Johnston540Milestone5

September 20, 2024

Property vs Crime Rate Analysis

[3]:
```
# Imports the necessary libraries/packages

from requests import Request, Session
from requests.exceptions import ConnectionError, Timeout, TooManyRedirects
import json
import pandas as pd
from pprint import pprint
import urllib.request
from bs4 import BeautifulSoup
from textwrap import wrap
import requests
from time import time, ctime
import re
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from bs4 import BeautifulSoup
import requests
import codecs
import os
import datetime
import time
```

MILESTONE 2: Cleaning/Formatting Flat File Source

[5]:
```
# Loads and displays the flat file

realtor_df = pd.read_csv("realtor-data.zip.csv")

realtor_df.head()
```

[5]:

	brokered_by	status	price	bed	bath	acre_lot	street	\
0	103378.0	for_sale	105000.0	3.0	2.0	0.12	1962661.0	
1	52707.0	for_sale	80000.0	4.0	2.0	0.08	1902874.0	
2	103379.0	for_sale	67000.0	2.0	1.0	0.15	1404990.0	
3	31239.0	for_sale	145000.0	4.0	2.0	0.10	1947675.0	

```
4      34632.0  for_sale   65000.0  6.0   2.0      0.05   331151.0

          city        state  zip_code  house_size prev_sold_date
0    Adjuntas  Puerto Rico     601.0       920.0            NaN
1    Adjuntas  Puerto Rico     601.0      1527.0            NaN
2  Juana Diaz  Puerto Rico     795.0       748.0            NaN
3       Ponce  Puerto Rico     731.0      1800.0            NaN
4   Mayaguez  Puerto Rico     680.0         NaN            NaN
```

[6]:
```
# displays the number of rows and columns

print("The dataframe has {} rows and {} columns".format(*realtor_df.shape))
```

```
The dataframe has 2226382 rows and 12 columns
```

[7]:
```
### First Transormation: Renaming Headers
```

[8]:
```
realtor_df = realtor_df.rename(columns = {'house_size': 'Square Footage',␣
 ↪"brokered_by": "Brokerage",
                                           "status": "Status", "price": "Price",␣
 ↪"bed": "Number of Beds",
                                           "bath": "Number of Baths", "acre_lot":␣
 ↪"Lot_Acerage",
                                           "street": "Street Name", "city": "City␣
 ↪Name", "state": "Location",
                                           "zip_code": "Zip Code",␣
 ↪"prev_sold_date": "Past Sale Date"})
print(pd.Series(realtor_df.columns))
```

```
0           Brokerage
1              Status
2               Price
3      Number of Beds
4     Number of Baths
5         Lot_Acerage
6         Street Name
7           City Name
8            Location
9            Zip Code
10     Square Footage
11     Past Sale Date
dtype: object
```

I am pretty happy with most of the columns names aside from 'house_size' I feel like square footage makes much more sense for this scenario. All other columns I slightly adjusted to be more readable/aesthetically pleasing.

Second Transormation: replacing missing values

[11]:
```
# Locates columns with missing values

missing = realtor_df.isna().any()
print("Columns with missing values:\n", missing)
```

```
Columns with missing values:
 Brokerage           True
Status              False
Price                True
Number of Beds       True
Number of Baths      True
Lot_Acerage          True
Street Name          True
City Name            True
Location             True
Zip Code             True
Square Footage       True
Past Sale Date       True
dtype: bool
```

[12]:
```
# Fills all missing values with NaN

realtor_df.fillna(np.nan, inplace = True)

print(realtor_df.head())
```

```
   Brokerage    Status     Price  Number of Beds  Number of Baths  \
0   103378.0  for_sale  105000.0             3.0              2.0
1    52707.0  for_sale   80000.0             4.0              2.0
2   103379.0  for_sale   67000.0             2.0              1.0
3    31239.0  for_sale  145000.0             4.0              2.0
4    34632.0  for_sale   65000.0             6.0              2.0

   Lot_Acerage  Street Name   City Name     Location  Zip Code  \
0         0.12    1962661.0    Adjuntas  Puerto Rico     601.0
1         0.08    1902874.0    Adjuntas  Puerto Rico     601.0
2         0.15    1404990.0  Juana Diaz  Puerto Rico     795.0
3         0.10    1947675.0       Ponce  Puerto Rico     731.0
4         0.05     331151.0    Mayaguez  Puerto Rico     680.0

   Square Footage Past Sale Date
0           920.0            NaN
1          1527.0            NaN
2           748.0            NaN
3          1800.0            NaN
4             NaN            NaN
```

The dataset currently has NaN values where data is not present. I chose to continue that process

instead of setting the NaN's to a different value. I did this to stop the possibility of creating unnecessary outliers within the dataset and in the future subset.

Third Transformation: creating a subset

[15]:
```
# Seletcs the most important columns

subset = realtor_df[["Price", "Number of Baths",
                     "Number of Beds", "Square Footage",
                     "Location", "Lot_Acerage"]]
```

[16]:
```
subset.head()
```

[16]:

	Price	Number of Baths	Number of Beds	Square Footage	Location	\
0	105000.0	2.0	3.0	920.0	Puerto Rico	
1	80000.0	2.0	4.0	1527.0	Puerto Rico	
2	67000.0	1.0	2.0	748.0	Puerto Rico	
3	145000.0	2.0	4.0	1800.0	Puerto Rico	
4	65000.0	2.0	6.0	NaN	Puerto Rico	

	Lot_Acerage
0	0.12
1	0.08
2	0.15
3	0.10
4	0.05

I chose these columns as I felt that they were the most valuable within the original dataset. The biggest possible issue could be my decision to not include zip codes. I made this decision with the viewpoint of having 50 states to evaluate. If I were to choose regions than there would be hundreds of thousands of areas to evaluate against eachother.

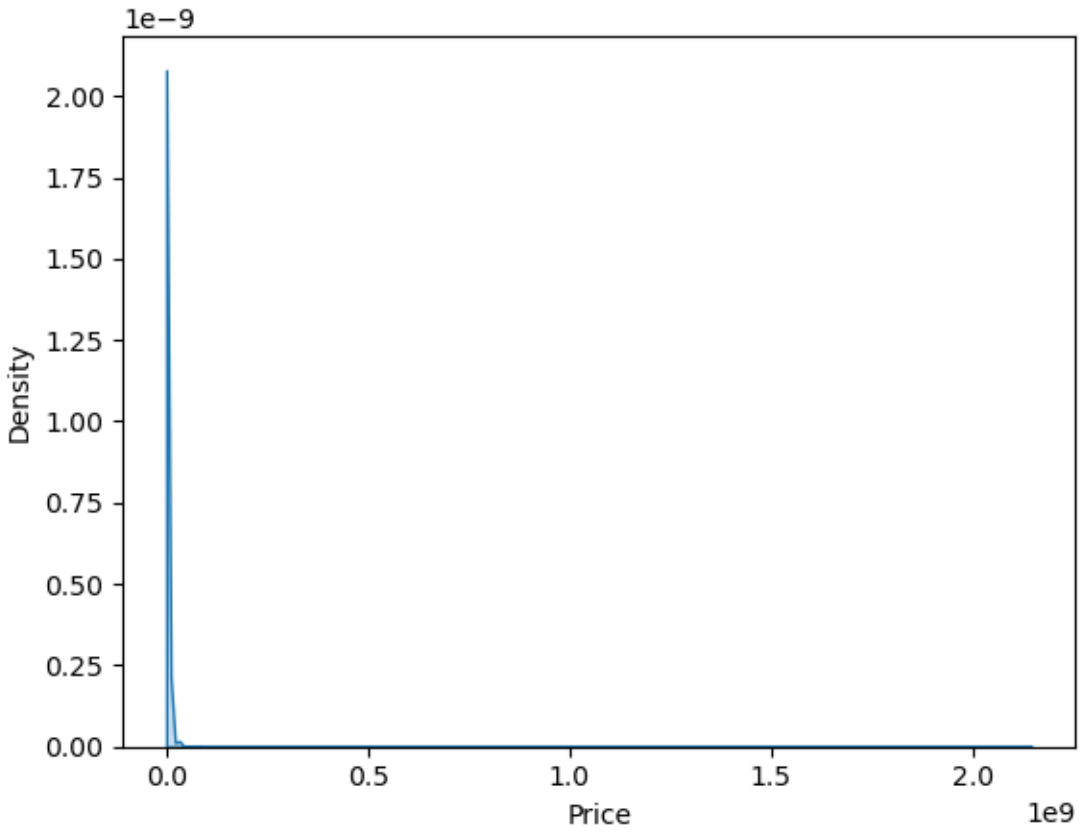
Fourth Transformation: checking for duplicates/outliers

