Problem Set 2

Eddie Andujar

2024-10-19

```
# the following code will turn off any Python warnings in your code output
import warnings
warnings.filterwarnings("ignore")
```

1. **PS2:** Due Sat Oct 19 at 5:00PM Central. Worth 50 points.

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

Steps to submit (5 points on PS1)

1. "This submission is my work alone and complies with the 30538 integrity policy." Add your initials to indicate your agreement: **EA**
2. "I have uploaded the names of anyone I worked with on the problem set **here**" **EA** (1 point)
3. Late coins used this pset: **1** Late coins left after submission: **2**
4. Knit your `ps2.qmd`
    - The PDF should not be more than 25 pages. Use `head()` and re-size figures when appropriate.
5. Submit to Gradescope (4 points)
6. Tag your submission in Gradescope

```
# Import libraries

import os
import pandas as pd
import time
import altair as alt

file_path = r"C:\Users\Eddie
 ↪  Andujar\Documents\GitHub\student30538\problem_sets\ps2\data\parking_tickets_one_percent.
```

```
#1.1

df = pd.read_csv(file_path)

na = df.isna().sum().sum()

na

def na_per_column(df):
    na_counts = df.isna().sum()
    na_df = pd.DataFrame({
        'Variable Name': na_counts.index,
        'NAs in Variable': na_counts.values
    })
    return na_df

print(na_per_column(df))
```

```
            Variable Name  NAs in Variable
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
```

1.2

Hearing disposition, notice level and zipcode have the most missing entries. Turning to the ProPublica data dictionary, we can see why. Hearing disposition and notice level are fields that are left blank when there is no disposition or when no notice was sent, respectively. Likely, there are also cases where they cannot track the zipcode associated with the vehicle registration.

1.3

First, we can obtain a list of all codes related to stickers

```
filtered_violations = df[df['violation_description'].str.contains(
    'STICKER')]

code_list = filtered_violations['violation_code'].unique()

# Convert to a list (optional)
codes = code_list.tolist()

print(codes)
```

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

With the narrowed-down list, we can see that there are two separate codes (ignoring letter sub-categories):

The old code: 0976170 and The new code: 0964125

1.4

```
# Extract the fine amounts at both levels for relevant codes

violation_costs =
 ↪  filtered_violations[filtered_violations['violation_code'].isin(
    ['0964125', '0964125B', '976170'])][['issue_date', 'violation_code',
 ↪  'fine_level1_amount', 'fine_level2_amount']]

# Sort by their fine amounts
violation_costs_ascending = violation_costs.sort_values(
```

```
    by='fine_level2_amount')
print(violation_costs_ascending)
```

```
                issue_date violation_code  fine_level1_amount  \
14      2007-01-01 10:51:00        0964125                 120
92706   2010-04-27 07:59:00        0964125                 120
92707   2010-04-27 08:15:00        0964125                 120
92710   2010-04-27 08:36:00        0964125                 120
92711   2010-04-27 08:41:00        0964125                 120
...                     ...            ...                 ...
189108  2014-02-24 22:11:00       0964125B                 200
189109  2014-02-24 22:25:00       0964125B                 200
189116  2014-02-24 23:33:00       0964125B                 200
188994  2014-02-23 18:30:00       0964125B                 200
287452  2018-05-14 14:30:00       0964125B                 200

        fine_level2_amount
14                     240
92706                  240
92707                  240
92710                  240
92711                  240
...                    ...
189108                 400
189109                 400
189116                 400
188994                 400
287452                 400

[25004 rows x 4 columns]
```

Using the output, we see that the level one used to be 120 and is now 200 (consistent with the article). For fine level 2, we observe 240 as the initial cost, then it became 400.

2.1

```
codes_to_replace = ['0976170', '0964125', '0964125B']

# Replace specific violation codes with 'universal_sticker_code'
df['new_code'] = df['violation_code'].apply(
    lambda x: 'universal_sticker_code' if x in codes_to_replace else x
)
```

```
# Convert issue_date to datetime
df['issue_date'] = pd.to_datetime(df['issue_date'], errors='coerce')

# Sort by six month intervals for easier legibility
df['six_month_bin'] = df['issue_date'].dt.to_period('6M').dt.start_time

# Filter the data for 'universal_sticker_code' tickets
missing_tickets_df = df[df['new_code'] == 'universal_sticker_code']

# Group by the new six-month intervals
total_missing_tickets_by_bin =
 ↪  missing_tickets_df.groupby('six_month_bin').size().reset_index(name='count')

# Plot
chart = alt.Chart(total_missing_tickets_by_bin).mark_area(
    interpolate='step-after'
).encode(
    x=alt.X('six_month_bin:T', title='Issue Date',
 ↪  axis=alt.Axis(format='%Y-%m')),  # Format for six-month intervals
    y=alt.Y('count:Q', title='Missing City Sticker Tickets')
).properties(
    width=800  # Adjust width to fit the date labels
)

chart.display()
```

```
alt.Chart(...)
```

2.2

```
codes_to_replace = ['0976170', '0964125', '0964125B']

# Replace specific violation codes with 'universal_sticker_code'
df['new_code'] = df['violation_code'].apply(
    lambda x: 'universal_sticker_code' if x in codes_to_replace else x
)

# Convert issue_date to datetime
df['issue_date'] = pd.to_datetime(df['issue_date'], errors='coerce')

# Sort by six month intervals for easier legibility
```

```
df['six_month_bin'] = df['issue_date'].dt.to_period('6M').dt.start_time

# Filter the data for 'universal_sticker_code' tickets
missing_tickets_df = df[df['new_code'] == 'universal_sticker_code']

# Group by the new six-month intervals
total_missing_tickets_by_bin =
 ↪  missing_tickets_df.groupby('six_month_bin').size().reset_index(name='count')

# Plot
chart = alt.Chart(total_missing_tickets_by_bin).mark_area(
    interpolate='step-after'
).encode(
    x=alt.X('six_month_bin:T', title='Issue Date',
 ↪  axis=alt.Axis(format='%Y-%m')),  # Format for six-month intervals
    y=alt.Y('count:Q', title='Missing City Sticker Tickets')
).properties(
    width=800  # Adjust width to fit the date labels
)

highlight_2012 = alt.Chart(pd.DataFrame({'date':
 ↪  ['2012-01-01']})).mark_rule(color='red', strokeDash=[5,5]).encode(
    x='date:T'
)

final_chart = chart + highlight_2012

final_chart.display()

# For help, I started with this Altair template:

# https://altair-viz.github.io/gallery/bar_chart_with_mean_line.html

# I also asked ChatGPT for cleanup
```

```
alt.LayerChart(...)
```

2.3

```
#Filter by relevant year
missing_tickets_2011 = (
    missing_tickets_df[missing_tickets_df['issue_date'].dt.year == 2011])
```

```
missing_tickets_2011.shape[0]
```

```
1935
```

We can conclude that there were 1935 issued tickets in 2011

If we assume issued tickets are constant, we can use quantity and cost to calculate by doing the following:

```
# 1935 tickets times the price increase, then scaled up

projection = 1935*(200-120)*100

print(projection)
```

```
15480000
```

The city should have predicted a revenue of $15,480,000 based off these assumptions. Based on these assumptions, their projection wasn't far off.

2.4

```
# Filter for 'universal_sticker_code' tickets
sticker_tickets = df[df['new_code'] == 'universal_sticker_code']

#Focus on 2013
tickets_2013 = sticker_tickets[sticker_tickets['issue_date'].dt.year == 2013]

paid_2013 = tickets_2013[tickets_2013["ticket_queue"] ==
 ↪  "Paid"]['ticket_queue'].count()

total_tickets_2013 = tickets_2013['ticket_queue'].count()

#Rate will be proportion paid relative to amount issued
ticket_pay_rate_2013 = (paid_2013 / total_tickets_2013) * 100

print(ticket_pay_rate_2013)
```

```
40.59213089209194
```

The payment rate in 2013 was 40.59%

```
#Same formula for 2011
tickets_2011 = sticker_tickets[sticker_tickets['issue_date'].dt.year == 2011]

paid_2011 = tickets_2011[tickets_2011["ticket_queue"] ==
 ↪  "Paid"]['ticket_queue'].count()

total_tickets_2011 = tickets_2011['ticket_queue'].count()

ticket_pay_rate_2011 = (paid_2011 / total_tickets_2011) * 100

print(ticket_pay_rate_2011)
```

```
53.95348837209303
```

The payment rate in 2011 was 53.95%

The difference in repayment rates was:

```
#Take the difference in rates by subtracting the values
difference = (ticket_pay_rate_2013)-(ticket_pay_rate_2011)
print(difference)
```

```
-13.36135748000109
```

There was a 13.36% drop

Let's use the new repayment rate holding issued tickets constant:

```
# Calculate 2011 revenue by holding tickets constant,
# scaling them up, then applying the payment rate and price

old_revenue = (1935*100)*(ticket_pay_rate_2011/100)*(120)
old_revenue

# Calculate 2013 revenue with same tickets, but new
# repayment rate and new price

new_revenue = (1935*100)*(ticket_pay_rate_2013/100)*(200)
new_revenue

# Take the difference between the two
```

```
revenue_increase = new_revenue-old_revenue
print(revenue_increase)
```

```
3181154.6552395783
```

-The projected 2013 revenue is 15,709,154 -The projected revenue increase with these assumptions is 3,181,154

2.5

```
# Creating a loop function to get the repayment rates
years = range(sticker_tickets['issue_date'].dt.year.min(),
              sticker_tickets['issue_date'].dt.year.max() + 1)

repayment_rates = []

for year in years:
    sticker_year = sticker_tickets[
        sticker_tickets['issue_date'].dt.year == year]

    issued = sticker_year['ticket_queue'].count()
    paid = sticker_year[sticker_year["ticket_queue"] ==
 ↪  "Paid"]['ticket_queue'].count()

    repayment_rate = (paid / issued) * 100 if issued > 0 else 0
    repayment_rates.append({'year': year, 'repayment_rate': repayment_rate})

# Create DataFrame from the results
df_repayment = pd.DataFrame(repayment_rates)
print(df_repayment)
```

```
    year  repayment_rate
0   2007       55.085865
1   2008       57.885224
2   2009       53.113383
3   2010       51.987921
4   2011       53.953488
5   2012       48.220803
6   2013       40.592131
7   2014       38.419753
8   2015       40.616133
9   2016       40.768551
```

```
10  2017       37.012411
11  2018       20.144928
```

```
# Repayment rate chart
repayment_rates_chart = alt.Chart(df_repayment).mark_line(point=True).encode(
    x=alt.X('year:O', title='Year'),
    y=alt.Y('repayment_rate:Q', title='Repayment Rate (%)',
 ↪  scale=alt.Scale(zero=False))
).properties(
    title='Repayment Rates for "Missing City Sticker" Tickets'
)

# Add vertical line for the 2012 change
policy_line = alt.Chart(pd.DataFrame({'year': [2012]})).mark_rule().encode(
    x='year:O'
)

# Add text for easier legibility
policy_text = alt.Chart(pd.DataFrame({'year': [2012], 'text': ['Policy
 ↪  Change']})).mark_text(
    align='center',
    baseline='top',
    dx=-40,
    dy=2
).encode(
    x='year:O',
    text='text'
)

# Combine the chart elements
repayment_rates_chart = (repayment_rates_chart + policy_line +
 ↪  policy_text).interactive()

# Display the chart
repayment_rates_chart
```

```
alt.LayerChart(...)
```

Based on this graph, it appears that the policy change's price increase led to a steep drop in repayment rates. This makes sense, as the more expensive prices get, the more we can expect individuals to be either unable to pay for the more expensive ticket or more reluctant to pay because they find it unfair.

2.6

```
# Calculate the total count of each violation type
total_tickets = df['violation_description'].value_counts().reset_index()
total_tickets.columns = ['violation_description', 'total_tickets']

# Calculate repayment rates for each violation type
repayment_rates = df.groupby('violation_description').apply(
    lambda x: (x['ticket_queue'] == 'Paid').sum() / len(x) * 100
).reset_index()
repayment_rates.columns = ['violation_description', 'repayment_rate']

# Merge the total ticket counts with the repayment rates
result = pd.merge(total_tickets, repayment_rates, on='violation_description')

# Calculate expected revenue as total_tickets * repayment_rate
result['projected_revenue'] = result['total_tickets'] *
 ↪  result['repayment_rate']

# Prepare the revenue table with selected columns
revenue_table = result[['violation_description', 'total_tickets',
 ↪  'repayment_rate', 'projected_revenue']]

# Sort by expected revenue in descending order
revenue_table = revenue_table.sort_values('projected_revenue',
 ↪  ascending=False)

# Display the revenue table
print(revenue_table)

revenue_table.head(3)
```

```
                      violation_description  total_tickets  repayment_rate
                      \
0    EXPIRED PLATES OR TEMPORARY REGISTRATION          44811       60.436054
1                             STREET CLEANING          28712       81.161187
2                  RESIDENTIAL PERMIT PARKING          23683       74.226238
3    EXP. METER NON-CENTRAL BUSINESS DISTRICT          20600       79.291262
5                   EXPIRED METER OR OVERSTAY          18756       80.635530
..                                        ...            ...             ...
117   COMMERCIAL IDENTIFICATION ETC. REQUIRED              1      100.000000
118        MORE THAN FOUR FRONT MOUNTED LAMPS              1      100.000000
```

```
103                         DEPR./DIMMED LAMPS              4        0.000000
111              HORN REQUIRED DURING OPERATION             2        0.000000
113                SUSPENSION MODIFIED BEYOND 3             1        0.000000

     projected_revenue
0            2708200.0
1            2330300.0
2            1757900.0
3            1633400.0
5            1512400.0
..                 ...
117             100.0
118             100.0
103               0.0
111               0.0
113               0.0

[119 rows x 4 columns]
```

	violation_description	total_tickets	repayment_rate	projected_
0	EXPIRED PLATES OR TEMPORARY REGISTRATION	44811	60.436054	2708200.0
1	STREET CLEANING	28712	81.161187	2330300.0
2	RESIDENTIAL PERMIT PARKING	23683	74.226238	1757900.0

Based on our generated revenue table, the most lucrative violation is expired plates or temporary registration, followed by street cleaning and residential permit parking. This makes a few key assumptions, such as the idea that individuals view all tickets with equal elasticity and will make identical behavioral changes across all violation types and prices. This is a hard assumption to prove, but focusing on projected revenue through repayment rates may give us a good proxy of what violations people are willing to pay back more frequently.

```
filtered_revenue_table = revenue_table[revenue_table['total_tickets'] >
 ↪  10000]
top_20_revenue_table = filtered_revenue_table.sort_values('repayment_rate',
 ↪  ascending=False).head(20)

chart = alt.Chart(top_20_revenue_table).mark_bar().encode(
    x=alt.X('repayment_rate:Q', title='Repayment Rate (%)'),
    y=alt.Y('violation_description:N', sort='-x', title='Violation
 ↪  Description'),
    tooltip=['violation_description', 'repayment_rate', 'total_tickets']
```

```
).properties(
    title='Top Repayment Rates (with over 10,000 Tickets)',
    width=600,
    height=400
)


chart.display()
```

```
alt.Chart(...)
```

This chart helps support our argument in a different way. It tracks the highest repayment rates among only the violation types with the highest volume of issued tickets. This helps us approximate projected revenue, which is why we can see all 3 of our proposed violation types near the top of this chart.

```
chart = alt.Chart(top_20_revenue_table).mark_bar().encode(
    x=alt.X('projected_revenue:Q', title='Repayment Rate (%)'),
    y=alt.Y('violation_description:N', sort='-x', title='Violation
 ↪  Description'),
    tooltip=['violation_description', 'repayment_rate', 'total_tickets']
).properties(
    title='Top Projected Revenues (with over 10,000 Tickets)',
    width=600,
    height=400
)

chart.display()
```

```
alt.Chart(...)
```

This chart allows us to directly visualize the violations that are projected to earn the most revenue, which is based not only on repayment rate, but also ticket cost and volume. Here we see our three proposed violations at the top.

3.1

```
# Create a new column that filters paid tickets

df['is_paid'] = df['total_payments'] > 0

# Group by violation description, aggregate by mean of paid tickets to obtain
 ↪  the fraction.
```

```
violation_summary = df.groupby('violation_description').agg(
    total_tickets=('violation_description', 'size'),  # Count total tickets
    fraction_paid=('is_paid', 'mean'),
    avg_fine_level1=('fine_level1_amount', 'mean')
).reset_index()

# Sort by descending

violation_summary = violation_summary.sort_values(by='total_tickets',
 ↪  ascending=False)

# Take top 5 violations

top_5_violation_summary = violation_summary.head(5)



print(top_5_violation_summary)
```

```
                      violation_description  total_tickets  fraction_paid
                      \
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

     avg_fine_level1
23         54.968869
101        54.004249
90         66.338302
19         46.598058
81         66.142864
```

3.2

```
violation_summary_filtered =
 ↪  violation_summary[violation_summary['total_tickets'] >= 100]

# Remove the outlier by identifying
# and then excluding the highest fine
```

```
outlier_violation = violation_summary_filtered['avg_fine_level1'].idxmax()
violation_summary_filtered =
 ↪  violation_summary_filtered.drop(outlier_violation)

# Scatterplot
scatter = alt.Chart(violation_summary_filtered).mark_point().encode(
    x=alt.X('avg_fine_level1:Q', title='Average Fine ($)'),
    y=alt.Y('fraction_paid:Q', title='Fraction of Tickets Paid')
).properties(
    title='Relationship Between Fine Amount and Fraction of Tickets Paid'
)

scatter.display()
```

```
alt.Chart(...)
```

Scatterplot takeaways: The scatterplot illustrates a weakly correlated negative relationship that tells us that as the average fine increases, the fraction of tickets paid tends to decrease. We observe a large cluster of ticket types clustered at the top left (i.e. there are several violations for which the average fine and paid ticket fractions are similar)

```
# Boxplot

alt.Chart(violation_summary_filtered).mark_boxplot().encode(
    x=alt.X('avg_fine_level1:Q', title='Average Fine Level 1 ($)'),
    y=alt.Y('fraction_paid:Q', title='Fraction of Tickets Paid')
).properties(
    title='Relationship Between Fine Amount and Fraction of Tickets Paid'
)
```

```
alt.Chart(...)
```

Boxplot takeaways: The boxplot tells us that violation types appear to have different ranges. This helps us get a bit more nuance relative to the cluster in the top-left seen in the scatterplot.

```
# Bar graph

alt.Chart(violation_summary_filtered).mark_bar().encode(
    x=alt.X('avg_fine_level1:Q', title='Average Fine Level 1 ($)'),
    y=alt.Y('fraction_paid:Q', title='Fraction of Tickets Paid'),
```

```
    tooltip=['violation_description']
).properties(
    title='Relationship Between Fine Amount and Fraction of Tickets Paid'
)
```

```
alt.Chart(...)
```

Bar graph takeaways: The bar graph tells us clearly which violations have the highest fraction that's paid, but it's a bit harder to see the variation within each violation description.

3.3

On the one hand, the boxplot contains a lot of granular and detailed information. On the other hand, I would likely still recommend the scatterplot. It contains what is, in my opinion, the easiest to discern relationship between how fine levels affect amount of tickets paid. We see a weak negative correlation, telling us that as fine level increases, ticket payment gradually decreases. While this is perhaps the best 'at-a-glance' data, it's not the most nuanced. There will always be trade-offs with different data styles, but the scatterplot prioritizes readability.

4.1

```
# Load your dataset (assumed already loaded as 'df')
# Check if the fine doubles
df['fine_doubles'] = df['fine_level2_amount'] == 2 * df['fine_level1_amount']

# Calculate the proportion of violations where the fine doubles
proportion_doubles = df['fine_doubles'].mean()
print(f"Proportion of violations that double in price:
 ↪  {proportion_doubles:.2%}")

# Get summary statistics for violations to see if most double
doubles_by_violation =
 ↪  df.groupby('violation_description')['fine_doubles'].mean().reset_index()
```

```
Proportion of violations that double in price: 98.61%
```

This appears to hold for 98.61% of violations

```
violation_counts =
 ↪  df.groupby('violation_description')['ticket_number'].count().reset_index()
violation_counts.columns = ['violation_description', 'citation_count']

# Merge the count data with the original dataframe
```

```
df = pd.merge(df, violation_counts, on='violation_description', suffixes=('',
 ↪  '_count'))

# Filter for violations with at least 100 citations that do not double
at_least_100_citations = df[(df['citation_count'] >= 100) &
 ↪  (df['fine_doubles'] == False)]

# List all such violations
non_doubling_violations = at_least_100_citations[['violation_description',
 ↪  'fine_level1_amount', 'fine_level2_amount']].drop_duplicates()
non_doubling_violations =
 ↪  non_doubling_violations.dropna(subset=['fine_level1_amount',
 ↪  'fine_level2_amount'])

# Calculate the increase for each non-doubling violation
non_doubling_violations['increase_amount'] =
 ↪  non_doubling_violations['fine_level2_amount'] -
 ↪  non_doubling_violations['fine_level1_amount']

non_doubling_violations
```

	violation_description	fine_level1_amount	fine_level2_amo
13	DISABLED PARKING ZONE	200	250
40	PARK OR BLOCK ALLEY	150	250
32664	BLOCK ACCESS/ALLEY/DRIVEWAY/FIRELANE	150	250
65739	SMOKED/TINTED WINDOWS PARKED/STANDING	250	250
65902	OBSTRUCTED OR IMPROPERLY TINTED WINDOWS	250	250
68067	PARK/STAND ON BICYCLE PATH	150	250
138699	NO CITY STICKER VEHICLE OVER 16,000 LBS.	500	775

Through this output, we can observe the violations that don't double. The increase_amount column takes the difference between fine amounts, letting us see that, since these aren't doubled, what the actual change will be. Interestingly, some don't change at all.

4.2

Figure 1: Notice level

Figure 2: Ticket queue

4.3

```
# First we'll generate a scatterplot with labels for everything

scatter = alt.Chart(violation_summary_filtered).mark_point().encode(
    x=alt.X('avg_fine_level1:Q', title='Average Fine Level 1 ($)'),
    y=alt.Y('fraction_paid:Q', title='Fraction of Tickets Paid'),
    tooltip=['violation_description', 'total_tickets', 'avg_fine_level1',
 ↪  'fraction_paid']
).properties(
    title='Relationship Between Fine Amount and Fraction of Tickets Paid'
)

scatter.display()

text = scatter.mark_text(
    align='center',
    baseline='middle'
).encode(
    text='violation_description:N'  # Label each point with the violation
 ↪  description
)

scatter + text
```

```
alt.Chart(...)
```

```
alt.LayerChart(...)
```

```
#This scatterplot will filter out the top 10 violations first

# Sort by total_tickets and get top 10 violations
top_10_violations = violation_summary_filtered.nlargest(10, 'total_tickets')

# Create a new column for labeling top 10 and others as 'Other'
violation_summary_filtered['violation_label'] =
 ↪  violation_summary_filtered['violation_description'].where(

 ↪  violation_summary_filtered['violation_description'].isin(top_10_violations['violation_des
    'Other'
)
```

```
scatter2 =
 ↪  alt.Chart(violation_summary_filtered).mark_point(filled=True).encode(
    x=alt.X('avg_fine_level1:Q', title='Average Fine Level 1 ($)'),
    y=alt.Y('fraction_paid:Q', title='Fraction of Tickets Paid'),
    color=alt.Color('violation_label:N', title='Violation Description')  #
 ↪  Use color to differentiate top 10 vs. 'Other'
).properties(
    title='Relationship Between Fine Amount and Fraction of Tickets Paid'
)

scatter2.display()
```

```
alt.Chart(...)
```

```
#Lastly, this scatterplot will collect the most common keywords
#to group up as many categories as possible

unique_strings = violation_summary_filtered['violation_description'].unique()

# Convert to a list
unique_strings_list = unique_strings.tolist()

# Print the result
unique_strings_df = pd.DataFrame(unique_strings_list,
 ↪  columns=['unique_values'])

#Keywords

def categorize_violation(description):
    if 'EXP' in description:
        return 'EXP'
    elif 'STICKER' in description:
        return 'STICKER'
    elif 'PARK' in description:
        return 'PARK'
    elif 'PLATE' in description:
        return 'PLATE'
    else:
        return 'OTHER'  # Group all other violations as 'OTHER'

# Apply the function to create a new column in the DataFrame
violation_summary_filtered['violation_group'] =
 ↪  violation_summary_filtered['violation_description'].apply(categorize_violation)
```

```
# Create the scatter plot with color encoding for the violation groups
scatter =
 ↪  alt.Chart(violation_summary_filtered).mark_point(filled=True).encode(
    x=alt.X('avg_fine_level1:Q', title='Average Fine Level 1 ($)'),
    y=alt.Y('fraction_paid:Q', title='Fraction of Tickets Paid'),
    color=alt.Color('violation_group:N', title='Violation Group',
 ↪  scale=alt.Scale(scheme='category10'))
).properties(
    title='Relationship Between Fine Amount and Fraction of Tickets Paid'
)

# Display the chart with filled points for better visualization
scatter.display()
```

```
alt.Chart(...)
```