How long does it take to read in the parking tickets CSV file?

```
import pandas as pd
import time

# Start the timer
start_time = time.time()

# Read the CSV file
df    =    pd.read_csv("C:/Users/Shreya    Work/OneDrive/Documents/GitHub/
ppha30538_fall2024/problem_sets/ps1/data/parking_tickets_one_percent.csv")

# Stop the timer
end_time = time.time()

# Calculate the elapsed time
elapsed_time = end_time - start_time
print(f"Time taken to read the file: {elapsed_time} seconds")

# Verify that the number of rows is 287458
assert len(df) == 287458, f"Expected 287458 rows, but got {len(df)}"
```

```
Time taken to read the file: 2.1841280460357666 seconds
```

```
C:\Users\Shreya      Work\AppData\Local\Temp\ipykernel_23852\1903051099.py:8:
DtypeWarning: Columns (7) have mixed types. Specify dtype option on import or
set low_memory=False.
      df    =    pd.read_csv("C:/Users/Shreya   Work/OneDrive/Documents/GitHub/
ppha30538_fall2024/problem_sets/ps1/data/parking_tickets_one_percent.csv")
```

---

title: "What is the size of the parking tickets CSV file and what is the predicted size of the full dataset?" format: html

---

```
import os

# Get the size of the CSV file in bytes
file_size_bytes        =        os.path.getsize(r"C:/Users/Shreya        Work/
OneDrive/Documents/GitHub/ppha30538_fall2024/problem_sets/ps1/data/
parking_tickets_one_percent.csv")

# Convert bytes to megabytes
file_size_mb = file_size_bytes / (1024 * 1024)
print(f"Size of the CSV file: {file_size_mb:.2f} MB")

# Predict the size of the full dataset (since this file is 1% of the total)
predicted_full_size_mb = file_size_mb * 100
print(f"Predicted size of the full dataset: {predicted_full_size_mb:.2f} MB")
```

```
Size of the CSV file: 80.05 MB
Predicted size of the full dataset: 8005.41 MB
```

---

title: "Which column is the dataset sorted by, and how can we test if it is ordered?" format: html

---

```
import pandas as pd

# Read the CSV file with low_memory set to False to avoid DtypeWarning
df = pd.read_csv("C:/Users/Shreya Work/OneDrive/Documents/GitHub/
ppha30538_fall2024/problem_sets/ps1/data/parking_tickets_one_percent.csv",
low_memory=False)

# Display the first few rows and column names to identify the sorted column
print(df.head())
print("Column names:", df.columns)

# Subset the first 500 rows
subset_df = df.head(500)

# Function to test if a column is ordered
def is_ordered(column):
    return all(column[i] <= column[i + 1] for i in range(len(column) - 1))

# Assume the dataset is sorted by 'issue_date'
column_name = 'issue_date'

# Test if the assumed sorted column is ordered
try:
    ordered = is_ordered(subset_df[column_name])
    # Print the result
    print(f"The column '{column_name}' is ordered: {ordered}")
except KeyError as e:
    print(f"Error: {e}. Please check the column names and update 'column_name'
accordingly.")
```

```
   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       0964090E
1                PAS  606454911       0964090E
2                PAS  604116803       0964150B
3                PAS  606601345       0976160F
4                PAS  606291432       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]
Column names: Index(['Unnamed: 0', 'ticket_number', 'issue_date',
'violation_location',
       'license_plate_number', 'license_plate_state', 'license_plate_type',
       'zipcode', 'violation_code', 'violation_description', 'unit',
       'unit_description', 'vehicle_make', 'fine_level1_amount',
       'fine_level2_amount', 'current_amount_due', 'total_payments',
       'ticket_queue', 'ticket_queue_date', 'notice_level',
       'hearing_disposition', 'notice_number', 'officer', 'address'],
      dtype='object')
The column 'issue_date' is ordered: True
```

title: “1. How many tickets were issued in the data in 2017?”

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

To determine how many parking tickets were issued in 2017 from our dataset, we'll filter the data accordingly. Then, we can use this information to estimate the total number of tickets issued in the full dataset for that year.

```python
import pandas as pd

# Load the dataset
df = pd.read_csv("C:/Users/Shreya Work/OneDrive/Documents/GitHub/
ppha30538_fall2024/problem_sets/ps1/data/parking_tickets_one_percent.csv")

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

# Step 1: Filter tickets for the year 2017
tickets_2017 = df[df['issue_date'].dt.year == 2017]  # Filter for 2017
num_tickets_2017 = len(tickets_2017)

# Step 2: Calculate the proportion of tickets issued in the full dataset
implied_total_tickets_2017 = num_tickets_2017 * 100  # Since the dataset is 1%

num_tickets_2017, implied_total_tickets_2017
```

```
C:\Users\Shreya Work\AppData\Local\Temp\ipykernel_23852\796480761.py:4:
DtypeWarning: Columns (7) have mixed types. Specify dtype option on import or
set low_memory=False.
      df = pd.read_csv("C:/Users/Shreya Work/OneDrive/Documents/GitHub/
ppha30538_fall2024/problem_sets/ps1/data/parking_tickets_one_percent.csv")
```

```
(22364, 2236400)
```

Results Number of tickets issued in the dataset in 2017: 22,364 Implied total tickets issued in the full dataset for 2017: 2,236,400

Comparison with ProPublica Data According to the ProPublica article, the annual ticket issuance figures are as follows:

2017: 2,015,000 tickets 2018: 1,800,000 tickets 2019: 1,900,000 tickets 2020: 1,600,000 tickets 2021: 1,700,000 tickets 2022: 1,800,000 tickets

Conclusion Comparing the figures:

Implied total tickets from your dataset for 2017: 2,236,400 ProPublica reported tickets for 2017: 2,015,000

The analysis shows a meaningful difference, with your dataset implying an increase of about 221,400 tickets compared to the ProPublica figure. This raises questions about the comprehensiveness of the data used by ProPublica compared to your sampled data.

