EDA-KhangNguyen

May 27, 2024

1 EXPLORATORY ANALYSIS PROJECT

1.1 *Explore car market*

1.1.1 Objective

In this project, we're aiming to conduct an exploratory data analysis (EDA) to explore the factors on which the pricing of cars depends as well as studying about **car market** trends. This segmentation will contribute as a research for company that want to enter a new market.

Objective: 1. Understanding the current **car market** -> Their models, price, fuel types and car options 2. Gaining **insights** about **trends** in the market (popular fuel types, popular options, popular car category) 3. Understanding which **factors/variable affect car's price** and vice versa 4. Gaining information about **car price** in the past **10 years**

[1]:
```
import numpy as np                    #Linear Algebra
import pandas as pd                   #Dataset manipulation
import matplotlib
import matplotlib.pyplot as plt       #Visualisation
import seaborn as sns                 #Visualisation
import warnings
warnings.filterwarnings('ignore')
```

1.1.2 I. Data Exploration

[2]:
```
df = pd.read_csv("car_data.csv") #importing the dataset
print("Dataset's Shape: ", df.shape)
```

```
Dataset's Shape:  (19237, 18)
```

Data Description Summary of all data's attributes appear in the dataset: 1. **ID**: Unique identifier of each entry within the dataset 2. **Price**: Price of car (Target variable for machine learning) 3. **Levy**: Registration/Ownership tax's amount on each car 4. **Manufacturer**: Name of the car's company 5. **Model**: Name or designation of the model belongs to Manufacturer 6. **Prod. year**: Year of car being manufactured 7. **Category**: Type or Sub-class of car 8. **Leather interior**: Binary indicator (Yes/No) presents car's interior option (Leather) 9. **Fuel**: Type of fuel used by the car 10. **Engine Volumne**: Size of car's engine, measured in liter 11. **Mileage**: Total Distance traveled by car, measure by kilometres 12. **Cylinders**: Number of cyclinders inside the car's engine 13. **Gear box type**: Type of transmission such as automatic, manual … 14. **Drive wheels**: The configuration of wheels responsible for powering the car 15. **Doors**: Number of doors

16. **Wheel**: Wheel configuration such as left-hand drive (LHD) or right-hand drive (RHD) 17. **Color**: Exterior color of the car 18. **Airbags**: Number of airbages within the car (Safety features)

Data Inspection

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

[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
5  45802912  39493   891      HYUNDAI  Santa FE        2016       Jeep
6  45656768   1803   761       TOYOTA     Prius        2010  Hatchback
7  45816158    549   751      HYUNDAI    Sonata        2013      Sedan
8  45641395   1098   394       TOYOTA     Camry        2014      Sedan
9  45756839  26657     -        LEXUS    RX 350        2007       Jeep

  Leather interior Fuel type Engine volume    Mileage  Cylinders  \
0              Yes    Hybrid           3.5  186005 km        6.0
1               No    Petrol             3  192000 km        6.0
2               No    Petrol           1.3  200000 km        4.0
3              Yes    Hybrid           2.5  168966 km        4.0
4              Yes    Petrol           1.3   91901 km        4.0
5              Yes    Diesel             2  160931 km        4.0
6              Yes    Hybrid           1.8  258909 km        4.0
7              Yes    Petrol           2.4  216118 km        4.0
8              Yes    Hybrid           2.5  398069 km        4.0
9              Yes    Petrol           3.5  128500 km        6.0

  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
5     Automatic        Front  04-May        Left wheel   White        4
6     Automatic        Front  04-May        Left wheel   White       12
7     Automatic        Front  04-May        Left wheel    Grey       12
8     Automatic        Front  04-May        Left wheel   Black       12
9     Automatic          4x4  04-May        Left wheel  Silver       12
```

[4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19237 entries, 0 to 19236
Data columns (total 18 columns):
 #   Column            Non-Null Count  Dtype
```

```
---  ------            --------------  -----
 0   ID                19237 non-null  int64
 1   Price             19237 non-null  int64
 2   Levy              19237 non-null  object
 3   Manufacturer      19237 non-null  object
 4   Model             19237 non-null  object
 5   Prod. year        19237 non-null  int64
 6   Category          19237 non-null  object
 7   Leather interior  19237 non-null  object
 8   Fuel type         19237 non-null  object
 9   Engine volume     19237 non-null  object
 10  Mileage           19237 non-null  object
 11  Cylinders         19237 non-null  float64
 12  Gear box type     19237 non-null  object
 13  Drive wheels      19237 non-null  object
 14  Doors             19237 non-null  object
 15  Wheel             19237 non-null  object
 16  Color             19237 non-null  object
 17  Airbags           19237 non-null  int64
dtypes: float64(1), int64(4), object(13)
memory usage: 2.6+ MB
```

> From observation, we can see that the dataset contains both **numerical and categorical variables**, but does **not** consist of any missing values.

Let split the dataset into 2 categories **Numerical** and **Categorical** for further inspection

[5]:
```python
#List of Categorical variables
categorical = [i for i in df.columns if df[i].dtypes == 'O']
#List of numerical variables
numerical = [i for i in df.columns if i not in categorical]
print('Categorical: ',categorical)
print('Numerical: ',numerical)
```

```
Categorical:  ['Levy', 'Manufacturer', 'Model', 'Category', 'Leather interior',
'Fuel type', 'Engine volume', 'Mileage', 'Gear box type', 'Drive wheels',
'Doors', 'Wheel', 'Color']
Numerical:  ['ID', 'Price', 'Prod. year', 'Cylinders', 'Airbags']
```

> Observing the columns in Categorical subset, as **"Levy"** demonstrate the tax on the vehicle, it should not belongs to **categorical category**. Similarly, **"Engine Volume"** represent the **size of Engine** and should not be treated as **"Object"**. **"Mileage"** column is also the case as it should depict numerical values.

Data Cleaning and Preproccessing In this part, we prepare the data through performing data cleaning task such as **data duplicated**, **wrong data format**, **removing outliers** and dealing with **invalid/missing values**. First, starting by eliminating **old** data, in this scope we will use data from **1980**.

[6]:
```python
df = df[df['Prod. year'] >= 1980]
```

Next, let move to by inspecting and dealing with **dupplicated** values.

Checking Dupplicated Values

[7]:
```python
dup_data = df.duplicated().sum()
print("Duplicated data counted: ",dup_data)
```

```
Duplicated data counted:  312
```

Quite the amount of dupplication in the dataset ! We will eliminate these using "drop_duplicates" function

[8]:
```python
df = df.drop_duplicates()
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 18902 entries, 0 to 19236
Data columns (total 18 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   ID                18902 non-null  int64
 1   Price             18902 non-null  int64
 2   Levy              18902 non-null  object
 3   Manufacturer      18902 non-null  object
 4   Model             18902 non-null  object
 5   Prod. year        18902 non-null  int64
 6   Category          18902 non-null  object
 7   Leather interior  18902 non-null  object
 8   Fuel type         18902 non-null  object
 9   Engine volume     18902 non-null  object
 10  Mileage           18902 non-null  object
 11  Cylinders         18902 non-null  float64
 12  Gear box type     18902 non-null  object
 13  Drive wheels      18902 non-null  object
 14  Doors             18902 non-null  object
 15  Wheel             18902 non-null  object
 16  Color             18902 non-null  object
 17  Airbags           18902 non-null  int64
dtypes: float64(1), int64(4), object(13)
memory usage: 2.7+ MB
```

After dropping, the dataset is left with **18924** columns

Converting Column's Data Type Next, we convert datatype of columns into usefull one through identifying **invalid entries/ errors** within the dataset. Starting by inpsecting column **"Levy"** and finding its problem

```
[9]: #Inspecting the Levy's column
df.Levy.unique()
```

```
[9]: array(['1399', '1018', '-', '862', '446', '891', '761', '751', '394',
       '1053', '1055', '1079', '810', '2386', '1850', '531', '586',
       '1249', '2455', '583', '1537', '1288', '915', '1750', '707',
       '1077', '1486', '1091', '650', '382', '1436', '1194', '503',
       '1017', '1104', '639', '629', '919', '781', '530', '640', '765',
       '777', '779', '934', '769', '645', '1185', '1324', '830', '1187',
       '1111', '760', '642', '1604', '1095', '966', '473', '1138', '1811',
       '988', '917', '1156', '687', '11714', '836', '1347', '2866',
       '1646', '259', '609', '697', '585', '475', '690', '308', '1823',
       '1361', '1273', '924', '584', '2078', '831', '1172', '893', '1872',
       '1885', '1266', '447', '2148', '1730', '730', '289', '502', '333',
       '1325', '247', '879', '1342', '1327', '1598', '1514', '1058',
       '738', '1935', '481', '1522', '1282', '456', '880', '900', '798',
       '1277', '442', '1051', '790', '1292', '1047', '528', '1211',
       '1493', '1793', '574', '930', '1998', '271', '706', '1481', '1677',
       '1661', '1286', '1408', '1090', '595', '1451', '1267', '993',
       '1714', '878', '641', '749', '1511', '603', '353', '877', '1236',
       '1141', '397', '784', '1024', '1357', '1301', '770', '922', '1438',
       '753', '607', '1363', '638', '490', '431', '565', '517', '833',
       '489', '1760', '986', '1841', '1620', '1360', '474', '1099', '978',
       '1624', '1946', '1268', '1307', '696', '649', '666', '2151', '551',
       '800', '971', '1323', '2377', '1845', '1083', '694', '463', '419',
       '345', '1515', '1505', '2056', '1203', '729', '460', '1356', '876',
       '911', '1190', '780', '448', '2410', '1848', '1148', '834', '1275',
       '1028', '1197', '724', '890', '1705', '505', '789', '2959', '518',
       '461', '1719', '2858', '3156', '2225', '2177', '1968', '1888',
       '1308', '2736', '1103', '557', '2195', '843', '1664', '723',
       '4508', '562', '501', '2018', '1076', '1202', '3301', '691',
       '1440', '1869', '1178', '418', '1820', '1413', '488', '1304',
       '363', '2108', '521', '1659', '87', '1411', '1528', '3292', '7058',
       '1578', '627', '874', '1996', '1488', '5679', '1234', '5603',
       '400', '889', '3268', '875', '949', '2265', '441', '742', '425',
       '2476', '2971', '614', '1816', '1375', '1405', '2297', '1062',
       '1113', '420', '2469', '658', '1951', '2670', '2578', '1995',
       '1032', '994', '1011', '2421', '1296', '155', '494', '426', '1086',
       '961', '2236', '1829', '764', '1834', '1054', '617', '1529',
       '2266', '637', '626', '1832', '1016', '2002', '1756', '746',
       '1285', '2690', '1118', '5332', '980', '1807', '970', '1228',
       '1195', '1132', '1768', '1384', '1080', '7063', '1817', '1452',
       '1975', '1368', '702', '1974', '1781', '1036', '944', '663', '364',
       '1539', '1345', '1680', '2209', '741', '1575', '695', '1317',
       '294', '1525', '424', '997', '1473', '1552', '2819', '2188',
       '1668', '3057', '799', '1502', '2606', '552', '1694', '1759',
       '1110', '399', '1470', '1174', '5877', '1474', '1688', '526',
```

```
'686', '5908', '1107', '2070', '1468', '1246', '1685', '556',
'1533', '1917', '1346', '732', '692', '579', '421', '362', '3505',
'1855', '2711', '1586', '3739', '681', '1708', '2278', '1701',
'722', '1482', '928', '827', '832', '527', '604', '173', '1341',
'3329', '1553', '859', '167', '916', '828', '2082', '1176', '1108',
'975', '3008', '1516', '2269', '1699', '2073', '1031', '1503',
'2364', '1030', '1442', '5666', '2715', '1437', '2067', '1426',
'2908', '1279', '866', '4283', '279', '2658', '3015', '2004',
'1391', '4736', '748', '1466', '644', '683', '2705', '1297', '731',
'1252', '2216', '3141', '3273', '1518', '1723', '1588', '972',
'682', '1094', '668', '175', '967', '402', '3894', '1960', '1599',
'2000', '2084', '1621', '714', '1109', '3989', '873', '1572',
'1163', '1991', '1716', '1673', '2562', '2874', '965', '462',
'605', '1948', '1736', '3518', '2054', '2467', '1681', '1272',
'1205', '750', '2156', '2566', '115', '524', '3184', '676', '1678',
'612', '328', '955', '1441', '1675', '2909', '623', '822', '867',
'3025', '1993', '792', '636', '4057', '3743', '2337', '2570',
'2418', '2472', '3910', '1662', '2123', '2628', '3208', '2080',
'3699', '2913', '864', '2505', '870', '7536', '1924', '1671',
'1064', '1836', '1866', '4741', '841', '1369', '5681', '3112',
'1366', '2223', '1198', '1039', '3811', '3571', '1387', '1171',
'1365', '1531', '1590', '11706', '2308', '4860', '1641', '1045',
'1901'], dtype=object)
```

[10]:
```python
print(df.iloc[2].Levy)
```

```
-
```

As we can see, the Lev columns contains some **non numerical** value such as "-" values.

