

AUTOMOBILE PRICE PREDICTION

Step-1: Import all the required libraries which are used to train the model and visualise the data

```
In [1]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
import plotly
import plotly.express as px
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn import metrics
```

Step-2: Read the data

```
In [2]: df=pd.read_csv("/home/silpa/Downloads/Automobile_data.csv")
df
```

```
Out[2]:
```

| | symboling | normalized- losses | make | fuel- type | aspiration | num- of- doors | body- style | drive- wheels | engine- location | wheel- base | ... | engine siz |
|-----|-----------|-----------------------|-------------|---------------|------------|----------------------|----------------|------------------|---------------------|----------------|-----|---------------|
| 0 | 3 | ? | alfa-romero | gas | std | two | convertible | rwd | front | 88.6 | ... | 13 |
| 1 | 3 | ? | alfa-romero | gas | std | two | convertible | rwd | front | 88.6 | ... | 13 |
| 2 | 1 | ? | alfa-romero | gas | std | two | hatchback | rwd | front | 94.5 | ... | 15 |
| 3 | 2 | 164 | audi | gas | std | four | sedan | fwd | front | 99.8 | ... | 10 |
| 4 | 2 | 164 | audi | gas | std | four | sedan | 4wd | front | 99.4 | ... | 13 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 200 | -1 | 95 | volvo | gas | std | four | sedan | rwd | front | 109.1 | ... | 14 |
| 201 | -1 | 95 | volvo | gas | turbo | four | sedan | rwd | front | 109.1 | ... | 14 |
| 202 | -1 | 95 | volvo | gas | std | four | sedan | rwd | front | 109.1 | ... | 17 |
| 203 | -1 | 95 | volvo | diesel | turbo | four | sedan | rwd | front | 109.1 | ... | 14 |
| 204 | -1 | 95 | volvo | gas | turbo | four | sedan | rwd | front | 109.1 | ... | 14 |

205 rows × 26 columns

EXPLORATORY DATA ANALYSIS

Step-3: To know number of rows and columns in the data set

```
In [3]: print(df.shape)

(205, 26)
```

Step-4: Describe the dataset which shows the minimum value, maximum value, mean value, count, standard deviation, etc.

In [4]: df.describe()

Out[4]:

| | symboling | wheel-base | length | width | height | curb-weight | engine-size | compression-ratio | |
|-------|------------|------------|------------|------------|------------|-------------|-------------|-------------------|----|
| count | 205.000000 | 205.000000 | 205.000000 | 205.000000 | 205.000000 | 205.000000 | 205.000000 | 205.000000 | 20 |
| mean | 0.834146 | 98.756585 | 174.049268 | 65.907805 | 53.724878 | 2555.565854 | 126.907317 | 10.142537 | 2 |
| std | 1.245307 | 6.021776 | 12.337289 | 2.145204 | 2.443522 | 520.680204 | 41.642693 | 3.972040 | |
| min | -2.000000 | 86.600000 | 141.100000 | 60.300000 | 47.800000 | 1488.000000 | 61.000000 | 7.000000 | 1 |
| 25% | 0.000000 | 94.500000 | 166.300000 | 64.100000 | 52.000000 | 2145.000000 | 97.000000 | 8.600000 | 1 |
| 50% | 1.000000 | 97.000000 | 173.200000 | 65.500000 | 54.100000 | 2414.000000 | 120.000000 | 9.000000 | 2 |
| 75% | 2.000000 | 102.400000 | 183.100000 | 66.900000 | 55.500000 | 2935.000000 | 141.000000 | 9.400000 | 3 |
| max | 3.000000 | 120.900000 | 208.100000 | 72.300000 | 59.800000 | 4066.000000 | 326.000000 | 23.000000 | 4 |

Step-5: Checking for missing values

In [5]: df.isna().sum()

Out[5]:

| | |
|-------------------|---|
| 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 | |

Step-6: There is no null values in the dataset but there is '?' suymbol.So replacing them in to np.nan.

In [6]: for colm in df.columns:
df[colm].replace({'?':np.nan},inplace=True)
df

Out[6]:

| | symboling | normalized-losses | make | fuel-type | aspiration | num-of-doors | body-style | drive-wheels | engine-location | wheel-base | ... | engine siz |
|---|-----------|-------------------|-------------|-----------|------------|--------------|-------------|--------------|-----------------|------------|-----|------------|
| 0 | 3 | NaN | alfa-romero | gas | std | two | convertible | rwd | front | 88.6 | ... | 13 |
| 1 | 3 | NaN | alfa-romero | gas | std | two | convertible | rwd | front | 88.6 | ... | 13 |

| | | | | | | | | | | | | |
|-----|-----|-----|-------------|--------|-------|------|-----------|-----|-------|-------|-----|-----|
| 2 | 1 | NaN | alfa-romero | gas | std | two | hatchback | rwd | front | 94.5 | ... | 15 |
| 3 | 2 | 164 | audi | gas | std | four | sedan | fwd | front | 99.8 | ... | 10 |
| 4 | 2 | 164 | audi | gas | std | four | sedan | 4wd | front | 99.4 | ... | 13 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 200 | -1 | 95 | volvo | gas | std | four | sedan | rwd | front | 109.1 | ... | 14 |
| 201 | -1 | 95 | volvo | gas | turbo | four | sedan | rwd | front | 109.1 | ... | 14 |
| 202 | -1 | 95 | volvo | gas | std | four | sedan | rwd | front | 109.1 | ... | 17 |
| 203 | -1 | 95 | volvo | diesel | turbo | four | sedan | rwd | front | 109.1 | ... | 14 |
| 204 | -1 | 95 | volvo | gas | turbo | four | sedan | rwd | front | 109.1 | ... | 14 |

205 rows × 26 columns

Step-7: Cheking missing values again

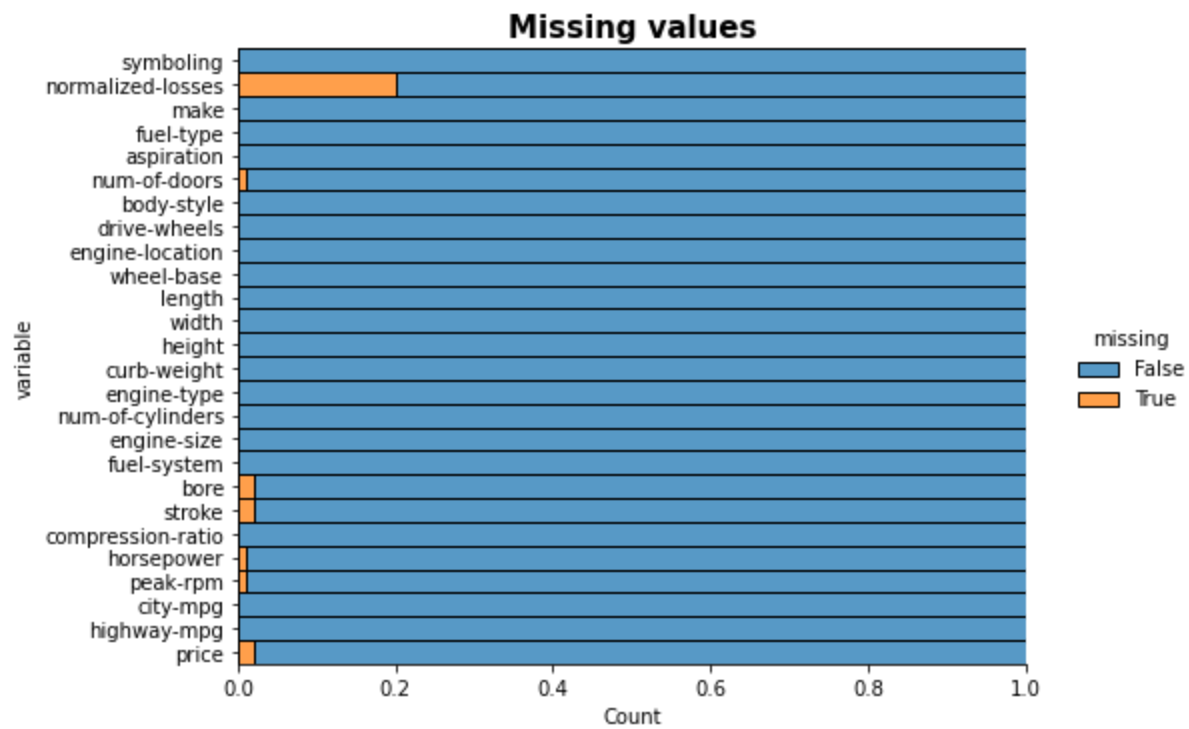
```
In [7]: df.isna().sum()
```

```
Out[7]: 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
```

Step-8: Plotting missing values

```
In [9]: # To create a heatmap of missing values of the df
sns.displot(data=df.isna().melt(value_name="missing"),
            y="variable",
            hue="missing",
            multiple="fill",
            height=5,
            aspect=1.5
)
plt.title("Missing values", fontsize=15, fontweight='bold')
```

```
Out[9]: Text(0.5, 1.0, 'Missing values')
```



Step-9: Handling the two missing values in 'num of doors'. So we are looking at the 'make' and 'body style' corresponding to the missing value and checking the same model's number of doors.

```
In [10]: df[(df["num-of-doors"].isna() == True)]
```

```
Out[10]:
```

| | symboling | normalized-losses | make | fuel-type | aspiration | num-of-doors | body-style | drive-wheels | engine-location | wheel-base | ... | engine-size | sy |
|----|-----------|-------------------|-------|-----------|------------|--------------|------------|--------------|-----------------|------------|-----|-------------|----|
| 27 | 1 | 148 | dodge | gas | turbo | NaN | sedan | fwd | front | 93.7 | ... | 98 | |
| 63 | 0 | NaN | mazda | diesel | std | NaN | sedan | fwd | front | 98.8 | ... | 122 | |

2 rows × 26 columns

Step-10: From the above data we can find that the 'body style' is sedan and 'make' is dodge and mazda. Then check the number of doors in dodge and mazda.

```
In [11]: df['num-of-doors'][(df['body-style']=='sedan') & (df['make']=='mazda')]
```

```
Out[11]:
```

| | |
|----|------|
| 53 | four |
| 54 | four |
| 60 | four |
| 62 | four |
| 63 | NaN |
| 65 | four |
| 66 | four |

Name: num-of-doors, dtype: object

```
In [12]: df['num-of-doors'][(df['body-style']=='sedan') & (df['make']=='dodge')]
```

```
Out[12]:
```

| | |
|----|------|
| 25 | four |
| 26 | four |
| 27 | NaN |

Name: num-of-doors, dtype: object

Step-11: Both have four doors. So, fill the missing values in 'num of doors' by four

```
In [13]: df['num-of-doors'] = df['num-of-doors'].fillna('four')
```

```
a=df['num-of-doors'].map({'two':2, 'four':4}) # dictionary mapping
df['num-of-doors']=a
df['num-of-doors'] = df['num-of-doors'].astype(str).astype(int) # converting datatype
```

Step-12: Filling the missing values in the numerical columns with mean

```
In [14]: ncol = ['normalized-losses', 'bore', 'stroke', 'horsepower', 'peak-rpm', 'price']

for col in ncol:
    df[col]=pd.to_numeric(df[col])
    df[col].fillna(df[col].mean(), inplace=True)
df.head()
```

```
Out[14]:
```

| | symboling | normalized-losses | make | fuel-type | aspiration | num-of-doors | body-style | drive-wheels | engine-location | wheel-base | ... | engine-size |
|---|-----------|-------------------|-------------|-----------|------------|--------------|-------------|--------------|-----------------|------------|-----|-------------|
| 0 | 3 | 122.0 | alfa-romero | gas | std | 2 | convertible | rwd | front | 88.6 | ... | 130 |
| 1 | 3 | 122.0 | alfa-romero | gas | std | 2 | convertible | rwd | front | 88.6 | ... | 130 |
| 2 | 1 | 122.0 | alfa-romero | gas | std | 2 | hatchback | rwd | front | 94.5 | ... | 152 |
| 3 | 2 | 164.0 | audi | gas | std | 4 | sedan | fwd | front | 99.8 | ... | 109 |
| 4 | 2 | 164.0 | audi | gas | std | 4 | sedan | 4wd | front | 99.4 | ... | 136 |

5 rows × 26 columns

Step-13: Checking missing value again

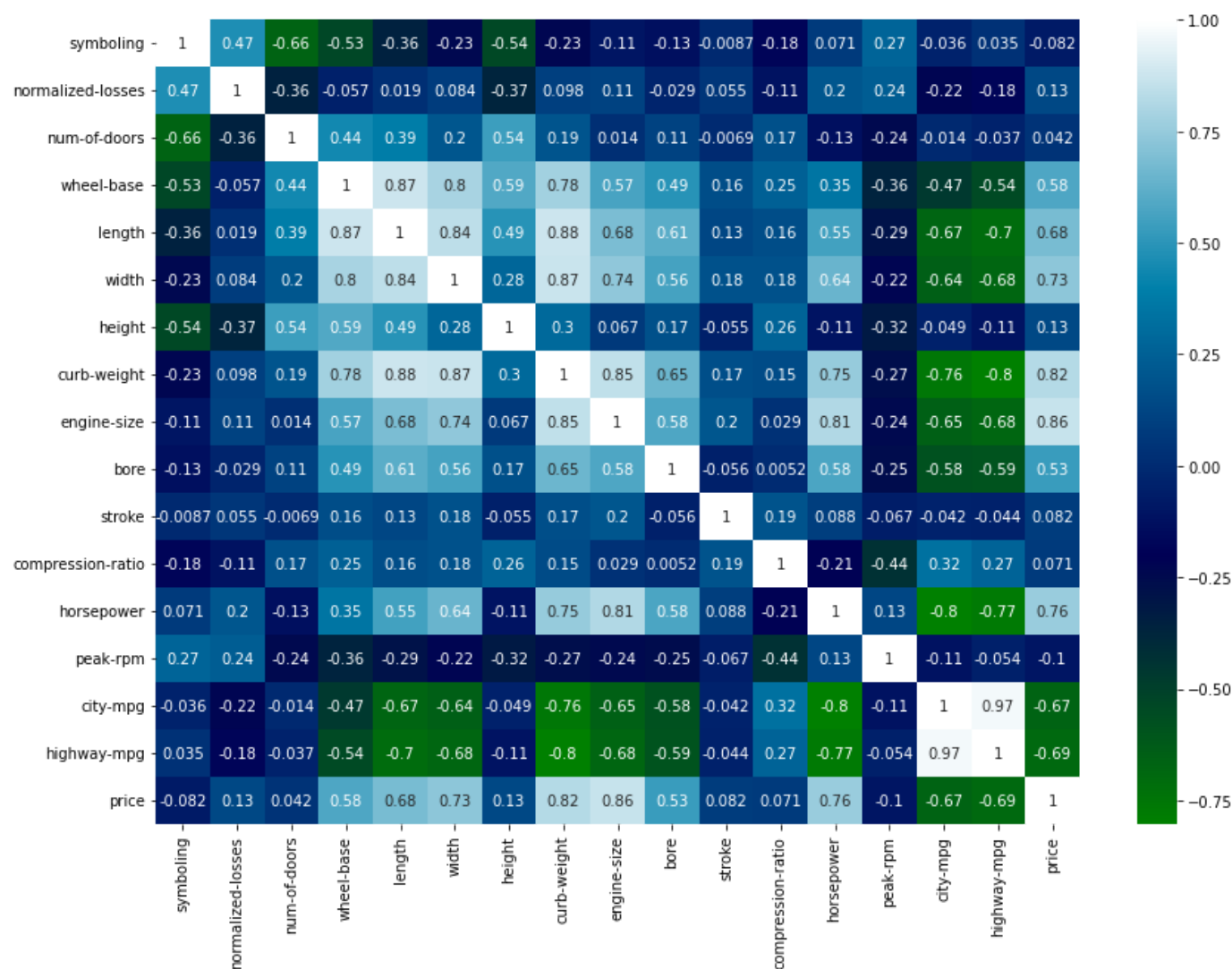
```
In [15]: print(df.isnull().sum())
```

```
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
```

Step-14: Checking the correlation between different variables

```
In [16]: plt.figure(figsize=(14,10))
sns.heatmap(df.corr(),annot=True,cmap="ocean")
```

```
Out[16]: <AxesSubplot:>
```



Positive Correlation:

** Price : wheel_base, length, width, curb_weight, engine_size, bore, horsepower

** wheelbase : length, width, height, curb_weight, engine_size, price

** horsepower : length, width, curb_weight, engine_size, bore, price

** Highway mpg : city mpg

Negative Correlation:

** Price : highway_mpg, city_mpg

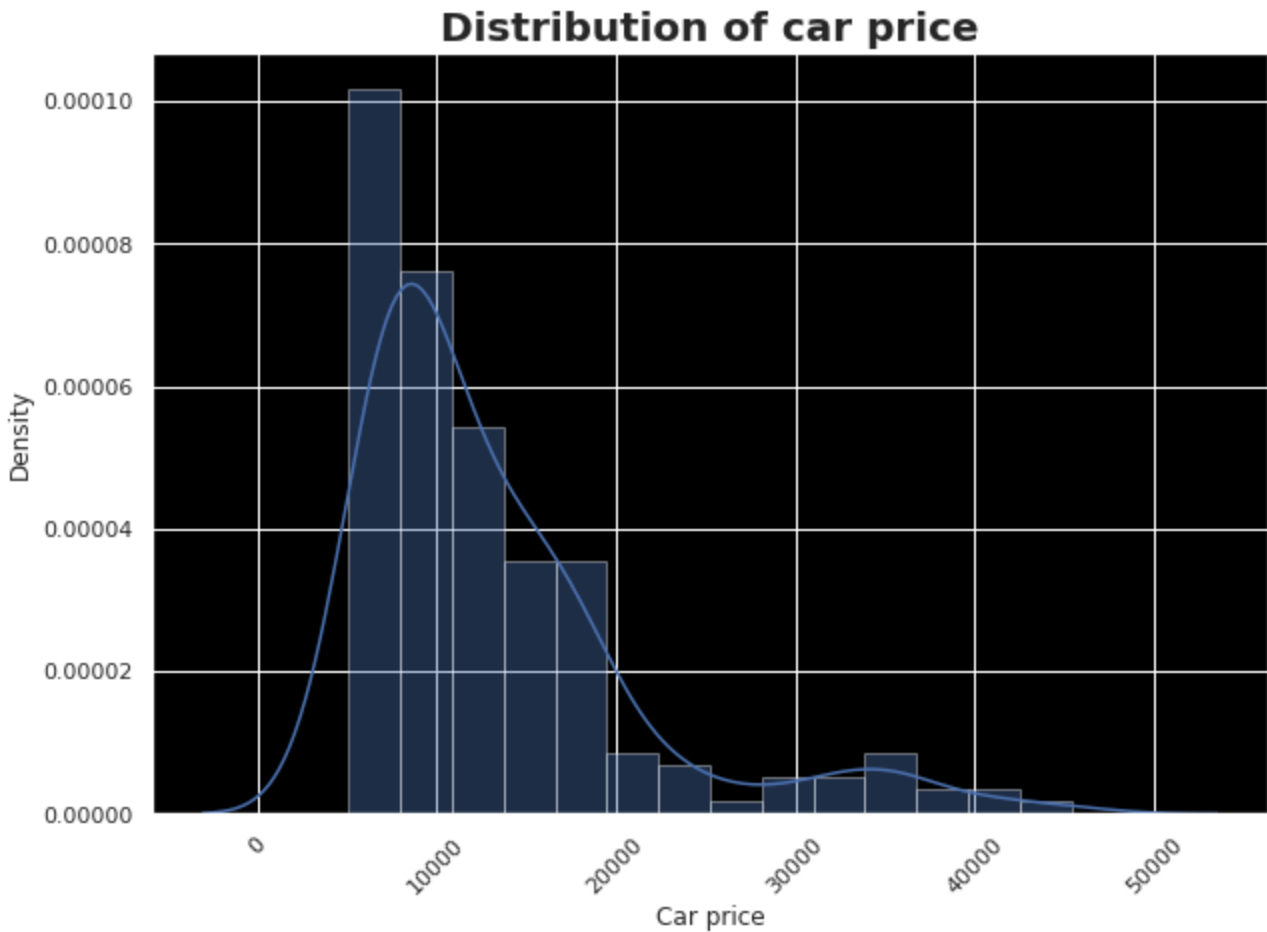
** highway_mpg : wheel base, length, width, curb_weight, engine_size, bore, horsepower, price

** city-mpg : wheel base, length, width, curb_weight, horsepower, engine_size, bore, price

Step-15: Plotting the disdtribution of car price

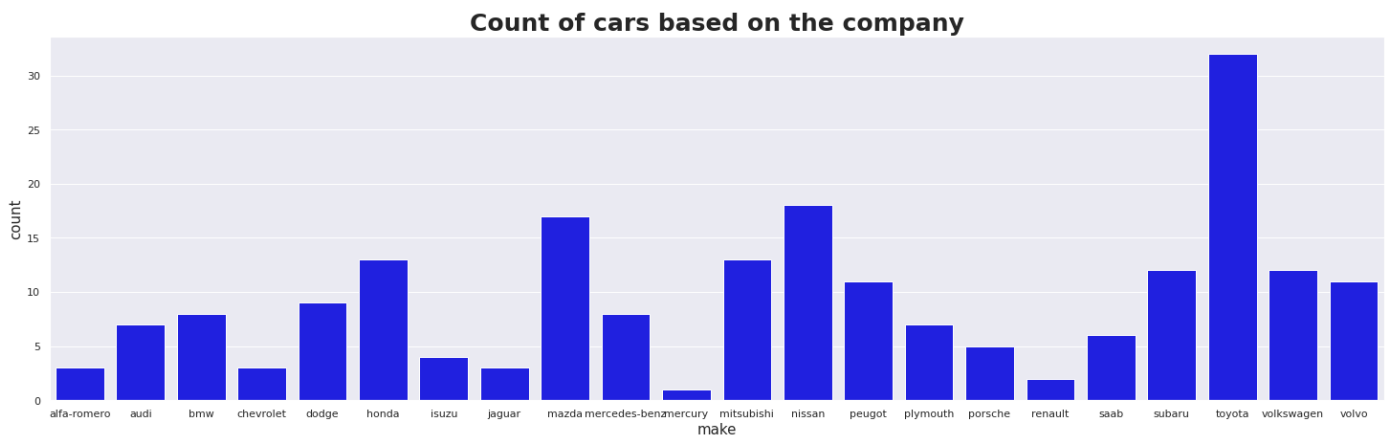
```
In [18]: ax=plt.axes()
ax.set(facecolor='black')
```

```
sns.set(rc={'figure.figsize':(10,7)},style='darkgrid')
ax.set_title('Distribution of car price',fontsize=20,fontweight='bold')
sns.distplot(df['price'])
plt.xticks(rotation=45)
plt.xlabel('Car price')
plt.show()
```



Step-16: Count plot of cars based on the company

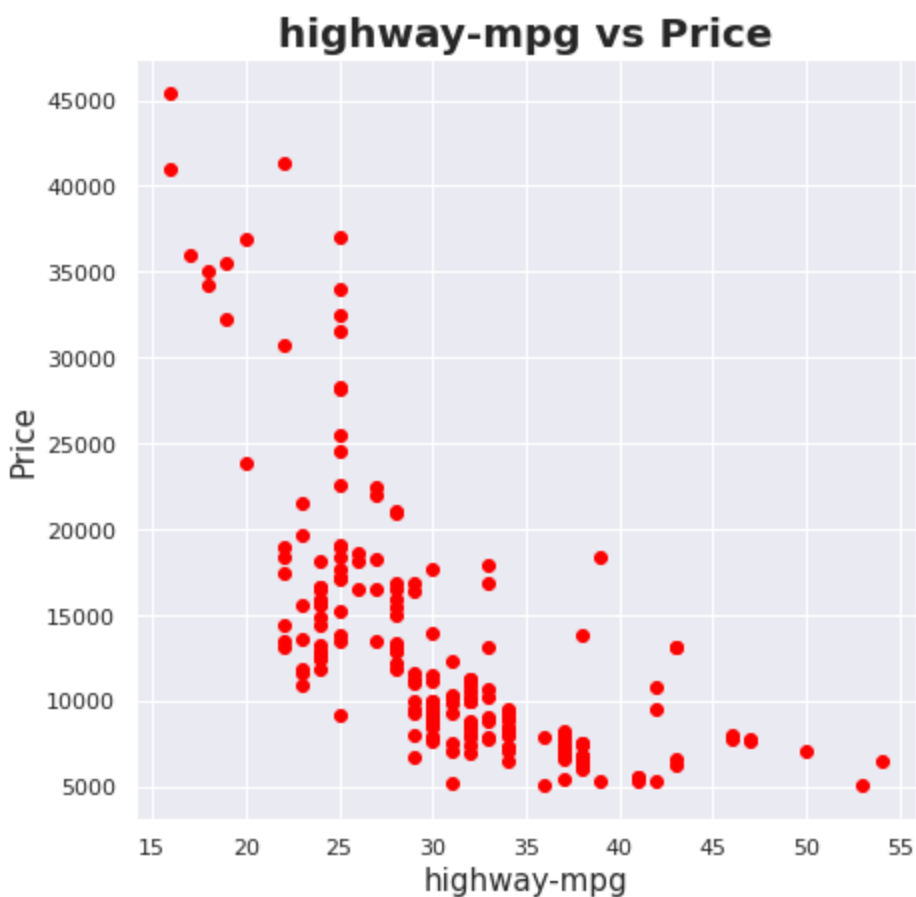
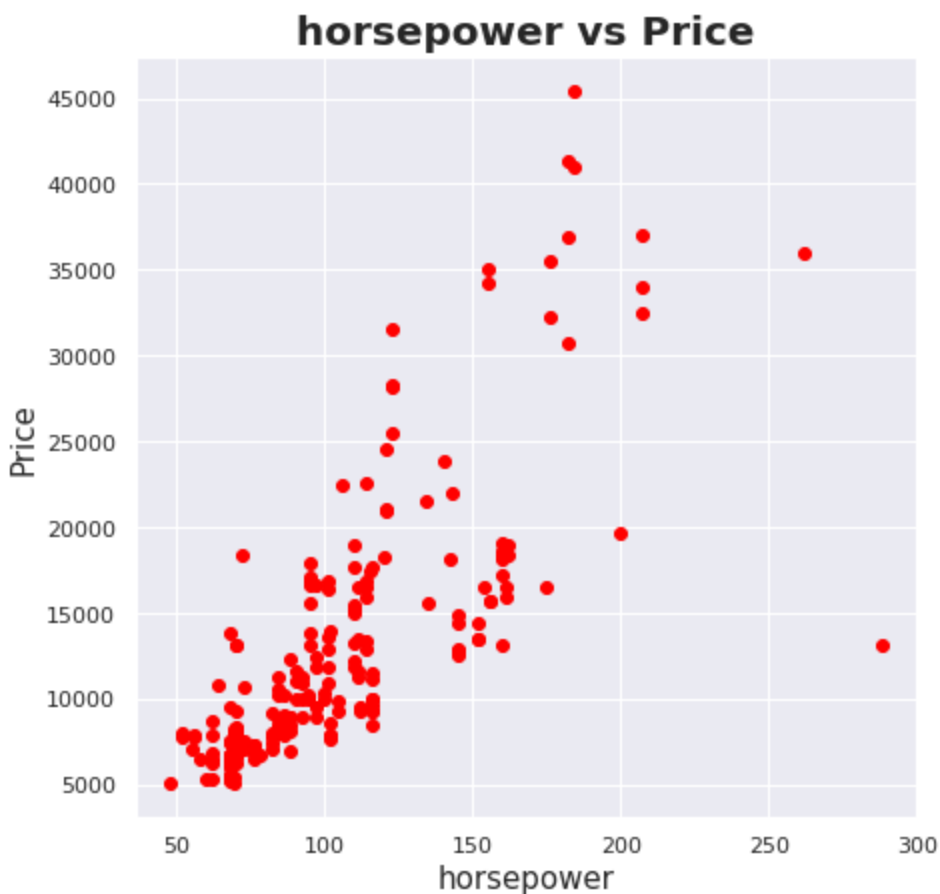
```
In [19]: plt.figure(figsize=(25,7))
sns.countplot(df['make'],color='blue')
plt.xlabel('make',fontsize=15)
plt.ylabel('count',fontsize=15)
plt.title('Count of cars based on the company',fontsize=25,fontweight='bold')
plt.show()
```

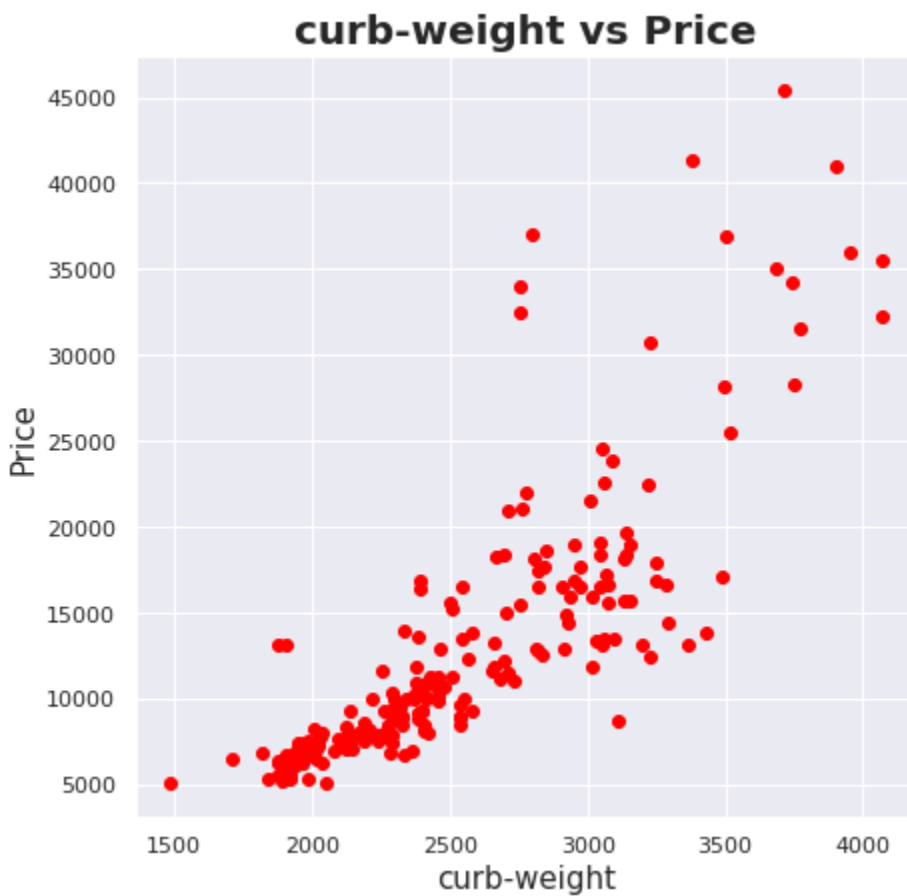
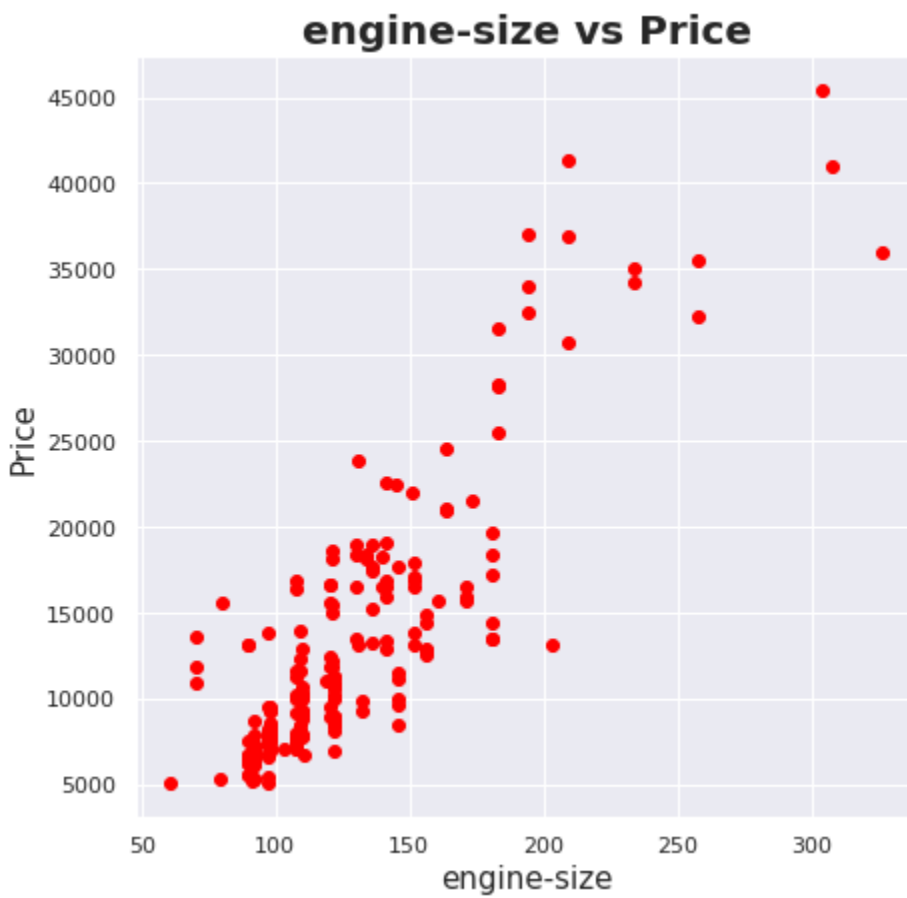


** From the graph we can see that, Toyota is the leading producer of cars followed by nissan and mazda

Step-17: Checking how the horse power,highway mpg,engine size and curb weight affect the price

```
In [20]: c=['horsepower','highway-mpg','engine-size','curb-weight']
for j in c:
    plt.figure(figsize=(7,7))
    plt.scatter(x=j,y='price',data=df,color='red')
    plt.xlabel(j,fontsize=15)
    plt.ylabel('Price',fontsize=15)
    plt.title(j+ ' vs Price',fontsize=20,fontweight='bold')
```

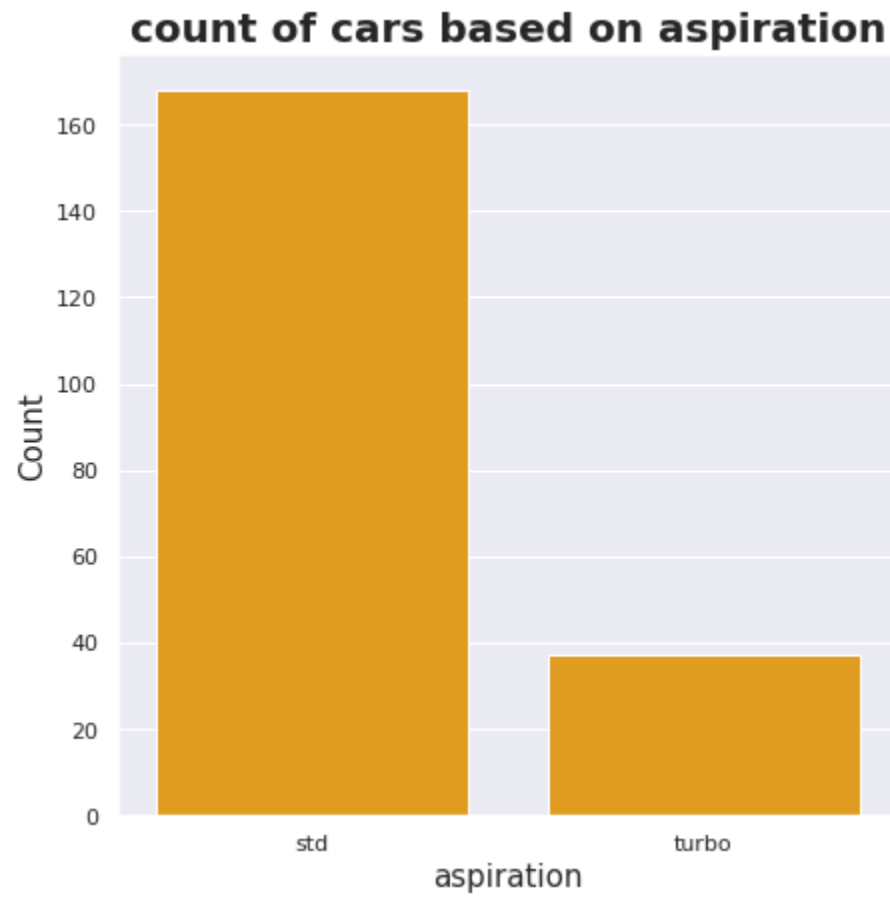
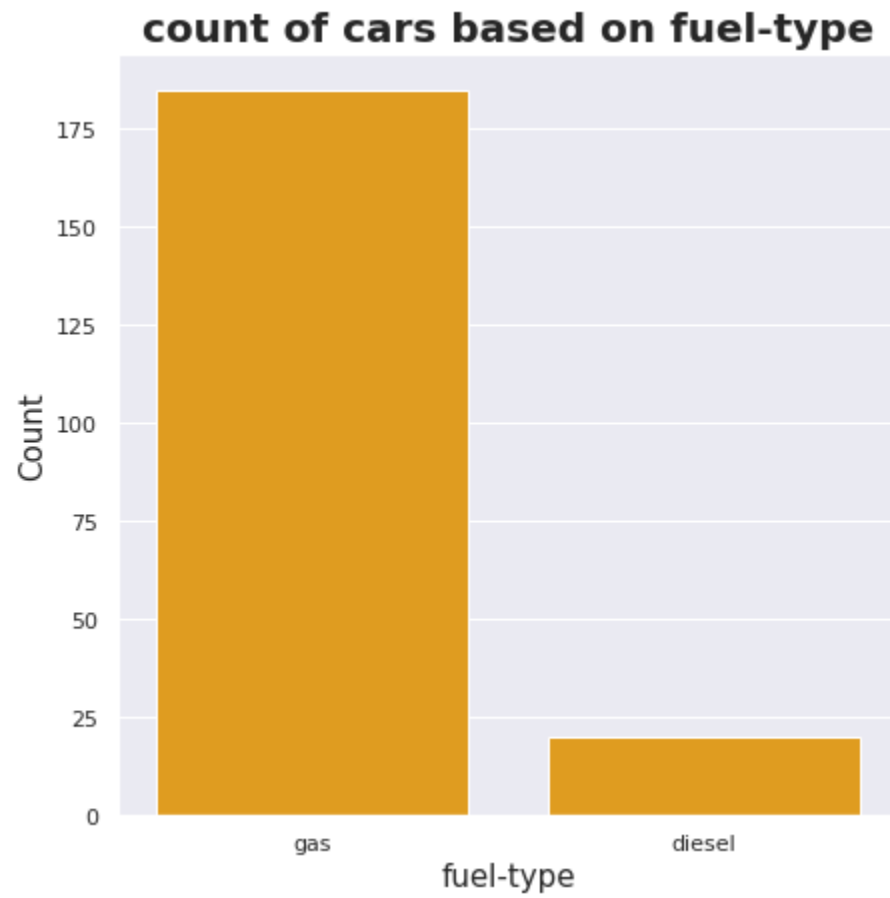




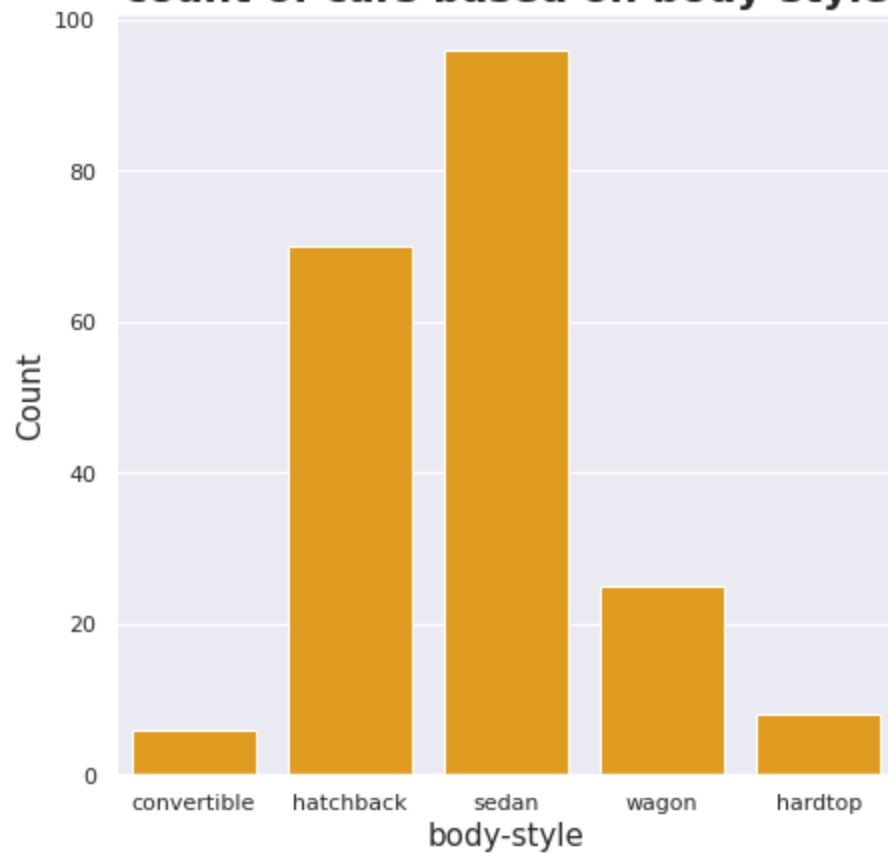
Step-18: Plotting count of cars based on different features

```
In [21]: col=['fuel-type', 'aspiration', 'body-style', 'engine-type', 'fuel-system']
for i in col:
    plt.subplots(figsize=(7,7))
    sns.countplot(x=i,data=df,color='orange')
    plt.xlabel(i,fontsize=15)
```

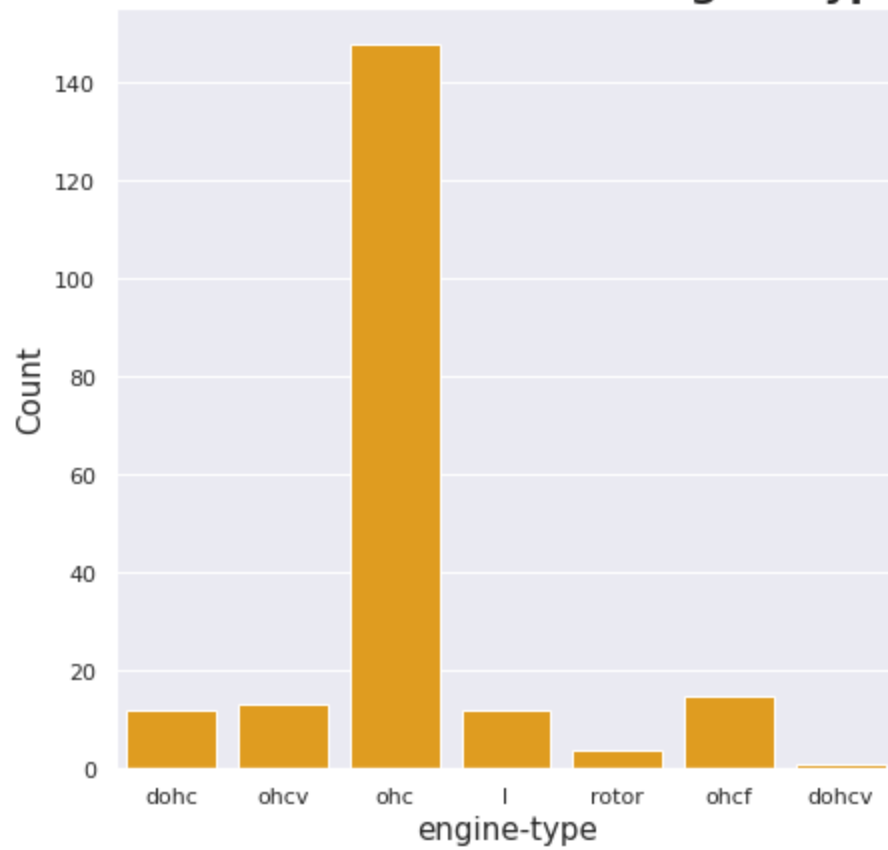
```
plt.ylabel('Count', fontsize=15)  
plt.title('count of cars based on ' + i, fontsize=20, fontweight='bold')
```



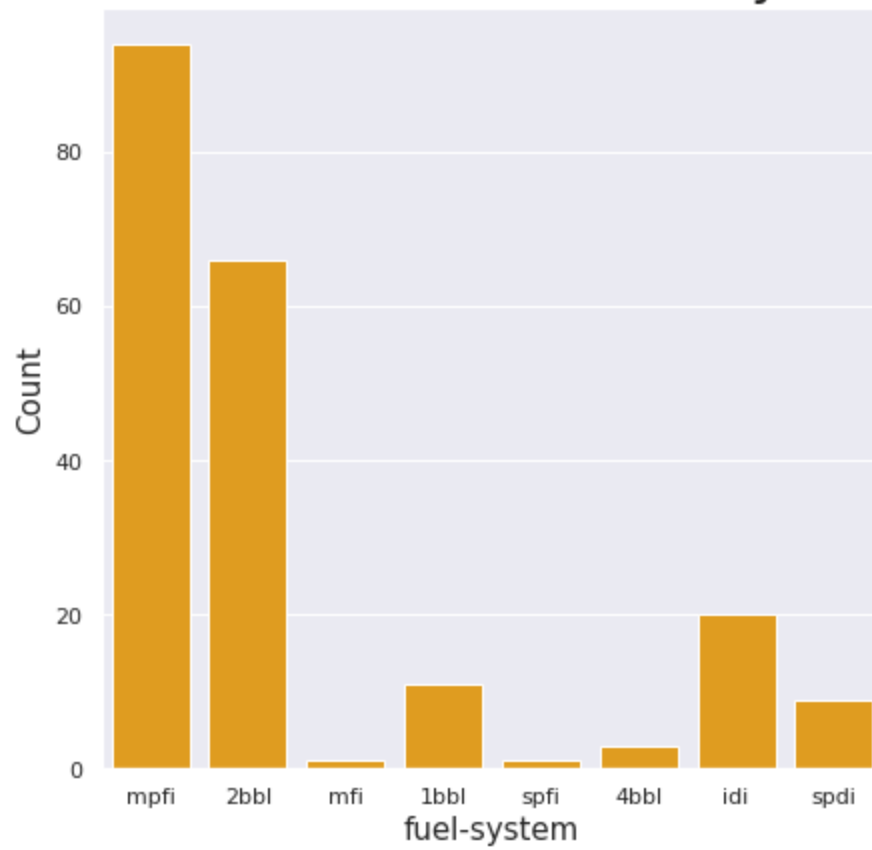
count of cars based on body-style



count of cars based on engine-type

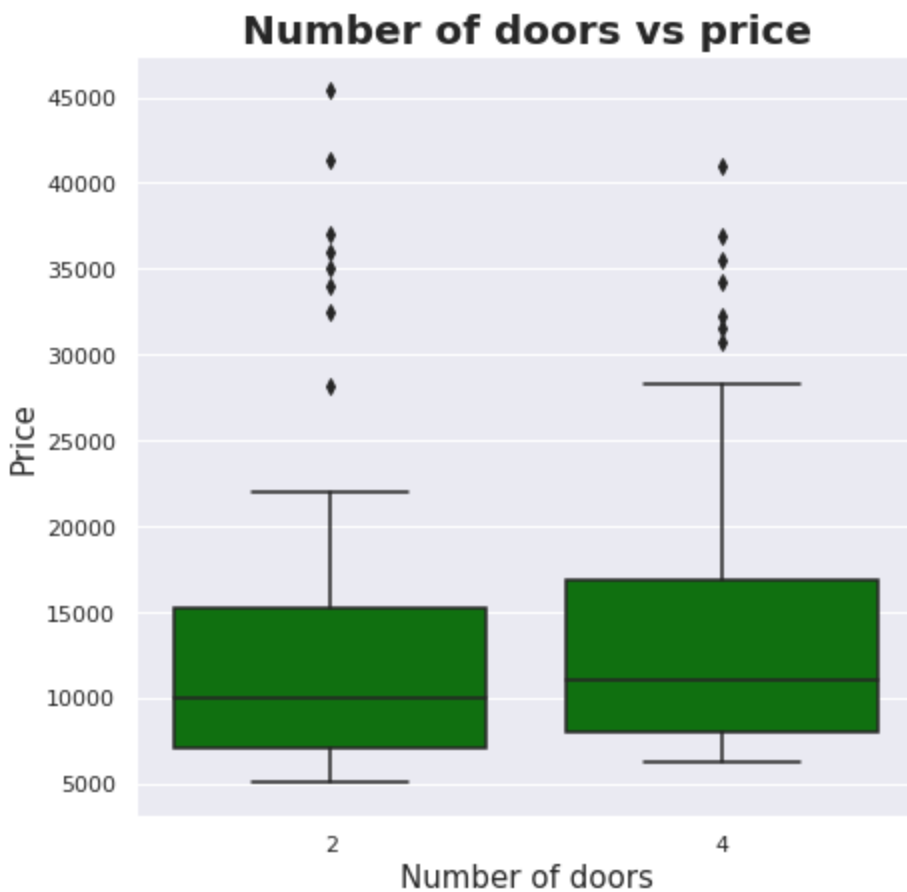


count of cars based on fuel-system



Step-18: Relation between number of doors and price in boxplot

```
In [22]: plt.subplots(figsize=(7,7))
sns.set_style("darkgrid")
sns.boxplot(x='num-of-doors',y='price',color='green',data=df)
plt.xlabel('Number of doors',fontsize=15)
plt.ylabel('Price',fontsize=15)
plt.title('Number of doors vs price',fontsize=20,fontweight='bold')
plt.show()
```



** From the boxplot we can understand that the average price of a vehicle with two doors is 10000, and the average price of a vehicle with four doors is 12000.

DATA CLEANING

Step-19: Checking the object data type columns in the dataframe

```
In [23]: df.select_dtypes(include='object')
```

Out[23]:

| | make | fuel-type | aspiration | body-style | drive-wheels | engine-location | engine-type | num-of-cylinders | fuel-system |
|-----|-------------|-----------|------------|-------------|--------------|-----------------|-------------|------------------|-------------|
| 0 | alfa-romero | gas | std | convertible | rwd | front | dohc | four | mpfi |
| 1 | alfa-romero | gas | std | convertible | rwd | front | dohc | four | mpfi |
| 2 | alfa-romero | gas | std | hatchback | rwd | front | ohcv | six | mpfi |
| 3 | audi | gas | std | sedan | fwd | front | ohc | four | mpfi |
| 4 | audi | gas | std | sedan | 4wd | front | ohc | five | mpfi |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 200 | volvo | gas | std | sedan | rwd | front | ohc | four | mpfi |
| 201 | volvo | gas | turbo | sedan | rwd | front | ohc | four | mpfi |
| 202 | volvo | gas | std | sedan | rwd | front | ohcv | six | mpfi |
| 203 | volvo | diesel | turbo | sedan | rwd | front | ohc | six | idi |
| 204 | volvo | gas | turbo | sedan | rwd | front | ohc | four | mpfi |

205 rows × 9 columns

Step-20: Convert categorical features to dummy variables

```
In [24]: dummy=pd.get_dummies(df[['make','fuel-type','body-style','aspiration','drive-wheels',  
                                'engine-location','engine-type','num-of-cylinders','fuel-system']])  
dummy
```

```
Out[24]:
```

| | make_audi | make_bmw | make_chevrolet | make_dodge | make_honda | make_isuzu | make_jaguar | make_maz |
|-----|-----------|----------|----------------|------------|------------|------------|-------------|----------|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 200 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 201 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 202 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 203 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 204 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

205 rows × 49 columns

Step-21: Dropping the categorical columns from the dataframe df

```
In [25]: df.drop(['make','fuel-type','aspiration','body-style','drive-wheels','engine-location',  
                'num-of-cylinders','drive-wheels','engine-type','fuel-system'],axis=1,inplace=True)
```

Step-22: Concat the dataframes

```
In [26]: dfe=pd.concat([df,dummy],axis=1)
```

MODEL BUILDING

Step-23: Assigning input and output

```
In [27]: x = dfe.drop("price",axis = 1).values  
y = dfe["price"].values
```

Step-24: Split data for train and test

```
In [28]: x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.30,random_state=0)
```

Step-24: MinMaxScaler: It transforms data by scaling features to a given range

```
In [29]: scaler=MinMaxScaler()  
scaler.fit(x_train)  
x_train=scaler.transform(x_train)  
x_test=scaler.transform(x_test)
```

Step-25: Training and Predicting data

```
In [30]: from sklearn.linear_model import LinearRegression # Using Linear Regression
reg=LinearRegression()
reg.fit(x_train,y_train)
y_pred=reg.predict(x_test)
y_pred
```

```
Out[30]: array([ 6317.49121234, 16279.25392884, 11415.57552673, -7498.03462069,
          9973.95321452, 11244.5168697 ,  5727.62674462,  4463.62557424,
          16165.08430352,  8323.51425568, 20427.37350994, 27181.45334551,
          12729.08315618, 14581.71849619,  6481.09359812, 10327.54271017,
          10864.15290736, 17042.14781679,  8620.13518081,  8012.56059349,
           9308.6426538 , 14990.29945036, 11434.6403626 , 11365.32092726,
          17616.51217513,  6971.63955767,  7396.11547906, 15509.88764122,
           7546.67581655,  5791.03344207,  9049.11448554, 11959.89341155,
          22694.86536296,  9279.52096138,  7277.64141404, 29293.50463344,
          14229.20438957, 14654.47618507,  4649.82351011, 37206.53011509,
           5235.2674832 , 12651.18929658, 34777.97286691, 21379.65645375,
          10828.45867882,  7670.19903561,  6605.30701906, 13136.61885706,
          10172.61881449, 10683.82963658, 19635.84795323,  6842.93281529,
           7255.97541734,  9756.98232837, 17449.71964832, 17396.42779201,
           8960.91296904, 17956.79718565, 10334.40348454,  6365.40562905,
           3125.09668801, 14392.30817763])
```

Step-26: Model Evaluation

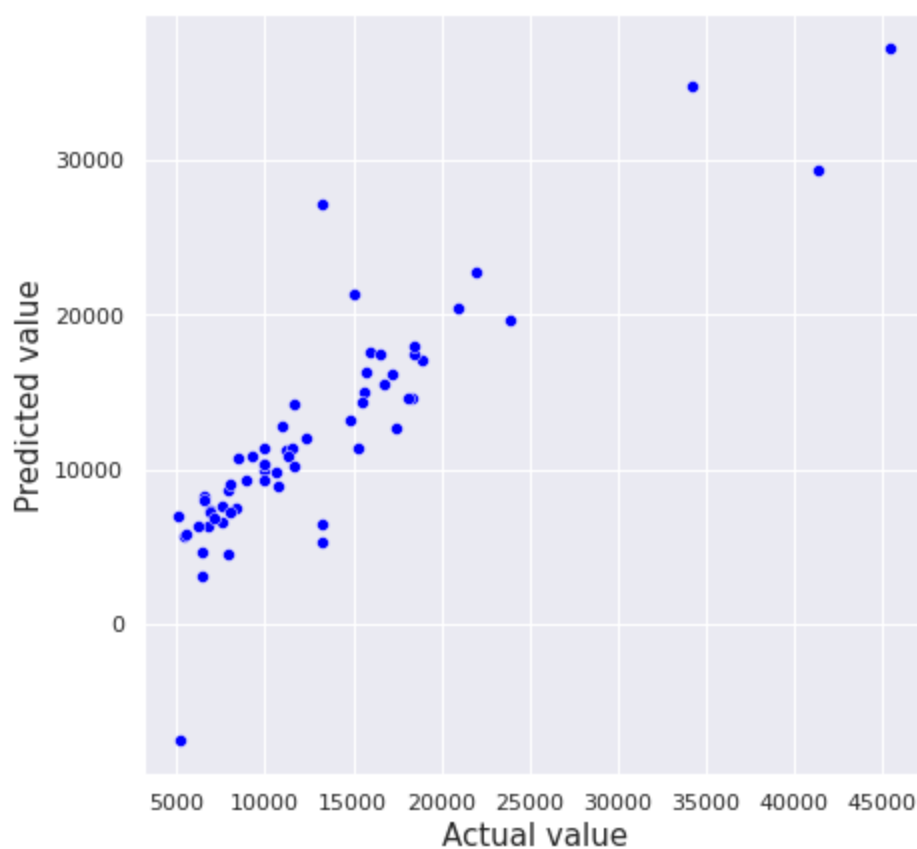
```
In [31]: dif=pd.DataFrame({'Actual Value':y_test,'Predicted Value':y_pred})
print(dif)
```

| | Actual Value | Predicted Value |
|-----|--------------|-----------------|
| 0 | 6795.0 | 6317.491212 |
| 1 | 15750.0 | 16279.253929 |
| 2 | 15250.0 | 11415.575527 |
| 3 | 5151.0 | -7498.034621 |
| 4 | 9995.0 | 9973.953215 |
| ... | ... | ... |
| 57 | 18420.0 | 17956.797186 |
| 58 | 9960.0 | 10334.403485 |
| 59 | 6229.0 | 6365.405629 |
| 60 | 6479.0 | 3125.096688 |
| 61 | 15510.0 | 14392.308178 |

[62 rows x 2 columns]

```
In [32]: plt.subplots(figsize=(7,7))
ax = sns.scatterplot(x=y_test, y=y_pred,color='blue')
plt.xlabel("Actual value",fontsize=15)
plt.ylabel("Predicted value",fontsize=15)
```

```
Out[32]: Text(0, 0.5, 'Predicted value')
```



```
In [33]: print("y intercept : ",reg.intercept_)
```

```
y intercept : 10291.759068734822
```

```
In [34]: print("coefficients:",reg.coef_)
```

```
coefficients: [ 7.03865875e+02 -3.46167020e+02  1.16376348e+03  1.00803440e+04
 -9.56327628e+03  6.91928571e+03 -1.60903865e+03  1.43060592e+04
  2.71641294e+04 -8.17542031e+03 -1.86470740e+03 -1.53873389e+04
 -5.52843825e+03  5.29499596e+03 -2.08350568e+03  2.10945186e+03
  4.70574191e+02  6.01887547e+03 -2.56721711e+03 -4.70973235e+03
 -9.60658414e+02 -3.35342442e+03 -6.24588301e+02 -1.71145149e+03
  4.83908998e+03  2.18278728e-11 -4.29784193e+03 -8.80163815e+02
 -2.96356235e+03 -4.23608093e+03  7.18325912e+03 -4.00177669e-11
  2.41511793e+03 -3.02960800e+03 -2.11830465e+03 -1.04025905e+03
 -5.25296047e+02 -7.13350032e+03 -3.80189308e+03 -3.90392484e+03
 -3.80409167e+03 -4.70633956e+03  1.80667568e+03  1.99556441e+03
  3.51854050e+03  7.18325912e+03  1.04591891e-11 -2.96356235e+03
  1.99761291e+02  4.15365112e+03 -2.38554668e+03  5.46059843e+03
 -1.35915801e+03  3.76388434e+03  1.49604437e+03  0.00000000e+00
  3.90876748e+03  5.46059843e+03  2.81580333e+03  2.08116509e+02
  7.13350032e+03  2.2224857e+03  2.11066630e+03  1.11920550e+03
  3.24581422e+03]
```

```
In [35]: print('MAE:', metrics.mean_absolute_error(y_test,y_pred)) # Mean Absolute Error (MAE)
print('MSE:', metrics.mean_squared_error(y_test,y_pred)) # Mean Squared Error (MSE)
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test,y_pred))) # Root Mean Squared
print('R^2 Score:', metrics.r2_score(y_test,y_pred)) # R-squared (R2)
```

```
MAE: 2243.300306103572
MSE: 14223461.941464137
RMSE: 3771.400527849586
R^2 Score: 0.7632813583401515
```

Training and Predicting using Decision Tree Regression


```
In [36]: from sklearn.tree import DecisionTreeRegressor # Using Decision Tree Regression
dt=DecisionTreeRegressor(max_depth=4)
dt.fit(x_train,y_train)
yd_pred = dt.predict(x_test)
yd_pred
```

```
Out[36]: array([ 6402.          , 14793.43592847,  9480.74074074,  6402.          ,
        9480.74074074, 14793.43592847,  6402.          ,  6402.          ,
        14793.43592847,  6402.          , 22835.          , 35978.66666667,
        13309.625       , 14793.43592847,  6402.          , 14793.43592847,
        14793.43592847, 17277.5       ,  7945.71428571,  6402.          ,
        13309.625       , 14793.43592847,  9480.74074074, 14793.43592847,
        14793.43592847,  7945.71428571,  7945.71428571, 14793.43592847,
        7945.71428571,  7945.71428571,  7945.71428571, 14793.43592847,
        14793.43592847,  9480.74074074,  7945.71428571, 33690.66666667,
        7945.71428571, 14793.43592847,  6402.          , 40960.          ,
        6402.          , 14793.43592847, 35978.66666667, 14793.43592847,
        9480.74074074,  7945.71428571,  6402.          , 14793.43592847,
        14793.43592847,  7945.71428571, 21406.25       ,  7945.71428571,
        7945.71428571,  9480.74074074, 14793.43592847, 14793.43592847,
        9480.74074074, 14793.43592847,  9480.74074074,  6402.          ,
        6402.          , 14793.43592847])
```

```
In [37]: print('MAE:', metrics.mean_absolute_error(y_test,yd_pred)) # Mean Absolute Error (MAE)
print('MSE:', metrics.mean_squared_error(y_test,yd_pred)) # Mean Squared Error (MSE)
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test,yd_pred))) # Root Mean Squared
print('R^2 Score:', metrics.r2_score(y_test,yd_pred)) # R-squared (R2)
```

```
MAE: 2416.618657914397
MSE: 16400319.731546937
RMSE: 4049.730822109901
R^2 Score: 0.7270522868752888
```

Training and Predicting using Random Forest Regression

```
In [38]: from sklearn.ensemble import RandomForestRegressor
model=RandomForestRegressor(n_estimators=100)
model.fit(x_train,y_train)
yr_pred=model.predict(x_test)
yr_pred
```

```
Out[38]: array([ 6048.59       , 16739.63129353, 12801.62109453,  5786.67       ,
        9695.63666667, 13735.885       ,  5889.53       ,  7383.00666667,
        17846.9109204 ,  6779.58       , 19523.75129353, 35788.          ,
        12075.76       , 14246.89166667,  6332.26       , 13326.405       ,
        12758.65629353, 17590.79258706,  8827.87       ,  6498.82       ,
        9719.62       , 15605.88       , 11449.72666667, 14005.915       ,
        17378.55129353,  7217.79       ,  7623.44       , 14373.85       ,
        8066.38       ,  6891.56       ,  8390.67       , 12810.36925373,
        15584.4529602 , 10059.67       ,  7208.41       ,  33314.5       ,
        9215.93       , 16077.45129353,  5990.79       ,  37320.38       ,
        6192.78       , 14246.12052239, 35248.63       , 12735.00083333,
        10368.615       ,  7756.86       ,  6657.93       , 14034.46258706,
        12909.20962687,  8609.54       , 18907.00517413,  7236.09       ,
        8475.2       ,  9194.00166667, 17360.39905473, 17672.75258706,
        9982.565       , 17252.9038806 ,  9404.88       ,  6353.8       ,
        5952.13       , 13084.79       ])
```

```
In [39]: print('MAE:', metrics.mean_absolute_error(y_test,yr_pred)) # Mean Absolute Error (MAE)
print('MSE:', metrics.mean_squared_error(y_test,yr_pred)) # Mean Squared Error (MSE)
```

```
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test,yr_pred))) # Root Mean Squared  
print('R^2 Score:', metrics.r2_score(y_test,yr_pred)) # R-squared (R2)
```

MAE: 2078.0760850987
MSE: 15004353.087856911
RMSE: 3873.545286666584
R^2 Score: 0.7502851206999029