PS2_Mahnoor Arif

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

```
tickets_data = pd.read_csv('C:\\Users\\arifm\\OneDrive\\Documents\\GitHub\\student30538\\PS2\\parl
```

```
C:\Users\arifm\AppData\Local\Temp\ipykernel_8892\413930792.py:1: DtypeWarning: Columns (7) have mixed types. Specify dtype option on import or set low_memory=False.
  tickets_data =
pd.read_csv('C:\\Users\\arifm\\OneDrive\\Documents\\GitHub\\student30538\\PS2\\parking_tickets_one_percent.csv')
```

```
#!pip install altair_saver
```

##Data cleaning continued (15 points)

```
#1.1
# Step 2: Define a function to count NAs for each column
def count_na_per_column(dataframe):
    # Create a dictionary where keys are column names and values are the number of NAs
    na_counts = dataframe.isna().sum()

    # Convert the result to a DataFrame called countNA
    na_df = pd.DataFrame({
        'Variable': na_counts.index,   # Column names
        'NA_Count': na_counts.values   # NA counts
    })

    return na_df

# Step 3: Apply the function to the parking tickets data frame
na_report = count_na_per_column(tickets_data)

# Step 4: Print the results
print(na_report)
```

```
                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

```
#1.2
#The three variables with most missing values include: notice_level, hearing_disposition and zip_
```

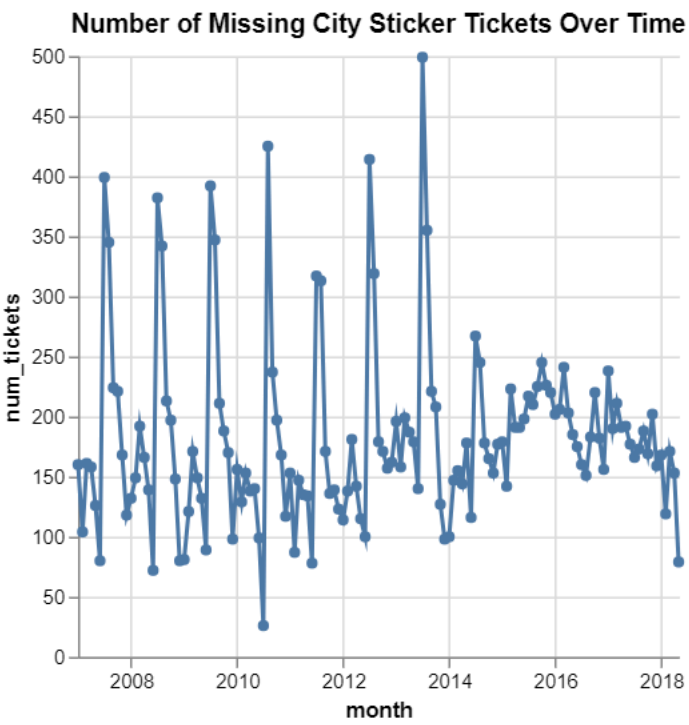
```
#1.3
## New code: 0964125B
## old code: 0964125
```

```
#1.4
#NO CITY STICKER OR IMPROPER DISPLAY    $ 120

#NO CITY STICKER VEHICLE UNDER/EQUAL TO 16,000 LBS. $200
```

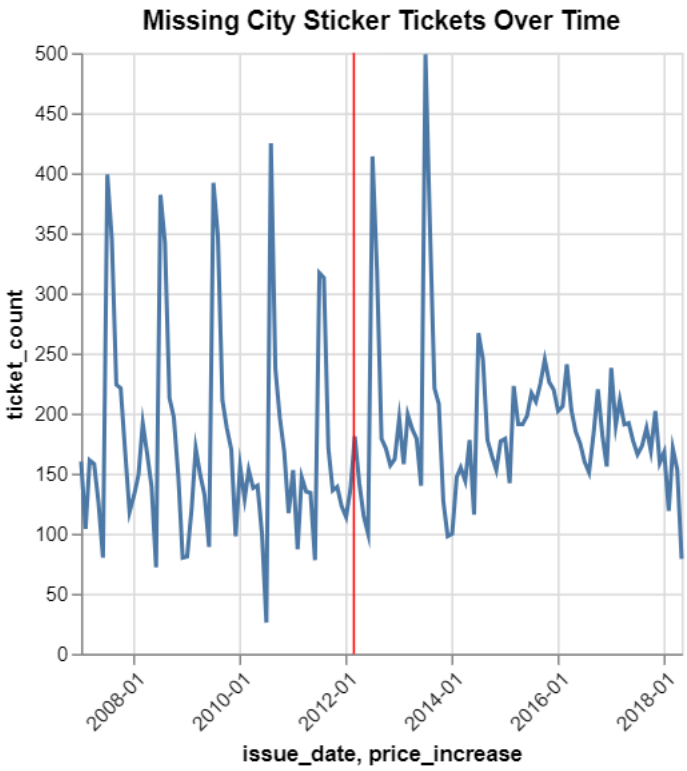
```
##2.1
#import pandas as pd
#import altair as alt

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

# Step 1: Combine old and new violation codes (0964125 and 0964125B)
tickets_data['violation_code_combined'] = tickets_data['violation_code'].replace({'0964125B': '096

# Step 2: Filter the data for the missing city sticker violation codes only
df_missing_sticker = tickets_data[tickets_data['violation_code_combined'] == '0964125']

# Step 3: Group by month and count the number of missing city sticker tickets
df_missing_sticker['month'] = df_missing_sticker['issue_date'].dt.to_period('M')
tickets_by_month = df_missing_sticker.groupby('month').size().reset_index(name='num_tickets')

# Convert 'month' Period objects to string format for Altair compatibility
```

```
tickets_by_month['month'] = tickets_by_month['month'].astype(str)

# Step 4: Plot the number of tickets over time using Altair
import altair as alt

chart = alt.Chart(tickets_by_month).mark_line(point=True).encode(
    x='month:T',  # Treat 'month' as time type
    y='num_tickets:Q',
    tooltip=['month:T', 'num_tickets:Q']
).properties(
    title='Number of Missing City Sticker Tickets Over Time'
).configure_axis(
    grid=True
).configure_view(
    stroke='transparent'
)

# Display the chart
chart.show()
```

```
C:\Users\arifm\AppData\Local\Temp\ipykernel_8892\1099920220.py:15: 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
  df_missing_sticker['month'] = df_missing_sticker['issue_date'].dt.to_period('M')
```



```
##2.2
#import pandas as pd
#import altair as alt

# Ensure issue_date is in datetime format
tickets_data['issue_date'] = pd.to_datetime(tickets_data['issue_date'])

# Combine the two violation codes
old_code = '0964125'
new_code = '0964125B'
tickets_data['city_sticker_violation'] = tickets_data['violation_code'].isin([old_code, new_code]

# Filter data for city sticker violations
filtered_data = tickets_data[tickets_data['city_sticker_violation']]

# Group by month for city sticker violations
df_monthly = filtered_data.groupby(filtered_data['issue_date'].dt.to_period('M')).size().reset_in

# Convert the period back to a timestamp for plotting
df_monthly['issue_date'] = df_monthly['issue_date'].dt.to_timestamp()

# Plot the number of tickets over time and add custom date labels
price_increase_date = '2012-02-24'  # Price increase occured in 3/6/2012.
chart = alt.Chart(df_monthly).mark_line().encode(
    x=alt.X('issue_date:T', axis=alt.Axis(format='%Y-%m', labelAngle=-45)),  # Custom date format
    y='ticket_count'
).properties(
    title='Missing City Sticker Tickets Over Time'
)

# Add a vertical line to indicate the price increase date
rule = alt.Chart(pd.DataFrame({'price_increase': [price_increase_date]})).mark_rule(color='red').
    x='price_increase:T'
)

# Combine the line chart and the rule
final_chart = chart + rule

final_chart.display()

##Help page: I filtered for the exact price change year using pivot tables on Excel.
```




```
##2.3
# Ticket count in the 1% sample
ticket_count_sample = 175  # The number of tickets in the 1% sample
ticket_price_before = 120   # Assuming the price was $120 before the increase
ticket_price_after = 200    # The new price after the increase
sample_percentage = 0.01    # 1% sample of the total data

# Step 1: Calculate revenue before the price increase (for 1% sample)
revenue_before_sample = ticket_count_sample * ticket_price_before

# Step 2: Calculate revenue after the price increase (for 1% sample)
revenue_after_sample = ticket_count_sample * ticket_price_after

# Step 3: Project total revenue increase (scale from 1% sample to 100% of tickets)
total_revenue_before = revenue_before_sample / sample_percentage
total_revenue_after = revenue_after_sample / sample_percentage

# Step 4: Calculate projected increase in revenue
projected_increase = total_revenue_after - total_revenue_before

# Step 5: Compare with the claimed $16 million increase
print(f"Projected Revenue Before Increase (scaled): ${total_revenue_before:,.2f}")
print(f"Projected Revenue After Increase (scaled): ${total_revenue_after:,.2f}")
print(f"Projected Revenue Increase: ${projected_increase:,.2f}")
print(f"City Clerk's Claimed Revenue Increase: $16,000,000")

# Compare if projected increase matches the claim
if projected_increase >= 16_000_000:
    print("The projected revenue increase meets or exceeds the City's claim of $16 million.")
```

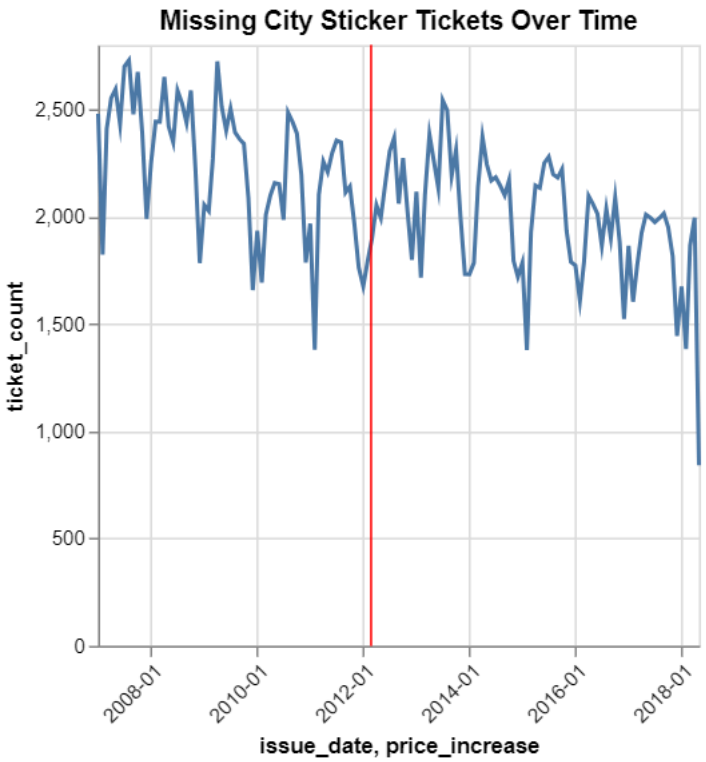
```
else:
    print(f"The projected increase falls short by ${16_000_000 - projected_increase:,.2f}.")

##Explanation
#Projected Revenue Before Increase (scaled): $2,100,000.00
#Projected Revenue After Increase (scaled): $3,500,000.00
#Projected Revenue Increase: $1,400,000.00
#City Clerk's Claimed Revenue Increase: $16,000,000
#The projected increase falls short by $14,600,000.00.
```

```
Projected Revenue Before Increase (scaled): $2,100,000.00
Projected Revenue After Increase (scaled): $3,500,000.00
Projected Revenue Increase: $1,400,000.00
City Clerk's Claimed Revenue Increase: $16,000,000
The projected increase falls short by $14,600,000.00.
```

```
##2.4
#import pandas as pd
#import altair as alt

# Ensure issue_date is in datetime format
tickets_data['issue_date'] = pd.to_datetime(tickets_data['issue_date'])

# Combine the two violation codes
old_code = '0964125'
new_code = '0964125B'
tickets_data['city_sticker_violation'] = tickets_data['violation_code'].isin([old_code, new_code]

# Define the price increase date
price_increase_date = pd.Timestamp('2012-02-24')

# Step 1: Separate data into two periods: before and after the price increase
before_increase = tickets_data[tickets_data['issue_date'] < price_increase_date]
after_increase = tickets_data[tickets_data['issue_date'] >= price_increase_date]

# Step 2: Calculate repayment rates for each period
repayment_before = before_increase['city_sticker_violation'].mean()  # Assuming binary (1 for payı
repayment_after = after_increase['city_sticker_violation'].mean()

# Step 3: Calculate the number of tickets issued before and after the price increase
tickets_before = before_increase['city_sticker_violation'].count()
tickets_after = after_increase['city_sticker_violation'].count()

# Assuming the number of tickets issued after the price increase remains the same as before

tickets_after_assumed = tickets_before

# Step 4: Calculate revenue for both periods
# Assuming each ticket has a fixed fine amount
fine_amount = 200
```

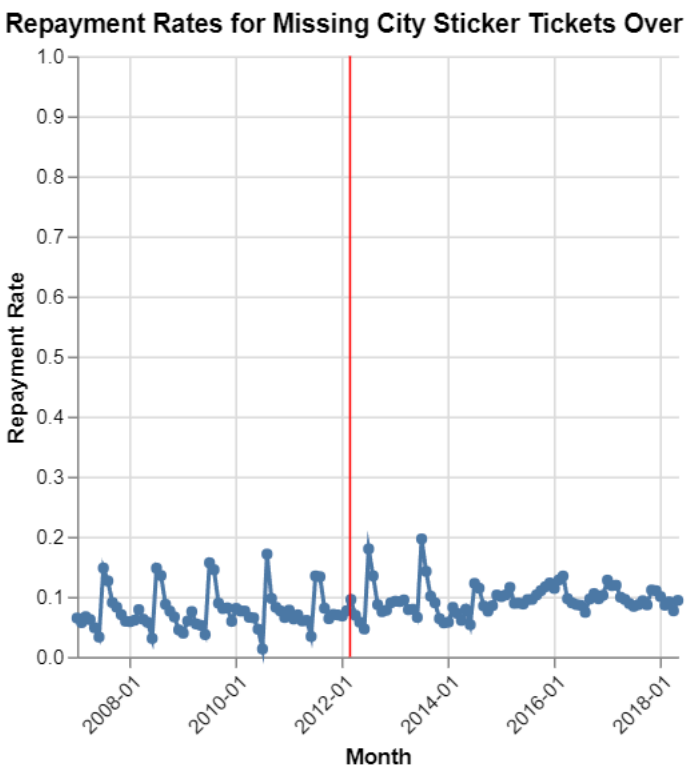
```
revenue_before = repayment_before * tickets_before * fine_amount
revenue_after = repayment_after * tickets_after_assumed * fine_amount

# Step 5: Calculate the change in revenue
change_in_revenue = revenue_after - revenue_before

# Print results
print(f"Repayment Rate Before Price Increase: {repayment_before:.2%}")
print(f"Repayment Rate After Price Increase: {repayment_after:.2%}")
print(f"Revenue Before Price Increase: ${revenue_before:.2f}")
print(f"Revenue After Price Increase (assuming unchanged ticket count): ${revenue_after:.2f}")
print(f"Change in Revenue: ${change_in_revenue:.2f}")

# Plot the number of tickets over time
df_monthly = tickets_data.groupby(tickets_data['issue_date'].dt.to_period('M')).size().reset_inde
df_monthly['issue_date'] = df_monthly['issue_date'].dt.to_timestamp()

# Plotting
chart = alt.Chart(df_monthly).mark_line().encode(
    x=alt.X('issue_date:T', axis=alt.Axis(format='%Y-%m', labelAngle=-45)),  # Custom date format
    y='ticket_count'
).properties(
    title='Missing City Sticker Tickets Over Time'
)

# Add a vertical line to indicate the price increase date
rule = alt.Chart(pd.DataFrame({'price_increase': [price_increase_date]})).mark_rule(color='red').
    x='price_increase:T'
)

# Combine the line chart and the rule
final_chart = chart + rule

# Display the chart
final_chart.display()



#Repayment Rate Before Price Increase: 7.76%
#Repayment Rate After Price Increase: 9.57%
#Revenue Before Price Increase: $2150800.00
#Revenue After Price Increase (assuming unchanged ticket count): $2651089.80
#Change in Revenue: $500289.80
```

```
Repayment Rate Before Price Increase: 7.76%
Repayment Rate After Price Increase: 9.57%
Revenue Before Price Increase: $2150800.00
Revenue After Price Increase (assuming unchanged ticket count): $2651089.80
Change in Revenue: $500289.80
```




```
##2.5
#import pandas as pd
#import altair as alt


# Ensure issue_date is in datetime format
tickets_data['issue_date'] = pd.to_datetime(tickets_data['issue_date'])

# Combine the two violation codes
old_code = '0964125'
new_code = '0964125B'
tickets_data['city_sticker_violation'] = tickets_data['violation_code'].isin([old_code, new_code]

# Define the price increase date (policy introduction date)
price_increase_date = pd.Timestamp('2012-02-24')

# Step 1: Calculate repayment rates by month
tickets_data['paid'] = tickets_data['city_sticker_violation'].astype(int)  # Assuming 1 for payme

# Grouping by month to calculate repayment rates
monthly_data = tickets_data.groupby(tickets_data['issue_date'].dt.to_period('M')).agg(
    total_tickets=('city_sticker_violation', 'size'),
    total_paid=('paid', 'sum')
).reset_index()

# Calculate repayment rate
monthly_data['repayment_rate'] = monthly_data['total_paid'] / monthly_data['total_tickets']

# Convert 'month' Period objects to string format for Altair compatibility
```

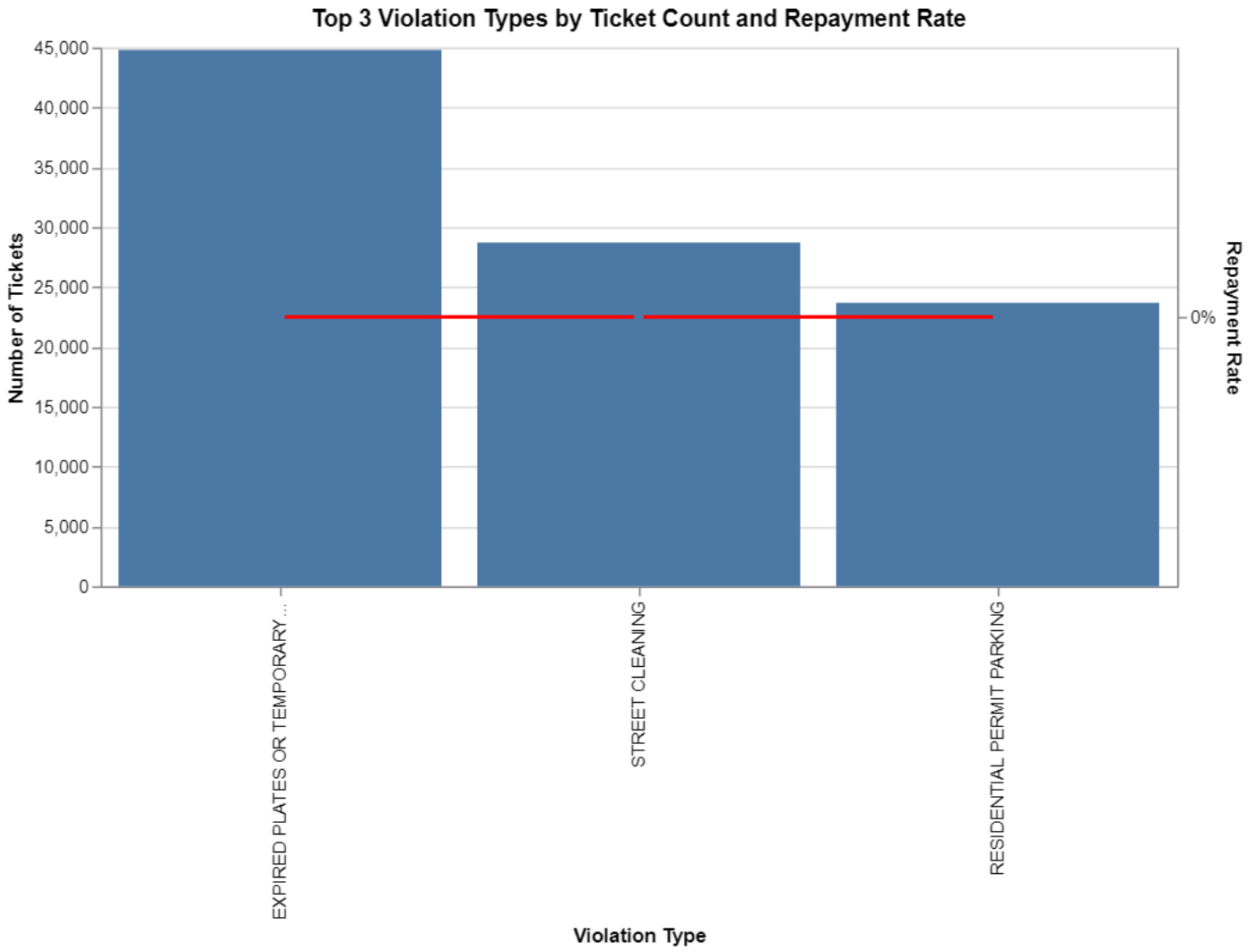
```
monthly_data['issue_date'] = monthly_data['issue_date'].dt.to_timestamp()

# Step 2: Plot repayment rates over time
chart = alt.Chart(monthly_data).mark_line(point=True).encode(
    x=alt.X('issue_date:T', title='Month', axis=alt.Axis(format='%Y-%m', labelAngle=-45)),
    y=alt.Y('repayment_rate:Q', title='Repayment Rate', scale=alt.Scale(domain=[0, 1])),
    tooltip=['issue_date:T', 'repayment_rate:Q']
).properties(
    title='Repayment Rates for Missing City Sticker Tickets Over Time'
)

# Add a vertical line to indicate the price increase date
rule = alt.Chart(pd.DataFrame({'price_increase': [price_increase_date]})).mark_rule(color='red').
    x='price_increase:T'
)

# Combine the line chart and the rule
final_chart = chart + rule

# Apply configurations to the LayerChart
final_chart = final_chart.configure_axis(
    grid=True
).configure_view(
    stroke='transparent'
)

# Display the chart
final_chart.display()

# Interpretation
repayment_rate_mean_before = monthly_data[monthly_data['issue_date'] < price_increase_date]['repay
repayment_rate_mean_after = monthly_data[monthly_data['issue_date'] >= price_increase_date]['repay

print(f"Average Repayment Rate Before Price Increase: {repayment_rate_mean_before:.2%}")
print(f"Average Repayment Rate After Price Increase: {repayment_rate_mean_after:.2%}")


#The graph shows that average repayment rate significantly drops after the price increase
```




```
Average Repayment Rate Before Price Increase: 7.62%
Average Repayment Rate After Price Increase: 9.55%
```

```
#2.6
##The other type of violation types whose prices should be increased should be the ones that are (



# Group by violation description to get the total tickets and the number of paid tickets
violation_summary = tickets_data.groupby('violation_description').agg(
    total_tickets=('violation_description', 'size'),  # Total number of tickets
    total_paid=('paid', 'sum')  # Number of paid tickets
).reset_index()

# Calculate the repayment rate by dividing the number of paid tickets by the total tickets
violation_summary['repayment_rate'] = violation_summary['total_paid'] / violation_summary['total_

# Step 1: Filter for violations with high ticket counts and repayment rates
# To focus on revenue, we sort by total tickets and repayment rates and select the top three viola
top_violations = violation_summary.sort_values(by=['total_tickets', 'repayment_rate'], ascending=

# Step 2: Create a bar chart to visualize the total tickets and repayment rates
bars = alt.Chart(top_violations).mark_bar().encode(
    x=alt.X('violation_description:N', title='Violation Type', sort='-y'),
    y=alt.Y('total_tickets:Q', title='Number of Tickets'),
    tooltip=['violation_description', 'total_tickets']
).properties(
    title='Top 3 Violation Types by Number of Tickets'
)

# Step 3: Create a line chart to visualize the repayment rates of these top violations
```

```
line = alt.Chart(top_violations).mark_line(color='red', point=True).encode(
    x=alt.X('violation_description:N', title='Violation Type', sort='-y'),
    y=alt.Y('repayment_rate:Q', title='Repayment Rate', axis=alt.Axis(format='%')),
    tooltip=['violation_description', 'repayment_rate']
)

# Step 4: Combine the bar and line charts
final_chart = alt.layer(bars, line).resolve_scale(
    y='independent'
).properties(
    width=600,
    title='Top 3 Violation Types by Ticket Count and Repayment Rate'
)

# Display the chart
final_chart.display()

# Show the top 3 violations with ticket counts and repayment rates
print("Top 3 Violation Types with Ticket Counts and Repayment Rates:")
print(top_violations[['violation_description', 'total_tickets', 'repayment_rate']])
```



Top 3 Violation Types with Ticket Counts and Repayment Rates:

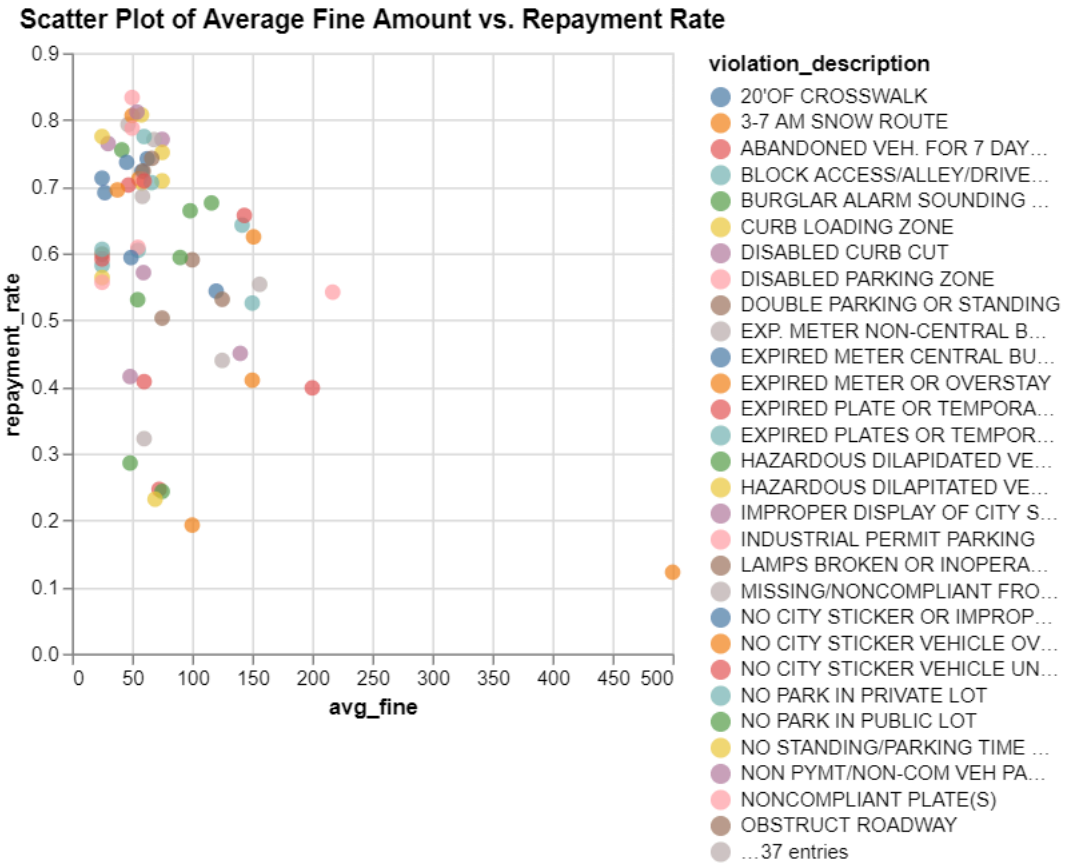
	violation_description	total_tickets	repayment_rate
23	EXPIRED PLATES OR TEMPORARY REGISTRATION	44811	0.0
101	STREET CLEANING	28712	0.0
90	RESIDENTIAL PERMIT PARKING	23683	0.0

```
##3.1

# tickets_data = pd.read_csv('path_to_your_data.csv')

# Create DataFrame
violation_df = tickets_data.groupby('violation_description').agg(
    repayment_rate=('ticket_queue', lambda x: (x == 'Paid').sum() / len(x)),
    avg_fine=('fine_level1_amount', 'mean'),
    total_tickets=('ticket_number', 'count')
).reset_index()

# Filter for violations that appear at least 100 times
violation_df = violation_df[violation_df['total_tickets'] >= 100]

# Sort by total tickets
```

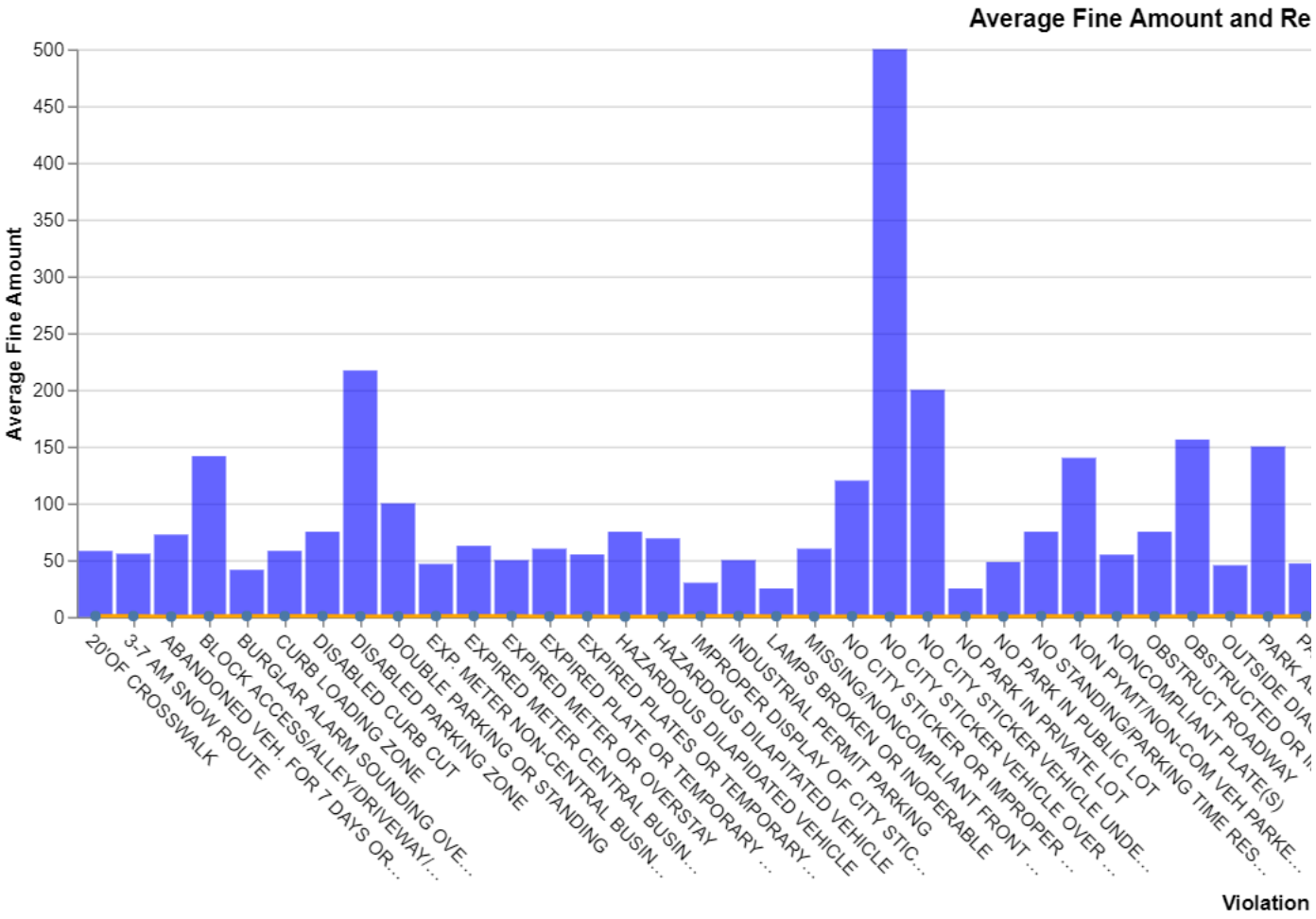
```
violation_df = violation_df.sort_values(by='violation_description', ascending=False)

# Print the top 5 violations
print(violation_df.head(5))
```

```
                     violation_description  repayment_rate    avg_fine  \
118         WRONG DIRECTION OR 12'' FROM CURB        0.774977   25.000000
117                WITHIN 15' OF FIRE HYDRANT        0.675459  116.079620
116       WINDOWS MISSING OR CRACKED BEYOND 6        0.605903   25.000000
110     TWO HEAD LAMPS REQUIRED VISIBLE 1000'        0.591422   25.000000
109  TRUCK,RV,BUS, OR TAXI RESIDENTIAL STREET        0.694926   37.758405

     total_tickets
118           1111
117           6104
116            576
110            443
109           4789
```

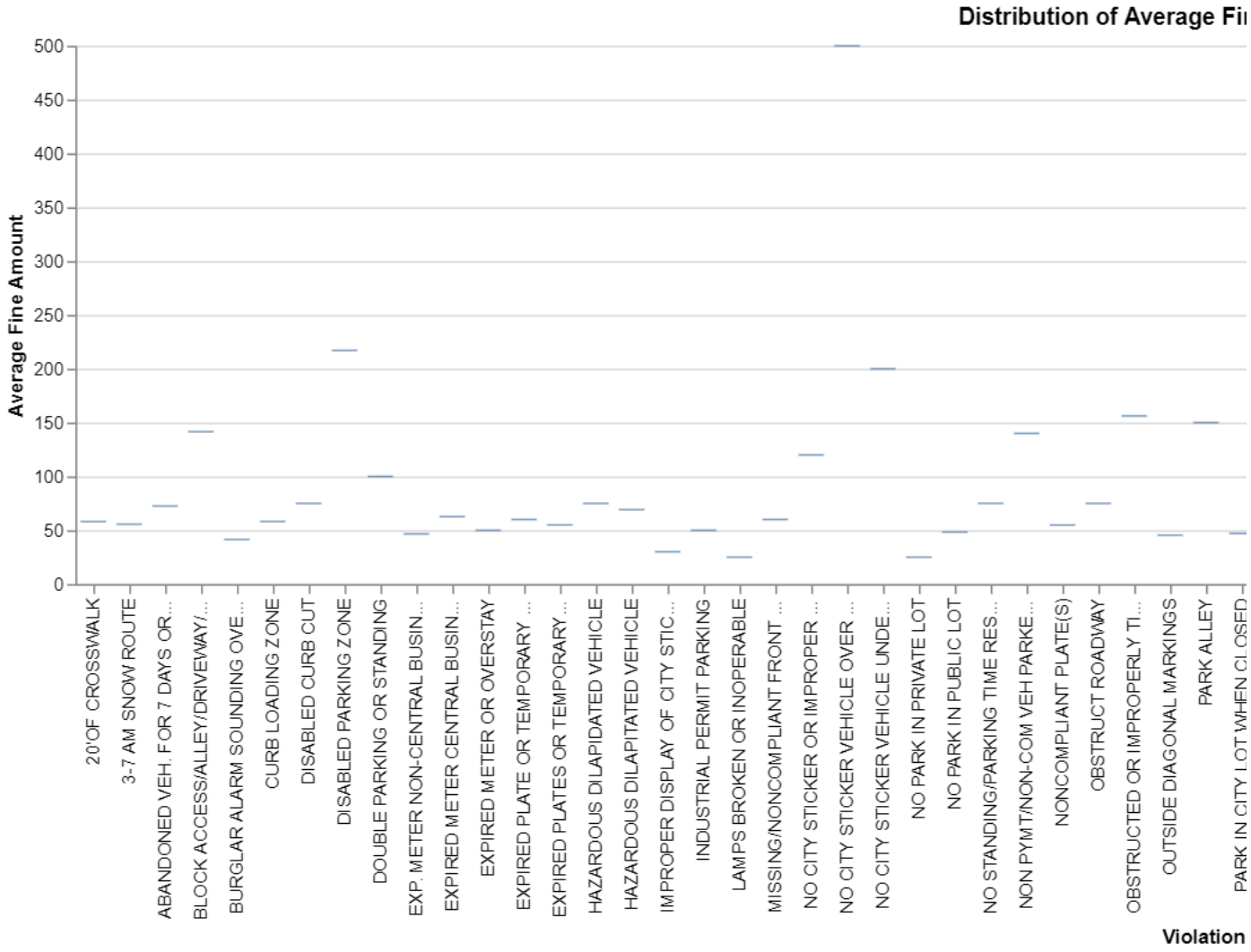
```
#3.2

# Create DataFrame
violation_df = tickets_data.groupby('violation_description').agg(
    repayment_rate=('ticket_queue', lambda x: (x == 'Paid').sum() / len(x)),
    avg_fine=('fine_level1_amount', 'mean'),
    total_tickets=('ticket_number', 'count')
).reset_index()

# Filter for violations that appear at least 100 times
violation_df = violation_df[violation_df['total_tickets'] >= 100]

# Remove outlier (specify the violation to exclude)
violation_df = violation_df[violation_df['violation_description'] != 'Outlier Violation']

# Scatter Plot
scatter_plot = alt.Chart(violation_df).mark_circle(size=60).encode(
    x='avg_fine:Q',
    y='repayment_rate:Q',
    color='violation_description:N',
    tooltip=['violation_description:N', 'avg_fine:Q', 'repayment_rate:Q']
).properties(
    title='Scatter Plot of Average Fine Amount vs. Repayment Rate'
)

scatter_plot
```




```
#3.2
# Bar Plot for Average Fine and Line Plot for Repayment Rate
bar_plot = alt.Chart(violation_df).mark_bar(opacity=0.6, color='blue').encode(
    x=alt.X('violation_description:N', title='Violation Description', sort=None),
    y=alt.Y('avg_fine:Q', title='Average Fine Amount')
).properties(
    title='Average Fine Amount and Repayment Rate by Violation Type'
)

line_plot = alt.Chart(violation_df).mark_line(point=True, color='orange').encode(
    x='violation_description:N',
    y='repayment_rate:Q'
)

combined_plot = bar_plot + line_plot
combined_plot.configure_axisX(labelAngle=45)
```




```
#3.2
#
# Box Plot for Distribution of Average Fine Amounts
box_plot = alt.Chart(violation_df).mark_boxplot().encode(
    x=alt.X('violation_description:N', title='Violation Description', sort=None),
    y=alt.Y('avg_fine:Q', title='Average Fine Amount')
).properties(
    title='Distribution of Average Fine Amount by Violation Type'
)

box_plot
```




```
#3.3
#The scatterplot seems the most meaningful in this context to analyse the relationship between fi
```

```
#4.1
#import pandas as pd

# Sample data loading
# tickets_data = pd.read_csv('path_to_your_data.csv')

# Create DataFrame for violations with at least 100 citations
violation_df = tickets_data.groupby('violation_description').agg(
    total_tickets=('ticket_number', 'count'),
    paid_fine=('fine_level1_amount', lambda x: x[tickets_data['ticket_queue'] == 'Paid'].mean()),
    unpaid_fine=('fine_level1_amount', lambda x: x[tickets_data['ticket_queue'] != 'Paid'].mean()
).reset_index()

# Filter for violations with at least 100 citations
violation_df = violation_df[violation_df['total_tickets'] >= 100]
```

```
# Calculate if the fine doubles and the increase in amount
violation_df['fine_doubles'] = violation_df['unpaid_fine'] == 2 * violation_df['paid_fine']
violation_df['increase_if_unpaid'] = violation_df['unpaid_fine'] - violation_df['paid_fine']

non_doubling_violations = violation_df[~violation_df['fine_doubles']]

# Display results
print(non_doubling_violations[['violation_description', 'paid_fine', 'unpaid_fine', 'increase_if_u
```

```
                      violation_description   paid_fine  unpaid_fine  \
1                           20'OF CROSSWALK   58.397887    57.110092
2                         3-7 AM SNOW ROUTE   55.427136    56.033058
3     ABANDONED VEH. FOR 7 DAYS OR INOPERABLE   71.599265    72.746394
5        BLOCK ACCESS/ALLEY/DRIVEWAY/FIRELANE  140.236686   144.026549
9       BURGLAR ALARM SOUNDING OVER 4 MINUTES   41.233766    42.000000
..                                        ...         ...          ...
109  TRUCK,RV,BUS, OR TAXI RESIDENTIAL STREET   35.877404    42.043121
110     TWO HEAD LAMPS REQUIRED VISIBLE 1000'   25.000000    25.000000
116       WINDOWS MISSING OR CRACKED BEYOND 6   25.000000    25.000000
117                WITHIN 15' OF FIRE HYDRANT  114.613146   119.131752
118         WRONG DIRECTION OR 12'' FROM CURB   25.000000    25.000000

     increase_if_unpaid
1             -1.287796
2              0.605922
3              1.147130
5              3.789862
9              0.766234
..                  ...
109            6.165717
110            0.000000
116            0.000000
117            4.518606
118            0.000000

[66 rows x 4 columns]
```

```
##Explanation for 4.1
##Negative Increases:

##For some violations (e.g., 20' OF CROSSWALK), the increase_if_unpaid is negative, meaning that

##Positive Increases:

##Violations like TRUCK,RV,BUS, OR TAXI RESIDENTIAL STREET show a positive increase of about $6.1
```

```
##Not Doubling: Out of the 66 violations listed, none follow the typical doubling rule for unpaid
```

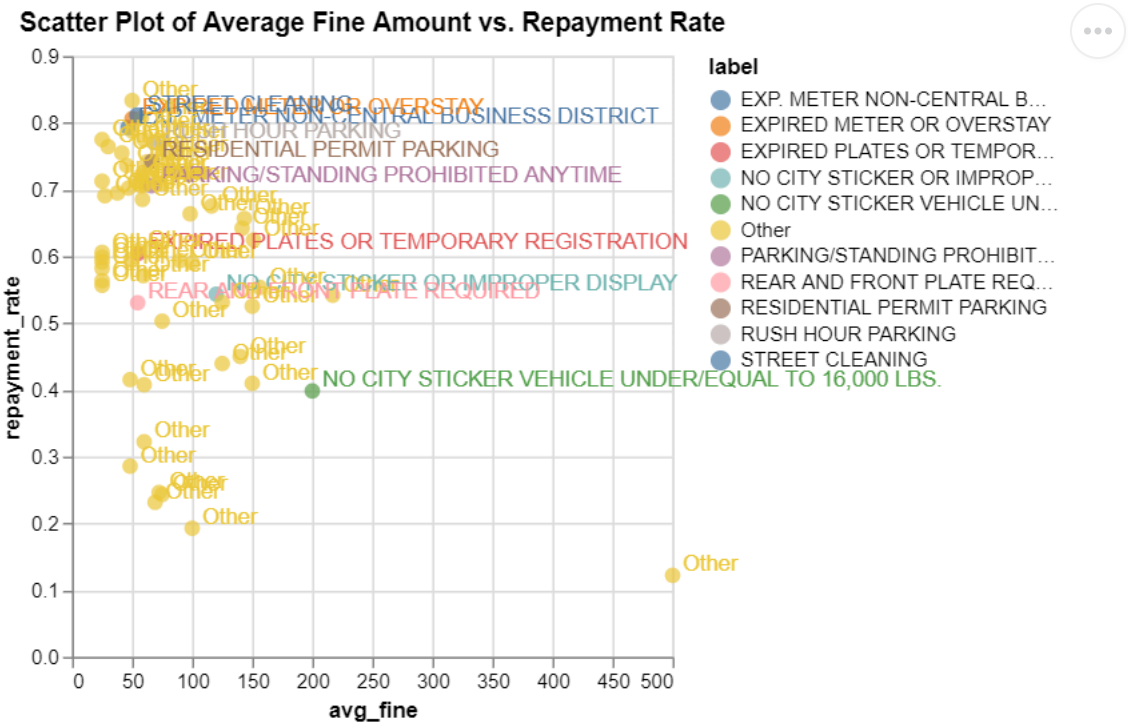
```
#4.2
##Identifying top-ten violations

top_10_violations = violation_df.nlargest(10, 'total_tickets')['violation_description'].tolist()

# Create a new column to label violations
violation_df['label'] = violation_df['violation_description'].apply(
    lambda x: x if x in top_10_violations else 'Other'
)

# Check if the labeling is done correctly
print(violation_df[['violation_description', 'label']].head())
```

```
                    violation_description  label
1                         20'OF CROSSWALK  Other
2                       3-7 AM SNOW ROUTE  Other
3  ABANDONED VEH. FOR 7 DAYS OR INOPERABLE  Other
5     BLOCK ACCESS/ALLEY/DRIVEWAY/FIRELANE  Other
9    BURGLAR ALARM SOUNDING OVER 4 MINUTES  Other
```

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

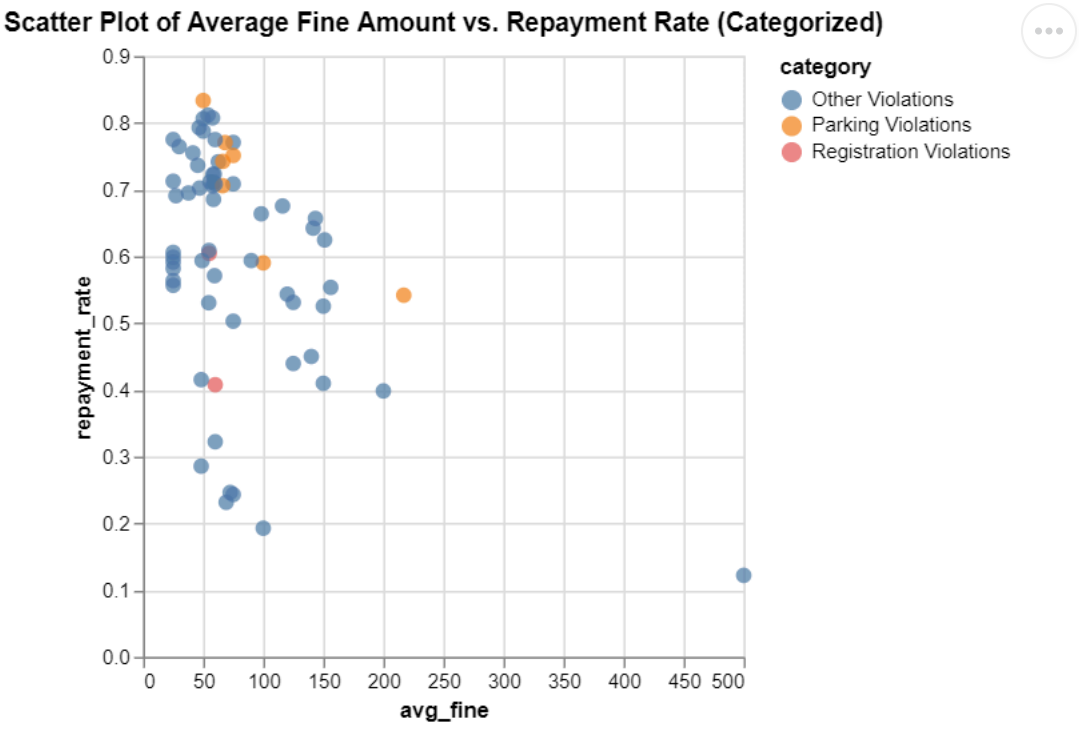
```
adjacent text




_data.groupby('violation_description').agg(
icket_queue', lambda x: (x == 'Paid').sum() / len(x)),
vel1_amount', 'mean'),
cket_number', 'count')


s that appear at least 100 times
on_df[violation_df['total_tickets'] >= 100]

tion descriptions
tion_df.nlargest(10, 'total_tickets')['violation_description'].tolist()

for labeling
```

```
= violation_df['violation_description'].where(violation_df['violation_description'].isin(top_violat

bels
alt.Chart(violation_df).mark_circle(size=60).encode(

Q',

n_description:N', 'avg_fine:Q', 'repayment_rate:Q']

t of Average Fine Amount vs. Repayment Rate'


with_labels.mark_text(




text labels
text_labels
```




```
#4.3(b)

# Define categories for violation descriptions
def categorize_violation(description):
    if 'parking' in description.lower():
        return 'Parking Violations'
```

```
    elif 'speed' in description.lower():
        return 'Speed Violations'
    elif 'license' in description.lower():
        return 'License Violations'
    elif 'registration' in description.lower():
        return 'Registration Violations'
    else:
        return 'Other Violations'

# Apply categorization
violation_df['category'] = violation_df['violation_description'].apply(categorize_violation)

# Scatter Plot with Categories
scatter_with_categories = alt.Chart(violation_df).mark_circle(size=60).encode(
    x='avg_fine:Q',
    y='repayment_rate:Q',
    color='category:N',
    tooltip=['violation_description:N', 'avg_fine:Q', 'repayment_rate:Q']
).properties(
    title='Scatter Plot of Average Fine Amount vs. Repayment Rate (Categorized)'
)

# Add a legend to show categories
scatter_with_categories
```




```
##BONUS QUESTION Q1
import pandas as pd


# Create a DataFrame with necessary columns
```

```
violation_data = tickets_data[['violation_code', 'violation_description']]

# Group by violation_code and count unique violation_descriptions
violation_summary = violation_data.groupby('violation_code').agg(
    num_descriptions=('violation_description', 'nunique'),
    most_common_description=('violation_description', lambda x: x.mode()[0]),
    total_observations=('violation_description', 'count')
).reset_index()

# Filter for codes with multiple descriptions
multiple_descriptions = violation_summary[violation_summary['num_descriptions'] > 1]

# Print the three codes with the most observations
top_three_codes = multiple_descriptions.nlargest(3, 'total_observations')
print(top_three_codes[['violation_code', 'num_descriptions', 'most_common_description', 'total_obs
```

```
   violation_code  num_descriptions                 most_common_description  \
9        0964040B                 2                          STREET CLEANING
90       0976160A                 2            REAR AND FRONT PLATE REQUIRED
91       0976160B                 2  EXPIRED PLATE OR TEMPORARY REGISTRATION

    total_observations
9                32082
90               16853
91                3072
```