30538 Problem Set 2: Parking Tickets

AUTHOR
Yusn Qi

PUBLISHED
October 19, 2024

1. “This submission is my work alone and complies with the 30538 integrity policy.” Add your initials to indicate your agreement: **_YQ_**
2. “I have uploaded the names of anyone I worked with on the problem set **here**” **_Yes_** (1 point)
3. Late coins used this pset: **_0_* *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

```
import pandas as pd
import altair as alt
alt.renderers.enable("png")
import time
import warnings
warnings.filterwarnings('ignore')
import numpy as np
```

Data cleaning continued (15 points)

1.

```
# Load the dataset
file_path = r"C:\Users\freya\Desktop\24 fall study\Python2\Pset2\parking_tickets_one_percent.c
pt = pd.read_csv(file_path)

# Set up a function to count the number of NA rows for each column
def count_na(df):
  na_count = df.isna().sum()
  # Convert a dictionary to a dataframe
  result_df = pd.DataFrame({
    "Variable": na_count.index,
    "Count of NA": na_count.values
  })
  return result_df

# Check the numbers of NA row in the parking ticket dataset
print(count_na(pt))
```

```
                 Variable  Count of NA
0             Unnamed: 0            0
```

```
1           ticket_number            0
2              issue_date            0
3      violation_location            0
4    license_plate_number            0
5     license_plate_state           97
6      license_plate_type         2054
7                 zipcode        54115
8          violation_code            0
9   violation_description            0
10                   unit           29
11       unit_description            0
12           vehicle_make            0
13     fine_level1_amount            0
14     fine_level2_amount            0
15     current_amount_due            0
16         total_payments            0
17           ticket_queue            0
18      ticket_queue_date            0
19           notice_level        84068
20    hearing_disposition       259899
21           notice_number            0
22                 officer            0
23                 address            0
```

2.

The most frequently data missing variables are hearing_disposition, notice_level and zipcode.

hearing_disposition reflects the outcome of the hearing. Liable means the defendant has been found responsible for the violation, while Not Liable indicates the defendant successfully presented a defense and was not found in violation. The missing data in this variable is largely due to many tickets were not contested.

notice_level describes the type of notice the city has sent a motorist. A lot of missing data is due to the drivers pay the fine immediately so no notice was sent.

zipz_code is the ZIP code associated with the vehicle registration.The missing zip code accompanies with missing notice level and probably related to the the missing the notice_level.

3.

```
import pandas as pd
# Filter rows where 'violation_description' contains 'NO CITY STICKER'
filtered_violations = pt[pt['violation_description'].str.contains('NO CITY STICKER', case=Fals

#Extract the year of issue date
filtered_violations['year'] = pd.to_datetime(filtered_violations['issue_date']).dt.year

# Calculte the unique violation code by year
violation_code_year = filtered_violations.groupby('year')['violation_code'].unique()

# Print the unique violation_code by year
print(violation_code_year)
```

```
# Calculate the counts of differnt violation code
violation_code_description_counts = filtered_violations.groupby(['violation_code','violation_d

# Print the unique violation_code, corresponding violation_description, and their counts
print(violation_code_description_counts)
```

```
year
2007                [0964125, 0976170]
2008                [0964125, 0976170]
2009                [0964125, 0976170]
2010                [0964125, 0976170]
2011                [0964125, 0976170]
2012    [0964125, 0964125B, 0964125C]
2013              [0964125B, 0964125C]
2014              [0964125B, 0964125C]
2015              [0964125B, 0964125C]
2016              [0964125B, 0964125C]
2017              [0964125B, 0964125C]
2018                         [0964125B]
Name: violation_code, dtype: object
  violation_code                              violation_description  counts
0        0964125                NO CITY STICKER OR IMPROPER DISPLAY   10758
1       0964125B  NO CITY STICKER VEHICLE UNDER/EQUAL TO 16,000 ...   14246
2       0964125C            NO CITY STICKER VEHICLE OVER 16,000 LBS.     131
3        0976170                NO CITY STICKER OR IMPROPER DISPLAY      15
```

The old violation code before 2012 is 0964125 (with 15 counts of 0976170). The new one is 0964125B for vehicle under/equal to 16,000 LBS, and 0964125C for vehicle over 16,000 LBS.

4.

```
# Filter the rows where 'violation_code' equals '0964125' and '0964125B'
violation_964125 = pt[pt['violation_code'] == '0964125']
violation_964125B = pt[pt['violation_code'] == '0964125B']

# Select the 'violation_code' and 'fine_level1_amount' columns and print the first few rows
violation_964125_fines = violation_964125[['violation_code', 'fine_level1_amount']].head()
violation_964125B_fines = violation_964125B[['violation_code', 'fine_level1_amount']].head()

# Print the result
print(violation_964125_fines)
print(violation_964125B_fines)
```

```
    violation_code  fine_level1_amount
14         0964125                 120
29         0964125                 120
82         0964125                 120
97         0964125                 120
104        0964125                 120
       violation_code  fine_level1_amount
138604       0964125B                 200
```

```
138614        0964125B                200
138624        0964125B                200
138625        0964125B                200
138631        0964125B                200
```

The initial fine under the old violation code is $120, and the cost of initial offese under the new violation code is $200.

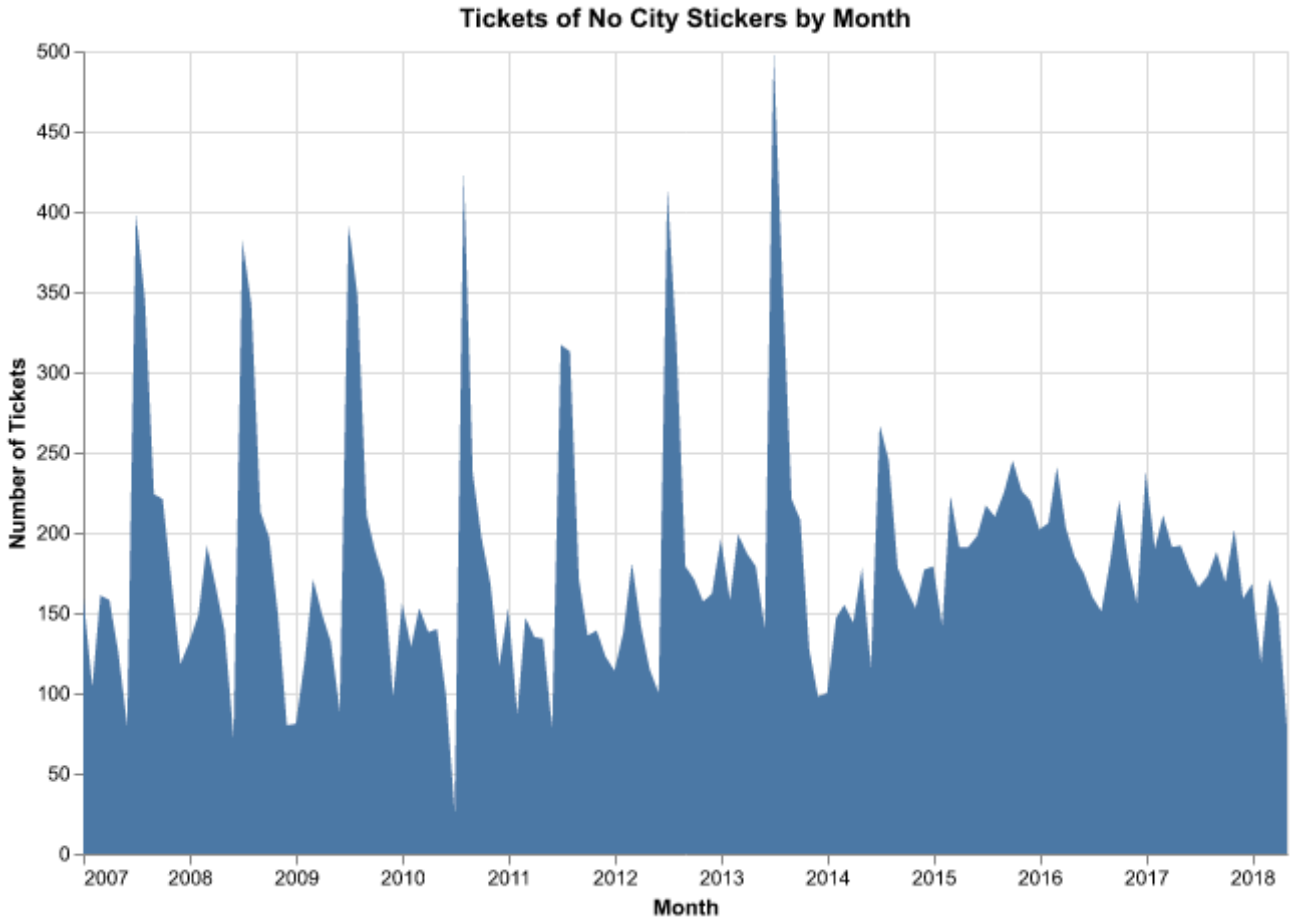
Revenue increase from “missing city sticker” tickets (20 Points)

1.

```
# Convert ticket issue date to datetime format
pt['issue_date'] = pd.to_datetime(pt['issue_date'])

# Create a new column combining both violation codes into a single category for city sticker t
pt['combined_violation_code'] = pt['violation_code'].apply(
    lambda x: 'City Sticker Violation' if x in ['0964125', '0964125B'] else x
)

# Filter for rows where the combined violation code is for city sticker violations
city_sticker_tickets = pt[pt['combined_violation_code'] == 'City Sticker Violation']

# Creat a colomn for the year and the month
city_sticker_tickets['issue_year_month'] = city_sticker_tickets['issue_date'].dt.to_period('M'

# Count the tickets of city stickers by group of months
monthly_tickets = city_sticker_tickets.groupby('issue_year_month').size().reset_index(name = '

# Display the result
print(monthly_tickets)

# Convert the 'issue_year_month' column to string format
monthly_tickets['issue_year_month'] = monthly_tickets['issue_year_month'].astype(str)

# Plot the graph using Altair
chart1 = alt.Chart(monthly_tickets).mark_area().encode(
    x = alt.X('issue_year_month:T',title='Month',axis=alt.Axis(tickCount=20)),
    y = alt.Y('ticket_count:Q',title ='Number of Tickets')
).properties(
    title='Tickets of No City Stickers by Month',
    width=600,
    height=400
)
chart1.show()
chart1.save('chart1.png')
```

```
   issue_year_month  ticket_count
0           2007-01           160
1           2007-02           104
```

```
2            2007-03          161
3            2007-04          158
4            2007-05          126
..               ...          ...
132          2018-01          168
133          2018-02          119
134          2018-03          171
135          2018-04          153
136          2018-05           79

[137 rows x 2 columns]
```

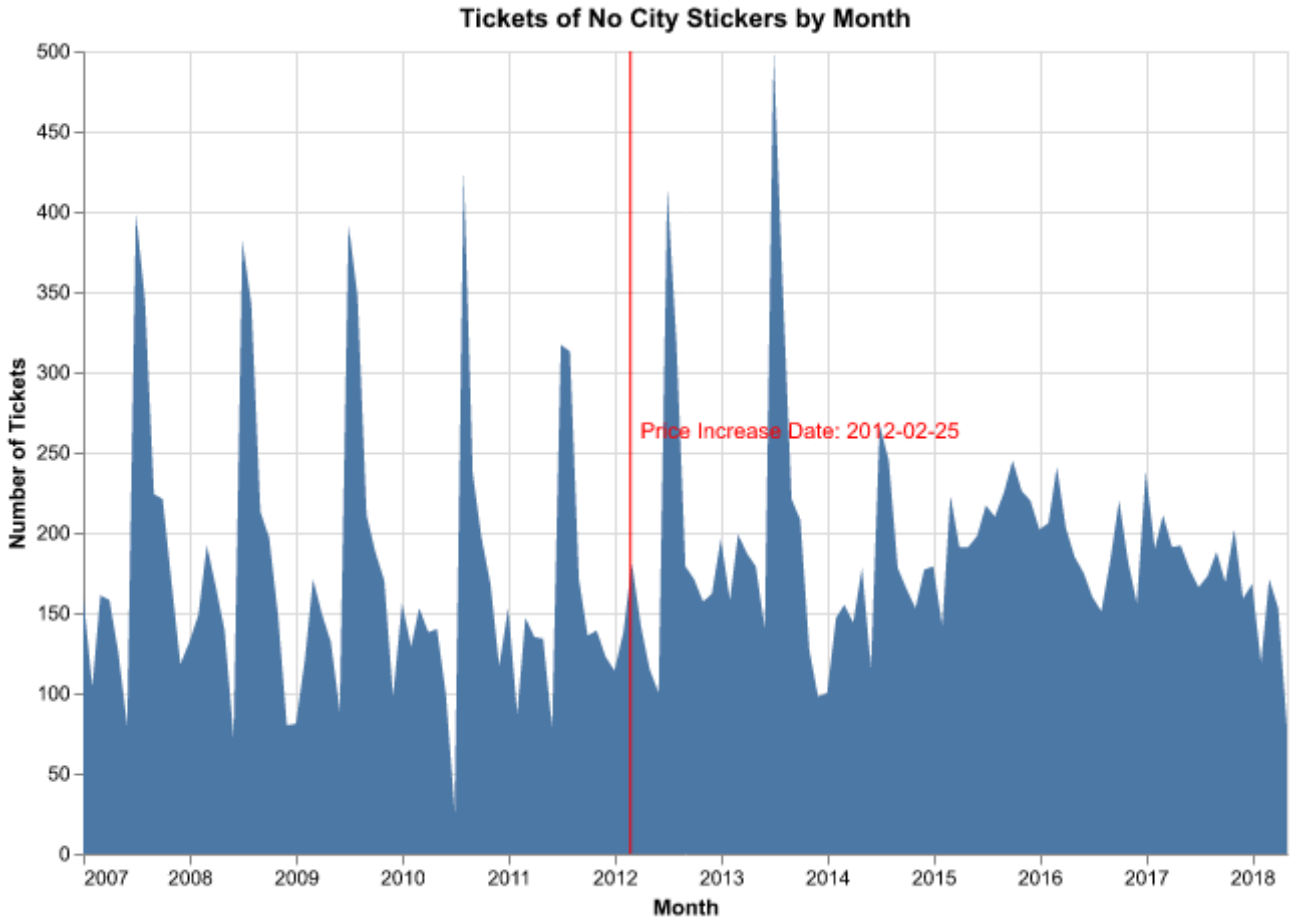


2.

```python
# Filter the tickets data of no city sticker
no_city_sticker_tickets = pt[pt['violation_description'].str.contains('NO CITY STICKER', case=

# Find the fine of $120 and $200
fine_120 = no_city_sticker_tickets[no_city_sticker_tickets['fine_level1_amount'] == 120]
fine_200 = no_city_sticker_tickets[no_city_sticker_tickets['fine_level1_amount'] == 200]

# Find the earlist date when fine changes from 120 to 200
increase_date = fine_200['issue_date'].min()

print(f"The fine amount increased from 120 to 200 on: {increase_date}")
```

```
The fine amount increased from 120 to 200 on: 2012-02-25 02:00:00
```

```
# Add the price increase date
increase_date_str = '2012-02-25'  # The date when the fine increased from 120 to 200

# Original area chart
chart1 = alt.Chart(monthly_tickets).mark_area().encode(
    x=alt.X('issue_year_month:T', title='Month', axis=alt.Axis(tickCount=20)),
    y=alt.Y('ticket_count:Q', title='Number of Tickets')
).properties(
    title='Tickets of No City Stickers by Month',
    width=600,
    height=400
)

# Vertical line at the price increase date
price_increase_line = alt.Chart(pd.DataFrame({'increase_date': [increase_date_str]})).mark_rul
    x='increase_date:T'
)

# Add a text label to show the date
price_increase_label = alt.Chart(pd.DataFrame({
    'increase_date': [increase_date_str],
    'label': ['Price Increase Date: 2012-02-25']
})).mark_text(align='left', dx=5, dy=-10, color='red').encode(
    x='increase_date:T',
    text='label'
)

# Combine the area chart, vertical line, and text label
final_chart = chart1 + price_increase_line + price_increase_label

# Display the final chart with the price increase line and label
final_chart.show()
final_chart.save('chart_with_price_increase_label.png')
```



The page I checked for adjusting the axis labels is here, https://altair-viz.github.io/user_guide/customization.html#axes

3.

```
# Filter the tickets data for 2011 and with violation description 'NO CITY STICKER OR IMPROPER
tickets_issued_2011 = pt[(pt['issue_date'].dt.year == 2011) &
                         (pt['violation_code'] == '0964125')].shape[0]

# Calculate the total revenue increased by the policy change
revenue_increased = tickets_issued_2011*(200-120)*100

print(f'The government projected they will raise the total revunue of ${revenue_increased/1000
```

```
The government projected they will raise the total revunue of $15.464 million anually.
```

4.

```
# Filter the tickets data in 2012
tickets_2013 = pt[pt['issue_date'].dt.year == 2013]

# Calculate the number of tickets issued in 2013
tickets_issued_2013 = tickets_2013[tickets_2013['violation_code'] == '0964125B'].shape[0]

# Calculate the number of tickets paid in 2013
tickets_paid_2013 = tickets_2013[
```

```
    (tickets_2013['violation_code'] == '0964125B') &
    (tickets_2013['ticket_queue'] == 'Paid')
].shape[0]

# Calculate the repayment rate
repayment_rate_2013 = (tickets_paid_2013 / tickets_issued_2013)

# Recalculate the the total revenue increased by the policy change
revenue_increased1 = tickets_issued_2011 * repayment_rate_2013 * (200 - 120) * 100

print(f'The repayment rate in 2013 is {repayment_rate_2013:.3f}.')

print(f'And the government projected they will raise the total revunue of ${revenue_increased1
```

The repayment rate in 2013 is 0.406.
And the government projected they will raise the total revunue of $6.277 million anually.

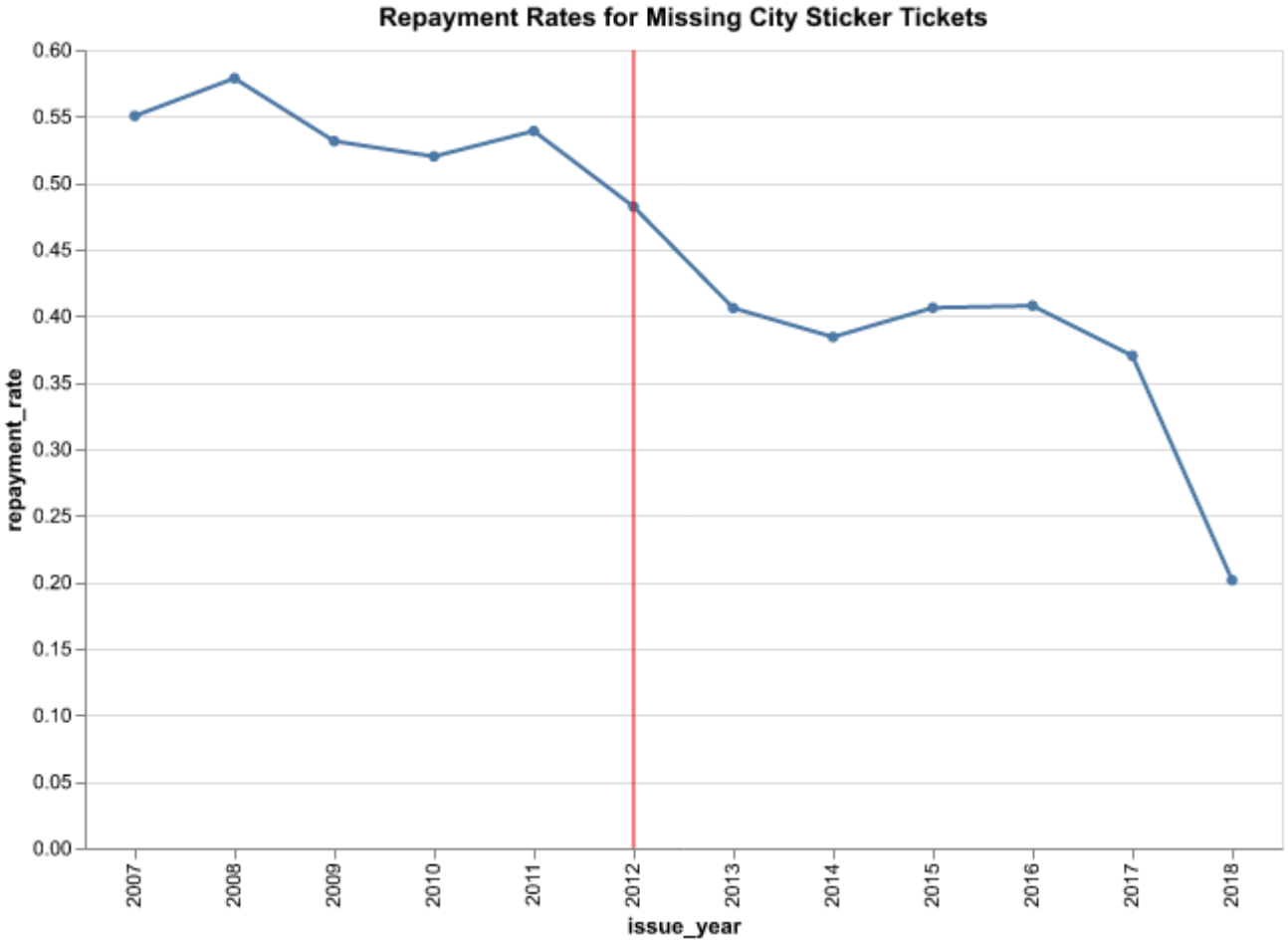
5.

```
# Filter the ticket data for city stickers
city_sticker_tickets = pt[pt['combined_violation_code'] == 'City Sticker Violation']

# Exract the Year of the issue date
city_sticker_tickets['issue_year'] = city_sticker_tickets['issue_date'].dt.year

# Calculate the city sticker tickets issued per year
tickets_issued = city_sticker_tickets.groupby('issue_year').size().reset_index(name = 'tickets_

# Filter for only paid tickets
paid_tickets = city_sticker_tickets[city_sticker_tickets['ticket_queue'] == 'Paid']

# Calculate the city sticker tickets paid per year
tickets_paid = paid_tickets.groupby('issue_year').size().reset_index(name='tickets_paid')

# Print the first few rows to confirm the data
print("Tickets Issued:\n", tickets_issued.head())
print("Tickets Paid:\n", tickets_paid.head())

# Merge issued and paid tickets into a single DataFrame
tickets_summary = pd.merge(tickets_issued, tickets_paid, on='issue_year', how='left')

# Fill NaN values in tickets_paid with 0 (in case some years had no payments)
tickets_summary['tickets_paid'].fillna(0, inplace=True)

# Print the merged DataFrame to confirm column names
print("Tickets Summary After Merge:\n", tickets_summary.head())

# Calculate the repayment per year
tickets_summary['repayment_rate'] = tickets_summary['tickets_paid'] / tickets_summary['tickets_

print(tickets_summary)
```

```
# Plot the graph
chart2 = alt.Chart(tickets_summary).mark_line(point=True).encode(
    x='issue_year:O',
    y='repayment_rate:Q',
    tooltip=['issue_year', 'repayment_rate']
).properties(
    title='Repayment Rates for Missing City Sticker Tickets',
    width=600,
    height=400
)

# Add a vertical line for the policy change date
policy_change_year = 2012
policy_change_line = alt.Chart(pd.DataFrame({'issue_year': [policy_change_year]})).mark_rule(c
    x=alt.X('issue_year:O')
)

# Combine the line chart and the policy change line
final_chart1 = chart2 + policy_change_line

# Show the plot
final_chart1.show()
final_chart1.save('final chart 1.png')
```

```
Tickets Issued:
    issue_year  tickets_issued
0         2007            2264
1         2008            2212
2         2009            2149
3         2010            1985
4         2011            1933
Tickets Paid:
    issue_year  tickets_paid
0         2007          1246
1         2008          1280
2         2009          1142
3         2010          1032
4         2011          1042
Tickets Summary After Merge:
    issue_year  tickets_issued  tickets_paid
0         2007            2264          1246
1         2008            2212          1280
2         2009            2149          1142
3         2010            1985          1032
4         2011            1933          1042
    issue_year  tickets_issued  tickets_paid  repayment_rate
0         2007            2264          1246        0.550353
1         2008            2212          1280        0.578662
2         2009            2149          1142        0.531410
3         2010            1985          1032        0.519899
4         2011            1933          1042        0.539058
5         2012            2192          1057        0.482208
6         2013            2567          1042        0.405921
```

7	2014	2025	778	0.384198
8	2015	2467	1002	0.406161
9	2016	2264	923	0.407686
10	2017	2256	835	0.370124
11	2018	690	139	0.201449

