Peer-graded Assignment: Course Project

-Aarohan Verma

Dataset:Automobile Data Set

Creator/Donor:

Jeffrey C. Schlimmer (Jeffrey.Schlimmer '@' a.gp.cs.cmu.edu)

Sources:

1) 1985 Model Import Car and Truck Specifications, 1985 Ward's Automotive Yearbook.

2) Personal Auto Manuals, Insurance Services Office, 160 Water Street, New York, NY 10038

3) Insurance Collision Report, Insurance Institute for Highway Safety, Watergate 600, Washington, DC 20037

Data Set Information:

This data set consists of three types of entities:

(a) the specification of an auto in terms of various characteristics,

(b) its assigned insurance risk rating,

(c) its normalized losses in use as compared to other cars.

```
The second rating corresponds to the degree to which the auto is more risky than its price indicates.
Cars are initially assigned a risk factor symbol associated with its price.
Then, if it is more risky (or less), this symbol is adjusted by moving it up (or down) the scale.
Actuarians call this process "symboling". A value of +3 indicates that the auto is risky, -3 that it is probably pretty safe.
```

The third factor is the relative average loss payment per insured vehicle year. This value is normalized for all autos within a particular size classification (two-door small, station wagons, sports/speciality, etc...), and represents the average loss per car per year.

Note: Several of the attributes in the database could be used as a "class" attribute.

Attribute Information:

Attribute: Attribute Range

1. symboling: -3, -2, -1, 0, 1, 2, 3.

2. normalized-losses: continuous from 65 to 256.

3. make:

alfa-romero, audi, bmw, chevrolet, dodge, honda,

isuzu, jaguar, mazda, mercedes-benz, mercury,

mitsubishi, nissan, peugot, plymouth, porsche,

renault, saab, subaru, toyota, volkswagen, volvo

4. fuel-type: diesel, gas.

5. aspiration: std, turbo.

6. num-of-doors: four, two.

7. body-style: hardtop, wagon, sedan, hatchback, convertible.

8. drive-wheels: 4wd, fwd, rwd.

9. engine-location: front, rear.

10. wheel-base: continuous from 86.6 120.9.

11. length: continuous from 141.1 to 208.1.

12. width: continuous from 60.3 to 72.3.

13. height: continuous from 47.8 to 59.8.

14. curb-weight: continuous from 1488 to 4066.

15. engine-type: dohc, dohcv, l, ohc, ohcf, ohcv, rotor.

16. num-of-cylinders: eight, five, four, six, three, twelve, two.

17. engine-size: continuous from 61 to 326.

18. fuel-system: 1bbl, 2bbl, 4bbl, idi, mfi, mpfi, spdi, spfi.

19. bore: continuous from 2.54 to 3.94.

20. stroke: continuous from 2.07 to 4.17.

21. compression-ratio: continuous from 7 to 23.

22. horsepower: continuous from 48 to 288.

23. peak-rpm: continuous from 4150 to 6600.

24. city-mpg: continuous from 13 to 49.

25. highway-mpg: continuous from 16 to 54.

26. price: continuous from 5118 to 45400.

Multiple Linear Regression Analysis:

Objective of Analysis: Automobile Price Prediction

- To build a Linear regression model which can predict the price of a car, given its attributes with a good degree of accuracy

In [1]:
```
#Importing necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet
from sklearn.linear_model import RidgeCV
from sklearn.linear_model import LassoCV
from sklearn.linear_model import ElasticNetCV
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.model_selection import KFold, cross_val_predict
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
#Importing Dataset
df=pd.read_csv('Automobile_data.csv')
df.head()
```

Out[2]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke	compression-ratio	horsepower	peak-rpm	city-mpg	highway-mpg	price
0	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	111	5000	21	27	13495
1	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	111	5000	21	27	16500
2	1	?	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	9.0	154	5000	19	26	16500
3	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.4	10.0	102	5500	24	30	13950
4	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.4	8.0	115	5500	18	22	17450

5 rows × 26 columns

It can be observed that the dataset has some missing values

In [3]:
```
#Statistical Figures about the dataset
df.describe()
```

Out[3]:

	symboling	wheel-base	length	width	height	curb-weight	engine-size	compression-ratio	city-mpg	highway-mpg
count	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000
mean	0.834146	98.756585	174.049268	65.907805	53.724878	2555.565854	126.907317	10.142537	25.219512	30.751220
std	1.245307	6.021776	12.337289	2.145204	2.443522	520.680204	41.642693	3.972040	6.542142	6.886443
min	-2.000000	86.600000	141.100000	60.300000	47.800000	1488.000000	61.000000	7.000000	13.000000	16.000000
25%	0.000000	94.500000	166.300000	64.100000	52.000000	2145.000000	97.000000	8.600000	19.000000	25.000000
50%	1.000000	97.000000	173.200000	65.500000	54.100000	2414.000000	120.000000	9.000000	24.000000	30.000000
75%	2.000000	102.400000	183.100000	66.900000	55.500000	2935.000000	141.000000	9.400000	30.000000	34.000000
max	3.000000	120.900000	208.100000	72.300000	59.800000	4066.000000	326.000000	23.000000	49.000000	54.000000

Upon a superficial analysis of the dataset, it can be presumed that the dataset is free of any serious outliers.

However, in practical scenarios, an in-depth analysis should be conducted under the guidance of domain-experts to study the presence and effect of outliers.

In [4]:
```
#Data type of various features
print(df.dtypes)
```

```
symboling              int64
normalized-losses     object
make                  object
fuel-type             object
aspiration            object
num-of-doors          object
body-style            object
drive-wheels          object
engine-location       object
wheel-base           float64
length               float64
width                float64
height               float64
curb-weight            int64
engine-type           object
num-of-cylinders      object
engine-size            int64
fuel-system           object
bore                  object
stroke                object
compression-ratio    float64
horsepower            object
peak-rpm              object
```

```
city-mpg              int64
highway-mpg           int64
price                object
dtype: object
```

In [5]:
```
#No. of unique observations for every feature
print(df.nunique())
```

```
symboling              6
normalized-losses     52
make                  22
fuel-type              2
aspiration             2
num-of-doors           3
body-style             5
drive-wheels           3
engine-location        2
wheel-base            53
length                75
width                 44
height                49
curb-weight          171
engine-type            7
num-of-cylinders       7
engine-size           44
fuel-system            8
bore                  39
stroke                37
compression-ratio     32
horsepower            60
peak-rpm              24
city-mpg              29
highway-mpg           30
price                187
dtype: int64
```

In [6]:
```
#List of features with missing values
feat_with_missing_val=list()
for feature in df.columns:
    if '?' in df[feature].unique().tolist():
        feat_with_missing_val.append(feature)
print(feat_with_missing_val)
```

```
['normalized-losses', 'num-of-doors', 'bore', 'stroke', 'horsepower', 'peak-rpm', 'price']
```

In [7]:
```
print(df[df[feat_with_missing_val]=='?'].count())
df.replace('?',np.nan,inplace=True) #Replacing '?' with NaN value

#Clearly 'normalized-losses' has too many missing values and we cannot afford to drop all such rows,
#hence it is advisable to replace the respective missing values by the median/mode value

#Other features have relatively less number of missing values,
#hence we can either simply drop them or replace them by the median/mode value
if 'normalized-losses' in feat_with_missing_val:
    feat_with_missing_val.remove('normalized-losses')
```

```
symboling             0
normalized-losses    41
make                  0
```

```
fuel-type            0
aspiration           0
num-of-doors         2
body-style           0
drive-wheels         0
engine-location      0
wheel-base           0
length               0
width                0
height               0
curb-weight          0
engine-type          0
num-of-cylinders     0
engine-size          0
fuel-system          0
bore                 4
stroke               4
compression-ratio    0
horsepower           2
peak-rpm             2
city-mpg             0
highway-mpg          0
price                4
dtype: int64
```

In [8]:
```
#Removing rows for features(excluding 'normalized-losses') with missing values
for feat in feat_with_missing_val:
    missing_val=df[df[feat].isnull()].index.tolist()
    df.drop(missing_val,axis=0,inplace=True)
```

In [9]:
```
#Typecasting certain 'object' types to 'float'/'int' types for better numerical compatibility
df['num-of-cylinders'].replace({'two':'2','four':'4','six':'6','five':'5','eight':'8','three':'3','twelve':'12'},inplace=True)
df['num-of-cylinders']=df['num-of-cylinders'].astype(np.int64)
df['num-of-doors'].replace({'two':'2','four':'4'},inplace=True)
df['num-of-doors']=df['num-of-doors'].astype(np.int64)
df['peak-rpm']=df['peak-rpm'].astype(np.int64)
df['bore']=df['bore'].astype(np.float64)
df['stroke']=df['stroke'].astype(np.float64)
df['horsepower']=df['horsepower'].astype(np.int64)
df['price']=df['price'].astype(np.float64)
```

In [10]:
```
#Replacing NaN values in 'normalized-losses' with the modal value
missing_losses_indices=df[df['normalized-losses'].isnull()].index.tolist()
df.loc[missing_losses_indices,'normalized-losses']='0'
df['normalized-losses']=df['normalized-losses'].astype(np.int32)
df['normalized-losses'].replace(0,df['normalized-losses'].mode()[0],inplace=True)
```

In [11]:
```
print(df.isnull().sum())
#Cleary all the missing values have been removed/imputed
```

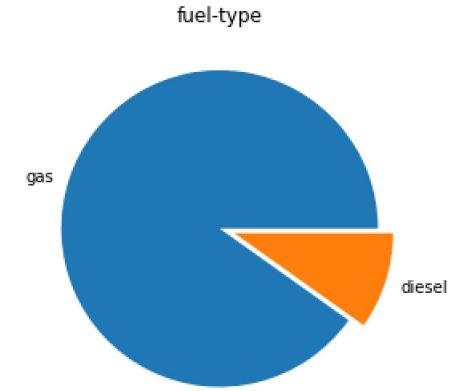
```
symboling           0
normalized-losses   0
make                0
fuel-type           0
aspiration          0
num-of-doors        0
body-style          0
```

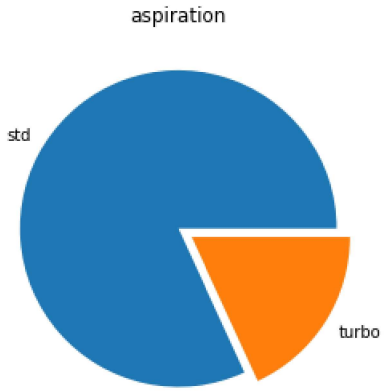
```
drive-wheels         0
engine-location      0
wheel-base           0
length               0
width                0
height               0
curb-weight          0
engine-type          0
num-of-cylinders     0
engine-size          0
fuel-system          0
bore                 0
stroke               0
compression-ratio    0
horsepower           0
peak-rpm             0
city-mpg             0
highway-mpg          0
price                0
dtype: int64
```

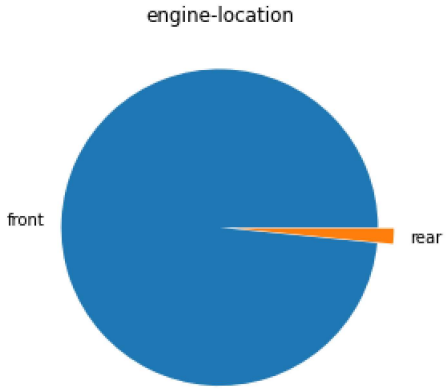
Data Exploration:

Pie-Charts:

In [12]:
```
fig, axs = plt.subplots(2, 2)
axs[0, 0].pie(x=df['fuel-type'].value_counts(), labels=list(df['fuel-type'].value_counts().index), explode = (0, 0.1))
axs[0, 0].set_title('fuel-type')
axs[0, 1].pie(x=df['aspiration'].value_counts(), labels=list(df['aspiration'].value_counts().index), explode = (0, 0.1))
axs[0, 1].set_title('aspiration')
axs[1, 0].pie(x=df['engine-location'].value_counts(), labels=list(df['engine-location'].value_counts().index), explode = (0, 0.1))
axs[1, 0].set_title('engine-location')
axs[1, 1].pie(x=df['num-of-doors'].value_counts(), labels=list(df['num-of-doors'].value_counts().index), explode = (0, 0.1))
axs[1, 1].set_title('num-of-doors')
fig = plt.gcf()
fig.set_size_inches(14, 10)
```







For the given dataset:

-Vast majority of the cars run on gas

-Majority of cars use std aspiration

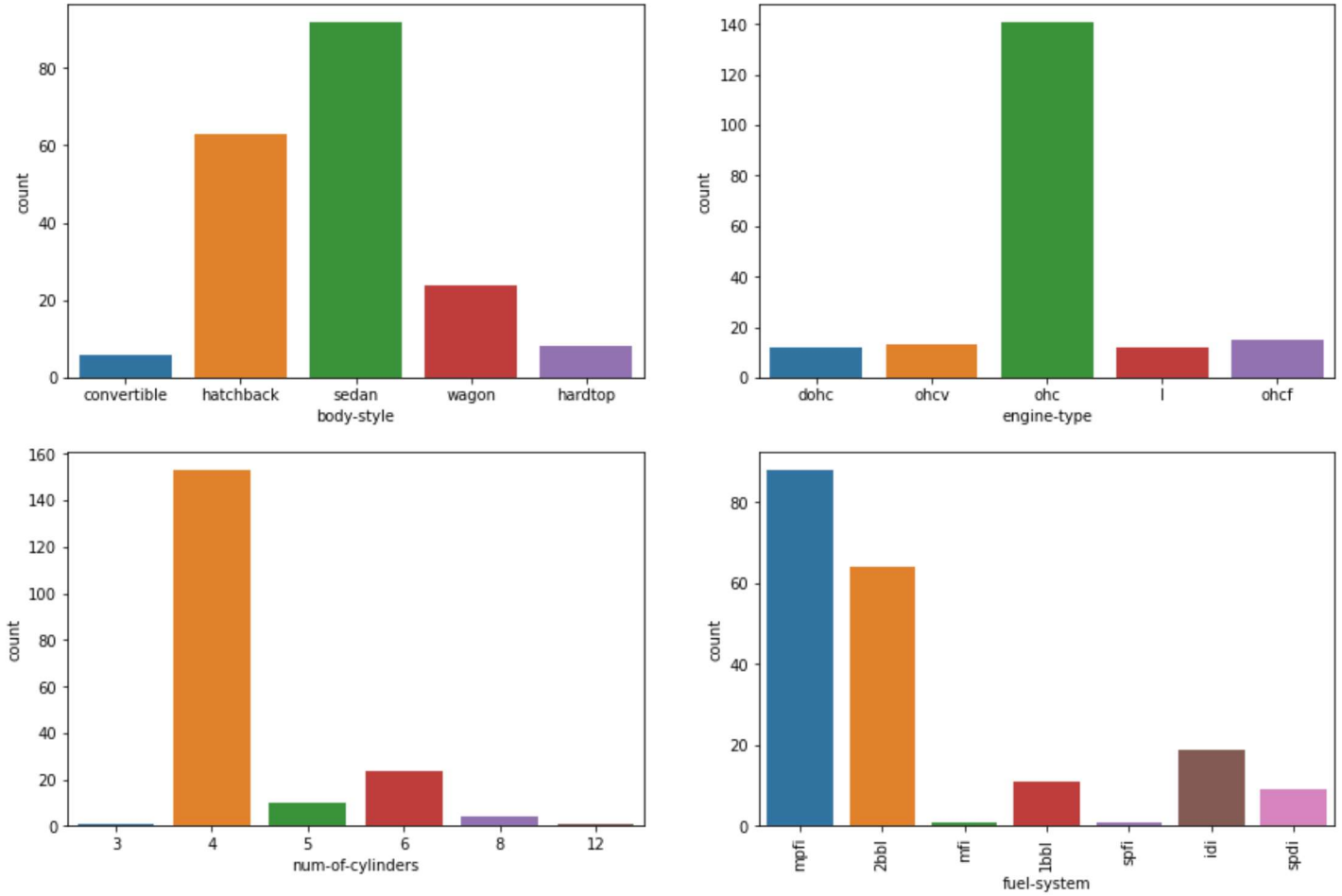
-Vast majority of the cars have engine at the front

-Majority of cars have 4 doors

Count-Plots:

In [13]:
```
fig, ax =plt.subplots(2,2)
sns.countplot(df['body-style'],ax=ax[0][0])
sns.countplot(df['engine-type'],ax=ax[0][1])
sns.countplot(df['num-of-cylinders'],ax=ax[1][0])
```

```
sns.countplot(df['fuel-system'],ax=ax[1][1])
plt.xticks(rotation=90)
fig.show()
fig = plt.gcf()
fig.set_size_inches(15, 10)
```



For the given dataset:

-Majority of cars have sedan body

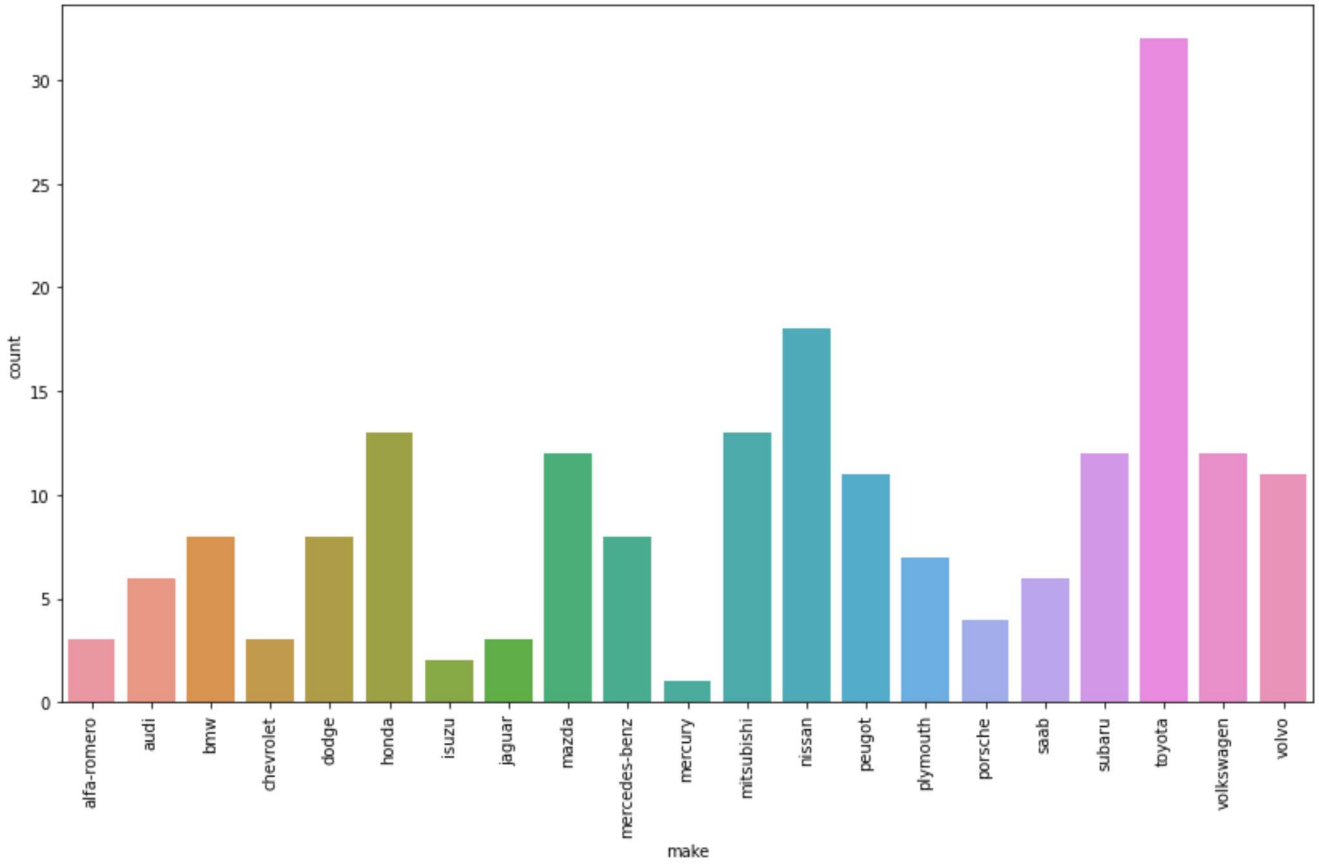
-Vast majority of cars have ohc-type engine

-Vast majority of cars have 4 cylinders

-Majority of cars have mpfi fuel-system

Count-Plot distribution of cars by manufacturers:

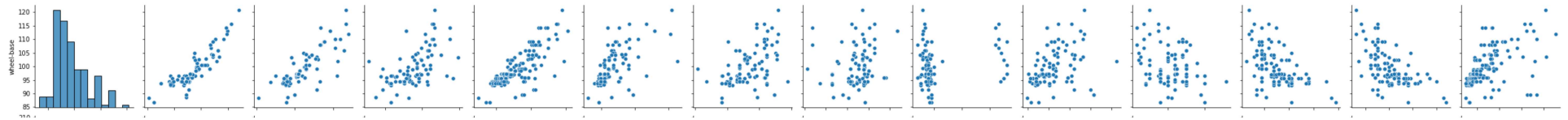
In [14]:
```
plt.figure(figsize=(14,8))
sns.countplot(df.make)
plt.xticks(rotation=90)
plt.show()
```

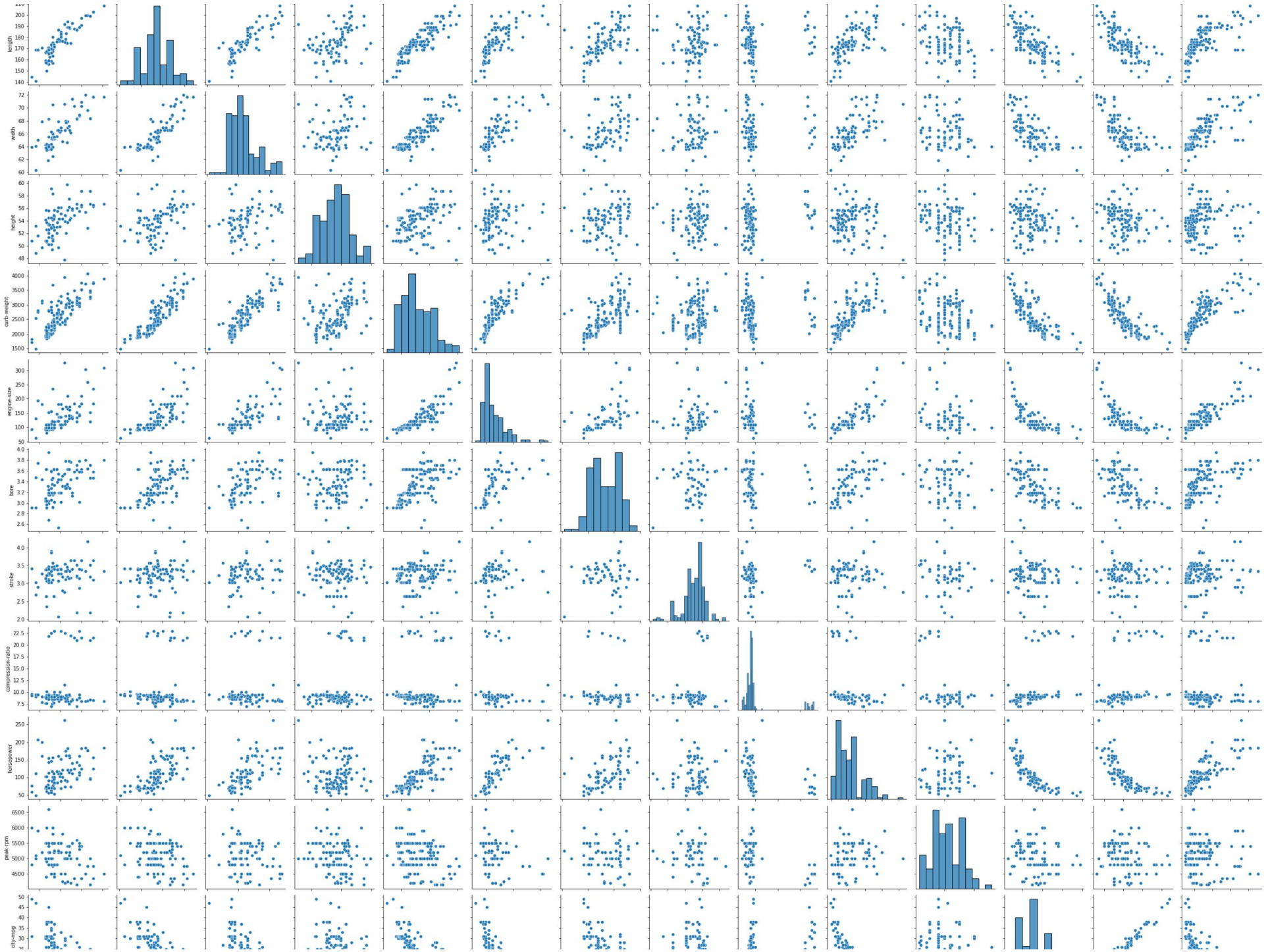


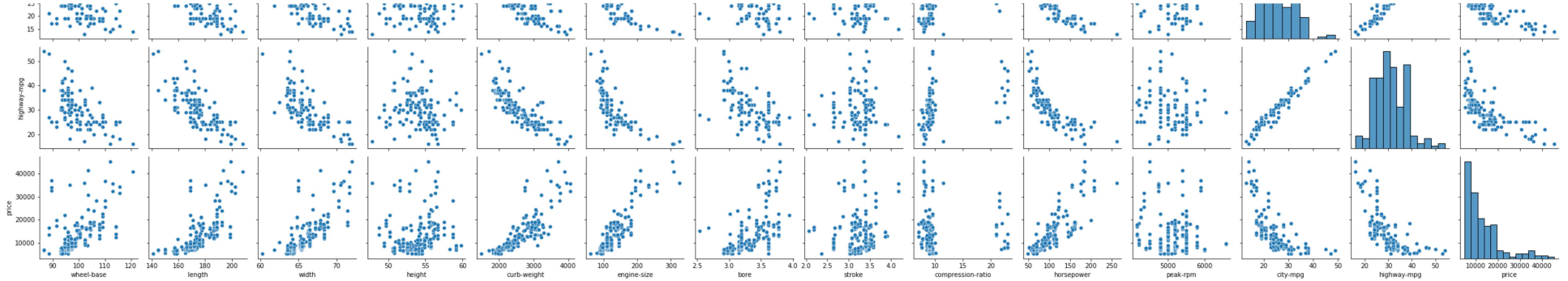
Pair-Plot of features:

In [15]:
```
rem_cols=['wheel-base','length','width','height','curb-weight','engine-size','bore', 'stroke','compression-ratio', 'horsepower', 'peak-rpm', 'city-mpg', 'highway-mpg',
sns.pairplot(df[rem_cols])
```

Out[15]: <seaborn.axisgrid.PairGrid at 0x131d2672280>



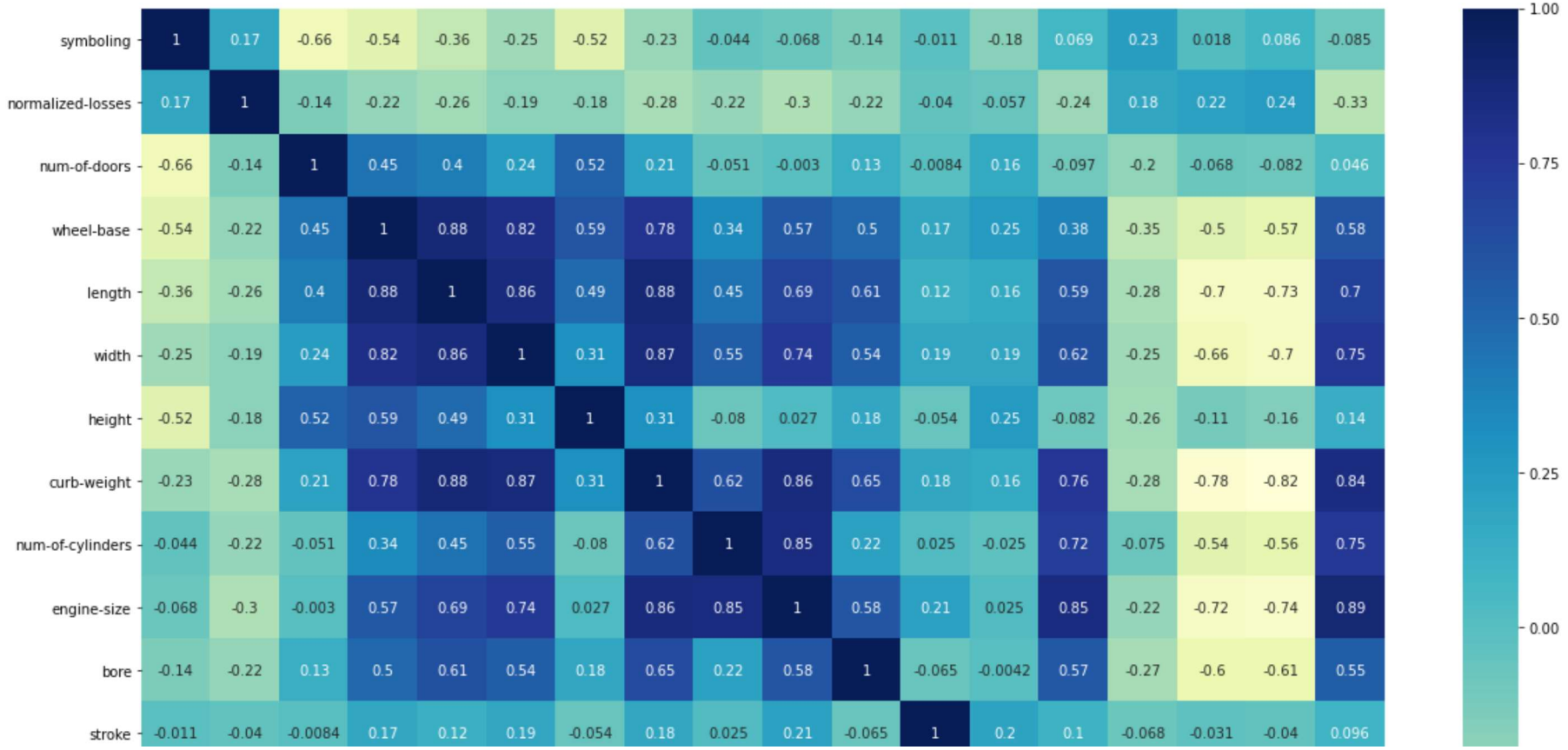

length
width
height
curb-weight
engine-size
bore
stroke
compression-ratio
horsepower
peak-rpm
city-mpg



Correlation-heatmap for features:

In [16]:
```
plt.subplots(figsize=(20,15))
sns.heatmap(df.corr(), cmap="YlGnBu", annot=True)
```

Out[16]: <AxesSubplot:>



Observations from Pair-Plot and Heatmap:

- Wheel base is highly positively corelated with length,width and curb-weight
- Length is highly positively corelated with wheel-base,width and curb-weight
- Length is negatively corelated with city and highway-mpg
- Width is highly positively corelated with wheel-base, ngth and curb-weight
- Curb weight is highly positively corelated with wheel-base, length, width, engine-size, horsepower and price
- Curb weight is highly negatively corelated with city and highway-mpg
- No. of cylinders is highly positively corelated with engine-size and horsepower
- Engine size is highly positively corelated with curb-weight, no.of cylinders, horsepower, price
- Engine size is negatively corelated with city and highway-mpg
- Horsepower is highly positively corelated with engine-size, curb-weight and price
- Horsepower is highly negatively corelated with city and highway-mpg
- City and Highway-mpg are highly negatively corelated with length, curb-weight, engine-size, horsepower
- Price is highly positively corelated with curb-weight, engine-size and horsepower

Regression Models under consideration:

Model 1: Model with only Numerical Features and no Categorical Features

Model 2: Model 1 with Polynomial and Interaction Features included

Model 3: Model with Numerical Features as well as Categorical Features that have been One-Hot Encoded

Model 4: Model 3 with Polynomial and Interaction Features included

Procedures followed:

- Scaling of Numerical Features only

- Avoided scaling of One-Hot Encoded Features

- Addition of Polynomial and Interaction Features prior to Scaling

- Addition of Polynomial and Interaction Features only for Numerical Features (and not One-Hot-Encoded Features)

In [17]:
```
df=df.reset_index(drop=True) #Index reset after reemoval of NaN values
#Separate lists for numerical and categorical features
cat=df.select_dtypes(np.object).columns.tolist()
num=df.select_dtypes(np.number).columns.tolist()
```

In [18]:
```
#DataFrame with only numerical features
df_num=df[num]
x_num=df_num.iloc[:,:-1]
y_num=df_num.iloc[:,-1]

#DataFrame with numerical features as well as One-hot encoded categorical features
df_ohc=pd.get_dummies(df,drop_first=True)
df_ohc_list=df_ohc.columns.tolist()
df_ohc_list.remove('price')
x_ohc=df_ohc[df_ohc_list]
y_ohc=df_ohc['price']
```

In [19]:
```
scaler=StandardScaler() #Initializing StandardScaler object
kf = KFold(shuffle=True, random_state=2, n_splits=10)
```

In [20]:
```
#This function takes in training and testing values and fits them for Linear Regression, Ridge Regularization, Lasso Regularization
#and ElasticNet Regularization.
#It also returns the Root Mean-Squared Errors and R2 Scores for the respective models
#The appropriate values for alphas are determined by using the respective Regularization CV Methods
def tune(xtrain,xtest,ytrain,ytest):
    scores=list()

    #Linear Regression:
    model=LinearRegression()
    model.fit(xtrain,ytrain)

    #Training Set
```

```
ypred=model.predict(xtrain)
scores.append(np.sqrt(mean_squared_error(ypred,ytrain))) #RMSE
scores.append(r2_score(ypred,ytrain)) #R2 Score

#Testing Set
ypred=model.predict(xtest)
scores.append(np.sqrt(mean_squared_error(ypred,ytest))) #RMSE
scores.append(r2_score(ypred,ytest)) #R2 Score


#Ridge Cross Validation for determination of best alpha
alphas = np.arange(5,200,10)
ridgeCV = RidgeCV(alphas=alphas, cv=kf).fit(xtrain,ytrain)


#Lasso Cross Validation for determination of best alpha
alphas2 = np.arange(5,500,10)
lassoCV = LassoCV(alphas=alphas2,max_iter=5e6, cv=kf).fit(xtrain,ytrain)


#ElasticNet Cross Validation for determination of best alpha and l1-ratio
l1_ratios = np.linspace(0.1, 0.9, 9)
elasticNetCV = ElasticNetCV(alphas=alphas, l1_ratio=l1_ratios, max_iter=5e6,cv=kf).fit(xtrain,ytrain)


#Ridge Regularization:
model=Ridge(alpha=ridgeCV.alpha_)
model.fit(xtrain,ytrain)

#Training Set
ypred=model.predict(xtrain)
scores.append(np.sqrt(mean_squared_error(ypred,ytrain))) #RMSE
scores.append(r2_score(ypred,ytrain)) #R2 Score

#Testing Set
ypred=model.predict(xtest)
scores.append(np.sqrt(mean_squared_error(ypred,ytest))) #RMSE
scores.append(r2_score(ypred,ytest)) #R2 Score


#Lasso Regularization:
model=Lasso(alpha=lassoCV.alpha_,max_iter=5e6)
model.fit(xtrain,ytrain)

#Training Set
ypred=model.predict(xtrain)
scores.append(np.sqrt(mean_squared_error(ypred,ytrain))) #RMSE
scores.append(r2_score(ypred,ytrain)) #R2 Score

#Testing Set
ypred=model.predict(xtest)
scores.append(np.sqrt(mean_squared_error(ypred,ytest))) #RMSE
scores.append(r2_score(ypred,ytest)) #R2 Score


#ElasticNet Regularization:
```

```
    model=ElasticNet(alpha=elasticNetCV.alpha_,l1_ratio=elasticNetCV.l1_ratio_,max_iter=5e6)
    model.fit(xtrain,ytrain)

    #Training Set
    ypred=model.predict(xtrain)
    scores.append(np.sqrt(mean_squared_error(ypred,ytrain))) #RMSE
    scores.append(r2_score(ypred,ytrain)) #R2 Score

    #Testing Set
    ypred=model.predict(xtest)
    scores.append(np.sqrt(mean_squared_error(ypred,ytest))) #RMSE
    scores.append(r2_score(ypred,ytest)) #R2 Score

    return scores
```

In [21]:
```
#This function takes in a list of evaluation metric scores for various models (cross-validated over 10 spits) and returns
#the aggregate mean for each of them for training and testing sets
def score_agg(scores):
    scores=np.array(scores)
    linear=list()
    ridge=list()
    lasso=list()
    elastic=list()
    for l in scores:
        linear.append(l.reshape(-1,4)[0])
        ridge.append(l.reshape(-1,4)[1])
        lasso.append(l.reshape(-1,4)[2])
        elastic.append(l.reshape(-1,4)[3])
    linear_mean=np.array(linear).mean(axis=0) #Aggregate mean for vanilla linear regression
    ridge_mean=np.array(ridge).mean(axis=0) #Aggregate mean for Ridge Regularization
    lasso_mean=np.array(lasso).mean(axis=0) #Aggregate mean for Lasso Regularization
    elastic_mean=np.array(elastic).mean(axis=0) #Aggregate mean for ElasticNet Regularization
    return pd.DataFrame({'Linear Regression':linear_mean,'Ridge':ridge_mean,'Lasso':lasso_mean,'ElasticNet':elastic_mean},index=['Training RMSE','Training R2','Testing
```

Model 1: Only Numerical Features Included (No Categorical Features)

In [22]:
```
scores=list()
#K-Folds Cross Validation
for train_index, test_index in kf.split(x_num):
    x_train, x_test, y_train, y_test = (x_num.iloc[train_index, :],
                                        x_num.iloc[test_index, :],
                                        y_num[train_index.tolist()],
                                        y_num[test_index.tolist()])
    #Scaling
    x_train=scaler.fit_transform(x_train)
    x_test=scaler.transform(x_test)
    scores.append(tune(x_train, x_test, y_train, y_test))
scores1=score_agg(scores)
scores1
```

Out[22]:

	Linear Regression	Ridge	Lasso	ElasticNet
Training RMSE	2945.799964	3152.936503	3047.332712	3341.441200

	Linear Regression	Ridge	Lasso	ElasticNet
Training R2	0.845793	0.798973	0.825364	0.752157
Testing RMSE	3209.598574	3372.323446	3180.521481	3383.489962
Testing R2	0.779635	0.735508	0.783119	0.711865

Model 2: Essentially Model 1 with Polynomial and Interaction Features added

In [23]:
```
scores=list()
#Inclusion of Polynomial Features
x_poly1=PolynomialFeatures(degree=2,include_bias=False)
x_num1=x_poly1.fit_transform(x_num)
x_num1=pd.DataFrame(x_num1,columns=[x_poly1.get_feature_names()])

#K-Folds Cross Validation
for train_index, test_index in kf.split(x_num1):
    x_train, x_test, y_train, y_test = (x_num1.iloc[train_index, :],
                                        x_num1.iloc[test_index, :],
                                        y_num[train_index.tolist()],
                                        y_num[test_index.tolist()])
    #Scaling
    x_train=scaler.fit_transform(x_train)
    x_test=scaler.transform(x_test)
    scores.append(tune(x_train, x_test, y_train, y_test))
scores2=score_agg(scores)
scores2
```

Out[23]:

	Linear Regression	Ridge	Lasso	ElasticNet
Training RMSE	2.347966e+02	2934.487343	2574.827953	3127.798404
Training R2	9.991362e-01	0.835124	0.857925	0.799688
Testing RMSE	3.277081e+13	3511.680566	3463.632836	3401.980616
Testing R2	-4.068509e-02	0.762624	0.760461	0.742670

Model 3: Numerical Features as well as One-Hot-Encoded Categorical Features

In [24]:
```
scores=list()
if 'price' in num:
    num.remove('price')
#K-Folds Cross Validation
for train_index, test_index in kf.split(x_ohc):
    x_train, x_test, y_train, y_test = (x_ohc.iloc[train_index, :],
                                        x_ohc.iloc[test_index, :],
                                        y_ohc[train_index.tolist()],
                                        y_ohc[test_index.tolist()])
    #Scaling
```

```
    x_train=scaler.fit_transform(x_train)
    x_test=scaler.transform(x_test)
    scores.append(tune(x_train, x_test, y_train, y_test))
scores3=score_agg(scores)
scores3
```

Out[24]:

	Linear Regression	Ridge	Lasso	ElasticNet
Training RMSE	1.462204e+03	1716.827264	1756.245386	2104.833424
Training R2	9.659599e-01	0.949841	0.947617	0.914466
Testing RMSE	6.438787e+15	2330.796926	2334.172997	2409.092241
Testing R2	5.951535e-01	0.889491	0.890838	0.868201

Model 4: Essentially Model 3 with Polynomial and Numerical Features added

In [25]:

```
scores=list()
#Inclusion of Polynomial Features
x_poly2=PolynomialFeatures(degree=2,include_bias=False)
x_ohc1=x_poly2.fit_transform(x_ohc[num])
x_ohc1=pd.DataFrame(x_ohc1,columns=[x_poly2.get_feature_names()])

num_cols=x_ohc1.columns.tolist()
del_num=x_ohc.columns.tolist()

for x in num:
    del_num.remove(x)

x_ohc1=pd.concat([x_ohc1.reset_index(drop=True), x_ohc[del_num].reset_index(drop=True)], axis=1)

#K-Folds Cross Validation
for train_index, test_index in kf.split(x_ohc1):
    x_train, x_test, y_train, y_test = (x_ohc1.iloc[train_index, :],
                                        x_ohc1.iloc[test_index, :],
                                        y_ohc[train_index.tolist()],
                                        y_ohc[test_index.tolist()])
    #Scaling
    x_train=scaler.fit_transform(x_train)
    x_test=scaler.transform(x_test)
    scores.append(tune(x_train, x_test, y_train, y_test))
scores4=score_agg(scores)
scores4
```

Out[25]:

	Linear Regression	Ridge	Lasso	ElasticNet
Training RMSE	2.254142e+02	1379.247617	1690.379268	1869.276830
Training R2	9.992064e-01	0.968591	0.950557	0.938747
Testing RMSE	5.438601e+13	2383.139546	2333.616932	2437.052627
Testing R2	-2.791912e-02	0.891051	0.893892	0.880509

Comparison:

Model 1: Only Numerical Features Included (No Categorical Features)

In [26]:
```
scores1
```

Out[26]:

	Linear Regression	Ridge	Lasso	ElasticNet
Training RMSE	2945.799964	3152.936503	3047.332712	3341.441200
Training R2	0.845793	0.798973	0.825364	0.752157
Testing RMSE	3209.598574	3372.323446	3180.521481	3383.489962
Testing R2	0.779635	0.735508	0.783119	0.711865

Comments: Similar RMSE for training set and testing set indicates that the model is neither underfit nor overfit, however the moderate R2 score indicates there is still scope for improvement by incorporating more no. of features.

Model 2: Essentially Model 1 with Polynomial and Interaction Features added

In [27]:
```
scores2
```

Out[27]:

	Linear Regression	Ridge	Lasso	ElasticNet
Training RMSE	2.347966e+02	2934.487343	2574.827953	3127.798404
Training R2	9.991362e-01	0.835124	0.857925	0.799688
Testing RMSE	3.277081e+13	3511.680566	3463.632836	3401.980616
Testing R2	-4.068509e-02	0.762624	0.760461	0.742670

Comments: Great difference in RMSE for training and testing sets for vanilla linear regression indicates that the model is overfit. By implementing various regularization methods, the model is no more underfit, also the RMSE for training and testing sets is now comparable. This model (with regulatization) offers slight improvement over Model 1. Inclusion of Polynomial and Interaction Features followed by Regularization has offered a better fit.

Model 3: Numerical Features as well as One-Hot-Encoded Categorical Features

In [28]:
```
scores3
```

Out[28]:

	Linear Regression	Ridge	Lasso	ElasticNet
Training RMSE	1.462204e+03	1716.827264	1756.245386	2104.833424
Training R2	9.659599e-01	0.949841	0.947617	0.914466

	Linear Regression	Ridge	Lasso	ElasticNet
Testing RMSE	6.438787e+15	2330.796926	2334.172997	2409.092241
Testing R2	5.951535e-01	0.889491	0.890838	0.868201

Comments: Great difference in RMSE for training and testing sets for vanilla linear regression indicates that the model is overfit. By implementing various regularization methods, the model is no more underfit, also the RMSE for training and testing sets is now comparable. This model (with regulatization) offers quite a better improvement over Model 2. Inclusion of One-Hot-Encoded Categorical Features has provided better fit and explainability.

Model 4: Essentially Model 3 with Polynomial and Numerical Features added

In [29]:
```
scores4
```

Out[29]:

	Linear Regression	Ridge	Lasso	ElasticNet
Training RMSE	2.254142e+02	1379.247617	1690.379268	1869.276830
Training R2	9.992064e-01	0.968591	0.950557	0.938747
Testing RMSE	5.438601e+13	2383.139546	2333.616932	2437.052627
Testing R2	-2.791912e-02	0.891051	0.893892	0.880509

Comments: Great difference in RMSE for training and testing sets for vanilla linear regression indicates that the model is overfit. By implementing various regularization methods, the model is no more overfit; also the RMSE for training and testing sets is now comparable. This model (with regulatization) offers very slight improvement over Model 3. Comparing Model 2 and Model 3 with Model 4, we can infer that although the inclusion of Polynomial and Interaction Features provides a subtle improvement to the vanilla regression model, however the inclusion of One-Hot-Encoded Features has resulted in a considerably better model.

Conclusion: For the Given dataset, Model 4 with Lasso Regularization offers the best prediction accuracy (highly comparable RMSE for training and testing sets as well as an R2 Score of ~0.95 for training set and ~0.90 for testing set).

PS:The models can be revisited to implement log-transformations on positively skewed features. Moreover, the outliers can be looked into with a greater depth in order to fine tune the model even more. The performance of the model can also be potentially improved further by collecting and training on even greater no. of observations for the dataset.

In [ ]:
```
```