Problem Set I

AUTHOR
Summer(Samar) Negahdar

PUBLISHED
Invalid Date

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: **\1** Late coins left after submission: **\3**
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

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

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

Read in one percent sample (15 Points)

1.

```
##I looked at a stackflow post to find how to do it!
import time

start_time= time.time()

data_set_1 = pd.read_csv(
    "/Users/samarnegahdar/Documents/school/Fall quarter 2024/Python II/ppha30538_fall2024

end_time= time.time()
time_spent= end_time - start_time
print("the time spent running this csv file is", time_spent)
## The Assert statement to see if the length of dataset is 287458 rows
```

```
assert len(data_set_1)== 287458
print("length of dataset is:",len(data_set_1))
```

```
the time spent running this csv file is 1.2493500709533691
length of dataset is: 287458
```

2.

```
import os
##I want to see how many Bytes are there in the csv file:
File_size_in_bytes = os.path.getsize("/Users/samarnegahdar/Documents/school/Fall quarter
File_size_in_GB = File_size_in_bytes / (1024 ** 3)
print("CSV file is:", File_size_in_GB, "GB")
##If we wanna predict how large the dataset is we have
##  to multiply the file size by 100
##(since the csv file is only 1% of the file)
Full_size_file= File_size_in_GB * 100
```

```
CSV file is: 0.07791011407971382 GB
```

3. I took a look at the CSV file and realized they are ordered based on date and time (issue-date column). now I wanna check if these column was supposed to be the default ordering column.

```
import pandas as pd

##so now I have to find that one column based on which the dataset is ordered.

##I made a mistake! I had forgotten to remove the first
##column that is the index column so:
valid_columns = data_set_1.columns[
    data_set_1.columns.str.contains('') == False]
def find_ordering_column(dataframe):
    for column in valid_columns:
        if dataframe[
            column].is_monotonic_increasing or dataframe[
                column].is_monotonic_decreasing:
            print(f"The dataset is ordered by the column: {column}")
            return column
    print("No column appears to order the dataset.")
    return None  ## I could not get the
            ##funciton right so I looked this up and it
            ## turns out adding "return" means exit the
            ## function once you have ofund one column
            ## that meets the if clause
    print("No single column appears to order the dataset")
    return None
```

No column is the base for ordering it but the first column which is the index?? should we consider that as the desired column basedo n which the dataset is ordered? if so, it will be like this:

```
import pandas as pd

def find_ordering_column(dataframe):
    for column in dataframe.columns:
        if dataframe[column
        ].is_monotonic_increasing or dataframe[
            column].is_monotonic_decreasing:
            print(f"The dataset is ordered by the column: {column}")
            return column
    print("No column appears to order the dataset.")
    return None

ordering_column = find_ordering_column(data_set_1)

##the subsetted dataset
first_500 = data_set_1.head(500)

##the function is going to be the same as above!
def is_column_ordered(subset, column):
    if subset[column
    ].is_monotonic_increasing or subset[
        column].is_monotonic_decreasing:
        return f"The column '{column}' is ordered."
    else:
        return f"The column '{column}' is not ordered."

##now I am oging to test it on the subsetted dataset
desired_col= find_ordering_column(data_set_1)

if desired_col:
    result = is_column_ordered(first_500, desired_col)
    print(result)
```

```
The dataset is ordered by the column: Unnamed: 0
The dataset is ordered by the column: Unnamed: 0
The column 'Unnamed: 0' is ordered.
```

Cleaning the data and benchmarking (15 Points)

1.

```
# Define start and end dates for 2017
start_2017 = "2017-01-01 00:00:00"
end_2017 = "2017-12-31 23:59:59"
```

```
##I had not written this line and the whole file
#crashed and I was panicking so I asked GPT for
#help by putting in the error and added
#this line to it!
data_set_1["issue_date"] = pd.to_datetime(data_set_1["issue_date"])

# I want a new dataset that filters all 2017!
data_2017_aggregated = data_set_1[(data_set_1["issue_date"] >= start_2017) &
(data_set_1["issue_date"] <= end_2017)]

# Print the number of tickets issued in 2017
print("The number of tickets issued in 2017 is", len(data_2017_aggregated))
```

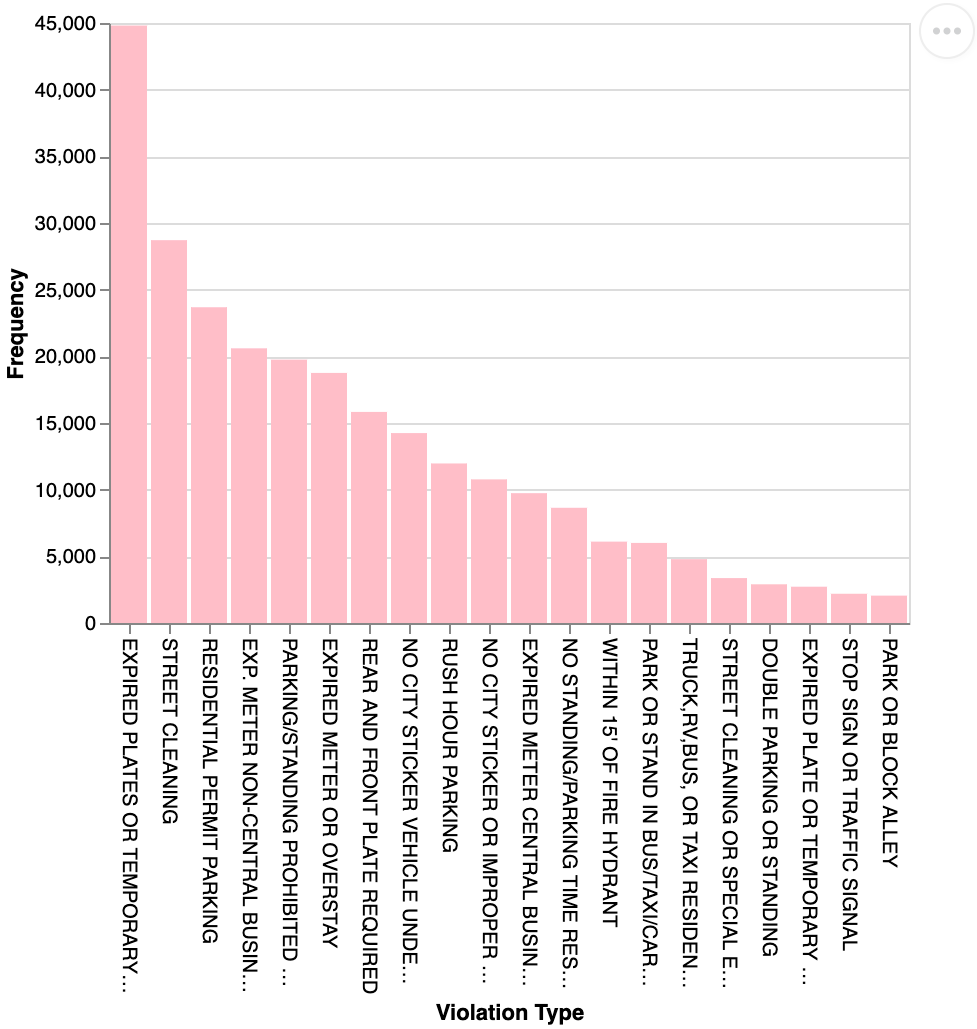
```
The number of tickets issued in 2017 is 22364
```

2.

```
##I first see how many times each violation has occured
violation_occurance= data_set_1["violation_description"].value_counts()
##now I see the top 20
top_20_viol_table= violation_occurance.head(20).reset_index()

##I am going to create a dataset(or a table)
##containing the top 20 violations and their
##frequencies
top_20_viol_table.columns = ['violation_type', 'frequency']
print(top_20_viol_table)
```

```
                                    violation_type  frequency
0            EXPIRED PLATES OR TEMPORARY REGISTRATION      44811
1                                     STREET CLEANING      28712
2                         RESIDENTIAL PERMIT PARKING      23683
3            EXP. METER NON-CENTRAL BUSINESS DISTRICT      20600
4                 PARKING/STANDING PROHIBITED ANYTIME      19753
5                           EXPIRED METER OR OVERSTAY      18756
6                       REAR AND FRONT PLATE REQUIRED      15829
7   NO CITY STICKER VEHICLE UNDER/EQUAL TO 16,000 ...      14246
8                                    RUSH HOUR PARKING      11965
9                NO CITY STICKER OR IMPROPER DISPLAY      10773
10            EXPIRED METER CENTRAL BUSINESS DISTRICT       9736
11               NO STANDING/PARKING TIME RESTRICTED       8640
12                         WITHIN 15' OF FIRE HYDRANT       6104
13           PARK OR STAND IN BUS/TAXI/CARRIAGE STAND       6004
14           TRUCK,RV,BUS, OR TAXI RESIDENTIAL STREET       4789
15                  STREET CLEANING OR SPECIAL EVENT       3370
16                        DOUBLE PARKING OR STANDING       2904
17           EXPIRED PLATE OR TEMPORARY REGISTRATION       2720
18                      STOP SIGN OR TRAFFIC SIGNAL       2191
19                               PARK OR BLOCK ALLEY       2050
```

```
##drawing the plot I'm guessing we should do it through altair?
import altair as alt


bar_chart = alt.Chart(top_20_viol_table).mark_bar(color= "pink").encode(
    x=alt.X('violation_type:N', sort='-y', title='Violation Type'),
    y=alt.Y('frequency:Q', title='Frequency'),
    tooltip=['violation_type', 'frequency']
).configure_axisX(
    labelAngle=90  ##rotating the labels on X axis
    ##so they're readable
)

# Display the chart
bar_chart
```



Visual Encoding (15 Points)

1.

```
##first of all I wanna see how many columns I have
columns= data_set_1.shape[1]
print(columns)
##I wanna see the name of columns:
col_names = data_set_1.columns
print(col_names)
column_data_types = data_set_1.dtypes
print(column_data_types)
```

```
24
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')
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
```

```
##Now I am going to create a new dataset:
##this table will have 23 rows since I am
##ignoring the first column since it is
##index column
Column_types= {
    "Variable Name": [
```

```
    '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'
       ],
       "Data Types": [
        'Ordinal', 'Temporal',
        'Nominal', 'Nominal', 'Nominal', 'Nominal','Ordinal/Nominal', 'Nominal',
        'Nominal', 'Quantitative', 'Nominal',
        'Nominal', 'Quantitative', 'quantitative',
        'Quantitative', 'Quantitative',
        'Nominal', 'Temporal', 'Nominal',
        'Nominal', 'Ordinal',
        'Nominal', 'Nominal'
       ]
}
column_types_table = pd.DataFrame(Column_types)

# Display the DataFrame
print(column_types_table)
```

```
           Variable Name       Data Types
0          ticket_number          Ordinal
1             issue_date         Temporal
2      violation_location          Nominal
3    license_plate_number          Nominal
4     license_plate_state          Nominal
5      license_plate_type          Nominal
6                 zipcode  Ordinal/Nominal
7          violation_code          Nominal
8   violation_description          Nominal
9                    unit     Quantitative
10       unit_description          Nominal
11           vehicle_make          Nominal
12     fine_level1_amount     Quantitative
13     fine_level2_amount     quantitative
14     current_amount_due     Quantitative
15         total_payments     Quantitative
16           ticket_queue          Nominal
17      ticket_queue_date         Temporal
18           notice_level          Nominal
19    hearing_disposition          Nominal
20          notice_number          Ordinal
21                officer          Nominal
22                address          Nominal
```

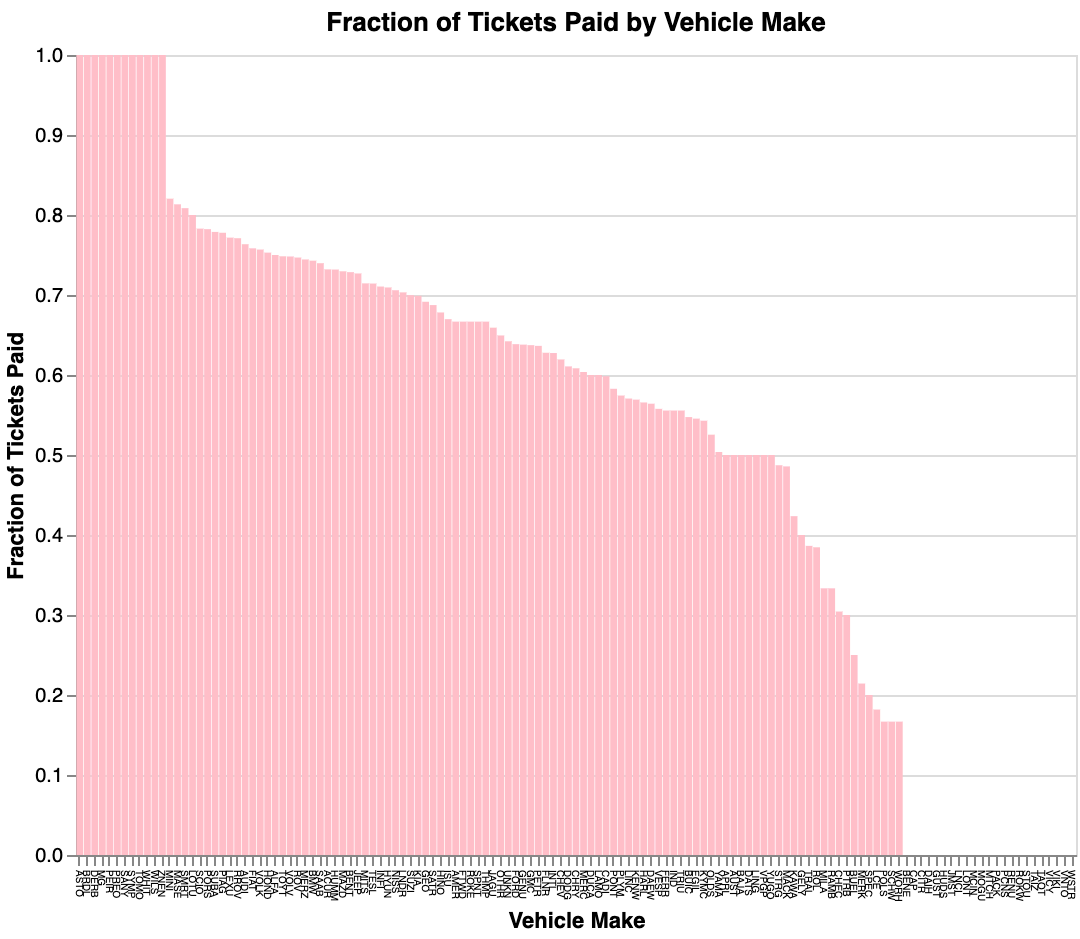
I perosnally think for the zip code we can have a kind of ranking especially in this dataset since we were talking about how lower income neighborhoods were affected so there can be a kind of order(higher

income neighborhoods and lower income neighborhoods!) Or even like we can order neighborhoods from North to South... so many ways we can rank them based on Zip code!

2. At first I did not get the quesiton(I think it is bc the column called vehicle make is weird) but now that I know, I will gorup my data based on the type of the car and sum the total number of tikets paid for each of them:

```
## I first check what values exist in the 'ticket_queue' column
print(data_set_1['ticket_queue'].unique())

## So I created a dummy column for whether the
## ticket_queue is "Paid" or not, case insensitive
data_set_1['is_paid'] = data_set_1['ticket_queue'].str.lower() == 'paid'.lower()

grouped_data = data_set_1.groupby('vehicle_make').agg(
    T_tickets=('ticket_number', 'count'),
    tickets_P=('is_paid', 'sum')
).reset_index()

grouped_data['paid_fraction'] = grouped_data['tickets_P'] / grouped_data['T_tickets']
bar_chart = alt.Chart(grouped_data).mark_bar(color="pink").encode(
    x=alt.X('vehicle_make:N', sort='-y', title='Vehicle Make'),
    y=alt.Y('paid_fraction:Q', title='Fraction of Tickets Paid'),
    tooltip=['vehicle_make', 'paid_fraction']
).properties(
    title='Fraction of Tickets Paid by Vehicle Make',
    width=500,
    height=400
).configure_axisX(
    labelAngle=90,
    ## Rotate x-axis labels so it's easier to read
    labelFontSize=5,
    labelLimit=100
)
bar_chart
```

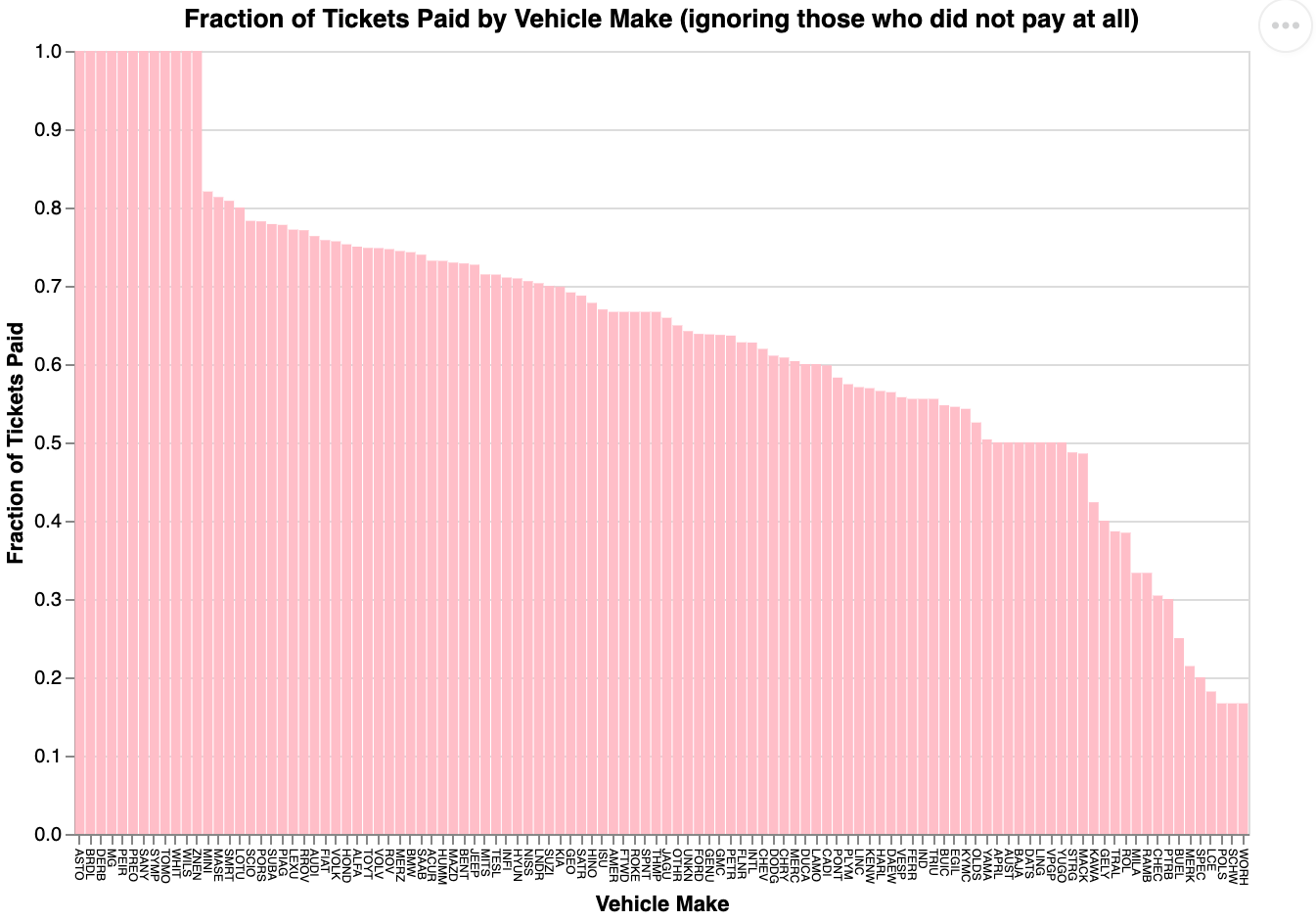
```
['Paid' 'Notice' 'Define' 'Dismissed' 'Bankruptcy' 'Court' 'Hearing Req']
```



This is a little difficult to read so I am going to draw a new one witohut the car brands that have a payment fraction of 0

```
import altair as alt


clean_dataset = grouped_data[grouped_data[
    'paid_fraction'] > 0]

bar_chart_2 = alt.Chart(clean_dataset).mark_bar(
    color="pink").encode(
    x=alt.X('vehicle_make:N', sort='-y',
    title='Vehicle Make'),
    y=alt.Y('paid_fraction:Q',
    title='Fraction of Tickets Paid'),
    tooltip=['vehicle_make', 'paid_fraction']
).properties(
    title=(
        'Fraction of Tickets Paid by Vehicle Make '
        '(ignoring those who did not pay at all)'
    ),
    width=600,
    height=400
).configure_axisX(
    labelAngle=90,
```

```
    labelFontSize=6,
    labelLimit=100
)
bar_chart_2
```

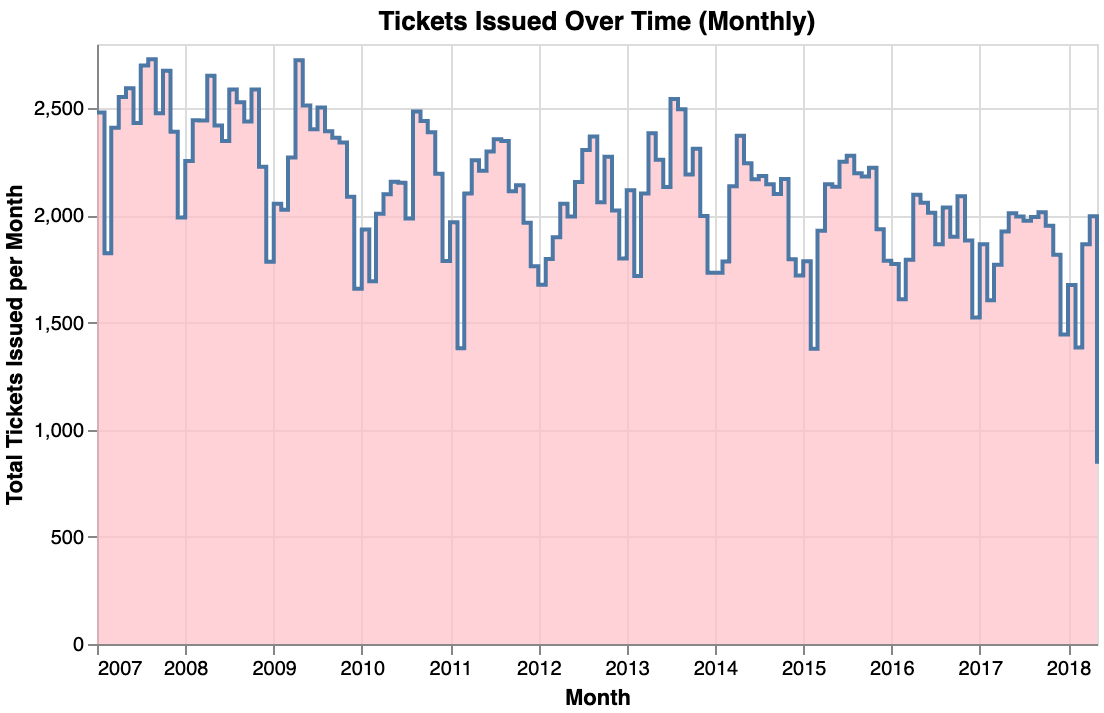


So just like the article also talked about the burden it had on lower income citizens, certian social classes drive certain types of car for example the fraction of tickets paid by vehicle ASTO is 1 (I am guessing it is Aston Martin) which is a luxury high-end car so mostly super rich people own such brands and therefore they all have paid their tickets!

3.

```
##first I am just making sure "issue_date" column is
##time type
data_set_1["issue_date"]= pd.to_datetime(data_set_1["issue_date"])

##now since the data is huge, I will group it by the
##total number of tickets issued every month!
##(I had to ask GPT cause I had no idea how to do it

Monthly_tickets = data_set_1.groupby(data_set_1['issue_date'].dt.to_period('M')).size(
).reset_index(name='ticket_count')
Monthly_tickets['issue_date'] = Monthly_tickets['issue_date'].dt.to_timestamp()

step_chart = alt.Chart(Monthly_tickets).mark_area(
```

