TASK 3

CAR PRICE PREDICTION

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import datasets, linear_model
from sklearn.preprocessing import LabelEncoder, OneHotEncoder,MinMaxScaler
from sklearn.preprocessing import StandardScaler
from scipy.stats import shapiro
from scipy.stats import anderson
from scipy.stats import normaltest
from scipy.stats import norm
from scipy.stats import skew
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
df=pd.read_csv("C:/Users/Namitha CV/OneDrive/Documents/CarPrice_Assignment.csv")
df
```

Out[2]:

	car_ID	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	wheelbase	...	enginesize	fuelsystem	boreratio	stroke	compressionratio	horsepower	peakrpm	citympg	highwaym
0	1	3	alfa-romero giulia	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	111	5000	21	
1	2	3	alfa-romero stelvio	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	111	5000	21	
2	3	1	alfa-romero Quadrifoglio	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	9.0	154	5000	19	
3	4	2	audi 100 ls	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	10.0	102	5500	24	
4	5	2	audi 100ls	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.40	8.0	115	5500	18	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
200	201	-1	volvo 145e (sw)	gas	std	four	sedan	rwd	front	109.1	...	141	mpfi	3.78	3.15	9.5	114	5400	23	
201	202	-1	volvo 144ea	gas	turbo	four	sedan	rwd	front	109.1	...	141	mpfi	3.78	3.15	8.7	160	5300	19	
202	203	-1	volvo 244dl	gas	std	four	sedan	rwd	front	109.1	...	173	mpfi	3.58	2.87	8.8	134	5500	18	
203	204	-1	volvo 246	diesel	turbo	four	sedan	rwd	front	109.1	...	145	idi	3.01	3.40	23.0	106	4800	26	
204	205	-1	volvo 264gl	gas	turbo	four	sedan	rwd	front	109.1	...	141	mpfi	3.78	3.15	9.5	114	5400	19	

205 rows × 26 columns

ATTRIBUTES INFO

car_ID: Unique ID # for each car

Symboling: Acturian assessment of risk of the car (numerical values where -3 is safe, +3 is risky)

CarName: Make and model of car

Fueltype: Car Fuel Type (gas or diesel)

Aspiration: Car Aspiration (standard or turbo)

Doornumber: Number of car doors (two or four)

Carbody: Type of car body (convertible, sedan, hatchback, wagon, or hardtop)

Drivewheel: Car drive wheel (rear wheel drive, 4 wheel drive or front wheel drive)

Enginelocation: Location of car engine (front of rear)

Wheelbase: Car wheel base in inches

Carlength:Car length in inches

Carwidth: Car width in inches

Carheight: Car height in inches

Curbweight: Car weight in pounds

Enginetype: Car engine type (dohc, dohcv, I, ohc, ohcf, ohcv, or rotor)

Cylindernumber: Number of car cylinders (two, three, four, five, six, eight, or twelve)

Enginesize: Size of engine (numerical values of cubic inches)

Fuelsystem: Type of car fuel system (1bbl, 2 bbl, 4 bbl, idi, mfi, mpfi, spdi, or spfi)

Boreratio: Car Bore-Stroke Ratio is the ratio between the dimensions of the engine cylinder bore diameter to its piston stroke-length

Stroke: Car strokes (numerical value in strokes)

Compressionratio: Car compression ratio (ratio between the volume of the cylinder with the piston in the bottom position, Vbottom (largest volume), and in the top position, Vtop (smallest volume))

Horsepower: Car horsepower (numerical values of horsepower)

Peakrpm: Car peak RPM (revolutions per minute)

Citympg: Car city MPG (miles per gallon)

Highwaympg: Car highway MPG (miles per gallon)

Price: total price of car in dollars

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

Out[3]:

	car_ID	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	wheelbase	...	enginesize	fuelsystem	boreratio	stroke	compressionratio	horsepower	peakrpm	citympg	highwaympg
0	1	3	alfa-romero giulia	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	111	5000	21	27
1	2	3	alfa-romero stelvio	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	111	5000	21	27
2	3	1	alfa-romero Quadrifoglio	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	9.0	154	5000	19	26
3	4	2	audi 100 ls	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	10.0	102	5500	24	30
4	5	2	audi 100ls	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.40	8.0	115	5500	18	22

5 rows × 26 columns

In [4]:
```
df.tail()
```

Out[4]:

	car_ID	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	wheelbase	...	enginesize	fuelsystem	boreratio	stroke	compressionratio	horsepower	peakrpm	citympg	highwaympg
200	201	-1	volvo 145e (sw)	gas	std	four	sedan	rwd	front	109.1	...	141	mpfi	3.78	3.15	9.5	114	5400	23	28
201	202	-1	volvo 144ea	gas	turbo	four	sedan	rwd	front	109.1	...	141	mpfi	3.78	3.15	8.7	160	5300	19	25
202	203	-1	volvo 244dl	gas	std	four	sedan	rwd	front	109.1	...	173	mpfi	3.58	2.87	8.8	134	5500	18	23
203	204	-1	volvo 246	diesel	turbo	four	sedan	rwd	front	109.1	...	145	idi	3.01	3.40	23.0	106	4800	26	27
204	205	-1	volvo 264gl	gas	turbo	four	sedan	rwd	front	109.1	...	141	mpfi	3.78	3.15	9.5	114	5400	19	25

5 rows × 26 columns

In [5]:
```
# Lets drop the column car_ID
df.drop("car_ID", axis=1, inplace=True)
```

In [6]:
```
df.shape
```

Out[6]:
```
(205, 25)
```

In [7]:
```
df.isnull().sum()
```

Out[7]:
```
symboling           0
CarName             0
fueltype            0
aspiration          0
doornumber          0
carbody             0
drivewheel          0
enginelocation      0
wheelbase           0
carlength           0
carwidth            0
carheight           0
curbweight          0
enginetype          0
cylindernumber      0
enginesize          0
fuelsystem          0
boreratio           0
stroke              0
compressionratio    0
horsepower          0
peakrpm             0
citympg             0
highwaympg          0
price               0
dtype: int64
```

this dataset do not have any null values

In [8]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 25 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   symboling         205 non-null    int64
 1   CarName           205 non-null    object
 2   fueltype          205 non-null    object
 3   aspiration        205 non-null    object
 4   doornumber        205 non-null    object
 5   carbody           205 non-null    object
 6   drivewheel        205 non-null    object
 7   enginelocation    205 non-null    object
 8   wheelbase         205 non-null    float64
 9   carlength         205 non-null    float64
 10  carwidth          205 non-null    float64
 11  carheight         205 non-null    float64
 12  curbweight        205 non-null    int64
 13  enginetype        205 non-null    object
 14  cylindernumber    205 non-null    object
 15  enginesize        205 non-null    int64
 16  fuelsystem        205 non-null    object
 17  boreratio         205 non-null    float64
 18  stroke            205 non-null    float64
 19  compressionratio  205 non-null    float64
 20  horsepower        205 non-null    int64
 21  peakrpm           205 non-null    int64
 22  citympg           205 non-null    int64
 23  highwaympg        205 non-null    int64
 24  price             205 non-null    float64
dtypes: float64(8), int64(7), object(10)
memory usage: 40.2+ KB
```

In [9]:
```
df.describe()
```

Out[9]:

	symboling	wheelbase	carlength	carwidth	carheight	curbweight	enginesize	boreratio	stroke	compressionratio	horsepower	peakrpm	citympg	highwaympg	price
count	205.000000	205.000000	205.000000	205.000000	205.000000	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	3.329756	3.255415	10.142537	104.117073	5125.121951	25.219512	30.751220	13276.710571
std	1.245307	6.021776	12.337289	2.145204	2.443522	520.680204	41.642693	0.270844	0.313597	3.972040	39.544167	476.985643	6.542142	6.886443	7988.852332
min	-2.000000	86.600000	141.100000	60.300000	47.800000	1488.000000	61.000000	2.540000	2.070000	7.000000	48.000000	4150.000000	13.000000	16.000000	5118.000000
25%	0.000000	94.500000	166.300000	64.100000	52.000000	2145.000000	97.000000	3.150000	3.110000	8.600000	70.000000	4800.000000	19.000000	25.000000	7788.000000
50%	1.000000	97.000000	173.200000	65.500000	54.100000	2414.000000	120.000000	3.310000	3.290000	9.000000	95.000000	5200.000000	24.000000	30.000000	10295.000000
75%	2.000000	102.400000	183.100000	66.900000	55.500000	2935.000000	141.000000	3.580000	3.410000	9.400000	116.000000	5500.000000	30.000000	34.000000	16503.000000
max	3.000000	120.900000	208.100000	72.300000	59.800000	4066.000000	326.000000	3.940000	4.170000	23.000000	288.000000	6600.000000	49.000000	54.000000	45400.000000

In [10]:
```
#checking for duplicates
df.loc[df.duplicated()]
```

Out[10]:

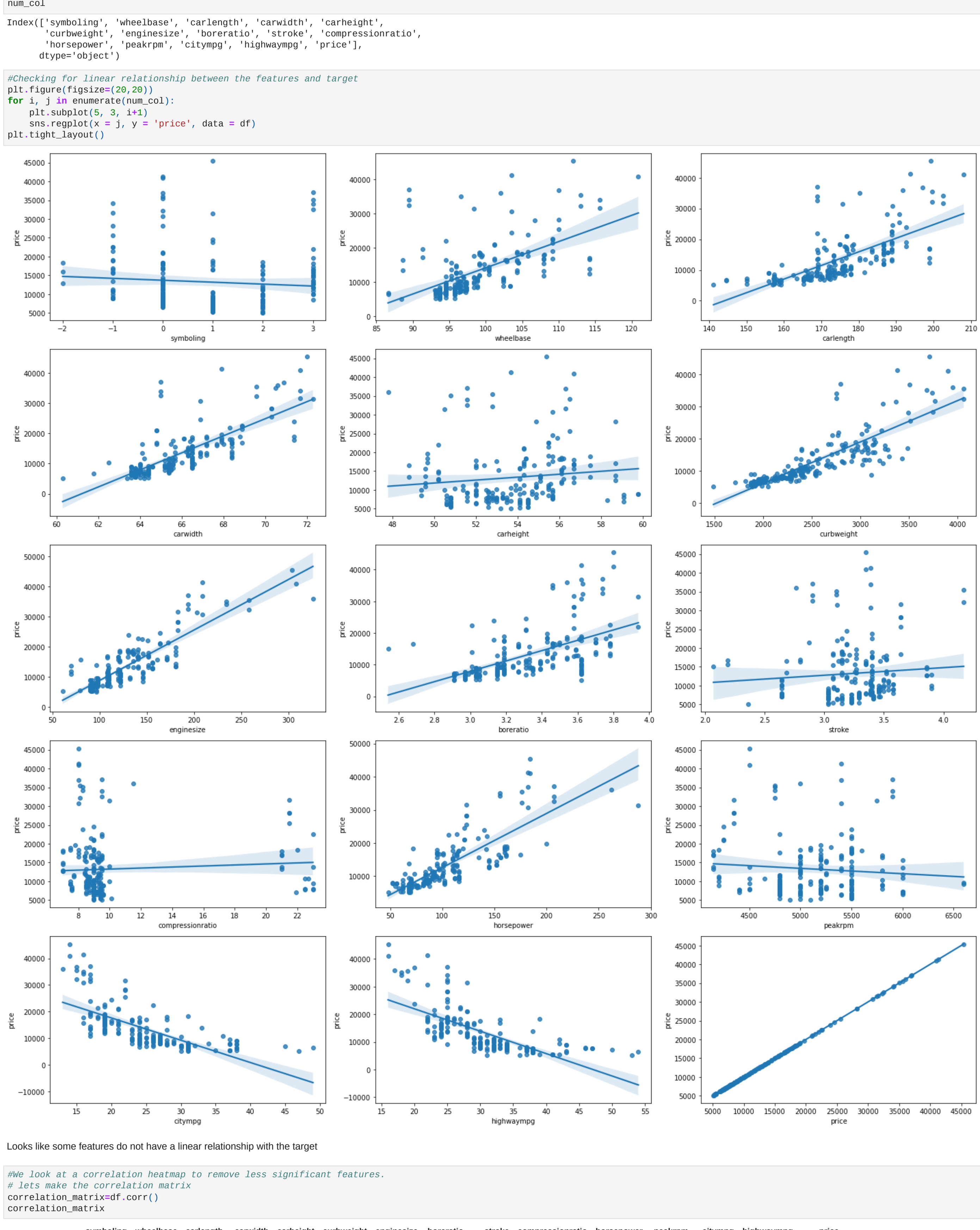
symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	wheelbase	carlength	...	enginesize	fuelsystem	boreratio	stroke	compressionratio	horsepower	peakrpm	citympg	highwaympg

0 rows × 25 columns

In [11]:
```
#Exploring the continuous features of the dataset
num_col = df.select_dtypes(exclude = 'object').columns
num_col
```

Out[11]:
```
Index(['symboling', 'wheelbase', 'carlength', 'carwidth', 'carheight',
       'curbweight', 'enginesize', 'boreratio', 'stroke', 'compressionratio',
       'horsepower', 'peakrpm', 'citympg', 'highwaympg', 'price'],
      dtype='object')
```

In [12]:
```
#Checking for linear relationship between the features and target
plt.figure(figsize=(20,20))
for i, j in enumerate(num_col):
    plt.subplot(5, 3, i+1)
    sns.regplot(x = j, y = 'price', data = df)
plt.tight_layout()
```



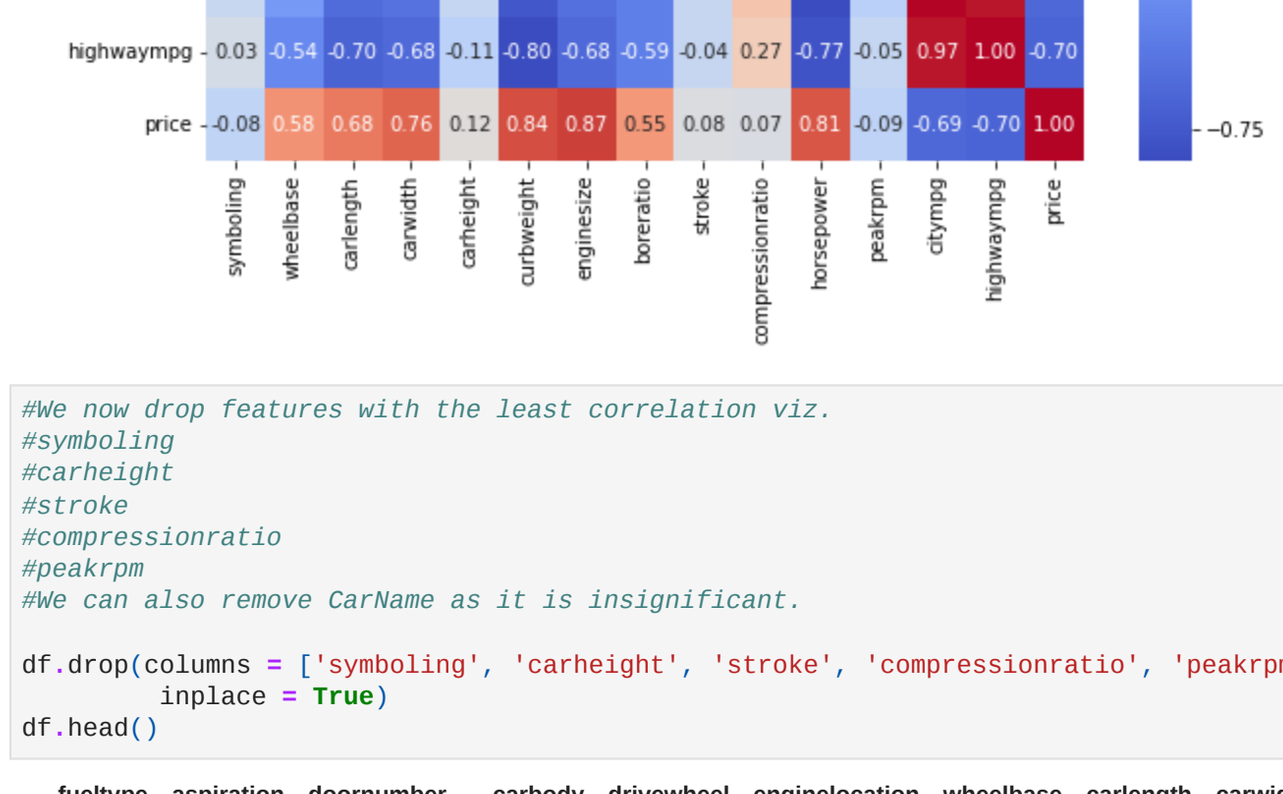
Looks like some features do not have a linear relationship with the target

In [13]:
```
#We look at a correlation heatmap to remove less significant features.
# lets make the correlation matrix
correlation_matrix=df.corr()
correlation_matrix
```

Out[13]:

	symboling	wheelbase	carlength	carwidth	carheight	curbweight	enginesize	boreratio	stroke	compressionratio	horsepower	peakrpm	citympg	highwaympg	price
symboling	1.000000	-0.531954	-0.357612	-0.232919	-0.541038	-0.227691	-0.105790	-0.130051	-0.008735	-0.178515	0.070873	0.273606	-0.035823	0.034606	-0.079978
wheelbase	-0.531954	1.000000	0.874587	0.795144	0.589435	0.776386	0.569329	0.488750	0.160959	0.249786	0.353294	-0.360469	-0.470414	-0.544082	0.577816
carlength	-0.357612	0.874587	1.000000	0.841118	0.491029	0.877728	0.683360	0.606454	0.129533	0.158414	0.552623	-0.287242	-0.670909	-0.704662	0.682920
carwidth	-0.232919	0.795144	0.841118	1.000000	0.279210	0.867032	0.735433	0.559150	0.182942	0.181129	0.640732	-0.220012	-0.642704	-0.677218	0.759325
carheight	-0.541038	0.589435	0.491029	0.279210	1.000000	0.295572	0.067149	0.171071	-0.055307	0.261214	-0.108802	-0.320411	-0.048640	-0.107358	0.119336
curbweight	-0.227691	0.776386	0.877728	0.867032	0.295572	1.000000	0.850594	0.648480	0.168790	0.151362	0.750739	-0.266243	-0.757414	-0.797465	0.835305
enginesize	-0.105790	0.569329	0.683360	0.735433	0.067149	0.850594	1.000000	0.583774	0.203129	0.028971	0.809769	-0.244660	-0.653658	-0.677470	0.874145
boreratio	-0.130051	0.488750	0.606454	0.559150	0.171071	0.648480	0.583774	1.000000	-0.055909	0.005197	0.573677	-0.254976	-0.584532	-0.587012	0.553173
stroke	-0.008735	0.160959	0.129533	0.182942	-0.055307	0.168790	0.203129	-0.055909	1.000000	0.186110	0.080940	-0.067964	-0.042145	-0.043931	0.079443
compressionratio	-0.178515	0.249786	0.158414	0.181129	0.261214	0.151362	0.028971	0.005197	0.186110	1.000000	-0.204326	-0.435741	0.324701	0.265201	0.067984
horsepower	0.070873	0.353294	0.552623	0.640732	-0.108802	0.750739	0.809769	0.573677	0.080940	-0.204326	1.000000	0.131073	-0.801456	-0.770544	0.808139
peakrpm	0.273606	-0.360469	-0.287242	-0.220012	-0.320411	-0.266243	-0.244660	-0.254976	-0.067964	-0.435741	0.131073	1.000000	-0.113544	-0.054275	-0.085267
citympg	-0.035823	-0.470414	-0.670909	-0.642704	-0.048640	-0.757414	-0.653658	-0.584532	-0.042145	0.324701	-0.801456	-0.113544	1.000000	0.971337	-0.685751
highwaympg	0.034606	-0.544082	-0.704662	-0.677218	-0.107358	-0.797465	-0.677470	-0.587012	-0.043931	0.265201	-0.770544	-0.054275	0.971337	1.000000	-0.697599
price	-0.079978	0.577816	0.682920	0.759325	0.119336	0.835305	0.874145	0.553173	0.079443	0.067984	0.808139	-0.085267	-0.685751	-0.697599	1.000000

In [14]:
```
# lets make the correlation matrix visualise
plt.figure(figsize=(10,10))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f")
plt.show()
```



In [15]:
```
#We now drop features with the least correlation viz.
#symboling
#carheight
#stroke
#compressionratio
#peakrpm
#We can also remove CarName as it is insignificant.

df.drop(columns = ['symboling', 'carheight', 'stroke', 'compressionratio', 'peakrpm', 'CarName'], axis = 1,
        inplace = True)
df.head()
```

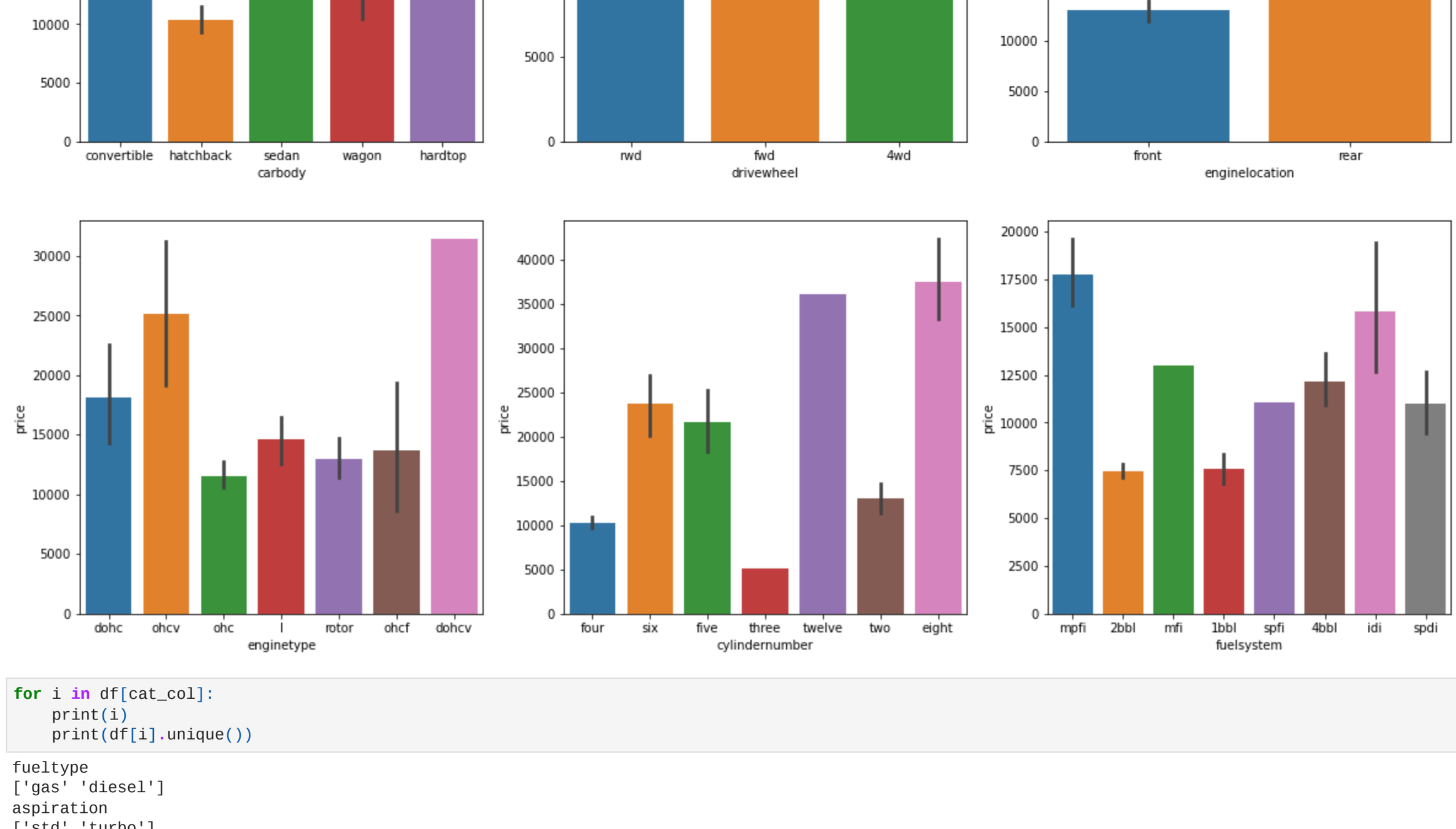
Out[15]:

	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	wheelbase	carlength	carwidth	curbweight	enginetype	cylindernumber	enginesize	fuelsystem	boreratio	horsepower	citympg	highwaympg	price
0	gas	std	two	convertible	rwd	front	88.6	168.8	64.1	2548	dohc	four	130	mpfi	3.47	111	21	27	13495.0
1	gas	std	two	convertible	rwd	front	88.6	168.8	64.1	2548	dohc	four	130	mpfi	3.47	111	21	27	16500.0
2	gas	std	two	hatchback	rwd	front	94.5	171.2	65.5	2823	ohcv	six	152	mpfi	2.68	154	19	26	16500.0
3	gas	std	four	sedan	fwd	front	99.8	176.6	66.2	2337	ohc	four	109	mpfi	3.19	102	24	30	13950.0
4	gas	std	four	sedan	4wd	front	99.4	176.6	66.4	2824	ohc	five	136	mpfi	3.19	115	18	22	17450.0

In [16]:
```
#Exploring the categorical features of the dataset
cat_col = df.select_dtypes(include = ['object']).columns
df[cat_col].nunique()
```

Out[16]:
```
fueltype          2
aspiration        2
doornumber        2
carbody           5
drivewheel        3
enginelocation    2
enginetype        7
cylindernumber    7
fuelsystem        8
dtype: int64
```

In [17]:
```
plt.figure(figsize=(20,20))
for i, j in enumerate(cat_col):
    plt.subplot(3, 3, i+1)
    sns.barplot(x = j, y = 'price', data = df)
plt.show()
```



In [18]:
```
for i in df[cat_col]:
    print(i)
    print(df[i].unique())
```

```
fueltype
['gas' 'diesel']
aspiration
['std' 'turbo']
doornumber
['two' 'four']
carbody
['convertible' 'hatchback' 'sedan' 'wagon' 'hardtop']
drivewheel
['rwd' 'fwd' '4wd']
enginelocation
['front' 'rear']
enginetype
['dohc' 'ohcv' 'ohc' 'l' 'rotor' 'ohcf' 'dohcv']
cylindernumber
['four' 'six' 'five' 'three' 'twelve' 'two' 'eight']
fuelsystem
['mpfi' '2bbl' 'mfi' '1bbl' 'spfi' '4bbl' 'idi' 'spdi']
```

One hot encoding

One hot encoding is a technique that we use to represent categorical variables as numerical values in a machine learning model.

In [19]:
```
df = pd.get_dummies(columns = ['fueltype', 'aspiration', 'doornumber', 'drivewheel', 'enginelocation'],
                    data = df, drop_first = True)
df.head()
```

Out[19]:

	carbody	wheelbase	carlength	carwidth	curbweight	enginetype	cylindernumber	enginesize	fuelsystem	boreratio	horsepower	citympg	highwaympg	price	fueltype_gas	aspiration_turbo	doornumber_two
0	convertible	88.6	168.8	64.1	2548	dohc	four	130	mpfi	3.47	111	21	27	13495.0	1	0	1
1	convertible	88.6	168.8	64.1	2548	dohc	four	130	mpfi	3.47	111	21	27	16500.0	1	0	1
2	hatchback	94.5	171.2	65.5	2823	ohcv	six	152	mpfi	2.68	154	19	26	16500.0	1	0	1
3	sedan	99.8	176.6	66.2	2337	ohc	four	109	mpfi	3.19	102	24	30	13950.0	1	0	0
4	sedan	99.4	176.6	66.4	2824	ohc	five	136	mpfi	3.19	115	18	22	17450.0	1	0	0

In [20]:
```
from sklearn.preprocessing import OrdinalEncoder, StandardScaler

encoder = OrdinalEncoder()
for i in df.select_dtypes(include = 'object').columns:
    df[i] = encoder.fit_transform(df[[i]])
df.head()
```

Out[20]:

	carbody	wheelbase	carlength	carwidth	curbweight	enginetype	cylindernumber	enginesize	fuelsystem	boreratio	horsepower	citympg	highwaympg	price	fueltype_gas	aspiration_turbo	doornumber_two
0	0.0	88.6	168.8	64.1	2548	0.0	2.0	130	5.0	3.47	111	21	27	13495.0	1	0	1
1	0.0	88.6	168.8	64.1	2548	0.0	2.0	130	5.0	3.47	111	21	27	16500.0	1	0	1
2	2.0	94.5	171.2	65.5	2823	5.0	3.0	152	5.0	2.68	154	19	26	16500.0	1	0	1
3	3.0	99.8	176.6	66.2	2337	3.0	2.0	109	5.0	3.19	102	24	30	13950.0	1	0	0
4	3.0	99.4	176.6	66.4	2824	3.0	1.0	136	5.0	3.19	115	18	22	17450.0	1	0	0

In [21]:
```
#model building
from sklearn.model_selection import train_test_split

X = df.drop('price', axis = 1)
y = df['price']

scaler = StandardScaler()
X = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 20)
```

In [22]:
```
#Linear Regression
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

lr_model = LinearRegression()
lr_model.fit(X_train, y_train)
```

Out[22]:
```
LinearRegression()
```

In [23]:
```
lr_model.fit(X_train, y_train)
y_pred =lr_model.predict(X_test)
```

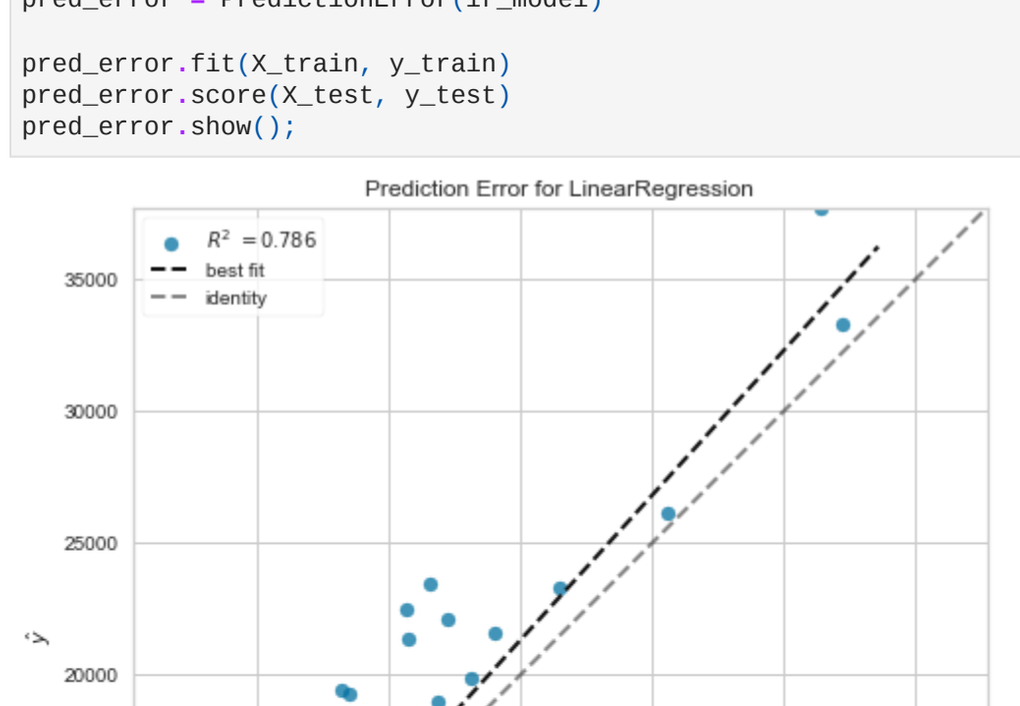
In [24]:
```
pip install yellowbrick
```

```
Requirement already satisfied: yellowbrick in d:\users\namitha cv\anaconda3\lib\site-packages (1.5)
Requirement already satisfied: numpy>=1.16.0 in d:\users\namitha cv\anaconda3\lib\site-packages (from yellowbrick) (1.22.4)
Requirement already satisfied: scipy>=1.0.0 in d:\users\namitha cv\anaconda3\lib\site-packages (from yellowbrick) (1.7.3)
Requirement already satisfied: cycler>=0.10.0 in d:\users\namitha cv\anaconda3\lib\site-packages (from yellowbrick) (0.11.0)
Requirement already satisfied: scikit-learn>=1.0.0 in d:\users\namitha cv\anaconda3\lib\site-packages (from yellowbrick) (1.0.2)
Requirement already satisfied: matplotlib!=3.0.0,>=2.0.2 in d:\users\namitha cv\anaconda3\lib\site-packages (from yellowbrick) (3.5.1)
Requirement already satisfied: pillow>=6.2.0 in d:\users\namitha cv\anaconda3\lib\site-packages (from matplotlib!=3.0.0,>=2.0.2->yellowbrick) (9.0.1)
Requirement already satisfied: python-dateutil>=2.7 in d:\users\namitha cv\anaconda3\lib\site-packages (from matplotlib!=3.0.0,>=2.0.2->yellowbrick) (2.8.2)
Requirement already satisfied: pyparsing>=2.2.1 in d:\users\namitha cv\anaconda3\lib\site-packages (from matplotlib!=3.0.0,>=2.0.2->yellowbrick) (3.0.4)
Requirement already satisfied: kiwisolver>=1.0.1 in d:\users\namitha cv\anaconda3\lib\site-packages (from matplotlib!=3.0.0,>=2.0.2->yellowbrick) (1.3.2)
Requirement already satisfied: fonttools>=4.22.0 in d:\users\namitha cv\anaconda3\lib\site-packages (from matplotlib!=3.0.0,>=2.0.2->yellowbrick) (4.25.0)
Requirement already satisfied: packaging>=20.0 in d:\users\namitha cv\anaconda3\lib\site-packages (from matplotlib!=3.0.0,>=2.0.2->yellowbrick) (21.3)
Requirement already satisfied: six>=1.5 in d:\users\namitha cv\anaconda3\lib\site-packages (from python-dateutil>=2.7->matplotlib!=3.0.0,>=2.0.2->yellowbrick) (1.16.0)
Requirement already satisfied: joblib>=0.11 in d:\users\namitha cv\anaconda3\lib\site-packages (from scikit-learn>=1.0.0->yellowbrick) (1.1.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in d:\users\namitha cv\anaconda3\lib\site-packages (from scikit-learn>=1.0.0->yellowbrick) (2.2.0)
Note: you may need to restart the kernel to use updated packages.
```

In [26]:
```
lr_pred = lr_model.predict(X_test)
mse1 = mean_absolute_error(y_test, lr_pred)
print(f'Mean Absolute Error = {mse1:.2f}')
r2 = r2_score(y_test, lr_pred)
print(f'R-Squared Score = {r2*100:.2f} %')
```

```
Mean Absolute Error = 1992.23
R-Squared Score = 78.63 %
```

In [27]:
```
from yellowbrick.regressor import ResidualsPlot, PredictionError

plt.figure(figsize=(20,8))
pred_error = PredictionError(lr_model)

pred_error.fit(X_train, y_train)
pred_error.score(X_test, y_test)
pred_error.show();
```



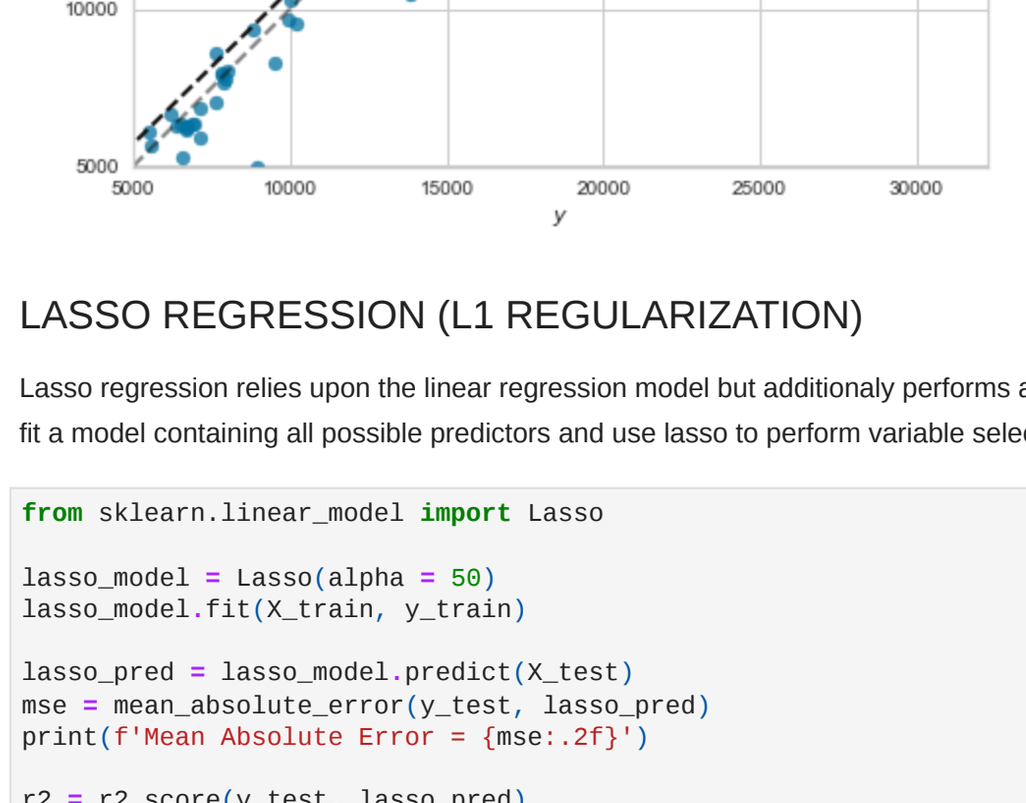
ridge regression(l2 regularization)

ridge is used when you have multiple features and wants to select a better feature correlating to y for that you can use ridge. this can help when features are highly correlated

In [28]:
```
from sklearn.linear_model import Ridge

rdg_model = Ridge(alpha = 50)
rdg_model.fit(X_train, y_train)

rdg_pred = rdg_model.predict(X_test)
mse2 = mean_absolute_error(y_test, rdg_pred)
print(f'Mean Absolute Error = {mse2:.2f}')

r2 = r2_score(y_test, rdg_pred)
print(f'R-Squared Score = {r2*100:.2f} %')
```

```
Mean Absolute Error = 1692.13
R-Squared Score = 84.78 %
```

In [29]:
```
plt.figure(figsize=(20,8))
pred_error = PredictionError(rdg_model)

pred_error.fit(X_train, y_train)
pred_error.score(X_test, y_test)
pred_error.show();
```



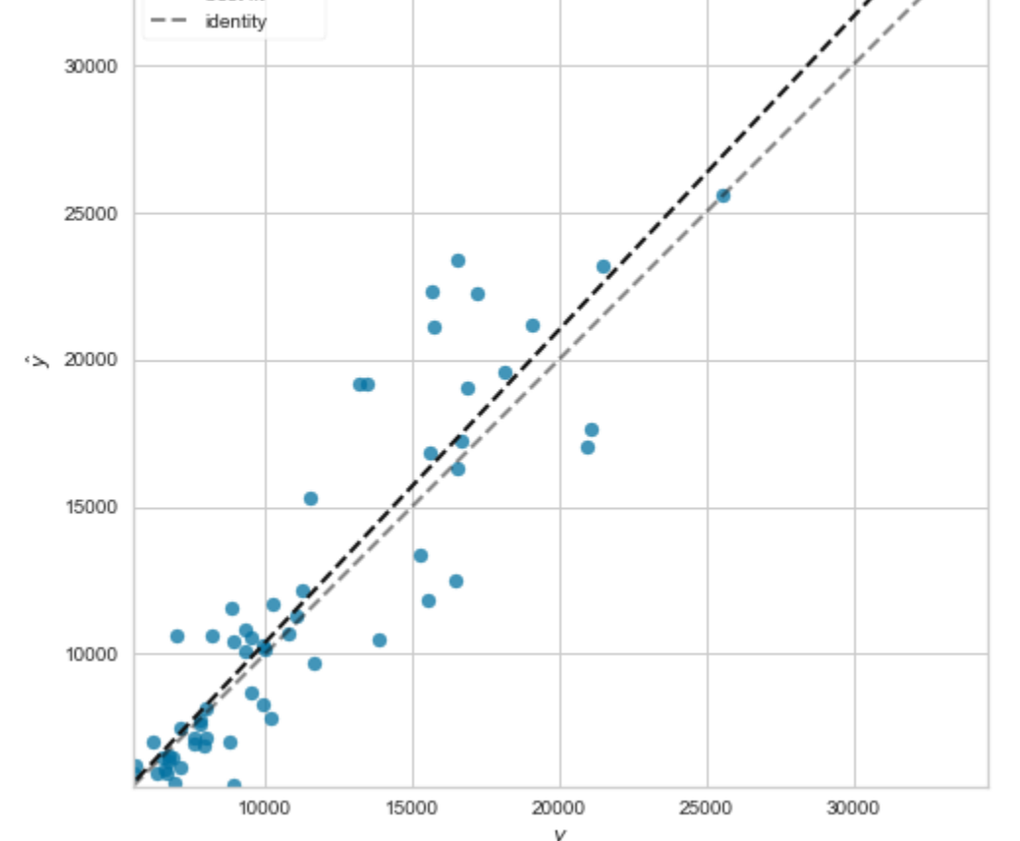
LASSO REGRESSION (L1 REGULARIZATION)

Lasso regression relies upon the linear regression model but additionaly performs a so called L1 regularization, which is a process of introducing additional information in order to prevent overfitting. As a consequence, we can fit a model containing all possible predictors and use lasso to perform variable selection by using a technique that regularizes the coefficient estimates

In [30]:
```
from sklearn.linear_model import Lasso

lasso_model = Lasso(alpha = 50)
lasso_model.fit(X_train, y_train)

lasso_pred = lasso_model.predict(X_test)
mse = mean_absolute_error(y_test, lasso_pred)
print(f'Mean Absolute Error = {mse:.2f}')

r2 = r2_score(y_test, lasso_pred)
print(f'R-Squared Score = {r2*100:.2f} %')
```

```
Mean Absolute Error = 1805.62
R-Squared Score = 81.67 %
```

In [31]:
```
plt.figure(figsize=(20,8))
pred_error = PredictionError(lasso_model)

pred_error.fit(X_train, y_train)
pred_error.score(X_test, y_test)
pred_error.show();
```



We can see that three models are a decent fit but Ridge Regression has better generalization.

In [ ]: