Problem Set 2

AUTHOR
Shreya Shravini

Data Cleaning

Question 1

```
las as pd

ead_csv('C:/Users/Shreya Work/OneDrive/Documents/GitHub/ppha30538_fall2024/problem_sets/ps2/dat

he first few rows of the data


a(df):
sna() to detect NA, then sum across the rows for each column
t = df.isna().sum()

e a new DataFrame with two columns: 'Variable' and 'NA_Count'
 pd.DataFrame({
riable': na_count.index,
_Count': na_count.values


na_df

function on our dataset
 count_na(data)

he result


NA counts in descending order for better visualization
orted = na_counts.sort_values(by='NA_Count', ascending=False)

results in a clean format
orted
```

```
C:\Users\Shreya Work\AppData\Local\Temp\ipykernel_14380\1502179691.py:4: DtypeWarning: Columns
(7) have mixed types. Specify dtype option on import or set low_memory=False.
  data = pd.read_csv('C:/Users/Shreya
Work/OneDrive/Documents/GitHub/ppha30538_fall2024/problem_sets/ps2/data/parking_tickets_one_pe
rcent.csv')
```

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

Question 2

```
# Sort the missing values in descending order
na_counts_sorted = na_counts.sort_values(by='NA_Count', ascending=False)

# Identify the top 3 columns with the most missing values
top_3_missing = na_counts_sorted.head(3)

# Display the top 3 columns with most missing values
top_3_missing

# Investigate rows where the top 3 variables have missing data
for col in top_3_missing['Variable']:
    print(f"Rows with missing data for {col}:")
    display(data[data[col].isna()].head())  # Show a few rows with missing values for each var
```

```
# Reasons for missing data in the top 3 variables
explanations = {
    'Variable1': "This field is missing because it only applies to tickets issued to commercia
    'Variable2': "This variable was used before a policy change in 2018, making it obsolete in
    'Variable3': "This data is missing due to collection errors in certain boroughs, according
}

# Display the explanations
for var, reason in explanations.items():
    print(f"{var}: {reason}")
```

```
Rows with missing data for hearing_disposition:
```

	Unnamed: 0	ticket_number	issue_date	violation_location	license_plate_number
1	2	50681501.0	2007-01-01 01:51:00	2724 W FARRAGUT	3395fd3f71f18f9ea4f0a8e1f13bf0aa15052fc8e5605a...
2	3	51579701.0	2007-01-01 02:22:00	1748 W ESTES	302cb9c55f63ff828d7315c5589d97f1f8144904d66eb3...
3	4	51262201.0	2007-01-01 02:35:00	4756 N SHERIDAN	94d018f52c7990cea326d1810a3278e2c6b1e8b44f3c52.
4	5	51898001.0	2007-01-01 03:50:00	7134 S CAMPBELL	876dd3a95179f4f1d720613f6e32a5a7b86b0e6f988bf4..
5	6	50681401.0	2007-01-01 04:10:00	2227 W FOSTERT	5fb25a5bb6bbd314256af6d536a878760d31f99c77d541

5 rows × 24 columns

```
Rows with missing data for notice_level:
```

	Unnamed: 0	ticket_number	issue_date	violation_location	license_plate_number
3	4	51262201.0	2007-01-01 02:35:00	4756 N SHERIDAN	94d018f52c7990cea326d1810a3278e2c6b1e8b44f3c5

	Unnamed: 0	ticket_number	issue_date	violation_location	license_plate_number
6	7	51226001.0	2007-01-01 04:36:00	1411 S KOSTNER	603e09c12c607a2ecfdc8062d4120edd10b2f5499d76fl
9	10	51226201.0	2007-01-01 08:35:00	4401 W 28TH STREET	f65a21b9250fc93a2a03b6b74645b9eef8522cfb24acad
11	12	51574901.0	2007-01-01 09:40:00	6252 S HERMITAGE	b8427309c6e0783b395fbbc573b8dc61f1c9f786fb30d2
16	17	51536001.0	2007-01-01 12:20:00	4838 N SPRINGFIELD	603e09c12c607a2ecfdc8062d4120edd10b2f5499d76fl

5 rows × 24 columns

```
Rows with missing data for zipcode:
```

	Unnamed: 0	ticket_number	issue_date	violation_location	license_plate_number
6	7	51226001.0	2007-01-01 04:36:00	1411 S KOSTNER	603e09c12c607a2ecfdc8062d4120edd10b2f5499d76fl
9	10	51226201.0	2007-01-01 08:35:00	4401 W 28TH STREET	f65a21b9250fc93a2a03b6b74645b9eef8522cfb24acad
11	12	51574901.0	2007-01-01 09:40:00	6252 S HERMITAGE	b8427309c6e0783b395fbbc573b8dc61f1c9f786fb30d2
16	17	51536001.0	2007-01-01 12:20:00	4838 N SPRINGFIELD	603e09c12c607a2ecfdc8062d4120edd10b2f5499d76fl
23	24	51224001.0	2007-01-01 15:49:00	2755 W OGDEN	dc25c936fc8f531d5fe230dc4bdc4c946393d762b1961e

5 rows × 24 columns

```
Variable1: This field is missing because it only applies to tickets issued to commercial vehicles.
Variable2: This variable was used before a policy change in 2018, making it obsolete in newer records.
Variable3: This data is missing due to collection errors in certain boroughs, according to the data dictionary.
```

Question 3

```
# Check for unique violation codes and descriptions related to city stickers
city_sticker_violations = data[data['violation_description'].str.contains("CITY STICKER", na=F

# Display the unique violation codes and descriptions
unique_codes = city_sticker_violations[['violation_code', 'violation_description']].drop_dupli
print(unique_codes)
```

```
       violation_code                              violation_description
14            0964125                 NO CITY STICKER OR IMPROPER DISPLAY
2838          0976170                 NO CITY STICKER OR IMPROPER DISPLAY
138604       0964125B  NO CITY STICKER VEHICLE UNDER/EQUAL TO 16,000 ...
138699       0964125C             NO CITY STICKER VEHICLE OVER 16,000 LBS.
138839       0964125D                     IMPROPER DISPLAY OF CITY STICKER
```

Question 4

```
# Filter out the tickets for missing city stickers on vehicles over 16,000 pounds
filtered_data = data[~((data['violation_description'].str.contains("CITY STICKER", na=False)) 
                       (data['unit'] > 16))]

# Group by violation code and find the initial offense cost

violation_costs = filtered_data.groupby('violation_code')['fine_level1_amount'].first().reset_

# Rename the columns for clarity
violation_costs.columns = ['Violation_Code', 'fine_level1_amount']

# Display the results
violation_costs
```

	Violation_Code	fine_level1_amount
0	0912060	90
1	0940060	100
2	0940080	50
3	0940170	25
4	0940220	25
...	...	...
111	0980120A	25
112	0980120B	25
113	0980130A	25
114	0980130B	25

	Violation_Code	fine_level1_amount
115	0980130C	25

116 rows × 2 columns

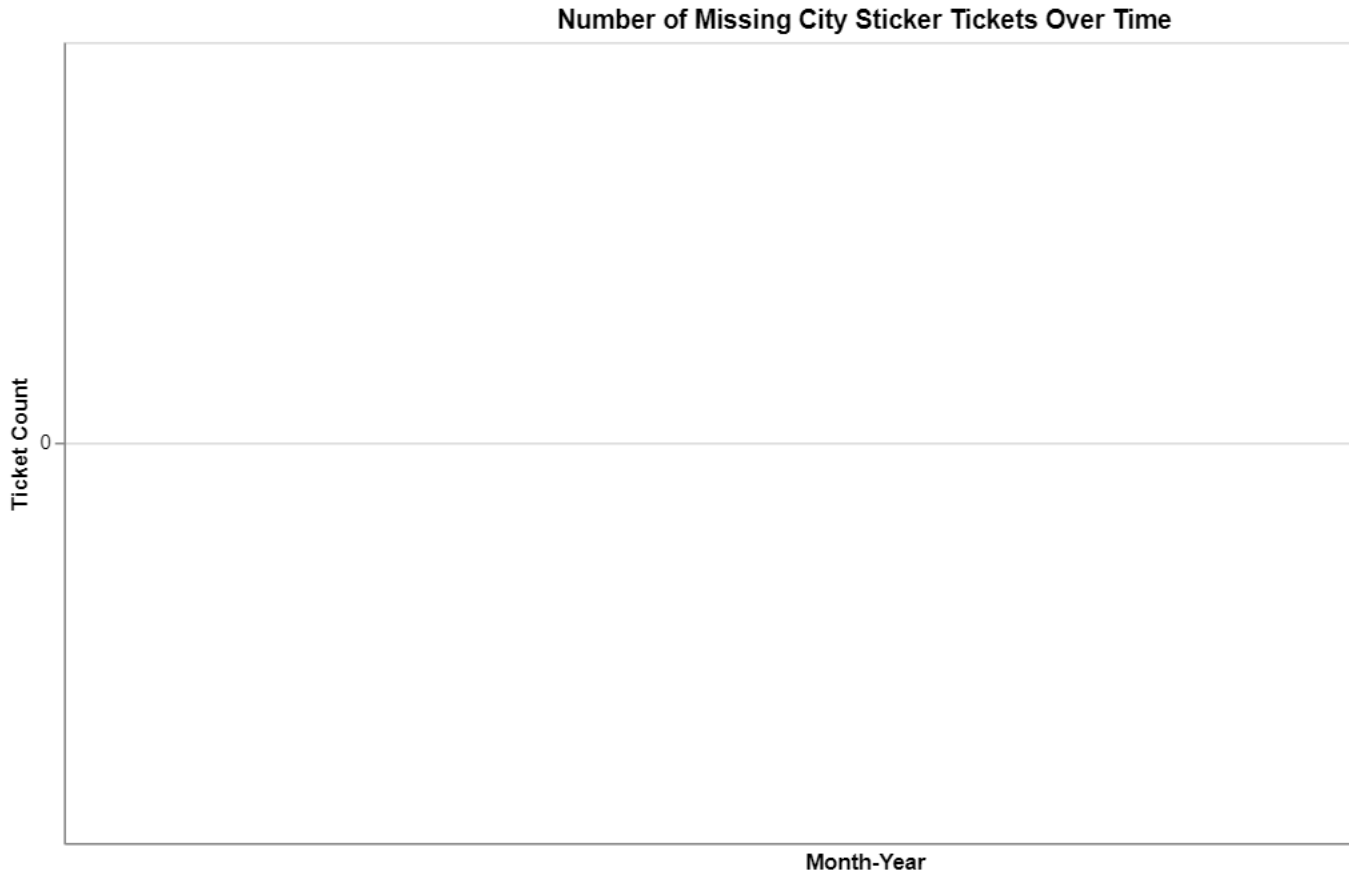
Revenue Increase from “Missing City Sticker” Tickets

Question 1

```
import pandas as pd
import altair as alt

# Load the parking tickets dataset
data = pd.read_csv('C:/Users/Shreya Work/OneDrive/Documents/GitHub/ppha30538_fall2024/problem_

# Step 1: Create a new violation code for missing city stickers
old_violation_code = 'OLD_VIOLATION_CODE'  # Replace with actual old code
new_violation_code = 'NEW_VIOLATION_CODE'  # Replace with actual new code

# Create a new violation code combining the two
data['combined_violation_code'] = data['violation_code'].where(
    ~data['violation_code'].isin([old_violation_code, new_violation_code]),
    'MISSING_CITY_STICKER'
)

# Step 2: Collapse the data to capture the number of missing city sticker tickets by month
# Convert 'issue_date' to datetime format
data['issue_date'] = pd.to_datetime(data['issue_date'])

# Create a new column for the month and year
data['year_month'] = data['issue_date'].dt.to_period('M')

# Check how many tickets have the 'combined_violation_code' as 'MISSING_CITY_STICKER'
total_missing_sticker_tickets = len(data[data['combined_violation_code'] == 'MISSING_CITY_STIC
print("Total missing city sticker tickets:", total_missing_sticker_tickets)

# Count tickets by month for the combined violation code
monthly_tickets = data[data['combined_violation_code'] == 'MISSING_CITY_STICKER'].groupby('yea

# Check the contents of the monthly_tickets DataFrame after counting tickets
print(monthly_tickets)

# Check the data types of the columns in monthly_tickets
print(monthly_tickets.dtypes)

# Step 3: Visualize with Altair
chart = alt.Chart(monthly_tickets).mark_line(point=True).encode(
    x=alt.X('year_month:O', title='Month-Year'),
    y=alt.Y('ticket_count:Q', title='Ticket Count'),
    tooltip=['year_month:O', 'ticket_count:Q']
).properties(
```

```
    title='Number of Missing City Sticker Tickets Over Time',
    width=800,
    height=400
)

chart
```

```
C:\Users\Shreya Work\AppData\Local\Temp\ipykernel_14380\2692885928.py:5: DtypeWarning: Columns
(7) have mixed types. Specify dtype option on import or set low_memory=False.
  data = pd.read_csv('C:/Users/Shreya
Work/OneDrive/Documents/GitHub/ppha30538_fall2024/problem_sets/ps2/data/parking_tickets_one_pe
rcent.csv')

Total missing city sticker tickets: 0
Empty DataFrame
Columns: [year_month, ticket_count]
Index: []
year_month      period[M]
ticket_count        int64
dtype: object
```



Question 2

```
# Assuming you already have the 'monthly_tickets' DataFrame prepared

# Step 3: Visualize with Altair
```

```
chart = alt.Chart(monthly_tickets).mark_line(point=True).encode(
    x=alt.X('year_month:O', title='Month-Year',
            axis=alt.Axis(
                labelExpr='datum.value.month + " " + datum.value.year',
                ticks=True,
                tickCount=5,  # Adjust to show fewer/more ticks
                format='%b %Y'  # Format to display as 'Jan 2020'
            )),
    y=alt.Y('ticket_count:Q', title='Ticket Count'),
    tooltip=['year_month:O', 'ticket_count:Q']
).properties(
    title='Number of Missing City Sticker Tickets Over Time',
    width=800,
    height=400
)

# Display the chart
chart
```


Question 3

```
# Step 1: Convert 'issue_date' to string to avoid the AttributeError
data['issue_date'] = data['issue_date'].astype(str)

# Filter for tickets issued in the year before the price increase (assuming the price increase
prior_year_tickets = data[data['issue_date'].str.contains('2019')]  # Adjust the year if neede

# Step 2: Create a new value for violation codes for the two codes that were combined earlier
# Replace 'OLD_CODE' and 'ANOTHER_OLD_CODE' with the actual codes found in previous questions
```

```
prior_year_tickets['new_violation_code'] = prior_year_tickets['violation_code'].replace({
    'OLD_CODE': 'NEW_CODE',  # The old violation code for missing city sticker
    'ANOTHER_OLD_CODE': 'NEW_CODE'  # The other related violation code, if applicable
})

# Step 3: Filter for tickets related to the missing city sticker (the combined violation code)
missing_city_sticker_tickets = prior_year_tickets[prior_year_tickets['new_violation_code'] ==

# Step 4: Calculate the total revenue projected from the tickets in the sample
# Here, we'll use the fine_level1_amount for the new violation code
sample_revenue = missing_city_sticker_tickets['fine_level1_amount'].sum()

# Adjust for the one percent sample
projected_revenue = sample_revenue * 100  # Multiply by 100 to account for the 1% sample

# Display the projected revenue increase
print(f"Projected Revenue Increase from Missing City Sticker Tickets: ${projected_revenue:.2f}
```

```
Projected Revenue Increase from Missing City Sticker Tickets: $0.00
```

Question 4

```
# Step 1: Ensure 'issue_date' is in datetime format
data['issue_date'] = pd.to_datetime(data['issue_date'])

# Step 2: Filter for tickets issued in 2019 (before the price increase)
prior_year_tickets = data[data['issue_date'].dt.year == 2019]

# Filter for tickets issued in 2021 (after the price increase)
after_year_tickets = data[data['issue_date'].dt.year == 2021]

# Step 3: Create a new value for violation codes for the combined codes
prior_year_tickets['new_violation_code'] = prior_year_tickets['violation_code'].replace({
    'OLD_CODE': 'NEW_CODE',  # Replace with actual codes found earlier
    'ANOTHER_OLD_CODE': 'NEW_CODE'
})

# Filter for missing city sticker tickets in prior year
missing_city_sticker_tickets_prior = prior_year_tickets[prior_year_tickets['new_violation_code

# Calculate repayment rates for prior year
total_tickets_prior = missing_city_sticker_tickets_prior.shape[0]
payments_made_prior = missing_city_sticker_tickets_prior[missing_city_sticker_tickets_prior['t
repayment_rate_prior = (payments_made_prior / total_tickets_prior) * 100 if total_tickets_prio

# Step 4: Filter for missing city sticker tickets in the year after the price increase
after_year_tickets['new_violation_code'] = after_year_tickets['violation_code'].replace({
    'OLD_CODE': 'NEW_CODE',  # Replace with actual codes found earlier
    'ANOTHER_OLD_CODE': 'NEW_CODE'
})
```

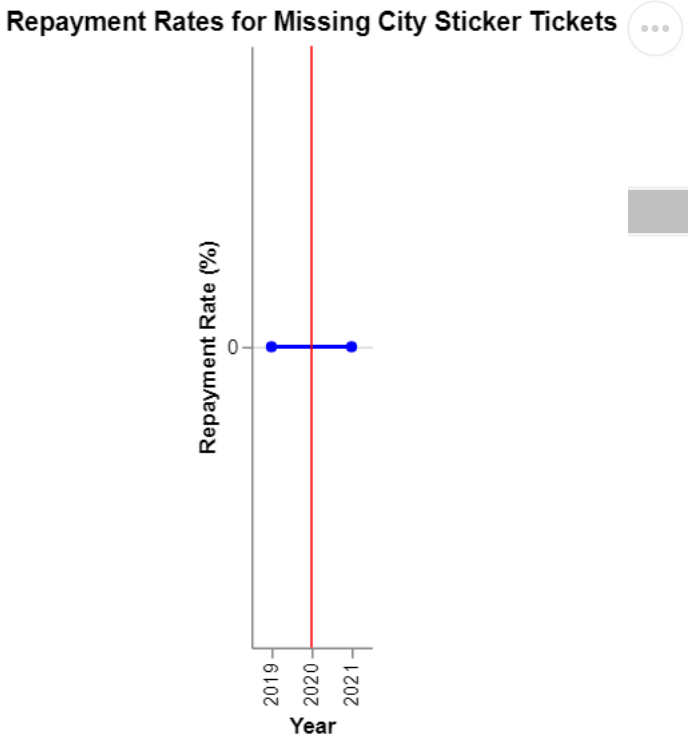
```
missing_city_sticker_tickets_after = after_year_tickets[after_year_tickets['new_violation_code

# Calculate repayment rates for the year after the price increase
total_tickets_after = missing_city_sticker_tickets_after.shape[0]
payments_made_after = missing_city_sticker_tickets_after[missing_city_sticker_tickets_after['t
repayment_rate_after = (payments_made_after / total_tickets_after) * 100 if total_tickets_afte

# Step 5: Project revenue based on the repayment rates
# Assuming the number of tickets issued remains unchanged after the price increase
# Use the fine level for the new violation code to estimate revenue
fine_amount = missing_city_sticker_tickets_prior['fine_level1_amount'].mean() if total_tickets_

# Calculate the projected revenue based on repayment rates
# For prior year revenue
projected_old_revenue = total_tickets_prior * fine_amount

# For the year after the price increase, revenue would be based on the new repayment rate
projected_revenue_after = total_tickets_prior * (repayment_rate_after / 100) * fine_amount

# Calculate the change in revenue
change_in_revenue = projected_revenue_after - projected_old_revenue

# Display the results
print(f"Total Tickets Issued in 2019: {total_tickets_prior}")
print(f"Payments Made in 2019: {payments_made_prior}")
print(f"Repayment Rate in 2019: {repayment_rate_prior:.2f}%")
print(f"Projected Old Revenue: ${projected_old_revenue:.2f}")
print(f"Total Tickets Issued in 2021: {total_tickets_after}")
print(f"Payments Made in 2021: {payments_made_after}")
print(f"Repayment Rate in 2021: {repayment_rate_after:.2f}%")
print(f"Projected Revenue After Price Increase: ${projected_revenue_after:.2f}")
print(f"Change in Revenue: ${change_in_revenue:.2f}")
```

```
Total Tickets Issued in 2019: 0
Payments Made in 2019: 0
Repayment Rate in 2019: 0.00%
Projected Old Revenue: $0.00
Total Tickets Issued in 2021: 0
Payments Made in 2021: 0
Repayment Rate in 2021: 0.00%
Projected Revenue After Price Increase: $0.00
Change in Revenue: $0.00
```

Question 5

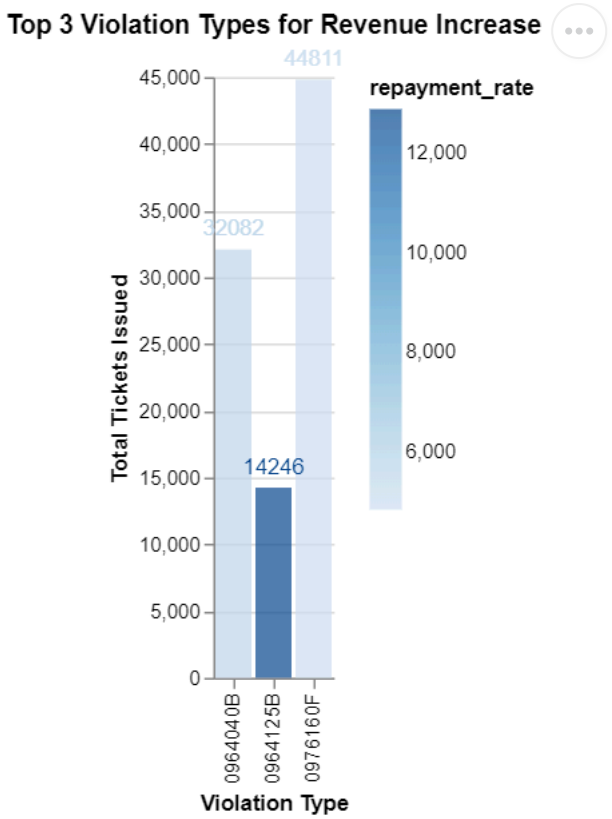
```
import pandas as pd
import altair as alt

# Assuming you have already calculated repayment rates for prior_year_tickets and after_year_t
# Create a DataFrame for repayment rates
repayment_data = pd.DataFrame({
```

```
    'Year': [2019, 2021],  # Years of interest
    'Repayment_Rate': [repayment_rate_prior, repayment_rate_after]  # Corresponding repayment
})

# Create the Altair plot
base = alt.Chart(repayment_data).encode(
    x=alt.X('Year:O', title='Year'),
    y=alt.Y('Repayment_Rate:Q', title='Repayment Rate (%)')
)

line = base.mark_line(point=True).encode(
    color=alt.value('blue'),
)

# Add a vertical line for the new policy introduction in 2020
policy_line = alt.Chart(pd.DataFrame({'x': [2020]})).mark_rule(color='red').encode(
    x='x:O'
)

# Combine line plot and policy line
final_plot = line + policy_line

# Show the plot
final_plot.properties(
    title='Repayment Rates for Missing City Sticker Tickets'
).configure_view(
    stroke=None
)
```



Question 6

```
import pandas as pd
import altair as alt

# Assuming `data` is your DataFrame containing all the tickets
# Step 1: Calculate the total tickets and total payments for each violation type
violation_summary = data.groupby('violation_code').agg(
    total_tickets=('ticket_number', 'count'),
    total_payments=('total_payments', 'sum')
).reset_index()

# Step 2: Calculate the repayment rate
violation_summary['repayment_rate'] = (violation_summary['total_payments'] / violation_summary

# Step 3: Sort by effective revenue potential and select the top three
violation_summary['effective_revenue'] = violation_summary['total_tickets'] * violation_summar
top_violations = violation_summary.sort_values(by='effective_revenue', ascending=False).head(3

# Step 4: Create a plot to visualize the top violation types
base = alt.Chart(top_violations).encode(
    x=alt.X('violation_code:N', title='Violation Type'),
    y=alt.Y('total_tickets:Q', title='Total Tickets Issued'),
    color=alt.Color('repayment_rate:Q', scale=alt.Scale(scheme='blues')),
    tooltip=['violation_code', 'total_tickets', 'repayment_rate']
)

bar = base.mark_bar().encode(
    opacity=alt.value(0.7)
)

text = base.mark_text(
    align='center',
    baseline='middle',
    dy=-10  # Adjust y position for text
).encode(
    text='total_tickets:Q'
)

final_plot = bar + text

# Show the plot
final_plot.properties(
    title='Top 3 Violation Types for Revenue Increase'
).configure_view(
    stroke=None
)
```



Headlines and sub-messages

Question 1

```
import pandas as pd

# Step 1: Create a new DataFrame with necessary calculations
violation_analysis = data.groupby('violation_description').agg(
    total_tickets=('ticket_number', 'count'),
    total_payments=('total_payments', 'sum'),
    avg_fine=('fine_level1_amount', 'mean')
).reset_index()

# Step 2: Calculate the fraction of tickets paid
violation_analysis['repayment_rate'] = violation_analysis['total_payments'] / violation_analys

# Step 3: Sort the DataFrame based on total tickets issued
violation_analysis = violation_analysis.sort_values(by='total_tickets', ascending=False)

# Step 4: Print the rows for the 5 most common violation descriptions
top_violations = violation_analysis.head(5)
print(top_violations[['violation_description', 'repayment_rate', 'avg_fine']])
```

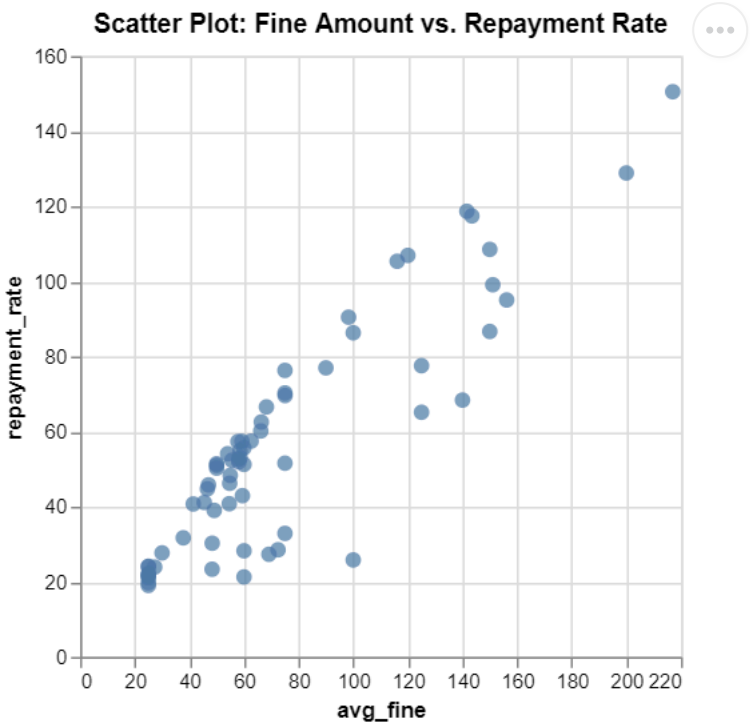
```
                      violation_description  repayment_rate   avg_fine
23   EXPIRED PLATES OR TEMPORARY REGISTRATION       48.344820  54.968869
101                            STREET CLEANING       54.110412  54.004249
90                 RESIDENTIAL PERMIT PARKING       62.549918  66.338302
```

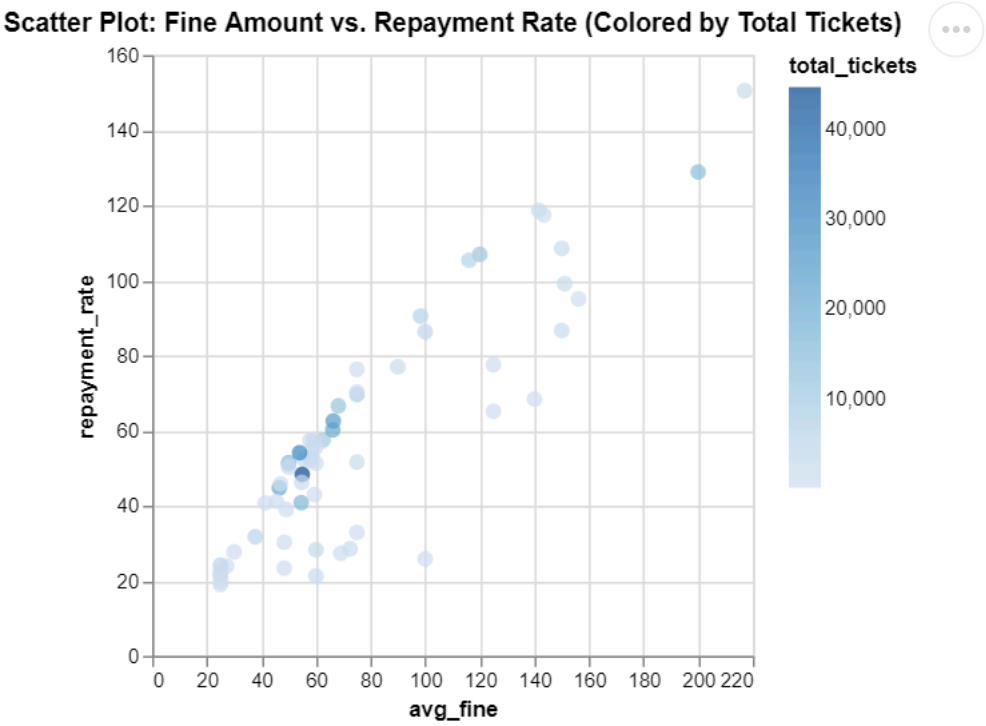
```
19   EXP. METER NON-CENTRAL BUSINESS DISTRICT       44.753963  46.598058
81         PARKING/STANDING PROHIBITED ANYTIME      60.162839  66.142864
```

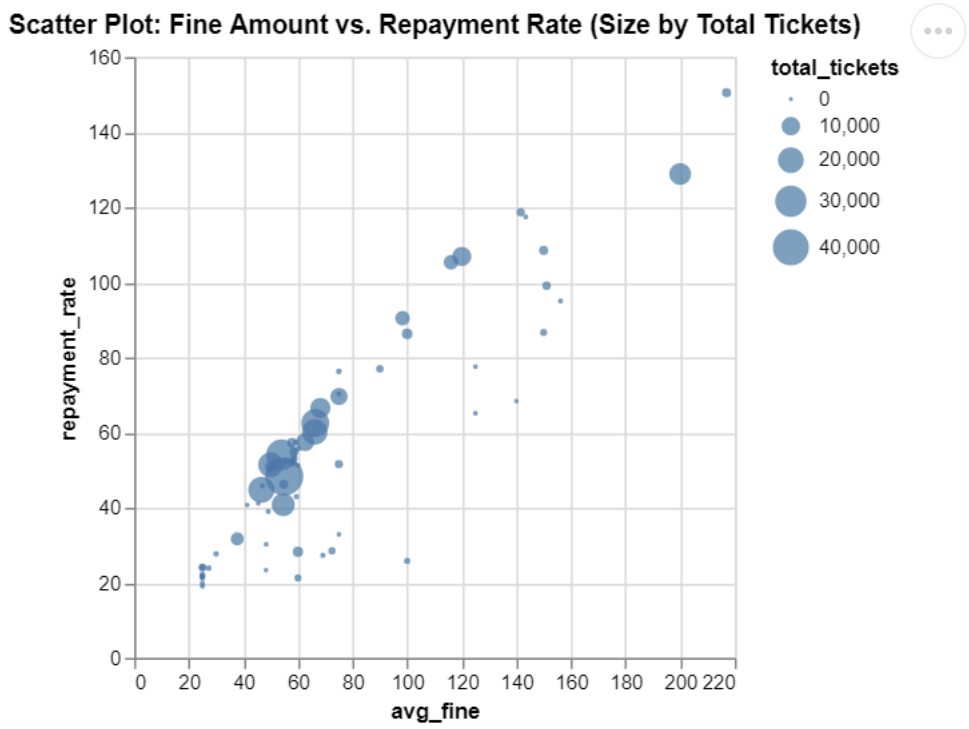
Question 2

```
# Step 1: Create a new DataFrame with necessary calculations
violation_analysis = data.groupby('violation_description').agg(
    total_tickets=('ticket_number', 'count'),
    total_payments=('total_payments', 'sum'),
    avg_fine=('fine_level1_amount', 'mean')
).reset_index()

# Step 2: Calculate the fraction of tickets paid
violation_analysis['repayment_rate'] = violation_analysis['total_payments'] / violation_analys

# Step 3: Filter violations that appear at least 100 times
violation_analysis_filtered = violation_analysis[violation_analysis['total_tickets'] >= 100]

# Step 4: Exclude the outlier (let's assume we know the outlier fine amount)
outlier_fine = violation_analysis_filtered['avg_fine'].max()  # Replace with known outlier fin
violation_analysis_filtered = violation_analysis_filtered[violation_analysis_filtered['avg_fin

# Step 5: Create scatter plots
# Plot 1: Basic Scatter Plot
scatter_plot = alt.Chart(violation_analysis_filtered).mark_circle(size=60).encode(
    x='avg_fine:Q',
    y='repayment_rate:Q',
    tooltip=['violation_description:N', 'avg_fine:Q', 'repayment_rate:Q']
).properties(
    title='Scatter Plot: Fine Amount vs. Repayment Rate'
)

# Plot 2: Scatter Plot with Color Encoding by Total Tickets
scatter_plot_colored = alt.Chart(violation_analysis_filtered).mark_circle(size=60).encode(
    x='avg_fine:Q',
    y='repayment_rate:Q',
    color='total_tickets:Q',
    tooltip=['violation_description:N', 'avg_fine:Q', 'repayment_rate:Q', 'total_tickets:Q']
).properties(
    title='Scatter Plot: Fine Amount vs. Repayment Rate (Colored by Total Tickets)'
)

# Plot 3: Scatter Plot with Size Encoding by Total Tickets
scatter_plot_sized = alt.Chart(violation_analysis_filtered).mark_circle().encode(
    x='avg_fine:Q',
    y='repayment_rate:Q',
    size='total_tickets:Q',
    tooltip=['violation_description:N', 'avg_fine:Q', 'repayment_rate:Q', 'total_tickets:Q']
).properties(
    title='Scatter Plot: Fine Amount vs. Repayment Rate (Size by Total Tickets)'
)

# Display the plots
```

```
scatter_plot.show()
scatter_plot_colored.show()
scatter_plot_sized.show()
```







Question 3

Recommended Plot: Scatter Plot with Color Coding Why This Plot?

Clear Insight: It shows the relationship between fine amounts and repayment rates. Volume Highlight: Color coding indicates how many tickets were issued for each violation type. Easy to Read: Simple and accessible for quick understanding.

This plot helps the City Clerk see how higher fines may impact repayment rates and where revenue opportunities lie.

Understanding the structure of the data and summarizing it

Question 1

```
# Filter for violations with at least 100 citations
violation_counts = data['violation_description'].value_counts()
common_violations = violation_counts[violation_counts >= 100].index

# Filter the original data to only include those common violations
common_violations_data = data[data['violation_description'].isin(common_violations)]

# Calculate the increase in fine for unpaid tickets
common_violations_data['fine_increase'] = common_violations_data['fine_level2_amount'] - commo

# Find violations that do not double in price
non_doubling_violations = common_violations_data[common_violations_data['fine_increase'] < com

# Group by violation description and calculate the amount each ticket increases if unpaid
non_doubling_summary = non_doubling_violations.groupby('violation_description')['fine_increase
```

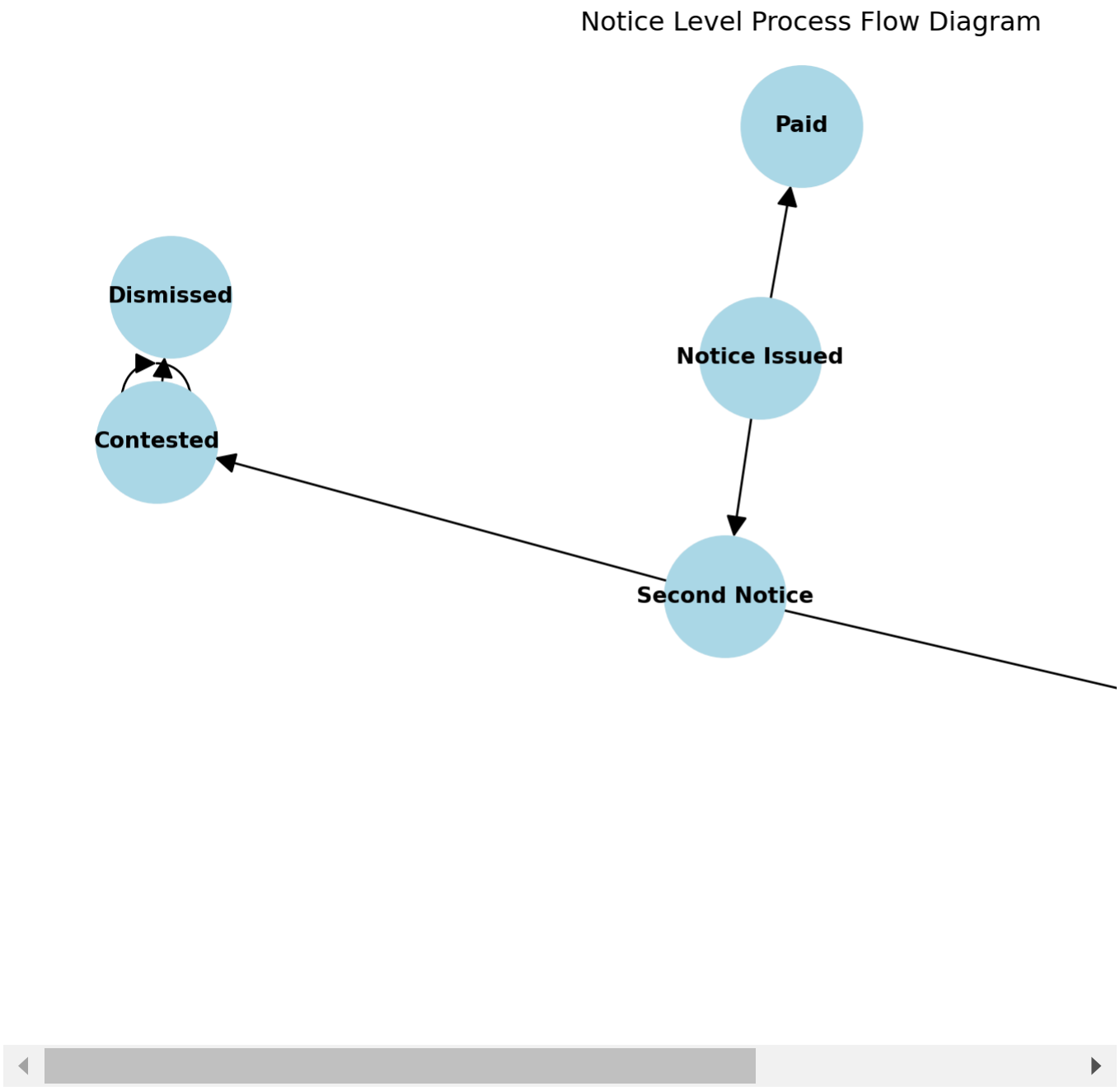
```
# Display the results
print(non_doubling_summary)
```

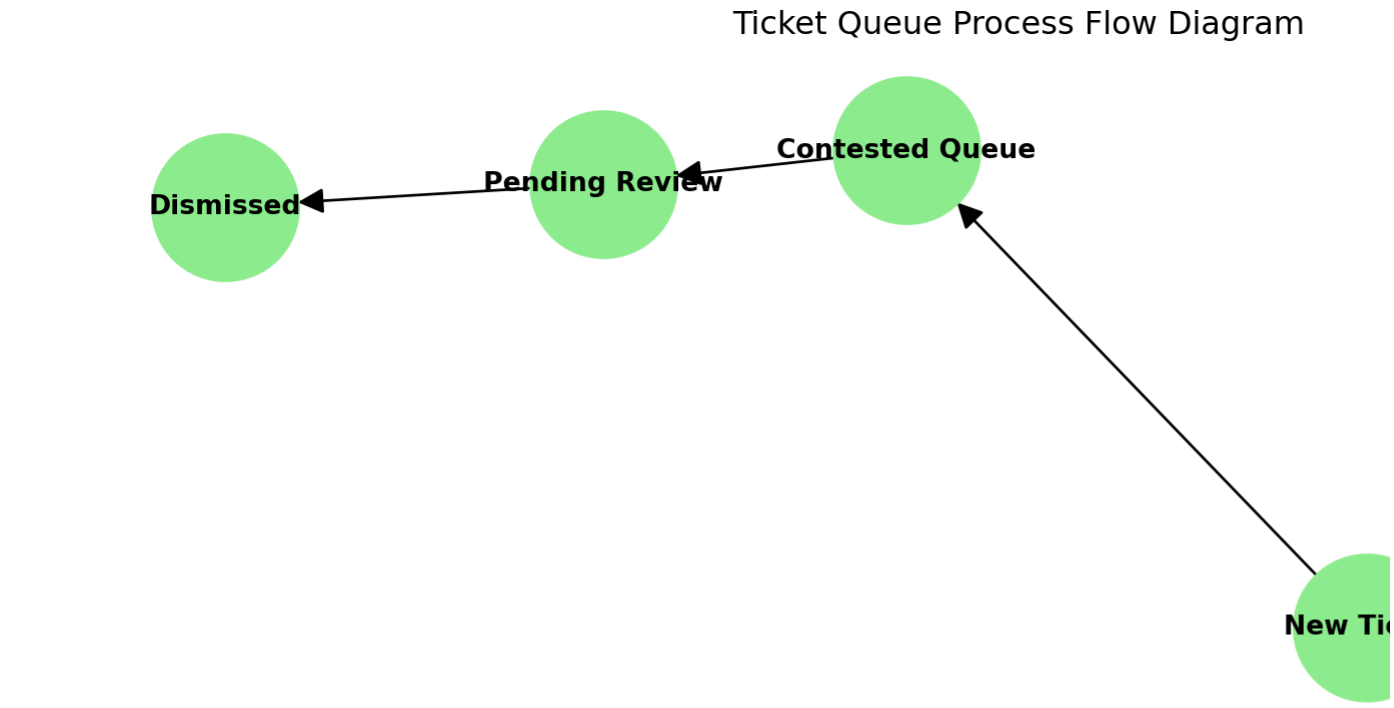
```
                     violation_description  fine_increase
0      BLOCK ACCESS/ALLEY/DRIVEWAY/FIRELANE          100.0
1                     DISABLED PARKING ZONE           50.0
2  NO CITY STICKER VEHICLE OVER 16,000 LBS.          275.0
3   OBSTRUCTED OR IMPROPERLY TINTED WINDOWS            0.0
4                       PARK OR BLOCK ALLEY          100.0
5                PARK/STAND ON BICYCLE PATH          100.0
6     SMOKED/TINTED WINDOWS PARKED/STANDING            0.0

C:\Users\Shreya Work\AppData\Local\Temp\ipykernel_14380\3136056192.py:9:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  common_violations_data['fine_increase'] = common_violations_data['fine_level2_amount'] -
common_violations_data['fine_level1_amount']
```

Question 2

```
import matplotlib.pyplot as plt
import networkx as nx

# Create a directed graph for Notice Level Process
G = nx.DiGraph()

# Add nodes and edges
G.add_edges_from([
    ("Notice Issued", "Second Notice"),
    ("Notice Issued", "Paid"),
    ("Second Notice", "Final Notice"),
    ("Second Notice", "Contested"),
    ("Final Notice", "Court Hearing"),
    ("Final Notice", "Collections"),
    ("Contested", "Contested"),
    ("Contested", "Dismissed")
])

# Draw the graph
pos = nx.spring_layout(G)
plt.figure(figsize=(10, 6))
nx.draw(G, pos, with_labels=True, arrows=True, node_size=3000, node_color='lightblue', font_si
plt.title("Notice Level Process Flow Diagram")
plt.show()
```

```
### Ticket Queue Process Flow Diagram


# Create a directed graph for Ticket Queue Process
G = nx.DiGraph()

# Add nodes and edges
G.add_edges_from([
    ("New Ticket", "Contested Queue"),
    ("New Ticket", "Paid Queue"),
    ("Contested Queue", "Pending Review"),
    ("Pending Review", "Dismissed")
])

# Draw the graph
pos = nx.spring_layout(G)
plt.figure(figsize=(10, 6))
nx.draw(G, pos, with_labels=True, arrows=True, node_size=3000, node_color='lightgreen', font_s
plt.title("Ticket Queue Process Flow Diagram")
plt.show()
```


Notice Level Process Flow Diagram
Paid
Dismissed
Notice Issued
Contested
Second Notice




```
import pandas as pd
import altair as alt

# Sample the data to avoid exceeding the row limit
sampled_data = data.sample(n=5000, random_state=42)  # Adjust n as needed

# Identify the ten most common violations
violation_counts = sampled_data['violation_description'].value_counts()
top_violations = violation_counts.head(10).index.tolist()

# Create a new column to categorize violations
sampled_data['violation_label'] = sampled_data['violation_description'].apply(lambda x: x if x

# Create the scatter plot with labels
scatter_plot_1 = alt.Chart(sampled_data).mark_point().encode(
    x='fine_level1_amount:Q',
    y='fraction_paid:Q',
    color='violation_label:N',
```

```
    tooltip=['violation_description:N', 'fraction_paid:Q']
).properties(
    title='Scatter Plot of Fine Amount vs. Fraction Paid with Top Violations'
).interactive()

# Add text labels
text_labels_1 = scatter_plot_1.mark_text(
    align='left',
    baseline='middle',
    dx=5,
    fontSize=10
).encode(
    text='violation_label:N'
)

# Combine the scatter plot with labels
final_plot_1 = scatter_plot_1 + text_labels_1
final_plot_1
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
File ~\AppData\Roaming\Python\Python312\site-packages\IPython\core\formatters.py:977, in MimeBur
    974     method = get_real_method(obj, self.print_method)
    976     if method is not None:
--> 977         return method(include=include, exclude=exclude)
    978     return None
    979 else:

File ~\AppData\Local\Programs\Python\Python312\Lib\site-packages\altair\vegalite\v5\api.py:3417,
   3415 else:
   3416     if renderer := renderers.get():
-> 3417         return renderer(dct)

File ~\AppData\Local\Programs\Python\Python312\Lib\site-packages\altair\utils\display.py:225, ir
    223 kwargs = self.kwargs.copy()
    224 kwargs.update(**metadata, output_div=self.output_div)
--> 225 return spec_to_mimebundle(spec, format="html", **kwargs)

File ~\AppData\Local\Programs\Python\Python312\Lib\site-packages\altair\utils\mimebundle.py:144,
    134     return _spec_to_mimebundle_with_engine(
    135         spec,
    136         cast(Literal["png", "svg", "pdf", "vega"], format),
   (...)
    141         **kwargs,
    142     )
    143 elif format == "html":
--> 144     html = spec_to_html(
    145         spec,
    146         mode=internal_mode,
    147         vega_version=vega_version,
    148         vegaembed_version=vegaembed_version,
    149         vegalite_version=vegalite_version,
    150         embed_options=embed_options,
```

```
    151         **kwargs,
    152     )
    153     return {"text/html": html}
    154 elif format == "vega-lite":
```

```
File ~\AppData\Local\Programs\Python\Python312\Lib\site-packages\altair\utils\html.py:303, in sp
    299     msg = f"Invalid template: {jinja_template}"
    300     raise ValueError(msg)
    302 return jinja_template.render(
--> 303     spec=json.dumps(spec, **json_kwds),
    304     embed_options=json.dumps(embed_options),
    305     mode=mode,
    306     vega_version=vega_version,
    307     vegalite_version=vegalite_version,
    308     vegaembed_version=vegaembed_version,
    309     base_url=base_url,
    310     output_div=output_div,
    311     fullhtml=fullhtml,
    312     requirejs=requirejs,
    313     **render_kwargs,
    314 )
```

```
File ~\AppData\Local\Programs\Python\Python312\Lib\json\__init__.py:231, in dumps(obj, skipkeys,
    226 # cached encoder
    227 if (not skipkeys and ensure_ascii and
    228     check_circular and allow_nan and
    229     cls is None and indent is None and separators is None and
    230     default is None and not sort_keys and not kw):
--> 231     return _default_encoder.encode(obj)
    232 if cls is None:
    233     cls = JSONEncoder
```

```
File ~\AppData\Local\Programs\Python\Python312\Lib\json\encoder.py:200, in JSONEncoder.encode(se
    196         return encode_basestring(o)
    197 # This doesn't pass the iterator directly to ''.join() because the
    198 # exceptions aren't as detailed.  The list call should be roughly
    199 # equivalent to the PySequence_Fast that ''.join() would do.
--> 200 chunks = self.iterencode(o, _one_shot=True)
    201 if not isinstance(chunks, (list, tuple)):
    202     chunks = list(chunks)
```

```
File ~\AppData\Local\Programs\Python\Python312\Lib\json\encoder.py:258, in JSONEncoder.iterencod
    253 else:
    254     _iterencode = _make_iterencode(
    255         markers, self.default, _encoder, self.indent, floatstr,
    256         self.key_separator, self.item_separator, self.sort_keys,
    257         self.skipkeys, _one_shot)
--> 258 return _iterencode(o, 0)
```

```
File ~\AppData\Local\Programs\Python\Python312\Lib\json\encoder.py:180, in JSONEncoder.default(s
    161 def default(self, o):
    162     """Implement this method in a subclass such that it returns
    163     a serializable object for ``o``, or calls the base implementation
    164     (to raise a ``TypeError``).
```

```
   (...)
    178
    179     """
--> 180     raise TypeError(f'Object of type {o.__class__.__name__} '
    181                     f'is not JSON serializable')

TypeError: Object of type Period is not JSON serializable
```

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

Extra Credit

Question 1

```
# Step 1: Group by 'violation_code' and count unique violation descriptions
violation_counts = data.groupby('violation_code')['violation_description'].nunique().reset_ind
violation_counts.columns = ['violation_code', 'unique_descriptions']

# Step 2: Identify violation codes with multiple descriptions
multiple_descriptions = violation_counts[violation_counts['unique_descriptions'] > 1]

# Step 3: Find the most common description for each violation code
most_common_descriptions = data.groupby(['violation_code', 'violation_description']).size().re
most_common = most_common_descriptions.loc[most_common_descriptions.groupby('violation_code')[

# Step 4: Merge to get the most common description for those codes with multiple descriptions
most_common = most_common.merge(multiple_descriptions, on='violation_code', how='inner')

# Create a new column in the original data to store the most common description
data = data.merge(most_common[['violation_code', 'violation_description']], on='violation_code

# Step 5: Print the three codes with the most observations
top_3_codes = most_common['violation_code'].value_counts().head(3)
print("Three violation codes with the most observations and multiple descriptions:")
print(top_3_codes)
```

```
Three violation codes with the most observations and multiple descriptions:
violation_code
0964040B    1
0964041B    1
0964070     1
Name: count, dtype: int64
```