Problem Set 2

AUTHOR
Anand kshirsagar

PUBLISHED
October 12, 2024

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: **anand kshirsagar**
2. "I have uploaded the names of anyone I worked with on the problem set **here**" **__** (1 point)
3. Late coins used this pset: **1** Late coins left after submission: **\0\2**
4. Knit your `ps1.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

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

Data Cleaning continued (15 Points)

1. Finding Columns and NA's in it:

```
import pandas as pd
```

```
#loading the dataframe data
data = pd.read_csv("C:/Users/Lenovo/Documents/GitHub/ppha30538_fall2024/problem_sets/ps1/data/
# creating Function to return a data set with column names and NA counts
def count_na(data):
# Use isna().sum() to count the NA values per column
    na_counts_ = data.isna().sum()

# Creating a return function DataFrame
    result_data = pd.DataFrame({
        'variable': na_counts_.index,
        'na_count': na_counts_.values
    })

    return result_data

# Create DataFrame
return_df = pd.DataFrame(data)


# Test the function
```

```
result = count_na(data)
print(result)
```

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

2.

```
# arranging the highest NA value variables
arrange_na_report = result.sort_values(by='na_count', ascending=False)

# Displaying the top 3 variables with the most missing values
print(arrange_na_report.head(3))
```

```
              variable  na_count
20  hearing_disposition    259899
19         notice_level     84068
7              zipcode     54115
```

```
# rows of top 3 columns with missing values
variable_missing = arrange_na_report['variable'].head(3).values
```

three variables i.e Hearing Disposition, Notice level, and Zipcode are having more missing values. This could be due to following reasons.-

a. Enforcement officer could not able to actually figure out the zipcode of where the violations by the vehicle had been taken place

b. Enforecement offficer could not able to identify the notice level for violation and may have registered it manually

c. Since we have the data gap in the zipcode and notice level information it increases the missing values of hearing disposition with respect to the violation cases

d. This situation may have been aggrevated by the missing information about the 'licence plate type' and 'license plate state'

3.

Increase in the dollar amount of the ticket for not having a city sticker increased from 120 USD to 200 USD.

the old violation code was violation code was 0964125 and new vioaltion code is 0964125B

4.

The cost of an initial offense under old violation code 0964125 was 120$ which increased to 200$ in new vioaltion code 0964125B ####confirm this answer

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

2.1

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

```
data = pd.read_csv("C:/Users/Lenovo/Documents/GitHub/ppha30538_fall2024/problem_sets/ps1/data/
```

```
#Creating a new column so that we can combine the two violation codes
data['combined_city_violation_code'] = data['violation_code'].apply(lambda x: '0964125_cs'
         if x in ['0964125', '0964125B'] else x)

#Checking proper data format
data['issue_date'] = pd.to_datetime(data['issue_date'])

#adding month column
data['month'] = data['issue_date'].dt.to_period('M')

#Collapsing the data so to we can capture missing city sticker tickets by month
monthly_tickets = data[data['combined_city_violation_code'] ==
         '0964125_cs'].groupby('month').size().reset_index(name='ticket_count')

# Converting the period to datetime
monthly_tickets['month'] = monthly_tickets['month'].dt.to_timestamp()

#Creating plot
chart = alt.Chart(monthly_tickets).mark_line().encode(
```

```
    x=alt.X('month:T', title='year Monthwise distribution '),
    y=alt.Y('ticket_count:Q', title=' Missing City Sticker Tickets'),
    tooltip=['month', 'ticket_count']
).properties(
    title='over time Missing City Stickers',
    width=800,
    height=400
)
chart
```

