DSC 540

Inman, Gracie

Project Milestone 5

11/18/23

In [25]:
```
# Step 1 Import libraries
import pandas as pd
import numpy as np
from scipy import stats
```

In [26]:
```
# Step 2 Load and check data
cars_df = pd.read_csv('cars_raw.csv')
cars_df.head()
```

Out[26]:

	Year	Make	Model	Used/New	Price	ConsumerRating	ConsumerReviews	SellerType	SellerName	SellerRating	...	Interior(
0	2019	Toyota	Sienna SE	Used	$39,998	4.6	45	Dealer	CarMax Murrieta - Now offering Curbside Pickup...	3.3	...	
1	2018	Ford	F-150 Lariat	Used	$49,985	4.8	817	Dealer	Giant Chevrolet	4.8	...	
2	2017	RAM	1500 Laramie	Used	$41,860	4.7	495	Dealer	Gill Auto Group Madera	4.6	...	
3	2021	Honda	Accord Sport SE	Used	$28,500	5.0	36	Dealer	AutoSavvy Las Vegas	4.6	...	
4	2020	Lexus	RX 350	Used	$49,000	4.8	76	Dealer	Lexus of Henderson	4.8	...	

5 rows × 32 columns

Loading the data and checking it is important to ensure that the data was properly loaded. If not, it could affect analysis and lead to problems.

In [27]:

```
# Step 3 Remove duplicate rows based on all columns
cars_df_rd = cars_df.drop_duplicates()
prev_shape = cars_df.shape
new_shape = cars_df_rd.shape
print('The old shape of the data frame is', prev_shape)
print('The new shape of the data frame is', new_shape)
```

```
The old shape of the data frame is (9379, 32)
The new shape of the data frame is (8507, 32)
```

I selected for unique values to help eliminate biases within the data. This also helps improve accuracy.

In [28]:
```
# Step 4 Check for missing values
missing_values = cars_df_rd.isnull().sum()
print("Missing values in each column:")
print(missing_values)
```

```
Missing values in each column:
Year                     0
Make                     0
Model                    0
Used/New                 0
Price                    0
ConsumerRating           0
ConsumerReviews          0
SellerType               0
SellerName               0
SellerRating             0
SellerReviews            0
StreetName               0
State                    0
Zipcode                  0
DealType               206
ComfortRating            0
InteriorDesignRating     0
PerformanceRating        0
ValueForMoneyRating      0
ExteriorStylingRating    0
ReliabilityRating        0
ExteriorColor            0
InteriorColor            0
Drivetrain               0
MinMPG                   0
MaxMPG                   0
FuelType                 0
Transmission             0
Engine                   0
VIN                      0
Stock#                   0
Mileage                  0
dtype: int64
```

I checked for missing values as they can create issues with anaylsis. To ensure that analysis would go smoothly I wanted to see where and how many missing values were in the dataset so I can figure out how to handle them.

In [29]:
```
# Step 5 Remove rows with missing values
cars_cleaned = cars_df_rd.dropna()
```

I drop the rows with missing values for two reasons. One was there was only 206 rows with missing values and all of the missing values were located in the dealtype column. Two and the most important reason was I do not have the criteria for what was considered for each deal type. I felt I would not be able to accuratly replace these values.

In [30]:
```
# Step 6 Check missing values again
missing_values = cars_cleaned.isnull().sum()
print("Missing values in each column:")
print(missing_values)
```

```
Missing values in each column:
Year                     0
Make                     0
Model                    0
Used/New                 0
Price                    0
ConsumerRating           0
ConsumerReviews          0
SellerType               0
SellerName               0
SellerRating             0
SellerReviews            0
StreetName               0
State                    0
Zipcode                  0
DealType                 0
ComfortRating            0
InteriorDesignRating     0
PerformanceRating        0
ValueForMoneyRating      0
ExteriorStylingRating    0
ReliabilityRating        0
ExteriorColor            0
InteriorColor            0
Drivetrain               0
MinMPG                   0
MaxMPG                   0
FuelType                 0
Transmission             0
Engine                   0
VIN                      0
Stock#                   0
Mileage                  0
dtype: int64
```

I just wanted to check and make sure that the NA rows were dropped sucessfully.

In [31]:
```
# Step 7 Check shape change
original_shape = cars_df.shape
prev_shape = cars_df_rd.shape
new_shape = cars_cleaned.shape
print('The original shape of the data frame is', original_shape)
print('The previous shape of the data frame is', prev_shape)
print('The new shape of the data frame is', new_shape)
```

```
The original shape of the data frame is (9379, 32)
The previous shape of the data frame is (8507, 32)
The new shape of the data frame is (8301, 32)
```

I wanted to check how the dataset had changed from the begining. Since the begining of tranformation we have omitted 1,078 rows of data.

In [32]:
```
# Step 8 Get the column names
column_names = cars_cleaned.columns
# Print the column names
print(column_names)
```

```
Index(['Year', 'Make', 'Model', 'Used/New', 'Price', 'ConsumerRating',
       'ConsumerReviews', 'SellerType', 'SellerName', 'SellerRating',
       'SellerReviews', 'StreetName', 'State', 'Zipcode', 'DealType',
       'ComfortRating', 'InteriorDesignRating', 'PerformanceRating',
       'ValueForMoneyRating', 'ExteriorStylingRating', 'ReliabilityRating',
       'ExteriorColor', 'InteriorColor', 'Drivetrain', 'MinMPG', 'MaxMPG',
       'FuelType', 'Transmission', 'Engine', 'VIN', 'Stock#', 'Mileage'],
      dtype='object')
```

I wanted to check the column names to help identify columns to remove later and look at ones that may not be as good for analysis.

In [33]:
```
# Step 9 Checking to make sure columns contain unique data
Seller_type = cars_cleaned['SellerType'].value_counts()
print(Seller_type)
```

```
Dealer     8271
Private      30
Name: SellerType, dtype: int64
```

Only thirty rows contain privately sold vehicles. This is not a factor I will be using in my analysis so I may omit this column later.

In [34]:
```
used_or_new = cars_cleaned['Used/New'].value_counts()
print(used_or_new)
```

```
Used                        6989
BMW Certified                220
Mercedes-Benz Certified      201
Honda Certified              178
Toyota Certified             131
Cadillac Certified            86
Ford Certified                64
Subaru Certified              55
Jeep Certified                49
Nissan Certified              48
Acura Certified               42
Chevrolet Certified           34
Kia Certified                 32
INFINITI Certified            31
Volvo Certified               30
Porsche Certified             22
RAM Certified                 22
Buick Certified               19
Volkswagen Certified          17
GMC Certified                 13
Dodge Certified               10
Alfa Romeo Certified           6
MINI Certified                 1
Maserati Certified             1
Name: Used/New, dtype: int64
```

All of the data appears to be from used cars. Some are just dealer certified which makes this column not important to analysis.

In [35]:
```
make = cars_cleaned['Make'].value_counts()
print(make)
```

```
BMW              825
Mercedes-Benz    719
Toyota           705
Honda            630
Ford             524
Jeep             458
Lexus            440
Chevrolet        377
Audi             370
Subaru           262
Cadillac         239
Nissan           239
GMC              218
Kia              214
Acura            203
Hyundai          198
INFINITI         191
Mazda            176
Tesla            166
Land             143
RAM              135
Dodge            132
Volkswagen       128
Volvo            127
Lincoln          110
Porsche          105
Buick            104
Alfa              35
Chrysler          32
Jaguar            28
Mitsubishi        19
Genesis           17
Maserati          14
MINI               5
Scion              4
Lamborghini        3
Bentley            2
Mercury            2
FIAT               1
Saturn             1
Name: Make, dtype: int64
```

All makes appear to be unique.

In [36]:
```
gas_type = cars_cleaned['FuelType'].value_counts()
print(gas_type)
```

```
Gasoline                        7886
Electric                         140
E85 Flex Fuel                    108
Hybrid                            61
Diesel                            40
–                                 29
Gasoline Fuel                     23
Gasoline/Mild Electric Hybrid      5
Electric Fuel System               4
Flex Fuel Capability               3
Flexible Fuel                      2
Name: FuelType, dtype: int64
```

The most prominent type of fuel is gas, however fuel type can affect pricing of vehicles.

In [37]:
```
#  Step 10 Replace Gasoline Fuel with Gasoline
cars_cleaned_copy = cars_cleaned.copy()

# Specify the column ("FuelType" in this case) and the value to replace
old_value = 'Gasoline Fuel'
new_value = 'Gasoline'

# Replace the old value with the new value in the specified column of the copy
cars_cleaned_copy['FuelType'] = cars_cleaned_copy['FuelType'].str.replace(old_value, new_value)
# Check to see if it worked
gas_type2 = cars_cleaned_copy['FuelType'].value_counts()
print(gas_type2)
```

```
Gasoline                        7909
Electric                         140
E85 Flex Fuel                    108
Hybrid                            61
Diesel                            40
–                                 29
Gasoline/Mild Electric Hybrid      5
Electric Fuel System               4
Flex Fuel Capability               3
Flexible Fuel                      2
Name: FuelType, dtype: int64
```

Gasoline and gasoline fuel appeared to be the same thing. I combined them both under the name gasoline.

In [38]:
```
# Step 11 Select for gasoline cars
gasoline_cars = cars_cleaned_copy[cars_cleaned_copy['FuelType'] == 'Gasoline']
gas_type3 = gasoline_cars['FuelType'].value_counts()
print(gas_type3)
```

```
Gasoline    7909
Name: FuelType, dtype: int64
```

I selected for gasoline fuel type to keep analysis consistent due to different fuel types usually have different price points.

In [39]:
```
# Step 12 Remove unnessacary columns
# List of columns to remove
columns_to_remove = ['Model', 'SellerType', 'Used/New', 'SellerName','ExteriorColor', 'InteriorColor',
                     'Transmission', 'Engine', 'VIN', 'Stock#', 'StreetName', 'Drivetrain', 'FuelType']

# Remove the specified columns
gas_cars_cleaned = gasoline_cars.drop(columns=columns_to_remove, axis=1)
gas_cars_cleaned.head()
```

Out[39]:

	Year	Make	Price	ConsumerRating	ConsumerReviews	SellerRating	SellerReviews	State	Zipcode	DealType	ComfortRating
0	2019	Toyota	$39,998	4.6	45	3.3	3	CA	92562	Great	4.7
1	2018	Ford	$49,985	4.8	817	4.8	131	CA	93292	Good	4.9
2	2017	RAM	$41,860	4.7	495	4.6	249	CA	93637	Good	4.8
4	2020	Lexus	$49,000	4.8	76	4.8	4755	NV	89011	Good	4.9
5	2012	Toyota	$23,541	4.7	34	4.4	1071	CA	94544	Fair	4.7

There were columns that were not aiding in anaylsis and were able to be removed. FuelType was removed since the data was selected to only contain gasoline.

In [40]:
```
# Step 13 Identify ratings are between 1 and 5
# Columns to check
columns_to_check = ['ConsumerRating', 'SellerRating',
       'ComfortRating', 'InteriorDesignRating', 'PerformanceRating',
       'ValueForMoneyRating', 'ExteriorStylingRating', 'ReliabilityRating',]

gas_cars_cleaned[columns_to_check] = gas_cars_cleaned[columns_to_check].fillna(-1)

# Check if ratings are outside the range
outside_range = (gas_cars_cleaned[columns_to_check] < 1.0) | (gas_cars_cleaned[columns_to_check] > 5.0)
```

```
# Print rows with values outside of the range
rows_outside_range = gas_cars_cleaned[outside_range]
print(rows_outside_range)
```

```
      Year Make Price  ConsumerRating  ConsumerReviews  SellerRating  \
0      NaN  NaN   NaN             NaN              NaN           NaN
1      NaN  NaN   NaN             NaN              NaN           NaN
2      NaN  NaN   NaN             NaN              NaN           NaN
4      NaN  NaN   NaN             NaN              NaN           NaN
5      NaN  NaN   NaN             NaN              NaN           NaN
...    ...  ...   ...             ...              ...           ...
9373   NaN  NaN   NaN             NaN              NaN           NaN
9374   NaN  NaN   NaN             NaN              NaN           NaN
9376   NaN  NaN   NaN             NaN              NaN           NaN
9377   NaN  NaN   NaN             NaN              NaN           NaN
9378   NaN  NaN   NaN             NaN              NaN           NaN

      SellerReviews State Zipcode DealType  ComfortRating  \
0               NaN   NaN     NaN      NaN            NaN
1               NaN   NaN     NaN      NaN            NaN
2               NaN   NaN     NaN      NaN            NaN
4               NaN   NaN     NaN      NaN            NaN
5               NaN   NaN     NaN      NaN            NaN
...             ...   ...     ...      ...            ...
9373            NaN   NaN     NaN      NaN            NaN
9374            NaN   NaN     NaN      NaN            NaN
9376            NaN   NaN     NaN      NaN            NaN
9377            NaN   NaN     NaN      NaN            NaN
9378            NaN   NaN     NaN      NaN            NaN

      InteriorDesignRating  PerformanceRating  ValueForMoneyRating  \
0                      NaN                NaN                  NaN
1                      NaN                NaN                  NaN
2                      NaN                NaN                  NaN
4                      NaN                NaN                  NaN
5                      NaN                NaN                  NaN
...                    ...                ...                  ...
9373                   NaN                NaN                  NaN
9374                   NaN                NaN                  NaN
9376                   NaN                NaN                  NaN
9377                   NaN                NaN                  NaN
9378                   NaN                NaN                  NaN

      ExteriorStylingRating  ReliabilityRating  MinMPG  MaxMPG  Mileage
0                       NaN                NaN     NaN     NaN      NaN
1                       NaN                NaN     NaN     NaN      NaN
2                       NaN                NaN     NaN     NaN      NaN
4                       NaN                NaN     NaN     NaN      NaN
5                       NaN                NaN     NaN     NaN      NaN
```

```
...                      ...                  ...     ...     ...     ...
9373                     NaN                  NaN     NaN     NaN     NaN
9374                     NaN                  NaN     NaN     NaN     NaN
9376                     NaN                  NaN     NaN     NaN     NaN
9377                     NaN                  NaN     NaN     NaN     NaN
9378                     NaN                  NaN     NaN     NaN     NaN

[7909 rows x 19 columns]
```

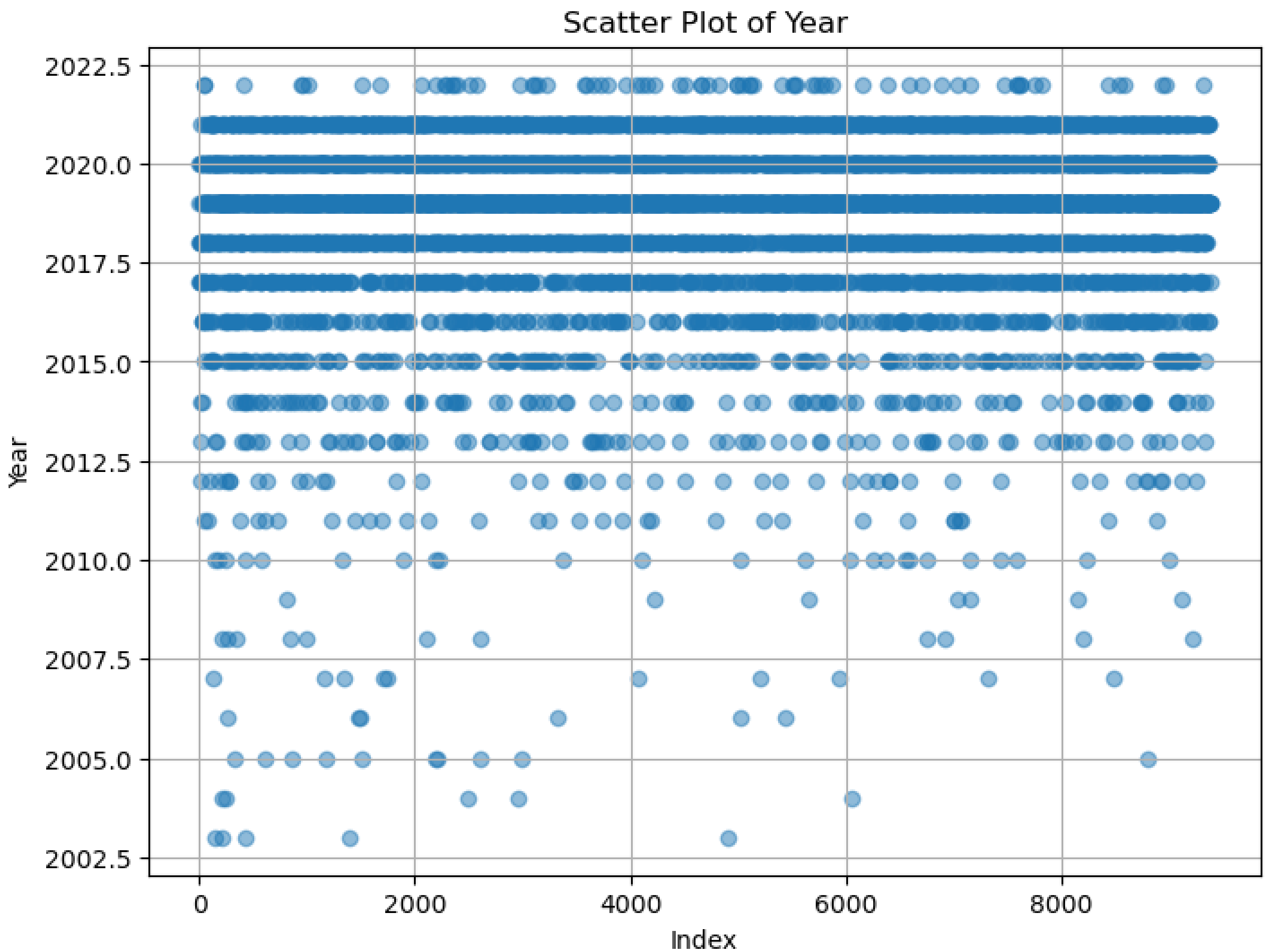
I wanted to check that all ratings were within the apparent 1 to 5 range. It appears that all values for the selected columns fall within this range.

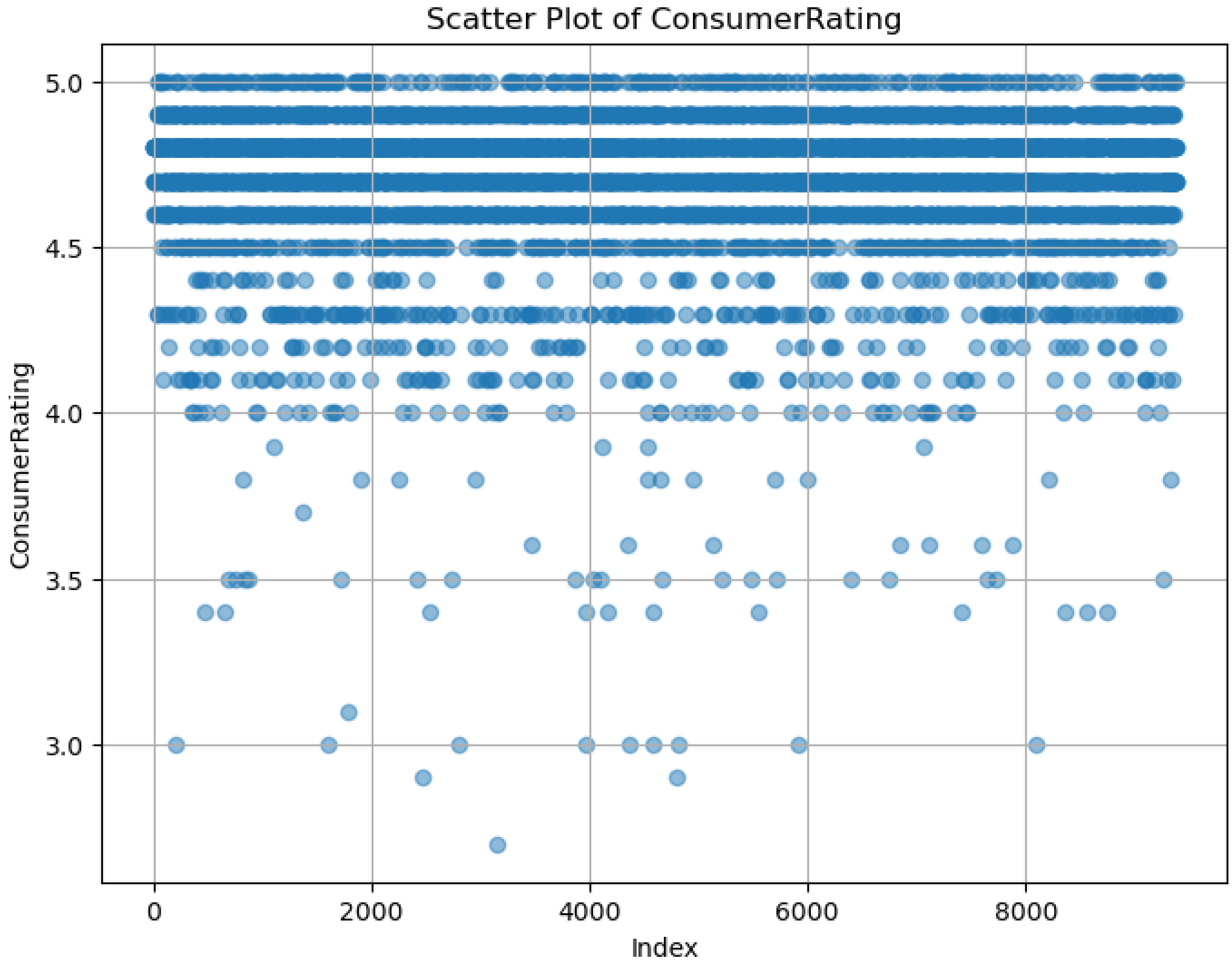
In [41]:
```
# Step 14 Identify Outliers
numeric_columns = gas_cars_cleaned.select_dtypes(include=[np.number]).columns

outliers = pd.DataFrame()

# Threshold for Z-scores for identifying outliers
threshold = 3

# Loop and identify outliers
for column_name in numeric_columns:
    z_scores = np.abs(stats.zscore(gas_cars_cleaned[column_name]))
    column_outliers = gas_cars_cleaned[z_scores > threshold]
    outliers = pd.concat([outliers, column_outliers])
outliers.shape
```

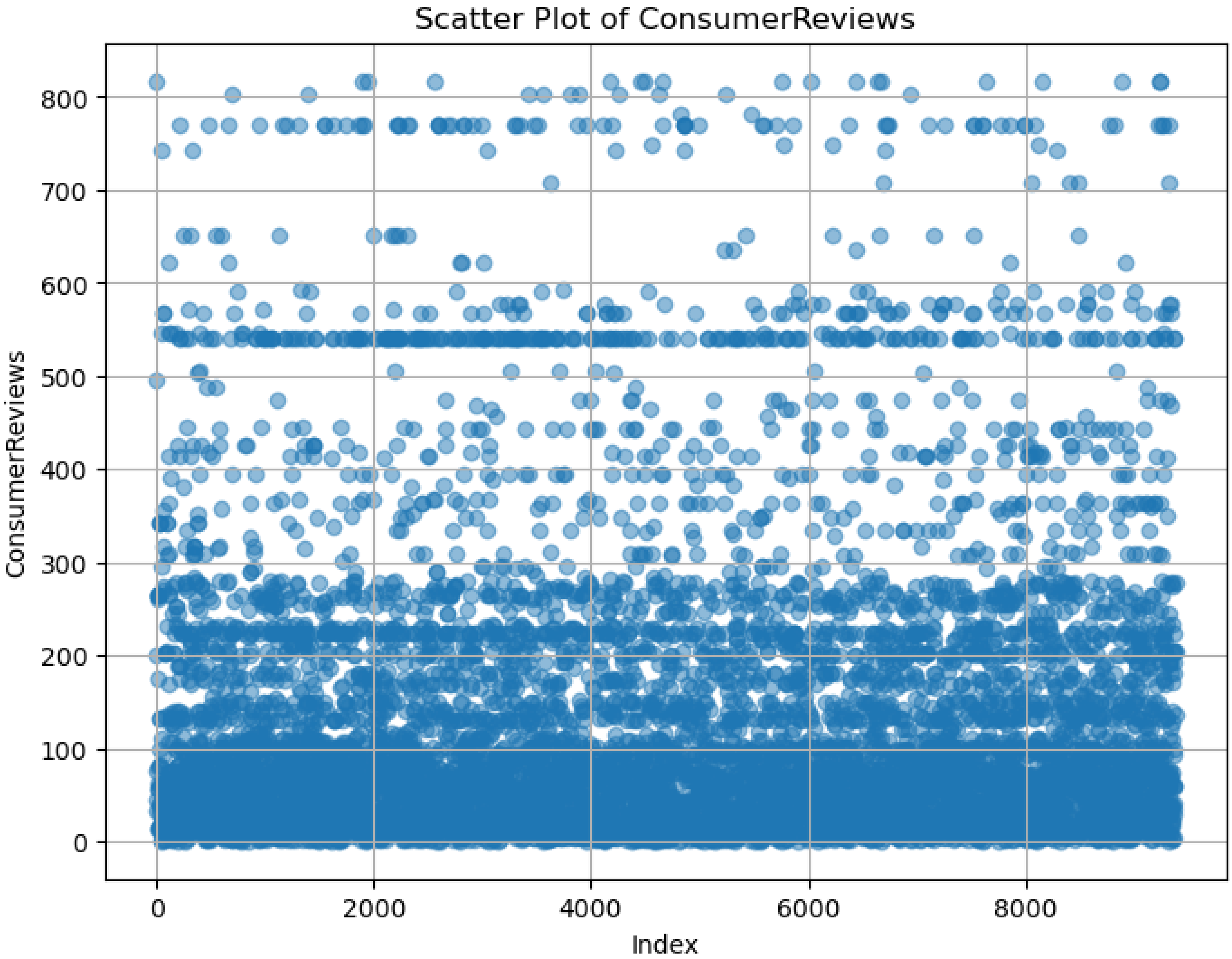
Out[41]:
```
(1806, 19)
```

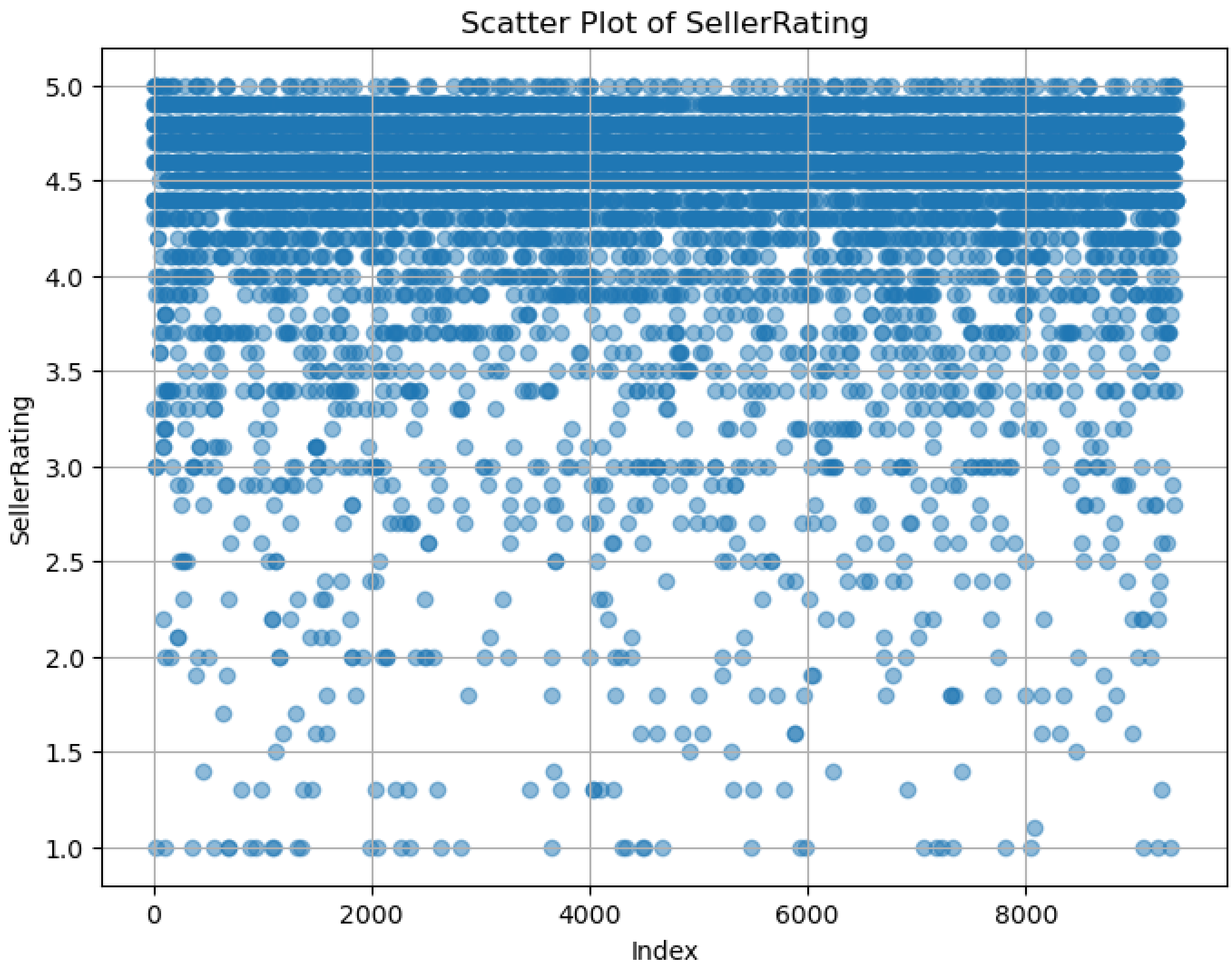
Checking for outliers is necessary to make sure data is not being skewed.

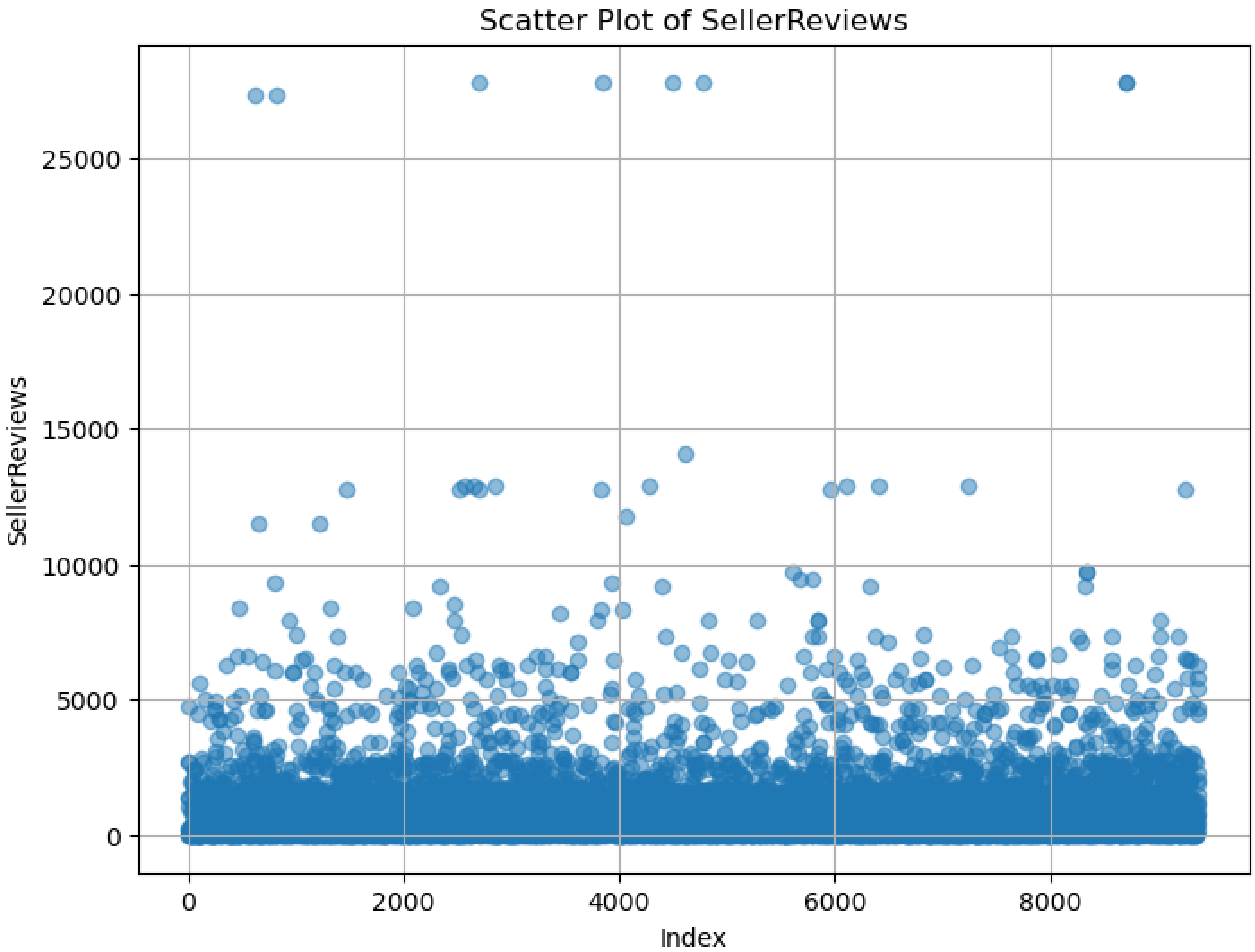
In [42]:
```
# Step 15 Visualize Outliers
import matplotlib.pyplot as plt
numerical_columns = gas_cars_cleaned.select_dtypes(include=['number']).columns
# Create scatter plots for each numerical column
for column in numerical_columns:
    plt.figure(figsize=(8, 6))
    plt.scatter(gas_cars_cleaned.index, gas_cars_cleaned[column], alpha=0.5)
    plt.title(f'Scatter Plot of {column}')
    plt.xlabel('Index')
    plt.ylabel(column)
    plt.grid(True)
    plt.show()
```

