In [72]:
```
import numpy as np
import pandas as pd
import matplotlib as mpl
import matplotlib.patches as mpatches
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from sklearn.ensemble import RandomForestRegressor
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import classification_report, confusion_matrix
from sklearn import metrics
from sklearn.metrics import accuracy_score, precision_score, recall_score
from sklearn.linear_model import LinearRegression,Ridge,Lasso
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics    import    r2_score
```

In [73]:
```
#reading csv file
data = pd.read_csv(r"/All data/Automobile_data.csv")
data.shape
```

Out[73]: (205, 26)

In [74]:
```
data.head(5)
```

Out[74]:

| | symboling | normalized-losses | make | fuel-type | aspiration | num-of-doors | body-style | drive-wheels | engine-location | wheel-base | ... | eng |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 3 | ? | alfa-romero | gas | std | two | convertible | rwd | front | 88.6 | ... | |
| **1** | 3 | ? | alfa-romero | gas | std | two | convertible | rwd | front | 88.6 | ... | |
| **2** | 1 | ? | alfa-romero | gas | std | two | hatchback | rwd | front | 94.5 | ... | |
| **3** | 2 | 164 | audi | gas | std | four | sedan | fwd | front | 99.8 | ... | |
| **4** | 2 | 164 | audi | gas | std | four | sedan | 4wd | front | 99.4 | ... | |

5 rows × 26 columns

## identify and handle misssing values

In [75]:
```
# convert "?" to Nan seeing in the normalized-losses column
data.replace("?", np.nan, inplace=True)
data
```

Out[75]:

| | symboling | normalized-losses | make | fuel-type | aspiration | num-of-doors | body-style | drive-wheels | engine-location | wheel-base | ... |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 3 | NaN | alfa-romero | gas | std | two | convertible | rwd | front | 88.6 | ... |
| **1** | 3 | NaN | alfa-romero | gas | std | two | convertible | rwd | front | 88.6 | ... |
| **2** | 1 | NaN | alfa-romero | gas | std | two | hatchback | rwd | front | 94.5 | ... |
| **3** | 2 | 164 | audi | gas | std | four | sedan | fwd | front | 99.8 | ... |
| **4** | 2 | 164 | audi | gas | std | four | sedan | 4wd | front | 99.4 | ... |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **200** | -1 | 95 | volvo | gas | std | four | sedan | rwd | front | 109.1 | ... |
| **201** | -1 | 95 | volvo | gas | turbo | four | sedan | rwd | front | 109.1 | ... |
| **202** | -1 | 95 | volvo | gas | std | four | sedan | rwd | front | 109.1 | ... |
| **203** | -1 | 95 | volvo | diesel | turbo | four | sedan | rwd | front | 109.1 | ... |
| **204** | -1 | 95 | volvo | gas | turbo | four | sedan | rwd | front | 109.1 | ... |

205 rows × 26 columns

In [76]:
```
#Evaluating for any missing data
missing_data=data.isnull()
missing_data.head(10)
```

Out[76]:

| | symboling | normalized-losses | make | fuel-type | aspiration | num-of-doors | body-style | drive-wheels | engine-location | wheel-base | ... | engine-size |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | False | True | False | False | False | False | False | False | False | False | ... | False |
| **1** | False | True | False | False | False | False | False | False | False | False | ... | False |
| **2** | False | True | False | False | False | False | False | False | False | False | ... | False |
| **3** | False | False | False | False | False | False | False | False | False | False | ... | False |
| **4** | False | False | False | False | False | False | False | False | False | False | ... | False |
| **5** | False | True | False | False | False | False | False | False | False | False | ... | False |
| **6** | False | False | False | False | False | False | False | False | False | False | ... | False |
| **7** | False | True | False | False | False | False | False | False | False | False | ... | False |
| **8** | False | False | False | False | False | False | False | False | False | False | ... | False |
| **9** | False | True | False | False | False | False | False | False | False | False | ... | False |

10 rows × 26 columns

**we can use that data are missing in normalized-losses, num-of-doors, bore, stroke, horsepower, pea-rpm and price column**

In [77]:
```
#count missing values in each column
data.isnull().sum()
```

Out[77]:
```
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 [78]:
```
data.dtypes
```

Out[78]:
```
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 [79]:
```
data.dtypes == "object"
```

Out[79]:
```
symboling            False
normalized-losses     True
make                  True
fuel-type             True
aspiration            True
num-of-doors          True
body-style            True
drive-wheels          True
engine-location       True
wheel-base           False
length               False
width                False
height               False
curb-weight          False
engine-type           True
num-of-cylinders      True
engine-size          False
fuel-system           True
bore                  True
stroke                True
compression-ratio    False
horsepower            True
peak-rpm              True
city-mpg             False
highway-mpg          False
price                 True
dtype: bool
```

***Dealing with missing data***

1. Drop data: (a) drop the whole columns (b) drop the whole rows
2. Replace data: (a) replace it by mean (b) replace it by frequency
3. Replace it based on other functions

In [80]:
```
# Repalcing "NaN" in various columns with the average of their columns
columns = ["normalized-losses", "bore", "stroke", "horsepower","peak-rpm"]
for column in columns:
    avg = data[column].astype("float").mean(axis=0)
    data[column].replace(np.nan, avg, inplace = True)
#for numerical variable
```

In [81]:
```
#Replace NAn in various columns with the most frequency class of their columns for ca
tegorical variable
data["num-of-doors"].value_counts()
```

Out[81]:
```
four    114
two      89
Name: num-of-doors, dtype: int64
```

**we see that four doors are the most common type. we can also use "idxmax()" method to automatically calculate the most common type**

In [82]:
```
data["num-of-doors"].value_counts().idxmax()
```

Out[82]:
```
'four'
```

In [83]:
```
#replace the missing "num-of-doors" values by the most frequency
data["num-of-doors"].replace(np.nan,"four", inplace=True)
data.head()
```

Out[83]:

| | symboling | normalized-losses | make | fuel-type | aspiration | num-of-doors | body-style | drive-wheels | engine-location | wheel-base | ... | eng |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 3 | 122 | alfa-romero | gas | std | two | convertible | rwd | front | 88.6 | ... | |
| **1** | 3 | 122 | alfa-romero | gas | std | two | convertible | rwd | front | 88.6 | ... | |
| **2** | 1 | 122 | alfa-romero | gas | std | two | hatchback | rwd | front | 94.5 | ... | |
| **3** | 2 | 164 | audi | gas | std | four | sedan | fwd | front | 99.8 | ... | |
| **4** | 2 | 164 | audi | gas | std | four | sedan | 4wd | front | 99.4 | ... | |

5 rows × 26 columns

In [84]:
```
data.dropna(subset=["price"], axis=0, inplace=True)
data.reset_index(drop=True, inplace=True)
data.head()
```

Out[84]:

| | symboling | normalized-losses | make | fuel-type | aspiration | num-of-doors | body-style | drive-wheels | engine-location | wheel-base | ... | eng |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 3 | 122 | alfa-romero | gas | std | two | convertible | rwd | front | 88.6 | ... | |
| **1** | 3 | 122 | alfa-romero | gas | std | two | convertible | rwd | front | 88.6 | ... | |
| **2** | 1 | 122 | alfa-romero | gas | std | two | hatchback | rwd | front | 94.5 | ... | |
| **3** | 2 | 164 | audi | gas | std | four | sedan | fwd | front | 99.8 | ... | |
| **4** | 2 | 164 | audi | gas | std | four | sedan | 4wd | front | 99.4 | ... | |

5 rows × 26 columns

**let's check to see if there is any mising values again**

In [85]:
```
data.isnull().sum()
```

Out[85]:
```
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 [86]:
```
#check for duplicate
data.duplicated().sum()
```

Out[86]:
```
0
```

In [87]:
```
data.dtypes
```

Out[87]:
```
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
```

let's convert bore, stroke, price, peak-rpm from object to float and normalized-losses to int

In [88]:
```
data[["bore", "stroke", "peak-rpm", "horsepower", "price"]]= data[["bore", "stroke",
"peak-rpm", "horsepower", "price"]].astype("float")
data[["normalized-losses"]]=data[["normalized-losses"]].astype("int")
```

In [89]:
```
data.dtypes
```

Out[89]:
```
symboling              int64
normalized-losses      int32
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                 float64
stroke               float64
compression-ratio    float64
horsepower           float64
peak-rpm             float64
city-mpg               int64
highway-mpg            int64
price                float64
dtype: object
```

# Data standardization

transform mpg(mile per gallon) unit to L/100km. formular for unit conversion is L/100km = 235/mpg

In [90]:
```
data["city-L/100km"]= 235 / data["city-mpg"]
data["highway-L/100km"]= 235 / data["highway-mpg"]
data[["city-L/100km", "city-mpg", "highway-L/100km", "highway-mpg"]].head()
```

Out[90]:

| | 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 [91]:
```
#replace (original value) by (original value)/maximum value
data["length"]= data["length"]/data["length"].max()
data["width"]= data["width"]/data["width"].max()
data["height"]= data["height"]/data["height"].max()
data[["length", "width", "height"]].head()
```

Out[91]:

| | length | width | height |
|---|---|---|---|
| **0** | 0.811148 | 0.890278 | 0.816054 |
| **1** | 0.811148 | 0.890278 | 0.816054 |
| **2** | 0.822681 | 0.909722 | 0.876254 |
| **3** | 0.848630 | 0.919444 | 0.908027 |
| **4** | 0.848630 | 0.922222 | 0.908027 |

## Binning

In [92]:
```
data["horsepower"].nunique()
```

Out[92]: 59

In [93]:
```
#let's reduce the unique value of horsepower by grouping them
binwidth=(max(data["horsepower"])- min(data["horsepower"]))/4
binwidth
```

Out[93]: 53.5

In [94]:
```
bins=np.arange(min(data["horsepower"]),max(data["horsepower"]), binwidth)
bins
```

Out[94]: array([ 48. , 101.5, 155. , 208.5])

In [95]:
```
#setting group names
group_names=["low","medium","high"]
```

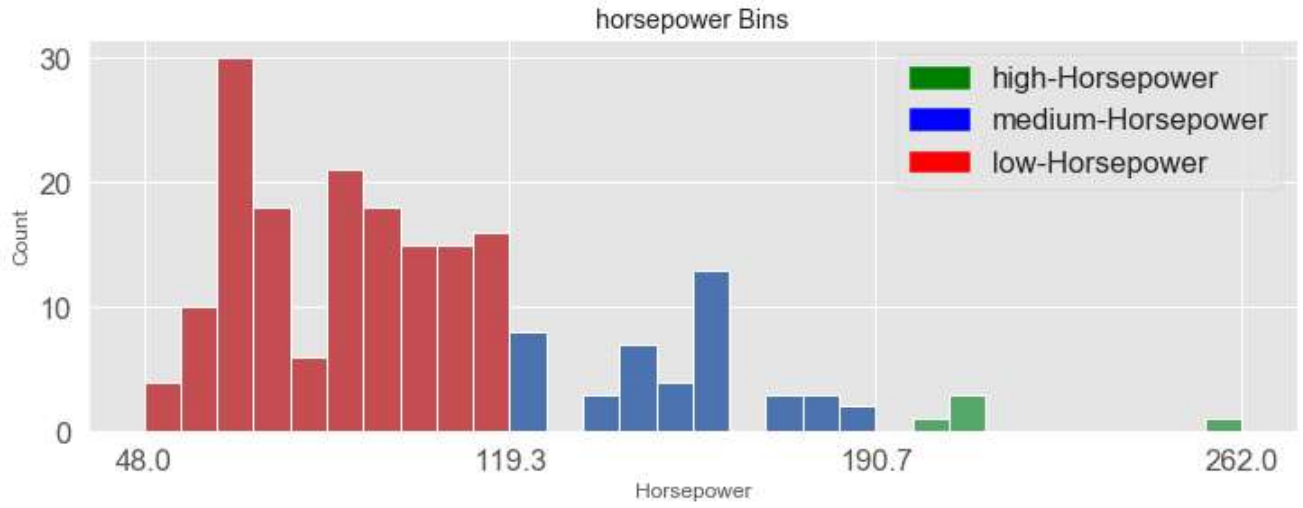
In [96]:
```
#using the function "cut" to determine what value of "data[horsepower]" value belongs to
data["horsepower-binned"]=pd.cut(data["horsepower"], bins, labels=group_names,include_lowest=True)
data[["horsepower", "horsepower-binned"]].head(10)
```

Out[96]:

| | horsepower | horsepower-binned |
|---|---|---|
| **0** | 111.0 | medium |
| **1** | 111.0 | medium |
| **2** | 154.0 | medium |
| **3** | 102.0 | medium |
| **4** | 115.0 | medium |
| **5** | 110.0 | medium |
| **6** | 110.0 | medium |
| **7** | 110.0 | medium |
| **8** | 140.0 | medium |
| **9** | 101.0 | low |

## Bins visualization

In [97]:
```
mpl.style.use(["ggplot"])
total_categories = 3
#let's get the x-ticks values
count, bin_edges = np.histogram(data["horsepower"], 3)
fig, ax=plt.subplots(figsize=(12,4))
N, bins, patches=ax.hist(data["horsepower"], bins=total_categories*10, edgecolor="whi
te", linewidth=1)

for i in range(0,10):
    patches [i].set_facecolor("r")
for i in range(10,20):
    patches [i].set_facecolor("b")
for i in range(20,30):
    patches [i].set_facecolor("g")


ax.set_xticks(bin_edges)
ax.set_title("horsepower Bins")
ax.set_xlabel("Horsepower")
ax.set_ylabel("Count")


#assigning colors to the patches
green_patch=mpatches.Patch(color="green", label="high-Horsepower")
blue_patch=mpatches.Patch(color="blue", label="medium-Horsepower")
red_patch=mpatches.Patch(color="red", label="low-Horsepower")
plt.legend(handles=[green_patch, blue_patch, red_patch])

plt.show()
```




## Creating indicator variable

In [98]:
```
data.columns
data["fuel-type"]
#get indicator variables and assign it to data frame "dummy_varaible_1";
dummy_variable_1 = pd.get_dummies(data["fuel-type"])
dummy_variable_1
```

Out[98]:

| | diesel | gas |
|---|---|---|
| **0** | 0 | 1 |
| **1** | 0 | 1 |
| **2** | 0 | 1 |
| **3** | 0 | 1 |
| **4** | 0 | 1 |
| **...** | ... | ... |
| **196** | 0 | 1 |
| **197** | 0 | 1 |
| **198** | 0 | 1 |
| **199** | 1 | 0 |
| **200** | 0 | 1 |

201 rows × 2 columns

In [99]:
```
dummy_variable_1.rename(columns={"gas":"fuel-type-gas", "diesel":"fuel-type-diesel"},
inplace= True)
dummy_variable_1.head()
```

Out[99]:

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

## we now have 0 to represent "gas" and 1 to represent "diesel" in the column "fuel-type"

In [100]:
```
#merge dataframe "data" and "dummy_variable_1"
data=pd.concat([data, dummy_variable_1], axis=1)
```

In [101]:
```
#drop original column "fuel-type" from data
data.drop("fuel-type", axis = 1, inplace = True)
data.head()
# we drop the fuel-type column because it has been convert from object to numerical values
```

Out[101]:

| | symboling | normalized-losses | make | aspiration | num-of-doors | body-style | drive-wheels | engine-location | wheel-base | length | ... |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 3 | 122 | alfa-romero | std | two | convertible | rwd | front | 88.6 | 0.811148 | ... |
| **1** | 3 | 122 | alfa-romero | std | two | convertible | rwd | front | 88.6 | 0.811148 | ... |
| **2** | 1 | 122 | alfa-romero | std | two | hatchback | rwd | front | 94.5 | 0.822681 | ... |
| **3** | 2 | 164 | audi | std | four | sedan | fwd | front | 99.8 | 0.848630 | ... |
| **4** | 2 | 164 | audi | std | four | sedan | 4wd | front | 99.4 | 0.848630 | ... |

5 rows × 30 columns

In [102]:
```
dummy_variable_2=pd.get_dummies(data["aspiration"])
dummy_variable_2.rename(columns={"std":"aspiration-std","turbo":"aspiration-turbo"},
inplace=True)
data=pd.concat([data, dummy_variable_2], axis=1)
data.drop("aspiration", axis=1, inplace=True)
data.head()
```

Out[102]:

| | symboling | normalized-losses | make | num-of-doors | body-style | drive-wheels | engine-location | wheel-base | length | width | ... |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 3 | 122 | alfa-romero | two | convertible | rwd | front | 88.6 | 0.811148 | 0.890278 | ... |
| **1** | 3 | 122 | alfa-romero | two | convertible | rwd | front | 88.6 | 0.811148 | 0.890278 | ... |
| **2** | 1 | 122 | alfa-romero | two | hatchback | rwd | front | 94.5 | 0.822681 | 0.909722 | ... |
| **3** | 2 | 164 | audi | four | sedan | fwd | front | 99.8 | 0.848630 | 0.919444 | ... |
| **4** | 2 | 164 | audi | four | sedan | 4wd | front | 99.4 | 0.848630 | 0.922222 | ... |

5 rows × 31 columns

In [103]:

```
for column in data.columns:
    if data[column].dtype=="object":
        print(column.upper(), ":", data[column].nunique())
        print(data[column].value_counts().sort_values())
        print("\n")
```

```
MAKE : 22
mercury           1
renault           2
isuzu             2
chevrolet         3
jaguar            3
alfa-romero       3
porsche           4
saab              6
audi              6
plymouth          7
bmw               8
mercedes-benz     8
dodge             9
peugot           11
volvo            11
volkswagen       12
subaru           12
mitsubishi       13
honda            13
mazda            17
nissan           18
toyota           32
Name: make, dtype: int64


NUM-OF-DOORS : 2
two      86
four    115
Name: num-of-doors, dtype: int64


BODY-STYLE : 5
convertible     6
hardtop         8
wagon          25
hatchback      68
sedan          94
Name: body-style, dtype: int64


DRIVE-WHEELS : 3
4wd      8
rwd     75
fwd    118
Name: drive-wheels, dtype: int64


ENGINE-LOCATION : 2
rear       3
front    198
Name: engine-location, dtype: int64


ENGINE-TYPE : 6
rotor      4
l         12
dohc      12
ohcv      13
ohcf      15
ohc      145
Name: engine-type, dtype: int64


NUM-OF-CYLINDERS : 7
```

```
three       1
twelve      1
eight       4
two         4
five       10
six        24
four      157
Name: num-of-cylinders, dtype: int64