Thus, we will try to convert these non numerical values into numeric, and dealing with invalid data by changing them to NaN

[11]:
```python
# Finding and replacing the non-numeric values with NaN
df.Levy = pd.to_numeric(df.Levy, errors = 'coerce') # coerce will set the␣
↪inconvertible variable to NaN
```

[12]:
```python
# Printing column
df.Levy
```

[12]:
```
0        1399.0
1        1018.0
2           NaN
3         862.0
4         446.0
          …
19232       NaN
19233     831.0
19234     836.0
```

```
19235    1288.0
19236     753.0
Name: Levy, Length: 18902, dtype: float64
```

Moving to inspect the data inside "Engine Volume"

[13]:
```
df['Engine volume'].unique()
```

[13]:
```
array(['3.5', '3', '1.3', '2.5', '2', '1.8', '2.4', '4', '1.6', '3.3',
       '2.0 Turbo', '2.2 Turbo', '4.7', '1.5', '4.4', '3.0 Turbo',
       '1.4 Turbo', '3.6', '2.3', '1.5 Turbo', '1.6 Turbo', '2.2',
       '2.3 Turbo', '1.4', '5.5', '2.8 Turbo', '3.2', '3.8', '4.6', '1.2',
       '5', '1.7', '2.9', '0.5', '1.8 Turbo', '2.4 Turbo', '3.5 Turbo',
       '1.9', '2.7', '4.8', '5.3', '0.4', '2.8', '3.2 Turbo', '1.1',
       '2.1', '0.7', '5.4', '1.3 Turbo', '3.7', '1', '2.5 Turbo', '2.6',
       '1.9 Turbo', '4.4 Turbo', '4.7 Turbo', '0.8', '0.2 Turbo', '5.7',
       '4.8 Turbo', '4.6 Turbo', '6.7', '6.2', '1.2 Turbo', '3.4',
       '1.7 Turbo', '6.3 Turbo', '2.7 Turbo', '4.3', '4.2', '2.9 Turbo',
       '0', '4.0 Turbo', '20', '3.6 Turbo', '0.3', '3.7 Turbo', '5.9',
       '5.5 Turbo', '0.2', '2.1 Turbo', '5.6', '6', '0.7 Turbo',
       '0.6 Turbo', '6.8', '4.5', '0.6', '7.3', '0.1', '1.0 Turbo', '6.3',
       '4.5 Turbo', '0.8 Turbo', '4.2 Turbo', '3.1', '5.0 Turbo', '6.4',
       '3.9', '5.7 Turbo', '0.9', '0.4 Turbo', '5.4 Turbo', '0.3 Turbo',
       '5.2', '5.8', '1.1 Turbo'], dtype=object)
```

There is an **inconsistent** in the engine volume data as we observe multiple entries with "Turbo" appended after the volume.

We create an additional binary column called **"Turbo_option"** and mark it 0 for none Turbo and 1 for Turbo

[14]:
```
# Check if the data entries contain Turbo -> Change the value in Turbo_option␣
↪accordingly
df['Turbo_option'] = df['Engine volume'].str.contains('Turbo').astype(int)
# Replace Turbo inside data entries
df['Engine volume'] = df['Engine volume'].str.replace(' Turbo', '').
↪astype(float)
```

[15]:
```
# Checking the Engine_Volume and Turbo_option
df.loc[df['Turbo_option']==1]
```

[15]:

	ID	Price	Levy	Manufacturer	Model	Prod. year	\
23	45814106	7840	NaN	FORD	Transit	2001	
25	45782859	20385	NaN	MERCEDES-BENZ	E 220	2006	
30	44944581	15681	1288.0	MERCEDES-BENZ	Vito	2007	
34	45542380	24462	NaN	JEEP	Grand Cherokee	2007	
42	45667253	20165	650.0	VOLKSWAGEN	Jetta	2016	
…	…	…	…	…	…	…	
19167	45799423	18817	1995.0	FORD	Transit	2003	

```
19170  45776725  10976     NaN  MERCEDES-BENZ          C 220        2001
19190  45790255  24462   642.0            BMW            528        2012
19225  45794580   8781  1107.0           OPEL          Combo        2007
19232  45798355   8467     NaN  MERCEDES-BENZ        CLK 200        1999

         Category Leather interior Fuel type  Engine volume       Mileage  \
23       Microbus               No    Diesel            2.0     230000 km
25          Sedan              Yes    Diesel            2.2     210000 km
30    Goods wagon               No    Diesel            2.0     180000 km
34           Jeep              Yes    Diesel            3.0     250000 km
42          Sedan              Yes    Petrol            1.4      11200 km
…             …                …       …            …             …
19167    Microbus               No    Diesel            2.4  2147483647 km
19170       Sedan               No    Diesel            2.2     320000 km
19190       Sedan              Yes    Petrol            2.0      96966 km
19225 Goods wagon               No    Diesel            1.7     236000 km
19232       Coupe              Yes       CNG            2.0     300000 km

      Cylinders Gear box type Drive wheels   Doors       Wheel   Color  \
23          4.0        Manual        Front  02-Mar  Left wheel   White
25          4.0     Tiptronic         Rear  04-May  Left wheel   Black
30          6.0        Manual         Rear  04-May  Left wheel   White
34          6.0     Tiptronic          4x4  04-May  Left wheel   Black
42          4.0     Tiptronic        Front  04-May  Left wheel   Black
…          …           …          …      …       …       …
19167       4.0        Manual        Front  02-Mar  Left wheel   White
19170       5.0     Automatic         Rear  04-May  Left wheel  Silver
19190       4.0     Tiptronic         Rear  04-May  Left wheel   Black
19225       4.0        Manual        Front  04-May  Left wheel   Beige
19232       4.0        Manual         Rear  02-Mar  Left wheel  Silver

      Airbags  Turbo_option
23          0             1
25          8             1
30          4             1
34         10             1
42          8             1
…        …           …
19167       2             1
19170       4             1
19190      12             1
19225       4             1
19232       5             1

[1890 rows x 19 columns]
```

[16]:
```
df['Engine volume']
```

```
[16]: 0        3.5
      1        3.0
      2        1.3
      3        2.5
      4        1.3
              …
      19232    2.0
      19233    2.4
      19234    2.0
      19235    2.0
      19236    2.4
      Name: Engine volume, Length: 18902, dtype: float64
```

Inspecting **"Mileage"** column data entries

```
[17]: df['Mileage']
```

```
[17]: 0        186005 km
      1        192000 km
      2        200000 km
      3        168966 km
      4         91901 km
                  …
      19232    300000 km
      19233    161600 km
      19234    116365 km
      19235     51258 km
      19236    186923 km
      Name: Mileage, Length: 18902, dtype: object
```

> **"Mileage"** column contains additional meansurement - standard unit make it "Object" datatype

Thus, we eliminate **"km"** from the data entries before converting it into **"int"** type

```
[18]: df['Mileage'] = df['Mileage'].str.replace(' km', '').astype(int)
```

```
[19]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 18902 entries, 0 to 19236
Data columns (total 19 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   ID                18902 non-null  int64
 1   Price             18902 non-null  int64
 2   Levy              13214 non-null  float64
 3   Manufacturer      18902 non-null  object
 4   Model             18902 non-null  object
 5   Prod. year        18902 non-null  int64
```

```
 6   Category          18902 non-null  object
 7   Leather interior  18902 non-null  object
 8   Fuel type         18902 non-null  object
 9   Engine volume     18902 non-null  float64
 10  Mileage           18902 non-null  int32
 11  Cylinders         18902 non-null  float64
 12  Gear box type     18902 non-null  object
 13  Drive wheels      18902 non-null  object
 14  Doors             18902 non-null  object
 15  Wheel             18902 non-null  object
 16  Color             18902 non-null  object
 17  Airbags           18902 non-null  int64
 18  Turbo_option      18902 non-null  int32
dtypes: float64(3), int32(2), int64(4), object(10)
memory usage: 2.7+ MB
```

Let inspect the list again! As we are using numerical subset for plotting and data visualization, **"ID"** column will not be needed as well as **"Doors"**, thus we will **not consider** this column.

[20]:
```
df = df.drop(['ID','Doors'],axis=1)
#List of Categorical variables
categorical = [i for i in df.columns if df[i].dtypes == 'O']
#List of numerical variables
numerical = [i for i in df.columns if i not in categorical ]

print('Categorical: ',categorical)
print('Numerical: ',numerical)
```

```
Categorical:  ['Manufacturer', 'Model', 'Category', 'Leather interior', 'Fuel
type', 'Gear box type', 'Drive wheels', 'Wheel', 'Color']
Numerical:  ['Price', 'Levy', 'Prod. year', 'Engine volume', 'Mileage',
'Cylinders', 'Airbags', 'Turbo_option']
```

Checking Null Values *Categorical Values*

[21]:
```
df[categorical].isnull().sum()
```

[21]:
```
Manufacturer        0
Model               0
Category            0
Leather interior    0
Fuel type           0
Gear box type       0
Drive wheels        0
Wheel               0
Color               0
dtype: int64
```

No null or missing values in categorical subset, let move to numerical

Numerical Values

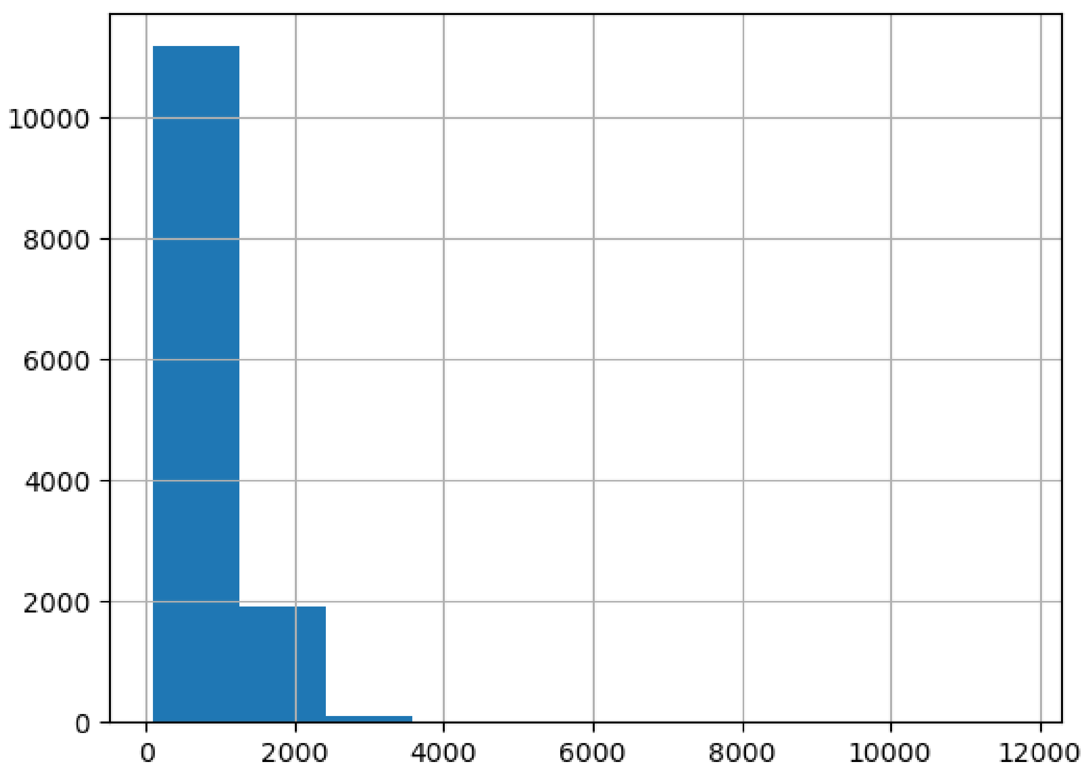
[22]:
```
df[numerical].isnull().sum()
```

[22]:
```
Price                 0
Levy               5688
Prod. year            0
Engine volume         0
Mileage               0
Cylinders             0
Airbags               0
Turbo_option          0
dtype: int64
```

From observation, all null values come from "Levy" which is understandable as we just changing **invalid** data entries of this column - "-" to NaN.

Let plot **"Levy"** column in the format of Histogram as it help visuallize the distribution of the data entries.

[23]:
```
df.Levy.hist()
plt.show()
```



We can see that the distribution of "Levy" in the dataset is highly right skewed

As **“Levy”** belongs to numerical and the histogram plot is skewed, it’s more efficient for us to use **median** values to fill out the missing instead of the **mean** (as it’s vulnerable by possible outliers).

[24]:
```
# Fill missing values
df.Levy = df['Levy'].fillna(df['Levy'].median())
```

[25]:
```
df[numerical].isnull().sum()
```

[25]:
```
Price            0
Levy             0
Prod. year       0
Engine volume    0
Mileage          0
Cylinders        0
Airbags          0
Turbo_option     0
dtype: int64
```

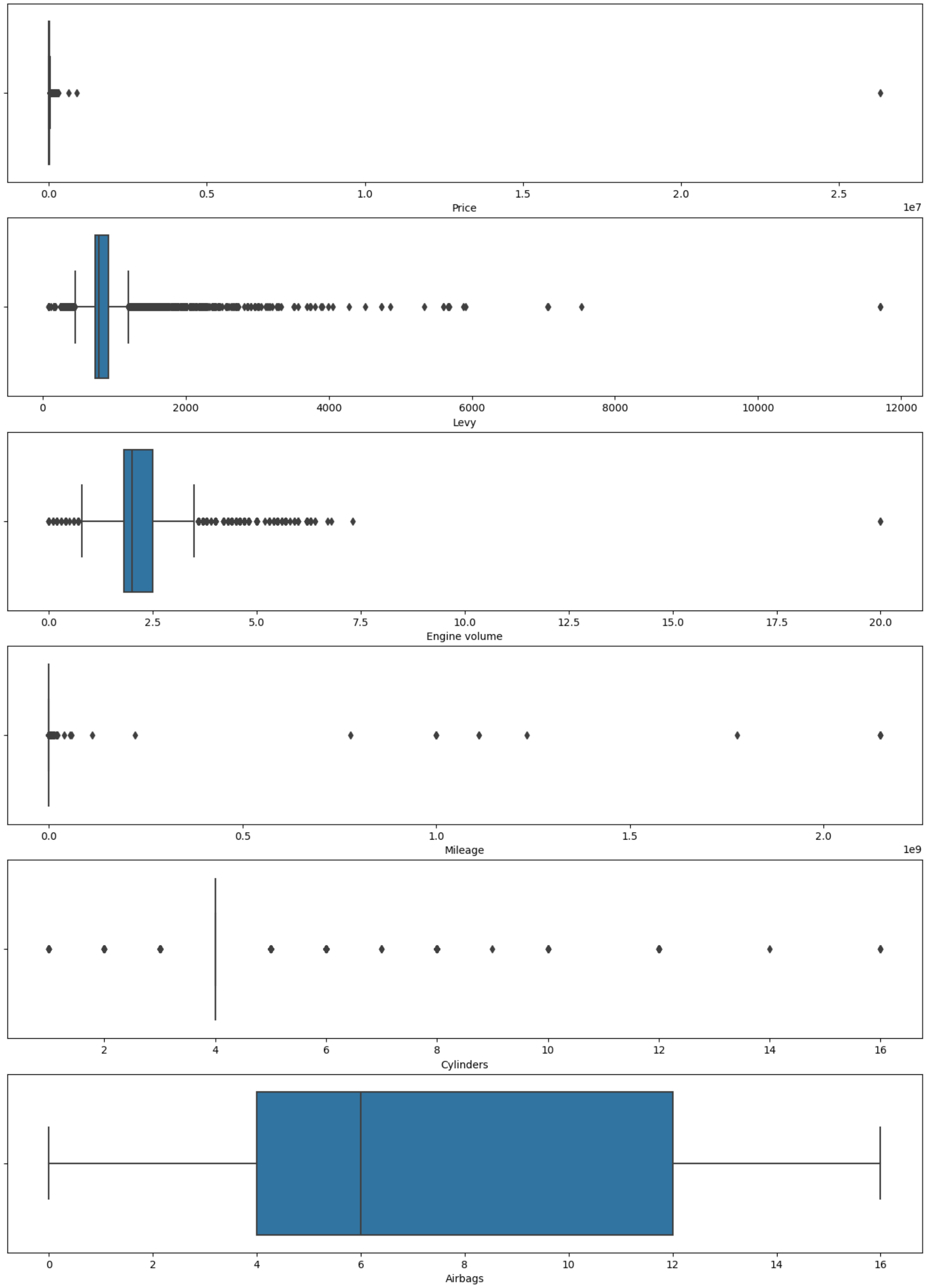
We’ve successfully filled in missing value.

Checking for Outliers

Starting by checking outliers in numerical data subset through ploting them in the boxplot format. As **Production’s year** and **Turbo_option** contributes less to our analysis, we will leave them aside for now

[26]:
```
# Creating numerical subset
numerical_sub = numerical

# Removing "Turbo_option" and "Production year"
numerical_sub.remove("Turbo_option")
numerical_sub.remove("Prod. year")
```

[27]:
```
# Start ploting the data
fig, axes = plt.subplots(len(numerical_sub), 1, figsize=(16,22))
for i,col in enumerate(numerical_sub):
    sns.boxplot(x=df[numerical_sub[i]],ax=axes[i],)
```



The boxplot reveals a great number of outliers in **Levy**, **Engine Volumne**, **Mileage**.

However, these outliers may not be of significant concern for several reasons. Firstly, in the case of **Levy**, it might be due to certain anomalies in the tax regulations such as exceptionally high tax for special vehicle types. Secondly, outliers in **Engine Volume** could stem from high-performance vehicles or outliers in the dataset such as rare vintage cars with unusually large engines. Similarly, outliers in **Mileage** could be attributed to certain vehicles being used for commercial purposes or experiencing extreme driving conditions. Thus, while these outliers deviate from the norm, they may not necessarily indicate errors or anomalies in the data but rather reflect legitimate variations within the dataset

1.1.3 II. Data Analysis & Visualization

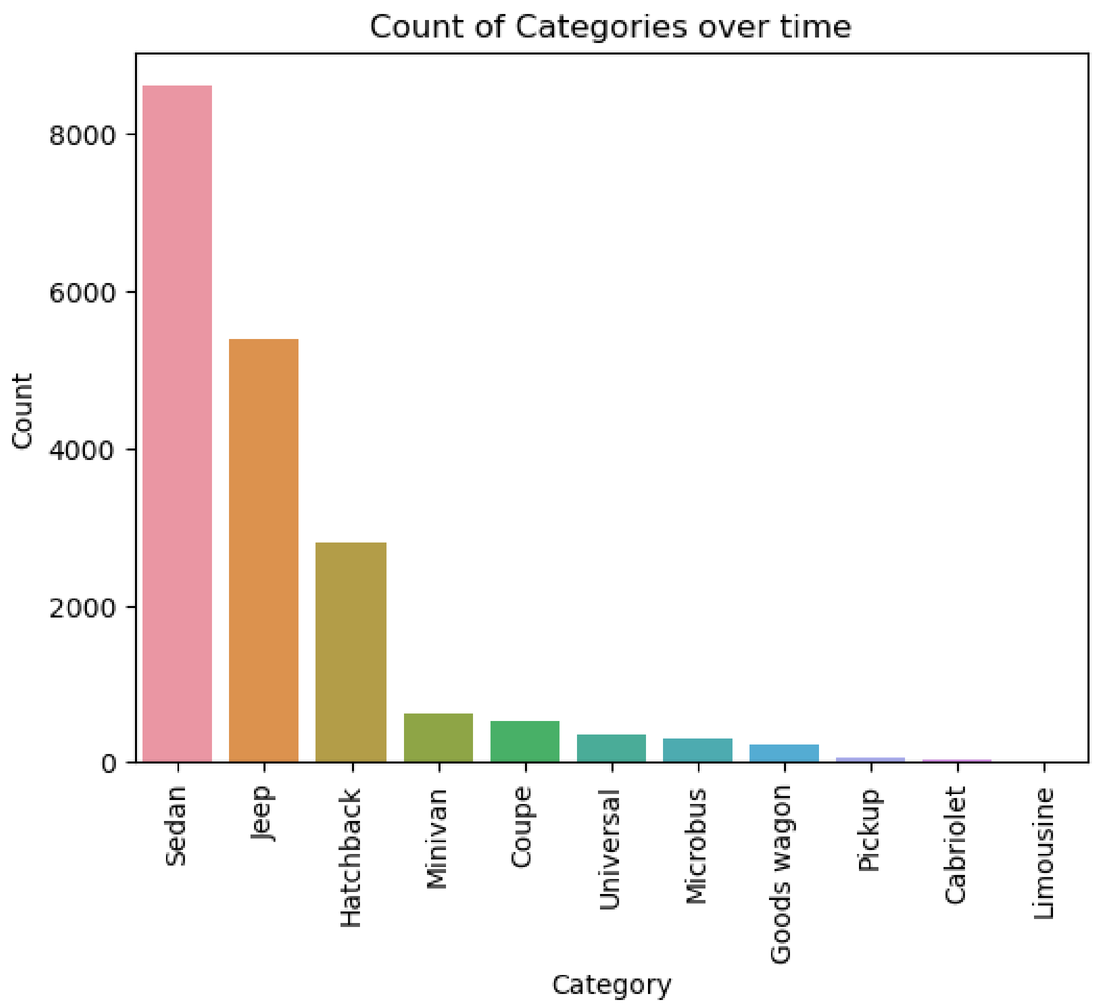
Data Distribution In this part, we will analyze the dataset by exploring the relationship between variables as well as the factors affecting the target variable - **price**

Car categories distribution First, let look at the distribution for car catergory

[28]:
```
cat_count = df['Category'].value_counts().reset_index()
cat_count.columns = ['Category', 'Count']
print(cat_count)
```

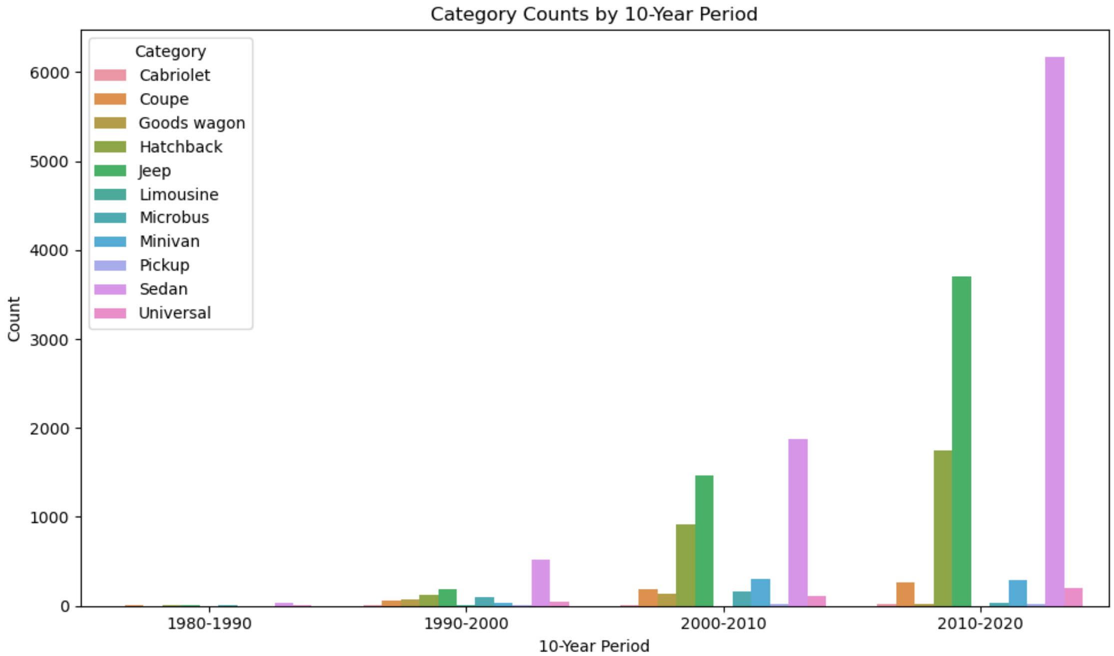
```
       Category  Count
0         Sedan   8594
1          Jeep   5373
2     Hatchback   2797
3       Minivan    633
4         Coupe    526
5     Universal    361
6      Microbus    299
7   Goods wagon    228
8        Pickup     51
9     Cabriolet     34
10    Limousine      6
```

[29]:
```
sns.barplot(x='Category',y='Count', data=cat_count)
plt.xlabel('Category')
plt.ylabel('Count')
plt.title('Count of Categories over time')
plt.xticks(rotation=90)
plt.show()
```



[30]:
```
df['10_Year_Period'] = pd.cut(df['Prod. year'], bins=[1980, 1990, 2000, 2010,␣
↪2020], labels=['1980-1990', '1990-2000', '2000-2010', '2010-2020'])
# Group the data by '10_Year_Period'
grouped_data = df.groupby(['10_Year_Period', 'Category']).size().
↪reset_index(name='Count')
```

[31]:
```
plt.figure(figsize=(10, 6))
sns.barplot(x='10_Year_Period', y='Count', hue='Category', data=grouped_data)
plt.xlabel('10-Year Period')
plt.ylabel('Count')
plt.title('Category Counts by 10-Year Period')
plt.legend(title='Category', loc='upper left')
plt.tight_layout()
plt.show()
```



Sedan appears to be the most popular car category in the dataset, following by **Jeep** and **Hatchback**.

Manufacturer data analysis

[32]:
```
m_count = df['Manufacturer'].value_counts()
m_count.columns = ['Manufacturer', 'Count']
print(m_count)
```

```
Manufacturer
HYUNDAI          3729
TOYOTA           3606
MERCEDES-BENZ    2041
FORD             1086
CHEVROLET        1046
                 …
MOSKVICH            1
PONTIAC             1
SATURN              1
ASTON MARTIN        1
GREATWALL           1
Name: count, Length: 64, dtype: int64
```

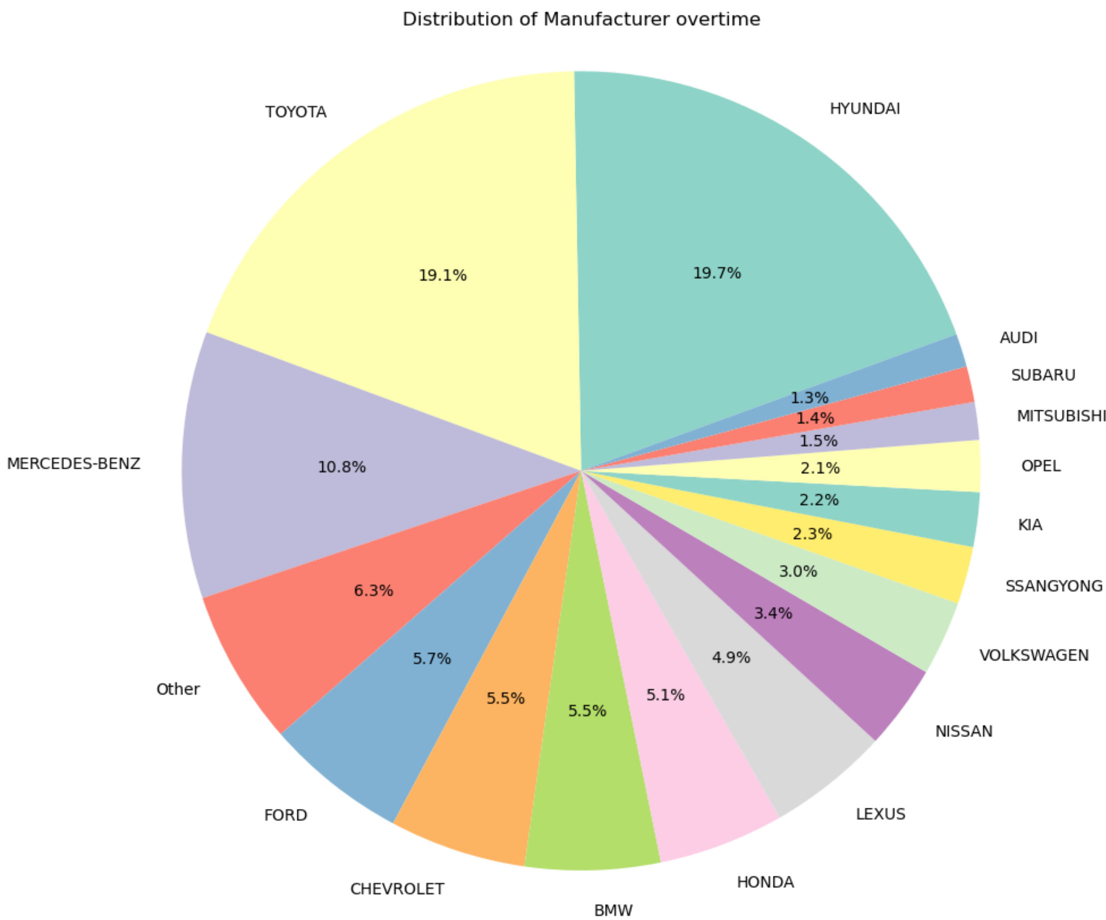
As the dataset contain multiple entry with insignificant count such as Tesla with 1 count, we will try to convert them to a larger subset called Other before plotting the **pie chart**

[33]:
```
# Placing any manufacturer with less than 2% to 'Other' group
ptg = m_count / m_count.sum()
threshold = 0.01
other_manufacturer = ptg[ptg < threshold].index
# Grouping small data manufacturer
df['Manufacturer'] = df['Manufacturer'].apply(lambda x: 'Other' if x in␣
↪other_manufacturer else x)
```