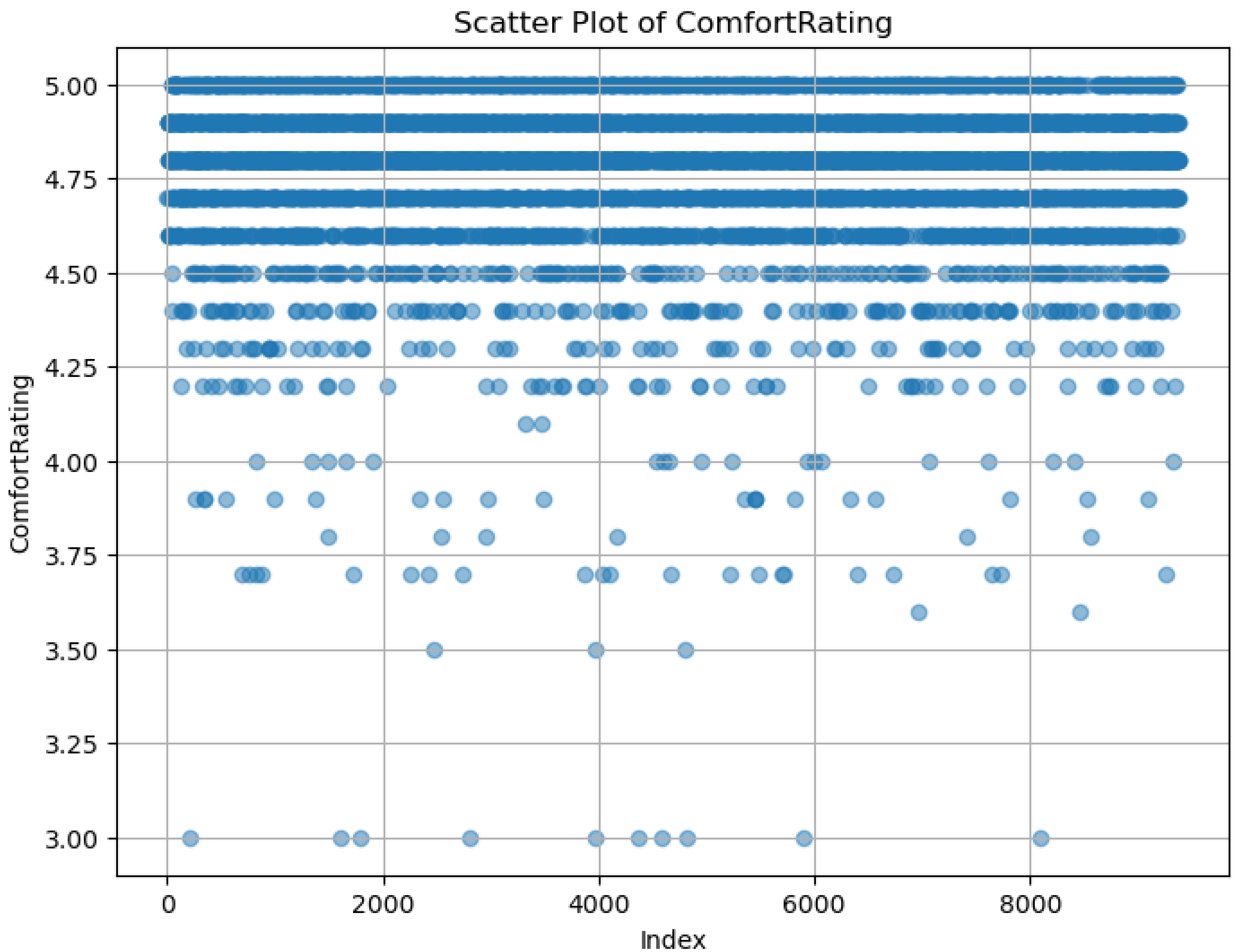

Scatter Plot of Year
Year
2022.5
2020.0
2017.5
2015.0
2012.5
2010.0
2007.5
2005.0
2002.5
0
2000
4000
6000
8000
Index

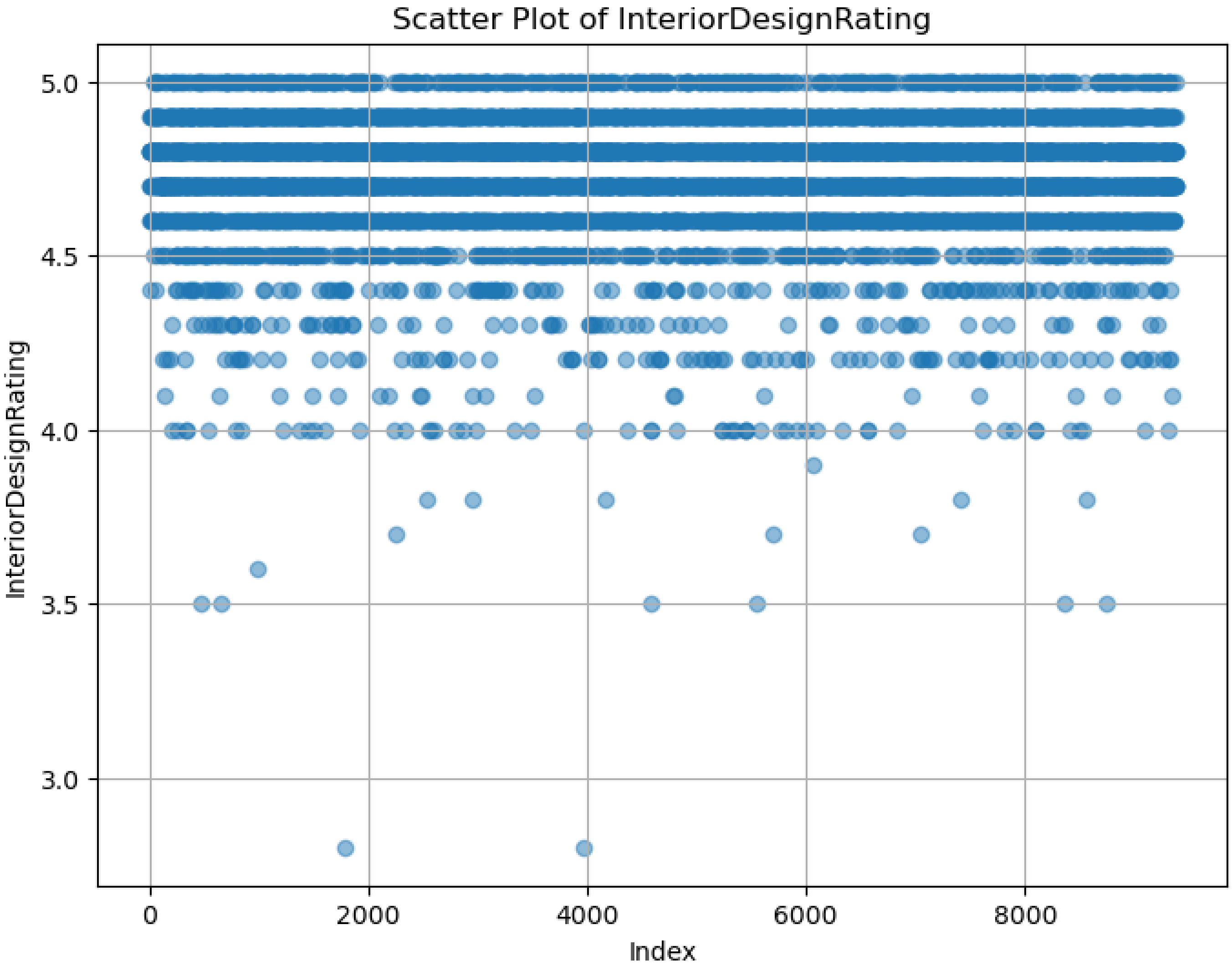

Scatter Plot of ConsumerRating
ConsumerRating
Index
5.0
4.5
4.0
3.5
3.0
0
2000
4000
6000
8000

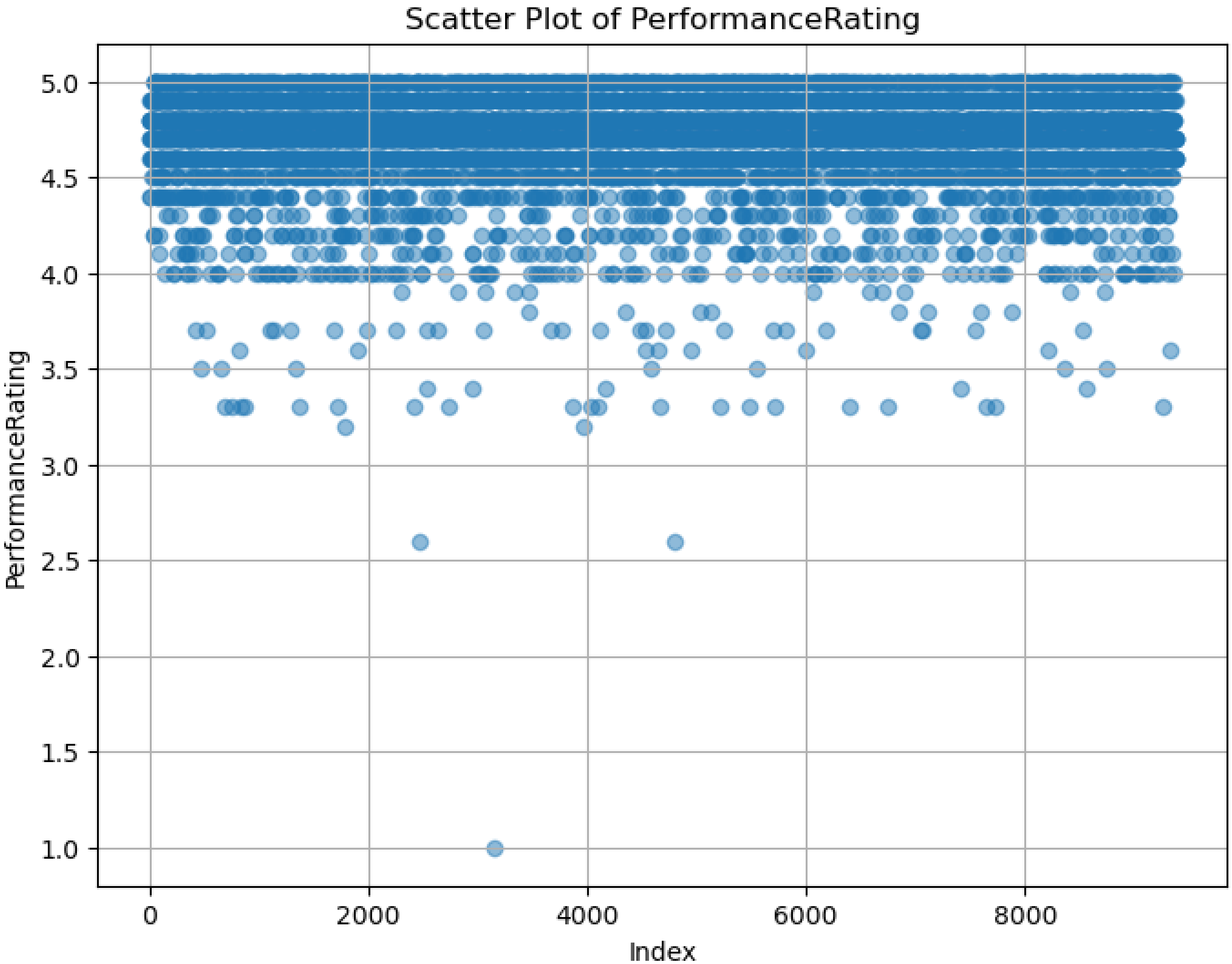

Scatter Plot of ConsumerReviews
ConsumerReviews
Index
0
100
200
300
400
500
600
700
800
0
2000
4000
6000
8000

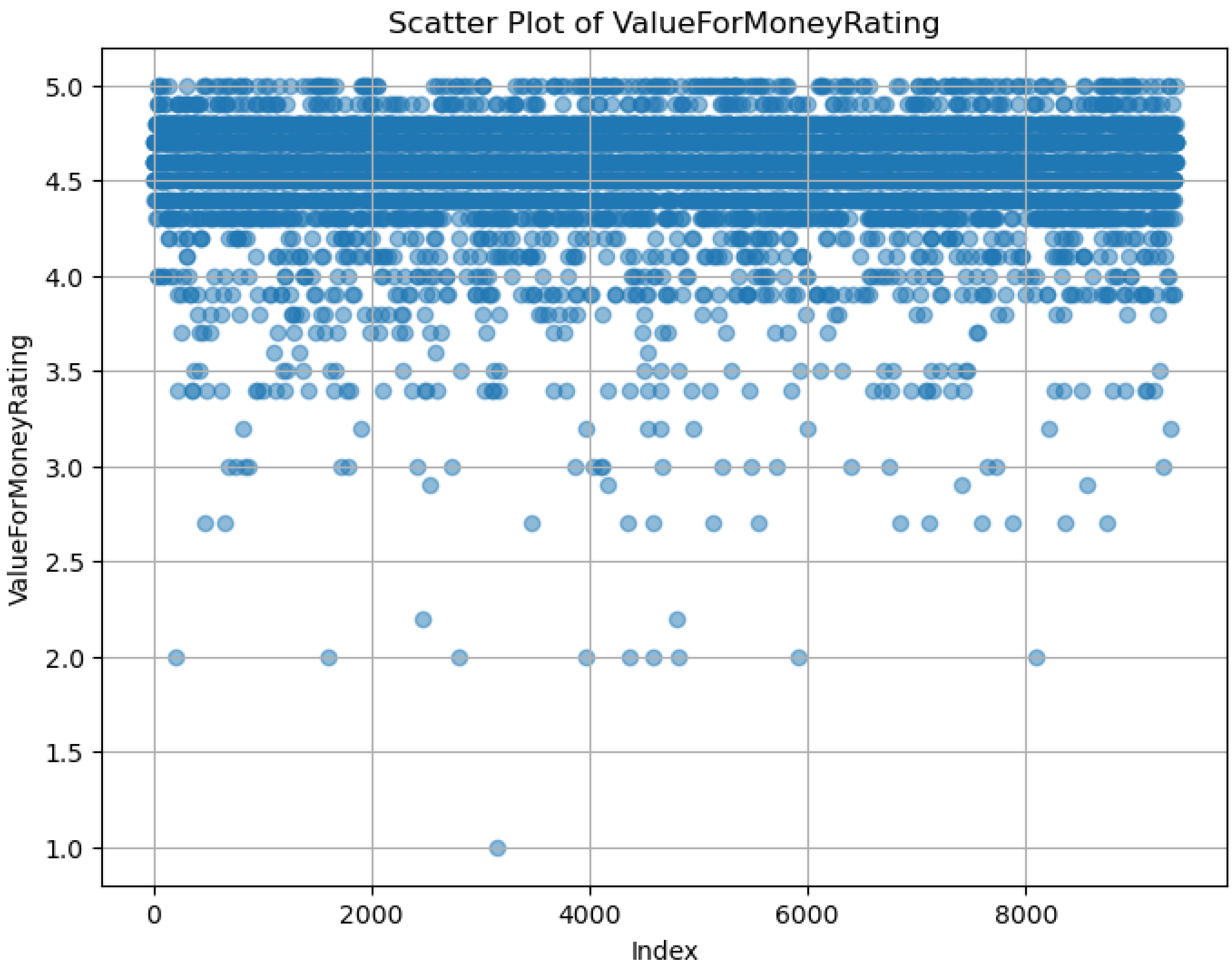

Scatter Plot of SellerRating
SellerRating
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0
2000
4000
6000
8000
Index

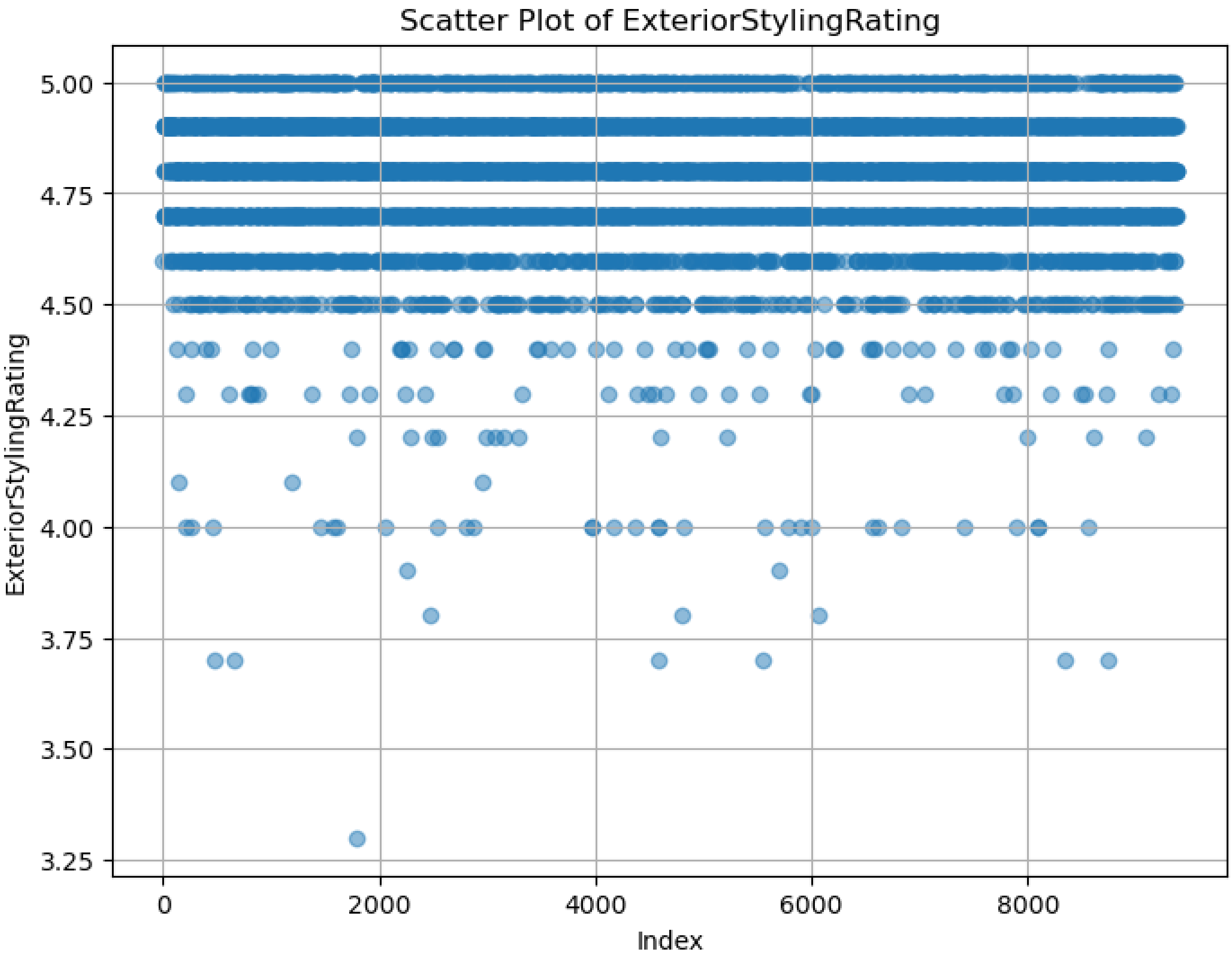

Scatter Plot of SellerReviews
SellerReviews
25000
20000
15000
10000
5000
0
0
2000
4000
6000
8000
Index

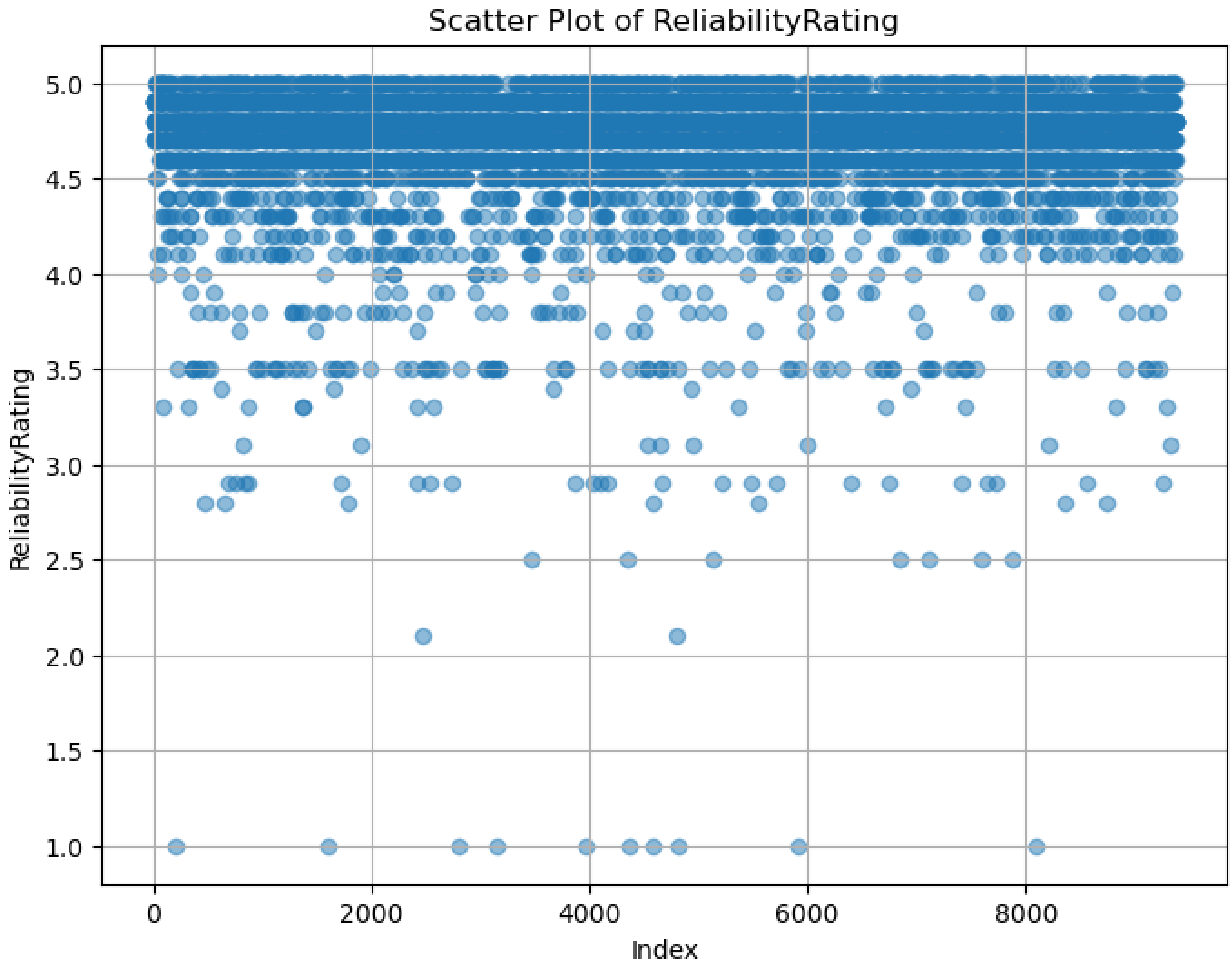

Scatter Plot of ComfortRating
ComfortRating
5.00
4.75
4.50
4.25
4.00
3.75
3.50
3.25
3.00
0
2000
4000
6000
8000
Index

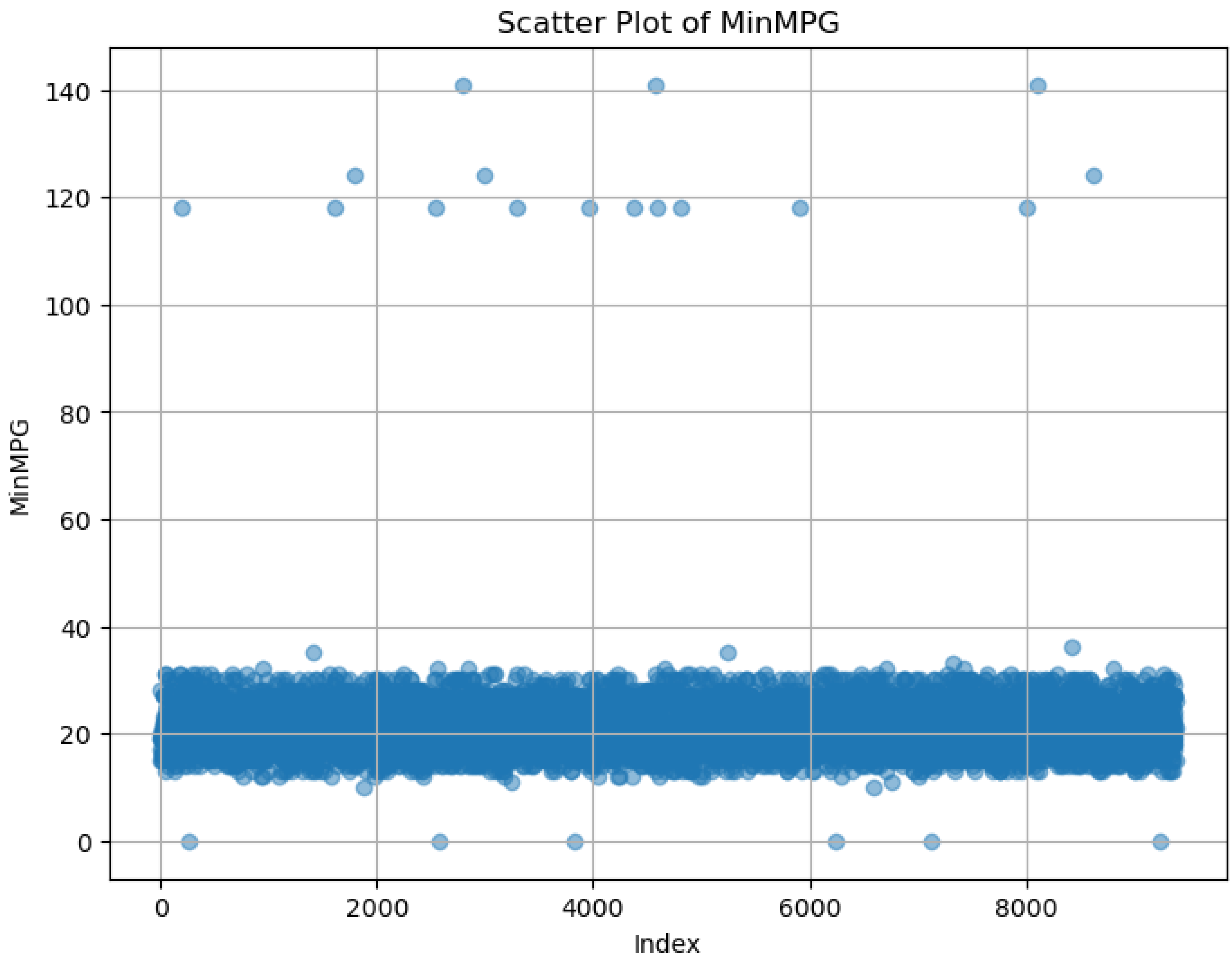

Scatter Plot of InteriorDesignRating
InteriorDesignRating
5.0
4.5
4.0
3.5
3.0
0
2000
4000
6000
8000
Index

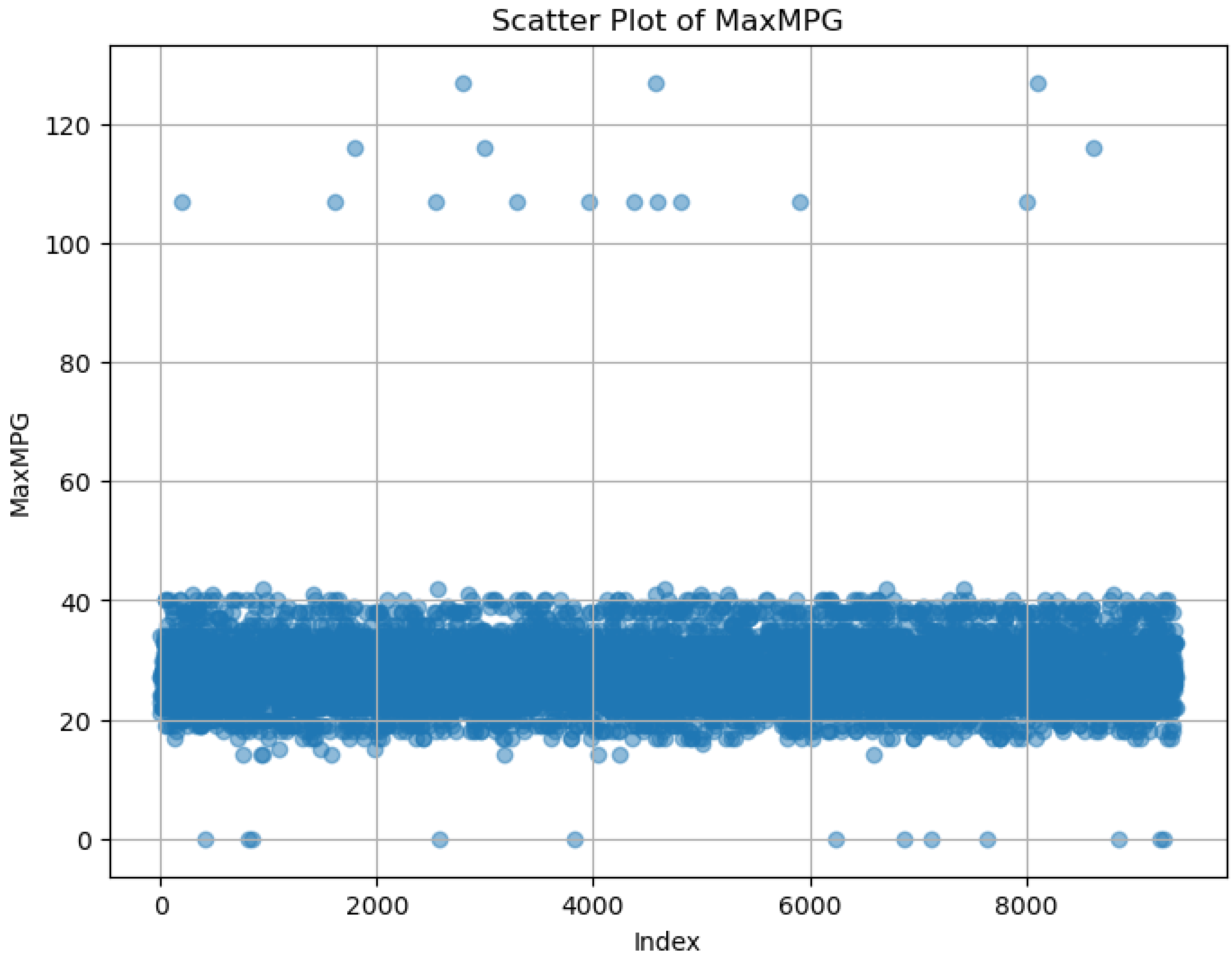

Scatter Plot of PerformanceRating
PerformanceRating
Index
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0
2000
4000
6000
8000

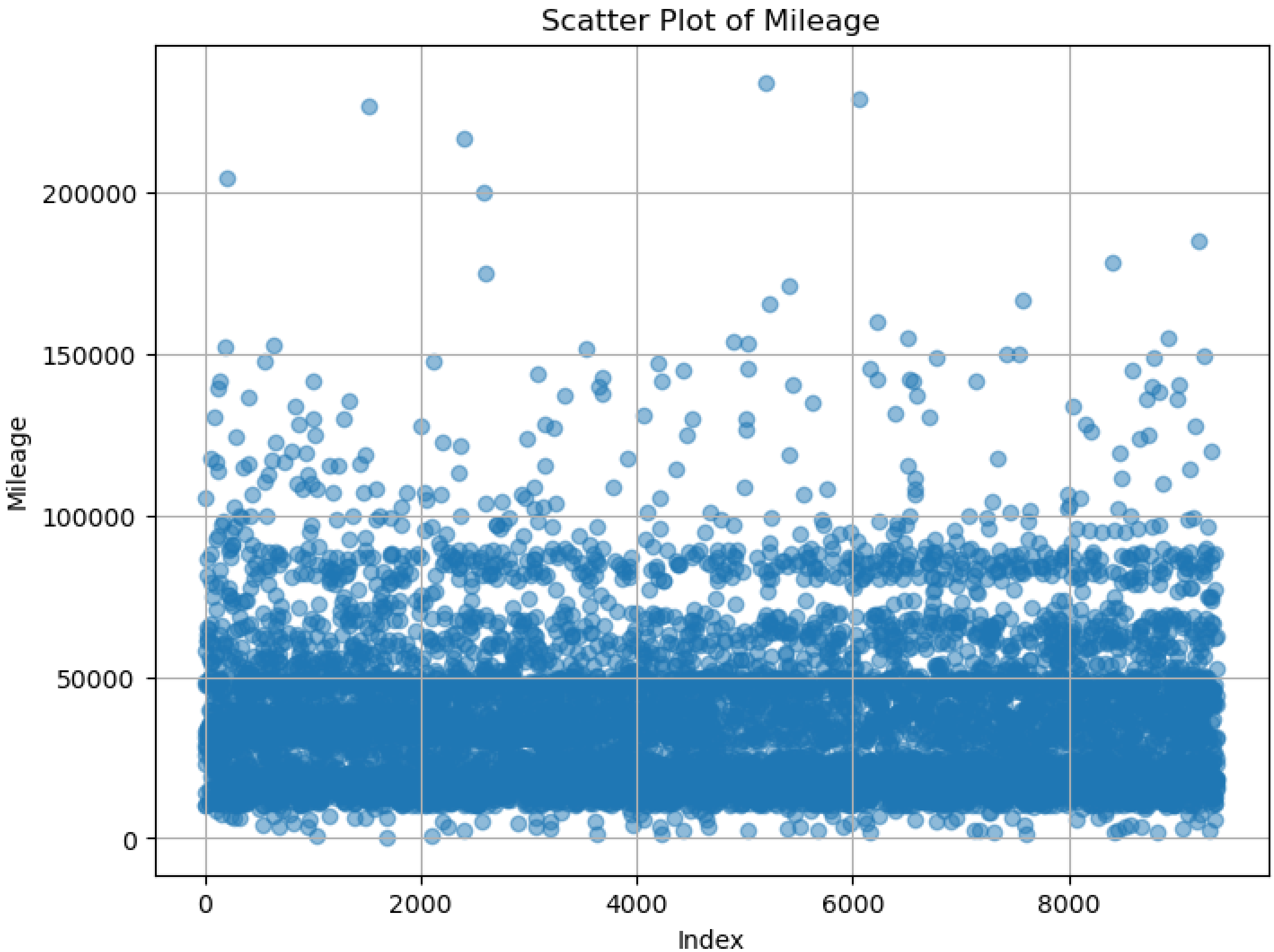

Scatter Plot of ValueForMoneyRating
ValueForMoneyRating
Index
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0
2000
4000
6000
8000


Scatter Plot of ExteriorStylingRating
ExteriorStylingRating
Index
5.00
4.75
4.50
4.25
4.00
3.75
3.50
3.25
0
2000
4000
6000
8000


Scatter Plot of ReliabilityRating
ReliabilityRating
Index
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0
2000
4000
6000
8000


Scatter Plot of MinMPG
MinMPG
Index
0
20
40
60
80
100
120
140
0
2000
4000
6000
8000


Scatter Plot of MaxMPG
MaxMPG
0
20
40
60
80
100
120
0
2000
4000
6000
8000
Index



Visualizing numerical data helps to determine outliers and allows you to look at them visually. I believe that the outliers are valid data points and do not plan to remove them at this point.

In [43]:
```
# Step 16: Fix Column Names
gas_cars_cleaned.rename(columns={'ConsumerRating': 'Consumer Rating', 'ConsumerReviews': 'Consumer Reviews',
                                'SellerRating': 'Seller Rating', 'SellerReviews':'Seller Reviews',
                                'DealType': 'Deal Type', 'ComfortRating': 'Comfort Rating', 'InteriorDesignRa
                                'Interior Design Rating', 'PerformanceRating': 'Performance Rating',
                                'ValueForMoneyRating': 'Value For Money Rating', 'ExteriorStylingRating'
```

```
                                    : 'Exterior Styling Rating', 'ReliabilityRating': 'Reliability Rating',
                                     'MinMPG': 'Minimum MPG', 'MaxMPG': 'Maximum MPG'}, inplace=True)
```

In [44]:
```
gas_cars_cleaned.head()
```

Out[44]:

	Year	Make	Price	Consumer Rating	Consumer Reviews	Seller Rating	Seller Reviews	State	Zipcode	Deal Type	Comfort Rating	Interior Design Rating	Performance Rating	Value For Money Rating	Ex S F
0	2019	Toyota	$39,998	4.6	45	3.3	3	CA	92562	Great	4.7	4.6	4.6	4.4	
1	2018	Ford	$49,985	4.8	817	4.8	131	CA	93292	Good	4.9	4.8	4.8	4.6	
2	2017	RAM	$41,860	4.7	495	4.6	249	CA	93637	Good	4.8	4.7	4.8	4.6	
4	2020	Lexus	$49,000	4.8	76	4.8	4755	NV	89011	Good	4.9	4.8	4.8	4.7	
5	2012	Toyota	$23,541	4.7	34	4.4	1071	CA	94544	Fair	4.7	4.6	4.4	4.6	

I fixed the column names to allow for easier user readability.

```
Changing data sets can lead to innacurate representations during analysis. Removing the
columns and making assumptions such as that gasoline fuel and gasoline are the same can also
lead to skewing data. This is due to not considering all the facts for the data point.
Assuming that the outliers were accuate data points can lead to an innacurate skew of the
data and misrepresentation. I did not adujust price based on model of car (Mercedes is
typical more expensive than Toyota) which can lead to a bias and/or misrepresentation due to
the higher cost of the cars. There are more data points for some years than others which
will skew results. It is important to consider all the ways the data could be skewed or
biased.
```

Website Source Milestone 3

In [48]:
```
import requests
from bs4 import BeautifulSoup
import pandas as pd

# Scrape data
url = "https://www.currentresults.com/Weather/US/average-annual-state-temperatures.php#google_vignette"
response = requests.get(url)
```

```
soup = BeautifulSoup(response.text, "html.parser")

# Empty list to store data
all_data = []

# Iterate through each table and scrape data
for table in soup.find_all("table"):
    data = []
    for row in table.find_all("tr"):
        cols = row.find_all("td")
        cols = [col.text.strip() for col in cols]
        data.append(cols)
    all_data.extend(data)
average_temp = pd.DataFrame(all_data)

print(average_temp)
```

	0	1	2	3
0	None	None	None	None
1	Alabama	62.8	17.1	7
2	Alaska	26.6	-3.0	50
3	Arizona	60.3	15.7	10
4	Arkansas	60.4	15.8	9
5	California	59.4	15.2	12
6	Colorado	45.1	7.3	39
7	Connecticut	49.0	9.4	29
8	Delaware	55.3	12.9	16
9	Florida	70.7	21.5	1
10	Georgia	63.5	17.5	5
11	Hawaii	70.0	21.1	2
12	Idaho	44.4	6.9	40
13	Illinois	51.8	11.0	23
14	Indiana	51.7	10.9	25
15	Iowa	47.8	8.8	36
16	Kansas	54.3	12.4	19
17	Kentucky	55.6	13.1	15
18	None	None	None	None
19	Louisiana	66.4	19.1	3
20	Maine	41.0	5.0	48
21	Maryland	54.2	12.3	20
22	Massachusetts	47.9	8.8	35
23	Michigan	44.4	6.9	40
24	Minnesota	41.2	5.1	47
25	Mississippi	63.4	17.4	6
26	Missouri	54.5	12.5	18
27	Montana	42.7	5.9	45
28	Nebraska	48.8	9.3	30
29	Nevada	49.9	9.9	28
30	New Hampshire	43.8	6.6	42
31	New Jersey	52.7	11.5	22
32	New Mexico	53.4	11.9	21
33	New York	45.4	7.4	37
34	North Carolina	59.0	15.0	13
35	North Dakota	40.4	4.7	49
36	None	None	None	None
37	Ohio	50.7	10.4	26
38	Oklahoma	59.6	15.3	11
39	Oregon	48.4	9.1	33
40	Pennsylvania	48.8	9.3	30
41	Rhode Island	50.1	10.1	27
42	South Carolina	62.4	16.9	8
43	South Dakota	45.2	7.3	38

```
44       Tennessee  57.6  14.2    14
45           Texas  64.8  18.2     4
46            Utah  48.6   9.2    32
47         Vermont  42.9   6.1    44
48        Virginia  55.1  12.8    17
49      Washington  48.3   9.1    34
50   West Virginia  51.8  11.0    23
51       Wisconsin  43.1   6.2    43
52         Wyoming  42.0   5.6    46
```

The webscrapping obtained data from the three tables containing state weather data.

In [49]:
```
# Check Shape
average_temp.shape
```

Out[49]:
```
(53, 4)
```

In [50]:
```
# Step 1: Drop the none (NA) values
average_temp = average_temp.dropna()
average_temp.head()
```

Out[50]:

	0	1	2	3
1	Alabama	62.8	17.1	7
2	Alaska	26.6	-3.0	50
3	Arizona	60.3	15.7	10
4	Arkansas	60.4	15.8	9
5	California	59.4	15.2	12

There were apparent rows with the value None. These had to be removed in order to have accurcy and prevent issues down the road.

In [51]:
```
# Step 2: Add column names for clarity
new_column_names = ['State', 'Average Temperature (F)', 'Average Temperature (C)', 'Rank']
average_temp.columns = new_column_names
average_temp.head()
```

Out[51]:

	State	Average Temperature (F)	Average Temperature (C)	Rank
1	Alabama	62.8	17.1	7
2	Alaska	26.6	-3.0	50
3	Arizona	60.3	15.7	10
4	Arkansas	60.4	15.8	9
5	California	59.4	15.2	12

Define column names to improve clarifity and allow for clear understanding of data frame and purpose.

In [52]:
```
# Step 3: Link state names to abbreviations
```

In [78]:
```
# Dictionary of state abbreviations
state_abbreviations = {
    "Alabama": "AL",
    "Alaska": "AK",
    "Arizona": "AZ",
    "Arkansas": "AR",
    "California": "CA",
    "Colorado": "CO",
    "Connecticut": "CT",
    "Delaware": "DE",
    "Florida": "FL",
    "Georgia": "GA",
    "Hawaii": "HI",
    "Idaho": "ID",
    "Illinois": "IL",
    "Indiana": "IN",
    "Iowa": "IA",
    "Kansas": "KS",
    "Kentucky": "KY",
    "Louisiana": "LA",
    "Maine": "ME",
    "Maryland": "MD",
    "Massachusetts": "MA",
    "Michigan": "MI",
    "Minnesota": "MN",
    "Mississippi": "MS",
    "Missouri": "MO",
    "Montana": "MT",
    "Nebraska": "NE",
```

```
    "Nevada": "NV",
    "New Hampshire": "NH",
    "New Jersey": "NJ",
    "New Mexico": "NM",
    "New York": "NY",
    "North Carolina": "NC",
    "North Dakota": "ND",
    "Ohio": "OH",
    "Oklahoma": "OK",
    "Oregon": "OR",
    "Pennsylvania": "PA",
    "Rhode Island": "RI",
    "South Carolina": "SC",
    "South Dakota": "SD",
    "Tennessee": "TN",
    "Texas": "TX",
    "Utah": "UT",
    "Vermont": "VT",
    "Virginia": "VA",
    "Washington": "WA",
    "West Virginia": "WV",
    "Wisconsin": "WI",
    "Wyoming": "WY"
}
```

Create dictionary of states and abrevations to allow for easy interchanging between both. This was done due to Fuzzy matching incorrecting matching states and abreviations.

In [79]:
```
# Step 4: Make the words uppercase for easier matching
states_dict = {key.upper(): value.upper() for key, value in state_abbreviations.items()}
states_dict
```

Out[79]:
```
{'ALABAMA': 'AL',
 'ALASKA': 'AK',
 'ARIZONA': 'AZ',
 'ARKANSAS': 'AR',
 'CALIFORNIA': 'CA',
 'COLORADO': 'CO',
 'CONNECTICUT': 'CT',
 'DELAWARE': 'DE',
 'FLORIDA': 'FL',
 'GEORGIA': 'GA',
 'HAWAII': 'HI',
 'IDAHO': 'ID',
 'ILLINOIS': 'IL',
 'INDIANA': 'IN',
 'IOWA': 'IA',
 'KANSAS': 'KS',
 'KENTUCKY': 'KY',
 'LOUISIANA': 'LA',
 'MAINE': 'ME',
 'MARYLAND': 'MD',
 'MASSACHUSETTS': 'MA',
 'MICHIGAN': 'MI',
 'MINNESOTA': 'MN',
 'MISSISSIPPI': 'MS',
 'MISSOURI': 'MO',
 'MONTANA': 'MT',
 'NEBRASKA': 'NE',
 'NEVADA': 'NV',
 'NEW HAMPSHIRE': 'NH',
 'NEW JERSEY': 'NJ',
 'NEW MEXICO': 'NM',
 'NEW YORK': 'NY',
 'NORTH CAROLINA': 'NC',
 'NORTH DAKOTA': 'ND',
 'OHIO': 'OH',
 'OKLAHOMA': 'OK',
 'OREGON': 'OR',
 'PENNSYLVANIA': 'PA',
 'RHODE ISLAND': 'RI',
 'SOUTH CAROLINA': 'SC',
 'SOUTH DAKOTA': 'SD',
 'TENNESSEE': 'TN',
 'TEXAS': 'TX',
 'UTAH': 'UT',
 'VERMONT': 'VT',
```

```
 'VIRGINIA': 'VA',
 'WASHINGTON': 'WA',
 'WEST VIRGINIA': 'WV',
 'WISCONSIN': 'WI',
 'WYOMING': 'WY'}
```

Changed to all caps to insure proper matching.

In [55]:
```
column_types = average_temp.dtypes
print(column_types)
```

```
State                        object
Average Temperature (F)      object
Average Temperature (C)      object
Rank                         object
dtype: object
```

In [56]:
```
# Step 5: Covert column types
columns_to_convert = ['Average Temperature (F)', 'Average Temperature (C)', 'Rank']
for column in columns_to_convert:
    average_temp[column] = pd.to_numeric(average_temp[column], errors='coerce')
```

In [57]:
```
average_temp['State'] = average_temp['State'].astype(str)
```

In [58]:
```
column_types = average_temp.dtypes
print(column_types)
```

```
State                         object
Average Temperature (F)      float64
Average Temperature (C)      float64
Rank                           int64
dtype: object
```

Change columns to their correct types to allow for proper analysis and prevent issues later.

In [59]:
```
# Step 6: Check for outliers
z_scores = np.abs(stats.zscore(average_temp['Average Temperature (F)']))
threshold = 3 # I'm going to use 3 deviations
outlier_indices = np.where(z_scores > threshold)
for index in outlier_indices:
    print(average_temp.iloc[index])
```

```
Empty DataFrame
Columns: [State, Average Temperature (F), Average Temperature (C), Rank]
Index: []
```

In [60]:

```
z_scores = np.abs(stats.zscore(average_temp['Average Temperature (F)']))
threshold = 2 # I'm going to use 2 deviations
outlier_indices = np.where(z_scores > threshold)
for index in outlier_indices:
    print(average_temp.iloc[index])
```

```
      State  Average Temperature (F)  Average Temperature (C)  Rank
2    Alaska                     26.6                     -3.0    50
9   Florida                     70.7                     21.5     1
11   Hawaii                     70.0                     21.1     2
```

Checking for outliers can help to increase accuarcy of analysis. In this case, outliers appear valid and will be used.

Milestone 4 API

In [61]:

```
# Step 1: Obtain data from api to begin analysis.
import requests

api_key = "76137d4840f08913000b9384494b9b202a7997bf"

# API for State Data
api_url = f"https://api.census.gov/data/2021/pep/population?get=DENSITY_2021,POP_2021,NAME,STATE&for=state:*&

# Make the API request
response = requests.get(api_url)

# Check if the request was successful (status code 200)
if response.status_code == 200:
    data = response.json()

    # Print the first 5 rows
    for row in data[:5]:
        print(row)
else:
    print("Error: Unable to fetch data from the API.")
```

```
['DENSITY_2021', 'POP_2021', 'NAME', 'STATE', 'state']
['58.1171593930', '3986639', 'Oklahoma', '40', '40']
['25.5629643700', '1963692', 'Nebraska', '31', '31']
['224.4561379100', '1441553', 'Hawaii', '15', '15']
['11.8108489860', '895376', 'South Dakota', '46', '46']
```

In [62]:
```
# Step 2: Create the data frame to allow for cleaning.
state_pop = pd.DataFrame(data[1:], columns=data[0])
state_pop.head()
```

Out[62]:

	DENSITY_2021	POP_2021	NAME	STATE	state
0	58.1171593930	3986639	Oklahoma	40	40
1	25.5629643700	1963692	Nebraska	31	31
2	224.4561379100	1441553	Hawaii	15	15
3	11.8108489860	895376	South Dakota	46	46
4	169.1679021400	6975218	Tennessee	47	47

In [63]:
```
# Step 3: Remove uneeded columns for more efficent code.
columns_to_remove = ['state', 'STATE']
state_pop_clean = state_pop.drop(columns=columns_to_remove)
state_pop_clean.head()
```

Out[63]:

	DENSITY_2021	POP_2021	NAME
0	58.1171593930	3986639	Oklahoma
1	25.5629643700	1963692	Nebraska
2	224.4561379100	1441553	Hawaii
3	11.8108489860	895376	South Dakota
4	169.1679021400	6975218	Tennessee

In [64]:
```
# Step 4: Change column names to be more readable.
state_popul = state_pop_clean.rename(columns={
    'DENSITY_2021': 'Density',
    'POP_2021': 'Population',
    'NAME': 'State'
})
```

```
state_popul.head()
```

Out[64]:

	Density	Population	State
0	58.1171593930	3986639	Oklahoma
1	25.5629643700	1963692	Nebraska
2	224.4561379100	1441553	Hawaii
3	11.8108489860	895376	South Dakota
4	169.1679021400	6975218	Tennessee

In [65]:
```
# Step 5: Change order of columns to help increase readability and understanding.
state_population = state_popul[['State', 'Population', 'Density']]
state_population.head()
```

Out[65]:

	State	Population	Density
0	Oklahoma	3986639	58.1171593930
1	Nebraska	1963692	25.5629643700
2	Hawaii	1441553	224.4561379100
3	South Dakota	895376	11.8108489860
4	Tennessee	6975218	169.1679021400

In [66]:
```
# Step 6: Sort the values by state name
pop_state = state_population.sort_values(by='State')
pop_state.head()
```

Out[66]:

	State	Population	Density
48	Alabama	5039877	99.5099129150
51	Alaska	732673	1.2830925496
45	Arizona	7276316	64.0221138030
12	Arkansas	3025891	58.1946786180
19	California	39237836	251.7543622300

In [67]:
```
# Step 7: Round density values to two decimal places to increase readability.
pop_state['Density'] = pop_state['Density'].astype(float).round(2)
pop_state.head()
```

Out[67]:

	State	Population	Density
48	Alabama	5039877	99.51
51	Alaska	732673	1.28
45	Arizona	7276316	64.02
12	Arkansas	3025891	58.19
19	California	39237836	251.75

In [68]:
```
# Step 8: Make all letter uppercase for matching and to prevent errors.
pop_state['State'] = pop_state['State'].str.upper()
pop_state.head()
```

Out[68]:

	State	Population	Density
48	ALABAMA	5039877	99.51
51	ALASKA	732673	1.28
45	ARIZONA	7276316	64.02
12	ARKANSAS	3025891	58.19
19	CALIFORNIA	39237836	251.75

In [69]:
```
# Step 9: Drop states with NA (Puerto Rico)
pop_states = pop_state.dropna()
print(pop_states)
```

	State	Population	Density
48	ALABAMA	5039877	99.51
51	ALASKA	732673	1.28
45	ARIZONA	7276316	64.02
12	ARKANSAS	3025891	58.19
19	CALIFORNIA	39237836	251.75
28	COLORADO	5812069	56.08
39	CONNECTICUT	3605597	744.56
23	DELAWARE	1003384	514.94
9	DISTRICT OF COLUMBIA	670050	10961.85
40	FLORIDA	21781128	405.98
25	GEORGIA	10799566	187.11
2	HAWAII	1441553	224.46
34	IDAHO	1900923	23.00
43	ILLINOIS	12671469	228.26
35	INDIANA	6805985	189.97
7	IOWA	3193079	57.17
8	KANSAS	2934582	35.89
30	KENTUCKY	4509394	114.19
21	LOUISIANA	4624047	107.01
32	MAINE	1372247	44.49
22	MARYLAND	6165129	634.85
42	MASSACHUSETTS	6984723	895.37
13	MICHIGAN	10050811	177.55
27	MINNESOTA	5707390	71.68
44	MISSISSIPPI	2949965	62.87
11	MISSOURI	6168187	89.72
36	MONTANA	1104271	7.59
1	NEBRASKA	1963692	25.56
5	NEVADA	3143991	28.62
14	NEW HAMPSHIRE	1388992	155.13
29	NEW JERSEY	9267130	1260.03
6	NEW MEXICO	2115877	17.44
37	NEW YORK	19835913	420.94
15	NORTH CAROLINA	10551162	217.00
20	NORTH DAKOTA	774948	11.23
16	OHIO	11780017	288.31
0	OKLAHOMA	3986639	58.12
26	OREGON	4246155	44.23
24	PENNSYLVANIA	12964056	289.75
50	RHODE ISLAND	1095610	1059.70
17	SOUTH CAROLINA	5190705	172.65
3	SOUTH DAKOTA	895376	11.81
4	TENNESSEE	6975218	169.17
10	TEXAS	29527941	113.02

```
46            UTAH    3337975     40.52
33         VERMONT     645570     70.04
41        VIRGINIA    8642274    218.89
31      WASHINGTON    7738692    116.45
49   WEST VIRGINIA    1782959     74.16
47       WISCONSIN    5895908    108.85
18         WYOMING     578803      5.96
```

Data was imported from the api and placed in the dataframe to allow for cleaning. The first step in cleaning was to remove the unneeded state columns as they were unneeded and provide no value to analysis. The column names were changed to improve readability and anlysis. The order of the columns was then changed to improve readibilty and understanding of the table. The values were then sorted by state name. This is consistent with previous tables. Denisity was rounded to two decimal places. There was a significant amount of extra decimal places that were not necissary for analysis. I capitalized all letes to improve matching later down the road. Lastly, I dropped Puerto Rico as it is not needed for analysis and contained NA values which could have caused issues for anlysis later on.

As always, changing data comes with ethical emplications. Removing Puerto Rico from my data set removes a state with a smaller population and can cause imbalance. However, since we are not considering state population seperately but as a means to scale data this is not a concern. Another possible delima is the rounding of the density values. This can cause innacurate representation due to rounding. However, I rounded to two decimal places so this should not be a concern. Lastly, I changed column names and removed columns. Removing columns could be seen as removing important data, however the data removed was state numbers which is not relevant to the analysis. The changed column names also were not changed to anything misrepresentive of the data. They were changed to improve readability abd were consistent with previous column names.

Project Milestone 5

In [94]:
```
# Make all of the states uppercase abbrevations
average_temp['State'] = average_temp['State'].map(state_abbreviations)
pop_states['State'] = pop_states['State'].map(states_dict)
```

In [95]:
```
# Check df
gas_cars_cleaned.head()
```

Out[95]:

	Year	Make	Price	Consumer Rating	Consumer Reviews	Seller Rating	Seller Reviews	State	Zipcode	Deal Type	Comfort Rating	Interior Design Rating	Performance Rating	Value For Money Rating	Ex S F
0	2019	Toyota	$39,998	4.6	45	3.3	3	CA	92562	Great	4.7	4.6	4.6	4.4	
1	2018	Ford	$49,985	4.8	817	4.8	131	CA	93292	Good	4.9	4.8	4.8	4.6	
2	2017	RAM	$41,860	4.7	495	4.6	249	CA	93637	Good	4.8	4.7	4.8	4.6	
4	2020	Lexus	$49,000	4.8	76	4.8	4755	NV	89011	Good	4.9	4.8	4.8	4.7	
5	2012	Toyota	$23,541	4.7	34	4.4	1071	CA	94544	Fair	4.7	4.6	4.4	4.6	

In [114…

```
# Check df
average_temp.head()
```

Out[114]:

	State	Average Temperature (F)	Average Temperature (C)	Rank
1	AL	62.8	17.1	7.0
2	AK	26.6	-3.0	50.0
3	AZ	60.3	15.7	10.0
4	AR	60.4	15.8	9.0
5	CA	59.4	15.2	12.0

In [115…

```
# Check df
pop_states.head()
```

Out[115]:

	State	Population	Density
48	AL	5039877	99.51
51	AK	732673	1.28
45	AZ	7276316	64.02
12	AR	3025891	58.19
19	CA	39237836	251.75

In [116…

```
# import packages
import sqlite3
```

```
import pandas as pd
```

In [134…
```
# Connect to SQLite database
conn = sqlite3.connect('CarsMerged1.db')
```

In [135…
```
# Create Tables from the dataframes
gas_cars_cleaned.to_sql('gas_cars_cleaned', conn, index=False, if_exists='replace')
average_temp.to_sql('temperature', conn, index=False, if_exists='replace')
pop_states.to_sql('population', conn, index=False, if_exists='replace')
```

Out[135]: 52

In [136…
```
# Close the connections (I did this due to persistent errors it helped)
conn.close()
```

In [217…
```
# Connect to the database
conn = sqlite3.connect('CarsMerged1.db')
```

In [166…
```
# SQL query to combine the tables
# Had to specify exact columns to prevent duplicate state columns
query_combine_tables = """
SELECT
    gc.*,
    at.'Average Temperature (F)' AS 'Average Temperature (F)',
    at.'Average Temperature (C)' AS 'Average Temperature (C))',
    at.Rank,
    ps.Population,
    ps.Density
FROM
    gas_cars_cleaned gc
JOIN
    temperature at ON gc.State = at.State
JOIN
    population ps ON gc.State = ps.State;
"""
```

In [224…
```
# Read the joined result into a new DF
combined_data = pd.read_sql_query(query_combine_tables, conn)

# Check
print(combined_data)
```

```
      Year     Make    Price  Consumer Rating  Consumer Reviews  \
0     2019   Toyota  $39,998              4.6                45   
1     2018     Ford  $49,985              4.8               817   
2     2017      RAM  $41,860              4.7               495   
3     2020    Lexus  $49,000              4.8                76   
4     2012   Toyota  $23,541              4.7                34   
...    ...      ...      ...              ...               ...   
7884  2019    Honda  $31,999              4.8               540   
7885  2019   Subaru  $27,374              4.7               205   
7886  2017    Buick  $26,944              4.8               137   
7887  2019   Subaru  $28,568              4.7               279   
7888  2019  Hyundai  $32,091              4.8               204   

      Seller Rating  Seller Reviews State Zipcode Deal Type  ...  \
0               3.3               3    CA   92562     Great  ...   
1               4.8             131    CA   93292      Good  ...   
2               4.6             249    CA   93637      Good  ...   
3               4.8            4755    NV   89011      Good  ...   
4               4.4            1071    CA   94544      Fair  ...   
...             ...             ...   ...     ...       ...  ...   
7884            4.8            1917    CT   06092      Good  ...   
7885            4.4             443    MA   01089      Good  ...   
7886            4.7             831    NH   03060      Good  ...   
7887            4.4             680    MA   01923      Good  ...   
7888            4.4            1105    MA   02767      Good  ...   

      Exterior Styling Rating  Reliability Rating  Minimum MPG  Maximum MPG  \
0                         4.6                 4.7           19           27   
1                         4.8                 4.7           19           24   
2                         4.8                 4.7           15           21   
3                         4.8                 4.9           20           27   
4                         4.9                 4.9           17           23   
...                       ...                 ...          ...          ...   
7884                      4.8                 4.8           27           33   
7885                      4.8                 4.8           27           33   
7886                      4.9                 4.8           15           22   
7887                      4.7                 4.8           26           33   
7888                      4.9                 4.8           21           27   

      Mileage  Average Temperature (F)  Average Temperature (C))  Rank  \
0       29403                     59.4                      15.2  12.0   
1       32929                     59.4                      15.2  12.0   
2       23173                     59.4                      15.2  12.0   
3       28137                     49.9                       9.9  28.0   
4      105469                     59.4                      15.2  12.0   
```

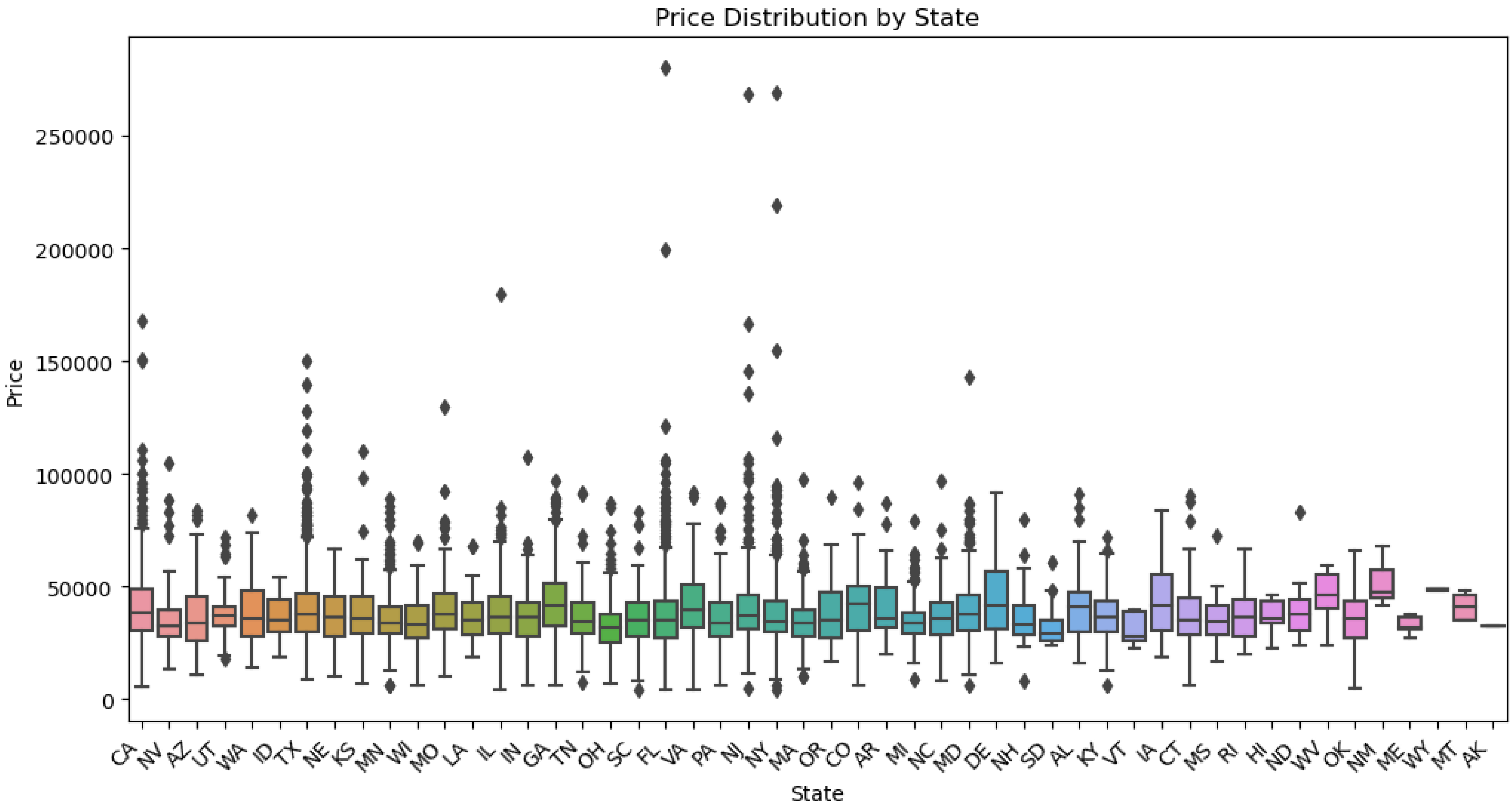
```
...     ...                  ...                      ...   ...
7884    44481                49.0                     9.4   29.0
7885    15606                47.9                     8.8   35.0
7886    62649                43.8                     6.6   42.0
7887    30760                47.9                     8.8   35.0
7888    41645                47.9                     8.8   35.0

      Population  Density
0       39237836   251.75
1       39237836   251.75
2       39237836   251.75
3        3143991    28.62
4       39237836   251.75
...          ...      ...
7884     3605597   744.56
7885     6984723   895.37
7886     1388992   155.13
7887     6984723   895.37
7888     6984723   895.37

[7889 rows x 24 columns]
```

In [251…
```
conn.close()
```

Create 5 Visulizations

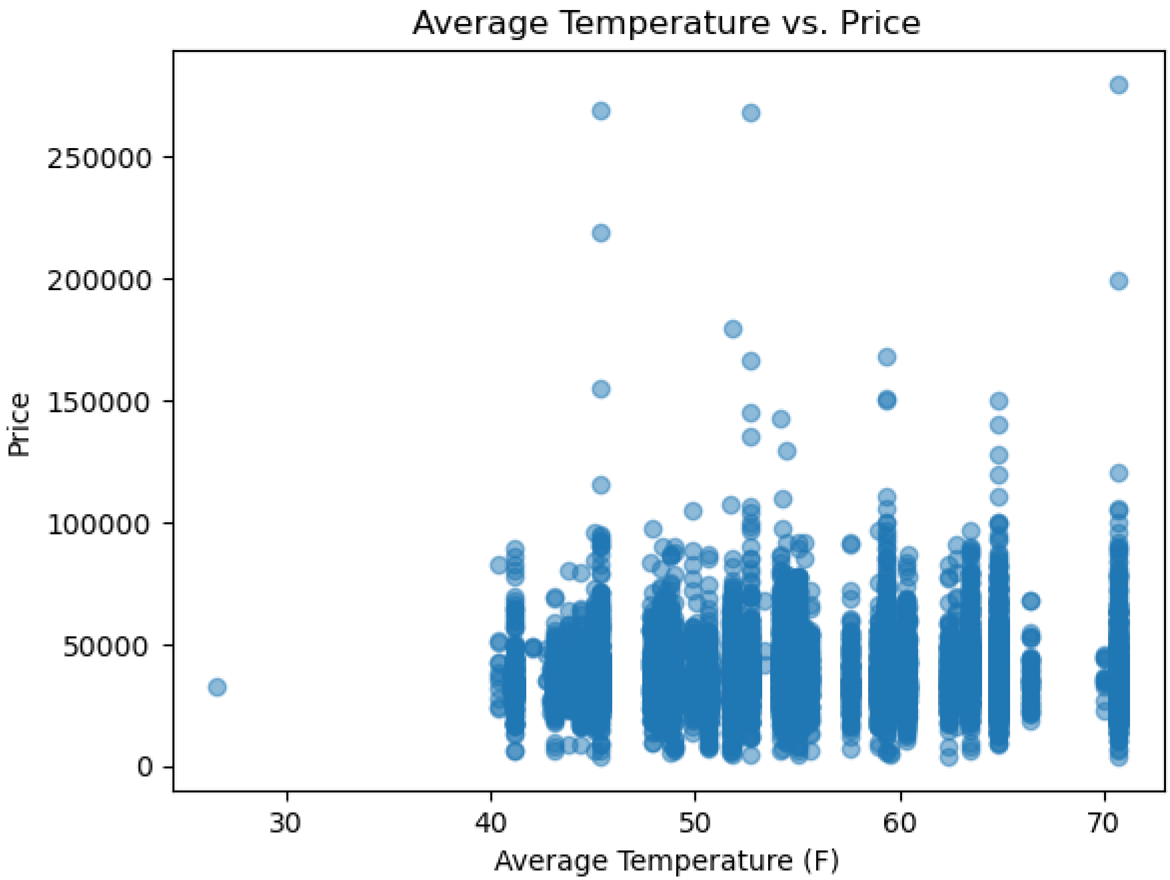
In [225…
```
# Fix issues with newly combined data
combined_data['Price'] = pd.to_numeric(combined_data['Price'].replace('[\$,]', '', regex=True),
                                       errors='coerce').astype(float)
combined_data['Population'] = pd.to_numeric(combined_data['Population'], errors='coerce')
combined_data['Average Temperature (F)'] = pd.to_numeric(combined_data['Average Temperature (F)'],
                                                         errors='coerce')
combined_data = combined_data.dropna()
```

In [226…
```
# Price by state
plt.figure(figsize=(12, 6))
sns.boxplot(x='State', y='Price', data=combined_data)
plt.title('Price Distribution by State')
plt.xlabel('State')
plt.ylabel('Price')
plt.xticks(rotation=45, ha='right')
plt.show()
```



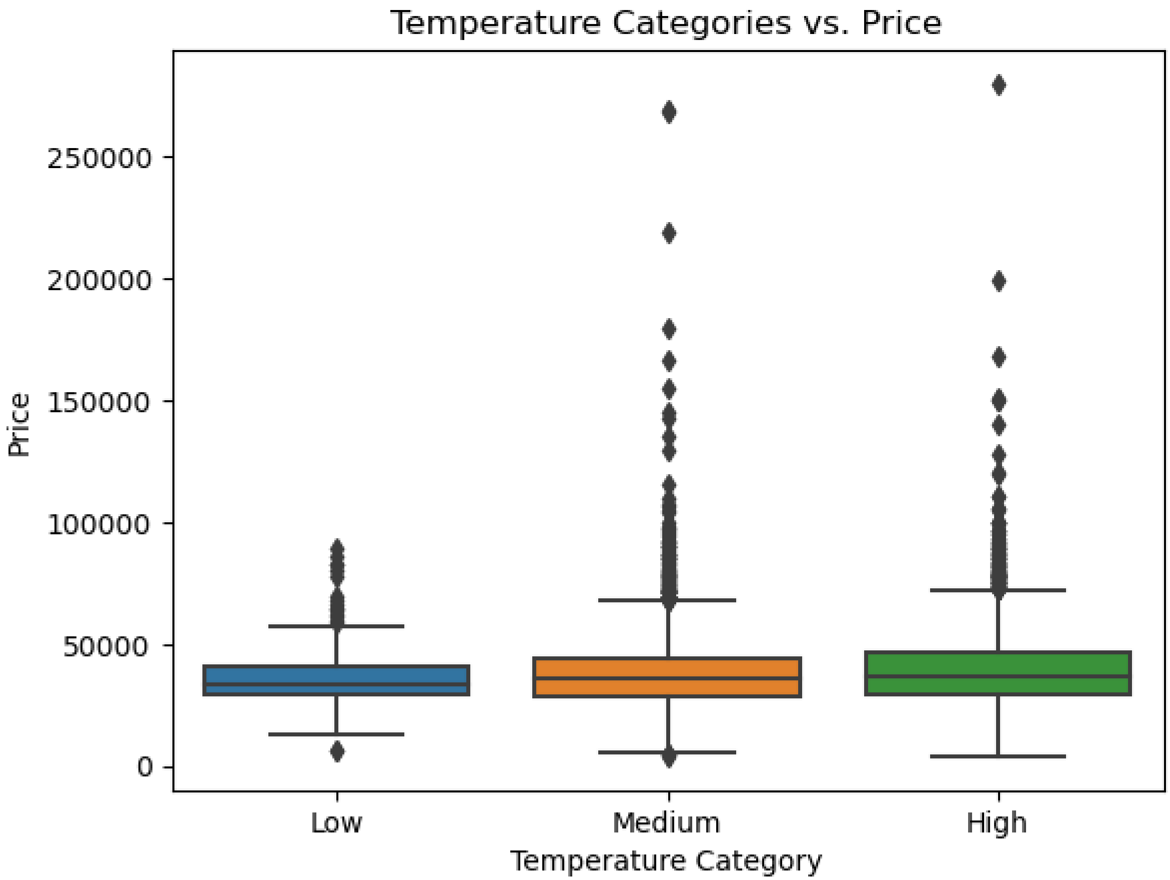
In [230…

```
# Average Temp vs Price
plt.scatter(combined_data['Average Temperature (F)'], combined_data['Price'], alpha=0.5)
plt.title('Average Temperature vs. Price')
plt.xlabel('Average Temperature (F)')
plt.ylabel('Price')
plt.show()
```

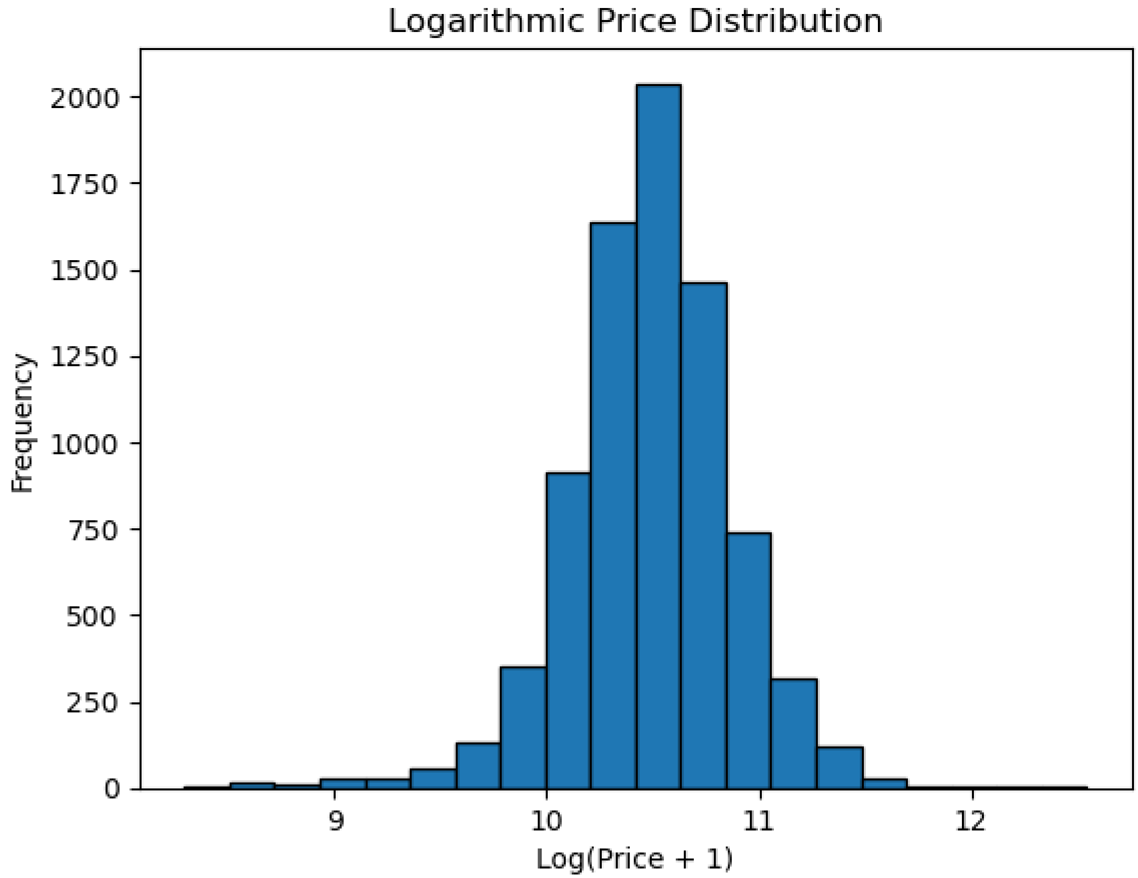


In [232…

```
# Temperature cateogires vs Price
combined_data['Temperature_Category'] = pd.cut(combined_data['Average Temperature (F)'],
                                               bins=3, labels=['Low', 'Medium', 'High'])
sns.boxplot(x='Temperature_Category', y='Price', data=combined_data)
plt.title('Temperature Categories vs. Price')
plt.xlabel('Temperature Category')
plt.ylabel('Price')
plt.show()
```

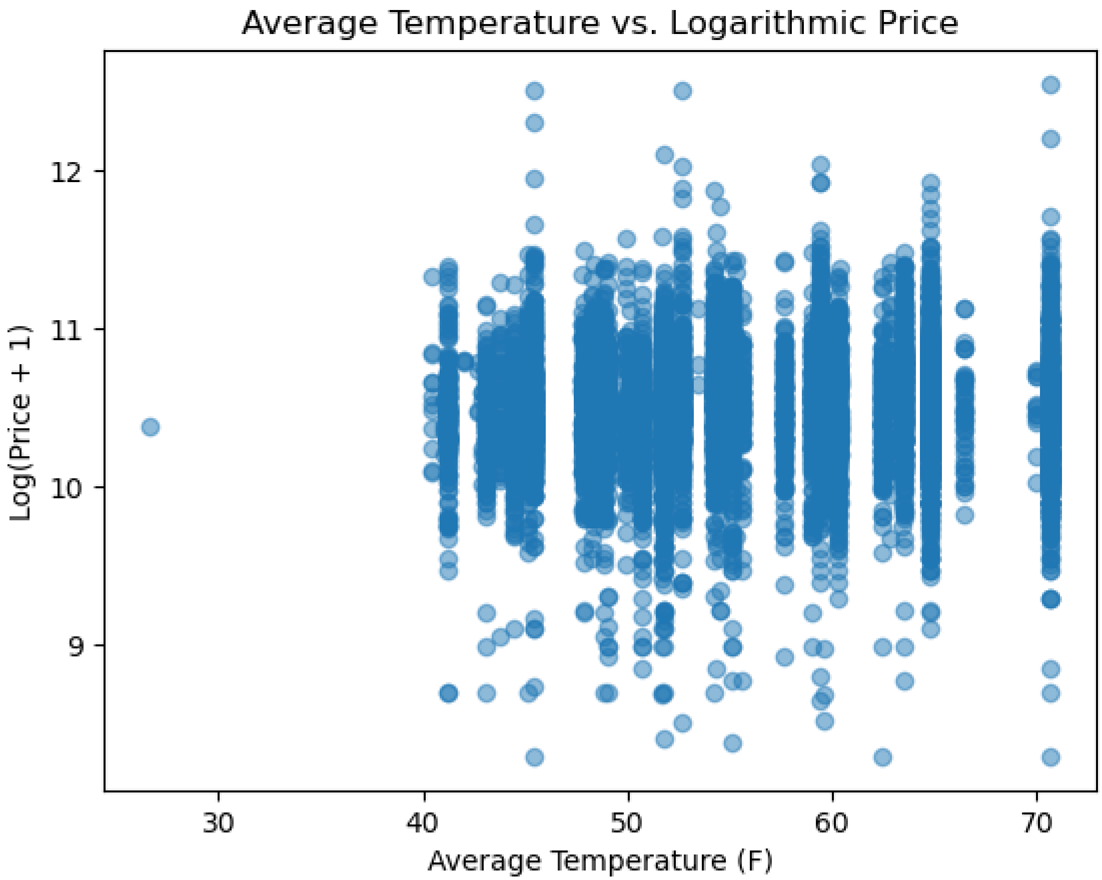


In [228…
```
# Distribution of Price(log)
plt.hist(np.log1p(combined_data['Price']), bins=20, edgecolor='black')
plt.title('Logarithmic Price Distribution')
plt.xlabel('Log(Price + 1)')
plt.ylabel('Frequency')
plt.show()
```



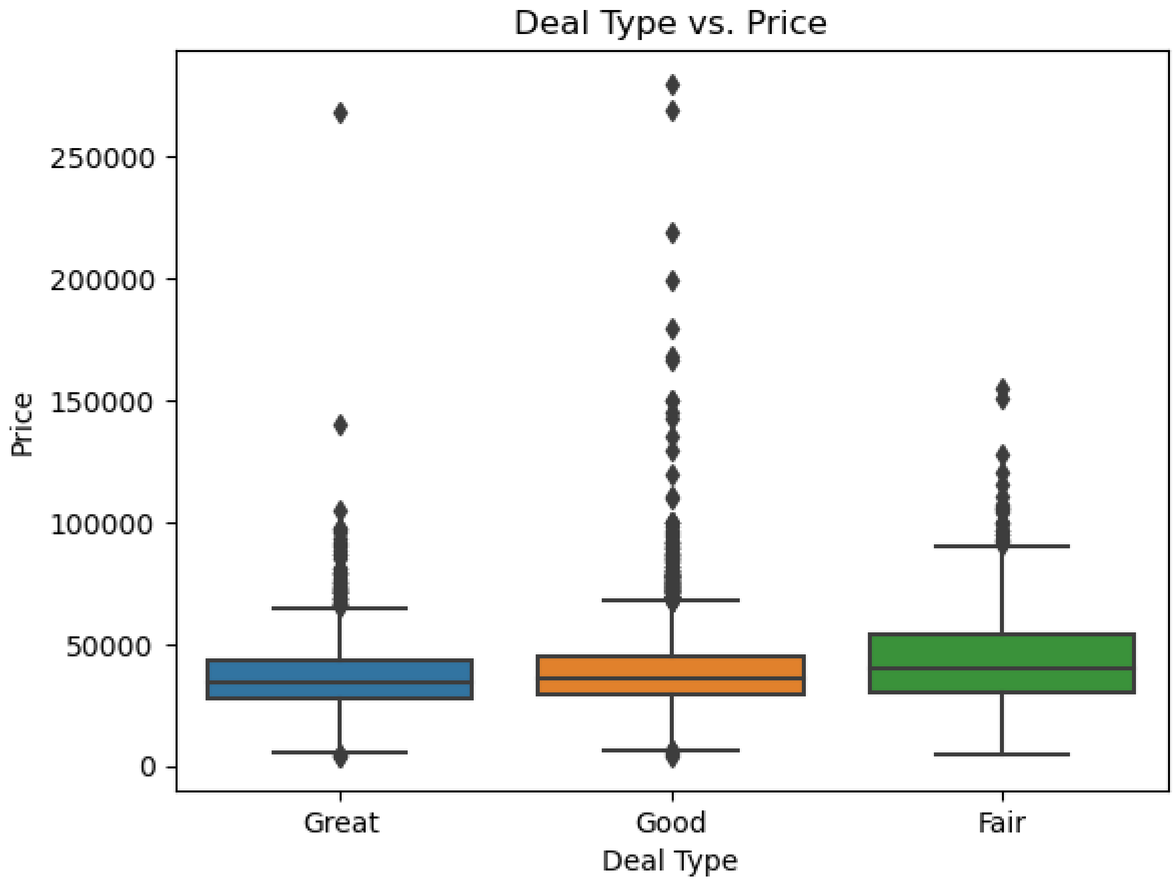
In [227…

```
# Temp vs Price (log)
plt.scatter(combined_data['Average Temperature (F)'], np.log1p(combined_data['Price']), alpha=0.5)
plt.title('Average Temperature vs. Logarithmic Price')
plt.xlabel('Average Temperature (F)')
plt.ylabel('Log(Price + 1)')
plt.show()
```



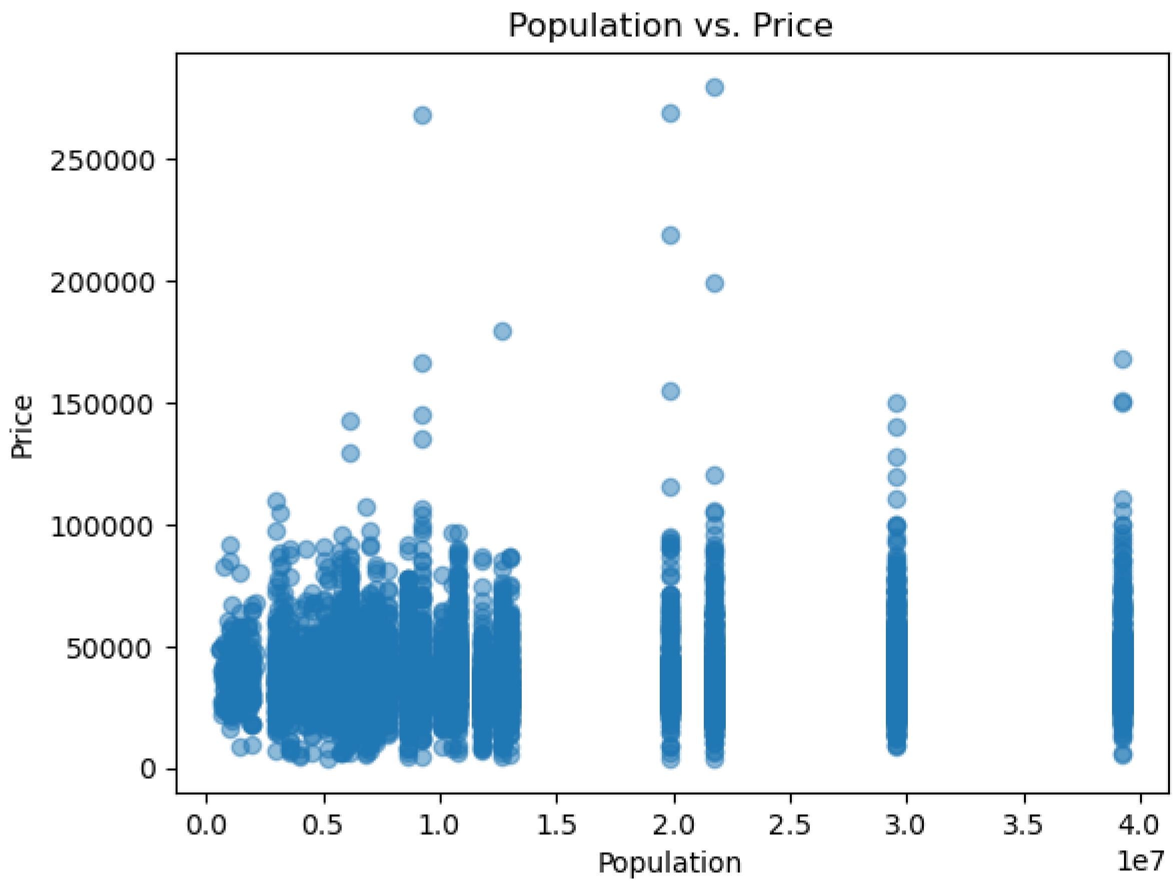
In [229…

```
# Price by deal type
sns.boxplot(x='Deal Type', y='Price', data=combined_data)
plt.title('Deal Type vs. Price')
plt.xlabel('Deal Type')
plt.ylabel('Price')
plt.show()
```



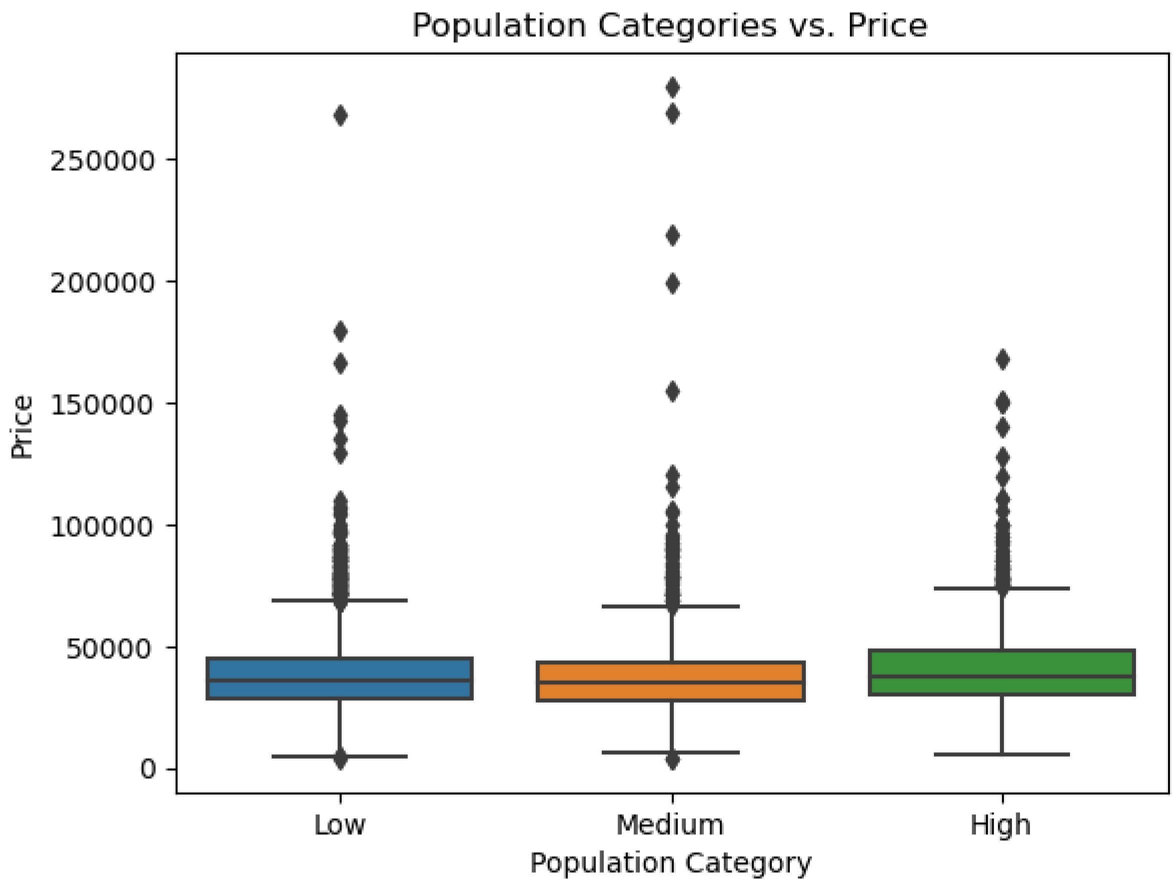
In [233…

```
# Population vs Price
plt.scatter(combined_data['Population'], combined_data['Price'], alpha=0.5)
plt.title('Population vs. Price')
plt.xlabel('Population')
plt.ylabel('Price')
plt.show()
```



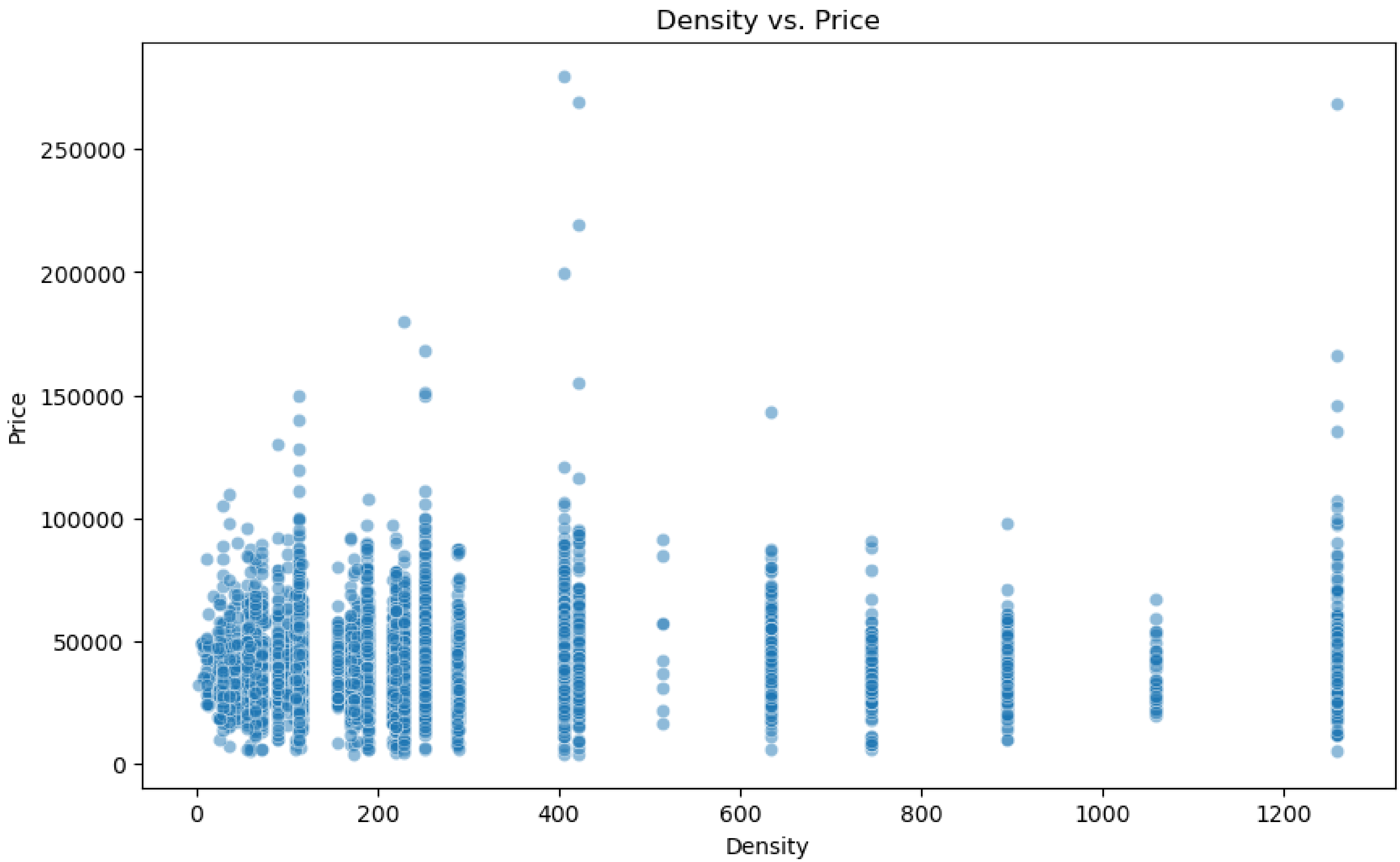
In [231…

```
# Population category vs Price
combined_data['Population_Category'] = pd.cut(combined_data['Population'], bins=3,
                                              labels=['Low', 'Medium', 'High'])
sns.boxplot(x='Population_Category', y='Price', data=combined_data)
plt.title('Population Categories vs. Price')
plt.xlabel('Population Category')
plt.ylabel('Price')
plt.show()
```



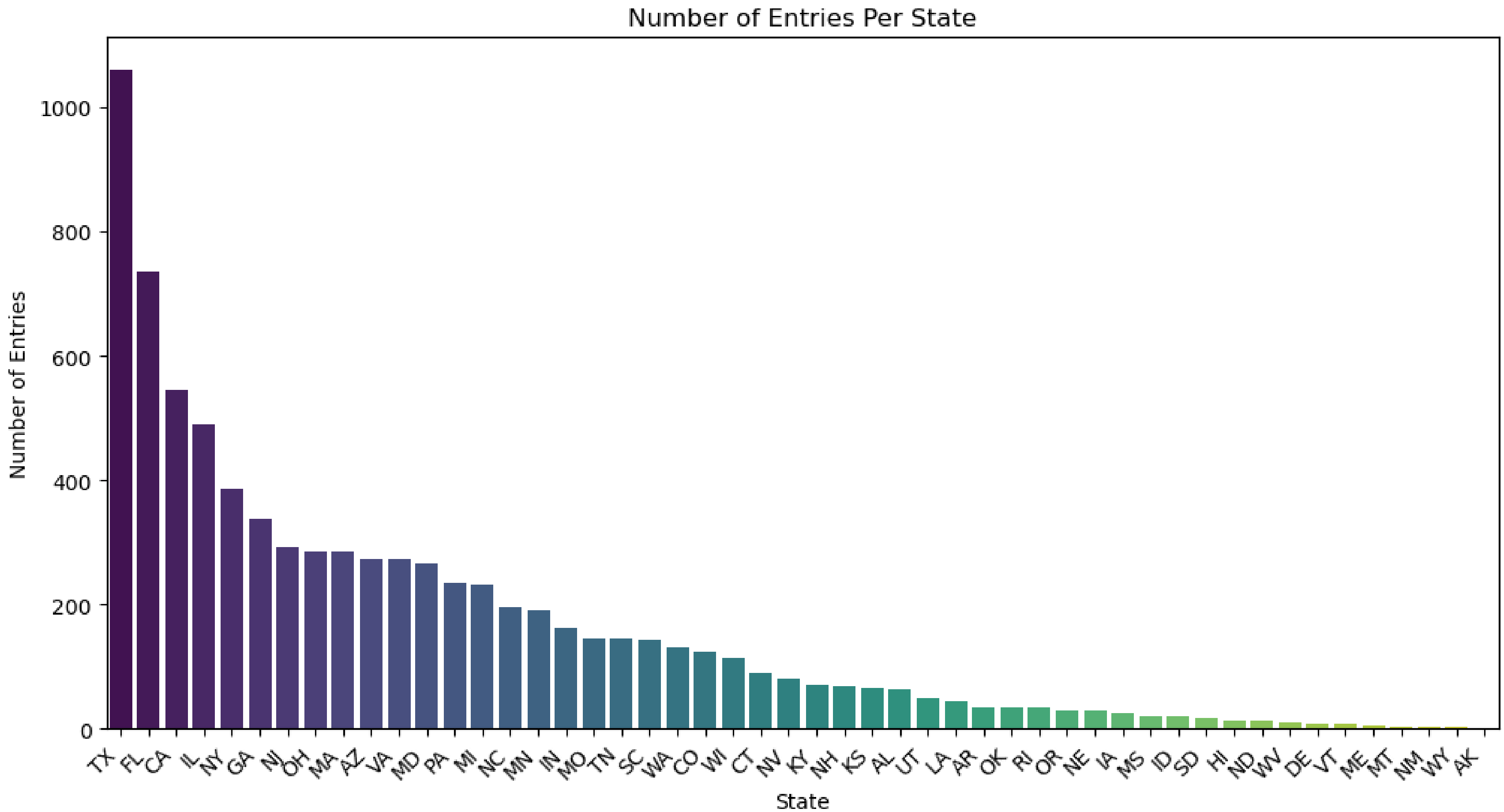
In [235…

```
# Population density vs Price
plt.figure(figsize=(10, 6))
sns.scatterplot(x='Density', y='Price', data=combined_data, alpha=0.5)
plt.title('Density vs. Price')
plt.xlabel('Density')
plt.ylabel('Price')
plt.show()
```



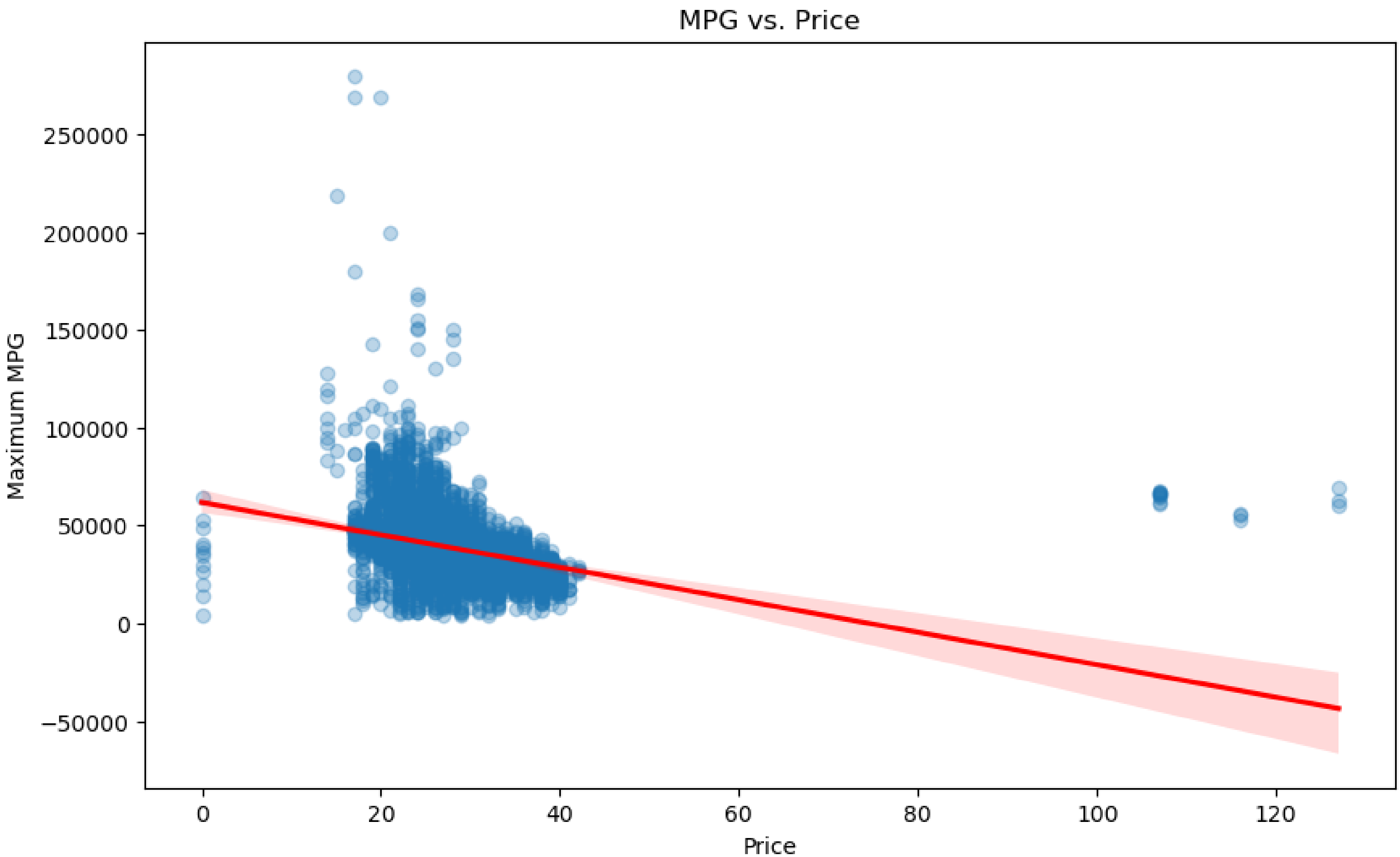
In [237…

```
# State distribution
plt.figure(figsize=(12, 6))
sns.barplot(x=state_counts.index, y=state_counts.values, palette="viridis")
plt.title('Number of Entries Per State')
plt.xlabel('State')
plt.ylabel('Number of Entries')
plt.xticks(rotation=45, ha='right')
plt.show()
```



In [248…

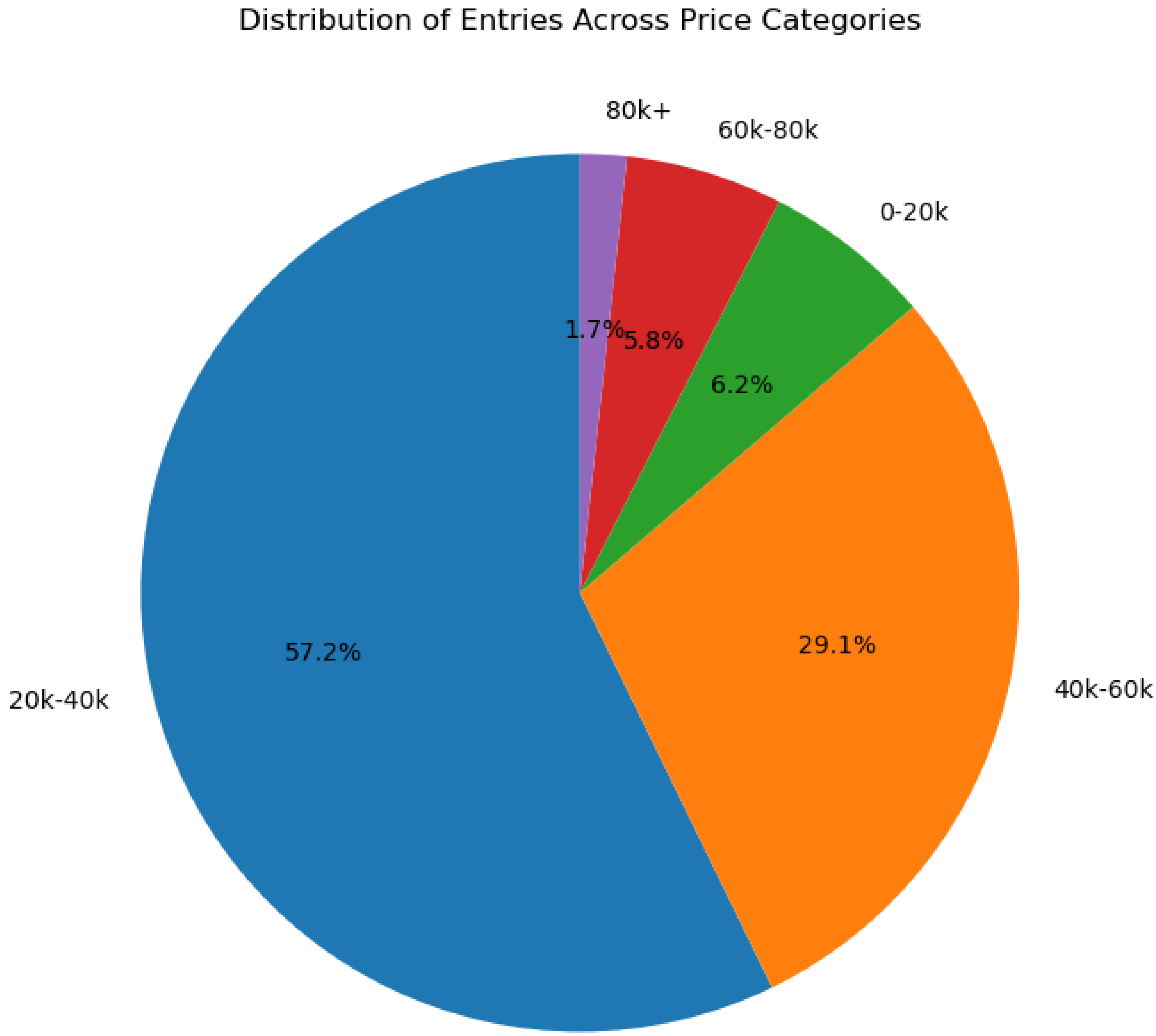
```
# Price vs Max MPG
plt.figure(figsize=(10, 6))
sns.regplot(x='Maximum MPG', y='Price', data=combined_data, scatter_kws={'alpha':0.3},
            line_kws={'color':'red'})
plt.title( 'MPG vs. Price')
plt.xlabel('Price')
plt.ylabel('Maximum MPG')
plt.show()
```



In [250…

```
# Distribution of Pricing
price_bins = [0, 20000, 40000, 60000, 80000, float('inf')]
price_labels = ['0-20k', '20k-40k', '40k-60k', '60k-80k', '80k+']
combined_data['Price_Category'] = pd.cut(combined_data['Price'], bins=price_bins, labels=price_labels)

# Calculate the count of entries for each price category
price_counts = combined_data['Price_Category'].value_counts()

# pie chart
plt.figure(figsize=(8, 8))
plt.pie(price_counts, labels=price_counts.index, autopct='%1.1f%%', startangle=90)
```

```
plt.title('Price Distribution')
plt.show()
```



In [252…

```
combined_data.head()
```

Out[252]:

	Year	Make	Price	Consumer Rating	Consumer Reviews	Seller Rating	Seller Reviews	State	Zipcode	Deal Type	...	Mileage	Average Temperature (F)	Average Temperature (C))	R
0	2019	Toyota	39998.0	4.6	45	3.3	3	CA	92562	Great	...	29403	59.4	15.2	
1	2018	Ford	49985.0	4.8	817	4.8	131	CA	93292	Good	...	32929	59.4	15.2	
2	2017	RAM	41860.0	4.7	495	4.6	249	CA	93637	Good	...	23173	59.4	15.2	
3	2020	Lexus	49000.0	4.8	76	4.8	4755	NV	89011	Good	...	28137	49.9	9.9	
4	2012	Toyota	23541.0	4.7	34	4.4	1071	CA	94544	Fair	...	105469	59.4	15.2	

5 rows × 28 columns

In [ ]: