Data Analysis

In [1]:

```
# identification and handling missing values
# Data standardization
# Data normalization
# binning
```

model development & evaluation

importing liabraries

In [2]:

```
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import matplotlib as plt
import matplotlib.pyplot as plt
```

In [3]:

```
# Read the online file by the URL provides above, and assign it to variable "df"
other_path = "https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/DA0101EN/auto.csv"
df = pd.read_csv(other_path, header=None)
```

In [4]:

```
# create headers list
headers = ["symboling","normalized-losses","make","fuel-type","aspiration", "num-of-doors","body-style",
         "drive-wheels","engine-location","wheel-base", "length","width","height","curb-weight","engine-type",
         "num-of-cylinders", "engine-size","fuel-system","bore","stroke","compression-ratio","horsepower",
         "peak-rpm","city-mpg","highway-mpg","price"]
print("headers\n", headers)
```

```
headers
 ['symboling', 'normalized-losses', 'make', 'fuel-type', 'aspiration', 'num-of-doors', 'body-style', 'drive-wheels', 'engin
e-location', 'wheel-base', 'length', 'width', 'height', 'curb-weight', 'engine-type', 'num-of-cylinders', 'engine-size', 'f
uel-system', 'bore', 'stroke', 'compression-ratio', 'horsepower', 'peak-rpm', 'city-mpg', 'highway-mpg', 'price']
```

In [5]:

```
df.columns = headers
```

In [6]:

```
df.columns
```

Out[6]:

```
Index(['symboling', 'normalized-losses', 'make', 'fuel-type', 'aspiration',
       'num-of-doors', 'body-style', 'drive-wheels', 'engine-location',
       'wheel-base', 'length', 'width', 'height', 'curb-weight', 'engine-type',
       'num-of-cylinders', 'engine-size', 'fuel-system', 'bore', 'stroke',
       'compression-ratio', 'horsepower', 'peak-rpm', 'city-mpg',
       'highway-mpg', 'price'],
      dtype='object')
```

In [7]:

```
df
```

Out[7]:

	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	hors
0	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	
1	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	
2	1	?	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	9.0	
3	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	10.0	
4	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.40	8.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
200	-1	95	volvo	gas	std	four	sedan	rwd	front	109.1	...	141	mpfi	3.78	3.15	9.5	
201	-1	95	volvo	gas	turbo	four	sedan	rwd	front	109.1	...	141	mpfi	3.78	3.15	8.7	
202	-1	95	volvo	gas	std	four	sedan	rwd	front	109.1	...	173	mpfi	3.58	2.87	8.8	
203	-1	95	volvo	diesel	turbo	four	sedan	rwd	front	109.1	...	145	idi	3.01	3.40	23.0	
204	-1	95	volvo	gas	turbo	four	sedan	rwd	front	109.1	...	141	mpfi	3.78	3.15	9.5	

205 rows × 26 columns

In [8]:

```
df.shape
```

Out[8]:

```
(205, 26)
```

In [9]:

```
df.replace("?", 'NaN', inplace= True)
```

identify and handling missing values

In [10]:

```
df.head(5)
```

Out[10]:

	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	horsep
0	3	NaN	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	
1	3	NaN	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	
2	1	NaN	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	9.0	
3	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	10.0	
4	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.40	8.0	

5 rows × 26 columns

evaluating for any missing data

In [11]:

```
df_new= df.isnull()
df_new.head(5)
```

Out[11]:

	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
0	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False

5 rows × 26 columns

count missing values in each column

In [12]:

```
df_new.isnull().sum()
```

Out[12]:

```
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
```

In [13]:

```
columns = ["normalized-losses", "bore" , "horsepower" ,"peak-rpm"]

for column in columns:
    avg = df[column].astype('float').mean(axis = 0)
    df[column].replace('NaN' , avg, inplace= True)
```

replace NaN in various columns

In [14]:

```
df["num-of-doors"].value_counts()
```

Out[14]:

```
four    114
two      89
NaN       2
Name: num-of-doors, dtype: int64
```

In [15]:

```
#replace missing by most frequent values:-

df["num-of-doors"].replace('NaN', "four" , inplace= True)
```

In [16]:

```
df["num-of-doors"].value_counts()
```

Out[16]:

```
four    116
two      89
Name: num-of-doors, dtype: int64
```

In [17]:

```
#certain columns drop entire rows
df.dropna(subset=["price"],axis=0, inplace= True)
```

In [18]:

```
df.reset_index(drop = True, inplace= True)
```

In [19]:

```
df.head()
```

Out[19]:

	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	horsep
0	3	122.0	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	
1	3	122.0	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	
2	1	122.0	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	9.0	
3	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	10.0	
4	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.40	8.0	

5 rows × 26 columns

In [20]:

```
df.dtypes
```

Out[20]:

```
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
```

Bring the columns into correct datatypes

In [21]:

```
#as we see,numerical variables should have tpe 'Float' or 'int' & categoricals are in 'object'
```

In [22]:

```
df[["bore", "stroke"]] = df[["bore", "stroke"]].astype("float")
df[["normalized-losses"]] = df[["normalized-losses"]].astype("int")
df[["price"]] = df[["price"]].astype("float")
df[["peak-rpm"]] = df[["peak-rpm"]].astype("float")
```

Data Standardization

convert milesper galoons (mpg) into litres per 100km

In [23]:

```
#formula for unit conversion is Litre/100km = 235/mpg
```

In [24]:

```
#transform mpg to L/100km
df['city-L/100km'] = 235/df['city-mpg']
df['highway-L/100km']= 235/df['highway-mpg']
df[['city-L/100km', 'city-mpg', 'highway-L/100km' ,'highway-mpg' ]].head()
```

Out[24]:

	city-L/100km	city-mpg	highway-L/100km	highway-mpg
0	11.190476	21	8.703704	27
1	11.190476	21	8.703704	27
2	12.368421	19	9.038462	26
3	9.791667	24	7.833333	30
4	13.055556	18	10.681818	22

Data Normalization

In [25]:

```
#replace original values by (original values/max value)

df["length"] = df["length"]/df["length"].max()
df["width"] = df["width"]/df["width"].max()
df["height"] = df["height"]/df["height"].max()
df[["length","width","height"]].head()
```

Out[25]:

	length	width	height
0	0.811148	0.886584	0.816054
1	0.811148	0.886584	0.816054
2	0.822681	0.905947	0.876254
3	0.848630	0.915629	0.908027
4	0.848630	0.918396	0.908027

Binning

convert data to correct format

In [26]:

```
df["horsepower"]= df["horsepower"].astype(float, copy=True)
```

In [27]:

```
band_width = (max(df["horsepower"])-min(df["horsepower"]))/4
band_width
```

Out[27]:

```
60.0
```

In [28]:

```
#we can build a bin array with min to max value with bandwith
```

In [29]:

```
bins = np.arange(min(df["horsepower"]),max(df["horsepower"]), band_width)
bins
```

Out[29]:

```
array([ 48., 108., 168., 228.])
```

In [30]:

```
# we set group names"
```

In [31]:

```
group_names = ["Low", "Medium", "High"]
```

In [32]:

```
df["horsepower_binned"]= pd.cut(df["horsepower"], bins, labels=group_names,include_lowest= True)
df[["horsepower", "horsepower_binned"]].head(20)
```

Out[32]:

	horsepower	horsepower_binned
0	111.0	Medium
1	111.0	Medium
2	154.0	Medium
3	102.0	Low
4	115.0	Medium
5	110.0	Medium
6	110.0	Medium
7	110.0	Medium
8	140.0	Medium
9	160.0	Medium
10	101.0	Low
11	101.0	Low
12	121.0	Medium
13	121.0	Medium
14	121.0	Medium
15	182.0	High
16	182.0	High
17	182.0	High
18	48.0	Low
19	70.0	Low

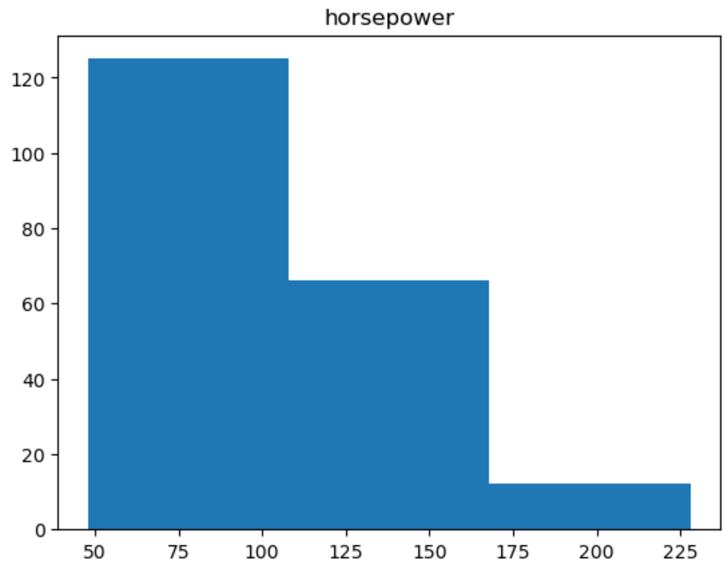
bins visualization

In [33]:

```
plt.title("horsepower")
plt.hist(df[["horsepower"]], bins=[48., 108., 168., 228.])
```