[34]:
```
m_count = df['Manufacturer'].value_counts()
m_count.columns = ['Manufacturer', 'Count']
print(m_count)
```

```
Manufacturer
HYUNDAI          3729
TOYOTA           3606
MERCEDES-BENZ    2041
Other            1189
FORD             1086
CHEVROLET        1046
BMW              1035
HONDA             960
LEXUS             927
NISSAN            645
VOLKSWAGEN        571
SSANGYONG         439
KIA               417
OPEL              395
MITSUBISHI        288
SUBARU            274
AUDI              254
Name: count, dtype: int64
```

[35]:
```
plt.figure(figsize=(10, 10))
plt.pie(m_count, labels=m_count.index, autopct='%1.1f%%',␣
↪startangle=20,colors=plt.cm.Set3.colors)
plt.title('Distribution of Manufacturer overtime')
plt.axis('equal')
plt.show()
```



Hyundai car is the most popular car in the dataset with **19.7%**, following by **Toyota** with **19.1%**. Beside that, less popular manufacturer add up to **10.7%** of the dataset

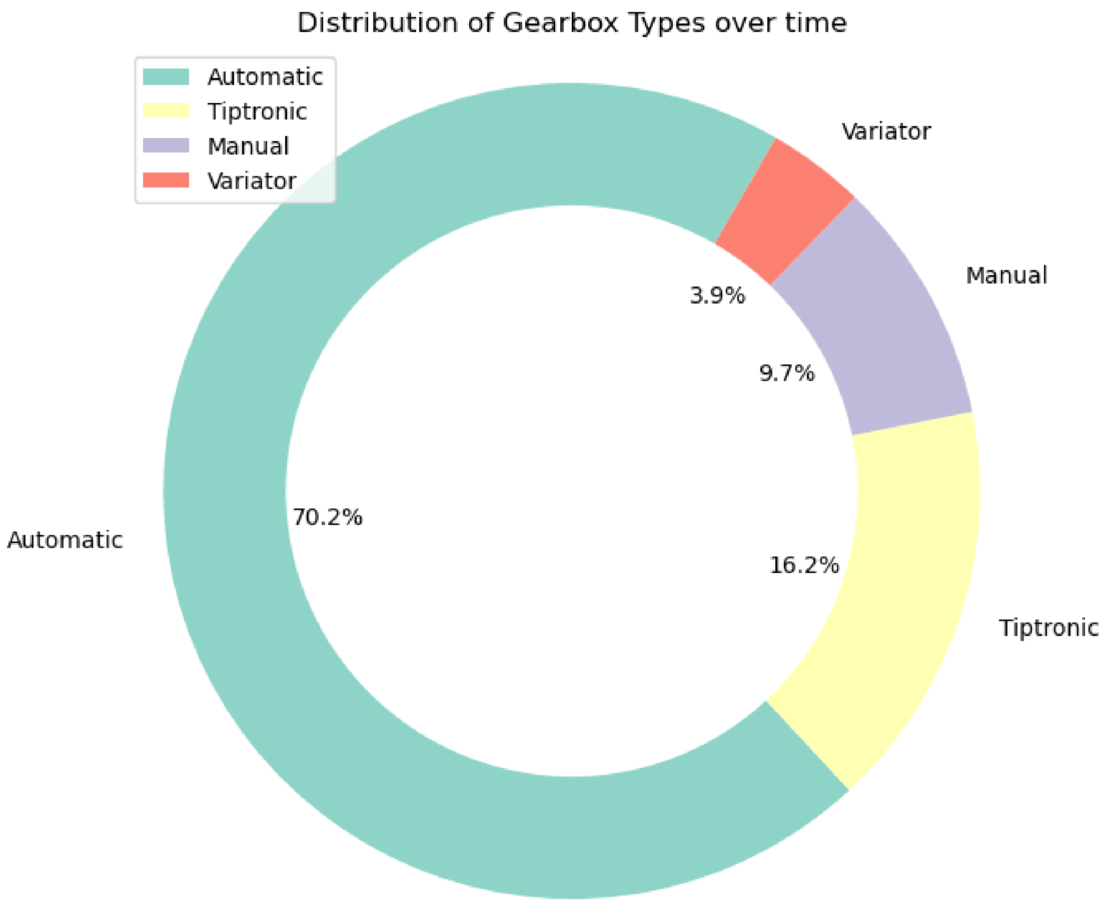
Distribution of Gear Box types

[36]:
```
g_count=df['Gear box type'].value_counts()
g_count.columns = ['Gear box type', 'Count']
print(g_count)
```

```
Gear box type
Automatic    13276
Tiptronic     3065
Manual        1828
Variator       733
Name: count, dtype: int64
```

[37]:
```
plt.figure(figsize=(7, 7))
plt.pie(g_count, labels=g_count.index, autopct='%1.1f%%', startangle=60,
  ↪colors=plt.cm.Set3.colors)
plt.axis('equal')
```

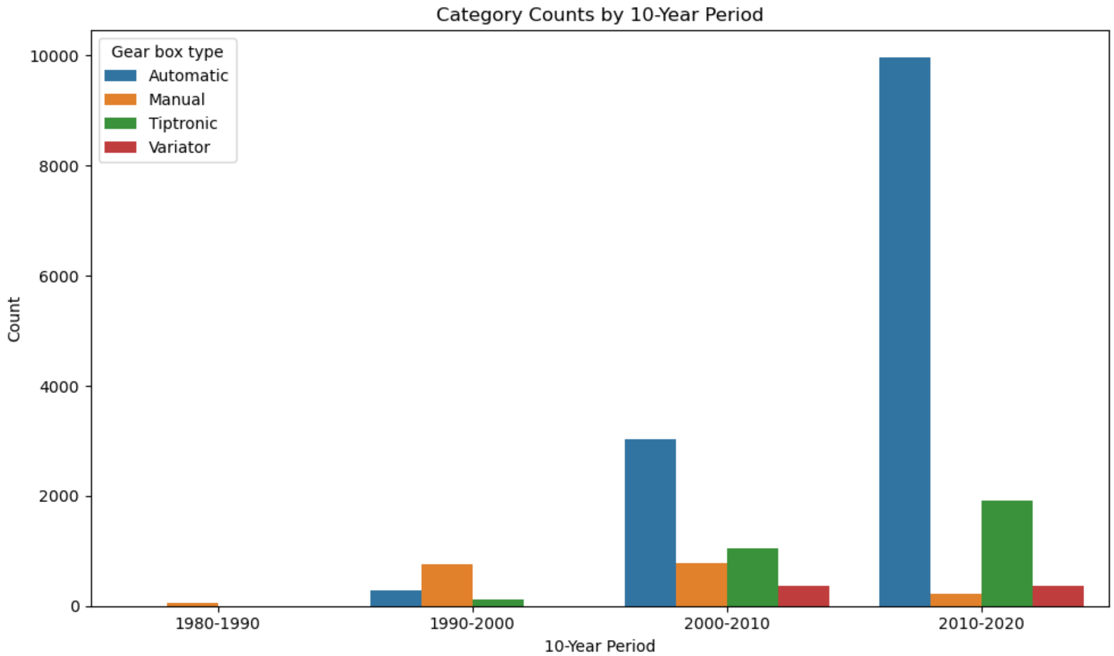
```
# Adding a circle at the center to make it look like a donut chart (optional)
centre_circle = plt.Circle((0, 0), 0.70, fc='white')
fig = plt.gcf()
fig.gca().add_artist(centre_circle)

plt.legend()
plt.title('Distribution of Gearbox Types over time')
plt.show()
```



Automatic gearbox appears to be the most popular gearbox with **70.2%** of car using. **Tiptronic** ranked second with **16.2%** popularity

[38]:
```
df['10_Year_Period'] = pd.cut(df['Prod. year'], bins=[1980, 1990, 2000, 2010,␣
↪2020], labels=['1980-1990', '1990-2000', '2000-2010', '2010-2020'])
# Group the data by '10_Year_Period'
grouped_data1 = df.groupby(['10_Year_Period', 'Gear box type']).size().
↪reset_index(name='Count')
```

[39]:
```
plt.figure(figsize=(10, 6))
sns.barplot(x='10_Year_Period', y='Count', hue='Gear box type',␣
↪data=grouped_data1)
plt.xlabel('10-Year Period')
plt.ylabel('Count')
plt.title('Category Counts by 10-Year Period')
plt.legend(title='Gear box type', loc='upper left')
plt.tight_layout()
plt.show()
```



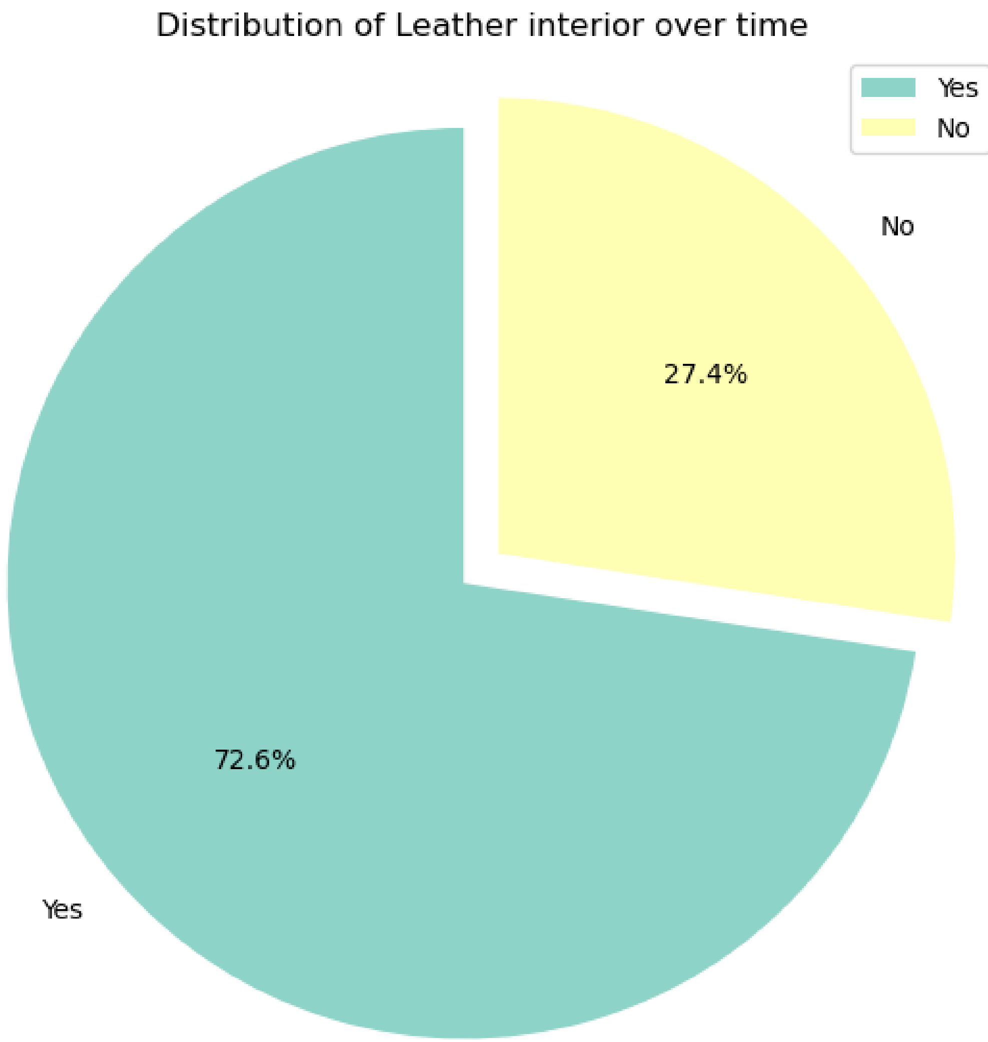
Since **2000**, the market witness the **dominance** of **Automatic** gearbox, as a **replacement** of the popular **Manual**. This trend is further **elevated** in the period of **2010-2020**. In addition, **Tiptronic** and **Variator** steadily increases over the year making them the **second** and **third** popular gearbox.

Distribution of Leather interior

[40]:
```
l_count=df['Leather interior'].value_counts()
l_count.columns = ['Leather interior', 'Count']
print(l_count)
```

```
Leather interior
Yes    13723
No      5179
Name: count, dtype: int64
```

[41]:
```
plt.figure(figsize=(7, 7))
explode = (0.1, 0)
plt.pie(l_count, labels=l_count.index, autopct='%1.1f%%', startangle=90,␣
 ↪colors=plt.cm.Set3.colors, explode=explode)
plt.axis('equal')
plt.title('Distribution of Leather interior over time')
plt.legend()
plt.show()
```

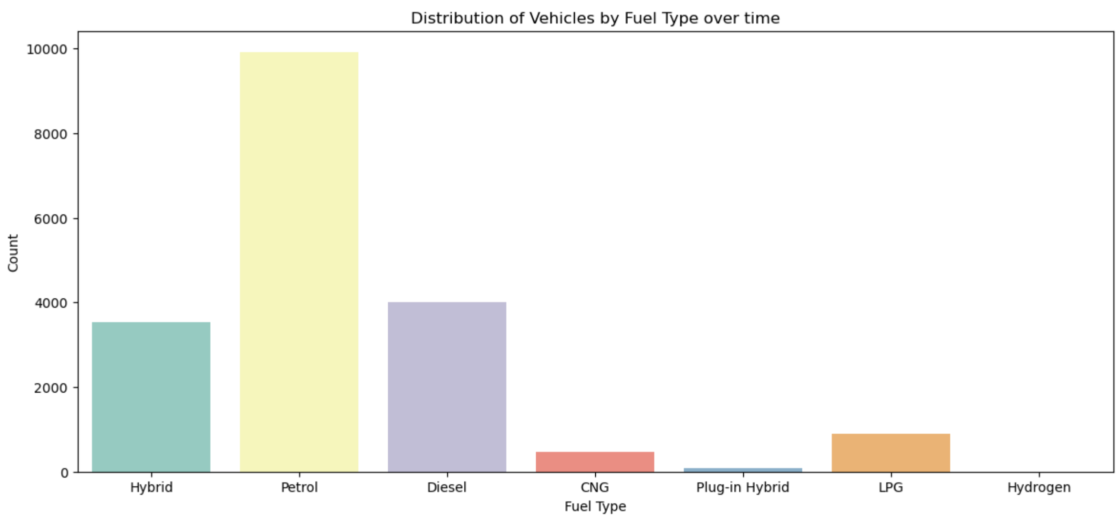


Distribution of fuel types

[42]:
```
f_count=df['Fuel type'].value_counts()
f_count.columns = ['Fuel type', 'Count']
print(f_count)
```

```
Fuel type
Petrol            9924
Diesel            4001
Hybrid            3539
LPG                885
CNG                467
Plug-in Hybrid      85
Hydrogen             1
Name: count, dtype: int64
```

[43]:
```
plt.figure(figsize=(14, 6))
ax = sns.countplot(x='Fuel type', data=df, palette='Set3')

plt.ylabel('Count')
plt.xlabel('Fuel Type')
plt.title('Distribution of Vehicles by Fuel Type over time')

plt.show()
```

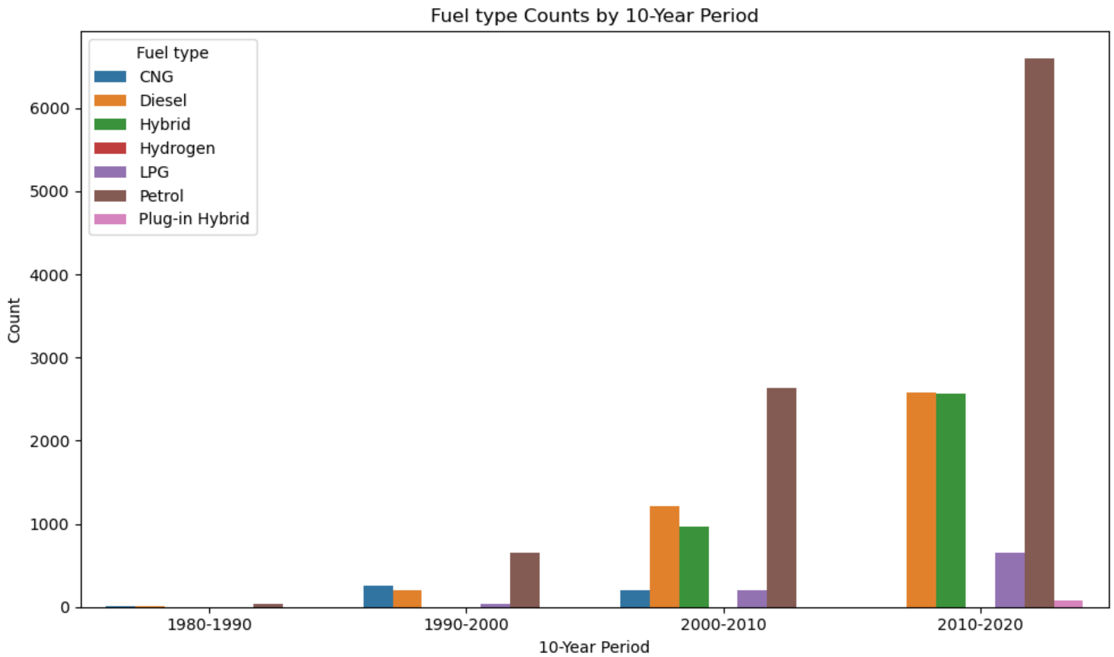


For the period of **1980-2020**, **Petrol** is the most popular **Fuel Type** with nearly 10000 cars using this type of fuel, following by **Diesel** and young **Hybrid**.

[44]:
```
df['10_Year_Period'] = pd.cut(df['Prod. year'], bins=[1980, 1990, 2000, 2010,␣
↪2020], labels=['1980-1990', '1990-2000', '2000-2010', '2010-2020'])
# Group the data by '10_Year_Period'
grouped_data1 = df.groupby(['10_Year_Period', 'Fuel type']).size().
↪reset_index(name='Count')
```

[45]:
```
plt.figure(figsize=(10, 6))
sns.barplot(x='10_Year_Period', y='Count', hue='Fuel type', data=grouped_data1)
```

```
plt.xlabel('10-Year Period')
plt.ylabel('Count')
plt.title('Fuel type Counts by 10-Year Period')
plt.legend(title='Fuel type', loc='upper left')
plt.tight_layout()
plt.show()
```

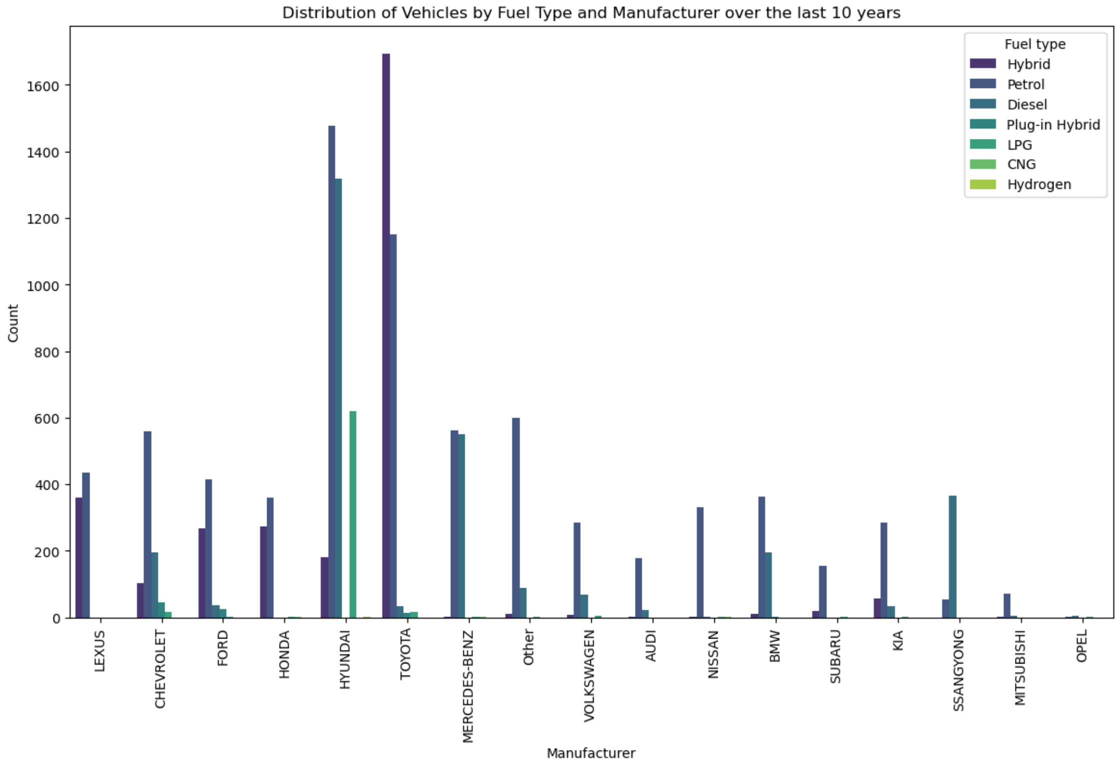


Since **1990** the world witness the **domination** of **petrol** in fuel type option. Despite being introduced earlier in **90s**, **Hybrid** take up to a decade to be widely produced. Despite this, the cars with this fuel type rapidly become popular within the market, especially with the introduce of its **upgraded version** - **Plug-in Hybrid** in 2008.

Let see within the **well-known manufacturer** which Fuel Type is the most popular!

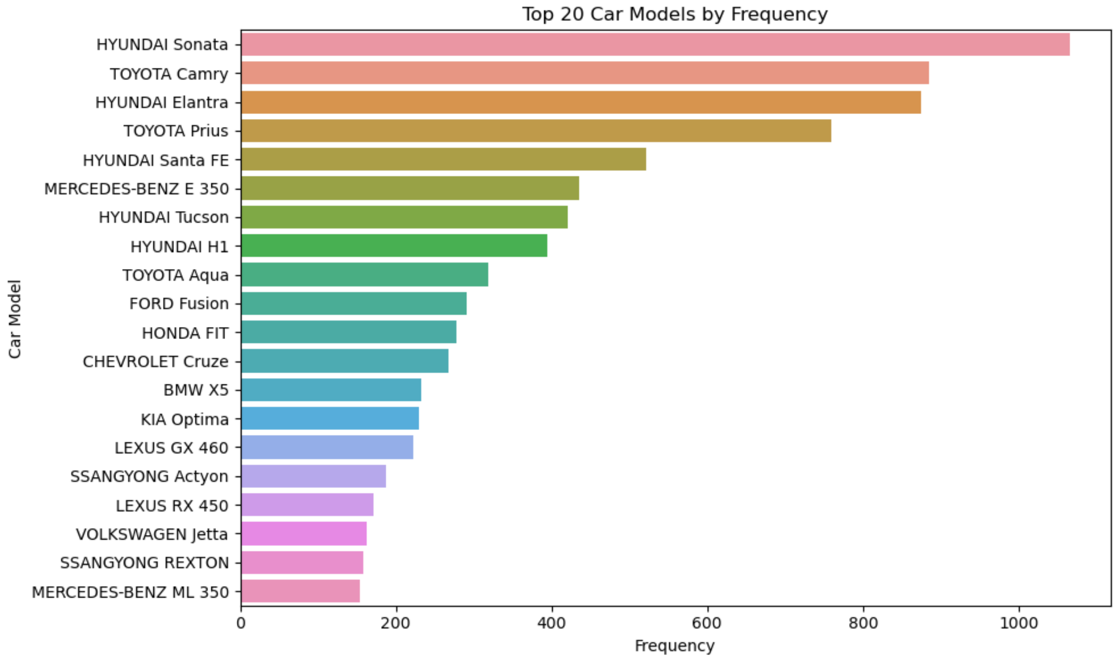
[46]:
```
# Obtaining car with production year bigger than 2010
dfy = df[df['Prod. year']>=2010]
#Getting top 10 manufacturer
top_10 = dfy['Manufacturer'].value_counts().head(10).index
plt.figure(figsize=(14, 8))
sns.countplot(x='Manufacturer', hue='Fuel type', data=dfy, palette='viridis')
plt.ylabel('Count')
plt.xlabel('Manufacturer')
plt.title('Distribution of Vehicles by Fuel Type and Manufacturer over the last␣
↪10 years')
plt.legend(title='Fuel type')
plt.xticks(rotation=90)  # Rotate x-axis labels for better readability
```

```
plt.show()
```



Toyota stands out for prioritising Hybrid fuel types. Conversely, **Hyundai excels** in the **Hydrogen** fuel sector, although it ranks third in its primary fuel preference. The **Other** category contains brands that refer the popular **Petrol**, a trend mirrored by companies like **Mercedes-Benz** and **BMW**.

[47]:
```
n = 20  # Number of top car models to plot
dfy['Manufacturer_Model'] = dfy['Manufacturer'] + ' ' + dfy['Model']
top_car_models = dfy['Manufacturer_Model'].value_counts().head(n)

plt.figure(figsize=(10, 6))
sns.barplot(x=top_car_models.values, y=top_car_models.index)
plt.title(f'Top {n} Car Models by Frequency')
plt.xlabel('Frequency')
plt.ylabel('Car Model')
plt.tight_layout()
plt.show()
```



Data Analysis

Getting average price by year In this segment, we will dive deeper into our target variable - **Price**, examining its correlations and relationship both on and by other variables. Starting by getting average price

[48]:
```
#Grouping price and production year
avg_price = df.groupby('Prod. year')['Price'].mean().reset_index()
```

For this, we will only consider cars that are produced in at least the year of **2000**

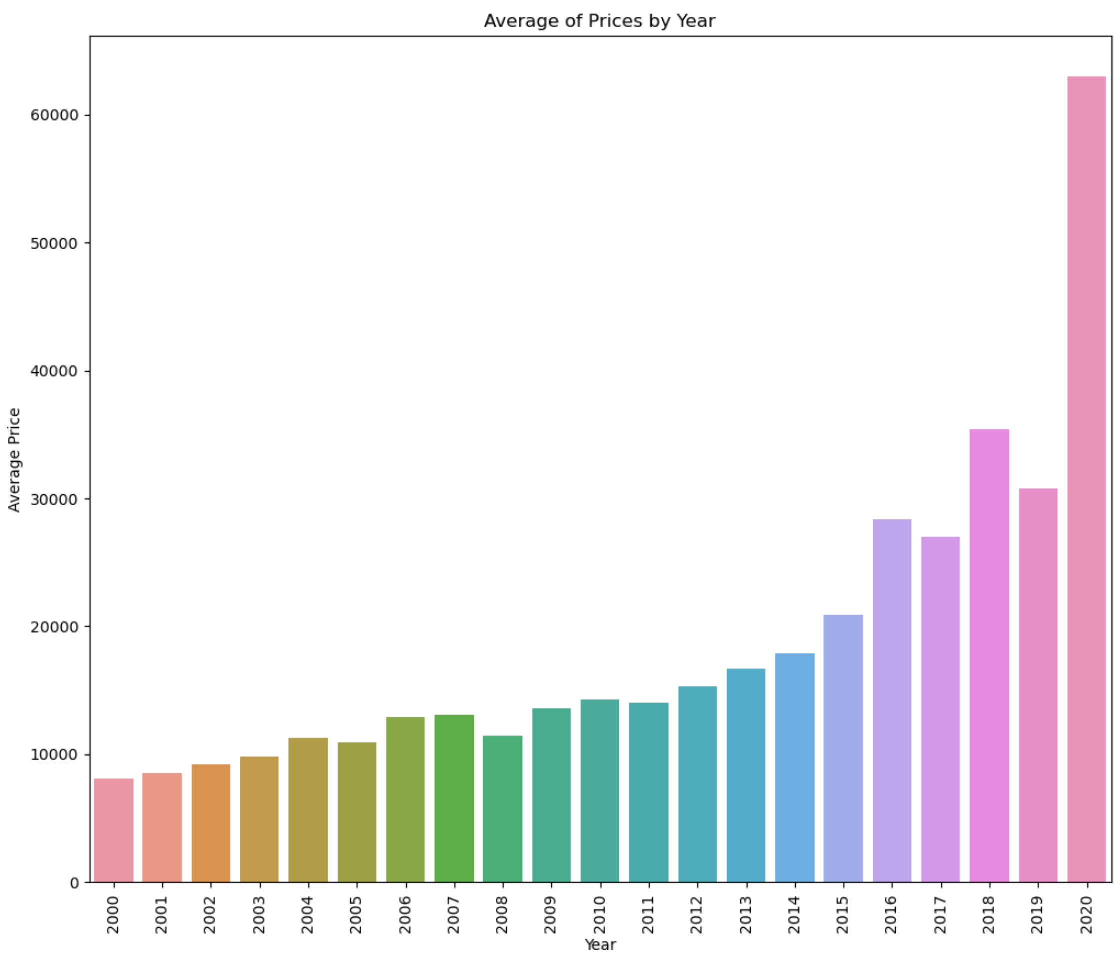
[49]:
```
avg_price.tail(21)
```

[49]:

	Prod. year	Price
20	2000	8095.519856
21	2001	8526.952569
22	2002	9243.901754
23	2003	9773.946927
24	2004	11249.971751
25	2005	10904.174242
26	2006	12926.117460
27	2007	13058.215217
28	2008	11472.160055
29	2009	13627.605042
30	2010	14289.215995
31	2011	14039.120101

```
32          2012  15352.572970
33          2013  16714.076320
34          2014  17914.195694
35          2015  20918.979699
36          2016  28336.581781
37          2017  26999.066950
38          2018  35386.830957
39          2019  30746.187500
40          2020  63006.106383
```

[50]:
```
plt.figure(figsize=(12, 10))
sns.barplot(x='Prod. year',y='Price', data=avg_price.tail(21))
plt.title('Average of Prices by Year')
plt.xlabel('Year')
plt.ylabel('Average Price')
plt.xticks(rotation=90)
plt.show()
```

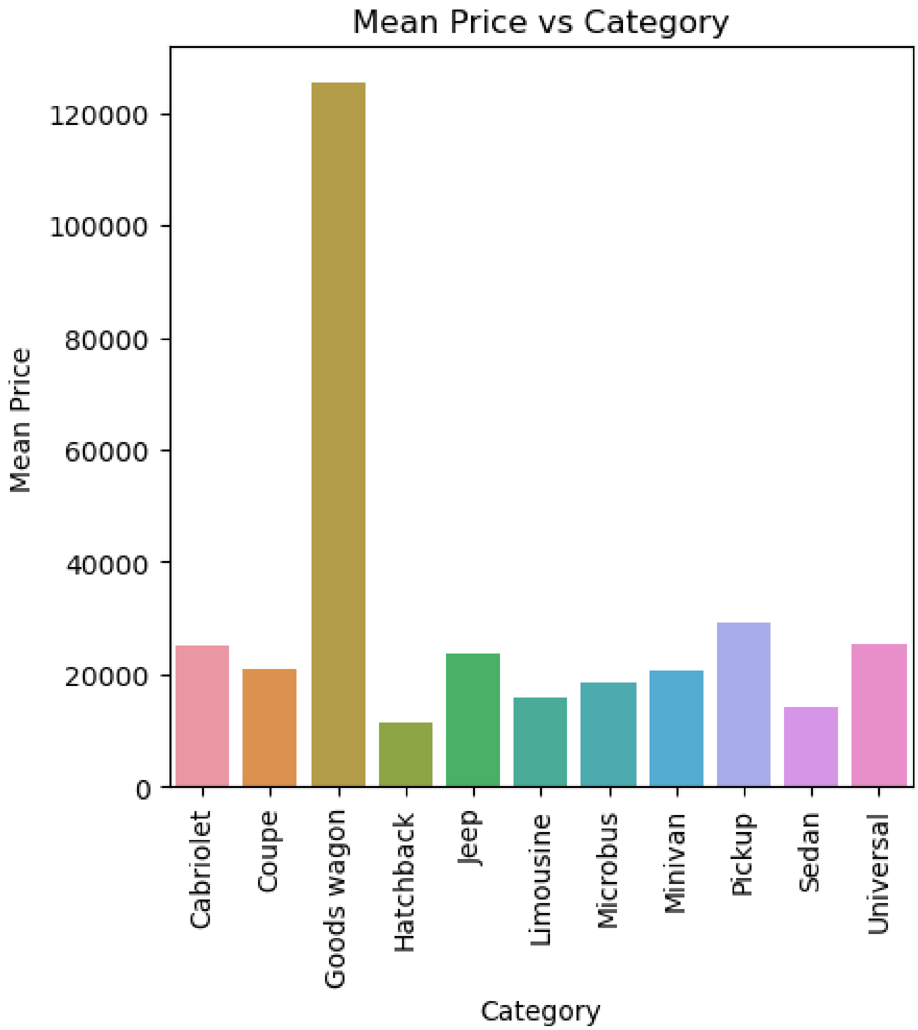


From the bar plot, overall car price increases over the time and reach its **peak** in **2020**. This could be because of the impact from **COVID-19** in **2019**.

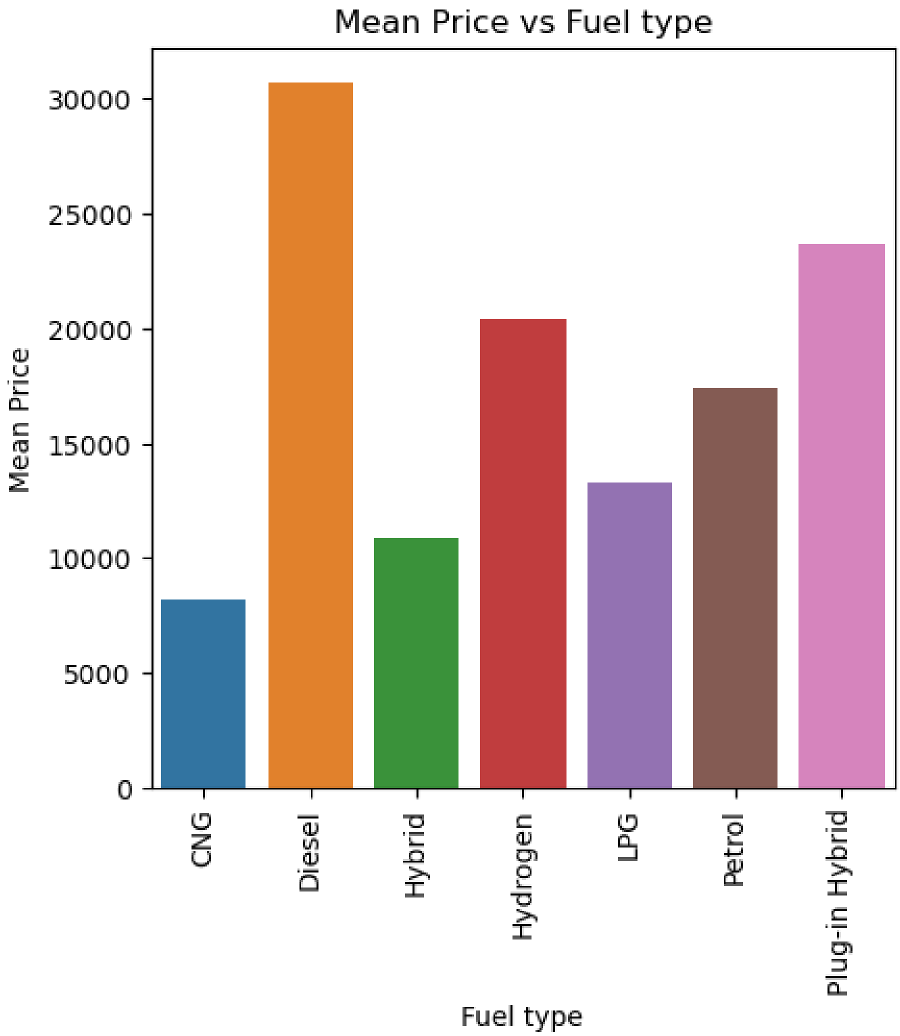
Next, let inspect the **relationship** between other variables versus average car prices

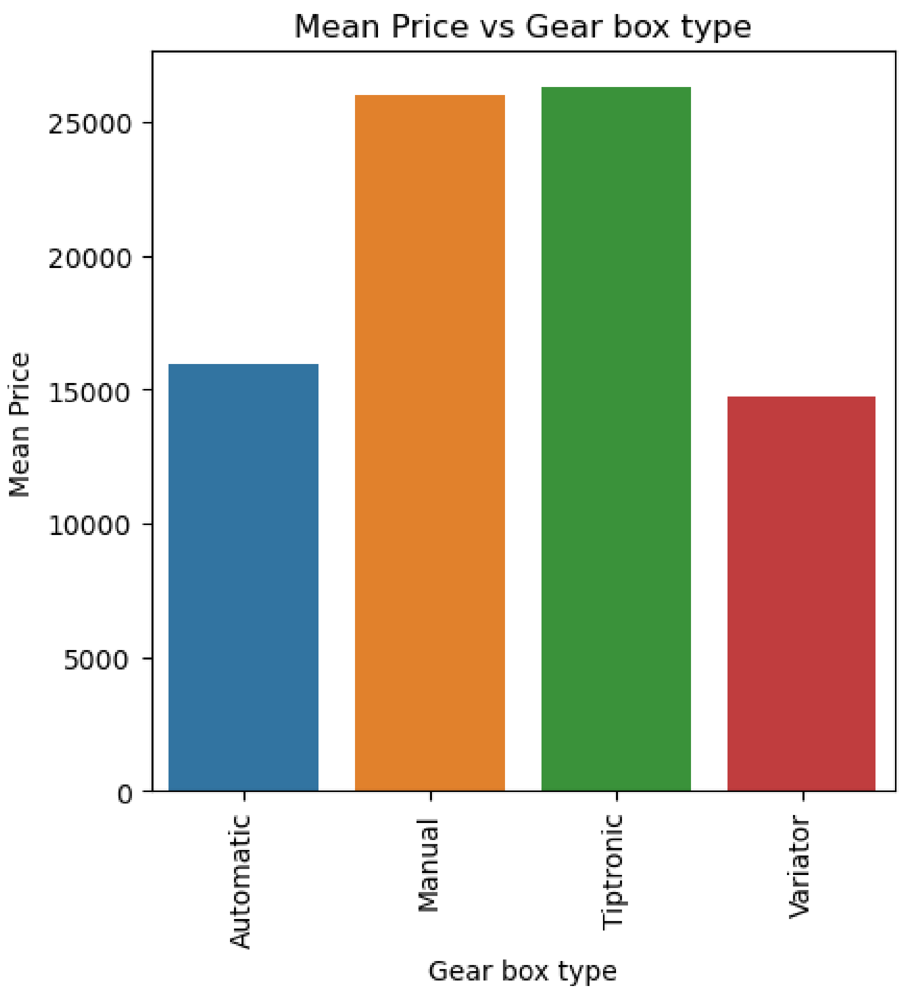
Plot between average car price and other variable

[51]:
```
# Noted: In this bar plot we will not checking manufacturer and model for now␣
↪as it's not optimal for barplot to demonstrate them
checking_variable = [ 'Category', 'Leather interior', 'Fuel type', 'Gear box␣
↪type', 'Drive wheels', 'Wheel', 'Color']
df1 = [col for col in dfy[categorical].columns if col in checking_variable]

for column in df1:
    plt.figure(figsize=(5,5))
    # Grouping by category and calculating mean price for each variable
    mean_values = df.groupby(column)['Price'].mean().reset_index()
    # Plotting
    sns.barplot(x=column, y='Price', data=mean_values)
    plt.xlabel(column)  # Set x-axis label
    plt.ylabel('Mean Price')          # Set y-axis label
    plt.title(f'Mean Price vs {column}')  # Set plot title
    plt.xticks(rotation=90)
    plt.show()
```

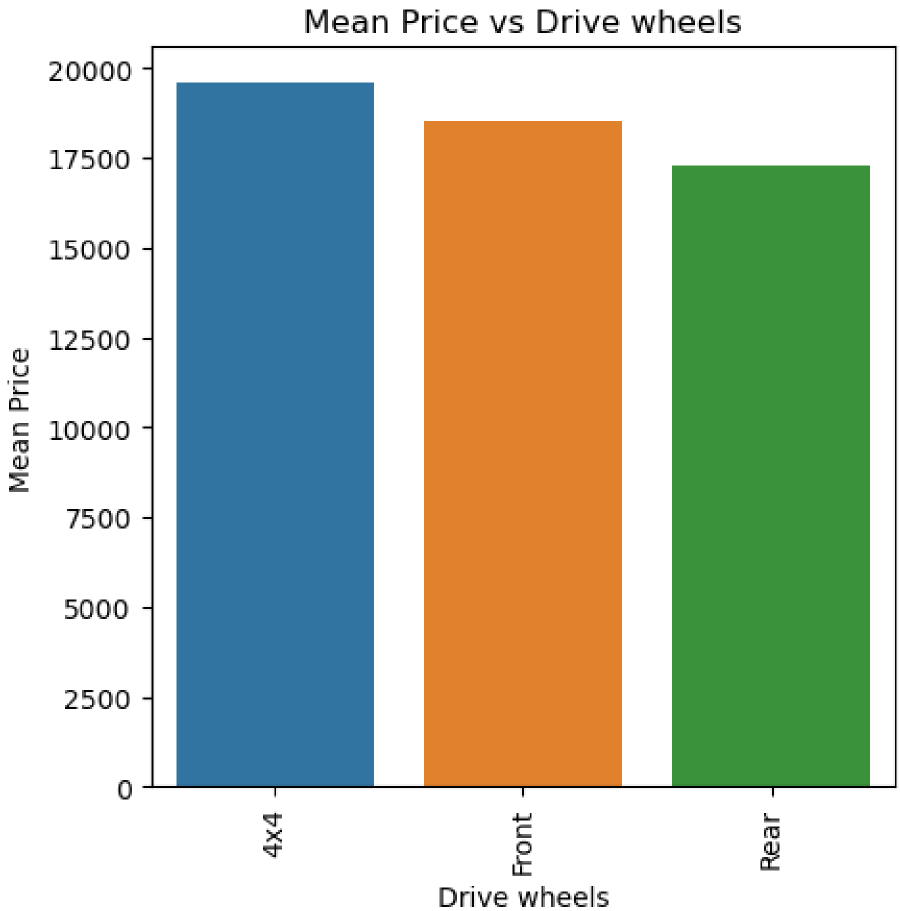

Mean Price vs Category
Mean Price
0
20000
40000
60000
80000
100000
120000
Cabriolet
Coupe
Goods wagon
Hatchback
Jeep
Limousine
Microbus
Minivan
Pickup
Sedan
Universal
Category

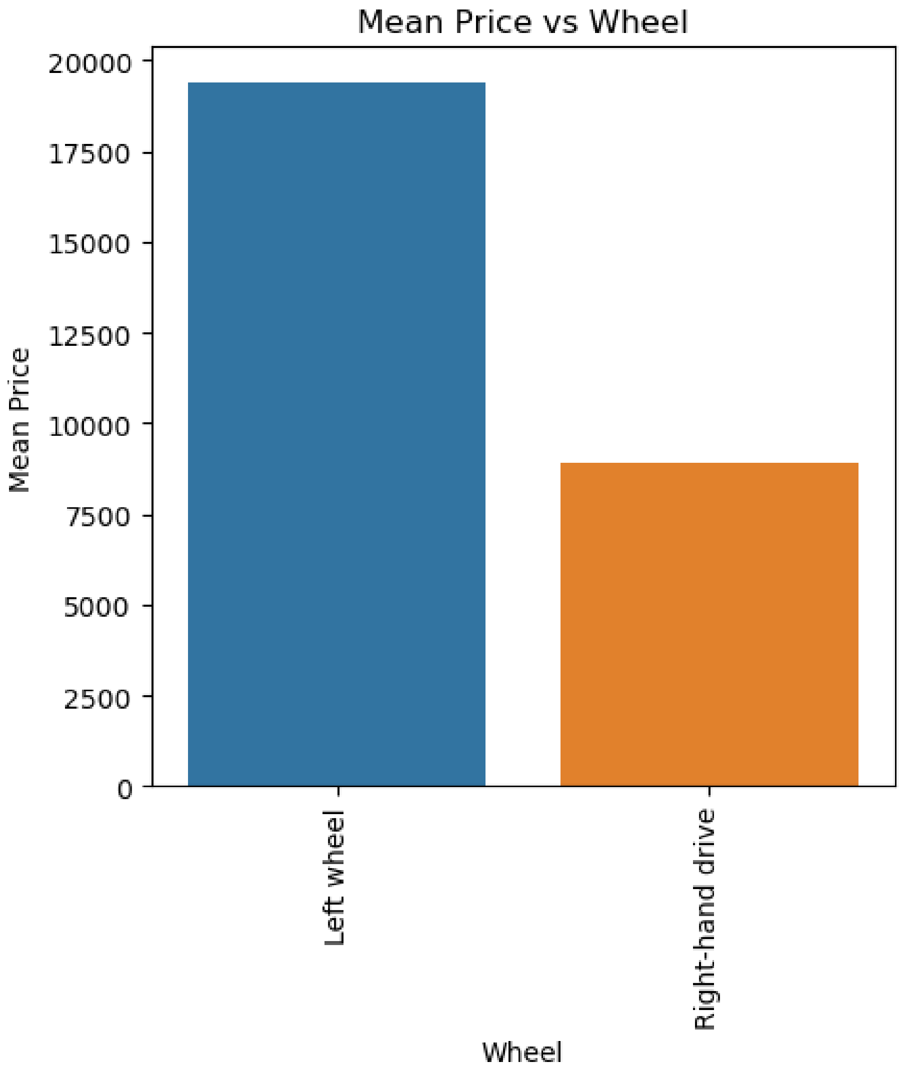

Mean Price vs Leather interior
Mean Price
17500
15000
12500
10000
7500
5000
2500
0
No
Yes
Leather interior

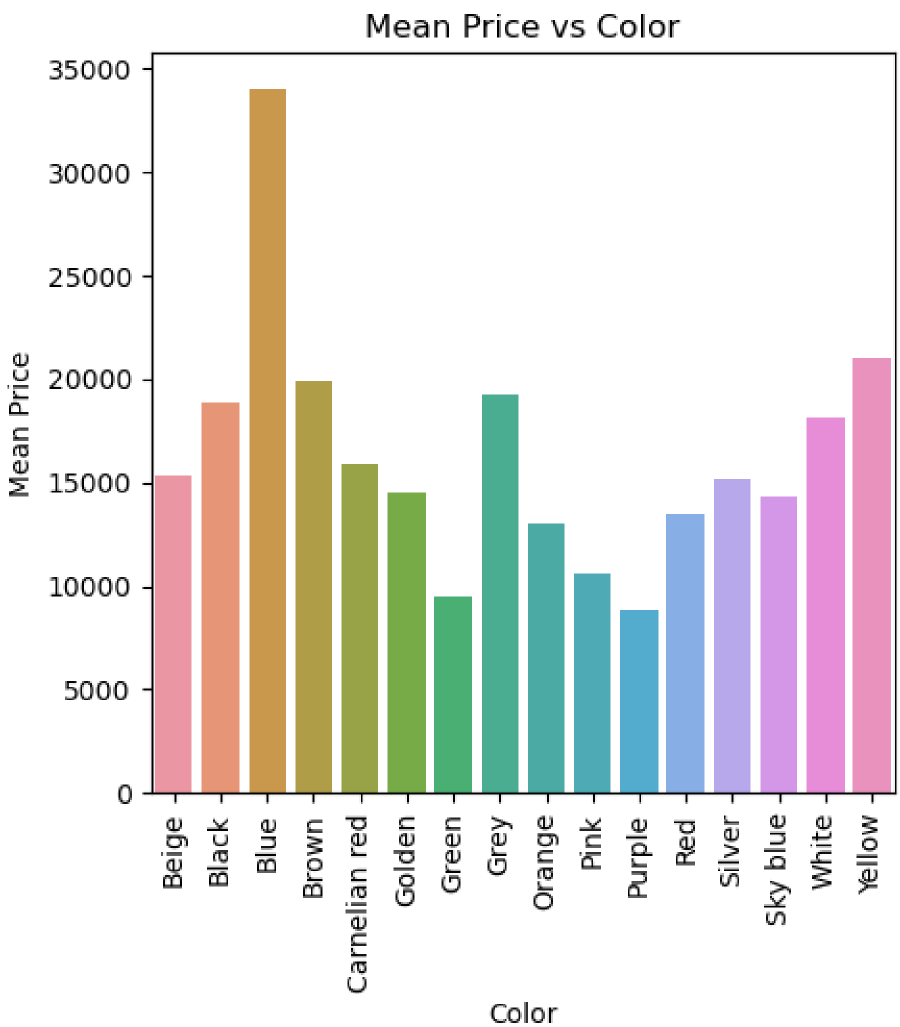

Mean Price vs Fuel type
Mean Price
30000
25000
20000
15000
10000
5000
0
CNG
Diesel
Hybrid
Hydrogen
LPG
Petrol
Plug-in Hybrid
Fuel type



Mean Price vs Gear box type
Mean Price
0
5000
10000
15000
20000
25000
Automatic
Manual
Tiptronic
Variator
Gear box type


Mean Price vs Drive wheels
Mean Price
20000
17500
15000
12500
10000
7500
5000
2500
0
4x4
Front
Rear
Drive wheels


Mean Price vs Wheel
20000
17500
15000
12500
10000
7500
5000
2500
0
Mean Price
Left wheel
Right-hand drive
Wheel

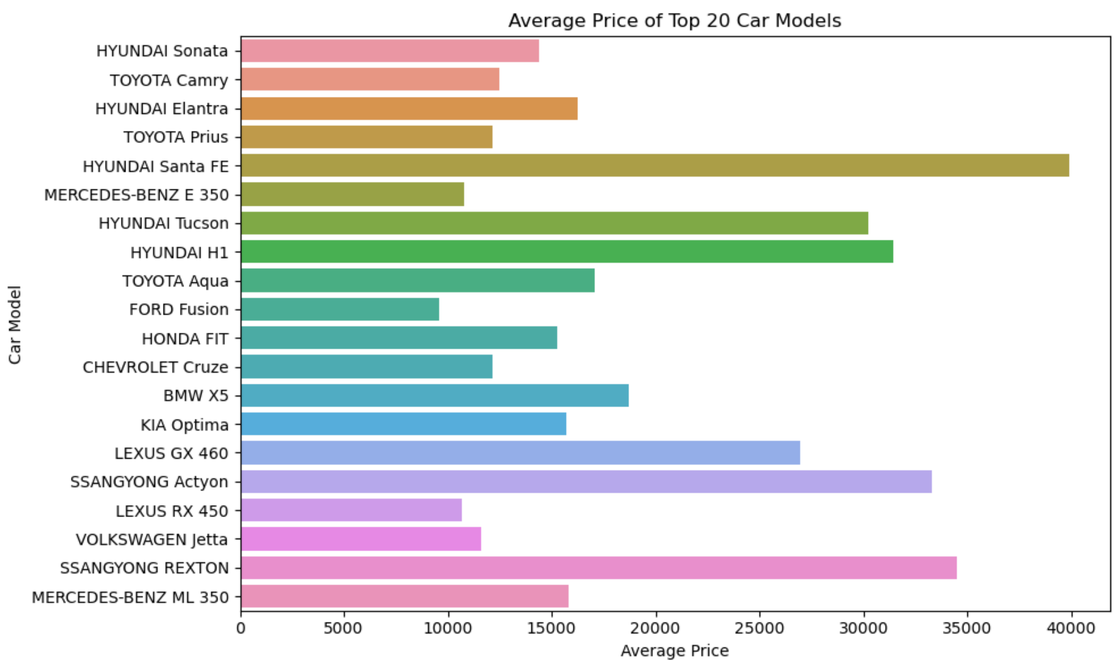


In the past 10 years, the visualization suggests that **Good Wagons** exhibit the highest mean price among all car categories. Likewise, **diesel cars**, on average, are the most costly. Additionally, vehicles with **leather** option tend to have a slightly higher mean price compared to those without it, and the presence of a **tiptronic gearbox** further elevates the price. **Left-wheel** drive cars typically have a higher mean price than right-hand drive ones, as do **4x4 vehicles**. Similarly, **blue** is also the most expensive color coated on a car.

Price and distribution of the top 20 most popular car models (Ranked by popularity) In this segmentation, we will consider the average product price cof the most popular car models (ranked by popularity) in the past 10 years.

[52]:
```
# Assuming dfy is the DataFrame containing car data
import matplotlib.pyplot as plt
import seaborn as sns
```

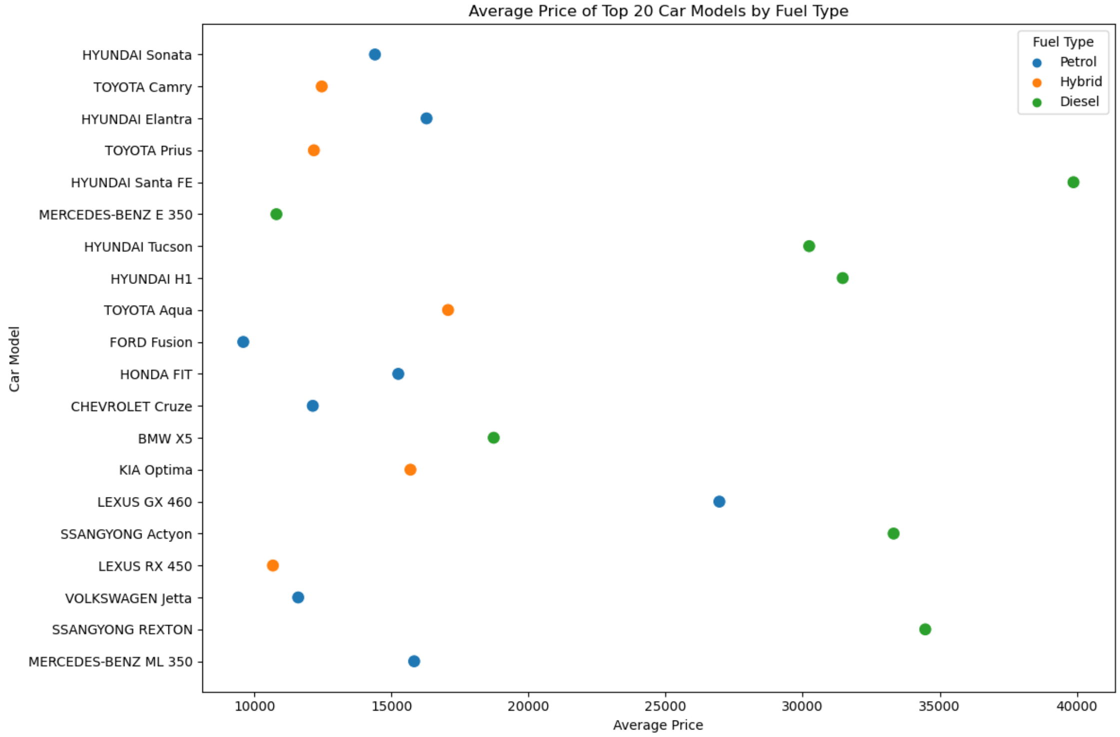
```python
# Combine manufacturer and model to create a new column
dfy['Manufacturer_Model'] = dfy['Manufacturer'] + ' ' + dfy['Model']

# Get the top 20 car models by frequency
top_car_models = dfy['Manufacturer_Model'].value_counts().head(20)

# Get the prices of the top 20 car models
top_car_prices = dfy.groupby('Manufacturer_Model')['Price'].mean().
    ↪loc[top_car_models.index]

# Plotting
plt.figure(figsize=(10, 6))
sns.barplot(x=top_car_prices.values, y=top_car_prices.index)
plt.title(f'Average Price of Top 20 Car Models')
plt.xlabel('Average Price')
plt.ylabel('Car Model')
plt.tight_layout()
plt.show()
```



Observation: Despite not being highly competitive in term of price, **Hyundai Sonata** is known as the most **popular** car model. This manufacturer's product also covers multiple price segmentation with **Hyundai Santa FE** being the most expensive model. Despite this, the product is also **5th** in term of popularity. In the **top 20**, we also witness the **notable** appearances of other manufacturers such as **Toyota**, **Mercedes-Benz**.

Before moving to discover largest manufacturers and their strategy for car option and prices. Let plot the graph for the top 20 car models focussing on **Fuel Type**.

[53]:
```
# Assuming dfy is the DataFrame containing car data
import matplotlib.pyplot as plt
import seaborn as sns

# Combine manufacturer and model to create a new column
dfy['Manufacturer_Model'] = dfy['Manufacturer'] + ' ' + dfy['Model']

# Get the top 20 car models by frequency
top_car_models = dfy['Manufacturer_Model'].value_counts().head(20)

# Get the prices of the top 20 car models
top_car_prices = dfy.groupby('Manufacturer_Model')['Price'].mean().
 ↪loc[top_car_models.index]

# Get the fuel types of the top 20 car models
top_car_fuel_types = dfy.groupby('Manufacturer_Model')['Fuel type'].first().
 ↪loc[top_car_models.index]

# Plotting
plt.figure(figsize=(12, 8))
sns.scatterplot(x=top_car_prices, y=top_car_models.index,␣
 ↪hue=top_car_fuel_types, s=100)
plt.title('Average Price of Top 20 Car Models by Fuel Type')
plt.xlabel('Average Price')
plt.ylabel('Car Model')
plt.legend(title='Fuel Type')
plt.tight_layout()
plt.show()
```

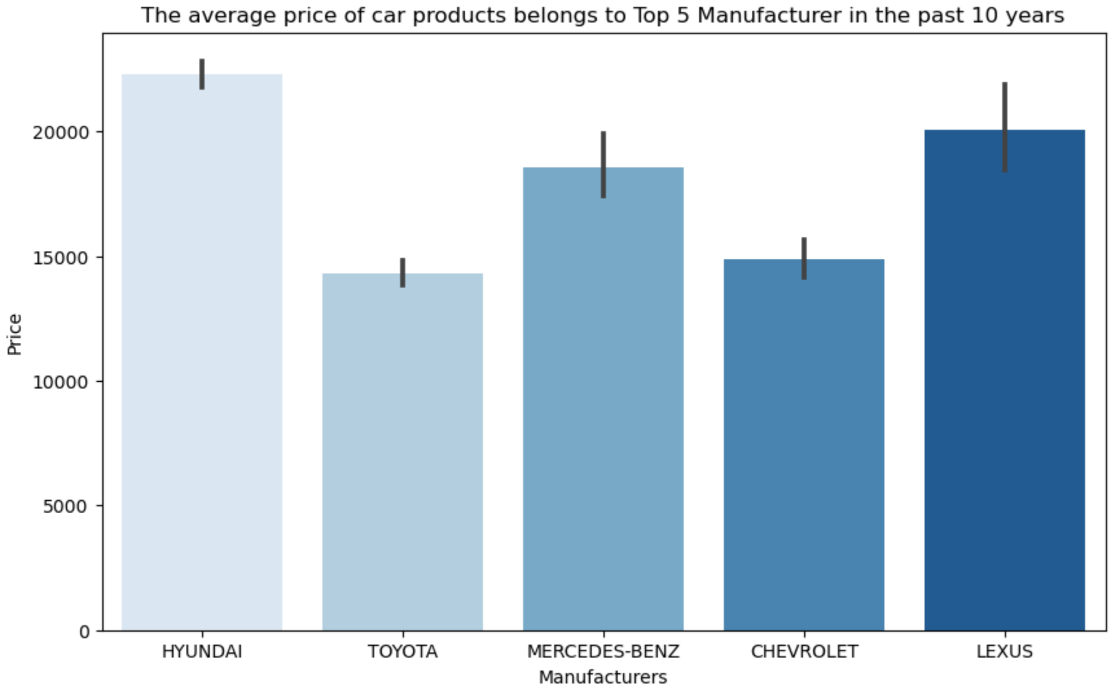


Observation: The distribution of car models's fuel types shows that **Petrol** ranked **first** in term of popularity. However, despite this, **Petrol vehicles** tends to have **lower prices** as most of the **blue dots** are located on the left. On the other hand, **green dot** indicates **Diesel** locates further to the right implies that more expensive cars tends to be equiped with this type of fuel engine. Despite the late introduction of **Hydrid** engine, it is still the focus products for some manufacturers such as **Lexus** or **Toyota**, following by a highly competitive price.

Now, let only focus on the top 5 Manufacturers (ranked by popularity) to obtain a clearer view on their pricing strategy and the idea behind.

[54]:
```
#Excluding 'Other'
top_manufacturers = dfy[dfy['Manufacturer'] != 'Other']['Manufacturer'].
 ↪value_counts().head(5).index
data_req  = df[df['Manufacturer'].isin(top_manufacturers)]

#Ploting the data
plt.figure(figsize=(10, 6))
sns.barplot(x='Manufacturer', y='Price', data=data_req,order=top_manufacturers,␣
 ↪palette="Blues")
plt.title("The average price of car products belongs to Top 5 Manufacturer in␣
 ↪the past 10 years") #Average price
plt.xlabel('Manufacturers')
plt.ylabel('Price')
```

```
plt.show()
```



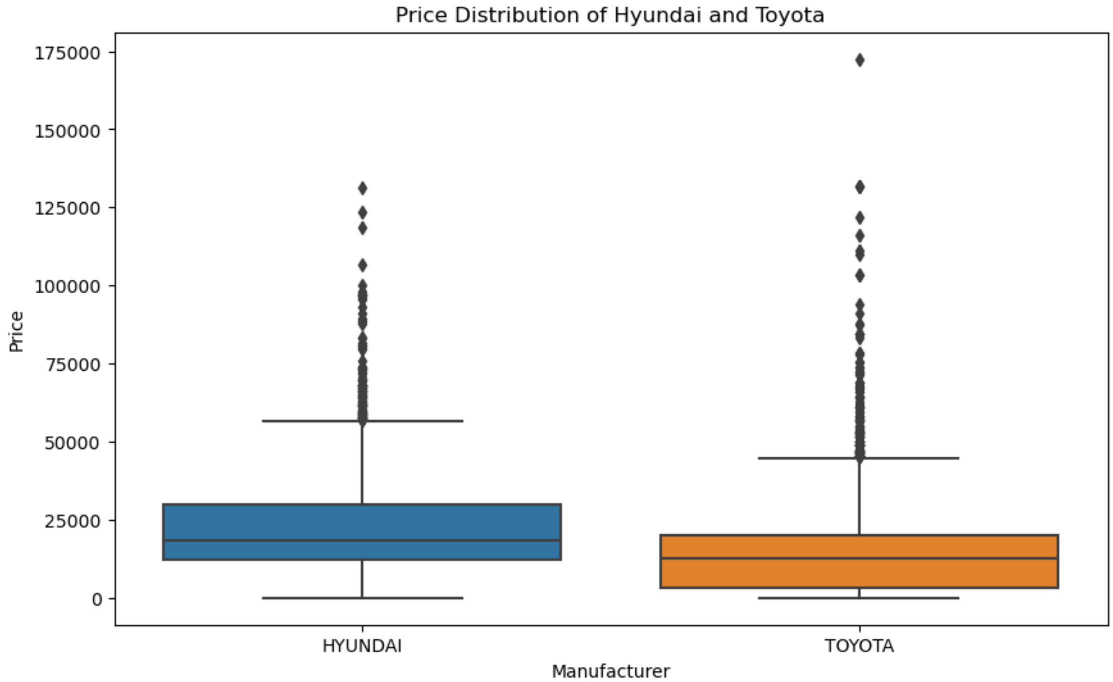
> Despite its popularity, **Hyundai** cars have the **highest** average price following by **Lexus - Toyota luxury brand** and **Mercedes-Benz**. In contrast, **Toyota** seems to have the most **affordable** cars in the top 5 with the **average price** staying below **15000$**.

From above graphs, we're given that **Hyundai** and **Toyota** are the **popular** manufacturers with market cap of **19.7%** and **19.1%** respectively. However despite this, the two manufacturers seems to deploy **different strategy** to achieve it, reflected through the **contradiction** in the price of their product. In the next part, we will observe in depth these differences.

[55]:
```
# Only considering top 2 which is Hyundai and Toyota for now
top_manufacturers = dfy[dfy['Manufacturer'] != 'Other']['Manufacturer'].
 ↪value_counts().head(2).index
data_req  = df[df['Manufacturer'].isin(top_manufacturers)]
```

[56]:
```
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
sns.boxplot(x='Manufacturer', y='Price', data=data_req)
plt.xlabel('Manufacturer')
plt.ylabel('Price')
plt.title('Price Distribution of Hyundai and Toyota')
```

```
plt.show()
```



> Despite having some **outlier models** pricing around **175000** dollars, **Toyota car** has a **lower median price** and **75** percent of their model pricing at **25000** dollars or less. **Hyundai** on the other hand, tends to distribute their product on a **wider** price range as the manufacturer owns a **larger** IQR. As result, only one-fourth of the brand's product shares the same price segmentation with 50% of **Toyota** cars which is around **15000**.

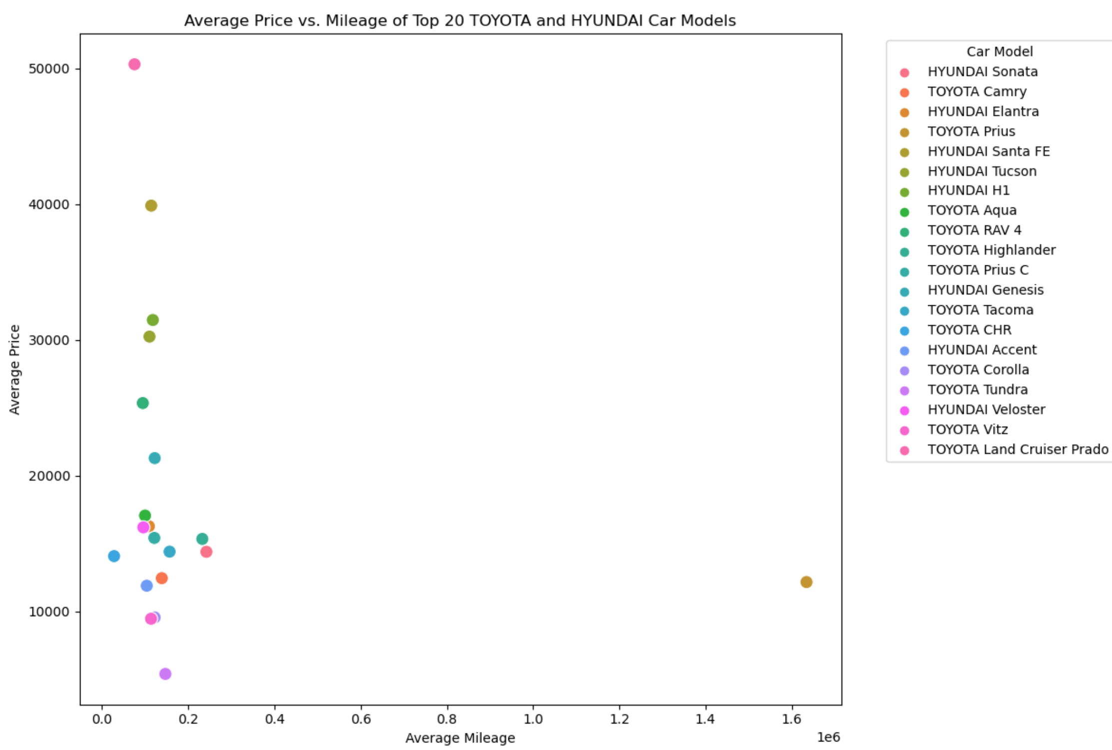
In summary, **Toyota** tends to focus on maintaining a lower median price and offering more affordable options to consumers, while **Hyundai** offers a wider range of pricing options, serving a wider range of customers with diverse budgetary preferences.

Suggestion for car model

Beside the investigation of current car market, the report also want to provide some suggestions for whom that want to choose a popular car model but also seeking for a good price on performance ratio products. This part will focus on **Hyundai** and **Toyota** models only, due to their **dominance** in popularity and number of model.

[57]:
```
# Filter the DataFrame for TOYOTA and HYUNDAI manufacturers
toyota_hyundai_df = dfy[dfy['Manufacturer'].isin(['TOYOTA', 'HYUNDAI'])]

# Get the top 20 car models by frequency for TOYOTA and HYUNDAI
top_car_models = toyota_hyundai_df['Manufacturer_Model'].value_counts().head(20)

# Get the average mileage of the top 20 car models for TOYOTA and HYUNDAI
```

```python
top_car_mileage = toyota_hyundai_df.groupby('Manufacturer_Model')['Mileage'].mean().loc[top_car_models.index]

# Get the prices of the top 20 car models for TOYOTA and HYUNDAI
top_car_prices = toyota_hyundai_df.groupby('Manufacturer_Model')['Price'].mean().loc[top_car_models.index]

# Plotting
plt.figure(figsize=(12, 8))
sns.scatterplot(x=top_car_mileage, y=top_car_prices, hue=top_car_models.index, s=100)
plt.title('Average Price vs. Mileage of Top 20 TOYOTA and HYUNDAI Car Models')
plt.xlabel('Average Mileage')
plt.ylabel('Average Price')
plt.legend(title='Car Model', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.show()
```



Toyota owns multiple popular models with **low price** such as Tundra or Venz. However, when consider **mileage** factor, **Toyota Highlander** is a good choice. **Hyundai** also offer **competitive models** with **Hyundai Sonata** as their main highlight. Despite that, **Toyota Prius** stands out as an considerable outlier.

1.1.4 III. Conclusion

In conclusion, the project obtain multiple insights that help manufacturers tailor product strategies and understand market dynamics while also trying to suggest popular but yet efficient model for personal use.

Market Overview: - Sedans are the most popular car type. - Automatic transmissions dominate, with an increasing trend towards hybrid engines. - Petrol remains the preferred fuel type, but hybrid engines are gaining traction. - Hyundai Sonata is the top-selling model despite not being the most competitively priced. - Price analysis reveals Toyota's affordability and Hyundai's wider price range.

Individual Suggestion: - Toyota offers popular models like Tundra and Venza, while Toyota Highlander excels in mileage. - Hyundai Sonata stands out for Hyundai, with Toyota Prius as a competitive outlier.

What i have learned: - Learned to explore and understand datasets as well as data cleaning, Process standardization. - Practiced creating effective visualizations, like scatter plots, boxplot, histogram…. - Applied statistical techniques (IQR,Outlier) to analyze trends and correlations in the data.