ECON 626 Prediction Competition 4 Code

Objective: Utilize regression algorithms (linear regression, LASSO, Ridge, Subset Selection) to train a model that predicts the natural logarithm of car price.

Importing Librarys

In [ ]:
```
import numpy as np
import pandas as pd
import re
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib.pyplot import subplots
import statsmodels.api as sm
from sklearn import preprocessing
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import linear_model # Linear regression
from sklearn.metrics import mean_absolute_percentage_error, r2_score, mean_s
from sklearn.preprocessing import LabelEncoder
```

Importing data

In [ ]:
```
small_data_path = "/Users/andrew/Downloads/UW courses/ECON 626/Prediction Co
small_df= pd.read_csv(small_data_path)
#create a dataframe for our smaller dataset

large_data_path = "/Users/andrew/Downloads/UW courses/ECON 626/Prediction Co
large_df = pd.read_csv(large_data_path)
#create a dataframe for our larger dataset

test_data_path = "/Users/andrew/Downloads/UW courses/ECON 626/Prediction Com
test_df = pd.read_csv(test_data_path)
#create a dataframe for our larger dataset

total_df  = pd.concat([small_df, large_df], axis = 0)
#create a dataframe containing both small and large df
```

Inspect data

In [ ]:
```
#function:
def inspect_dataset(dataset):
    # Print the head of the dataset
    print("Head of the dataset:")
    print(dataset.head())
```

```
    print("\n")

    # Print the info of the dataset
    print("Info of the dataset:")
    print(dataset.info())
    print("\n")

    # Print the shape of the dataset
    print("Shape of the dataset:")
    print(dataset.shape)
    print("\n")

    # Print value counts for columns of type object
    object_columns = dataset.select_dtypes(include=['object']).columns
    for column in object_columns:
        print(f"Value counts for column '{column}':")
        print(dataset[column].value_counts())
        print("\n")
```

In [ ]:
```
small_df.head()
```

Out[ ]:

	price	back_legroom	body_type	engine_displacement	exterior_color	fuel_type
0	18495.0	44.5 in	Pickup Truck	5700	MAROON	Gasoline
1	16422.0	41.4 in	Sedan	1800	Black Sand Pearl	Gasoline
2	39935.0	36.5 in	Sedan	2000	JET BLACK	Gasoline
3	23949.0	38.7 in	SUV / Crossover	3500	Brilliant Silver	Gasoline
4	37545.0	35.2 in	Sedan	2000	Black	Gasoline

In [ ]:
```
small_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 18 columns):
 #   Column               Non-Null Count   Dtype  
---  ------               --------------   -----  
 0   price                100000 non-null  float64
 1   back_legroom         100000 non-null  object 
 2   body_type            100000 non-null  object 
 3   engine_displacement  100000 non-null  int64  
 4   exterior_color       97934 non-null   object 
 5   fuel_type            100000 non-null  object 
 6   height               100000 non-null  object 
 7   highway_fuel_economy 100000 non-null  int64  
 8   horsepower           100000 non-null  int64  
 9   latitude             100000 non-null  float64
 10  length               100000 non-null  object 
 11  listed_date          100000 non-null  object 
 12  longitude            100000 non-null  float64
 13  mileage              100000 non-null  int64  
 14  wheel_system         100000 non-null  object 
 15  wheelbase            100000 non-null  object 
 16  width                100000 non-null  object 
 17  year                 100000 non-null  int64  
dtypes: float64(3), int64(5), object(10)
memory usage: 13.7+ MB
```

In [ ]:
```
small_df.isna().sum()
```

Out[ ]:
```
price                     0
back_legroom              0
body_type                 0
engine_displacement       0
exterior_color         2066
fuel_type                 0
height                    0
highway_fuel_economy      0
horsepower                0
latitude                  0
length                    0
listed_date               0
longitude                 0
mileage                   0
wheel_system              0
wheelbase                 0
width                     0
year                      0
dtype: int64
```

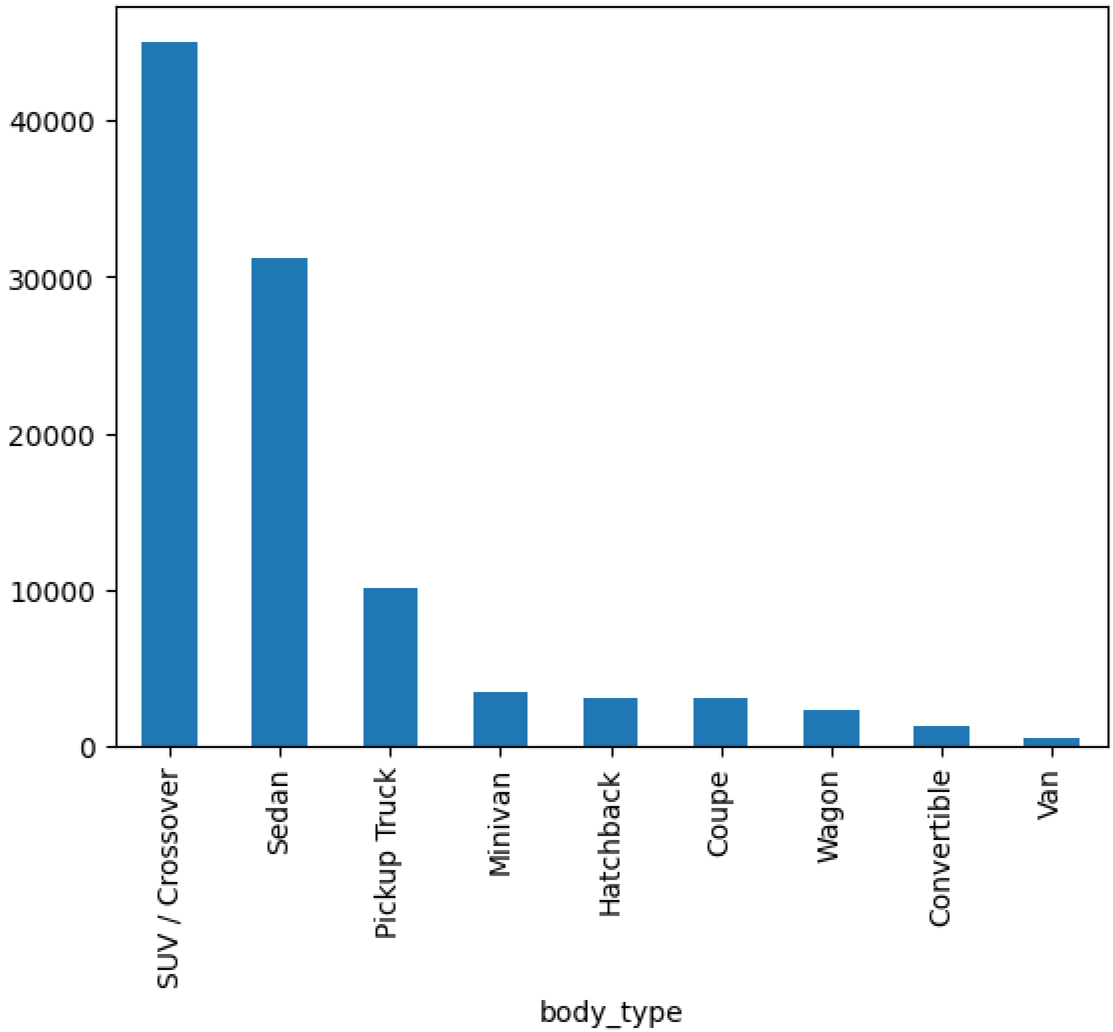
In [ ]:
```
small_df.shape
```

Out[ ]:
```
(100000, 18)
```

Data Visualizations

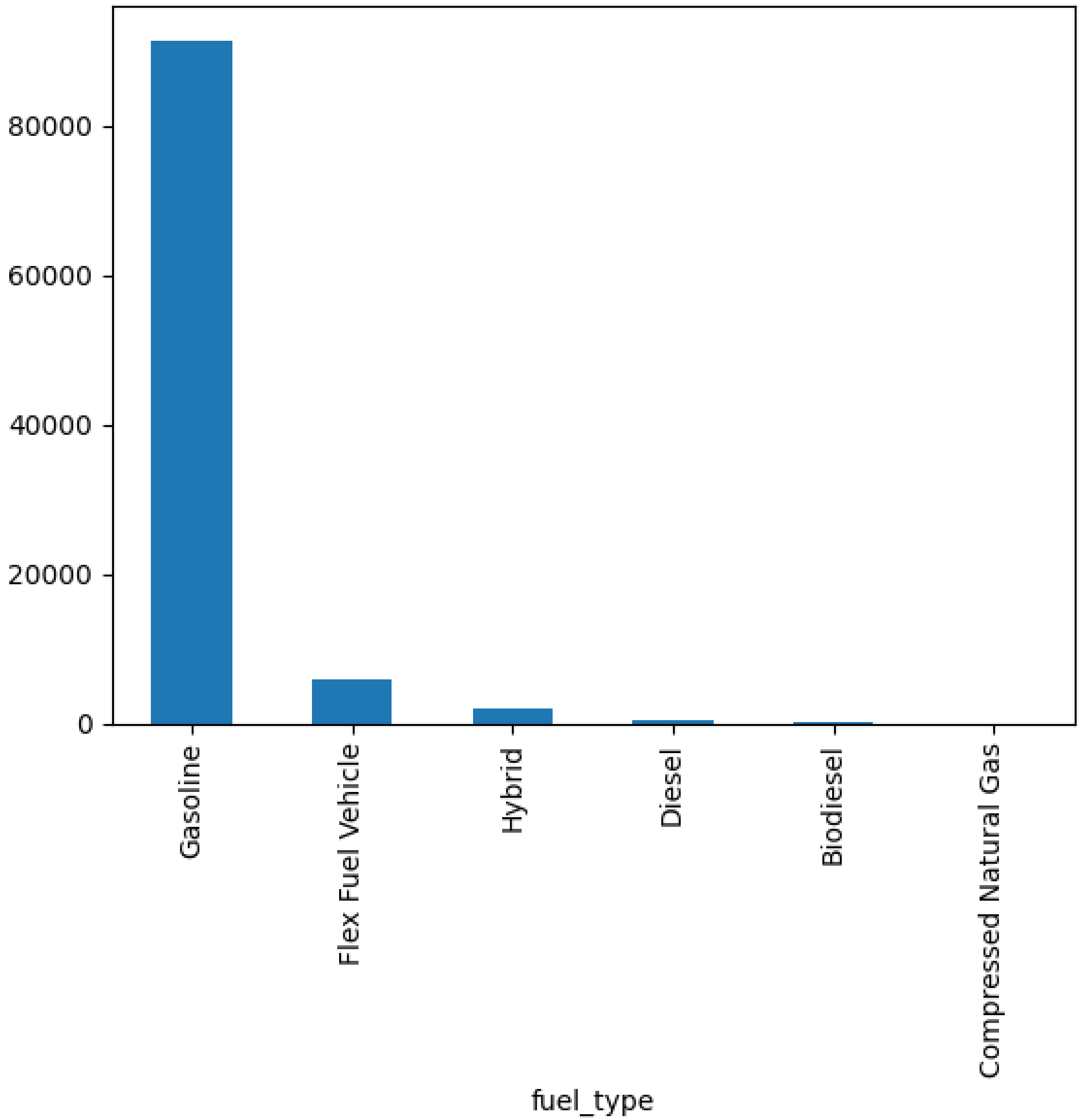
In [ ]:
```
small_df['body_type'].value_counts().plot(kind='bar')
```

Out[ ]: <Axes: xlabel='body_type'>



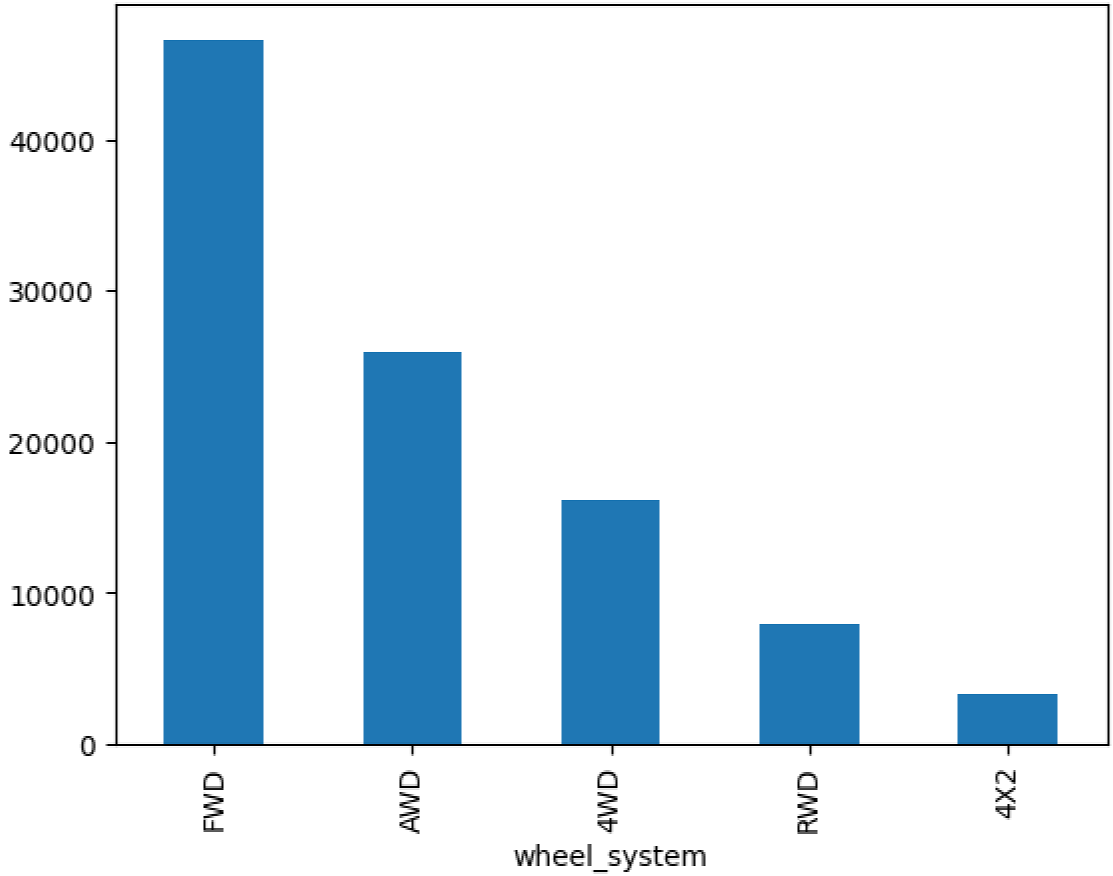
In [ ]:
```
small_df['fuel_type'].value_counts().plot(kind='bar')
```

Out[ ]: <Axes: xlabel='fuel_type'>



In [ ]:
```
small_df['wheel_system'].value_counts().plot(kind='bar')
```

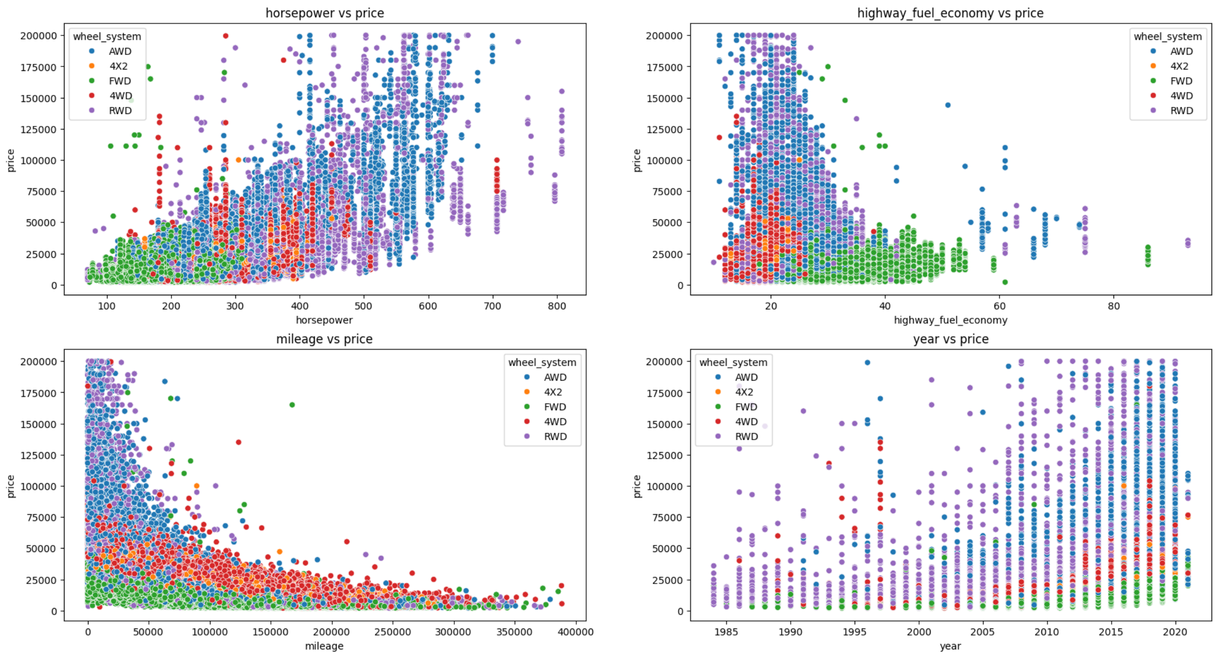
Out[ ]: <Axes: xlabel='wheel_system'>



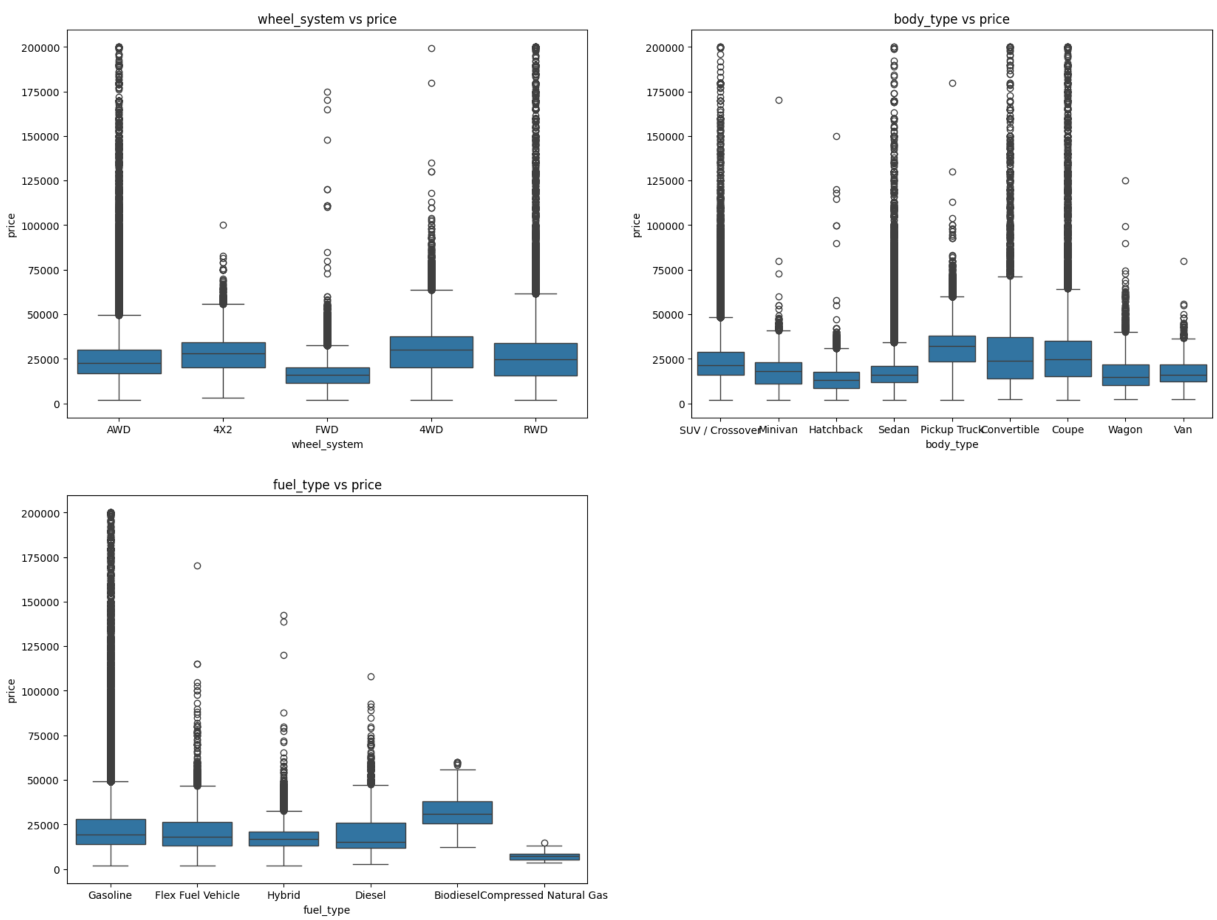
In [ ]:
```
numerical_columns = small_df.select_dtypes(include=np.number).columns.tolist

num_drop_list = 'latitude', 'price', 'longitude'

numerical_columns = list(set(numerical_columns) - set(num_drop_list))
```

In [ ]:
```
categorical_columns = small_df.select_dtypes(exclude=np.number).columns.toli
categorical_columns

cat_drop_list = 'back_legroom', 'height', 'length', 'listed_date', 'wheelbas

categorical_columns = list(set(categorical_columns) - set(cat_drop_list))
```

In [ ]:
```
plt.figure(figsize = (21, 11))
for i, value in enumerate(numerical_columns[:-1]):
    plt.subplot(2,2,i+1)
    plt.title( value + ' vs price')
    sns.scatterplot(data = large_df, x = value, y = 'price', hue = 'wheel_sy
```



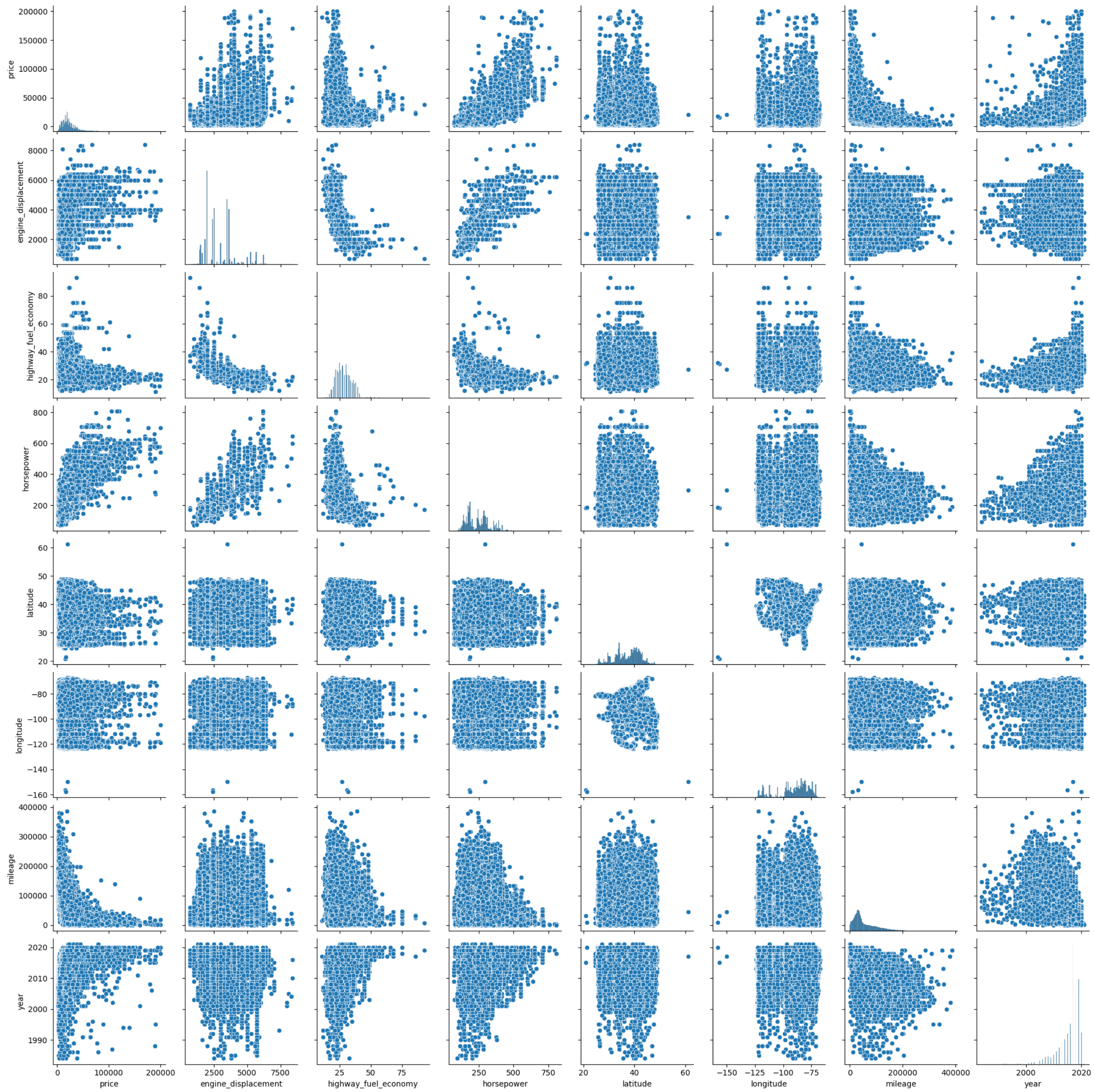
In [ ]:
```
plt.figure(figsize = (20, 15))
for i, value in enumerate(categorical_columns):
    plt.subplot(2,2,i+1)
    plt.title( value + ' vs price')
    sns.boxplot(data = large_df, x = value, y = 'price')
```



In [ ]:
```
numeric_df = small_df.select_dtypes(include='number')
```

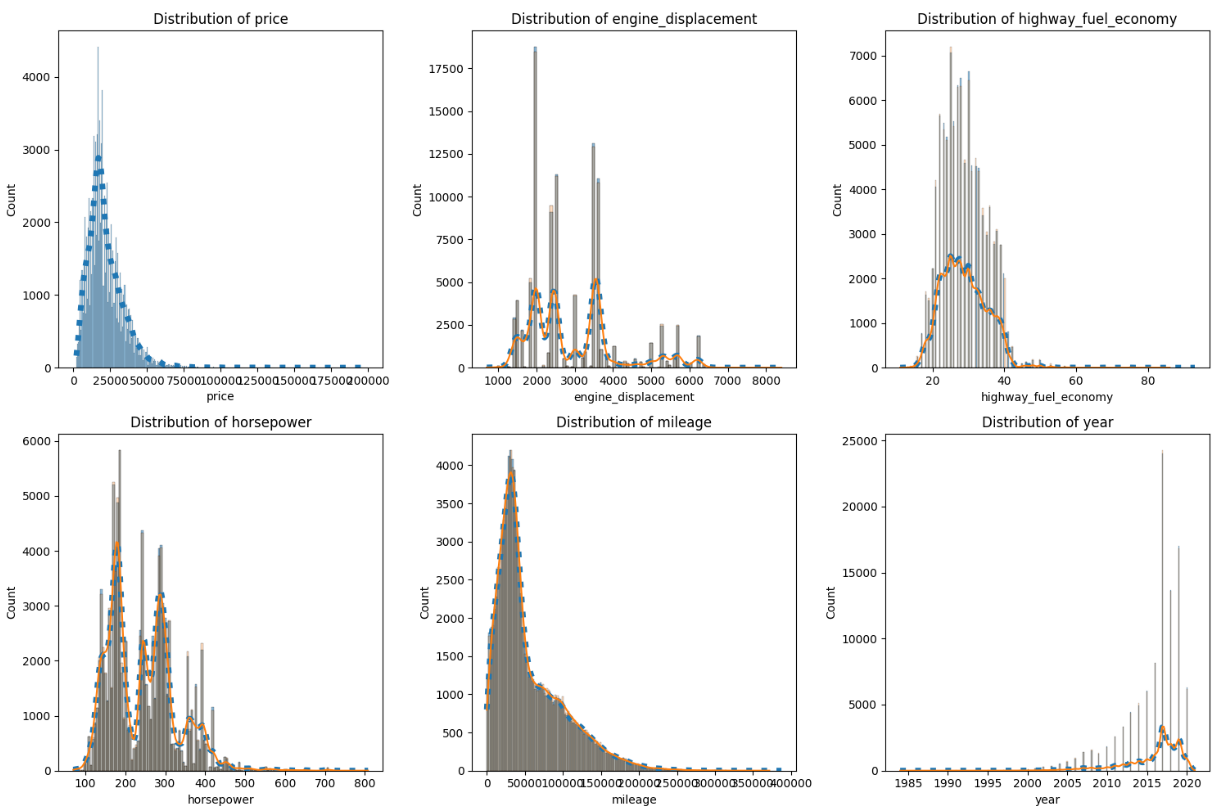
```
# Create pairplot
sns.pairplot(numeric_df)
```

Out[ ]: <seaborn.axisgrid.PairGrid at 0x290165c70>



In [ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns

og_cols = ['price', 'engine_displacement', 'highway_fuel_economy', 'horsepow
num_cols = len(og_cols)

# Calculate the number of rows and columns needed for subplots
num_rows = (num_cols + 2) // 3  # Ceiling division to ensure we have enough
num_cols = min(num_cols, 3)  # Limit the number of columns to 3

fig, axes = plt.subplots(num_rows, num_cols, figsize=(15, 10))  # Adjust fig

for i, col in enumerate(og_cols):
    row_idx = i // num_cols
    col_idx = i % num_cols
```

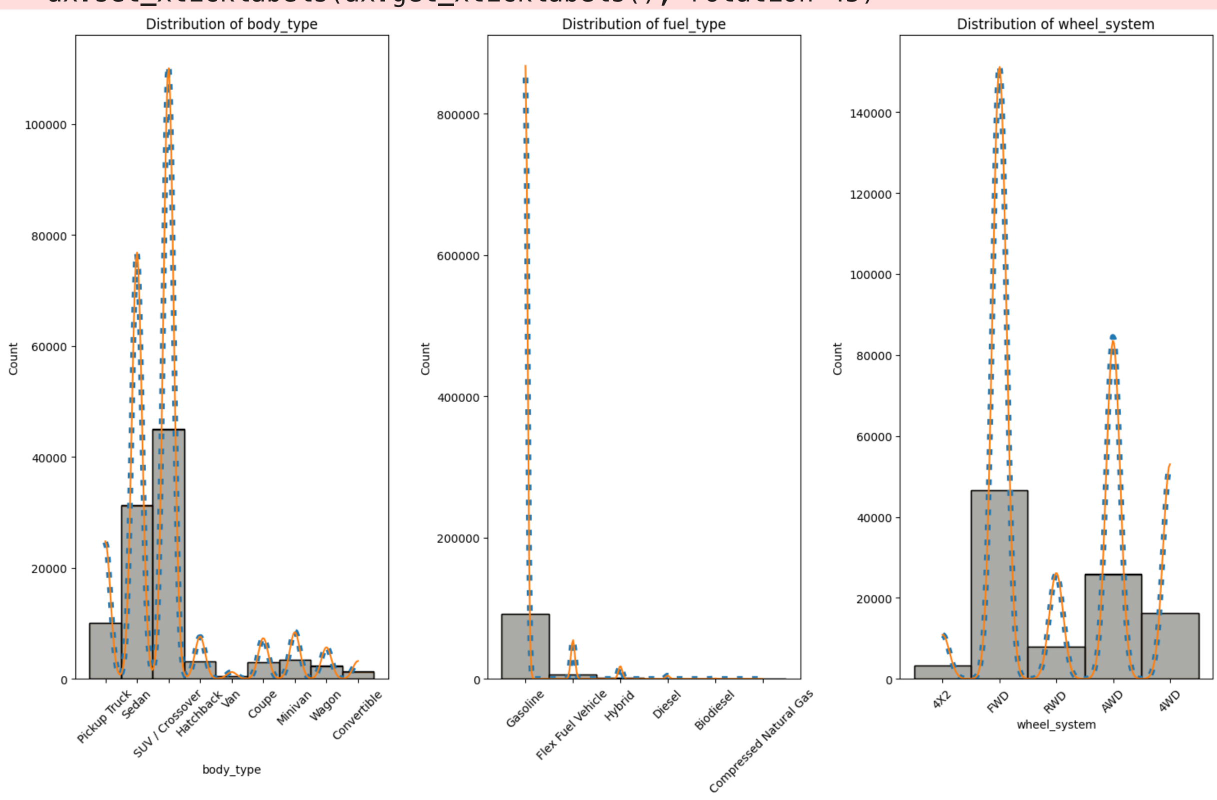
```
    if num_rows == 1:
        ax = axes[col_idx]
    else:
        ax = axes[row_idx, col_idx]

    sns.histplot(small_df[col], ax=ax, kde=True, line_kws={'lw': 5, 'ls': ':
    sns.histplot(test_df[col], ax=ax, kde=True, alpha=0.25)

    ax.set_title('Distribution of ' + col)

# Adjust layout to prevent overlap of subplots
plt.tight_layout()
plt.show()
```



In [ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns

og_cat_cols = ['body_type', 'fuel_type', 'wheel_system']
num_cols = len(og_cat_cols)

# Calculate the number of rows and columns needed for subplots
num_rows = (num_cols + 2) // 3  # Ceiling division to ensure we have enough
num_cols = min(num_cols, 3)  # Limit the number of columns to 3

fig, axes = plt.subplots(num_rows, num_cols, figsize=(15, 10))  # Adjust fig

for i, col in enumerate(og_cat_cols):
    row_idx = i // num_cols
    col_idx = i % num_cols
```

```
    if num_rows == 1:
        ax = axes[col_idx]
    else:
        ax = axes[row_idx, col_idx]

    sns.histplot(small_df[col], ax=ax, kde=True, line_kws={'lw': 5, 'ls': ':
    sns.histplot(test_df[col], ax=ax, kde=True, alpha=0.25)

    ax.set_title('Distribution of ' + col)

    ax.set_xticklabels(ax.get_xticklabels(), rotation=45)
# Adjust layout to prevent overlap of subplots
plt.tight_layout()
plt.show()
```

```
/var/folders/g9/7gqbb_gn4tv717l8v7pyt28c0000gn/T/ipykernel_13898/2936335544.
py:27: UserWarning: set_ticklabels() should only be used with a fixed number
of ticks, i.e. after set_ticks() or using a FixedLocator.
  ax.set_xticklabels(ax.get_xticklabels(), rotation=45)
/var/folders/g9/7gqbb_gn4tv717l8v7pyt28c0000gn/T/ipykernel_13898/2936335544.
py:27: UserWarning: set_ticklabels() should only be used with a fixed number
of ticks, i.e. after set_ticks() or using a FixedLocator.
  ax.set_xticklabels(ax.get_xticklabels(), rotation=45)
/var/folders/g9/7gqbb_gn4tv717l8v7pyt28c0000gn/T/ipykernel_13898/2936335544.
py:27: UserWarning: set_ticklabels() should only be used with a fixed number
of ticks, i.e. after set_ticks() or using a FixedLocator.
  ax.set_xticklabels(ax.get_xticklabels(), rotation=45)
```



In [ ]:

```
small_df['listed_date'] = pd.to_datetime(small_df['listed_date'])
listed_year = small_df['listed_date'].dt.year
small_df['age_at_listing'] = listed_year - small_df['year']
```

In [ ]:
```
small_df
```

Out[ ]:

	price	back_legroom	body_type	engine_displacement	exterior_color	fuel_
0	18495.0	44.5 in	Pickup Truck	5700	MAROON	Ga:
1	16422.0	41.4 in	Sedan	1800	Black Sand Pearl	Ga:
2	39935.0	36.5 in	Sedan	2000	JET BLACK	Ga:
3	23949.0	38.7 in	SUV / Crossover	3500	Brilliant Silver	Ga:
4	37545.0	35.2 in	Sedan	2000	Black	Ga:
...	...	...	...	...	...	
99995	41008.0	39.7 in	SUV / Crossover	5300	Green	Ga:
99996	13933.0	35.1 in	Sedan	2000	Alpine White	Ga:
99997	5597.0	36.2 in	Hatchback	2400	Silver	Ga:
99998	8942.0	35.6 in	SUV / Crossover	2500	White	Ga:
99999	10990.0	34.5 in	Hatchback	1500	Blue	Ga:

100000 rows × 19 columns

Data Preprocessing

In [ ]:
```
def extract_numeric_value(text):
    # Use regular expression to extract numeric value
    match = re.match(r'(\d+)', str(text))
    if match:
        return int(match.group(1))
    else:
        return None
```

In [ ]:
```
#Testing
# small_df['length'] = pd.to_numeric(small_df['length'].str.replace(' in', '
# small_df['length']

small_df['length'] = pd.to_numeric(small_df['length'].astype(str).str.replac
small_df['width'] = pd.to_numeric(small_df['width'].astype(str).str.replace(
```

```
small_df['height'] = pd.to_numeric(small_df['height'].astype(str).str.replac

small_df.isna().sum()

#small_df["car_vol"] = small_df["length"] * small_df["width"]* small_df["hei
```

Out[ ]:

```
price                      0
back_legroom               0
body_type                  0
engine_displacement        0
exterior_color          2066
fuel_type                  0
height                    33
highway_fuel_economy       0
horsepower                 0
latitude                   0
length                    33
listed_date                0
longitude                  0
mileage                    0
wheel_system               0
wheelbase                  0
width                     33
year                       0
age_at_listing             0
dtype: int64
```

In [ ]:

In [ ]:

```
small_df[small_df['length'].isnull()].head()
```

Out[ ]:

	price	back_legroom	body_type	engine_displacement	exterior_color	fuel_
1232	20981.0	--	SUV / Crossover	2000	Mineral Silver	Gas
5010	21976.0	--	SUV / Crossover	2000	Black Cherry	Gas
6256	48995.0	--	Sedan	2900	Trofeo White Tri-Coat	Gas
6600	23988.0	--	SUV / Crossover	2000	Mineral Silver	Gas
11525	24655.0	--	SUV / Crossover	2000	Mineral Silver	Gas

In [ ]:

```
#Groups the lengths by catagorys then fills NA with mean from that catagory
small_df['length'] = small_df['length'].fillna(small_df.groupby('body_type')
small_df['width'] = small_df['width'].fillna(small_df.groupby('body_type')['
small_df['height'] = small_df['height'].fillna(small_df.groupby('body_type')
```

In [ ]:

```
small_df.isna().sum()
```

Out[ ]:
```
price                    0
back_legroom             0
body_type                0
engine_displacement      0
exterior_color        2066
fuel_type                0
height                   0
highway_fuel_economy     0
horsepower               0
latitude                 0
length                   0
listed_date              0
longitude                0
mileage                  0
wheel_system             0
wheelbase                0
width                    0
year                     0
age_at_listing           0
dtype: int64
```

In [ ]:
```
small_df['fuel_type'].value_counts()
```

Out[ ]:
```
fuel_type
Gasoline                  91538
Flex Fuel Vehicle          5902
Hybrid                     1963
Diesel                      507
Biodiesel                    87
Compressed Natural Gas        3
Name: count, dtype: int64
```

In [ ]:
```
#Creating a function to preprocess the data
def prep_data(dataset):

    dataset_cols = dataset.columns
    if 'price' in dataset_cols:
        dataset['log_price'] = np.log(dataset['price'])
    else:
        pass

    #Feature Engineering
    dataset['length'] = pd.to_numeric(dataset['length'].astype(str).str.repl
    dataset['length'] = dataset['length'].fillna(dataset.groupby('body_type'

    dataset['width'] = pd.to_numeric(dataset['width'].astype(str).str.replac
    dataset['width'] = dataset['width'].fillna(dataset.groupby('body_type')[

    dataset['height'] = pd.to_numeric(dataset['height'].astype(str).str.repl
    dataset['height'] = dataset['height'].fillna(dataset.groupby('body_type'

    # dataset['wheelbase'] = pd.to_numeric(dataset['wheelbase'].astype(str).
    # dataset['wheelbase'] = dataset['wheelbase'].fillna(dataset.groupby('bo

    dataset["car_vol"] = dataset["length"] * dataset["width"]* dataset["heig
```

```
    dataset['listed_date'] = pd.to_datetime(dataset['listed_date'])
    listed_year = dataset['listed_date'].dt.year
    dataset['age_at_listing'] = listed_year - dataset['year']

    # fuel_type_map = {'Gasoline': 1, 'Diesel': 0, 'Flex Fuel Vehicle' : 0,

    # # Apply the mapping to the 'fuel_type' column
    # dataset['fuel_type_binary'] = dataset['fuel_type'].map(fuel_type_map)

    drop = ['price', 'back_legroom', 'wheelbase', 'latitude', 'longitude', '
    for col in drop:
        dataset = dataset.drop([col], axis=1)

    col_encode = [ 'body_type', 'fuel_type', 'wheel_system']
    le = LabelEncoder()
    for col in col_encode:
        new_col = col+'_enc'
        dataset[new_col] = le.fit_transform(dataset[col])
    dataset['litres'] = (dataset['engine_displacement']/1000).astype(float)
    dataset = dataset.drop(['engine_displacement'], axis=1)
    drop_enc= ['body_type', 'fuel_type', 'wheel_system']
    for col in drop_enc:
        dataset = dataset.drop([col], axis=1)
    return dataset
```

In [ ]:
```
train_df = prep_data(total_df)
```

In [ ]:
```
test_df = prep_data(test_df)
```

In [ ]:
```
train_df
```

Out[ ]:

	height	highway_fuel_economy	horsepower	length	mileage	width	year
0	75.6	18	381	228.7	167184	79.9	2008
1	57.3	38	132	182.6	29451	69.9	2016
2	58.2	34	248	194.6	14984	83.7	2019
3	67.8	28	260	192.8	15697	75.4	2020
4	56.3	33	255	184.5	6907	79.4	2020
...	...	...	...	...	...	...	...
499995	65.7	26	172	173.6	102204	69.1	2012
499996	64.8	29	168	171.9	34234	71.3	2018
499997	69.3	25	295	189.8	38511	84.8	2018
499998	48.9	29	455	176.9	9073	73.9	2016
499999	69.0	18	283	203.7	36328	88.5	2019

600000 rows × 14 columns

In [ ]:
```
test_df
```

Out[ ]:

	height	highway_fuel_economy	horsepower	length	mileage	width	year	lo
0	58.1	22	310	180.9	10265	79.5	2019	
1	57.3	36	132	183.1	35574	69.9	2017	
2	66.5	33	190	180.6	10885	73.0	2019	
3	58.0	40	188	191.8	2986	83.5	2019	
4	70.6	24	278	212.3	17085	75.2	2019	
...	...	...	...	...	...	...	...	
99995	69.1	22	201	173.0	173629	70.1	2002	
99996	70.7	25	310	204.3	17214	78.6	2018	
99997	67.1	29	176	183.5	33638	72.6	2017	
99998	77.7	21	395	228.9	51327	82.1	2019	
99999	73.9	22	355	230.0	25529	80.0	2018	