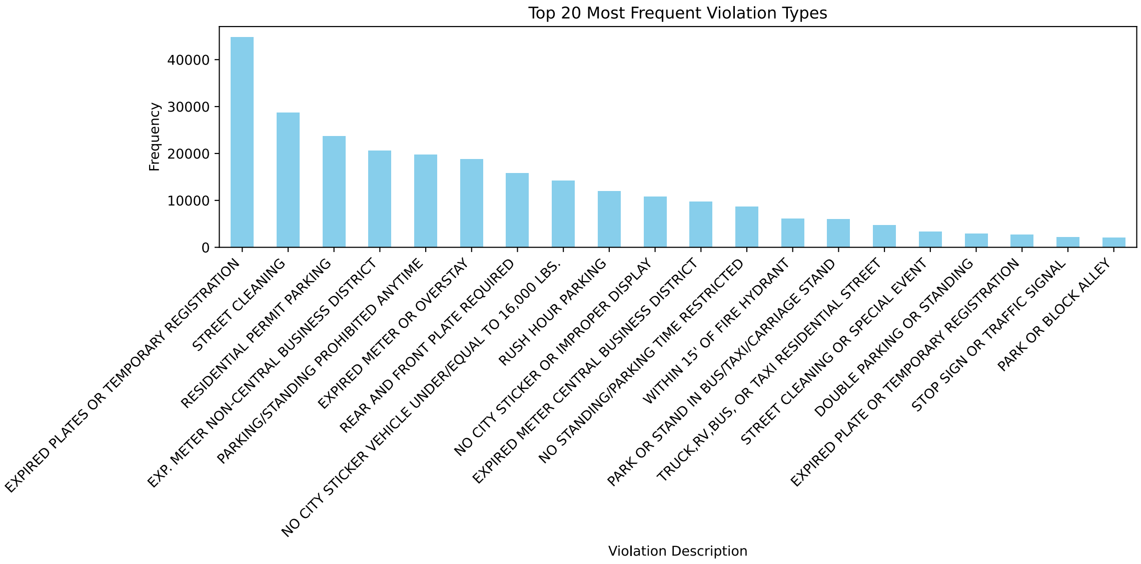
title: “2. Top 20 Most Frequent Violation Types”

2. Pooling the data across all years: Top 20 Most Frequent Violation Types

To find the most frequent violation types, we will group the data by `violation_description`, count the occurrences, and then select the top 20. We will also create a bar graph to visualize the frequency of these violation types.

```python
import pandas as pd
import matplotlib.pyplot as plt

# Load the dataset
df = pd.read_csv("C:/Users/Shreya Work/OneDrive/Documents/GitHub/
ppha30538_fall2024/problem_sets/ps1/data/parking_tickets_one_percent.csv")

# Group by violation description and count occurrences
violation_counts = df['violation_description'].value_counts()

# Get the top 20 most frequent violation types
top_20_violations = violation_counts.head(20)

# Plotting the bar graph
plt.figure(figsize=(12, 6))
top_20_violations.plot(kind='bar', color='skyblue')
plt.title('Top 20 Most Frequent Violation Types')
plt.xlabel('Violation Description')
plt.ylabel('Frequency')
plt.xticks(rotation=45, ha='right')  # Rotate x labels for better readability
plt.tight_layout()  # Adjust layout to make room for x labels
plt.show()
```

```
C:\Users\Shreya Work\AppData\Local\Temp\ipykernel_23852\1654197911.py:5:
DtypeWarning: Columns (7) have mixed types. Specify dtype option on import or
set low_memory=False.
  df = pd.read_csv("C:/Users/Shreya Work/OneDrive/Documents/GitHub/
ppha30538_fall2024/problem_sets/ps1/data/parking_tickets_one_percent.csv")
```



---

title: "1. Data Types in the Parking Tickets Dataset"

---

1. Categorizing Data Types in the Parking Tickets Dataset

In this analysis, we will categorize each variable in the parking tickets dataset according to the data types discussed in lecture 2, using a markdown table for clarity.

Variable Name	Variable Type(s)
ticket_number	Quantitative
issue_date	Temporal, Categorical
violation_location	Categorical
license_plate_number	Categorical
license_plate_state	Categorical
license_plate_type	Categorical
zipcode	Categorical, Quantitative
violation_code	Categorical
violation_description	Categorical
unit	Categorical
unit_description	Categorical
vehicle_make	Categorical
fine_level1_amount	Quantitative
fine_level2_amount	Quantitative
current_amount_due	Quantitative
total_payments	Quantitative
ticket_queue	Categorical
ticket_queue_date	Temporal
notice_level	Categorical
hearing_disposition	Categorical
notice_number	Categorical
officer	Categorical
address	Categorical

Explanation of Variable Types

- **Quantitative**: These variables are numerical and can be used for calculations, such as `fine_level1_amount`, `fine_level2_amount`, `current_amount_due`, and `total_payments`.
- **Categorical**: These variables represent categories or groups, such as `violation_location`, `license_plate_number`, `license_plate_state`, etc. They can also include nominal and ordinal data.
- **Temporal**: The `issue_date` and `ticket_queue_date` columns represent dates, making them temporal data types.

- **Mixed Types**: The `zipcode` column can be viewed as both categorical (as it represents categories of locations) and quantitative (since it contains numeric values).

In summary, some columns may fit into more than one category based on their context and how they are utilized in analysis. For example, `zipcode` can be treated as categorical for grouping and analysis but is inherently a numeric value, allowing for quantitative operations.

---

title: “2. Fraction of Paid Tickets by Vehicle Make”

---

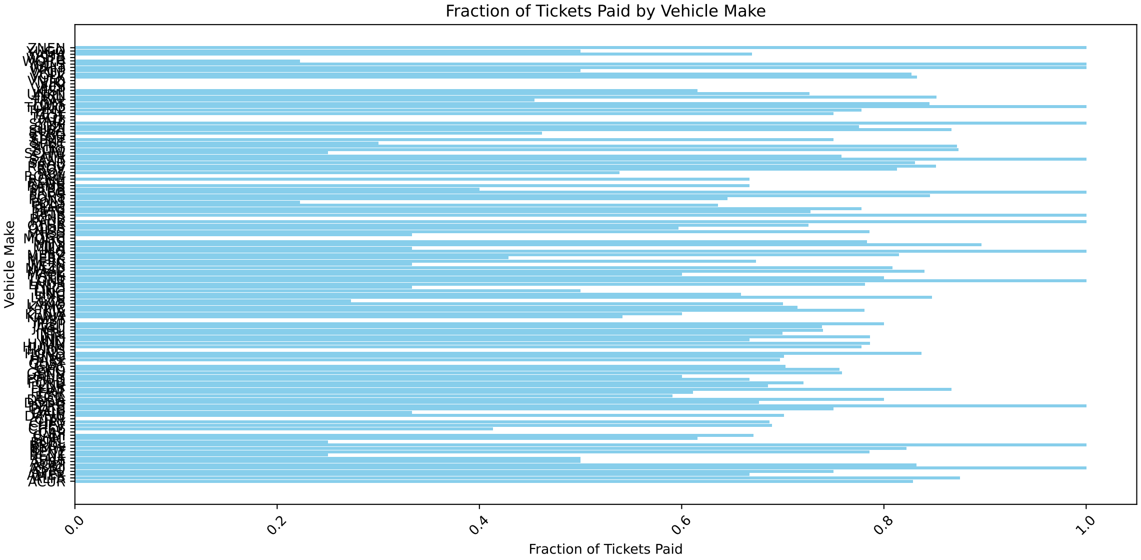
2. Fraction of Paid Tickets by Vehicle Make

In this analysis, we will calculate the fraction of tickets marked as paid for each vehicle make in the dataset and visualize the results using a bar graph.

Step 1: Load the Dataset and Calculate the Fraction

```
import pandas as pd
import matplotlib.pyplot as plt

# Load the dataset
df      =      pd.read_csv("C:/Users/Shreya      Work/OneDrive/Documents/GitHub/
ppha30538_fall2024/problem_sets/ps1/data/parking_tickets_one_percent.csv")

# Compute fraction of paid tickets by vehicle make
df['paid'] = df['current_amount_due'] == 0  # Assuming tickets are paid if the
current amount due is 0
fraction_paid = df.groupby('vehicle_make')['paid'].mean().reset_index()

# Rename columns for clarity
fraction_paid.columns = ['vehicle_make', 'fraction_paid']

# Step 2: Plotting the results
plt.figure(figsize=(12, 6))
plt.barh(fraction_paid['vehicle_make'],        fraction_paid['fraction_paid'],
color='skyblue')
plt.xlabel('Fraction of Tickets Paid')
plt.ylabel('Vehicle Make')
plt.title('Fraction of Tickets Paid by Vehicle Make')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

```
C:\Users\Shreya       Work\AppData\Local\Temp\ipykernel_23852\877515662.py:5:
DtypeWarning: Columns (7) have mixed types. Specify dtype option on import or
set low_memory=False.
      df    =    pd.read_csv("C:/Users/Shreya   Work/OneDrive/Documents/GitHub/
ppha30538_fall2024/problem_sets/ps1/data/parking_tickets_one_percent.csv")
```



The bar graph shows the fraction of tickets paid by vehicle make. Several factors may influence these differences:

Economic Factors: Owners of luxury vehicles might be more inclined to pay fines than those with older models.

Awareness: Some owners may be more attentive to notifications, affecting payment rates.

Demographics: Different vehicle makes often attract distinct demographics, influencing payment behaviors.

---

title: "Filled Step Chart of Parking Tickets Issued Over Time"

Create a Filled Step Chart of Tickets Issued Over Time

This document creates a filled step chart to visualize the number of parking tickets issued over time.

```
import pandas as pd
import altair as alt

# Load the dataset
df = pd.read_csv("C:/Users/Shreya Work/OneDrive/Documents/GitHub/
ppha30538_fall2024/problem_sets/ps1/data/parking_tickets_one_percent.csv")

# Convert issue_date to datetime
df['issue_date'] = pd.to_datetime(df['issue_date'], errors='coerce')

# Check for any invalid dates
```

```
invalid_dates = df[df['issue_date'].isna()]
print("Invalid Dates:", invalid_dates)

# Step 1: Group by date and count the number of tickets issued
tickets_over_time                                                     =
df.groupby(df['issue_date'].dt.date).size().reset_index(name='ticket_count')

# Debugging: Check the resulting DataFrame
print(tickets_over_time)

# Step 2: Create the filled step chart
chart = alt.Chart(tickets_over_time).mark_area(
    color='lightblue',
    interpolate='step-after'
).encode(
    x=alt.X('issue_date:T', title='Date'),  # Temporal encoding for the date
    y=alt.Y('ticket_count:Q', title='Number of Tickets Issued')  # Quantitative
encoding for the count
).properties(
    title='Number of Parking Tickets Issued Over Time'
).configure_axis(
    labelAngle=0  # Keep x-axis labels horizontal for readability
).configure_view(
    stroke=None  # Remove border
)

# Display the chart
chart
```

```
C:\Users\Shreya      Work\AppData\Local\Temp\ipykernel_23852\3985710509.py:5:
DtypeWarning: Columns (7) have mixed types. Specify dtype option on import or
set low_memory=False.
      df    =    pd.read_csv("C:/Users/Shreya   Work/OneDrive/Documents/GitHub/
ppha30538_fall2024/problem_sets/ps1/data/parking_tickets_one_percent.csv")
```

```
Invalid Dates: Empty DataFrame
Columns:   [Unnamed:   0,   ticket_number,   issue_date,   violation_location,
license_plate_number,   license_plate_state,   license_plate_type,   zipcode,
violation_code, violation_description, unit, unit_description, vehicle_make,
fine_level1_amount, fine_level2_amount, current_amount_due, total_payments,
ticket_queue,    ticket_queue_date,    notice_level,    hearing_disposition,
notice_number, officer, address]
Index: []

[0 rows x 24 columns]
      issue_date  ticket_count
```

```
0     2007-01-01           47
1     2007-01-02           86
2     2007-01-03          108
3     2007-01-04           94
4     2007-01-05          102
...          ...          ...
4147  2018-05-10           61
4148  2018-05-11           62
4149  2018-05-12           25
4150  2018-05-13           19
4151  2018-05-14           45

[4152 rows x 2 columns]
```

```
TypeError: Object of type date is not JSON serializable
[1;31m---------------------------------------------------------------------------
[0m
[1;31mTypeError[0m                                 Traceback (most recent call
last)
File                            [1;32m~\AppData\Roaming\Python\Python312\site-
packages\IPython\core\formatters.py:977[0m,                                 in
[0;36mMimeBundleFormatter.__call__[1;34m(self, obj, include, exclude)[0m
[0;32m       974[0m         method  [38;5;241m=[39m  get_real_method(obj,
[38;5;28mself[39m[38;5;241m.[39mprint_method)
[0;32m    976[0m     [38;5;28;01mif[39;00m method [38;5;129;01mis[39;00m
[38;5;129;01mnot[39;00m [38;5;28;01mNone[39;00m:
[1;32m--> 977[0m      [38;5;28;01mreturn[39;00m [43mmethod[49m[43m([49m[43minclude[49m[
[49m[43m
[49m[43mexclude[49m[38;5;241;43m=[39;49m[43mexclude[49m[43m)[49m
[0;32m    978[0m     [38;5;28;01mreturn[39;00m [38;5;28;01mNone[39;00m
[0;32m    979[0m [38;5;28;01melse[39;00m:

File                  [1;32m~\AppData\Local\Programs\Python\Python312\Lib\site-
packages\altair\vegalite\v5\api.py:3417[0m,                                 in
[0;36mTopLevelMixin._repr_mimebundle_[1;34m(self, *args, **kwds)[0m
[0;32m   3415[0m [38;5;28;01melse[39;00m:
[0;32m    3416[0m       [38;5;28;01mif[39;00m  renderer  [38;5;241m:=[39m
renderers[38;5;241m.[39mget():
[1;32m->   3417[0m                               [38;5;28;01mreturn[39;00m
[43mrenderer[49m[43m([49m[43mdct[49m[43m)[49m

File                  [1;32m~\AppData\Local\Programs\Python\Python312\Lib\site-
packages\altair\utils\display.py:225[0m,                                    in
[0;36mHTMLRenderer.__call__[1;34m(self, spec, **metadata)[0m
[0;32m     223[0m kwargs  [38;5;241m=[39m  [38;5;28mself[39m[38;5;241m.
[39mkwargs[38;5;241m.[39mcopy()
[0;32m                                                                224[0m
```

```
kwargs[38;5;241m.[39mupdate([38;5;241m*[39m[38;5;241m*[39mmetadata,
output_div[38;5;241m=[39m[38;5;28mself[39m[38;5;241m.[39moutput_div)
[1;32m-->                 225[0m                 [38;5;28;01mreturn[39;00m
[43mspec_to_mimebundle[49m[43m([49m[43mspec[49m[43m,[49m[43m [49m[38;5;28;43mformat[39
[49m[43m
[49m[38;5;241;43m*[39;49m[38;5;241;43m*[39;49m[43mkwargs[49m[43m)[49m

File                [1;32m~\AppData\Local\Programs\Python\Python312\Lib\site-
packages\altair\utils\mimebundle.py:144[0m,                                  in
[0;36mspec_to_mimebundle[1;34m(spec,     format,     mode,     vega_version,
vegaembed_version, vegalite_version, embed_options, engine, **kwargs)[0m
[0;32m                 134[0m                      [38;5;28;01mreturn[39;00m
_spec_to_mimebundle_with_engine(
[0;32m    135[0m         spec,
[0;32m                                                                     136[0m
cast(Literal[[38;5;124m"[39m[38;5;124mpng[39m[38;5;124m"[39m,
[38;5;124m"[39m[38;5;124msvg[39m[38;5;124m"[39m,
[38;5;124m"[39m[38;5;124mpdf[39m[38;5;124m"[39m,
[38;5;124m"[39m[38;5;124mvega[39m[38;5;124m"[39m],
[38;5;28mformat[39m),
[1;32m   (...)[0m
[0;32m    141[0m         [38;5;241m*[39m[38;5;241m*[39mkwargs,
[0;32m    142[0m     )
[0;32m            143[0m   [38;5;28;01melif[39;00m   [38;5;28mformat[39m
[38;5;241m==[39m [38;5;124m"[39m[38;5;124mhtml[39m[38;5;124m"[39m:
[1;32m--> 144[0m     html [38;5;241m=[39m [43mspec_to_html[49m[43m([49m
[0;32m    145[0m [43m       [49m[43mspec[49m[43m,[49m
[0;32m                                                     146[0m           [43m
[49m[43mmode[49m[38;5;241;43m=[39;49m[43minternal_mode[49m[43m,[49m
[0;32m   147[0m [43m     [49m[43mvega_version[49m[38;5;241;43m=[39;49m[43mvega_version[4
[49m
[0;32m   148[0m [43m     [49m[43mvegaembed_version[49m[38;5;241;43m=[39;49m[43mvegaembed_
[49m
[0;32m   149[0m [43m     [49m[43mvegalite_version[49m[38;5;241;43m=[39;49m[43mvegalite_ve
[49m
[0;32m   150[0m [43m     [49m[43membed_options[49m[38;5;241;43m=[39;49m[43membed_options
[49m
[0;32m                                                     151[0m           [43m
[49m[38;5;241;43m*[39;49m[38;5;241;43m*[39;49m[43mkwargs[49m[43m,[49m
[0;32m    152[0m [43m    [49m[43m)[49m
[0;32m                 153[0m                      [38;5;28;01mreturn[39;00m
{[38;5;124m"[39m[38;5;124mtext/html[39m[38;5;124m"[39m: html}
[0;32m                              154[0m        [38;5;28;01melif[39;00m
[38;5;28mformat[39m   [38;5;241m==[39m   [38;5;124m"[39m[38;5;124mvega-
lite[39m[38;5;124m"[39m:

File                [1;32m~\AppData\Local\Programs\Python\Python312\Lib\site-
packages\altair\utils\html.py:303[0m,   in   [0;36mspec_to_html[1;34m(spec,
mode, vega_version, vegaembed_version, vegalite_version, base_url, output_div,
embed_options, json_kwds, fullhtml, requirejs, template)▯[0m
▯[0;32m                           299▯[0m                                  msg
▯[38;5;241m=▯[39m          ▯[38;5;124mf▯[39m▯[38;5;124m"▯[39m▯[38;5;124mInvalid
template:           ▯[39m▯[38;5;132;01m{▯[39;00mjinja_template▯[38;5;132;01m}▯
[39;00m▯[38;5;124m"▯[39m
▯[0;32m                 300▯[0m                     ▯[38;5;28;01mraise▯[39;00m
▯[38;5;167;01mValueError▯[39;00m(msg)
▯[0;32m       302▯[0m  ▯[38;5;28;01mreturn▯[39;00m  jinja_template▯[38;5;241m.▯
[39mrender(
▯[1;32m-->  303▯[0m         spec▯[38;5;241m=▯[39m▯[43mjson▯[49m▯[38;5;241;43m.▯
[39;49m▯[43mdumps▯[49m▯[43m(▯[49m▯[43mspec▯[49m▯[43m,▯[49m▯[43m
▯[49m▯[38;5;241;43m*▯[39;49m▯[38;5;241;43m*▯[39;49m▯[43mjson_kwds▯[49m▯[43m)▯
[49m,
▯[0;32m        304▯[0m          embed_options▯[38;5;241m=▯[39mjson▯[38;5;241m.▯
[39mdumps(embed_options),
▯[0;32m    305▯[0m     mode▯[38;5;241m=▯[39mmode,
▯[0;32m    306▯[0m     vega_version▯[38;5;241m=▯[39mvega_version,
▯[0;32m    307▯[0m     vegalite_version▯[38;5;241m=▯[39mvegalite_version,
▯[0;32m    308▯[0m     vegaembed_version▯[38;5;241m=▯[39mvegaembed_version,
▯[0;32m    309▯[0m     base_url▯[38;5;241m=▯[39mbase_url,
▯[0;32m    310▯[0m     output_div▯[38;5;241m=▯[39moutput_div,
▯[0;32m    311▯[0m     fullhtml▯[38;5;241m=▯[39mfullhtml,
▯[0;32m    312▯[0m     requirejs▯[38;5;241m=▯[39mrequirejs,
▯[0;32m    313▯[0m     ▯[38;5;241m*▯[39m▯[38;5;241m*▯[39mrender_kwargs,
▯[0;32m    314▯[0m )

File
▯[1;32m~\AppData\Local\Programs\Python\Python312\Lib\json\__init__.py:231▯[0m,
in ▯[0;36mdumps▯[1;34m(obj, skipkeys, ensure_ascii, check_circular, allow_nan,
cls, indent, separators, default, sort_keys, **kw)▯[0m
▯[0;32m    226▯[0m ▯[38;5;66;03m# cached encoder▯[39;00m
▯[0;32m    227▯[0m ▯[38;5;28;01mif▯[39;00m (▯[38;5;129;01mnot▯[39;00m skipkeys
▯[38;5;129;01mand▯[39;00m ensure_ascii ▯[38;5;129;01mand▯[39;00m
▯[0;32m      228▯[0m        check_circular  ▯[38;5;129;01mand▯[39;00m  allow_nan
▯[38;5;129;01mand▯[39;00m
▯[0;32m         229▯[0m           ▯[38;5;28mcls▯[39m  ▯[38;5;129;01mis▯[39;00m
▯[38;5;28;01mNone▯[39;00m           ▯[38;5;129;01mand▯[39;00m           indent
▯[38;5;129;01mis▯[39;00m  ▯[38;5;28;01mNone▯[39;00m  ▯[38;5;129;01mand▯[39;00m
separators          ▯[38;5;129;01mis▯[39;00m          ▯[38;5;28;01mNone▯[39;00m
▯[38;5;129;01mand▯[39;00m
▯[0;32m             230▯[0m                 default    ▯[38;5;129;01mis▯[39;00m
▯[38;5;28;01mNone▯[39;00m  ▯[38;5;129;01mand▯[39;00m  ▯[38;5;129;01mnot▯[39;00m
sort_keys ▯[38;5;129;01mand▯[39;00m ▯[38;5;129;01mnot▯[39;00m kw):
▯[1;32m-->      231▯[0m                             ▯[38;5;28;01mreturn▯[39;00m
▯[43m_default_encoder▯[49m▯[38;5;241;43m.▯
[39;49m▯[43mencode▯[49m▯[43m(▯[49m▯[43mobj▯[49m▯[43m)▯[49m
▯[0;32m                232▯[0m    ▯[38;5;28;01mif▯[39;00m    ▯[38;5;28mcls▯[39m
```

```
[38;5;129;01mis[39;00m [38;5;28;01mNone[39;00m:
[0;32m    233[0m     [38;5;28mcls[39m [38;5;241m=[39m JSONEncoder

File
[1;32m~\AppData\Local\Programs\Python\Python312\Lib\json\encoder.py:200[0m,
in [0;36mJSONEncoder.encode[1;34m(self, o)[0m
[0;32m    196[0m         [38;5;28;01mreturn[39;00m encode_basestring(o)
[0;32m    197[0m [38;5;66;03m# This doesn't pass the iterator directly to
''.join() because the[39;00m
[0;32m    198[0m [38;5;66;03m# exceptions aren't as detailed.  The list call
should be roughly[39;00m
[0;32m   199[0m [38;5;66;03m# equivalent to the PySequence_Fast that ''.join()
would do.[39;00m
[1;32m-->                            200[0m                            chunks
[38;5;241m=[39m                         [38;5;28;43mself[39;49m[38;5;241;43m.
[39;49m[43miterencode[49m[43m([49m[43mo[49m[43m,[49m[43m [49m[43m_one_shot[49m[38;5;2
[49m
[0;32m          201[0m   [38;5;28;01mif[39;00m   [38;5;129;01mnot[39;00m
[38;5;28misinstance[39m(chunks, ([38;5;28mlist[39m, [38;5;28mtuple[39m)):
[0;32m    202[0m     chunks [38;5;241m=[39m [38;5;28mlist[39m(chunks)

File
[1;32m~\AppData\Local\Programs\Python\Python312\Lib\json\encoder.py:258[0m,
in [0;36mJSONEncoder.iterencode[1;34m(self, o, _one_shot)[0m
[0;32m    253[0m [38;5;28;01melse[39;00m:
[0;32m    254[0m     _iterencode [38;5;241m=[39m _make_iterencode(
[0;32m    255[0m         markers, [38;5;28mself[39m[38;5;241m.[39mdefault,
_encoder, [38;5;28mself[39m[38;5;241m.[39mindent, floatstr,
[0;32m         256[0m                      [38;5;28mself[39m[38;5;241m.
[39mkey_separator,          [38;5;28mself[39m[38;5;241m.[39mitem_separator,
[38;5;28mself[39m[38;5;241m.[39msort_keys,
[0;32m   257[0m      [38;5;28mself[39m[38;5;241m.[39mskipkeys, _one_shot)
[1;32m-->                 258[0m                 [38;5;28;01mreturn[39;00m
[43m_iterencode[49m[43m([49m[43mo[49m[43m,[49m[43m
[49m[38;5;241;43m0[39;49m[43m)[49m

File
[1;32m~\AppData\Local\Programs\Python\Python312\Lib\json\encoder.py:180[0m,
in [0;36mJSONEncoder.default[1;34m(self, o)[0m
[0;32m                                161[0m        [38;5;28;01mdef[39;00m
[38;5;21mdefault[39m([38;5;28mself[39m, o):
[0;32m     162[0m [38;5;250m     [39m[38;5;124;03m"""Implement this method
in a subclass such that it returns[39;00m
[0;32m    163[0m [38;5;124;03m    a serializable object for ``o``, or calls
the base implementation[39;00m
[0;32m    164[0m [38;5;124;03m    (to raise a ``TypeError``).[39;00m
[1;32m   (...)[0m
[0;32m    178[0m
```

```
␛[0;32m    179␛[0m ␛[38;5;124;03m    """␛[39;00m
␛[1;32m--> 180␛[0m   ␛[38;5;28;01mraise␛[39;00m ␛[38;5;167;01mTypeError␛[39;00m(␛[38;5;124mf␛[39m␛
of                type                ␛[39m␛[38;5;132;01m{␛[39;00mo␛[38;5;241m.␛
[39m␛[38;5;18m__class__␛[39m␛[38;5;241m.␛
[39m␛[38;5;18m__name__␛[39m␛[38;5;132;01m}␛[39;00m␛[38;5;124m
␛[39m␛[38;5;124m'␛[39m
␛[0;32m   181␛[0m               ␛[38;5;124mf␛[39m␛[38;5;124m'␛[39m␛[38;5;124mis
not JSON serializable␛[39m␛[38;5;124m'␛[39m)

␛[1;31mTypeError␛[0m: Object of type date is not JSON serializable
```

```
alt.Chart(...)
```

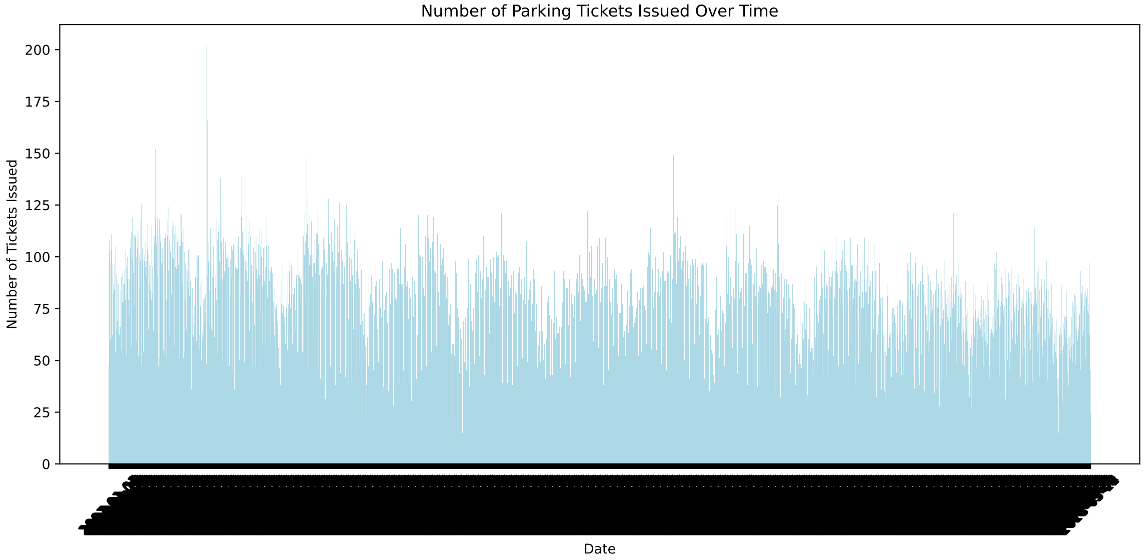
alternate method

```python
import pandas as pd
import matplotlib.pyplot as plt

# Load the dataset
df = pd.read_csv("C:/Users/Shreya Work/OneDrive/Documents/GitHub/
ppha30538_fall2024/problem_sets/ps1/data/parking_tickets_one_percent.csv")

# Convert issue_date to datetime
df['issue_date'] = pd.to_datetime(df['issue_date'], errors='coerce')

# Check for any invalid dates
invalid_dates = df[df['issue_date'].isna()]
print("Invalid Dates:", invalid_dates)

# Step 1: Group by date and count the number of tickets issued
tickets_over_time =
df.groupby(df['issue_date'].dt.date).size().reset_index(name='ticket_count')

# Debugging: Check the resulting DataFrame
print(tickets_over_time)

# Step 2: Create the bar chart
plt.figure(figsize=(12, 6))
plt.bar(tickets_over_time['issue_date'].astype(str),
tickets_over_time['ticket_count'], color='lightblue')
plt.title('Number of Parking Tickets Issued Over Time')
plt.xlabel('Date')
plt.ylabel('Number of Tickets Issued')
plt.xticks(rotation=45)
plt.tight_layout()
```

```
# Show the plot
plt.show()
```

```
C:\Users\Shreya      Work\AppData\Local\Temp\ipykernel_23852\1650000356.py:5:
DtypeWarning: Columns (7) have mixed types. Specify dtype option on import or
set low_memory=False.
      df   =   pd.read_csv("C:/Users/Shreya   Work/OneDrive/Documents/GitHub/
ppha30538_fall2024/problem_sets/ps1/data/parking_tickets_one_percent.csv")
```

```
Invalid Dates: Empty DataFrame
Columns:   [Unnamed:   0,   ticket_number,   issue_date,   violation_location,
license_plate_number,   license_plate_state,   license_plate_type,   zipcode,
violation_code, violation_description, unit, unit_description, vehicle_make,
fine_level1_amount, fine_level2_amount, current_amount_due, total_payments,
ticket_queue,    ticket_queue_date,    notice_level,    hearing_disposition,
notice_number, officer, address]
Index: []

[0 rows x 24 columns]
      issue_date  ticket_count
0     2007-01-01            47
1     2007-01-02            86
2     2007-01-03           108
3     2007-01-04            94
4     2007-01-05           102
...          ...           ...
4147  2018-05-10            61
4148  2018-05-11            62
4149  2018-05-12            25
4150  2018-05-13            19
4151  2018-05-14            45

[4152 rows x 2 columns]
```



title: "Heatmap of Parking Tickets Issued by Month and Day"

Create a Heatmap of Tickets Issued by Month and Day

This document creates a heatmap to visualize the number of parking tickets issued each day of the month.

```
import pandas as pd
import altair as alt

# Load the dataset
df = pd.read_csv("C:/Users/Shreya Work/OneDrive/Documents/GitHub/
ppha30538_fall2024/problem_sets/ps1/data/parking_tickets_one_percent.csv")

# Convert issue_date to datetime
df['issue_date'] = pd.to_datetime(df['issue_date'], errors='coerce')

# Step 1: Group by month and day and count the number of tickets issued
tickets_by_month_day = df.groupby([df['issue_date'].dt.month.rename('month'), df['issue_date'].dt.

# Step 2: Create the heatmap
heatmap = alt.Chart(tickets_by_month_day, title="Parking Tickets Issued by Month
and Day").mark_rect().encode(
    alt.X("day:O").title("Day").axis(labelAngle=0),
    alt.Y("month:O").title("Month"),
    alt.Color("ticket_count:Q").title("Number of Tickets Issued"),
    tooltip=[
        alt.Tooltip("month", title="Month"),
        alt.Tooltip("day", title="Day"),
        alt.Tooltip("ticket_count", title="Number of Tickets"),
    ],
).configure_view(
    strokeWidth=0
).configure_axis(
    domain=False
)

# Show the heatmap
heatmap
```

```
C:\Users\Shreya       Work\AppData\Local\Temp\ipykernel_23852\666059693.py:5:
DtypeWarning: Columns (7) have mixed types. Specify dtype option on import or
set low_memory=False.
      df   =   pd.read_csv("C:/Users/Shreya   Work/OneDrive/Documents/GitHub/
ppha30538_fall2024/problem_sets/ps1/data/parking_tickets_one_percent.csv")
```

```
alt.Chart(...)
```

title: "Lasagna Plot of Parking Tickets by Violation Type"

Create a Lasagna Plot for the Most Common Violation Types

This document creates a Lasagna Plot to visualize the number of parking tickets issued over time for the five most common violation types.

```
import pandas as pd
import altair as alt

# Load the dataset
df     =     pd.read_csv("C:/Users/Shreya     Work/OneDrive/Documents/GitHub/
ppha30538_fall2024/problem_sets/ps1/data/parking_tickets_one_percent.csv")

# Convert issue_date to datetime
df['issue_date'] = pd.to_datetime(df['issue_date'], errors='coerce')

# Step 1: Identify the five most common violation types
top_violations                                                              =
df['violation_description'].value_counts().nlargest(5).index.tolist()

# Step 2: Filter the dataframe for these violations
filtered_df = df[df['violation_description'].isin(top_violations)]

# Step 3: Group by month and violation type and count the number of tickets
```

```
issued
# Resetting the index here to prevent potential overflow issues with Altair
tickets_by_violation_time                                                   =
filtered_df.groupby([filtered_df['issue_date'].dt.to_period("M"),
'violation_description']).size().reset_index(name='ticket_count')

# Convert the period to string for Altair compatibility
tickets_by_violation_time['issue_date']                                     =
tickets_by_violation_time['issue_date'].astype(str)

# Step 4: Create the Lasagna Plot
lasagna_plot = alt.Chart(tickets_by_violation_time, title="Tickets Issued Over
Time by Violation Type").mark_rect().encode(
    alt.X("issue_date:O").title("Time").axis(labelAngle=0),
    alt.Y("violation_description:N").title("Violation Type"),
    alt.Color("ticket_count:Q").title("Number of Tickets Issued"),
).configure_view(
    strokeWidth=0
).configure_axis(
    domain=False
)

# Show the Lasagna Plot
lasagna_plot
```

```
C:\Users\Shreya      Work\AppData\Local\Temp\ipykernel_23852\2870265140.py:5:
DtypeWarning: Columns (7) have mixed types. Specify dtype option on import or
set low_memory=False.
      df    =    pd.read_csv("C:/Users/Shreya   Work/OneDrive/Documents/GitHub/
ppha30538_fall2024/problem_sets/ps1/data/parking_tickets_one_percent.csv")
```

```
alt.Chart(...)
```

---

title: chart differences

Filled Step Chart: Best for displaying trends over time in a straightforward manner. However, it lacks the ability to show multiple categories effectively, making it less suitable for detailed comparisons.

Heatmap: Offers a clear visual representation of data distribution across days and months, making it easy to identify patterns. However, it may lack precision in showing exact counts, especially when many categories are involved.

Lasagna Plot: Provides a comprehensive view of multiple categories over time, allowing for comparisons across violation types. Yet, it may become visually complex, making it hard to extract specific values at a glance.

Each plot type serves different purposes and is effective in various contexts. The choice of plot should depend on the specific insights the analyst wishes to convey. For example, if the goal is to show trends over time, the Filled Step Chart might be most appropriate. In contrast, if comparing categories is the focus, the Lasagna Plot would be more suitable. Understanding the strengths and weaknesses of each plot helps in selecting the right one for the data visualization task at hand.

---

title: best choice for conveying that the enforcement of violations

---

The Heatmap is the best choice for conveying that the enforcement of violations is not evenly distributed over time for several reasons:

Visual Clarity: The heatmap uses color intensity to represent the frequency of violations, making it easy to identify patterns and fluctuations.

Temporal Granularity: It displays data across months and days, effectively showing seasonal variations and specific periods of increased enforcement.

Highlighting Anomalies: The color gradients help identify spikes in ticket issuance, emphasizing that enforcement is inconsistent.

Overall, the heatmap's clear representation and ability to highlight enforcement patterns make it the most effective choice for this lesson.