Andrew_White_Pset_2

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

"I have uploaded the names of anyone I worked with on the problem set here(https://docs.google.com/forms/d/18
A.W.

```
import pandas as pd
import altair as alt
import os
import json
alt.renderers.enable("png")
```

```
RendererRegistry.enable('png')
```

```
raw_debt = r"C:/Users/andre/Documents/GitHub/ppha30538_fall2024/problem_sets/ps1/data"
path_debt = os.path.join(raw_debt, "parking_tickets_one_percent.csv")
tickets = pd.read_csv(path_debt, parse_dates = ["issue_date"]) #Will be useful for Part 2
```

```
C:\Users\andre\AppData\Local\Temp\ipykernel_31000\3646373265.py:3: DtypeWarning: Columns (7)
  tickets = pd.read_csv(path_debt, parse_dates = ["issue_date"]) #Will be useful for Part 2
```

Part 1

Question 1 Finding all NA values

```
#This portion is designed to be run on each column
def nan_search(X):
    NA_box = [] #box to hold the number of NA values
    for i in X:
        if pd.isnull(i):
            NA_box.append(1) #will have a
            # value of 1 for each instance of NaN
    return(sum(NA_box))
    #adding up each instance to return
    # the full number of instances
```

Attribution: Troubleshooting. Asked ChatGTP why my code was returning a sum of 0, and it advised that, instead of setting my condition to pd.isnull(i) == "True", it should just be pd.isnull(i)

Creating a function to iterate through columns

```
#I'm embedding na_search() inside
def nan_iterator(Y):
    sums_holder = [] #meant to hold the different
    #sums for each column
    for column in Y:
        sums_holder.append(nan_search(Y[column]))
        #running #na_search on each column of the
        # full dataset
        #Because dataframe iteration
        # defaults to iterating through
        #  columns, python knows to
        # run na_search on each column.
    return(sums_holder)
```

Making the final table to hold everything. Also note that the previous functions are embedded within this one.

```
def nan_table_maker(Z):
    my_dictionary = {
        "Variables": Z.columns, "NaN_Count": nan_iterator(Z)
    } #first make a dictionary, then
    #turn it into a DataFrame. Using this as
    # #inspiration:
    # https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html
    values = pd.DataFrame(data = my_dictionary)
    #make a series with the column names
    # as the index and the sum of NA
    # values generated for each column
    #  by na_iterator as the contents
    return values

#little subset of tickets for quick tests
lil_tickets = tickets.iloc[0:500]
nan_table_maker(lil_tickets)

#full data:
my_NA_table = nan_table_maker(tickets)
```

Answer Q 1: It appears that only 6 variables have NaN values, with zipcode, notice_level, and hearing_disposition having, by far, the greatest number. Specifically, under zipcode there are 54,115, for notice level 84,068, and completely blowing them away is hearing disposition at 259,899.

Question 2

Answer Q 2: I believe that the reason that zipcode, notice_level, and hearing_disposition have the greatest number of NA values is that they each use empty entries to convey information, according to the data dictionary here: https://github.com/propublica/il-tickets-notebooks, The variable "hearing_disposition" will be empty when a ticket was not contested. So each NaN value is an uncontested ticket. "Notice_level" is similar, blank entries mean that no notice was sent. "Zipcode" is less well-defined in the data dictionary, but I would reason that a similar phenomenon is at work here, the dictionary says that "zipcode" is the zip code associated "with the vehicle registration," so perhaps when a vehicle is unregistered, it will include an empty value?

Question 3

```
#find all descriptions associated with the word sticker

#Using: https://www.geeksforgeeks.org/select-rows-that-contain-specific-text-using-pandas/
sticker_things = tickets[tickets["violation_description"].str.contains("NO CITY STICKER")] #
#entries that include the words "NO CITY STICKER".


old_sticker_things = sticker_things[sticker_things["fine_level1_amount"] <= 120]
 #subset for only tickets that meet
 #  the old sticker ticket cost
 # to find old sticker ticket entries.
#I do not believe the article
# (included in the Answer below)
# noted the timeline of the changes
#  exactly, this way we can
#  see old and new tickets.

#view the relevant variables
old_sticker_things.loc[:, ["fine_level1_amount", "fine_level2_amount", "violation_descriptio

# find all fine amounts that are
# less than or equal to $120
old_sticker_things["fine_level1_amount"].unique()
```

```
array([120])
```

```
#subset for only tickets that
#  cost more than the old sticker
#  ticket cost to find new
# sticker ticket entries.
new_sticker_things = sticker_things[sticker_things["fine_level1_amount"] > 120]

#view the relevant variables
new_sticker_things.loc[:, ["fine_level1_amount", "fine_level2_amount","violation_description'

#find all fine amounts that are greater than $120
new_sticker_things["fine_level1_amount"].unique()
```

```
array([200, 500])
```

```
#Making a dataset where I remove rows for cars
#over 16000 LBS.

#make a binary variable, 1
# for little car, 0 for big car
def little_car_finder(X):
    B = []
    for entry in X:
        if "NO CITY STICKER VEHICLE OVER 16,000 LBS" in entry :
            B.append(0)
        else:
            B.append(1)
    return(B)
#Attribution: I asked Chat GPT why my code
# contained no zeros, and was advised to
#change the syntax of my if statement

#subset into a series containing
# only violation_description
vd_tickets = tickets["violation_description"]

#plug this subset into
# little_car_finder() in order to creat
#the binary variable
little_cars = little_car_finder(vd_tickets)

#add the binary variable to the main dataset
tickets["is_little_car"] = little_cars
```

