

# Problem Set 2 Answer

1. “This submission is my work alone and complies with the 30538 integrity policy.” Add your initials to indicate your agreement: **HX**
2. “I have uploaded the names of anyone I worked with on the problem set here” **HX** (2 point)
3. Late coins used this pset: **0** Late coins left after submission: **4**
4. Knit your ps2.qmd to a pdf named ps2.pdf. • The PDF should not be more than 25 pages. Use head() and re-size figures when appropriate.
4. Push ps2.qmd and ps2.pdf to your github repo. It is fine to use Github Desktop.
6. Submit ps2.pdf via Gradescope (8 points)
5. Tag your submission in Gradescope

```
import pandas as pd
import altair as alt
import os
alt.renderers.enable('png')

# Read in the csv file
# Warning message is suppressed for output clarity.
data = pd.read_csv(
    'problem_sets/ps2/data/parking_tickets_one_percent.csv',
    index_col=[0]
)
df = pd.DataFrame(data)
```

## Data Cleaning continued

*Q1*

```
# Define the function to count NAs
def count_na(df):
    labels = []
```

```

na_counts = []
for label, content in df.items():
    count = content.isna().sum()
    labels.append(label)
    na_counts.append(count)
result_dict = {'Variable': labels, 'NA Count': na_counts}
result_df = pd.DataFrame.from_dict(result_dict)
return result_df

# Conduct a test
df_test = pd.DataFrame({
    'A': [1, 2, None, None],
    'B': [1, 2, 3, 4],
    'C': [1, 'Apple', 3, None]
})
print(df_test, '\n')
print(count_na(df_test), '\n')

print('So the function works.\nApplying to our dataset:', '\n')
print(count_na(df))

```

	A	B	C
0	1.0	1	1
1	2.0	2	Apple
2	NaN	3	3
3	NaN	4	None

	Variable	NA Count
0	A	2
1	B	0
2	C	1

So the function works.  
Applying to our dataset:

	Variable	NA Count
0	ticket_number	0
1	issue_date	0
2	violation_location	0
3	license_plate_number	0
4	license_plate_state	97

5	license_plate_type	2054
6	zipcode	54115
7	violation_code	0
8	violation_description	0
9	unit	29
10	unit_description	0
11	vehicle_make	0
12	fine_level1_amount	0
13	fine_level2_amount	0
14	current_amount_due	0
15	total_payments	0
16	ticket_queue	0
17	ticket_queue_date	0
18	notice_level	84068
19	hearing_disposition	259899
20	notice_number	0
21	officer	0
22	address	0

## Q2

The zipcode, notice\_level and hearing\_disposition are missing much more frequently than others, probably because of the way data was collected. According to the dictionary, zipcode is the ZIP code associated with the vehicle registration. If the registration information is unclear or incomplete, this field might be left empty. notice\_level describes the type of notice the city has sent a motorist. If no notice was sent, this field would be blank. Similarly, hearing\_disposition represents the outcome of a hearing. If the ticket was not contested, this field would be blank.

In conclusion, these three variables contain more NAs than average because of their data entry styles. These fields are filled only if certain conditions are met, such as ticket got contested. Other variables that capture essential information, such as ticket numbers and issue dates, are routinely collected with each ticket issued. Thus, they are more likely to be NA than others.

## Q3

```
df_Q3 = df[df['violation_description'].str.contains('NO CITY STICKER', na=False)]
print(df_Q3['violation_code'].unique()) # Ordered by first apperance in dataset
```

```
['0964125' '0976170' '0964125B' '0964125C']
```

By inspection, we know that 0976170 and 0964125C are extremely rare cases in the dataset. Here we ignore these two values. Old violation code: 0964125; New violation code: 0964125B.

#### Q4

According to the articles, citations for not having a required vehicle sticker rose from \$120 to \$200. So we have the cost under old violation code: \$120; under new code: \$200. In the dataset, we refer to the variable of `fine_level1_amount`.

```
# For '0964125'
print(df[df['violation_code'] == '0964125']['fine_level1_amount'].unique())
```

[120]

```
# For '0964125B'
print(df[df['violation_code'] == '0964125B']['fine_level1_amount'].unique())
# '0964125C' ignored as instructed.
```

[200]

The result above aligns with the article. To be more specific, the cost of an initial offense under 0964125 was \$120, and the cost of an initial offense under 0964125B was \$200.

### Revenue increase from ‘missing city sticker’ tickets

#### Q1

```
# Unify the codes with 000
df_unified = df.copy()
df_unified = df_unified.replace('0964125', '000')
df_unified = df_unified.replace('0964125B', '000')
df_unified = df_unified[df_unified['violation_code'] == '000']
# Extract year-month and count
df_unified['issue_date'] = pd.to_datetime(
df_unified['issue_date'])
df_unified['issue_y_m'] = df_unified['issue_date'].dt.strftime('%Y-%m')
grouped_date = df_unified.groupby('issue_y_m')
Q1_count = grouped_date.size().reset_index(name='count')
print(Q1_count.head(10))

# Aggregate by year and month
chart_1 = alt.Chart(Q1_count).mark_area(
    interpolate='step-after',
```

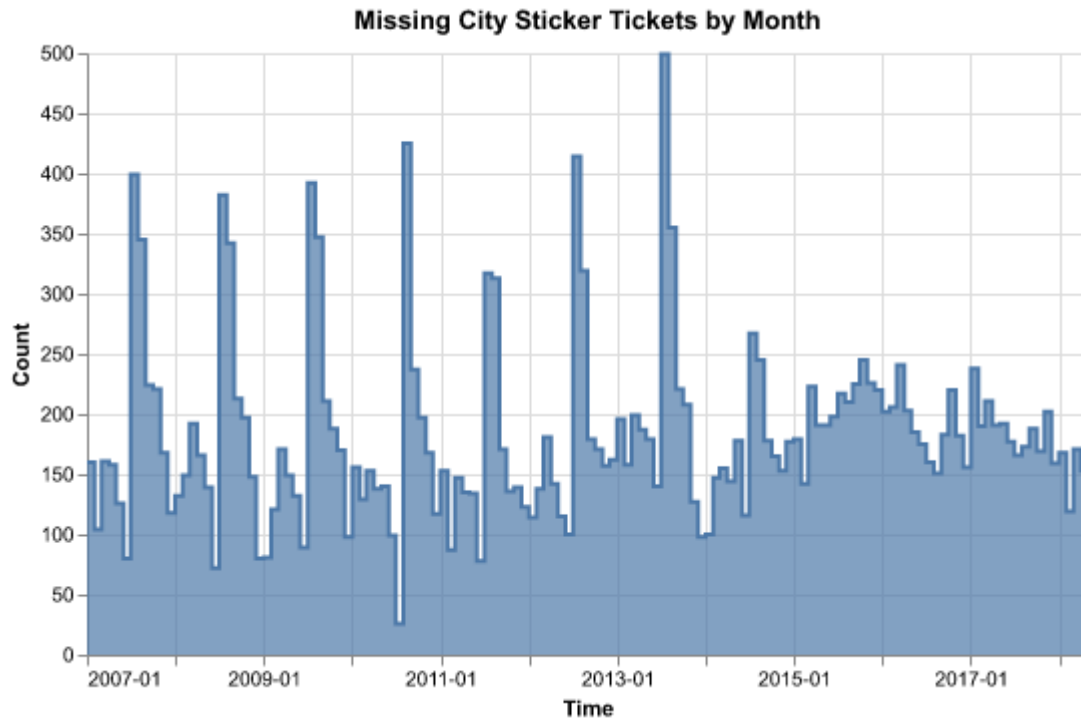
```

        line=True
    ).encode(
        alt.X(
            'issue_y_m:T',
            title='Time',
            axis=alt.Axis(format='%Y-%m')
        ),
        alt.Y(
            'count:Q',
            title='Count'
        )
    ).properties(
        width=500,
        title='Missing City Sticker Tickets by Month'
    )

chart_1

```

	issue_y_m	count
0	2007-01	160
1	2007-02	104
2	2007-03	161
3	2007-04	158
4	2007-05	126
5	2007-06	80
6	2007-07	399
7	2007-08	345
8	2007-09	224
9	2007-10	221



*Q2*

```
# Find the date of cost increase
df_new_code = df_unified[df_unified['fine_level1_amount'] == 200]
print(df_new_code['issue_date'].iloc[0])
print('So we know the date of cost increase was 2012-02-25.')
cost_increase_date = '2012-02-25'

# Add a vertical line at 2012-02-25
rule = alt.Chart(
    pd.DataFrame(
        {
            'cost_increase_date': [cost_increase_date],
            'label': ['Cost Increase']
        }
    )
).mark_rule(
    color='red',
    strokeDash=[8, 4]
).encode(
    x='cost_increase_date:T',
    tooltip=['label']
)
```

```

)

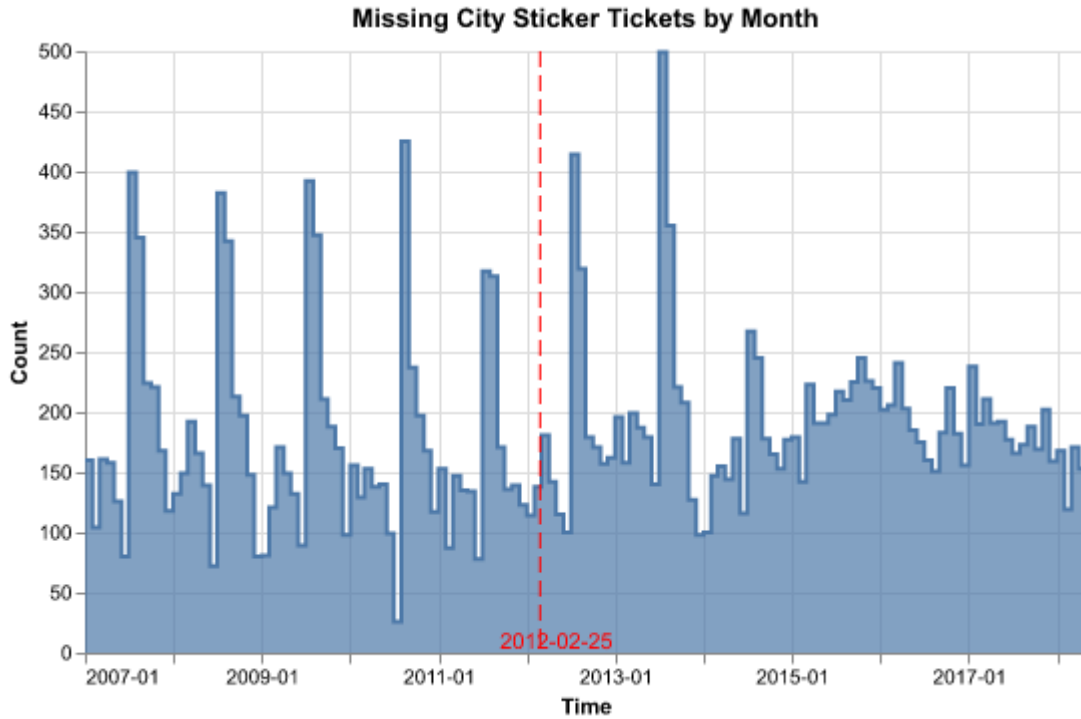
# Label for cost increase
text = alt.Chart(
    pd.DataFrame(
        {
            'cost_increase_date': [cost_increase_date],
            'label': ['2012-02-25']
        }
    )
    .mark_text(
        align='left',
        baseline='bottom',
        dx=-20,
        dy=150,
        color='red'
    ).encode(
        x='cost_increase_date:T',
        text='label'
    )
)

chart_1 + rule + text

```

2012-02-25 02:00:00

So we know the date of cost increase was 2012-02-25.



[Help page](https://altair-viz.github.io/user_guide/marks/rule.html#) ([https://altair-viz.github.io/user\\_guide/marks/rule.html#](https://altair-viz.github.io/user_guide/marks/rule.html#))

*Q3*

```
# Filter out data in 2011
df_2011 = df_unified[df_unified['issue_date'].dt.year == 2011]
tickets_2011 = df_2011.shape[0]
print(f'{tickets_2011} no city sticker tickets were issued in 2011.')
# We assume the same amount later on
revenue_increase = tickets_2011 * (200 - 120) * 100 # Dataset is one percent
print(revenue_increase)
```

1933 no city sticker tickets were issued in 2011.  
15464000

So they should have expected \$15464000 of revenue increase, which is approximately 15.5 million US dollars per year.



#### Q4

```
# Before price increased
df_before_increase = df_unified[df_unified['fine_level1_amount'] == 120]
df_before_paid = df_before_increase[df_before_increase['ticket_queue'] == 'Paid']
paid_rate_before = df_before_paid.shape[0] / df_before_increase.shape[0]
print(f'Before increase, the repayment rate was {paid_rate_before}.')

# Filter out data in 2013
df_2013 = df_unified[df_unified['issue_date'].dt.year == 2013]
df_2013_paid = df_2013[df_2013['ticket_queue'] == 'Paid']
paid_rate_2013 = df_2013_paid.shape[0] / df_2013.shape[0]
print(f'In the calendar year after increase, repayment rate was {paid_rate_2013}.')

print('So we see a drop in repayment rate in tickets after the price was increased.')
print('From around 54% to 41%.')

# Under repayment rate of 41%
# We still assume 1933 tickets per year
revenue_increase_rate = (
    tickets_2011 * paid_rate_2013 * 200 - tickets_2011 * paid_rate_before * 120
) * 100
print(revenue_increase_rate)
```

Before increase, the repayment rate was 0.5431306934374419.

In the calendar year after increase, repayment rate was 0.4059213089209194.

So we see a drop in repayment rate in tickets after the price was increased.

From around 54% to 41%.

3094458.2379078404

Under the new repayment rates, the change in revenue should have been about approximately 3.1 million US dollars per year.

#### Q5

```
# Again use the dataframe aggregated by month and year
# We already have total tickets for each month-year in the previous question
# Now we only need to count paid tickets
df_paid = df_unified[df_unified['ticket_queue'] == 'Paid']
group_paid = df_paid.groupby('issue_y_m')
df_paid_count = group_paid.size().reset_index(name='count_paid')
```

```

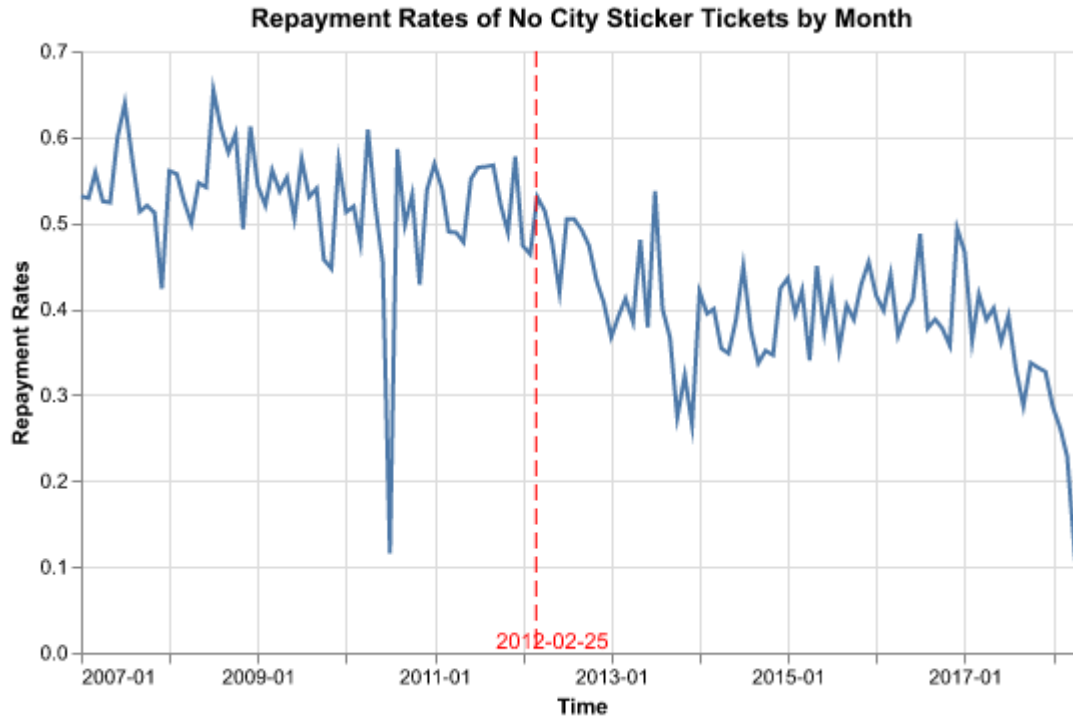
# Combine with the count of total tickets
Q5_count = pd.merge(df_paid_count, Q1_count)
# Calculate repayment rates in a new column
Q5_count['repayment_rate'] = Q5_count['count_paid'] / Q5_count['count']
print(Q5_count.head(10)) # Ready for plotting

chart2 = alt.Chart(Q5_count).mark_line().encode(
    alt.X('issue_y_m:T', title='Time', axis=alt.Axis(format='%Y-%m')),
    alt.Y('repayment_rate:Q', title='Repayment Rates')
).properties(
    width=500,
    title='Repayment Rates of No City Sticker Tickets by Month'
)

# The verticle line and annotation have been set in Question 2.
chart2 + rule + text

```

	issue_y_m	count_paid	count	repayment_rate
0	2007-01	85	160	0.531250
1	2007-02	55	104	0.528846
2	2007-03	90	161	0.559006
3	2007-04	83	158	0.525316
4	2007-05	66	126	0.523810
5	2007-06	48	80	0.600000
6	2007-07	255	399	0.639098
7	2007-08	198	345	0.573913
8	2007-09	115	224	0.513393
9	2007-10	115	221	0.520362



From the plot, we can see a downward trend in repayment rates of no city sticker tickets over time, after the new policy was introduced. The repayment rates kept fluctuating, but after the price went up, repayment rates were in general significantly lower than before. The average level under the new policy was apparently below that under the old code. Moreover, toward the end of our dataset, there was an ongoing sharp decline in repayment rates. The rates dropped to extremely low by 2018.

This indicates the need to balance repayment rates and penalty amounts, in order to boost the government revenue.

### Q6

```
# 000 represents no city sticker
df_Q6 = df.copy()
df_Q6 = df_Q6.replace('0964125', '000')
df_Q6 = df_Q6.replace('0964125B', '000')

# Filter out records before 2012
df_Q6['issue_date'] = pd.to_datetime(df_Q6['issue_date'])
df_Q6 = df_Q6[df_Q6['issue_date'].dt.year < 2012]

# Calculate repayment rates
grouped_code = df_Q6.groupby('violation_code')
```

```

df_code_count = grouped_code.size().reset_index(name='total_tickets')
# Those never paid are ignored here, they are missed in filter and merge
# Never paid tickets are of no relevance
df_Q6_paid = df_Q6[df_Q6['ticket_queue'] == 'Paid']
grouped_code_paid = df_Q6_paid.groupby('violation_code')
df_code_count_paid = grouped_code_paid.size().reset_index(name='paid_tickets')

Q6_count = pd.merge(df_code_count, df_code_count_paid)
Q6_count['repayment_rates'] = Q6_count['paid_tickets'] / Q6_count['total_tickets']
# We have both ticket counts and repayment rates

# Explore the dataset: tickets issued
Q6_count = Q6_count.sort_values(by='total_tickets', ascending=False)
print(Q6_count.head(15))

```

	violation_code	total_tickets	paid_tickets	repayment_rates
73	0976160F	21906	13316	0.607870
51	0964190	18117	14580	0.804769
10	0964040B	14740	12037	0.816621
20	0964090E	11452	8718	0.761264
0	000	10543	5742	0.544627
43	0964150B	9673	7032	0.726972
69	0976160A	8339	5066	0.607507
18	0964080A	7121	5576	0.783036
52	0964190A	3503	2951	0.842421
19	0964080B	3435	2643	0.769432
41	0964140B	3276	2290	0.699023
23	0964100A	2967	2083	0.702056
46	0964170A	2323	1670	0.718898
40	0964130	1594	817	0.512547
30	0964110A	1409	905	0.642300

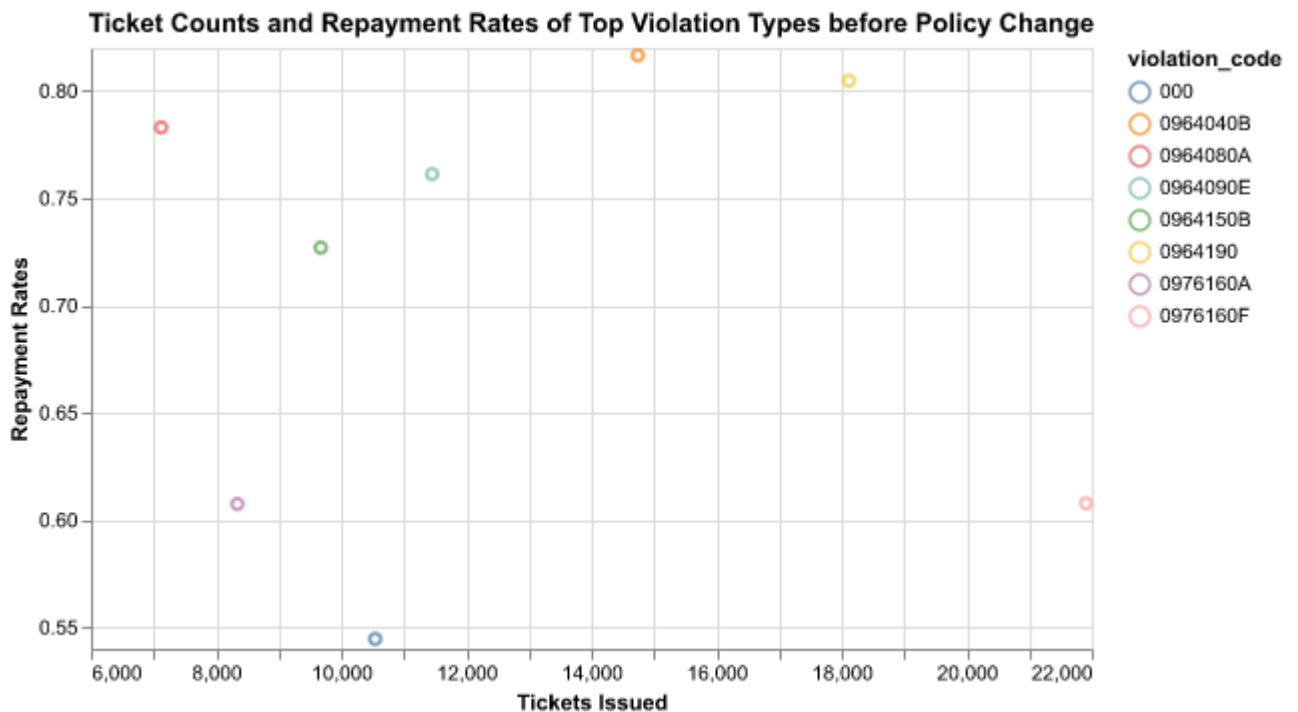
The suggested three violation types would be 0964190 ('EXPIRED METER OR OVER-STAY'), 0964040B ('STREET CLEANING'), and 0964090E ('RESIDENTIAL PERMIT PARKING').

This is because we need to consider both total number of tickets and the repayment rates. Both need to be maximized to boost revenue effectively. From the dataframe Q6\_count as shown above, we could tell that these three violation codes have relatively higher total\_ticket counts and repayment rates.

```
# To boost revenue, the amount of tickets should be large to make a difference.
# By inspection, we set the bar of 5000 total tickets before 2012 here.
Q6_count = Q6_count[Q6_count['total_tickets'] > 5000]

chart_3 = alt.Chart(Q6_count).mark_point().encode(
    alt.X('total_tickets:Q', title='Tickets Issued').scale(zero=False),
    alt.Y('repayment_rates:Q', title='Repayment Rates').scale(zero=False),
    color='violation_code:N').properties(
    width=500,
    title='Ticket Counts and Repayment Rates of Top Violation Types before Policy Change'
)

chart_3
```



The plot supports the argument by providing a contrast among different violation codes. To increase revenue efficiently, we need to identify points that are positioned both high on the y-axis, indicating higher repayment rates, and far to the right on the x-axis, representing a larger number of tickets issued. In this plot, we could find 0964040B, 0964090E and 0964190 might be the three best choices.

## Headlines and sub-messages

Q1

```
grouped_description = df.groupby('violation_description')

# Description and repayment rate
total_count = grouped_description.size().reset_index(name='total_tickets')

df_paid = df[df['ticket_queue'] == 'Paid'].copy()
grouped_des_paid = df_paid.groupby('violation_description')
paid_count = grouped_des_paid.size().reset_index(name='paid_tickets')

# Those left out in df_paid are violation never paid
all_violations = df['violation_description'].unique()
paid_violations = df_paid['violation_description'].unique()
unpaid_violations = [item for item in all_violations if item not in paid_violations]
# Fill in with 0 paid tickets for these three types
unpaid_dict = {
    'violation_description': unpaid_violations,
    'paid_tickets': [0, 0, 0]
}
unpaid_df = pd.DataFrame(unpaid_dict)
paid_count = pd.concat([paid_count, unpaid_df], ignore_index=True)

description_payment = pd.merge(
    total_count, paid_count,
    on='violation_description'
)
description_payment['repayment_rates'] = description_payment['paid_tickets'] / \
    description_payment['total_tickets']

# Description and average fine amount
description_price = grouped_description['fine_level1_amount'].mean()
description_price = description_price.reset_index(name='avg_price')

description_summary = pd.merge(description_payment, description_price)

# Sort by frequency and print most common 5
Q_3_1 = description_summary.sort_values('total_tickets', ascending=False)
Q_3_1 = Q_3_1[['violation_description', 'repayment_rates', 'avg_price']]
print(Q_3_1.head(5))
```

	violation_description	repayment_rates	avg_price
23	EXPIRED PLATES OR TEMPORARY REGISTRATION	0.604361	54.968869
101	STREET CLEANING	0.811612	54.004249
90	RESIDENTIAL PERMIT PARKING	0.742262	66.338302
19	EXP. METER NON-CENTRAL BUSINESS DISTRICT	0.792913	46.598058
81	PARKING/STANDING PROHIBITED ANYTIME	0.705817	66.142864

*Q2*

```
# Filter out types more than 100 times
Q_3_2 = description_summary[description_summary['total_tickets'] > 99]

# Exclude the extreme value with high fine amount
print(Q_3_2.sort_values('avg_price', ascending=False)[0: 5])
print('So we need to exclude NO CITY STICKER VEHICLE OVER 16,000 LBS.')
Q_3_2 = Q_3_2[Q_3_2['violation_description'] != 'NO CITY STICKER VEHICLE OVER 16,000 LBS.']

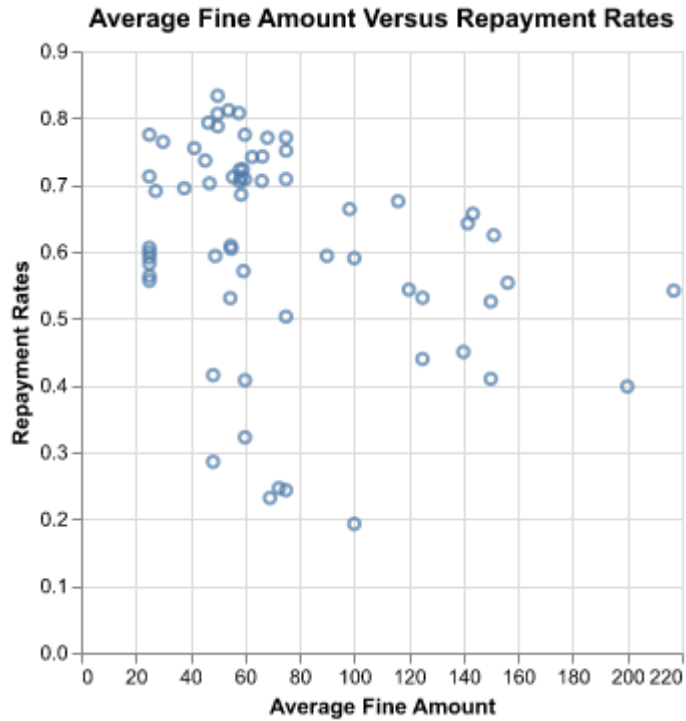
# Scatter plot
chart_4 = alt.Chart(Q_3_2).mark_point().encode(
    alt.X('avg_price:Q', title='Average Fine Amount'),
    alt.Y('repayment_rates:Q', title='Repayment Rates')
).properties(
    title='Average Fine Amount Versus Repayment Rates'
)

chart_4
```

	violation_description	total_tickets \
42	NO CITY STICKER VEHICLE OVER 16,000 LBS.	131
15	DISABLED PARKING ZONE	2034
43	NO CITY STICKER VEHICLE UNDER/EQUAL TO 16,000 ...	14246
54	OBSTRUCTED OR IMPROPERLY TINTED WINDOWS	271
95	SMOKED/TINTED WINDOWS PARKED/STANDING	1697

	paid_tickets	repayment_rates	avg_price
42	16	0.122137	500.000000
15	1102	0.541790	216.986234
43	5675	0.398357	200.000000
54	150	0.553506	156.180812
95	1060	0.624632	151.090159

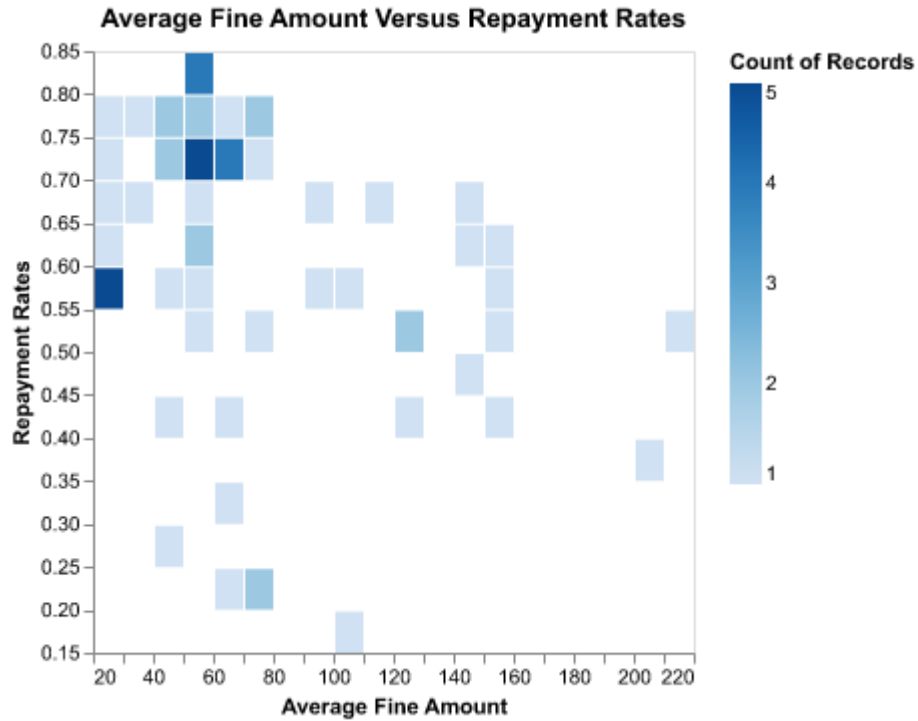
So we need to exclude NO CITY STICKER VEHICLE OVER 16,000 LBS.



```
chart_5 = alt.Chart(Q_3_2).mark_bar().encode(
    alt.X(
        'avg_price:Q',
        title='Average Fine Amount',
        bin=alt.BinParams(maxbins=20)
    ),
    alt.Y(
        'repayment_rates:Q',
        title='Repayment Rates',
        bin=alt.BinParams(maxbins=20)
    ),
    alt.Color('count()')
).properties(
    title='Average Fine Amount Versus Repayment Rates'
)

chart_5
```





```
chart_6 = alt.Chart(Q_3_2).mark_boxplot().encode(
    alt.X(
        'avg_price:Q',
        title='Average Fine Amount',
        bin=alt.BinParams(maxbins=20)
    ),
    alt.Y(
        'repayment_rates:Q',
        title='Repayment Rates'
    )
).properties(
    title='Average Fine Amount Versus Repayment Rates'
)

chart_6
```