2.2

```
import pandas as pd
import altair as alt

# Load the DataFrame
data = pd.read_csv("C:/Users/Lenovo/Documents/GitHub/ppha30538_fall2024/problem_sets/ps1/data/

# Create a new column to combine the two violation codes
data['combined_city_violation_code'] = data['violation_code'].apply(lambda x: '0964125_cs'
         if x in ['0964125', '0964125B'] else x)

# Checking proper data format
data['issue_date'] = pd.to_datetime(data['issue_date'])

# Adding month column
data['month'] = data['issue_date'].dt.to_period('M')

# Collapsing the data to capture missing city sticker tickets by month
monthly_tickets = data[data['combined_city_violation_code'] ==
         '0964125_cs'].groupby('month').size().reset_index(name='ticket_count')

# Convert the period to datetime
monthly_tickets['month'] = monthly_tickets['month'].dt.to_timestamp()

# Creating the main plot
chart = alt.Chart(monthly_tickets).mark_line().encode(
    x=alt.X(
        'month:T',
        title='Year and Month',
        axis=alt.Axis(
            format='%m/%d/%Y',  # Custom date format to show exact start of years
            tickCount='year',  # One tick per year
            labelAngle=-45      # Rotating labels for better readability
        )
    ),
    y=alt.Y('ticket_count:Q', title='Missing City Sticker Tickets'),
    tooltip=['month:T', 'ticket_count']
).properties(
    title='Missing City Sticker Tickets Over Time',
    width=800,
    height=400
)
```

```
# Add a vertical line for the price increase date
price_increase_date = alt.Chart(pd.DataFrame({'date': [pd.Timestamp('2012-02-
        25')]})).mark_rule(
    color='red',
    strokeDash=[10, 5]
).encode(
    x='date:T',
).properties(
    title='Price Increase Date (Feb 25, 2012)',
)

# Add text label for the price increase date
price_increase_label = alt.Chart(pd.DataFrame({'date': [pd.Timestamp('2012-02-
        25')]})).mark_text(
    align='left',
    dx=5,  # Adjusting x position in the chart
    dy=-10,  # Adjusting y position in the chart
    color='red'
).encode(
    x='date:T',
    text=alt.value('Price Increase\nFeb 25, 2012')
)

# Combine the line chart, the price increase line, and the label
final_chart = chart + price_increase_date + price_increase_label

# Display the final chart
final_chart
```

To understand the Altair's date labeling features I took help from How to filter data based on date? · Issue #263 · vega/altair . link : https://github.com/vega/altair/issues/263 and I also refered and researched 'stak over flow' with link -https://stackoverflow.com/questions/tagged/altair I also took help from CHAT GPT to understand what is being asked in the question.

2.3

```
data['issue_date'] = pd.to_datetime(data['issue_date'])
#lets find out number of total tickets in 2011
total_2011 = data[data['issue_date'].dt.year == 2011].shape[0]
print(f"Total tickets in 2011 (all codes, 1% sample): {total_2011}")
#number of tickets in 2011 for missing city stickers
year_2011_citysticker = data[(data['issue_date'].dt.year == 2011) &
                     (data['violation_code'].isin(['0964125', '0964125B']))]

city_sticker_ticket_count_2011 = len(year_2011_citysticker)
print(f"Number of tickets for codes '0964125' and '0964125B' in 2011 (1% sample):
        {city_sticker_ticket_count_2011}")
#prices
old_price = 120
new_price = 200
#projected revenue for year 2011
total_tickets = city_sticker_ticket_count_2011 * 100  # we had 1% data set so we are Scaling
        it up the observation up to 100%
```

```
old_revenue = total_tickets * old_price
new_revenue = total_tickets * new_price
projected_increase = new_revenue - old_revenue

print("\nProjected Revenue Increase Calculation:")
print(f"Estimated total annual tickets: {total_tickets}")
print(f"Old revenue: ${old_revenue:,.2f}")
print(f"Projected new revenue: ${new_revenue:,.2f}")
print(f"Projected revenue increase: ${projected_increase:,.2f}")
```

```
Total tickets in 2011 (all codes, 1% sample): 24898
Number of tickets for codes '0964125' and '0964125B' in 2011 (1% sample): 1933

Projected Revenue Increase Calculation:
Estimated total annual tickets: 193300
Old revenue: $23,196,000.00
Projected new revenue: $38,660,000.00
Projected revenue increase: $15,464,000.00
```

2.4

```
import pandas as pd

def calculate_repayment_rate(data, year):
    data_year = data[(data['issue_date'].dt.year == year) &
                     (data['violation_code'].isin(['0964125', '0964125B']))]
    tickets_total = len(data_year)
    tickets_paid = data_year[data_year['ticket_queue'].str.lower() == 'paid'].shape[0]
    return tickets_paid / tickets_total if tickets_total > 0 else 0

# Extract relevant columns and convert date
data_relevant = data[['issue_date', 'violation_code', 'ticket_queue']]
data_relevant['issue_date'] = pd.to_datetime(data_relevant['issue_date'])

# Calculate repayment rates for 2011 and 2012
repayment_rate_2011 = calculate_repayment_rate(data_relevant, 2011)
repayment_rate_2012 = calculate_repayment_rate(data_relevant, 2012)

print(f"Repayment rate in 2011: {repayment_rate_2011:.2%}")
print(f"Repayment rate in 2012: {repayment_rate_2012:.2%}")
print(f"Change in repayment rate: {repayment_rate_2012 - repayment_rate_2011:.2%}")

# Revenue calculation
estimated_total_tickets_numbers = 193300
old_price, new_price = 120, 200

revenue_year_2011 = estimated_total_tickets_numbers * old_price * repayment_rate_2011
revenue_year_2012 = estimated_total_tickets_numbers * new_price * repayment_rate_2012
revenue_change = revenue_year_2012 - revenue_year_2011

print(f"\nRevenue in 2011: ${revenue_year_2011:,.2f}")
print(f"Revenue in 2012: ${revenue_year_2012:,.2f}")
print(f"Change in revenue: ${revenue_change:,.2f}")
print(f"Percentage change in revenue: {(revenue_change / revenue_year_2011) * 100:.2f}%")
```

```
Repayment rate in 2011: 53.91%
Repayment rate in 2012: 48.22%
Change in repayment rate: -5.69%

Revenue in 2011: $12,504,000.00
Revenue in 2012: $18,642,162.41
Change in revenue: $6,138,162.41
Percentage change in revenue: 49.09%
```

2.5

```
import pandas as pd
import altair as alt
data['issue_date'] = pd.to_datetime(data['issue_date'])
repayment_monthly_rates = (data[data['violation_code'].isin(['0964125', '0964125B'])]
                .groupby(data['issue_date'].dt.to_period('M'))
                .agg({'ticket_queue': lambda x: (x.str.lower() == 'paid').mean()})
                .reset_index())
repayment_monthly_rates.columns = ['year_month_', 'rate_repayment']
repayment_monthly_rates['year_month_'] =
         repayment_monthly_rates['year_month_'].dt.to_timestamp()
# Define the policy change date
policy_date_change = pd.to_datetime('2012-02-25')
# Create the main chart
main_chart = alt.Chart(repayment_monthly_rates).mark_line().encode(
    x=alt.X('year_month_:T', title='Date'),
    y=alt.Y('rate_repayment:Q', title='Missing Sticker Repayment Rate',
         scale=alt.Scale(domain=[0, 1])),
    tooltip=['year_month_', 'rate_repayment']
)
# Create a vertical line for the policy change date
vertical_line = alt.Chart(pd.DataFrame({'date':
         [policy_date_change]})).mark_rule(color='red').encode(
    x='date:T'
)
# Combine the main chart and the vertical line
chart = (main_chart + vertical_line).properties(
    width=800,
    height=400,
    title='Missing City Sticker Tickets Monthly Repayment Rates (2007-2017)'
)
chart
```

The chart shows that from year 2007 to 2017 there is trend of declining in missing sticker repayment rate with respect to the increase in the price rise in the tickets. We explicitly observe this phenonmeon post the vertical red line which shows the poliy decision of the rise in the ticekt price.

2.6

```
import pandas as pd
import altair as alt

# Drop NAs
data_clean = data.dropna(subset=['violation_code', 'violation_description', 'ticket_queue'])
```

```
# lets Calculate total tickets and repayment rates for each violation type
violation_type_calculation = (
    data_clean.groupby(['violation_code', 'violation_description'])
    .agg(total_tickets_numbers=('ticket_queue', 'count'),
         repayment_rate_violation=('ticket_queue', lambda x: (x.str.lower() ==
         'paid').mean()))
    .reset_index()
)
# Select top three violations
top_violations_type = violation_type_calculation.nlargest(3, ['total_tickets_numbers',
         'repayment_rate_violation'])

# Plot
chart = alt.Chart(top_violations_type).mark_bar().encode(
    x=alt.X('violation_description:N', title='Violation Description Type', sort='-y',
         axis=alt.Axis(labelAngle=-45)),
    y=alt.Y('total_tickets_numbers:Q', title='Total Tickets'),
    color=alt.Color('repayment_rate_violation:Q', scale=alt.Scale(scheme='blues')),
    tooltip=['violation_code', 'violation_description', 'total_tickets_numbers',
         'repayment_rate_violation']
).properties(
    width=450,
    height=400,
    title= 'Top Violation Types'
)
chart
```

By concentrating on these specific violations, the city could increase its revenue without necessarily changing people's behavior. The visual representation backs up this point by highlighting which violations result in the most tickets and are more likely to be paid off.

By focusing on the specefic violations, we can say that city should focus on increasing the revenue without neccesarily focusing on the changing peopl'e behaviour through rise in ticket price. chart highlights the phenomenon that rise in the ticket price may not be corelated to the change in people's behaviour.
Headlines and sub-messages (20 points)

1.

```
import pandas as pd


# Drop NAs for transforming relevant columns
data_clean = data.dropna(subset=['violation_description', 'ticket_queue',
         'fine_level1_amount'])

# Creating the working DataFrame
violation_summary = (
    data_clean.groupby('violation_description')
    .agg(
        total_tickets=('ticket_queue', 'count'),
        fraction_paid=('ticket_queue', lambda x: (x.str.lower() == 'paid').mean()),
        average_fine=('fine_level1_amount', 'mean')
    )
```

```
    .reset_index()
)

# Sorting through total tickets issued
violation_summary_sorted = violation_summary.sort_values(by='total_tickets',
         ascending=False)

# Get the 5 most common violation descriptions
top_5_violations = violation_summary_sorted.head(5)

# Print the result
print(top_5_violations)
```

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

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

2.

Plot 1: Scatter Plot (Fine Amount vs Fraction of Tickets Paid)

```
import pandas as pd
import altair as alt


# Drop NAs for relevant columns
data_clean = data.dropna(subset=['violation_description', 'ticket_queue',
         'fine_level1_amount'])

# Create the required DataFrame with total tickets and fraction paid
violation_summary = (
    data_clean.groupby('violation_description')
    .agg(
        total_tickets=('ticket_queue', 'count'),
        fraction_paid=('ticket_queue', lambda x: (x.str.lower() == 'paid').mean()),
        average_fine=('fine_level1_amount', 'mean')
    )
    .reset_index()
)

# Filter violations that appear at least 100 times
violation_filtered = violation_summary[violation_summary['total_tickets'] >= 100]

# Exclude the outlier with a high fine (e.g., above $500)
violation_filtered = violation_filtered[violation_filtered['average_fine'] <= 500]
```

```
# Scatter plot: Fine amount vs Fraction of tickets paid
scatter_plot = alt.Chart(violation_filtered).mark_point().encode(
    x=alt.X('average_fine:Q', title='Average Amount of fine'),
    y=alt.Y('fraction_paid:Q', title='Fraction of Tickets Paid'),
    tooltip=['violation_description', 'total_tickets', 'fraction_paid', 'average_fine']
).properties(
    width=600,
    height=400,
    title='Relationship between avarage Fine Amount and Fraction of Tickets Paid'
)

# Display the scatter plot
scatter_plot
```

Headline: Higher Fines do not positively Correlate with Higher Repayment Rates of tickets

Sub-message: The scatter plot shows that some violations with lower fines have high repayment rates. Also, increase in fine amount may not always lead to higher repayment rates.

Plot 2: Bar Plot (Grouped by Fine Buckets) lets group the fine amounts into buckets and reflect the average fraction of tickets paid for each group.

```
# Creating buckets for fine amounts
violation_filtered['fine_bucket'] = pd.cut(violation_filtered['average_fine'], bins=[0, 100,
         200, 300, 400, 500], labels=['0-100', '101-200', '201-300', '301-400', '401-500'])

# lets create the Bar plot
bar_plot = alt.Chart(violation_filtered).mark_bar().encode(
    x=alt.X('fine_bucket:N', title='Fine Amount Bucket'),
    y=alt.Y('mean(fraction_paid):Q', title='Average Repayment Rate'),
    tooltip=['fine_bucket', 'mean(fraction_paid)']
).properties(
    width=600,
    height=400,
    title='Average Repayment Rate for Different Fine Amount Buckets'
)

# Display the bar plot
bar_plot
```

Headline: Moderate fine levels on violations create possibility of more higher repayment rates

Sub-message: very low and very high fines show lower repayment rates where as ticket fines in the mid range from 100 to 200 dollars tend to have the highest repayment rates. it also shows higher the amount of fine , lower is its repayment rates.

plot-3 # Line plot: Fine amount vs fraction of tickets paid

```
line_plot = alt.Chart(violation_filtered).mark_line().encode(
    x=alt.X('average_fine:Q', title='Average Fine Amount'),
    y=alt.Y('fraction_paid:Q', title='Fraction of Tickets Paid'),
    tooltip=['violation_description', 'total_tickets', 'fraction_paid', 'average_fine']
```

```
).properties(
    width=800,
    height=400,
    title='Trend of Repayment Rates Across Different Fine Amounts'
)

# Display the line plot
line_plot
```

Headline: Repayment Rates is fluctuating trend with respect to Fine Increases

Sub-message: The line plot reveals no clear linear relationship between fine amounts and repayment rates. But we can observe the general trend in reduction in the avarage fine amount with respect to fraction of tickets paid.

3.

If the City Clerk can only look at one plot, I would recommend Plot 2: Fine Buckets vs. Average Repayment Rates as effective visual mode of communication.

Why the Bar Plot?

It provides simplicity and clarity in understanding. The bar plot helps us easily understand the relationship between fine amount and avarage repayment rates. It helps us clearly grasp the discripancy in the policy argument that increase in the fine will lead to the higher ticket repayment rate and help us increase the city revenue.

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

1.

```
import pandas as pd
```

```
# Droping rows where fine level columns are missing
data_clean = data.dropna(subset=['fine_level1_amount', 'fine_level2_amount'])

# Group by violation code and calculate total citations
violation_stats = data_clean.groupby('violation_code').agg(
    total_citations=('ticket_queue', 'count'),
    avg_fine_level1=('fine_level1_amount', 'mean'),
    avg_fine_level2=('fine_level2_amount', 'mean')
).reset_index()

# Filtering violations which have at least 100 citations
violation_100 = violation_stats[violation_stats['total_citations'] >= 100]

# lets Find those violations where the fine does not double
violation_100['fine_doubles'] = violation_100['avg_fine_level2'] == 2 *
         violation_100['avg_fine_level1']

# Filtering violations where the fine does not double
no_doubling_violations = violation_100[violation_100['fine_doubles'] == False]
```

```
# Calculating the increase for non-doubling violations
no_doubling_violations['fine_increase'] = no_doubling_violations['avg_fine_level2'] -
        no_doubling_violations['avg_fine_level1']
no_doubling_violations['percentage_increase'] = (no_doubling_violations['fine_increase'] /
        no_doubling_violations['avg_fine_level1']) * 100

# results
no_doubling_violations = no_doubling_violations[['violation_code', 'total_citations',
        'avg_fine_level1', 'avg_fine_level2', 'fine_increase', 'percentage_increase']]

# Sort by total citations and get the top 5 violation types
top_5_violations = no_doubling_violations.sort_values(by='total_citations',
        ascending=False).head(5)

# Merge with the original data to get violation descriptions
top_5_violations_with_description = pd.merge(top_5_violations, data[['violation_code',
        'violation_description']].drop_duplicates(), on='violation_code', how='left')

# Final result with violation descriptions
top_5_violations_with_description = top_5_violations_with_description[['violation_code',
        'violation_description', 'total_citations', 'avg_fine_level1', 'avg_fine_level2',
        'fine_increase', 'percentage_increase']]
```

so we can observe that not all violation types double in price and following are the violation types codes which are not doubled in their price. the following table also shows the percentage increase in their ticket increase if unpaid.

```
# Display the result
top_5_violations_with_description
```

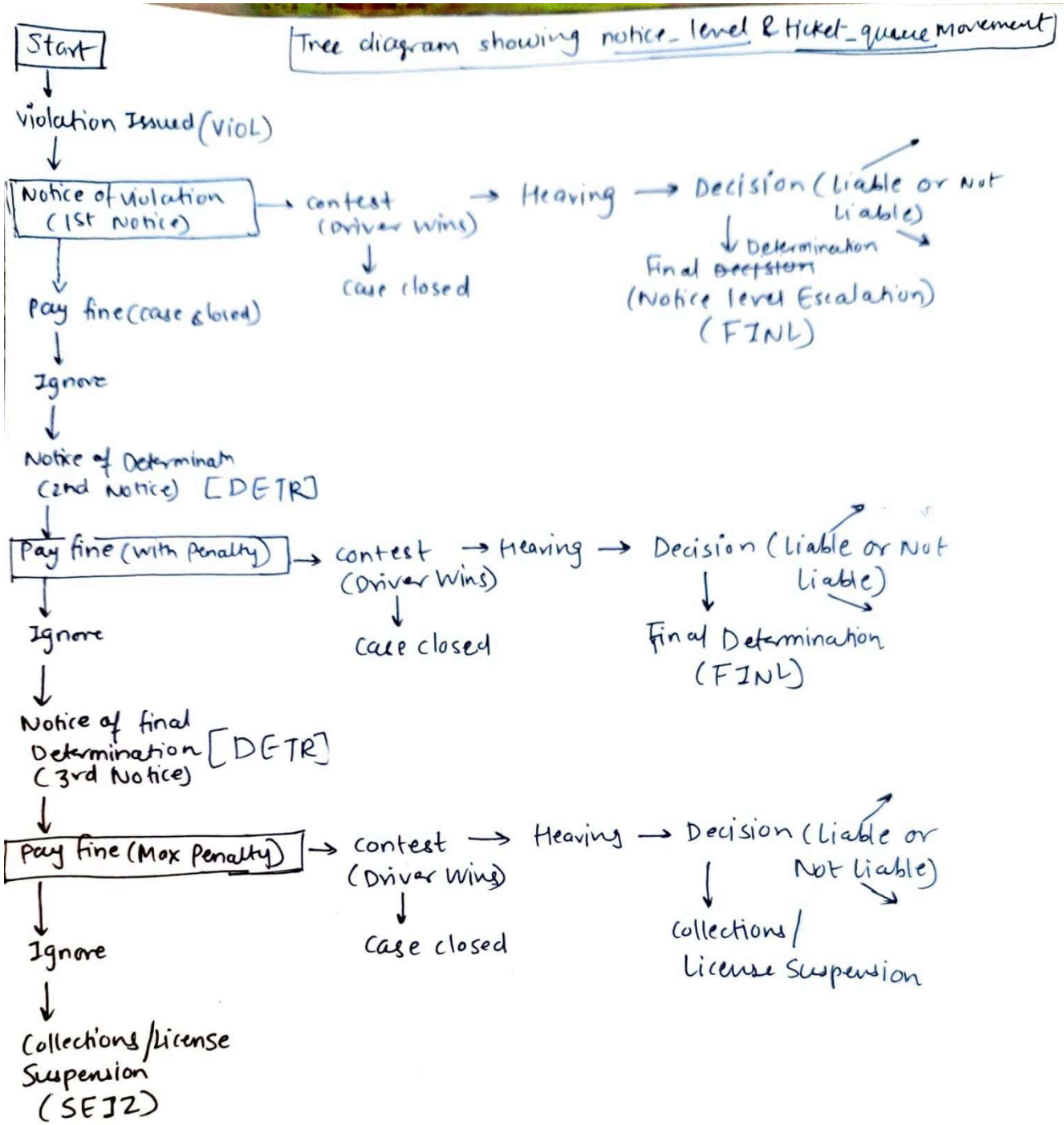
	violation_code	violation_description	total_citations	avg_fine_level1	avg_fine_level2	fine_incre
0	0964130	PARK OR BLOCK ALLEY	2050	150.000000	259.926829	109.92682
1	0964050J	DISABLED PARKING ZONE	2034	216.986234	358.308751	141.32251
2	0976220B	SMOKED/TINTED WINDOWS PARKED/STANDING	1697	151.090159	209.516794	58.426635
3	0964100C	BLOCK ACCESS/ALLEY/DRIVEWAY/FIRELANE	1579	141.592780	266.751108	125.15832
4	0976220A	OBSTRUCTED OR IMPROPERLY TINTED WINDOWS	271	156.180812	225.645756	69.464945

2.

```
from IPython.display import Image  # Import Image class from IPython.display

# Specify the path to your image
image_path = "C:/Users/Lenovo/Downloads/WhatsApp Image 2024-10-20 at 3.41.13 PM.jpeg"

# Load and display the image
display(Image(filename=image_path))
```



I have compiled the requiremetn of the two tree diagrams in to one diagram. with this image we can observe that the structure of movement of notice_level and ticket_queue are interconnected with each other.where there exist the specific mechanism through which one can appeal to the courts against the ticek/ fine they recieved from the enforcing authorities. After understandng this strucutre one can say that an individul who cannot afford to pay fine at 1st instance can be more vulenrable to the further fines which can create financian burden.

3.

a. lets add label of the violation types to each points in the scatter plot.

```
import pandas as pd
import altair as alt

# Drop NAs
data_clean = data.dropna(subset=['violation_description', 'ticket_queue',
         'fine_level1_amount'])

# Create the dataFrame violation_repository with total tickets and fraction paid
violation_repository = (
    data_clean.groupby('violation_description')
    .agg(
        total_tickets=('ticket_queue', 'count'),
        fraction_paid=('ticket_queue', lambda x: (x.str.lower() == 'paid').mean()),
        average_fine=('fine_level1_amount', 'mean')
    )
    .reset_index()
)

# Filter traffic/ vehicle violations that appear at least 100 times
violation_filtered = violation_repository[violation_summary['total_tickets'] >= 100]

# drop the outlier violation type with a high fine
violation_filtered = violation_filtered[violation_filtered['average_fine'] <= 500]

# Scatter plot: Fine amount vs Fraction of tickets paid
scatter_plot = alt.Chart(violation_filtered).mark_point().encode(
    x=alt.X('average_fine:Q', title='Average Amount of Fine'),
    y=alt.Y('fraction_paid:Q', title='Fraction of Tickets Paid'),
    tooltip=['violation_description', 'total_tickets', 'fraction_paid', 'average_fine']
).properties(
    width=600,
    height=400,
    title='Relationship between Average Fine Amount and Fraction of Tickets Paid'
)

# Adding labels to each point
text_labels = scatter_plot.mark_text(
    align='left',
    baseline='middle',
    dx=7  # Adjust this value to shift text horizontally
).encode(
    text='violation_description:N'  # Label each point with violation description
)

# Combine the scatter plot with text labels
final_chart = scatter_plot + text_labels
```

```
# Display the final chart
final_chart
```

b.

```
import pandas as pd
import altair as alt

# Drop NAs
data_clean = data.dropna(subset=['violation_description', 'ticket_queue',
         'fine_level1_amount'])

# Creating required dataframe with total tickets and fraction paid
violation_summary = (
    data_clean.groupby('violation_description')
    .agg(
        total_tickets=('ticket_queue', 'count'),
        fraction_paid=('ticket_queue', lambda x: (x.str.lower() == 'paid').mean()),
        average_fine=('fine_level1_amount', 'mean')
    )
    .reset_index()
)

# Filter violations that appear at least 100 times
violation_filtered = violation_summary[violation_summary['total_tickets'] >= 100]

# drop the outlier violation type with a high fine
violation_filtered = violation_filtered[violation_filtered['average_fine'] <= 500]

# Scatter plot: Fine amount vs Fraction of tickets paid with color legend for
         violation_description
scatter_plot = alt.Chart(violation_filtered).mark_point().encode(
    x=alt.X('average_fine:Q', title='Average Amount of Fine'),
    y=alt.Y('fraction_paid:Q', title='Fraction of Tickets Paid'),
    color=alt.Color('violation_description:N', legend=alt.Legend(title="Violation
         Description")),  # Encoding violation description as color in the legend
    tooltip=['violation_description', 'total_tickets', 'fraction_paid', 'average_fine']
).properties(
    width=600,
    height=400,
    title='Relationship between Average Fine Amount and Fraction of Tickets Paid'
)

# Display the final chart with legend
scatter_plot
```

now, as we observe we tried to do both the ways to develop the charts as question suggested we face the issue that there are too many labels and the plot is illegible.

so, now focus on top 10 violation types and transform all the remaining violation as ‘other’

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

```
# Drop NAs
data_clean = data.dropna(subset=['violation_description', 'ticket_queue',
         'fine_level1_amount'])

# Create the DataFrame violation_repository with total tickets and fraction paid
violation_repository = (
    data_clean.groupby('violation_description')
    .agg(
        total_tickets=('ticket_queue', 'count'),
        fraction_paid=('ticket_queue', lambda x: (x.str.lower() == 'paid').mean()),
        average_fine=('fine_level1_amount', 'mean')
    )
    .reset_index()
)

# Filter traffic/vehicle violations that appear at least 100 times
violation_filtered = violation_repository[violation_repository['total_tickets'] >= 100]

# Drop the outlier violation type with a high fine
violation_filtered = violation_filtered[violation_filtered['average_fine'] <= 500]

# Get the ten most common violation descriptions
top_violations = violation_filtered.nlargest(10, 'total_tickets')
         ['violation_description'].tolist()

# Create a new column to replace less common descriptions with "Other"
violation_filtered['violation_display'] = violation_filtered['violation_description'].apply(
    lambda x: x if x in top_violations else 'Other'
)

# Define a custom color scale with a limited number of colors
custom_colors = {
    **{violation: color for violation, color in zip(top_violations, ['#1f77b4', '#ff7f0e',
         '#2ca02c', '#d62728', '#9467bd',
                                                                     '#8c564b', '#e377c2',
         '#7f7f7f', '#bcbd22', '#17becf'])},
    'Other': '#7f7f7f'  # Grey for other
}

# Scatter plot: Fine amount vs Fraction of tickets paid
scatter_plot = alt.Chart(violation_filtered).mark_point().encode(
    x=alt.X('average_fine:Q', title='Average Amount of Fine'),
    y=alt.Y('fraction_paid:Q', title='Fraction of Tickets Paid'),
    color=alt.Color('violation_display:N', title='Violation Description',
         scale=alt.Scale(domain=list(custom_colors.keys()),
         range=list(custom_colors.values()))),
    tooltip=['violation_description', 'total_tickets', 'fraction_paid', 'average_fine']
).properties(
    width=600,
    height=400,
    title='Relationship between Average Fine Amount and Fraction of Tickets Paid'
)

# Adding labels to each point for the top violations
text_labels = scatter_plot.mark_text(
    align='left',
```

```
    baseline='middle',
    dx=7  # Adjust this value to shift text horizontally
).encode(
    text='violation_display:N'  # Label each point with the violation display
)

# Combine the scatter plot with text labels
final_chart = scatter_plot + text_labels

# Display the final chart
final_chart
```

now, lets construct meaningful categories by marking violation descriptions which sound similar with a common label and a common color.

```
import pandas as pd
import altair as alt

# Drop NAs
data_clean = data.dropna(subset=['violation_description', 'ticket_queue',
         'fine_level1_amount'])

# Define a function to group similar violations into categories
def categorize_violation(description):
    description = description.lower()
    if 'expired' in description or 'expiration' in description:
        return 'Expired Vehicle Documents'
    elif 'parking' in description or 'parked' in description:
        return 'Parking Violation'
    elif 'speed' in description or 'speeding' in description:
        return 'Speeding'
    elif 'meter' in description:
        return 'Meter Violation'
    elif 'sign' in description or 'zone' in description:
        return 'Sign/Zone Violation'
    else:
        return 'Other Violation'

# Apply the categorization function
data_clean['violation_category'] =
         data_clean['violation_description'].apply(categorize_violation)

# Create the required dataframe with total tickets and fraction paid
violation_summary = (
    data_clean.groupby('violation_category')
    .agg(
        total_tickets=('ticket_queue', 'count'),
        fraction_paid=('ticket_queue', lambda x: (x.str.lower() == 'paid').mean()),
        average_fine=('fine_level1_amount', 'mean')
    )
    .reset_index()
)

# Sort by total_tickets and select the top 10 violation categories
```

```
top_10_violations = violation_summary.sort_values(by='total_tickets',
        ascending=False).head(10)

# Drop outlier violation types with a high fine
top_10_violations = top_10_violations[top_10_violations['average_fine'] <= 500]

# Scatter plot: Fine amount vs Fraction of tickets paid with color legend for
        violation_category
scatter_plot = alt.Chart(top_10_violations).mark_point().encode(
    x=alt.X('average_fine:Q', title='Average Amount of Fine'),
    y=alt.Y('fraction_paid:Q', title='Fraction of Tickets Paid'),
    color=alt.Color('violation_category:N', legend=alt.Legend(title="Violation Category")),
    tooltip=['violation_category', 'total_tickets', 'fraction_paid', 'average_fine']
).properties(
    width=600,
    height=400,
    title='Top 10 Violations: Relationship between Average Fine Amount and Fraction of
        Tickets Paid'
)

# Display the final chart with legend
scatter_plot
```

Extra Credit (max 5 points)

1.

```
import pandas as pd
data = pd.read_csv("C:/Users/Lenovo/Documents/GitHub/ppha30538_fall2024/problem_sets/ps1/data/
```

```
# Group by 'violation_code' and count unique descriptions
violation_counts = data.groupby('violation_code')['violation_description'].nunique()

# Filtering codes with multiple descriptions
multiple_descriptions = violation_counts[violation_counts > 1].index

# Getting most common description for each violation code
most_common_description =
        data[data['violation_code'].isin(multiple_descriptions)].groupby('violation_code')
        ['violation_description'].agg(lambda x: x.mode()[0])

# Creating a new column for the most common description
data['most_common_description'] =
        data['violation_code'].map(most_common_description).fillna('')

# Counting occurrences of each violation code
code_counts = data['violation_code'].value_counts()

# Getting three codes with the most observations
top_three_codes = code_counts.head(3)
```

```
# Print results
print("Most common violation descriptions for codes with multiple descriptions:")
print(most_common_description)
print("\nTop three violation codes with the most observations:")
print(top_three_codes)
```

```
Most common violation descriptions for codes with multiple descriptions:
violation_code
0964040B                            STREET CLEANING
0964041B                SPECIAL EVENTS RESTRICTION
0964070             SNOW ROUTE: 2'' OF SNOW OR MORE
0964170D      TRUCK TRAILOR/SEMI/TRAILER PROHIBITED
0964200B                SPECIAL EVENTS RESTRICTION
0976160A              REAR AND FRONT PLATE REQUIRED
0976160B    EXPIRED PLATE OR TEMPORARY REGISTRATION
0980110B              HAZARDOUS DILAPITATED VEHICLE
Name: violation_description, dtype: object

Top three violation codes with the most observations:
violation_code
0976160F    44811
0964040B    32082
0964090E    23683
Name: count, dtype: int64
```