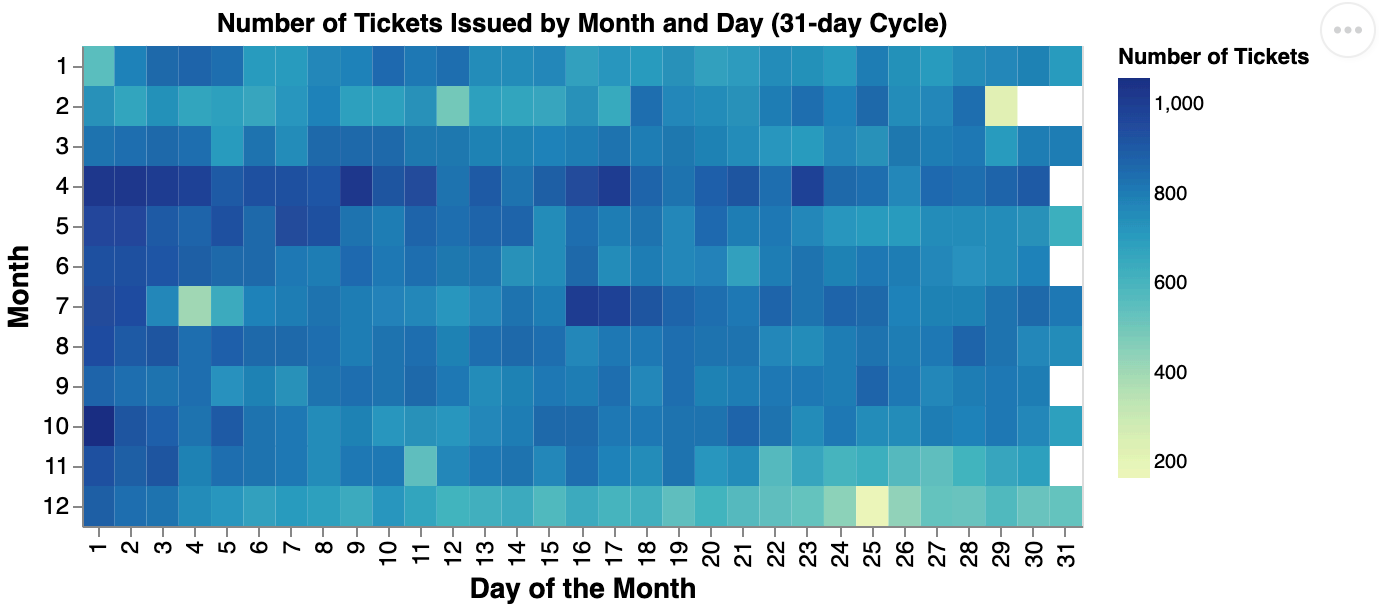
```
    color="pink",
    interpolate='step-after',
    line=True
).encode(
    x=alt.X('issue_date:T', title='Month'),
    y=alt.Y('ticket_count:Q', title='Total Tickets Issued per Month'),
    tooltip=['issue_date', 'ticket_count']
).properties(
    title='Tickets Issued Over Time (Monthly)',
    width= 500,
)

step_chart

# Display the chart
step_chart
```



4.

```
##I am adding these two lines after searching
##for the error I had(Max rows)

data_set_1['issue_date'] = pd.to_datetime(data_set_1['issue_date'])

data_set_1['day'] = data_set_1['issue_date'].dt.day
data_set_1['month'] = data_set_1['issue_date'].dt.month

tickets_by_day_month = data_set_1.groupby(['day', 'month']).size().reset_index(
    name='ticket_count')

heatmap = alt.Chart(tickets_by_day_month).mark_rect().encode(
```

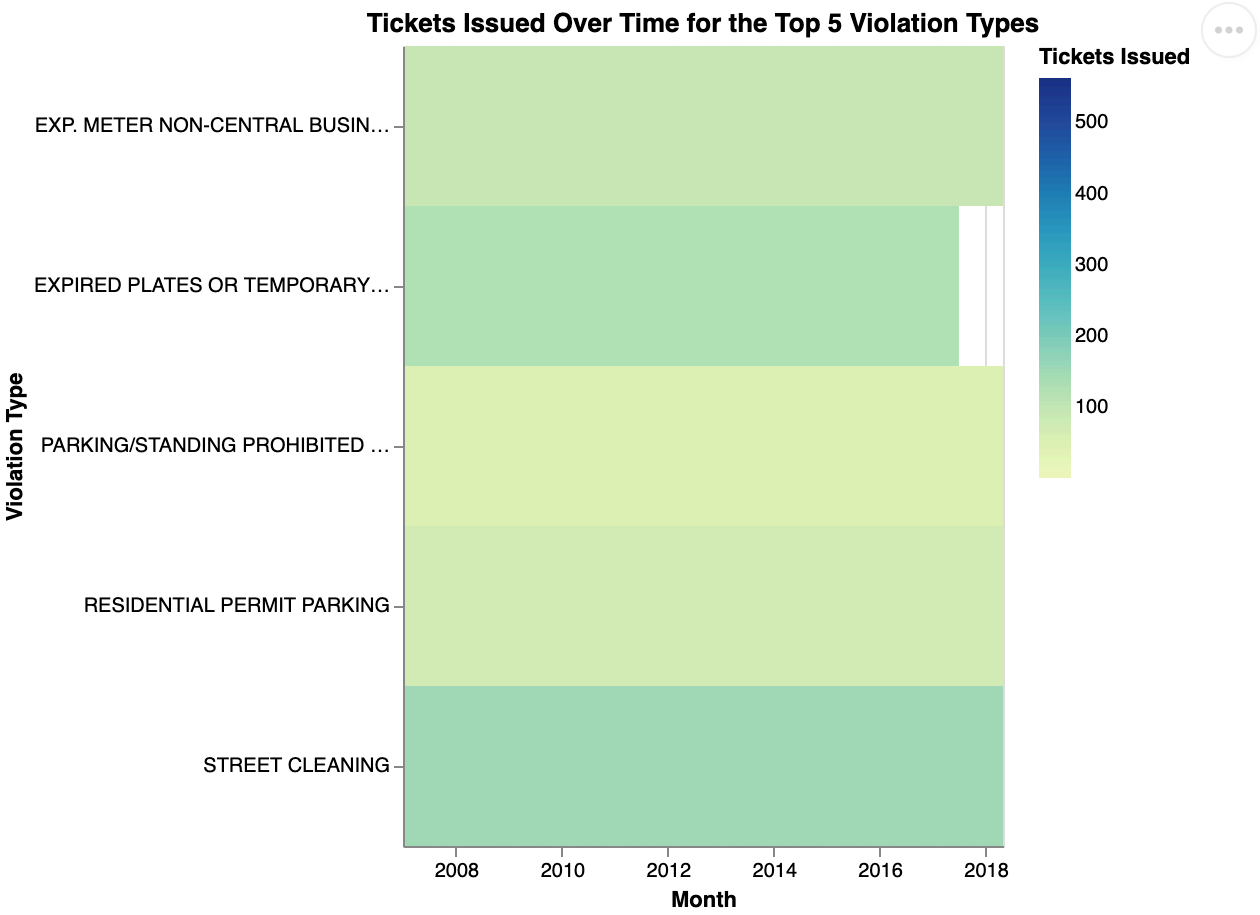
```
    x=alt.X('day:O',
    scale=alt.Scale(domain=list(range(1, 32))), title='Day of the Month'),
    y=alt.Y('month:O', title='Month'),
    color=alt.Color('ticket_count:Q',
    title='Number of Tickets'),
    tooltip=['day', 'month', 'ticket_count']
).properties(
    title='Number of Tickets Issued by Month and Day (31-day Cycle)',
    width= 500,
).configure_axis(
    labelFontSize=12,
    titleFontSize=14
)

heatmap
```



5.

```
data_set_1['issue_date'] = pd.to_datetime(data_set_1['issue_date'])

top_5_violations = data_set_1['violation_description'].value_counts().head(5).index

subset_data = data_set_1[data_set_1['violation_description'].isin(top_5_violations)]

subset_data['issue_month'] = subset_data['issue_date'].dt.to_period('M')

tickets_over_time = subset_data.groupby(['issue_month', 'violation_description']
).size().reset_index(name='ticket_count')

tickets_over_time['issue_month'] = tickets_over_time['issue_month'].dt.to_timestamp()

lasagna_plot = alt.Chart(tickets_over_time).mark_rect().encode(
    x=alt.X('issue_month:T', title='Month'),
    y=alt.Y('violation_description:N', title='Violation Type'),
```

```
    color=alt.Color('ticket_count:Q', title='Tickets Issued'),
).properties(
    title='Tickets Issued Over Time for the Top 5 Violation Types',
    width=300,
    height=400
)
lasagna_plot
```



6. Bar plot: the first thing is that bar plots are easiest to understand and read, they give some general informaiton but the problem is that it is limited, like we only mentioned 20 violaitons but imagine having more(it would have been either a huge plot or nonlegible) Heatmap: I perosnally really like heatmaps for actual climate related but in our case, it shows the intensity of tickets issued per day each month but first you cannot gwet all 11 years, second it does not give us specificaiton of ciolations and other details. it can also be very overwhelming if we have such a large dataset! (altohugh knowing that certain days have higher issuance rate to others it owuldnt really give us the cause like why did that happen? (violation type)) Lasagna map: I owuld say this is a combination of bar plots and heatmaps where you can see the number of tickets per month and the violation tied to those issuances, but it is not that delicate(it only shows very large bins of different number of tickets issued)
7. As I mentioned in the previous question since Lasagna plot is somehow the combinaiton of barplot and heatmap I would say that is our best bet since it is giving info both on violaiton type and also the date.