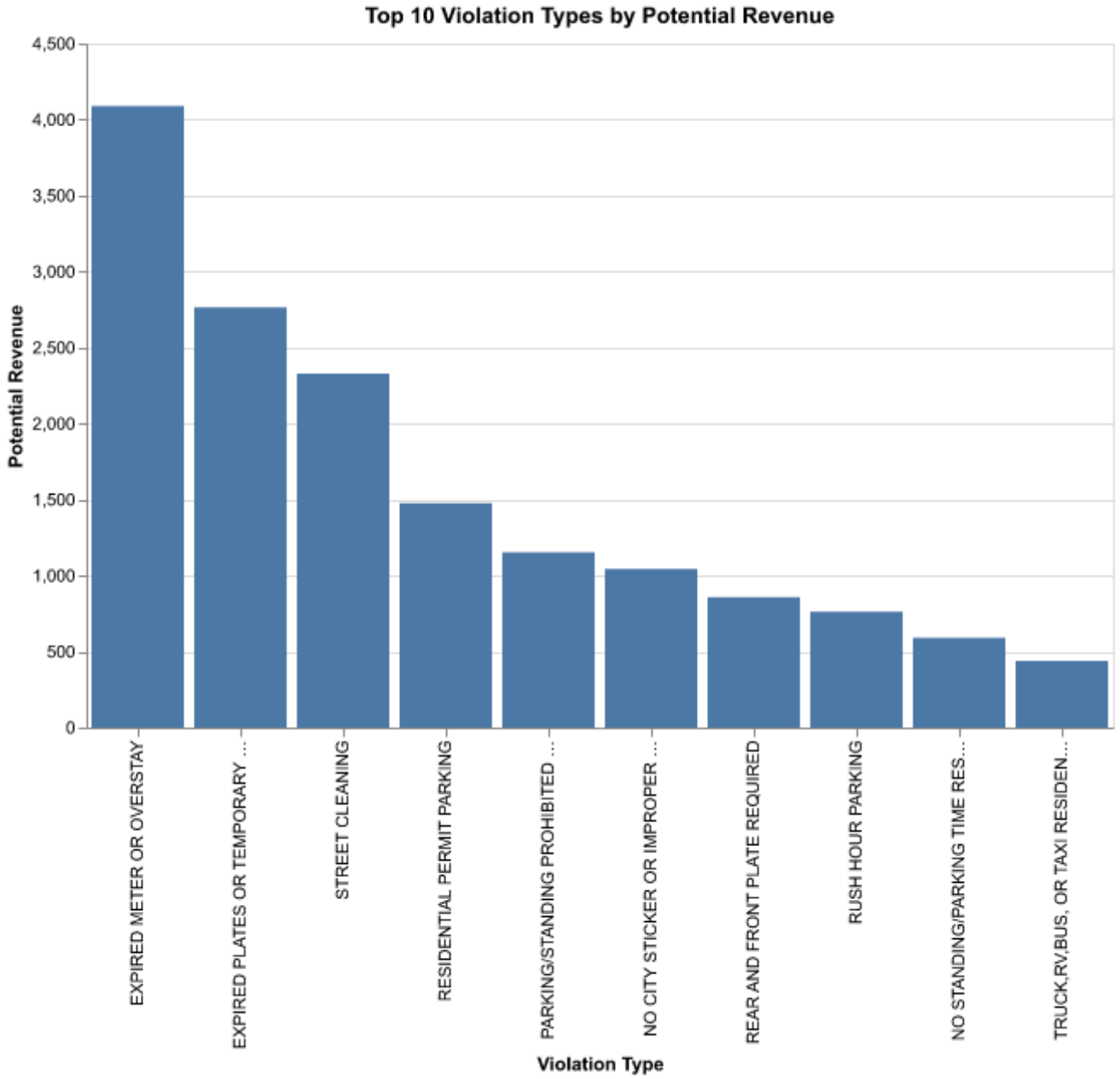


6.

```
# Calculate the number of tickets issued in 2011 by violation type
tickets_issued_2011_by_viotype = pt[pt['issue_date'].dt.year == 2011].groupby('violation_descr

# Calculate the number of tickets paid in 2011 by violation type
tickets_paid_2011_by_viotype = pt[(pt['issue_date'].dt.year == 2011) & (pt['ticket_queue'] ==

# Merge the two dataframes on 'violation_description'
violation_summary = pd.merge(tickets_issued_2011_by_viotype, tickets_paid_2011_by_viotype, on=

# Fill missing values for unpaid violations
violation_summary['tickets_issued_2011'] = violation_summary['tickets_issued_2011'].fillna(0)
violation_summary['tickets_paid_2011'] = violation_summary['tickets_paid_2011'].fillna(0)

# Calculate the repayment rate for 2011 by violation type
violation_summary['repayment_rate_2011'] = (violation_summary['tickets_paid_2011'] / violation

# Calculate potential revenue based on issued tickets and repayment rate
violation_summary['potential_revenue'] = violation_summary['tickets_issued_2011'] * violation_
```

```
# Sort by revenue potential and select the top 3 violation types
top_violations = violation_summary.sort_values(by='potential_revenue', ascending=False).head(3

# Print the columns and top violations
print(violation_summary.columns)
print(top_violations)

# Sort by potential revenue and take the top 10 violation types
top_violations = violation_summary.sort_values(by='potential_revenue', ascending=False).head(1(

# Create the bar chart for the top 10 violation types by potential revenue
chart3 = alt.Chart(top_violations).mark_bar().encode(
    x=alt.X('violation_description:O', title='Violation Type', sort='-y'),
    y=alt.Y('potential_revenue:Q', title='Potential Revenue'),
    tooltip=['violation_description', 'potential_revenue']
).properties(
    title='Top 10 Violation Types by Potential Revenue',
    width=600,
    height=400
)
chart3.show()
chart3.save('chart3.png')
```

```
Index(['violation_description', 'tickets_issued_2011', 'tickets_paid_2011',
       'repayment_rate_2011', 'potential_revenue'],
      dtype='object')
                     violation_description  tickets_issued_2011  \
9                  EXPIRED METER OR OVERSTAY                 4967
10  EXPIRED PLATES OR TEMPORARY REGISTRATION                 4345
65                          STREET CLEANING                 2810

    tickets_paid_2011  repayment_rate_2011  potential_revenue
9              4088.0             0.823032             4088.0
10             2764.0             0.636133             2764.0
65             2330.0             0.829181             2330.0
```



I would recommend that the City Clerk consider increasing the fines for EXPIRED METER OR OVERSTAY, EXPIRED PLATES OR TEMPORARY REGISTRATION, and STREET CLEANING violations. Based on my analysis of potential revenue, without factoring in price changes, I calculated the revenue by multiplying the number of tickets issued by the repayment rate using 2011 data. These three violations ranked highest when considering both the number of tickets issued and the repayment rate.

Headlines and sub-messages (20 points)

1.

```
# Calculate the tickets issued and tickets paid for each violation type
violation_summary2 = pt.groupby('violation_description').agg(
    tickets_issued = ('ticket_number','count'),
    fraction_paid = ('ticket_queue',lambda x: (x == 'Paid').mean()),
    fine_level_1 =('fine_level1_amount','mean')
```

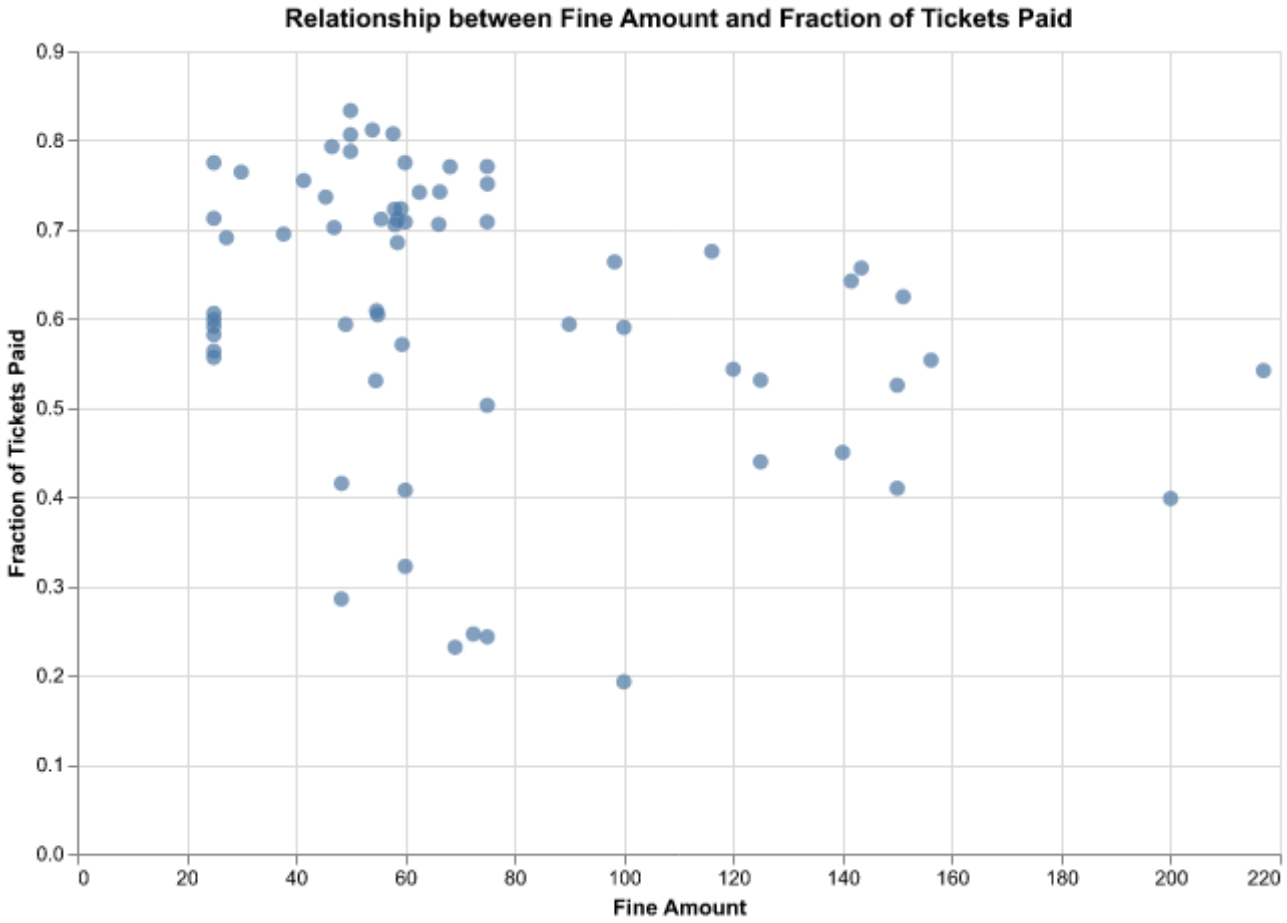
```
).reset_index().sort_values(by='tickets_issued', ascending=False)
print(violation_summary2.head(5))
```

```
                      violation_description  tickets_issued  fraction_paid  \
23   EXPIRED PLATES OR TEMPORARY REGISTRATION           44811       0.604361
101                          STREET CLEANING           28712       0.811612
90                RESIDENTIAL PERMIT PARKING           23683       0.742262
19   EXP. METER NON-CENTRAL BUSINESS DISTRICT           20600       0.792913
81        PARKING/STANDING PROHIBITED ANYTIME           19753       0.705817

     fine_level_1
23      54.968869
101     54.004249
90      66.338302
19      46.598058
81      66.142864
```

2.

```
# Filter for violation types that appear at least 100 times
violation_summary3 = violation_summary2[violation_summary2['tickets_issued'] >= 100]

# Exclude the outlier with a high fine
violation_summary3_filtered = violation_summary3[violation_summary3['fine_level_1'] <= 300]
```

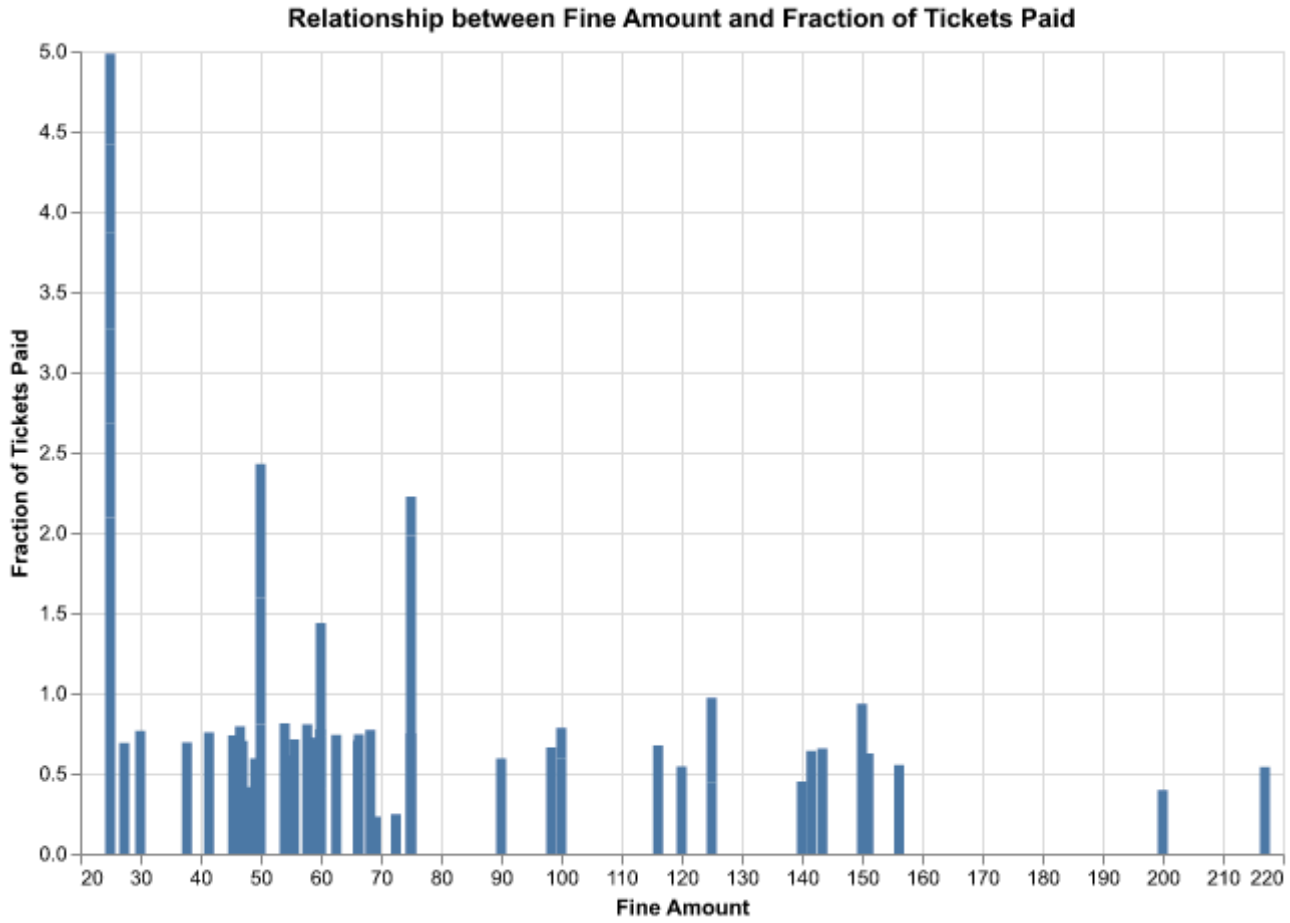
```
# Scatter plot of fine amount vs. fraction of tickets paid
chart4_1 = alt.Chart(violation_summary3_filtered).mark_circle(size=60).encode(
    x=alt.X('fine_level_1:Q', title='Fine Amount'),
    y=alt.Y('fraction_paid:Q', title='Fraction of Tickets Paid'),
    tooltip=['violation_description', 'fine_level_1', 'fraction_paid']
).properties(
    title='Relationship between Fine Amount and Fraction of Tickets Paid',
    width=600,
    height=400
)
chart4_1.show()
chart4_1.save('chart4_1.png')
```



Headlines: The scatter is more densely distributed when the fine amount is below $80, with sparser distribution for fines above $80. Overall, lower fines are associated with a higher fraction of tickets paid.

Sub-messages: Variance is smaller when the fine is below $80, and increases significantly for fines above $80.

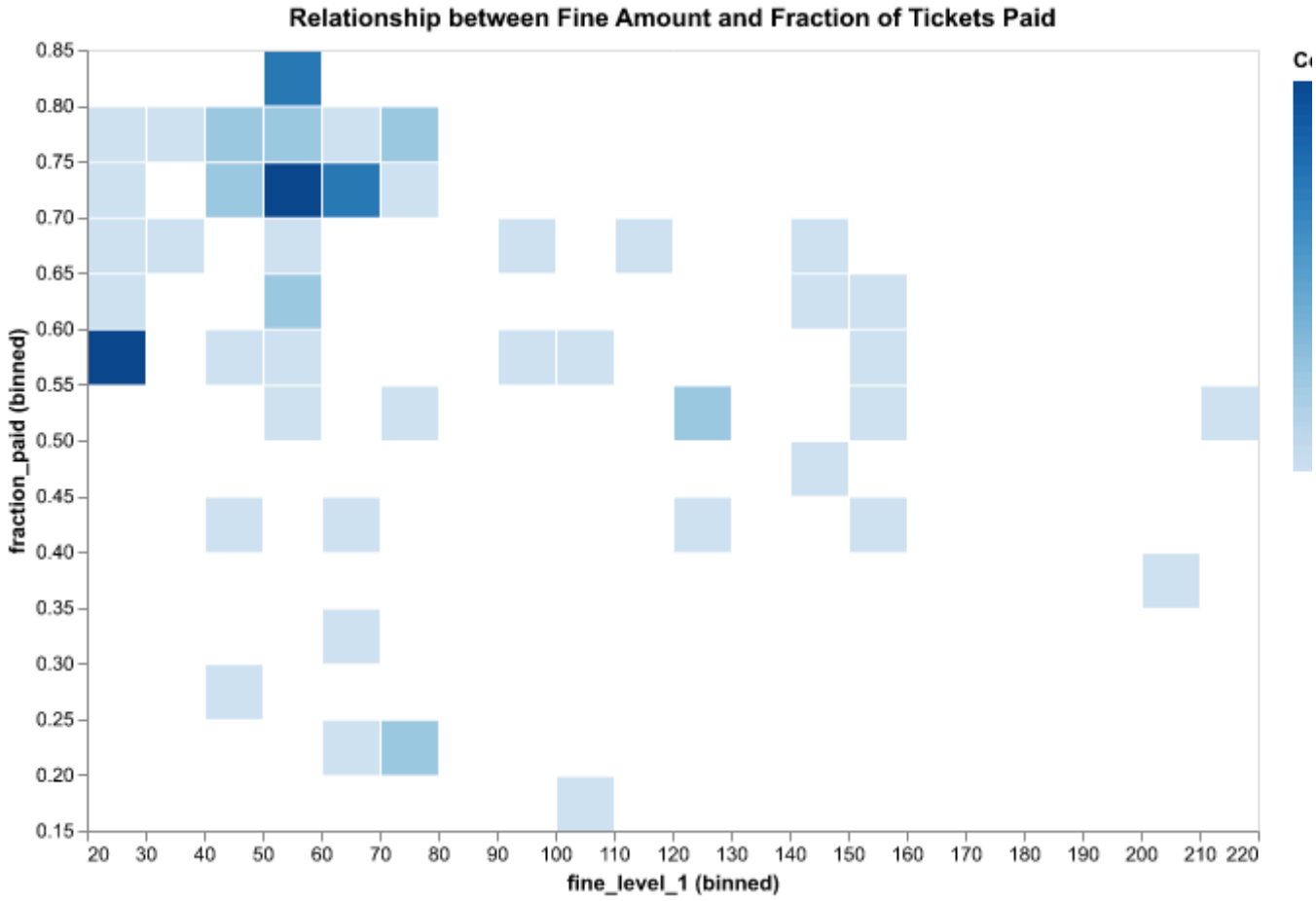
```
# Bar chart of fine amounts vs. fraction of tickets paid
chart4_2 = alt.Chart(violation_summary3_filtered).mark_bar().encode(
    x=alt.X('fine_level_1:Q', title='Fine Amount'),
    y=alt.Y('fraction_paid:Q', title='Fraction of Tickets Paid'),
    tooltip=['violation_description','fine_level_1','fraction_paid']
).properties(
    title='Relationship between Fine Amount and Fraction of Tickets Paid',
    width=600,
    height=400
)
chart4_2.show()
chart4_2.save('chart4_2.png')
```



Headlines: There is a negative correlation between the fine amount and the fraction of tickets paid: as the fine decreases, the fraction of tickets paid increases.

Sub-messages: The fraction of tickets paid is not evenly distributed across fine amounts. In the ranges of \$80 to \$90 and \$160 to \$200, no tickets were paid.

```
# Bin and color plot  of fine amounts vs. fractions of ticket paid
chart4_3 = alt.Chart(violation_summary3_filtered).mark_bar().encode(
    alt.X('fine_level_1:Q', bin=alt.BinParams(maxbins=20)),
    alt.Y('fraction_paid:Q', bin=alt.BinParams(maxbins=20)),
    alt.Color('count()')
).properties(
    title='Relationship between Fine Amount and Fraction of Tickets Paid',
    width=600,
    height=400
)
chart4_3.show()
chart4_3.save('chart4_2.png')
```



Headlines: The scatter is more densely distributed when the fine amount is below $80, with sparser distribution for fines above $80. Overall, lower fines are associated with a higher fraction of tickets paid.

Sub-messages: There are some extremely dense clusters of violation types, such as when the fine amount is between $20 and $30 with a fraction of 0.55 to 0.6, and when the fine is between $50 and $60 with a fraction of 0.7 to 0.75.

3.

I would choose the bar chart of fine amount and fraction of tickets paid. This plot clearly demonstrates the negative correlation between fine amount and the fraction of tickets paid, more effectively than the other two plots. It shows that higher fines lead to a lower average fraction of tickets paid, providing insights similar to those a regression analysis would reveal.

Understanding the structure of the data and summarizing it (Lecture 5, 20 Points)

1.

```
# Filter the data of unpaid fine
violation_summary4 = pt[pt['ticket_queue'] != 'Paid']

# Creat a new colum fine_double to count if the unpaid fine is doubled
```

```
violation_summary4['fine_double'] = np.where(violation_summary4['fine_level2_amount'] / violat

# Filter the violation types where at least 100 citations not doubled
violation_summary4_filtered = violation_summary4[violation_summary4['fine_double'] == 0] \
                              .groupby('violation_description') \
                              .filter(lambda x: len(x) >= 100)

# Check if all violation fine double if unpaid
all_double = violation_summary4['fine_double'].all()

# Summarize the number of not doubled fine citations by each violation type
violation_count = violation_summary4_filtered.groupby('violation_description').size().reset_in

# Calculate the increase of unpaid for each ticket
violation_summary4_filtered['fine_increase'] = violation_summary4_filtered['fine_level2_amount

# Print all the results
print("If all violations double in price: ", all_double)
print(violation_count)
print(violation_summary4_filtered[['violation_description', 'fine_level2_amount', 'fine_level1
```

```
If all violations double in price:  False
                  violation_description  not_double_count
0   BLOCK ACCESS/ALLEY/DRIVEWAY/FIRELANE               180
1               DISABLED PARKING ZONE               431
2                 PARK OR BLOCK ALLEY               799
3  SMOKED/TINTED WINDOWS PARKED/STANDING              306
                      violation_description  fine_level2_amount  \
67                      PARK OR BLOCK ALLEY                 250
200                    DISABLED PARKING ZONE                250
430                     PARK OR BLOCK ALLEY                 250
744                    DISABLED PARKING ZONE                250
805                    DISABLED PARKING ZONE                250
...                                      ...                 ...
138358  SMOKED/TINTED WINDOWS PARKED/STANDING                250
138363  SMOKED/TINTED WINDOWS PARKED/STANDING                250
138364                 DISABLED PARKING ZONE                 250
138464                  PARK OR BLOCK ALLEY                  250
138465                  PARK OR BLOCK ALLEY                  250

        fine_level1_amount  fine_increase
67                     150            100
200                    200             50
430                    150            100
744                    200             50
805                    200             50
...                    ...            ...
138358                 250              0
138363                 250              0
138364                 200             50
138464                 150            100
138465                 150            100
```

```
[1716 rows x 4 columns]
```

2.

```
# Check which unique values are in the variable ticket_queue
unique_ticket_queue_values = pt['ticket_queue'].unique()
print(unique_ticket_queue_values)
```

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

The expalnation of different notice levels: ***Initial Violation (VIOL):*** The first stage.

Detriment (DETR): After a warning, moving to a more serious stage.

Seizure (SEIZ): Vehicle seizure due to non-payment.

Final Notice (FINL): The last notice before collections.

Delinquent (DLS): Represents overdue fines or final stages.

NA: The notice has been resloved

```
# Check which unique values are in the variable ticket_queue
unique_ticket_queue_values = pt['ticket_queue'].unique()
print(unique_ticket_queue_values)
```

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

The Explanation of these different values: ***Paid:*** The ticket has been paid, and the case is resolved.

Notice: The violator has been notified of the ticket, awaiting payment or further action. This is the initial state of the ticket.

Define: Further clarification or action is required, possibly waiting for more information or processing steps.

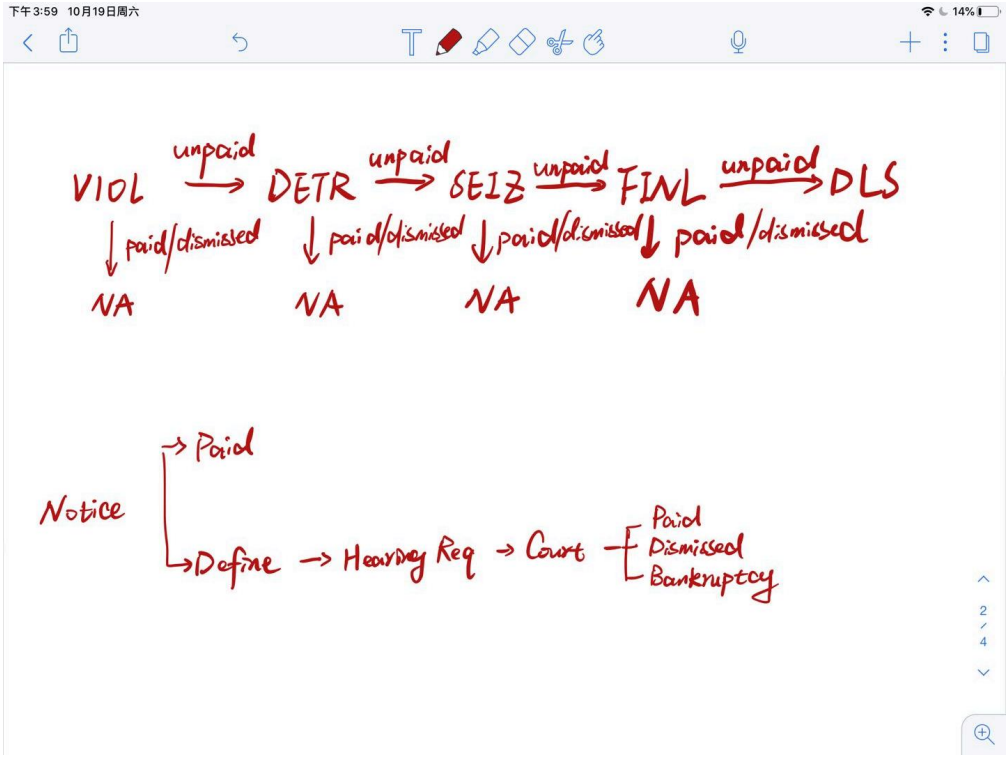
Dismissed: The ticket has been dismissed, and the case is closed. Typically, this occurs when the violator is found not liable.

Bankruptcy: The violator has declared bankruptcy, which may affect how the ticket is processed or collected.

Court: The case has escalated to court for legal action and resolution.

Hearing Req: The violator has requested a hearing to dispute the ticket, initiating a legal process.

So the diagrams of the process of moving between different notice levels and ticket queue levels are as below,

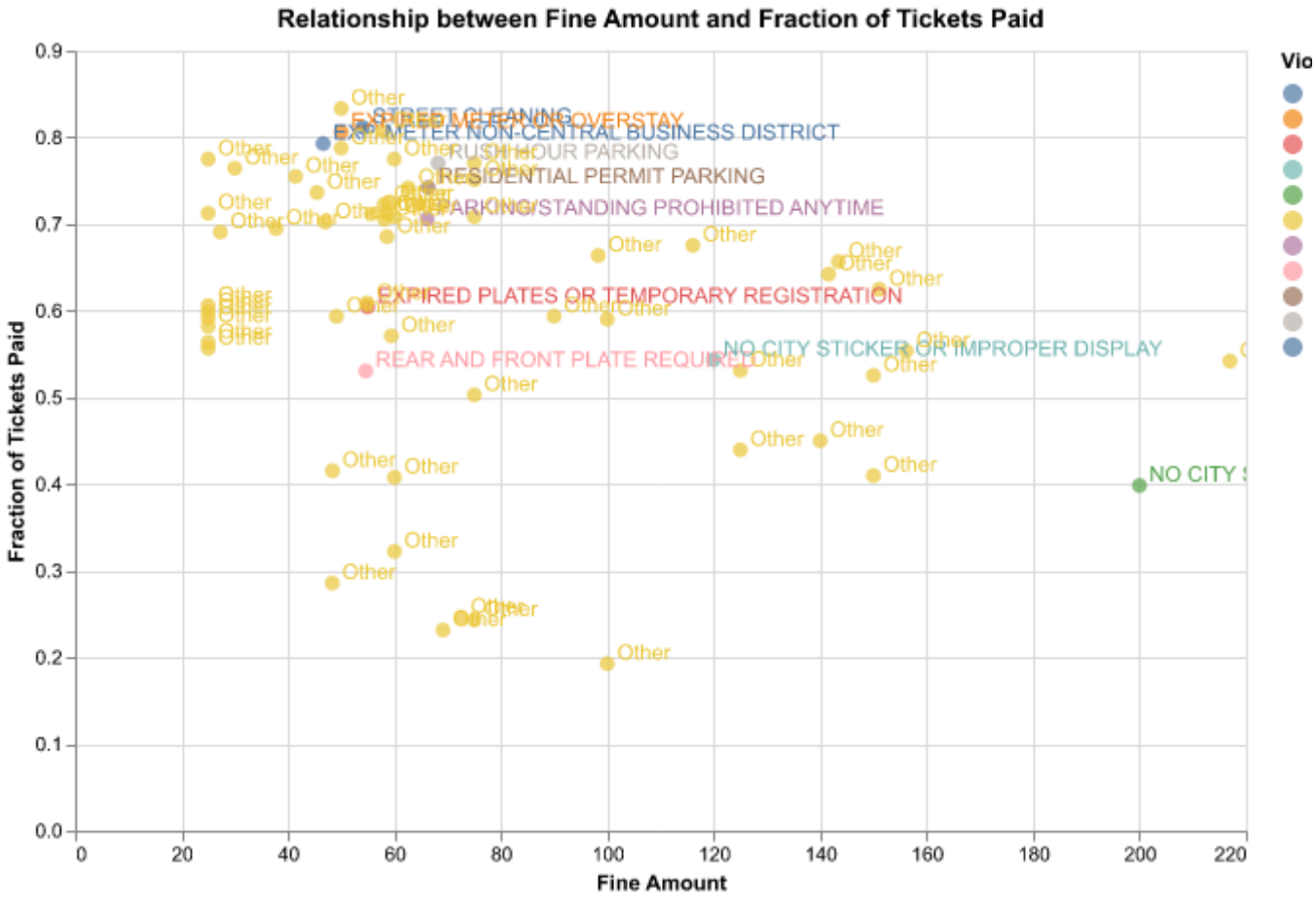


3.

```
# Get the top 10 most common violation descriptions
top_10_violation_descriptions = (
    violation_summary3_filtered['violation_description']
    .value_counts()
    .head(10)
    .index.tolist()
)

# Add a column to mark whether the violation is in the top 10, otherwise label as "Other"
violation_summary3_filtered['violation_group'] = violation_summary3_filtered['violation_descri
    lambda x: x if x in top_10_violation_descriptions else 'Other'
)
```

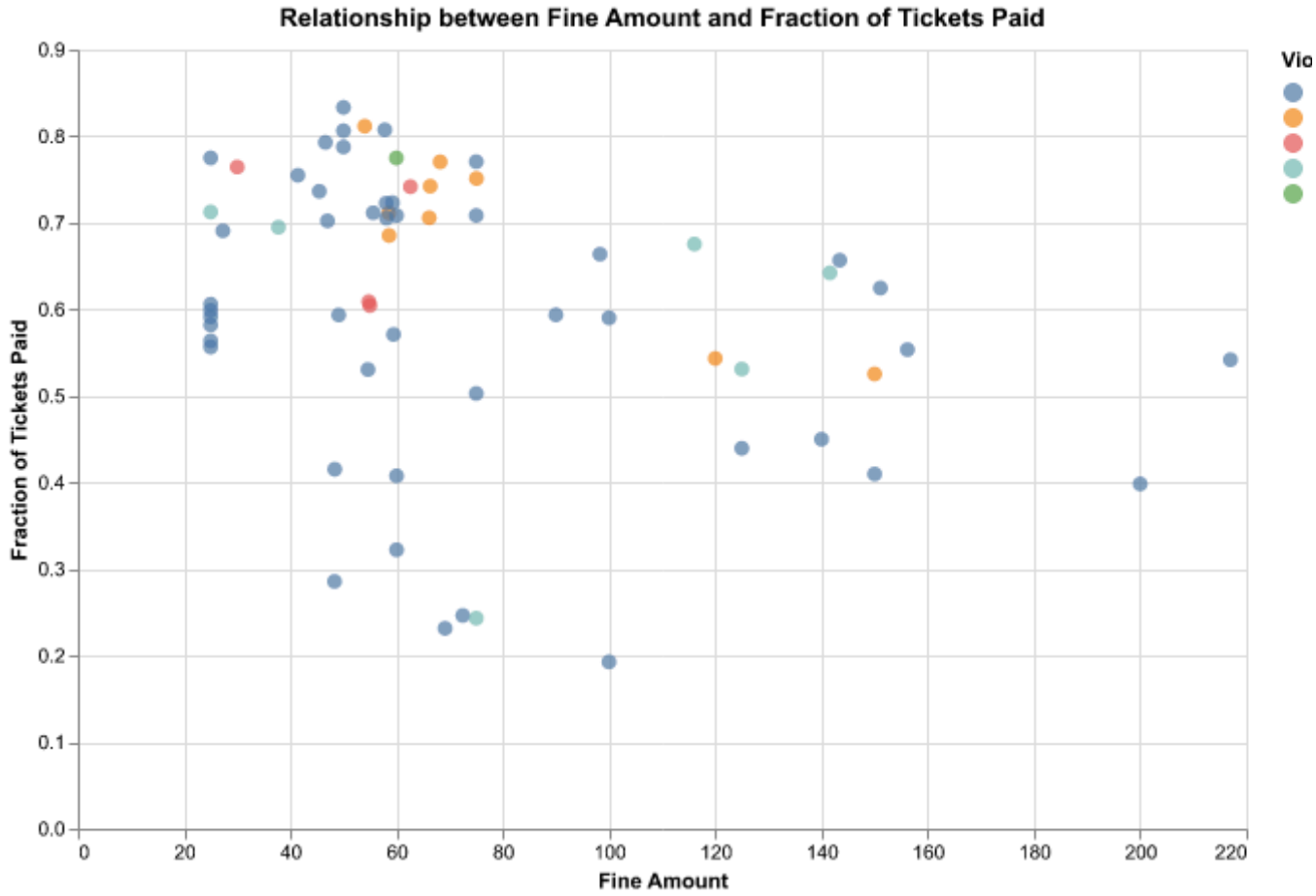
```
# Plot the scatter plot and implement both methods
## Method 1: Label every dot and mark other categories as "Other"
chart5 = alt.Chart(violation_summary3_filtered).mark_circle(size=60).encode(
    x=alt.X('fine_level_1:Q', title='Fine Amount'),
    y=alt.Y('fraction_paid:Q', title='Fraction of Tickets Paid'),
    color=alt.Color('violation_group:N', title='Violation Description'),
    tooltip=['violation_description', 'fine_level_1', 'fraction_paid']
).properties(
    title='Relationship between Fine Amount and Fraction of Tickets Paid',
    width=600,
    height=400
).interactive()

# Add text label adjacent
text = chart5.mark_text(align = 'left', dx = 5, dy = -5).encode(
    text = 'violation_group'
```

```
)

# Combine the label and the chart
chart5_final = chart5 + text
chart5_final.show()
chart5_final.save('chart5_final.png')
```




```
## Method 2: Construct categories by marking violation descriptions which sound similar with a

# Create a dictionary to map each violation description to a category
violation_category_mapping = {
    'RESIDENTIAL PERMIT PARKING': 'Parking',
    'PARKING/STANDING PROHIBITED ANYTIME': 'Parking',
    'NO STANDING/PARKING TIME RESTRICTED': 'Parking',
    'PARK OR BLOCK ALLEY': 'Parking',
    'RUSH HOUR PARKING': 'Parking',
    'STREET CLEANING': 'Parking',
    'PARK OR STAND ON SIDEWALK': 'Parking',
    'NO CITY STICKER OR IMPROPER DISPLAY': 'Parking',
    'PARK OR STAND ON CROSSWALK': 'Parking',
    'EXPIRED PLATES OR TEMPORARY REGISTRATION': 'Registration',
    'EXPIRED METER CENTRAL BUSINESS DISTRICT': 'Registration',
    'EXPIRED METER NON-CENTRAL BUSINESS DISTRICT': 'Registration',
    'NONCOMPLIANT PLATE(S)': 'Registration',
    'IMPROPER DISPLAY OF CITY STICKER': 'Registration',
    'WITHIN 15\' OF FIRE HYDRANT': 'Safety',
```

```
    'TRUCK,RV,BUS, OR TAXI RESIDENTIAL STREET': 'Safety',
    'TRUCK OR SEMI-TRAILER PROHIBITED': 'Safety',
    'BLOCK ACCESS/ALLEY/DRIVEWAY/FIRELANE': 'Safety',
    'SAFETY BELTS REQUIRED': 'Safety',
    'ABANDONED VEHICLE FOR 7 DAYS OR INOPERABLE': 'Safety',
    'HAZARDOUS DILAPIDATED VEHICLE': 'Safety',
    'SPECIAL EVENTS RESTRICTION': 'Special Event',
    'PARK/STAND IN WRIGLEY BUS PERMIT ZONE': 'Special Event',
    'THEATER ENTRANCE/EXIT': 'Special Event',
    # Remaining as 'Other'
}

# Apply the mapping to the dataset
violation_summary3_filtered['violation_category'] = violation_summary3_filtered['violation_des

# Create the scatter plot
chart6 = alt.Chart(violation_summary3_filtered).mark_circle(size=60).encode(
    x=alt.X('fine_level_1:Q', title='Fine Amount'),
    y=alt.Y('fraction_paid:Q', title='Fraction of Tickets Paid'),
    color=alt.Color('violation_category:N', title='Violation Category'),
    tooltip=['violation_description', 'fine_level_1', 'fraction_paid']
).properties(
    title='Relationship between Fine Amount and Fraction of Tickets Paid',
    width=600,
    height=400
)
chart6
```



Extra Credit (max 5 points)

1.

```
# Group by violation_code and violation_description, and count occurrences
violation_grouped = pt.groupby(['violation_code', 'violation_description']).size().reset_index
print(violation_grouped)

# Identify violation codes associated with multiple descriptions
multiple_descriptions = violation_grouped.groupby('violation_code').size().reset_index(name='d
multiple_descriptions = multiple_descriptions[multiple_descriptions['description_count'] > 1]
print(multiple_descriptions)
```

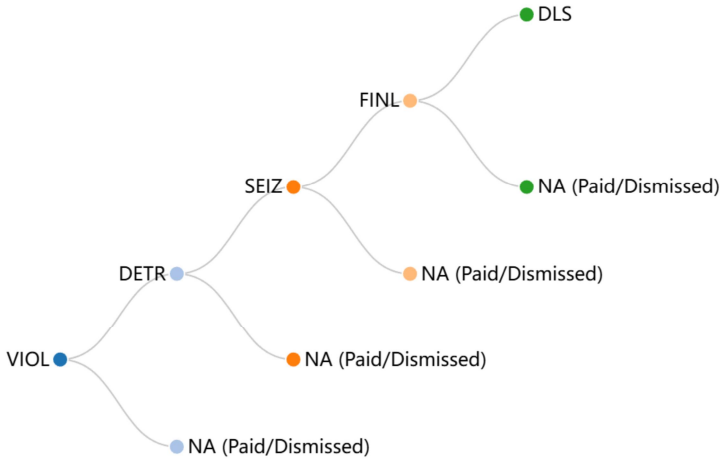
```
    violation_code                    violation_description  counts
0          0912060     STAND, PARK, OR OTHER USE OF BUS LANE    1233
1          0940060                PARK/STAND ON BICYCLE PATH     236
2          0940080  PARKED/STANDING UNATTENDED W/MOTOR RUNNI      81
3          0940170                           UNSAFE CONDITION      64
4          0940220                      NO OPERATOR SIGNAL      32
..             ...                                     ...     ...
119       0980120A                   NO PARK IN PUBLIC LOT     119
120       0980120B                  NO PARK IN PRIVATE LOT     378
121       0980130A  FAIL TO PAY OR OUTSIDE SPACE IN CITY LOT      90
```

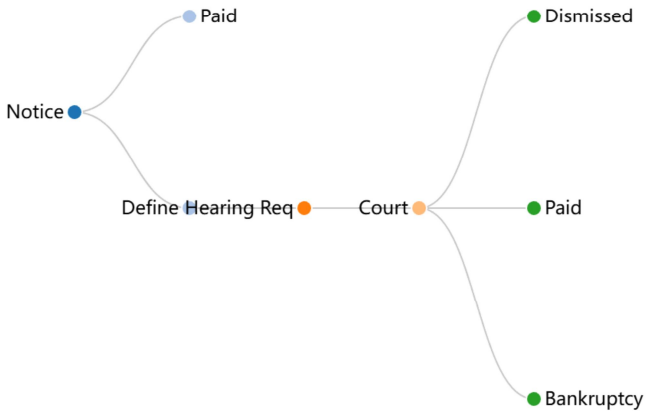
```
122       0980130B              PARK IN CITY LOT WHEN CLOSED     225
123       0980130C             PARK IN CITY LOT OVER 30 DAYS       9

[124 rows x 3 columns]
    violation_code  description_count
9         0964040B                  2
11        0964041B                  2
17         0964070                  2
57        0964170D                  2
63        0964200B                  2
90        0976160A                  2
91        0976160B                  2
110       0980110B                  2
```

```
# Group by violation_code and get the most frequent violation description
most_common_description = violation_grouped.sort_values(['violation_code', 'counts'], ascendin

# Get the most common description for each violation code
most_common_description = most_common_description.groupby('violation_code').first().reset_inde

# Merge the most common descriptions back into the original dataframe
pt = pt.merge(most_common_description[['violation_code', 'violation_description']], on='violat

# Find the top three codes with the most observations
top_three_codes = most_common_description.nlargest(3, 'counts')[['violation_code', 'counts']]
print(top_three_codes)
```

```
  violation_code  counts
94       0976160F   44811
9        0964040B   28712
20       0964090E   23683
```

2.


DLS
FINL
NA (Paid/Dismissed)
SEIZ
NA (Paid/Dismissed)
DETR
NA (Paid/Dismissed)
VIOL
NA (Paid/Dismissed)


Paid
Dismissed
Notice
Define Hearing Req
Court
Paid
Bankruptcy