MULTIPLE LINEAR REGRESSION

Dataset taken fom kaggle - https://www.kaggle.com/datasets/hellbuoy/c price-prediction?resource=download

.

Problem statement : The price of the american cars are different from the price from Chinese cars

    😯 To know the varibales that significantly effects the dependent variable.

    😮 To know how those variables describes the price of the car.

Let's import the all necassary Libraries!!!!

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
import plotly.express as px
import statsmodels.api as sm
from scipy.stats import boxcox
from statsmodels.formula.api import ols
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

let's load the dataset!!!!!!!!!!!!

In [2]:
```
df=pd.read_csv("CarPrice_Assignment.csv")
```

In [3]:
```
df.head(10)
```

Out[3]:

	car_ID	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel	eng
0	1	3	alfa-romero giulia	gas	std	two	convertible	rwd	
1	2	3	alfa-romero stelvio	gas	std	two	convertible	rwd	
2	3	1	alfa-romero Quadrifoglio	gas	std	two	hatchback	rwd	
3	4	2	audi 100 ls	gas	std	four	sedan	fwd	
4	5	2	audi 100ls	gas	std	four	sedan	4wd	
5	6	2	audi fox	gas	std	two	sedan	fwd	
6	7	1	audi 100ls	gas	std	four	sedan	fwd	
7	8	1	audi 5000	gas	std	four	wagon	fwd	
8	9	1	audi 4000	gas	turbo	four	sedan	fwd	
9	10	0	audi 5000s (diesel)	gas	turbo	two	hatchback	4wd	

10 rows × 26 columns

In [4]:
```
df.describe().T
```

Out[4]:

	count	mean	std	min	25%	50%	75%	ma:
car_ID	205.0	103.000000	59.322565	1.00	52.00	103.00	154.00	205.0(
symboling	205.0	0.834146	1.245307	-2.00	0.00	1.00	2.00	3.0(
wheelbase	205.0	98.756585	6.021776	86.60	94.50	97.00	102.40	120.9(
carlength	205.0	174.049268	12.337289	141.10	166.30	173.20	183.10	208.1(
carwidth	205.0	65.907805	2.145204	60.30	64.10	65.50	66.90	72.3(
carheight	205.0	53.724878	2.443522	47.80	52.00	54.10	55.50	59.8(
curbweight	205.0	2555.565854	520.680204	1488.00	2145.00	2414.00	2935.00	4066.0(
enginesize	205.0	126.907317	41.642693	61.00	97.00	120.00	141.00	326.0(
boreratio	205.0	3.329756	0.270844	2.54	3.15	3.31	3.58	3.9
stroke	205.0	3.255415	0.313597	2.07	3.11	3.29	3.41	4.1
compressionratio	205.0	10.142537	3.972040	7.00	8.60	9.00	9.40	23.0(
horsepower	205.0	104.117073	39.544167	48.00	70.00	95.00	116.00	288.0(
peakrpm	205.0	5125.121951	476.985643	4150.00	4800.00	5200.00	5500.00	6600.0(
citympg	205.0	25.219512	6.542142	13.00	19.00	24.00	30.00	49.0(
highwaympg	205.0	30.751220	6.886443	16.00	25.00	30.00	34.00	54.0(
price	205.0	13276.710571	7988.852332	5118.00	7788.00	10295.00	16503.00	45400.0(

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 26 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   car_ID            205 non-null    int64
 1   symboling         205 non-null    int64
 2   CarName           205 non-null    object
 3   fueltype          205 non-null    object
 4   aspiration        205 non-null    object
 5   doornumber        205 non-null    object
 6   carbody           205 non-null    object
 7   drivewheel        205 non-null    object
 8   enginelocation    205 non-null    object
 9   wheelbase         205 non-null    float64
 10  carlength         205 non-null    float64
 11  carwidth          205 non-null    float64
 12  carheight         205 non-null    float64
 13  curbweight        205 non-null    int64
 14  enginetype        205 non-null    object
 15  cylindernumber    205 non-null    object
 16  enginesize        205 non-null    int64
 17  fuelsystem        205 non-null    object
 18  boreratio         205 non-null    float64
 19  stroke            205 non-null    float64
 20  compressionratio  205 non-null    float64
 21  horsepower        205 non-null    int64
 22  peakrpm           205 non-null    int64
 23  citympg           205 non-null    int64
 24  highwaympg        205 non-null    int64
 25  price             205 non-null    float64
dtypes: float64(8), int64(8), object(10)
memory usage: 41.8+ KB
```

Let's summarize about dataset (Data Understanding)

Dependent variable - price(float) [price starts from : 5118.00 to 45400.00 and mean - 13276.710571 std from mean - 7988.852332 on average, about 7988.852332 i.e away from the mean of 13276.710571.]

Indepenedent variables - the count is same for all variables which means that there is no missing values. we need to use standardscaler to standraize the variables as there is not normally distributed.

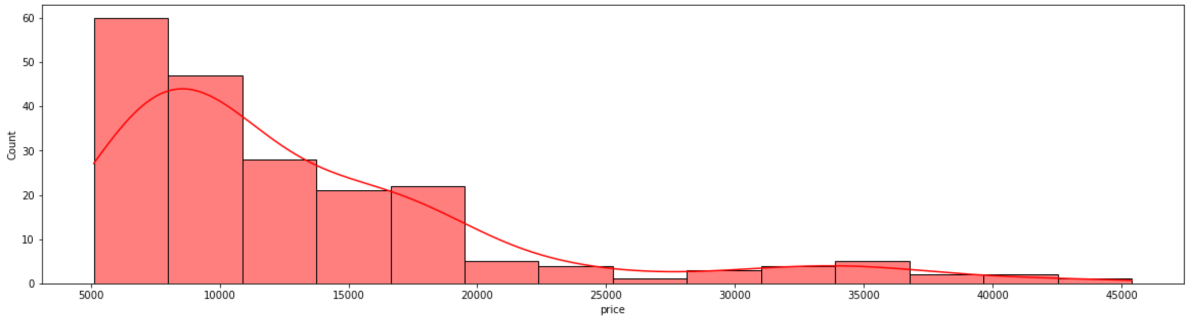
Let's know more about Price variable

In [6]:
```
df.price.describe()
```

Out[6]:
```
count      205.000000
mean     13276.710571
std       7988.852332
min       5118.000000
25%       7788.000000
50%      10295.000000
75%      16503.000000
max      45400.000000
Name: price, dtype: float64
```

In [7]:
```
plt.figure(figsize=(20,5))
sns.histplot(df["price"],kde=True,color="red")
```

Out[7]:
```
<AxesSubplot:xlabel='price', ylabel='Count'>
```

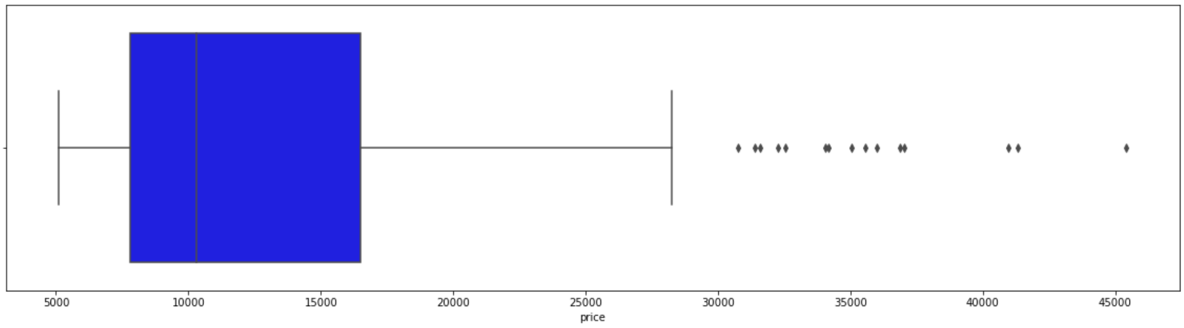


Price is not normally distributed.

In [8]:
```
plt.figure(figsize=(20,5))
sns.boxplot(df.price,color="blue")
```

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

Out[8]: `<AxesSubplot:xlabel='price'>`



As there is high price from the mean price

Selection the variables

Performing ANOVA to choose categorical variable

In [9]:
```
model = ols('price ~ CarName', data=df).fit()
anova_table = sm.stats.anova_lm(model, typ=2)
print(anova_table)
```

```
                 sum_sq     df         F        PR(>F)
CarName    1.243317e+10  146.0  8.421908  6.414987e-16
Residual   5.864709e+08   58.0       NaN           NaN
```

- CarName is 1.243317e+10, which indicates that the CarName variable explains a large amount of the variability in the data.
- There are 146 degrees of freedom, indicating that there are 146 groups in the dataset.
- F-value is 8.421908, which is quite large and indicates that there is likely a significant difference between the means of the groups.
- PR(>F), the p-value is 6.414987e-16, which is much smaller than the significance level (usually 0.05), indicating that there is strong evidence against the null hypothesis and that the CarName variable is a significant predictor of the outcome.

- Residual represents the sum of squares for the residual or error term, which is the variability in the data that is not explained by the independent variable (CarName). In this case, the residual sum of squares is 5.864709e+08, which indicates that there is still a significant amount of unexplained variability in the data.

p-value of 6.414987e-16. This suggests that the "CarName" variable has a significant effect on the outcome variable you are analyzing.

In [10]:
```
model = ols('price ~ fueltype', data=df).fit()
anova_table = sm.stats.anova_lm(model, typ=2)
print(anova_table)
```

```
                sum_sq     df         F    PR(>F)
fueltype  1.454053e+08    1.0  2.292741  0.131536
Residual  1.287423e+10  203.0       NaN       NaN
```

- sum_sq indicates that it explains small amount of variabilty in the model
- F value is signifivcantly less there is no significant difference between the group.
- p value is 0.131536 which is high

There is no need to consider the variable fueltype in the analysis.😔

In [11]:
```
model = ols('price ~ aspiration', data=df).fit()
anova_table = sm.stats.anova_lm(model, typ=2)
print(anova_table)
```

```
                  sum_sq     df         F  PR(>F)
aspiration  4.121724e+08    1.0  6.636622  0.0107
Residual    1.260747e+10  203.0       NaN     NaN
```

p-value is less than the typical significance level of 0.05 therefor we can consider "aspiration" in our study.

In [12]:
```
model = ols('price ~ doornumber', data=df).fit()
anova_table = sm.stats.anova_lm(model, typ=2)
print(anova_table)
```

```
                  sum_sq     df         F    PR(>F)
doornumber  1.319520e+07    1.0  0.205946  0.650448
Residual    1.300644e+10  203.0       NaN       NaN
```

Don't need to consier the variable doornumber

In [13]:
```
model = ols('price ~ carbody', data=df).fit()
anova_table = sm.stats.anova_lm(model, typ=2)
print(anova_table)
```

```
                sum_sq     df         F    PR(>F)
carbody   1.801997e+09    4.0  8.031976  0.000005
Residual  1.121764e+10  200.0       NaN       NaN
```

Yes we can consider this variable because of the above 😁

In [14]:
```
model = ols('price ~ drivewheel', data=df).fit()
anova_table = sm.stats.anova_lm(model, typ=2)
print(anova_table)
```

```
                  sum_sq     df          F        PR(>F)
drivewheel  5.344065e+09    2.0  70.320553  6.632887e-24
Residual    7.675574e+09  202.0        NaN           NaN
```

Yes we can consider this variable because of the above 😁

In [15]:
```
model = ols('price ~ enginelocation', data=df).fit()
anova_table = sm.stats.anova_lm(model, typ=2)
print(anova_table)
```

```
                      sum_sq     df         F    PR(>F)
enginelocation  1.374973e+09    1.0  23.96974  0.000002
Residual        1.164467e+10  203.0       NaN       NaN
```

Yes we can consider this variable enginelocation because of the above 😁

In [16]:
```
model = ols('price ~ enginetype', data=df).fit()
anova_table = sm.stats.anova_lm(model, typ=2)
print(anova_table)
```

```
                  sum_sq     df        F        PR(>F)
enginetype  2.880743e+09    6.0  9.37622  4.692665e-09
Residual    1.013890e+10  198.0      NaN           NaN
```

Yes we can consider this variable enginetype because of the above 😁

In [17]:
```
model = ols('price ~ cylindernumber', data=df).fit()
anova_table = sm.stats.anova_lm(model, typ=2)
print(anova_table)
```

```
                      sum_sq     df          F        PR(>F)
cylindernumber  8.275757e+09    6.0  57.568881  8.065780e-41
Residual        4.743882e+09  198.0        NaN           NaN
```

We include cylindernumber in study

In [18]:
```
model = ols('price ~ fuelsystem', data=df).fit()
anova_table = sm.stats.anova_lm(model, typ=2)
print(anova_table)
```

```
                  sum_sq     df          F        PR(>F)
fuelsystem  4.651199e+09    7.0  15.641865  2.990386e-16
Residual    8.368441e+09  197.0        NaN           NaN
```

We have to include the fuelsystem in the study

Therefore using ANOVA we need to consider all categorical variables except the fueltype😎.

But the probelm there comes is Multicollinearity.

In [19]:
```
df=df.drop(["fueltype"],axis=1)
```

In [20]:
```
df
```

Out[20]:

	car_ID	symboling	CarName	aspiration	doornumber	carbody	drivewheel	enginelocat
0	1	3	alfa-romero giulia	std	two	convertible	rwd	fı
1	2	3	alfa-romero stelvio	std	two	convertible	rwd	fı
2	3	1	alfa-romero Quadrifoglio	std	two	hatchback	rwd	fı
3	4	2	audi 100 ls	std	four	sedan	fwd	fı
4	5	2	audi 100ls	std	four	sedan	4wd	fı
...	...	...	...	...	...	...	...	
200	201	-1	volvo 145e (sw)	std	four	sedan	rwd	fı
201	202	-1	volvo 144ea	turbo	four	sedan	rwd	fı
202	203	-1	volvo 244dl	std	four	sedan	rwd	fı
203	204	-1	volvo 246	turbo	four	sedan	rwd	fı
204	205	-1	volvo 264gl	turbo	four	sedan	rwd	fı

205 rows × 25 columns

- Multicollinearity

In [21]:
```
label=LabelEncoder()
df["CarName"]=label.fit_transform(df["CarName"])
```

In [22]:
```
label=LabelEncoder()
df["cylindernumber"]=label.fit_transform(df["cylindernumber"])
```

In [23]:
```
label=LabelEncoder()
df["enginelocation"]=label.fit_transform(df["enginelocation"])
```

In [24]:
```
label=LabelEncoder()
df["drivewheel"]=label.fit_transform(df["drivewheel"])
```

In [25]:
```
label=LabelEncoder()
df["carbody"]=label.fit_transform(df["carbody"])
```

In [26]:
```
label=LabelEncoder()
df["doornumber"]=label.fit_transform(df["doornumber"])
```

In [27]:
```
label=LabelEncoder()
df["aspiration"]=label.fit_transform(df["aspiration"])
```

In [28]:
```
label=LabelEncoder()
df["drivewheel"]=label.fit_transform(df["drivewheel"])
```

In [29]:
```
label=LabelEncoder()
df["enginetype"]=label.fit_transform(df["enginetype"])
```

In [30]:
```
df.fuelsystem.unique()
```

Out[30]:
```
array(['mpfi', '2bbl', 'mfi', '1bbl', 'spfi', '4bbl', 'idi', 'spdi'],
      dtype=object)
```

In [31]:
```
df["fuelsystem"]=df["fuelsystem"].map({"mpfi":1,"2bbl":2,"1bbl":3,"4bbl":4,"spdi":
```

In [32]:
```
df=df.drop(["fuelsystem"],axis=1)
```

In [33]:
```
df
```

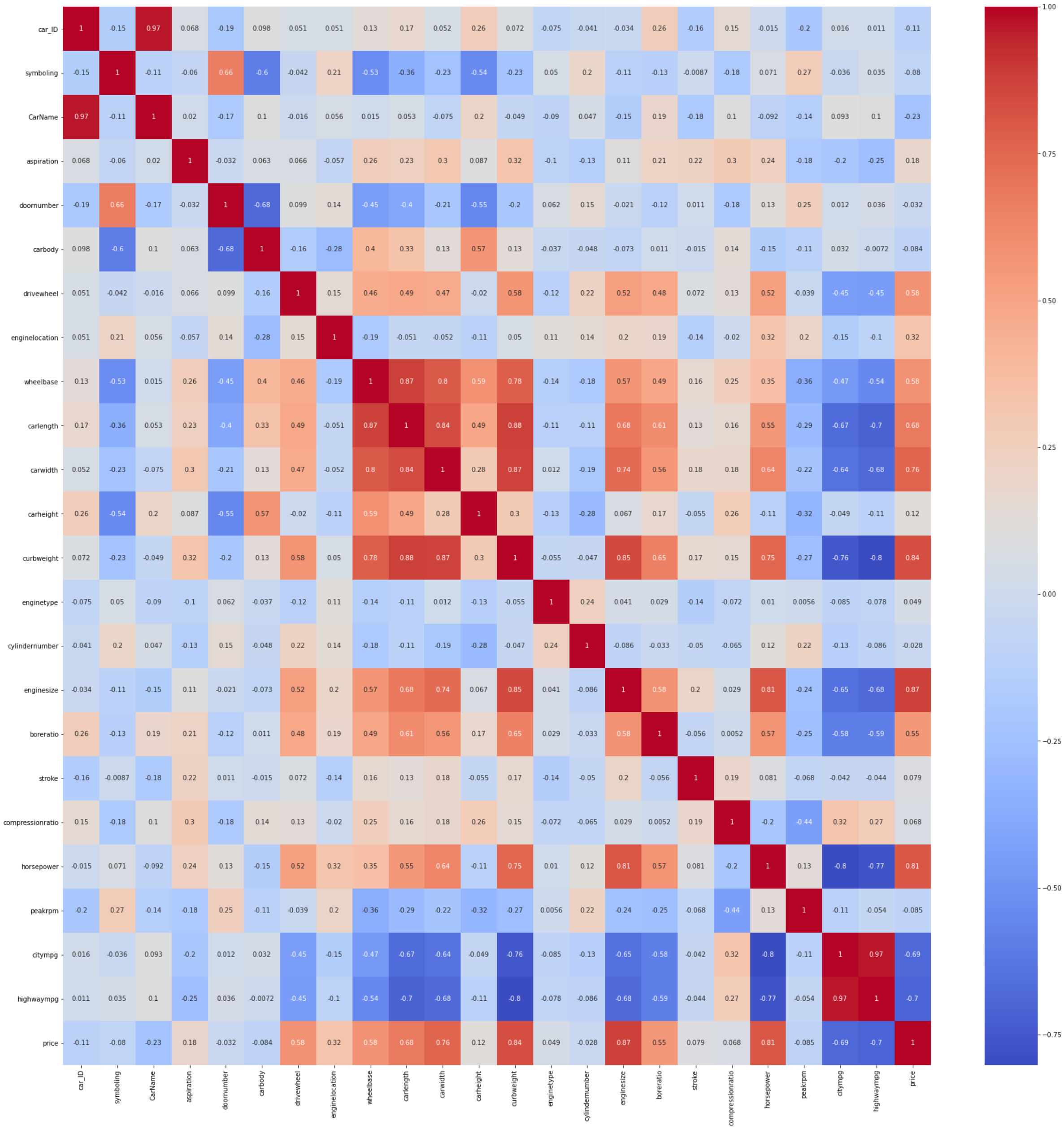
Out[33]:

	car_ID	symboling	CarName	aspiration	doornumber	carbody	drivewheel	enginelocation
0	1	3	2	0	1	0	2	0
1	2	3	3	0	1	0	2	0
2	3	1	1	0	1	2	2	0
3	4	2	4	0	0	3	1	0
4	5	2	5	0	0	3	0	0
...	...	...	...	...	...	...	...	...
200	201	-1	139	0	0	3	2	0
201	202	-1	138	1	0	3	2	0
202	203	-1	140	0	0	3	2	0
203	204	-1	142	1	0	3	2	0
204	205	-1	143	1	0	3	2	0

205 rows × 24 columns

In [34]:
```
plt.figure(figsize=(30,30))
sns.heatmap(df.corr(),cmap="coolwarm",annot=True)
```

Out[34]:
```
<AxesSubplot:>
```



Through correlation map : there is stron relation between

- 🙄 `CarName and CarID`
- 🙄 `curbweight and carlength and carweight`
- 🙄 `curbweight with enginesize`
- 🙄 `highwaympg and citympg`
- 🙄 `enginesize with price`

Use if VIF (variation inflation factor)

In [35]:
```
#drop price (dependent variable to check multicollinearity)
vif_df=df.drop("price",axis=1)
```

In [36]:
```
vif_df
```

Out[36]:

	car_ID	symboling	CarName	aspiration	doornumber	carbody	drivewheel	enginelocation
0	1	3	2	0	1	0	2	0
1	2	3	3	0	1	0	2	0
2	3	1	1	0	1	2	2	0
3	4	2	4	0	0	3	1	0
4	5	2	5	0	0	3	0	0
...	...	...	...	...	...	...	...	...
200	201	-1	139	0	0	3	2	0
201	202	-1	138	1	0	3	2	0
202	203	-1	140	0	0	3	2	0
203	204	-1	142	1	0	3	2	0
204	205	-1	143	1	0	3	2	0

205 rows × 23 columns

In [37]:

```
vif_data = pd.DataFrame()
vif_data['feature'] = vif_df.columns
vif_data['VIF'] = [variance_inflation_factor(vif_df.values, i) for i in range(vif_
vif_data
```

Out[37]:

	feature	VIF
0	car_ID	107.995455
1	symboling	3.966694
2	CarName	123.933712
3	aspiration	2.907535
4	doornumber	5.024581
5	carbody	28.761413
6	drivewheel	17.174009
7	enginelocation	1.750043
8	wheelbase	2777.239608
9	carlength	2311.173735
10	carwidth	4299.425315
11	carheight	1163.346110
12	curbweight	449.827414
13	enginetype	15.147461
14	cylindernumber	15.012311
15	enginesize	137.311365
16	boreratio	347.252599
17	stroke	153.384730
18	compressionratio	20.164384
19	horsepower	108.093329
20	peakrpm	279.347593
21	citympg	506.939637
22	highwaympg	596.062703

In [38]:

```
df=df.drop(["car_ID","curbweight","citympg","enginesize"],axis=1)
```

In [39]:

```
df
```

Out[39]:

	symboling	CarName	aspiration	doornumber	carbody	drivewheel	enginelocation	wheelba
0	3	2	0	1	0	2	0	8
1	3	3	0	1	0	2	0	8
2	1	1	0	1	2	2	0	9
3	2	4	0	0	3	1	0	9
4	2	5	0	0	3	0	0	9
...	...	...	...	...	...	...	...	
200	-1	139	0	0	3	2	0	10
201	-1	138	1	0	3	2	0	10
202	-1	140	0	0	3	2	0	10
203	-1	142	1	0	3	2	0	10
204	-1	143	1	0	3	2	0	10

205 rows × 20 columns

In [40]:

```
#drop price (dependent variable to check multicollienarity)
vif_df1=df.drop("price",axis=1)
```

In [41]:

```
vif_data = pd.DataFrame()
vif_data['feature'] = vif_df1.columns
vif_data['VIF'] = [variance_inflation_factor(vif_df1.values, i) for i in range(vif_
vif_data
```

Out[41]:

	feature	VIF
0	symboling	3.791713
1	CarName	5.820628
2	aspiration	1.866386
3	doornumber	4.807985
4	carbody	28.333542
5	drivewheel	16.550477
6	enginelocation	1.687555
7	wheelbase	2581.076500
8	carlength	2055.289098
9	carwidth	4126.692673
10	carheight	1123.439200
11	enginetype	13.208384
12	cylindernumber	11.305128
13	boreratio	342.234667
14	stroke	131.433165
15	compressionratio	15.706597
16	horsepower	39.354194
17	peakrpm	192.212943
18	highwaympg	89.999288

we dropped some variables but still multicollinerity effects the data....so we can use transformation method to reduce.

we can combine carwidth and carheight and create new variable cararea (linear transformation)

In [42]:

```
#drop price (dependent variable to check multicolliearity)
vif_df2=df.drop("price",axis=1)
```

In [43]:

```
#drop price (dependent variable to check multicolliearity)
vif_df2=df.drop(["price"],axis=1)
```

In [44]:

```
vif_data = pd.DataFrame()
vif_data['feature'] = vif_df2.columns
vif_data['VIF'] = [variance_inflation_factor(vif_df2.values, i) for i in range(vif_
vif_data
```

Out[44]:

	feature	VIF
0	symboling	3.791713
1	CarName	5.820628
2	aspiration	1.866386
3	doornumber	4.807985
4	carbody	28.333542
5	drivewheel	16.550477
6	enginelocation	1.687555
7	wheelbase	2581.076500
8	carlength	2055.289098
9	carwidth	4126.692673
10	carheight	1123.439200
11	enginetype	13.208384
12	cylindernumber	11.305128
13	boreratio	342.234667
14	stroke	131.433165
15	compressionratio	15.706597
16	horsepower	39.354194
17	peakrpm	192.212943
18	highwaympg	89.999288

In [45]:
```
df=df.drop(["symboling","aspiration","carheight","boreratio","peakrpm","carlength"
```

In [46]:
```
df
```

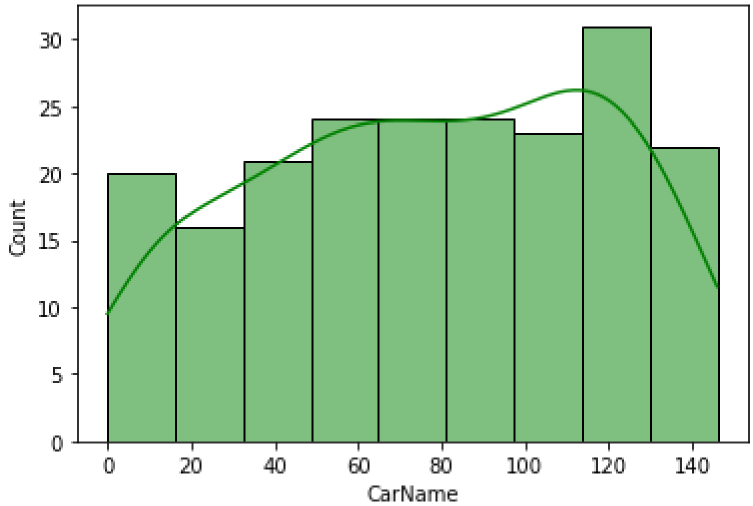
Out[46]:

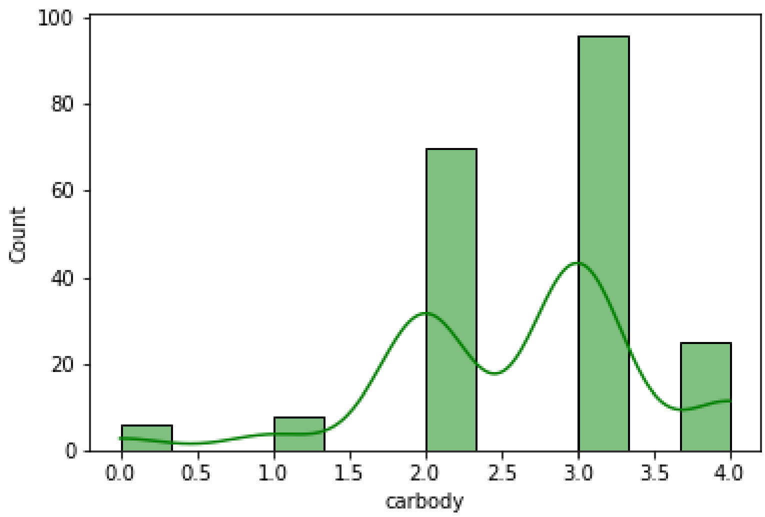
	CarName	carbody	enginelocation	carwidth	enginetype	horsepower	price
0	2	0	0	64.1	0	111	13495.0
1	3	0	0	64.1	0	111	16500.0
2	1	2	0	65.5	5	154	16500.0
3	4	3	0	66.2	3	102	13950.0
4	5	3	0	66.4	3	115	17450.0
...	...	...	...	...	...	...	...
200	139	3	0	68.9	3	114	16845.0
201	138	3	0	68.8	3	160	19045.0
202	140	3	0	68.9	5	134	21485.0
203	142	3	0	68.9	3	106	22470.0
204	143	3	0	68.9	3	114	22625.0

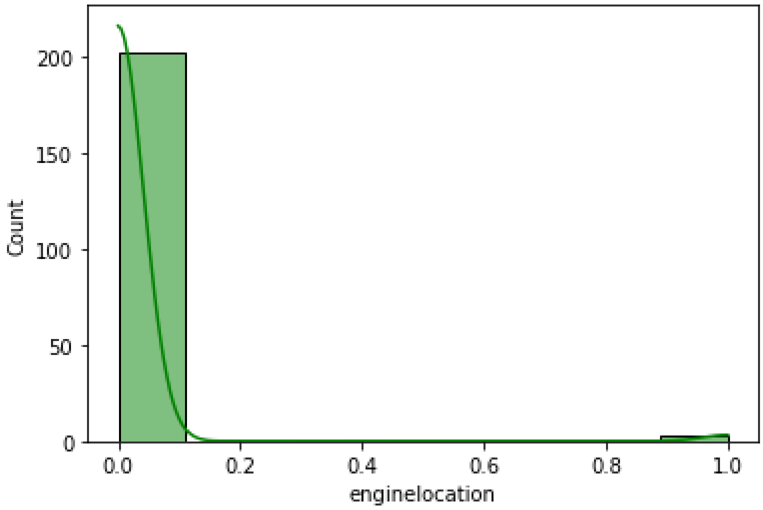
205 rows × 7 columns

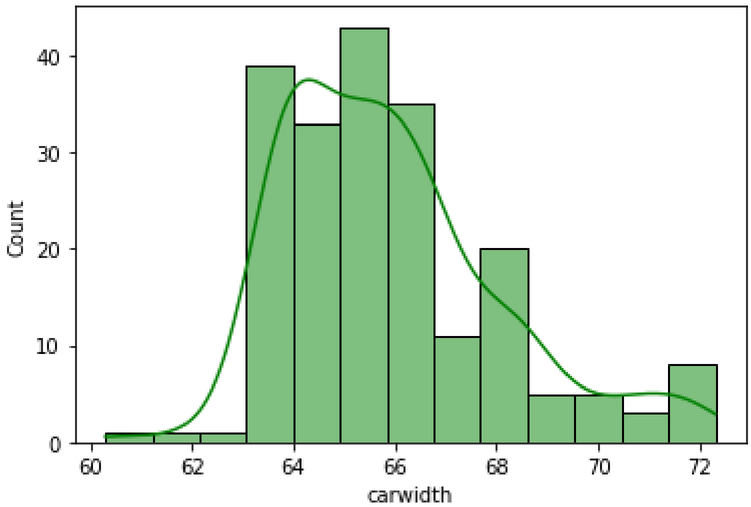
In [47]:

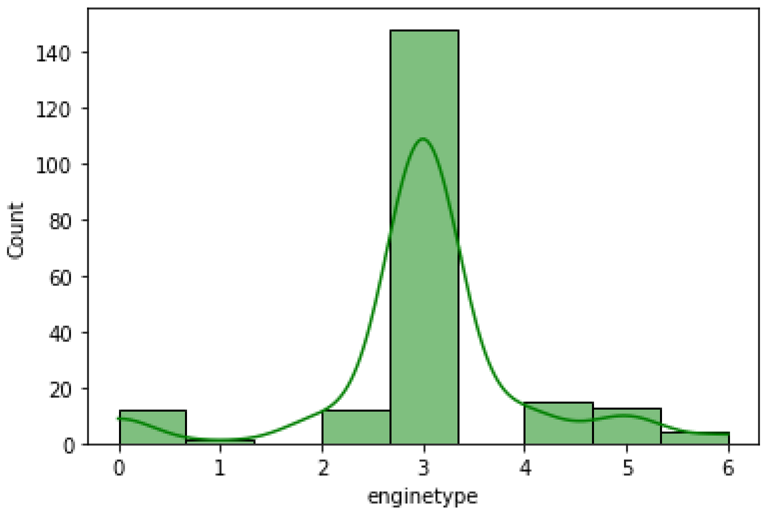
```
for i in df.columns:
    sns.histplot(df[i],kde=True,color="green")
    plt.show()
```

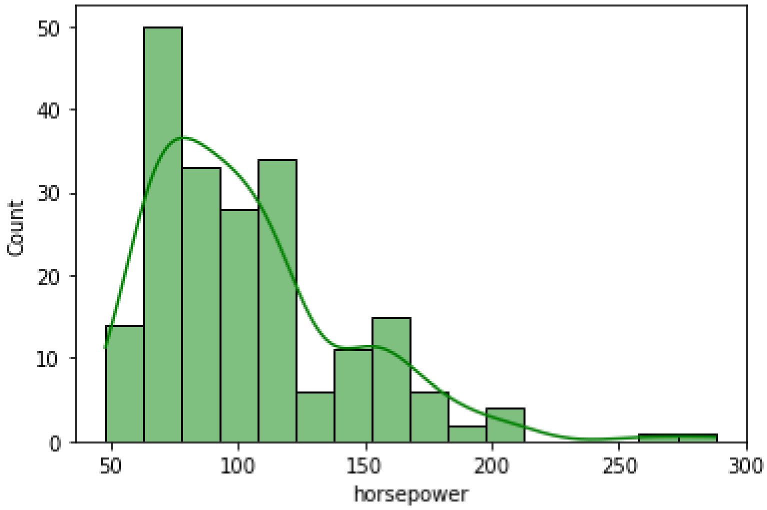


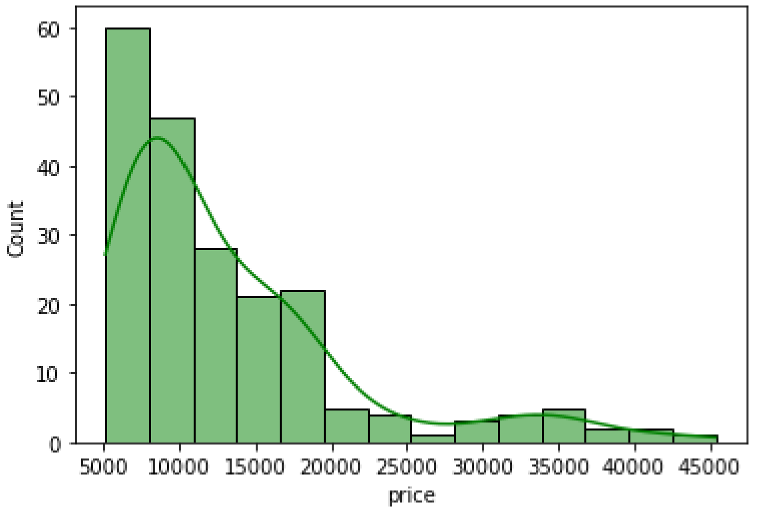



Count
200
150
100
50
0
0.0
0.2
0.4
0.6
0.8
1.0
enginelocation


Count
40
30
20
10
0
60
62
64
66
68
70
72
carwidth


Count
140
120
100
80
60
40
20
0
0
1
2
3
4
5
6
enginetype





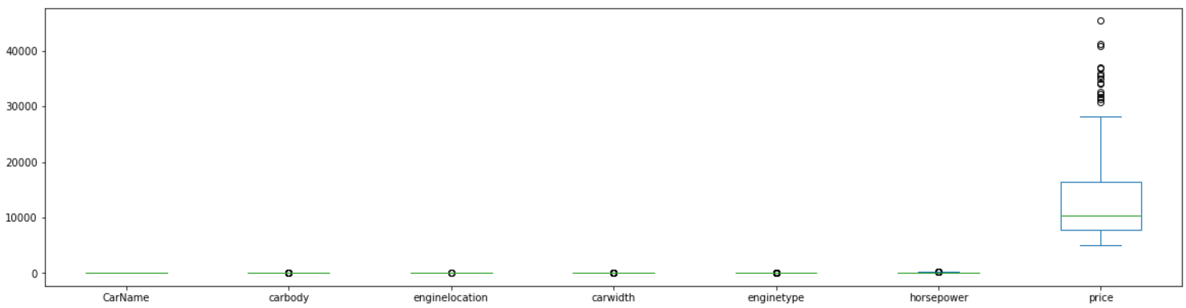
In [48]:
```
#for i in df.columns:
    #df[i]=np.log(df[i]+1)
    #sns.histplot(df1[i],kde=True,color="violet")
    #plt.show()
```

Log transformation didn't helped us!!!!!!!😪😯😯😯

Outlier Detection

In [49]:
```
df.plot(kind="box",figsize=(20,5))
```

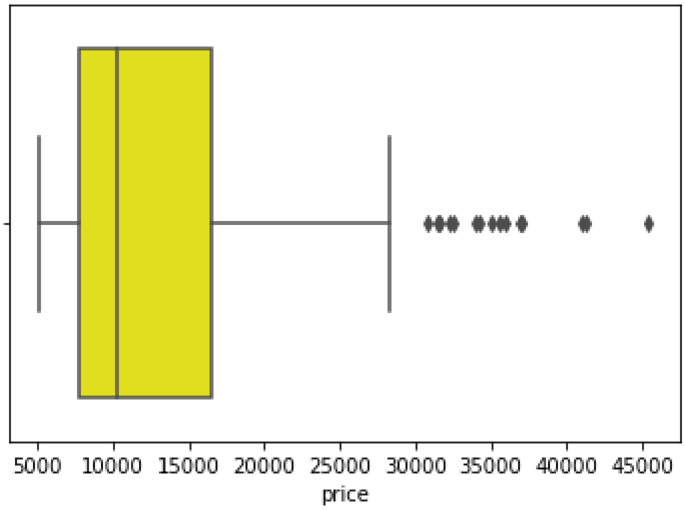
Out[49]: <AxesSubplot:>



In [50]:
```
sns.boxplot(df["price"],color="yellow")
```

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

Out[50]:
```
<AxesSubplot:xlabel='price'>
```



Model Buiding

In [51]:
```
from sklearn.linear_model import LinearRegression
```

In [52]:
```
x=df.drop(["price"],axis=1)
y=df["price"]
```

In [53]:
```
from sklearn.preprocessing import StandardScaler
```

In [54]:
```
scaled=StandardScaler()
x_scaled=scaled.fit_transform(x)
```

In [55]:
```
x_scaled[:3]
```

Out[55]:
```
array([[-1.83822103, -3.05097525, -0.12186667, -0.84478235, -2.86510549,
         0.17448278],
       [-1.81377978, -3.05097525, -0.12186667, -0.84478235, -2.86510549,
         0.17448278],
       [-1.86266229, -0.71720687, -0.12186667, -0.19056612,  1.88688986,
         1.26453643]])
```

In [56]:
```
y[:3]
```

Out[56]:
```
0    13495.0
1    16500.0
2    16500.0
Name: price, dtype: float64
```

In [57]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x_scaled,y,test_size=0.2)
```

You can also view the documentation for the LinearRegression class by running help(LinearRegression) or by visiting the scikit-learn website.

In [58]:
```
model=LinearRegression(fit_intercept=True)
```

In [59]:
```
model.fit(x_train,y_train)
```

Out[59]:
```
▾ LinearRegression
LinearRegression()
```

In [60]:
```
print('Coefficients:', model.coef_)
print('Intercept:', model.intercept_)
```

```
Coefficients: [-1312.03047863  -216.46203106  1853.41634306  4051.71539085
   -61.07855823  3155.69010618]
Intercept: 13167.510419934066
```

when x=0 the price = 13167.510419934066 therefore the starting price of the car is 13167.510419934066

In [61]:
```
y_pred=model.predict(x_test)
```

In [62]:
```
from sklearn.metrics import r2_score
r2 = r2_score(y_test, y_pred)
print("R-squared value:", r2)
```

```
R-squared value: 0.8890044170241304
```

R-squared value of 0.89 or higher is considered to be a good fit for a model, but the interpretation of the value depends on the context of the problem and the field of application.

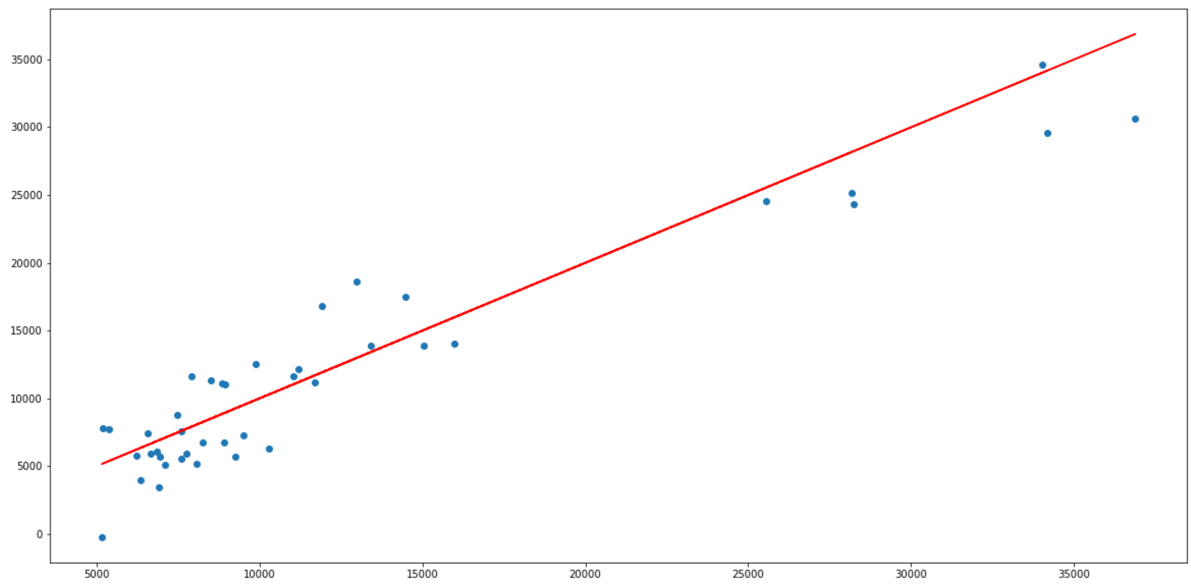
There it can explain 0.73 of total variance in the dependent variable.

In [63]:
```
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(y_test, y_pred)
print("Mean squared error:", mse)
```

```
Mean squared error: 7815647.671916967
```

In [64]:
```
plt.figure(figsize=(20,10))
plt.scatter(y_test,y_pred)
plt.plot(y_test,y_test,"r")
```

Out[64]:
```
[<matplotlib.lines.Line2D at 0x1e3aa290850>]
```



Since there is multicollinearity let us go for ridge regression and compare the model.

In [65]:
```
from sklearn.linear_model import Ridge
```

In [66]:
```
model=Ridge(alpha=4) #choose bets alpha value
model.fit(x_train,y_train)
```

Out[66]:
```
▾ Ridge
Ridge(alpha=4)
```

In [67]:
```
y_pred=model.predict(x_test)
```

In [68]:
```
from sklearn.metrics import r2_score
r2 = r2_score(y_test, y_pred)
print("R-squared value:", r2 )
```

```
R-squared value: 0.8889326092110726
```

In [69]:
```
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(y_test, y_pred)
print("Mean squared error:", mse)
```

```
Mean squared error: 7820703.950301237
```

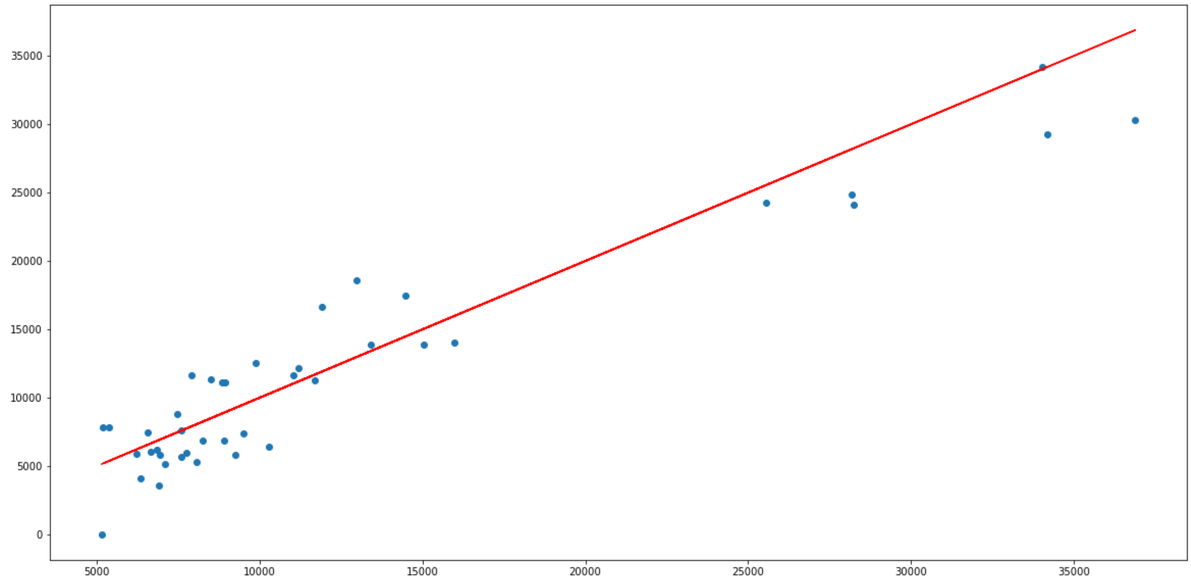
In [70]:
```
print('Coefficients:', model.coef_)
print('Intercept:', model.intercept_)
```

```
Coefficients: [-1279.8517459   -211.72396689  1789.44291557  3936.58215236
   -53.12699984  3168.30503263]
Intercept: 13170.809819055667
```

when x=0 the price = 13170.809819055667 therefore the starting price of the car is 13170.809819055667

In [71]:
```
plt.figure(figsize=(20,10))
plt.scatter(y_test,y_pred)
plt.plot(y_test,y_test,"r")
```

Out[71]:
```
[<matplotlib.lines.Line2D at 0x1e3a8197b50>]
```



In [72]:

```
# Plot the histogram of the error terms
fig = plt.figure()
sns.distplot((y_test - y_pred), bins = 20)
fig.suptitle('Error Terms Analysis', fontsize = 20)
plt.xlabel('Errors', fontsize = 18)
```

```
C:\Users\Dayana Vincent\anaconda3\lib\site-packages\seaborn\distributions.py:2551:
FutureWarning: `distplot` is a deprecated function and will be removed in a future
version. Please adapt your code to use either `displot` (a figure-level function w
ith similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[72]:

```
Text(0.5, 0, 'Errors')
```