30538 Problem Set 2: Parking Tickets Solutions

Peter Ganong, Maggie Shi, and Ozzy Houck

2024-09-30

1. **PS2:** Due Sat Oct 19 at 5:00PM Central. Worth 100 points.

We use (*) to indicate a problem that we think might be time consuming.

Steps to submit (10 points on PS2)

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**" **___** (2 point)
3. Late coins used this pset: **___** Late coins left after submission: **___**
4. Knit your `ps2.qmd` as an html document and print to a pdf 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 (8 points)
7. Tag your submission in Gradescope

Background Recap

Read **this** article and **this** shorter article. If you are curious to learn more, **this** page has all of the articles that ProPublica has done on this topic. This problem set is a continuation of PS1 using the same data. Please start by loading the data in the same way as PS1.

```
import pandas as pd
import altair as alt
import unittest
```

```
import time

import warnings
warnings.filterwarnings('ignore')
alt.renderers.enable("png")
```

```
RendererRegistry.enable('png')
```

```
# Load the data in the same way as ps1
def read_parking_tickets(file_path):
    start_time = time.time()
    df = pd.read_csv(file_path)
    end_time = time.time()
    elapsed_time = end_time - start_time
    assert len(df) == 287458, "The number of rows is not as expected."
    print(f"Time taken to read the file: {elapsed_time:.2f} seconds")
    df['issue_date'] = pd.to_datetime(df['issue_date'])
    return df

# Example usage
file_path = 'data/parking_tickets_one_percent.csv'
df = read_parking_tickets(file_path)

# make issue_date a datetime
df['issue_date'] = pd.to_datetime(df['issue_date'])
```

```
Time taken to read the file: 0.93 seconds
```

Data cleaning continued (15 points)

1. For each column, how many rows are `NA`? Write a function which returns a two column data frame where each row is a variable, the first column of the data frame is the name of each variable, and the second column of the data frame is the number of times that the column is NA. Test your function. Then, report the results applied to the parking tickets data frame. There are several ways to do this, but we haven't covered them yet in class, so you will need to work independently to set this up.

```
# adding typing to the function signature for clarity
# and to demonstrate how to use type hints
from pandas import DataFrame, Series
def get_number_of_NAs(df: DataFrame) -> Series:
```

```
    """
    Takes in a pandas dataframe and returns a pandas series with the
    number of missing values in each column."""

    na_df = df.isna().sum()

    return na_df

# multiple ways to do this does not need to be a full unittest class
import numpy as np # used for nan values can use other methods
class TestGetNumberOfNAs(unittest.TestCase):
    def test_mixed_nan_values(self):
        data = {
            'A': [1, 2, np.nan, 4, 5],
            'B': [np.nan, 2, 3, np.nan, 5],
            'C': [1, 2, 3, 4, 5],
            'D': [np.nan, np.nan, np.nan, np.nan, np.nan]
        }
        df = pd.DataFrame(data)
        result = get_number_of_NAs(df)
        expected = pd.Series({'A': 1, 'B': 2, 'C': 0, 'D': 5})
        pd.testing.assert_series_equal(result, expected)

    def test_empty_dataframe(self):
        empty_df = pd.DataFrame()
        result = get_number_of_NAs(empty_df)
        self.assertTrue(result.empty)

    def test_no_nan_values(self):
        no_nan_df = pd.DataFrame({'X': [1, 2, 3], 'Y': [4, 5, 6]})
        result = get_number_of_NAs(no_nan_df)
        expected = pd.Series({'X': 0, 'Y': 0})
        pd.testing.assert_series_equal(result, expected)

# Run the tests
test_suite = unittest.TestLoader().loadTestsFromTestCase(TestGetNumberOfNAs)
test_runner = unittest.TextTestRunner(verbosity=2)
test_result = test_runner.run(test_suite)

# Print summary
print(f"Failures: {len(test_result.failures)}")
print(f"Errors: {len(test_result.errors)}")
```

```
test_empty_dataframe (__main__.TestGetNumberOfNAs.test_empty_dataframe) ...
ok
test_mixed_nan_values (__main__.TestGetNumberOfNAs.test_mixed_nan_values) ...
ok
test_no_nan_values (__main__.TestGetNumberOfNAs.test_no_nan_values) ... ok

----------------------------------------------------------------------
Ran 3 tests in 0.004s

OK

Failures: 0
Errors: 0
```

```
# Run the function on the parking tickets data
na_df = get_number_of_NAs(df)
print("Number of missing values in each column:")
print(na_df)
```

```
Number of missing values in each column:
Unnamed: 0                   0
ticket_number                0
issue_date                   0
violation_location           0
license_plate_number         0
license_plate_state         97
license_plate_type        2054
zipcode                  54115
violation_code               0
violation_description        0
unit                        29
unit_description             0
vehicle_make                 0
fine_level1_amount           0
fine_level2_amount           0
current_amount_due           0
total_payments               0
ticket_queue                 0
ticket_queue_date            0
notice_level             84068
hearing_disposition     259899
notice_number                0
officer                      0
```

```
address                       0
dtype: int64
```

2. Three variables are missing much more frequently than the others. Why? (Hint: look at some rows and read the data dictionary written by ProPublica)

- **Solution**: The three columns with the most missing values are `zipcode`, `hearing_disposition`, and `notice_level`. ZIP is matched to a car registration and so is missing if there is no car regestration. Notice level is messing if no notice was sent and hearing disposition is missing if there was no hearing

3. Some of the other articles on the propublica website discuss an increase in the dollar amount of the ticket for not having a city sticker. What was the old violation code and what is the new violation code?

```
# Keep only the rows with violation descriptions that contain 'STICKER'
sticker_violations = df[df['violation_description'].str.contains('STICKER')]
print("All descriptions containing 'STICKER':")
print(sticker_violations['violation_description'].unique())

# drop rows for over 16,000 pounds
sticker_violations =
 ↪  sticker_violations[~sticker_violations['violation_description'].str.contains('OVER
 ↪  16,000 LBS.')]

# drop rows for improper display of city sticker
sticker_violations =
 ↪  sticker_violations[~sticker_violations['violation_description'].str.contains('IMPROPER
 ↪  DISPLAY OF CITY STICKER')]

print("Remaining descriptions containing 'STICKER':")
print(sticker_violations['violation_description'].unique())

grouped = sticker_violations.groupby(['violation_description',
 ↪  'violation_code']).size().reset_index(name='n')
print("Comparing violation description and violation code:")
print(grouped)

# take the second index because the first is just the start of the data
date_of_change =
 ↪  sticker_violations.groupby('violation_description')['issue_date'].min().iloc[1]
```

```
print("Date the fine changed:")
print(date_of_change)
```

```
All descriptions containing 'STICKER':
['NO CITY STICKER OR IMPROPER DISPLAY'
 'NO CITY STICKER VEHICLE UNDER/EQUAL TO 16,000 LBS.'
 'NO CITY STICKER VEHICLE OVER 16,000 LBS.'
 'IMPROPER DISPLAY OF CITY STICKER']
Remaining descriptions containing 'STICKER':
['NO CITY STICKER OR IMPROPER DISPLAY'
 'NO CITY STICKER VEHICLE UNDER/EQUAL TO 16,000 LBS.']
Comparing violation description and violation code:
                              violation_description violation_code      n
0                NO CITY STICKER OR IMPROPER DISPLAY        0964125  10758
1                NO CITY STICKER OR IMPROPER DISPLAY        0976170     15
2  NO CITY STICKER VEHICLE UNDER/EQUAL TO 16,000 ...       0964125B  14246
Date the fine changed:
2012-02-25 02:00:00
```

- **Solution:** From the output above, we can see the three violation codes that are associated with not having a city sticker. While there are three, one has very few observations and so the focus of this question is on the other two. We can see that the code changed if Feb 2012 from "0964125" to "0964125B". We will also give you full credit if you discuss the very rare codes "0964125C" and "0964125D".

4. How much was the cost of an initial offense under each code? (You can ignore the ticket for a missing city sticker on vehicles over 16,000 pounds.)

```
# use output from table above
sticker_codes = ["0964125", "0976170", "0964125B"]

# get fine amount for first violation by violation code
grouped = (

 ↪  sticker_violations[sticker_violations["violation_code"].isin(sticker_codes)]
    .groupby('violation_code')['fine_level1_amount']
    .mean()
)
print("Fine amount by code:")
print(grouped)
```

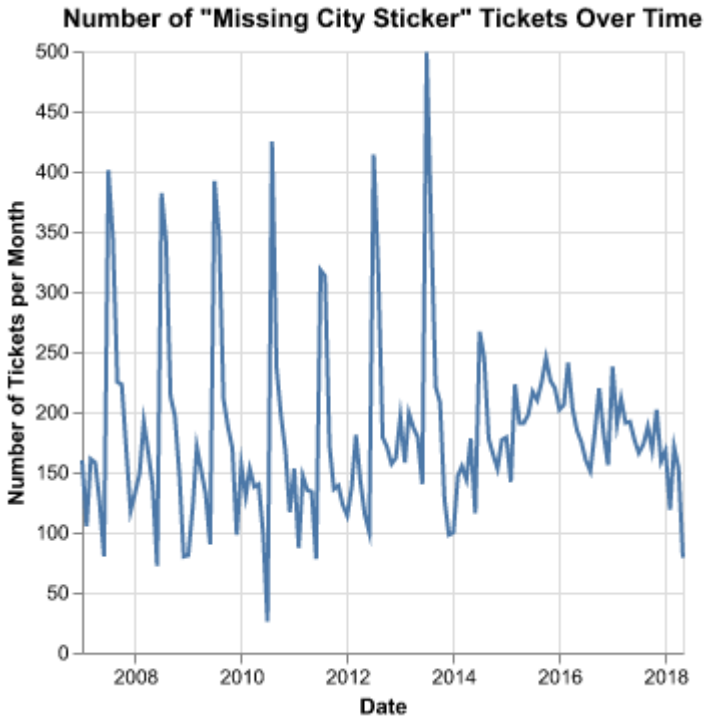
```
Fine amount by code:
violation_code
0964125     120.0
0964125B    200.0
0976170     120.0
Name: fine_level1_amount, dtype: float64
```

- **Solution:** The city switched codes which increased the fine from $120 to $200 for a first offense of not having a city sticker.

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

1. Using pandas, create a new value for violation codes which combines the two codes that you found in the previous question. Again using pandas, collapse the data to capture the number of missing city sticker tickets by month. Then, using Altair, plot the number of tickets over time.

```
# defined above but also here for clarity
sticker_codes = ["0964125", "0976170", "0964125B"]

# create new violation_code variable with all the sticker codes being the
 ↪  same
df.loc[df['violation_code'].isin(sticker_codes), 'new_violation_code'] =
 ↪  "0976170"
df.loc[~df['violation_code'].isin(sticker_codes), 'new_violation_code'] =
 ↪  df['violation_code']

# Aggregate data by month
df['yearmonth'] = df['issue_date'].dt.to_period('M').astype(str)
df_monthly = df[df['new_violation_code'] ==
 ↪  "0976170"].groupby('yearmonth').size().reset_index(name='count')


# Create the plot
chart = alt.Chart(df_monthly).mark_line().encode(
    x=alt.X('yearmonth:T', title='Date'),
    y=alt.Y('count:Q', title='Number of Tickets per Month')
).properties(
    title='Number of "Missing City Sticker" Tickets Over Time'
)

chart
```



2. Suppose that your reader wants to be able to use the plot to deduce when the price increase occurred. Add frequent or custom date labels on the x-axis of your plot such that the date of the price increase is readily apparent. We haven't covered Altair's date labeling features in class so you'll first need to find the relevant help page in the documentation. Which help page did you use?

```
# Convert month_of_change to a string in the format that matches your data
df_monthly['yearmonth'] = pd.to_datetime(df_monthly['yearmonth'],
 ↪  format='%Y-%m')
df_monthly['yearmonth'] =
 ↪  df_monthly['yearmonth'].dt.to_period('M').astype(str)
month_of_change_str = date_of_change.strftime('%Y-%m')

# Create the annotation
annotation = alt.Chart(df_monthly[df_monthly['yearmonth'] ==
 ↪  month_of_change_str]).mark_text(
    align='left',
    baseline='bottom',
    dy= 50,
    dx=5,
    fontSize=14,
```

```
).encode(
    x=alt.X('yearmonth:T'),
    y=alt.Y('count:Q'),
    text=alt.value('Fine Increase'),
    color = alt.value('red')
)

# Create the vertical line
vertical_line = alt.Chart(pd.DataFrame({'yearmonth':
 ↪  [month_of_change_str]})).mark_rule(
    color='red',
    strokeWidth=2
).encode(
    x='yearmonth:T'
)

# Combine the chart, annotation, and vertical line
final_chart = chart + annotation + vertical_line

final_chart
```

- **Solution:** The [**Text**]**{https://altair-viz.github.io/user_guide/marks/text.html}** page on the documentation is useful

3. The City Clerk said the price increase would raise revenue by $16 million per year. For now, ignore the fact that many tickets are not paid and assume that the number of tickets issued is the same before and after the policy change. Using only the data available in the calendar year prior to the increase, how much of a revenue increase should they have projected? Remember that you are working with a one percent sample of the data.

Assume that the number of tickets of this type issued afterward would be constant and you can assume that there are no late fees or collection fees, so a ticket is either paid at its face value or is never paid.

```
# get the number of sticker tickets given in 2011
sticker_ticket_count_2011= (df
    .loc[df['issue_date'].dt.year == 2011]
    .loc[df['new_violation_code'] == '0976170']
    ['ticket_number'].count()
)
# remember to mulitply by 100 to get full sample
print(f"Number of sticker tickets in 2011: {sticker_ticket_count_2011 *
 ↪  100:,.0f}")

print(f"Naive Estimate for Increased Revenue: ${(sticker_ticket_count_2011 *
 ↪  100 * 80):,.2f}")
```

```
Number of sticker tickets in 2011: 193,500
Naive Estimate for Increased Revenue: $15,480,000.00
```

- **Solution:** There were a bit under 200k sticker tickets issued in 2011. Increasing the ticket cost from 120 to 200 would have increased revenue by about $15.5M which is close to $16M.

4. What happened to repayment rates (percentage of tickets issued that had payments made) on this type of ticket in the calendar year after the price increase went into effect? Suppose for a moment that the number of tickets issued was unchanged after the price increase. Using the new repayment rates in the year after the price increase occurred, what would the change in revenue have been?

```
df_2011 = (
    df
    .loc[df['issue_date'].dt.year == 2011]
    .loc[df['new_violation_code'] == '0976170']
```

```
)
df_2013 = (
    df
    .loc[df['issue_date'].dt.year == 2013]
    .loc[df['new_violation_code'] == '0976170']
)

repayment_rate_2011 = df_2011[df_2011['ticket_queue'] == 'Paid'].shape[0] /
 ↪  len(df_2011)
repayment_rate_2013 = df_2013[df_2013['ticket_queue'] == 'Paid'].shape[0] /
 ↪  len(df_2013)

print(f"Repayment rate in 2011: {round(repayment_rate_2011, 2)}")
print(f"Repayment rate in 2013: {round(repayment_rate_2013, 2)}")

# Assuming the number of tickets issued was unchanged
naive_revenue_2011 = sticker_ticket_count_2011 * 100 * 120 *
 ↪  repayment_rate_2011
naive_revenue_2013 = sticker_ticket_count_2011 * 100 * 200 *
 ↪  repayment_rate_2013

change_in_revenue = naive_revenue_2013 - naive_revenue_2011

print(f"Change in revenue: ${change_in_revenue:,.0f}")
```

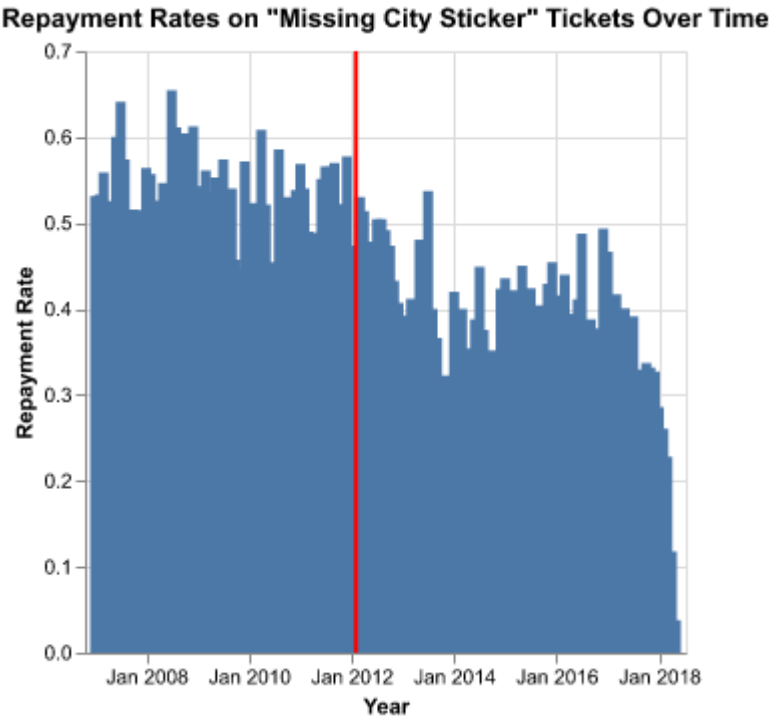
```
Repayment rate in 2011: 0.54
Repayment rate in 2013: 0.41
Change in revenue: $3,181,155
```

- **Solution:** After the new rule went into effect, the repayment rate for these tickets fropped from 54% to 41%. If we recalcuate the increase in revenue taking into account the fact that not all tickets are paid, we find that revenue increases by about $6M.

5. Make a plot with the repayment rates on "missing city sticker" tickets and a vertical line at when the new policy was introduced. Interpret.

```
# calculate repayment rates by year

repayment_rates = (df
    [df['new_violation_code'] == '0976170']
    .assign(ticket_paied = df['ticket_queue'] == 'Paid')
    .groupby(df['yearmonth'])
```

```
    .agg(repayment_rate=('ticket_paied', 'mean'))
    .reset_index()
)

repayment_rates['yearmonth'] = pd.to_datetime(repayment_rates['yearmonth'],
 ↪  format='%Y-%m')

# make a bar chart of repayment rates by year and add a vertical line at the
 ↪  year of the policy change
chart = alt.Chart(repayment_rates).mark_bar().encode(
    alt.X('yearmonth:T', title='Year'),
    alt.Y('repayment_rate:Q', title='Repayment Rate'),
).properties(
    title='Repayment Rates on "Missing City Sticker" Tickets Over Time'
)

vertical_line = alt.Chart(pd.DataFrame({'date':
 ↪  [date_of_change]})).mark_rule(
    color='red',
    strokeWidth=2
).encode(
    alt.X('yearmonth(date):T')
)

chart + vertical_line
```

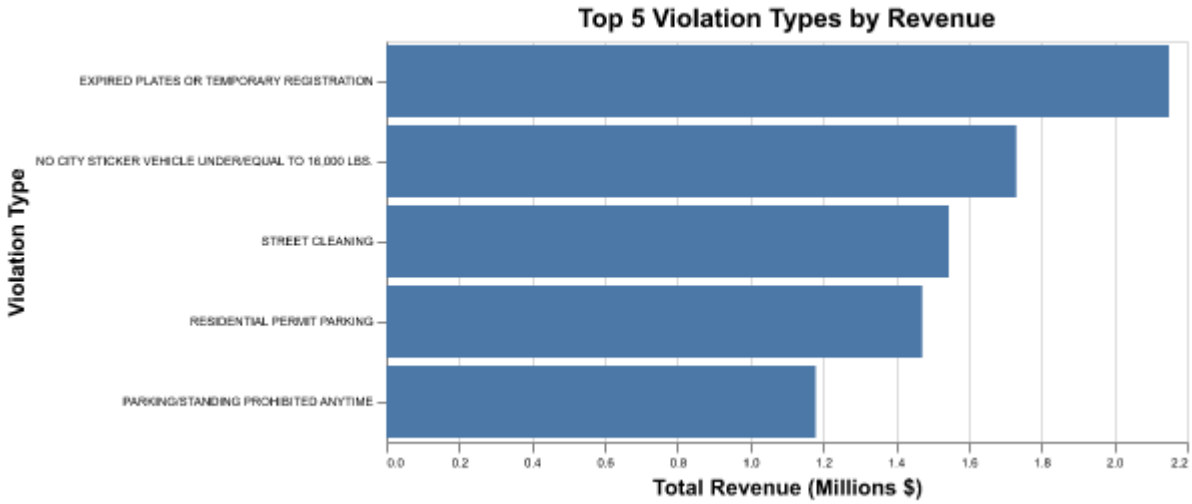


- **Solution:** Repayment rates were falling a bit over time but appear to have fallen more sharply during the two years after the price increase.

6. Suppose that the City Clerk were committed to getting more revenue from tickets. What three violation types would you as an analyst have recommended they increase the price of? Consider both the number of tickets issued for each violation type and the repayment rate for each violation type. You may assume there is no behavioral response to price changes (ie. people continue to commit violations at the same rate and repay at the same rate). Make a plot to support your argument and explain in writing why it supports your argument.

```
# get revenue generated per year for each violation code
revenue_by_code = (df
    [df['ticket_queue'] == 'Paid']
    .groupby(['new_violation_code', 'violation_description'])
    .agg(revenue=('total_payments', 'sum'))
    .sort_values('revenue', ascending=False)
    .reset_index()
    .head(5)
)

# Create a new column 'revenue_millions' which is 'revenue' divided by
 ↪ 1,000,000
```

```
revenue_by_code['revenue_millions'] = revenue_by_code['revenue'] / 1_000_000

# Create the horizontal bar chart
chart = alt.Chart(revenue_by_code).mark_bar().encode(
    y=alt.Y('violation_description:N', title='Violation Type', sort='-x'),
    x=alt.X('revenue_millions:Q', title='Total Revenue (Millions $)')
).properties(
    title='Top 5 Violation Types by Revenue',
    width=400,
    height=200
).configure_axis( # Configure the axis so that the labels are not cut off
    labelFontSize=6,
    labelLimit=400
)

chart
```



- **Solution:** The plot shows the five types of tickets that generate the most revenue. The top three are for expired plates, no stickers, and no street cleaning. These revenue measures are a function of both the number of tickets issued and the repayment rate. Therefore, if we assume repayment rate and the number of of tickets issued are constant, then we would expect the revenue generated by these tickets to increase the most if the price of the tickets were increased.

Headlines and sub-messages (20 points)

1. The City Clerk has now begun to wonder… maybe raising ticket prices will lead to a decline in repayment rates after all. Make a data frame where each row is a violation description, the fraction of time that the ticket is paid, and the average level 1 fine. Sort this dataframe based on how many total tickets of each type have been issued. Print the rows for the 5 most common violation descriptions.

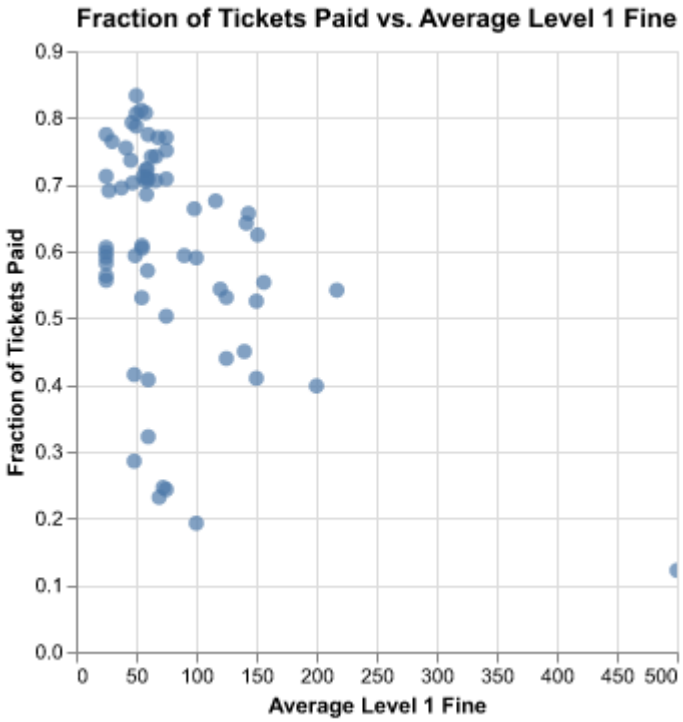
```
repayment_rate_by_code = (df
    .assign(ticket_paid = df['ticket_queue'] == 'Paid')
    .groupby(['violation_description'])
    .agg(fraction_paid=('ticket_paid', 'mean'),
        average_level1_fine=('fine_level1_amount', 'mean'),
        n_tickets=('ticket_number', 'count'))
    .sort_values('n_tickets', ascending=False)
    .reset_index()
)
print(repayment_rate_by_code.head(5))
```

```
                    violation_description  fraction_paid  \
0  EXPIRED PLATES OR TEMPORARY REGISTRATION       0.604361
1                           STREET CLEANING       0.811612
2              RESIDENTIAL PERMIT PARKING       0.742262
3  EXP. METER NON-CENTRAL BUSINESS DISTRICT       0.792913
4       PARKING/STANDING PROHIBITED ANYTIME       0.705817

   average_level1_fine  n_tickets
0            54.968869      44811
1            54.004249      28712
2            66.338302      23683
3            46.598058      20600
4            66.142864      19753
```

2. Make a scatter plot which shows the relationship between fine amount and the fraction of tickets that are paid. Focus only on violations that appear at least 100 times. There will be one outlier with a high fine and you can exclude that ticket type from the plot. Then make two other plots which show the same relationship in different ways. For all three plots, write out what are the headlines and what are sub-messages.

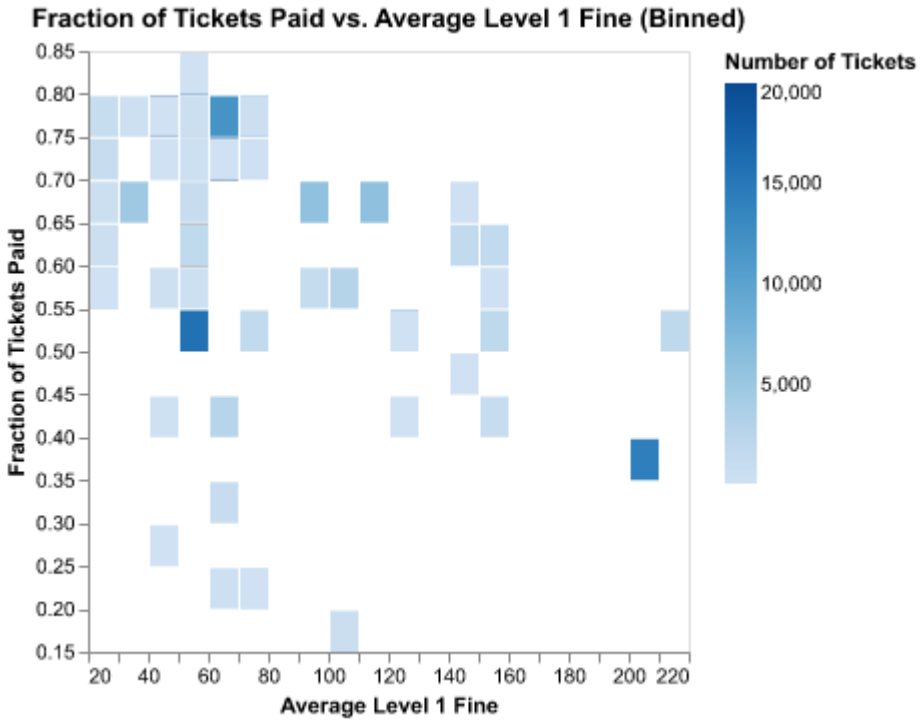
```
# filter to only include violation descriptions that appear at least 100
 ↪  times
filtered_result = repayment_rate_by_code[repayment_rate_by_code['n_tickets']
 ↪  >= 100]
```

```
# Create a scatter plot of the fraction of tickets paid versus the fine level
scatter_plot = alt.Chart(filtered_result).mark_circle(size=60).encode(
    x=alt.X('average_level1_fine', title='Average Level 1 Fine'),
    y=alt.Y('fraction_paid', title='Fraction of Tickets Paid'),
    tooltip=['violation_description', 'n_tickets', 'fraction_paid',
 ↪  'average_level1_fine']
).properties(
    title='Fraction of Tickets Paid vs. Average Level 1 Fine'
).interactive()

scatter_plot
```



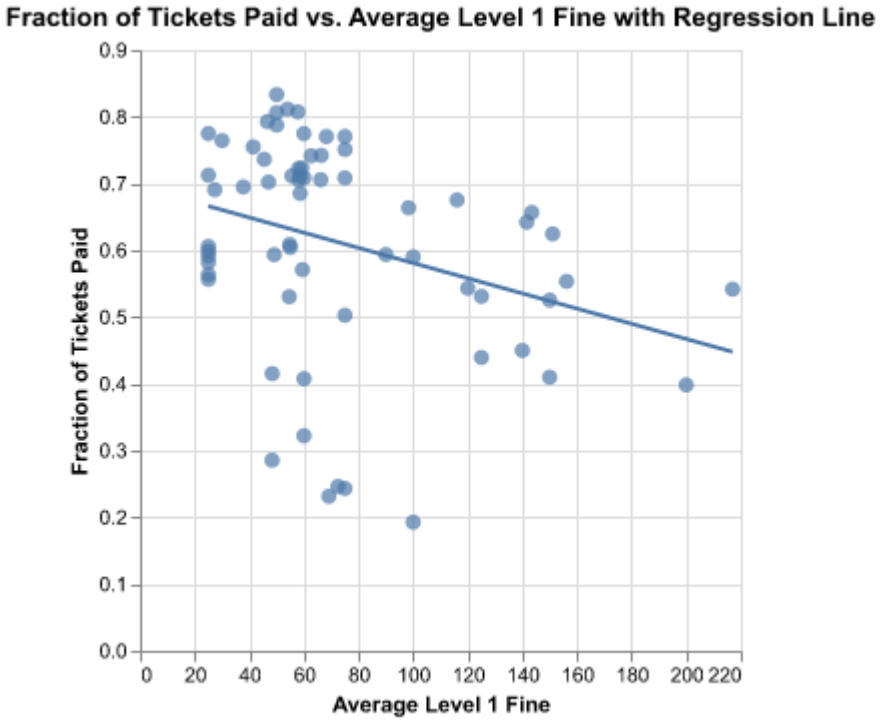

```
# drop the outlier with fine over 400
filtered_result = filtered_result[filtered_result['average_level1_fine'] <
 ↪  400]

binned_scatter_plot = alt.Chart(filtered_result).mark_bar().encode(
    alt.X('average_level1_fine:Q', title='Average Level 1 Fine',
 ↪  bin=alt.Bin(maxbins=20)),
```

```
    alt.Y('fraction_paid:Q', title='Fraction of Tickets Paid',
 ↪  bin=alt.Bin(maxbins=20)),
    alt.Color('n_tickets:Q', title='Number of Tickets',
        scale=alt.Scale(domain=[100, 20000])),
    tooltip=['violation_description', 'n_tickets', 'fraction_paid',
 ↪  'average_level1_fine']
).properties(
    title='Fraction of Tickets Paid vs. Average Level 1 Fine (Binned)'
).interactive()

binned_scatter_plot
```




```
# Create geom_smooth type plot

# Create a scatter plot of the fraction of tickets paid versus the fine level
 ↪  with a regression line
scatter_plot_with_regression =
 ↪  alt.Chart(filtered_result).mark_circle(size=60).encode(
    x=alt.X('average_level1_fine', title='Average Level 1 Fine'),
    y=alt.Y('fraction_paid', title='Fraction of Tickets Paid'),
```

```
    tooltip=['violation_description', 'n_tickets', 'fraction_paid',
 ↪  'average_level1_fine']
).properties(
    title='Fraction of Tickets Paid vs. Average Level 1 Fine with Regression
 ↪  Line'
).interactive()

scatter_plot_with_regression +
 ↪  scatter_plot_with_regression.transform_regression('average_level1_fine',
 ↪  'fraction_paid').mark_line()
```



3. The City Clerk doesn't understand regressions and only has time to look at one plot. Which plot are you going to bring to them and why?

- **Solution:** The binned scatter plot does the best job of showing the loose negative relationship between fine size and repayment rate. Unlike the regression plot, it also gives a sense of which points are the most important (the ones with the most tickets issued) instead of treating all violation types equally. (note this the answer to this question is subjective and other thoughtout answers are possible)

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

4. Most violation types double in price if unpaid.
    - Does this hold for all violations?
    - If not, find all violations with at least 100 citations that do not double. How much does each ticket increase if unpaid?

```
# create a data frame that is the realtive increase in fine if unpaid
increase_in_fine = (df
    .assign(increase_in_fine = df['fine_level2_amount'] /
 ↪  df['fine_level1_amount'])
    .groupby('violation_description')
    .agg(increase_in_fine=('increase_in_fine', 'mean'),
        n_tickets=('ticket_number', 'count'))
    .reset_index()
)
# get number that double
double_fine = increase_in_fine[increase_in_fine['increase_in_fine'] ==
 ↪  2].shape[0]

# get number that do not double
not_double_fine = increase_in_fine[increase_in_fine['increase_in_fine'] !=
 ↪  2].shape[0]

print(f"Number of violations that double: {double_fine}")
print(f"Number of violations that do not double: {not_double_fine}")

not_doubled_violations = (increase_in_fine
    .loc[increase_in_fine['increase_in_fine'] != 2]
    .loc[increase_in_fine['n_tickets'] >= 100]
    .sort_values("increase_in_fine", ascending=False)
    .reset_index()
)
print("Violations that do not double with >100 tickets:")
print(not_doubled_violations)
```

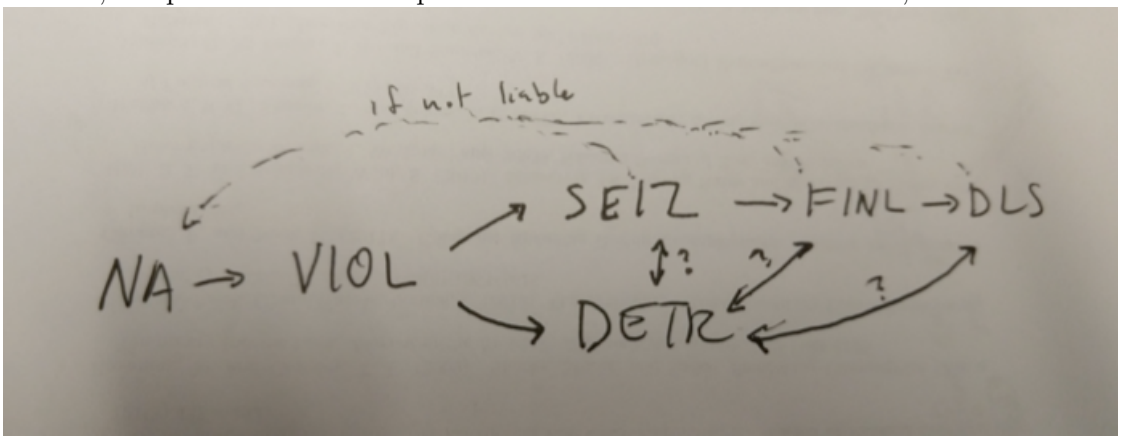
```
Number of violations that double: 109
Number of violations that do not double: 10
Violations that do not double with >100 tickets:
   index                    violation_description  increase_in_fine  \
0     79                PARK/STAND ON BICYCLE PATH          1.944915
```

```
1     42  NO CITY STICKER VEHICLE OVER 16,000 LBS.          1.910687
2      5      BLOCK ACCESS/ALLEY/DRIVEWAY/FIRELANE          1.890437
3     62                          PARK OR BLOCK ALLEY       1.732846
4     54   OBSTRUCTED OR IMPROPERLY TINTED WINDOWS          1.653137
5     95     SMOKED/TINTED WINDOWS PARKED/STANDING          1.629346
6     15                       DISABLED PARKING ZONE        1.621681
```

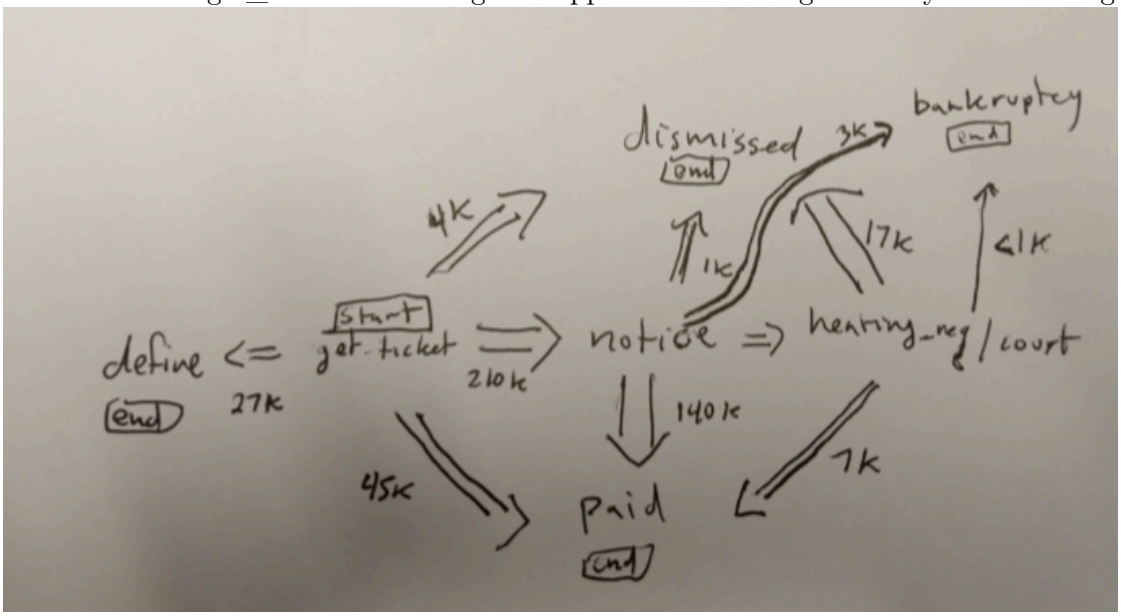
```
   n_tickets
0        236
1        131
2       1579
3       2050
4        271
5       1697
6       2034
```

5. Many datasets implicitly contain information about how a case can progress. Draw a diagram explaining the process of moving between the different values of `notice_level` (if you draw it on paper, take a picture and include the image in your write up). Draw a second diagram explaining the different values of `ticket_queue`. If someone contests their ticket and is found not liable, what happens to `notice_level` and to `ticket_queue`? Include this in your tree drawings above.

- **Solution:** The progression through notice_level is obvious for the first two steps and then it becomes less clear how cases progress. We have tools at our disposal however. First we can look at counts. For example, if we make an assumption that at each step of the process some people pay, we can roughly order the levels based on how frequent they are. Further we can make a crosstab with ticket_queue; This gives a sense of how long people remain in a level. Particularly, looking at 'Notice' is helpful, since it represents a phase that is not final (as opposed to 'Paid' or 'Dismissed'). Nearly half of 'SEIZ' are on 'Notice', compared to less than 5 percent of 'DETR'. With these in mind, I made this tree:



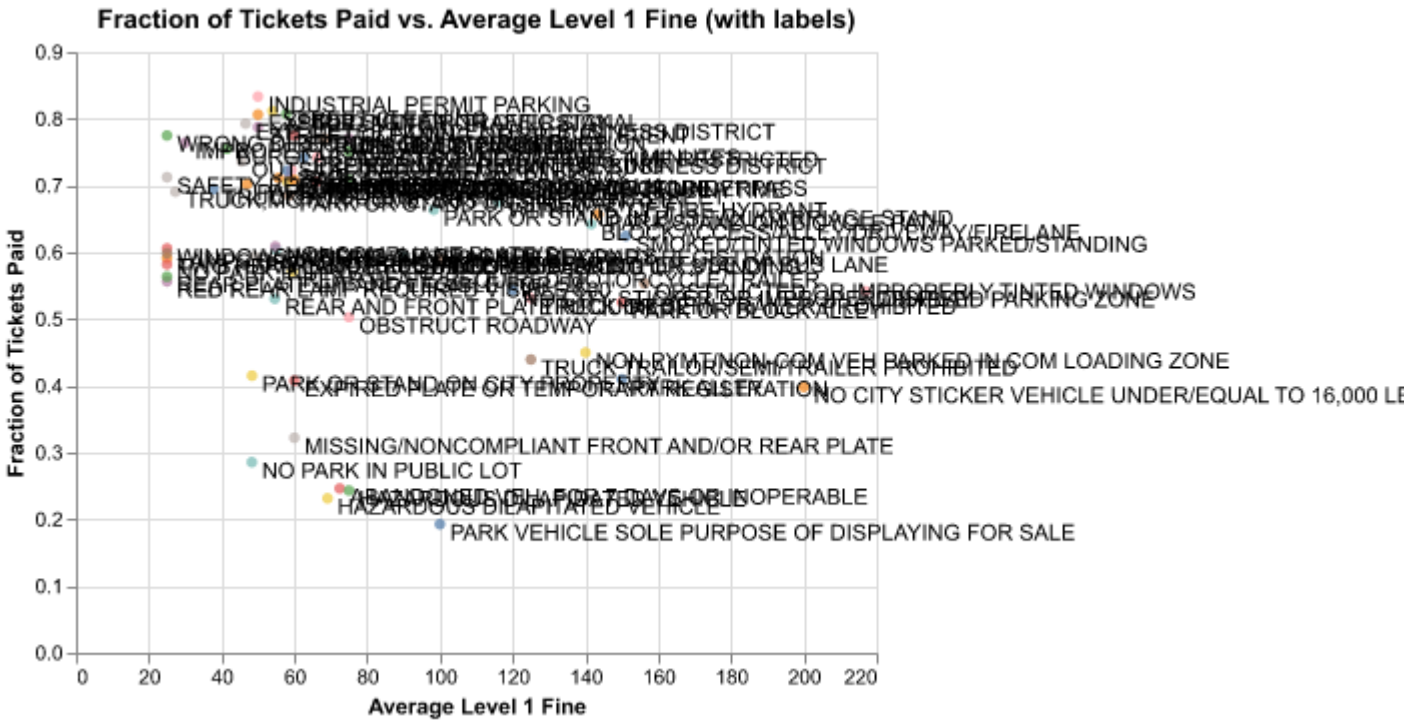
- **Solution:** The ticket_queue provides a more satisfactory story. I guessed at the size of the flows from one value to another using the context from hearing_disposition and notice_level. There are people who skip some steps (e.g. 8 people in the subsample go directly to bankruptcy), but these are exceptions to the normal progressions through the system. As a correction, I should have an arrow from get_ticket to hearing. It appears about 6k go directly to a hearing.



6. Go back to your scatter plot from the previous section. We want to add labels to each dot (which conveniently you constructed in the previous step). Implement this in two ways: (a) label every dot with adjacent text or (b) put the text in a legend. Either way, you will find the same problem – there are too many labels and the plot is illegible. Revise the plots. First, do this the easy way, which is to pick the ten most commonly used violation descriptions and mark all the other dots as "Other". Second, for (b), try to construct meaningful categories by marking violation descriptions which sound similar with a common label and a common color.

```
# (a) Scatter plot with adjacent text labels for all dots
plot_a = alt.Chart(filtered_result).mark_circle().encode(
    x=alt.X('average_level1_fine', title='Average Level 1 Fine'),
    y=alt.Y('fraction_paid', title='Fraction of Tickets Paid'),
    color=alt.Color('violation_description:N', legend=None),
    tooltip=['violation_description', 'n_tickets', 'fraction_paid',
 ↪  'average_level1_fine']
)

text_a = alt.Chart(filtered_result).mark_text(align='left', dx=5,
 ↪  dy=5).encode(
```

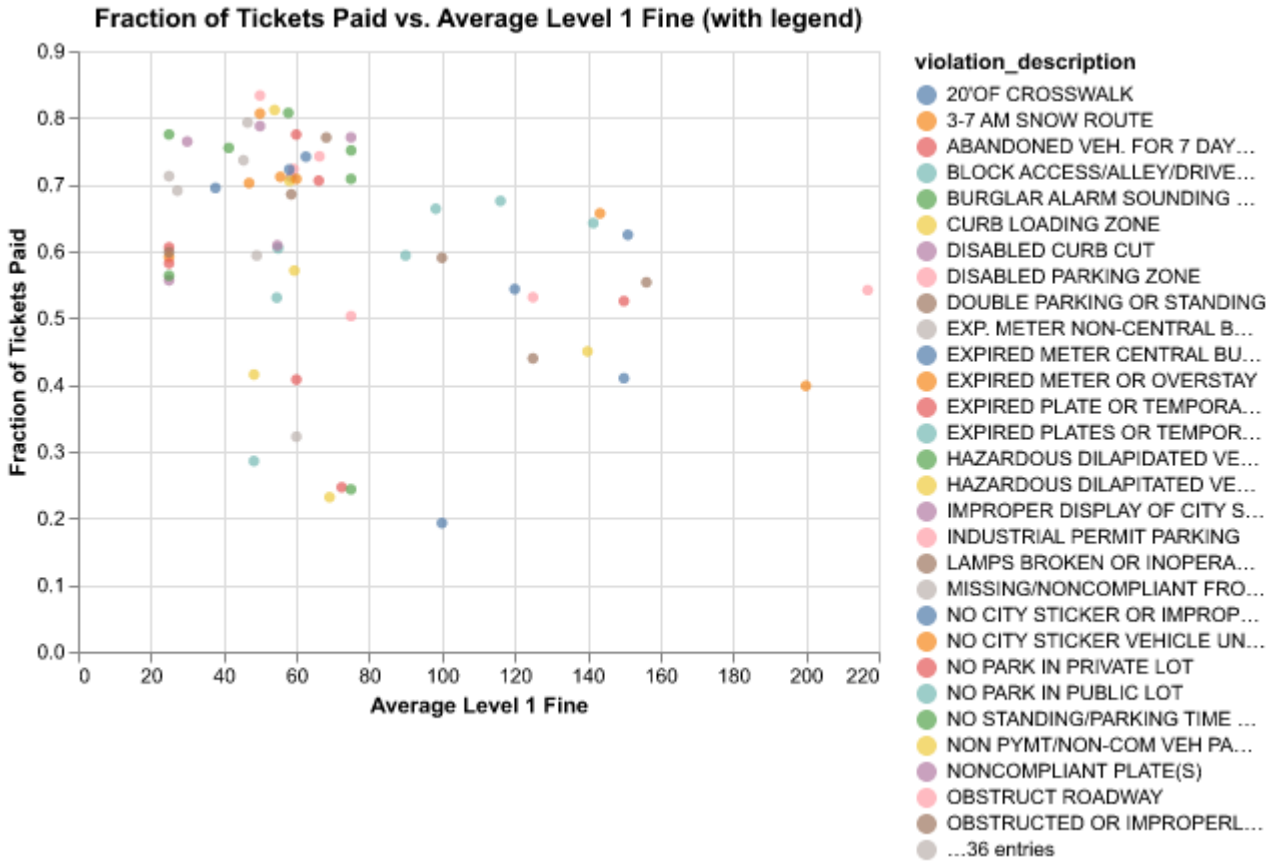
```
    x='average_level1_fine',
    y='fraction_paid',
    text='violation_description'
)

chart_a = (plot_a + text_a).properties(
    width=400,
    height=300,
    title='Fraction of Tickets Paid vs. Average Level 1 Fine (with labels)'
)

chart_a
```



Solution: Unreadable!

```
# (b) Scatter plot with legend for all dots
chart_b = alt.Chart(filtered_result).mark_circle().encode(
    x=alt.X('average_level1_fine', title='Average Level 1 Fine'),
    y=alt.Y('fraction_paid', title='Fraction of Tickets Paid'),
    color=alt.Color('violation_description:N'),
    tooltip=['violation_description', 'n_tickets', 'fraction_paid',
↪   'average_level1_fine']
```

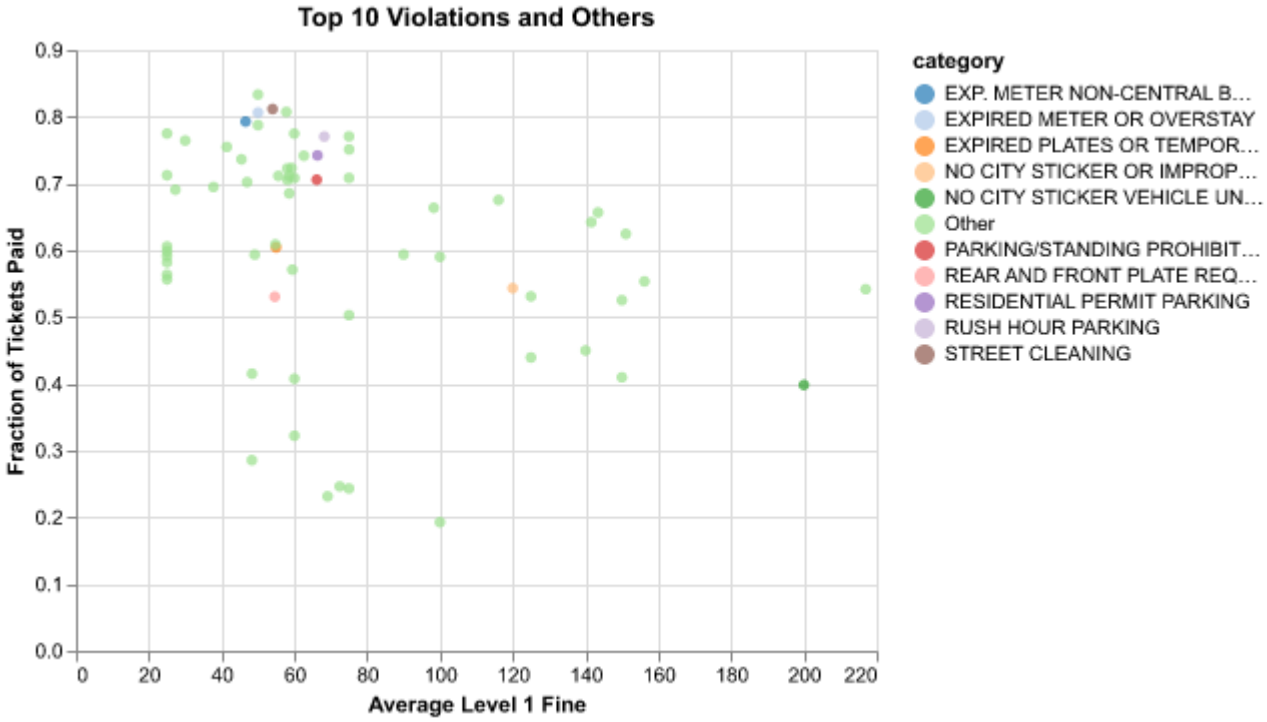
```
).properties(
    width=400,
    height=300,
    title='Fraction of Tickets Paid vs. Average Level 1 Fine (with legend)'
)

chart_b
```



- **Solution:** Better but still not useful!

```
# 1. Easy way: Pick the ten most commonly used violation descriptions
top_10_violations = filtered_result.nlargest(10,
 ↪  'n_tickets')['violation_description'].tolist()
filtered_result['category'] = 'Other'
filtered_result.loc[filtered_result['violation_description'].isin(top_10_violations),
 ↪  'category'] = filtered_result['violation_description']
```

```
chart_top_10 = alt.Chart(filtered_result).mark_circle().encode(
    x=alt.X('average_level1_fine', title='Average Level 1 Fine'),
    y=alt.Y('fraction_paid', title='Fraction of Tickets Paid'),
    color=alt.Color('category', scale=alt.Scale(scheme='category20')),
    tooltip=['violation_description', 'n_tickets', 'fraction_paid',
 ↪  'average_level1_fine']
).properties(
    width=400,
    height=300,
    title='Top 10 Violations and Others'
)

chart_top_10
```

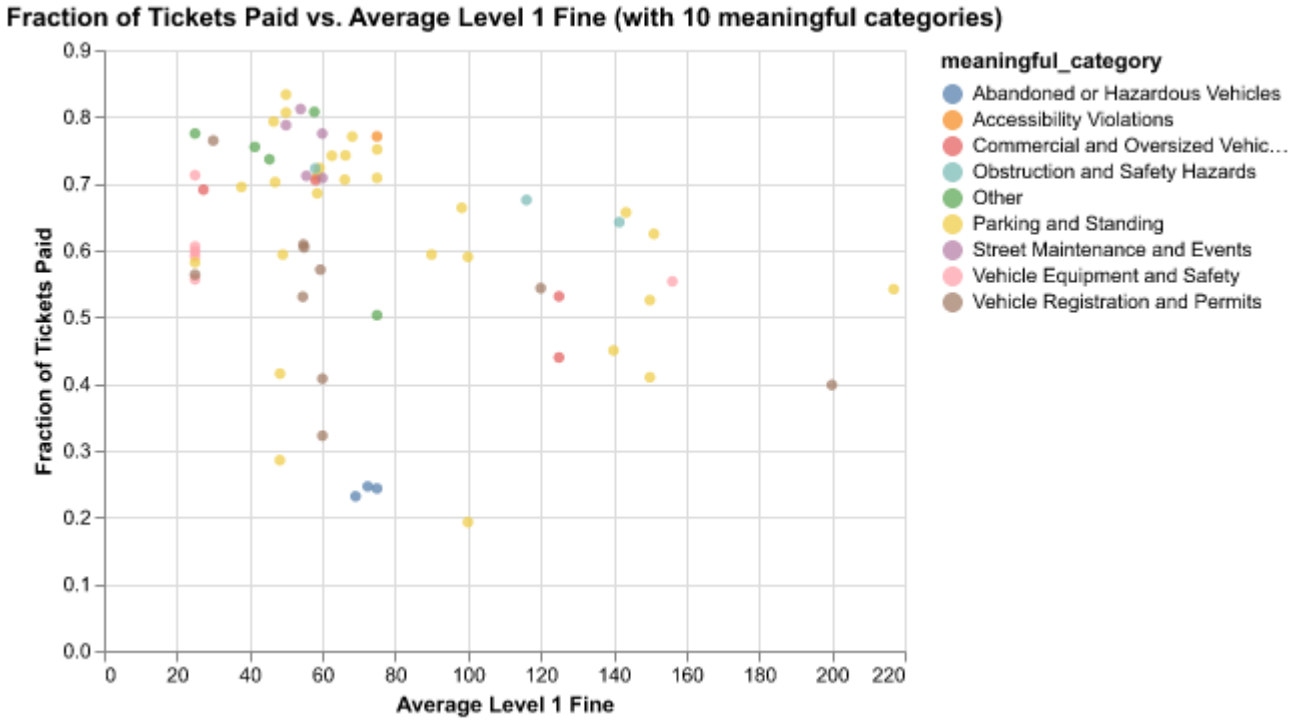


- **Solution:** Better but too many dots labeled "Other" to be super useful!

```
def categorize_violation(description):
    if any(word in description for word in ['PARK', 'STAND', 'METER',
     ↪  'RESIDENTIAL', 'INDUSTRIAL']):
```

```
        return 'Parking and Standing'
    elif 'STREET CLEAN' in description or 'SNOW ROUTE' in description or
     ↪  'SPECIAL EVENT' in description:
        return 'Street Maintenance and Events'
    elif any(word in description for word in ['PLATE', 'REGISTRATION',
     ↪  'STICKER', 'DISPLAY', 'EXPIRED']):
        return 'Vehicle Registration and Permits'
    elif 'DISABLED' in description or 'HANDICAP' in description:
        return 'Accessibility Violations'
    elif any(word in description for word in ['LAMP', 'LIGHT', 'REFLECTOR',
     ↪  'WINDOW', 'MIRROR', 'BRAKE', 'HORN', 'MUFFLER', 'SAFETY BELT']):
        return 'Vehicle Equipment and Safety'
    elif 'ABANDONED' in description or 'HAZARDOUS' in description or
     ↪  'DILAPIDATED' in description:
        return 'Abandoned or Hazardous Vehicles'
    elif any(word in description for word in ['TRUCK', 'BUS', 'TAXI', 'RV',
     ↪  'TRAILER', 'LOAD']):
        return 'Commercial and Oversized Vehicles'
    elif 'ALLEY' in description or 'SIDEWALK' in description or 'CROSSWALK'
     ↪  in description or 'FIRE' in description or 'HYDRANT' in description:
        return 'Obstruction and Safety Hazards'
    elif any(word in description for word in ['SMOKE', 'FUME', 'ENGINE
     ↪  RUNNING']):
        return 'Environmental Violations'
    else:
        return 'Other'

filtered_result['meaningful_category'] =
 ↪  filtered_result['violation_description'].apply(categorize_violation)

chart_categories = alt.Chart(filtered_result).mark_circle().encode(
    x=alt.X('average_level1_fine', title='Average Level 1 Fine'),
    y=alt.Y('fraction_paid', title='Fraction of Tickets Paid'),
    color=alt.Color('meaningful_category',
 ↪  scale=alt.Scale(scheme='tableau10')),
    tooltip=['violation_description', 'n_tickets', 'fraction_paid',
 ↪  'average_level1_fine']
).properties(
    width=400,
    height=300,
    title='Fraction of Tickets Paid vs. Average Level 1 Fine (with 10
 ↪  meaningful categories)'
```

```
)

chart_categories
```



- **Solution:** Much better!

extra credit (max 5 points)

1. Which violation codes, if any, are associated with multiple violation descriptions? In these cases, using pandas, create a new column which records the most common violation description associated with this code. If there are any with multiple descriptions print the three codes with the most observations.

```
violation_code_description_count = (df
[['violation_code', 'violation_description']]
.drop_duplicates()
.groupby('violation_code')
.size()
.sort_values(ascending=False)
.reset_index(name='n_descriptions')
```

```
)

multiple_codes =
 ↪  violation_code_description_count[violation_code_description_count['n_descriptions']
 ↪  > 1]['violation_code']

# Create a new dataframe with the most common description for each code
most_common_description = (df
    .groupby(['violation_code', 'violation_description'])
    .size()
    .reset_index(name='count')
    .sort_values(['violation_code', 'count'], ascending=[True, False])
    .groupby('violation_code')
    .first()
    .reset_index()
    [['violation_code', 'violation_description']]
    .rename(columns={'violation_description': 'most_common_description'})
)

# Merge this information back into the original dataframe
df= df.merge(most_common_description, on='violation_code', how='left')


# Count how many codes have multiple descriptions
codes_with_multiple_descriptions =
 ↪  violation_code_description_count[violation_code_description_count['n_descriptions']
 ↪  > 1]
print(f"\nNumber of violation codes with multiple descriptions:
 ↪  {len(codes_with_multiple_descriptions)}")

# Filter the main dataframe to only include codes with multiple descriptions
df_multiple =
 ↪  df[df['violation_code'].isin(codes_with_multiple_descriptions['violation_code'])]

# Count the number of observations for each code
code_counts = df_multiple['violation_code'].value_counts().reset_index()
code_counts.columns = ['violation_code', 'count']

# Get the top 3 most frequent codes
top_3_codes = code_counts.head(3)

print(f"Top 3 codes with multiple descriptions:")
```

```
print(top_3_codes)
```

```
Number of violation codes with multiple descriptions: 8
Top 3 codes with multiple descriptions:
  violation_code  count
0       0964040B  32082
1       0976160A  16853
2       0976160B   3072
```

2. Above you made a diagram on paper to show how a case can progress. Although Vega-Lite cannot support tree layout plots like this, Vega can!. Make the digital version of your diagram using Vega. You can use the (online Vega console](https://vega.github.io/editor/#/custom/vega). Submit your `JSON` as a separate file and submit your plot alongside your pset as a `.png`.

- No solution provided