problem set II

AUTHOR
Summer Negahdar

PUBLISHED
October 19, 2024

This submission is my work alone and complies with the 30538 integrity policy." Add your initials to indicate your agreement: **SN**

2. "I have uploaded the names of anyone I worked with on the problem set here" **Genevieve Madigan** (2 point)

3. Late coins used this pset: **01** Late coins left after submission: **02**

```
import time
import pandas as pd
import numpy as np
import altair as alt
parking_df= pd.read_csv('data/parking_tickets_one_percent.csv')
#I am just going to take a brief look at
# the heads to see whether I imported the
# right thing or not.
parking_df.head()
```

```
/var/folders/j5/rv933w1173s068kbzq0kp2xh0000gn/T/ipykernel_7815/3115072065.py:5:
DtypeWarning:

Columns (7) have mixed types. Specify dtype option on import or set low_memory=False.
```

	Unnamed: 0	ticket_number	issue_date	violation_location	license_plate_number
0	1	51482901.0	2007-01-01 01:25:00	5762 N AVONDALE	d41ee9a4cb0676e641399ad14aaa20d06f2c6896d
1	2	50681501.0	2007-01-01 01:51:00	2724 W FARRAGUT	3395fd3f71f18f9ea4f0a8e1f13bf0aa15052fc8e560
2	3	51579701.0	2007-01-01 02:22:00	1748 W ESTES	302cb9c55f63ff828d7315c5589d97f1f8144904d6
3	4	51262201.0	2007-01-01 02:35:00	4756 N SHERIDAN	94d018f52c7990cea326d1810a3278e2c6b1e8b44

	Unnamed: 0	ticket_number	issue_date	violation_location	license_plate_number
4	5	51898001.0	2007-01-01 03:50:00	7134 S CAMPBELL	876dd3a95179f4f1d720613f6e32a5a7b86b0e6f98

5 rows × 24 columns

Section One: Data cleaning

1.1

```
def NA_counter(df):
    na_counts = pd.DataFrame({
        'Column Name': df.columns,
        'Number of NAs': df.isna().sum().values
    })

    return na_counts


NA_table = NA_counter(parking_df)
print(NA_table.to_string(index=False))
#I removed the index by to_string command
```

```
          Column Name  Number of NAs
           Unnamed: 0              0
        ticket_number              0
           issue_date              0
   violation_location              0
 license_plate_number              0
  license_plate_state             97
   license_plate_type           2054
              zipcode          54115
       violation_code              0
violation_description              0
                 unit             29
     unit_description              0
         vehicle_make              0
   fine_level1_amount              0
   fine_level2_amount              0
   current_amount_due              0
       total_payments              0
         ticket_queue              0
    ticket_queue_date              0
         notice_level          84068
  hearing_disposition         259899
```

```
notice_number             0
      officer             0
      address             0
```

1.2

Zipcode:

this might be due to the fact that many car plat numbers are out-of state (which was discussed in the essay) or it might be an error of manually entering the zip codes!

Hearing Disposition:

If the ticket was not contested this field is blank. this also makes sense since

Notice Level:

the cells that have no notice level(NAs) mean there was no notice sent! this means that a huge majority of ticket receivers were not even notified(which is in accordance with the argument propublica is making)

1.3

```
##I could not understand what this question is asking and Genevieve helped me here!
# I am going to find all rows whose in "violation desc" cell I can
# find the word "sticker"
city_sticker_violations = parking_df[parking_df['violation_description'
].str.contains('city sticker',
case=False, na=False)]
print(city_sticker_violations[
    'violation_code'].unique())
## so these are the one that involve "sticker" let's see what they each are:
```

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

```
parking_df['issue_date'] = pd.to_datetime(
    parking_df['issue_date'
    ], format='mixed',
    errors='coerce')

city_sticker_violations = parking_df[parking_df['violation_description'
].str.contains(
    'city sticker', case=False, na=False)]

sorted_city_sticker_violations = city_sticker_violations.sort_values(by='issue_date')

unique_city_sticker_codes = sorted_city_sticker_violations[
    ['violation_code', 'violation_description',
 'issue_date']
```

```
 ].drop_duplicates(subset='violation_code')

print(unique_city_sticker_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

                issue_date
14     2007-01-01 10:51:00
2838   2007-02-06 17:29:00
138604 2012-02-25 02:00:00
138699 2012-02-26 08:36:00
138839 2012-02-28 08:24:00
```

the explanation goes like this: for 0964125 and 0976170 there is no weight of vehicle involved(these are the old ones) for 0964125B and 0964125C which are new ones, there is a factor of weight. and 0964125D is for those who have a sticker but they display it improparly. (I am going to eliminate these ones form not having sticker violation!)

1.4

```
#these will be the codes I will be checking that are less than 16k lbs,
# and only include missing or improper (not improper solely)
violation_codes_light = ['0964125', '0976170', '0964125B']

# Filter the DataFrame for the specified violation codes
sticker_fines = parking_df[parking_df[
    'violation_code'].isin(violation_codes_light)]

fine_amounts_lev1 = sticker_fines[['violation_code', 'fine_level1_amount']].drop_duplicat
print(fine_amounts_lev1)
```

```
       violation_code  fine_level1_amount
14            0964125                 120
2838          0976170                 120
138604       0964125B                 200
```

#as we can see here, the fine amounts used to be 120$ in the past but is 200 right now(the article also mentioned these numbers)

Section Two: Revenue increase from “missing city sticker” tickets

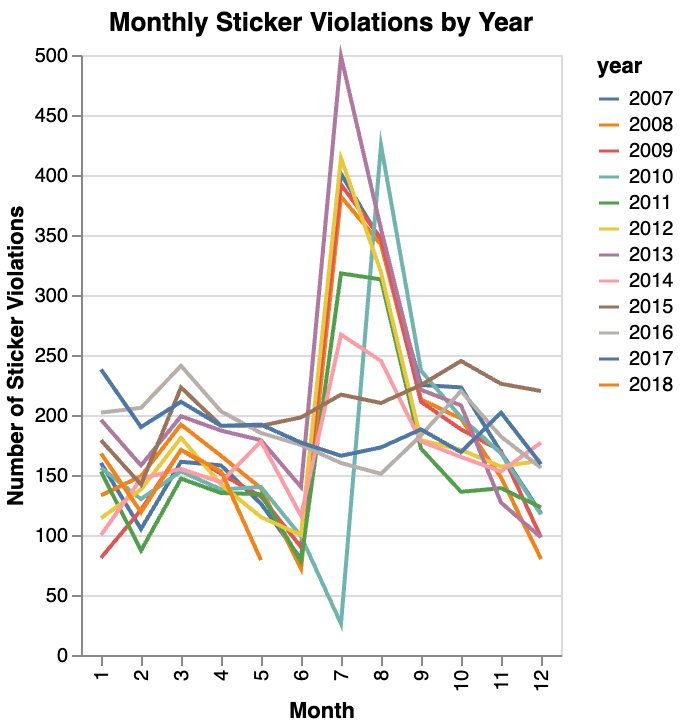
###2.1

```
##so I want to create a dummy variable for sticker
# violation to make life easier for myself and you!
print(violation_codes_light)

# Using apply to create the dummy column
parking_df['sticker_violation_dummy'] = parking_df[
    'violation_code'].apply(
    lambda x: 1 if x in
    violation_codes_light else 0
)

print(parking_df[['violation_code',
'sticker_violation_dummy']].head(21))

##Now I will create three new columns using date column
# one for showing day, one for showing month and one for
#showing year!
parking_df['issue_date'] = pd.to_datetime(parking_df['issue_date'], errors='coerce')

parking_df['month'] = parking_df['issue_date'].dt.month
parking_df['day'] = parking_df['issue_date'].dt.day
parking_df['year'] = parking_df['issue_date'].dt.year
print(parking_df[['issue_date', 'month', 'day', 'year']].head(100))
sticker_parking_df= parking_df[parking_df['sticker_violation_dummy']== 1]
violation_summary = sticker_parking_df.groupby(['year', 'month'])['sticker_violation_dumm

print(violation_summary)
```

```
['0964125', '0976170', '0964125B']
   violation_code  sticker_violation_dummy
0        0964090E                        0
1        0964090E                        0
2        0964150B                        0
3        0976160F                        0
4        0964100A                        0
5         0964060                        0
6        0976160A                        0
7        0964140B                        0
8         0964060                        0
9        0964150B                        0
10       0976160F                        0
11       0976160F                        0
12       0976160F                        0
13       0964050J                        0
14        0964125                        1
15       0964100A                        0
16       0976160A                        0
17       0964200B                        0
```

```
18       0964020B                       0
19       0976160F                       0
20       0964150B                       0
            issue_date  month  day  year
0  2007-01-01 01:25:00      1    1  2007
1  2007-01-01 01:51:00      1    1  2007
2  2007-01-01 02:22:00      1    1  2007
3  2007-01-01 02:35:00      1    1  2007
4  2007-01-01 03:50:00      1    1  2007
..                 ...    ...  ...   ...
95 2007-01-02 11:55:00      1    2  2007
96 2007-01-02 11:57:00      1    2  2007
97 2007-01-02 12:12:00      1    2  2007
98 2007-01-02 12:13:00      1    2  2007
99 2007-01-02 13:10:00      1    2  2007

[100 rows x 4 columns]
     year  month  sticker_violation_dummy
0    2007      1                      160
1    2007      2                      105
2    2007      3                      161
3    2007      4                      158
4    2007      5                      126
..    ...    ...                      ...
132  2018      1                      168
133  2018      2                      119
134  2018      3                      171
135  2018      4                      153
136  2018      5                       79

[137 rows x 3 columns]
```

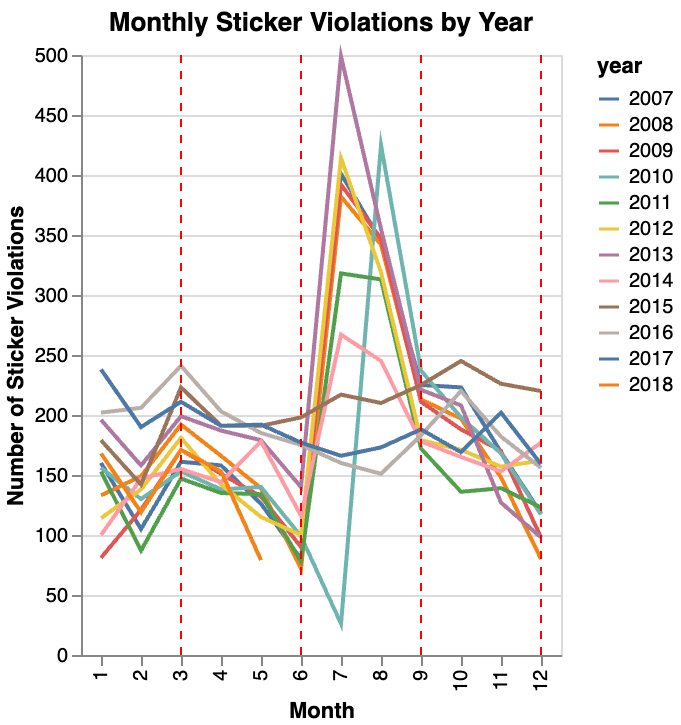
```
##now I am going to plot it I will have 12 months on the
# X axis and then use mark_line to create different years!
import altair as alt

# Create a line chart for sticker violations over time
total_ticket_count_month_year = alt.Chart(violation_summary).mark_line().encode(
    x=alt.X('month:O', axis=alt.Axis(title='Month')),
    y=alt.Y('sticker_violation_dummy:Q', axis=alt.Axis(
        title='Number of Sticker Violations')),
    color='year:N',
    tooltip=['year', 'month', 'sticker_violation_dummy']
).properties(
    title='Monthly Sticker Violations by Year'
)

total_ticket_count_month_year.display()
```



2.2

I want to label each quarter so here is what I do:

```
# I have to create a mark_rule chart with the months 3,6,9 and 12
quarters = [3, 6, 9, 12]  # End of each quarter
quarter_lines = alt.Chart(pd.DataFrame({'month': quarters})
).mark_rule(color='red', strokeDash= [4,4]).encode(
    x='month:O'
)

##now we put the two charts on top of each other:
guided_chart= total_ticket_count_month_year + quarter_lines
guided_chart
```



we can extract from the chart that the highest spike usually happens in the months of March, July to September and occasionally november. it is intersting how july has the largest jump in almost all years! Also, I only asked GPT for how to draw a chart with specified vertical lines and then combined that with my original plot!

2.3

```
#I will filter only the year 2011 and find out the sum of veriables
#with the sticker dummy column equalling 1
sticker_parking_2011 = parking_df[(
    parking_df['year'] == 2011) & (parking_df['sticker_violation_dummy'] == 1)
    ]['sticker_violation_dummy'].sum()
print(sticker_parking_2011)

total_revenue_before= sticker_parking_2011 * 120
full_rev_before= total_revenue_before * 100/ 1000000
# this will give us the total revenue for 100 percent in Million
total_revenue_after= sticker_parking_2011* 200
full_rev_after= total_revenue_after * 100/1000000
#this will give us total revenue after change in Billion
revenue_increase= full_rev_after - full_rev_before
print(revenue_increase)
```

```
1935
15.480000000000004
```

It is actually as the predicted so the increase in revenue would be around 15.5 million or 16 million dollars.

2.4

```
#I first want to see what is the payment fraction for 2011
issued_tickets_pre = parking_df[parking_df['year'] == 2011]
paid_tickets_pre = issued_tickets_pre[issued_tickets_pre['ticket_queue'] == 'Paid']

payment_fraction = len(paid_tickets_pre) / len(issued_tickets_pre)
print(f"The fraction of tickets paid in {2011} is: {payment_fraction:.2f}")

#this means that 71% of the 193500 tickets issued were paid! this
#will bring us at the revenue below:
paid_fraction_rev_2011= 193500 * 0.71*120


#now I want to see how the payment fraction has changed after the
#new policy
issued_tickets_post = parking_df[parking_df['year'] == 2012]
paid_tickets_post = issued_tickets_post[issued_tickets_post['ticket_queue'] == 'Paid']

payment_fraction = len(paid_tickets_post) / len(issued_tickets_post)
print(f"The fraction of tickets paid in {2012} is: {payment_fraction:.2f}")

#well the fraction of payment has not changed that much but
#let's calculate the rvenue:
paid_fraction_rev_2012= 193500 *0.7 *200

total_revenue_paid= (paid_fraction_rev_2012 - paid_fraction_rev_2011) / 1000000
print(total_revenue_paid)
#so the actual revenue (supposing the number of tickets issued is
#the same) will be 10 M$ not 16!
```

```
The fraction of tickets paid in 2011 is: 0.71
The fraction of tickets paid in 2012 is: 0.70
10.6038
```

2.5

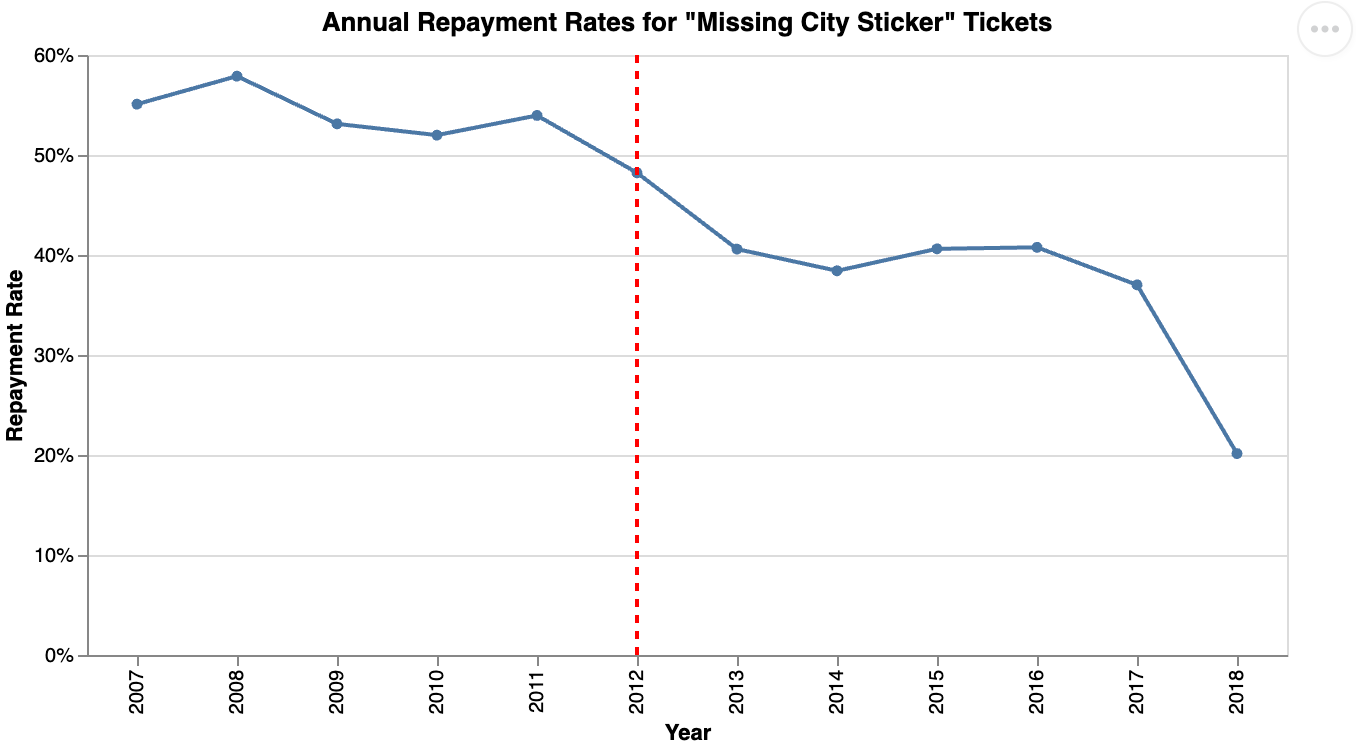
```
#first, I want to create a paid dummy variable as well!
sticker_parking_df['paid_dummy'] = sticker_parking_df['ticket_queue'].apply(
    lambda x: 1 if x == 'Paid' else 0
)
#now I calculate the repayment rate using our two dummies:
filtered_df = sticker_parking_df[sticker_parking_df['sticker_violation_dummy'] == 1]
repayment_rates = filtered_df.groupby('year')['paid_dummy'].mean().reset_index()
repayment_rates.rename(columns={'paid_dummy': 'repayment_rate'}, inplace=True)
print(repayment_rates)
```

```
   year  repayment_rate
0  2007        0.550859
```

```
1   2008        0.578852
2   2009        0.531134
3   2010        0.519879
4   2011        0.539535
5   2012        0.482208
6   2013        0.405921
7   2014        0.384198
8   2015        0.406161
9   2016        0.407686
10  2017        0.370124
11  2018        0.201449
```

```
/var/folders/j5/rv933w1173s068kbzq0kp2xh0000gn/T/ipykernel_7815/3313706562.py:2:
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
```

```
#now I will plot this
payment_rate_trend = alt.Chart(repayment_rates).mark_line(point=True).encode(
    x=alt.X('year:O', axis=alt.Axis(title='Year')),
    y=alt.Y('repayment_rate:Q', axis=alt.Axis(title='Repayment Rate', format='.0%'))
).properties(
    title='Annual Repayment Rates for "Missing City Sticker" Tickets',
    width=600,
    height=300
)
#and now I draw another plot for the cutoff line
policy_cutoff = alt.Chart(pd.DataFrame({'year': [2012]})).mark_rule(color='red', strokeWi
    x='year:O'
)

dif_in_dif= payment_rate_trend + policy_cutoff
dif_in_dif
```



So what we can see here is RDD where the payment rate drops upon the introduction of the new policy. since we have a continuous line, we can say that the decrease in payment rate is uniquely as a resault of the introduction of the new policy. the article is also talking about the same issue where with the increase of fine for not having a sticker the percentage of people who can pay it off drops, leaving lower income families in more debt!

2.6

```
#I am oging to find the three most repeated violation_code
counts = parking_df['violation_code'].value_counts()
top_three = counts.nlargest(3)
print(top_three)

top_three_viol= ['0976160F','0964040B' ,'0964090E']
top3_viol= parking_df[parking_df['violation_code'].isin(top_three_viol)][['violation_code
print(top3_viol['violation_description'].unique())
#these three are the highest committed street crimes:
##residential permit parking
##expired plates or temp registration
##street cleaning!

#now I will group by violation type and calculate the repayment rate
repayment_rates_by_type = filtered_df.groupby('violation_code')['paid_dummy'].mean().rese
repayment_rates_by_type.rename(columns={'paid_dummy': 'repayment_rate'}, inplace=True)
three_highest_payment = repayment_rates_by_type.nlargest(3, 'repayment_rate')
print(three_highest_payment)
three_highest_paid_viol_code= ['0976170', '0964125','0964125B']
```

```
top3_paid_viol= parking_df[parking_df['violation_code'].isin(three_highest_paid_viol_code
print(top3_paid_viol['violation_description'].unique())
```

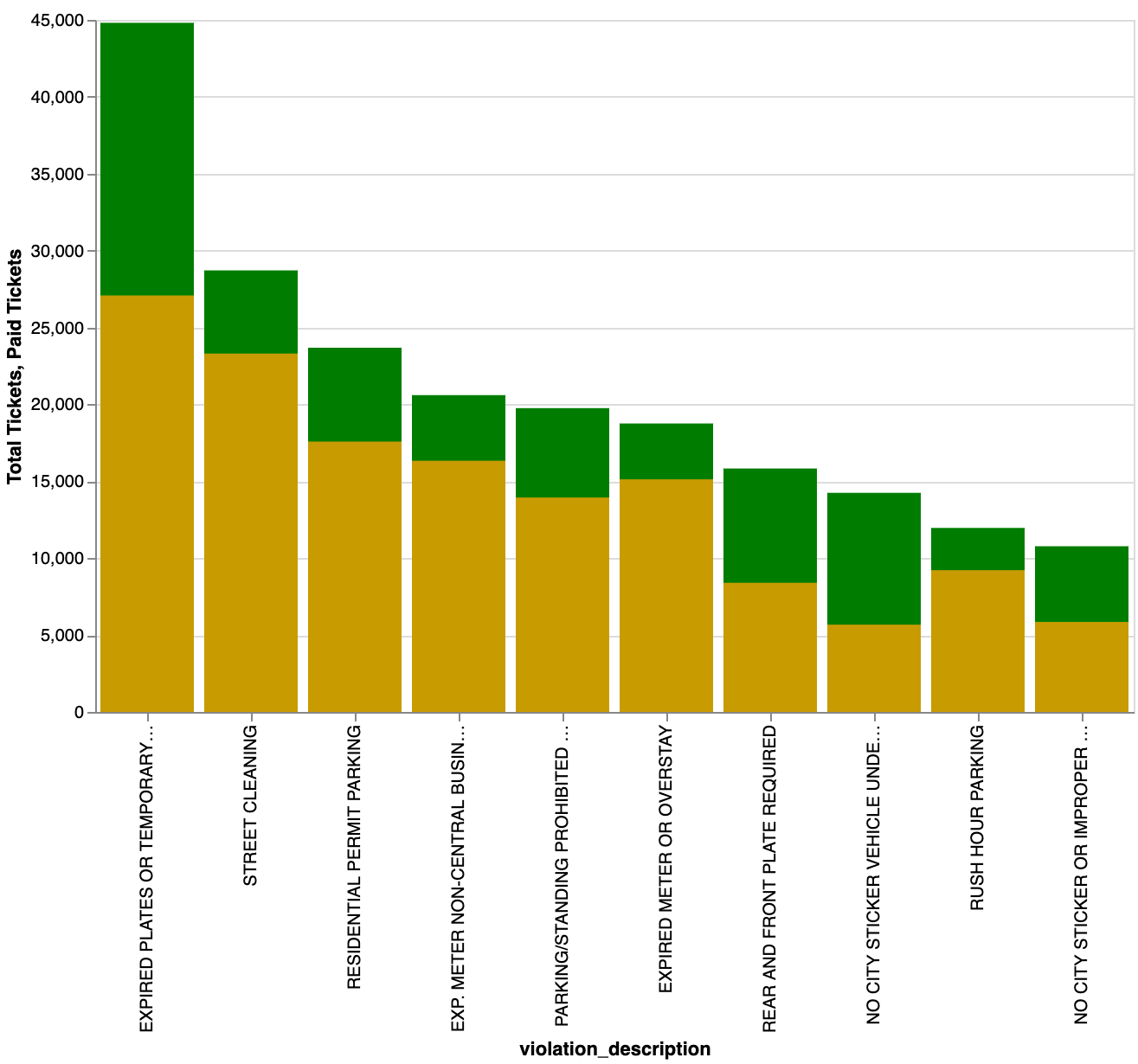
```
violation_code
0976160F    44811
0964040B    32082
0964090E    23683
Name: count, dtype: int64
['RESIDENTIAL PERMIT PARKING' 'EXPIRED PLATES OR TEMPORARY REGISTRATION'
 'STREET CLEANING OR SPECIAL EVENT' 'STREET CLEANING']
  violation_code  repayment_rate
2        0976170        0.666667
0        0964125        0.543131
1       0964125B        0.398357
['NO CITY STICKER OR IMPROPER DISPLAY'
 'NO CITY STICKER VEHICLE UNDER/EQUAL TO 16,000 LBS.']
```

so looking at the three highest paid violations I can see where the decision to raise the sticker ticket comes from as they are all sticker-related.

```
import pandas as pd

# Ensure 'issue_date' is parsed as datetime
parking_df['issue_date'] = pd.to_datetime(parking_df['issue_date'], errors='coerce')

# Group by 'violation_code' and 'ticket_queue', counting occurrences
viol_rating = parking_df.groupby(["violation_description", "ticket_queue"])["Unnamed: 0"]

# Rename the count column for clarity
viol_rating = viol_rating.rename(columns={"Unnamed: 0": "violation_count"})

# Pivot the table to have 'ticket_queue' values as columns
viol_rating = viol_rating.pivot(
    index='violation_description',
    columns='ticket_queue',
    values='violation_count'
).reset_index()

# Fill NaN values with 0
viol_rating = viol_rating.fillna(0)

# Calculating the total number of tickets across all statuses
viol_rating["total_ticket"] = viol_rating[
    ["Bankruptcy", "Court", "Define", "Dismissed", "Hearing Req", "Notice", "Paid"]
].sum(axis=1)
viol_rating["repayment_rate"] = viol_rating["Paid"] / viol_rating["total_ticket"]

viol_rating['repayment_rate'] = viol_rating['repayment_rate'].fillna(0)
print(viol_rating.head(5))
```

```
top_ten_violations = viol_rating.nlargest(10, 'total_ticket')

# Bar plot for total tickets
ticket_plot = alt.Chart(top_ten_violations).mark_bar(
    color='green', opacity=1).encode(
    x=alt.X('violation_description:N', sort=alt.EncodingSortField(field='total_ticket', o
    y=alt.Y('total_ticket:Q', axis=alt.Axis(title='Total Tickets'))
).properties(
    width=600, height=400
)

# Bar plot for paid tickets
paid_plot = alt.Chart(top_ten_violations).mark_bar(
    color='orange', opacity=0.8).encode(
    x=alt.X('violation_description:N', sort=alt.EncodingSortField(field='total_ticket', o
    y=alt.Y('Paid:Q', axis=alt.Axis(title='Paid Tickets'))
)
comparison_chart = ticket_plot + paid_plot

comparison_chart
```

```
ticket_queue                    violation_description  Bankruptcy  Court  \
0             2 REAR TRAILER LAMPS REQ'D VISIBLE 500'         0.0    0.0
1                                  20'OF CROSSWALK         2.0    0.0
2                                3-7 AM SNOW ROUTE         3.0    3.0
3             ABANDONED VEH. FOR 7 DAYS OR INOPERABLE         7.0    4.0
4                   BACK-UP LAMP LIT DURING OPERATION         0.0    0.0

ticket_queue  Define  Dismissed  Hearing Req  Notice   Paid  total_ticket  \
0                1.0        1.0          0.0     1.0    1.0           4.0
1               20.0       66.0          0.0    21.0  284.0         393.0
2               99.0       43.0          0.0    94.0  597.0         839.0
3              358.0      200.0          0.0   263.0  272.0        1104.0
4                0.0        0.0          0.0     0.0    2.0           2.0

ticket_queue  repayment_rate
0                   0.250000
1                   0.722646
2                   0.711561
3                   0.246377
4                   1.000000
```



following a conversation I had with a friend, I realized we have got two different things for this question and thus I tried it myself as well. when looking at this, we can conclude that “expired or temporary plate” has the highest number of tickets while the payment rate is also relatively higher.(I think the city officials did it my way and therefore thought of charging more for sticker biolations!)

Section Three: Headlines and sub-message

3.1

```
#I had created paid dummy for sticker partking,
#now I am going to do it for the whole df
import pandas as pd

# Create the 'paid_dummy' column: 1 if 'PAID', else 0
parking_df['paid_dummy'] = parking_df['ticket_queue'].apply(lambda x: 1 if x == 'Paid' el
```

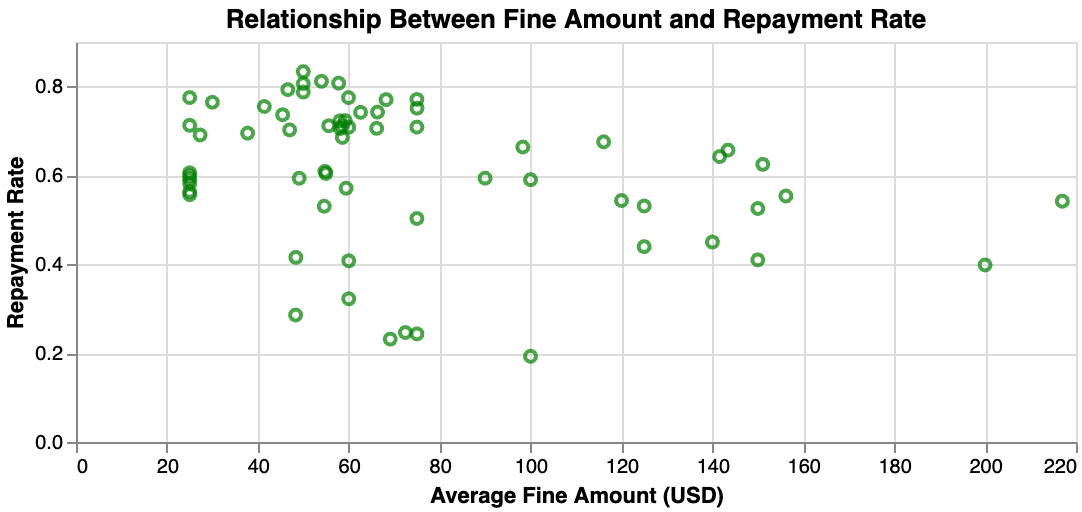
```
# Group by 'violation_description' to calculate metrics
violation_summary = parking_df.groupby('violation_description').agg(
    paid_fraction=('paid_dummy', 'mean'),  # Fraction of paid tickets
    avg_fine=('fine_level1_amount', 'mean'),  # Average fine (level 1)
    total_tickets=('violation_code', 'count')  # Total tickets issued
).reset_index()
# Sort by 'total_tickets' in descending order
violation_summary = violation_summary.sort_values(by='total_tickets', ascending=False)

# Display the top 6 most common violation descriptions
print(violation_summary.head(6))
```

```
                      violation_description  paid_fraction   avg_fine  \
23   EXPIRED PLATES OR TEMPORARY REGISTRATION       0.604361  54.968869
101                          STREET CLEANING       0.811612  54.004249
90                RESIDENTIAL PERMIT PARKING       0.742262  66.338302
19   EXP. METER NON-CENTRAL BUSINESS DISTRICT       0.792913  46.598058
81       PARKING/STANDING PROHIBITED ANYTIME       0.705817  66.142864
21                  EXPIRED METER OR OVERSTAY       0.806355  50.000000

     total_tickets
23           44811
101          28712
90           23683
19           20600
81           19753
21           18756
```

