INSTRUCTIONS ON RUNNING THIS NOTEBOOK

- It requires nltk and tries to download it using !pip install command
- It by default it is using a large training set so the model tuning and feature importance calcluations can be accurate
- It is by default set to NOT , REPEAT NOT, RUN TIME CONSUMING CALCULATIONS.
- As you work throught the notebook, some cells will ask you to manually set OK_TO_RUN_TUNING or OK_TO_RUN_FEAT_IMPORT to True.
- Doing so will trigger the lengthy calcuations - up to an hour
- I have included images of the key charts in the README.MD file and in this notebook if you want to see the results without running the full notebook
- With the default OK_TO_RUN_TUNING or OK_TO_RUN_FEAT_IMPORT set to False, the notebook runs in about 20 seconds

What drives the price of a car?

OVERVIEW

In this application, you will explore a dataset from kaggle. The original dataset contained information on 3 million used cars. The provided dataset contains information on 426K cars to ensure speed of processing. Your goal is to understand what factors make a car more or less expensive. As a result of your analysis, you should provide clear recommendations to your client -- a used car dealership -- as to what consumers value in a used car.

CRISP-DM Framework

No description has been provided for this image

To frame the task, throughout our practical applications we will refer back to a standard process in industry for data projects called CRISP-DM. This process provides a framework for working through a data problem. Your first step in this application will be to read through a brief overview of CRISP-DM here. After reading the overview, answer the questions below.

Business Understanding

From a business perspective, we are tasked with identifying key drivers for used car prices. In the CRISP-DM overview, we are asked to convert this business framing to a data problem definition. Using a few sentences, reframe the task as a data task with the appropriate technical vocabulary.

Background: Used car sales dealerships want to fine tune their inventory to improve profits. Their strategy is to identify what factors make a car more or less expensive. Implicit in this strategy is understanding profit margin and return on investment, not just selling more and higher priced vehicles. However, the focus of this study is on understanding what features in a car customers value. We make the assumption that customers express this 'value' by paying a higher price for cars with more valuable features versus those with less valuable features.

Data Problem Definition:

Business Objective:

- Identify the car features in the data that have the strongest positive correlation with selling price.
- The ability to identify these corelations should be part of a bigger discussion with stakeholders about the wider business project goals. Assuming ROI of investment in inventory is the ultimate goal, clarify that developing a causal-based model and subsequently factoring in profit margins of car features would be important 2nd and 3rd stages of the project.
- This correlation study, along with future causal and profit-margin analysis projects can enable dealers to optimize the ROI of their businesses by more systematically choosing their inventory of cars to sell.

Data Analytics Objective: Develop a model and process to ingest used car data, analyze it and rank the most significant features of a car and the least significant features of a car. These rankings will be made based on how they impact the price at which the car sells. Use the most significant features to predict the price that customers would pay for a given set of features.

Data Sources: Kaggle data set of information about 426K used cars

Key Performance Indicators (KPIs): % of rows of data with valid data in most feature columns: If too much data is missing or invalid, then steps to address this issue must be taken prior to successfully completing the project

Feature Importance: Relative feature importance using the coefficients of a linear regression model tuned and regularized for this context

Change in Error by Feature: Differences in mean squared error for several linear regression models using different subsets of features

Error for Optimized Model: Mean squared error of the best performing linear regression model

Overfitting Check: Difference in training error vs validation error across multiple hyper parameters

Success Criteria:

- The ranking process identifies the top 5 MOST signifcant features correlated to sale price of a car based on above KPIs

- The optimized model has a test data MSE less than 5% of the average price of the cars in inventory

Our result at this stage will be a correlation study. An additional success criteria for this stage is that the analysis guides the efficient design of a randomized control experiments to determine causal impact of features on sale price.

In [ ]:

```

```

Data Understanding

After considering the business understanding, we want to get familiar with our data. Write down some steps that you would take to get to know the dataset and identify any quality issues within. Take time to get to know the dataset and explore what information it contains and how this could be used to inform your business understanding.

Steps to take to gain understanding:

- Evaluate total number of rows and also number of rows per various categorical groupings.
- Is there a massive amount of data to be managed such that simple queries, regressions and etc will be time-consuming and expensive?
- Examine the schema (structure and type) of the data. Identify the sales price numerical field(s).
- Are any fields compound or nested data that needs to be further processed (de-normalized, flattended) to be understood?
- Is the data spread out across many data sources such as a relational datamodel with foreign keys?
- Does the data need to be concatenated over multiple similar data sources?
- Visually review the data distribution and range of values of the data. Look for obvious patterns using histograms and box plots.
- Compare subsets of the data by feature columns grouping to look for relationships like correlation.
- Note if there are major imbalances in the category groupings of the data.
- Look for nulls, suspicious duplicates, outliers, and invalid values.
- Look for data mistakes/inconsistencies in which two domain values are different on different rows, but likely meant to be the same value. Example: 'Blue' and 'blue'.
- Look at mode, frequency and averages of the total and a variety of subgroups of the data, especially in regards to the fields holding the sale price.
- Identify if the data has a time-series aspects like date of sale. Examine the range and distribution of sales price along these time-series axes.

Code and observations regarding data structure and size

In [1]:
```
# All imports needed to run this notebook code
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from sklearn.calibration import LabelEncoder
from sklearn.pipeline import Pipeline
from sklearn.model_selection import StratifiedKFold, cross_val_score, train_test_sp
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SequentialFeatureSelector, SelectFromModel
from sklearn.model_selection import GridSearchCV
import math
import re
import string
import itertools
import time
import random
from sklearn.preprocessing import PolynomialFeatures
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import make_column_transformer
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error
#!pip install statsmodels
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.decomposition import PCA
from sklearn.feature_selection import VarianceThreshold, SelectKBest, f_classif
import pickle
#!pip install nltk
import nltk
from sklearn.preprocessing import FunctionTransformer
from sklearn.inspection import permutation_importance
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import make_pipeline
from sklearn import set_config
import winsound # Remove this if not on windows machine. Using to signal when a lon
```

In [2]:
```
def beep():
  winsound.Beep(frequency=1000, duration=7000)
```

In [3]:
```
car_df = pd.read_csv('data/vehicles.csv')
```

In [4]:
```
print(f"Car df number of rows: {car_df.shape[0]} and column count: {car_df.shape[1]
```

```
Car df number of rows: 426880 and column count: 18
```

In [5]:
```
car_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 [6]:

```python
print("@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@")
print("@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@")
print("@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@")
#print(f"EXPERIMENT FILTERING TO A SINGLE TYPE")
state_filter = ['ny']
print(f"Running by State of {state_filter}")
car_state_df = car_df[car_df['state'].isin(state_filter)]
car_df = car_state_df
#print("running as normal")
print("@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@")
print("@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@")
print("@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@")
```

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
Running by State of ['ny']
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
```

Comments on the structure:

- The 'id' fields looks to be a unique value per row
- 'price' seems to be the target value and the rest besides ID are features
- The structure of the data source is simple - one file. No need for joins, concatenation or integrations
- There are a reasonable number of rows and columns. Not too many to work with using standard tools.

Code block to help with visual analysis of distribution

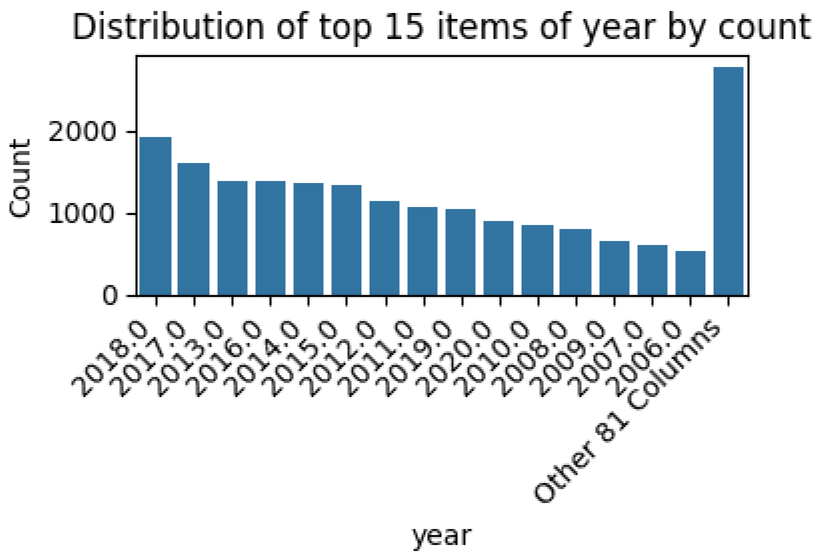
In [7]:
```
# eval_col_counts() gives a sense of the distribution of different distinct values
# For columns with less distinct values less than max_detail(default=15), a bar cha
# with that value of that column is show for each distinct value.
# For columns with more than max_detail distinct values, it shows the top max_detai
# It also creates a column 'Rest of values' and shows the count of the remaining va
# This approach lets us get a sense of the distribution visually even for categorie
# You can sort the bar chart by the highest count or by the column value (e.g., sho
# To do the latter, set sort_by_col parameter to True

import pandas as pd

def get_value_counts(data, column_name, sort_by_col=False):
  # Use value_counts() to get the counts and reset the index to create a DataFrame
  if sort_by_col:
      return data[column_name].value_counts().reset_index(name='count').sort_values
  else:
      return data[column_name].value_counts().reset_index(name='count')

def get_all_value_counts(data, col_list):
    # Assuming your data is loaded into a pandas DataFrame named 'df'
    for column_to_count in col_list:
        value_counts_df = get_value_counts(car_df, column_to_count)
        print(value_counts_df)


def eval_col_counts(data, col_list, max_detail = 15, sort_by_col = False):
    for column_to_count in col_list:
        value_counts_df = get_value_counts(data, column_to_count, sort_by_col)
        unq_count = value_counts_df.shape[0]
        print(f"{column_to_count} has {unq_count} distinct values")
        disp_count = min(max_detail,unq_count)
        print(f"See {disp_count} of them")
        print(value_counts_df.head(disp_count))
        if unq_count < 10:
            plt_title = f'Distribution of {column_to_count}'
            plt_data = value_counts_df.copy()
        else:
            plt_title = f'Distribution of top {max_detail} items of {column_to_coun
            plt_data_slice = value_counts_df.head(max_detail)
            plt_data = plt_data_slice.copy()
            plt_data_sum = sum(plt_data['count'])
            all_data_sum = sum(value_counts_df['count'])
            rest_data_sum = all_data_sum - plt_data_sum
            print(f'Count of the rest of the values not shown: {rest_data_sum}')
            # Create a dictionary for the new row
            #new_row = pd.Series({column_to_count: 'Other Columns', 'count': rest_d
            plt_data.loc[len(plt_data)] = [f'Other {unq_count-max_detail} Columns',

        plt.figure(figsize=(4, 3))  # Adjust figure size as needed
        sns.barplot(x=column_to_count, y="count", data=plt_data)
        plt.xlabel(column_to_count)
        plt.ylabel('Count')
```

```
        plt.title(plt_title)
        plt.xticks(rotation=45, ha='right')  # Rotate category labels for readabili
        plt.tight_layout()  # Adjust spacing between elements
        plt.show()

        print("----")
```

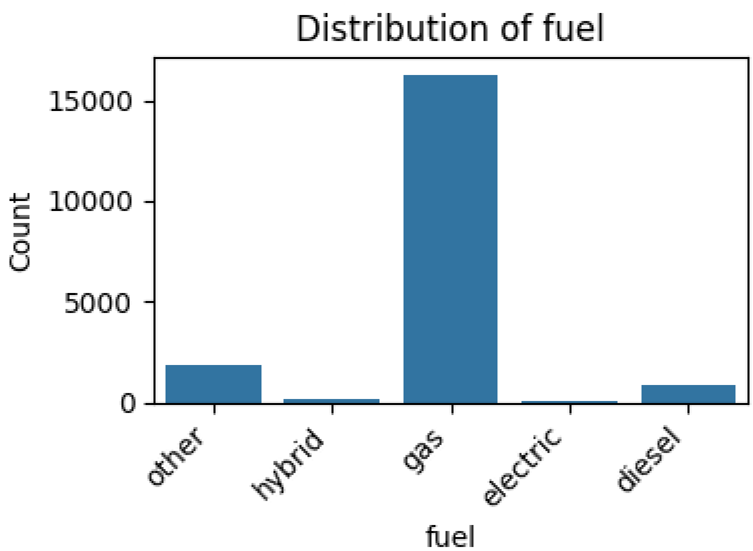
Distribution of data charts

In [8]:
```
# Grader: I have run with all columns but for submission only listing interesting c
#col_list = ['region', 'year', 'manufacturer', 'model', 'condition', 'cylinders','f
#             'odometer','title_status', 'transmission','drive','size','type','paint
freq_sort_col_list = [ 'manufacturer', 'model', 'condition', 'cylinders','fuel',
            'odometer','title_status', 'transmission','drive','size','type','paint_
max_detail = 15
freq_sort_col_list = ['year']
eval_col_counts(car_df, freq_sort_col_list, max_detail = max_detail, sort_by_col =

col_sort_col_list = ['fuel','type','manufacturer', 'state', 'size']
eval_col_counts(car_df, col_sort_col_list, max_detail = max_detail,  sort_by_col =
```

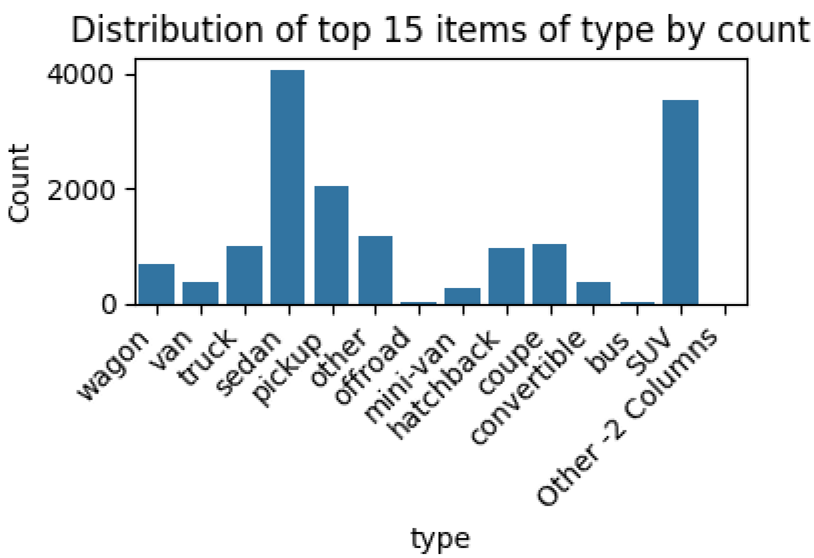
```
year has 96 distinct values
See 15 of them
      year  count
0   2018.0   1927
1   2017.0   1613
2   2013.0   1394
3   2016.0   1375
4   2014.0   1362
5   2015.0   1339
6   2012.0   1151
7   2011.0   1058
8   2019.0   1034
9   2020.0    894
10  2010.0    836
11  2008.0    810
12  2009.0    646
13  2007.0    614
14  2006.0    540
Count of the rest of the values not shown: 2781
```



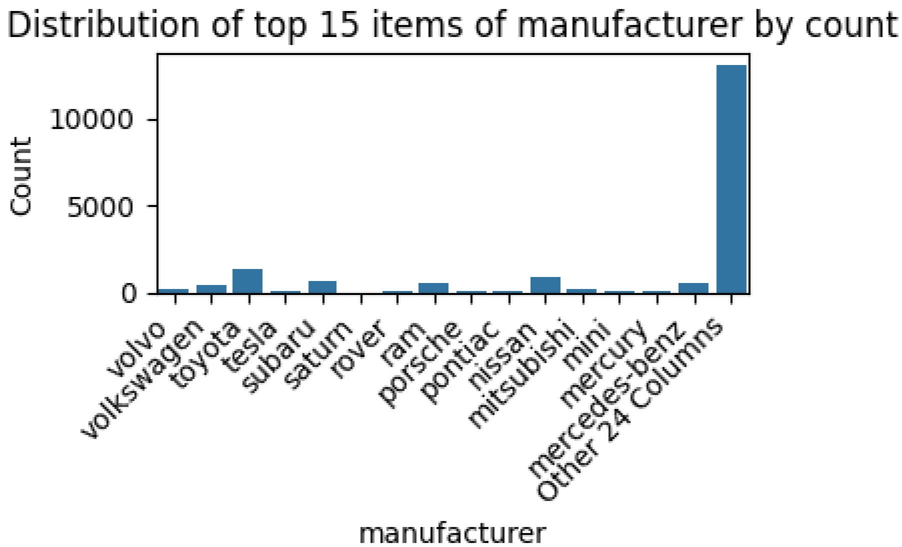

```
----
fuel has 5 distinct values
See 5 of them
       fuel  count
1     other   1892
3    hybrid    186
0       gas  16288
4  electric     74
2    diesel    870
```



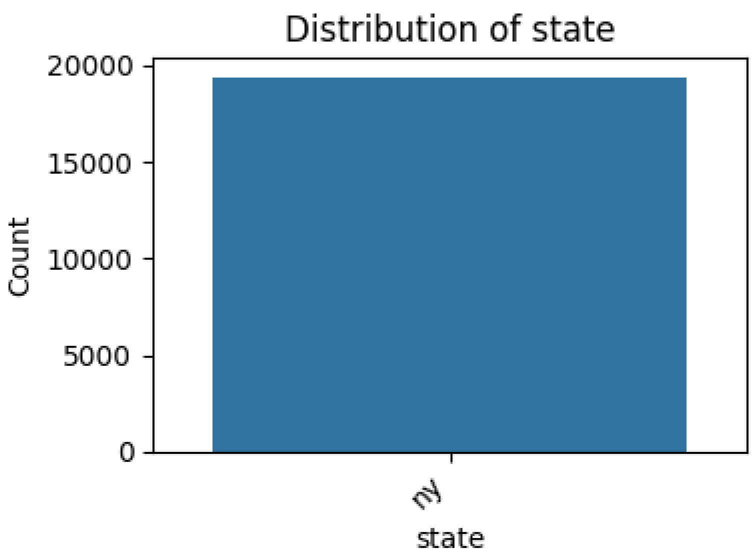
```
----
type has 13 distinct values
See 13 of them
           type  count
7         wagon    682
9           van    363
5         truck   1012
0         sedan   4063
2        pickup   2040
3         other   1170
12      offroad     22
10     mini-van    276
6     hatchback    955
4         coupe   1029
8   convertible    377
11          bus     35
1           SUV   3535
Count of the rest of the values not shown: 0
```



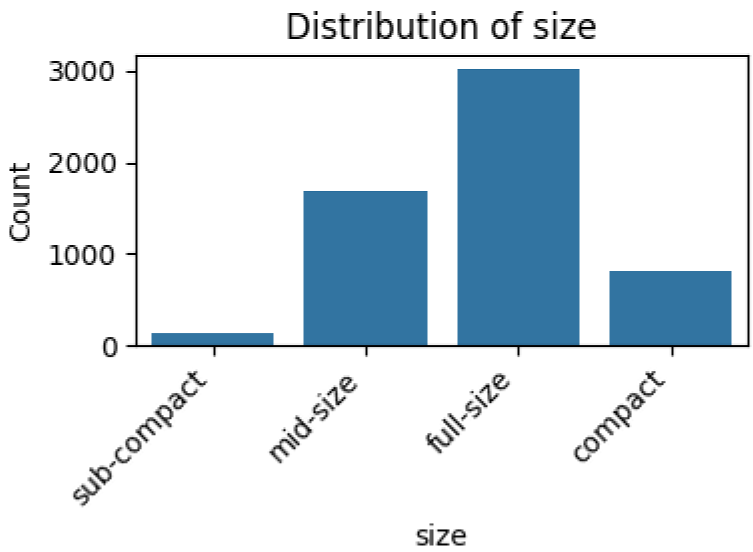

```
----
manufacturer has 39 distinct values
See 15 of them
     manufacturer  count
25          volvo    169
13     volkswagen    466
2          toyota   1375
31          tesla     53
7          subaru    713
35         saturn     31
29          rover     93
12            ram    567
33        porsche     49
28        pontiac    110
5          nissan    917
24     mitsubishi    205
26           mini    123
30        mercury     70
10  mercedes-benz    583
Count of the rest of the values not shown: 13036
```




```
----
state has 1 distinct values
See 1 of them
  state  count
0    ny  19386
```




```
----
size has 4 distinct values
See 4 of them
         size  count
3  sub-compact    141
1     mid-size   1696
0    full-size   3011
2      compact    806
```



Code block to show relationship of price to each feature column

In [9]:
```
# support functions for eval_col_avg_price()

def get_average_car_price(data, col):
    average_price_per_category = data.groupby(col)['price'].mean().reset_index()
    return average_price_per_category.sort_values(by=['price','count'], ascending=F


def get_average_and_count_car_price(data, col):
    average_price_per_category = data.groupby(col).agg(price=('price', 'mean'), cou
    return average_price_per_category.sort_values(by='price', ascending=False)


def filter_by_price_quantile(data, lower_price_q, upper_price_q):
    price_quantiles = data['price'].quantile([lower_price_q, upper_price_q])
    price_lower_bound = price_quantiles[lower_price_q]
    price_upper_bound = price_quantiles[upper_price_q]

    # Filter DataFrame
    filtered_df = data[(data['price'] >= price_lower_bound) & (data['price'] <= pri

    return filtered_df


def filter_by_price_and_count_quantile(data, lower_price_q, upper_price_q, lower_co
    price_quantiles = data['price'].quantile([lower_price_q, upper_price_q])
    price_lower_bound = price_quantiles[lower_price_q]
    price_upper_bound = price_quantiles[upper_price_q]

    # Calculate quantiles for count
    count_quantiles = data['count'].quantile([lower_count_q, upper_count_q])
    count_lower_bound = count_quantiles[lower_count_q]
    count_upper_bound = count_quantiles[upper_count_q]
```

```
    # Filter DataFrame
    filtered_df = data[(data['price'] >= price_lower_bound) & (data['price'] <= pri
                  (data['count'] >= count_lower_bound) & (data['count'] <= count_upp

    return filtered_df

def drop_outlier(data, target_col, IQR_mult):
    # Calculate Interquartile Range (IQR) for price
    Q1 = data[target_col].quantile(0.25)
    Q3 = data[target_col].quantile(0.75)
    IQR = Q3 - Q1

    # Define outlier threshold (1.5 times IQR)
    threshold = IQR_mult * IQR

    # Identify outliers (outside lower and upper bounds)
    lower_bound = Q1 - threshold
    upper_bound = Q3 + threshold
    outliers = data[(data[target_col] < lower_bound) | (data[target_col] > upper_bo
    print(f"drop_outlier(): lower bound = {lower_bound}")
    print(f"drop_outlier(): upper bound = {upper_bound}")
    # Drop outliers (consider alternative approaches if needed)
    return data.drop(outliers.index)

def handle_extreme_min_max(data_slice, column_to_get_avg, categories, subtitle, siz
        sample_avg = data_slice['price'].mean()

        # min
        min_value_index = data_slice["price"].idxmin()
        category_w_min_value = data_slice.loc[min_value_index][column_to_get_avg]
        min_value = data_slice.loc[min_value_index]["price"]
        # max
        max_value_index = data_slice["price"].idxmax()
        category_w_max_value = data_slice.loc[max_value_index][column_to_get_avg]
        max_value = data_slice.loc[max_value_index]["price"]

        show_min = min_value > size_mult*sample_avg
        show_max =  max_value < size_mult*sample_avg

        if not show_min:
            if len(subtitle)>0:
                subtitle = subtitle + '\n'
            if prefix is None:
                subtitle = f"{subtitle} MIN is not shown: {category_w_min_value} =
            else:
               subtitle = f"{subtitle} {prefix} not shown: {category_w_min_value} =

        if not show_max:
            if len(subtitle)>0:
                subtitle = subtitle + '\n'
            if prefix is None:
                subtitle = f"{subtitle} MAX is not shown: {category_w_max_value} =
            else:
                subtitle = f"{subtitle} {prefix} is not shown: {category_w_min_valu
```

```
        plt_categories = []
        for category in categories:
            ok_to_add = True
            if category == category_w_min_value:
                ok_to_add =  show_min
            if category == category_w_max_value:
                ok_to_add = show_max
            if ok_to_add:
                plt_categories.append(category)

        return plt_categories, subtitle

def show_min_max_calc(full_avg_data, sample_data_slice, column_to_get_avg, categori

        # min
        min_value_index = full_avg_data["price"].idxmin()
        category_w_min_value = full_avg_data.loc[min_value_index][column_to_get_avg
        min_value = full_avg_data.loc[min_value_index]["price"]
        min_row = full_avg_data.loc[min_value_index]
        # max
        max_value_index = full_avg_data["price"].idxmax()
        category_w_max_value = full_avg_data.loc[max_value_index][column_to_get_avg
        max_value = full_avg_data.loc[max_value_index]["price"]
        max_row = full_avg_data.loc[max_value_index]


        sample_plus_min_max = sample_data_slice.copy()
        sample_plus_min_max.loc[len(sample_plus_min_max)] = min_row
        sample_plus_min_max.loc[len(sample_plus_min_max)] = max_row

        sample_avg = sample_plus_min_max['price'].mean()


        show_min = min_value > size_mult*sample_avg
        show_max =  max_value < size_mult*sample_avg

        plt_categories = []
        if not show_min:
            if len(subtitle)>0:
                subtitle = subtitle + '\n'
            subtitle = f"{subtitle} MIN is not shown: {category_w_min_value} = {min
        else:
            plt_categories.append(category_w_min_value)

        if not show_max:
            if len(subtitle)>0:
                subtitle = subtitle + '\n'
            subtitle = f"{subtitle} MAX is not shown: {category_w_max_value} = {max
        else:
            plt_categories.append(category_w_max_value)

        for category in categories:
            ok_to_add = True
            if category == category_w_min_value:
                ok_to_add =  show_min
            if category == category_w_max_value:
```

```python
                ok_to_add = show_max
            if ok_to_add:
                plt_categories.append(category)

        return plt_categories, subtitle

def handle_many_distinct_averages(data, column_to_get_avg, value_avg_df, subtitle):

        plt_title = f'Average price and spread of min avg, max avg and sample of {m

        # too many categoires to show on a chart so get a sample of max_detail of t
        # sample 2 more than I need so I can drop ones that might put out of scale
        sample_data_slice = value_avg_df.sample(max_detail).sort_values(by='price')
        initial_len = sample_data_slice.shape[0]
        sample_data_slice = sample_data_slice[1:initial_len-1]
        sample_categories = sample_data_slice[column_to_get_avg]

        # add overall min and max but handlife if it so vastly different than sampl
        plt_categories, subtitle = show_min_max_calc(value_avg_df, sample_data_slic

        plt_data = data[data[column_to_get_avg].isin(plt_categories)]

        return plt_data, plt_title, subtitle
```

In [10]:
```python
# eval_col_avg_price() routine helps get a sense of the rawnge of values for each c
# For columns with less than max_detail(default=15) distinct values, it shows each
# For columns with more, it shows the min, max and a sample of 13 values.
# Specifically, it finds the average for each distinct column into a separate dataf
# then it samples (max_detail - 2) columnns from that set of averages
# finally it shows the box plot for the min, the samples, and the max average.
# By default it does not include outliers in the chart but passing a parameter can
# Note 1: outliers can scale the chart such that it hard to read for other boxplots
# Note 2: For the column values corresponding to the min/max average are less/more
# the boxplot for it is not shown but the average is shown in a subtitle of the plo
# This solution reduces the chance of big mis matches in scale for boxplots shown.
# If the sampling of averages hits a particularly high or low value, the chart will

def draw_box_plot(plt_data_orig, column_to_get_avg, showfliers_flag,  subtitle, plt

    plt_data = plt_data_orig.copy()
    plt_data[column_to_get_avg] = plt_data[column_to_get_avg].astype(str).str[:30].
    plt.figure(figsize=(8, 5))  # Adjust figure size as needed
    sns.boxplot(
        x = column_to_get_avg,
        y = "price",
        showmeans=True,  # Add means (optional)
        showfliers = showfliers_flag,
        data=plt_data)

    plt.xlabel(column_to_get_avg)
    plt.ylabel('Average Price')
    plt.title(subtitle)
    plt.suptitle(plt_title)
    plt.xticks(rotation=45, ha='right')  # Rotate category labels for readability
    plt.tight_layout()  # Adjust spacing between elements
```

```
    plt.show()

    plt.cla()
    plt.clf()

def draw_price_and_count_plot(plt_data_orig, column_to_get_avg, title, subtitle, ma

    # next check if need to downsample
    print("Processing Avg Price and Count Chart - May take up to 30 seconds for som
    skip_data_msg = ""
    orig_row_count = plt_data_orig.shape[0]
    if orig_row_count > 2000:
        skip_amount = plt_data_orig.shape[0] // 2000
        plt_data = plt_data_orig.iloc[::skip_amount]
        row_count_w_skip = plt_data.shape[0]
        skip_data_msg = f"\nLarge # of values({orig_row_count:,.0f}), charting ever
    else:
        plt_data = plt_data_orig
    duplicates = plt_data[column_to_get_avg].duplicated()
    has_duplicates = plt_data[column_to_get_avg].duplicated().any()
    if has_duplicates:
        print("has duplicates")
    else:
        print("no duplicates")

    # Create the plot
    fig, ax1 = plt.subplots(figsize=(5, 3))

    #Line plot for value1 (left y-axis)
    if plt_data.shape[0] <max_detail:
            ax1.plot(
                plt_data[column_to_get_avg].astype(str).str[:30].replace('$', ' ',
                plt_data["price"], label='Average Price', marker= 'o')
    else:
        if len(skip_data_msg)>0:
             ax1.plot(
                plt_data[column_to_get_avg],
                #plt_data[column_to_get_avg],
                plt_data["price"], label='Average Price')
        else:
                ax1.plot(
                plt_data[column_to_get_avg].astype(str).str[:30].replace('$', ' ',
                #plt_data[column_to_get_avg],
                plt_data["price"], label='Average Price')

    ax1.set_ylabel('Average Price', color='b')
    ax1.tick_params(axis='y', labelcolor='b')


    if plt_data.shape[0] <= max_detail:
        ax1.tick_params(axis='x', rotation=45)  # Rotate x-axis labels
    else:
        plt.xticks([])

    # Bar chart for value2 (right y-axis)
    ax2 = ax1.twinx()  # Create a twin axes for value2
```

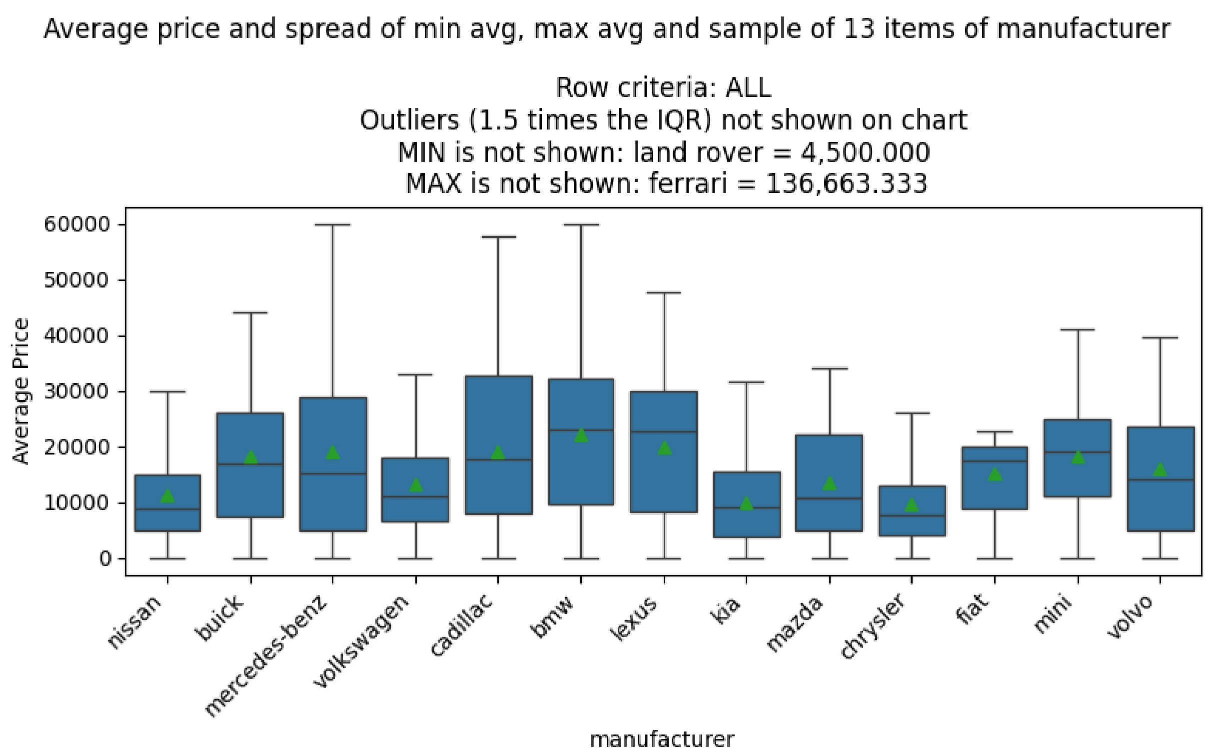
```
    ax2.bar(plt_data[column_to_get_avg].astype(str).str[:30].replace('$', ' ', rege
    ax2.set_ylabel('Count', color='g')
    ax2.tick_params(axis='y', labelcolor='g')

    # Customize the plot
    plt.title(f"{subtitle}" + skip_data_msg )
    plt.suptitle(title)
    plt.xlabel(column_to_get_avg)


    if plt_data.shape[0] <= max_detail:
        ax2.tick_params(axis='x', rotation=45)  # Rotate x-axis labels
    else:
        plt.xticks([])

    lines1, labels1 = ax1.get_legend_handles_labels()
    lines2, labels2 = ax2.get_legend_handles_labels()
    plt.legend(lines1 + lines2, labels1 + labels2, loc='upper center')
    plt.tight_layout()

    # show
    plt.show()

    # release shared mem related to plotting
    plt.cla()
    plt.clf()

def set_base_box_subtitle(showfliers_flag,
                        lower_price_q, upper_price_q, lower_count_q, upper_count_q)

    if lower_price_q == 0 and upper_price_q ==1 and lower_count_q == 0 and upper_co
        box_subtitle = "Row criteria: ALL"
    else:
        box_subtitle = f"Price criteria: start % = {lower_price_q:.2%}, end % = {up
        box_subtitle = box_subtitle + f"\nCount criteria: start % = {lower_count_q:
    if showfliers_flag:
        box_subtitle = box_subtitle + "\nOutliers (1.5 times the IQR) shown"
    else:
        box_subtitle = box_subtitle + "\nOutliers (1.5 times the IQR) not shown on
    return box_subtitle

def get_quantile(data, column_name, value, data_max, data_quantiles):

  if value < data_quantiles[0.25]:
    return 'Zero to 25th quantile'
  elif value < data_quantiles[0.5]:
    return 'Above 25th to 50th quantile'
  elif value < data_quantiles[0.75]:
    return 'Above 50th to 75th quantile'
  else:
    return 'Above 75th to 100th quantile'

def eval_col_avg_price(data, col_list, max_detail = 15, showfliers_flag = False,
                        lower_price_q = 0, upper_price_q = 1, lower_count_q = 0, up
    for column_to_get_avg in col_list:
        base_box_subtitle = set_base_box_subtitle(showfliers_flag,
```

```
                lower_price_q, upper_price_q, lower_count_q, upper_count_q)
# aggregate primary data to get avg price and count

value_avg_df_orig = get_average_and_count_car_price(data, column_to_get_avg

# apply percentile criteria to value averages.
# goal is to filter by quantiles so to reduce the number of categories incl
# goal is NOT to limit/filter the core price data set by quantile for the c
value_avg_df = filter_by_price_and_count_quantile(value_avg_df_orig,
                lower_price_q, upper_price_q, lower_count_q, upper_count_q)
# Saving to database to manually reivew for bad data set in excel
#value_avg_df.to_csv(f'saved_output/{column_to_get_avg}_avg_prices.csv')

# share some basic info about the column
unq_count = value_avg_df.shape[0]
print(f"{column_to_get_avg} has {unq_count} distinct values")
disp_count = min(max_detail,unq_count)
print(f"See {disp_count} of them")
top_of_value_avg_df = value_avg_df.head(disp_count)
top_of_value_avg_df = top_of_value_avg_df.copy()
top_of_value_avg_df['price'] = top_of_value_avg_df['price'].apply(lambda x:
print(top_of_value_avg_df)

# prepare to plot the box plot
if disp_count != 0:
    if unq_count < max_detail:
        box_plt_title = f'Average price and spread of {column_to_get_avg}'
        categories = value_avg_df[column_to_get_avg]
        plt_categories, box_subtitle = handle_extreme_min_max(value_avg_df,
        print("Will display all categories:")
        print(plt_categories)
        plt_data = data[data[column_to_get_avg].isin(plt_categories)]
    else:
        plt_data, box_plt_title, box_subtitle = handle_many_distinct_averag


    # draw boxplot of price for category
    draw_box_plot(plt_data, column_to_get_avg, showfliers_flag,  box_subtit

    # prepare to plot both average price and count per category value on sa
    # first get a version without outlier averages
    IQR_mult = 1.5
    data_no_outlier = drop_outlier(value_avg_df, "price", IQR_mult)
    print(f"Potential outliers for {column_to_get_avg} = {value_avg_df.shap

    # set title
    multi_title = f"Plot of Avg Price and Count for {column_to_get_avg}"
    multi_subtitle = base_box_subtitle
    draw_price_and_count_plot(data_no_outlier, column_to_get_avg, multi_tit
else:
    print(f"****No values of {column_to_get_avg} are in the given critera s

print("----")
```

Charts relating price to feature columns

In [11]:
```
col_list = ['manufacturer','condition','fuel',
            'state','year']

eval_col_avg_price(car_df, col_list,
                   lower_price_q = 0.0, upper_price_q = 1, lower_count_q = 0.0, up
```

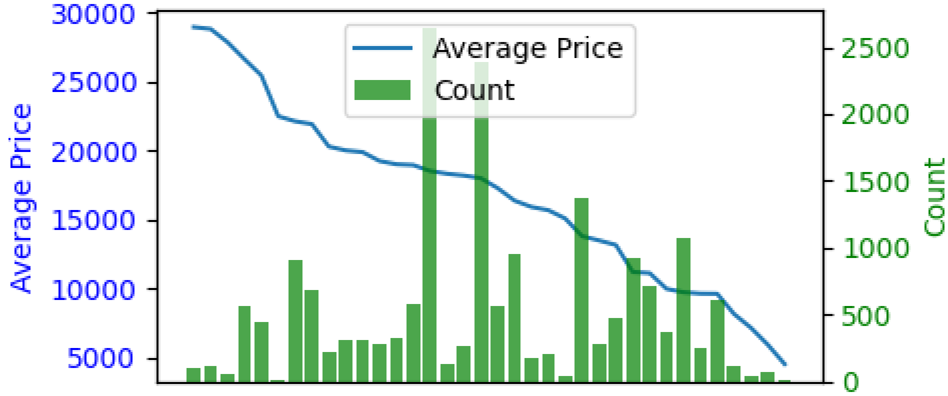
```
manufacturer has 39 distinct values
See 15 of them
       manufacturer     price  count
9           ferrari  $136,663      3
30          porsche   $45,438     49
35            tesla   $40,906     53
32            rover   $28,968     93
17           jaguar   $28,834    116
1        alfa-romeo   $27,871     51
31              ram   $26,646    567
2              audi   $25,468    447
13  harley-davidson   $22,500      1
3               bmw   $22,134    913
12              gmc   $21,928    674
22          lincoln   $20,297    212
21            lexus   $20,016    303
0             acura   $19,894    311
16         infiniti   $19,250    278
```




```
drop_outlier(): lower bound = -2615.979783593966
drop_outlier(): upper bound = 36818.59559569029
Potential outliers for manufacturer = 3
Processing Avg Price and Count Chart - May take up to 30 seconds for some charts
no duplicates
<Figure size 640x480 with 0 Axes>
```

Plot of Avg Price and Count for manufacturer

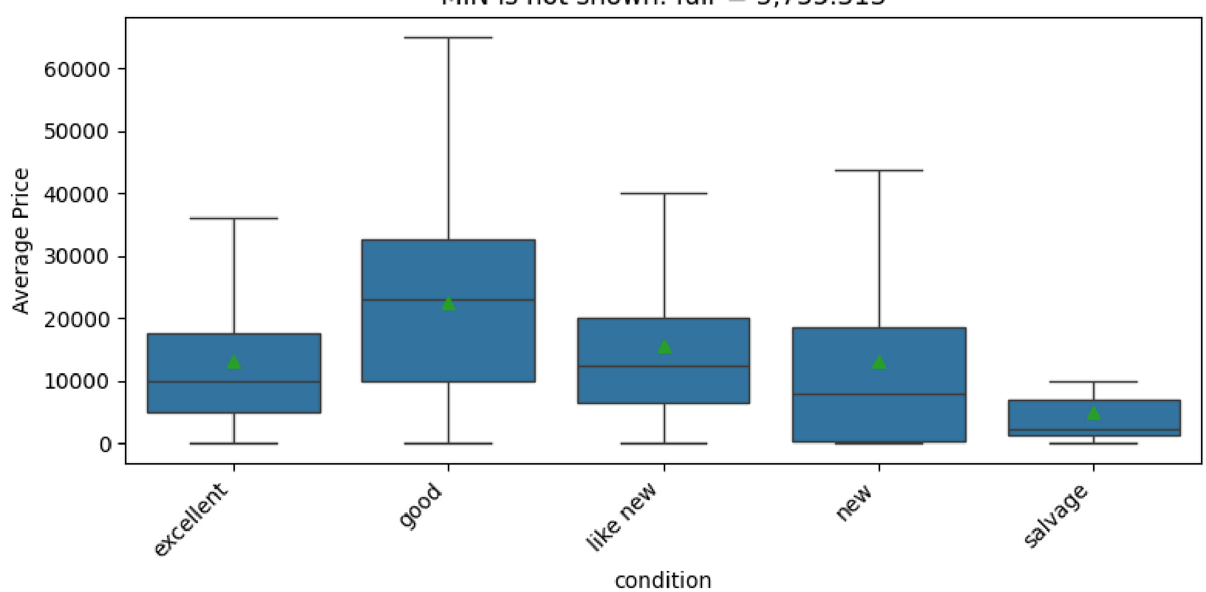
Row criteria: ALL
Outliers (1.5 times the IQR) not shown on chart



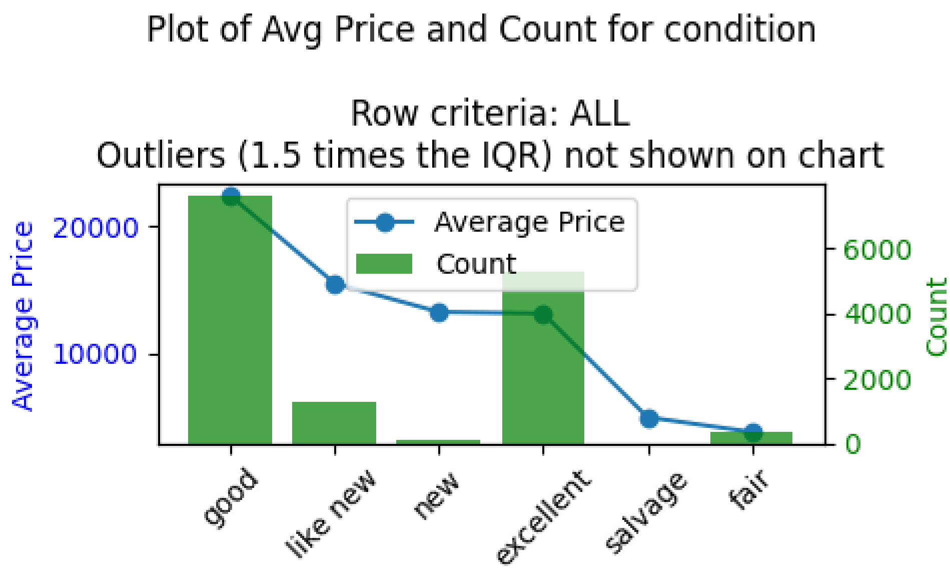

```
----
condition has 6 distinct values
See 6 of them
   condition    price  count
2       good  $22,443   7612
3   like new  $15,519   1267
4        new  $13,249    108
0  excellent  $13,128   5283
5    salvage   $4,926     17
1       fair   $3,755    364
Will display all categories:
['good', 'like new', 'new', 'excellent', 'salvage']
<Figure size 640x480 with 0 Axes>
```

Average price and spread of condition

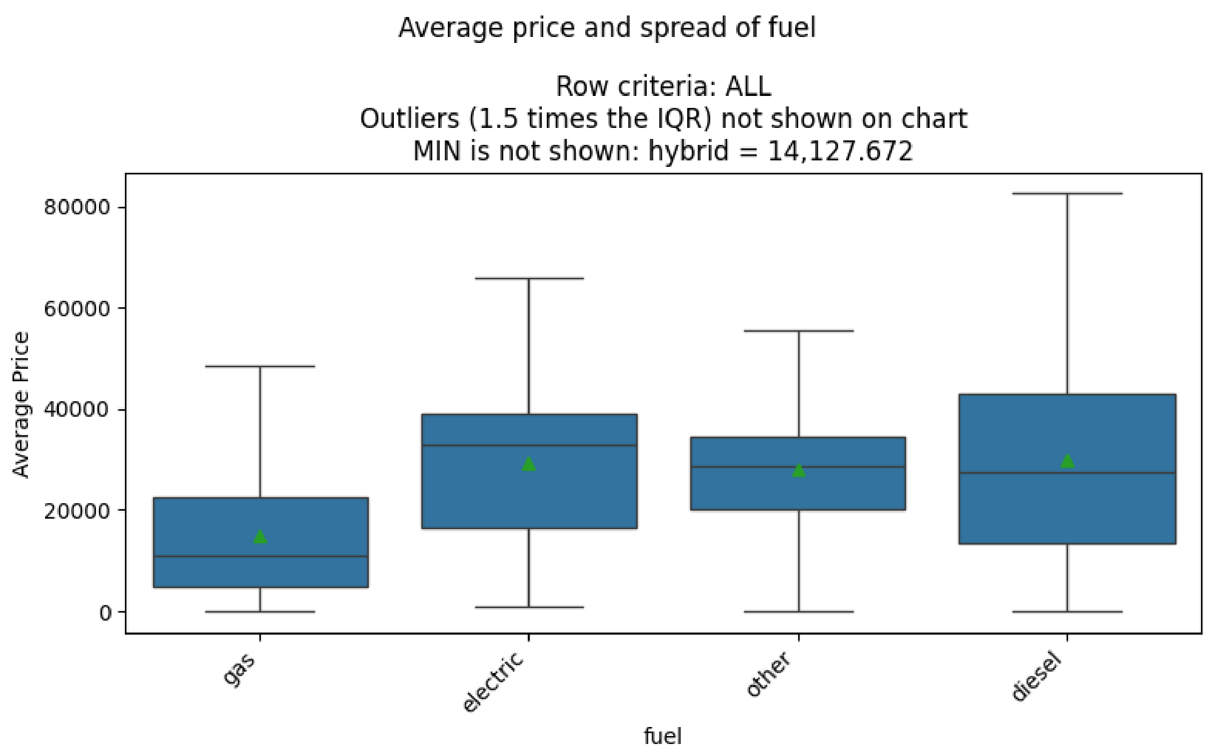
Row criteria: ALL
Outliers (1.5 times the IQR) not shown on chart
MIN is not shown: fair = 3,755.313



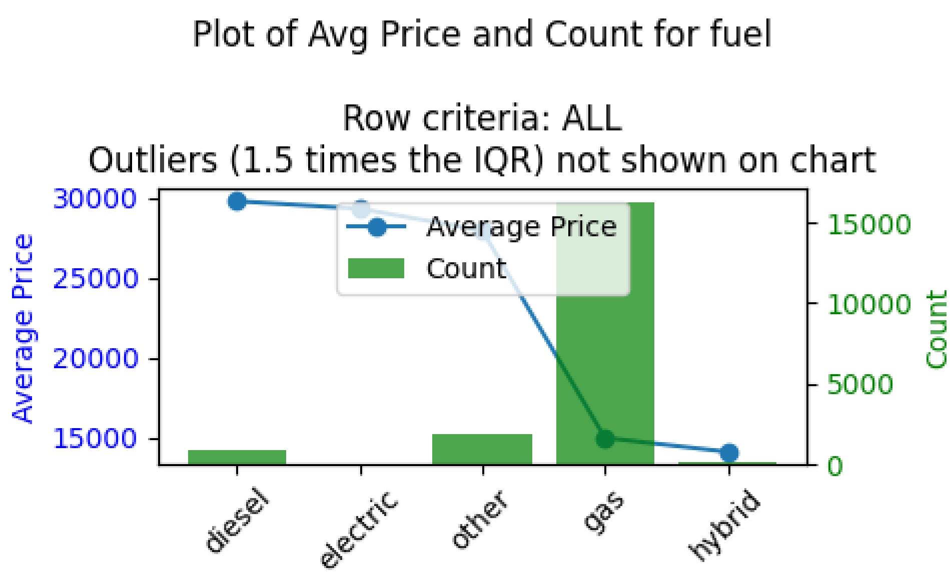
```
drop_outlier(): lower bound = -4985.611862240961
drop_outlier(): upper bound = 26914.062972236563
Potential outliers for condition = 0
Processing Avg Price and Count Chart - May take up to 30 seconds for some charts
no duplicates
<Figure size 640x480 with 0 Axes>
```



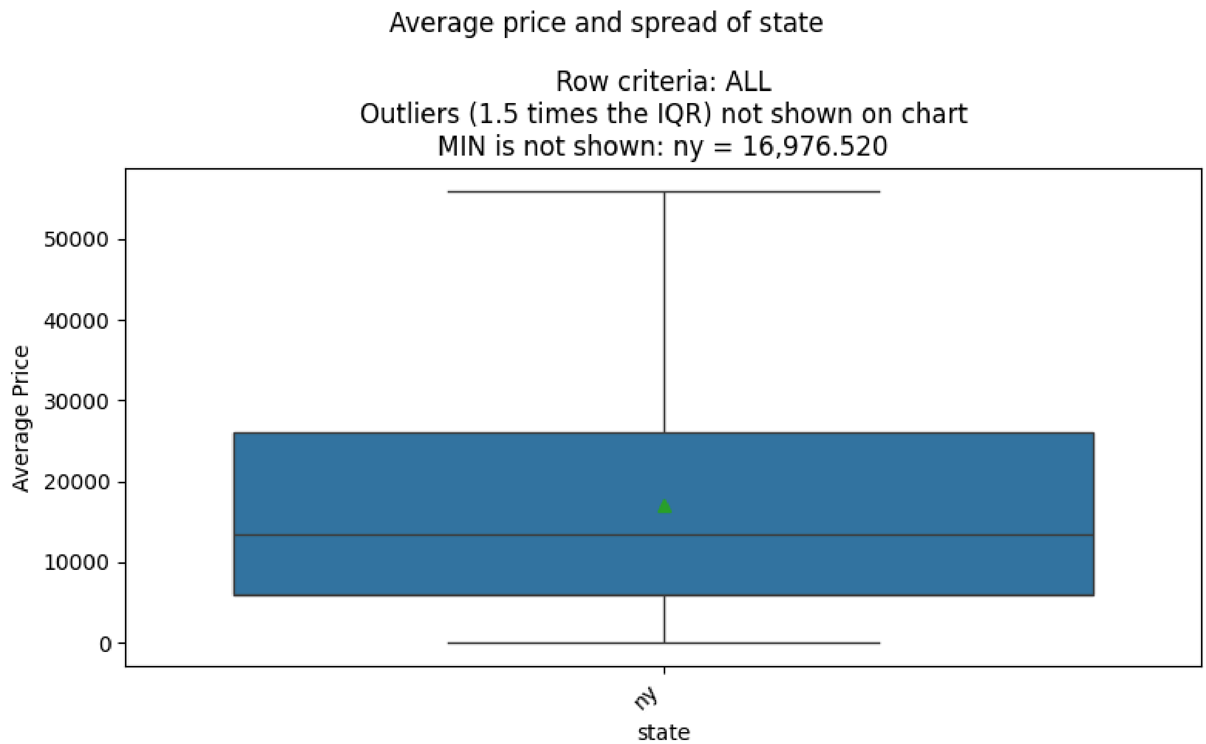

```
----
fuel has 5 distinct values
See 5 of them
       fuel    price  count
0    diesel  $29,795    870
1  electric  $29,322     74
4     other  $27,963   1892
2       gas  $15,005  16288
3    hybrid  $14,128    186
Will display all categories:
['diesel', 'electric', 'other', 'gas']
<Figure size 640x480 with 0 Axes>
```



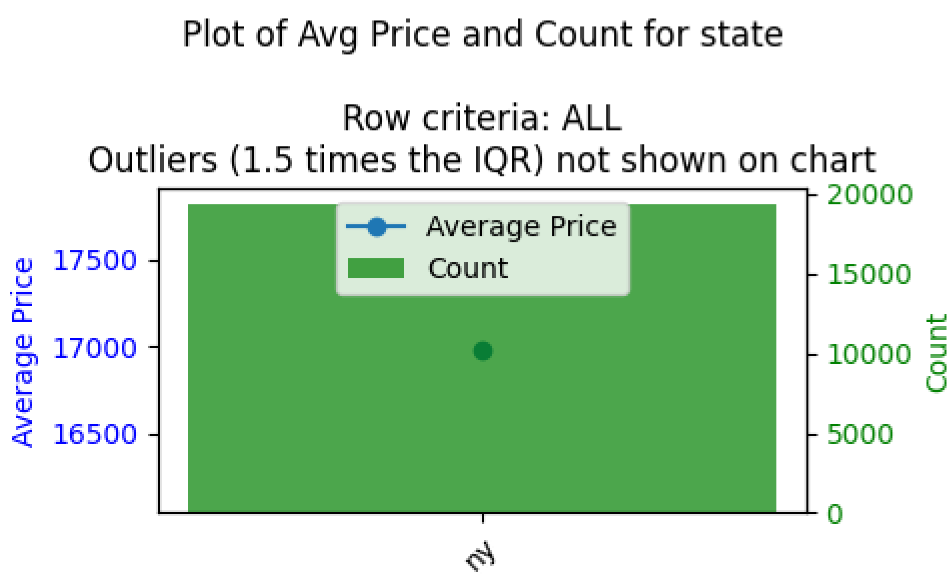

```
drop_outlier(): lower bound = -6470.167137637389
drop_outlier(): upper bound = 50798.07866096081
Potential outliers for fuel = 0
Processing Avg Price and Count Chart - May take up to 30 seconds for some charts
no duplicates
<Figure size 640x480 with 0 Axes>
```



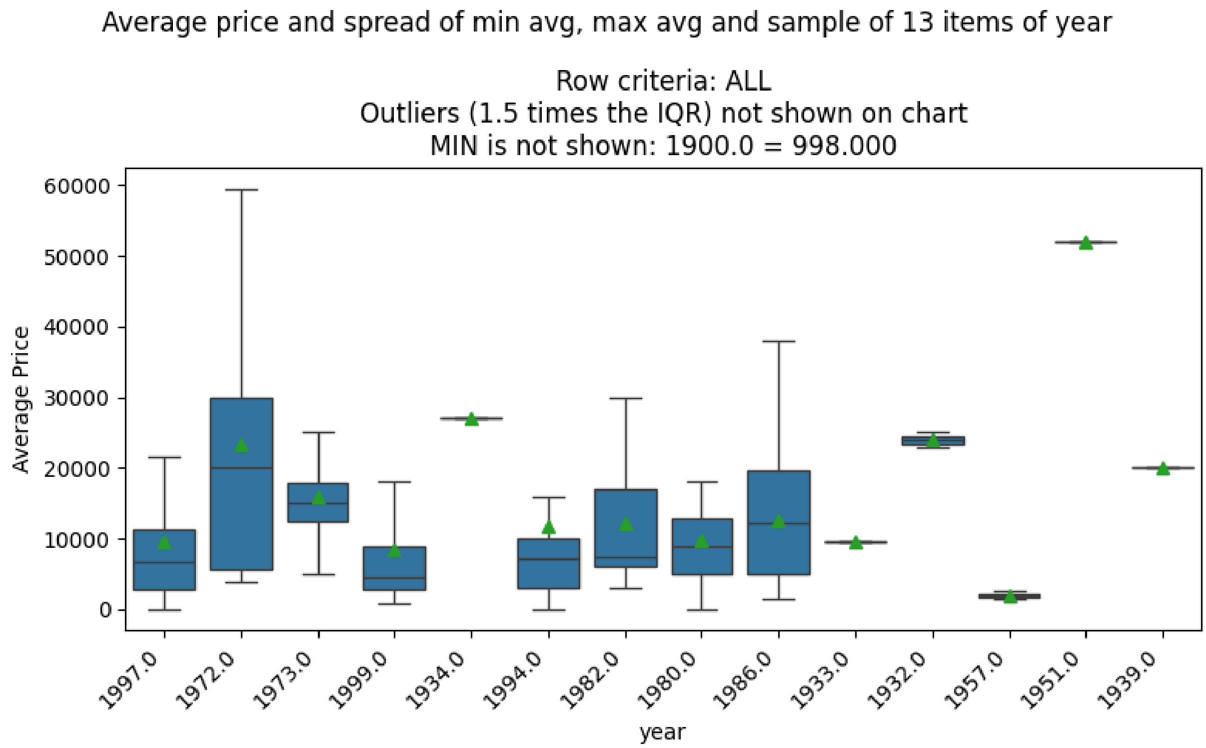

```
----
state has 1 distinct values
See 1 of them
  state    price  count
0    ny  $16,977  19386
Will display all categories:
['ny']
<Figure size 640x480 with 0 Axes>
```



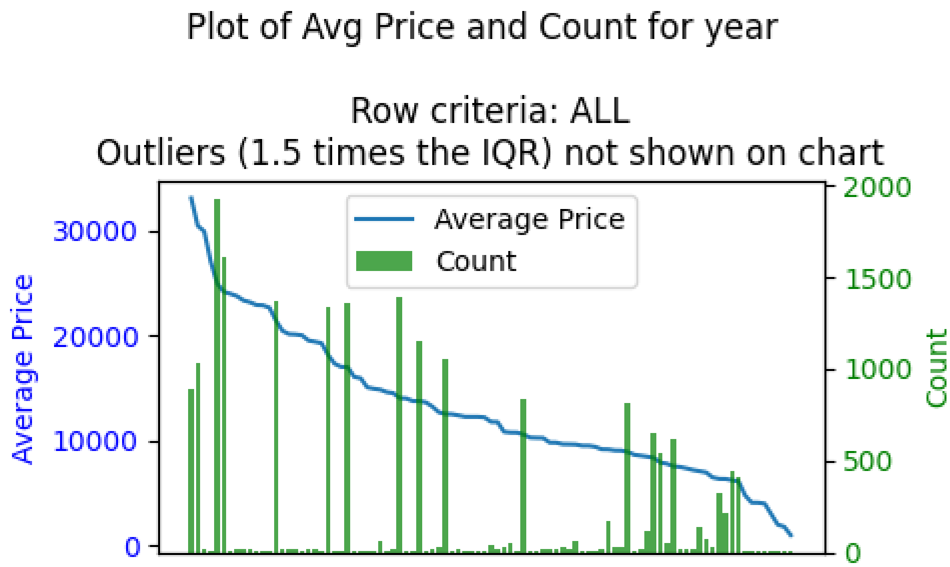

```
drop_outlier(): lower bound = 16976.520117610646
drop_outlier(): upper bound = 16976.520117610646
Potential outliers for state = 0
Processing Avg Price and Count Chart - May take up to 30 seconds for some charts
no duplicates
<Figure size 640x480 with 0 Axes>
```



```
----
year has 96 distinct values
See 15 of them
      year    price  count
24  1951.0  $52,000      1
42  1969.0  $42,836     31
5   1928.0  $38,833      3
93  2020.0  $33,041    894
92  2019.0  $30,415   1034
21  1948.0  $29,877     14
11  1934.0  $27,000      2
91  2018.0  $24,908   1927
90  2017.0  $24,128   1613
9   1932.0  $23,950      2
39  1966.0  $23,730     13
45  1972.0  $23,321     13
38  1965.0  $23,150     13
8   1931.0  $22,883      6
28  1955.0  $22,850      6
<Figure size 640x480 with 0 Axes>
```




```
drop_outlier(): lower bound = -6068.637142590422
drop_outlier(): upper bound = 33096.82380907927
Potential outliers for year = 3
Processing Avg Price and Count Chart - May take up to 30 seconds for some charts
no duplicates
<Figure size 640x480 with 0 Axes>
```




```
----
<Figure size 640x480 with 0 Axes>
```

Observations from chart analysis

Task: Visually review the data distribution and range of values of the data. Look for obvious patterns using histograms and box plots.

The following features look to have interesting combination of high volume of sales and at a high price:

- feature: cylinders, value = 6
- feature: title_status, value = lien
- feature: drive, value = 4wd
- feature: type, value = SUV
- feature: paint_color, values: (Black, White)
- feature: manufacturer, value: will drill down but one manufacturer has a relatively high avg price and very high volume

NOTE: There are other more obvious relationship of high volume and price combinations not mentioned (like condition:excellent)

- There are many price-volume variations for features state, manufacturer, model and year. However, there are so many distinct values of each of these features that it is hard to visualize pattern.
- So I have built into the visualizations a way to slice the data by quantiles of price and sales volume(count) and look at those charts. Will do so after clean-up.

Task: Note if there are major imbalances in the category groupings of the data.

The most non-obvious category data imbalances are:

- condition: fair has very low volume compared to good and excellent. I expected 'fair' to be lower but it is several times lower volume
- paint_color: Both black and white are significantly more popular than other colors.

NOTE: Will redo these charts after the outlier and null data clean-up

Data Preparation

After our initial exploration and fine tuning of the business understanding, it is time to construct our final dataset prior to modeling. Here, we want to make sure to handle any integrity issues and cleaning, the engineering of new features, any transformations that we believe should happen (scaling, logarithms, normalization, etc.), and general preparation for modeling with `sklearn`.

Remove VIN feature

- VIN is a unique number per vehicle (per row)
- We will drop this row from our modeling analysis because it will not have general predictive power

Data with nulls

In [12]:
```
print(f"Before Clean-up: Total cells in dataframe with NULLs = { car_df.isnull().su
```

```
Before Clean-up: Total cells in dataframe with NULLs = 52110
```

In [13]:
```
print(f"Before Clean-up: Total rows with one or more NULLs  = {car_df.isnull().any(
```

```
Before Clean-up: Total rows with one or more NULLs  = 17651
```

Code to identify trade off of null data and number of features

In [14]:
```
def nan_count_in_a_col(data, col):
    return data[col].isnull().sum()

def nan_count_by_col(data):
    nans_df = pd.DataFrame(columns=['src_col', 'nan_count'])
    for col in data.columns:
        nan_count = nan_count_in_a_col(data, col)
        new_row = {'src_col': col, 'nan_count':nan_count}
        nans_df.loc[len(nans_df)] = new_row
    nans_df = nans_df.sort_values(by='nan_count', ascending = False)

    return nans_df
```

In [ ]:

In [15]:
```
def find_feat_to_max_non_null_rows(car_df, start_count, end_count,
                                   col_list = ['region', 'manufactur
```

```
                                                    'title_status', 'transmission','d
                                                    use_previous_run = True):
    # Uses itertools.combinations() to search for the best combinations of features
    # It finds combinations of length start_count to end_count
    # Returns two lists: max_non_null_count, max_non_null_combo
    # NOTE: The run time for this function is over an hour when used with start_cou
    # Therefore, I have a hard-coded list for that size and a flag to use that cach
    if use_previous_run == True:
        final_list_counts_from_previous_run = [426880, 426880, 425675, 423187, 4210
        final_list_features_from_previous_run = [['region'], ['region', 'state'], [
                                                ['region', 'fuel', 'transmission',
                                                ['region', 'model', 'fuel', 'trans

                                                ['region', 'manufacturer', 'model'
                                                ['region', 'manufacturer', 'model'
                                                ['region', 'manufacturer', 'model'
                                                ['region', 'manufacturer', 'model'
                                                ['region', 'manufacturer', 'model'
                                                 'paint_color', 'state', 'odometer

                                                ['region', 'manufacturer', 'model'
                                                 'type', 'paint_color', 'state', '
        return final_list_counts_from_previous_run,final_list_features_from_previou
    else:
        # longer run-time branch so use print() statements to keep user informed
        cur_non_null_row_count = []
        max_non_null_count = []
        max_non_null_combo  = []
        col_list = ['region', 'manufacturer', 'model', 'condition', 'cylinders','fu
                    'title_status', 'transmission','drive','size','type','paint_col
        total_rows = car_df.shape[0]
        start_count = 0
        end_count = 15
        cur_non_null_row_count = [None]*(end_count)
        max_non_null_count = [None]*(end_count)
        max_non_null_combo = [None]*(end_count)
        for col_len in range(start_count,end_count):
                print("new outer loop")
                print(col_len)
                #cur_non_null_row_count.append(None)
                #max_non_null_count.append(None)
                #max_non_null_combo.append(None)
                combinations = list(itertools.combinations(col_list, col_len+1))
                print(f"Combos to process: {len(combinations)}")
                start_time = time.time()
                for combo in combinations:
                    combo_list = list(combo)
                    #print(f"new combo: {combo_list}")
                    cur_count_of_rows_wth_nulls = car_df[car_df[combo_list].isnull(
                    cur_non_null_row_count[col_len] = total_rows - cur_count_of_row
                    if max_non_null_count[col_len] is None or cur_non_null_row_coun
                        max_non_null_count[col_len] = cur_non_null_row_count[col_le
                        max_non_null_combo[col_len] = combo_list
                end_time = time.time()
                elapsed_time = end_time - start_time
                print("Elapsed time:", elapsed_time, "seconds")
```

```
                print("---")

        print(max_non_null_count)
        print(max_non_null_combo)
        return max_non_null_count, max_non_null_combo

#rows_with_nulls = car_df[combo][car_df[combo].isnull().any(axis=1)].reset_index()
#rows_with_nulls.rename(columns={'index': 'src_index'}, inplace=True)
```

In [16]:
```
max_non_null_count, max_non_null_combo = find_feat_to_max_non_null_rows(car_df,0,15
```

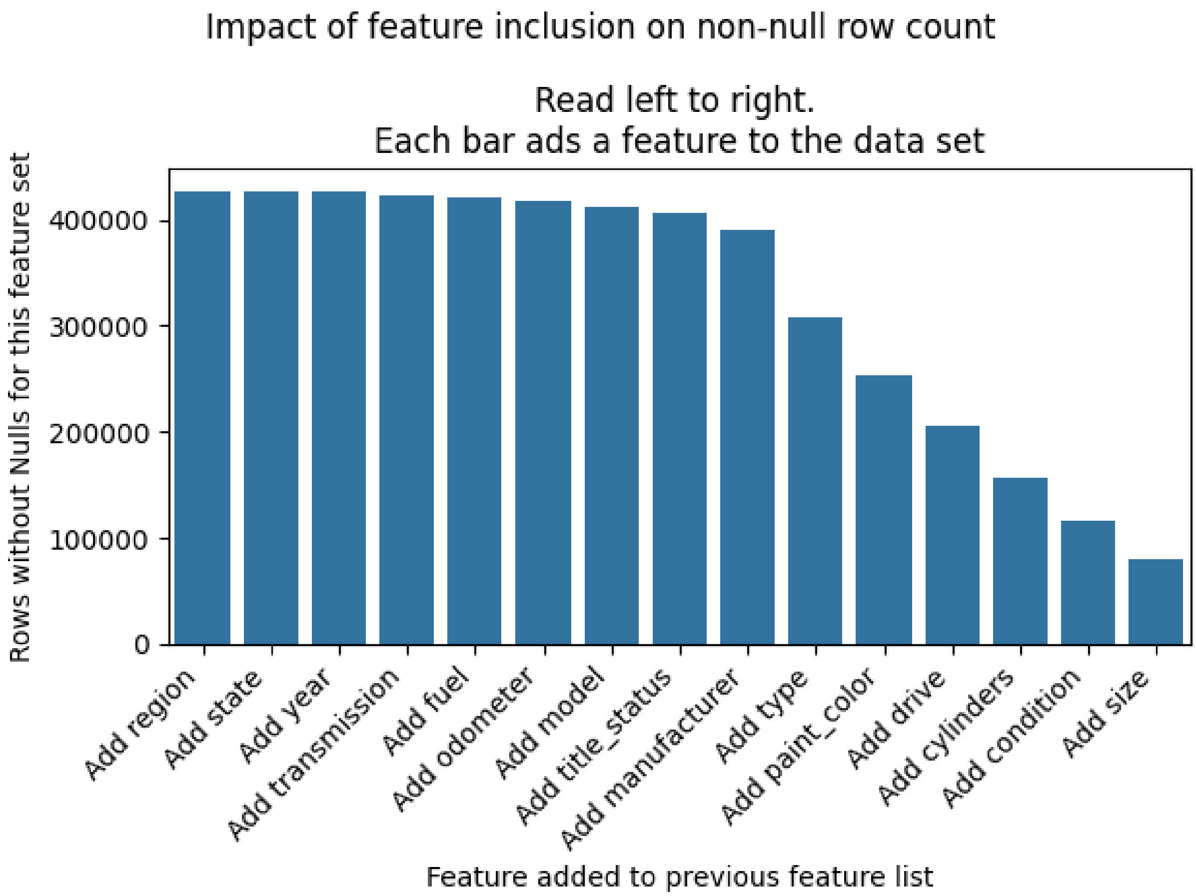
In [17]:
```
def chart_features_vs_non_null_rows(max_non_null_count, max_non_null_combo):
    final_list = zip(max_non_null_count, max_non_null_combo)
    feature_choices = pd.DataFrame(final_list, columns = ['non_null_row_count', 'fe
    f_c = feature_choices[feature_choices['features'].notnull()]
    f_c = f_c.copy()
    f_c['p_features'] = f_c['features'].shift(periods=1,fill_value = ['no features'
    f_c['feature_change'] = f_c.apply(lambda row: list(set(row['features']) - set(r
    f_c['feature_change_desc'] = f_c.apply(lambda row: "Add " +  " , ".join(row['fe
    f_c
    sns.barplot(x='feature_change_desc', y='non_null_row_count', data=f_c)
    plt.xlabel('Feature added to previous feature list')
    plt.ylabel('Rows without Nulls for this feature set')
    plt.suptitle('Impact of feature inclusion on non-null row count')
    plt.title('Read left to right. \nEach bar ads a feature to the data set')
    plt.xticks(rotation=45, ha='right')  # Rotate category labels for readability
    plt.tight_layout()  # Adjust spacing between elements
    plt.show()
    plt.cla()
    plt.clf()
```

Charts to identify trade off of null data and number of features

In [18]:
```
print(nan_count_by_col(car_df))
```

```
         src_col  nan_count
14          size      13732
7      cylinders       8443
12           VIN       8220
13         drive       6774
16    paint_color       5018
6      condition       4735
15          type       3827
4    manufacturer        826
5          model        222
11   transmission         99
10   title_status         79
8           fuel         76
9       odometer         47
3           year         12
0             id          0
1         region          0
2          price          0
17         state          0
```

In [19]:
```
chart_features_vs_non_null_rows(max_non_null_count, max_non_null_combo)
```




```
<Figure size 640x480 with 0 Axes>
```

Strategy to handle null data

The above chart (Imapct of feature inclusion...) is a useful tool I created to make practical decisions about outliers. If there is value in including all the above features, then our total data set shrinks from over 400K rows to under 100K rows. While 100K rows is significant, losing 300K+ rows of information could easily degrade predictive capabilities. This chart helps evaluate the combinations that manage this trade-off.

We won't know until we run actual modeling how valuable a given feature is. However, looking at the price-volume charts in the previous section we can estimate the potential of **size, condition, cylinders, drive, paint color and type**. Each of these columns significantly reduce the number of non-null rows.

- size has good price and volumne variation but the most null values
- drive, fuel, cylinders, and condition do not have strong variation in BOTH price and volume and they significantly reduce rows available for training and testing.
- type and paint color are attractive to keep because they do have strong price and volume variation while reducing the number of available rows less significantly than others.

Thus, our best initial estimates of features to explore further are including up to paint_color in our main data set. Making for a feature list as follows:

- region, state, year, transmission, fuel, odometer, model, title_status, manufacturer, type, paint_color
- This will give us rows of data 252,977 - a loss of 40% of the rows available
- Note this won't be a good choice if there is significant collinearity of paint_color and type with region, state, year, transmission, fuel, odometer, model, title_status, or manufacturer. This will show up when we do linear regression. Therefore, we will start with this column list and revisit as needed

NOTE: Time permitting and as needed, we will also repeat the modeling assuming we have all features (79,195 rows) and also assuming we have only up to manufacturer (drop type and paint color for total of 389,604 rows)

Code to remove outliers

In [20]:
```
balanced_col = ['region', 'manufacturer', 'model', 'fuel', 'title_status', 'transmi
print(f"original dataframe row count = {car_df.shape[0]}")
balanced_col_keep = balanced_col.copy()
balanced_col_keep.append('id')
balanced_col_keep.append('price')
car_df_no_nulls_balanced = car_df[balanced_col_keep][car_df[balanced_col].notnull()
print(f"balanced dataframe row count = {car_df_no_nulls_balanced.shape[0]}")
print(car_df_no_nulls_balanced.columns)

# for later try more rows and less features
more_rows_col = ['region', 'manufacturer', 'model', 'fuel', 'title_status', 'transm
more_rows_col_keep = more_rows_col.copy()
more_rows_col_keep.append('id')
more_rows_col_keep.append('price')

car_df_no_nulls_more_rows = car_df[more_rows_col_keep][car_df[more_rows_col].notnul
print(f"'more rows' dataframe row count = {car_df_no_nulls_more_rows.shape[0]}")

# Try these if time or if above doesn't perform well
# don't use 'size' in any case. probably overlap with type and cylinder and cuts to
more_feat_col =   ['region', 'manufacturer', 'model', 'condition', 'cylinders', 'fu
                                              'type', 'paint_color', 'state', '
more_feat_col_keep = more_feat_col.copy()
more_feat_col_keep.append('id')
more_feat_col_keep.append('price')

car_df_no_nulls_more_feat = car_df[more_feat_col_keep][car_df[more_feat_col].notnul
print(f"'more features' dataframe row count = {car_df_no_nulls_more_feat.shape[0]}"
```

```
original dataframe row count = 19386
balanced dataframe row count = 12753
Index(['region', 'manufacturer', 'model', 'fuel', 'title_status',
       'transmission', 'type', 'paint_color', 'state', 'odometer', 'year',
       'id', 'price'],
      dtype='object')
'more rows' dataframe row count = 18197
'more features' dataframe row count = 6506
```

In [21]:

```
print("^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^")
print("^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^")
print("^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^")
print("^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^")
print("Choice of null data strategy")
print("I choose 'balanced'")
cars_no_nulls = car_df_no_nulls_balanced
#print("I choose 'more_feat'")
#cars_no_nulls = car_df_no_nulls_more_feat
print("^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^")
print("^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^")
print("^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^")
print("^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^")
print(cars_no_nulls.info())

print("cars_no_nulls row count:")
print(cars_no_nulls.shape[0])
```

```
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
Choice of null data strategy
I choose 'balanced'
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
<class 'pandas.core.frame.DataFrame'>
Index: 12753 entries, 263504 to 282886
Data columns (total 13 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   region        12753 non-null  object
 1   manufacturer  12753 non-null  object
 2   model         12753 non-null  object
 3   fuel          12753 non-null  object
 4   title_status  12753 non-null  object
 5   transmission  12753 non-null  object
 6   type          12753 non-null  object
 7   paint_color   12753 non-null  object
 8   state         12753 non-null  object
 9   odometer      12753 non-null  float64
 10  year          12753 non-null  float64
 11  id            12753 non-null  int64
 12  price         12753 non-null  int64
dtypes: float64(2), int64(2), object(9)
memory usage: 1.4+ MB
None
cars_no_nulls row count:
12753
```

Outliers

Typical approach is to look at 1.5 times the IQR

- For this project we will model twice: once using 1.5 X IQR and again using 3.0 X IQR
- Important to understand if there are traunches or clusters of outliers. This could be legitimate data when data gets segregated by a particular feature combination
- For example, Ferraris cost far more than Mercuries. The Ferrari price might seem like an outlier but compared to other luxury manufacturers it will be legitimate data

For category fields with a manageable range of distinct values we will try to manually review all outliers and decide based on judgment

Lastly, an outlier may be a bad data point or it may be a datapoint with a typo or other recognizable mistake that if corrected would no longer be an outlier.

- We will try to identify these situations

Code to analyze outlier removal process

In [22]:
```
def plot_outliers_vs_orig(orig, no_nulls, outlier_level1_removed, outlier_level2_re
  plt.figure(figsize=(5, 3))

  # assumption we would never plot 10 million points
  cur_min = 1000000
  if samp_size == 'All' and ids_to_use is None:
      if 1 in plots:
          cur_min = min(orig.shape[0], cur_min)
      if 2 in plots:
          cur_min = min(no_nulls.shape[0], cur_min)
      if 3 in plots:
          cur_min = min(outlier_level1_removed.shape[0], cur_min)
      if 4 in plots:
          cur_min = min(outlier_level2_removed.shape[0], cur_min)
      plt_samp_size = cur_min
  else:
      if ids_to_use is None:
          plt_samp_size = samp_size
      else:
          plt_samp_size = ids_to_use.shape[0]

  if ids_to_use is None:
      plt_id = orig.sample(plt_samp_size)
  else:
      plt_id = ids_to_use

  plt_orig = pd.merge(plt_id["id"], orig, on='id', how='left')
  plt_no_nulls = pd.merge(plt_id["id"], no_nulls, on='id', how='left')
  plt_outlier_level1_removed = pd.merge(plt_id["id"], outlier_level1_removed, on='i
  plt_outlier_level2_removed = pd.merge(plt_id["id"], outlier_level2_removed, on='i

  plt_orig = plt_orig.sort_values(by="id")
  plt_no_nulls = plt_no_nulls.sort_values(by="id")
  plt_outlier_level1_removed = plt_outlier_level1_removed.sort_values(by="id")
  plt_outlier_level2_removed = plt_outlier_level2_removed.sort_values(by="id")

  # adjust values so less overlap
  adj_range = 0
  plt_orig_a = plt_orig.copy()
  plt_orig_a['price'] = plt_orig_a['price'] + random.randint(-adj_range, adj_range)
  plt_no_nulls_a = plt_no_nulls.copy()
  plt_no_nulls_a['price'] = plt_no_nulls_a['price'] + random.randint(-adj_range, ad
  plt_outlier_level1_removed_a = plt_outlier_level1_removed.copy()
  plt_outlier_level1_removed_a['price'] = plt_outlier_level1_removed_a['price'] + r
  plt_outlier_level2_removed_a = plt_outlier_level2_removed.copy()
  plt_outlier_level2_removed_a['price'] = plt_outlier_level2_removed_a['price'] + r

  # Plot each DataFrame with a different color
  if 1 in plots:
      plt.plot(plt_orig_a['id'], plt_orig_a['price'], label='orig', color='black')
  if 2 in plots:
      plt.plot(plt_no_nulls_a['id'], plt_no_nulls_a['price'], label='Before outlier
  if 3 in plots:
      plt.plot(plt_outlier_level1_removed_a['id'], plt_outlier_level1_removed_a['pr
  if 4 in plots:
```

```
    plt.plot(plt_outlier_level2_removed_a['id'], plt_outlier_level2_removed_a['pr


# Add labels and title
plt.xlabel('ID')
plt.ylabel('Price')
plt.title(f'Comparison of Prices \n(Sample Size = {plt_samp_size})')


# Add legend
plt.legend()

plt.show()
return ids_to_use
```

Data and charts about outlier removal

In [23]:

```
# drop price outliers 1.5 and 2.0 IQR
# base assumption of outliers
IQR_mult1=1.5
car_df_no_outliers_1_IQR = drop_outlier(cars_no_nulls, 'price', IQR_mult1)
rows_removed1 = cars_no_nulls.shape[0]  - car_df_no_outliers_1_IQR.shape[0]
rows_removed_pct1 = rows_removed1/cars_no_nulls.shape[0]
# 2nd assumption: keep more outliers in the analysis
IQR_mult2=3

# chart with sample size equal to all the rows after dropping nulls
car_df_no_outliers_2_IQR = drop_outlier(cars_no_nulls, 'price', IQR_mult2)
rows_removed2 = cars_no_nulls.shape[0]  - car_df_no_outliers_2_IQR.shape[0]
rows_removed_pct2 = rows_removed2/cars_no_nulls.shape[0]
print(f"Rows before outlier removal = {cars_no_nulls.shape[0]}")
print(f"With IQR*{IQR_mult1} assumption, {rows_removed1} rows are removed ({rows_re
print(f"With IQR*{IQR_mult2} assumption, {rows_removed2} rows are removed ({rows_re

ids_to_use1 = cars_no_nulls.sample(cars_no_nulls.shape[0])
#plot_outliers_vs_orig(car_df, cars_no_nulls,car_df_no_outliers_1_IQR,car_df_no_out
#plot_outliers_vs_orig(car_df, cars_no_nulls,car_df_no_outliers_1_IQR,car_df_no_out
#plot_outliers_vs_orig(car_df, cars_no_nulls,car_df_no_outliers_1_IQR,car_df_no_out
#plot_outliers_vs_orig(car_df, cars_no_nulls,car_df_no_outliers_1_IQR,car_df_no_out

# now show smaller sample size to get a better feel
ids_to_use2 = car_df_no_nulls_balanced.sample(min(1000,car_df_no_nulls_balanced.sha
#plot_outliers_vs_orig(car_df, cars_no_nulls,car_df_no_outliers_1_IQR,car_df_no_out
#plot_outliers_vs_orig(car_df, cars_no_nulls,car_df_no_outliers_1_IQR,car_df_no_out
#plot_outliers_vs_orig(car_df, cars_no_nulls,car_df_no_outliers_1_IQR,car_df_no_out
#plot_outliers_vs_orig(car_df, cars_no_nulls,car_df_no_outliers_1_IQR,car_df_no_out
#plot_outliers_vs_orig(car_df, cars_no_nulls,car_df_no_outliers_1_IQR,car_df_no_out
#ids_used = plot_outliers_vs_orig(car_df, cars_no_nulls,car_df_no_outliers_1_IQR,ca
```

```
drop_outlier(): lower bound = -24610.0
drop_outlier(): upper bound = 59550.0
drop_outlier(): lower bound = -56170.0
drop_outlier(): upper bound = 91110.0
Rows before outlier removal = 12753
With IQR*1.5 assumption, 135 rows are removed (1.06%) leaving 12618 rows in data set
With IQR*3 assumption, 25 rows are removed (0.20%) leaving 12728 rows in data set
```

Make choice on outlier strategy

- choice IQR 1(mult is 1.5) or IQR 2(mult is 3)
- CHOOSING IQR 1

In [24]:
```python
car_df_no_outliers = car_df_no_outliers_1_IQR
```

Unrealistically low prices

The above outlier approach used InterQuartileRange approach to identifying outliers. We should also look at unrealistic prices from a business perspective. Prices for a car less then $100 are most likely invalid transactions/bad data

In [25]:
```python
# find prices less than $100
car_w_price_lt_100 = car_df_no_outliers[car_df_no_outliers['price'] < 100]
car_w_price_lt_100.sample(10)
count_car_w_price_lt_100 = car_w_price_lt_100.shape[0]
print(f'count of cars with price less than $100 = {count_car_w_price_lt_100:,.0f}')
```

```
count of cars with price less than $100 = 886
```

In [26]:
```python
# let's analyze these very cheap cars
# use the commented col_list to see all columns
#col_list =  car_w_price_lt_100.columns
# I ran this and most look very 'normal' in terms of count distribution
# See manufacturer
col_list = ['manufacturer']
max_detail = 15
#eval_col_counts(car_w_price_lt_100, col_list, max_detail = max_detail, sort_by_col

col_list = ['year']
max_detail = 15
#eval_col_counts(car_w_price_lt_100, col_list, max_detail = max_detail, sort_by_col
```

In [27]:
```python
def plot_cars_data(data, samp_size = 'All', title = 'Price chart', ids_to_use=None)
  plt.figure(figsize=(10, 6))

  if samp_size == 'All' and ids_to_use is None:
      plt_samp_size = data.shape[0]
  else:
      if ids_to_use is None:
          plt_samp_size = samp_size
      else:
          plt_samp_size = ids_to_use.shape[0]
```

```python
    if ids_to_use is None:
        plt_id = data.sample(plt_samp_size)
    else:
        plt_id = ids_to_use

    plt_data = pd.merge(plt_id["id"], data, on='id', how='left')

    plt_data = plt_data.sort_values(by="id")

    # Plot each DataFrame with a different color
    plt.plot(plt_data['id'], plt_data['price'], label='Price', color='black')

    # Add labels and title
    plt.xlabel('ID')
    plt.ylabel('Price')
    plt.title(f'{title} \n(Sample Size = {plt_samp_size})')

    # Add legend
    plt.legend()

    plt.show()
    return ids_to_use
```

In [28]:
```python
cars_clean_df = car_df_no_outliers_1_IQR[car_df_no_outliers_1_IQR['price'] >= 100]
print(f'By dropping rows that have price less than $100, we now have {cars_clean_df
print(cars_clean_df.columns)
#plot_cars_data(cars_clean_df)
#plot_cars_data(cars_clean_df, samp_size = 1000)

print(cars_clean_df.isnull().any().sum())
print(cars_clean_df['year'])
```

```
By dropping rows that have price less than $100, we now have 11,732 rows in the prim
ary data set
Index(['region', 'manufacturer', 'model', 'fuel', 'title_status',
       'transmission', 'type', 'paint_color', 'state', 'odometer', 'year',
       'id', 'price'],
      dtype='object')
0
263504    2017.0
263505    1998.0
263506    2020.0
263508    2009.0
263509    2015.0
           ...
282880    2019.0
282882    2018.0
282884    2018.0
282885    2017.0
282886    2018.0
Name: year, Length: 11732, dtype: float64
```

Decision about cars less than $100

- I will drop these cars from the analysis for now.

- I do not see a clear pattern or justification for the price being so low for a vehicle in the USA
- Keeping these would skew the data analysis (and may have already skewed the IQR outlier analysis)
- For now we will not redo the IQR outlier analysis
- The cleanest data set so far is now called 'cars_clean_df'

Recap of price outlier removal

I will use IQR times 1.5 on the price column to remove outliers

- They are mostly large unrealistic numbers. Even if they are real, they are rare situations and not helpful to the core project goals of managing overall inventory optimally

I will drop prices less than $100, reducing available rows

The cleanest data set so far is now called 'cars_clean_df'

Unusual characters analysis

In [29]:
```
def detect_unusual_chars(df, allowed_chars=None):
  if allowed_chars is None:
    allowed_chars = string.ascii_letters + string.digits + string.punctuation + ' '

  def has_unusual_chars(text):
    return bool(re.search(f'[^{allowed_chars}]', text))

  string_cols = df.select_dtypes(include=['object'])
  mask = string_cols.apply(lambda col: col.map(has_unusual_chars))
  mask = mask.any(axis=1)

  return df[mask]
```

In [30]:
```
# find unusual characters in string columns
fld = 'model'
u_df = detect_unusual_chars(car_df[[fld]].astype(str))
unique = u_df[fld].unique()
print(f"{len(unique)} rows have unusual charaters in the column {fld}:")
print(unique)
print()
# find characters with $ embedded in string columns
dollar_rows = car_df[car_df[fld].astype(str).str.contains('\$')]
print(f"Number of rows with $ in field {fld} is {dollar_rows.shape[0]}")
```

```
1 rows have unusual charaters in the column model:
['X5M']

Number of rows with $ in field model is 0
```

Decision about unusual characters

- We will keep the unusual characters discovered in the 'model' feature. This feature has almost 30,000 unique values and in its current form cannot be very helpful in our analysis (see section 'Interpreting the Model Feature' further down in this notebook)
- We will keep the '$' in the model feature but will need to account for it while doing string parsing code routines.
- The other features in the data set do not have unusual characters

Quality of the domain of feature values

All feature field domains (range of distinct values) have been manually reviewed

The 'region' field has some potential duplicates or at minimum unclear values:

- 'bloomington' and 'bloomington-normal'
- 'kansas city' and 'kansas city/MO'
- 'florence' and 'florence / muscle shoals'

The 'drive' field has approximately 50,000 rows with value 'rwd':

- This may be a typo as I assume it means 'rear wheel drive' which every car has

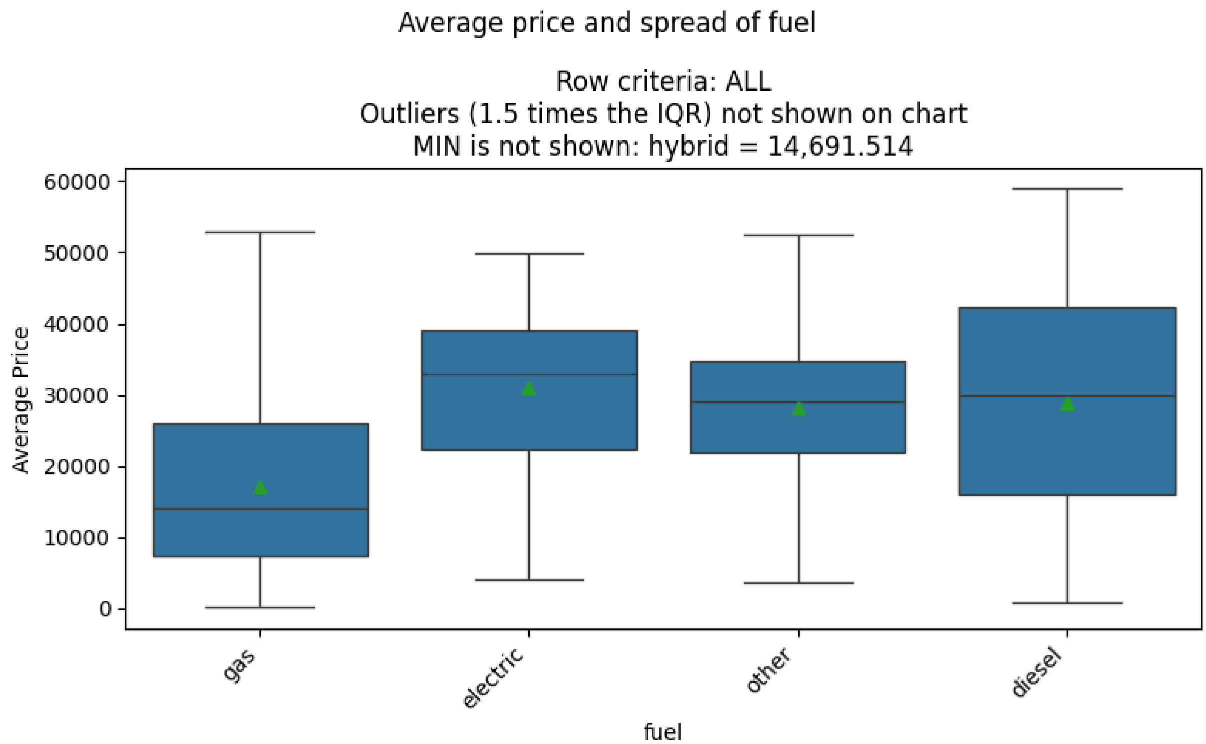
We will keep these values in the data set until we see the impact of them on the regression. They are not neccessarily wrong but unclear.

The other fields besides 'model' and 'drive' have reasonable values upon visual inspection of each .csv file generated

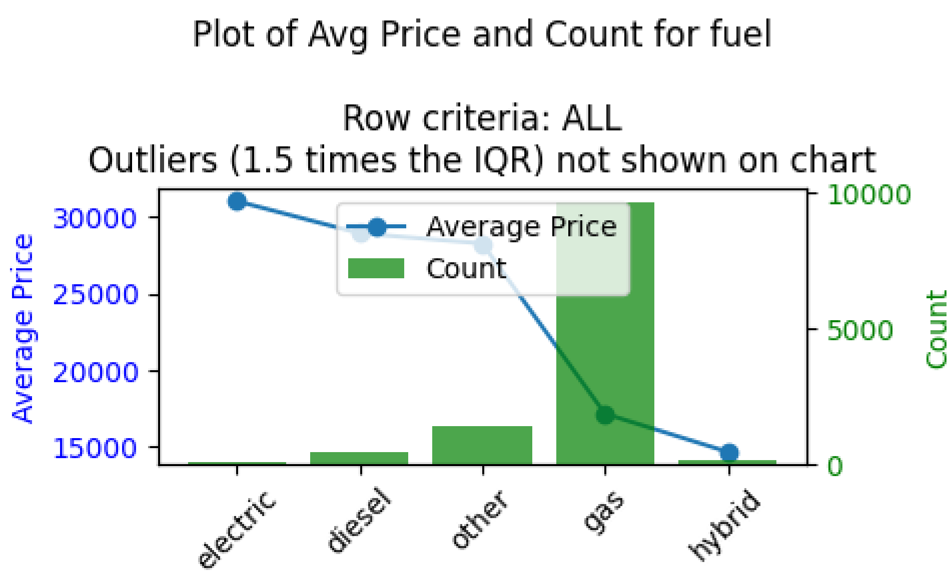
Revisit price and count charts after clean-up

In [31]:
```
col_list = ['fuel','year']

eval_col_avg_price(cars_clean_df, col_list,
                    lower_price_q = 0.0, upper_price_q = 1, lower_count_q = 0.0, up
```

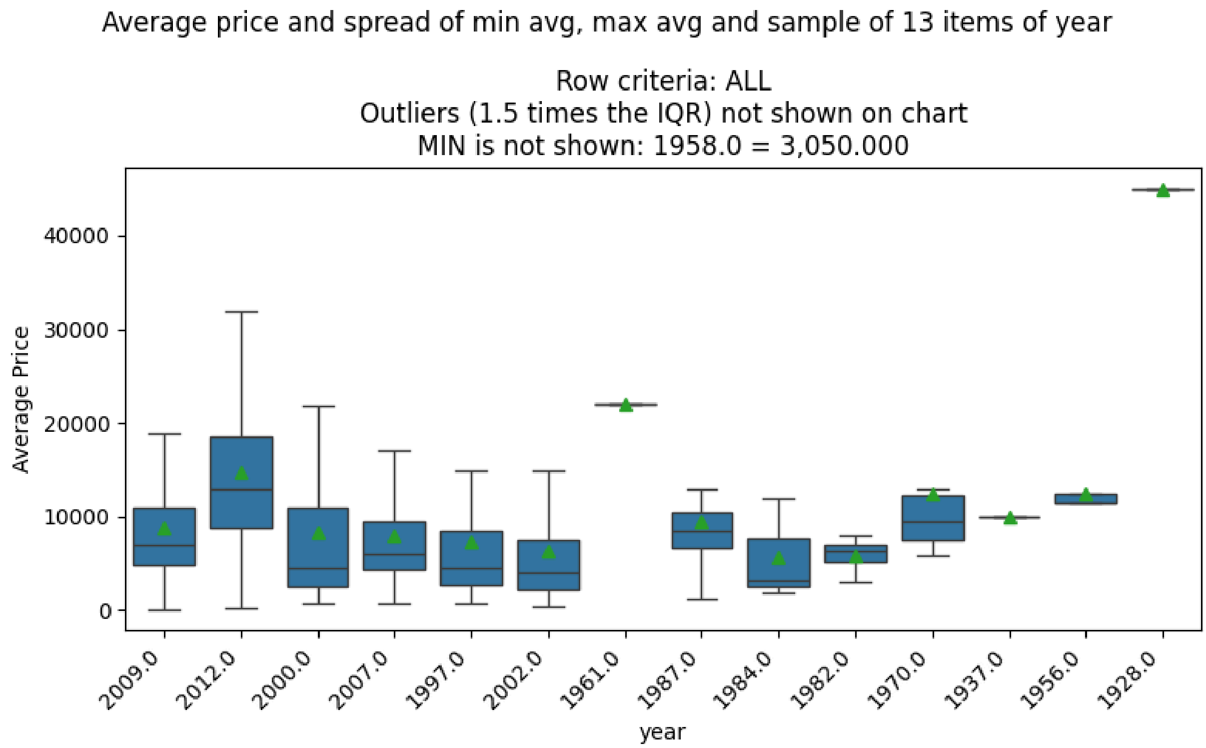
```
fuel has 5 distinct values
See 5 of them
       fuel    price  count
1  electric  $31,010     53
0    diesel  $28,920    465
4     other  $28,255   1400
2       gas  $17,174   9676
3    hybrid  $14,692    138
Will display all categories:
['electric', 'diesel', 'other', 'gas']
```



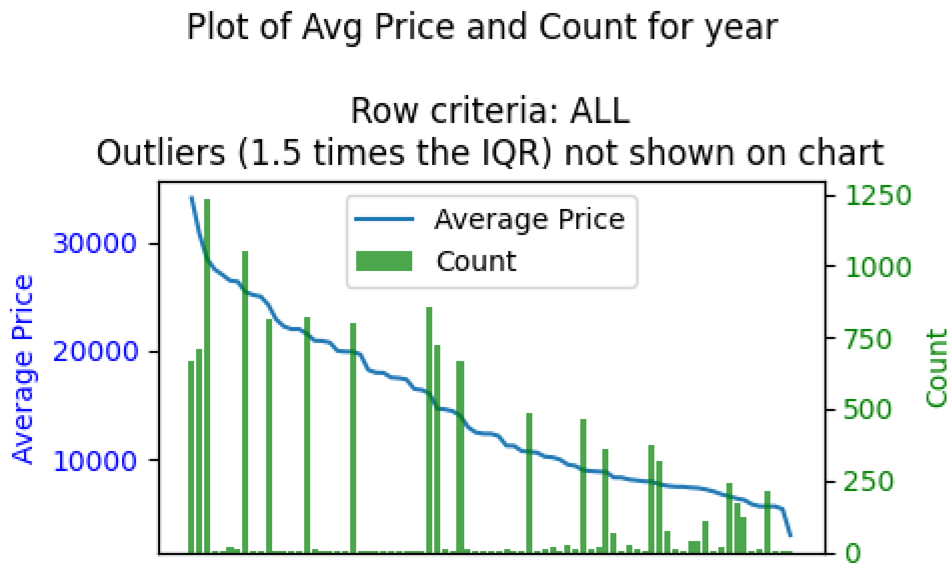

```
drop_outlier(): lower bound = -444.2382802811044
drop_outlier(): upper bound = 46538.116301502
Potential outliers for fuel = 0
Processing Avg Price and Count Chart - May take up to 30 seconds for some charts
no duplicates
<Figure size 640x480 with 0 Axes>
```



```
----
year has 80 distinct values
See 15 of them
      year    price  count
1   1928.0  $45,000      1
78  2020.0  $34,048    668
77  2019.0  $30,797    706
76  2018.0  $28,465   1233
16  1955.0  $27,496      2
4   1934.0  $27,000      2
79  2021.0  $26,462     20
26  1968.0  $26,369      8
75  2017.0  $25,413   1048
30  1972.0  $25,144      4
24  1966.0  $25,000      4
74  2016.0  $24,228    816
3   1932.0  $22,900      1
29  1971.0  $22,258      3
20  1962.0  $22,000      2
<Figure size 640x480 with 0 Axes>
```




```
drop_outlier(): lower bound = -10130.096153846152
drop_outlier(): upper bound = 38386.05769230769
Potential outliers for year = 1
Processing Avg Price and Count Chart - May take up to 30 seconds for some charts
no duplicates
<Figure size 640x480 with 0 Axes>
```




```
----
<Figure size 640x480 with 0 Axes>
```

Let's drill down on data where avg price is high and count is also high

- Simplistically, these would seem to be valuable cars to the dealer
- This rule might help us understand data and get some intuition about features that drive revenue

In [32]:
```
print('interactive tool to use')
print('Uncomment to run if want to explore different quantiles')
col_list = ['region', 'manufacturer', 'model', 'fuel',
            'title_status', 'transmission','type','paint_color','state','year']
# Choose lower_price_q (quantile lower bound for average price)
# Choose lower_count_q (quantile lower bound for quantity sold)
#eval_col_avg_price(cars_clean_df, col_list,
#                     lower_price_q = 0.7, upper_price_q = 1, lower_count_q = 0.4, u
```

```
interactive tool to use
Uncomment to run if want to explore different quantiles
```

Recap of charts drilling down to top 30% in price and top 60% in volume

- Diesel fel and type pick-ups and trucks seem to sell in this high price-high volume range
- White and black color seem popular
- GMC, audi, and cadilac are more common manufacturers in this range
- States from middle and southern part of the USA have highest volume in this range

In [33]:
```
cars_clean_df['paint_color']
```

```
Out[33]: 263504     black
263505      grey
263506     white
263508     black
263509     black
           ...
282880    silver
282882    silver
282884     white
282885     white
282886     white
Name: paint_color, Length: 11732, dtype: object
```

Recap of outlier section

Prices outliers have been removed bsaed on IQR and how close price is to $0 (<$100 removed)

The cleanest data set is called 'cars_clean_df' with 230,998 rows and 10 feature columns

Data Split

- I need a hold our or test data set to test our final model
- I will use a k-fold cross-validation technique for hyperparamter tuning (cv=5)
- However, to also vary the feature set, we have an explicit valiation set of 10% also
- We will use 70% of data for training, 10% for feature validation, and cross-validatoin and 20% for final testing

In [34]:
```
def train_val_test_split(X, y, test_size=0.2, val_size=0.1, random_state=42):
  # Split into train and test sets
  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size, ra

  # Split the training set into train and validation sets
  X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=val

  return X_train, X_val, X_test, y_train, y_val, y_test
```

In [35]:
```
X_df = cars_clean_df.drop(columns=['id','price'])
print(f"Feature columns are: {X_df.columns}")
y = cars_clean_df['price']

X_train, X_val, X_test, y_train, y_val, y_test = train_val_test_split(X_df, y, test
```

```
Feature columns are: Index(['region', 'manufacturer', 'model', 'fuel', 'title_statu
s',
       'transmission', 'type', 'paint_color', 'state', 'odometer', 'year'],
      dtype='object')
```

Feature engineering and hyperparamter tuning

To improve our ability to predict car prices form the input data we will:

- Create useful columns from categorical columns (OneHotEncoding)
- Create standard scaled polynomial features for numerical columns

There are many, many other options but will focus on tools discussed to this point (Module 11) in the course

In [36]:
```python
# Create one hot columns from the words in the model column.
# Use nltk to parse into lower case punctuation-free words without stop words.
# Also make sure the resulting list does not repeat values in the 'type' and 'manuf
# Use this list and resulting histogram count to create a large number of one_hot_c
# if a given wword has a count more than 50 in the given data frame (X_train usuall
#
# Keep a separate list one_hot_cols of these columns to instruct the system's prepr
# Eventually could
```

In [37]:
```python
nltk.download('stopwords')
from nltk.corpus import stopwords

the_stop_words = stopwords.words('english')
def identify_model_keywords( X_df,  sample_size=1000000, src_col_name = 'model', st
    #todo: 7000 min for testing - move to 50
    # custom feature for model column for now
    if stop_words is None:
        print("stop words invalid")
    col_to_clean = src_col_name
    act_sample_size = min(sample_size,X_df.shape[0])
    df = X_df[col_to_clean].reset_index().sample(act_sample_size).copy()
    #print(df.head())

    # Define stopwords list (includes 'the')
    stop_words = stop_words
    #print(f"the available df columns are: {X_df.columns}")
    type_words_set = set(X_df['type'])
    type_words = list(type_words_set)
    manufacturer_words_set = set(X_df['manufacturer'])
    manufacturer_words = list(manufacturer_words_set)

    # as we find one_hots that have low importance can add here to officially drop
    learned_low_value_words = []

    # Function to clean text and remove stopwords
    # Assumes inner function can see variables in outer scope
    def clean_text(text):
      # Lowercase text
      text = text.lower()
      # Remove punctuation
      text = re.sub(r'[^\w\s]', '', text)
      # Tokenize words
      words = text.split()
      # Remove stopwords
      filtered_words1 = [word for word in words if word not in stop_words]

      filtered_words2 = [word for word in filtered_words1 if word not in type_words
```

```
      filtered_words3 = [word for word in filtered_words2 if word not in manufactur

      filtered_words4 = [word for word in filtered_words3 if word not in learned_lo

      return filtered_words4


    # Apply cleaning function to 'text' column
    df['cleaned_text'] = df[col_to_clean].apply(clean_text)

    # Combine all cleaned text into a single list
    all_words = []
    for words in df['cleaned_text']:
      all_words.extend(words)

    # Create a dictionary to store word frequencies
    word_counts = {}
    for word in all_words:
      if word not in word_counts:
        word_counts[word] = 0
      word_counts[word] += 1

    # Filter out low-frequency words (optional)
    min_count = min_occurrence  # Adjust minimum count as needed

    filtered_counts = {word: count for word, count in word_counts.items() if count
    if verbose:
        print(f"word count is {len(filtered_counts)} using minimum occurrence level

    return filtered_counts

#todo: get rid of this one after we get things working
def plot_model_keyword(filtered_counts_df):
    # Create a histogram
    plt.bar(filtered_counts_df.index, filtered_counts_df['word_count'])
    plt.xlabel("Word From Model feature")
    plt.ylabel("Frequency")
    plt.title("Histogram of Words (excluding stopwords, types, and manufacturers)")
    if filtered_counts_df.shape[0] > 30:
        plt.xticks([])
    else:
        plt.xticks(rotation=90)  # Rotate x-axis labels for better readability

    plt.show()
    plt.cla()
    plt.clf()

def get_expected_one_hot_cols(filtered_counts):
    return ['my_one_hot_' + word for word in filtered_counts.keys()]
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\bbfor\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [38]:

```python
class ModelofCarTransformer(TransformerMixin,BaseEstimator):
    def __init__(self, column_names, stop_words, min_occurrence = 4000, max_one_hot
        self.column_names = column_names # it better be called 'model'!
        self.transformed_feature_names = []
        self.min_occurrence = min_occurrence
        self.max_one_hots = max_one_hots
        self.valid_words = valid_words # normally and recommended created by fit()
        # requires nltk only lightly for stopwords. could pickle?
        self.stop_words = stop_words
        self.my_one_hot_prefix = 'my_one_hot_col_'
        self.already_fit = False

    def identify_model_keywords( self, X_df, min_occurrence = 7000, sample_size=100
        #print("in ModelofCarTransformer, calling identify_model_keywords()")
        return identify_model_keywords( X_df, sample_size, src_col_name, stop_words

    # Create one_hot features from the model field
    def gen_model_one_hots( self, data, filtered_counts, valid_words):

        # Function for replacement
        def remove_special_chars(text):
            return re.sub(r'[^\w\s]', '', text)

        # Function to check if word exists (vectorized for efficiency)
        def check_for_word(text, word):
          # added lower case conversion
          return  text.str.lower().str.contains(word, case=False)

        df = data.copy()
        new_col_and_data = []
        new_col_list = []
        new_data_list = []
        count = 1
        df['model_w_o_special'] = 0
        sorted_filtered_words_list = sorted(filtered_counts.keys(), key=lambda x: x
        for word_to_find in sorted_filtered_words_list:
            if count > self.max_one_hots:
                break
            if word_to_find in valid_words:
                #print(f'preparing {word_to_find} to be a one_hot_col')
                #new_col_array_bool = empty_array = np.empty(min(self.max_one_hots,
                #new_col_array_int = empty_array = np.empty(min(self.max_one_hots,
                new_col_array_bool = np.empty(df.shape[0])
                new_col_array_int = np.empty(df.shape[0])

                # Apply the function with vectorized operations

                df['model_w_o_special'] = df['model'].apply(remove_special_chars)
                new_col_array_bool = check_for_word(df['model_w_o_special'], word_t

                # Convert the boolean column to 0 or 1 (optional)
                new_col_array_int = new_col_array_bool.astype(int)

                # make sure you found some non-zero values
                condition = new_col_array_int == 1
```

```
            non_zeros = np.where(condition)
            if len(non_zeros[0]) > 0:
                # create a dictionary of the column name and the associated ar
                # todo: check for characters of word_to_find that can't be used
                new_col_name = self.my_one_hot_prefix+ word_to_find
                new_col_dict = {'new_col_name': new_col_name, 'new_col_one_hot'
                new_col_and_data.append(new_col_dict)
                new_col_list.append(new_col_name)
                new_data_list.append(new_col_array_int)

            else:
                #todo: raise exception here
                print(f"************************************all zeros for : {wo
                # for debug reasons
                print(f"non_zeros : {non_zeros}")
                print(f"new_col_array_int : {new_col_array_int}")
                print(f"new_col_array_bool : {new_col_array_bool}")
        else:
            print(f"skipping {word_to_find}")


    #print("in fit,gen_model, df rows = ", df.shape[0])
    if len(new_data_list)>0:
        #print("length of new_data_list", len(new_data_list))
        #print("df",np.shape(df))
        #print("df columns:", df.columns)
        # Stack arrays horizontally
        data_array = np.column_stack(new_data_list)
        #print("data_array",np.shape(data_array))

        df_merge_cols = [col for col in df.columns]
        for col in new_col_list:
            df_merge_cols.append(col)

        df_merged = pd.DataFrame(np.column_stack([df.to_numpy(), data_array]),
        #print("df_final",np.shape(df_merged))

        df_final = df_merged.copy()
        #print("df_final",np.shape(df_final))
        #print("df_final cols", list(df_final.columns))
    else:
        df_final = df

    return df_final

def fit(self, X, y=None):
    if 'model' in X.columns:
        X_df = pd.DataFrame(X)
        if self.already_fit:
            print("Already fit but refitting")
        #print("in ModelofCarTransformer.fit(), calling identify_model_keywords
        filtered_counts =self.identify_model_keywords(X_df)
        self.valid_words = filtered_counts.keys()
    else:
        self.valid_words = None
```

```
        return self


    def rationalize_cols(self, X_w_some_one_hots):
        #print("In rationalize_cols()")
        new_zero_col_list = []
        for col in self.cols_after_fit:
            if not col in X_w_some_one_hots:
                if self.my_one_hot_prefix in col:
                    #print(f"adding {col} to rationalize shape to the original fit"
                    new_zero_col_list.append(col)

        if len(new_zero_col_list)>0:
            #print(f"shape to create zeros col array {X_w_some_one_hots.shape[0]} ,
            new_zero_col_array = np.zeros((X_w_some_one_hots.shape[0],len(new_zero_
            X_w_some_one_hot_rationalized = X_w_some_one_hots.copy()
            #print("length of new_data_list", len(new_zero_col_list))
            #print("X_w_some_one_hot_rationalized",np.shape(X_w_some_one_hot_ration
            #print("X_w_some_one_hot_rationalized columns:", X_w_some_one_hot_ratio
            #print("new_zero_col_array",np.shape(new_zero_col_array))

            df_merge_cols = [col for col in X_w_some_one_hot_rationalized.columns]
            for col in new_zero_col_list:
                df_merge_cols.append(col)

            df_merged = pd.DataFrame(np.column_stack([X_w_some_one_hot_rationalized
            #print("df_final",np.shape(df_merged))

            df_final = df_merged.copy()
            #print("df_final cols", list(df_final.columns))
        else:
            df_final = X_w_some_one_hots
        return df_final

    def transform(self, X):
        X_transformed = X.copy()  # Copy the input DataFrame to avoid modifying the
        if 'model' in X.columns:
            #print("in ModelofCarTransformer,transform()")
            X_transformed = X.copy()  # Copy the input DataFrame to avoid modifying
            #print("in ModelofCarTransformer.transform(), calling identify_model_ke
            filtered_counts =self.identify_model_keywords(X_transformed)
            #todo: should raise or warn if valid_words is empty
            X_w_one_hots = self.gen_model_one_hots(X_transformed, filtered_counts,
            X_w_one_hots = X_w_one_hots.drop(['model_w_o_special'], axis=1)

            for col in self.column_names:
                X_w_one_hots = X_w_one_hots.drop([col], axis=1)

        else:
            print("^^^^^^^^^^^^^^^^^^  NO MODEL COL ^^^^^^^^^^^^^^^^^^^^")
            X_transformed['model_inactive'] = 1
            X_w_one_hots = X_transformed[X_transformed['model_inactive']]

        self.transformed_feature_names = X_w_one_hots.columns
        if not self.already_fit:
            self.cols_after_fit = X_w_one_hots.columns
```

```
            self.already_fit = True
        else:
            # rationalize_columns creates any one_hot columns that were missing fro
            # column set of my_one_hots matches what was there at fit
            # Set them to zeros (because we know we didn't see any of these values
            X_w_one_hots = self.rationalize_cols(X_w_one_hots)

        #X_w_one_hots.to_csv("saved_output/last_transform.csv")
        #print(f"I was transformed: {self.transformed_feature_names} columns now")
        print("in modelofcartransform, X_w_one_hots shape",np.shape(X_w_one_hots))

        return X_w_one_hots

    def get_feature_names_out(self, input_features):
        return self.transformed_feature_names
```

In [39]:
```
def set_up_one_hot_preprocessors(custom_model_cols, categorical_cols, numerical_col

    my_model_of_car_transformer = ModelofCarTransformer(column_names=['model','manu
    #todo: target_col
    my_one_hot_preprocessor = make_column_transformer(
        (my_model_of_car_transformer, custom_model_cols),
        (Pipeline([
            ('scaler', StandardScaler()),
            ('poly', PolynomialFeatures(degree=degrees))
        ]), numerical_cols),
        (OneHotEncoder(sparse_output=False,drop='first', handle_unknown='ignore'),
        remainder="drop"
    )

    return my_one_hot_preprocessor

def set_up_pipeline(preprocessor, alpha=None):
    if alpha is None:
        pipeline1 = Pipeline([
        ('preprocessor', preprocessor),
        ('selector', SelectFromModel(Lasso(max_iter = 3000, alpha = 100))),
        ('regressor', Ridge(max_iter=1000))
        ])
    else:
        pipeline1 = Pipeline([
        ('preprocessor', preprocessor),
        ('selector', SelectFromModel(Lasso(max_iter = 3000, alpha = 100))),
        ('regressor', Ridge(alpha=alpha, max_iter = 1000))
        ])
    return pipeline1
```

In [40]:
```
# Default global. Set by gridsearch to discovered value
BEST_ALPHA = 1

def run_gridsearchcv(pipeline1, X_train,y_train, param_grid = {'regressor__alpha':
    grid_search = GridSearchCV(pipeline1, param_grid, scoring='neg_mean_squared_err
    grid_search.fit(X_train, y_train)
    best_alpha = grid_search.best_params_['regressor__alpha']
    print("best alpha",best_alpha)
```

```
    return grid_search, best_alpha
```

In [41]:
```
def prep_to_save_grid_search_details(grid_search, categorical_cols, numerical_cols,
    # Get the best model and its coefficients
    best_model = grid_search.best_estimator_
    best_lasso = best_model.named_steps['regressor']
    best_coef = best_lasso.coef_
    #print(best_coef)

    # Get feature names
    feature_names_in = categorical_cols + numerical_cols
    feat_names_preprocessor = grid_search.best_estimator_.named_steps['preprocessor
    feat_names_selector = grid_search.best_estimator_.named_steps['selector'].get_f

    # Get the best score
    best_score = grid_search.best_score_
    print("Best score:", best_score)

    # get mse
    best_model = grid_search.best_estimator_
    y_pred = best_model.predict(X_val)
    mse = mean_squared_error(y_val, y_pred)
    print("mse:",mse)

    # Calculate RMSE
    rmse_train = np.sqrt(mse_train)
    print("RMSE train:", rmse_train)

    rmse_val = np.sqrt(mse_val)
    print("RMSE val:", rmse_val)

    alpha = grid_search.best_params_['regressor__alpha']
    print("alpha:", alpha)
    # Set global BEST_ALPHA
    BEST_ALPHA = alpha

    details = {'alpha':alpha, 'best_score': best_score,'best_model': best_model, \
               'feature_names_in': feature_names_in, \
               'feat_names_preprocessor': feat_names_preprocessor, 'feat_names_sele
               'mse_train': mse_train, 'mse_val': mse_val, \
               'rmse_train': rmse_train, 'rmse_val': rmse_val}
    return details
```

In [42]:
```
def run_pipe_and_predict(pipeline2, X_train, y_train, X_val, y_val, verbose=True):
    if verbose:
        print("Running fit")
    pipeline2.fit(X_train, y_train)
    if verbose:
        print("running predict for X_train")
    train_pred = pipeline2.predict(X_train)
    if verbose:
        print("running predict for X_val")
    val_pred = pipeline2.predict(X_val)
```

```
    mse_train = mean_squared_error(y_train, train_pred)
    mse_val = mean_squared_error(y_val, val_pred)
    if verbose:
        print(f"model predict rmse_train: {np.sqrt(mse_train):,f}")
        print(f"model predict rmse_val: {np.sqrt(mse_val):,f}")
        print(f"model predict rmse gap :{abs(np.sqrt(mse_train)-np.sqrt(mse_val)):,


    return mse_train, mse_val
```

In [43]:
```
def run_grid_search_experiment(categorical_cols, numerical_cols, target_col, X_trai
    try:
        details = None
        best_alpha = None
        set_config(transform_output="default")
        start_time = time.time()
        time_struct = time.localtime(start_time)
        formatted_time = time.strftime("%I:%M:%S", time_struct)
        print(f'Starting experiment {exp_id} at {formatted_time}')
        details_list = []

        model_cols = ['manufacturer', 'type', 'model']

        preprocessor = set_up_one_hot_preprocessors(model_cols, categorical_cols, n

        pipeline1 = set_up_pipeline(preprocessor)

        param_grid = {'regressor__alpha': [ 1e4, 1, 1e2, 1e-2, 1e-1]}
        grid_search, best_alpha = run_gridsearchcv(pipeline1, X_train,y_train, para

        if best_alpha is None:
            best_alpha = 1
        end_time = time.time()
        elapsed_time = end_time - start_time
        print(f"Grid Search done. Elapsed_time: {elapsed_time}")
        pipeline2 = set_up_pipeline(preprocessor, best_alpha)

        mse_train, mse_val = run_pipe_and_predict(pipeline2, X_train, y_train, X_va
        end_time = time.time()
        elapsed_time = end_time - start_time
        print(f"Pipe and Predict done. Elapsed_time: {elapsed_time}")
        print("train ",mse_train, " val", mse_val)
        details = prep_to_save_grid_search_details(grid_search, categorical_cols, n

        details_list.append(details)

        if dump_to_pickle:
            with open(f"saved_output/{exp_id}_details.pickle", "wb") as f:
              # Pickle the list and write it to the file
              pickle.dump(details, f)
        end_time = time.time()
        elapsed_time = end_time - start_time
        print(f"finished experiment elapsed_time: {elapsed_time}")
    finally:
        set_config(transform_output="default")
```

```
    return details
```

In [44]:
```
print(cars_clean_df.columns)
print(X_train.shape)
print(X_test.shape)
```

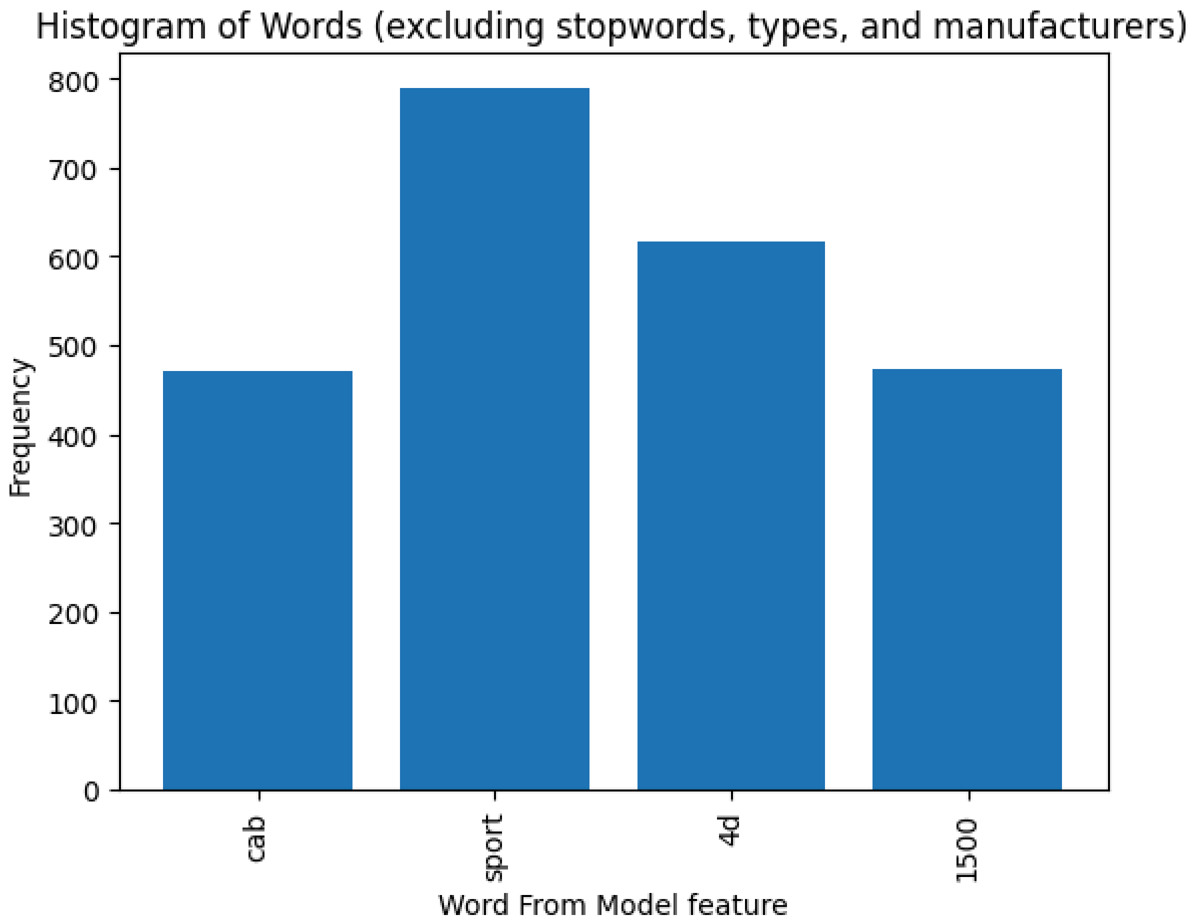
```
Index(['region', 'manufacturer', 'model', 'fuel', 'title_status',
       'transmission', 'type', 'paint_color', 'state', 'odometer', 'year',
       'id', 'price'],
      dtype='object')
(8211, 11)
(2347, 11)
```

We created a dynamic one-hot encoding based on phrase in the model field

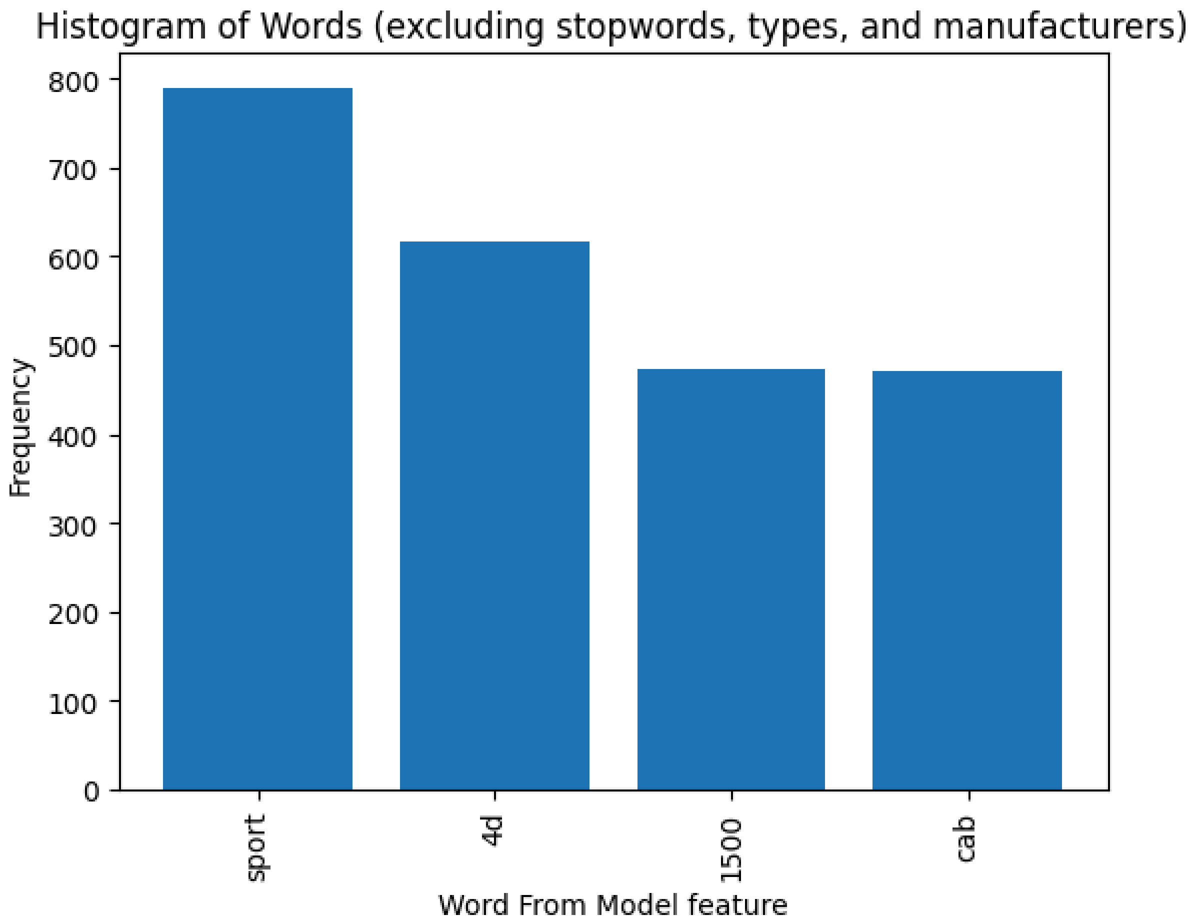
- The above process uses them but if you want to see examples of popular words used to encode, use this cell below

In [ ]:

In [45]:
```
# Set min_occurrence to different values to see the distribution of popular words i
# % of rows in X_train
pct_tuning_min_occurrence = 4
tuning_min_occurrence = round(pct_tuning_min_occurrence/100*X_train.shape[0]) #
filtered_counts =identify_model_keywords(X_train, min_occurrence = tuning_min_occur
filtered_counts_df = pd.DataFrame.from_dict(filtered_counts, orient='index', column
plot_model_keyword(filtered_counts_df)
print("number of one hots to be created for model", filtered_counts_df.shape[0])
df_sorted = filtered_counts_df.sort_values(by='word_count', ascending=False)
top_values_df = df_sorted.iloc[:15]
plot_model_keyword(top_values_df)
```




```
number of one hots to be created for model 4
```




```
<Figure size 640x480 with 0 Axes>
```

THIS CODE WILL TAKE MANY MINUTES TO RUN.

SET OK_TO_RUN_TUNING = True, if you want to run it

In [46]:
```
OK_TO_RUN_TUNING = True
if OK_TO_RUN_TUNING:
    #categorical_cols = ['region', 'manufacturer', 'model', 'fuel', 'title_status',

    #categorical_cols =  ['region', 'manufacturer', 'model', 'condition', 'cylinder
    #                                                    'type', 'paint_color', 'stat

    #categorical_cols =  [ 'manufacturer', 'model', 'fuel', 'title_status', 'transm
    #                                                    'type', 'paint_color', 'stat

    #categorical_cols = ['state', 'type', 'manufacturer','paint_color','fuel', 'tit
    #categorical_cols = [ 'type']
    #categorical_cols = ['state', 'type', 'manufacturer','paint_color','fuel', 'tit
    #categorical_cols = ['state', 'type', 'manufacturer','paint_color','fuel', 'tit
    #categorical_cols = ['type', 'state', 'manufacturer', 'fuel','title_status', 't
    categorical_cols = ['type', 'state', 'manufacturer', 'fuel','title_status', 'tr


    numerical_cols = ['year']
    #numerical_cols = ['year']
```

```python
    #numerical_cols = []
    target_col = 'price'
    details = run_grid_search_experiment(categorical_cols, numerical_cols, target_c
    #beep()
```

```
Starting experiment 1 at 11:19:06
Fitting 2 folds for each of 5 candidates, totalling 10 fits
in modelofcartransform, X_w_one_hots shape (4105, 1)
```

```
C:\Users\bbfor\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\lin
ear_model\_coordinate_descent.py:697: ConvergenceWarning: Objective did not converg
e. You might want to increase the number of iterations, check the scale of the featu
res or consider increasing regularisation. Duality gap: 3.188e+08, tolerance: 6.641e
+07
  model = cd_fast.enet_coordinate_descent(
```

```
in modelofcartransform, X_w_one_hots shape (4106, 1)
[CV] END ..........................regressor__alpha=10000.0; total time=   1.1s
in modelofcartransform, X_w_one_hots shape (4106, 1)
in modelofcartransform, X_w_one_hots shape (4105, 1)
[CV] END ..........................regressor__alpha=10000.0; total time=   0.6s
in modelofcartransform, X_w_one_hots shape (4105, 1)
```

```
C:\Users\bbfor\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\lin
ear_model\_coordinate_descent.py:697: ConvergenceWarning: Objective did not converg
e. You might want to increase the number of iterations, check the scale of the featu
res or consider increasing regularisation. Duality gap: 3.188e+08, tolerance: 6.641e
+07
  model = cd_fast.enet_coordinate_descent(
```

```
in modelofcartransform, X_w_one_hots shape (4106, 1)
[CV] END ................................regressor__alpha=1; total time=   1.0s
in modelofcartransform, X_w_one_hots shape (4106, 1)
in modelofcartransform, X_w_one_hots shape (4105, 1)
[CV] END ................................regressor__alpha=1; total time=   0.5s
in modelofcartransform, X_w_one_hots shape (4105, 1)
```

```
C:\Users\bbfor\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\lin
ear_model\_coordinate_descent.py:697: ConvergenceWarning: Objective did not converg
e. You might want to increase the number of iterations, check the scale of the featu
res or consider increasing regularisation. Duality gap: 3.188e+08, tolerance: 6.641e
+07
  model = cd_fast.enet_coordinate_descent(
```

```
in modelofcartransform, X_w_one_hots shape (4106, 1)
[CV] END ............................regressor__alpha=100.0; total time=   1.2s
in modelofcartransform, X_w_one_hots shape (4106, 1)
in modelofcartransform, X_w_one_hots shape (4105, 1)
[CV] END ............................regressor__alpha=100.0; total time=   0.5s
in modelofcartransform, X_w_one_hots shape (4105, 1)
```

```
C:\Users\bbfor\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\lin
ear_model\_coordinate_descent.py:697: ConvergenceWarning: Objective did not converg
e. You might want to increase the number of iterations, check the scale of the featu
res or consider increasing regularisation. Duality gap: 3.188e+08, tolerance: 6.641e
+07
  model = cd_fast.enet_coordinate_descent(
```

```
in modelofcartransform, X_w_one_hots shape (4106, 1)
[CV] END .............................regressor__alpha=0.01; total time=   0.9s
in modelofcartransform, X_w_one_hots shape (4106, 1)
in modelofcartransform, X_w_one_hots shape (4105, 1)
[CV] END .............................regressor__alpha=0.01; total time=   0.6s
in modelofcartransform, X_w_one_hots shape (4105, 1)
C:\Users\bbfor\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\lin
ear_model\_coordinate_descent.py:697: ConvergenceWarning: Objective did not converg
e. You might want to increase the number of iterations, check the scale of the featu
res or consider increasing regularisation. Duality gap: 3.188e+08, tolerance: 6.641e
+07
  model = cd_fast.enet_coordinate_descent(
in modelofcartransform, X_w_one_hots shape (4106, 1)
[CV] END ..............................regressor__alpha=0.1; total time=   0.9s
in modelofcartransform, X_w_one_hots shape (4106, 1)
in modelofcartransform, X_w_one_hots shape (4105, 1)
[CV] END ..............................regressor__alpha=0.1; total time=   0.7s
in modelofcartransform, X_w_one_hots shape (8211, 4)
best alpha 1
Grid Search done. Elapsed_time: 10.492499113082886
Running fit
in modelofcartransform, X_w_one_hots shape (8211, 4)
running predict for X_train
in modelofcartransform, X_w_one_hots shape (8211, 4)
running predict for X_val
in modelofcartransform, X_w_one_hots shape (1174, 4)
model predict rmse_train: 7,457.571592
model predict rmse_val: 7,621.361447
model predict rmse gap :163.789855
Pipe and Predict done. Elapsed_time: 12.48499321937561
train  55615374.05170851  val 58085150.30317131
Best score: -56765780.64975144
in modelofcartransform, X_w_one_hots shape (1174, 4)
mse: 58085150.30317131
RMSE train: 7457.571592127595
RMSE val: 7621.361446826368
alpha: 1
finished experiment elapsed_time: 12.507627725601196
C:\Users\bbfor\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\pre
processing\_encoders.py:242: UserWarning: Found unknown categories in columns [2] du
ring transform. These unknown categories will be encoded as all zeros
  warnings.warn(
C:\Users\bbfor\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\pre
processing\_encoders.py:242: UserWarning: Found unknown categories in columns [2] du
ring transform. These unknown categories will be encoded as all zeros
  warnings.warn(
```

In [47]:
```
#details
```

In [ ]:

Modeling

With your (almost?) final dataset in hand, it is now time to build some models. Here, you should build a number of different regression models with the price as the target. In building your models, you should explore different parameters and be sure to cross-validate your findings.

Feature Selection

I will let the lasso regularization decide the feature selection through linear regression coefficients

- I will use the GridSearchCV to find the optimal lasso regression hyperparameter

To improve our ability to predict car prices form the input data we will generate 3 types of features: polynomial, interaction (x1*x2) and exponential.

There are many, many other options but will focus on tools discussed to this point (Module 11) in the course

-

In [ ]:

```
```

In [ ]:

```
```

In [48]:

```
# feat import after encoding
def get_importance_by_partial_match(feat_map, search_string):
    print(feat_map)
    matching_keys = [key for key in feat_map.keys() if search_string in key]
    feat_impt = 0
    for a_match in matching_keys:
        feat_impt = feat_impt + feat_map[a_match]

    return feat_impt

def run_feat_importance_perm(X_train, y_train, X_val, y_val, feat_cols, preprocesso


    X_train_cols = X_train[feat_cols]
    X_val_cols = X_val[feat_cols]

    if verbose:
        print('X_train passed in cols',X_train.columns)
        print('X_train feature cols',X_train_cols.columns)
    pipeline = set_up_pipeline(preprocessor, alpha = alpha)

    #pipeline.fit(X_train_cols, y_train)
    pipeline.fit(X_train, y_train)
```

```
    if verbose:
        print("original feature names")
        print(feat_cols)
        print("---")
        print(f"Number original features is {len(X_train.columns)}")

    preprocessor_feature_names = pipeline.named_steps['preprocessor'].get_feature_n
    selector_feature_names = pipeline.named_steps['selector'].get_feature_names_out

    if verbose:
        print(f"Number features in preprocessor step (feature engineering) is {len(
        print("Number of features sent to model after feature selection is ", len(s
        print("---")

        print("Run with all features to get MSE and RMSE")
    mse_train, mse_val = run_pipe_and_predict(pipeline, X_train_cols, y_train, X_va
    if verbose:
        print(f"MSE train: {mse_train:,.0f}")
        print(f"MSE val: {mse_val:,.0f}")

    print("----")
    print("Calculating permutations to find feature importance per feature")
    # Calculate permutation importance using the pipeline
    feat_import_results = permutation_importance(estimator=pipeline, X=X_val_cols,


    return feat_import_results, pipeline, mse_train, mse_val
```

In [49]:

```
def prep_to_save_feat_import_details(feat_import_results, pipeline, feat_cols, targ


    print("Feature columns with mean - 2*std GREATER THAN  0")
    for i in feat_import_results.importances_mean.argsort()[::-1]:
        if feat_import_results.importances_mean[i] - 2 * feat_import_results.import
            print(f"{feat_cols[i]:<40}"
                  f"{feat_import_results.importances_mean[i]:,.0f}"
                  f" +/- {feat_import_results.importances_std[i]:,.0f}")
    print("----")
    if verbose:
        print("Feature columns with mean - 2*std LESS THAN OR EQUAL TO 0")
        for i in feat_import_results.importances_mean.argsort()[::-1]:
            if feat_import_results.importances_mean[i] - 2 * feat_import_results.im
                print(f"{feat_cols[i]:<40}"
                      f"{feat_import_results.importances_mean[i]:,.0f}"
                      f" +/- {feat_import_results.importances_std[i]:,.0f}")

    # capture the change in rmse of the model field
    for i in feat_import_results.importances_mean.argsort()[::-1]:
        if feat_cols[i] == 'model':
            if math.isnan(feat_import_results.importances_mean[i]):
                mean = 0
            else:
                mean = feat_import_results.importances_mean[i]
```

```
            if math.isnan(feat_import_results.importances_std[i]):
                std = 0
            else:
                std = feat_import_results.importances_std[i]

            mean_change_rmse_for_model_field = np.sqrt(abs(mean))
            std_change_rmse_for_model_field = np.sqrt(abs(std))


    importance_map = dict(zip(feat_cols, feat_import_results.importances_mean))
    if verbose:
        print("Full list of feature columns")
        for orig_feat in feat_cols:
            orig_impt = importance_map[orig_feat]
            print(f"Original Feature: {orig_feat}, Average Importance (MSE change):

        # Get feature names

    preprocessor_feature_names = pipeline.named_steps['preprocessor'].get_feature_n
    selector_feature_names = pipeline.named_steps['selector'].get_feature_names_out


    # Calculate RMSE
    rmse_train = np.sqrt(mse_train)
    print("RMSE train:", rmse_train)

    rmse_val = np.sqrt(mse_val)
    print("RMSE val:", rmse_val)


    details = {'alpha':alpha, 'min_occurrence': min_occurrence,
            'feature_names_in': feat_cols,
            'preprocessor_feature_names': preprocessor_feature_names, 'selector_fea
            'mse_train': mse_train, 'mse_val': mse_val,
            'rmse_train': rmse_train, 'rmse_val': rmse_val,
            'orig_feat_importance_map': importance_map,
            'raw_feat_importance_mean' : feat_import_results.importances_mean,
            'raw_feat_importance_std' : feat_import_results.importances_std,
            'mean_change_rmse_for_model_field' : mean_change_rmse_for_model_field,
            'std_change_rmse_for_model_field' : std_change_rmse_for_model_field
            }
    return details
```

In [50]:
```
def run_feat_import_experiment(categorical_cols, numerical_cols, target_col, X_trai
                               y_train, X_val, y_val, alpha, min_occurrence = None,
    start_time = time.time()
    time_struct = time.localtime(start_time)
    formatted_time = time.strftime("%I:%M:%S", time_struct)
    print(f'Starting experiment {xp_id} at {formatted_time}')
    if verbose:
        print("rows in training data",X_train.shape[0])
        print("rows in validation data",X_val.shape[0])

    model_cols = ['manufacturer','model', 'type']
    if verbose:
        print(min_occurrence)
```

```
    if min_occurrence is not None:
        preprocessor = set_up_one_hot_preprocessors(model_cols, categorical_cols, n
    else:
        preprocessor = set_up_one_hot_preprocessors(model_cols, categorical_cols, n

    feat_cols = categorical_cols + numerical_cols
    for col in model_cols:
        if col not in feat_cols:
            feat_cols.append(col)

    feat_import_results, pipeline, mse_train, mse_val = run_feat_importance_perm(X_

    details = prep_to_save_feat_import_details(feat_import_results, pipeline, feat_

    if dump_to_pickle:
        with open(f"saved_output/{xp_id}_feat_import_details.pickle", "wb") as f:
          # Pickle the list and write it to the file
          pickle.dump(details, f)
    end_time = time.time()
    elapsed_time = end_time - start_time
    print(f"finished experiment {xp_id} in elapsed_time: {elapsed_time}")
    return details
```

Evaluation

With some modeling accomplished, we aim to reflect on what we identify as a high quality model and what we are able to learn from this. We should review our business objective and explore how well we can provide meaningful insight on drivers of used car prices. Your goal now is to distill your findings and determine whether the earlier phases need revisitation and adjustment or if you have information of value to bring back to your client.

This code may run for many tens of minutes or more depending on data size.

- Set OK_TO_RUN_FEAT_IMPORT = True, if you want to run it

In [51]:

```
OK_TO_RUN_FEAT_IMPORT = True
if OK_TO_RUN_FEAT_IMPORT:
    categorical_cols = ['type', 'state', 'manufacturer', 'fuel','title_status', 'tr
    #one_hot_cols  = [col for col in df2.columns if col.startswith('my_one')]
    numerical_cols = ['year']
    target_col = 'price'
    alpha = BEST_ALPHA
    feat_import_details = run_feat_import_experiment(categorical_cols, numerical_co
                                                     X_train, y_train, X_test, y_te
```

```
Starting experiment default at 11:19:19
in modelofcartransform, X_w_one_hots shape (8211, 150)
Running fit
in modelofcartransform, X_w_one_hots shape (8211, 150)
running predict for X_train
in modelofcartransform, X_w_one_hots shape (8211, 150)
running predict for X_val
in modelofcartransform, X_w_one_hots shape (2347, 150)
model predict rmse_train: 7,526.778617
model predict rmse_val: 8,151.215857
model predict rmse gap :624.437239
----
Calculating permutations to find feature importance per feature
in modelofcartransform, X_w_one_hots shape (2347, 150)
Feature columns with mean - 2*std GREATER THAN  0
year                                        112,782,812 +/- 3,438,494
type                                        25,303,306 +/- 1,057,633
manufacturer                                6,469,288 +/- 625,758
transmission                                5,915,169 +/- 451,540
fuel                                        3,234,129 +/- 399,172
region                                      838,976 +/- 215,611
----
RMSE train: 7526.778617451536
RMSE val: 8151.215856920535
finished experiment default in elapsed_time: 66.8907117843628
```

This code may run for many hours or more depending on data size.

- Set OK_TO_RUN_FEAT_IMPORT = True, if you want to run it

In [53]:
```
OK_TO_RUN_FEAT_IMPORT_ADDITIONAL = True
```

In [54]:
```
if OK_TO_RUN_FEAT_IMPORT_ADDITIONAL:
    # run for several alpha values and chart rmse
    alphas = [1e-2,1e-1,1,BEST_ALPHA, 1e1,1e2,1e3]
    alphas = list(set(alphas))
    alphas.sort()
    exp_alphas = []
    for alpha in alphas:
        xp_id = "exp_alpha_"+f"{alpha}"
        print("++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++")
        print(f"Running experiment {xp_id} for alpha: {alpha:,.4f}")
        feat_import_details = run_feat_import_experiment(categorical_cols, numerica
                                                         X_train, y_train, X_test, y_te
        rmse_train = feat_import_details['rmse_train']
        rmse_val = feat_import_details['rmse_val']
        exp_dict = {'alpha':alpha, 'rmse_train':rmse_train, 'rmse_val':rmse_val}
        exp_alphas.append(exp_dict)
        print("++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++")
        print('')
```

```
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Running experiment exp_alpha_0.01 for alpha: 0.0100
Starting experiment exp_alpha_0.01 at 11:27:37
in modelofcartransform, X_w_one_hots shape (8211, 150)
Running fit
in modelofcartransform, X_w_one_hots shape (8211, 150)
running predict for X_train
in modelofcartransform, X_w_one_hots shape (8211, 150)
running predict for X_val
in modelofcartransform, X_w_one_hots shape (2347, 150)
model predict rmse_train: 7,408.228998
model predict rmse_val: 8,147.955534
model predict rmse gap :739.726536
----
Calculating permutations to find feature importance per feature
in modelofcartransform, X_w_one_hots shape (2347, 150)
Feature columns with mean - 2*std GREATER THAN  0
year                                    113,485,587 +/- 3,705,786
type                                    25,931,821 +/- 1,121,473
manufacturer                            6,378,508 +/- 658,022
transmission                            5,762,228 +/- 630,945
model                                   3,718,086 +/- 648,053
fuel                                    3,264,545 +/- 448,031
region                                  700,210 +/- 277,782
----
RMSE train: 7408.228997844578
RMSE val: 8147.955533790506
finished experiment exp_alpha_0.01 in elapsed_time: 62.21104145050049
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Running experiment exp_alpha_0.1 for alpha: 0.1000
Starting experiment exp_alpha_0.1 at 11:28:39
in modelofcartransform, X_w_one_hots shape (8211, 150)
Running fit
in modelofcartransform, X_w_one_hots shape (8211, 150)
running predict for X_train
in modelofcartransform, X_w_one_hots shape (8211, 150)
running predict for X_val
in modelofcartransform, X_w_one_hots shape (2347, 150)
model predict rmse_train: 7,422.347835
model predict rmse_val: 8,150.013573
model predict rmse gap :727.665738
----
Calculating permutations to find feature importance per feature
in modelofcartransform, X_w_one_hots shape (2347, 150)
Feature columns with mean - 2*std GREATER THAN  0
year                                    112,302,323 +/- 3,685,962
type                                    25,874,226 +/- 1,179,321
manufacturer                            6,292,446 +/- 699,456
transmission                            5,711,725 +/- 555,026
model                                   4,061,660 +/- 768,926
fuel                                    3,213,066 +/- 425,707
region                                  617,309 +/- 267,165
----
RMSE train: 7422.347835115877
```

```
RMSE val: 8150.013573419533
finished experiment exp_alpha_0.1 in elapsed_time: 70.8969988822937
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Running experiment exp_alpha_1 for alpha: 1.0000
Starting experiment exp_alpha_1 at 11:29:50
in modelofcartransform, X_w_one_hots shape (8211, 150)
Running fit
in modelofcartransform, X_w_one_hots shape (8211, 150)
running predict for X_train
in modelofcartransform, X_w_one_hots shape (8211, 150)
running predict for X_val
in modelofcartransform, X_w_one_hots shape (2347, 150)
model predict rmse_train: 7,704.039470
model predict rmse_val: 8,330.003361
model predict rmse gap :625.963891
----
Calculating permutations to find feature importance per feature
in modelofcartransform, X_w_one_hots shape (2347, 150)
Feature columns with mean - 2*std GREATER THAN  0
year                                        113,045,450 +/- 3,764,349
type                                        23,752,975 +/- 1,019,231
manufacturer                                6,390,203 +/- 604,579
transmission                                5,718,016 +/- 558,247
fuel                                        3,143,174 +/- 353,003
region                                      826,790 +/- 202,642
----
RMSE train: 7704.039470143797
RMSE val: 8330.00336098477
finished experiment exp_alpha_1 in elapsed_time: 71.86946868896484
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Running experiment exp_alpha_10.0 for alpha: 10.0000
Starting experiment exp_alpha_10.0 at 11:31:02
in modelofcartransform, X_w_one_hots shape (8211, 150)
Running fit
in modelofcartransform, X_w_one_hots shape (8211, 150)
running predict for X_train
in modelofcartransform, X_w_one_hots shape (8211, 150)
running predict for X_val
in modelofcartransform, X_w_one_hots shape (2347, 150)
model predict rmse_train: 7,619.533688
model predict rmse_val: 8,162.373130
model predict rmse gap :542.839442
----
Calculating permutations to find feature importance per feature
in modelofcartransform, X_w_one_hots shape (2347, 150)
Feature columns with mean - 2*std GREATER THAN  0
year                                        111,030,269 +/- 3,259,307
type                                        25,200,930 +/- 1,006,213
manufacturer                                6,384,177 +/- 573,075
transmission                                5,945,673 +/- 526,782
fuel                                        3,230,748 +/- 399,501
region                                      829,856 +/- 218,088
```

```
----
RMSE train: 7619.533687924433
RMSE val: 8162.37313033349
finished experiment exp_alpha_10.0 in elapsed_time: 79.42385840415955
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Running experiment exp_alpha_100.0 for alpha: 100.0000
Starting experiment exp_alpha_100.0 at 11:32:21
in modelofcartransform, X_w_one_hots shape (8211, 150)
Running fit
in modelofcartransform, X_w_one_hots shape (8211, 150)
running predict for X_train
in modelofcartransform, X_w_one_hots shape (8211, 150)
running predict for X_val
in modelofcartransform, X_w_one_hots shape (2347, 150)
model predict rmse_train: 7,481.094412
model predict rmse_val: 8,253.213902
model predict rmse gap :772.119490
----
Calculating permutations to find feature importance per feature
in modelofcartransform, X_w_one_hots shape (2347, 150)
Feature columns with mean - 2*std GREATER THAN  0
year                                    105,139,525 +/- 2,955,264
type                                    22,049,538 +/- 1,069,369
transmission                            7,226,669 +/- 517,655
manufacturer                            5,614,027 +/- 461,409
fuel                                    3,826,443 +/- 427,102
region                                  1,040,212 +/- 241,561
paint_color                             754,551 +/- 216,783
----
RMSE train: 7481.094412369809
RMSE val: 8253.213902197647
finished experiment exp_alpha_100.0 in elapsed_time: 87.34238719940186
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Running experiment exp_alpha_1000.0 for alpha: 1,000.0000
Starting experiment exp_alpha_1000.0 at 11:33:49
in modelofcartransform, X_w_one_hots shape (8211, 150)
Running fit
in modelofcartransform, X_w_one_hots shape (8211, 150)
running predict for X_train
in modelofcartransform, X_w_one_hots shape (8211, 150)
running predict for X_val
in modelofcartransform, X_w_one_hots shape (2347, 150)
model predict rmse_train: 8,112.179100
model predict rmse_val: 8,786.227117
model predict rmse gap :674.048017
----
Calculating permutations to find feature importance per feature
in modelofcartransform, X_w_one_hots shape (2347, 150)
Feature columns with mean - 2*std GREATER THAN  0
year                                    79,022,866 +/- 2,386,530
type                                    11,614,875 +/- 661,155
transmission                            10,385,351 +/- 672,046
```

```
fuel                                        3,686,017 +/- 476,378
manufacturer                                2,470,770 +/- 366,094
paint_color                                 868,502 +/- 345,581
----
RMSE train: 8112.179100318709
RMSE val: 8786.227117194867
finished experiment exp_alpha_1000.0 in elapsed_time: 92.93063497543335
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

In [55]:

```
details
```

```
Out[55]: {'alpha': 1,
          'best_score': -56765780.64975144,
          'best_model': Pipeline(steps=[('preprocessor',
                           ColumnTransformer(transformers=[('modelofcartransformer',
                                                            ModelofCarTransformer(column_na
mes=['model',

'manufacturer',

'type'],
                                                                                  min_occur
rence=328,
                                                                                  stop_word
s=['i',

'me',

'my',

'myself',

'we',

'our',

'ours',

'ourselves',

'you',

"you're",

"you've",

"you'll",

"you'd",

'your',

'yours',

'yourself',

'yourselves',

'he',

'him',

'his',

'himself',

'sh...
```

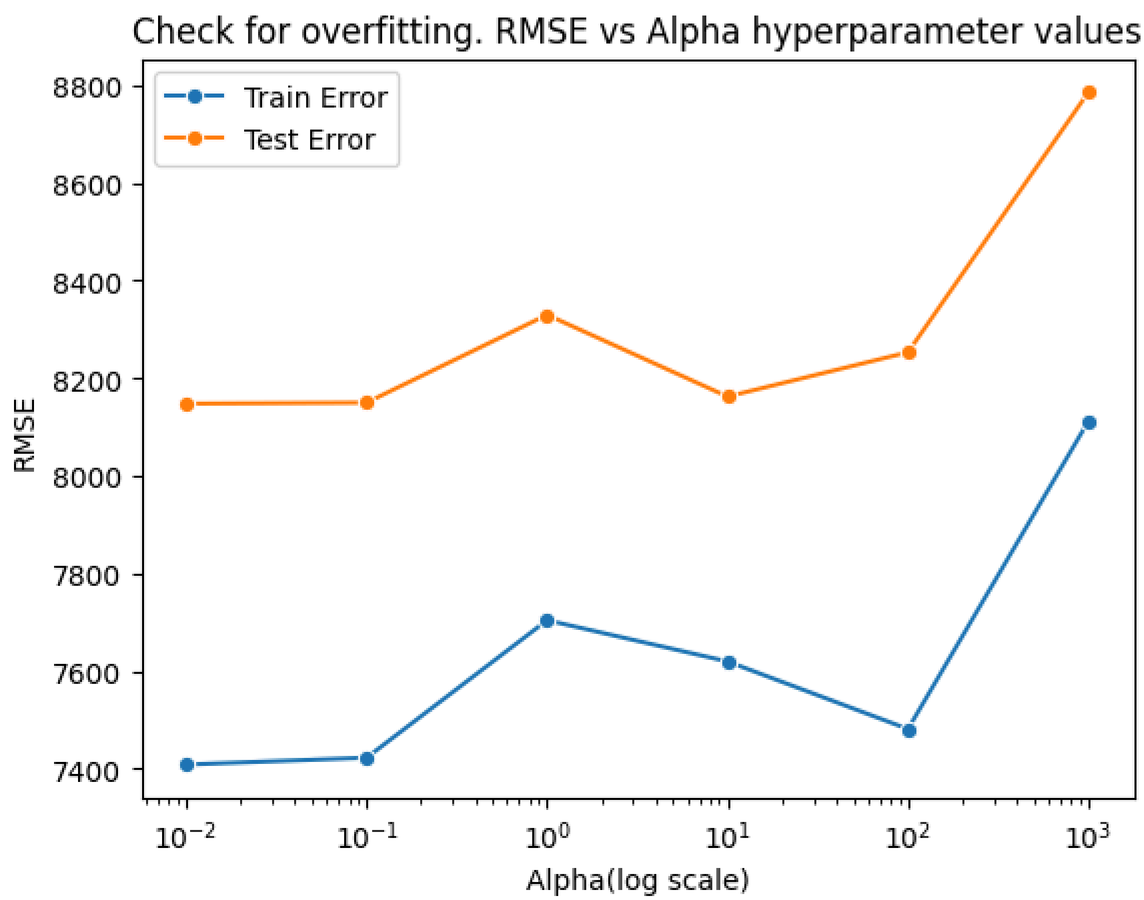
```
                                                                   ('poly',
                                                                    PolynomialFeat
ures(degree=4))]),
                                                     ['year']),
                                                    ('onehotencoder',
                                                     OneHotEncoder(drop='first',
                                                                   handle_unknown='i
gnore',
                                                                   sparse_output=Fal
se),
                                                     ['type', 'state',
                                                      'manufacturer', 'fuel',
                                                      'title_status',
                                                      'transmission',
                                                      'paint_color',
                                                      'region'])])),
                 ('selector',
                  SelectFromModel(estimator=Lasso(alpha=100, max_iter=3000))),
                 ('regressor', Ridge(alpha=1, max_iter=1000))]),
 'feature_names_in': ['type',
  'state',
  'manufacturer',
  'fuel',
  'title_status',
  'transmission',
  'paint_color',
  'region',
  'year'],
 'feat_names_preprocessor': array(['modelofcartransformer__my_one_hot_col_1500',
        'modelofcartransformer__my_one_hot_col_4d',
        'modelofcartransformer__my_one_hot_col_cab',
        'modelofcartransformer__my_one_hot_col_sport', 'pipeline__1',
        'pipeline__year', 'pipeline__year^2', 'pipeline__year^3',
        'pipeline__year^4', 'onehotencoder__type_bus',
        'onehotencoder__type_convertible', 'onehotencoder__type_coupe',
        'onehotencoder__type_hatchback', 'onehotencoder__type_mini-van',
        'onehotencoder__type_offroad', 'onehotencoder__type_other',
        'onehotencoder__type_pickup', 'onehotencoder__type_sedan',
        'onehotencoder__type_truck', 'onehotencoder__type_van',
        'onehotencoder__type_wagon',
        'onehotencoder__manufacturer_alfa-romeo',
        'onehotencoder__manufacturer_audi',
        'onehotencoder__manufacturer_bmw',
        'onehotencoder__manufacturer_buick',
        'onehotencoder__manufacturer_cadillac',
        'onehotencoder__manufacturer_chevrolet',
        'onehotencoder__manufacturer_chrysler',
        'onehotencoder__manufacturer_dodge',
        'onehotencoder__manufacturer_fiat',
        'onehotencoder__manufacturer_ford',
        'onehotencoder__manufacturer_gmc',
        'onehotencoder__manufacturer_honda',
        'onehotencoder__manufacturer_hyundai',
        'onehotencoder__manufacturer_infiniti',
        'onehotencoder__manufacturer_jaguar',
        'onehotencoder__manufacturer_jeep',
```

```
'onehotencoder__manufacturer_kia',
'onehotencoder__manufacturer_lexus',
'onehotencoder__manufacturer_lincoln',
'onehotencoder__manufacturer_mazda',
'onehotencoder__manufacturer_mercedes-benz',
'onehotencoder__manufacturer_mercury',
'onehotencoder__manufacturer_mini',
'onehotencoder__manufacturer_mitsubishi',
'onehotencoder__manufacturer_nissan',
'onehotencoder__manufacturer_pontiac',
'onehotencoder__manufacturer_porsche',
'onehotencoder__manufacturer_ram',
'onehotencoder__manufacturer_rover',
'onehotencoder__manufacturer_saturn',
'onehotencoder__manufacturer_subaru',
'onehotencoder__manufacturer_tesla',
'onehotencoder__manufacturer_toyota',
'onehotencoder__manufacturer_volkswagen',
'onehotencoder__manufacturer_volvo',
'onehotencoder__fuel_electric', 'onehotencoder__fuel_gas',
'onehotencoder__fuel_hybrid', 'onehotencoder__fuel_other',
'onehotencoder__title_status_lien',
'onehotencoder__title_status_missing',
'onehotencoder__title_status_parts only',
'onehotencoder__title_status_rebuilt',
'onehotencoder__title_status_salvage',
'onehotencoder__transmission_manual',
'onehotencoder__transmission_other',
'onehotencoder__paint_color_blue',
'onehotencoder__paint_color_brown',
'onehotencoder__paint_color_custom',
'onehotencoder__paint_color_green',
'onehotencoder__paint_color_grey',
'onehotencoder__paint_color_orange',
'onehotencoder__paint_color_purple',
'onehotencoder__paint_color_red',
'onehotencoder__paint_color_silver',
'onehotencoder__paint_color_white',
'onehotencoder__paint_color_yellow',
'onehotencoder__region_binghamton',
'onehotencoder__region_buffalo', 'onehotencoder__region_catskills',
'onehotencoder__region_chautauqua',
'onehotencoder__region_elmira-corning',
'onehotencoder__region_finger lakes',
'onehotencoder__region_glens falls',
'onehotencoder__region_hudson valley',
'onehotencoder__region_ithaca',
'onehotencoder__region_long island',
'onehotencoder__region_new york city',
'onehotencoder__region_oneonta',
'onehotencoder__region_plattsburgh-adirondacks',
'onehotencoder__region_potsdam-canton-massena',
'onehotencoder__region_rochester',
'onehotencoder__region_syracuse',
'onehotencoder__region_twin tiers NY/PA',
'onehotencoder__region_utica-rome-oneida',
```

```
        'onehotencoder__region_watertown'], dtype=object),
 'feat_names_selector': array(['x3', 'x5', 'x6', 'x7', 'x8', 'x10', 'x11', 'x12',
'x15', 'x16',
        'x17', 'x18', 'x20', 'x23', 'x26', 'x30', 'x32', 'x33', 'x37',
        'x41', 'x45', 'x54', 'x57', 'x59', 'x66', 'x76', 'x87', 'x88'],
       dtype=object),
 'mse_train': 55615374.05170851,
 'mse_val': 58085150.30317131,
 'rmse_train': 7457.571592127595,
 'rmse_val': 7621.361446826368}
```

In [56]:
```python
def overfit_plot_check(df, x, y_train,y_test, xlabel, ylabel, xlog=False):

    # Create the lineplot
    sns.lineplot(df,x=x,y=y_train, label='Train Error', marker='o')
    sns.lineplot(df, x=x, y=y_test, label='Test Error', marker='o')

    # Customize plot (optional)
    plt.legend()  # Add a legend
    if xlog:
        xlabel = xlabel + '(log scale)'
        plt.xscale('log')

    # Add labels and title
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.title('Check for overfitting. RMSE vs Alpha hyperparameter values')
    # Show the plot
    plt.show()
    plt.cla()
    plt.clf()
```

In [57]:
```python
if OK_TO_RUN_FEAT_IMPORT_ADDITIONAL:
    df = pd.DataFrame(exp_alphas)
    overfit_plot_check(df, 'alpha', 'rmse_train', 'rmse_val', 'Alpha', 'RMSE', xlog
```




```
<Figure size 640x480 with 0 Axes>
```

In [58]:
```
if OK_TO_RUN_FEAT_IMPORT_ADDITIONAL:
    # minimumn occurrence settings as a function of the percent size of the X_train
    min_occurrences = [max(2, round((i/100)*X_train.shape[0]))for i in [0.05,0.5,1,
    min_occurrences = list(set(min_occurrences))

    print(min_occurrences)
```

```
[164, 4, 2053, 41, 4106, 82, 821, 246, 411]
```

In [59]:
```
OK_TO_RUN_FEAT_IMPORT = True
# run for several min_occurence values and chart rmse
min_occurrences = [max(2, round((i/100)*X_train.shape[0]))for i in [0.25,0.5,1,2,3,
min_occurrences = list(set(min_occurrences))
print(f"Running experiments for this range of minimum occurrences: {min_occurrences
exp_min_o = []
if OK_TO_RUN_FEAT_IMPORT:
    for min_occurrence in min_occurrences:
        xp_id = "exp_min_occurence_"+f"{min_occurrence}"
        print("++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++")
        print(f"Running experiment {xp_id} for min_occurrence: {min_occurrence:,.4f
        feat_import_details = run_feat_import_experiment(categorical_cols, numerica
                                                         X_train, y_train, X_test, y_te
        rmse_train = feat_import_details['rmse_train']
        rmse_val = feat_import_details['rmse_val']
        mean_change_rmse_for_model_field = feat_import_details['mean_change_rmse_fo
        std_change_rmse_for_model_field = feat_import_details['std_change_rmse_for_
```

```
        exp_dict = {'min_occurrence':min_occurrence, 'rmse_train':rmse_train, 'rmse
                    'mean_change_rmse_for_model_field':mean_change_rmse_for_model_f
                    'std_change_rmse_for_model_field':std_change_rmse_for_model_fie
                    }
        exp_min_o.append(exp_dict)
        print("+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++")
        print('')
```

```
Running experiments for this range of minimum occurrences: [164, 2053, 41, 82, 21, 2
46, 821, 411]
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Running experiment exp_min_occurrence_164 for min_occurrence: 164.0000
Starting experiment exp_min_occurrence_164 at 11:35:22
in modelofcartransform, X_w_one_hots shape (8211, 15)
Running fit
in modelofcartransform, X_w_one_hots shape (8211, 15)
running predict for X_train
in modelofcartransform, X_w_one_hots shape (8211, 15)
running predict for X_val
in modelofcartransform, X_w_one_hots shape (2347, 15)
model predict rmse_train: 7,512.397226
model predict rmse_val: 7,799.820148
model predict rmse gap :287.422922
----
Calculating permutations to find feature importance per feature
in modelofcartransform, X_w_one_hots shape (2347, 15)
Feature columns with mean - 2*std GREATER THAN  0
year                                    112,401,577 +/- 2,994,170
type                                    26,323,560 +/- 1,319,132
manufacturer                            8,391,626 +/- 677,710
transmission                            6,751,405 +/- 482,734
fuel                                    5,300,083 +/- 491,769
region                                  1,576,595 +/- 199,236
paint_color                             291,984 +/- 81,337
----
RMSE train: 7512.397226207319
RMSE val: 7799.820147832804
finished experiment exp_min_occurrence_164 in elapsed_time: 26.27104425430298
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Running experiment exp_min_occurrence_2053 for min_occurrence: 2,053.0000
Starting experiment exp_min_occurrence_2053 at 11:35:49
in modelofcartransform, X_w_one_hots shape (8211, 0)
Running fit
in modelofcartransform, X_w_one_hots shape (8211, 0)
running predict for X_train
in modelofcartransform, X_w_one_hots shape (8211, 0)
running predict for X_val
in modelofcartransform, X_w_one_hots shape (2347, 0)
model predict rmse_train: 7,483.275678
model predict rmse_val: 7,760.929004
model predict rmse gap :277.653326
----
Calculating permutations to find feature importance per feature
in modelofcartransform, X_w_one_hots shape (2347, 0)
Feature columns with mean - 2*std GREATER THAN  0
year                                    112,599,385 +/- 3,159,113
type                                    27,005,164 +/- 1,300,490
manufacturer                            8,015,204 +/- 633,968
transmission                            6,904,808 +/- 482,593
fuel                                    5,392,888 +/- 534,215
region                                  1,683,133 +/- 227,493
paint_color                             320,204 +/- 98,501
```

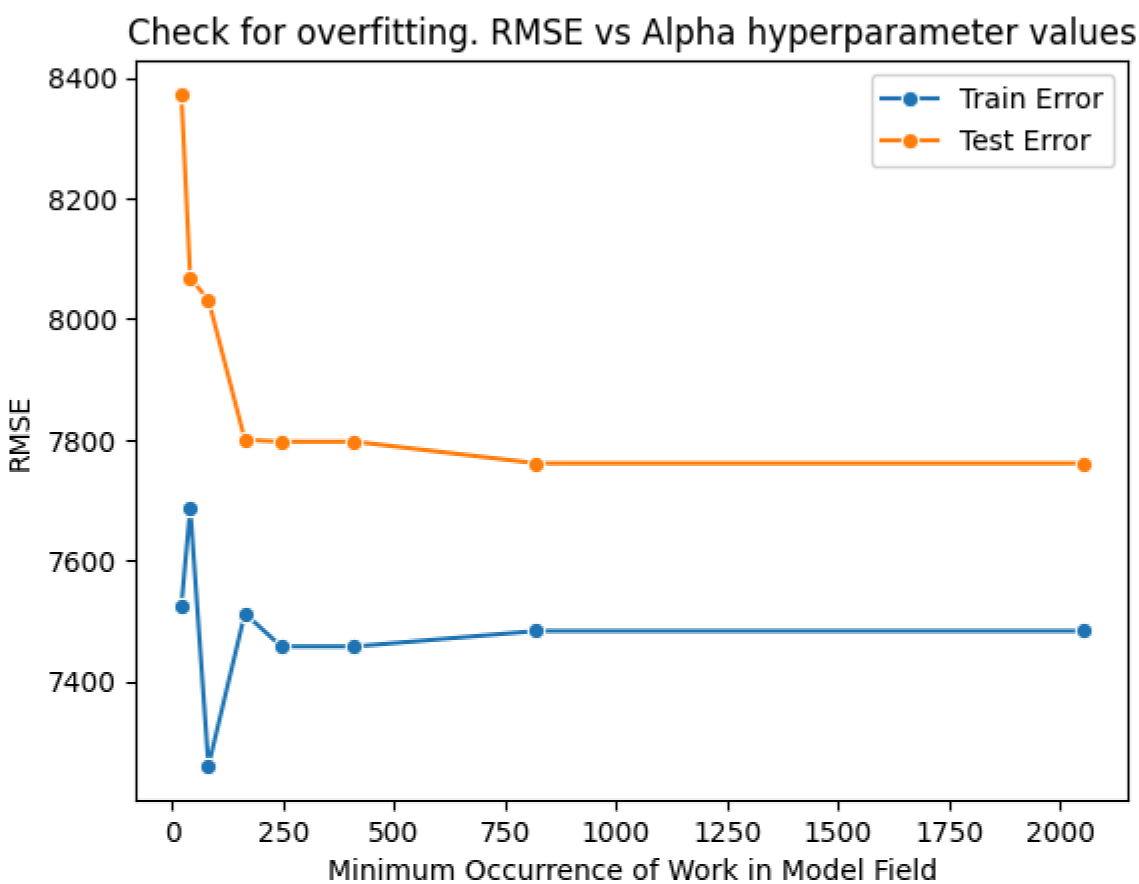
```
----
RMSE train: 7483.275678113098
RMSE val: 7760.929004415661
finished experiment exp_min_occurrence_2053 in elapsed_time: 14.798235416412354
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Running experiment exp_min_occurrence_41 for min_occurrence: 41.0000
Starting experiment exp_min_occurrence_41 at 11:36:03
in modelofcartransform, X_w_one_hots shape (8211, 106)
Running fit
in modelofcartransform, X_w_one_hots shape (8211, 106)
running predict for X_train
in modelofcartransform, X_w_one_hots shape (8211, 106)
running predict for X_val
in modelofcartransform, X_w_one_hots shape (2347, 106)
model predict rmse_train: 7,686.387667
model predict rmse_val: 8,067.523713
model predict rmse gap :381.136046
----
Calculating permutations to find feature importance per feature
in modelofcartransform, X_w_one_hots shape (2347, 106)
Feature columns with mean - 2*std GREATER THAN  0
year                                    110,460,439 +/- 3,784,489
type                                    25,509,047 +/- 1,277,455
manufacturer                            6,461,694 +/- 570,608
transmission                            5,940,636 +/- 756,363
fuel                                    3,314,622 +/- 477,175
model                                   1,275,032 +/- 279,448
region                                  1,068,649 +/- 363,539
----
RMSE train: 7686.387667330918
RMSE val: 8067.523713475712
finished experiment exp_min_occurrence_41 in elapsed_time: 59.44782042503357
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Running experiment exp_min_occurrence_82 for min_occurrence: 82.0000
Starting experiment exp_min_occurrence_82 at 11:37:03
in modelofcartransform, X_w_one_hots shape (8211, 40)
Running fit
in modelofcartransform, X_w_one_hots shape (8211, 40)
running predict for X_train
in modelofcartransform, X_w_one_hots shape (8211, 40)
running predict for X_val
in modelofcartransform, X_w_one_hots shape (2347, 40)
model predict rmse_train: 7,259.910468
model predict rmse_val: 8,031.839188
model predict rmse gap :771.928720
----
Calculating permutations to find feature importance per feature
in modelofcartransform, X_w_one_hots shape (2347, 40)
Feature columns with mean - 2*std GREATER THAN  0
year                                    109,271,004 +/- 3,389,685
type                                    23,702,707 +/- 1,327,905
manufacturer                            6,949,793 +/- 634,631
```

```
transmission                            4,210,017 +/- 647,273
fuel                                    3,085,435 +/- 512,088
region                                  1,606,270 +/- 259,959
paint_color                             415,189 +/- 200,159
----
RMSE train: 7259.910468231124
RMSE val: 8031.839188477242
finished experiment exp_min_occurrence_82 in elapsed_time: 28.711049795150757
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Running experiment exp_min_occurrence_21 for min_occurrence: 21.0000
Starting experiment exp_min_occurrence_21 at 11:37:32
in modelofcartransform, X_w_one_hots shape (8211, 196)
Running fit
in modelofcartransform, X_w_one_hots shape (8211, 196)
running predict for X_train
in modelofcartransform, X_w_one_hots shape (8211, 196)
running predict for X_val
in modelofcartransform, X_w_one_hots shape (2347, 196)
model predict rmse_train: 7,525.756904
model predict rmse_val: 8,372.709264
model predict rmse gap :846.952360
----
Calculating permutations to find feature importance per feature
in modelofcartransform, X_w_one_hots shape (2347, 196)
Feature columns with mean - 2*std GREATER THAN  0
year                                    110,992,403 +/- 3,664,754
type                                    23,844,417 +/- 1,091,864
manufacturer                            6,504,234 +/- 618,215
transmission                            5,508,469 +/- 597,672
fuel                                    3,324,313 +/- 385,836
region                                  922,696 +/- 170,878
paint_color                             363,581 +/- 104,998
----
RMSE train: 7525.756903776145
RMSE val: 8372.709263538922
finished experiment exp_min_occurrence_21 in elapsed_time: 107.94560551643372
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Running experiment exp_min_occurrence_246 for min_occurrence: 246.0000
Starting experiment exp_min_occurrence_246 at 11:39:20
in modelofcartransform, X_w_one_hots shape (8211, 7)
Running fit
in modelofcartransform, X_w_one_hots shape (8211, 7)
running predict for X_train
in modelofcartransform, X_w_one_hots shape (8211, 7)
running predict for X_val
in modelofcartransform, X_w_one_hots shape (2347, 7)
model predict rmse_train: 7,457.571592
model predict rmse_val: 7,796.765256
model predict rmse gap :339.193664
----
Calculating permutations to find feature importance per feature
in modelofcartransform, X_w_one_hots shape (2347, 7)
```

```
Feature columns with mean - 2*std GREATER THAN  0
year                                    112,450,768 +/- 2,989,979
type                                    27,025,856 +/- 1,299,315
manufacturer                            8,074,959 +/- 702,662
transmission                            6,422,223 +/- 471,019
fuel                                    5,251,939 +/- 549,080
region                                  1,583,084 +/- 205,452
paint_color                             292,231 +/- 118,980
----
RMSE train: 7457.571592127595
RMSE val: 7796.765256108857
finished experiment exp_min_occurrence_246 in elapsed_time: 15.390422821044922
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Running experiment exp_min_occurrence_821 for min_occurrence: 821.0000
Starting experiment exp_min_occurrence_821 at 11:39:35
in modelofcartransform, X_w_one_hots shape (8211, 0)
Running fit
in modelofcartransform, X_w_one_hots shape (8211, 0)
running predict for X_train
in modelofcartransform, X_w_one_hots shape (8211, 0)
running predict for X_val
in modelofcartransform, X_w_one_hots shape (2347, 0)
model predict rmse_train: 7,483.275678
model predict rmse_val: 7,760.929004
model predict rmse gap :277.653326
----
Calculating permutations to find feature importance per feature
in modelofcartransform, X_w_one_hots shape (2347, 0)
Feature columns with mean - 2*std GREATER THAN  0
year                                    113,005,273 +/- 3,314,237
type                                    26,850,918 +/- 1,214,546
manufacturer                            7,893,209 +/- 704,248
transmission                            6,887,823 +/- 531,629
fuel                                    5,295,006 +/- 540,493
region                                  1,671,377 +/- 254,062
paint_color                             286,800 +/- 96,756
----
RMSE train: 7483.275678113098
RMSE val: 7760.929004415661
finished experiment exp_min_occurrence_821 in elapsed_time: 13.882542371749878
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Running experiment exp_min_occurrence_411 for min_occurrence: 411.0000
Starting experiment exp_min_occurrence_411 at 11:39:49
in modelofcartransform, X_w_one_hots shape (8211, 4)
Running fit
in modelofcartransform, X_w_one_hots shape (8211, 4)
running predict for X_train
in modelofcartransform, X_w_one_hots shape (8211, 4)
running predict for X_val
in modelofcartransform, X_w_one_hots shape (2347, 4)
model predict rmse_train: 7,457.571592
model predict rmse_val: 7,796.765256
```

```
model predict rmse gap :339.193664
----
Calculating permutations to find feature importance per feature
in modelofcartransform, X_w_one_hots shape (2347, 4)
Feature columns with mean - 2*std GREATER THAN  0
year                                        113,200,131 +/- 3,303,432
type                                        26,959,129 +/- 1,246,513
manufacturer                                8,163,324 +/- 780,536
transmission                                6,464,254 +/- 464,755
fuel                                        5,365,895 +/- 580,427
region                                      1,592,793 +/- 207,028
paint_color                                 307,807 +/- 107,357
----
RMSE train: 7457.571592127595
RMSE val: 7796.765256108857
finished experiment exp_min_occurrence_411 in elapsed_time: 14.69137454032898
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

In [60]:
```python
if OK_TO_RUN_FEAT_IMPORT_ADDITIONAL:
    df = pd.DataFrame(exp_min_o)
    overfit_plot_check(df, 'min_occurrence', 'rmse_train', 'rmse_val', 'Minimum Occ
```

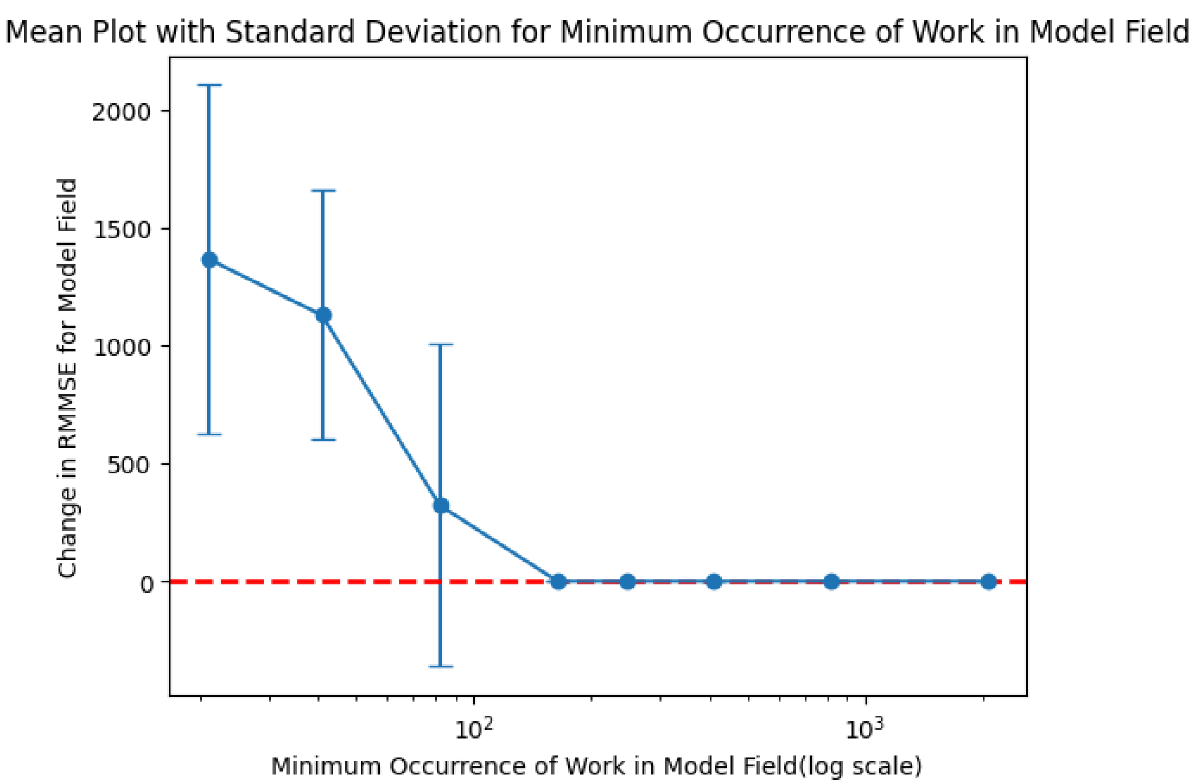


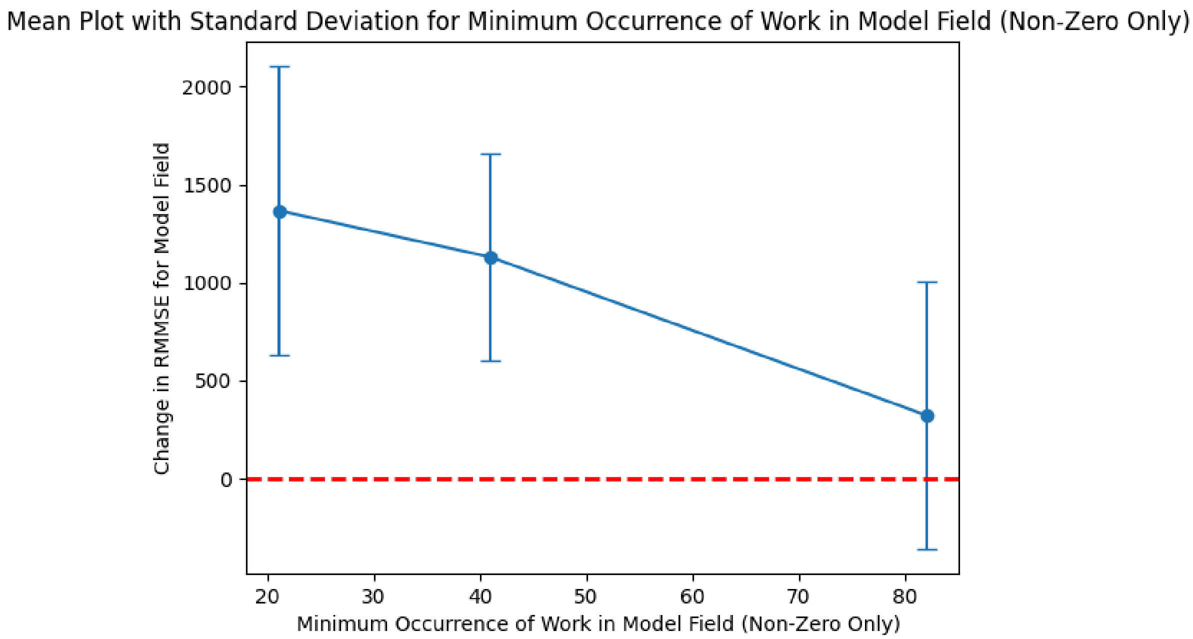

```
<Figure size 640x480 with 0 Axes>
```

In [61]:
```python
def feat_import_check_plot_w_std(x, y_mean, y_err, xlabel, ylabel, xlog=False):

    # Create the error bar plot
    plt.errorbar(x, y_mean, yerr=y_err, fmt='o-', capsize=5)
```

```
    # Add labels and title
    plt.title(f'Mean Plot with Standard Deviation for {xlabel}')

    # Customize plot (optional)
    if xlog:
        xlabel = xlabel + '(log scale)'
        plt.xscale('log')

    # Add labels and title
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)

    plt.axhline(y=0, color='red', linestyle='--', linewidth=2)  # Adjust styles as

    # Show the plot
    plt.show()
    plt.cla()
    plt.clf()
```

In [62]:
```
if OK_TO_RUN_FEAT_IMPORT_ADDITIONAL:
    df.fillna(0, inplace=True)
    df = df.sort_values(by='min_occurrence', ascending=True)
    feat_import_check_plot_w_std(df['min_occurrence'], df['mean_change_rmse_for_mod
    df_non_zero = df[df['mean_change_rmse_for_model_field'] != 0]
    feat_import_check_plot_w_std(df_non_zero['min_occurrence'], df_non_zero['mean_c
```






```
<Figure size 640x480 with 0 Axes>
```

In [63]:
```
beep()
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: