NYPD Shooting Incident Data Report

A. Lenters

Overview

Welcome to my analysis of the **NYPD Shooting Incident Data (Historic)** dataset.[1] In this analysis, I will build a model to predict the monthly number of shooting incident reports in New York City. This model could help departments plan staffing levels.

Contents

- Setup performs setup, including user definitions (e.g. data locations) and libraries.
- Verbs creates custom verbs for data wrangling, for use with `dplyr`'s pipes.
- Wrangle Data imports, tidies and transforms the data.
- Exploratory Data Analysis uses visualization to look for trends in the data and get model ideas.
- Data summary summarizes the data and ensures there are no missing values.
- Modelling progressively fits models to the data, guided by graphical model evaluation.
- Conclusion summarizes my findings and next steps, and discusses potential sources of bias.
- Appendix shows my session information and the final model's performance/diagnostic plots.

Note to peer graders

You can find my data source overview in Wrangle Data, my missing data check in Data summary, my plots in Exploratory Data Analysis/Modelling, my model fits in Modelling, and my discussion of bias in Conclusion.

[1] Available by searching for "NYPD Shooting Incident Data (Historic)" at https://catalog.data.gov/dataset.

Setup

Setup ► Packages

Necessary packages are loaded via `library()`. Optional ones are loaded via `require()`.

You can install the necessary packages on your machine by running `install.packages(c("tidyverse", "broom", "knitr", "modelr"))` in the console, and following the prompt to restart R (if requested). You may want to run this command even if you have them installed already, as I have found that rendering works better with current versions of the packages. (You can see exactly which versions I used in the Appendix A - Session Info section in my PDF/html version of this report.)

```r
# necessary packages
library(rlang)
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.0     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x purrr::%@%()         masks rlang::%@%()
## x dplyr::filter()      masks stats::filter()
## x purrr::flatten()     masks rlang::flatten()
## x purrr::flatten_chr() masks rlang::flatten_chr()
## x purrr::flatten_dbl() masks rlang::flatten_dbl()
## x purrr::flatten_int() masks rlang::flatten_int()
## x purrr::flatten_lgl() masks rlang::flatten_lgl()
## x purrr::flatten_raw() masks rlang::flatten_raw()
## x purrr::invoke()      masks rlang::invoke()
## x dplyr::lag()         masks stats::lag()
## x purrr::splice()      masks rlang::splice()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```r
library(broom)
library(knitr)
library(modelr)
```

```
##
## Attaching package: 'modelr'
##
## The following object is masked from 'package:broom':
##
##     bootstrap
```

```r
# should already be part of tidyverse, but loading it just in case
library(lubridate)

# optional packages
require(pdftools)
```

```
## Loading required package: pdftools
## Using poppler version 23.04.0
```

```r
set.seed(1)
```

Setup ► User definitions

Here, users of this report can set the locations and backup locations of the files used in this analysis. The four URLs that have been provided are publicly accessible, so no changes should be necessary.

```
nypd_data_url <- paste0("https://data.cityofnewyork.us/api/views/",
                        "833y-fsy8/rows.csv?accessType=DOWNLOAD")

nypd_data_url_bak <- paste0("https://github.com/alenters/dsaaf-nypd/raw/",
                            "main/backup_data/",
                            "NYPD_Shooting_Incident_Data__Historic_.csv")

nyc_temps_url <- paste0("https://www.weather.gov/media/okx/Climate/",
                        "CentralPark/monthlyannualtemp.pdf")

nyc_temps_url_bak <- paste0("https://github.com/alenters/dsaaf-nypd/raw/",
                            "main/backup_data/",
                            "nyc_monthly_temps__historic.csv")
```

Setup ► Fixed definitions

Below are constants I use in my analysis, which you can optionally adjust. But the document should knit on any machine with the values I have set.

```
colors <- c(incident_reports = "black", temperature = "black",
            "Trend" = "darkblue", "Seasonal" = "#2fccff",
            "Actual" = "gray", "Predicted" = "blue", "Residual" = "red")

suggested_theme_base_size <- switch (
    knitr::pandoc_to(),
    latex = 9,
    html = 11,
    revealjs = 14,
    10 # conservative default
)

perf_plot_ymin <- -99
perf_plot_ymax <- 249
covid_start_date <- ymd("2020-03-01")
covid_end_date <- ymd("2020-12-31")
```

Verbs

All verbs used in the analysis will be created in this section. I have given them self-explanatory names, so you can skip this section and proceed to Wrangle Data, and return here to look at details as necessary.

Verbs ► Import

Along with the file we were asked to analyze (`NYPD_Shooting_Incident_Data__Historic_.csv`), I have identified two external sources I want to use in my analysis (monthly temperatures, and NYPD police commissioner names). This section creates all of the functions necessary to import the three data sources.

```
read_historic_shooting_data <- function(url) {
  read_csv(url,
              na = c("", "NA", "UNKNOWN", "U"),
              col_types = cols_only(
                INCIDENT_KEY = col_integer(),
                OCCUR_DATE = col_date(format = "%m/%d/%Y"),
                STATISTICAL_MURDER_FLAG = col_logical()
              )) %>%
    rename(incident_key = INCIDENT_KEY,
           date = OCCUR_DATE,
           statistical_murder_flag = STATISTICAL_MURDER_FLAG)
}

get_historic_shooting_data <- function() {
  tryCatch(read_historic_shooting_data(nypd_data_url),
         error = function(w){
           warning(paste("Could not read historic shooting data file from",
                         "source. Reverting to github backup."))
           read_historic_shooting_data(nypd_data_url_bak)
         })
}
```

```
read_backup_historic_temperatures <- function(url = nyc_temps_url_bak) {
  read_csv(url,
           col_types = cols(
             date = col_date(format = "%Y-%m-%d"),
             temperature = col_double()
             ))
}

read_historic_temperatures_from_source <- function (url = nyc_temps_url) {
  tmp_filename <- tempfile("data", fileext = c(".pdf"))[[1]]
  download.file(url, tmp_filename)

  headers <- c("YEAR", "01", "02", "03", "04", "05", "06", "07", "08", "09",
               "10", "11", "12", "ANNUAL")
  pages <- pdftools::pdf_text(tmp_filename)

  .is_data_row <- function(cells) {
    if ("Average" %in% cells | "Last" %in% cells | "YEAR" %in% cells) {
      FALSE # known identifiers of headers
    } else if (length(cells) == 14) {
      TRUE # expected format of data rows
    } else {
```

```r
      FALSE # ignore anything unexpected
    }
  }

  .process_row <- function(row) {
    cells <- unlist(strsplit(row, "\\s+"))
    if (.is_data_row(cells)) {
      as_tibble_row(setNames(cells, headers))
    } else {
      NA
    }
  }

  .process_page <- function(page) {
    scan(textConnection(page), what = "character", sep = "\n") %>%
      map(.process_row) %>%
      purrr::discard(~ all(is.na(.x))) %>%
      reduce(union_all)
  }

  map(pages, .process_page) %>%
    reduce(union_all) %>%
    select(!c(ANNUAL)) %>%
    pivot_longer(cols = -c(YEAR),
                 names_to = c("MONTHNAME"),
                 values_to = "temperature") %>%
    unite(date, YEAR:MONTHNAME) %>%
    mutate(date = parse_date(date, format = "%Y_%m")) %>%
    mutate(temperature = as.double(temperature))
}

get_historic_temperatures <- function() {
  if(require(pdftools)) {
    tryCatch(read_historic_temperatures_from_source(),
           error = function(w){
             warning(paste("Could not read historic temperature data file from",
                           "source. Reverting to github backup."))
             read_backup_historic_temperatures()
           })
  } else {
    warning(paste("Reading backup temperatures file from github because",
                  "pdftools is not installed."))
    read_backup_historic_temperatures()
  }
}
```

```r
# https://en.wikipedia.org/wiki/New_York_City_Police_Commissioner
# commissioner_start_date: inclusive
# commissioner_end_date: exclusive
get_historic_police_commissioners <- function() {
  tribble(
  ~commissioner, ~commissioner_start_date, ~commissioner_end_date,
  "Raymond Walter Kelly", "2002-01-01", "2013-12-31",
  "William Joseph Bratton", "2014-01-01", "2016-09-15",
```

```
  "James P. O'Neill", "2016-09-16", "2019-11-30",
  "Dermot F. Shea", "2019-12-01", "2021-12-31",
  "Keechant Sewell", "2022-01-01", "2023-06-30",
  "Edward A. Caban", "2023-07-01", "2024-04-10" # date of data capture
) %>%
    mutate(
      across(commissioner, as.factor),
      across(starts_with("commissioner_"), as.Date)
      )
}
```

Verbs ▶ Tidy

```
collapse_incident_reports <- function(df) {
  df %>% group_by(incident_key, date) %>%
    summarize(incident_reports = 1,
              murdered_victims = sum(statistical_murder_flag),
              injured_victims = sum(!statistical_murder_flag),
              .groups = "drop")
}
```

Verbs ▶ Transform

```
safe_lookup_join <- function (x, y, join_cols) {
  y_lookup <- y %>%
                inner_join(x %>%
                             select(all_of(join_cols)) %>%
                             unique(),
                           by = join_cols,
                           relationship = "one-to-one")
  inner_join(x,
             y_lookup,
             by = join_cols,
             relationship = "many-to-one",
             unmatched = "error")
}
```

```
roll_up <- function(df, ..., date_unit = "week") {
  df %>%
    mutate(date = floor_date(date, unit = date_unit)) %>%
    group_by(pick(where(~!is.numeric(.x)))) %>%
    summarize(..., .groups = "drop")
}
```

```
add_temperatures <- function(df) {
  df %>%
    safe_lookup_join(get_historic_temperatures(), c("date"))
}
```

```
add_commissioners <- function(df) {
  df %>%
    left_join(get_historic_police_commissioners(),
              by = join_by(between(x$date,
```

```
                           y$commissioner_start_date,
                           y$commissioner_end_date))) %>%
    select(!c(commissioner_start_date, commissioner_end_date))
}
```

Verbs ► EDA

```
timeseries_base <- function(df, .date_col, .ts_col,
                            ...,
                            theme_base_size = suggested_theme_base_size,
                            color_mapping = colors) {
  df %>%
    ggplot(mapping = aes(x = {{ .date_col }})) +
      theme_minimal(base_size = theme_base_size) +
      theme(legend.position = "none") +
      scale_color_manual(values = color_mapping) +
      geom_line(aes(y = {{ .ts_col }},
                    color = rlang::englue("{{ .ts_col }}")),
                linewidth = 1.1,
                alpha = 0.8)
}
```

```
do_decomp <- function(df, .col, .date_col) {
  date_col <- pull(df, {{ .date_col }})
  decompose_col = pull(df, {{ .col }})
  min_date <- min(date_col)
  max_date <- max(date_col)
  decomposed <- ts(decompose_col,
                   start = c(year(min_date), month(min_date)),
                   end = c(year(max_date), month(max_date)),
                   frequency = 12) %>%
                     decompose()
  df %>%
    mutate(IR_seasonal = tibble(decomposed$seasonal)[[1]]) %>%
    mutate(IR_trend = tibble(decomposed$trend)[[1]]) %>%
    mutate(IR_resid = tibble(decomposed$random)[[1]])
}
```

```
plot_decomp <- function(df,
                        min_date = NULL,
                        max_date = NULL,
                        ...,
                        .colorize_col = NULL,
                        theme_base_size = suggested_theme_base_size,
                        color_mapping = colors) {
  data_to_visualize <- df

  if (!is_null(min_date)) {
    data_to_visualize <- data_to_visualize %>%
      dplyr::filter(date >= min_date)
  }
  if (!is_null(max_date)) {
    data_to_visualize <- data_to_visualize %>%
      dplyr::filter(date <= max_date)
```

```r
  }

  data_to_visualize %>%
    ggplot(aes(x = date)) +
      theme_minimal(base_size = theme_base_size) +
      theme(legend.position = "top", legend.title = element_blank()) +
      scale_color_manual(values = color_mapping) +
      geom_bar(aes(y = IR_resid, color = "Residual"),
               fill = color_mapping["Residual"],
               stat = "identity",
               linewidth = 0,
               alpha = 0.8) +
      geom_line(aes(y = IR_seasonal, color = "Seasonal"),
                linewidth = 1.1,
                alpha = 0.8) +
      geom_line(aes(y = IR_trend, color = "Trend"),
                linetype = "dotted",
                linewidth = 1,
                alpha = 0.8) +
      labs(
        ...,
        color = "Legend")
}
```

Verbs ▶ Model

```r
.adjust_pred_for_poisson <- function(df, .col) {
  df %>%
    mutate({{ .col }} := exp({{ .col }}))
}

.adjust_resid_for_poisson <- function(df, .col, .actual_col) {
  df %>%
    mutate({{ .col }} := {{ .actual_col }} -
           {{ .actual_col }}/exp({{ .col }}))
}

make_poisson_predictions <- function(df, .model, .actual_col) {
  df %>%
    modelr::add_predictions(.model) %>%
    .adjust_pred_for_poisson(pred) %>%
    modelr::add_residuals(.model) %>%
    .adjust_resid_for_poisson(resid, {{ .actual_col }})
}
```

Verbs ▶ Communicate

```r
has_na <- function(df) {
  sum(is.na(df)) > 0
}
```

```r
present_table <- function(df, ..., n = 4, caption = "") {
  knitr::kable(head(df,
```

```r
                     n = n),
                align = "l",
                caption = caption)
}
```

```r
model_adj_rs <- function(model, format = "number") {
  ret <- round(broom::glance(model)$adj.r.squared, 4)
  if (format == "percent") {
    ret <- rlang::englue("{ret * 100}%")
  }
  ret
}
```

```r
plot_model_performance <- function(df, .model, .actual_col,
                                   ...,
                                   subtitle = "Model performance",
                                   theme_base_size = suggested_theme_base_size,
                                   color_mapping = colors,
                                   ymin = NA,
                                   ymax = NA) {
  m <- broom::glance(.model)
  df %>%
    make_poisson_predictions(.model, {{ .actual_col }}) %>%
    ggplot(mapping = aes(x = date)) +
      theme_minimal(base_size = theme_base_size) +
      theme(legend.position = "top", legend.title = element_blank()) +
      scale_color_manual(values = color_mapping) +
      ylim(ymin, ymax) +
      geom_line(aes(y = {{ .actual_col }}, color = "Actual"),
                linewidth = 1.1,
                alpha = 0.7) +
      geom_line(aes(y = pred, color = "Predicted"),
                linewidth = 0.8) +
      geom_bar(aes(y = resid, color = "Residual"),
               fill = color_mapping["Residual"],
               linewidth = 0,
               alpha = 0.9,
               stat="identity") +
      labs(x = "Month",
           y = "Number of incident reports",
           color = "Legend",
           subtitle = subtitle,
           caption = paste0("Adj. R squared: ", round(m$adj.r.squared[[1]], 4),
                            ", AIC: ", round(m$AIC[[1]]),
                            ", BIC: ", round(m$BIC[[1]])),
           ...)
}
```

```r
date_span_str <- function(start_date, end_date) {
  sd_month <- format(start_date, "%B")
  sd_year <- format(start_date, "%Y")
  ed_month <- format(end_date, "%B")
  ed_year <- format(end_date, "%Y")
```

```r
  if (sd_year == ed_year) {
    rlang::englue("{ sd_month} to { ed_month } { ed_year }")
  } else {
    rlang::englue("{ sd_month} { sd_year } to { ed_month } { ed_year }")
  }
}
```

Wrangle Data

I will now use the verbs I created in Verbs, to import, tidy, transform and visualize the data, in preparation for modelling.

Wrangle ▶ Data Source

The data source I will be using is the **NYPD Shooting Incident Data (Historic)** dataset.[2]

Each time a shooting incident (that results in injury or death) is reported in New York City, an incident report is filed. The incidents from 2006 - 2022 are available in this historic file. The column definitions (provided by the dataset's Data Dictionary[3]) are shown below. I have decided to start with a high-level analysis, looking at monthly counts of incident reports, so I only need to use a few of these columns. I have marked the columns I will use in this analysis with **.

Table 1: Column definitions for the `NYPD Shooting Incident Data (Historic)` dataset. {#tbl-col-defs}

Column Name	Description
**INCIDENT_KEY	Randomly generated persistent ID for each incident
**OCCUR_DATE	Exact date of the shooting incident
OCCUR_TIME	Exact time of the shooting incident
BORO	Borough where the shooting incident occurred
PRECINCT	Precinct where the shooting incident occurred
JURISDICTION_CODE	Jurisdiction where the shooting incident occurred. Jurisdiction codes 0(Patrol), 1(Transit) and 2(Housing) represent NYPD whilst codes 3 and more represent non NYPD jurisdictions
LOCATION_DESC	Location of the shooting incident
**STATISTICAL_MURDER_FLAG	Shooting resulted in the victim's death which would be counted as a murder
PERP_AGE_GROUP	Perpetrator's age within a category
PERP_SEX	Perpetrator's sex description
PERP_RACE	Perpetrator's race description
VIC_AGE_GROUP	Victim's age within a category
VIC_SEX	Victim's sex description
VIC_RACE	Victim's race description
X_COORD_CD	Midblock X-coordinate for New York State Plane Coordinate System, Long Island Zone, NAD 83, units feet (FIPS 3104)
Y_COORD_CD	Midblock Y-coordinate for New York State Plane Coordinate System, Long Island Zone, NAD 83, units feet (FIPS 3104)

Wrangle ▶ Import

Let's start by reading the data in from the source csv. The `get_historic_shooting_data()` verb I created for this purpose selects only the columns I want to use in this analysis. Importing a subset of columns is cleanly achieved by using `cols_only()` to create the `col_types` argument to `read_csv()`.

[2] Available by searching for "NYPD Shooting Incident Data (Historic)" at https://catalog.data.gov/dataset.

[3] https://data.cityofnewyork.us/api/views/833y-fsy8/files/f5f61d94-6961-47bd-8d3c-e57ebeb4cb55?download=true&filename=NYPD_Shootings_Historic_DataDictionary.xlsx

```r
shooting_data <- get_historic_shooting_data()
```

Let's have a quick look at the data:

```r
glimpse(shooting_data)
```

```
## Rows: 27,312
## Columns: 3
## $ incident_key             <int> 228798151, 137471050, 147998800, 146837977, 58~
## $ date                     <date> 2021-05-27, 2014-06-27, 2015-11-21, 2015-10-0~
## $ statistical_murder_flag <lgl> FALSE, FALSE, TRUE, FALSE, TRUE, TRUE, FALSE, ~
```

Wrangle ▶ Tidy

In this section, I will check the NYPD Historic Shooting dataframe to confirm that it is tidy.

Tidy ▶ Are there any missing values?

We want to avoid having any missing values in the dataframe, because they will cause issues in any calculation or plot they are included in.

```r
if(shooting_data %>% has_na()) {
  stop("NA values found in dataset. Please fix.")
}
```

Currently, there are no missing (`NA`) values in the data, but I have included a `stop()` condition in case some appear in future. The columns I am using (`incident_key`, `date` and `statistical_murder_flag`) are so central to the dataset that they should not be missing, and I cannot a priori say how missing values should be handled. The ideal fix would be to correct the source file by filling in the missing values. (And in my opinion, for reproducibility purposes, a loud error up front is better than mysterious issues later on caused by the missing values.)

The three tidiness requirements:

For a tibble to be tidy, three requirements should be met:

1. Each variable has its own column
2. Each observation has its own row
3. Each value has its own cell

The book "R For Data Science" notes[4] that if two of these are true the third is also guaranteed to be true, so I will check the first two.

Tidy ▶ Does each variable have its own column?

Short answer: Yes.

```r
shooting_data %>%
  present_table(caption = "Sample of the shooting data.")
```

Table 2: Sample of the shooting data.

incident_key	date	statistical_murder_flag
228798151	2021-05-27	FALSE
137471050	2014-06-27	FALSE
147998800	2015-11-21	TRUE

[4] https://r4ds.had.co.nz/tidy-data.html

incident_key	date	statistical_murder_flag
146837977	2015-10-09	FALSE

I have chosen to use three columns from the source csv. No variable is spread across multiple columns, and no column contains multiple variables, so this requirement is met.

Tidy ▶ Does each observation have its own row?

Short answer: No.

To check this requirement, we need to know what defines an "observation" in this dataset. I would consider each incident to be a single observation. The dataset's official landing page[5] links to a footnotes PDF, which states (in footnote 3) that incident reports can span multiple rows if multiple victims were involved. This would violate tidy data principles, by giving multiple rows to a single observation. Let's check for this:

```
duplicates <- shooting_data %>%
  count(incident_key, sort = TRUE)

least_tidy_incident_key <- as.integer(duplicates$incident_key[[1]])
least_tidy_incident_n <- as.integer(duplicates$n[[1]])

duplicates %>%
  present_table(caption = paste("Top number of data rows used by various",
                                "`incident_keys` (should be 1 row per",
                                "`incident_key`)."))
```

Table 3: Top number of data rows used by various `incident_keys` (should be 1 row per `incident_key`).

incident_key	n
173354054	18
23749375	12
24717013	12
33478089	12

This shows that single incident reports have up to 18 rows in the dataframe! Let's check the least tidy `incident_key`, 173354054:

```
shooting_data %>%
  dplyr::filter(incident_key == least_tidy_incident_key) %>%
  present_table(caption = paste0("Records for a single `incident_key`, ",
                                 least_tidy_incident_key,
                                 "."),
                n = Inf)
```

Table 4: Records for a single `incident_key`, 173354054.

incident_key	date	statistical_murder_flag
173354054	2018-01-06	TRUE
173354054	2018-01-06	TRUE
173354054	2018-01-06	FALSE

[5] https://data.cityofnewyork.us/Public-Safety/NYPD-Shooting-Incident-Data-Historic-/833y-fsy8

incident_key	date	statistical_murder_flag
173354054	2018-01-06	FALSE
173354054	2018-01-06	TRUE
173354054	2018-01-06	TRUE
173354054	2018-01-06	TRUE
173354054	2018-01-06	TRUE
173354054	2018-01-06	TRUE
173354054	2018-01-06	FALSE
173354054	2018-01-06	FALSE
173354054	2018-01-06	FALSE
173354054	2018-01-06	FALSE
173354054	2018-01-06	TRUE
173354054	2018-01-06	FALSE
173354054	2018-01-06	FALSE
173354054	2018-01-06	TRUE
173354054	2018-01-06	FALSE

Tidy ▶ Collapse duplicates

In the sample above, all rows for a single `incident_key` have the same `date`, but different values for `statistical_murder_flag`.

I think the best way to modify the tibble to meet requirement 2 is to add two new features to the dataframe: `injured_victims` and `murdered_victims`. This will be handled by logic I defined earlier, in the verb `collapse_incident_reports()`.

```
tidy_shooting_data <- shooting_data %>%
  collapse_incident_reports()

tidy_shooting_data %>%
  present_table(caption = "Post-collapse, sample of the shooting data.")
```

Table 5: Post-collapse, sample of the shooting data.

incident_key	date	incident_reports	murdered_victims	injured_victims
9953245	2006-01-01	1	0	1
9953246	2006-01-01	1	0	1
9953247	2006-01-01	1	0	1
9953248	2006-01-01	1	0	1

Let's re-run our duplicates checks from above:

```
tidy_shooting_data %>%
  count(incident_key, sort = TRUE) %>%
  present_table(caption = paste("Post-collapse, top number of data rows used",
                                "by various `incident_keys`."))
```

Table 6: Post-collapse, top number of data rows used by various `incident_keys`.

incident_key	n
9953245	1
9953246	1

incident_key	n
9953247	1
9953248	1

No duplicates!

```
tidy_confirm <- tidy_shooting_data %>%
  dplyr::filter(incident_key == least_tidy_incident_key)

tidy_confirm %>%
  present_table(caption = paste("Post-collapse, records for the same",
                                "`incident_key` we looked at earlier."),
                n = least_tidy_incident_n)
```

Table 7: Post-collapse, records for the same `incident_key` we looked at earlier.

incident_key	date	incident_reports	murdered_victims	injured_victims
173354054	2018-01-06	1	9	9

The data for this incident has been tidied up to a single row, with the 18 victims we found earlier aggregated into a single incident report with 9 `murdered_victims` and 9 `injured_victims`.

Done tidying!

The `data` dataframe has been successfully tidied, with one row per observation, and one column per variable, so we can move on to transforming the data to make it work for this particular analysis.

Wrangle ► Transform

Transform ► Roll up monthly

To get a clearer view of trends in the data, I want to roll it up to monthly. I will use the verb I created for this purpose, `roll_up()`. This change will also be helpful later on if we want to bring in additional data, because some data is only available at monthly granularity.

```
data <- tidy_shooting_data %>%
  roll_up(incident_reports = sum(incident_reports),
          murdered_victims = sum(murdered_victims),
          injured_victims = sum(injured_victims),
          date_unit = "month")
```

```
glimpse(data)
```

```
## Rows: 204
## Columns: 4
## $ date             <date> 2006-01-01, 2006-02-01, 2006-03-01, 2006-04-01, 2006~
## $ incident_reports <dbl> 112, 81, 79, 113, 134, 146, 169, 177, 152, 150, 121, ~
## $ murdered_victims <int> 29, 27, 14, 37, 40, 36, 47, 46, 44, 38, 40, 47, 14, 1~
## $ injured_victims  <int> 100, 70, 88, 119, 133, 144, 186, 199, 152, 161, 127, ~
```

Exploratory Data Analysis

My EDA will primarily reply on visualization to explore the data and look for relationships between variables. I will use the EDA process to generate modelling ideas, which I will use later on during modelling.

EDA ▶ Response variable timeseries

Let's look at a timeseries of the number of monthly incident reports:

```
min_year <- min(year(data$date))
max_year <- max(year(data$date))
data %>%
  timeseries_base(date, incident_reports) +
    labs(title = "NYC Shooting Incident Reports",
         subtitle = rlang::englue("Monthly, { min_year } - { max_year }"),
         caption = paste("Source: NYPD Shooting Incident Data (Historic)",
                         "dataset from https://catalog.data.gov/dataset"),
         x = "Month",
         y = "Number of incident reports",
         color = "Legend")
```

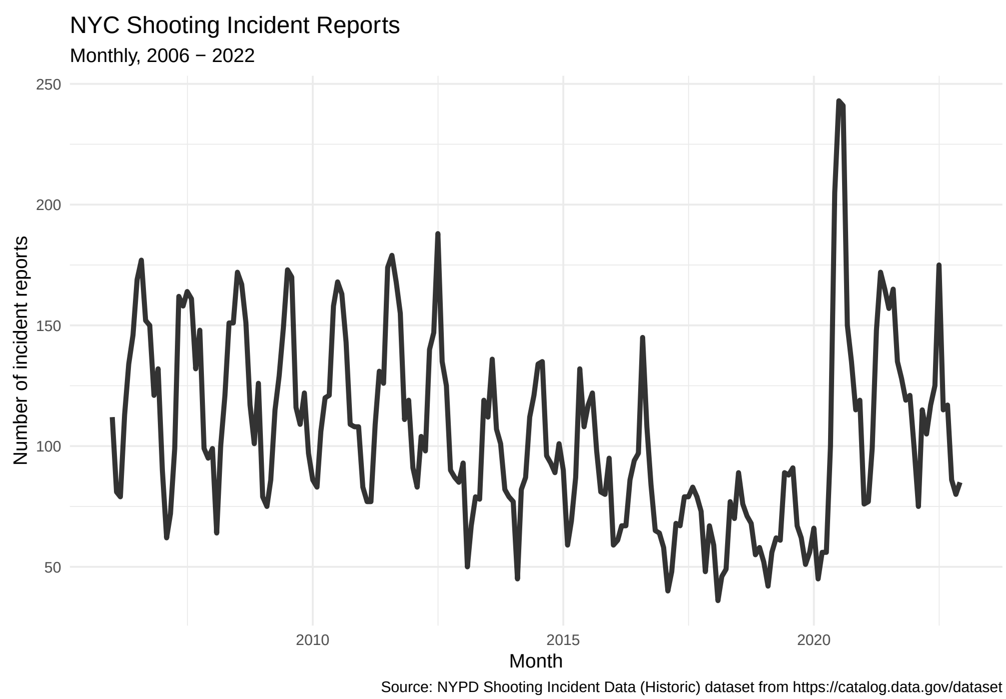

Figure 1: Monthly incident reports timeseries.

There is a strong yearly cyclical pattern in the number of incident reports. Also, the number of monthly incident reports had been declining during the 2010s, but it jumped up in 2020.

EDA ▶ Decompose

We can decompose the `incident_reports` timeseries into three components (`trend` + `seasonal` + `residual`) to formalize the seasonality and trend patterns we've spotted in the data, and gain insights into possible predictors. This decomposition uses the standard R function `decompose()` from the `stats` package, with its default settings.

```
data %>%
  do_decomp(incident_reports, date) %>%
  plot_decomp(min_date = ymd("2007-01-01"),
              max_date = ymd("2021-12-01"),
              title = "Incident reports = Trend + Seasonal + Residual",
              subtitle = "Monthly, 2007 - 2021",
              caption = paste("Incident reports timeseries decomposed into",
                              "Trend + Seasonal + Residual timeseries."),
                           x = "Date",
                           y = "Number of incident reports")
```

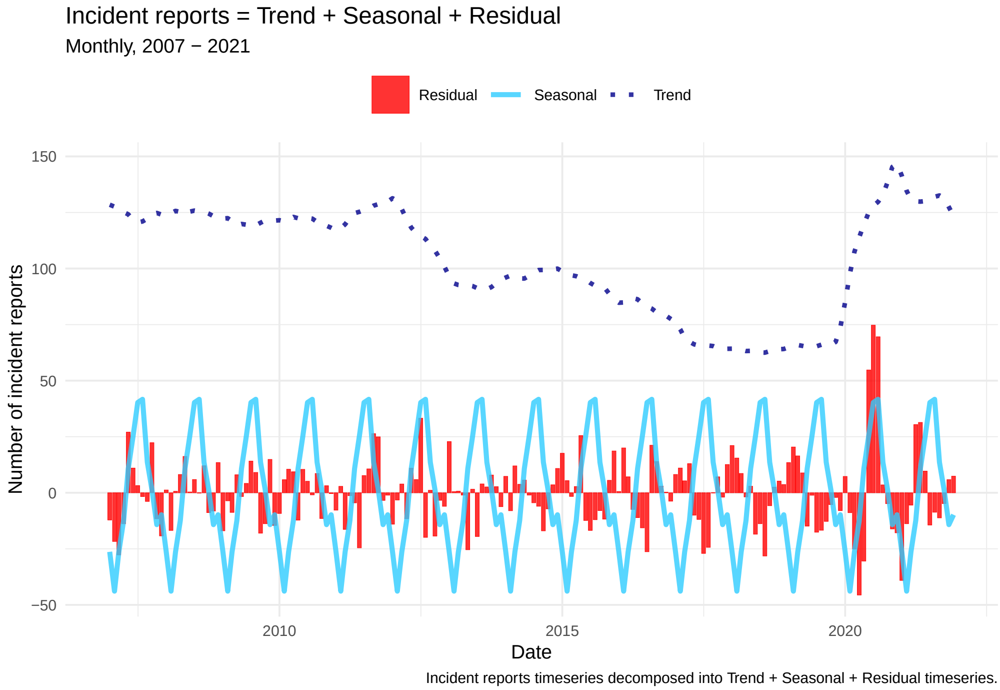

Figure 2: Decomposed incident reports timeseries.

Feature engineering

The decomposition suggests some features we can use in our model:

- Seasonal: The yearly cycle has its peaks in the summertime and its troughs in the winter. Perhaps there is a relation to **temperature**.
- Trend: There is a stepwise declining trend over **time** until 2020, where it jumps up. I wonder if these step-changes correspond to changes in leadership in the NYPD (e.g. the **commissioner**)–perhaps due to changes in police operations and/or incident reporting.

Transform ▶ Bring in additional data

I will use data on NYC's monthly temperatures[6] and police commissioners[7] to explore the questions brought up by the timeseries decomposition. I wrote the verbs `add_temperatures()` and `add_commissioners()` to fetch the data from source and join it to our dataframe.

[6] https://www.weather.gov/media/okx/Climate/CentralPark/monthlyannualtemp.pdf
[7] https://en.wikipedia.org/wiki/New_York_City_Police_Commissioner

Internally, `add_temperatures()` uses another verb I wrote called `safe_lookup_join()`, which performs a "lookup join" (adding a new column to an existing dataframe) without any risk of: 1) missing values in the new column, or 2) silently dropping rows from the main dataframe, which are the risks of traditional **left** and **inner** joins. An error will be issued and execution will be halted if the lookup table lacks a value for any row in the main dataframe.

`add_commissioners()` uses a different type of join, due to the nature of the commissioners dataframe, which contains one row per commissioner, with start and end dates. It uses the `between()` overlap helper function inside of the `join_by()` specification to state that each row should be assigned to the commissioner who was active at the time of the incident. This could be performed at daily granularity, but to simplify the data model, I will apply it after aggregating to monthly granularity, so the commissioner for each month will be whoever was in charge at the beginning of the month.

```r
glimpse(data)
```

```
## Rows: 204
## Columns: 4
## $ date             <date> 2006-01-01, 2006-02-01, 2006-03-01, 2006-04-01, 2006~
## $ incident_reports <dbl> 112, 81, 79, 113, 134, 146, 169, 177, 152, 150, 121, ~
## $ murdered_victims <int> 29, 27, 14, 37, 40, 36, 47, 46, 44, 38, 40, 47, 14, 1~
## $ injured_victims  <int> 100, 70, 88, 119, 133, 144, 186, 199, 152, 161, 127, ~
```

```r
data <- data %>%
    add_commissioners() %>%
    add_temperatures()
```

Let's have a look at the new dataframe:

```r
glimpse(data)
```

```
## Rows: 204
## Columns: 6
## $ date             <date> 2006-01-01, 2006-02-01, 2006-03-01, 2006-04-01, 2006~
## $ incident_reports <dbl> 112, 81, 79, 113, 134, 146, 169, 177, 152, 150, 121, ~
## $ murdered_victims <int> 29, 27, 14, 37, 40, 36, 47, 46, 44, 38, 40, 47, 14, 1~
## $ injured_victims  <int> 100, 70, 88, 119, 133, 144, 186, 199, 152, 161, 127, ~
## $ commissioner     <fct> Raymond Walter Kelly, Raymond Walter Kelly, Raymond W~
## $ temperature      <dbl> 40.9, 35.7, 43.1, 55.7, 63.1, 71.0, 77.9, 75.8, 66.6,~
```

EDA ► Temperature timeseries

```r
data %>%
  timeseries_base(date, temperature) +
  labs(title = "Temperature timeseries",
       y = "Avg monthly temperature (°F)")
```

EDA ► Incident reports distribution

In order to use linear regression, the response variable should be normally distributed. Let's check the general shape of the distribution for `incident_reports` by using `geom_density()`.

```r
data %>%
  ggplot(aes(x = incident_reports)) +
    geom_density(linewidth = 1.1, alpha = 0.8) +
    theme_minimal(base_size = suggested_theme_base_size) +
    labs(title = "Number of incident reports is not normally distributed")
```

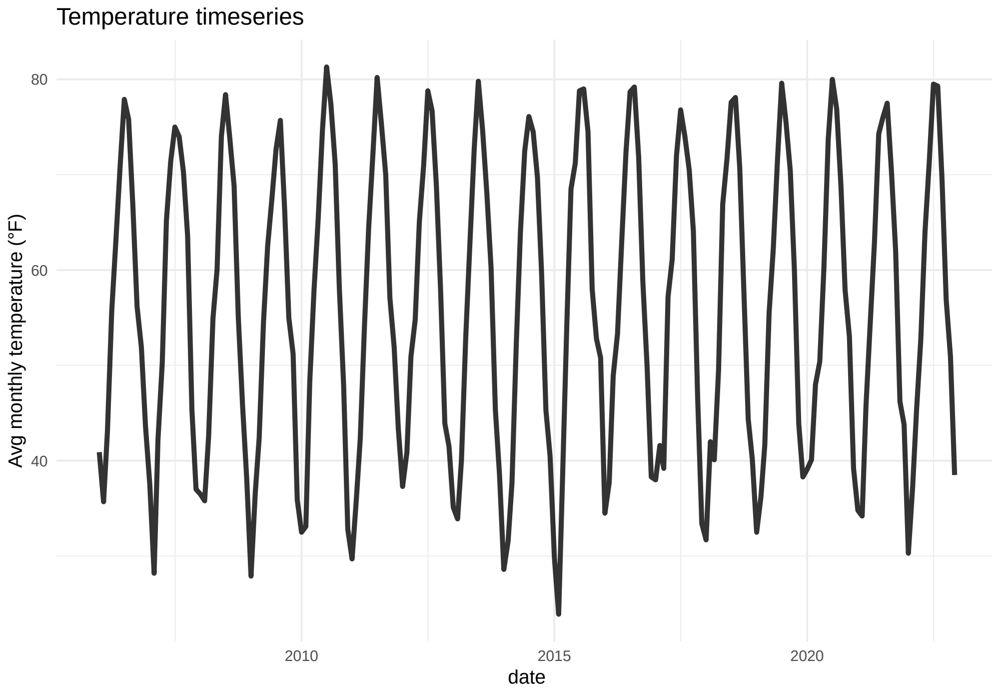

Figure 3: Monthly average temperature timeseries.

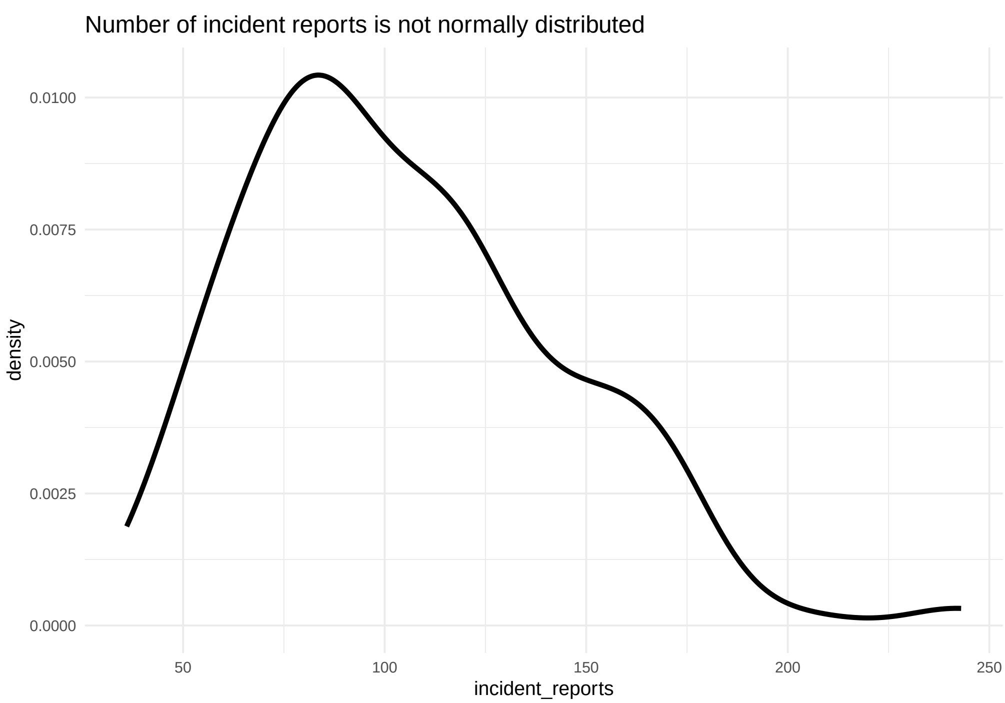

Figure 4: Density plot for 'incident reports'.

This does not look like a normal distribution. On the left side, it has a hard minimum of 0, while the right side is unbounded, with a long tail. Given that the `incident_reports` variable counts events in a fixed time period, it is more likely to follow a **poisson** distribution. A log transform should therefore make it more normal.

```
data %>%
  ggplot(aes(x = log(incident_reports))) +
    geom_density(linewidth = 1.1, alpha = 0.8) +
    theme_minimal(base_size = suggested_theme_base_size) +
    labs(title = "log(Number of incident reports) is more normally distributed")
```

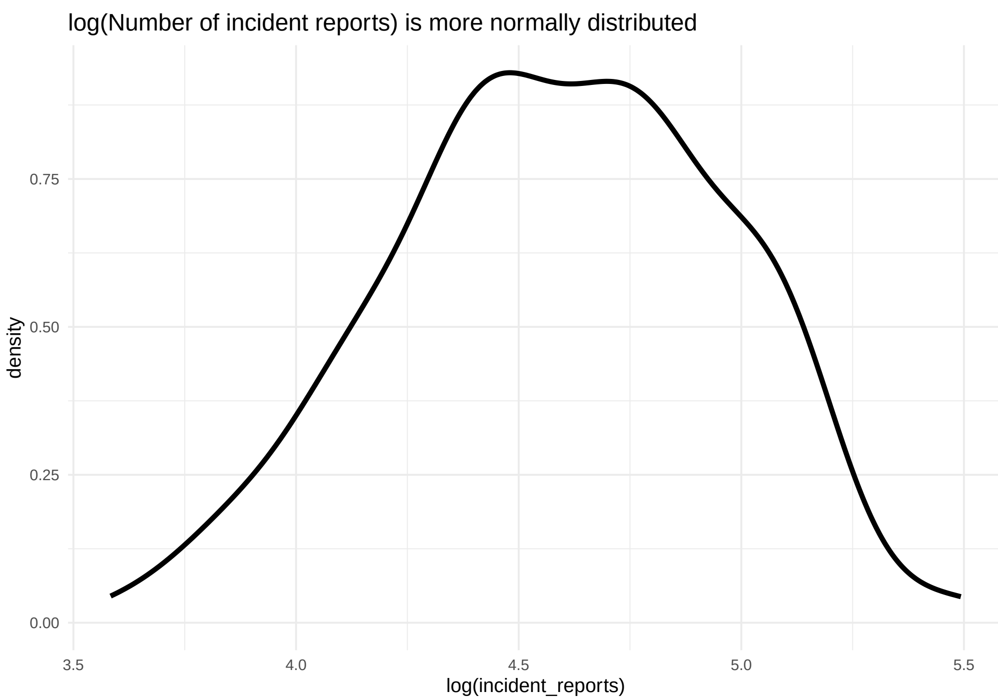

Figure 5: Density plot for ‘log(incident reports)‘.

This does look more normal. I will therefore use `log(incident_reports)` as the response variable.

Next, let’s look for relationships between our response variable, `log(incident_reports)`, and features that could be used in the model (`date`, `temperature`, `commissioner`).

EDA ▶ log(Incident reports) ~ Temperature

Let’s start with the relationship between `log(incident_reports)` and `temperature`.

```
data %>%
  ggplot(aes(x = temperature, y = log(incident_reports))) +
  geom_point() +
  geom_smooth(method = "loess", formula = "y ~ x") +
  theme_minimal(base_size = suggested_theme_base_size) +
  labs(title = "How does temperature affect incident report counts?",
       x = "Avg monthly temperature (°F)",
       y = "log(Number of incident reports)")
```

The relationship between `temperature` and `log(incident_reports)` is roughly linear, making `temperature` a great candidate to add to a linear model.

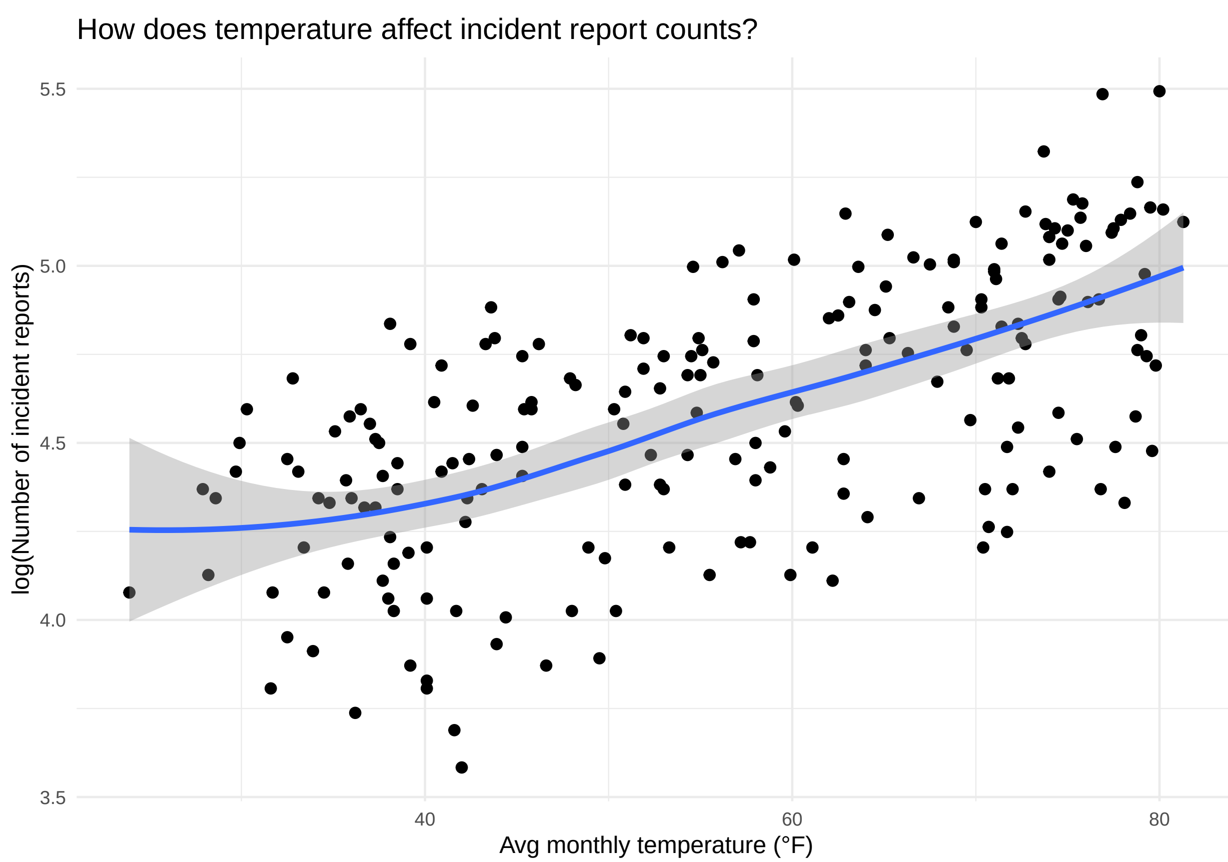

Figure 6: `temperature` vs `log(incident reports)`.

EDA ▶ log(Incident reports) ~ Date, Commissioner

Let's look for an interaction between `date` and `commissioner`.

```r
data %>%
  ggplot(aes(x = date, y = incident_reports)) +
  geom_line(aes(color = commissioner), linewidth = 1.1) +
  geom_smooth(aes(group = commissioner),
              method = "lm",
              formula = "y ~ x",
              se = FALSE,
              color = "black",
              linewidth = 0.9,
              alpha = 0.7,
              linetype = "dotted") +
  theme_minimal(base_size = suggested_theme_base_size) +
  theme(legend.position = "top",
        legend.title = element_blank()) +
  guides(color = guide_legend(nrow = 2)) +
  labs(title = paste("NYPD Commissioner seems to influence",
                     "number of incident reports"),
       x = "Date",
       y = "Number of incident reports")
```

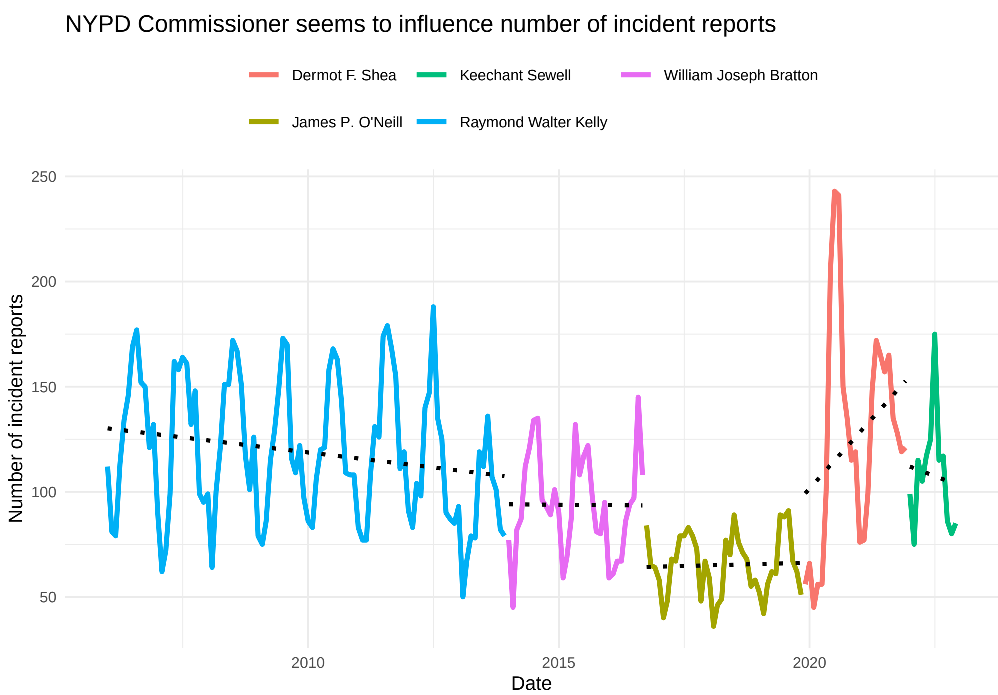

Figure 7: 'commissioner' vs 'incident reports'.

After fitting separate trend lines for each police Commissioner, there is a fairly clear correspondence between `commissioner` and the step-changes in `incident_reports`. For example, see the abrupt change where James P. O'Neill took over from William Joseph Bratton.

In general, the peaks and troughs of each commissioner's timeseries stay at the same level over time. So if we removed the seasonal component, the remaining trend for each commissioner would be a roughly linear relationship between `date` and `incident_reports`. Therefore, including `date` as a continuous feature and `commissioner` as a factor feature in the model should provide useful information and help improve the model.

EDA ► Recap

We have ended up with three candidate features for a linear model: `temperature`, `date`, and `commissioner`. We are ready to move onto modelling.

Data summary

Before moving on to modelling, let's do a quick data summary to make sure there is no missing data.

Here are the columns we ended up with in the dataframe.

```
glimpse(data)
```

```
## Rows: 204
## Columns: 6
## $ date              <date> 2006-01-01, 2006-02-01, 2006-03-01, 2006-04-01, 2006~
## $ incident_reports <dbl> 112, 81, 79, 113, 134, 146, 169, 177, 152, 150, 121, ~
## $ murdered_victims <int> 29, 27, 14, 37, 40, 36, 47, 46, 44, 38, 40, 47, 14, 1~
## $ injured_victims  <int> 100, 70, 88, 119, 133, 144, 186, 199, 152, 161, 127, ~
## $ commissioner     <fct> Raymond Walter Kelly, Raymond Walter Kelly, Raymond W~
## $ temperature      <dbl> 40.9, 35.7, 43.1, 55.7, 63.1, 71.0, 77.9, 75.8, 66.6,~
```

Actually, I am not planning on using `murdered_victims` or `injured_victims`, so those columns can be dropped at this time.

```
data <- data %>%
  select(!c(murdered_victims, injured_victims))
```

The remaining columns are:

```
glimpse(data)
```

```
## Rows: 204
## Columns: 4
## $ date              <date> 2006-01-01, 2006-02-01, 2006-03-01, 2006-04-01, 2006~
## $ incident_reports <dbl> 112, 81, 79, 113, 134, 146, 169, 177, 152, 150, 121, ~
## $ commissioner     <fct> Raymond Walter Kelly, Raymond Walter Kelly, Raymond W~
## $ temperature      <dbl> 40.9, 35.7, 43.1, 55.7, 63.1, 71.0, 77.9, 75.8, 66.6,~
```

Let's use `summarize()` to check the range of the data, and check for missing (`NA`) data. For non-numeric columns (`date`, `commissioner`), `min()` , `mean()` and `max()` do not apply, but we can still produce the count columns.

```
data %>%
  summarize(across(where(is.numeric),
                   list(min = min,
                        mean = mean,
                        max = max,
                        count = ~ sum(!is.na(.x)),
                        nacount = ~ sum(is.na(.x))
                        )),
            across(where(~!is.numeric(.)),
                   list(count = ~ sum(!is.na(.x)),
                        nacount = ~ sum(is.na(.x))
                        ))
            ) %>%
  t() %>%
  present_table(n = Inf,
                caption = "Final data summary.")
```

Table 8: Final data summary.

incident_reports_min	36.00000

incident_reports_mean	105.00000
incident_reports_max	243.00000
incident_reports_count	204.00000
incident_reports_nacount	0.00000
temperature_min	23.90000
temperature_mean	56.07255
temperature_max	81.30000
temperature_count	204.00000
temperature_nacount	0.00000
date_count	204.00000
date_nacount	0.00000
commissioner_count	204.00000
commissioner_nacount	0.00000

In addition to the manual check for missing data via summaries, let's repeat the automatic missing value check from earlier, which will halt execution if there are any `NA`s in the data.

```
if(data %>% has_na()) {
  stop("NA values found in dataset. Please fix.")
}
```

Modelling

First, I will build separate models from each of the three features. Then, I will look at how we can combine them into a single model.

Model ▶ v0: log(Incident reports) ~ temperature

Let's build a model using `temperature` as the only predictor for `log(incident_reports)`.

```
# fit the model
model_v01 <- lm(log(incident_reports) ~ temperature, data)

# generate model performance plot
data %>%
  plot_model_performance(model_v01,
                         incident_reports,
                         title = "log(incident_reports) ~ temperature",
                         ymin = perf_plot_ymin,
                         ymax = perf_plot_ymax)
```

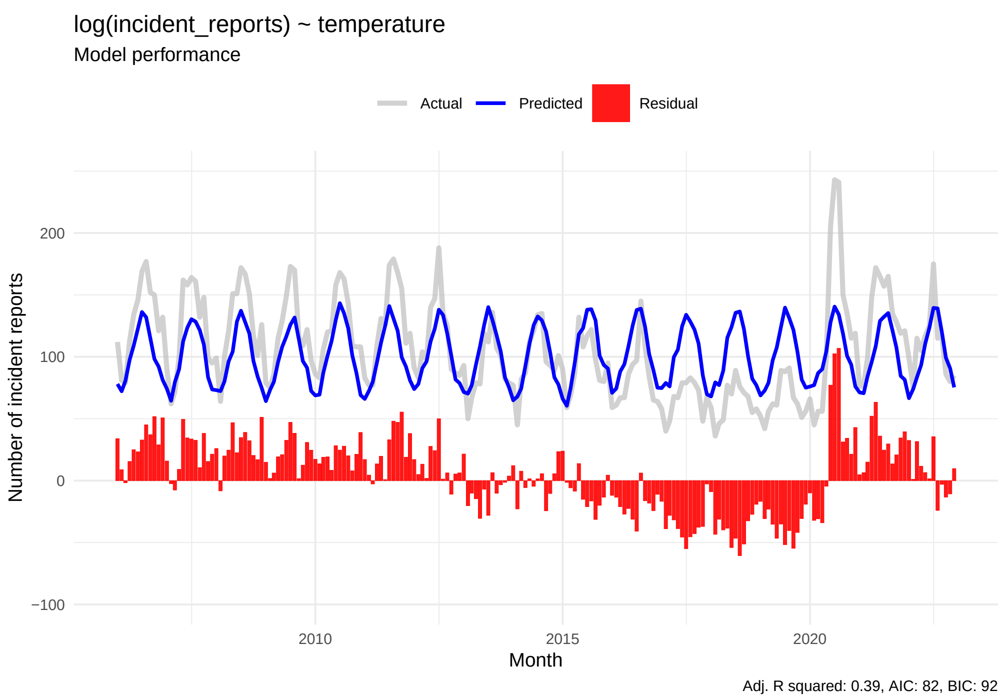

Figure 8: Model with one feature: ‘temperature‘.

As anticipated, this model captures seasonal variation fairly well, but it fails to capture longer-term trends.

Model ▶ v0: log(Incident reports) ~ date

Now let's build a model using `date` as the only predictor for `log(incident_reports)`.

```r
# fit the model
model_v02 <- lm(log(incident_reports) ~ date, data)

# generate model performance plot
data %>%
  plot_model_performance(model_v02,
                         incident_reports,
                         title = "log(incident_reports) ~ date ",
                         ymin = perf_plot_ymin,
                         ymax = perf_plot_ymax)
```

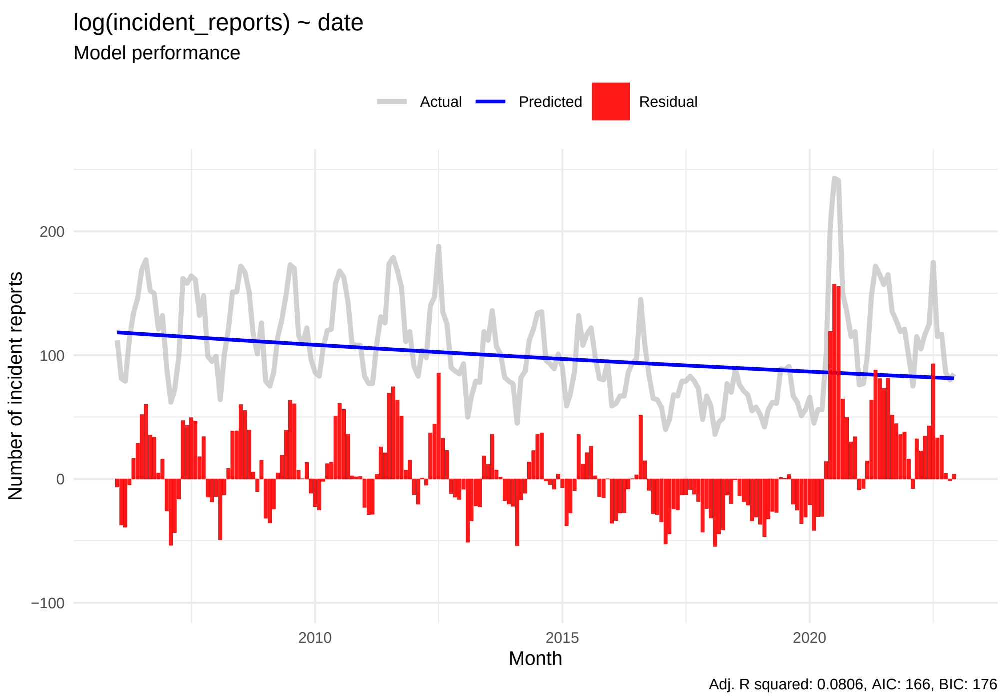

Figure 9: Model with one feature: 'date'.

This model is able to capture some aspects of the longer-term trend, but all the model does is scale and shift the linear variable `date`, so it is not able to handle things such as the increase in incidents from 2020 onwards, or the seasonal oscillations.

Model ▶ v0: log(Incident reports) ~ commissioner

Let's also build a model using `commissioner` as the only predictor for `log(incident_reports)`.

```
# fit the model
model_v03 <- lm(log(incident_reports) ~ commissioner, data)

# generate model performance plot
data %>%
  plot_model_performance(model_v03,
                         incident_reports,
                         title = "log(incident_reports) ~ commissioner",
                         ymin = perf_plot_ymin,
                         ymax = perf_plot_ymax)
```

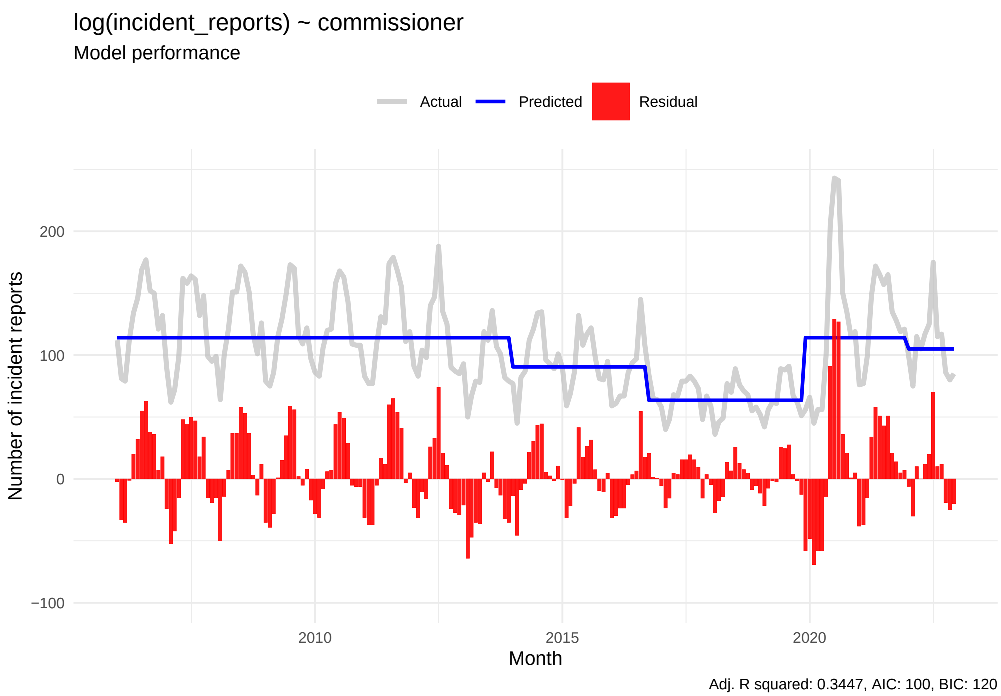

Figure 10: Model with one feature: 'commissioner'.

This model is able to make an upwards adjustment from 2020 onwards, because it turns out there was a change in commissioners between 2019 and 2020. But again, it does not have enough degrees of freedom to capture the seasonal oscillations.

Model ► v1: Modelling the trend

Next, let's combine features to create a model that accounts for trends only (not seasonality). Earlier, I hypothesized that the trend we see in `incident_reports` may be related to the passage of time (i.e. `date`), and the police `commissioner` in charge at the time. We can model this relationship as follows:

```r
model_v1 <- lm(log(incident_reports) ~ commissioner + date, data)

data %>%
  plot_model_performance(model_v1,
                         incident_reports,
                         title = paste("log(incident_reports) ~ commissioner",
                                       "+ date"),
                         ymin = perf_plot_ymin,
                         ymax = perf_plot_ymax)
```

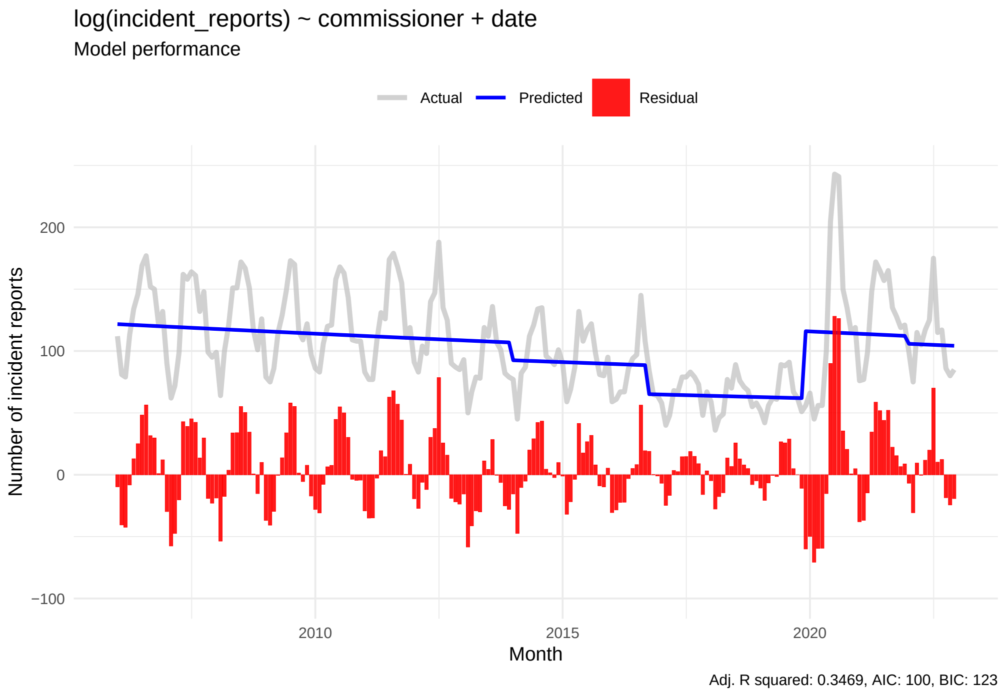

Figure 11: Modelling the trend with a universal slope, and per-commissioner intercepts.

This does a decent job of capturing the trend in `incident_reports`, but one problem is that all segments of the line have the same slope. It would be better if each segment could have its own slope.

Model ▶ v2: Improving the trend model with an interaction term

We can achieve per-commissioner slopes by adding an interaction term between `commissioner:date`.

```
model_v2 <- lm(log(incident_reports) ~ commissioner + commissioner:date, data)

data %>%
  plot_model_performance(model_v2,
                         incident_reports,
                         title = paste("log(incident_reports) ~ commissioner",
                                       "+ commissioner:date"),
                         ymin = perf_plot_ymin,
                         ymax = perf_plot_ymax)
```

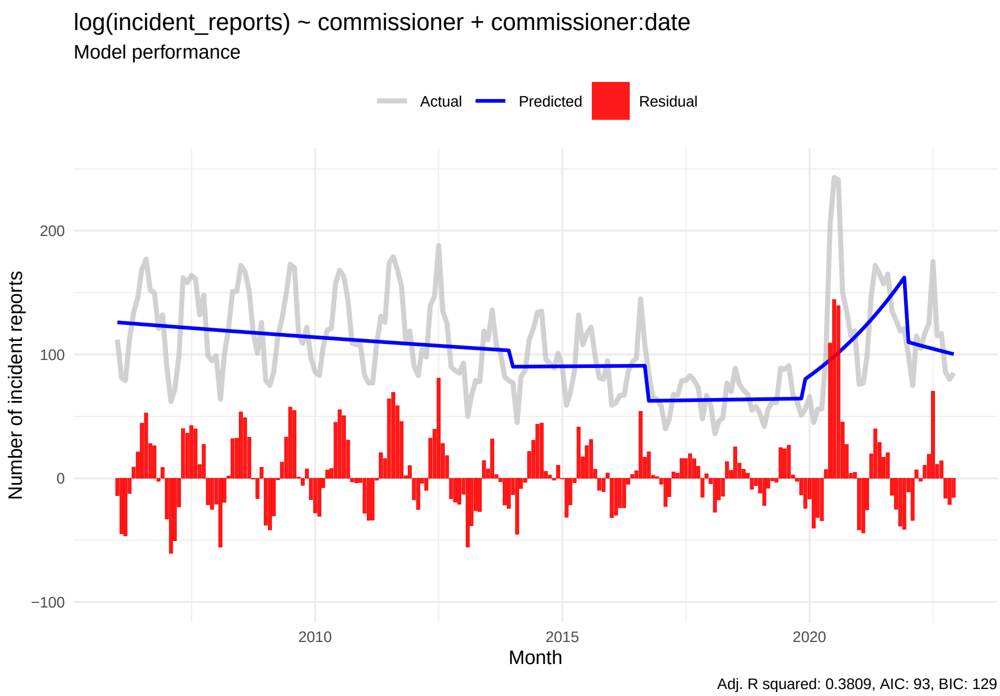

Figure 12: A 'commissioner:date' interaction term allows for per-commissioner slopes in addition to the per-commnissioner intercepts we had previously.

This looks better. All line segments have their own slope now. Let's move on to adding seasonality into the model.

Model ▶ v3: Incorporating seasonality

The variable I have in my dataset that potentially explains the seasonal oscillations is `temperature`. Let's add it into the model:

```
model_v3 <- lm(log(incident_reports) ~ commissioner + commissioner:date +
                 temperature,
               data)

data %>%
  plot_model_performance(model_v3,
                         incident_reports,
                         title = paste("log(incident_reports) ~ commissioner +",
                                       "commissioner:date + temperature"),
                         ymin = perf_plot_ymin,
                         ymax = perf_plot_ymax)
```

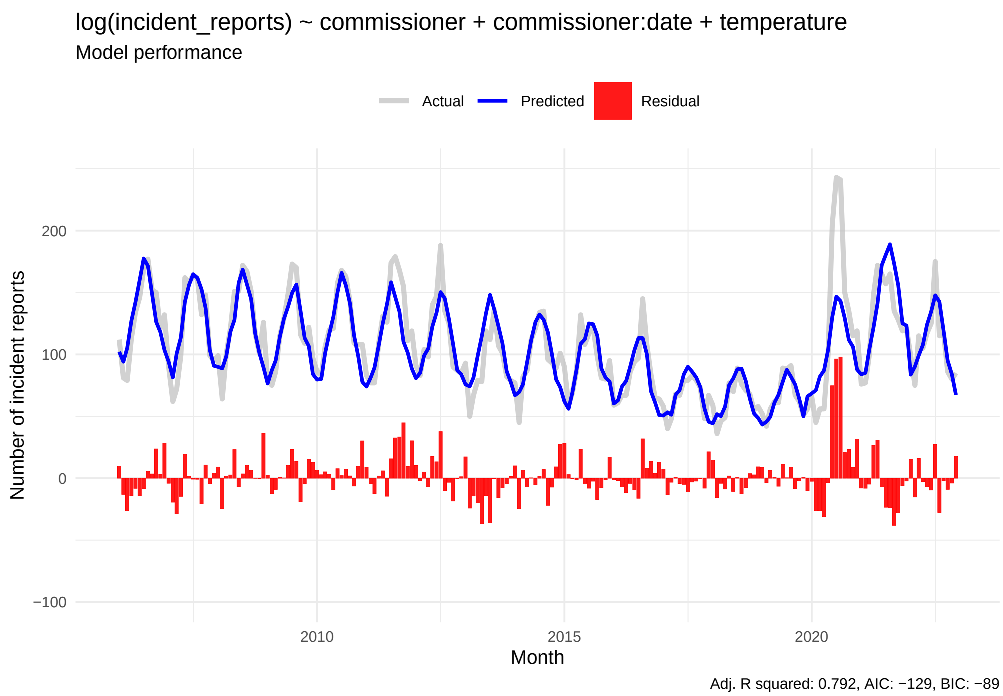

Figure 13: Incorporating seasonality with the addition of the temperature term.

Given that the model only relies on only 3 explanatory variables, an adjusted R squared of 79.2%[8] is great! The model underestimates incident report counts in 2020, but otherwise, it gets close to predicting the correct numbers of incident reports, with no obvious bias towards over- or under-estimating.

Model weakness analysis

According to the Wikipedia article on the COVID-19 pandemic in NYC,[9] there were multiple unusual police-related happenings at the start of the pandemic. Firstly, an unusually high percentage of the police force was out sick. Secondly, some prisoners were released early to reduce the risk of them contracting COVID-19. Separately, I know that there were changes to many people's living and employment situations, such as lockdowns, working from home, and increased unemployment. This information is not captured in the

[8] 79.2% of the variance in `log(incident_reports)` is explained by `commissioner`, `commissioner:date` and `temperature`

[9] https://en.wikipedia.org/wiki/COVID-19_pandemic_in_New_York_City#Police_and_crime

simple dataset I have collected, so it is unsurprising that the model struggles to make accurate predictions in this time period.

We can remove the time period in question, to see if model performance is better under "normal" conditions:

Model ▶ v3: Performance excluding Covid-19 time period

```
filtered_data <- data %>%
  dplyr::filter(date < covid_start_date | date > covid_end_date)

model_v3_filtered <- lm(
  log(incident_reports) ~ commissioner + commissioner:date +
    temperature,
  filtered_data)

filtered_data %>%
  plot_model_performance(model_v3_filtered,
                         incident_reports,
                         title = paste("log(incident_reports) ~ commissioner +",
                                       "commissioner:date + temperature"),
                         subtitle = paste("Model performance excluding the",
                                          "first 10 months of the pandemic"),
                         ymin = perf_plot_ymin,
                         ymax = perf_plot_ymax)
```

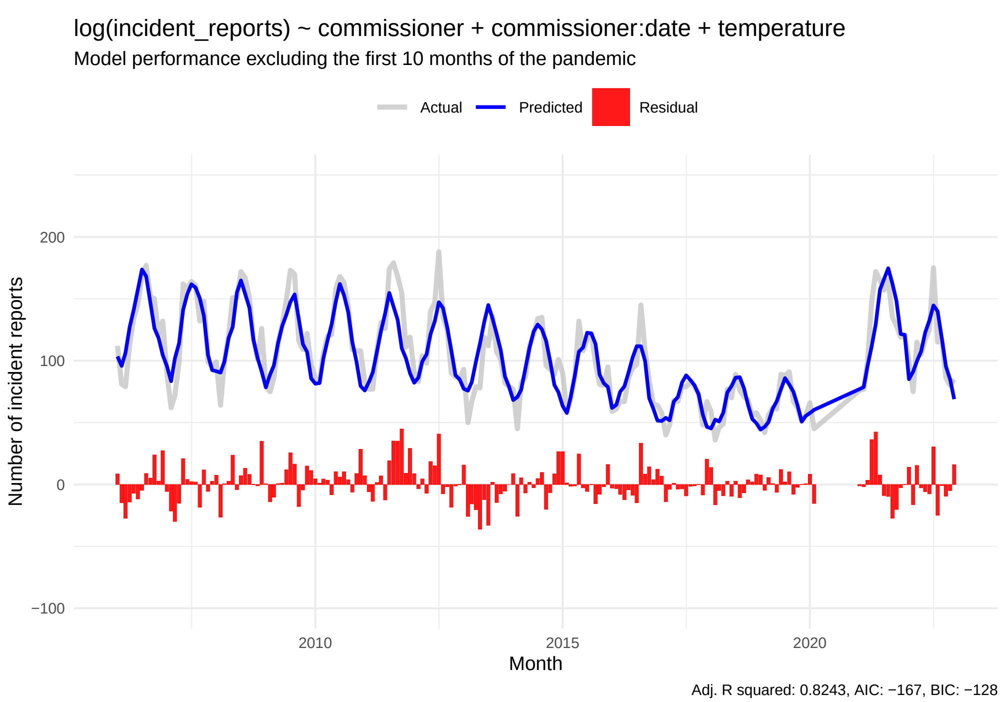

Figure 14: Ignoring the anomalous Covid-19/lockdown period, from March to December 2020.

The model's performance did indeed improve to an adjusted R squared of 82.43%.

Model ▶ Performance

We can summarize the model and its performance using some standard R functions/visualizations:

Summary

```r
summary(model_v3_filtered)
```

```
##
## Call:
## lm(formula = log(incident_reports) ~ commissioner + commissioner:date +
##     temperature, data = filtered_data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.45146 -0.07651  0.00172  0.08263  0.36412
##
## Coefficients:
##                                           Estimate Std. Error t value Pr(>|t|)
## (Intercept)                              -1.402e+01  3.181e+00  -4.407 1.78e-05
## commissionerJames P. O'Neill              1.896e+01  3.430e+00   5.527 1.11e-07
## commissionerKeechant Sewell               3.666e+01  8.610e+00   4.258 3.29e-05
## commissionerRaymond Walter Kelly          1.914e+01  3.196e+00   5.988 1.10e-08
## commissionerWilliam Joseph Bratton        2.185e+01  3.504e+00   6.235 3.05e-09
## temperature                               1.440e-02  7.167e-04  20.092  < 2e-16
## commissionerDermot F. Shea:date           9.589e-04  1.705e-04   5.625 6.86e-08
## commissionerJames P. O'Neill:date        -9.037e-05  7.421e-05  -1.218  0.22488
## commissionerKeechant Sewell:date         -9.813e-04  4.196e-04  -2.339  0.02043
## commissionerRaymond Walter Kelly:date    -8.135e-05  1.841e-05  -4.417 1.71e-05
## commissionerWilliam Joseph Bratton:date -2.501e-04  9.225e-05  -2.711  0.00734
##
## (Intercept)                              ***
## commissionerJames P. O'Neill             ***
## commissionerKeechant Sewell              ***
## commissionerRaymond Walter Kelly         ***
## commissionerWilliam Joseph Bratton       ***
## temperature                              ***
## commissionerDermot F. Shea:date          ***
## commissionerJames P. O'Neill:date
## commissionerKeechant Sewell:date         *
## commissionerRaymond Walter Kelly:date    ***
## commissionerWilliam Joseph Bratton:date **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1521 on 183 degrees of freedom
## Multiple R-squared:  0.8334, Adjusted R-squared:  0.8243
## F-statistic: 91.52 on 10 and 183 DF,  p-value: < 2.2e-16
```

Glance

```r
broom::glance(model_v3_filtered) %>%
  t() %>%
  present_table(n = Inf,
                caption = paste("Filtered v3 model's performance across",
                                "various metrics, courtesy of",
                                "`broom::glance()`."))
```

Table 9: Filtered v3 model's performance across various metrics, courtesy of `broom::glance()`.

r.squared	0.8333677
adj.r.squared	0.8242621
sigma	0.1521045
statistic	91.5226187
p.value	0.0000000
df	10.0000000
logLik	95.7264308
AIC	-167.4528615
BIC	-128.2385636
deviance	4.2338455
df.residual	183.0000000
nobs	194.0000000

Plots

There are six standard model diagnostics plots that we can draw, just by passing the fitted model into the `plot()` function. Plots are shown in Appendix B.

Conclusion

Conclusion ► Model findings

I built a model to estimate monthly `incident_reports`, using `temperature`, `date`, and NYPD `commissioner` as explanatory variables. I was able to explain 79.2% of the variance in `log(incident_reports)` using this approach (82.43% if the anomalous Covid-19 period from March to December 2020 is excluded). These findings could be used as the basis for a predictive model, or for more fine-grained analyses (daily data, precise locations, types of incidents, etc).

Follow-up research could dig further into the relationship between `commissioner` and `incident_reports`, to see what is driving it, and whether there are any changes that could be made to data collection or analysis to better reflect the true number of shooting-related incidents in the city.

Conclusion ► Bias

Bias is important to consider during data analysis and modelling. It may not always be possible (or even desirable) to eliminate *all* bias (see discussion below on model bias). But at a minimum we should aware of possible biases in our work, including personal bias and the biases passed on to us by others (e.g. through the data) so that we can mitigate any negative effects of the biases.

Bias ► Personal

My modelling biases For this project, we were asked to pay attention to our personal biases, and take steps to mitigate them. I know that in the past, I have had a bias towards creating the "best" model possible. (As I am sure is true for many new data scientists!) I understood "best" to mean:

- Model performance metrics are as high as they can possibly be. I would want to keep on tuning the model to squeeze out every last drop of performance.
- Model is incredibly detailed. I saw being able to predict **hourly** incident report numbers as clearly better than only being able to predict **daily** or **monthly** numbers. And I would have considered making predictions at a **per-borough** level as a better starting place than zooming out to NYC as a **whole**. (After all, we could always add up the borough predictions to get a citywide prediction.)

I think this previous bias towards prioritization of numbers and details came at the expense of gaining a clear understanding of the problem. And possibly even at the expense of developing robust solutions.

Mitigation Therefore, I tried to mitigate my bias tendencies by deliberately staying high-level, predicting monthly incident report numbers for NYC as a whole. (Note that even my choice to use this mitigation strategy was a personal bias!) I think this approach had three additional benefits:

- Aggregating to monthly and to citywide meant that the law of large numbers was more likely to apply, allowing me to get good performance while using simple models.
- This approach allowed me to avoid introducing biases that might have crept into more fine-grained models. For example, if I had included variables such as victim/perpetrator demographics, location, etc, I would have needed to decide how to handle cases of missing values. If I wanted to look at weekdays vs weekends, I would have needed to decide on the definition of "weekend" (e.g. does Sunday night count?) All of those decisions could have been additional sources of bias.
- By not prioritizing model performance above all else, I was able to land on a parsimonious model (only 3 features, all of which are relatively predictable) that does a good job of explaining the variance in the response variable. Fewer, simpler features means a more robust model that is less likely to break after it is put into production.

Bias ► Data

Problem Another category of bias is the bias implicit in a dataset. It is tricky to deal with, because without additional information, it may be difficult to detect the bias in the first place. And even if we knew of the existence of some bias, it might not be clear what should be done about it.

For example, what if some demographics of victims or perpetrators are more or less likely to have incident reports filed than others? How could we detect this, and how could we address or compensate for it? Each of those questions is likely the subject of an entire study.

As another example, we saw that there is a strong relationship between `commissioner` and `incident_reports`. Without additional information about how the incident reports data is collected, it is difficult to tell what is responsible for this relationship. Maybe it relates to real-world changes, such as a change in policing policy. Or maybe different commissioners have different standards for data collection, i.e. each commissioner's policies on incident reporting may be a source of bias.

Mitigation To address this issue, I tried to make clear at all stages that I was modelling count of incident **reports** rather than count of incidents themselves, knowing that there may be a biased difference between the two. I think determining causality here would require outside research that is beyond the scope of this project. But at least my analysis highlighted the relationship, so that it can be followed up on.

Conclusion ▶ Next steps

If I were to continue on with this analysis, the next thing I would do is to convert my modelling to the `tidymodels` paradigm. Although there is a bit of a learning curve to get started, it should pay off in the long run since the framework is so widely applicable. `tidymodels` was designed to provide guide rails that promote good data modelling practices, as well as guard against common pitfalls. Using standardized tooling offloads some of the decision-making and implementation work we would otherwise need to do, freeing up our minds to focus on modelling questions (including potential sources of bias).

I would also use `renv` to further improve reproducibility (I didn't want to try it here, as the increased complexity might make it more difficult for peers to knit my file).

Appendix

Appendix A ► Session Info

```r
sessionInfo()
```

```
## R version 4.3.2 (2023-10-31)
## Platform: x86_64-apple-darwin20 (64-bit)
## Running under: macOS Monterey 12.7.3
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.3-x86_64/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.3-x86_64/Resources/lib/libRlapack.dylib;  LAPACK v
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## time zone: America/Edmonton
## tzcode source: internal
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] pdftools_3.4.0  modelr_0.1.11   knitr_1.45      broom_1.0.5
##  [5] lubridate_1.9.3 forcats_1.0.0   stringr_1.5.1   dplyr_1.1.4
##  [9] purrr_1.0.2     readr_2.1.5     tidyr_1.3.1     tibble_3.2.1
## [13] ggplot2_3.5.0   tidyverse_2.0.0 rlang_1.1.3
##
## loaded via a namespace (and not attached):
##  [1] utf8_1.2.4        generics_0.1.3    lattice_0.22-6    stringi_1.8.3
##  [5] hms_1.1.3         digest_0.6.34     magrittr_2.0.3    evaluate_0.23
##  [9] grid_4.3.2        timechange_0.3.0  fastmap_1.1.1     Matrix_1.6-1.1
## [13] backports_1.4.1   mgcv_1.9-0        fansi_1.0.6       scales_1.3.0
## [17] cli_3.6.2         crayon_1.5.2      splines_4.3.2     bit64_4.0.5
## [21] munsell_0.5.0     withr_3.0.0       yaml_2.3.8        parallel_4.3.2
## [25] tools_4.3.2       tzdb_0.4.0        colorspace_2.1-0  curl_5.2.0
## [29] vctrs_0.6.5       R6_2.5.1          lifecycle_1.0.4   bit_4.0.5
## [33] vroom_1.6.5       pkgconfig_2.0.3   pillar_1.9.0      gtable_0.3.4
## [37] glue_1.7.0        Rcpp_1.0.12       highr_0.10        xfun_0.43
## [41] tidyselect_1.2.0  rstudioapi_0.15.0 farver_2.1.1      nlme_3.1-163
## [45] htmltools_0.5.7   labeling_0.4.3    rmarkdown_2.25    qpdf_1.3.2
## [49] compiler_4.3.2    askpass_1.2.0
```

Appendix B ► Model diagnostics

Here is the output of `plot.lm()` (i.e. `plot(fitted_model)`) for the final model.

```r
plot(model_v3_filtered, which=1:6)
```

:::

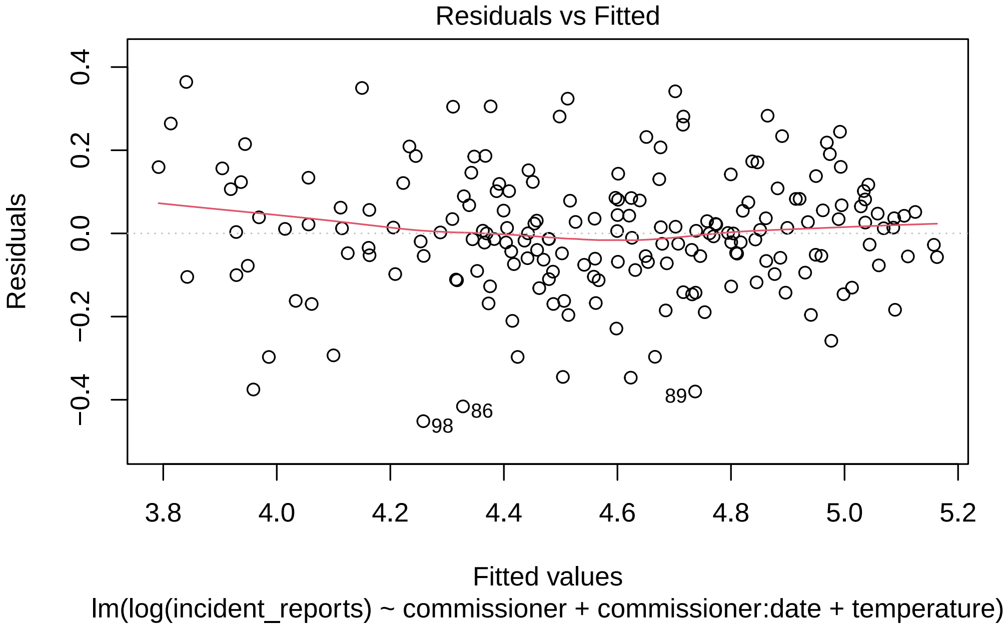

Figure 15: Residuals vs Fitted

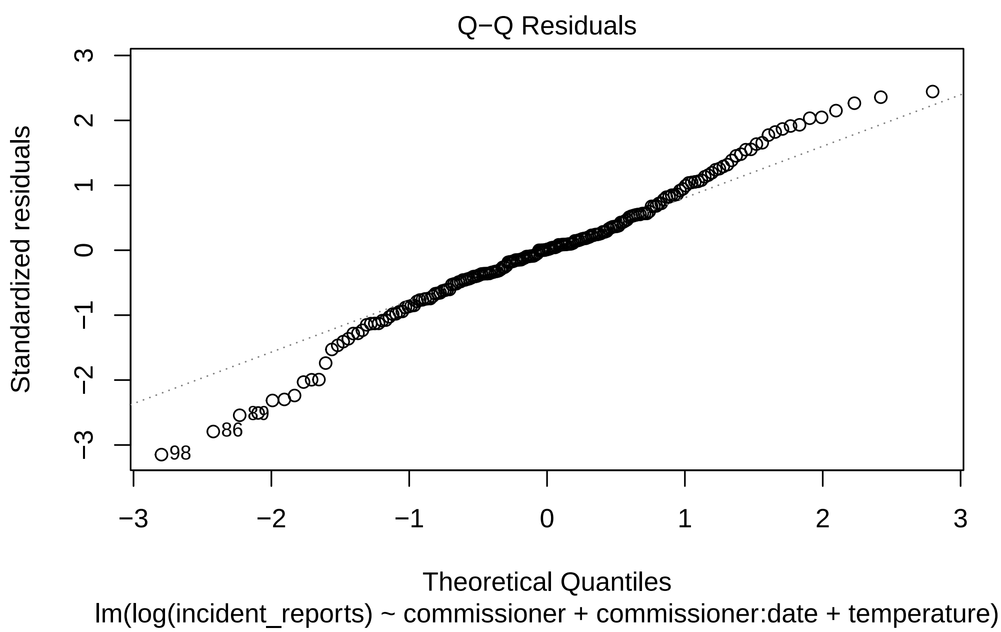

Figure 16: Normal Q-Q

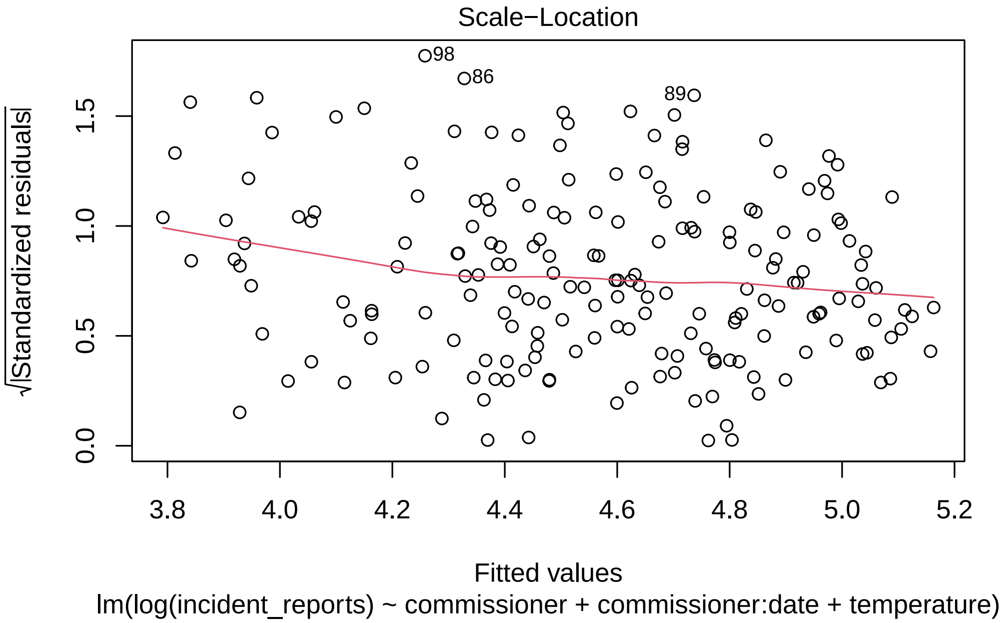

Figure 17: Scale-Location

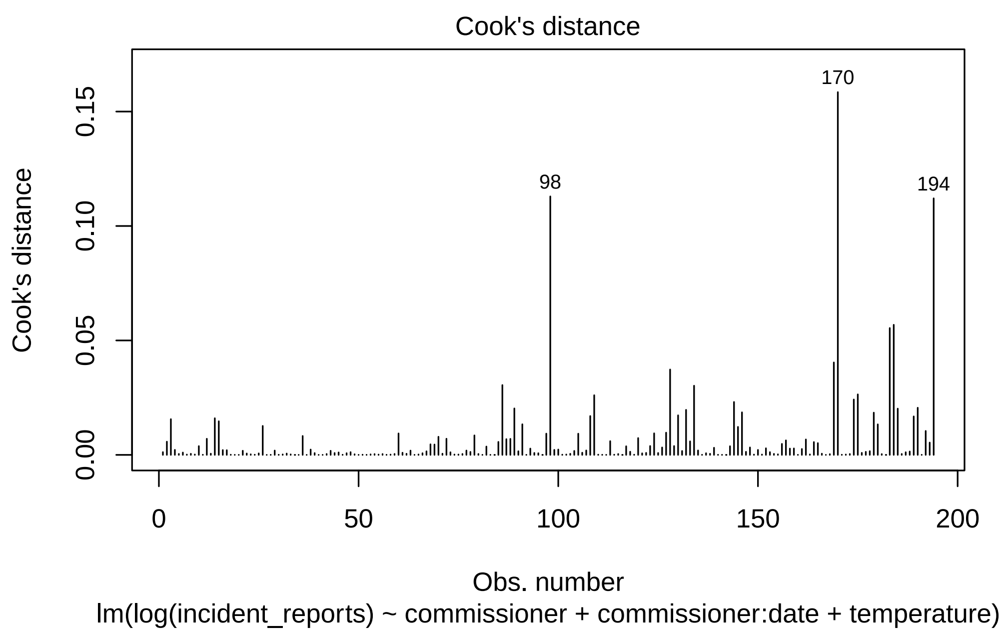

Figure 18: Cook's distance

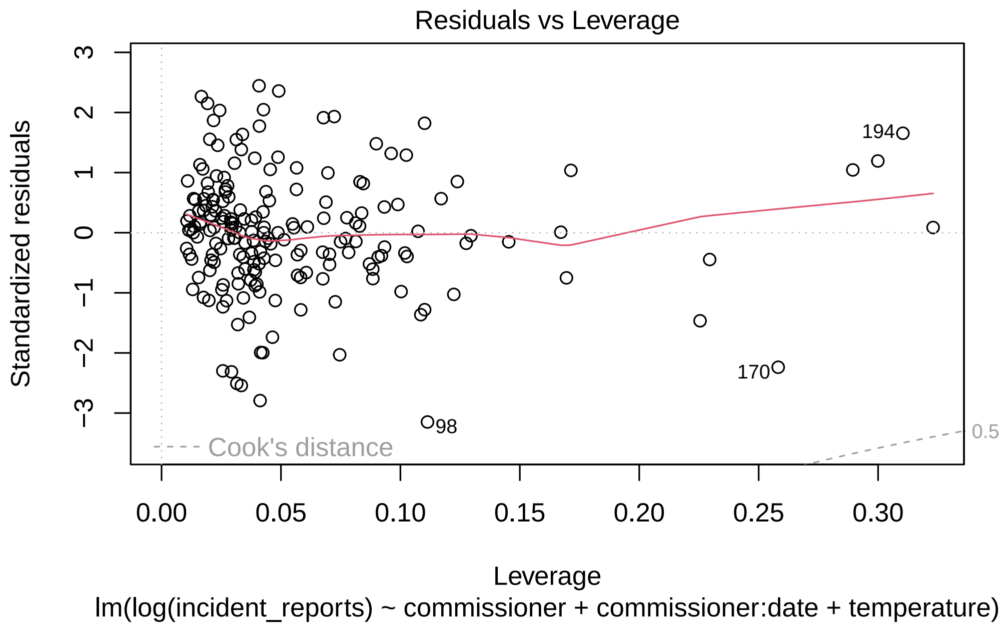

Figure 19: Residuals vs Leverage

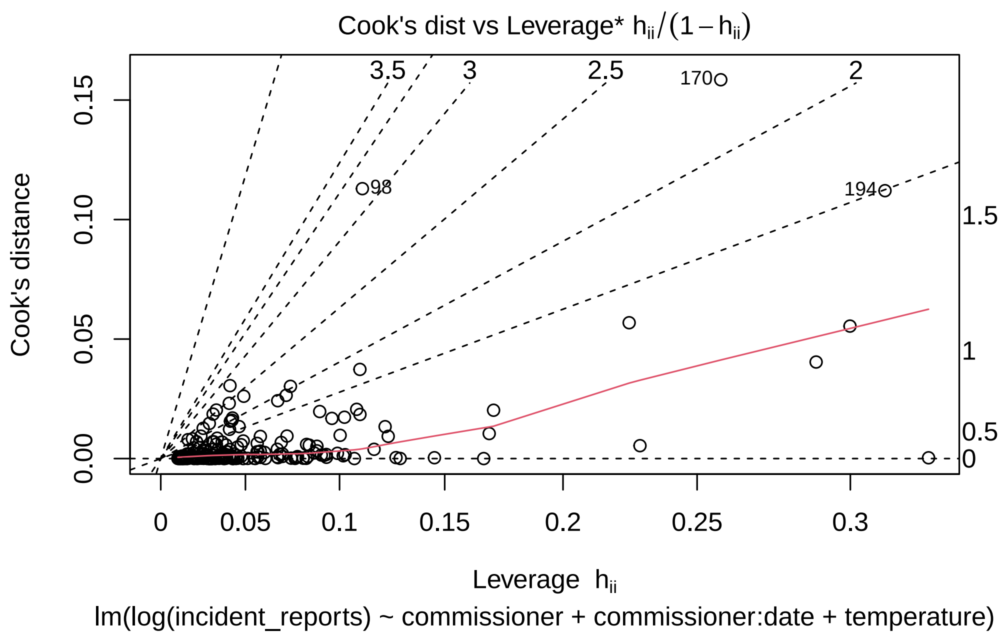

Figure 20: Cook's dist vs Leverage $h_{ii}/(1 - h_{ii})$