Data exploration using this binary variable

```
big_cars = tickets[tickets["is_little_car"] == 0]
# subset only for cars over 16,000 LBS
big_cars["fine_level1_amount"].unique()
#check fine amounts

tiny_cars = tickets[tickets["is_little_car"] == 1]
# subset only for cars under 16,000 LBS
tiny_cars["fine_level1_amount"].unique()
#check fine amounts

#Some more exploration of sticker violations
any_sticker = tickets[tickets["violation_description"].str.contains("STICKER")]

all_sticker_violation_codes = list(any_sticker["violation_code"].unique())

print(all_sticker_violation_codes)
```

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

Code for finding violation descriptions based on codes

```python
#find all unique violation
# descriptions within that subset
#this gives us the violation
#  description associated
# with a given code
def subsetter(df, the_code): #df = dataframe,
    #the_code = a violation code
    subset = df[df["violation_code"] == the_code]
    #subset by violation code
    return(subset["violation_description"].unique())


subsetter(df = tickets, the_code = "0964125")

subsetter(df = tickets, the_code = "0964125B")

subsetter(df = tickets, the_code = "0964125C")

subsetter(df = tickets, the_code = "0964125D")

subsetter(df = tickets, the_code = "0976170")
```

```
array(['NO CITY STICKER OR IMPROPER DISPLAY'], dtype=object)
```

Similar concept to the function above, but selecting initial fine amounts

```python
def subset_fine_one(df, the_code): #df = dataframe,
    #the_code = a violation code
    subset = df[df["violation_code"] == the_code]
    #subset by violation code
    return(subset["fine_level1_amount"].unique())

subset_fine_one(df = tickets, the_code = "0964125")

subset_fine_one(df = tickets, the_code = "0964125B")

subset_fine_one(df = tickets, the_code = "0964125C")

subset_fine_one(df = tickets, the_code = "0964125D")

subset_fine_one(df = tickets, the_code = "0976170")
```

```
array([120])
```

Answer Q3: This article: https://www.propublica.org/article/chicago-vehicle-sticker-law-ticket-price-hike-black-drivers-debt explains that the cost of not having a city sticker increased from $120 to $200. Using the data exploration above, the new violation code is 0964125B, while the old one was 0964125. I have come to these conclusions in a roundabout way:

I first subset the data to include only violation_description entries that contained the word "STICKER", I've since updated that code, however, because this captured 0964125D, or tickets for an improperly displayed sticker. Reasoning that improper ticket display implies the possession of a sticker, and therefore should not be included in missing sticker analysis, I decided to remove these from the subset by selecting for only "NO CITY STICKER".

Then I used the original and new fine costs ($120 and $200) to further disaggregate the subsetted data, finding that, in my subsetted data, the only initial fine amounts were $120, $200, and $500. Based on what the article, and our directions for this assignment had said, and my own data exploration, I hypothesized that the $500 was the fine for cars missing a sticker and weighing over 16,000 LBS.

By this point, I knew that there were five sticker-related ticket codes: '0964125' and '0976170', which are both "NO CITY STICKER OR IMPROPOER DISPLAY"; '0964125B' and '0964125C' which both only include tickets for not having a city sticker, but disaggregate by weight, with the former capturing vehicles under 16,000 LBS and the latter capturing vehicles over that weight; and finally '0964125D', which only captures cars that have improperly displayed their sticker. Based on the instructions, I am ignoring '0964125D' and '0964125C'

I also found the initial fine amounts associated with each code. '0964125' and '0976170' have initial fines of $120. '0964125B' has an initial fine of $200. Taking all of this information together, I believe it is reasonable to assert that the old fine cost is indeed $120, and the new fine cost is $200. It is also notable that because the old fines ('0964125' and '0976170') don't disaggregate by weight or sticker placement (they both lump missing sticker violations with improperly placed ones), the numbers are not strictly comparable.

Part 2

Question 1 Collapsing the different sticker codes into one code called "missing_sticker_violations"
Step 1: Create a binary variable equal to 1 when violation code is equal to 0964125B or 0964125

```python
def no_sticker_sort(X):
    no_stickers = []
    for entry in X:
        if "0964125C" in entry or "0964125D" in entry: #I am not counting improper display o
```

```
            #or the lack of a city sticker on a car over 16,000 LBS (0964125C), based on the
            no_stickers.append(0)
        elif "0964125B" in entry or "0964125" in entry or "0976170" in entry:
            no_stickers.append(1)
        else:
            no_stickers.append(0)
    return(no_stickers)
# Accreditation: I asked ChatGPT why
# my function was returning only zeros, and it
#  explained that I needed to edit the syntax of my conditions
```

Step 2: save the output as a variable

```
sticker_violation = no_sticker_sort(tickets["violation_code"])

#Step 3: save the variable to the main dataset
tickets["missing_sticker_violations"] = sticker_violation
```

Collapsing the data around month

```
# Using:
#https://pandas.pydata.org/pandas-docs/version/1.3.1/getting_started/intro_tutorials/09_times

# and reviewing lecture (viz_5_explore...)

#Use Timestamp object
# ("issue_date") to groupby month
# and display ticket
# sums for relevant missing
# sticker violations

#reset_index() so that issue_date
#  becomes a column, rather
#than the index
grouped_stickers = tickets["missing_sticker_violations"].groupby(tickets["issue_date"].dt.mon


print(grouped_stickers)
```

```
    issue_date  missing_sticker_violations
0            1                        1880
```

```
1              2                         1692
2              3                         2205
3              4                         1959
4              5                         1790
5              6                         1325
6              7                         3242
7              8                         3225
8              9                         2232
9             10                         2120
10            11                         1841
11            12                         1508
```

The data in grouped_stickers appear to show that missing sticker violations occur most frequently in July and August, and least frequently in June and December. Note that this table does not disaggregate by year, only by month, so the counts lump all years together, which could conceal some complexity.

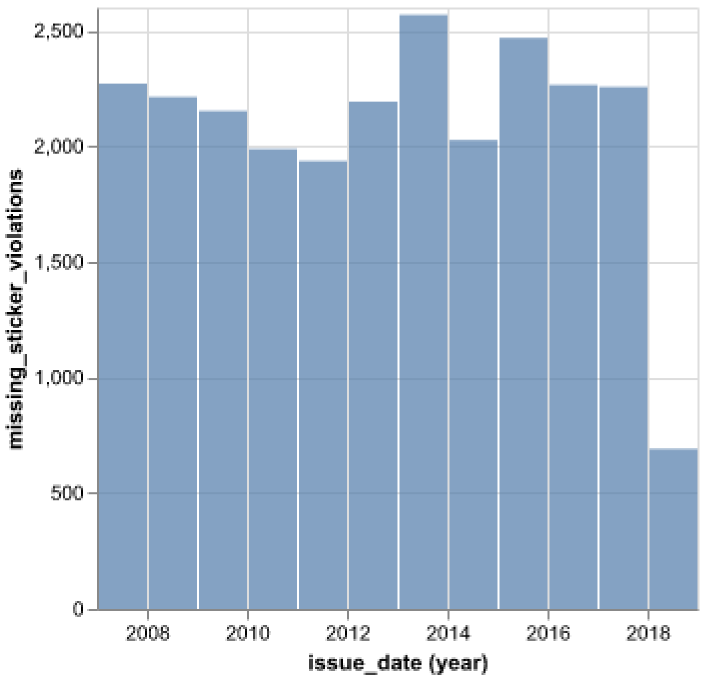
Use altair to graph missing sticker violations over time Answer Q 1 Focusing on year

```python
# Using lecture (viz_5_explore...):

#raise the row limit:
alt.data_transformers.enable("vegafusion")

year_graph_1 = alt.Chart(tickets).mark_bar().encode(
    x = alt.X("year(issue_date):T"),
    y  = alt.Y("missing_sticker_violations:Q"),
)

year_graph_1.show()
```

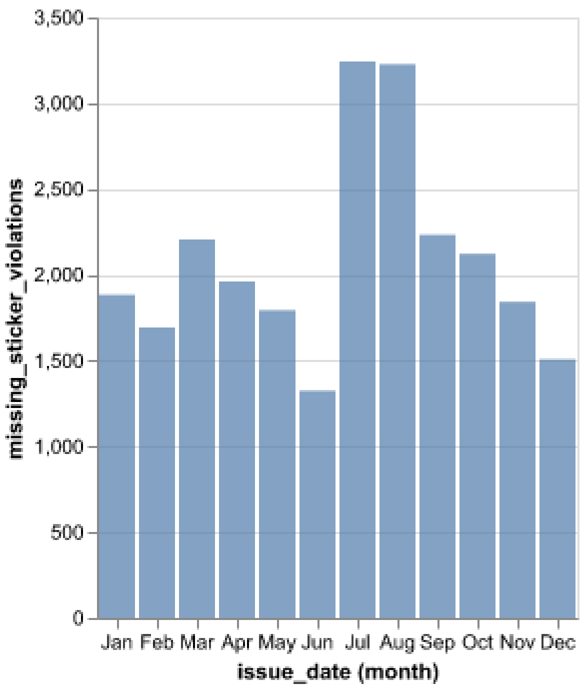


Answer Q 1 continued Focusing on Month

```
month_graph_1 = alt.Chart(tickets).mark_bar().encode(
    x = alt.X("month(issue_date):N"),
    y = alt.Y("missing_sticker_violations:Q")
)

month_graph_1.save("month_graph_1.png")

month_graph_1.show()
```



Question 2

Identify the date upon which the new missing sticker code was launched

```
tickets["issue_date"][tickets["violation_code"].str.contains("0964125B")]
```

```
138604   2012-02-25 02:00:00
138614   2012-02-25 06:25:00
138624   2012-02-25 09:47:00
138625   2012-02-25 09:50:00
138631   2012-02-25 11:00:00
                 ...
287420   2018-05-14 08:53:00
287440   2018-05-14 11:50:00
287441   2018-05-14 12:09:00
287450   2018-05-14 14:11:00
287452   2018-05-14 14:30:00
Name: issue_date, Length: 14246, dtype: datetime64[ns]
```

It appears that the first time the new violation code for missing stickers appears is February 25th, 2012.

Modifying year_graph

```
year_graph_1 = alt.Chart(tickets).mark_bar().encode(
    x = alt.X("year(issue_date):T"),
    y  = alt.Y("missing_sticker_violations:Q"),
)
```

Question 3

Identify year before the new sticker code was launched and calculate projected revenue

```
#2011 is the year before the new missing sticker code
grouped_stickers_year = tickets["missing_sticker_violations"].groupby(tickets["issue_date"].

grouped_stickers_year[grouped_stickers_year["issue_date"] == 2011]
```

	issue_date	missing_sticker_violations
4	2011	1935

1,935 missing sticker tickets were issued in 2011.

Some math

```
old_revenue = 1935 * 120

print(old_revenue)

new_revenue = 1935 * 200

print(new_revenue)

(new_revenue - old_revenue) * 100
```

```
232200
387000

15480000
```

Answer Q 3: With the assumptions specified in the instructions, the City Clerk should have projected an increase in revenue of $15,480,000, fairly close to $16,000,000.

Question 4

Create a graph showing ticket payment rate by year

```
#Make a dummy variable for having paid off a ticket
def paid_it_all(X):
    paid_tickets = []
    for i in X["ticket_queue"]:
        if i == "Paid":
            paid_tickets.append(1)
        else:
             paid_tickets.append(0)
    return(paid_tickets)

#run on df and save results to variable
paid_tickets = paid_it_all(tickets)
#add the result back to the df
tickets["paid_off_tickets"] = paid_tickets
```

Use my code from pset1 to make a dummy equal to 1 when a ticket exists

```
def one_dummy(C):
  ticket_entry = []
  i = 0
  while 1 > 0 and i < C:
    # C is the number of '1's to generate
    # should be equal to len(df)
    ticket_entry.append(1)
    i += 1
  return(ticket_entry)

#use the len(df) to ensure that each entry has a 1
ticket_exists = one_dummy(len(tickets))

# save to df
tickets["ticket_here"] = ticket_exists
```

Use groupby() to create a dataframe capturing all missing_sticker tickets and all paid off tickets and then find payment rate by year

```
grouped_stickers_ultra = tickets[["ticket_here", "paid_off_tickets"]][tickets["missing_stick

#add the payment rate variable
grouped_stickers_ultra["payment_rate"] = grouped_stickers_ultra["paid_off_tickets"] / groupe

print(grouped_stickers_ultra)
```

```
    issue_date  ticket_here  paid_off_tickets  payment_rate
0         2007         2271              1251      0.550859
1         2008         2213              1281      0.578852
2         2009         2152              1143      0.531134
3         2010         1987              1033      0.519879
4         2011         1935              1044      0.539535
5         2012         2192              1057      0.482208
6         2013         2567              1042      0.405921
7         2014         2025               778      0.384198
8         2015         2467              1002      0.406161
9         2016         2264               923      0.407686
10        2017         2256               835      0.370124
11        2018          690               139      0.201449
```

In 2013, the year after 2012, the year the fee change was implemented, the payment rate was about 41%.

Answer Q 4 The grouped_stickers_ultra dataframe above displays ticket payment rates over time. Note that I am defining a paid ticket as a ticket that has a "ticket_queue" value of "Paid," which may filter out some amount of tickets that could conceivably have been partially paid, such as tickets that were contested because, perhaps, a ticket was partially paid on a payment plan and subsequently contested. Based off of the payment rates in 2011 (about 54%) and in 2013 (about 41%), the year after the year in which the sticker ticket cost increased (2012), the revenue change would actually have been:

```
new_old_revenue = (1935 * 120) * 0.539535

print(new_old_revenue)

new_new_revenue = (1935 * 200) * 0.405921

print(new_new_revenue)

print((new_new_revenue - new_old_revenue) * 100)
```

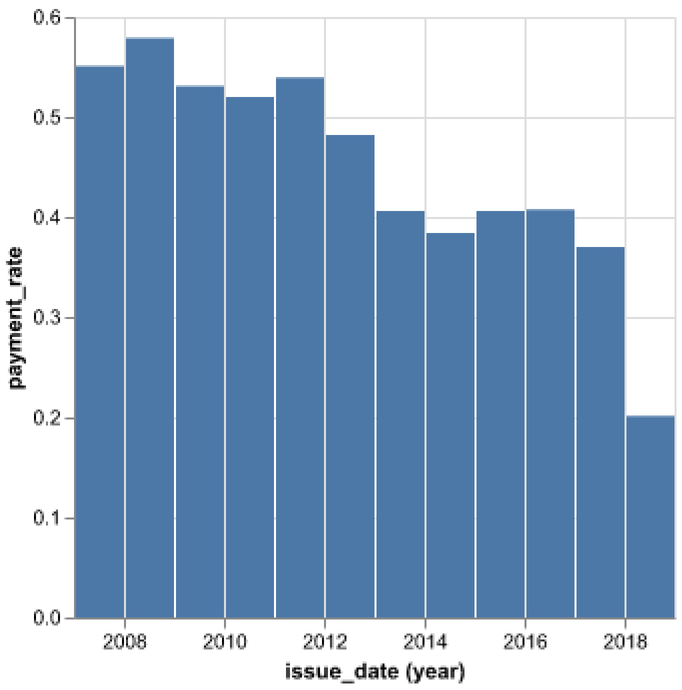
```
125280.027
157091.427
3181139.9999999995
```

Answer Q 4 continued: about $3,181,139

Question 5

Graph payment rates

```
alt.Chart(grouped_stickers_ultra).mark_bar().encode(
    x = alt.X("year(issue_date):T"),
    y = alt.Y("payment_rate:Q")
    )
```



Answer Q5 It appears that after the date in which the new sticker ticket cost was launched (or after the year it was launched, 2012, at any rate) the rate of payment decreased somewhat. This suggests that the higher cost may have made it more difficult for drivers to repay the ticket, which would be consistent with the Propublica reporting, as the reporters spoke at length about the fact that working class and poor folks were more likely to not have a sticker, and more likely not to have enough money to pay a fine for not having one. That would suggest that there is a negative relationship between changes in ticket price and changes in payment rate, which the data appears to bear out.

Question 6

Group by violation code and check ticket number and payment rate

```
grouped_up = tickets[["ticket_here", "paid_off_tickets"]].groupby(tickets["violation_code"])

#add the payment rate variable
grouped_up["payment_rate"] = grouped_up["paid_off_tickets"] / grouped_up["ticket_here"]
```

Subset for payment rates over 60% and ticket numbers over 1,000

```
grouped_up[(grouped_up["ticket_here"] >= 2000) & (grouped_up["payment_rate"] >= .75)]

#Accreditation: asked ChatGPT
# "why my code wasn't working"
# and was told I need to add
# parentheses around my conditions
```

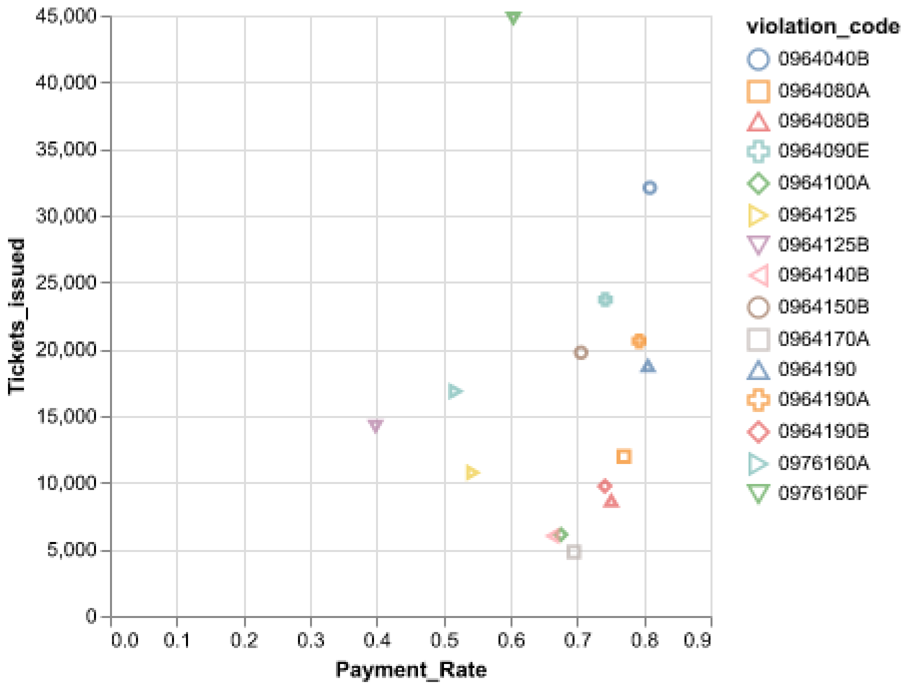
	violation_code	ticket_here	paid_off_tickets	payment_rate
9	0964040B	32082	25957	0.809083
18	0964080A	11965	9218	0.770414
19	0964080B	8640	6490	0.751157
29	0964100G	2191	1769	0.807394
59	0964190	18756	15124	0.806355
60	0964190A	20600	16334	0.792913

Making a graph showing ticket numbers and payment rates

```
grouped_up_high_rate = grouped_up[grouped_up["ticket_here"] >= 4000]

alt.Chart(grouped_up_high_rate).mark_point().encode(
    alt.X("payment_rate:Q", title = "Payment_Rate"),
    alt.Y("ticket_here:Q", title = "Tickets_issued"),
    alt.Color("violation_code"),
    shape = "violation_code")
```

```
#using lecture viz_5 and
# https://altair-viz.github.io/user_guide/encodings/index.html for reference
```



Answer Q 6 As the graph above displays, there are a few violation codes that maximize payment rate and ticket issuance. Assuming that costs don't change behavior here, the codes for the 3 ticket types that I believe would be the best investment are 0976160F, 0964040B, and 0964190A. This is because these 3 codes have very high rates of payment and a very high number have been issued. There are some others that could be contenders as well, based on their position on the graph.

Part 3

Question 1 Modify groupby code from earlier to add average fine_level1_amount

```
g_avg_fine_one = tickets[["fine_level1_amount"]].groupby(tickets["violation_code"]).mean().r

#join to grouped_up, the df containing
# payment rates and violation quantities
total_group = pd.merge(grouped_up, g_avg_fine_one, how = "inner",
on = "violation_code")
```

```
#Accreditation, I asked ChatGPT how to
# merge without NA values appearing,
# merge() using the common column
# violation_code was one of the
# solutions offered
```

Sort by ticket quantities

```
#using:
# https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.sort_values.html
total_group.sort_values(by = ["ticket_here"], ascending = False)

# Subset for 5 highest violation
# code ticket quantities
print(total_group.iloc[[94, 9, 20, 60, 50], :])
```

```
   violation_code  ticket_here  paid_off_tickets  payment_rate  \
94       0976160F        44811             27082      0.604361
9        0964040B        32082             25957      0.809083
20       0964090E        23683             17579      0.742262
60       0964190A        20600             16334      0.792913
50       0964150B        19753             13942      0.705817

    fine_level1_amount
94           54.968869
9            53.583629
20           66.338302
60           46.598058
50           66.142864
```

Question 2

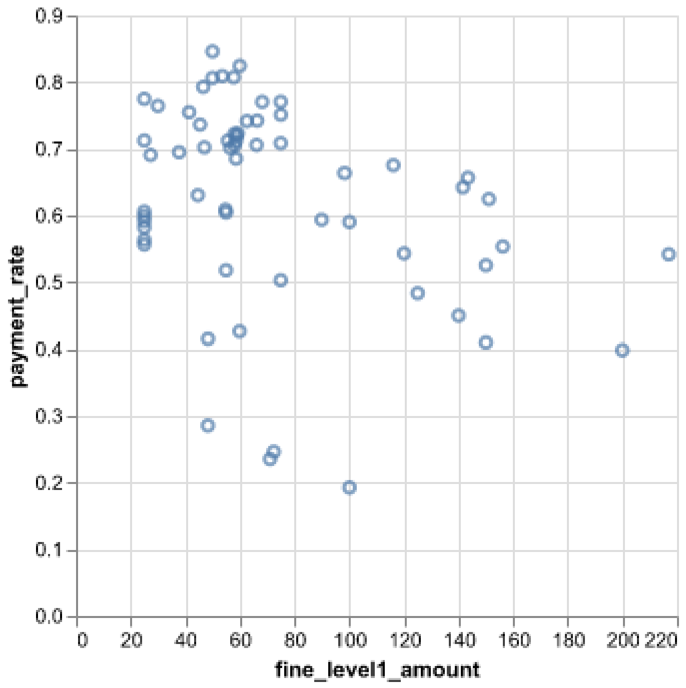
Scatterplot Remember that in total_group, "finelevel1_amount" has been averaged already.

```
#subset to remove outlier and codes with
# less than 100 tickets
mod_total_group = total_group[(total_group["ticket_here"] >= 100) & (total_group["fine_level1

scatter_plot = alt.Chart(mod_total_group).mark_point().encode(
    x = alt.X("fine_level1_amount"),
    y = alt.Y("payment_rate"),
   )

#Attribution, I was receiving
# this error message:
# SyntaxError: positional argument
# follows keyword argument
# and ChatGPT advised me to change the
# syntax of alt.Color

scatter_plot.show()
```

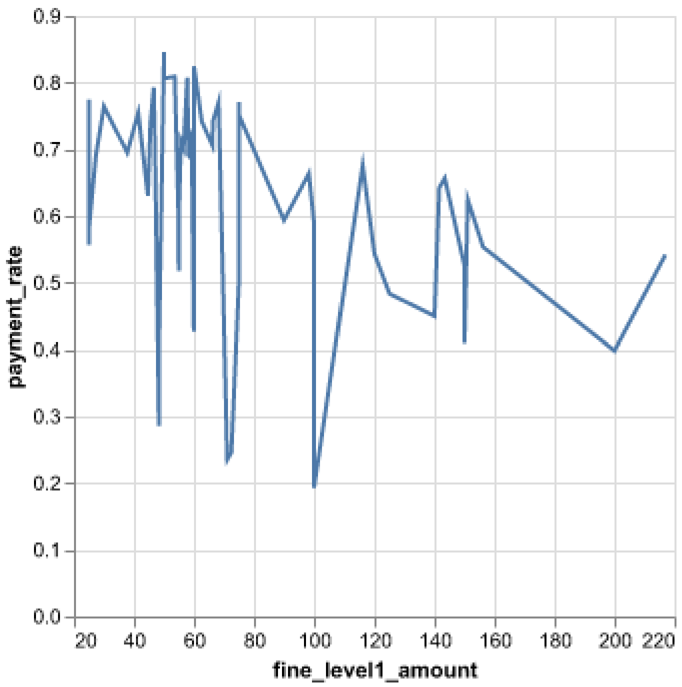


Scatterplot analysis: Headlines: This scatterplot is useful because it appears to show a negative relationship between payment rate and average fine amount. Sub-messages: The majority of high-citation-amount tickets (we filtered out the low-citation-amount ones) seem to have fine levels between \$20 and \$65, and payment rates between 60% and 80%. Tickets with higher fines and lower rates appear less common.

Line plot Remember that in total_group, "finelevel1_amount" has been averaged already.

```
# reuse mod_total_group from before to screen
# (some) outliers
line_plot = alt.Chart(mod_total_group).mark_line().encode(
    x = alt.X("fine_level1_amount"),
    y = alt.Y("payment_rate:Q"),
)

line_plot.show()
```



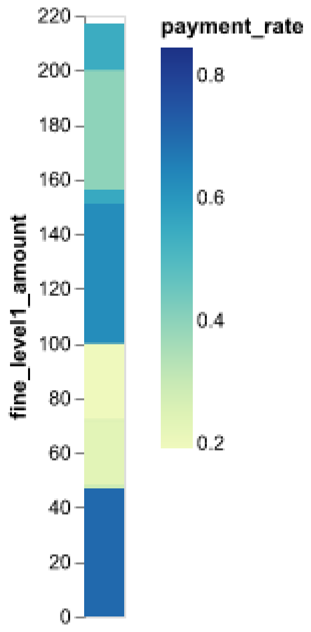
Line plot analysis: Headlines: This scatterplot is useful because it appears to show a negative relationship between payment rate and average fine amount, just like the scatterplot. But this version draws more attention to the outliers, which cause the line to leap down and then up in several places, indicating that there are a number of ticket types with payment rates that seem strange when compared to their fine levels. Sub-messages: The clustering mentioned in the scatterplot remains, and this graph adds little that the scatterplot has not captured already. But I think it would be easier for a person without much experience reading graphs to quickly comprehend.

Heatmap Remember that in total_group, "finelevel1_amount" has been averaged already.

Using: https://idl.uw.edu/visualization-curriculum/altair_scales_axes_legends.html#configuring-scales-and-axes

```
heat_plot = alt.Chart(mod_total_group).mark_rect().encode(
    y = alt.Y("fine_level1_amount:Q",
    sort = alt.EncodingSortField("payment_rate:Q")),
    color = alt.Color("payment_rate:Q")
)

heat_plot.show()
```



Heatmap analysis Headlines: The heatmap tells a slightly different story than the line or scatter plots, because it draws attention to the fact that the lowest payment rates are actually not associated with the tickets with the highest fines. Instead they appear to be associated with tickets that have relatively middle-of-the-road fine costs, between around \$60 to \$100. This could be related to the clustering that the line and scatter plots suggest, where most high-issuance-amount tickets have payment rates between 60% and 80%, and fine levels between \$20 and \$65. Sub-message: this plot takes the most time to understand. Also, viewing the plots in conversation with each other maximizes understanding.

Question 3

Answer Q 3: I would show the Clerk my line plot. Although I suppose technically the line plot works, in some ways, like a regression (though technically it cannot be interpreted as one at all, since it does not utilize any actual regression techniques, it does draw a line between points) what it lacks in causal analysis it makes up for in clarity. The line plot is intuitive and easy to explain, and so it is useful for explaining the data to a person in a hurry, though it should be interpreted as a causal analysis. Also, I would likely layer the scatterplot on top of the line plot, as the two work well together, in terms of communication.

Part 4

Question 1

Finding doubled fine_level2_amount codes

```
# select fine_level2_amount and groupby
# violation code
# and take the average of the fine
#don't reset index because this
# object is going into
#total_group

#save as DataFrame
group_fine_2 = tickets["fine_level2_amount"].groupby(tickets["violation_code"]).mean()

# merge with total_group
total_group = pd.merge(total_group, group_fine_2,
how = "inner", on = "violation_code")
```

subset for entries where fine_level2_amount that are greater than or less than 2 * fine_level1_amount

```
weird_group = total_group[(total_group["fine_level2_amount"] > total_group["fine_level1_amou
| (total_group["fine_level2_amount"] < total_group["fine_level1_amount"] * 2) ]
```

Subset weird_group for tickets with more than 100 entries

```
big_weird_group = weird_group[weird_group["ticket_here"] >= 100]

print(big_weird_group)
```

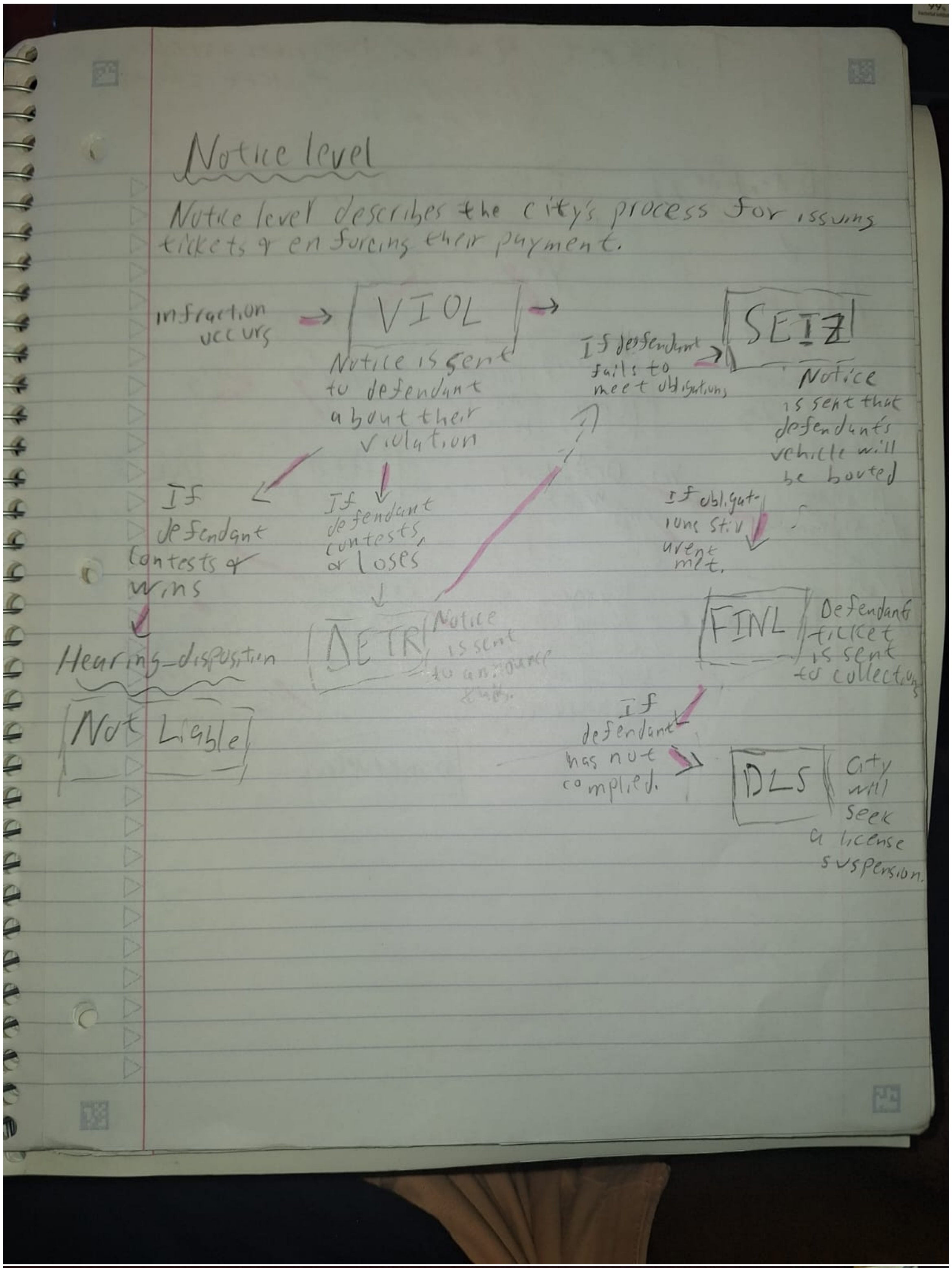
```
    violation_code  ticket_here  paid_off_tickets  payment_rate  \
1          0940060          236               155      0.656780
```

```
13        0964050J         2034              1102      0.541790
25        0964100C         1579              1014      0.642179
43        0964125C          131                16      0.122137
45         0964130         2050              1077      0.525366
103       0976220A          271               150      0.553506
104       0976220B         1697              1060      0.624632

     fine_level1_amount  fine_level2_amount
1            143.432203          278.601695
13           216.986234          358.308751
25           141.592780          266.751108
43           500.000000          955.343511
45           150.000000          259.926829
103          156.180812          225.645756
104          151.090159          209.516794
```

Answer Q1 As the table above demonstrates, the violation codes where ticket prices do not, on average double when they are not paid (acknowledging that we screened out codes with particularly low entry numbers) are all fines with relatively high costs, the lowest (0964100C) costs about $142, and none of them actually have fine penalties that result in secondary fines increasing to more than twice the original fine. Perhaps it was decided that they were too high to be doubled.

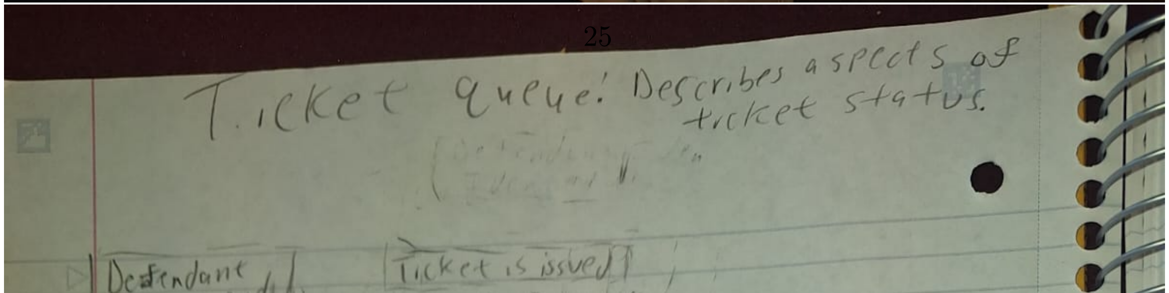
Question 2



Notice level

Notice level describes the city's process for issuing tickets & enforcing their payment.

infraction occurs
VIOL
Notice is sent to defendant about their violation
If defendant contests & wins
If defendant contests, or loses
Hearing_disposition
Not Liable
DETR
Notice is sent to announce [illegible]
If defendant fails to meet obligations
SEIZ
Notice is sent that defendant's vehicle will be booted
If obligations still arent met.
FINL
Defendant's ticket is sent to collections
If defendant has not complied.
DLS
City will seek a license suspension.



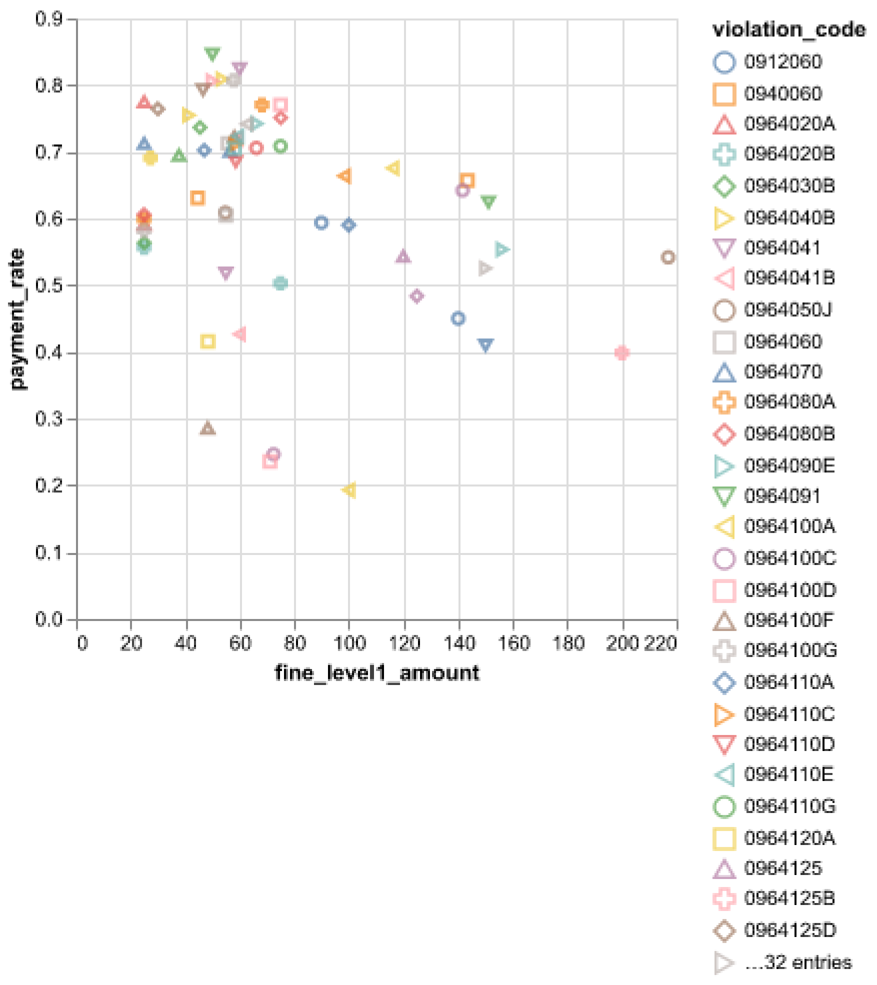
Ticket queue: Describes aspects of ticket status.

Question 3

Modifying Part 3 scatterplot

```
mod_scatter_plot = alt.Chart(mod_total_group).mark_point().encode(
    x = alt.X("fine_level1_amount"),
    y = alt.Y("payment_rate"),
    color = alt.Color("violation_code:N"),
    shape = "violation_code:N")

mod_scatter_plot.show()
```


payment_rate
fine_level1_amount
violation_code
0964020A
0964020B
0964030B
0964040B
0964041B
0964050J
0964080A
0964080B
0964090E
0964100A
0964100C
0964100D
0964100F
0964100G
0964110A
0964110C
0964110D
0964110E
0964110G
0964120A
0964125B
0964125D
…32 entries