Used Car Price Analysis

Jim Beno, for U.C. Berkeley Certificate in Machine Learning & Artificial Intelligence

Table of Contents

Business Understanding

Background

A group of used car dealerships would like to know what factors make a car more or less expensive. They are looking for clear recommendations on what consumers value in a used car so they can fine-tune their inventory. A dataset of 426K used car sales has been provided as the source for these insights. We will follow the CRISP-DM framework as a guide for our data analysis process, and strive to meet the following objectives.

Business Objectives and Success Criteria

- **Identify key drivers for used car prices** – The features in the provided dataset that have the highest correlation with price (positive and negative), and exhibit clear linear trends or separation of means, will be identified and shared with the client
- **Provide recommendations on what consumers value** – After analyzing and sythesizing the data, recommendations will be provided on how to adjust their inventory to align with consumer wants or needs.
- **Create price prediction model** – A machine learning model will be created that will attempt to predict the price of a car based on the relationships between features in our dataset. This will help the client accurately price the cars they have in inventory, and get the maximum value out of their assets.

Inventory of Resources

For this project, we have the following resources:

- **Personnel:** (1) Data analyst (part-time)
- **Hardware:** (1) MacBook Pro 2.4 GHz 8-Core Intel Core i9 with 32GB memory. Note: We are limited to CPU processing.
- **Software:** Jupyter Notebook with the following Python libraries:
    - Data analysis: Pandas, NumPy, Missingno
    - Visualization: Matplotlib, Seaborn, Plotly
    - Modeling: SciKit Learn Pre-processing, Compose, Pipeline, Feature selection, Scaling, Linear models, Model selection
- **Data:** (1) Dataset of 426K used car sales records provided by the client

Requirements, Assumptions and Constraints

- **Deliverables** must include:
  - A non-technical presentation that summarizes the key findings for the objectives listed above
  - A Jupyter notebook that shows every step of data processing and analysis
  - Both of these will be published in a public repository on Github that includes a Readme file
- **Data constraint:** We are limited to the dataset provided by the client of 426K records, and cannot utilize any additional data sources
- **Time constraint:** We have one week to complete the project, it's due August 29 at 9:30 AM Pacific time
- No prior work of this dataset by other parties can be leveraged – we will be starting from scratch

Risks and Contingencies

- **Data risk:** The data may not be sufficient to develop a model with strong predictive power. However, we are confident that we can at a minimum describe the data and identify features that are highly correlated with increasing or decreasing price
- **Time and skills risk:** Given the skillset of the data analyst, there are limited machine learning techniques at our disposal. If we don't achieve conclusive results in the allotted timeframe, we can enlist additional resources in a second round.

Terminology

There are a few terms used in this dataset that are unique to the automotive domain. We will define them here:

- **Cylinders:** The number of chambers in an engine where fuel is burned to produce power.
- **Odometer:** A device that measures and displays the total distance a vehicle has traveled.
- **Fuel:** Energy sources used to power a vehicle's engine or motor, with gas and diesel derived from petroleum, and electric from stored electrical energy.
- **Transmission:** A system that controls the power from the engine to the wheels, with manual requiring driver gear shifting and automatic doing it electronically.
- **VIN:** A unique alphanumeric code assigned to each vehicle for identification purposes.
- **Drive:** Describes which wheels receive power from the engine, with "rwd" being rear-wheel drive, "fwd" front-wheel drive, and "4wd" all-wheel drive.

Costs and Benefits

Given the low cost required for this type of analysis, there is great potential return on investment (ROI) that should give the client a competitive edge:

- Knowing what kind of cars are valued by customers will help the dealers be **more selective** when purchasing vehicles for the inventory. If they purchase a vehicle that does not have in-demand features:
  - They **may not be able to sell it** at all
  - Or they will have to sell it for a **very low price**, perhaps less than what they paid for
- Knowing the key drivers of price will allow the used car dealers to **accurately price** the cars they are trying to sell, and maximize the value of their inventory.
  - If they **underprice**, they are leaving money on the table.
  - If they **overprice**, the car may not sell.

Data Mining Goals and Success Criteria

The following data analysis approach will be taken to meet the business goals identified above:

1. **Data Cleaning:** The data set will be examined and cleansed to prepare it for analysis. Missing values and outliers will be handled.

2. **Data Transformation/Encoding:** To maximize the use of the data we have, categorical/nominal values will be encoded into a suitable numeric form. If any variables exhibit a skewed distribution, log transformation may be considered to bring them closer to normal distribution.
3. **Feature Engineering:** In addition, there may be opportunities to engineer new features that are derived from the variables present in the dataset.
4. **Exploratory Data Analysis:** The dataset will be explored using descriptive statistics, correlations, and by plotting charts of variable distributions and multivariate relationships.
5. **Model Iteration:** Data will be split into training and test sets. Pipelines will be constructed to process data, select features, and train regression models. A variety of hyper-parameters will be explored to find the optimum model during cross-validation.
6. **Model Evaluation:** Model performance will be evaluated against the test dataset. The best performing model will be selected based on it's ability to predict the test data, ensuring it was not overfitted to the training data set and can be generalized to new data.

Project Plan

Here is the timeline for the project:

- **August 22:** Project kick-off, document business requirements and plan
- **August 23-24:** Exploratory data analysis, correlations, multivariate charting
- **August 25-26:** Model definition, iteration and evaluation
- **August 27-28:** Document findings and create presentation to client
- **August 29:** Publish Github repo and deliver to client

Initial Assessment of Tools and Techniques

Given what we know if the requirements, the following tools and techniques are planned to be utilized:

- **One-hot Encoding:** The dataset has a number of categorical variables with nominal values, so these will need to be encoded, most likely through one-hot encoding.
- **Correlation Matrix:** Since we're trying to identify the main drivers of price, correlations will be calculated for the various features and encoded values.
- **Regression Models:** Since we're trying to predict the price of a car from a number of variables, we're most likely dealing with a non-liner regression problem. This is not a classification problem, nor a time series problem.
- **Pipelines:** We will be evaluating a variety of models and pre-processing approaches, so will use Pipelines to define these. These also ensure that data for train and test will always be transformed, encoded, scaled consistently.
- **GridSearchCV:** Because we're trying to find the optimal model, and some pipeline components will have parameters that can be tuned or adjusted, will will use GridSearchCV to specify a range of hyperparameter values to explore.

Data Understanding

Data Collection

The data for this project was provided by the client. It is a subset of a dataset that was created by Austin Reese by scraping used car entries on Craigslist, a large classified advertisement website. Here is the latest dataset on Kaggle, and here is the script that was used to scrape the data from Craigslist.

Data Description

A data dictionary was not provided with the source dataset. However, to aid our understanding, the following field descriptions were found on Used Vehicle Data Analysis by Ahmed Sayed. Only the subset of fields in our source dataset are shown.

- **id:** Entry identification.
- **region:** Region in which car are listed for sale.
- **price:** Entry price of the car.
- **year:** The year in which the car was manufactured.
- **manufacturer:** Manufacturer of the vehicle.
- **model:** Model of the vehicle.
- **condition:** Condition of the vehicle e.g. new, good, fair etc.
- **cylinders:** Number of cylinders in the vehicle.
- **fuel:** Fuel type e.g. gas, diesel, electric etc.
- **odometer:** miles traveled by vehicle.
- **title_status:** Vehicle status e.g. clean, rebuilt etc.
- **transmission:** Transmission type of the vehicle e.g. manual, automatic.
- **VIN:** Vehicle identification number.
- **drive:** Type of drive by vehicle e.g. rwd, fwd, 4wd.
- **size:** Size of the vehicle e.g. compact, full-size, etc.
- **type:** Body type of the vehicle e.g. SUV, sedan, etc.
- **paint_color:** Color of the vehicle.
- **state:** Region in which the vehicle are listed for sale.

Import Libraries

In addition to the usual libraries, I created a library with some helper functions called "mytools," it's imported as "my".

In [1]:
```
# Import the initial libraries needed for exploration and visualization
import numpy as np
import pandas as pd
import seaborn as sns
import plotly.express as px
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
from matplotlib.ticker import FuncFormatter
import missingno as msno # Library to visualize missing values
import mytools as my # Helper functions that I wrote
import importlib # Enables reload of library on-demand
```

In [2]:
```
#importlib.reload(my) # Reload custom library during development and testing
```

In [3]:
```
# Fix auto completion issue in some browsers
%config Completer.use_jedi = False
```

Load Data

First step is to load the data and examine it. I'm using a few functions to do this:

- Scan the columns and values using `df.head` and `df.tail`
- In case there are too many columns, I set `display.max_columns` to "None" which is no limit. Now I can see if any column values need cleaning or processing.
- In case there are a long values, I also set `display.max_colwidth` to "None" so I can see those values in full.

In [4]:
```
# Read in the data in CSV format
df = pd.read_csv('data/vehicles.csv')
```

In [5]:
```
# Set the display to maximum columns so we don't miss anything
pd.set_option('display.max_columns', None)

# Turn off truncation of long values in column cells
pd.set_option('display.max_colwidth', None)
#pd.reset_option('display.max_colwidth')
```

In [6]:
```
# Show the first few records of data, and visually scan them
df.head()
```

```
# Show the last few records of data, and visually scan them
#df.tail()
```

Out[6]:

	id	region	price	year	manufacturer	model	condition	cylinders	fuel	odometer	title_status	transmissi
0	7222695916	prescott	6000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
1	7218891961	fayetteville	11900	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
2	7221797935	florida keys	21000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
3	7222270760	worcester / central MA	1500	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
4	7210384030	greensboro	4900	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na

Observation: Already I can see there are many null values in this dataset, the first few rows are filled with them.

Investigate Missing or Problematic Data

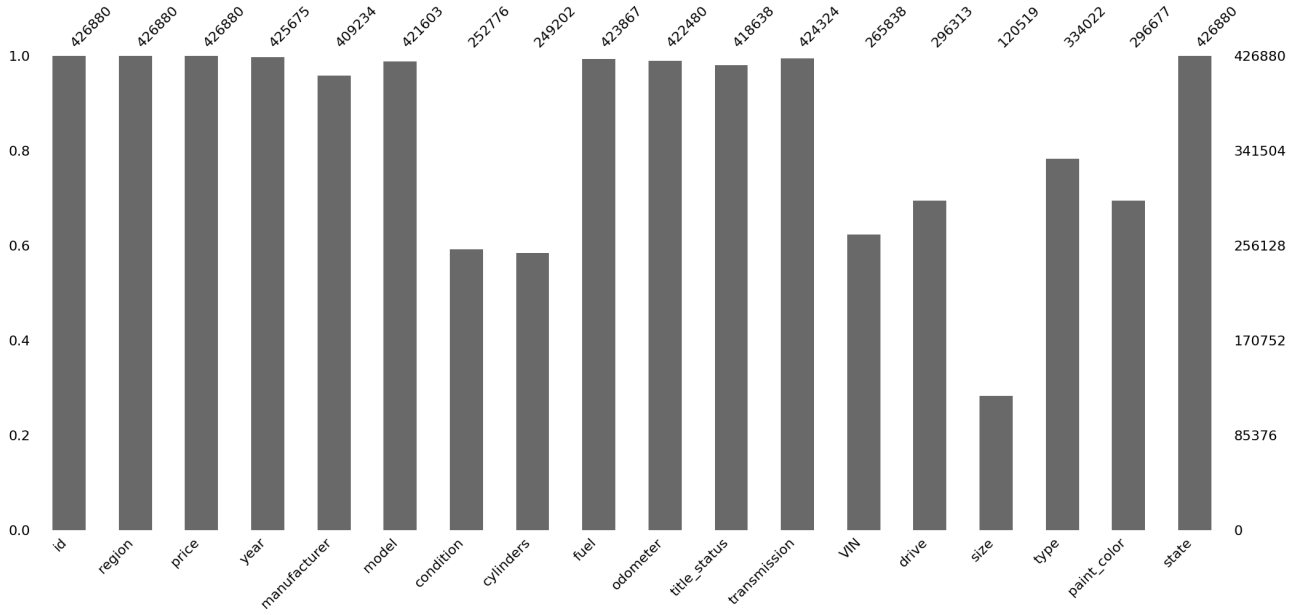
Check for Null Values

Next we need to see what kind of variables we're looking at, and how many null values we have. I'll use `df.info` to start, and look at the non-null counts. I'll also visualize the relationships using `msno.matrix`.

In [7]:
```
# Look at the info for each column, especially the non-null count
df.info()
```

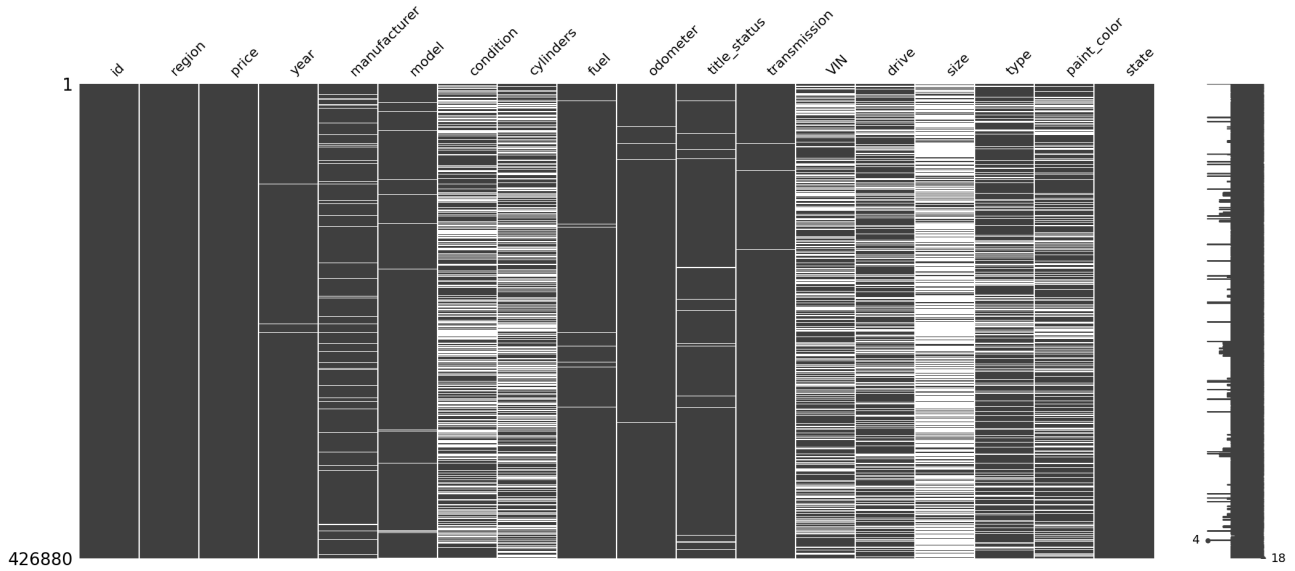
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 426880 entries, 0 to 426879
Data columns (total 18 columns):
 #   Column        Non-Null Count   Dtype  
---  ------        --------------   -----  
 0   id            426880 non-null  int64  
 1   region        426880 non-null  object 
 2   price         426880 non-null  int64  
 3   year          425675 non-null  float64
 4   manufacturer  409234 non-null  object 
 5   model         421603 non-null  object 
 6   condition     252776 non-null  object 
 7   cylinders     249202 non-null  object 
 8   fuel          423867 non-null  object 
 9   odometer      422480 non-null  float64
 10  title_status  418638 non-null  object 
 11  transmission  424324 non-null  object 
 12  VIN           265838 non-null  object 
 13  drive         296313 non-null  object 
 14  size          120519 non-null  object 
 15  type          334022 non-null  object 
 16  paint_color   296677 non-null  object 
 17  state         426880 non-null  object 
dtypes: float64(2), int64(2), object(14)
memory usage: 58.6+ MB
```

In [8]:
```
# Visualize the amount of missing values across the dataset features
msno.bar(df);
```



Observation: From the non-null column and the chart above, we can see that some potentially important features are missing almost half their values: condition, cylinders, drive. There are also other features missing significant values: paint_color, and size.

In [9]:
```
# Visualize the missing data across the dataset to understand potential relationships between variables
msno.matrix(df);
```



Observation: This matrix shows a somewhat random distribution of the missing values, there are no obvious clusters or groupings.

Check for Zero Values

In [10]:
```
df.eq(0).sum()
```

Out[10]:
```
id                  0
region              0
price           32895
year                0
manufacturer        0
model               0
condition           0
cylinders           0
fuel                0
odometer         1965
title_status        0
transmission        0
VIN                 0
drive               0
size                0
type                0
paint_color         0
state               0
dtype: int64
```

Observation: Price has a large number of zero values: 32,895. Since we're focused on helping used car dealers sell cars for a profit, we will likely drop these "$0" records. **Odometer** also has 1,965 values of "0", which is unlikely. It's a relatively small number, so we could just drop those.

Check for Outliers

To check for outliers, I'll look at the descriptive statistics for numerical values with `df.describe`, and look at the maximum and minimum values compared to the Interquartile Range. (I will look for outliers in charts later, this is just a quick check at the beginning)

In [11]:
```
# Set dataframe output to not show scientific notation
pd.set_option('display.float_format', '{:.2f}'.format)
```

In [12]:
```
# Look for extreme outliers in the quantitative data
df.drop(columns=['id']).describe()
```

Out[12]:

	price	year	odometer
count	426880.00	425675.00	422480.00
mean	75199.03	2011.24	98043.33
std	12182282.17	9.45	213881.50
min	0.00	1900.00	0.00
25%	5900.00	2008.00	37704.00
50%	13950.00	2013.00	85548.00
75%	26485.75	2017.00	133542.50
max	3736928711.00	2022.00	10000000.00

Observation: Immediately we can see there are non-sensical values as outliers, in both the maximum "price" and maximum "odometer" reading. Perhaps due to error or intentionally inputting a fake value. This seems to have a significant impact on the mean of the price (\$75,199) vs. the median (\$13,950). I do not know if the minimum values of $0 and 0 miles are legitimate.

Look at Unique Values

Let's take a closer look at the values of each variable. To start, I run `df.unique` to find the number of unique values for each variable. This gives me clues to which ones are categorical vs. continuous numerical. I then use a custom helper function called `my.get_unique` (GitHub) that iterates through each variable that has a unique value count below a specified threshold. It then displays the unique values indented in plain text.

In [13]:
```
# Assemble the unique counts with the data types and null counts for greater insight
unique_counts = df.nunique(axis=0)
dtypes = df.dtypes
null_percent = (df.isna().sum() / len(df)) * 100
unique_dtypes_df = pd.DataFrame({'Unique Values': unique_counts,'Dtype': dtypes,'% Null': null_percent}
```

In [14]:
```
# Get a count of unique values for each variable
unique_dtypes_df
```

Out[14]:

	Unique Values	Dtype	% Null
id	426880	int64	0.00
VIN	118246	object	37.73
odometer	104870	float64	1.03
model	29649	object	1.24
price	15655	int64	0.00
region	404	object	0.00
year	114	float64	0.28
state	51	object	0.00
manufacturer	42	object	4.13
type	13	object	21.75
paint_color	12	object	30.50
cylinders	8	object	41.62
condition	6	object	40.79
title_status	6	object	1.93
fuel	5	object	0.71
size	4	object	71.77
transmission	3	object	0.60
drive	3	object	30.59

Observation: It looks like there are a number of categorical values, everything with the "object" data type. This is everything below 51 unique values, but there are some larger categorical values like region, model and VIN. Model is odd – it's probably categorical, but there are so many unique values it's likely open text input. Will need to take a closer look at its values.

Unique Value Details

In [15]:
```
# Run my custom helper function to show the unique values of each variable
my.get_unique(df, 51, count=True, percent=True)
```

CATEGORICAL: Variables with unique values equal to or below: 51

manufacturer has 42 unique values:

ford	70985	16.63%
chevrolet	55064	12.9%
toyota	34202	8.01%
honda	21269	4.98%
nissan	19067	4.47%
jeep	19014	4.45%
ram	18342	4.3%
nan	17646	4.13%
gmc	16785	3.93%
bmw	14699	3.44%
dodge	13707	3.21%
mercedes-benz	11817	2.77%
hyundai	10338	2.42%
subaru	9495	2.22%
volkswagen	9345	2.19%
kia	8457	1.98%
lexus	8200	1.92%
audi	7573	1.77%
cadillac	6953	1.63%
chrysler	6031	1.41%
acura	5978	1.4%
buick	5501	1.29%
mazda	5427	1.27%
infiniti	4802	1.12%
lincoln	4220	0.99%
volvo	3374	0.79%
mitsubishi	3292	0.77%
mini	2376	0.56%
pontiac	2288	0.54%
rover	2113	0.49%
jaguar	1946	0.46%
porsche	1384	0.32%
mercury	1184	0.28%
saturn	1090	0.26%
alfa-romeo	897	0.21%
tesla	868	0.2%
fiat	792	0.19%
harley-davidson	153	0.04%
ferrari	95	0.02%
datsun	63	0.01%
aston-martin	24	0.01%
land rover	21	0.0%
morgan	3	0.0%

condition has 6 unique values:

nan	174104	40.79%
good	121456	28.45%
excellent	101467	23.77%
like new	21178	4.96%
fair	6769	1.59%
new	1305	0.31%
salvage	601	0.14%

cylinders has 8 unique values:

nan	177678	41.62%
6 cylinders	94169	22.06%
4 cylinders	77642	18.19%
8 cylinders	72062	16.88%
5 cylinders	1712	0.4%
10 cylinders	1455	0.34%
other	1298	0.3%
3 cylinders	655	0.15%
12 cylinders	209	0.05%

fuel has 5 unique values:

gas	356209	83.44%
other	30728	7.2%
diesel	30062	7.04%

```
        hybrid         5170     1.21%
        nan            3013     0.71%
        electric       1698     0.4%

title_status has 6 unique values:

        clean            405117   94.9%
        nan              8242     1.93%
        rebuilt          7219     1.69%
        salvage          3868     0.91%
        lien             1422     0.33%
        missing          814      0.19%
        parts only       198      0.05%

transmission has 3 unique values:

        automatic       336524   78.83%
        other           62682    14.68%
        manual          25118    5.88%
        nan             2556     0.6%

drive has 3 unique values:

        4wd       131904   30.9%
        nan       130567   30.59%
        fwd       105517   24.72%
        rwd       58892    13.8%

size has 4 unique values:

        nan               306361   71.77%
        full-size         63465    14.87%
        mid-size          34476    8.08%
        compact           19384    4.54%
        sub-compact       3194     0.75%

type has 13 unique values:

        nan               92858   21.75%
        sedan             87056   20.39%
        SUV               77284   18.1%
        pickup            43510   10.19%
        truck             35279   8.26%
        other             22110   5.18%
        coupe             19204   4.5%
        hatchback         16598   3.89%
        wagon             10751   2.52%
        van               8548    2.0%
        convertible       7731    1.81%
        mini-van          4825    1.13%
        offroad           609     0.14%
        bus               517     0.12%

paint_color has 12 unique values:

        nan          130203   30.5%
        white        79285    18.57%
        black        62861    14.73%
        silver       42970    10.07%
        blue         31223    7.31%
        red          30473    7.14%
        grey         24416    5.72%
        green        7343     1.72%
        custom       6700     1.57%
        brown        6593     1.54%
        yellow       2142     0.5%
        orange       1984     0.46%
        purple       687      0.16%

state has 51 unique values:

        ca       50614   11.86%
        fl       28511   6.68%
        tx       22945   5.38%
        ny       19386   4.54%
```

```
oh        17696   4.15%
or        17104   4.01%
mi        16900   3.96%
nc        15277   3.58%
wa        13861   3.25%
pa        13753   3.22%
wi        11398   2.67%
co        11088   2.6%
tn        11066   2.59%
va        10732   2.51%
il        10387   2.43%
nj        9742    2.28%
id        8961    2.1%
az        8679    2.03%
ia        8632    2.02%
ma        8174    1.91%
mn        7716    1.81%
ga        7003    1.64%
ok        6792    1.59%
sc        6327    1.48%
mt        6294    1.47%
ks        6209    1.45%
in        5704    1.34%
ct        5188    1.22%
al        4955    1.16%
md        4778    1.12%
nm        4425    1.04%
mo        4293    1.01%
ky        4149    0.97%
ar        4038    0.95%
ak        3474    0.81%
la        3196    0.75%
nv        3194    0.75%
nh        2981    0.7%
dc        2970    0.7%
me        2966    0.69%
hi        2964    0.69%
vt        2513    0.59%
ri        2320    0.54%
sd        1302    0.31%
ut        1150    0.27%
wv        1052    0.25%
ne        1036    0.24%
ms        1016    0.24%
de        949     0.22%
wy        610     0.14%
nd        410     0.1%
```

Insight: We're starting to learn some things about our data:

- **Ford** (16.63%) is the most popular manufacturer, followed by Chevrolet (12.9%) and Toyota (8.01%)
- Surprisingly, **Ferraris** do get sold on Craigslist – 95 times in this dataset!
- Most people did not specify a **condition** (40.79%). Perhaps they did not want to advertise anything negative. I think we can handle this by filling the null values with "Not specified"
- Most people did not specify the **cylinders** (41.62%), and a large number did not specify the **drive train** (30.59%). Perhaps they did not know. We could fill these null values in with "Not specified" as well.
- Most people did not specify the **size** (71.77%), which is surprising. Perhaps they did not know how to properly classify, and it must have been optional. This could also be "Not specified"
- **Gas** was the dominant fuel type at 83.44%
- Almost all the titles were **Clean** (94.9%). There are some interesting title status values: Rebuilt, Lien, Salvage, Parts Only.
- **Automatic** was the dominant transmissino (78.83%)
- **White** was the most popular color, after the 30.5% that did not specify a color. Why couldn't they name the color? Purple was the last popular color! (0.16%)
- Most of the listings were in **CA** (11.86%) > FL (6.68%) > TX (5.38%) > NY (4.54%), in that order.

In [16]:
```
# Look at the unique values for some of the larger categorical values (Region)
#my.get_unique(df[['region']], 404, count=True, percent=True)
```

In [17]:
```
# Look at the unique values for some of the larger categorical values (Model)
#my.get_unique(df[['model']], 29649, count=True, percent=True)
# Caution: This is CPU intensive
```

Observation: Upon taking a closer look at the model values, we see all sorts of open-text input by the people that created the listings. There might be a way to capture some of the information in here, perhaps by categorizing everything below 0.5% as "Other," and then either trying to associate by keyword the ones that make up the majority of the dataset. It's too many values to one-hot enocde, I think, but we could look at target encoding them with the mean price. We could also one-hot encode and then use Principal Component Analysis (PCA) to eliminate all the noise we created. This might be an option to see if any of the model values make it into a principal component, otherwise we can just drop the feature entirely.

Check Unique Identifiers

Let's see if we truly have unique identifiers in "id" and "VIN".

In [18]:
```
# Check if id is a unique identifier
df['id'].nunique(), len(df) # Yes, if counts match
```

Out[18]:
```
(426880, 426880)
```

In [19]:
```
# Check if VIN is a unique identifier
df['VIN'].nunique(), len(df) # Yes, if counts match
```

Out[19]:
```
(118246, 426880)
```

In [20]:
```
# Convert id to an object data type, to avoid numerical operations on it
df['id'] = df['id'].astype('object')
```

Check for Duplicate Records

There is a chance the same VIN could be listed more than once. Let's see if we have duplicate records with the same VIN.

In [21]:
```
# See if VIN is unique, but guessing not, since the same car can be listed multiple times
vin_counts = df['VIN'].value_counts()
duplicated_vin_counts = vin_counts[vin_counts > 1]
print(duplicated_vin_counts)
```

```
VIN
1FMJU1JT1HEA52352    261
3C6JR6DT3KG560649    235
1FTER1EH1LLA36301    231
5TFTX4CN3EX042751    227
1GCHTCE37G1186784    214
                    ...
SHHFK7H43JU227804      2
3N1CN7AP1KL801368      2
3C6UR5FL0GG178879      2
5N1DL0MM6JC505090      2
KNDPMCAC1J7393528      2
Name: count, Length: 40280, dtype: int64
```

In [22]:
```
df_1FMJU1JT1HEA52352 = df.query("VIN == '1FMJU1JT1HEA52352'")
```

In [351…
```
df_1FMJU1JT1HEA52352[['id','VIN','price','odometer','year','manufacturer','model','region']][:50]
```

Out[351]:

	id	VIN	price	odometer	year	manufacturer	model	region
76	7311818189	1FMJU1JT1HEA52352	29590	70760.0000	2017.0000	ford	expedition xlt sport	auburn
707	7311865995	1FMJU1JT1HEA52352	29590	70760.0000	2017.0000	ford	expedition xlt sport	birmingham
1935	7311801032	1FMJU1JT1HEA52352	29590	70760.0000	2017.0000	ford	expedition xlt sport	dothan
2201	7311915565	1FMJU1JT1HEA52352	29590	70760.0000	2017.0000	ford	expedition xlt sport	florence / muscle shoals
2377	7311823034	1FMJU1JT1HEA52352	29590	70760.0000	2017.0000	ford	expedition xlt sport	gadsden-anniston
3031	7311871863	1FMJU1JT1HEA52352	29590	70760.0000	2017.0000	ford	expedition xlt sport	huntsville / decatur
4591	7311801056	1FMJU1JT1HEA52352	29590	70760.0000	2017.0000	ford	expedition xlt sport	montgomery
4878	7311843143	1FMJU1JT1HEA52352	29590	70760.0000	2017.0000	ford	expedition xlt sport	tuscaloosa
8765	7311773165	1FMJU1JT1HEA52352	29590	70760.0000	2017.0000	ford	expedition xlt sport	flagstaff / sedona
13056	7311801128	1FMJU1JT1HEA52352	29590	70760.0000	2017.0000	ford	expedition xlt sport	prescott
13667	7311865754	1FMJU1JT1HEA52352	29590	70760.0000	2017.0000	ford	expedition xlt sport	sierra vista
16199	7311928717	1FMJU1JT1HEA52352	29590	70760.0000	2017.0000	ford	expedition xlt sport	tucson
18899	7311854351	1FMJU1JT1HEA52352	29590	70760.0000	2017.0000	ford	expedition xlt sport	jonesboro
19858	7311767733	1FMJU1JT1HEA52352	29590	70760.0000	2017.0000	ford	expedition xlt sport	little rock
21041	7311767744	1FMJU1JT1HEA52352	29590	70760.0000	2017.0000	ford	expedition xlt sport	texarkana
22081	7311865951	1FMJU1JT1HEA52352	29590	70760.0000	2017.0000	ford	expedition xlt sport	bakersfield
24286	7311827774	1FMJU1JT1HEA52352	29590	70760.0000	2017.0000	ford	expedition xlt sport	chico
27261	7311767811	1FMJU1JT1HEA52352	29590	70760.0000	2017.0000	ford	expedition xlt sport	fresno / madera
28587	7311884132	1FMJU1JT1HEA52352	29590	70760.0000	2017.0000	ford	expedition xlt sport	gold country
29018	7311922032	1FMJU1JT1HEA52352	29590	70760.0000	2017.0000	ford	expedition xlt sport	hanford-corcoran
36199	7311767782	1FMJU1JT1HEA52352	29590	70760.0000	2017.0000	ford	expedition xlt sport	merced
37938	7311766823	1FMJU1JT1HEA52352	29590	70760.0000	2017.0000	ford	expedition xlt sport	modesto
40102	7311866035	1FMJU1JT1HEA52352	29590	70760.0000	2017.0000	ford	expedition xlt sport	monterey bay
46822	7311770016	1FMJU1JT1HEA52352	29590	70760.0000	2017.0000	ford	expedition xlt sport	redding
50469	7311771948	1FMJU1JT1HEA52352	29590	70760.0000	2017.0000	ford	expedition xlt sport	reno / tahoe
59089	7311771844	1FMJU1JT1HEA52352	29590	70760.0000	2017.0000	ford	expedition xlt sport	santa barbara
60039	7311766813	1FMJU1JT1HEA52352	29590	70760.0000	2017.0000	ford	expedition xlt sport	santa maria
65049	7311766828	1FMJU1JT1HEA52352	29590	70760.0000	2017.0000	ford	expedition xlt sport	stockton

	id	VIN	price	odometer	year	manufacturer	model	region
67558	7311767765	1FMJU1JT1HEA52352	29590	70760.0000	2017.0000	ford	expedition xlt sport	ventura county
69472	7311801044	1FMJU1JT1HEA52352	29590	70760.0000	2017.0000	ford	expedition xlt sport	visalia-tulare
70791	7311837799	1FMJU1JT1HEA52352	29590	70760.0000	2017.0000	ford	expedition xlt sport	yuba-sutter
72054	7311860244	1FMJU1JT1HEA52352	29590	70760.0000	2017.0000	ford	expedition xlt sport	boulder
73813	7311948282	1FMJU1JT1HEA52352	29590	70760.0000	2017.0000	ford	expedition xlt sport	colorado springs
79281	7311916305	1FMJU1JT1HEA52352	29590	70760.0000	2017.0000	ford	expedition xlt sport	fort collins / north CO
81186	7311770020	1FMJU1JT1HEA52352	29590	70760.0000	2017.0000	ford	expedition xlt sport	pueblo
83107	7311784000	1FMJU1JT1HEA52352	29590	70760.0000	2017.0000	ford	expedition xlt sport	eastern CT
84389	7311980297	1FMJU1JT1HEA52352	29590	70760.0000	2017.0000	ford	expedition xlt sport	hartford
86667	7311955062	1FMJU1JT1HEA52352	29590	70760.0000	2017.0000	ford	expedition xlt sport	new haven
87830	7311854347	1FMJU1JT1HEA52352	29590	70760.0000	2017.0000	ford	expedition xlt sport	northwest CT
91361	7311768648	1FMJU1JT1HEA52352	29590	70760.0000	2017.0000	ford	expedition xlt sport	delaware
92629	7311795206	1FMJU1JT1HEA52352	29590	70760.0000	2017.0000	ford	expedition xlt sport	daytona beach
93689	7311814233	1FMJU1JT1HEA52352	29590	70760.0000	2017.0000	ford	expedition xlt sport	florida keys
95126	7311916308	1FMJU1JT1HEA52352	29590	70760.0000	2017.0000	ford	expedition xlt sport	ft myers / SW florida
97121	7311848554	1FMJU1JT1HEA52352	29590	70760.0000	2017.0000	ford	expedition xlt sport	gainesville
97761	7311767780	1FMJU1JT1HEA52352	29590	70760.0000	2017.0000	ford	expedition xlt sport	heartland florida
99309	7311902587	1FMJU1JT1HEA52352	29590	70760.0000	2017.0000	ford	expedition xlt sport	jacksonville
101189	7311928907	1FMJU1JT1HEA52352	29590	70760.0000	2017.0000	ford	expedition xlt sport	lakeland
102768	7311793553	1FMJU1JT1HEA52352	29590	70760.0000	2017.0000	ford	expedition xlt sport	ocala
106898	7311909263	1FMJU1JT1HEA52352	29590	70760.0000	2017.0000	ford	expedition xlt sport	orlando
109262	7311896232	1FMJU1JT1HEA52352	29590	70760.0000	2017.0000	ford	expedition xlt sport	sarasota-bradenton

Observation: So "id" is a unique identifier for each record, but "VIN" is not. In fact, some vehicles have been listed numerous times. Vehicle 1FMJU1JT1HEA52352261 was listed the most at **261 times!** But it looks like the **price and odometer did not change**, even though it was listed in multiple states. This is probably an example of a seller posting multiple listings in different states. We can eliminate most of these as duplicates, but we can't assume this is the same situation for every VIN listed more than once. However, I will consider dropping duplicates where the "VIN" and "price" and "odometer" are the same.

Define Column Lists

Since many data processing operations only make sense on numerical vs. categorical features, and we also have to consider our X and y features in mind, let's create lists that we can use later, based on what we've learned so far.

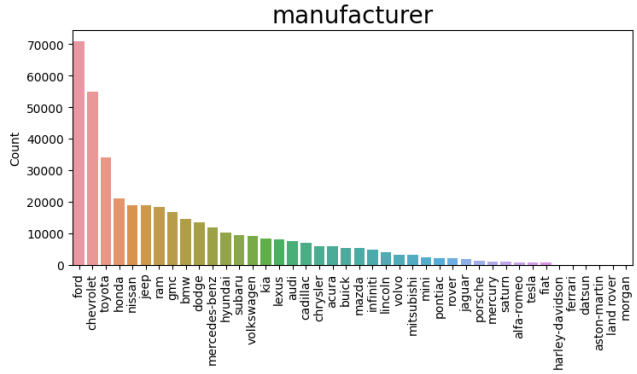
In [24]:
```
# Create lists of columns we can use later
#
# all_columns
#     x_columns
#         x_num_columns
#         x_cat_columns
#         x_bool_columns
#     y_column
#     num_columns
#     cat_columns
#     bool_columns
#
# Note: If categorical columns are numeric dtypes than convert them to objects/categories or manually s
# df['column_name'] = df['column_name'].astype('object') # or 'category'
#
all_columns = list(df.columns)
id_columns = ['id']
x_columns = [col for col in all_columns if col not in ['price','id', 'VIN']]
x_num_columns = [col for col in x_columns if df[col].dtype in ['int64', 'float64']]
x_cat_columns = [col for col in x_columns if df[col].dtype in ['object', 'category', 'string']]
x_bool_columns = [col for col in x_columns if df[col].dtype in ['bool']]
y_column = ['price']
num_columns = [col for col in all_columns if df[col].dtype in ['int64', 'float64']]
cat_columns = [col for col in all_columns if df[col].dtype in ['object', 'category', 'string']]
bool_columns = [col for col in all_columns if df[col].dtype in ['bool']]
```

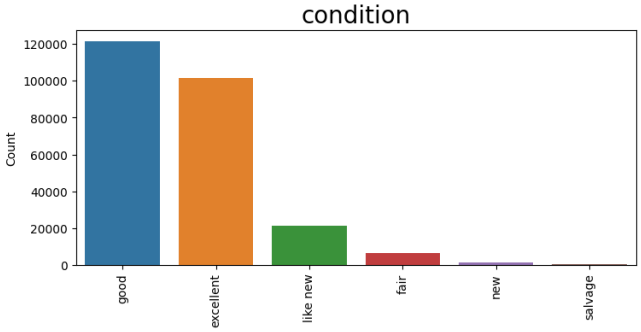
Data Exploration

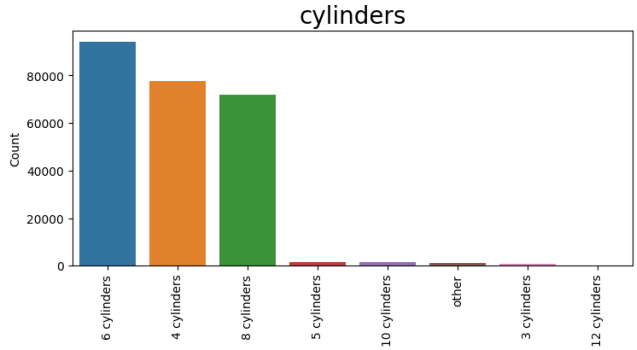
Next step is to start visualizing some of the data. I like to plot the distributions for all the variables. I created a function called `my.plot_charts` that outputs a bunch of Seaborn bar charts or histograms.

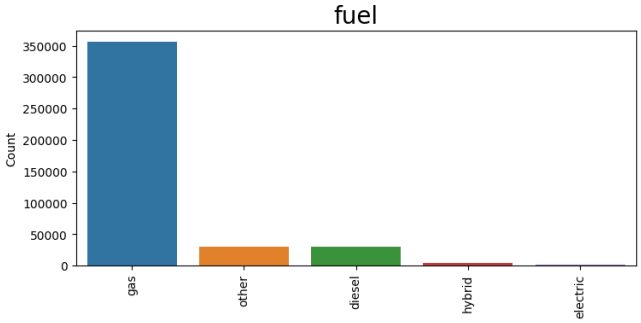
Categorical Distributions

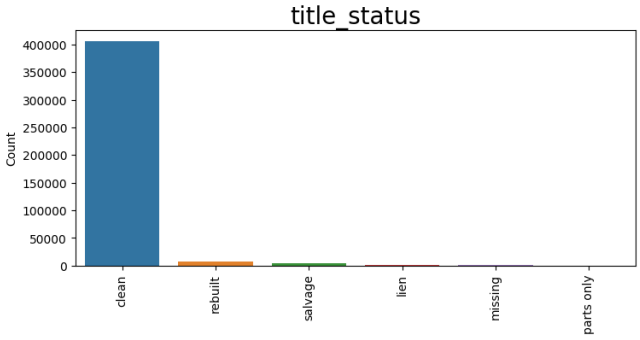
In [25]:
```
# Plot bar charts of categorical variables
my.plot_charts(df, plot_type='cat', n=51, ncols=2, fig_width=15, rotation=90)
```

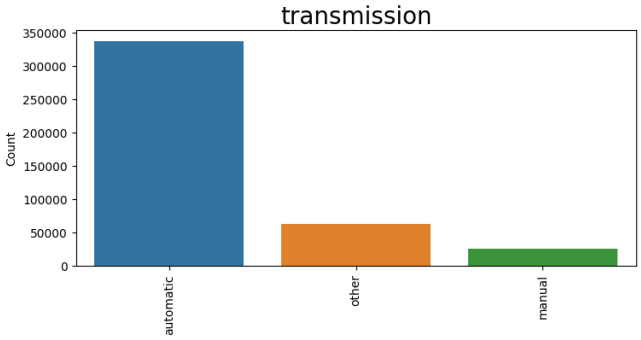

manufacturer
Count
ford
chevrolet
toyota
honda
nissan
jeep
ram
gmc
bmw
dodge
mercedes-benz
hyundai
subaru
volkswagen
kia
lexus
audi
cadillac
chrysler
acura
buick
mazda
infiniti
lincoln
volvo
mitsubishi
mini
pontiac
rover
jaguar
porsche
mercury
saturn
alfa-romeo
tesla
fiat
harley-davidson
ferrari
datsun
aston-martin
land rover
morgan

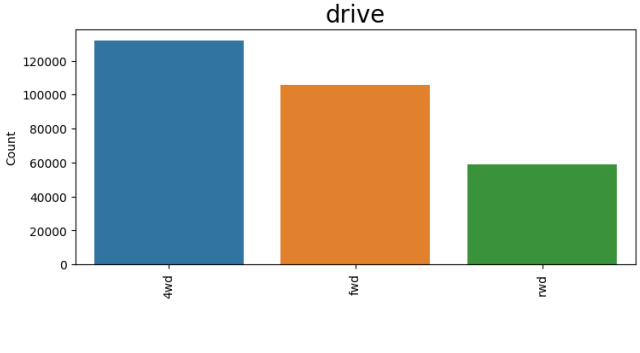

condition
Count
good
excellent
like new
fair
new
salvage

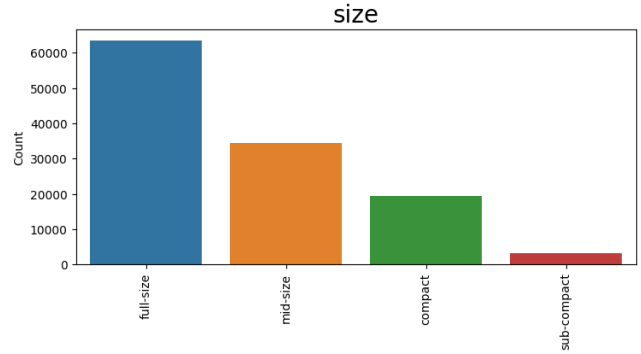

cylinders
Count
6 cylinders
4 cylinders
8 cylinders
5 cylinders
10 cylinders
other
3 cylinders
12 cylinders

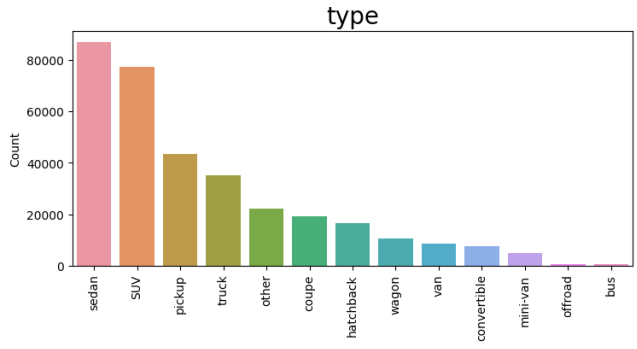

fuel
Count
gas
other
diesel
hybrid
electric

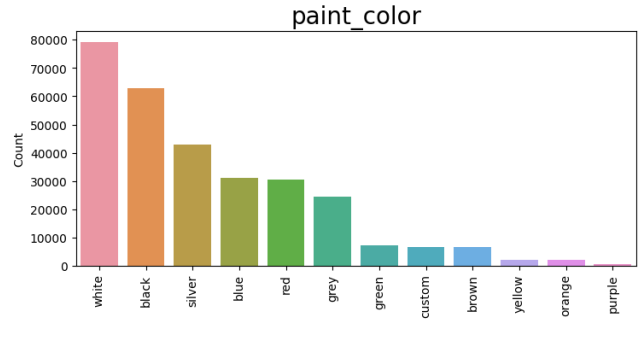

title_status
Count
clean
rebuilt
salvage
lien
missing
parts only


transmission
Count
automatic
other
manual

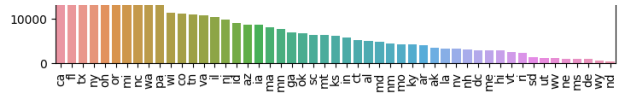

drive
Count
4wd
fwd
rwd


size
Count
full-size
mid-size
compact
sub-compact

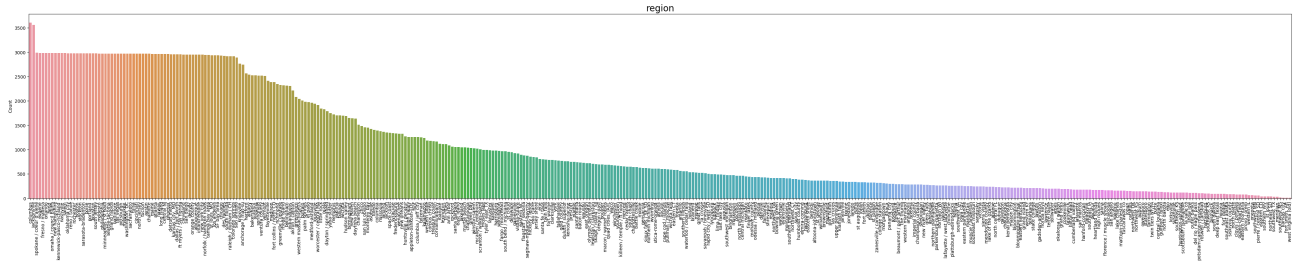

type
Count
sedan
SUV
pickup
truck
other
coupe
hatchback
wagon
van
convertible
mini-van
offroad
bus


paint_color
Count
white
black
silver
blue
red
grey
green
custom
brown
yellow
orange
purple


state
Count



In [26]:
```
# Plot a single chart for the larger value sets
my.plot_charts(df, plot_type='cat', n=404, cat_cols=['region'], ncols=1, fig_width=40, subplot_height=8
```

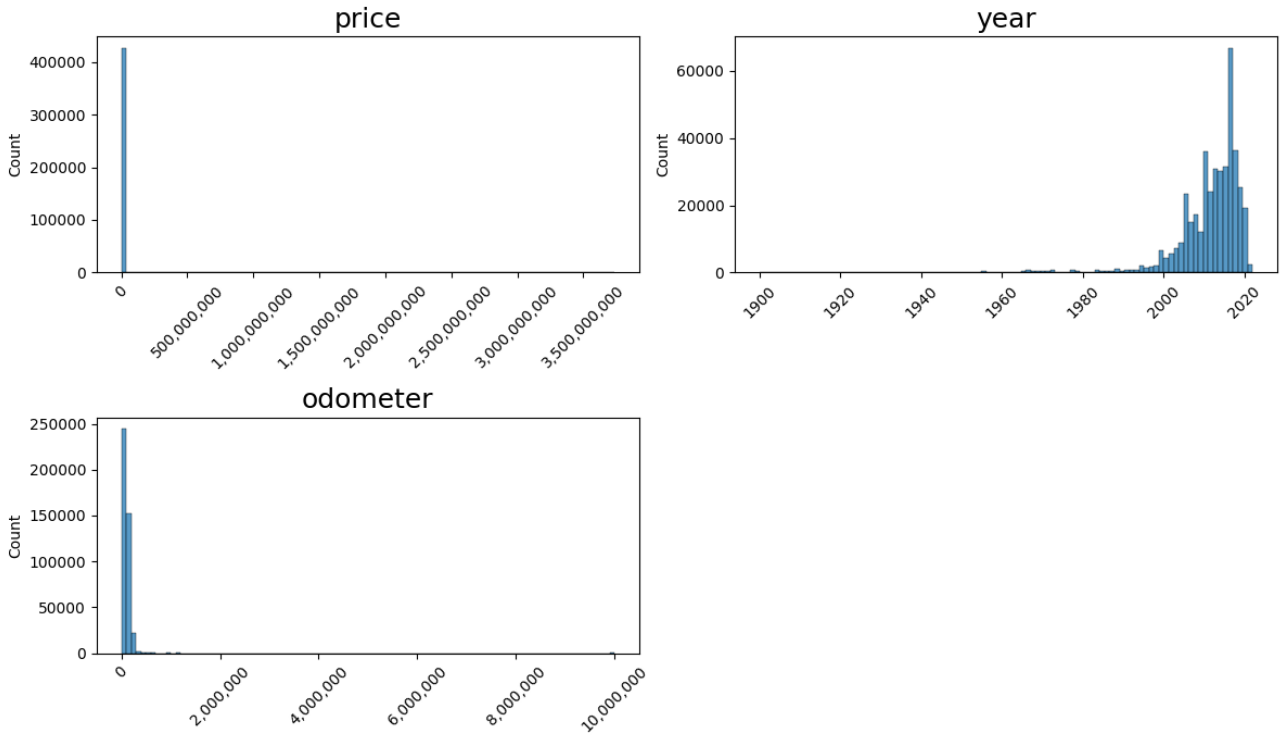


Observation: Most of the plots above confirm what we saw in the unique value summaries and breakdowns earlier. The Region plot is the first we're looking at it's details. The "plateau" shape is interesting. I do not know what causes that. Perhaps the same sellers are posting many listings to those major locations over and over.

Continuous Distributions

Now we'll look at the continuous variables in histograms, with our variable of interest as a dimension.

In [27]:
```
# Plot the histograms for continuous variables
plt.figure(figsize=(12,7))
for i in range(len(num_columns)):
    plt.subplot(2,2,i+1)
    plt.title(num_columns[i], fontsize=18)
    sns.histplot(x=num_columns[i], data=df, bins=100)
    plt.xlabel('')
    plt.xticks(rotation=45)
    if num_columns[i] != 'year':
        plt.gca().xaxis.set_major_formatter(FuncFormatter(my.thousands))
plt.tight_layout()
plt.show()
```



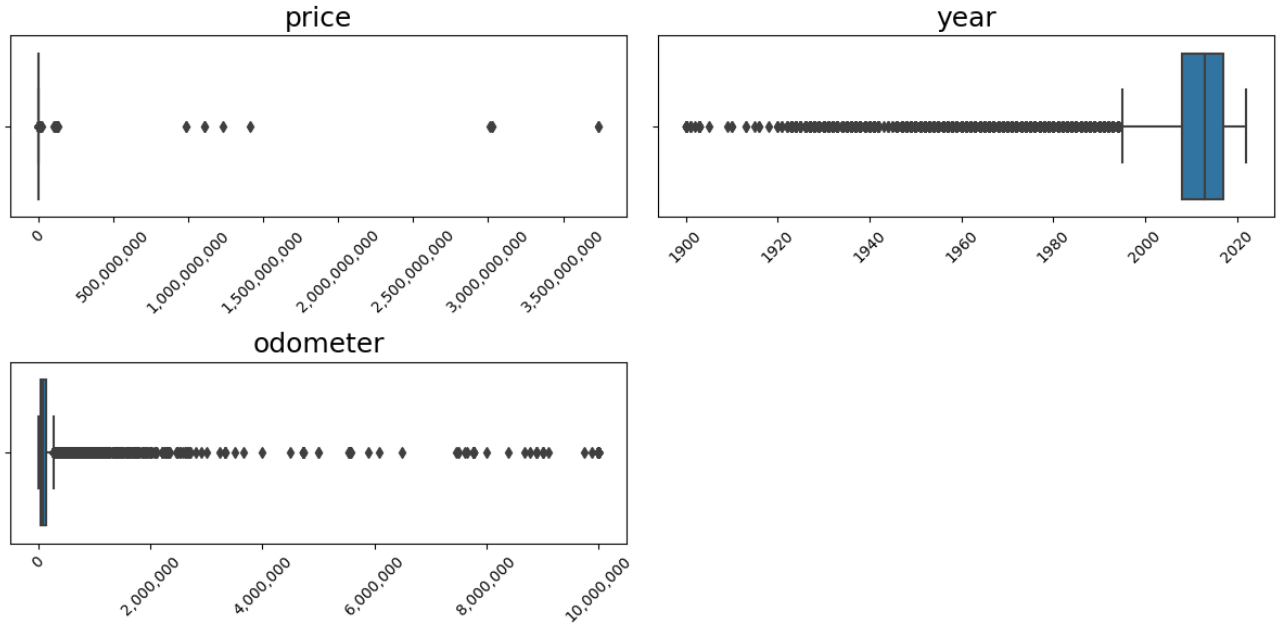
Observation: The histograms above show a few things that we're going to have to address. The true distribution of the data is completely obscured due to the following:

- **Price** may have a large number of **$0 values**, it's hard to tell from this scale
- **Price** also has impossibly **extreme values**, clearly this is junk data that we need to drop above a realistic price threshold
- **Odometer** also has impossibly **extreme values**, we will need to drop those.
- **Year** has a **skewed distribution** with a long tail. I doubt that values in the early 1900s are realistic, we may need to drop those

Outlier Investigation

I'll now take a closer look at the potential outliers in the long tails and see if there's anything we can do about them.

In [28]:
```
# Plot box plots to inspect outlier distribution
plt.figure(figsize=(12,6))
for i in range(len(num_columns)):
    plt.subplot(2,2,i+1)
    plt.title(num_columns[i], fontsize=18)
    sns.boxplot(x=num_columns[i], data=df)
    plt.xlabel('')
    plt.xticks(rotation=45)
    if num_columns[i] != 'year':
        plt.gca().xaxis.set_major_formatter(FuncFormatter(my.thousands))
plt.tight_layout()
plt.show()
```



In [29]:
```
# Let's look at the 95th and 99th percentiles
df.describe(percentiles=[.25, .5, .75, .95, .99])
```

Out[29]:

	price	year	odometer
count	426880.00	425675.00	422480.00
mean	75199.03	2011.24	98043.33
std	12182282.17	9.45	213881.50
min	0.00	1900.00	0.00
25%	5900.00	2008.00	37704.00
50%	13950.00	2013.00	85548.00
75%	26485.75	2017.00	133542.50
95%	44500.00	2020.00	204000.00
99%	66995.00	2020.00	280000.00
max	3736928711.00	2022.00	10000000.00

In [30]:

```
# Count prices at each level, note the bins are not equal intervals
df_bins = pd.DataFrame.copy(df[['price','odometer','year']])

price_bins = [0, 1, 10, 100, 1000, 5000, 10000, 20000, 30000, 50000, 100000, 200000, 400000, 600000, 8(
price_labels = ['0-1', '1-10', '10-100', '100-1k', '1k-5k', '5k-10k', '10k-20k', '20k-30k', '30k-50k',

odometer_bins = [0, 1, 5000, 15000, 30000, 50000, 75000, 100000, 200000, 400000, 600000, 800000, 10000(
odometer_labels = ['0-1', '1-5k', '5k-15k', '15k-30k', '30k-50k', '50k-75k', '75k-100k', '100k-200k',

year_bins = list(range(1900, 2031, 10))
year_labels = ['{}-{}'.format(i, i + 9) for i in range(1900, 2021, 10)]

df_bins['price_bin'] = pd.cut(df_bins['price'], bins=price_bins, labels=price_labels, right=False)
print("Price bins:\n", df_bins['price_bin'].value_counts(sort=False))

df_bins['odometer_bin'] = pd.cut(df_bins['odometer'], bins=odometer_bins, labels=odometer_labels, right
print("\nOdometer bins:\n", df_bins['odometer_bin'].value_counts(sort=False))

df_bins['year_bin'] = pd.cut(df_bins['year'], bins=year_bins, labels=year_labels, right=False)
print("\nYear bins:\n", df_bins['year_bin'].value_counts(sort=False))
```

```
Price bins:
 price_bin
0-1           32895
1-10           2034
10-100         1293
100-1k        10093
1k-5k         48997
5k-10k        79652
10k-20k      102918
20k-30k       69758
30k-50k       66199
50k-100k      12344
100k-200k       570
200k-400k        55
400k-600k         7
600k-800k         3
800k-1M           2
1M+              60
Name: count, dtype: int64

Odometer bins:
 odometer_bin
0-1            1965
1-5k          16614
5k-15k        23812
15k-30k       42028
30k-50k       50353
50k-75k       52395
75k-100k      57711
100k-200k    152704
200k-400k     23165
400k-600k       473
600k-800k       221
800k-1M         400
1M+             639
Name: count, dtype: int64

Year bins:
 year_bin
1900-1909        30
1910-1919         8
1920-1929       208
1930-1939       480
1940-1949       487
1950-1959      1245
1960-1969      2824
1970-1979      3131
1980-1989      3970
1990-1999     12867
2000-2009     97317
2010-2019    281281
2020-2029     21827
Name: count, dtype: int64
```

Observation: The box plots above provide another perspective on our outliers. In the histograms, we couldn't see the frequency and spread between the points in the long tail. Here we can see the individual data points very clearly as diamonds. The binned counts give us more details at periodic intervals. There are just a handful of extreme values for Price, but quite a number more for Odometer. We need to remove these, either by specifying a logical threshold, or perhaps Inter-Quartile Range (IQR).

Data Quality Plan

Here is the summary of the issues we discoverd, and our plan to adress them.

Data Cleaning

1. **Duplicate Values:** VIN is a unique identifier for a vehicle, but it's not a unique identifier in this dataset. It has a number of missing values, and it's also duplicated in multiple records. For some of these, the price, odometer and

car description do not change – the only thing that changes is the location. So we will remove these duplicates.

   A. **VIN**
      a. Remove duplicate records with the same VIN
      b. Drop the VIN column

2. **Unnecessary Features:** Some identifier columns are not valuable as features, so we will now drop those. Title Status is 94% "Clean," so it's not worth trying to get an effect without any variance. And Region has too many values to one-hot encode, and is represented at a higher level by State. So we will OHE State and drop Region.
   A. **ID**
      a. Drop column
   B. **VIN**
      a. Drop column
   C. **Title Status**
      a. Drop column
   D. **Region**
      a. Drop column
3. **Zero Values** Price has 32k zero values, and Odometer 1,965 zero values. Our goal is to help used car dealers predict price, and \$0 is not a price they care about. We will drop these for now.
   A. **Price**
      a. Drop all \$0 values.
   B. **Odometer**
      a. Drop all 0 values.
4. **Null Values:** Condition, Cylinders, and Drive are missing almost half their values are are likely important features that we want to retain. Paint Color and VIN are also missing some values. Size is missing the most. We'll try to retain these features for now, but may drop them later if they don't increase accuracy of the model. Some numeric values are also empty. I experimented with imputing them, but believe dropping them is best.
   A. **Condition**
      a. Replace "NaN" with "Not Specified"
   B. **Cylinders**
      a. Replace "NaN" with "Not Specified"
   C. **Drive**
      a. Replace "NaN" with "Not Specified"
   D. **Paint Color**
      a. Replace "NaN" with "Not Specified"
   E. **Size**
      a. Replace "NaN" with "Not Specified"
   F. **Type**
      a. Replace "NaN" with "Not Specified"
   G. **Transmission**
      a. Replace "NaN" with "Not Specified"
   H. **Year**
      a. Drop all "NaN" values
   I. **Odometer**
      a. Drop all "NaN" values
5. **Outliers** – Price and Odometer have extreme values that are clearly junk data. I considered a few approaches to eliminating the junk data: (a) IQR method, (b) logical cut-off thresholds, (c) clustering to find the noise. I will use a combination of clustering and thresholds.
   A. **Removal via Clustering** – I will experiment with this, but also model with a dataset that doesn't use the clustering technique. I do not know if this will do more harm than good.
      a. Create dataframe of just Price and Odometer
      b. Create scatterplot of Price and Odometer
      c. Run HDBSCAN clustering on dataframe with Price and Odometer
      d. Color scatterplot by cluster ID and adjust parameters until noise is separated from core data
      e. Drop noise and outlier clusters from the dataset
   B. **Removal via Thresholds** – I will experiment with different thresholds, depending on how the model performs.

    a. Define minimum and maximum thresholds for Price and Odometer (and possibly Year)
    b. Drop all values above and below these thresholds

Data Encoding

1. **Categorical Variables:** There are numerous categorical variables with nominal values in this dataset. Since we replaced the missing values with "Not Specified", we no longer have any null values. We will now be able to One-Hot Encode them. In addition, there are some that represent a progressive sequence, like Cylinders, Size and Condition. For those, we will Ordinal Encode them.
    A. **Condition**
        a. Ordinal Encode
    B. **Cylinders**
        a. Ordinal Encode
    C. **Size**
        a. Ordinal Encode
    D. **Drive**
        a. One-Hot Encode (OHE)
    E. **Fuel**
        a. One-Hot Encode (OHE)
    F. **Paint Color**
        a. One-Hot Encode (OHE)
    G. **Type**
        a. One-Hot Encode (OHE)
    H. **Transmission**
        a. One-Hot Encode (OHE)
    I. **Manufacturer**
        a. One-Hot Encode (OHE)
    J. **State**
        a. One-Hot Encode (OHE)
2. **Text Input Values** – The Model column has 29k unique values. It clearly has open text input. In case the model keywords have more effect than Manufacturer, we will will try to convert them to numeric form using TF-IDF and PCA.
    A. **Model**
        a. Use Texthero to Clean text
        b. Transform the words into a vector using TF-IDF
        c. Reduce dimensions with PCA
        d. Add new columns for the 2 components that will be features in the model
        e. Drop the Model column

Data Transformation and Scaling

1. **Skewed Distributions** – Price and Odometer have right-skewed distributions with very long tails. The outlier removal approach may shorten some of these tails. I will also try doing a log conversion to see if it brings it closer to a normal distribution.
    A. **Price**
        a. Apply log transformation in pipelines
    B. **Odometer**
        a. Apply log transformation in pipelines
2. **Different Scales** – Price, Odometer and Year have different scales that will need to be normalized. I will experiment with (a) StandardScaler, (b) MinMaxScaler, and (c) RobustScaler.
    A. **Price**
        a. Apply scaler in pipelines
    B. **Odometer**
        a. Apply scaler in pipelines
    C. **Year**

a. Apply scaler in pipelines

Data Preparation

Now we will execute on our plan to address the data quality issues. Let's start by creating a copy of the dataset for data cleaning.

Data Cleaning

In [31]:
```
# Create copy of dataset for cleaning
df_clean = pd.DataFrame.copy(df)
```

In [355…
```
# Alternate dataset that retains duplicates for comparison
df_dupes = pd.DataFrame.copy(df)
```

Remove Duplicate Values

In [ ]:
```
# Drop duplicates based on the 'VIN', 'price', and 'odometer' columns
df_clean = df_clean.drop_duplicates(subset=['VIN', 'price', 'odometer'])

# Drop duplicates based on the 'VIN' alone
#df_clean = df_clean.drop_duplicates(subset=['VIN'])

# Reset index after dropping duplicates
#df = df.reset_index(drop=True)
```

In [342…
```
rows_dropped = len(df) - len(df.drop_duplicates(subset=['VIN', 'price', 'odometer']))
print("Duplicate rows dropped: ", rows_dropped)
```

```
Duplicate rows dropped:  215109
```

In [343…
```
duplicates = df[df.duplicated(subset=['VIN', 'price', 'odometer'], keep=False)]
```

In [344…
```
sorted_duplicates = duplicates.sort_values(by=['VIN', 'price', 'odometer'])
```

In [33]:
```
# Check that there are no duplicate VINs now
#vin_count_check = df_clean['VIN'].value_counts()
#df_clean['VIN'].value_counts()[vin_count_check > 1]
```

Drop Columns

In [34]:
```
# Drop columns that are no longer needed, not valuable as features
df_clean = df_clean.drop(columns=['id','VIN','title_status','region'])
```

In [356…
```
df_dupes = df_dupes.drop(columns=['id','VIN','title_status','region'])
```

Remove Zero Values

In [35]:
```
# Drop all values where price is $0 or odometer is 0
df_clean = df_clean[~((df_clean['price'] == 0) | (df_clean['odometer'] == 0))]
#df_clean = df_clean[df_clean['price'] != 0]
```

In [357…
```
df_dupes = df_dupes[~((df_dupes['price'] == 0) | (df_dupes['odometer'] == 0))]
```

Fill Categorical Null Values

In [36]:
```
# Check current state of null values
df_clean.isna().sum().loc[lambda x: x > 0]
```

Out[36]:
```
year              659
manufacturer     7950
model            2365
condition       77765
cylinders       67720
fuel             1437
odometer         1127
transmission      995
drive           52053
size           126380
type            44781
paint_color     54657
dtype: int64
```

In [37]:
```
# Replace NaNs with "Not specified" for condition, cylinders, drive, paint_color, size
fill_columns = ['condition','cylinders','drive','paint_color','size','type',
                'transmission','manufacturer','model','fuel']
for col in fill_columns:
    df_clean[col] = df_clean[col].fillna('Not specified')
```

In [358…
```
for col in fill_columns:
    df_dupes[col] = df_dupes[col].fillna('Not specified')
```

~~Impute Numeric Null Values~~

In [400…
```
# NOTE: I tried this, but decided not to use it and just drop the missing values instead
```

In [38]:
```
# Check current state of null values
#df_clean.isna().sum().loc[lambda x: x > 0]
```

In [39]:
```
# Convert the 0 values to NaN values so we can process them the same
#df_clean['odometer'] = df_clean['odometer'].replace(0, np.nan)
```

In [40]:
```
# Import the libraries we need to scale and impute the missing values
# from sklearn.impute import KNNImputer
# from sklearn.preprocessing import MinMaxScaler

# cols_to_impute = ['price', 'odometer', 'year']

# # Scale the columns
# scaler = MinMaxScaler()
# df_scaled = pd.DataFrame(scaler.fit_transform(df_clean[cols_to_impute]), columns=cols_to_impute)
```

In [41]:
```
# Impute the missing values
# imputer = KNNImputer(n_neighbors=2)
# df_imputed_scaled = pd.DataFrame(imputer.fit_transform(df_scaled), columns=cols_to_impute)
```

In [42]:
```
# Reverse the scaling
# df_imputed = pd.DataFrame(scaler.inverse_transform(df_imputed_scaled), columns=cols_to_impute)
```

In [43]:
```
# Round the values to the nearest integer and convert to int
# df_imputed['year'] = df_imputed['year'].round().astype(int)
# df_imputed['odometer'] = df_imputed['odometer'].round().astype(int)
# df_imputed['price'] = df_imputed['price'].round().astype(int)
```

In [44]:
```
# Add the missing values back to df_clean
# df_clean[cols_to_impute] = df_imputed.values
```

In [45]:
```
# Confirm null values are removed
# df_clean.isna().sum().loc[lambda x: x > 0]
```

Drop Remaining Null Values

In [46]:
```
# Check current state of null values
df_clean.isna().sum().loc[lambda x: x > 0]
```

Out[46]:
```
year         659
odometer    1127
dtype: int64
```

In [47]:
```
# Drop remaining null values
df_clean = df_clean.dropna()
```

In [359…
```
df_dupes = df_dupes.dropna()
```

~~Convert Data Types~~

In [48]:
```
# NOTE: This caused problems with calculations later
# Now that NaNs are removed, let's convert to the best datatype
#df_clean = df_clean.convert_dtypes()
```

In [49]:
```
#df_clean.info()
```

Remove Outliers

I will use a combination of thresholds and clustering to identify and remove outliers.

In [50]:
```
# Create a new dataframe for working on outlier renewal
df_outliers = pd.DataFrame.copy(df_clean)
```

In [51]:
```
# Create a new column to store outlier flags
df_outliers['outlier'] = False
```

In [52]:
```
# Log transform the data
df_outliers[['price_log','odometer_log','year_log']] = np.log1p(df_outliers[['price','odometer','year']
```

~~IQR~~

In [53]:
```
# Q1 = df['price'].quantile(0.25)
# Q3 = df['price'].quantile(0.75)
# IQR = Q3 - Q1
# df_clean = df[~((df['price'] < (Q1 - 1.5 * IQR)) | (df['price'] > (Q3 + 1.5 * IQR)))]
```

Flag Outliers Based on Thresholds

I started with these thresholds, but later ended up exploring alternate thresholds to exclude more of the data.

In [54]:
```
# Define outlier thresholds
upper_thresholds = {'price': 500000, 'year': 2023, 'odometer': 900000}
lower_thresholds = {'price': 2, 'year': 1900, 'odometer': 2}
```

In [55]:
```
# Iterate over columns and set outlier flag
outliers_counts = {}
for col in num_columns:
    upper_bound = upper_thresholds[col]
    lower_bound = lower_thresholds[col]
    is_outlier = (df_outliers[col] > upper_bound) | (df_outliers[col] < lower_bound)
    df_outliers[f'{col}_outlier'] = is_outlier
    df_outliers['outlier'] = df_outliers['outlier'] | is_outlier
    outliers_counts[col] = is_outlier.sum()

# Print the count of outliers for each column
for col, count in outliers_counts.items():
    print(f"{col} has {count} outliers.")

# Print the total count of outliers
print(f"Total rows marked as outliers: {df_outliers['outlier'].sum()}")
```

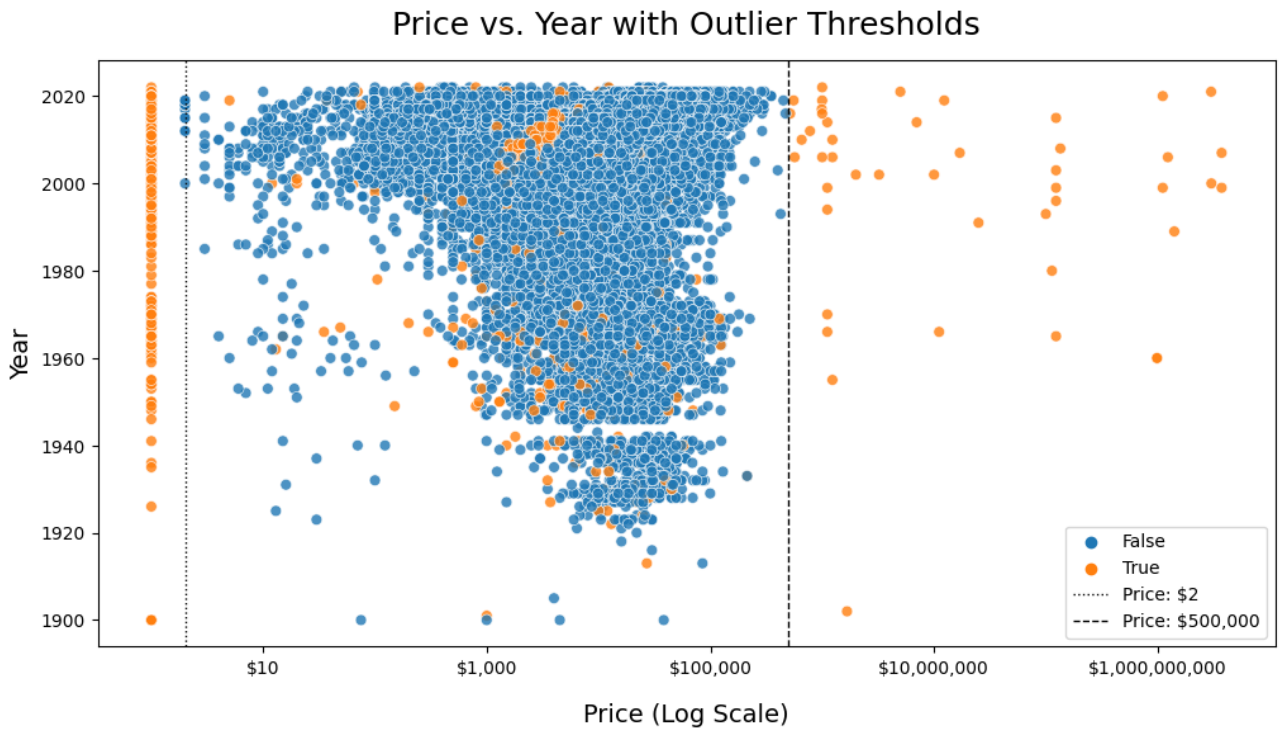
```
price has 588 outliers.
year has 0 outliers.
odometer has 1366 outliers.
Total rows marked as outliers: 1939
```

Examine Outliers on Plots

In [56]:
```
my.plot_3d(df=df_outliers, x='price_log', y='odometer_log', z='year', color='outlier')
```

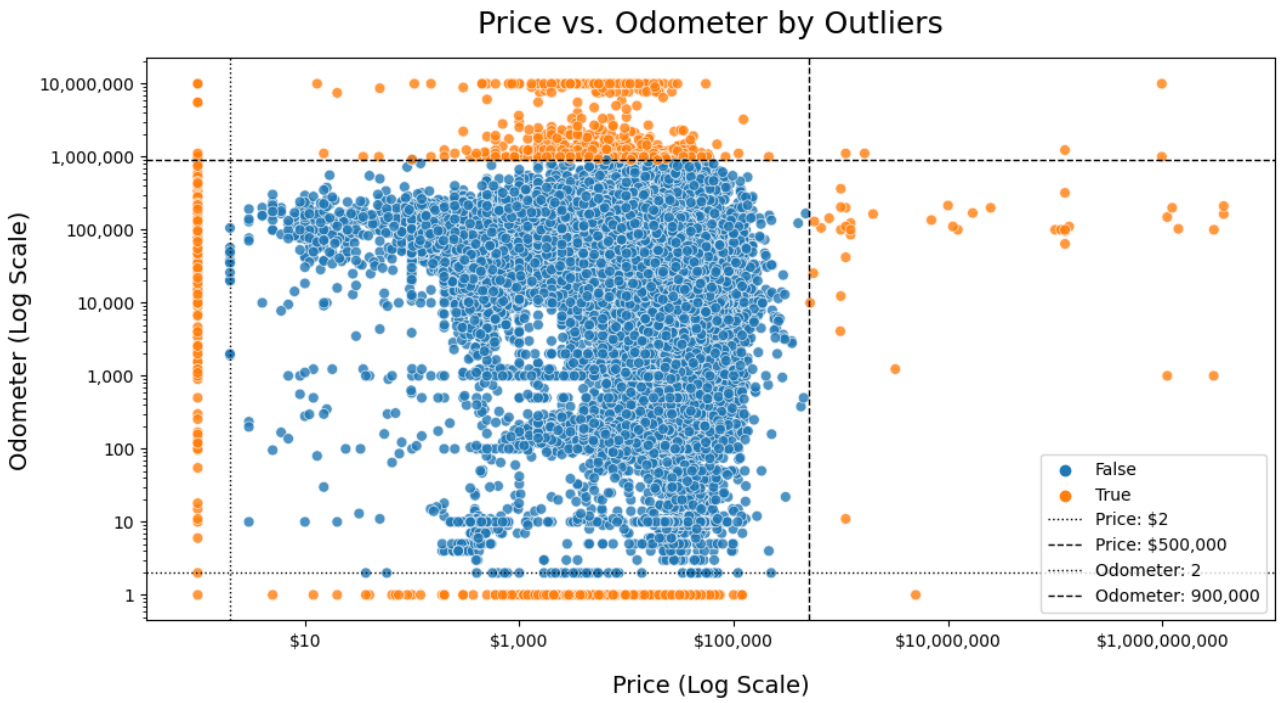
Observation: The 3D plot shows the values outside of the thresholds as potential outliers in orange. We can see how most of these are not part of the central cluster in blue. But there are still are some "outlier" data points floating detached from the main body. Perhaps we can try clustering to filter out the noise.

In [57]:
```
plt.figure(figsize=(12,6))
plt.title('Price vs. Year with Outlier Thresholds', fontsize=18, pad=15)
sns.scatterplot(data=df_outliers, x='price', y='year', s=40, alpha=0.8, hue='outlier')
plt.xscale('log')
#plt.yscale('log')
plt.xlabel('Price (Log Scale)', fontsize=14, labelpad=15)
plt.ylabel('Year', fontsize=14)
plt.gca().xaxis.set_major_formatter(FuncFormatter(my.thousand_dollars))
#plt.gca().yaxis.set_major_formatter(FuncFormatter(my.thousands))
plt.axvline(x = 2, color = 'black', linestyle=':', lw=1, label='Price: $2')
plt.axvline(x = 500000, color = 'black', linestyle='--', lw=1, label='Price: $500,000')
#plt.axhline(y = 25, color = 'black', linestyle=':', lw=1, label='Odometer: 25')
#plt.axhline(y = 2000000, color = 'black', linestyle='--', lw=1, label='Odometer: 2,000,000')
plt.legend(loc='lower right')
plt.show()
```



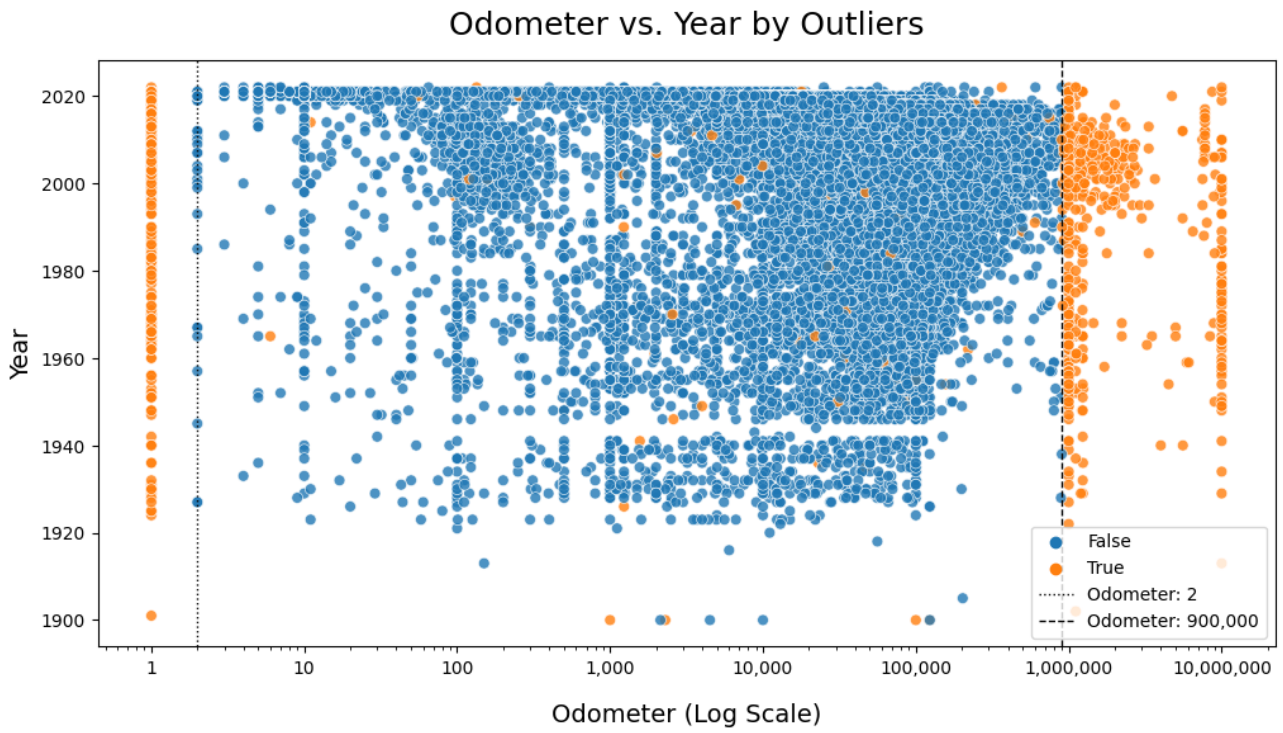
Observation: The scatter plot shows the outlier threshold cut-off points, with the flagged outliers in orange. Orange values will be removed. Hopefully that helps a stronger signal to come out of the blue cluster.

In [58]:
```
plt.figure(figsize=(12,6))
plt.title('Price vs. Odometer by Outliers', fontsize=18, pad=15)
sns.scatterplot(data=df_outliers, x='price', y='odometer', s=40, alpha=0.8, hue='outlier')
plt.xscale('log')
plt.yscale('log')
plt.xlabel('Price (Log Scale)', fontsize=14, labelpad=15)
plt.ylabel('Odometer (Log Scale)', fontsize=14)
plt.gca().xaxis.set_major_formatter(FuncFormatter(my.thousand_dollars))
plt.gca().yaxis.set_major_formatter(FuncFormatter(my.thousands))
plt.axvline(x = 2, color = 'black', linestyle=':', lw=1, label='Price: $2')
plt.axvline(x = 500000, color = 'black', linestyle='--', lw=1, label='Price: $500,000')
plt.axhline(y = 2, color = 'black', linestyle=':', lw=1, label='Odometer: 2')
plt.axhline(y = 900000, color = 'black', linestyle='--', lw=1, label='Odometer: 900,000')
plt.legend(loc='lower right')
plt.show()
```



Observation: The scatter plot shows the outlier threshold cut-off points, with the flagged outliers in orange. Orange values will be removed. Hopefully that helps a stronger signal to come out of the blue cluster.

In [59]:
```
plt.figure(figsize=(12,6))
plt.title('Odometer vs. Year by Outliers', fontsize=18, pad=15)
sns.scatterplot(data=df_outliers, x='odometer', y='year', s=40, alpha=0.8, hue='outlier')
plt.xscale('log')
#plt.yscale('log')
plt.xlabel('Odometer (Log Scale)', fontsize=14, labelpad=15)
plt.ylabel('Year', fontsize=14)
plt.gca().xaxis.set_major_formatter(FuncFormatter(my.thousands))
#plt.gca().yaxis.set_major_formatter(FuncFormatter(my.thousands))
#plt.axvline(x = 5, color = 'black', linestyle=':', lw=1, label='Price: $5')
#plt.axvline(x = 500000, color = 'black', linestyle='--', lw=1, label='Price: $500,000')
plt.axvline(x = 2, color = 'black', linestyle=':', lw=1, label='Odometer: 2')
plt.axvline(x = 900000, color = 'black', linestyle='--', lw=1, label='Odometer: 900,000')
plt.legend(loc='lower right')
plt.show()
```



Observation: The scatter plot shows the outlier threshold cut-off points, with the flagged outliers in orange. Orange values will be removed. Hopefully that helps a stronger signal to come out of the blue cluster.

Flag Outliers Based on Clustering

The thresholds will catch some extreme high and low values, but there still were a lot of random noise in the previous plots. Let's see if HDBSCAN clustering can select that noise, and we can drop them as well.

In [60]:
```
# Import the libraries needed for scaling and clustering
from sklearn.preprocessing import MinMaxScaler
import hdbscan
```

In [61]:
```
# Copy the dataframe for clustering work
df_cluster = pd.DataFrame.copy(df_outliers)
```

In [62]:
```
# Scale the log-transformed data
scaler = MinMaxScaler()
df_cluster_scaled = pd.DataFrame.copy(df_cluster)
df_cluster_scaled[['price_log','odometer_log','year_log']] = scaler.fit_transform(df_cluster_scaled[['p
```

In [63]:
```
# Configure the clustering parameters
clusterer = hdbscan.HDBSCAN(min_cluster_size=100, min_samples=20, gen_min_span_tree=True,
                            cluster_selection_epsilon=0.05)
cluster_labels = clusterer.fit_predict(df_cluster_scaled[['price_log','odometer_log','year_log']])
df_cluster_scaled['cluster'] = cluster_labels
df_cluster['cluster'] = cluster_labels

# Results
print(f"Number of clusters found: {len(set(cluster_labels)) - (1 if -1 in cluster_labels else 0)}")
print(f"Number of noise points: {(cluster_labels == -1).sum()}")
```

```
Number of clusters found: 2
Number of noise points: 1329
```

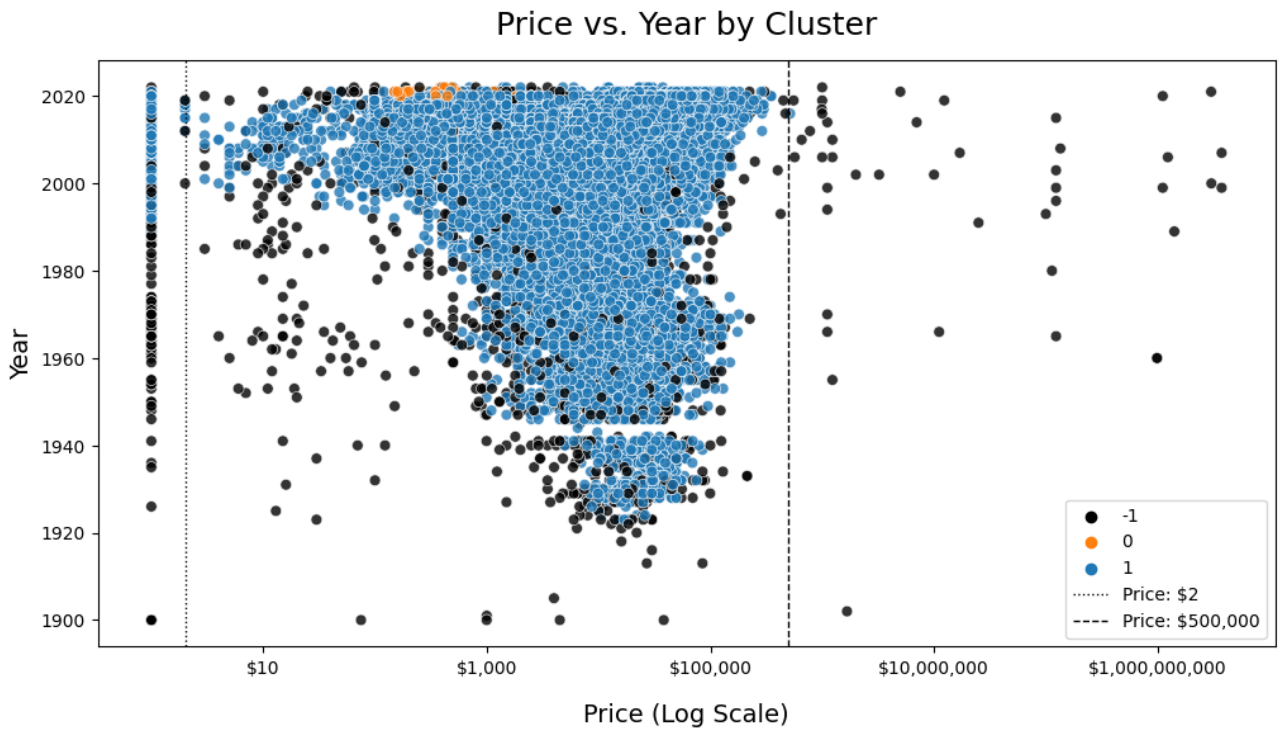
Examine Clusters on Plots

In [64]:
```
# Color map for clustering
color_map_cluster_value = {-1: 'black', 0: px.colors.qualitative.D3[1], 1: px.colors.qualitative.D3[0],
                           2: px.colors.qualitative.D3[2], 3: px.colors.qualitative.D3[3], 4: px.colors.qu
                           5: px.colors.qualitative.D3[5], 6: px.colors.qualitative.D3[6]}
```

In [65]:
```
my.plot_3d(df=df_cluster_scaled, x='price_log', y='odometer_log', z='year', color='cluster', color_map=
```

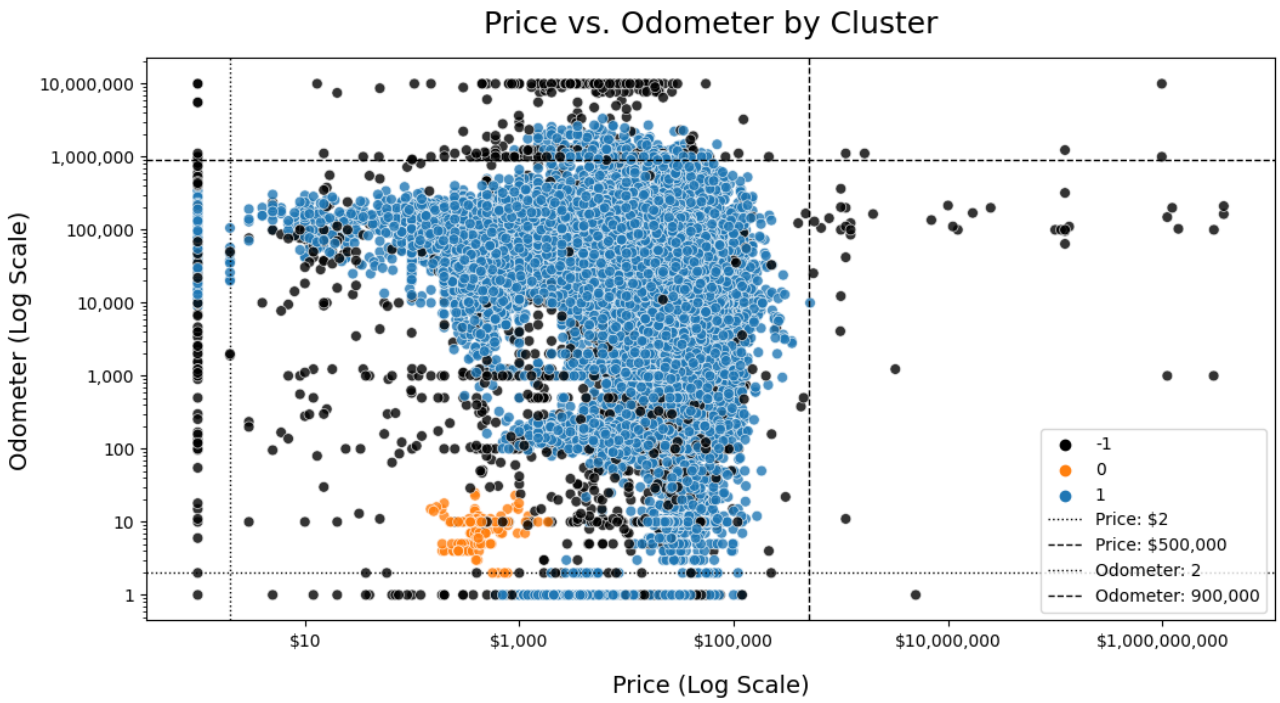
Observation: Here we see the main central cluster in yellow, with a number of noise points floating outside of it in the dark blue. There is a smaller red cluster that represents recent years, with low odometer readings and lower prices than the main cluster. I will experiment with dropping this as well.

In [66]:
```
plt.figure(figsize=(12,6))
plt.title('Price vs. Year by Cluster', fontsize=18, pad=15)
sns.scatterplot(data=df_cluster, x='price', y='year', s=40, alpha=0.8, hue='cluster', palette=color_map
plt.xscale('log')
#plt.yscale('log')
plt.xlabel('Price (Log Scale)', fontsize=14, labelpad=15)
plt.ylabel('Year', fontsize=14)
plt.gca().xaxis.set_major_formatter(FuncFormatter(my.thousand_dollars))
#plt.gca().yaxis.set_major_formatter(FuncFormatter(my.thousands))
plt.axvline(x = 2, color = 'black', linestyle=':', lw=1, label='Price: $2')
plt.axvline(x = 500000, color = 'black', linestyle='--', lw=1, label='Price: $500,000')
#plt.axhline(y = 25, color = 'black', linestyle=':', lw=1, label='Odometer: 25')
#plt.axhline(y = 2000000, color = 'black', linestyle='--', lw=1, label='Odometer: 2,000,000')
plt.legend(loc='lower right')
plt.show()
```



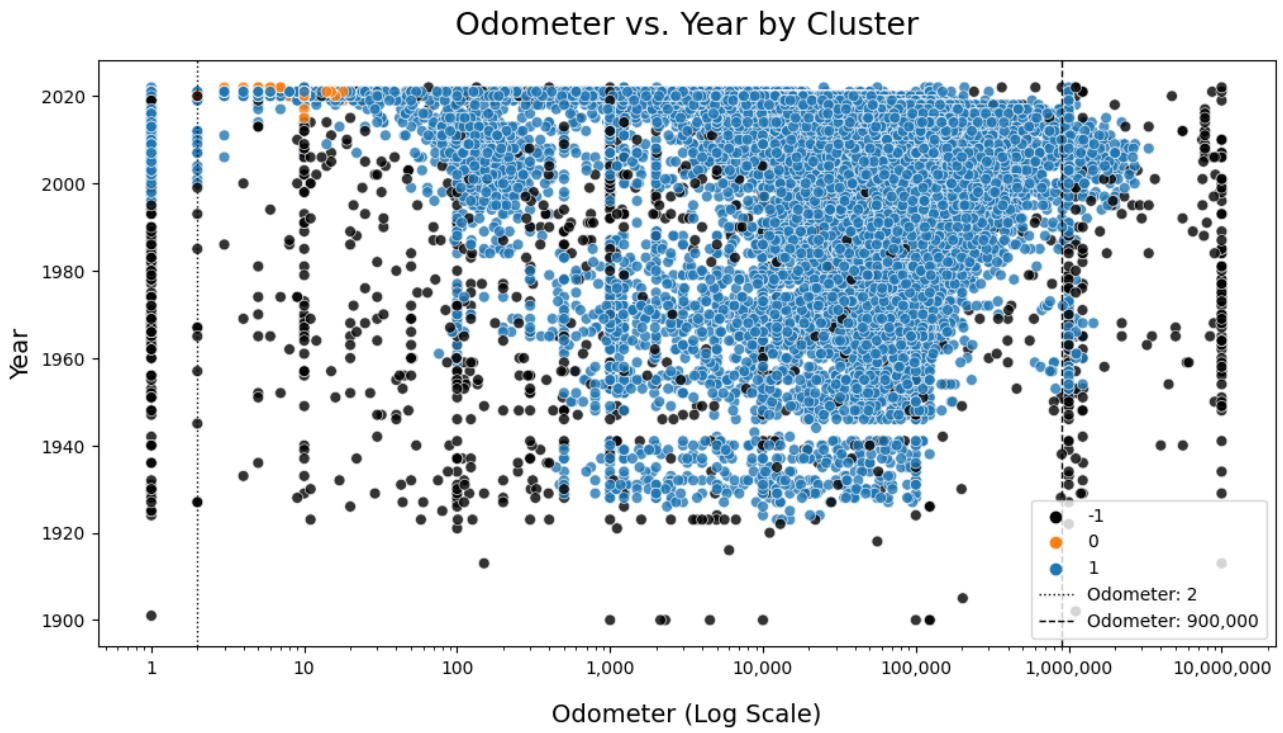
Observation: The scatter plot shows the results of the clustering. The blue points represent the main cluster that will be retained. The black points are "noise" and will be dropped. The smaller orange cluster will be dropped as well. The hope is this will increase the signal in the model, but we can revert if it doesn't help.

In [67]:
```
plt.figure(figsize=(12,6))
plt.title('Price vs. Odometer by Cluster', fontsize=18, pad=15)
sns.scatterplot(data=df_cluster, x='price', y='odometer', s=40, alpha=0.8, hue='cluster', palette=color
plt.xscale('log')
plt.yscale('log')
plt.xlabel('Price (Log Scale)', fontsize=14, labelpad=15)
plt.ylabel('Odometer (Log Scale)', fontsize=14)
plt.gca().xaxis.set_major_formatter(FuncFormatter(my.thousand_dollars))
plt.gca().yaxis.set_major_formatter(FuncFormatter(my.thousands))
plt.axvline(x = 2, color = 'black', linestyle=':', lw=1, label='Price: $2')
plt.axvline(x = 500000, color = 'black', linestyle='--', lw=1, label='Price: $500,000')
plt.axhline(y = 2, color = 'black', linestyle=':', lw=1, label='Odometer: 2')
plt.axhline(y = 900000, color = 'black', linestyle='--', lw=1, label='Odometer: 900,000')
plt.legend(loc='lower right')
plt.show()
```



Observation: The scatter plot shows the results of the clustering. The blue points represent the main cluster that will be retained. The black points are "noise" and will be dropped. The smaller orange cluster will be dropped as well. The hope is this will increase the signal in the model, but we can revert if it doesn't help.

In [68]:
```
plt.figure(figsize=(12,6))
plt.title('Odometer vs. Year by Cluster', fontsize=18, pad=15)
sns.scatterplot(data=df_cluster, x='odometer', y='year', s=40, alpha=0.8, hue='cluster', palette=color_
plt.xscale('log')
#plt.yscale('log')
plt.xlabel('Odometer (Log Scale)', fontsize=14, labelpad=15)
plt.ylabel('Year', fontsize=14)
plt.gca().xaxis.set_major_formatter(FuncFormatter(my.thousands))
#plt.gca().yaxis.set_major_formatter(FuncFormatter(my.thousands))
#plt.axvline(x = 5, color = 'black', linestyle=':', lw=1, label='Price: $5')
#plt.axvline(x = 500000, color = 'black', linestyle='--', lw=1, label='Price: $500,000')
plt.axvline(x = 2, color = 'black', linestyle=':', lw=1, label='Odometer: 2')
plt.axvline(x = 900000, color = 'black', linestyle='--', lw=1, label='Odometer: 900,000')
plt.legend(loc='lower right')
plt.show()
```



Observation: The scatter plot shows the results of the clustering. The blue points represent the main cluster that will be retained. The black points are "noise" and will be dropped. The smaller orange cluster will be dropped as well. The hope is this will increase the signal in the model, but we can revert if it doesn't help.

Remove Outliers by Cluster and Threshold

In [69]:
```
# Create dataframe that will not have outliers
df_clean_no_outliers = pd.DataFrame.copy(df_cluster)
```

In [70]:
```
# Drop all clusters except cluster 1, the main cluster
df_clean_no_outliers = df_clean_no_outliers[df_clean_no_outliers['cluster'] == 1]
```

In [71]:
```
# Drop all values flagged as outliers by thresholds
df_clean_no_outliers = df_clean_no_outliers[df_clean_no_outliers['outlier'] == False]
```

In [72]:
```
df_clean_no_outliers.drop(columns=['price_outlier','year_outlier','odometer_outlier','outlier','cluste
```

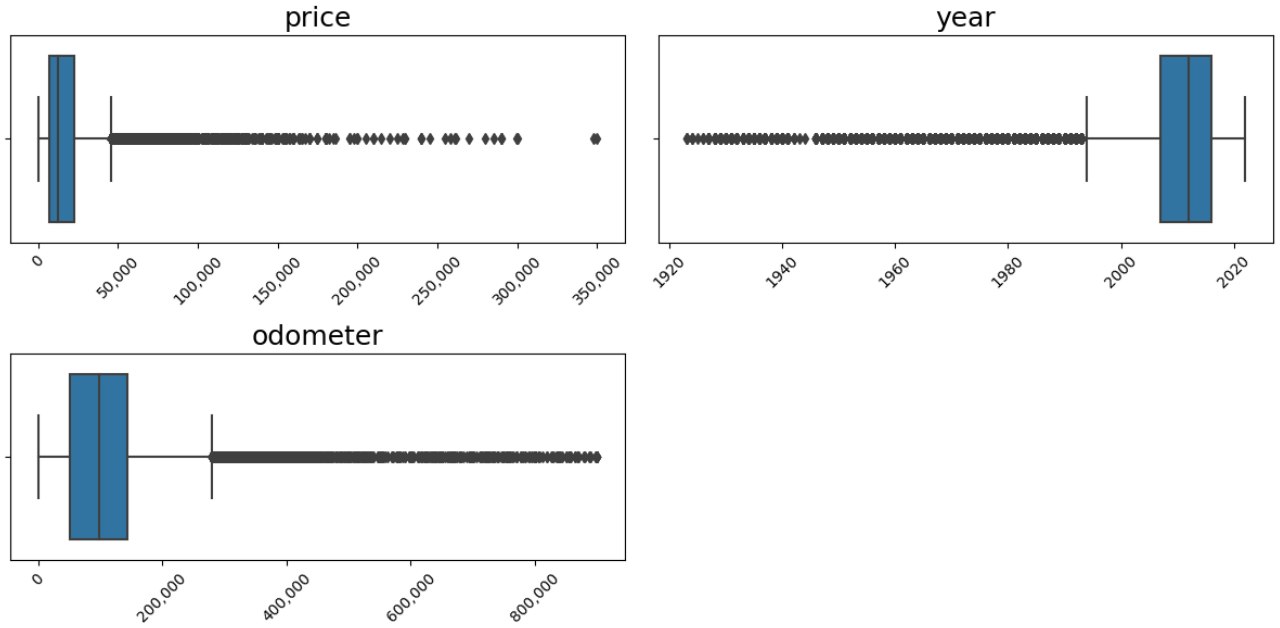
In [73]:
```
#df_clean_no_outliers.sample(10)
#df_clean_no_outliers.info()
df_clean_no_outliers.describe()
```

Out[73]:

	price	year	odometer	price_log	odometer_log	year_log
count	191075.00	191075.00	191075.00	191075.00	191075.00	191075.00
mean	16520.84	2010.16	103346.10	9.32	11.21	7.61
std	14450.65	9.61	68209.92	1.03	1.12	0.00
min	2.00	1923.00	2.00	1.10	1.10	7.56
25%	6750.00	2007.00	51806.50	8.82	10.86	7.60
50%	12800.00	2012.00	97751.00	9.46	11.49	7.61
75%	22380.50	2016.00	142929.00	10.02	11.87	7.61
max	349999.00	2022.00	900000.00	12.77	13.71	7.61

Review Plot of Data with No Outliers

In [74]:
```
# Plot box plots to see how outliers have improved
plt.figure(figsize=(12,6))
for i in range(len(num_columns)):
    plt.subplot(2,2,i+1)
    plt.title(num_columns[i], fontsize=18)
    sns.boxplot(x=num_columns[i], data=df_clean_no_outliers)
    plt.xlabel('')
    plt.xticks(rotation=45)
    if num_columns[i] != 'year':
        plt.gca().xaxis.set_major_formatter(FuncFormatter(my.thousands))
plt.tight_layout()
plt.show()
```

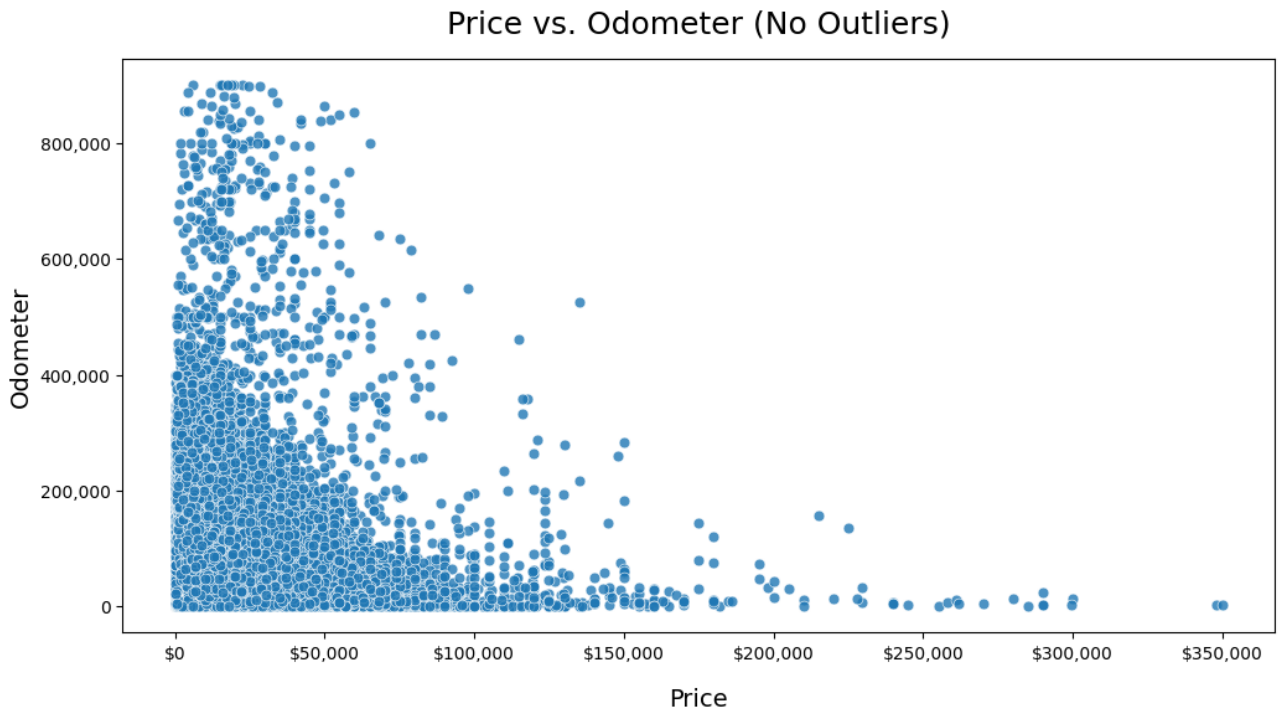


Observation: Here we can see the tails of the box plots are significantly improved. There still is a long tail, so our outlier removal was conservative. But the most extreme values are gone now.

In [75]:
```
my.plot_3d(df=df_clean_no_outliers, x='price_log', y='odometer_log', z='year_log')
```

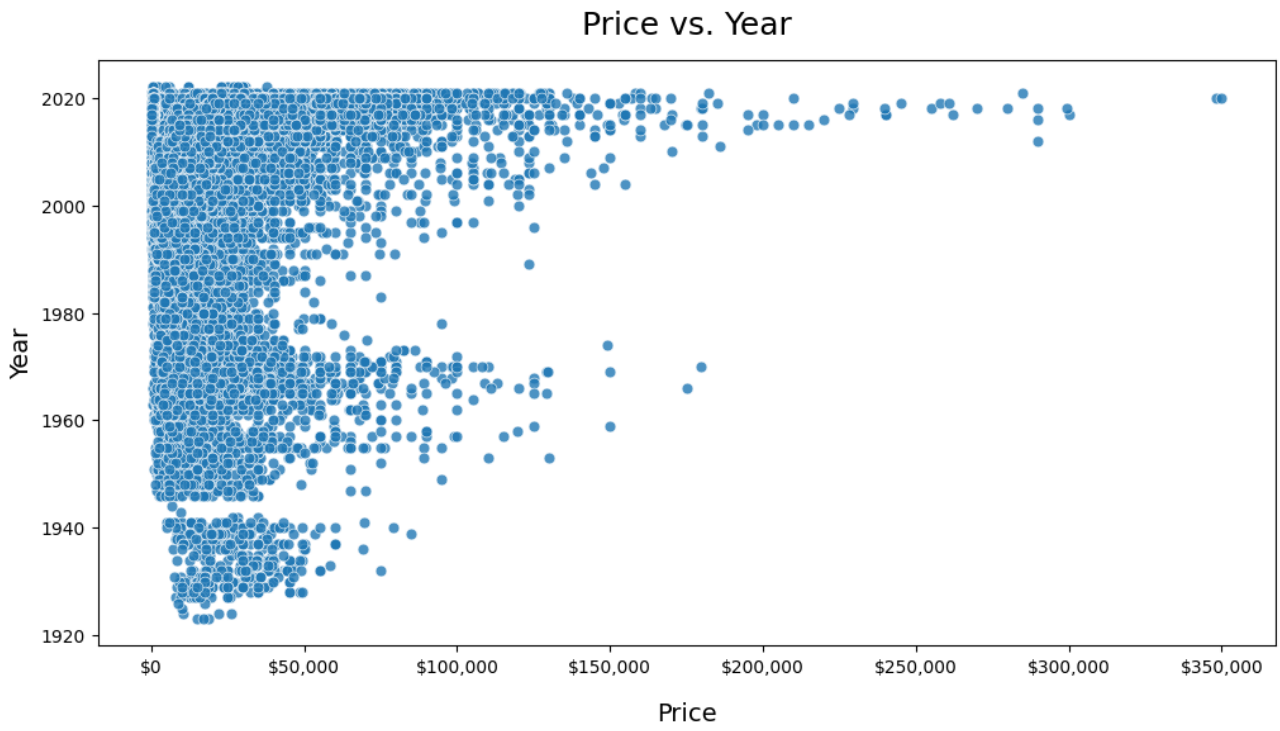
Observation: This is our final dataset after cleaning and dropping outliers and noise points. It's shown in log scale. There is a distinct pyramid shape with slight curves from the apex at high price, high odometer, and recent years.

In [76]:
```
plt.figure(figsize=(12,6))
plt.title('Price vs. Odometer (No Outliers)', fontsize=18, pad=15)
sns.scatterplot(data=df_clean_no_outliers, x='price', y='odometer', s=40, alpha=0.8)
plt.xlabel('Price', fontsize=14, labelpad=15)
plt.ylabel('Odometer', fontsize=14)
plt.gca().xaxis.set_major_formatter(FuncFormatter(my.thousand_dollars))
plt.gca().yaxis.set_major_formatter(FuncFormatter(my.thousands))
plt.show()
```



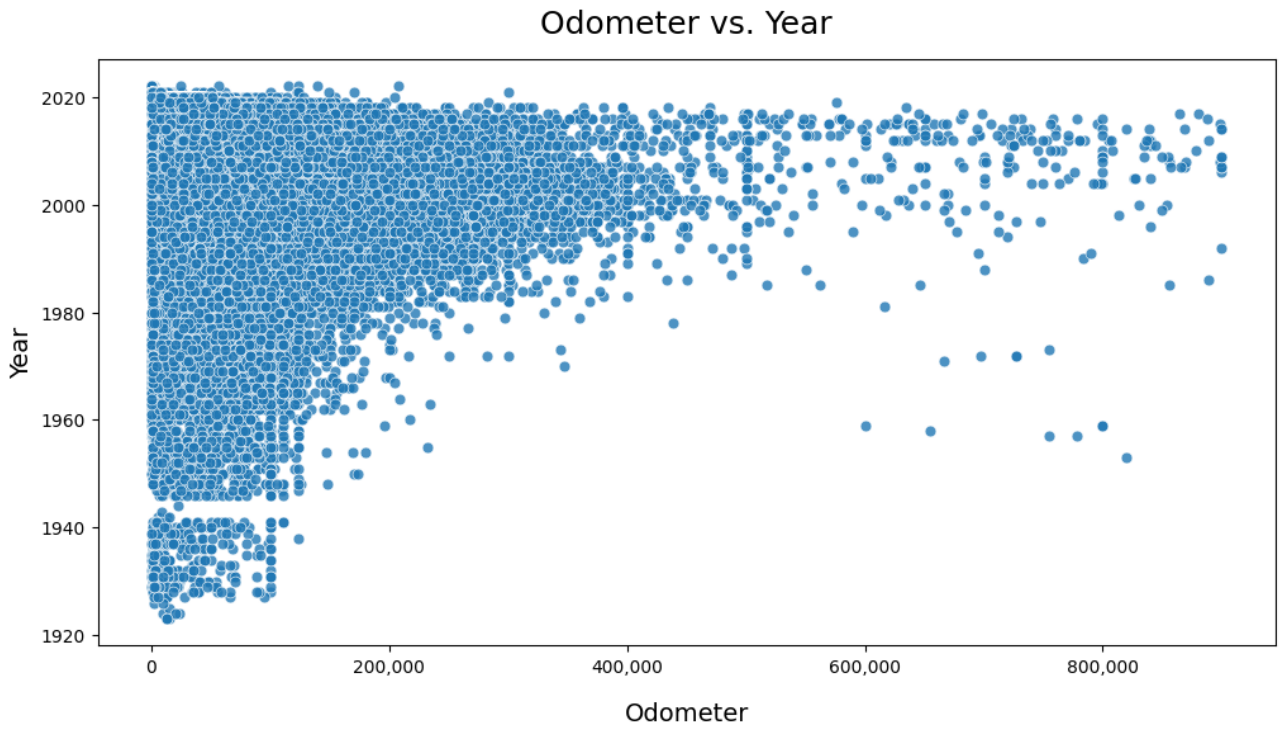
Insight: This shows price vs. odometer in log scale with the extreme outliers and noise removed. We can start to discern a relationship between these two variables here. This looks like an exponential drop-off from high odometer / low price, to low odometer and high price. Logically this makes sense. It doesn't look linear, so maybe a log transformation will need to be applied.

In [77]:
```
plt.figure(figsize=(12,6))
plt.title('Price vs. Year', fontsize=18, pad=15)
sns.scatterplot(data=df_clean_no_outliers, x='price', y='year', s=40, alpha=0.8)
plt.xlabel('Price', fontsize=14, labelpad=15)
plt.ylabel('Year', fontsize=14)
plt.gca().xaxis.set_major_formatter(FuncFormatter(my.thousand_dollars))
plt.show()
```



Insight: This shows price vs. year. You can see a slight upward diagonal shape, suggesting a higher price for more recent years. But there is also a second grouping here, where there's higher price for older cars. This could be reflecting the higher price for "classic cars" that have value because of their antiquity. There seems to be a gap in cars made during the early 1940s, is that due to the war? Were there not many cars made at that time? Or is this just junk data at the bottom.

In [78]:
```
plt.figure(figsize=(12,6))
plt.title('Odometer vs. Year', fontsize=18, pad=15)
sns.scatterplot(data=df_clean_no_outliers, x='odometer', y='year', s=40, alpha=0.8)
plt.xlabel('Odometer', fontsize=14, labelpad=15)
plt.ylabel('Year', fontsize=14)
plt.gca().xaxis.set_major_formatter(FuncFormatter(my.thousands))
plt.show()
```



Insight: This shows odometer vs. year. I do not understand this chart. Logically, you would expect a drop from the upper left to the lower right – newer cars should have less mileage, older cars should have more. You see a little bit of that in the upper half of the chart. But these are advertised prices, and reflect a decision of the person posting the listing. It could be that this chart is showing human psychology, where people are posting lower mileage than actual. Or it could be that older "antique" cars are not being driven much.

Clean Up the Index

In [79]:
```
# Reset the index since we dropped a bunch of rows
df_clean_no_outliers = df_clean_no_outliers.reset_index(drop=True)
```

In [80]:
```
df_clean_no_outliers.info()
```

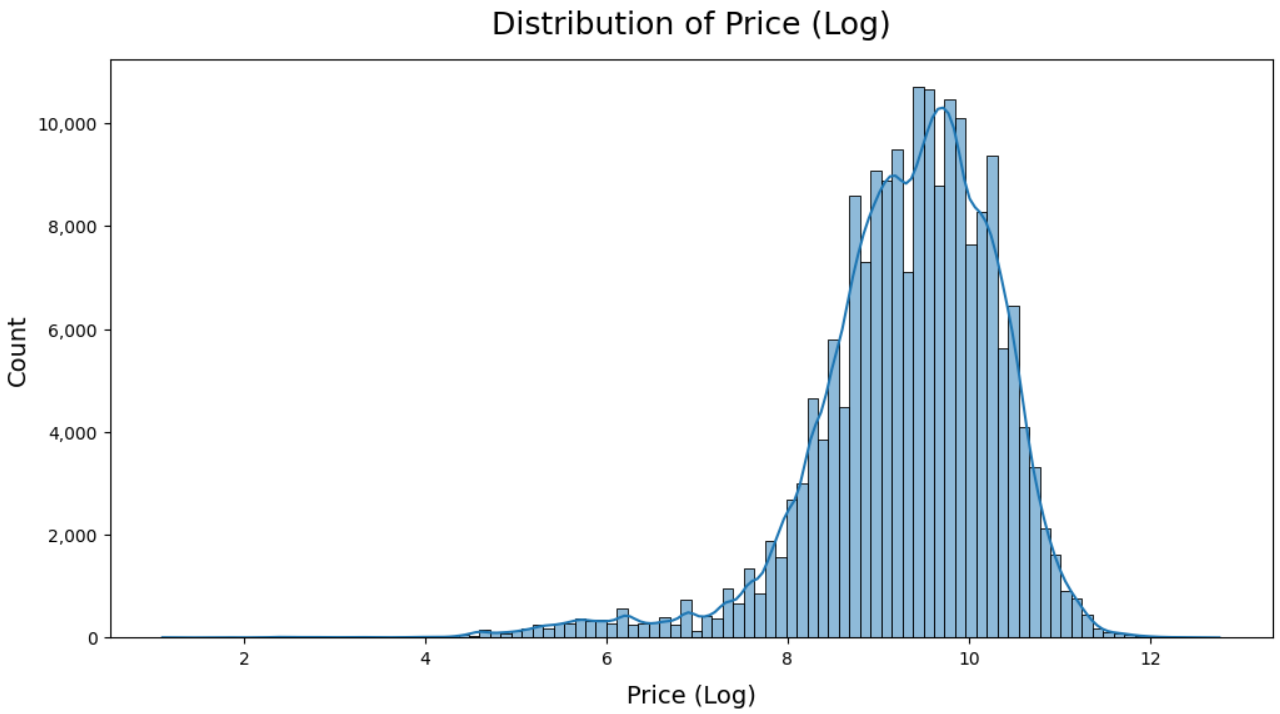
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 191075 entries, 0 to 191074
Data columns (total 17 columns):
 #   Column         Non-Null Count   Dtype
---  ------         --------------   -----
 0   price          191075 non-null  int64
 1   year           191075 non-null  float64
 2   manufacturer   191075 non-null  object
 3   model          191075 non-null  object
 4   condition      191075 non-null  object
 5   cylinders      191075 non-null  object
 6   fuel           191075 non-null  object
 7   odometer       191075 non-null  float64
 8   transmission   191075 non-null  object
 9   drive          191075 non-null  object
 10  size           191075 non-null  object
 11  type           191075 non-null  object
 12  paint_color    191075 non-null  object
 13  state          191075 non-null  object
 14  price_log      191075 non-null  float64
 15  odometer_log   191075 non-null  float64
 16  year_log       191075 non-null  float64
dtypes: float64(5), int64(1), object(11)
memory usage: 24.8+ MB
```

Results of Data Cleaning

In [81]:
```
# Evaluate impact on data set
data_retention = len(df_clean_no_outliers) / len(df) * 100
print(f"Amount of original data set retained after cleaning: {data_retention:.2f}%")
```

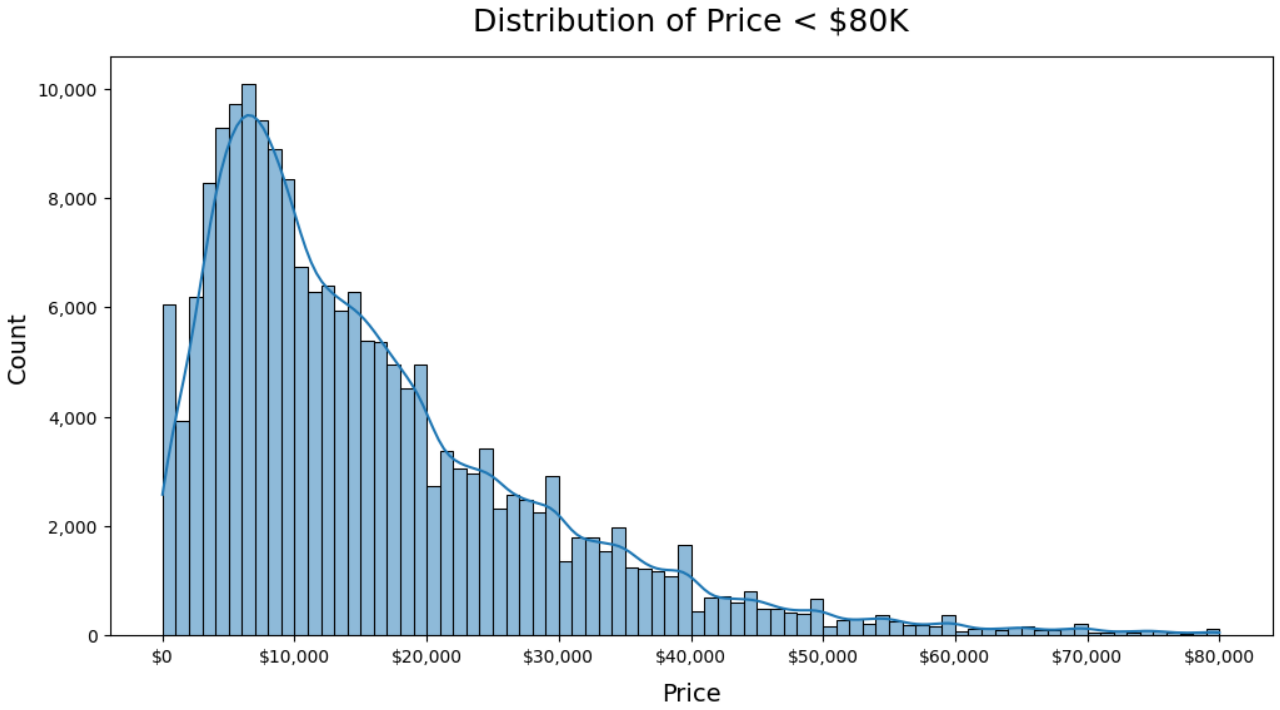
```
Amount of original data set retained after cleaning: 44.76%
```

In [82]:
```
plt.figure(figsize=(12,6))
sns.histplot(x='price_log', data=df_clean_no_outliers, kde=True, bins=100)
plt.title('Distribution of Price (Log)', fontsize=18, pad=15)
plt.ylabel('Count', fontsize=14)
plt.xlabel('Price (Log)', fontsize=14, labelpad=10)
#plt.gca().xaxis.set_major_formatter(FuncFormatter(my.thousand_dollars))
plt.gca().yaxis.set_major_formatter(FuncFormatter(my.thousands))
plt.show()
```



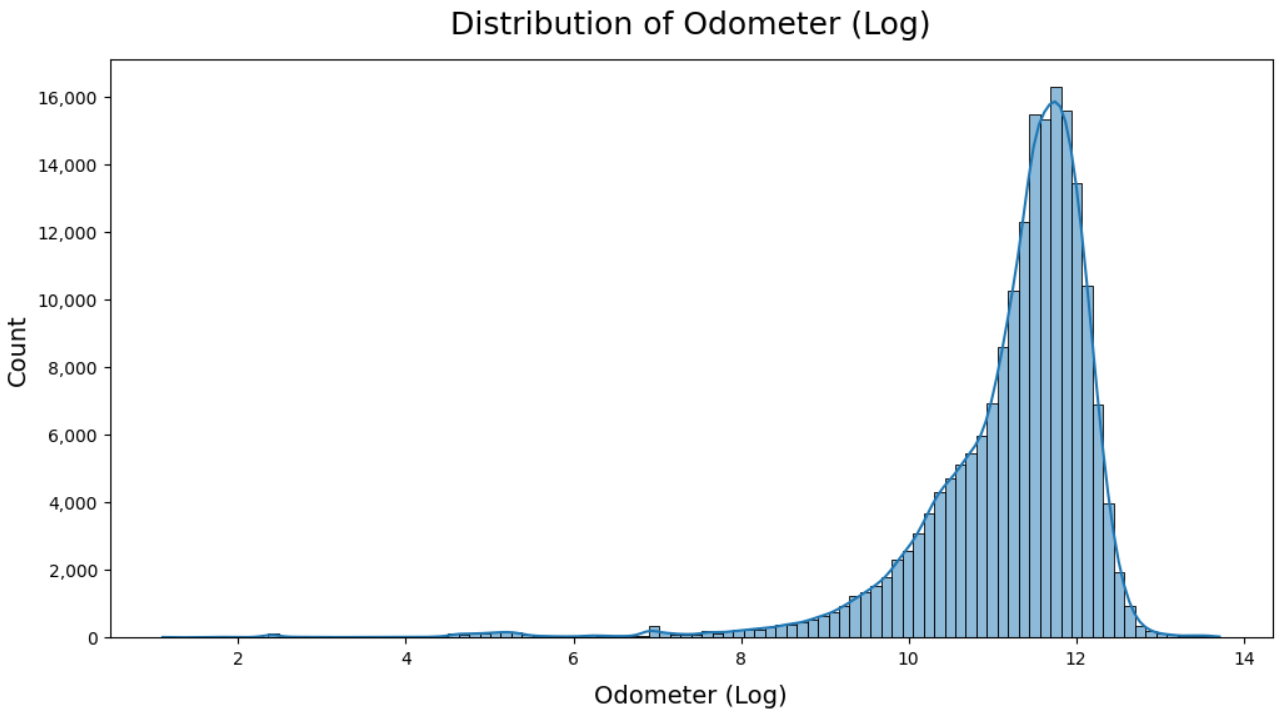
Observation: Price still has a significant right skew, so it has to be viewed in log. With this log transformation, it resembles a roughly normal distribution. This is a good result.

In [83]:
```
plt.figure(figsize=(12,6))
sns.histplot(x='price', kde=True, bins=80, data=df_clean_no_outliers.query("price < 80000"))
plt.title('Distribution of Price < $80K', fontsize=18, pad=15)
plt.ylabel('Count', fontsize=14)
plt.xlabel('Price', fontsize=14, labelpad=10)
plt.gca().xaxis.set_major_formatter(FuncFormatter(my.thousand_dollars))
plt.gca().yaxis.set_major_formatter(FuncFormatter(my.thousands))
plt.show()
```



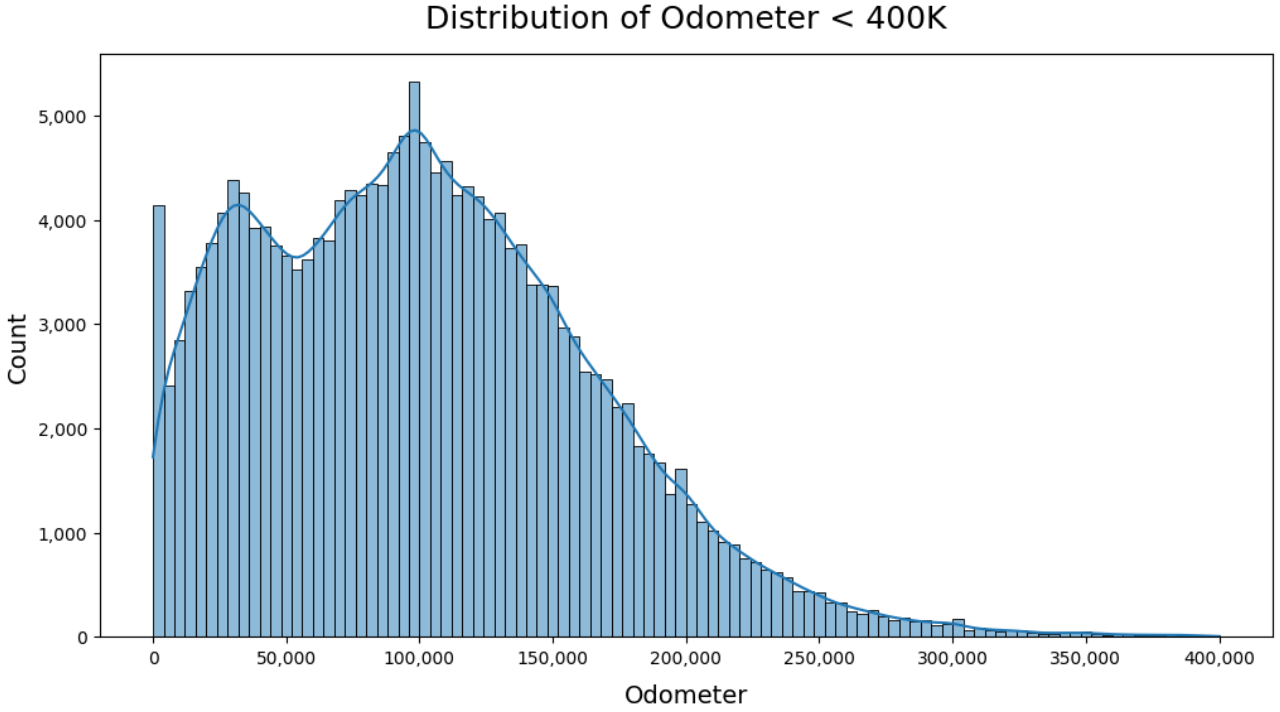
Observation: This is a zoomed in view of Price below $80k. The majority of the price values are in this range. You can see the skewed distribution, but the tail extends much further to the right.

In [88]:
```
plt.figure(figsize=(12,6))
sns.histplot(x='odometer_log', kde=True, bins=100, data=df_clean_no_outliers)
plt.title('Distribution of Odometer (Log)', fontsize=18, pad=15)
plt.ylabel('Count', fontsize=14)
plt.xlabel('Odometer (Log)', fontsize=14, labelpad=10)
plt.gca().yaxis.set_major_formatter(FuncFormatter(my.thousands))
plt.show()
```



Observation: This is the log transformation of Odometer. The result is not as close to a normal distribution as we saw with Price.

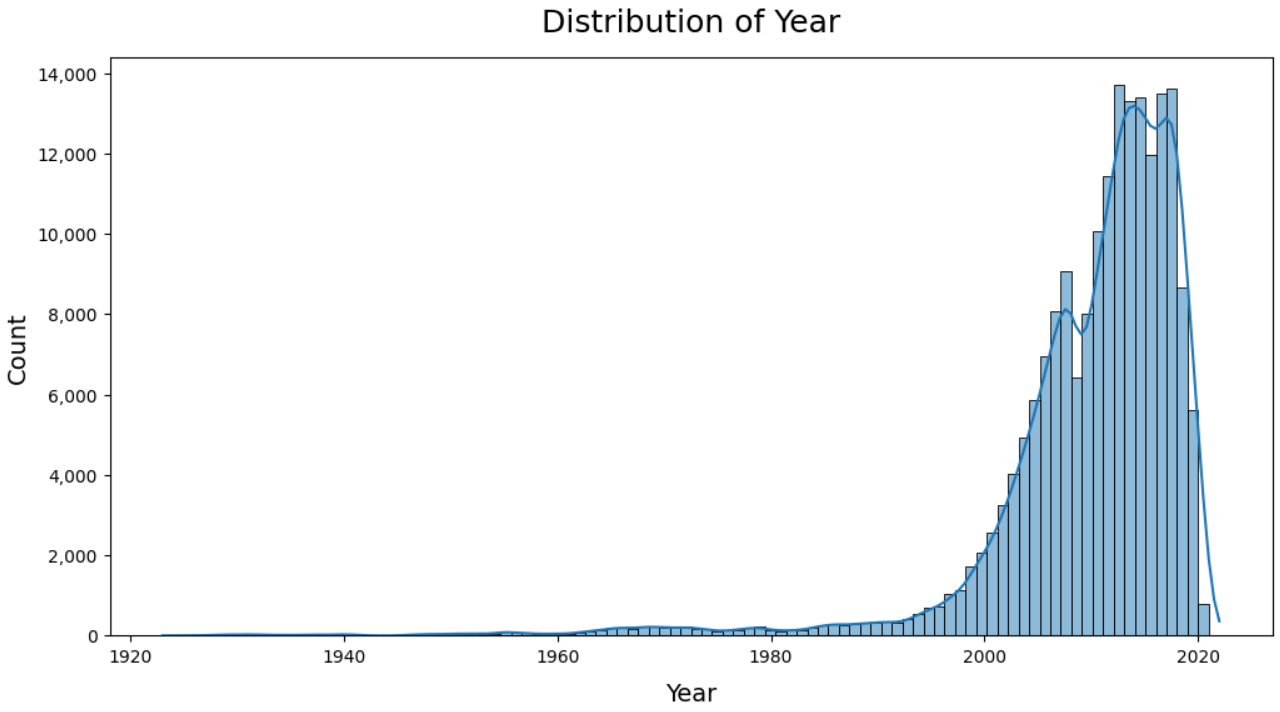
In [89]:
```
plt.figure(figsize=(12,6))
sns.histplot(x='odometer', kde=True, bins=100, data=df_clean_no_outliers.query("odometer < 400000"))
plt.title('Distribution of Odometer < 400K', fontsize=18, pad=15)
plt.ylabel('Count', fontsize=14)
plt.xlabel('Odometer', fontsize=14, labelpad=10)
plt.gca().xaxis.set_major_formatter(FuncFormatter(my.thousands))
plt.gca().yaxis.set_major_formatter(FuncFormatter(my.thousands))
plt.show()
```



Observation: Here we can see a zoomed in view of Odometer. The majority of data points fall in this range. A long tail is cut off that extends to the right. There seem to be 3 peaks here: extremely low readings, around 40k, and around

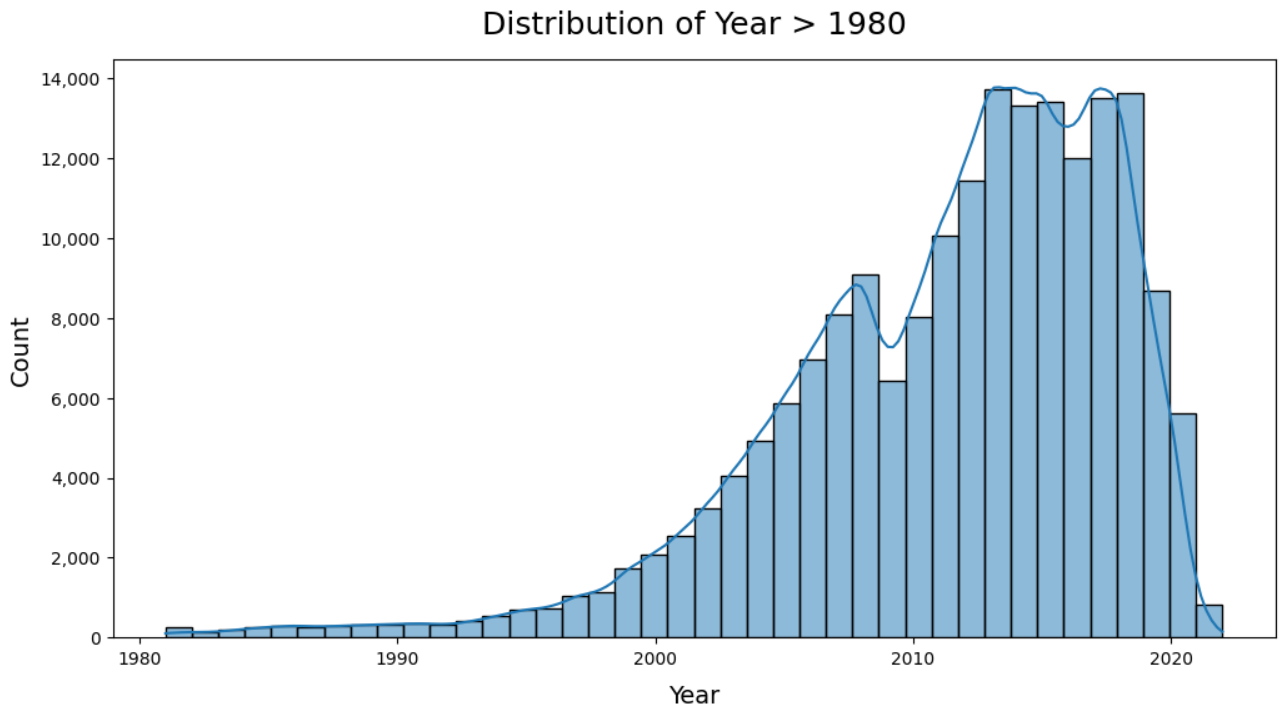
100k. Could these be popular times to sell?

In [90]:
```
plt.figure(figsize=(12,6))
sns.histplot(x='year', kde=True, bins=100, data=df_clean_no_outliers)
plt.title('Distribution of Year', fontsize=18, pad=15)
plt.ylabel('Count', fontsize=14)
plt.xlabel('Year', fontsize=14, labelpad=10)
plt.gca().yaxis.set_major_formatter(FuncFormatter(my.thousands))
plt.show()
```



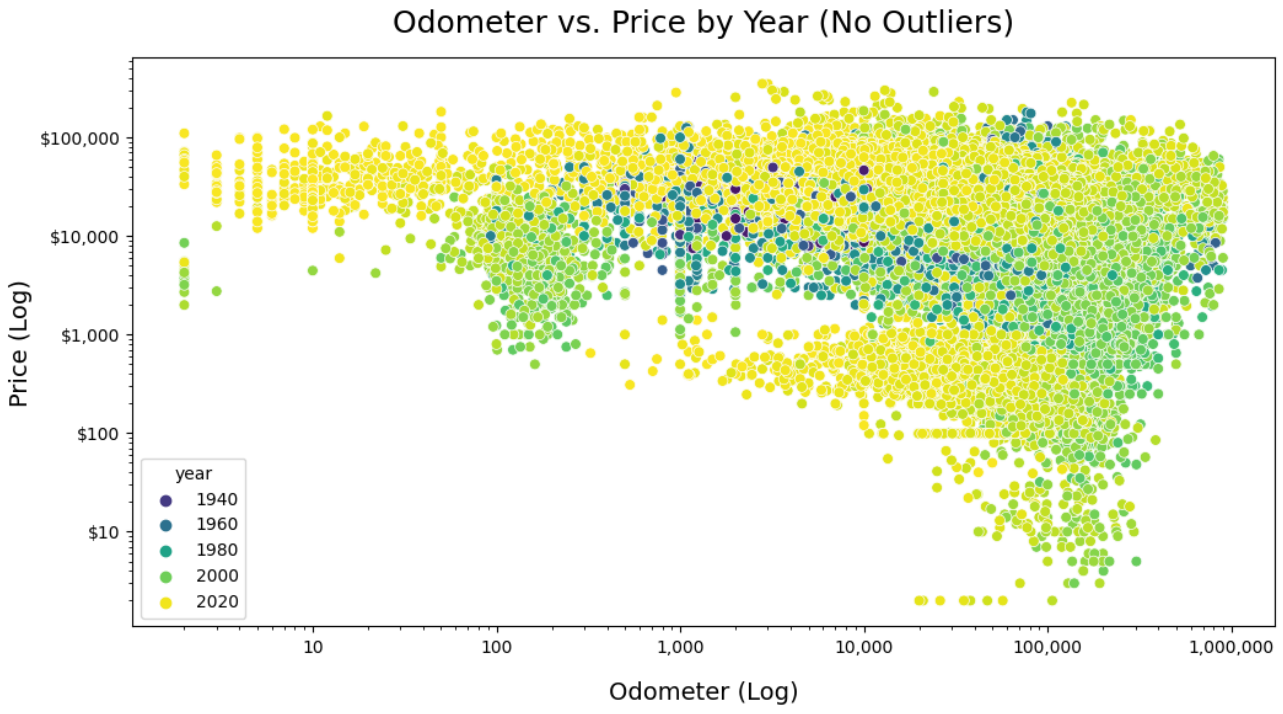
Observation: This is the distribution of Year. Log won't have an affect, and neither did exponential, square root, or cubic root transformations. Perhaps this is fine, but I don't know of a way to get it closer to normal.

In [91]:
```
plt.figure(figsize=(12,6))
sns.histplot(x='year', kde=True, bins=40, data=df_clean_no_outliers.query("year > 1980"))
plt.title('Distribution of Year > 1980', fontsize=18, pad=15)
plt.ylabel('Count', fontsize=14)
plt.xlabel('Year', fontsize=14, labelpad=10)
plt.gca().yaxis.set_major_formatter(FuncFormatter(my.thousands))
plt.show()
```



Observation: This is a zoomed in view of Year from 1980 to 2022. Most of the data falls in this range.

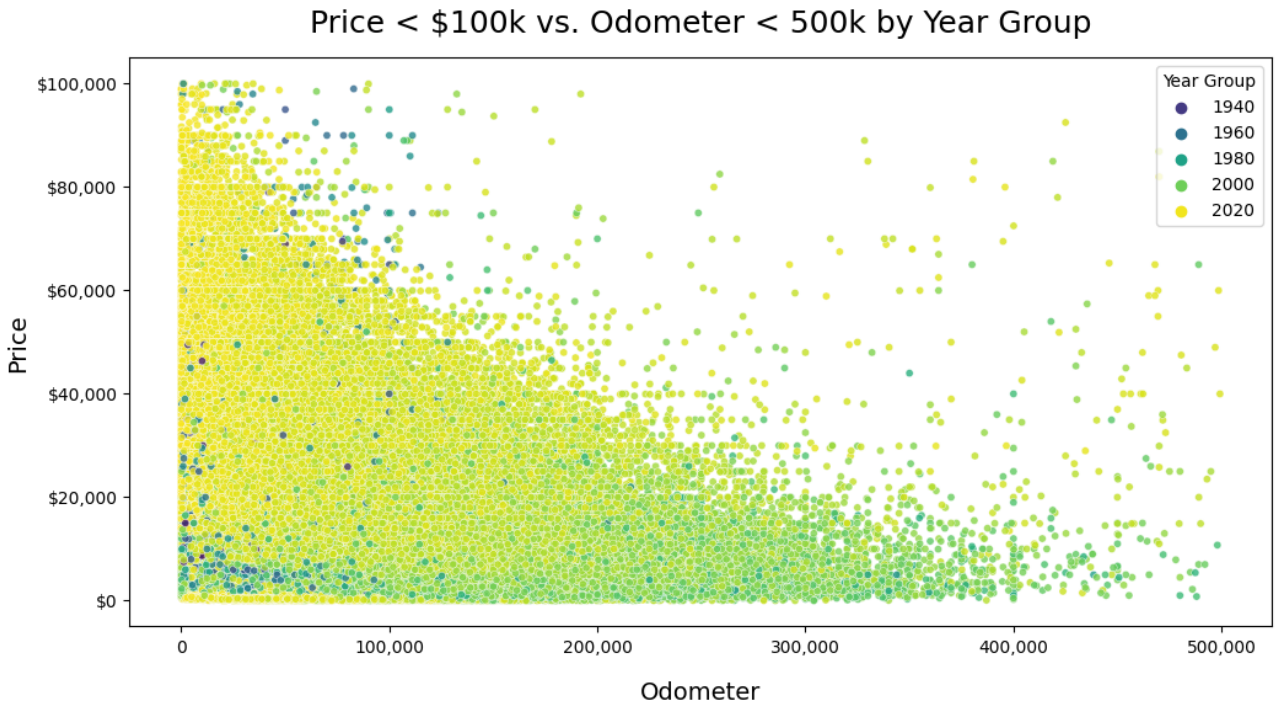
In [95]:
```
plt.figure(figsize=(12,6))
plt.title('Odometer vs. Price by Year (No Outliers)', fontsize=18, pad=15)
sns.scatterplot(data=df_clean_no_outliers, y='price', x='odometer', hue='year', palette='viridis', s=40
plt.xlabel('Odometer (Log)', fontsize=14, labelpad=15)
plt.ylabel('Price (Log)', fontsize=14)
plt.xscale('log')
plt.yscale('log')
plt.gca().yaxis.set_major_formatter(FuncFormatter(my.thousand_dollars))
plt.gca().xaxis.set_major_formatter(FuncFormatter(my.thousands))
plt.show()
```



Insight: This shows the 3 numerical variables in one plot, with all the data in the cleaned dataset, but in log scale to counter the skew. We can see roughly a negative correlation between Price and Odometer. As Odometer increases,

Price decreases. The lighter colors (yellow, light green) show the more recent years. The darker colors (dark green, blue, purple) show older years. There are different clusters here, representing different behaviors of the people posting the listings, or perhaps actually different car segments.

In [96]:
```
plt.figure(figsize=(12,6))
plt.title('Price < $100k vs. Odometer < 500k by Year Group', fontsize=18, pad=15)
sns.scatterplot(data=df_clean_no_outliers.query("price < 100000 and odometer < 500000"), y='price', x=
                hue='year', palette='viridis', s=20, alpha=0.8)
plt.xlabel('Odometer', fontsize=14, labelpad=15)
plt.ylabel('Price', fontsize=14)
plt.gca().yaxis.set_major_formatter(FuncFormatter(my.thousand_dollars))
plt.gca().xaxis.set_major_formatter(FuncFormatter(my.thousands))
plt.legend(title='Year Group')
plt.show()
```

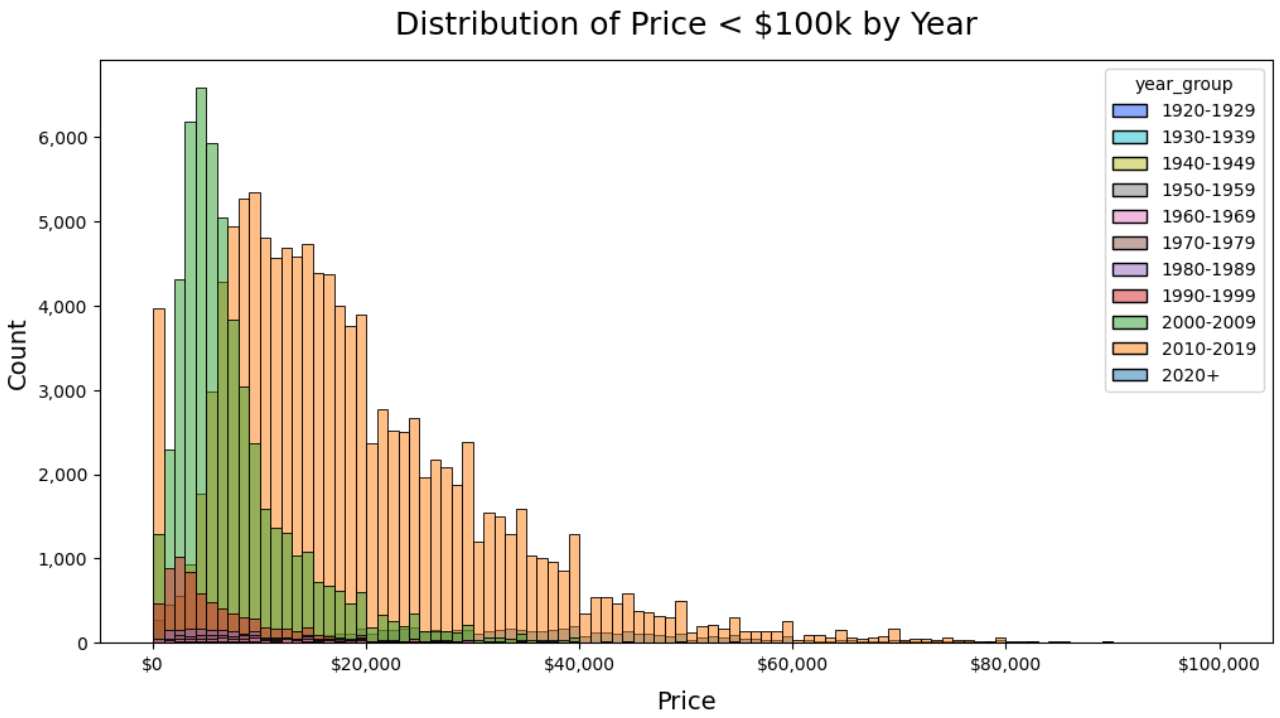


Insight: This is a zoomed in view of the data on regular scale. Only showing Price less than $100k, and Odometer less than 500k. You can see the downward trend quite clearly. As Odometer increases, Price decreases. Most of the data is within this region, the rest are outliers.

In [97]:
```
# Define the bins
yr_bins = list(range(1920, 2031, 10))
yr_names = [f'{i}-{i+9}' for i in range(1920, 2020, 10)]
yr_names.append('2020+')

# Create a new column 'year_group'
df_clean_no_outliers['year_group'] = pd.cut(df_clean_no_outliers['year'], bins=yr_bins, labels=yr_names
```

In [98]:
```
# Reverse the color palette for aesthetics
pal1 = sns.color_palette(["#1f77b4", "#ff7f0e", "#2ca02c", "#d62728", "#9467bd", "#8c564b", "#e377c2",
pal1r = pal1[::-1]
```

In [99]:
```
plt.figure(figsize=(12,6))
sns.histplot(x='price', data=df_clean_no_outliers.query("price < 100000"), kde=False, bins=100, hue='ye
             multiple='layer', palette=pal1r)
plt.title('Distribution of Price < $100k by Year', fontsize=18, pad=15)
plt.ylabel('Count', fontsize=14)
plt.xlabel('Price', fontsize=14, labelpad=10)
plt.gca().xaxis.set_major_formatter(FuncFormatter(my.thousand_dollars))
plt.gca().yaxis.set_major_formatter(FuncFormatter(my.thousands))
plt.show()
```



Insight: In this distribution of price, the slight advantage that newer models have is apparent in the shift of the orange peak (2010-2019) vs. the green peak (2000-2009). There's also a tiny red peak (1990-1999) even further to the left. Once again suggesting that year has a slight effect on price.

Data Encoding

Now let's encode some features in preparation for the modeling.

In [104…
```
# Create a copy of the dataset for encoding
df_enc = pd.DataFrame.copy(df_clean_no_outliers)
```

Text Processing

We will try using TF-IDF to encode the words in the "model" text input. I don't know if this will have an impact on price, but I thought I would try this instead of just dropping the column.

In [105…
```
# Import Texthero library
import texthero as hero
```

In [106…
```
# Clean the text, apply TF-IDF to the words, and calculate PCA of the resulting arrays
# df_enc['pca'] = (
#     df_enc['model']
#     .pipe(hero.clean)
#     .pipe(hero.tfidf)
#     .pipe(hero.pca)
# )
```

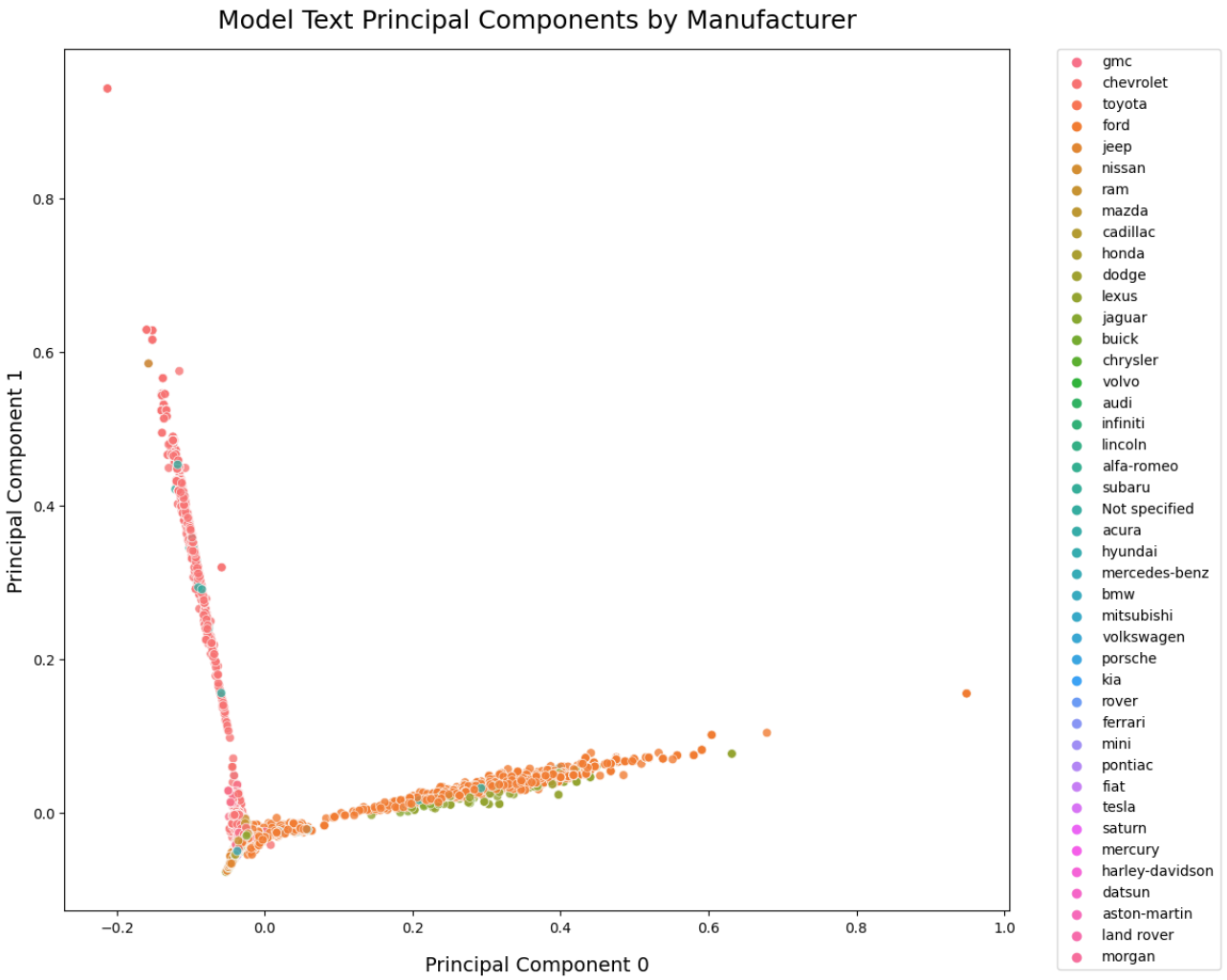
In [107…
```
# Split the components into separate columns
# df_enc['model_pca_0'] = df_enc['pca'].apply(lambda x: x[0])
# df_enc['model_pca_1'] = df_enc['pca'].apply(lambda x: x[1])
```

In [108…
```
# Save the results to CSV since it takes a long time to process the text
#df_enc.to_csv('df_enc.csv', index=False)
```

In [112…
```
# Load the results from the CSV
df_enc = pd.read_csv('data/df_enc.csv')
df_enc.drop(columns=['region','title_status'], inplace=True)
```

In [114…

```
# Plot the principal components
plt.figure(figsize=(12,11))
plt.title('Model Text Principal Components by Manufacturer', fontsize=18, pad=15)
sns.scatterplot(data=df_enc, x='model_pca_0', y='model_pca_1', hue='manufacturer', s=40, alpha=0.8)
plt.xlabel('Principal Component 0', fontsize=14, labelpad=15)
plt.ylabel('Principal Component 1', fontsize=14)
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
plt.show()
```



Insight: This is experimental, but what I see in this chart is interesting. The words have been analyzed and scored based on frequency, and each phrase in the "model" field is converted to an array. PCA reduces these down to 2 components, and here we see a plot of these across all our data points. There appear to be 2 vectors here, the one pointing to the upper left is most associated with "Chevrolet" (red) – so perhaps this represents the scoring of words from Chevrolet models. The one pointing to the right mostly represents "Ford" (orange). So this represents the great American debate of **"Chevrolet vs. Ford"**

Ordinal Encoding

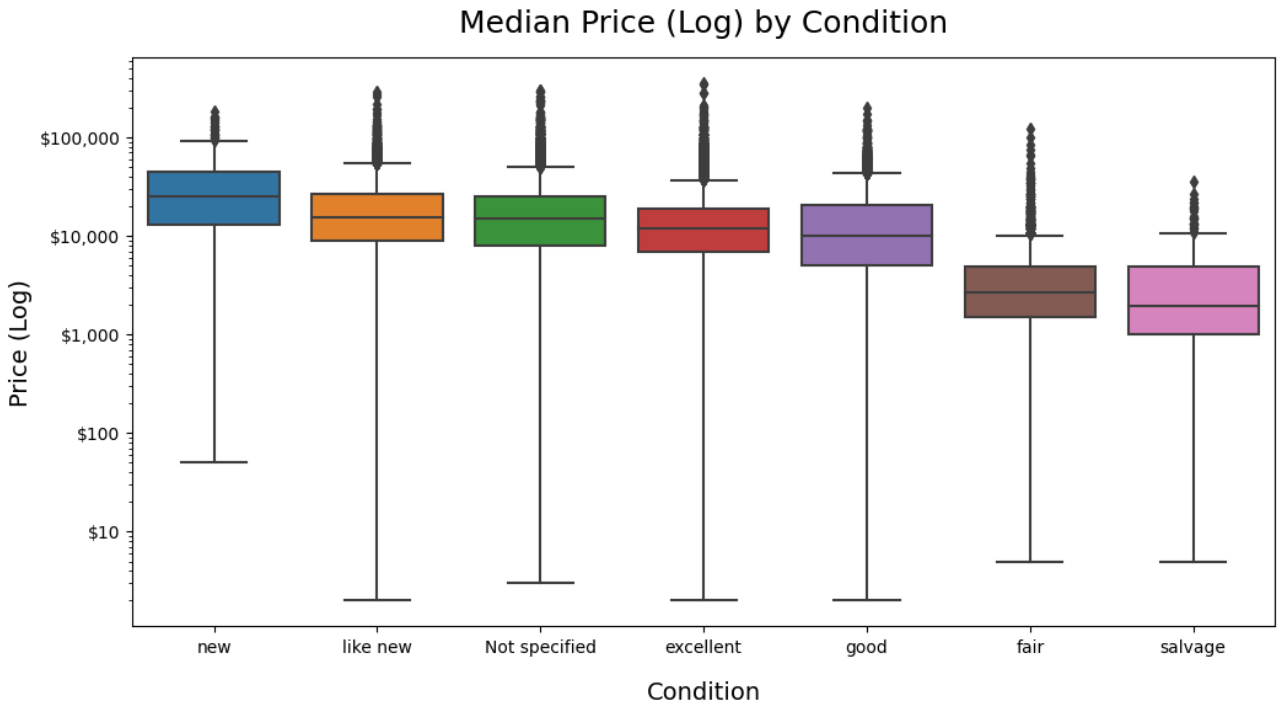
The following variables represent progressive sequences or stages, and so we will create ordinal scales for them. We are going to have too many one-hot encoded variables if we apply that to everything. The ordering is based on logical ordering and an evaluation of the mean of each value.

- **Condition**
  - New: 7
  - Like New: 6
  - Not Specified: 5
  - Excellent: 4

    - Good: 3
    - Fair: 2
    - Salvage: 1
- **Cylinders**
    - 12 cylinders: 12
    - 10 cylinders: 10
    - Other: 9
    - 8 cylinders: 8
    - Not specified: 7
    - 6 cylinders: 6
    - 5 cylinders: 5
    - 4 cylinders: 4
    - 3 cylinders: 3
- **Size**
    - Not specified: 5
    - full-size: 4
    - mid-size: 3
    - compact: 2
    - sub-compact: 1

These will be encoded during pipelines while modeling, but I will encode them here for correlation analysis.

In [115…
```
plt.figure(figsize=(12,6))
plt.title('Median Price (Log) by Condition', fontsize=18, pad=15)
sns.boxplot(y='price', x='condition', data=df_clean_no_outliers,
            order=['new', 'like new', 'Not specified', 'excellent', 'good', 'fair', 'salvage'])
plt.xlabel('Condition', fontsize=14, labelpad=15)
plt.ylabel('Price (Log)', fontsize=14)
plt.yscale('log')
plt.gca().yaxis.set_major_formatter(FuncFormatter(my.thousand_dollars))
plt.show()
```

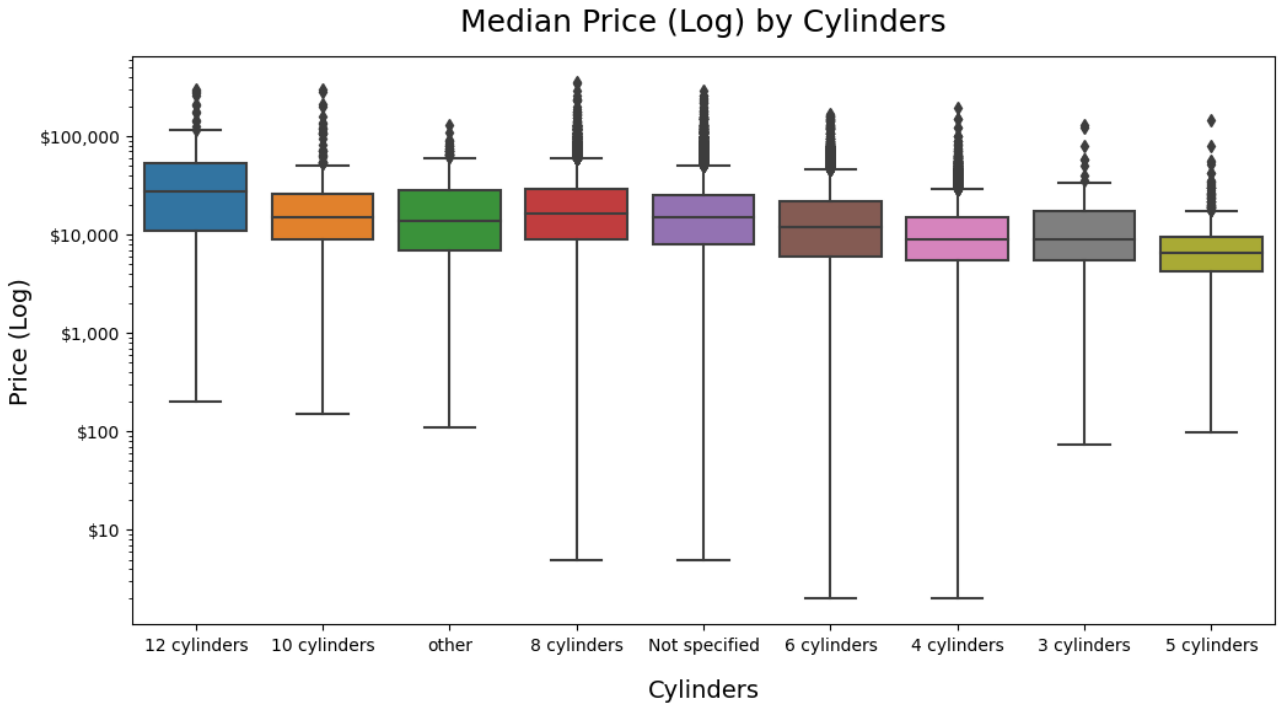


In [116…
```
df_clean_no_outliers.groupby('condition')['price'].mean().sort_values(ascending=False)
```

Out[116]:
```
condition
new              32673.66
like new         20321.40
Not specified    18521.30
excellent        15378.96
good             14351.13
fair              4122.09
salvage           3919.62
Name: price, dtype: float64
```

Insight: There appears to be a relation between Price and Condition. When logically ordered from "New" to "Salvage", the means of Price are ordered from highest to lowest. We also can see the optimal placement for "Not specified" in this sequence.

In [117…
```
plt.figure(figsize=(12,6))
plt.title('Median Price (Log) by Cylinders', fontsize=18, pad=15)
sns.boxplot(y='price', x='cylinders', data=df_clean_no_outliers,
            order=['12 cylinders', '10 cylinders', 'other', '8 cylinders', 'Not specified', '6 cylinder
plt.xlabel('Cylinders', fontsize=14, labelpad=15)
plt.ylabel('Price (Log)', fontsize=14)
plt.yscale('log')
plt.gca().yaxis.set_major_formatter(FuncFormatter(my.thousand_dollars))
plt.show()
```

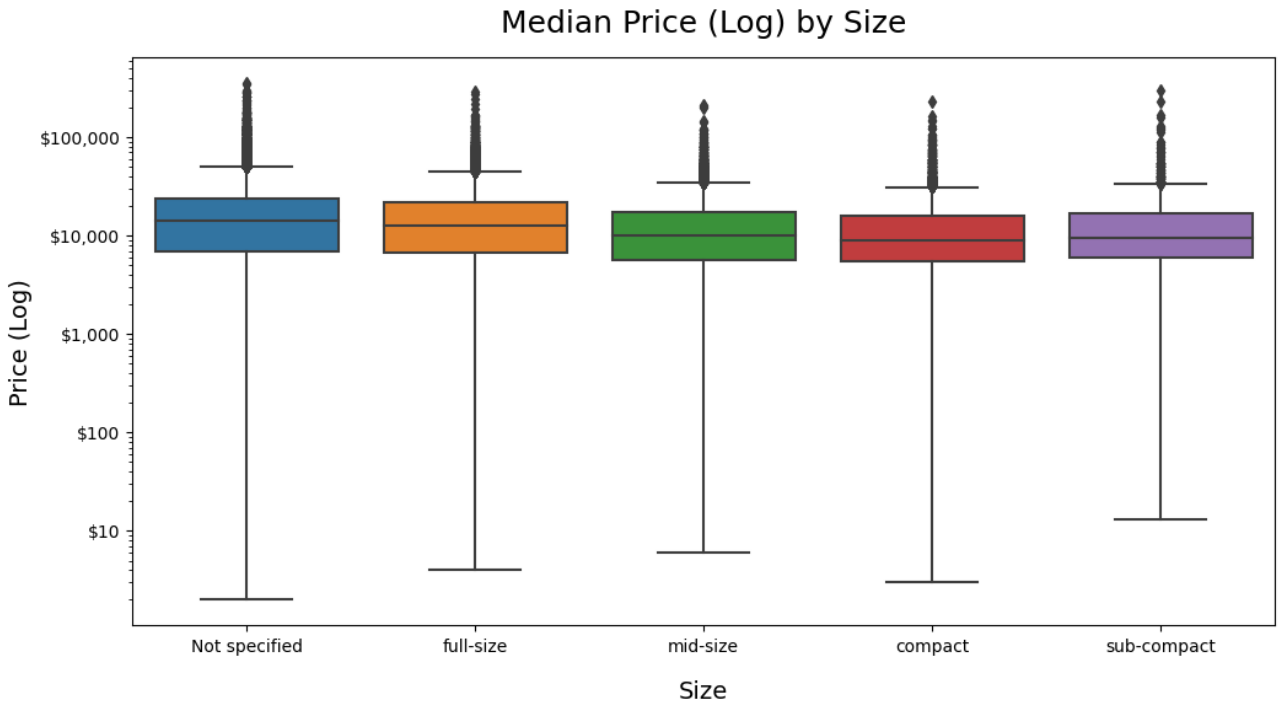


In [118…
```
df_clean_no_outliers.groupby('cylinders')['price'].mean().sort_values(ascending=False)
```

Out[118]:
```
cylinders
12 cylinders    49115.06
10 cylinders    22722.71
other           21054.79
8 cylinders     21051.73
Not specified   18708.32
6 cylinders     15553.83
3 cylinders     12712.41
4 cylinders     11170.98
5 cylinders      7993.50
Name: price, dtype: float64
```

Insight: There seems to be a slight relation between the number of Cylinders and Price. For the most part, more cylinders cost more. But 3 and 5 cylinders are not respecting the strict order. I don't think they are very common configurations either.

In [119…
```
plt.figure(figsize=(12,6))
plt.title('Median Price (Log) by Size', fontsize=18, pad=15)
sns.boxplot(y='price', x='size', data=df_clean_no_outliers,
            order=['Not specified', 'full-size', 'mid-size', 'compact', 'sub-compact'])
plt.xlabel('Size', fontsize=14, labelpad=15)
plt.ylabel('Price (Log)', fontsize=14)
plt.yscale('log')
plt.gca().yaxis.set_major_formatter(FuncFormatter(my.thousand_dollars))
plt.show()
```



Insight: Size doesn't seem to have much of an affect on price. "Not specified" had the highest mean, but it was petty close to "Full Size." The rest are similar in mean Price, but "Sub-Compact" seems more expensive than "Mid-Size" or "Compact." So I'm tempted to drop this feature.

In [120…
```
df_clean_no_outliers.groupby('size')['price'].mean().sort_values(ascending=False)
```

Out[120]:
```
size
Not specified   17651.50
full-size       16360.93
sub-compact     13696.49
mid-size        12885.98
compact         11937.38
Name: price, dtype: float64
```

Create Ordinal Scales

In [121…
```
# Map the value changes for condition
condition_map = {
    'new': 7,
    'like new': 6,
    'Not specified': 5,
    'excellent': 4,
    'good': 3,
    'fair': 2,
    'salvage': 1
}

# Map the value changes for cylinders
cylinders_map = {
    '12 cylinders': 12,
    '10 cylinders': 10,
    'other': 9,
    '8 cylinders': 8,
```

```
    'Not specified': 7,
    '6 cylinders': 6,
    '5 cylinders': 5,
    '4 cylinders': 4,
    '3 cylinders': 3
}

# Map the value changes for size
size_map = {
    'Not specified': 5,
    'full-size': 4,
    'mid-size': 3,
    'compact': 2,
    'sub-compact': 1
}
```

In [122…
```
# Apply the maps
df_enc['condition'] = df_enc['condition'].map(condition_map)
df_enc['cylinders'] = df_enc['cylinders'].map(cylinders_map)
df_enc['size'] = df_enc['size'].map(size_map)
```

One-Hot Encoding

The following variables have non-binary nominal values, and so I will use one-hot encoding to convert them to separate columns for each value (with a 1 to indicate presence, and 0 none).

- **Drive**
- **Fuel**
- **Paint Color**
- **Type**
- **Transmission**
- **Manufacturer**
- **State**

These will be encoded during pipelines while modeling, but I will encode them here for correlation analysis.

In [129…
```
# Create a copy of the dataframe for one-hot encoding
df_ohe = pd.DataFrame.copy(df_enc.drop(columns=['year_log','odometer_log','price_log','pca','model']))
```

In [130…
```
# Create list of remaining categorical columns to one-hot encode
ohe_columns = ['drive', 'fuel', 'paint_color', 'type', 'transmission', 'manufacturer','state']
```

In [131…
```
# Create new dataframe using pd.get_dummies to encode categorical values into separate columns
df_ohe = pd.get_dummies(df_ohe, columns=ohe_columns, dtype='int', drop_first=False,
                        prefix=['drive','fuel','paint','type','trans','manu','state'])
```

In [133…
```
df_ohe['year'] = df_ohe['year'].astype(int)
df_ohe['odometer'] = df_ohe['odometer'].astype(int)
```

Sanity Check

To make sure everything was transformed correctly, need to check for any nulls or NaNs in each column by using `df.isna().sum()`. Then I'll do a sanity check by looking at some rows from the new dataframe (first, last, and a random sample using `df.sample()`.

In [134…
```
# Double-check for any nulls/NaNs as a result of our transformations or encodings
df_ohe.isna().sum().loc[lambda x: x > 0]
```

Out[134]:
```
Series([], dtype: int64)
```

In [138…
```
#df_ohe[:10]
#df_ohe[-10:]
#df_ohe.sample(10)
```

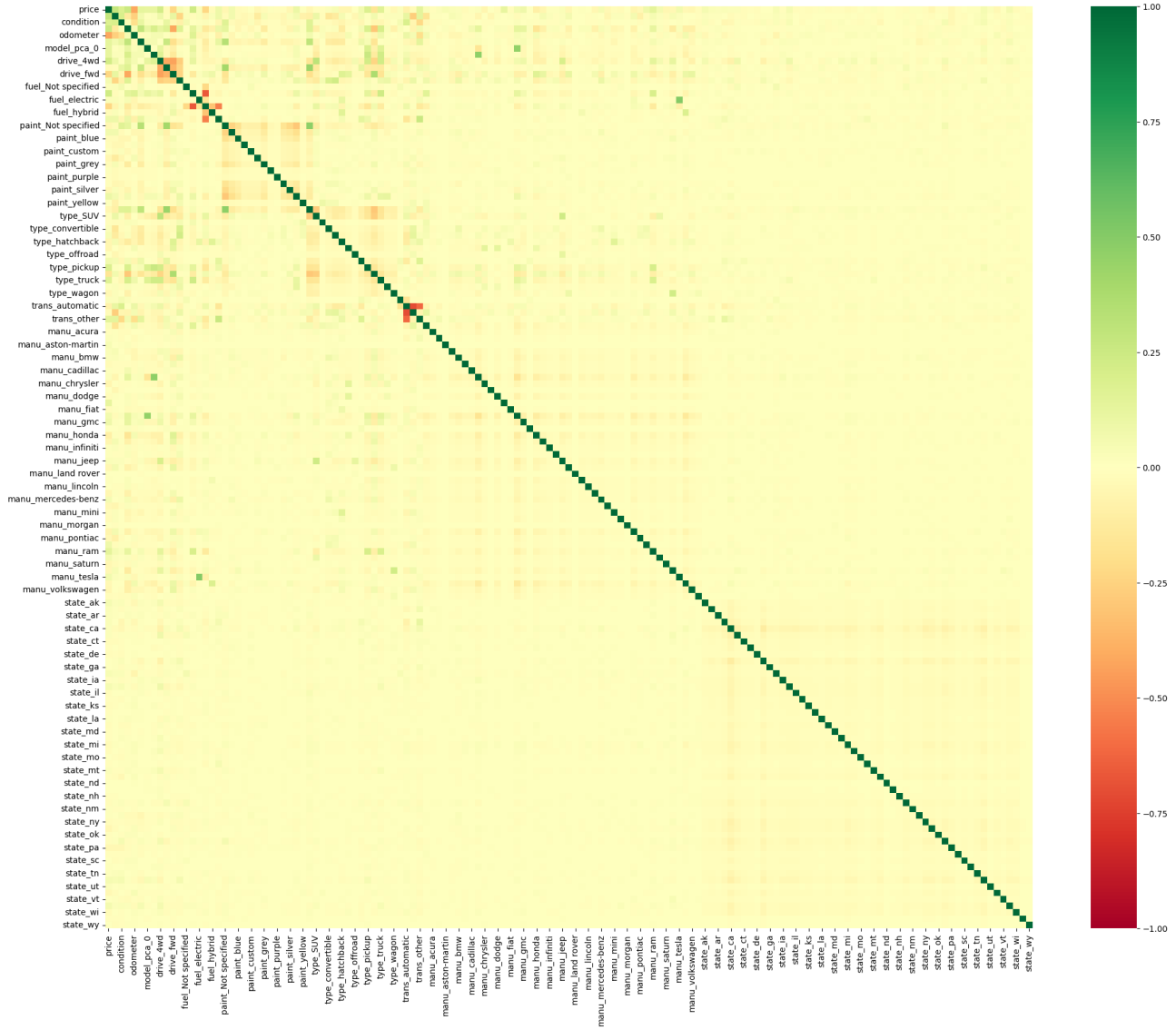
Explore Correlations

Correlation Matrix

Now that everything is encoded numerically, it's time to look for correlations. I run `df.corr` and round the decimels up to 2. I then plot the correlations using a Seaborn heatmap `sns.heatmap`. I like to use the "RdYlGn" color map with "vmin" set to -1, and "vmax" set to 1. This way the strongest negative correlations are solid red, and the strongest positive correlations are solid green. Everything in the middle is yellow (or orange and light green).

In [139…
```
# Create the correlation from the encoded dataset, rounding decimels to 2
corr = round(df_ohe.corr(numeric_only=True), 2)
```

In [140…
```
# Plot the correlation using sns.heatmap
plt.figure(figsize=(25, 20))
sns.heatmap(corr, annot=False, cmap="RdYlGn", vmin=-1, vmax=1)
plt.show()
```



In [145…
```
# Show the top positive and negative correlations
my.get_corr(df_ohe, n=20)
```

```
Top 20 positive correlations:
             Variable 1           Variable 2  Correlation
0         fuel_electric           manu_tesla         0.50
1        manu_chevrolet          model_pca_1         0.47
2   paint_Not specified   type_Not specified         0.47
3             manu_ford          model_pca_0         0.46
4   drive_Not specified   type_Not specified         0.43
5   drive_Not specified  paint_Not specified         0.40
6             drive_fwd           type_sedan         0.37
7   paint_Not specified                size         0.33
8             drive_4wd            type_SUV         0.31
9                  size   type_Not specified         0.30
10            cylinders                size         0.29
11                price                year         0.28
12          model_pca_1          type_pickup         0.26
13  drive_Not specified                size         0.26
14           fuel_other          trans_other         0.26
15            cylinders               price         0.25
16          fuel_diesel               price         0.24
17          fuel_diesel          type_truck         0.24
18            manu_jeep            type_SUV         0.24
19          manu_subaru          type_wagon         0.23

Top 20 negative correlations:
             Variable 1           Variable 2  Correlation
0       trans_automatic         trans_manual        -0.71
1           fuel_diesel             fuel_gas        -0.67
2       trans_automatic          trans_other        -0.62
3              fuel_gas           fuel_other        -0.54
4             drive_4wd            drive_fwd        -0.43
5              odometer                price        -0.42
6             cylinders            drive_fwd        -0.42
7             drive_4wd  drive_Not specified        -0.41
8   drive_Not specified            drive_fwd        -0.37
9              fuel_gas          fuel_hybrid        -0.31
10   type_Not specified           type_sedan        -0.29
11  paint_Not specified          paint_white        -0.29
12            cylinders           type_sedan        -0.29
13   type_Not specified            type_SUV        -0.28
14            type_SUV           type_sedan        -0.27
15        trans_manual                 year        -0.27
16            drive_4wd            drive_rwd        -0.27
17             odometer                 year        -0.26
18            drive_fwd            drive_rwd        -0.25
19  paint_Not specified          paint_black        -0.25
```

Insight: I'll look at the correlations with price separately, but we see some interesting correlations between the independent variables. Perhaps we can drop some of these to simplify the model. I'll ignore the negative correlations within the same feature values due to OHE.

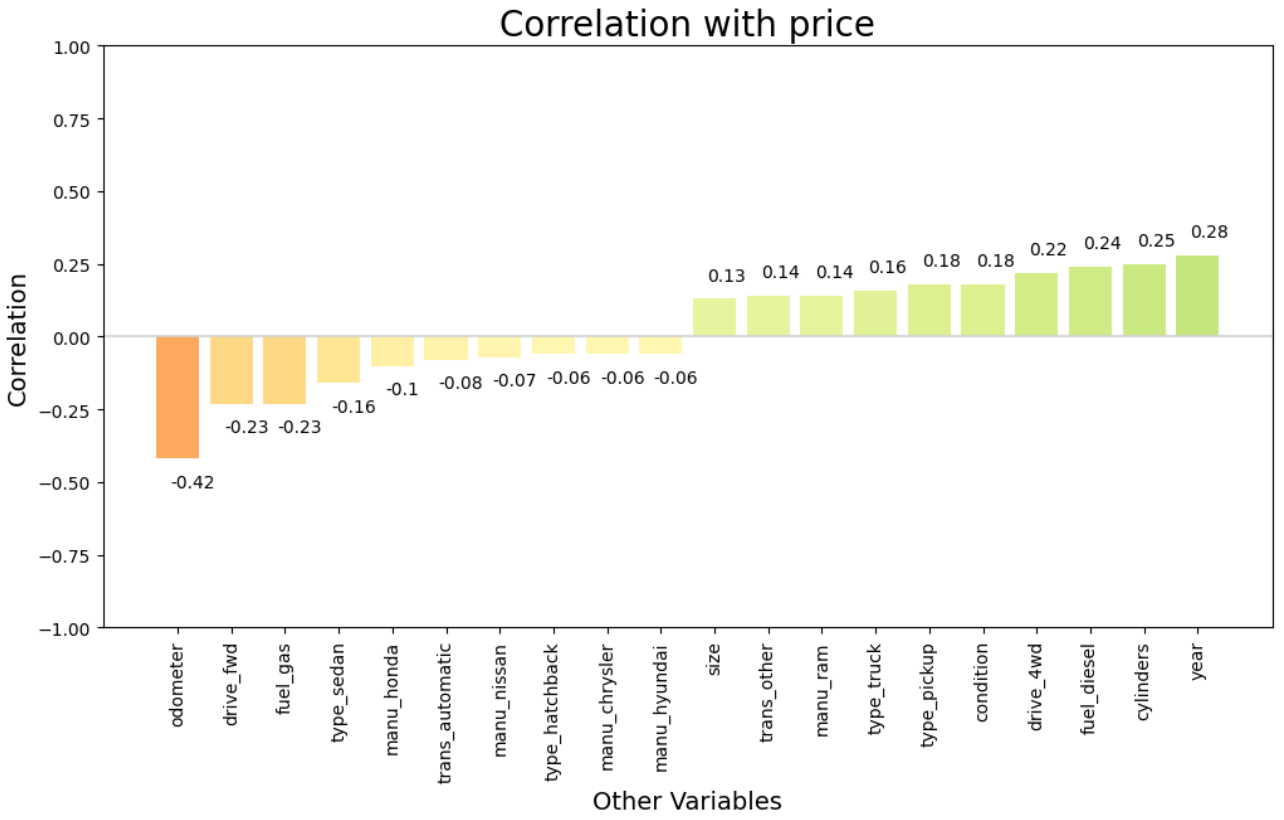
- **Electric** fuel has a moderate correlation with **Tesla**, it's the largest positive at 0.50. This makes sense, as they are the dominant electric vehicle manufacturer.
- As noted from our text analysis, **Model PCA 1** is moderately correlated with **Chevrolet** (0.47) and **Pickup** (0.26), and **Model PCA 0** is moderately correlated with **Ford** (0.46).
- **FWD** is slightly correlated with **Sedan**, and **4WD** is slightly correlated with **SUV**, as one would expect.
- **Diesel** is slightly correlated with **Truck** (0.24), and **Jeep** is slightly correlated with **SUV** (0.24). Both representative of what we know about these kinds of vehicles.
- **Cylinders** is slightly correlated with **Size** (0.29), which makes sense. It's also moderately negatively correlated with **FWD** (-0.42), and slightly negatively correlated with **Sedan** (-0.29).
- **Manual** transmission is slightly negatively correlated with **Year** (-0.27), suggesting less modern vehicles have manual transmission.
- **Odometer** is slightly negatively correlated with **Year** (-0.26), meaning it's typically higher for older vehicles.

Individual Correlation Charts

The heatmap matrix is nice to scan broadly for things of interest. But when presenting the correlation results, I didn't find a way to show just one slice of the heatmap (other than cropping a screenshot of it). It seemed like a good idea to summarize the correlations for one variable (against all others) on a bar chart. So I created a function called `my.plot_corr` (GitHub) that does this.

Every chart is on a fixed y axis from -1 to 1, so visually you can compare/contrast one chart with another to see where there are stronger vs. weaker correlations.

In [147…
```
# Use a custom function to plot a chart showing the correlations with one variable
my.plot_corr(df_ohe, 'price', n=20, size=(12,6), rot=90)
```



Insight: Looks like there a few moderate to slight correlations with our dependent or target variable, Price. This gives us some hope that we can find a model to predict it.

- **Odometer** is moderately negatively correlated with Price (-0.42), which makes sense. Vehicles with more mileage are going to sell for less. This is the strongest relationship in our data set, and may be the main independent variable.
- **Year** is slightly correlated with Price (0.28), which also makes sense. Newer cars tend to cost more than older cars, but we saw from our plots that there may be a class of older cars that cost more which is holding this correlation down.
- **Cylinders** is slightly correlated with Price (0.25), suggesting the more cylinders, the more expensive the vehicle.
- **Diesel** (0.24) and **4WD** (0.22) are slightly correlated with Price, suggesting those type of work or offroad vehicles command a premium.
- Conversely, **Gas** (-0.23) and **FWD** (-0.23) are slighlty negatively correlated with Price, suggesting those less complicated fuel and drive trains are cheaper.
- **Condition** (0.18) is slightly correlated with Price, suggesting cars that are in better condition will sell for more. But it's not the main driver of price (in this dataset, at least)
- **Pickup** (0.18), **Truck** (0.16), and **Ram** (0.14) are slightly correlated with Price, suggesting these types of vehicles are slightly more expensive.
- Conversely, **Sedan** (-0.16) and **Honda** (-0.1) are slightly negatively correlated with Price, suggesting they are slightly cheaper options.
- **Size** is slightly correlated with Price (0.13), suggesting larger vehicles cost a little more.

- In addition to Ram, a few manufacturers also seem to have a slight correlation with Price: **Ferrari** (0.12), **Porsche** (0.08), **GMC** (0.08), **Tesla** (0.07), and **Ford** (0.07).
- **Model PCA 0** (0.12) and **Model PCA 1** (0.12) are also slightly correlated with price, and we know these are correlated with Ford and Chevrolet respectively.
- In addition to Honda, a few manufacturers seem to have a weak negative correlation with Price: **Nissan** (-0.07), **Chrysler** (-0.06), and **Hyundai** (-0.06).
- And lastly, vehicles with **Automatic** transmission (-0.08), or that are **Hatchbacks** (-0.06) or **Minivans** (-0.06), tend to be slightly cheaper.

In [150…
```
# Get the top positive and negative correlations
pos_features, neg_features = my.get_corr(df_ohe, n=20, var='price', return_arrays=True)
```

```
Top 20 positive correlations:
      Variable 1   Variable 2  Correlation
0          price         year         0.28
1      cylinders        price         0.25
2    fuel_diesel        price         0.24
3      drive_4wd        price         0.22
4          price  type_pickup         0.18
5      condition        price         0.18
6          price   type_truck         0.16
7       manu_ram        price         0.14
8          price  trans_other         0.14
9          price         size         0.13
10  manu_ferrari        price         0.12
11   model_pca_1        price         0.12
12   model_pca_0        price         0.12
13   paint_white        price         0.10
14    fuel_other        price         0.09
15  manu_porsche        price         0.08
16     manu_gmc        price         0.08
17   manu_tesla        price         0.07
18         price   type_other         0.07
19     manu_ford        price         0.07

Top 20 negative correlations:
             Variable 1          Variable 2  Correlation
0              odometer               price        -0.42
1             drive_fwd               price        -0.23
2              fuel_gas               price        -0.23
3                 price          type_sedan        -0.16
4            manu_honda               price        -0.10
5                 price     trans_automatic        -0.08
6           manu_nissan               price        -0.07
7         manu_chrysler               price        -0.06
8          manu_hyundai               price        -0.06
9                 price      type_hatchback        -0.06
10                price       type_mini-van        -0.06
11        paint_silver               price        -0.05
12  paint_Not specified               price        -0.05
13            manu_kia               price        -0.05
14           manu_mazda               price        -0.05
15          manu_saturn               price        -0.05
16                price  type_Not specified        -0.05
17                price            state_oh        -0.05
18      manu_volkswagen               price        -0.05
19          paint_green               price        -0.05
```

Create Dataframe of Top Correlated Features

In [151…
```
print("Top positive correlation features:\n", pos_features)
print("\nTop negative correlation features:\n", neg_features)
```

```
Top positive correlation features:
 ['year' 'cylinders' 'fuel_diesel' 'drive_4wd' 'type_pickup' 'condition'
 'type_truck' 'manu_ram' 'trans_other' 'size' 'manu_ferrari' 'model_pca_1'
 'model_pca_0' 'paint_white' 'fuel_other' 'manu_porsche' 'manu_gmc'
 'manu_tesla' 'type_other' 'manu_ford']

Top negative correlation features:
 ['odometer' 'drive_fwd' 'fuel_gas' 'type_sedan' 'manu_honda'
 'trans_automatic' 'manu_nissan' 'manu_chrysler' 'manu_hyundai'
 'type_hatchback' 'type_mini-van' 'paint_silver' 'paint_Not specified'
 'manu_kia' 'manu_mazda' 'manu_saturn' 'type_Not specified' 'state_oh'
 'manu_volkswagen' 'paint_green']
```

In [152…
```
# Combine them together and add the target variable
top_features = np.concatenate((pos_features, neg_features))
top_features = np.concatenate((top_features, ['price']))
```

In [154…
```
# Create a dataframe with just these columns
df_top_features = pd.DataFrame.copy(df_ohe[top_features])
```

In [155…
```
df_top_features
```

Out[155]:

	year	cylinders	fuel_diesel	drive_4wd	type_pickup	condition	type_truck	manu_ram	trans_other	size	manu_f
0	2014	8	0	0	1	3	0	0	1	5	
1	2010	8	0	0	1	3	0	0	1	5	
2	2020	8	0	0	1	3	0	0	1	5	
3	2017	8	0	0	1	3	0	0	1	5	
4	2013	6	0	0	0	4	1	0	0	4	
...	...	...	...	...	...	...	...	...	...	...	
191070	2014	8	0	1	0	6	1	1	0	4	
191071	2006	8	0	1	0	3	1	0	0	4	
191072	1990	8	0	1	0	3	0	0	0	4	
191073	2016	7	0	0	0	3	0	0	0	5	
191074	1997	8	0	0	0	3	0	0	0	2	

191075 rows × 41 columns

Model Building and Analysis

Now let's see if we can meet the third business objective, which is to create a model that has predictive power for pricing used cars. We'll take an iterative approach with this process. And things will get messy. We will need to go back and revisit some of the decisions we made with the dataset cleaning, outlier handling, and encoding strategies.

Test Design

We will start with a baseline and iterate models with the following approach:

1. **Split Dataset** – We will shuffle and split the dataset into X and y features, in Train/Test sets at an 80/20 ratio. Validation will be handled using Cross Validation during modeling as a subset of the Train set.
2. **Establish baseline** – We'll start by establishing a baseline using the mean of our training and test sets. Our model should be better than this.
3. **Start with Simple Linear Regression** – We'll build a Pipeline consisting of One-Hot-Encoding of our categorical features, followed by a Linear Regression. We'll fit the data, run predictions, and evaluate error manually. This will serve as a second baseline for a simple regression model.
4. **Iterate Models** – I'll then iterate models experimenting with various combinations of Ordinal Encoding, One-Hot Encoding, Log Transformation, and Polynomial Transformation. The performance will be evaluated with each

iteration. To expedite this process, I will use a custom function I built called `my.iterate_model`. By passing parameters to it, it will automatically build the pipeline, fit and predict, summarize the error metrics, and plot performance charts.

5. **Feature Selection** – If performance is not adequate, we will try the following approaches to fine tune the features in our model:
   A. **Manual Selection** – We will try the dataset of Top Correlated Features that we created from our correlation matrix results. We will also look at Permutation Feature Importance and Variance Inflation Factor (VIF) for clues to which features have the most effect.
   B. **Algorithmic Selection** – We will also try using Sequential Feature Selection (SFS) with "n" number of features (variable parameter in GridSearchCV), and Lasso to algorithmically suppress the coefficients of some features, which is basically selecting them.
6. **Dataset Modification** – We could end up with a good model, but there could still be issues in our dataset. If performance is still low, we will revist some of the earlier decisions made around data cleaning, outlier handling, duplicate removal, and cut-off thresholds.
7. **Fine Tuning** – Once we have a solid model with decent performance, we will invest resource into fine-tuning the model's hyper-parameters using GridSearchCV. This can take a lot of time given the limited compute resources at our disposal, so we have to save this for targeted tuning of our best candidate model.
8. **Evaluation** – To evaluate the performance and error, we will use **RR²** as our primary measure comparing model performance. We want to see that our model captures the majority of the variance. But we will also look at MSE, RMSE and MAE. RMSE and MAE will give us the real dollar difference in our predictions vs. actuals. We will also look at Residual plots for flatness, and Performance vs. Actual plots for a clear linear relationship.

Data Processing

Load Libraries

In [197…
```
import math
from datetime import datetime
from random import shuffle, seed
from joblib import dump, load
import pytz
from sklearn.preprocessing import StandardScaler, OneHotEncoder, PolynomialFeatures, RobustScaler
from sklearn.preprocessing import OrdinalEncoder, MinMaxScaler, FunctionTransformer
from sklearn.feature_selection import SequentialFeatureSelector, RFE
from sklearn.linear_model import Ridge, Lasso
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from category_encoders import JamesSteinEncoder, TargetEncoder
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.inspection import permutation_importance
from sklearn.impute import KNNImputer, SimpleImputer
from sklearn.pipeline import Pipeline
from sklearn.compose import make_column_transformer, ColumnTransformer, TransformedTargetRegressor
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn.model_selection import LeavePOut, cross_validate, KFold, StratifiedKFold
from sklearn.neighbors import NearestNeighbors
from sklearn.cluster import KMeans, DBSCAN
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

In [157…
```
# Setup pipelines to display as diagrams
from sklearn import set_config
set_config(display="diagram")
```

Update X and y Columns

Let's update our column mappings since we've dropped some variables and identified our encoding plan.

In [210…
```
# Identify the X independent variables and y dependent variable
x_columns = ['year', 'manufacturer', 'condition', 'cylinders', 'fuel', 'odometer',
             'transmission', 'drive', 'size', 'type', 'paint_color', 'state']
x_num_columns = [col for col in x_columns if df[col].dtype in ['int64', 'float64']]
```

```
x_cat_columns = [col for col in x_columns if df[col].dtype in ['object', 'category', 'string']]
y_column = ['price']
```

In [211…
```
# Review our X and y columns
print("X columns: ", x_columns)
print("X numeric columns: ", x_num_columns)
print("X categorical columns: ", x_cat_columns)
print("y column: ", y_column)
```

```
X columns:  ['year', 'manufacturer', 'condition', 'cylinders', 'fuel', 'odometer', 'transmission', 'dr
ive', 'size', 'type', 'paint_color', 'state']
X numeric columns:  ['year', 'odometer']
X categorical columns:  ['manufacturer', 'condition', 'cylinders', 'fuel', 'transmission', 'drive', 's
ize', 'type', 'paint_color', 'state']
y column:  ['price']
```

Separate X and Y

Let's separate the X independent variables from the Y dependent variable. We will do this for a few different datasets we may experiment with in our modeling.

Note: I started with just X and y initially, but had to come back multiple times to create various flavors of X2, X3, X4, X5, X6, X7, and X8. I could probably keep going, but our deadline is approaching. Please pardon the chaos that ensues.

In [294…
```
# Revise x columns to drop low signal features for X4
x4_columns = ['year', 'manufacturer', 'condition', 'cylinders', 'fuel', 'odometer',
              'transmission', 'drive', 'size', 'type', 'model_pca_0', 'model_pca_1']
x4_num_columns = [col for col in x4_columns if df_enc[col].dtype in ['int64', 'float64']]
x4_cat_columns = [col for col in x4_columns if df_enc[col].dtype in ['object', 'category', 'string']]
```

In [295…
```
# Separate cleaned dataframe with no outliers into X and y, requires encoding in Pipeline
X = pd.DataFrame.copy(df_clean_no_outliers[x_columns])
y = pd.DataFrame.copy(df_clean_no_outliers[y_column])

# Alternate dataset with top correlation features after Ordinal/OHE encoding (no encoding needed in Pi
X2 = pd.DataFrame.copy(df_top_features.drop(columns=['price']))
y2 = pd.DataFrame.copy(df_top_features[y_column])

# Alternate dataset with no outliers, requires encoding in Pipeline
X3 = pd.DataFrame.copy(df_clean[x_columns])
y3 = pd.DataFrame.copy(df_clean[y_column])

# Manually ordinal encoded, based on df_enc, dropping low signal features
X4 = pd.DataFrame.copy(df_enc[x4_columns])
y4 = pd.DataFrame.copy(df_enc[y_column])
```

In [311…
```
# Dataset with more strict outlier removal: price < $100k, odometer < 400k
df_100_400 = pd.DataFrame.copy(df_enc.query("price < 100000 and odometer < 400000"))
X5 = pd.DataFrame.copy(df_100_400[x4_columns])
y5 = pd.DataFrame.copy(df_100_400[y_column])
```

In [323…
```
# Dataset based on df_clean, doesn't use previous clustering to remove outliers
# Thresholds: $10 < price < $100k, 10 < odometer < 400k
df_clean_100_400 = pd.DataFrame.copy(df_clean.query("price < 100000 and odometer < 400000"))
df_clean_100_400 = df_clean_100_400.query("price > 10 and odometer > 10")
```

In [329…
```
# Revise x columns to align with X6 pre encoding
x6_columns = ['year', 'manufacturer', 'condition', 'cylinders', 'fuel', 'odometer',
              'transmission', 'drive', 'size', 'type']
x6_num_columns = [col for col in x6_columns if df_enc[col].dtype in ['int64', 'float64']]
x6_cat_columns = [col for col in x6_columns if df_enc[col].dtype in ['object', 'category', 'string']]
```

In [330…
```
# Dataset based on df_clean, no clustering outliers, drops below 10, requires encoding in Pipeline
X6 = pd.DataFrame.copy(df_clean_100_400[x6_columns])
y6 = pd.DataFrame.copy(df_clean_100_400[y_column])
```

In [379…
```
# Dataset with no duplicate removal, no clustered outlier removal
# Thresholds: $100 < price < $100k, 100 < odometer < 400k, 1960 < year
df_dupes_100_400 = pd.DataFrame.copy(df_dupes.query("price < 100000 and odometer < 400000"))
```

```
df_dupes_100_400 = df_dupes_100_400.query("price > 100 and odometer > 100")
df_dupes_100_400 = df_dupes_100_400.query("year > 1960")
```

In [380…
```
X7 = pd.DataFrame.copy(df_dupes_100_400[x6_columns])
y7 = pd.DataFrame.copy(df_dupes_100_400[y_column])
```

In [388…
```
# Dataset with no duplicate removal, no clustered outlier removal
# Thresholds: $1000 < price < $100k, 1000 < odometer < 400k, 1980 < year
df_dupes_1_100_1_400_1980 = pd.DataFrame.copy(.query("price < 100000 and odometer < 400000"))
df_dupes_1_100_1_400_1980 = df_dupes_1_100_1_400_1980.query("price > 1000 and odometer > 1000")
df_dupes_1_100_1_400_1980 = df_dupes_1_100_1_400_1980.query("year > 1980")
```

In [389…
```
X8 = pd.DataFrame.copy(df_dupes_1_100_1_400_1980[x6_columns])
y8 = pd.DataFrame.copy(df_dupes_1_100_1_400_1980[y_column])
```

In [440…
```
# Dataset with top correlated features
# Thresholds: $1000 < price < $100k, 1000 < odometer < 400k, 1980 < year
df_top_features_thresh = pd.DataFrame.copy(df_top_features.query("price < 100000 and odometer < 400000'
df_top_features_thresh = df_top_features_thresh.query("price > 1000 and odometer > 1000")
df_top_features_thresh = df_top_features_thresh.query("year > 1980")
```

In [441…
```
# Top features, with thresholds, manually ordinal encoded
X9 = pd.DataFrame.copy(df_top_features_thresh.drop(columns=['price']))
y9 = pd.DataFrame.copy(df_top_features_thresh[y_column])
```

In [503…
```
# Dataset with top correlated features
# Thresholds: $1000 < price < $80k, 1000 < odometer < 400k, 1980 < year
df_top_features_thresh80 = pd.DataFrame.copy(df_top_features.query("price < 80000 and odometer < 40000(
df_top_features_thresh80 = df_top_features_thresh80.query("price > 1000 and odometer > 1000")
df_top_features_thresh80 = df_top_features_thresh80.query("year > 1980")
```

In [504…
```
# Top features, with thresholds price < $80k, manually ordinal encoded
X10 = pd.DataFrame.copy(df_top_features_thresh80.drop(columns=['price']))
y10 = pd.DataFrame.copy(df_top_features_thresh80[y_column])
```

In [505…
```
# Verify X/Y split
#X.head(), y.head()
print("X, y shapes: ", X.shape, y.shape)
print("X2, y2 shapes: ", X2.shape, y2.shape)
print("X3, y3 shapes: ", X3.shape, y3.shape)
print("X4, y4 shapes: ", X4.shape, y4.shape)
print("X5, y5 shapes: ", X5.shape, y5.shape)
print("X6, y6 shapes: ", X6.shape, y6.shape)
print("X7, y7 shapes: ", X7.shape, y7.shape)
print("X8, y8 shapes: ", X8.shape, y8.shape)
print("X9, y9 shapes: ", X9.shape, y9.shape)
print("X10, y10 shapes: ", X10.shape, y10.shape)
```

```
X, y shapes:  (191075, 12) (191075, 1)
X2, y2 shapes:  (191075, 40) (191075, 1)
X3, y3 shapes:  (193939, 12) (193939, 1)
X4, y4 shapes:  (191075, 12) (191075, 1)
X5, y5 shapes:  (190257, 12) (190257, 1)
X6, y6 shapes:  (190554, 10) (190554, 1)
X7, y7 shapes:  (378007, 10) (378007, 1)
X8, y8 shapes:  (359867, 10) (359867, 1)
X9, y9 shapes:  (177828, 40) (177828, 1)
X10, y10 shapes:  (177482, 40) (177482, 1)
```

Training, Validation and Test Splits

We will split into Train/Test datasets, since cross-validation will handle our Validation set (derived from the Training set).

In [506…
```
# Create training and test datasets, if cross-validation strategy derives the validation set from trai
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)
X2_train, X2_test, y2_train, y2_test = train_test_split(X2, y2, test_size=0.20, random_state=42)
X3_train, X3_test, y3_train, y3_test = train_test_split(X3, y3, test_size=0.20, random_state=42)
X4_train, X4_test, y4_train, y4_test = train_test_split(X4, y4, test_size=0.20, random_state=42)
X5_train, X5_test, y5_train, y5_test = train_test_split(X5, y5, test_size=0.20, random_state=42)
X6_train, X6_test, y6_train, y6_test = train_test_split(X6, y6, test_size=0.20, random_state=42)
X7_train, X7_test, y7_train, y7_test = train_test_split(X7, y7, test_size=0.20, random_state=42)
```

```
X8_train, X8_test, y8_train, y8_test = train_test_split(X8, y8, test_size=0.20, random_state=42)
X9_train, X9_test, y9_train, y9_test = train_test_split(X9, y9, test_size=0.20, random_state=42)
X10_train, X10_test, y10_train, y10_test = train_test_split(X10, y10, test_size=0.20, random_state=42)
```

In [507…
```
# Verify train/test split
#X_train.info(), X_test.info(), y_train.info(), y_test.info()
print("X, y train/test shapes: ", X_train.shape, X_test.shape, y_train.shape, y_test.shape)
print("X2, y2 train/test shapes: ", X2_train.shape, X2_test.shape, y2_train.shape, y2_test.shape)
print("X3, y3 train/test shapes: ", X3_train.shape, X3_test.shape, y3_train.shape, y3_test.shape)
print("X4, y4 train/test shapes: ", X4_train.shape, X4_test.shape, y4_train.shape, y4_test.shape)
print("X5, y5 train/test shapes: ", X5_train.shape, X5_test.shape, y5_train.shape, y5_test.shape)
print("X4, y4 train/test shapes: ", X4_train.shape, X4_test.shape, y4_train.shape, y4_test.shape)
print("X6, y6 train/test shapes: ", X6_train.shape, X6_test.shape, y6_train.shape, y6_test.shape)
print("X7, y7 train/test shapes: ", X7_train.shape, X7_test.shape, y7_train.shape, y7_test.shape)
print("X8, y8 train/test shapes: ", X8_train.shape, X8_test.shape, y8_train.shape, y8_test.shape)
print("X9, y9 train/test shapes: ", X9_train.shape, X9_test.shape, y9_train.shape, y9_test.shape)
print("X10, y10 train/test shapes: ", X10_train.shape, X10_test.shape, y10_train.shape, y10_test.shape)
```

```
X, y train/test shapes:  (152860, 12) (38215, 12) (152860, 1) (38215, 1)
X2, y2 train/test shapes:  (152860, 40) (38215, 40) (152860, 1) (38215, 1)
X3, y3 train/test shapes:  (155151, 12) (38788, 12) (155151, 1) (38788, 1)
X4, y4 train/test shapes:  (152860, 12) (38215, 12) (152860, 1) (38215, 1)
X5, y5 train/test shapes:  (152205, 12) (38052, 12) (152205, 1) (38052, 1)
X4, y4 train/test shapes:  (152860, 12) (38215, 12) (152860, 1) (38215, 1)
X6, y6 train/test shapes:  (152443, 10) (38111, 10) (152443, 1) (38111, 1)
X7, y7 train/test shapes:  (302405, 10) (75602, 10) (302405, 1) (75602, 1)
X8, y8 train/test shapes:  (287893, 10) (71974, 10) (287893, 1) (71974, 1)
X9, y9 train/test shapes:  (142262, 40) (35566, 40) (142262, 1) (35566, 1)
X10, y10 train/test shapes:  (141985, 40) (35497, 40) (141985, 1) (35497, 1)
```

Establish Baseline

Now we'll establish a baseline using the mean of our dataset.

In [187…
```
# Create a baseline of same shape as y_train and y_test populated with their means as values
baseline_train = np.ones(shape=y_train.shape) * y_train.iloc[:, 0].mean()
baseline_test = np.ones(shape=y_test.shape) * y_test.iloc[:, 0].mean()
#y_train.shape, baseline_train.shape, y_test.shape, baseline_test.shape
```

Linear Regression Model with OHE

Build a Pipeline

We'll use Scikit-Learn's `Pipeline()` feature to setup the stages of processing that will include `OneHotEncoder()`

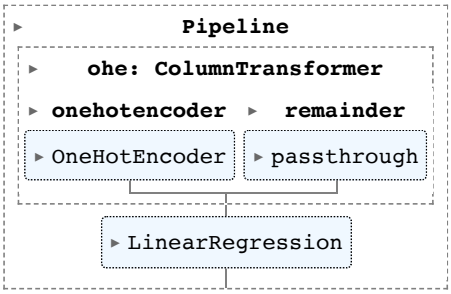
In [192…
```
# Create a column transformer to apply one-hot to selected variables
col_transformer = make_column_transformer((OneHotEncoder(drop = 'if_binary'), x_cat_columns),
                                          remainder='passthrough')

# Create a Pipeline for data processing with one-hot encoding
pipe = Pipeline([
    ('ohe', col_transformer),
    ('linreg', LinearRegression())
])
```

Fit the Model

Now let's fit the model with the trianing data.

In [193…
```
# Train the model using our training data
pipe.fit(X_train, y_train)
```

Out[193]:



Make Predictions

And finally make predictions using our test data (we'll include predictions from our training data for comparison, it will help give us a sense if the model is overfitted or underfitted).

In [194…

```
# Predict y from our test dataset
y_train_pred = pipe.predict(X_train)
y_test_pred = pipe.predict(X_test)
```

Evalute Performance

Now let's see how well this initial model performs. We'll look at a few variables here:

- **Mean Squared Error (MSE)** – The average of the squared differences between the predicted and actual values.
- **Root Mean Squared Error (RMSE)** – The square root of the MSE, it represents the average magnitude of errors between predicted and actual values.
- **Mean Absolute Error (MAE)** – The average of the absolute differences between the predicted and actual values.
- **R² Score** – The proportion of variance in the dependent variable that is predictable from the independent variable(s).

Baseline Performance

In [198…

```
# Evaluate baseline MSE, RMSE, MAE, and R^2 score
baseline_train_mse = mean_squared_error(y_train, baseline_train)
baseline_test_mse = mean_squared_error(y_test, baseline_test)
baseline_train_mae = mean_absolute_error(y_train, baseline_train)
baseline_test_mae = mean_absolute_error(y_test, baseline_test)
baseline_train_r2 = r2_score(y_train, baseline_train)
baseline_test_r2 = r2_score(y_test, baseline_test)
print(f'Baseline Train MSE: {baseline_train_mse: ,.0f}')
print(f'Baseline Test MSE: {baseline_test_mse: ,.0f}')
print(f'Baseline Train RMSE: {np.sqrt(baseline_train_mse): ,.0f}')
print(f'Baseline Test RMSE: {np.sqrt(baseline_test_mse): ,.0f}')
print(f'Baseline Train MAE: {baseline_train_mae: ,.0f}')
print(f'Baseline Test MAE: {baseline_test_mae: ,.0f}')
print(f'Baseline Train R^2 Score: {baseline_train_r2: ,.2f}')
print(f'Baseline Test R^2 Score: {baseline_test_r2: ,.2f}')
```

```
Baseline Train MSE:  209,810,775
Baseline Test MSE:  204,856,890
Baseline Train RMSE:  14,485
Baseline Test RMSE:  14,313
Baseline Train MAE:  10,382
Baseline Test MAE:  10,338
Baseline Train R^2 Score:  0.00
Baseline Test R^2 Score:  0.00
```

Model Performance

In [199…

```
# Evaluate model MSE, RMSE, MAE and R^2 score
train_mse = mean_squared_error(y_train, y_train_pred)
test_mse = mean_squared_error(y_test, y_test_pred)
train_mae = mean_absolute_error(y_train, y_train_pred)
test_mae = mean_absolute_error(y_test, y_test_pred)
train_r2 = r2_score(y_train, y_train_pred)
test_r2 = r2_score(y_test, y_test_pred)
print(f'Model Train MSE: {train_mse: ,.0f}')
```

```
print(f'Model Test MSE: {test_mse: ,.0f}')
print(f'Model Train RMSE: {np.sqrt(train_mse): ,.0f}')
print(f'Model Test RMSE: {np.sqrt(test_mse): ,.0f}')
print(f'Model Train MAE: {train_mae: ,.0f}')
print(f'Model Test MAE: {test_mae: ,.0f}')
print(f'Model Train R^2 Score: {train_r2: ,.2f}')
print(f'Model Test R^2 Score: {test_r2: ,.2f}')
```

```
Model Train MSE:  121,490,199
Model Test MSE:  116,243,824
Model Train RMSE:  11,022
Model Test RMSE:  10,782
Model Train MAE:  6,837
Model Test MAE:  6,819
Model Train R^2 Score:  0.42
Model Test R^2 Score:  0.43
```

Insight: We have an R^2 score on the Test dataset of 0.43. This shows that about 43% of the variance is accounted for by the model. It's just a starting point, we will try to improve this in model iteration. We also see the MAE, which indicates that on average, the model's predictions are off by \$6,819 from the actual values.

Permutation Feature Importance

Let's calculate the Permutation Feature Importance, which "removes" features one at a time (randomly shuffles their values), and evaluates their impact on the model's performance. Those that reduce the model performance the most get a higher score.

In [201…
```
# Set dataframe output to not show scientific notation
pd.set_option('display.float_format', '{:.4f}'.format)
```

In [203…
```
# Calculate feature permutation importance
my.calc_fpi(pipe, X_train, y_train)
```

Out[203]:

	Variables	Score Mean	Score Std
5	odometer	0.3228	0.0017
0	year	0.0922	0.0008
7	drive	0.0659	0.0008
4	fuel	0.0644	0.0007
3	cylinders	0.0417	0.0004
9	type	0.0238	0.0004
1	manufacturer	0.0088	0.0002
10	paint_color	0.0073	0.0002
6	transmission	0.0071	0.0002
2	condition	0.0054	0.0001
11	state	0.0042	0.0001
8	size	-0.0002	0.0000

Insight: It looks like the most important feature in this model is Odometer (0.32), followed by Year (0.09), Drive (0.07), and Fuel (0.06). This confirms the correlations we saw earlier with Odometer and Year.

Variance Inflation Factor

The Variance Inflation Factor is a measure of multicollinearity, or how correlated independent variables are with each other. Ideally, we want them to have low correlation with each other, but strong correlation with the dependent variable.

In [205…
```
# Calculate VIF
my.calc_vif(X_train.drop(columns=x_cat_columns))
```

Out[205]:

	variables	VIF
0	year	3.2711
1	odometer	3.2711

Insight: This shows us that Year and Odometer have low multicollinearity, as we would expect. These are the only numeric variables I can calcualte VIF on right now. The rest are categorical and the OHE encoded version is massive and crashes VIF.

Model Iterations

I created a function called `my.iterate_model` (GitHub) that takes X and y dataframes as input, along with parameters for: transformers or encoders, scalers, feature selectors, and the model. It will then build a pipeline dynamically, and perform the (optional) cross validation you specify. It also allows you to plot residuals and actual vs. predicted charts, calculate feature permutation, VIF, or analyze the coefficients. It has an option to save the results to a dataframe, so the performance of each model can be tracked and analyzed later.

This function relies on lists of columns, and dictionaries of the available transformers/encoders, scalers, feature selectors, models, parameters (for GridSearchCV), and cross-validation strategies. When you call the `my.iterate_model` function, you specify keys to these dictionaries, and their definitions will be used to assemble the pipeline in each iteration. The function is in the imported mytools.py library file, but these definitions now follow.

Create Dataframe to Track Results

In [209…

```
# Create a dataframe to store the results of each model iteration
my.results_df = pd.DataFrame(columns=['Iteration', 'Train MSE', 'Test MSE', 'Train RMSE', 'Test RMSE',
                                      'Train MAE', 'Test MAE', 'Train R^2 Score', 'Test R^2 Score',
                                      'Best Grid Mean Score', 'Best Grid Params', 'Pipeline', 'Note',
```

Identify Columns to Transform

In [337…

```
# Identify columns for various transformations and encodings
#
# all_columns
#     x_columns
#         x_num_columns
#         x_cat_columns
#     y_column
#     num_columns
#         skew_columns
#     cat_columns

ohe2_columns = ['manufacturer','fuel','transmission','drive','type']
ord_columns = ['condition','cylinders','size']
ohe_columns = ['manufacturer','fuel','transmission','drive','type','paint_color','state']
log_columns = ['odometer']
ohe_drop_columns = ohe_columns
poly2_columns = x_num_columns
poly3_columns = x_num_columns
js_columns = x_cat_columns
targ_columns = x_cat_columns
drop_columns = x_cat_columns

# One-hot encoding optional drop value, one per column that is encoded
# drop_categories = ['TBD', None, None] # Drop 'TBD' for first feature, don't drop any category for sec
ohe_drop_categories = []

# Define the order for ordinal encoding
#feature = [['1', '2', '3', '4', '5']]
#feature_order = [[]]
condition_order = ['salvage','fair','good','excellent','Not specified','like new','new']
cylinders_order = ['3 cylinders', '4 cylinders','5 cylinders','6 cylinders','Not specified','8 cylinder
size_order = ['sub-compact','compact','mid-size','full-size','Not specified']
```

```
ord_categories = [condition_order, cylinders_order, size_order]

# Placeholders in case we use array-based train/test splits
train_index = []
test_index = []
```

In [231…
```
X_train['size'].unique()
```

Out[231]:
```
array(['full-size', 'Not specified', 'compact', 'mid-size', 'sub-compact'],
      dtype=object)
```

Define Encoders, Transformers and Models

In [547…
```
# Create dictionaries of encoders/transformers, scalers, selectors, and models

my_config = {
    'transformers': {
        'ohe': (OneHotEncoder(handle_unknown='ignore'), ohe_columns),
        'ohe2': (OneHotEncoder(handle_unknown='ignore'), ohe2_columns),
        'ohe_all': (OneHotEncoder(handle_unknown='ignore'), x_cat_columns),
        'ohe_drop': (OneHotEncoder(drop='if_binary', handle_unknown='ignore'), ohe_drop_columns),
        'ord': (OrdinalEncoder(categories=ord_categories), ord_columns),
        'js': (JamesSteinEncoder(), js_columns),
        'targ': (TargetEncoder(cols=targ_columns), targ_columns),
        'poly2_bias': (PolynomialFeatures(degree=2, include_bias=True), poly2_columns),
        'poly2': (PolynomialFeatures(degree=2, include_bias=False), poly2_columns),
        'poly3_bias': (PolynomialFeatures(degree=3, include_bias=True), poly3_columns),
        'poly3': (PolynomialFeatures(degree=3, include_bias=False), poly3_columns),
        'log': (FunctionTransformer(np.log1p, validate=True), log_columns),
        'imp_mean':(SimpleImputer(strategy='mean')),
        'imp_median':(SimpleImputer(strategy='median')),
        'imp_freq':(SimpleImputer(strategy='most_frequent')),
        'imp_fill':(SimpleImputer(strategy='constant', fill_value=0)),
     },
    'scalers': {
        'stand': StandardScaler(with_mean=False),
        'robust': RobustScaler(),
        'minmax': MinMaxScaler(),
    },
    'selectors': {
        'sfs': SequentialFeatureSelector(LinearRegression()),
        'sfs_lasso': SequentialFeatureSelector(Lasso(tol=0.02, max_iter=10000)),
        'sfs_bw': SequentialFeatureSelector(LinearRegression(), direction='backward'),
        'rfe_lasso': RFE(Lasso()),
    },
    'models': {
        'linreg': LinearRegression(),
        'ridge': Ridge(),
        'lasso': Lasso(),
        'lasso_tol': Lasso(tol=0.2, max_iter=10000),
        'ttr_log': TransformedTargetRegressor(regressor=LinearRegression(), func=np.log, inverse_func=n
        'random_forest': RandomForestRegressor(),
        'gradient_boost': GradientBoostingRegressor(),
    },
    'params': {
        'ttr_log': {
            'Model: ttr_log__regressor': [LinearRegression()],
        },
        'rfe_lasso_4': {
            'Selector: rfe_lasso__n_features_to_select': [4],
        },
        'sfs_lasso_4': {
            'Selector: sfs_lasso__n_features_to_select': [4],
        },
        'sfs_lasso_4': {
            'Selector: sfs_lasso__n_features_to_select': [4],
        },
        'sfs_25_35': {
            'Selector: sfs__n_features_to_select': np.arange(25, 35, 1),
        },
        'sfs_bw': {
            'Selector: sfs_bw__n_features_to_select': np.arange(3, 13, 1),
        },
        'linreg': {
```

```
            'Model: linreg__fit_intercept': [True],
        },
        'ridge': {
            'Model: ridge__tol': [0.001],
        },
        'ridge_100000': {
            'Model: ridge__alpha': np.array([0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000, 100000]),
        },
        'ridge_100': {
            'Model: ridge__alpha': np.arange(0.00, 100, 5),
        },
        'ridge_10': {
            'Model: ridge__alpha': np.arange(0.00, 10, 0.5),
        },
        'ridge_1': {
            'Model: ridge__alpha': np.arange(0.00, 1, 0.05),
        },
        'lasso': {
            'Model: lasso__alpha': np.arange(0.00, 1.0, 0.05),
        },
        'lasso_tol': {
            'Model: lasso_tol__alpha': [1.0],
        },
        'lasso_tol_1': {
            'Model: lasso_tol__alpha': np.array([0.001, 0.01, 0.1, 1, 10, 100]),
        }
    },
    'cv': {
        'kfold_5': KFold(n_splits=5, shuffle=True, random_state=42),
        'kfold_10': KFold(n_splits=10, shuffle=True, random_state=42),
        'skf_5': StratifiedKFold(n_splits=5, shuffle=True, random_state=42),
        'skf_10': StratifiedKFold(n_splits=10, shuffle=True, random_state=42),
        'loo': LeaveOneOut(),
        'lpo_2': LeavePOut(p=2),
        'indices': [[train_index, test_index]],
    }
}
#my_config
```

In [401…

```
#importlib.reload(my) # Reload custom library during development and testing
```

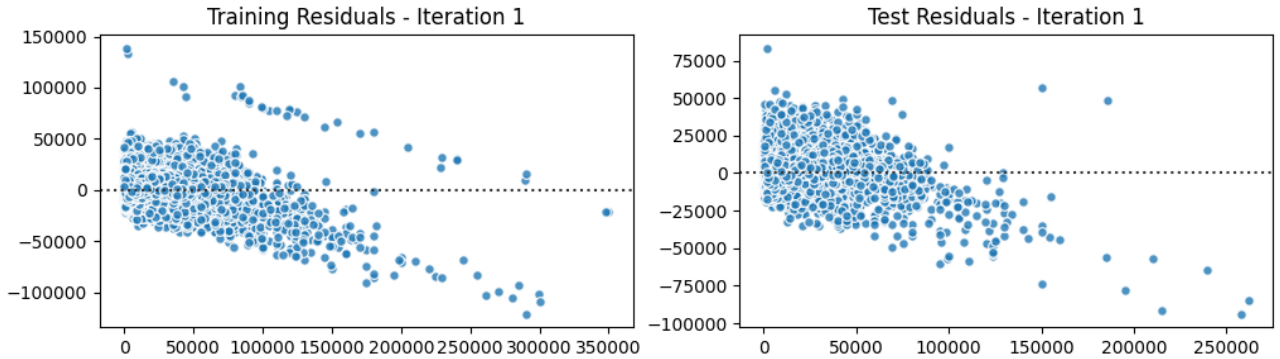
Model Iteration 1

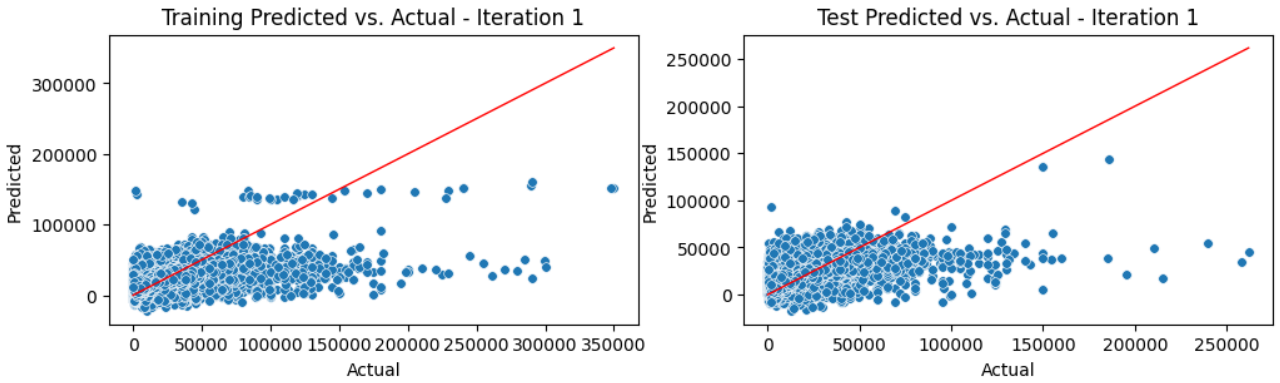
X, Log, Ordinal, OHE, Standard, LinReg

In [402…

```
i1_model = my.iterate_model(X_train, X_test, y_train, y_test,
                  transformers=['log','ord','ohe'], scaler='stand', model='linreg',
                  iteration='1', note='Log. Ordinal. OHE. Linear Regression.',
                  grid=False, grid_params='linreg', grid_cv='kfold_10', grid_verbose=3,
                  save=True, export=False, config=my_config, debug=False, decimal=4,
                  plot=True, perm=False, vif=False, coef=True)
```

```
ITERATION 1 RESULTS

Pipeline: Transformers: log_ord_ohe -> Scaler: stand -> Model: linreg
Note: Log. Ordinal. OHE. Linear Regression.
Aug 28, 2023 09:18 PM PST

Predictions:
                          Train            Test
MSE:            110,769,211.7304 109,990,192.3691
RMSE:               10,524.6953     10,487.6209
MAE:                 6,550.3411      6,542.0263
R^2 Score:               0.4721          0.4631
```

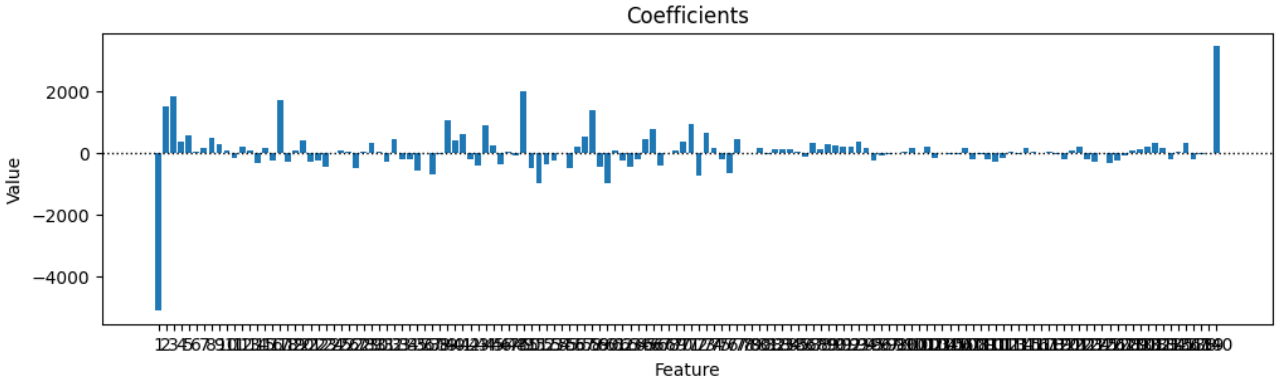




Coefficients:

	Feature	Value
1	odometer	-5,129.9768
2	condition	1,483.1489
3	cylinders	1,809.1892
4	size	356.4251
5	manufacturer_Not specified	548.9049
..	...	...
136	state_wa	316.6746
137	state_wi	-202.6266
138	state_wv	-33.9991
139	state_wy	-7.7173
140	0	3,452.3641

[140 rows x 2 columns]



Observation: The Test R^2 is **0.46**, which is a slight improvement from our baseline Linear Regression, but it's really not good. The residuals are not flat, there's actually some odd separation in the plot. I wonder if it's a problem with the data processing. We also have 140 coefficients due to OHE. The complexity is too high.

Model Iteration 2

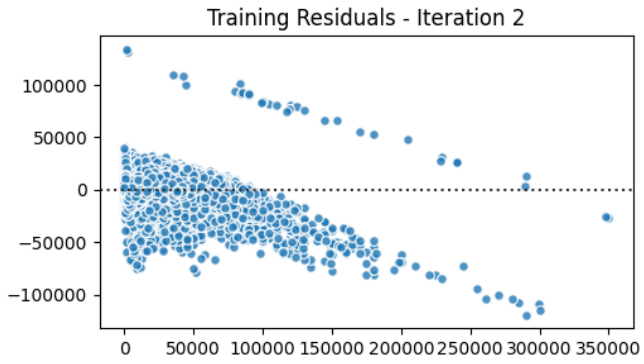
X2 Top Correlated Features, Linear Regression

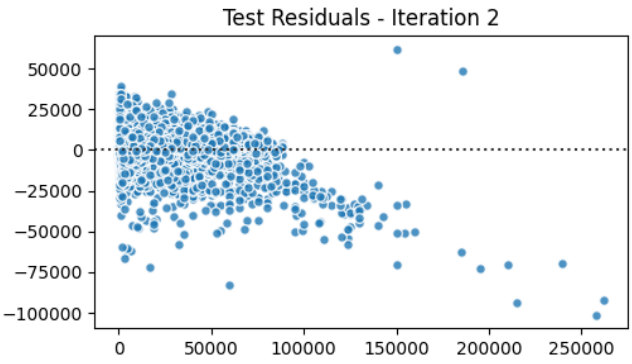
In [404…

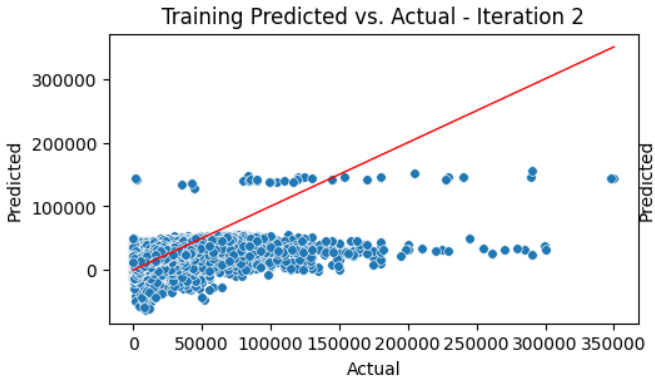
```
i2_model = my.iterate_model(X2_train, X2_test, y2_train, y2_test,
                            model='linreg',
```

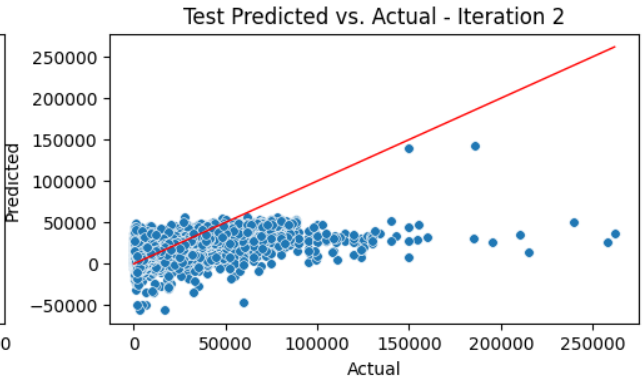
```
                  iteration='2', note='X2 top correlated features after OHE and Ordinal. Linear Regressi
                  grid=False, grid_params='linreg', grid_cv='kfold_10', grid_verbose=3,
                  save=True, export=False, config=my_config, debug=False, decimal=4,
                  plot=True, perm=False, vif=False, coef=True)
```

```
ITERATION 2 RESULTS

Pipeline: Model: linreg
Note: X2 top correlated features after OHE and Ordinal. Linear Regression.
Aug 28, 2023 09:27 PM PST

Predictions:
                          Train            Test
MSE:            112,586,311.1345 109,536,664.6736
RMSE:               10,610.6697      10,465.9765
MAE:                 6,619.9673       6,577.5393
R^2 Score:               0.4634           0.4653
```



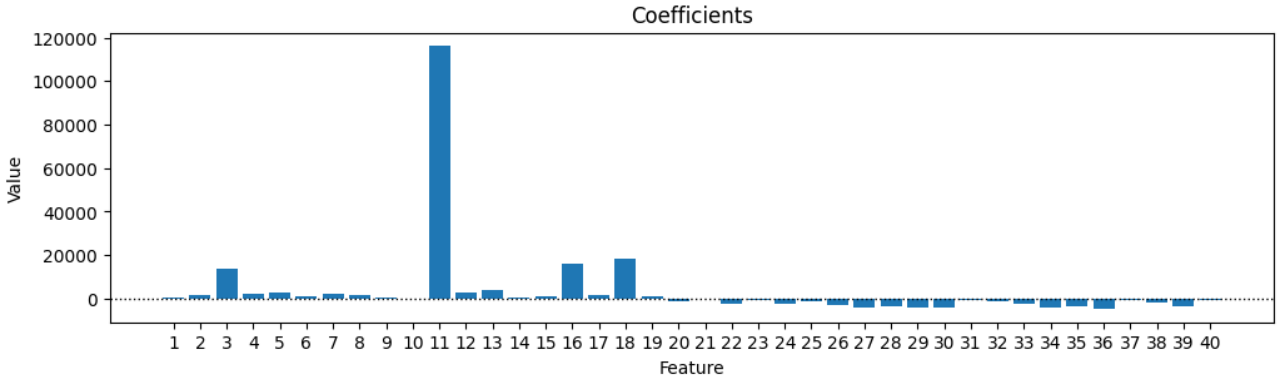






Coefficients:

	Feature	Value
1	year	289.4980
2	cylinders	1,469.8982
3	fuel_diesel	13,784.2521
4	drive_4wd	2,193.8809
5	type_pickup	3,001.6141
6	condition	1,192.0283
7	type_truck	2,333.7499
8	manu_ram	2,001.2846
9	trans_other	736.3973
10	size	170.7397
11	manu_ferrari	116,132.8092
12	model_pca_1	2,950.4106
13	model_pca_0	3,853.1749
14	paint_white	450.6043
15	fuel_other	1,065.4783
16	manu_porsche	16,102.6621
17	manu_gmc	1,910.8376
18	manu_tesla	18,647.5572
19	type_other	1,203.1257
20	manu_ford	-1,011.5847
21	odometer	-0.0857
22	drive_fwd	-2,108.0223
23	fuel_gas	-456.2096
24	type_sedan	-2,231.2591
25	manu_honda	-895.8089
26	trans_automatic	-3,105.4681
27	manu_nissan	-3,790.0732
28	manu_chrysler	-3,462.3338
29	manu_hyundai	-3,821.0052
30	type_hatchback	-3,778.4644
31	type_mini-van	-494.3528
32	paint_silver	-993.1493
33	paint_Not specified	-2,315.5839
34	manu_kia	-4,245.3760
35	manu_mazda	-3,191.3321
36	manu_saturn	-4,741.2302
37	type_Not specified	-528.3093
38	state_oh	-1,755.0392
39	manu_volkswagen	-3,730.1032
40	paint_green	-658.1732



Observation: The Test R^2 is still **0.46**, despite changing to a reduced dataset of the top correlated features (after manually performing Ordinal and OHE and then evaluating correlations in a matrix). However, we get the same performance for a munch simpler model – only 40 coefficients. Interesting that "Ferrari" is the strongest one! The data still does not look right, I think I either cut too much data out in cleaning, or didn't cut out enough.

Model Iteration 3

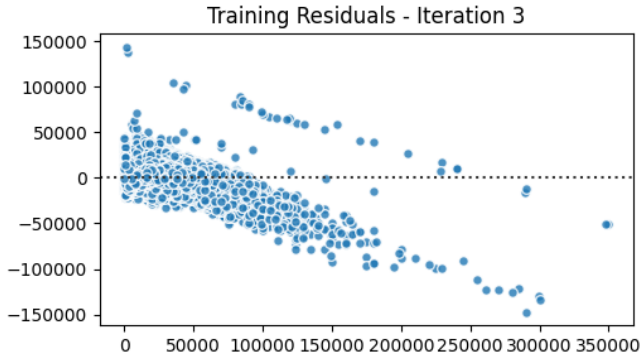
X, Ordinal, OHE, Poly 2, Standard, LinReg

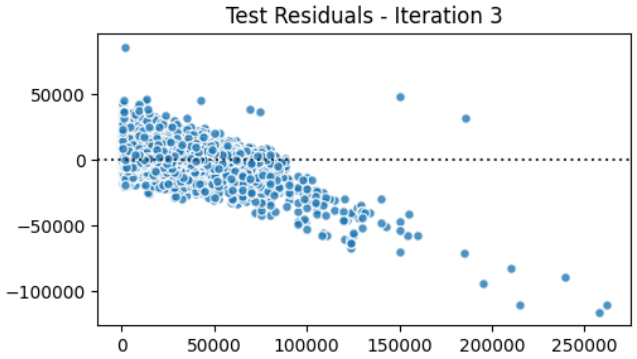
In [405…

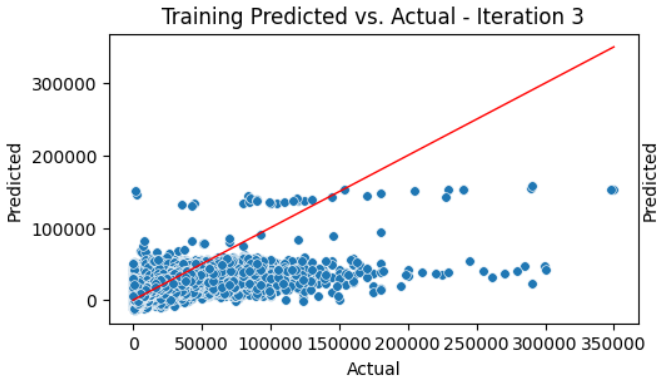
```
i3_model = my.iterate_model(X_train, X_test, y_train, y_test,
                transformers=['ord','ohe','poly2'], scaler='stand', model='linreg',
                iteration='3', note='X. Ordinal. OHE. Poly2. Standard Scaler. Linear Regression.',
```

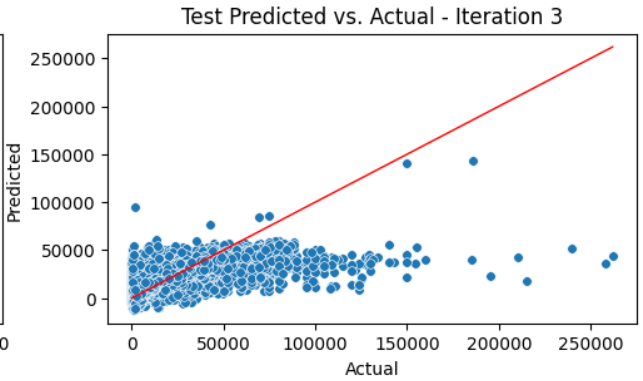
```
                grid=False, grid_params='linreg', grid_cv='kfold_10', grid_verbose=3,
                save=True, export=False, config=my_config, debug=False, decimal=4,
                plot=True, perm=False, vif=False, coef=True)
```

```
ITERATION 3 RESULTS

Pipeline: Transformers: ord_ohe_poly2 -> Scaler: stand -> Model: linreg
Note: X. Ordinal. OHE. Poly2. Standard Scaler. Linear Regression.
Aug 28, 2023 09:36 PM PST

Predictions:
                        Train            Test
MSE:           91,234,007.2247 89,786,932.2100
RMSE:                9,551.6495      9,475.5967
MAE:                 5,749.9688      5,735.0700
R^2 Score:               0.5652          0.5617
```

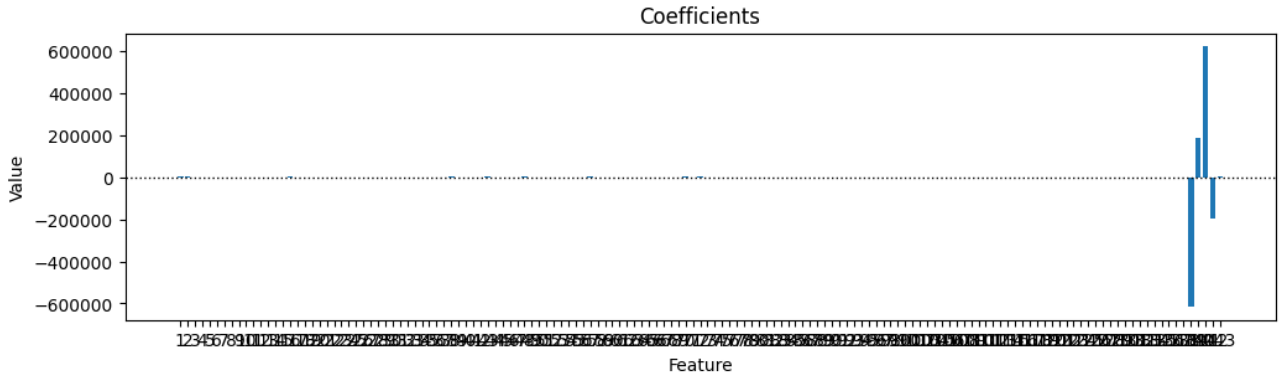









```
Coefficients:
                          Feature         Value
1                       condition      739.6623
2                       cylinders    2,117.8079
3                            size      162.2291
4    manufacturer_Not specified      573.3575
5              manufacturer_acura       94.2823
..                            ...           ...
139                          year -616,237.4201
140                      odometer  187,828.7516
141                        year^2  620,835.7095
142                 year odometer -193,861.7918
143                     odometer^2    2,928.1100

[143 rows x 2 columns]
```



Observation: Back to the original X dataset. The Test R^2 is better at **0.56**, thanks to the Polynomial degree 2 transformation. However, the residuals and predicted vs. actuals plots are not clean. There are 143 coefficients but it looks like only 4 are being used. The data still does not look right, I think I either cut too much data out in cleaning, or didn't cut out enough.

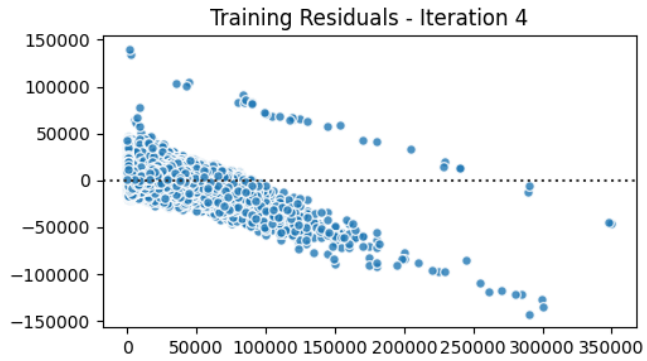
Model Iteration 4

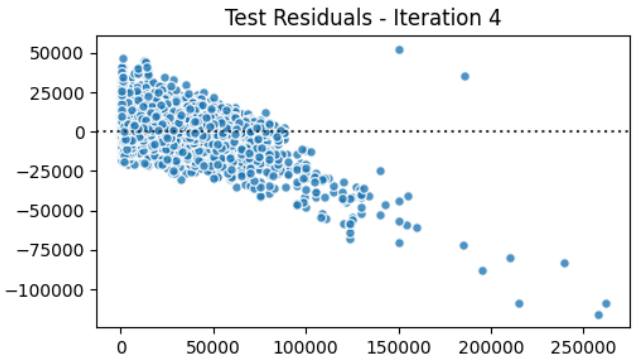
X2 Top Features, Poly 2, Standard, LinReg

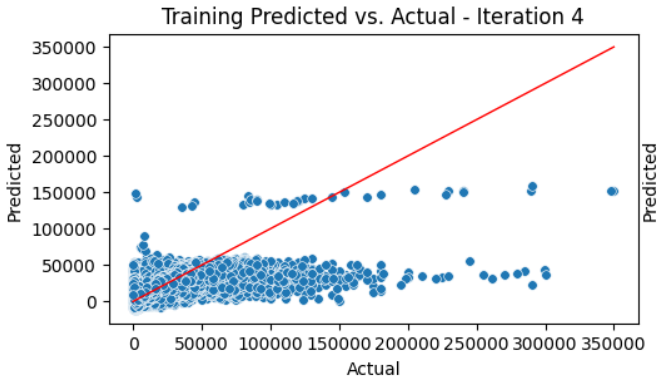
In [406…

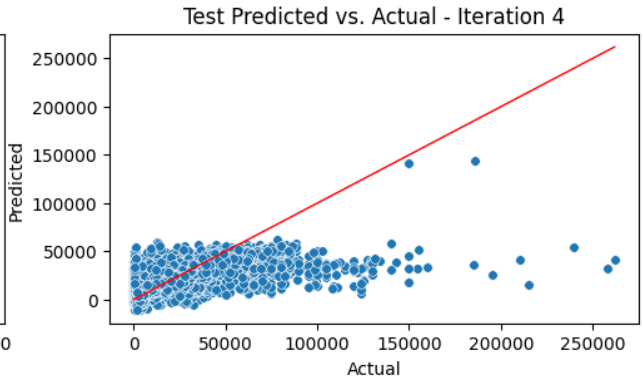
```
i4_model = my.iterate_model(X2_train, X2_test, y2_train, y2_test,
                transformers='poly2', scaler='stand', model='linreg',
                iteration='4', note='X2 top correlated features after OHE and Ordinal. Poly 2. Standar
                grid=False, grid_params='linreg', grid_cv='kfold_10', grid_verbose=3,
                save=True, export=False, config=my_config, debug=False, decimal=4,
                plot=True, perm=False, vif=False, coef=True)
```

```
ITERATION 4 RESULTS

Pipeline: Transformers: poly2 -> Scaler: stand -> Model: linreg
Note: X2 top correlated features after OHE and Ordinal. Poly 2. Standard. Linear Regression.
Aug 28, 2023 10:01 PM PST

Predictions:
                          Train            Test
MSE:            95,274,957.3645 93,217,551.0331
RMSE:                 9,760.8892      9,654.9237
MAE:                  5,897.8263      5,865.6659
R^2 Score:                0.5459          0.5450
```



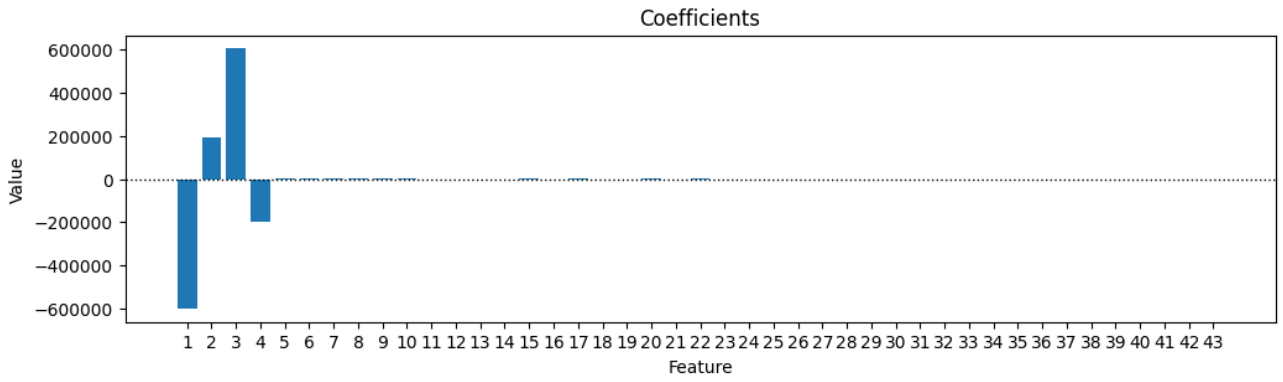






Coefficients:

	Feature	Value
1	year	-602,328.1155
2	odometer	193,943.3564
3	year^2	606,602.0115
4	year odometer	-200,416.8104
5	odometer^2	3,206.7226
6	1	2,276.9860
7	2	3,067.8309
8	3	1,019.3302
9	4	760.8257
10	5	716.0960
11	6	537.4502
12	7	318.0594
13	8	-292.7003
14	9	-7.4210
15	10	1,755.8441
16	11	456.2253
17	12	753.2950
18	13	102.4063
19	14	123.4318
20	15	1,091.5403
21	16	323.3149
22	17	635.4736
23	18	146.1285
24	19	-564.8092
25	21	-878.2991
26	22	5.8673
27	23	-906.8216
28	24	-151.8954
29	25	-810.7650
30	26	-888.2757
31	27	-359.5151
32	28	-665.9617
33	29	-714.7300
34	30	-31.3046
35	31	-172.5670
36	32	-964.5136
37	33	-728.5824
38	34	-310.9079
39	35	-112.4394
40	36	-187.1650
41	37	-272.7825
42	38	-549.7058
43	39	-3.9362



Observation: Trying the X2 dataset with Top Correlated Features, but this time with Polynomial degree 2 transformation and Standard scaler. The Test R^2 is slightly worse at **0.55**. The residuals and predicted vs. actuals plots still do not look good.

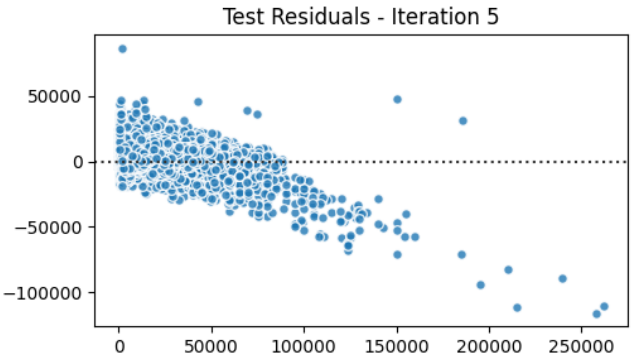
Model Iteration 5

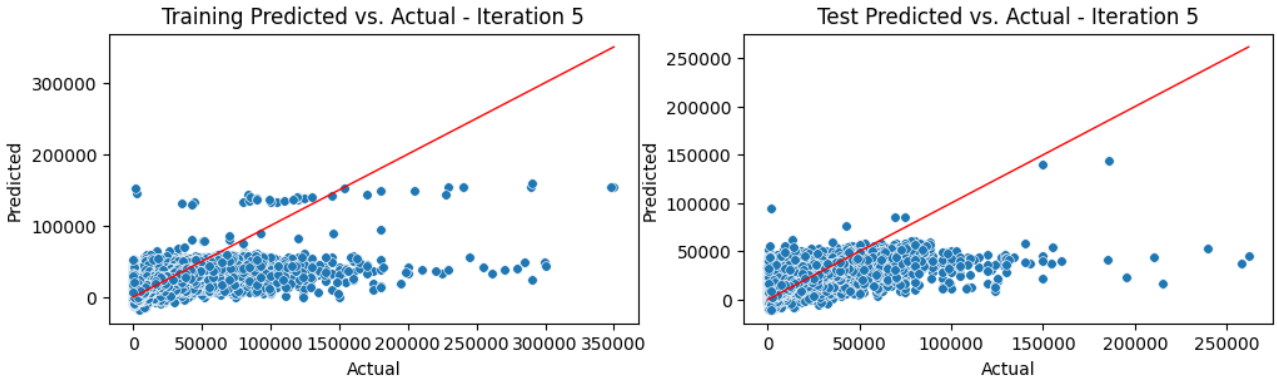
X, Ordinal, OHE, Poly 3, Standard, LinReg

In [408…

```
i5_model = my.iterate_model(X_train, X_test, y_train, y_test,
                  transformers=['ord','ohe','poly3'], scaler='stand', model='linreg',
                  iteration='5', note='X. Ordinal. OHE. Poly3. Standard Scaler. Linear Regression.',
                  grid=False, grid_params='linreg', grid_cv='kfold_10', grid_verbose=3,
                  save=True, export=False, config=my_config, debug=False, decimal=4,
                  plot=True, perm=False, vif=False, coef=True)
```

```
ITERATION 5 RESULTS

Pipeline: Transformers: ord_ohe_poly3 -> Scaler: stand -> Model: linreg
Note: X. Ordinal. OHE. Poly3. Standard Scaler. Linear Regression.
Aug 28, 2023 10:08 PM PST

Predictions:
                          Train            Test
MSE:             90,228,040.6360 89,085,064.6422
RMSE:                9,498.8442      9,438.4885
MAE:                 5,725.1258      5,716.8127
R^2 Score:               0.5700          0.5651
```

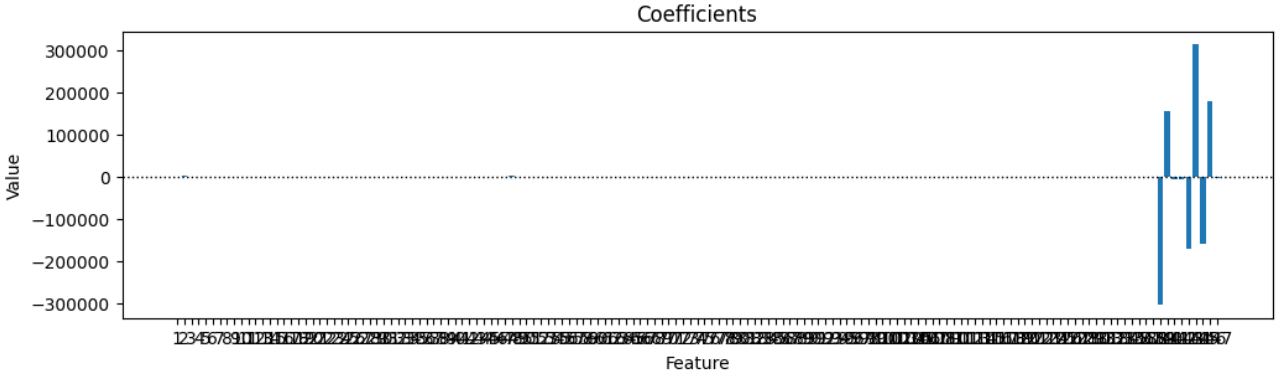




Coefficients:

	Feature	Value
1	condition	784.8125
2	cylinders	2,080.1903
3	size	180.0396
4	manufacturer_Not specified	539.5672
5	manufacturer_acura	87.0161
..	...	...
143	odometer^2	-171,588.7276
144	year^3	314,554.8946
145	year^2 odometer	-159,433.3610
146	year odometer^2	179,643.3019
147	odometer^3	-3,503.6017

[147 rows x 2 columns]



Observation: Back to the original X dataset. The Test R^2 is **0.56** with Polynomial degree 3 transformation, not any different from degree 2, but with much more model complexity. Lots of coefficients, residuals and predicted vs. actuals still look bad.

Model Iteration 6

X3 with Outliers, Ordinal, OHE, Poly 2, Standard, LinReg

In [409…

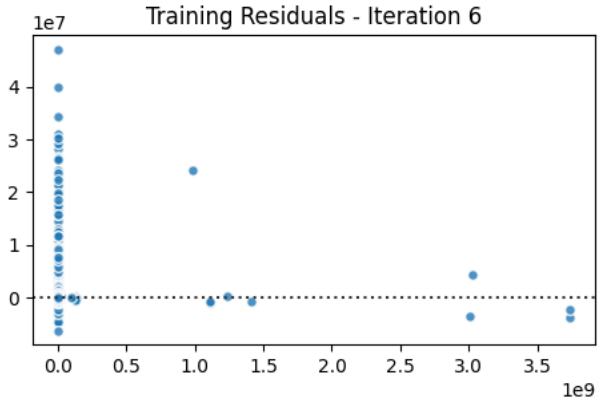
```
i6_model = my.iterate_model(X3_train, X3_test, y3_train, y3_test,
                transformers=['ord','ohe','poly2'], scaler='stand', model='linreg',
                iteration='6', note='X3 with outliers. Ordinal. OHE. Poly2. Standard Scaler. Linear Re
                grid=False, grid_params='linreg', grid_cv='kfold_10', grid_verbose=3,
                save=True, export=False, config=my_config, debug=False, decimal=4,
                plot=True, perm=False, vif=False, coef=True)
```

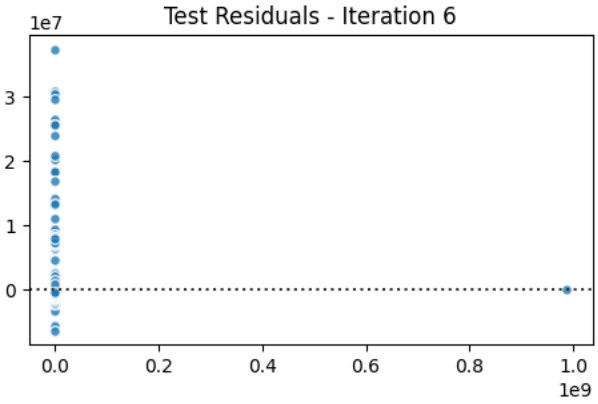
ITERATION 6 RESULTS

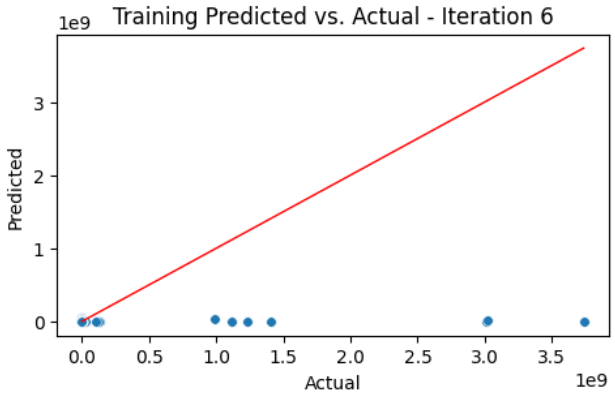
Pipeline: Transformers: ord_ohe_poly2 -> Scaler: stand -> Model: linreg
Note: X3 with outliers. Ordinal. OHE. Poly2. Standard Scaler. Linear Regression.
Aug 28, 2023 10:14 PM PST

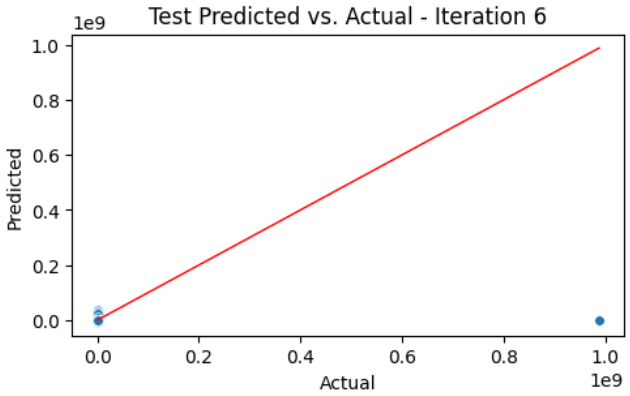
Predictions:

	Train	Test
MSE:	342,459,444,808,332.6875	25,671,639,015,069.0781
RMSE:	18,505,659.8047	5,066,718.7622
MAE:	488,792.5099	387,735.6568
R^2 Score:	0.0014	-0.0208





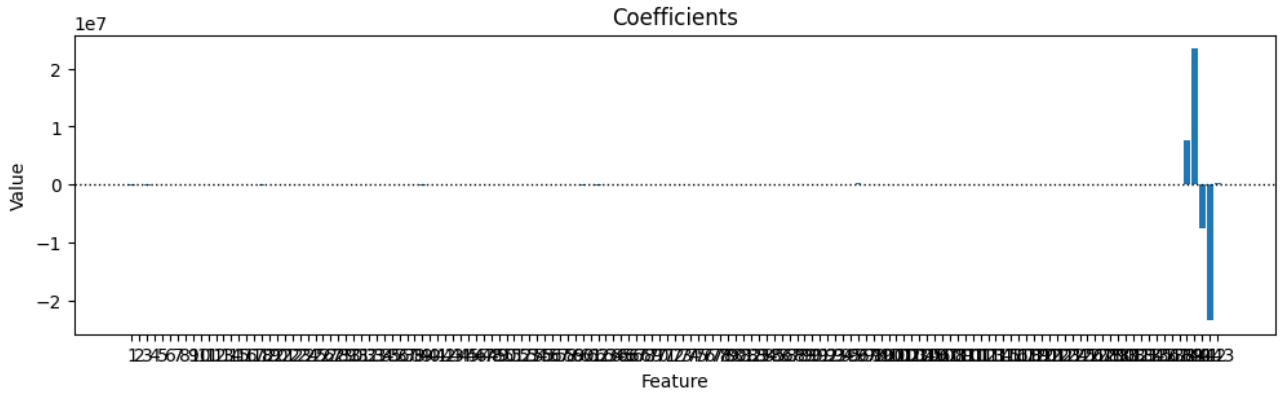




Coefficients:

	Feature	Value
1	condition	-52,548.2340
2	cylinders	91,848.0344
3	size	-73,493.5178
4	manufacturer_Not specified	-34,061.0697
5	manufacturer_acura	-8,857.3165
..	...	...
139	year	7,554,589.5418
140	odometer	23,438,238.3008
141	year^2	-7,542,878.1167
142	year odometer	-23,417,725.7555
143	odometer^2	246,875.4950

[143 rows x 2 columns]



Observation: This is with the X3 dataset, which has all the outliers intact – none are removed. I was getting worried that maybe my outlier removal was causing problems, but this proves it to not be the case. The Test R^2 is basically zero at **0.02**, it's a complete fail due to the extreme outliers. Good to know that removing them was teh right course.

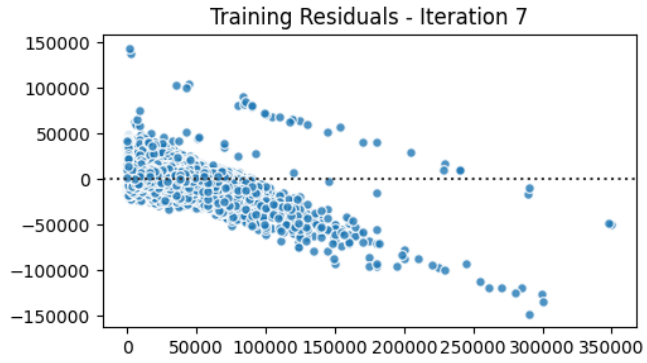
Model Iteration 7

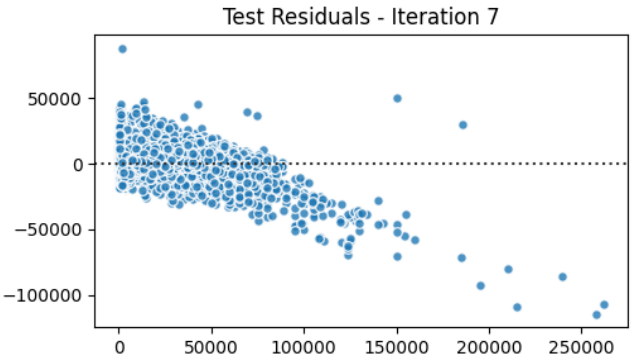
X4 no Paint/State, Manual Ord, OHE, Poly 2, Standard, LinReg

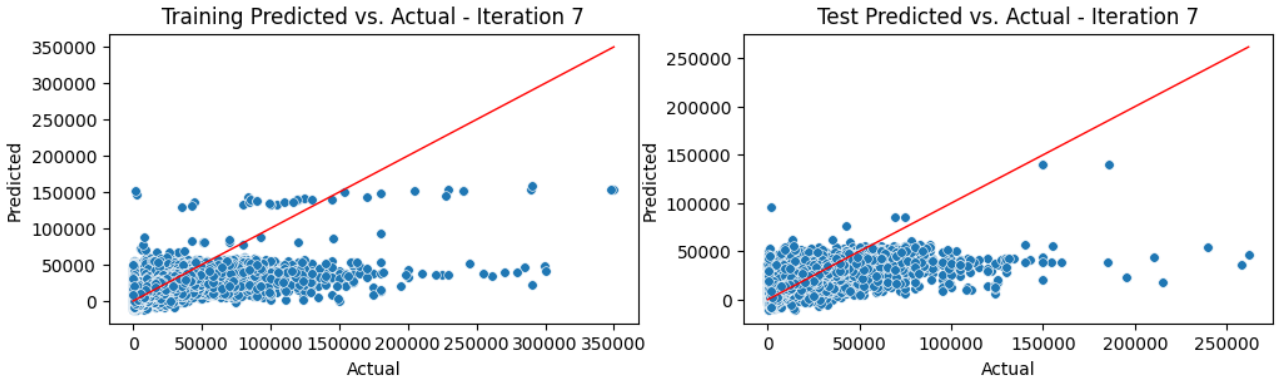
In [412…

```
i7_model = my.iterate_model(X4_train, X4_test, y4_train, y4_test,
                  transformers=['ohe2','poly2'], scaler='stand', model='linreg',
                  iteration='7', note='X4 no Paint/State. Manual Ordinal. OHE. Poly2. Standard Scaler. I
                  grid=False, grid_params='linreg', grid_cv='kfold_10', grid_verbose=3,
                  save=True, export=False, config=my_config, debug=False, decimal=4,
                  plot=True, perm=False, vif=False, coef=True)
```

```
ITERATION 7 RESULTS

Pipeline: Transformers: ohe2_poly2 -> Scaler: stand -> Model: linreg
Note: X4 no Paint/State. Manual Ordinal. OHE. Poly2. Standard Scaler. Linear Regression.
Aug 28, 2023 10:28 PM PST

Predictions:
                          Train            Test
MSE:            92,406,276.2234 91,000,540.1661
RMSE:                 9,612.8183      9,539.4203
MAE:                  5,747.6261      5,736.0661
R^2 Score:                0.5596          0.5558
```



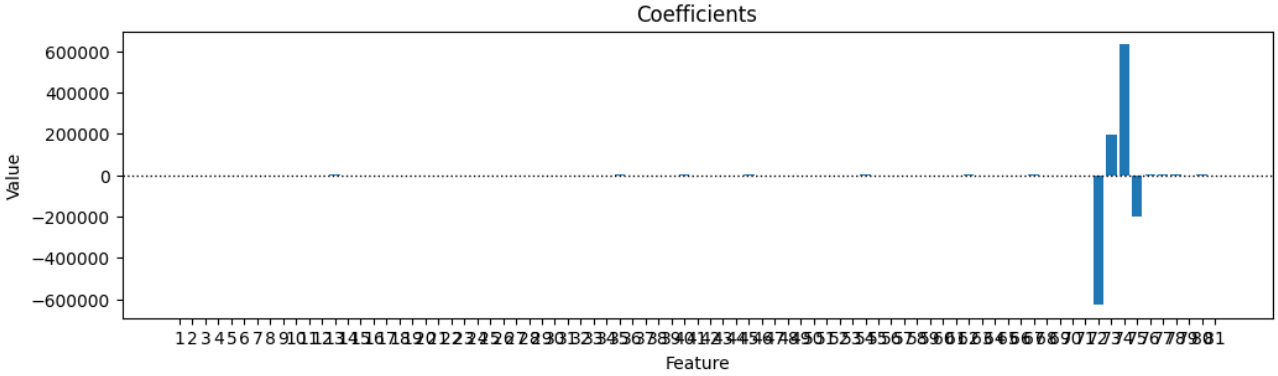




Coefficients:

	Feature	Value
1	manufacturer_Not specified	596.8432
2	manufacturer_acura	109.8032
3	manufacturer_alfa-romeo	120.8279
4	manufacturer_aston-martin	483.9967
5	manufacturer_audi	201.6896
..	...	...
77	2	666.7583
78	3	2,118.3397
79	8	-11.7458
80	10	703.7017
81	11	597.7274

[81 rows x 2 columns]



Observation: This is the X4 dataset, which was based on the manual encoding (Ordinal) I did, and dropped low correlation features: Paint Color, State. The Test R^2 is **0.55**, which is no improvement.

Model Iteration 8

X4 no Paint/State, Manual Ord, OHE, Poly 3, Standard, LinReg

In [413…

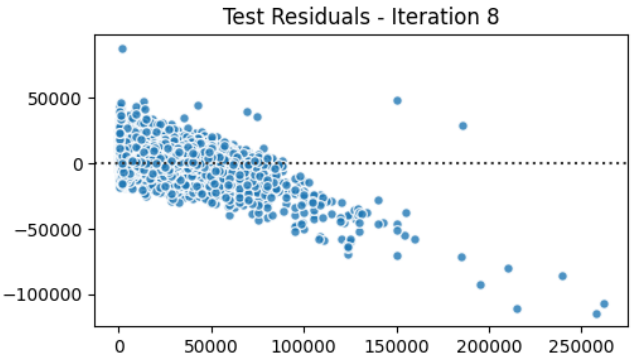
```
i8_model = my.iterate_model(X4_train, X4_test, y4_train, y4_test,
                transformers=['ohe2','poly3'], scaler='stand', model='linreg',
                iteration='8', note='X4 no Paint/State. Manual Ordinal. OHE. Poly3. Standard Scaler. I
                grid=False, grid_params='linreg', grid_cv='kfold_10', grid_verbose=3,
                save=True, export=False, config=my_config, debug=False, decimal=4,
                plot=True, perm=False, vif=False, coef=True)
```

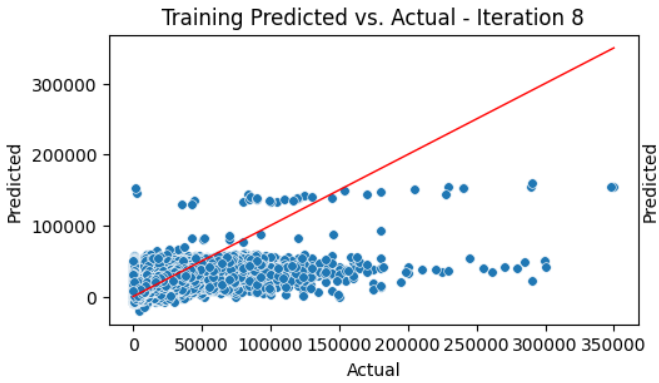
ITERATION 8 RESULTS

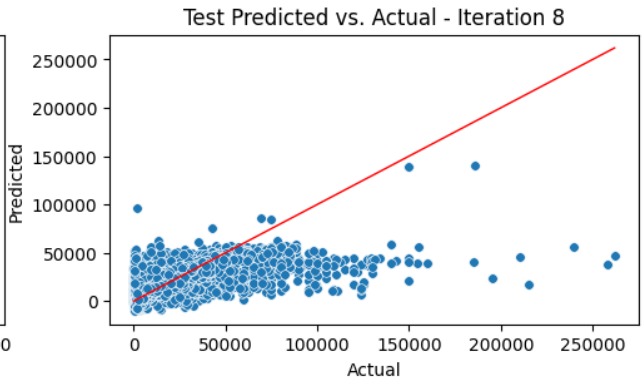
Pipeline: Transformers: ohe2_poly3 -> Scaler: stand -> Model: linreg
Note: X4 no Paint/State. Manual Ordinal. OHE. Poly3. Standard Scaler. Linear Regression.
Aug 28, 2023 10:31 PM PST

Predictions:

	Train	Test
MSE:	91,324,878.3378	90,271,236.2149
RMSE:	9,556.4051	9,501.1176
MAE:	5,717.3026	5,712.3129
R^2 Score:	0.5647	0.5593



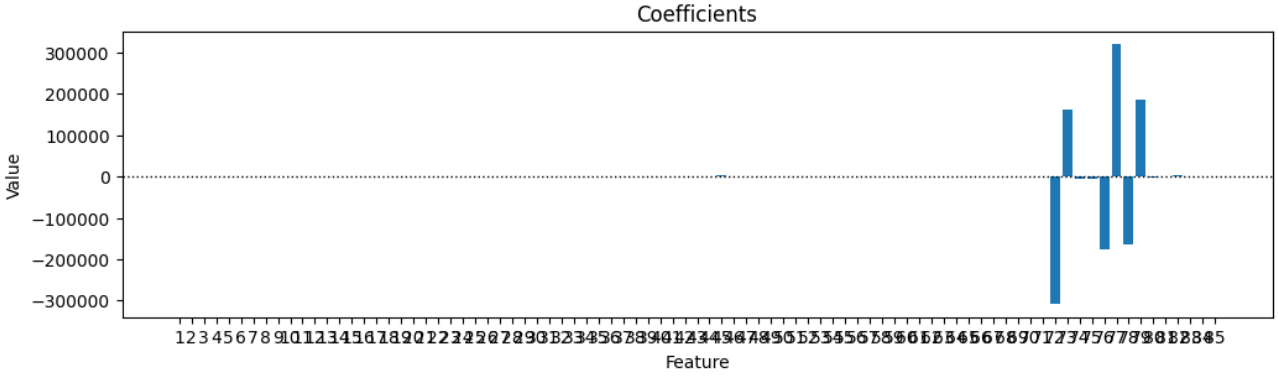




Coefficients:

	Feature	Value
1	manufacturer_Not specified	562.5142
2	manufacturer_acura	102.0692
3	manufacturer_alfa-romeo	104.8897
4	manufacturer_aston-martin	478.7358
5	manufacturer_audi	213.8124
..	...	...
81	2	711.2666
82	3	2,078.0923
83	8	4.3973
84	10	708.3656
85	11	603.0043

[85 rows x 2 columns]



Observation: This is the X4 dataset, which was based on the manual encoding (Ordinal) I did, and dropped low correlation features: Paint Color, State. The Test R^2 is **0.56**, which is barely an improvememnt due to the Polynomial degree 3 vs. 2 previously.

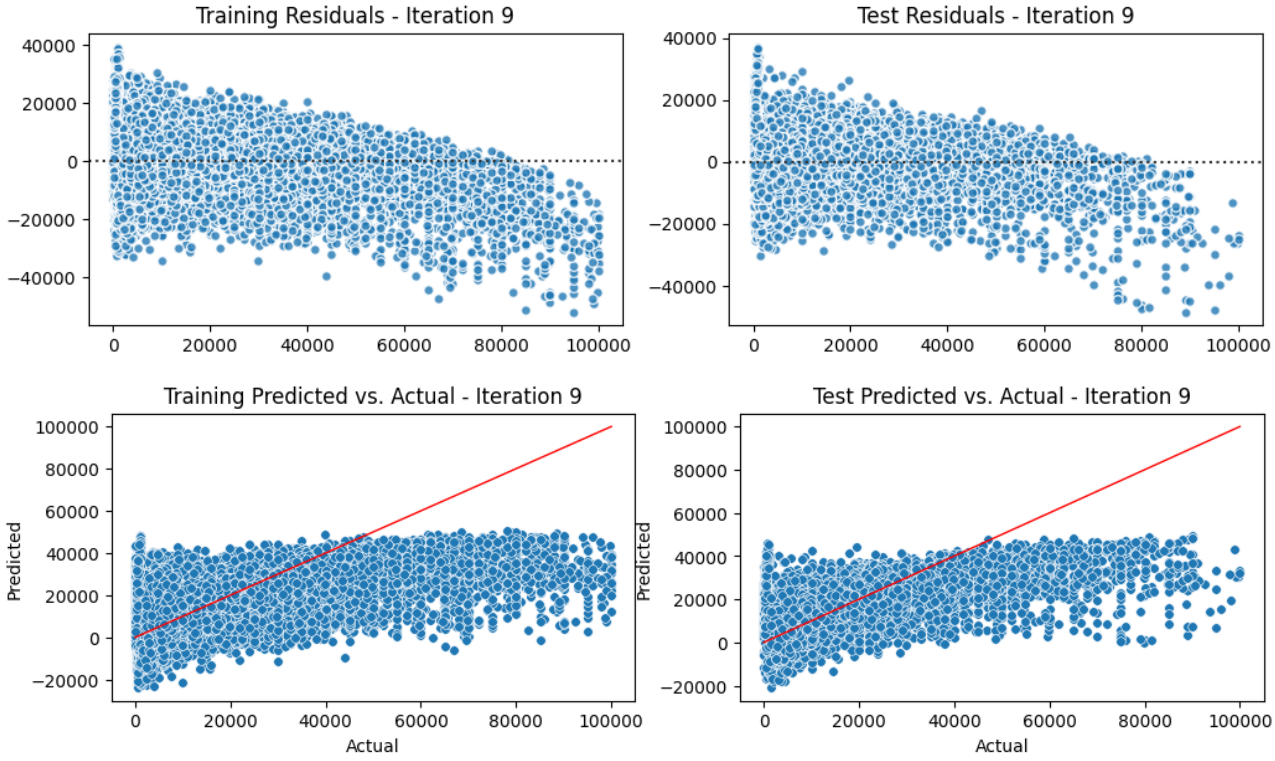
Model Iteration 9

X5 no Paint/State. Price < $100K, Odometer < 400k. Manual Ord, OHE, LinReg

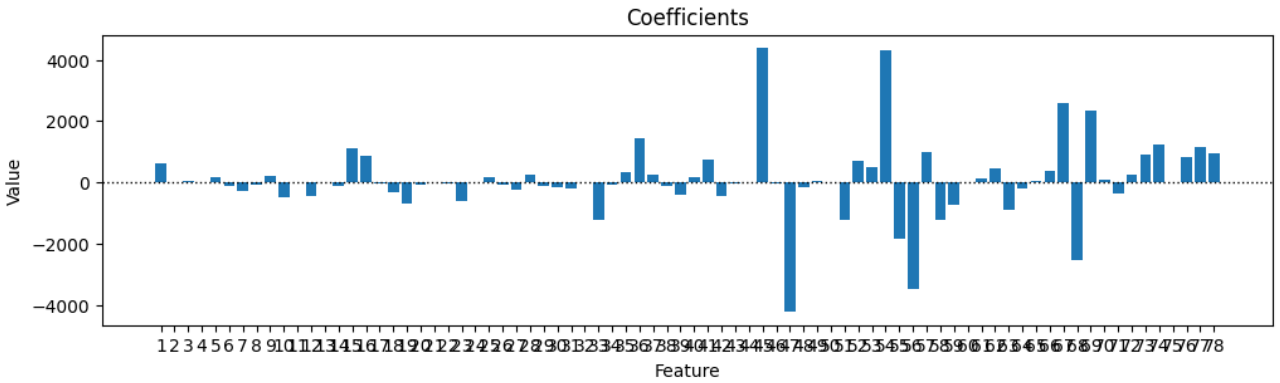
In [415…

```
i9_model = my.iterate_model(X5_train, X5_test, y5_train, y5_test,
                  transformers=['ohe2'], model='linreg',
                  iteration='9', note='X5 Price < $100k, Odometer < 400k. Manual Ordinal. OHE. Linear Re
                  grid=False, grid_params='linreg', grid_cv='kfold_10', grid_verbose=3,
                  save=True, export=False, config=my_config, debug=False, decimal=4,
                  plot=True, perm=False, vif=False, coef=True)
```

```
ITERATION 9 RESULTS

Pipeline: Transformers: ohe2 -> Model: linreg
Note: X5 Price < $100k, Odometer < 400k. Manual Ordinal. OHE. Linear Regression.
Aug 28, 2023 10:43 PM PST

Predictions:
                          Train            Test
MSE:             91,466,320.1586 93,509,202.3961
RMSE:                9,563.8026      9,670.0156
MAE:                 6,553.6878      6,583.5307
R^2 Score:               0.4936          0.4838
```



```
Coefficients:
                        Feature      Value
1    manufacturer_Not specified   619.2595
2            manufacturer_acura   -14.6592
3       manufacturer_alfa-romeo    27.3980
4     manufacturer_aston-martin    11.2152
5              manufacturer_audi   160.3318
..                           ...        ...
74                             3 1,233.6340
75                             5    -0.0959
76                             8   804.9570
77                            10 1,137.5576
78                            11   964.0699

[78 rows x 2 columns]
```



Observation: This is the X5 dataset, which was based on the manual encoding (Ordinal) I did, and dropped low correlation features: Paint Color, State. I also applied stricter upper thresholds: Price < \$100k, Odometer < 400k. The Test R^2 is **0.48**, which is going in the wrong direction. However, the plots look much better! I think the larger (less frequent) prices were wreaking havoc.

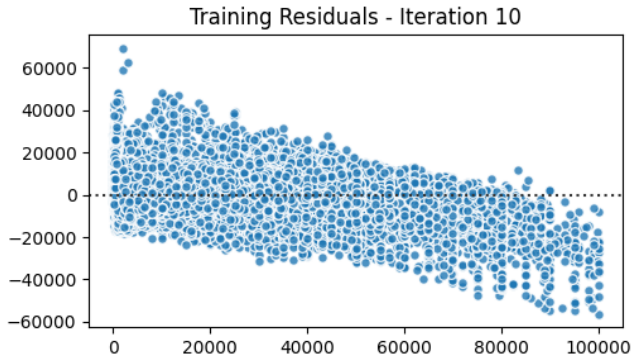
Model Iteration 10

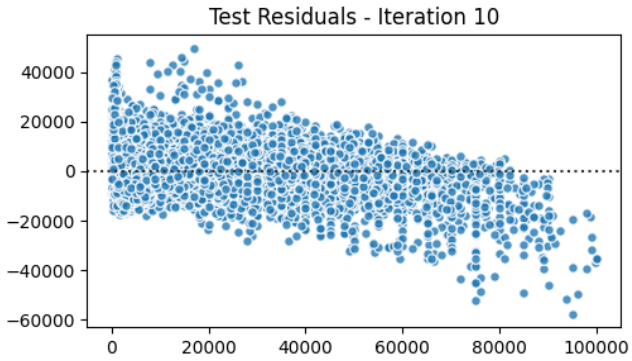
X5 no Paint/State. Price < $100K, Odometer < 400k. Manual Ord, OHE, Poly 2, Standard, LinReg

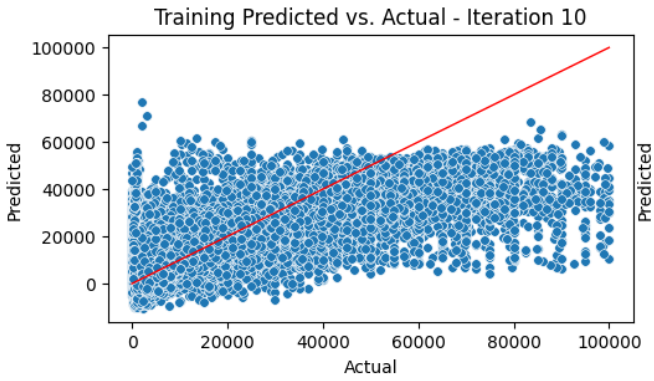
In [417…

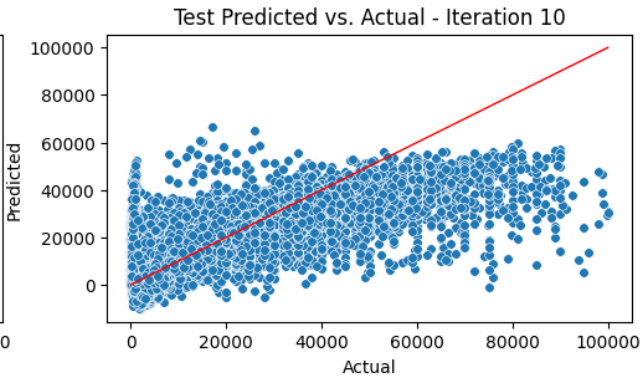
```
i10_model = my.iterate_model(X5_train, X5_test, y5_train, y5_test,
                 transformers=['ohe2','poly2'], scaler='stand', model='linreg',
                 iteration='10', note='X5 Price < $100k, Odometer < 400k. Manual Ordinal. OHE. Poly2. S
                 grid=False, grid_params='linreg', grid_cv='kfold_10', grid_verbose=3,
                 save=True, export=False, config=my_config, debug=False, decimal=4,
                 plot=True, perm=False, vif=False, coef=True)
```

```
ITERATION 10 RESULTS

Pipeline: Transformers: ohe2_poly2 -> Scaler: stand -> Model: linreg
Note: X5 Price < $100k, Odometer < 400k. Manual Ordinal. OHE. Poly2. Standard Scaler. Linear Regressio
n.
Aug 28, 2023 10:53 PM PST

Predictions:
                          Train            Test
MSE:             71,682,968.1867 73,580,803.8148
RMSE:                 8,466.5795      8,577.9254
MAE:                  5,478.2419      5,536.6706
R^2 Score:                0.6031          0.5938
```





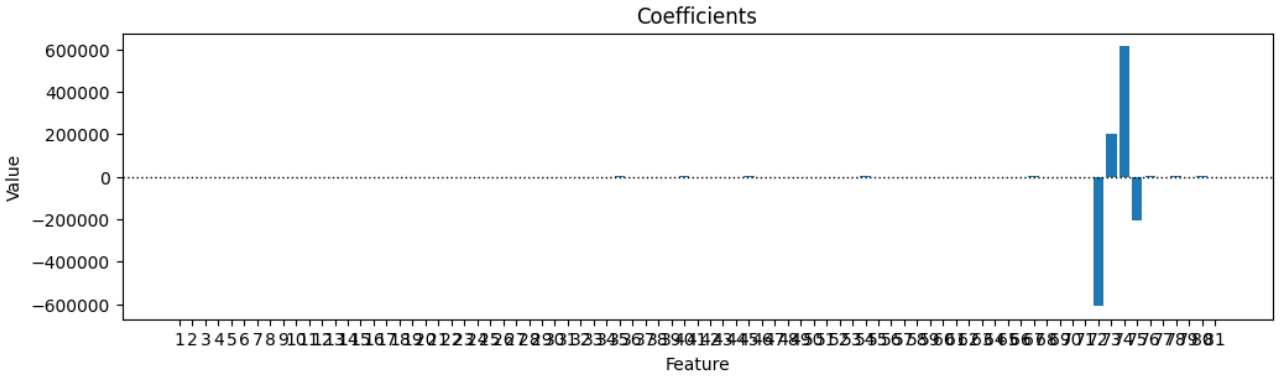




Coefficients:

	Feature	Value
1	manufacturer_Not specified	264.2864
2	manufacturer_acura	115.5425
3	manufacturer_alfa-romeo	99.6788
4	manufacturer_aston-martin	227.9650
5	manufacturer_audi	210.7082
..	...	...
77	2	651.4992
78	3	1,867.2722
79	8	46.2348
80	10	716.0830
81	11	619.3282

[81 rows x 2 columns]



Observation: This is the X5 dataset, which was based on the manual encoding (Ordinal) I did, and dropped low correlation features: Paint Color, State. I also applied stricter upper thresholds: Price < \$100k, Odometer < 400k. The Test R^2 is **0.59**, which improved thanks to Polynomial degree 2. Going in the right direction at least now.

Model Iteration 11

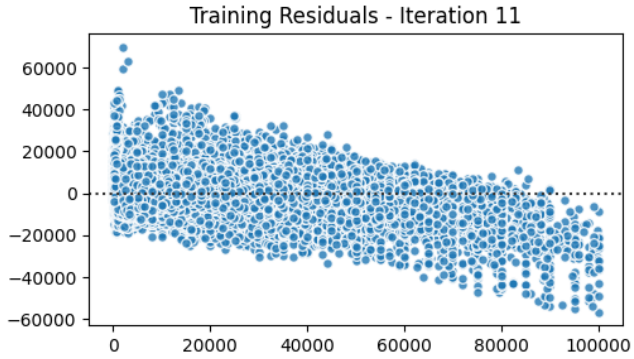
X5 no Paint/State. Price < $100K, Odometer < 400k. Manual Ord, OHE, Poly 3, Standard, LinReg

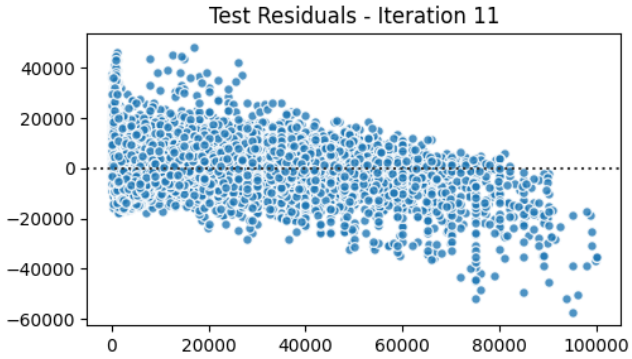
In [418…

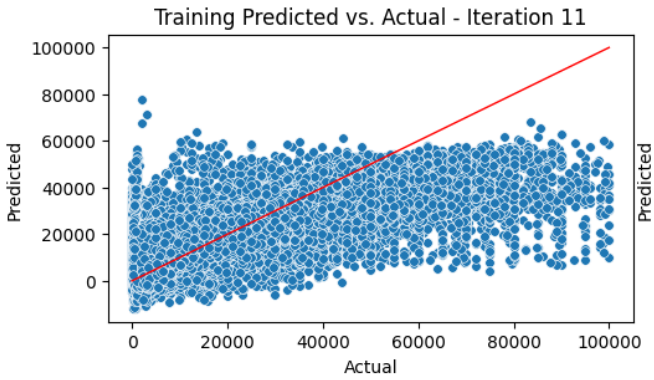
```
i11_model = my.iterate_model(X5_train, X5_test, y5_train, y5_test,
                transformers=['ohe2','poly3'], scaler='stand', model='linreg',
```

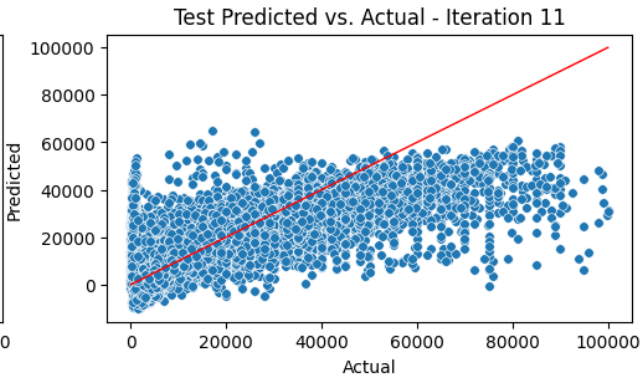
```
                  iteration='11', note='X5 Price < $100k, Odometer < 400k. Manual Ordinal. OHE. Poly3. S
                  grid=False, grid_params='linreg', grid_cv='kfold_10', grid_verbose=3,
                  save=True, export=False, config=my_config, debug=False, decimal=4,
                  plot=True, perm=False, vif=False, coef=True)
```

```
ITERATION 11 RESULTS

Pipeline: Transformers: ohe2_poly3 -> Scaler: stand -> Model: linreg
Note: X5 Price < $100k, Odometer < 400k. Manual Ordinal. OHE. Poly3. Standard Scaler. Linear Regressio
n.
Aug 28, 2023 10:57 PM PST

Predictions:
                         Train            Test
MSE:            71,426,631.0234 73,283,882.9740
RMSE:                8,451.4278      8,560.6006
MAE:                 5,464.6993      5,524.6137
R^2 Score:               0.6045          0.5954
```

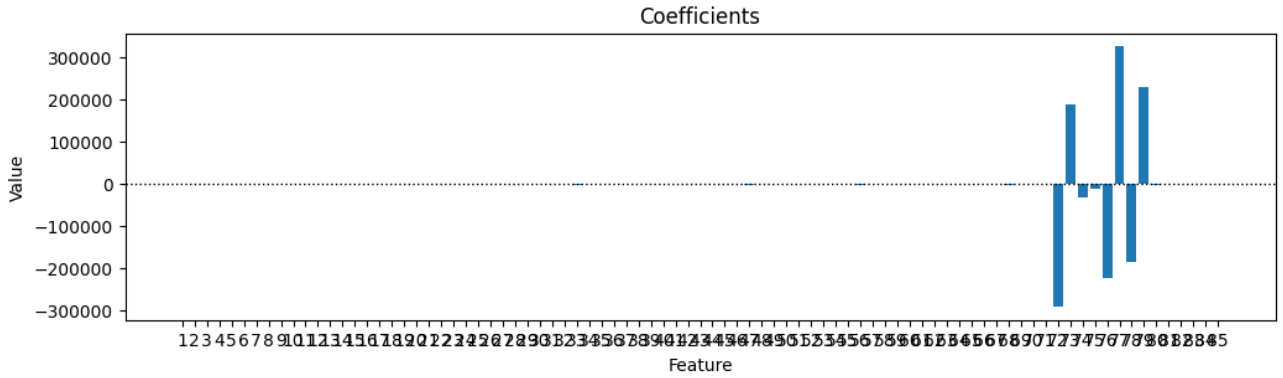









```
Coefficients:
                        Feature       Value
1    manufacturer_Not specified    254.2940
2            manufacturer_acura    109.9268
3       manufacturer_alfa-romeo     92.9394
4     manufacturer_aston-martin    227.2961
5              manufacturer_audi    219.6615
..                           ...         ...
81                            2    655.3785
82                            3  1,846.4048
83                            8     52.6005
84                           10    717.3639
85                           11    621.4886

[85 rows x 2 columns]
```



Observation: This is the X5 dataset, which was based on the manual encoding (Ordinal) I did, and dropped low correlation features: Paint Color, State. I also applied stricter upper thresholds: Price < \$100k, Odometer < 400k. The Test R² is **0.60**, which only improved slightly with Polynomial degree 3.

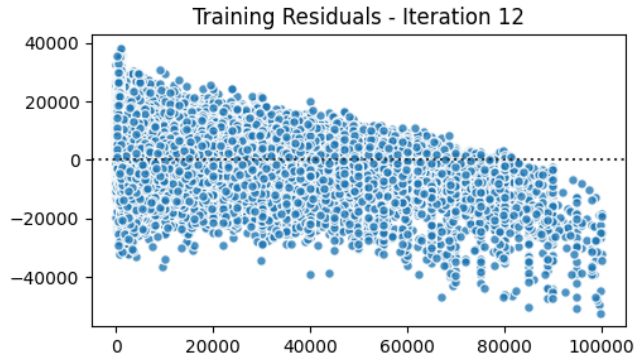
Model Iteration 12

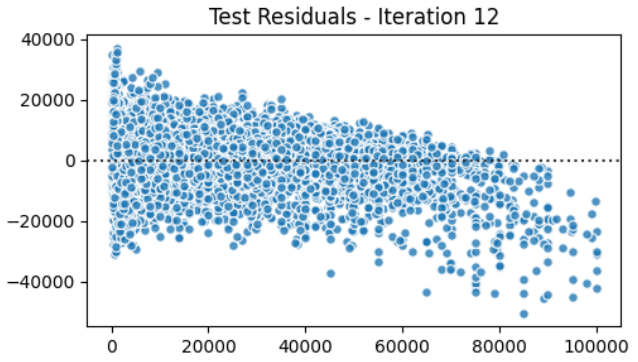
X6 no Outlier Removal via Cluster. \$10 < Price < \$100K, 10 < Odometer < 400k. Ord, OHE, LinReg

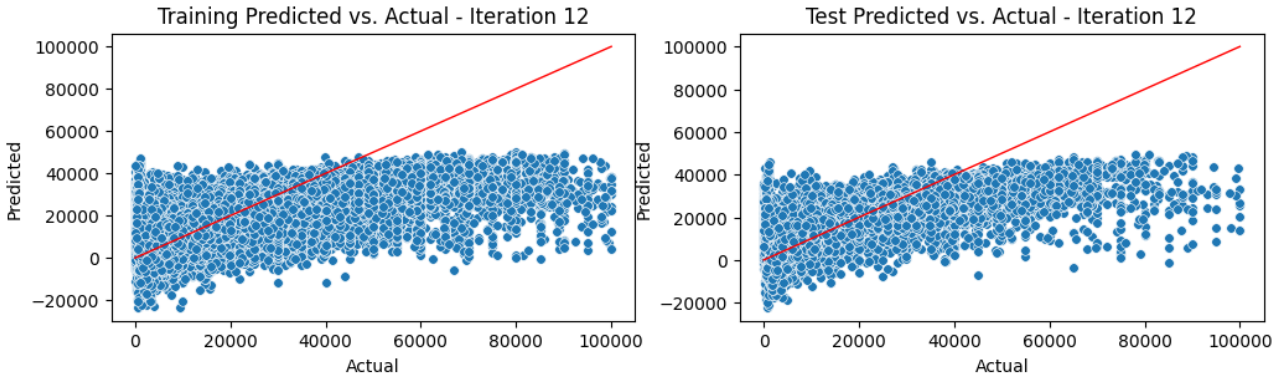
In [419…

```
i12_model = my.iterate_model(X6_train, X6_test, y6_train, y6_test,
                 transformers=['ord','ohe2'], model='linreg',
                 iteration='12', note='X6 no Outlier removal via Cluster. $10 < Price < $100K, 10 < Odo
                 grid=False, grid_params='linreg', grid_cv='kfold_10', grid_verbose=3,
                 save=True, export=False, config=my_config, debug=False, decimal=4,
                 plot=True, perm=False, vif=False, coef=True)
```

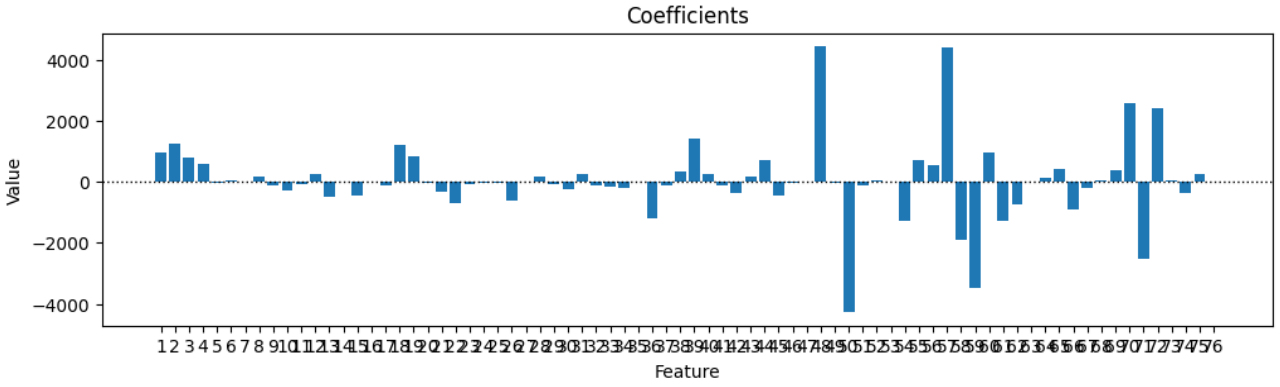
```
ITERATION 12 RESULTS

Pipeline: Transformers: ord_ohe2 -> Model: linreg
Note: X6 no Outlier removal via Cluster. $10 < Price < $100K, 10 < Odometer < 400k. Ordinal. OHE. Line
ar Regression.
Aug 28, 2023 11:07 PM PST

Predictions:
                         Train            Test
MSE:           93,461,430.6622 93,388,622.0039
RMSE:                9,667.5452      9,663.7789
MAE:                 6,616.0075      6,605.4700
R^2 Score:               0.4837          0.4815
```








```
Coefficients:
                        Feature      Value
1                     condition   946.9255
2                     cylinders 1,257.8133
3                          size   805.4291
4   manufacturer_Not specified   575.2591
5            manufacturer_acura   -19.5726
..                          ...        ...
72                   type_truck 2,405.3151
73                     type_van    51.1687
74                   type_wagon  -371.1531
75                            0   254.6128
76                            5    -0.0954

[76 rows x 2 columns]
```



Observation: This is the X6 dataset, which is based on the first pass of cleaning and does NOT remove outliers via Clustering. I also applied stricter lower thresholds: Price > \$10, Odometer > 10, and upper thresholds: Price < \$100k, Odometer < 400k. The Test R^2 is **0.48** with just Ordinal encoding, OHE, and Linear Regression. I thought my removal of outliers via clustering was causing a problem, but maybe not.

Model Iteration 13

X6 no Outlier Removal via Cluster. \$10 < Price < \$100K, 10 < Odometer < 400k. Ord, OHE, Poly 2, Standard, LinReg

In [421…

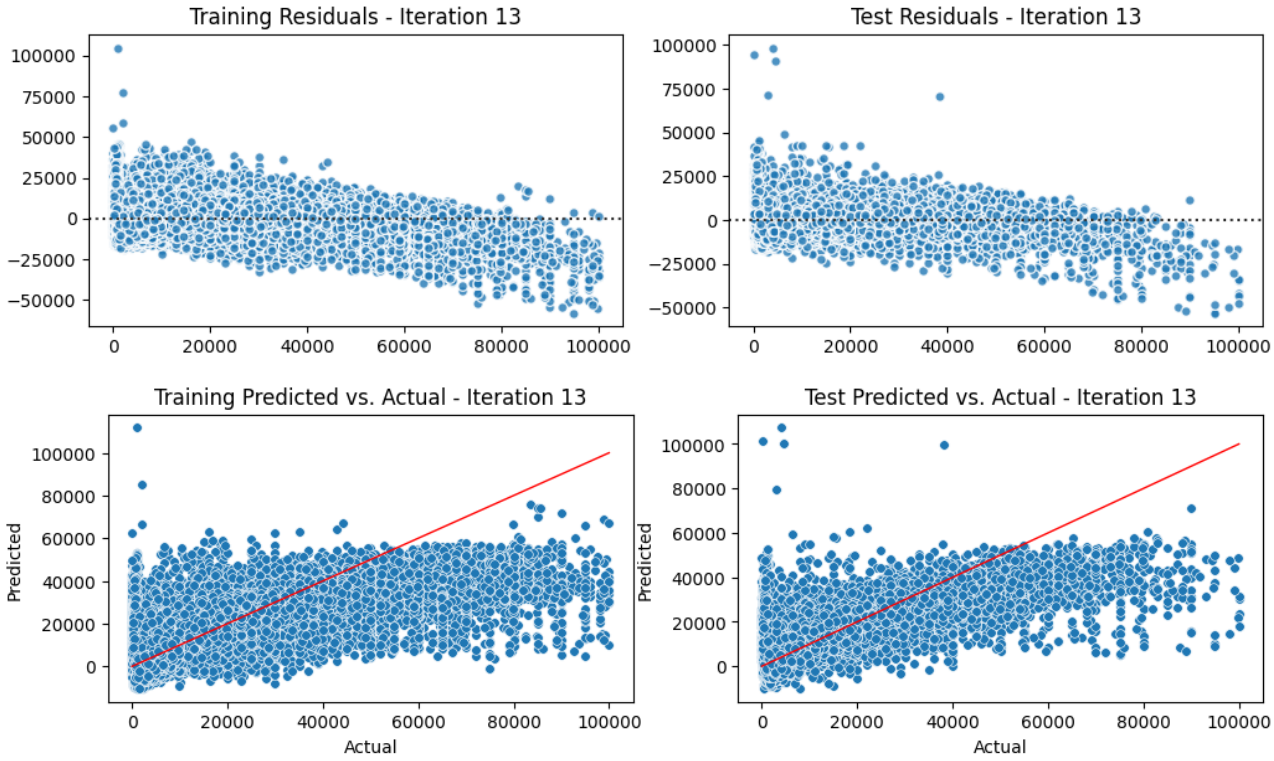
```
i13_model = my.iterate_model(X6_train, X6_test, y6_train, y6_test,
                  transformers=['ord','ohe2','poly2'], scaler='stand', model='linreg',
                  iteration='13', note='X6 no Outlier removal via Cluster. 10 < Price < 100K, 10 < Odome
                  grid=False, grid_params='linreg', grid_cv='kfold_10', grid_verbose=3,
                  save=True, export=False, config=my_config, debug=False, decimal=4,
                  plot=True, perm=False, vif=False, coef=True)
```

ITERATION 13 RESULTS

Pipeline: Transformers: ord_ohe2_poly2 -> Scaler: stand -> Model: linreg
Note: X6 no Outlier removal via Cluster. 10 < Price < 100K, 10 < Odometer < 400k. Ordinal. OHE. Poly2. Standard. Linear Regression.
Aug 28, 2023 11:11 PM PST

Predictions:

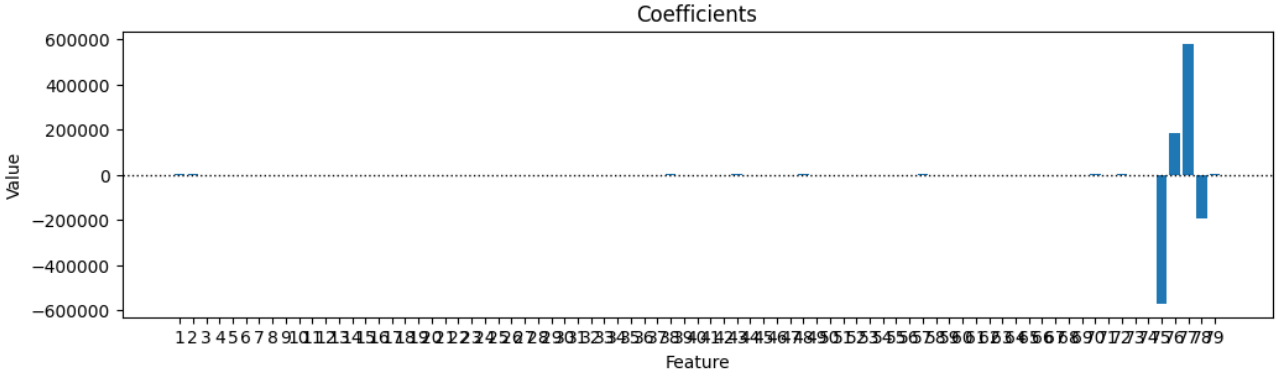
	Train	Test
MSE:	74,233,053.7068	76,346,117.0804
RMSE:	8,615.8606	8,737.6265
MAE:	5,582.4804	5,605.7400
R^2 Score:	0.5899	0.5761



Coefficients:

	Feature	Value
1	condition	721.0546
2	cylinders	1,928.8549
3	size	64.0270
4	manufacturer_Not specified	203.9977
5	manufacturer_acura	103.0017
..	...	...
75	year	-573,245.0091
76	odometer	184,635.7963
77	year^2	577,686.7394
78	year odometer	-191,431.4608
79	odometer^2	3,144.7065

[79 rows x 2 columns]



Observation: This is the X6 dataset, which is based on the first pass of cleaning and does NOT remove outliers via Clustering. I also applied stricter lower thresholds: Price > \$10, Odometer > 10, and upper thresholds: Price < \$100k, Odometer < 400k. The Test R^2 is **0.58** now with the additional of Polynomial degree 2. Heading in the right direction again.

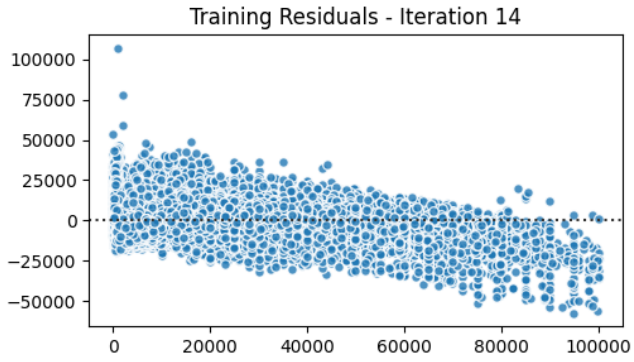
Model Iteration 14

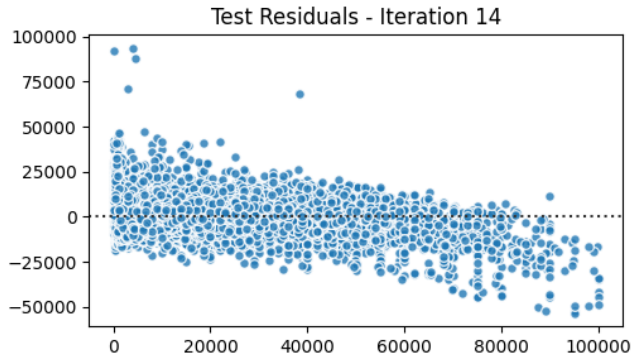
X6 no Outlier Removal via Cluster. \$10 < Price < \$100K, 10 < Odometer < 400k. Ord, OHE, Poly 3, Standard, LinReg

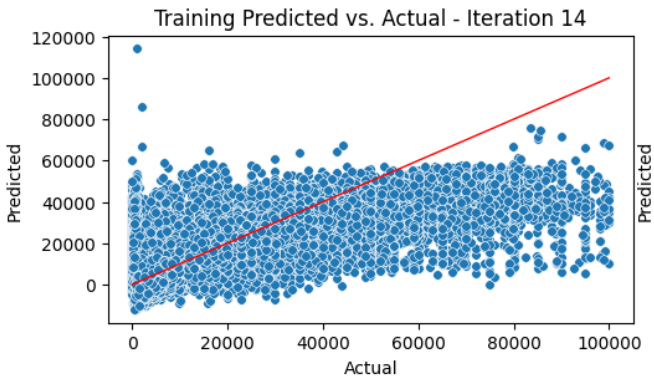
In [422…

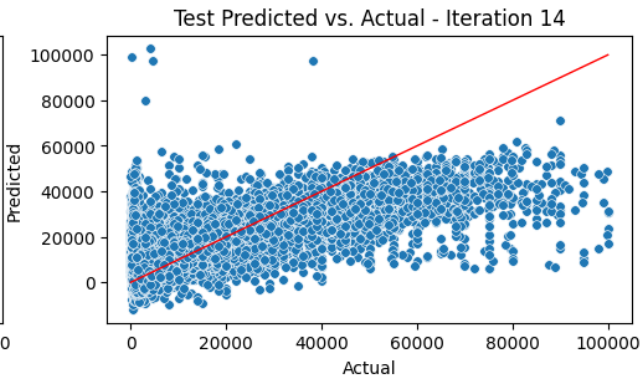
```
i14_model = my.iterate_model(X6_train, X6_test, y6_train, y6_test,
                  transformers=['ord','ohe2','poly3'], scaler='stand', model='linreg',
                  iteration='14', note='X6 no Outlier removal via Cluster. 10 < Price < 100K, 10 < Odome
                  grid=False, grid_params='linreg', grid_cv='kfold_10', grid_verbose=3,
                  save=True, export=False, config=my_config, debug=False, decimal=4,
                  plot=True, perm=False, vif=False, coef=True)
```

```
ITERATION 14 RESULTS

Pipeline: Transformers: ord_ohe2_poly3 -> Scaler: stand -> Model: linreg
Note: X6 no Outlier removal via Cluster. 10 < Price < 100K, 10 < Odometer < 400k. Ordinal. OHE. Poly3.
Standard. Linear Regression.
Aug 28, 2023 11:16 PM PST

Predictions:
                          Train            Test
MSE:             73,979,981.9299 76,037,667.8583
RMSE:                8,601.1617      8,719.9580
MAE:                 5,569.1316      5,591.6461
R^2 Score:               0.5913          0.5779
```

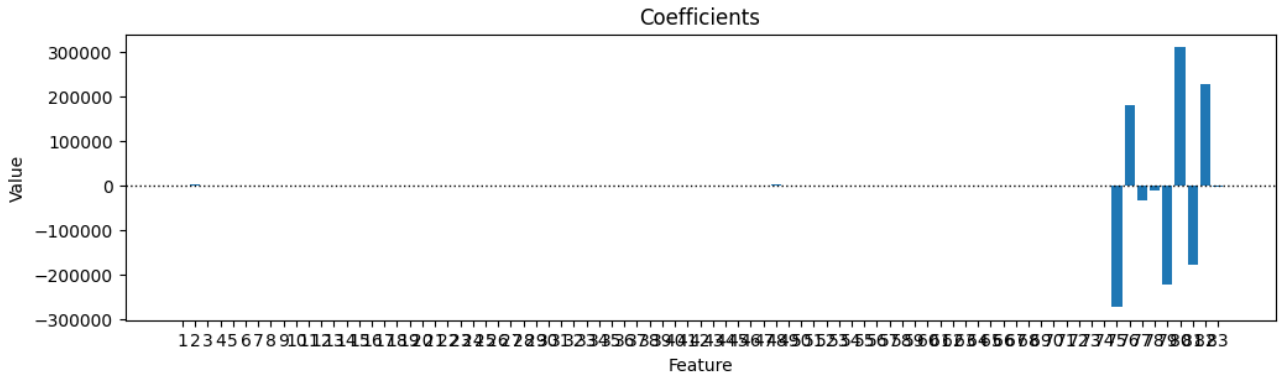








```
Coefficients:
                        Feature         Value
1                     condition      725.6215
2                     cylinders    1,906.4510
3                          size       70.8060
4   manufacturer_Not specified      192.5332
5            manufacturer_acura       98.8084
..                          ...           ...
79                  odometer^2 -224,009.6327
80                      year^3  311,282.9404
81             year^2 odometer -177,755.7247
82             year odometer^2  229,540.9371
83                  odometer^3   -1,535.9148

[83 rows x 2 columns]
```



Observation: This is the X6 dataset, which is based on the first pass of cleaning and does NOT remove outliers via Clustering. I also applied stricter lower thresholds: Price > \$10, Odometer > 10, and upper thresholds: Price < \$100k, Odometer < 400k. The Test R^2 is **0.58** still, no difference by changing to Polynomial degree 3.

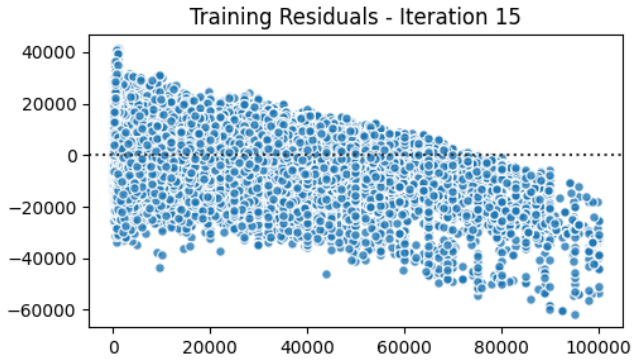
Model Iteration 15

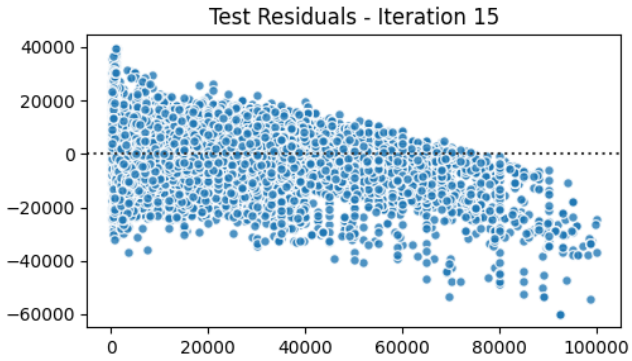
X7 no Duplicate or Outlier Removal via Cluster. \$10 < Price < \$100K, 10 < Odometer < 400k. Year > 1960. Ord, OHE, LinReg

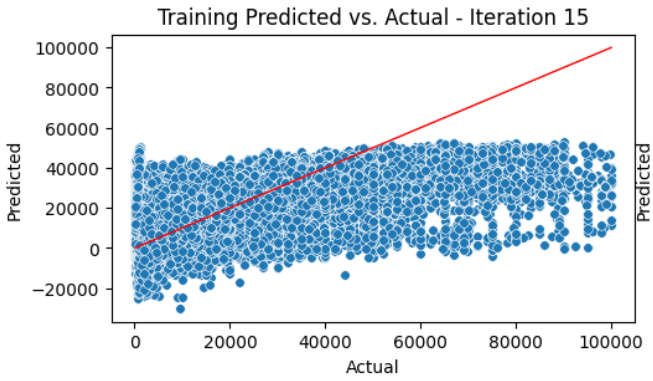
In [423…

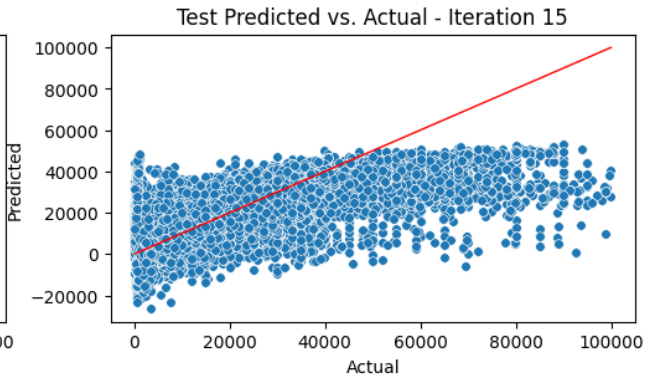
```
i15_model = my.iterate_model(X7_train, X7_test, y7_train, y7_test,
                 transformers=['ord','ohe2'], model='linreg',
                 iteration='15', note='X7 no Duplicate or Outlier removal via Cluster. 10 < Price < 10(
                 grid=False, grid_params='linreg', grid_cv='kfold_10', grid_verbose=3,
                 save=True, export=False, config=my_config, debug=False, decimal=4,
                 plot=True, perm=False, vif=False, coef=True)
```

```
ITERATION 15 RESULTS

Pipeline: Transformers: ord_ohe2 -> Model: linreg
Note: X7 no Duplicate or Outlier removal via Cluster. 10 < Price < 100K, 10 < Odometer < 400k, Year >
1960. Ordinal. OHE. Linear Regression.
Aug 28, 2023 11:49 PM PST

Predictions:
                          Train            Test
MSE:            88,171,916.5909 88,994,396.2333
RMSE:                9,389.9902      9,433.6841
MAE:                 6,552.2754      6,564.9908
R^2 Score:               0.5681          0.5657
```





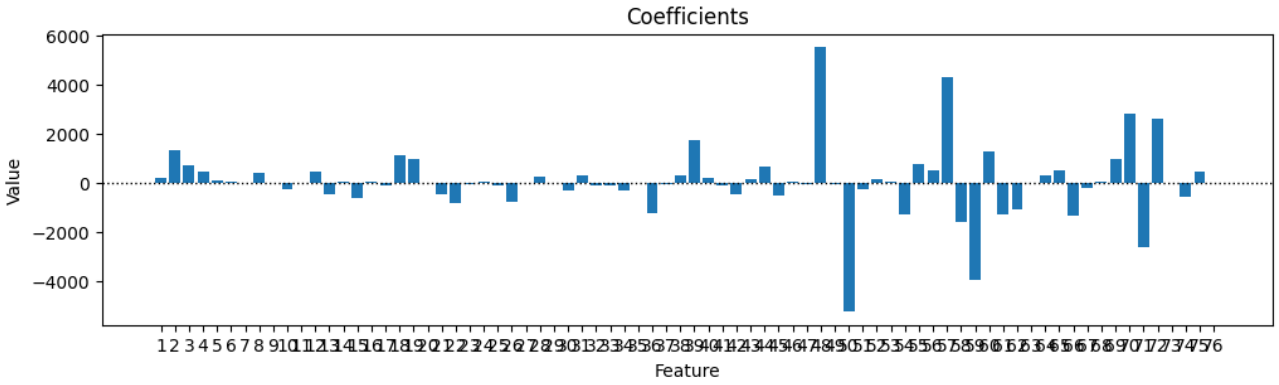




Coefficients:

	Feature	Value
1	condition	192.8236
2	cylinders	1,297.2238
3	size	723.4350
4	manufacturer_Not specified	444.5683
5	manufacturer_acura	71.7229
..	...	...
72	type_truck	2,620.5668
73	type_van	-6.2472
74	type_wagon	-591.7360
75	0	447.1995
76	5	-0.0935

[76 rows x 2 columns]



Observation: This is the X7 dataset, which does NOT remove duplicates, and does NOT remove outliers via Clustering. I also applied the same stricter thresholds: Price > \$10, Odometer > 10, and upper thresholds: Price < \$100k, Odometer < 400k, and added Year > 1960. The Test R^2 is **0.57** without any polynomials.

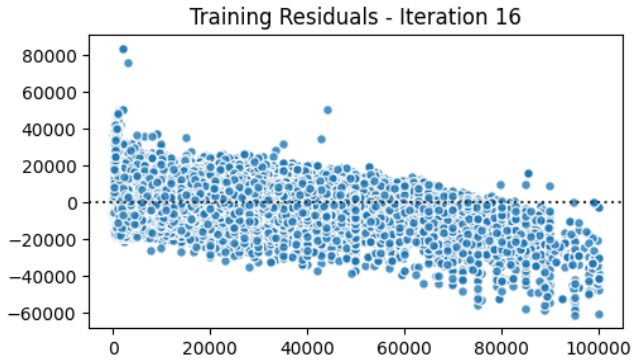
Model Iteration 16

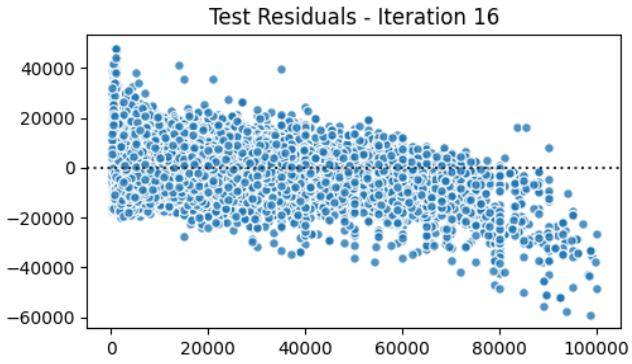
X7 no Duplicate or Outlier Removal via Cluster. \$10 < Price < \$100K, 10 < Odometer < 400k. Year > 1960. Ord, OHE, Poly 2, Standard, LinReg

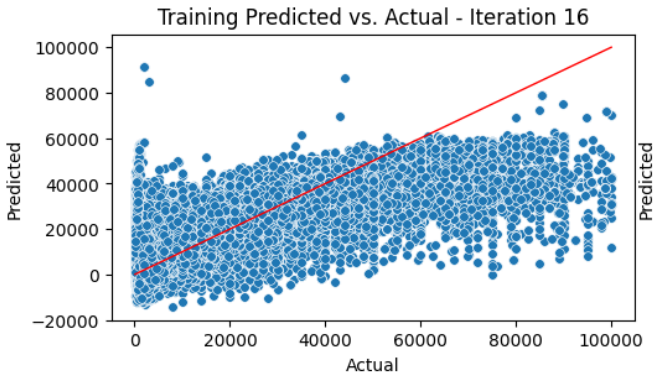
In [424…

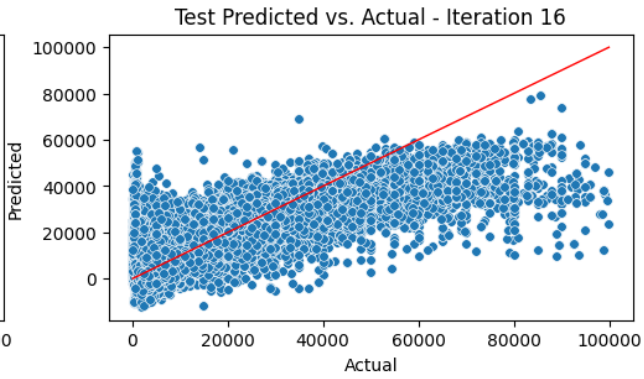
```
i16_model = my.iterate_model(X7_train, X7_test, y7_train, y7_test,
                  transformers=['ord','ohe2','poly2'], scaler='stand', model='linreg',
                  iteration='16', note='X7 no Duplicate or Outlier removal via Cluster. 10 < Price < 10(
                  grid=False, grid_params='linreg', grid_cv='kfold_10', grid_verbose=3,
                  save=True, export=False, config=my_config, debug=False, decimal=4,
                  plot=True, perm=False, vif=False, coef=True)
```

```
ITERATION 16 RESULTS

Pipeline: Transformers: ord_ohe2_poly2 -> Scaler: stand -> Model: linreg
Note: X7 no Duplicate or Outlier removal via Cluster. 10 < Price < 100K, 10 < Odometer < 400k, Year >
1960. Ordinal. OHE. Poly2. Standard. Linear Regression.
Aug 28, 2023 11:54 PM PST

Predictions:
                          Train            Test
MSE:            69,143,493.1642 69,618,322.4992
RMSE:                8,315.2567      8,343.7595
MAE:                 5,602.1216      5,604.8387
R^2 Score:               0.6613          0.6602
```

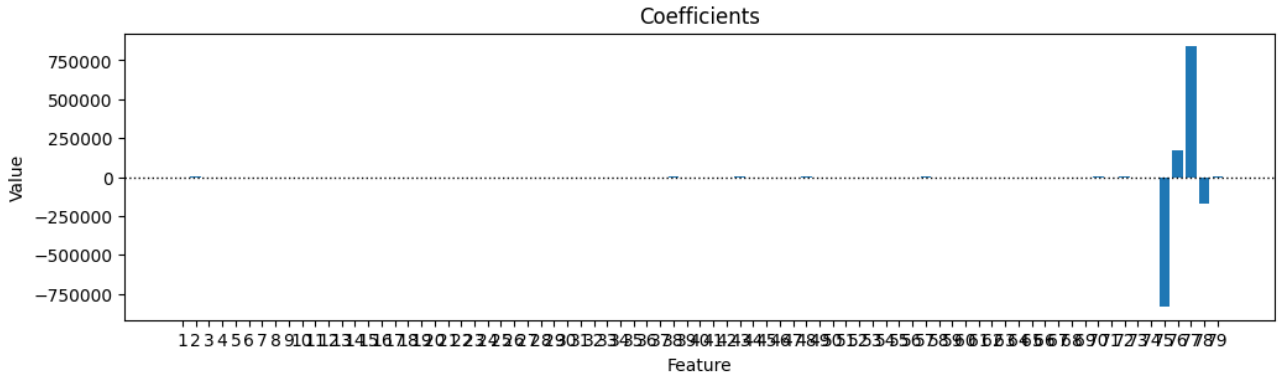









```
Coefficients:
                       Feature         Value
1                    condition      569.8368
2                    cylinders    1,784.5290
3                         size      137.2377
4   manufacturer_Not specified       30.7909
5            manufacturer_acura      176.9266
..                          ...           ...
75                        year -833,270.6731
76                    odometer  168,759.0500
77                      year^2  839,065.1136
78               year odometer -174,448.8266
79                  odometer^2    2,434.0733

[79 rows x 2 columns]
```



Observation: This is the X7 dataset, which does NOT remove duplicates, and does NOT remove outliers via Clustering. I also applied the same stricter thresholds: Price > \$10, Odometer > 10, and upper thresholds: Price < \$100k, Odometer < 400k, and added Year > 1960. The Test R^2 is **0.66** now with Polynomial degree 2. This is an improvement at least!

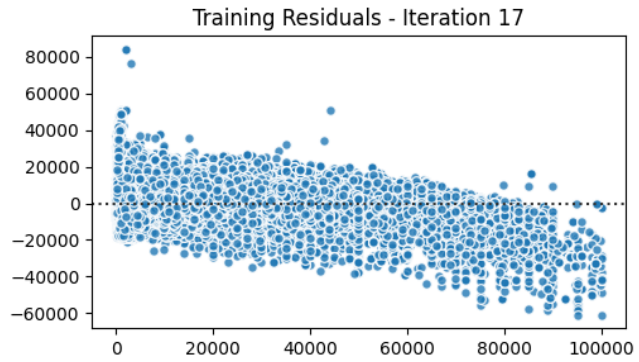
Model Iteration 17

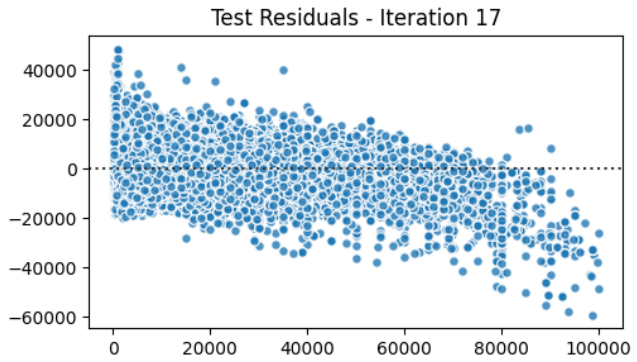
X7 no Duplicate or Outlier Removal via Cluster. \$10 < Price < \$100K, 10 < Odometer < 400k. Year > 1960. Ord, OHE, Poly 3, Standard, LinReg

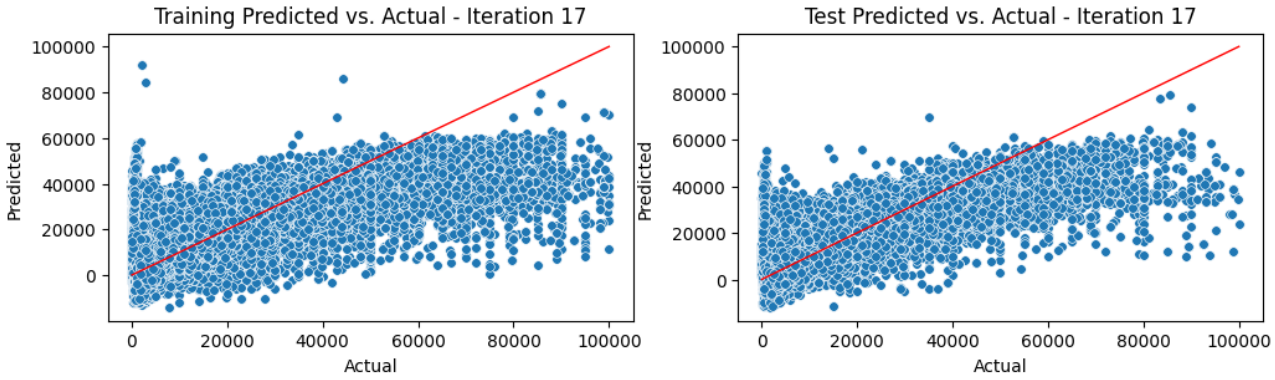
In [425…

```
i17_model = my.iterate_model(X7_train, X7_test, y7_train, y7_test,
                 transformers=['ord','ohe2','poly3'], scaler='stand', model='linreg',
                 iteration='17', note='X7 no Duplicate or Outlier removal via Cluster. 10 < Price < 10(
                 grid=False, grid_params='linreg', grid_cv='kfold_10', grid_verbose=3,
                 save=True, export=False, config=my_config, debug=False, decimal=4,
                 plot=True, perm=False, vif=False, coef=True)
```

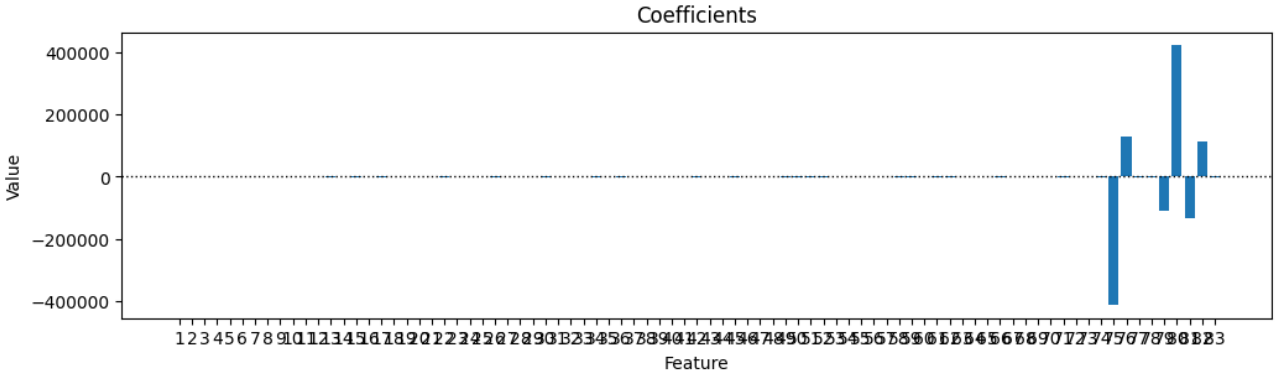
```
ITERATION 17 RESULTS

Pipeline: Transformers: ord_ohe2_poly3 -> Scaler: stand -> Model: linreg
Note: X7 no Duplicate or Outlier removal via Cluster. 10 < Price < 100K, 10 < Odometer < 400k, Year >
1960. Ordinal. OHE. Poly3. Standard. Linear Regression.
Aug 28, 2023 11:58 PM PST

Predictions:
                         Train            Test
MSE:            69,076,642.1940 69,557,184.6447
RMSE:                8,311.2359      8,340.0950
MAE:                 5,600.1994      5,603.1456
R^2 Score:               0.6616          0.6605
```








```
Coefficients:
                       Feature          Value
1                    condition       577.1512
2                    cylinders     1,772.2733
3                         size       142.0640
4   manufacturer_Not specified        31.3582
5           manufacturer_acura       173.6566
..                         ...            ...
79                  odometer^2 -110,049.0855
80                      year^3  421,963.7510
81            year^2 odometer -132,689.8008
82            year odometer^2  114,067.0097
83                  odometer^3   -1,013.0527

[83 rows x 2 columns]
```



Observation: This is the X7 dataset, which does NOT remove duplicates, and does NOT remove outliers via Clustering. I also applied the same stricter thresholds: Price > \$10, Odometer > 10, and upper thresholds: Price < \$100k, Odometer < 400k, and added Year > 1960. The Test R^2 is **0.66**, unchanged with Polynomial degree 3. It seems degree 2 is best for this dataet.

Model Iteration 18

X8 no Duplicate or Outlier Removal via Cluster. \$1000 < Price < \$100K, 1000 < Odometer < 400k. Year > 1980. Ord, OHE, LinReg

In [426…

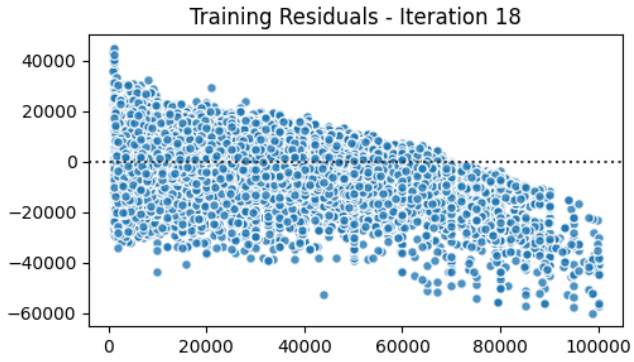
```
i18_model = my.iterate_model(X8_train, X8_test, y8_train, y8_test,
                transformers=['ord','ohe2'], model='linreg',
                iteration='18', note='X8 no Duplicate or Outlier removal via Cluster. 1000 < Price < 1
                grid=False, grid_params='linreg', grid_cv='kfold_10', grid_verbose=3,
                save=True, export=False, config=my_config, debug=False, decimal=4,
                plot=True, perm=False, vif=False, coef=True)
```

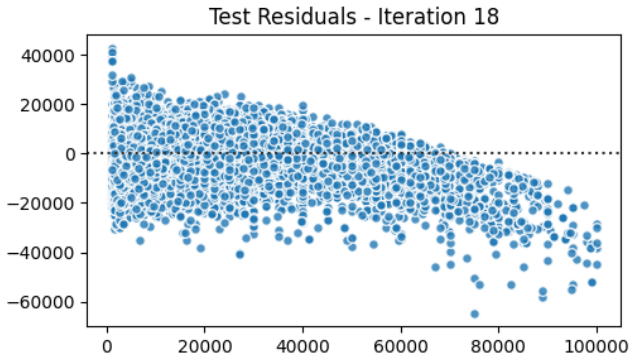
ITERATION 18 RESULTS

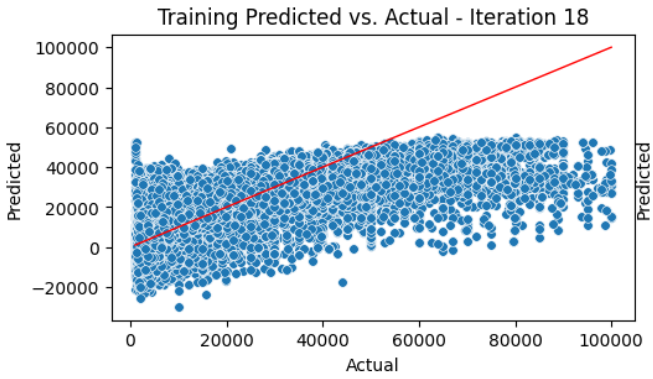
Pipeline: Transformers: ord_ohe2 -> Model: linreg
Note: X8 no Duplicate or Outlier removal via Cluster. 1000 < Price < 100K, 1000 < Odometer < 400k, Yea
r > 1980. Ordinal. OHE. Linear Regression.
Aug 29, 2023 12:05 AM PST

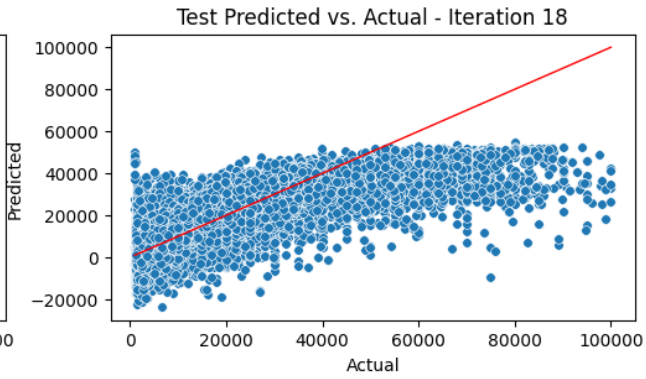
Predictions:

	Train	Test
MSE:	69,877,946.9565	70,505,629.1719
RMSE:	8,359.3030	8,396.7630
MAE:	5,990.9787	5,989.0265
R^2 Score:	0.6467	0.6479





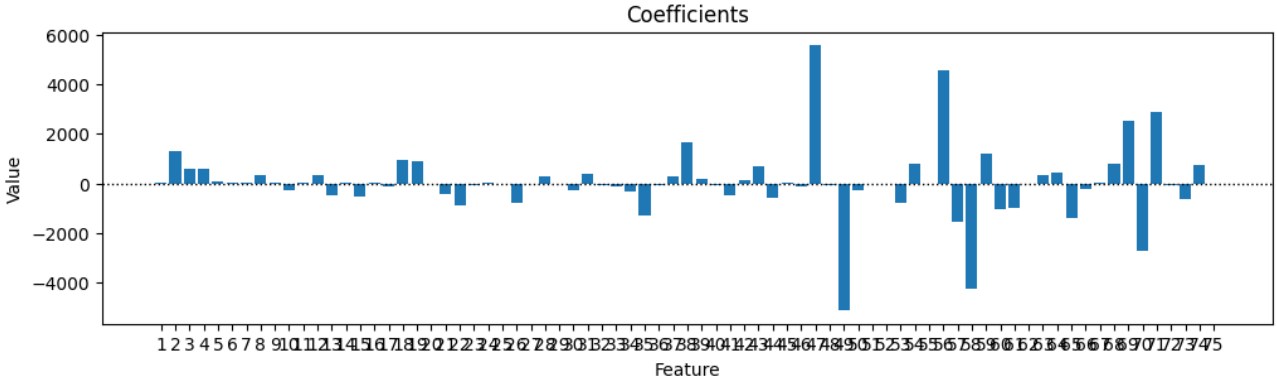




Coefficients:

	Feature	Value
1	condition	35.5801
2	cylinders	1,294.9884
3	size	591.2327
4	manufacturer_Not specified	573.0721
5	manufacturer_acura	76.7720
..	...	...
71	type_truck	2,886.9552
72	type_van	-64.9712
73	type_wagon	-631.9403
74	0	766.9604
75	5	-0.0818

[75 rows x 2 columns]



Observation: This is the X8 dataset, which does NOT remove duplicates, and does NOT remove outliers via Clustering. I applied even stricter thresholds: Price > \$1000, Odometer > 1000, and upper thresholds: Price < \$100k, Odometer < 400k, and added Year > 1980. The Test R^2 is **0.65** without any polynomials.

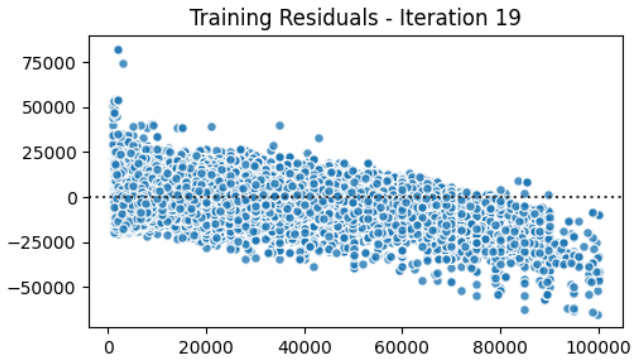
Model Iteration 19

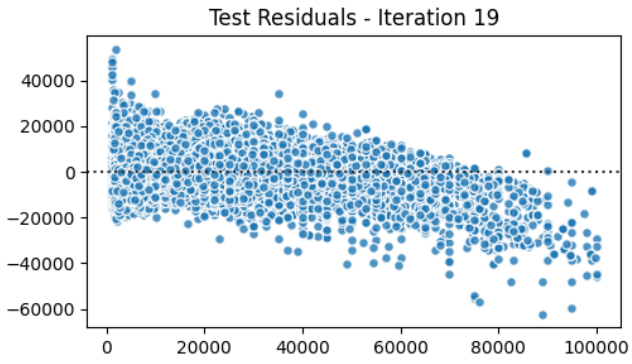
X8 no Duplicate or Outlier Removal via Cluster. \$1000 < Price < \$100K, 1000 < Odometer < 400k. Year > 1980. Ord, OHE, Poly 2, Standard, LinReg

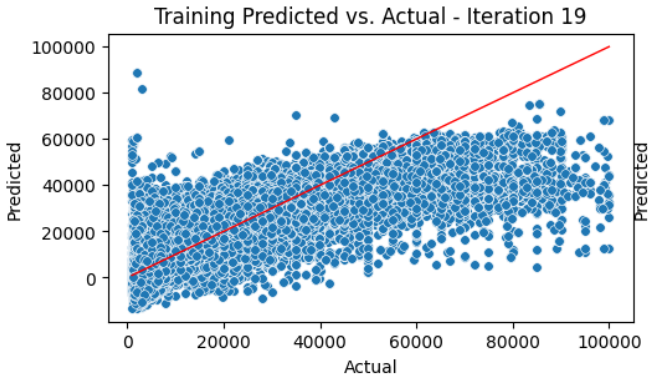
In [428…

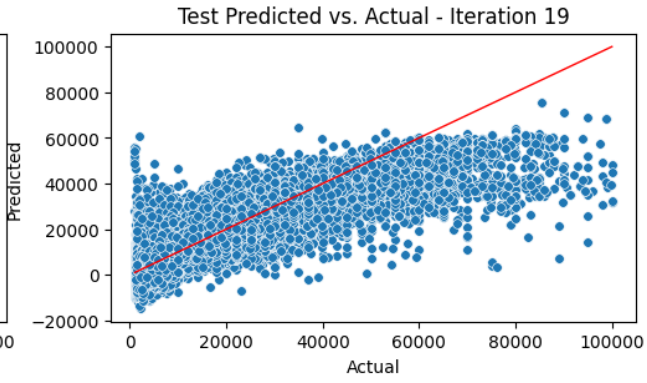
```
i19_model = my.iterate_model(X8_train, X8_test, y8_train, y8_test,
                  transformers=['ord','ohe2','poly2'], scaler='stand', model='linreg',
                  iteration='19', note='X8 no Duplicate or Outlier removal via Cluster. 1000 < Price < 1
                  grid=False, grid_params='linreg', grid_cv='kfold_10', grid_verbose=3,
                  save=True, export=False, config=my_config, debug=False, decimal=4,
                  plot=True, perm=False, vif=False, coef=True)
```

```
ITERATION 19 RESULTS

Pipeline: Transformers: ord_ohe2_poly2 -> Scaler: stand -> Model: linreg
Note: X8 no Duplicate or Outlier removal via Cluster. 1000 < Price < 100K, 1000 < Odometer < 400k, Yea
r > 1980. Ordinal. OHE. Poly2. Standard. Linear Regression.
Aug 29, 2023 12:09 AM PST

Predictions:
                          Train            Test
MSE:             54,646,989.8230 55,007,322.6450
RMSE:                7,392.3602      7,416.6922
MAE:                 5,186.3437      5,187.6802
R^2 Score:               0.7237          0.7253
```

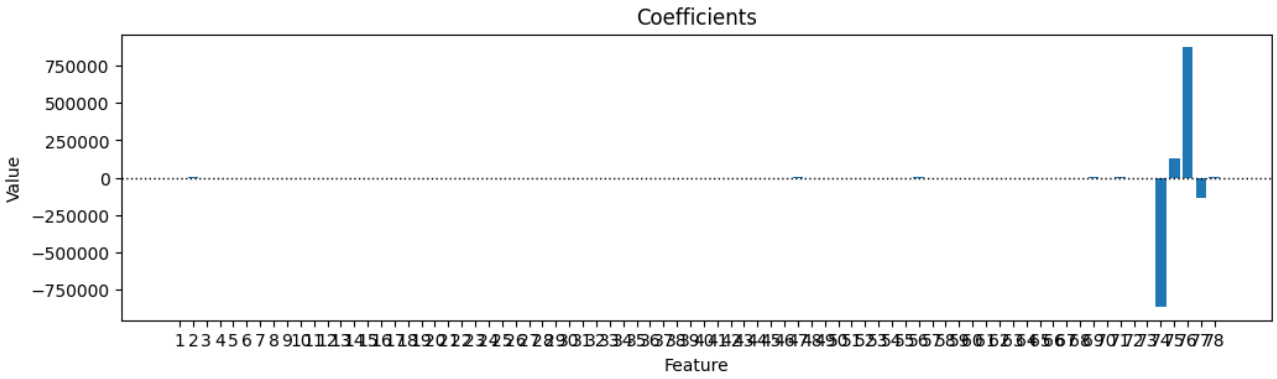








```
Coefficients:
                       Feature         Value
1                    condition      645.4819
2                    cylinders    1,607.6680
3                         size      349.0766
4   manufacturer_Not specified      178.1782
5            manufacturer_acura      193.6498
..                          ...           ...
74                        year -867,600.6455
75                    odometer  129,586.0422
76                      year^2  874,118.8631
77               year odometer -136,007.7252
78                  odometer^2    2,847.4162

[78 rows x 2 columns]
```



Observation: This is the X8 dataset, which does NOT remove duplicates, and does NOT remove outliers via Clustering. I applied even stricter thresholds: Price > \$1000, Odometer > 1000, and upper thresholds: Price < \$100k, Odometer < 400k, and added Year > 1980. The Test R² is now **0.73** with Polynomial degree 2. This is our best score yet. The major coefficients are just Year and Odometer due to the polynomial transformation.

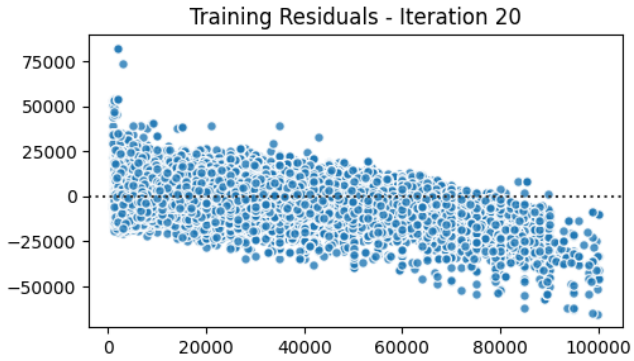
Model Iteration 20

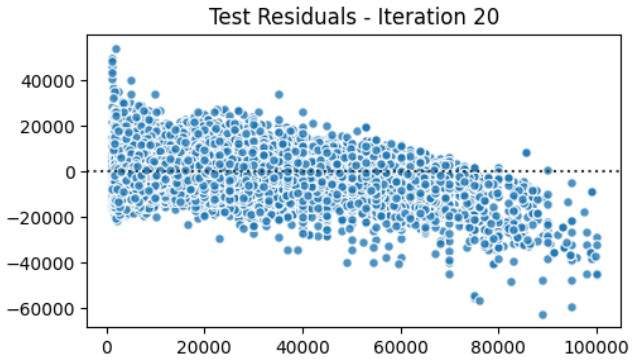
X8 no Duplicate or Outlier Removal via Cluster. \$1000 < Price < \$100K, 1000 < Odometer < 400k. Year > 1980. Ord, OHE, Poly 3, Standard, LinReg

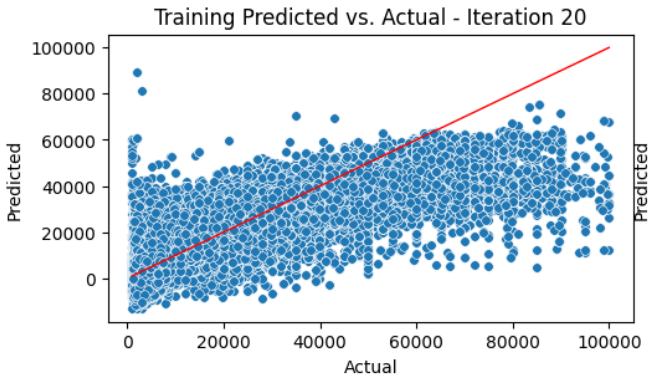
In [429…

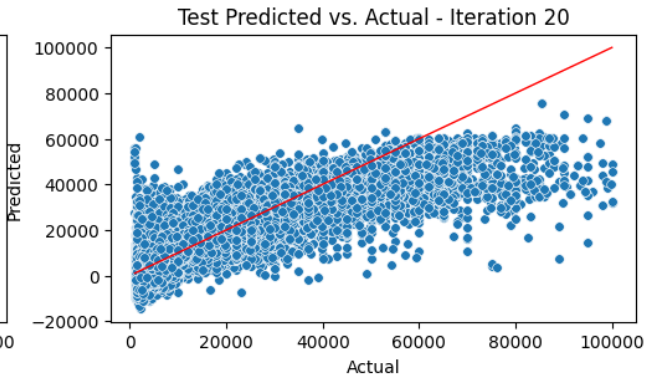
```
i20_model = my.iterate_model(X8_train, X8_test, y8_train, y8_test,
                transformers=['ord','ohe2','poly3'], scaler='stand', model='linreg',
                iteration='20', note='X8 no Duplicate or Outlier removal via Cluster. 1000 < Price < 1
                grid=False, grid_params='linreg', grid_cv='kfold_10', grid_verbose=3,
                save=True, export=False, config=my_config, debug=False, decimal=4,
                plot=True, perm=False, vif=False, coef=True)
```

```
ITERATION 20 RESULTS

Pipeline: Transformers: ord_ohe2_poly3 -> Scaler: stand -> Model: linreg
Note: X8 no Duplicate or Outlier removal via Cluster. 1000 < Price < 100K, 1000 < Odometer < 400k, Yea
r > 1980. Ordinal. OHE. Poly3. Standard. Linear Regression.
Aug 29, 2023 12:25 AM PST

Predictions:
                         Train            Test
MSE:           54,571,993.6583 54,901,155.8495
RMSE:               7,387.2859      7,409.5314
MAE:                5,183.2308      5,183.9748
R^2 Score:              0.7241          0.7258
```

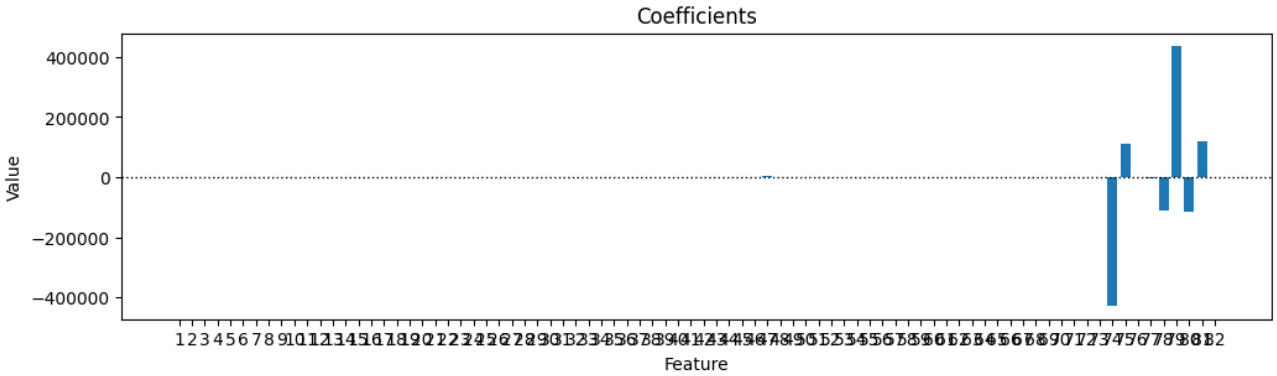









```
Coefficients:
                        Feature         Value
1                     condition      654.9748
2                     cylinders    1,594.2065
3                          size      354.4081
4   manufacturer_Not specified      180.6004
5            manufacturer_acura      189.7418
..                          ...           ...
78                  odometer^2 -112,590.2851
79                      year^3  435,144.0900
80             year^2 odometer -113,406.7238
81             year odometer^2  117,624.7313
82                  odometer^3   -1,373.9170

[82 rows x 2 columns]
```



Observation: This is the X8 dataset, which does NOT remove duplicates, and does NOT remove outliers via Clustering. I applied even stricter thresholds: Price > \$1000, Odometer > 1000, and upper thresholds: Price < \$100k, Odometer < 400k, and added Year > 1980. The Test R^2 is **0.73** still with Polynomial degree 3. As expected, it did not improve. So far model 19 is our top performer.

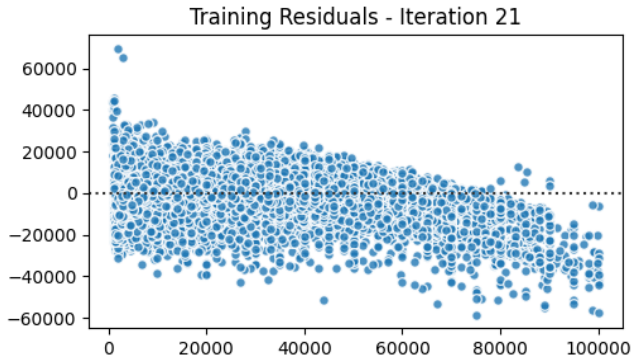
Model Iteration 21

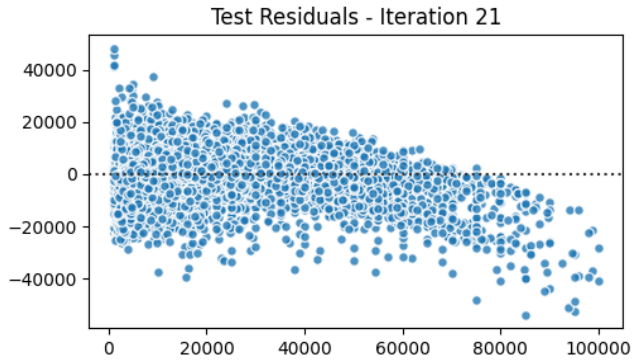
X9 Top Correlated Features. \$1000 < Price < \$100K, 1000 < Odometer < 400k. Year > 1980. LinReg

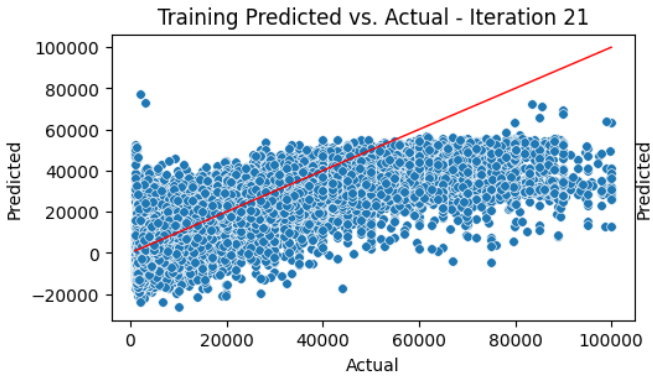
In [455…

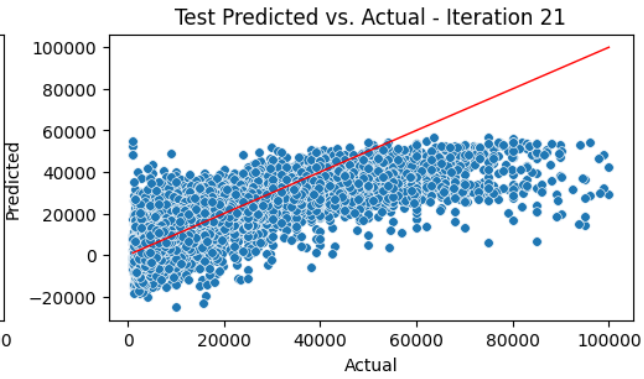
```
i21_model = my.iterate_model(X9_train, X9_test, y9_train, y9_test,
                  model='linreg',
                  iteration='21', note='X9 top correlated features. 1000 < Price < 100K, 1000 < Odometer
                  grid=False, grid_params='linreg', grid_cv='kfold_10', grid_verbose=3,
                  save=True, export=False, config=my_config, debug=False, decimal=4,
                  plot=True, perm=False, vif=False, coef=True)
```

```
ITERATION 21 RESULTS

Pipeline: Model: linreg
Note: X9 top correlated features. 1000 < Price < 100K, 1000 < Odometer < 400k, Year > 1980. Linear Reg
ression.
Aug 29, 2023 12:57 AM PST

Predictions:
                         Train            Test
MSE:            62,146,559.3581 60,720,230.6302
RMSE:                 7,883.3089      7,792.3187
MAE:                  5,454.0315      5,400.0490
R^2 Score:                0.6408          0.6457
```



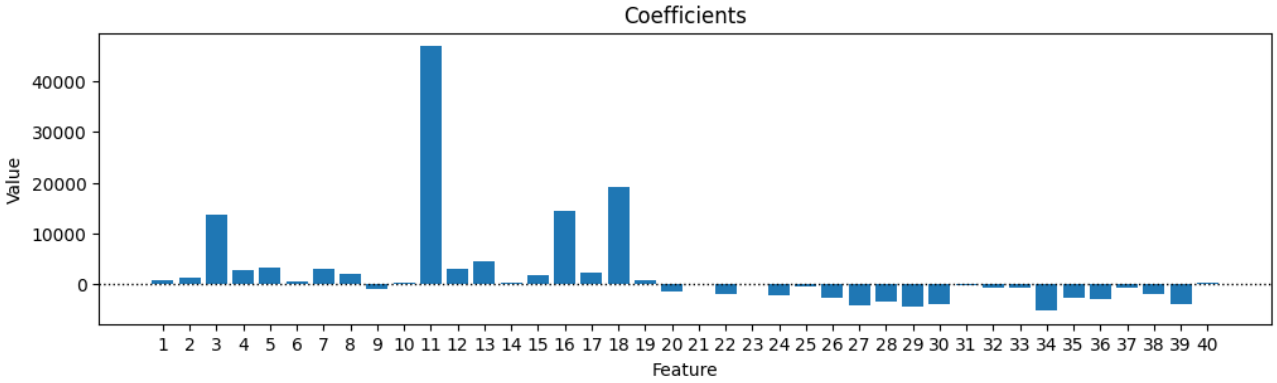






Coefficients:

	Feature	Value
1	year	719.2471
2	cylinders	1,275.8876
3	fuel_diesel	13,671.1383
4	drive_4wd	2,729.5662
5	type_pickup	3,226.3243
6	condition	569.2758
7	type_truck	3,135.7480
8	manu_ram	1,982.8152
9	trans_other	-913.8467
10	size	206.6791
11	manu_ferrari	46,865.8881
12	model_pca_1	3,061.4196
13	model_pca_0	4,420.6629
14	paint_white	414.8044
15	fuel_other	1,676.4380
16	manu_porsche	14,385.8845
17	manu_gmc	2,353.4509
18	manu_tesla	19,037.1101
19	type_other	873.0325
20	manu_ford	-1,319.7998
21	odometer	-0.0809
22	drive_fwd	-2,036.7194
23	fuel_gas	71.7090
24	type_sedan	-2,072.0387
25	manu_honda	-508.1113
26	trans_automatic	-2,571.1788
27	manu_nissan	-4,123.4490
28	manu_chrysler	-3,355.5575
29	manu_hyundai	-4,348.7311
30	type_hatchback	-3,803.2733
31	type_mini-van	-284.5187
32	paint_silver	-661.0197
33	paint_Not specified	-716.4239
34	manu_kia	-5,216.5615
35	manu_mazda	-2,798.4082
36	manu_saturn	-2,905.5096
37	type_Not specified	-801.6227
38	state_oh	-2,010.5480
39	manu_volkswagen	-3,988.7259
40	paint_green	408.0068



Observation: This is the X9 dataset, which is a result of manual encoding (Ordinal and OHE) and then selecting the top 40 correlated features from the matrix. In addition, I applied the stricter thresholds: Price > \$1000, Odometer > 1000, and upper thresholds: Price < \$100k, Odometer < 400k, and added Year > 1980. The Test R^2 is **0.65** without any polynomials. This is pretty good, and what's even better is that this does have the duplicates removed, so it is very generalizable to unseen data. Also, take a look at the coefficients. There are 40, and most of them seem to have a purpose – a meaningful value. The biggest factors are: Ferraris, Teslas, Porsches, and Diesel fuel.

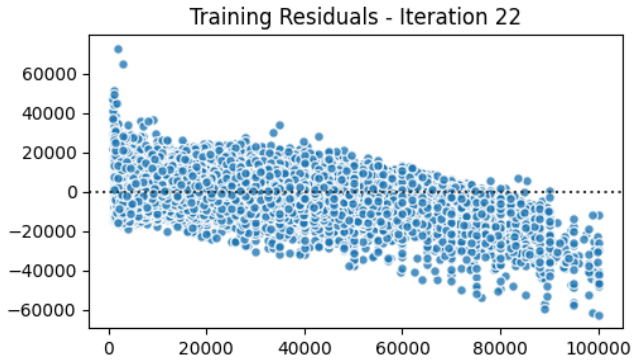
Model Iteration 22

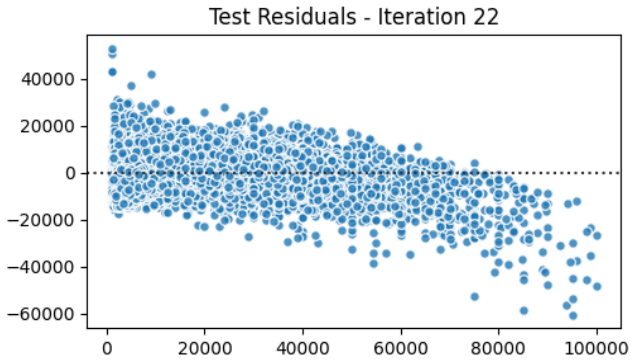
X9 Top Correlated Features. \$1000 < Price < \$100K, 1000 < Odometer < 400k. Year > 1980. Poly 2, LinReg

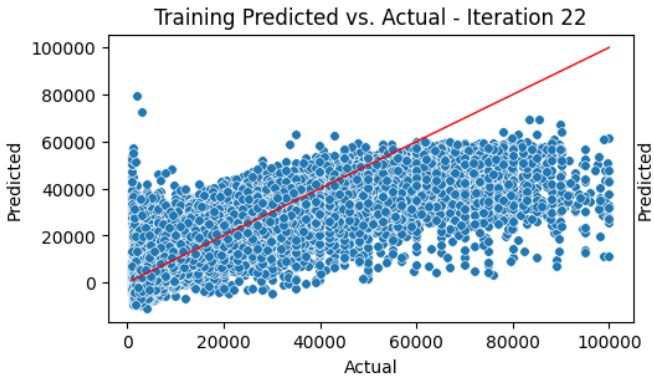
In [509…

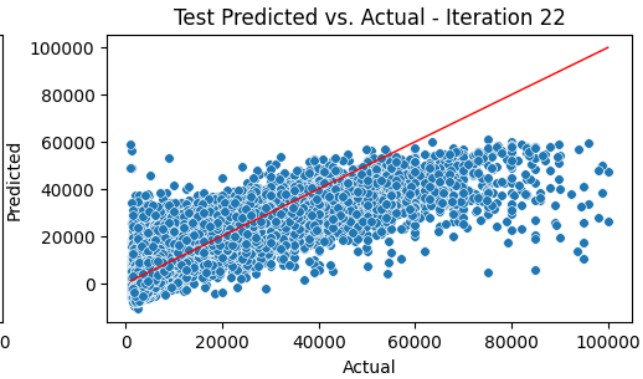
```
i22_model = my.iterate_model(X9_train, X9_test, y9_train, y9_test,
                  transformers=['poly2'], model='linreg',
                  iteration='22', note='X9 top correlated features. 1000 < Price < 100K, 1000 < Odometer
                  grid=False, grid_params='linreg', grid_cv='kfold_10', grid_verbose=3,
                  save=True, export=False, config=my_config, debug=False, decimal=4,
                  plot=True, perm=False, vif=False, coef=True)
```

```
ITERATION 22 RESULTS

Pipeline: Transformers: poly2 -> Model: linreg
Note: X9 top correlated features. 1000 < Price < 100K, 1000 < Odometer < 400k, Year > 1980. Poly 2, Li
near Regression.
Aug 29, 2023 01:35 AM PST

Predictions:
                         Train            Test
MSE:            54,443,553.5873 53,070,200.6242
RMSE:                7,378.5875      7,284.9297
MAE:                 5,024.6713      4,979.2792
R^2 Score:               0.6853          0.6903
```



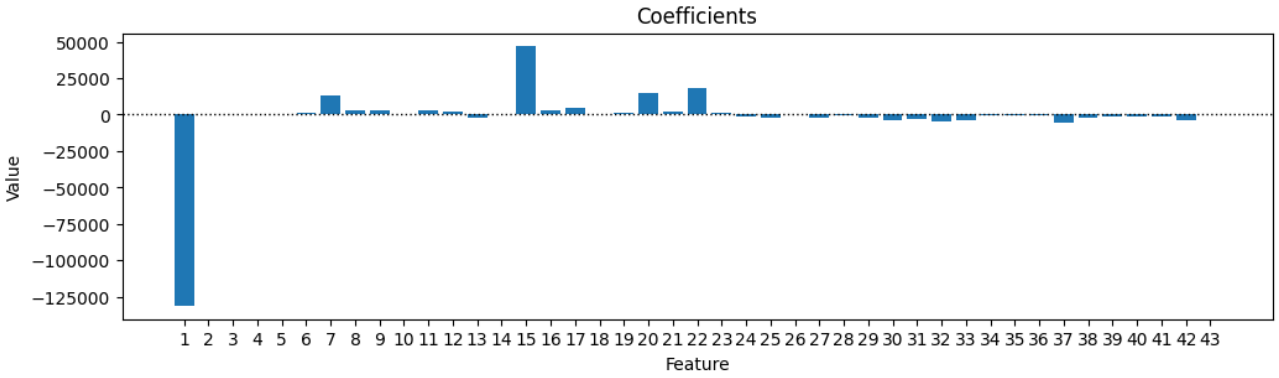






Coefficients:

	Feature	Value
1	year	-131,285.2684
2	odometer	2.8258
3	year^2	32.9367
4	year odometer	0.0015
5	odometer^2	0.0000
6	1	1,272.8072
7	2	12,926.5822
8	3	2,722.7214
9	4	2,835.2737
10	5	619.3512
11	6	2,700.9258
12	7	1,875.5999
13	8	-1,979.1969
14	9	193.2860
15	10	46,849.8832
16	11	3,028.5346
17	12	4,427.8985
18	13	193.3351
19	14	1,533.6827
20	15	14,829.1517
21	16	2,174.7734
22	17	17,851.3099
23	18	877.5132
24	19	-1,313.5758
25	21	-1,928.4055
26	22	254.4776
27	23	-1,790.9990
28	24	-416.1988
29	25	-1,837.4133
30	26	-4,174.5847
31	27	-2,644.8248
32	28	-4,316.9448
33	29	-3,581.2126
34	30	-83.7624
35	31	-378.4568
36	32	-661.1794
37	33	-5,395.4609
38	34	-2,438.0035
39	35	-1,092.4920
40	36	-963.8123
41	37	-1,733.1569
42	38	-3,931.9528
43	39	57.0646



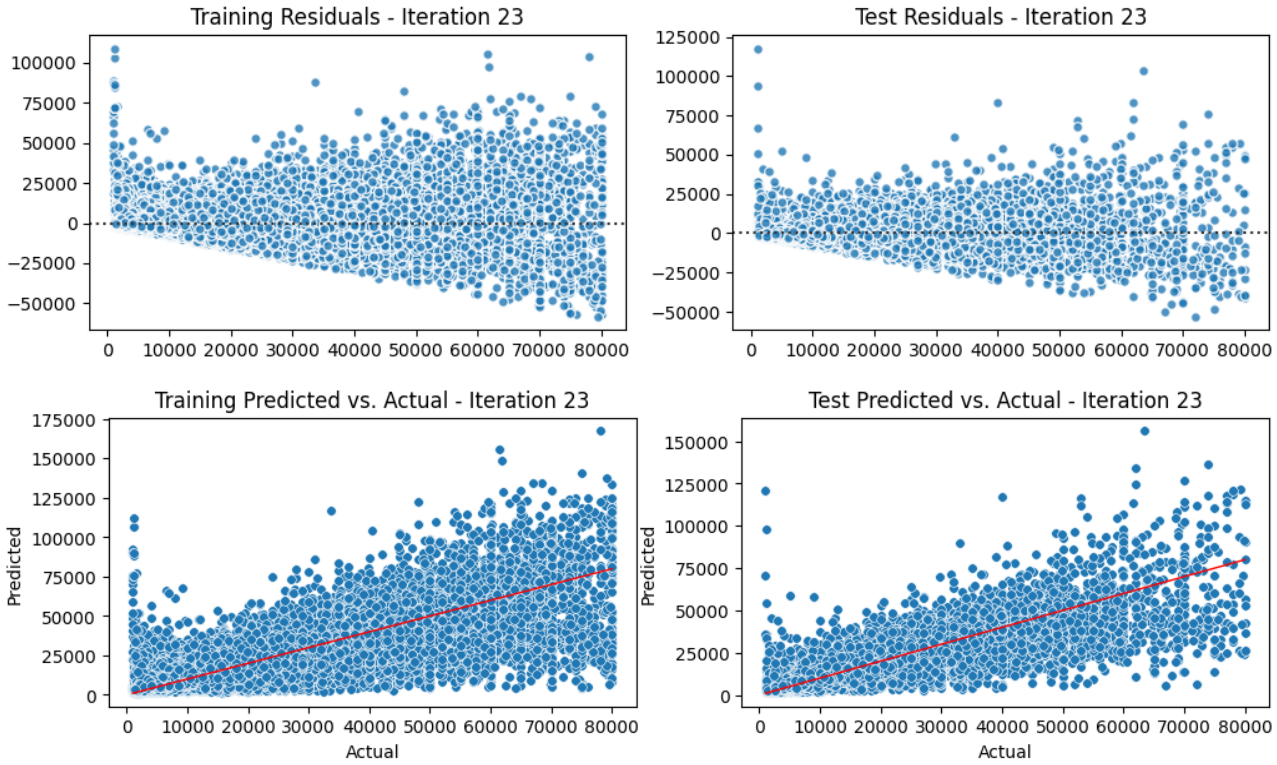
Observation: This is the X9 dataset, which is a result of manual encoding (Ordinal and OHE) and then selecting the top 40 correlated features from the matrix. In addition, I applied the stricter thresholds: Price > \$1000, Odometer > 1000, and upper thresholds: Price < \$100k, Odometer < 400k, and added Year > 1980. The Test R^2 is **0.69** now with Polynomial degree 2. This is a nice improvement.

Model Iteration 23

X10 Top Correlated Features. \$1000 < Price < \$80K, 1000 < Odometer < 400k. Year > 1980. Log, Poly 2, LinReg

In [528…
```
i23_model = my.iterate_model(X10_train, X10_test, y10_train, y10_test,
                transformers=['poly2'], model='ttr_log',
                iteration='23', note='X10 top correlated features. 1000 < Price < 80K, 1000 < Odometer
                grid=False, grid_params='linreg', grid_cv='kfold_10', grid_verbose=3,
                save=True, export=False, config=my_config, debug=False, decimal=4,
                plot=True, perm=False, vif=False, coef=True)
```

```
ITERATION 23 RESULTS

Pipeline: Transformers: poly2 -> Model: ttr_log
Note: X10 top correlated features. 1000 < Price < 80K, 1000 < Odometer < 400k, Year > 1980. Log. Poly
2, Linear Regression.
Aug 29, 2023 01:47 AM PST

Predictions:
                          Train            Test
MSE:            51,905,704.0984 50,450,993.5430
RMSE:                7,204.5613      7,102.8863
MAE:                 4,495.2763      4,442.4388
R^2 Score:               0.6832          0.6849
```



Observation: This is the X10 dataset, which is a result of manual encoding (Ordinal and OHE) and then selecting the top 40 correlated features from the matrix. In addition, I applied the stricter thresholds: Price > \$1000, Odometer > 1000, and upper thresholds: Price < \$80k, Odometer < 400k, and added Year > 1980. The Test R^2 is **0.68** with Log transformation and Polynomial degree 2. The MAE is also lower.

Model Iteration 24

X10 Top Correlated Features. \$1000 < Price < \$80K, 1000 < Odometer < 400k. Year > 1980. Poly 2, Lasso

GridSearchCV: Lasso, alpha 0.001 - 100

In [542…
```
i24_model, i24_results = my.iterate_model(X10_train, X10_test, y10_train, y10_test,
                transformers=['poly2'], model='lasso_tol',
                iteration='24', note='X10 top correlated features. 1000 < Price < 80K, 1000 < Odometer
                grid=True, grid_params='lasso_tol_1', grid_cv='kfold_5', grid_verbose=3,
                save=True, export=False, config=my_config, debug=False, decimal=4,
                plot=True, perm=False, vif=False, coef=True)
```

ITERATION 24 RESULTS

Pipeline: Transformers: poly2 -> Model: lasso_tol
Note: X10 top correlated features. 1000 < Price < 80K, 1000 < Odometer < 400k, Year > 1980. Log. Poly 2, Lasso, GridSearchCV alpha 0.001 - 100.
Aug 29, 2023 02:15 AM PST

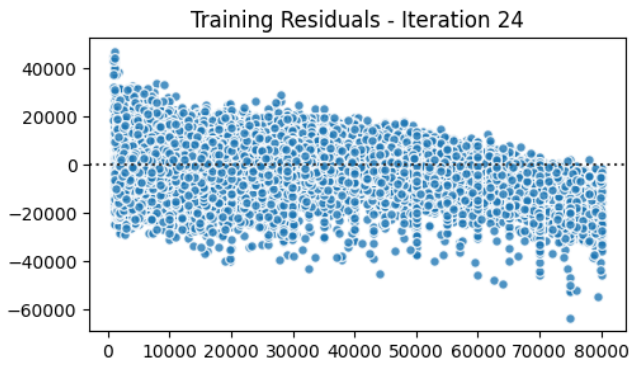
Cross Validation:

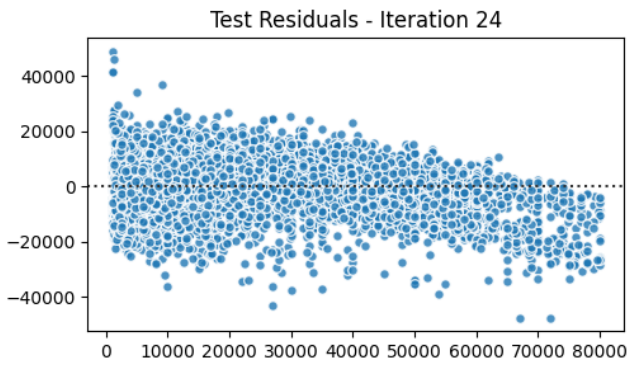
```
Fitting 5 folds for each of 6 candidates, totalling 30 fits
[CV 1/5] END .....Model: lasso_tol__alpha=0.001;, score=0.653 total time=   0.2s
[CV 2/5] END .....Model: lasso_tol__alpha=0.001;, score=0.648 total time=   0.2s
[CV 3/5] END .....Model: lasso_tol__alpha=0.001;, score=0.653 total time=   0.2s
[CV 4/5] END .....Model: lasso_tol__alpha=0.001;, score=0.659 total time=   0.1s
[CV 5/5] END .....Model: lasso_tol__alpha=0.001;, score=0.646 total time=   0.2s
[CV 1/5] END ......Model: lasso_tol__alpha=0.01;, score=0.653 total time=   0.2s
[CV 2/5] END ......Model: lasso_tol__alpha=0.01;, score=0.648 total time=   0.2s
[CV 3/5] END ......Model: lasso_tol__alpha=0.01;, score=0.653 total time=   0.2s
[CV 4/5] END ......Model: lasso_tol__alpha=0.01;, score=0.659 total time=   0.2s
[CV 5/5] END ......Model: lasso_tol__alpha=0.01;, score=0.646 total time=   0.2s
[CV 1/5] END .......Model: lasso_tol__alpha=0.1;, score=0.653 total time=   0.2s
[CV 2/5] END .......Model: lasso_tol__alpha=0.1;, score=0.648 total time=   0.1s
[CV 3/5] END .......Model: lasso_tol__alpha=0.1;, score=0.653 total time=   0.1s
[CV 4/5] END .......Model: lasso_tol__alpha=0.1;, score=0.659 total time=   0.1s
[CV 5/5] END .......Model: lasso_tol__alpha=0.1;, score=0.646 total time=   0.2s
[CV 1/5] END .......Model: lasso_tol__alpha=1.0;, score=0.654 total time=   0.2s
[CV 2/5] END .......Model: lasso_tol__alpha=1.0;, score=0.648 total time=   0.1s
[CV 3/5] END .......Model: lasso_tol__alpha=1.0;, score=0.653 total time=   0.1s
[CV 4/5] END .......Model: lasso_tol__alpha=1.0;, score=0.659 total time=   0.1s
[CV 5/5] END .......Model: lasso_tol__alpha=1.0;, score=0.646 total time=   0.1s
[CV 1/5] END ......Model: lasso_tol__alpha=10.0;, score=0.653 total time=   0.1s
[CV 2/5] END ......Model: lasso_tol__alpha=10.0;, score=0.647 total time=   0.2s
[CV 3/5] END ......Model: lasso_tol__alpha=10.0;, score=0.652 total time=   0.1s
[CV 4/5] END ......Model: lasso_tol__alpha=10.0;, score=0.658 total time=   0.1s
[CV 5/5] END ......Model: lasso_tol__alpha=10.0;, score=0.645 total time=   0.1s
[CV 1/5] END .....Model: lasso_tol__alpha=100.0;, score=0.642 total time=   0.2s
[CV 2/5] END .....Model: lasso_tol__alpha=100.0;, score=0.636 total time=   0.1s
[CV 3/5] END .....Model: lasso_tol__alpha=100.0;, score=0.642 total time=   0.2s
[CV 4/5] END .....Model: lasso_tol__alpha=100.0;, score=0.645 total time=   0.1s
[CV 5/5] END .....Model: lasso_tol__alpha=100.0;, score=0.635 total time=   0.2s
```

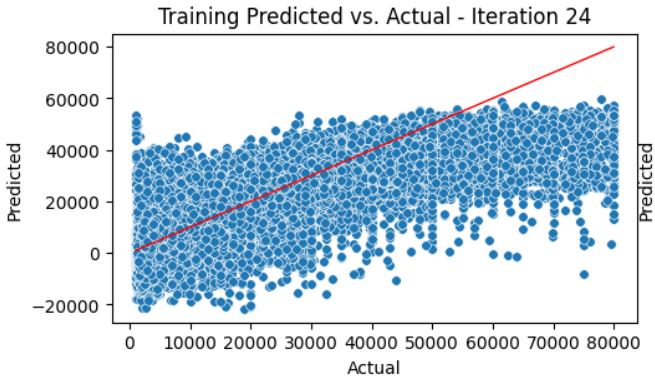
Best Grid mean score (r2): 0.6520
Best Grid parameters: Model: lasso_tol__alpha: 1.0

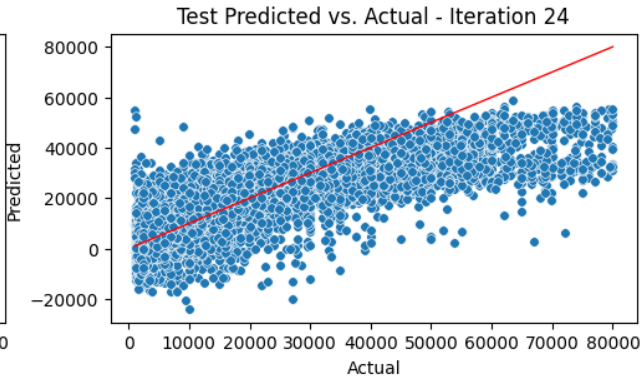
Predictions:

	Train	Test
MSE:	56,962,963.3165	55,350,822.9726
RMSE:	7,547.3812	7,439.8134
MAE:	5,352.9442	5,307.2641
R^2 Score:	0.6523	0.6543



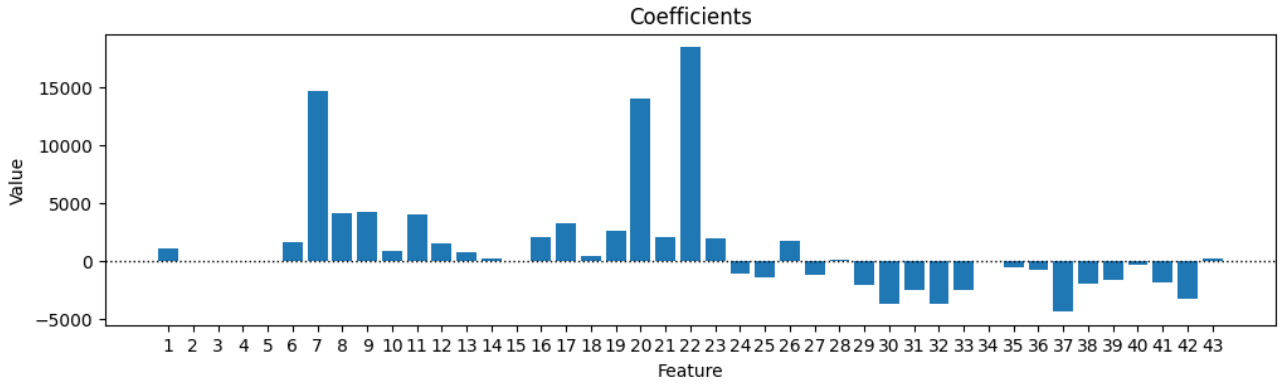






Coefficients:

	Feature	Value
1	year	1,120.2582
2	odometer	-0.0685
3	year^2	-0.0903
4	year odometer	0.0000
5	odometer^2	0.0000
6	1	1,685.1673
7	2	14,684.9271
8	3	4,133.9870
9	4	4,240.9061
10	5	872.0836
11	6	3,981.9539
12	7	1,559.9384
13	8	752.7752
14	9	232.5051
15	10	0.0000
16	11	2,072.6927
17	12	3,310.1862
18	13	422.7869
19	14	2,632.8929
20	15	14,001.9629
21	16	2,112.9974
22	17	18,460.4808
23	18	1,954.4440
24	19	-1,050.1004
25	21	-1,449.1387
26	22	1,698.3906
27	23	-1,167.9203
28	24	73.8987
29	25	-2,067.7071
30	26	-3,703.8454
31	27	-2,502.7527
32	28	-3,670.9088
33	29	-2,497.2597
34	30	9.4881
35	31	-544.2667
36	32	-788.8185
37	33	-4,368.3949
38	34	-1,982.7079
39	35	-1,619.3175
40	36	-290.0153
41	37	-1,834.8961
42	38	-3,298.6024
43	39	169.5702



Observation: This is the X10 dataset, which is a result of manual encoding (Ordinal and OHE) and then selecting the top 40 correlated features from the matrix. In addition, I applied the stricter thresholds: Price > \$1000, Odometer > 1000, and upper thresholds: Price < \$80k, Odometer < 400k, and added Year > 1980. We applied GridSearchCV with Lasso exploring values of alpha between 0.001 and 100. The best alpha value was 1.0, and this produced a Test R^2 is **0.65** with Polynomial degree 2.

Model Iteration 25

X10 Top Correlated Features. \$1000 < Price < \$80K, 1000 < Odometer < 400k. Year > 1980. Poly 2, Ridge

GridSearchCV: Ridge, alpha 0.01 - 100,000

In [543…

```
i25_model, i25_results = my.iterate_model(X10_train, X10_test, y10_train, y10_test,
                 transformers=['poly2'], scaler='stand', model='ridge',
                 iteration='25', note='X10 top correlated features. 1000 < Price < 80K, 1000 < Odometer
                 grid=True, grid_params='ridge_100000', grid_cv='kfold_5', grid_verbose=3,
                 save=True, export=False, config=my_config, debug=False, decimal=4,
                 plot=True, perm=False, vif=False, coef=True)
```

ITERATION 25 RESULTS

Pipeline: Transformers: poly2 -> Scaler: stand -> Model: ridge
Note: X10 top correlated features. 1000 < Price < 80K, 1000 < Odometer < 400k, Year > 1980. Poly 2, Ridge, GridSearchCV alpha 0.001 - 100,000.
Aug 29, 2023 02:23 AM PST

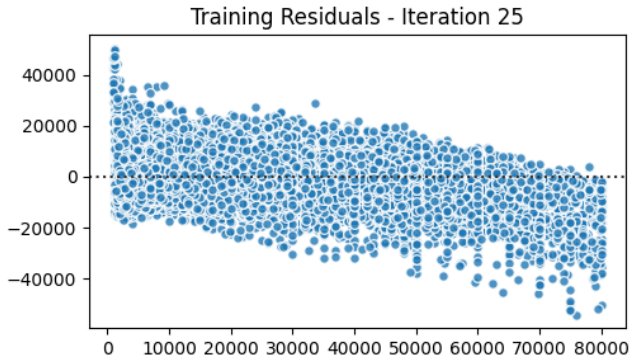
Cross Validation:

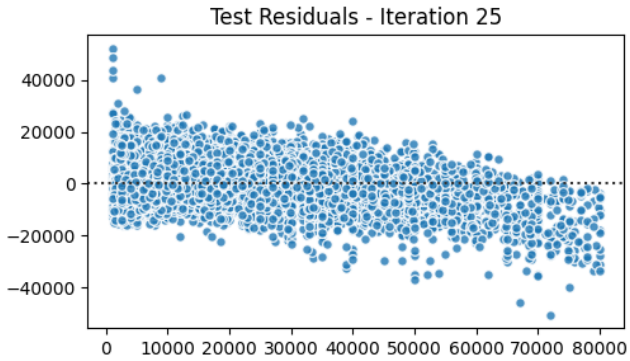
Fitting 5 folds for each of 8 candidates, totalling 40 fits
[CV 1/5] END .....Model: ridge__alpha=0.01;, score=0.697 total time=   0.2s
[CV 2/5] END .....Model: ridge__alpha=0.01;, score=0.691 total time=   0.2s
[CV 3/5] END .....Model: ridge__alpha=0.01;, score=0.694 total time=   0.2s
[CV 4/5] END .....Model: ridge__alpha=0.01;, score=0.703 total time=   0.2s
[CV 5/5] END .....Model: ridge__alpha=0.01;, score=0.690 total time=   0.2s
[CV 1/5] END .....Model: ridge__alpha=0.1;, score=0.694 total time=   0.2s
[CV 2/5] END .....Model: ridge__alpha=0.1;, score=0.690 total time=   0.3s
[CV 3/5] END .....Model: ridge__alpha=0.1;, score=0.693 total time=   0.2s
[CV 4/5] END .....Model: ridge__alpha=0.1;, score=0.700 total time=   0.2s
[CV 5/5] END .....Model: ridge__alpha=0.1;, score=0.689 total time=   0.2s
[CV 1/5] END .....Model: ridge__alpha=1.0;, score=0.682 total time=   0.2s
[CV 2/5] END .....Model: ridge__alpha=1.0;, score=0.679 total time=   0.2s
[CV 3/5] END .....Model: ridge__alpha=1.0;, score=0.683 total time=   0.2s
[CV 4/5] END .....Model: ridge__alpha=1.0;, score=0.687 total time=   0.2s
[CV 5/5] END .....Model: ridge__alpha=1.0;, score=0.678 total time=   0.2s
[CV 1/5] END .....Model: ridge__alpha=10.0;, score=0.674 total time=   0.2s
[CV 2/5] END .....Model: ridge__alpha=10.0;, score=0.671 total time=   0.2s
[CV 3/5] END .....Model: ridge__alpha=10.0;, score=0.675 total time=   0.2s
[CV 4/5] END .....Model: ridge__alpha=10.0;, score=0.679 total time=   0.2s
[CV 5/5] END .....Model: ridge__alpha=10.0;, score=0.670 total time=   0.2s
[CV 1/5] END .....Model: ridge__alpha=100.0;, score=0.672 total time=   0.2s
[CV 2/5] END .....Model: ridge__alpha=100.0;, score=0.669 total time=   0.2s
[CV 3/5] END .....Model: ridge__alpha=100.0;, score=0.674 total time=   0.2s
[CV 4/5] END .....Model: ridge__alpha=100.0;, score=0.678 total time=   0.2s
[CV 5/5] END .....Model: ridge__alpha=100.0;, score=0.669 total time=   0.2s
[CV 1/5] END .....Model: ridge__alpha=1000.0;, score=0.672 total time=   0.2s
[CV 2/5] END .....Model: ridge__alpha=1000.0;, score=0.669 total time=   0.2s
[CV 3/5] END .....Model: ridge__alpha=1000.0;, score=0.673 total time=   0.2s
[CV 4/5] END .....Model: ridge__alpha=1000.0;, score=0.677 total time=   0.2s
[CV 5/5] END .....Model: ridge__alpha=1000.0;, score=0.669 total time=   0.2s
[CV 1/5] END ......Model: ridge__alpha=10000.0;, score=0.661 total time=   0.2s
[CV 2/5] END ......Model: ridge__alpha=10000.0;, score=0.658 total time=   0.2s
[CV 3/5] END ......Model: ridge__alpha=10000.0;, score=0.661 total time=   0.2s
[CV 4/5] END ......Model: ridge__alpha=10000.0;, score=0.667 total time=   0.2s
[CV 5/5] END ......Model: ridge__alpha=10000.0;, score=0.657 total time=   0.2s
[CV 1/5] END .....Model: ridge__alpha=100000.0;, score=0.604 total time=   0.2s
[CV 2/5] END .....Model: ridge__alpha=100000.0;, score=0.602 total time=   0.2s
[CV 3/5] END .....Model: ridge__alpha=100000.0;, score=0.601 total time=   0.2s
[CV 4/5] END .....Model: ridge__alpha=100000.0;, score=0.608 total time=   0.2s
[CV 5/5] END .....Model: ridge__alpha=100000.0;, score=0.599 total time=   0.2s

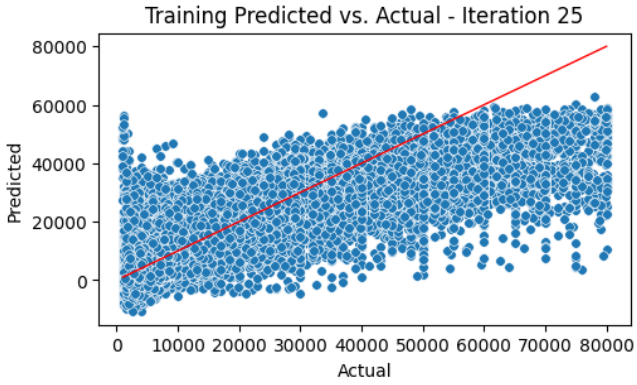
Best Grid mean score (r2): 0.6948
Best Grid parameters: Model: ridge__alpha: 0.01

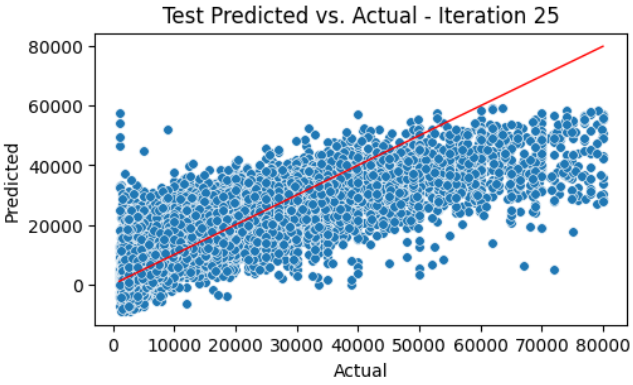
Predictions:

	Train	Test
MSE:	49,912,437.5044	48,041,369.5607
RMSE:	7,064.8735	6,931.1882
MAE:	4,920.7847	4,857.8946
R^2 Score:	0.6954	0.6999



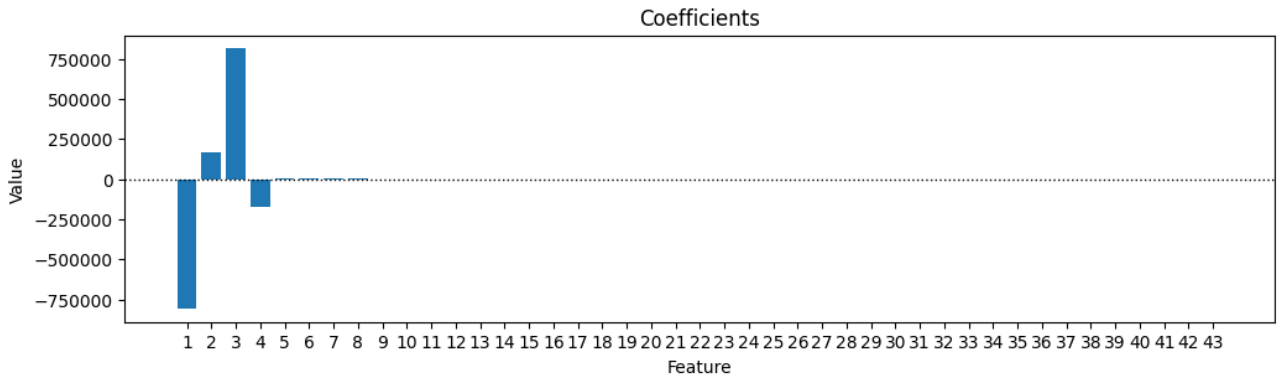






Coefficients:

	Feature	Value
1	year	-807,985.0091
2	odometer	169,517.0783
3	year^2	813,963.7864
4	year odometer	-175,641.5726
5	odometer^2	2,913.1285
6	1	1,766.2070
7	2	2,873.7639
8	3	1,271.1413
9	4	801.9822
10	5	585.5045
11	6	758.1160
12	7	383.2075
13	8	-384.4960
14	9	188.2592
15	10	71.4143
16	11	502.2422
17	12	764.0263
18	13	99.0490
19	14	298.9425
20	15	813.8810
21	16	407.5234
22	17	631.0405
23	18	169.8484
24	19	-487.8585
25	21	-856.7526
26	22	91.4371
27	23	-741.7201
28	24	-92.9894
29	25	-585.9363
30	26	-907.5258
31	27	-326.3291
32	28	-686.7712
33	29	-661.7540
34	30	-14.4674
35	31	-97.5455
36	32	-278.4486
37	33	-748.4805
38	34	-281.5057
39	35	-63.3602
40	36	-395.4815
41	37	-310.0376
42	38	-569.9461
43	39	-8.4498



Observation: This is the X10 dataset, which is a result of manual encoding (Ordinal and OHE) and then selecting the top 40 correlated features from the matrix. In addition, I applied the stricter thresholds: Price > \$1000, Odometer > 1000, and upper thresholds: Price < \$80k, Odometer < 400k, and added Year > 1980. We applied GridSearchCV with Ridge exploring values of alpha between 0.01 and 100,000. The best alpha value was 0.01, and this produced a Test R^2 of **0.70** with Polynomial degree 2.

Model Iteration 26

X10 Top Correlated Features. \$1000 < Price < \$80K, 1000 < Odometer < 400k. Year > 1980. Log, Poly 2, Ridge

GridSearchCV: Ridge, alpha 0.001 - 100,000

In [546…

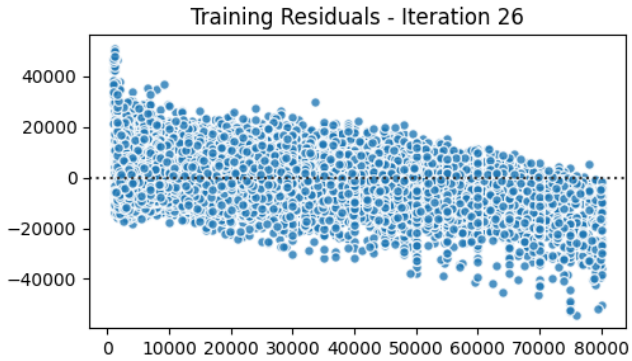
```
i26_model, i26_results = my.iterate_model(X10_train, X10_test, y10_train, y10_test,
                 transformers=['log','poly2'], scaler='stand', model='ridge',
                 iteration='26', note='X10 top correlated features. 1000 < Price < 80K, 1000 < Odometer
                 grid=True, grid_params='ridge_100000', grid_cv='kfold_5', grid_verbose=3,
                 save=True, export=False, config=my_config, debug=False, decimal=4,
                 plot=True, perm=False, vif=False, coef=True)
```

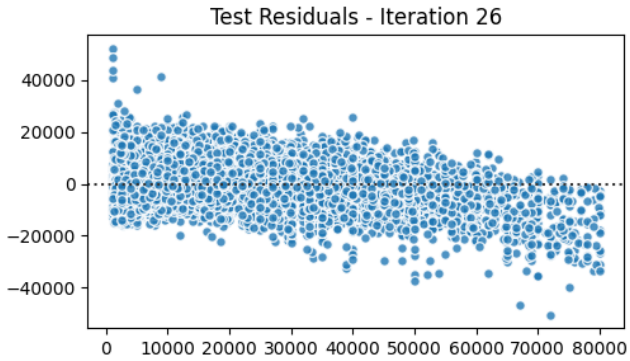
ITERATION 26 RESULTS

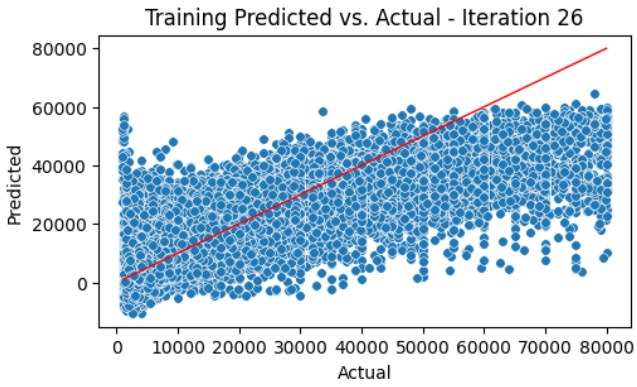
Pipeline: Transformers: log_poly2 -> Scaler: stand -> Model: ridge
Note: X10 top correlated features. 1000 < Price < 80K, 1000 < Odometer < 400k, Year > 1980. Log, Poly 2, Ridge, GridSearchCV alpha 0.001 - 100,000.
Aug 29, 2023 02:31 AM PST

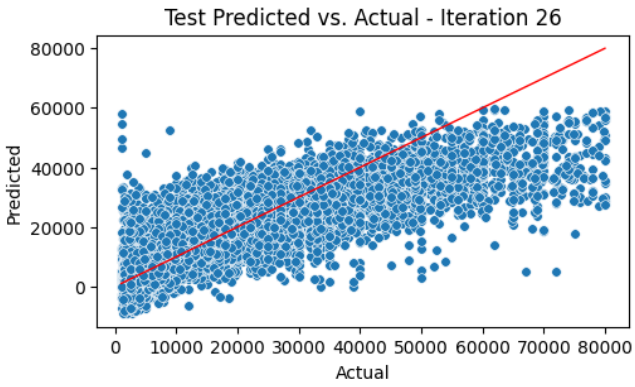
Cross Validation:

```
Fitting 5 folds for each of 9 candidates, totalling 45 fits
[CV 1/5] END .........Model: ridge__alpha=0.001;, score=0.697 total time=   0.2s
[CV 2/5] END .........Model: ridge__alpha=0.001;, score=0.691 total time=   0.2s
[CV 3/5] END .........Model: ridge__alpha=0.001;, score=0.694 total time=   0.2s
[CV 4/5] END .........Model: ridge__alpha=0.001;, score=0.703 total time=   0.3s
[CV 5/5] END .........Model: ridge__alpha=0.001;, score=0.690 total time=   0.2s
[CV 1/5] END ..........Model: ridge__alpha=0.01;, score=0.697 total time=   0.2s
[CV 2/5] END ..........Model: ridge__alpha=0.01;, score=0.691 total time=   0.2s
[CV 3/5] END ..........Model: ridge__alpha=0.01;, score=0.694 total time=   0.2s
[CV 4/5] END ..........Model: ridge__alpha=0.01;, score=0.703 total time=   0.2s
[CV 5/5] END ..........Model: ridge__alpha=0.01;, score=0.690 total time=   0.2s
[CV 1/5] END ...........Model: ridge__alpha=0.1;, score=0.694 total time=   0.2s
[CV 2/5] END ...........Model: ridge__alpha=0.1;, score=0.690 total time=   0.2s
[CV 3/5] END ...........Model: ridge__alpha=0.1;, score=0.693 total time=   0.2s
[CV 4/5] END ...........Model: ridge__alpha=0.1;, score=0.700 total time=   0.2s
[CV 5/5] END ...........Model: ridge__alpha=0.1;, score=0.689 total time=   0.2s
[CV 1/5] END ...........Model: ridge__alpha=1.0;, score=0.682 total time=   0.2s
[CV 2/5] END ...........Model: ridge__alpha=1.0;, score=0.679 total time=   0.2s
[CV 3/5] END ...........Model: ridge__alpha=1.0;, score=0.683 total time=   0.2s
[CV 4/5] END ...........Model: ridge__alpha=1.0;, score=0.687 total time=   0.2s
[CV 5/5] END ...........Model: ridge__alpha=1.0;, score=0.679 total time=   0.2s
[CV 1/5] END ..........Model: ridge__alpha=10.0;, score=0.674 total time=   0.2s
[CV 2/5] END ..........Model: ridge__alpha=10.0;, score=0.671 total time=   0.2s
[CV 3/5] END ..........Model: ridge__alpha=10.0;, score=0.676 total time=   0.2s
[CV 4/5] END ..........Model: ridge__alpha=10.0;, score=0.679 total time=   0.2s
[CV 5/5] END ..........Model: ridge__alpha=10.0;, score=0.671 total time=   0.2s
[CV 1/5] END .........Model: ridge__alpha=100.0;, score=0.673 total time=   0.2s
[CV 2/5] END .........Model: ridge__alpha=100.0;, score=0.670 total time=   0.2s
[CV 3/5] END .........Model: ridge__alpha=100.0;, score=0.675 total time=   0.2s
[CV 4/5] END .........Model: ridge__alpha=100.0;, score=0.678 total time=   0.2s
[CV 5/5] END .........Model: ridge__alpha=100.0;, score=0.670 total time=   0.2s
[CV 1/5] END ........Model: ridge__alpha=1000.0;, score=0.672 total time=   0.2s
[CV 2/5] END ........Model: ridge__alpha=1000.0;, score=0.670 total time=   0.2s
[CV 3/5] END ........Model: ridge__alpha=1000.0;, score=0.675 total time=   0.2s
[CV 4/5] END ........Model: ridge__alpha=1000.0;, score=0.677 total time=   0.2s
[CV 5/5] END ........Model: ridge__alpha=1000.0;, score=0.670 total time=   0.2s
[CV 1/5] END .......Model: ridge__alpha=10000.0;, score=0.668 total time=   0.2s
[CV 2/5] END .......Model: ridge__alpha=10000.0;, score=0.666 total time=   0.2s
[CV 3/5] END .......Model: ridge__alpha=10000.0;, score=0.670 total time=   0.2s
[CV 4/5] END .......Model: ridge__alpha=10000.0;, score=0.673 total time=   0.2s
[CV 5/5] END .......Model: ridge__alpha=10000.0;, score=0.665 total time=   0.2s
[CV 1/5] END ......Model: ridge__alpha=100000.0;, score=0.623 total time=   0.2s
[CV 2/5] END ......Model: ridge__alpha=100000.0;, score=0.622 total time=   0.2s
[CV 3/5] END ......Model: ridge__alpha=100000.0;, score=0.622 total time=   0.2s
[CV 4/5] END ......Model: ridge__alpha=100000.0;, score=0.627 total time=   0.2s
[CV 5/5] END ......Model: ridge__alpha=100000.0;, score=0.620 total time=   0.2s

Best Grid mean score (r2): 0.6950
Best Grid parameters: Model: ridge__alpha: 0.001

Predictions:
                         Train            Test
MSE:            49,870,544.5638 48,003,474.3807
RMSE:                7,061.9080      6,928.4540
MAE:                 4,914.2174      4,852.3211
R^2 Score:               0.6956          0.7002
```


Training Residuals - Iteration 26
40000
20000
0
−20000
−40000
0
10000
20000
30000
40000
50000
60000
70000
80000

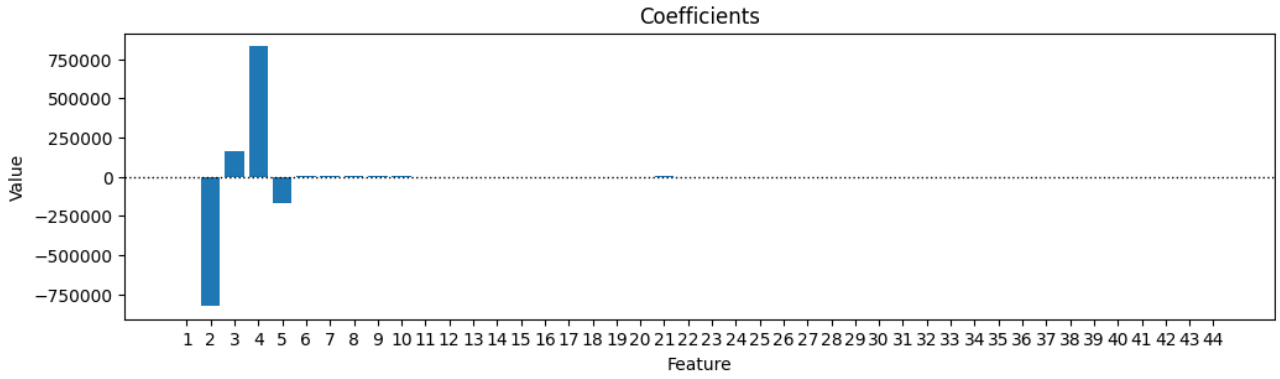

Test Residuals - Iteration 26
40000
20000
0
−20000
−40000
0
10000
20000
30000
40000
50000
60000
70000
80000


Training Predicted vs. Actual - Iteration 26
Predicted
80000
60000
40000
20000
0
0
10000
20000
30000
40000
50000
60000
70000
80000
Actual


Test Predicted vs. Actual - Iteration 26
Predicted
80000
60000
40000
20000
0
0
10000
20000
30000
40000
50000
60000
70000
80000
Actual

Coefficients:

	Feature	Value
1	odometer	-674.6739
2	year	-826,494.4426
3	odometer	162,588.6406
4	year^2	832,447.6580
5	year odometer	-167,437.4697
6	odometer^2	2,195.2024
7	1	1,762.3661
8	2	2,883.9633
9	3	1,270.1109
10	4	797.9745
11	5	585.4115
12	6	755.1616
13	7	381.9297
14	8	-399.8954
15	9	187.2655
16	10	70.1329
17	11	501.0412
18	12	766.4941
19	13	97.6233
20	14	299.9285
21	15	816.6322
22	16	406.0007
23	17	630.9520
24	18	170.5791
25	19	-492.5144
26	21	-856.8677
27	22	96.3184
28	23	-734.7784
29	24	-97.2549
30	25	-583.2509
31	26	-905.9158
32	27	-324.3390
33	28	-686.0597
34	29	-658.8584
35	30	-16.8122
36	31	-96.2484
37	32	-277.3052
38	33	-749.4297
39	34	-282.7767
40	35	-63.3935
41	36	-401.5427
42	37	-310.6989
43	38	-570.6685
44	39	-8.8205



Observation: This is the X10 dataset, which is a result of manual encoding (Ordinal and OHE) and then selecting the top 40 correlated features from the matrix. In addition, I applied the stricter thresholds: Price > \$1000, Odometer > 1000, and upper thresholds: Price < \$80k, Odometer < 400k, and added Year > 1980. We applied GridSearchCV with Ridge exploring values of alpha between 0.001 and 100,000. The best alpha value was 0.001, and this produced a Test R^2 of **0.70** with Log and Polynomial degree 2.

Review Model Performance

Here's the dataframe that tracked the results of each iteration, including a definition of the pipeline, and a note that was manually entered. It has all the major error and performance scores, plus a timestamp.

In [ ]:
```
#pd.reset_option('display.float_format')
```

In [550…
```
#my.results_df
```

In [552…
```
# Set dataframe output to not show scientific notation
pd.set_option('display.float_format', '{:.2f}'.format)
my.results_df[['Iteration','Train MSE','Test MSE','Train MAE','Test MAE','Test R^2 Score','Train R^2 Sc
```

Out[552]:

	Iteration	Train MSE	Test MSE	Train MAE	Test MAE	Test R^2 Score	Train R^2 Score	Note
0	1	110769211.73	109990192.37	6550.34	6542.03	0.46	0.47	Log. Ordinal. OHE. Linear Regression.
1	2	112586311.13	109536664.67	6619.97	6577.54	0.47	0.46	X2 top correlated features after OHE and Ordinal. Linear Regression.
2	3	91234007.22	89786932.21	5749.97	5735.07	0.56	0.57	X. Ordinal. OHE. Poly2. Standard Scaler. Linear Regression.
3	4	95274957.36	93217551.03	5897.83	5865.67	0.54	0.55	X2 top correlated features after OHE and Ordinal. Poly 2. Standard. Linear Regression.
4	5	90228040.64	89085064.64	5725.13	5716.81	0.57	0.57	X. Ordinal. OHE. Poly3. Standard Scaler. Linear Regression.
5	6	342459444808332.69	25671639015069.08	488792.51	387735.66	-0.02	0.00	X3 with outliers. Ordinal. OHE. Poly2. Standard Scaler. Linear Regression.
6	7	92406276.22	91000540.17	5747.63	5736.07	0.56	0.56	X4 no Paint/State. Manual Ordinal. OHE. Poly2. Standard Scaler. Linear Regression.
7	8	91324878.34	90271236.21	5717.30	5712.31	0.56	0.56	X4 no Paint/State. Manual Ordinal. OHE. Poly3. Standard Scaler. Linear Regression.
8	9	91466320.16	93509202.40	6553.69	6583.53	0.48	0.49	X5 Price < $100k, Odometer < 400k. Manual Ordinal. OHE. Linear Regression.
9	10	71682968.19	73580803.81	5478.24	5536.67	0.59	0.60	X5 Price < $100k, Odometer < 400k. Manual Ordinal. OHE. Poly2. Standard Scaler. Linear Regression.
10	11	71426631.02	73283882.97	5464.70	5524.61	0.60	0.60	X5 Price < $100k, Odometer < 400k. Manual Ordinal. OHE. Poly3. Standard Scaler. Linear Regression.
11	12	93461430.66	93388622.00	6616.01	6605.47	0.48	0.48	X6 no Outlier removal via Cluster. $10 < Price < $100K, 10 < Odometer < 400k. Ordinal. OHE. Linear Regression.
12	13	74233053.71	76346117.08	5582.48	5605.74	0.58	0.59	X6 no Outlier removal via Cluster. 10 < Price < 100K, 10 < Odometer < 400k. Ordinal. OHE. Poly2. Standard. Linear Regression.
13	14	73979981.93	76037667.86	5569.13	5591.65	0.58	0.59	X6 no Outlier removal via Cluster. 10 < Price < 100K, 10 < Odometer < 400k. Ordinal. OHE. Poly3. Standard. Linear Regression.
14	15	88171916.59	88994396.23	6552.28	6564.99	0.57	0.57	X7 no Duplicate or Outlier removal via Cluster. 10 < Price < 100K, 10 < Odometer < 400k, Year > 1960. Ordinal. OHE. Linear Regression.
15	16	69143493.16	69618322.50	5602.12	5604.84	0.66	0.66	X7 no Duplicate or Outlier removal via Cluster. 10 <

	Iteration	Train MSE	Test MSE	Train MAE	Test MAE	Test R^2 Score	Train R^2 Score	Note
								Price < 100K, 10 < Odometer < 400k, Year > 1960. Ordinal. OHE. Poly2. Standard. Linear Regression.
16	17	69076642.19	69557184.64	5600.20	5603.15	0.66	0.66	X7 no Duplicate or Outlier removal via Cluster. 10 < Price < 100K, 10 < Odometer < 400k, Year > 1960. Ordinal. OHE. Poly3. Standard. Linear Regression.
17	18	69877946.96	70505629.17	5990.98	5989.03	0.65	0.65	X8 no Duplicate or Outlier removal via Cluster. 1000 < Price < 100K, 1000 < Odometer < 400k, Year > 1980. Ordinal. OHE. Linear Regression.
18	19	54646989.82	55007322.64	5186.34	5187.68	0.73	0.72	X8 no Duplicate or Outlier removal via Cluster. 1000 < Price < 100K, 1000 < Odometer < 400k, Year > 1980. Ordinal. OHE. Poly2. Standard. Linear Regression.
19	20	54571993.66	54901155.85	5183.23	5183.97	0.73	0.72	X8 no Duplicate or Outlier removal via Cluster. 1000 < Price < 100K, 1000 < Odometer < 400k, Year > 1980. Ordinal. OHE. Poly3. Standard. Linear Regression.
20	21	62146559.36	60720230.63	5454.03	5400.05	0.65	0.64	X9 top correlated features. 1000 < Price < 100K, 1000 < Odometer < 400k, Year > 1980. Linear Regression.
21	22	54443553.59	53070200.62	5024.67	4979.28	0.69	0.69	X9 top correlated features. 1000 < Price < 100K, 1000 < Odometer < 400k, Year > 1980. Poly 2, Linear Regression.
22	23	51905704.10	50450993.54	4495.28	4442.44	0.68	0.68	X10 top correlated features. 1000 < Price < 80K, 1000 < Odometer < 400k, Year > 1980. Log. Poly 2, Linear Regression.
23	24	56962963.32	55350822.97	5352.94	5307.26	0.65	0.65	X10 top correlated features. 1000 < Price < 80K, 1000 < Odometer < 400k, Year > 1980. Log. Poly 2, Lasso, GridSearchCV alpha 0.001 - 100.
24	25	49912437.50	48041369.56	4920.78	4857.89	0.70	0.70	X10 top correlated features. 1000 < Price < 80K, 1000 < Odometer < 400k, Year > 1980. Poly 2, Ridge, GridSearchCV alpha 0.001 - 100,000.
25	26	49870544.56	48003474.38	4914.22	4852.32	0.70	0.70	X10 top correlated features. 1000 < Price < 80K, 1000 < Odometer < 400k, Year > 1980. Log, Poly 2, Ridge, GridSearchCV alpha 0.001 - 100,000.

Save Results to a File

In [553…
```
# Save a timestamped backup file
now = datetime.now()
timestamp_str = now.strftime('%Y%m%d_%H%M%S')
my.results_df.to_csv(f'results_df_{timestamp_str}.csv')
```

Prepare the Results for Plotting

In [554…
```
# Melt the dataframe to a long form
results_df_long = my.results_df.melt(id_vars='Iteration',
                  value_vars=['Train MSE', 'Test MSE', 'Train RMSE', 'Test RMSE', 'Train MAE', 'Test MA
                  var_name='Metric', value_name='Value')

results_df_long['Iteration'] = results_df_long['Iteration'].astype(int)
```

In [555…
```
# Filtering R^2 and MAE data separately
r2_data = results_df_long[results_df_long['Metric'].isin(['Train R^2 Score', 'Test R^2 Score'])]
mae_data = results_df_long[results_df_long['Metric'].isin(['Train MAE', 'Test MAE'])]
```

In [556…
```
r2_data.query("Metric == 'Test R^2 Score'").sort_values(by='Value', ascending=False)[0:5]
```

Out[556]:

	Iteration	Metric	Value
201	20	Test R^2 Score	0.73
200	19	Test R^2 Score	0.73
207	26	Test R^2 Score	0.70
206	25	Test R^2 Score	0.70
203	22	Test R^2 Score	0.69

In [557…
```
mae_data.query("Metric == 'Test MAE'").sort_values(by='Value', ascending=True)[0:5]
```

Out[557]:

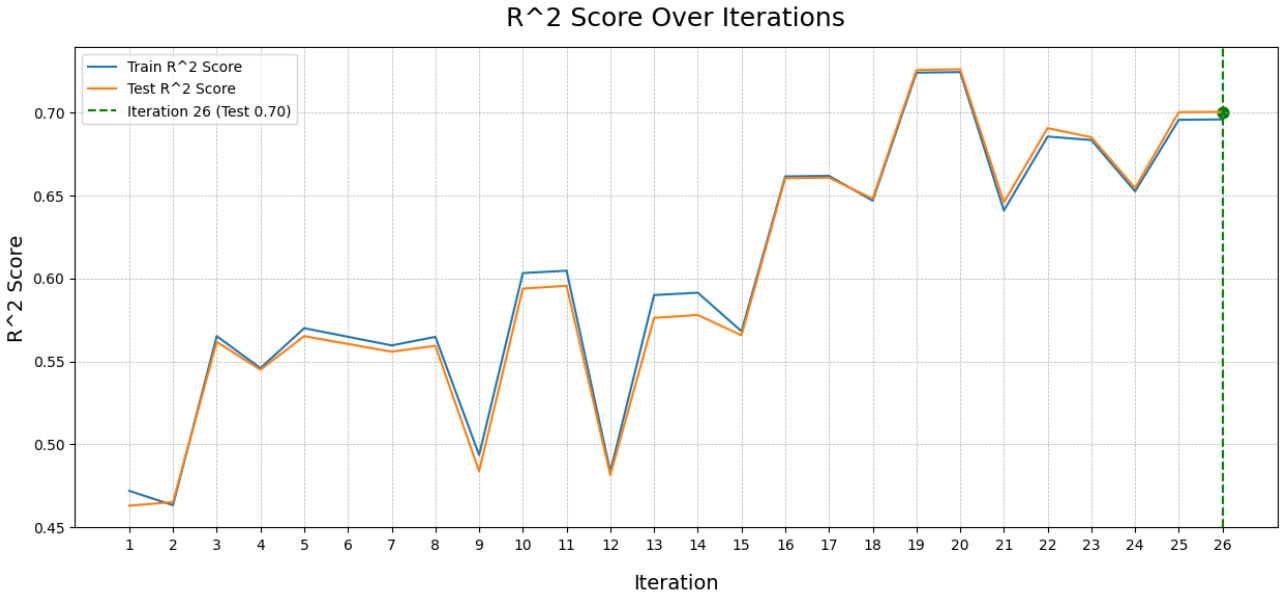
	Iteration	Metric	Value
152	23	Test MAE	4442.44
155	26	Test MAE	4852.32
154	25	Test MAE	4857.89
151	22	Test MAE	4979.28
149	20	Test MAE	5183.97

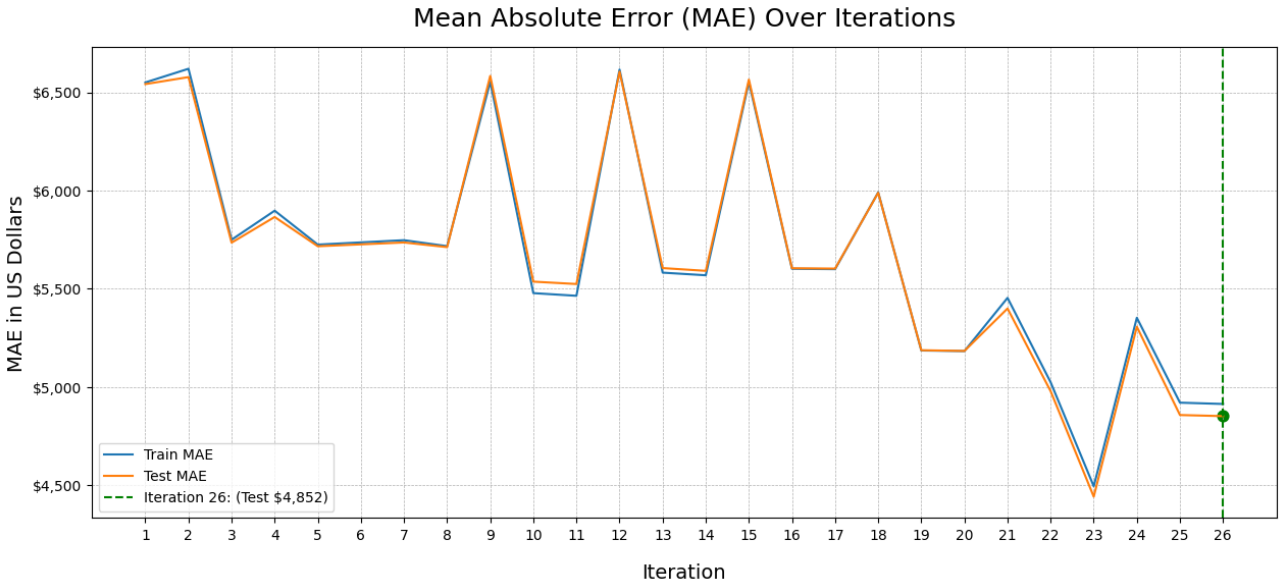
In [563…
```
# Drop iteration 6 which had the extreme outliers so we can the differences in other iterations
r2_data_drop = r2_data.drop([161, 187])
mae_data_drop = mae_data.drop([109, 135])
```

Plot the Results on a Chart

In [566…
```
# Plot the R^2 data
plt.figure(figsize=(15, 6))
sns.lineplot(data=r2_data_drop, x='Iteration', y='Value', hue='Metric')
plt.title('R^2 Score Over Iterations', fontsize=18, pad=15)
plt.xlabel('Iteration', fontsize=14, labelpad=15)
plt.ylabel('R^2 Score', fontsize=14)
plt.xticks(mae_data['Iteration'].unique())
plt.grid(True, which='both', linestyle='--', linewidth=0.5)
plt.axvline(x=26, color='green', linestyle='--', label='Iteration 26 (Test 0.70)')
plt.scatter(26, 0.70, color='green', s=60)
plt.legend(loc='best')
plt.show()

# Plot the MAE data
plt.figure(figsize=(15, 6))
sns.lineplot(data=mae_data_drop, x='Iteration', y='Value', hue='Metric')
plt.title('Mean Absolute Error (MAE) Over Iterations', fontsize=18, pad=15)
```

```
plt.xlabel('Iteration', fontsize=14, labelpad=15)
plt.ylabel('MAE in US Dollars', fontsize=14)
plt.gca().yaxis.set_major_formatter(FuncFormatter(my.thousand_dollars))
plt.xticks(mae_data['Iteration'].unique())
plt.grid(True, which='both', linestyle='--', linewidth=0.5)
plt.axvline(x=26, color='green', linestyle='--', label='Iteration 26: (Test $4,852)')
plt.scatter(26, 4852.32, color='green', s=60)
plt.legend(loc='best')
plt.show()
```





Summary of Findings

Descriptive Statistics

Here's what we learned about the listings in this dataset. Keep in mind these are self-reported by the sellers that are creating these listings, so the data is likely biased and not entirely accurate:

Numerical

After the data was cleaned to remove outliers and junk data, we can say the following about Price, Odometer, and Year:

- **Price:** The average price was **\$16,520**. The majority of the listings were between \$6,750 and \$22,380. Some were listed close to zero, and a number of them went much higher (\$349,999 for example).
- **Odometer:** The average odometer reading was **103,346**. The majority of the listings were between 51,806 and 142,929. Some were listed close to zero, and a number of them went much higher (900,000 for example).
- **Year:** The average year of the vehicle was **2010**. The majority of the listings were between 2007 and 2016. Some went as far back as 1923, with the most recent being 2022.

Categorical

- Of the 42 Manufacturers, **Ford** (16.63%) was the most popular, followed by **Chevrolet** (12.9%) and **Toyota** (8.01%)
- Surprisingly, **Ferraris** do get sold on Craigslist – 95 times in this dataset!
- The majority of the vehicles were in **Good** (28.45%) or **Excellent** (23.77%) condition. However, many people did not specify a condition (40.79%). Perhaps they did not want to advertise anything negative.
- The most popular engine configuration was **6 Cylinders** (22.06%), followed by **4 Cylinders** (18.19%) and **8 Cylinders** (16.88%). However, most people did not specify the cylinders (41.62%). Perhaps they did not know.
- For the drive train, **4-Wheel Drive (4WD)** (30.9%) was the most popular, followed by **Front-Wheel Drive (FWD)** (24.72%), and then **Rear-Wheel Drive (RWD)** (13.8%). A large number did not specify the drive train (30.59%). Perhaps they did not know.
- Most people did not specify the size of the vehicle (71.77%), which is surprising. Perhaps they did not know how to properly classify it. After that, the order was: **Full-Size** (14.87%), **Mid-Size** (8.08%), **Compact** (4.54%), and **Sub-Compact** (0.75%).
- **Sedan** (20.39%) was the most popular type, followed by **Sport-Utility Vehicle (SUV)** (18.1%) and **Pickup** (10.19%). However, 21.75% did not specify a type.
- **Gas** was the dominant fuel type at 83.44%
- Almost all the titles were **Clean** (94.9%). There are some interesting title status values: Rebuilt, Lien, Salvage, Parts Only.
- **Automatic** was the dominant transmission (78.83%)
- **White** was the most popular color, after the 30.5% that did not specify a color. **Purple** was the least popular color (0.16%)
- Most of the listings were in **CA** (11.86%) > **FL** (6.68%) > **TX** (5.38%) > **NY** (4.54%), in that order. **Columbus, OH** and **Jacksonville, FL** were the most popular listing regions.

Business Objectives

1. Identify key drivers for used car prices

Here are the main drivers of used car prices.

Characteristics that Increase Price

- **Year** is slightly correlated with Price (0.28), which makes sense. Newer cars tend to cost more than older cars, but we saw from our plots that there may be a class of older cars that cost more, which may be holding this correlation down.
- **Cylinders** is slightly correlated with Price (0.25), suggesting the more cylinders, the more expensive the vehicle.
- **Diesel** (0.24) and **4WD** (0.22) are slightly correlated with Price, suggesting those type of work or offroad vehicles command a premium.
- **Condition** (0.18) is slightly correlated with Price, suggesting cars that are in better condition will sell for more. But it's not the main driver of price (in this dataset, at least)
- **Pickup** (0.18), **Truck** (0.16), and **Ram** (0.14) are slightly correlated with Price, suggesting these types of vehicles are slightly more expensive.
- **Size** is slightly correlated with Price (0.13), suggesting larger vehicles cost a little more.
- In addition to Ram, a few manufacturers also seem to have a slight correlation with Price: **Ferrari** (0.12), **Porsche** (0.08), **GMC** (0.08), **Tesla** (0.07), and **Ford** (0.07).

Characteristics that Decrease Price

- **Odometer** is moderately negatively correlated with Price (-0.42), which makes sense. Vehicles with more mileage are going to sell for less. This is the **strongest relationship** in our data set, and may be the main independent variable.
- **Gas** (-0.23) and **FWD** (-0.23) are slighlty negatively correlated with Price, suggesting those less complicated fuel and drive trains are cheaper.
- Conversely, **Sedan** (-0.16) and **Honda** (-0.1) are slightly negatively correlated with Price, suggesting they are slightly cheaper options.
- In addition to Honda, a few manufacturers seem to have a weak negative correlation with Price: **Nissan** (-0.07), **Chrysler** (-0.06), and **Hyundai** (-0.06).
- And lastly, vehicles with **Automatic** transmission (-0.08), or that are **Hatchbacks** (-0.06) or **Minivans** (-0.06), tend to be slightly cheaper.

2. Provide recommendations on what consumers value

Here is how used car dealears can adjust their inventory to align with consumer wants and needs. This assumes higher prices represent consumer demand, and lower prices represent a lack of demand:

Increase Supply of the Following

- Modern cars in recent years, with low odometer readings, in good condition
- Cars with more cylinders (8-cylinder, 6-cylinder or more) and larger size (Full-size, Mid-size)
- Diesel and 4WD vehicles classified as Pickup or Truck
- Stock more of these manufacturers: Ram, Ferrari, Porsche, GMC, Tesla, Ford

Decrease Supply of the Following

- Older cars, with high odometer readings, in poor condition (or with unknown condition)
- Cars with less cylinders (4-cylinder or lower) and smaller size
- Gas and FWD vehicles with Automatic transmission
- Vehicle of type Sedan, Hatchback or Minivan
- Stock less of these manufacturers: Honda, Nissan, Chrysler, Hyundai

It's important to note that all of these car types exist for a reason: There are people that really want an automatic, gas-powered sedan; or that need a hatchback or minivan; or love Hondas, or older cars. There is likely some demand for these, they just don't seem to command a premium.

3. Create price prediction model

After several iterations, we were able to deliver a model that can predict price with roughly a 70% "accuracy." Here are some things to note about this model:

- The model accounts for about **70% of the variance** in used car prices (R^2 0.70)
- The model has an average error of about **$4,852 (MAE)**
- The model is based on a **filtered dataset** of used car listings with prices between \$1,000 and \$80,000, odometers between 1,000 and 400,000, and older than 1980.
- We believe this is representative of the typical range of **most used cars sold at dealers**
- Given this range, it excludes certain classic cars or antiques that are likely priced on a different model
- **All duplicates and junk data were stripped** from the dataset, so it should generalize well to unseen data
- The model is a Linear Regression model based on the **top 40 correlated features** from our analysis
- The main drivers of the model (its top coefficients) are: **Odometer, Year, Ferraris, Teslas, Porsches, and Diesel fuel.**
- Some technical details (feel free to ignore): Pipeline was: Ordinal encoding, One-Hot Encoding, Log transformation, Polynomial degree 2 transformation, Ridge, GridSearchCV of alpha 0.001 - 100,000. Best alpha was 0.001.

A detailed log of the model iterations can be found here: results_df_20230829_024020.csv

Limitations

There are a few limitations in the data:

- **Junk Data:** This dataset was filled with a lot of junk data, duplicate listings, missing values, and required a significant amount of cleaning to remove duplicates or impossible values. Some of it is clearly not accurate.
- **Cleaning Impact:** As part of our data cleaning, a number of records were removed. Only 44.76% of the original dataset was retained. We may have unintentionally removed data that was legitimate, which could impact our conclusions.
- **Biased Listings:** The basis of this dataset are postings on Craigslist by sellers, so there is clearly a bias in them – they want to sell their vehicle.
- **Advertisements Only:** The numbers in this dataset are only the listing price, we do not know (a) if this transaction ever closed, nor (b) what the final sales price was.
- **Dataset Questions:** Some of the listings could also be from people that are looking to buy ("Wanted" or "Looking for" ads), it's not clear if they are included in this dataset. But that could explain the large amount of missing values.

Next Steps and Recommendations

To better understand the drivers of supply and demand in used car prices, this project could be followed up in the following ways:

- **New Dataset:** Obtain a dataset of actual verified sales transactions – not just the advertised listings. This will increase our confidence and accuracy of any findings significantly.
- **Survey Consumers:** Conduct a survey of consumers to find out what they value or care about when looking for a used car. The dataset we have is possibly one-sided, representing only the supply of sellers. But there could be a mismatch in supply and demand.
- **Different Models:** In this project, we only used linear regression models, but we could also explore alternative models that may be able to better utilize the existing dataset.

In [ ]: