |> select(id) |> pull()

delay_2022 <- get_resource(delay_2022_ids)

# make the column names nicer to work with
delay_2022 <- clean_names(delay_2022)
```

Let's also download the delay code and readme, as reference.

```
# note: I obtained these codes from the 'id' column in the `res` object above
delay_codes <- get_resource("3900e649-f31e-4b79-9f20-4731bbfd94f7")
```

```
New names:
* `` -> `...1`
* `CODE DESCRIPTION` -> `CODE DESCRIPTION...3`
* `` -> `...4`
* `` -> `...5`
* `CODE DESCRIPTION` -> `CODE DESCRIPTION...7`
```

```
delay_data_codebook <- get_resource("ca43ac3d-3940-4315-889b-a9375e7b8aa4")
```

This dataset has a bunch of interesting variables. You can refer to the readme for descriptions. Our outcome of interest is `min_delay`, which give the delay in mins.

```
head(delay_2022)
```

```
# A tibble: 6 x 10
  date                time  day      station   code  min_d~1 min_gap bound line
  <dttm>              <chr> <chr>    <chr>     <chr>   <dbl>   <dbl> <chr> <chr>
1 2022-01-01 00:00:00 15:59 Saturday LAWRENCE~ SRDP        0       0 N     SRT
2 2022-01-01 00:00:00 02:23 Saturday SPADINA ~ MUIS        0       0 <NA>  BD
3 2022-01-01 00:00:00 22:00 Saturday KENNEDY ~ MRO         0       0 <NA>  SRT
4 2022-01-01 00:00:00 02:28 Saturday VAUGHAN ~ MUIS        0       0 <NA>  YU
5 2022-01-01 00:00:00 02:34 Saturday EGLINTON~ MUATC       0       0 S     YU
6 2022-01-01 00:00:00 05:40 Saturday QUEEN ST~ MUNCA       0       0 <NA>  YU
# ... with 1 more variable: vehicle <dbl>, and abbreviated variable name
#   1: min_delay
```

3 EDA and data viz

The following section highlights some tools that might be useful for you when you are getting used to a new dataset. There's no one way of exploration, but it's important to always keep in mind:

- what should your variables look like (type, values, distribution, etc)
- what would be surprising (outliers etc)
- what is your end goal (here, it might be understanding factors associated with delays, e.g. stations, time of year, time of day, etc)

In any data analysis project, if it turns out you have data issues, surprising values, missing data etc, it's important you **document** anything you found and the subsequent steps or **assumptions** you made before moving onto your data analysis / modeling.

3.1 Data checks

3.1.1 Sanity Checks

We need to check variables should be what they say they are. If they aren't, the natural next question is to what to do with issues (recode? remove?)

E.g. check days of week

```
unique(delay_2022$day)
```

```
[1] "Saturday"  "Sunday"    "Monday"    "Tuesday"   "Wednesday" "Thursday"
[7] "Friday"
```

Check lines: oh no. some issues here. Some have obvious recodes, others, not so much.

```
unique(delay_2022$line)
```

```
 [1] "SRT"              "BD"               "YU"               "YU/BD"
 [5] "SHP"              NA                 "BD/YU"            "YU / BD"
 [9] "YU/ BD"           "B/D"              "Y/BD"             "YU/BD LINES"
[13] "YUS"              "YU & BD"          "YUS AND BD"       "YUS/BD"
[17] "69 WARDEN SOUTH" "YU/BD LINE"       "LINE 2 SHUTTLE" "57 MIDLAND"
[21] "96 WILSON"        "506 CARLTON"
```

The `skimr` package might also be useful here

```
skim(delay_2022)
```

Table 1: Data summary

Name	delay_2022
Number of rows	19895
Number of columns	10

Table 1: Data summary

Column type frequency:	
character	6
numeric	3
POSIXct	1
Group variables	None

Variable type: character

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
time	0	1.00	5	5	0	1406	0
day	0	1.00	6	9	0	7	0
station	0	1.00	5	22	0	296	0
code	0	1.00	3	5	0	179	0
bound	5546	0.72	1	1	0	5	0
line	39	1.00	2	15	0	21	0

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
min_delay	0	1	3.67	12.00	0	0	0	4	458	
min_gap	0	1	5.33	12.66	0	0	0	8	463	
vehicle	0	1	3571.59	2646.62	0	0	5192	5701	8871	

Variable type: POSIXct

skim_variable	n_missing	complete_rate	min	max	median	n_unique
date	0	1	2022-01-01	2022-12-31	2022-06-29	365

3.1.2 Missing values

Calculate number of NAs by column

```
delay_2022 |>
  summarize(across(everything(), ~ sum(is.na(.x))))
```

```
# A tibble: 1 x 10
   date  time   day station  code min_delay min_gap bound  line vehicle
  <int> <int> <int>   <int> <int>     <int>   <int> <int> <int>   <int>
1     0     0     0       0     0         0       0  5546    39       0
```

The visdat package is useful here, particularly to see how missing values are distributed. (commented out because couldn't get pdf to render in quarto)

```
#vis_dat(delay_2022)
#vis_miss(delay_2022)
```

3.1.3 Duplicates?

The get_dupes function from the janitor package is useful for this.

```
get_dupes(delay_2022)
```

```
No variable names specified - using all columns.
```

```
# A tibble: 28 x 11
   date                time  day       station code  min_d~1 min_gap bound line
   <dttm>              <chr> <chr>     <chr>   <chr>   <dbl>   <dbl> <chr> <chr>
 1 2022-01-12 00:00:00 13:27 Wednesday FINCH ~ TUNOA       3       6 S     YU
 2 2022-01-12 00:00:00 13:27 Wednesday FINCH ~ TUNOA       3       6 S     YU
 3 2022-01-12 00:00:00 17:49 Wednesday FINCH ~ TUNOA       3       6 S     YU
 4 2022-01-12 00:00:00 17:49 Wednesday FINCH ~ TUNOA       3       6 S     YU
 5 2022-01-17 00:00:00 02:00 Monday    SCARBO~ TRST        0       0 <NA>  SRT
 6 2022-01-17 00:00:00 02:00 Monday    SCARBO~ TRST        0       0 <NA>  SRT
 7 2022-01-20 00:00:00 02:30 Thursday  YONGE ~ TUST        0       0 <NA>  YU
 8 2022-01-20 00:00:00 02:30 Thursday  YONGE ~ TUST        0       0 <NA>  YU
 9 2022-01-20 00:00:00 08:51 Thursday  WILSON~ TUNOA       3       6 S     YU
10 2022-01-20 00:00:00 08:51 Thursday  WILSON~ TUNOA       3       6 S     YU
# ... with 18 more rows, 2 more variables: vehicle <dbl>, dupe_count <int>, and
#   abbreviated variable name 1: min_delay
```

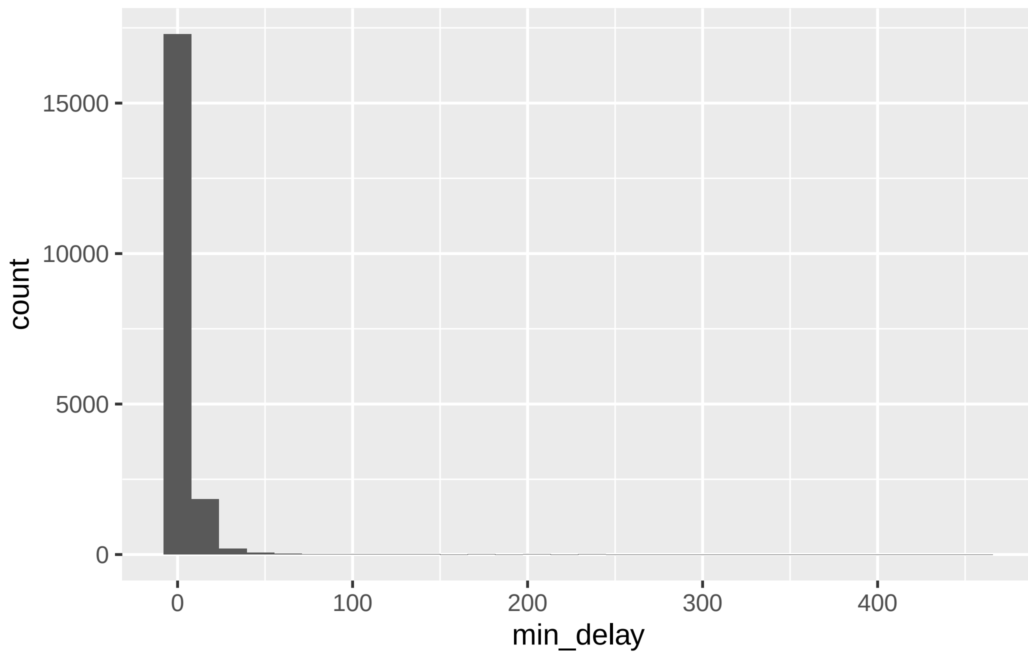
3.2 Visualizing distributions

Histograms, barplots, and density plots are your friends here.

Let's look at the outcome of interest: min_delay. First of all just a histogram of all the data:

```r
## Removing the observations that have non-standardized lines

delay_2022 <- delay_2022 |> filter(line %in% c("BD", "YU", "SHP", "SRT"))

ggplot(data = delay_2022) +
  geom_histogram(aes(x = min_delay))
```

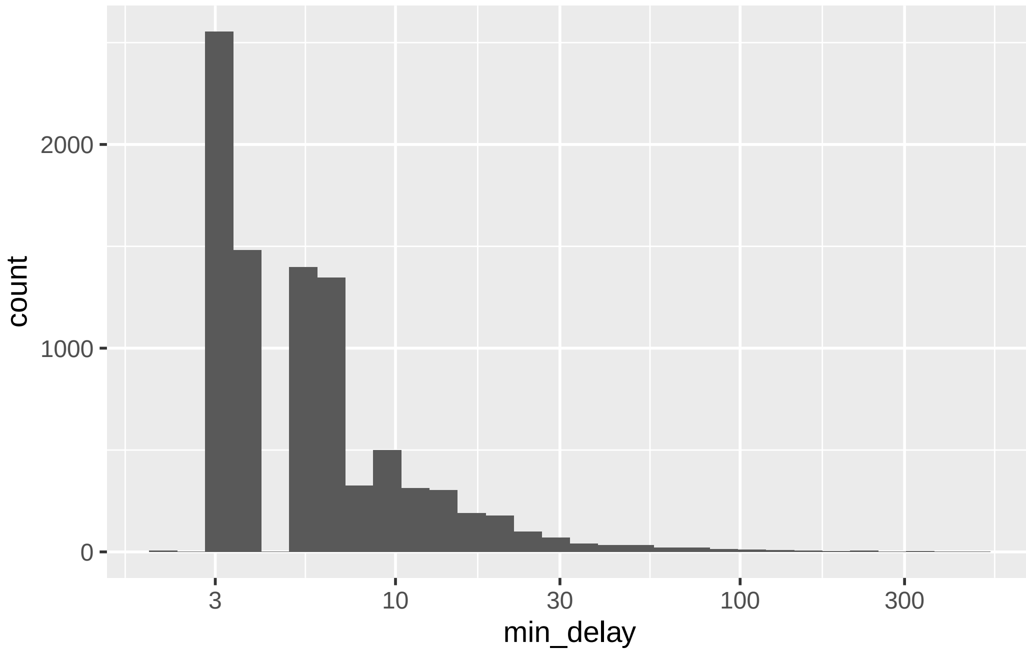
```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



To improve readability, could plot on logged scale:

```r
ggplot(data = delay_2022) +
  geom_histogram(aes(x = min_delay)) +
  scale_x_log10()
```

```
Warning: Transformation introduced infinite values in continuous x-axis

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

Warning: Removed 10500 rows containing non-finite values (stat_bin).
```



Our initial EDA hinted at an outlying delay time, let's take a look at the largest delays below. Join the `delay_codes` dataset to see what the delay is. (Have to do some mangling as SRT has different codes).

```
delay_2022 <- delay_2022 |>
  left_join(delay_codes |> rename(code = `SUB RMENU CODE`, code_desc = `CODE DESCRIPTION..
```

```
Joining, by = "code"
```

```
delay_2022 <- delay_2022 |>
  mutate(code_srt = ifelse(line=="SRT", code, "NA")) |>
  left_join(delay_codes |> rename(code_srt = `SRT RMENU CODE`, code_desc_srt = `CODE DESCR
  mutate(code = ifelse(code_srt=="NA", code, code_srt),
         code_desc = ifelse(is.na(code_desc_srt), code_desc, code_desc_srt)) |>
  select(-code_srt, -code_desc_srt)
```

```
Joining, by = "code_srt"
```

The largest delay is due to "Signals Other".

```
delay_2022 |>
  left_join(delay_codes |> rename(code = `SUB RMENU CODE`, code_desc = `CODE DESCRIPTION..
  arrange(-min_delay) |>
  select(date, time, station, line, min_delay, code, code_desc)
```

```
Joining, by = c("code", "code_desc")
```

```
# A tibble: 19,473 x 7
   date                time  station              line  min_de~1 code  code_~2
   <dttm>              <chr> <chr>                <chr>    <dbl> <chr> <chr>
 1 2022-12-08 00:00:00 17:52 MIDLAND STATION      SRT        458 MRPLB Fire/S~
 2 2022-08-22 00:00:00 12:20 SRT LINE             SRT        451 PRSO  Signal~
 3 2022-04-28 00:00:00 06:02 JANE STATION         BD         388 PUTR  Rail R~
 4 2022-07-26 00:00:00 07:06 YONGE BD STATION     BD         382 MUPLB Fire/S~
 5 2022-08-15 00:00:00 12:57 DUFFERIN STATION     BD         327 MUPR1 Priori~
 6 2022-01-26 00:00:00 20:15 KENNEDY SRT STATION  SRT        315 MRWEA Weathe~
 7 2022-08-02 00:00:00 21:23 HIGHWAY 407 STATION  YU         312 MUPR1 Priori~
 8 2022-01-17 00:00:00 21:30 SHEPPARD WEST TO UNION YU       291 MUFM  Force ~
 9 2022-01-25 00:00:00 21:03 SCARBOROUGH CTR STATIO SRT      285 PRSL  Loop R~
10 2022-06-17 00:00:00 12:25 KIPLING STATION      BD         241 SUUT  Unauth~
# ... with 19,463 more rows, and abbreviated variable names 1: min_delay,
#   2: code_desc
```

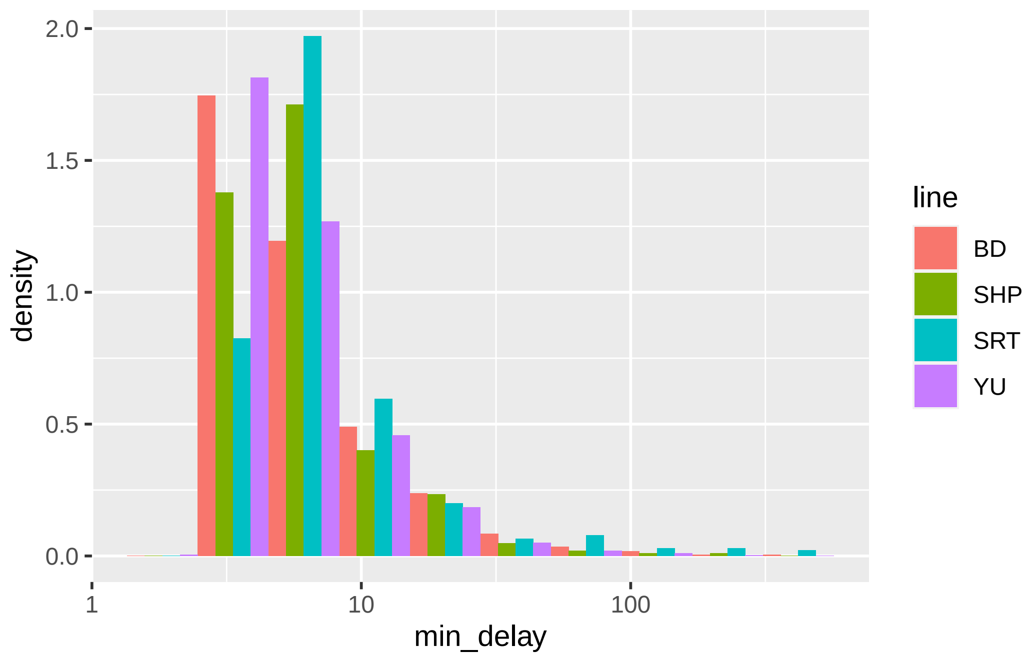
3.2.0.1 Grouping and small multiples

A quick and powerful visualization technique is to group the data by a variable of interest, e.g. `line`

```
ggplot(data = delay_2022) +
  geom_histogram(aes(x = min_delay, y = ..density.., fill = line), position = 'dodge', bin
  scale_x_log10()
```

```
Warning: Transformation introduced infinite values in continuous x-axis
```

```
Warning: Removed 10500 rows containing non-finite values (stat_bin).
```

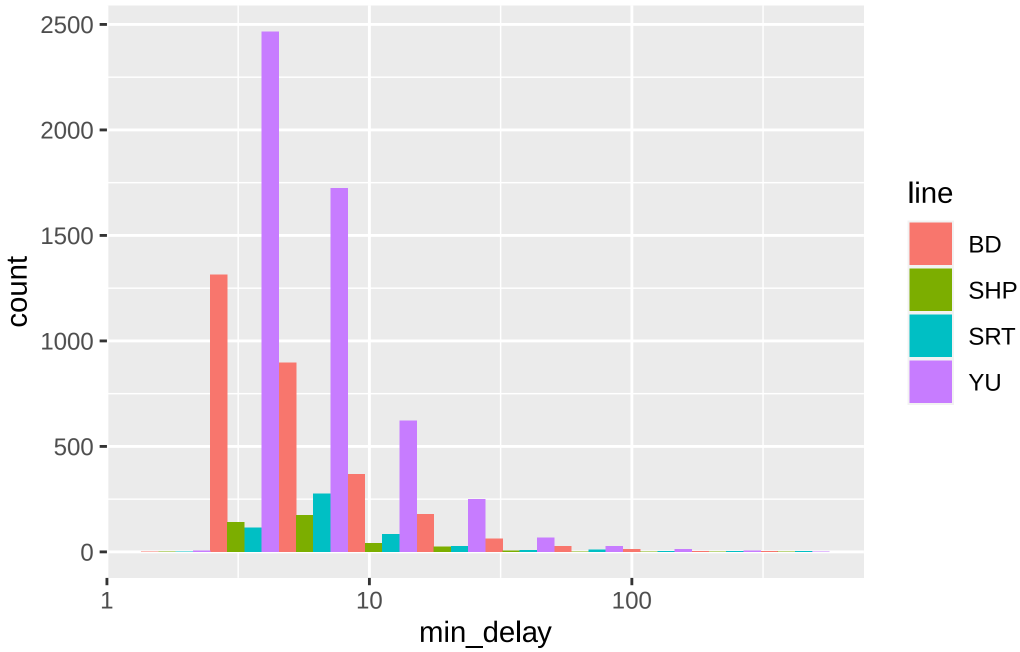


I switched to density above to look at the the distributions more comparably, but we should also be aware of differences in frequency, in particular, SHP and SRT have much smaller counts:

```
ggplot(data = delay_2022) +
  geom_histogram(aes(x = min_delay, fill = line), position = 'dodge', bins = 10) +
  scale_x_log10()
```

```
Warning: Transformation introduced infinite values in continuous x-axis
```

```
Warning: Removed 10500 rows containing non-finite values (stat_bin).
```

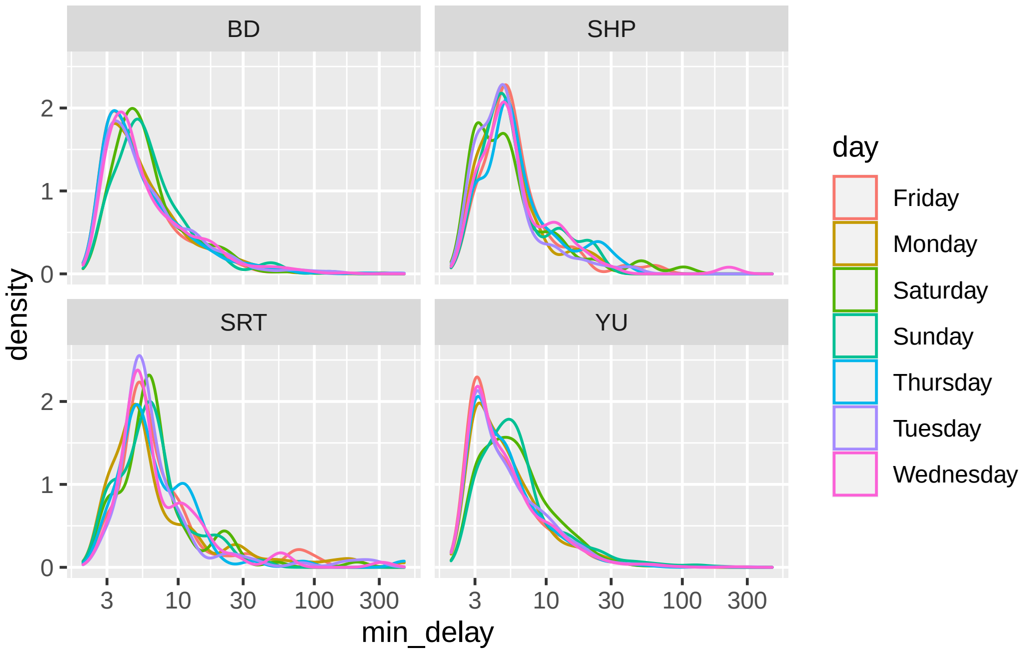


If you want to group by more than one variable, facets are good:

```
ggplot(data = delay_2022) +
  geom_density(aes(x = min_delay, color = day), bw = .08) +
  scale_x_log10() +
  facet_wrap(~line)
```

```
Warning: Transformation introduced infinite values in continuous x-axis
```

```
Warning: Removed 10500 rows containing non-finite values (stat_density).
```



Side note: the station names are a mess. Try and clean up the station names a bit by taking just the first word (or, the first two if it starts with "ST"):

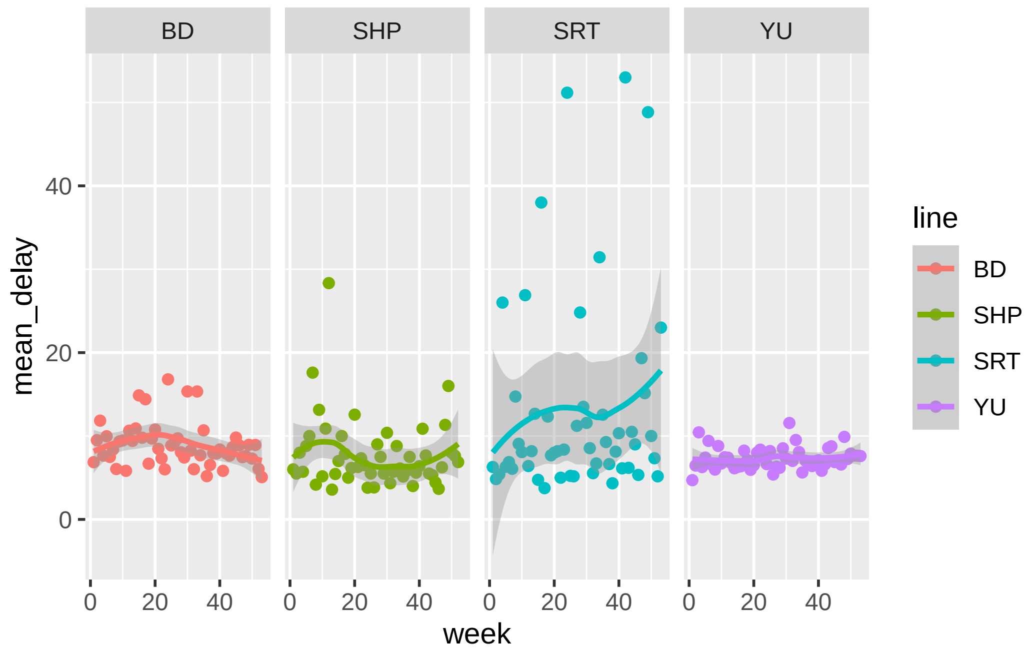
```
delay_2022 <- delay_2022 |>
  mutate(station_clean = ifelse(str_starts(station, "ST"), word(station, 1,2), word(statio
```

3.3 Visualizing time series

Daily plot is messy (you can check for yourself). Let's look by week to see if there's any seasonality. The `lubridate` package has lots of helpful functions that deal with date variables. First, mean delay (of those that were delayed more than 0 mins):

```
delay_2022 |>
  filter(min_delay>0) |>
  mutate(week = week(date)) |>
  group_by(week, line) |>
  summarise(mean_delay = mean(min_delay)) |>
  ggplot(aes(week, mean_delay, color = line)) +
  geom_point() +
  geom_smooth() +
  facet_grid(~line)
```

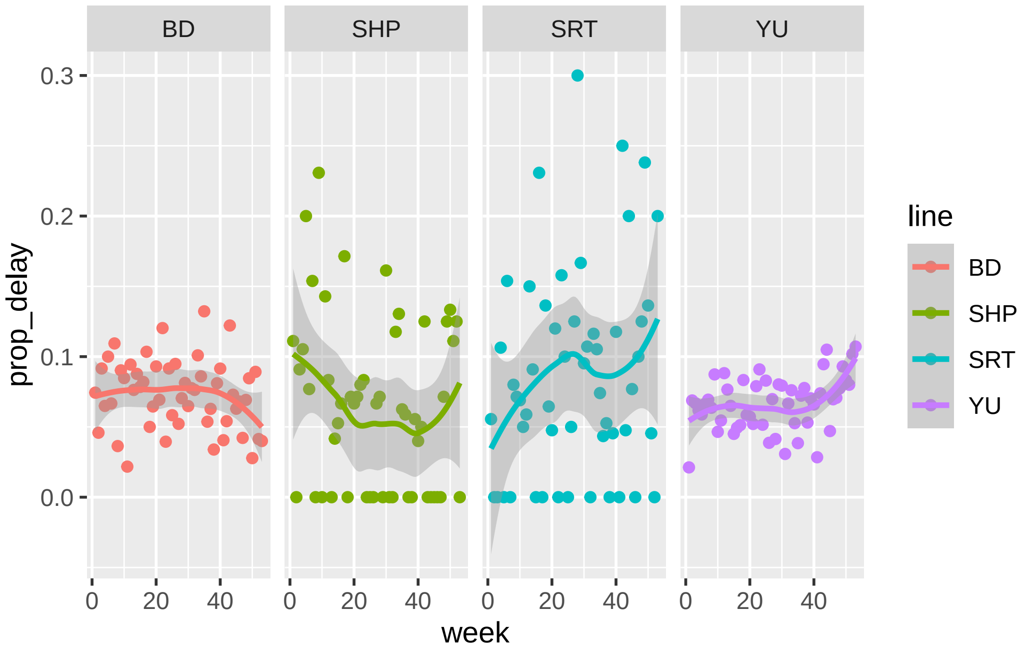
```
`summarise()` has grouped output by 'week'. You can override using the
`.groups` argument.
`geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



What about proportion of delays that were greater than 10 mins?

```
delay_2022 |>
  mutate(week = week(date)) |>
  group_by(week, line) |>
  summarise(prop_delay = sum(min_delay>10)/n()) |>
  ggplot(aes(week, prop_delay, color = line)) +
  geom_point() +
  geom_smooth() +
  facet_grid(~line)
```

```
`summarise()` has grouped output by 'week'. You can override using the
`.groups` argument.
`geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



3.4 Visualizing relationships

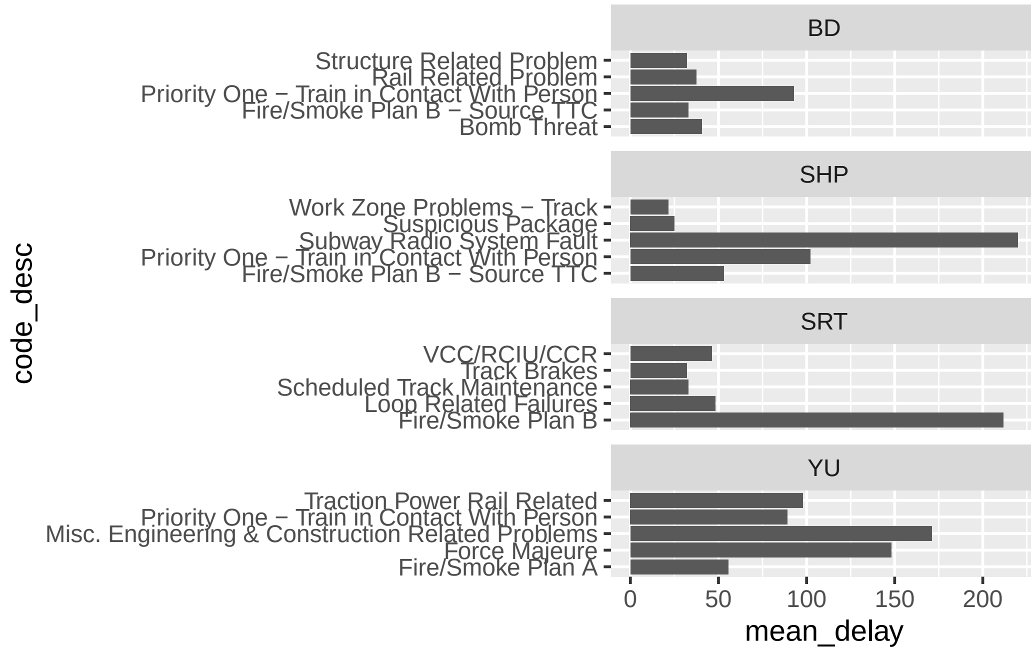
Note that **scatter plots** are a good precursor to modeling, to visualize relationships between continuous variables. Nothing obvious to plot here, but easy to do with `geom_point`.

Look at top five reasons for delay by station. Do they differ? Think about how this could be modeled.

```
delay_2022 |>
  group_by(line, code_desc) |>
  summarise(mean_delay = mean(min_delay)) |>
  arrange(-mean_delay) |>
  slice(1:5) |>
  ggplot(aes(x = code_desc,
             y = mean_delay)) +
  geom_col() +
  facet_wrap(vars(line),
             scales = "free_y",
             nrow = 4) +
  coord_flip()
```

```
`summarise()` has grouped output by 'line'. You can override using the
```

`.groups` argument.



3.5 PCA (additional)

Principal components analysis is a really powerful exploratory tool, particularly when you have a lot of variables. It allows you to pick up potential clusters and/or outliers that can help to inform model building.

Let's do a quick (and imperfect) example looking at types of delays by station.

The delay categories are a bit of a mess, and there's hundreds of them. As a simple start, let's just take the first word:

```
delay_2022 <- delay_2022 |>
  mutate(code_red = case_when(
    str_starts(code_desc, "No") ~ word(code_desc, 1, 2),
    str_starts(code_desc, "Operator") ~ word(code_desc, 1,2),
    TRUE ~ word(code_desc,1))
         )
```

Let's also just restrict the analysis to causes that happen at least 50 times over 2022 To do the PCA, the dataframe also needs to be switched to wide format:

```r
dwide <- delay_2022 |>
  group_by(line, station_clean) |>
  mutate(n_obs = n()) |>
  filter(n_obs>1) |>
  group_by(code_red) |>
  mutate(tot_delay = n()) |>
  arrange(tot_delay) |>
  filter(tot_delay>50) |>
  group_by(line, station_clean, code_red) |>
  summarise(n_delay = n()) |>
  pivot_wider(names_from = code_red, values_from = n_delay) |>
  mutate(
    across(everything(), ~ replace_na(.x, 0))
  )
```

```
`summarise()` has grouped output by 'line', 'station_clean'. You can override
using the `.groups` argument.
```

Do the PCA:

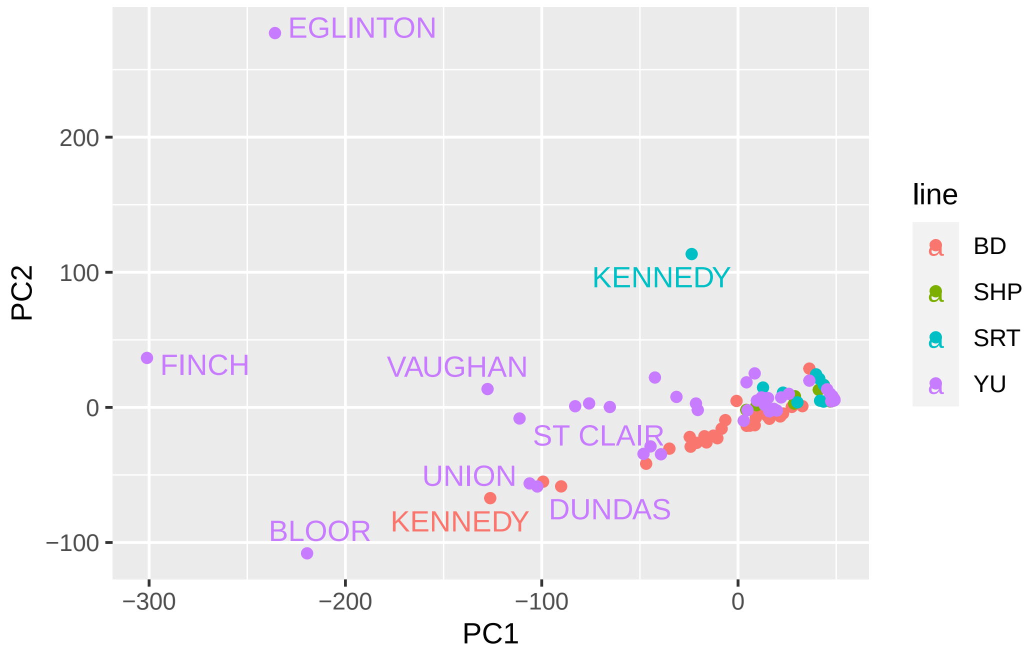
```r
delay_pca <- prcomp(dwide[,3:ncol(dwide)])

df_out <- as_tibble(delay_pca$x)
df_out <- bind_cols(dwide |> select(line, station_clean), df_out)
head(df_out)
```

```
# A tibble: 6 x 41
# Groups:   line, station_clean [6]
  line  station~1    PC1    PC2    PC3   PC4    PC5    PC6    PC7    PC8     PC9
  <chr> <chr>      <dbl>  <dbl>  <dbl> <dbl>  <dbl>  <dbl>  <dbl>  <dbl>   <dbl>
1 BD    BATHURST  -16.3  -24.3  -6.41  10.6  -3.15   5.36  -3.27  -8.68  -10.2
2 BD    BAY         8.50 -13.2  -6.28   8.05  0.804  0.401  6.09   0.311  -1.66
3 BD    BLOOR      36.3   28.7  34.2   14.7   9.70   7.70  -4.51   1.13   -2.31
4 BD    BLOOR-DA~  48.8    6.38 -0.483  1.44 -9.26   3.75  -0.671  0.388   0.189
5 BD    BROADVIEW -22.6  -26.1  -6.18  11.8   4.18  -2.66   4.48   7.57    8.27
6 BD    CASTLE     15.9   -8.44 -3.21   6.64 -3.65   0.366  0.624 -3.89   -2.99
# ... with 30 more variables: PC10 <dbl>, PC11 <dbl>, PC12 <dbl>, PC13 <dbl>,
#   PC14 <dbl>, PC15 <dbl>, PC16 <dbl>, PC17 <dbl>, PC18 <dbl>, PC19 <dbl>,
#   PC20 <dbl>, PC21 <dbl>, PC22 <dbl>, PC23 <dbl>, PC24 <dbl>, PC25 <dbl>,
#   PC26 <dbl>, PC27 <dbl>, PC28 <dbl>, PC29 <dbl>, PC30 <dbl>, PC31 <dbl>,
#   PC32 <dbl>, PC33 <dbl>, PC34 <dbl>, PC35 <dbl>, PC36 <dbl>, PC37 <dbl>,
```

```
#   PC38 <dbl>, PC39 <dbl>, and abbreviated variable name 1: station_clean
```

Plot the first two PCs, and label some outlying stations:

```
ggplot(df_out,aes(x=PC1,y=PC2,color=line )) + geom_point() + geom_text_repel(data = df_out
```



Plot the factor loadings. Some evidence of public v operator?

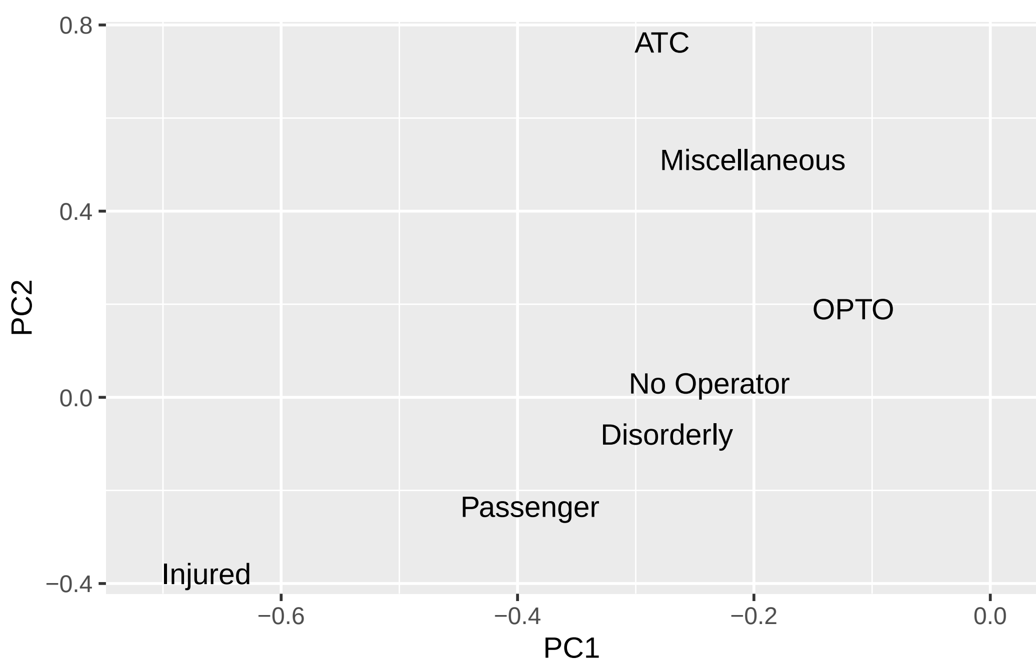
```
df_out_r <- as_tibble(delay_pca$rotation)
df_out_r$feature <- colnames(dwide[,3:ncol(dwide)])

df_out_r
```

```
# A tibble: 39 x 40
      PC1      PC2       PC3      PC4     PC5     PC6     PC7     PC8     PC9
    <dbl>    <dbl>     <dbl>    <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
 1 -0.127  -0.0381  -0.0174    0.0271  0.0387 -0.0425  0.122   -0.0238  0.159
 2 -0.305  -0.127   -0.0743    0.0461  0.103  -0.183   0.190   -0.647  -0.493
 3 -0.0530 -0.0113   0.0380    0.0382  0.0573 -0.0460 -0.0608  -0.116   0.250
 4 -0.0135 -0.0171  -0.0117   -0.00271 0.0454 -0.0367  0.0137   0.0191 -0.0712
 5 -0.0119 -0.00470  0.000218  0.00865 -0.0173 -0.0471 -0.0315  -0.0952  0.0587
```

```
 6 -0.0904 -0.0245   0.0512   -0.0164  -0.0338 -0.0658  0.0721   0.203   0.261
 7 -0.0161 -0.00185 -0.00131   0.00543  0.0134 -0.0361  0.0145   0.0371 -0.0392
 8 -0.712  -0.366   -0.0114    0.0895  -0.163   0.273  -0.435    0.211  -0.0519
 9 -0.232   0.463    0.700     0.263    0.380   0.0672 -0.0669   0.0107 -0.0663
10 -0.0402  0.00714  0.0999   -0.0387  -0.0922 -0.509   0.00130  0.303  -0.103
# ... with 29 more rows, and 31 more variables: PC10 <dbl>, PC11 <dbl>,
#   PC12 <dbl>, PC13 <dbl>, PC14 <dbl>, PC15 <dbl>, PC16 <dbl>, PC17 <dbl>,
#   PC18 <dbl>, PC19 <dbl>, PC20 <dbl>, PC21 <dbl>, PC22 <dbl>, PC23 <dbl>,
#   PC24 <dbl>, PC25 <dbl>, PC26 <dbl>, PC27 <dbl>, PC28 <dbl>, PC29 <dbl>,
#   PC30 <dbl>, PC31 <dbl>, PC32 <dbl>, PC33 <dbl>, PC34 <dbl>, PC35 <dbl>,
#   PC36 <dbl>, PC37 <dbl>, PC38 <dbl>, PC39 <dbl>, feature <chr>
```

```r
ggplot(df_out_r,aes(x=PC1,y=PC2,label=feature )) + geom_text_repel()
```

```
Warning: ggrepel: 32 unlabeled data points (too many overlaps). Consider
increasing max.overlaps
```



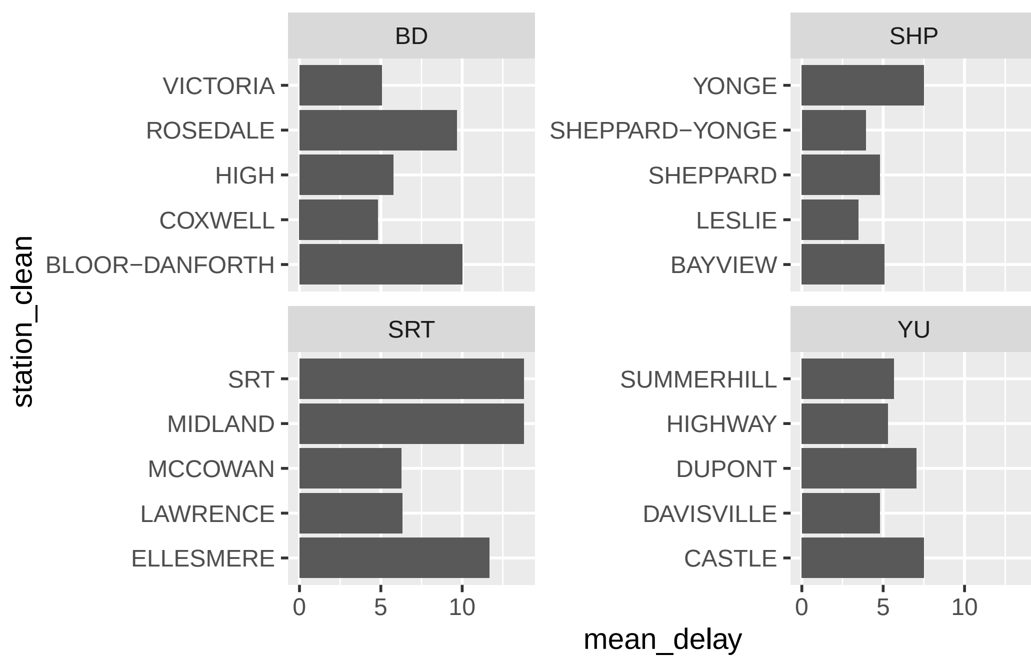
4 Lab Exercises

To be handed in via submission of quarto file (and rendered pdf) to GitHub.

1. Using the `delay_2022` data, plot the five stations with the highest mean delays. Facet the graph by `line`

```
delay_2022 |>
  group_by(line, station_clean) |>
  summarise(mean_delay = mean(min_delay), n_obs = n()) |>
  filter(n_obs>1) |>
  arrange(line, -mean_delay) |>
  slice(1:5) |>
  ggplot(aes(station_clean, mean_delay)) +
  geom_col() +
  coord_flip() +
  facet_wrap(~line, scales = "free_y")
```

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



2. Using the `opendatatoronto` package, download the data on mayoral campaign contributions for 2014. Hints:

   - find the ID code you need for the package you need by searching for 'campaign' in the `all_data` tibble above

- you will then need to `list_package_resources` to get ID for the data file
- note: the 2014 file you will get from `get_resource` has a bunch of different campaign contributions, so just keep the data that relates to the Mayor election

::: {.cell}

```
list_package_resources("f6651a40-2f52-46fc-9e04-b760c16edd5c")
```

::: {.cell-output .cell-output-stdout} # A tibble: 2 x 4 name format last_mod~1 <chr> <chr> <chr> <date> 1 campaign-contributions-2014-data 5b230e92-0a22-4a15-9~ ZIP 2019-07-23 2 campaign-contributions-2014-readme-xls aaf736f4-7468-4bda-9~ XLS 2019-07-23 # ... with abbreviated variable name 1: last_modified
:::

```
all_campaigns <- get_resource("5b230e92-0a22-4a15-9572-0b19cc222985")
```

::: {.cell-output .cell-output-stderr} "' New names: New names: New names: New names: New names: New names: New names:

- '->...2'
- '->...3'

```
:::

```{.r .cell-code}
df <- all_campaigns[[2]]
```

:::

3. Clean up the data format (fixing the parsing issue and standardizing the column names using `janitor`)

```
df <- df |>
  janitor::row_to_names(1) |>
  janitor::clean_names()
```

4. Summarize the variables in the dataset. Are there missing values, and if so, should we be worried about them? Is every variable in the format it should be? If not, create new variable(s) that are in the right format.

```
skim(df)
```
```

Table 5: Data summary

Name	df
Number of rows	10199
Number of columns	13
Column type frequency:	
character	13
Group variables	None

Variable type: character

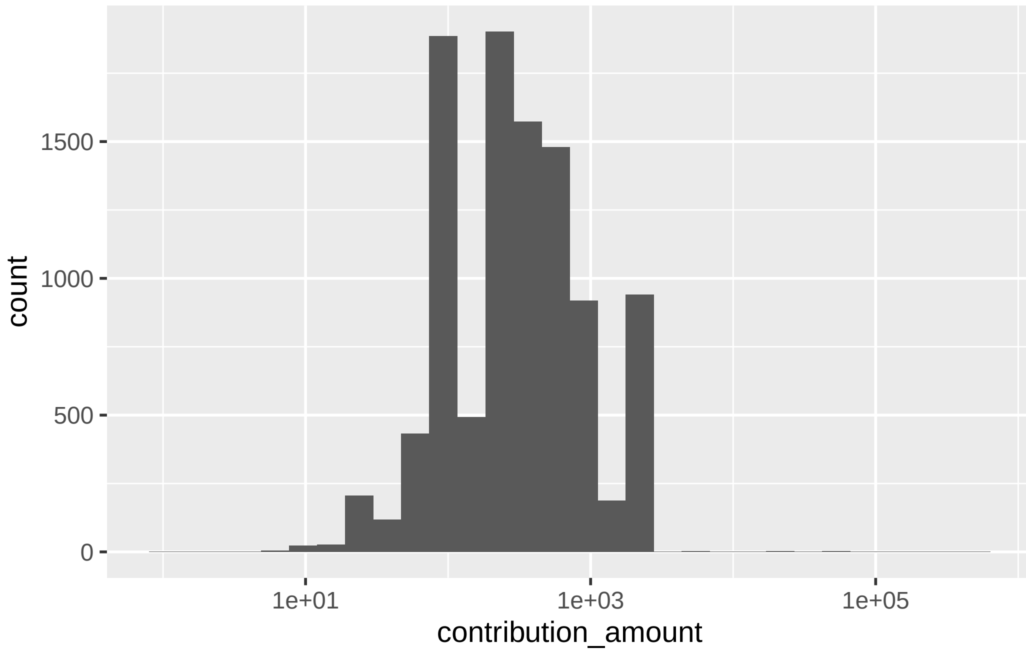
skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
contributors_name	0	1	4	31	0	7545	0
contributors_address	10197	0	24	26	0	2	0
contributors_postal_code	0	1	7	7	0	5284	0
contribution_amount	0	1	1	18	0	209	0
contribution_type_desc	0	1	8	14	0	2	0
goods_or_service_desc	10188	0	11	40	0	9	0
contributor_type_desc	0	1	10	11	0	2	0
relationship_to_candidate	10166	0	6	9	0	2	0
president_business_manager	10197	0	13	16	0	2	0
authorized_representative	10197	0	13	16	0	2	0
candidate	0	1	9	18	0	27	0
office	0	1	5	5	0	1	0
ward	10199	0	NA	NA	0	0	0

```
df <- df |>
  mutate(contribution_amount = as.numeric(contribution_amount))
```

5. Visually explore the distribution of values of the contributions. What contributions are notable outliers? Do they share a similar characteristic(s)? It may be useful to plot the distribution of contributions without these outliers to get a better sense of the majority of the data.

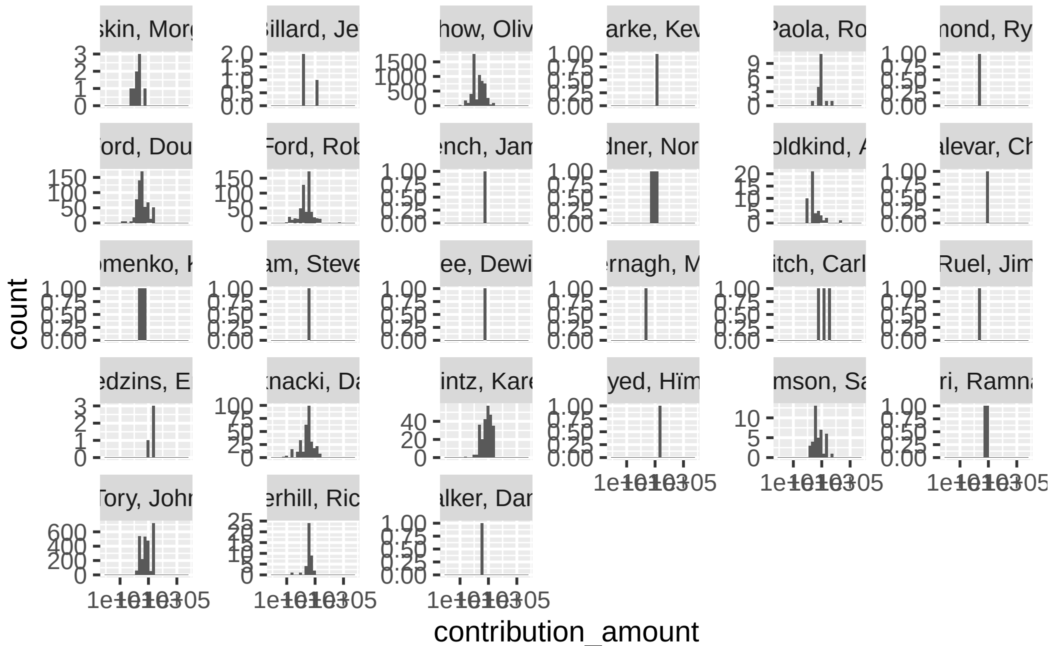
```
df |>
  ggplot(aes(contribution_amount)) + geom_histogram() + scale_x_log10()
```

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




```
df |>
  ggplot(aes(contribution_amount)) + geom_histogram() + scale_x_log10() + facet_wrap(~cand
```

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




```
# The big outliers are from Fords to Fords
```

6. List the top five candidates in each of these categories:

   - total contributions
   - mean contribution
   - number of contributions

```
# total contributions
df |>
  group_by(candidate) |>
  summarise(total_contr = sum(contribution_amount)) |>
  arrange(-total_contr)
```

```
# A tibble: 27 x 2
   candidate          total_contr
   <chr>                    <dbl>
 1 Tory, John            2767869.
 2 Chow, Olivia          1638266.
 3 Ford, Doug             889897.
 4 Ford, Rob              387648.
 5 Stintz, Karen          242805
```

```
 6 Soknacki, David       132431
 7 Goldkind, Ari          41125.
 8 Thomson, Sarah         34628.
 9 Di Paola, Rocco        21126
10 Underhill, Richard     15660
# ... with 17 more rows
```

```
# mean contributions
df |>
  group_by(candidate) |>
  summarise(mean_contr = mean(contribution_amount)) |>
  arrange(-mean_contr)
```

```
# A tibble: 27 x 2
   candidate          mean_contr
   <chr>                   <dbl>
 1 Sniedzins, Erwin        2025
 2 Syed, Hïmy              2018
 3 Ritch, Carlie           1887.
 4 Ford, Doug              1456.
 5 Clarke, Kevin           1200
 6 Di Paola, Rocco         1174.
 7 Tory, John              1064.
 8 Gardner, Norman         1000
 9 Stintz, Karen            995.
10 Kalevar, Chai            900
# ... with 17 more rows
```

```
# number
df |>
  group_by(candidate) |>
  tally() |>
  arrange(-n)
```

```
# A tibble: 27 x 2
   candidate               n
   <chr>               <int>
 1 Chow, Olivia         5708
 2 Tory, John           2602
 3 Ford, Doug            611
```

```
 4 Ford, Rob            538
 5 Soknacki, David      314
 6 Stintz, Karen        244
 7 Goldkind, Ari         47
 8 Underhill, Richard    41
 9 Thomson, Sarah        40
10 Di Paola, Rocco       18
# ... with 17 more rows
```

7. Repeat 6 but without contributions from the candidates themselves.

```r
df_not_to_self <- df |>
  filter(contributors_name!=candidate)


df_not_to_self |>
  group_by(candidate) |>
  summarise(total_contr = sum(contribution_amount)) |>
  arrange(-total_contr)
```

```
# A tibble: 17 x 2
   candidate          total_contr
   <chr>                    <dbl>
 1 Tory, John            2765369.
 2 Chow, Olivia          1634766.
 3 Ford, Doug             331173.
 4 Stintz, Karen          242805
 5 Ford, Rob              174510.
 6 Soknacki, David        132431
 7 Thomson, Sarah          27702.
 8 Goldkind, Ari           17501
 9 Underhill, Richard      15660
10 Di Paola, Rocco         15126
11 Ritch, Carlie            5660
12 Sniedzins, Erwin         5600
13 Gardner, Norman          3000
14 Baskin, Morgan           1550
15 Billard, Jeff            1486.
16 Tiwari, Ramnarine        1000
17 Lam, Steven               300
```

```r
# mean contributions
df_not_to_self |>
  group_by(candidate) |>
  summarise(mean_contr = mean(contribution_amount)) |>
  arrange(-mean_contr)
```

```
# A tibble: 17 x 2
   candidate           mean_contr
   <chr>                    <dbl>
 1 Ritch, Carlie            1887.
 2 Sniedzins, Erwin         1867.
 3 Tory, John               1063.
 4 Gardner, Norman          1000
 5 Tiwari, Ramnarine        1000
 6 Stintz, Karen             995.
 7 Di Paola, Rocco           890.
 8 Thomson, Sarah            729.
 9 Ford, Doug                545.
10 Billard, Jeff             496.
11 Soknacki, David           422.
12 Underhill, Richard        382.
13 Goldkind, Ari             380.
14 Ford, Rob                 329.
15 Lam, Steven               300
16 Chow, Olivia              286.
17 Baskin, Morgan            194.
```

```r
# number
df_not_to_self |>
  group_by(candidate) |>
  tally() |>
  arrange(-n)
```

```
# A tibble: 17 x 2
   candidate              n
   <chr>              <int>
 1 Chow, Olivia        5706
 2 Tory, John          2601
 3 Ford, Doug           608
 4 Ford, Rob            531
```

```
 5 Soknacki, David      314
 6 Stintz, Karen        244
 7 Goldkind, Ari         46
 8 Underhill, Richard    41
 9 Thomson, Sarah        38
10 Di Paola, Rocco       17
11 Baskin, Morgan         8
12 Billard, Jeff          3
13 Gardner, Norman        3
14 Ritch, Carlie          3
15 Sniedzins, Erwin       3
16 Lam, Steven            1
17 Tiwari, Ramnarine      1
```

8. How many contributors gave money to more than one candidate?

```r
df |>
  group_by(contributors_name) |>
  distinct(candidate) |>
  tally() |>
  filter(n>1) |>
  nrow()
```

```
[1] 184
```

```r
# OR

df |>
  group_by(contributors_name, candidate) |>
  tally() |>
  group_by(contributors_name) |>
  tally() |>
  filter(n>1) |> nrow()
```

```
[1] 184
```