In [185…
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from scipy import stats
from sklearn.linear_model import LinearRegression
lm = LinearRegression()
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
```

The aim of this project is to find the impact of different factors to number of car sales and also the price of the cars

Data Exploring and Cleaning

In [202…
```
df = pd.read_csv('Car_sales.csv')
```

In [203…
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 157 entries, 0 to 156
Data columns (total 16 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   Manufacturer         157 non-null    object 
 1   Model                157 non-null    object 
 2   Sales_in_thousands   157 non-null    float64
 3   __year_resale_value  121 non-null    float64
 4   Vehicle_type         157 non-null    object 
 5   Price_in_thousands   155 non-null    float64
 6   Engine_size          156 non-null    float64
 7   Horsepower           156 non-null    float64
 8   Wheelbase            156 non-null    float64
 9   Width                156 non-null    float64
 10  Length               156 non-null    float64
 11  Curb_weight          155 non-null    float64
 12  Fuel_capacity        156 non-null    float64
 13  Fuel_efficiency      154 non-null    float64
 14  Latest_Launch        157 non-null    object 
 15  Power_perf_factor    155 non-null    float64
dtypes: float64(12), object(4)
memory usage: 19.8+ KB
```

In [204…
```
df.isna().sum()
```

Out[204]:
```
Manufacturer            0
Model                   0
Sales_in_thousands      0
__year_resale_value    36
Vehicle_type            0
Price_in_thousands      2
Engine_size             1
Horsepower              1
Wheelbase               1
Width                   1
Length                  1
Curb_weight             2
Fuel_capacity           1
Fuel_efficiency         3
Latest_Launch           0
Power_perf_factor       2
dtype: int64
```

In [205…
```
# Dropping NA values, lowering the column names, changing the necessary column name to
df.dropna(inplace=True)
df.columns = df.columns.str.lower()
df.rename(columns = {'__year_resale_value': 'resale_value'}, inplace=True)
df.rename(columns = {'sales_in_thousands': 'sales'}, inplace=True)
df.rename(columns = {'price_in_thousands': 'price'}, inplace=True)
df.rename(columns = {'sales_in_thousands': 'sales'}, inplace=True)
df.rename(columns = {'power_perf_factor': 'performance_factor'}, inplace=True)
```

In [206…
```
df.replace(to_replace="Car", value="Non-Passenger", inplace= True)
```

Explaratory Data Analysis

In [207…
```
df = df.round(1)
df.head()
```

Out[207]:

	manufacturer	model	sales	resale_value	vehicle_type	price	engine_size	horsepower	wheelbase
0	Acura	Integra	16.9	16.4	Passenger	21.5	1.8	140.0	101.2
1	Acura	TL	39.4	19.9	Passenger	28.4	3.2	225.0	108.1
3	Acura	RL	8.6	29.7	Passenger	42.0	3.5	210.0	114.6
4	Audi	A4	20.4	22.3	Passenger	24.0	1.8	150.0	102.6
5	Audi	A6	18.8	23.6	Passenger	34.0	2.8	200.0	108.7

In [ ]:

In [208…
```
# Which manufacturer sold most and least
df.groupby('manufacturer').sum()['sales'].sort_values(ascending = False)
```

Out[208]:
```
manufacturer
Ford          1846.9
Dodge          720.7
Toyota         675.0
Honda          592.7
Chevrolet      446.3
Pontiac        330.9
Jeep           293.2
Nissan         280.5
Buick          242.2
Mercury        238.1
Mitsubishi     180.7
Volkswagen     159.8
Hyundai        137.3
Chrysler       117.6
Saturn         110.3
Cadillac        81.4
Mercedes-B      66.1
Acura           64.9
Lincoln         62.7
Plymouth        62.1
Oldsmobile      60.2
Lexus           43.2
Audi            40.6
BMW             26.7
Infiniti        23.7
Porsche         12.2
Name: sales, dtype: float64
```

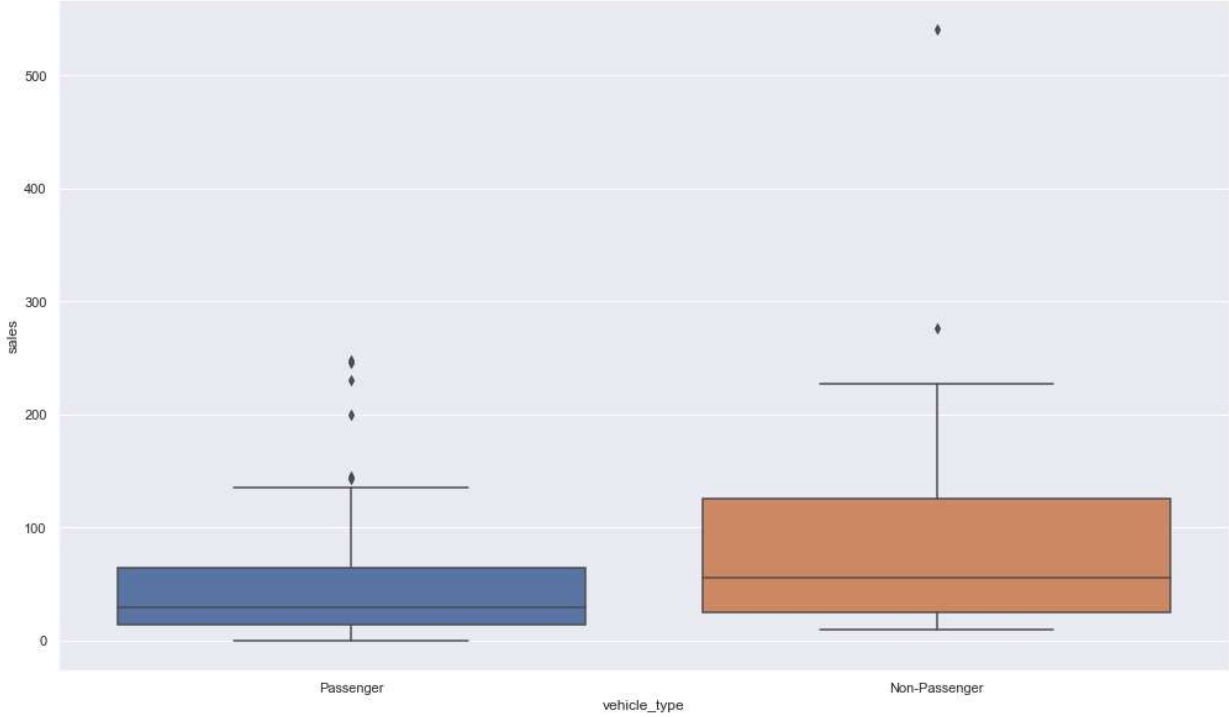
In [209…
```
df.head()
```

Out[209]:

	manufacturer	model	sales	resale_value	vehicle_type	price	engine_size	horsepower	wheelbase
0	Acura	Integra	16.9	16.4	Passenger	21.5	1.8	140.0	101.2
1	Acura	TL	39.4	19.9	Passenger	28.4	3.2	225.0	108.1
3	Acura	RL	8.6	29.7	Passenger	42.0	3.5	210.0	114.6
4	Audi	A4	20.4	22.3	Passenger	24.0	1.8	150.0	102.6
5	Audi	A6	18.8	23.6	Passenger	34.0	2.8	200.0	108.7

In [210…
```
# Lets check the descriptive sales numbers by vehicle type
sns.boxplot(x = 'vehicle_type', y = 'sales', data = df)
```

Out[210]:
```
<AxesSubplot:xlabel='vehicle_type', ylabel='sales'>
```

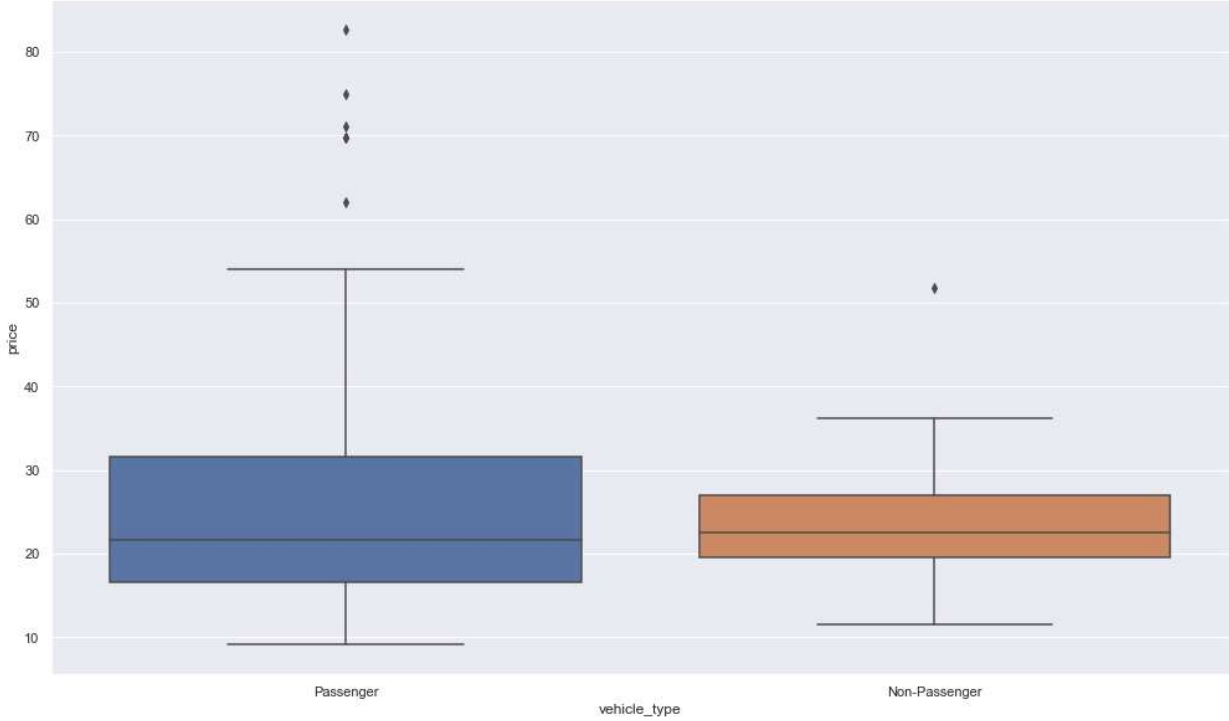


In [211…

```
# Lets check the descriptive price by vehicle type
sns.boxplot(x = 'vehicle_type', y = 'price', data = df)
```

Out[211]:

```
<AxesSubplot:xlabel='vehicle_type', ylabel='price'>
```



In [212…

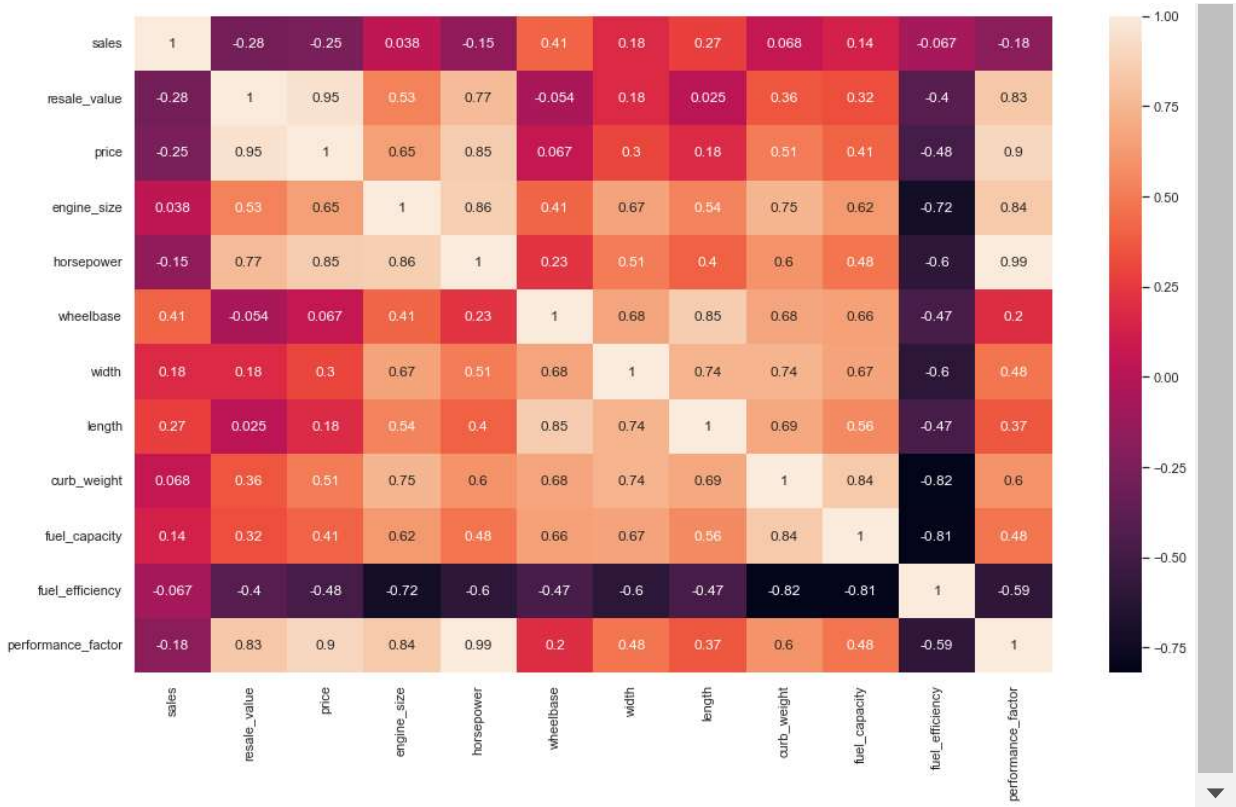
```
# Lets create a subset of our date frame which has only the numeric values that we car
df_numeric = df.drop(['manufacturer', 'model','vehicle_type', 'latest_launch'], axis =
```

In [213…

```
sns.set(rc = {'figure.figsize':(17,10)})
sns.heatmap(df_numeric.corr(), annot = True)
```

Out[213]:

```
<AxesSubplot:>
```



In [ ]:

Linear & Logistic regression Development

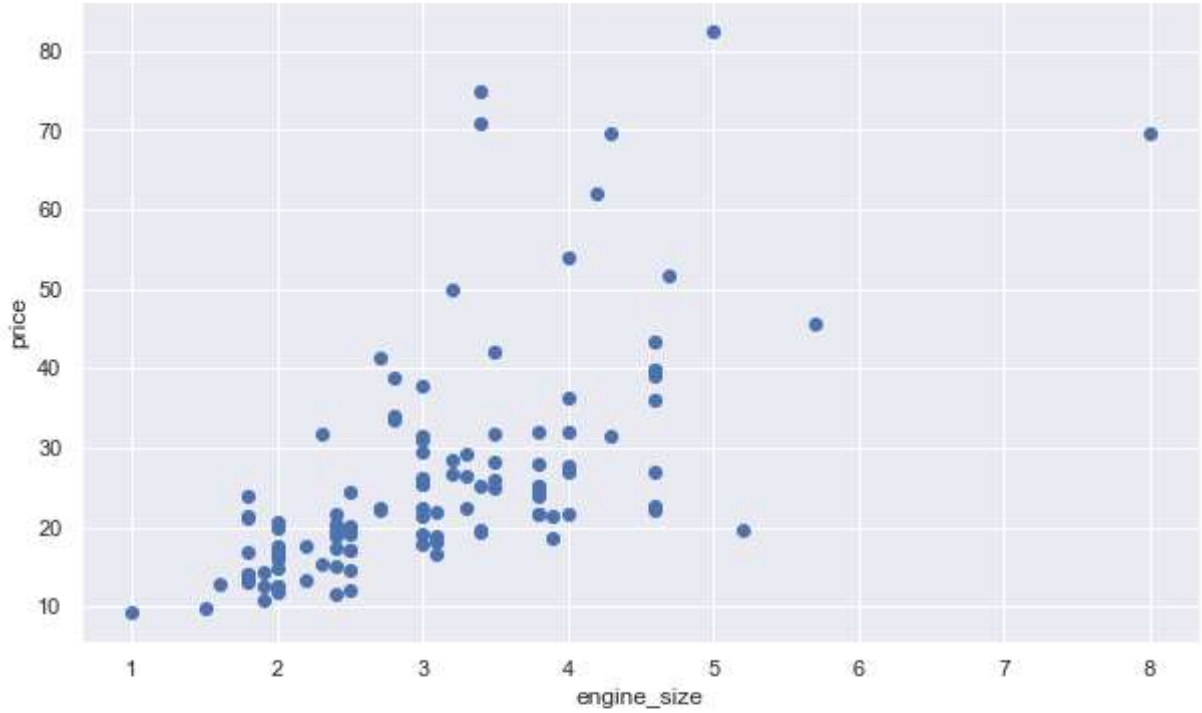
Question 1 : Lets choose price as our target variable, engine_size, horsepower, performance_factor as predictor and try to predict the price

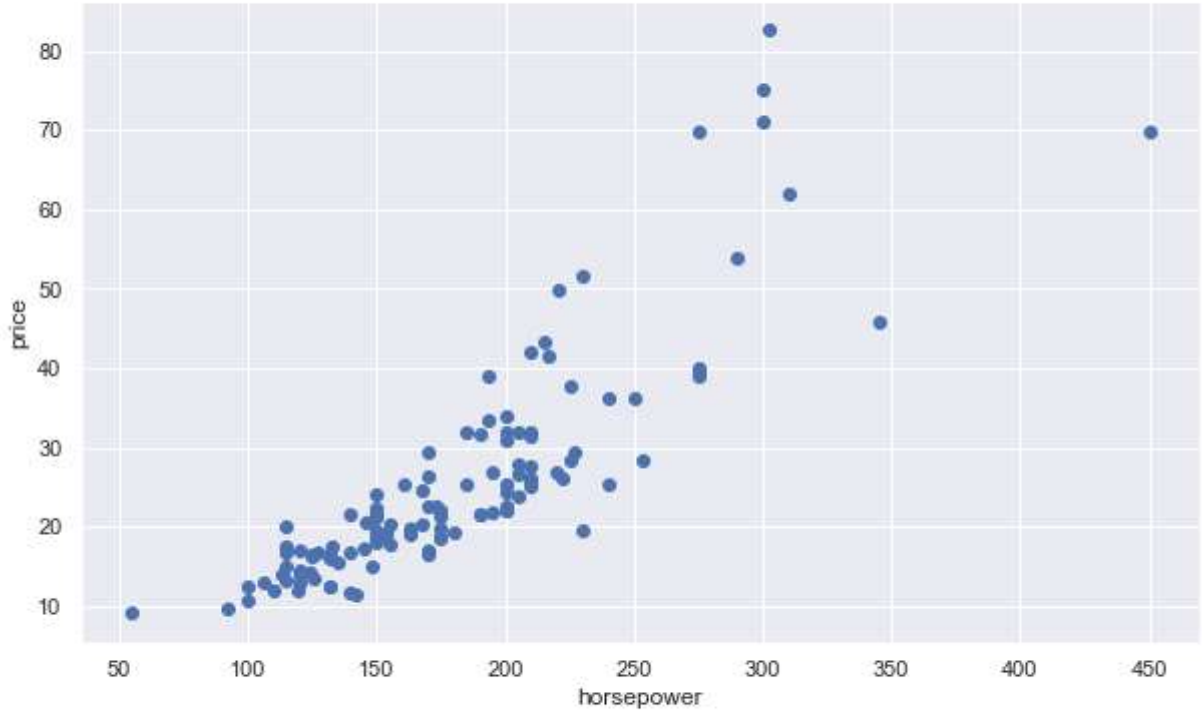
In [261…

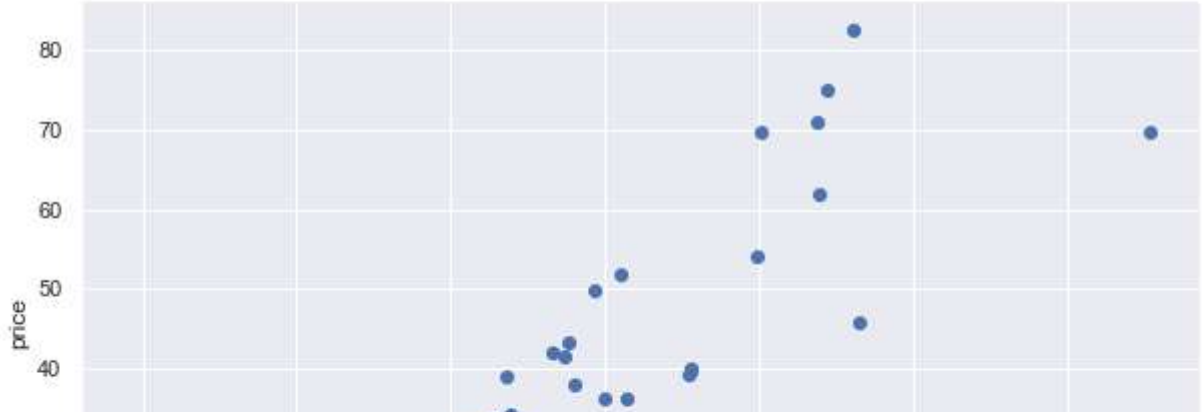
```
#LETS START WITH Question 1 and get the scatter plots to have overview on the data vis

fig, ax = plt.subplots(3, figsize=(10,20))
ax[0].scatter(x = df['engine_size'], y = df['price'], )
ax[0].set_xlabel("engine_size")
ax[0].set_ylabel("price")
print(" For the first plot pearson coefficient is", pearson_coef, "p value is", p_valu


ax[1].scatter(x = df['horsepower'], y = df['price'])
ax[1].set_xlabel("horsepower")
ax[1].set_ylabel("price")
print(" For the second plot pearson coefficient is", pearson_coef1, "p value is", p_va

ax[2].scatter(x = df['performance_factor'], y = df['price'])
ax[2].set_xlabel("performance_factor")
ax[2].set_ylabel("price")
print(" For the third plot pearson coefficient is", pearson_coef2, "p value is", p_val
```

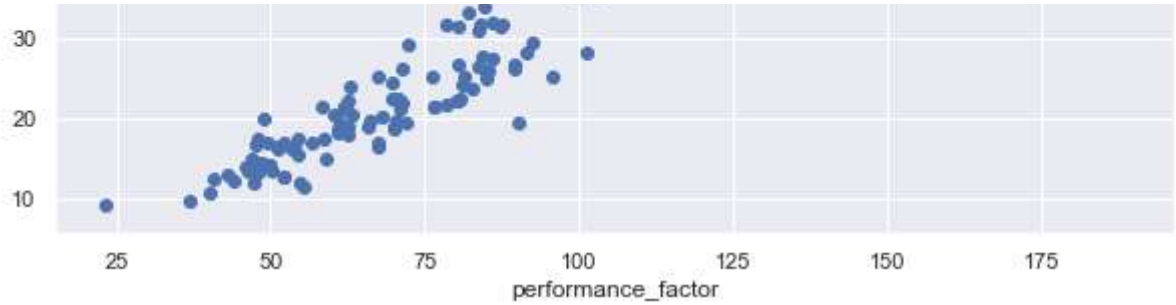
```
plt.show()
```

```
 For the first plot pearson coefficient is 0.6491030352082842 p value is 2.4714399679
611915e-15
 For the second plot pearson coefficient is 0.853390050979465 p value is 2.5154500710
144327e-34
 For the third plot pearson coefficient is 0.9049974199378207 p value is 1.6861501853
757998e-44
```


price
engine_size


price
horsepower


price



In [136…
```
# Now lets create the pearson coefficient and also the p-value to see if the correlati
pearson_coef, p_value = stats.pearsonr(df['engine_size'], df['price'])
pearson_coef1, p_value1 = stats.pearsonr(df['horsepower'], df['price'])
pearson_coef2, p_value2 = stats.pearsonr(df['performance_factor'], df['price'])
```

In [158…
```
# Now lets create our linear regression model and find the coeffiecn and intercept
X = df[['performance_factor', 'horsepower']]
Y = df['price']
lm.fit(X, Y)
Yhat = lm.predict(X)
print (lm.coef_, lm.intercept_)
```

```
[ 2.62655545 -0.94207181] -0.0646886402070983
```

In [262…
```
#Here are the predictions
Yhat
```

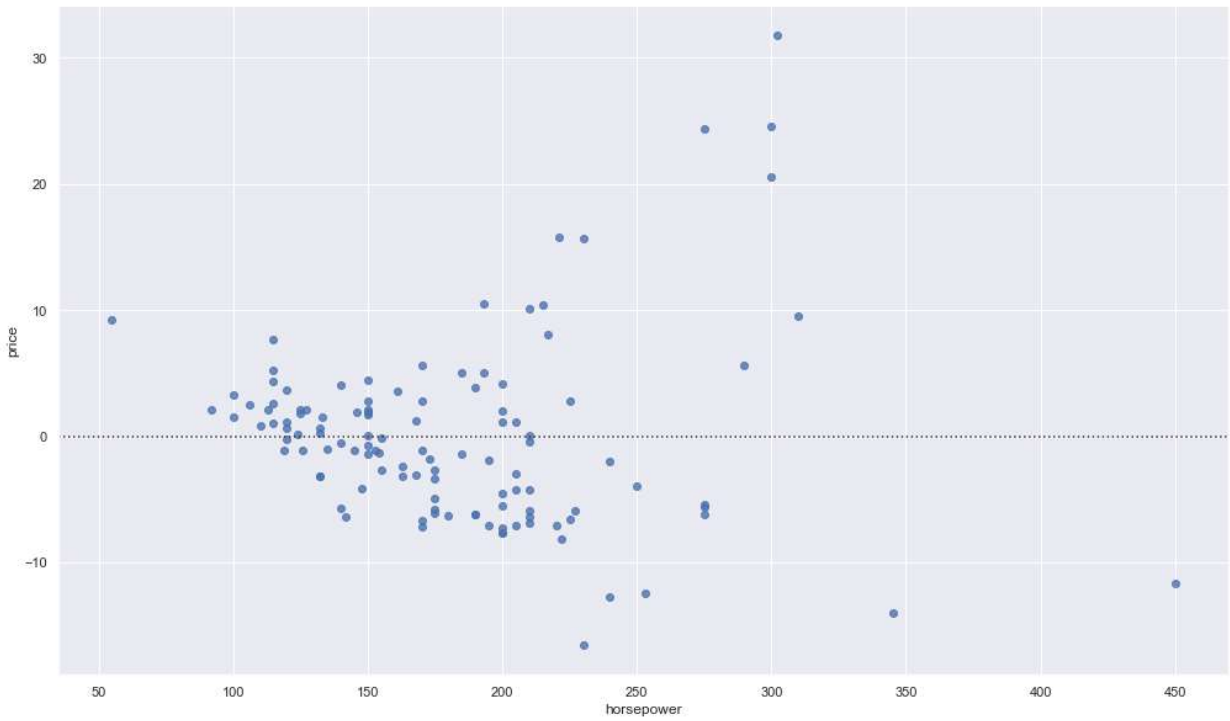
Out[262]:
```
array([21.1734409 , 28.03632257, 42.16739969, 23.57222234, 33.72754072,
       61.69006982, 33.23034366, 38.74611011, 22.08349291, 24.93677833,
       32.16904822, 28.22921504, 40.03022976, 39.76757422, 30.83832973,
       13.46922626, 16.55028569, 18.93162637, 19.47437822, 24.53459665,
       45.52751141, 13.74932257,  9.32010387, 19.73026627, 24.2128513 ,
       21.64538565, 16.36520475, 27.66121089, 12.42537158, 18.94229963,
       20.01036258, 70.05807766, 20.17409699, 21.82083736, 19.19428191,
       17.16384465, 19.36973362, 12.13850777, 21.87315966, 16.81294124,
       18.07389666, 22.95866338, 32.18648898, 21.73363352, 36.49362231,
       12.32749447, 27.49357071, 12.75492846, 15.37757814, 20.24880986,
       26.39062623, 25.88275591,  9.65928999, 11.45518574, 14.95014416,
       28.7787344 , 14.79994475, 22.13581521, 27.40636687, 31.1358668 ,
       37.22926664, 53.47798486, 38.97960758, 44.28608482, 13.7774366 ,
       18.75331292, 17.25104849, 24.83213373, 25.55424306, 31.8889519 ,
       22.6543588 , 16.13066324, 16.39331879, 19.17007364, 23.48397447,
       27.72134472, 22.85401877, 31.31131851, 49.66518686, 69.97296189,
       83.14738817, 13.08734707, 20.17514102, 25.60942709, 26.79385195,
       29.68306294, 18.58176698, 36.00319275, 32.37938146, 25.53289652,
       12.42537158, 15.83989366, 18.84442253, 21.47097798, 19.98224855,
       25.58521883, 21.89060043, 24.02672632, 40.82600793, 70.32282127,
       74.26265444, 10.79034851, 12.62893732, 14.18352406, 12.96135594,
       17.5243773 , 25.09478927, 16.44564109, 11.40963093, 16.87307507,
       22.52160016, 52.48072901, 14.78250398, 16.6210928 , 20.68301134,
       20.03561488, 17.40905943])
```

In [91]:
```
#Now lets have a look to residualplots in order to see if they are distributed evenly
sns.residplot(df['horsepower'], df['price'])
```

```
C:\Users\tural\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning:
Pass the following variables as keyword args: x, y. From version 0.12, the only valid
positional argument will be `data`, and passing other arguments without an explicit k
eyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[91]:

```
<AxesSubplot:xlabel='horsepower', ylabel='price'>
```

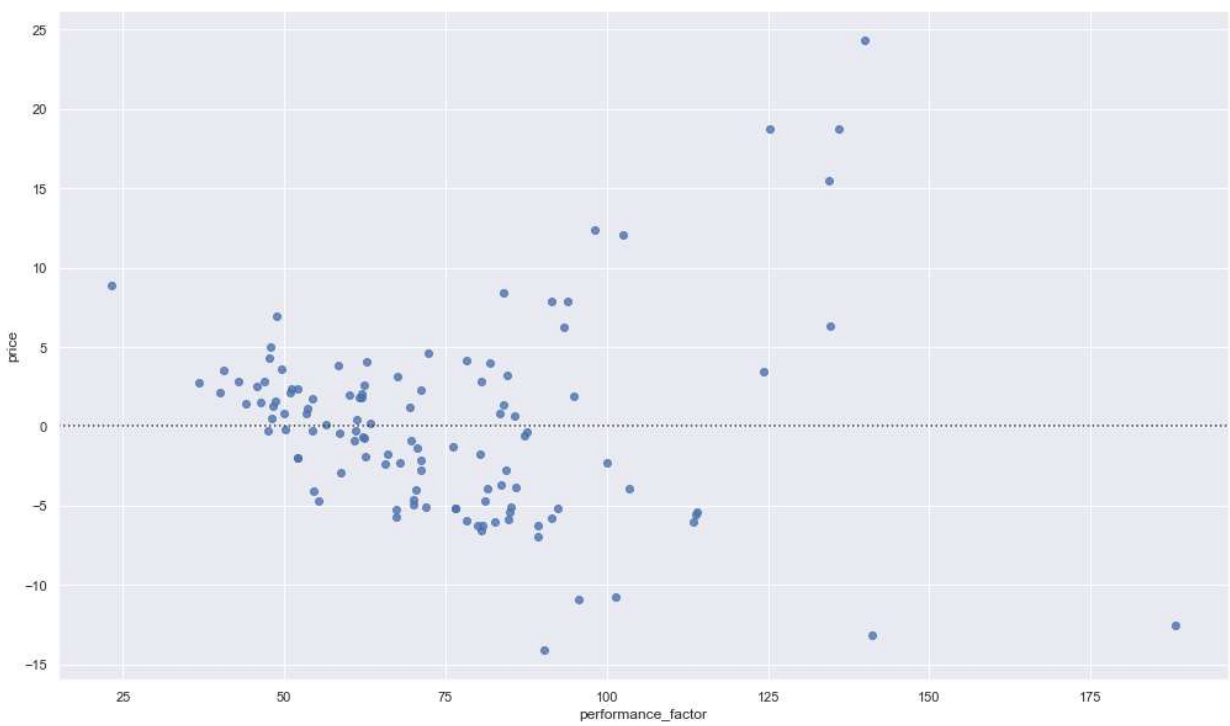


In [98]:

```
sns.residplot(df['performance_factor'], df['price'])
```

Out[98]:

```
<AxesSubplot:xlabel='performance_factor', ylabel='price'>
```



In [102…

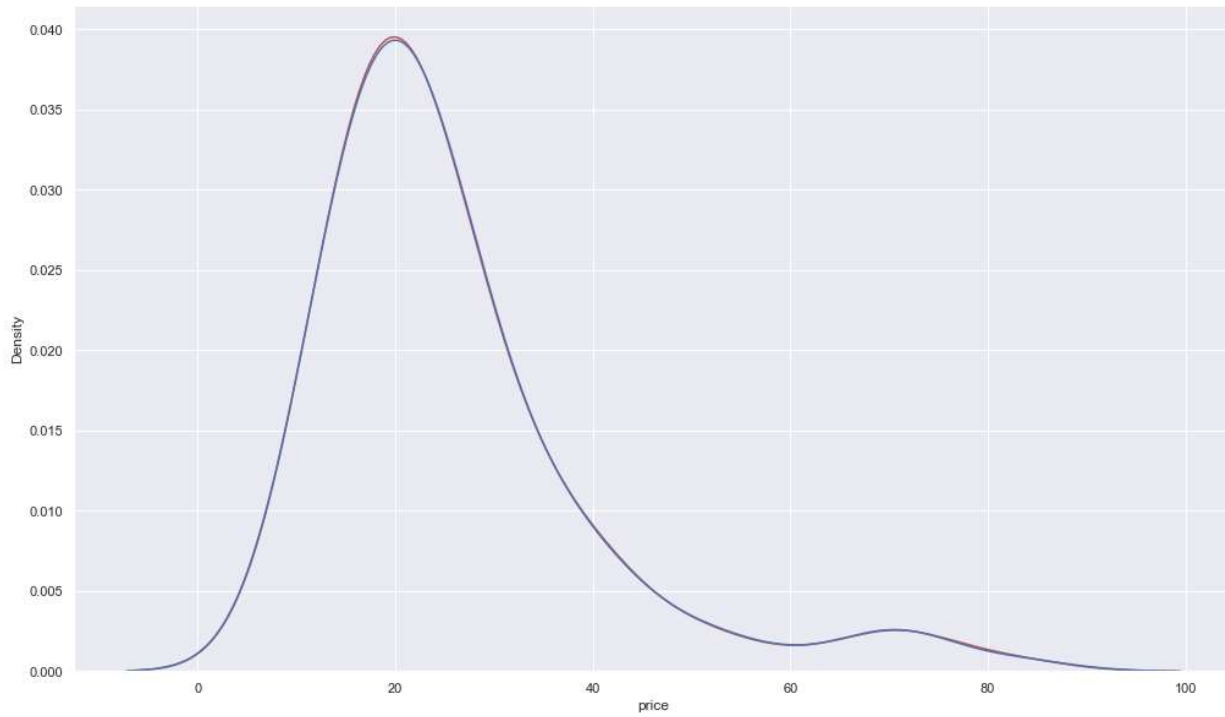
```
# Finally lets draw a distribution plot in order to compare the actual prices with our
ax1 = sns.distplot(df['price'], hist = False, color = 'r', label = "Actual Price")
```

```
sns.distplot(Yhat, hist= False, color = 'b', label = "Fitted Value", ax = ax1)
```

```
C:\Users\tural\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarni
ng: `distplot` is a deprecated function and will be removed in a future version. Plea
se adapt your code to use either `displot` (a figure-level function with similar flex
ibility) or `kdeplot` (an axes-level function for kernel density plots).
  warnings.warn(msg, FutureWarning)
C:\Users\tural\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarni
ng: `distplot` is a deprecated function and will be removed in a future version. Plea
se adapt your code to use either `displot` (a figure-level function with similar flex
ibility) or `kdeplot` (an axes-level function for kernel density plots).
  warnings.warn(msg, FutureWarning)
```

Out[102]:

```
<AxesSubplot:xlabel='price', ylabel='Density'>
```



As it can be seen above, our model is near to perfect!

In [ ]:

```
```

Question 2 Lets try to estimate the number of sales based on vehicle type using Logistic Regression

Passenger is 1 Non Passenger is 0

In [235…

```
df.groupby('vehicle_type').sum()['sales']
```

Out[235]:
```
vehicle_type
Non-Passenger    2766.7
Passenger        4149.3
Name: sales, dtype: float64
```

In [ ]:

In [248…
```
#Lets create the training and testing dataset, then create a logistic regreession mode
x = df[['sales']]
y = df['vehicle_type']
model = LogisticRegression(solver='liblinear', random_state=0)
model.fit(x,y)
model.predict(x)
```

Out[248]:
```
array(['Passenger', 'Passenger', 'Passenger', 'Passenger', 'Passenger',
       'Passenger', 'Passenger', 'Passenger', 'Passenger', 'Passenger',
       'Passenger', 'Passenger', 'Passenger', 'Passenger', 'Passenger',
       'Passenger', 'Passenger', 'Passenger', 'Passenger', 'Passenger',
       'Passenger', 'Passenger', 'Passenger', 'Passenger', 'Passenger',
       'Passenger', 'Passenger', 'Passenger', 'Passenger', 'Passenger',
       'Passenger', 'Passenger', 'Passenger', 'Passenger', 'Passenger',
       'Passenger', 'Passenger', 'Non-Passenger', 'Passenger',
       'Passenger', 'Passenger', 'Passenger', 'Passenger', 'Passenger',
       'Passenger', 'Non-Passenger', 'Passenger', 'Non-Passenger',
       'Passenger', 'Passenger', 'Non-Passenger', 'Non-Passenger',
       'Passenger', 'Non-Passenger', 'Passenger', 'Passenger',
       'Passenger', 'Passenger', 'Passenger', 'Passenger', 'Passenger',
       'Passenger', 'Passenger', 'Passenger', 'Passenger', 'Passenger',
       'Passenger', 'Passenger', 'Passenger', 'Passenger', 'Passenger',
       'Passenger', 'Passenger', 'Passenger', 'Passenger', 'Passenger',
       'Passenger', 'Passenger', 'Passenger', 'Passenger', 'Passenger',
       'Passenger', 'Passenger', 'Passenger', 'Passenger', 'Passenger',
       'Passenger', 'Passenger', 'Passenger', 'Passenger', 'Passenger',
       'Passenger', 'Passenger', 'Passenger', 'Passenger', 'Passenger',
       'Passenger', 'Passenger', 'Passenger', 'Passenger',
       'Non-Passenger', 'Passenger', 'Passenger', 'Passenger',
       'Passenger', 'Passenger', 'Passenger', 'Passenger', 'Passenger',
       'Passenger', 'Passenger', 'Passenger'], dtype=object)
```

In [249…
```
model.predict_proba(x)
```

Out[249]:
```
array([[0.20032658, 0.79967342],
       [0.22760751, 0.77239249],
       [0.19090256, 0.80909744],
       [0.20440374, 0.79559626],
       [0.20253229, 0.79746771],
       [0.18300504, 0.81699496],
       [0.1915723 , 0.8084277 ],
       [0.20102117, 0.79897883],
       [0.30045366, 0.69954634],
       [0.22760751, 0.77239249],
       [0.21334712, 0.78665288],
       [0.28801582, 0.71198418],
       [0.25989871, 0.74010129],
       [0.18857256, 0.81142744],
       [0.19381771, 0.80618229],
       [0.38790243, 0.61209757],
       [0.3702381 , 0.6297619 ],
       [0.20937731, 0.79062269],
       [0.23169312, 0.76830688],
       [0.21153589, 0.78846411],
       [0.20148523, 0.79851477],
       [0.21872509, 0.78127491],
       [0.20616987, 0.79383013],
       [0.19012346, 0.80987654],
       [0.21934236, 0.78065764],
       [0.21724876, 0.78275124],
       [0.21872509, 0.78127491],
       [0.19642457, 0.80357543],
       [0.27733392, 0.72266608],
       [0.18659284, 0.81340716],
       [0.27044481, 0.72955519],
       [0.18246613, 0.81753387],
       [0.53314531, 0.46685469],
       [0.20021099, 0.79978901],
       [0.21712605, 0.78287395],
       [0.33115584, 0.66884416],
       [0.45143373, 0.54856627],
       [0.26902321, 0.73097679],
       [0.33452128, 0.66547872],
       [0.22219792, 0.77780208],
       [0.56653686, 0.43346314],
       [0.25948246, 0.74051754],
       [0.62028149, 0.37971851],
       [0.40569237, 0.59430763],
       [0.35390389, 0.64609611],
       [0.52145305, 0.47854695],
       [0.91647056, 0.08352944],
       [0.4837627 , 0.5162373 ],
       [0.53996478, 0.46003522],
       [0.27330215, 0.72669785],
       [0.19574198, 0.80425802],
       [0.27733392, 0.72266608],
       [0.22989939, 0.77010061],
       [0.26408496, 0.73591504],
       [0.21516962, 0.78483038],
       [0.2083041 , 0.7916959 ],
       [0.24881292, 0.75118708],
       [0.28403651, 0.71596349],
       [0.40778216, 0.59221784],
       [0.20878058, 0.79121942],
```

```
       [0.19551484, 0.80448516],
       [0.18846215, 0.81153785],
       [0.19676655, 0.80323345],
       [0.23988541, 0.76011459],
       [0.21129525, 0.78870475],
       [0.23156468, 0.76843532],
       [0.24881292, 0.75118708],
       [0.1876907 , 0.8123093 ],
       [0.18160644, 0.81839356],
       [0.1939305 , 0.8060695 ],
       [0.22748065, 0.77251935],
       [0.19745184, 0.80254816],
       [0.21165629, 0.78834371],
       [0.26591236, 0.73408764],
       [0.28491793, 0.71508207],
       [0.21298397, 0.78701603],
       [0.20440374, 0.79559626],
       [0.20206643, 0.79793357],
       [0.21298397, 0.78701603],
       [0.20021099, 0.79978901],
       [0.18506417, 0.81493583],
       [0.23169312, 0.76830688],
       [0.29517143, 0.70482857],
       [0.28301026, 0.71698974],
       [0.21262128, 0.78737872],
       [0.23169312, 0.76830688],
       [0.18268155, 0.81731845],
       [0.19779516, 0.80220484],
       [0.20393468, 0.79606532],
       [0.20913846, 0.79086154],
       [0.21921881, 0.78078119],
       [0.18714115, 0.81285885],
       [0.20889982, 0.79110018],
       [0.2434565 , 0.7565435 ],
       [0.36353269, 0.63646731],
       [0.20381754, 0.79618246],
       [0.30166857, 0.69833143],
       [0.22319736, 0.77680264],
       [0.19134885, 0.80865115],
       [0.18289716, 0.81710284],
       [0.18354519, 0.81645481],
       [0.28403651, 0.71596349],
       [0.20925786, 0.79074214],
       [0.18714115, 0.81285885],
       [0.38277433, 0.61722567],
       [0.57043176, 0.42956824],
       [0.26003756, 0.73996244],
       [0.21996088, 0.78003912],
       [0.28920123, 0.71079877],
       [0.20997528, 0.79002472],
       [0.26647626, 0.73352374],
       [0.19224383, 0.80775617],
       [0.19224383, 0.80775617],
       [0.28860816, 0.71139184],
       [0.24279247, 0.75720753],
       [0.19201979, 0.80798021],
       [0.18758069, 0.81241931]])
```

In [250…

```
model.score(x, y)
```

Out[250]:
```
0.7606837606837606
```

In [251…
```
#4 times model predicted non-passenger correctly
#3 times model predicted non-passengers incorrectly instead of passenger
#25 times model predicted passenger incorreclty instead of non - passenger
#85 times mdoel predicted passenger correctly
confusion_matrix(y, model.predict(x))
```

Out[251]:
```
array([[ 4, 25],
       [ 3, 85]], dtype=int64)
```

Conclusion

As we can see from the above there is some relationship between vehicle type and number of sales, also our model has 7.6 acccuracty out of 10 whihc is not ideal. We can predict the sales of cars but not with a great accuracy with this model