30538 Problem Set 2: Parking Tickets

Peter Ganong, Maggie Shi, and Richard Chen

2026-01-07

Background

Read **this** article and **this** shorter article. If you are curious to learn more, **this** page has all of the articles that ProPublica has done on this topic. See the documentation from Propublica's Github on these data **here** and **here**

Visualization guidelines (5%)

To receive full points on visualizations throughout the problem set, your visualizations should be coded using `altair`, and should:

- Explicitly declare encodings and data types within `altair`
- Format the graph such that:
  - All axes and units are properly labeled and legible
  - No words or data points are cut off in your compiled PDF
  - Variables should be encoded in a sensible/intuitive way

Read in and characterize data (30%)

1. To help you get started, we pushed a file to the course repo called `parking_tickets_one_percent.csv` which gives you a one percent sample of tickets. We constructed the sample by selecting ticket numbers that end in `01`. How long does it take to read in this file? (Find a function to measure how long it takes the command to run. Note that everytime you run, there will be some difference in how long the code takes to run). Add an `assert` statement which verifies that there are 287458 rows.

Solution:

```
import pandas as pd
import time
import os
import unittest
# Only change directory if running interactively
try:
    os.chdir(os.path.dirname(os.path.abspath(__file__)))
except NameError:
    pass  # __file__ is not defined in notebooks or quarto

print("Current working directory:", os.getcwd())

def read_parking_tickets(file_path):
    start_time = time.time()
    df = pd.read_csv(file_path)
    end_time = time.time()
    elapsed_time = end_time - start_time
    assert len(df) == 287458, "The number of rows is not as expected."
    print(f"Time taken to read the file: {elapsed_time:.2f} seconds")
    return df

file_path = os.getcwd() + '/data/parking_tickets_one_percent.csv'
df = read_parking_tickets(file_path)
```

```
Current working directory:
/Users/richard/Documents/uchicago/dap/winter2026/ps/ps2
Time taken to read the file: 0.73 seconds

/var/folders/0k/rght1bkd1536yby9h4ql2ych0000gn/T/ipykernel_4469/4095908744.py:15:
DtypeWarning: Columns (7) have mixed types. Specify dtype option on import or
set low_memory=False.
  df = pd.read_csv(file_path)
```

2. Using a function in the `os` library calculate how many megabytes is the CSV file? Using math, how large would you predict the full data set is?

Solution:

```
    import os
    def get_file_size(file_path):
        file_size = os.path.getsize(file_path) / (1024 * 1024)  # Convert
↪   bytes to megabytes
        return file_size
```

```python
    def get_file_size_mb(file_path):
        file_size_bytes = os.path.getsize(file_path)

    # Example usage
    file_size = get_file_size(file_path)
    print(f"File size: {file_size:.2f} MB")

    # Predict the full dataset size
    one_percent_size = file_size
    full_dataset_size = one_percent_size * 100
    print(f"Predicted full dataset size: {full_dataset_size:.2f} MB")
```

```
File size: 79.78 MB
Predicted full dataset size: 7978.00 MB
```

3. The rows in the dataset are ordered or sorted by a certain column by default. Which column? Then, subset the dataset to the first 500 rows and write a function that tests if the column is ordered.

Solution:

```
   Unnamed: 0  ticket_number           issue_date violation_location  \
0           1     51482901.0  2007-01-01 01:25:00    5762 N AVONDALE
1           2     50681501.0  2007-01-01 01:51:00    2724 W FARRAGUT
2           3     51579701.0  2007-01-01 02:22:00       1748 W ESTES
3           4     51262201.0  2007-01-01 02:35:00    4756 N SHERIDAN
4           5     51898001.0  2007-01-01 03:50:00    7134 S CAMPBELL

                              license_plate_number license_plate_state  \
0  d41ee9a4cb0676e641399ad14aaa20d06f2c6896de6366...                   IL
1  3395fd3f71f18f9ea4f0a8e1f13bf0aa15052fc8e5605a...                   IL
2  302cb9c55f63ff828d7315c5589d97f1f8144904d66eb3...                   IL
3  94d018f52c7990cea326d1810a3278e2c6b1e8b44f3c52...                   IL
4  876dd3a95179f4f1d720613f6e32a5a7b86b0e6f988bf4...                   IL

  license_plate_type      zipcode violation_code  \
0                PAS  606184118.0       0964090E
1                PAS  606454911.0       0964090E
2                PAS  604116803.0       0964150B
3                PAS  606601345.0       0976160F
4                PAS  606291432.0       0964100A

                    violation_description  ...  fine_level2_amount  \
```

```
0                RESIDENTIAL PERMIT PARKING  ...                 100
1                RESIDENTIAL PERMIT PARKING  ...                 100
2       PARKING/STANDING PROHIBITED ANYTIME  ...                 100
3  EXPIRED PLATES OR TEMPORARY REGISTRATION  ...                 100
4                WITHIN 15' OF FIRE HYDRANT  ...                 200

  current_amount_due total_payments  ticket_queue  ticket_queue_date  \
0                0.0           50.0          Paid         2007-03-20
1                0.0           50.0          Paid         2007-01-31
2              122.0            0.0        Notice         2007-02-28
3                0.0           50.0          Paid         2007-01-11
4                0.0          100.0          Paid         2007-04-25

   notice_level  hearing_disposition notice_number officer  \
0          DETR               Liable  5.080059e+09   17266
1          VIOL                  NaN  5.079876e+09   10799
2          SEIZ                  NaN  5.037862e+09   17253
3           NaN                  NaN  5.075310e+09    3307
4          DETR                  NaN  5.073568e+09   16820

                       address
0  5700 n avondale, chicago, il
1  2700 w farragut, chicago, il
2     1700 w estes, chicago, il
3  4700 n sheridan, chicago, il
4  7100 s campbell, chicago, il

[5 rows x 24 columns]
The result of checking whether issue_date is increasing is: True
```

4. For each column, how many rows are `NA`? Write a function which returns a two column data frame where each row is a variable, the first column of the data frame is the name of each variable, and the second column of the data frame is the number of times that the column is NA. Test your function. Then, report the results applied to the parking tickets data frame. There are several ways to do this, but we haven't covered them yet in class, so you will need to work independently to set this up.

```
# adding typing to the function signature for clarity
# and to demonstrate how to use type hints
from pandas import DataFrame, Series
import numpy as np
def get_number_of_NAs(df: DataFrame) -> Series:
    """
```

```
    Takes in a pandas dataframe and returns a pandas series with the
    number of missing values in each column."""

    na_df = df.isna().sum()

    return na_df

# multiple ways to do this does not need to be a full unittest class

class TestGetNumberOfNAs(unittest.TestCase):
    def test_mixed_nan_values(self):
        data = {
            'A': [1, 2, np.nan, 4, 5],
            'B': [np.nan, 2, 3, np.nan, 5],
            'C': [1, 2, 3, 4, 5],
            'D': [np.nan, np.nan, np.nan, np.nan, np.nan]
        }
        df = pd.DataFrame(data)
        result = get_number_of_NAs(df)
        expected = pd.Series({'A': 1, 'B': 2, 'C': 0, 'D': 5})
        pd.testing.assert_series_equal(result, expected)

    def test_empty_dataframe(self):
        empty_df = pd.DataFrame()
        result = get_number_of_NAs(empty_df)
        self.assertTrue(result.empty)

    def test_no_nan_values(self):
        no_nan_df = pd.DataFrame({'X': [1, 2, 3], 'Y': [4, 5, 6]})
        result = get_number_of_NAs(no_nan_df)
        expected = pd.Series({'X': 0, 'Y': 0})
        pd.testing.assert_series_equal(result, expected)

# Run the tests
test_suite = unittest.TestLoader().loadTestsFromTestCase(TestGetNumberOfNAs)
test_runner = unittest.TextTestRunner(verbosity=2)
test_result = test_runner.run(test_suite)

# Print summary
print(f"Failures: {len(test_result.failures)}")
print(f"Errors: {len(test_result.errors)}")
```

```
test_empty_dataframe (__main__.TestGetNumberOfNAs.test_empty_dataframe) ...
ok
test_mixed_nan_values (__main__.TestGetNumberOfNAs.test_mixed_nan_values) ...
ok
test_no_nan_values (__main__.TestGetNumberOfNAs.test_no_nan_values) ... ok

----------------------------------------------------------------------
Ran 3 tests in 0.002s

OK

Failures: 0
Errors: 0
```

```
# Run the function on the parking tickets data
na_df = get_number_of_NAs(df)
print("Number of missing values in each column:")
print(na_df)
```

```
Number of missing values in each column:
Unnamed: 0                   0
ticket_number                0
issue_date                   0
violation_location           0
license_plate_number         0
license_plate_state         97
license_plate_type        2054
zipcode                  54115
violation_code               0
violation_description        0
unit                        29
unit_description             0
vehicle_make                 0
fine_level1_amount           0
fine_level2_amount           0
current_amount_due           0
total_payments               0
ticket_queue                 0
ticket_queue_date            0
notice_level             84068
hearing_disposition     259899
notice_number                0
officer                      0
```

```
address                       0
dtype: int64
```

5. Three variables are missing much more frequently than the others. Why? (Hint: look at some rows and read the data dictionary written by ProPublica)

Solution: The three columns with the most missing values are `zipcode`, `hearing_disposition`, and `notice_level`. ZIP is matched to a car registration and so is missing if there is no car regestration. Notice level is messing if no notice was sent and hearing disposition is missing if there was no hearing

6. How many tickets were issued in the data in 2017? How many tickets does that imply were issued in the full data in 2017? How many tickets are issued each year according to the ProPublica article? Do you think that there is a meaningful difference? **Solution:**

```
df['issue_date'] = pd.to_datetime(df['issue_date'])
df_2017 = df[df['issue_date'].dt.year == 2017]
print(df_2017.shape)
```

```
(22364, 24)
```

There are 22k tickets in 2017 which implies that 2.2 million tickets were issued in 2017. The article mentions 3 million tickets which is a significant difference.

Visual encodings (10%)

1. In Lecture 2, we discussed how `altair` thinks about categorizing data series into four different types. Which data type or types would you associate with each column in the data frame? Your response should take the form of a Markdown table where each row corresponds to one of the variables in the parking tickets dataset, the first column is the variable name and the second column is the variable type or types. If you argue that a column might be associated with than one type, explain why in writing below the table.

Solution:

Variable Name	Altair Data Type
`ticket_number`	Nominal Ordinal
`issue_date`	Temporal
`violation_location`	Nominal
`license_plate_number`	Nominal
`license_plate_state`	Nominal
`license_plate_type`	Nominal
`zipcode`	Nominal

Variable Name	Altair Data Type
violation_code	Nominal
violation_description	Nominal
unit	Quantitative
unit_description	Nominal
vehicle_make	Nominal
fine_level1_amount	Quantitative
fine_level2_amount	Quantitative
current_amount_due	Quantitative
total_payments	Quantitative
ticket_queue	Nominal
ticket_queue_date	Temporal
notice_level	Ordinal
hearing_disposition	Nominal
notice_number	Nominal or Ordinal
officer	Nominal
address	Nominal
date	Temporal

While written as a number, `ticket_number` seems to be used for identification and so it will most likely be useful as a nominal data type. However, if the `ticket_number` can be used to tell us which order tickets were written it could be ordinal. The same argument holds for `notice_number`.

`zipcode`s are most often used as nominal data types, but they can also be used as quantitative data types if you are using the geographic information that is encoded in the zipcode numbering.

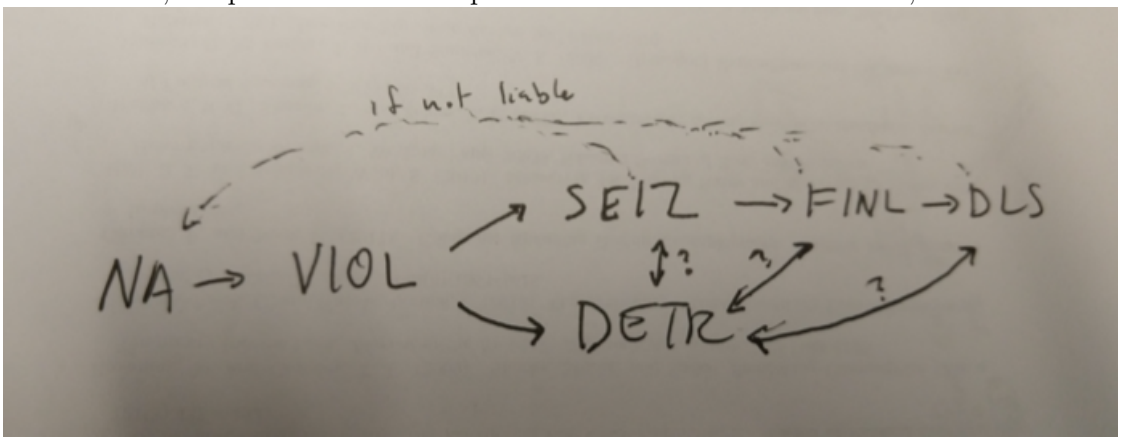
Understanding the structure of the data and summarizing it (20%)

1. Many datasets implicitly contain information about how the data are generated. In this setting, we can use the data to learn how a case progresses.

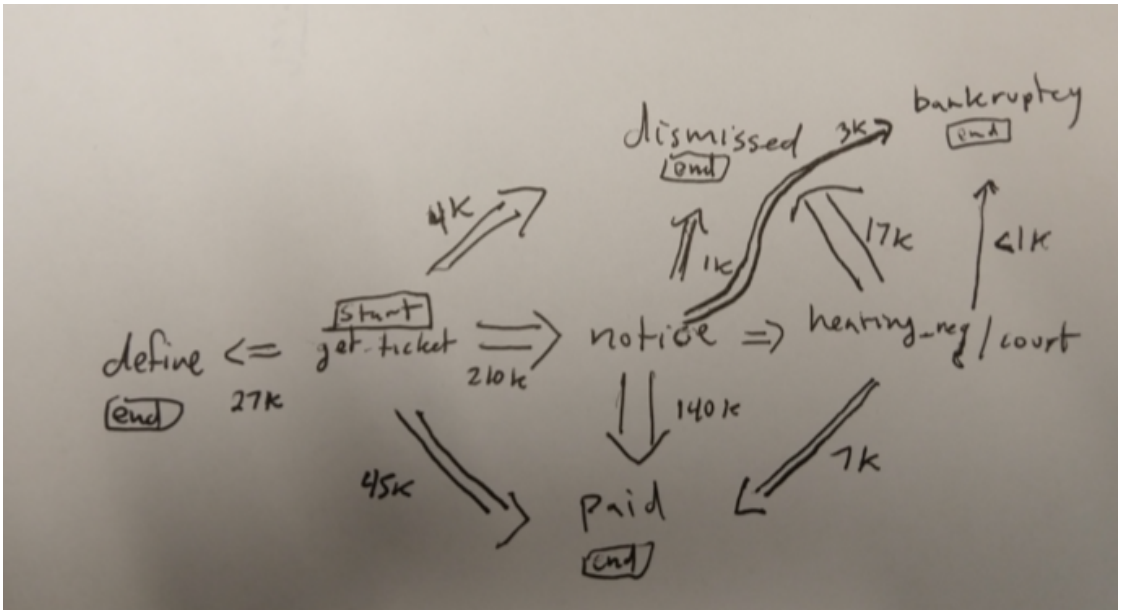
- Draw a diagram explaining your understanding of the process of moving between the different values of `notice_level`, and how you used the data or other sources to arrive at this conclusion. If you draw it on paper, take a picture and include the image in your write up.
- Draw a second diagram explaining the different values of `ticket_queue`. If someone contests their ticket and is found not liable, what happens to `notice_level` and to `ticket_queue`? Include this in your diagram above.

Solution:

- The progression through `notice_level` is obvious for the first two steps and then it becomes less clear how cases progress. We have tools at our disposal however. First we can look at counts. For example, if we make an assumption that at each step of the process some people pay, we can roughly order the levels based on how frequent they are. Further we can make a crosstab with `ticket_queue`; This gives a sense of how long people remain in a level. Particularly, looking at `'Notice'` is helpful, since it represents a phase that is not final (as opposed to `'Paid'` or `'Dismissed'`). Nearly half of `'SEIZ'` are on `'Notice'`, compared to less than 5 percent of `DETR`. With these in mind, I made this tree:



- The `ticket_queue` provides a more satisfactory story. I guessed at the size of the flows from one value to another using the context from `hearing_disposition` and `notice_level`. There are people who skip some steps (e.g. 8 people in the subsample go directly to bankruptcy), but these are exceptions to the normal progressions through the system. As a correction, I should have an arrow from `get_ticket` to `hearing`. It appears about 6000 go directly to a hearing.



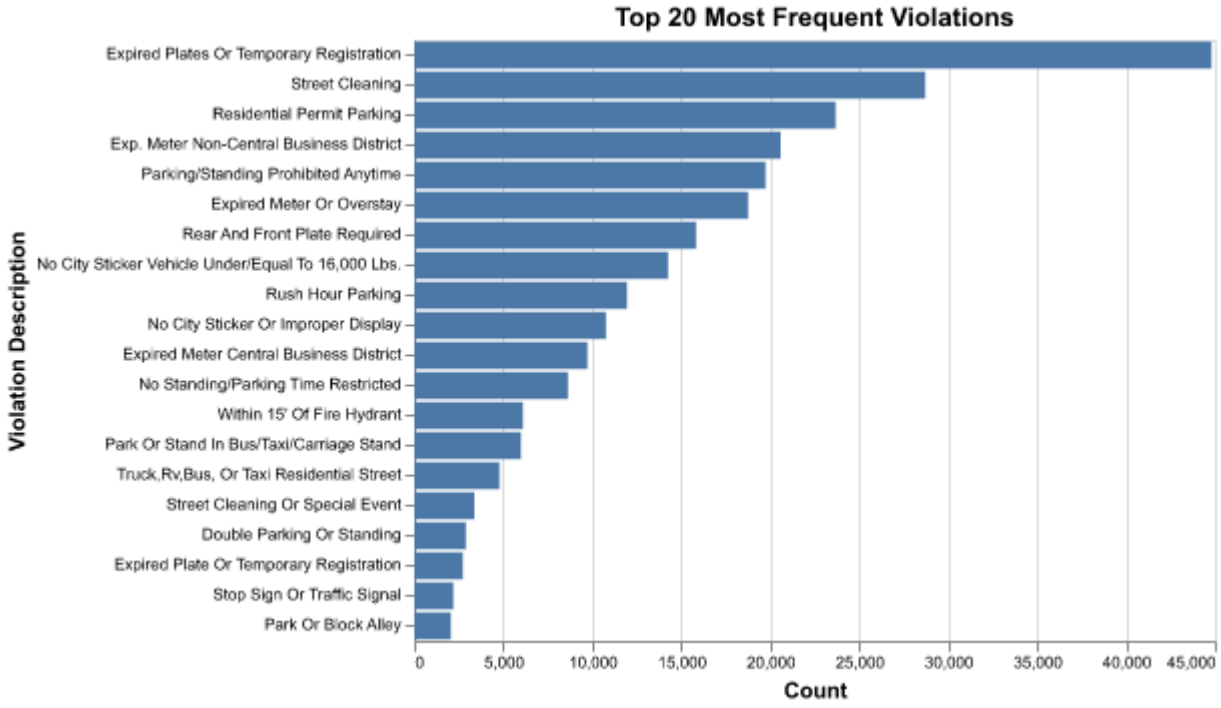
Aggregating, transforming, visualizing the data (30%)

1. Pooling the data across all years, what are the top 20 most frequent violation types? Make a bar graph to show the frequency of these ticket types.

Solution:

```
# convert violation description to title case instead of all caps
df['violation_description'] = df['violation_description'].str.title()

alt.Chart(df).transform_aggregate(
    frequency='count()',
    groupby=['violation_description']
).transform_window(
    rank='rank(frequency)',
    sort=[alt.SortField('frequency', order='descending')]
).transform_filter(
    (alt.datum.rank <= 20)
).mark_bar().encode(
    alt.X('frequency:Q', title='Count'),
    alt.Y('violation_description:N', sort='-x', title="Violation
 ↪  Description")
).properties(
    title='Top 20 Most Frequent Violations',
```

```
    width=400,
    height=300
).configure_axis(
    labelFontSize=8,
    labelLimit=400
)
```



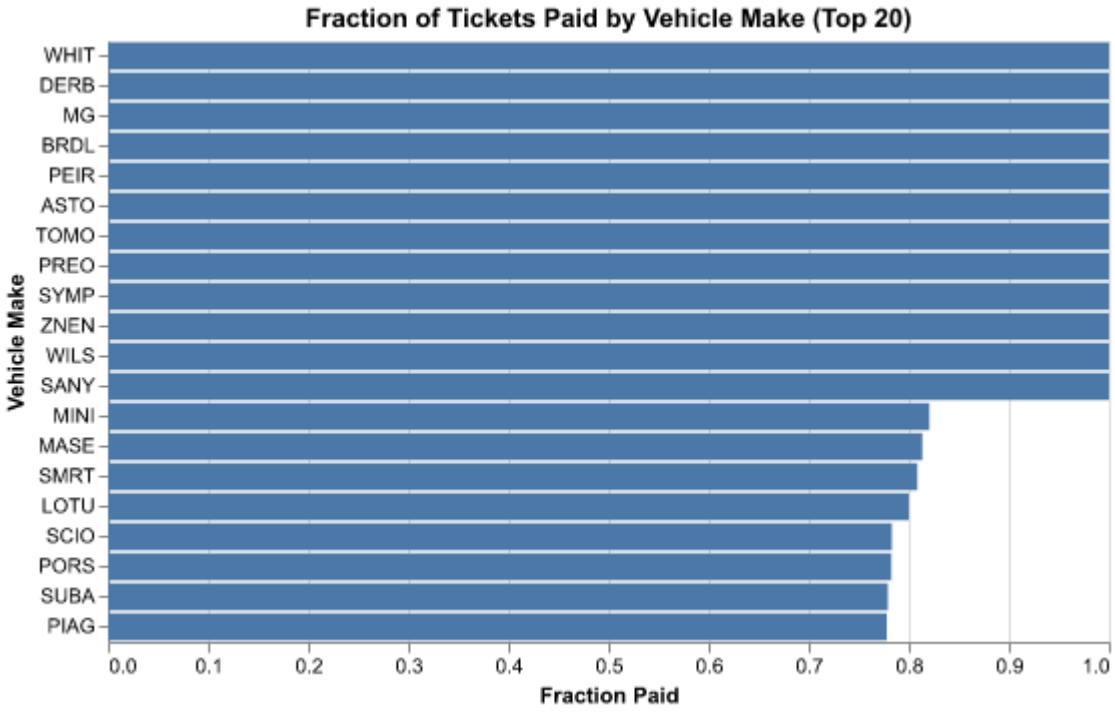
The code does the following:

- `transform_aggregate` calculates the counts by violation type
- `transform_window` calculates rank by count
- `transform_filter` limits to highest-ranking values
- `sort='-x'` within `alt.Y` ensures that the bars are sorted by frequency, rather than alphabetically (which is the default because `violation_description` is nominal)

2. Compute the fraction of time that tickets issued to each vehicle make are marked as paid. Show the results as a bar graph. Why do you think that some vehicle makes are more or less likely to have paid tickets?

Solution:

```
# create a new variable which == 1 *only* when the ticket is paid.
df['paid_ticket'] = df['ticket_queue'] == 'Paid'

alt.Chart(df).transform_aggregate(
    paid_count = 'sum(paid_ticket)',
    total_count = 'count()',
    groupby=['vehicle_make']
).transform_calculate(
    fraction_paid = 'datum.paid_count/datum.total_count'
).transform_window(
    rank='rank(datum.fraction_paid)',
    sort=[alt.SortField('fraction_paid', order='descending')]
).transform_filter(
    alt.datum.rank <= 20
).mark_bar().encode(
    alt.X('fraction_paid:Q', title='Fraction Paid'),
    alt.Y('vehicle_make:N', sort='-x', title='Vehicle Make')
).properties(
    title='Fraction of Tickets Paid by Vehicle Make (Top 20)',
    width=500,
    height=300
).configure_axis(
    labelFontSize=10
)
```



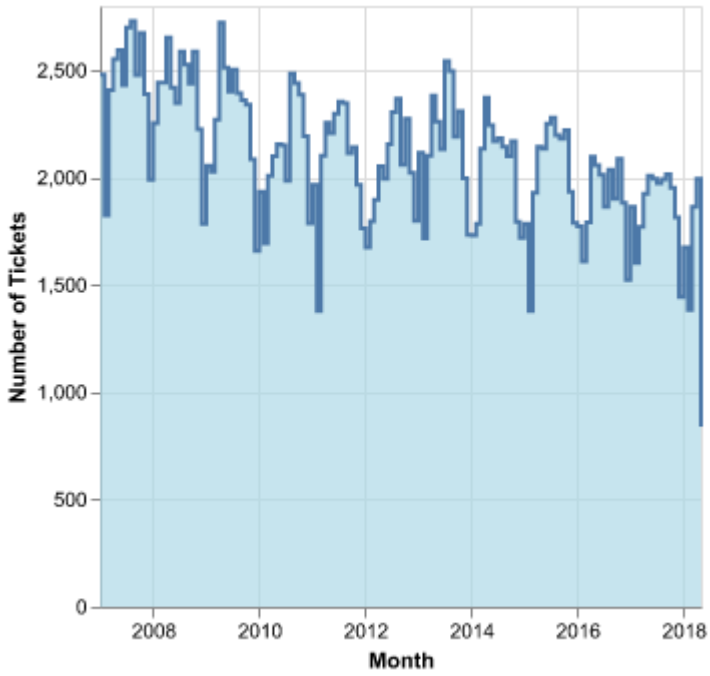
Vehicle makes with the highest fraction paid also look like they are more expensive brands. Car brand is likely a proxy for income and wealth which would affect the ability to pay tickets.

3. Make a plot for the number of tickets issued over time by adapting the Filled Step Chart example online. List two observations or takeaways that jump out from this plot. Go back to the taxonomy of visual encodings we discussed in lecture. What visual encoding channel (or channels) does this use?

Solution:

```
# make sure issue_date is a datetime
df['issue_date'] = pd.to_datetime(df['issue_date'])
#df['month'] = df['issue_date'].dt.to_period('M').dt.strftime('%Y-%m')
# # aggregate to the month level
# df['date'] = df['issue_date'].dt.strftime('%Y-%m')
# tickets_by_date = df.groupby('date').size().reset_index()
# tickets_by_date.columns = ['Date', 'Number of Tickets per Month']

alt.Chart(df).transform_timeunit(
    ym = 'yearmonth(issue_date)' #yearmonth() preserves the year + month,
 ↪  whereas month() only extracts the month
).transform_aggregate(
```

```
    ticket_count='count()',
    groupby=['ym']
).mark_area(
    color="lightblue",
    interpolate='step-after',
    line=True
).encode(
    alt.X('ym:T', title = 'Month'),
    alt.Y('ticket_count:Q', title = 'Number of Tickets')
)
```



- The number of tickets issued is decreasing over time and looks like it is seasonal.
- The visual encoding channel for ticket count is position on a common scale (or area).

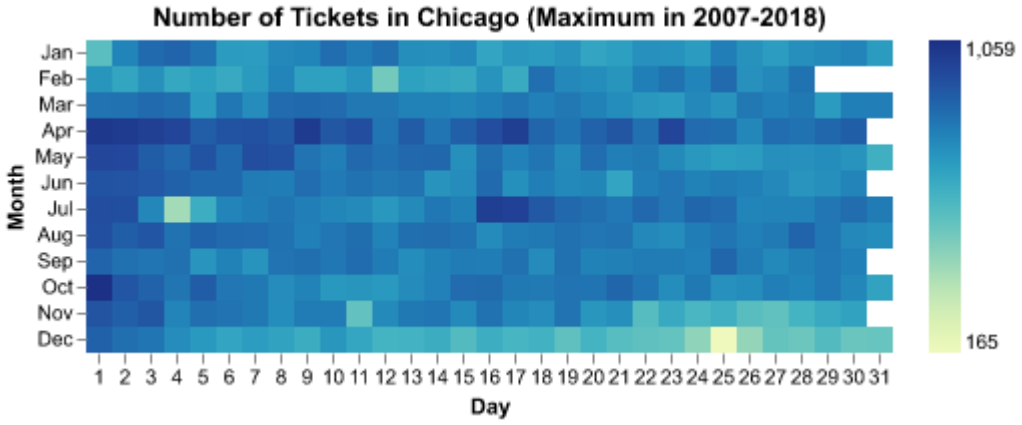
4. Make a plot for the number of tickets issued by month and day by adapting the Annual Weather Heatmap example online. List two observations or takeaways that jump out from this plot. What visual encoding channel (or channels) does this use?

Solution:

```
# want to look at by calendar day for all years
df['date'] = df['issue_date'].dt.strftime('%m-%d')
```

```
# count the number of tickets issued each day
daily_tickets = df.groupby('date').size().reset_index()
daily_tickets.columns = ['date', 'num_tickets']

# we're reporting the maximum value, so we want to know what time frame this
 ↪  is calculated from
min_year = df['issue_date'].dt.year.min()
max_year = df['issue_date'].dt.year.max()

print(f"Earliest year: {min_year}, Latest year: {max_year}")

alt.Chart(daily_tickets, title="Number of Tickets in Chicago (Maximum in
 ↪  2007-2018)").mark_rect().encode(
    alt.X("date(date):O").title("Day").axis(format="%e", labelAngle=0),
    alt.Y("month(date):O").title("Month"),
    alt.Color("max(num_tickets)").title(None),
).configure_view(
    step=13,
    strokeWidth=0
).configure_axis(
    domain=False
)
```

```
Earliest year: 2007, Latest year: 2018
```



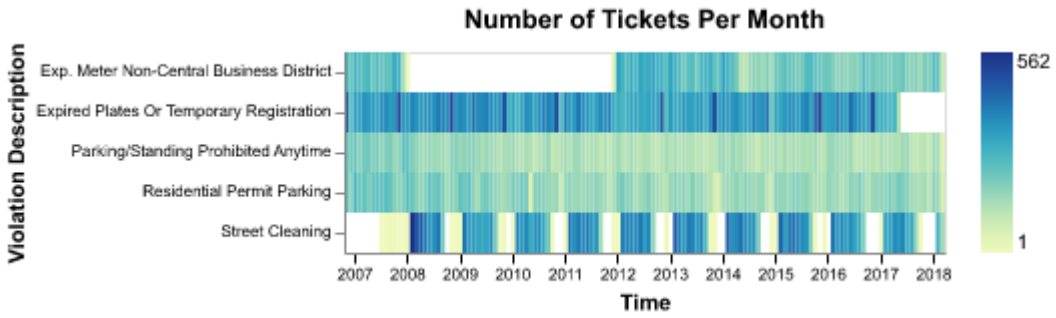
- Christmas and 4th of July have very few tickets issued.
- There are more tickets at the start of the month.
- The visual encoding channel for number of tickets is color saturation.
- (Note that the plot title makes it clear we're plotting the *maximum* value across 2007-2018).

5. Subset to the five most common types of violations. Make a plot for the number of tickets issued over time by adapting the Lasagna Plot example online. Explore what the use of `color_condition = alt.condition(...)` does in the plot, and describe below. List two observations or takeaways that jump out from this plot. What visual encoding channel (or channels) does this use?

Solution:

```
# calculate counts of each unique violation, then identify the indices of the
 ↪  5 most common ones
five_most_common = df['violation_description'].value_counts().head(5).index

# filter dataset to just those
five_most_common_df =
 ↪  df[df['violation_description'].isin(five_most_common)].copy()

# create a column that is the year-month
five_most_common_df['date'] =
 ↪  five_most_common_df['issue_date'].dt.strftime('%Y-%m')

# make violation description title case
five_most_common_df['violation_description'] =
 ↪  five_most_common_df['violation_description'].str.title()

# aggregate to the monthly-level
five_most_common_df = five_most_common_df.groupby(['date',
 ↪  'violation_description']).size().reset_index()
five_most_common_df.columns = ['date', 'violation_description',
 ↪  'num_tickets']

# we will use this to ensure there is a black tick at January 1
color_condition = alt.condition(
    "month(datum.value) == 1 && date(datum.value) == 1",
    alt.value("black"),
    alt.value(None),
)

alt.Chart(five_most_common_df, width=300, height=100, title = "Number of
 ↪  Tickets Per Month").mark_rect().encode(
    alt.X("yearmonth(date):O")
        .title("Time")
        .axis(
            format="%Y",
```

```
            labelAngle=0,
            labelOverlap=False,
            labelColor=color_condition,
            tickColor=color_condition,
        ),
    alt.Y("violation_description:N").title("Violation Description"),
    alt.Color("sum(num_tickets)").title(None)
).configure_axis( # Configure the axis so that the labels are not cut off
    labelFontSize=8,
    labelLimit=400
)
```



- `color_condition = alt.condition(...)` ensures that the x-axis label is only on January 1 for each year, and that it is black. Removing it leads to an overcrowded x-axis label.
- Some violations, like "expired meter non-central business district" are not used for several years and then come back into use.
- Street cleaning tickets appear to be seasonal and mostly happen outside of the winter time.
- The visual encoding channel for number of tickets is color saturation.

6. Compare and contrast the plots you made for the prior three questions. What are the pros and cons of each plot? What kinds of questions is each plot best-suited for answering?

Solution:

- The line plot is easy to read and interpret and successfully shows the overall trend over time. However, it doesn't give details for what is driving the trend and what the composition of tickets is like.
- The heat map does a good job at showing the seasonality of tickets and prompts us to ask interesting questions like, "why are there more tickets at the start of each month" and "why does April seem to be a high ticket month?". The downside it that it is very

busy and takes some time to interpret. It also reports only the maximum value across all years which might mask interesting trends across years.

- The lasagna plot shows us some interesting facts about the most common ticket types which motivate potentially interesting questions. However, it is the hardest of the three to quickly interpret and only shows the top 5 ticket types which may not be the most policy relevant.

7. Suppose that the lesson you want a reader to take away is that the enforcement of violations is not evenly distributed over time. Which plot is best and why?

Solution:

The heat plot does the best job of showing that tickets are not given evenly across the year. This tells us that either the violation rate or the enforcement rate (or both!) varies over time. This plot can be used to motivate further questions or be used in conjuction with other analysis to make the case that enforcement changes over the year and why that might be.