ps1

AUTHOR
Clarice Tee

PUBLISHED
October 3, 2024

Academic integrity statement

I checked Googled commands/ways to go about getting the output needed. I have included the links where I learned from. Additionally, I used ChatGPT for help with debugging double checking the flow of my code. I also referred to code from last quarter's python class for grouping, datetime, dictionary, length, float, round, print,a nd reset index functinos.

1. "This submission is my work alone and complies with the 30538 integrity policy." **CT**
2. "I have uploaded the names of anyone I worked with on the problem set **here**" **__** (1 point)
3. Late coins used this pset: **1** Late coins left after submission: **3**
4. Knit your `ps1.qmd` to make `ps1.pdf`.
    - The PDF should not be more than 25 pages. Use `head()` and re-size figures when appropriate.
5. Push `ps1.qmd` and `ps1.pdf` to your github repo. It is fine to use Github Desktop.
6. Submit `ps1.pdf` via Gradescope (4 points)
7. Tag your submission in Gradescope

Read in one percent sample (15 Points)

1.

```
import pandas as pd
import os

# Reading in file
file_path = r'C:\Users\clari\OneDrive\Documents\Python II\ppha30538_fall2024\problem_sets\ps1\data
df = pd.read_csv(file_path)
df
```

```
C:\Users\clari\AppData\Local\Temp\ipykernel_9888\712313320.py:6: DtypeWarning:

Columns (7) have mixed types. Specify dtype option on import or set low_memory=False.
```

	Unnamed: 0	ticket_number	issue_date	violation_location	license_plate_number
0	1	5.148290e+07	2007-01-01 01:25:00	5762 N AVONDALE	d41ee9a4cb0676e641399ad14aaa20d06f2c6896de63
1	2	5.068150e+07	2007-01-01	2724 W FARRAGUT	3395fd3f71f18f9ea4f0a8e1f13bf0aa15052fc8e5605a..

	Unnamed: 0	ticket_number	issue_date	violation_location	license_plate_number
			01:51:00		
2	3	5.157970e+07	2007-01-01 02:22:00	1748 W ESTES	302cb9c55f63ff828d7315c5589d97f1f8144904d66eb
3	4	5.126220e+07	2007-01-01 02:35:00	4756 N SHERIDAN	94d018f52c7990cea326d1810a3278e2c6b1e8b44f3c
4	5	5.189800e+07	2007-01-01 03:50:00	7134 S CAMPBELL	876dd3a95179f4f1d720613f6e32a5a7b86b0e6f988bf
...	...	...	...	...	...
287453	287454	9.190000e+09	2018-05-14 14:51:00	1128 W MONROE	31f8a07f5ffc423447ceb2e99a5dc5dfd7b626cfb5d4e4
287454	287455	9.190000e+09	2018-05-14 16:34:00	1820 N MILWAUKEE AVE	7533cbfcb6cd16743d6f3c77af51b288641823bba9eef
287455	287456	9.190000e+09	2018-05-14 16:52:00	122 E 21ST ST	3af6ff915e8335ea396cb09761a1f3675219b3f0d24ecc
287456	287457	9.190000e+09	2018-05-14 18:04:00	10 S DEARBORN ST	fa986644c31a2c97f4c260334175e2cc5b22ee88a2808
287457	287458	9.190000e+09	2018-05-14 20:56:00	2201 W ARTHUR	5fdb8dd85edbb1d262388d0b33e849d06affc0f33510

287458 rows × 24 columns

Checking the length of the file

```
assert len(df) == 287458
print(f"We found {len(df)} rows in the dataset")
```

```
We found 287458 rows in the dataset
```

https://stackoverflow.com/questions/5142418/what-is-the-use-of-assert-in-python

```
import time
```

Checking how long it takes to read in the file

```
start_time = time.time()

df = pd.read_csv(file_path)
df
end_time = time.time()
run_time = float(round(end_time-start_time, 2))

print(f"It took : {run_time} seconds to read in the file")
```

```
It took : 1.28 seconds to read in the file

C:\Users\clari\AppData\Local\Temp\ipykernel_9888\4022165332.py:3: DtypeWarning:

Columns (7) have mixed types. Specify dtype option on import or set low_memory=False.
```

https://stackoverflow.com/questions/6786990/find-out-time-it-took-for-a-python-script-to-complete-execution

Getting number of MBs

2. How many magebytes?

```
about_file = os.stat(file_path)
file_size = about_file.st_size
print(file_size)
file_mb = float(round(file_size/ (1024 * 1024),2))
print(f"The file is {file_mb} megabytes")
```

```
83942807
The file is 80.05 megabytes
```

We multiply it by 100 because this is called one percent. Therefore, the full file would be 80,000 megabytes.

https://stackoverflow.com/questions/5194057/better-way-to-convert-file-sizes-in-python # Order by column 3. Ordered by which column # Ordering It is ordered by issue date

```
df_subset = df.head(500)
def is_col_sorted(df_subset, issue_date):
    return df_subset[issue_date].is_monotonic_increasing

answer = is_col_sorted(df_subset, "issue_date")
print(f"{answer}, the columns are sorted by 'Issue Date'")
```

```
True, the columns are sorted by 'Issue Date'
```

Cleaning the data and benchmarking (15 Points)

1. How many tickets were issued in the data in 2017?

```
# Changing to  datetime
df['issue_date'] = pd.to_datetime(df['issue_date'])

# Filter to 2017 tickets and count
tickets_2017 = df[df['issue_date'].dt.year == 2017]['ticket_number'].count()

print(f"There were {tickets_2017} tiickets issued in 2017")
```

```
There were 22364 tiickets issued in 2017
```

1a. How many tickets does that imply were issued in the full data in 2017?

```
full_tickets_2017 = tickets_2017 * 100
commas_full_tickets_2017 = f"{full_tickets_2017:,}"
print(f"This implies that there were {commas_full_tickets_2017} issued in the full data for
```

```
This implies that there were 2,236,400 issued in the full data for the year 2017
```

The article said Chicago issues more than 3 million tickets per year, but we found that there were 22,364 tickets issued in 2017 in the one percent dataset, meaning there were around 2.24 million tickets issued in 2017 (x100 for full dataset). ITHis is a meaningful difference between the two sources, with the the ProPublica number being 34% higher than the actual number. Even if we round the number, it would round down to 2 Million.

2a. Pooling the data across all years what are the top 20 most frequent violation types?

```
violations_by_type = df['violation_description'].value_counts()

# Top 20 most frequent violations
top_20_violations = violations_by_type.head(20)
top_20_df = top_20_violations.reset_index()


print(f"Top 20 Most Frequent Violation Types Across All Years are the following, along with the fr
```

```
Top 20 Most Frequent Violation Types Across All Years are the following, along with the frequency
of the violations:
violation_description
EXPIRED PLATES OR TEMPORARY REGISTRATION              44811
STREET CLEANING                                      28712
RESIDENTIAL PERMIT PARKING                           23683
```

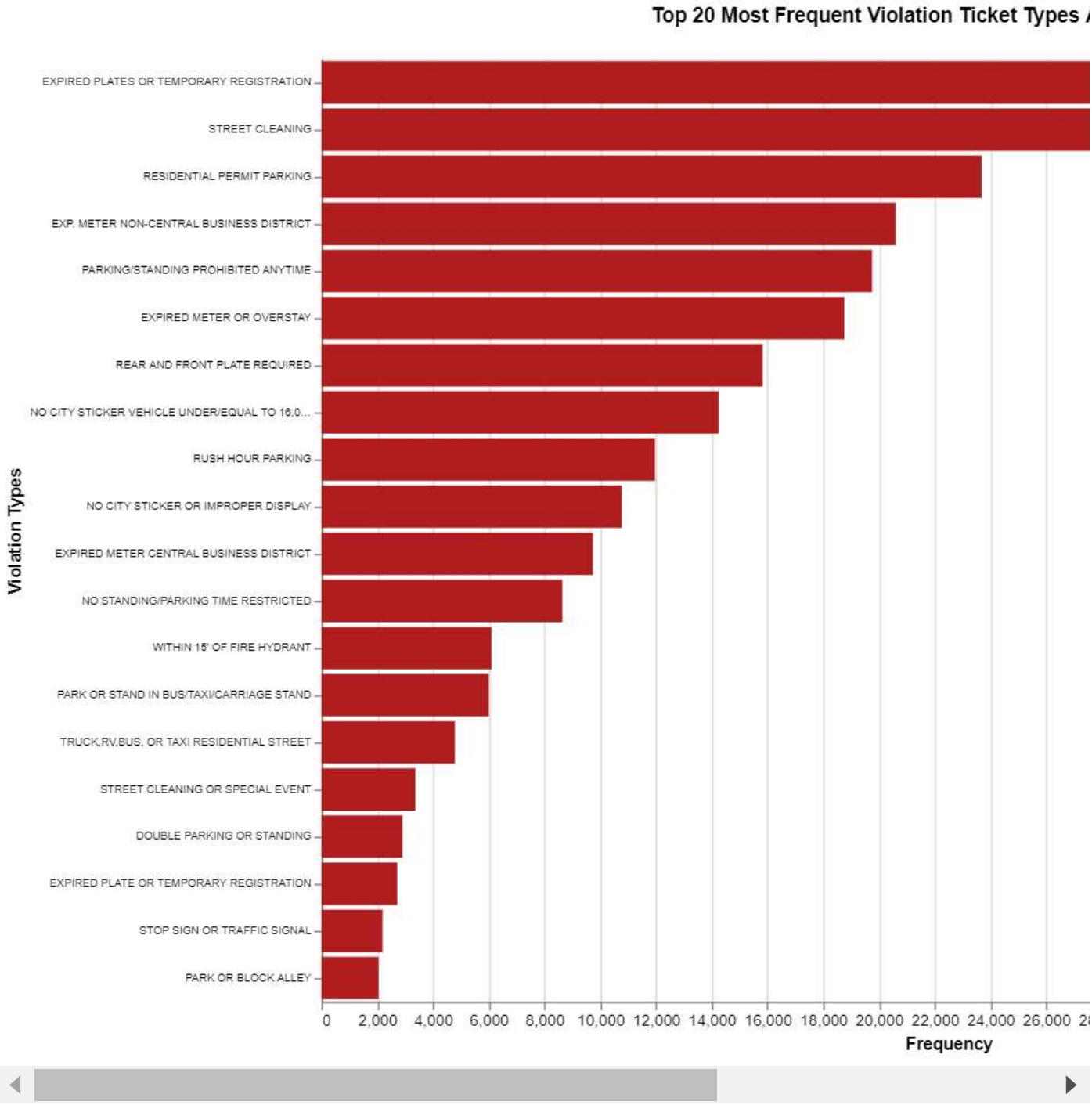
```
EXP. METER NON-CENTRAL BUSINESS DISTRICT              20600
PARKING/STANDING PROHIBITED ANYTIME                   19753
EXPIRED METER OR OVERSTAY                             18756
REAR AND FRONT PLATE REQUIRED                         15829
NO CITY STICKER VEHICLE UNDER/EQUAL TO 16,000 LBS.    14246
RUSH HOUR PARKING                                     11965
NO CITY STICKER OR IMPROPER DISPLAY                   10773
EXPIRED METER CENTRAL BUSINESS DISTRICT                9736
NO STANDING/PARKING TIME RESTRICTED                    8640
WITHIN 15' OF FIRE HYDRANT                             6104
PARK OR STAND IN BUS/TAXI/CARRIAGE STAND               6004
TRUCK,RV,BUS, OR TAXI RESIDENTIAL STREET               4789
STREET CLEANING OR SPECIAL EVENT                       3370
DOUBLE PARKING OR STANDING                             2904
EXPIRED PLATE OR TEMPORARY REGISTRATION                2720
STOP SIGN OR TRAFFIC SIGNAL                            2191
PARK OR BLOCK ALLEY                                    2050
Name: count, dtype: int64
```

2b. Make a bar graph to show the frequency of these ticket types. Format the graph such that the violation descriptions are legible and no words are cut off.

```
import altair as alt
```

```
top_20_df["count_thousands"] = top_20_df["count"] / 1000

bar_graph = alt.Chart(top_20_df).mark_bar(color='firebrick').encode(
    x=alt.X("count:Q", title="Frequency"),
    y=alt.Y("violation_description:N",
            title="Violation Types",
            sort="-x",
            axis=alt.Axis(labelFontSize=7))
).properties(
    title={
        "text": "Top 20 Most Frequent Violation Ticket Types Across All Years",
        "offset": 20
    },
    width=800,
    height=600
)

bar_graph.display()
```

```
C:\Users\clari\anaconda3\Lib\site-packages\altair\utils\core.py:395: FutureWarning:

the convert_dtype parameter is deprecated and will be removed in a future version.  Do
``ser.astype(object).apply()`` instead if you want ``convert_dtype=False``.
```



https://stackoverflow.com/questions/71215156/how-to-wrap-axis-label-in-altair I also used the lecture material for graphing here.

Visual Encoding (15 Points)

1.

```
# Pandas Data Types
data_types = df.dtypes
print(data_types)

# Mapping dictionary
```

```
type_mapping = {
    'int64': 'Q',
    'float64': 'Q',
    'object': 'N',
    'bool': 'N',
    'datetime64[ns]': 'T',
    'category': 'N'
}

# Dictionary mapping to altair
altair_data_types = {col: type_mapping.get(str(dtype), 'N') for col, dtype in data_types.items()}

print("Altair types:")
for col, alt_type in altair_data_types.items():
    print(f"{col}: {alt_type}")
```

```
Unnamed: 0                      int64
ticket_number                 float64
issue_date               datetime64[ns]
violation_location             object
license_plate_number           object
license_plate_state            object
license_plate_type             object
zipcode                        object
violation_code                 object
violation_description          object
unit                          float64
unit_description               object
vehicle_make                   object
fine_level1_amount              int64
fine_level2_amount              int64
current_amount_due            float64
total_payments                float64
ticket_queue                   object
ticket_queue_date              object
notice_level                   object
hearing_disposition            object
notice_number                 float64
officer                        object
address                        object
dtype: object
Altair types:
Unnamed: 0: Q
ticket_number: Q
issue_date: T
violation_location: N
license_plate_number: N
license_plate_state: N
license_plate_type: N
zipcode: N
```

```
violation_code: N
violation_description: N
unit: Q
unit_description: N
vehicle_make: N
fine_level1_amount: Q
fine_level2_amount: Q
current_amount_due: Q
total_payments: Q
ticket_queue: N
ticket_queue_date: N
notice_level: N
hearing_disposition: N
notice_number: Q
officer: N
address: N
```

https://altair-viz.github.io/user_guide/data.html

Table of Variable Data Types in Altair

** Variable	Data Type/s
Unnamed: 0	Ordinal
ticket_number	Quantitative/Ordinal/Nominal
violation_location	Nominal
issue_date	Temporal/Ordinal
license_plate_number	Nominal
license_plate_state	Nominal
license_plate_type	Nominal
zipcode	Nominal/ Ordinal
violation_code	Nominal
violation_descriptoin	Nominal
unit	Nominal
unit_description	Nominal
vehicle_make	Nominal
fine_level1_amount	Quantitative

** Variable	Data Type/s
fine_level2_amount	Wuantitative
current_amount_due	Quantitative
total_payments	Quantitative
ticket_que	Nominal
ticket_que_date	Temporal/ Ordinal
notice_level	Nominal
hearing_disposition	Nominal
notice_number	Quantitative/Ordinal/Nominal
officer	Nominal
address	Nominal

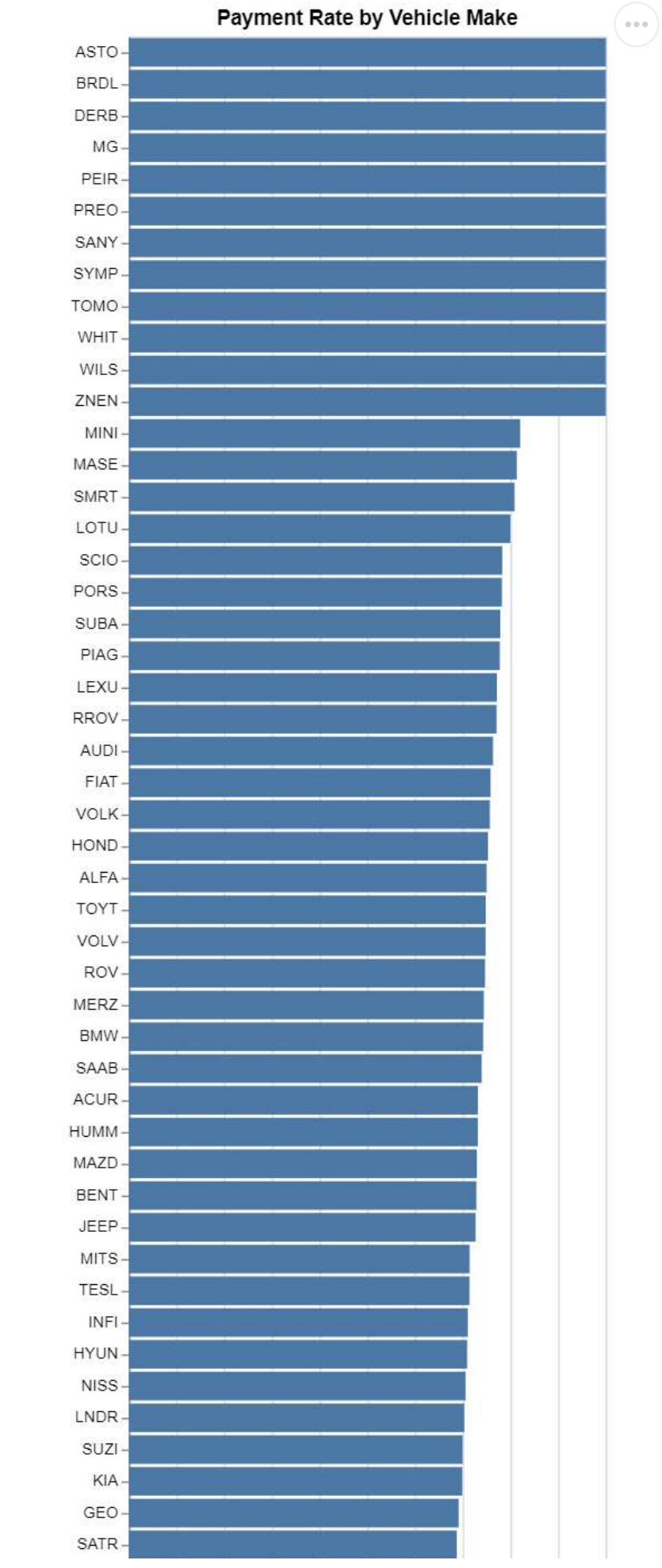
Explanations

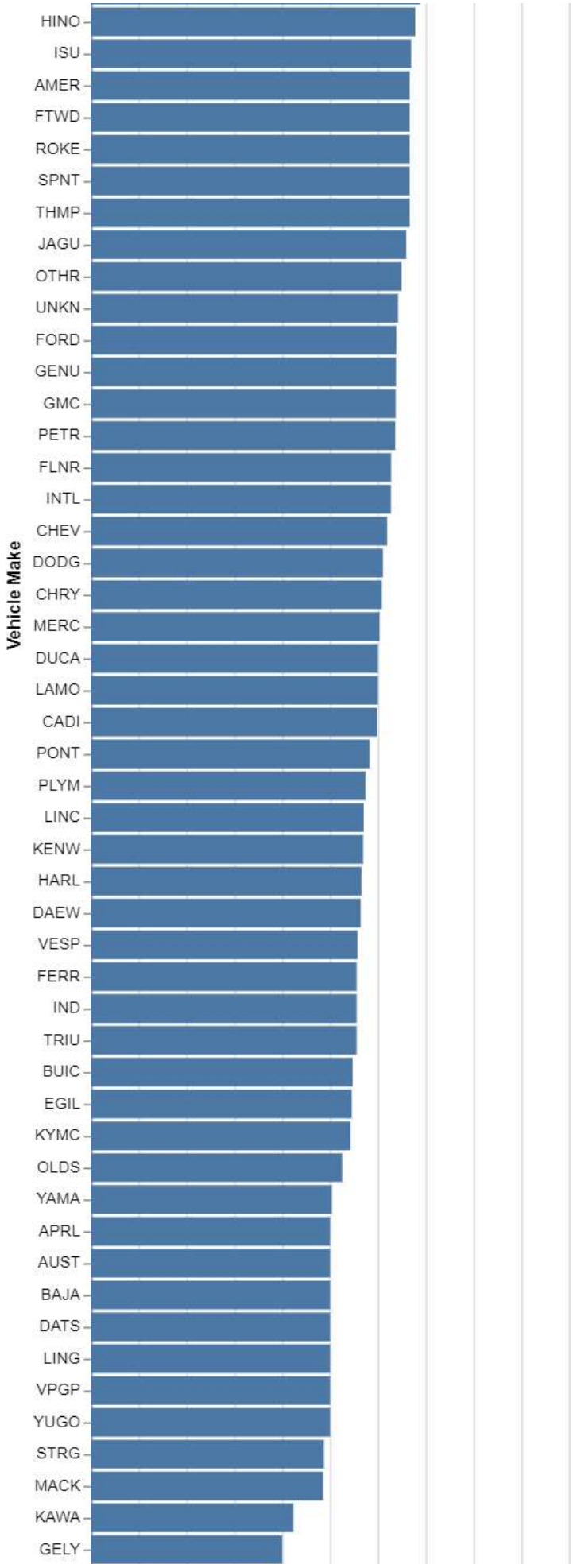
- ticket_number may be Quantitative if used as an interval, Ordinal when we are looking at the sequence of tickets (older vs.. newer), and nominal if used purely as a way to identify the traffic violation instance without any numerical meaning.
- issue_date could be Temporal when we want the specific date/ime, Ordinal when we want to know the order in which tickets were given (i.e. which was first or last).
- zipcode is Nominal when categorizing locations without any order, so each zipcode is an ID fo a specific area, iordinal if it's used to identify the locations/neighborhoods in order (i.e. more south or nort)
- ticket_que_date is Temporal when looking at the date and time, Ordinal if we want to know the order of the tickets.
- notice_number is Quantitative could help if we want to do calculations, like the frequency of notices, Ordinal for theorder of notices (i.e.., first notice vs. final notice), and Nominal if we're just using it as an ID of the notices.

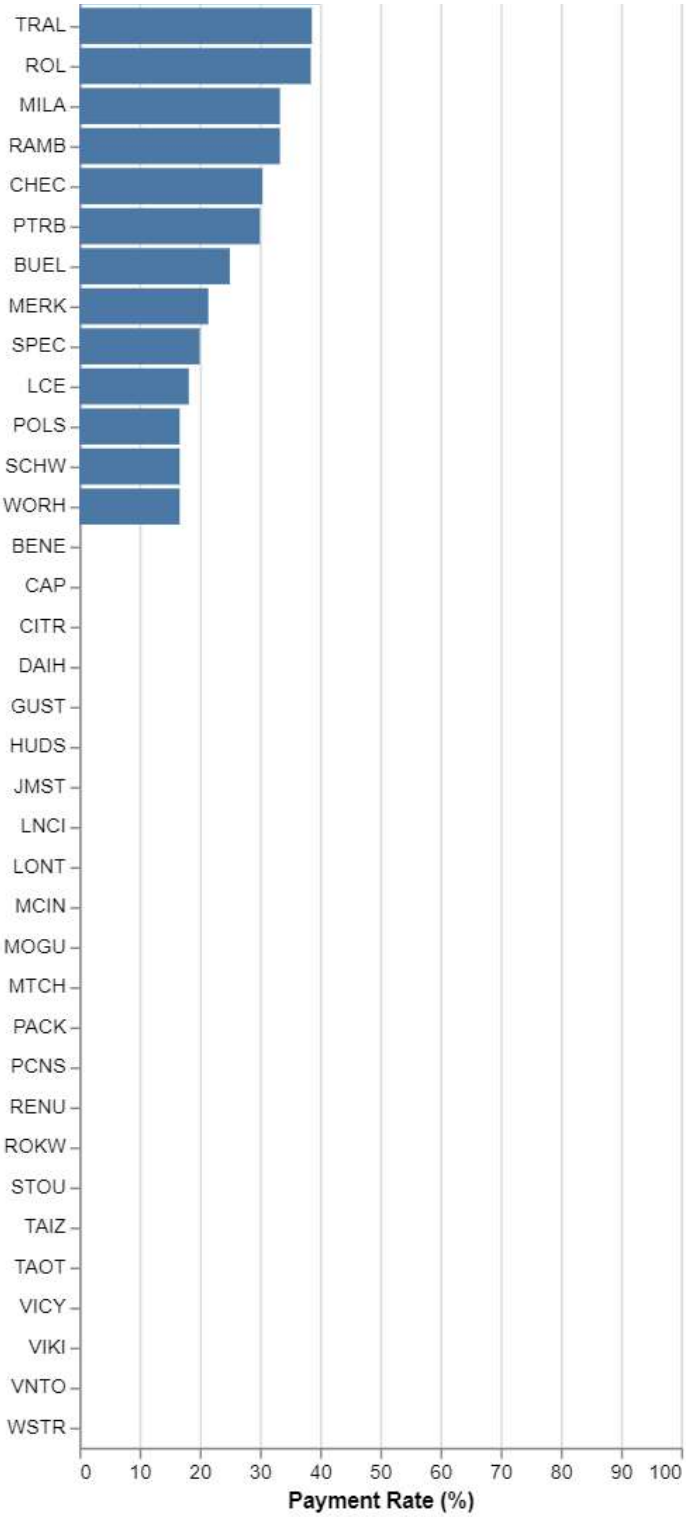
Fraction of time that tickets issued to each Vehick Make are marked as paid

```
# Getting ratio
df_vehicle_make_tickets = df.groupby("vehicle_make")["ticket_queue"].count()
df_paid = df[df["ticket_queue"] == "Paid"].groupby("vehicle_make")["ticket_queue"].count()
ticket_pay_rate = (df_paid/df_vehicle_make_tickets) * 100

# Reset index and rename columns
ticket_pay_rate = ticket_pay_rate.reset_index()
ticket_pay_rate.columns = ["Vehicle Make", "Payment Rate"]

# Making bar chart
bar_chart = alt.Chart(ticket_pay_rate).mark_bar().encode(
    x=alt.X("Payment Rate:Q", title="Payment Rate (%)"),
    y=alt.Y("Vehicle Make:N", sort="-x", title="Vehicle Make"),
).properties(
    title="Payment Rate by Vehicle Make"
)

bar_chart
```

```
C:\Users\clari\anaconda3\Lib\site-packages\altair\utils\core.py:395: FutureWarning:

the convert_dtype parameter is deprecated and will be removed in a future version.  Do
``ser.astype(object).apply()`` instead if you want ``convert_dtype=False``.
```


Payment Rate by Vehicle Make
ASTO
BRDL
DERB
MG
PEIR
PREO
SANY
SYMP
TOMO
WHIT
WILS
ZNEN
MINI
MASE
SMRT
LOTU
SCIO
PORS
SUBA
PIAG
LEXU
RROV
AUDI
FIAT
VOLK
HOND
ALFA
TOYT
VOLV
ROV
MERZ
BMW
SAAB
ACUR
HUMM
MAZD
BENT
JEEP
MITS
TESL
INFI
HYUN
NISS
LNDR
SUZI
KIA
GEO
SATR


Vehicle Make
HINO
ISU
AMER
FTWD
ROKE
SPNT
THMP
JAGU
OTHR
UNKN
FORD
GENU
GMC
PETR
FLNR
INTL
CHEV
DODG
CHRY
MERC
DUCA
LAMO
CADI
PONT
PLYM
LINC
KENW
HARL
DAEW
VESP
FERR
IND
TRIU
BUIC
EGIL
KYMC
OLDS
YAMA
APRL
AUST
BAJA
DATS
LING
VPGP
YUGO
STRG
MACK
KAWA
GELY

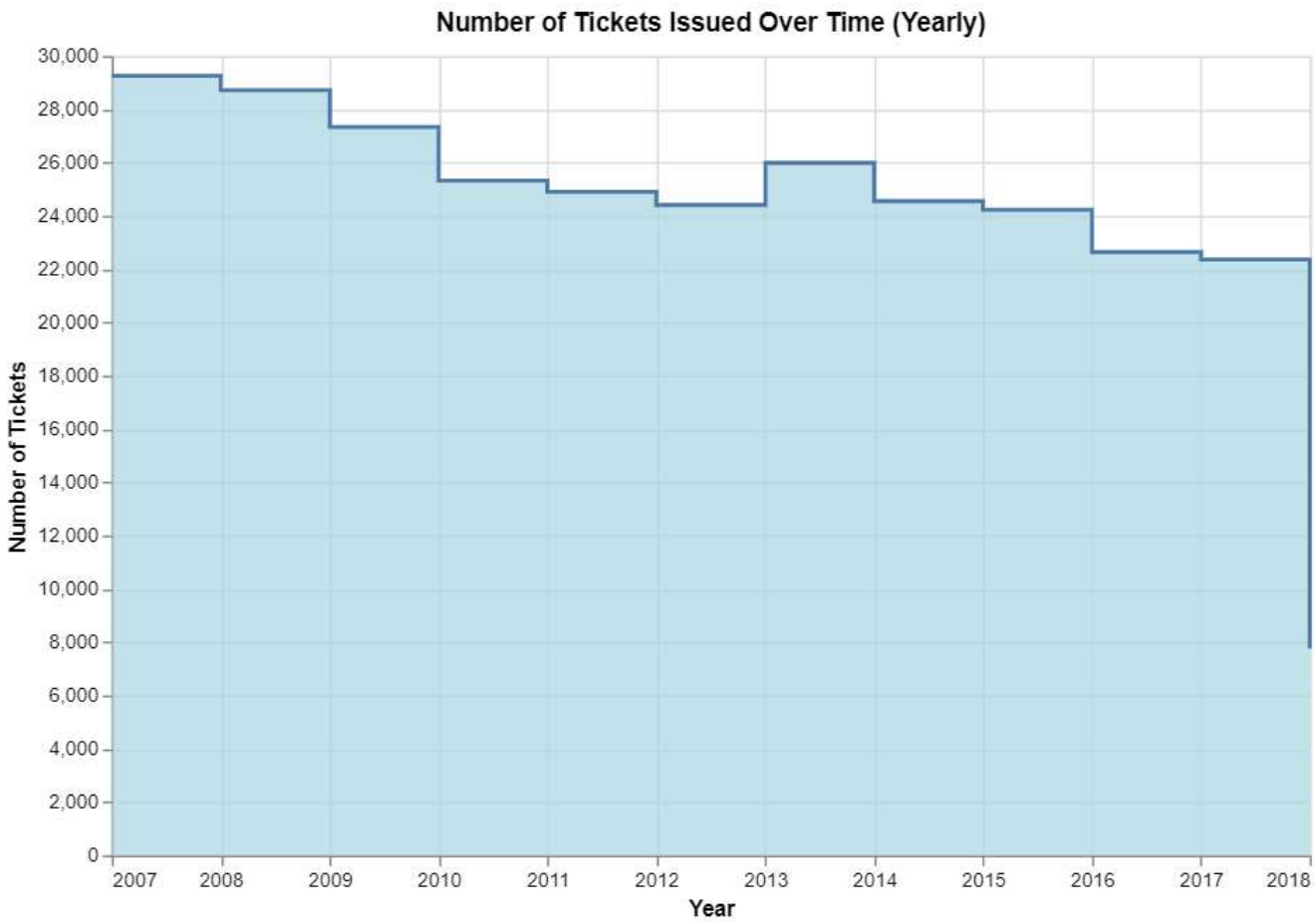


We can infer that those who have more expensive vehicle makes are also wealthier/ have more disposable income. This means that they are able to afford the high prices of the fines and pay them off quickly.This ties in well with the article, where it says that “Tickets issued in low-income, majority black neighborhoods were more likely to remain unpaid.” ALso, it seems that certain vehicle types found in pooer neighborhoods were stopped by police more often and some vehicles received multiple tickets on the same day, in these neighborhoods, making it harder to pay off all tickets, thus a lower payment rate.

Used the lecture slides for help with graphing

3. Make a plot for the number of tickets issued over time by adapting the Filled Step Chart example online. Go back to Bertin's taxonomy of visual encoding, which we discussed in lecture. What visual encoding channel or channels does this use? # Number of tickets issued over time

```
df["issue_year"] = df["issue_date"].dt.year

# Group by year and count tickets
df_overtime = df.groupby("issue_year").size().reset_index(name="count")

# Convert issue_year to datetime
df_overtime["date"] = pd.to_datetime(df_overtime["issue_year"], format="%Y")

# Making filled_step_chart
filled_step_chart = alt.Chart(df_overtime).mark_area(
    color="lightblue",
    interpolate='step-after',
    line=True
).encode(
    x=alt.X('date:T', title='Year', axis=alt.Axis(format="%Y")),
    y=alt.Y('count:Q', title='Number of Tickets')
).properties(
    title="Number of Tickets Issued Over Time (Yearly)",
    width=600,
    height=400
)

filled_step_chart
```



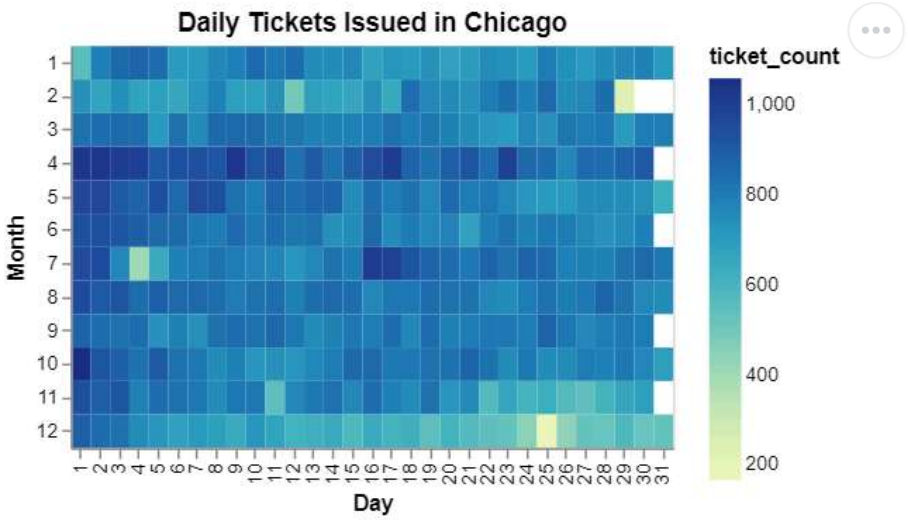
The Filled Step Chart uses the ff visual encoding channels:

- positional, where the x axis is usually the date/time/ Temporal variable and the y axis is the quantitative value (independent var) being measured.
- Area, where filled area under the step line shows the magnitude of the value over time.
- Shape- the step-like shape of the line itself creates a pattern, which alows viewers to easily see changes in the data.

4. Make a plot for the number of tickets issued by month and day by adapting the Annual Weather Heatmap example online. What visual encoding channel or channels does this use? # Heat map of tickets issued by month and day

```
df["Month"] = df['issue_date'].dt.month
df["Day"] = df['issue_date'].dt.day

# Aggregate the data
ticket_counts = df.groupby(["Month", "Day"]).size().reset_index(name="ticket_count")

# Making heat map
heat_map = alt.Chart(ticket_counts, title="Daily Tickets Issued in Chicago").mark_rect().encode(
    x="Day:O",
    y="Month:O",
    color='ticket_count:Q'
).properties(
```

```
    width=300,
    height=200
)

heat_map
```



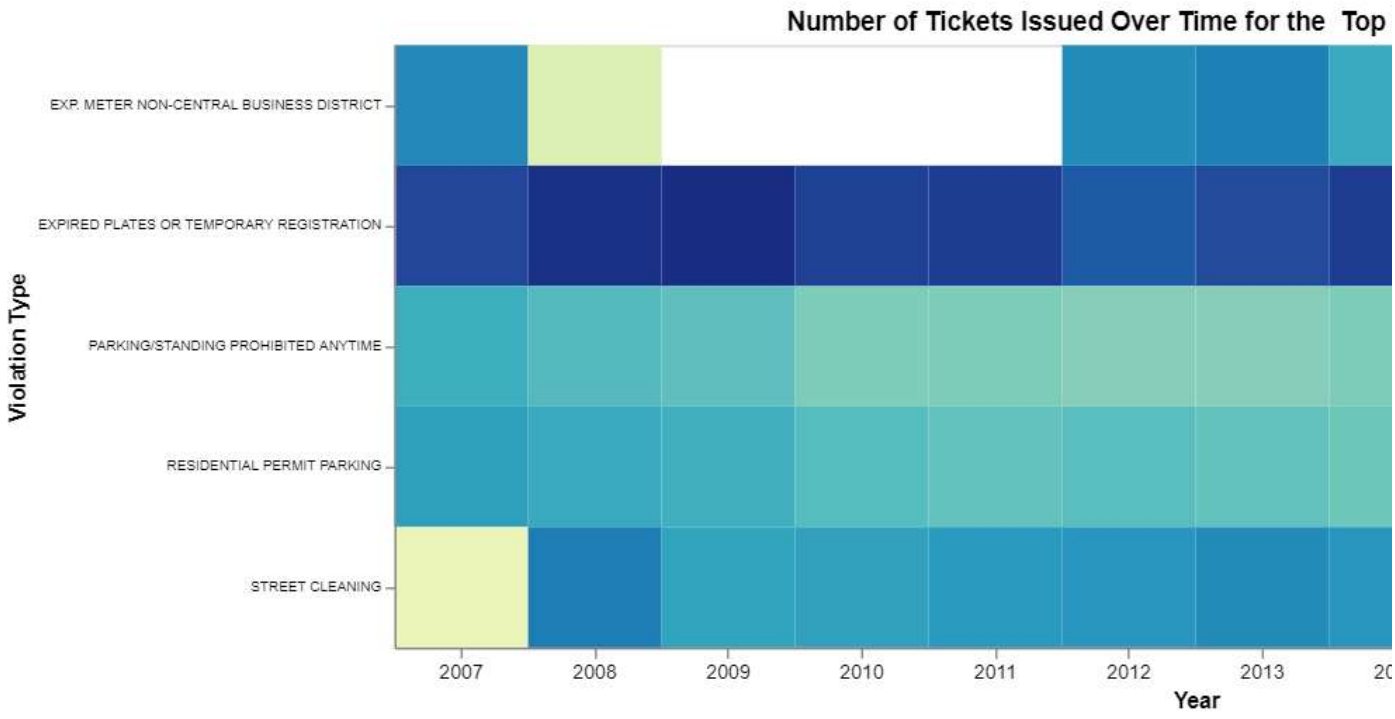
The heat map uses the ff visual encoding channels: - psotional (x and y axes), which represent the w dimensions or categories of data (month and day in this case). - color channel, which provides important information through the darkness/intensity of the color on frequency of the data for the specific month, day combination (where x and y axes meet). - area channel, where each cell in the heatmap symbolizes a specific data point.

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. What visual encoding channel or channels does this use? # Lasagna Plot

```
# List of top 5 violations
top_5_list = df['violation_description'].value_counts().head(5).index.tolist()

# FIltering to show only top 5 violations
df_top_5_violations = df[df['violation_description'].isin(top_5_list)]

# Getting year from issue_date
df_top_5_violations["issue_year"] = df_top_5_violations["issue_date"].dt.year

# Group by year and violation type andcounting the tickets
df_year_violation_type = df_top_5_violations.groupby(["issue_year", "violation_description"]).size


lasagna_plot = alt.Chart(df_year_violation_type).mark_rect().encode(
    x=alt.X("issue_year:O")
        .title("Year")
        .axis(labelAngle=0, labelOverlap=False),
    y=alt.Y("violation_description:N")
        .title("Violation Type")
```

```
        .sort('-x')
        .axis(labelFontSize=7),
    color=alt.Color("count:Q")
).properties(
    width=800,
    height=300,
    title="Number of Tickets Issued Over Time for the  Top 5 Violation Types"
)

lasagna_plot
```

```
C:\Users\clari\AppData\Local\Temp\ipykernel_9888\793750761.py:8: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

C:\Users\clari\anaconda3\Lib\site-packages\altair\utils\core.py:395: FutureWarning:

the convert_dtype parameter is deprecated and will be removed in a future version.  Do
``ser.astype(object).apply()`` instead if you want ``convert_dtype=False``.
```



This plot is a type of heat map that is able to display longitudinal data. It also uses: - position channel through the x and y axes, which create a grid where each cell represents a specific year and violation type. - area channel which clearly demarcates between different data points and helps in comparing neighboring

years or violation types. - color channel, where it chagges color based on how many tickets are there are for that violation in that year. It's like a color coded legend that tells you if there are more or less tickets. The area channel (the rectangles) provides a clear separation between different data points and helps in comparing adjacent years or violation types.

6.

Filled Step Chart

Pros: - Clear way of showing changes over time (trends), with the reader being able to get specific a more accurate reading of the variable being measured - It's easy to find the min and max points

Cons: - Limited to showing just one point of measurement over time, meaning we can't know, for example, the vehicle make or the violation type. We just know the number of violations.

Heatmap

Pros: - Effectively shows patterns and trends across two dimensions (e.g. month and day) - Color provides a clear idea to readers high/low ticket volumesbased on seasonalpatters - Can display information from large

Cons: - Are less likely to have exact values - Can't show several years at a time

Lasagna Plot

Pros: - Easier to spot trends over multiple years for multiple categories (violation types) - Allows us to compare patterns across different vcategories and years

Cons: - We can't get a good read on exact values - Colors may make it hard to decipher

Key differences:

- The step chart is the simplest to read, even for those less attuned to reading graphs. It is a simple 2 dimmensional graph showing a clear pattern, though it can provide less information. Meanwhile, the heatmap and Lasagna Plot or more complex and can show more information through both the x and y axes, as well as the colors (intensity could be frequency/magnitude). Heatmap shows less information, where we see month and day, then frequency based on the color of the cells, while the Lasagna plot can show multip, categories too.

7.

For a reader who isn't used to data analysis, I think that the filled step chart is the best choice for illustrating that the enforcement of violations is not evenly distributed over time. Because it is simple and the pattern clearly showhows changes in number of tickets issued, readers can easily identify periods of significant increase or decrease in enforcement. The year being on the x-axis also makes comparison between years very simple and the y-axis shows the measurement of the number of tickets associated witht he year. Overall, although there is less information, this chart is simple and easy for readers to understand the message/see the pattern.