BIA 5302_Group Project

August 6, 2023

1 Machine Learning and Programming 1 - BIA-5302 - Group Project

1.1 *Group Members :*

1.1.1 Damandeep Singh - N01533482

1.1.2 Ishan Sahni - N01530356

1.1.3 Chandeep Singh - N01530793

1.1.4 Manoj Maddukuri - N01530629

2 MILESTONE 2

2.1 Importing libraries, loading and viewing the data

[1]:
```
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings('ignore')
import matplotlib.pyplot as plt
```

[2]:
```
cars = pd.read_csv("E:/Documents/Humber/Semester 3/Machine Learning and␣
↪Programming 1/Group Project/BIA 5302_Group Project_Data.csv")
```

[3]:
```
cars.head()
```

[3]:
```
         ID  Price  Levy Manufacturer    Model  Prod. year   Category  \
0  45654403  13328  1399        LEXUS   RX 450        2010       Jeep
1  44731507  16621  1018    CHEVROLET  Equinox        2011       Jeep
2  45774419   8467     -        HONDA      FIT        2006  Hatchback
3  45769185   3607   862         FORD   Escape        2011       Jeep
4  45809263  11726   446        HONDA      FIT        2014  Hatchback

  Leather interior Fuel type Engine volume    Mileage  Cylinders  \
0              Yes    Hybrid           3.5  186005 km          6
1               No    Petrol             3  192000 km          6
2               No    Petrol           1.3  200000 km          4
3              Yes    Hybrid           2.5  168966 km          4
```

```
4              Yes    Petrol          1.3   91901 km          4

  Gear box type Drive wheels   Doors             Wheel   Color  Airbags
0     Automatic          4x4  04-May        Left wheel  Silver       12
1     Tiptronic          4x4  04-May        Left wheel   Black        8
2      Variator        Front  04-May  Right-hand drive   Black        2
3     Automatic          4x4  04-May        Left wheel   White        0
4     Automatic        Front  04-May        Left wheel  Silver        4
```

[4]:
```python
#Checking the datatypes of the variables
cars.dtypes
```

[4]:
```
ID                    int64
Price                 int64
Levy                 object
Manufacturer         object
Model                object
Prod. year            int64
Category             object
Leather interior     object
Fuel type            object
Engine volume        object
Mileage              object
Cylinders             int64
Gear box type        object
Drive wheels         object
Doors                object
Wheel                object
Color                object
Airbags               int64
dtype: object
```

[5]:
```python
#Printing the summary statistics
cars.describe()
```

[5]:

	ID	Price	Prod. year	Cylinders	Airbags
count	1.923700e+04	1.923700e+04	19237.000000	19237.000000	19237.000000
mean	4.557654e+07	1.855593e+04	2010.912824	4.582991	6.582627
std	9.365914e+05	1.905813e+05	5.668673	1.199933	4.320168
min	2.074688e+07	1.000000e+00	1939.000000	1.000000	0.000000
25%	4.569837e+07	5.331000e+03	2009.000000	4.000000	4.000000
50%	4.577231e+07	1.317200e+04	2012.000000	4.000000	6.000000
75%	4.580204e+07	2.207500e+04	2015.000000	4.000000	12.000000
max	4.581665e+07	2.630750e+07	2020.000000	16.000000	16.000000

3 Data Preparation and Cleaning

3.0.1 1. Missing Values

[6]:
```
cars.isnull().sum()
```

[6]:
```
ID                  0
Price               0
Levy                0
Manufacturer        0
Model               0
Prod. year          0
Category            0
Leather interior    0
Fuel type           0
Engine volume       0
Mileage             0
Cylinders           0
Gear box type       0
Drive wheels        0
Doors               0
Wheel               0
Color               0
Airbags             0
dtype: int64
```

There are no null/missing values in the dataset.

3.0.2 2. Duplicate Data

[7]:
```
duplicates = cars.duplicated()
duplicates_num = duplicates.sum()
print("Number of duplicate rows in the dataset are", duplicates_num)
```

```
Number of duplicate rows in the dataset are 313
```

[8]:
```
cars.drop_duplicates(inplace=True)
cars
```

[8]:
```
             ID  Price  Levy   Manufacturer    Model  Prod. year   Category  \
0      45654403  13328  1399          LEXUS   RX 450        2010       Jeep
1      44731507  16621  1018      CHEVROLET  Equinox        2011       Jeep
2      45774419   8467     -          HONDA      FIT        2006  Hatchback
3      45769185   3607   862           FORD   Escape        2011       Jeep
4      45809263  11726   446          HONDA      FIT        2014  Hatchback
…           …      …     …            …        …          …
19232  45798355   8467     -  MERCEDES-BENZ  CLK 200        1999      Coupe
19233  45778856  15681   831        HYUNDAI   Sonata        2011      Sedan
```

```
19234  45804997  26108   836        HYUNDAI   Tucson        2010        Jeep
19235  45793526   5331  1288      CHEVROLET  Captiva        2007        Jeep
19236  45813273    470   753        HYUNDAI   Sonata        2012       Sedan

      Leather interior Fuel type Engine volume    Mileage  Cylinders  \
0                  Yes    Hybrid           3.5  186005 km          6
1                   No    Petrol             3  192000 km          6
2                   No    Petrol           1.3  200000 km          4
3                  Yes    Hybrid           2.5  168966 km          4
4                  Yes    Petrol           1.3   91901 km          4
…                    …       …             …        …        …
19232              Yes       CNG     2.0 Turbo  300000 km          4
19233              Yes    Petrol           2.4  161600 km          4
19234              Yes    Diesel             2  116365 km          4
19235              Yes    Diesel             2   51258 km          4
19236              Yes    Hybrid           2.4  186923 km          4

      Gear box type Drive wheels   Doors             Wheel   Color  Airbags
0         Automatic          4x4  04-May        Left wheel  Silver       12
1         Tiptronic          4x4  04-May        Left wheel   Black        8
2          Variator        Front  04-May  Right-hand drive   Black        2
3         Automatic          4x4  04-May        Left wheel   White        0
4         Automatic        Front  04-May        Left wheel  Silver        4
…               …          …    …              …      …      …
19232        Manual         Rear  02-Mar        Left wheel  Silver        5
19233     Tiptronic        Front  04-May        Left wheel     Red        8
19234     Automatic        Front  04-May        Left wheel    Grey        4
19235     Automatic        Front  04-May        Left wheel   Black        4
19236     Automatic        Front  04-May        Left wheel   White       12

[18924 rows x 18 columns]
```

We have dropped 313 rows from the dataset as these rows would not add anything to our learning process of the model.

3.0.3 3. Irrelevant and Incorrect Data

[9]:
```
#Displaying the number of "-" values in the 'Levy', i.e., Tax variable
cars['Levy'].describe()
```

[9]:
```
count      18924
unique       559
top            -
freq        5709
Name: Levy, dtype: object
```

[10]:
```
#Replacing the "-" values to be blank or NULL
cars['Levy'].replace({'-':np.nan}, inplace=True)
```

[11]:
```
cars.isnull().sum()
```

[11]:
```
ID                     0
Price                  0
Levy                5709
Manufacturer           0
Model                  0
Prod. year             0
Category               0
Leather interior       0
Fuel type              0
Engine volume          0
Mileage                0
Cylinders              0
Gear box type          0
Drive wheels           0
Doors                  0
Wheel                  0
Color                  0
Airbags                0
dtype: int64
```

[12]:
```
#Converting the datatype of 'Levy' from object to float
cars['Levy'] = cars['Levy'].astype(float)
```

[13]:
```
# Replacing the NULL values in 'Levy' with the mean
mean_levy = cars['Levy'].mean()
mean_levy
```

[13]:
```
906.2992054483541
```

[14]:
```
cars['Levy'].fillna(mean_levy, inplace=True)
```

The incorrect data in 'Levy' variable has been replaced with its mean.

[15]:
```
#Changing the 'Doors' format into appropriate numbers
cars['Doors'].value_counts()
cars['Doors'].replace({'04-May':4, '02-Mar':2, '>5':5}, inplace=True)
```

The 'Doors' variable has now been formatted in the correct way.

[16]:
```
#Removing the "km" unit from the 'Mileage' variable
cars['Mileage']=cars['Mileage'].str.replace('km','')
```

[17]:
```
cars['Mileage'].value_counts()
```

[17]:
```
0          714
200000     181
150000     159
160000     120
180000     117
          …
100563       1
354300       1
21178        1
110539       1
186923       1
Name: Mileage, Length: 7687, dtype: int64
```

[18]:
```
#Converting the datatype of 'Mileage'
cars['Mileage']=cars['Mileage'].astype(int)
```

[19]:
```
cars['Mileage'].head()
```

[19]:
```
0    186005
1    192000
2    200000
3    168966
4     91901
Name: Mileage, dtype: int32
```

[20]:
```
cars['Mileage'].mean()
```

[20]:
```
1555372.718928345
```

The 'Mileage' variable has now been formatted as per the requirements of our model.

[21]:
```
cars['Engine volume'].value_counts()
```

[21]:
```
2             3856
2.5           2246
1.8           1743
1.6           1446
1.5           1289
              …
6.8              1
6.7              1
3.1              1
0.8 Turbo        1
1.1 Turbo        1
Name: Engine volume, Length: 107, dtype: int64
```

[22]:
```
#Removing "Turbo" from the values
cars['Engine volume']=cars['Engine volume'].str.split(' ').str.get(0)
```

[23]:
```python
#Converting the datatype of 'Engine Volume'
cars['Engine volume']=cars['Engine volume'].astype(float)
```

[24]:
```python
cars['Engine volume'].mean()
```

[24]: 2.306251321073769

The 'Engine volume' variable has now been formatted as per the requirements of our model.

[25]:
```python
cars['Wheel'].value_counts()
```

[25]:
```
Left wheel          17471
Right-hand drive     1453
Name: Wheel, dtype: int64
```

[26]:
```python
cars['Wheel'].replace({'Right-hand drive':'Right wheel'}, inplace=True)
```

The 'Wheel' variable has now been formatted correctly.

[27]:
```python
#Dropping the irrelevant record in Manufacturer = áƒ¡áƒ®áƒ•áƒ  using boolean␣
↪indexing
cars = cars[cars['ID']!=45779593]
cars = cars[cars['ID']!=39223518]
```

[28]:
```python
cars
```

[28]:
```
             ID  Price         Levy   Manufacturer    Model  Prod. year  \
0      45654403  13328  1399.000000          LEXUS   RX 450        2010
1      44731507  16621  1018.000000      CHEVROLET  Equinox        2011
2      45774419   8467   906.299205          HONDA      FIT        2006
3      45769185   3607   862.000000           FORD   Escape        2011
4      45809263  11726   446.000000          HONDA      FIT        2014
…           …    …            …             …       …         …
19232  45798355   8467   906.299205  MERCEDES-BENZ  CLK 200        1999
19233  45778856  15681   831.000000        HYUNDAI   Sonata        2011
19234  45804997  26108   836.000000        HYUNDAI   Tucson        2010
19235  45793526   5331  1288.000000      CHEVROLET  Captiva        2007
19236  45813273    470   753.000000        HYUNDAI   Sonata        2012

        Category Leather interior Fuel type  Engine volume  Mileage  \
0           Jeep              Yes    Hybrid            3.5   186005
1           Jeep               No    Petrol            3.0   192000
2      Hatchback               No    Petrol            1.3   200000
3           Jeep              Yes    Hybrid            2.5   168966
4      Hatchback              Yes    Petrol            1.3    91901
…            …                …       …              …        …
19232      Coupe              Yes       CNG            2.0   300000
19233      Sedan              Yes    Petrol            2.4   161600
```

```
19234       Jeep            Yes    Diesel           2.0   116365
19235       Jeep            Yes    Diesel           2.0    51258
19236      Sedan            Yes    Hybrid           2.4   186923

       Cylinders Gear box type Drive wheels  Doors        Wheel   Color  \
0              6     Automatic          4x4      4   Left wheel  Silver
1              6     Tiptronic          4x4      4   Left wheel   Black
2              4      Variator        Front      4  Right wheel   Black
3              4     Automatic          4x4      4   Left wheel   White
4              4     Automatic        Front      4   Left wheel  Silver
…          …          …          …     …       …      …
19232          4        Manual         Rear      2   Left wheel  Silver
19233          4     Tiptronic        Front      4   Left wheel     Red
19234          4     Automatic        Front      4   Left wheel    Grey
19235          4     Automatic        Front      4   Left wheel   Black
19236          4     Automatic        Front      4   Left wheel   White

       Airbags
0           12
1            8
2            2
3            0
4            4
…        …
19232        5
19233        8
19234        4
19235        4
19236       12

[18922 rows x 18 columns]
```

[29]:
```
#Saving the cleaned dataset (so far) in a new dataframe variable
cars_2 = cars.copy()
```

3.0.4 4. Categorical Data

[30]:
```
#Categorising the data
cars['Wheel'].replace({'Left wheel':1,'Right wheel':0}, inplace=True)
cars['Leather interior'].replace({'Yes':1,'No':0}, inplace=True)
```

[31]:
```
cars.head()
```

[31]:
```
         ID  Price         Levy Manufacturer    Model  Prod. year   Category  \
0  45654403  13328  1399.000000        LEXUS   RX 450        2010       Jeep
1  44731507  16621  1018.000000    CHEVROLET  Equinox        2011       Jeep
2  45774419   8467   906.299205        HONDA      FIT        2006  Hatchback
```

```
3  45769185   3607   862.000000         FORD   Escape         2011        Jeep
4  45809263  11726   446.000000        HONDA      FIT         2014   Hatchback

   Leather interior Fuel type  Engine volume  Mileage  Cylinders  \
0                 1    Hybrid            3.5   186005          6
1                 0    Petrol            3.0   192000          6
2                 0    Petrol            1.3   200000          4
3                 1    Hybrid            2.5   168966          4
4                 1    Petrol            1.3    91901          4

  Gear box type Drive wheels  Doors  Wheel   Color  Airbags
0     Automatic          4x4      4      1  Silver       12
1     Tiptronic          4x4      4      1   Black        8
2      Variator        Front      4      0   Black        2
3     Automatic          4x4      4      1   White        0
4     Automatic        Front      4      1  Silver        4
```

[32]:
```python
#Categorising more data
cars['Fuel type'].replace({'Plug-in Hybrid':6,'Petrol':5,'LPG':4,'Hydrogen':
 ↪3,'Hybrid':2,'Diesel':1,'CNG':0}, inplace=True)
cars['Gear box type'].replace({'Variator':3,'Tiptronic':2,'Manual':
 ↪1,'Automatic':0}, inplace=True)
cars['Drive wheels'].replace({'Rear':2,'Front':1,'4x4':0}, inplace=True)
cars['Category'].replace({'Universal':10,'Sedan':9,'Pickup':8,'Minivan':
 ↪7,'Microbus':6,'Limousine':5,'Jeep':4,'Hatchback':3,'Goods wagon':2,'Coupe':
 ↪1,'Cabriolet':0}, inplace=True)
```

[33]:
```python
#Changing the datatypes for the converted categorical variables
cars['Leather interior'] = cars['Leather interior'].astype(int)
cars['Wheel'] = cars['Wheel'].astype(int)
cars['Fuel type'] = cars['Fuel type'].astype(int)
cars['Gear box type'] = cars['Gear box type'].astype(int)
cars['Drive wheels'] = cars['Drive wheels'].astype(int)
cars['Category'] = cars['Category'].astype(int)
```

[34]:
```python
cars.dtypes
```

[34]:
```
ID                    int64
Price                 int64
Levy                float64
Manufacturer         object
Model                object
Prod. year            int64
Category              int32
Leather interior      int32
Fuel type             int32
Engine volume       float64
```

```
Mileage              int32
Cylinders            int64
Gear box type        int32
Drive wheels         int32
Doors                int64
Wheel                int32
Color               object
Airbags              int64
dtype: object
```

3.0.5 5. Outliers

[35]:
```
import plotly.express as px
```

[36]:
```
#Checking for the outliers in 'Price' using the quartile method
q1, q3 = np.percentile(cars["Price"], [25, 75])
iqr = q3 - q1
lower_bound = q1 - 1.5*iqr
upper_bound = q3 + 1.5*iqr

#Creating conditions to isolate the outliers
outliers_price_1 = cars["Price"][(cars["Price"] < lower_bound) | (cars["Price"]␣
 ↪> upper_bound)]
```

[37]:
```
outliers_price_1
```

[37]:
```
14       59464
36       51746
47       55390
56       87112
73       53154
         …
19144    56814
19161    64290
19180    63886
19188    61154
19211    50037
Name: Price, Length: 1055, dtype: int64
```

[38]:
```
mean_Price_without_outliers_1 = cars["Price"][(cars["Price"] >= lower_bound) &␣
 ↪(cars["Price"] <= upper_bound)].mean()

#Replacing the outliers with the mean_Price_without_outliers
cars["Price"] = cars["Price"].where(~((cars["Price"] < lower_bound) |␣
 ↪(cars["Price"] > upper_bound)), mean_Price_without_outliers_1)
```

[39]:
```python
#Checking for the outliers in 'Mileage' using the quartile method
q1, q3 = np.percentile(cars["Mileage"], [25, 75])
iqr = q3 - q1
lower_bound = q1 - 1.5*iqr
upper_bound = q3 + 1.5*iqr

#Creating conditions to isolate the outliers
outliers_Mileage = cars["Mileage"][(cars["Mileage"] < lower_bound) |
    (cars["Mileage"] > upper_bound)]
```

[40]:
```python
mean_without_outliers = cars["Mileage"][(cars["Mileage"] >= lower_bound) &
    (cars["Mileage"] <= upper_bound)].mean()
mean_without_outliers
```

[40]:
```
129507.99163340077
```

[41]:
```python
#Replacing the outliers with the mean_without_outliers
cars["Mileage"] = cars["Mileage"].where(~((cars["Mileage"] < lower_bound) |
    (cars["Mileage"] > upper_bound)), mean_without_outliers)
```

[42]:
```python
cars["Mileage"].mean()
```

[42]:
```
129507.99163340076
```

[43]:
```python
cars.head(15)
```

[43]:
```
          ID         Price         Levy   Manufacturer     Model  Prod. year  \
0   45654403  13328.000000  1399.000000          LEXUS    RX 450        2010
1   44731507  16621.000000  1018.000000      CHEVROLET   Equinox        2011
2   45774419   8467.000000   906.299205          HONDA       FIT        2006
3   45769185   3607.000000   862.000000           FORD    Escape        2011
4   45809263  11726.000000   446.000000          HONDA       FIT        2014
5   45802912  39493.000000   891.000000        HYUNDAI  Santa FE        2016
6   45656768   1803.000000   761.000000         TOYOTA     Prius        2010
7   45816158    549.000000   751.000000        HYUNDAI    Sonata        2013
8   45641395   1098.000000   394.000000         TOYOTA     Camry        2014
9   45756839  26657.000000   906.299205          LEXUS    RX 350        2007
10  45621750    941.000000  1053.000000  MERCEDES-BENZ     E 350        2014
11  45814819   8781.000000   906.299205           FORD   Transit        1999
12  45815568   3000.000000   906.299205           OPEL    Vectra        1997
13  45661288   1019.000000  1055.000000          LEXUS    RX 450        2013
14  45732604  14039.732636   891.000000        HYUNDAI  Santa FE        2016

    Category  Leather interior  Fuel type  Engine volume         Mileage  \
0          4                 1          2            3.5   186005.000000
1          4                 0          5            3.0   192000.000000
2          3                 0          5            1.3   200000.000000
```

```
3         4                 1         2            2.5  168966.000000
4         3                 1         5            1.3   91901.000000
5         4                 1         1            2.0  160931.000000
6         3                 1         2            1.8  258909.000000
7         9                 1         5            2.4  216118.000000
8         9                 1         2            2.5  129507.991633
9         4                 1         5            3.5  128500.000000
10        9                 1         1            3.5  184467.000000
11        6                 0         0            4.0       0.000000
12        2                 0         0            1.6  350000.000000
13        4                 1         2            3.5  138038.000000
14        4                 1         1            2.0   76000.000000

    Cylinders  Gear box type  Drive wheels  Doors  Wheel   Color  Airbags
0           6              0             0      4      1  Silver       12
1           6              2             0      4      1   Black        8
2           4              3             1      4      0   Black        2
3           4              0             0      4      1   White        0
4           4              0             1      4      1  Silver        4
5           4              0             1      4      1   White        4
6           4              0             1      4      1   White       12
7           4              0             1      4      1    Grey       12
8           4              0             1      4      1   Black       12
9           6              0             0      4      1  Silver       12
10          6              0             2      4      1   White       12
11          8              1             2      2      1    Blue        0
12          4              1             1      4      1   White        4
13          6              0             1      4      1   White       12
14          4              0             1      4      1   White        4
```

The outliers have now been replaced with the mean values without the outliers.

[44]:
```
cars.describe()
```

[44]:
```
                 ID         Price          Levy    Prod. year      Category  \
count  1.892200e+04  18922.000000  18922.000000  18922.000000  18922.000000
mean   4.557571e+07  14039.732636    906.299205   2010.914755      6.266938
std    9.364573e+05  11062.279973    387.172475      5.665814      2.792043
min    2.074688e+07      1.000000     87.000000   1939.000000      0.000000
25%    4.569503e+07   5331.000000    730.000000   2009.000000      4.000000
50%    4.577191e+07  13172.000000    906.299205   2012.000000      7.000000
75%    4.580174e+07  19444.000000    917.000000   2015.000000      9.000000
max    4.581665e+07  47120.000000  11714.000000   2020.000000     10.000000

       Leather interior     Fuel type  Engine volume        Mileage  \
count      18922.000000  18922.000000   18922.000000   18922.000000
mean           0.725610      3.427016       2.306252  129507.991633
std            0.446218      1.806268       0.877637   80134.138864
```

```
min             0.000000       0.000000        0.000000        0.000000
25%             0.000000       2.000000        1.800000    70195.250000
50%             1.000000       5.000000        2.000000   126400.000000
75%             1.000000       5.000000        2.500000   179200.000000
max             1.000000       6.000000       20.000000   367053.000000

          Cylinders  Gear box type  Drive wheels         Doors         Wheel  \
count  18922.000000   18922.000000  18922.000000  18922.000000  18922.000000
mean       4.580277       0.537522      0.909576      3.925378      0.923211
std        1.200271       0.897231      0.566505      0.404158      0.266263
min        1.000000       0.000000      0.000000      2.000000      0.000000
25%        4.000000       0.000000      1.000000      4.000000      1.000000
50%        4.000000       0.000000      1.000000      4.000000      1.000000
75%        4.000000       1.000000      1.000000      4.000000      1.000000
max       16.000000       3.000000      2.000000      5.000000      1.000000

            Airbags
count  18922.000000
mean       6.568914
std        4.322234
min        0.000000
25%        4.000000
50%        6.000000
75%       12.000000
max       16.000000
```

We now have a new mean in the 'Price' and 'Mileage' variables.

3.0.6 6. Feature Scaling

[45]:
```
#Feature Scaling on numerical variables won't be applicable since we can't have␣
↪a fraction for the
#number of car doors/number of airbags/price a car has. Moreover, since our␣
↪main business problem relates to predicting
#the car price(s) using linear regression, Feature Scaling is not necessary.
```

[46]:
```
#Since the 'Mileage' variable is in km units, we can change it into Miles
cars["Mileage"] = cars["Mileage"] / 1.609
```

[47]:
```
cars
```

[47]:
```
         ID    Price         Levy  Manufacturer    Model  Prod. year  \
0  45654403  13328.0  1399.000000         LEXUS   RX 450        2010
1  44731507  16621.0  1018.000000     CHEVROLET  Equinox        2011
2  45774419   8467.0   906.299205         HONDA      FIT        2006
3  45769185   3607.0   862.000000          FORD   Escape        2011
4  45809263  11726.0   446.000000         HONDA      FIT        2014
```

```
…          …         …          …              …        …           …
19232  45798355    8467.0   906.299205  MERCEDES-BENZ  CLK 200        1999
19233  45778856   15681.0   831.000000        HYUNDAI   Sonata        2011
19234  45804997   26108.0   836.000000        HYUNDAI   Tucson        2010
19235  45793526    5331.0  1288.000000      CHEVROLET  Captiva        2007
19236  45813273     470.0   753.000000        HYUNDAI   Sonata        2012

       Category  Leather interior  Fuel type  Engine volume        Mileage  \
0             4                 1          2            3.5  115602.858919
1             4                 0          5            3.0  119328.775637
2             3                 0          5            1.3  124300.807955
3             4                 1          2            2.5  105013.051585
4             3                 1          5            1.3   57116.842759
…          …                 …         …             …            …
19232         1                 1          0            2.0  186451.211933
19233         9                 1          5            2.4  100435.052828
19234         4                 1          1            2.0   72321.317589
19235         4                 1          1            2.0   31857.054071
19236         9                 1          2            2.4  116173.399627

       Cylinders  Gear box type  Drive wheels  Doors  Wheel   Color  Airbags
0              6              0             0      4      1  Silver       12
1              6              2             0      4      1   Black        8
2              4              3             1      4      0   Black        2
3              4              0             0      4      1   White        0
4              4              0             1      4      1  Silver        4
…            …            …           …    …    …     …      …
19232          4              1             2      2      1  Silver        5
19233          4              2             1      4      1     Red        8
19234          4              0             1      4      1    Grey        4
19235          4              0             1      4      1   Black        4
19236          4              0             1      4      1   White       12

[18922 rows x 18 columns]
```

3.0.7 7. Feature Engineering & Selection aka EDA

[48]:
```
import seaborn as sns
```

[49]:
```
cars.columns
```

[49]:
```
Index(['ID', 'Price', 'Levy', 'Manufacturer', 'Model', 'Prod. year',
       'Category', 'Leather interior', 'Fuel type', 'Engine volume', 'Mileage',
       'Cylinders', 'Gear box type', 'Drive wheels', 'Doors', 'Wheel', 'Color',
       'Airbags'],
      dtype='object')
```

[50]:
```python
year_intervals = [1930,1940,1950,1960,1970,1980,1990,2000,2010,2020]

#Creating a new column 'Year Interval' using pd.cut()
cars['Year Interval'] = pd.cut(cars['Prod. year'], bins=year_intervals,␣
↪labels=[1930,1940,1950,1960,1970,1980,1990,2000,2010])
```

[51]:
```python
cars['Year Interval'] = cars['Year Interval'].astype(int)
```

[52]:
```python
cars.dtypes
```

[52]:
```
ID                    int64
Price               float64
Levy                float64
Manufacturer         object
Model                object
Prod. year            int64
Category              int32
Leather interior      int32
Fuel type             int32
Engine volume       float64
Mileage             float64
Cylinders             int64
Gear box type         int32
Drive wheels          int32
Doors                 int64
Wheel                 int32
Color                object
Airbags               int64
Year Interval         int32
dtype: object
```

[53]:
```python
#Dropping the columns that we don't need for our analysis
cars = cars.drop(['ID', 'Manufacturer', 'Model', 'Color', 'Prod. year'], axis =␣
↪1)
cars
```

[53]:

	Price	Levy	Category	Leather interior	Fuel type	\
0	13328.0	1399.000000	4	1	2	
1	16621.0	1018.000000	4	0	5	
2	8467.0	906.299205	3	0	5	
3	3607.0	862.000000	4	1	2	
4	11726.0	446.000000	3	1	5	
…	…	…	…	…	…	
19232	8467.0	906.299205	1	1	0	
19233	15681.0	831.000000	9	1	5	
19234	26108.0	836.000000	4	1	1	
19235	5331.0	1288.000000	4	1	1	

```
19236    470.0   753.000000         9                  1          2

       Engine volume        Mileage  Cylinders  Gear box type  Drive wheels  \
0                3.5  115602.858919          6              0             0
1                3.0  119328.775637          6              2             0
2                1.3  124300.807955          4              3             1
3                2.5  105013.051585          4              0             0
4                1.3   57116.842759          4              0             1
…                 …              …          …              …            …
19232            2.0  186451.211933          4              1             2
19233            2.4  100435.052828          4              2             1
19234            2.0   72321.317589          4              0             1
19235            2.0   31857.054071          4              0             1
19236            2.4  116173.399627          4              0             1

       Doors  Wheel  Airbags  Year Interval
0          4      1       12           2000
1          4      1        8           2010
2          4      0        2           2000
3          4      1        0           2010
4          4      1        4           2010
…        …     …       …             …
19232      2      1        5           1990
19233      4      1        8           2010
19234      4      1        4           2000
19235      4      1        4           2000
19236      4      1       12           2010

[18922 rows x 14 columns]
```

[54]:
```python
#Creating a Correlation Matrix to identify the relationships
cor_matrix = cars.corr()
cor_matrix
```

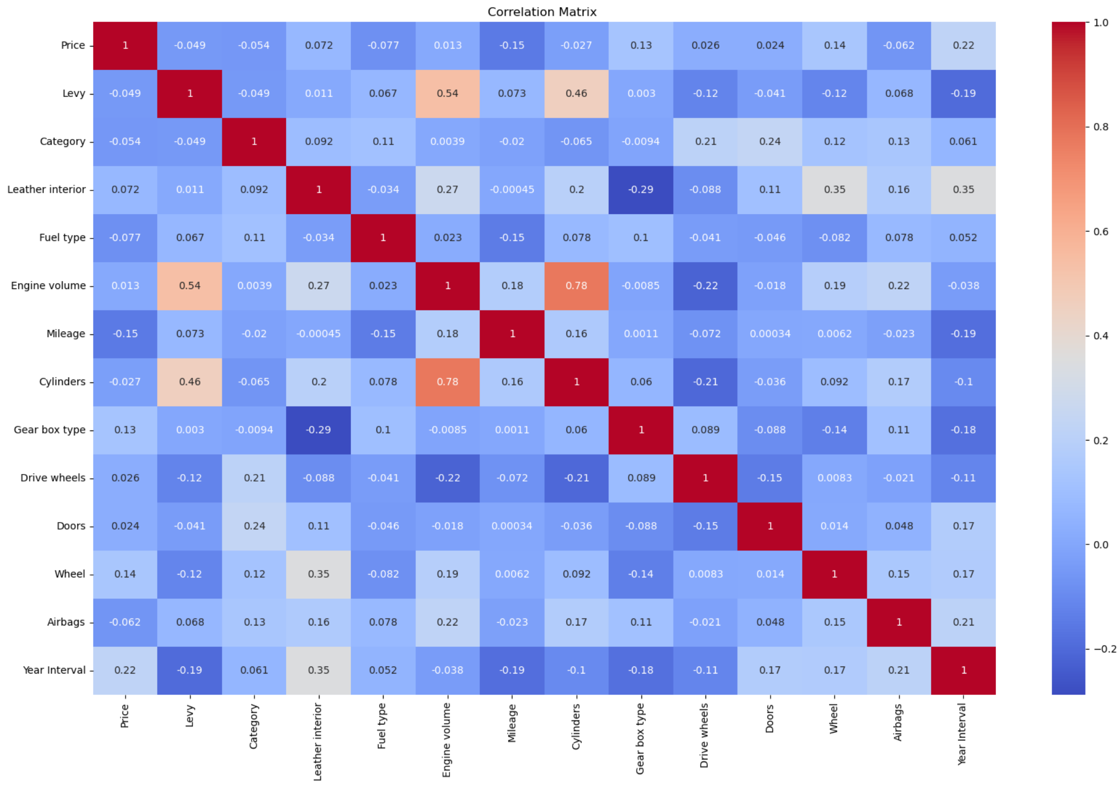
[54]:
```
                      Price      Levy  Category  Leather interior  Fuel type  \
Price              1.000000 -0.049339 -0.054161          0.071535  -0.077177
Levy              -0.049339  1.000000 -0.048773          0.011110   0.066871
Category          -0.054161 -0.048773  1.000000          0.092222   0.112219
Leather interior   0.071535  0.011110  0.092222          1.000000  -0.033962
Fuel type         -0.077177  0.066871  0.112219         -0.033962   1.000000
Engine volume      0.012611  0.537900  0.003950          0.271770   0.022790
Mileage           -0.149281  0.073371 -0.020399         -0.000446  -0.145110
Cylinders         -0.027497  0.459809 -0.064786          0.199709   0.078432
Gear box type      0.126295  0.003041 -0.009389         -0.288332   0.103600
Drive wheels       0.025958 -0.121991  0.212471         -0.087913  -0.041288
Doors              0.024269 -0.040903  0.237971          0.106839  -0.045759
Wheel              0.138458 -0.116834  0.119425          0.346664  -0.081819
```

Airbags	-0.061931	0.067882	0.134575	0.161838	0.078001
Year Interval	0.220925	-0.193565	0.060869	0.347812	0.051609

	Engine volume	Mileage	Cylinders	Gear box type	\
Price	0.012611	-0.149281	-0.027497	0.126295	
Levy	0.537900	0.073371	0.459809	0.003041	
Category	0.003950	-0.020399	-0.064786	-0.009389	
Leather interior	0.271770	-0.000446	0.199709	-0.288332	
Fuel type	0.022790	-0.145110	0.078432	0.103600	
Engine volume	1.000000	0.175918	0.777237	-0.008490	
Mileage	0.175918	1.000000	0.155393	0.001147	
Cylinders	0.777237	0.155393	1.000000	0.060021	
Gear box type	-0.008490	0.001147	0.060021	1.000000	
Drive wheels	-0.222435	-0.072303	-0.206221	0.088663	
Doors	-0.017697	0.000345	-0.036024	-0.087742	
Wheel	0.185996	0.006190	0.091971	-0.136051	
Airbags	0.222006	-0.022878	0.174067	0.109907	
Year Interval	-0.038372	-0.185211	-0.099701	-0.179269	

	Drive wheels	Doors	Wheel	Airbags	Year Interval
Price	0.025958	0.024269	0.138458	-0.061931	0.220925
Levy	-0.121991	-0.040903	-0.116834	0.067882	-0.193565
Category	0.212471	0.237971	0.119425	0.134575	0.060869
Leather interior	-0.087913	0.106839	0.346664	0.161838	0.347812
Fuel type	-0.041288	-0.045759	-0.081819	0.078001	0.051609
Engine volume	-0.222435	-0.017697	0.185996	0.222006	-0.038372
Mileage	-0.072303	0.000345	0.006190	-0.022878	-0.185211
Cylinders	-0.206221	-0.036024	0.091971	0.174067	-0.099701
Gear box type	0.088663	-0.087742	-0.136051	0.109907	-0.179269
Drive wheels	1.000000	-0.145813	0.008274	-0.020755	-0.114501
Doors	-0.145813	1.000000	0.013542	0.048115	0.170801
Wheel	0.008274	0.013542	1.000000	0.146663	0.166060
Airbags	-0.020755	0.048115	0.146663	1.000000	0.212378
Year Interval	-0.114501	0.170801	0.166060	0.212378	1.000000

[55]:
```python
#Plotting a heatmap (multivariate)
import seaborn as sns

fig, ax = plt.subplots(figsize=(20, 12))
sns.heatmap(cor_matrix, annot=True, cmap='coolwarm', ax=ax)
ax.set_title("Correlation Matrix ")
plt.show()
```



We can see that ‘Cylinders’ and ‘Engine Volume’ have a high correlation - This shows that a car with a bigger engine can hold more cylinders in its capacity.

[56]:
```
cars.columns
```

[56]:
```
Index(['Price', 'Levy', 'Category', 'Leather interior', 'Fuel type',
       'Engine volume', 'Mileage', 'Cylinders', 'Gear box type',
       'Drive wheels', 'Doors', 'Wheel', 'Airbags', 'Year Interval'],
      dtype='object')
```

[57]:
```
#Dividing into Predictors and Outcome(s)
#Feature Selection will be based on the demographic information and the␣
 ↪relationships mentioned in the question
predictors = ['Levy', 'Category', 'Leather interior', 'Engine volume',␣
 ↪'Mileage', 'Fuel type','Gear box type','Drive wheels',
       'Cylinders', 'Doors', 'Wheel', 'Airbags', 'Year Interval']
outcome = 'Price'
```

[58]:
```
# Partitioning the data into predictors (X) and output (Y)
X = pd.get_dummies(cars[predictors], drop_first=True)
Y = cars[outcome]
```

3.0.8 8. Validation Split

[59]:
```
from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

from dmba import regressionSummary, exhaustive_search
from dmba import backward_elimination, forward_selection, stepwise_selection
from dmba import adjusted_r2_score, AIC_score, BIC_score
```

[60]:
```
#Splitting the data into training and test set
train_X, valid_X, train_Y, valid_Y = train_test_split(X, Y, test_size = 0.4 ,␣
↪random_state = 3)
```

[61]:
```
#Performing backward elimination feature selection
def train_model(variables):
    model = LinearRegression()
    model.fit(train_X[variables], train_Y)
    return model

def score_model(model, variables):
    return AIC_score(train_Y, model.predict(train_X[variables]), model)

best_model, best_variables = backward_elimination(train_X.columns, train_model,␣
↪score_model, verbose=True)

print(best_variables)

#Performing backward elimination feature selectionregressionSummary(valid_Y,␣
↪best_model.predict(valid_X[best_variables]))
```

```
Variables: Levy, Category, Leather interior, Engine volume, Mileage, Fuel type,
Gear box type, Drive wheels, Cylinders, Doors, Wheel, Airbags, Year Interval
Start: score=241787.05
Step: score=241785.66, remove Levy
Step: score=241784.82, remove Leather interior
Step: score=241784.82, remove None
['Category', 'Engine volume', 'Mileage', 'Fuel type', 'Gear box type', 'Drive
wheels', 'Cylinders', 'Doors', 'Wheel', 'Airbags', 'Year Interval']
```

[62]:
```
#Performing forward feature selection
def train_model(variables):
    if len(variables) == 0:
        return None
    model = LinearRegression()
    model.fit(train_X[variables], train_Y)
    return model
```

```
def score_model(model, variables):
    if len(variables) == 0:
        return AIC_score(train_Y, [train_Y.mean()] * len(train_Y), model, df=1)
    return AIC_score(train_Y, model.predict(train_X[variables]), model)

best_model, best_variables = forward_selection(train_X.columns, train_model,␣
 ↪score_model, verbose=True)

print(best_variables)

regressionSummary(valid_Y, best_model.predict(valid_X[best_variables]))
```

```
Variables: Levy, Category, Leather interior, Engine volume, Mileage, Fuel type,
Gear box type, Drive wheels, Cylinders, Doors, Wheel, Airbags, Year Interval
Start: score=243663.84, constant
Step: score=243081.34, add Year Interval
Step: score=242730.10, add Gear box type
Step: score=242497.39, add Airbags
Step: score=242224.38, add Wheel
Step: score=242082.13, add Mileage
Step: score=241917.37, add Fuel type
Step: score=241875.51, add Category
Step: score=241840.75, add Engine volume
Step: score=241811.19, add Drive wheels
Step: score=241792.19, add Cylinders
Step: score=241784.82, add Doors
Step: score=241784.82, add None
['Year Interval', 'Gear box type', 'Airbags', 'Wheel', 'Mileage', 'Fuel type',
'Category', 'Engine volume', 'Drive wheels', 'Cylinders', 'Doors']

Regression statistics

                      Mean Error (ME) : -68.2835
       Root Mean Squared Error (RMSE) : 10237.5982
            Mean Absolute Error (MAE) : 7977.0347
          Mean Percentage Error (MPE) : -1583.4091
Mean Absolute Percentage Error (MAPE) : 1619.2330
```

Thus, it can be seen that 'Levy' and 'Leather interior' must be removed from our linear regression model in order to have the algorithm predict car price(s) more accurately.

3.0.9 9. Exploratory Data Analysis (EDA)

[63]:
```
cars_2.head()
```

[63]:
```
         ID  Price         Levy Manufacturer    Model  Prod. year   Category  \
0  45654403  13328  1399.000000        LEXUS   RX 450        2010       Jeep
```

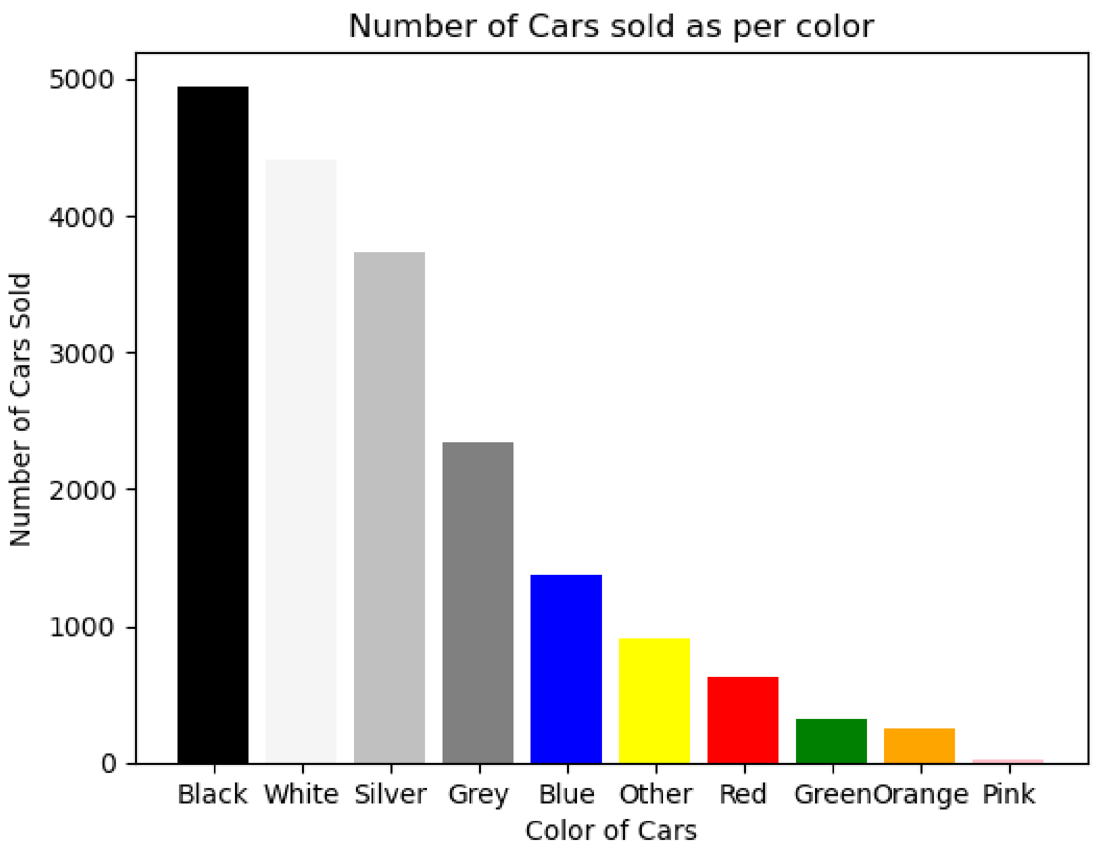
```
1  44731507  16621  1018.000000    CHEVROLET  Equinox        2011        Jeep
2  45774419   8467   906.299205        HONDA      FIT        2006   Hatchback
3  45769185   3607   862.000000         FORD   Escape        2011        Jeep
4  45809263  11726   446.000000        HONDA      FIT        2014   Hatchback

  Leather interior Fuel type  Engine volume  Mileage  Cylinders Gear box type  \
0              Yes    Hybrid            3.5   186005          6     Automatic
1               No    Petrol            3.0   192000          6     Tiptronic
2               No    Petrol            1.3   200000          4      Variator
3              Yes    Hybrid            2.5   168966          4     Automatic
4              Yes    Petrol            1.3    91901          4     Automatic

  Drive wheels  Doors        Wheel   Color  Airbags
0          4x4      4   Left wheel  Silver       12
1          4x4      4   Left wheel   Black        8
2        Front      4  Right wheel   Black        2
3          4x4      4   Left wheel   White        0
4        Front      4   Left wheel  Silver        4
```

[64]:
```
#Analyzing the cars sold based on color types (univariate)

#Categorising and clubbing a few car colors into one 'Other'
cars_2['Color'].replace({'Purple' : 'Other', 'Yellow' : 'Other', 'Sky blue' :␣
 ↪'Other',
                        'Golden' : 'Other', 'Carnelian red' : 'Other', 'Beige' :␣
 ↪'Other', 'Brown' : 'Other'}, inplace=True)
color_counts = cars_2['Color'].value_counts()
color_counts
```

[64]:
```
Black     4944
White     4406
Silver    3728
Grey      2343
Blue      1376
Other      905
Red        622
Green      321
Orange     252
Pink        25
Name: Color, dtype: int64
```

[65]:
```
#Creating a custom color palette

color_palette = ['black', 'whitesmoke', 'silver', 'grey', 'blue', 'yellow',␣
 ↪'red', 'green', 'orange', 'pink']

plt.bar(color_counts.index, color_counts.values, color = color_palette)
```

```
plt.xlabel('Color of Cars')
plt.ylabel('Number of Cars Sold')
plt.title('Number of Cars sold as per color')

plt.show()
```



As seen, "Black" is the most preferred color for car buyers, followed by "White" and "Silver".

[66]:
```
import seaborn as sns
```

[67]:
```
#Analyzing the cars sold based on the their Category (univariate)

#Categorising and clubbing a few car categories into one 'Other'
cars_2['Category'].replace({'Cabriolet' : 'Other', 'Pickup' : 'Other', 'Goods␣
↪wagon' : 'Other',
                          'Universal' : 'Other', 'Limousine' : 'Other',␣
↪'Microbus' : 'Other'}, inplace=True)

Categories = cars_2['Category'].value_counts()
```

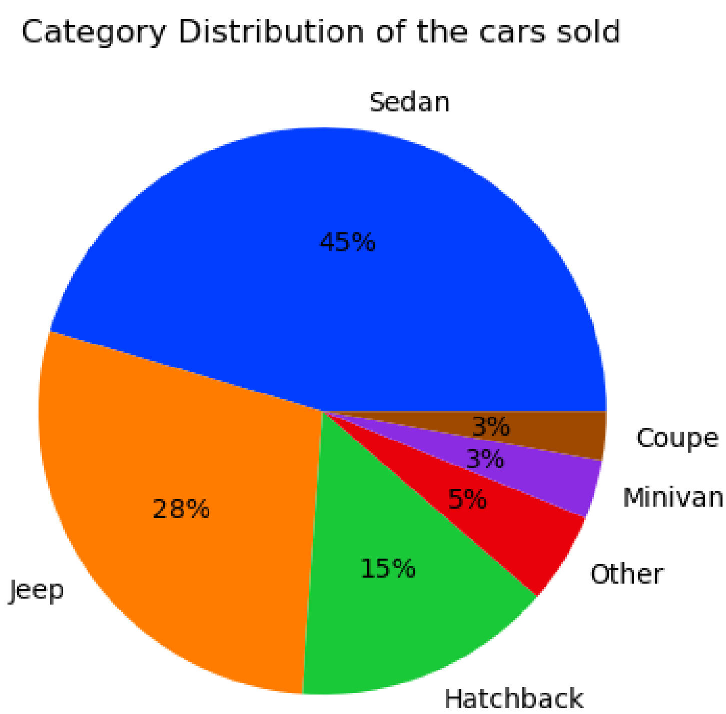
```
Categories
```

[67]:

```
Sedan        8600
Jeep         5377
Hatchback    2799
Other         985
Minivan       633
Coupe         528
Name: Category, dtype: int64
```

[68]:

```
palette_color = sns.color_palette('bright')
plt.pie(Categories, labels = Categories.index, colors = palette_color, autopct␣
 ↪= '%.0f%%')
plt.title('Category Distribution of the cars sold')

plt.show()
```

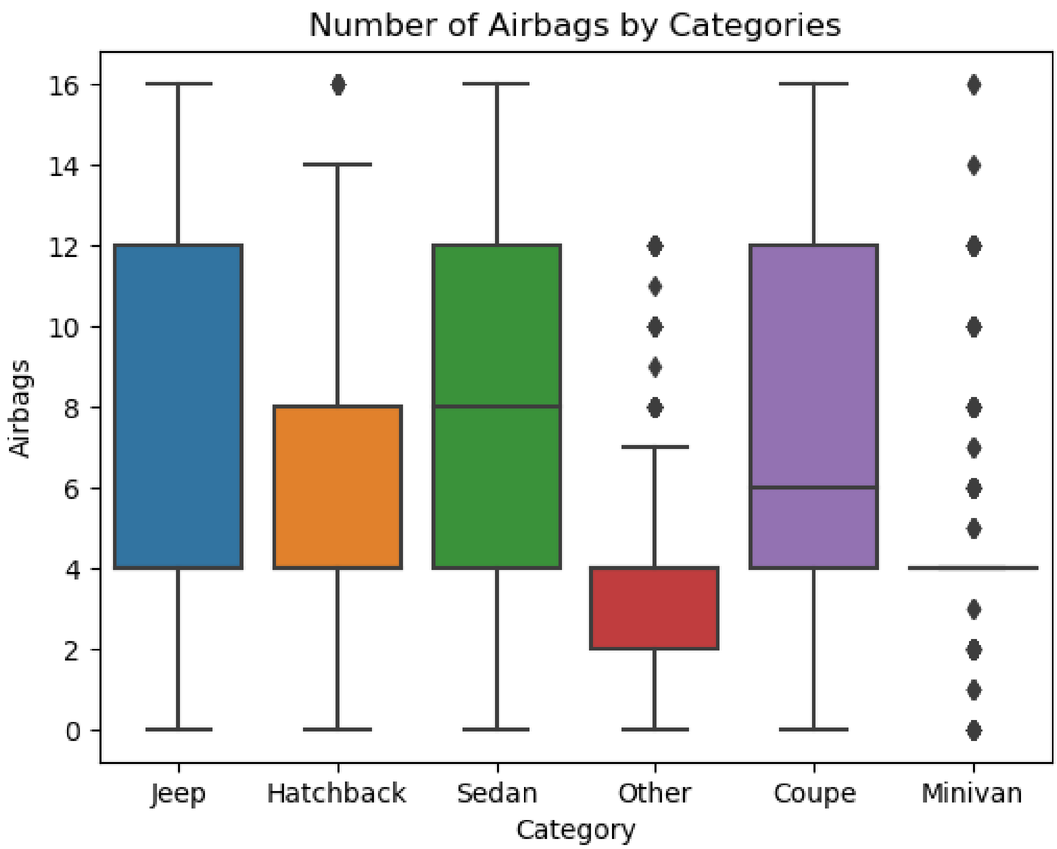


As from the chart above, "Sedan" is the highest sold category in second-hand car sales, followed by "Jeep" and "Hatchback". The other categories of cars account about a tenth of the total car sales.

[69]:

```
#Creating a Boxplot for each category (univariate)
```

```
sns.boxplot(x = "Category" , y = "Airbags" , data = cars_2)
plt.title("Number of Airbags by Categories")

plt.show()
```



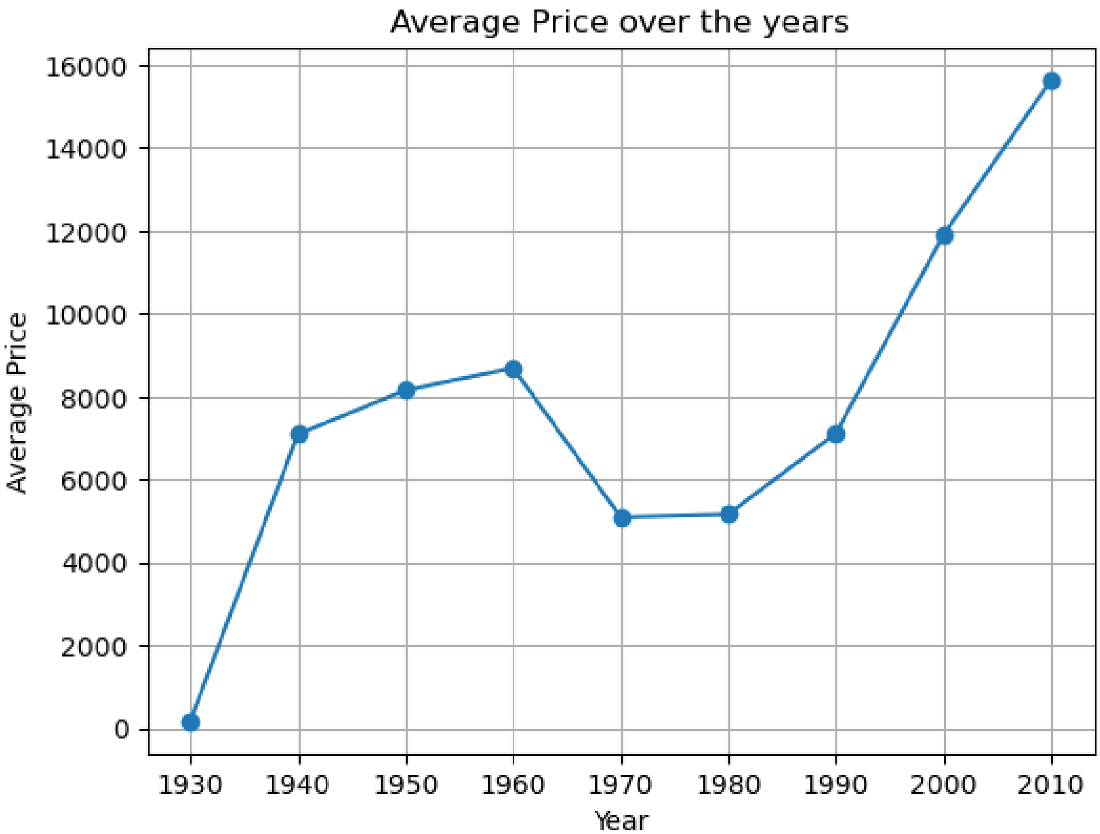
From the boxplot above, it can be seen that the average range of airbags count for most categories lies between 4 and 12, with a few outliers extending from 0 to 16, prominently for "Minivan".

[70]:
```
grouped_data_1 = cars.groupby('Year Interval')

#Calculating the mean value for each group
mean_values_1 = grouped_data_1.mean()
mean_values_1
```

[70]:
```
                     Price        Levy  Category  Leather interior  \
Year Interval
1930            171.333333  906.299205  3.333333          1.000000
1940           7094.866318  906.299205  7.000000          0.500000
1950           8152.346527  906.299205  4.600000          0.000000
```

```
1960             8690.093054  1518.039364  3.800000          0.600000
1970             5094.333333   906.299205  6.555556          0.111111
1980             5164.467742   906.299205  6.903226          0.145161
1990             7113.367273   965.903490  6.345186          0.252385
2000            11898.407163  1052.655317  5.842166          0.600422
2010            15633.366095   839.445091  6.435912          0.825251

               Fuel type  Engine volume       Mileage  Cylinders  \
Year Interval
1930            5.000000       3.200000  87839.237622   5.333333
1940            5.000000       2.100000  72715.972654   5.000000
1950            5.000000       1.760000  62150.403978   4.000000
1960            4.000000       3.320000  31006.835504   5.600000
1970            4.444444       2.100000  51675.850537   4.777778
1980            3.709677       2.062903  72977.956320   4.290323
1990            3.176930       2.228187  88645.285799   4.615785
2000            3.278802       2.424597  99299.942609   4.847926
2010            3.508377       2.265050  71967.959321   4.466132

               Gear box type  Drive wheels     Doors     Wheel   Airbags
Year Interval
1930                0.333333      2.000000  4.000000  1.000000  0.000000
1940                0.500000      2.000000  4.000000  1.000000  0.000000
1950                1.000000      1.200000  3.200000  1.000000  0.200000
1960                0.600000      1.000000  2.800000  1.000000  4.800000
1970                0.888889      1.888889  3.555556  1.000000  2.777778
1980                0.935484      1.274194  3.580645  1.000000  1.112903
1990                0.856028      1.241977  3.666956  0.890720  3.392021
2000                0.754992      0.884601  3.901882  0.829301  6.137865
2010                0.414910      0.886172  3.961764  0.964890  7.078236
```

[71]:
```python
#Making a Linechart to understand the Price changes over the years (bivariate)

x_values_1 = mean_values_1.index
y_values_1 = mean_values_1['Price']
plt.plot(x_values_1, y_values_1, marker = 'o')
plt.xlabel('Year')
plt.ylabel('Average Price')
plt.title('Average Price over the years')

plt.grid(True)
plt.show()
```



As seen, average car prices have risen over the years with an exception (dip) around the 1970s-1980s, and a steep increase thereafter.

[72]:
```
#Making a Treemap to understand the distribution of cars by different␣
↪manufacturers (multivariate)

fig = px.treemap(data_frame = cars_2, path=["Manufacturer", "Category",␣
↪"Model"],
                 values = 'Price', title = 'Sales distribution by Manufacturer')
fig.show()
```

As from the treemap above, "Hyundai" seems to be the manufacturer with the most number of cars sold, followed by "Toyota" and "Mercedes". This exhibits that Hyundai targets a range of customers with varied tastes and preferences.

[73]:
```
#Imputing the outliers in 'Price' using the quartile method for cars_2
q1, q3 = np.percentile(cars_2["Price"], [25, 75])
iqr = q3 - q1
lower_bound = q1 - 1.5*iqr
upper_bound = q3 + 1.5*iqr
```

```
#Creating conditions to isolate the outliers
outliers_price_2 = cars_2["Price"][(cars_2["Price"] < lower_bound) |
 ↪(cars_2["Price"] > upper_bound)]

mean_Price_without_outliers_2 = cars_2["Price"][(cars_2["Price"] >=
 ↪lower_bound) & (cars_2["Price"] <= upper_bound)].mean()

#Replacing the outliers with the mean_Price_without_outliers_2
cars_2["Price"] = cars_2["Price"].where(~((cars_2["Price"] < lower_bound) |
 ↪(cars_2["Price"] > upper_bound)), mean_Price_without_outliers_2)
```

[74]:
```
grouped_data_2 = cars_2.groupby('Fuel type')

#Calculating the mean value for each group
mean_values_2 = grouped_data_2.mean()
mean_values_2
```

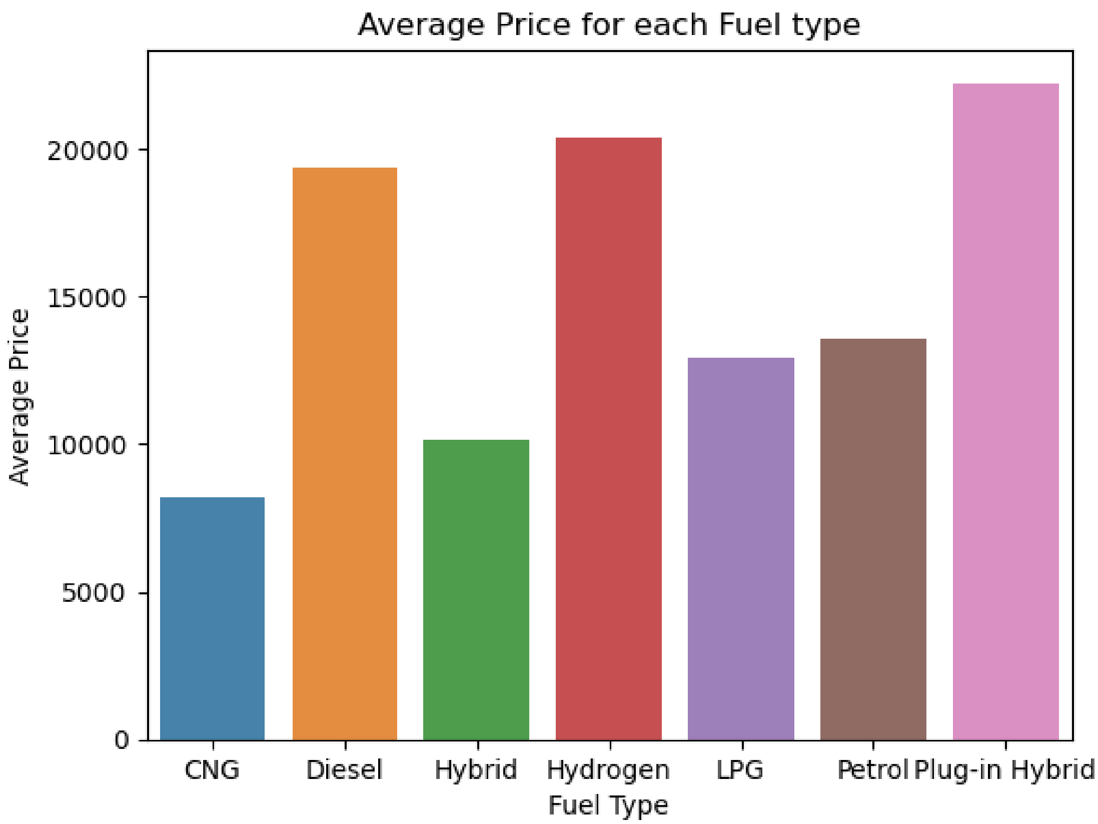
[74]:

Fuel type	ID	Price	Levy	Prod. year
CNG	4.565077e+07	8186.859275	948.270665	1999.880597
Diesel	4.562996e+07	19380.497469	936.673209	2010.976500
Hybrid	4.551084e+07	10187.959660	771.696828	2012.179147
Hydrogen	4.578407e+07	20385.000000	906.299205	2012.000000
LPG	4.571719e+07	12943.936868	839.868818	2012.004520
Petrol	4.556194e+07	13564.942696	947.793357	2010.844815
Plug-in Hybrid	4.544400e+07	22247.181944	687.378412	2013.070588

Fuel type	Engine volume	Mileage	Cylinders	Doors	Airbags
CNG	2.479744	2.352272e+07	4.916844	3.925373	4.703625
Diesel	2.387850	7.167544e+05	4.543250	3.913000	5.436000
Hybrid	2.081153	4.753292e+05	4.273241	3.998587	7.784402
Hydrogen	2.400000	1.168000e+05	6.000000	4.000000	8.000000
LPG	2.236836	2.722707e+05	4.355932	3.992090	4.639548
Petrol	2.357075	1.367855e+06	4.712562	3.897717	6.829327
Plug-in Hybrid	1.657647	1.226019e+05	4.094118	4.000000	9.176471

[75]:
```
#Making a Barchart to understand the distribution of cars' Fuel type(s) by
 ↪their prices (bivariate)

x_values_2 = mean_values_2.index
y_values_2 = mean_values_2['Price']
sns.barplot(x = x_values_2, y = y_values_2, alpha = 0.9)
plt.xlabel('Fuel Type')
plt.ylabel('Average Price')
plt.title('Average Price for each Fuel type')
```

```
plt.show()
```



As seen in the Barchart above, cars with the fuel type "Plug-in Hybrid" are the most expensive, on average, followed by "Hydrogen". Cars with the fuel type "CNG" are the cheapest overall.

3.1 We have performed the necessary data cleaning, wrangling and mining along with EDA, and are prepared to build a machine learning model.

4 MILESTONE 3

[76]:
```
cars.head()
```

[76]:

	Price	Levy	Category	Leather interior	Fuel type	Engine volume	\
0	13328.0	1399.000000	4	1	2	3.5	
1	16621.0	1018.000000	4	0	5	3.0	
2	8467.0	906.299205	3	0	5	1.3	
3	3607.0	862.000000	4	1	2	2.5	
4	11726.0	446.000000	3	1	5	1.3	

```
        Mileage  Cylinders  Gear box type  Drive wheels  Doors  Wheel  \
0  115602.858919          6              0             0      4      1
1  119328.775637          6              2             0      4      1
2  124300.807955          4              3             1      4      0
3  105013.051585          4              0             0      4      1
4   57116.842759          4              0             1      4      1

   Airbags  Year Interval
0       12           2000
1        8           2010
2        2           2000
3        0           2010
4        4           2010
```

[77]:
```
cars.dtypes
```

[77]:
```
Price               float64
Levy                float64
Category              int32
Leather interior      int32
Fuel type             int32
Engine volume       float64
Mileage             float64
Cylinders             int64
Gear box type         int32
Drive wheels          int32
Doors                 int64
Wheel                 int32
Airbags               int64
Year Interval         int32
dtype: object
```

4.0.1 We saw in the Feature Selection process that 'Levy' and 'Leather interior' variables need to be dropped; We can then work on our chosen Machine Learning models.

[78]:
```
cars_3 = cars.drop(['Levy', 'Leather interior'], axis = 1)
```

[79]:
```
cars_3.columns
```

[79]:
```
Index(['Price', 'Category', 'Fuel type', 'Engine volume', 'Mileage',
       'Cylinders', 'Gear box type', 'Drive wheels', 'Doors', 'Wheel',
       'Airbags', 'Year Interval'],
      dtype='object')
```

[80]:
```
predictors_2 = ['Category', 'Fuel type', 'Engine volume', 'Mileage',
       'Cylinders', 'Gear box type', 'Drive wheels', 'Doors', 'Wheel',
```

```
        'Airbags', 'Year Interval']
outcome_2 = 'Price'
```

[81]:
```
x = pd.get_dummies(cars_3[predictors_2], drop_first = True)
y = cars_3[outcome_2]
```

[82]:
```
y
```

[82]:
```
0         13328.0
1         16621.0
2          8467.0
3          3607.0
4         11726.0
           …
19232      8467.0
19233     15681.0
19234     26108.0
19235      5331.0
19236       470.0
Name: Price, Length: 18922, dtype: float64
```

4.0.2 Training and Test Split

[83]:
```
#Splitting the data into training and test set
train_x, valid_x, train_y, valid_y = train_test_split(x, y, test_size = 0.4,␣
 ↪random_state = 1)
```

4.1 We will build two predictive models - Linear Regression (with Lasso and Ridge optimization) and Random Forest.

4.2 Linear Regression Algorithm

[84]:
```
Cars_LR = LinearRegression()
```

[85]:
```
Cars_LR.fit(train_x, train_y)
```

[85]:
```
LinearRegression()
```

[86]:
```
#Printing the coefficients
print('intercept:', Cars_LR.intercept_)
print(pd.DataFrame({'Predictor': x.columns, 'coefficient': Cars_LR.coef_}))
```

```
intercept: -876709.3427277593
        Predictor  coefficient
0        Category  -294.347078
1       Fuel type  -586.421917
2   Engine volume  1435.465649
3         Mileage    -0.028959
```

```
4         Cylinders   -508.355536
5     Gear box type   2876.507413
6      Drive wheels   1002.106395
7             Doors    701.383172
8             Wheel   5906.281333
9           Airbags   -403.231897
10    Year Interval    442.721882
```

[87]:

```python
#Making predictions on the test data
y_pred_1 = Cars_LR.predict(valid_x)
y_pred_df = pd.DataFrame({'Predicted': y_pred_1})
y_pred_df
```

[87]:

```
         Predicted
0     19435.853672
1     10113.230229
2     20682.767588
3     12073.983795
4     14022.946113
…              …
7564  10730.177845
7565   4643.017585
7566  11786.637663
7567  20041.002204
7568  13378.501259

[7569 rows x 1 columns]
```

[88]:

```python
from sklearn.metrics import r2_score
```

[89]:

```python
#Calculating the R-squared score
r1 = r2_score(valid_y, y_pred_1)
print("R-squared score:", r1)
```

```
R-squared score: 0.14188155699947191
```

4.3 Random Forest Algorithm

[90]:

```python
from sklearn.ensemble import RandomForestRegressor
```

[91]:

```python
#We have taken our hyper-parameter (number of decision trees) as 100 to try to␣
↪improve the model performance
rf_model = RandomForestRegressor(n_estimators = 100, random_state = 1)
```

[92]:

```python
rf_model.fit(train_x, train_y)
```

[92]:

```
RandomForestRegressor(random_state=1)
```

[93]:
```
#Making predictions on the test data
y_pred_2 = rf_model.predict(valid_x)
y_pred_2df = pd.DataFrame({'Predicted': y_pred_2})
y_pred_2df
```

[93]:
```
         Predicted
0     30851.077916
1       545.850000
2      8900.427333
3     10847.730000
4     16310.527560
…              …
7564  15170.390000
7565   7658.964167
7566  15508.500000
7567  16243.353571
7568  17898.200080

[7569 rows x 1 columns]
```

[94]:
```
#Calculating the R-squared score
r2 = r2_score(valid_y, y_pred_2)
print("R-squared score:", r2)
```

```
R-squared score: 0.5357291795068264
```

4.4 Lasso (Linear Regression) Algorithm

[95]:
```
from sklearn.linear_model import Lasso
```

[96]:
```
#We have taken our hyper-parameter (regularization strength) as 0.1 to try to␣
↪improve the model performance by penalizing features
lasso_model = Lasso(alpha = 0.1)
```

[97]:
```
lasso_model.fit(train_x, train_y)
```

[97]: Lasso(alpha=0.1)

[98]:
```
#Making predictions on the test data
y_pred_3 = lasso_model.predict(valid_x)
y_pred_3df = pd.DataFrame({'Predicted': y_pred_3})
y_pred_3df
```

[98]:
```
         Predicted
0     19435.512099
1     10113.197278
2     20683.542968
3     12073.876024
```

```
4     14023.121445
…              …
7564  10730.559644
7565   4644.490640
7566  11786.700788
7567  20040.667372
7568  13378.220397

[7569 rows x 1 columns]
```

[99]:
```
#Calculating the R-squared score
r3 = r2_score(valid_y, y_pred_3)
print("R-squared score:", r3)
```

```
R-squared score: 0.14188334035891337
```

4.5 Ridge (Linear Regression) Algorithm

[100]:
```
from sklearn.linear_model import Ridge
```

[101]:
```
#We have taken our hyper-parameter (regularization strength) as 1 to try to␣
↪improve the model performance by penalizing features
ridge_model = Ridge(alpha = 1)
```

[102]:
```
ridge_model.fit(train_x, train_y)
```

[102]:
```
Ridge(alpha=1)
```

[103]:
```
#Making predictions on the test data
y_pred_4 = ridge_model.predict(valid_x)
y_pred_4df = pd.DataFrame({'Predicted': y_pred_4})
y_pred_4df
```

[103]:
```
         Predicted
0     19435.306770
1     10113.416212
2     20688.334992
3     12073.754776
4     14022.485274
…              …
7564  10730.283237
7565   4650.003525
7566  11786.452580
7567  20039.433361
7568  13378.389630

[7569 rows x 1 columns]
```

[104]:
```
#Calculating the R-squared score
r4 = r2_score(valid_y, y_pred_4)
print("R-squared score:", r4)
```

```
R-squared score: 0.14189020110392536
```

R-squared (R^2) is a statistical metric used to evaluate the goodness of fit of a regression model. It provides a measure of how well the independent variables (features) explain the variability of the dependent variable (target) in the regression model. R-squared is also known as the coefficient of determination.

Mean Error (ME): The average difference between the predicted and actual values, indicating the overall bias of the model's predictions.

Root Mean Squared Error (RMSE): The square root of the average of the squared differences between predicted and actual values, representing the model's overall accuracy with a focus on larger errors.

Mean Absolute Error (MAE): The average of the absolute differences between predicted and actual values, providing a measure of the model's overall accuracy without considering the direction of errors.

Mean Percentage Error (MPE): The average percentage difference between predicted and actual values, indicating the model's overall bias in percentage terms.

Mean Absolute Percentage Error (MAPE): The average percentage difference between predicted and actual values, providing a relative measure of the model's accuracy.

4.5.1 The best metrics to evaluate model performance are R-squared, RMSE and MAE.

4.6 Regression Summary for Training Data

[105]:
```
#Simple Linear Regression
regressionSummary(train_y, Cars_LR.predict(train_x))
```

```
Regression statistics

                      Mean Error (ME) : 0.0000
       Root Mean Squared Error (RMSE) : 10229.2364
            Mean Absolute Error (MAE) : 7963.2868
          Mean Percentage Error (MPE) : -1372.4081
Mean Absolute Percentage Error (MAPE) : 1407.1201
```

[106]:
```
#Random Forest
regressionSummary(train_y, rf_model.predict(train_x))
```

```
Regression statistics

                      Mean Error (ME) : 17.4339
```

```
         Root Mean Squared Error (RMSE) : 3037.3739
              Mean Absolute Error (MAE) : 1877.3106
            Mean Percentage Error (MPE) : -339.6282
Mean Absolute Percentage Error (MAPE) : 348.3782
```

[107]:
```
#Lasso Regression
regressionSummary(train_y, lasso_model.predict(train_x))
```

```
Regression statistics

                       Mean Error (ME) : -0.0000
         Root Mean Squared Error (RMSE) : 10229.2364
              Mean Absolute Error (MAE) : 7963.2566
            Mean Percentage Error (MPE) : -1372.4356
Mean Absolute Percentage Error (MAPE) : 1407.1450
```

[108]:
```
#Ridge Regression
regressionSummary(train_y, ridge_model.predict(train_x))
```

```
Regression statistics

                       Mean Error (ME) : 0.0000
         Root Mean Squared Error (RMSE) : 10229.2366
              Mean Absolute Error (MAE) : 7963.1661
            Mean Percentage Error (MPE) : -1372.5140
Mean Absolute Percentage Error (MAPE) : 1407.2194
```

4.7 Regression Summary for Validation Data

[109]:
```
#Simple Linear Regression
regressionSummary(valid_y, Cars_LR.predict(valid_x))
```

```
Regression statistics

                       Mean Error (ME) : -254.5642
         Root Mean Squared Error (RMSE) : 10169.6145
              Mean Absolute Error (MAE) : 7934.0439
            Mean Percentage Error (MPE) : -1580.6005
Mean Absolute Percentage Error (MAPE) : 1616.1786
```

[110]:
```
#Random Forest
regressionSummary(valid_y, rf_model.predict(valid_x))
```

```
Regression statistics

                       Mean Error (ME) : -188.1848
```

```
       Root Mean Squared Error (RMSE) : 7480.2579
            Mean Absolute Error (MAE) : 4868.6604
          Mean Percentage Error (MPE) : -978.1230
Mean Absolute Percentage Error (MAPE) : 999.6086
```

[111]:
```
#Lasso Regression
regressionSummary(valid_y, lasso_model.predict(valid_x))
```

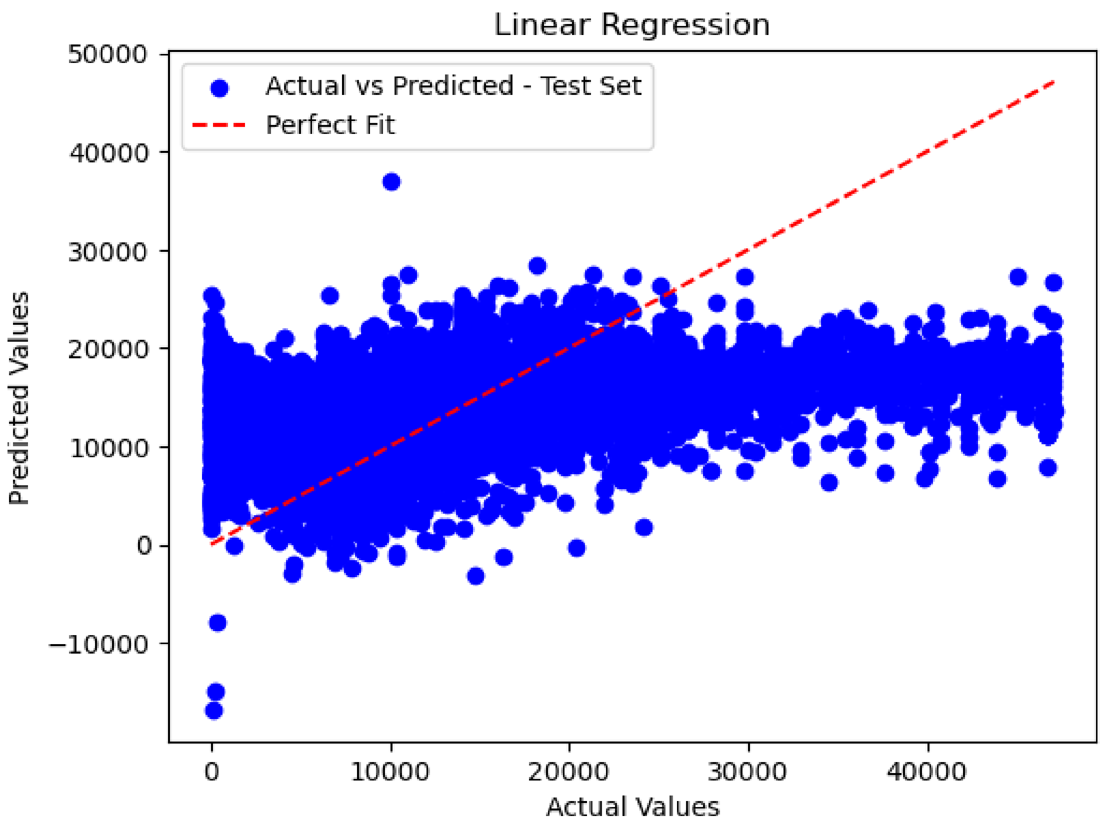
```
Regression statistics

                      Mean Error (ME) : -254.5767
       Root Mean Squared Error (RMSE) : 10169.6040
            Mean Absolute Error (MAE) : 7934.0087
          Mean Percentage Error (MPE) : -1580.6291
Mean Absolute Percentage Error (MAPE) : 1616.2065
```

[112]:
```
#Ridge Regression
regressionSummary(valid_y, ridge_model.predict(valid_x))
```

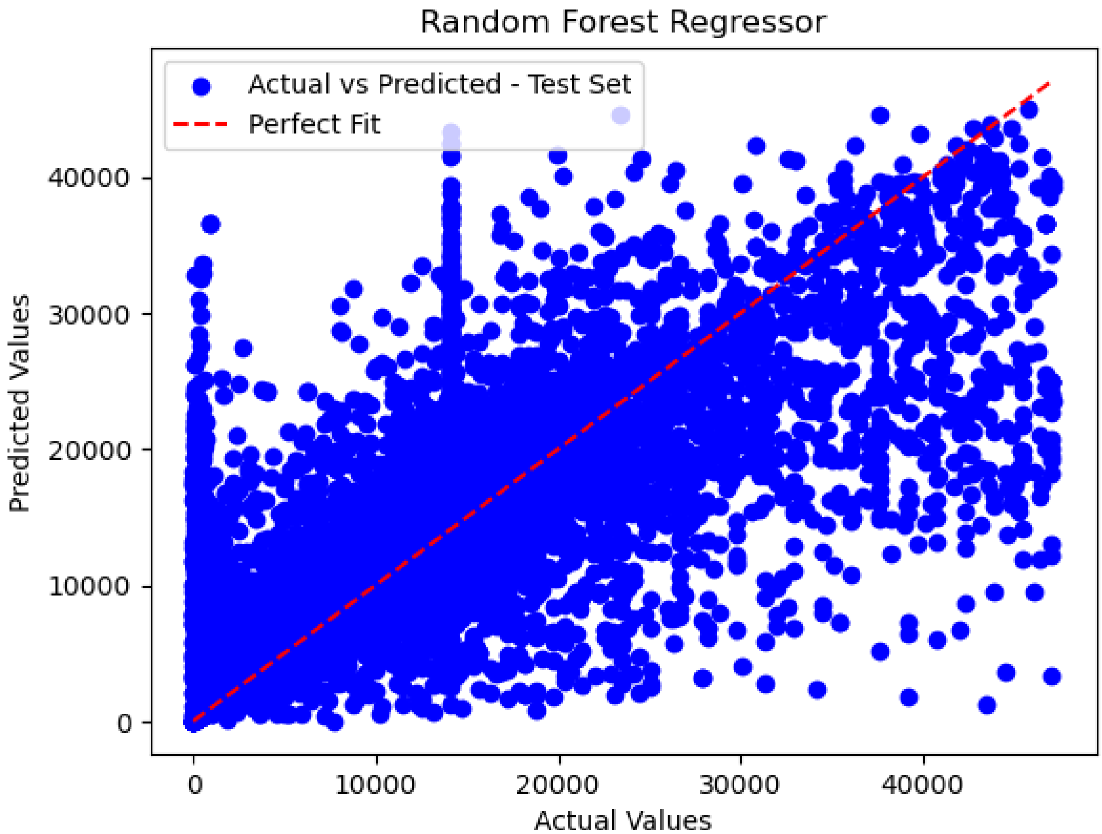
```
Regression statistics

                      Mean Error (ME) : -254.5805
       Root Mean Squared Error (RMSE) : 10169.5633
            Mean Absolute Error (MAE) : 7933.9005
          Mean Percentage Error (MPE) : -1580.6605
Mean Absolute Percentage Error (MAPE) : 1616.2362
```

[113]:
```
#Plotting the graphs for the two main models
plt.scatter(x = valid_y, y = y_pred_1, color = 'blue', label = 'Actual vs␣
 ↪Predicted - Test Set')
plt.plot([min(y), max(y)], [min(y), max(y)], color = 'red', linestyle='--',␣
 ↪label = 'Perfect Fit')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Linear Regression')
plt.legend()
plt.show()

plt.scatter(x = valid_y, y = y_pred_2, color = 'blue', label = 'Actual vs␣
 ↪Predicted - Test Set')
plt.plot([min(y), max(y)], [min(y), max(y)], color = 'red', linestyle='--',␣
 ↪label = 'Perfect Fit')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Random Forest Regressor')
plt.legend()
```

```
plt.show()
```



Linear Regression
Actual vs Predicted - Test Set
Perfect Fit
Predicted Values
Actual Values
50000
40000
30000
20000
10000
0
−10000
0
10000
20000
30000
40000



4.8 **From the Regression summary metrics above, we can see that the Linear Regression models have higher error as compared to the Random Forest Regressor. Therefore, it is in the best interest of ABC Motors to use a Random Forest algorithm in order to predict car prices and improve their profitability.**