Out[33]:

```
(array([125.,  66.,  12.]),
 array([ 48., 108., 168., 228.]),
 <BarContainer object of 3 artists>)
```



one hot encoding

In [34]:

```
# as "fuel type" column has two values 'GAS' & 'disel' convert it into indicator variable
```

In [35]:

```
df["fuel-type"]
```

Out[35]:

```
0         gas
1         gas
2         gas
3         gas
4         gas
        ...  
200       gas
201       gas
202       gas
203    diesel
204       gas
Name: fuel-type, Length: 205, dtype: object
```

In [36]:

```
dummy_variable_1= pd.get_dummies(df["fuel-type"])
dummy_variable_1
```

Out[36]:

	diesel	gas
0	0	1
1	0	1
2	0	1
3	0	1
4	0	1
...	...	...
200	0	1
201	0	1
202	0	1
203	1	0
204	0	1

205 rows × 2 columns

In [37]:

```
dummy_variable_1= pd.get_dummies(df["fuel-type"])
dummy_variable_1.rename(columns = {'gas':'fuel-type-gas', 'diesel':'fuel-type-diesel'}, inplace=True)
dummy_variable_1.head()
```

Out[37]:

	fuel-type-diesel	fuel-type-gas
0	0	1
1	0	1
2	0	1
3	0	1
4	0	1

In [38]:

```
# we now have values 0 to represent 'GAS' & 1 to represent 'DIESEL'in column name"fuel-type". we will now insert this ew columns to o
```

In [39]:

```
#merge dataframe
df = pd.concat([df, dummy_variable_1], axis=1)

#drop original column 'fuel-type' from df
df.drop('fuel-type', axis=1, inplace=True)
```

In [40]:

```
df.head()
```

Out[40]:

	symboling	normalized-losses	make	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	length	...	horsepower	peak-rpm	city-mpg	highway-mpg	price	L
0	3	122	alfa-romero	std	two	convertible	rwd	front	88.6	0.811148	...	111.0	5000.0	21	27	13495.0	11.
1	3	122	alfa-romero	std	two	convertible	rwd	front	88.6	0.811148	...	111.0	5000.0	21	27	16500.0	11.
2	1	122	alfa-romero	std	two	hatchback	rwd	front	94.5	0.822681	...	154.0	5000.0	19	26	16500.0	12.
3	2	164	audi	std	four	sedan	fwd	front	99.8	0.848630	...	102.0	5500.0	24	30	13950.0	9.
4	2	164	audi	std	four	sedan	4wd	front	99.4	0.848630	...	115.0	5500.0	18	22	17450.0	13.

5 rows × 30 columns

In [41]:

```python
#repeat same to create dummy variables for "aspiration" features
```

In [42]:

```python
dummy_variable_2= pd.get_dummies(df['aspiration'])
dummy_variable_2.rename(columns={'std':'aspiration-std', 'turbo':'aspiration-turbo'}, inplace=True)
df= pd.concat([df,dummy_variable_2],axis=1)
df.drop('aspiration',axis = 1 , inplace=True)
```

In [43]:

```python
df.to_csv('clean_df.csv')
```

In [ ]:

```python

```

findind correlation between variables

In [44]:

```python
df.corr()
```

Out[44]:

	symboling	normalized-losses	wheel-base	length	width	height	curb-weight	engine-size	bore	stroke	...	peak-rpm	city-mpg	high
symboling	1.000000	0.465190	-0.531954	-0.357612	-0.232919	-0.541038	-0.227691	-0.105790	-0.130083	-0.008965	...	0.273679	-0.035823	0.03
normalized-losses	0.465190	1.000000	-0.056518	0.019209	0.084195	-0.370706	0.097785	0.110997	-0.029266	0.055363	...	0.237748	-0.218749	-0.17
wheel-base	-0.531954	-0.056518	1.000000	0.874587	0.795144	0.589435	0.776386	0.569329	0.488760	0.161477	...	-0.360704	-0.470414	-0.54
length	-0.357612	0.019209	0.874587	1.000000	0.841118	0.491029	0.877728	0.683360	0.606462	0.129739	...	-0.287031	-0.670909	-0.70
width	-0.232919	0.084195	0.795144	0.841118	1.000000	0.279210	0.867032	0.735433	0.559152	0.182956	...	-0.219859	-0.642704	-0.67
height	-0.541038	-0.370706	0.589435	0.491029	0.279210	1.000000	0.295572	0.067149	0.171101	-0.056999	...	-0.320602	-0.048640	-0.10
curb-weight	-0.227691	0.097785	0.776386	0.877728	0.867032	0.295572	1.000000	0.850594	0.648485	0.168929	...	-0.266283	-0.757414	-0.79
engine-size	-0.105790	0.110997	0.569329	0.683360	0.735433	0.067149	0.850594	1.000000	0.583798	0.206675	...	-0.244599	-0.653658	-0.67
bore	-0.130083	-0.029266	0.488760	0.606462	0.559152	0.171101	0.648485	0.583798	1.000000	-0.055909	...	-0.254761	-0.584508	-0.58
stroke	-0.008965	0.055363	0.161477	0.129739	0.182956	-0.056999	0.168929	0.206675	-0.055909	1.000000	...	-0.069212	-0.042906	-0.04
compression-ratio	-0.178515	-0.114525	0.249786	0.158414	0.181129	0.261214	0.151362	0.028971	0.005201	0.186170	...	-0.435936	0.324701	0.26
horsepower	0.071389	0.203434	0.351957	0.554434	0.642195	-0.110137	0.750968	0.810713	0.575737	0.088400	...	0.130971	-0.803162	-0.77
peak-rpm	0.273679	0.237748	-0.360704	-0.287031	-0.219859	-0.320602	-0.266283	-0.244599	-0.254761	-0.069212	...	1.000000	-0.113723	-0.05
city-mpg	-0.035823	-0.218749	-0.470414	-0.670909	-0.642704	-0.048640	-0.757414	-0.653658	-0.584508	-0.042906	...	-0.113723	1.000000	0.97
highway-mpg	0.034606	-0.178221	-0.544082	-0.704662	-0.677218	-0.107358	-0.797465	-0.677470	-0.586992	-0.044528	...	-0.054257	0.971337	1.00
price	-0.082391	0.133999	0.584642	0.690628	0.751265	0.135486	0.834415	0.872335	0.543155	0.082310	...	-0.101616	-0.686571	-0.70
city-L/100km	0.063165	0.232682	0.474040	0.659165	0.682850	-0.002333	0.791911	0.744952	0.555960	0.043677	...	0.120653	-0.950493	-0.92
highway-L/100km	-0.030190	0.178527	0.578128	0.711597	0.728044	0.085892	0.836742	0.777077	0.551943	0.056222	...	0.016127	-0.908439	-0.95
fuel-type-diesel	-0.194311	-0.101437	0.308346	0.212679	0.233880	0.284631	0.217275	0.069594	0.054457	0.242081	...	-0.477060	0.255963	0.19
fuel-type-gas	0.194311	0.101437	-0.308346	-0.212679	-0.233880	-0.284631	-0.217275	-0.069594	-0.054457	-0.242081	...	0.477060	-0.255963	-0.19
aspiration-std	0.059866	0.006823	-0.257611	-0.234539	-0.300567	-0.087311	-0.324902	-0.108217	-0.212623	-0.223460	...	0.183629	0.202362	0.25
aspiration-turbo	-0.059866	-0.006823	0.257611	0.234539	0.300567	0.087311	0.324902	0.108217	0.212623	0.223460	...	-0.183629	-0.202362	-0.25

22 rows × 22 columns

In [45]:

```python
df[['price','bore', 'stroke', 'compression-ratio','horsepower']].corr()["price"]
```

Out[45]:

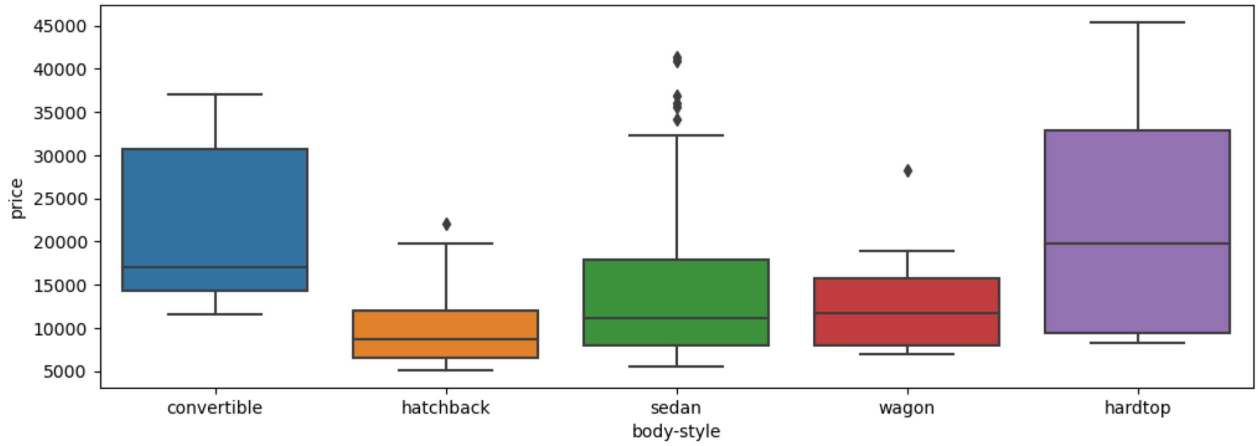
```
price                1.000000
bore                 0.543155
stroke               0.082310
compression-ratio    0.071107
horsepower           0.809575
Name: price, dtype: float64
```

In [47]:

```
import seaborn as sns
plt.figure(figsize=(12,4))
sns.boxplot(x= "body-style", y = "price", data=df )
```

Out[47]:

```
<AxesSubplot:xlabel='body-style', ylabel='price'>
```



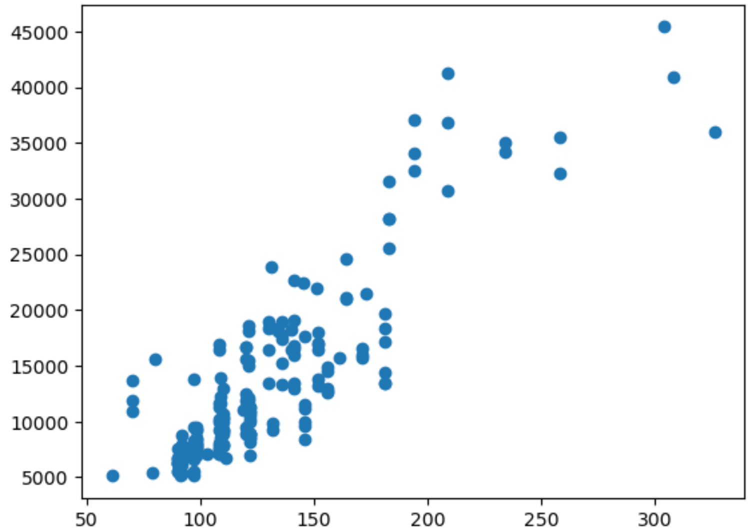
In [48]:

```
#################
```

find scatterplot of "engine-size" & "price"

In [49]:

```
plt.scatter(x="engine-size", y="price", data=df)
plt.show()
```

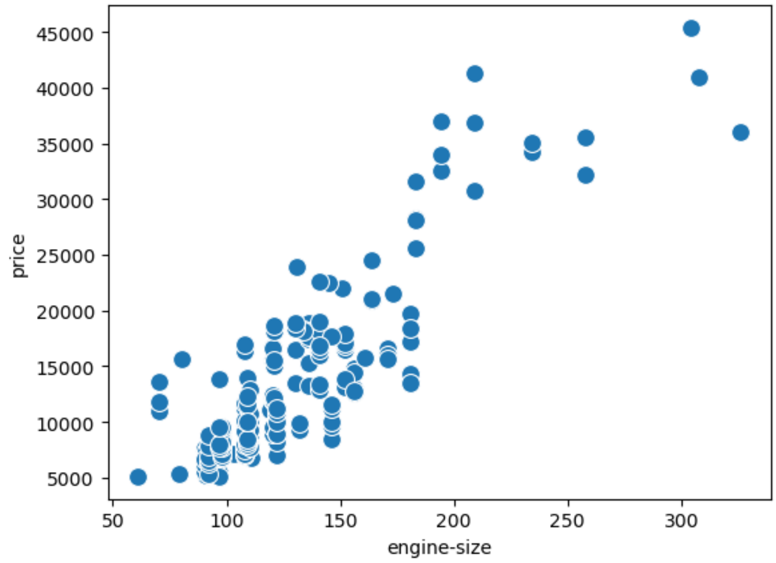


In [50]:

```
sns.scatterplot(x= "engine-size", y= "price", data=df, s=100)
```

Out[50]:

```
<AxesSubplot:xlabel='engine-size', ylabel='price'>
```

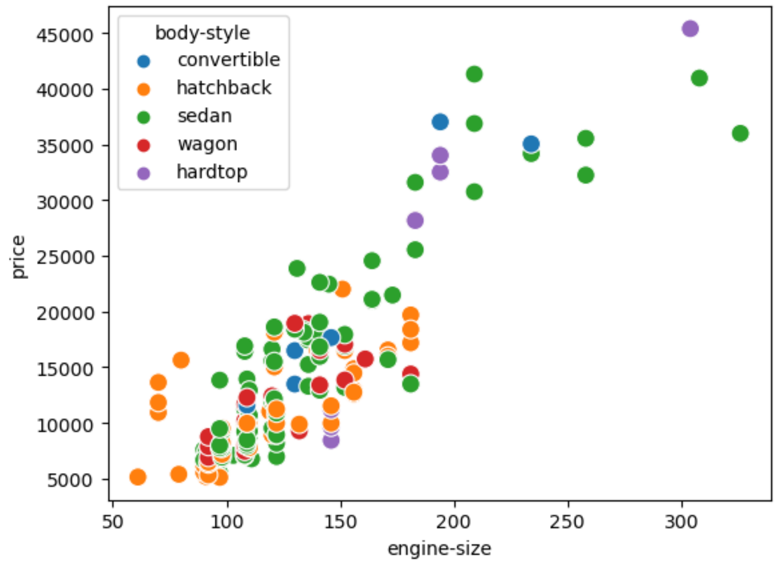


In [51]:

```
colors= ["r","g","y","b","o"]
sns.scatterplot(x= "engine-size", y= "price", data=df, s=100, hue="body-style", cmap="rainbow")
```

Out[51]:

```
<AxesSubplot:xlabel='engine-size', ylabel='price'>
```



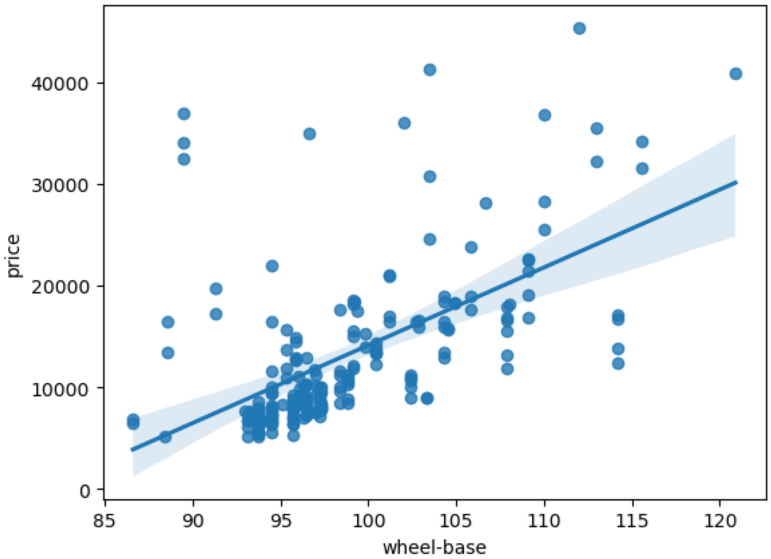
find which variable is suitable for predictor of price

In [52]:

```
#regplots - regression plot
```

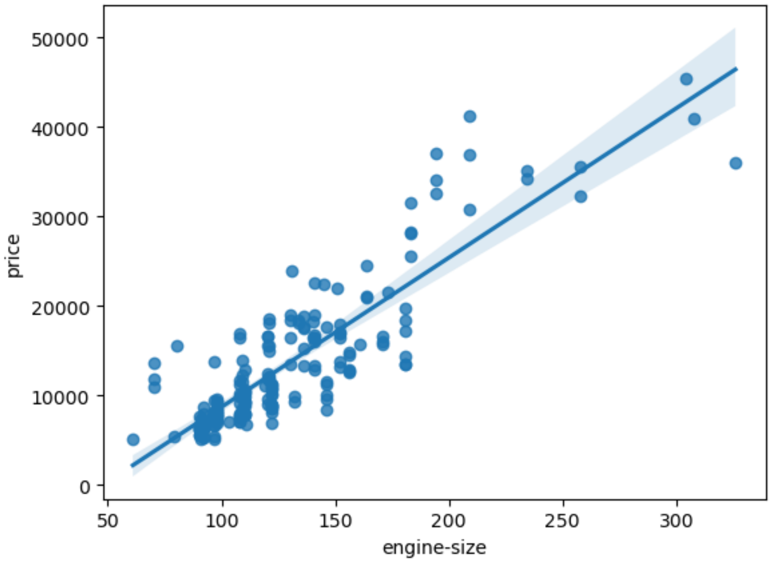
In [53]:

```
sns.regplot(x= "wheel-base", y="price", data=df)
plt.show()
```



In [54]:

```
#engine-size
sns.regplot(x= "engine-size", y="price", data=df)
plt.show()
```



In [55]:

```
df.describe()
```

Out[55]:

	symboling	normalized-losses	wheel-base	length	width	height	curb-weight	engine-size	bore	stroke	...	peak-rpm	city-
count	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	201.000000	...	205.000000	205.00
mean	0.834146	122.000000	98.756585	0.836373	0.911588	0.898409	2555.565854	126.907317	3.329751	3.255423	...	5125.369458	25.21
std	1.245307	31.681008	6.021776	0.059285	0.029671	0.040862	520.680204	41.642693	0.270844	0.316717	...	476.979093	6.54
min	-2.000000	65.000000	86.600000	0.678039	0.834025	0.799331	1488.000000	61.000000	2.540000	2.070000	...	4150.000000	13.00
25%	0.000000	101.000000	94.500000	0.799135	0.886584	0.869565	2145.000000	97.000000	3.150000	3.110000	...	4800.000000	19.00
50%	1.000000	122.000000	97.000000	0.832292	0.905947	0.904682	2414.000000	120.000000	3.310000	3.290000	...	5200.000000	24.00
75%	2.000000	137.000000	102.400000	0.879865	0.925311	0.928094	2935.000000	141.000000	3.580000	3.410000	...	5500.000000	30.00
max	3.000000	256.000000	120.900000	1.000000	1.000000	1.000000	4066.000000	326.000000	3.940000	4.170000	...	6600.000000	49.00

8 rows × 22 columns

In [56]:

```
df.describe(include=["object"])
```

Out[56]:

	make	num-of-doors	body-style	drive-wheels	engine-location	engine-type	num-of-cylinders	fuel-system
count	205	205	205	205	205	205	205	205
unique	22	2	5	3	2	7	7	8
top	toyota	four	sedan	fwd	front	ohc	four	mpfi
freq	32	116	96	120	202	148	159	94

In [57]:

```
#value_countis a goodway of understanding how mwny units of each characterstic/variables we have.
```

In [58]:

```
df["drive-wheels"].value_counts().to_frame()
```

Out[58]:

	drive-wheels
fwd	120
rwd	76
4wd	9

In [59]:

```
drive_wheels_counts = df['drive-wheels'].value_counts().to_frame()
drive_wheels_counts.rename(columns={'drive-wheels':'value_counts'}, inplace=True)
drive_wheels_counts.index.name='drive-wheels'
drive_wheels_counts
```

Out[59]:

	value_counts
drive-wheels	
fwd	120
rwd	76
4wd	9

In [60]:

```
engine_loc_counts= df['engine-location'].value_counts().to_frame()
engine_loc_counts.rename(columns={'engine-location':'value_counts'}, inplace=True)
engine_loc_counts.index.name='engine-location'
engine_loc_counts
```

Out[60]:

	value_counts
engine-location	
front	202
rear	3

Draw heatmap between correlated features

In [61]:

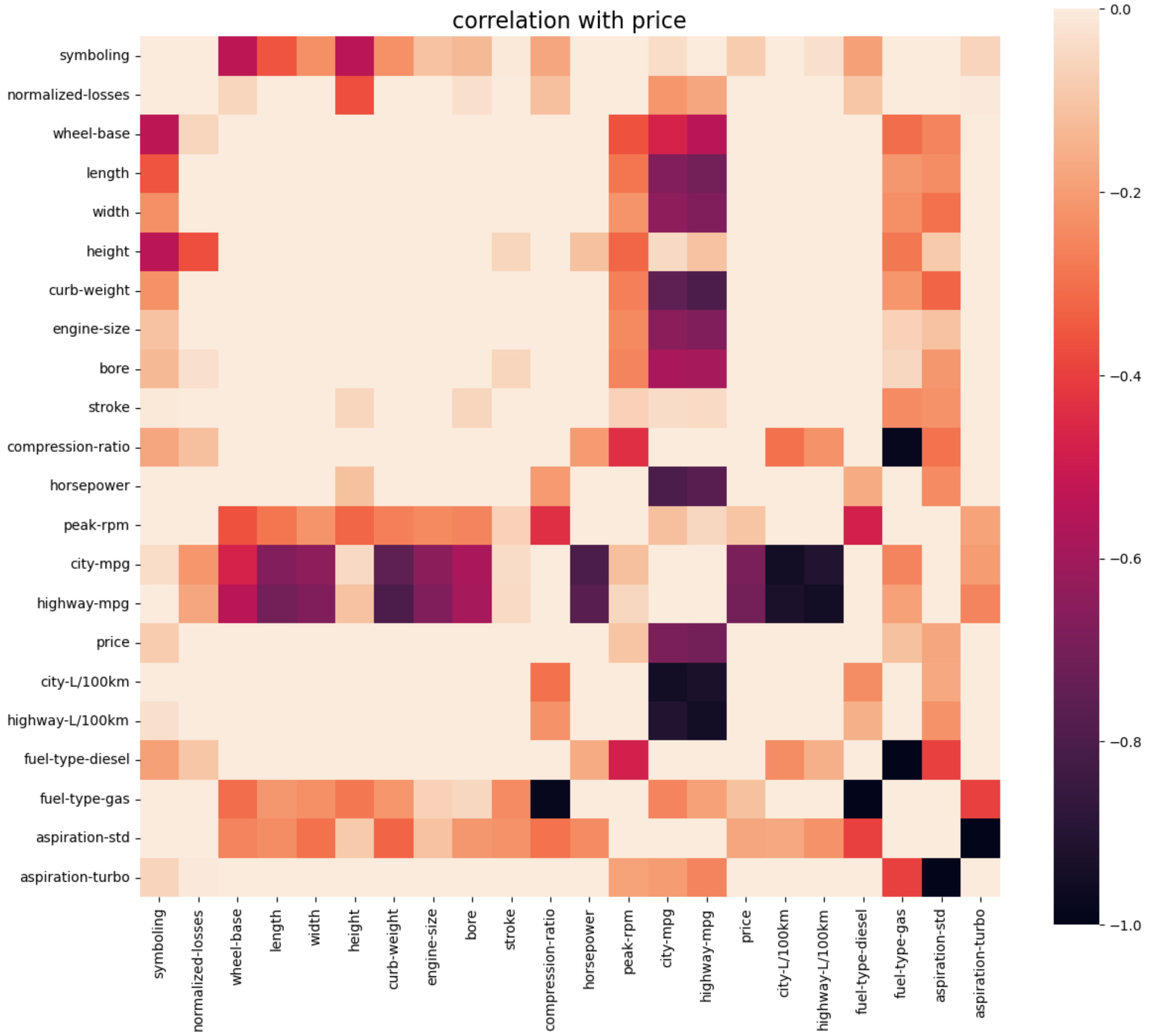
```
#correlation= df.corr()    #allready is there
```

In [62]:

```
f,ax = plt.subplots(figsize = (14,12))
plt.title('correlation with price' , y=1, size=16)

sns.heatmap(df.corr() , square=True , vmax=0)
```

Out[62]:

```
<AxesSubplot:title={'center':'correlation with price'}>
```



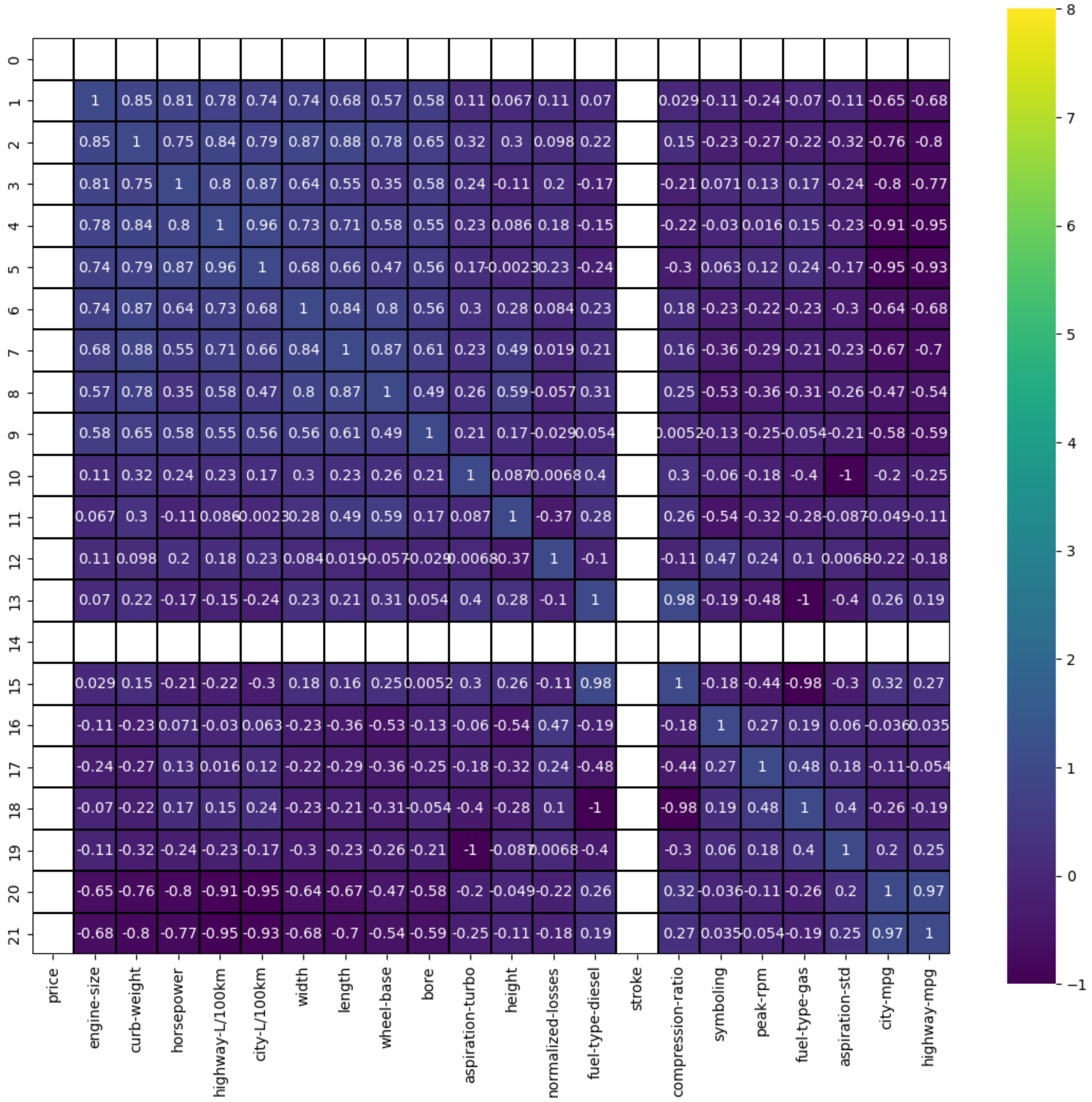
In [63]:

```
k = 22
cols = df.corr().nlargest(k, 'price')['price'].index
print(cols)
cm= np.corrcoef(df[cols].values.T)
f, ax = plt.subplots(figsize=(14,12))
sns.heatmap(cm,vmax=8, linewidth=0.01,square=True,annot= True,cmap='viridis' , linecolor='black', xticklabels= cols.values)
```

```
Index(['price', 'engine-size', 'curb-weight', 'horsepower', 'highway-L/100km',
       'city-L/100km', 'width', 'length', 'wheel-base', 'bore',
       'aspiration-turbo', 'height', 'normalized-losses', 'fuel-type-diesel',
       'stroke', 'compression-ratio', 'symboling', 'peak-rpm', 'fuel-type-gas',
       'aspiration-std', 'city-mpg', 'highway-mpg'],
      dtype='object')
```

Out[63]:

```
<AxesSubplot:>
```



find correlation of predictor variables with price using scatter-plots:-

In [64]:

```
#scatterplot between most correlated variables
```

In [65]:

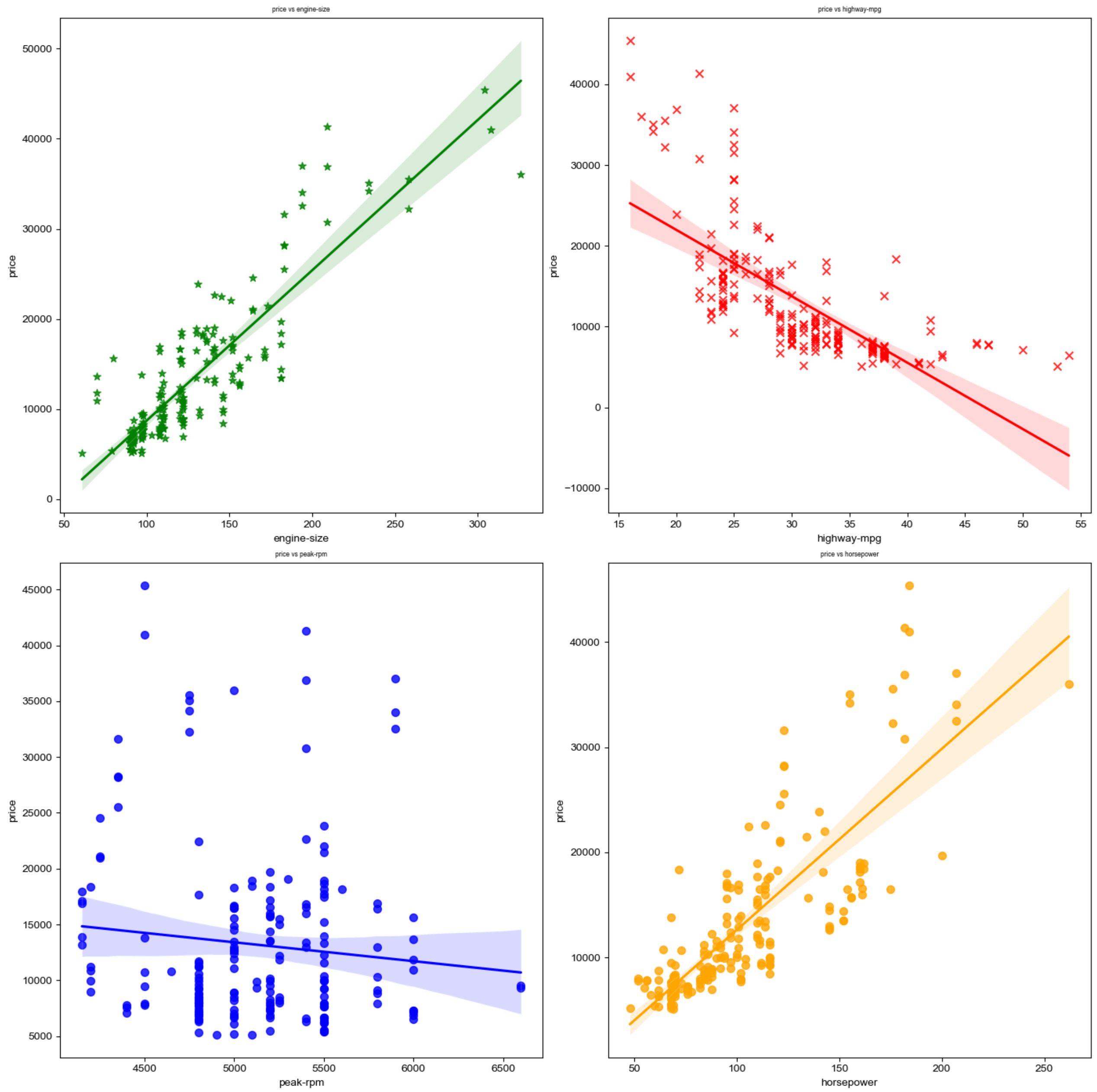
```
df[['price','engine-location', 'engine-size','peak-rpm','highway-mpg' ,'horsepower']].corr()["price"]
```

Out[65]:

```
price          1.000000
engine-size    0.872335
peak-rpm      -0.101616
highway-mpg   -0.704692
horsepower     0.809575
Name: price, dtype: float64
```

In [66]:

```
 fig = plt.figure(figsize=(14,14))#create figure

ax0= fig.add_subplot(2,2,1)
ax1= fig.add_subplot(2,2,2)
ax2= fig.add_subplot(2,2,3)
ax3= fig.add_subplot(2,2,4)



sns.set(font_scale= 0.5)

sns.regplot(x= 'engine-size', y='price' , data=df,color='green',marker='*',scatter_kws={'s': 50},  ax=ax0)
ax0.set_title('price vs engine-size')

sns.regplot(x= 'highway-mpg' , y='price', data=df,color='red',marker='x',scatter_kws={'s': 50}, ax=ax1)
ax1.set_title('price vs highway-mpg')

sns.regplot(x= 'peak-rpm' , y='price', data=df,color='blue',marker= 'o',scatter_kws={'s': 50}, ax=ax2)
ax2.set_title('price vs peak-rpm')

sns.regplot(x= 'horsepower' , y='price', data=df, color='orange',marker= 'o',scatter_kws={'s': 50}, ax=ax3)
ax3.set_title('price vs horsepower')



fig.tight_layout()

plt.show()


```



In [ ]:

```
```

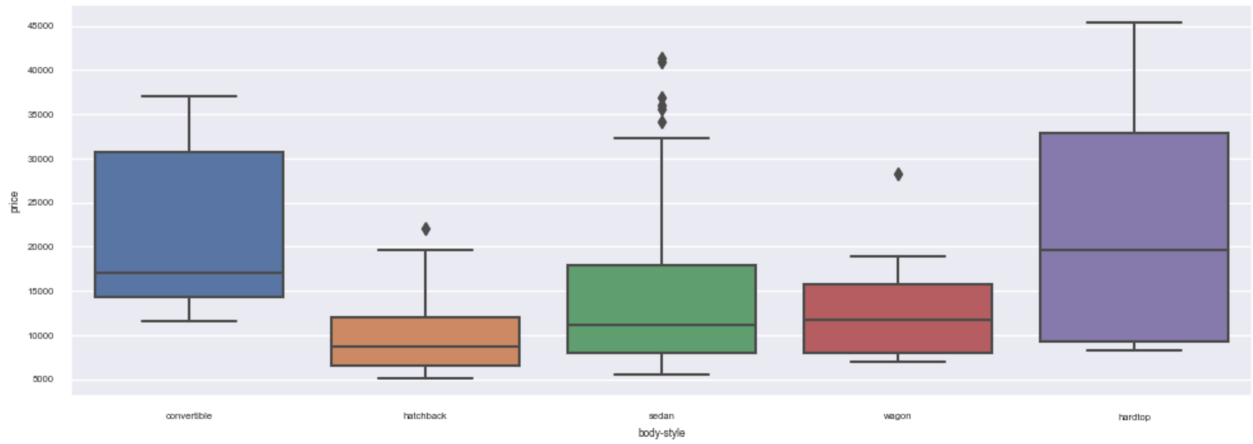
relationship between 'body-style' & 'price' using boxplot

In [67]:

```
import seaborn as sns
plt.figure(figsize=(12,4))
sns.boxplot(x= "body-style", y = "price", data=df )
```

Out[67]:

```
<AxesSubplot:xlabel='body-style', ylabel='price'>
```

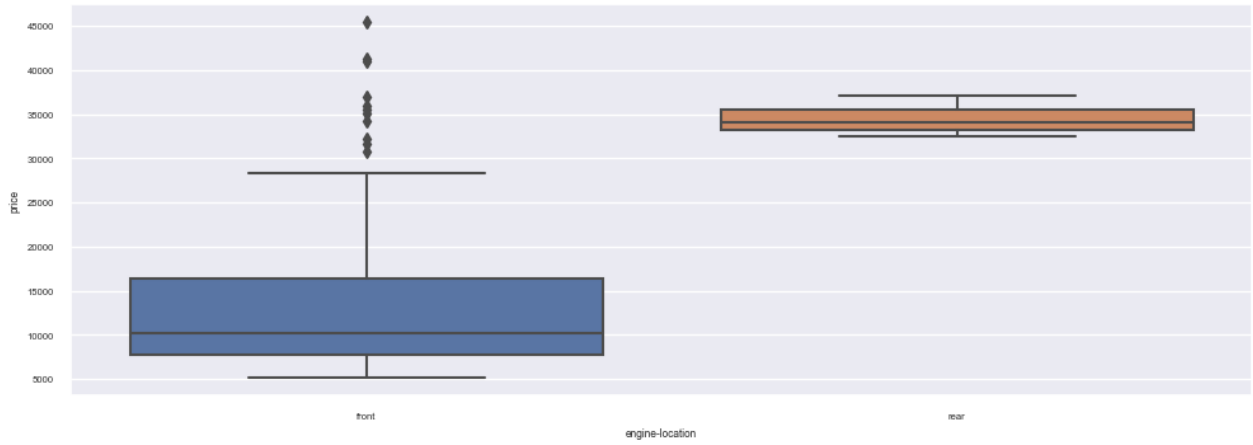


In [68]:

```
import seaborn as sns
plt.figure(figsize=(12,4))
sns.boxplot(x= "engine-location", y = "price", data=df )
```

Out[68]:

```
<AxesSubplot:xlabel='engine-location', ylabel='price'>
```

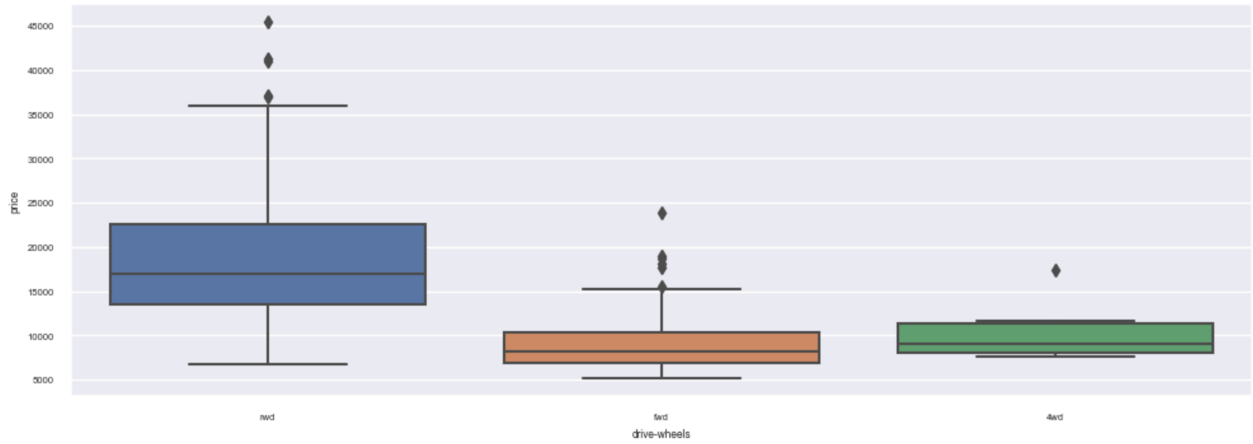


In [69]:

```
import seaborn as sns
plt.figure(figsize=(12,4))
sns.boxplot(x= "drive-wheels", y = "price", data=df )
```

Out[69]:

```
<AxesSubplot:xlabel='drive-wheels', ylabel='price'>
```



using'groupby'function to finf avg price of car based on 'body-style':-

In [70]:

```
df['drive-wheels'].unique()
```

Out[70]:

```
array(['rwd', 'fwd', '4wd'], dtype=object)
```

In [71]:

```
df_group= df[['drive-wheels', 'body-style', 'price']]
df_group_result = df_group.groupby(['drive-wheels', 'body-style'], as_index = False).mean()
df_group_result
```

Out[71]:

	drive-wheels	body-style	price
0	4wd	hatchback	7603.000000
1	4wd	sedan	12647.333333
2	4wd	wagon	9095.750000
3	fwd	convertible	11595.000000
4	fwd	hardtop	8249.000000
5	fwd	hatchback	8396.387755
6	fwd	sedan	9811.800000
7	fwd	wagon	9997.333333
8	rwd	convertible	23949.600000
9	rwd	hardtop	24202.714286
10	rwd	hatchback	14337.777778
11	rwd	sedan	21711.833333
12	rwd	wagon	16994.222222

In [72]:

```
#from groupby we can se multiple variables.
```

In [73]:

```
df_group= df[['engine-size', 'body-style', 'price']]
df_group_result = df_group.groupby(['engine-size', 'body-style'], as_index = False).mean()
df_group_result
```

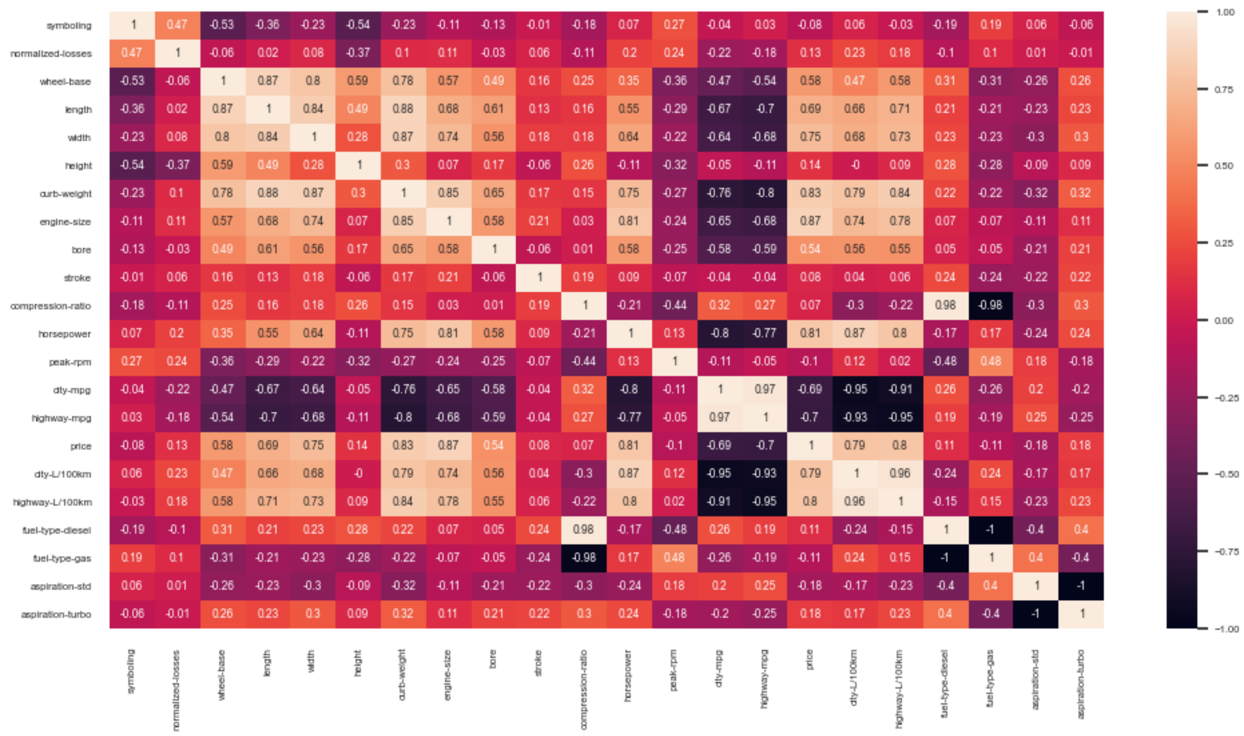
Out[73]:

	engine-size	body-style	price
0	61	hatchback	5151.000000
1	70	hatchback	12145.000000
2	79	hatchback	5399.000000
3	80	hatchback	15645.000000
4	90	hatchback	6045.666667
...	...	...	...
76	234	sedan	34184.000000
77	258	sedan	33900.000000
78	304	hardtop	45400.000000
79	308	sedan	40960.000000
80	326	sedan	36000.000000

81 rows × 3 columns

In [74]:

```
plt.figure(figsize=(12,6))
plot= sns.heatmap(df.corr().round(2), annot = True)
```



Correlation and Causation Analysis

In [75]:

```
#correlation: means measure of extent of interdependance between variables.
#causation: means relationship between cause andeffect between two vaiables.
```

In [76]:

```
from scipy import stats
```

In [77]:

```
df = df.replace([np.inf, -np.inf], np.nan)
df = df.dropna()
df = df.reset_index()
```

In [78]:

```
pearson_coef,p_value = stats.pearsonr(df['wheel-base'], df['price'])
print("The PeasonR Coeffiicient for wheel-base vs price is", pearson_coef, "with a p-value of p=", p_value)
```

```
The PeasonR Coeffiicient for wheel-base vs price is 0.591955763054654 with a p-value of p= 6.41333881539141e-20
```

In [79]:

```

pearson_coef,p_value = stats.pearsonr(df['length'], df['price'])
print("The PeasonR Coeffiicient for length vs price is", pearson_coef, "with a p-value of p=", p_value)
pearson_coef,p_value = stats.pearsonr(df['width'], df['price'])
print("The PeasonR Coeffiicient for width vs price is", pearson_coef, "with a p-value of p=", p_value)
pearson_coef,p_value = stats.pearsonr(df['curb-weight'], df['price'])
print("The PeasonR Coeffiicient for curb-weight vs price is", pearson_coef, "with a p-value of p=", p_value)
pearson_coef,p_value = stats.pearsonr(df['horsepower'], df['price'])
print("The PeasonR Coeffiicient for horsepower vs price is", pearson_coef, "with a p-value of p=", p_value)
pearson_coef,p_value = stats.pearsonr(df['engine-size'], df['price'])
print("The PeasonR Coeffiicient for engine-size vs price is", pearson_coef, "with a p-value of p=", p_value)
pearson_coef,p_value = stats.pearsonr(df['bore'], df['price'])
print("The PeasonR Coeffiicient for bore vs price is", pearson_coef, "with a p-value of p=", p_value)
pearson_coef,p_value = stats.pearsonr(df['city-mpg'], df['price'])
print("The PeasonR Coeffiicient for city-mpg vs price is", pearson_coef, "with a p-value of p=", p_value)
pearson_coef,p_value = stats.pearsonr(df['highway-mpg'], df['price'])
print("The PeasonR Coeffiicient for highway-mpg vs price is", pearson_coef, "with a p-value of p=", p_value)
```

```
The PeasonR Coeffiicient for length vs price is 0.6894657962054989 with a p-value of p= 5.531750161103238e-29
The PeasonR Coeffiicient for width vs price is 0.7441763798094059 with a p-value of p= 7.736546677140488e-36
The PeasonR Coeffiicient for curb-weight vs price is 0.8284829891299339 with a p-value of p= 9.745571059120956e-51
The PeasonR Coeffiicient for horsepower vs price is 0.8020402807928606 with a p-value of p= 2.68640466763373e-45
The PeasonR Coeffiicient for engine-size vs price is 0.8892649648855933 with a p-value of p= 8.015344864946905e-68
The PeasonR Coeffiicient for bore vs price is 0.5443747906183409 with a p-value of p= 1.623426821501105e-16
The PeasonR Coeffiicient for city-mpg vs price is -0.6925501912683503 with a p-value of p= 2.4962452990006994e-29
The PeasonR Coeffiicient for highway-mpg vs price is -0.7074662427380923 with a p-value of p= 4.6006254911951 27e-31
```

In [ ]:

```

```

df.columns

In [80]:

```
df.columns
```

Out[80]:

```
Index(['index', 'symboling', 'normalized-losses', 'make', 'num-of-doors',
       'body-style', 'drive-wheels', 'engine-location', 'wheel-base', 'length',
       'width', 'height', 'curb-weight', 'engine-type', 'num-of-cylinders',
       'engine-size', 'fuel-system', 'bore', 'stroke', 'compression-ratio',
       'horsepower', 'peak-rpm', 'city-mpg', 'highway-mpg', 'price',
       'city-L/100km', 'highway-L/100km', 'horsepower_binned',
       'fuel-type-diesel', 'fuel-type-gas', 'aspiration-std',
       'aspiration-turbo'],
      dtype='object')
```

ANOVA: Analysis of Variance

In [87]:

```
df_group = df[['drive-wheels', 'price']]
groups= df_group[['drive-wheels', 'price']].groupby(['drive-wheels'])
groups.head(2)
```

Out[87]:

	drive-wheels	price
0	rwd	13495.0
1	rwd	16500.0
3	fwd	13950.0
4	4wd	17450.0
5	fwd	15250.0
131	4wd	7603.0

In [88]:

```
groups.get_group('4wd')['price']
```

Out[88]:

```
4      17450.0
131     7603.0
135     9233.0
136    11259.0
139     8013.0
140    11694.0
145     7898.0
146     8778.0
Name: price, dtype: float64
```

In [89]:

```
# ANOVA
f_val, p_val = stats.f_oneway(groups.get_group('fwd')['price'], groups.get_group('rwd')['price'], groups.get_group('4wd')['price'])

print( "ANOVA results: F=", f_val, ", P =", p_val)
```

```
ANOVA results: F= 69.54140581577971 , P = 1.80303006824333396e-23
```

In [90]:

```
# this f_value suggest that mean price value varies a lot.
```

Separately: fwd and rwd

In [91]:

```
f_val, p_val = stats.f_oneway(groups.get_group('fwd')['price'], groups.get_group('rwd')['price'])

print( "ANOVA results: F=", f_val, ", P =", p_val )
```

```
ANOVA results: F= 133.68058112336752 , P = 1.1991081909665041e-23
```

In [92]:

```
# 4wd and rwd
```

In [93]:

```
f_val, p_val = stats.f_oneway(groups.get_group('4wd')['price'], groups.get_group('rwd')['price'])

print( "ANOVA results: F=", f_val, ", P =", p_val)
```

```
ANOVA results: F= 8.918937140036293 , P = 0.003796935906545933
```

In [94]:

```
# 4wd and fwd
```

In [96]:

```
f_val, p_val = stats.f_oneway(groups.get_group('4wd')['price'], groups.get_group('fwd')['price'])

print("ANOVA results: F=", f_val, ", P =", p_val)
```

```
ANOVA results: F= 0.665465750252303 , P = 0.41620116697845666
```

In [97]:

```
at main variation of price mean values is in fwd and rwd groups.so even inside feature drive-wheels,these two groups are more importanat.
```

Conclusion: Important Variables

We now have a better idea of what our data looks like and which variables are important to take into account when predicting the car price. We have narrowed it down to the following variables:

Continuous numerical variables:

Length Width Curb-weight Engine-size Horsepower City-mpg Highway-mpg Wheel-base Bore Categorical variables:

Drive-wheels As we now move into building machine learning models to automate our analysis, feeding the model with variables that meaningfully affect our target variable will improve our model's prediction performance.

Model Development and Evaluation

In [98]:

```
#model development using predictor variable identified as important in data analysis
```

In [99]:

```
df_new= df[['horsepower', 'curb-weight', 'engine-size', 'highway-mpg','bore', 'wheel-base','city-mpg', 'length', 'width']]
```

splitting data into training n testing

In [100]:

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(df_new , df['price'], test_size=0.2, random_state=1)
```

In [101]:

```
x_train.shape
```

Out[101]:

```
(156, 9)
```

In [102]:

```
x_test.shape
```

Out[102]:

```
(40, 9)
```

In [103]:

```
y_train.shape
```

Out[103]:

```
(156,)
```

In [104]:

```
y_test.shape
```

Out[104]:

```
(40,)
```

Multiple LinearRegression

In [105]:

```
ln = LinearRegression()
ln.fit(x_train, y_train)
```

Out[105]:

```
▾ LinearRegression
LinearRegression()
```

multiple linear regression evaluation

In [107]:

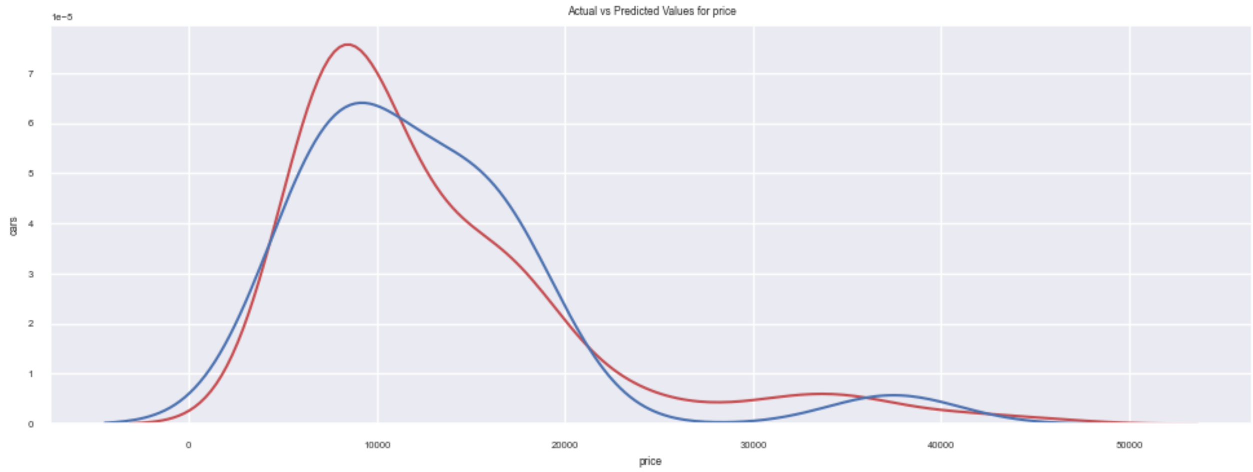
```
print("The R_squared value for Multiple Linear Regression Model is:" , ln.score(x_test,y_test))
```

```
The R_squared value for Multiple Linear Regression Model is: 0.8125272966433501
```

In [112]:

```
predicted = Rf.predict(x_test)
import seaborn as sns
plt.figure(figsize=(12,4))


ax1= sns.distplot(df["price"], hist= False, color="r", label= 'Actual Value')
sns.distplot(predicted, hist= False, color = 'b', label= 'Predicted Values', ax= ax1)

plt.title('Actual vs Predicted Values for price')
plt.xlabel('price')
plt.ylabel('cars')

plt.show()
plt.close()
```

```
C:\Users\Acer\anaconda37\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated functio
n and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with sim
ilar flexibility) or `kdeplot` (an axes-level function for kernel density plots).
  warnings.warn(msg, FutureWarning)
C:\Users\Acer\anaconda37\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated functio
n and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with sim
ilar flexibility) or `kdeplot` (an axes-level function for kernel density plots).
  warnings.warn(msg, FutureWarning)
```



RandomForestregressor

In [113]:

```
#it is non-linear regreesor
Rf = RandomForestRegressor()
Rf.fit(x_train, y_train)
```

Out[113]:

```
▾ RandomForestRegressor
RandomForestRegressor()
```

In [114]:

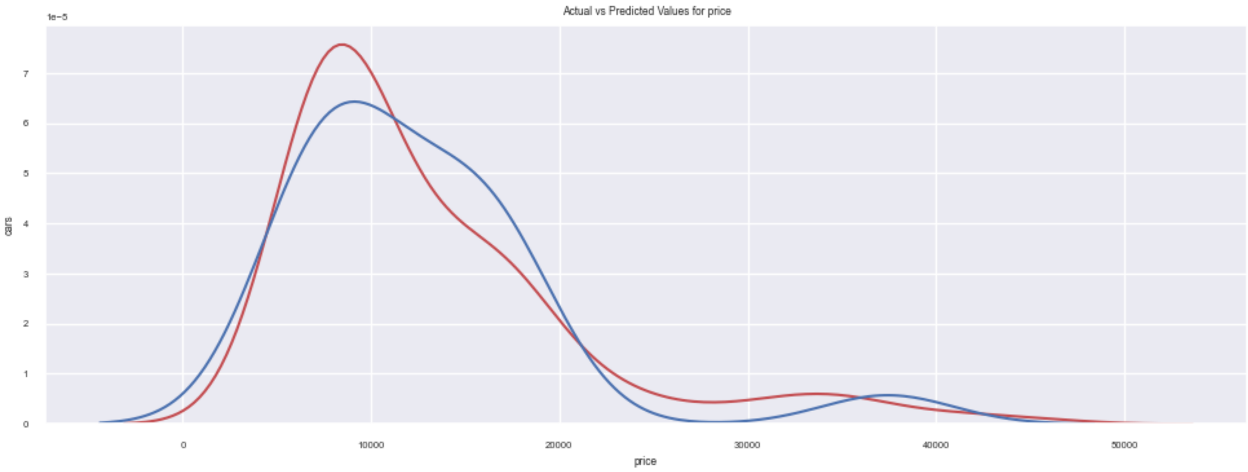
```
print("The R_squared value for Random Forest Regression Model is:", Rf.score(x_test, y_test))
```

```
The R_squared value for Random Forest Regression Model is: 0.9398079770869349
```

In [115]:

```
predicted = Rf.predict(x_test)
import seaborn as sns
plt.figure(figsize=(12,4))


ax1= sns.distplot(df["price"], hist= False, color="r", label= 'Actual Value')
sns.distplot(predicted, hist= False, color = 'b', label= 'Predicted Values', ax= ax1)

plt.title('Actual vs Predicted Values for price')
plt.xlabel('price')
plt.ylabel('cars')

plt.show()
plt.close()
```

```
C:\Users\Acer\anaconda37\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated functio
n and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with sim
ilar flexibility) or `kdeplot` (an axes-level function for kernel density plots).
  warnings.warn(msg, FutureWarning)
C:\Users\Acer\anaconda37\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated functio
n and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with sim
ilar flexibility) or `kdeplot` (an axes-level function for kernel density plots).
  warnings.warn(msg, FutureWarning)
```



Best Model Refinement ¶

In [123]:

```
def build_and_compile_model(norm):
    model = keras.Sequential ([
    norm,
    layers.Dense(64, activation='relu'),
    layers.Dense(64, activation='relu'),
    layers.Dense(1)])

    model.compile(loss='mean_absolute_error',
                  optimizer=tf.keras.optimizers.Adam(0.001))
    return model
```

In [126]:

```
import tensorflow as tf

from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.layers.experimental import preprocessing

print(tf.__version__)
```

```
2.11.0
```

In [127]:

```
normalizer = preprocessing.Normalization()
```

In [128]:

```
normalizer.adapt(np.array(x_train))
```

In [129]:

```
print(normalizer.mean.numpy())
```

```
[[1.03419952e+02 2.55034619e+03 1.27275635e+02 3.07243595e+01
  3.33301258e+00 9.85557709e+01 2.51666679e+01 8.36324394e-01
  9.10504699e-01]]
```

In [130]:

```
dnn_model = build_and_compile_model(normalizer)
dnn_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 normalization (Normalizatio  (None, 9)                19
 n)

 dense (Dense)               (None, 64)                640

 dense_1 (Dense)             (None, 64)                4160

 dense_2 (Dense)             (None, 1)                 65

=================================================================
Total params: 4,884
Trainable params: 4,865
Non-trainable params: 19
_________________________________________________________________
```

In [142]:

```
best_dnn_model = dnn_model.evaluate(x_train, y_train, verbose=0)
```

In [143]:

```
best_dnn_model
```

Out[143]:

```
13316.3154296875
```

In [144]:

```
dnn_model.save('dnn_model')
```

```
WARNING:absl:Found untraced functions such as _update_step_xla while saving (showing 1 of 1). These functions will not be d
irectly callable after loading.

INFO:tensorflow:Assets written to: dnn_model\assets

INFO:tensorflow:Assets written to: dnn_model\assets
```

In [ ]:

```

```