100000 rows × 14 columns

In [ ]:
```
#small_df['fuel_type_binary'].value_counts()
```

In [ ]:
```
drop_low_importance = ['age_at_listing', 'car_vol', 'width']

for col in drop_low_importance:
    train_df = train_df.drop([col], axis=1)
    test_df = test_df.drop([col], axis=1)

#After running through the predictions previously I have found that these tw
```

Linear Regression

In [ ]:
```
def split_data(dataset):
    global x_train, x_val_test, y_train, y_val_test, x_val, x_test, y_val, y
    #Must make the variables global to access the variables outside of the f

    columns_x = list(dataset.columns)
    if 'log_price' in columns_x:
        columns_x.remove('log_price')
    else:
        pass
    x_train, x_val_test, y_train, y_val_test = train_test_split(dataset[colu
    x_val, x_test, y_val, y_test = train_test_split(x_val_test, y_val_test,

    #return x_train, x_val, x_test, y_train, y_val, y_test
```

In [ ]:
```
split_data(train_df)
```

In [ ]:
```
print(x_train.shape, x_val.shape, x_test.shape)
```

```
(480000, 10) (60000, 10) (60000, 10)
```

In [ ]:
```
x_train.head()
```

Out[ ]:

	height	highway_fuel_economy	horsepower	length	mileage	year	body_ty
431111	59.6	39	109	175.4	29279	2019	
352633	67.5	33	170	182.3	24365	2016	
423966	67.3	23	305	186.7	110974	2013	
301502	57.1	27	272	191.1	136789	2007	
71067	57.3	36	152	175.6	30849	2019	

In [ ]:
```
x_train.isna().sum()
```

Out[ ]:
```
height                  0
highway_fuel_economy    0
horsepower              0
length                  0
mileage                 0
year                    0
body_type_enc           0
fuel_type_enc           0
wheel_system_enc        0
litres                  0
dtype: int64
```

In [ ]:
```
min_max_scaler = preprocessing.MinMaxScaler()

min_max_scaler.fit(x_train)
# transform
x_train_scaled = min_max_scaler.transform(x_train)
x_val_scaled = min_max_scaler.transform(x_val)
x_test_scaled = min_max_scaler.transform(x_test)
```

In [ ]:
```
#  Create linear regression object
lr = linear_model.LinearRegression()

# Train the model using the training set
lr.fit(x_train_scaled, y_train)

# Make predictions on the training and validation sets
y_train_pred_lr = lr.predict(x_train_scaled)
y_val_pred_lr = lr.predict(x_val_scaled)
y_test_pred_lr = lr.predict(x_test_scaled)

# You can use either x_train or x_train_scaled with regression models.
# To easily interpret the coefficients, unscaled variables are preferred.
```

In [ ]:
```
 # Print sq root of MSE on both sets
print('MSE root and mean on training set:', mean_squared_error(y_train, y_tr
print('MSE root and mean on validation set:', mean_squared_error(y_val, y_va
print('MSE root and mean on test set:', mean_squared_error(y_test, y_test_pr
# Print R squared on both sets
print('R squared on training set:', round(r2_score(y_train, y_train_pred_lr)
print('R squared on validation set:', round(r2_score(y_val, y_val_pred_lr),
print('R squared on test set:', round(r2_score(y_test, y_test_pred_lr), 3))
```

```
MSE root and mean on training set: 0.23624663944596228 9.843306998066895
MSE root and mean on validation set: 0.2365634725757123 9.843306998066895
MSE root and mean on test set: 0.23461254566629317 9.843306998066895
R squared on training set: 0.83
R squared on validation set: 0.828
R squared on test set: 0.833
```

LASSO

In [ ]:
```
lr_lasso = linear_model.Lasso(alpha=0.0005) #alpha is the lambda in the regu
lr_lasso.fit(x_train_scaled, y_train)
```

```
# Make predictions on the training and validation sets
y_train_pred = lr_lasso.predict(x_train_scaled)
y_val_pred = lr_lasso.predict(x_val_scaled)
y_test_pred = lr_lasso.predict(x_test_scaled)
```

In [ ]:
```
# Print sq root of MSE on both sets
print('MSE and mean on training set:', mean_squared_error(y_train, y_train_p
print('MSE and mean on validation set:', mean_squared_error(y_val, y_val_pre
print('MSE and mean on test set:', mean_squared_error(y_test, y_test_pred)**
# Print R squared on both sets
print('R squared on training set:', r2_score(y_train, y_train_pred))
print('R squared on validation set:', r2_score(y_val, y_val_pred))
print('R squared on test set:', r2_score(y_test, y_test_pred))
```

```
MSE and mean on training set: 0.23700996779216296 9.843306998066895
MSE and mean on validation set: 0.23736782868169776 9.843306998066895
MSE and mean on test set: 0.23553246667254493 9.843306998066895
R squared on training set: 0.8286484912537151
R squared on validation set: 0.8266494804793343
R squared on test set: 0.8312183992909303
```

In [ ]:
```
coefficients = pd.DataFrame()
coefficients['feature_name'] = x_train.columns
coefficients['coefficients'] = pd.Series(lr_lasso.coef_)
coefficients
```

Out[ ]:

	feature_name	coefficients
0	height	0.143040
1	highway_fuel_economy	-0.000000
2	horsepower	2.547239
3	length	-0.094635
4	mileage	-2.139767
5	year	1.719423
6	body_type_enc	-0.058861
7	fuel_type_enc	0.159535
8	wheel_system_enc	-0.240478
9	litres	-0.132448

LASSO Hyperparameter tuning

In [ ]:
```
lambdas = 1 * 0.90 ** np.arange(1,100)
```

In [ ]:
```
best_lambda = None
r2 = 0
# Step 2
# Estimate Lasso regression for each regularization parameter in grid
```

```
# Save if performance on validation is better than that of previous regressi
for lambda_j in lambdas:
    linear_reg_j = linear_model.Lasso(alpha = lambda_j)
    linear_reg_j.fit(x_train_scaled, y_train)
    # evaluate on validation set
    y_val_pred_j = linear_reg_j.predict(x_val_scaled)
    r2_j = r2_score(y_val, y_val_pred_j)
    if r2_j > r2:
        best_lambda = lambda_j
        r2 = r2_j
print(best_lambda, r2)
```

```
2.9512665430652825e-05 0.8278175667559583
```

In [ ]:

```
x_train_scaled_final = np.concatenate((x_train_scaled, x_val_scaled))
y_train_final = pd.concat([y_train,y_val], axis = 0)
lr_lasso_best = linear_model.Lasso(alpha = best_lambda)
lr_lasso_best.fit(x_train_scaled_final, y_train_final)
```

Out[ ]:

```
▾ Lasso
Lasso(alpha=2.9512665430652825e-05)
```

In [ ]:

```
y_test_pred = lr_lasso_best.predict(x_test_scaled)
# Print MAPE
print('MSE and mean on test set:', mean_squared_error(y_test, y_test_pred),
# Print R squared
print('R squared on test set:', r2_score(y_test, y_test_pred))
```

```
MSE and mean on test set: 0.055047148975792014 9.843306998066895
R squared on test set: 0.8325217665273015
```

In [ ]:

```
from sklearn.linear_model import LassoCV
lr_lasso_cv = LassoCV(cv=10, alphas= lambdas)
lr_lasso_cv.fit(x_train_scaled_final, y_train_final)
```

Out[ ]:

```
▾ LassoCV
LassoCV(alphas=array([9.00000000e-01, 8.10000000e-01, 7.29000000
e-01, 6.56100000e-01,
       5.90490000e-01, 5.31441000e-01, 4.78296900e-01, 4.3046721
0e-01,
       3.87420489e-01, 3.48678440e-01, 3.13810596e-01, 2.8242953
6e-01,
       2.54186583e-01, 2.28767925e-01, 2.05891132e-01, 1.8530201
9e-01,
       1.66771817e-01, 1.50094635e-01, 1.35085172e-01, 1.2157665
```

In [ ]:

```
lr_lasso_cv.alpha_
```

Out[ ]:

```
2.9512665430652825e-05
```

In [ ]:
```
y_test_pred = lr_lasso_cv.predict(x_test_scaled)
# Print MAPE
print('MSE and mean on test set:', mean_squared_error(y_test, y_test_pred)**

r2_cv = r2_score(y_test, y_test_pred)

# Print R squared
print('R squared on test set:', r2_cv)
```

```
MSE and mean on test set: 0.23462128841132898 9.843306998066895
R squared on test set: 0.8325217665273015
```

In [ ]:
```
coefficients = pd.DataFrame()
coefficients['feature_name'] = x_train.columns
coefficients['coefficients_val_best'] = pd.Series(lr_lasso_best.coef_)
coefficients['coefficients_cv'] = pd.Series(lr_lasso_cv.coef_)
coefficients
```

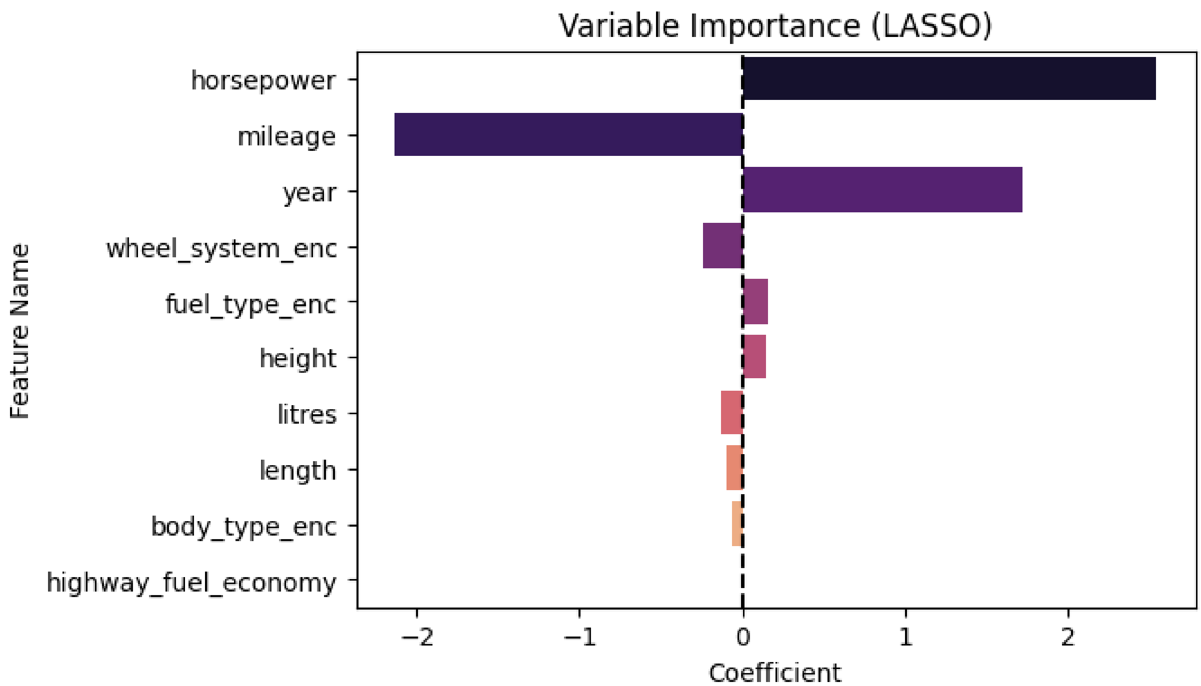
Out[ ]:

	feature_name	coefficients_val_best	coefficients_cv
0	height	0.264731	0.264731
1	highway_fuel_economy	-0.027391	-0.027391
2	horsepower	2.738881	2.738881
3	length	-0.207786	-0.207786
4	mileage	-2.135798	-2.135798
5	year	1.727112	1.727112
6	body_type_enc	-0.070655	-0.070655
7	fuel_type_enc	0.255833	0.255833
8	wheel_system_enc	-0.235606	-0.235606
9	litres	-0.262384	-0.262384

In [ ]:
```
coefficients = pd.DataFrame()
coefficients['feature_name'] = x_train.columns
coefficients['coefficients'] = pd.Series(lr_lasso.coef_)


# Sort the coefficients by absolute value
coefficients = coefficients.reindex(coefficients['coefficients'].abs().sort_

# Plot the variable importance
plt.figure(figsize=(6, 4))
sns.barplot(data=coefficients, x='coefficients', y='feature_name', palette='
plt.xlabel('Coefficient')
plt.ylabel('Feature Name')
plt.title('Variable Importance (LASSO)')

# Add a dotted vertical line at x = 0
plt.axvline(x=0, color='black', linestyle='--')
```

```
plt.show()
```

```
/var/folders/g9/7gqbb_gn4tv717l8v7pyt28c0000gn/T/ipykernel_13898/1551789741.
py:11: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed
in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the
same effect.

  sns.barplot(data=coefficients, x='coefficients', y='feature_name', palette
='magma')
```



In [ ]:
```
small_df.head()
```

Out[ ]:

	price	back_legroom	body_type	engine_displacement	exterior_color	fuel_type
0	18495.0	44.5 in	Pickup Truck	5700	MAROON	Gasoline
1	16422.0	41.4 in	Sedan	1800	Black Sand Pearl	Gasoline
2	39935.0	36.5 in	Sedan	2000	JET BLACK	Gasoline
3	23949.0	38.7 in	SUV / Crossover	3500	Brilliant Silver	Gasoline
4	37545.0	35.2 in	Sedan	2000	Black	Gasoline

In [ ]:
```
test_df = test_df.drop('log_price', axis=1)
test_df.head()
```

Out[ ]:

	height	highway_fuel_economy	horsepower	length	mileage	year	body_type_enc
0	58.1	22	310	180.9	10265	2019	6
1	57.3	36	132	183.1	35574	2017	6
2	66.5	33	190	180.6	10885	2019	5
3	58.0	40	188	191.8	2986	2019	6
4	70.6	24	278	212.3	17085	2019	4

In [ ]:

```
min_max_scaler.fit(test_df)

# transform
X_final_scaled = min_max_scaler.transform(test_df)

# Make predictions on the test set using the lr_lasso_cv model
final_pred_test = lr_lasso_cv.predict(X_final_scaled)

# Print the predictions
print(final_pred_test)
```

```
[10.50148272  9.55570395 10.12622109 ...  9.75724422 10.62547105
 10.57497591]
```

In [ ]:

```
predictions_df = pd.DataFrame({'predictions': final_pred_test})

header = pd.DataFrame({
    'predictions': [21108082, 'GojoSatoru', round(r2_cv,3)]
})

header

output_df = pd.concat([header, predictions_df], axis=0)

output_df.to_csv('predictions_output.csv', index=False, header=False)
```