Comparative Analysis of Machine Learning Models for Used Car Price Estimation

Ritesh KC

May 29, 2025

1 Import libraries

[1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
sns.set(style="whitegrid")
import matplotlib.pyplot as plt
plt.style.use('seaborn-v0_8-whitegrid')

pd.options.display.max_columns = None
pd.set_option('display.max_rows', 40)
```

2 Import Data

The dataset I am trying to import here is substantial. Upon reviewing the data, I have identified the specific column required for this analysis. The majority of the data comes from the years 2016 through 2021. Therefore, I have chosen to focus on the data from the last five years.

[2]:
```
cols = ['vin', 'city', 'back_legroom','body_type', 'city_fuel_economy',␣
 ↪'highway_fuel_economy',
       'daysonmarket',␣
 ↪'engine_cylinders','engine_displacement','fuel_tank_volume',
       'fuel_type','horsepower','is_new',␣
 ↪'listing_color','make_name','franchise_dealer',
       'franchise_make','front_legroom', 'height','length',␣
 ↪'mileage','model_name','power','torque',
      ␣
 ↪'trim_name','wheel_system','wheelbase','listed_date','maximum_seating','price',␣
 ↪'savings_amount',
       'seller_rating','transmission', 'year']
```

Due to the substantial size of the dataset, which contains over 2 million rows and multiple columns, I utilized the `chunksize` function to extract the data in segments and subsequently appended these segments to create a filtered DataFrame.

[3]:
```
chunksize = 10000
filtered_chunks =[]

for chunk in pd.read_csv('archive.zip', usecols= cols, chunksize=chunksize):
    filtered_chunk = chunk[chunk['year'] > 2016]
    filtered_chunks.append(filtered_chunk)

filtered_df = pd.concat(filtered_chunks)
```

[4]:
```
filtered_df.head()
```

[4]:

```
                vin back_legroom        body_type      city  \
0  ZACNJABB5KPJ92081      35.1 in  SUV / Crossover   Bayamon
1  SALCJ2FX1LH858117      38.1 in  SUV / Crossover  San Juan
3  SALRR2RV0L2433391      37.6 in  SUV / Crossover  San Juan
4  SALCJ2FXXLH862327      38.1 in  SUV / Crossover  San Juan
5  SALYK2EX1LA261711      37.1 in  SUV / Crossover  San Juan

   city_fuel_economy  daysonmarket engine_cylinders  engine_displacement  \
0                NaN           522               I4               1300.0
1                NaN           207               I4               2000.0
3                NaN           196               V6               3000.0
4                NaN           137               I4               2000.0
5                NaN           242               I4               2000.0

   franchise_dealer franchise_make front_legroom fuel_tank_volume fuel_type  \
0              True           Jeep       41.2 in         12.7 gal  Gasoline
1              True     Land Rover       39.1 in         17.7 gal  Gasoline
3              True     Land Rover         39 in         23.5 gal  Gasoline
4              True     Land Rover       39.1 in         17.7 gal  Gasoline
5              True     Land Rover       40.2 in         16.6 gal  Gasoline

    height  highway_fuel_economy  horsepower  is_new    length listed_date  \
0  66.5 in                   NaN       177.0    True  166.6 in  2019-04-06
1    68 in                   NaN       246.0    True    181 in  2020-02-15
3    73 in                   NaN       340.0    True  195.1 in  2020-02-26
4    68 in                   NaN       246.0    True    181 in  2020-04-25
5  66.3 in                   NaN       247.0    True  188.9 in  2020-01-11

  listing_color   make_name maximum_seating  mileage         model_name  \
0        YELLOW        Jeep         5 seats      7.0           Renegade
1         BLACK  Land Rover         7 seats      8.0    Discovery Sport
3          GRAY  Land Rover         7 seats     11.0          Discovery
4         BLACK  Land Rover         7 seats      7.0    Discovery Sport
5       UNKNOWN  Land Rover         5 seats     12.0  Range Rover Velar

               power    price  savings_amount  seller_rating  \
```

```
0  177 hp @ 5,750 RPM  23141.0              0            2.8
1  246 hp @ 5,500 RPM  46500.0              0            3.0
3  340 hp @ 6,500 RPM  67430.0              0            3.0
4  246 hp @ 5,500 RPM  48880.0              0            3.0
5  247 hp @ 5,500 RPM  66903.0              0            3.0

                  torque transmission             trim_name wheel_system  \
0  200 lb-ft @ 1,750 RPM            A          Latitude FWD          FWD
1  269 lb-ft @ 1,400 RPM            A                S AWD          AWD
3  332 lb-ft @ 3,500 RPM            A            V6 HSE AWD          AWD
4  269 lb-ft @ 1,400 RPM            A                S AWD          AWD
5  269 lb-ft @ 1,200 RPM            A  P250 R-Dynamic S AWD          AWD

  wheelbase  year
0  101.2 in  2019
1  107.9 in  2020
3    115 in  2020
4  107.9 in  2020
5  113.1 in  2020
```

[5]:
```
filtered_df.to_parquet("selected_car_dataframe.parquet", index=False) # Saving␣
↪the in parquet format.
```

3 Data Preprocessing

[6]:
```
filtered_df.shape # Check data dimention
```

[6]:
```
(2369450, 34)
```

[7]:
```
# Check for the missingness in each column
filtered_df.isnull().sum()
```

[7]:
```
vin                        0
back_legroom          120470
body_type              12375
city                       0
city_fuel_economy     363793
daysonmarket               0
engine_cylinders       80107
engine_displacement   137482
franchise_dealer           0
franchise_make        224514
front_legroom         120470
fuel_tank_volume      120470
fuel_type              64624
height                120470
```

```
highway_fuel_economy    363793
horsepower              137482
is_new                       0
length                  120470
listed_date                  0
listing_color                0
make_name                    0
maximum_seating         120470
mileage                 137737
model_name                   0
power                   422204
price                        0
savings_amount               0
seller_rating            24990
torque                  448098
transmission             52089
trim_name                91526
wheel_system            118168
wheelbase               120470
year                         0
dtype: int64
```

[8]:
```
# Check the missingness percentage in each column
filtered_df.isnull().sum()/filtered_df.shape[0]*100
```

[8]:
```
vin                      0.000000
back_legroom             5.084302
body_type                0.522273
city                     0.000000
city_fuel_economy       15.353479
daysonmarket             0.000000
engine_cylinders         3.380827
engine_displacement      5.802275
franchise_dealer         0.000000
franchise_make           9.475363
front_legroom            5.084302
fuel_tank_volume         5.084302
fuel_type                2.727384
height                   5.084302
highway_fuel_economy    15.353479
horsepower               5.802275
is_new                   0.000000
length                   5.084302
listed_date              0.000000
listing_color            0.000000
make_name                0.000000
maximum_seating          5.084302
```

```
mileage                 5.813037
model_name              0.000000
power                  17.818650
price                   0.000000
savings_amount          0.000000
seller_rating           1.054675
torque                 18.911477
transmission            2.198358
trim_name               3.862753
wheel_system            4.987149
wheelbase               5.084302
year                    0.000000
dtype: float64
```

It is evident that several columns in the data frames contain missing values, which require both attention and resolution.

[9]:
```
filtered_df.columns # Printing columns names
```

[9]:
```
Index(['vin', 'back_legroom', 'body_type', 'city', 'city_fuel_economy',
       'daysonmarket', 'engine_cylinders', 'engine_displacement',
       'franchise_dealer', 'franchise_make', 'front_legroom',
       'fuel_tank_volume', 'fuel_type', 'height', 'highway_fuel_economy',
       'horsepower', 'is_new', 'length', 'listed_date', 'listing_color',
       'make_name', 'maximum_seating', 'mileage', 'model_name', 'power',
       'price', 'savings_amount', 'seller_rating', 'torque', 'transmission',
       'trim_name', 'wheel_system', 'wheelbase', 'year'],
      dtype='object')
```

[10]:
```
# Since I have a large number of rows, I will drop anything with less than 20% NaN values
df = filtered_df.dropna(subset=['back_legroom', 'body_type', 'engine_cylinders', 'engine_displacement',
                        'front_legroom','fuel_tank_volume', 'fuel_type', 'height', 'horsepower',
                        'length', 'maximum_seating','mileage','seller_rating','transmission',
                        'trim_name', 'wheel_system', 'wheelbase'])
```

[11]:
```
# Check again after dropping
df.isnull().sum()/df.shape[0]*100
```

[11]:
```
vin                     0.000000
back_legroom            0.000000
body_type               0.000000
city                    0.000000
city_fuel_economy       9.425085
daysonmarket            0.000000
```

```
engine_cylinders          0.000000
engine_displacement       0.000000
franchise_dealer          0.000000
franchise_make           10.213078
front_legroom             0.000000
fuel_tank_volume          0.000000
fuel_type                 0.000000
height                    0.000000
highway_fuel_economy      9.425085
horsepower                0.000000
is_new                    0.000000
length                    0.000000
listed_date               0.000000
listing_color             0.000000
make_name                 0.000000
maximum_seating           0.000000
mileage                   0.000000
model_name                0.000000
power                    12.790016
price                     0.000000
savings_amount            0.000000
seller_rating             0.000000
torque                   13.993584
transmission              0.000000
trim_name                 0.000000
wheel_system              0.000000
wheelbase                 0.000000
year                      0.000000
dtype: float64
```

[12]:
```
df.shape
```

[12]:
```
(2020576, 34)
```

The df contains fewer rows than the original indexed df, which aligns with the expectation that several rows have been removed.

[13]:
```
df['power'].dtype
```

[13]:
```
dtype('O')
```

[14]:
```
# Lets split power into hp and rpm
df = df.copy() # otherwise SettingWithCopyWarning:

df[['hp', 'hp_rpm']] = df['power'].str.extract(r'(\d+)\s+hp\s+@\s+([\d,]+)',␣
 ↪expand=True)
df['hp_rpm'] = df['hp_rpm'].str.replace(',', '', regex=True).astype('Int64')
df['hp'] = df['hp'].astype('Int64')
```

[15]:
```
# Lets split torque into tor and tor_rpm
df = df.copy()

df[['tor', 'tor_rpm']] = df['torque'].str.
 ↪extract(r'(\d+)\s+lb-ft\s+@\s+([\d,]+)')
df['tor_rpm'] = df['tor_rpm'].str.replace(',', '', regex=True).astype('Int64')
df['tor'] = df['tor'].astype('Int64')
```

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

[16]:

	vin	back_legroom	body_type	city	city_fuel_economy	daysonmarket	engine_cylinders	engine_displacement	franchise_dealer	franchise_make	front_legroom	fuel_tank_volume	fuel_type	height	highway_fuel_economy	horsepower	is_new	length	listed_date	listing_color	make_name	maximum_seating	mileage	model_name	power	price	savings_amount	seller_rating
0	ZACNJABB5KPJ92081	35.1 in	SUV / Crossover	Bayamon	NaN	522	I4	1300.0	True	Jeep	41.2 in	12.7 gal	Gasoline	66.5 in	NaN	177.0	True	166.6 in	2019-04-06	YELLOW	Jeep	5 seats	7.0	Renegade	177 hp @ 5,750 RPM	23141.0	0	2.8
1	SALCJ2FX1LH858117	38.1 in	SUV / Crossover	San Juan	NaN	207	I4	2000.0	True	Land Rover	39.1 in	17.7 gal	Gasoline	68 in	NaN	246.0	True	181 in	2020-02-15	BLACK	Land Rover	7 seats	8.0	Discovery Sport				
3	SALRR2RV0L2433391	37.6 in	SUV / Crossover	San Juan	NaN	196	V6	3000.0	True	Land Rover	39 in	23.5 gal	Gasoline	73 in	NaN	340.0	True	195.1 in	2020-02-26	GRAY	Land Rover	7 seats	11.0	Discovery				
4	SALCJ2FXXLH862327	38.1 in	SUV / Crossover	San Juan	NaN	137	I4	2000.0	True	Land Rover	39.1 in	17.7 gal	Gasoline	68 in	NaN	246.0	True	181 in	2020-04-25	BLACK	Land Rover	7 seats	7.0	Discovery Sport				
5	SALYK2EX1LA261711	37.1 in	SUV / Crossover	San Juan	NaN	242	I4	2000.0	True	Land Rover	40.2 in	16.6 gal	Gasoline	66.3 in	NaN	247.0	True	188.9 in	2020-01-11	UNKNOWN	Land Rover	5 seats	12.0	Range Rover Velar				

```
1  246 hp @ 5,500 RPM  46500.0              0            3.0
3  340 hp @ 6,500 RPM  67430.0              0            3.0
4  246 hp @ 5,500 RPM  48880.0              0            3.0
5  247 hp @ 5,500 RPM  66903.0              0            3.0

                  torque transmission            trim_name wheel_system  \
0  200 lb-ft @ 1,750 RPM            A          Latitude FWD          FWD
1  269 lb-ft @ 1,400 RPM            A                S AWD          AWD
3  332 lb-ft @ 3,500 RPM            A            V6 HSE AWD          AWD
4  269 lb-ft @ 1,400 RPM            A                S AWD          AWD
5  269 lb-ft @ 1,200 RPM            A  P250 R-Dynamic S AWD          AWD

  wheelbase  year   hp  hp_rpm  tor  tor_rpm
0  101.2 in  2019  177    5750  200     1750
1  107.9 in  2020  246    5500  269     1400
3    115 in  2020  340    6500  332     3500
4  107.9 in  2020  246    5500  269     1400
5  113.1 in  2020  247    5500  269     1200
```

[17]:
```
df[['hp', 'hp_rpm','tor', 'tor_rpm']].dtypes
```

[17]:
```
hp         Int64
hp_rpm     Int64
tor        Int64
tor_rpm    Int64
dtype: object
```

[18]:
```
df = df.copy()

df['tor_rpm_index'] = df['tor'] * df['tor_rpm']
```

[19]:
```
df = df.copy()

df['hp_rpm_index'] = df['hp'] * df['hp_rpm'] # I already have horsepower column.
 ↪ I will define index unsing this step.
```

[20]:
```
df.head(3)
```

[20]:
```
                 vin back_legroom        body_type      city  \
0  ZACNJABB5KPJ92081      35.1 in  SUV / Crossover   Bayamon
1  SALCJ2FX1LH858117      38.1 in  SUV / Crossover  San Juan
3  SALRR2RV0L2433391      37.6 in  SUV / Crossover  San Juan

   city_fuel_economy  daysonmarket engine_cylinders  engine_displacement  \
0                NaN           522               I4               1300.0
1                NaN           207               I4               2000.0
3                NaN           196               V6               3000.0
```

```
   franchise_dealer franchise_make front_legroom fuel_tank_volume fuel_type  \
0              True           Jeep       41.2 in         12.7 gal  Gasoline
1              True     Land Rover       39.1 in         17.7 gal  Gasoline
3              True     Land Rover         39 in         23.5 gal  Gasoline

    height  highway_fuel_economy  horsepower  is_new    length listed_date  \
0  66.5 in                   NaN       177.0    True  166.6 in  2019-04-06
1    68 in                   NaN       246.0    True   181 in  2020-02-15
3    73 in                   NaN       340.0    True  195.1 in  2020-02-26

  listing_color   make_name maximum_seating  mileage       model_name  \
0        YELLOW        Jeep         5 seats      7.0         Renegade
1         BLACK  Land Rover         7 seats      8.0  Discovery Sport
3          GRAY  Land Rover         7 seats     11.0        Discovery

                power    price  savings_amount  seller_rating  \
0  177 hp @ 5,750 RPM  23141.0               0            2.8
1  246 hp @ 5,500 RPM  46500.0               0            3.0
3  340 hp @ 6,500 RPM  67430.0               0            3.0

                  torque transmission     trim_name wheel_system wheelbase  \
0  200 lb-ft @ 1,750 RPM            A  Latitude FWD          FWD  101.2 in
1  269 lb-ft @ 1,400 RPM            A         S AWD          AWD  107.9 in
3  332 lb-ft @ 3,500 RPM            A    V6 HSE AWD          AWD    115 in

   year   hp  hp_rpm  tor  tor_rpm  tor_rpm_index  hp_rpm_index
0  2019  177    5750  200     1750         350000       1017750
1  2020  246    5500  269     1400         376600       1353000
3  2020  340    6500  332     3500        1162000       2210000
```

[21]:
```
df[['torque', 'tor', 'tor_rpm', 'tor_rpm_index']].isna().sum() # checking if na␣
↪match
```

[21]:
```
torque           282751
tor              282751
tor_rpm          282751
tor_rpm_index    282751
dtype: int64
```

[22]:
```
df[['power', 'hp', 'hp_rpm', 'hp_rpm_index']].isna().sum()
```

[22]:
```
power           258432
hp              258432
hp_rpm          258432
hp_rpm_index    258432
dtype: int64
```

[23]:
```
# Now I am dropping torque, tor, tor_rpm, power, hp, hp_rpm from the dataframe
df1 = df.drop(['torque', 'tor', 'tor_rpm','power', 'hp', 'hp_rpm'], axis=1)
```

[24]:
```
df1.head() # In this dataframe, I created two numerical columns modifying power
 ↪and torque values.
```

[24]:

	vin	back_legroom	body_type	city
0	ZACNJABB5KPJ92081	35.1 in	SUV / Crossover	Bayamon
1	SALCJ2FX1LH858117	38.1 in	SUV / Crossover	San Juan
3	SALRR2RV0L2433391	37.6 in	SUV / Crossover	San Juan
4	SALCJ2FXXLH862327	38.1 in	SUV / Crossover	San Juan
5	SALYK2EX1LA261711	37.1 in	SUV / Crossover	San Juan

	city_fuel_economy	daysonmarket	engine_cylinders	engine_displacement
0	NaN	522	I4	1300.0
1	NaN	207	I4	2000.0
3	NaN	196	V6	3000.0
4	NaN	137	I4	2000.0
5	NaN	242	I4	2000.0

	franchise_dealer	franchise_make	front_legroom	fuel_tank_volume	fuel_type
0	True	Jeep	41.2 in	12.7 gal	Gasoline
1	True	Land Rover	39.1 in	17.7 gal	Gasoline
3	True	Land Rover	39 in	23.5 gal	Gasoline
4	True	Land Rover	39.1 in	17.7 gal	Gasoline
5	True	Land Rover	40.2 in	16.6 gal	Gasoline

	height	highway_fuel_economy	horsepower	is_new	length	listed_date
0	66.5 in	NaN	177.0	True	166.6 in	2019-04-06
1	68 in	NaN	246.0	True	181 in	2020-02-15
3	73 in	NaN	340.0	True	195.1 in	2020-02-26
4	68 in	NaN	246.0	True	181 in	2020-04-25
5	66.3 in	NaN	247.0	True	188.9 in	2020-01-11

	listing_color	make_name	maximum_seating	mileage	model_name
0	YELLOW	Jeep	5 seats	7.0	Renegade
1	BLACK	Land Rover	7 seats	8.0	Discovery Sport
3	GRAY	Land Rover	7 seats	11.0	Discovery
4	BLACK	Land Rover	7 seats	7.0	Discovery Sport
5	UNKNOWN	Land Rover	5 seats	12.0	Range Rover Velar

	price	savings_amount	seller_rating	transmission	trim_name
0	23141.0	0	2.8	A	Latitude FWD
1	46500.0	0	3.0	A	S AWD
3	67430.0	0	3.0	A	V6 HSE AWD
4	48880.0	0	3.0	A	S AWD
5	66903.0	0	3.0	A	P250 R-Dynamic S AWD

	wheel_system	wheelbase	year	tor_rpm_index	hp_rpm_index
0	FWD	101.2 in	2019	350000	1017750
1	AWD	107.9 in	2020	376600	1353000
3	AWD	115 in	2020	1162000	2210000
4	AWD	107.9 in	2020	376600	1353000
5	AWD	113.1 in	2020	322800	1358500

[25]:
```
# I noticed presence of '--' in some of the columns

(df1 == '--').sum()
```

[25]:
```
vin                       0
back_legroom          52440
body_type                 0
city                      0
city_fuel_economy         0
daysonmarket              0
engine_cylinders          0
engine_displacement       0
franchise_dealer          0
franchise_make            0
front_legroom         11039
fuel_tank_volume        276
fuel_type                 0
height                  257
highway_fuel_economy      0
horsepower                0
is_new                    0
length                  257
listed_date               0
listing_color             0
make_name                 0
maximum_seating         278
mileage                   0
model_name                0
price                     0
savings_amount            0
seller_rating             0
transmission              0
trim_name                 0
wheel_system              0
wheelbase               257
year                      0
tor_rpm_index             0
hp_rpm_index              0
dtype: Int64
```

[26]:
```
df1['back_legroom'] = df1['back_legroom'].replace('--', pd.NA) # datasets used placeholders "--" to indicate missing data.

df1['back_legroom'] = df1['back_legroom'].str.replace(' in', '', regex=False) # removing the string ' in'

df1['back_legroom'] = pd.to_numeric(df1['back_legroom'], errors='coerce') # converting column to numeric (float) type.
```

[27]:
```
df1['back_legroom'].dtype
```

[27]:
```
dtype('float64')
```

[28]:
```
df1.head(3)
```

[28]:

	vin	back_legroom	body_type	city	city_fuel_economy	daysonmarket	engine_cylinders	engine_displacement	franchise_dealer	franchise_make	front_legroom	fuel_tank_volume	fuel_type	height	highway_fuel_economy	horsepower	is_new	length	listed_date	listing_color	make_name	maximum_seating	mileage	model_name	price	savings_amount	seller_rating	transmission	trim_name	wheel_system	wheelbase	year	tor_rpm_index	hp_rpm_index
0	ZACNJABB5KPJ92081	35.1	SUV / Crossover	Bayamon	NaN	522	I4	1300.0	True	Jeep	41.2 in	12.7 gal	Gasoline	66.5 in	NaN	177.0	True	166.6 in	2019-04-06	YELLOW	Jeep	5 seats	7.0	Renegade	23141.0	0	2.8	A	Latitude FWD	FWD	101.2 in	2019	350000	1017750
1	SALCJ2FX1LH858117	38.1	SUV / Crossover	San Juan	NaN	207	I4	2000.0	True	Land Rover	39.1 in	17.7 gal	Gasoline	68 in	NaN	246.0	True	181 in	2020-02-15	BLACK	Land Rover	7 seats	8.0	Discovery Sport	46500.0	0	3.0	A	S AWD					
3	SALRR2RV0L2433391	37.6	SUV / Crossover	San Juan	NaN	196	V6	3000.0	True	Land Rover	39 in	23.5 gal	Gasoline	73 in	NaN	340.0	True	195.1 in	2020-02-26	GRAY	Land Rover	7 seats	11.0	Discovery	67430.0	0	3.0	A	V6 HSE AWD					

```
1          AWD  107.9 in  2020         376600       1353000
3          AWD    115 in  2020        1162000       2210000
```

[29]:
```
(df1['front_legroom']=='--').sum() # since string count
```

[29]:
```
11039
```

[30]:
```
df1['front_legroom'] = df1['front_legroom'].replace('--', pd.NA)

df1['front_legroom'] = df1['front_legroom'].str.replace(' in', '', regex=False)

df1['front_legroom'] = pd.to_numeric(df1['front_legroom'], errors='coerce')
```

[31]:
```
df1['front_legroom'].dtype
```

[31]:
```
dtype('float64')
```

[32]:
```
# Now we convert height to float
df1['height'] = df1['height'].replace('--', pd.NA)
df1['height'] = df1['height'].str.replace(' in', '', regex=False)
df1['height'] = pd.to_numeric(df1['height'], errors='coerce')
```

[33]:
```
# Now we convert length to float
df1['length'] = df1['length'].replace('--', pd.NA)
df1['length'] = df1['length'].str.replace(' in', '', regex=False)
df1['length'] = pd.to_numeric(df1['length'], errors='coerce')
```

[34]:
```
# Now we convert wheelbase to float
df1['wheelbase'] = df1['wheelbase'].replace('--', pd.NA)
df1['wheelbase'] = df1['wheelbase'].str.replace(' in', '', regex=False)
df1['wheelbase'] = pd.to_numeric(df1['wheelbase'], errors='coerce')
```

[35]:
```
(df1['fuel_tank_volume']=='--').sum()
```

[35]:
```
276
```

[36]:
```
# Now we convert wheelbase to float
df1['fuel_tank_volume'] = df1['fuel_tank_volume'].replace('--', pd.NA)
df1['fuel_tank_volume'] = df1['fuel_tank_volume'].str.replace(' gal', '',␣
 ↪regex=False)
df1['fuel_tank_volume'] = pd.to_numeric(df1['fuel_tank_volume'],␣
 ↪errors='coerce')
```

[37]:
```
(df1['maximum_seating']=='--').sum()
```

[37]:
```
278
```

[38]:
```
# Now we convert maximum seating to float
df1['maximum_seating'] = df1['maximum_seating'].replace('--', pd.NA)
df1['maximum_seating'] = df1['maximum_seating'].str.replace(' seats', '',␣
 ↪regex=False)
df1['maximum_seating'] = pd.to_numeric(df1['maximum_seating'], errors='coerce')
```

[39]:
```
df1.tail()
```

[39]:

	vin	back_legroom	body_type	city	city_fuel_economy	daysonmarket	engine_cylinders	engine_displacement	franchise_dealer	franchise_make	front_legroom	fuel_tank_volume	fuel_type	height	highway_fuel_economy	horsepower	is_new	length	listed_date	listing_color	make_name	maximum_seating	mileage	model_name	price	savings_amount	seller_rating
3000034	3TMCZ5AN6HM083436	32.6	Pickup Truck	Ukiah	18.0	89	V6	3500.0	True	Ford	42.9	21.1	Gasoline	70.6	23.0	278.0	False	212.3	2020-06-15	WHITE	Toyota	5.0	20009.0	Tacoma	40993.0	2220	5.000000
3000035	2GNAXJEV0J6261526	39.7	SUV / Crossover	Fairfield	26.0	16	I4	1500.0	False	NaN	40.9	14.9	Gasoline	65.4	32.0	170.0	False	183.1	2020-08-27	SILVER	Chevrolet	5.0	41897.0	Equinox	17998.0	381	4.272727
3000036	1GNERFKW0LJ225508	38.4	SUV / Crossover	Vallejo	18.0	171	V6	3600.0	True	Chevrolet	41.0	19.4	Gasoline	70.7	27.0	310.0	True	204.3	2020-03-25	BLACK	Chevrolet	8.0	5.0	Traverse	36490.0	0	4.533333
3000038	SAJAJ4BNXHA968809	35.0	Sedan	Fairfield	30.0	11	I4 Diesel	2000.0	False	NaN	41.5	14.8	Diesel	55.7	40.0	180.0	False	183.9	2020-09-01	GREEN	Jaguar	5.0					
3000039	JN8AT2MT1HW400805	37.9	SUV / Crossover	Napa	26.0	17	I4	2500.0	True	Nissan	43.0	14.5	Gasoline	68.1	33.0	170.0	False	184.5	2020-08-26	SILVER	Nissan	7.0					

```
3000038  27857.0         XE  26998.0             849       4.272727
3000039  22600.0      Rogue  19900.0            1203       4.333333

        transmission                    trim_name wheel_system  wheelbase  \
3000034            A  TRD Sport V6 Double Cab 4WD          4WD      127.4
3000035            A                1.5T LT FWD          FWD      107.3
3000036            A                     LS FWD          FWD      120.9
3000038            A           20d Premium AWD          AWD      111.6
3000039            A             2017.5 SV FWD          FWD      106.5

         year  tor_rpm_index  hp_rpm_index
3000034  2017        1219000       1668000
3000035  2018           <NA>          <NA>
3000036  2020         744800       2108000
3000038  2017         556500        720000
3000039  2017         770000       1020000
```

[40]:
```
# I will convert listed_date to listed year only. I will use this column to␣
 ↪identify the age
# of the car at the time of listing.

df1['list_year'] = pd.to_datetime(df1['listed_date']).dt.year
```

[41]:
```
df1['age'] = df1['list_year']-df1['year']
```

[42]:
```
(df1['age'] < 0).sum() # I noticed some of the values are less than 0.
# This migh be due to the fact the car for next year can be sold this year. I␣
 ↪will keep list year and year for now.
# If necessary I can drop them after checking correlation plot.
```

[42]:
```
224882
```

[43]:
```
# I will drop listed_date, the original column before modificiation
df1.drop(['listed_date'], axis =1, inplace=True)
```

[44]:
```
# I will drop is_new column as well
df1.drop(['is_new'], axis =1, inplace=True)
```

[45]:
```
df1['franchise_dealer'].value_counts()
```

[45]:
```
franchise_dealer
True     1817287
False     203289
Name: count, dtype: int64
```

[46]:
```
df1['franchise_dealer'] = df1['franchise_dealer'].replace({True:'yes', False:
 ↪'no'}) # little modification
```

[47]:
```
# Since I added na for '--', I will check for na in df
df1.isna().sum()
```

[47]:
```
vin                       0
back_legroom          52440
body_type                 0
city                      0
city_fuel_economy    190441
daysonmarket              0
engine_cylinders          0
engine_displacement       0
franchise_dealer          0
franchise_make       206363
front_legroom         11039
fuel_tank_volume        276
fuel_type                 0
height                  257
highway_fuel_economy 190441
horsepower                0
length                  257
listing_color             0
make_name                 0
maximum_seating         278
mileage                   0
model_name                0
price                     0
savings_amount            0
seller_rating             0
transmission              0
trim_name                 0
wheel_system              0
wheelbase               257
year                      0
tor_rpm_index        282751
hp_rpm_index         258432
list_year                 0
age                       0
dtype: int64
```

[48]:
```
# I will remove NA from these columns
df1.dropna(subset=['back_legroom', 'front_legroom', 'fuel_tank_volume',␣
 ↪'height', 'length', 'maximum_seating','wheelbase'], inplace=True)
```

[49]:
```
df1.isna().sum()
```

[49]:
```
vin                   0
back_legroom          0
```

```
body_type                   0
city                        0
city_fuel_economy      168104
daysonmarket                0
engine_cylinders            0
engine_displacement         0
franchise_dealer            0
franchise_make         200632
front_legroom               0
fuel_tank_volume            0
fuel_type                   0
height                      0
highway_fuel_economy   168104
horsepower                  0
length                      0
listing_color               0
make_name                   0
maximum_seating             0
mileage                     0
model_name                  0
price                       0
savings_amount              0
seller_rating               0
transmission                0
trim_name                   0
wheel_system                0
wheelbase                   0
year                        0
tor_rpm_index          274669
hp_rpm_index           250588
list_year                   0
age                         0
dtype: int64
```

[50]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1968136 entries, 0 to 3000039
Data columns (total 34 columns):
 #   Column                Dtype
---  ------                -----
 0   vin                   object
 1   back_legroom          float64
 2   body_type             object
 3   city                  object
 4   city_fuel_economy     float64
 5   daysonmarket          int64
 6   engine_cylinders      object
```

```
 7   engine_displacement   float64
 8   franchise_dealer      object
 9   franchise_make        object
 10  front_legroom         float64
 11  fuel_tank_volume      float64
 12  fuel_type             object
 13  height                float64
 14  highway_fuel_economy  float64
 15  horsepower            float64
 16  length                float64
 17  listing_color         object
 18  make_name             object
 19  maximum_seating       float64
 20  mileage               float64
 21  model_name            object
 22  price                 float64
 23  savings_amount        int64
 24  seller_rating         float64
 25  transmission          object
 26  trim_name             object
 27  wheel_system          object
 28  wheelbase             float64
 29  year                  int64
 30  tor_rpm_index         Int64
 31  hp_rpm_index          Int64
 32  list_year             int32
 33  age                   int64
dtypes: Int64(2), float64(14), int32(1), int64(4), object(13)
memory usage: 521.8+ MB
```

Note that a number of variables are still obect type. I will have to convert them to dummy for ML algorithms later.

[51]:
```
df1.nunique(axis=0)
```

[51]:
```
vin                    1968106
back_legroom               145
body_type                    9
city                      4049
city_fuel_economy           63
daysonmarket              1170
engine_cylinders            24
engine_displacement         42
franchise_dealer             2
franchise_make              45
front_legroom               71
fuel_tank_volume           126
fuel_type                    6
```

```
height                      288
highway_fuel_economy         60
horsepower                  261
length                      427
listing_color                15
make_name                    38
maximum_seating               9
mileage                   91783
model_name                  444
price                     74484
savings_amount             8445
seller_rating              1735
transmission                  4
trim_name                  2749
wheel_system                  5
wheelbase                   241
year                          5
tor_rpm_index               424
hp_rpm_index                457
list_year                     5
age                           5
dtype: int64
```

[52]:
```
df1.duplicated().sum()
```

[52]:
```
30
```

We noticed we have 30 rows of duplicated values. Lets print them.

[53]:
```
duplicated = df1[df1.duplicated()]

duplicated.head(5)
```

[53]:
```
                        vin  back_legroom        body_type  \
1000021  3TMCZ5AN4KM203273          32.6     Pickup Truck
1000022  1FTEW1EP0HKC03244          43.6     Pickup Truck
1000023  3FADP4EJ0KM129514          31.2        Hatchback
1000024  1N6AD0EV6KN713209          33.6     Pickup Truck
1000025  5N1AZ2MS4KN121455          38.7  SUV / Crossover

                           city  city_fuel_economy  daysonmarket  \
1000021               Groveport               18.0             6
1000022               Groveport               18.0            65
1000023  Washington Court House               27.0           253
1000024                  Heath               15.0            32
1000025                  Heath               20.0            29

         engine_cylinders  engine_displacement franchise_dealer franchise_make  \
```

```
1000021              V6              3500.0              no            NaN
1000022              V6              3500.0              no            NaN
1000023              I4              1600.0             yes          Ford
1000024              V6              4000.0             yes         Nissan
1000025              V6              3500.0             yes         Nissan

         front_legroom  fuel_tank_volume fuel_type  height  \
1000021           42.9              21.1  Gasoline    70.6
1000022           43.9              23.0  Gasoline    76.9
1000023           43.6              12.4  Gasoline    58.1
1000024           42.4              21.1  Gasoline    70.1
1000025           40.5              19.0  Gasoline    67.8

         highway_fuel_economy  horsepower  length listing_color make_name  \
1000021                  22.0       278.0   212.3         WHITE    Toyota
1000022                  23.0       375.0   231.9           RED      Ford
1000023                  37.0       120.0   159.7       UNKNOWN      Ford
1000024                  21.0       261.0   205.5          GRAY    Nissan
1000025                  28.0       260.0   192.8          GRAY    Nissan

         maximum_seating  mileage model_name    price  savings_amount  \
1000021              5.0  18320.0     Tacoma  33888.0               4
1000022              6.0  29917.0      F-150  35890.0              97
1000023              5.0   4230.0     Fiesta  14400.0             588
1000024              5.0  30096.0   Frontier  29990.0            1133
1000025              5.0   3750.0     Murano  29990.0            1421

         seller_rating transmission             trim_name wheel_system  \
1000021       4.211765            A  SR V6 Double Cab 4WD          4WD
1000022       4.211765            A     XLT SuperCrew 4WD          4WD
1000023       4.000000            A      SE Hatchback FWD          FWD
1000024       5.000000            A   PRO-4X Crew Cab 4WD          4WD
1000025       5.000000          CVT                SV AWD          AWD

         wheelbase  year  tor_rpm_index  hp_rpm_index  list_year  age
1000021      127.4  2019        1219000       1668000       2020    1
1000022      145.0  2017        1175000       1875000       2020    3
1000023       98.0  2019         560000        600000       2020    1
1000024      125.9  2019        1124000       1461600       2020    1
1000025      111.2  2019        1056000       1560000       2020    1
```

[54]:
```
df1.drop_duplicates(inplace=True)  # Now lets drop duplicated values.
```

[55]:
```
df1.duplicated().sum() # Check to make sure duplicates are gone.
```

[55]: 0

[56]:
```
# Check for any anomalies in object type
for i in df1.select_dtypes(include = 'object').columns:
    print(df1[i].value_counts())
    print('***'*15)
```

```
vin
ZACNJABB5KPJ92081    1
1GYKNCRS6LZ220267    1
1FTEW1E58LFA16009    1
3GCUYDED9LG333715    1
3GCUYDED2LG335872    1
                    ..
2T2BZMCA2HC088413    1
WDC0G4KB1JV088265    1
1G6DU5RK0L0151919    1
3C63RRKL4LG220960    1
JN8AT2MT1HW400805    1
Name: count, Length: 1968106, dtype: int64
*********************************************
body_type
SUV / Crossover    1026982
Sedan               484554
Pickup Truck        292528
Minivan              52337
Hatchback            48478
Coupe                31189
Wagon                17721
Convertible           7263
Van                   7054
Name: count, dtype: int64
*********************************************
city
Houston         28413
San Antonio     17721
Columbus        12266
Miami           11318
Jacksonville    10452
                …
Forest Grove        1
Dacula              1
Southbridge         1
Cedar Creek         1
Becker              1
Name: count, Length: 4049, dtype: int64
*********************************************
engine_cylinders
I4                          1043043
V6                           477435
```

```
V8                            176463
I4 Hybrid                      54505
H4                             48325
I3                             46206
V6 Flex Fuel Vehicle           34837
V8 Flex Fuel Vehicle           29171
I6 Diesel                      17581
V8 Biodiesel                   14818
I6                             13534
I4 Flex Fuel Vehicle            3120
V6 Biodiesel                    2710
V6 Hybrid                       1798
V6 Diesel                       1312
I4 Diesel                       1098
H6                               818
I2                               640
V12                              239
V8 Diesel                        148
I5                               142
W12                              140
I5 Biodiesel                      17
V8 Compressed Natural Gas          6
Name: count, dtype: int64
*****************************************
franchise_dealer
yes    1770431
no      197675
Name: count, dtype: int64
*****************************************
franchise_make
Ford           282684
Chevrolet      245935
Honda          149840
Toyota         141465
Nissan         118990
                …
Lamborghini        35
Shelby             31
Ferrari             9
SRT                 9
Pagani              3
Name: count, Length: 45, dtype: int64
*****************************************
fuel_type
Gasoline                  1806985
Flex Fuel Vehicle           67128
Hybrid                      56303
Diesel                      20139
```

```
Biodiesel                   17545
Compressed Natural Gas          6
Name: count, dtype: int64
******************************************
listing_color
WHITE      420484
BLACK      394417
UNKNOWN    275569
GRAY       259010
SILVER     254631
BLUE       164904
RED        161507
GREEN       12196
BROWN        8293
ORANGE       7472
TEAL         4121
GOLD         3071
YELLOW       1799
PURPLE        574
PINK           58
Name: count, dtype: int64
******************************************
make_name
Ford             305987
Chevrolet        249839
Toyota           160811
Nissan           155102
Honda            154532
Jeep             109259
Hyundai           89616
Kia               74877
RAM               64560
GMC               64499
Dodge             59921
Volkswagen        49194
Subaru            48974
Buick             47918
Mercedes-Benz     39892
Mazda             35693
BMW               34132
Cadillac          32621
Audi              25081
Lincoln           24644
Lexus             21862
Chrysler          21685
Acura             21128
INFINITI          20351
Volvo             14857
```

```
Mitsubishi        12808
Land Rover        10821
Jaguar             5762
MINI               4607
Alfa Romeo         2985
Genesis            1546
Maserati           1218
FIAT                848
Bentley             246
Rolls-Royce         112
Porsche             100
Karma                13
Aston Martin          5
Name: count, dtype: int64
******************************************
model_name
F-150                   81131
Equinox                 50070
Silverado 1500          49674
1500                    46966
Escape                  43096
                        …
Rapide                      5
LS Hybrid                   2
GS Hybrid                   1
Escape Hybrid Plug-in       1
QX60 Hybrid                 1
Name: count, Length: 444, dtype: int64
******************************************
transmission
A              1575364
CVT             368432
M                17617
Dual Clutch       6693
Name: count, dtype: int64
******************************************
trim_name
SE FWD                          72579
S FWD                           38355
XLT SuperCrew 4WD               33009
LE FWD                          32884
SV FWD                          32331
                                …
1.8T Dune Convertible               1
SR5 Double Cab 5.7L FFV             1
2.0T ultra Prestige Sedan FWD       1
Platinum Crew Cab LB DRW RWD        1
Retro Hatchback FWD                 1
```

```
Name: count, Length: 2749, dtype: int64
*****************************************
wheel_system
FWD    874421
AWD    513232
4WD    408844
RWD     96146
4X2     75463
Name: count, dtype: int64
*****************************************
```

4 Exploratory Data Analysis

[57]:
```
df2 = df1.reset_index(drop=True) # I will reset the index here.
df2.tail()
```

[57]:

	vin	back_legroom	body_type	city	\
1968101	3TMCZ5AN6HM083436	32.6	Pickup Truck	Ukiah	
1968102	2GNAXJEV0J6261526	39.7	SUV / Crossover	Fairfield	
1968103	1GNERFKW0LJ225508	38.4	SUV / Crossover	Vallejo	
1968104	SAJAJ4BNXHA968809	35.0	Sedan	Fairfield	
1968105	JN8AT2MT1HW400805	37.9	SUV / Crossover	Napa	

	city_fuel_economy	daysonmarket	engine_cylinders	\
1968101	18.0	89	V6	
1968102	26.0	16	I4	
1968103	18.0	171	V6	
1968104	30.0	11	I4 Diesel	
1968105	26.0	17	I4	

	engine_displacement	franchise_dealer	franchise_make	front_legroom	\
1968101	3500.0	yes	Ford	42.9	
1968102	1500.0	no	NaN	40.9	
1968103	3600.0	yes	Chevrolet	41.0	
1968104	2000.0	no	NaN	41.5	
1968105	2500.0	yes	Nissan	43.0	

	fuel_tank_volume	fuel_type	height	highway_fuel_economy	horsepower	\
1968101	21.1	Gasoline	70.6	23.0	278.0	
1968102	14.9	Gasoline	65.4	32.0	170.0	
1968103	19.4	Gasoline	70.7	27.0	310.0	
1968104	14.8	Diesel	55.7	40.0	180.0	
1968105	14.5	Gasoline	68.1	33.0	170.0	

	length	listing_color	make_name	maximum_seating	mileage	model_name	\
1968101	212.3	WHITE	Toyota	5.0	20009.0	Tacoma	
1968102	183.1	SILVER	Chevrolet	5.0	41897.0	Equinox	

```
1968103   204.3         BLACK  Chevrolet              8.0      5.0   Traverse
1968104   183.9         GREEN     Jaguar              5.0  27857.0         XE
1968105   184.5        SILVER     Nissan              7.0  22600.0      Rogue

           price  savings_amount  seller_rating transmission  \
1968101  40993.0            2220       5.000000            A
1968102  17998.0             381       4.272727            A
1968103  36490.0               0       4.533333            A
1968104  26998.0             849       4.272727            A
1968105  19900.0            1203       4.333333            A

                          trim_name wheel_system  wheelbase  year  \
1968101  TRD Sport V6 Double Cab 4WD          4WD      127.4  2017
1968102                1.5T LT FWD          FWD      107.3  2018
1968103                     LS FWD          FWD      120.9  2020
1968104             20d Premium AWD          AWD      111.6  2017
1968105               2017.5 SV FWD          FWD      106.5  2017

         tor_rpm_index  hp_rpm_index  list_year  age
1968101        1219000       1668000       2020    3
1968102           <NA>          <NA>       2020    2
1968103         744800       2108000       2020    0
1968104         556500        720000       2020    3
1968105         770000       1020000       2020    3
```

[58]:
```
df2.shape # Rows match with the end of df,  keep in mind the first row is␣
 ↪indexed as 0.
```

[58]:
```
(1968106, 34)
```

[59]:
```
pd.options.display.float_format = '{:.2f}'.format # Format to 2 decimal places
df2.describe().T
```

[59]:
```
                         count       mean       std       min        25%  \
back_legroom         1968106.00      38.21      2.97     24.60      36.20
city_fuel_economy    1800006.00      22.85      6.47     10.00      18.00
daysonmarket         1968106.00      77.57     99.42      0.00      16.00
engine_displacement  1968106.00    2841.95   1299.69    700.00    2000.00
front_legroom        1968106.00      42.23      1.44     35.80      41.00
fuel_tank_volume     1968106.00      18.19      5.15      2.30      14.50
height               1968106.00      65.82      6.91     50.60      58.20
highway_fuel_economy 1800006.00      29.61      6.20     11.00      25.00
horsepower           1968106.00     246.19     89.35     78.00     174.00
length               1968106.00     193.42     19.79    139.60     182.00
maximum_seating      1968106.00       5.55      1.07      4.00       5.00
mileage              1968106.00   12966.32  20036.66      0.00       5.00
price                1968106.00   32576.64  15518.94   3490.00   21863.00
```

savings_amount	1968106.00	478.07	934.17	0.00	0.00
seller_rating	1968106.00	4.26	0.51	1.00	4.00
wheelbase	1968106.00	115.17	14.27	90.60	106.30
year	1968106.00	2019.25	1.21	2017.00	2018.00
tor_rpm_index	1693439.00	885840.50	469211.87	29600.00	524400.00
hp_rpm_index	1717519.00	1435474.60	531792.64	104000.00	1020000.00
list_year	1968106.00	2019.92	0.28	2016.00	2020.00
age	1968106.00	0.68	1.26	-1.00	0.00

	50%	75%	max
back_legroom	38.30	39.90	52.80
city_fuel_economy	22.00	26.00	107.00
daysonmarket	39.00	86.00	1574.00
engine_displacement	2500.00	3500.00	7300.00
front_legroom	42.00	43.10	52.50
fuel_tank_volume	17.10	20.50	63.50
height	66.10	69.90	117.60
highway_fuel_economy	29.00	33.00	93.00
horsepower	243.00	300.00	808.00
length	189.80	199.60	282.30
maximum_seating	5.00	6.00	15.00
mileage	15.00	25128.00	4290461.00
price	28900.00	40062.00	2698500.00
savings_amount	0.00	633.00	74595.00
seller_rating	4.33	4.60	5.00
wheelbase	111.20	118.40	179.80
year	2020.00	2020.00	2021.00
tor_rpm_index	744800.00	1236100.00	3226500.00
hp_rpm_index	1316000.00	1875000.00	5548000.00
list_year	2020.00	2020.00	2020.00
age	0.00	2.00	3.00

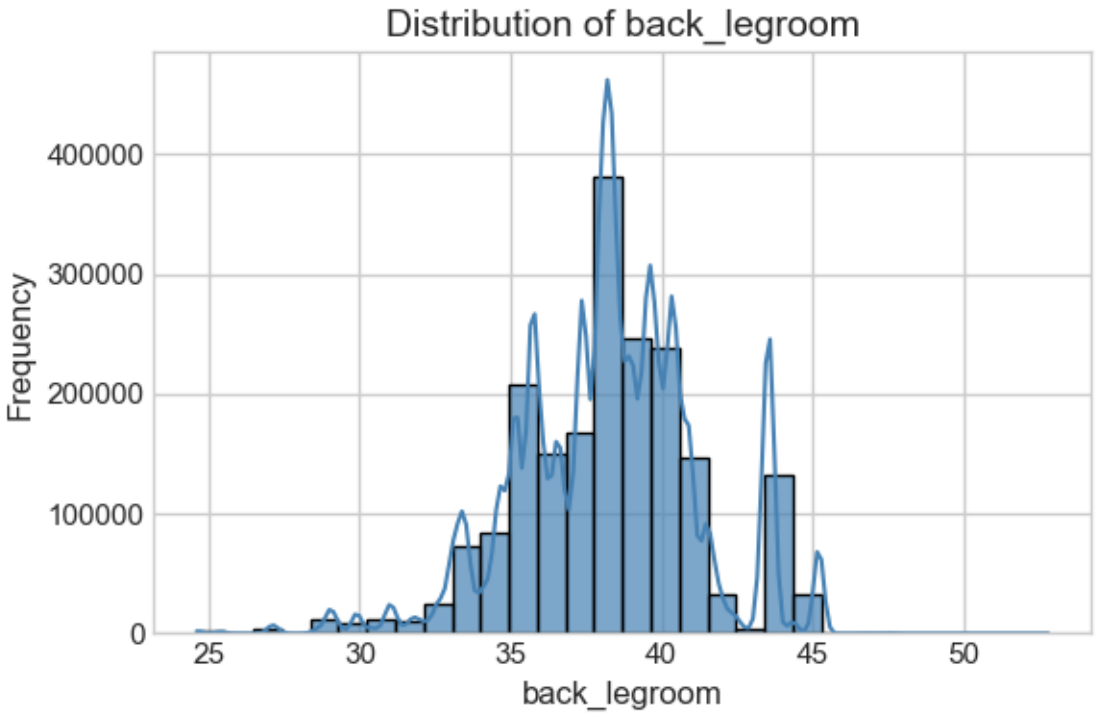
[60]:
```
df2.describe(include = 'object').T
```

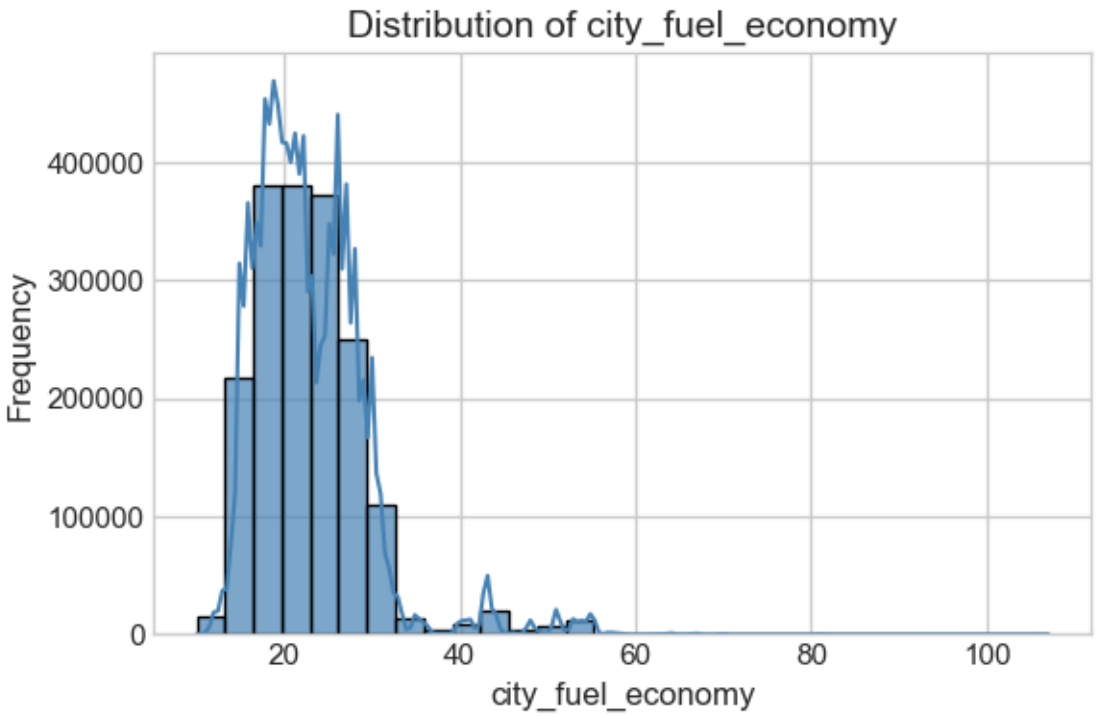
[60]:

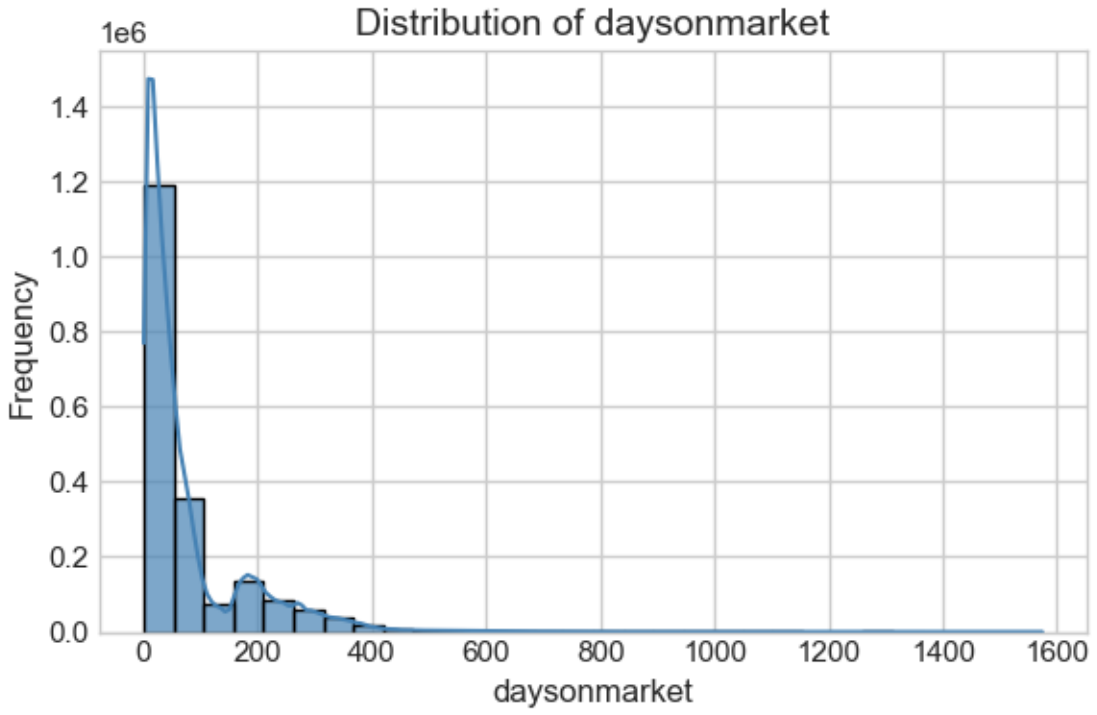
	count	unique	top	freq
vin	1968106	1968106	ZACNJABB5KPJ92081	1
body_type	1968106	9	SUV / Crossover	1026982
city	1968106	4049	Houston	28413
engine_cylinders	1968106	24	I4	1043043
franchise_dealer	1968106	2	yes	1770431
franchise_make	1767487	45	Ford	282684
fuel_type	1968106	6	Gasoline	1806985
listing_color	1968106	15	WHITE	420484
make_name	1968106	38	Ford	305987
model_name	1968106	444	F-150	81131
transmission	1968106	4	A	1575364
trim_name	1968106	2749	SE FWD	72579

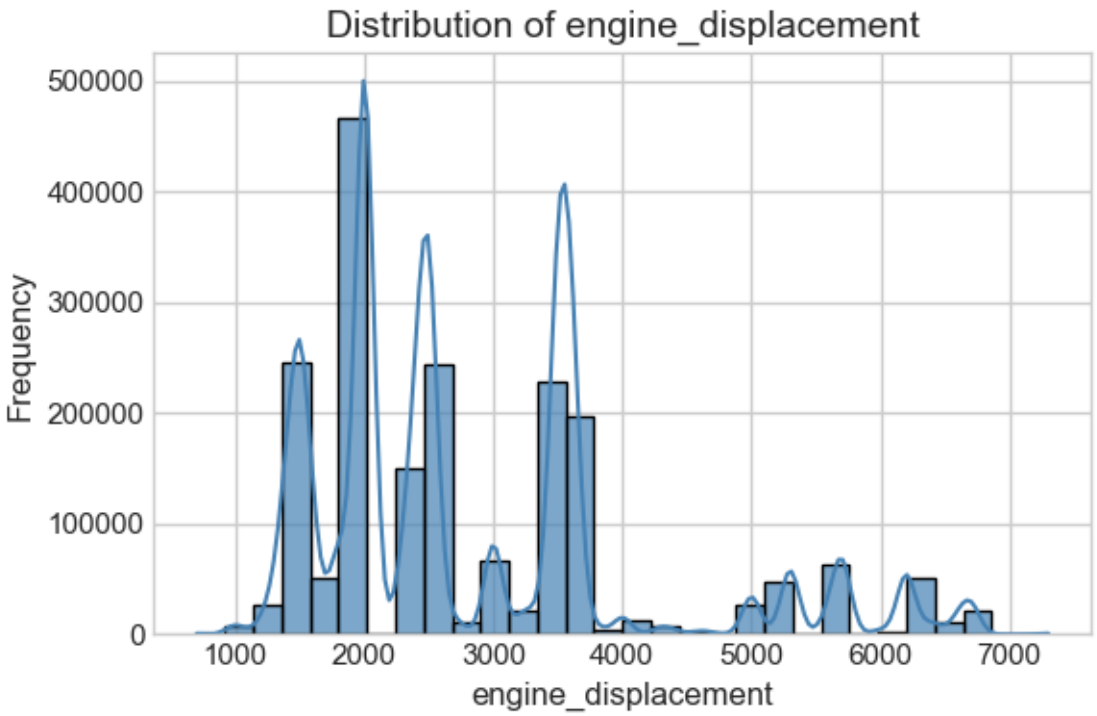
```
wheel_system      1968106        5                FWD    874421
```

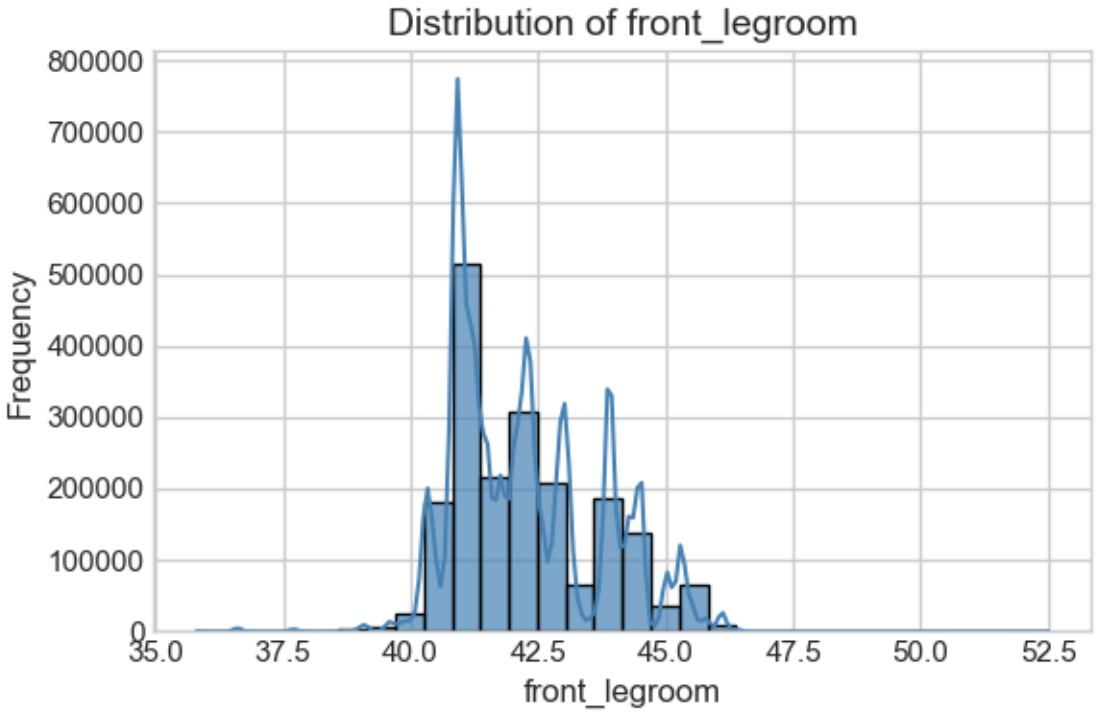
[61]:
```
# Histogram to check the distribution
for i in df2.select_dtypes(include='number').columns:
    plt.figure(figsize=(6, 4))
    sns.histplot(data=df2, x=i, kde=True, bins=30, color='steelblue',␣
 ↪edgecolor='black', alpha=0.7)
    plt.title(f'Distribution of {i}', fontsize=14)
    plt.xlabel(i, fontsize=12)
    plt.ylabel('Frequency', fontsize=12)
    plt.tight_layout()
    plt.show()
```

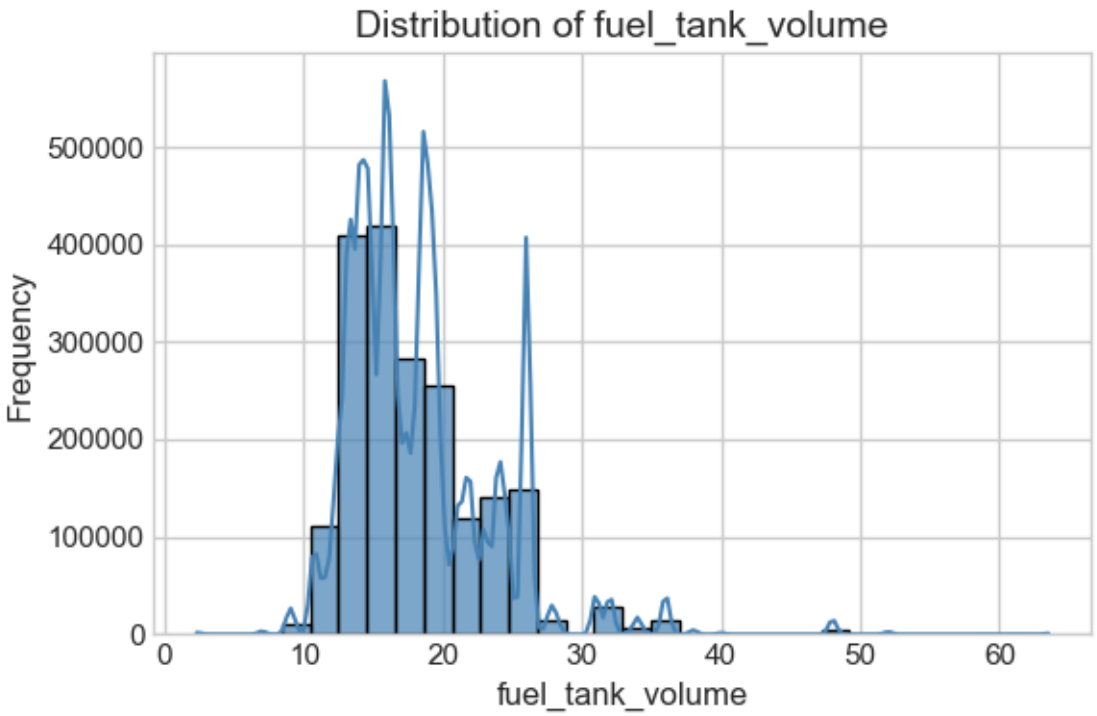


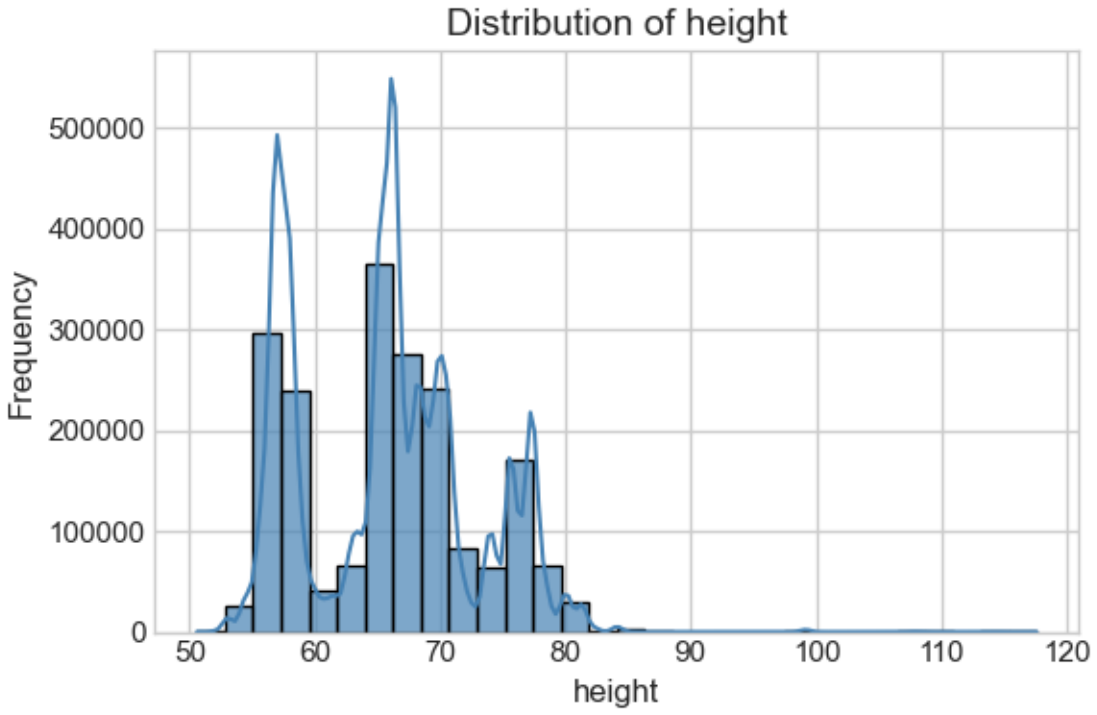

Distribution of city_fuel_economy
Frequency
400000
300000
200000
100000
0
20
40
60
80
100
city_fuel_economy

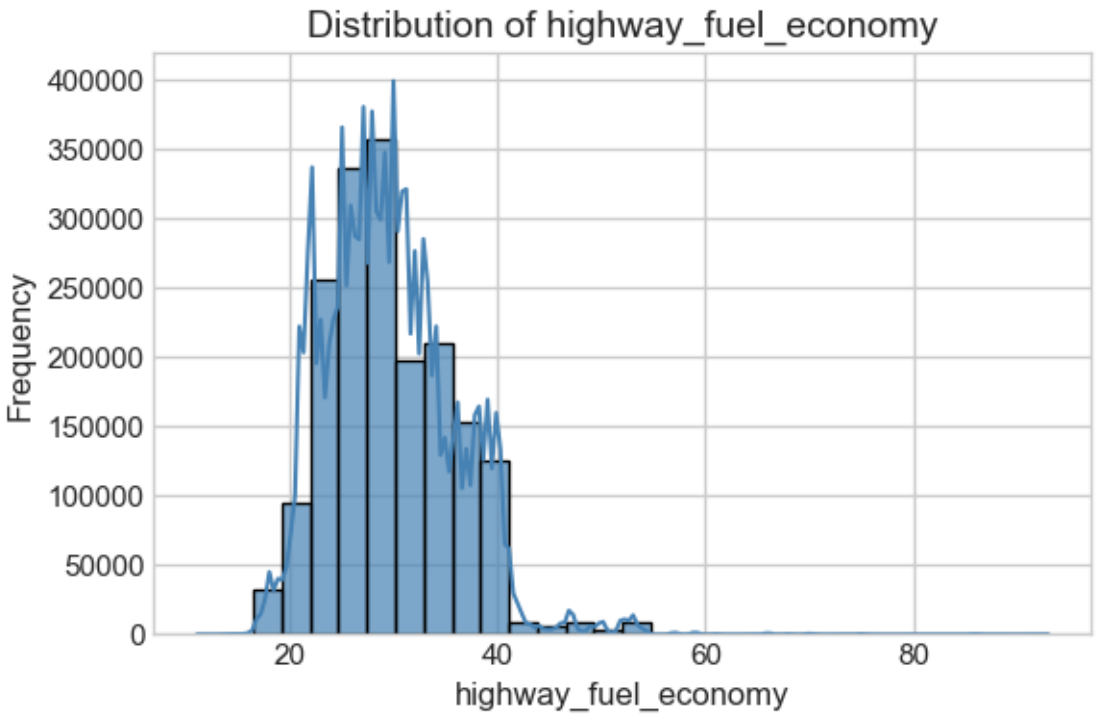

Distribution of daysonmarket
1e6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Frequency
0
200
400
600
800
1000
1200
1400
1600
daysonmarket

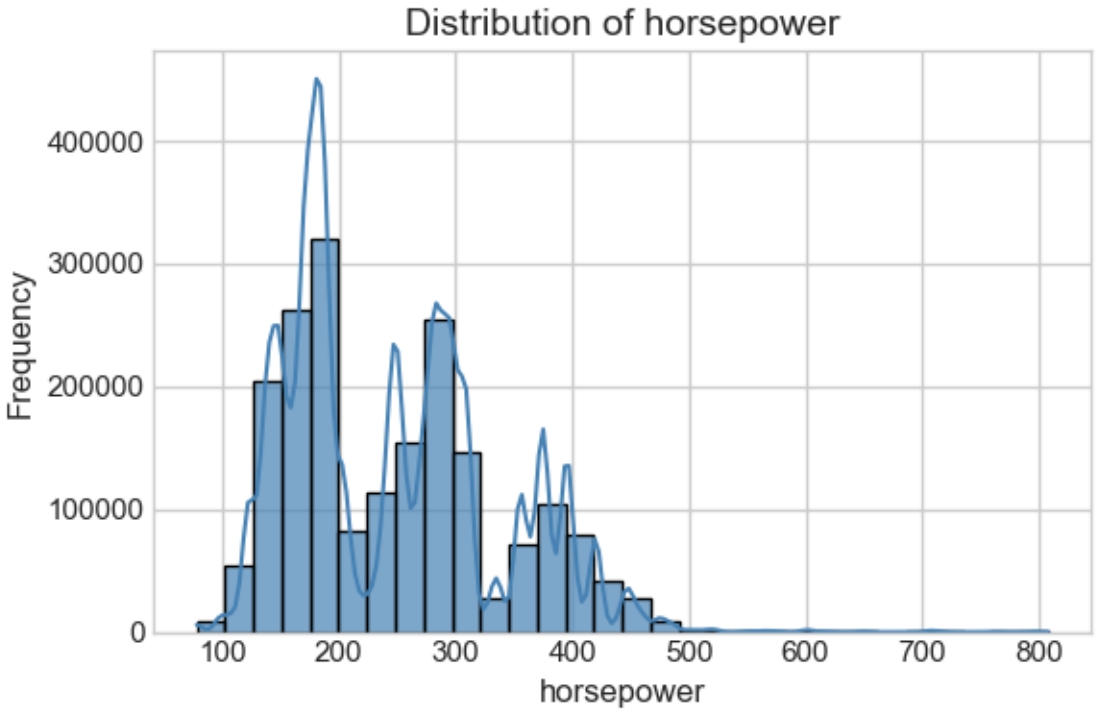

Distribution of engine_displacement
Frequency
500000
400000
300000
200000
100000
0
1000
2000
3000
4000
5000
6000
7000
engine_displacement

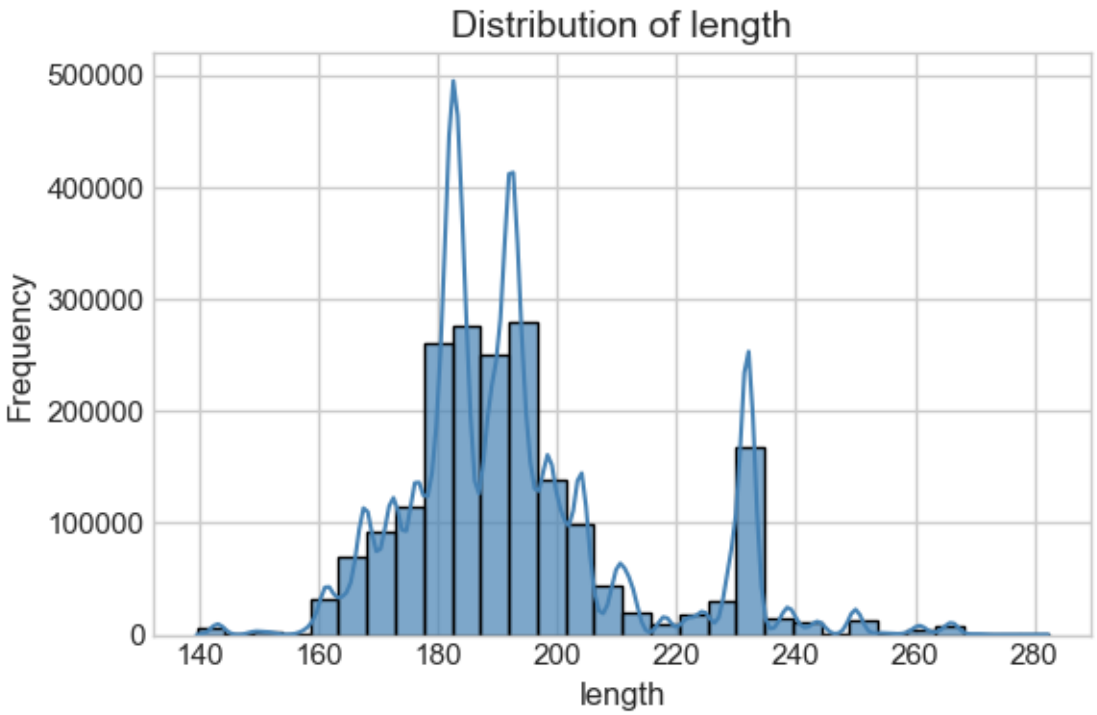

Distribution of front_legroom
Frequency
800000
700000
600000
500000
400000
300000
200000
100000
0
35.0
37.5
40.0
42.5
45.0
47.5
50.0
52.5
front_legroom

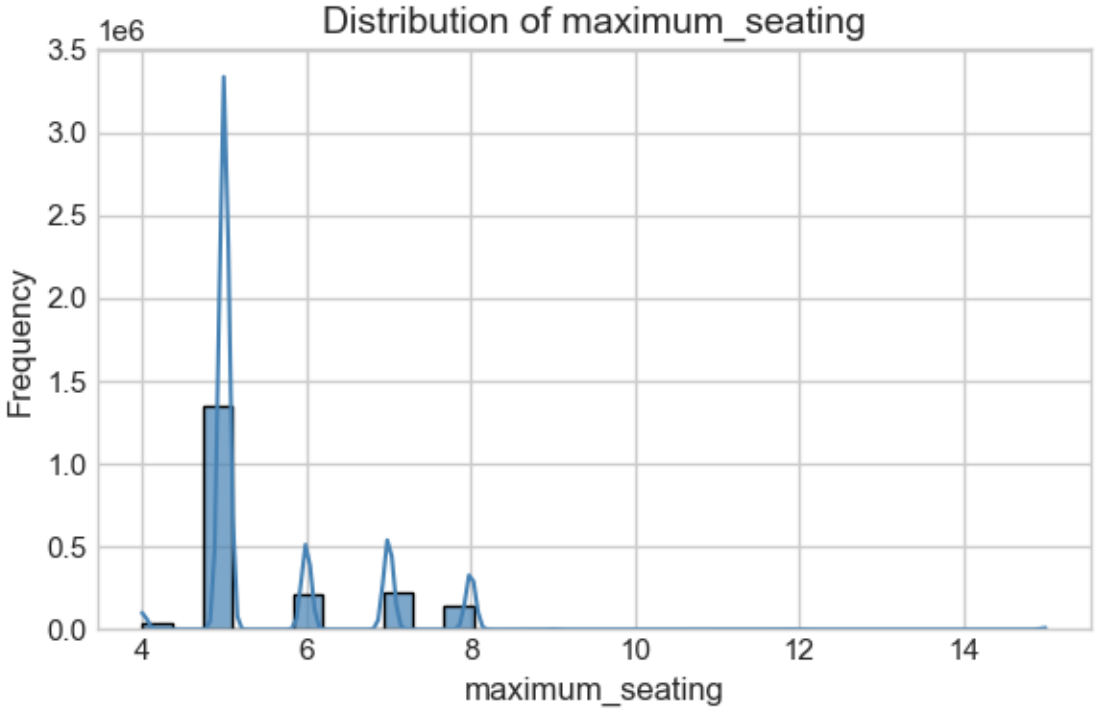

Distribution of fuel_tank_volume
Frequency
500000
400000
300000
200000
100000
0
0
10
20
30
40
50
60
fuel_tank_volume

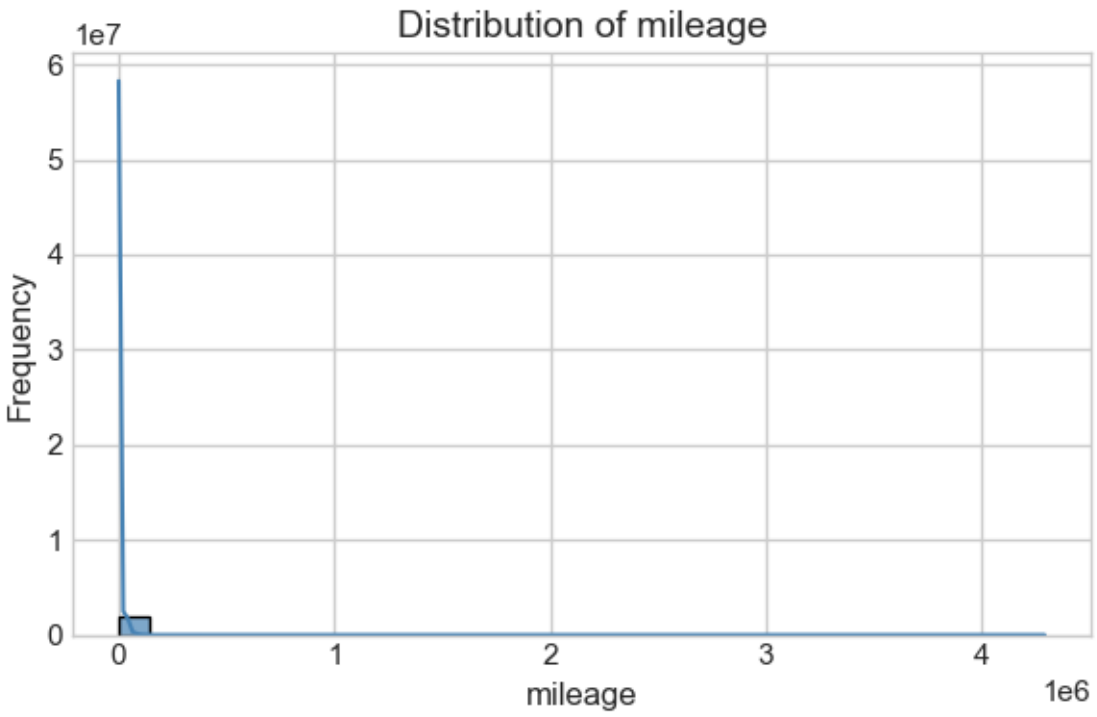

Distribution of height
Frequency
500000
400000
300000
200000
100000
0
50
60
70
80
90
100
110
120
height

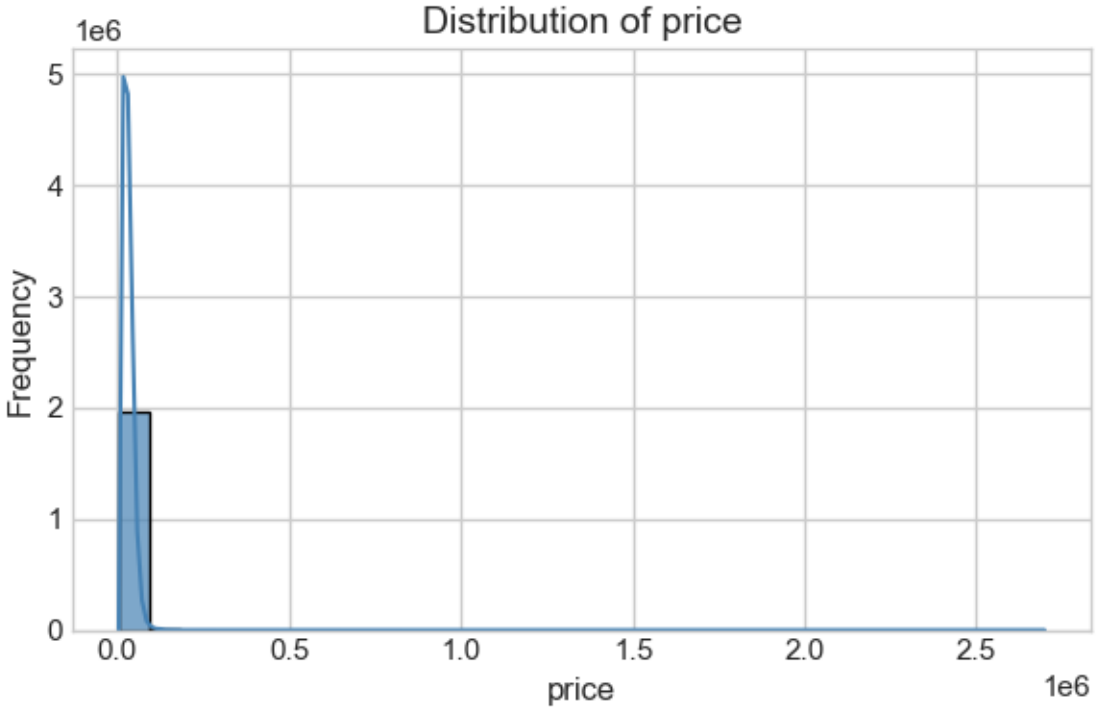

Distribution of highway_fuel_economy
Frequency
400000
350000
300000
250000
200000
150000
100000
50000
0
20
40
60
80
highway_fuel_economy


Distribution of horsepower
Frequency
400000
300000
200000
100000
0
100
200
300
400
500
600
700
800
horsepower

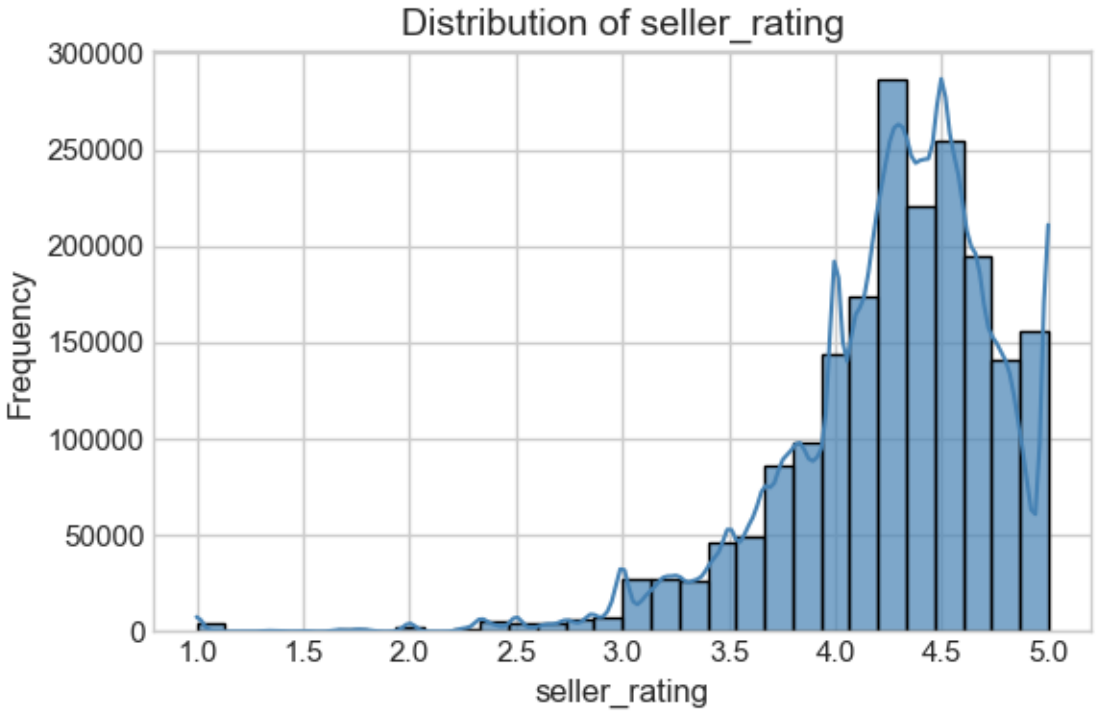

Distribution of length
Frequency
500000
400000
300000
200000
100000
0
140
160
180
200
220
240
260
280
length

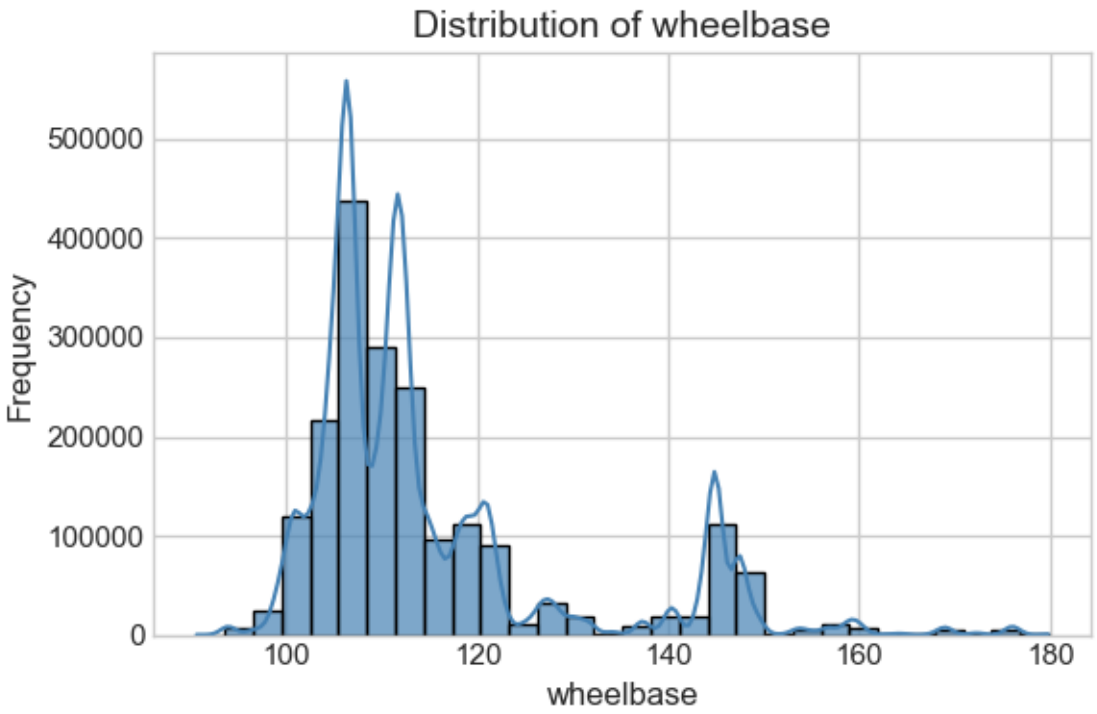

Distribution of maximum_seating
1e6
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Frequency
4
6
8
10
12
14
maximum_seating

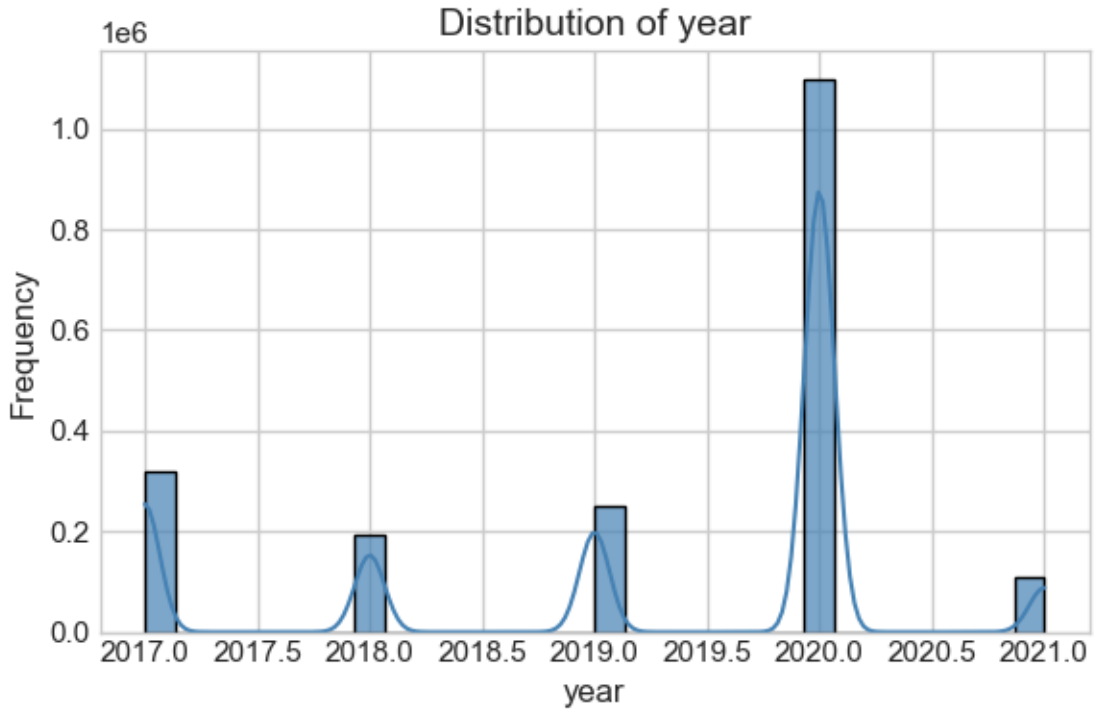

Distribution of mileage
1e7
6
5
4
3
2
1
0
Frequency
0
1
2
3
4
mileage
1e6

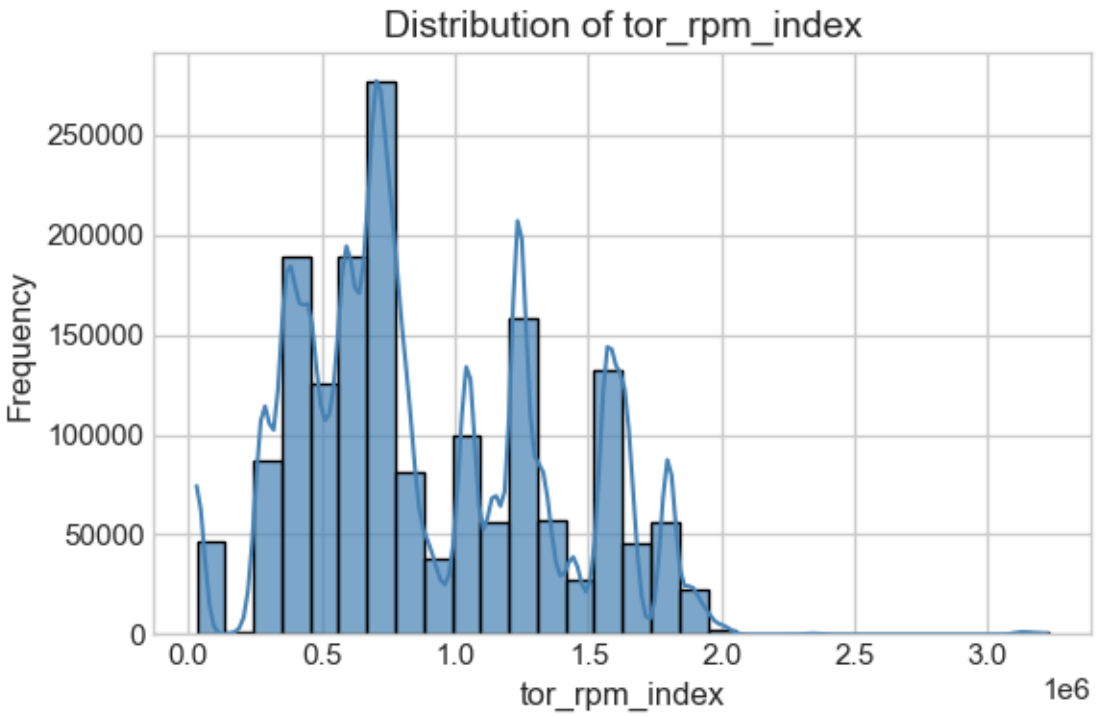

Distribution of price
1e6
5
4
3
2
1
0
Frequency
0.0
0.5
1.0
1.5
2.0
2.5
price
1e6

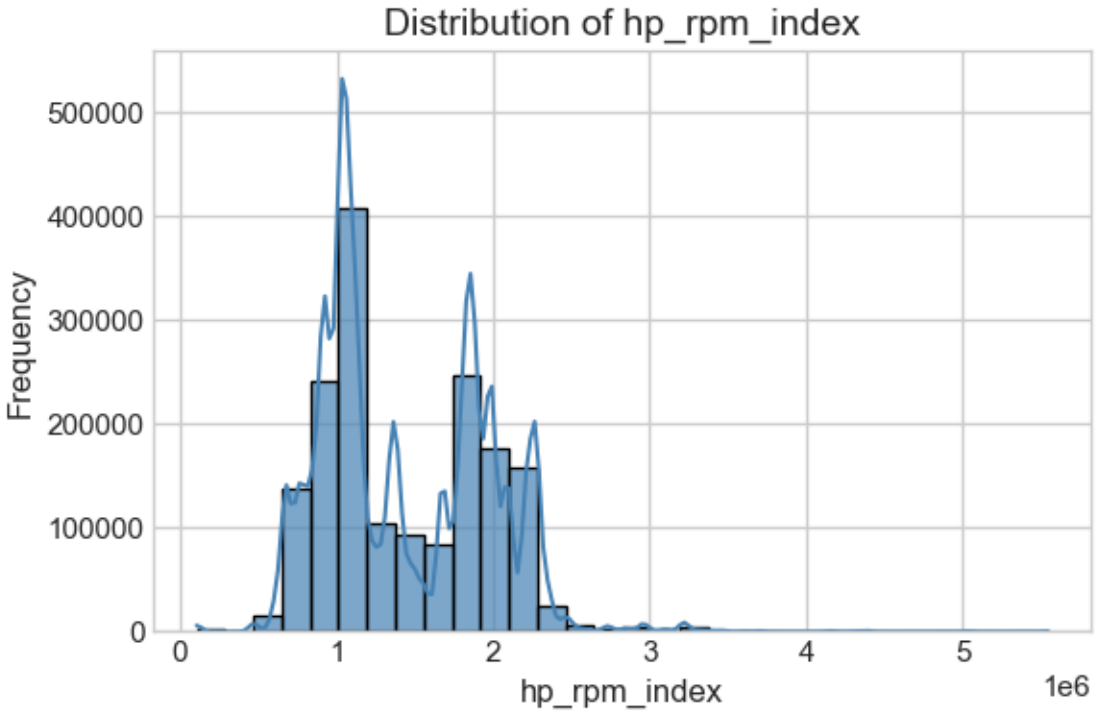

Distribution of savings_amount
1e7
2.0
1.5
1.0
0.5
0.0
Frequency
0
10000
20000
30000
40000
50000
60000
70000
savings_amount

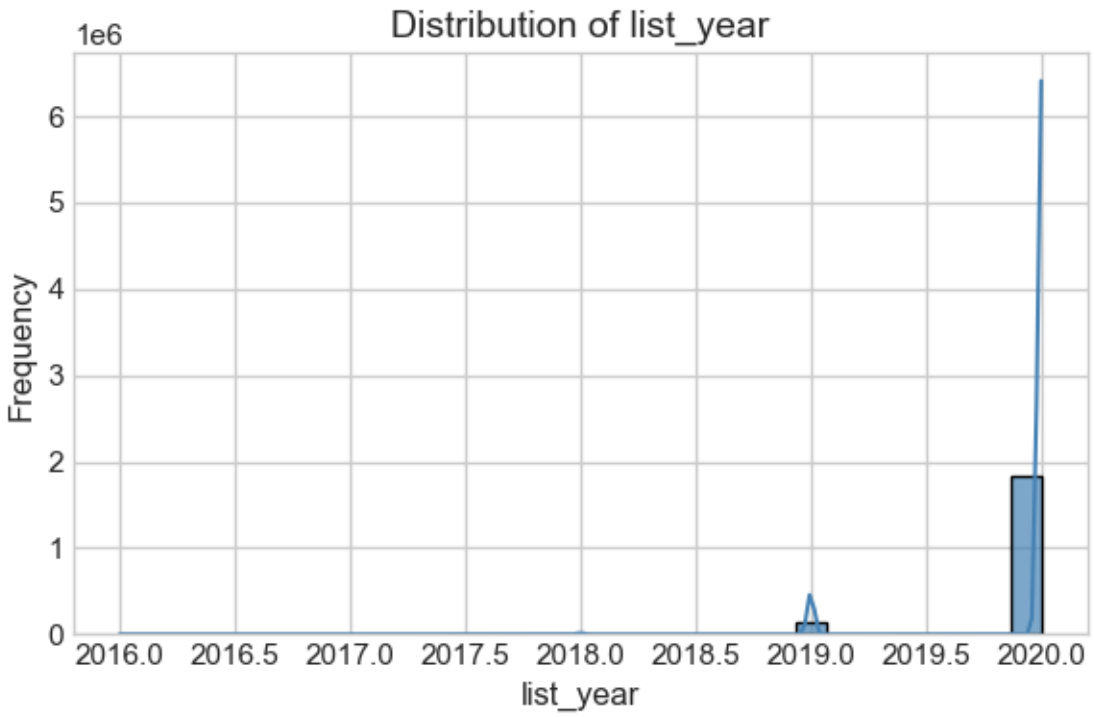

Distribution of seller_rating
300000
250000
200000
150000
100000
50000
0
Frequency
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
seller_rating

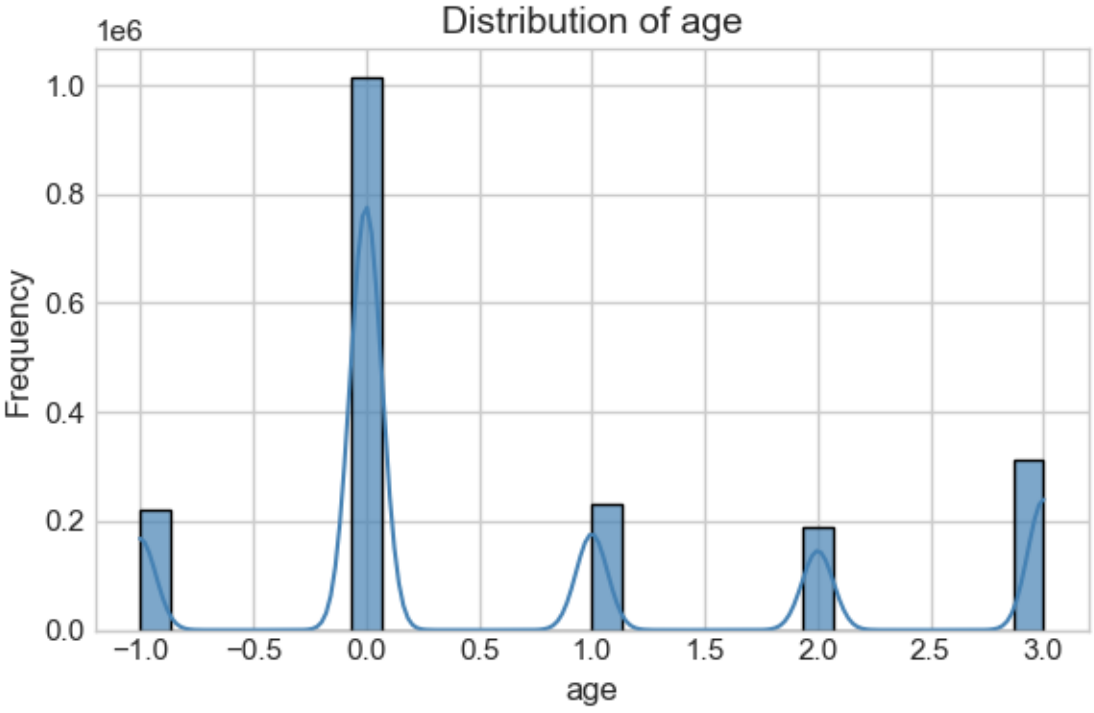

Distribution of wheelbase
Frequency
500000
400000
300000
200000
100000
0
100
120
140
160
180
wheelbase


Distribution of year
1e6
Frequency
1.0
0.8
0.6
0.4
0.2
0.0
2017.0
2017.5
2018.0
2018.5
2019.0
2019.5
2020.0
2020.5
2021.0
year

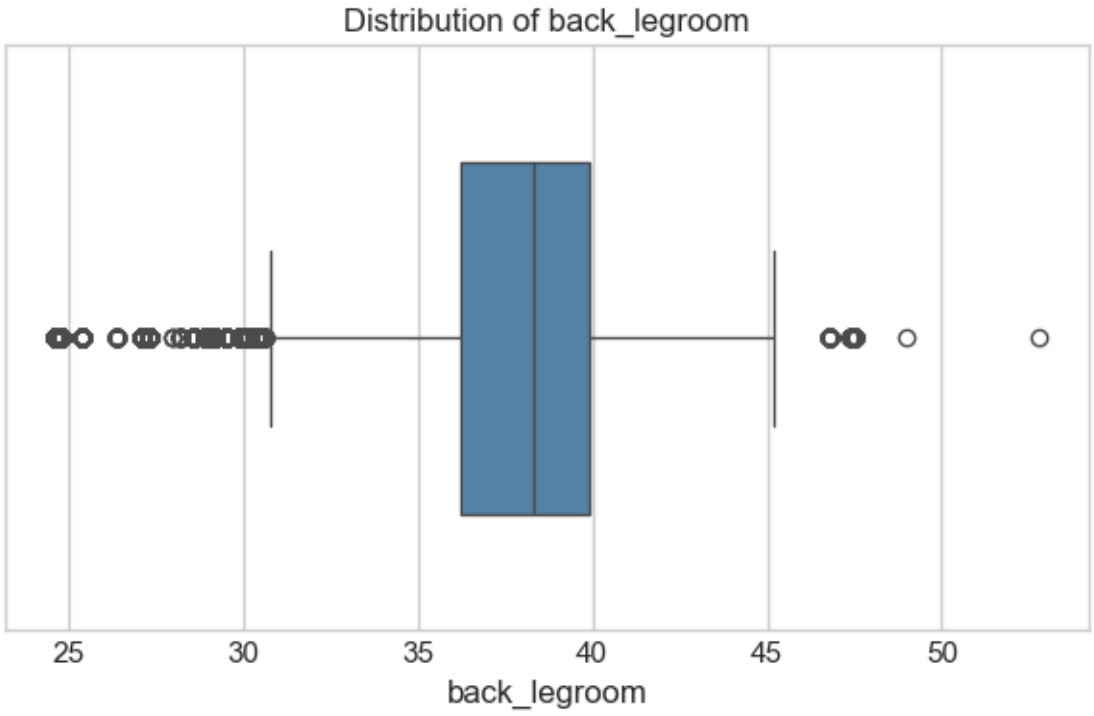

Distribution of tor_rpm_index
Frequency
250000
200000
150000
100000
50000
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
tor_rpm_index
1e6

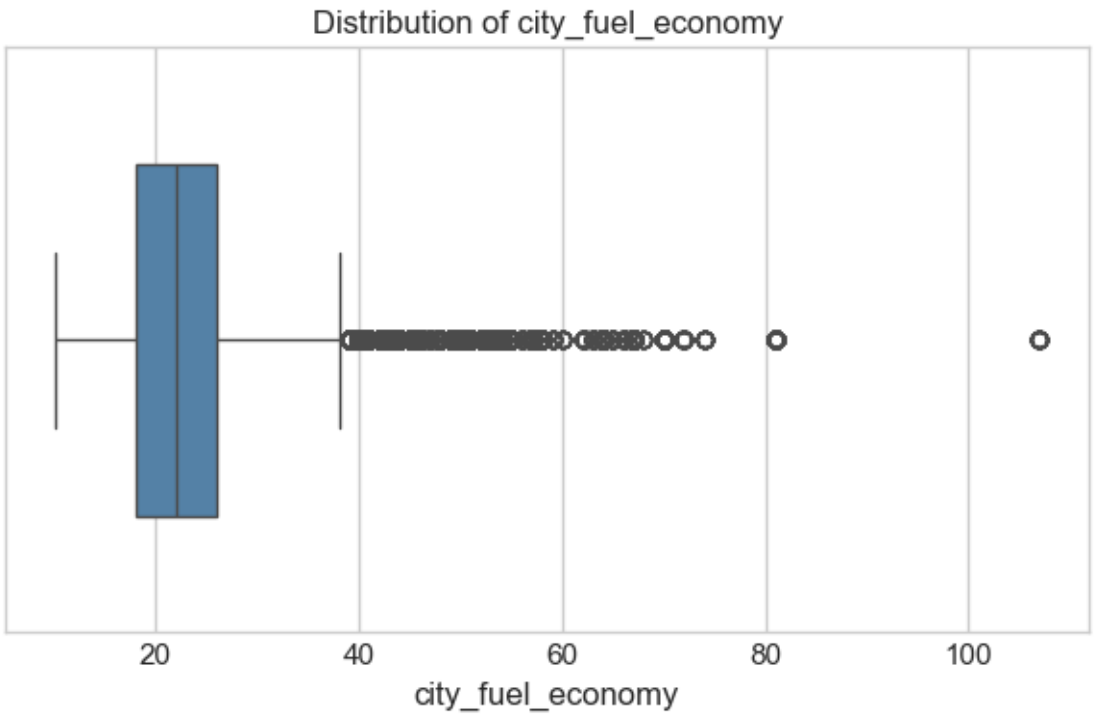

Distribution of hp_rpm_index
Frequency
500000
400000
300000
200000
100000
0
0
1
2
3
4
5
hp_rpm_index
1e6

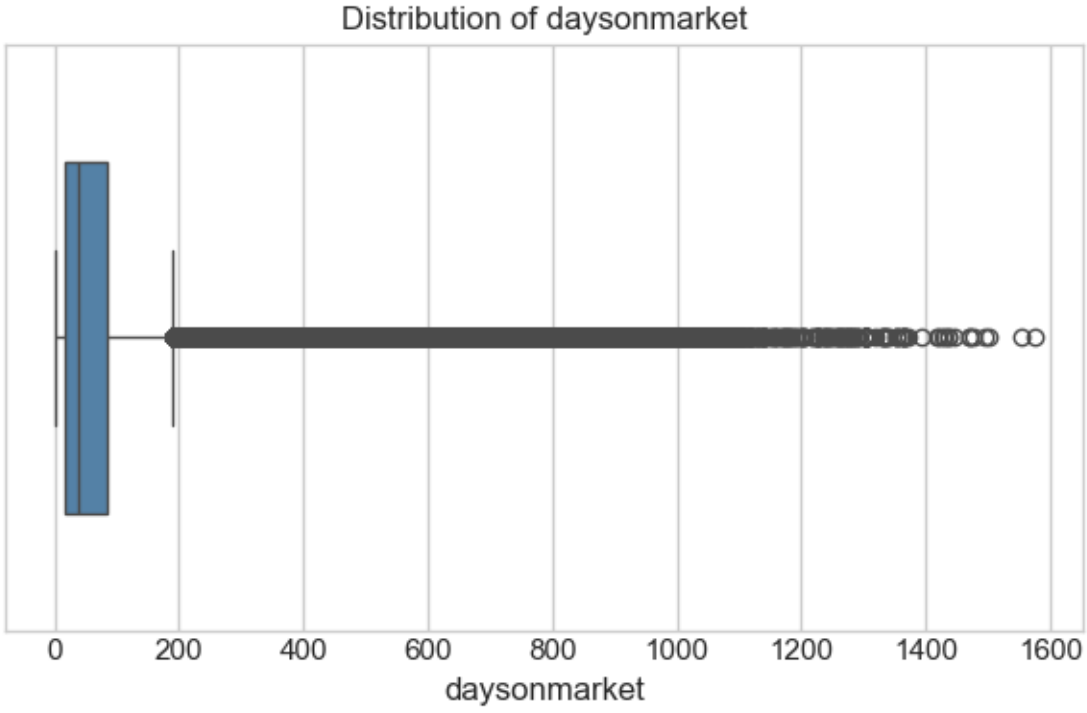

Distribution of list_year
1e6
6
5
4
3
2
1
0
Frequency
2016.0
2016.5
2017.0
2017.5
2018.0
2018.5
2019.0
2019.5
2020.0
list_year

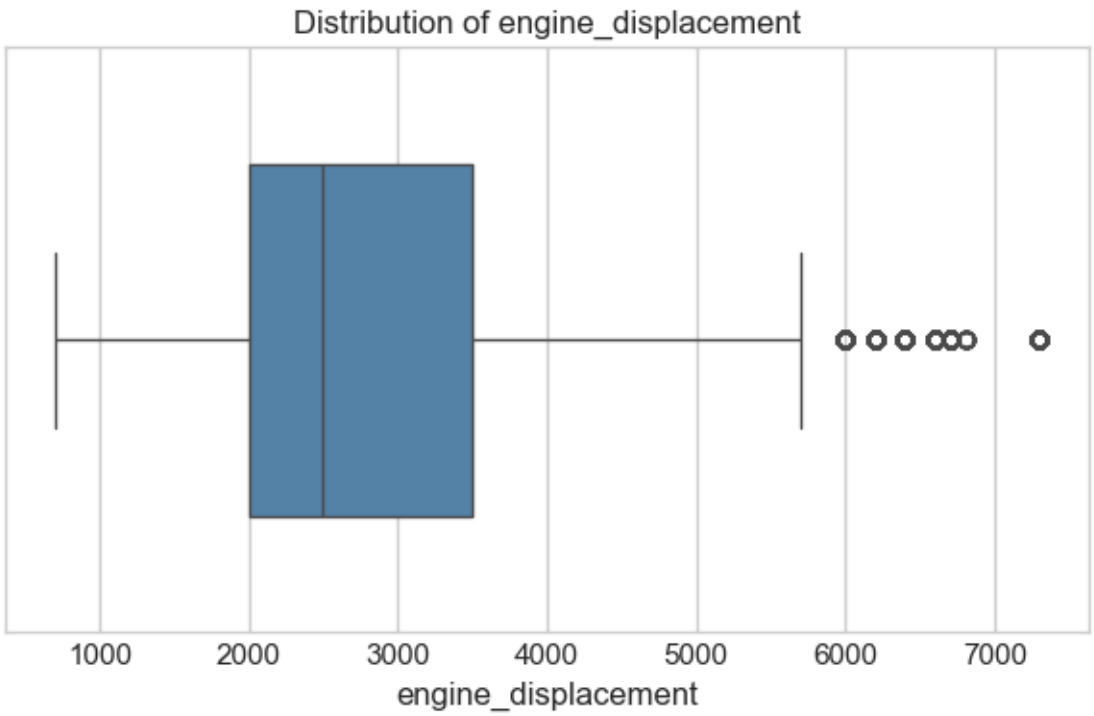

Distribution of age
1e6
1.0
0.8
0.6
0.4
0.2
0.0
Frequency
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0
age

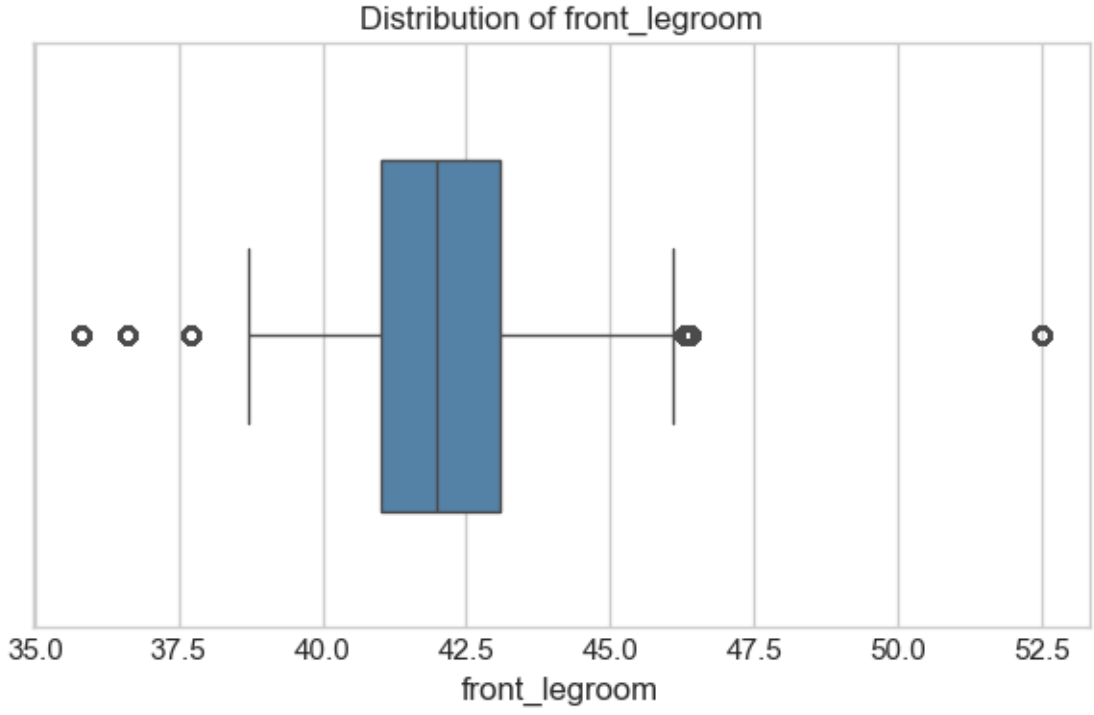
[62]:
```
# Identify the outliers in the data
for col in df2.select_dtypes(include='number').columns:
    plt.figure(figsize=(6, 4))
    sns.boxplot(data=df2, x=col, color='steelblue', width=0.6, showfliers=True)
    plt.title(f'Distribution of {col}', fontsize=12)
    plt.xlabel(col)
    plt.tight_layout()
    plt.show()
```

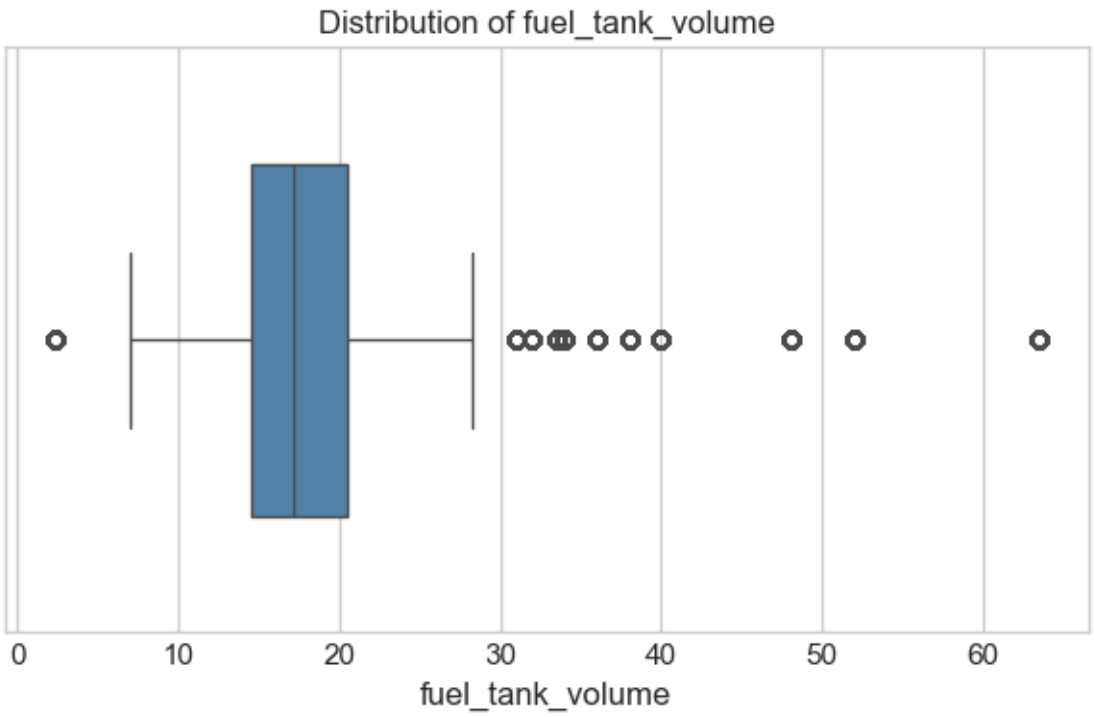


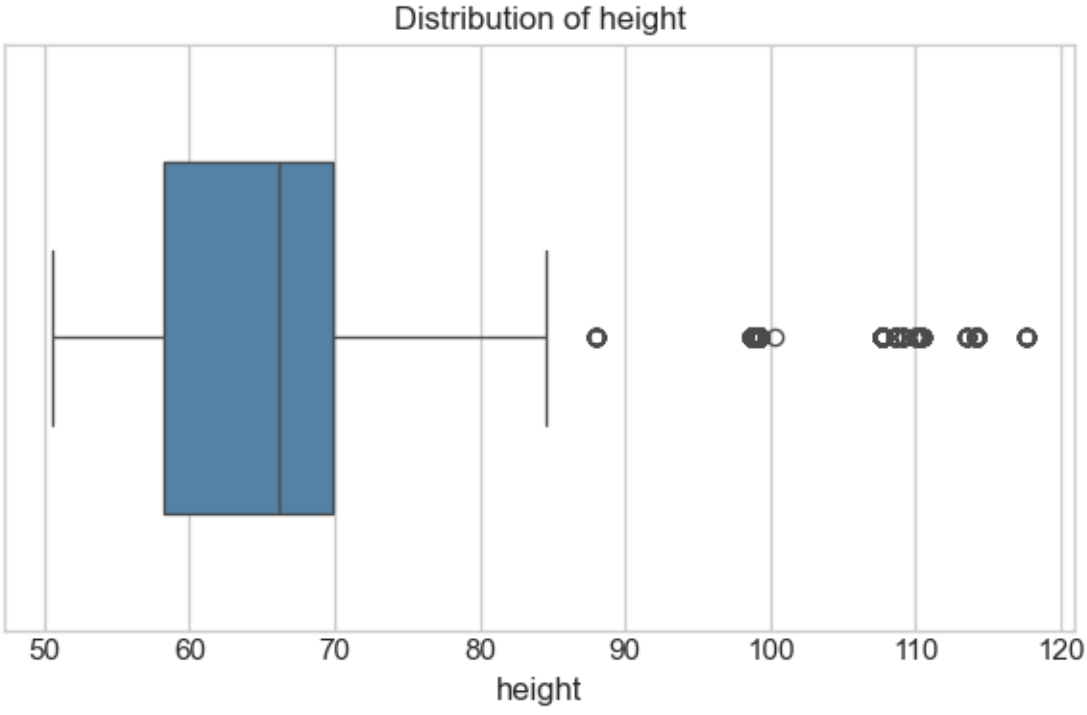

Distribution of city_fuel_economy
20
40
60
80
100
city_fuel_economy

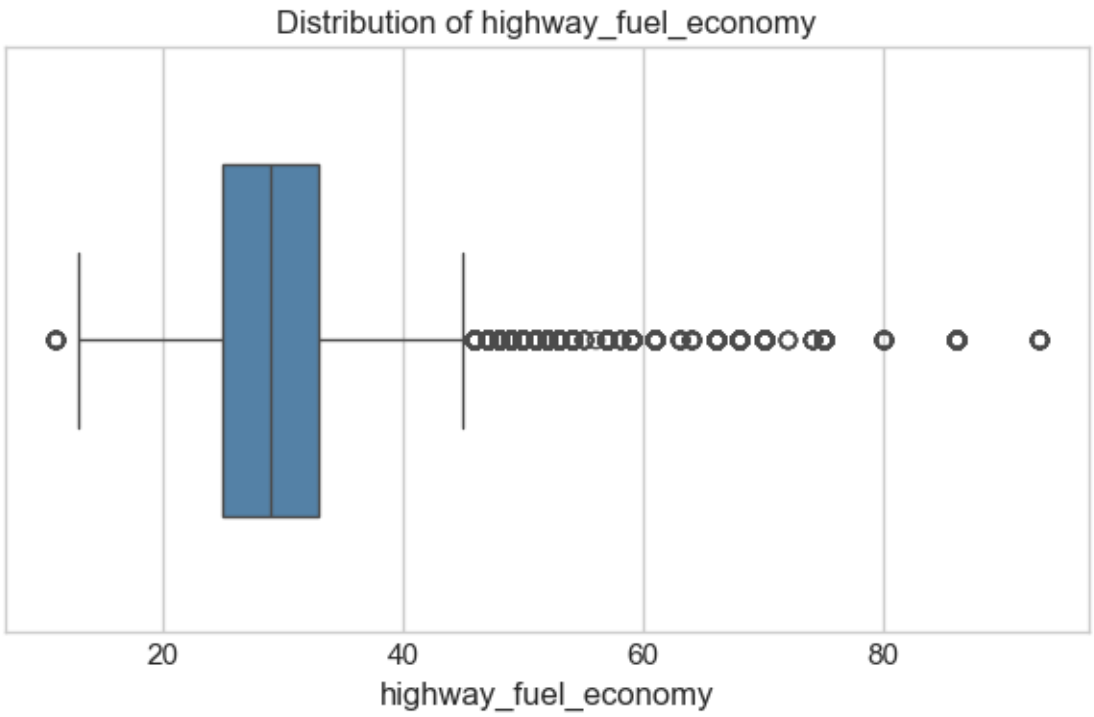

Distribution of daysonmarket
0
200
400
600
800
1000
1200
1400
1600
daysonmarket

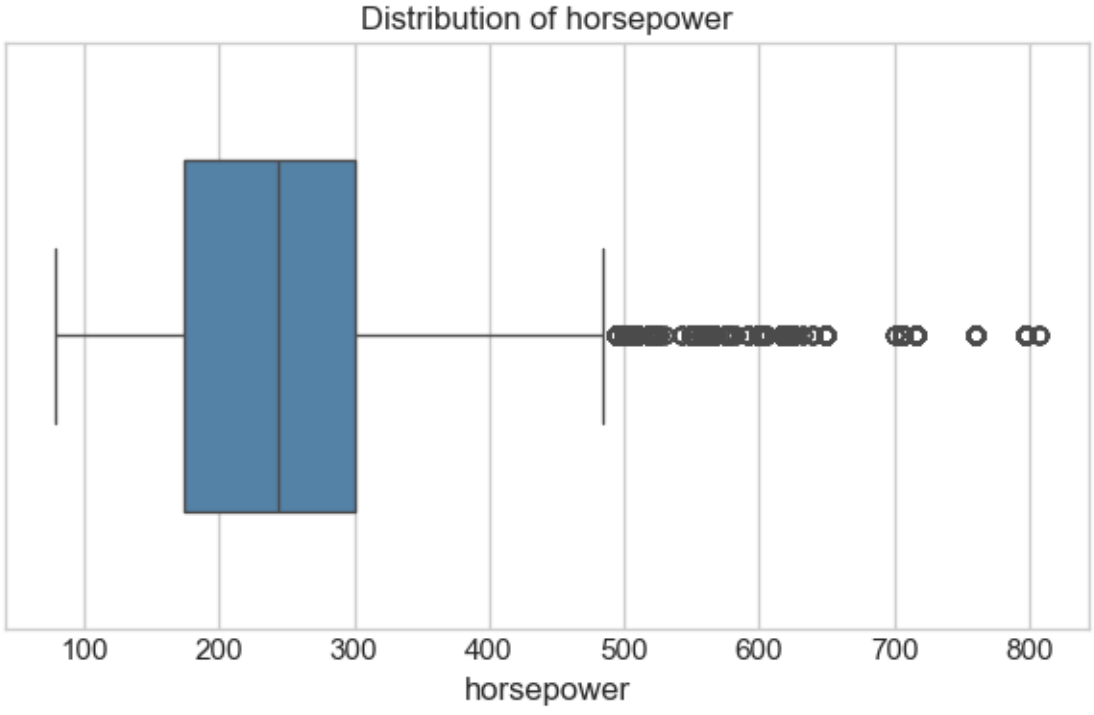

Distribution of engine_displacement
1000
2000
3000
4000
5000
6000
7000
engine_displacement

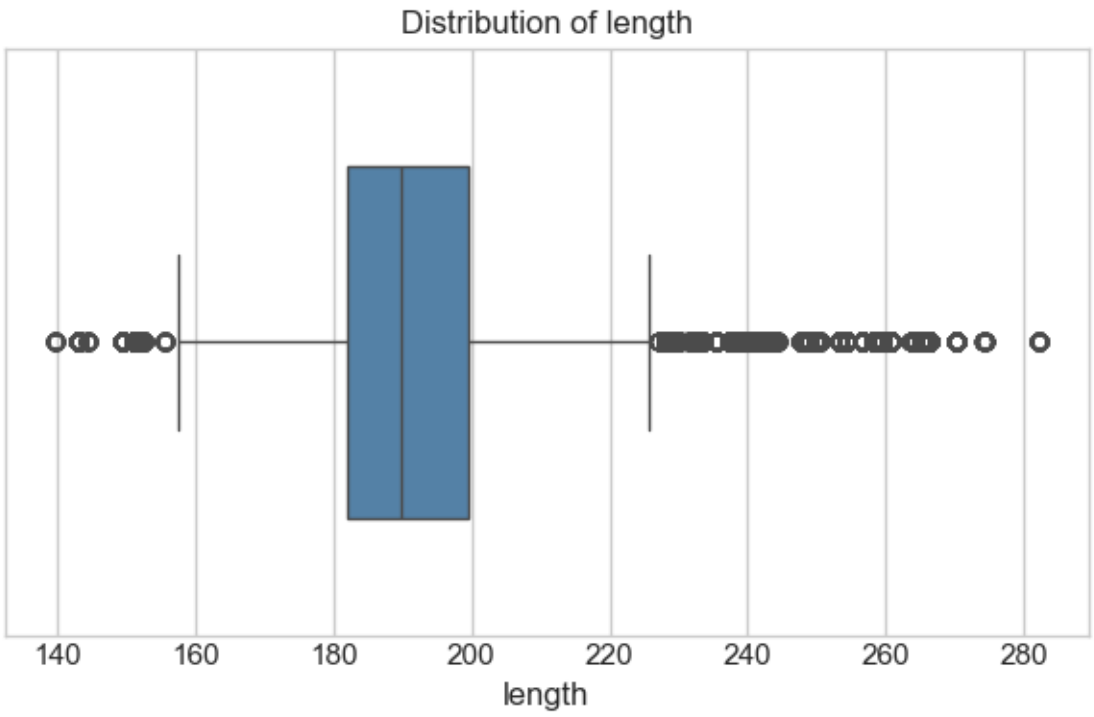

Distribution of front_legroom
35.0
37.5
40.0
42.5
45.0
47.5
50.0
52.5
front_legroom

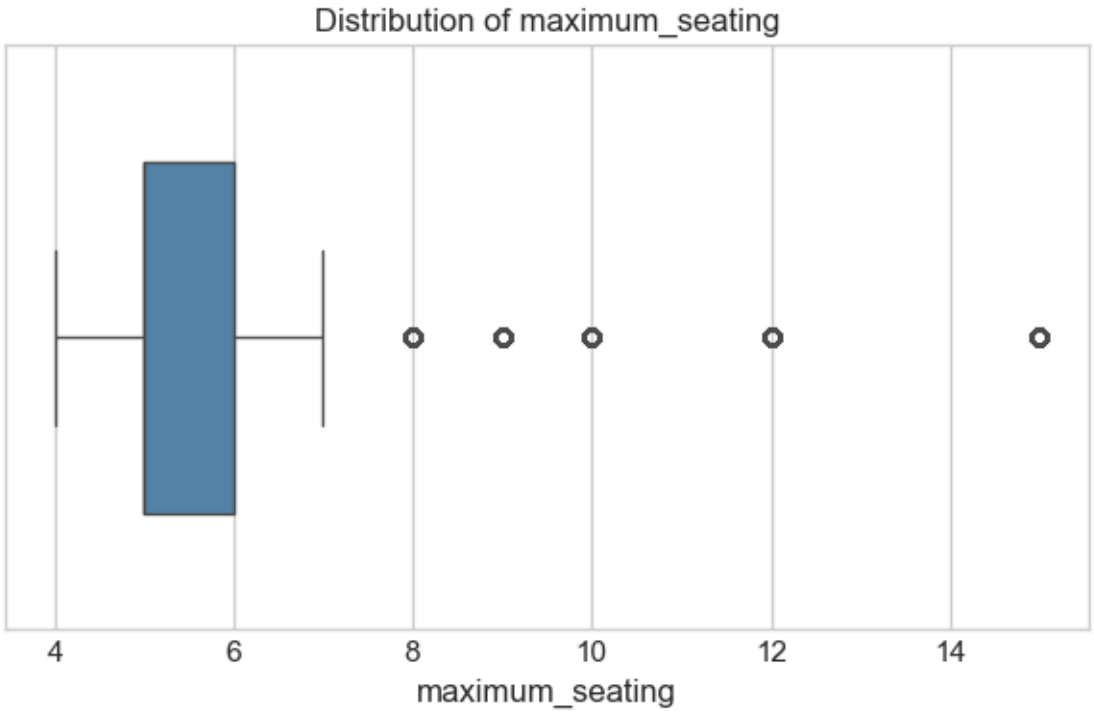

Distribution of fuel_tank_volume
0
10
20
30
40
50
60
fuel_tank_volume

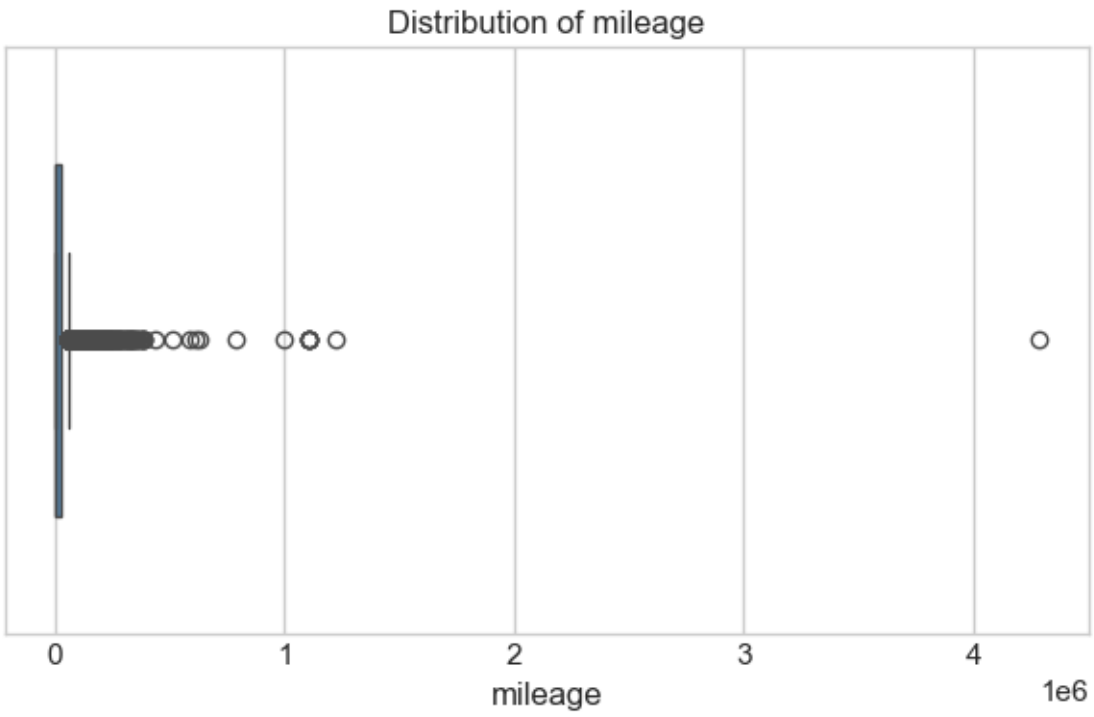

Distribution of height
50
60
70
80
90
100
110
120
height

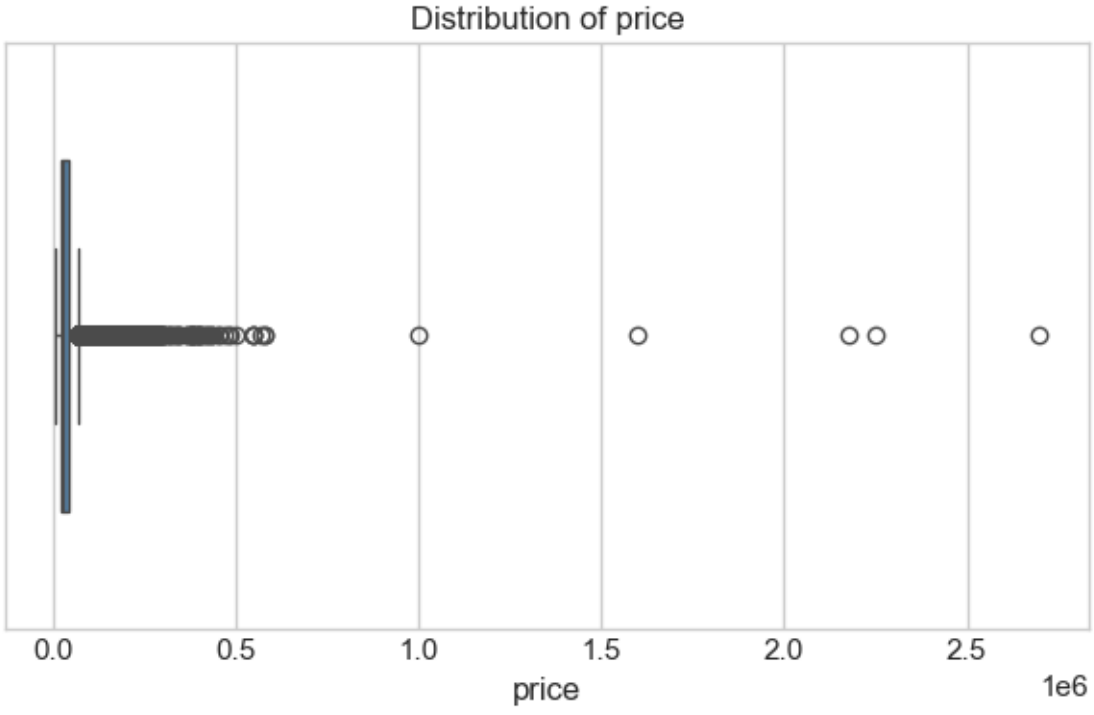

Distribution of highway_fuel_economy
20
40
60
80
highway_fuel_economy

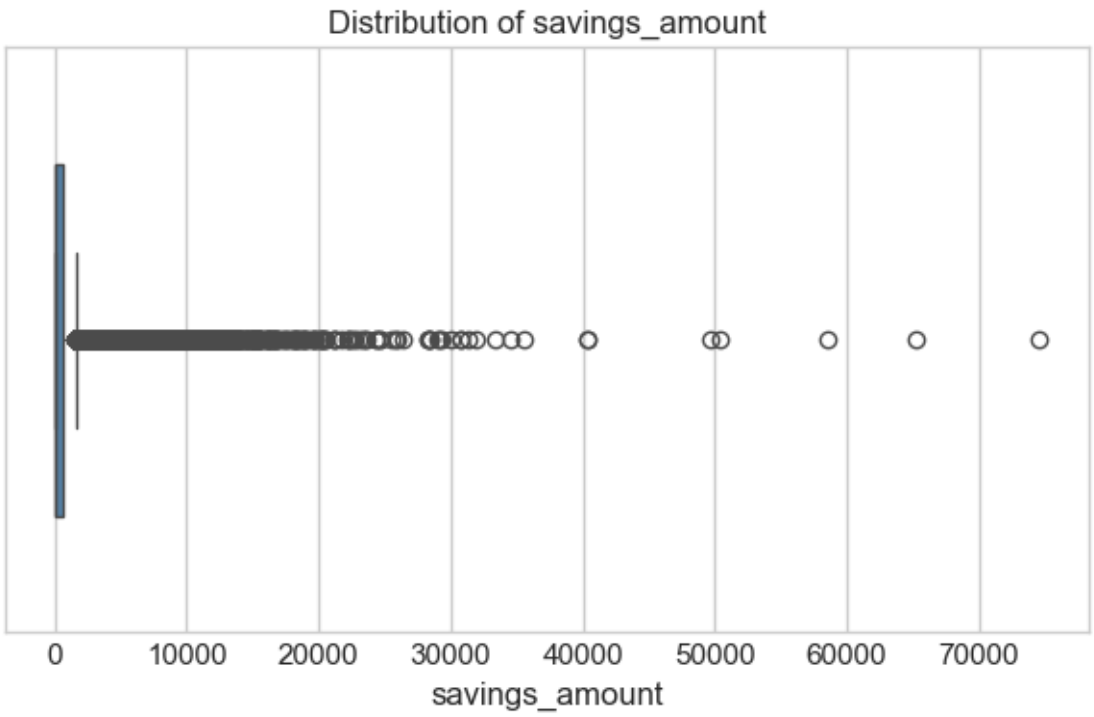

Distribution of horsepower
100
200
300
400
500
600
700
800
horsepower

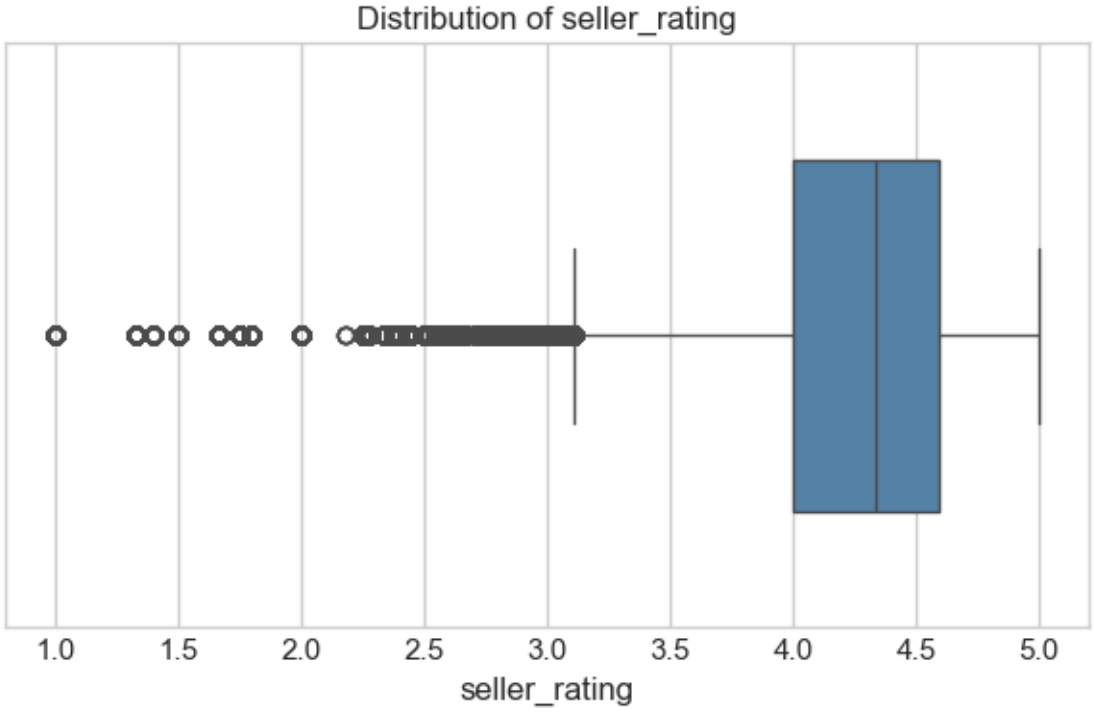

Distribution of length
140
160
180
200
220
240
260
280
length

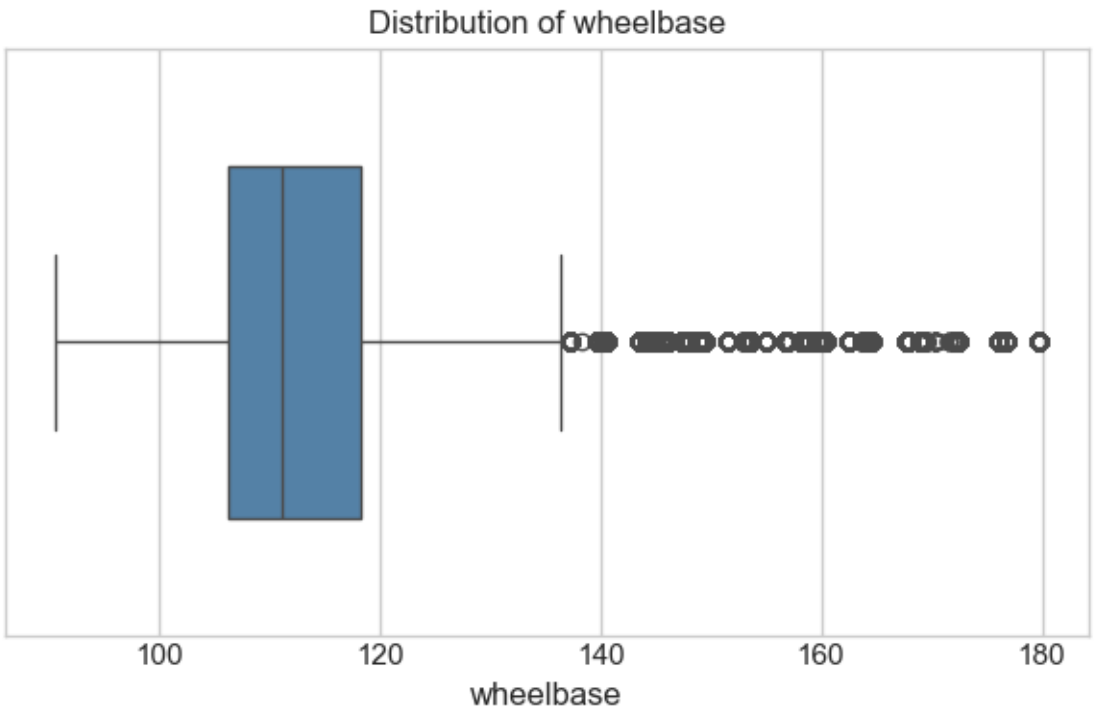

Distribution of maximum_seating
4
6
8
10
12
14
maximum_seating

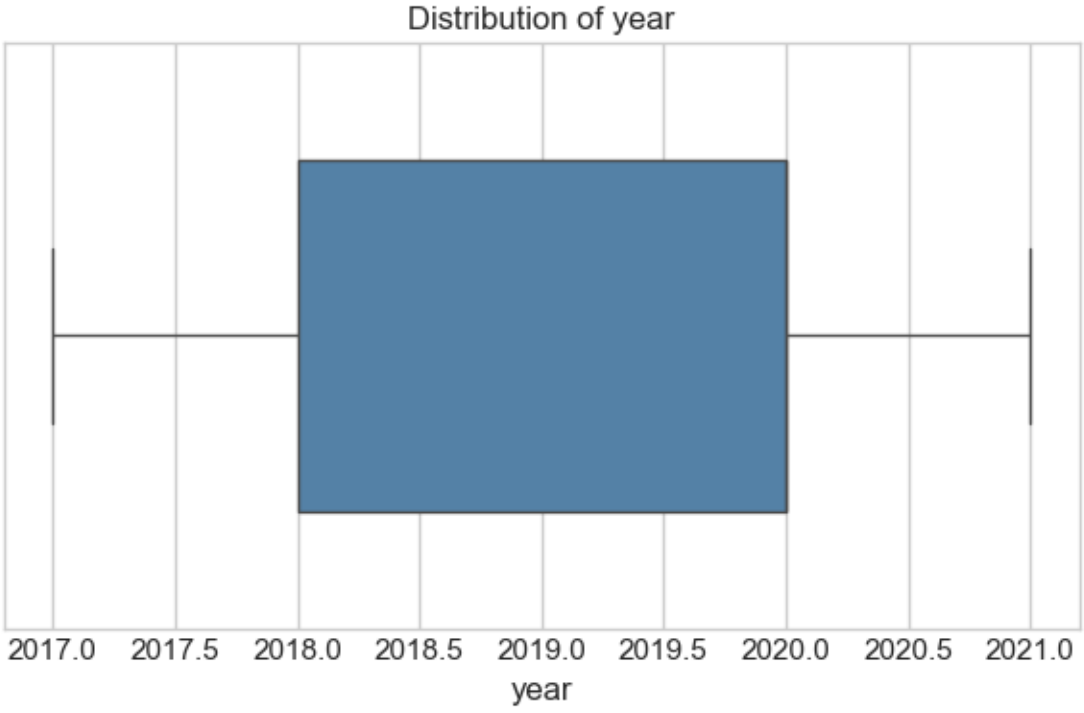

Distribution of mileage
0
1
2
3
4
mileage
1e6

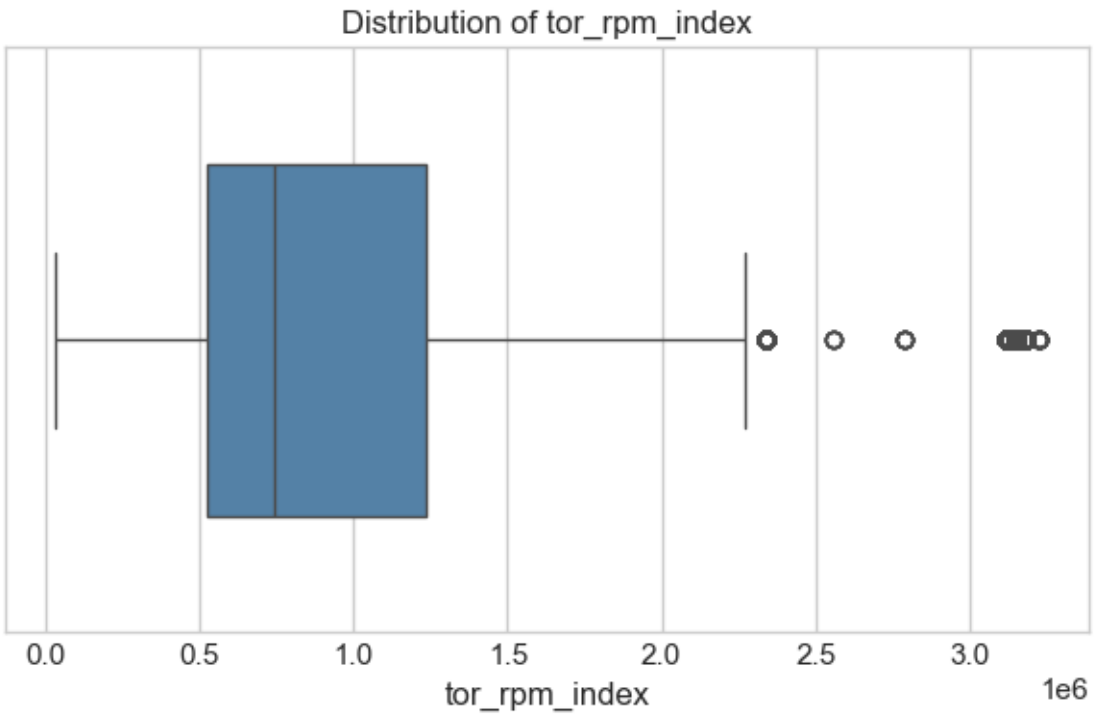

Distribution of price
0.0
0.5
1.0
1.5
2.0
2.5
price
1e6

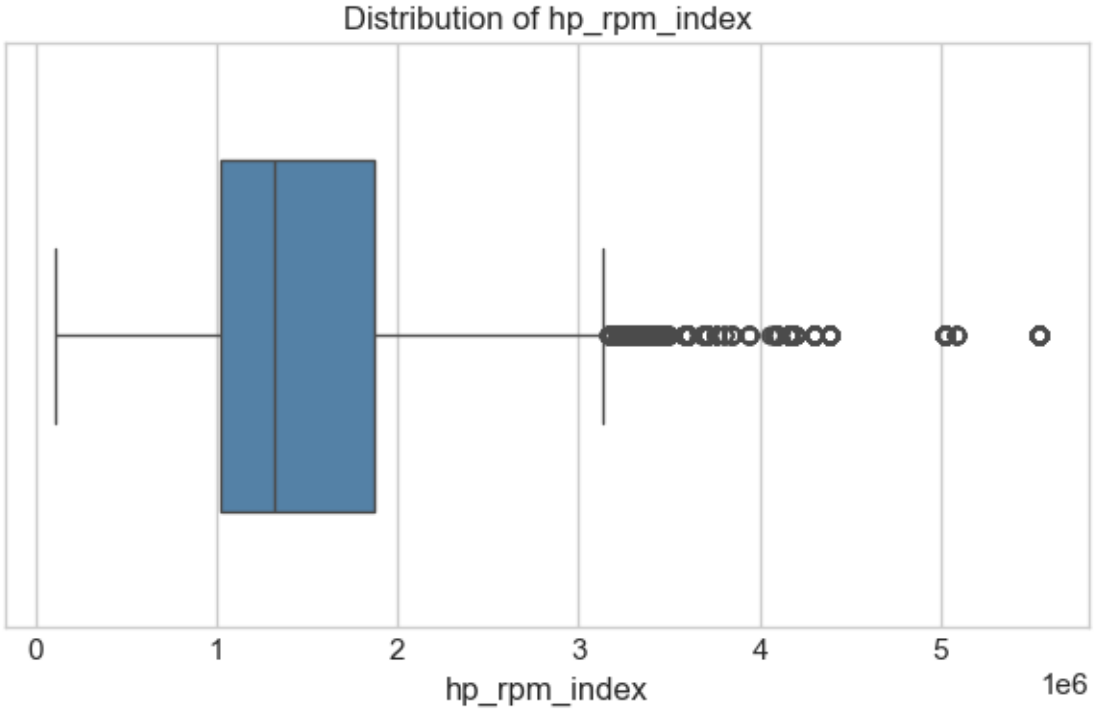

Distribution of savings_amount
0
10000
20000
30000
40000
50000
60000
70000
savings_amount

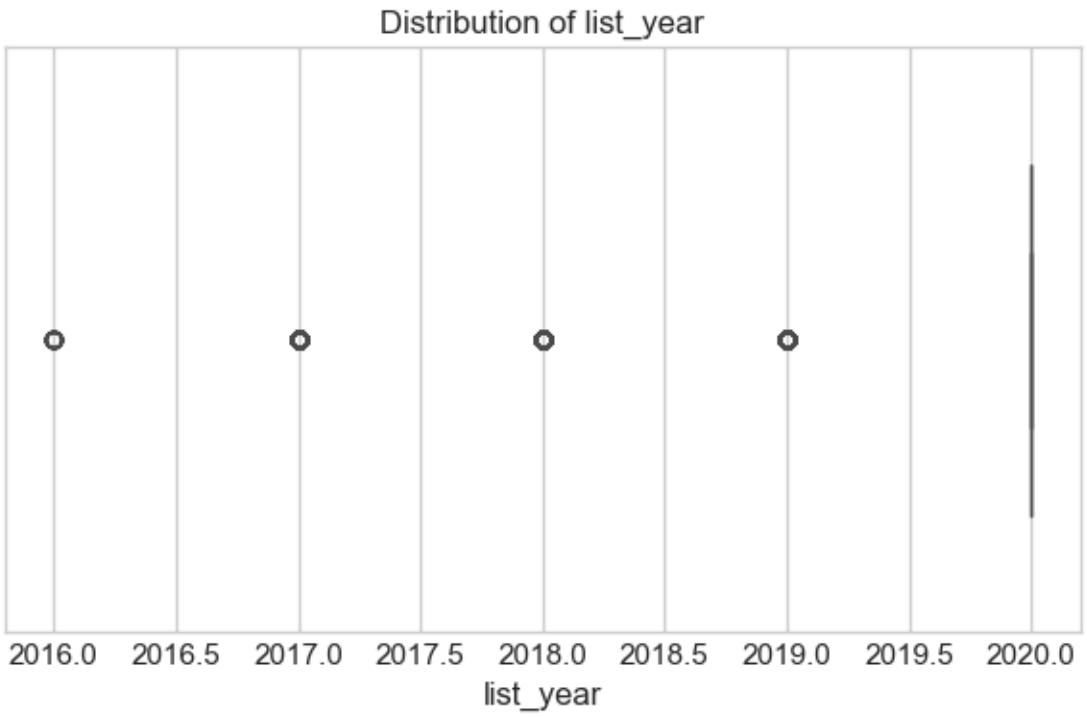

Distribution of seller_rating
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
seller_rating

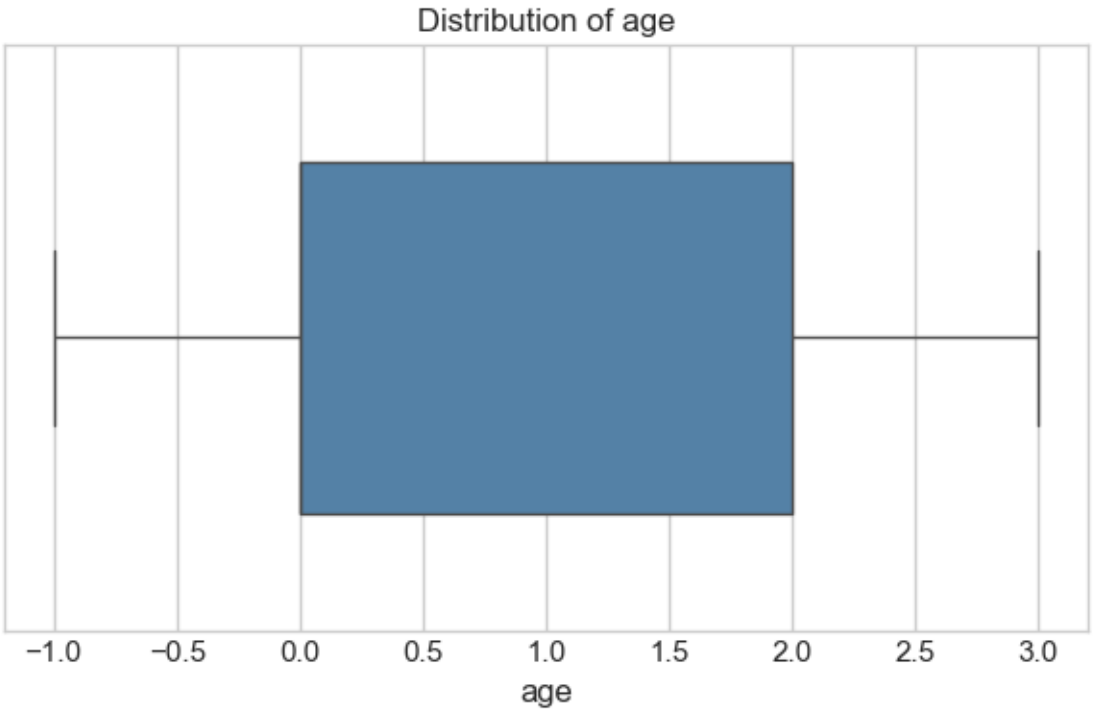

Distribution of wheelbase
100
120
140
160
180
wheelbase


Distribution of year
2017.0
2017.5
2018.0
2018.5
2019.0
2019.5
2020.0
2020.5
2021.0
year


Distribution of tor_rpm_index
0.0
0.5
1.0
1.5
2.0
2.5
3.0
tor_rpm_index
1e6


Distribution of hp_rpm_index
0
1
2
3
4
5
hp_rpm_index
1e6





I will check outliers of the following variables

[63]:
```
# Check some of the outliers.
df2.sort_values(by='mileage', ascending = False).head(30)

#'back_legroom','city_fuel_economy', 'daysonmarket', 'engine_displacement',␣
 ↪'fuel_tank_volume',
#'height', 'highway_fuel_economy', 'horsepower', 'length', 'maximum_seating',␣
 ↪'savings_amount','wheelbase'
#'tor_rpm_index', 'hp_rpm_index'
```

[63]:

	vin	back_legroom	body_type	city	\
140200	1GCUYDED6LZ367093	43.40	Pickup Truck	Nappanee	
863939	1FTEW1CP3LFC54483	43.60	Pickup Truck	Belleview	
1353391	1GCRYDED4LZ304807	35.20	Pickup Truck	Boise	
1353952	1GYKNBR40LZ189271	39.00	SUV / Crossover	Boise	
1352579	3GNAXUEV9LS700051	39.90	SUV / Crossover	Boise	
1353061	1GCRYDED1LZ303520	35.20	Pickup Truck	Boise	
1352531	1GCRYDED8LZ312540	35.20	Pickup Truck	Boise	
1543669	KMHH35LE6LU131604	34.80	Hatchback	Kingman	
42332	KL4CJESB6LB048238	35.80	SUV / Crossover	Englewood	
1807613	1FTEW1E4XLKE38250	43.60	Pickup Truck	Grand Junction	
1800556	1C4RJFBG8LC384050	38.60	SUV / Crossover	Aurora	
1175746	1GB4YSE73LF300181	43.40	Pickup Truck	Louisville	
338257	2GNAXVEX4L6267065	39.90	SUV / Crossover	Bath	
154516	1C4PJMDX3LD629162	40.30	SUV / Crossover	Portsmouth	
1967018	1N4BL4BVXKC231964	35.20	Sedan	Vacaville	
597412	1N4AL3AP3HC186514	36.10	Sedan	Conover	
1726032	5FNRL6H57JB024736	40.90	Minivan	Corpus Christi	
43903	5GAEVAKW6LJ115958	38.90	SUV / Crossover	Englewood	
1429394	1VWBT7A34HC023192	39.10	Sedan	Phoenix	
1070032	1FAHP2F83KG115439	38.10	Sedan	Nashville	
1791012	MAJ6S3JL4LC330067	36.70	SUV / Crossover	Colorado Springs	
1684600	1G1ZD5ST5LF005169	38.10	Sedan	Reno	
1421470	1FMCU9H63LUB07373	40.70	SUV / Crossover	Burlington	
1804893	1FMCU9F67LUA10969	40.70	SUV / Crossover	Cortez	
396063	1FMCU9G92KUB83421	37.30	SUV / Crossover	White Hall	
150648	5XYPGDA37HG276448	39.40	SUV / Crossover	Swansea	
968273	2GKALMEK6H6317575	39.90	SUV / Crossover	West Palm Beach	
1070023	KNDJP3A56K7666818	39.10	Wagon	Nashville	
230062	1C6RR7TT8KS732242	40.30	Pickup Truck	Glassboro	
1651544	2C4RDGCG9KR557026	36.50	Minivan	Humble	

	city_fuel_economy	daysonmarket	engine_cylinders	\
140200	16.00	4	V8	
863939	20.00	11	V6	
1353391	16.00	56	V8	
1353952	21.00	198	I4	
1352579	NaN	61	I4	

1353061	16.00	71	V8
1352531	16.00	63	V8
1543669	25.00	114	I4
42332	24.00	234	I4
1807613	16.00	48	V6
1800556	18.00	10	V6
1175746	NaN	36	V8
338257	22.00	15	I4
154516	19.00	71	V6
1967018	28.00	15	I4
597412	27.00	727	I4
1726032	19.00	24	V6
43903	17.00	424	V6
1429394	23.00	30	I4
1070032	18.00	78	V6
1791012	23.00	59	I4
1684600	29.00	41	I4
1421470	26.00	28	I3
1804893	26.00	330	I3
396063	21.00	14	I4
150648	21.00	188	I4
968273	21.00	49	I4
1070023	25.00	78	I4
230062	15.00	8	V8
1651544	12.00	66	V6 Flex Fuel Vehicle

	engine_displacement	franchise_dealer	franchise_make	front_legroom	\
140200	5300.00	yes	Chevrolet	44.50	
863939	3500.00	yes	Ford	43.90	
1353391	5300.00	yes	Chevrolet	44.50	
1353952	2000.00	yes	Cadillac	41.20	
1352579	1500.00	yes	Chevrolet	40.90	
1353061	5300.00	yes	Chevrolet	44.50	
1352531	5300.00	yes	Chevrolet	44.50	
1543669	2000.00	yes	Hyundai	42.20	
42332	1400.00	yes	Buick	40.80	
1807613	3500.00	yes	Ford	43.90	
1800556	3600.00	yes	Jeep	40.30	
1175746	6600.00	yes	Chevrolet	44.50	
338257	1500.00	yes	Chevrolet	40.90	
154516	3200.00	yes	Jeep	41.10	
1967018	2500.00	yes	Jeep	43.80	
597412	2500.00	no	NaN	45.00	
1726032	3500.00	yes	Buick	40.90	
43903	3600.00	yes	Buick	41.20	
1429394	1800.00	yes	Volkswagen	42.40	
1070032	3500.00	no	NaN	44.20	

1791012	2000.00	yes	Ford	42.90
1684600	1500.00	no	NaN	41.50
1421470	1500.00	yes	Ford	42.40
1804893	1500.00	yes	Ford	42.40
396063	2000.00	yes	Ford	43.10
150648	2400.00	yes	Kia	44.10
968273	2400.00	yes	Cadillac	41.20
1070023	2000.00	no	NaN	40.90
230062	5700.00	yes	Mitsubishi	41.00
1651544	3600.00	yes	Chevrolet	40.70

	fuel_tank_volume	fuel_type	height	highway_fuel_economy	\
140200	24.00	Gasoline	75.50	22.00	
863939	26.00	Gasoline	75.60	26.00	
1353391	24.00	Gasoline	75.60	22.00	
1353952	22.00	Gasoline	66.10	26.00	
1352579	15.60	Gasoline	65.40	NaN	
1353061	24.00	Gasoline	75.60	22.00	
1352531	24.00	Gasoline	75.60	22.00	
1543669	14.00	Gasoline	57.70	32.00	
42332	14.00	Gasoline	65.30	29.00	
1807613	26.00	Gasoline	77.20	22.00	
1800556	24.60	Gasoline	69.30	25.00	
1175746	63.50	Gasoline	80.10	NaN	
338257	15.60	Gasoline	65.40	28.00	
154516	15.90	Gasoline	66.20	27.00	
1967018	16.20	Gasoline	56.70	39.00	
597412	18.00	Gasoline	57.80	39.00	
1726032	19.50	Gasoline	69.60	28.00	
43903	21.90	Gasoline	69.90	25.00	
1429394	18.50	Gasoline	58.50	34.00	
1070032	19.00	Gasoline	60.70	26.00	
1791012	13.60	Gasoline	65.10	29.00	
1684600	15.80	Gasoline	57.90	36.00	
1421470	14.70	Gasoline	68.60	31.00	
1804893	14.70	Gasoline	66.10	31.00	
396063	15.70	Gasoline	66.30	27.00	
150648	18.80	Gasoline	66.50	25.00	
968273	18.80	Gasoline	66.30	31.00	
1070023	14.20	Gasoline	63.50	30.00	
230062	26.00	Gasoline	78.40	21.00	
1651544	20.00	Flex Fuel Vehicle	69.00	18.00	

	horsepower	length	listing_color	make_name	maximum_seating	\
140200	355.00	231.70	GRAY	Chevrolet	6.00	
863939	375.00	231.90	UNKNOWN	Ford	6.00	
1353391	355.00	231.70	WHITE	Chevrolet	6.00	

```
1353952      237.00  189.60         WHITE    Cadillac             5.00
1352579      170.00  183.10         WHITE   Chevrolet             5.00
1353061      355.00  231.70       UNKNOWN   Chevrolet             6.00
1352531      355.00  231.70        SILVER   Chevrolet             6.00
1543669      161.00  170.90         WHITE     Hyundai             5.00
42332        138.00  168.40       UNKNOWN       Buick             5.00
1807613      375.00  231.90         WHITE        Ford             6.00
1800556      295.00  189.80         WHITE        Jeep             5.00
1175746      445.00  270.30         WHITE   Chevrolet             6.00
338257       170.00  183.10          GRAY   Chevrolet             5.00
154516       271.00  182.00        SILVER        Jeep             5.00
1967018      188.00  192.90        SILVER      Nissan             5.00
597412       182.00  191.90        SILVER      Nissan             5.00
1726032      280.00  203.20          GRAY       Honda             8.00
43903        310.00  204.30       UNKNOWN       Buick             7.00
1429394      170.00  191.90        SILVER  Volkswagen             5.00
1070032      288.00  202.90         BLACK        Ford             5.00
1791012      166.00  161.30          BLUE        Ford             5.00
1684600      160.00  194.20          GRAY   Chevrolet             5.00
1421470      180.00  180.50         WHITE        Ford             5.00
1804893      180.00  180.50         WHITE        Ford             5.00
396063       245.00  178.10           RED        Ford             5.00
150648       185.00  187.40         BLACK         Kia             7.00
968273       182.00  185.50         WHITE         GMC             5.00
1070023      161.00  163.00          GRAY         Kia             5.00
230062       395.00  237.90          GRAY         RAM             6.00
1651544      283.00  203.70         WHITE       Dodge             7.00

          mileage               model_name    price  savings_amount  \
140200  4290461.00           Silverado 1500 50775.00               0
863939  1225238.00                    F-150 45149.00               0
1353391 1111111.00           Silverado 1500 41420.00               0
1353952 1111111.00                      XT5 45970.00               0
1352579 1111111.00                  Equinox 23620.00               0
1353061 1111111.00           Silverado 1500 41420.00               0
1352531 1111111.00           Silverado 1500 39523.00               0
1543669  999999.00               Elantra GT 22907.00               0
42332    785778.00                   Encore 28230.00               0
1807613  631835.00                    F-150 51427.00               0
1800556  610288.00           Grand Cherokee 42763.00               0
1175746  590072.00  Silverado 3500HD Chassis 45288.00               0
338257   514033.00                  Equinox 27999.00               0
154516   434718.00                 Cherokee 36585.00               0
1967018  387347.00                   Altima 19281.00               0
597412   385729.00                   Altima 15295.00               0
1726032  384820.00                  Odyssey 24507.00               0
43903    381519.00                  Enclave 45765.00               0
```

1429394	379647.00	Passat	7900.00	0	
1070032	373109.00	Taurus	17890.00	0	
1791012	360012.00	EcoSport	21499.00	0	
1684600	357313.00	Malibu	23988.00	0	
1421470	353935.00	Escape	28995.00	0	
1804893	352910.00	Escape	27580.00	0	
396063	350031.00	Escape	18698.00	0	
150648	341690.00	Sorento	18644.00	0	
968273	336317.00	Terrain	15995.00	0	
1070023	334971.00	Soul	12890.00	0	
230062	332105.00	1500	35495.00	0	
1651544	331297.00	Grand Caravan	17498.00	0	

	seller_rating	transmission	trim_name	\
140200	5.00	A	LT Crew Cab 4WD	
863939	4.50	A	XLT SuperCrew RWD	
1353391	3.82	A	LT Double Cab 4WD	
1353952	3.82	A	Luxury AWD	
1352579	3.82	A	1.5T LT AWD	
1353061	3.82	A	LT Double Cab 4WD	
1352531	3.82	A	LT Double Cab 4WD	
1543669	3.50	A	FWD	
42332	4.29	A	Preferred AWD	
1807613	4.13	A	XLT SuperCrew 4WD	
1800556	4.43	A	Limited 4WD	
1175746	4.40	A	Work Truck Crew Cab 4WD	
338257	4.56	A	2.0T LT AWD	
154516	4.44	A	Limited 4WD	
1967018	3.50	CVT	2.5 S FWD	
597412	4.69	A	2.5 SV	
1726032	4.57	A	EX FWD	
43903	4.29	A	Essence AWD	
1429394	3.00	A	1.8T SE	
1070032	4.41	A	Limited FWD	
1791012	4.15	A	SES AWD	
1684600	3.87	CVT	LT FWD	
1421470	3.50	A	SEL AWD	
1804893	4.67	A	S AWD	
396063	4.09	A	SE AWD	
150648	4.17	A	LX AWD	
968273	5.00	A	SLE1	
1070023	4.41	A	+ FWD	
230062	4.39	A	Classic Big Horn Crew Cab LB 4WD	
1651544	3.71	A	SXT FWD	

	wheel_system	wheelbase	year	tor_rpm_index	hp_rpm_index	list_year	\
140200	4WD	147.40	2020	1570300	1988000	2020	

863939	4X2	145.00	2020	1800000	2271250	2020
1353391	4WD	147.50	2020	1570300	1988000	2020
1353952	4WD	112.50	2020	387000	1185000	2020
1352579	4WD	107.30	2020	406000	952000	2020
1353061	4WD	147.50	2020	1570300	1988000	2020
1352531	4WD	147.50	2020	1570300	1988000	2020
1543669	FWD	104.30	2020	705000	998200	2020
42332	AWD	100.60	2020	29600	676200	2020
1807613	4WD	145.00	2020	1800000	2271250	2020
1800556	4WD	114.70	2020	1657500	1854000	2020
1175746	4WD	176.80	2020	1856000	2085200	2020
338257	4WD	107.30	2020	406000	952000	2020
154516	4WD	106.60	2020	885000	1417500	2020
1967018	FWD	111.20	2019	648000	1128000	2020
597412	FWD	109.30	2017	720000	1092000	2018
1726032	FWD	118.10	2018	1231400	1680000	2020
43903	4WD	120.90	2020	744800	2108000	2019
1429394	FWD	110.40	2017	276000	1054000	2020
1070032	FWD	112.90	2019	1016000	1872000	2020
1791012	AWD	99.20	2020	663050	996000	2020
1684600	FWD	111.40	2020	460000	912000	2020
1421470	AWD	106.70	2020	<NA>	<NA>	2020
1804893	AWD	106.70	2020	<NA>	<NA>	2019
396063	AWD	105.90	2019	825000	1347500	2020
150648	AWD	109.40	2017	712000	1110000	2020
968273	FWD	112.50	2017	842800	1219400	2020
1070023	FWD	101.20	2019	705000	998200	2020
230062	4WD	149.50	2019	1619500	2212000	2020
1651544	FWD	121.20	2019	1144000	1811200	2020

	age
140200	0
863939	0
1353391	0
1353952	0
1352579	0
1353061	0
1352531	0
1543669	0
42332	0
1807613	0
1800556	0
1175746	0
338257	0
154516	0
1967018	1
597412	1

```
1726032    2
43903     -1
1429394    3
1070032    1
1791012    0
1684600    0
1421470    0
1804893   -1
396063     1
150648     3
968273     3
1070023    1
230062     1
1651544    1
```

[64]:
```
df2.drop(df2['mileage'].nlargest(8).index, inplace=True) # Some of the top values for mileage looks odd. I will drop them out.
```

[65]:
```
df2.sort_values(by='savings_amount', ascending = False).head(10)
```

[65]:

	vin	back_legroom	body_type	city	city_fuel_economy	daysonmarket	engine_cylinders	engine_displacement	franchise_dealer	franchise_make	front_legroom
1644003	WDCYC5FFXHX274483	41.90	SUV / Crossover	Houston	11.00	56	V8	4000.00	yes	Lamborghini	52.50
981110	SCA687S55JU104554	43.50	Sedan	Pompano Beach	12.00	27	V12	6800.00	no	NaN	41.00
30938	WDCYC3KF4HX266373	41.90	SUV / Crossover	Huntington	11.00	49	V8				
972072	SCA664S50JUX54414	42.30	Sedan	West Palm Beach	12.00	442	V12				
1084195	WDCYC3KF9HX270662	41.90	SUV / Crossover	Martin	11.00	11	V8				
1569848	SJAAC2ZV0JC020854	40.90	SUV / Crossover	Addison	12.00	51	W12				
117371	SCA665C53HUX86840	36.90	Coupe	Greenwich	12.00	24	V12				
1349220	SCA664L57HUX66472	49.00	Sedan	Chesterfield	12.00	21	V12				
1520852	WDCYC5FH1JX292474	41.90	SUV / Crossover	Austin	11.00	414	V8				
1630649	WBSJF0C03LCD05697	36.50	Sedan	The Woodlands	15.00	330	V8				

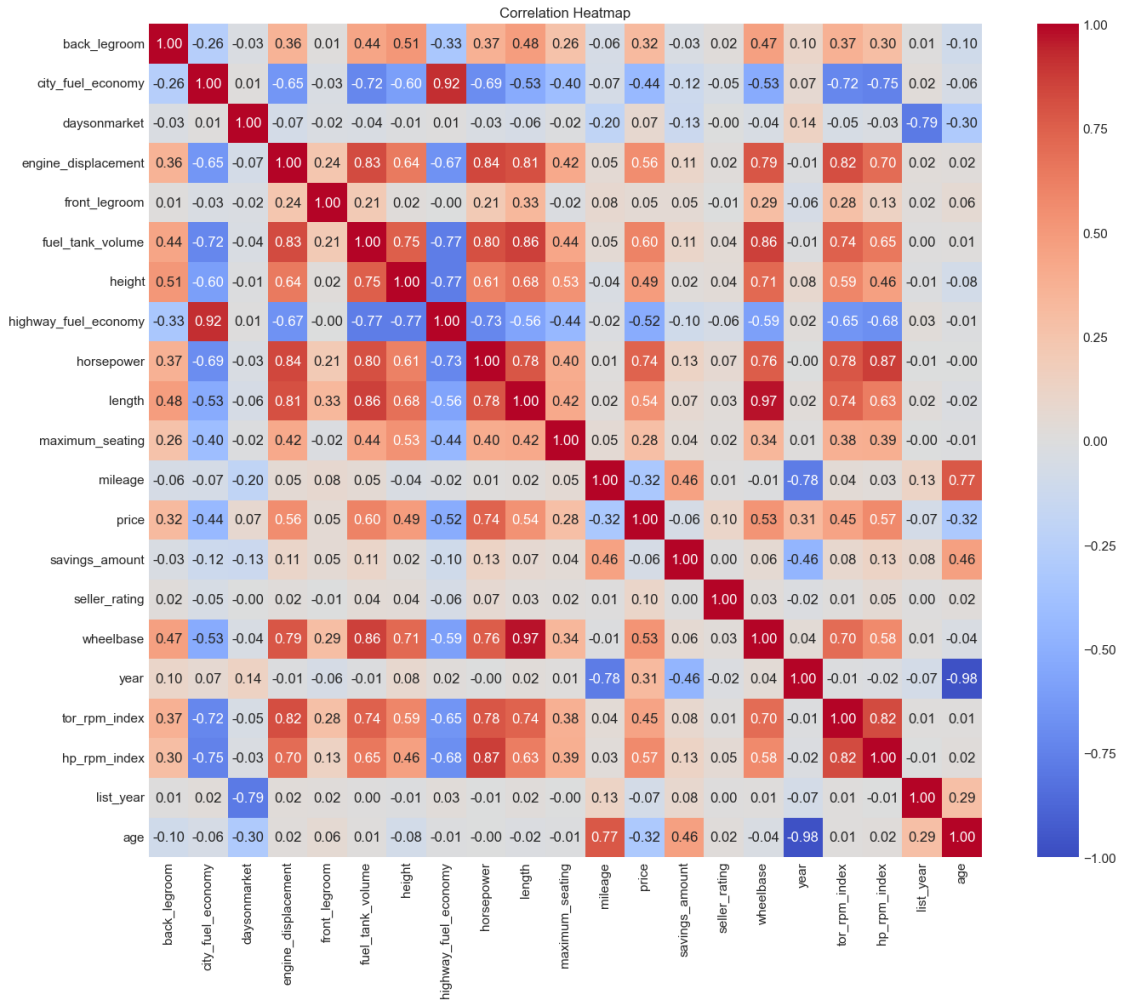
```
30938                4000.00              yes     Land Rover          52.50
972072               6600.00              yes    Rolls-Royce          41.70
1084195              4000.00               no            NaN          52.50
1569848              6000.00               no            NaN          41.70
117371               6600.00              yes    Rolls-Royce          41.50
1349220              6600.00              yes    Rolls-Royce          41.70
1520852              4000.00              yes  Mercedes-Benz          52.50
1630649              4400.00              yes            BMW          41.40

         fuel_tank_volume fuel_type  height  highway_fuel_economy  horsepower  \
1644003             25.40  Gasoline   88.00                 11.00      416.00
981110              23.80  Gasoline   64.80                 19.00      563.00
30938               25.40  Gasoline   88.00                 11.00      416.00
972072              21.80  Gasoline   61.00                 18.00      563.00
1084195             25.40  Gasoline   88.00                 11.00      416.00
1569848             22.50  Gasoline   68.60                 19.00      600.00
117371              21.90  Gasoline   59.30                 19.00      624.00
1349220             21.80  Gasoline   61.00                 19.00      563.00
1520852             25.40  Gasoline   88.00                 11.00      416.00
1630649             20.10  Gasoline   57.80                 21.00      617.00

         length listing_color     make_name  maximum_seating  mileage  \
1644003  177.10         BLACK  Mercedes-Benz             5.00 39222.00
981110   227.20         BLACK    Rolls-Royce             5.00  1154.00
30938    177.10       UNKNOWN  Mercedes-Benz             5.00 19057.00
972072   212.60         BLACK    Rolls-Royce             5.00  9353.00
1084195  177.10         WHITE  Mercedes-Benz             5.00 29851.00
1569848  202.40         WHITE        Bentley             5.00 11885.00
117371   208.50          GRAY    Rolls-Royce             4.00  5756.00
1349220  219.30         BLACK    Rolls-Royce             5.00  7115.00
1520852  177.20          BLUE  Mercedes-Benz             5.00  4892.00
1630649  195.50          BLUE            BMW             5.00  4495.00

        model_name     price  savings_amount  seller_rating transmission  \
1644003    G-Class 174888.00           74595           4.64            A
981110     Phantom 469950.00           65209           4.60            A
30938      G-Class  94897.00           58542           4.60            A
972072       Ghost 209900.00           50406           4.92            A
1084195    G-Class  86990.00           49610           4.50            A
1569848   Bentayga 187900.00           40304           4.60            A
117371      Wraith 269900.00           40293           4.52            A
1349220       Ghost 249900.00           35559           4.75            A
1520852    G-Class 209000.00           34503           4.71            A
1630649         M5 139635.00           33432           4.27            A

                    trim_name wheel_system  wheelbase  year  tor_rpm_index  \
1644003      G 550 4x4 Squared          AWD     112.20  2017         787500
```

```
981110                     RWD          RWD     139.80  2018        1128800
30938        G 550 4x4 Squared          AWD     112.20  2017         787500
972072                Series II          RWD     129.70  2018         907500
1084195      G 550 4x4 Squared          AWD     112.20  2017         787500
1569848       W12 Mulliner AWD          AWD     117.90  2018         896400
117371                   Coupe          RWD     122.50  2017         885000
1349220     Extended Wheelbase          RWD     136.40  2017         907500
1520852  G 550 4x4 Squared AWD          AWD     112.20  2018         787500
1630649        Competition AWD          AWD     117.40  2020         995400

         hp_rpm_index  list_year  age
1644003       2288000       2020    3
981110        2815000       2020    2
30938         2288000       2020    3
972072        2955750       2019    1
1084195       2288000       2020    3
1569848       3000000       2020    2
117371        3494400       2020    3
1349220       2955750       2020    3
1520852       2288000       2019    1
1630649       3702000       2019   -1
```

[66]:
```
df2.drop(df2['price'].nlargest(5).index, inplace=True)
```

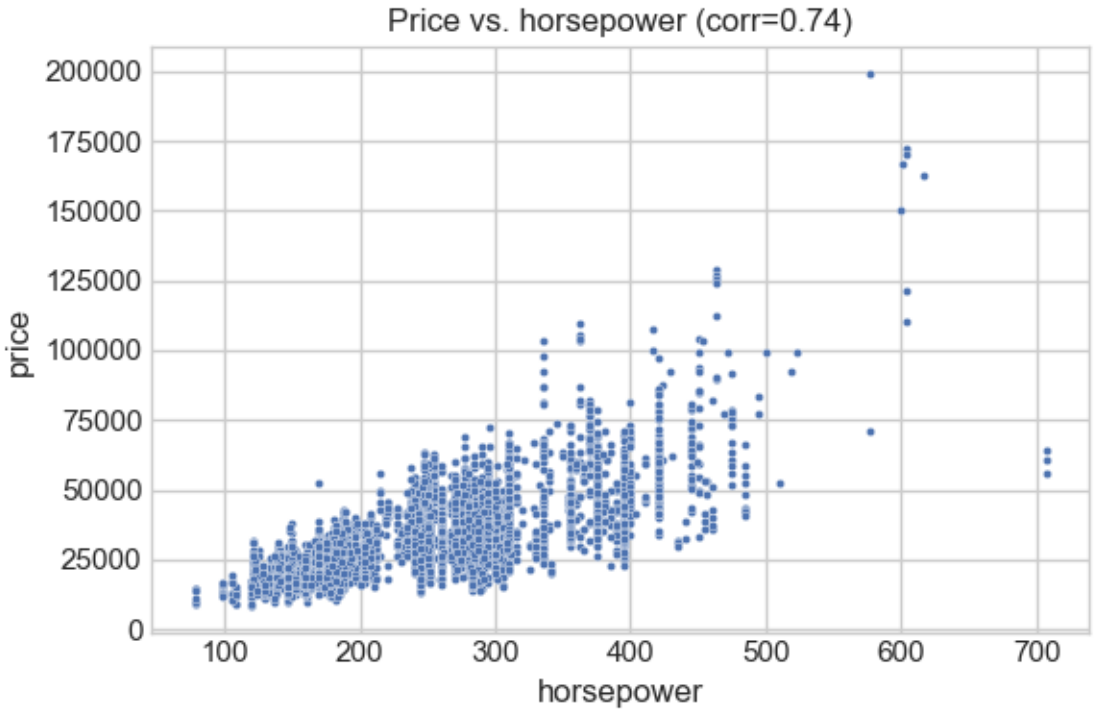
[67]:
```
# Correlation
cor=df2.select_dtypes(include='number').corr()
```

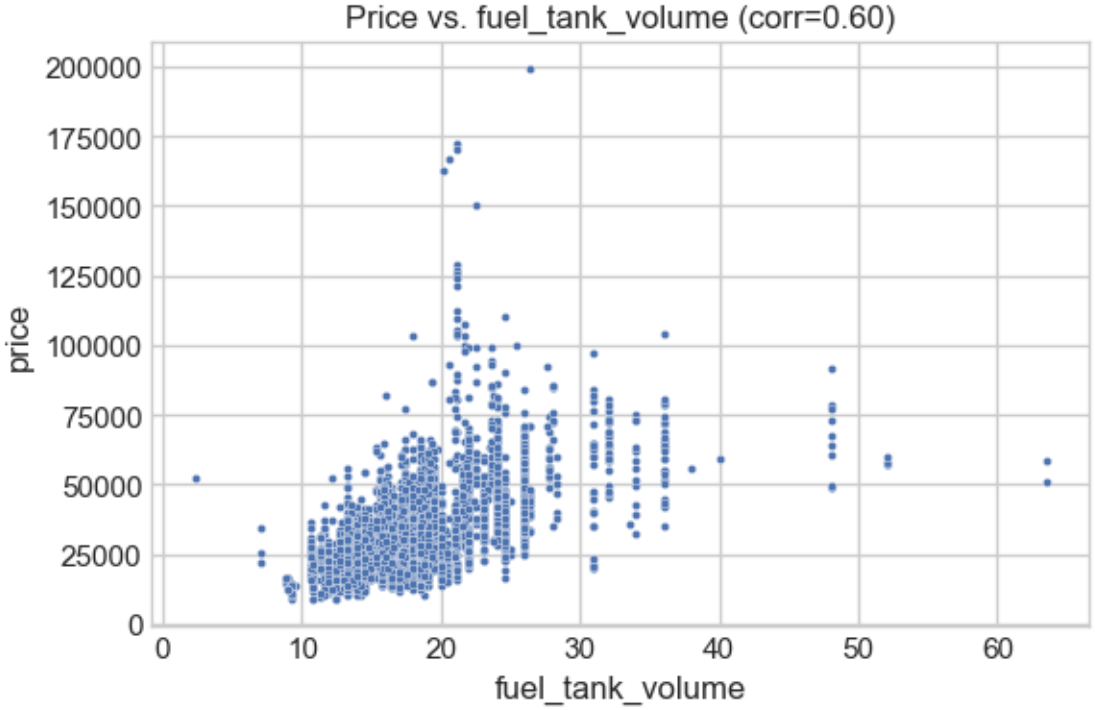
[68]:
```
# Correlation heatmap plot
plt.figure(figsize = (15,12))
sns.heatmap(cor, annot=True, fmt=".2f", cmap='coolwarm', vmin=-1, vmax=1,␣
 ↪square=True)
plt.title('Correlation Heatmap')

# Show the plot
plt.tight_layout()
plt.show()
```

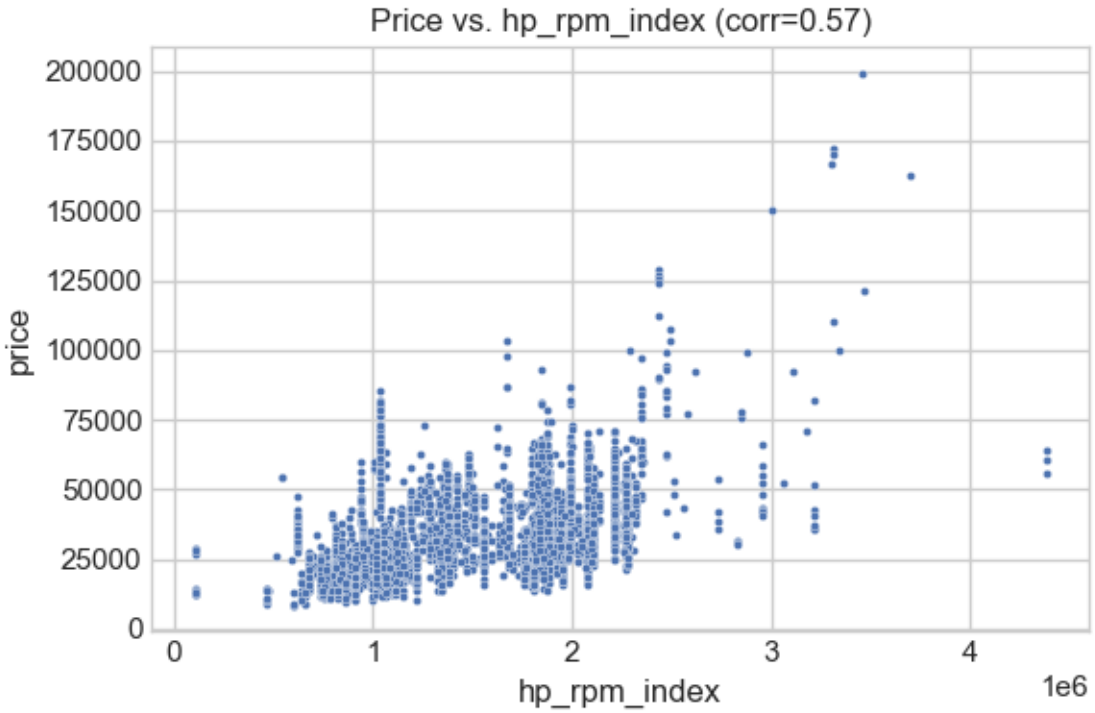


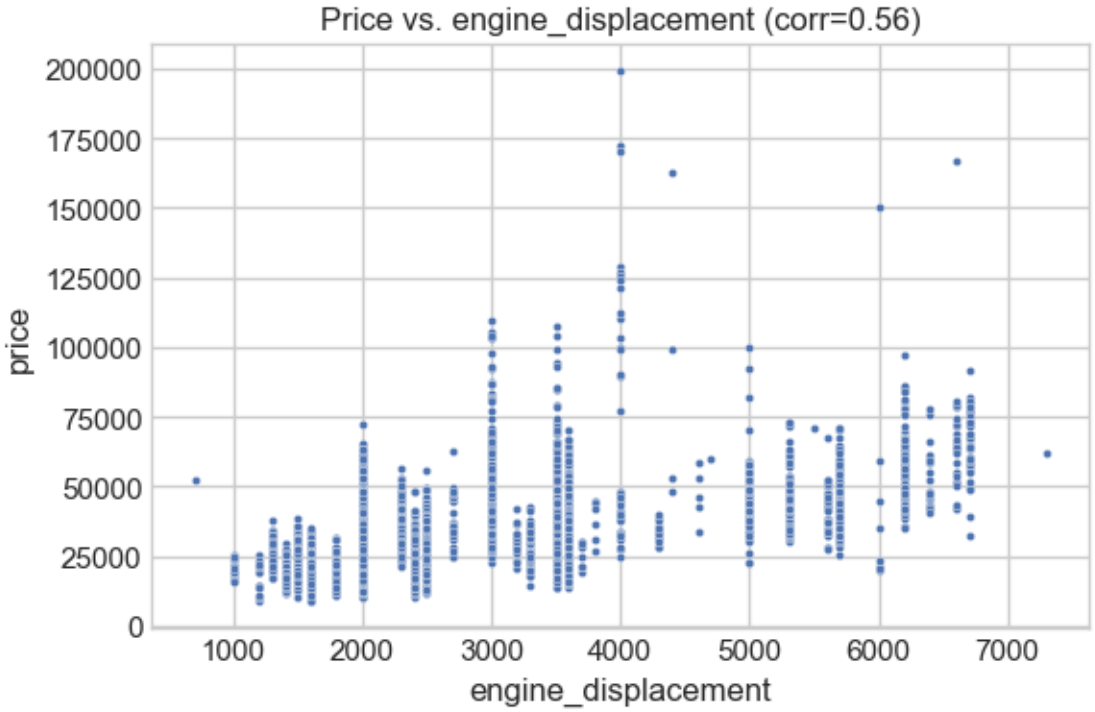
[69]:
```
# I will plot the correlation plot of top 10 categories using 5000 samples, otherwise it will take too long

# Compute correlations
corr = df2.corr(numeric_only=True)['price'].abs().sort_values(ascending=False)
top_features = corr.drop('price').head(10).index  # pick top 10 correlated features

# Subsample for speed
sample_df = df2.sample(n=5000, random_state=56)

# Plot
for col in top_features:
    plt.figure(figsize=(6, 4))
    sns.scatterplot(data=sample_df, x=col, y='price', s=10)
    plt.title(f'Price vs. {col} (corr={corr[col]:.2f})')
```

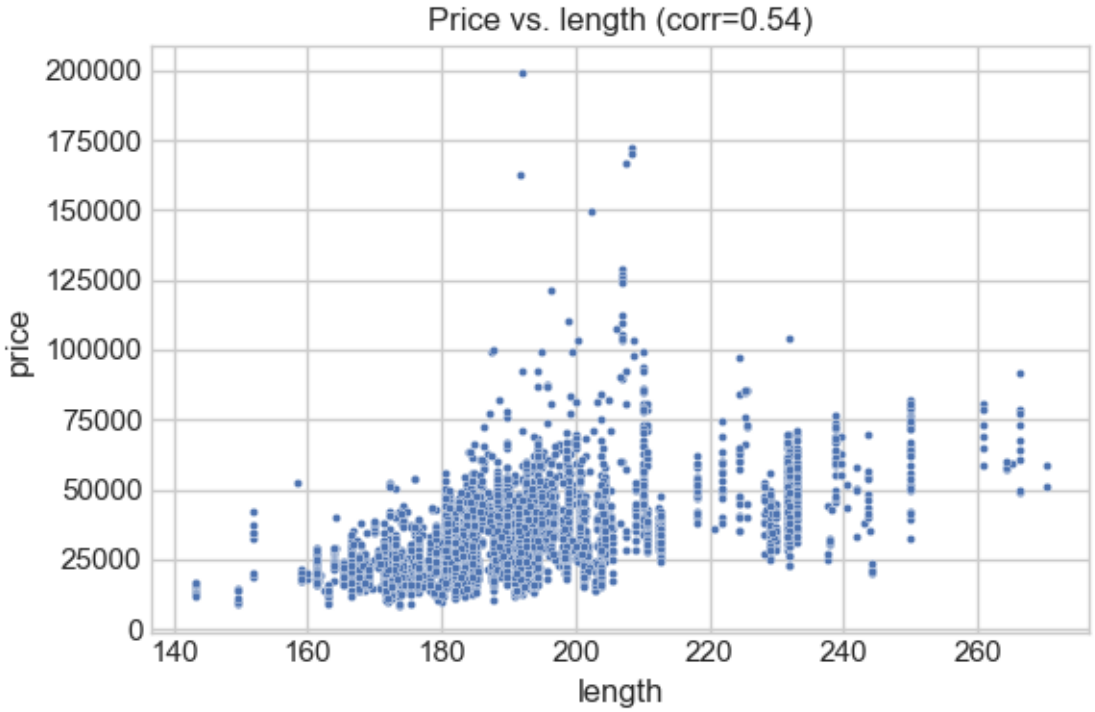
```
    plt.tight_layout()
    plt.show()
```



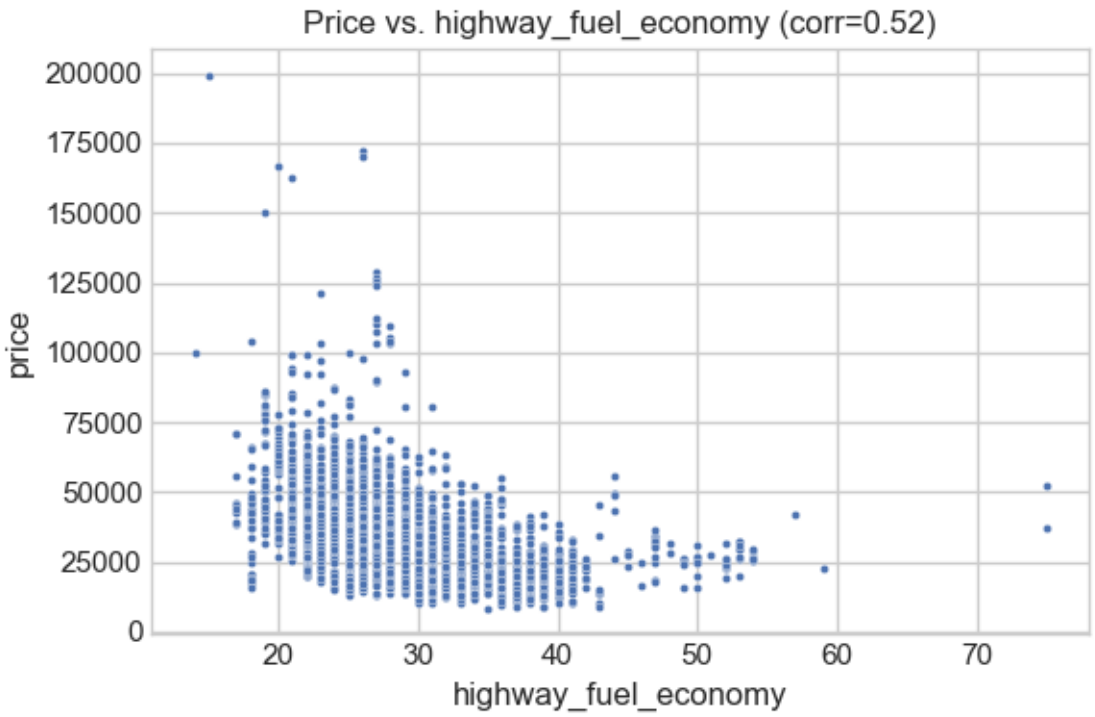


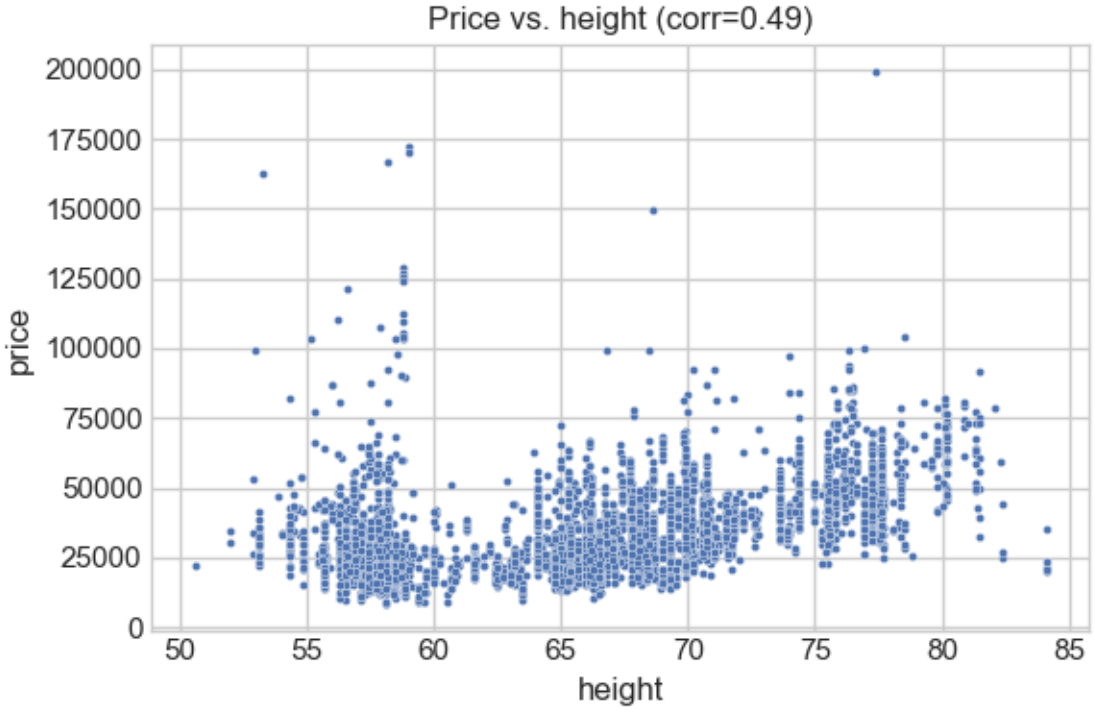

Price vs. hp_rpm_index (corr=0.57)
price
200000
175000
150000
125000
100000
75000
50000
25000
0
0
1
2
3
4
1e6
hp_rpm_index

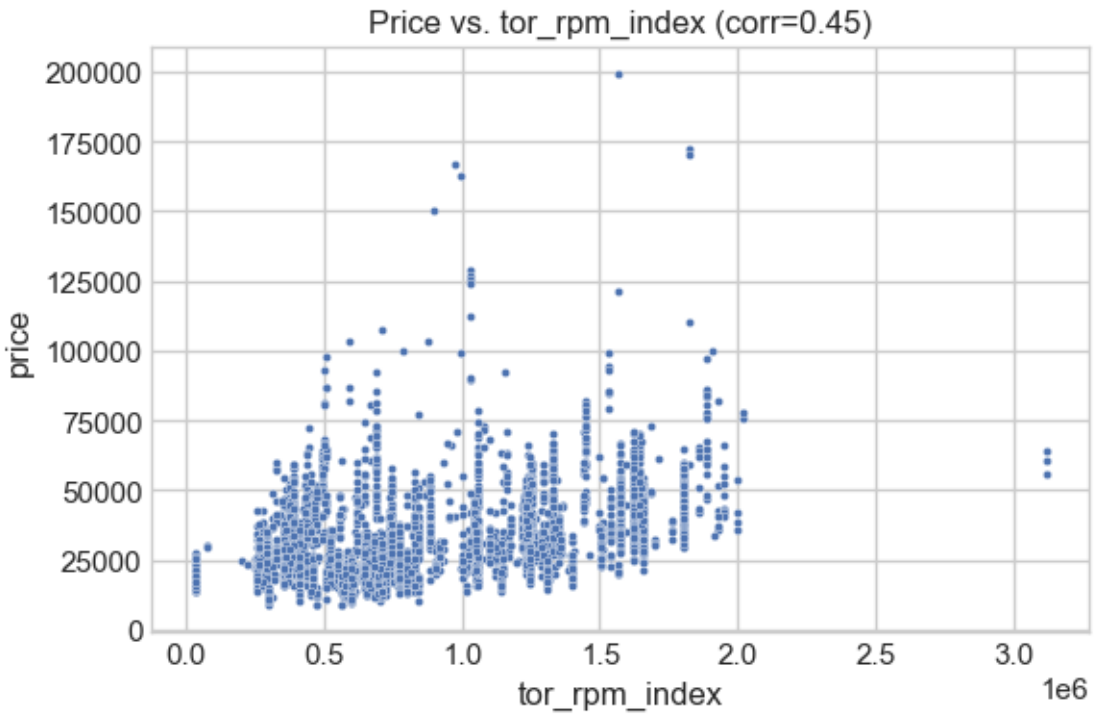

Price vs. engine_displacement (corr=0.56)
price
200000
175000
150000
125000
100000
75000
50000
25000
0
1000
2000
3000
4000
5000
6000
7000
engine_displacement

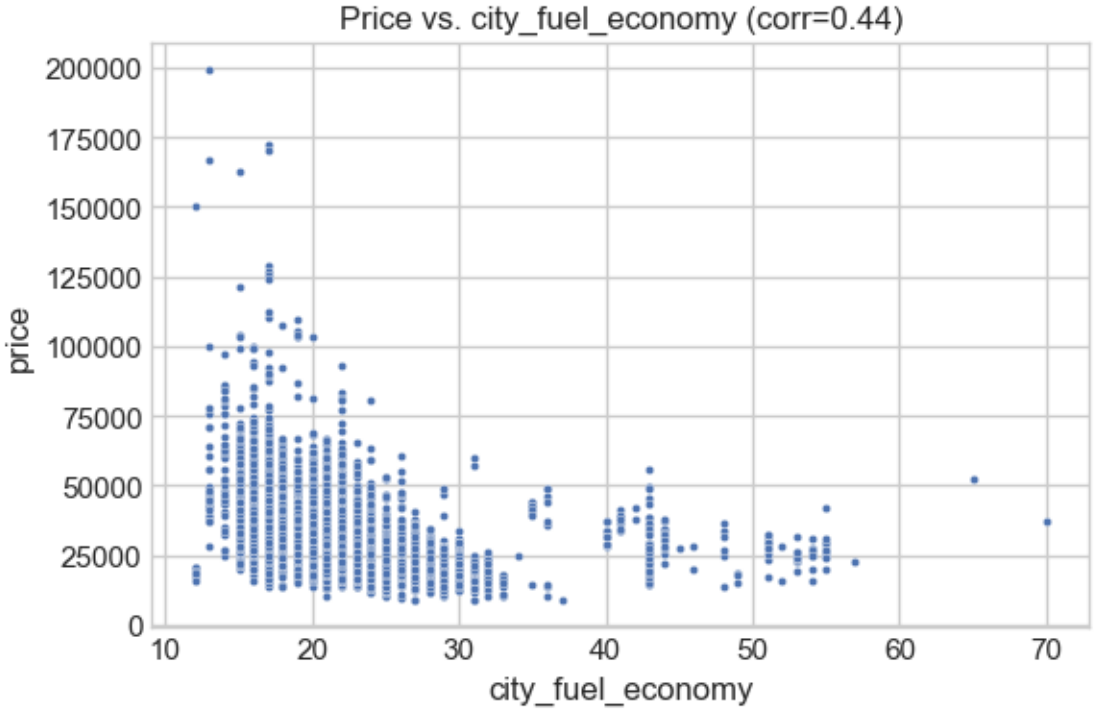

Price vs. length (corr=0.54)
price
length
200000
175000
150000
125000
100000
75000
50000
25000
0
140
160
180
200
220
240
260


Price vs. wheelbase (corr=0.53)
price
wheelbase
200000
175000
150000
125000
100000
75000
50000
25000
0
100
120
140
160
180


Price vs. highway_fuel_economy (corr=0.52)
price
highway_fuel_economy
200000
175000
150000
125000
100000
75000
50000
25000
0
20
30
40
50
60
70


Price vs. height (corr=0.49)
price
height
200000
175000
150000
125000
100000
75000
50000
25000
0
50
55
60
65
70
75
80
85


Price vs. tor_rpm_index (corr=0.45)
price
200000
175000
150000
125000
100000
75000
50000
25000
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
1e6
tor_rpm_index


Price vs. city_fuel_economy (corr=0.44)
price
200000
175000
150000
125000
100000
75000
50000
25000
0
10
20
30
40
50
60
70
city_fuel_economy

[70]:
```
df2.isnull().sum()/df2.shape[0]*100 #check for missing value.
```

[70]:
```
vin                     0.00
back_legroom            0.00
body_type               0.00
city                    0.00
city_fuel_economy       8.54
daysonmarket            0.00
engine_cylinders        0.00
engine_displacement     0.00
franchise_dealer        0.00
franchise_make         10.19
front_legroom           0.00
fuel_tank_volume        0.00
fuel_type               0.00
height                  0.00
highway_fuel_economy    8.54
horsepower              0.00
length                  0.00
listing_color           0.00
make_name               0.00
maximum_seating         0.00
mileage                 0.00
model_name              0.00
price                   0.00
savings_amount          0.00
seller_rating           0.00
transmission            0.00
trim_name               0.00
wheel_system            0.00
wheelbase               0.00
year                    0.00
tor_rpm_index          13.96
hp_rpm_index           12.73
list_year               0.00
age                     0.00
dtype: float64
```

[71]:
```
df2[['hp_rpm_index','tor_rpm_index','city_fuel_economy','highway_fuel_economy']].
↪info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1968093 entries, 0 to 1968105
Data columns (total 4 columns):
 #   Column                Dtype
---  ------                -----
 0   hp_rpm_index          Int64
 1   tor_rpm_index         Int64
```

```
 2   city_fuel_economy     float64
 3   highway_fuel_economy  float64
dtypes: Int64(2), float64(2)
memory usage: 78.8 MB
```

[72]:
```
df2[['tor_rpm_index', 'hp_rpm_index']]= df2[['tor_rpm_index', 'hp_rpm_index']].
 ↪astype('float')
```

[73]:
```
# Impute mising values.
#df2['city_fuel_economy'] = df2.groupby(['make_name',␣
 ↪'model_name'])['city_fuel_economy'].transform(lambda x: x.fillna(x.mean()))

#I tried to impute the column by adding mean value of the group(using make name␣
 ↪and model name, in addition year too... however
#this still leave many values NaN, which might be due to the feature of the car␣
 ↪types.

# Replace NaNs with -1 as a placeholder.
df2['city_fuel_economy'] = df2['city_fuel_economy'].fillna(-1)

# Flag the rows where this happened
df2['city_fuel_economy_missing'] = df2['city_fuel_economy'] == -1
```

[74]:
```
# Replace NaNs with -1 as a placeholder.
df2['highway_fuel_economy'] = df2['highway_fuel_economy'].fillna(-1)

# Flag the rows where this happened
df2['highway_fuel_economy_missing'] = df2['highway_fuel_economy'] == -1
```

[75]:
```
# Replace NaNs with -1 as a placeholder.
df2['hp_rpm_index'] = df2['hp_rpm_index'].fillna(-1)

# Flag the rows where this happened
df2['hp_rpm_index_missing'] = df2['hp_rpm_index'] == -1
```

[76]:
```
# Replace NaNs with -1 as a placeholder.
df2['tor_rpm_index'] = df2['tor_rpm_index'].fillna(-1)

# Flag the rows where this happened
df2['tor_rpm_index_missing'] = df2['tor_rpm_index'] == -1
```

[77]:
```
df2.sample(10)
```

[77]:

	vin	back_legroom	body_type	city	\
1623717	5GAERBKWXLJ293575	38.90	SUV / Crossover	Dallas	
1766596	1GKKNULSXLZ233287	39.70	SUV / Crossover	Boulder	
1582721	3PCAJ5M1XKF104775	38.70	SUV / Crossover	Grapevine	

```
882123   WDDZF4KB1HA194769         35.80            Sedan     Orlando
1405547  1HGCV1F10LA122815         40.40            Sedan     Wichita
963017   2T2BZMCA3HC102383         38.00  SUV / Crossover    Sarasota
1635469  JTDEPMAE9MJ125565         34.80            Sedan     Houston
185066   5XXGT4L38JG273948         35.60            Sedan  Queensbury
1542200  LRBFX2SA1LD168667         37.50  SUV / Crossover     Kingman
1332487  SALWZ2FE6HA126215         37.00  SUV / Crossover      Eureka

         city_fuel_economy  daysonmarket engine_cylinders  \
1623717              18.00            27               V6
1766596              18.00            27               V6
1582721              24.00           222               I4
882123               22.00            37               I4
1405547              30.00            21               I4
963017               19.00            30               V6
1635469              30.00            45               I4
185066               24.00            18               I4
1542200              21.00           110               I4
1332487              14.00            91               V8

         engine_displacement franchise_dealer franchise_make  front_legroom  \
1623717              3600.00              yes          Buick          41.20
1766596              3600.00              yes            GMC          41.00
1582721              2000.00              yes       INFINITI          39.60
882123               2000.00              yes  Mercedes-Benz          41.30
1405547              1500.00              yes          Honda          42.30
963017               3500.00              yes          Lexus          44.10
1635469              1800.00              yes         Toyota          42.00
185066               2400.00              yes          Honda          45.50
1542200              2500.00              yes          Buick          40.90
1332487              5000.00               no            NaN          42.20

         fuel_tank_volume fuel_type  height  highway_fuel_economy  horsepower  \
1623717             19.00  Gasoline   69.90                 26.00      310.00
1766596             22.00  Gasoline   66.70                 25.00      310.00
1582721             16.00  Gasoline   66.00                 31.00      268.00
882123              17.40  Gasoline   57.80                 29.00      241.00
1405547             14.80  Gasoline   57.10                 38.00      192.00
963017              19.20  Gasoline   67.70                 26.00      295.00
1635469             13.20  Gasoline   56.50                 38.00      139.00
185066              18.50  Gasoline   57.70                 34.00      185.00
1542200             17.30  Gasoline   66.80                 27.00      197.00
1332487             27.70  Gasoline   70.10                 19.00      550.00

         length listing_color      make_name  maximum_seating  mileage  \
1623717  204.30         WHITE          Buick             7.00     0.00
1766596  193.40       UNKNOWN            GMC             7.00     0.00
```

```
1582721  184.70         WHITE       INFINITI             5.00 24750.00
882123   193.80        SILVER  Mercedes-Benz             5.00 23669.00
1405547  192.20        SILVER          Honda             5.00     5.00
963017   192.50         WHITE          Lexus             5.00 19503.00
1635469  182.30           RED         Toyota             5.00    10.00
185066   191.10         BLACK            Kia             5.00 42818.00
1542200  183.70          BLUE          Buick             5.00     8.00
1332487  191.00          BLUE     Land Rover             5.00 72734.00

                model_name    price  savings_amount  seller_rating  \
1623717            Enclave 48485.00               0           4.65
1766596             Acadia 47605.00               0           4.71
1582721               QX50 37407.00            1089           4.45
882123             E-Class 38991.00            2252           3.95
1405547             Accord 25225.00               0           4.71
963017              RX 350 43990.00            2432           4.61
1635469            Corolla 19858.00               0           4.40
185066              Optima 15999.00             542           4.29
1542200           Envision 37735.00               0           4.70
1332487  Range Rover Sport 59996.00            1537           4.63

        transmission      trim_name wheel_system  wheelbase  year  \
1623717            A    Essence FWD          FWD     120.90  2020
1766596            A        SLT AWD          4WD     112.50  2020
1582721          CVT  Essential FWD          FWD     110.20  2019
882123             A   E 300 4MATIC          AWD     115.70  2017
1405547          CVT    1.5T LX FWD          FWD     111.40  2020
963017             A    F Sport AWD          AWD     109.80  2017
1635469          CVT         LE FWD          FWD     106.30  2021
185066             A             LX          FWD     110.40  2018
1542200            A    Essence AWD          AWD     108.30  2020
1332487            A     V8 SVR 4WD          AWD     115.10  2017

         tor_rpm_index  hp_rpm_index  list_year  age  \
1623717      744800.00    2108000.00       2020    0
1766596     1327900.00    2077000.00       2020    0
1582721      448000.00    1500800.00       2020    1
882123       354900.00    1337550.00       2020    3
1405547      288000.00    1056000.00       2020    0
963017      1254900.00    1858500.00       2020    3
1635469          -1.00         -1.00       2020   -1
185066       712000.00    1110000.00       2020    2
1542200          -1.00         -1.00       2020    0
1332487     1757000.00    3300000.00       2020    3

         city_fuel_economy_missing  highway_fuel_economy_missing  \
1623717                      False                         False
```

```
1766596                      False                      False
1582721                      False                      False
882123                       False                      False
1405547                      False                      False
963017                       False                      False
1635469                      False                      False
185066                       False                      False
1542200                      False                      False
1332487                      False                      False

         hp_rpm_index_missing  tor_rpm_index_missing
1623717                 False                 False
1766596                 False                 False
1582721                 False                 False
882123                  False                 False
1405547                 False                 False
963017                  False                 False
1635469                  True                  True
185066                  False                 False
1542200                  True                  True
1332487                 False                 False
```

[78]:
```
df2.isna().sum()
```

[78]:
```
vin                          0
back_legroom                 0
body_type                    0
city                         0
city_fuel_economy            0
daysonmarket                 0
engine_cylinders             0
engine_displacement          0
franchise_dealer             0
franchise_make          200617
front_legroom                0
fuel_tank_volume             0
fuel_type                    0
height                       0
highway_fuel_economy         0
horsepower                   0
length                       0
listing_color                0
make_name                    0
maximum_seating              0
mileage                      0
model_name                   0
price                        0
```

```
savings_amount                  0
seller_rating                   0
transmission                    0
trim_name                       0
wheel_system                    0
wheelbase                       0
year                            0
tor_rpm_index                   0
hp_rpm_index                    0
list_year                       0
age                             0
city_fuel_economy_missing       0
highway_fuel_economy_missing    0
hp_rpm_index_missing            0
tor_rpm_index_missing           0
dtype: int64
```

[79]:
```
# I will drop franchise make as this column is mostly covered by is franchise␣
↪column.
df3 = df2.drop('franchise_make', axis = 1, inplace = False)
```

[80]:
```
df3.isna().sum()
```

[80]:
```
vin                     0
back_legroom            0
body_type               0
city                    0
city_fuel_economy       0
daysonmarket            0
engine_cylinders        0
engine_displacement     0
franchise_dealer        0
front_legroom           0
fuel_tank_volume        0
fuel_type               0
height                  0
highway_fuel_economy    0
horsepower              0
length                  0
listing_color           0
make_name               0
maximum_seating         0
mileage                 0
model_name              0
price                   0
savings_amount          0
seller_rating           0
```

```
transmission                    0
trim_name                       0
wheel_system                    0
wheelbase                       0
year                            0
tor_rpm_index                   0
hp_rpm_index                    0
list_year                       0
age                             0
city_fuel_economy_missing       0
highway_fuel_economy_missing    0
hp_rpm_index_missing            0
tor_rpm_index_missing           0
dtype: int64
```

When working with large datasets, managing memory can be quite challenging. Consequently, I will implement safe downcasting for numeric data types.

[81]:
```
print(df3.dtypes)
print(df3.memory_usage(deep=True).sum() / 1024**2, "MB")
```

```
vin                             object
back_legroom                   float64
body_type                       object
city                            object
city_fuel_economy              float64
daysonmarket                     int64
engine_cylinders                object
engine_displacement            float64
franchise_dealer                object
front_legroom                  float64
fuel_tank_volume               float64
fuel_type                       object
height                         float64
highway_fuel_economy           float64
horsepower                     float64
length                         float64
listing_color                   object
make_name                       object
maximum_seating                float64
mileage                        float64
model_name                      object
price                          float64
savings_amount                   int64
seller_rating                  float64
transmission                    object
trim_name                       object
wheel_system                    object
```

```
wheelbase                        float64
year                               int64
tor_rpm_index                    float64
hp_rpm_index                     float64
list_year                          int32
age                                int64
city_fuel_economy_missing           bool
highway_fuel_economy_missing        bool
hp_rpm_index_missing                bool
tor_rpm_index_missing               bool
dtype: object
1593.9349222183228 MB
```

[82]:
```
# Downcast numeric safely
for col in df3.select_dtypes(include=['int64']).columns:
    df3[col] = pd.to_numeric(df3[col], downcast='integer')

for col in df3.select_dtypes(include=['float64']).columns:
    df3[col] = pd.to_numeric(df3[col], downcast='float')
```

[83]:
```
print(df3.dtypes)
print(df3.memory_usage(deep=True).sum() / 1024**2, "MB")
```

```
vin                               object
back_legroom                     float32
body_type                         object
city                              object
city_fuel_economy                float32
daysonmarket                       int64
engine_cylinders                  object
engine_displacement              float32
franchise_dealer                  object
front_legroom                    float32
fuel_tank_volume                 float32
fuel_type                         object
height                           float32
highway_fuel_economy             float32
horsepower                       float32
length                           float32
listing_color                     object
make_name                         object
maximum_seating                  float32
mileage                          float32
model_name                        object
price                            float64
savings_amount                     int64
seller_rating                    float32
transmission                      object
```

```
trim_name                          object
wheel_system                       object
wheelbase                         float32
year                                int64
tor_rpm_index                     float32
hp_rpm_index                      float32
list_year                           int32
age                                 int64
city_fuel_economy_missing            bool
highway_fuel_economy_missing         bool
hp_rpm_index_missing                 bool
tor_rpm_index_missing                bool
dtype: object
1481.3197374343872 MB
```

[84]:
```
df3['age'] = pd.to_numeric(df3['age'], downcast='integer')
print(df3.memory_usage(deep=True).sum() / 1024**2, "MB")
```

```
1468.1812992095947 MB
```

The memory drop in this case seems moderate.

[85]:
```
for i in df3.select_dtypes(include='object').columns:
    print(f"{i}: {df3[i].nunique()}")                    # check how many␣
 ↪categories each object has.
```

```
vin: 1968093
body_type: 9
city: 4049
engine_cylinders: 24
franchise_dealer: 2
fuel_type: 6
listing_color: 15
make_name: 38
model_name: 444
transmission: 4
trim_name: 2749
wheel_system: 5
```

It has come to my attention that several categorical variables exhibit high cardinality, which may pose challenges in machine learning applications. I will proceed to recategorize or encode these variables to mitigate the risk of overfitting and reduce noise in the data.

[86]:
```
# Count frequency of each city
city_counts = df3['city'].value_counts()

# Map city names to 'High', 'Medium', or 'Low' based on frequency
def categorize_city(city):
    count = city_counts[city]
```

```
    if count > 10000:
        return 'High'
    elif count >= 1000:
        return 'Medium'
    else:
        return 'Low'

# Create a new column with the categories
df3['city_level'] = df3['city'].map(categorize_city)
```

[87]:
```
df3['city_level'].value_counts()
```

[87]:
```
city_level
Medium    1142744
Low        735139
High        90210
Name: count, dtype: int64
```

[88]:
```
# I will use frequency encoding for columns: engine_cylinders,listing_color,␣
 ↪make_name, model_name, trim_name
# List of columns I want to frequency encode
cols_to_encode = ['engine_cylinders','listing_color','make_name','model_name',␣
 ↪'trim_name']

# Loop through each column and apply frequency encoding
for col in cols_to_encode:
    freq = df3[col].value_counts()
    df3[col + '_freq'] = df3[col].map(freq)
```

[89]:
```
df3.sample(5)
```

[89]:

	vin	back_legroom	body_type	city	\
71103	1FTER4FH6KLA04264	34.50	Pickup Truck	Marlborough	
1859906	YV4BR0DK7M1695665	38.00	SUV / Crossover	Mission Viejo	
1372795	5TDDZRBH6LS015540	41.00	SUV / Crossover	Merriam	
1071452	1N4BL4EV8KC235197	35.20	Sedan	Madison	
521680	3FA6P0HD2LR109728	38.30	Sedan	Grand Rapids	

	city_fuel_economy	daysonmarket	engine_cylinders	\
71103	20.00	582	I4	
1859906	-1.00	1	I4	
1372795	20.00	195	V6	
1071452	28.00	20	I4	
521680	23.00	279	I4	

	engine_displacement	franchise_dealer	front_legroom	\
71103	2300.00	yes	43.10	

```
1859906              2000.00              yes          41.50
1372795              3500.00              yes          40.40
1071452              2500.00              yes          43.80
521680               2500.00              yes          44.30

         fuel_tank_volume fuel_type  height  highway_fuel_economy  horsepower  \
71103               18.00  Gasoline   71.50                 24.00      270.00
1859906             18.50  Gasoline   65.30                 -1.00      400.00
1372795             17.90  Gasoline   68.10                 27.00      295.00
1071452             16.20  Gasoline   56.80                 39.00      188.00
521680              16.50  Gasoline   58.10                 34.00      175.00

         length listing_color make_name  maximum_seating  mileage  model_name  \
71103    210.80         WHITE      Ford             5.00     0.00      Ranger
1859906  184.60        SILVER     Volvo             5.00     0.00        XC60
1372795  194.90        SILVER    Toyota             8.00     1.00  Highlander
1071452  192.90        SILVER    Nissan             5.00 39183.00      Altima
521680   191.70         WHITE      Ford             5.00  4034.00      Fusion

           price  savings_amount  seller_rating transmission  \
71103   43800.00               0           3.82            A
1859906 61360.00               0           4.47            A
1372795 47587.00               0           4.23            A
1071452 19653.00            1079           4.52          CVT
521680  24395.00               0           4.64            A

                                                trim_name wheel_system  \
71103                                  Lariat SuperCrew 4WD          4WD
1859906  Hybrid Plug-in Recharge Inscription Expression…          AWD
1372795                                      Limited AWD           AWD
1071452                                       2.5 SL FWD           FWD
521680                                            SE FWD           FWD

         wheelbase  year  tor_rpm_index  hp_rpm_index  list_year  age  \
71103       126.80  2019          -1.00         -1.00       2019    0
1859906     112.80  2021     1038400.00    2400000.00       2020   -1
1372795     112.20  2020     1236100.00    1947000.00       2020    0
1071452     111.20  2019      648000.00    1128000.00       2020    1
521680      112.20  2020      799200.00    1086000.00       2019   -1

         city_fuel_economy_missing  highway_fuel_economy_missing  \
71103                        False                         False
1859906                       True                          True
1372795                      False                         False
1071452                      False                         False
521680                       False                         False
```

```
        hp_rpm_index_missing  tor_rpm_index_missing city_level  \
71103                   True                   True        Low
1859906                False                  False        Low
1372795                False                  False        Low
1071452                False                  False     Medium
521680                 False                  False     Medium

        engine_cylinders_freq  listing_color_freq  make_name_freq  \
71103                 1043039              420478          305985
1859906               1043039              254629           14857
1372795                477433              254629          160811
1071452               1043039              254629          155101
521680                1043039              420478          305985

        model_name_freq  trim_name_freq
71103             10762           11889
1859906            4617              53
1372795           19125           17448
1071452           28596            2544
521680            31955           72579
```

[90]:
```
# I will drop columns that are frequency encoded and vin and city
df4 = df3.drop(['vin','city',␣
 ↪'engine_cylinders','listing_color','make_name','model_name', 'trim_name'],␣
 ↪axis = 1, inplace = False)
```

[91]:
```
# Since the age values are -1 to 4, I would like to modify it as well and make␣
 ↪all the values positive.
df4['age'] = df4['age']+1
```

[92]:
```
df4['age'].min().astype(int)   #as type added.
```

[92]:
```
0
```

[93]:
```
# Now I will do one hot encoding of my categorical variables.
cat_cols = ['body_type', 'franchise_dealer','fuel_type',␣
 ↪'transmission','wheel_system','city_level']

# Apply one-hot encoding
df5 = pd.get_dummies(df4, columns=cat_cols, drop_first=True)
```

[94]:
```
df5.head()
```

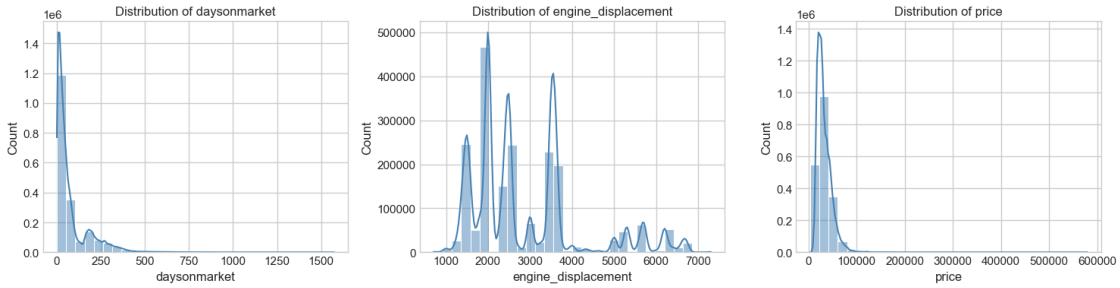
[94]:
```
   back_legroom  city_fuel_economy  daysonmarket  engine_displacement  \
0         35.10              -1.00           522              1300.00
1         38.10              -1.00           207              2000.00
2         37.60              -1.00           196              3000.00
```

```
3         38.10              -1.00           137              2000.00
4         37.10              -1.00           242              2000.00

   front_legroom  fuel_tank_volume  height  highway_fuel_economy  horsepower  \
0          41.20             12.70   66.50                 -1.00      177.00
1          39.10             17.70   68.00                 -1.00      246.00
2          39.00             23.50   73.00                 -1.00      340.00
3          39.10             17.70   68.00                 -1.00      246.00
4          40.20             16.60   66.30                 -1.00      247.00

   length  maximum_seating  mileage    price  savings_amount  seller_rating  \
0  166.60             5.00     7.00 23141.00               0           2.80
1  181.00             7.00     8.00 46500.00               0           3.00
2  195.10             7.00    11.00 67430.00               0           3.00
3  181.00             7.00     7.00 48880.00               0           3.00
4  188.90             5.00    12.00 66903.00               0           3.00

   wheelbase  year  tor_rpm_index  hp_rpm_index  list_year  age  \
0     101.20  2019      350000.00    1017750.00       2019    1
1     107.90  2020      376600.00    1353000.00       2020    1
2     115.00  2020     1162000.00    2210000.00       2020    1
3     107.90  2020      376600.00    1353000.00       2020    1
4     113.10  2020      322800.00    1358500.00       2020    1

   city_fuel_economy_missing  highway_fuel_economy_missing  \
0                       True                          True
1                       True                          True
2                       True                          True
3                       True                          True
4                       True                          True

   hp_rpm_index_missing  tor_rpm_index_missing  engine_cylinders_freq  \
0                 False                  False                1043039
1                 False                  False                1043039
2                 False                  False                 477433
3                 False                  False                1043039
4                 False                  False                1043039

   listing_color_freq  make_name_freq  model_name_freq  trim_name_freq  \
0                1799          109259            13281            8649
1              394415           10821             2842           12490
2              259009           10821             1123             297
3              394415           10821             2842           12490
4              275567           10821             1353             360

   body_type_Coupe  body_type_Hatchback  body_type_Minivan  \
0            False                False              False
```

```
1           False                False              False
2           False                False              False
3           False                False              False
4           False                False              False

   body_type_Pickup Truck  body_type_SUV / Crossover  body_type_Sedan  \
0                   False                       True            False
1                   False                       True            False
2                   False                       True            False
3                   False                       True            False
4                   False                       True            False

   body_type_Van  body_type_Wagon  franchise_dealer_yes  \
0          False            False                  True
1          False            False                  True
2          False            False                  True
3          False            False                  True
4          False            False                  True

   fuel_type_Compressed Natural Gas  fuel_type_Diesel  \
0                             False             False
1                             False             False
2                             False             False
3                             False             False
4                             False             False

   fuel_type_Flex Fuel Vehicle  fuel_type_Gasoline  fuel_type_Hybrid  \
0                        False               True             False
1                        False               True             False
2                        False               True             False
3                        False               True             False
4                        False               True             False

   transmission_CVT  transmission_Dual Clutch  transmission_M  \
0             False                     False           False
1             False                     False           False
2             False                     False           False
3             False                     False           False
4             False                     False           False

   wheel_system_4X2  wheel_system_AWD  wheel_system_FWD  wheel_system_RWD  \
0             False             False              True             False
1             False              True             False             False
2             False              True             False             False
3             False              True             False             False
4             False              True             False             False
```

```
   city_level_Low  city_level_Medium
0            True              False
1            True              False
2            True              False
3            True              False
4            True              False
```

[95]:
```
# These columns look right-skewed.
# Features to plot
features = ['daysonmarket', 'engine_displacement', 'price']

# Create subplots
fig, axes = plt.subplots(1, 3, figsize=(15, 4))  # 1 row, 3 columns

# Loop through each feature and plot
for i, col in enumerate(features):
    sns.histplot(data=df5, x=col, kde=True, bins=30, ax=axes[i],␣
 ↪color='steelblue')
    axes[i].set_title(f'Distribution of {col}')
    axes[i].set_xlabel(col)
    axes[i].set_ylabel('Count')

plt.tight_layout()
plt.show()
```



I have chosen to apply log transformation to these certain heavily right-skewed variables, particularly the price, which serves as my target variable. While tree-based models such as Random Forest, XGBoost, and Gradient Boosting may not require transformation, models like linear regression or gradient boosting tend to yield improved performance with this adjustment.

[96]:
```
# Log transform certain variables
cols_to_log = ['daysonmarket', 'engine_displacement', 'price']  # replace with␣
 ↪your actual column names

# Apply log1p (which is log(1 + x)) for safety
df5[cols_to_log] = df5[cols_to_log].apply(np.log1p)
```

In the analysis, I will particularly concentrate on price, which is also my target variable. A limited number of vehicles are identified as significantly expensive. These vehicles will be retained in the dataset as they represent valid values.

[97]:
```
# Define the log-transformed features
features = ['daysonmarket', 'engine_displacement', 'price']

# Create subplots
fig, axes = plt.subplots(1, 3, figsize=(15, 4))

# Loop to create histograms with KDE
for i, col in enumerate(features):
    sns.histplot(data=df5, x=col, kde=True, bins=30, ax=axes[i],␣
 ↪color='lightcoral')
    axes[i].set_title(f'Log-Transformed {col.capitalize()}', fontsize=12)
    axes[i].set_xlabel(f'log({col})')
    axes[i].set_ylabel('Count')

plt.tight_layout()
plt.show()
```

The transformed columns look better than the non transformed one.

[98]:
```
# I will rename the log-transformed columns to indicate that they have␣
 ↪undergone transformation.
df5.rename(columns={
    'daysonmarket': 'daysonmarket_log',
    'engine_displacement': 'engine_displacement_log',
    'price': 'price_log'
}, inplace=True)
```

[99]:
```
df5.head() # Check df one more time.
```

[99]:

	back_legroom	city_fuel_economy	daysonmarket_log	engine_displacement_log	\
0	35.10	-1.00	6.26	7.17	
1	38.10	-1.00	5.34	7.60	
2	37.60	-1.00	5.28	8.01	

```
3         38.10              -1.00              4.93                    7.60
4         37.10              -1.00              5.49                    7.60

   front_legroom  fuel_tank_volume  height  highway_fuel_economy  horsepower  \
0          41.20             12.70   66.50                 -1.00      177.00
1          39.10             17.70   68.00                 -1.00      246.00
2          39.00             23.50   73.00                 -1.00      340.00
3          39.10             17.70   68.00                 -1.00      246.00
4          40.20             16.60   66.30                 -1.00      247.00

   length  maximum_seating  mileage  price_log  savings_amount  seller_rating  \
0  166.60             5.00     7.00      10.05               0           2.80
1  181.00             7.00     8.00      10.75               0           3.00
2  195.10             7.00    11.00      11.12               0           3.00
3  181.00             7.00     7.00      10.80               0           3.00
4  188.90             5.00    12.00      11.11               0           3.00

   wheelbase  year  tor_rpm_index  hp_rpm_index  list_year  age  \
0     101.20  2019      350000.00    1017750.00       2019    1
1     107.90  2020      376600.00    1353000.00       2020    1
2     115.00  2020     1162000.00    2210000.00       2020    1
3     107.90  2020      376600.00    1353000.00       2020    1
4     113.10  2020      322800.00    1358500.00       2020    1

   city_fuel_economy_missing  highway_fuel_economy_missing  \
0                       True                          True
1                       True                          True
2                       True                          True
3                       True                          True
4                       True                          True

   hp_rpm_index_missing  tor_rpm_index_missing  engine_cylinders_freq  \
0                 False                  False                1043039
1                 False                  False                1043039
2                 False                  False                 477433
3                 False                  False                1043039
4                 False                  False                1043039

   listing_color_freq  make_name_freq  model_name_freq  trim_name_freq  \
0                1799          109259            13281            8649
1              394415           10821             2842           12490
2              259009           10821             1123             297
3              394415           10821             2842           12490
4              275567           10821             1353             360

   body_type_Coupe  body_type_Hatchback  body_type_Minivan  \
0            False                False              False
```

```
1           False                False              False
2           False                False              False
3           False                False              False
4           False                False              False

   body_type_Pickup Truck  body_type_SUV / Crossover  body_type_Sedan  \
0                   False                       True            False
1                   False                       True            False
2                   False                       True            False
3                   False                       True            False
4                   False                       True            False

   body_type_Van  body_type_Wagon  franchise_dealer_yes  \
0         False            False                  True
1         False            False                  True
2         False            False                  True
3         False            False                  True
4         False            False                  True

   fuel_type_Compressed Natural Gas  fuel_type_Diesel  \
0                             False             False
1                             False             False
2                             False             False
3                             False             False
4                             False             False

   fuel_type_Flex Fuel Vehicle  fuel_type_Gasoline  fuel_type_Hybrid  \
0                        False               True             False
1                        False               True             False
2                        False               True             False
3                        False               True             False
4                        False               True             False

   transmission_CVT  transmission_Dual Clutch  transmission_M  \
0             False                     False           False
1             False                     False           False
2             False                     False           False
3             False                     False           False
4             False                     False           False

   wheel_system_4X2  wheel_system_AWD  wheel_system_FWD  wheel_system_RWD  \
0             False             False              True             False
1             False              True             False             False
2             False              True             False             False
3             False              True             False             False
4             False              True             False             False
```

```
   city_level_Low  city_level_Medium
0            True              False
1            True              False
2            True              False
3            True              False
4            True              False
```

5 Creating Train and Test Datasets

[100]:
```
from sklearn.model_selection import train_test_split

# X = all features (predictors), y = target variable (price_log)
X = df5.drop('price_log', axis=1)
y = df5['price_log']

# Split into training and testing datasets
X_train, X_test, y_train_log, y_test_log = train_test_split( # target is log␣
 ↪transformed to make it easier to remember
    X, y,
    test_size=0.2,      # 20% for testing
    random_state=96     # for reproducibility
)
```

Our datasets is ready to be used for ML model.

6 Machine Learning Model Implementation

I am utilizing various machine learning (ML) methods on my analysis and applying the same evaluation metrics across all methods to assess their performance. I have developed code to evaluate the models and visualize different aspects of their performance. This approach will exhibit a comprehensive comparison of all ML methods.

[101]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error,␣
 ↪root_mean_squared_error, r2_score


def evaluate_model(model, X_train, X_test, y_train_log, y_test_log, features):
    """
    Evaluates a regression model and plots performance and feature importance.
    """

    # Inverse log transformation
    y_test = np.expm1(y_test_log)
    y_pred = np.expm1(model.predict(X_test))
    y_train = np.expm1(y_train_log)
    y_train_pred = np.expm1(model.predict(X_train))
```

```
    # Evaluation metrics
    mae = mean_absolute_error(y_test, y_pred)
    mse = mean_squared_error(y_test, y_pred)
    rmse = root_mean_squared_error(y_test, y_pred)
    r2_train = r2_score(y_train, y_train_pred)
    r2_test = r2_score(y_test, y_pred)
    r2_test_adj = 1 - (1 - r2_test) * ((len(X_test) - 1) / (len(X_test) -␣
↪X_test.shape[1] - 1))

    # Print evaluation
    print(f"Model: {model.__class__.__name__}")
    print("MAE:", mae)
    print("MSE:", mse)
    print("RMSE:", rmse)
    print("R\u00b2.train:", r2_train)
    print("R\u00b2:", r2_test)
    print("R\u00b2.adj:", r2_test_adj)

    # Plot actual and predicted values
    plt.figure(figsize=(12, 4))
    plt.plot(y_test[:100].values, label="Actual", linestyle='-', marker='o',␣
↪markersize=3)
    plt.plot(y_pred[:100], label="Predicted", linestyle='--', marker='x',␣
↪markersize=3)
    plt.title("Actual vs. Predicted Car Prices (First 100 Samples)",␣
↪fontsize=14)
    plt.xlabel("Sample Index")
    plt.ylabel("Price")
    plt.legend()
    plt.grid(True, linestyle='--', alpha=0.5)
    plt.tight_layout()
    plt.show()

    # Plot scatter plot (ideal: a diagonal line)
    plt.figure(figsize=(6, 6))
    plt.scatter(y_test, y_pred, alpha=0.5)
    plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')␣
↪ # Ideal line
    plt.xlabel("Actual Price")
    plt.ylabel("Predicted Price")
    plt.title("Predicted vs. Actual Car Prices")
    plt.grid(True)
    plt.tight_layout()
    plt.show()

    # Plot residual plots
```

```
    # Calculate residuals
    residuals = y_test - y_pred

    # Create the subplots side by side
    fig, axes = plt.subplots(1, 2, figsize=(15, 5))
    # Plot 1: Residuals vs Predicted
    sns.scatterplot(x=y_pred, y=residuals, ax=axes[0])
    axes[0].axhline(0, color='blue', linestyle='--')
    axes[0].set_xlabel("Predicted Values")
    axes[0].set_ylabel("Residuals")
    axes[0].set_title("Residuals vs Predicted")

    # Plot 2: Distribution of Residuals
    sns.histplot(residuals, bins=50, kde=True, ax=axes[1])
    axes[1].set_title("Distribution of Residuals")
    axes[1].set_xlabel("Residuals")

    # Adjust layout
    plt.tight_layout()
    plt.show()

    # Feature importance
    try:
        importance = model.feature_importances_
    except AttributeError:
        importance = model.coef_

    importance = np.abs(importance)
    if importance.ndim > 1:
        importance = importance[0]

    if len(importance) == len(features):
        feat = pd.Series(importance, index=features)
        plt.figure(figsize=(7, 4))
        plt.title(f'Feature Importances (Top 10) for {model.__class__.
↪__name__}', fontsize=12)
        plt.xlabel('Relative Importance')
        feat.nlargest(10).plot(kind='barh', color='steelblue')
        plt.gca().invert_yaxis()  # So the most important is at the top
        plt.tight_layout()
        plt.show()
    else:
        print("Warning: Number of features does not match importance length.␣
↪Skipping plot.")

    return [mae, mse, rmse, r2_train, r2_test, r2_test_adj]
```

6.1 Linear Regression

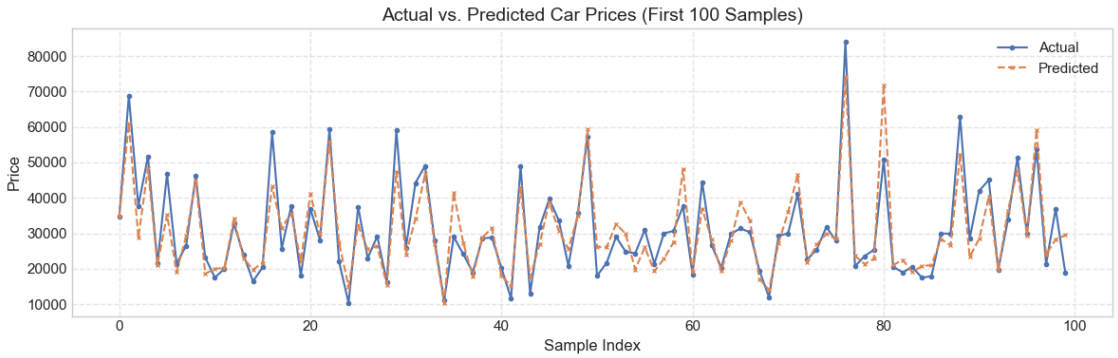
The first model that we implement will be a linear regression model. Scikit-learn's LinearRegression() employs Ordinary Least Squares (OLS), which does not necessitate scaling for operation; however, scaling can enhance the interpretability and stability of the model. For this data, scaling presented challenges for generating feature importance charts and didn't seem to improve model performance (I checked its performance with scaling). Therefore, I opted to forgo scaling for the linear regression analysis.

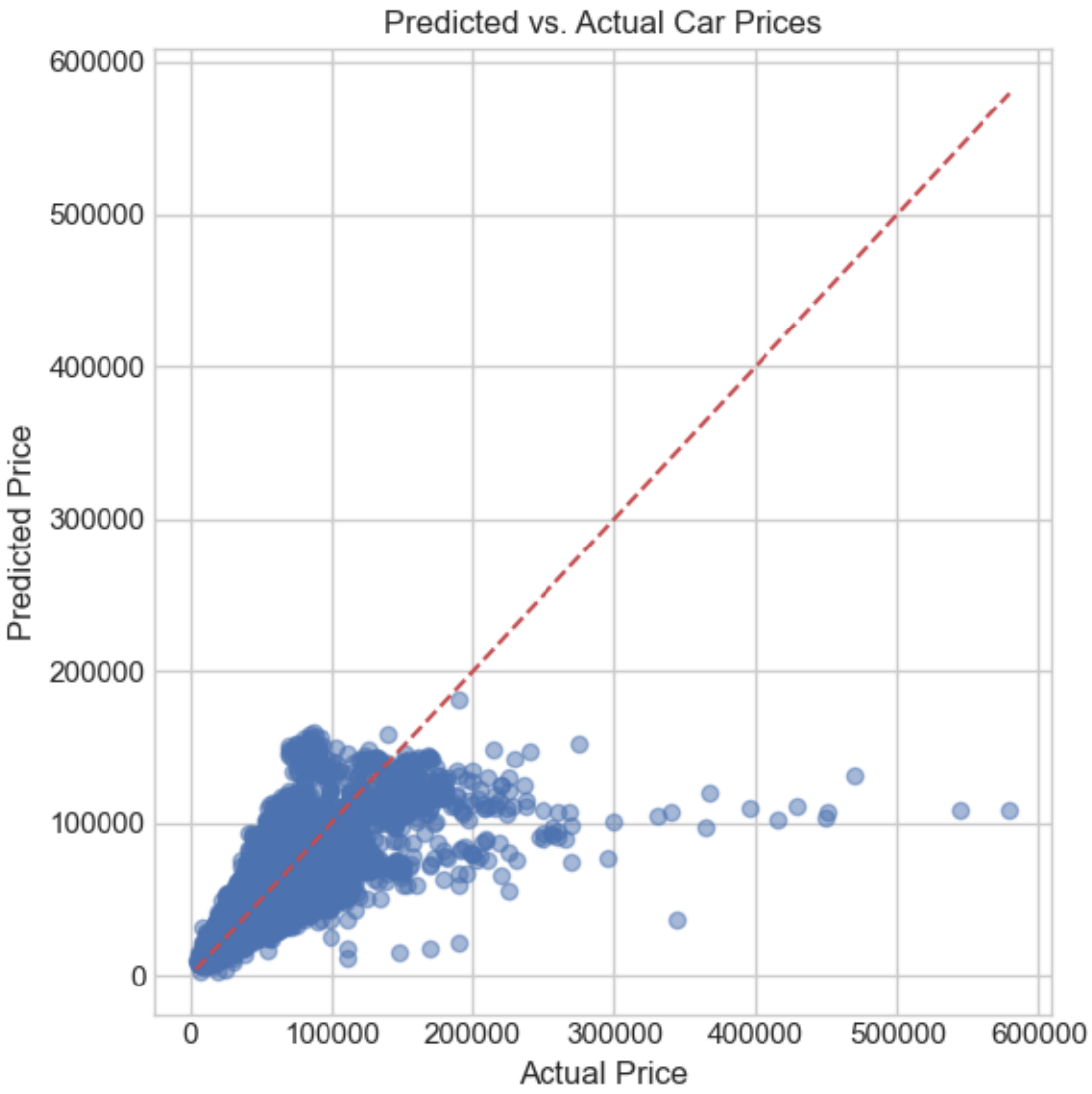
[102]:
```
# Initialize and fit the linear regression model
from sklearn.linear_model import LinearRegression

modLR = LinearRegression()

modLR.fit(X_train, y_train_log)
```

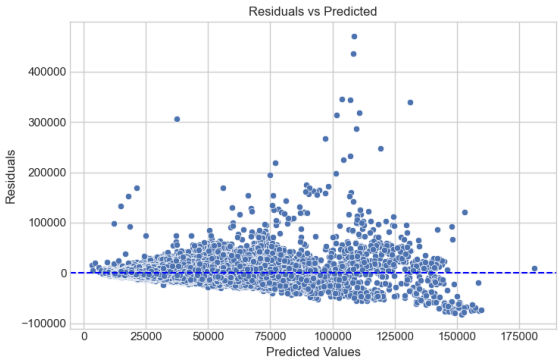
[102]: LinearRegression()

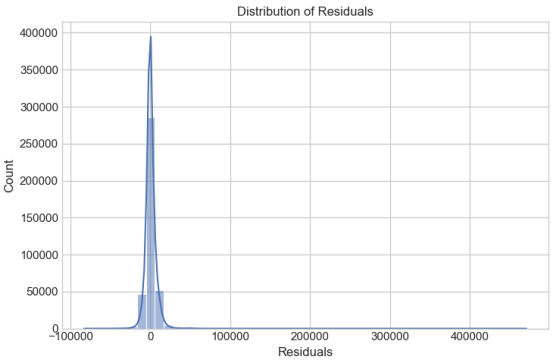
[103]:
```
# Evaluate Model
features = X_train.columns
evaluate_model(modLR, X_train, X_test, y_train_log, y_test_log, features)
```

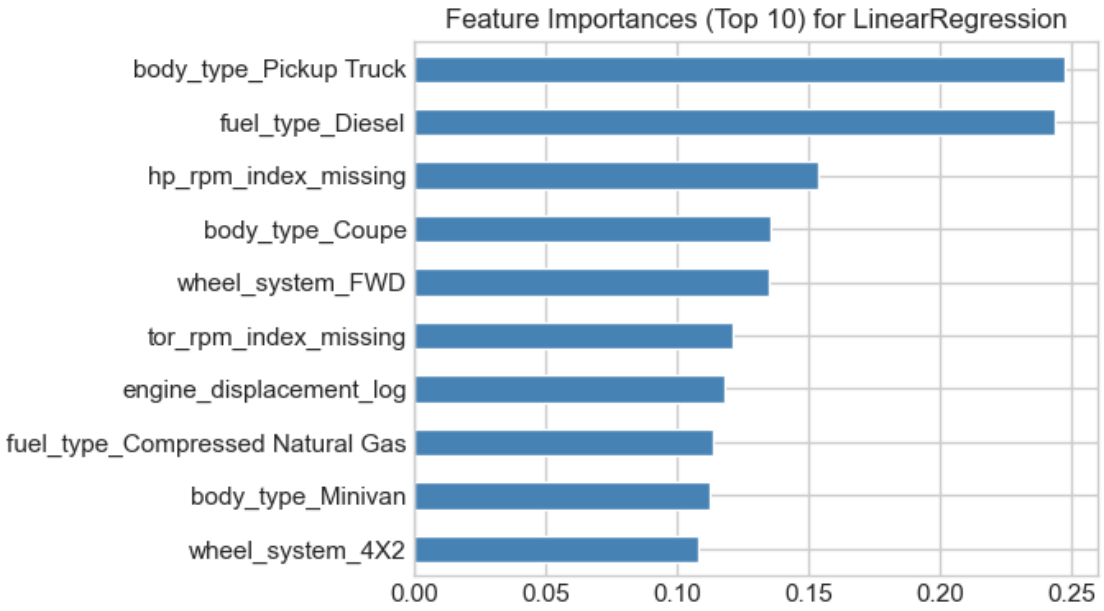
```
Model: LinearRegression
MAE: 4453.579376362097
MSE: 49467985.91861194
RMSE: 7033.348130059534
R².train: 0.7930394179109975
R²: 0.7871839761212494
R².adj: 0.7871558577542113
```




Predicted vs. Actual Car Prices
Predicted Price
Actual Price
600000
500000
400000
300000
200000
100000
0
0
100000
200000
300000
400000
500000
600000


Residuals vs Predicted
Residuals
Predicted Values
400000
300000
200000
100000
0
−100000
0
25000
50000
75000
100000
125000
150000
175000


Distribution of Residuals
Count
Residuals
400000
350000
300000
250000
200000
150000
100000
50000
0
−100000
0
100000
200000
300000
400000



[103]:
```
[4453.579376362097,
 49467985.91861194,
 7033.348130059534,
 0.7930394179109975,
 0.7871839761212494,
 0.7871558577542113]
```

The first plot is fairly straightforward. It shows the actual value (blue) of the car vs predicted value (orange) of the cars for the first 100 cars.

The Mean Absolute Error (MAE) is a simple metric with less sensitivity to outliers. The model deviates, on average, around \$4,454 from the true values. The Root Mean Square Error (RMSE) indicates a typical prediction error of about \$7,033 as it heavily penalizes larger errors. The model explains about 79.3% of the variation in the training data. This, generally, is a good fit! In the test (or validation) dataset, it explains around \$78.7% of the variance, which is closely aligned with the training R^2 (79.3%), suggesting little overfitting and successful generalization. The adjusted R^2 for my model is around 0.7871 which is nearly identical to the unadjusted R^2. This suggests that the vast majority of my predictors are significant indicating that most predictors are significant and not merely contributing noise.

The close clustering of points along the red diagonal line in the low-price range (less than \$150,000) in the Predicted vs Actual Car Price plot suggests the prediction of the model is very accurate and well-correlated with the actual values for car prices in this range. An R^2 value of approximately 0.79 suggests a satisfactory level of explanatory power regarding the variance in the data. However, regarding actual prices above \$150,000, the model consistently underpredicts values, implying systematic underprediction for higher end vehicles. This discrepancy may be attributed to the existence of fewer high-value cars within the dataset. To improve model performance, we could consider stratified sampling or upsampling methods for these underrepresented expensive automobiles.

The residual versus predicted plot indicates the presence of heteroscedasticity as it is funnel shaped (increasing spread), signifying that the variance of errors increases with higher predicted values. This condition violates a fundamental assumption of linear regression. The observation of more positive residuals at lower predicted values and more negative residuals at higher predicted values suggests potential underfitting or indicating that the model may not adequately capture certain nonlinear patterns.

The distribution of the residual plot shows a large peak at zero, which is good because it indicates most of the predictions are close to the actual values. Although there is a long positive skew to the right, suggesting the existence of extreme outliers in the data. Again, this could be due to the presence of a subset of cars that are significantly expensive compared to other cars in the dataset. Interestingly, these expensive cars are still a valid observation. One potential approach that can be used to to address this issue is by the the implementation of nonlinear models, such as random forest or gradient boosting, which are better suited to capture complex patterns in the data.

I have inverse-transformed the predicted log prices, resulting in the predicted prices being in the original price scale, which is great for the performance metrics and model output. However, it is essential to recognize that the model was trained on log(price) or np.log1p. and the coefficients and feature importances from the Linear Regression model pertain to log(price) rather than actual price. Therefore, the feature importance plot reflects the impact of features on log(price), rather than actual dollar values. For example, the classification of a vehicle as a pickup truck significantly increases the log(price) compared to the baseline body type. This indicates that pickups are generally more expensive than the baseline vehicle types, such as sedans or compacts. Diesel vehicles also exhibit higher prices relative to the baseline fuel type, likely gasoline. This price differential may be attributed to factors such as increased torque, enhanced fuel efficiency, or greater commercial value. The absence of horsepower RPM data in feature importance chart is noteworthy, potentially indicating older or lower-spec vehicles with ambiguous specifications. Coupes also contribute to an increase in the log(price), albeit to a lesser extent than pickups, likely due to their sportier design or performance characteristics. For instance, in the case of diesel vehicles, a diesel vehicle is projected to be 27% more expensive (calculated as $\exp(0.24) = 1.27$ or using $\text{np.expm1}(0.24) = 0.271$) than a comparable non-diesel vehicle, assuming all other factors remain constant.

6.2 Ridge Regression

6.2.1 Feature Scaling

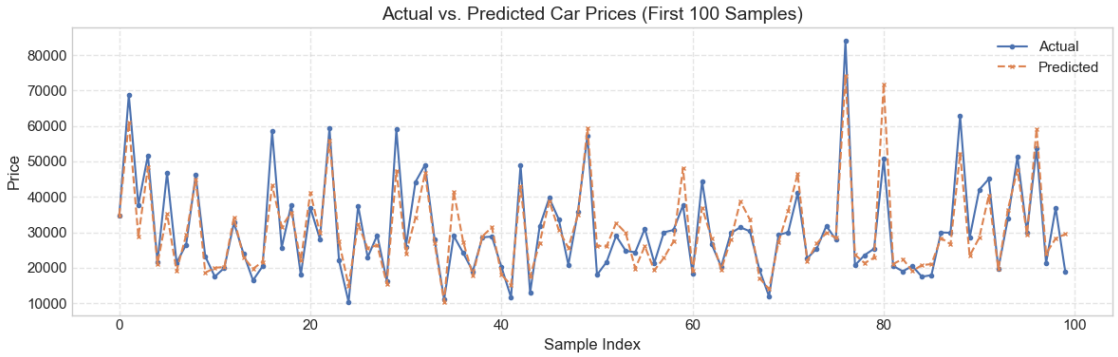
[104]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

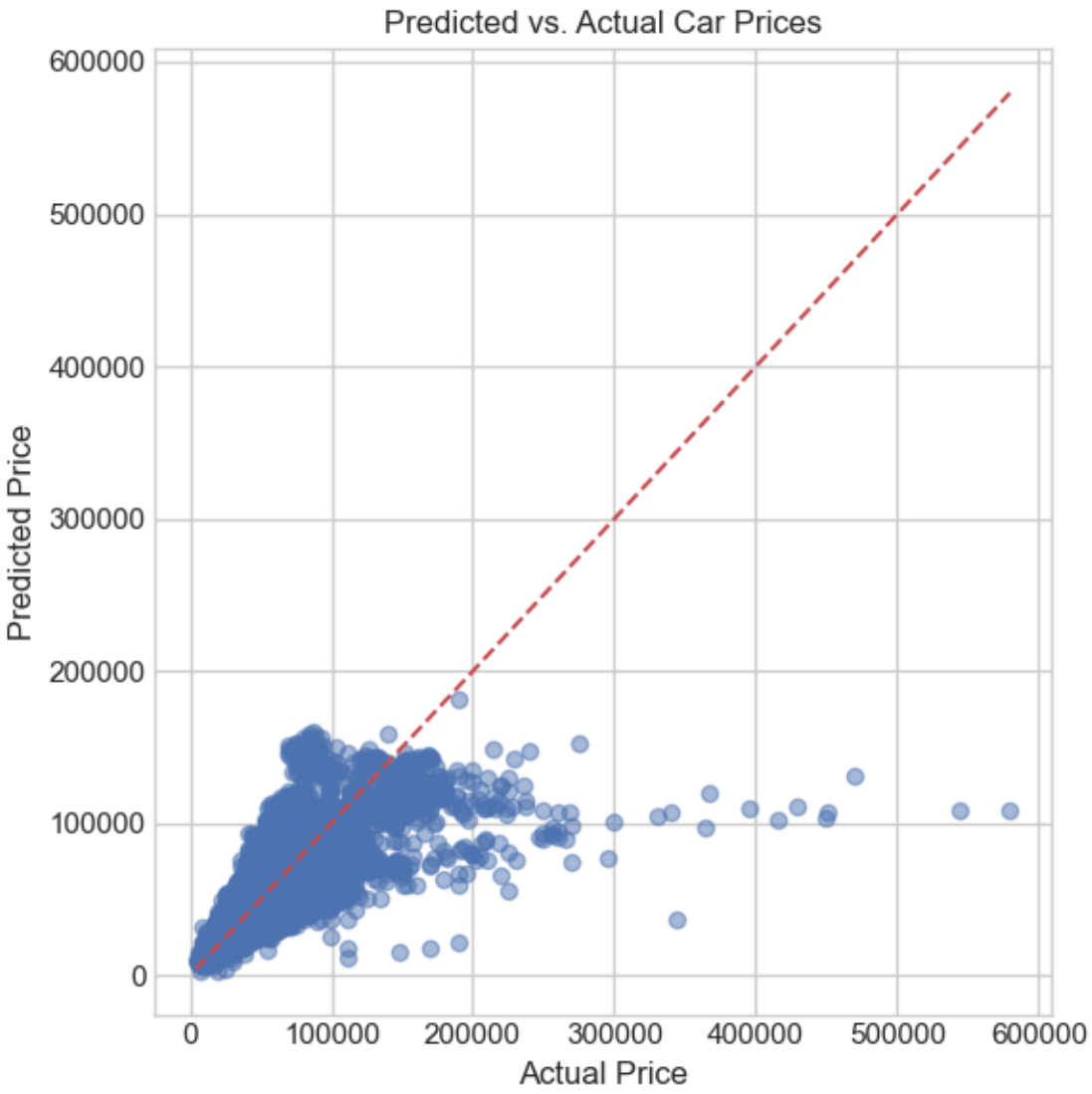
[105]:
```
from sklearn.linear_model import Ridge

ridge = Ridge(alpha=1.0)
ridge.fit(X_train_scaled, y_train_log)
```

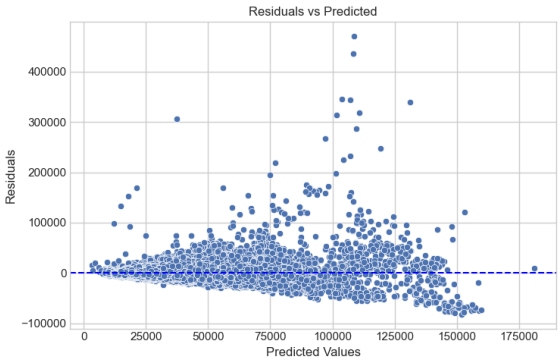
[105]:
```
Ridge()
```

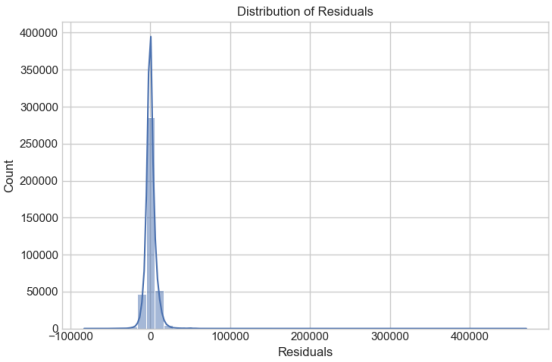
[106]:
```
features = X_train.columns #While this doesn't change, I will just keep it␣
↪regardless.
evaluate_model(ridge, X_train_scaled, X_test_scaled, y_train_log, y_test_log,␣
↪features)
```

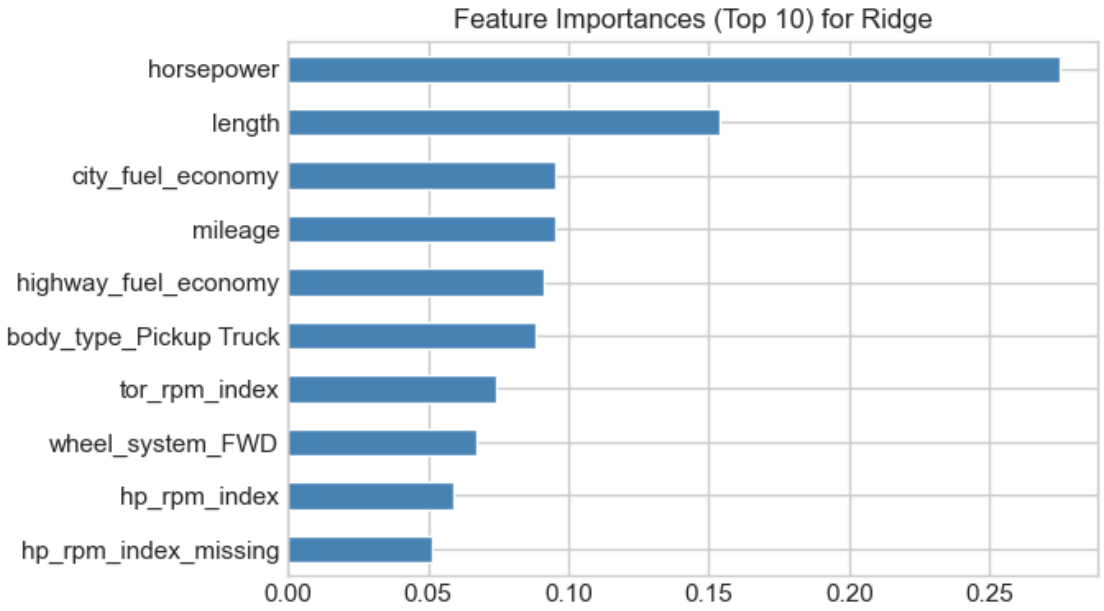
```
Model: Ridge
MAE: 4453.580094773598
MSE: 49468058.60449063
RMSE: 7033.353297289325
R².train: 0.7930391263063545
R²: 0.7871836634196221
R².adj: 0.7871555450112683
```




Predicted vs. Actual Car Prices
Predicted Price
Actual Price
600000
500000
400000
300000
200000
100000
0
0
100000
200000
300000
400000
500000
600000


Residuals vs Predicted
Residuals
Predicted Values
400000
300000
200000
100000
0
−100000
0
25000
50000
75000
100000
125000
150000
175000


Distribution of Residuals
Count
Residuals
400000
350000
300000
250000
200000
150000
100000
50000
0
−100000
0
100000
200000
300000
400000



[106]:
```
[4453.580094773598,
 49468058.60449063,
 7033.353297289325,
 0.7930391263063545,
 0.7871836634196221,
 0.7871555450112683]
```

6.2.2 Cross Validation and Hyperparameter Tuning

[107]:
```
from sklearn.linear_model import RidgeCV

ridge_cv = RidgeCV(alphas=[0.01, 0.1, 1, 10, 100], cv=5)
ridge_cv.fit(X_train_scaled, y_train_log)
```

[107]:
```
RidgeCV(alphas=[0.01, 0.1, 1, 10, 100], cv=5)
```

[108]:
```
print("Best Ridge alpha:", ridge_cv.alpha_)
```

```
Best Ridge alpha: 1.0
```

[109]:
```
# Import the Lasso Regression class with best alpha
"""ridge = Ridge(alpha = ridge_cv.alpha_)
ridge.fit(X_train_scaled, y_train_log)

features = X_train.columns
evaluate_model(ridge, X_train_scaled, X_test_scaled, y_train_log, y_test_log,␣
 ↪features)"""
```

```
# We already performed using alpha = 1.0
```

[109]:
```
'ridge = Ridge(alpha = ridge_cv.alpha_)\nridge.fit(X_train_scaled,
y_train_log)\n\nfeatures = X_train.columns\nevaluate_model(ridge,
X_train_scaled, X_test_scaled, y_train_log, y_test_log, features)'
```

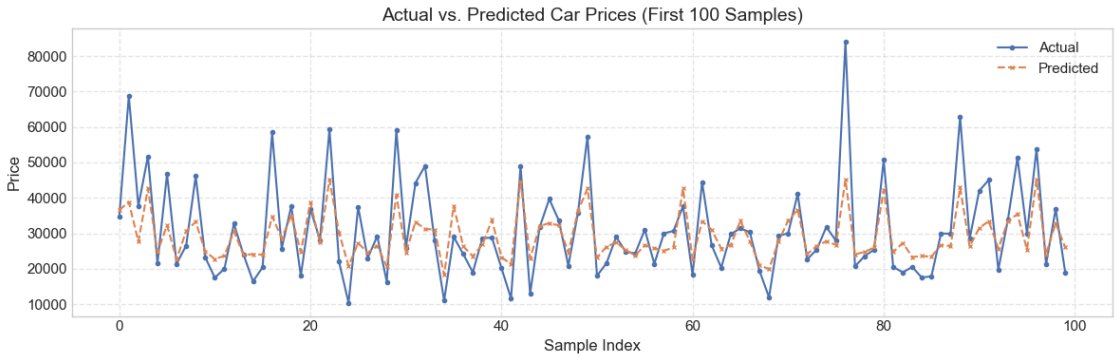
6.3 Lasso Regression

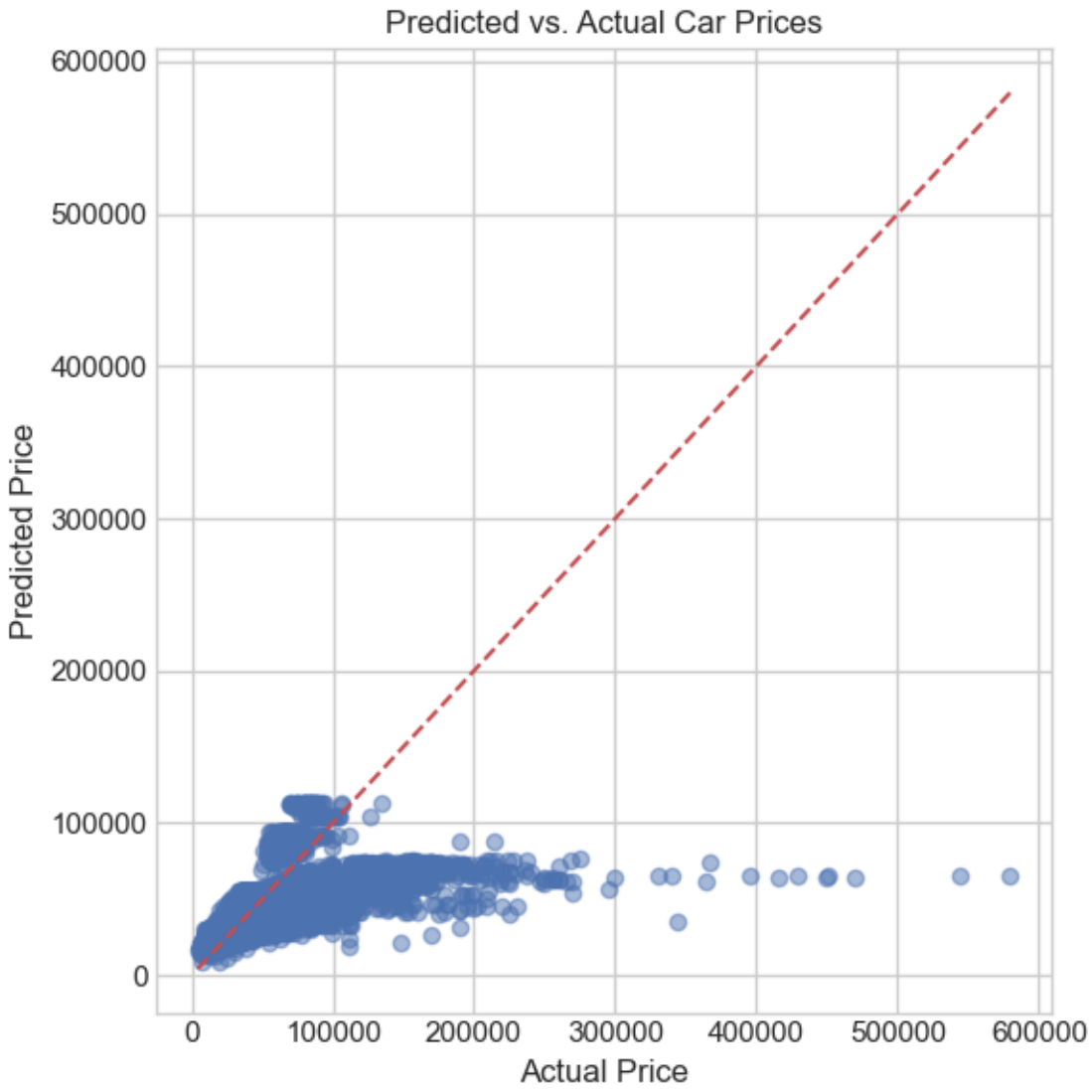
[110]:
```
from sklearn.linear_model import Lasso

lasso = Lasso(alpha=0.1)
lasso.fit(X_train_scaled, y_train_log)
```

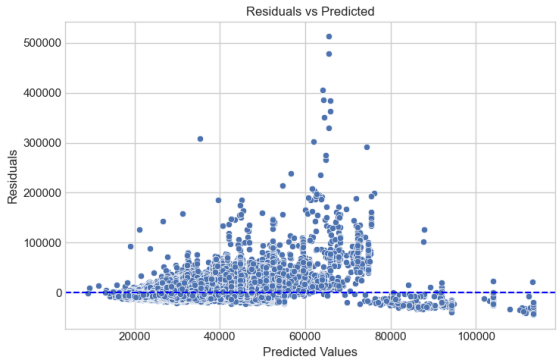
[110]:
```
Lasso(alpha=0.1)
```

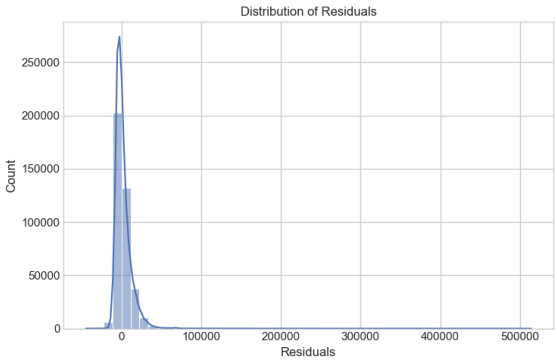
[111]:
```
features = X_train.columns
evaluate_model(lasso, X_train_scaled, X_test_scaled, y_train_log, y_test_log,␣
↪features)
```

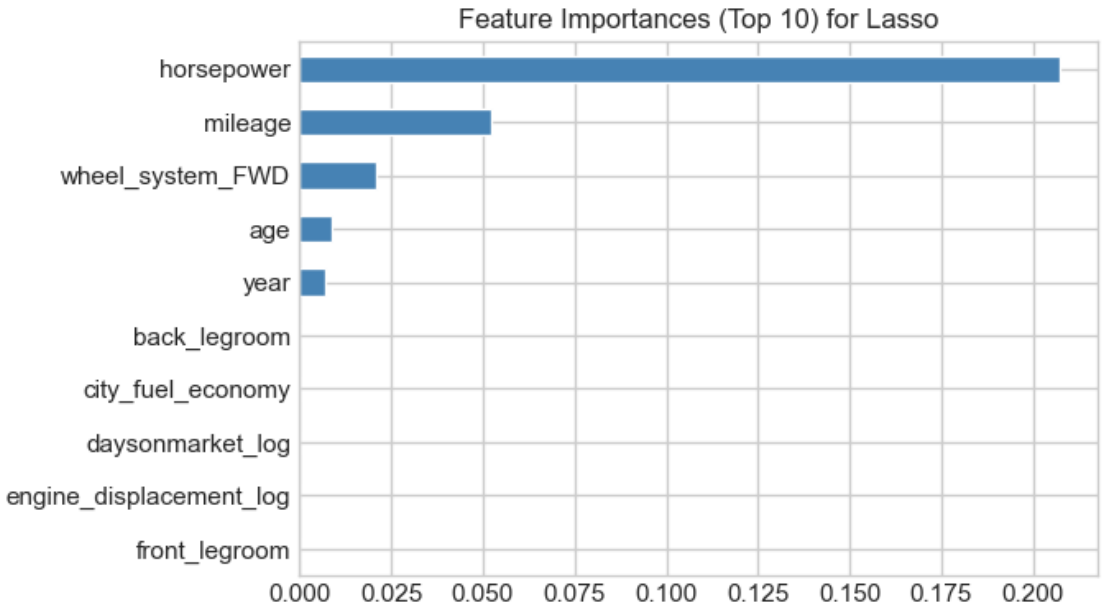
```
Model: Lasso
MAE: 6712.08263994807
MSE: 108206042.959566
RMSE: 10402.21336829648
R².train: 0.540899172737772
R²: 0.5344872164353873
R².adj: 0.5344257104497449
```




Predicted vs. Actual Car Prices
Predicted Price
Actual Price
0
100000
200000
300000
400000
500000
600000


Residuals vs Predicted
Residuals
Predicted Values
500000
400000
300000
200000
100000
0
20000
40000
60000
80000
100000


Distribution of Residuals
Count
Residuals
250000
200000
150000
100000
50000
0
0
100000
200000
300000
400000
500000



[111]:
```
[6712.08263994807,
 108206042.959566,
 10402.21336829648,
 0.540899172737772,
 0.5344872164353873,
 0.5344257104497449]
```

6.3.1 Cross Validation and Hyperparameter Tuning

[112]:
```
from sklearn.linear_model import LassoCV

lasso_cv = LassoCV(alphas=[0.001, 0.01, 0.1, 1], cv=5)
lasso_cv.fit(X_train_scaled, y_train_log)
```

[112]:
```
LassoCV(alphas=[0.001, 0.01, 0.1, 1], cv=5)
```

[113]:
```
print("Best Lasso alpha:", lasso_cv.alpha_)
```

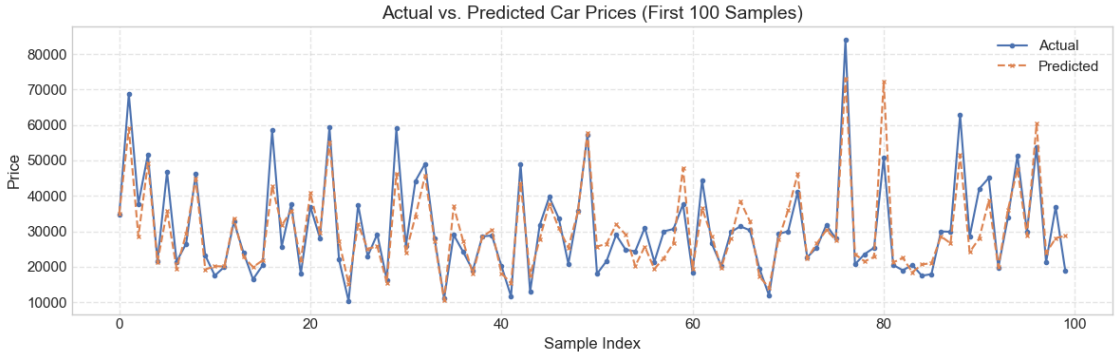
```
Best Lasso alpha: 0.001
```

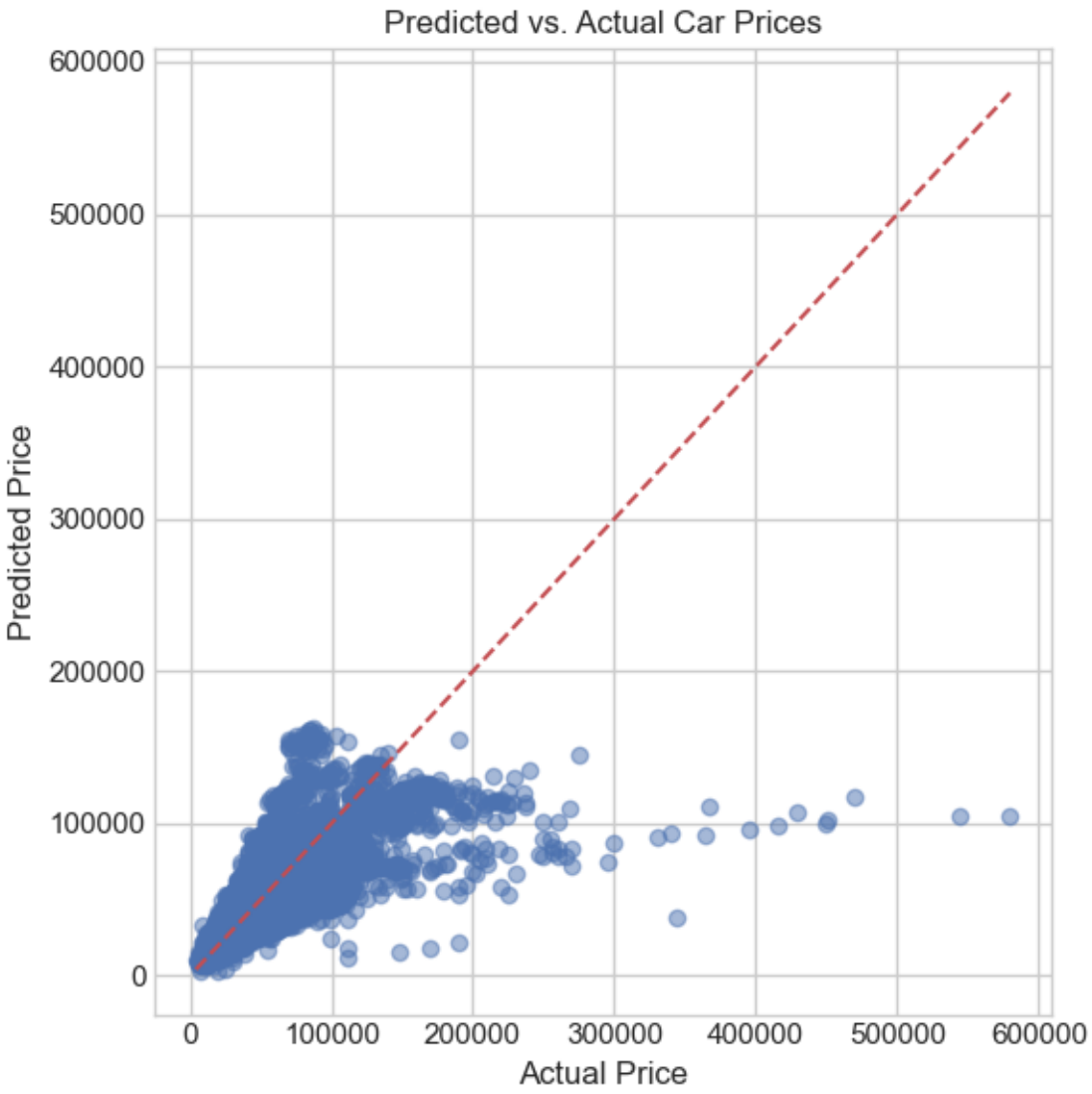
[114]:
```
# Import the Lasso Regression class with best alpha
lasso2 = Lasso(alpha = lasso_cv.alpha_) #previously we used alpha=0.1
lasso2.fit(X_train_scaled, y_train_log)
```

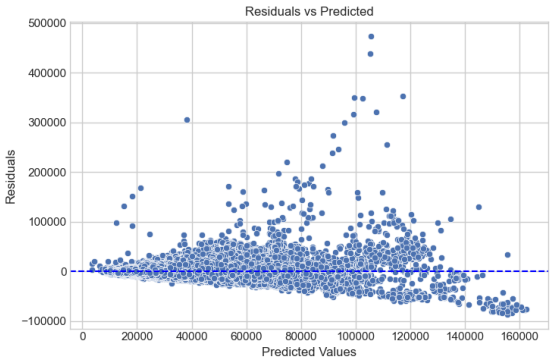
[114]:
```
Lasso(alpha=0.001)
```

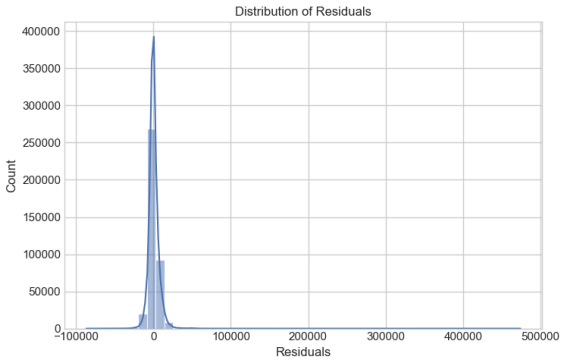
[115]:
```
features = X_train.columns
evaluate_model(lasso2, X_train_scaled, X_test_scaled, y_train_log, y_test_log,␣
↪features)
```

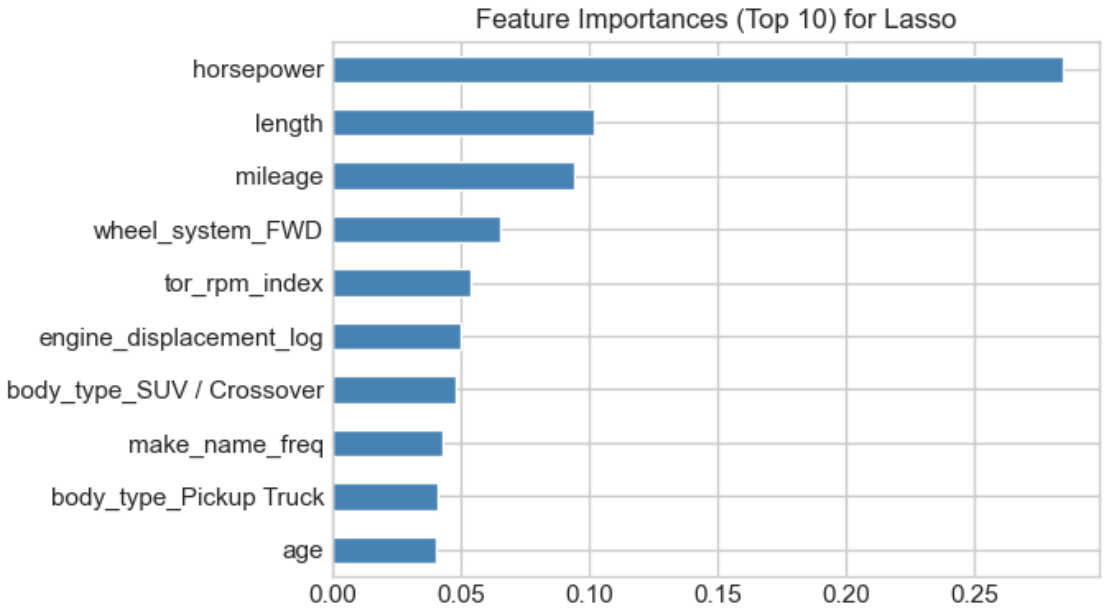
```
Model: Lasso
MAE: 4493.681916359157
MSE: 51372044.32607929
RMSE: 7167.429408517344
R².train: 0.7853669521352743
R²: 0.7789925340807994
R².adj: 0.7789633334175616
```




Predicted vs. Actual Car Prices
Predicted Price
Actual Price
600000
500000
400000
300000
200000
100000
0
0
100000
200000
300000
400000
500000
600000


Residuals vs Predicted
Residuals
Predicted Values
500000
400000
300000
200000
100000
0
−100000
0
20000
40000
60000
80000
100000
120000
140000
160000


Distribution of Residuals
Count
Residuals
400000
350000
300000
250000
200000
150000
100000
50000
0
−100000
0
100000
200000
300000
400000
500000



[115]:
```
[4493.681916359157,
 51372044.32607929,
 7167.429408517344,
 0.7853669521352743,
 0.7789925340807994,
 0.7789633334175616]
```

The evaluation of these alpha values did not yield significant improvements in the ridge regression model. However, there was a notable enhancement in the lasso regression model, with an R^2 value of 0.7790 compared to 0.5345.

6.3.2 Create a Comparison Table

[116]:
```
# Define metrics for each model using best tuned alpha value
metrics = {
    'Metric': ['MAE', 'MSE', 'RMSE', 'R².train', 'R²', 'R².adj'],
    'Linear': [4453.579376362097, 49467985.91861194, 7033.348130059534, 0.
 ↪7930394179109975, 0.7871839761212494, 0.7871558577542113],
    'Ridge':  [4453.580078969657, 49468057.0702986, 7033.35318822385, 0.
 ↪7930391321516614, 0.7871836700198636, 0.7871555516123818],
    'Lasso':  [4493.681916359157, 51372044.32607929, 7167.429408517344, 0.
 ↪7853669521352743, 0.7789925340807994, 0.7789633334175616]
}

# Create DataFrame
comparison_df = pd.DataFrame(metrics)
```

```
# Set 'Metric' column as index
comparison_df.set_index('Metric', inplace=True)

# Show the table
comparison_df
```

[116]:

Metric	Linear	Ridge	Lasso
MAE	4453.58	4453.58	4493.68
MSE	49467985.92	49468057.07	51372044.33
RMSE	7033.35	7033.35	7167.43
R^2.train	0.79	0.79	0.79
R^2	0.79	0.79	0.78
R^2.adj	0.79	0.79	0.78

The performances of Linear Regression and Ridge Regression were very similar. The Ridge did not yield much performance improvement, exhibiting regularization may not be particularly important in this case. In fact, this could also mean that, in the context of this particular data, the linear model does not show much multicollinearity and/or is not too overfitting, particularly because the Ridge regularization penalizes large coefficients and did not significantly improve performance. Lasso Regression had slightly worse performance (higher MAE, MSE, RMSE; slightly lower R^2 and adjusted R^2), which could indicate that the fact that lasso can reduce some of the coefficients to zero and could be removing useful features leading to slightly reduced predictive performance.

6.4 Decision Tree

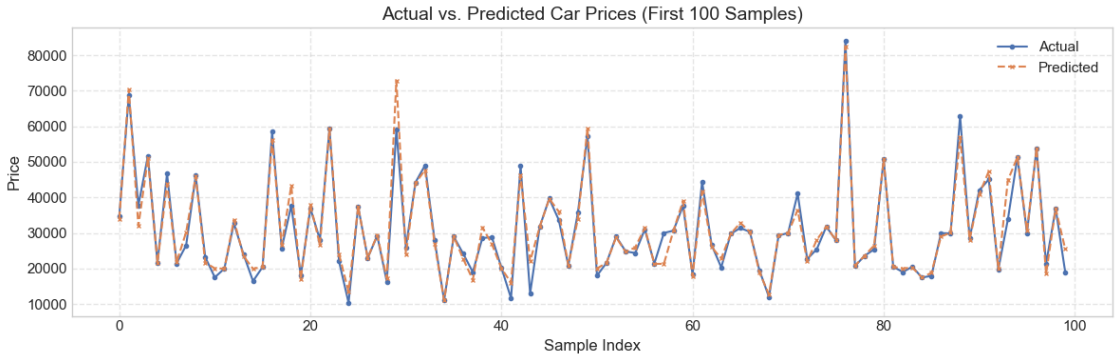
Tree-based models such as Decision Tree, Random Forest, XGBoost don't need scaling.

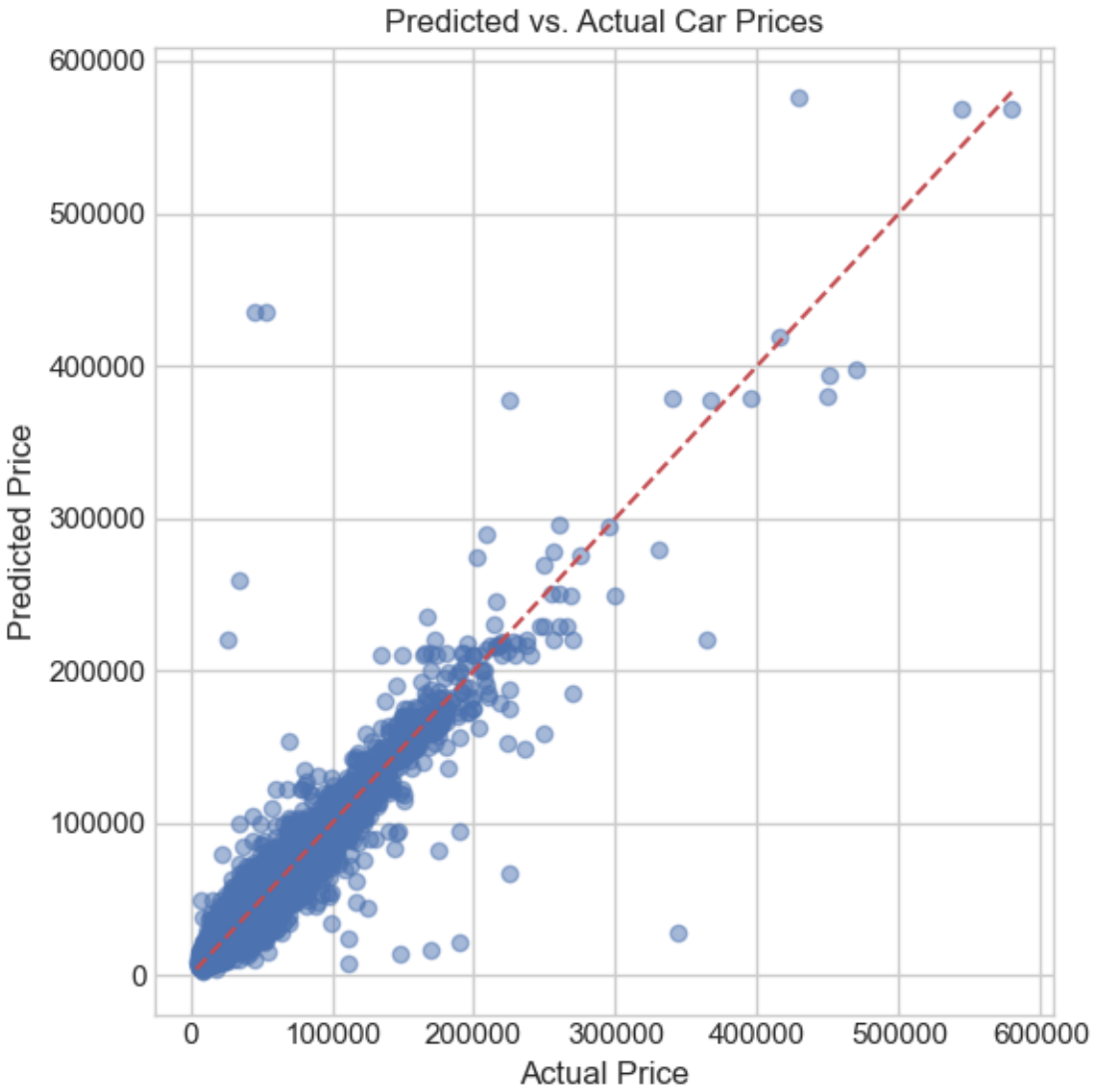
[117]:
```
# Import libraries
from sklearn.tree import DecisionTreeRegressor

# Initialize the model
modTre = DecisionTreeRegressor(random_state=54)

# Fit the model
modTre.fit(X_train, y_train_log)
```

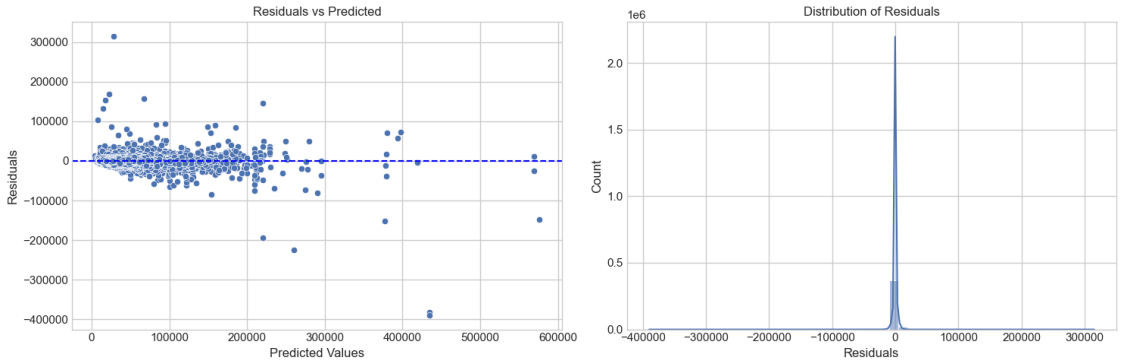
[117]:
```
DecisionTreeRegressor(random_state=54)
```

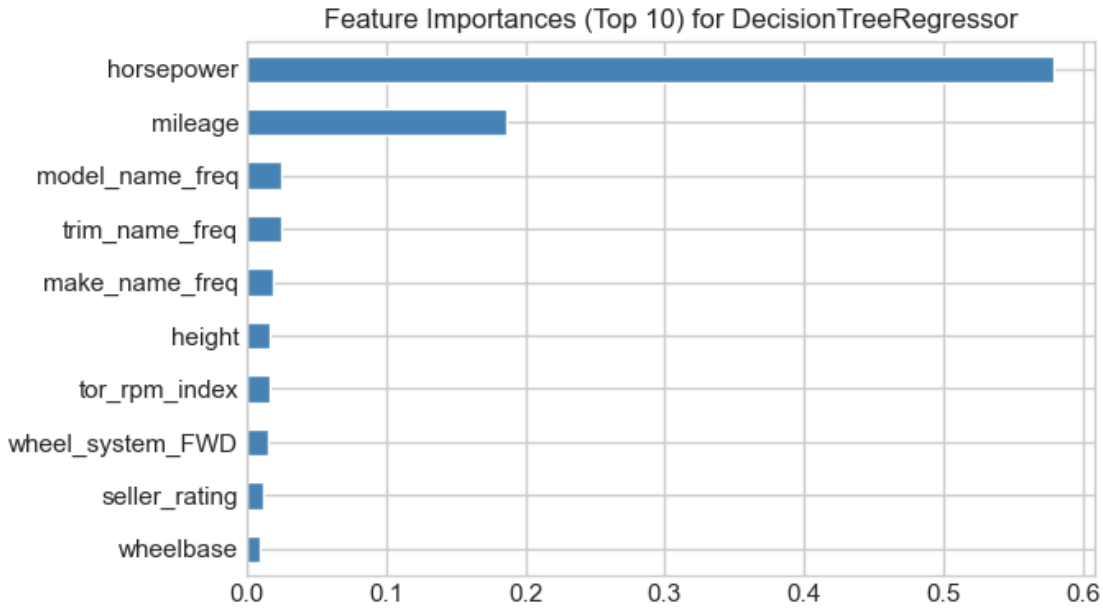
[118]:
```
features = X_train.columns
evaluate_model(modTre, X_train, X_test, y_train_log, y_test_log, features)
```

```
Model: DecisionTreeRegressor
MAE: 1852.675740372032
MSE: 11417562.74385536
RMSE: 3378.988420201431
R².train: 0.9998684628017082
R²: 0.9508805490983362
R².adj: 0.9508740591793724
```


Actual vs. Predicted Car Prices (First 100 Samples)
Actual
Predicted
Price
Sample Index
80000
70000
60000
50000
40000
30000
20000
10000
0
20
40
60
80
100


Predicted vs. Actual Car Prices
Predicted Price
Actual Price
600000
500000
400000
300000
200000
100000
0
0
100000
200000
300000
400000
500000
600000





[118]:
```
[1852.675740372032,
 11417562.74385536,
 3378.988420201431,
 0.9998684628017082,
 0.9508805490983362,
 0.9508740591793724]
```

Comparing R^2, Training R^2: 0.999 and testing R^2: 0.951 suggest that the model is overfitting. GridSearchCV employs cross-validation (CV) to divide the data into multiple training and validation folds. This approach provides a reliable estimate of model performance and minimizes the risk of overfitting to a single train-test split.

6.4.1 Cross Validation and Hyperparameter Tuning

[119]:
```
from sklearn.model_selection import GridSearchCV

# Define the grid of hyperparameters
param_grid = {
    'max_depth': [3, 5, 10, 15, 20],
    'min_samples_split': [2, 5, 10]
}

# Set up GridSearchCV
grid_search = GridSearchCV(estimator=modTre,
                           param_grid=param_grid,
                           cv=5,
                           scoring='neg_mean_squared_error',
                           n_jobs=2,
                           pre_dispatch='2*n_jobs')


# Fit on training data
grid_search.fit(X_train, y_train_log)
```

```
/opt/anaconda3/lib/python3.12/site-
packages/joblib/externals/loky/process_executor.py:752: UserWarning: A worker
stopped while some jobs were given to the executor. This can be caused by a too
short worker timeout or by a memory leak.
  warnings.warn(
```

[119]:
```
GridSearchCV(cv=5, estimator=DecisionTreeRegressor(random_state=54), n_jobs=2,
             param_grid={'max_depth': [3, 5, 10, 15, 20],
                         'min_samples_split': [2, 5, 10]},
             scoring='neg_mean_squared_error')
```

GridSearchCV is a way to determine the best combinations of hyperparameters for machine learning models. Many algorithms, such as Decision Trees, Random Forests, SVMs, and Ridge/Lasso, include hyperparameters that are set not from training, but based on best practices. It determines hyperparameter values to maximize performance metrics (R^2, for instance) or minimize error on validation data. It uses CV by splitting up the data into different training and validation folds for multiple iterations, allowing for reliable estimations of the model performance with minimal overfitting concerns. However, It can be very computationally expensive, particularly with large parameter grids or datasets. If I run into a situation like this, I will first consider RandomizedSearchCV or potentially Bayesian optimization to maximize possible efficiency in exploring large parameter spaces.

[120]:
```
# Best parameters and score

print("Best Parameters:", grid_search.best_params_)
print("Best Score (Negative MSE):", grid_search.best_score_)
```

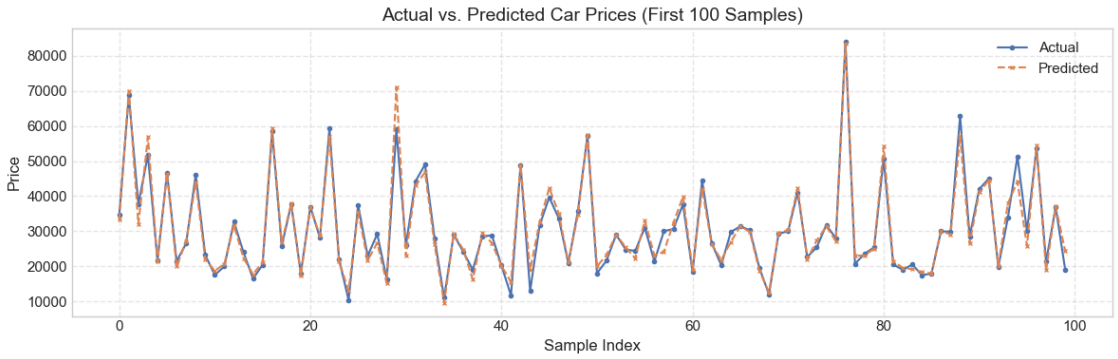
```
Best Parameters: {'max_depth': 20, 'min_samples_split': 10}
Best Score (Negative MSE): -0.007227181768133596
```

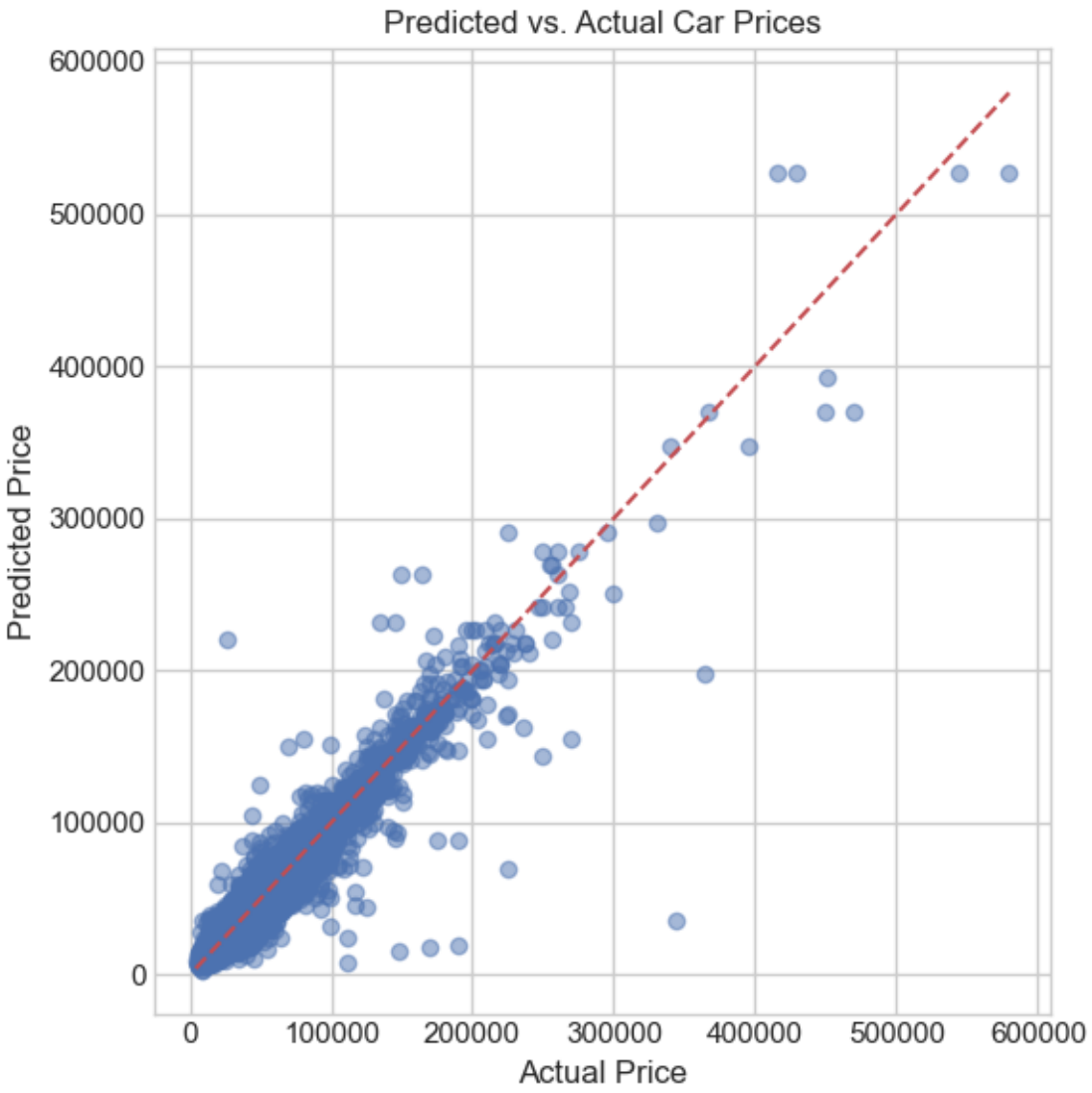
[121]:
```
# Building Decision Tree model with best parameters

modTre2 = DecisionTreeRegressor(max_depth=grid_search.best_params_['max_depth'],
                                min_samples_split=grid_search.
 ↪best_params_['min_samples_split'],
                                random_state=56)

# Fitting model
modTre2.fit(X_train,y_train_log)
```

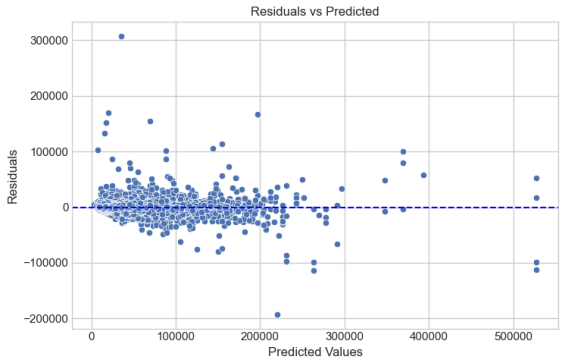
[121]:
```
DecisionTreeRegressor(max_depth=20, min_samples_split=10, random_state=56)
```

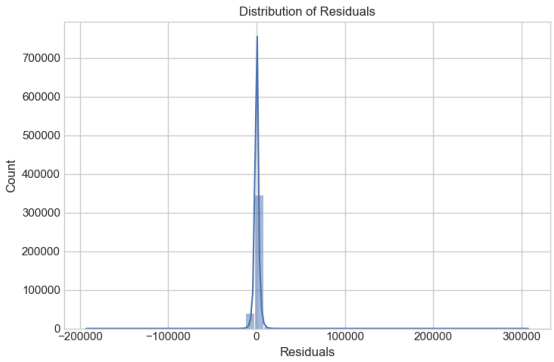
[122]:
```
features = X_train.columns
evaluate_model(modTre2, X_train, X_test, y_train_log, y_test_log, features)
```

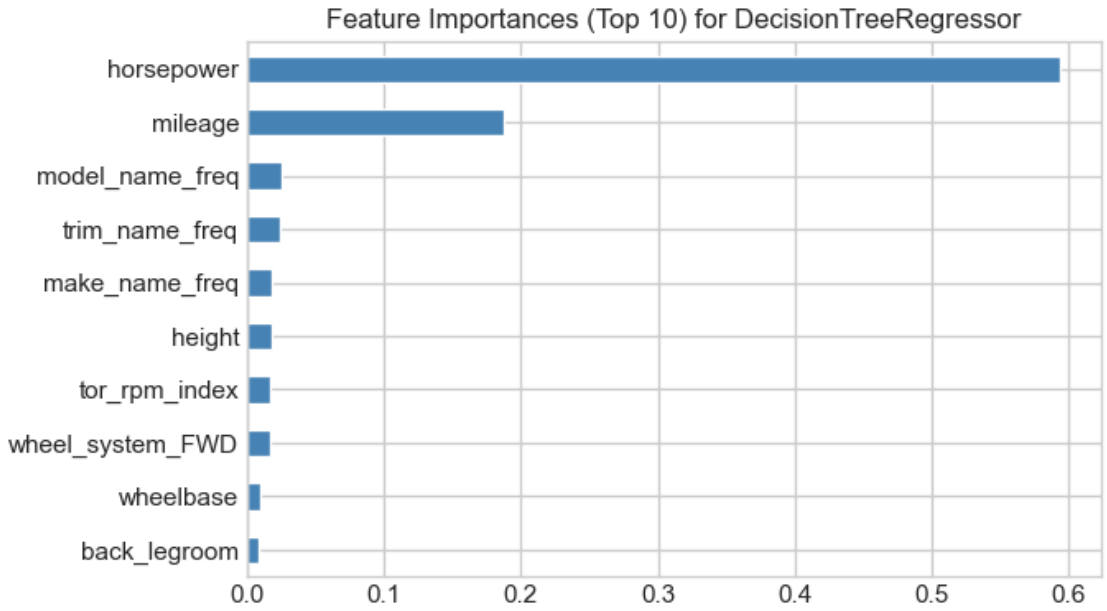
```
Model: DecisionTreeRegressor
MAE: 1874.6519179595823
MSE: 8832659.612869004
RMSE: 2971.9790734238027
R².train: 0.9769398887003303
R²: 0.9620010504939929
R².adj: 0.9619960298738827
```




Predicted vs. Actual Car Prices
Predicted Price
Actual Price
0
100000
200000
300000
400000
500000
600000


Residuals vs Predicted
Residuals
Predicted Values
300000
200000
100000
0
−100000
−200000
0
100000
200000
300000
400000
500000


Distribution of Residuals
Count
Residuals
700000
600000
500000
400000
300000
200000
100000
0
−200000
−100000
0
100000
200000
300000



[122]:
```
[1874.6519179595823,
 8832659.612869004,
 2971.9790734238027,
 0.9769398887003303,
 0.9620010504939929,
 0.9619960298738827]
```

The predicted vs actual plot has a close proximity of points in a cluster around the red line (Ideal line), especially in low prices from about \$0 to \$100,000. This tells us that we have at least a decent accuracy in predicting typical/average priced vehicles. There is more scatter on the high price side, indicating that up to the upper price range, predictions become more dispersed as prices reach over $200,000. A number of the points are above and below the red line in significant ways, indicating that there were larger residual errors for luxury and high-end vehicles. Plus, there are few clear outliers - some points in the upper right had very large variations, which would likely influence a larger root mean square error (RMSE), but these did not happen that often.

In the Residuals vs. Predicted Plot we can see that the residuals are mostly around the horizontal zero line, meaning we did not under- or over-predict systematically, which is good! However, I can see a greater spread of residuals as the predicted values get higher, especially in the upper range. This suggests heteroscedasticity, which indicates that the model may not be performing consistently in all the value ranges and may not be very accurate in the larger values.

The Distribution of Residuals plot shows that the distribution is heavily centered around zero, which is a good thing. The distribution has a narrow and leptokurtic shape (a high peak and heavier tails) indicating that most of the predictions are close to the true value with some significant errors. There is very little skew, suggesting that the model does not have a systematic bias in either direction.

The analysis of the top ten feature importances for the DecisionTreeRegressor shows that horse-

power, at ~58%, is the most important feature. Horsepower as a feature means that engine power has a significant effect on price in this dataset. One would expect, for example, that higher horsepower vehicles would generally ask higher prices. Mileage, with an importance of ~18%, is the second most important feature with much less importance that horsepower. As expected, lower mileage increases a vehicles value, especially in a used car. Horsepower and mileage together can account for ~76% of the model's predictive ability. The categorical encodings of model_name_freq, trim_name_freq, and make_name_freq also have an effect on price, but not really a very strong one. Physical dimensions such as height, wheelbase, and back legroom seem to have a negligible effect, probably due to collinearity with vehicle class and type.

[123]:
```
# Add to the comparision df
comparison_df['DecisionTree'] = [1874.6519179595823,
 8832659.612869004,
 2971.9790734238027,
 0.9769398887003303,
 0.9620010504939929,
 0.9619960298738827]

comparison_df
```

[123]:

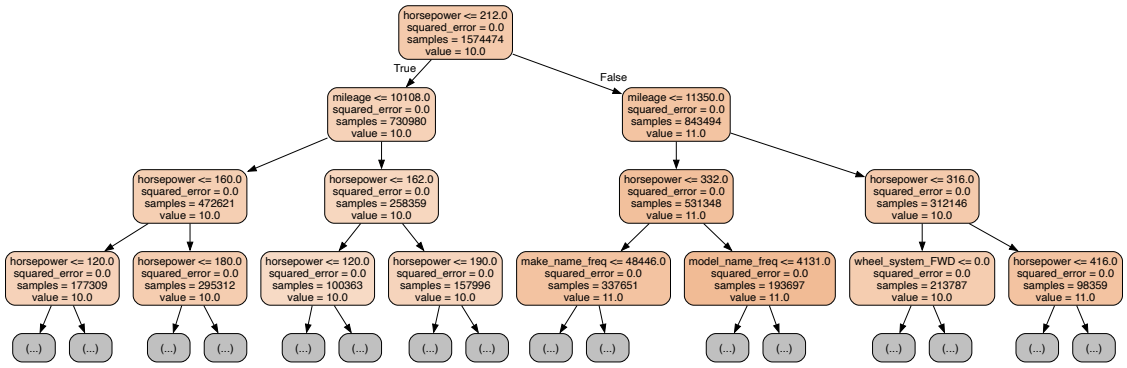
Metric	Linear	Ridge	Lasso	DecisionTree
MAE	4453.58	4453.58	4493.68	1874.65
MSE	49467985.92	49468057.07	51372044.33	8832659.61
RMSE	7033.35	7033.35	7167.43	2971.98
R^2.train	0.79	0.79	0.79	0.98
R^2	0.79	0.79	0.78	0.96
R^2.adj	0.79	0.79	0.78	0.96

The Decision Tree Regressor has the best prediction accuracy compared to the other models. Compared to Linear, Ridge, Lasso, It has a significantly lower Mean Absolute Error (MAE) of 1874.65 and Root Mean Square Error (RMSE) of 2971.98, which give a higher predictive accuracy. The R^2 value for the training data is 0.98 and for the test data, it is 0.96 which indicates that the model performed very well on the training data, and it generalizes well. The small gap between the training and test R^2 scores of 0.98 and 0.96 could suggest that the model is slightly overfitted, but still very acceptable.

6.4.2 Decision Tree diagram

Decision tree diagrams provide a straight-forward visualization of the decision-making process made in a decision tree model. The decision tree illustrates the splits in data sequentially based on feature values that inform the prediction, so multiple decision rules will be shown in the decision tree. The decision tree diagram makes it easy to understand the decision-making process while at the same time highlighting the most important features with the important features at the top of the tree. The decision tree diagram is particularly effective when communicating model information to non-technical audiences and can easily be imported into decision rules like: "If price $<$ \$10K and year $>$ 2015, then …"

[124]:
```
from sklearn.tree import export_graphviz
from graphviz import Source
from IPython.display import SVG, display

# Decision tree diagram - with modified parameters to avoid memory issues
graph = Source(export_graphviz(modTre,
                               out_file=None,
                               feature_names=features,
                               filled=True,
                               max_depth=3,       # Limit depth to avoid␣
 ↪complexity
                               proportion=False,  # Simplify the output
                               rounded=True,      # Rounded boxes look better
                               precision=0))      # Reduce precision of numbers

# Try alternative rendering approach
try:
    display(SVG(graph.pipe(format='svg')))
except:
    # Save to file instead of direct rendering
    graph.render("decision_tree", format="svg", cleanup=True)
    from IPython.display import Image
    display(Image("decision_tree.svg"))
```



6.5 Random Forest Regressor

The next algorithm of machine learning, which I will be using here will be Random Forest Regressor. Random Forest Regressor consists of a several decision trees (bagging), and reduces overfitting and improves generalization. Bagging averages the predictions of different trees, leading to more stable predictions. Since it aggregates the results of multiple trees, it is also less influenced by outliers and noise, as the influence of any one anomalous prediction is diluted through the ensemble average. Also, it can accommodate high-dimensionality, often working very well with large feature sets and datasets. However, it might not be as easy to visualize the entire forest and to know how individual decisions arrive at conclusions. It also can be slower to train and predict with, especially if the

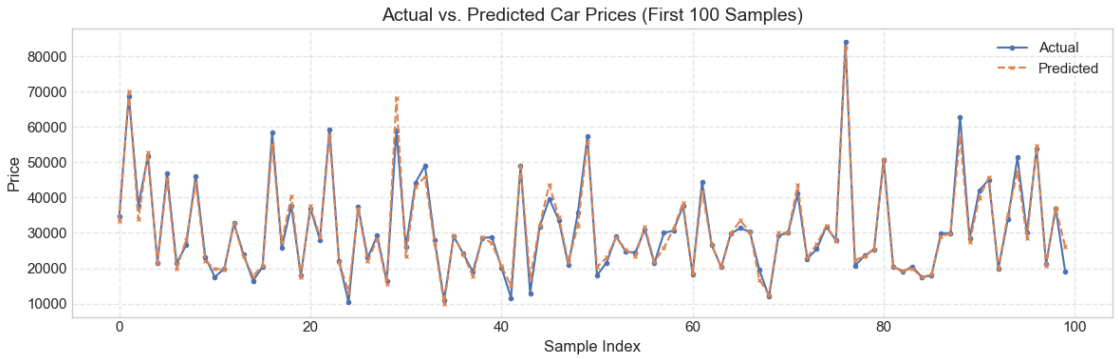
model consists of a high number of trees or features. This model is also heavier; it will require more memory and CPU/GPU resources during both training and inference.

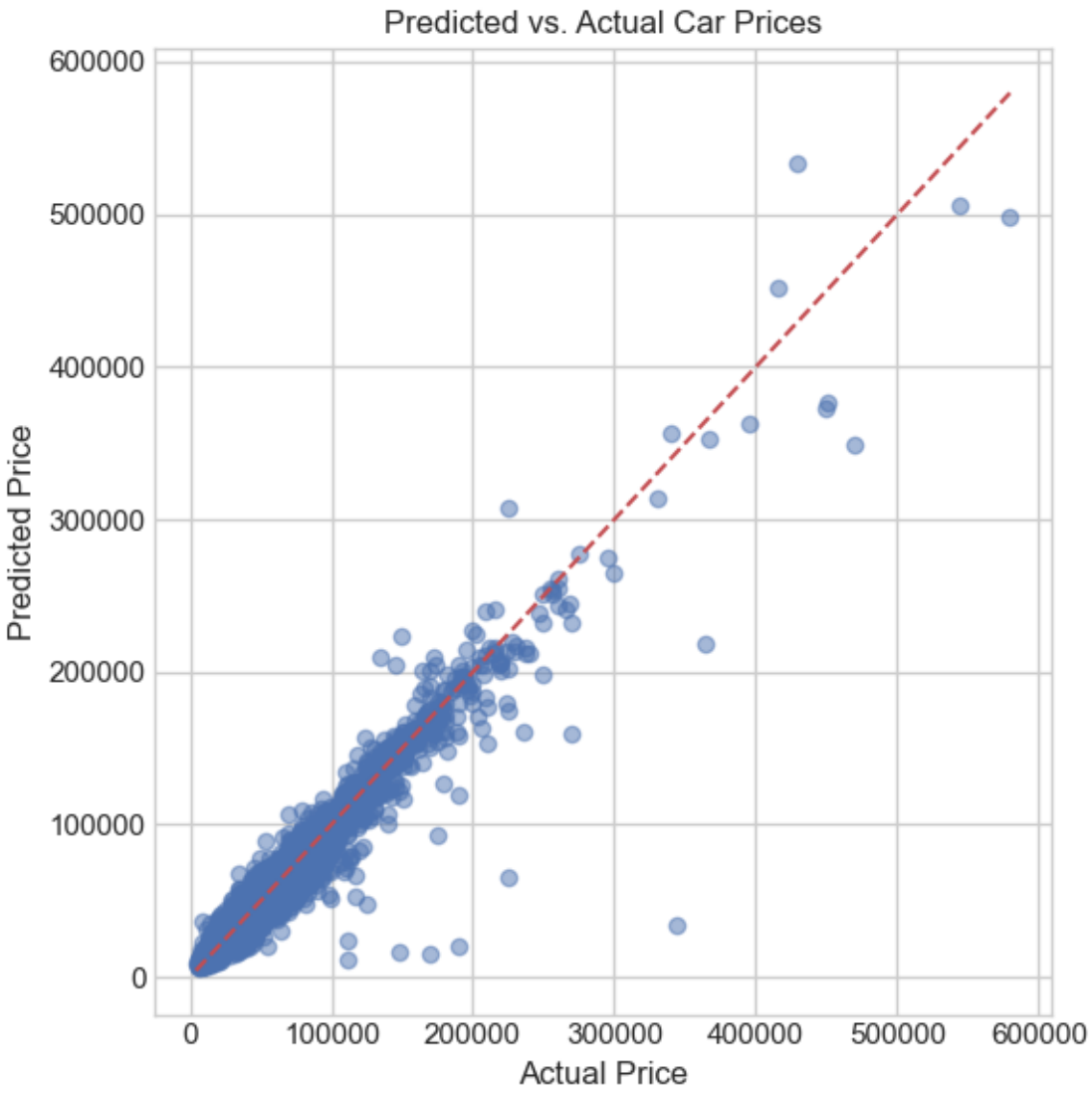
[125]:
```
from sklearn.ensemble import RandomForestRegressor

# Initialize and train the Random Forest
modRF = RandomForestRegressor(random_state=56)
modRF.fit(X_train, y_train_log)
```

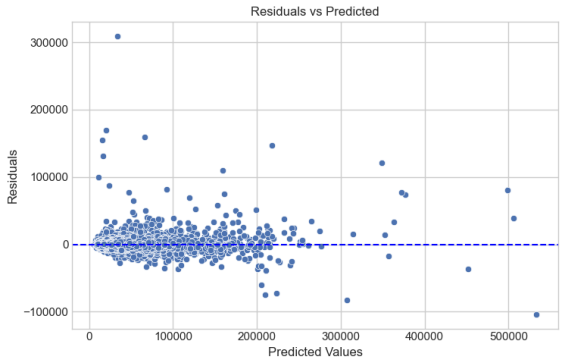
[125]:
```
RandomForestRegressor(random_state=56)
```

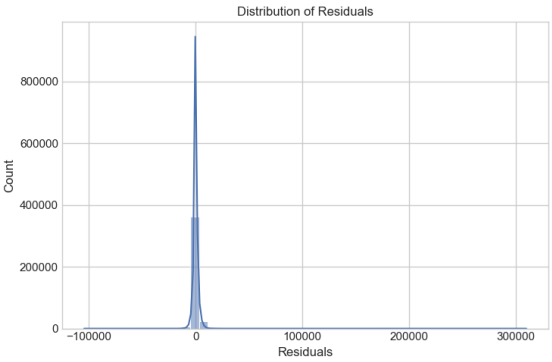
[126]:
```
features = X_train.columns
evaluate_model(modRF, X_train, X_test, y_train_log, y_test_log, features)
```

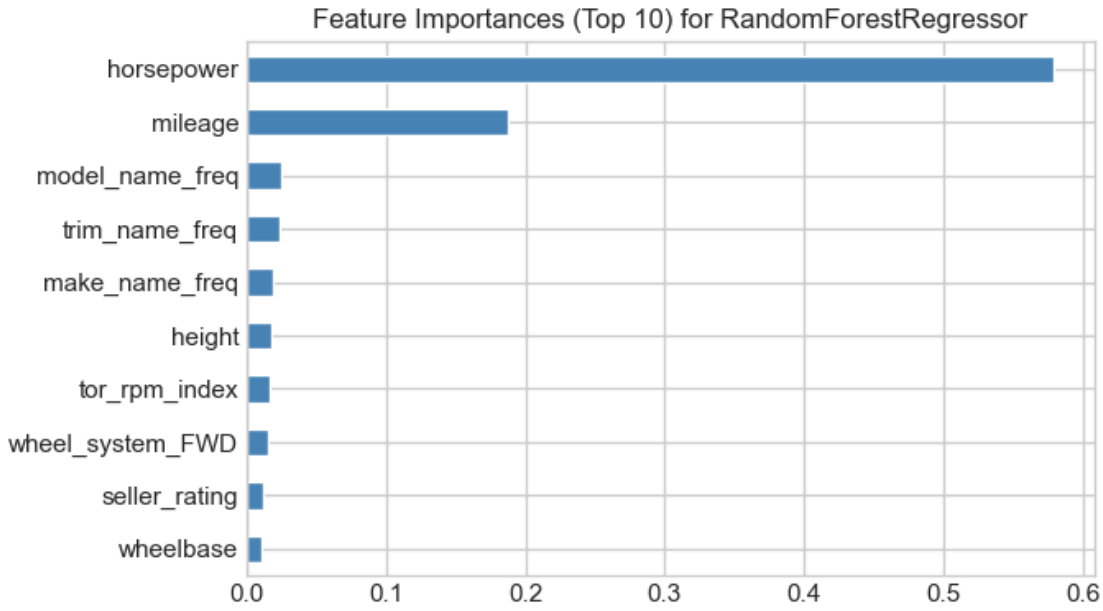
```
Model: RandomForestRegressor
MAE: 1460.0437125152891
MSE: 5983477.277501953
RMSE: 2446.1147310586134
R².train: 0.9955679727778771
R²: 0.9742585063951893
R².adj: 0.974255105294313
```




Predicted vs. Actual Car Prices
Predicted Price
Actual Price
0
100000
200000
300000
400000
500000
600000


Residuals vs Predicted
Residuals
Predicted Values
300000
200000
100000
0
−100000
0
100000
200000
300000
400000
500000


Distribution of Residuals
Count
Residuals
800000
600000
400000
200000
0
−100000
0
100000
200000
300000



[126]:
```
[1460.0437125152891,
 5983477.277501953,
 2446.1147310586134,
 0.9955679727778771,
 0.9742585063951893,
 0.974255105294313]
```

When working with large datasets and trying to optimize a machine learning model quickly, it is crucial to choose an appropriate hyperparameter tuning strategy and process data size effectively to improve performance and scalability. The most common hyperparameter tuning techniques are GridSearchCV and RandomizedSearchCV in machine learning models.

GridSearchCV evaluates every combination of parameters which can be computationally intensive depending on the amount of parameters and options; whereas RandomizedSearchCV randomly samples a set number of parameters, which enables more efficient exploration of the hyperparameter space. RandomizedSearchCV is also suitable for constrained computing environments or tight deadlines, which is often the case in industry settings.

Training with massive datasets can also take longer when training with the entire dataset, so data sampling is critical. Using sampling allows for rapid prototyping and debugging of models, as an strategy that allows for substantially decrease of iteration time or or to be able to test many algorithms and tuning strategies without waiting hours for models to train for long periods of time.

6.5.1 Cross Validation and Hyperparameter Tuning

[127]:
```
from sklearn.model_selection import RandomizedSearchCV

# Step 1: Subsample the data
sample_size = 50000   #I also tried with 100000 samples, got the same paramters.
X_sub = X.sample(sample_size, random_state=36)
y_sub_log = y.loc[X_sub.index]

# Step 2: Define hyperparameter search space
param_dist = {
    'n_estimators': [100, 200, 300],
    'max_depth': [10, 20, 100, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['sqrt', 'log2']
}

# Step 3: Configure RandomizedSearchCV with safe memory settings
random_search = RandomizedSearchCV(
    modRF,
    param_distributions=param_dist,
    n_iter=20,
    cv=3,
    scoring='neg_mean_squared_error',
    verbose=1,
    n_jobs=2,
    pre_dispatch='2*n_jobs',
    random_state=56
)

# Step 4: Run tuning
random_search.fit(X_sub, y_sub_log)
```

```
Fitting 3 folds for each of 20 candidates, totalling 60 fits
```

```
/opt/anaconda3/lib/python3.12/site-
packages/joblib/externals/loky/process_executor.py:752: UserWarning: A worker
stopped while some jobs were given to the executor. This can be caused by a too
short worker timeout or by a memory leak.
  warnings.warn(
```

[127]:
```
RandomizedSearchCV(cv=3, estimator=RandomForestRegressor(random_state=56),
                   n_iter=20, n_jobs=2,
                   param_distributions={'max_depth': [10, 20, 100, None],
                                        'max_features': ['sqrt', 'log2'],
                                        'min_samples_leaf': [1, 2, 4],
```

```
                                    'min_samples_split': [2, 5, 10],
                                    'n_estimators': [100, 200, 300]},
                        random_state=56, scoring='neg_mean_squared_error',
                        verbose=1)
```

[128]:
```
# Get best parameters
random_search.best_params_
```

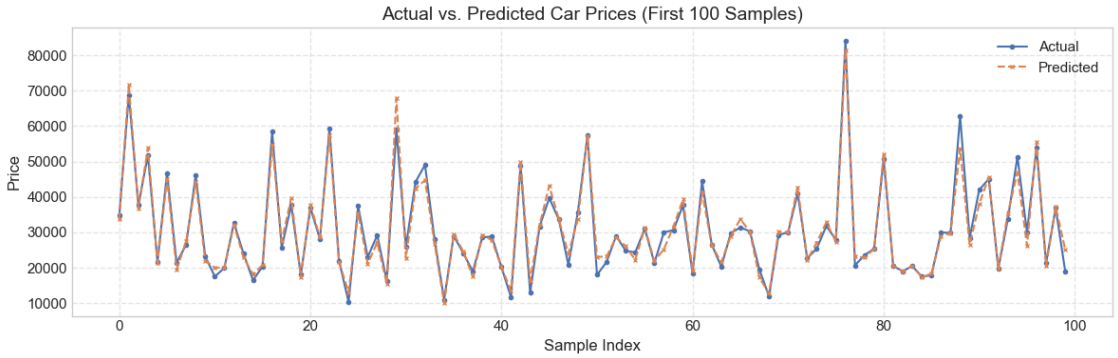
[128]:
```
{'n_estimators': 200,
 'min_samples_split': 5,
 'min_samples_leaf': 1,
 'max_features': 'sqrt',
 'max_depth': 100}
```

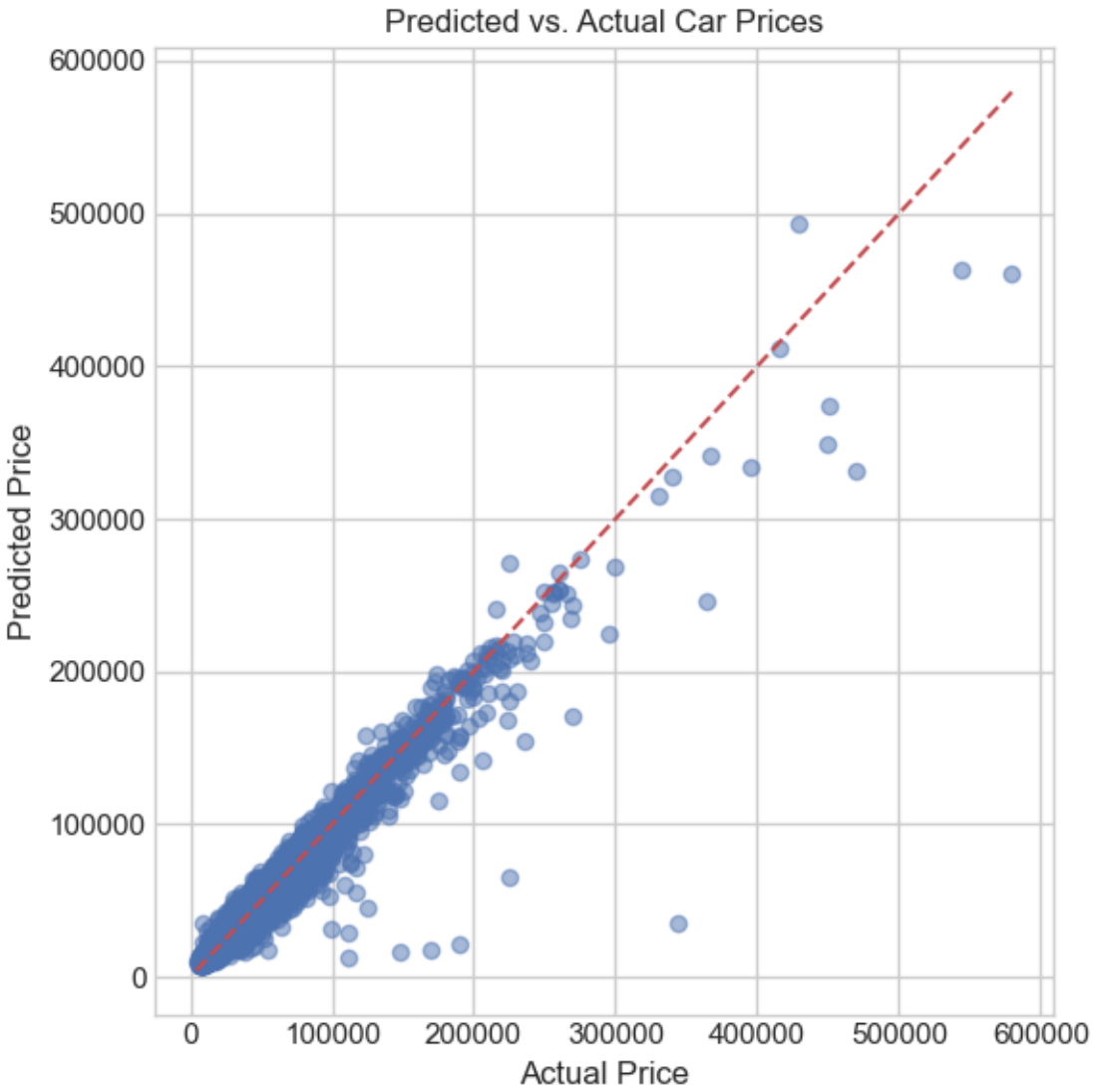
[129]:
```
# Initialize and train the Random Forest
modRF2 = RandomForestRegressor(n_estimators=random_search.
 ↪best_params_['n_estimators'],
                               min_samples_split=random_search.
 ↪best_params_['min_samples_split'],
                               min_samples_leaf=random_search.
 ↪best_params_['min_samples_leaf'],
                               max_features=random_search.
 ↪best_params_['max_features'],
                               max_depth=random_search.
 ↪best_params_['max_depth'],
                               random_state=56)
modRF2.fit(X_train, y_train_log)
```

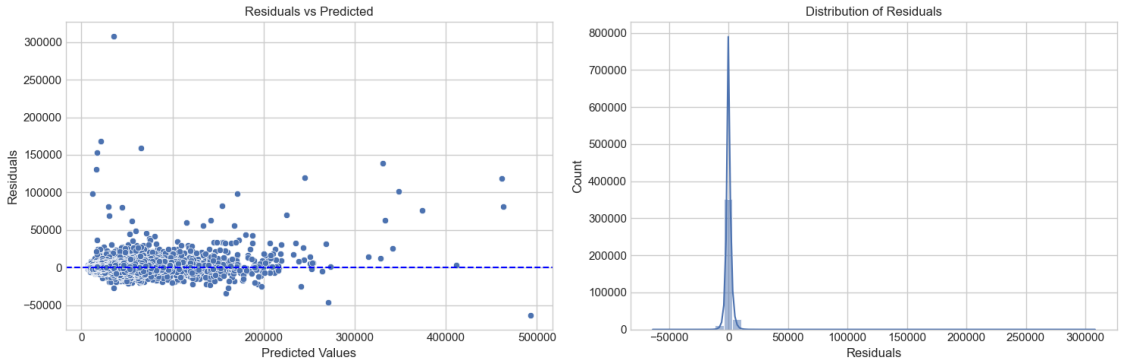
[129]:
```
RandomForestRegressor(max_depth=100, max_features='sqrt', min_samples_split=5,
                      n_estimators=200, random_state=56)
```

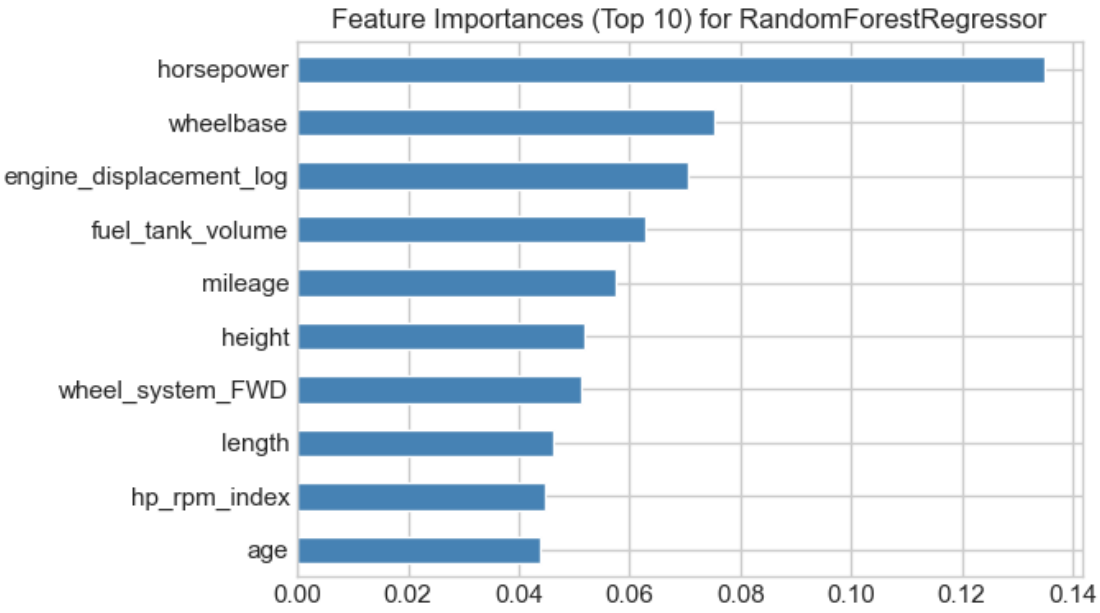
[130]:
```
evaluate_model(modRF2, X_train, X_test, y_train_log, y_test_log, features)
```

```
Model: RandomForestRegressor
MAE: 1599.6188576418836
MSE: 6349831.401792971
RMSE: 2519.887180369981
R².train: 0.9897777956165154
R²: 0.9726824157859724
R².adj: 0.9726788064437549
```


Actual vs. Predicted Car Prices (First 100 Samples)
Actual
Predicted
Price
Sample Index
80000
70000
60000
50000
40000
30000
20000
10000
0
20
40
60
80
100


Predicted vs. Actual Car Prices
Predicted Price
Actual Price
600000
500000
400000
300000
200000
100000
0
0
100000
200000
300000
400000
500000
600000





[130]:
```
[1599.6188576418836,
 6349831.401792971,
 2519.887180369981,
 0.9897777956165154,
 0.9726824157859724,
 0.9726788064437549]
```

[131]:
```
# Add to the comparision df
comparison_df['RandomForest'] = [1599.6188576418836,
 6349831.401792971,
 2519.887180369981,
 0.9897777956165154,
 0.9726824157859724,
```

```
 0.9726788064437549]

comparison_df
```

[131]:

Metric	Linear	Ridge	Lasso	DecisionTree	RandomForest
MAE	4453.58	4453.58	4493.68	1874.65	1599.62
MSE	49467985.92	49468057.07	51372044.33	8832659.61	6349831.40
RMSE	7033.35	7033.35	7167.43	2971.98	2519.89
R^2.train	0.79	0.79	0.79	0.98	0.99
R^2	0.79	0.79	0.78	0.96	0.97
R^2.adj	0.79	0.79	0.78	0.96	0.97

The Random Forest model demonstrated a better performance across all metrics we have evaluated as it performed much better than Linear, Ridge, Lasso, and Decision Tree models. It achieved the least errors and the smallest difference between the train.R^2 (0.99) and the test.R^2 (0.97), which implies minimal overfitting. This is likely due to a result from its ensemble characteristics. This model provides effective accuracy and generalization, making it the most dependable option in this analysis.

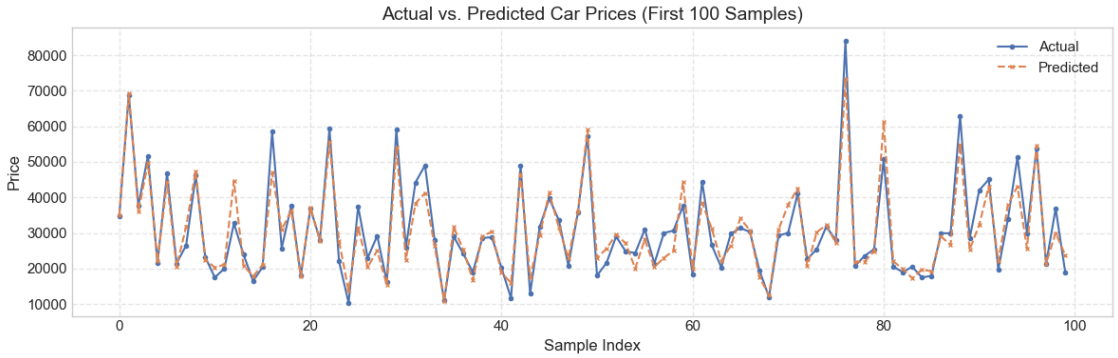
6.6 Gradient Boosting Regressor

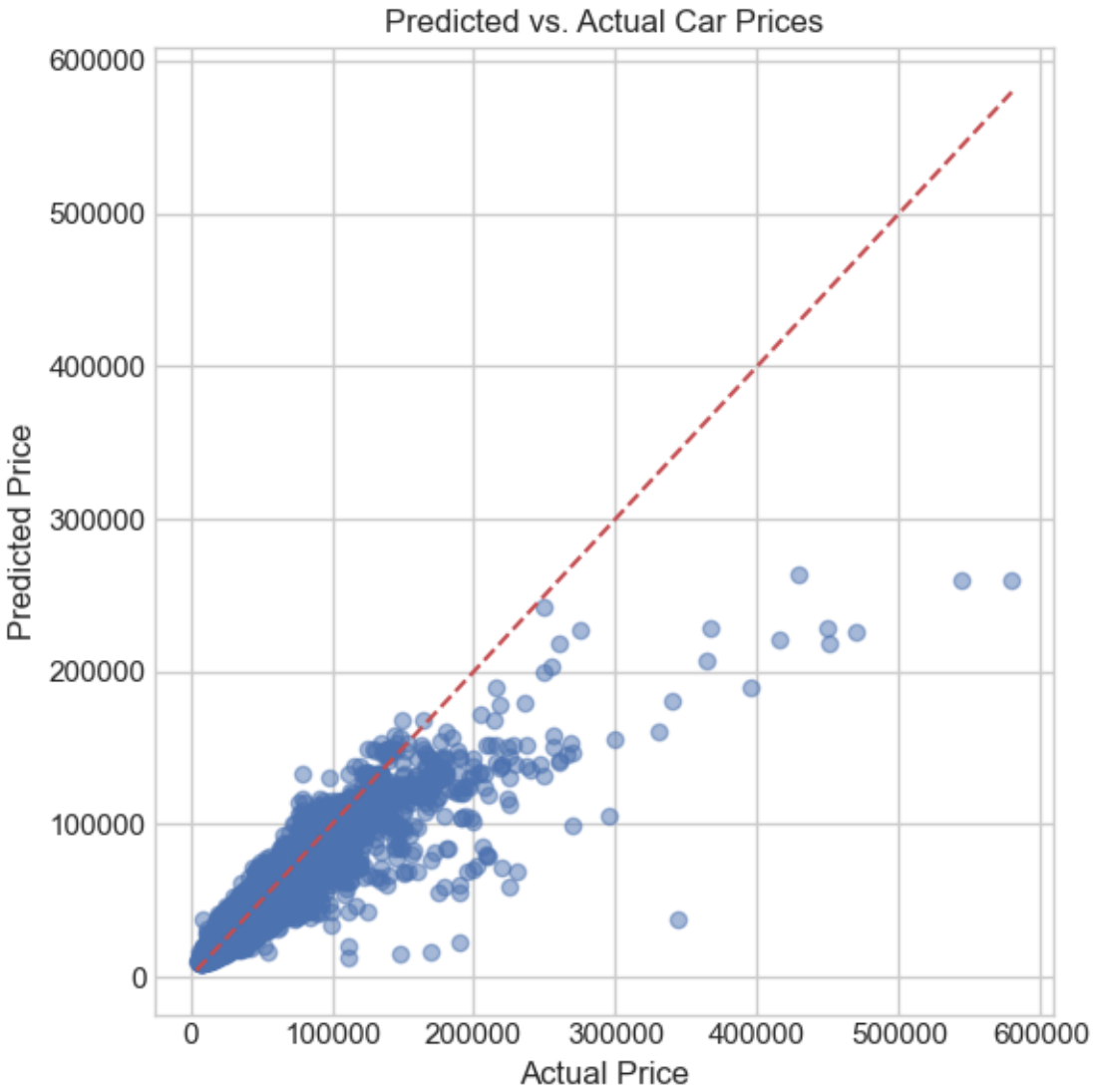
[132]:
```
from sklearn.ensemble import GradientBoostingRegressor

# Import Gradient Boosting Regressor class
modGB = GradientBoostingRegressor(random_state=56)

# Fit the Gradient Boosting model to the training data
modGB.fit(X_train,y_train_log)
```

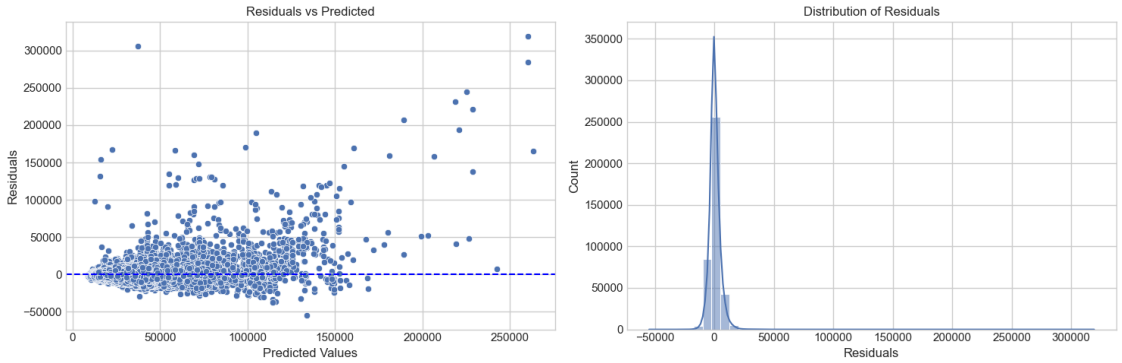
[132]:
```
GradientBoostingRegressor(random_state=56)
```

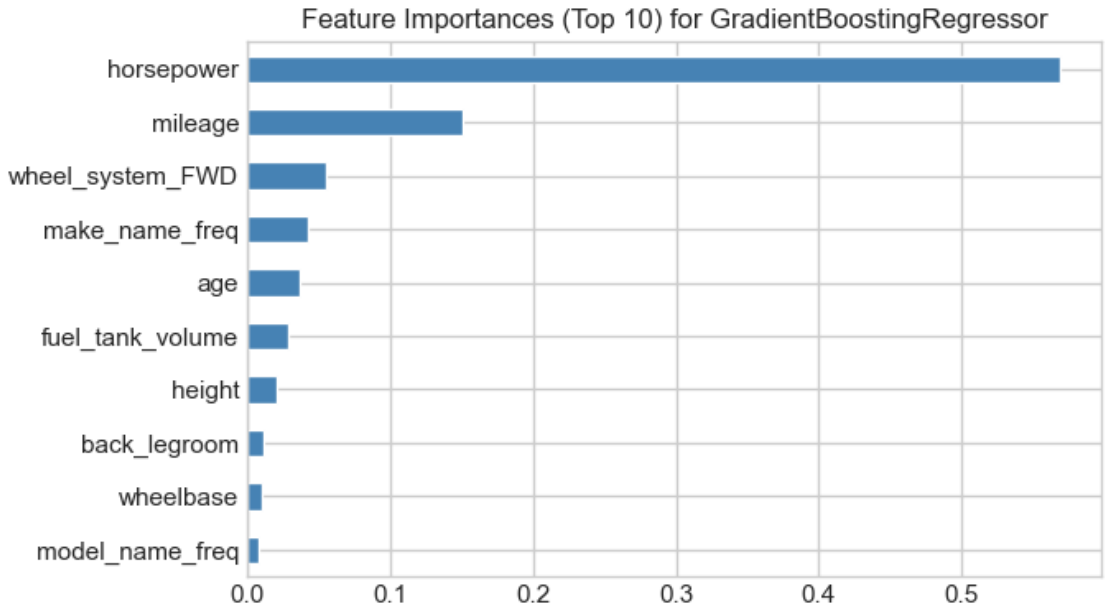
[133]:
```
evaluate_model(modGB, X_train, X_test, y_train_log, y_test_log, features)
```

```
Model: GradientBoostingRegressor
MAE: 3313.681812275416
MSE: 25579124.18017485
RMSE: 5057.5808624454885
R².train: 0.8931872328509118
R²: 0.8899561524238753
R².adj: 0.8899416128547205
```


Actual vs. Predicted Car Prices (First 100 Samples)
Actual
Predicted
Price
Sample Index
80000
70000
60000
50000
40000
30000
20000
10000
0
20
40
60
80
100


Predicted vs. Actual Car Prices
Predicted Price
Actual Price
600000
500000
400000
300000
200000
100000
0
0
100000
200000
300000
400000
500000
600000





[133]:
```
[3313.681812275416,
 25579124.18017485,
 5057.5808624454885,
 0.8931872328509118,
 0.8899561524238753,
 0.8899416128547205]
```

6.6.1 Cross Validation and Hyperparameter Tuning

[134]:
```
# Step 2: Define hyperparameter search space
param_dist = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.01, 0.05, 0.1],
```

```
    'max_depth': [3, 5, 7],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 3, 5],
    'max_features': ['sqrt', 'log2']
}

# Step 3: Configure RandomizedSearchCV with safe memory settings
random_search = RandomizedSearchCV(
    modGB,
    param_distributions=param_dist,
    n_iter=20,
    cv=3,
    scoring='neg_mean_squared_error',
    verbose=1,
    n_jobs=2,
    pre_dispatch='2*n_jobs',
    random_state=56
)

# Step 4: Run tuning (this is our subsampled data)
random_search.fit(X_sub, y_sub_log)
```

```
Fitting 3 folds for each of 20 candidates, totalling 60 fits
```

```
/opt/anaconda3/lib/python3.12/site-
packages/joblib/externals/loky/process_executor.py:752: UserWarning: A worker
stopped while some jobs were given to the executor. This can be caused by a too
short worker timeout or by a memory leak.
  warnings.warn(
```

[134]:
```
RandomizedSearchCV(cv=3, estimator=GradientBoostingRegressor(random_state=56),
                   n_iter=20, n_jobs=2,
                   param_distributions={'learning_rate': [0.01, 0.05, 0.1],
                                        'max_depth': [3, 5, 7],
                                        'max_features': ['sqrt', 'log2'],
                                        'min_samples_leaf': [1, 3, 5],
                                        'min_samples_split': [2, 5, 10],
                                        'n_estimators': [100, 200, 300]},
                   random_state=56, scoring='neg_mean_squared_error',
                   verbose=1)
```

[135]:
```
# Get best parameters
random_search.best_params_
```

[135]:
```
{'n_estimators': 200,
 'min_samples_split': 10,
 'min_samples_leaf': 5,
 'max_features': 'sqrt',
```

```
 'max_depth': 7,
 'learning_rate': 0.05}
```

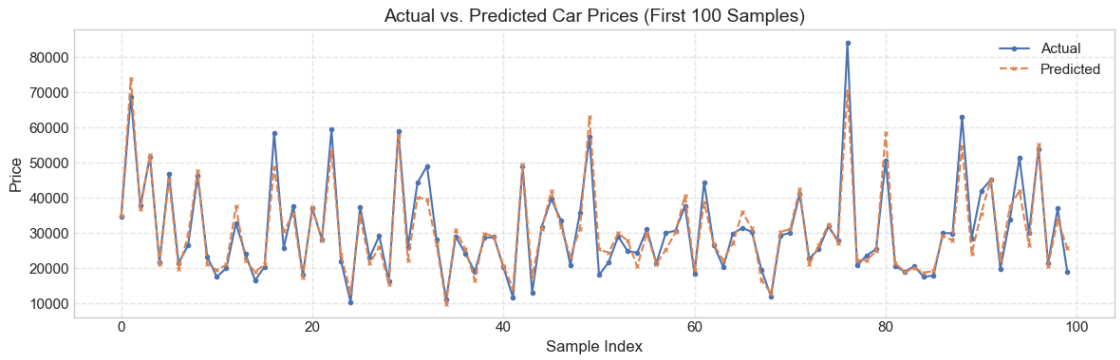
[136]:
```
# Initialize and train the Random Forest
modGB2 = GradientBoostingRegressor(n_estimators=random_search.
 ↪best_params_['n_estimators'],
                                   min_samples_split=random_search.
 ↪best_params_['min_samples_split'],
                                   min_samples_leaf=random_search.
 ↪best_params_['min_samples_leaf'],
                                   max_features= random_search.
 ↪best_params_['max_features'],
                                   max_depth= random_search.
 ↪best_params_['max_depth'],
                                   learning_rate= random_search.
 ↪best_params_['learning_rate'],
                                   random_state=56)
```

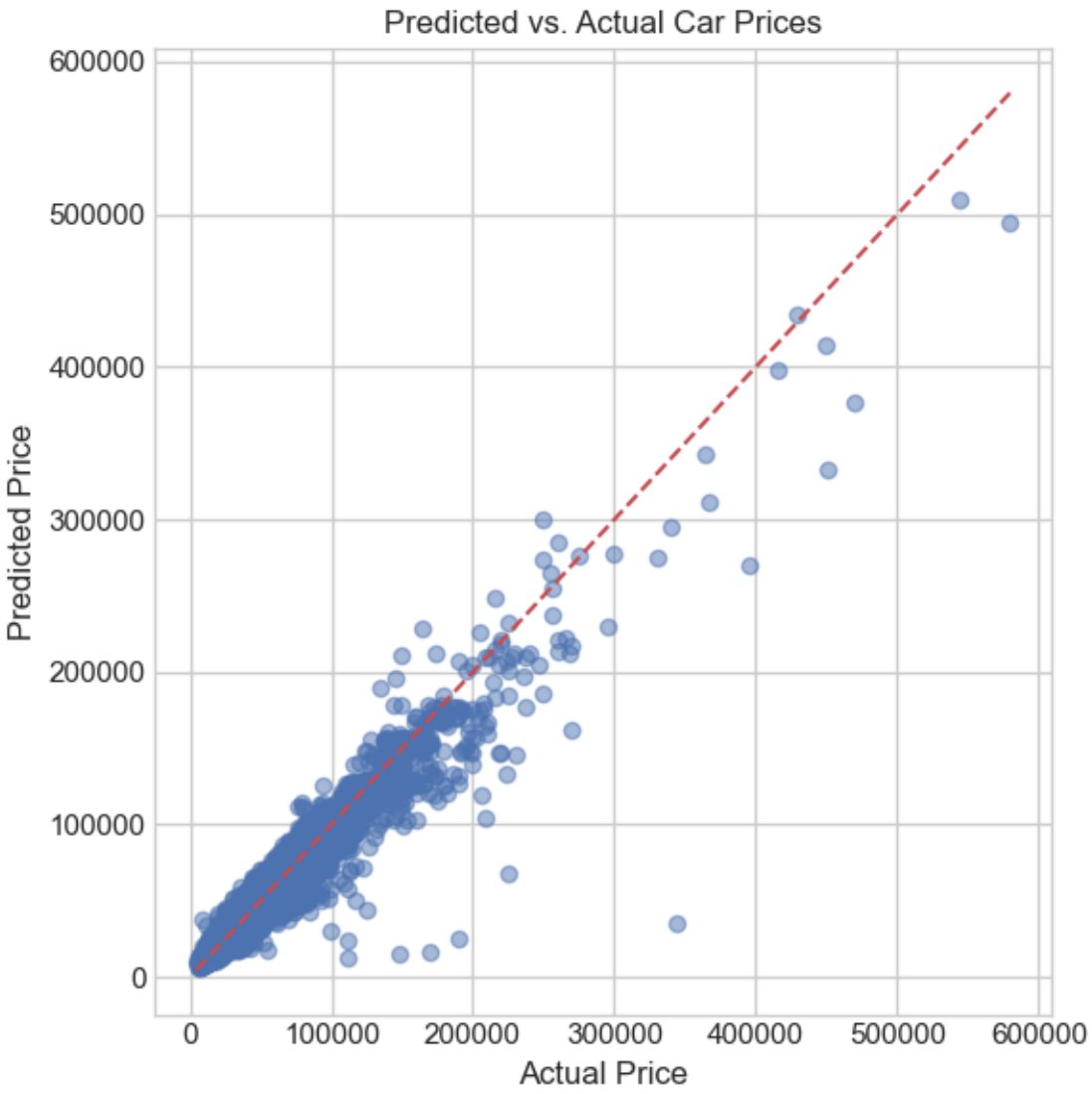
[137]:
```
modGB2.fit(X_train, y_train_log)
```

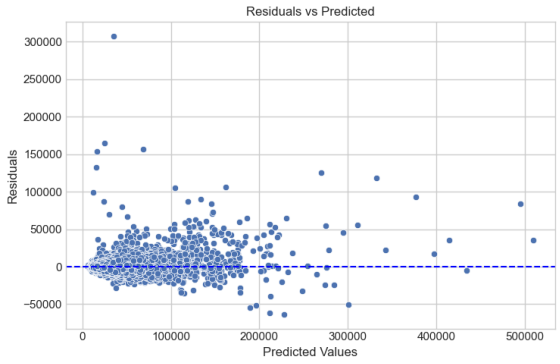
[137]:
```
GradientBoostingRegressor(learning_rate=0.05, max_depth=7, max_features='sqrt',
                          min_samples_leaf=5, min_samples_split=10,
                          n_estimators=200, random_state=56)
```

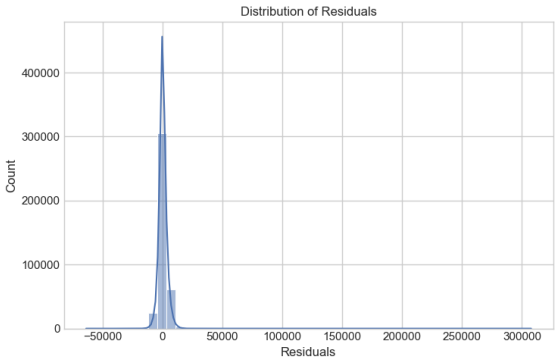
[138]:
```
evaluate_model(modGB2, X_train, X_test, y_train_log, y_test_log, features)
```

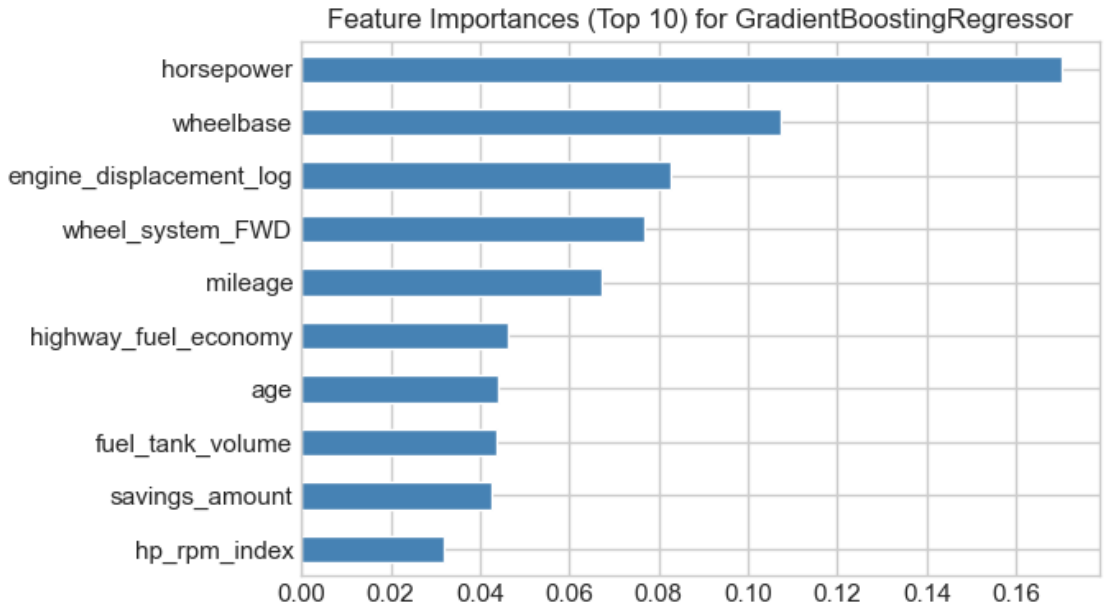
```
Model: GradientBoostingRegressor
MAE: 2457.6326299967836
MSE: 12628947.434802778
RMSE: 3553.7230385615
R².train: 0.9457301704164425
R²: 0.9456690558764538
R².adj: 0.9456618773877266
```




Predicted vs. Actual Car Prices
Predicted Price
Actual Price
0
100000
200000
300000
400000
500000
600000


Residuals vs Predicted
Residuals
Predicted Values


Distribution of Residuals
Count
Residuals



[138]:
```
[2457.6326299967836,
 12628947.434802778,
 3553.7230385615,
 0.9457301704164425,
 0.9456690558764538,
 0.9456618773877266]
```

[139]:
```
# Add to the comparision df
comparison_df['GradientBoost'] = [2457.6326299967836,
 12628947.434802778,
 3553.7230385615,
 0.9457301704164425,
 0.9456690558764538,
 0.9456618773877266]

comparison_df
```

[139]:

Metric	Linear	Ridge	Lasso	DecisionTree	RandomForest	\
MAE	4453.58	4453.58	4493.68	1874.65	1599.62	
MSE	49467985.92	49468057.07	51372044.33	8832659.61	6349831.40	
RMSE	7033.35	7033.35	7167.43	2971.98	2519.89	
R^2.train	0.79	0.79	0.79	0.98	0.99	
R^2	0.79	0.79	0.78	0.96	0.97	
R^2.adj	0.79	0.79	0.78	0.96	0.97	

Metric	GradientBoost
MAE	2457.63
MSE	12628947.43
RMSE	3553.72
R^2.train	0.95
R^2	0.95
R^2.adj	0.95

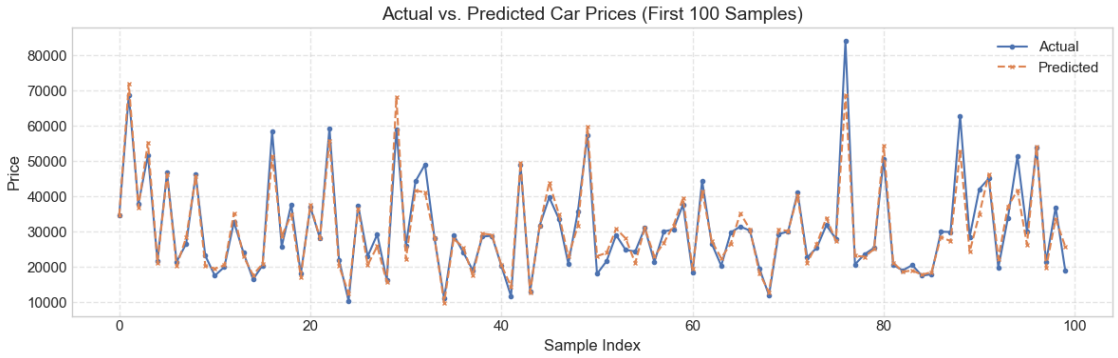
6.7 Extreme Gradient Boosting Regressor

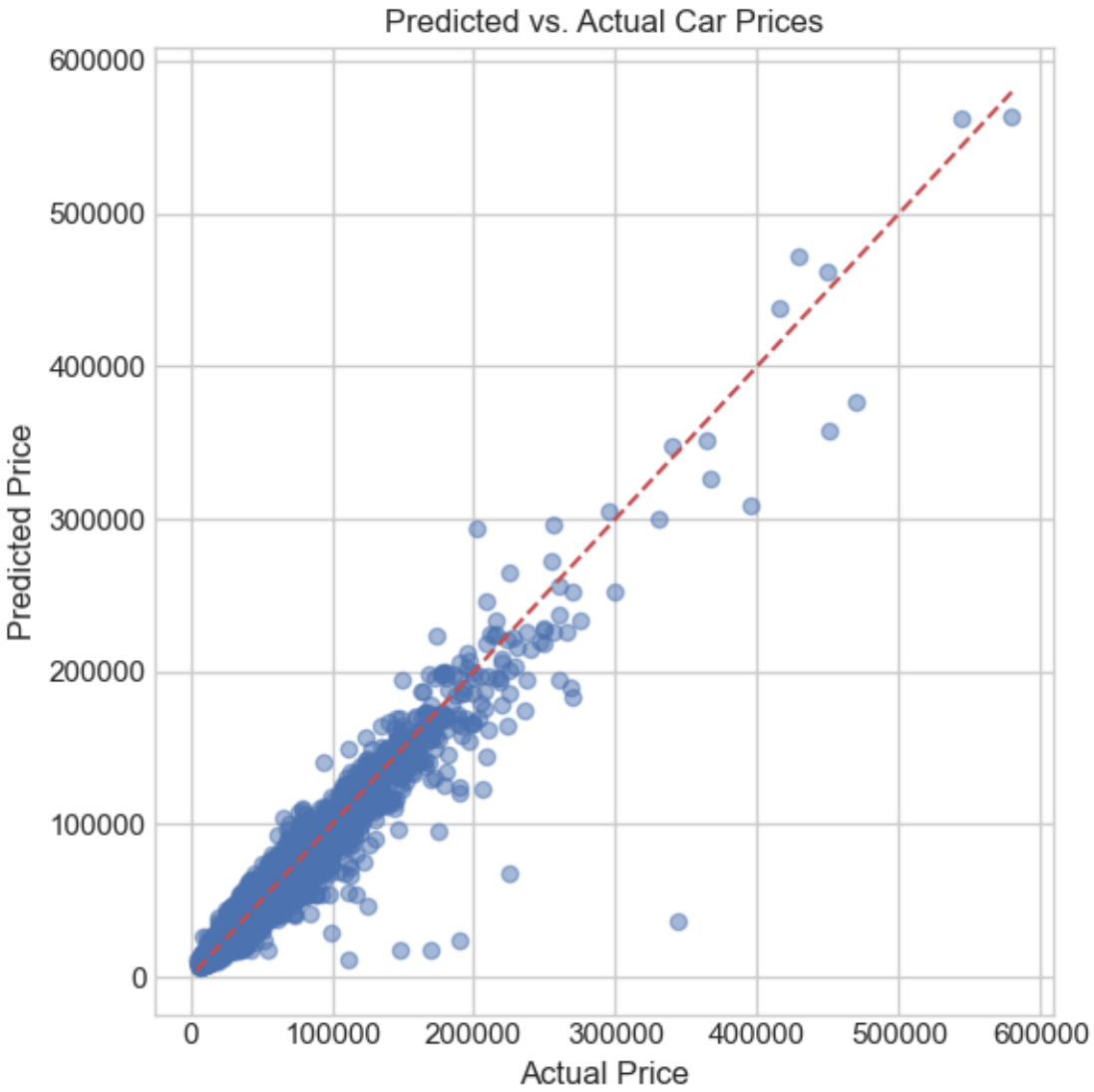
[140]:
```
from xgboost import XGBRegressor

# Initialize with default parameters
modXGB = XGBRegressor(random_state=56)

# Fit model
modXGB.fit(X_train, y_train_log)
```

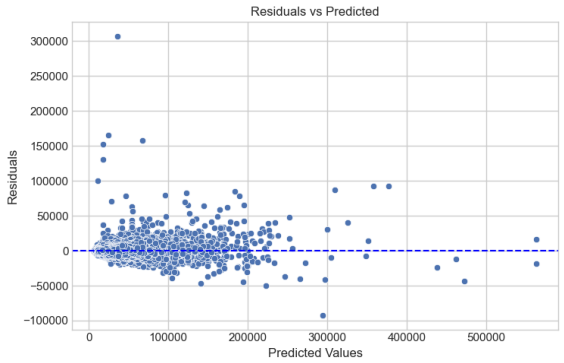
[140]:
```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             feature_weights=None, gamma=None, grow_policy=None,
             importance_type=None, interaction_constraints=None,
             learning_rate=None, max_bin=None, max_cat_threshold=None,
             max_cat_to_onehot=None, max_delta_step=None, max_depth=None,
             max_leaves=None, min_child_weight=None, missing=nan,
             monotone_constraints=None, multi_strategy=None, n_estimators=None,
             n_jobs=None, num_parallel_tree=None, …)
```

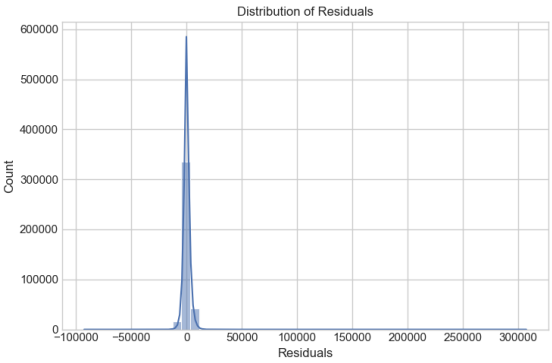
[141]:
```
evaluate_model(modXGB, X_train, X_test, y_train_log, y_test_log, features)
```

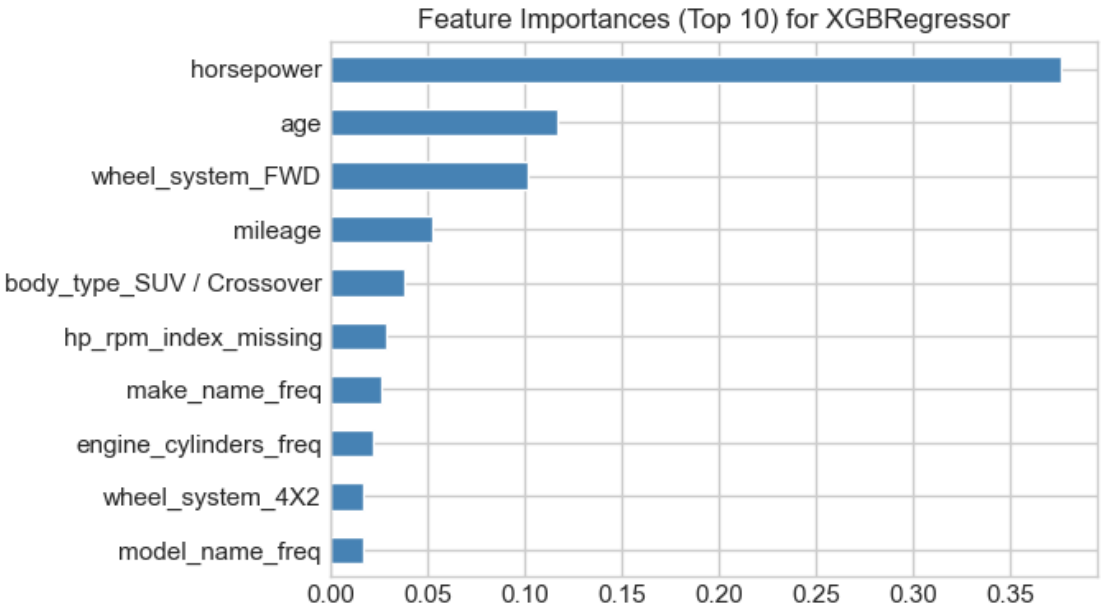
```
Model: XGBRegressor
MAE: 2107.621276314668
MSE: 9501192.546158735
RMSE: 3082.400451946297
R².train: 0.9594069981538065
R²: 0.9591249576421675
R².adj: 0.9591195570176151
```


Actual vs. Predicted Car Prices (First 100 Samples)
Actual
Predicted
Price
Sample Index
80000
70000
60000
50000
40000
30000
20000
10000
0
20
40
60
80
100


Predicted vs. Actual Car Prices
Predicted Price
Actual Price
600000
500000
400000
300000
200000
100000
0
0
100000
200000
300000
400000
500000
600000







[141]:
```
[2107.621276314668,
 9501192.546158735,
 3082.400451946297,
 0.9594069981538065,
 0.9591249576421675,
 0.9591195570176151]
```

6.7.1 Cross Validation and Hyperparameter Tuning

[142]:
```
# Define parameter grid
param_dist = {
    'n_estimators': [100, 300, 500],
    'learning_rate': [0.01, 0.05, 0.1, 0.2],
```

```
    'max_depth': [3, 5, 7, 10],
    'min_child_weight': [1, 3, 5],
    'subsample': [0.6, 0.8, 1.0],
    'colsample_bytree': [0.6, 0.8, 1.0],
    'gamma': [0, 0.1, 0.2, 0.5],
    'reg_alpha': [0, 0.1, 1],
    'reg_lambda': [1, 1.5, 2]
}

# Initialize XGBRegressor
xgb = XGBRegressor(
    objective='reg:squarederror',  # for regression
    random_state=56,
    n_jobs=2,
)

# Initialize RandomizedSearchCV
random_search = RandomizedSearchCV(
    estimator=xgb,
    param_distributions=param_dist,
    n_iter=30,                # number of parameter settings to sample
    scoring='neg_root_mean_squared_error',  # or 'neg_mean_squared_error'
    cv=3,                     # 5-fold cross-validation
    verbose=1,
    random_state=56,
    n_jobs=2,
)

# Run tuning (this is our subsampled data)
random_search.fit(X_sub, y_sub_log)
```

```
Fitting 3 folds for each of 30 candidates, totalling 90 fits
```

```
/opt/anaconda3/lib/python3.12/site-
packages/joblib/externals/loky/process_executor.py:752: UserWarning: A worker
stopped while some jobs were given to the executor. This can be caused by a too
short worker timeout or by a memory leak.
  warnings.warn(
```

[142]:
```
RandomizedSearchCV(cv=3,
                   estimator=XGBRegressor(base_score=None, booster=None,
                                          callbacks=None,
                                          colsample_bylevel=None,
                                          colsample_bynode=None,
                                          colsample_bytree=None, device=None,
                                          early_stopping_rounds=None,
                                          enable_categorical=False,
                                          eval_metric=None, feature_types=None,
```

```
                                    feature_weights=None, gamma=None,
                                    grow_policy=None,
                                    importance_type=None,
                                    interaction_constraint…
                                    num_parallel_tree=None, …),
                   n_iter=30, n_jobs=2,
                   param_distributions={'colsample_bytree': [0.6, 0.8, 1.0],
                                        'gamma': [0, 0.1, 0.2, 0.5],
                                        'learning_rate': [0.01, 0.05, 0.1, 0.2],
                                        'max_depth': [3, 5, 7, 10],
                                        'min_child_weight': [1, 3, 5],
                                        'n_estimators': [100, 300, 500],
                                        'reg_alpha': [0, 0.1, 1],
                                        'reg_lambda': [1, 1.5, 2],
                                        'subsample': [0.6, 0.8, 1.0]},
                   random_state=56, scoring='neg_root_mean_squared_error',
                   verbose=1)
```

[143]:
```
# Get best parameters
random_search.best_params_
```

[143]:
```
{'subsample': 1.0,
 'reg_lambda': 1,
 'reg_alpha': 0.1,
 'n_estimators': 500,
 'min_child_weight': 1,
 'max_depth': 10,
 'learning_rate': 0.05,
 'gamma': 0,
 'colsample_bytree': 0.6}
```

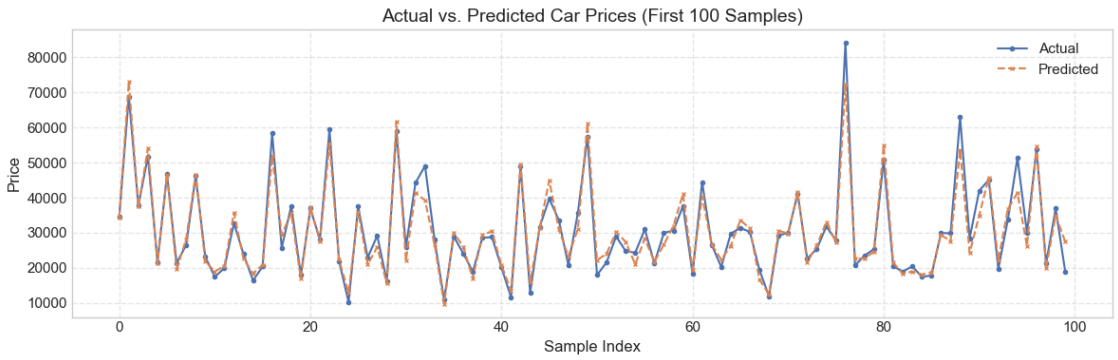
[144]:
```
# Initialize and train the Random Forest
modXGB2 = XGBRegressor(n_estimators=random_search.best_params_['n_estimators'],
                                    subsample=random_search.
 ↪best_params_['subsample'],
                                    reg_lambda=random_search.
 ↪best_params_['reg_lambda'],
                                    reg_alpha= random_search.
 ↪best_params_['reg_alpha'],
                                    min_child_weight= random_search.
 ↪best_params_['min_child_weight'],
                         learning_rate= random_search.
 ↪best_params_['learning_rate'],
                        gamma= random_search.best_params_['gamma'],
                        colsample_bytree = random_search.
 ↪best_params_['colsample_bytree'],
                                    random_state=56)
```

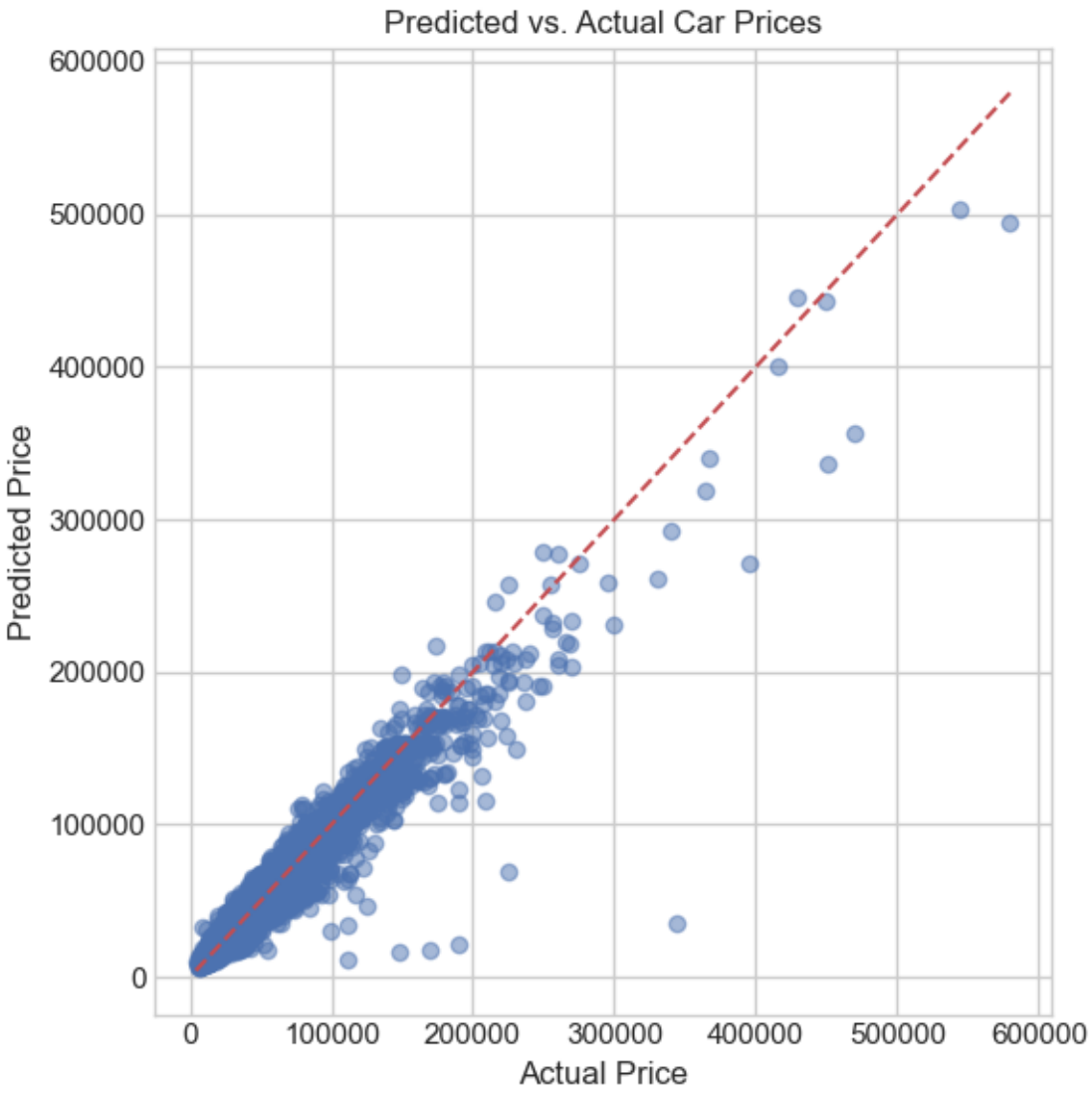
```
# Fit model
modXGB2.fit(X_train, y_train_log)
```

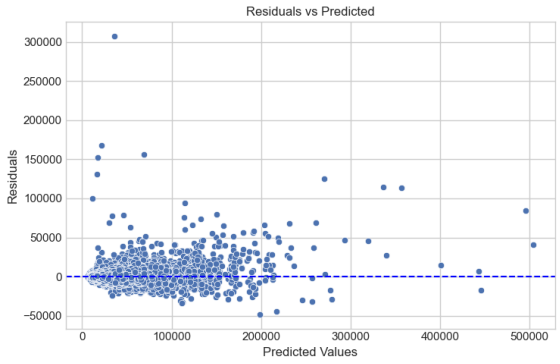
[144]:
```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=0.6, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             feature_weights=None, gamma=0, grow_policy=None,
             importance_type=None, interaction_constraints=None,
             learning_rate=0.05, max_bin=None, max_cat_threshold=None,
             max_cat_to_onehot=None, max_delta_step=None, max_depth=None,
             max_leaves=None, min_child_weight=1, missing=nan,
             monotone_constraints=None, multi_strategy=None, n_estimators=500,
             n_jobs=None, num_parallel_tree=None, …)
```

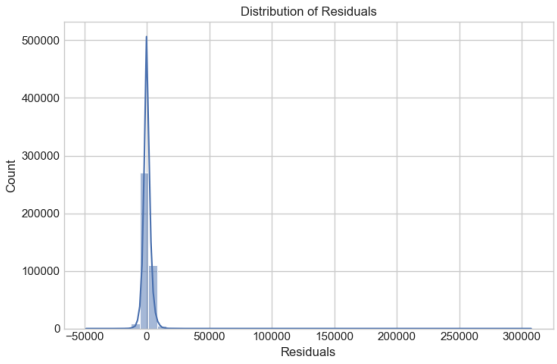
[145]:
```
evaluate_model(modXGB2, X_train, X_test, y_train_log, y_test_log, features)
```

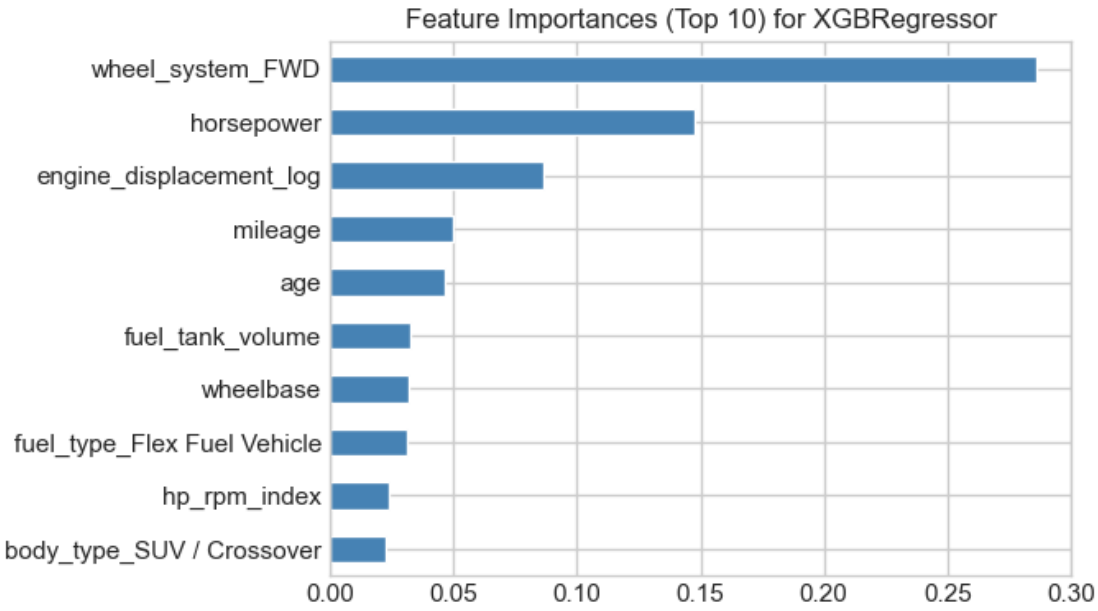
```
Model: XGBRegressor
MAE: 2158.032189755622
MSE: 10028916.334456243
RMSE: 3166.8464336712386
R².train: 0.9569815016003383
R²: 0.9568546392484394
R².adj: 0.9568489386575371
```




Predicted vs. Actual Car Prices
Predicted Price
Actual Price
0
100000
200000
300000
400000
500000
600000


Residuals vs Predicted
Residuals
Predicted Values


Distribution of Residuals
Count
Residuals



[145]:
```
[2158.032189755622,
 10028916.334456243,
 3166.8464336712386,
 0.9569815016003383,
 0.9568546392484394,
 0.9568489386575371]
```

[146]:
```
# Add to the comparision df
comparison_df['XGBoost'] = [2158.032189755622,
 10028916.334456243,
 3166.8464336712386,
 0.9569815016003383,
 0.9568546392484394,
 0.9568489386575371]

comparison_df
```

[146]:

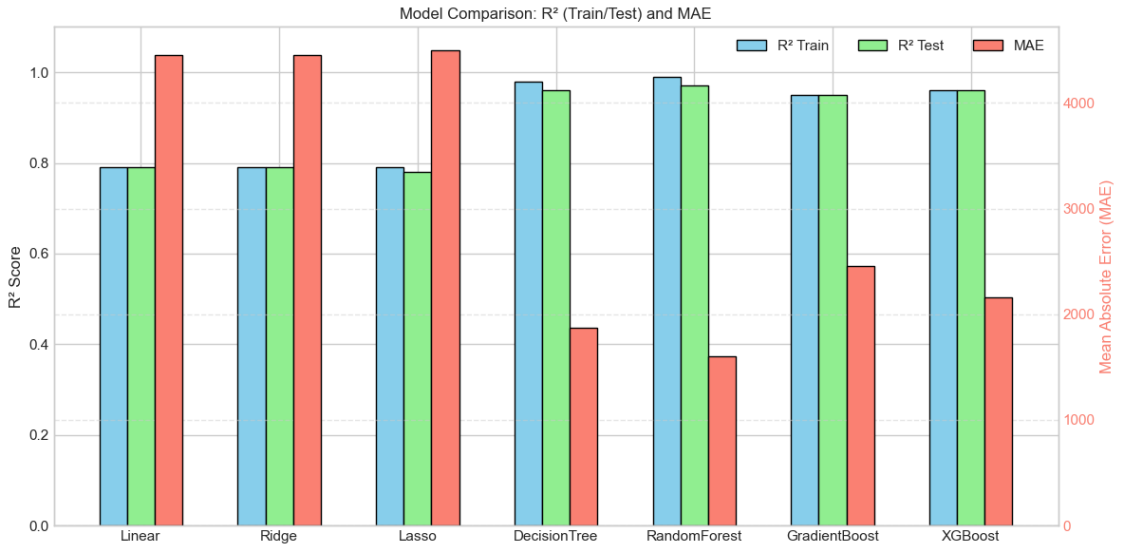
Metric	Linear	Ridge	Lasso	DecisionTree	RandomForest	\
MAE	4453.58	4453.58	4493.68	1874.65	1599.62	
MSE	49467985.92	49468057.07	51372044.33	8832659.61	6349831.40	
RMSE	7033.35	7033.35	7167.43	2971.98	2519.89	
R^2.train	0.79	0.79	0.79	0.98	0.99	
R^2	0.79	0.79	0.78	0.96	0.97	
R^2.adj	0.79	0.79	0.78	0.96	0.97	

Metric	GradientBoost	XGBoost
MAE	2457.63	2158.03
MSE	12628947.43	10028916.33
RMSE	3553.72	3166.85
R^2.train	0.95	0.96
R^2	0.95	0.96
R^2.adj	0.95	0.96

6.8 Plot R^2 and MAE Scores for Each Model

[147]:
```
# Create the comparison table data
# Data
data = {
    'Model': ['Linear', 'Ridge', 'Lasso', 'DecisionTree', 'RandomForest',␣
 ↪'GradientBoost', 'XGBoost'],
    'R²_train': [0.79, 0.79, 0.79, 0.98, 0.99, 0.95, 0.96],
    'R²_test': [0.79, 0.79, 0.78, 0.96, 0.97, 0.95, 0.96],
    'MAE': [4453.58, 4453.58, 4493.68, 1874.65, 1599.62, 2457.63, 2158.03]
}
df = pd.DataFrame(data)

# Setup
x = np.arange(len(df['Model']))
bar_width = 0.2

fig, ax1 = plt.subplots(figsize=(12, 6))

# Plot R² values on left Y-axis
bar1 = ax1.bar(x - bar_width, df['R²_train'], width=bar_width, label='R²␣
 ↪Train', color='skyblue', edgecolor='black')
bar2 = ax1.bar(x, df['R²_test'], width=bar_width, label='R² Test',␣
 ↪color='lightgreen', edgecolor='black')
ax1.set_ylabel('R² Score')
ax1.set_ylim(0, 1.1)

# Plot MAE on right Y-axis
ax2 = ax1.twinx()
bar3 = ax2.bar(x + bar_width, df['MAE'], width=bar_width, label='MAE',␣
 ↪color='salmon', edgecolor='black')
ax2.set_ylabel('Mean Absolute Error (MAE)', color='salmon')
ax2.tick_params(axis='y', labelcolor='salmon')

# Common x-axis and labels
plt.xticks(x, df['Model'], rotation=45)
ax1.set_title('Model Comparison: R² (Train/Test) and MAE')
```

```
# Combine legends - fixed approach
handles1, labels1 = ax1.get_legend_handles_labels()
handles2, labels2 = ax2.get_legend_handles_labels()
ax1.legend(handles1 + handles2, labels1 + labels2, loc='upper right', ncol=3)

plt.tight_layout()
plt.grid(axis='y', linestyle='--', alpha=0.5)
plt.show()
```



6.8.1 Which ML model did I consider for final prediction model and why?

After analyzing the evaluation metrics, I am concluding that Random Forest Regressor is the best model to use moving forward. The Random Forest model is consistently out-performing the other models over the different error metrics (MAE, MSE and RMSE). However, we can see, there is a bit of overfitting since the training R^2 value is 0.99 which is larger that the testing R^2 of 0.97, but I think the gap between them is reasonable and acceptable which shows that the model generalizes very well. Additionally, while models such as XGBoost do have less overfitting (less of a gap between the training and test R^2 values), Random forest has added benefits in terms of less complexity, better interpretability, and less extensive hyperparameter tuning. If interpretability or resource use is very important, I may want to consider XGBoost or Gradient Boosting with SHAP values for better explainability. However for just the simplest justification, I prefer Random Forest the most.

However, it is evident that these models show increasing spread of residuals for higher priced vehicles, easily seen in the actual vs predicted plot. This is evidence of heteroscedasticity that suggests that the model does not perform consistently well throughout the entire range of values. Greater RMSE compared to MAE suggests that the car with higher price will have a greater penalty, possibly due to there being a small subset of cars with high prices. I didn't remove them as they appear to be vaild price for luxury vehicles. These data are extreme outliers with large residuals in both positive and negative direction, which suggest issue in their accurate prediction. To better

deal with some of these issues the price data could be stratified.

6.8.2 Save the Best Performing ML Model

[148]:
```
# export the chosen model using joblib or pickle
import joblib

# Save the best model (e.g., XGB2)
joblib.dump(modRF2, "randomForest_model.joblib")

# Later: load it back
# loaded_model = joblib.load("xgboost_model.joblib")
```

[148]:
```
['randomForest_model.joblib']
```

6.8.3 Predict Using Unseen Data for a Sanity Check.

[149]:
```
# Load the saved model
loaded_rf_model = joblib.load("randomForest_model.joblib")

# Make predictions on test data
new_test_preds = loaded_rf_model.predict(X_test)


# Sanity check: Are predictions in a reasonable range?
print(np.expm1(new_test_preds[:5])) #since our price is in log scale.

# Sanity Check - Evaluate performance
mse = mean_squared_error(y_test_log, new_test_preds)
mae = mean_absolute_error(y_test_log, new_test_preds)
rmse = root_mean_squared_error(y_test_log, new_test_preds)
r2 = r2_score(y_test_log, new_test_preds)

print("MSE:", mse)
print("MAE:", mae)
print("RMSE:", rmse)
print("R² Score:", r2)
```

```
[33634.61640796 71594.33283789 36740.90194193 53822.18561394
 21345.00029153]
MSE: 0.004911006131001403
MAE: 0.05044936864055616
RMSE: 0.07007857112556878
R² Score: 0.9721976089795049
```

A sanity check is a necessary part of the machine learning deployment pipeline because it checks whether the saved model works as expected when run outside of the training environment. This step is similar to starting the engine after servicing a car; you don't just want to verify the engine operates, you want to confirm it operates correctly, and demonstrates the behavior your training

phase would have anticipated. Thus, when checked, my model generalizes effectively to unseen data presenting the same performance it was demonstrating on the training and validation datasets. With an R^2 value of 0.97 and error rates being negligible, the model is accurate and reliable, and demonstrates a legitimate potential for deployment.