30538 Problem Set 2: Parking Tickets

luyao Guo

2024-10-19

1. "This submission is my work alone and complies with the 30538 integrity policy." Add your initials to indicate your agreement: LG
2. "I have uploaded the names of anyone I worked with on the problem set **LG (https://docs.google.com/forms/d/1-zzHx762odGlpVWtgdIC55vqF-j3gqdAp6Pno1rIGK0/edit)**"
3. Late coins used this pset: 1 Late coins left after submission: 2
4. Knit your `ps2.qmd` to make `ps2.pdf`.
    - The PDF should not be more than 25 pages. Use `head()` and re-size figures when appropriate.
5. Push `ps2.qmd` and `ps2.pdf` to your github repo. It is fine to use Github Desktop.
6. Submit `ps2.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')
```

Data cleaning continued (15 points)

1.

```
df = pd.read_csv('parking_tickets_one_percent.csv')
# count the Na value
def count_na_values(df):
    na_counts = df.isna().sum()
```

```
    return na_counts[na_counts >
     ↪  0].reset_index(name='NA_Count').rename(columns={'index': 'Column'})

na_results = count_na_values(df)
print(na_results)
na_results.to_csv('na_results.csv', index=False)
```

```
                Column  NA_Count
0  license_plate_state        97
1   license_plate_type      2054
2              zipcode     54115
3                 unit        29
4         notice_level     84068
5  hearing_disposition    259899
```

2.

Hearing Disposition: This field is only filled when a ticket is contested and requires a formal hearing process. If a ticket is resolved without any dispute or legal proceedings, the field remains empty, as no hearing took place

Notice Level: This field is likely populated only when additional notices or escalations are issued to the violator. In cases where the ticket is paid promptly or resolved without further escalation, no notice is issued, leaving the field blank

Zipcode: The missing values in the zipcode field could be due to certain locations, such as highways, intersections, or large public areas, where a precise ZIP code is not applicable or necessary. This often happens for parking violations in areas without easily defined addresses

3.

```
# Filter the rows with CITY STICKER
city_sticker_tickets = df[df['violation_description'].str.contains('CITY
 ↪  STICKER', na=False)]

# Identify distinct violation codes
distinct_city_sticker_codes =
 ↪  city_sticker_tickets['violation_code'].drop_duplicates()
print("Distinct city sticker violation codes identified:",
 ↪  distinct_city_sticker_codes)

# fine amounts   questiion 4 coding part
fine_summary =
 ↪  city_sticker_tickets.groupby('violation_code')[['fine_level1_amount',
 ↪  'fine_level2_amount']].agg({
```

```
    'fine_level1_amount': ['min', 'max', 'mean'],
    'fine_level2_amount': ['min', 'max', 'mean']
})
print(fine_summary)

# Filter out irrelevant codes (according to the part description , exclude
 ↪  new violation code C & D)
codes_to_exclude = ['0964125C', '0964125D']
relevant_city_sticker_tickets =
 ↪  city_sticker_tickets[~city_sticker_tickets['violation_code'].isin(codes_to_exclude)]
print("Filtered city sticker tickets (after removing irrelevant codes):")
print(relevant_city_sticker_tickets[['violation_code', 'fine_level1_amount',
 ↪  'fine_level2_amount']])
```

```
Distinct city sticker violation codes identified: 14         0964125
2838       0976170
138604    0964125B
138699    0964125C
138839    0964125D
Name: violation_code, dtype: object
              fine_level1_amount             fine_level2_amount        \
                             min  max   mean                 min   max
violation_code
0964125                      120  120  120.0                 240   240
0964125B                     200  200  200.0                 400   400
0964125C                     500  500  500.0                 775  1000
0964125D                      30   30   30.0                  60    60
0976170                      120  120  120.0                 240   240


                     mean
violation_code
0964125         240.000000
0964125B        400.000000
0964125C        955.343511
0964125D         60.000000
0976170         240.000000
Filtered city sticker tickets (after removing irrelevant codes):
       violation_code  fine_level1_amount  fine_level2_amount
14            0964125                 120                 240
29            0964125                 120                 240
```

```
82            0964125              120              240
97            0964125              120              240
104           0964125              120              240
...               ...              ...              ...
287420       0964125B              200              400
287440       0964125B              200              400
287441       0964125B              200              400
287450       0964125B              200              400
287452       0964125B              200              400

[25019 rows x 3 columns]
```

Old violation code: The violation code with the earliest issue_date. Old violation code is 0964125

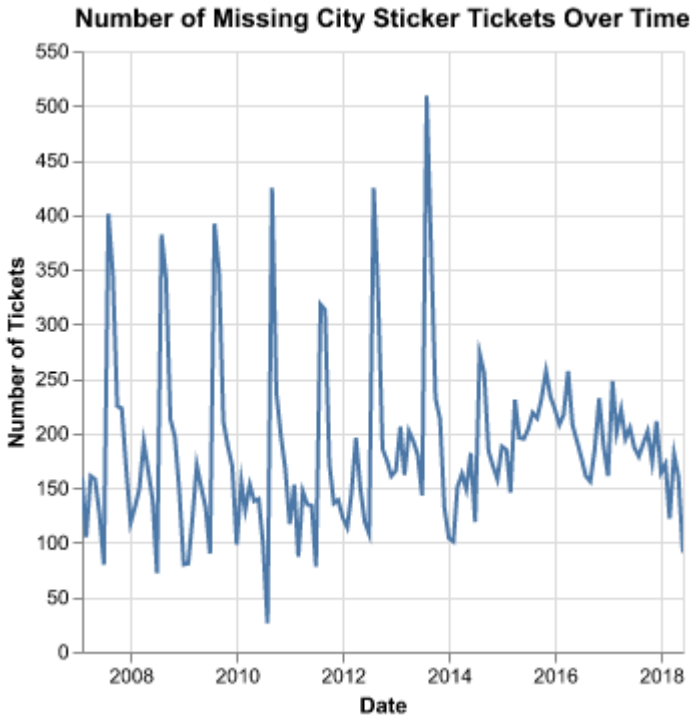
New violation code: The violation code with the latest issue_date that appears after a change in policy 0964125B, 0964125C, and 0964125D

4. This question answer alreday got from the last question coding part 0964125: Initial fine is $120. 0964125B: Initial fine is $200. 0964125C: Initial fine is $500. 0964125D: Initial fine is $30. Since ignore the tickets for vehicles over 16,000 pounds and focusing on violation codes 0964125 and 0964125B

Revenue increase from "missing city sticker" tickets (20 Points) 1.

```
# change to datetime format
city_sticker_tickets['issue_date'] =
 ↪  pd.to_datetime(city_sticker_tickets['issue_date'], errors='coerce')

# Combine the old and new violation code
city_sticker_tickets['combined_violation_code'] =
 ↪  city_sticker_tickets['violation_code'].replace({
    '0964125': 'combined_code',
    '0964125B': 'combined_code'
})

city_sticker_tickets_monthly =
 ↪  city_sticker_tickets.groupby(pd.Grouper(key='issue_date',
 ↪  freq='M')).size().reset_index(name='ticket_count')

# Plot
chart = alt.Chart(city_sticker_tickets_monthly).mark_line().encode(
    x=alt.X('issue_date:T', title='Date'),
```

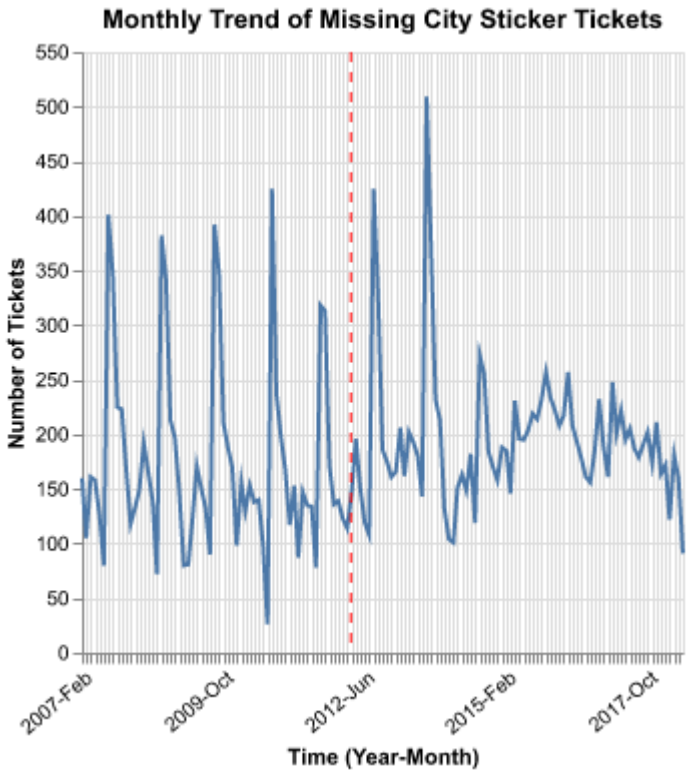
```
    y=alt.Y('ticket_count:Q', title='Number of Tickets')
).properties(
    title='Number of Missing City Sticker Tickets Over Time'
).interactive()
chart
```



2.

```
city_sticker_tickets_monthly['issue_date'] =
 ↪  pd.to_datetime(city_sticker_tickets_monthly['issue_date'])

# Base chart with area plot
base_chart = alt.Chart(city_sticker_tickets_monthly).mark_line().encode(
    x=alt.X('issue_date:T', title='Time (Year-Month)',
            axis=alt.Axis(format='%Y-%b', tickCount='month',
 ↪  labelAngle=-40)),
    y=alt.Y('ticket_count:Q', title='Number of Tickets')
).properties(
    title='Monthly Trend of Missing City Sticker Tickets'
)
```

```
# Add a vertical line for the price increase
price_increase_date = pd.DataFrame({'issue_date':
 ↪  pd.to_datetime(['2012-02-25'])})  # Ensure it's datetime
price_increase_line = alt.Chart(price_increase_date).mark_rule(color='red',
 ↪  strokeDash=[5, 5]).encode(
    x='issue_date:T'
)

# Combine the charts
final_chart = base_chart + price_increase_line
final_chart
```



I found the relevant help page Altair documentation on date and time formatting feature.Control how dates are displayed and create custom axes by specifying intervals (Altair's documentation on Time and Date handling and Axis configuration)

3.

```
# Tickets issued for old-increase code)
pre_change_tickets = city_sticker_tickets[
    (city_sticker_tickets['issue_date'].dt.year == 2011) &
    (city_sticker_tickets['violation_code'] == '0964125')
]
num_tickets_2011 = len(pre_change_tickets)

# old revenue (fine = $120)
pre_change_revenue = num_tickets_2011 * 120

# new revenue (fine = $200)
post_change_revenue = num_tickets_2011 * 200
revenue_increase_sample = post_change_revenue - pre_change_revenue
projected_revenue_increase = revenue_increase_sample * 100
print(f"Revenue increase based on 1% sample: ${projected_revenue_increase}")
```

```
Revenue increase based on 1% sample: $15464000
```

Revenue increase based on 1% sample: $15464000

4.

```
# Filter data for the year 2012
df_citysticker_2012 =
 ↪  city_sticker_tickets[city_sticker_tickets["issue_date"].dt.year == 2012]

# Convert the ticket_queue status to binary: 1 for "Paid", 0 for others
df_citysticker_2012['is_paid'] =
 ↪  df_citysticker_2012['ticket_queue'].apply(lambda x: 1 if x == "Paid" else
 ↪  0)

# Calculate the repayment rate for 2012
repayment_rate_2012 = df_citysticker_2012['is_paid'].mean()
df_citysticker_2011 =
 ↪  city_sticker_tickets[city_sticker_tickets["issue_date"].dt.year == 2011]
df_citysticker_2011['is_paid'] =
 ↪  df_citysticker_2011['ticket_queue'].apply(lambda x: 1 if x == "Paid" else
 ↪  0)

# epayment rate for 2011
repayment_rate_2011 = df_citysticker_2011['is_paid'].mean()
```

```
print(f"The repayment rate for 2011 is {repayment_rate_2011:.2f}, and the
 ↪  repayment rate for 2012 is {repayment_rate_2012:.2f}. \nThe repayment
 ↪  rate decreased after the price increase.")

# Calculate revenue changes
revenue_2011 = repayment_rate_2011 * num_tickets_2011 * 120
revenue_2012 = repayment_rate_2012 * num_tickets_2011 * 200
revenue_difference = revenue_2012 - revenue_2011
print(f"The change in revenue is ${revenue_difference:.2f}.")
```

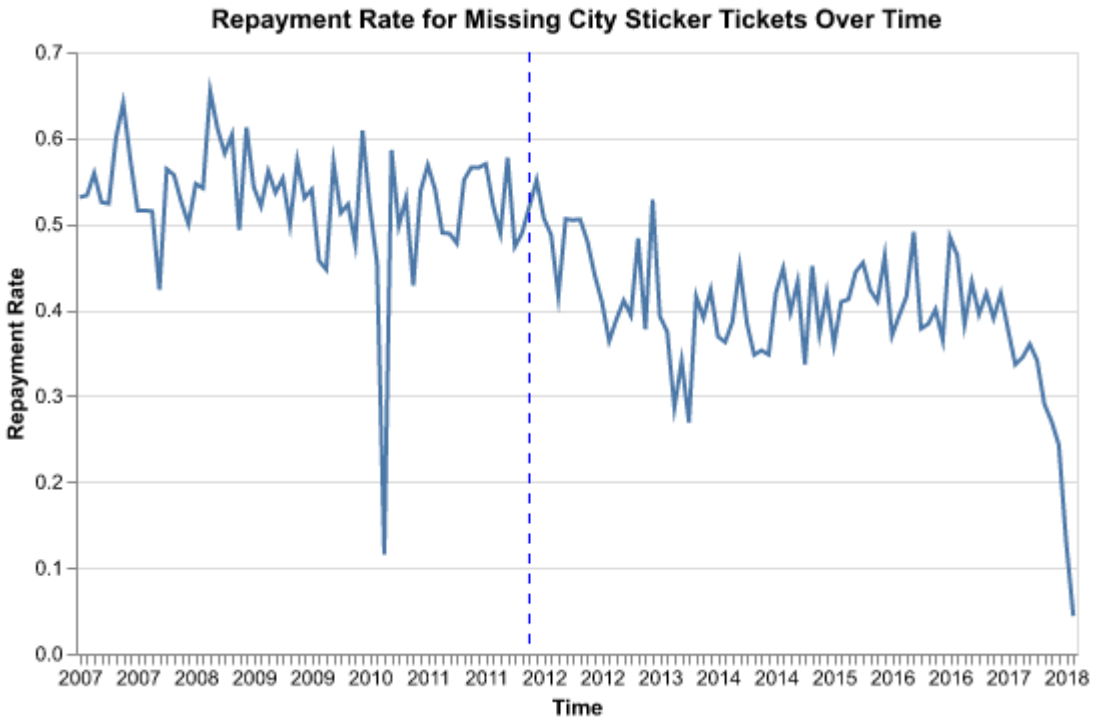
```
The repayment rate for 2011 is 0.54, and the repayment rate for 2012 is 0.49.
The repayment rate decreased after the price increase.
The change in revenue is $63551.16.
```

The repayment rate for 2011 is 0.54, and the repayment rate for 2012 is 0.49. The repayment rate decreased after the price increase. The change in revenue is $63551.16.

5.

```
city_sticker_tickets['is_paid'] =
 ↪  city_sticker_tickets['ticket_queue'].apply(lambda x: 1 if x == "Paid"
 ↪  else 0)
# the repayment rates over time
monthly_repayment_rate =
 ↪  city_sticker_tickets.groupby(city_sticker_tickets['issue_date'].dt.to_period('M'))['is_pa
monthly_repayment_rate['issue_date'] =
 ↪  monthly_repayment_rate['issue_date'].dt.to_timestamp()
base_chart = alt.Chart(monthly_repayment_rate, title="Repayment Rate for
 ↪  Missing City Sticker Tickets Over Time").mark_line().encode(
    x=alt.X("yearmonth(issue_date):O", title="Time",
 ↪  axis=alt.Axis(format="%Y", labelAngle=0)),
    y=alt.Y('repayment_rate:Q', title='Repayment Rate'),
    tooltip=['issue_date', 'repayment_rate']
).properties(
    width=500,
    height=300
)

# Add a vertical line
policy_change_date = pd.DataFrame({'date': pd.to_datetime(['2012-02-25'])})
policy_line = alt.Chart(policy_change_date).mark_rule(color='blue',
 ↪  strokeDash=[5, 5]).encode(
```

```
    x='date:T'
)
final_chart = base_chart + policy_line
final_chart
```

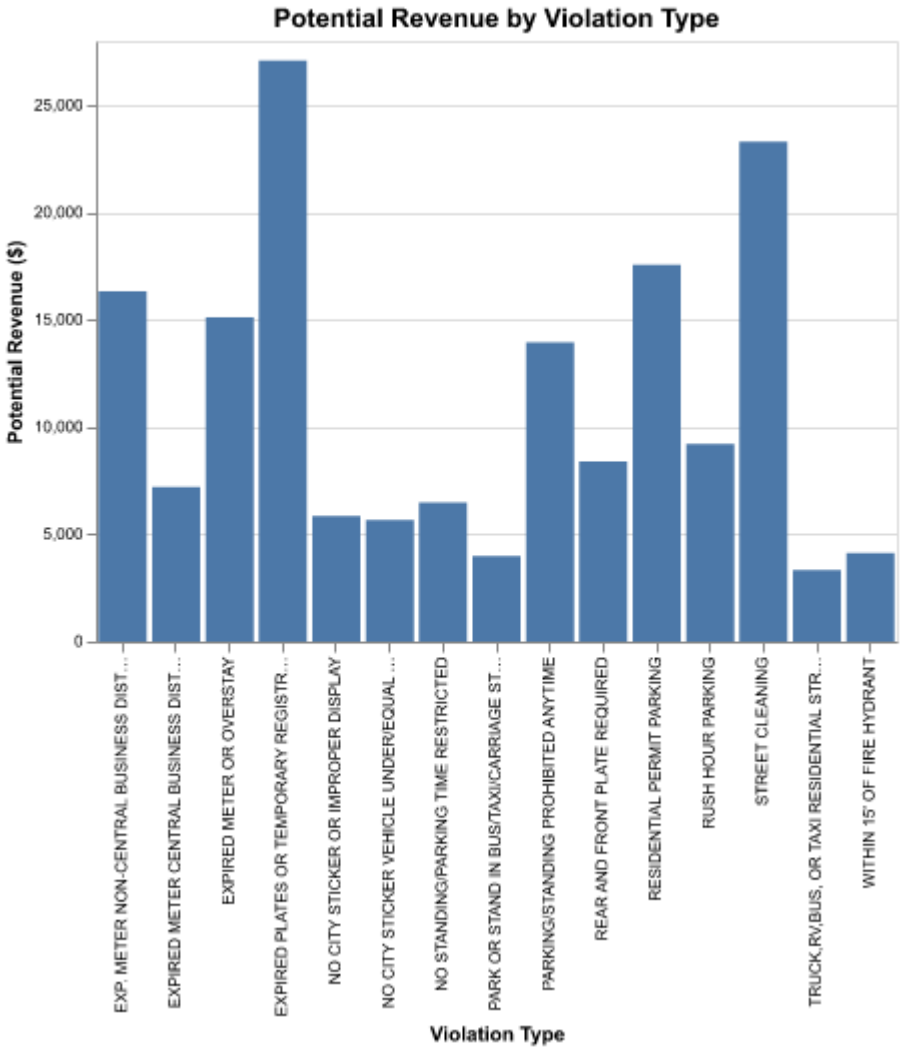


The drop in repayment rates after 2012 suggests that the price increase made it more difficult for people to pay their tickets. As fines became more expensive, fewer individuals were able or willing to settle them, leading to a noticeable decline in payments. This indicates that the higher cost may have discouraged compliance with the policy.

6.

```
df_violation_count =
 ↪  df.groupby('violation_description').size().reset_index(name='num_of_tickets')
df['is_paid'] = df['ticket_queue'].apply(lambda x: 1 if x == "Paid" else 0)
df_repayment_rate =
 ↪  df.groupby('violation_description')['is_paid'].mean().reset_index(name='repayment_rate')

# Combine the datasets
df_combined = pd.merge(df_violation_count, df_repayment_rate,
 ↪  on='violation_description')
```

```
# Calculate potential revenue
df_combined['potential_revenue'] = df_combined['num_of_tickets'] *
 ↪  df_combined['repayment_rate']
df_top_violations = df_combined.sort_values(by='potential_revenue',
 ↪  ascending=False).head(15)
chart = alt.Chart(df_top_violations, title="Potential Revenue by Violation
 ↪  Type").mark_bar().encode(
    x=alt.X("violation_description:N", title="Violation Type"),
    y=alt.Y("potential_revenue:Q", title="Potential Revenue ($)"),
    tooltip=['violation_description', 'num_of_tickets', 'repayment_rate',
 ↪  'potential_revenue']
).properties(
    width=400,
    height=300
).configure_axis(
    labelFontSize=8,
    titleFontSize=10
)
chart
```



Expired Plates or Temporary Registration: This violation generates the highest potential revenue due to the large number of tickets issued and a high repayment rate. Raising the fine here would likely result in a significant revenue increase without affecting repayment behavior. Street Cleaning: This is another violation with frequent occurrences and a relatively high repayment rate. Increasing the fine for this type of violation would effectively boost revenue, given the high volume of tickets issued. Residential Permit Parking: This violation also shows strong potential for revenue growth due to the combination of a high number of tickets and a solid repayment rate. Increasing fines here could yield higher revenue with minimal risk of reduced compliance. These three violations stand out because they have both a large number of tickets issued and a high likelihood of repayment, meaning that increasing the fines should directly lead to more revenue without changing behavior.

Headlines and sub-messages (20 points)

1.

```
# Drop the na
df_filtered = df.dropna(subset=['violation_description',
 ↪  'fine_level1_amount', 'total_payments'])

# different type of vilationa and repayment rate
df_filtered['paid'] = df_filtered['total_payments'] > 0
violation_summary = df_filtered.groupby('violation_description').agg(
    num_tickets=('ticket_number', 'count'),
    repayment_rate=('paid', 'mean'),
    fine_level1_mean=('fine_level1_amount', 'mean')
).reset_index()

# the most common violation type
violation_summary_sorted = violation_summary.sort_values(by='num_tickets',
 ↪  ascending=False)
top_5_violations = violation_summary_sorted.head(5)
print(top_5_violations)
```

```
                      violation_description  num_tickets  repayment_rate  \
23   EXPIRED PLATES OR TEMPORARY REGISTRATION        44811        0.608065
101                          STREET CLEANING        28712        0.815896
90                RESIDENTIAL PERMIT PARKING        23683        0.745978
19   EXP. METER NON-CENTRAL BUSINESS DISTRICT        20600        0.795485
81        PARKING/STANDING PROHIBITED ANYTIME        19753        0.710677

     fine_level1_mean
23          54.968869
101         54.004249
90          66.338302
19          46.598058
81          66.142864
```

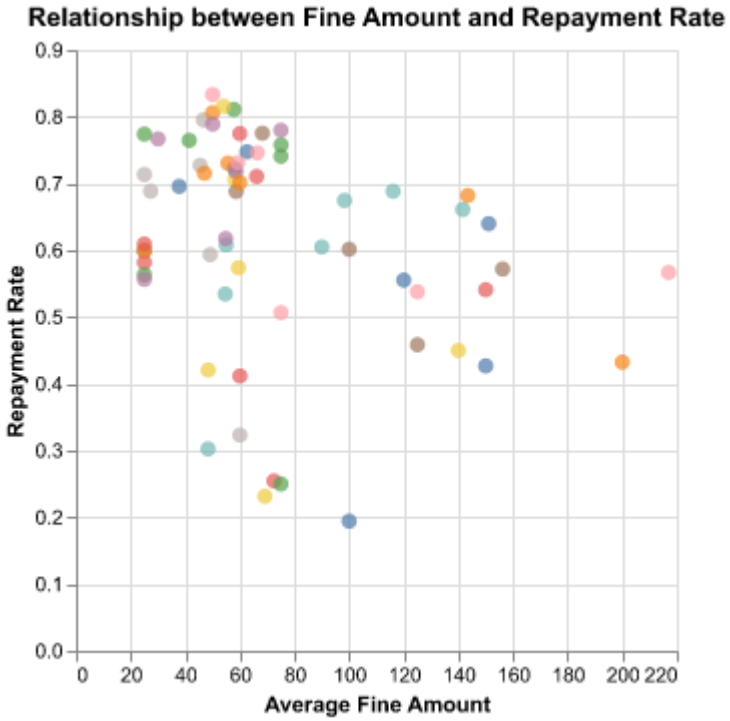
The five most common violations : Expired Plates or Temporary Registration: 60.8% repayment rate, $54.97 average fine. Street Cleaning: 81.6% repayment rate, $54.00 average fine. Residential Permit Parking: 74.6% repayment rate, $66.34 average fine. Expired Meter Non-Central Business District: 79.5% repayment rate, $46.60 average fine. Parking/Standing Prohibited Anytime: 71.1% repayment rate, $66.14 average fine.

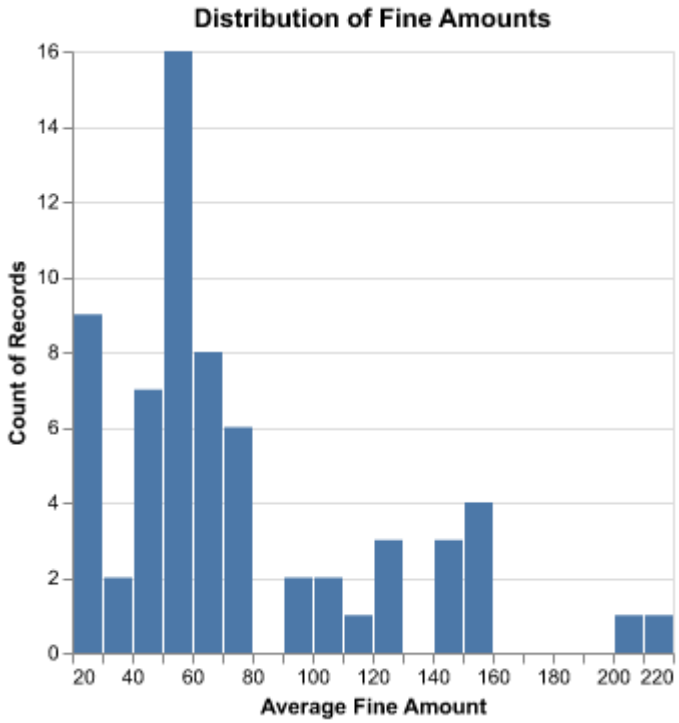
The top three violations to increase fines could be Street Cleaning, Residential Permit Parking, and Expired Plates or Temporary Registration based on their high repayment rates and frequent occurrence, which would likely increase revenue.

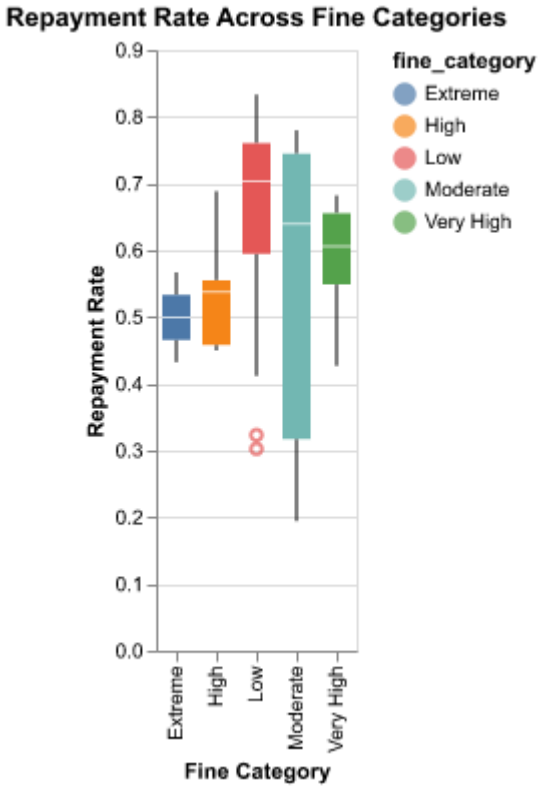
2.

```
# violations appear at least 100 times
violation_summary_filtered =
 ↪  violation_summary[violation_summary['num_tickets'] >= 100]
# Exclude the outlier
max_fine = violation_summary_filtered['fine_level1_mean'].max()
violation_summary_filtered_no_outlier =
 ↪  violation_summary_filtered[violation_summary_filtered['fine_level1_mean']
 ↪  < max_fine]

# Scatter plot
chart1 =
 ↪  alt.Chart(violation_summary_filtered_no_outlier).mark_circle(size=60).encode(
    x=alt.X('fine_level1_mean:Q', title='Average Fine Amount'),
    y=alt.Y('repayment_rate:Q', title='Repayment Rate'),
    color=alt.Color('violation_description:N', legend=None)
).properties(
    title='Relationship between Fine Amount and Repayment Rate'
).interactive()

#  Histogram
chart2 = alt.Chart(violation_summary_filtered_no_outlier).mark_bar().encode(
    x=alt.X('fine_level1_mean:Q', bin=alt.Bin(maxbins=30), title='Average
 ↪  Fine Amount'),
    y='count()'
).properties(
    title='Distribution of Fine Amounts'
)

# Box plot
violation_summary_filtered_no_outlier['fine_category'] = pd.cut(
    violation_summary_filtered_no_outlier['fine_level1_mean'], bins=5,
 ↪  labels=['Low', 'Moderate', 'High', 'Very High', 'Extreme']
)

chart3 =
 ↪  alt.Chart(violation_summary_filtered_no_outlier).mark_boxplot().encode(
    x=alt.X('fine_category:N', title='Fine Category'),
```

```
    y=alt.Y('repayment_rate:Q', title='Repayment Rate'),
    color='fine_category:N'
).properties(
    title='Repayment Rate Across Fine Categories'
)

chart1.display()
chart2.display()
chart3.display()
```




Distribution of Fine Amounts
Count of Records
0
2
4
6
8
10
12
14
16
20
40
60
80
100
120
140
160
180
200
220
Average Fine Amount


Repayment Rate Across Fine Categories
fine_category
Extreme
High
Low
Moderate
Very High
Repayment Rate
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
Extreme
High
Low
Moderate
Very High
Fine Category

3. I'd choose the scatter plot. It clearly shows how the repayment rates change with different fine amounts. This makes it easy for the City Clerk to see that higher fines might lead to lower repayment rates. The plot is simple and gets the point across without being too complicated.

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

1.

```
df['double_fine'] = df['fine_level2_amount'] == df['fine_level1_amount'] * 2

#  tickets for each violation description Merge
ticket_summary =
 ↪  df.groupby('violation_description').size().reset_index(name='total_tickets')
df_merged = pd.merge(df, ticket_summary, on='violation_description')
df_non_double = df_merged[(df_merged['total_tickets'] >= 100) &
 ↪  (df_merged['double_fine'] == False)]

# The absolute fine increase and percentage increase
df_non_double['fine_increase'] = df_non_double['fine_level2_amount'] -
 ↪  df_non_double['fine_level1_amount']
df_non_double['fine_percentage_increase'] = (df_non_double['fine_increase'] /
 ↪  df_non_double['fine_level1_amount']) * 100
final_summary = df_non_double[['violation_description', 'total_tickets',
 ↪  'double_fine', 'fine_increase', 'fine_percentage_increase']]
final_summary =
 ↪  final_summary.drop_duplicates(subset=['violation_description',
 ↪  'total_tickets'])
final_summary = final_summary.sort_values(by='total_tickets',
 ↪  ascending=False)
print(final_summary)
```

```
                        violation_description  total_tickets  double_fine  \
40                        PARK OR BLOCK ALLEY           2050        False
13                      DISABLED PARKING ZONE           2034        False
65739     SMOKED/TINTED WINDOWS PARKED/STANDING           1697        False
32664      BLOCK ACCESS/ALLEY/DRIVEWAY/FIRELANE           1579        False
65902   OBSTRUCTED OR IMPROPERLY TINTED WINDOWS            271        False
68067                PARK/STAND ON BICYCLE PATH            236        False
138699  NO CITY STICKER VEHICLE OVER 16,000 LBS.            131        False
```

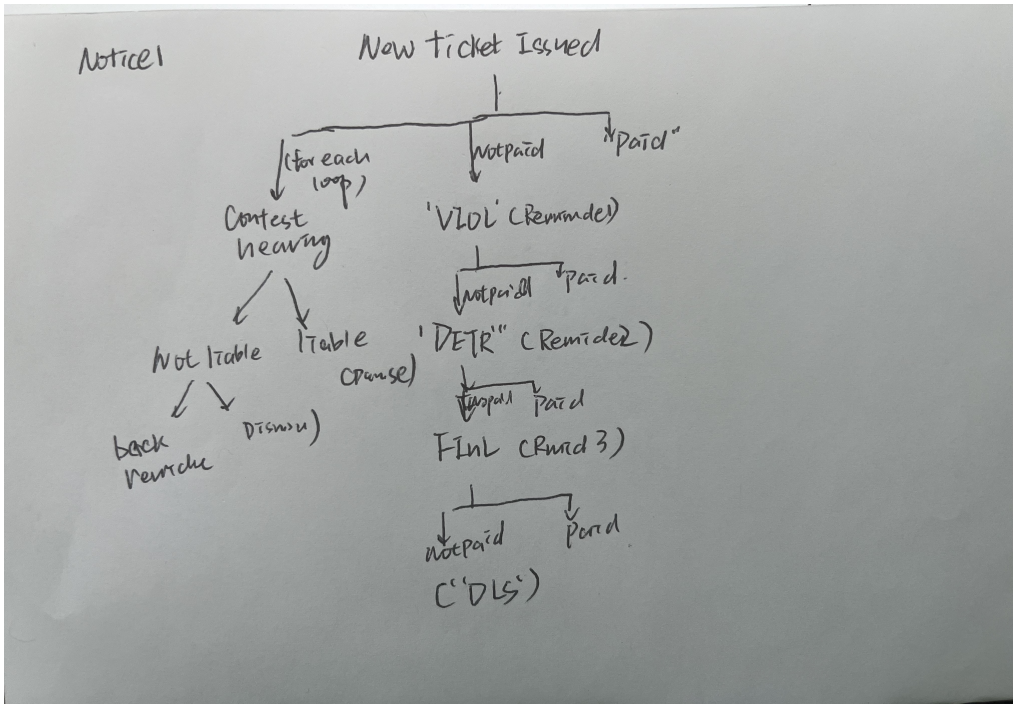
	fine_increase	fine_percentage_increase
40	100	66.666667
13	50	25.000000
65739	0	0.000000
32664	100	66.666667
65902	0	0.000000
68067	100	66.666667
138699	275	55.000000

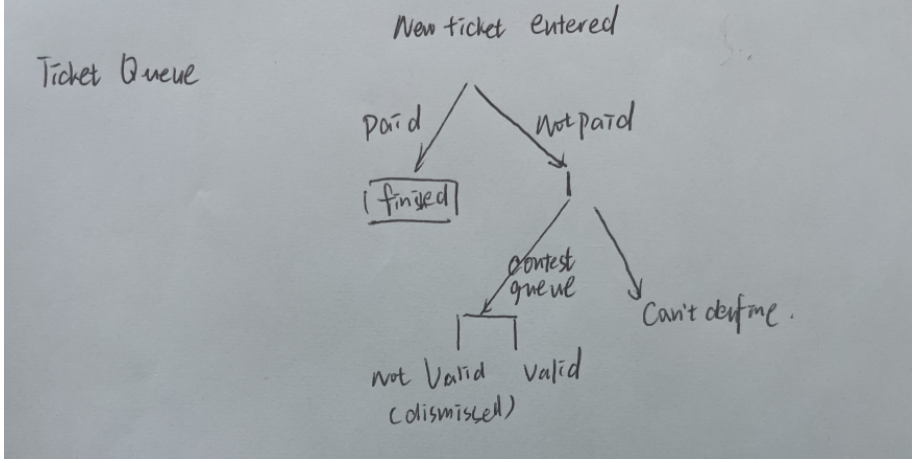
No, it does not hold for all violations.

Here are the violations with at least 100 citations that do not double in price:

PARK OR BLOCK ALLEY has 2,050 tickets, and the fine increases by $100 when unpaid, which is a 66.67% increase. DISABLED PARKING ZONE has 2,034 tickets, and the fine increases by $50, representing a 25.00% increase. SMOKED/TINTED WINDOWS PARKED/STANDING has 1,697 tickets, but the fine does not increase at all when unpaid. BLOCK ACCESS/ALLEY/DRIVEWAY/FIRELANE has 1,579 tickets, with the fine increasing by $100, a 66.67% increase. OBSTRUCTED OR IMPROPERLY TINTED WINDOWS has 271 tickets, but the fine does not increase when unpaid. PARK/STAND ON BICYCLE PATH has 236 tickets, and the fine increases by $100, a 66.67% increase. NO CITY STICKER VEHICLE OVER 16,000 LBS. has 131 tickets, and the fine increases by $275, which is a 55.00% increase.

2.

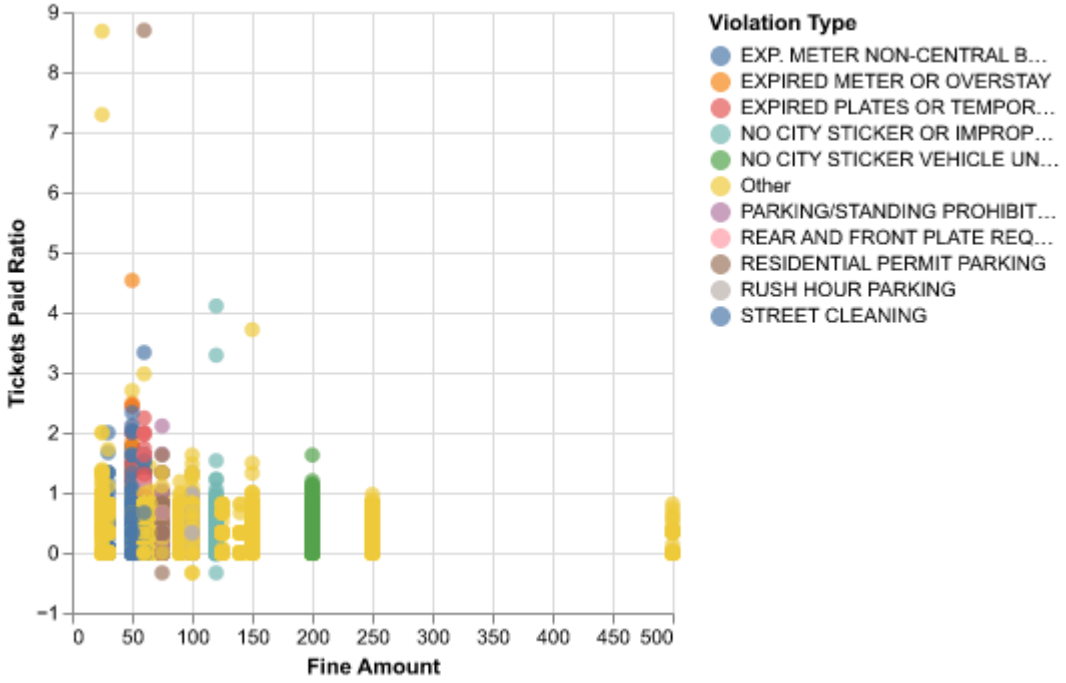


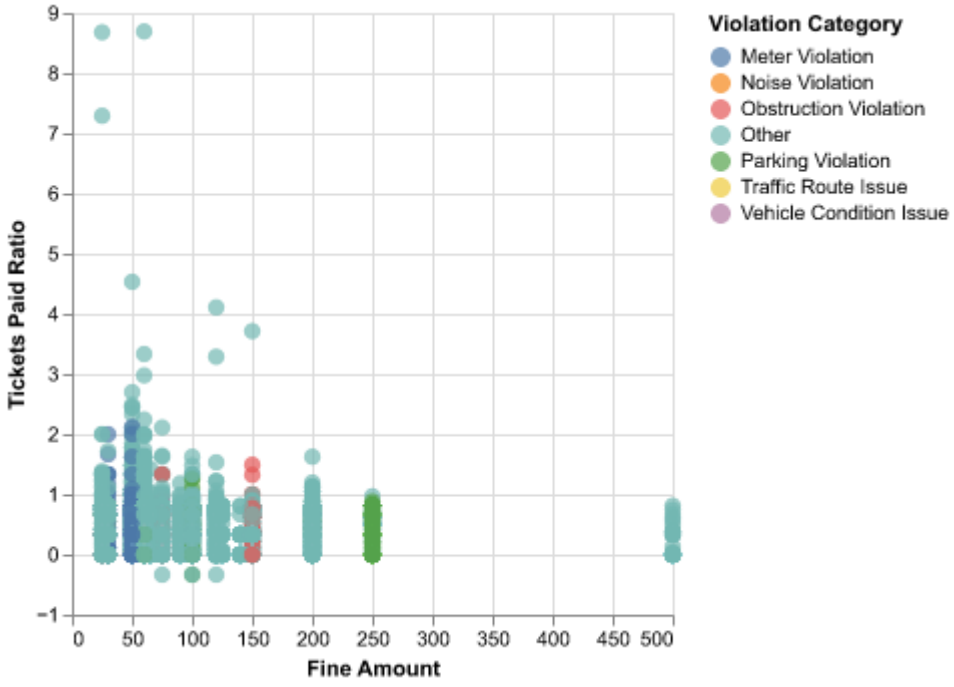


3.

```
import altair as alt
import numpy as np

# payment_ratio
df_filtered['payment_ratio'] = df_filtered['total_payments'] /
 ↪  (df_filtered['fine_level1_amount'] + df_filtered['fine_level2_amount'])

# the 10 most frequent violation descriptions
top_violation_types =
 ↪  df_filtered['violation_description'].value_counts().nlargest(10).index
df_filtered['violation_label'] =
 ↪  np.where(df_filtered['violation_description'].isin(top_violation_types),

 ↪  df_filtered['violation_description'], 'Other')

scatter_plot_a = alt.Chart(df_filtered).mark_circle(size=60).encode(
    x=alt.X("fine_level1_amount:Q", title="Fine Amount"),
    y=alt.Y("payment_ratio:Q", title="Tickets Paid Ratio"),
    color=alt.Color("violation_label:N", title="Violation Type")
)
def categorize_violation_type(description):
    if description in ["20'OF CROSSWALK", "CURB LOADING ZONE", "DISABLED
     ↪  PARKING ZONE", "DOUBLE PARKING OR STANDING"]:
        return "Parking Violation"
    elif description in ["BLOCK ACCESS/ALLEY/DRIVEWAY/FIRELANE", "DISABLED
     ↪  CURB CUT"]:
        return "Obstruction Violation"
    elif description == "ABANDONED VEH. FOR 7 DAYS OR INOPERABLE":
        return "Vehicle Condition Issue"
    elif description == "3-7 AM SNOW ROUTE":
        return "Traffic Route Issue"
    elif description == "EXP. METER NON-CENTRAL BUSINESS DISTRICT":
        return "Meter Violation"
```

```
    elif description == "BURGLAR ALARM SOUNDING OVER 4 MINUTES":
        return "Noise Violation"
    else:
        return "Other"
df_filtered['violation_category'] =
 ↪  df_filtered['violation_description'].apply(categorize_violation_type)
scatter_plot_b = alt.Chart(df_filtered).mark_circle(size=70).encode(
    x=alt.X("fine_level1_amount:Q", title="Fine Amount"),
    y=alt.Y("payment_ratio:Q", title="Tickets Paid Ratio"),
    color=alt.Color('violation_category:N', title="Violation Category")
)
alt.data_transformers.disable_max_rows()
scatter_plot_a.display()
scatter_plot_b.display()
```





Extra Credit (max 5 points)