###3.2

```
#first I will subset a df with at least 100 tickets:
import altair as alt

# Subset the DataFrame for violation types with at least 100 tickets
subset_df = violation_summary[violation_summary['total_tickets'] >= 100]
##I have to remove the outlier, I plotted the
# scatterplot and then saw the outlier, I can subset my
# df for all fines less than 400 and be safe, but I
# asked GPT for a general way to do it without
# running a plot once
subset_df = subset_df[subset_df['avg_fine'] < subset_df['avg_fine'].quantile(0.99)]
##it got rid of all fines that are above
# the 0.99 quartile
scatter_plot = alt.Chart(subset_df).mark_point(color='green').encode(
    x=alt.X('avg_fine:Q', title="Average Fine Amount (USD)"),
    y=alt.Y('paid_fraction:Q', title="Repayment Rate"),
    tooltip=['violation_description', 'total_tickets', 'avg_fine', 'paid_fraction']
).properties(
    width=500,
```

```
    height=200,
    title="Relationship Between Fine Amount and Repayment Rate"
)
scatter_plot
```



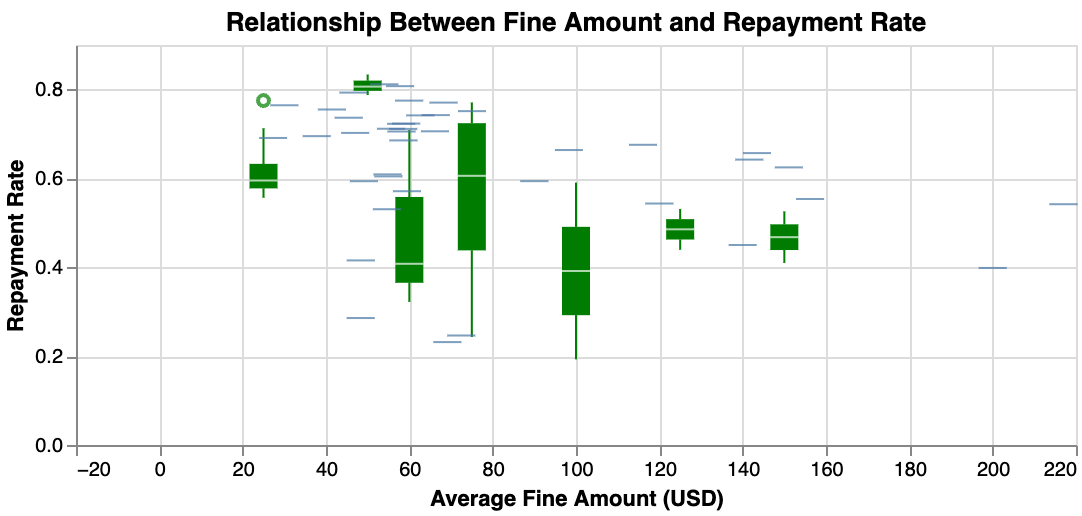
headline:

the highest payment rate is for fines less than 80$

submessage:

the repayment rate has no solid patterns as fine amount increases!

```
##now a box plot!
box_plot = alt.Chart(subset_df).mark_boxplot(color='green').encode(
    x=alt.X('avg_fine:Q', title="Average Fine Amount (USD)"),
    y=alt.Y('paid_fraction:Q', title="Repayment Rate"),
    tooltip=['violation_description', 'total_tickets', 'avg_fine', 'paid_fraction']
).properties(
    width=500,
    height=200,
    title="Relationship Between Fine Amount and Repayment Rate"
)
box_plot
```



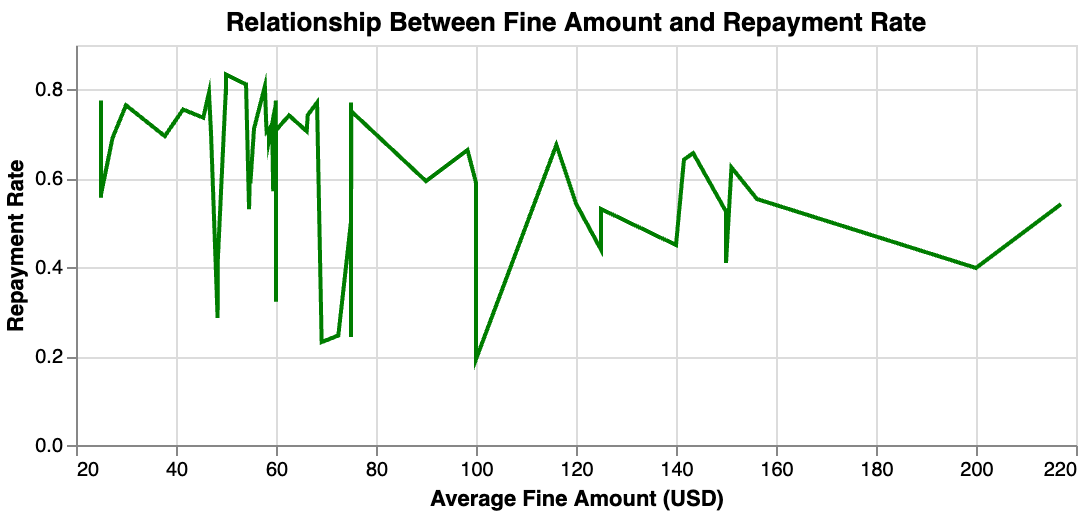
headline:

the fines around 75 USD vary a lot in their payment rate(people react very differently to this amount)

submessage:

by using the box plot we can see how variant the payment rate reacts to fine amount. for examply for higher fine rates(around 200) there isn't much divergence in payment rate and it stays pretty ocnstant (more predictable)

```
##and finally a line plot
line_plot = alt.Chart(subset_df).mark_line(color='green').encode(
    x=alt.X('avg_fine:Q', title="Average Fine Amount (USD)"),
    y=alt.Y('paid_fraction:Q', title="Repayment Rate"),
    tooltip=['violation_description', 'total_tickets', 'avg_fine', 'paid_fraction']
).properties(
    width=500,
    height=200,
    title="Relationship Between Fine Amount and Repayment Rate"
)
line_plot
```



headline:

people react to 2-digit fines(under 100) jump to three-digit(100 and above) drastically!

submessage:

payment rate is pretty noisy around under 100 fines (people's payment pattern is not predictable)

3.3

since they need to see how people react to change in fine amount, I would take the line plot as it shows the drastic changes. justl ike I mentioned one the first digit of the fine changes (jump from 99 to 101 and 199 to 201) people suddenly react. (from a consumer analysis standpoint the psychological reasoning behind pricing something with a 0.99 is also the same. people suddenly react negativly to raise in price if the first digit changes!)

Section Four: Understanding the structure of the data and summarizing it

4.1.

```
print(parking_df.dtypes)
#I wanted to make sure fine amount cols are
# numeric which they are!

#Now I am going to create a new column to see the ratio between first and second round of
#than 100 citations
# Create the 'total_ticket' column with ticket counts for each violation code
parking_df['total_ticket'] = parking_df['violation_description'].map(parking_df['violatio
print(parking_df.head())
subset_df_Q3= parking_df[parking_df['total_ticket'] >= 100]

subset_df_Q3['fine_ratio']= subset_df_Q3['fine_level2_amount'] / subset_df_Q3['fine_level
#Now I will see if there are any rows where
# fine ratio !=2
non_double= subset_df_Q3[subset_df_Q3['fine_ratio'] !=2]
non_double_sorted = non_double.sort_values(
    by='fine_level1_amount', ascending=False
)
print(non_double_sorted[['violation_description', 'fine_level1_amount',
                         'fine_level2_amount', 'fine_ratio']].head(5))
```

```
Unnamed: 0                      int64
ticket_number                 float64
issue_date                datetime64[ns]
violation_location                object
```

```
license_plate_number              object
license_plate_state               object
license_plate_type                object
zipcode                           object
violation_code                    object
violation_description             object
unit                             float64
unit_description                  object
vehicle_make                      object
fine_level1_amount                 int64
fine_level2_amount                 int64
current_amount_due               float64
total_payments                   float64
ticket_queue                      object
ticket_queue_date                 object
notice_level                      object
hearing_disposition               object
notice_number                    float64
officer                           object
address                           object
sticker_violation_dummy            int64
month                              int32
day                                int32
year                               int32
paid_dummy                         int64
dtype: object
```

	Unnamed: 0	ticket_number	issue_date	violation_location	\
0	1	51482901.0	2007-01-01 01:25:00	5762 N AVONDALE	
1	2	50681501.0	2007-01-01 01:51:00	2724 W FARRAGUT	
2	3	51579701.0	2007-01-01 02:22:00	1748 W ESTES	
3	4	51262201.0	2007-01-01 02:35:00	4756 N SHERIDAN	
4	5	51898001.0	2007-01-01 03:50:00	7134 S CAMPBELL	

	license_plate_number	license_plate_state	\
0	d41ee9a4cb0676e641399ad14aaa20d06f2c6896de6366...	IL	
1	3395fd3f71f18f9ea4f0a8e1f13bf0aa15052fc8e5605a...	IL	
2	302cb9c55f63ff828d7315c5589d97f1f8144904d66eb3...	IL	
3	94d018f52c7990cea326d1810a3278e2c6b1e8b44f3c52...	IL	
4	876dd3a95179f4f1d720613f6e32a5a7b86b0e6f988bf4...	IL	

	license_plate_type	zipcode	violation_code	\
0	PAS	606184118.0	0964090E	
1	PAS	606454911.0	0964090E	
2	PAS	604116803.0	0964150B	
3	PAS	606601345.0	0976160F	
4	PAS	606291432.0	0964100A	

	violation_description	...	hearing_disposition	\
0	RESIDENTIAL PERMIT PARKING	...	Liable	
1	RESIDENTIAL PERMIT PARKING	...	NaN	
2	PARKING/STANDING PROHIBITED ANYTIME	...	NaN	

```
3  EXPIRED PLATES OR TEMPORARY REGISTRATION  ...                 NaN
4               WITHIN 15' OF FIRE HYDRANT  ...                 NaN

  notice_number officer                      address  \
0  5.080059e+09   17266  5700 n avondale, chicago, il
1  5.079876e+09   10799  2700 w farragut, chicago, il
2  5.037862e+09   17253     1700 w estes, chicago, il
3  5.075310e+09    3307  4700 n sheridan, chicago, il
4  5.073568e+09   16820  7100 s campbell, chicago, il

   sticker_violation_dummy  month  day  year paid_dummy total_ticket
0                        0      1    1  2007          1        23683
1                        0      1    1  2007          1        23683
2                        0      1    1  2007          0        19753
3                        0      1    1  2007          1        44811
4                        0      1    1  2007          1         6104

[5 rows x 30 columns]
                          violation_description  fine_level1_amount  \
160124  NO CITY STICKER VEHICLE OVER 16,000 LBS.                 500
145779  NO CITY STICKER VEHICLE OVER 16,000 LBS.                 500
138699  NO CITY STICKER VEHICLE OVER 16,000 LBS.                 500
139079  NO CITY STICKER VEHICLE OVER 16,000 LBS.                 500
139163  NO CITY STICKER VEHICLE OVER 16,000 LBS.                 500

        fine_level2_amount  fine_ratio
160124                 775        1.55
145779                 775        1.55
138699                 775        1.55
139079                 775        1.55
139163                 775        1.55
/var/folders/j5/rv933w1173s068kbzq0kp2xh0000gn/T/ipykernel_7815/2939784390.py:12:
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
```

```
non_double_sorted['fine_increase']= non_double_sorted['fine_level2_amount'] - non_double_

subset_df = parking_df.groupby('violation_code').filter(lambda x: len(x) >= 100)
subset_df['fine_ratio'] = subset_df['fine_level2_amount'] / subset_df['fine_level1_amount

non_double_fines = subset_df[subset_df['fine_ratio'] != 2]

non_double_fines['fine_increase'] = non_double_fines['fine_level2_amount'] - non_double_f
fine_increase_summary = non_double_sorted.groupby('violation_description').agg(
```

```
    avg_fine_level1=('fine_level1_amount', 'mean'),
    avg_fine_level2=('fine_level2_amount', 'mean'),
    avg_fine_increase=('fine_increase', 'mean')
)
fine_increase_summary
```

```
/var/folders/j5/rv933w1173s068kbzq0kp2xh0000gn/T/ipykernel_7815/3994779139.py:8:
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
```

	avg_fine_level1	avg_fine_level2	avg_fine_increase
violation_description			
BLOCK ACCESS/ALLEY/DRIVEWAY/FIRELANE	150.0	250.0	100.0
DISABLED PARKING ZONE	200.0	250.0	50.0
NO CITY STICKER VEHICLE OVER 16,000 LBS.	500.0	775.0	275.0
OBSTRUCTED OR IMPROPERLY TINTED WINDOWS	250.0	250.0	0.0
PARK OR BLOCK ALLEY	150.0	250.0	100.0
PARK/STAND ON BICYCLE PATH	150.0	250.0	100.0
SMOKED/TINTED WINDOWS PARKED/STANDING	250.0	250.0	0.0

4.2

```
from PIL import Image
from PIL import Image
import matplotlib.pyplot as plt

# Load the images
image1 = Image.open("1.png")
image2 = Image.open("2.png")

# Create a figure and display the images side-by-side
fig, axes = plt.subplots(1, 2, figsize=(10, 5))

# Plot the first image
axes[0].imshow(image1)
axes[0].axis('off')  # Hide the axis
axes[0].set_title("Image 1")
```

```
# Plot the second image
axes[1].imshow(image2)
axes[1].axis('off')  # Hide the axis
axes[1].set_title("Image 2")

# Display the plot
plt.show()
```

Image 1

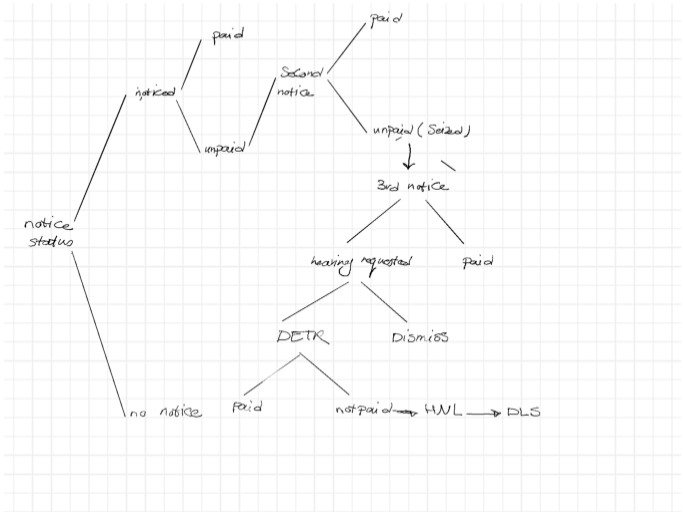
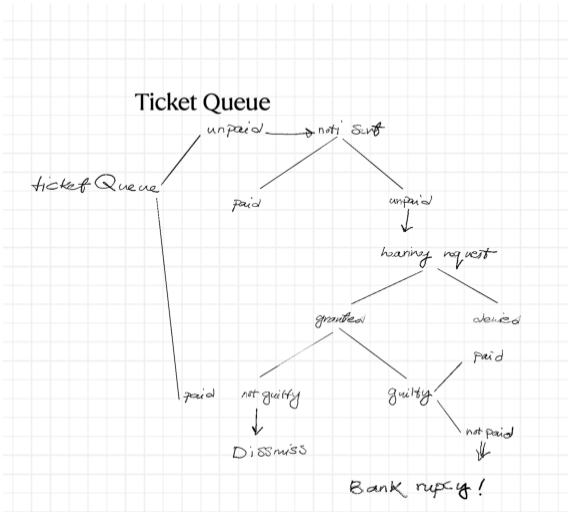


Image 2



4.3

```
aggregated_df_4 = subset_df_Q3.groupby('violation_description').agg(
    paid_fraction=('paid_dummy', 'mean'),  # Fraction of paid tickets
    avg_fine=('fine_level1_amount', 'mean'),  # Average fine (level 1)
    total_ticket=('violation_code', 'count')  # Total tickets issued
).reset_index()

# Create the 'non_common_viol' column for grouping less common violations
aggregated_df_4['non_common_viol'] = np.where(
    aggregated_df_4['total_ticket'] < aggregated_df_4['total_ticket'].nlargest(11).min(),
    'Other',
    aggregated_df_4['violation_description']
)

# Disable Altair's row limit restriction
alt.data_transformers.disable_max_rows()

# Create the scatter plot
scatter_plot_Q4 = alt.Chart(aggregated_df_4).mark_point().encode(
    x=alt.X('avg_fine:Q', title="Average Fine Amount (USD)"),
    y=alt.Y('paid_fraction:Q', title="Repayment Rate"),
    color=alt.Color(
```

```
        'non_common_viol:N',
        legend=alt.Legend(title="Violation Description")
    )
).properties(
    width=500,
    height=200,
    title="Relationship Between Fine Amount and Repayment Rate"
)

scatter_plot_Q4
```