MTA Exploration

Andrew Leung

3/31/2023

COVID, NYC, and the MTA

The New York City subway is one of the busiest subway systems in the world, with trains regularly carrying upwards of 1.6 billion passengers yearly from 2016-2019. The COVID-19 pandemic changed this, with ridership dropping by 62% from 2020 to 2021. While a majority of New Yorkers were required to stay home, essential workers including hospital staff, sanitation crews, restaurant workers, and many others continued to ride the subways to keep the City running. In this investigation, I want to visualize the change in NYC ridership before, during, and after the peaks of the pandemic.

Data

For this project, I sourced MTA turnstile usage data from the MTA Open Data Catalog and NYC COVID data from NYC Health's Github page.

Load in Libraries

```
library(tidyverse)
library(janitor)
library(ggplot2)
library(lubridate)
library(dplyr)
```

Load in MTA Turnstile Data from 2019-2021

```
tud19 <- read_csv(file = "data/Turnstile_Usage_Data__2019.csv")
tud20 <- read_csv(file = "data/Turnstile_Usage_Data__2020.csv")
tud21 <- read_csv(file = "data/Turnstile_Usage_Data__2021.csv")
```

Cleaning Time

Before we get cleaning, let's take a look at the data:

```
head(tud19)
```

```
## # A tibble: 6 x 11
##   `C/A` Unit  SCP    Station Line ~1 Divis~2 Date  Time   Descr~3 Entries  Exits
##   <chr> <chr> <chr>  <chr>   <chr>   <chr>   <chr> <time> <chr>     <dbl>  <dbl>
## 1 A033  R170  02-00~ 14 ST-~ LNQR45~ BMT     12/2~ 00'00" REGULAR  1.75e7 7.03e6
## 2 A033  R170  02-00~ 14 ST-~ LNQR45~ BMT     12/2~ 00'00" REGULAR  1.50e7 1.46e7
## 3 A033  R170  02-06~ 14 ST-~ LNQR45~ BMT     12/2~ 00'00" REGULAR  7.69e5 5.59e5
## 4 A033  R170  02-00~ 14 ST-~ LNQR45~ BMT     12/2~ 00'00" REGULAR  7.19e6 8.42e6
## 5 A033  R170  02-06~ 14 ST-~ LNQR45~ BMT     12/2~ 00'00" REGULAR  7.10e7 2.09e7
## 6 A033  R170  02-00~ 14 ST-~ LNQR45~ BMT     12/2~ 00'00" REGULAR  2.43e6 2.92e6
## # ... with abbreviated variable names 1: `Line Name`, 2: Division,
## #   3: Description
```

Okay, interesting. The data doesn't look as straightforward as one would expect. What does `C/A`, `Unit`, and `SCP` mean? Why are `Entries` and `Exits` sometimes in the 10s of millions? What do the `date` and `time` columns signify? Thankfully, there is a large community of public transportation data fans like myself who can answer some of those questions.

An SCP, or "subunit channel position," represents the address of a specific turnstile. These specific turnstiles are usually found in groups known as a CA, or "control area." Each CA is then grouped in a station known as a "remote unit", or "unit" in this data set. They are also called "remotes", as we'll see later on. Each of these stations are then divided into divisions, which represent the transit companies that used to manage each line. The three major companies were the Brooklyn-Manhattan Transit Company (BMT), Interborough Rapid Transit Company (IRT), and the Independent Subway System (IND), which merged in 1940 and became the NYC Subway System that we know today.

The other divisions in this data set are the PATH Train (PTH), which travels from NYC to New Jersey, the Roosevelt Island Tramway (RIT), which travels from upper Manhattan to Roosevelt Island, and the Staten Island Railway (SRT), which services Staten Island via the Staten Island Ferry. Since most of NYC's ridership is in the NYC Subway System, we will be excluding these divisions.

Entries and exits are a misleading metric because the number is actually a cumulative value that has been counting since the device was installed. So, to get the true count of how many entries or exits there were at a turnstile at a given time (`time` describes when the data was recorded, which is usually in 4-hr intervals), you need to take the difference of the current and previous value.

With these definitions in mind, let's start the process of cleaning up this data.

First, let's filter out the PTH, RIT, and SRT from the dataset.

```
tud19_clean <- tud19 %>%
  clean_names() %>%
  filter(division == "BMT" | division == "IND" | division == "IRT" )
```

Now let's take a look at the data:

```
head(tud19_clean)
```

```
## # A tibble: 6 x 11
##   c_a   unit  scp    station line_~1 divis~2 date  time   descr~3 entries  exits
##   <chr> <chr> <chr>  <chr>   <chr>   <chr>   <chr> <time> <chr>     <dbl>  <dbl>
## 1 A033  R170  02-00~ 14 ST-~ LNQR45~ BMT     12/2~ 00'00" REGULAR  1.75e7 7.03e6
## 2 A033  R170  02-00~ 14 ST-~ LNQR45~ BMT     12/2~ 00'00" REGULAR  1.50e7 1.46e7
```

```
## 3 A033  R170  02-06~ 14 ST-~ LNQR45~ BMT     12/2~ 00'00" REGULAR  7.69e5 5.59e5
## 4 A033  R170  02-00~ 14 ST-~ LNQR45~ BMT     12/2~ 00'00" REGULAR  7.19e6 8.42e6
## 5 A033  R170  02-06~ 14 ST-~ LNQR45~ BMT     12/2~ 00'00" REGULAR  7.10e7 2.09e7
## 6 A033  R170  02-00~ 14 ST-~ LNQR45~ BMT     12/2~ 00'00" REGULAR  2.43e6 2.92e6
## # ... with abbreviated variable names 1: line_name, 2: division, 3: description
```

Perfect! Now we have data the only pertains to the NYC Subway System. Next, we have to deal with these turnstiles. Because a specific turnstile can only be addressed by its `scp`, `c_a`, and `unit`, let's make an identifier that combines each of these columns so that we can sort by it. In addition to this identifier, we also want to make an ID for each row, since every 4 hours is a new observation for each turnstile. To do this, we will combine the turnstile ID with the date and time.

```
tud19_clean <- tud19_clean %>%
  # Combine the three columns together to make an ID
  mutate(turnstile_id = paste0(unit, c_a, scp),
         # Combine date and time to make a time stamp
         timestamp = paste(date, time,
                           sep = " "),
         # With the turnstile ID and time stamp, we can now make a unique
         # ID for every observation
         observation_id = paste(turnstile_id, timestamp,
                                sep = " "))

col_order <- c("observation_id",
               "turnstile_id",
               "station",
               "timestamp",
               "entries",
               "exits",
               "line_name",
               "unit",
               "c_a",
               "scp",
               "division",
               "date",
               "time",
               "description")

tud19_clean <- tud19_clean[, col_order]

head(tud19_clean)
```

```
## # A tibble: 6 x 14
##   observation~1 turns~2 station times~3 entries  exits line_~4 unit  c_a   scp
##   <chr>         <chr>   <chr>   <chr>     <dbl>  <dbl> <chr>   <chr> <chr> <chr>
## 1 R170A03302-0~ R170A0~ 14 ST-~ 12/27/~  1.75e7 7.03e6 LNQR45~ R170  A033  02-0~
## 2 R170A03302-0~ R170A0~ 14 ST-~ 12/27/~  1.50e7 1.46e7 LNQR45~ R170  A033  02-0~
## 3 R170A03302-0~ R170A0~ 14 ST-~ 12/27/~  7.69e5 5.59e5 LNQR45~ R170  A033  02-0~
## 4 R170A03302-0~ R170A0~ 14 ST-~ 12/27/~  7.19e6 8.42e6 LNQR45~ R170  A033  02-0~
## 5 R170A03302-0~ R170A0~ 14 ST-~ 12/27/~  7.10e7 2.09e7 LNQR45~ R170  A033  02-0~
## 6 R170A03302-0~ R170A0~ 14 ST-~ 12/27/~  2.43e6 2.92e6 LNQR45~ R170  A033  02-0~
## # ... with 4 more variables: division <chr>, date <chr>, time <time>,
## #   description <chr>, and abbreviated variable names 1: observation_id,
## #   2: turnstile_id, 3: timestamp, 4: line_name
```

Creating a unique ID for each observation will make this data set more versatile to use for future visualizations. For now, I want to aggregate the number of entries and exits for each turnstile in each station per day.

To do this, I will use the `lag` function to take the difference between the current observation and the previous one. However, there are a couple wrinkles. First, not every day is recorded. Second, when transitioning from one turnstile to another, the date resets from 12/31 to 1/1. Thankfully, we can solve both of these issues by performing the `lag` calculation only if the difference the date of the observation is either 0 (e.g. 1/1/19 at 4am and 1/1/19 at 8am) or 1 (e.g. 1/1/19 at 8am and 1/2/19 at 1/2/19). A difference that is negative would indicate a switch in machines, since the date would reset from 12/31 to 1/1. A difference that is greater than 1 indicates that a day was skipped in the recording process. In both of these cases, the count is set to 0.

This is not the end of our problems unfortunately. Sometimes, the turnstiles simply just start counting down instead of up. Many in the MTA data community believe that when there is maintenance, workers may sometimes reverse the counter. Since this is the case, we can just take the absolute value of the number, as long as it is not less than -10,000.

```
# Convert date column into class Date
tud19_clean$date <- gsub('/', '-', tud19_clean$date)
tud19_clean$date <- mdy(tud19_clean$date)

# Organize the data frame by turnstile and date and
# add in column with the differences in dates between each observation
tud19_clean <- tud19_clean %>%
  arrange(turnstile_id, timestamp) %>%
  mutate(yday = yday(date),
         yday_diff = yday - dplyr::lag(yday, n = 1))

# Calculate turnstile entries and exits by taking the difference between observations
tud19_clean <- tud19_clean %>%
  mutate(net_entries = dplyr::if_else(yday_diff < 0 | yday_diff > 1,
                                      0,
                                      entries - dplyr::lag(entries, n = 1))) %>%
  mutate(net_exits = dplyr::if_else(yday_diff < 0 | yday_diff > 1,
                                    0,
                                    exits - dplyr::lag(exits, n = 1)))

# The lag function makes the first entry "NA." So I want to make sure those values are changed to 0.
tud19_clean <- tud19_clean %>%
  mutate(net_entries = coalesce(net_entries, 0),
         yday_diff = coalesce(yday_diff, 0),
         net_exits = coalesce(net_exits, 0))

# Change the column order so the data frame can be read more logically
col_order <- c("observation_id",
               "turnstile_id",
               "station",
               "timestamp",
               "entries",
               "net_entries",
               "exits",
               "net_exits",
               "date",
               "time",
```

```
               "yday",
               "yday_diff",
               "line_name",
               "unit",
               "c_a",
               "scp",
               "division",
               "description")

tud19_clean <- tud19_clean[, col_order]

# Turnstiles with extra large or small numbers set to 0
# Take absolute value of turnstiles with reversed counters
tud19_clean <- tud19_clean %>%
  mutate(
  net_entries = case_when(
    net_entries > 10000 | net_entries < -10000 ~ 0,
    net_entries < 0 | net_entries > -10000 ~ abs(net_entries),
    TRUE ~ net_entries
  ),
  net_exits = case_when(
    net_exits > 10000 | net_exits < -10000 ~ 0,
    net_exits < 0 | net_exits > -10000 ~ abs(net_exits),
    TRUE ~ net_exits))

# Aggregate the number of entries and exits by turnstile.
tud19_sum <- tud19_clean %>%
  group_by(turnstile_id,
           date,
           station,
           line_name,
           unit,
           c_a,
           scp) %>%
  summarise(entry_total = sum(net_entries),
            exit_total = sum(net_exits))

head(tud19_clean)
```

```
## # A tibble: 6 x 18
##   observatio~1 turns~2 station times~3 entries net_e~4  exits net_e~5 date
##   <chr>        <chr>   <chr>   <chr>     <dbl>   <dbl>  <dbl>   <dbl> <date>
## 1 R001A05801-~ R001A0~ WHITEH~ 01/01/~ 1495467       0 3.47e6       0 2019-01-01
## 2 R001A05801-~ R001A0~ WHITEH~ 01/01/~ 1495484      17 3.47e6     104 2019-01-01
## 3 R001A05801-~ R001A0~ WHITEH~ 01/01/~ 1495488       4 3.47e6      40 2019-01-01
## 4 R001A05801-~ R001A0~ WHITEH~ 01/01/~ 1495513      25 3.47e6     209 2019-01-01
## 5 R001A05801-~ R001A0~ WHITEH~ 01/01/~ 1495598      85 3.47e6     313 2019-01-01
## 6 R001A05801-~ R001A0~ WHITEH~ 01/01/~ 1495716     118 3.47e6     171 2019-01-01
## # ... with 9 more variables: time <time>, yday <dbl>, yday_diff <dbl>,
## #   line_name <chr>, unit <chr>, c_a <chr>, scp <chr>, division <chr>,
## #   description <chr>, and abbreviated variable names 1: observation_id,
## #   2: turnstile_id, 3: timestamp, 4: net_entries, 5: net_exits
```

Great! Now we have a data frame containing the total number of entries and exits for each turnstile at each

station for (almost) every day of 2019. Now, I want to aggregate the number of entries and exits per month at each station. However, as Chris Whong discovered in his "Taming" of the MTA turnstile data, there are some discrepancies. To show them, let's look at the 14th Street Stations.

```
complexes <- tud19 %>%
  filter(Station == "14 ST" | Station == "6 AV") %>%
  distinct(Unit,
           Station,
           .keep_all = TRUE)

head(complexes)
```

```
## # A tibble: 4 x 11
##   `C/A` Unit  SCP     Station Line ~1 Divis~2 Date  Time  Descr~3 Entries  Exits
##   <chr> <chr> <chr>   <chr>   <chr>   <chr>   <chr> <tim> <chr>     <dbl>  <dbl>
## 1 N078  R175  01-03-~ 14 ST   ACEL    IND     12/2~ 00:00 REGULAR  210834 6.48e4
## 2 H003  R163  01-00-~ 6 AV    FLM123  BMT     12/2~ 03:00 REGULAR 6566369 1.74e6
## 3 N510  R163  02-00-~ 14 ST   FLM123  IND     12/2~ 03:00 REGULAR  199853 1.17e5
## 4 R127  R105  00-00-~ 14 ST   123FLM  IRT     12/2~ 03:00 REGULAR 1251805 6.33e5
## # ... with abbreviated variable names 1: `Line Name`, 2: Division,
## #   3: Description
```

As you can see, there are two stations (6th Avenue and 14th Street) with the same remote unit (R163). There are also instances where the same station has multiple remote units. These are known as "station complexes", or stations "connected with a passageway inside fare control." In the `Turnstile_Usage_Data`, there is no way to distinguish a complex. Thankfully, again as Chris Whong already found, we can join a couple tables together from the MTA to get there.

The first is called `Stations`, which contains the complex IDs as well as the non-abbreviated names for each station. The second is called `Remote-Booth-Station`, which contains all of the remote IDs of each station. We can join these two tables by each station's complex ID; however, because the MTA's `Remote-Booth-Station` data set is outdated, we will need to create our own.

```
library(readxl)

# Thank you [@ martinctc]("https://gist.github.com/martinctc/56b3fb701a182f5b8dffceecd65b6d86")
# for the function! This function will be useful to keep the line name
# formatting consistent. (i.e. LMF123 -> 123FLM)
str_arrange <- function(x){
  x %>%
    stringr::str_split("") %>% # Split string into letters
    purrr::map(~sort(.) %>% paste(collapse = "")) %>% # Sort and re-combine
    as_vector() # Convert list into vector
}

# Read in the Remote-Booth-Station
remote_booth_station <- tud21 %>%
  clean_names()

remote_booth_station$line_name <- str_arrange(remote_booth_station$line_name)

remote_booth_station <- remote_booth_station %>%
  filter(division == "BMT" | division == "IND" | division == "IRT") %>%
  distinct(unit, station, line_name, division,
```

```
            .keep_all = TRUE) %>%
  select(-c(date, time, description, entries, exits))

remote_booth_station <- remote_booth_station %>%
  rename(booth = c_a)

# Read in the Station data set
mta_locations <- read_csv("data/station_locations_2022.csv") %>%
  clean_names()
```

Because there are no alike columns in both data frames, I will have to add a complex ID column manually into `remote_booth_station`. I would have changed all of the names in `remote_booth_station` into the better formatted version in `mta_locations` to join the two tables; however, because of the lack of consistent naming conventions, the manual addition will be easier.

```
library(writexl)

# If a new RBS file doesn't exist, create one to edit.
# Once edited, it will automatically overwrite the current RBS
if(!file.exists("/Users/andrew/Desktop/data-science/projects/mta-data-exploration/data/remote_booth_sta
  write_xlsx(remote_booth_station,
          path = "/Users/andrew/Desktop/data-science/projects/mta-data-exploration/data/remote_booth_st
} else {
  remote_booth_station <- read_excel("data/remote_booth_station_2021.xlsx")
}
```

Now let's join the new `remote_booth_station` to our `tud19_sum` data frame:

```
remote_booth_station$complex_id <- as.numeric(remote_booth_station$complex_id)


# First join the RBS data frame to the turnstile sum data frame
tud19_rbs_join <- left_join(tud19_sum,
                            remote_booth_station,
                            by = c("station" = "station",
                                   "unit" = "unit")) %>%
  select(-c(line_name.x,
            booth,
            scp.y))

# Next, join the location data to the previous joined data frame
tud19_location_join <- left_join(tud19_rbs_join,
                                 mta_locations,
                                 by = "complex_id") %>%
  distinct(turnstile_id, date, station,
           .keep_all = TRUE)

# Clean up column names / reorganize columns
tud19_location_join <- tud19_location_join %>%
  select(-c(division.y,
            daytime_routes,
            ada_direction_notes,
            ada_nb,
```

```
            ada_sb,
            capital_outage_nb,
            capital_outage_sb)) %>%
  rename(station_abbr = station,
         scp = scp.x,
         line_name = line_name.y,
         station_long = stop_name)

tud19_location_join <- tud19_location_join %>%
  rename(division = division.x,
         complex_lat = gtfs_latitude,
         complex_long = gtfs_longitude)
```

Now we can finally start visualizing! Let's see what the system-wide entry/exit data looks like

```
system_total_2019 <- tud19_location_join %>%
  group_by(date) %>%
  summarise(daily_entries = sum(entry_total),
            daily_exits = sum(exit_total))

system_2019_plot <- ggplot(data = system_total_2019,
                           aes(x = date)) +
  geom_line(aes(y = daily_entries, color = "daily_entries")) +
  geom_line(aes(y = daily_exits, color = "daily_exits")) +
  scale_x_date(date_breaks = "1 month", date_labels = "%b %Y") +
  scale_y_continuous(name = "Daily Entries / Exits", labels = scales::comma) +
  theme(legend.position = "bottom",
        legend.text = element_text(color = "grey20", size = 10),
        legend.title = element_text(color = "grey20", size = 10),
        axis.text.x = element_text(color = "grey20", size = 10),
        axis.title.x = element_text(color = "grey20", size = 10),
        axis.text.y = element_text(color = "grey20", size = 10),
        axis.title.y = element_text(color = "grey20", size = 10))

system_2019_plot
```

Great! Now we have a plot visualizing the number of entries and exits across the NYC subway system for the year 2019. One interesting thing to notice is how there seem to be more entries than exits. Intuitively, these number should be equal. While this could simply be chalked up to measurement errors (which, after dealing with this turnstile data is certainly possible), one more practical reason could be the use of exit doors. Especially during rush hour, exiting via single person turnstiles can be slow. So, some people open the emergency exit doors, which allow hundreds of people to exit quicker. Compound that over all NYC stations, and you have made up at least some of the difference show in the graph.

Now, let's repeat this cleaning process for the 2020 and 2021 data. First, the 2020 data:

```
tud20_clean <- tud20 %>%
  clean_names() %>%
  filter(division == "BMT" | division == "IND" | division == "IRT" ) %>%
  mutate(turnstile_id = paste0(unit, c_a, scp),
         # Combine date and time to make a time stamp
         timestamp = paste(date, time,
                           sep = " "),
         # With the turnstile ID and time stamp, we can now make a unique
         # ID for every observation
         observation_id = paste(turnstile_id, timestamp,
                                sep = " "))
```

```r
# Convert date column into class Date
tud20_clean$date <- gsub('/', '-', tud20_clean$date)
tud20_clean$date <- mdy(tud20_clean$date)

# Organize the data frame by turnstile and date and
# add in column with the differences in dates between each observation
tud20_clean <- tud20_clean %>%
  arrange(turnstile_id, timestamp) %>%
  mutate(yday = yday(date),
         yday_diff = yday - dplyr::lag(yday, n = 1))

# Calculate turnstile entries and exits by taking the difference between observations
tud20_clean <- tud20_clean %>%
  mutate(net_entries = dplyr::if_else(yday_diff < 0 | yday_diff > 1,
                                      0,
                                      entries - dplyr::lag(entries, n = 1))) %>%
  mutate(net_exits = dplyr::if_else(yday_diff < 0 | yday_diff > 1,
                                    0,
                                    exits - dplyr::lag(exits, n = 1)))

# The lag function makes the first entry "NA." So I want to make sure those values are changed to 0.
tud20_clean <- tud20_clean %>%
  mutate(net_entries = coalesce(net_entries, 0),
         yday_diff = coalesce(yday_diff, 0),
         net_exits = coalesce(net_exits, 0))

# Change the column order so the data frame can be read more logically
col_order <- c("observation_id",
               "turnstile_id",
               "station",
               "timestamp",
               "entries",
               "net_entries",
               "exits",
               "net_exits",
               "date",
               "time",
               "yday",
               "yday_diff",
               "line_name",
               "unit",
               "c_a",
               "scp",
               "division",
               "description")

tud20_clean <- tud20_clean[, col_order]

# Turnstiles with extra large or small numbers set to 0
# Take absolute value of turnstiles with reversed counters
tud20_clean <- tud20_clean %>%
  mutate(
  net_entries = case_when(
```

```
    net_entries > 10000 | net_entries < -10000 ~ 0,
    net_entries < 0 | net_entries > -10000 ~ abs(net_entries),
    TRUE ~ net_entries
  ),
  net_exits = case_when(
    net_exits > 10000 | net_exits < -10000 ~ 0,
    net_exits < 0 | net_exits > -10000 ~ abs(net_exits),
    TRUE ~ net_exits))

# Aggregate the number of entries and exits by turnstile.
tud20_sum <- tud20_clean %>%
  group_by(turnstile_id,
           date,
           station,
           line_name,
           unit,
           c_a,
           scp) %>%
  summarise(entry_total = sum(net_entries),
            exit_total = sum(net_exits))

tud20_rbs_join <- left_join(tud20_sum,
                            remote_booth_station,
                            by = c("station" = "station",
                                   "unit" = "unit")) %>%
  select(-c(line_name.x,
            booth,
            scp.y))

# Next, join the location data to the previous joined data frame
tud20_location_join <- left_join(tud20_rbs_join,
                                 mta_locations,
                                 by = "complex_id") %>%
  distinct(turnstile_id, date, station,
           .keep_all = TRUE)

# Clean up column names / reorganize columns
tud20_location_join <- tud20_location_join %>%
  select(-c(division.y,
            daytime_routes,
            ada_direction_notes,
            ada_nb,
            ada_sb,
            capital_outage_nb,
            capital_outage_sb)) %>%
  rename(station_abbr = station,
         scp = scp.x,
         line_name = line_name.y,
         station_long = stop_name)

tud20_location_join <- tud20_location_join %>%
  rename(division = division.x,
         complex_lat = gtfs_latitude,
```

```
         complex_long = gtfs_longitude)
```

Next, the 2021 data:

```
tud21_clean <- tud21 %>%
  clean_names() %>%
  filter(division == "BMT" | division == "IND" | division == "IRT" ) %>%
  mutate(turnstile_id = paste0(unit, c_a, scp),
         # Combine date and time to make a time stamp
         timestamp = paste(date, time,
                           sep = " "),
         # With the turnstile ID and time stamp, we can now make a unique
         # ID for every observation
         observation_id = paste(turnstile_id, timestamp,
                                sep = " "))

# Convert date column into class Date
tud21_clean$date <- gsub('/', '-', tud21_clean$date)
tud21_clean$date <- mdy(tud21_clean$date)

# Organize the data frame by turnstile and date and
# add in column with the differences in dates between each observation
tud21_clean <- tud21_clean %>%
  arrange(turnstile_id, timestamp) %>%
  mutate(yday = yday(date),
         yday_diff = yday - dplyr::lag(yday, n = 1))

# Calculate turnstile entries and exits by taking the difference between observations
tud21_clean <- tud21_clean %>%
  mutate(net_entries = dplyr::if_else(yday_diff < 0 | yday_diff > 1,
                                      0,
                                      entries - dplyr::lag(entries, n = 1))) %>%
  mutate(net_exits = dplyr::if_else(yday_diff < 0 | yday_diff > 1,
                                      0,
                                      exits - dplyr::lag(exits, n = 1)))

# The lag function makes the first entry "NA." So I want to make sure those values are changed to 0.
tud21_clean <- tud21_clean %>%
  mutate(net_entries = coalesce(net_entries, 0),
         yday_diff = coalesce(yday_diff, 0),
         net_exits = coalesce(net_exits, 0))

# Change the column order so the data frame can be read more logically
col_order <- c("observation_id",
               "turnstile_id",
               "station",
               "timestamp",
               "entries",
               "net_entries",
               "exits",
               "net_exits",
               "date",
               "time",
```

```
                "yday",
                "yday_diff",
                "line_name",
                "unit",
                "c_a",
                "scp",
                "division",
                "description")

tud21_clean <- tud21_clean[, col_order]

# Turnstiles with extra large or small numbers set to 0
# Take absolute value of turnstiles with reversed counters
tud21_clean <- tud21_clean %>%
  mutate(
  net_entries = case_when(
    net_entries > 10000 | net_entries < -10000 ~ 0,
    net_entries < 0 | net_entries > -10000 ~ abs(net_entries),
    TRUE ~ net_entries
  ),
  net_exits = case_when(
    net_exits > 10000 | net_exits < -10000 ~ 0,
    net_exits < 0 | net_exits > -10000 ~ abs(net_exits),
    TRUE ~ net_exits))

# Aggregate the number of entries and exits by turnstile.
tud21_sum <- tud21_clean %>%
  group_by(turnstile_id,
           date,
           station,
           line_name,
           unit,
           c_a,
           scp) %>%
  summarise(entry_total = sum(net_entries),
            exit_total = sum(net_exits))

tud21_rbs_join <- left_join(tud21_sum,
                            remote_booth_station,
                            by = c("station" = "station",
                                   "unit" = "unit")) %>%
  select(-c(line_name.x,
            booth,
            scp.y))

# Next, join the location data to the previous joined data frame
tud21_location_join <- left_join(tud21_rbs_join,
                                 mta_locations,
                                 by = "complex_id") %>%
  distinct(turnstile_id, date, station,
           .keep_all = TRUE)

# Clean up column names / reorganize columns
```

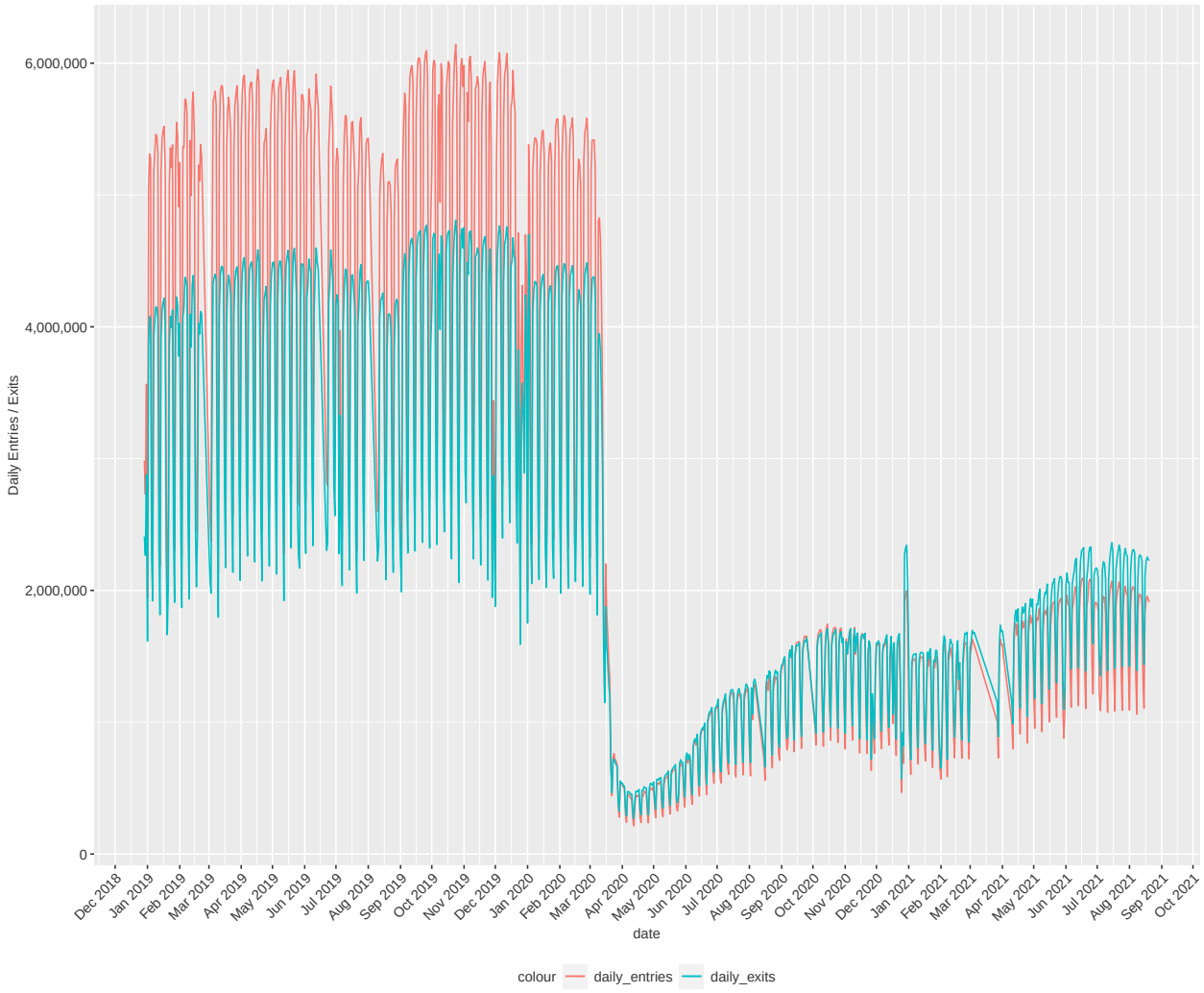
```
tud21_location_join <- tud21_location_join %>%
  select(-c(division.y,
            daytime_routes,
            ada_direction_notes,
            ada_nb,
            ada_sb,
            capital_outage_nb,
            capital_outage_sb)) %>%
  rename(station_abbr = station,
         scp = scp.x,
         line_name = line_name.y,
         station_long = stop_name)

tud21_location_join <- tud21_location_join %>%
  rename(division = division.x,
         complex_lat = gtfs_latitude,
         complex_long = gtfs_longitude)
```

Now let's combine the 2019, 2020, and 2021 data frames together for a complete data set.

```
turnstile_usage_2019_2021 <- bind_rows(tud19_location_join,
                                       tud20_location_join,
                                       tud21_location_join)

saveRDS(turnstile_usage_2019_2021, "turnstile_usage_2019_2021.rds")
```

Now we can start to visualize what MTA ridership looked like during the pandemic:

```
system_total <- turnstile_usage_2019_2021 %>%
  group_by(date) %>%
  summarise(daily_entries = sum(entry_total),
            daily_exits = sum(exit_total))

system_plot <- ggplot(data = system_total,
                          aes(x = date)) +
  geom_line(aes(y = daily_entries, color = "daily_entries")) +
  geom_line(aes(y = daily_exits, color = "daily_exits")) +
  scale_x_date(date_breaks = "1 month", date_labels = "%b %Y") +
  scale_y_continuous(name = "Daily Entries / Exits", labels = scales::comma) +
  theme(legend.position = "bottom",
        legend.text = element_text(color = "grey20", size = 10),
        legend.title = element_text(color = "grey20", size = 10),
        axis.text.x = element_text(color = "grey20", size = 10, angle = 45, vjust = 1, hjust = 1),
        axis.title.x = element_text(color = "grey20", size = 10),
        axis.text.y = element_text(color = "grey20", size = 10),
        axis.title.y = element_text(color = "grey20", size = 10))

system_plot
```



Adding COVID-19 Data

Now that we have our MTA ridership graph, we need to add a bit of context to it by overlaying COVID-19 case counts on top of that. Let's do that by importing NYC COVID-19 data from NYC Health and make the data set comptaible to simply join it by date:

```
# Import NYC COVID Data
nyc_covid_url <- "https://raw.githubusercontent.com/nychealth/coronavirus-data/master/trends/data-by-da

nyc_covid_data <- read_csv(nyc_covid_url) %>%
  clean_names()

# We only need cases by date for now, so let's only select these columns
cases_by_date <- nyc_covid_data %>%
  select(c(date_of_interest, case_count))

# Convert date column into class Date
cases_by_date$date_of_interest <- gsub('/', '-', cases_by_date$date_of_interest)
cases_by_date$date_of_interest <- mdy(cases_by_date$date_of_interest)
```

```
cases_by_date <- cases_by_date %>%
  rename(date = date_of_interest)
```

Let's join this COVID data to the turnstile data we have:

```
system_total <- left_join(system_total,
                          cases_by_date,
                          by = "date")


# Set NA to 0
system_total <- system_total %>%
  mutate(case_count = coalesce(case_count, 0))
```

Visualizing the Data

One issue with the previous plots was the "spikiness" of that data, which makes it hard to interpret trends over time. To better represent trends, I want to instead plot 7-day moving averages of the entries, exits, and COVID cases. We can easily calculate these averages using the `rollmean()` function in the `zoo` package:

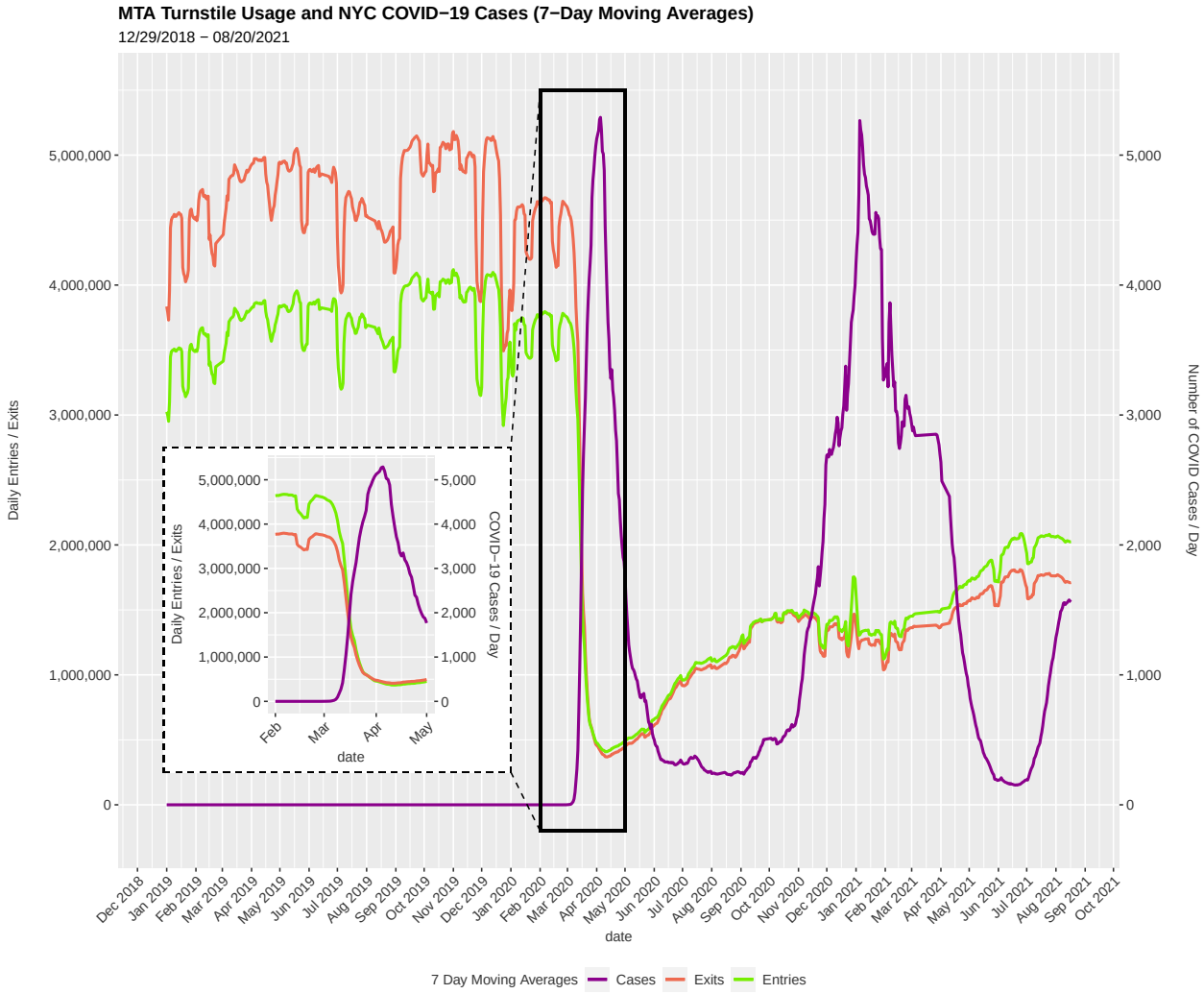
```
system_total <- system_total %>%
  mutate(entries_7da_ma = zoo::rollmean(daily_entries, k = 7, fill = NA),
         exits_7da_ma = zoo::rollmean(daily_exits, k = 7, fill = NA),
         cases_7da_ma = zoo::rollmean(case_count, k = 7, fill = NA))
```

With the COVID-19 case counts added and moving averages calculated, let's re-visualize the system wide turnstile data with the NYC COVID data:

```
coeff <- 1000
system_plot <- ggplot(data = system_total,
                          aes(x = date)) +
  geom_line(aes(y = entries_7da_ma, color = "entries_7da_ma"), size = 1) +
  geom_line(aes(y = exits_7da_ma, color = "exits_7da_ma"), size = 1) +
  geom_line(aes(y = cases_7da_ma*coeff, color = "cases_7da_ma"), size = 1) +
  scale_color_manual(labels = c("Cases", "Exits", "Entries"),
                     values = c("darkmagenta", "coral2", "chartreuse2")) +
  geom_rect(aes(xmin = ymd('2020-02-01'),
                    xmax = ymd('2020-05-01'),
                    ymin = -200000,
                    ymax = 5500000),
            color = "black",
            alpha = 0,
            size = 1) +
  scale_x_date(date_breaks = "1 month",
               date_labels = "%b %Y") +
  scale_y_continuous(name = "Daily Entries / Exits",
                     labels = scales::comma,
                     breaks = seq(0, 6000000, by = 1000000),
                     sec.axis = sec_axis(trans = ~./coeff,
                                         name = "Number of COVID Cases / Day",
```

```
                                        labels = scales::comma,
                                        breaks = seq(0, 6000, by = 1000))) +
  labs(title = "MTA Turnstile Usage and NYC COVID-19 Cases (7-Day Moving Averages)",
       subtitle = "12/29/2018 - 08/20/2021",
       color = "7 Day Moving Averages") +
  theme(plot.title = element_text(family = "Helvetica", face = "bold"),
        legend.position = "bottom",
        legend.text = element_text(color = "grey20", size = 10),
        legend.title = element_text(color = "grey20", size = 10),
        axis.text.x = element_text(color = "grey20", size = 10, angle = 45, vjust = 1, hjust = 1),
        axis.title.x = element_text(color = "grey20", size = 10),
        axis.text.y = element_text(color = "grey20", size = 10),
        axis.title.y = element_text(color = "grey20", size = 10, margin = margin(r = 20)),
        axis.title.y.right = element_text(color = "grey20", size = 10, margin = margin(l = 20)))

# Mini-plot
system_subplot <- ggplot(data = system_total,
                         aes(x = date)) +
  geom_line(aes(y = entries_7da_ma), color = "chartreuse2", size = 1) +
  geom_line(aes(y = exits_7da_ma), color = "coral2", size = 1) +
  geom_line(aes(y = cases_7da_ma*coeff), color = "darkmagenta", size = 1) +
  scale_x_date(limit = c(as.Date("2020-02-01"), as.Date("2020-05-01"))) +
  scale_y_continuous(name = "Daily Entries / Exits",
                     labels = scales::comma,
                     breaks = seq(0, 6000000, by = 1000000),
                     sec.axis = sec_axis(trans = ~./coeff,
                                         name = "COVID-19 Cases / Day",
                                         labels = scales::comma,
                                         breaks = seq(0, 6000, by = 1000))) +
  theme(plot.title = element_text(family = "Helvetica", face = "bold"),
        axis.text.x = element_text(color = "grey20", size = 10, angle = 45, vjust = 1, hjust = 1),
        axis.title.x = element_text(color = "grey20", size = 10),
        axis.text.y = element_text(color = "grey20", size = 10),
        axis.title.y = element_text(color = "grey20", size = 10, margin = margin(r = 10)),
        axis.title.y.right = element_text(color = "grey20", size = 10, margin = margin(l = 10)))

mta_covid_plot <- system_plot +
  annotation_custom(ggplotGrob(system_subplot),
                    xmin = ymd('2018-12-29'),
                    xmax = ymd('2020-01-01'),
                    ymin = 250000,
                    ymax = 2750000) +
  geom_rect(aes(xmin = ymd('2018-12-29'),
                xmax = ymd('2020-01-01'),
                ymin = 250000,
                ymax = 2750000),
            color='black',
            linetype = 'dashed',
            alpha = 0) +
  geom_path(aes(x, y, group = grp),
            data = data.frame(x = c(ymd('2020-02-01'),
                                    ymd('2020-01-01'),
                                    ymd('2020-02-01'),
```

```
                                ymd('2020-01-01')),
                      y = c(5500000,
                            2750000,
                            -200000,
                            250000),
                      grp = c(1, 1, 2, 2)),
          linetype = 'dashed')

mta_covid_plot
```



And there we have it! Our completed graph of MTA ridership throughout the COVID-19 pandemic.