FUEL-SYSTEM : 8
spfi     1
mfi      1
4bbl     3
spdi     9
1bbl    11
idi     20
2bbl    64
mpfi    92
Name: fuel-system, dtype: int64
```

## Exploratory data Analysis

finding potential continuous predictor variables for price

In [104]:
```
data.corr()["price"]
```

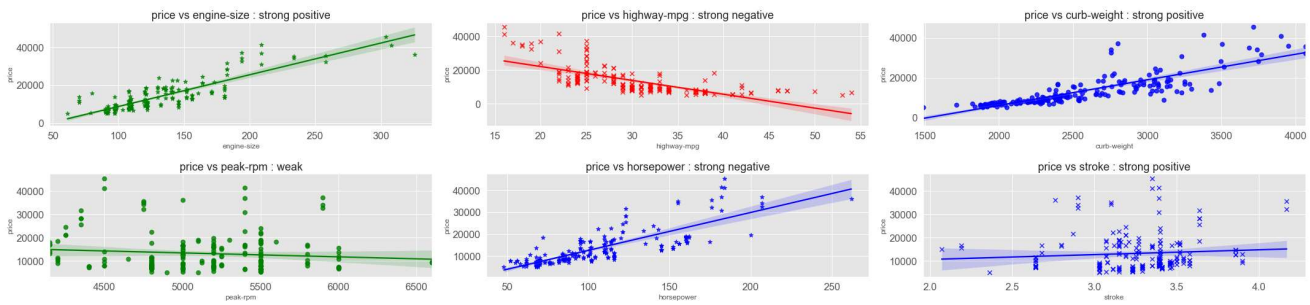
Out[104]:
```
symboling           -0.082391
normalized-losses    0.133999
wheel-base           0.584642
length               0.690628
width                0.751265
height               0.135486
curb-weight          0.834415
engine-size          0.872335
bore                 0.543155
stroke               0.082269
compression-ratio    0.071107
horsepower           0.809575
peak-rpm            -0.101616
city-mpg            -0.686571
highway-mpg         -0.704692
price                1.000000
city-L/100km         0.789898
highway-L/100km      0.801118
fuel-type-diesel     0.110326
fuel-type-gas       -0.110326
aspiration-std      -0.179578
aspiration-turbo     0.179578
Name: price, dtype: float64
```

In [105]:
```
data[["price", "engine-size", "curb-weight", "highway-mpg", "peak-rpm","stroke"]].cor
r()["price"]
```

Out[105]:
```
price          1.000000
engine-size    0.872335
curb-weight    0.834415
highway-mpg   -0.704692
peak-rpm      -0.101616
stroke         0.082269
Name: price, dtype: float64
```

**from the above results it can be concluded that engine-size and curb-weight have strong positive linear relationship; highway-mpg has strong negative linear relationship while peak-rpm and stroke have weak linear relationship with price**

In [106]:
```
#visualizing the linear relationship for possible predictor variables
fig=plt.figure(figsize=(30,10))
ax0=fig.add_subplot(3,3,1)
ax1=fig.add_subplot(3,3,2)
ax2=fig.add_subplot(3,3,3)
ax3=fig.add_subplot(3,3,4)
ax4=fig.add_subplot(3,3,5)
ax5=fig.add_subplot(3,3,6)
sns.set(font_scale=1.5)

#subplot 1
sns.regplot(x="engine-size", y="price", data=data, color="green", marker="*", scatter
_kws={"s":50}, ax=ax0)
ax0.set_title("price vs engine-size : strong positive")

#subplot 2
sns.regplot(x="highway-mpg", y="price", data=data, color="red", marker="x", scatter_k
ws={"s":50}, ax=ax1)
ax1.set_title("price vs highway-mpg : strong negative")

#subplot 3
sns.regplot(x="curb-weight", y="price", data=data, color="blue", marker="o", scatter_
kws={"s":50}, ax=ax2)
ax2.set_title("price vs curb-weight : strong positive")

#subplot 4
sns.regplot(x="peak-rpm", y="price", data=data, color="green", marker="o", scatter_kw
s={"s":50}, ax=ax3)
ax3.set_title("price vs peak-rpm : weak")

#subplot 5
sns.regplot(x="horsepower", y="price", data=data, color="blue", marker="*", scatter_k
ws={"s":50}, ax=ax4)
ax4.set_title("price vs horsepower : strong negative")

#subplot 6
sns.regplot(x="stroke", y="price", data=data, color="blue", marker="x", scatter_kws={
"s":50}, ax=ax5)
ax5.set_title("price vs stroke : strong positive")

fig.tight_layout()
plt.show()
plt.savefig("correlation.jpg")
```



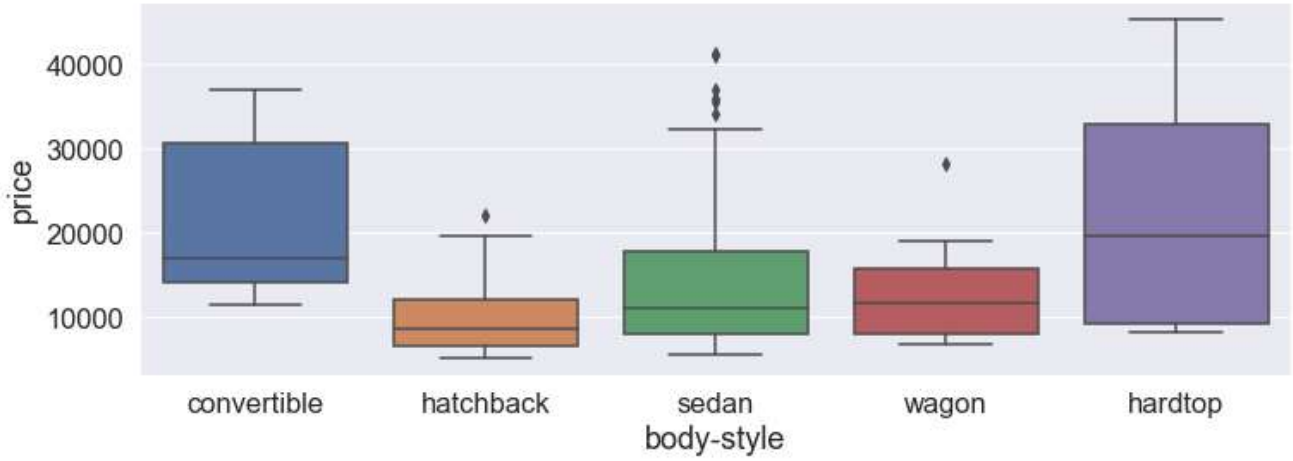

```
<Figure size 432x288 with 0 Axes>
```

## finding potential categorical predictor variables for price

A good way to visualize categorical variables is by using boxplot

In [107]:
```
plt.figure(figsize=(12,4))
sns.boxplot(x="body-style", y="price", data=data)
```

Out[107]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1471908ddc8>
```



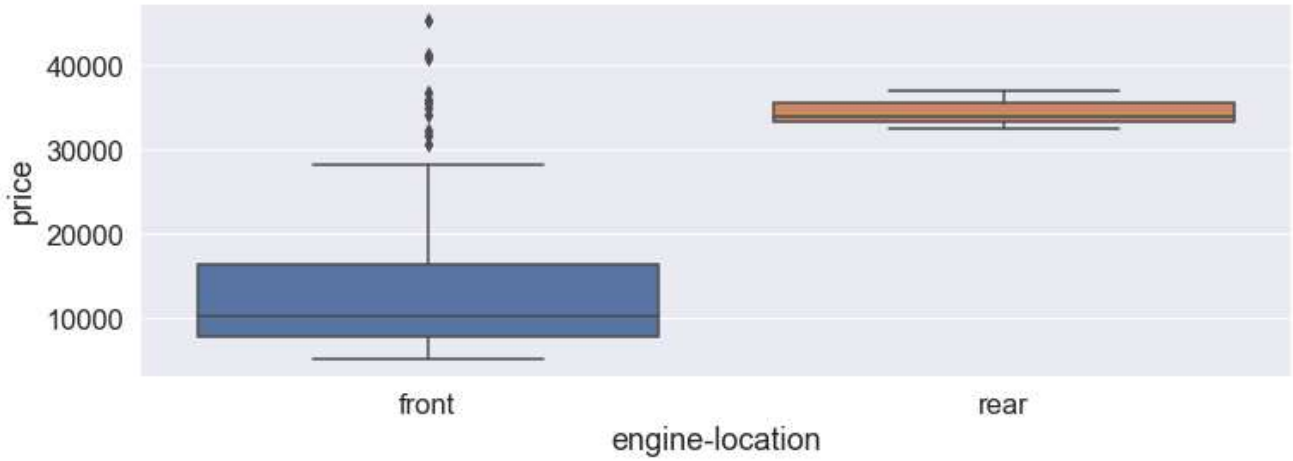

we can see that the distributions of price between the different body-style categories have a significant overlap, and so body-style would not be a good predictor of price

In [108]:
```
#let's assume drive wheel and price
plt.figure(figsize=(12,4))
sns.boxplot(x="drive-wheels", y="price", data=data)
```

Out[108]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x14718fdeec8>
```


In [109]:
```
plt.figure(figsize=(12,4))
sns.boxplot(x="engine-location", y="price", data=data)
```

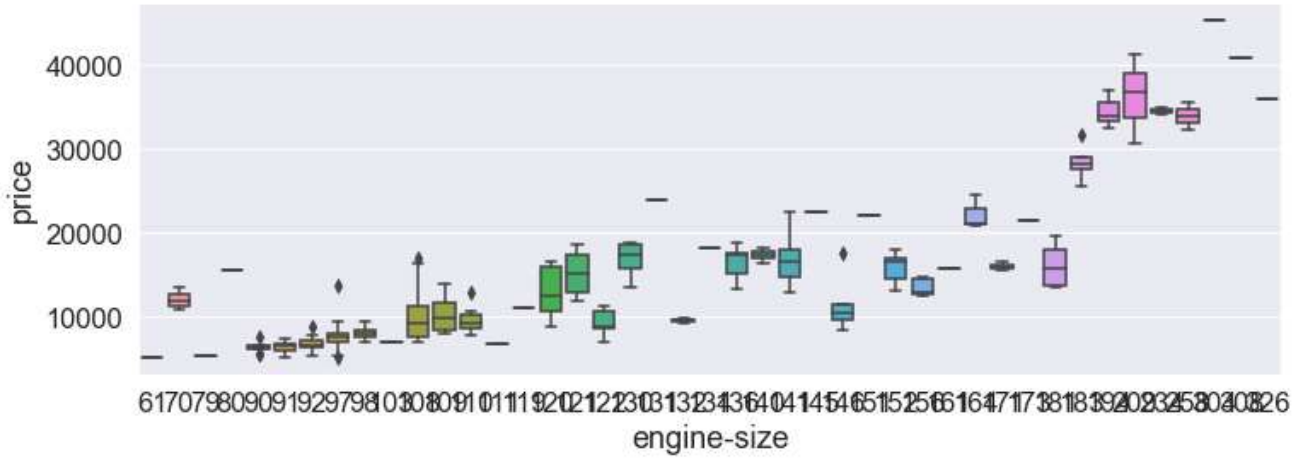
Out[109]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x147179f08c8>
```



Engine-location is a good predictor of price

In [110]:
```
plt.figure(figsize=(12,4))
sns.boxplot(x="engine-size", y="price", data=data)
```

Out[110]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x14718f55908>
```




Descriptive Statistical Analysis

In [111]:
```
#this will skip variable of type object
data.describe()
```

Out[111]:

| | symboling | normalized-losses | wheel-base | length | width | height | curb-weight | engine-size |
|---|---|---|---|---|---|---|---|---|
| **count** | 201.000000 | 201.00000 | 201.000000 | 201.000000 | 201.000000 | 201.000000 | 201.000000 | 201.000000 |
| **mean** | 0.840796 | 122.00000 | 98.797015 | 0.837102 | 0.915126 | 0.899108 | 2555.666667 | 126.875622 |
| **std** | 1.254802 | 31.99625 | 6.066366 | 0.059213 | 0.029187 | 0.040933 | 517.296727 | 41.546834 |
| **min** | -2.000000 | 65.00000 | 86.600000 | 0.678039 | 0.837500 | 0.799331 | 1488.000000 | 61.000000 |
| **25%** | 0.000000 | 101.00000 | 94.500000 | 0.801538 | 0.890278 | 0.869565 | 2169.000000 | 98.000000 |
| **50%** | 1.000000 | 122.00000 | 97.000000 | 0.832292 | 0.909722 | 0.904682 | 2414.000000 | 120.000000 |
| **75%** | 2.000000 | 137.00000 | 102.400000 | 0.881788 | 0.925000 | 0.928094 | 2926.000000 | 141.000000 |
| **max** | 3.000000 | 256.00000 | 120.900000 | 1.000000 | 1.000000 | 1.000000 | 4066.000000 | 326.000000 |

8 rows × 22 columns

In [112]:
```
#let's include variable type object
data.describe(include=["object"])
```

Out[112]:

| | make | num-of-doors | body-style | drive-wheels | engine-location | engine-type | num-of-cylinders | fuel-system |
|---|---|---|---|---|---|---|---|---|
| **count** | 201 | 201 | 201 | 201 | 201 | 201 | 201 | 201 |
| **unique** | 22 | 2 | 5 | 3 | 2 | 6 | 7 | 8 |
| **top** | toyota | four | sedan | fwd | front | ohc | four | mpfi |
| **freq** | 32 | 115 | 94 | 118 | 198 | 145 | 157 | 92 |

In [113]:
```
#value_counts is a good way of understanding how many units of each characteristics variable we have.
data["drive-wheels"].value_counts().to_frame()
```

Out[113]:

| | drive-wheels |
|---|---|
| **fwd** | 118 |
| **rwd** | 75 |
| **4wd** | 8 |

## Group Analysis

In [114]:
```
# To know the type of drive wheel and body style that is most valuable
data_group=data[["drive-wheels", "body-style", "price"]]
data_group_result= data_group.groupby(["drive-wheels", "body-style"], as_index=False).mean()
data_group_result
```

Out[114]:

| | 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 |

let's make the grouped data into a pivot table for better visualization

In [115]:
```
group_pivot=data_group_result.pivot(index="drive-wheels", columns="body-style")
group_pivot
```

Out[115]:

| | price | | | | |
|---|---|---|---|---|---|
| **body-style** | **convertible** | **hardtop** | **hatchback** | **sedan** | **wagon** |
| **drive-wheels** | | | | | |
| **4wd** | NaN | NaN | 7603.000000 | 12647.333333 | 9095.750000 |
| **fwd** | 11595.0 | 8249.000000 | 8396.387755 | 9811.800000 | 9997.333333 |
| **rwd** | 23949.6 | 24202.714286 | 14337.777778 | 21711.833333 | 16994.222222 |

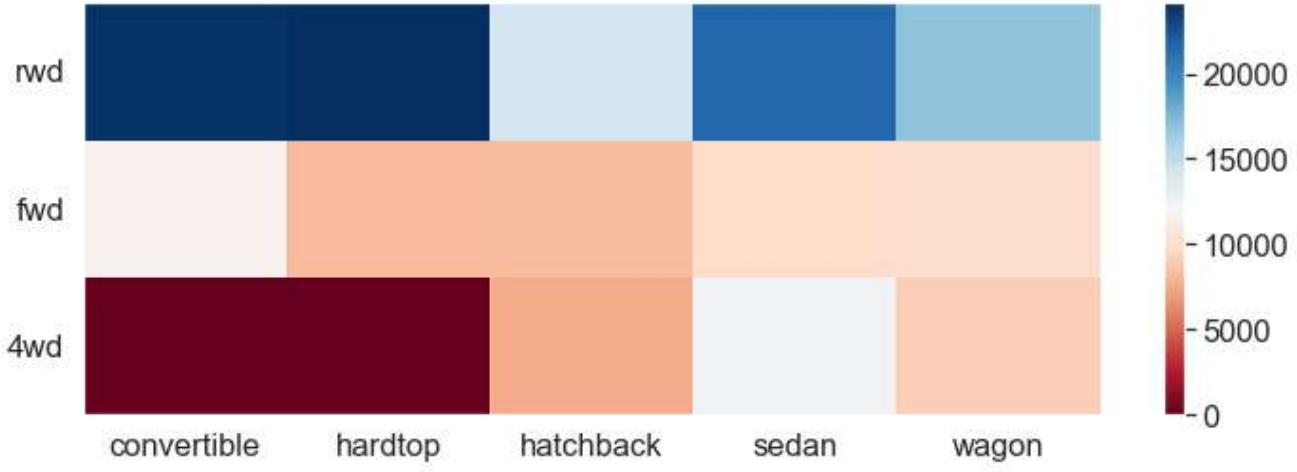
In [116]:
```
#let's fill the missing cell with value 0
group_pivot=group_pivot.fillna(0)
group_pivot
```

Out[116]:

| | price | | | | |
|---|---|---|---|---|---|
| **body-style** | **convertible** | **hardtop** | **hatchback** | **sedan** | **wagon** |
| **drive-wheels** | | | | | |
| **4wd** | 0.0 | 0.000000 | 7603.000000 | 12647.333333 | 9095.750000 |
| **fwd** | 11595.0 | 8249.000000 | 8396.387755 | 9811.800000 | 9997.333333 |
| **rwd** | 23949.6 | 24202.714286 | 14337.777778 | 21711.833333 | 16994.222222 |

## Let's use a heat map to visualize the relationship between body style vs price

In [117]:
```
fig, ax=plt.subplots(figsize=(12,4))
im=ax.pcolor(group_pivot, cmap="RdBu")

#label names
row_labels=group_pivot.columns.levels[1]
col_labels=group_pivot.index
#move ticks and label to the centre
ax.set_xticks(np.arange(group_pivot.shape[1]) +0.5, minor=False)
ax.set_yticks(np.arange(group_pivot.shape[0]) +0.5, minor=False)

#insert labels
ax.set_xticklabels(row_labels, minor=False)
ax.set_yticklabels(col_labels, minor=False)

#rotate label if too long
plt.xticks(rotation=0)
fig.colorbar(im)
plt.show()
```




from the heat map,we notice that the "rwd" to convertible and hardtop are the most expensive cars. since will fill the nan values on "4wd" row with 0 so it appears as the least car

# Correlation and Causation

(1) Correlation is the measure of the extent of interdependence between variables. (2) Causation is the relationship between cause and effect between two variables

In [118]:

```
pearson_coef, p_value=stats.pearsonr(data["wheel-base"], data["price"])
print("The pearson coeficient for wheel base vs price is", pearson_coef, "with a p-va
lue of p =", p_value)


pearson_coef, p_value=stats.pearsonr(data["wheel-base"], data["price"])
print("The pearson coeficient for wheel base vs price is", pearson_coef, "with a p-va
lue of p =", p_value)


pearson_coef, p_value=stats.pearsonr(data["horsepower"], data["price"])
print("The pearson coeficient for horsepower vs price is", pearson_coef, "with a p-va
lue of p =", p_value)


pearson_coef, p_value=stats.pearsonr(data["length"], data["price"])
print("The pearson coeficient for length vs price is", pearson_coef, "with a p-value
 of p =", p_value)


pearson_coef, p_value=stats.pearsonr(data["width"], data["price"])
print("The pearson coeficient for width vs price is", pearson_coef, "with a p-value o
f p =", p_value)


pearson_coef, p_value=stats.pearsonr(data["curb-weight"], data["price"])
print("The pearson coeficient for curb-weight vs price is", pearson_coef, "with a p-v
alue of p =", p_value)

pearson_coef, p_value=stats.pearsonr(data["engine-size"], data["price"])
print("The pearson coeficient for engine-size vs price is", pearson_coef, "with a p-v
alue of p =", p_value)


pearson_coef, p_value=stats.pearsonr(data["bore"], data["price"])
print("The pearson coeficient for bore vs price is", pearson_coef, "with a p-value of
p =", p_value)


pearson_coef, p_value=stats.pearsonr(data["city-mpg"], data["price"])
print("The pearson coeficient for city-mpg vs price is", pearson_coef, "with a p-valu
e of p =", p_value)


pearson_coef, p_value=stats.pearsonr(data["highway-mpg"], data["price"])
print("The pearson coeficient for highway-mpg vs price is", pearson_coef, "with a p-v
alue of p =", p_value)
```

```
The pearson coeficient for wheel base vs price is 0.584641822265508 with a p-value o
f p = 8.076488270733218e-20
The pearson coeficient for wheel base vs price is 0.584641822265508 with a p-value o
f p = 8.076488270733218e-20
The pearson coeficient for horsepower vs price is 0.809574567003656 with a p-value o
f p = 6.369057428259557e-48
The pearson coeficient for length vs price is 0.6906283804483639 with a p-value of p
= 8.016477466159328e-30
The pearson coeficient for width vs price is 0.7512653440522673 with a p-value of p
= 9.200335510481646e-38
The pearson coeficient for curb-weight vs price is 0.8344145257702844 with a p-value
of p = 2.189577238893878e-53
The pearson coeficient for engine-size vs price is 0.8723351674455185 with a p-value
of p = 9.265491622198389e-64
The pearson coeficient for bore vs price is 0.5431553832626602 with a p-value of p =
8.049189483935489e-17
The pearson coeficient for city-mpg vs price is -0.6865710067844678 with a p-value o
f p = 2.321132065567641e-29
The pearson coeficient for highway-mpg vs price is -0.704692265058953 with a p-value
of p = 1.7495471144476358e-31
```

## Anova on Drive Wheels Variable

In [119]:
```
data_group=data[["drive-wheels", "price"]]
groups=data_group[["drive-wheels", "price"]].groupby(["drive-wheels"])
groups.head()
```

Out[119]:

| | drive-wheels | price |
|---|---|---|
| **0** | rwd | 13495.0 |
| **1** | rwd | 16500.0 |
| **2** | rwd | 16500.0 |
| **3** | fwd | 13950.0 |
| **4** | 4wd | 17450.0 |
| **5** | fwd | 15250.0 |
| **6** | fwd | 17710.0 |
| **7** | fwd | 18920.0 |
| **8** | fwd | 23875.0 |
| **9** | rwd | 16430.0 |
| **10** | rwd | 16925.0 |
| **136** | 4wd | 7603.0 |
| **140** | 4wd | 9233.0 |
| **141** | 4wd | 11259.0 |
| **144** | 4wd | 8013.0 |

In [120]:
```
groups.get_group("4wd")["price"]
```

Out[120]:
```
4      17450.0
136     7603.0
140     9233.0
141    11259.0
144     8013.0
145    11694.0
150     7898.0
151     8778.0
Name: price, dtype: float64
```

In [121]:
```
#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= 67.95406500780399 P= 3.3945443577151245e-23
```

The large f-test score shows that drive wheels is important for price prediction

In [122]:
```
f_val, p_val = stats.f_oneway(groups.get_group("fwd")["price"], groups.get_group("rwd")["price"])
print("ANOVA results for separate fwd, rwd:F=:" , f_val, "P=", p_val)


f_val, p_val = stats.f_oneway(groups.get_group("fwd")["price"], groups.get_group("4wd")["price"])
print("ANOVA results for separate fwd, 4wd:F=:" , f_val, "P=", p_val)


f_val, p_val = stats.f_oneway(groups.get_group("4wd")["price"], groups.get_group("rwd")["price"])
print("ANOVA results for separate 4wd, rwd:F=:" , f_val, "P=", p_val)
```

```
ANOVA results for separate fwd, rwd:F=: 130.5533160959111 P= 2.2355306355677845e-23
ANOVA results for separate fwd, 4wd:F=: 0.6654657502523033 P= 0.41620116697845666
ANOVA results for separate 4wd, rwd:F=: 8.580681368924756 P= 0.004411492211225333
```

The separate results show that the main variation of price mean values is in the fwd and rwd groups.

## MODEL DEVELOPMENT AND EVALUATION

In this stage, we will make use of the variable in drive wheels column

In [123]:
```
data.columns
```

Out[123]:
```
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')
```

In [124]:
```
#dropping the column that are not useful for price prediction
data_new=data[["horsepower", "curb-weight", "engine-size", "highway-mpg",
               "bore", "wheel-base", "city-mpg", "stroke", "length","width"]]
```

In [125]:
```
#splitting data into training and testing with ratio 80:20 respectively
x_train, x_test, y_train, y_test = train_test_split(data_new, data["price"], test_siz
e=0.20, random_state=1)
print("number of test samples :", x_test.shape[0])
print("number of training samples :", x_train.shape[0])
```

```
number of test samples : 41
number of training samples : 160
```

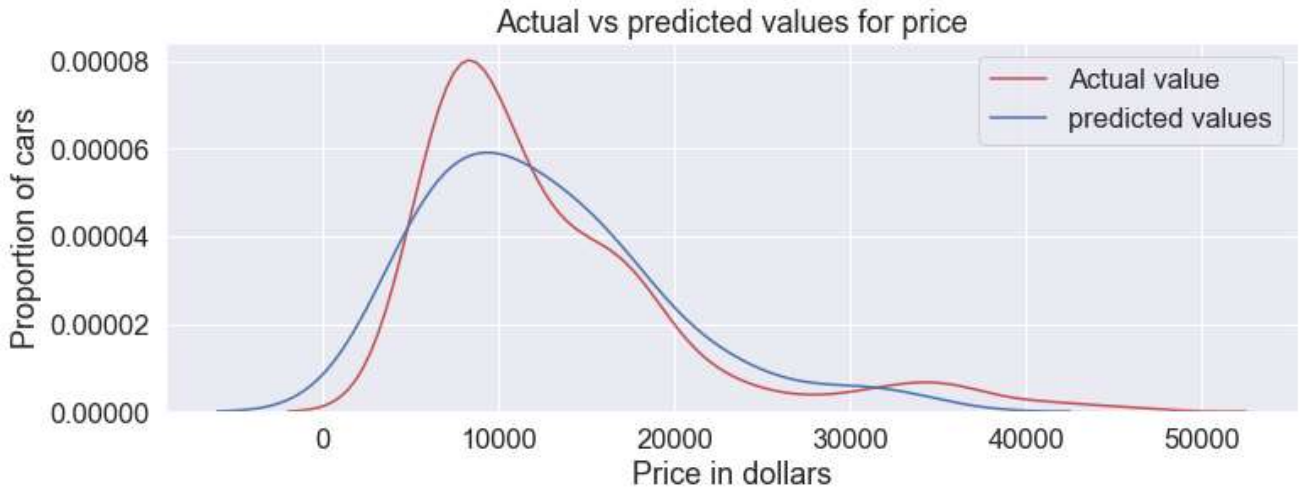
## MULTIPLE LINEAR REGRESSION

In [126]:
```
lm = LinearRegression()
lm.fit(x_train, y_train)
print("The R_square value for multiple linear regression model is:", lm.score(x_test,
y_test))
```

```
The R_square value for multiple linear regression model is: 0.774261438664706
```

We can say that 77% of the variation of the price is explained by this model

In [130]:
```
predicted = lm.predict(x_test)
plt.figure(figsize=(12,4))

ax1 = sns.distplot(data["price"], hist = False, color = "r", label = "Actual value")
sns.distplot(predicted, hist = False, color = "b", label = "predicted values", ax = a
x1)

plt.title("Actual vs predicted values for price")
plt.xlabel("Price in dollars")
plt.ylabel("Proportion of cars")


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




## Polynomial linear regression

In [136]:
```
Input = [("scaler", StandardScaler()), ("polynomial", PolynomialFeatures(include_bias
= False)), ("model", LinearRegression())]
pipe = Pipeline(Input)

pipe.fit(x_train, y_train)
```

Out[136]:
```
Pipeline(memory=None,
         steps=[('scaler',
                 StandardScaler(copy=True, with_mean=True, with_std=True)),
                ('polynomial',
                 PolynomialFeatures(degree=2, include_bias=False,
                                    interaction_only=False, order='C')),
                ('model',
                 LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
                                  normalize=False))],
         verbose=False)
```

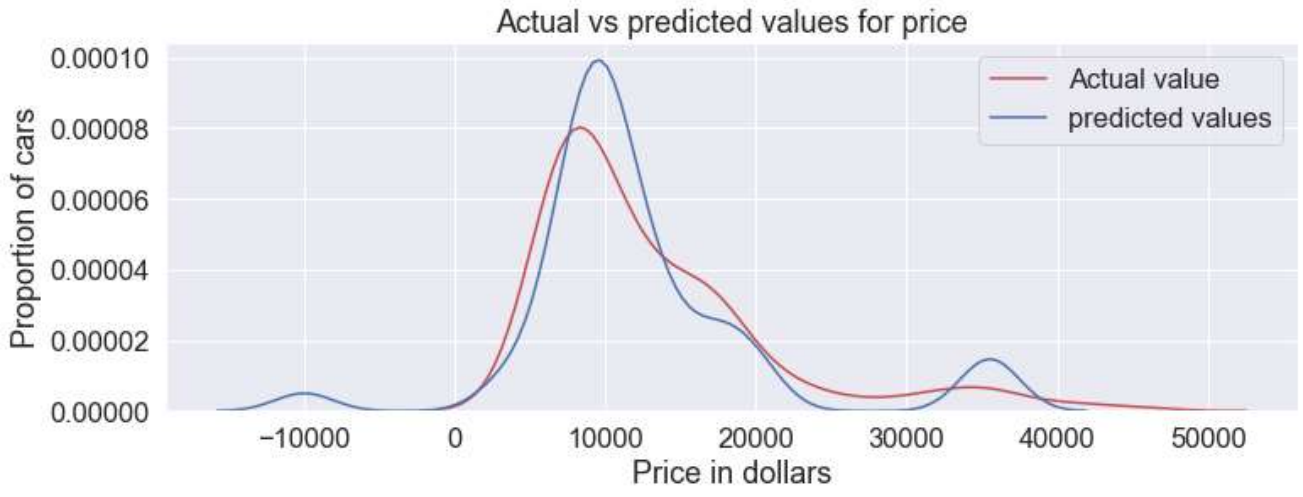
## Polynomial Linear Regression Evaluation

In [137]:
```
print("The R_square value for polynomial linear regression model is:", pipe.score(x_t
est, y_test))
```

```
The R_square value for polynomial linear regression model is: 0.7507750845087007
```

75% of the variation of the model was explained by polynomial linear regression

In [138]:
```
predicted = pipe.predict(x_test)
plt.figure(figsize=(12,4))

ax1 = sns.distplot(data["price"], hist = False, color = "r", label = "Actual value")
sns.distplot(predicted, hist = False, color = "b", label = "predicted values", ax = a
x1)

plt.title("Actual vs predicted values for price")
plt.xlabel("Price in dollars")
plt.ylabel("Proportion of cars")


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




## Random Forest RegressionEvaluation

In [139]:
```
Rf = RandomForestRegressor()
Rf.fit(x_train, y_train)
```

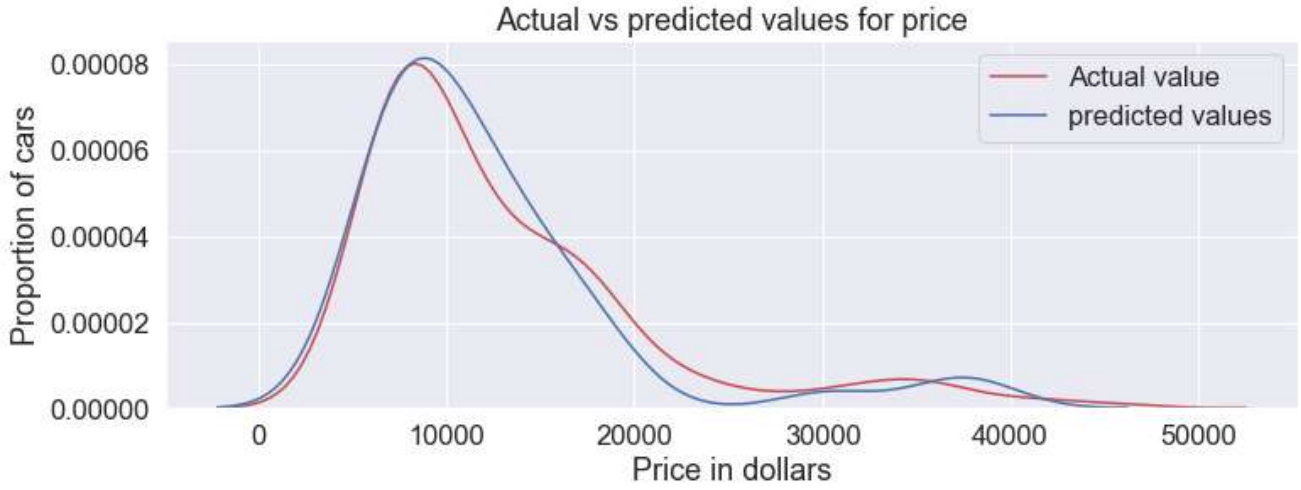
Out[139]:
```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=100, n_jobs=None, oob_score=False,
                      random_state=None, verbose=0, warm_start=False)
```

In [140]:
```
print("The R_square value for Random Forest regression model is:", Rf.score(x_test, y
_test))
```

```
The R_square value for Random Forest regression model is: 0.9570893526844866
```

We can say that 96% of the variation of the price is explained by RandomForestRegression

In [141]:
```
predicted = Rf.predict(x_test)
plt.figure(figsize=(12,4))

ax1 = sns.distplot(data["price"], hist = False, color = "r", label = "Actual value")
sns.distplot(predicted, hist = False, color = "b", label = "predicted values", ax = a
x1)

plt.title("Actual vs predicted values for price")
plt.xlabel("Price in dollars")
plt.ylabel("Proportion of cars")


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




In [ ]: