PS1

AUTHOR
Luis Señires

PUBLISHED
October 5, 2024

1. **PS1:** Due Sat Oct 5 at 5:00PM Central. Worth 50 points.

We use (`*`) to indicate a problem that we think might be time consuming.

Steps to submit (5 points on PS1)

1. "This submission is my work alone and complies with the 30538 integrity policy." Add your initials to indicate your agreement: ****
2. "I have uploaded the names of anyone I worked with on the problem set **here**" **** (1 point)
3. Late coins used this pset: **\0\0** Late coins left after submission: **\0\4**
4. Knit your `ps1.qmd` to HTML
5. Convert your HTML to PDF by printing to PDF from a browser.
6. Submit resulting ps1.pdf to Gradescope (4 points)
7. Tag your submission in Gradescope

```python
# set up
import pandas as pd
import altair as alt

import warnings
warnings.filterwarnings('ignore')
```

Read in one percent sample (15 Points)

1.

```python
import time

path = "/Users/LUIS/Documents/GitHub/ppha30538_fall2024/problem_sets/ps1/data/parking_ticket

# Measure time to read data
start = time.time()
df = pd.read_csv(path)
end = time.time()
run_time = round(end - start, 2)
print(run_time, "seconds")

# https://pynative.com/python-get-execution-time-of-program/
```

```
1.85 seconds
```

```
# Verify number of rows
assert len(df.index) == 287458
```

2.

```
import os

# Calculate filesize
size = os.stat(path).st_size/1000000
size = round(size, 4)
print(size, "megabytes")

# https://www.digitalocean.com/community/tutorials/how-to-get-file-size-in-python
```

```
83.9428 megabytes
```

Given our dataset represents a one percent sample of the full dataset, we can predict the full dataset to be around 100 times larger, or about 8.39 GB.

3.

```
# Check columns
df.head()
```

	Unnamed: 0	ticket_number	issue_date	violation_location	license_plate_number
0	1	51482901.0	2007-01-01 01:25:00	5762 N AVONDALE	d41ee9a4cb0676e641399ad14aaa20d06f2c6896de
1	2	50681501.0	2007-01-01 01:51:00	2724 W FARRAGUT	3395fd3f71f18f9ea4f0a8e1f13bf0aa15052fc8e5605
2	3	51579701.0	2007-01-01 02:22:00	1748 W ESTES	302cb9c55f63ff828d7315c5589d97f1f8144904d66
3	4	51262201.0	2007-01-01 02:35:00	4756 N SHERIDAN	94d018f52c7990cea326d1810a3278e2c6b1e8b44f
4	5	51898001.0	2007-01-01 03:50:00	7134 S CAMPBELL	876dd3a95179f4f1d720613f6e32a5a7b86b0e6f988

5 rows × 24 columns

Looking at a snapshot of the data, it seems the rows are sorted in increasing order (with a minimum

Looking at a snapshot of the data, it seems the rows are sorted in increasing order (with a minimum value of 1 and increasing by 1 per row) by the "Unnamed: 0" column. To test:

```
# Create a subset of the data
df_500 = df[0:500]

# Define function
def order_test(df):
    return all(x < y for x, y in zip(df["Unnamed: 0"], df["Unnamed: 0"][1:]))

# Test if column is ordered
order_test(df_500)

# https://stackoverflow.com/questions/4983258/check-list-monotonicity
```

```
True
```

Cleaning the data and benchmarking (15 Points)

1.

```
# Count the number of tickets issued in 2017
df["issue_date"] = pd.to_datetime(df["issue_date"])
df[df["issue_date"].dt.year == 2017].count()["issue_date"]
```

```
np.int64(22364)
```

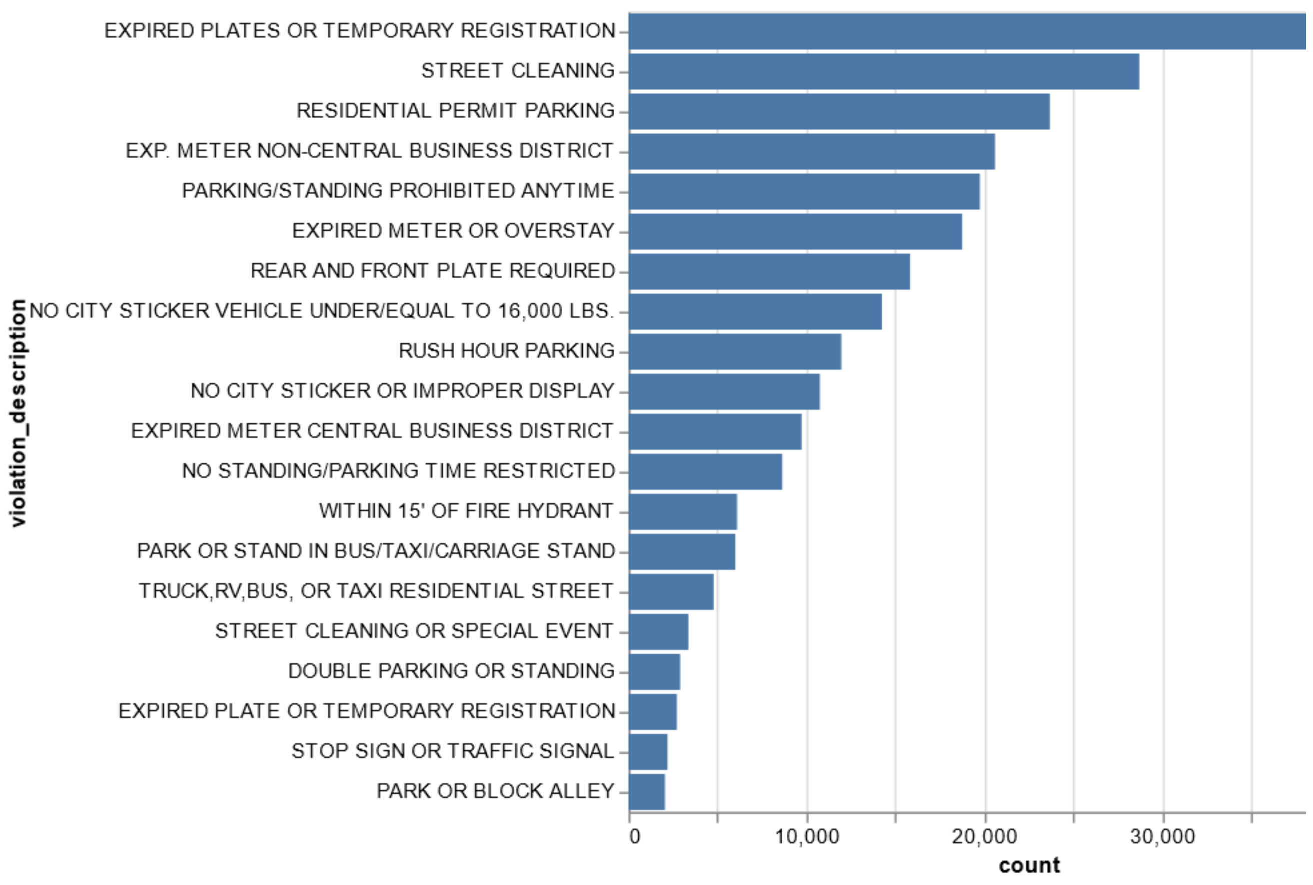
In our sample dataset, 22,364 tickets were issued in 2017 which implies that around 2.24 million tickets were issued in the same year if we use the full dataset. On the other hand, ProPublica reported that the City of Chicago issues more than 3 million tickets annually. I think there is a meaningful difference between the two sources as the ProPublica figure is inflated by about 34% of the actual number.

2.

```
# Determine violation frequency and subset top 20 violations
df_violation = df.groupby(["violation_code", "violation_description"]).size()
df_violation = df_violation.reset_index(name = "count")
df_violation = df_violation.sort_values("count", ascending = False)
df_violation = df_violation[0:20]
```

```
# Graph violation frequency
chart_violation = alt.Chart(df_violation).mark_bar().encode(
    alt.X("count"),
    alt.Y("violation_description")
        .sort(field = "-count")
        .axis(labelLimit = 0, titlePadding = 100)
).properties(
    width = 400
)
```

```
)

chart_violation
```



Visual Encoding (15 Points)

1.

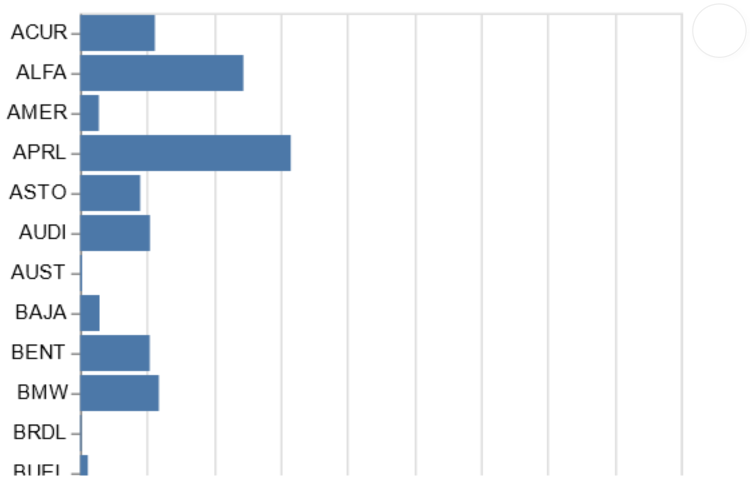
```
# Create df with column names as row values
columns = df.columns
df_types = pd.DataFrame(columns)

# Manually associate data types
df_types["types"] = ["O", "N", "T", "N", "N", "N", "N", "N", "N", "N", "N", "N", "N", "Q", '

# Print table
print(df_types.to_markdown())
```

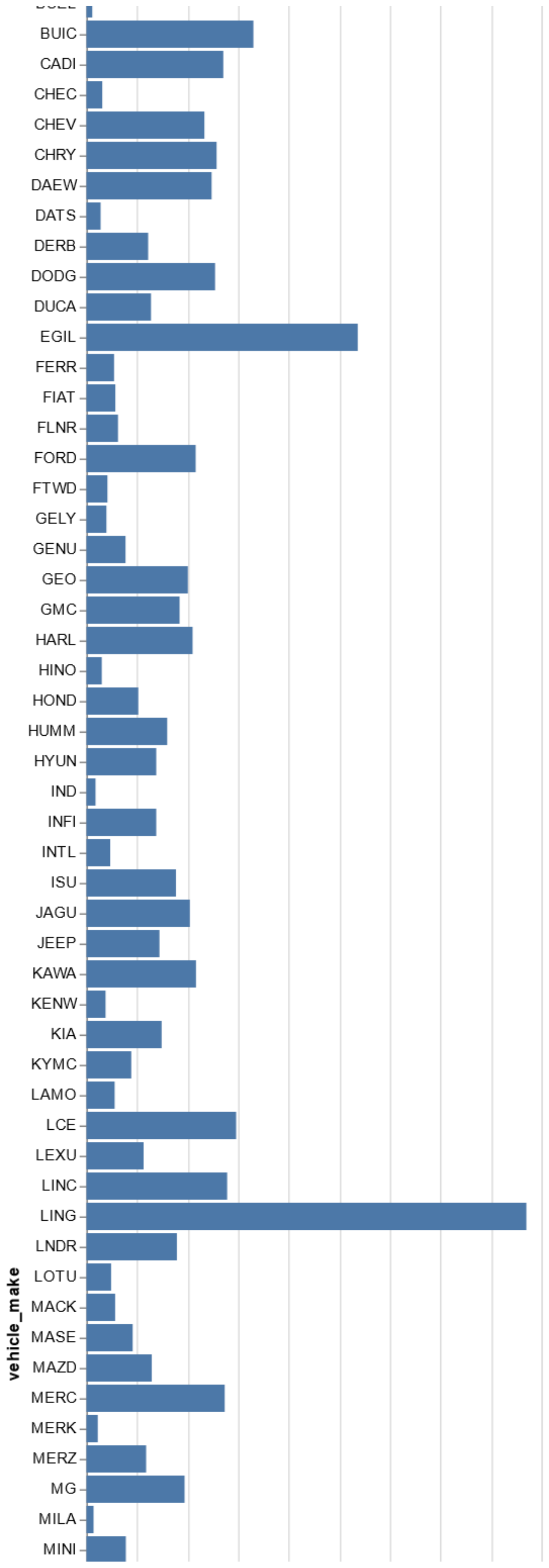
```
|    | 0                      | types   |
|---:|:-----------------------|:--------|
|  0 | Unnamed: 0             | O       |
|  1 | ticket_number          | N       |
|  2 | issue_date             | T       |
|  3 | violation_location     | N       |
|  4 | license_plate_number   | N       |
|  5 | license_plate_state    | N       |
|  6 | license_plate_type     | N       |
|  7 | zipcode                | N       |
|  8 | violation_code         | N        |
|  9 | violation_description  | N        |
| 10 | unit                   | N        |
| 11 | unit_description       | N        |
| 12 | vehicle_make           | N        |
| 13 | fine_level1_amount     | Q        |
| 14 | fine_level2_amount     | Q        |
| 15 | current_amount_due     | Q        |
| 16 | total_payments         | Q        |
| 17 | ticket_queue           | N        |
| 18 | ticket_queue_date      | T        |
| 19 | notice_level           | N        |
| 20 | hearing_disposition    | N        |
| 21 | notice_number          | N        |
| 22 | officer                | N        |
| 23 | address                | N        |
```

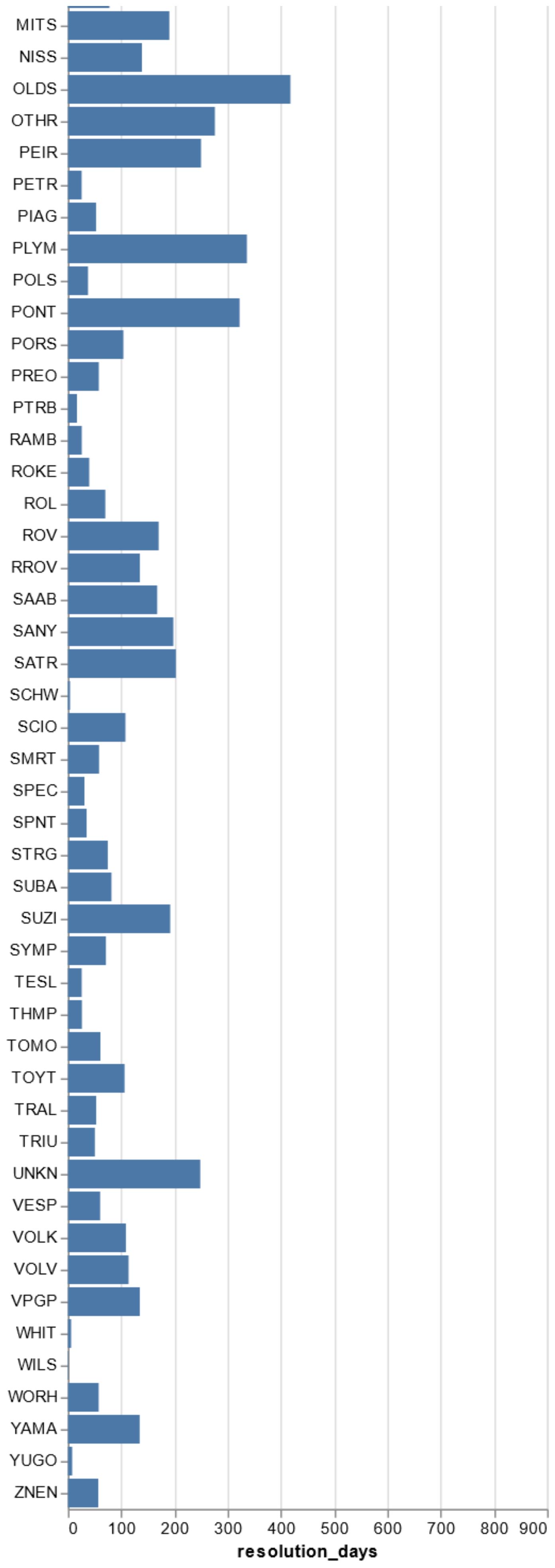
2.

```python
import numpy as np

# Calculate number of days to resolve for "Paid" tickets
df["ticket_queue_date"] = pd.to_datetime(df["ticket_queue_date"])
df["resolution_days"] = np.where(df["ticket_queue"] == "Paid", (df["ticket_queue_date"] - d

# Clean up data
df["resolution_days"] = np.where(df["resolution_days"] == -1, 0, df["resolution_days"])

# Group data
df_make = df.groupby("vehicle_make").mean("resolution_days").reset_index()
```

```python
# Graph resolution time
chart_resolution = alt.Chart(df_make).mark_bar().encode(
    alt.X("resolution_days"),
    alt.Y("vehicle_make")
        .axis(labelLimit = 0)
)

chart_resolution
```




vehicle_make
BUIC
CADI
CHEC
CHEV
CHRY
DAEW
DATS
DERB
DODG
DUCA
EGIL
FERR
FIAT
FLNR
FORD
FTWD
GELY
GENU
GEO
GMC
HARL
HINO
HOND
HUMM
HYUN
IND
INFI
INTL
ISU
JAGU
JEEP
KAWA
KENW
KIA
KYMC
LAMO
LCE
LEXU
LINC
LING
LNDR
LOTU
MACK
MASE
MAZD
MERC
MERK
MERZ
MG
MILA
MINI

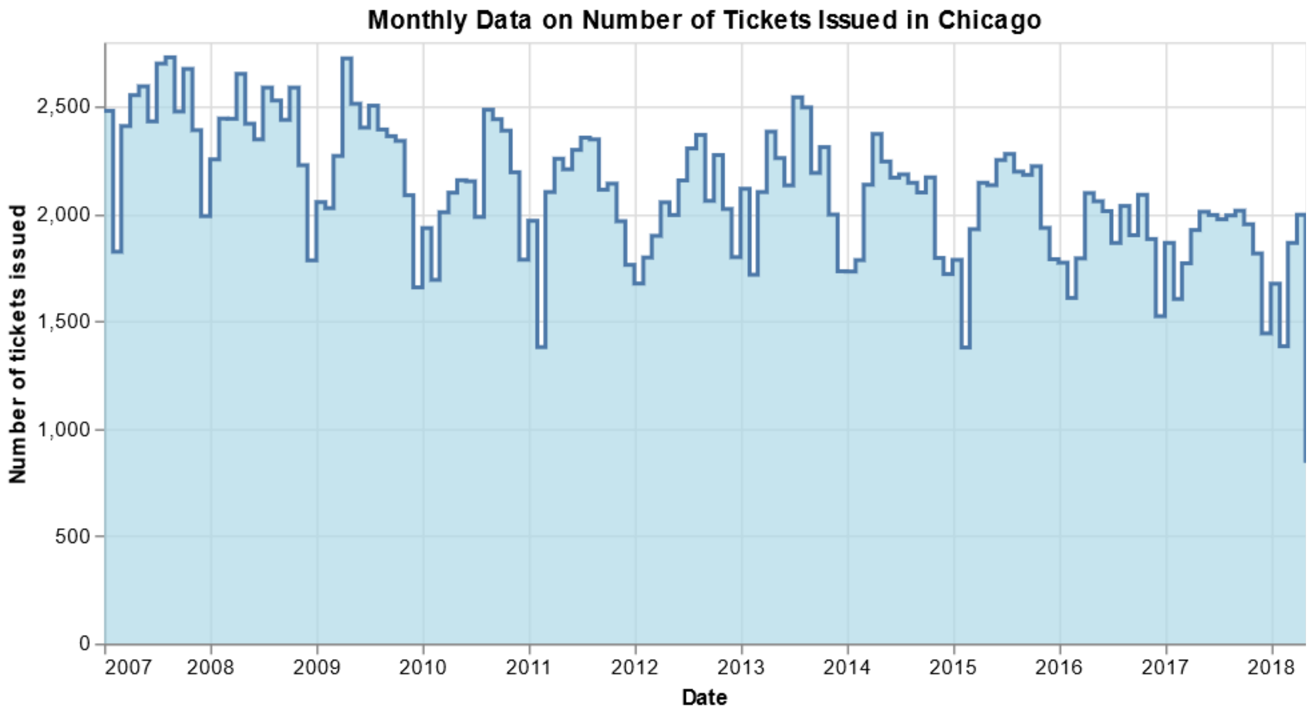


I think one possible explanation is that car quality (often determined by its make) is likely positively correlated with the financial capacity of its owner, i.e. luxury cars are owned by richer individuals, whereas cheaper cars are owned by less wealthy individuals. In such a case, we can expect tickets issued to luxury car owners to be resolved relatively more quickly as those owners would have greater financial capacity to afford paying fines. On the other hand, poorer individuals with cheaper cars will have a more difficult time to pay off any ticketing debt especially as fines get steeper and interest accumulates.

3.

```
# Create df with monthly count on number of tickets issued
df["issue_year"] = df["issue_date"].dt.year
df["issue_month"] = df["issue_date"].dt.month
df_yr_month = df.groupby(["issue_year", "issue_month"]).size()
df_yr_month = df_yr_month.reset_index(name = "count")
df_yr_month["combined"] = pd.to_datetime(df_yr_month["issue_year"].astype(str) +
df_yr_month["issue_month"].astype(str), format = "%Y%m")

# https://stackoverflow.com/questions/48304927/cleanly-combine-year-and-month-columns-to-sir
```

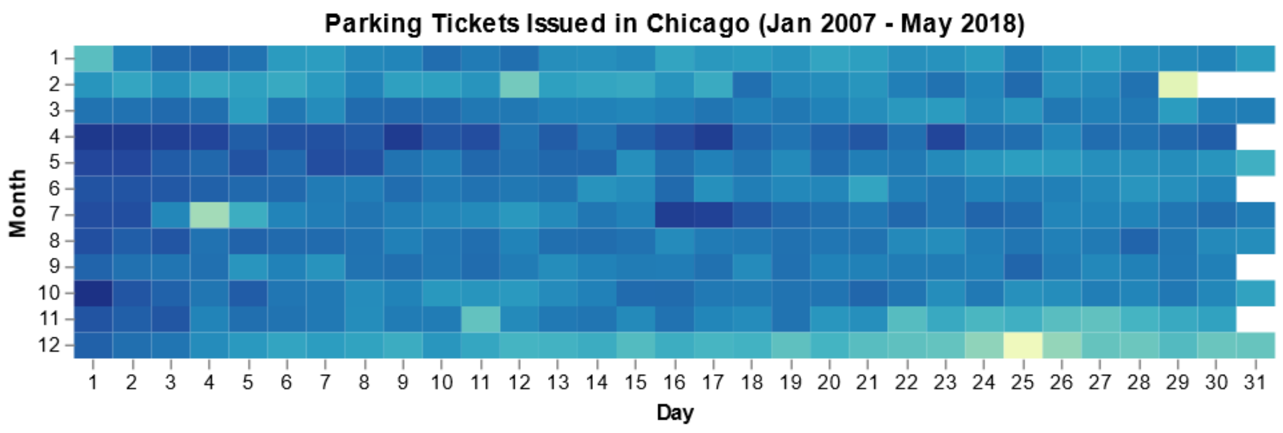
```
# Graph monthly count using filled step chart
chart_filled_step = alt.Chart(df_yr_month, title = "Monthly Data on Number of Tickets Issuec
    color="lightblue",
    interpolate='step-after',
    line=True
).encode(
    alt.X("combined").title("Date"),
    alt.Y("count").title("Number of tickets issued")
).properties(
    width = 600
)

chart_filled_step
```



4.

```
# Create df with month and day information
df["issue_day"] = df["issue_date"].dt.day
df_month_day = df.groupby(["issue_month", "issue_day"]).size()
df_month_day = df_month_day.reset_index(name = "count")
```

```
# Graph using annual weather heatmap
chart_heatmap = alt.Chart(df_month_day, title="Parking Tickets Issued in Chicago (Jan 2007 -
    alt.X("issue_day:O").title("Day").axis(labelAngle=0),
    alt.Y("issue_month:O").title("Month"),
    alt.Color("count").title("Count")
).configure_view(
    step = 13,
    strokeWidth = 0
).configure_axis(
    domain = False
).properties(
    width = 600
)

chart_heatmap
```

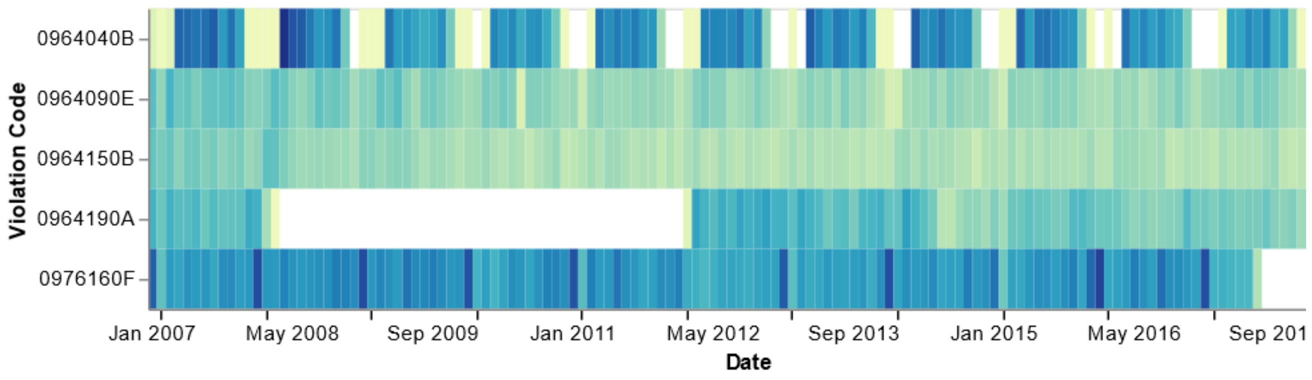


5.

```
# Determine top 5 violations
top5_violations = df_violation.head()["violation_code"]
df_top5 = df[df["violation_code"].isin(top5_violations)]

# Create df
df_top5_grouped = df_top5.groupby(["violation_code", "violation_description","issue_year", '
df_top5_grouped = df_top5_grouped.reset_index(name = "count")
df_top5_grouped["combined"] = pd.to_datetime(df_top5_grouped["issue_year"].astype(str) + df_
```

```
# Graph using lasagna plot
color_condition = alt.condition(
    "month(datum.value) == 1 && date(datum.value) == 1",
    alt.value("black"),
    alt.value(None),
)
```

```
chart_lasagna = alt.Chart(df_top5_grouped, width=500, height=150).mark_rect().encode(
    alt.X("yearmonth(combined):O").title("Date").axis(
      labelAngle = 0,
      labelOverlap = True,
      labelColor = "black",
      tickColor = color_condition,
    ),
    alt.Y("violation_code").title("Violation Code"),
    alt.Color("count").title("Count")
).properties(
    width = 600
)

chart_lasagna
```



6.

Filled step chart:

- Pros:
  - Easy to see highest and lowest points over time
  - Easier for estimating quantities
  - Useful for spotting trends
- Cons:
  - Not easy to compare same time periods across years (e.g. Jan 2007 vs Jan 2010)

Annual weather heatmap:

- Pros:
  - Helpful for seeing seasonal patterns/behaviors
- Cons:
  - Not useful for year on year comparisons
  - Not easy to measure quantities

Lasagna plot:

- Pros:
  - Useful for viewing evolution of multiple categories over time
  - Helpful for comparing across categories
- Cons:
  - Not easy to measure quantities

    - May be more helpful for viewing larger time periods (larger bins)

7.

To show uneven distribution of enforcement over time, I think the filled step chart would work best because it allows readers to visualize and quantify periodic fluctuations over time in a much easier way compared to the other two. While the lasagna plot also shows changes over time, it is also a much busier chart because its strength lies in cross-category comparisons, while the filled step chart is more simple.