[19]:
```
# displays the statistics for the subset

print(subset.describe())
```

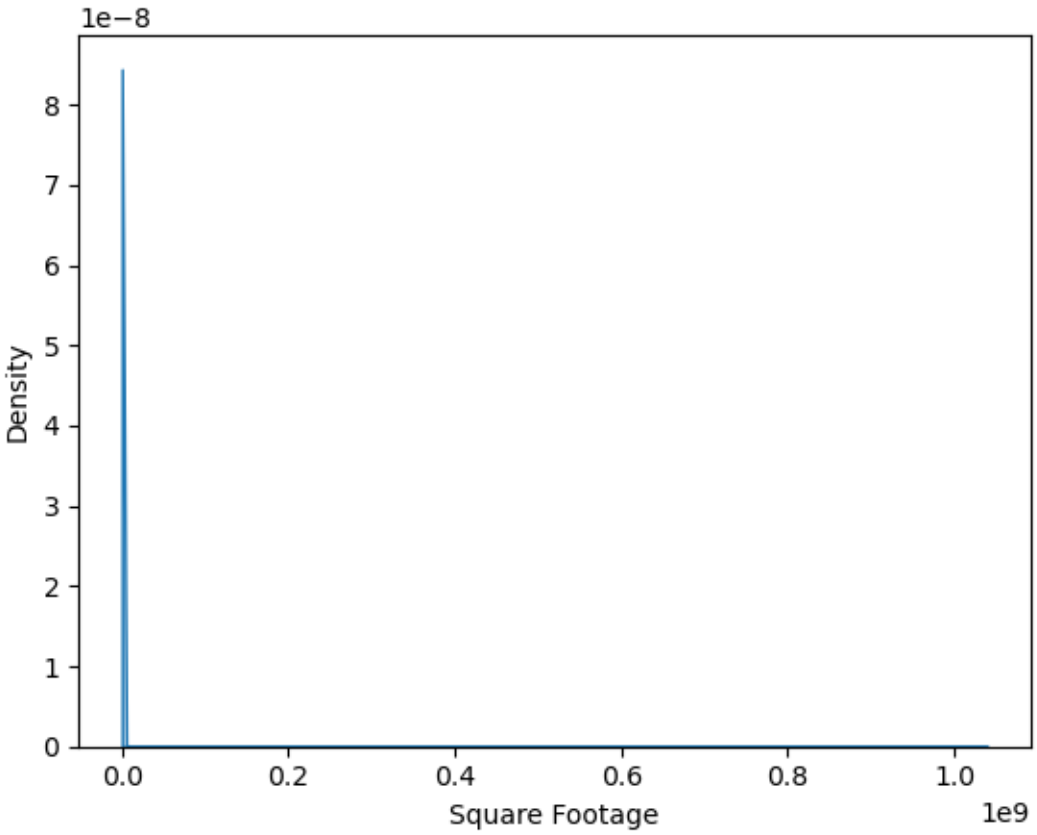
```
              Price  Number of Baths  Number of Beds  Square Footage  \
count  2.224841e+06     1.714611e+06    1.745065e+06    1.657898e+06
mean   5.241955e+05     2.496440e+00    3.275841e+00    2.714471e+03
std    2.138893e+06     1.652573e+00    1.567274e+00    8.081635e+05
min    0.000000e+00     1.000000e+00    1.000000e+00    4.000000e+00
25%    1.650000e+05     2.000000e+00    3.000000e+00    1.300000e+03
50%    3.250000e+05     2.000000e+00    3.000000e+00    1.760000e+03
75%    5.500000e+05     3.000000e+00    4.000000e+00    2.413000e+03
max    2.147484e+09     8.300000e+02    4.730000e+02    1.040400e+09

        Lot_Acerage
```

```
count  1.900793e+06
mean   1.522303e+01
std    7.628238e+02
min    0.000000e+00
25%    1.500000e-01
50%    2.600000e-01
75%    9.800000e-01
max    1.000000e+05
```

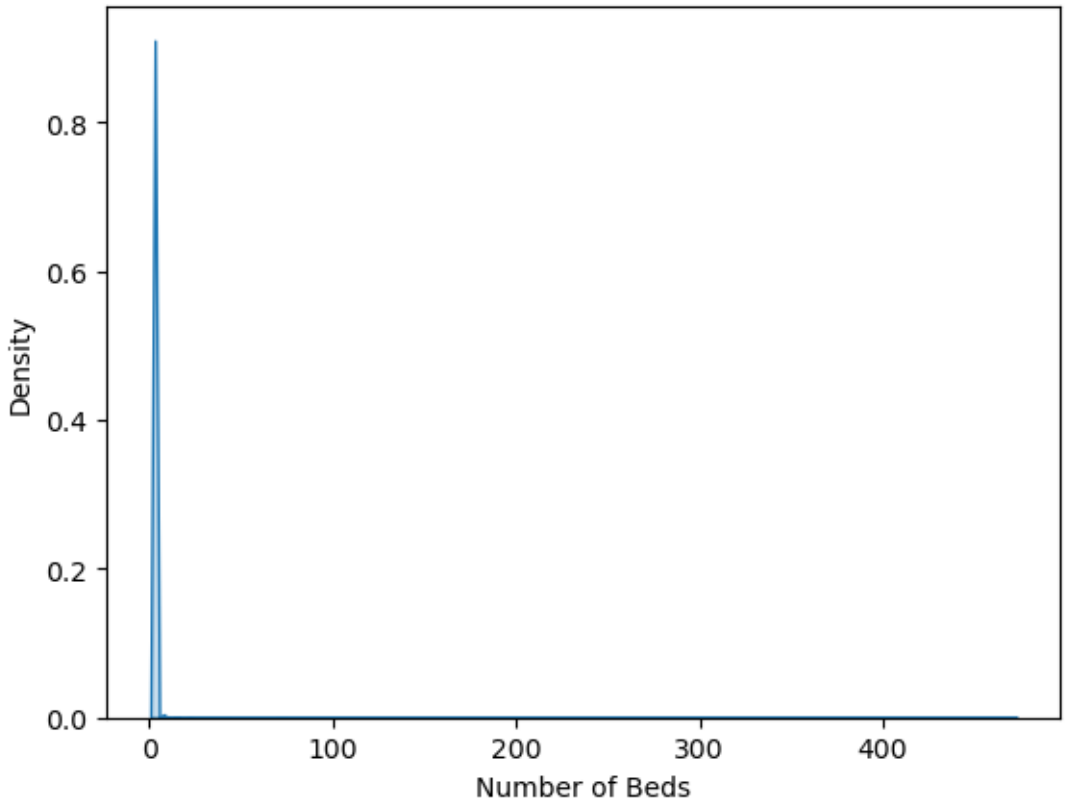
[20]:
```
# displays a density chart to see any outliers for each column

sns.kdeplot(subset["Price"], fill = True)
plt.show()
```



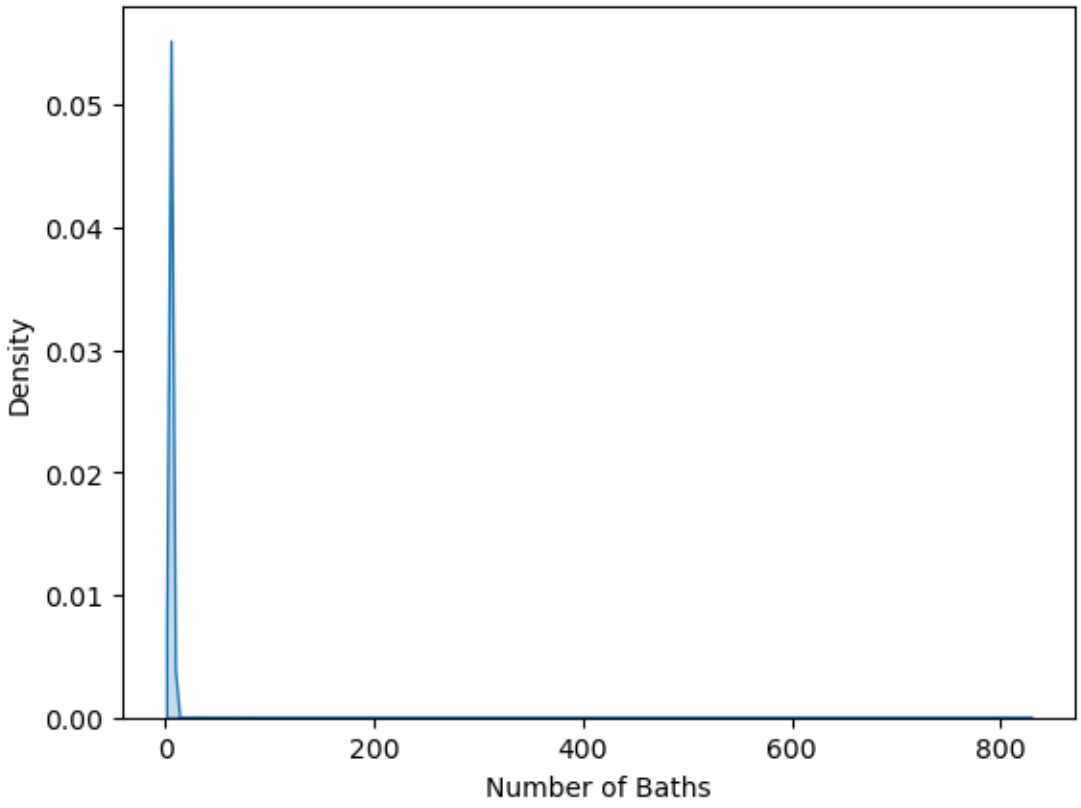
[21]:
```
sns.kdeplot(subset["Square Footage"], fill = True)
plt.show()
```



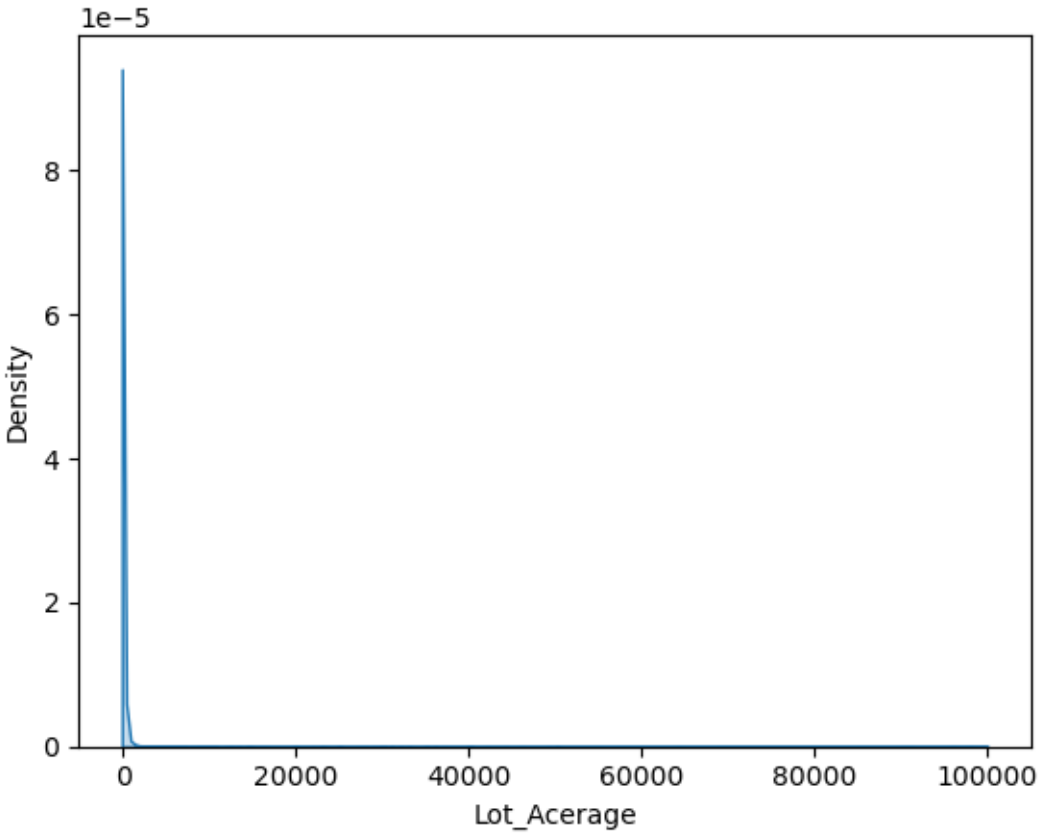
[22]:
```
sns.kdeplot(subset["Number of Beds"], fill = True)
plt.show()
```



[23]:
```
sns.kdeplot(subset["Number of Baths"], fill = True)
plt.show()
```



[24]:
```
sns.kdeplot(subset["Lot_Acerage"], fill = True)
plt.show()
```



There are not any spikes within the graphs beyond each ones most dense point. This indicates that whilst the appearance of higher than average value points exist that within these selected columns for the subset that there are no massive outliers in need of removal.

Fifth Transformation: Re-orders the data in alphabetical order of the State column

[27]:
```
subset = subset.sort_values(by = "Location")
print(display(subset.head(5)))
```

```
          Price  Number of Baths  Number of Beds  Square Footage Location  \
589894  220000.0              2.0             3.0          2243.0  Alabama
583362  549900.0              3.0             4.0          2400.0  Alabama
583363  780000.0              3.0             4.0          2400.0  Alabama
583364  749000.0              NaN             NaN             NaN  Alabama
583365  279000.0              NaN             NaN             NaN  Alabama

        Lot_Acerage
589894         0.34
583362        17.00
583363        56.08
583364         7.95
```

```
583365          0.62
```

```
None
```

The original format was in order of input ID in accordance to the datas first form. I have decided to order it by the only non numerical column within the dataset, State.

Ethical Implications:

The first transformation does not have any ethical implications as the column names are not changed to anything substantially different than they originally were. The second transformation does not pose any ethical complications either as it only fills missing values. The third transformation does pose a potential complication. The removal/exclusion of zip codes into the subset means that I will be classifying states as a whole rather than certain areas within them. The fourth and fifth transformation do not pose any ethical worries as the data is not changed in any way it is only adjusted format and order wise.

Step 0/Prep Phase

[31]:
```
# Connects to the wiki link

url = 'https://en.wikipedia.org/wiki/List_of_U.S.
 ↪_states_and_territories_by_violent_crime_rate'
page = requests.get(url)
crime_rate = BeautifulSoup(page.text, 'html')
```

[32]:
```
# Retrieves the appropriate table

crime_table = crime_rate.find_all('table', class_ = 'wikitable')[0]
```

[33]:
```
# Sets the structure of the table

def table_structure(crime_table):

    rows = [x for x in crime_table.find_all('tr')]

    num_rows = len(rows)

    num_cols = max([len(x.find_all(['th','td'])) for x in rows])

    header_rows_set = [x.find_all(['th', 'td']) for x in rows if len(x.
 ↪find_all(['th', 'td']))>num_cols/2]

    num_cols_set = []

    for header_rows in header_rows_set:
        num_cols = 0
        for cell in header_rows:
            row_span, col_span = locate_spans(cell)
            num_cols+=len([cell.getText()]*col_span)
```

```
        num_cols_set.append(num_cols)

    num_cols = max(num_cols_set)

    return (rows, num_rows, num_cols)
```

[34]:
```
# Locates which columns & rows have spans

def locate_spans(cell):
        if cell.has_attr('rowspan'):
            rep_row = int(cell.attrs['rowspan'])
        else:
            rep_row = 1
        if cell.has_attr('colspan'):
            rep_col = int(cell.attrs['colspan'])
        else:
            rep_col = 1

        return (rep_row, rep_col)
```

[35]:
```
# Applies the data to the rows/columns

def apply_data(rows, num_rows, num_cols):
    data = pd.DataFrame(np.ones((num_rows, num_cols))*np.nan)
    for i, row in enumerate(rows):
        try:
            col_stat = data.iloc[i,:][data.iloc[i,:].isnull()].index[0]
        except IndexError:
            print(i, row)

        for j, cell in enumerate(row.find_all(['td', 'th'])):
            rep_row, rep_col = locate_spans(cell)

            while any(data.iloc[i,col_stat:col_stat+rep_col].notnull()):
                col_stat+=1

            data.iloc[i:i+rep_row,col_stat:col_stat+rep_col] = cell.
 ↪getText(strip = True)
            if col_stat<data.shape[1]-1:
                col_stat+=rep_col
    return data
```

[36]:
```
# Applies the above functions to the crime_table dataset

rows, num_rows, num_cols = table_structure(crime_table)
```

```
crime_table_df = apply_data(rows, num_rows, num_cols)

crime_table_df.head()
```

[36]:

	0	1	2	3	4	\
0	Location	Violentcrime	Homicide	Rape	Robbery	
1	United States	380.7	6.3	40.0	66.1	
2	District of Columbia	812.3	29.3	41.5	357.5	
3	New Mexico	780.5	12.0	54.6	110.6	
4	Alaska	758.9	9.5	134.0	75.1	

	5
0	Aggravatedassault
1	268.2
2	383.9
3	603.3
4	540.2

Transformation 1 - Re-organizing row 1 into the tables headers

[38]:
```
# displays current column names

list(crime_table_df.columns)
```

[38]:
```
[0, 1, 2, 3, 4, 5]
```

[39]:
```
# Adjusts the column names to the name of the first row

crime_table_df.columns = crime_table_df.iloc[0]

crime_table_df = crime_table_df[1:]

list(crime_table_df.columns)
```

[39]:
```
['Location',
 'Violentcrime',
 'Homicide',
 'Rape',
 'Robbery',
 'Aggravatedassault']
```

[40]:
```
# Displays the current state of the dataset

crime_table_df.head()
```

[40]:
```
0              Location Violentcrime Homicide   Rape Robbery Aggravatedassault
1         United States        380.7      6.3   40.0    66.1              268.2
2  District of Columbia        812.3     29.3   41.5   357.5              383.9
3            New Mexico        780.5     12.0   54.6   110.6              603.3
4                Alaska        758.9      9.5  134.0    75.1              540.2
5              Arkansas        645.3     10.2   76.0    39.7              519.4
```

Transformation 2 - Renaming/spacing column headers

[42]:
```
# Adjusts the column names

crime_table_df = crime_table_df.rename(columns = {'Violentcrime' :␣
 ↪'Violent_Crime', 'Aggravatedassault' : 'Aggravated_Assault'})

crime_table_df.head()
```

[42]:
```
0              Location Violent_Crime Homicide   Rape Robbery  \
1         United States         380.7      6.3   40.0    66.1
2  District of Columbia         812.3     29.3   41.5   357.5
3            New Mexico         780.5     12.0   54.6   110.6
4                Alaska         758.9      9.5  134.0    75.1
5              Arkansas         645.3     10.2   76.0    39.7

0 Aggravated_Assault
1              268.2
2              383.9
3              603.3
4              540.2
5              519.4
```

Transformation 3 - Find/remove duplicates

[44]:
```
# Searches for duplicates

crime_table_df.duplicated()
```

[44]:
```
1     False
2     False
3     False
4     False
5     False
6     False
7     False
8     False
9     False
10    False
11    False
12    False
```

```
13    False
14    False
15    False
16    False
17    False
18    False
19    False
20    False
21    False
22    False
23    False
24    False
25    False
26    False
27    False
28    False
29    False
30    False
31    False
32    False
33    False
34    False
35    False
36    False
37    False
38    False
39    False
40    False
41    False
42    False
43    False
44    False
45    False
46    False
47    False
48    False
49    False
50    False
51    False
52    False
dtype: bool
```

[45]:
```
# Removes any duplicate values not represented in the output above

crime_table_df = crime_table_df.drop_duplicates()

crime_table_df.shape
```

[45]:
```
(52, 6)
```

Transformation 4 - Removes the aggrevated assault column

I decided to do this as I believe that aggrevated assault is a violent crime which is already represented in the table

[47]:
```
# Removes column for the dataset

del crime_table_df['Aggravated_Assault']

crime_table_df.head(10)
```

[47]:

0	Location	Violent_Crime	Homicide	Rape	Robbery
1	United States	380.7	6.3	40.0	66.1
2	District of Columbia	812.3	29.3	41.5	357.5
3	New Mexico	780.5	12.0	54.6	110.6
4	Alaska	758.9	9.5	134.0	75.1
5	Arkansas	645.3	10.2	76.0	39.7
6	Louisiana	628.6	16.1	43.0	67.3
7	Tennessee	621.6	8.6	38.2	67.1
8	California	499.5	5.7	37.4	123.5
9	Colorado	492.5	6.4	63.4	72.6
10	South Carolina	491.3	11.2	38.2	40.6

Transformation 5 - Creates a hierarchy index

[49]:
```
# Assigns a hierarchy in accordance to crime severity (severity ranked in my␣
 ↪opinion, NOT INCLUDING location)

hierarchy = crime_table_df.copy()
hierarchy.index = [crime_table_df['Location'], crime_table_df['Homicide'],
                   crime_table_df['Rape'], crime_table_df['Violent_Crime'],␣
 ↪crime_table_df['Robbery']]

hierarchy.head(10)
```

[49]:

0	Location					\
Location	Homicide	Rape	Violent_Crime	Robbery		
United States	6.3	40.0	380.7	66.1	United States	
District of Columbia	29.3	41.5	812.3	357.5	District of Columbia	
New Mexico	12.0	54.6	780.5	110.6	New Mexico	
Alaska	9.5	134.0	758.9	75.1	Alaska	
Arkansas	10.2	76.0	645.3	39.7	Arkansas	
Louisiana	16.1	43.0	628.6	67.3	Louisiana	
Tennessee	8.6	38.2	621.6	67.1	Tennessee	
California	5.7	37.4	499.5	123.5	California	

```
Colorado             6.4      63.4  492.5         72.6                Colorado
South Carolina       11.2     38.2  491.3         40.6          South Carolina

0                                                          Violent_Crime  \
Location             Homicide Rape  Violent_Crime Robbery
United States        6.3      40.0  380.7         66.1            380.7
District of Columbia 29.3     41.5  812.3         357.5           812.3
New Mexico           12.0     54.6  780.5         110.6           780.5
Alaska               9.5      134.0 758.9         75.1            758.9
Arkansas             10.2     76.0  645.3         39.7            645.3
Louisiana            16.1     43.0  628.6         67.3            628.6
Tennessee            8.6      38.2  621.6         67.1            621.6
California           5.7      37.4  499.5         123.5           499.5
Colorado             6.4      63.4  492.5         72.6            492.5
South Carolina       11.2     38.2  491.3         40.6            491.3

0                                                          Homicide   Rape  \
Location             Homicide Rape  Violent_Crime Robbery
United States        6.3      40.0  380.7         66.1         6.3   40.0
District of Columbia 29.3     41.5  812.3         357.5       29.3   41.5
New Mexico           12.0     54.6  780.5         110.6       12.0   54.6
Alaska               9.5      134.0 758.9         75.1         9.5  134.0
Arkansas             10.2     76.0  645.3         39.7        10.2   76.0
Louisiana            16.1     43.0  628.6         67.3        16.1   43.0
Tennessee            8.6      38.2  621.6         67.1         8.6   38.2
California           5.7      37.4  499.5         123.5        5.7   37.4
Colorado             6.4      63.4  492.5         72.6         6.4   63.4
South Carolina       11.2     38.2  491.3         40.6        11.2   38.2

0                                                          Robbery
Location             Homicide Rape  Violent_Crime Robbery
United States        6.3      40.0  380.7         66.1         66.1
District of Columbia 29.3     41.5  812.3         357.5       357.5
New Mexico           12.0     54.6  780.5         110.6       110.6
Alaska               9.5      134.0 758.9         75.1         75.1
Arkansas             10.2     76.0  645.3         39.7         39.7
Louisiana            16.1     43.0  628.6         67.3         67.3
Tennessee            8.6      38.2  621.6         67.1         67.1
California           5.7      37.4  499.5         123.5       123.5
Colorado             6.4      63.4  492.5         72.6         72.6
South Carolina       11.2     38.2  491.3         40.6         40.6
```

Review:

The first change that I decided to make was to set row 0 as the headers of the dataset/ remove the current headers. This was purely an aesthetical step as having columns named '0,1,2,3,4,5' did not seem like the best state for the dataset to be in. The next change that I decided to make was renaming the column headers to have _ inbetween words for columns that are more than just

one word such as 'violentcrime' to 'Violent_Crime'. This change makes the dataset easier to read and overall more correct in a gramatical sense. The third change that I decided to make was to search for and drop any duplicates within the dataset. Looking back on this at the time of writing I should of also addressed any possible null/NaN values within the dataset as well during this step. I will make sure to do so when this dataset is next utilized. I checked for duplicates within the table and did not initially find any but I made sure to utilize the drop_duplicates function just to be safe. The next change I decided to make was to remove the Aggrevated_Assault column. There were 2 major reasons that I decided to make this change even though it is slightly ethically challenging as it means I see no value in the column. The first reason is that I believed that all aggrevated assaults to be violent crimes and that there is already a violent crime column displayed in the dataset/table. The second reason and the reason that I felt correct in doing this step is that there are 0 occurances in which the aggrevated assault column had a higher value than the violent crime column. This led me to believe that the aggrevated assault statistics were being measured within the violent crime column. If there were instances in which the aggrevated assault column was higher than I would have combined the columns together. The final change I decided to make was to create a hierarchy index in accordance of which crimes were more severe. Saying one crime is more severe than another is a process of displaying an opinion that will almost ALWAYS be ethically questionable. Personally I believe that ranking the columns for severity in the order of Homicide then Rape then Violent_Crime then Robbery is the correct order but if put to a group survey setting I am sure different opinions would arise. In terms of legal/regulatory guidelines I may have crossed a line when removing the aggrevated assault column but I believe I made an acceptable decision. A risk induced by my transformations is that my findings are inaccurate do to my assumptions during the column removal step and that my data is scrutinized for the order in which my hierarchy index is ranked. I believe the data I have chosen is credible as the last sentence of the first paragraph reads as follows, "These data have been taken from the FBI's Uniform Crime Reports". The data was aquired via FBI/police reports within each state so I believe that it can be labeled as 'ethically aquired'. I would mitigate the hierarchy order situation by placing an 'order placed in terms of personal opinion' disclosure right at the top of my report. I would mitigate the column removal by combining the two columns or reporting my reasoning for removing the column within my report as well.

Milestone 4 - Connecting to an API/Pulling in the Data and Cleaning/Formatting

[52]:
```
# Accesses an api call for a weekly forecast

weather_url ='https://api.openweathermap.org/data/2.5/forecast?lat=30.
  ↪2672&lon=97.7431&appid=5fa22e71723cd0ceb4edb5d0d39fcbcc'

weather_data = requests.get(weather_url).json()
```

[53]:
```
### Transformation 1 - Formats the api data into a table

weather_df = weather_data['list']
weather_table = pd.json_normalize(weather_df, max_level = 1)
weather_table.head(5)
```

[53]:
```
           dt                                             weather  visibility  \
0  1717297200  [{'id': 801, 'main': 'Clouds', 'description': …     10000.0
```

```
1  1717308000  [{'id': 802, 'main': 'Clouds', 'description': …     10000.0
2  1717318800  [{'id': 804, 'main': 'Clouds', 'description': …     10000.0
3  1717329600  [{'id': 803, 'main': 'Clouds', 'description': …     10000.0
4  1717340400  [{'id': 801, 'main': 'Clouds', 'description': …     10000.0

   pop               dt_txt  main.temp  main.feels_like  main.temp_min  \
0  0.0  2024-06-02 03:00:00     278.90           275.95         278.90
1  0.0  2024-06-02 06:00:00     283.09           280.65         283.09
2  0.0  2024-06-02 09:00:00     287.58           285.78         287.58
3  0.0  2024-06-02 12:00:00     283.84           281.96         283.84
4  0.0  2024-06-02 15:00:00     279.75           277.24         279.75

   main.temp_max  main.pressure  …  main.grnd_level  main.humidity  \
0         281.84           1012  …              582             58
1         285.92           1007  …              581             42
2         287.58            999  …              579             27
3         283.84           1002  …              580             38
4         279.75           1008  …              581             60

   main.temp_kf  clouds.all  wind.speed  wind.deg  wind.gust  sys.pod rain.3h  \
0         -2.94          13        3.96       231       6.41        d     NaN
1         -2.83          29        5.04       220       8.01        d     NaN
2          0.00          92        7.63       217       8.10        d     NaN
3          0.00          61        5.84       203       6.97        d     NaN
4          0.00          19        3.53       214       4.95        n     NaN

   snow.3h
0      NaN
1      NaN
2      NaN
3      NaN
4      NaN

[5 rows x 21 columns]
```

[54]:
```
### Transformation 2 - Isolates the weather variable

var_w = weather_table['weather']

var_w_list = []

for i in range(len(var_w)):
    a = var_w[i][0]
    print(a['description'])
    var_w_list.append(a['description'])
```

```
few clouds
```

```
scattered clouds
overcast clouds
broken clouds
few clouds
few clouds
clear sky
few clouds
broken clouds
broken clouds
overcast clouds
overcast clouds
broken clouds
light rain
light rain
light rain
light rain
snow
snow
light rain
overcast clouds
overcast clouds
overcast clouds
overcast clouds
overcast clouds
light rain
light rain
overcast clouds
broken clouds
broken clouds
scattered clouds
scattered clouds
scattered clouds
broken clouds
overcast clouds
overcast clouds
scattered clouds
few clouds
clear sky
few clouds
```

[55]:
```
### Transformation 3 - Converts the strings into datetime data

dt = weather_table['dt_txt']

fd = []
for i in range(len(dt)):
    newdt = datetime.datetime.strptime(dt[i], "%Y-%m-%d %H:%M:%S")
```

```
    fd.append(newdt)
```

[56]:
```
### Transformation 4 - Column manipulation

weather_table2 = weather_table.drop(weather_table.columns[[1,2,3,10,11]], axis
 ↪= 1)
weather_table2.head(3)
```

[56]:

	dt	dt_txt	main.temp	main.feels_like	main.temp_min	\
0	1717297200	2024-06-02 03:00:00	278.90	275.95	278.90	
1	1717308000	2024-06-02 06:00:00	283.09	280.65	283.09	
2	1717318800	2024-06-02 09:00:00	287.58	285.78	287.58	

	main.temp_max	main.pressure	main.humidity	main.temp_kf	clouds.all	\
0	281.84	1012	58	-2.94	13	
1	285.92	1007	42	-2.83	29	
2	287.58	999	27	0.00	92	

	wind.speed	wind.deg	wind.gust	sys.pod	rain.3h	snow.3h
0	3.96	231	6.41	d	NaN	NaN
1	5.04	220	8.01	d	NaN	NaN
2	7.63	217	8.10	d	NaN	NaN

[57]:
```
# Renames and organizes the data columns

weather_table2['weather'] = var_w_list

weather_table2['date_time'] = fd

weather_table2.rename(columns = {'main.feels_like':'feel temp in C'}, inplace =
 ↪True)

weather_table2['feel temp in C'] = weather_table2['feel temp in C']-273.15

weather_table3 = weather_table2[['weather','feel temp in C']]

weather_table3 = weather_table3.rename(columns = {'wind.speed' : 'wind_speed',
 ↪'Aggravatedassault' : 'Aggravated_Assault',
                                                   'feel temp in C' :
 ↪'Temperature'})

weather_table3.head(5)
```

[57]:

	weather	Temperature
0	few clouds	2.80
1	scattered clouds	7.50
2	overcast clouds	12.63

```
3     broken clouds         8.81
4        few clouds         4.09
```

[58]:
```
list = ['Texas', 'Texas', 'Texas', 'Texas', 'Texas', 'Texas', 'Texas', 'Texas',␣
 ↪'Texas', 'Texas', 'Texas','Texas',
       'Texas', 'Texas', 'Texas', 'Texas', 'Texas', 'Texas', 'Texas', 'Texas',␣
 ↪'Texas', 'Texas', 'Texas', 'Texas',
       'Texas', 'Texas', 'Texas', 'Texas', 'Texas', 'Texas', 'Texas', 'Texas',␣
 ↪'Texas', 'Texas', 'Texas', 'Texas',
       'Texas', 'Texas', 'Texas', 'Texas']

weather_table3['Location'] = list

weather_table3.head()
```

[58]:
```
            weather  Temperature Location
0        few clouds         2.80    Texas
1  scattered clouds         7.50    Texas
2   overcast clouds        12.63    Texas
3     broken clouds         8.81    Texas
4        few clouds         4.09    Texas
```

[59]:
```
### Transformation 5 - Checks for any duplicate/NaN values

w3_cols = weather_table3.columns

def Check_Duplicates():
    for i in np.arange(0,len(w3_cols)):
        print('There are {} of unique values in {} column out of {}'.
 ↪format(weather_table3[w3_cols[i]].nunique(), w3_cols[i],␣
 ↪len(weather_table3)))
print(Check_Duplicates())

print('variables with NA values\n', weather_table3.isna().sum())
```

```
There are 7 of unique values in weather column out of 40
There are 40 of unique values in Temperature column out of 40
There are 1 of unique values in Location column out of 40
None
variables with NA values
 weather        0
Temperature    0
Location       0
dtype: int64
```

Review:

Firstly, once again my data was aquired in an ethical way through a free-to-use public source

weather API. The first change I decided to make was to format the data into a table format as that is what I am most comfortable with at this point. The data for this step is set to the 'lat' and 'lon' of Austin TX as that is the area of the country that I would like to relocate to if I am unable to stay at my current location of Jacksonville FL. I will most likely do these steps for my ideal destinations that are 'tech' heavily orientated such as San Francisco CA and Atlanta GA. The next change I decided to make was to Isolate the weather variable description as it was deemed to me to be the most important variable. The next change I decided to make was to convert the date-time text into string format to be able to format is better in the following steps and ultimately set the foundations of a final subset. The next change I decided to make was to drop the unnecessary columns and rename the columns that were kept to a more readable/easier to interpret format and sement them into a final subset. The final change I made was to check for any duplicate or missing values within the final subset where I found that there were not any.

Milestone 5 - Merging the Data and Storing in a Database/Visualizing Data

[62]:
```
# Imports the required libraries

import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import requests
from bs4 import BeautifulSoup
import urllib.request
from urllib.request import urlopen
import seaborn as sns
import sqlite3
```

[63]:
```
# Connects to sqlite3 and creates a new database

conn = sqlite3.connect('final_clean_database.db')
```

[64]:
```
# Begins the table creating process

subset.to_sql('flat_file_table', conn, if_exists='replace', index=False)
crime_table_df.to_sql('web_data_table', conn, if_exists='replace', index=False)
weather_table3.to_sql('api_data_table', conn, if_exists='replace', index=False)
```

[64]: 40

[65]:
```
cursor = conn.cursor()

# join the tables through query

join_query = """
SELECT *
FROM flat_file_table AS fft
LEFT JOIN web_data_table AS fwd ON fft."Location" = fwd."Location"
```

```
LEFT JOIN api_data_table AS fad ON fft."Location"
"""


joined_df = pd.read_sql_query(join_query, conn)

joined_df.shape
```

[65]:
```
(2226382, 14)
```

[66]:
```
joined_df.head(3)
```

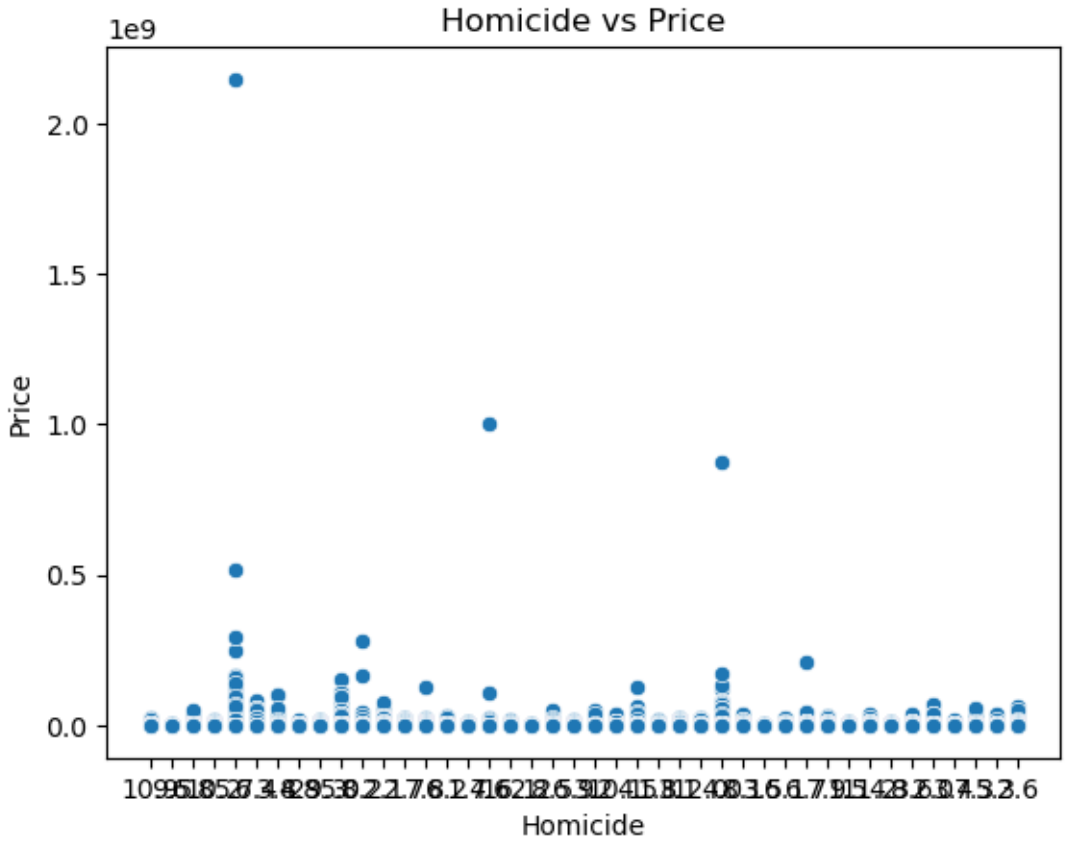
[66]:
```
       Price  Number of Baths  Number of Beds  Square Footage Location  \
0  220000.0              2.0             3.0          2243.0  Alabama
1  549900.0              3.0             4.0          2400.0  Alabama
2  780000.0              3.0             4.0          2400.0  Alabama

   Lot_Acerage Location Violent_Crime Homicide  Rape Robbery weather  \
0         0.34  Alabama         409.1     10.9  29.6    34.5    None
1        17.00  Alabama         409.1     10.9  29.6    34.5    None
2        56.08  Alabama         409.1     10.9  29.6    34.5    None

  Temperature Location
0        None     None
1        None     None
2        None     None
```

[67]:
```
joined_df.dtypes
```

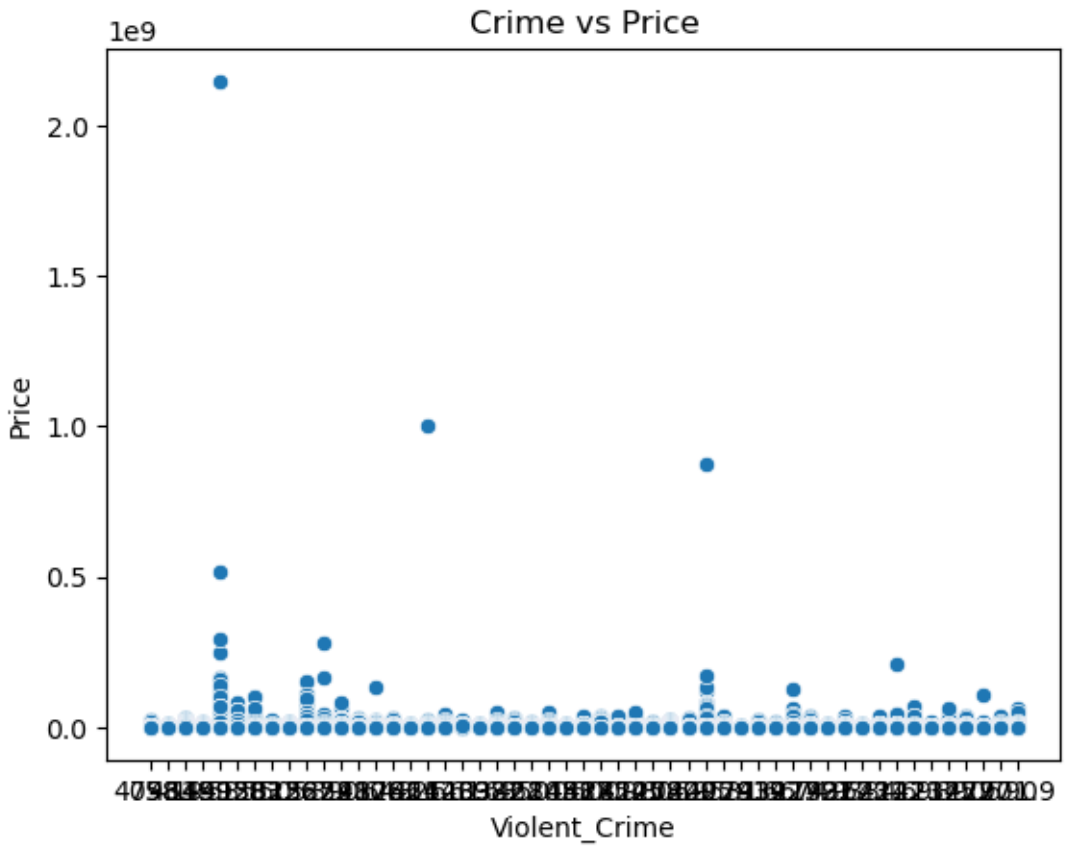
[67]:
```
Price              float64
Number of Baths    float64
Number of Beds     float64
Square Footage     float64
Location            object
Lot_Acerage        float64
Location            object
Violent_Crime       object
Homicide            object
Rape                object
Robbery             object
weather             object
Temperature         object
Location            object
dtype: object
```

[68]:
```
conn.close()
```

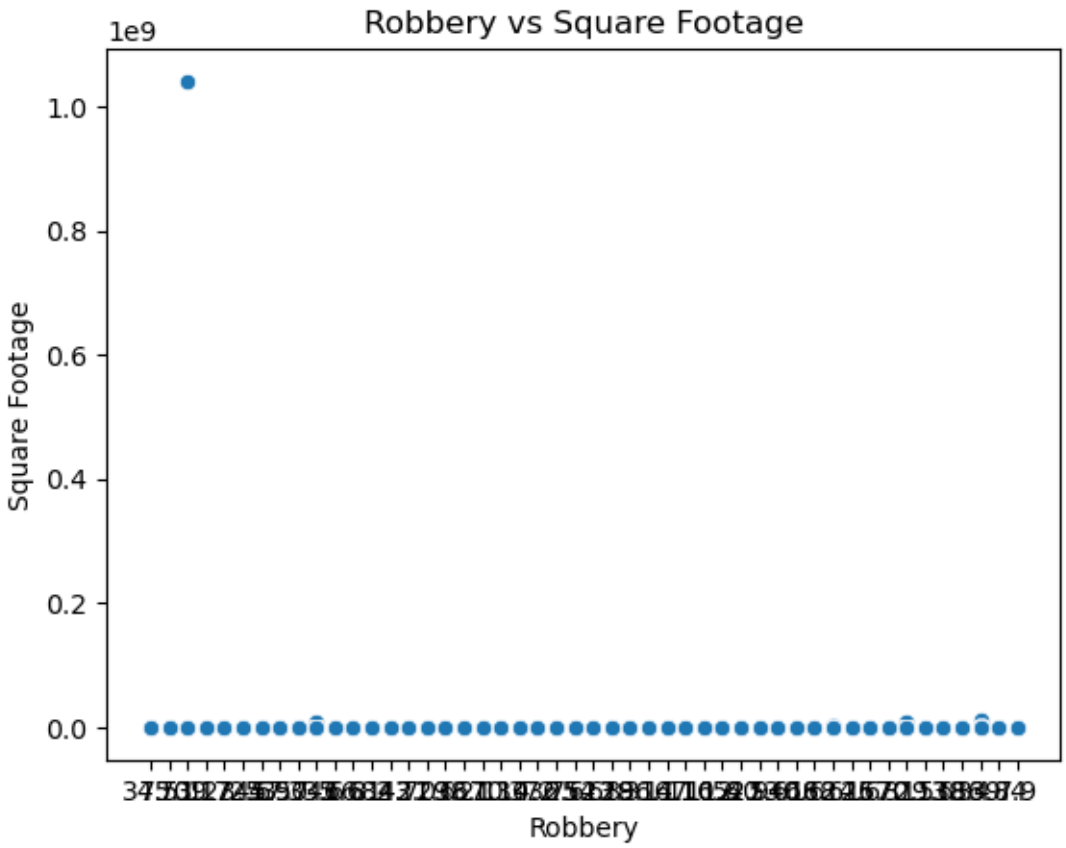
[69]:
```
# Visual 1: Homicide vs Price


sns.scatterplot(data = joined_df, x = "Homicide", y = "Price",)
plt.title("Homicide vs Price")
plt.xlabel("Homicide")
plt.ylabel("Price")
plt.show()
```



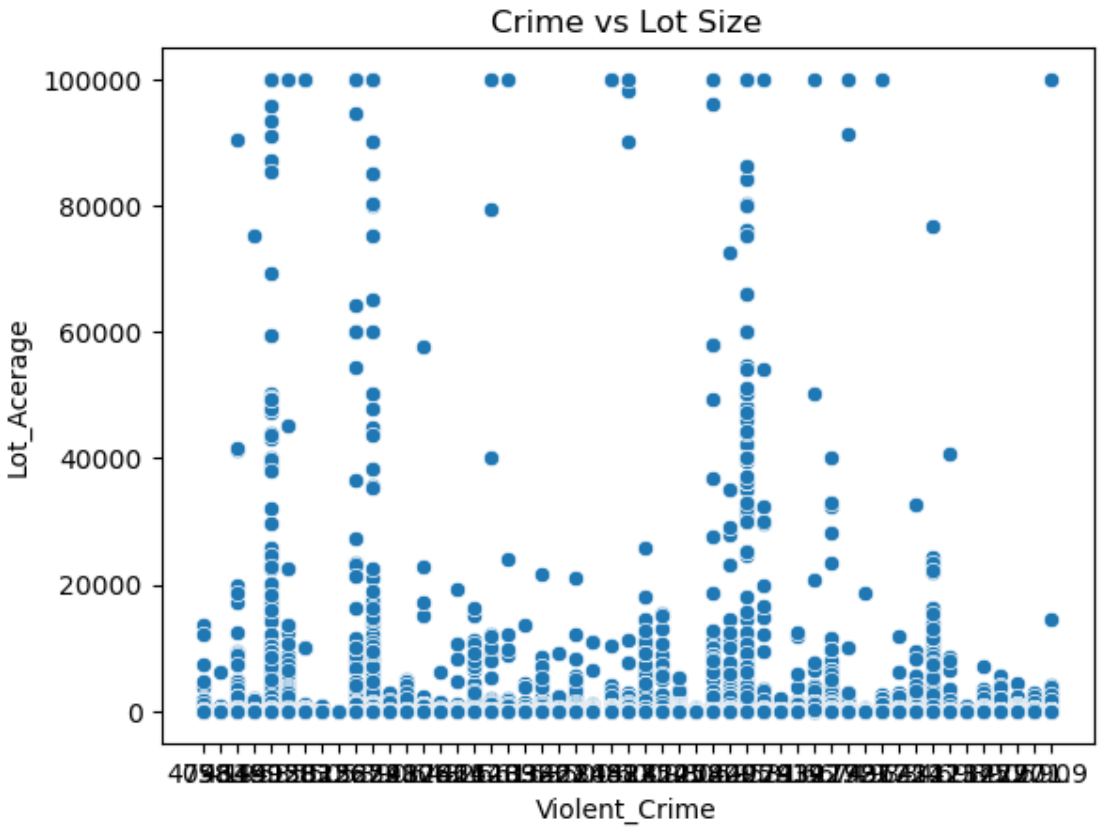
[70]:
```
# Visual 2: Crime vs Price


sns.scatterplot(data = joined_df, x = "Violent_Crime", y = "Price",)
plt.title("Crime vs Price")
plt.xlabel("Violent_Crime")
plt.ylabel("Price")
plt.show()
```



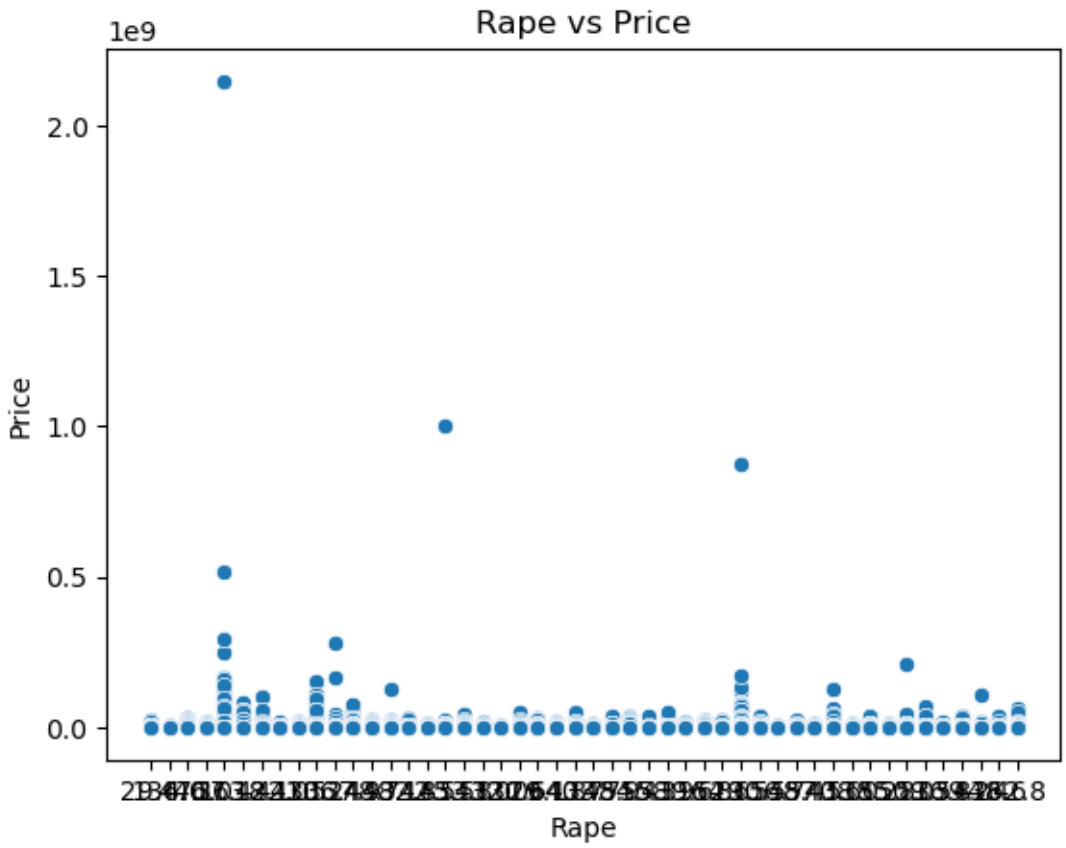
[71]:
```
# Visual 3: Robbery vs Square Footage


sns.scatterplot(data = joined_df, x = "Robbery", y = "Square Footage",)
plt.title("Robbery vs Square Footage")
plt.xlabel("Robbery")
plt.ylabel("Square Footage")
plt.show()
```



[72]:
```
# Visual 4: Crime vs Lot Size


sns.scatterplot(data = joined_df, x = "Violent_Crime", y = "Lot_Acerage",)
plt.title("Crime vs Lot Size")
plt.xlabel("Violent_Crime")
plt.ylabel("Lot_Acerage")
plt.show()
```



[73]:
```
# Visual 5: Rape vs Price


sns.scatterplot(data = joined_df, x = "Rape", y = "Price",)
plt.title("Rape vs Price")
plt.xlabel("Rape")
plt.ylabel("Price")
plt.show()
```



End of Project Write-up

There are plenty of ethical implications surrounding my results being that they are involved with finding out what factors into the committing of different crimes within an area. Whether that be Robbery, Rape, etc… . For what I have learned and vastly struggled with during this project is much easier to explain for me. I have developed a fear of the term API. Previously the most difficult milestone for me was the loading and cleaning of our API file with beautiful soup. This has by far surpassed it in terms of having to assign it a 'Location' column to parser through SQL and even achieving success in the query loading the data was extremely difficult to achieve. What I am happy to have achieved and further developed in is the use/area of html files. Retrieving them, converting them and cleaning them is something that I have noticeably gotten better at and more comfortable with. Like all final projects this was at times extremely frustrating but somewhat enjoyable to look back upon in review now that its over. This last milestone started with setting the cleaned files from the previous milestones into files that the query could form into a table. I had the table form them off of the 'Location' variable after going back and adding it to the API subset. Then fitted the data and compared different aspects of safety and housing and they are my more important factors for selecting a place to move and begin work after graduation. Ultimately I had a very good time during this course aside from a few frustrating moments on assignments and felt as though my ability to clean/prepare data has gone up a considerable amount. I also recently found out how to change these cells to markdown so that they're easier to read.